Univerza
v Ljubljani
Fakulteta
za elektrotehniko

SIMULACIJA DINAMIČNIH SISTEMOV

Založba
FE in FRI

BORUT ZUPANČIČ
s prispevki
Riharda Karbe
Draga Matka in
Igorja Škrjanca

Univerza v Ljubljani
Fakulteta za elektrotehniko

SIMULACIJA DINAMIČNIH SISTEMOV

BORUT ZUPANČIČ

Prispevki: R. Karba, D. Matko, I. Škrjanc

Ljubljana 2010

CIP - Kataložni zapis o publikaciji
Narodna in univerzitetna knjižnica, Ljubljana

681.5.033:004.94(075.8)(0.034.2)
519.876.5(075.8)(0.034.2)

ZUPANČIČ, Borut, Simulacija dinamičnih sistemov [Elektronski vir]
Borut Zupančič; s prispevki Riharda Karbe, Draga Matka in Igorja Škrjanca; izdajatelj Fakulteta za elektrotehniko. - El. knjiga. - Ljubljana: Založba FE in FRI, 2010

Način dostopa (URL):
`http://msc.fe.uni-lj.si/Download/Zupancic/SIM.pdf`

ISBN 978-961-243-144-0 (Fakulteta za elektrotehniko)

250598912

Založnik: Založba FE in FRI, Ljubljana

Izdajatelj: Fakuleta za elektrotehniko, Ljubljana

Urednik: mag. Peter Šega

Recenzenta: prof.dr. Rihard Karba, izr.prof.dr. Aleš Belič

Predgovor

Računalniška simulacija pomeni, da s pomočjo računalniškega programa čim bolj verno ponazorimo dinamično delovanje nekega realnega procesa. Je sodobna in izredno priljubljena metodologija pri analizi in načrtovanju različnih sistemov. Čeprav so pristopi že precej stari in izhajajo iz analogno-hibridne simulacije, pa so resnično zaživeli šele v osemdesetih letih preteklega stoletja predvsem z nenehnim večanjem sposobnosti osebnih računalnikov. Šele s tem je simulacija postala dostopna vsakomur in ne le tistim, ki so si lahko privoščili izredno drago opremo.

Simulacija je najtesneje povezana s področjem modeliranja. Nemogoče je postaviti ločnico med obema področjema. Zato je najbolj hvaležno obe področji obravnavati v enem delu, bodisi pod imenom modeliranje ali simulacija. V tem delu obravnavamo predvsem postopke, ki nas pripeljejo od že razvitega matematičnega modela do računalniškega programa in seveda do rezultatov simulacije. Torej je v tem učbeniku poudarek na osnovnih simulacijskih metodah in na uporabi simulacijskih orodij.

Tako kot modeli, so seveda tudi simulacije različne. V delu se omejujemo na simulacijo pretežno zveznih dinamičnih sistemov, t.j. sistemov, ki jih opisujemo z algebrajskimi in diferencialnimi enačbami. Postopki so enako primerni za kakršne koli tehniške ali celo netehniške sisteme.

Učbenik je sicer namenjen študentom Avtomatike na Fakulteti za elektrotehniko pri predmetu Simulacije, vendar ga priporočam vsem, ki se kakorkoli ukvarjajo s proučevanjem dinamičnih sistemov. Kajti vsak model realnega procesa je običajno nelinearen, časovno spremenljiv in v takem primeru so analitični postopki ponavadi neuspešni. Edino simulacija nas pripelje v relativno kratkem času in brez zamotane matematike do zanesljivih in uporabnih rezultatov.

Delo je razdeljeno na sedem poglavij. Prvo poglavje je uvod v simulacijo dinamičnih sistemov. Vpeljemo osnovne pojme in nekatere primere (modele), ki jih kasneje uporabljamo pri predstavitvi metod in orodij. V drugem poglavju uvedemo zvezno, diskretno in kombinirano simulacijo ter simulacijo v realnem času. Tretje

poglavje je posvečeno metodam pri reševanju problemov s simulacijo. Obravnavamo indirektno, direktno in implicitno metodo, simulacijo prenosnih funkcij, ter koncept digitalne simulacije. Četrto poglavje uvaja orodja za digitalno simulacijo, peto pa obravnava jezike za simulacijo zveznih dinamičnih sistemov. Podajamo programsko zgradbo jezikov, enačbne in bločne simulacijske jezike, simulacijo s pomočjo Matlabovih funkcij ter možnosti uporabe splošnonamenskih programskih jezikov. V šestem poglavju smo obdelali numerične postopke. Omejili smo se na integracijske metode, na integracijo preko nezveznosti ter na reševanje problema algebrajske zanke. Zadnje poglavje pa opisuje zgodovino nastanka osnovnih in prvih simulacijskih konceptov, t.j. analogno-hibridno simulacijo. Čeprav se analogno- hibridni računalniki več ne uporabljajo, pa so nekateri koncepti vseeno pomembni in uporabni tudi v digitalni simulaciji.

Del gradiva v tej knjigi temelji na prevodu pa tudi ustrezni predelavi vsebin iz knjige: D. Matko, B. Zupančič, R. Karba: Simulation and Modelling of Continuous Systems, A Case Study Approach (Matko in ostali, 1992). Kratek opis okolja Matlab-Simulink je delo prof. dr. I. Škrjanca. Prav posebno vlogo pa ima nekdanji predstojnik našega laboratorija prof. dr. F. Bremšak, ki je najbolj zaslužen, da je simulacija kot metoda in orodje že dolgo ustaljena praksa tako v izobraževanju na Fakulteti za elektrotehniko Univere v Ljubljani kot pri raziskovalnem delu v Laboratoriju za modeliranje, simulacijo in vodenje in v Laboratoriju za avtonomne mobilne sisteme.

Ljubljana, marec 2010

Borut Zupančič

Kazalo

Poglavje 1

Uvod v simulacijo dinamičnih sistemov

Simulacija dinamičnih sistemov je ena najpomembnejših metod na področju analize in načrtovanja vodenja sistemov. Predstavlja sodobno metodologijo, ki direktno ali pa vsaj posredno spremlja vse sodobne metode analize in načrtovanja. Kljub izrednemu razvoju teorije vodenja sistemov je še vedno očitno, da najboljše rezultate dosegamo le z interaktivnim delom na računalniku, pri katerem imajo pomembno vlogo učinkoviti simulacijski jeziki in izkušnje uporabnika. Simulacija se uporablja praktično na vseh področjih, ki temeljijo na sistemskem pristopu obravnave procesov in sicer: pri vodenju sistemov, v robotiki in v računalništvu, v fiziki, biologiji, kemiji, medicini, farmaciji, pa tudi v ekonomskih, sociooloških in političnih vedah. Predstavlja pa navadno tudi eno od osnovnih metodologij pri reševanju interdisciplinarnih projektov.

Simulacija dinamičnih procesov je področje, ki je v zadnjih petdesetih letih takoj za signalnim procesiranjem najbolj drastično vplivalo na razvoj računalnikov. Obe področji namreč razvijata v smislu porabe računalniškega časa zelo potratne metode, ki so seveda učinkovite le ob uporabi sposobnih računalnikov.

Področje vodenja sistemov je eno prvih področij (takoj za letalsko industrijo), kjer so že pred več kot štiridesetimi leti s pridom uporabljali računalniško simulacijo. Orodje so takrat seveda predstavljali razni diferencialni analizatorji in pozneje analogni računalniki. Kasneje so se orodja zelo izpopolnila. Poleg analognih so se pojavili sposobni hibridni in digitalni računalniki skupno z ustrezno programsko opremo z ustreznimi simulacijskimi jeziki. Ker pa je bila računalniška

oprema, na kateri je bilo možno izvajati simulacijo, zelo draga in ker je simulacija veljala za računalniško zelo potratno metodo, se je pred razvojem cenenih mini in mikroračunalnikov uporabljala le kot skrajna možnost za reševanje omenjenih problemov.

V zadnjih desetletjih pa je simulacija zaradi izrednega razvoja materialne in programske opreme postala metodologija, ki jo načrtovalci sistemov vodenja običajno najprej uporabijo. Bistveno vlogo je imel pri tem razvoj sposobnih simulacijskih jezikov na cenenih mini in mikroračunalnikih (npr. osebni računalniki). To je omogočilo, da je simulacija postala dostopna vsakomur, tudi izobraževalnim institucijam in študentom. Zato je simulacija doživela nov vzpon in oživila številna teoretična področja sistemskih ved, pri čemer naj omenimo še zlasti področje nelinearnih sistemov. Raziskave kaotičnih sistemov predstavljajo aktualno področje nelinearnih sistemov, k razmahu tega področja pa je prispevala prav simulacija.

1.1 Definicije simulacije

Avtorji različno definirajo pojem simulacije. Omenili bomo nekatere definicije.

Simulacija dinamičnih sistemov je iterativna metoda, s pomočjo katere proučujemo funkcionalne lastnosti dinamičnih sistemov z eksperimentiranjem na ustreznem modelu realnega objekta. Kljub temu, da literatura mnogokrat bolj poudarja metode analize in sinteze, pa se pri reševanju realnih problemov vedno znova potrjuje velika vloga simulacije. Zato jo inženirji uspešno uporabljajo kot pomoč pri razumevanju problemov, kot orodje pri načrtovanju vodenja sistemov, za učenje osebja, ki nadzoruje vodene procese ter kot orodje, ki omogoča študirati nove ideje v fazi načrtovanja.

Pritsker (Pritsker, 1979) obravnava simulacijo in modeliranje kot dva nerazdružljiva pojma. Model predstavlja poenostavljen sistem, simulacija pa je posnemanje obnašanja sistema v realnem, skrčenem ali raztegnjenem času na ta način, da eksperimentiramo z modelom. Model, ki ga izvedemo na računalniku, je računalniški simulacijski model ali krajše simulacijski model.

Mc Leod (Mc Leod, 1987b) definira simulacijo kot uporabo modela za eksperimentiranje. Na ta način skušamo napovedati možno obnašanje sistema ali situacije, ki jo proučujemo. Podobno definicijo je uporabil tudi Korn (Korn, Wait, 1978).

Podobno kot Pritsker tudi Schmidt (Schmidt, 1987) obravnava modeliranje in simulacijo kot dva nerazdružljiva pojma. Simulacija je postopek, ki omogoča proučevanje realnega sistema s pomočjo drugega sistema, ki ga imenujemo realni model. Pomembno je, da imata realni sistem in realni model enak konceptualni model glede na značilnosti, ki jih proučujemo, vendar z realnim modelom laže eksperimentiramo. Simulacija pomeni torej določitev realnega (simulacijskega) modela in eksperimentiranje z njim z namenom študija oz. analize realnega sistema in napovedovanja njegovega obnašanja.

1.2 Splošno o simulaciji dinamičnih sistemov

Atherton (Atherton, 1986) omenja naslednja dva bistvena razloga, zakaj načrtovalci sistemov vodenja uporabljajo simulacijo:

- Analitične metode, pa čeprav jih lahko uporabljamo v obsežnih CACSD (Computer Aided Control System Design) paketih, so često zelo omejene, če jih uporabljamo za reševanje realnih problemov. Tipično je npr. področje nelinearnih sistemov in še bolj področje kaotičnih sistemov in stohastičnih sistemov. Simulacija predstavlja najuporabnejšo metodo tudi pri študiju vpliva sprememb parametrov na obnašanje sistema in s tem tudi pri študiju robustnosti.
- Za simulacijo v realnem času ob priključenih realnih objektih (hardware in the loop - HIL). Analogni računalniki so bili dolgo edino primerni za to vrsto simulacije. Z razvojem sodobne materialne in programske opreme lahko tudi digitalni računalniki izredno uspešno vršijo simulacijo v realnem času.

Fischlin (Fischlin, 1986) opisuje vlogo simulacije v inženirskem izobraževanju. Omenja, da je glavna prednost simulacije v tem, da omogoča skoraj na vseh področjih zamenjati realni svet, kompleksne eksperimente in pilotne naprave z enostavnim in cenenim mikroračunalnikom, na katerem poteka simulacija. S takim modelom je potem možno delati brez tveganja. Zaradi izredne ilustrativnosti je pristop primeren za začetnike, ki o realnem objektu ne vedo dosti, pa tudi za izkušenejše uporabnike.

Schmid (Schmid, 1988) omenja, da se simulacija uporablja prav v vsaki projektni fazi. Zato mora imeti vsak kvaliteten paket za CACSD vgrajen sposoben splošno

namenski simulator. Le-ta mora omogočati simulacijo zveznih in diskretnih sistemov.

Tudi Annino (Annino, 1979) ugotavlja, da je simulacija učinkovita metoda za preizkušanje modelov, pred razvojem in gradnjo dragih prototipov in implementacijo. V primerjavi z metodami analize je simulacija bolj realistična in laže razumljiva toda le, če je pravilno uporabljana. Izkušnje kažejo, da zaradi nepravilne uporabe večina simulacijskih projektov v sedemdesetih letih v Ameriki ni dala zadovoljivih rezultatov.

Mc Leod (Mc Leod, 1987b) opozarja, da je simulacija ena temeljnih ved modernih področij. Nekateri se sploh ne zavedajo, da uporabljajo simulacijo, drugim pa se zdi ta veda preveč klasična, da bi jo priznali. Te trditve avtor dokazuje na nekaterih sodobnih področjih. Tako pravi, da emulacija pomeni, da se en računalnik vede kot drugi, torej ga simulira. Celotno področje iger je osnovano na modelih in eksperimentiranju z njimi. Na področju umetne inteligence se uporablja modeliranje nekaterih aspektov mentalnih sposobnosti. Če ta model uporabimo za eksperimente, je to simulacija. Roboti so nastali kot modeli nekaterih človekovih aktivnosti in jih torej simulirajo. Razen tega robotika vsebuje elemente umetne inteligence in je torej zelo ozko povezana s simulacijo. Podobno velja za področje ekspertnih sistemov. Ekspertni sistem emulira človeka eksperta v procesu analize in načrtovanja. Torej simulira določene človekove akcije. Tudi navidezna resničnost (virtual reality) je osnovana na simuliranih modelih.

Herget (Herget, 1988) je z anketo pokazal, da je simulacija osrednja operacija na področju računalniškega načrtovanja vodenja sistemov. V anketi je sodelovalo 63 posameznikov in 47 ustanov, ki se ukvarjajo s področjem CACSD v ZDA. Anketa je pokazala, da je najpogosteje uporabljeni CACSD programski paket simulacijski jezik ACSL. Med prvimi desetimi paketi je še nekaj pretežno simulacijskih produktov. Anketa je tudi pokazala, da uporabniki na splošno svoje pakete največkrat uporabljajo za simulacijo (indeks 85), sledi področje načrtovanja sistemov vodenja (indeks 74) in pa področje identifikacije sistemov (indeks 43). Pri tem je treba omeniti, da se lahko simulacija implicitno skriva tudi v drugem in tretjem omenjenem področju. Podobne rezultate je dala tudi analiza uporabe paketov CACSD na Japonskem (Araki, 1985) in na Nizozemskem (Van den Bosch, Van den Boom, 1985).

Heinrich (Heinrich, 1986) je naredil analizo, ki je pokazala, da je veliko število člankov na svetovno znanih konferencah iz področja vodenja (npr. IFAC) precej simulacijsko obarvanih, čeprav to niso simulacijsko orientirane konference in tudi nimajo simulacijskih sekcij. Pravi, da ni to nič presenetljivega, saj se vsaka

metoda lahko dokaže le na tri načine: z aplikacijo v realnem svetu, s simulacijo ali analitično "na papirju".

V naslednjih letih lahko pričakujemo, da bo simulacija postala še bolj razširjena. Superračunalniki se približujejo tudi po hitrosti analognim računalnikom. Še več si seveda obetamo od multiprocesorskih in paralelnoprocesorskih sistemov, ki bodo v povezavi z izredno sposobnimi grafičnimi zasloni ter potrebnimi perifernimi enotami predstavljali učinkovite simulacijske delovne postaje. Glavno prednost bodo te delovne postaje pokazale pri simulaciji v realnem času s priključenimi realnimi objekti (HIL).

1.3 Uporabnost simulacije in njene omejitve

Problematiko uporabe simulacije in njenih omejitev, je najbolj široko obdelal Neelamkavil (Neelamkavil, 1987). Simulacijo uporabljamo v naslednjih primerih:

- Realni sistem ne obstaja, pri tem pa je gradnja prototipov in eksperimentiranje drago in časovno zamudno. Včasih je tudi nemogoče graditi prototipe (npr. jedrski reaktor).
- Eksperimentiranje z realnim sistemom je drago, nevarno, dolgotrajno, zahteva kompleksno opremo in lahko povzroči resne motnje v delovanju (npr. transportni sistem, jedrski reaktor).
- Kaže se potreba po študiju obnašanja sistema v preteklosti, sedanjosti ali prihodnosti v realnem, skrčenem ali raztegnjenem času (npr. sistemi vodenja, naraščanje prebivalstva).
- Matematični modeli nimajo enostavne in praktično uporabne analitične rešitve (npr. nelinearne diferencialne enačbe, ki opisujejo kaotične sisteme).
- Možno je zadovoljivo ovrednotiti simulacijski model glede na podatke o realnem objektu - vrednotenje modela.
- V inženirskem izobraževanju predstavlja simulacija najbolj nazorno in hkrati tudi ceneno metodo, ki omogoča delo brez tveganja tako pri začetnikih kot pri izkušenih uporabnikih.

Seveda ima simulacija tudi nekatere slabosti, ki omejujejo njeno uporabo. Te so:

- Digitalna simulacija je relativno počasna, iterativna tehnika in zato računalniško potratna.
- Včasih nas vodi do suboptimalnih rešitev, če ne uporabljamo optimizacijskih postopkov.
- Brez dobrega poznavanja realnega procesa ali objekta je včasih težko ovrednotiti rezultate simulacijskih modelov.
- Analiza in interpretacija rezultatov zahteva v nekaterih primerih tudi dobro poznavanje teorije verjetnostnega računa in statistike.
- Rezultate lahko napačno interpretiramo.
- Včasih je težko odkriti, odkod napaka izvira.

1.4 Modeliranje in simulacija kot nerazdružljivi metodi

Vsak realni objekt, za katerega želimo načrtati vodenje, začnemo proučevati s pomočjo podatkov, ki so nam na voljo in s pomočjo eksperimentov, ki jih je možno izvajati na objektu. Na ta način si ustvarimo bazo podatkov o sistemu. S pomočjo analize podatkov o sistemu zgradimo t.i. konceptualni - matematični model (Schmidt, 1987). Le-ta predstavlja sliko realnega objekta po neki človekovi zamisli. Zgrajen mora biti tako, da se vede kot realni objekt, vendar le za tiste namene, za katere služi in v okviru določenih toleranc. Zato moramo pri gradnji konceptualnega (matematičnega) modela:

- jasno opredeliti namen modeliranja,
- definirati omejitve, znotraj katerih deluje model,
- izbrati lastnosti (atribute), ki jih bomo upoštevali in zanemariti nepomembne ter idealizirati določene realne zakonitosti,
- definirati strukturo in določiti parametre modela, t.j. definirati moramo povezave posameznih atributov v sistem ter opisati dinamiko posameznih atributov z diferencialnimi in diferenčnimi enačbami. Diferencialne enačbe dobimo ob upoštevanju masnega ali energijskega ravnotežja oz. ob upoštevanju zakonov o ohranitvi mase, energije in gibalne količine.

Ko zgradimo konceptualni (matematični) model, moramo zbrati čim več informacij o njegovem vedenju. Samo v enostavnih primerih lahko izpeljemo analitično rešitev konceptualnega modela. V splošnem pa uporabljamo pristop, da iz konceptualnega modela zgradimo t.i. realni ali simulacijski model, ki služi nato za preizkušanje realnega sistema. Pomembno je torej, da imata realni sistem in realni model enak konceptualni (matematični) model. Izgradnja realnega modela je prvi korak pri simulaciji. Drugi korak pa je eksperimentiranje s tem modelom. S pomočjo dedukcije (analitične obravnave) konceptualnega modela in eksperimentiranja z realnim ali simulacijskim modelom dobimo podatke o modelu. S pomočjo teh podatkov izvedemo vrednotenje modela, t.j. analizo ujemanja obnašanja realnega sistema in konceptualnega modela. Pri tem moramo poudariti, da vrednotenje modela ni isto kot verifikacija. Verifikacija je analiza ujemanja realnega in konceptualnega modela. Pri računalniški simulaciji to pomeni, da preverjamo, če je bil simulacijski program pravilno izveden.

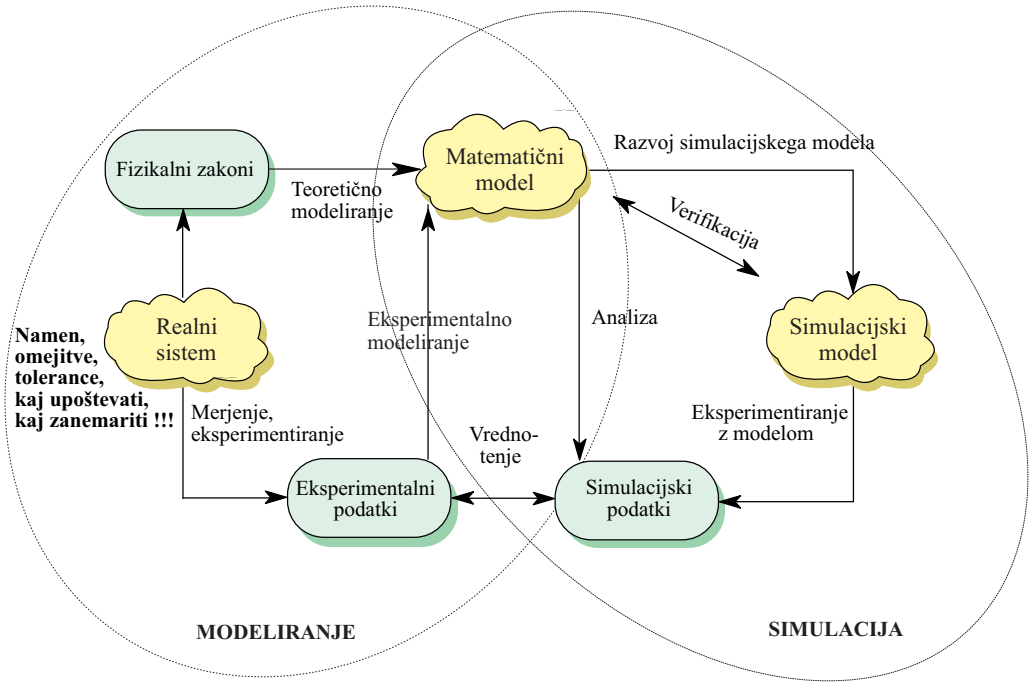
Analizo sistema, konstrukcijo modela, eksperimentiranje z modelom, vrednotenje in verifikacijo je običajno potrebno večkrat ponoviti, dokler ne pridemo do zahtevanih rezultatov. Zato Neelamkavil (Neelamkavil, 1987) simulacijo v širšem smislu definira kot počasno, iterativno in eksperimentalno orientirano tehniko.

Iterativni postopek modeliranja in simulacije prikazuje slika 1.1. Znano je, da za vsak mehanski ali hidravlični sistem lahko poiščemo električni ekvivalent. To pomeni, da so sistemi izrazljivi z enakimi diferencialnimi enačbami, torej imajo isti matematični model. V praksi je lahko eden izmed njih realni sistem, drugi pa realni model, ali obratno. Seveda za realne modele izberemo vedno take, s katerimi je možno enostavno eksperimentirati.

Čeprav sta modeliranje in simulacija tako neposredno povezani metodi, je namen tega učbenika pokriti zgolj simulacijski del. Da pa bo povezava vendarle dovolj poudarjena, bomo podali tri primere, v katerih bo poudarek na modeliranju. Te modele bomo uporabljali tudi v naslednjih poglavjih za prikaz simulacijskih metod in orodij.

Primer 1.1 Modeliranje avtomobilskega vzmetenja

Modeliranje poenostavljenega avtomobilskega vzmetenja predstavlja primer teoretičnega modeliranja na osnovi ravnotežja gibalnih količin. Običajno je model sistem dveh linearnih diferencialnih enačb drugega reda, ki opisujeta premike karoserije avtomobila in premike kolesa. Predpostavljamo torej model s koncentriranimi parametri. Pri tem upoštevamo vzmeti in dušilnike avtomobila ter



Slika 1.1: Iterativni postopek modeliranja in simulacije

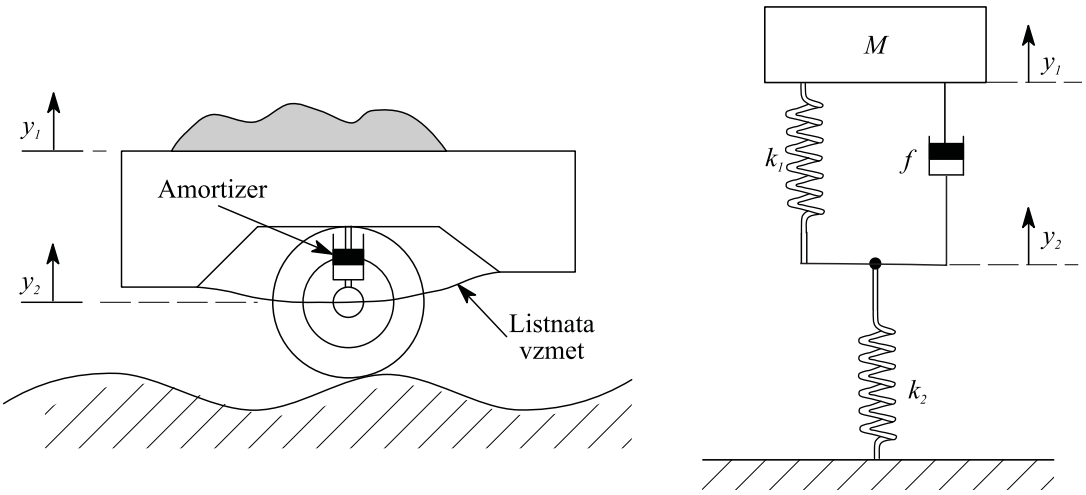
elastičnost pnevmatike. Oblika podlage predstavlja vzbujanje sistema. Model lahko služi kot dober pripomoček za spoznavanje modeliranega procesa, za študij vpliva posameznih elementov na obnašanje sistema, pa tudi za grobo dimenzioniranje elementov.

V našem primeru pa bomo uvedli še eno predpostavko. Ker smo zainteresirani le za obliko gibanja karoserije, bomo zanemarili še maso kolesa in obese, ki je majhna v primeri s četrtino mase avtomobila. Tako lahko obravnavani sistem prikažemo kot kombinacijo translatornih mehanskih elementov na sliki 1.2.

Na sliki 1.2 predstavlja M četrtino mase avtomobila, k_1 je konstanta togosti vzmeti avtomobila, f je konstanta dušenja dušilnika avtomobila, k_2 je konstanta togosti pnevmatike, y_1 je premik karoserije, y_2 pa premik kolesa iz mirovne lege.

Osnovno relacijo za začetek teoretičnega modeliranja predstavlja *drugi Newtonov zakon*, saj predstavlja eno od oblik zakona o ravnotežju gibalnih količin

$$\Sigma F = Ma = M\ddot{y} \tag{1.1}$$



Slika 1.2: Poenostavljen prikaz vzmetenja avtomobila

kjer je M masa [kg], a je pospešek [ms^{-2}] in F je sila [N]. Glede na to enačbo moramo najti ustrezne relacije še za ostale sile v sistemu (njihove povezave s premikom, ki nas zanimajo). Za silo vzmeti je na razpolago Hook-ov zakon, ki podaja silo, ki je potrebna, da raztegnemo (ali pa stisnemo) vzmet za razdaljo y glede na njeno osnovno dolžino

$$F_s = ky \tag{1.2}$$

pri čemer je k konstanta vzmeti [Nm^{-1}].

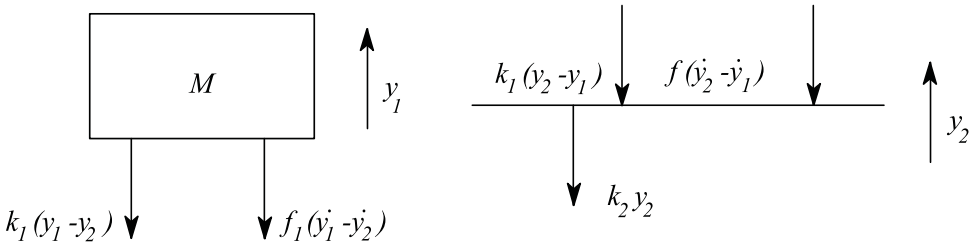
Silo, ki je potrebna, da premaknemo en konec viskoznega dušilnika s hitrostjo v relativno glede na drugi konec, podaja zveza

$$F_D = fv = f\dot{y} \tag{1.3}$$

kjer je f dušilni koeficient [Nsm^{-1}].

Tako se moramo odločiti le še o predznaku sil, kar najlaže naredimo s pomočjo diagrama sil. Pri tem se dogovorimo, da so sile v smeri predpostavljenega premika (največkrat ga izberemo v smeri sile, ki vzbuja sistem) pozitivne. Zavedati pa se

moramo, da se sile na vzmeteh in dušilnikih vedno upirajo kakršnemkoli premiku. Prav tako je jasno, da premiki v mehanskem vezju niso vezani le na mase, ki so vključene vanj, temveč se, kot v našem primeru, lahko pojavijo tudi ob zaporednih vezavah elementov, ali pa če vzbujevalna sila deluje na vzmet ali na dušilnik, ne pa na maso. Kadar je vsak konec elementa podvržen različnima premikoma (ni vezan na mirujočo točko), se namesto premika y oz. odvoda $\dot{y}$ v zvezah pojavi razlika premikov oz. odvodov (v tej razliki ima prva spremenljivka isti indeks, kot ga ima premik točke, v kateri delujejo obravnavane sile). Glede na povedano lahko za naš primer narišemo diagram sil na sliki 1.3.



Slika 1.3: Diagram sil za sistem avtomobilskega vzmetenja

Iz diagrama sil na sliki 1.3 lahko direktno napišemo enačbi za oba premika tako, da sile s predznakom, kot ga narekuje diagram sil, vpišemo na desno stran enačbe (1.1)

$$M\ddot{y}_1 = -f(\dot{y}_1 - \dot{y}_2) - k_1(y_1 - y_2) \tag{1.4}$$
$$0 = -f(\dot{y}_2 - \dot{y}_1) - k_1(y_2 - y_1) - k_2 y_2 \tag{1.5}$$

Enačbi (1.4) in (1.5) že predstavljata matematični model našega procesa. Ker pa nas zanima le y_1, eliminirajmo y_2. Vsota enačb (1.4) in (1.5) daje

$$M\ddot{y}_1 = -k_2 y_2 \tag{1.6}$$

iz česar izrazimo y_2 kot

$$y_2 = -\frac{M}{k_2}\ddot{y}_1 \tag{1.7}$$

Če y_2 sedaj vnesemo v enačbo (1.4), dobimo

$$\frac{fM}{k_2}\dddot{y}_1 + M(1+\frac{k_1}{k_2})\ddot{y}_1 + f\dot{y}_1 + k_1 y_1 = 0 \tag{1.8}$$

in končno

$$\dddot{y}_1 + \frac{k_1+k_2}{f}\ddot{y}_1 + \frac{k_2}{M}\dot{y}_1 + \frac{k_1 k_2}{Mf}y_1 = 0 \tag{1.9}$$

Dobili smo torej navadno linearno diferencialno enačbo tretjega reda s konstantnimi koeficienti, ki opisuje gibanje karoserije avtomobila.

Do sedaj nismo rekli še ničesar o vzbujanju sistema. Kot vemo, je sistem lahko vzbujan z vhodnim signalom, ali pa ima v trenutku, ko ga začnemo opazovati, določeno začetno stanje. Zunanji vhod bi lahko predstavljala podlaga po kateri vozi avto. Kakšna ovira ali jama na podlagi bi vzbudila sistem, pri čemer pa ne smemo pozabiti, da je glede na predpostavko naše kolo infinitezimalno majhno. Izberimo začetno stanje

$$y_1(0) = -y_{10} \tag{1.10}$$

Začetno stanje si lahko predstavljamo tako, da voznik vstopi v avtomobil, v trenutku $t = 0$ pa izstopi.

Predpostavimo naslednje konkretne podatke avtomobilskega vzmetenja:

M=500 kg, k_1=7500 N/m, k_2=150000 N/m, f=2250 Ns/m in y_{10}=0.05 m.

Tako lahko enačbo (1.9) zapišemo v splošni obliki

$$\dddot{y}_1 + a\ddot{y}_1 + b\dot{y}_1 + cy_1 = 0 \tag{1.11}$$

pri čemer so

$a = 70$, $b = 300$, $c = 1000$ in $y_1(0) = -y_{10} = -0.05$

□

Primer 1.2 Modeliranje regulacije gretja prostora

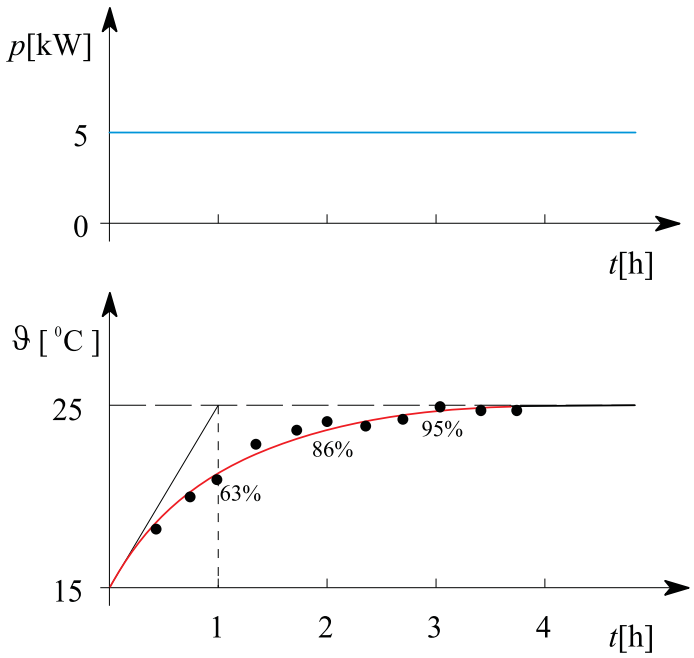
Oglejmo si še možnost "intuitivnega" pristopa k modeliranju gretja prostora z električnim grelom, ki ga prižiga ali ugaša termostatski regulator. To je klasična in cenena možnost regulacije temperature v prostoru, ki jo v vsakdanjem življenju največkrat uporabljamo. Učinki regulacije zavisijo v glavnem od treh faktorjev:

- moči grelnega telesa,
- histereze termostatskega regulatorja in
- zakasnitev, ki so prisotne v procesu.

Ti faktorji določajo nihanja temperature okrog želene vrednosti. Ker želimo majhna nihanja, bi morala biti tudi histereza termostata majhna, kar pa po drugi strani povzroča pogosto prižiganje in ugašanje grela in zopet predstavlja nezaželen pojav. Seveda pa je za čas, v katerem dosežemo želeno temperaturo, odločilna moč grela. Načrtovanje sistema gretja (dimenzioniranje gretja in izbira ter postavitev termostata) bi lahko potekalo v samem prostoru, vendar bi potrebne meritve zahtevale ogromno časa, saj so časovne konstante procesa v razredu ur. Pristop z natančno matematično analizo pa je tudi problematičen saj histereza, ki predstavlja nelinearnost, že v zelo enostavni povratnozančni strukturi in pri do skrajnosti poenostavljenem modelu procesa povzroči kompliciranо dinamično obnašanje obravnavanega sistema. Zato je pristop z modeliranjem in simulacijo najustreznejši. Ko se odločimo za enostaven model gretja prostora, lahko namreč celotno regulacijsko zanko simuliramo s pomočjo primernega simulacijskega orodja in tako mnogo hitreje kot pa na realnem sistemu s poskušanjem dobimo potrebne informacije o zaprtozančnem obnašanju sistema.

Omenjena dejstva so pomenila osnovne podatke o sistemu in namenu modeliranja. Glede na to skušajmo priti do modela gretja prostora z grelom na najenostavnejši "intuitivni"način, pri čemer pa mora dobljeni model vseeno zadovoljiti naše zahteve.

Zato v prostoru pri relativno konstantni temperaturi (npr. 15^{o}C) prižgemo grelo moči 5KW ($t = 0$, $p = 5k$W) in v določenih časovnih intervalih (npr. 20 min) merimo temperaturo. Tako dobimo rezultate na sliki 1.4.



Slika 1.4: Gretje prostora ob vklopu grela (p...moč gretja, ϑ...temperatura v prostoru). Pike pomenijo merjene vrednosti, krivulja pa odziv modela.

Kot vidimo na sliki 1.4, je proces proporcionalen, kar smo tudi pričakovali, saj višanje temperature povzroča tudi večanje izgub toplote (skozi zidove, vrata, okna itd.). Izgube so namreč proporcionalne razliki med temperaturama prostora in okolice ($\vartheta - \vartheta_e$). Tako pridemo do ustaljenega stanja pri približno 25^{o}C (pri tej temperaturi je toplota, ki jo proizvaja grelo, enaka izgubam). Glede na zakon o energijskem ravnotežju lahko predpostavimo, da je odvod (sprememba) temperature proporcionalen razliki gretja in izgub, kar lahko zapišemo v obliki naslednje diferencialne enačbe:

$$k_1\dot{\vartheta} = k_2 p - k_3(\vartheta - \vartheta_e) \tag{1.12}$$

Če enačbo (1.2) delimo s konstanto k_1, namesto preostalih konstant pa vpeljemo časovno konstanto T in ojačenje k, dobimo diferencialno enačbo prvega reda

$$\dot{\vartheta} + \frac{1}{T}(\vartheta - \vartheta_e) = \frac{k}{T}p \tag{1.13}$$

kjer je pomen oznak naslednji:

ϑ - temperatura v prostoru [^{o}C],
ϑ_e - temperatura okolice [^{o}C], ki je približno konstantna in znaša 15^{o}C,
p - moč grela [kW] v našem primeru 5 kW,
k - ojačenje sistema prvega reda,
T - časovna konstanta sistema prvega reda.

Tako dobljeni linearni model dobro opisuje realni proces za temperaturni interval $15^o\text{C} \leq \vartheta \leq 25^o\text{C}$ pri konstantni temperaturi okolice. Običajno pa uporabljamo tako imenovane deviacijske modele, ki podajajo razmere relativno na delovno točko (v našem primeru je to temperatura okolice). To storimo s pomočjo spremenljivke ϑ_w, ki jo definiramo kot razliko

$$\vartheta_w = \vartheta - \vartheta_e \tag{1.14}$$

kar daje končno obliko modela ($\dot{\vartheta} = \dot{\vartheta}_w$)

$$\dot{\vartheta}_w + \frac{1}{T}\vartheta_w = \frac{k}{T}p \tag{1.15}$$

Model, ki ga podaja enačba (1.15) lahko zapišemo še v obliki prenosne funkcije

$$\frac{\theta_w(s)}{P(s)} = \frac{k}{Ts+1} \tag{1.16}$$

Postavitev enačbe (1.15) predstavlja teoretični del modeliranja. Konstanti k in T pa skušajmo določiti iz merjenih rezultatov, kar pomeni neko vrsto eksperimentalnega modeliranja. To kaže, da smo pri tem problemu uporabili pravzaprav kombinirano modeliranje. Za naš sistem prvega reda lahko časovno konstanto določimo s pomočjo tangente na krivuljo ob času t=0 (kot kaže slika 1.4). Kjer tangenta seka premico ustaljenega stanja, odčitamo časovno konstanto T. Le-ta je v našem primeru ocenjena na

$$T = 1\text{h}$$

Ojačenje sistema pa določa razmerje med spremembo temperature v ustaljenem stanju in spremembo vhodnega signala (v našem primeru gretja)

$$k = \frac{\Delta\vartheta}{\Delta p} = \frac{25 - 15}{5 - 0} = 2^o\text{C/kW} \tag{1.17}$$

Če dobljene vrednosti vstavimo v enačbo (1.15), dobimo model ogrevanja prostora

$$\dot{\vartheta}_w + \vartheta_w = 2p \tag{1.18}$$

oz. v obliki prenosne funkcije

$$\frac{\theta_w(s)}{P(s)} = \frac{2}{s+1} \tag{1.19}$$

V praksi pa se izkaže, da bi lahko ta model uporabili le v primeru, ko bi bil termostat, ki zaznava temperaturo prostora, zelo blizu grelnega telesa. Če temu ni tako, bi problem bolje opisovala diferencialna enačba višjega reda, ki jo lahko zadovoljivo aproksimiramo z modelom prvega reda v kombinaciji z mrtvim časom. V tem primeru je potrebno zapis (1.15) spremeniti v zapis z dvema enačbama

$$\begin{aligned} \dot{\vartheta}_w + \frac{1}{T}\vartheta_w &= \frac{k}{T}p_d \\ p_d(t) &= p(t - T_d) \end{aligned} \tag{1.20}$$

kjer je T_d mrtvi čas. Prenosna funkcija takega sistema je

$$\frac{\theta_w(s)}{P(s)} = \frac{k}{Ts+1}e^{-T_d s} \tag{1.21}$$

Zapis s pomočjo prenosne funkcije ponuja zelo ugodno in ilustrativno možnost predstavitve sistema s pomočjo bločnih diagramov. Pri študiju regulacijskih problemov (kot je to v našem primeru) lahko tako jasno prikažemo celotno strukturo regulacijske zanke.

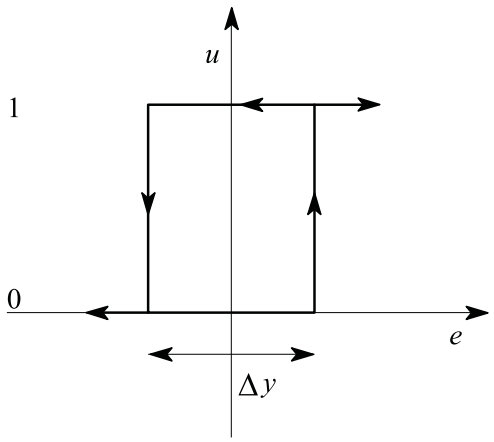
Zgradimo bločni diagram regulacije gretja prostora. Želeno temperaturo prostora (ϑ_r) nastavimo na termostatskem regulatorju. Vhod v slednjega je tudi temperatura v prostoru ϑ. Termostatski regulator ustvari razliko

$$e = \vartheta_r - \vartheta \tag{1.22}$$

iz katere preko histerezne karakteristike

$$u(t) = \begin{cases} 0, & e < -\frac{\Delta y}{2} \\ 1, & e > \frac{\Delta y}{2} \\ u(t - \Delta t), & -\frac{\Delta y}{2} \leq e \leq \frac{\Delta y}{2} \end{cases} \tag{1.23}$$

določi regulirni signal. Ustrezno zakonitost prikazuje slika 1.5. Pri tem je $u(t-\Delta t)$ pretekla vrednost regulirnega signala (seveda 0 ali 1, pri tem je Δt poljubno majhen).



Slika 1.5: Histereza termostata

Regulirni signal u vklaplja in izklaplja grelec (npr. preko releja, kontaktorja). Letega lahko modeliramo kar z nekim konstantnim ojačenjem p_{max}, oz. z enačbo

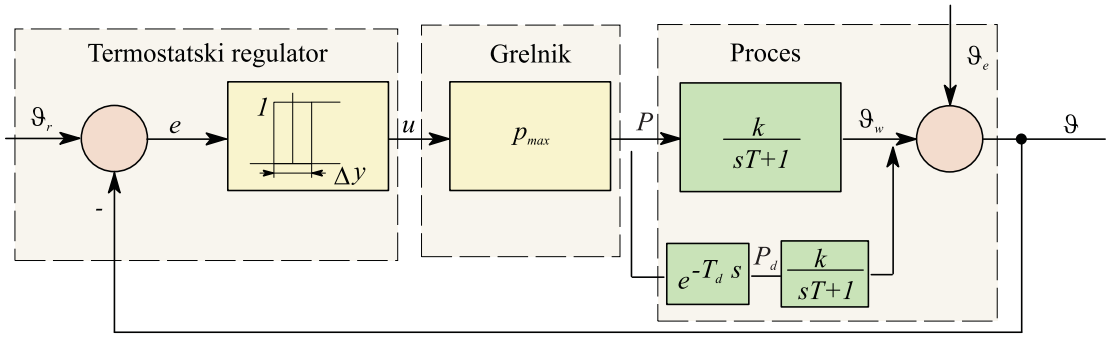
$$p = p_{max} u \tag{1.24}$$

Dobljeni signal p predstavlja vhod v model procesa (enačba (1.15) za $T_d = 0$ in enačbi (1.20) za $T_d \neq 0$). Ker smo razvili deviacijski model, s simulacijo pa bi radi

opazovali absolutne iznose spremenljivk, dobimo absolutni iznos temperature z vsoto temperature v okolici delovne točke (ϑ_w) in temperature delovne točke, ki je kar temperatura okolice (ϑ_e)

$$\vartheta = \vartheta_w + \vartheta_e \tag{1.25}$$

Bločni diagram regulacijske zanke pri gretju prostora tako prikazuje slika 1.6. V gradniku za proces sta navedeni obe možnosti: model prvega reda in model prvega reda z mrtvim časom.



Slika 1.6: Bločni diagram regulacije temperature prostora

Tako je problem pripravljen za nadaljnjo obravnavo. Naštejmo nekaj možnosti za študij različnih aspektov našega problema s pomočjo razvitega modela:

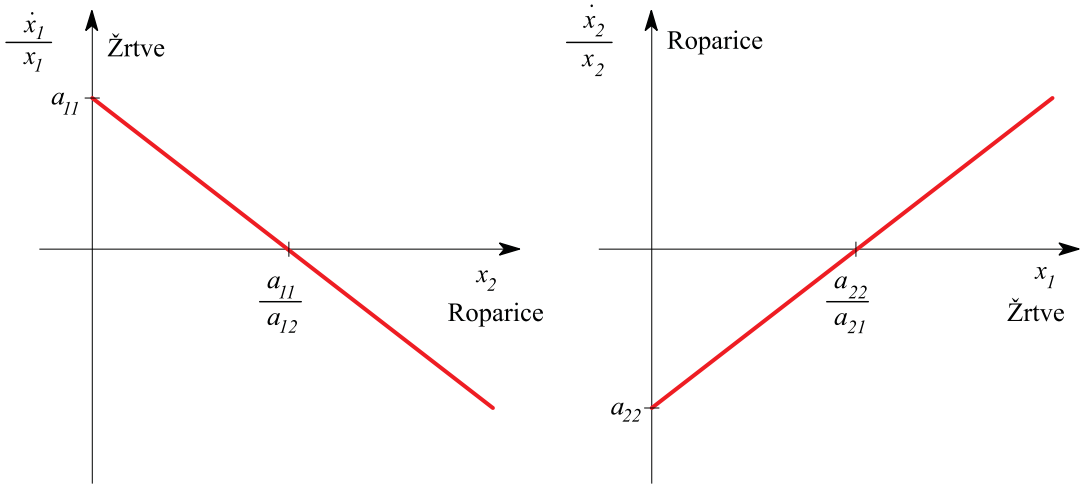
- določitev širine histereze termostata, ki daje majhna nihanja temperature ob ne preveč pogostem preklapljanju grela,
- določitev minimalne moči grela, ki še zagotavlja zadovoljivo regulacijo (sprejemljivo hitrost prilagajanja temperature novi želeni vrednosti, neobčutljivost na motnje itd.),
- študij vpliva motenj na regulirano temperaturo (možno je enostavno simulirati različne motnje na vhodu in izhodu procesa ali pa tudi na kakšnem drugem mestu),
- določitev optimalnega poteka želene temperature, ki obenem s primernim temperaturnim profilom v prostoru zagotavlja optimalno izrabo energije,
- študij vpliva lege termostata na temperaturo prostora (kar je posebno pomembno pri slabem mešanju zraka v prostoru),

- študij vpliva različnih časovnih konstant gretja in hlajenja prostora na temperaturo v prostoru itd.

□

Primer 1.3 Primer modeliranja populacijske dinamike

Oglejmo si še primer modeliranja enostavnega netehniškega problema. Pri populacijski dinamiki gre za študij rasti ali upadanja različnih populacij. Ta problem je dobro znan iz literature in če ga razširimo na obravnavo dveh populacij, ga imenujemo *model ekosistema Lhotka-Volterra*, še večkrat pa *problem roparic in žrtev*. Če jemljemo populaciji kot zvezni spremenljivki, je logaritmična rast ene odvisna le od druge populacije. Logaritmična rast populacije roparic je sorazmerna populaciji žrtev in logaritmična rast populacije žrtev je sorazmerna populaciji roparic. Ob predpostavki, da je sorazmernost linearna, dobimo odvisnosti, kot jih prikazuje slika 1.7. Pri tem smo označili populacijo žrtev z x_1, populacijo roparic z x_2, $a_{11}, a_{12}, a_{21}, a_{22}$ pa so konstante modela.



Slika 1.7: Logaritmični rasti populacij žrtev in roparjev

S pomočjo slike 1.7 lahko napišemo naslednji matematični model

$$\begin{aligned} \frac{\dot{x}_1}{x_1} &= a_{11} - a_{12}x_2 \\ \frac{\dot{x}_2}{x_2} &= a_{21}x_1 - a_{22} \end{aligned} \tag{1.26}$$

ali

$$\begin{aligned} \dot{x}_1 &= (a_{11} - a_{12}x_2)x_1 \\ \dot{x}_2 &= (a_{21}x_1 - a_{22})x_2 \end{aligned} \tag{1.27}$$

Pri tem zanemarimo faktorje, ki bi povzročali upadanje rasti neke populacije in bi izvirali v tej isti populaciji. Tako se pri odsotnosti žrtev populacija roparic zmanjšuje, pri odsotnosti roparic pa se populacija žrtev zvečuje. Pozitivni konstanti a_{11} in a_{22} zavisita od hitrosti rasti, pozitivni konstanti a_{12} in a_{21} pa predstavljata vzajemne faktorje zadrževanja rasti in sta proporcionalni velikosti druge populacije.

Če bi naša razmišljanja prenesli na zajce (žrtve) in lisice (roparice), lahko konstante v enačbah (1.27) ocenimo na teoretičen način s pomočjo naslednjih predpostavk:

1. Vsak par zajcev ima povprečno 10 mladičev letno.
2. Vsaka lisica poje povprečno 25 zajcev letno.
3. Povprečna starost lisic je 5 let, kar pomeni, da letno umre 20 % lisic.
4. V povprečju število mladih lisic, ki preživijo, zavisi od dosegljive hrane (število preživelih mladih lisic je torej enako številu zajcev deljeno s 25).

Za časovno enoto eno leto lahko torej izračunamo konstanti a_{11} in a_{22} iz predpostavk 1 in 3 na naslednji način:

$$\begin{aligned} a_{11} &= \left.\frac{\dot{x}_1\prime}{x_1}\right|_{a_{12}=0} \approx \frac{\Delta x_1\prime/\Delta t}{x_1} = \frac{10}{2} = 5 \\ a_{22} &= \left.-\frac{\dot{x}_2\prime}{x_2}\right|_{a_{21}=0} \approx \frac{\Delta x_2\prime/\Delta t}{x_2} = -\frac{-1}{5} = 0.2 \end{aligned} \tag{1.28}$$

Kot smo omenili, sta konstanti a_{12} in a_{21} odvisni tudi od področja na katerem opazujemo populaciji, kar pomeni, da sta odvisni od povprečnega števila zajcev in lisic. Če npr. opazujemo območje 50 km^2, kjer je povprečno število zajcev $\overline{x}_1 = 500$ in lisic $\overline{x}_2 = 100$, predpostavka 2 daje

$$a_{12} = -\frac{\dot{x}_1''}{\overline{x}_1\overline{x}_2}\bigg|_{a_{11}=0} \approx \frac{\Delta x_1''/\Delta t}{\overline{x}_1\overline{x}_2} = -\frac{25\overline{x}_2}{\overline{x}_1\overline{x}_2} = \frac{25}{500} = 0.05 \quad (1.29)$$

predpostavka 4 pa

$$a_{21} = \frac{\dot{x}_2''}{\overline{x}_1\overline{x}_2}\bigg|_{a_{22}=0} \approx \frac{\Delta x_2''/\Delta t}{\overline{x}_1\overline{x}_2} = \frac{\overline{x}_1/25}{\overline{x}_1\overline{x}_2} = \frac{1}{25\overline{x}_2} = \frac{1}{2500} = 0.0004 \quad (1.30)$$

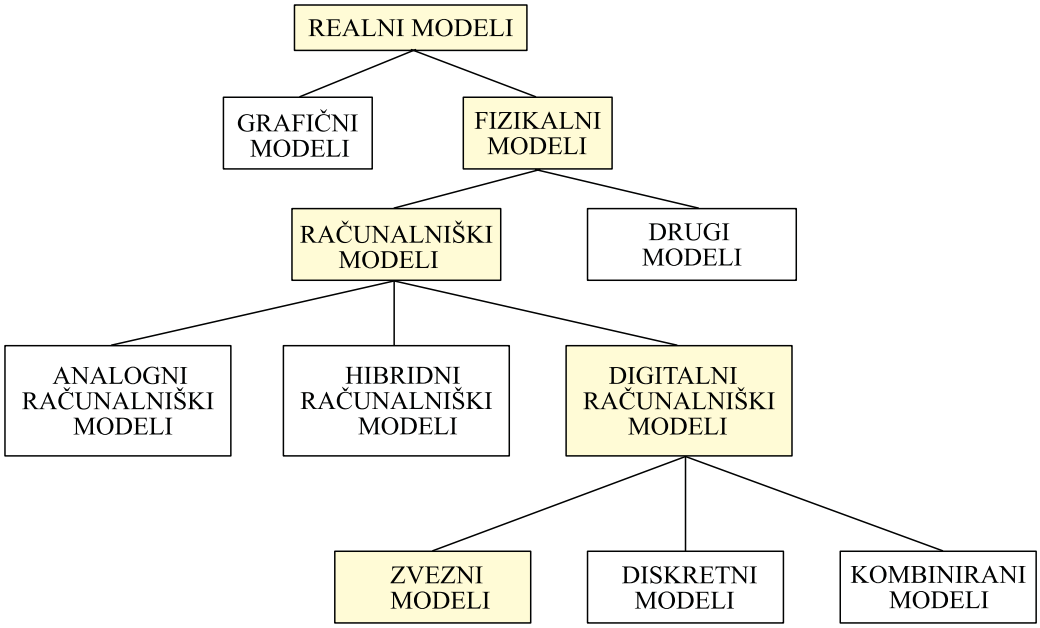
Tako smo prišli do matematičnega modela za obravnavani primer:

$$\begin{aligned} \dot{x}_1 &= 5x_1 - 0.05x_1x_2 \\ \dot{x}_2 &= 0.0004x_1x_2 - 0.2x_2 \end{aligned} \quad (1.31)$$

Model predstavlja sistem dveh navadnih nelinearnih diferencialnih enačb prvega reda s konstantnimi koeficienti. □

1.5 Razvrstitev realnih oz. simulacijskih modelov

Slika 1.7 prikazuje razvrstitev realnih oz. simulacijskih modelov. Pri grafičnih modelih elemente in funkcije konceptualnega modela predstavljamo v grafični obliki (npr. x-y krivulja, skica arhitekta, ...). Fizikalni modeli pa so opisani s fizikalnimi zakoni. Najbolj uporabna podmnožica fizikalnih modelov so računalniški modeli. Glede na to, s kakšnimi računalniki jih izvedemo, jih delimo na analogne, hibridne in digitalne računalniške modele. Analogni računalniški modeli so ugodni za simulacijo zveznih dinamičnih sistemov, odlikuje jih predvsem velika hitrost simulacije in so zato tudi zelo primerni za simulacijo v realnem času. Digitalni računalniški modeli so zlasti primerni za simulacijo diskretnih dogodkov in diskretnih sistemov (diskretni modeli), toda z uporabo numerične integracije se uspešno uporabljajo za simulacijo zveznih dinamičnih sistemov (zvezni modeli) in za simulacijo kombiniranih sistemov (kombinirani modeli). Odlikuje jih večja natančnost, široka možnost uporabe in dobre zmožnosti vhodno-izhodnih operacij. Hibridni računalniški modeli izkoriščajo dobre lastnosti analognih in digitalnih računalnikov.



Slika 1.8: Klasifikacija modelov

1.6 Načrtovanje sistemov vodenja, varnosti in zanesljivosti

Zaradi izrednega napredka znanja in tehnologije so računalniške obravnave za vodenje procesov, ki so še pred nekaj leti izgledale utopične, postale rutinsko delo. Načrtovanje sistemov vodenja pa ne more biti ločena naloga inženirja za načrtovanje vodenja, ampak je le del kompleksnega postopka, v katerem simulacija kot metoda za načrtovanje vodenja, varnosti in zanesljivosti igra pomembno vlogo. Simulacija se uporablja v naslednjih projektnih fazah (Tyso, 1985):

- za načrtovanje vodenja sistemov,
- za odkrivanje in spoznavanje napak,
- za varen zagon in ustavitev procesa,
- za vadbo operaterjev in
- kot orodje za pomoč operaterjem pri odločanju.

Uporaba simulacije za načrtovanje vodenja sistemov

Simulacija se v povezavi z načrtovanjem vodenja procesov v glavnem uporablja za razvoj metode ter za razvoj konkretne rešitve vodenja in eksperimentiranje (Heinrich, 1986).

Klasični primer uporabe simulacije kot orodja za razvoj metode predstavlja razvoj postopkov za načrtovanja regulatorjev PID, ki so jih podali Chien, Hrones in Reswick ter Ziegler in Nichols v začetku petdesetih let. Te metode se še danes uporabljajo. Novejše metode vodenja so seveda bolj zahtevne, imajo pa tudi več omejitev, ki jih v praksi ponavadi težko izpolnimo. Zato je simulacija primerna za preverjanje uspešnega delovanja metode, za analizo stabilnosti, konvergentnosti in robustnosti. Razen tega je simulacija vgrajena tudi v številne druge metode za analizo in načrtovanje vodenja sistemov (npr. optimalni, adaptivni sistemi, načrtovanje multivariabilnih regulatorjev, ...).

Pri razvoju konkretnega vodenja in pri eksperimentiranju uporabljamo simulacijo, ker eksperimenti na realnem objektu običajno niso možni, saj so mnogokrat povezani z velikimi stroški ali tveganjem. Uspešnost preizkušanja konkretne rešitve s simulacijo seveda v največji meri zavisi od vernosti simulacijskega modela. Prav zato, ker je potrebno zgraditi kvaliteten model, se danes v industrijskem okolju relativno malo uporabljajo paketi CACSD, čeprav jih je veliko na trgu. Postavitev kvalitetnega modela zahteva veliko znanja, izkušenj, časa ter ustrezne instrumentacije, kar je povezano s stroški, ki pa so mnogo manjši od potrebnih vlaganj za gradnjo prototipov ali pilotnih obratov, kar predstavlja alternativno možnost modeliranju in simulaciji.

Uporaba simulacije za odkrivanje in spoznavanje napak

Odkrivanje in spoznavanje napak (FDD - fault detection and diagnosis) predstavlja sodobno področje pri vodenju sistemov. S sprotnim spremljanjem dogajanja (parametrov) v sistemu vodenja je možno pravočasno odkriti oz. predvideti napako, preden povzroči padec kvalitete proizvodov ali celo prekinitev proizvodnje. To pa lahko pomeni precejšnje prihranke. Napredni sistemi uporabljajo simulacijski model, ki deluje paralelno z realnim objektom. Tako je možno delati primerjavo med realnimi meritvami in simuliranimi vrednostmi. Razen tega je možno s sodobnimi metodami izvesti tudi ocenjevanje stanj in parametrov, ki niso merljivi in napovedovanje določenih spremenljivk. Ob nenormalnih situacijah sistem za odkrivanje lahko sproži delovanje sistema za spoznavanje napak,

ki na podlagi ugotovljenih dejstev pri sprotnem spremljanju ugotovi napako. Metode spoznavanja v zadnjem času temeljijo na pristopih umetne inteligence (npr. spoznavanje z nevronsko mrežo). Sistem za odkrivanje in spoznavanje napak pravočasno opozori operaterja, ki nato posreduje.

Uporaba simulacije za varen zagon in ustavitev procesa

Pri zagonu in ustavitvi procesa nastanejo prehodni pojavi, ki igrajo pomembno vlogo pri projektiranju sistema vodenja. S simulacijo zagona ali ustavitve lahko preizkušamo razne možne načine delovanja v nenormalnih pogojih. Preizkušamo lahko tudi zanesljivost in robustnost sistemov ob morebitnih napakah senzorjev in aktuatorjev ob upoštevanju njihove dinamike.

Uporaba simulacije za vadbo operaterjev

Dobro izurjeni operater je nujno potreben za nemoteno obratovanje vsakega realnega kompleksnega procesa, čeprav le-ta že vsebuje sistem vodenja. Računalniška simulacija omogoča hitro učenje operaterja. Na simuliranem sistemu lahko operater v enem dnevu doživi več težkih pogojev, v katerih mora poseči v proces, kot kasneje morda v več letih na realnem objektu. Pomembno pa je, da je simulator čim bolj verna reprodukcija dejanskega objekta z vsemi detajli, sicer bi lahko bila izurjenost operaterja le navidezna.

Uporaba simulacije za pomoč operaterju pri odločanju

Čeprav je vodenje v nekaterih primerih povsem avtomatizirano, lahko pride do situacij, ko mora operater ročno poseči v proces. Ročno vodenje pa je problematično pri procesih z velikimi zakasnitvami, pri fazno neminimalnih ter multivariabilnih sistemih ipd. V takih primerih je potrebno, da je paralelno k procesu priključen simulator, ki deluje hitreje kot v realnem času. Z njegovo pomočjo lahko operater preizkusi brez tveganja vse potrebne akcije.

1.7 Razlogi za morebitno neuspešnost simulacijskih projektov

Simulacija je sicer učinkovita metoda za preizkušanje modelov preden gremo v gradnjo dragih prototipov in njih dejansko implementacijo. V primerjavi z drugimi metodami analize in načrtovanja je simulacija bolj realistična in lažje razumljiva, toda le, če je pravilno uporabljena. Annino (Annino, 1979) omenja, da v sedemdesetih letih veliko izredno dragih simulacijskih projektov ni dalo želenih rezultatov. Omenili bomo nekaj glavnih razlogov za neuspešno uporabo simulacije.

Slabo definiran končni cilj

Cilj simulacijskega projekta ne sme biti nikoli model zaradi modela. Najbolj pomembno je natančno definirati, za kaj bodo služili rezultati simulacije. Zato je skrbna postavitev končnih ciljev najpomembnejša naloga. Načrtovalci prav tu največkrat zagrešijo napako.

Slabo sestavljena ekipa

Simulacijski projekt lahko zadovoljivo opravi le ekipa, ki ima izkušnje iz različnih področij:

- vodenje projekta (sposobnost motiviranja, vodenja),
- modeliranje (zmožnost razviti konceptualni model, ki naj čim bolje opisuje sistem, toda le do ustreznega nivoja podrobnosti),
- programiranje (zmožnost pretvoriti konceptualni model v simulacijski model v obliki dobro čitljivega programa z možnostmi za enostavno spreminjanje),
- znanje o realnem sistemu (zmožnost vrednotenja rezultatov simulacije).

Najbolj problematična je prav zadnja točka, ki pomeni povezavo med načrtovalci in uporabniki rezultatov.

Neustrezni nivo podrobnosti

Model vedno predstavlja poenostavljen sistem, zato naj upošteva le tiste značilnosti, ki so pomembne za uporabnike rezultatov. Načrtovalci običajno nekatere dele sistema poznajo in razumejo bolj kot druge. Pri tem je nevarno, da bolj podrobno modelirajo tiste dele, ki jih bolje razumejo. Zato je potrebno ugotoviti podrobnosti, ki v večji meri vplivajo na rezultate in te vključiti v model. Torej je zelo pomembno v vsakem simulacijskem projektu definirati ustrezni nivo podrobnosti.

Slabo komuniciranje

Pomembno je dobro komuniciranje med člani moštva, saj morajo vsi sodelovati pri postavitvi končnih ciljev. Pri tem ima lahko velik učinek izbira ustreznega problemsko orientiranega simulacijskega jezika, ki omogoča dobro in enostavno komuniciranje med člani ekipe tudi preko simulacijskega modela, hkrati pa seveda zelo olajša programiranje.

Izbira neustreznega računalniškega jezika

Nekateri menijo, da ima visok problemsko orientiran simulacijski jezik bistvene prednosti, drugi pa menijo, da je možno uporabiti splošno namenske jezike kot npr. Matlab (le kot programski jezik, brez simulacijskih dodatkov), Basic, Visual Basic, C++, Fortran, Java script, Java, ASP, Phyton, Visual Studio, itd. in s tem priti tudi do določenih prednosti. Izkušnje kažejo, da problemsko orientiran simulacijski jezik bistveno skrajša čas za razvoj modela in simulacijskega projekta, omogoča pa tudi večjo fleksibilnost in čitljivost.

Pomanjkljiva dokumentacija

Dokumentacija simulacijskega projekta je zelo pomembna, zahteva veliko časa in navadno povzroča precej težav. V končni fazi, ko je dokumentacija v glavnem že izdelana, še vedno prihaja do manjših ali večjih sprememb, včasih tudi konceptualnih. Ponavadi je laže popraviti program kot pa priročnike. Za same razvijalce

je zelo pomembno tudi komentiranje v izvornih programih, ki ga lahko relativno enostavno in hitro sproti dopolnjujemo.

Uporaba nepreverjenega simulacijskega modela

Zelo pomembno je verificirati simulacijski model (program) s konceptualnim modelom. To najbolj učinkovito naredita načrtovalec simulacijskega modela in oseba, ki je sodelovala pri snovanju konceptualnega modela in dobro pozna realni objekt. Zato pa je izredno pomembno, da je izvorni program dobro čitljiv.

Napake zaradi neuporabe modernih programskih orodij za vodenje in načrtovanje velikih projektov

Vzrok za kasnitev pri mnogih velikih projektih je v tem, da razvijajo simulacijski model, še predno je dokončan konceptualni model ali ker je plan razvoja preveč optimističen (ne predvidi se časa za nepredvidljive stvari, ki v določeni fazi zahtevajo veliko časa). Moderna programska orodja za vodenje in načrtovanje velikih projektov (orodja programskega inženirstva) lahko precej pomagajo pri premostitvi takih problemov in lahko zelo skrajšajo čas projekta (Steppard, 1983).

Neprimerna oblika rezultatov simulacije

Rezultati simulacije morajo biti v taki obliki, da jih uporabnik lahko na enostaven način poveže in primerja z realnim sistemom. V nasprotnem primeru uporabnik ne dobi zaupanja v model.

1.8 Zgodovinski razvoj simulacijskih orodij, metod in organiziranosti

Različni modeli, ki so ponazarjali realne objekte (npr. zgradbo atoma) so znani že iz prejšnjih stoletij. Šele z razvojem računalnikov pa lahko govorimo o sodobni simulaciji. Leta 1930 sta Howard Aiken iz Harvarda in George Stibitz iz Bell Telephone Laboratory razvila prvi električni relejski računalnik. Leta 1946 pa

sta John Mauckley in Prisper Eckert iz Univerze v Pensylvaniji končala z razvojem sistema ENIAC (Electronic Numerical Integrator and Calculator), ki je imel namesto relejev elektronske cevi. To je bil prvi elektronski digitalni računalnik. Razvijalci so ustanovili računalniško podjetje, iz katerega je leta 1950 prišel na trg Eckert-Mauckleyev UNIVAC. Toda ti prvi računalniki so bili za simulacijo neprimerni zaradi skromnih računskih zmožnosti, majhne kapacitete pomnilnika in zaradi izredno slabih vhodno - izhodnih naprav. K sreči pa je za uporabnike tistega časa Georg A. Philbrick iz podjetja Foxboro že v letih 1937-38 razvil "An automatic control analyser", ki je bil prvi analogni računalnik za posebne namene. V letu 1943 so John R. Regazzini, Robert H. Randall in Frederick A. Russell s kolumbijske univerze razvili analogni računalnik, ki so ga imenovali "An electronic system for obtaining an engineering solution for integrodifferential equations of physical systems". Okoli leta 1945 je bila tudi že znana simulacija po metodi Monte Carlo. John von Neuman in Stanislav Ulam iz Laboratorija v Los Alamosu sta z njo reševala problem difuzije neutronov.

V tistih časih so bili analogni računalniki edino primerno orodje za simulacijo zveznih sistemov, digitalni računalniki pa so se uporabljali za simulacijo diskretnih dogodkov. Glavna odlika analognih računalnikov je bila velika hitrost, digitalne računalnike pa je odlikovala velika natančnost. V Evropi pomeni eno prvih simulacijskih orodij večjih sposobnosti analogni računalnik TRIDAC, ki je začel delovati leta 1955 v Angliji. Imel je elektronske, mehanične in hidravlične komponente.

Že zgodaj pa so se pokazale tudi slabosti obeh vrst računalnikov. Zato so se že v petdesetih letih pojavile ideje o združitvi analognega in digitalnega računalnika. Sredi petdesetih let pa se pojavijo prvi programi za digitalne računalnike, ki omogočajo reševanje diferencialnih enačb z numerično integracijo. Leta 1955 je Selfridge prvi podal idejo bločno orientiranega simulacijskega jezika, s pomočjo katerega bi probleme na digitalnem računalniku reševali podobno kot na analognem. Leta 1959 pa so Stein, Rose in Parker poročali o prevajalniškem konceptu digitalne simulacije.

Šestdeseta leta predstavljajo velik napredek na področju analognih in digitalnih računalnikov. Z njihovo pomočjo je bilo že možno izvajati kompleksne simulacije. Leta 1963 je Electronic Associetes (EAI) dal na trg hibridni računalnik HYDAC, t.j. prvi računalnik z digitalnim krmiljenjem in logiko. V sredini šestdesetih let so se pojavili prvi hibridni sistemi s povezavo splošno namenskega analognega in splošno namenskega digitalnega računalnika. Prav tako so takrat prišli na trg prvi sposobni simulacijski jeziki, ki so temeljili na jeziku FORTRAN. Leta 1965 je bil dokončan DSL 90, prvi IBM-ov bločno orientirani simulacijski jezik, ki je

služil za razvoj kasnejših verzij jezikov CSMP. Pravi mejnik glede na sposobnosti pa predstavlja CSMP 360, ki je prišel na trg leta 1966. Že leta 1967 pa je Simulation Council (predhodnik današnje The Society for Computer Simulation) v reviji Simulation (Strauss, 1967) objavil standard, po katerem naj bi bili izdelani prihodnji jeziki za simulacijo zveznih dinamičnih sistemov. Standard se je uspešno obdržal vse do danes.

V sedemdesetih letih so se začeli digitalni računalniki mnogo bolj množično uporabljati kot analogni zaradi bistveno nižje cene ter vedno večje računske sposobnosti in interaktivnosti. Analogni in hibridni računalniki so se največ uporabljali v posebne namene (simulacija v realnem času, uporaba v visokotehnoloških letalskih, vesoljskih in vojaških projektih, zahtevne regulacije, jedrski reaktorji) in pa v pedagoške namene. Najbolj znani hibridni računalniki tega obdobja so računalniki firme Electronic Asociates (EAI 580, EAI 680, EAI 1000, EAI 2000). Sredi sedemdesetih let so pri ADI (Applied Dynamics International) ocenili, da hibridni sistemi nimajo prave prihodnosti v vedno bolj kompleksnem modeliranju, pri katerem se zahteva tudi velika natančnost. Odločili so se, da gredo v razvoj povsem digitalnega večprocesorskega simulacijsko specializiranega procesorja AD-10. Tako je že konec sedemdesetih let prišel na trg izredno sposoben večprocesorski simulacijski sistem. Tudi digitalni simulacijski produkti so doživeli velik razvoj. Na ta razvoj je razen simulacijskega standarda vplival tudi razvoj numeričnih metod, matematičnih knjižnic in kasneje obsežnih paketov CACSD. Zato je bil pri načrtovanju jezikov večji poudarek na numerični robustnosti, fleksibilnosti in interaktivnosti. Najbolj znan produkt tega obdobja je simulacijski jezik CSSL.

V osemdesetih letih je prišlo do ekspanzije mikroračunalnikov in osebnih računalnikov. Ker so le - ti postajali vedno bolj zmogljivi, so začeli na njih prenašati simulacijske pakete, ki so do takrat delovali le na večjih računalnikih. Najbolj znana splošno namenska simulacijska jezika sta CSSL IV in ACSL. V tem obdobju se začenja kazati težnja po novem standardu CSSL in na podlagi priporočil dveh delovnih komisij (pri IMACS -International Association for Mathematics and Computers in Simulation in pri SCS - The Society for Computer Simulation) se sredi 80 let začnejo razvijati simulacijski jeziki nove generacije (ESL, SYSMOD, COSMOS). Simulacijska orodja so postala dostopna vsakomur in množična uporaba simulacije je močno vplivala na razvoj nekaterih teoretičnih področij. Prav tako je za to obdobje značilen velik razmah modernih paketov za analizo in načrtovanje vodenja sistemov, v katerih je vedno vključena tudi simulacija. Značilen za to obdobje je tudi razvoj superračunalnikov, vrstičnih procesorjev in paralelnih procesorjev. Nova materialna oprema je zelo vplivala na razvoj simulacijskih orodij. Pojavljajo se simulacijske delovne postaje z vrstičnimi procesorji, z barvnimi

grafičnimi zasloni velike ločljivosti, z veliko stopnjo interaktivnosti ter z izrednimi zmožnostmi za delo v realnem času (po hitrosti se približujejo analognim računalnikom). Tudi izdelovalci hibridnih sistemov niso ostali na nivoju sedemdesetih let. Electronic Associates je dala leta 1983 na trg paralelnoprocesorski sistem SIMSTAR, ki konceptualno predstavlja povsem novo vrsto hibridnega sistema. Povezava računalnikovih elementov v program se izvrši avtomatsko, programira pa se s pomočjo simulacijskega jezika.

Devetdeseta leta so prinesla velik napredek predvsem pri razvoju uporabniških vmesnikov simulacijskih orodij. Tako na osebnih računalnikih kot na sposobnih delovnih postajah prevladujejo koncepti okenskih vmesnikov. Ena najpomembnejših lastnosti je opis modela na grafični način. Take vmesnike so razvili za starejše simulacijske produkte (npr. za ACSL), imajo pa jih seveda vsi na novo razviti produkti (npr. SIMULINK v okolju MATLAB, SYSTEM_BUILT v okolju MATRIXx, MODEL-C v okolju CONROL-C, paket CC, EASY5 - to so vse paketi za računalniško podprto načrtovanje vodenja sistemov, kjer pa je simulacija osrednja metoda). Nesluten razvoj je naredilo programsko okolje MATLAB tudi na področju simulacij zlasti s paketom SIMULINK. MATLAB-SIMULINK je postalo standardno okolje na vseh akademskih institucijah, kasneje pa se je izdatno začelo uporabljati tudi v industriji. Kasneje so za SIMULINK razvili razne namensko uporabne dodatke, eno je npr. STATE FLOW, ki omogoča modeliranje dogodkovnih procesov. Velik je tudi poudarek na objektni orientiranosti (npr.Xmath). Nekatera orodja uvajajo tudi večjo podporo v smislu modeliranja (npr. DYMOLA z orodji DYMODRAW, DYMOVIEW, DYMOSIM).

V novem tisočletju je simulacija najbolj zaznamovana z objektno orientiranim in več domenskim jezikom Modelica. Podobno idejo objektne orientiranosti vsebujejo tudi Bond grafi - grafična modelerska tehnika, ki opisuje pretok energije med komponentami (podpira npr. simulacijski paket 20-sim). Modelica postaja priznani standard za modeliranje zveznih pa tudi diskretnih in hibridnih dinamičnih sistemov. Omogoča zlasti veliko podporo modeliranju, saj ni potrebno izraziti odvodov stanj, kot v primeru večine konvencionalnih simulacijskih orodij. Zlasti je pomemben način povezovanja komponent, ki omogoči gradnjo knjižnic ponovno uporabljivih komponent. Povezujemo pa lahko komponente različnih področij, kar je zlasti pomembno v mehatroniki, robotiki, v avtomobilski industriji, v vodenju sistemov ipd. Zlasti sposobni okolji, ki podpirata jezik Modelica, sta Dymola in MathModelica. Podoben način je uvedel tudi MathWorks l. 2008 z okoljem Simscape v sklopu programskega paketa Matlab-Simulink. Žal pa ta način ne spoštuje standarda Modelica.

Tabela 1.1 predstavlja pregled razvoja simulacijskih metod in orodij.

Tabela 1.1: Pregled razvoja simulacijskih metod in orodij

1930	prvi električni relejski računalnik - Howard Aiken (Harvard) in George Stibitz (Bell Telephone Laboratory)
1938	prvi analogni računalnik - "An automatic control analyser- Georg A. Philbrick (Foxboro)
1943	"An electronic system for obtaining an engineering solution for integrodifferential equations of physical systems- John R. Regazzini, Robert H. Randall in Frederick A. Russell (univerza Kolumbija)
1946	ENIAC (Electronic Numerical Integrator and Calculator) - John Mauckley in Prisper Eckert (Univerza v Pennsylvaniji)
1950	začetek razvoja splošnonamenskih analognih računalnikov
1955	SELFRIDGE - prvi simulacijski jezik
1960	prvi hibridni sistemi (EAI)
1965	DSL 90 - prvi prevajalniški enačbno orientirani jezik
1967	CSMP 360, CSMP III - prvi sposobni simulacijski jeziki, funkcijski generatorji, 60 operatorjev, reševanje algebrajske zanke
1967	**standard CSSL 67**
1968	MIMIC - prvi jezik po vzoru CSSL 67
1969	CSSL III - prvi sposobni jezik CSSL
1970	hibridni računalniki (EAI 580, EAI 680, EAI 2000)
1972	CSSL IV - v preteklosti eden najsposobnejših simulacijskih jezikov
1972	HYSIM - kombinirana simulacija, razvoj na takratni Fak. za el. in rač., UL
1975	SIMNON - sposoben interaktivni jezik, prevajanje direktno na strojni nivo, uporabniku ni potrebno integrirati odvodov
1975	ACSL - vsestransko dober, komercialno uspešen simulacijski jezik
1975	AD-10, AD-100 - simulacijsko specializirana digitalna procesorja (Applied Dynamics International)
1980	ekspanzija mikroračunalnikov, CACSD paketov, superračunalnikov, paralelnoprocesorskih sistemov
1983	SIMSTAR - hibridni večprocesorski sistem
1983	HYBSIS, STARTRAN - simulacijska jezika za hibridne sisteme, simulacija v realnem času
1984	ADSIM, PARSIM - simulacijska jezika za namenske simulacijske računalnike, simulacija v realnem času
1984	ESL, SYSMOD, COSMOS - simulacijski jeziki nove generacije
1988	Xanalog- simulacijska delovna postaja, grafifični vnos modela
1989	SIMCOS - zvezna in diskretna simulacija, eksperimentiranje, delovanje v realnem času, razvoj na takratni Fak. za el. in rač., UL
1990	MATRIXx, SYSTEM-Built, CONTROL-C, MODEL-C, EASY5 - kompleksni CACSD sistemi
1990	SIMULINK - grafični uporabniški vmesnik, delovanje v okolju MATLAB
1992	okolje DYMOLA - podpora za modeliranje, objektna orientiranost
1996	jezik MODELICA - poizkus standardizacije jezika za objektno orientirano več domensko modeliranje - nov standard po CSSL'67
2008	Simscape, rešitev podjetja Mathworks za več domensko objektno orientirano modeliranje

Hkrati z razvojem simulacijskih orodij in simulacijskih metod se je razvijala tudi simulacijska organiziranost, vendar predvsem v ZDA. Tam se je že okoli leta 1950 pokazalo, da različni laboratoriji zaradi slabe povezave delajo na enakih ali podobnih problemih. John Mc Leod iz Naval Air Missile Test Centra v Kaliforniji je skušal združiti laboratorije pod eno organizacijo, vendar ni uspel. Zato je leta 1952 ustanovil lastno zvezo, ki je bila predhodnica današnje "The Society for Modeling and Simulation International (SCS)- Simulation Council. Tej zvezi se je kmalu priključilo veliko laboratorijev. Začela je izdajati tudi revijo Newsletter, ki je pozneje prerasla v revijo na področju simulacije SIMULATION.

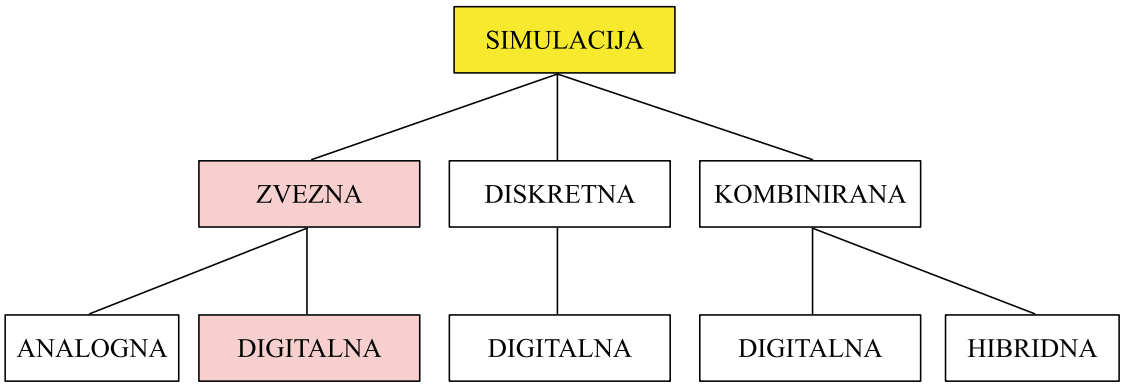
V Evropi je SCS šele leta 1985 odprla t.i. evropski urad v Ghentu. Leta 1989 pa se je ustanovilo evropsko združenje EUROSIM. Osnovna naloga je koordinacija dela, konferenc, simpozijev in drugih dejavnosti nacionalnih simulacijskih združenj: ASIM Arbeitsgemeinschaft Simulation (Avstrija, Nemčija, Švica) CROSSIM Croatian Society for Simulation Modelling, CSSS Czech and Slovak Simulation Society, DBSS Dutch Benelux Simulation Society (Belgija, Nizozemska), FRANCOSIM Societe Francophone de Simulation (Belgija, Francija), HSS Hungarian Simulation Society, ISCS Italian Society for Computer Simulation, PSCS Polish Society for Computer Simulation, SIMS Simulation Society of Scandinavia (Danska, Finska, Norveška, Švedska), SLOSIM Slovenian Society for Simulation and Modelling, UKSIM United Kingdom Simulation Society (Velika Britanija, Irska), CEA SMSG Spanish Modelling and Simulation Group, LSS- Latvian Society for Simulation in ROMSIM - Romanian Society for Modelling and Simulation. Pod okriljem EUROSIMa izdaja založba Elsevier revijo Simulation Modelling Practice and Theory, zveza ASIM pa časopis/revijo Simulation News Europe.

Zveze prirejajo številne konference. V Evropi so najpomembnejši simulacijski kongresi EUROSIM (l. 2007 ga je v Ljubljani organiziralo slovensko društvo SLOSIM) v zvezi s simulacijo na področju vodenja pa so zlasti pomembne razne konference, ki jih organizira mednarodna zveza za avtomatsko vodenje IFAC. Prav tako so številne konference organizirane pod okriljem mednarodne elektrotehniške zveze IEEE.

Poglavje 2

Vrste simulacije

Simulacijo kot metodologijo za analizo in načrtovanje sistemov delimo glede na vrste modelov na zvezno, diskretno in kombinirano simulacijo. V odvisnosti od orodja oz. tehnike, s katero se izvaja (vrsta računalnika), pa jo delimo na analogno, digitalno in hibridno simulacijo. Glede na to razdelitev je zvezna simulacija lahko analogna ali digitalna, diskretna je vedno digitalna, kombinirana simulacija pa je digitalna ali hibridna. Vrste simulacije prikazuje slika 2.1.



Slika 2.1: Vrste simulacije

Hitrost izvrševanja simulacije z ozirom na realni čas, v katerem deluje realni sistem, določa, kako model simuliramo:

- počasneje kot v realnem času,
- v realnem času,

- hitreje kot v realnem času.

Simulacija, ki ne poteka v realnem času, je najbolj običajna in se lahko izvaja na splošnonamenskih računalnikih. Ali se izvaja hitreje ali počasneje kot v realnem času, je odvisno od časovnih konstant realnega sistema ter od sposobnosti simulacijskega orodja, s katerim želimo dani sistem ponavadi čim hitreje simulirati. Simulacija v realnem času pa uvaja precej novih problemov, saj si pod tem pojmom predstavljamo običajno tudi priključitev realnega sistema na računalnik (HIL -hardware in the loop). Učinkovita simulacija v realnem času je možna le s specifično programsko in materialno opremo (moderne simulacijske delovne postaje, analogno-hibridni računalniki).

2.1 Zvezna simulacija

Zvezna simulacija omogoča simulirati sisteme, ki jih lahko opišemo z linearnimi ali nelinearnimi navadnimi (ODE) ali parcialnimi (PDE) diferencialnimi enačbami s konstantnimi ali spremenljivimi koeficienti. Pri tem pa je pogoj, da so spremenljivke stanj in njeni odvodi zvezni preko celotnega simulacijskega teka, v katerem je neodvisna spremenljivka običajno čas. To področje predstavlja najstarejšo pa tudi najnaravnejšo obliko simulacije, ki se je sprva izvajala na analognih, kasneje pa tudi na digitalnih računalnikih. Osnovni princip pri reševanju z zvezno simulacijo je uporaba integracije, s pomočjo katere iz višjih odvodov spremenljivk stanj v modelu izračunamo spremenljivke stanj, le-te pa z uporabo ustreznih operacij povežemo v paralelni simulacijski model. Običajno delimo zvezno simulacijo glede na vrsto uporabljenega računalnika na

- analogno simulacijo in
- digitalno simulacijo.

Sistemi, ki jih simuliramo z zvezno simulacijo, so zvezni in paralelni. Zato analogni način simulacije predstavlja najbolj naravno obliko. Pri digitalni simulaciji pa je potrebno integrator zamenjati z diskretnim numeričnim algoritmom, paralelno strukturo modela pa reševati zaporedno. Tak postopek zahteva več računalniškega časa, vendar je možno s primerno numerično integracijsko metodo in ustreznim vrstnim algoritmom problem zadovoljivo rešiti.

2.2 Diskretna simulacija

Pri diskretni simulaciji se stanja sistema spreminjajo v diskretnih trenutkih. Tem spremembam pravimo diskretni dogodki. Le-ti se lahko izvajajo periodično v natančno določenih trenutkih (npr. regulirni signal diskretnega regulatorja) ali pa nesinhrono v odvisnosti od pogojev, ki jih določajo vrednosti spremenljivk stanj. Prva oblika je bolj običajna v teoriji avtomatskega vodenja. Metodologija te vrste diskretne simulacije ima veliko skupnega z zvezno simulacijo. Drugi obliki, ki je bolj značilna za diskretno simulacijo, pa navadno pravimo simulacija diskretnih dogodkov. Lastnosti, ki jih bomo omenili v nadaljevanju, so značilne predvsem za to vrsto simulacije.

Simulacija diskretnih dogodkov omogoča določanje stanj sistema v odvisnosti od časa, zbiranje podatkov in ustrezno statistično analizo. Statistika je ena temeljnih operacij pri tej vrsti diskretne simulacije, kajti modeli, ki jih na ta način simuliramo, so običajno stohastične (naključne) narave.

Klasični primer, ki ga največkrat obravnava literatura (npr. Neelamkavil, 1987), je poštni urad z enim strežnikom in čakajočo vrsto. Stranke prihajajo naključno v sistem in so postrežene po sistemu FIFO (first in - first out). Bistveni parametri, ki jih proučujemo z diskretno simulacijo, so:

- povprečni časi med prihodoma zaporednih strank,
- povprečni časi streženja strank,
- povprečno število streženj na časovno enoto,
- izkoriščenost strežnika,
- povprečna dolžina vrste,
- povprečno število strank v vrsti,
- povprečno število strank v sistemu,
- povprečni časi čakanja strank,
- povprečni časi zadrževanja strank v sistemu.

V splošnem je možno tak problem rešiti z metodo opazovanja realnega sistema, s teoretično metodo ali z diskretno simulacijo.

Z metodo opazovanja realnega sistema v dovolj dolgem časovnem intervalu zgradimo tabelo, v kateri so naslednji podatki: indeks stranke, čas prihoda, čas med dvema zaporednima prihodoma, čas začetka strežbe, čas konca strežbe, čas strežbe, čas čakanja v vrsti, čas stranke v sistemu. Razen tega sestavimo tabele kumulativnih časov (čas, ko ni v vrsti nobene stranke, ko je v vrsti ena stranka, dve stranki, ..., n_{max} strank, čas, ko ni v sistemu nobene stranke, ko je ena stranka, dve stranki, ..., n_{max} strank). S pomočjo teh tabel lahko enostavno izračunamo karakteristične parametre diskretnega sistema. Če so kateri od parametrov socialno ali ekonomsko nesprejemljivi, se mora zgoraj navedeni urad preurediti.

Pri teoretični metodi uporabimo za izračun statistike analitične postopke. Le-ti so dobro razviti le za sistem z enim strežnikom in eno čakajočo vrsto in veljajo ob določenih predpostavkah:

- neomejena dolžina vrste,
- neskončni vir strank,
- opazovanje preko dolgega časovnega intervala in
- eksponencialni verjetnostni porazdelitvi za čas med dvema prihodoma in za čas streženja ($f(t) = \lambda e^{-\lambda t}$).

Z metodo opazovanja izračunamo parametre porazdelitve (λ), vendar moramo upravičenost eksponencialne porazdelitve statistično preveriti (npr. s pomočjo Chi square testa). Nato uporabimo analitične enačbe za izračun parametrov diskretnega sistema.

S simulacijo na digitalnem računalniku lahko problem rešujemo, če sta poznani eksponencialni verjetnostni porazdelitvi za čas med dvema prihodoma in za čas streženja. S pomočjo znanih porazdelitev računalnik generira naključna števila, ki predstavljajo čase med dvema prihodoma in čase streženja. Računalniški program torej simulira prihode strank v sistem in delovanje znotraj sistema, vsebovati pa mora tudi pogoj za končanje simulacijskega teka. Na tak način lahko obravnavamo kompleksne sisteme, ki jih analitično ni možno rešiti.

Nekateri avtorji ne ločijo med simulacijo diskretnih dogodkov in t.i. simulacijo po metodi Monte Carlo, čeprav je med njima precejšnja razlika. Ustrezno poimenovanje je izvedel Von Neumann v 60. letih prejšnjega stoletja v okviru vojaškega projekta v Los Alamosu. V obeh primerih je simulacijski model vzbujan z naključnimi signali, vendar je sistem, ki ga modeliramo, v primeru simulacije

diskretnih dogodkov stohastičen, v primeru simulacije po metodi Monte Carlo pa ima deterministični značaj. S simulacijo po metodi Monte Carlo lahko na primer izračunamo vrednost določenega integrala $\int_0^1 f(x)dx$, t.j. povsem determinističnega problema tako, da generiramo naključne vrednosti odvisne in neodvisne spremenljivke x in $f(x)$ z ustrezno statistiko (npr. uniformna naključna signala na znanem področju) in pri dovolj velikem številu vzorcev izračunamo vrednost integrala. Simulacija po metodi Monte Carlo se uporablja v primerih slabo razvitih numeričnih metod ali pa, če le-te sploh ne obstajajo. Primer za računanje določenega integrala ne prvi pogled sicer nima neposredne zveze s simulacijo dinamičnih sistemov. Vendar smo že omenili in bomo kasneje to še globje spoznali, da je integracija osnovna operacija simulacije zveznih dinamičnih sistemov. Le-ta je pri digitalni simulaciji izvedena z numerično metodo. Lahko pa bi namesto neke uveljavljene numerične metode za integracijo uporabili tudi metodo Monte Carlo. Tak način lepo ilustrira koncept simulacije po metodi Monte Carlo, vendar se ne uporablja, ker numerična integracija ne spada med slabo razvite numerične postopke.

2.3 Kombinirana ali hibridna simulacija

Cellier (Cellier, 1979) je označil kombinirano ali hibridno simulacijo kot simulacijo sistemov, ki jih lahko opišemo na celotnem intervalu opazovanja ali na delu tega intervala z diferencialnimi enačbami, pri čemer pa vsaj ena spremenljivka stanj ali njen odvod ni zvezna veličina. Po tej definiciji vidimo, da je praktično vsaka simulacija realnega problema v bistvu kombinirana simulacija. Vendar je tako stroga definicija kombinirane simulacije smiselna, ker smo na isti način definirali tudi zvezno simulacijo. To pa ne pomeni, da orodja za zvezno simulacijo sistemov niso primerna za obravnavo realnih sistemov. Ta orodja imajo običajno tudi sicer zelo omejene možnosti kombinirane simulacije. Niso pa primerna za simulacijo sistemov, kjer so diskretni dogodki ali nezveznosti zelo pogoste. Imajo namreč slabo razvite mehanizme za prehod iz zveznega v diskretni del modela in obratno. Medtem ko orodja za kombinirano simulacijo vedno numerično pravilno obdelujejo nezveznosti, imajo pri zveznih digitalnih simulacijskih orodjih to lastnost samo redki obstoječi simulacijski jeziki. Šele v zadnjem času posvečajo proizvajalci tej problematiki večjo pozornost.

2.4 Simulacija v realnem času

Simulacija v realnem času predstavlja posebno zahtevno obliko simulacije, ko je neodvisna spremenljivka sinhronizirana z realnim časom. Ponavadi je čas opazovanja (simulacije) neomejen (precej dolg), tako da redkeje govorimo o simulacijskih tekih oz. o njihovih dolžinah.

Prav področje vodenja sistemov je izredno vplivalo na razvoj materialne in programske opreme za simulacijo v realnem času. Omejene zmožnosti preizkušanja sistemov vodenja ter vadbe operaterjev na realnih industrijskih sistemih zahtevajo uporabo kompleksnih simulatorjev za delo v realnem času. Na ta način je možno v realnem času preizkušati tudi postopke, ki lahko proces pripeljejo v napačno delovanje ali celo v katastrofo.

Razen sinhronizacije z realnim časom je pri tovrstni simulaciji običajno značilno tudi:

- zajemanje za računalniški sistem primerno pripravljenih podatkov realnih signalov,
- posredovanje rezultatov v obliki realnih signalov in
- neomejena dolžina simulacijskega teka.

Večina sodobnih konvencionalnih računalnikov ni sposobnih učinkovito izvajati simulacije v realnem času zaradi (Lincoln, 1988):

- preskromnih sposobnosti materialne opreme,
- zaradi preveč kompleksne in za potrebe simulacije v realnem času redundantne programske opreme,
- zaradi premalo učinkovitih možnosti za programiranje,
- zaradi preskromnih vhodno-izhodnih zmožnosti.

S stališča uporabe simulacije v realnem času ločimo dve glavni področji:

- razvoj in preizkušanje novih sistemov,

- izobraževanje in vadba.

V zvezi z razvojem in preizkušanjem novih sistemov predstavlja simulacija inženirsko načrtovalno orodje. Tovrstna simulacija lahko zahteva velike procesne zmožnosti. Kot primer naj omenimo simulacijo leta letala, ki se lahko izredno uspešno uporablja pri razvoju in preizkušanju sistema za avtomatsko vodenje leta. Upoštevati mora natančen in kompleksen model z visokofrekvenčnimi in nizkofrekvenčnimi komponentami. Zaradi hitrih pojavov je potreben majhen računski korak (frame time cca. $1ms$). Na drugi strani pa tak sistem ne potrebuje posebnih vhodno-izhodnih zmožnosti, saj se lahko učinkovitost sistema vodenja preveri ob spremljanju manjšega števila signalov.

Simulatorji za izobraževanje in vadbo pa morajo zagotoviti čimbolj realno okolje. Ker ima človek omejene možnosti reagiranja, zajemajo modeli samo ustrezno frekvenčno področje. Zato take simulacije niso časovno kritične (tipični računski korak simulatorjev za vadbo pilotov je v razredu desetinke sekunde) in ne zahtevajo ekstremnih procesnih sposobnosti. Zato pa taki simulatorji zahtevajo zelo sposobne vhodno-izhodne zmožnosti za realno predstavitev problema (npr. prikaz vseh instrumentov na zaslonu v primeru simulatorja za vadbo pilota).

Simulacija v realnem času zahteva običajno velike procesne zmožnosti računalnikov. Le-te so povezane s:

- potrebnim računskim korakom (odvisnost od časovnih konstant sistema),
- kompleksnostjo simuliranega objekta (odvisnost od kompleksnosti realnega objekta, zahtevane natančnosti, ...) in
- potrebo po vhodno-izhodnih operacijah (odvisnost od namena simulatorja).

Zato si v povezavi s simulacijo v realnem času v glavnem predstavljamo uporabo namenskih digitalnih računalnikov, seveda pa tudi analogno-hibridnih računalnikov.

V zadnjem času se simulacija v realnem času vse več uporablja na področju izobraževanja. V laboratorijih se vaje na realnih napravah zamenjujejo z ustrezno simuliranimi napravami, ki delujejo v realnem času. Realne pilotne naprave so zaradi omejenih zmožnosti praviloma preveč enostranske, da bi omogočale učinkovit in kreativen pedagoški proces. S simulacijo v realnem času je možno pripraviti več problemov, tako da lahko vsak študent rešuje svoj problem. Pri

vajah na pilotni napravi morajo praktično vsi študenti delati isto. Tak pristop k laboratorijskim vajam predstavlja najboljšo uskladitev med realnostjo, kompleksnostjo, stroški in varnostjo.

Poglavje 3

Osnovne metode pri reševanju problemov s simulacijo

V tem delu bomo pokazali osnovne metode pri reševanju problemov s simulacijo. *Indirektni postopek* pri reševanju diferencialnih enačb prikazuje bistvo zvezne simulacije. Razen tega postopka bomo opisali direktno in implicitno metodo ter več načinov simulacije prenosnih funkcij. Prikazali bomo tudi metodo za simulacijo sistemov z mrtvim časom in način simulacije kompleksnih sistemov. Postopki nas bodo pripeljali do simulacijskih shem, ki so neodvisne od uporabljenega simulacijskega orodja in predstavljajo osnovo pri uporabi simulacijskega orodja. Končno bomo razčlenili tudi koncept digitalne simulacije.

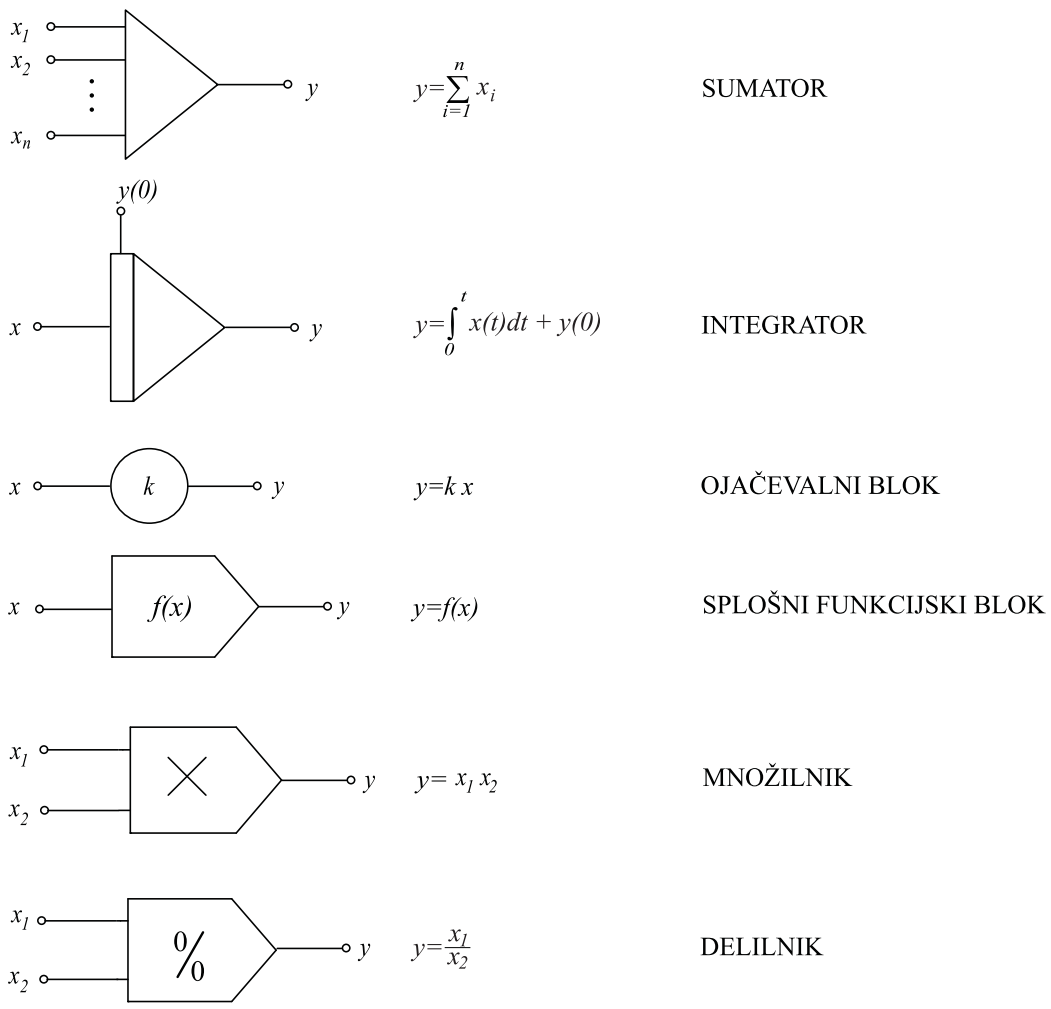
3.1 Simulacijska shema

Osnova za zvezno simulacijo nekega modela je posebna grafična predstavitev, ki jo bomo imenovali *simulacijska shema*. Le-ta ima veliko skupnega z bločnimi diagrami, ki jih zlasti uporabljamo pri zapisu sistemov vodenja. Zato bomo tudi osnovne gradnike simulacijskih shem imenovali *bloke*. Blok predstavlja funkcijo določenega gradnika simulacijske sheme. Grafično predstavitev bloka pa imenujemo *ikona*.

Ker osnove simulacije izvirajo iz konceptov analogne simulacije, izhajajo tudi simulacijske sheme iz analognih simulacijskih shem. Lord Kelvin je imenoval

ustrezno shemo "shemo diferencialne analize". Danes se v praksi uporabljajo zelo različne oblike simulacijskih shem tako, da praktično vsak izdelovalec simulacijskega orodja uvede kakšne svoje bloke oz. ikone. S čim manjšim naborom blokov oz. ikon bomo skušali uvesti shemo, ki bo uporabna za kakršno koli simulacijsko orodje. Tako bodo take sheme uporabne za simulacijo z digitalnim simulacijskim jezikom kot tudi za simulacijo z analognim računalnikom, kjer pa je potrebno dodatno upoštevati, da elementi analognega računalnika obračajo signalom predznak.

Osnovne bloke, ki jih bomo uporabljali v simulacijski shemi, prikazuje slika 3.1. Po potrebi pa bomo kasneje definirali še kakšen nov blok oz. ikono.



Slika 3.1: Pogosto uporabljeni bloki oz. ikone v simulacijski shemi

Predznake, ki učinkujejo v posameznih blokih, lahko vpeljemo preko ustreznih parametrov (npr. predznak konstante k pri ojačevalnem bloku, ki signal x pomnoži s konstanto k), lahko pa ga definiramo v bližini, kjer vhodni signal vstopa v blok (če predznaka ni, privzamemo pozitivni predznak). Sumator ima poljubno število vhodov, integrator pa le enega. Splošni funkcijski blok je najsplošnejši. Predstavlja lahko vir signalov (takrat običajno ne rišemo vhodnega priključka) ali pa poljubne nelinearne zakonitosti med vhodom in izhodom. V konkretnih shemah namesto $f(x)$ vpišemo v blok ustrezno matematično relacijo (npr. SIN za funkcijo sinus) ali pa vrišemo grafični simbol, ki nazorno pove, za kakšen tip bloka gre. Vlogo bloka lahko nadalje posplošimo, če definiramo vhod in izhod kot vektorska signala ($\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$).

3.2 Indirektna metoda

Ker so matematični modeli dinamičnih sistemov običajno opisani s sistemom diferencialnih enačb, predstavlja indirektna metoda za reševanje diferencialnih enačb osnovni simulacijski pristop. Po tej metodi je potrebno najvišji odvod integrirati tolikokrat, kolikor je njegov red. S tem indirektno generiramo vse nižje odvode in samo spremenljivko. V tem načinu se skriva bistvo simulacije. Analitična rešitev diferencialne enačbe in tabeliranje rešitve v določenih točkah neodvisne spremenljivke nima nobene zveze s simulacijo. Včasih sicer na ta način lahko pridemo hitreje do bolj točnih rezultatov, vendar je v praksi uporabnost takega analitičnega pristopa zelo omejena (npr. le za linearne sisteme).

Indirektna metoda je uporabna, če je možno iz diferencialne enačbe izraziti najvišji odvod in če ne nastopajo višji odvodi vhodnega signala. Indirektno metodo opišimo za sistem

$$y^{(n)} + f(y^{(n-1)}, y^{(n-2)}, \ldots, y', y, u; t) = 0 \tag{3.1}$$

y je izhodni signal, u je vhodni signal, t pa je neodvisna spremenljivka simulacije (čas). Postopek opišemo v treh točkah:

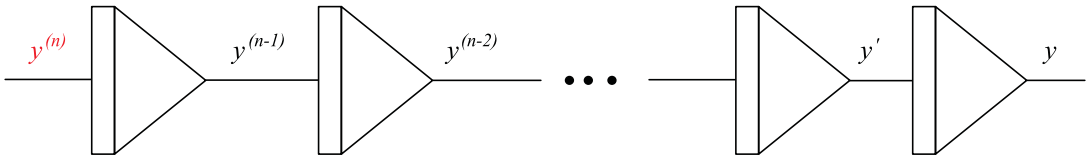
1. korak

Preuredimo diferencialno enačbo tako, da ostane na levi strani najvišji odvod, vse ostalo pa prenesemo na desno stran. Če je sistem zapisan v prostoru stanj (s sistemom diferencialnih enačb 1. reda), je zapis že ustrezen in prvi korak odpade

$$y^{(n)} = -f(y^{(n-1)}, y^{(n-2)}, \ldots, y', y, u; t) \tag{3.2}$$

2. korak

Narišemo kaskado n integratorjev, če je n red najvišjega odvoda. 2. korak prikazuje slika 3.2.

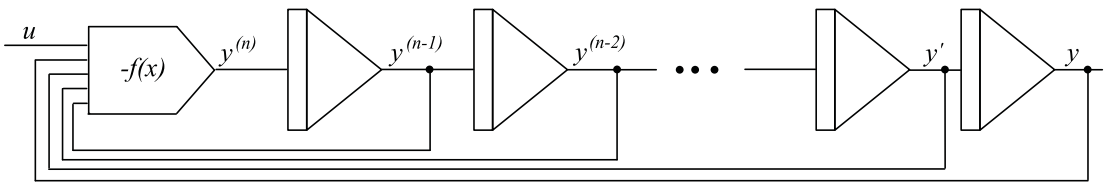


Slika 3.2: 2. korak indirektnega postopka

Predpostavimo, da je vhod v prvi integrator (najvišji odvod) znan, preostali integratorji pa generirajo nižje odvode in samo spremenljivko oz. rešitev diferencialne enačbe.

3. korak

Z upoštevanjem (virtualnih) nižjih odvodov in rešitve diferencialne enačbe generiramo negativno funkcijsko odvisnost, ki realizira desno stran enačbe (3.2). Izhod bloka, ki generira negativno funkcijsko odvisnost, je enak najvišjemu odvodu, zato ga moramo povezati na vhod prvega integratorja. Pri generaciji funkcije -f uporabljamo različne bloke (razen integratorja), kar zavisi od oblike diferencialne enačbe. Dobljeni obliki, ki jo prikazuje slika 3.3, pravimo tudi kanonična oblika.



Slika 3.3: 3. korak indirektnega postopka

Postopek, ki smo ga opisali, je neposredno uporaben, če diferencialna enačba ne vsebuje odvodov vhodnega signala. Če pa le-ti nastopajo, je v primeru li-

nearnih sistemov bolj smiselno sistem simulirati po konceptu prenosnih funkcij (podpoglavje 3.5).

Uporabnost postopka bomo prikazali na primerih.

Primer 3.1 Temperaturni proces

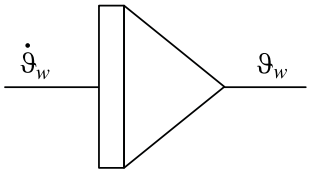
Model temperaturnega procesa je opisan v primeru 1.2. Matematični model opisuje diferencialna enačba

$$\dot{\vartheta}_w + \frac{1}{T}\vartheta_w = \frac{k}{T}p \tag{3.3}$$

Z upoštevanjem 1. koraka indirektnega postopka moramo enačbo (3.3) preurediti v obliko

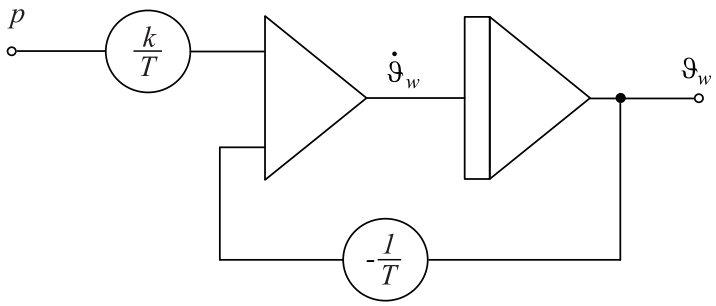
$$\dot{\vartheta}_w = -\frac{1}{T}\vartheta_w + \frac{k}{T}p \tag{3.4}$$

V drugem koraku narišemo le en integrator, ker je diferencialna enačba 1. reda. Ustrezen korak prikazuje slika 3.4.



Slika 3.4: Simulacijska shema za 2. korak

V 3. koraku generiramo desno stran enačbe (3.4). Končno simulacijsko shemo prikazuje slika 3.5.



Slika 3.5: Simulacijska shema temperaturnega procesa

Kot lahko vidimo na sliki 3.5, smo desno stran enačbe (3.4) realizirali s sumatorjem in dvema ojačevalnima blokoma. Potrebna predznaka sta vključena v ojačevalnih blokih. □

Primer 3.2 Avtomobilsko vzmetenje

Model avtomobilskega vzmetenja smo predstavili v primeru 1.1. Matematični model opisuje diferencialna enačba

$$\dddot{y}_1 + \frac{k_1 + k_2}{f}\ddot{y}_1 + \frac{k_2}{M}\dot{y}_1 + \frac{k_1 k_2}{Mf}y_1 = 0 \qquad y_1(0) = -y_{10} \tag{3.5}$$

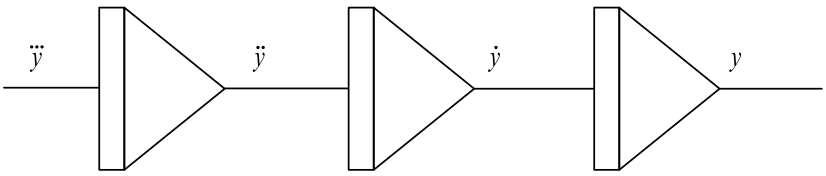
ki jo lahko poenostavimo v obliko

$$\dddot{y} + a\ddot{y} + b\dot{y} + cy = 0 \qquad y(0) = -d \tag{3.6}$$

1. korak: Preuredimo enačbo (3.6)

$$\dddot{y} = -a\ddot{y} - b\dot{y} - cy \qquad y(0) = -d \tag{3.7}$$

2. korak: Narišemo kaskado treh integratorjev, kar prikazuje slika 3.6

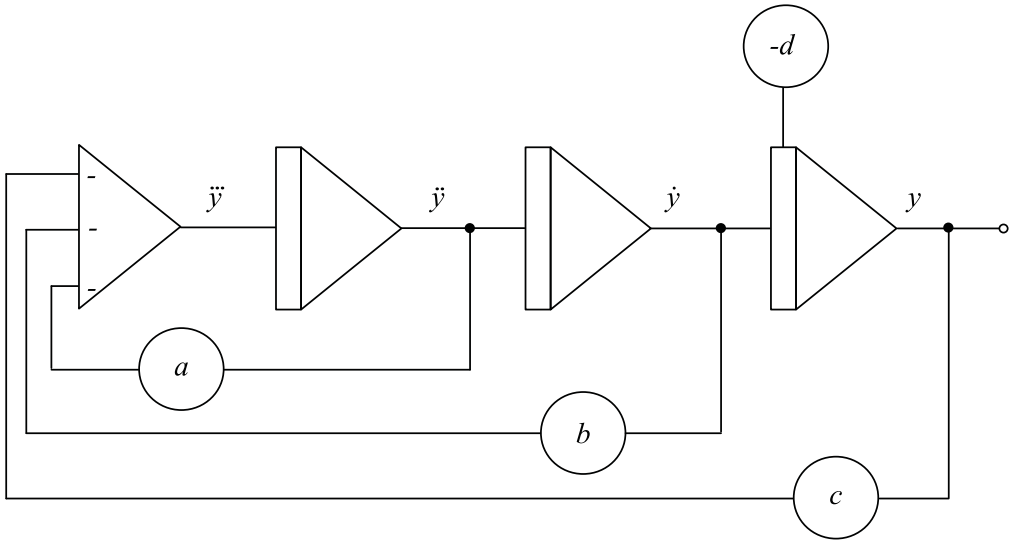


Slika 3.6: Simulacijska shema za 2. korak

3. korak: Zaključimo simulacijsko shemo, kot prikazuje slika 3.7. V tem primeru smo nekatere predznake podali v sumacijskem bloku.

□

Primer 3.3 Ekološki sistem žrtev in roparjev

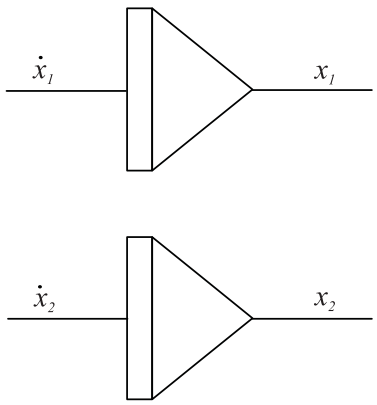


Slika 3.7: Simulacijska shema sistema avtomobilskega vzmetenja

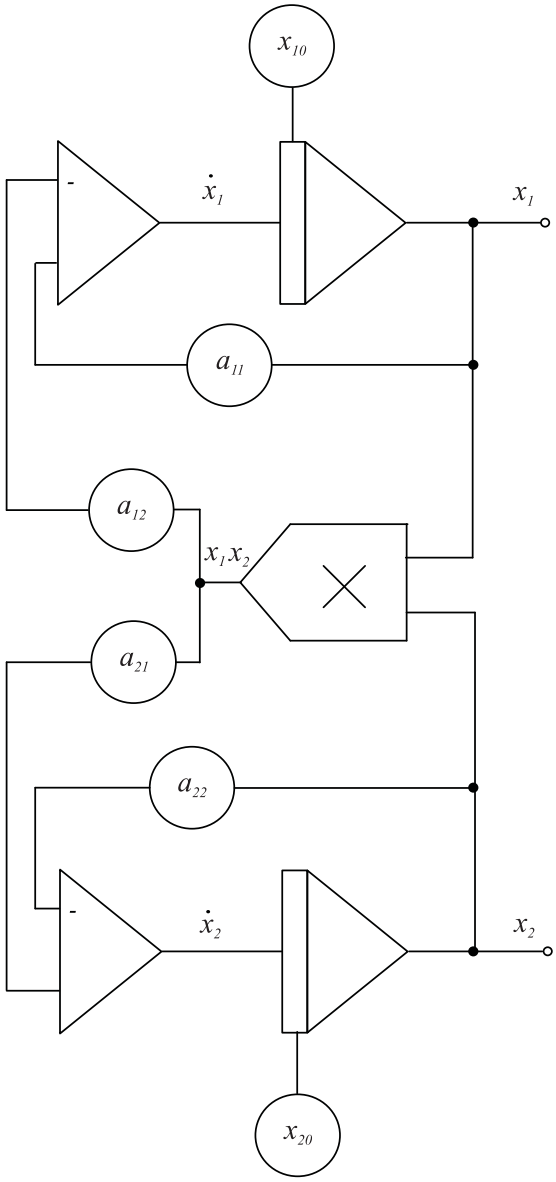
Model ekološkega sistema, v katerem nastopajo roparji in žrtve, smo predstavili v primeru 1.3. Primer prikazuje uporabnost indirektne metode pri reševanju sistema nelinearnih diferencialnih enačb. Matematični model ima obliko

$$\begin{aligned} \dot{x}_1 &= a_{11}x_1 - a_{12}x_1x_2 \qquad x_1(0) = x_{10} \\ \dot{x}_2 &= a_{21}x_1x_2 - a_{22}x_2 \qquad x_2(0) = x_{20} \end{aligned} \tag{3.8}$$

1. korak: Ni potreben, ker imata enačbi že predpisano obliko.
2. korak: Za vsako enačbo narišemo en integrator, kar prikazuje slika 3.8.
3. korak: Zaključimo simulacijsko shemo, kot to prikazuje slika 3.9.



Slika 3.8: Simulacijska shema za 2. korak



Slika 3.9: Simulacijska shema ekološkega sistema

Kot vidimo iz slik 3.8 in 3.9, je postopek simulacije nelinearnega sistema enako enostaven kot pri linearnih sistemih. Razen zank okoli posameznih integratorjev dobimo v tem primeru tudi ustrezne križne povezave. Nelinearnost vnaša v shemo množilnik. □

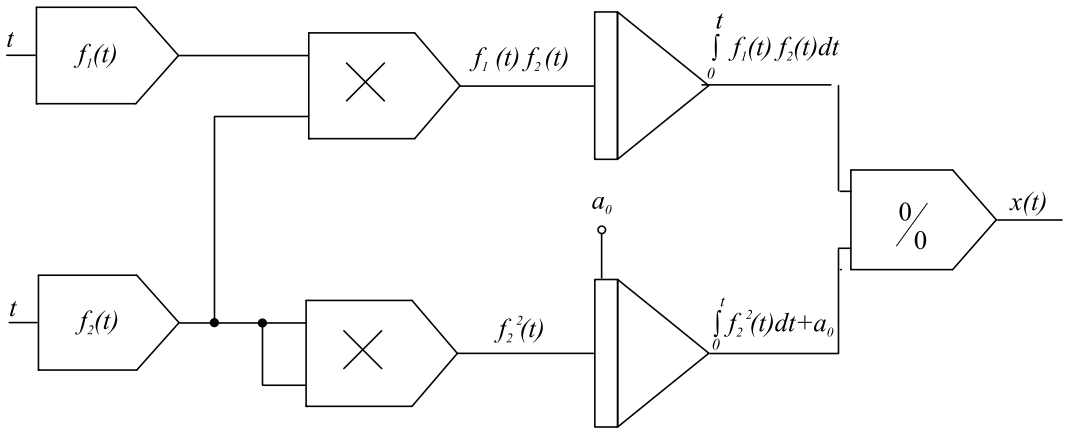
3.3 Direktna metoda

S pomočjo direktne metode realiziramo izraze v obliki formul, ki so ponavadi algebraične narave. Tako realiziramo zlasti razne izraze, ki jih potrebujemo v indirektnem načinu. Tudi v tem primeru je smiselno uporabiti simulacijsko shemo.

Primer 3.4 Potrebno je izračunati izraz

$$x(t) = \frac{\int_0^t f_1(t) f_2(t) dt}{\int_0^t f_2^2(t) dt + a_0} \tag{3.9}$$

Simulacijsko shemo prikazuje slika 3.10.



Slika 3.10: Simulacijska shema pri uporabi direktne metode

□

3.4 Implicitna metoda

S pomočjo implicitne metode uporabljamo simulacijo za generiranje funkcij. Poiskati poizkušamo neko diferencialno enačbo, katere rešitev je analitična funkcija, ki jo želimo generirati. Zaradi enostavnosti indirektnega postopka, ki ga uporabljamo pri reševanju diferencialne enačbe, je tak postopek upravičen.

Pri implicitni metodi izraz oz. funkcijo, ki jo želimo generirati, enkrat ali večkrat odvajamo na neodvisno spremenljivko. Po vsakem odvajanju moramo izraziti čim več členov s funkcijo ali njenimi odvodi. S postopkom oz. odvajanjem končamo, ko dobimo obliko, v kateri nastopa le funkcija in njeni odvodi. Seveda vedno ne moremo priti do take oblike. Na koncu je potrebno določiti tudi začetne pogoje.

Primer 3.5 Najenostavnejši problem predstavlja implicitna generacija eksponentne funkcije

$$y = e^{-at} \tag{3.10}$$

Z enkratnim odvajanjem dobimo enačbo

$$\dot{y} = -ae^{-at} \tag{3.11}$$

in če člen e^{-at} zamenjamo z y (enačba 3.10), dobimo diferencialno enačbo

$$\dot{y} = -ay \tag{3.12}$$

Začetni pogoj izračunamo s pomočjo enačbe (3.10) za $t = 0$

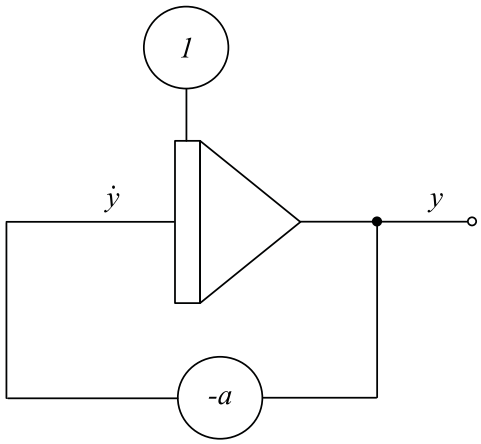
$$y(0) = 1 \tag{3.13}$$

Enačbi (3.12) in (3.13) rešimo (simuliramo) z indirektno metodo. Shemo prikazuje slika 3.11.

□

3.5 Simulacija prenosnih funkcij

Prenosne funkcije imajo pomembno vlogo pri analizi in načrtovanju sistemov. Zlasti je učinkovita nazornost bločnih diagramov, v katerih nastopajo med ostalimi funkcionalnimi bloki tudi prenosne funkcije.



Slika 3.11: Implicitna generacija funkcije $y = e^{-at}$

Od številnih metod za simulacijo prenosnih funkcij bomo prikazali dve, ki sta najbolj nazorni in se največ uporabljata. To sta *vgnezdena* in *delitvena metoda*. Razen teh dveh osnovnih metod pa bomo obravnavali še dve razčlenitveni metodi. Le-ti sta zlasti primerni pri simulaciji bolj kompleksnih prenosnih funkcij.

3.5.1 Vgnezdena metoda

Postopek bomo prikazali na primeru 3.6.

Primer 3.6 Prenosno funkcijo

$$G(s) = \frac{Y(s)}{U(s)} = \frac{as^3 + bs^2 + cs + d}{s^3 + es^2 + fs + g} \tag{3.14}$$

simuliramo z vgnezdeno metodo s pomočjo naslednjih korakov:

- Enačbo (3.14) preuredimo v obliko

$$(s^3 + es^2 + fs + g)Y = (as^3 + bs^2 + cs + d)U \tag{3.15}$$

- Če koeficient pri členu z najvišjo potenco v imenovalcu ni enak 1, je potrebno vse člene polinomov v števcu in imenovalcu deliti s tem koeficientom.
- Združimo vse člene, ki imajo enako potenco spremenljivke s

$$s^3(Y - aU) + s^2(eY - bU) + s(fY - cU) + (gY - dU) = 0 \tag{3.16}$$

- Preuredimo enačbo (3.16) tako, da imamo na levi najvišji "odvod"(potenco operatorja s) izhodne veličine

$$s^3Y = s^3aU - s^2(eY - bU) - s(fY - cU) - (gY - dU) \tag{3.17}$$

- Delimo enačbo (3.17) z najvišjo potenco spremenljivke s

$$Y = aU - \frac{1}{s}(eY - bU) - \frac{1}{s^2}(fY - cU) - \frac{1}{s^3}(gY - dU) \tag{3.18}$$

- Preuredimo enačbo (3.18) v vgnezdeno obliko

$$Y = aU + \frac{1}{s}\{(bU - eY) + \frac{1}{s}[(cU - fY) + \frac{1}{s}(dU - gY)]\} \tag{3.19}$$

- Ob vpeljavi pomožnih spremenljivk

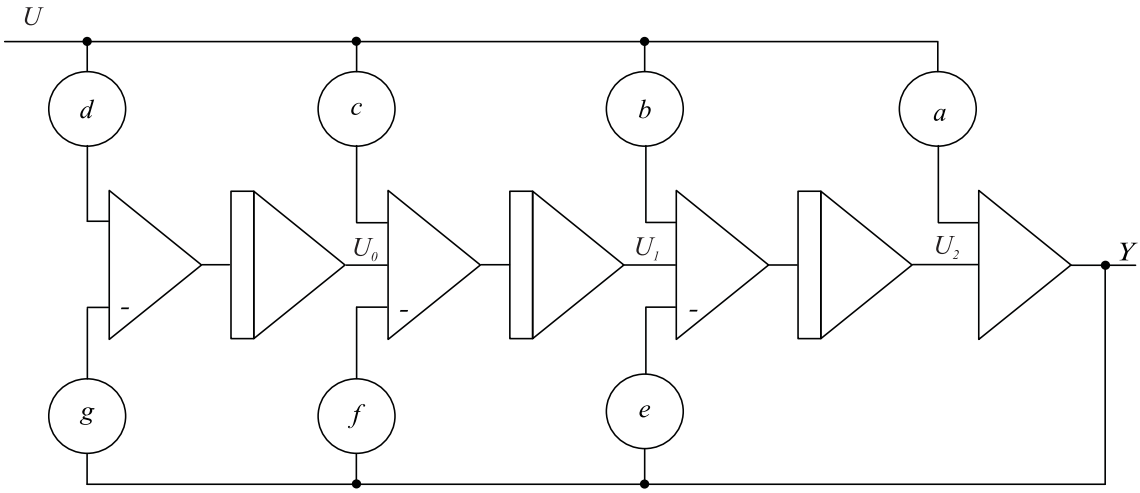
$$\begin{aligned} U_0 &= \frac{1}{s}(dU - gY) \\ U_1 &= \frac{1}{s}(cU - fY + U_0) \\ U_2 &= \frac{1}{s}(bU - eY + U_1) \end{aligned} \tag{3.20}$$

dobi enačba (3.19) obliko

$$Y = aU + U_2 \tag{3.21}$$

- Z upoštevanjem enačb (3.20) in (3.21) dobimo simulacijsko shemo, ki jo prikazuje slika 3.12.

Če koeficient pri členu s^3 ne bi bil 1, bi bile vrednosti ojačevalnih blokov v sliki 3.12 kvocienti koeficientov a, b, c, d, e, f in g s tem koeficientom.



Slika 3.12: Simulacijska shema ob uporabi vgnezdene oblike

Shema na sliki 3.12 predstavlja realizacijo prenosne funkcije, ki je v teoriji vodenja znana kot *spoznavnostna kanonična oblika* (Ogata, 2010). Koeficienti prenosne funkcije nastopajo kot ojačevalni bloki. Vpeljava pomožnih spremenljivk poenostavi postopek risanja simulacijske sheme kot tudi programiranje pri uporabi enačbno orientiranih simulacijskih jezikov. □

3.5.2 Delitvena metoda

Postopek bomo prikazali na primeru 3.7.

Primer 3.7 Prenosno funkcijo

$$G(s) = \frac{Y(s)}{U(s)} = \frac{as^3 + bs^2 + cs + d}{s^3 + es^2 + fs + g} \tag{3.22}$$

simuliramo z delitveno metodo s pomočjo naslednjih korakov:

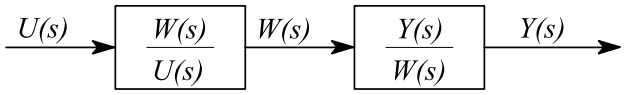
- Če koeficient pri členu z najvišjo potenco v imenovalcu ni enak 1, je potrebno vse člene polinomov v števcu in imenovalcu deliti s tem koeficientom.
- Prenosno funkcijo $\frac{Y(s)}{U(s)}$ razdelimo s pomočjo pomožne spremenljivke $W(s)$ v dve prenosni funkciji; prva predstavlja realizacijo imenovalca

$$\frac{W(s)}{U(s)} = \frac{1}{s^3 + es^2 + fs + g} \tag{3.23}$$

druga pa realizacijo števca

$$\frac{Y(s)}{W(s)} = as^3 + bs^2 + cs + d \tag{3.24}$$

Delitveni postopek nazorno prikazuje slika 3.13.



Slika 3.13: Bločna shema, ki predstavlja delitveni postopek

- Preuredimo enačbo (3.23) v obliko

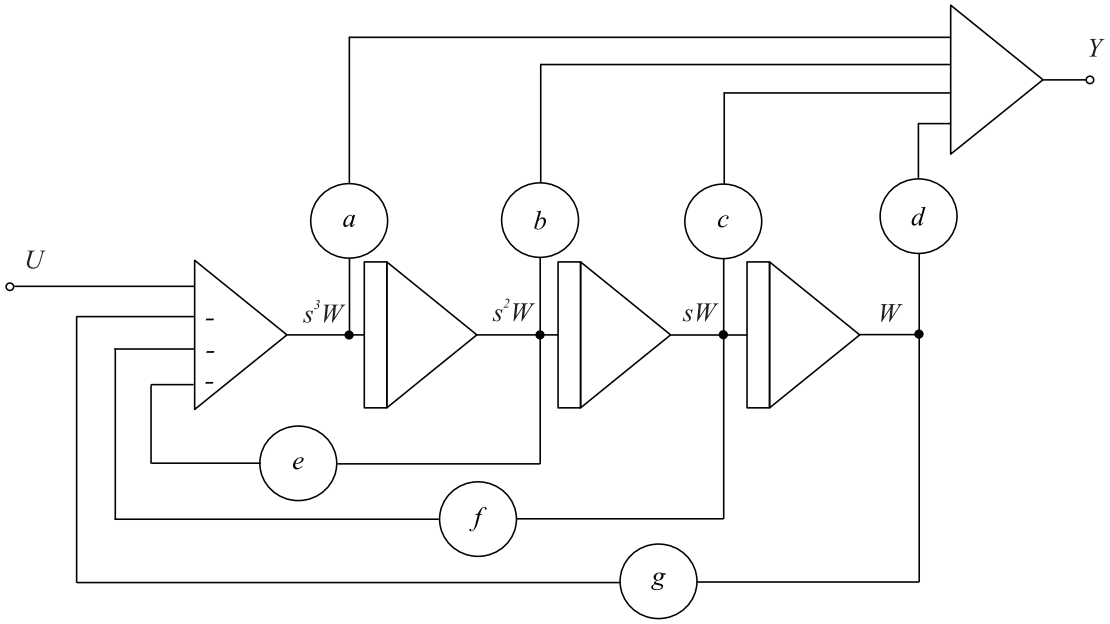
$$s^3W = U - es^2W - fsW - gW \tag{3.25}$$

- Preuredimo enačbo (3.24) v obliko

$$Y = as^3W + bs^2W + csW + dW \tag{3.26}$$

- S pomočjo enačb (3.25) in (3.26) realiziramo simulacijsko shemo, ki jo prikazuje slika 3.14.

Če koeficient pri členu s^3 ne bil 1, bi bile vrednosti ojačevalnih blokov v sliki 3.14 kvocienti koeficientov a, b, c, d, e, f in g s tem koeficientom. Kot vidimo iz slike 3.14, sestoji struktura iz dveh delov. Eden realizira imenovalec z indirektno metodo (le ta namreč omogoča, da so razen izhodne spremenljivke dostopni tudi odvodi), drugi del pa generira števec s pomočjo direktne metode. Metoda je zelo



Slika 3.14: Simulacijska shema ob uporabi delitvene metode

primerna za simulacijo več prenosnih funkcij z enakim imenovalcem, saj le enkrat simuliramo prenosno funkcijo $\frac{W}{U}$.

Shema na sliki 3.14 predstavlja realizacijo prenosne funkcije, ki je v teoriji vodenja znana kot *vodljivostna kanonična oblika* (Ogata, 2010). □

Obe metodi potrebujeta toliko integratorjev, kolikor je red prenosne funkcije. Uporabljamo jih tudi pri simulaciji sistemov, ki so opisani z diferencialnimi enačbami, pri čemer nastopajo tudi višji odvodi vhodnega signala (npr. $\dddot{y} + e\ddot{y} + f\dot{y} + gy = a\ddot{u} + b\ddot{u} + c\dot{u} + du$).

3.5.3 Vzporedna razčlenitev

Po tej metodi razčlenimo prenosno funkcijo v vsoto več prenosnih funkcij. Postopek lahko opišemo z naslednjimi koraki:

- Poiščemo realne pole prenosne funkcije.

- Po metodi nedoločenih koeficientov razčlenimo prenosno funkcijo na vsoto delnih ulomkov. Imenovalci delnih ulomkov so določeni z realnimi poli, v primeru konjugirano kompleksnih polov pa z ustreznimi kvadratnimi členi. Če je stopnja števca enaka stopnji imenovalca, delimo polinom v števcu s polinomom v imenovalcu. S tem dobimo v vzporedni razčlenitvi tudi konstantni člen.
- Narišemo simulacijsko shemo vsakega člena. Vhode vseh členov povežemo skupaj, izhode vseh členov pa seštejemo na sumatorju.

Primer 3.8 Simulirajmo prenosno funkcijo

$$G(s) = \frac{Y(s)}{U(s)} = \frac{4s + 10}{s^2 + 6s + 8} \tag{3.27}$$

Če izračunamo pole, lahko enačbo (3.27) zapišemo v obliki

$$G(s) = \frac{4(s + 2.5)}{(s + 4)(s + 2)} \tag{3.28}$$

Z razčlenitvijo na delne ulomke dobimo iz enačbe (3.28) izraz

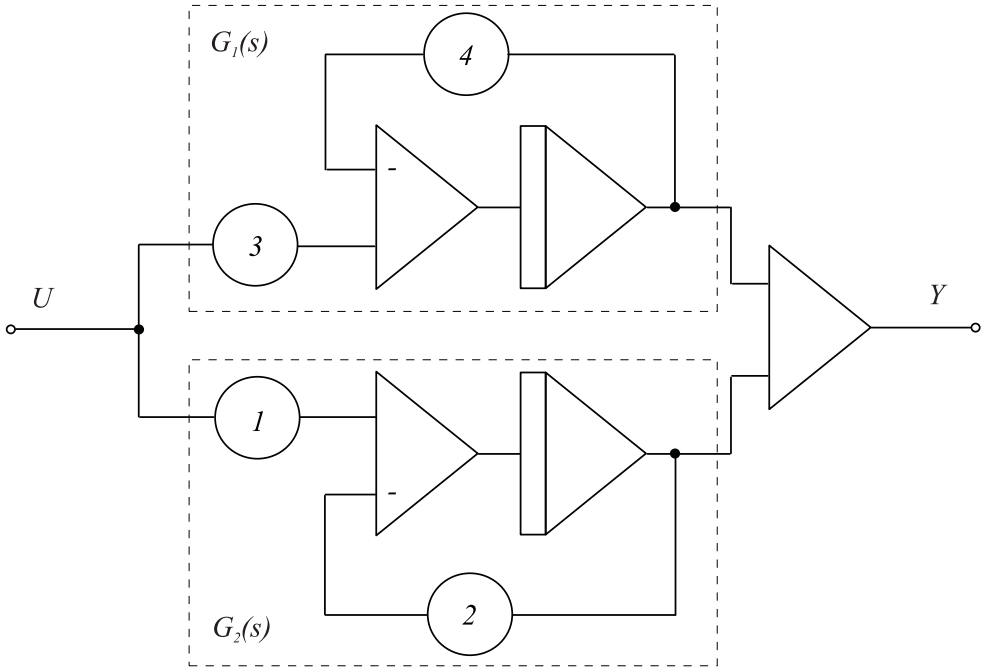
$$G(s) = G_1(s) + G_2(s) = \frac{3}{s + 4} + \frac{1}{s + 2} \tag{3.29}$$

Gre torej za enak postopek razčlenitve, kot pri računanju inverzne Laplace-ove transformacije. Ustrezno simulacijsko shemo, kjer smo $G_1(s)$ in $G_2(s)$ realizirali z vgnezdeno metodo, prikazuje slika 3.15.

□

3.5.4 Zaporedna razčlenitev

Po tej metodi poiščemo pole in ničle prenosne funkcije in zapišemo ustrezno faktorizirano obliko. V primeru konjugirano kompleksnih korenov ohranimo



Slika 3.15: Realizacija z metodo vzporedne razčlenitve

kvadratne člene. Nato iz prenosne funkcije generiramo produkt več prenosnih funkcij, pri čemer ena prenosna funkcija vsebuje elementarne gradnike kot npr.: ojačenje, en pol, kvocient ničle in pola, kvadratni člen v imenovalcu, kvocient kvadratnih členov, ... Te osnovne gradnike zaporedno povežemo v simulacijsko shemo.

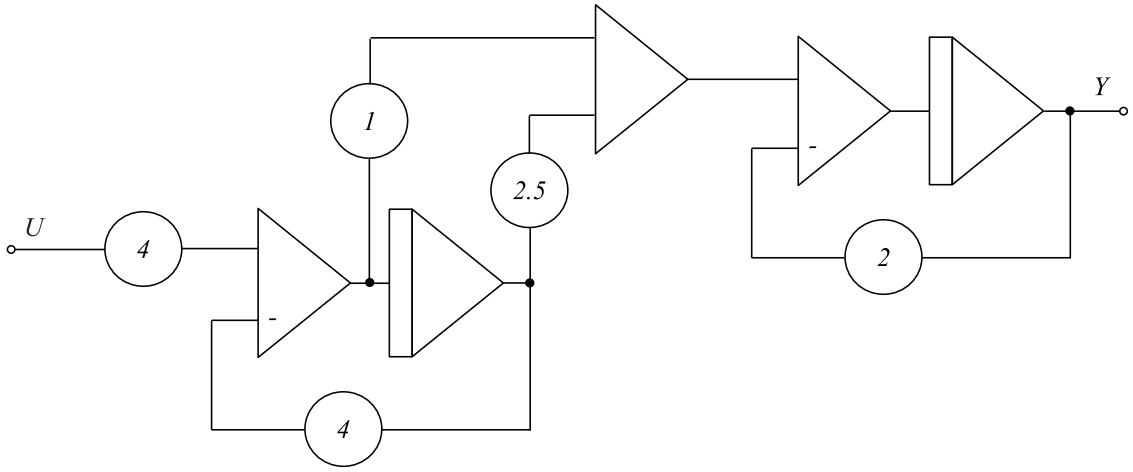
Primer 3.9 Simulirajmo enako funkcijo, kot v primeru 3.8

$$G(s) = \frac{Y(s)}{U(s)} = \frac{4s + 10}{s^2 + 6s + 8} \tag{3.30}$$

Po izračunu polov in ničel lahko enačbo (3.30) zapišemo v obliki

$$\begin{aligned} G(s) &= 4G_1(s)G_2(s) \\ G_1(s) &= \frac{s + 2.5}{s + 4} \\ G_2(s) &= \frac{1}{s + 2} \end{aligned} \tag{3.31}$$

$G_1(s)$ in $G_2(s)$ realizirajmo po delitveni metodi. Po ustrezni zaporedni povezavi dobimo simulacijsko shemo, ki jo prikazuje slika 3.16.



Slika 3.16: Realizacija z metodo zaporedne razčlenitve

□

3.6 Simulacija sistemov z mrtvim časom

Sistem, ki vsebuje le mrtvi čas, opisuje prenosna funkcija

$$\frac{Y(s)}{U(s)} = \frac{\mathcal{L}[y(t)]}{\mathcal{L}[u(t)]} = \frac{\mathcal{L}[u(t-T)]}{\mathcal{L}[u(t)]} = G(s) = e^{-sT} \tag{3.32}$$

kjer je T mrtvi čas, $y(t)$ je izhod, $u(t)$ pa vhod v sistem. Mrtvi čas vnaša tako na digitalnih kot na analognih računalnikih v simulacijskem postopku precej problemov, ki jih je mogoče rešiti le z aproksimacijskimi postopki. Medtem, ko postopki na digitalnih računalnikih temeljijo na čimbolj pogostem "vzorčenju"vhodnega signala $u(t)$ in na zakasnjevanju s pomočjo pomnjenja vzorcev v obsežnih poljih in ustreznem premikanju elementov v teh poljih, pa postopki, ki so bili razviti za analogne računalnike, upoštevajo metode razvrstitve transcedentne funkcije e^{-sT} v vrsto. Najbolj so znane t.i. Padé-jeve aproksimacije. Največ se uporabljajo aproksimacije 1., 2. in 4. reda.

Aproksimacija 1. reda

$$e^{-sT} \doteq \frac{1 - \frac{T}{2}s}{1 + \frac{T}{2}s} = \frac{-sT + 2}{sT + 2} \tag{3.33}$$

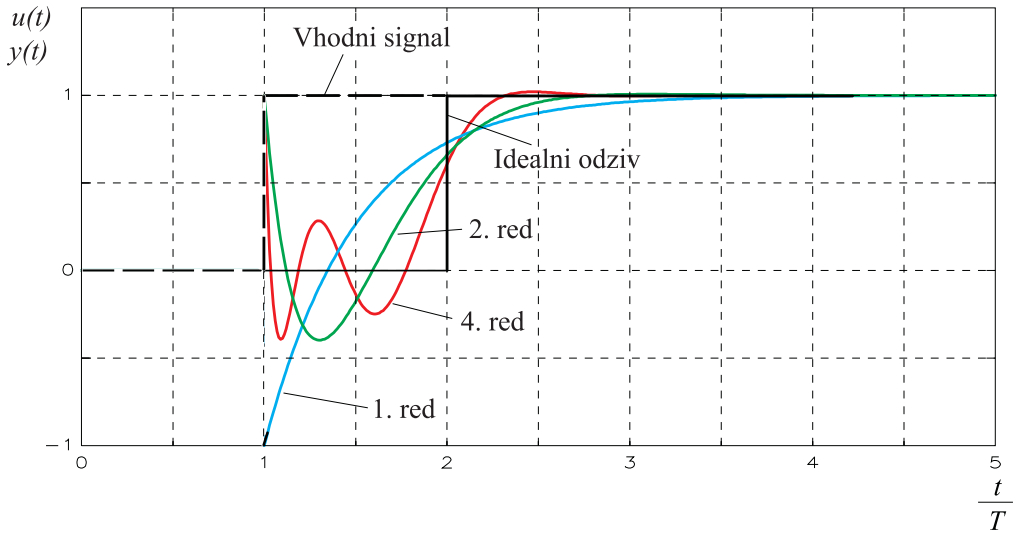
Aproksimacija 2. reda

$$e^{-sT} \doteq \frac{(sT)^2 - 6sT + 12}{(sT)^2 + 6sT + 12} \tag{3.34}$$

Aproksimacija 4. reda

$$e^{-sT} \doteq \frac{(sT)^4 - 20(sT)^3 + 180(sT)^2 - 840sT + 1680}{(sT)^4 + 20(sT)^3 + 180(sT)^2 + 840sT + 1680} \tag{3.35}$$

Čim višji je red, boljšo aproksimacijo dobimo. Vse prenosne funkcije imajo lastnost sistemov z neminimalno fazo. Slika 3.17 prikazuje idealni odziv sistema z mrtvim časom in odzive Padé-jevih aproksimacij 1., 2. in 4. reda. Vhodni signal je enotina stopnica in nastopi v trenutku $\frac{t}{T} = 1$.



Slika 3.17: Odzivi sistema z mrtvim časom pri uporabi Padé-jevih aproksimacij 1., 2. in 4. reda

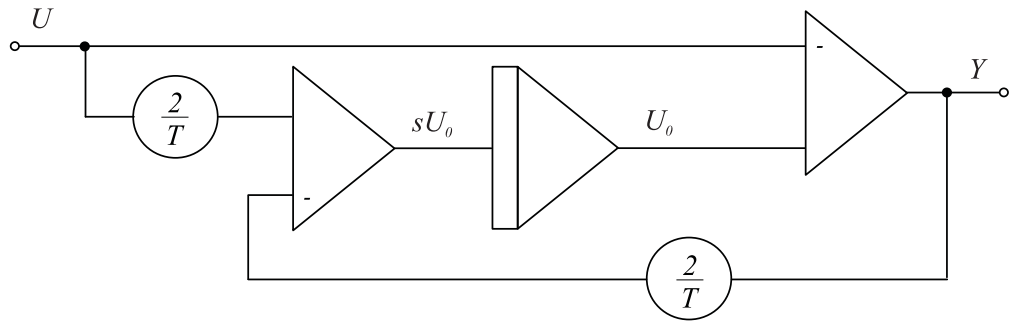
Primer 3.10 Mrtvi čas e^{-sT} aproksimirajmo s Padé-jevo aproksimacijo 1. reda

$$G(s) = \frac{Y(s)}{U(s)} = \frac{1 - \frac{T}{2}s}{1 + \frac{T}{2}s} \tag{3.36}$$

Prenosno funkcijo 3.36 simulirajmo s pomočjo vgnezdene metode

$$\begin{aligned} Y(1 + \frac{T}{2}s) &= U(1 - \frac{T}{2}s) \\ s\frac{T}{2}Y &= -s\frac{T}{2}U + U - Y \\ Y &= -U + \frac{1}{s}(\frac{2}{T}U - \frac{2}{T}Y) \\ U_0 &= \frac{1}{s}(\frac{2}{T}U - \frac{2}{T}Y) \\ Y &= -U + U_0 \end{aligned} \tag{3.37}$$

Simulacijsko shemo prikazuje slika 3.18, rezultate simulacije pri stopničastem vhodnem signalu pa ena od krivulj na sliki 3.17.



Slika 3.18: Simulacijska shema sistema, ki aproksimira mrtvi čas po Padé-jevi metodi 1. reda

□

3.7 Simulacija kompleksnih sistemov

Pri simulaciji kompleksnih sistemov je treba že v fazi modeliranja upoštevati principe modularnosti, kar pomeni, da večje sklope realiziramo z manjšimi zaključenimi sklopi (podmodeli, prenosne funkcije, ...). Medtem ko je pri analitičnih metodah analize in načrtovanja potrebno take podsklope združevati oz. poenostavljati (npr. poenostavljanje bločnih diagramov s pomočjo algebre bločnih diagramov), pa je pomembna prednost pri simulaciji v tem, da vsak podsklop ločeno simuliramo po eni izmed obravnavanih metod in nato posamezne podsklope ustrezno povežemo. Na ta način so dosegljive tudi vse spremenljivke podsklopov, simulacijske sheme so pregledne in uspešno služijo tudi za dokumentacijo.

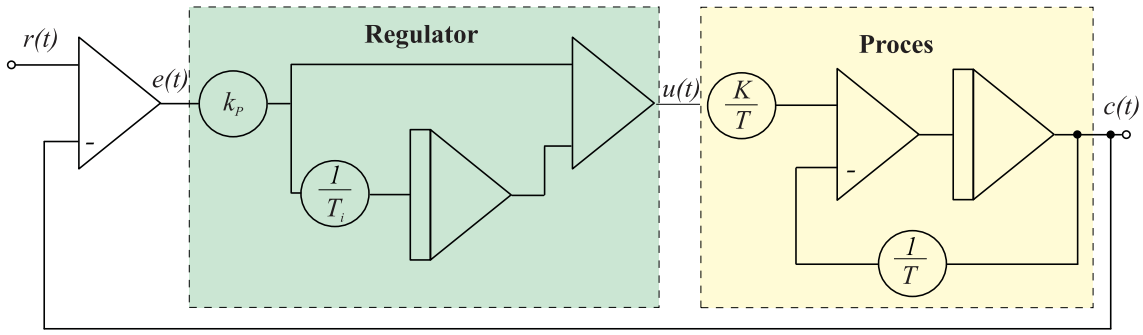
Primer 3.11 Regulacijski sistem vsebuje regulator PI

$$G_R(s) = \frac{U(s)}{E(s)} = K_P(1 + \frac{1}{T_I s}) \tag{3.38}$$

in proces 1. reda

$$G_P(s) = \frac{C(s)}{U(s)} = \frac{K}{Ts + 1} \tag{3.39}$$

Ob upoštevanju nazornosti in modularnosti pri simulaciji kompleksnejših sistemov dobimo simulacijsko shemo, ki jo prikazuje slika 3.19. Slika pregledno kaže regulacijsko strukturo. Če bi npr. simulirali regulacijski sistem s pomočjo ene zaprtozančne prenosne funkcije $\frac{C(s)}{R(s)}$, ne bi bila dostopna regulirna veličina $u(t)$.



Slika 3.19: Simulacijska shema regulacijskega sistema

□

3.8 Koncept digitalne simulacije

Metode, ki smo jih spoznali v prejšnjih podpoglavjih, omogočajo programiranje problema na analognem računalniku (ob predhodnem normiranju in skaliranju). Prav tako te metode omogočajo, da s pomočjo simulacijske sheme direktno napišemo program v simulacijskem jeziku. Če pa želimo sami programirati simulacijo v nekem splošnonamenskem programskem jeziku, (npr. Visual Basic, C++, Fortran, Java, Matlab ...) je potrebno poznati koncept digitalne simulacije zveznih dinamičnih sistemov, kajti integrator je potrebno realizirati z numeričnim postopkom, paralelni sistem pa računati zaporedno.

Da lahko simuliramo zvezni sistem na digitalnem računalniku, moramo neodvisno spremenljivko simulacije (običajno čas) diskretizirati, tako da diferencialne enačbe postanejo diferenčne enačbe. Ker so vse spremenljivke modela odvisne od neodvisne spremenljivke t, so torej tudi definirane samo v diskretnih vrednostih (trenutkih) neodvisne spremenljivke. Če simulacija poteka s konstantnim prirastkom neodvisne spremenljivke Δt (*računski korak*), lahko trenutno vrednost neodvisne spremenljivke izrazimo kot

$$t_i = t_0 + i\Delta t \qquad\qquad i = 0, 1, 2, \ldots, i_{max} \tag{3.40}$$

Torej simulacija prične v trenutku $t = t_0$ (*začetni čas simulacije*) in se konča v trenutku $t = t_{max} = t_0 + i_{max}\Delta t$ (*končni čas simulacije*). Dogajanju med časoma t_0 in t_{max} pa pravimo *simulacijski tek*.

Vsi sistemi (orodja) za digitalno simulacijo so osnovani na zapisu sistema v prostoru stanj, t.j. s sistemom diferencialnih enačb 1. reda. Uporabniku pa se tega običajno niti ni potrebno zavedati, saj se ustrezni zapis avtomatično vzpostavi. Izhodišče je torej vektorska diferencialna enačba

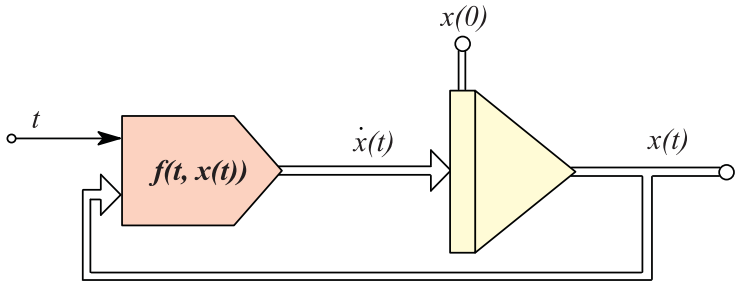
$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(t, \boldsymbol{x}(t)) \tag{3.41}$$

Vsa stanja je potrebno hraniti v vektorju $\boldsymbol{x}(t)$, vse odvode pa v vektorju $\dot{\boldsymbol{x}}(t)$.

Pri uporabi indirektnega načina je torej prvi korak že opravljen. Drugi korak pa se realizira z enim vektorskim integratorjem

$$\boldsymbol{x}(t) = \int \dot{\boldsymbol{x}}(t)dt = \int \boldsymbol{f}(t, \boldsymbol{x}(t))dt \tag{3.42}$$

Z upoštevanjem tretjega koraka indirektnega postopka dobimo simulacijsko shemo, ki jo prikazuje slika 3.20.



Slika 3.20: Osnovna simulacijska shema

Da lahko shemo na sliki 3.20 simuliramo na digitalnem računalniku, mora imeti vsak simulacijski sistem tri pomembne značilnosti:

- Ker splošnonamenski sekvenčni digitalni računalniki ne zmorejo paralelnega računanja kot npr. analogni računalniki, *je potrebno vse zanke, ki jih prikazuje slika* 3.20 *na nek način prekiniti, tako da je možno spremenljivke računati zaporedno.* Zanke vedno prekinemo na izhodih t.i. spominskih blokov (včasih jim pravimo tudi bloki z zakasnitvenim atributom), katerih izhodi so običajno stanja sistema. V splošnem je to lahko vsak blok, ki ima lastnost, da trenutna vrednost njegovega izhoda ni odvisna od vrednosti vhoda v istem trenutku (npr. integrator, blok z mrtvim časom, diskretna zakasnitev, zadrževalnik ničtega reda (ZOH), zakasnilni blok, prenosna funkcija, ki ima red števca manjši od reda imenovalca,...). Najpogosteje pa so taki bloki integratorji (glej enačbo (3.43) v primeru Euler-jevega algoritma).

- *Vrstni algoritem* mora razvrstiti algebrajske enačbe (bloke) za izračun odvodov ($\boldsymbol{f}(t, \boldsymbol{x}(t))$, enačba (3.41)) tako, da je možno pravilno izračunavanje, t.j. da se vsi vhodi v nek blok izračunajo prej, preden se izračunavajo enačbe (oz. izhodi) tega bloka.

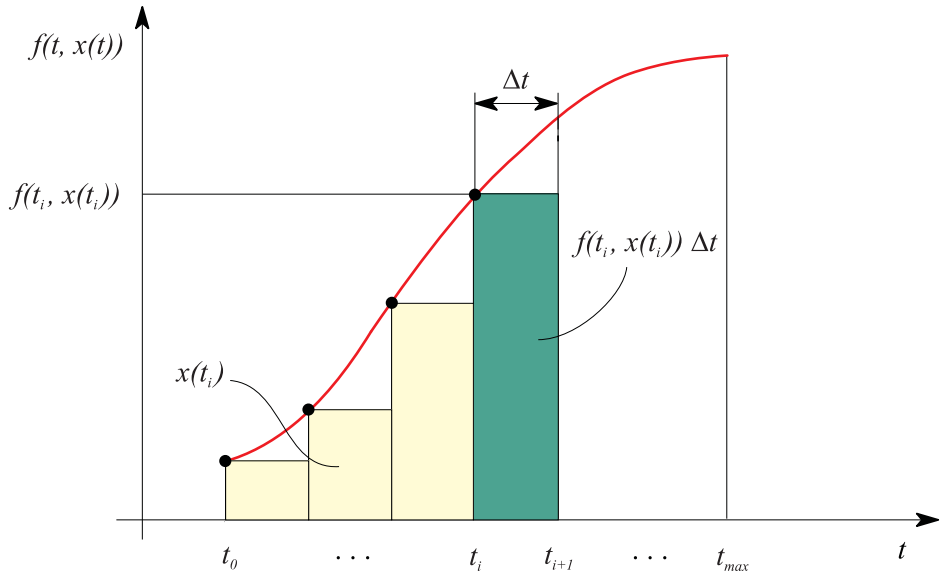
- *Enačbo* (3.42) *je potrebno rešiti z numeričnim integracijskim postopkom.*

  Integracija je osrednja operacija vsakega simulacijskega orodja. Če ne potrebujemo velike natančnosti, jo je možno realizirati z zelo enostavnim numeričnim postopkom. V sodobnih numerično izpopolnjenih simulacijskih

jezikih pa ti postopki omogočajo veliko natančnost in numerično zanesljivost in so zato zelo kompleksni. Princip numerične integracije pa najlepše prikazuje najenostavnejši postopek, t.j. Euler-jev algoritem, ki opisuje reševanje enačbe (3.42) v obliki

$$\boldsymbol{x}(t_{i+1}) = \boldsymbol{x}(t_i + \Delta t) = \boldsymbol{x}(t_i) + \boldsymbol{f}(t_i, \boldsymbol{x}(t_i))\Delta t \tag{3.43}$$

Metoda predpostavlja stopničasto aproksimacijo funkcije $\boldsymbol{f}(t, \boldsymbol{x}(t))$, ustrezen integral pa je vsota pravokotnikov pod krivuljo $\boldsymbol{f}(t, \boldsymbol{x}(t))$. Postopek prikazuje slika 3.21.

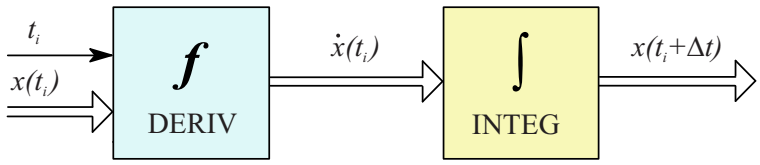


Slika 3.21: Integracija z Euler-jevo metodo

Integracijska metoda torej uporablja trenutne vrednosti stanj $\boldsymbol{x}(t_i)$ in odvodov $\boldsymbol{f}(t_i, \boldsymbol{x}(t_i))$ za ovrednotenje stanj za naslednji računski korak $\boldsymbol{x}(t_i + \Delta t)$.

Če zanke sistema, ki ga prikazuje slika 3.20, prekinemo na izhodu vektorskega integratorja in če zamenjamo zvezno integracijo z numeričnim postopkom, dobimo shemo, ki jo prikazuje slika 3.22. Ta shema jasno prikazuje koncept digitalne simulacije zveznih dinamičnih sistemov.

Na začetku simulacijskega teka se vedno izvede inicializacija, ki vključuje tudi ovrednotenje začetnih vrednosti stanj ($\boldsymbol{x}(t_0)$).

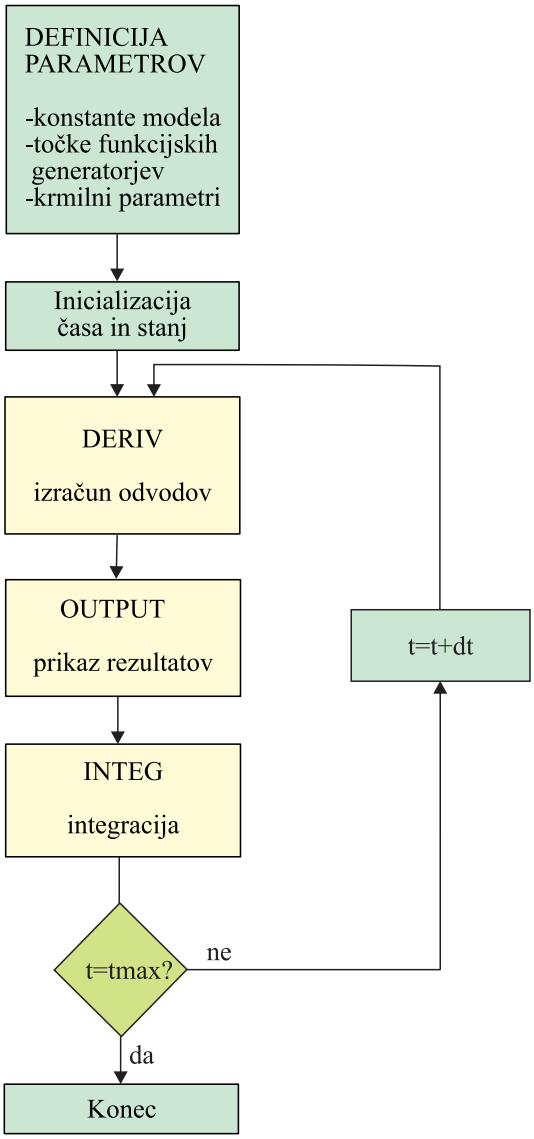


Slika 3.22: Osnovni koncept digitalne simulacije

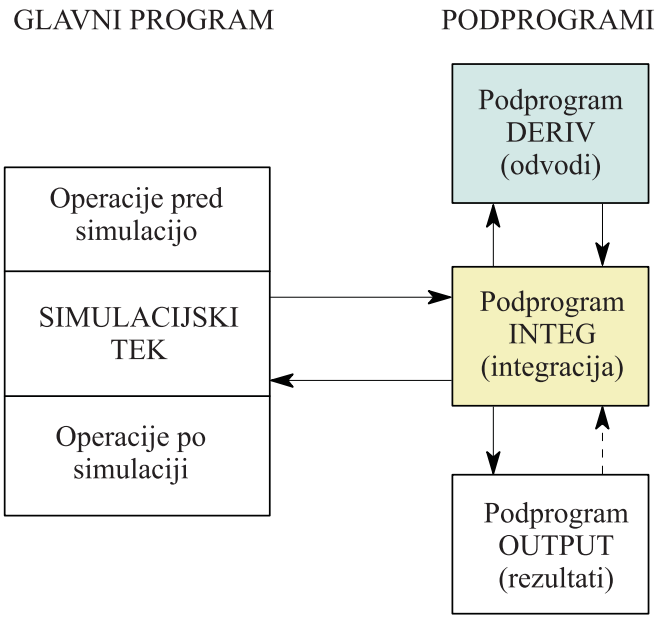
Simulacijski algoritem lahko opišemo z *dvokoračno iteracijo* v vsakem računskem koraku. V prvem koraku se izračunajo vsi odvodi spremenljivk stanja ($\dot{\boldsymbol{x}}(t_i)$) iz trenutnih vrednosti spremenljivk stanja ($\boldsymbol{x}(t_i)$) in iz vrednosti neodvisne spremenljivke simulacije (t_i). Odvodi se izračunajo v podprogramu, ki ga bomo imenovali DERIV in vključuje izračun funkcije $\boldsymbol{f}(\boldsymbol{x}, t)$ oz. ustrezno razvrščene enačbe modela. V drugem koraku se v podprogramu, ki ga bomo imenovali INTEG, izvrši integracijski postopek, ki izračuna stanja za naslednji računski korak ($\boldsymbol{x}(t_i + \Delta t)$).

Tako poenostavljena dvokoračna iteracija velja samo pri uporabi Euler-jeve integracijske metode. Programsko realizacijo take dvokoračne iteracije prikazuje slika 3.23.

Pri uporabi numerično bolj izpopolnjenih postopkov potrebuje integracijska metoda na enem računskem koraku več ovrednotenj odvodov, torej je potrebnih več klicev podprograma DERIV. Če ustrezno razširimo osnovno strukturo programa, dobimo običajno izvedbo (v orodjih, simulacijskih paketih), kot prikazuje slika 3.24. Pred simulacijskim tekom se izvedejo določene operacije (npr. inicializacija stanj). Simulacijski tek pa se izvrši s klicem podprograma za integracijo (INTEG), ki po potrebi kliče podprogram za izračun odvodov spremenljivk stanja (DERIV). V trenutkih, ki jih definira uporabnik, pa integracijski podprogram kliče podprogram (OUTPUT), ki skrbi za ustrezno posredovanje rezultatov simulacije. Po izpolnjenem pogoju za končanje simulacijskega teka se izvršijo še morebitne operacije po koncu simulacijskega teka.



Slika 3.23: Diagram poteka simulacijskega programa pri Eulerjevi integracijski metodi

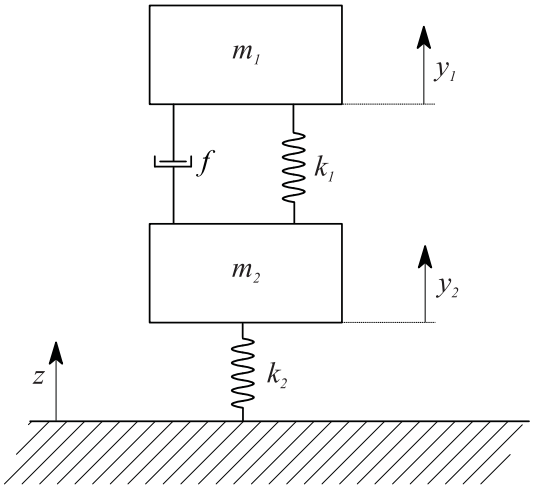


Slika 3.24: Običajna struktura simulacijskega programa

3.9 Rešene naloge

Naloga 3.1 Spremenjen model avtomobilskega vzmetenja

Kot prvi primer vzemimo nekoliko spremenjen model avtomobilskega vzmetenja glede na primer 1.2. V tem primeru ne zanemarimo mase kolesa in polovice osi. Sistem vzbujamo z oviro na podlagi (npr. če kolo zapelje na pločnik). Ustrezen mehanski model prikazuje slika 3.25.



Slika 3.25: Mehanski model avtomobilskega vzmetenja

Z y_1 smo označili gibanje karoserije, z y_2 gibanje kolesa, z z pa oviro, ki je v našem primeru stopničasta sprememba, pri čemer velikost stopnice določa višina pločnika. Matematični model opišemo z enačbama

$$\begin{aligned} m_1\ddot{y}_1 &= -f(\dot{y}_1 - \dot{y}_2) - k_1(y_1 - y_2) \\ m_2\ddot{y}_2 &= -f(\dot{y}_2 - \dot{y}_1) - k_1(y_2 - y_1) - k_2(y_2 - z) \end{aligned} \tag{3.44}$$

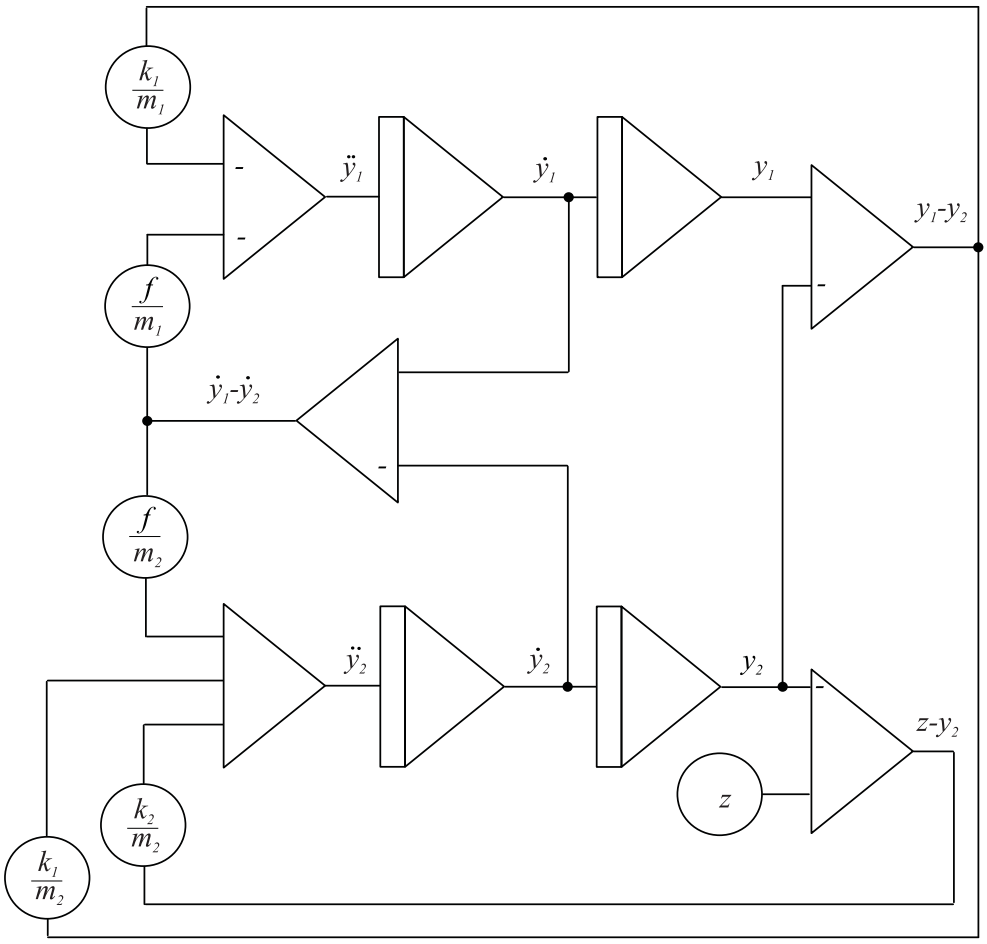
Z uporabo indirektne metode simuliramo model v naslednjih korakih:

1. korak: preureditev enačb

$$\ddot{y}_1 = -\frac{f}{m_1}(\dot{y}_1 - \dot{y}_2) - \frac{k_1}{m_1}(y_1 - y_2) \tag{3.45}$$

$$\ddot{y}_2 = \frac{f}{m_2}(\dot{y}_1 - \dot{y}_2) + \frac{k_1}{m_2}(y_1 - y_2) + \frac{k_2}{m_2}(z - y_2)$$

2. in 3. korak: generacija sheme, ki jo prikazuje slika 3.26.



Slika 3.26: Simulacijska shema modela avtomobilskega vzmetenja

Naloga 3.2 Uporaba implicitne metode

V tem primeru bomo na implicitni način realizirali signal

$$y = At^2e^{-bt} \tag{3.46}$$

Z odvajanjem enačbe (3.46) dobimo

$$\dot{y} = 2Ate^{-bt} - At^2be^{-bt} = 2Ate^{-bt} - by \tag{3.47}$$

Drugi člen v enačbi (3.47) smo dobili z upoštevanjem enačbe (3.46).

Enačbo (3.47) ponovno odvajamo

$$\ddot{y} = 2Ae^{-bt} - 2Abte^{-bt} - b\dot{y} \tag{3.48}$$

Drugi člen na desni strani enačbe (3.48) izrazimo s pomočjo enačbe (3.47)

$$2Ate^{-bt} = \dot{y} + by \tag{3.49}$$

tako da dobimo izraz

$$\ddot{y} = 2Ae^{-bt} - b\dot{y} - b^2y - b\dot{y} \tag{3.50}$$

Na tem mestu bi lahko končali s postopkom, saj bi lahko člen $2Ae^{-bt}$ realizirali z implicitnim postopkom, celotno enačbo (3.50) pa z indirektnim načinom. Če pa nadaljujemo z odvajanjem

$$\dddot{y} = -2Abe^{-bt} - 2b\ddot{y} - b^2\dot{y} \tag{3.51}$$

in nadomestimo prvi člen na desni strani enačbe (3.51) iz enačbe (3.50)

$$2Ae^{-bt} = \ddot{y} + 2b\dot{y} + b^2y \tag{3.52}$$

dobimo končno obliko

$$\dddot{y} = -3b\ddot{y} - 3b^2\dot{y} - b^3y \tag{3.53}$$

z začetnimi pogoji

$$y(0) = 0 \qquad \dot{y}(0) = 0 \qquad \ddot{y}(0) = 2A \tag{3.54}$$

Enačbo (3.53) simuliramo z indirektno metodo.

Naloga 3.3 Mathieu-jeva diferencialna enačba

V tem primeru bomo pokazali, da je indirektna metoda primerna tudi za reševanje diferencialnih enačb s spremenljivimi koeficienti. Te koeficiente, ki so običajno funkcije neodvisne spremenljivke, je potrebno dodatno generirati.

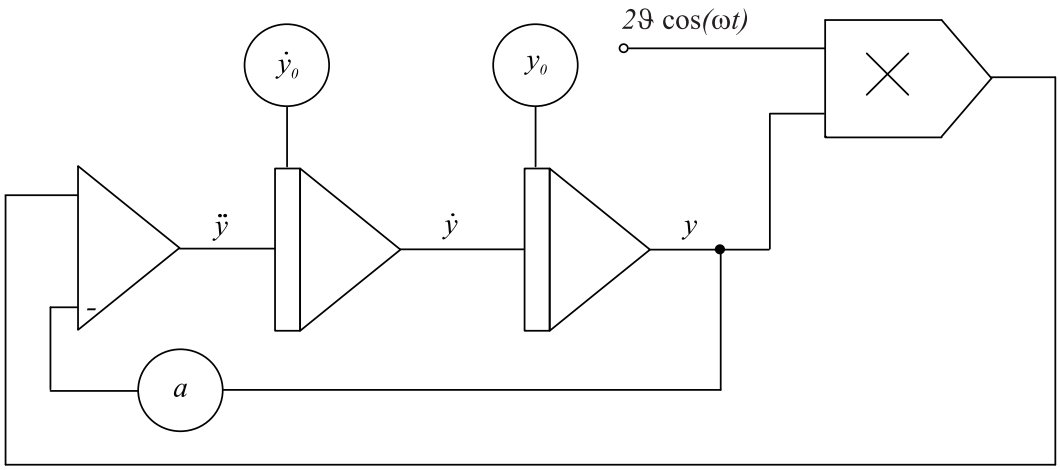
Mathieu-jeva enačba ima obliko

$$\ddot{y} + [a - 2\vartheta\cos(\omega t)]y = 0 \qquad y(0) = y_0 \qquad \dot{y}(0) = \dot{y}_0 \tag{3.55}$$

in se uporablja pri študiju frekvenčno moduliranih nihanj. Enačbo (3.55) preuredimo za indirektno metodo

$$\ddot{y} = -ay + 2\vartheta\cos(\omega t)y \qquad y(0) = y_0 \qquad \dot{y}(0) = \dot{y}_0 \tag{3.56}$$

Simulacijsko shemo prikazuje slika 3.27.



Slika 3.27: Simulacijska shema za Mathieu-jevo diferencialno enačbo

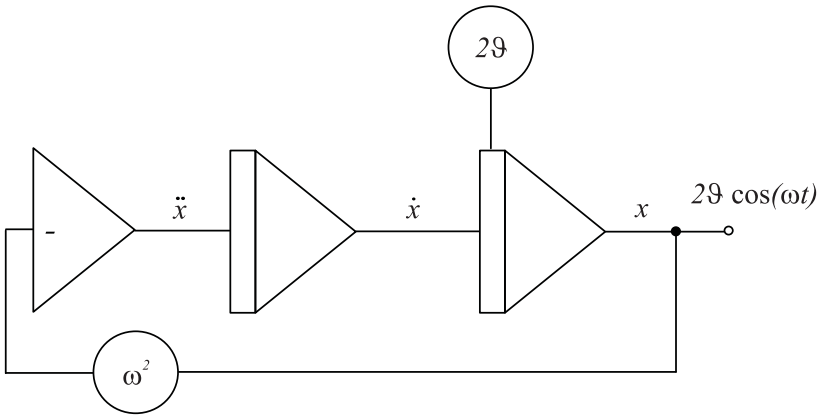
Časovno spremenljivi koeficient $2\vartheta\cos(\omega t)$ je zlasti pri simulaciji z analognim računalnikom smiselno realizirati na implicitni način

$$\begin{aligned} x &= 2\vartheta\cos(\omega t) \\ \dot{x} &= -2\vartheta\omega\sin(\omega t) \\ \ddot{x} &= -2\vartheta\omega^2\cos(\omega t) = -\omega^2 x \end{aligned} \tag{3.57}$$

Koeficient $2\vartheta\cos(\omega t)$ se torej lahko realizira s simulacijo sistema

$$\ddot{x} = -\omega^2 x \qquad x(0) = 2\vartheta \qquad \dot{x}(0) = 0 \tag{3.58}$$

z indirektnim načinom. Simulacijsko shemo prikazuje slika 3.28.



Slika 3.28: Generacija izraza $2\vartheta\cos(\omega t)$ z implicitno metodo

Naloga 3.4 Simulacija prenosne funkcije

Če je model predstavljen s prenosno funkcijo v faktorizirani obliki ali s produktom več prenosnih funkcij, lahko pomeni, da je le ta sestavljen iz več podmodelov, ki opisujejo zaključene funkcionalne sklope. Take podmodele ločeno simuliramo in jih na koncu ustrezno povežemo.[1]

Realni proces opisuje model

[1] Metoda zaporedne razčlenitve

$$G_P(s) = \frac{Y(s)}{U(s)} = \frac{1-4s}{(1+4s)(1+10s)} \tag{3.59}$$

ki ga lahko predstavimo kot zaporedno vezavo dveh prenosnih funkcij

$$G_P(s) = G_1(s)G_2(s) \tag{3.60}$$

$$\begin{aligned} G_1(s) &= \frac{1-4s}{1+4s} \\ G_2(s) &= \frac{1}{1+10s} \end{aligned} \tag{3.61}$$

Delitev prenosne funkcije (3.59) na funkciji $G_1(s)$ in $G_2(s)$ smo naredili na osnovi fizikalnega poznavanja realnega sistema. Prenosna funkcija $G_1(s)$ je namreč Padé-jeva aproksimacija zakasnitve (mrtvega časa) $2s$. Prenosno funkcijo $G_1(s)$ realizirajmo po delitveni metodi

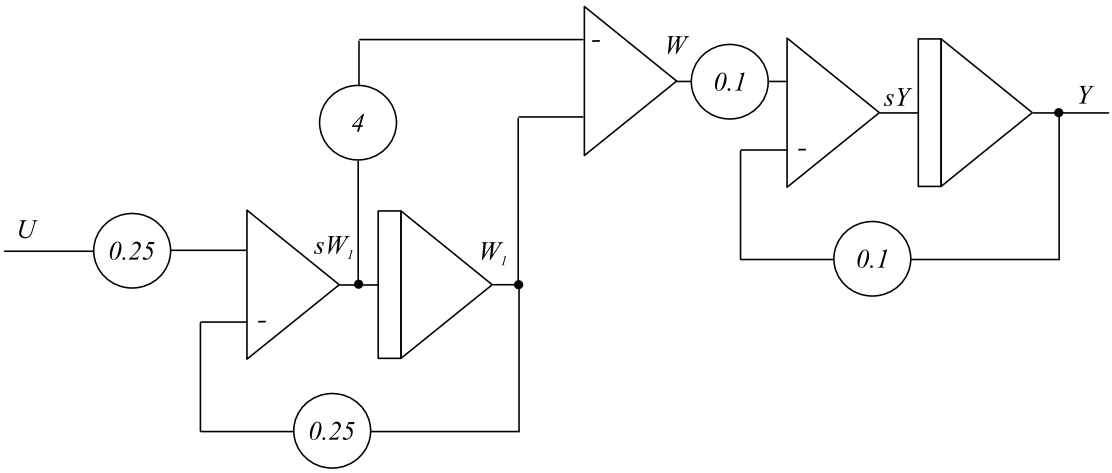
$$\begin{aligned} G_1(s) = \frac{W}{U} &= \frac{1-4s}{1+4s} \\ \frac{W_1}{U} &= \frac{1}{1+4s} \\ sW_1 &= 0.25U - 0.25W_1 \\ \frac{W}{W_1} &= 1-4s \\ W &= W_1 - 4sW_1 \end{aligned} \tag{3.62}$$

Prenosno funkcijo $G_2(s)$ pa realiziramo po vgnezdeni metodi[2]

$$\begin{aligned} G_2(s) = \frac{Y}{W} &= \frac{1}{1+10s} \\ Y &= \frac{1}{s}(0.1W - 0.1Y) \end{aligned} \tag{3.63}$$

[2]Ker prenosna funkcija nima končne ničle, v shemi ni razlike med vgnezdeno in delitveno metodo

Slika 3.29 prikazuje simulacijsko shemo.



Slika 3.29: Shema za simulacijo prenosne funkcije v faktorizirani obliki

Naloga 3.5 Simulacija diferencialne enačbe s kompleksnimi koeficienti

Diferencialna enačba ima obliko

$$\ddot{x} + (a + jb)\dot{x} + (c + jd)x = 0 \tag{3.64}$$
$$x(0) = e + jf \qquad \dot{x}(0) = g + jh$$

Ker so koeficienti kompleksni, moramo tudi rešitev diferencialne enačbe predpostaviti v kompleksni obliki

$$x(t) = x_r(t) + jx_i(t) \tag{3.65}$$

V tem primeru je sistem vzbujan s kompleksnimi začetnimi pogoji, lahko pa bi bil seveda vzbujan z realno ali kompleksno vhodno funkcijo, ki bi jo podajal izraz na desni strani enačbe (3.64). Enačbo (3.64) preuredimo v obliko

$$\ddot{x}_r + j\ddot{x}_i + (a + jb)(\dot{x}_r + j\dot{x}_i) + (c + jd)(x_r + jx_i) = 0 \tag{3.66}$$

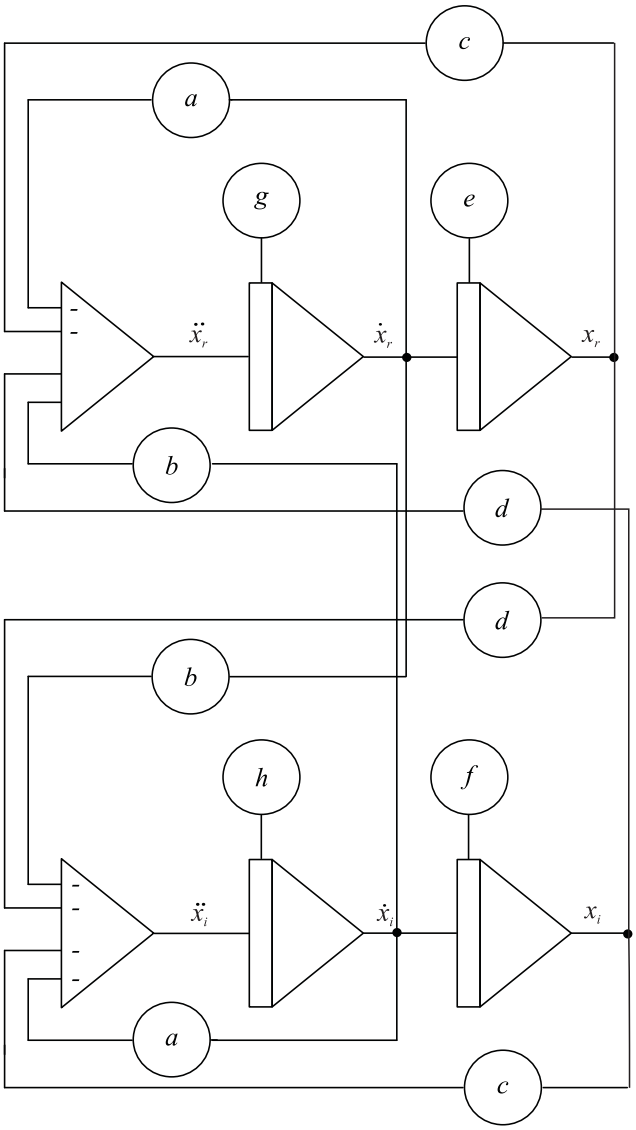
Ker je enačba (3.66) kompleksna, mora veljati ločeno za realni in imaginarni del, torej dobimo sistem dveh realnih diferencialnih enačb

$$\begin{aligned} \ddot{x}_r + a\dot{x}_r - b\dot{x}_i + cx_r - dx_i &= 0 \\ x_r(0) = e \qquad\qquad \dot{x}_r(0) &= g \\ \ddot{x}_i + b\dot{x}_r + a\dot{x}_i + dx_r + cx_i &= 0 \\ x_i(0) = f \qquad\qquad \dot{x}_i &= h \end{aligned} \tag{3.67}$$

Obe enačbi rešimo na običajni način z uporabo indirektne metode. Zato preuredimo enačbi (3.67) v obliko

$$\begin{aligned} \ddot{x}_r &= -a\dot{x}_r + b\dot{x}_i - cx_r + dx_i \\ \ddot{x}_i &= -b\dot{x}_r - a\dot{x}_i - dx_r - cx_i \end{aligned} \tag{3.68}$$

Simulacijsko shemo prikazuje slika 3.30.



Slika 3.30: Simulacijska shema za reševanje diferencialne enačbe s kompleksnimi koeficienti

Poglavje 4

Orodja za simulacijo dinamičnih sistemov

Metode za simulacijo zveznih dinamičnih sistemov lahko učinkovito uporabljamo le v povezavi z modernimi računalniškimi orodji. V tem poglavju bomo opisali osnovne lastnosti in razvrstitve simulacijskih sistemov in simulacijskih jezikov. V zadnjem delu bomo z dvema primeroma prikazali, kako pridemo do realnega modela (programa), če uporabimo simulacijski jezik tipa CSSL.

4.1 Osnovne lastnosti in razvrstitev simulacijskih sistemov

Moderna simulacijska orodja omogočajo simulacijo dinamičnih modelov na tak način, da se uporabnik lahko osredotoči na samo problematiko (modeliranje, simulacija), ne pa na programiranje problema. Posamezne vrste orodij se z uporabniškega vidika med seboj razlikujejo glede na:

- potreben čas, da postane uporabnik vešč simulacijskega orodja,
- uporabniško prijaznost,
- potreben čas za razvoj simulacijskega modela,
- možnosti pri spreminjanju simulacijskega modela,

- sposobnosti pri izvajanju simulacije (hitrost, interaktivnost, numerična robustnost, možnost simulacije v realnem času, ...),
- možnosti dokumentiranja modelov, rezultatov simulacije in dr.

Glede na uporabljeno programsko in materialno opremo lahko razdelimo moderna simulacijska orodja v tri skupine, t.j. v

- simulacijske sisteme na splošnonamenskih digitalnih računalnikih,
- simulacijske sisteme na namenskih digitalnih računalnikih in
- analogno - hibridne sisteme.

V prvo skupino prištevamo simulacijske sisteme, ki delujejo na tistih digitalnih računalnikih, ki delujejo po Von Neumann-ovem principu (SISD - single instruction, single data). V drugo skupino uvrščamo sposobne kompleksne računalniške sisteme, ki temeljijo na vektorskem (SIMD - single instruction, multiple data) ali na paralelnem izvrševanju operacij (MIMD - multiple instruction, multiple data) in v zvezi s tem potrebujejo ustrezno materialno in programsko opremo. Uporabljajo se za simulacijo velikih in zahtevnih problemov, predvsem pa so primerni za simulacijo v realnem času. V tretjo skupino pa sodijo analogno - hibridni sistemi, ki so z multiprocesorskim sistemom SIMSTAR ter z nekaterimi manjšimi in predvsem cenejšimi sistemi z mikroračunalniškim upravljanjem v osemdesetih letih še enkrat oživeli, danes pa se uporabljajo le še izjemoma v zelo specifičnih simulacijah (simulacija v realnem času s HIL, simulatorji v vojski, ...).

Simulacijski sistemi na splošnonamenskih računalnikih predstavljajo težišče nadaljnje obravnave.

4.1.1 Simulacijski sistemi na splošnonamenskih digitalnih računalnikih

Digitalni simulacijski sistemi na splošnonamenskih računalnikih so povzročili velik razmah simulacije kot moderne metodologije v osemdesetih letih. Namestitev ustreznega simulacijskega sistema na enem ali več računalnikih istega tipa ali celo na računalnikih različnih tipov je daleč najcenejša možnost, ki je zadovoljiva tako za pedagoške namene kot tudi za reševanje raznih znanstveno raziskovalnih

in razvojnih problemov. Na ta način navadno ni možno simulirati v realnem času zaradi omejene hitrosti splošnonamenskih računalnikov.

Schmidt (Schmidt, 1987) je priporočil razdelitev digitalnih simulacijskih sistemov na

- simulacijske pakete in
- simulacijske jezike.

Simulacijski paketi predstavljajo starejšo obliko digitalnih simulacijskih sistemov. Sestavljeni so iz glavnega programa in knjižnice podprogramov. Uporabnik mora v jeziku simulacijskega paketa (Visual Basic, C++, Fortran, Java, Matlab) opisati strukturo modela, pri tem pa lahko kliče vnaprej programirane podprograme iz knjižnice ali dodaja svoje programe. Prednost takega sistema je v tem, da ima uporabnik na voljo izvirni program za opis modela, ki ga dobro obvlada, kar daje določeno fleksibilnost. Slabost takega simulacijskega sistema pa je v tem, da mora uporabnik poznati vsaj bistvene značilnosti numeričnih postopkov (integracijskih metod), obvladati pa mora tudi programiranje, pri čemer obstaja velika verjetnost napak.

Zelo znan je paket ODEPACK (Ordinary Differential Equation PACKage) (Hindmarsh, 1983), ki je sestavljen iz modulov za reševanje navadnih diferencialnih enačb v jeziku FORTRAN. Enačbe modela je možno podati v eksplicitni ($\dot{x} = f(x,t)$) ali implicitni obliki (npr. $F(\dot{x},x,t) = 0$). Paket vsebuje tudi podprograme za reševanje togih sistemov. Znan je tudi paket DASSL (Differential / Algebraic System SoLver), ki omogoča numerično reševanje implicitnih sistemov diferencialnih in algebrajskih enačb.

Simulacijski jeziki pa predstavljajo sodobne in uporabnejše simulacijske sisteme. Uporabnik mora podati model in določene pogoje za izvrševanje simulacije na način, ki ga predpisuje sintaksa uporabljenega simulacijskega jezika.

Pri delu s simulacijskim jezikom je uporabnik sicer nekoliko manj fleksibilen kot s simulacijskim paketom, vendar pa v prevajalniško orientiranih jezikih tudi lahko dodaja svoje operacije. Prednost simulacijskega jezika je v bistveno večji enostavnosti pri uporabi. Uporabniku ni potrebno poznati višjega programskega jezika temveč le enostavnejši simulacijski jezik. Za neizkušenega uporabnika je simulacijski jezik bistveno primernejši.

Primerjava paketov in jezikov vodi do naslednjih zaključkov:

- Neizkušen uporabnik, ki bi rad hitro in enostavno rešil standardne simulacijske probleme, naj uporabi simulacijski jezik.

- Izjemoma je zelo komplicirane modele, ki vključujejo tudi nestandardne pristope, smiselno simulirati s simulacijskimi paketi.

Sodobni simulacijski sistemi omogočajo podajanje modela z grafičnim urejevalnikom. Čeprav izraz simulacijski jezik ob takem podajanju modela ni več najprimernejši, ker uporabnik ne piše programa v sintaksi simulacijskega jezika, pa večina literature tudi tak sistem imenuje simulacijski jezik. Nekateri sistemi iz grafično podanega modela generirajo program v simulacijskem jeziku (npr. okolja Modelica), nekateri pa ga prevajajo v druge oblike ali pa interpretersko izvajajo simulacijo (npr. Matlab-Simulink).

V nadaljevanju bomo od digitalnih simulacijskih sistemov obravnavali le simulacijske jezike.

4.1.2 Simulacijski sistemi na namenskih digitalnih računalnikih

Simulacija dinamičnih sistemov spada med operacije, ki zahtevajo največ izračunavanj in predvsem za delo v realnem času najsposobnejše računalnike. Zato je simulacija skupaj s področjem obdelave signalov najbolj vplivala na razvoj ustrezne namenske materialne in programske opreme.

Glavna omejitev konvencionalnih digitalnih računalnikov je Von Neumann-ov princip, ki temelji na istočasnem procesiranju enega podatka z eno instrukcijo (SISD - single instruction, single data). Sodobnejši supermini in superračunalniki so razširili svoje zmožnosti s t.i. vektorskim procesiranjem, kjer lahko ena operacija deluje nad nizom podatkov (SIMD - single instruction, multiple data). Superračunalniki so danes najsposobnejši računalniki (CRAY 1, CRAY X-MP, CRAY 2, CYBER-205, HITACHI, FUJITSU, NAS,...). Ekstremne sposobnosti pa nudijo za ekstremno visoko ceno. Zato je bilo ob koncu leta 1987 instaliranih vsega okrog 300 takih sistemov (Hallin, 1988). Imajo dobro dodelano programsko opremo (FORTRAN, PASCAL), za simulacijo je že možno uporabljati simulacijske jezike. Znano je, da simulacijska jezika CSSL in ACSL delujeta na nekaterih superračunalnikih (CRAY, CYBER 205 - Colijn, 1986). Toda v splošnem obstaja na superračunalnikih le simulacijska programska oprema za posebne namene. Zato se superračunalniki uporabljajo skoraj izključno za

reševanje takšnih problemov, za katere druge vrste računalnikov ne zadoščajo (meteorologija, dinamika tekočin, fizika plazme idr.).

Sodobni razvoj materialne in programske opreme pa nudi dve cenejši možnosti:

- vrstične procesorje in
- sisteme paralelnih procesorjev.

Oba načina se po zmožnostih približujeta superračunalnikom, žal pa še ni na voljo ustrezno izpopolnjene programske opreme.

Vrstični procesorji

Vrstični procesorji (array processor) so periferne računalniške enote, ki zelo povečajo sposobnosti računalnika, na katerega se priključijo. Omogočajo paralelno izvajanje določenih operacij s pomočjo več aritmetičnih enot. Lahko so v obliki modulov, ki se dodajo na vodilo glavnemu računalniku. Glavni računalnik služi za pripravo, prevajanje in nalaganje programa v periferni vrstični procesor. Ker programska oprema še ni dovolj dodelana, je programiranje zahtevno. Za nekatere vrste procesorjev obstojajo osiromašeni prevajalniki v jeziku FORTRAN, povezovalniki in ustrezne knjižnice.

V začetku (sredi sedemdesetih let) so vrstične procesorje največ uporabljali v procesiranju signalov, konec sedemdesetih let pa so se pojavile že prve simulacijske aplikacije. Leta 1984 je predstavljala simulacija že 30% vseh aplikacij z vrstičnimi procesorji. Najprej so se uporabljali predvsem za simulacijo v realnem času, saj so bili sposobni dovolj hitro simulirati kompleksne dinamične sisteme, v osemdesetih letih pa so se začeli uporabljati tudi za reševanje parcialnih diferencialnih enačb. Slednja problematika sicer navadno ne zahteva računanja v realnem času, zato pa so problemi računsko tako kompleksni, da je računanje s konvencionalnimi računalniki preveč dolgotrajno.

Med komercialnimi vrstičnimi procesorji obstaja tudi nekaj simulacijsko specializiranih procesorjev. Zanje je značilno, da imajo tako procesno enoto kot ustrezno programsko opremo prilagojeno specifiki simulacijskih operacij. Sicer je večina vrstičnih procesorjev bolj prirejena za obdelavo signalov (večje dimenzije vektorjev).

Med najbolj znanimi vrstičnimi simulacijskimi procesorji naj omenimo procesorja AD-10 in AD-100 (Applied Dynamic International), ki sta bila projektirana kot nadomestilo za hibridne sisteme (Grierson, 1986). Simulacijske probleme je na sistemih AD-10 in AD-100 bilo možno programirati zelo udobno z uporabo simulacijskega jezika ADSIM (ADvanced SIMulation language) na glavnem računalniku. Ta jezik sicer ni bil tako sposoben, kot so sorodni simulacijski jeziki vrste CSSL na splošnonamenskih računalnikih, omogočal pa je strukturno programiranje (sekcije INITIAL, DYNAMIC, TERMINAL), vseboval je več integracijskih metod, omogočal pa je tudi enostavnejše eksperimentiranje.

Enega najsodobnejših simulacijskih sistemov, ki uporablja vrstični procesor, je predstavljala simulacijska delovna postaja XANALOG-1000 (Schrage, Mc Ardle, 1986, Schmidt, Scheider, 1988). Proizvajalci so uporabili ceneni vrstični procesor in razvili popolnoma novo programsko opremo, ki je omogočala programiranje s pomočjo grafičnega bločnega urejevalnika. Ta je izvajal kompletno diagnostiko, prevajalnik pa je nato iz bločne sheme generiral program za vrstični procesor. Ob ustreznih perifernih enotah je bila postaja predvsem namenjena simulaciji v realnem času. Obstajala pa je tudi verzija programske opreme za osebni računalnik.

Paralelni procesorski sistemi

Čas za simulacijo na enoprocesorskem računalniku se zelo povečuje s kompleksnostjo problema. Če se hočemo temu vsaj delno izogniti, moramo uporabiti računalniške sisteme s paralelnimi procesorji (MIMD - multiple instruction, multiple data).

Pri simulaciji na paralelnih procesorskih sistemih si morajo procesorji med integracijo izmenjavati podatke. Učinkovito računanje zahteva ustrezno razdelitev programskih (modulnih) segmentov med procesorje, pri čemer morata biti izpolnjena predvsem naslednja dva pogoja:

- posamezni procesorji naj potrebujejo približno enake čase izračunavanja v fazi, ko ni potrebna medsebojna komunikacija in
- pri komunikaciji naj se prenaša čim manj podatkov.

Komunikacija je zelo kritična. Ker je za zvezno simulacijo značilno, da je čas izračunavanja relativno velik v primerjavi s časom komuniciranja, je uporaba paralelnih procesorskih sistemov pri simulaciji zelo učinkovita.

Pri programiranju je posebno pomembno, da dosežemo učinkovito izvajanje paralelnih operacij. Že pripravljeni program za enoprocesorski sistem je sicer možno reorganizirati tako, da se izvaja na več procesorjih. Vendar je ta metoda za simulacijo neučinkovita. Za učinkovito simulacijo je potrebno izhajati iz paralelne strukture, ki je vsebovana v problemu. Sheme modelov, ki so pripravljene za simulacijo na analognem računalniku ali za simulacijo z bločno-orientiranim digitalnim simulacijskim jezikom, so zelo primerne za razgradnjo problema za obravnavo na paralelnem procesorskem sistemu. Znani so nekateri algoritmični postopki (Boullard, Coen, 1985), ki razgradijo model na ustrezne paralelne poti. Osnovni elementi takega algoritmičnega postopka so bloki (sumator, integrator, prenosna funkcija, ...), za katere je potrebno poznati čas za izračunavanje. Algoritem zahteva, da prekinemo zanke v modelu na izhodih blokov z zakasnitvenimi atributi (integrator, zakasnitev, zadrževalnik). S pomočjo ustreznih postopkov pridemo do paralelnih poti, ki se zaključijo v blokih (vozliščih) z zakasnitvenimi atributi. Vsak simulacijski cikel se začne z znanimi vrednostmi izhodov teh blokov in se konča z izračunom izhodov teh blokov za naslednji cikel. Število paralelnih poti definira maksimalno potrebno število procesorjev. Vsaki paralelni poti je pridružen tudi čas izvajanja in najdaljši čas vpliva na hitrost izvrševanja simulacije.

Na podlagi takšnih algoritmičnih postopkov bo možno avtomatizirati celoten postopek. Uporabnik bo definiral model na običajen način z uporabo simulacijskega jezika.

Eden bolj znanih paralelnih procesorskih sistemov, ki se je uporabljal v simulaciji, je bil paralelni procesor DELFT (DPP) (Bruijn, Soppers, 1986). Programiranje je bilo možno s pomočjo glavnega računalniku tipa VAX v nekakšnem makro zbirniku in v simulacijskem jeziku PARSIM. Jezik je vseboval le osnovne bloke (kot analogni računalnik) ter dvokoračno Adams-Bashforth-ovo integracijsko metodo. Uporabnik je moral v simulacijskem modelu definirati, v kateri procesni enoti naj se izvršuje določen del modela.

Na področju digitalne simulacije zveznih sistemov pa največ obetajo transputerji (Hamblen, 1987, Eckelmann, 1987). S pomočjo teh integriranih vezij je možno sestaviti cenen paralelni procesorski računalnik izjemnih sposobnosti.

Transputer je računalnik v obliki integriranega vezja. Eden prvih transputerjev T800 je imel 32 bitni procesor (10 MIPS (milijon instrukcij v sekundi)) in 32 bitno aritmetično enoto z plavajočo vejico (1.5 MFLOPS), 4K zlogov pomnilnika RAM, vhodno izhodne vmesnike ter uro. S pomočjo serijskih povezav se lahko zelo enostavno poveže poljubno število transputerjev. Vsak ima štiri seri-

jske linije, zato komunikacija ne predstavlja posebno ozkega grla. Komunikacija med transputerji poteka asinhrono. Pretvornike A/D in D/A je možno direktno priključiti na transputer.

Programska oprema za tovrstne računalnike še ni dodelana. Podjetje INMOS je prvo razvilo visok programski jezik OCCAM, možno pa je programirati tudi v jeziku C. OCCAM je strukturiran jezik, ki omogoča zaporedne in vzporedne operacije na enem ali več transputerjih (SEQ blok - operacije se izvajajo zaporedno, PAR blok - operacije se izvajajo vzporedno z drugimi operacijami na drugih transputerjih). V vsakem bloku je potrebno definirati operacije čitanja podatka s serijske linije in operacije pošiljanja podatkov na serijske linije. Te operacije omogočajo komuniciranje med transputerji (primer 5.13).

Do paralelne razgradnje pridemo na enak način kot pri drugih paralelnih procesorskih sistemih. Prednost pa je v tem, da imajo transputerski sistemi veliko število paralelnih procesorjev, tako da je možno enostavne modele razgraditi na posamezne integratorje (sumacijske z vhodnimi ojačenji). Ob inicializaciji vsi transputerji pošljejo na izhodne serijske linije začetne vrednosti blokov z zakasnitvenim atributom, nakar ločeno računajo, dokler v ustreznem trenutku ni potrebna komunikacija.

Delovanje transputerskega sistema je tem bolj učinkovito, čim več je računanja v transputerjih v primerjavi s prenašanjem podatkov preko serijskih linij. Zato je delovanje transputerjev učinkovitejše v primeru kompleksnejših integracijskih metod.

4.1.3 Analogno-hibridni sistemi

Analogno - hibridni sistem je simulacijski sistem, ki ga dobimo s povezavo analognega in digitalnega računalnika. Analogni računalnik omogoča izredno hitro simulacijo zveznih dinamičnih sistemov s pomočjo povezave elektronskih komponent (integratorji, sumatorji, potenciometri, množilniki,...). Digitalni računalnik pa ima predvsem razne krmilne funkcije, v modernih izvedbah pa tudi funkcije za opis modela, za dokumentacijo rezultatov, včasih pa omogoča tudi pravo hibridno simulacijo, t.j. da se nekateri deli modela simulirajo na analognem, nekateri pa na digitalnem delu. Običajno je možno analogni del uporabiti kot samostojni simulacijski sistem.

Osemdeseta leta predstavljajo zadnji preporod analogno-hibridnih sistemov. Po-

javili so se cenejši mikroprocesorsko vodeni sistemi s cenenimi in zanesljivimi integriranimi komponentami. Leta 1983 pa je prišel na trg novi večprocesorski sistem SIMSTAR podjetja EAI, ki združuje prednosti pravega paralelnega računanja (predvsem analogna integracija) in večprocesorskega izvajanja drugih operacij. Ima izredno sposobno programsko opremo, ki omogoča, da uporabnik programira računalnik s pomočjo simulacijskega jezika (primer 7.16).

Nove vrste računalnikov so bolj ali manj odpravile večino slabosti analognega računanja. Te slabosti so bile v glavnem naslednje (Havranek, 1983):

- zahtevno programiranje in težko odpravljanje napak,
- zahtevno vzdrževanje in
- visoka cena.

Digitalni simulacijski sistemi so ob današnjem stanju primernejši za reševanje večine problemov. Toda še vedno obstojajo primeri, kjer prinaša hibridna simulacija predvsem zaradi velike hitrosti določeno prednost (npr. simulacija v realnem času).Učinkovita je tudi uporaba v izobraževanju, saj je z njimi možno bolj ilustrativno prikazovati določene lastnosti dinamičnih sistemov. Ne glede na to, da so hibridni sistemi vsaj v smislu materialne opreme v zatonu, pa koncepti analognohibridne simulacije ostajajo pomembni tudi v prihodnje (npr. paralelizacija kode za paralelno - procesorske sisteme, skaliranje, ...).

4.2 Osnovne lastnosti in razvrstitev simulacijskih jezikov

Simulacijski jeziki predstavljajo moderno simulacijsko orodje, ki se zaradi cenenosti največ uporabljajo. V glavnem so precej neodvisni od materialne in sistemske programske opreme. Edina njihova slabost je relativna počasnost izvajanja, kar zlasti omejuje njihovo uporabnost za simulacijo v realnem času.

Simulacijske jezike glede na splošnost pri opisu modelov lahko razvrstimo v:

- splošnonamenske jezike in

- namenske oz. problemsko orientirane jezike.

Pri *splošnonamenskih simulacijskih jezikih* je način opisa modela precej neodvisen od vrste problema. Znani splošnonamenski jeziki so npr. ACSL, TUTSIM, CSMP. Z njimi opisujemo probleme, ki so opisani z algebrajskimi in diferencialnimi enačbami. Lahko rečemo, da so toliko splošni, kolikor so v teoretičnem modeliranju splošne algebrajsko - diferencialne enačbe. Zato so taki jeziki uporabni na številnih področjih, vendar pa morajo imeti uporabniki osnovna znanja iz modeliranja in simulacije. Zavedati pa se moramo, da je simulacijski jezik splošen le zato, ker z njim lahko obravnavamo splošne matematične in fizikalne zakonitosti, ne pa zato, ker bi bil sposoben reševati vse probleme. Splošnonamenski jeziki so sposobni na specialnih področjih reševati ne preveč zahtevne, predvsem pa ne preveč kompleksne probleme.

Problemsko orientirani simulacijski jeziki pa so namenjeni za učinkovito modeliranje in simulacijo na posebnih področjih. Znani so npr. simulacijski jeziki za simulacijo električnih vezij SPICE, PSPICE (Micro Sim Corp.), ECAP. Te jezike je možno rutinsko uporabljati tudi brez poglobljenega znanja o modeliranju in simulaciji. Z razumevanjem analogije fizikalnih in matematičnih zakonitosti med posameznimi področji (npr. analogija električnih in mehanskih veličin) jih je možno uporabljati tudi na drugih področjih. Vendar se ta možnost redko uporablja, saj tako delo ponovno zahteva poglobljeno znanje iz modeliranja in simulacije, vprašljiva pa je tudi učinkovitost numeričnih postopkov.

Ena od značilnosti modernih simulacijskih jezikov je tudi v tem, da lahko uporabnik na enostavne načine vključi submodele, ki jih potrebuje za reševanje svojih problemov (npr. z makro jezikom) in na ta način dejansko generira nov jezik, ki je bolj primeren za posebne namene. Vprašljiva pa ostane učinkovitost matematičnih postopkov in programov, do katerih uporabnik nima dostopa. Takšnim jezikom pravimo višji splošnonamenski jeziki, ker vsebujejo kompleksne operatorje za opise modelov. Nižji simulacijski jeziki vsebujejo le osnovne gradnike za opis modela, elemente programskega krmiljenja, ustrezne numerične postopke ter možnosti za predstavitev rezultatov.

Objektno orientiran in več domenski jezik Modelica pa združuje dobre lastnosti splošnonamenskih in namenskih jezikov. Osnovni koncept je povsem splošnonamenski, ob uporabi knjižnic pa postane zelo sposobno modelersko orodje za neko konkretno domeno.

V tem delu pretežno obravnavamo splošnonamenske simulacijske jezike. Uporabl-

jamo jih lahko za reševanje relativno zahtevnih problemov na vseh področjih znanosti in tehnike, učinkoviti pa so tudi v izobraževanju.

Glede na vrsto modela, ki ga opisujejo, delimo simulacijske jezike na:

- zvezne,
- diskretne in
- kombinirane.

Slednja delitev je pomembna tako v smislu modeliranja kot tudi razvoja simulacijskih jezikov, saj posamezne vrste simulacij (zvezna, diskretna in kombinirana) zahtevajo zelo različne koncepte pri zasnovi jezikov. Zato bomo v naslednjih podpoglavjih te tri vrste simulacijskih jezikov nekoliko podrobneje obravnavali.

Posebno problematiko predstavljajo jeziki, ki omogočajo simulacijo v realnem času. Ti so sicer lahko zvezni, diskretni ali kombinirani, vendar si bomo v tem delu ogledali le problematiko, ki v glavnem zadeva zvezno simulacijo.

4.2.1 Zvezni simulacijski jeziki

Zvezni simulacijski jeziki se uporabljajo za simulacijo zveznih dinamičnih sistemov. Običajno imajo tudi nekatere sicer omejene sposobnosti kombinirane simulacije, vendar v glavnem le v smislu realizacije nezveznosti, ne pa tudi v smislu numerično pravilne obdelave nezveznosti.

Glede na to, ali so modeli opisani z navadnimi (ODE) ali parcialnimi diferencialnimi enačbami (PDE), ločimo simulacijske jezike:

- za simulacijo problemov, opisanih z ODE in
- za simulacijo problemov, opisanih s PDE.

Prvo imenovani simulacijski jeziki so danes izredno popolni, medtem ko dosedaj še ni na voljo splošnonamenskih jezikov za reševanje PDE, čeprav so se prvi taki programski sistemi pojavili že okoli leta 1970. Numerična problematika je v

sistemih, ki jih opisujejo PDE mnogo zahtevnejša, kar lahko ocenimo z definicijo učinkovitosti integracijskega algoritma za reševanje določenega problema:

$$\eta(a,p) = \frac{t_{cpu}(a^*,p)}{t_{cpu}(a,p)} \tag{4.1}$$

kjer pomenijo

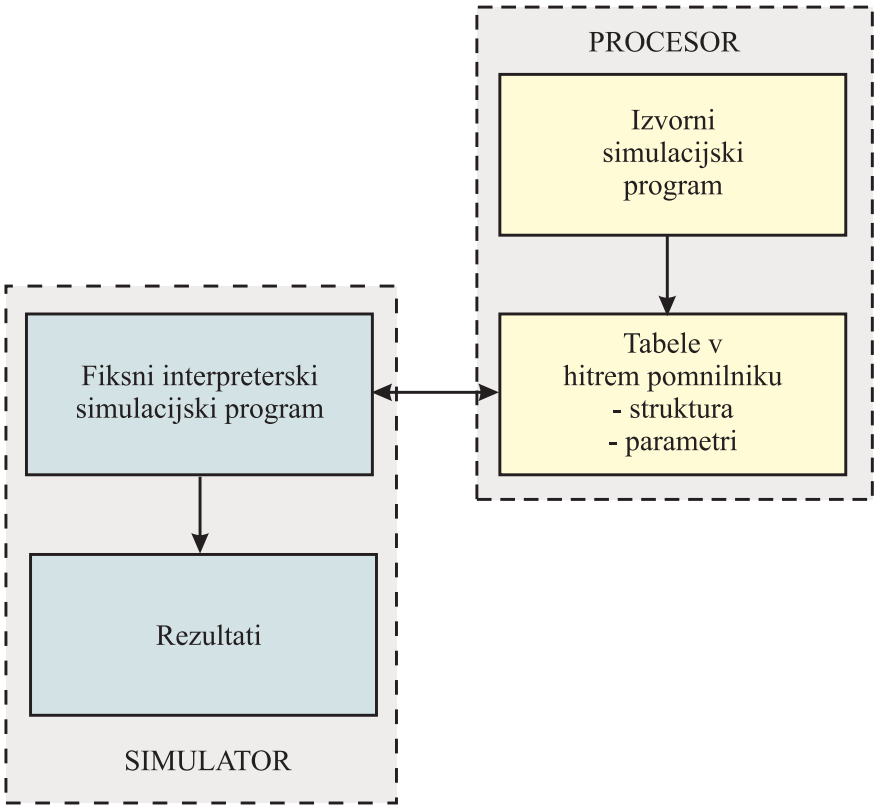
t_{cpu}	...	potreben čas CPU za rešitev problema,
η	...	učinkovitost integracijskega algoritma,
a	...	oznaka uporabljenega algoritma,
a^*	...	oznaka optimalnega algoritma za dani problem in
p	...	simulacijski problem.

Pri reševanju problemov ODE učinkovitost integracijskega algoritma η zelo redko doseže vrednost, ki je manjša od 0.01 za kakršenkoli problem in algoritem razen v primerih zelo togih ali oscilatornih sistemov. Pri reševanju problemov PDE pa je η mnogo manjši. Če upoštevamo samo metode, ki se često uporabljajo pri reševanju eliptičnih problemov, analiza pokaže, da je vrednost učinkovitosti integracijske metode η reda 10^{-3} do 10^{-6}. Torej je v tem primeru še bolj pomembna izbira ustreznega integracijskega algoritma. Metode za reševanje problemov PDE se tako razlikujejo, da je vprašljivo, če je sploh možno narediti povsem splošnonamenski simulacijski jezik. Danes znani simulacijski jeziki se lahko uporabljajo le za specializirane probleme (DSS, LEANS, FORSIM,...). V nadaljnjem delu bomo obravnavali le simulacijske jezike za reševanje ODE problemov.

Iz načina realizacije sledi najpomembnejša razdelitev tovrstnih simulacijskih jezikov na:

- interpreterske in
- prevajalniške.

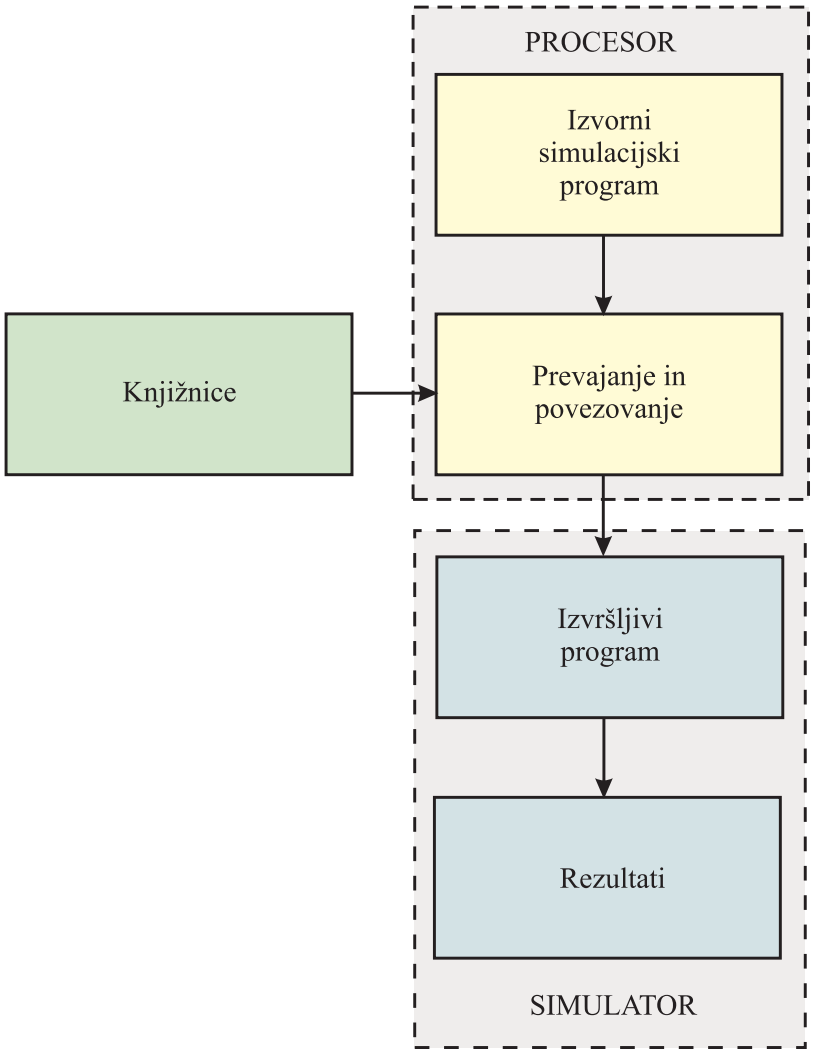
Pri interpreterskih simulacijskih jezikih (slika 4.1) se iz programa, ki ga napiše uporabnik, zgradijo v hitrem pomnilniku tabele, ki opisujejo strukturo in parametre modela. Fiksni interpreterski simulacijski program v fazi simulacije izvaja bloke, ki opisujejo model, po ustreznem vrstnem redu. Uporabnik lahko interaktivno spreminja strukturo in parametre modela brez kakršnegakoli prevajanja. Zato je interpreterski način zelo primeren pri razvoju modela, ko pogosto spreminjamo tudi strukturo. Slabost interpreterskega načina pa je počasnejša simulacija,



Slika 4.1: Zgradba interpreterskega simulacijskega jezika

kar ni primerno za simulacijo že preizkušenih kompleksnih modelov. Interpreterski jeziki so zlasti neprimerni, če simulacijski tek predstavlja samo del kompleksnega eksperimenta (npr. optimizacija, analiza robustnosti, ...). Slabost je tudi, da uporabnik ne more dodajati svojih blokov, ker interpreterski način zahteva glavni simulacijski program, ki je predhodno povezan s knjižnico, kar povzroča nefleksibilnost interpreterskih jezikov. Tretja slabost interpreterskih jezikov pa je ta, da so skoraj vedno bločno orientirani, kar pomeni, da je možno model opisati le z relativno enostavnimi vnaprej pripravljenimi bloki. Uporabnik mora definirati parametre blokov in načine povezave. Zato so simulacijski programi ponavadi dolgi, nepregledni in nemodularni. Omenjene slabosti rešujejo nekateri simulacijski jeziki z velikim številom blokov. To daje jeziku določeno kompaktnost, kar pomeni, da uporabnik redkokdaj pride do problema, ko bi potreboval vključitev novega lastnega bloka. Množica blokov pa predstavlja problem pri učenju jezika.

Prevajalniški simulacijski jeziki (splošno strukturo prikazuje slika 4.2) imajo manj slabih lastnosti. Nekateri jeziki prevajajo izvorni program direktno na strojni



Slika 4.2: Zgradba prevajalniškega simulacijskega jezika

nivo. Ker so taki prevajalniki zelo hitri, omogočajo visoko stopnjo interaktivnosti tudi pri spreminjanju strukture modela. V glavnem pa imajo omejene zmožnosti za reševanje kompleksnih problemov. Sodobni simulacijski jeziki ponavadi prevajajo izvorni program v nek višji splošnonamenski programski jezik (običajno FORTRAN). Zato lahko uporabnik sam v simulacijski jezik na enostaven način vključuje nove operacije. Prevajalniški jeziki omogočajo lažjo prenosljivost na druge sisteme, hitrejšo simulacijo, veliko fleksibilnost in reševanje obsežnejših sistemov. Ker se generirajo neodvisni simulacijski programi za izvajanje simulacije, so primerni tudi za gradnjo specifičnih aplikativnih simulacijskih programov. Njihova slabost pa je v tem, da je prevajanje, ki je potrebno ob vsaki spremembi

strukture simulacijskega programa, lahko precej dolgotrajno. Zato prevajalniški simulacijski jeziki niso najprimernejši v fazi razvoja modelov.

Medtem ko so interpreterski jeziki skoraj vedno *bločno orientirani*, pa razdelimo prevajalniške jezike glede na način podajanja programa na *bločne* (eksplicitna bločna struktura) in *enačbne* (implicitna bločna struktura). Pri enačbnih jezikih je možno bloke modela opisati s poljubnimi matematičnimi izrazi, ki jih dovoljuje ciljni jezik prevajalnika. Razvoj simulacijskih programov je hitrejši, le-ti so bistveno krajši, razumljivejši in bolj modularni. Nekateri sodobni simulacijski jeziki (ESL, PSI) omogočajo interpreterski in prevajalniški način delovanja.

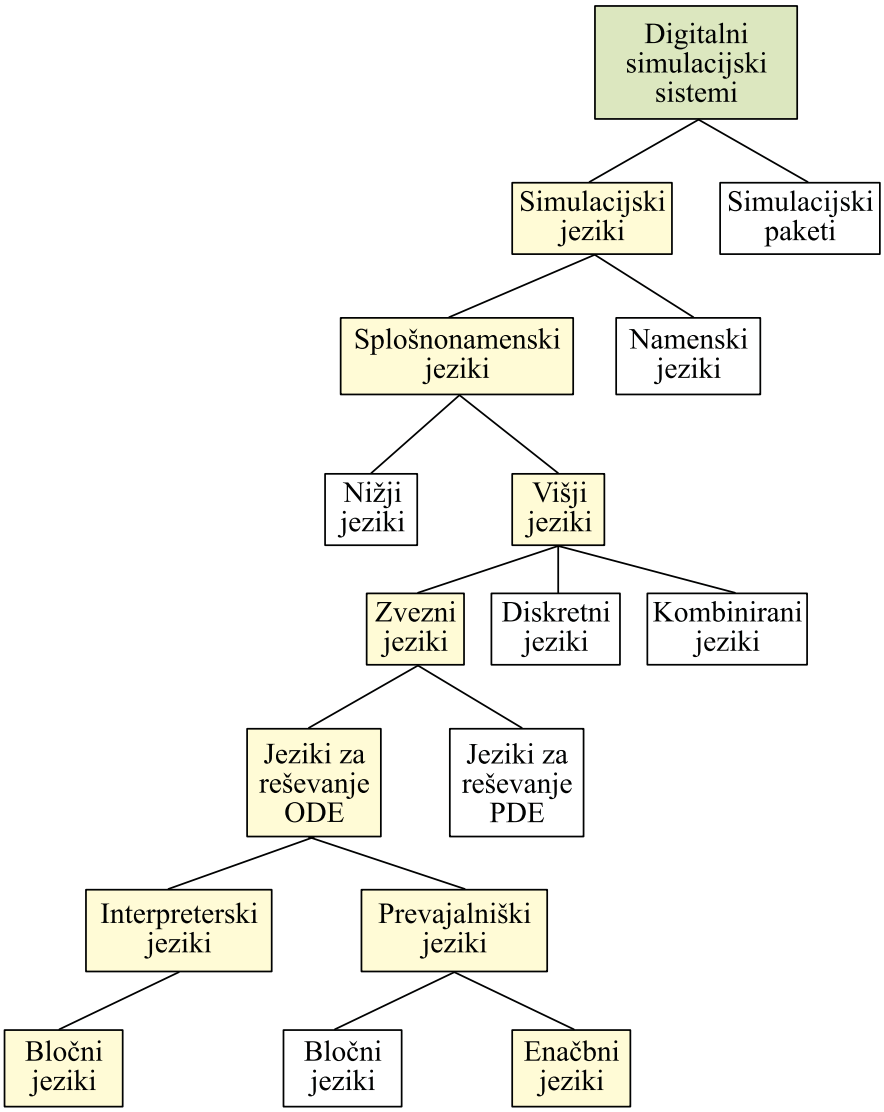
Celotno razdelitev digitalnih simulacijskih sistemov na splošnonamenskih računalnikih prikazuje slika 4.3. Razdelitev upošteva le jezike, ki jih bomo obravnavali v nadaljevanju.

4.2.2 Diskretni simulacijski jeziki

Diskretni simulacijski jeziki služijo za simulacijo diskretnih sistemov oz. dogodkov. Le-te delimo v dve skupini glede na način, kako se med simulacijo povečuje neodvisna spremenljivka, t.j. običajno čas. V večini tovrstnih simulacijskih jezikov se čas ob nastopu nekega dogodka avtomatično inkrementira na čas nastopa tega diskretnega dogodka. Ti jeziki se zato imenujejo dogodkovno orientirani simulacijski jeziki. V nekaterih, predvsem starejših diskretnih simulacijskih jezikih (npr. ena od verzij GPSS) pa se čas sinhrono povečanje z enakomernim prirastkom. Uporabnik mora v tem primeru paziti, da je prirastek zadosti majhen, da ne pride do večjih napak. Tem jezikom pravimo intervalno orientirani simulacijski jeziki.

Sodobni diskretni simulacijski jeziki so dogodkovno orientirani. Glede na vgrajene značilnosti se med seboj zelo razlikujejo. Najpomembnejša značilnost je strategija izbire naslednjega dogodka (podobno kot integracija pri zveznih simulacijskih jezikih). V glavnem uporabljajo simulacijski jeziki tri strategije (Hooper, 1987 , Neelamkavil, 1987):

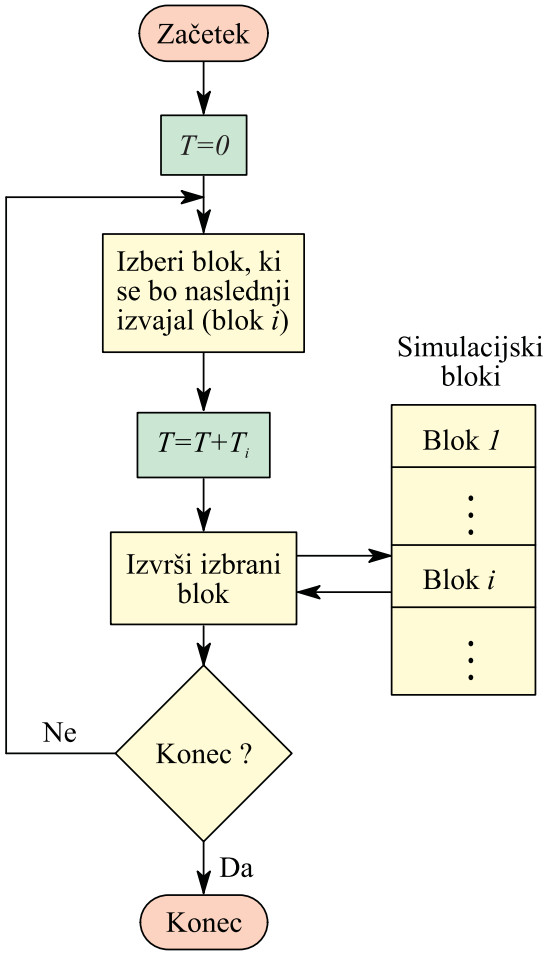
- strategija razporeda dogodkov (event scheduling),
- strategija odbiranja aktivnosti (activity scanning) in
- strategija procesne interakcije (process interaction).



Slika 4.3: Vrste digitalnih simulacijskih sistemov na splošnonamenskih računalnikih

Moderni simulacijski jeziki lahko vsebujejo tudi kombinacije omenjenih strategij.

Za vse strategije velja, da se ob naslednjem dogodku izvrši ustrezni modelni blok (podprogram), ki spremeni stanje sistema. Vse strategije vsebujejo tudi koncept pogojnih in brezpogojnih dogodkov. Brezpogojni dogodki se izvršijo ob točno določenih trenutkih, pogojni dogodki pa so odvisni tudi od stanj sistema. Splošno strukturo diskretne simulacije prikazuje slika 4.4.



Slika 4.4: Splošna struktura diskretne simulacije

Strategija razporeda dogodkov zahteva, da se brezpogojni dogodki izvajajo drug za drugim, kakor so programirani. Ob nastopu dogodka program iz liste dogodkov izbere tistega, ki ima najzgodnejši čas nastopa, čas pa se avtomatsko poveča na čas nastopa tega dogodka. Vsi pogoji, razen časovnih, morajo biti znotraj blokov, ki opisujejo dogodke. Simulacija se zaključi, ko se izvršijo vsi dogodki.

Aktivnost je stanje sistema med dvema zaporednima dogodkoma in definira prehod med njima. Strategija odbiranja aktivnosti ne vsebuje liste dogodkov, temveč poteka simulacija od dogodka do dogodka z odbiranjem aktivnosti. Vsaka aktivnost vsebuje logični pogoj, ki zavisi od simulacijskega časa in stanja sistema. V vsakem časovnem koraku se odbira status vseh aktivnosti (testirajo se stanja vseh objektov v sistemu). Na ta način se vrstni red dogodkov vzpostavlja avtomatično glede na izbrane aktivnosti. Večina modernih jezikov ne uporablja te

metode.

Objekte diskretnega modela (npr. stranka v poštnem uradu) in dogodke lahko povežemo v procese (npr. proces je čakanje stranke in streženje stranke) in uporabimo metodo procesne interakcije. Pri tem opišemo vedenje sistema z nizom procesov, ki sestojijo iz nizov medsebojno izključujočih se aktivnosti, kar pomeni, da se v določenem trenutku lahko začne le ena aktivnost. Procesi se lahko prekrivajo, interakcije med procesi pa se uporabljajo za opis modela. Poudarek je na razvrščanju procesov, dogodki pa se izvršujejo indirektno z aktiviranjem procesa (razporejanje in izvrševanje blokov, ki opisujejo akcije procesov), ki se v danem trenutku nahaja na vrhu liste. Procesi se lahko prekinjajo, obstajajo pa tudi t.i. reaktivacijske točke. Konflikt prekrivanja procesov se vrši s čakanjem v vrsti ali z zakasnitvijo.

Modeliranje z metodo procesne interakcije je enostavnejše, ker je programska struktura zelo podobna modelni strukturi, nudi pa manj programskih možnosti in fleksibilnosti v primerjavi z metodo razporeda dogodkov.

Tabela 4.1 prikazuje razvrstitev diskretnih simulacijskih jezikov glede na različne strategije. Vidimo, da imajo nekateri simulacijski jeziki različne možne strategije (SIMSCRIPT, SLAM).

Tabela 4.1: Razvrstitev diskretnih simulacijskih jezikov glede na strategije

Strategija razporeda dogodkov	Strategija odbiranja aktivnosti	Strategija procesne interakcije
GASP (II,IV)	AS	GPSS(/360,V,/H)
SIMPSCRIPT (I.5,II,II.5)	CSL	Q-GERT
SLAM, SLAM II	ECSL	SIMSCRIPT II.5
SIMAN	ESP	SLAM,SLAM II
	SIMON	SIMAN
		SIMULA
		ARENA

4.2.3 Kombinirani simulacijski jeziki

Kombinirani simulacijski jeziki vsebujejo zmožnosti zveznih in diskretnih simulacijskih jezikov (integracija, simulacija diskretnih dogodkov). Dolgo so menili,

da so tovrstni simulacijski jeziki nepotrebni, ker so jeziki za simulacijo zveznih sistemov z minimalnimi zmožnostmi simulacije diskretnih dogodkov zadostovali za simulacijo realnih tovrstnih problemov, diskretni jeziki pa so zadostovali za simulacijo diskretnih dogodkov. Večina jezikov in paketov za kombinirano simulacijo je nastala z razširitvijo čistih diskretnih simulacijskih jezikov (npr. GASP - IV oz. GASP - II). Razlog je v tem, ker so zvezni simulacijski jeziki zahtevnejši glede uporabe numeričnih postopkov, pri diskretnih simulacijskih jezikih pa so mnogo bolj kompleksni strukturni koncepti. Diskretne simulacijske jezike je v glavnem potrebno nadgraditi le z ustreznimi integracijskimi postopki. V praksi pa v glavnem večina potreb po kombinirani simulaciji izhaja iz problemov, ki se rešujejo s simulacijskimi jeziki za zvezne sisteme, medtem ko problemi, ki se rešujejo z diskretnimi simulacijskimi jeziki, le redko potrebujejo integracijske postopke. Nekateri uporabniki zveznih simulacijskih jezikov želijo, da bi imeli na voljo določene zmožnosti kombinirane simulacije. Zato razvijalci širijo tudi zvezne simulacijske jezike v smeri, za katero smo sicer omenili, da je dosti zahtevnejša.

Bistvo sposobnih kombiniranih simulacijskih jezikov je v tem, da imajo razen mehanizmov zveznih in diskretnih jezikov tudi mehanizme za prehod iz zveznega dela modela v diskretnega in obratno. Zaradi pogostih nastopov nezveznosti morajo imeti vgrajene ustrezne numerične postopke za pravilno obdelavo nezveznosti. To lastnost imajo le nekateri obstoječi zvezni simulacijski jeziki ter nekateri jeziki nove generacije.

Za razliko od drugih simulacijskih jezikov, je na trgu zelo malo jezikov, ki zadovoljivo rešujejo probleme kombinirane simulacije (GASP V, COSY, SYSMOD, COSMOS). Zelo izčrpen prispevek, ki obravnava probleme kombiniranih simulacijskih jezikov, je objavil Cellier, 1979.

4.2.4 Jeziki za simulacijo v realnem času

Nekateri splošnonamenski simulacijski jeziki se že relativno dolgo uporabljajo tudi za simulacijo v realnem času. Zaradi težav pri tovrstni simulaciji pa je njih uporaba zelo omejena. Zato v literaturi zasledimo sorazmerno malo podatkov (ACSL - Gauthier, 1987, MUSIC - Cser in ostali, 1986).

Osnovna zahteva, da simulacijski jezik lahko izvaja simulacijo v realnem času, je sinhronizacija z realnim časom. Za uspešno uporabo pa mora jezik imeti še druge lastnosti:

- čim hitrejše izvajanje simulacije in s tem v zvezi učinkovite numerične postopke; to je zlasti pomembno pri hitrih mehanskih in električnih sistemih, medtem ko pri procesnih sistemih ponavadi hitrost ni tako pomembna,
- večje interaktivne zmožnosti med simulacijo ter
- programsko opremo za komuniciranje s priključenimi napravami in/ali procesi.

Za hitro izvajanje simulacije so v preteklosti zlasti uporabljali bločne jezike, ki so s prevajanjem na strojni nivo omogočili učinkovito simulacijo. Prav tako so v preteklosti veliko uporabljali poenostavljeno aritmetiko (s fiksno decimalno vejico ali celo celoštevilčno), zato je bilo potrebno normirati enačbe. Danes, ko imajo že ceneni osebni računalniki aritmetične koprocesorje, se tak način ne izplača več, saj so prihranki minimalni in ne odtehtajo vloženega truda. Moderna materialna oprema zato tudi enačbnim jezikom omogoča, da se uspešno uporabljajo za simulacijo v realnem času.

Učinkoviti in poenostavljeni numerični postopki lahko prihranijo precej časa v obeh fazah simulacije in sicer pri izvajanju

- podprograma za integracijo in
- podprograma za izračun odvodov.

Po nekaterih ocenah zahteva prva faza za izvajanje metode Runge-Kutta višjega reda ob ne preveč kompleksnih modelih cca. 70% potrebnega računalniškega časa, druga faza pa 30%. Zato je zelo pomembno, da izločimo pri simulaciji v realnem času iz integracijskega podprograma vse, kar ni nujno potrebno. Metode s prilagodljivim korakom se ne uporabljajo. Veliko se uporabljajo algoritmi, ki jih pri simulaciji v nerealnem času le redko uporabljamo, npr. Eulerjeve metode (metode prvih in zadnjih diferenc), trapezno pravilo (bilinearna transformacija) in druge. Če pa je možno simulacijski model pretvoriti v diskretno obliko, pa je seveda najbolj učinkovita diskretna simulacija. V specializiranih orodjih za simulacijo zveznih dinamičnih sistemov je včasih možno precej pridobiti na hitrosti tudi s kodiranjem integracijskega algoritma s pomočjo jezikov, ki so bližji strojnemu nivoju.

Pri računanju odvodov pa je pomembno, da že enačbe modela podamo na tak način, da zahtevajo čim manj izračunavanja. Vnaprej programirani bloki ne smejo

vsebovati redundantnosti. Veliko časa lahko pridobimo z uporabo učinkovitih aproksimacijskih postopkov za korenjenje, logaritmiranje, eksponenciranje in za računanje trigonometričnih funkcij ter drugih funkcijskih generatorjev.

Interaktivne zmožnosti med simulacijo morajo omogočati sprotno spremljanje rezultatov ter direktne posege v simulacijo. Zlasti je pomembno, da lahko uporabnik med simulacijo spreminja parametre modela in da lahko realne signale iz okolice zamenja v vsakem trenutku s simuliranimi signali. Učinkovito spremljanje rezultatov pa naj omogoča prikaz spremenljivk na načine, ki so prilagojeni obravnavanim fizikalnim veličinam (ne le funkcijska odvisnost (npr. y(t)) ampak tudi v obliki "voltmetrov", "termometrov", ...).

Pri simulaciji v realnem času imamo navadno na računalnik priključeno zunanjo materialno opremo. Zato mora imeti sistem ustrezne periferne enote: module za zajemanje in oddajanje podatkov (pretvorniki A/D in D/A, digitalni vhodno-izhodni moduli), časovni modul, po potrebi modul za sprejem prekinitev in druge.

4.2.5 Primeri uporabe zveznih enačbno orientiranih simulacijskih jezikov

Uporabnost enačbno orientiranih simulacijskih jezikov bomo prikazali na razvoju dveh simulacijskih programov. Model ekološkega sistema roparjev in žrtev in regulacijski sistem ogrevanja prostora bomo simulirali v jeziku, ki upošteva standard CSSL'67 (Strauss, 1967).

Primer 4.1 Model ekološkega sistema roparjev in žrtev

Namen tega primera je, da seznani bralca z osnovno uporabnostjo digitalnih simulacijskih jezikov. Ekološki sistem roparjev in žrtev, ki ga opisujejo enačbe

$$\begin{aligned} \dot{x}_1 &= a_{11}x_1 - a_{12}x_1x_2 \\ \dot{x}_2 &= a_{21}x_1x_2 - a_{22}x_2 \\ x_1(0) &= x_{10} \\ x_2(0) &= x_{20} \end{aligned} \tag{4.2}$$

želimo simulirati pri konstantah $a_{11} = 5, a_{12} = 0.05, a_{21} = 0.0004, a_{22} = 0.2$ in pri začetnem številu zajcev in lisic $x_{10} = 520, x_{20} = 85$. V pomoč nam bo simulacijska shema, ki jo prikazuje slika 3.9 in enačbe (4.2).

Preden se lotimo pisanja programa, določimo vsem spremenljivkam in konstantam modela imena, ki jih bomo uporabljali v računalniškem programu. Smiselno je izbrati taka imena, ki so povezana z realnim problemom, oz. s fizikalnimi zakonitostmi. Z ozirom na oznake, ki jih uporabljamo v shemi na sliki 3.9 izberemo naslednja imena

x_1	`RAB`	$\dot{x}_1$	`RABDOT`
x_2	`FOX`	$\dot{x}_2$	`FOXDOT`
x_{10}	`RAB0`	x_{20}	`FOX0`
a_{11}	`A11`	a_{12}	`A12`
a_{21}	`A21`	a_{22}	`A22`

Simulacijski program sestoji iz stavkov. Vrstni red stavkov je v jezikih tipa CSSL običajno sicer poljuben, toda že zaradi boljše preglednosti programa je priporočljiv naslednji vrstni red:

1. stavki, ki opisujejo parametre modela,
2. stavki, ki opisujejo strukturo,
3. stavki, ki definirajo krmilne parametre simulacije.

Prvi stavek v simulacijskem programu je lahko stavek `PROGRAM`, s katerim damo programu želeno ime.

```
PROGRAM PREY_AND_PREDATOR
```

Nato definiramo konstante simulacijskega modela:

```
"konstante modela
   CONSTANT A11=5,A12=0.05,A21=0.0004,A22=0.2
   CONSTANT RAB0=520,FOX0=85
```

Vrstico s komentarjem uvedemo z znakom ". Komentarji povečajo dokumentacijsko uporabnost simulacijskega programa.

Nato opišemo strukturo modela s stavki, ki imajo poljuben vrstni red. Pri tem niti ni potrebno uporabiti simulacijske sheme na sliki 3.9, ampak lahko zaradi enačbne

orientiranosti jezika tipa CSSL direktno uporabimo enačbi (3.8), ki specificirata oba odvoda modela.

```
RABDOT=A11*RAB-A12*RAB*FOX
FOXDOT=A21*RAB*FOX-A22*FOX
```

Da generiramo rešitvi obravnavanega modela, je potrebno oba odvoda integrirati

```
RAB=INTEG(RABDOT,RAB0)
FOX=INTEG(FOXDOT,FOX0)
```

Prvi argument stavkov `INTEG` je vhod v integrator, t.j. odvod, drugi parameter pa je začetni pogoj (začetno število zajcev in lisic).

Potem, ko smo opisali strukturo modela, je potrebno definirati še krmilne parametre simulacije. Potrebno je izbrati ustrezen čas opazovanja, t.j. dolžino simulacijskega teka. Ker so v konstante kot časovne enote vključena leta ($a_{11} = 5$ je število potomcev enega zajca v enem letu), je enota neodvisne spremenljivke eno leto. Ker želimo opazovati časovne poteke vsaj eno ekološko periodo, ki je v tem primeru približno sedem let, izberemo dolžino simulacijskega teka deset let. Pogoj za končanje simulacije podamo v stavku `TERMT`

```
TERMT T.GE.10
```

kar pomeni, da je pogoj za končanje simulacije izpolnjen, če je čas simulacije enak, ali pa presega deset let. Če pa imamo namen, da bomo med simulacijo interaktivno spreminjali pogoj za končanje simulacijskega teka, pa je smiselno, da v stavek `TERMT` uvedemo spremenljivko, katere vrednost inicializiramo s stavkom `CONSTANT`

```
CONSTANT TFIN = 10
TERMT T.GE.TFIN
```

Trenutke, v katerih uporabnik dobi rezultate simulacije, pa definiramo s stavkom

```
CINTERVAL CI=0.01
```

kar pomeni en rezultat na 0.01 leta oz. na 3.65 dni. Končno izberemo še spremenljivke, ki naj se med simulacijo izpisujejo na zaslon (stavek `OUTPUT`) in spremenljivke, ki naj se vpisujejo v datoteko (stavek `PREPAR`) za nadaljnjo obdelavo

```
   OUTPUT 100,RAB,FOX
   PREPAR 2,RAB,FOX
```

Številka 100 v prvem stavku pove, da naj se izpiše le vsak stoti rezultat (en rezultat v enem letu), številka dva v drugem stavku pa pomeni, da se v datoteko rezultatov vpiše vsak drugi rezultat (t.j. vsakih 7.3 dni). Simulacijski program zaključimo s stavkom `END`

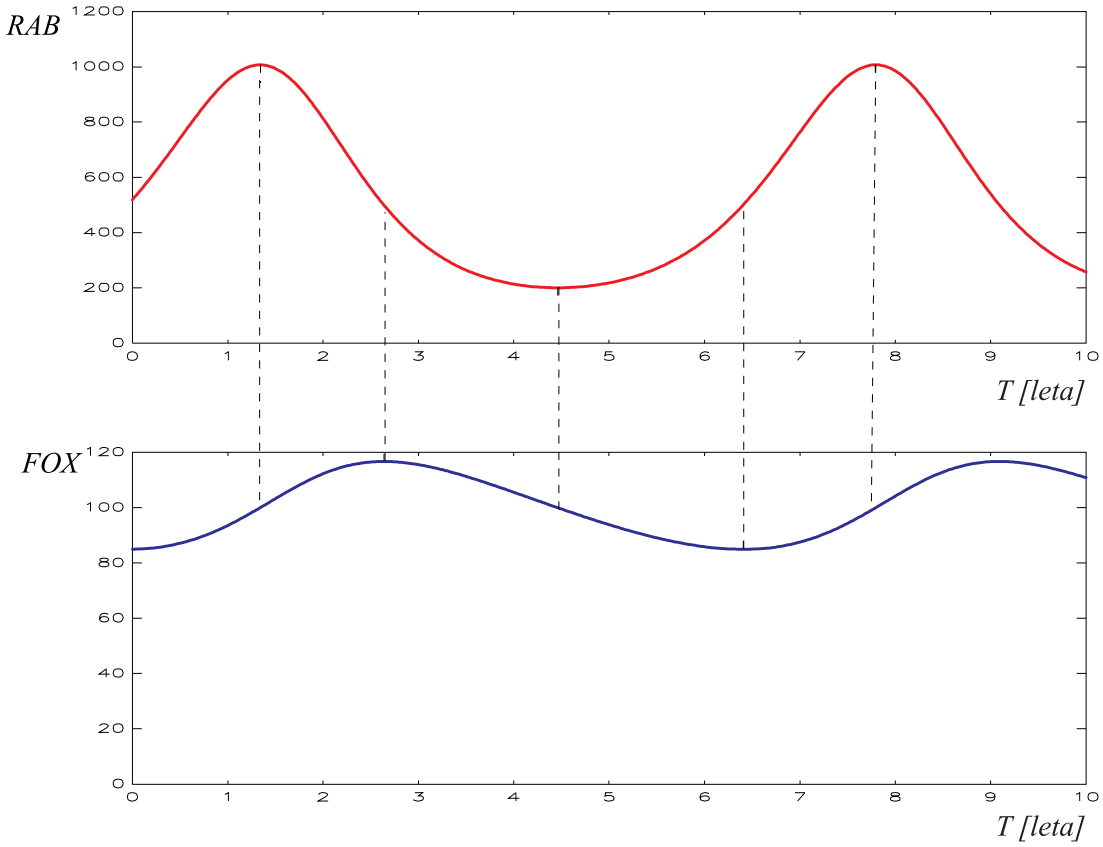
```
END
```

CSSL jeziki so prevajalniško orientirani. Simulacijski program se najprej prevede v module v splošnonamenskem jeziku (npr. FORTRAN) in v ustrezne podatkovne datoteke. Po prevajanju in povezovanju v splošnonamenskem jeziku dobimo program za simulacijo. Ko izvedemo program, se na zaslonu izpisujejo naslednje vrednosti: vrednost časa, število zajcev in število lisic. Rezultate si lahko ogledamo tudi v grafični obliki po simulacijskem teku. Celotni simulacijski program je naslednji:

```
PROGRAM PREY_AND_PREDATOR
"-------------------------------------------------
 " konstante modela
   CONSTANT A11=5,A12=0.05,A21=0.0004,A22=0.2
   CONSTANT RAB0=520,FOX0=85
"-------------------------------------------------
 " struktura modela
       RABDOT=A11*RAB-A12*RAB*FOX
       FOXDOT=A21*RAB*FOX-A22*FOX
       RAB=INTEG(RABDOT,RAB0)
       FOX=INTEG(FOXDOT,FOX0)
"-------------------------------------------------
 "dolzina simulacijskega teka
   CONSTANT TFIN = 10
   TERMT(T.GE.TFIN)
"-------------------------------------------------
```

```
 "dolzina komunikacijskega intervala
   CINTERVAL CI=0.01
"-------------------------------------------------
 "izhodne zahteve
   OUTPUT 100, RAB,FOX
   PREPAR 2,RAB,FOX
"-------------------------------------------------
END
```

Slika 4.5 prikazuje populaciji zajcev in lisic.



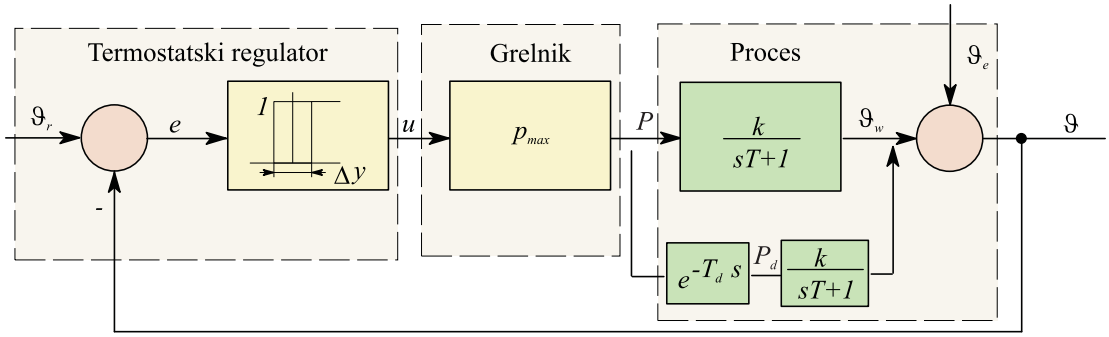
Slika 4.5: Časovna poteka števila zajcev in lisic

Opazimo, da je ekološka perioda približno sedem let, da število lisic najhitreje narašča ob največjem številu zajcev in najhitreje upada ob najmanjšem številu zajcev. □

Primer 4.2 Model regulacije ogrevanja prostora

Primer omogoča nekoliko globlji vpogled v jezike tipa CSSL in nakazuje tudi značilne korake, ki so potrebni pri analizi nekega problema s simulacijo.

Temperaturni regulacijski sistem smo uvedli v primeru 1.2. Ustrezen bločni diagram prikazuje slika 4.6.



Slika 4.6: Bločni diagram temperaturnega regulacijskega sistema

Slika 4.6 nakazuje dve možni obravnavi. Najprej bomo upoštevali model procesa brez mrtvega časa, nato pa še z mrtvim časom. Enačba

$$\dot{\vartheta}_w + \frac{1}{T}\vartheta_w = \frac{k}{T}p \tag{4.3}$$

opisuje model procesa v delovni točki, če ne upoštevamo mrtvega časa. Podatki za simulacijo so naslednji: širina histereze je $\Delta y = 1^oC$, moč grela je $p = p_{max} = 5kW$, temperatura okolice je $\vartheta_e = 15^oC$, ojačenje procesa je $k = 2^oC/KW$, časovna konstanta procesa je $T = 1h$, začetna temperatura v prostoru je $\vartheta(0) = 16^oC$ oz. $\vartheta_w(0) = 1^oC$. Referenčna temperatura je časovno spremenljiva funkcija in jo opisuje tabela 4.2.

Tabela 4.2: Želena sobna temperatura

t[h]	0	5.99	6	8.99	9	14.99	15	20.99	21
$\vartheta_r[^oC]$	15	15	20	20	18	18	20	20	15

Na začetku ponovno izberemo ustrezna imena, ki jih bomo uporabljali v programu:

ϑ_r	THR	ϑ_e	THE
ϑ	TH	$\vartheta_w(0)$	THWO
$\dot{\vartheta}_w$	THWD	ϑ_w	THW
e	E	p	P
p_{max}	PMAX	Δy	DELTAY
u	U	T	TIMCON
k	GAIN		

Na začetku programa običajno definiramo parametre modela, t.j. konstante in morebitne funkcijske generatorje

```
"konstante modela
   CONSTANT DELTAY=1,PMAX=5,GAIN=2,TIMCON=1
   CONSTANT THE=15,THWO=1
"tocke funkcijskega generatorja
   TABLE REF,1,9,...
     0.,  5.99, 6.,  8.99, 9., 14.99, 15., 20.99, 21.,...
    15., 15.,  20., 20.,  18., 18.,   20., 20.,   15.
```

Funkcijski generator definiramo z imenom (REF), s številom neodvisnih spremenljivk (1), s številom točk, v katerih je podana funkcija (9) ter z vrednostmi neodvisne (naslednjih 9 podatkov) in odvisne spremenljivke (zadnjih 9 podatkov).

Nato je potrebno definirati strukturo modela. Vrstni red stavkov je povsem poljuben. Referenčni signal v regulacijskem sistemu realiziramo s klicem funkcije za realizacijo funkcijskega generatorja

```
THR=REF(T)
```

kjer je T neodvisna spremenljivka (čas) in REF ime funkcijskega generatorja, ki smo ga definirali s stavkom TABLE.

Pogrešek v regulacijskem sistemu definiramo s stavkom

```
E=THR-TH
```

Vodimo ga v blok, ki modelira stopenjski (ON-OFF) termostatski regulator. Ustrezni model podaja histerezna funkcija s histerezno širino Δy, spodnjim nivojem nič in zgornjim nivojem ena.

```
CONSTANT STATE =0.
U=HSTRSS(E,-DELTAY/2.,DELTAY/2.,0.,1.,STATE)
```

Zadnji parameter v funkciji `HSTRSS` je začetni pogoj, ki določa izhodno vrednost v primeru, če ob prvem klicu pogrešek e leži v področju $-\frac{\Delta y}{2} \leq e \leq \frac{\Delta y}{2}$.

Grelo modeliramo s pomočjo ojačevalnega bloka

```
P=PMAX*U
```

Proces pa simuliramo z indirektno metodo s pomočjo simulacijske sheme, ki jo prikazuje slika 3.5 ali pa s pomočjo enačbe (4.3 oz. enačbe 3.4)

```
THWD=-1./TIMCON*THW+GAIN/TIMCON*P
THW=INTEG(THWD,THW0)
```

Drugi parameter stavka `INTEG` je začetni pogoj modela v delovni točki. Absolutno temperaturo pa dobimo tako, da temperaturi, ki jo daje model v delovni točki, prištejemo temperaturo okolice.

```
TH=THW+THE
```

Po opisu strukture moramo definirati še krmilne parametre simulacije. Ker želimo opazovati časovne poteke v teku enega dneva, je pogoj za končanje simulacije

```
CONSTANT TFIN=24
TERMT T.GT.TFIN
```

Integracija je osrednji postopek vsakega simulacijskega sistema. Dobri simulacijski sistemi imajo več integracijskih postopkov. Če ga posebej ne navedemo, se uporabi privzeta metoda (običajno Runge-Kutta s prilagodljivim računskim korakom). V modelih, ki vsebujejo histerezno funkcijo, pa je priporočljivo uporabiti metodo s konstantnim računskim korakom. To dosežemo s stavkom

```
ALGORITHM IALGOR=1,JALGOR=5
```

Prvi parameter (`IALGOR=1`) se uporablja za izbiro metode za inicializacijo integracijskega postopka, drugi parameter (`JALGOR=5`) pa za izbiro integracijskega postopka.

Komunikacijski interval izberemo s stavkom

```
CINTERVAL CI=0.02
```

kar pomeni, da ima uporabnik na voljo rezultate simulacije vsake 0.02 h (1.2 min). Ponovno naj poudarimo, da morajo biti enote konsistentne. Čas mora biti povsod izražen v enaki enoti (npr. konstante modela, dolžina simulacijskega teka, komunikacijski interval). Če stavka `CINTERVAL` ne uporabljamo, se uporabi privzeta vrednost (1) komunikacijskega intervala. Končno definiramo še spremenljivke, ki se med simulacijo izpisujejo na zaslon in v datoteko

```
OUTPUT 10,THR,P,TH
PREPAR THR,P,TH
```

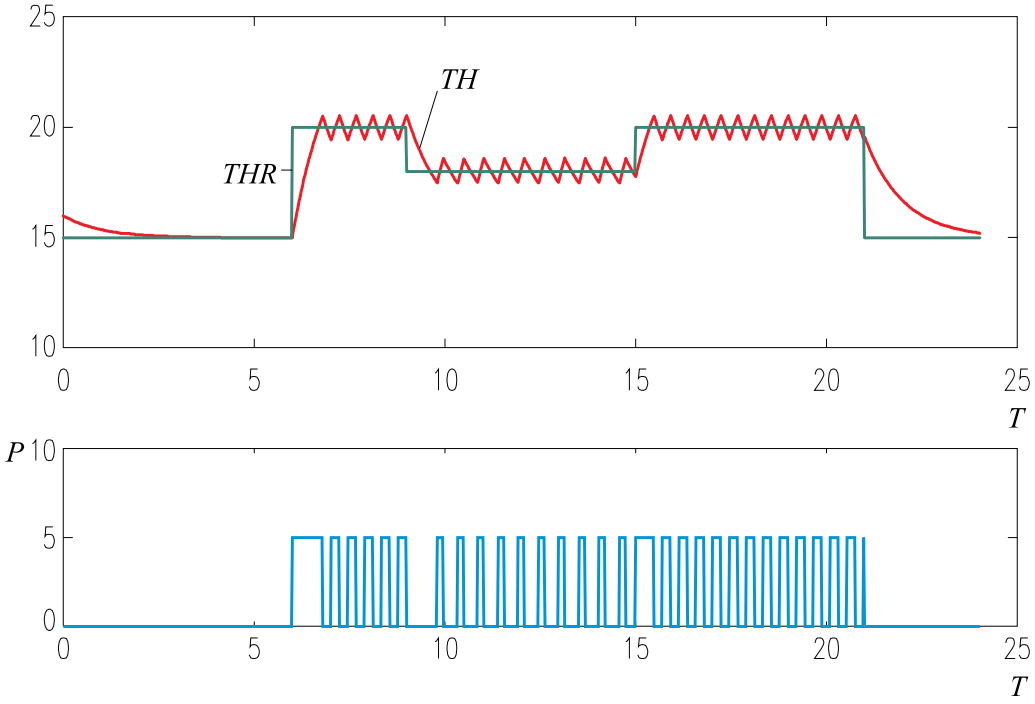
Simulacijski program zaključimo s stavkom

```
END
```

Po procesiranju lahko uporabnik izvede simulacijski tek. Slika 4.7 prikazuje rezultate simulacije (temperaturo v prostoru ϑ in želeno temperaturo ϑ_r v zgornjem in moč grela p v spodnjem diagramu). Grelo se vključuje približno na 30 min, temperatura v prostoru pa niha v pasu 1^oC. Če podvojimo velikost histereze (2^oC), dobimo rezultate, ki jih prikazuje slika 4.8. Nihanja temperature postanejo v tem primeru prevelika za ugodno počutje v prostoru. Perioda preklapljanja grela pa se poveča na približno 50 min.

Če je termostatsko tipalo precej oddaljeno od grela, je smiselno v model ogrevanja vključiti mrtvi čas. Ustrezen bločni diagram je razviden iz slike 4.6. V ta namen vpeljemo dve novi spremenljivki

p_d	`PD`
T_d	`TDELAY`



Slika 4.7: Rezultati temperaturnega regulacijskega problema

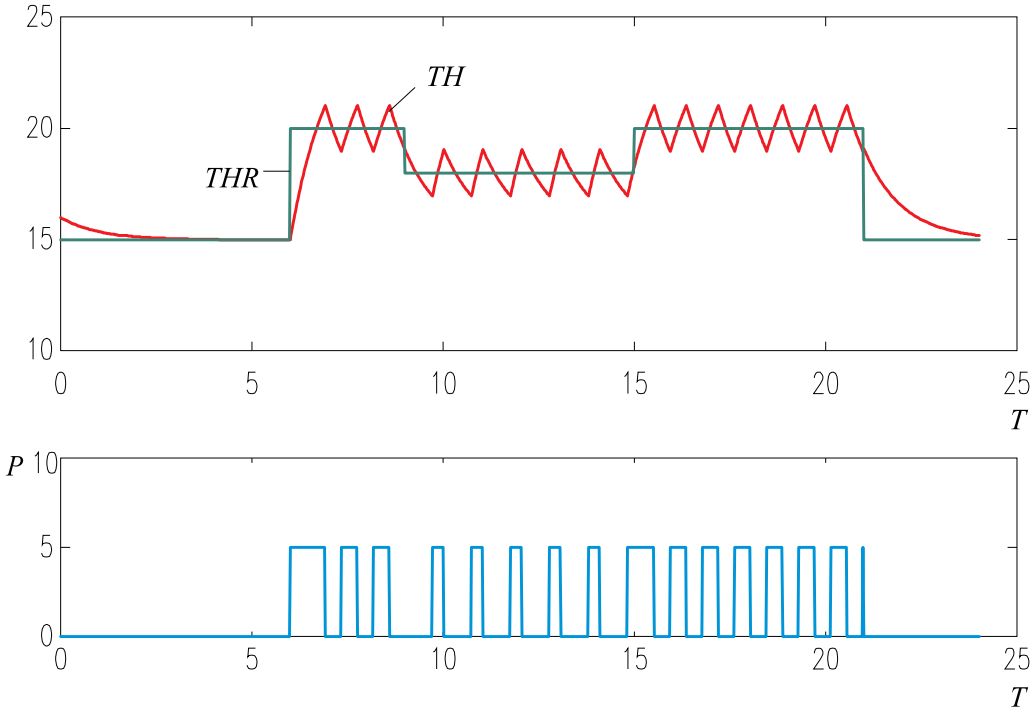
Mrtvi čas realiziramo s klicem vgrajene funkcije `DELAY`

```
CONSTANT TDELAY=0.1,WORK=50*0
ARRAY WORK(50)
    PD = DELAY(P,TDELAY,WORK,CI)
```

Prvi parameter v klicnem stavku je spremenljivka, ki jo želimo zakasniti. Drugi parameter je mrtvi čas, tretji parameter pa je delovno polje, ki se uporablja za realizacijo mrtvega časa. Za delovno polje je potrebno rezervirati dimenzije (s stavkom `ARRAY`) in določiti njegove začetne vrednosti. Četrti parameter je čas vzorčenja, ki določa trenutke, v katerih se v delovnem polju (premikalnem registru) izvedejo pomiki. Čim manjši je čas vzorčenja, boljši približek idealnemu zveznemu mrtvemu času dosežemo. Zato smo izbrali minimalni možni čas vzorčenja, ki je enak komunikacijskemu intervalu `CI`.

Ker sedaj spremenljivka p_d predstavlja vhod v proces, moramo modelno enačbo

```
THWD=-1./TIMCON*THW+GAIN/TIMCON*P
```



Slika 4.8: Rezultati simulacije pri podvojeni širini histereze

zamenjati z enačbo

```
THWD=-1./TIMCON*THW+GAIN/TIMCON*PD
```

Celoten simulacijski program je naslednji:

```
"-------------------------------------------------
"TEMPERATURNI REGULACIJSKI SISTEM
"-------------------------------------------------
"konstante modela
   CONSTANT DELTAY=1,PMAX=5,GAIN=2,TIMCON=1
   CONSTANT THE=15,TDELAY=0.1,THW0=1
   CONSTANT WORK=50*0,STATE=0
"tocke funkcijskega generatorja
   TABLE REF,1,9,...
     0.,  5.99, 6.,  8.99, 9., 14.99, 15., 20.99, 21.,...
    15., 15.,  20., 20.,  18., 18.,   20., 20.,   15.
```

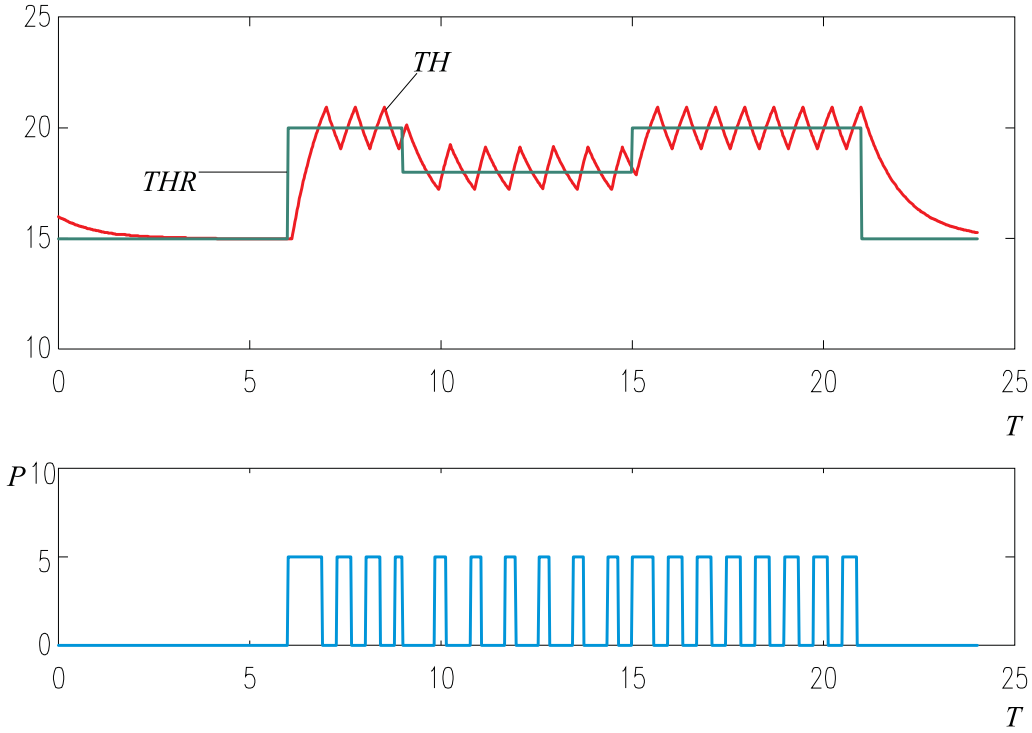
```
"-------------------------------------------------
"delovno polje za realizacijo mrtvega casa
   ARRAY WORK(50)
"-------------------------------------------------

"struktura modela
       THR=REF(T)
       E=THR-TH
       U=HSTRSS(E,-DELTAY/2.,DELTAY/2.,0.,1.,STATE)
       P=PMAX*U
       PD=DELAY(P,TDELAY,WORK,CI)
       THWD=-1./TIMCON*THW+GAIN/TIMCON*PD
       THW=INTEG(THWD,THW0)
       TH=THW+THE
"-------------------------------------------------

"trajanje simulacijskega teka
   CONSTANT TFIN=24
   TERMT T.GT.TFIN
"-------------------------------------------------
"izbira integracijske metode
   ALGORITHM IALGOR=1,JALGOR=5
"dolocitev komunikacijskega intervala
   CINTERVAL CI=0.02
"-------------------------------------------------
"zahteve za spremljanje rezultatov
   HDR HEATING CONTROL SYSTEM
   OUTPUT 10,THR,P,TH
   PREPAR THR,P,TH
"-------------------------------------------------
   END
```

Slika 4.9 prikazuje rezultate simulacije pri mrtvem času $T_d = 0.1\ h$. Če primerjamo rezultate simulacije z rezultati na sliki 4.7, vidimo, da temperatura niha v širšem pasu (približno 2^oC). Nihanje temperature je torej odvisno od širine histereze in od zakasnitev procesa. Očitno je, da je termostatsko tipalo predaleč od grela, da bi lahko dosegli ugodno temperaturo.

Ko dobimo preveden simulacijski program, lahko eksperimentiramo pri različnih vrednostih konstant, ne da bi bilo potrebno spreminjati in ponovno procesirati

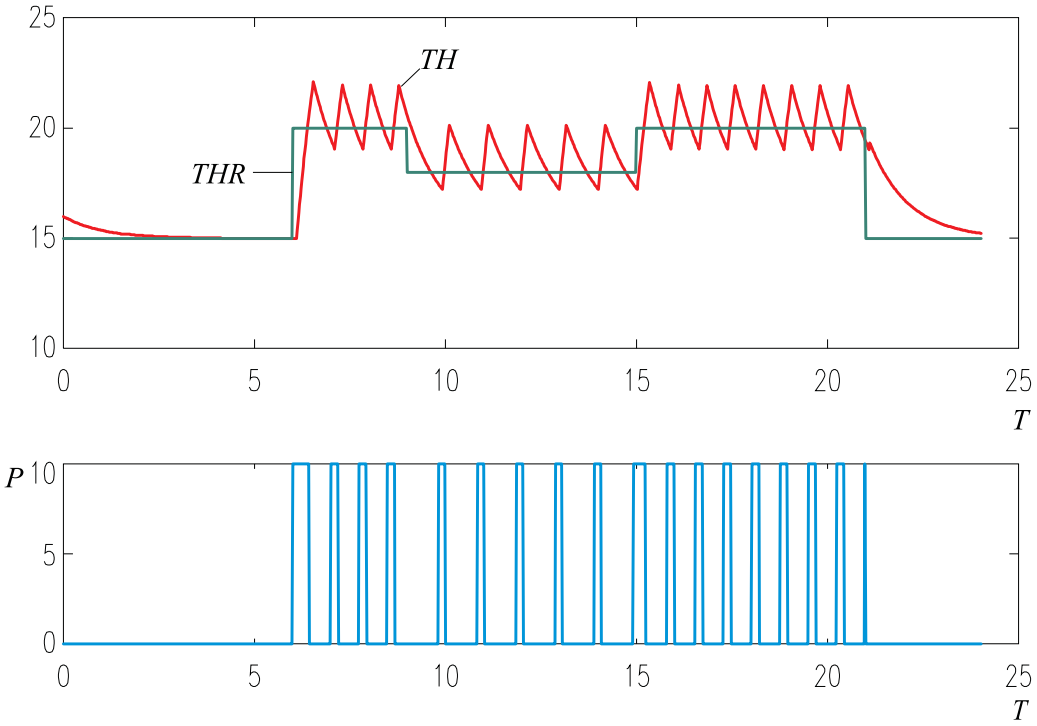


Slika 4.9: Rezultati pri dodatnem mrtvem času

izvorni program. Če želimo npr. spremeniti moč grela, interaktivno spremenimo konstanto `PMAX`. Slika 4.10 prikazuje rezultate simulacije pri 10 kW grelu. Opazimo, da temperatura narašča hitreje, vendar so nihanja temperature večja kot v primeru na sliki 4.9 (približno 3^oC). Vpliv zunanje temperature proučujemo s spreminjanjem konstante `THE`. Slika 4.11 prikazuje rezultate, če je zunanja temperatura 17^oC. Razumljivo je, da je nemogoče doseči referenčno temperaturo, ki je nižja od temperature okolice, v kolikor ne uporabljamo hlajenja.

Še bolj realno sliko dogajanja običajno dosežemo z modelom, ki upošteva različni časovni konstanti v fazah ogrevanja in ohlajanja. Blok `PROCEDURAL` predstavlja učinkovito možnost za vključitev nelinearnosti (preklapljanje med dvema časovnima konstantama). V našem primeru fazo ogrevanja in fazo ohlajanja natančno določa odvod temperature v prostoru. Konstanto T=1 med ogrevanjem ($\dot{\vartheta}_w \geq 0$) in T = 4 med ohlajanjem ($\dot{\vartheta}_w < 0$) realiziramo tako, da definicijo konstante

```
CONSTANT TIMCON=1
```



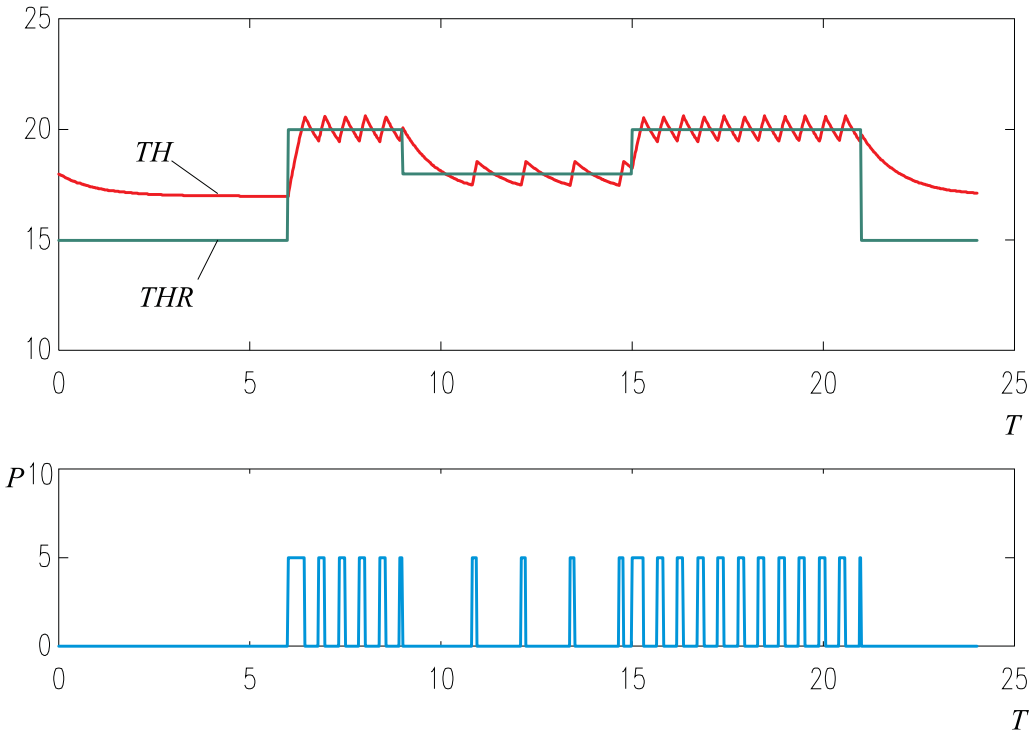
Slika 4.10: Rezultati simulacije pri 10 kW grelu

zamenjamo z blokom, katerega vhod je spremenljivka $\dot{\vartheta}_w$, izhod pa časovna konstanta T. Blok prikazuje slika 4.12.

Blok `PROCEDURAL` opišemo z naslednjim delom programa:

```
CONSTANT T1=1, T2=4
PROCEDURAL (TIMCON = THWD)
        TIMCON = T1
        IF (THWD.LT.0) TIMCON = T2
END
```

Blok `PROCEDURAL` vpeljemo s stavkom `PROCEDURAL`, ki vsebuje v oklepaju spisek izhodnih spremenljivk (levo od enačaja) in spisek vhodnih spremenljivk (desno od enačaja). Na žalost pa taka realizacija povzroči t.i. algebrajsko zanko, kajti za izračun spremenljivke $\dot{\vartheta}_w$ mora biti dana časovna konstanta T. Da pa bi lahko izračunali T, mora biti dan temperaturni odvod $\dot{\vartheta}_w$. Algebrajsko zanko prikazuje slika 4.13.



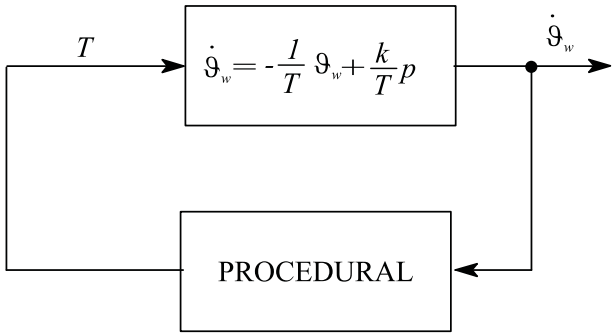
Slika 4.11: Rezultati simulacije pri višji temperaturi okolice

Slika 4.12: Blok `PROCEDURAL` za določitev konstante T

Nekateri simulacijski jeziki vsebujejo algoritme za numerično reševanje algebrajske zanke, vendar je taka simulacija zelo počasna.

V našem primeru pa lahko enoumno določimo fazo ogrevanja in ohlajanja s pomočjo spremenljivke p_d, saj pozitivna vrednost spremenljivke p_d vedno pomeni naraščanje temperature. Pri problemu brez dodatnega mrtvega časa pa lahko uporabimo kar regulirno veličino u.

```
CONSTANT T1=1, T2=4
PROCEDURAL (TIMCON = U)
       TIMCON = T1
       IF (U.LE.0) TIMCON = T2
```



Slika 4.13: Algebrajska zanka

```
END
```

□

Ekološki sistem roparjev in žrtev in temperaturni regulacijski sistem smo simulirali s simulacijskim jezikom SIMCOS (Zupančič, 1992). Vendar bi bili programi v katerem koli drugem jeziku, ki spoštuje standard CSSL (npr. CSSL IV, ACSL) skoraj identični.

4.2.6 Primeri uporabe bločno orientiranega orodja z grafičnim vnosom modela - Matlab-Simulink

Uporabnost bločno orientiranih simulacijskih jezikov bomo prikazali na razvoju treh simulacijskih programov -model ekološkega sistema roparjev in žrtev, regulacijski sistem ogrevanja prostora in model avtomobilskega vzmetenja bomo simulirali v okolju Matlab-Simulink.

Primer 4.3 Ekološki sistem roparjev in žrtev

Primer 3.3 obravnava razvoj simulacijske sheme modela ekološkega sistema roparjev in žrtev, ki je predhodno opisan v Primeru 1.3. Model želimo simulirati pri konstantah $a_{11} = 5, a_{12} = 0.05, a_{21} = 0.0004, a_{22} = 0.2$ in pri začetnem številu zajcev in lisic $x_{10} = 520, x_{20} = 85$. V pomoč nam bo simulacijska shema, ki jo prikazuje slika 3.9 in enačbe (3.8). Ker okolje Matlab-Simulink zahteva vnos modela na grafični, bločno orientirani način, prenesemo na zaslon vse potrebne bloke,

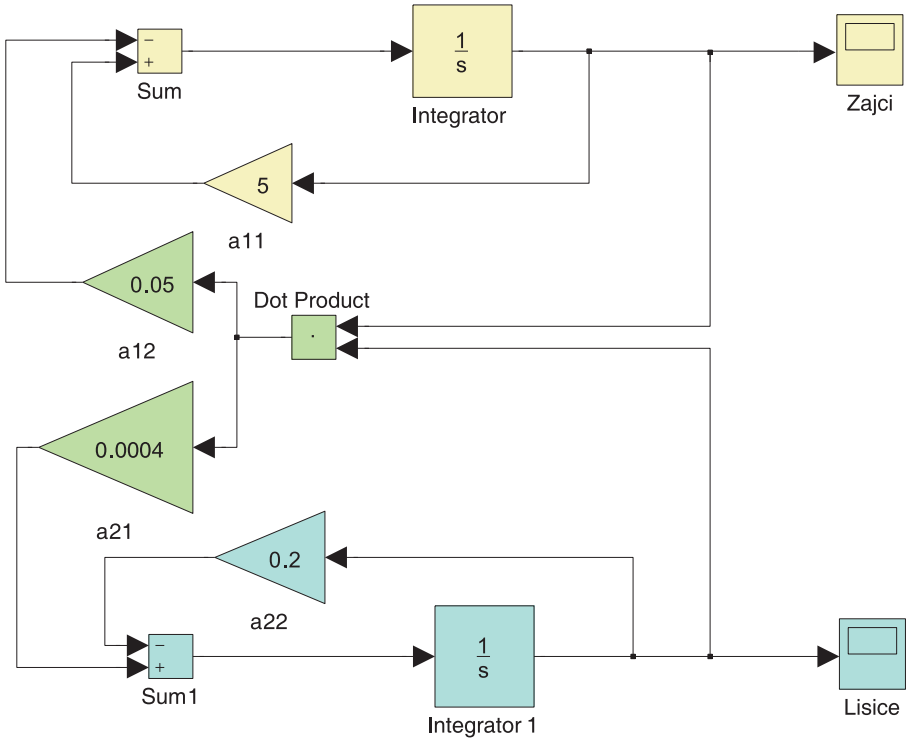
ki jih vsebuje slika 3.9. Potrebujemo dva integratorja (na zaslon jih povlečemo iz knjižničnega okna Continuous, privzeti imeni sta `Integrator` in `Integrator1`), štiri ojačevalne bloke (privzeta imena so `Gain`, `Gain1`, `Gain2` in `Gain3`, a smo ta imena v shemi nadomestili z imeni konstant `a11`, `a12`, `a21` in `a22`), dva sumatorja (privzeti imeni `Sum` in `Sum1`) in en množilnik (`DotProduct`). Vse te bloke prenesemo iz knjižničnega okna Math Operations. Na koncu iz knjižničnega okna Sinks dodamo dva prikazovalna bloka (privzeti imeni `Scope` in `Scope1` smo zamenjali z besedama `Zajci` in `Lisice`). Bloke razporedimo tako, kot kaže slika 3.9 in jih nato ustrezno povežemo (z miško potegnemo od izhoda bloka proti ustreznemu vhodu bloka). Ko je shema povezana, nastavimo parametre blokov z dvojnim klikom, ki odpre uporabniški vmesnik ustreznega bloka. Ojačevalnim blokom podamo vrednost ojačenja, integratorjema pa začetna pogoja. Ker je problem elementaren, ni potrebno nastaviti nobenih simulacijskih parametrov razen trajanja simulacijskega teka - vrednost 10 nastavimo v posebnem okencu v opravilni vrstici desno zgoraj. V kakšnih enotah je neodvisna spremenljivka, lahko ve le uporabnik, ki je problem tudi modeliral. Spomnimo se, da je enota neodvisne spremenljivke eno leto, torej traja simulacija deset let. Rezultati simulacije so enaki, kot na sliki 4.5. Opazimo, da je ekološka perioda približno sedem let, da število lisic najhitreje narašča ob največjem številu zajcev in najhitreje upada ob najmanjšem številu zajcev. Simulink shemo prikazuje slika 4.14.

□

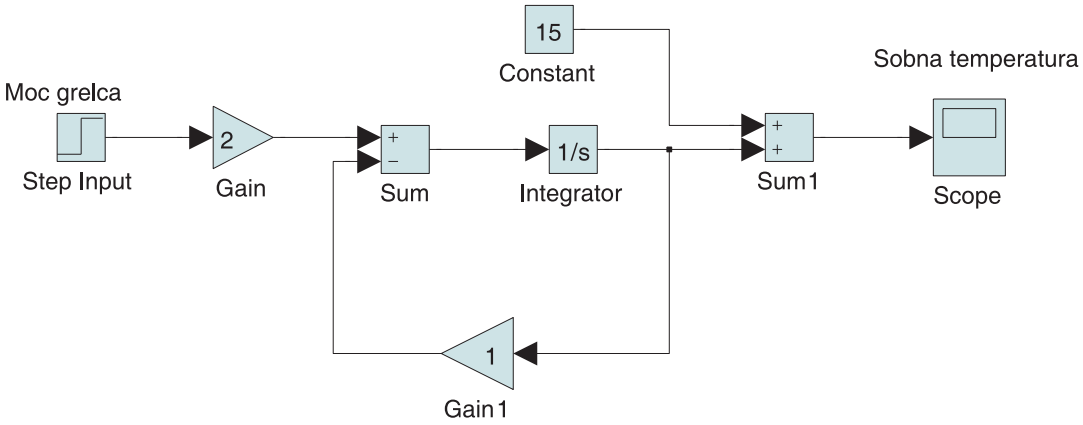
Primer 4.4 Regulacijski sistem ogrevanja prostora

Primer 3.1 obravnava razvoj simulacijske sheme temperaturnega procesa, ki je predhodno opisan v Primeru 1.2. Simulacijsko shemo razvijemo s pomočjo slike 3.5. Potrebujemo sumator, integrator in dva ojačevalna bloka. Zaradi veljavnosti enačbe (1.14) smo dodali še en sumator, s pomočjo katerega smo iz temperature v delovni točki in temperature okolice izračunali absolutno temperaturo. Simulink shemo prikazuje slika 4.15.

Simulacijski model procesa nadgradimo v shemo za regulacijo po sliki 1.6. Oporabimo blok `Signal builder` iz knjižnice Sources za generacijo spremenljivega referenčnega signala, ki ga opisuje tabela 4.3. Z blokom `Signal builder` opišemo referenčni signal s pomočjo grafičnega uporabniškega vmesnika. Iz knjižnice Discontinuous uporabimo `Relay` blok (privzeto ime smo nadomestili z imenom `Controller`). Bloku podamo vrednosti, pri katerih vklopi oz. izklopi ter vklopno in izklopno vrednost. Grelo simuliramo z ojačevalnim blokom `pmax`, dodatno zakasnitev pa z blokom `Transport Delay` iz knjižnice Continuous. Simulink shemo



Slika 4.14: Simulacijska shema v Simulinku za ekološki primer zajcev in lisic



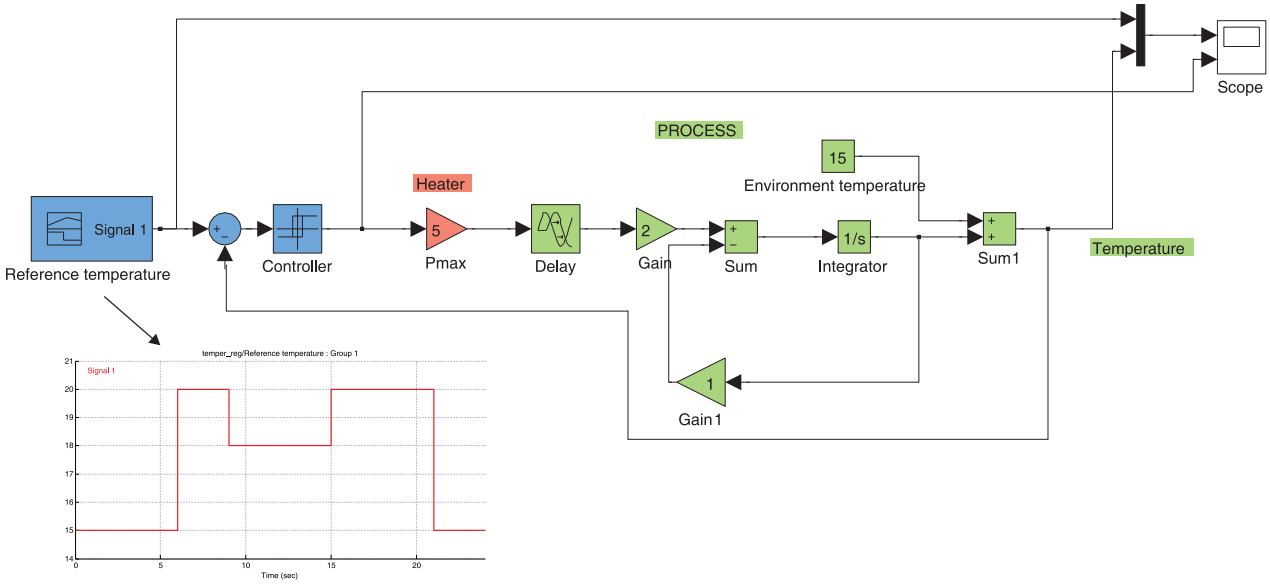
Slika 4.15: Simulacijska shema v Simulinku za model ogrevanja prostora

prikazuje slika 4.16.

Temperatura v prostoru ter vklapljanje in izklapljanje grela sta prikazana na sliki 4.7. □

Tabela 4.3: Želena sobna temperatura

t[h]	0	6	6	9	9	15	15	21	21
$\vartheta_r[^oC]$	15	15	20	20	18	18	20	20	15

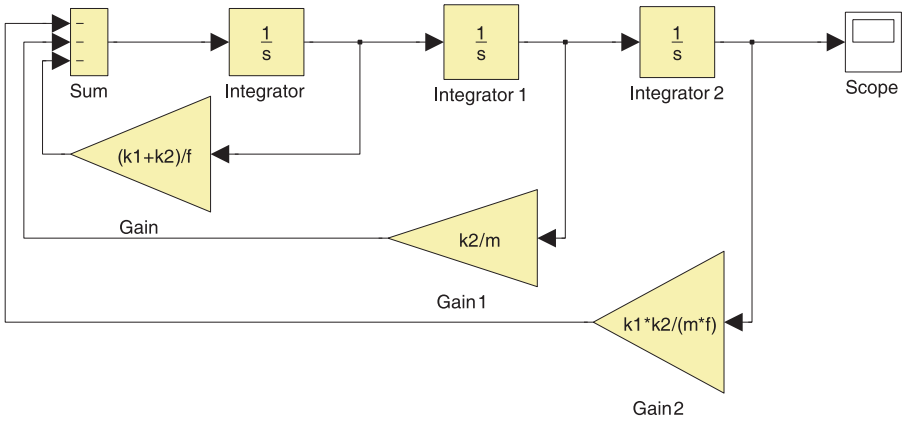


Slika 4.16: Simulacijaska shema regulacije temperature v prostoru in prikaz referenčnega signala

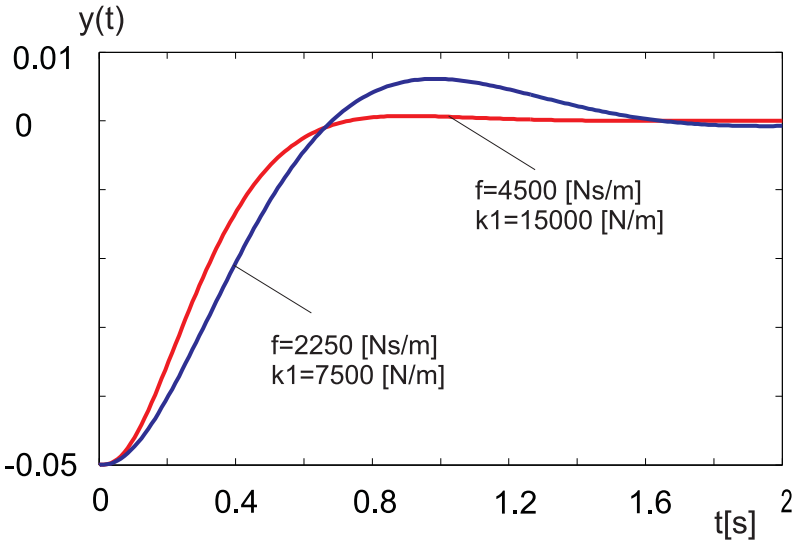
Primer 4.5 Avtomobilsko vzmetenje

Primer 3.2 obravnava razvoj simulacijske sheme avtomobilskega vzmetenja, ki je predhodno opisan v Primeru 1.1. Simulacijsko shemo razvijemo s pomočjo slike 3.7. Potrebujemo tri integratorje, tri ojačevalne bloke in sumator. Tokrat v ojačevalne bloke ne vpišemo številskih vrednosti konstant ampak kar aritmetične izraze, ki se prikažejo tudi znotraj blokovih ikon, če le te dovolj povečamo. Vrednosti za `k1`, `k2`, `m` in `f` pa podamo v komandnem oknu Matlaba. Simulacijsko shemo prikazuje slika 4.17.

Opazujemo pomik karoserije pri negativnem začetnem pogoju `y(0)=-0.05 m`. Praktično to lahko pomeni, da se je voznik vsedel v avtomobil in v trenutku t=0 izstopil iz vozila. Opazujemo, kako se karoserija vrne v mirovno lego. Rezultate simulacije prikazuje slika 4.18.



Slika 4.17: Simulink shema modela avtomobilskega vzmetenja



Slika 4.18: Pomik karoserije

Izhodiščni podatki za študijo so bili: m=500 kg, k1=7500 N/m, k2=150000 N/m, f=2250 Ns/m. Ker je prišlo do premalo dušenega odziva, smo povečali togost vzmeti na k1=15000 N/m (2x povečanje), dušenje pa smo tudi 2x povečali (f=4500 Ns/m). Tako smo dobili optimalnejše delovanje avtomobilskega vzmetenja.

□

Poglavje 5

Jeziki za simulacijo zveznih dinamičnih sistemov

V poglavju bomo podali pregled razvoja simulacijskih jezikov, programsko zgradbo simulacijskih jezikov ter opisali možnosti simulacije z enačbno in bločno orientiranimi simulacijskimi jeziki ter s splošnonamenskimi programskimi jeziki.

5.1 Pregled razvoja simulacijskih jezikov

V razvoju jezikov za simulacijo dinamičnih sistemov je pomembno leto 1967, ko je bil sprejet standard za t.i. simulacijske jezike CSSL (Continuous System Simulation Language). Zato bomo pregled razdelili v obdobji pred in po letu 1967.

5.1.1 Razvoj simulacijskih jezikov pred sprejetjem standarda

Prve korake v digitalno simulacijo so naredili uporabniki analognih računalnikov. Na digitalnih računalnikih so želeli reševati probleme na podoben način kot na analognih. Zato so bili prvi jeziki podobni konceptom analognega računanja.

Za začetek razvoja digitalnih simulacijskih jezikov jemljemo leto 1955, ko je Selfridge prvi podal idejo o bločnem simulacijskem jeziku. Leta 1959 je tako prišel v uporabo prvi simulacijski jezik SELFRIDGE. Uporabnik je moral spremenljivke podajati s številkami, jezik ni vseboval vrstnega algoritma, za integracijo pa je bila uporabljena neprimerna Simpsonova metoda. Istega leta so Stein, Rose in Parker prvi podali idejo prevajalniškega simulacijskega jezika. Jezik MIDAS, ki so ga razvili pri IBM leta 1963 pa predstavlja prvi zares uporabni simulacijski sistem. Za spremenljivke in parametre je uporabljal alfanumerične oznake, vseboval je tudi vrstni algoritem. Za integracijo je uporabljal Milnejevo metodo 4. reda vrste prediktor-korektor s spremenljivim računskim korakom.

Od tedaj dalje lahko zasledujemo izredno hiter razvoj simulacijskih jezikov. V letu 1964 se pojavi jezik COBLOC (univerza Wisconsin) za računalnik CDC 1604. Vseboval je že tri integracijske algoritme in v nasprotju z dotedanjimi jeziki, ki so zahtevali točno določen format pisanja programa, omogoča že prosti format. Istega leta je prišel na trg tudi PACTOLUS za računalnik IBM 1620, ki je bil predhodnik enega kasneje najuspešnejših simulacijskih jezikov CSMP (Continuous System Modelling Program). Prva verzija jezika CSMP je bila prirejena za računalnik IBM 1130 (CSMP 1130). Za večje tipe računalnikov IBM 360/370 pa so leta 1965 predelali sicer že v letu 1959 razviti simulacijski jezik DYNAMO. Jezik je imel že dokaj sposobno diagnostiko, enostaven makro jezik, prikaz rezultatov v obliki tabel in grafov, odkril je celo algebrajske zanke. Vendar pa je imel le predpisani format podajanja modela. Za integracijo je uporabljal samo Eulerjev integracijski algoritem.

Zaradi precej omejenih sposobnosti takratnih računalnikov, pomanjkljivih programerskih izkušenj in zlasti neustreznih numeričnih metod so imeli omenjeni jeziki veliko slabosti:

- Bili so povsem bločno orientirani, zato je moral uporabnik izdelati program na osnovi bločne sheme podobno kot na analognem računalniku, odpadlo je le normiranje. Zato je bilo veliko možnosti za napake.
- Jeziki niso imeli možnosti za razširitve.
- Programi so bili nepregledni in zato manj primerni za dokumentiranje modela.
- Simulacija je bila numerično precej nezanesljiva.

Leta 1965 je IBM razvil jezik DSL 90 (Dynamic Simulation Language) za

računalnik IBM 7090. Le-ta predstavlja pomemben mejnik v razvoju simulacijskih jezikov, ker je prvi primer prevajalniškega, enačbno orientiranega jezika. Vseboval je tudi že t.i. blok PROCEDURE, s katerim je bilo možno blok modela opisati s stavki v jeziku FORTRAN. Iz jezika DSL 90 so leta 1967 razvili prvi sposobni simulacijski jezik za hitre in velike računalnike - CSMP 360. Le-ta je omogočal simulacijo kompleksnih sistemov. Istega leta se je pojavila še verzija CSMP III, ki je omogočala tudi uporabo grafičnega terminala IBM 2250. S to za takrat zelo moderno verzijo je IBM brez večjih modifikacij obvladal trg kar do leta 1978. Za oba jezika je značilno, da sta prevajalniškega tipa, ciljni jezik pa je FORTRAN. Uporabnik je na ta način pri simulaciji lahko uporabljal tudi vse zmožnosti jezika FORTRAN. Oba jezika sta vsebovala že funkcijske generatorje, makro jezik, možnost reševanja algebrajske zanke, sedem integracijskih metod in približno 60 operatorjev za opis modela. Verzije za manjše računalniške sisteme (IBM 1130, IBM 1180) pa so se imenovale DSL 1130/1180 in predstavljajo prve zametke interaktivnosti v simulaciji. S pomočjo funkcijskih stikal je bilo možno prekiniti simulacijski tek, spremeniti parametre in ponovno začeti s simulacijo. Na računalnik je bilo možno priključiti digitalni risalnik in ustrezni osciloskop.

Zaradi razvoja številnih simulacijskih jezikov so se sredi šestdesetih let pojavile težnje po standardizaciji jezikov za simulacijo zveznih dinamičnih sistemov. Jeziki so bili precej nezanesljivi, neprenosljivi in interaktivnost je bila prisotna le v zametkih. Njihova glavna slabost pa je bila, da so bili simulacijski modeli zaradi zelo različnih opisov povsem neprenosljivi. Zato je že leta 1965 Simulation Council (SCi), ki je bil predhodnik današnje Society for Computer Simulation (SCS) ustanovil standardizacijski komite, ki je leta 1967 izdelal predlog standarda (Strauss ,1967), ki je dobil popularno ime standard CSSL 67 (Continuous System Simulation Language). Na koncept tega standarda so precej vplivali do takrat razviti simulacijski jeziki, predvsem MIDAS in DSL 90.

Zaradi izredno dobre zasnove standarda, je le ta v uporabi kar dve desetletji in še danes so komercialno najuspešnejši jeziki osnovani na standardu CSSL 67.

5.1.2 Standard za zvezne simulacijske jezike

Standard CSSL 67 je imel tri glavne cilje:

- Zagotovil naj bi enostavno simulacijsko programsko opremo tudi za uporabnike brez večjih izkušenj. Zato so predvideli razumljivo sintakso za opis

diferencialnih enačb, blokov, diagnostiko napak, sortiranje blokov ter komande za interaktivno simulacijo.

- Izkušenemu uporabniku naj bi jezik predstavljal fleksibilno orodje za modeliranje in simulacijo večjih in kompleksnih sistemov. Zato so se odločili za prevajalniški tip jezika, kar daje uporabniku možnost, da lahko dodaja kompleksne operacije v ciljnem jeziku prevajalnika. To enostavno dodajanje novih operacij je omogočalo odprtost jezikov.
- Zaradi predvidenega tehnološkega razvoja (grafika, interaktivni računalniški sistemi) naj bi standard omogočal fleksibilne razširitve. To je kasneje omogočalo vključiti v jezike več novih lastnosti, kar je povzročilo, da je standard CSSL 67 ostal tako dolgo v veljavi. Zato pa so nastali tudi številni CSSL dialekti, kar je v osemdesetih letih pripeljalo do teženj po novem standardu.

Standard CSSL 67 vsebuje strukturne in funkcionalne elemente.

Strukturni elementi

Še preden so bili sploh znani pojmi strukturnega programiranja in podatkovnih struktur, je standard CSSL 67 že vseboval take elemente, kot so:

- Elementi, ki omogočajo razvrščanje (zaradi te lastnosti je program navidezno paralelen).
- Makro jezik, ki omogoča, da enake dele programa ne vnašamo večkrat.
- Modelne sekcije, ki lahko uporabljajo tudi različne integracijske metode.
- Strukturni opis modela s sekcijami INITIAL (proceduralni program, ki se izvrši pred simulacijskim tekom), DYNAMIC (neproceduralni del za opis modela) in TERMINAL (proceduralni program, ki se izvrši po simulaciji).
- Možnost za krmiljenje izvajanja simulacije. Standard je že predvidel dve možnosti: s programom v ciljnem jeziku prevajalnika, ki kliče simulacijski tek kot podprogram ali pa z uporabo interaktivnega komandnega jezika.

Te strukturne lastnosti so bile vsaj dve desetletji ustrezne, pozneje pa so začele danes utesnjevati jezike.

Funkcionalni elementi

Standard CSSL 67 je definiral samo osnovne funkcionalne elemente. Za opis modela so predvideli integrirni operator, diferencirni operator, implicitno reševanje algebrajske zanke, operator zakasnitve in nelinearnosti. Predvidene so bile konstante, navadne in indeksirane spremenljivke ter stavki za krmiljenje integracije (vrste metode, dolžina računskega koraka, dopustni pogreški) in simulacije (dolžina komunikacijskega intervala, dolžina simulacijskega teka, izbira načina prikaza rezultatov).

5.1.3 Razvoj simulacijskih jezikov po sprejetju standarda

Standard CSSL 67 je močno vplival na nadaljnji razvoj prevajalniško orientiranih simulacijskih jezikov. Jeziki, ki so se držali omenjenega standarda, so bili vse do danes tudi komercialno najuspešnejši.

Leta 1968 so pri podjetju CDC iz MIDASA razvili MIMIC, ki je predstavljal prvi simulacijski jezik po vzoru standarda CSSL 67. Vendar je jezik precej zaostajal za obstoječimi jeziki družine CSMP. Ni imel prostega formata za opis modela in diagnostika je bila zelo slaba. Jezik je vseboval t.i. logične krmilne spremenljivke, ki so omogočale izvajanje določenih delov modela ob izpolnitvi zahtevanih pogojev.

Leta 1969 pa je prišel v uporabo CSSL III, prvi jezik, ki je strogo upošteval navodila standarda. Ta jezik je skupno z nadaljnjimi verzijami (leta 1972 CSSL IV) naslednjih petnajst let močno vplival na razvoj digitalnih simulacijskih jezikov. Imel je prosti format za podajanje modela, bil je prevajalniški enačbni jezik in je vseboval veliko funkcij in operatorjev. Model je bilo možno podajati s strukturnimi sekcijami INITIAL, DYNAMIC in TERMINAL, možno pa je bilo uporabljati sposoben a uporabniško neprijazen makro jezik. Uporabnik je lahko izbiral med sedmimi numerično že dokaj robustnimi integracijskimi metodami, lahko pa je vključil tudi lastno metodo. Kasnejše verzije so dobile tudi komandni interaktivni jezik za krmiljenje simulacijskih tekov. Jezik je v začetku dobavljal CDC za svoje računalnike CDC 6000/7000, kasneje pa je razvoj prevzel Simulation Service. Zaradi velike sposobnosti in relativne enostavnosti za uporabo je CSSL IV kmalu dobil veliko privržencev. Jezik se je sicer nenehno razvijal, a le funkcionalno. Strukturno pa se še današnja verzija, ki predstavlja enega najsposobnejših simulacijskih jezikov, ne loči mnogo od prvih verzij (Nilsen, 1984).

Standard CSSL 67 je upošteval tudi simulacijski jezik SL-1 (l. 1970) za računalnik Sigma XDS. Zaradi vezanosti na eno vrsto računalnikov se ni dolgo uporabljal. Imel pa je nekaj posebnih lastnosti. Deloval je kot dvojni prevajalnik (v FORTRAN in nato v zbirnik). V različnih delih modela je bilo možno uporabiti različne integracijske postopke. Možno je bilo uporabiti tudi več neodvisnih spremenljivk, kar je omogočalo reševanje parcialnih diferencialnih enačb. Najbolj pa je zanimivo, da je jezik SL-1 omogočal sinhronizacijo z realnim časom, prekinitve ter uporabo pretvornikov A/D in D/A, torej simulacijo v realnem času. Bolj kot širša uporabnost tega jezika so bile pomembne nove ideje.

Podjetje IBM, ki je imelo sredi šestdesetih let vodilno vlogo na področju digitalne simulacije zveznih dinamičnih sistemov, je postalo po nastanku sposobnih jezikov CSSL precej neaktivno na področju simulacije. Leta 1972 so sicer izdali novo verzijo jezika CSMP III, ki pa je ostala pri starih konceptih in ni strožje upoštevala standarda. Verzija je bila primerna samo za računalnike IBM in je omogočala tudi uporabo grafičnega terminala IBM 2250. Glavna prednost jezika CSMP III in tudi kasnejših dialektov je bila v izredno dobrih zmožnostih za grafično predstavitev rezultatov. Jezik CSMP je precej pred drugimi jeziki omogočal grafično podajanje časovnih odzivov, faznih trajektorij, združevanje rezultatov različnih simulacijskih tekov in celo trodimenzionalno prikazovanje ter senčenje.

Simulacijski jeziki so imeli v začetku sedemdesetih let še vedno izredno slabe interaktivne zmožnosti in so v glavnem delovali v t.i. paketnem načinu obdelave na velikih računalnikih, čeprav so nekatere vrste računalnikov že omogočale določeno interaktivnosti. Eden prvih simulacijskih jezikov, kjer so načrtovalci dali večji poudarek interaktivnosti, je simulacijski jezik SIMNON (SIMulation progam for NONlinear systems, Lund University of Technology). Prva verzija tega jezika je bila razvita l. 1972, prva uporabna verzija pa je prišla na trg leta 1975. Jezik ni upošteval standarda CSSL 67. Interaktivno zmožnost so jeziku povečali tudi tako, da so izvedli prevajanje direktno na nivo strojnega jezika. Zaradi lažje prenosljivosti so to prevajanje realizirali v dveh delih. V prvem delu se izvorni program prevede v kodo virtualnega procesorja, v drugem delu, ki je specifičen za določeno implementacijo, pa v strojno kodo določenega procesorja. Ker je prevajalnik relativno hiter, je možno na interaktivni način spreminjati tudi strukturo modela. SIMNON predstavlja prvi simulacijski jezik, ki je imel moderno zasnovo interaktivnega komandnega jezika za krmiljenje simulacijskih tekov. Jezik je bil relativno enostaven za uporabo, ni poznal indeksiranih spremenljivk, imel pa je novost v opisu modela z diferencialnimi enačbami. Odvode v diferencialnih enačbah je bilo možno označiti povsem matematično (npr. y', y'') in jih v opisu modela ni bilo potrebno integrirati. Ta princip sedaj uporabljajo nekateri najsodobnejši simulacijski jeziki. Slabost tega načina pa je v tem, da mo-

dela ni možno normirati, kar je pri numerično zahtevnih problemih lahko zelo pomembno. Kasnejše verzije jezika SIMNON so omogočale vključevanje modulov v jeziku FORTRAN, zapise z diferenčnimi enačbami, povezovanje več modelov ter optimizacijo. SIMNON še danes predstavlja uspešen simulacijski jezik (Åström, 1985a).

Leta 1972 smo na Fakulteti za elektrotehniko v Ljubljani razvili simulacijski jezik HYSIM (HYbrid SIMulation, Divjak, 1975) za računalnik IBM 1130. Jezik je bil v tistem času izredno napreden, saj je vseboval tudi elemente kombinirane simulacije.

Od leta 1972 naprej se je začela razvijati tudi družina zelo naprednih simulacijskih jezikov DARE (Korn, Wait, 1978). Jeziki se niso strogo držali standarda CSSL 67, zato pa so prinašali precej novosti. Leta 1975 je prišla v uporabo verzija DARE P, ki je delovala sicer še na paketni način, a je predstavljala prvi relativno dobro prenosljiv simulacijski jezik, ki je deloval tudi na miniračunalnikih. Namesto sekcij INITIAL, DYNAMIC in TERMINAL je imel DARE P t.i. blok LOGIC, v katerem je uporabnik s pomočjo proceduralnega programa opisal krmiljenje simulacijskih tekov. Ta način je predstavljal prvi poizkus ločitve modela in eksperimenta. V nekaterih drugih dialektih jezika DARE so bile vgrajene določene rešitve, ki so povečale učinkovitost simulacije v realnem času (DARE/ELEVEN). Uporabljali so bločni princip, saj je bločno strukturo možno relativno enostavno prevajati v zbirnik. To je bilo zlasti učinkovito, ko so se pojavili sposobni makro prevajalniki za zbirne jezike. Pokazalo se je, da je na ta način možno simulirati probleme štiri do desetkrat hitreje kot z prevajanjem v FORTRAN. Še večji časovni prihranek pa je prinesla uporaba aritmetike s fiksno decimalno vejico (cca. 4krat), kar seveda v jeziku FORTRAN ni možno. Uporabnik je v DARE/ELEVEN lahko kombiniral simulacijske segmente z enačbnim načinom ter aritmetiko s pomično vejico in bločne segmente z aritmetiko z fiksno decimalno vejico za simulacijo časovno kritičnih operacij (nekakšna simulacija hibridnega sistema). Nekateri naslednji dialekti (MICRODARE, DESIRE, DESKTOP), ki so se pojavili okoli leta 1980, pa so imeli t.i. ultra hitre prevajalnike, ki so prevajali samo del programa, ki opisuje model, na strojni nivo (Korn, 1983b). Taki jeziki so znani kot jeziki z direktnim izvajanjem, saj uporabnik pri delu praktično ne opazi prevajanja. Ti jeziki so vsebovali tudi zelo dodelane t.i. grafične postprocesorje, ki so ločeni od simulacije omogočali visoko stopnjo interaktivnosti. V nekaterih ozirih so prekašali celo grafiko na sistemih CSMP.

Hitrost izvajanja simulacijskih programov je predstavljala v sedemdesetih letih zelo aktualno problematiko. Zato so nastale prve ideje, da bi se simulacijski jeziki uporabljali v povezavi s hibridnimi računalniki. Prvi jezik, ki je kombiniral pro-

gramske učinkovitosti jezikov CSSL in računske zmožnosti hibridnih računalnikov je bil jezik HL-1 (leta 1973). To so bili prvi zametki konceptov, ki so postali resnično uporabni šele čez deset let.

Leta 1975 je prišel na trg danes komercialno najuspešnejši simulacijski jezik ACSL (Advanced Continuous Simulation Language - Mitchel & Gauthier Ass.). Jezik temelji na standardu CSSL 67, vsebuje pa tudi veliko izkušenj iz razvoja jezika CSSL IV in ima podobne karakteristike. Z dodatno modelno sekcijo DISCRETE so zelo povečali uporabnost jezika za simulacijo kombiniranih sistemov (Mitchel & Gauthier, 1981). V primerjavi z drugimi simulacijskimi jeziki se je ACSL najhitreje prilagodil na različne računalnike. V sredini osemdesetih let so bile dosegljive verzije na številnih tipih računalnikov (od osebnega računalnika do superračunalnika CRAY).

Pojavile so se tudi potrebe po večji računalniški podpori v fazi modeliranja. Jezik DYMOLA (Elmqvist, 1978) je omogočal opis modela v obliki fizikalnih zakonov (npr. Kirchoffove enačbe)), zapis v prostoru stanj pa se je izračunal avtomatično. Zaradi načina povezovanja podmodelov, ki je bilo bolj splošno, kot povezovanje preko vhodov in izhodov, je DYMOLA eno od prvih objektno orientiranih orodij. Računalniške sposobnosti pa so bile takrat še preslabe, da bi koncepti resnično zaživeli.

Sedemdeseta leta so torej prinesla velik napredek na področju digitalne simulacije. Jeziki so že vključevali interaktivne zmožnosti računalnikov in njihove grafične sposobnosti. Velik napredek so omogočile tudi nove numerične metode (knjižnice EISPACK, LINPACK), kar je omogočilo večjo numerično robustnost simulacijskih jezikov.

Osemdeseta leta so prinesla pomemben nadaljnji razvoj omenjenih značilnosti. V zvezi z interaktivnimi zmožnostmi so se razvili vsi znani načini dialoga: vprašanje-odgovor, menujski način in komandni način. Kot zelo uporaben eksperiment se je začela uporabljati optimizacija. Ker so bili obstoječi jeziki strukturno omejeni, so jih v glavnem širili z novimi funkcijami. Tako so začeli v nekatere simulacijske jezike (ACSL, CSSL) vgrajevati postopke iz sistemske teorije kot npr. Bodejev in Nyquistov diagram, diagram lege korenov, hitro Fourier-jevo transformacijo, analizo ustaljenega stanja, linearizacijo itd. Na ta način so jeziki začeli preraščati v pakete CACSD. V začetku osemdesetih let so se pokazale tudi potrebe po prenosu simulacijskih jezikov na mini in predvsem mikroračunalnike. Le-ti so postali toliko sposobni (velik pomnilnik, prevajalniki za FORTRAN, PASCAL, C), da prenos posameznih sicer osiromašenih verzij jezikov z večjih sistemov ni delal resnejših težav. Prav simulacijski jeziki na mikroračunalnikih so šele omogočili, da je si-

mulacija postala dostopna vsakomur.

Zelo sposoben jezik za mikroračunalnike predstavlja ISIM, ki so ga razvili iz jezika ISIS (Crosbie, 1984, Hay, Crosbie, 1984). Jezik se strogo drži standarda CSSL 67, deluje pa na osem in šestnajst bitnih računalnikih pod operacijskima sistemoma CP/M in MS-DOS. Prevajalnik prevede izvorni program v vmesno obliko, ki se potem izvaja na interpreterski način. Zato ima jezik zelo dobre interaktivne zmožnosti. Ker pa ga ni mogoče kombinirati z jezikom FORTRAN, so v sintakso vnesli dodatne stavke kot npr. DO, IF in GOTO. Diferencialne enačbe se direktno vnašajo, odvodov pa v izvornem programu ni potrebno integrirati. Jezik omogoča tudi enostavno eksperimentiranje s proceduralnim programom. Nekatere nadaljnje verzije vsebujejo tudi prve poizkuse modularnega programiranja.

Iz tega obdobja je tudi jezik TUTSIM (Twente University of Technology (1980), TUTSIM, 1983), ki deluje na operacijskih sistemih CP/M in MS-DOS. Predstavlja interpreterski bločno orientirani jezik, napisan v zbirniku. Je enostaven za uporabo, a neprimeren za reševanje kompleksnih problemov. Model je možno podati tudi z bond grafi. Za današnje razmere pa vsebuje relativno slabe integracijske metode.

Podobne lastnosti ima tudi interpreterski bločno orientirani jezik PSI (Delft University of Technology (1983), Van den Bosch, 1987). Slabe lastnosti interpreterskih jezikov skuša odpraviti z velikim številom blokov za opis modela (cca 60), ima pa tudi dodatni uporabniški blok, tako da lahko uporabnik po določenih pravilih v jeziku FORTRAN napiše svoj blok, ki po prevajanju in povezovanju postane standardni PSI blok. Ima tudi sposoben komandni jezik za interaktivno krmiljenje simulacijskih tekov (100 komand). Omogoča pa tudi optimizacijo.

Od simulacijskih jezikov na večjih računalnikih prinaša novost jezik DSL/VS podjetja IBM iz leta 1984 (Syn, Dost, 1985). Tako se je IBM po skoraj 15 letih spet bolj intenzivno vključil v razvoj simulacijskih jezikov, čeprav je vseskozi za svoje lastne potrebe veliko uporabljal in tudi sproti posodabljal simulacijske jezike CSMP in DSL. Tudi novo verzijo, ki v glavnem temelji na predhodnih jezikih, so v glavnem razvili za lastne potrebe. Jezik deluje na računalnikih 370 in 4300, je vsestransko sposoben in omogoča reševanje kompleksnih problemov. Poleg običajnih sekcij za opis modela INITIAL, DYNAMIC in TERMINAL so dodali tudi sekcijo SAMPLE, ki je namenjena predvsem za opis diskretnih regulatorjev. Jezik vsebuje več sposobnih integracijskih postopkov, vključena je tudi optimizacija. Kot je značilno za vse predhodne jezike CSMP, ima zelo sposobno grafiko. Za delo na osebnih računalnikih so razvili jezik PCESP (Personal Computer Engineering Simulation Program, Shah, 1988) iz predhodnega jezika DSL

1130. Le-ta je kompatibilen z DSL/VS, tako da je možno probleme razvijati na PC računalniku (kot na delovni postaji), po potrebi pa je možno simulacijo izvajati na velikem računalniku. Podoben je tudi jezik SYSL/M (90% kompatibilen s CSMP).

V osemdesetih letih so nastali tudi simulacijski jeziki za delo na sodobnih hibridnih sistemih. Jezika HYBSIS (Kleinert in ostali, 1983) in STARTRAN (Landauer, 1988) smo že omenili pri pregledu hibridnih sistemov. Znani pa so tudi simulacijski jeziki na namenskih simulacijskih digitalnih računalnikih kot npr. ADSIM (Grierson, 1986) in PARSIM (Bruijn, Soppers, 1986), ki smo jih omenili pri opisu simulacijskih sistemov na namenskih računalnikih.

V začetku osemdesetih let so se začele kazati potrebe po novem standardu CSSL, ki ne bi več strukturno utesnjeval jezikov ampak bi upošteval sodobne koncepte programskega inženirstva. Toda zaradi zelo različnih interesov sta dve delovni skupini (pri IMACS in SCS) uspeli izdelati le priporočila. Na osnovi teh priporočil so se sredi osemdesetih let začeli razvijati simulacijski jeziki nove generacije (ESL, SYSMOD, COSMOS). Ti jeziki so se razvijali pod močnim vplivom programskega inženirstva in imajo predvsem naslednje pomembne lastnosti:

- modularnost pri opisu modela (hierarhična gradnja modela iz podmodelov),
- vgradnja bolj kompleksnih eksperimentov (razen simulacijskega teka omogočajo jeziki še optimizacijo, linearizacijo, parametrizacijo, izračun ustaljenega stanja, analizo občutljivosti, analize v frekvenčnem prostoru, ...),
- fleksibilne programske in podatkovne strukture,
- značilnosti kombinirane simulacije,
- numerična robustnost (algoritmi za integracijo, optimizacijo, obdelava nezveznosti,...).

Konec osemdesetih let smo tudi na Fakulteti za elektrotehniko in računalništvo v Ljubljani (v sodelovanju z Institutom Jožef Stefan) razvili in dali v uporabo simulacijski jezik tipa CSSL z imenom SIMCOS (SIMulation of COntinuous Systems) (Zupančič, 1989, Zupančič, 1992). Jezik omogoča simulacijo, optimizacijo, linearizacijo in parametrizacijo zveznih in diskretnih dinamičnih sistemov. Omogoča tudi simulacijo v realnem času.

Devetdeseta leta so prinesla velik napredek predvsem pri razvoju uporabniških vmesnikov simulacijskih orodij. Tako na osebnih računalnikih kot na sposobnih

delovnih postajah prevladujejo koncepti okenskih vmesnikov. Nesluten razvoj je naredilo programsko okolje MATLAB tudi na področju simulacij zlasti s paketom SIMULINK (Simulink, 2009). MATLAB-SIMULINK je postalo standardno okolje na vseh akademskih institucijah, kasneje pa se je izdatno začelo uporabljati tudi v industriji. Kasneje so za SIMULINK razvili razne namensko uporabne dodatke, eno je npr. STATE FLOW, ki omogoča vključevanje dogodkov. Velik je tudi poudarek na objektni orientiranosti (npr.Xmath). Nekatera orodja uvajajo tudi večjo podporo v smislu modeliranja (npr. DYMOLA z orodji DYMODRAW, DYMOVIEW, DYMOSIM (Elmqvist, 1994, Cellier, 1991).

V novem tisočletju je simulacija najbolj zaznamovana z objektno orientiranim, fizikalnim in več domenskim jezikom Modelica (Fritzson, 2004, Modelica, 2007, Sodja, Zupančič, 2009). Podobno idejo fizikalnega modeliranja vsebujejo tudi Bond grafi - grafična modelerska tehnika, ki opisuje pretok energije med komponentami (pristop podpira npr. simulacijski paket 20-sim). Modelica postaja priznani standard za modeliranje zveznih pa tudi diskretnih in hibridnih dinamičnih sistemov. Omogoča zlasti veliko podporo za modeliranje, saj ni potrebno izraziti odvodov stanj, kot v primeru večine konvencionalnih simulacijskih orodij. Zlasti je pomemben način povezovanja komponent, ki omogoči gradnjo knjižnic ponovno uporabljivih komponent. Povezujemo pa lahko komponente različnih področij, kar je zlasti pomembno v mehatroniki, robotiki, v avtomobilski industriji, v vodenju sistemov ipd. Zlasti sposobni okolji, ki podpirata jezik Modelica, sta Dymola (Dymola, 2008) in MathModelica. Podoben način uvaja tudi podjetje MathWorks - l. 2008 je prišlo na trg okolje Simscape v sklopu programskega paketa Matlab-Simulink. Žal pa ta način upošteva jezik Modelica le kot idejni koncept.

V tem pregledu smo se omejili predvsem na simulacijske jezike, ki so največ prispevali k napredku in uporabi simulacijskih postopkov. Izpustili smo simulacijske zmožnosti paketov za računalniško podprto načrtovanje vodenja sistemov. Le-ti imajo v mnogočem podobne lastnosti, kot obravnavani jeziki in so se tudi razvijali pod njihovimi vplivi. Pregled teh orodij smo podali v podpoglavju 1.8.

Tabela 5.1 predstavlja v skrčeni obliki glavne karaketeristike razvoja simulacijskih jezikov za simulacijo dinamičnih sistemov.

Tabela 5.1: Razvoj in glavne karakteristike simulacijskih jezikov

1955	SELFRIDGE	prvi simulacijski jezik
1956		prevajalniški koncept (Stein,Rose, Parker)
1963	MIDAS	prvi sposobnejši jezik, vrstni algoritem, integracijska metoda spremenljivega koraka
1964	COBLOC	več integracijskih metod, prosti format pisanja programa
1965	DYNAMO	diagnostika, makro jezik, prikaz rezultatov v obliki grafa, odkrije algebrajsko zanko
1965	DSL 90	prvi prevajalniški enačbno orientirani jezik
1967	CSMP 360, CSMP III	prvi sposobni simulacijski jeziki, funkcijski generatorji, 60 operatorjev, reševanje algebrajske zanke
1967	DSL 1130/1180	prve interaktivne zmožnosti, možna uporaba digitalnega risalnika, spominskega osciloskopa
1967		**standard CSSL 67**
1968	MIMIC	prvi jezik po vzoru CSSL 67
1969	CSSL III	prvi sposobni jezik CSSL
1970	SL-1	prvi jezik za simulacijo v realnem času
1972	CSSL IV	vse do danes eden najsposobnejših simulacijskih jezikov
1972	CSMP III	učinkovit grafični prikaz rezultatov
1972	HYSIM	kombinirana simulacija
1973	HL-1	prvi jezik za programiranje hibridnega sistema
1975	SIMNON	sposoben interaktivni jezik, prevajanje direktno na strojni nivo, uporabniku ni potrebno integrirati odvodov
1975	ACSL	vsestransko dober, danes komercialno najuspešnejši simulacijski jezik
1975	DARE P	prvi jezik za miniračunalnik, prenosljiv, poskus ločitve modela in eksperimenta
1976	DARE/ELEVEN	kombinacija enačbnega in bločnega jezika za hitro simulacijo v realnem času
1978	DYMOLA	nadgradnja za modeliranje, objektna orientiranost
1980	MICRODARE, DESIRE DESKTOP	ultra hitri prevajalnik
1981		nova priporočila CSSL 81 (SCS, IMACS)
1983	ISIM	na osem in šestnajst bitnih mikroračunalnikih, modularno programiranje
1983	HYBSIS, STARTRAN	simulacijska jezika za hibridne sisteme, simulacija v realnem času
1984	ADSIM, PARSIM	simulacijska jezika za namenske simulacijske računalnike, simulacija v realnem času
1984	ESL, SYSMOD, COSMOS	simulacijski jeziki nove generacije
1989	SIMCOS	zvezna in diskretna simulacija, eksperimentiranje, delovanje v realnem času
1990	SIMULINK	grafični uporabniški vmesnik, delovanje v okolju MATLAB
1992	okolje DYMOLA	podpora za modeliranje, objektna orientiranost
1995	20-sim	fizikalno modeliranje, uporaba Bond grafov
1996	jezik MODELICA	poizkus standardizacije jezika za objektno orientirano več domensko modeliranje - nov standard po CSSL'67
2008	Simscape	rešitev podjetja Mathworks za več domensko objektno orientirano modeliranje

5.2 Programska zgradba zveznih simulacijskih jezikov

Pod izrazom simulacijski jezik si običajno ne predstavljamo le jezika za podajanje modela, ampak celotni programski paket, ki omogoča simulacijo. Celotno programsko zgradbo sodobnih jezikov običajno razdelimo v:

- del za opis eksperimenta t.j. modela in metode,
- procesor,
- sistem za izvajanje simulacije oz. eksperimenta (simulator),
- postprocesor za grafično predstavitev rezultatov,
- nadzorni program.

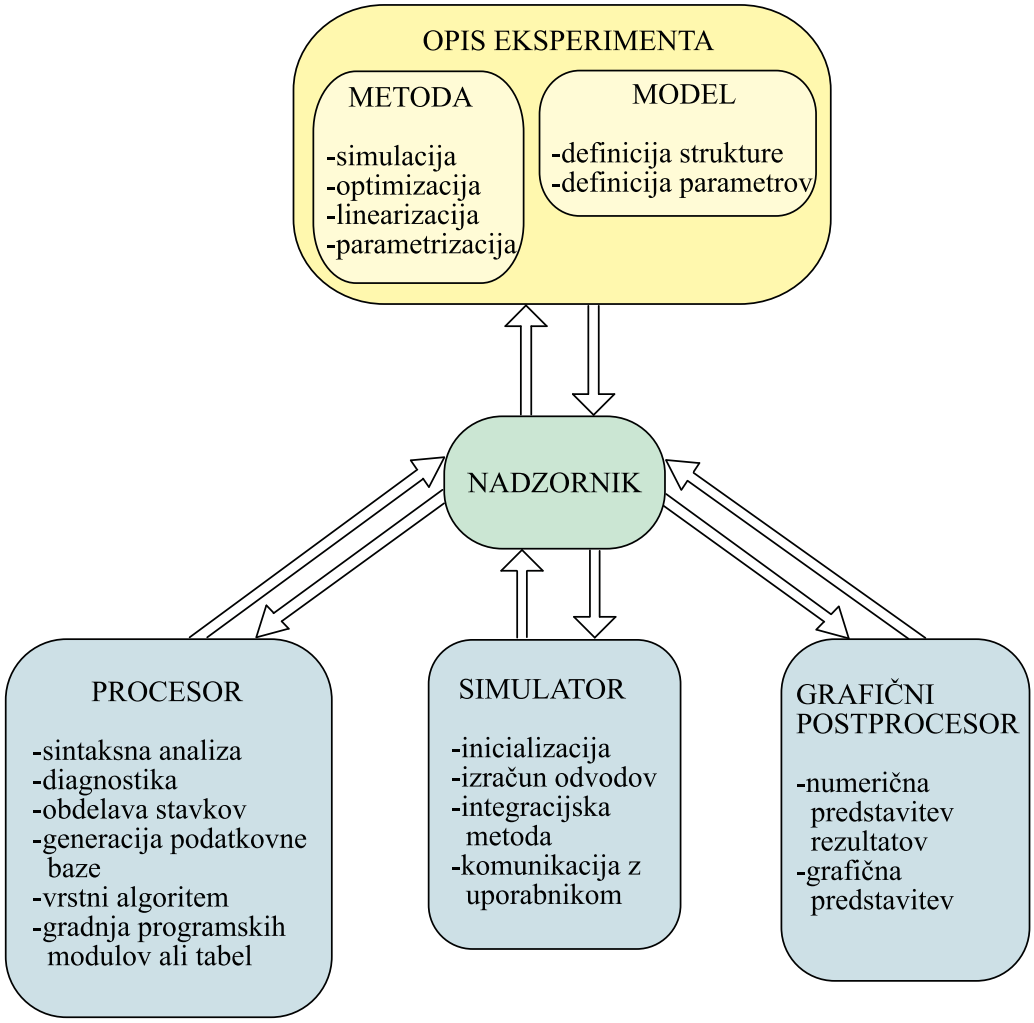
Celotno programsko zgradbo prikazuje slika 5.1.

5.2.1 Opis eksperimenta

Do nedavnega so imeli simulacijski jeziki le en eksperiment, t.j. simulacijski tek. Opis eksperimenta je zahteval le opis modela in nekaterih parametrov za krmiljenje simulacijskega teka. Programski modul za opis eksperimenta (modela) je torej omogočal definirati

- strukturo modela,
- parametre modela,
- krmilne parametre simulacije,
- parametre oz. zahteve za prikaz rezultatov.

Dejstvo, da je bilo v istem programskem modulu potrebno opisati model in nekatere krmilne parametre, imamo za precejšnjo slabost tovrstnih sistemov.



Slika 5.1: Programska zgradba zveznih simulacijskih jezikov

Sodobna simulacijska orodja striktno ločijo med opisom modela in metode. Model in metodo je potrebno opisati v različnih programskih modulih (včasih v različnih datotekah). Poleg elementarne metode - simulacijski tek ima uporabnik na voljo tudi optimizacijo, linearizacijo, parametrizacijo, analizo ustaljenega stanja, analizo občutljivosti itd.

Struktura modela

Glede na zmožnosti, kako definirati strukturo modela, se simulacijski jeziki zelo razlikujejo. Nekateri so bločno orientirani in vsebujejo le osnovne bloke na nivoju analognega računalnika. Moderni simulacijski jeziki pa imajo izredno razvite zmožnosti za opis strukture. Možno je direktno vnašanje kompliciranih diferencialnih enačb, vgnezdenih izrazov itd. Vedno več orodij omogoča opis modela z grafično simulacijsko shemo.

Parametri modela

V večini simulacijskih jezikov razlikujemo med dvema vrstama parametrov:

- konstante modela,
- podatki, ki opisujejo generacijo funkcij (signalov).

Krmilni parametri simulacije

To so podatki za metodo simulacija. S temi parametri definiramo:

- Simulacijski (integracijski) algoritem.
- Komunikacijski interval, ki definira točke neodvisne spremenljivke, v kateri želi uporabnik dobiti rezultate simulacije.
- Računski korak, t.j. korak integracijskega postopka. Velikost koraka je odvisna od časovnih konstant sistema, ki ga simuliramo, od zahtev po relativni oz. absolutni natančnosti ter od izbranega integracijskega postopka.
- Začetno in končno vrednost neodvisne spremenljivke. Pogoj za končanje simulacijskega teka lahko pogosto opišemo tudi z aritmetičnim ali logičnim izrazom.
- Dopustno relativno oz. absolutno napako, če uporabljamo metodo s prilagodljivim računskim korakom.

Parametri oz. zahteve za prikaz rezultatov

S temi parametri uporabnik pove, katere spremenljivke bi rad spremljal, na katerem mediju in v kakšni obliki naj se prikazujejo. V splošnem lahko rezultate spremljamo med simulacijo ali pa jih prikažemo po simulaciji.

Najmodernejši simulacijski jeziki pa dopuščajo tudi bolj kompleksna eksperimentiranja. V takih jezikih lahko uporabnik izbere že vključene metode (npr. optimizacijo, linearizacijo, parametrizacijo, ...) ali pa sam programira svojo metodo. Za take jezike pravimo, da imajo popolno ločitev metode in modela, eksperiment pa je možno izvršiti z uporabo katerekoli metode nad katerimkoli modelom (hierarhični koncept model, metoda eksperiment - Breitenecker, Solar, 1986).

Klasični simulacijski jeziki, ki so komercialno še vedno najuspešnejši in v glavnem še spoštujejo standard CSSL'67 (npr. ACSL, CSSL IV) tudi omogočajo bolj kompleksna eksperimentiranja, vendar je ločitev modela in metode bolj navidezna. Eksperimenetiranje omogočajo t.i. sekcije INITIAL, DYNAMIC in TERMINAL. Z njimi uporabnik definira operacije, ki se izvajajo pred simulacijo, med simulacijo paralelno z modelom in po simulaciji.

5.2.2 Procesor

Procesor simulacijskega jezika sestoji iz enega ali več programskih modulov, ki prevedejo simulacijski model (oz. eksperiment) v programske module in ustrezno podatkovno bazo v primeru prevajalniških jezikov oz. v ustrezne tabele (datoteke) v primeru interpreterskih jezikov. Procesor torej zgradi najbolj vitalni del, ki ga potrebuje simulator.

Procesor najprej izvrši sintaksno analizo nad podanim modelom (simulacijskim programom). Nato se specifično obdelujejo različni stavki izvornega simulacijskega programa. Predvsem se specifično obdelajo stavki (bloki), ki definirajo stanje sistema (bloki, ki vsebujejo spomin, oz. zakasnitveni atribut). Realizirati je namreč potrebno prekinitev vseh zank na izhodih omenjenih blokov (podpoglavje 3.8). Obdelavi stavkov sledi razvrščanje stavkov (blokov), kar omogoča pravilno simulacijo fizikalno gledano paralelno delujočega sistema. Vrstni algoritem lahko razvrsti stavke le v primeru, če je v vsaki zanki vsaj en blok z zakasnitvenim atributom (običajno integrator), saj se v tem primeru že pri predhodni obdelavi stavkov vse zanke prekinejo. Zanke, v katerih ni blokov z zakasnitvenim

atributom, jemlje vrstni algoritem za algebrajske zanke. Pri nekaterih simulacijskih jezikih se v tem primeru le sporoči ustrezna diagnostika, procesiranje pa se prekine. Nekateri jeziki pa imajo vgrajene posebne numerične postopke, ki omogočajo reševanje algebrajskih zank iterativno. Taki postopki pa izredno upočasnijo simulacijo.

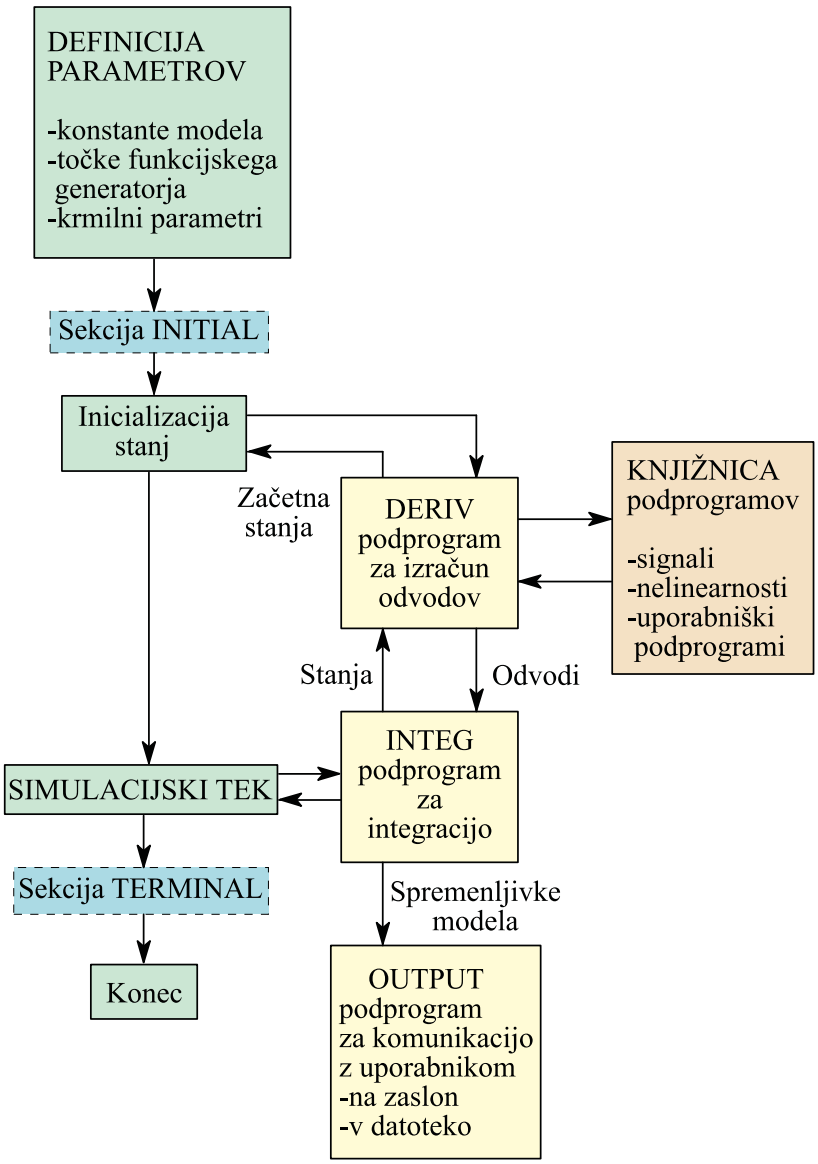
Med obdelavo stavkov se zgradi tudi podatkovna baza modela. Nato se zgradijo ustrezne tabele, ki omogočajo interpretersko simulacijo, v primeru prevajalniških jezikov pa se zgradi simulacijski program (iz enega ali več modulov), ki je lahko direktno v strojni obliki, ali pa v nekem splošnonamenskem programskem jeziku (npr. FORTRAN, PASCAL, C, ...). V slednjem primeru zahteva procesiranje tudi nadaljnje prevajanje in povezovanje (linkanje) v splošnonamenskem jeziku.

Dobri procesorji imajo učinkovito diagnostiko v vseh fazah procesiranja.

5.2.3 Simulator

Simulator ali sistem za izvajanje simulacije je fiksni program (v primeru interpreterskih jezikov) ali pa je program, ki ga zgradi simulacijski procesor (v primeru prevajalniških jezikov). Torej je simulator najbolj vitalni oz. osrednji del simulacijskega jezika. Za kvaliteto simulatorja je najpomembnejša natančnost, numerična stabilnost in računska učinkovitost (hitrost) simulacijskega (integracijskega) algoritma. Slika 5.2 prikazuje programsko zgradbo simulatorja.

Pred začetkom simulacijskega teka mora simulator dobiti določene parametre, t.j. konstante modela, podatke za funkcijske generatorje, krmilne parametre simulacije in zahteve za prikaz rezultatov. Ti podatki so shranjeni na eni ali več datotekah, ki jih je predhodno zgradil simulacijski procesor. Nekateri simulacijski jeziki vsebujejo sekcijo INITIAL, ki opisuje operacije pred simulacijskim tekom. V tem primeru se po čitanju podatkov torej izvede sekcija INITIAL, nato pa se s pomočjo klica podprograma DERIV (le-ta vsebuje enačbe modela) ovrednotijo začetne vrednosti stanj. Simulacijski tek pa se začne s klicem podprograma INTEG, ki med simulacijo zahteva številne klice podprograma za izračun odvodov DERIV (npr. en klic podprograma DERIV na računski korak pri Eulerjevi metodi, štirje klici pri metodi Runge-Kutta, ...). V t.i. komunikacijskih intervalih pa se kliče podprogram OUTPUT, ki skrbi za ustrezno posredovanje rezultatov simulacije uporabniku. Pri jezikih, ki vsebujejo sekcijo TERMINAL se po zaključku simulacije izvedejo še operacije, ki so opisane v tej sekciji.



Slika 5.2: Programska zgradba simulatorja

5.2.4 Postprocesor za grafično predstavitev rezultatov

Uporabnik simulacijskega jezika lahko spremlja rezultate med simulacijskim tekom v grafični ali numerični obliki, bolj kompleksen pregled rezultatov pa se običajno izvede po simulaciji s pomočjo grafičnega postprocesorja. Le-ta je zlasti učinkovit, če omogoča vključitev rezultatov različnih simulacij oz. eksperimentov. Grafični predprocesor običajno omogoča:

- risanje več krivulj na zaslon,
- avtomatsko skaliranje,
- spreminjanje območij,
- povečevanje (zoom),
- odbiranje vrednosti iz krivulj (trace),
- različne vrste interpolacij med sosednjima točkama (ničti red, linearna interpolacija, točkovna predstavitev, ...),
- linearna ali logaritmična skala na oseh, z mrežo ali brez.

Vse funkcije morajo biti dosegljive na interaktivni in uporabniško prijazni način.

5.2.5 Nadzornik

Nadzornik omogoča povezavo vseh na sliki 5.1 prikazanih delov simulacijskega jezika. Najpomembnejše funkcije nadzornika so naslednje:

- Izbira simulacijskega modela. Pri prevajalniških jezikih lahko izberemo model v izvirni ali prevedeni obliki.
- Izbira metode za eksperiment (npr. simulacije, optimizacije, ...).
- Možnost za opis modela oz. metode ali za editiranje obstoječega modela oz. metode.
- Procesiranje modela (pri prevajalniških jezikih vključuje tudi prevajanje in povezovanje v vmesnem splošnonamenskem jeziku).
- Izvršitev simulacijskega teka ali bolj kompleksnega eksperimenta.
- Uporabniško prijazno editiranje konstant modela, podatkov za generacijo funkcij, krmilnih parametrov simulacije, zahtev za komunikacijo z uporabnikom ne da bi bilo potrebno editiranje izvornega programa in ponovno prevajanje v primeru prevajalniških jezikov.

Tako kot pri grafičnem postprocesorju morajo biti tudi tu vse funkcije dosegljive na interaktivni in uporabniško prijazni način.

5.2.6 Uporabniški vmesnik

Uporabniški vmesnik predstavljajo tisti programski moduli, ki omogočajo komunikacijo med simulacijskim jezikom in uporabnikom. Na sliki 5.1 sicer ni omenjen, vendar se nahaja v več delih opisane programske strukture, kot npr. pri opisu eksperimenta (npr. z grafično simulacijsko shemo), v nadzorniku in v grafičnem postprocesorju. Glavna odlika uporabniškega vmesnika je visoka interaktivnost in uporabniška prijaznost.

5.3 Enačbno orientirani simulacijski jeziki

Kot smo omenili v poglavju 4.1, so prevajalniški jeziki običajno enačbno orientirani, kar pomeni, da lahko uporabnik diferencialne enačbe direktno vključuje v izvorni program, ne da bi moral pred tem razviti simulacijsko shemo. Razvoj simulacijskega modela je tako enostavnejši in hitrejši, simulacijski programi pa so krajši, bolj modularni in bolj razumljivi in s tem zelo primerni za dokumentacijo modela.

Zaradi izjemne vloge standarda CSSL'67 na razvoj digitalnih simulacijskih jezikov, bomo kot prvi jezik predstavili jezik SIMCOS (Zupančič, 1992), ki je bil razvit pod vplivom tega standarda. Sicer sta v svetovnem merilu najuspešnejša tovrstna simulacijska jezika ACSL (Mitchel & Gauthier, 1981) in CSSL IV (Nilsen, 1984). Izkušnje, ki jih pridobimo ob uporabi enega jezika tipa CSSL, omogočajo rabo kateregakoli drugega CSSL jezika brez večjih problemov.

Razen CSSL jezika SIMCOS bomo prikazali še jezik SIMNON. Le-ta je uspešen predstavnik enačbno orientiranih jezikov, ki niso upoštevali standarda CSSL.

5.3.1 Simulacijski jezik SIMCOS

Simulacijski jezik SIMCOS (Zupančič, 1989, Zupančič, 1992) smo razvili na Fakulteti za elektrotehniko in računalništvo v Ljubljani v sodelovanju z Institutom Jožef Stefan in spada med prevajalniške jezike tipa CSSL. Izvorni program v sintaksi CSSL se prevaja v module v jeziku FORTRAN, ob tem pa se generira tudi potrebna podatkovna baza. Moduli v jeziku FORTRAN se nato prevajajo s prevajalnikom FORTRAN, tako dobljeni objektni moduli pa se nato povežejo

(linkajo) s knjižnicami jezikov FORTRAN in SIMCOS v izvršljivi simulacijski program.

Opis modela

Simulacijski model je možno opisati s tekstovnim načinom ali pa s pomočjo grafičnega predprocesorja BLOCK v obliki simulacijske sheme (Zupančič, 1992).

Pri opisu modela s tekstovnim načinom moramo poznati sintakso jezika SIMCOS. Program, ki ga napišemo v datoteko, sestavljajo naslednji stavki:

- osnovni stavki,
- krmilni stavki,
- predstavitveni stavki in
- izhodni stavki.

Ker je jezik SIMCOS natančno opisan v priročniku (Zupančič, 1992), bomo na tem mestu podali le nekoliko posplošen in skrajšan pregled stavkov.

Osnovni stavki

Osnovni stavki omogočajo definicijo nekaterih osnovnih lastnosti simulacijskega modela (npr. deklaracija spremenljivk, definicije konstant,...). Prvi stavek vsakega programa je običajno stavek `PROGRAM`. Stavek določa ime modela. Zadnji stavek mora biti stavek `END`. Komentar uvedemo s stavkom `COMMENT` ali z znakom "v prvi koloni.

Stavek `ARRAY` določa imena in dimenzije indeksiranih spremenljivk in ima obliko

```
ARRAY sprem(dim[,dim][,dim]) [,sprem(dim[,dim][,dim])]...
```

`sprem` ... ime indeksirane spremenljivke

`dim`	`...`	dimenzija indeksirane spremenljivke

Oglati oklepaji vključujejo izbirne parametre, spremenljivke ali izraze.

Stavek `CONSTANT` uporabljamo za definicijo konstant oz. za inicializacijo spremenljivk (navadnih in enodimenzionalnih indeksiranih). Stavek ima obliko

```
CONSTANT sprem = konst [,sprem = konst]...
```

`sprem`	...	ime spremenljivke
`konst`	...	vrednost(i) spremenljivke (vrednosti indeksirane spremenljivke so ločene z vejicami)

Stavek `TABLE` določa funkcijo ene ali dveh neodvisnih spremenljivk. Ima obliko

```
TABLE ime,n,dim,pod
```

`ime`	...	ime funkcijskega generatorja
`n`	...	število neodvisnih spremenljivk (celoštevilčna konstanta 1 ali 2)
`dim`	...	število lomnih točk neodvisnih spremenljivk (dim1, če je ena neodvisna spremenljivka oz. dim1, dim2, če sta dve neodvisni spremenljivki)
`pod`	...	vrednosti neodvisnih in odvisne spremenljivke v lomnih točkah (realne ali celoštevilčne konstante)

Stavek `TERMT` določa pogoj za končanje simulacijskega teka. Ima obliko

```
TERMT izraz
```

Ko vrednost logičnega izraza `izraz` postane `.TRUE.`, se simulacijski tek konča.

Krmilni stavki

Krmilne stavke uporabljamo za definicijo imen in vrednosti krmilnih spremenljivk. Z njimi podajamo zahteve v zvezi z neodvisno spremenljivko simulacije, zahteve za simulacijski algoritem ter izhodne zahteve. Običajno ni potrebno

navesti vseh krmilnih stavkov. V takem primeru se uporabijo privzeta imena in vrednosti. Stavki imajo obliko

```
VARIABLE  sprem  =  konst
CINTERVAL sprem  =  konst
NSTEPS    sprem  =  konst
MERROR    sprem  =  konst
XERROR    sprem  =  konst
ALGORITHM sprem1 =  konst1, sprem2 = konst2
```

`sprem, sprem1, sprem2 ...`	poljubna imena spremenljivk
`konst, konst1, konst2 ...`	vrednosti spremenljivk (realna ali celoštevilčna)

Stavek `VARIABLE` določa ime neodvisne spremenljivke in njeno začetno vrednost. Stavek `CINTERVAL` določa velikost komunikacijskega intervala v enotah neodvisne spremenljivke (običajno čas). Stavek `NSTEPS` določa pri integracijski metodi s prilagodljivim računskim korakom začetno število računskih korakov znotraj komunikacijskega intervala, pri metodah s stalnim korakom pa število računskih korakov znotraj komunikacijskega intervala. Stavka `MERROR` in `XERROR` določata dopustno relativno in absolutno napako pri integracijskih postopkih s prilagodljivim računskim korakom. Stavek `ALGORITHM` določa uporabljeni simulacijski algoritem. Uporabnik lahko z vrednostjo `konst2` izbira med naslednjimi metodami: diskretna simulacija (`konst2=1`), Eulerjeva metoda (`konst2=3`), metoda Runge-Kutta-Gill s stalnim korakom (`konst2=1`), metoda Runge-Kutta-Gill s prilagodljivim korakom (`konst2=8`), metoda Runge-Kutta-Merson (`konst2=11`), Rosenbrock-ova polimplicitna metoda (`konst2=6`), ekstrapolacijska metoda z linearno implicitnim sredinskim pravilom (`konst2=7`), Gear-ova metoda za toge sisteme (`konst2=9`) in Adams-Moulton-ova prediktor korektor metoda (`konst2=10`). Prve tri metode uporabljajo stalni računski korak, preostale pa prilagodljiv računski korak, kar pomeni, da se med simulacijo ocenjuje dejanska napaka, glede na velikost predpisane napake pa se sproti avtomatsko prilagaja velikost računskega koraka. Izbira optimalnega integracijskega algoritma zahteva poglobljeno znanje o numerični problematiki integracijskih algoritmov. Na tem mestu naj povemo le, da so najbolj splošno uporabne metode Runge-Kutta, v primeru togih sistemov (zelo različne časovne konstante) pa je smiselno uporabiti Gearovo metodo. Nekateri od zgoraj omenjenih algoritmov omogočajo tudi simulacijo v realnem v času.

Predstavitveni stavki

Predstavitveni stavki (bloki) podajajo sistem, ki ga simuliramo. Vrstni red predstavitvenih stavkov ni važen, ker simuliramo sistem, ki deluje paralelno. SIMCOS prevajalnik z vgrajenim vrstnim algoritmom uredi stavke tako, da so vse vhodne spremenljivke definirane prej, preden se stavek izvrši.

Predstavitveni stavek določi vrednost ene ali več izhodnih spremenljivk kot rezultat določenih operacij na naboru vhodnih spremenljivk. Uporabljamo lahko simulacijsko orientirane operatorje (kot na primer integratorje in proceduralne bloke), običajne aritmetične prireditvene stavke (v njih lahko kličemo tudi vse sistemske in uporabniške funkcijske podprograme), stavke za realizacijo signalov, nelinearnosti in stavke za realizacijo zveznih in diskretnih dinamičnih podmodelov.

Simulacijsko orientirani stavki

`INTEG` operator predstavlja temeljni stavek simulacijskega jezika in realizira simulacijski algoritem (integriranje ali zakasnitev). Stavek ima obliko

```
sprem = INTEG(izraz1,izraz2)
```

`sprem` ... ime spremenljivke, ki predstavlja izhod iz operatorja (stanje)

`izraz1` ... vhod v operator (odvod)

`izraz2` ... začetni pogoj

Stavek `PROCEDURAL` uvaja skupino stavkov v jeziku FORTRAN, ki jih napiše uporabnik za realizacijo določene strukture (bloka) oz. za opis relacij med vhodnimi in izhodnimi spremenljivkami. Zapis ima obliko

```
PROCEDURAL (sprem1 = sprem2)
```

`sprem1` ... seznam izhodnih spremenljivk

`sprem2` ... seznam vhodnih spremenljivk

Vsak blok `PROCEDURAL` se mora začenjati s stavkom `PROCEDURAL` in končati s stavkom `END`. Vrstni algoritem jezika SIMCOS obravnava blok `PROCEDURAL` kot en predstavitveni stavek in ga uvrsti glede na seznama vhodnih in izhodnih spremenljivk. Vrstni red stavkov znotraj bloka ostane popolnoma nespremenjen.

Stavki za realizacijo signalov in nelinearnosti

Slika 5.3 prikazuje v jezik SIMCOS vgrajene signale in nelinearnosti.

Signali in nelinearnosti so kot funkcijski podprogrami vključeni v knjižnici jezika SIMCOS. Klicni stavki imajo naslednje oblike:

Stopnica
```
Y=STEP(X,P)
```

Linearno naraščajoči signal
```
Y=RAMP(X,P)
```

Vlak impulzov
```
Y=PULSE(X,TZ,P,W)
```

Harmonska funkcija
```
Y=HARM(X,TZ,W,P)
```

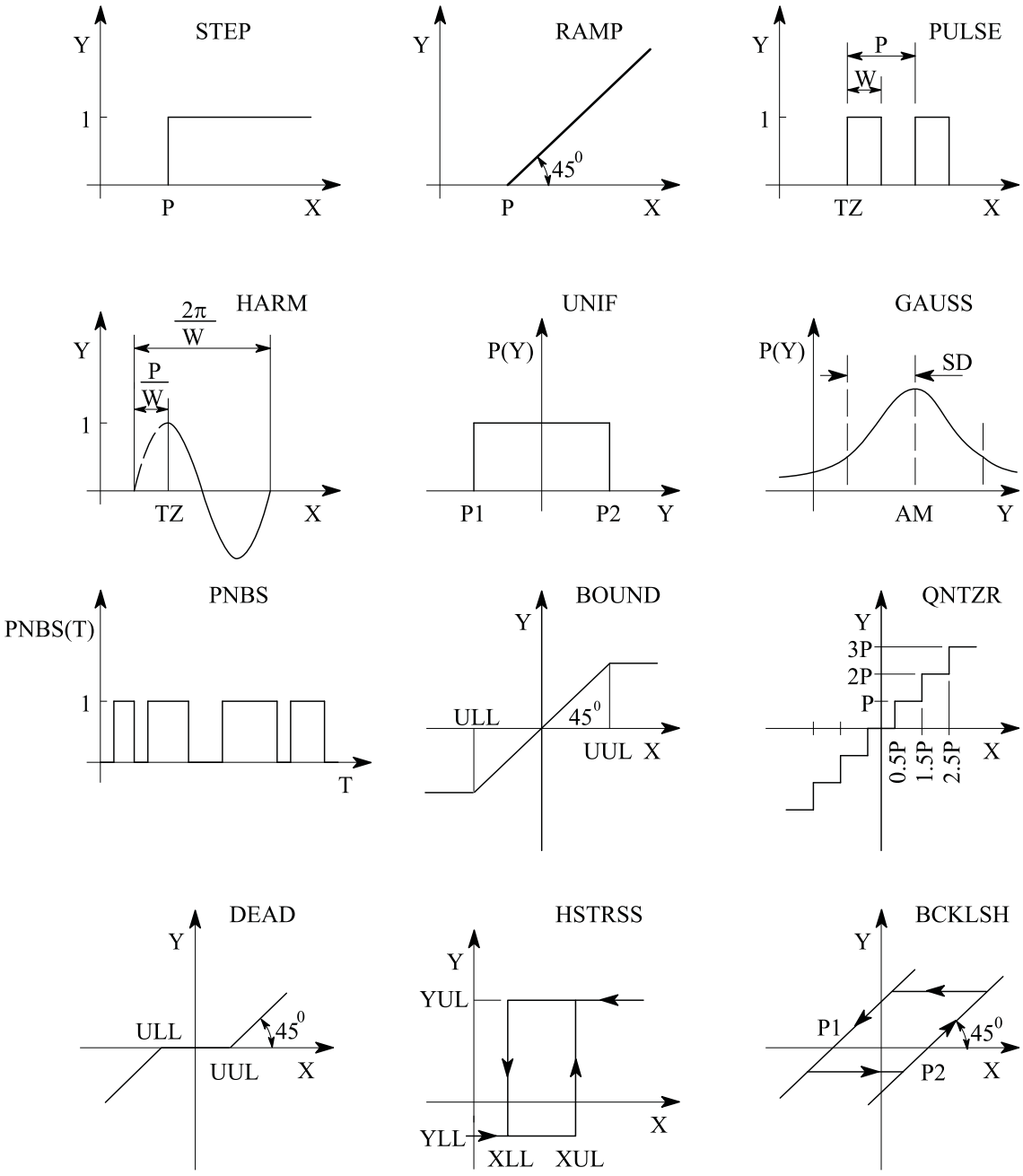
Uniformni šum
```
Y=UNIF(P1,P2,R0,R1,TS)
```

Gaussov šum
```
Y=GAUSS(AM,SD,R0,R1,TS)
```

Pseudonaključni binarni šum
```
Y=PNBS(R1,R2,R3,TS)
```
`R1`, `R2`, `R3` so začetni pogoji,
`TS` je čas vzorčenja

Omejevalnik
```
Y=BOUND(X,ULL,UUL)
```

Kvantizator



Slika 5.3: Signali in nelinearnosti

```
Y=QNTZR(X,P)
```

Mrtva cona

```
Y=DEAD(X,ULL,UUL)
```

Histereza

```
Y=HSTRSS(X,XLL,XUL,YLL,YUL,STANJE)
```

Mrtvi hod

```
Y=BCKLSH(X,P1,P2,STANJE)
```

Komparator

```
Y=COMPAR(X1,X2)
```

$$\begin{aligned} \texttt{Y} &= \texttt{0.} \quad \check{c}e \quad \texttt{X1} < \texttt{X2} \\ \texttt{Y} &= \texttt{1.} \quad \check{c}e \quad \texttt{X1} \geq \texttt{X2} \end{aligned}$$

Funkcijsko stikalo

```
Y=FCNSW(X,P1,P2,P3)
```

$$\begin{aligned} \texttt{Y} &= \texttt{P1} \quad \check{c}e \quad \texttt{X} < \texttt{0.} \\ \texttt{Y} &= \texttt{P2} \quad \check{c}e \quad \texttt{X} = \texttt{0.} \\ \texttt{Y} &= \texttt{P3} \quad \check{c}e \quad \texttt{X} > \texttt{0.} \end{aligned}$$

Vhodno stikalo

```
Y=SWIN(X,P1,P2)
```

$$\begin{aligned} \texttt{Y} &= \texttt{P1} \quad \check{c}e \quad \texttt{X} < \texttt{0.} \\ \texttt{Y} &= \texttt{P2} \quad \check{c}e \quad \texttt{X} \geq \texttt{0.} \end{aligned}$$

Pri vseh funkcijah velja, da je `X` vhodni signal, `Y` izhodni signal, vloga večine parametrov pa je razvidna iz slike 5.3.

Dinamični podmodeli

Dinamični podmodeli so vnaprej prevedeni moduli, uporabnik pa jih kliče kot funkcije (podobno kot pri signalih, nelinearnostih). V vsaki zanki simulacijske sheme se mora nahajati vsaj en podmodel z zakasnitvenim atributom (trenutna vrednost vhoda ne vpliva na trenutno vrednost izhoda). Podmodeli, ki opisujejo splošne dinamične strukture (npr. prenosna funkcija, zapis v prostoru stanj), imajo zato dva primerka: brez zakasnitve in z zakasnitvijo. Matrike je potebno podajati kot enodimenzionalna polja tako, da jih prepisujemo po vrsticah.

Zvezni dinamični podmodeli

Integrator ($I0$)

```
Y=FIN(U,YO)
```

$G(s) = \frac{Y(s)}{U(s)} = \frac{1}{s}$ $y(0) = y_0$

Vektorski integrator

```
Y=VIN(U,N,YO)
```

$N \cdots$število integratorjev

Krmiljeni integrator ($I0$)

```
Y=FKIN(U,YO,IC,OP)
```

$G(s) = \frac{Y(s)}{U(s)} = \frac{1}{s}$ $y(0) = y_0$

IC	OP	Stanje integratorja
0	0	drži (HD)
0	1	integrira (OP)
1	0	začetni pogoji (IC)
1	1	začetni pogoji (IC)

Sistem 1.reda ($P1$)

```
Y=FLAG(U,K,TAU)
```

$G(s) = \frac{Y(s)}{U(s)} = \frac{K}{\tau s+1}$

Integrirni sistem ($I1$)

```
Y=FINTLG(U,K,TAU)
```

$G(s) = \frac{Y(s)}{U(s)} = \frac{K}{s(\tau s+1)}$

Diferencirni sistem ($D1$)

```
Y=DIFF(U,KD,TF)
```

$G(s) = \frac{Y(s)}{U(s)} = \frac{K_D s}{T_f s+1}$

Zakasnilno - prehitevalni člen

```
Y=FLEDLG(U,K,Z,P)
```

$G(s) = \frac{Y(s)}{U(s)} = K\frac{s-z}{s-p}$

Sistem 2.reda ($P2$)

```
Y=SECOR(U,ZETA,OMEGAN)
```

$G(s) = \frac{Y(s)}{U(s)} = \frac{\omega_n^2}{s^2+2\zeta\omega_n s+\omega_n^2}$

Sistem z mrtvim časom

```
Y=DELAY(U,TD,STATES,TS)
```

$G(s) = \frac{Y(s)}{U(s)} = e^{-T_d s}$
$T_s \cdots$ perioda vzorčenja

Prenosna funkcija v polinomski obliki (brez zakasnitve)

```
Y=CTF(U,N,B,A)
```

$G(s) = \frac{Y(s)}{U(s)} = \frac{b_0 s^n+b_1 s^{n-1}+\cdots+b_{n-1}s+b_n}{s^n+a_1 s^{n-1}+\cdots+a_{n-1}s+a_n} = \frac{B(1)s^N+B(2)s^{N-1}+\cdots+B(N)s+B(N+1)}{s^N+A(1)s^{N-1}+\cdots+A(N-1)s+A(N)}$

Prenosna funkcija v polinomski obliki (z zakasnitvijo)

```
Y=CTFD(U,M,N,B,A)
```

$G(s) = \frac{Y(s)}{U(s)} = \frac{b_0 s^m + b_1 s^{m-1} + \cdots + b_{m-1} s + b_m}{s^n + a_1 s^{n-1} + \cdots + a_{n-1} s + a_n} = \frac{B(1) s^M + B(2) s^{M-1} + \cdots + B(M) s + B(M+1)}{s^N + A(1) s^{N-1} + \cdots + A(N-1) s + A(N)}$
$m < n$

Prenosna funkcija v faktorizirani obliki (brez zakasnitve)

```
Y=CZP(U,N,K,Z,P)
```

$G(s) = \frac{Y(s)}{U(s)} = K \frac{(s-z_1)\cdots(s-z_n)}{(s-p_1)\cdots(s-p_n)} = K \frac{(s-Z(1))\cdots(s-Z(N))}{(s-P(1))\cdots(s-P(N))}$

Prenosna funkcija v faktorizirani obliki (z zakasnitvijo)

```
Y=CZPD(U,M,N,K,Z,P)
```

$G(s) = \frac{Y(s)}{U(s)} = K \frac{(s-z_1)\cdots(s-z_m)}{(s-p_1)\cdots(s-p_n)} = K \frac{(s-Z(1))\cdots(s-Z(M))}{(s-P(1))\cdots(s-P(N))}$
$m < n$

Sistem v prostoru stanj (brez zakasnitve)

```
Y=CSS(U,N,A,B,C,D,X)
```

$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}u$
$y = \boldsymbol{C}\boldsymbol{x} + \boldsymbol{D}u$
$n \cdots$red sistema

Sistem v prostoru stanj (z zakasnitvijo)

```
Y=CSSD(U,N,A,B,C,X)
```

$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}u$
$y = \boldsymbol{C}\boldsymbol{x}$
$n \cdots$red sistema

Multivariabilni sistem v prostoru stanj (brez zakasnitve)

```
Y=MCSS(U,N,M,L,A,B,C,D,X)
```

$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u}$
$\boldsymbol{y} = \boldsymbol{C}\boldsymbol{x} + \boldsymbol{D}\boldsymbol{u}$

n	$\cdots$	red sistema
m	$\cdots$	število vhodov
l	$\cdots$	število izhodov

Multivariabilni sistem v prostoru stanj (z zakasnitvijo)

```
Y=MCSSD(U,N,M,L,A,B,C,X)
```

$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u}$
$\boldsymbol{y} = \boldsymbol{C}\boldsymbol{x}$

n	$\cdots$	red sistema
m	$\cdots$	število vhodov
l	$\cdots$	število izhodov

PI regulator

```
U=PI(E,KP,KI)
```

$G(s) = \frac{U(s)}{E(s)} = K_P + \frac{K_I}{s}$

PID regulator

```
U=PID(E,KP,KI,KD,TF)
```

$G(s) = \frac{U(s)}{E(s)} = K_P + \frac{K_I}{s} + \frac{K_D s}{T_f s+1}$

Industrijski PID regulator

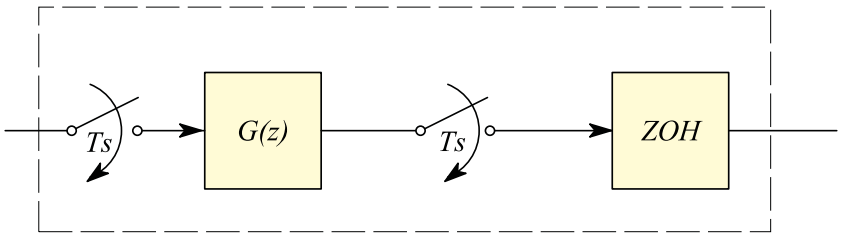
```
U=PIDAB(Y,R,UOO,MA,KP,TI,TD,TF,KA,GAMA,UMIN,UMAX)
```

$E = R - Y$
$G(s) = \frac{U(s)}{E(s)} = K_P(1 + \frac{1}{T_i s} + \frac{T_D s}{T_f s+1})$

U_{00}	$\cdots$	signal za vodenje v režimu ROČNO
`MA`	$\cdots$	preklop ROČNO/AVTOMATSKO (1/0)
K_a	$\cdots$	konstanta zaščite pred integralskim pobegom (priporočljivo $k_a = k_p$)
γ	$\cdots$	faktor, ki določa način uporabe D-člena $\gamma = 0 \cdots$na D-člen je pripeljan pogrešek $\gamma = 1 \cdots$na D-člen je pripeljana regulirana veličina
U_{min}, U_{max}	$\cdots$	minimalna in maksimalna vrednost regulirne veličine

Diskretni dinamični podmodeli

Vsak diskretni podmodel ima na vhodu vzorčevalnik, na izhodu pa zadrževalnik ničtega reda (zero - order hold ZOH ali D/A pretvornik). T_s je perioda vzorčenja. Izvedbo v primeru, če ga opisuje prenosna funkcija, prikazuje slika 5.4.



Slika 5.4: Izvedba diskretnega podmodela

Diskretna zakasnitev

```
Y=DDLY(U,D,STATES,TS)
```

$G(z) = \frac{Y(z)}{U(z)} = z^{-d}$

d	$\cdots$	vrednost diskretne zakasnitve
`STATES`	$\cdots$	začetne vrednosti zakasnitev

Prenosna funkcija v polinomski obliki (brez zakasnitve)

```
Y=DTF(U,N,B,A,TS)
```

$$G(z) = \frac{Y(z)}{U(z)} = \frac{b_0+b_1z^{-1}+\cdots+b_{n-1}z^{-(n-1)}+b_nz^{-n}}{1+a_1z^{-1}+\cdots+a_{n-1}z^{-(n-1)}+a_nz^{-n}} = \frac{B(1)+B(2)z^{-1}+\cdots+B(N)z^{-(N-1)}+B(N+1)z^{-N}}{1+A(1)z^{-1}+\cdots+A(N-1)z^{-(N-1)}+A(N)z^{-N}}$$

Prenosna funkcija v polinomski obliki (z zakasnitvijo)

```
Y=DTFD(U,M,N,B,A,TS)
```

$G(z) = \frac{Y(z)}{U(z)} = \frac{b_1 z^{-1} + \cdots + b_{m-1} z^{-(m-1)} + b_m z^{-m}}{1 + a_1 z^{-1} + \cdots + a_{n-1} z^{-(n-1)} + a_n z^{-n}} = \frac{B(1)z^{-1} + B(2)z^{-2} + \cdots + B(M-1)z^{-(M-1)} + B(M)z^{-M}}{1 + A(1)z^{-1} + \cdots + A(N-1)z^{-(N-1)} + A(N)z^{-N}}$

Prenosna funkcija v faktorizirani obliki (brez zakasnitve)

```
Y=DZP(U,M,N,K,Z,P,TS)
```

$G(z) = \frac{Y(z)}{U(z)} = K\frac{(1-z_1 z^{-1})\cdots(1-z_m z^{-1})}{(1-p_1 z^{-1})\cdots(1-p_N z^{-1})} = K\frac{(1-Z(1)z^{-1})\cdots(1-z(M)z^{-1})}{(1-P(1)z^{-1})\cdots(1-P(N)z^{-1})}$

Prenosna funkcija v faktorizirani obliki (z zakasnitvijo)

```
Y=DZPD(U,D,M,N,K,Z,P,TS)
```

$G(z) = \frac{Y(z)}{U(z)} = K\frac{(1-z_1 z^{-1})\cdots(1-z_m z^{-1})}{(1-p_1 z^{-1})\cdots(1-p_N z^{-1})} z^{-d} = K\frac{(1-Z(1)z^{-1})\cdots(1-z(M)z^{-1})}{(1-P(1)z^{-1})\cdots(1-P(N)z^{-1})} z^{-d}$
$d > 0$

Sistem v prostoru stanj (brez zakasnitve)

```
Y=DSS(U,N,A,B,C,D,X,TS)
```

$\boldsymbol{x}(k+1) = \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{B}u(k)$
$y(k) = \boldsymbol{C}\boldsymbol{x}(k) + \boldsymbol{D}u(k)$
$n \cdots$red sistema

Sistem v prostoru stanj (z zakasnitvijo)

```
Y=DSSD(U,N,A,B,C,X,TS)
```

$\boldsymbol{x}(k+1) = \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{B}u(k)$
$y(k) = \boldsymbol{C}\boldsymbol{x}(k)$
$n \cdots$red sistema

Multivariabilni sistem v prostoru stanj (brez zakasnitve)

```
Y=MDSS(U,N,M,L,A,B,C,D,X,TS)
```

$\boldsymbol{x}(k+1) = \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{B}\boldsymbol{u}(k)$
$\boldsymbol{y}(k) = \boldsymbol{C}\boldsymbol{x}(k) + \boldsymbol{D}\boldsymbol{u}(k)$
n $\cdots$ red sistema
m $\cdots$ število vhodov
l $\cdots$ število izhodov

Multivariabilni sistem v prostoru stanj (z zakasnitvijo)

```
Y=MDSSD(U,N,M,L,A,B,C,X,TS)
```

$\boldsymbol{x}(k+1) = \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{B}\boldsymbol{u}(k)$
$\boldsymbol{y}(k) = \boldsymbol{C}\boldsymbol{x}(k)$
n $\cdots$ red sistema
m $\cdots$ število vhodov
l $\cdots$ število izhodov

Sistem vzorči, drži (sample&hold)

```
Y=SH(U,STATE,TS)
```

PID regulator

```
U=DPID(E,Q,STATES,TS)
```

$G(z) = \frac{Y(z)}{U(z)} = \frac{q_0+q_1z^{-1}+q_2z^{-2}}{1-z^{-1}}$

Industrijski PID regulator

```
U=DPIDAB(Y,R,UOO,MA,KP,TI,TD,TF,KA,GAMA,UMIN,UMAX,TS)
```

$E = R - Y$
$G(z) = \frac{U(z)}{E(z)} = K_P[1 + (\frac{1}{T_i s})_{s=\frac{z-1}{T_s}} + (\frac{T_D s}{T_f s+1})_{s=\frac{2}{T_s}\frac{z-1}{z+1}}]$

U_{00}	$\cdots$	signal za vodenje v režimu ROČNO
`MA`	$\cdots$	preklop ROČNO/AVTOMATSKO (1/0)
K_a	$\cdots$	konstanta zaščite pred integralskim pobegom (priporočljivo $k_a = k_p$)
γ	$\cdots$	faktor, ki določa način uporabe D-člena $\gamma = 0 \cdots$na D-člen je pripeljan pogrešek $\gamma = 1 \cdots$na D-člen je pripeljana regulirana veličina
U_{min}, U_{max}	$\cdots$	minimalna in maksimalna vrednost regulirne veličine

Izhodni stavki

Izhodni stavki omogočajo prikaz rezultatov na zaslon in shranjevanje v datoteko.

Stavek `OUTPUT` povzroči izpis spremenljivk med simulacijskim tekom na vsakih n komunikacijskih intervalov na zaslon. Ima obliko

```
OUTPUT [n,] sprem [,sprem,]...
```

Stavek `PREPAR` povzroči vpis spremenljivk med simulacijskim tekom na vsakih n komunikacijskih intervalov v datoteko. Ima obliko

```
PREPAR [n,] sprem [,sprem,]...
```

Zmožnosti eksperimentiranja

Simulacijski jezik SIMCOS omogoča programiranje kompleksnih eksperimentov s simulacijskim modelom. Običajno so v take eksperimente vključeni iterativni simulacijski teki (analiza odvisnosti od parametrov, optimizacija ipd.). V ta namen mora uporabnik definirati razen modela v sintaksi jezika SIMCOS še tri sekcije (datoteke):

INITIAL	(datoteka model.INI)
DYNAMIC	(datoteka model.DYN)
TERMINAL	(datoteka model.TER)

INITIAL

V tej sekciji uporabnik s stavki v jeziku FORTRAN opiše operacije, ki naj se izvršijo pred simulacijskim tekom.

TERMINAL

V tej sekciji uporabnik s stavki v jeziku FORTRAN opiše operacije, ki naj se izvršijo po simulacijskem teku.

DYNAMIC

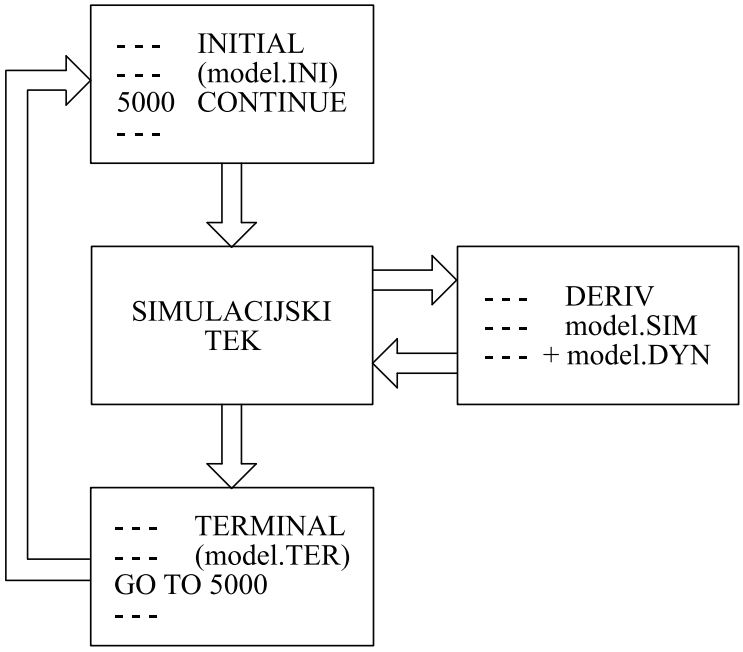
Sekcijo (datoteka model.DYN) se opiše s programom v jeziku SIMCOS. Uporablja se za opis tistih funkcij eksperimenta, ki se morajo izvajati med simulacijo (paralelno z modelom). V tej sekciji navadno definiramo vhodne signale in cenilko za optimizacijo. Zaradi sistematičnosti lahko vključimo tudi stavke za določitev konstant eksperimenta, stavke za krmiljenje simulacije in stavke za izhodne operacije. Tako lahko v izvornem simulacijskem programu (model.SIM) ostanejo le enačbe modela.

Način eksperimentiranja v simulacijskem jeziku SIMCOS prikazuje slika 5.5. S pomočjo take strukture lahko dosežemo ponavljanje simulacijskih tekov z ustreznimi skočnimi stavki v sekcijah INITIAL oziroma TERMINAL.

Za učinkovito eksperimentiranje s simulacijskim jezikom SIMCOS so vgrajeni nekateri eksperimenti s pomočjo vnaprej pripravljenih eksperimentalnih sekcij. Ti eksperimenti so optimizacija, parametrizacija in linearizacija.

Optimizacija

Metoda omogoča določiti optimalne vrednosti parametrov glede na predpisano cenilko, ki se izračuna s pomočjo simulacijskega teka. Optimizacija se izvaja v iteracijah, vsaka iteracija pa sestoji iz več simulacijskih tekov. Število tekov v eni iteraciji je odvisno od uspešnosti poizkusov in od števila optimiranih parametrov.



Slika 5.5: Koncept eksperimentiranja v jeziku SIMCOS

Rezultat vsake iteracije so nove vrednosti optimiranih parametrov, ki običajno predstavljajo izboljšano vrednost cenilke.

Pred optimizacijo mora uporabnik izbrati parametre (konstante), katerih optimalne vrednosti želi poiskati, definirati mora cenilko in omejitve. Razen tega mora podati še začetne iskalne korake, ter pogoje za končanje optimizacije.

Parametrizacija

Eksperiment parametrizacija omogoča avtomatsko ponavljanje simulacijskih tekov ob spreminjajoči se vrednosti ene izmed konstant simulacijskega modela. Ime ustrezne konstante, njeno začetno in končno vrednost ter korak spreminjanja podamo v nadzornem programu.

Linearizacija

S pomočjo linearizacije lahko uporabnik dobi linearizirani zapis modela v prostoru stanj (matrike $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ in $\boldsymbol{D}$) v končni točki simulacije. V nadzornem programu je potrebno izbrati vhode in izhode modela.

Interaktivne zmožnosti

Nadzorni program skupaj z uporabniškim vmesnikom omogoča avtomatsko procesiranje od izvornega do izvršljivega programa ter uporabniško prijazno opisovanje in editiranje izvornega modela, prevedenega modela ali eksperimenta (npr. editiranje konstant modela, editiranje točk funkcijskih generatorjev, editiranje izhodnih zahtev, editiranje krmilnih parametrov simulacije, editiranje parametrov eksperimenta,...). Delo je menujsko orientirano in zato primerno tudi za manj izkušene uporabnike.

5.3.2 Simulacijski jezik SIMNON

Ni prav veliko simulacijskih sistemov, ki ne spoštujejo standarda CSSL, a so vendarle komercialno uspešni. Eden od njih je jezik SIMNON (Elmqvist, 1975), (Åström, 1985a). Jezik sicer deluje kot prevajalnik, vendar ker ne prevaja v višje splošnonamenske jezike, kot to delajo običajno jeziki tipa CSSL, odpade dolgotrajno nadaljnje (npr. fortransko) prevajanje, zato je celotno procesiranje modela bistveno hitrejše. Zaradi te hitrosti in zaradi zelo sposobnega ukaznega načina za interaktivno delo, ga v smislu interaktivnosti lahko primerjamo z interpreterskimi (običajno bločno orientiranimi) jeziki. Jezik SIMNON so razvili na Oddelku za avtomatsko vodenje Instituta za tehnologijo v mestu Lund na Švedskem.

Opis modela

Uporabnik opiše model s pomočjo posebnega jezika, seveda ob poznavanju ustreznih metod za simulacijo. Vhodni jezik je enostaven in omogoča tudi neposredno preverjanje vnosa, kajti prevajalnik deluje 'vzporedno' z vnašanjem. Če prevajalnik odkrije napako, takoj javi ustrezno sporočilo, tako da uporabnik lahko nemudoma popravi napačno vrstico programa.

Jezik SIMNON omogoča vnos modela na zelo modularen način, saj je le-tega možno podati s pomočjo podmodelov. To sicer niso pravi hierarhični podmodeli, saj podmodel ne more biti sestavljen iz podmodelov. Podmodele lahko organiziramo v posebnih knjižnicah.

Jezik SIMNON omogoča vključitev zveznih (diferencialne enačbe) in diskretnih (diferenčne enačbe) podmodelov. Čas vzorčenja je lahko različen v različnih podmodelih, lahko se tudi spreminja med simulacijo. Trenutke vzorčenja definira uporabnik s pomočjo posebne spremenljivke v vsakem diskretnem podmodelu posebej. Ob vsakem vzorčnem trenutku se ta spremenljivka ustrezno poveča na vrednost naslednjega vzorčnega trenutka. Izhodi in stanja diskretnega podmodela so zaključena z zadrževalnikom ničtega reda (zero order hold).

Zvezni in diskretni podmodeli imajo naslednjo zgradbo:

```
CONTINUOUS SYSTEM <ime>          DISCRETE SYSTEM <ime>
deklaracije                      deklaracije
prireditve                       prireditve
END                              END
```

V glavi definicije podmodela navedemo tip podmodela in ime podmodela.

Lahko uporabljamo naslednje deklaracije:

```
INPUT   deklaracije vhodnih spremenljivk
OUTPUT  deklaracije izhodnih spremenljivk
TIME    deklaracija neodvisne spremenljivke
STATE   deklaracije stanj (za zvezne in diskretne podmodele)
DER     deklaracije odvodov za zvezne podmodele
NEW     deklaracije predikcij za diskretne podmodele
TSAMP   deklaracija časa vzorčenja diskretnega podmodela
```

Opis podmodela lahko vsebuje tudi parametre in pomožne spremenljivke, ki niso deklarirane.

Parametrom priredimo vrednosti s stavkom, ki ima obliko

<parameter> : <value>

Na enak način priredimo stanjem začetne vrednosti.

Bistveni stavki za opis strukture modela pa so prireditve v obliki

<spremenljivka> = <izraz>

To so stavki, ki dajejo jeziku SIMNON enačbno orientiranost. Z njimi opišemo izhode podmodelov, odvode in predikcije. Razen običajnih prireditvenih stavkov pa lahko uporabljamo tudi 'if-then-else' stavek, kar seveda zelo poveča uporabnost jezika SIMNON. INTEG stavkov za integracijo ne uporabljamo, saj so stanja in odvodi eksplicitno deklarirani in je relacija med njimi jasna. Seveda se stavki za opis modela avtomatsko razvrstijo med prevajanjem.

Razen običajnih prireditvenih stavkov je možno uporabljati številne vgrajene funkcije. Le-te je možno uporabljati tudi v povezovalnem sistemu, s pomočjo katerega uporabnik definira povezave podmodelov v model. Nekatere standardne funkcije so:

ABS(X)	abs. vrednost	LOG(X)	desetiški logaritem
ATAN(X)	arctg $(-\frac{\pi}{2}, \frac{\pi}{2})$	MAX(X,Y)	maksimum spremenljivk
ATAN2(X,Y)	arctg $(-\pi, \pi)$	MIN(X,Y)	minimum spremenljivk x,y
COS(X)	cos. funkcija	MOD(X,Y)	ostanek pri deljenju x z y
EXP(X)	eksp. funkcija	SIGN(X)	predznak spremenljivke x
INT(X)	celi del od x	SIN(X)	sinusna funkcija
LN(X)	naravni log.	SQRT(X)	kvadratni koran
		TAN(X)	tangens

Prav tako so že vključeni nekateri podmodeli:

DELAY (zvezni)	mrtvi čas
FUNC (zvezni)	funkcijski generator
IFILE (diskretni)	čitanje spremenljivk iz datoteke
LOGGER (diskretni)	vzorči spremenljivke in jih shranjuje
NOISE1 (diskretni)	Gaussov ali uniformni naključni generator
OPTA (diskretni)	sistem za optimizacijo

Kako so vhodi in izhodi podmodelov povezani, definira uporabnik v povezovalnem sistemu. To je statični sistem, lahko pa vsebuje časovno spremenljive veličine. Povezovalni sistem ima naslednjo strukturo:

```
CONNECTING SYSTEM <ime>
deklaracije
povezovalna sekcija
```

```
END
```

Povezovalna sekcija vsebuje prireditvene stavke, ki povedo, kako so vhodi in izhodi podmodelov povezani v sistem. Isto spremenljivko lahko uporabljamo v različnih podmodelih. Zato moramo taki spremenljivki v povezovalni sekciji navesti ob imenu tudi ime podmodela

<spremenljivka> [<podmodel>]

Desna stran prireditvenih stavkov sme vsebovati tudi stanja in izhodne spremenljivke podmodelov. Le-te označimo na enak način.

Interaktivne zmožnosti

Jedro modula za interaktivno delo z jezikom SIMNON je knjižnica podprogramov, imenovana INTRAC. INTRAC dekodira uporabnikove ukaze in omogoča pri pisanju precej fleksibilnosti. Velikokrat je namreč možno opustiti kakšne operande. Ob tem se upoštevajo privzete vrednosti. Zaporedju ukazov je možno s pomočjo makra prirediti nov ukaz, ki ima identične lastnosti, kot osnovni ukaz. V definicija makra lahko uporabljamo zanke in razvejitve.

Primer 5.1 Simulacija modela ekološkega sistema roparjev in žrtev v jeziku SIMNON

Pri simulaciji ekološkega sistema roparjev in žrtev lahko direktno uporabimo enačbi (3.8), saj je SIMNON enačbno orientirani simulacijski jezik

$$\begin{aligned} \dot{x}_1 &= a_{11}x_1 - a_{12}x_1x_2 \qquad x_1(0) = x_{10} \\ \dot{x}_2 &= a_{21}x_1x_2 - a_{22}x_2 \qquad x_2(0) = x_{20} \end{aligned} \tag{5.1}$$

Uporabili bomo enake oznake za računalniške spremenljivke, kot v primeru 4.1.

Model lahko opišemo s pomočjo enega zveznega podmodela. Najprej deklarira uporabnik stanja in odvode, nato pa s prireditvami opiše model (strukturo). Na koncu pa je potrebno definirati konstante modela in začetne pogoje. Tako dobimo naslednji simulacijski program:

```
CONTINUOUS SYSTEM PREY_PREDATOR
"deklaracije
state RAB, FOX
der RABDOT, FOXDOT
"prireditve
"struktura modela
RABDOT = A11*RAB - A12*RAB*FOX
FOXDOT = A21*RAB*FOX-A22*FOX
"konstante modela
A11: 5
A12: 0.05
A21: 0.0004
A22: 0.2
"zacetni pogoji
RAB: 520
FOX: 85
END
```

Ko definiramo model, ga prevajamo in simuliramo z naslednjimi ukazi

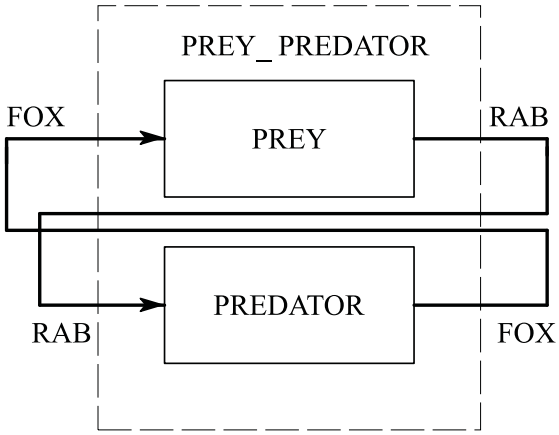
```
syst PREY_PREDATOR
store RAB,FOX
simu 0, 10, 0.01
ashow RAB, FOX
```

Ukaz `syst` prevede model. Z ukazom `store` izbere uporabnik spremenljivke, ki jih želi shranjevati med simulacijo. Ukaz `simu` pa izvrši simulacijski tek. Le-ta poteka od začetnega časa 0 do končnega časa 10 z začetnim računskim korakom 0.01. Ukaz `ashow` izriše po simulacijskem teku shranjene spremenljivke na zaslon.

Da pa bi prikazali še uporabnost podmodelov, bomo simulirali sistem z dvema podmodeloma. Podmodel `PREY` naj vsebuje enačbo za žrtve, podmodel `PREDATOR` pa enačbo za roparje. Oba podmodela sta povezana preko vhodov in izhodov, kakor prikazuje slika 5.6.

Program v jeziku SIMNON vsebuje razen dveh zveznih podmodelov še povezovalni sistem:

```
CONTINUOUS SYSTEM PREY       CONTINUOUS SYSTEM PREDATOR
```



Slika 5.6: Realizacija problema žrtev in roparjev z dvema podmodeloma

```
state RAB                      state FOX
der RABDOT                     der FOXDOT
input FOX                      input RAB
RABDOT=A11*RAB-A12*RAB*FOX     FOXDOT=A21*RAB*FOX-A22*FOX
A11: 5                         A21: 0.0004
A12: 0.05                      A22: 0.2
RAB: 520                       FOX: 85
END                            END

CONNECTING SYSTEM PREY_PREDATOR
FOX [PREY]     =  FOX [PREDATOR]
RAB [PREDATOR] =  RAB [PREY]
END
```

Spremenljivki `RAB` in `FOX` sta izhoda podmodelov `PRAY` in `PREDATOR`. Ker pa sta hkrati tudi stanji, ju ne smemo deklarirati kot izhoda.

V tem primeru izvršimo simulacijo z naslednjimi interaktivnimi ukazi:

```
syst PREY, PREDATOR, PREY_PREDATOR
store RAB [PREY], FOX [PREDATOR]
simu 0, 10, 0.01
ashow RAB [PREY], FOX [PREDATOR]
```

V ukazu `syst` navedemo imeni obeh podmodelov in ime povezovalnega sistema. Spremenljivke, ki se nahajajo v obeh podmodelih, moramo v povezovalnem sis-

temu opremiti z imeni podmodelov. □

5.4 Bločno orientirani simulacijski jeziki

Kot smo omenili v podpoglavju 4.2, so interpreterski jeziki skoraj vedno bločno orientirani, tako da običajno tudi ne razlikujemo med prednostmi in slabostmi interpreterskih in bločno orientiranih jezikov. Bistvene lastnosti slednjih so:

- Bločni jeziki običajno zahtevajo opis modela z relativno enostavnimi gradniki, oz. z diagramom, ki ga sestavljajo vgrajeni bloki.
- Bloki se navadno izvajajo hitreje kot enačbe.
- Strukturo in parametre modela je možno hitro in enostavno spremeniti ter nato ponovno sprožiti simulacijo.
- Bločni jeziki so enostavnejši za začetnike, ki simulirajo le modele z osnovnimi bloki.

Kot primera bločnega jezika bomo opisali simulacijski jezik PSI (Interactive Simulation Program) in Matlabovo simulacijsko okolje Simulink.

5.4.1 Simulacijski jezik PSI

Jezik so razvili v laboratoriju za vodenje sistemov na nizozemski univerzi Delft University of Technology (Van den Bosch, 1987). Jezik je osnovan na FORTRAN-u, deluje pa na različnih računalnikih (seveda tudi na osebnih). PSI je interpreterski jezik.

Opis modela

Vhodni ukazni jezik omogoča kvaliteten interaktivni vnos Osnova za programiranje je simulacijska shema, ki vsebuje relativno enostavne gradnike (bločni diagram). Ker uporabnik ne more vgrajevati svojih blokov, je proizvajalec že vgradil

veliko število blokov, kar daje jeziku precejšnjo kompaktnost. Vsak gradnik (blok) jezika PSI vsebuje en izhod, maksimalno tri vhode in tri parametre. Omogoča prosti format kodiranja in uporabo simboličnih imen, kar ni bilo značilno za starejše bločno orientirane simulacijske jezike.

Na voljo so naslednji standardni bloki:

ABS	absolutna vrednost	LIM	omejilnik
ADC	analogno - digitalna pretvorba	LOG	logaritem
BLN	blok boolove algebre	MAX	maksimum
BNG	bang-bang (vklop - izklop)	MIN	minimum
CON	konstanta	MUL	množilnik
DAC	digitalno - analogna pretvorba	NOI	šum
DIV	delilnik	PDC	zvezni PD regulator
DPY	blok za izpis	PDZ	diskretni PD regulator
DRW	blok za grafični prikaz	PIC	zvezni PI regulator
DSP	mrtva cona	PIZ	diskretni PI regulator
EXP	eksponentna funkcija	REL	rele
FFL	flip - flop	SIN	sinusna funkcija
FIX	kvantizator	SPL	sample & hold
FNG	funkcijski generator	SQT	kvadratni koren
GAI	ojačenje	STP	blok za končanje
HYS	histereza	SUB	odštevalnik
INC	krmiljeni integrator	SUM	seštevalnik
INF	sistem prvega reda	TDE	mrtvi čas
INL	integrator z omejitvijo	XXi	uporabniški blok
INT	integrator	ZZZ	diskretna zakasnitev

Pri pregledu smo izpustili nekatere manj pomembne bloke. Funkcijski generatorji imajo eno ali dve neodvisni spremenljivki, točke pa je možno definirati le ekvidistantno. Med lomnimi točkami lahko uporabljamo linearno ali kvadratično interpolacijo. Nekateri kompleksnejši dinamični bloki, kot npr. sistem 1.reda, PD in PI regulatorji pa omogočajo učinkovito rabo jezika PSI na področju vodenja sistemov. V nekaterih verzijah je možno uporabljati A/D in D/A pretvornike. Vgrajenih je tudi več blokov za integracijo. Krmiljeni integratorji omogočajo simulacijske študije, ki so znane iz hibridnega računanja. Jezik PSI vsebuje tudi algoritem za simulacijo sistemov z algebrajskimi zankami.

Možnosti eksperimentiranja

Razen naštetih zmožnosti, ki so značilne za mnoge bločne jezike, pa omogoča PSI tudi nekatere bolj kompleksne eksperimente. Ti eksperimenti temeljijo na večkratnih simulacijskih tekih:

- Primerjava spremenljivk v različnih simulacijskih tekih ali uporaba spremenljivk predhodnega teka v novem teku.
- Optimizacija zelo poveča uporabnost slehernega simulacijskega orodja. Potrebno je definirati cenilko kot funkcijo enega ali več parametrov. Metoda Hook - Jeves poišče parametre, ki omogočajo minimum ali maksimum cenilke. Lahko se vključijo tudi omejitve.

Interaktivne zmožnosti

Podobno kot SIMNON ima tudi PSI poseben ukazni jezik za interaktivno delo. Ukaze lahko delimo v različne skupine. Nekatere pomembnejše so:

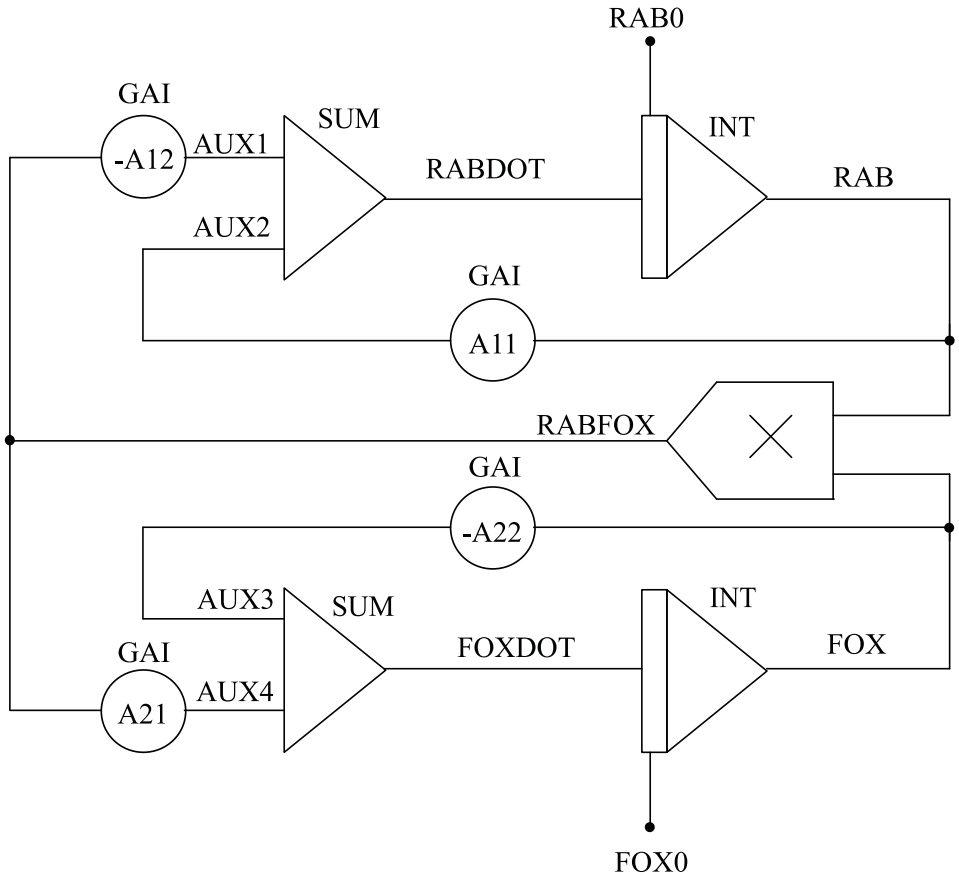
- ukazi za opis modela (opis strukture, parametri, nekatere časovne zahteve, navedba izhodnih zahtev,...),
- ukazi za prikaz parametrov, signalov,
- ukazi za krmiljenje simulacije,
- ukazi za shranjevanje modela,
- ukazi za funkcijske generatorje,
- ukazi za optimizacijo.

Prej omenjena trditev, da je bločni jezik primernejši za neizkušenega uporabnika, ne velja za rabo ukaznega jezika za eksperimentiranje. Ukazni jezik namreč daje v primerjavi z menujsko orientiranim načinom (npr. SIMCOS) prednost prav izkušenim uporabnikom, ki orodje tudi pogosto uporabljajo.

Primer 5.2 Simulacija modela ekološkega sistema roparjev in žrtev v jeziku PSI

Za opis modela v jeziku PSI moramo razviti digitalno simulacijsko shemo. Osnova za to je seveda simulacijska shema, ki smo jo vpeljali v podpoglavju 3.1, razširjena z nekaterimi dodatnimi bloki (ikonami), ki jih uporablja posamezni simulacijski jezik.

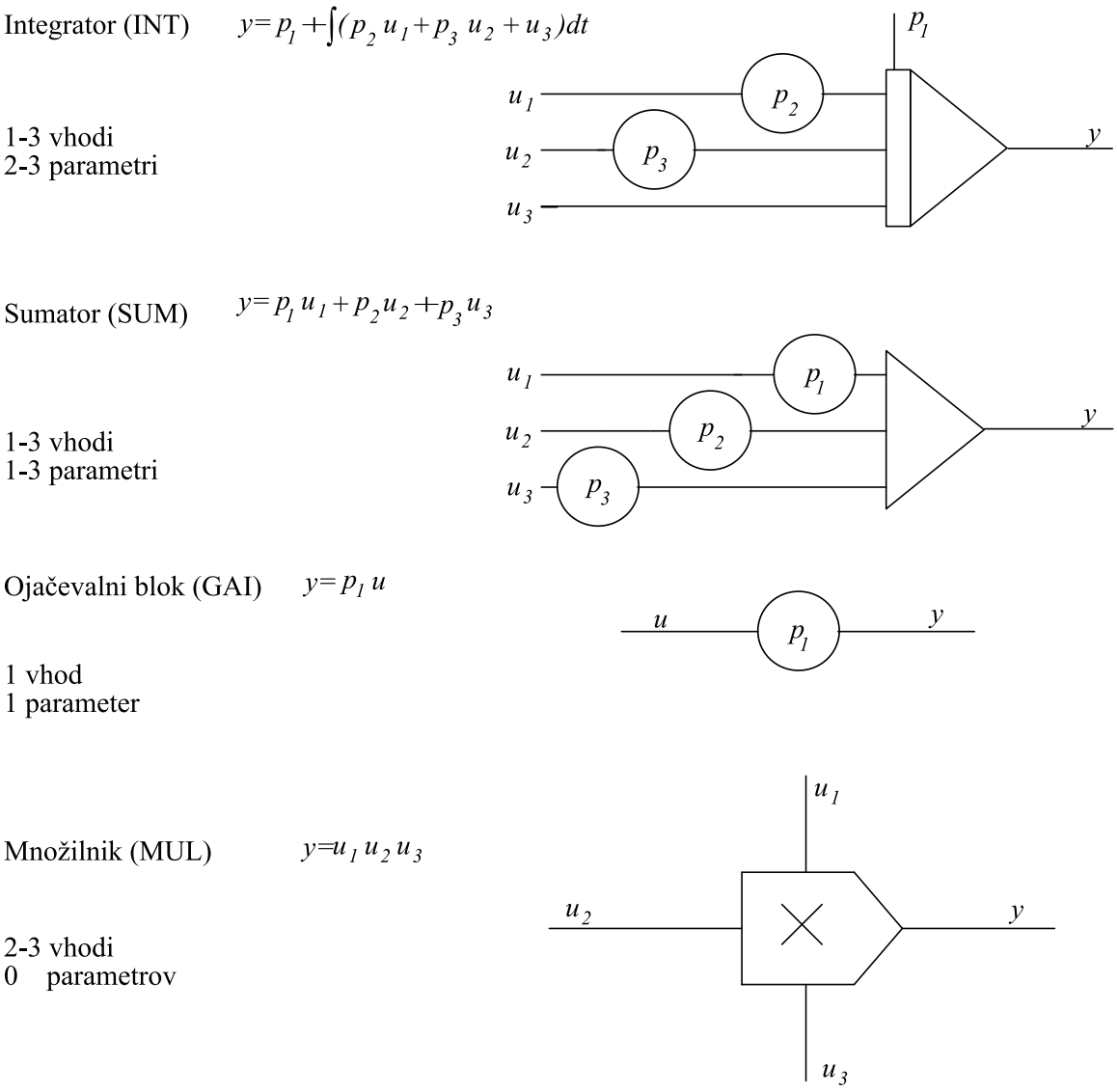
Ponovno narišimo simulacijsko shemo s slike 3.8 in na njej označimo izhode vseh blokov s spremenljivkami in konstantami simulacijskega programa. Relacija med spremenljivkami matematičnega modela in računalniškega programa je enaka kot v primeru 4.1. Ker je potrebno označiti vse izhode, smo vpeljali nekatere nove spremenljivke (`AUX1, AUX2, AUX3, AUX4, RABFOX`). Ustrezno digitalno simulacijsko shemo prikazuje slika 5.7.



Slika 5.7: Simulacijska shema za ekološki sistem

Da realiziramo shemo na sliki 5.7, moramo uporabiti naslednje bloke jezika PSI: dva integratorja (`INT`), dva sumatorja (`SUM`), štiri ojačevalne bloke (`GAI`), ki pred-

stavljajo konstante modela in en množilnik (`MUL`). Ti bloki so prikazani na sliki 5.8, pri čemer uporabljamo naslednje oznake: y je edini izhod vsakega bloka, u_i je i-ti vhod in p_i je i-ti parameter bloka.



Slika 5.8: Osnovni bloki jezika PSI

S pomočjo ukaznega jezika moramo definirati:

- strukturo modela (CONFIGURATION SPECIFICATION PART),

- konstante modela (PARAMETER PART),
- krmilne parametre simulacije.

Strukturo definiramo, potem ko vtipkamo ukaz `B` po zagonu jezika PSI. Opišemo vse bloke v poljubnem vrstnem redu. Vsaka definicija bloka vsebuje: ime izhodne spremenljivke bloka, tip bloka, imena vhodnih spremenljivk.

```
PSI*B
CONFIGURATION SPECIFICATION PART
Block, Type, Input 1, Input 2, Input 3
B*RABDOT, SUM, AUX1, AUX2
B*FOXDOT, SUM, AUX3, AUX4
B*RAB, INT, RABDOT
B*FOX, INT, FOXDOT
B*AUX1, GAI, RABFOX
B*AUX2, GAI, RAB
B*AUX3, GAI, FOX
B*AUX4, GAI, RABFOX
B*RABFOX, MUL, RAB, FOX
```

Konstante definiramo, potem ko vtipkamo ukaz `P`. Vsak blok je tu označen z njegovo izhodno spremenljivko, kateri sledijo parametri.

```
PSI*P
PARAMETER PART
Block, Par 1, Par 2, Par 3
P*RABDOT, 1, 1
P*FOXDOT, 1, 1
P*RAB, 520, 1
P*FOX, 85, 1
P*AUX1,-0.05
P*AUX2, 5
P*AUX3, -0.2
P*AUX4, 0.0004
```

S tem smo definirali enake konstante in začetne pogoje kot v primeru 4.1.

Preden pa poženemo simulacijo, moramo podati še krmilne parametre simulacije in zahteve po prikazu rezultatov. V eni vrstici lahko navedemo več ukazov.

```
PSI*T, O, R
Integration interval = 0.01
Total time = 10
Names of blocks to be shown = RAB, FOX
```

Izbrali smo računski korak 0.01 in trajanje simulacijskega teka 10. Nato se rezultati simulacije (`RAB, FOX`) pojavijo na zaslonu.

□

5.4.2 Simulacijsko okolje Matlab-Simulink

Matlab-Simulink (Simulink, 2009, Oblak, Škrjanc, 2008, Dabney, Harman, 2004) je programski paket, ki omogoča simulacijo dinamičnih sistemov s pomočjo grafičnega bločno orientiranega vnosa. Nastal je kot razširitev paketa Matlab in ponuja določene prednosti pri simulaciji, hkrati pa ohranja vse funkcije, ki jih omogoča Matlab.

Simulink je namenjen predvsem naslednjima dvema načinoma uporabe:

- definiranju modelov in
- analizi modelov.

V praksi se ta dva načina pogosto uporabljata iterativno, ko želimo s spreminjanjem parametrov sistema doseči določeno želeno obnašanje.

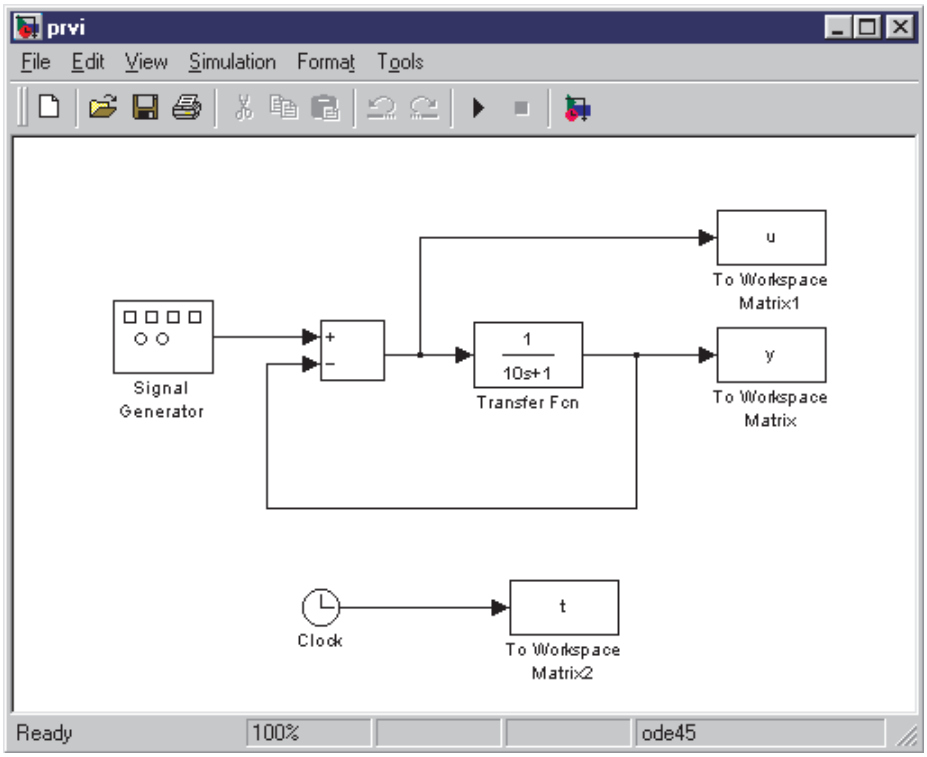
Zaradi uporabniške prijaznosti je paket organiziran v obliki oken, kjer so shranjeni bloki posameznih podsistemov (*block diagram windows*). Načrtovanje novih modelov s pomočjo definiranih podblokov in njihovo editiranje poteka s pomočjo miške. Analiza modela je možna z uporabo ukazov v menuju Simulinka ali pa z uporabo ukazov znotraj ukaznega okna v Matlabu. Vgrajeno orodje za analizo omogoča uporabo različnih simulacijskih algoritmov, `linmod` funkcijo, ki omogoča linearizacijo sistema in `trim` funkcijo, ki izvede analizo ustaljenega stanja. Paket omogoča tudi analizo splošnega linearnega sistema.

Rezultate simulacije lahko opazujemo *on-line* z uporabo prikazovalnikov ali pa na koncu simulacije, znotraj Matlab ukaznega okna.

Opisovali bomo predvsem osnovne lastnosti, ki veljajo v sedanji pa tudi v starejših verzijah.

Kako najhitreje začeti z delom

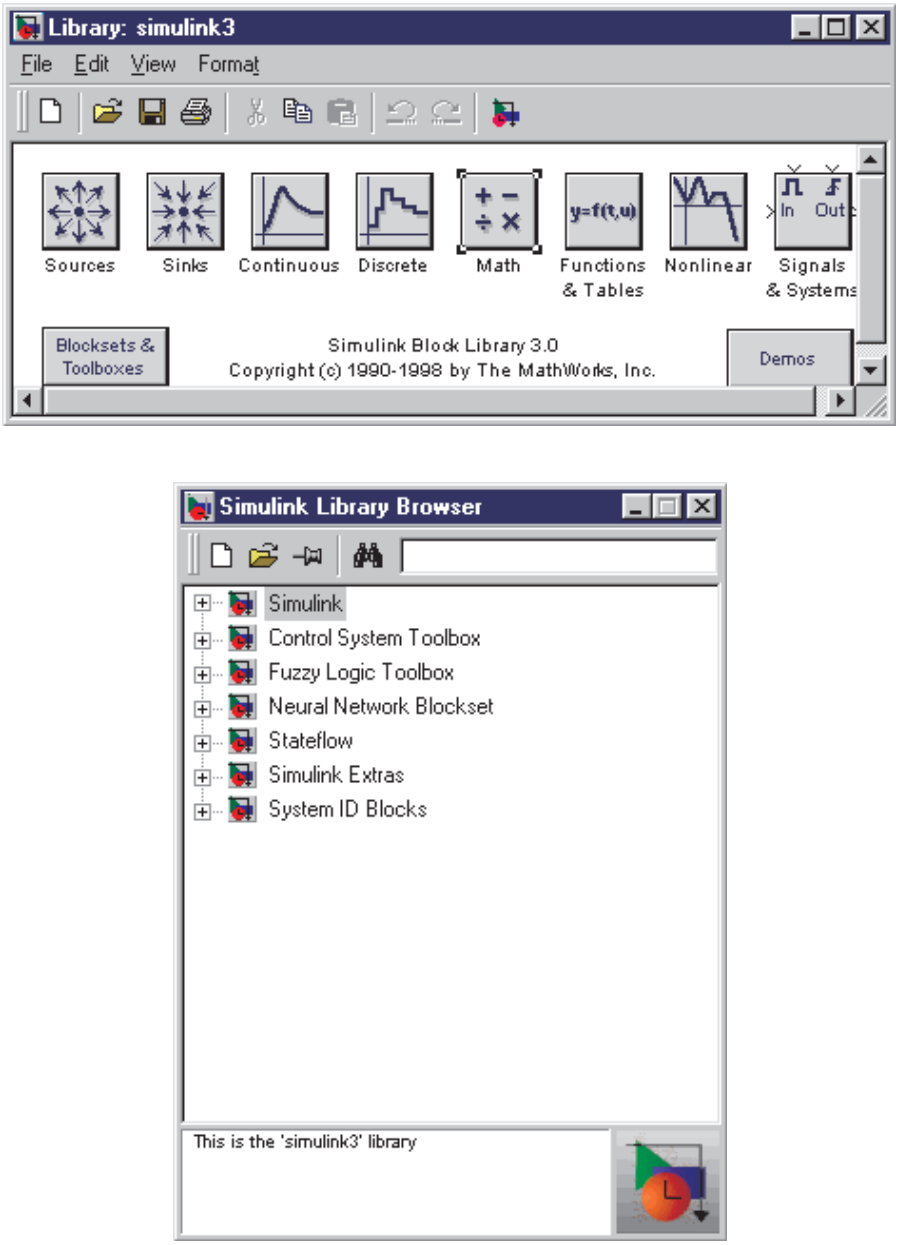
Če želimo kar najhitreje začeti z delom, moramo poznati vsaj nekaj najosnovnejših ukazov. Tak način pristopa je prikazan v nadaljevanju v devetih korakih, v katerih bomo narisali shemo (shranjeno v datoteki s podaljškom `*.mdl`, ki jo prikazuje slika 5.9.



Slika 5.9: Model `prvi.mdl` v Simulinku

1. Znotraj Matlab ukaznega okna zaženemo ukaz Simulink, ki odpre Simulink Library Browser. S pritiskom na desni gumb na izbiri Simulink, odpremo lahko tudi Simulink Block Library okno. Okni sta na sliki 5.10.

2. Izberemo opcijo **New** v menuju **File** Simulink Block Library okna, ki odpre novo prazno okno v katerem kreiramo model. Nov model ima ime *Untitled*. Ime lahko ob shranjevanju spremenimo.



Slika 5.10: Osnovni okni programskega paketa Simulink (knjižnica blokov in brskalnik po knjižnicah

3. Odpremo eno ali več knjižnic podblokov (dvojni klik na želeno knjižnico) in povlečemo (drag) želene podbloke v aktivno okno.

4. Povežemo bloke med seboj tako, da potegnemo črte med izhodi in vhodi v

bloke. Črto potegnemo tako, da kazalec miške pozicioniramo na izhod bloka in s pritiskom levega gumba kazalec premaknemo do želene točke.

5. Odpremo enega ali več blokov (dvojni klik z miško na določen blok) in spremenimo njegove parametre. Parametri morajo biti podani v sintaksi jezika Matlab.

6. Shranimo kreirani model z uporabo ukaza **Save As** v menuju **File**.

7. Zaženemo simulacijo z izbiro ukaza **Start** znotraj menuja **Simulation**. Med izvajanjem simulacije se omogoči menujska opcija **Stop**.

8. Znotraj menuja **Simulation** lahko spreminjamo krmilne parametre simulacije v opciji **Parameters**.

9. Za nadzor sistemskih spremenljivk lahko uporabimo prikazovalnike (**Scope block**) ali bloke, ki zapisuje spremenljivke v delovni prostor Matlaba (**To Workspace**).

Delo z objekti

Izbira objektov

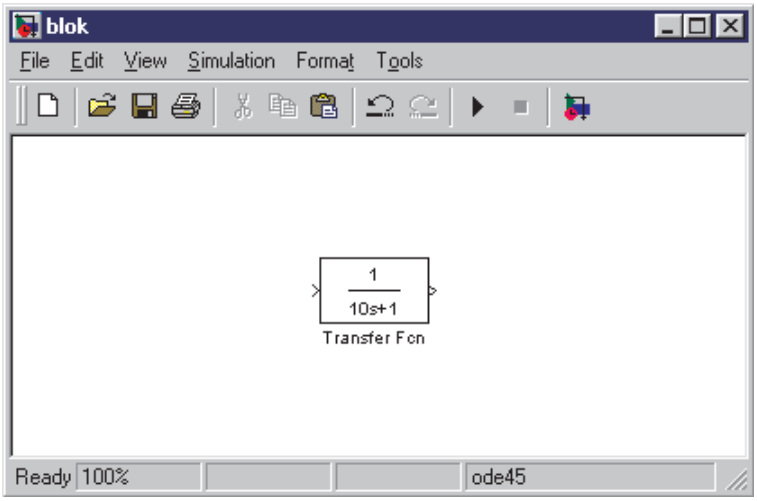
Določene funkcije delujejo le, če objekt izberemo tako, da kliknemo nanj. Izbrani objekt je označen z majhnimi črnimi kvadratki v ogliščih ikone. Za izbiro več objektov je potrebno držati pritisnjeno tipko **Shift** in z levim gumbom izbrati želene objekte. Za izbiro vseh objektov v aktivnem oknu lahko uporabimo ukaz **Select All** v menuju **Edit**.

Množico objektov lahko izberemo tudi tako, da s pritisnjenim levim gumbom označimo polje, kjer se nahajajo objekti, ki jih želimo izbrati.

Delo z bloki

Bloki imajo tako splošne kot specifične lastnosti. Splošne lastnosti so tiste, ki so skupne vsem blokom. Te lastnosti so velikost in pozicija. Specifične lastnosti pa so tiste, ki so lastne samo določenemu bloku. Če preslikamo blok v nov blok se

ohranijo vse lastnosti prednika razen pozicije. Blok - prenosno funkcijo prikazuje slika 5.11.



Slika 5.11: Blok - prenosna funkcija v okolju Simulink

Premikanje in kopiranje blokov. Bloke lahko premikamo ali kopiramo iz enega v drugo Simulink okno z vlečenjem (dragging) s pomočjo miške ali z uporabo ukazov **Cut**, **Copy** ali **Paste** v menuju **Edit**.

Uporaba miške. S kurzorjem izberemo blok, ki ga želimo premakniti in ga s pritisnjenim z levim gumbom izvlečemo na želeno mesto, kjer sprostimo gumb. Če sta prvotna pozicija bloka in končna pozicija bloka v istem aktivnem oknu, potem je rezultat akcije premik bloka. Vse povezave med bloki ostanjo nespremenjene. V primeru, ko je končna pozicija v drugem oknu, se blok kopira na novo pozicijo.

S pritiskom tipke **Ctrl** (ali z uporabo desnega gumba) med to akcijo se blok kopira znotraj aktivnega okna. V tem primeru dobi ime bloka podaljšek v obliki zaporedne številke.

Brisanje blokov. Bloke brišemo tako, da najprej izberemo tiste, ki jih želimo brisati in nato pritisnemo tipko **Delete** ali opcijo **Clear** ali **Cut** v meniju **Edit**.

Spreminjanje velikosti blokov. Dimenzijo bloka lahko spreminjamo z izvlačenjem kvadratkov, ki označujejo izbrani blok.

Editiranje imen blokov. Vsi bloki znotraj okna morajo imeti različna imena. Ime je lahko vidno ali pa nevidno, kar lahko določimo v stilski opciji **Style**.

Ime lahko editiramo tako, da izberemo ime bloka in ga nadomestimo z novim, pozicioniramo kurzor na staro ime in vrinemo novo ali napišemo novo ime nekje v oknu, ga izberemo in shranimo v odložišče (clipboard) s pritiskom tipk **Shift-Delete**, pozicioniramo kurzor na mesto imena in s pritiskom tipk **Shift-Insert** prenesemo novo ime.

Odpiranje blokov. Blok odpremo z dvojnim klikom nanj. Običajno se odpre okno s parametri bloka, ki jih lahko spreminjamo. Na začetku imajo privzete vrednosti.

Spreminjanje stila. Po definiciji potekajo signali skozi blok od leve proti desni. Vhodi v blok so na levi in izhodi iz bloka na desni. Orientacija bloka se lahko spremini z uporabo ukazov **Flip Block** in **Rotate Block**, ki so dostopni znotraj menuja **Format**. **Rotate Block** ukaz obrne blok v smeri urinega kazalca za 90 stopinj, ukaz **Flip Block** pa za 180 stopinj. Znotraj menuja **Fromat** je omogočeno tudi spreminjanje tipa zapisa v možnosti **Font**, obračanje imena bloka **Flip Name**, odstranitev imena z možnostjo **Hide Name** in senčenje blokov v možnosti **Show Drop Shadow**.

Povezovanje blokov

Za povezovanje blokov uporabljamo črte med izhodi in vhodi blokov. Če je vhod nepovezan je izhod vedno enak nič. Število povezav na posameznem izhodu iz bloka je neomejeno, medtem ko je dovoljena le ena linija na vhod bloka. Seveda tudi ne moremo povezovati vhodov med seboj in ravnotako tudi ne izhodov med seboj.

Risanje povezav med bloki. Dva bloka povežemo tako, da se z miško pozicioniramo na izhod določenega bloka, ki ga želimo povezati in s pritisnjenim levim gumbom izvlečemo črto do želenega vhoda v blok. Ko sprostimo levi gumb se na mestu vhoda in izhoda bloka pojavi povezava s puščico, ki kaže smer poteka signala. Povezave med bloki lahko rišemo od izhoda na vhod bloka ali pa tudi obratno. Smer signala ostaja nespremenjena, od izhoda bloka na vhod bloka.

Brisanje povezav. Povezavo brišemo tako, da jo izberemo in nato pritisnemo tipko **Delete** ali pa izberemo ukaz **Clear** ali **Cut** v menuju **Edit**.

Segmentiranje in dodajanje povezav. Če želimo določeno povezavo segmentirati na več lomljenih segmentov, potem se pozicioniramo s kurzorjem na mesto,

kjer želimo lomljeno črto in pritisnemo tipko **Shift**. Hkrati pritisnemo levi gumb in izvlečemo kurzor na želeno mesto.

Novo povezavo lahko kreiramo iz katerekoli točke obstoječe povezave. Povezavo dodamo tako, da se pozicioniramo s kurzorjem na mesto, kjer želimo odjemati signal in s pritisnjenim desnim gumbom izvlečemo novo povezavo. Vrednost signala na izhodu iz povezave je enaka vrednosti na odjemnem mestu. Enako lahko storimo s pritiskom levega gumba in hkratnim pritiskom tipke **Ctrl**.

Povezave med bloki so lahko multipleksirane. Informacijo o multipleksiranih signalih lahko prikažemo na shemi na ta način, da v menuji **Format** izberemo možnost **Vector Line Widths** in odebelimo povezave, ki so multipleksirane **Wide Vector Lines**. Hkrati pa lahko v shemo vnesemo informacijo o tem, kaksen tip podatkov se prenaša po povezavi **Port Data Types**.

Menuji

Simulink je organiziran s pomočjo menujev, ki so dostopni na vrhu delovnega okna. V nadaljevanju bomo predstavili pomembnejše zmožnosti posameznih menujev in možne izbire.

File
New Kreiranje novega sistema
Open Izbira sistema, ki že obstaja nekje na disku.
Close Odstranitev sistema iz delovnega spomina.
Save Zapis sistema na datoteko z istim imenom.
Save As Zapis sistema na novo datoteko.
Model Properties Dokumentacija modela (avtor, datum, opis)
Print Tiskanje bločnega diagrama.
Exit Matlab Izstop.

Edit
Undo Prekliči zadnje spreminjanje sheme.
Redo Ponovi zadnje spreminjanje sheme.
Cut Zapis izbranega objekta v v odložišče in brisanje objekta v aktivnem oknu.
Copy Zapis objekta v v odložišče.
Paste Kopiranje vsebine iz odložišča na izbrano pozicijo.
Delete Brisanje izbranega objekta.
Select All Izbira vseh objektov znotraj aktivnega okna.

Copy Model Kopiranje modela.
Create Subsystem Grupiranje izbranih objektov v podsistemski blok.
Mask Subsystem Kreiranje vmesnika (maske) za blok.
Look Under Mask Pogled pod masko.
Go To Library Link V knjižnico blokov iz katere je določen blok.
Break Library Link Prekinitev povezave s knjižnico (kadar želimo spreminjati atribute bloka, ki smo ga vnesli iz knjižnice).
Unlock Library Spreminjanje atributov blokov v določeni knjižnici.
Update Diagram Osveževanje diagrama.

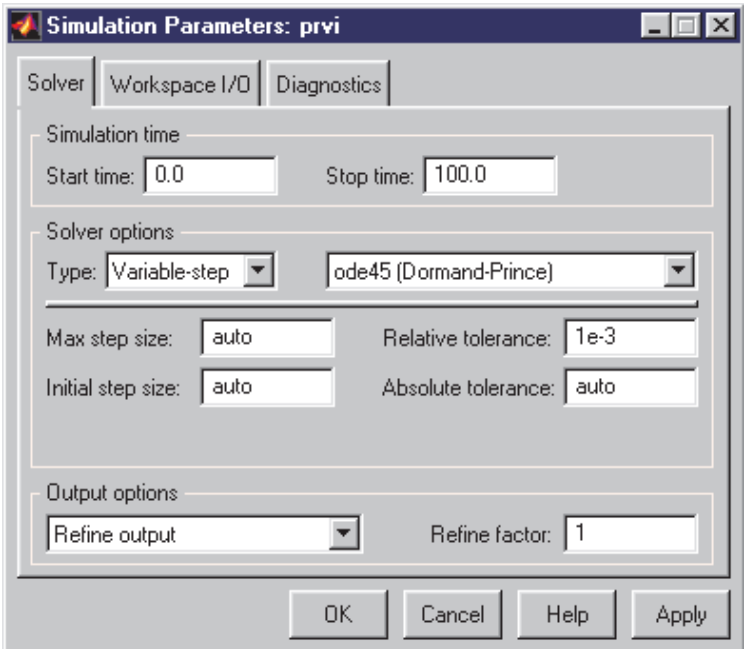
View
Toolbar Izpis orodne vrstice pod menujsko vrstico.
Status Bar Izpis statusne vrstice na spodnjem robu okna. Statusna vrstica vsebuje podatke o simulaciji.
Model Browser Brskalnik modela, ki je uporaben v primeru, ko je model sestavljen iz množice podmodelov.
Block Data Tips Izpis oblačka z atributi nad blokom, ki se ga dotaknemo s kurzorjem.
Library Browser Prikaz knjižničnega brskalnika.
Zoom In Povečava diagrama.
Zoom Out Odstranitev povečave.
Fit System to View Prilagajanje izgleda.
Normal Običajni izgled.

Simulation
Start Zagon simulacije modela v aktivnem oknu.
Stop Prekinitev simulacije.
Configuration parameters Editiranje parametrov simulacije. Nastavitev začetnega in končnega časa simulacije (`Start time`, `Stop time`), nastavitev integracijskega algoritma (Solver options), maksimalnega in minimalnega koraka pri integraciji (`Max step size`, `Min step size`) in relativne ter absolutne napake pri simulaciji (`Relative tolerance`, `Absolute tolerance`).

Med simulacijo se računski korak sproti prilagaja (spreminja med `Min Step Size` in `Max Step Size`), tako da napaka ni večja od dopustne (`Tolerance`). Medtem ko v enostavnejših primerih lahko pustimo `Min Step Size` in `Tolerance` na privzetih (default) vrednostih, pa moramo `Max Step Size` izbrati tako, da bo na opazovanem intervalu (`Stop Time - Start Time`) dovolj izračunanih točk (npr.

```
Max Step Size = (Stop Time - Start Time)/100
```

100 točk). Ustrezno času opazovanja (le ta zavisi od dolžine prehodnih pojavov, časovnih konstant, lastnih in vzbujalnih frekvenc, ...) moramo izbrati tudi parametre blokov za opazovanje.



Slika 5.12: Simulacijski parametri

Format V tem menuju je omogočeno redefiniranje orientacije blokov, kar je bilo opisano že v prejšnjih poglavljih, izgleda blokov (**Foreground Color, Background Color**) in okna (**Screen Color**).

Knjižnice blokov

V tem razdelku si bomo ogledali pomembnejše bloke iz nekaterih knjižnic. Večina blokov omogoča nastavljanje parametrov s pomočjo interaktivnega okna, ki ga odpremo z dvojnim klikom na izbranem bloku. Interaktivno okno vsebuje naslednje elemente:

- ime in tip bloka na vrhu okna,

- kratek opis bloka in njegovo funkcijo,

- polje, kjer lahko vnašamo funkcijske parametre, ki morajo biti v sintaksi MATLABA,

- v spodnjem delu so gumbi **OK** s katerim potrdimo trenutne izbrane vrednosti parametrov, **Cancel**, ki omogoča prekinitev in nastavitev starih vrednosti, ki so bile nastavljene pred vstopom v interaktivno okno in **Help**, ki nudi pomoč uporabniku.

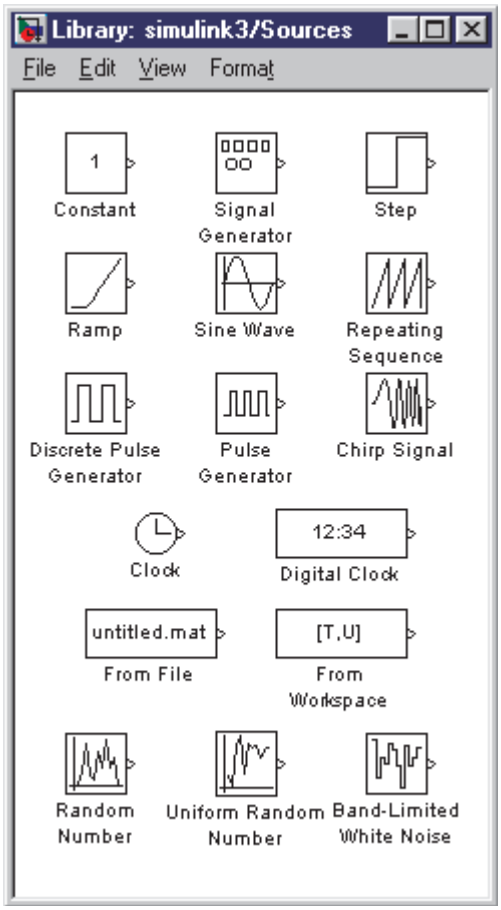
Simulink shranjuje vrednosti parametrov, ki so podane skozi interaktivna okna in jih prenaša v Matlab, ko zaženemo simulacijo.

V naslednjih razdelkih bodo opisane knjižnice blokov, ki so standardne v Simulinku.

Sources

Knjižnica, ki je prikazana na sliki 5.13, vsebuje generatorje (izvore) naslednjih signalov:

Constant Generator konstantne vrednosti.
Signal Generator Generator različnih periodičnih signalov.
Step Generator stopničaste funkcije.
Ramp Generator linearno naraščajočega signala.
Sine Wave Generator sinusnega signala.
Repeating Sequence Ponavljajoča sekvenca izhodnega signala glede na čas.
Discrete Pulse Generator Diskretni pulzni generator.
Pulse Generator Pulzni generator.
Chirp Signal Sinusni signal s spremenljivo frekvenco.
Clock Generira in izpiše sistemski čas.
Digital Clock Generira in izpiše sistemski čas v digitalni obliki.
From File Generiranje signala s pomočjo datoteke v Matlabu.
From Workspace Generiranje signala s pomočjo spremenljivke v Matlabu.
Random Number Generator naključnih števil.
Band-Limited White Noise Generator belega šuma z določeno pasovno širino.

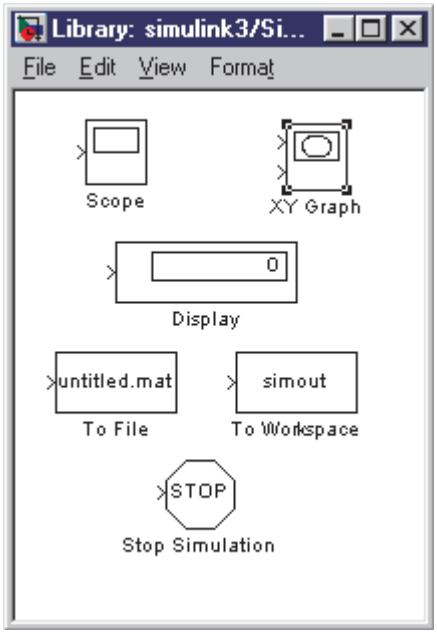


Slika 5.13: Knjižnica izvorov

Sinks

Knjižnica, ki vsebuje prikazovalnike in ponore signalov, je prikazana na sliki 5.14.

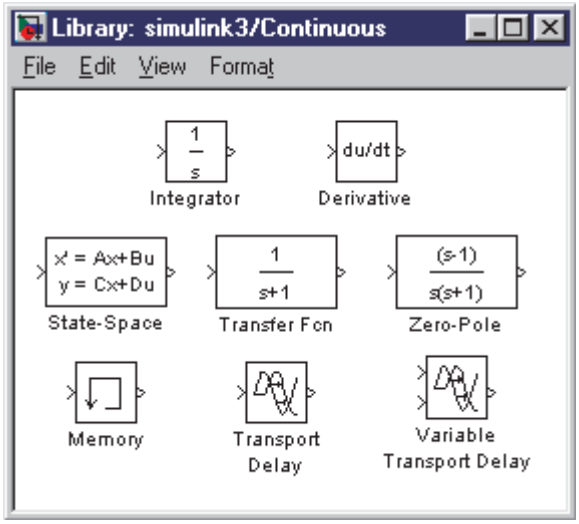
Scope Prikazovalnik signalov med simulacijo.
XY Graph Prikazovalnik v obliki Matlab diagrama.
Display Prikazovalnik vrednosti signala.
To File Zapisovanje signala na datoteko.
To Workspace Zapisuje signala v delovni pomnilnik Matlaba v obliki vektorja.
Stop Simulation Zaključek simulacije pri določenem pogoju.



Slika 5.14: Knjižnica ponorov signalov

Continuous

Knjižnica časovno zveznih blokov je na sliki 5.15.

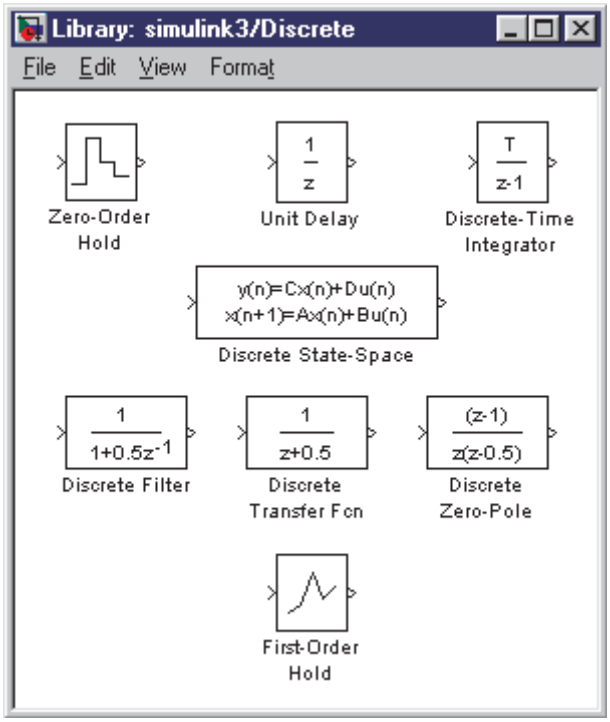


Slika 5.15: Knjižnica zveznih blokov

Integrator Integriranje vhodnega signala.
Derivative Odvajanje vhodnega signala.
State-Space Zvezni model podan v prostoru stanj.
Transfer Fcn Zvezna prenosna funkcija - polinomska oblika.
Zero-Pole Zvezni model podan z ničlami, poli in ojačenjem - faktorizirana oblika.
Memory Zakasnitev enega integracijskega koraka.
Transport Delay Transportni mrtvi čas.
Variable Transport Delay Spremenljiv transportni čas.

Discrete

Slika 5.16, prikazuje knjižnico časovno diskretnih blokov.



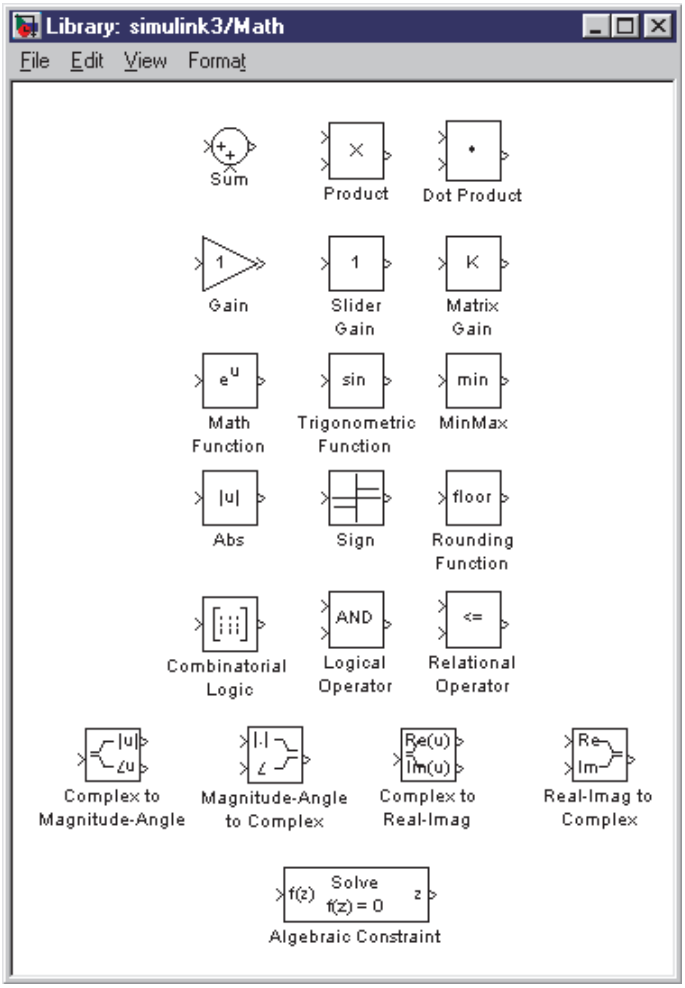
Slika 5.16: Knjižnica diskretnih blokov

Zero-Order Hold Zadrževalnik ničtega reda.
Unit Delay Časovna zakasnitev signala za en vzorčni interval.
Discrete-Time Integrator Diskretni integrator.

Discrete State-Space Diskretni sistem podan v prostoru stanj.
Discrete Filter Filter.
Discrete Transfer Fcn Diskretna prenosna funkcija - polinomska oblika.
Discrete Zero-Pole Diskretna prenosna funkcija podana z ničlami, poli in ojačanjem - faktorizirana oblika.
First-Order Hold Zadrževalnik prvega reda.

Math

Slika 5.17, prikazuje knjižnico matematičnih blokov.

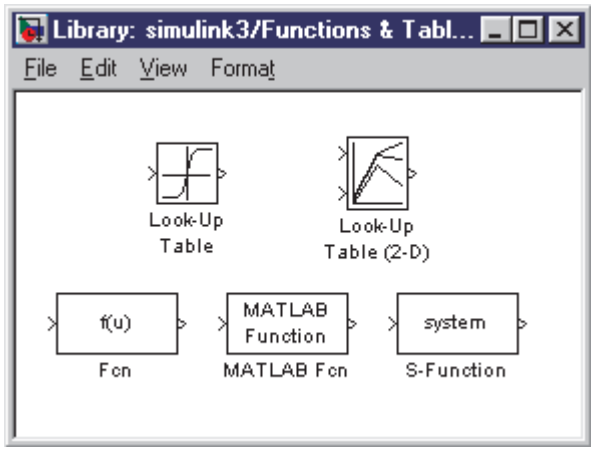


Slika 5.17: Knjižnica matematičnih blokov

Sum Sumator - sešteva ali odšteva signale.
Gain Signal množi s poljubno realno konstanto.
Omenili smo le bloka, ki se uporabljata skoraj v vsaki simulacijski shemi. Funkcije ostalih blokov so razvidne iz ikon.

Functions and Tables

Slika 5.18, prikazuje knjižnico matematičnih funkcijskih in tabelaričnih blokov.



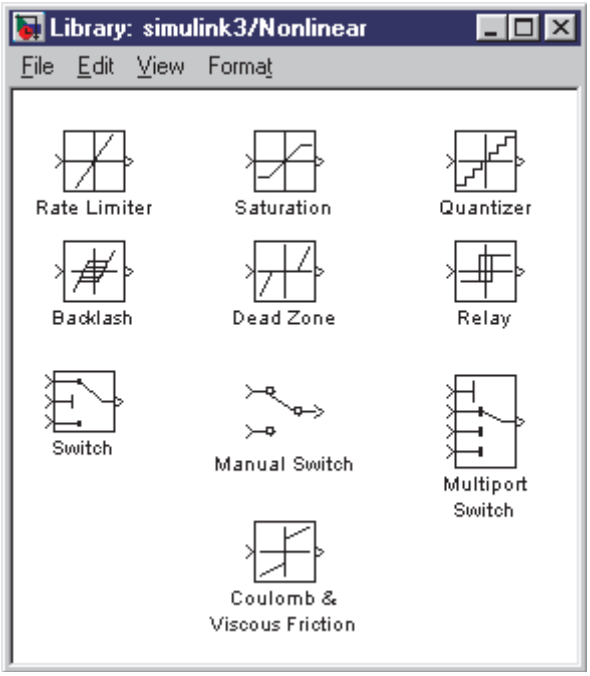
Slika 5.18: Knjižnica matematičnih funkcijskih in tabelaričnih blokov

Look-Up Table Generator funkcije ene neodvisne spremenljivke.
Look-Up Table (2-D) Generator funkcije dveh neodvisnih spremenljivk.
Fcn Realizacija matematičnega izraza v sintaksi Matlaba.
MATLAB Fcn Realizacije Matlab funkcije na vhodu v blok.
S-Function Realizacija S-funkcije v obliki bloka.

Nonlinear

Knjižnico nelinearnih blokov prikazuje slika 5.19.

Funkcije blokov so razvidne iz ikon.



Slika 5.19: Knjižnica nelinearnih blokov

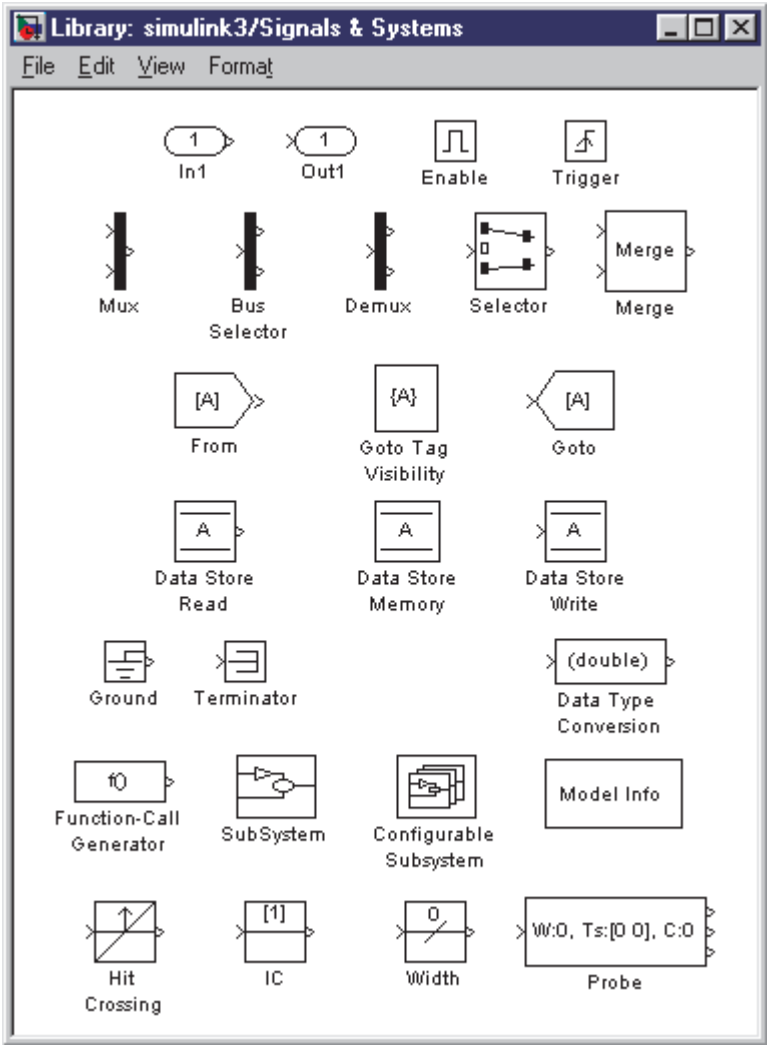
Signals and Systems

Knjižnica blokov, ki omogočajo povezave med objekti znotraj Simulinka je predstavljena na sliki 5.20.

In1 Omogoča povezavo z zunanjim vhodnim portom.
Out1 Omogoča povezavo z zunanjim izhodnim portom.
Mux Združevanje množice skalarnih vrednosti v vektor.
Demux Razdruževanje vektorske vrednosti v več skalarnih. Funkcije ostalih blokov so razvidne iz ikon.

Analiza modelov

Analiziranje modelov zgrajenih v Simulinku je možno na več različnih načinov: z uporabo orodja **Linear Analysis** v izbirni vrstici (**Tools**) ali iz Matlaba z uporabo funkcij `linmod` in `trim`.



Slika 5.20: Knjižnica blokov povezav in sistemov

Linear Analysis

Orodje omogoča linearno analizo dela modela, ki ga označimo z vhodno in izhodno točko, ki se nahajata v oknu **Model Inputs and Outputs**. Pojavi se okno **LTI Viewer** v katerem v izbiri **Simulation** izberemo v kakšni obliki želimo predstaviti linearni model. Predstavitve so možne v časovnem in frekvenčnem prostoru in so naslednje: odziv na stopnico **Step**, impulzni odziv **Impulse**, Bodejev diagram **Bode**, Nyquistov diagram **Nyquist**, Nicholsov diagram **Nichols** in izris polov in ničel v s-ravnini **PZmap**.

Linmod

`Linmod` je funkcija s pomočnjo katere lahko linearizirамо splošni nelinearni sistem diferencialnih enačb okrog delovne točke. Rezultat linearizacije je zapis linearizirанega modela v prostoru stanj.

Če linearizirамо Simulink model, potem moramo definirati vhode in izhode iz dela, ki ga želimo linearizirati s pomočjo blokov **In1** in **Out1**.

Trim

Funkcija `trim` omogoča izračun ustaljenega stanja obravnavanega modela. Ustaljeno stanje lahko izračnavamo pri določeni vrednosti vhoda, lahko izračunavamo vhodni signal in stanja pri znanem izhodu v ustaljenem stanju.

Poglobljena uporaba Simulinka

V tem delu bo natančneje predstavljeno delovanje Simulinka ter zgradba S-funkcij.

Delovanje Simulinka

Gradnja modelov v Simulinku je smiselna predvsem s stališča enostavnosti, učinkovitost izvajanja simulacije pa ni optimalno. Prvi korak procesiranja modela je genereranje podatkovne strukture, ki je optimalna za simulacijo. Vljučijo se ustrezni bloki, ki se tudi razvrstijo. Simulacija poteka s pomočjo numerične integracije diferencialnih enačb. Simulacijo lahko zaganjamo iz Simulink okna ali pa iz Matlab ukaznega okna z uporabo ukaza `sim`.

S-funkcije

Ko zgradimo Simulink model in ga shranimo, sistem zgradi t.i. S-funkcijo, ki je dostopna iz Matlaba. V tej funkciji je definirana dinamika modela. Tako definiran

model lahko integriramo, lineariziramo in iščmo ravnotežne točke modela. S-funkcija deluje enako kot katerakoli Matlab funkcija, ima pa določeno sintakso:

sys=proces(t,x,u,flag)

kjer je proces ime modela in parameter flag določa informacijo, ki jo funkcija vrne v izhodno spremenljivko sys. Na primer ob vrednosti flag=1 funkcija vrne odvode stanj modela pri vektorju stanj x in vhodni spremenljivki u. S-funkcijo lahko napišemo tudi v obliki navadne M-datoteke ali MEX-datoteke, ki je C-jevska ali Fortranska funkcija. Tako lahko ločimo tri različne načine pisanja S-funkcij:

- grafični GUI Simulink,
- M-datoteka Matlab,
- MEX-datoteka C ali Fortran funkcija.

Način podajanja S-funkcij je odvisen od zahtevnosti podajanja, uporabe S-funkcije in hitrosti, ki je zahtevana za izvajanje.

Uporabniško definirano S-funkcijo lahko uvedemo v Simulink shemo tako, da jo pridružimo posebnemu bloku (S-function block). Z maskiranjem lahko takemu bloku dodamo še običajni vmesnik, ki ga imajo ostali Simulink bloki. Na ta način je uporabniku omogočeno lastno generiranje modelov.

Kako delujejo S-funkcije? Funkcije so zgrajene tako, da omogočajo izgradnjo modelov za reševanje časovno zveznih, diskretni in hibridnih problemov. Za to ima S-funkcija točno določeno strukturo. Ključni element S-funkcije je parameter **flag**, ki definira informacijo, ki jo daje funkcija na izhodu:

- flag=0 S-funkcija vrne velikost parametrov in začetnih pogojev,
- flag=1 S-funkcija vrne odvode stanj dx/dt,
- flag=2 S-funkcija vrne diskretna stanja x(k+1),
- flag=3 S-funkcija vrne izhodno spremenljivko,
- flag=4 S-funkcija vrne čas naslednjega časovnega intervala.

Vsaka od vrednosti zastavice posreduje del informacije, ki je potrebna za izvajanje simulacije.

V primeru flag=0 kličemo funkcijo takole:

```
>>flag=0
>>[sizes,x0]=sfun([],[],[],flag)
```

Vektor x0 je vektor začetnih stanj, v vektorju **sizes** pa je naslednja informacija:

- sizes(1) število zveznih stanj,
- sizes(2) število diskretnih stanj,
- sizes(3) število izhodnih spremenljivk,
- sizes(4) število vhodnih spremenljivk,
- sizes(5) število nezveznosti,
- sizes(6) število algebrajskih zank.

V primeru zveznega sistema sta za simulacijo potrebni le informaciji o odvodih stanj v vsakem trenutku simulacije (flag=1) in izhodi sistema (flag=3). Opozoriti velja, da se za razliko od klica funkcije z zastavico flag=0, ostali klici pojavljajo v vsakem računskem koraku simulacije.

Pretvorba S-funkcije v Simulink blok. Katerokoli obliko S-funkcije lahko pretvorimo v Simulink blok. V ta namen je potrebno v funkcijski blok **S-function block** oz. v njen uporabniški vmesnik vpisati ime funkcije in parametre. Blok namreč omogoča prenos določenih parametrov, ki sledijo formalni strukturi parametrov t, x, u in flag. Medsebojno jih ločimo z vejico. Na koncu postopka običajno tak blok tudi maskiramo.

Izgradnja S-funkcije v obliki M-datoteke je najlažja, če si pomagamo s predlogo **sfuntmpl**.

5.5 Simulacija s pomočjo Matlabovih funkcij

Razen simulacije v Simulinku nudi Matlab in dodatek Control Toolbox še nekaj drugih možnosti:

- odziv linearnih sistemov na enotino stopnico s pomočjo funkcije `step` ali odziv na delta impulz s pomočjo funkcije `impulse`,
- simulacijo linearnih časovno nespremenljivih sistemov s pomočjo funkcije `lsim`,
- simulacijo linearnih in nelinearnih, časovno nespremenljivih in časovno spremenljivih sistemov s pomočjo vgrajenih funkcij za numerično integracijo,
- simulacijo Simulink modela iz okolja Matlab s pomočjo funkcije `sim`.

5.5.1 Določitev odziva linearnega sistema s funkcijama `step` in `impulse`

Funkcija `step` izračuna odziv na enotino stopnico linearnega sistema (LTI), ki je opisan s prenosno funkcijo v polinomski (`tf`) ali faktorizirani obliki (`zpk`) ali v prostoru stanj (`ss`).

`step(sys)`
izračuna in nariše odziv sistema `sys` na enotino stopnico. Časovno območje in časovni korak se izbereta avtomatsko.

`step(sys,tf)`
izračuna in nariše odziv na časovnem intervalu `[0 tf]`.

`step(sys,t)`
izračuna in nariše odziv v trenutkih, ki jih določa vektor `t`, npr. `t=0:dt:tf`

`y=step(sys,t)`
izračuna se odziv `y` v trenutkih, ki jih določa vektor `t`, npr. `t=0:dt:tf`. Funkcija ne nariše odziva, pač pa ga lahko narišemo z ukazom `plot(t,y)`.

`[y,t]=step(sys)`
izračuna odziv `y` v trenutkih, ki jih določa `t`. Funkcija ne nariše odziva, pač pa

ga lahko narišemo z ukazom `plot(t,y)`. Časovno območje in časovni korak se izbereta avtomatsko.

`[y,t]=step(sys,tf)`
izračuna odziv `y` v trenutkih, ki jih določa `t` na časovnem intervalu `[0 tf]`. Funkcija ne nariše odziva, pač pa ga lahko narišemo z ukazom `plot(t,y)`.

`[y,t,x]=step(sys)`
funkcija vrne še stanja `x`, kar je predvsem uporabno pri sistemih v prostoru stanj.

Na podoben način se uporablja funkcija `impulse` za določitev odziva na delta impulz.

Primer 5.3 Primer prikazuje izračun in izris odziva prenosne funkcije sistema 2. reda z ojačenjem 2, lastno frekvenco 1 in dušilnim koeficientom 1 na enotino stopnico. Čas opazovanja je 10s. Program v Matlabu

```
G=tf([2],[1 2 1]);
tf=10;
[y,t]=step(G,tf);
plot(t,y)
```

nariše sliko 5.21.

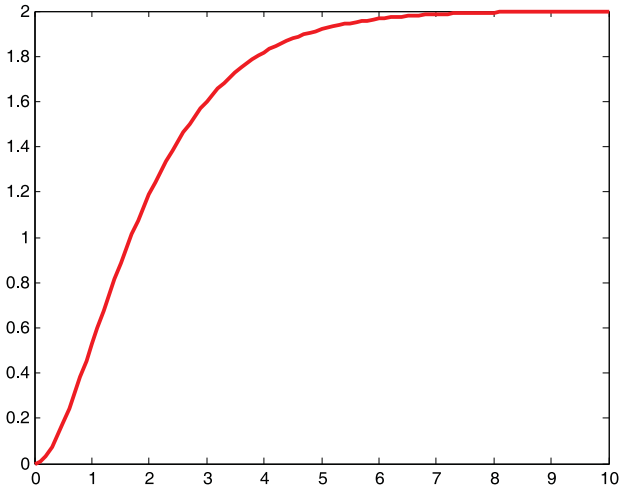
□

5.5.2 Simulacija s funkcijo `lsim`

S funkcijo `lsim` simuliramo zvezne (ali diskretne) dinamične sisteme, ki jih v Matlab vnesemo s funkcijami `ss` (prostor stanj), `tf` (prenosna funkcija v polinomski obliki) in `zpk` (prenosna funkcija v faktorizirani obliki). To so t.i. LTI modeli (linear time invariant model). Vhodni signal definiramo s pomočjo numerično podane funkcijske odvisnosti (s točkami podana funkcija).

Brez argumentov na levi strani enačaja lahko uporabljamo naslednji obliki:

```
lsim(sys,u,t)
lsim(sys,u,t,x0)
```



Slika 5.21: Odziv s pomočjo funkcije `step`

Funkcija `lsim` na zaslonu nariše odziv LTI modela. Pri tem je `sys` sistem, ki ga podamo s fukcijami `ss`, `tf` ali `zpk`. Par `u` in `t` je sestavljen iz dveh vektorjev, ki določata vhodni signel *u(t)*. `t` torej določa trenutke, v katerih je določen vhodni signal, `u` pa so vrednosti signala v ustreznih trenutkih. Trenutke lahko definiramo z naslednjo Matlabovo definicijo vektorja:

```
t = 0:dt:t_koncni
```

sinusni signal pa z

```
u=sin(t)
```

Vektorja `u` in `t` morata imeti enako število elementov (`length(t)=length(u)`).

Med dvema zaporednima točkama vhodnega signala se upošteva linearna interpolacija.

`x0` je matrika začetnih stanj (vsaka kolona opisuje časovni potek enega stanja). Predvsem se začetna stanja uporabljajo v zvezi z modelom v prostoru stanj.

Če kličemo funkcijo `lsim` le z enim argumentom - imenom modela

```
lsim(sys)
```

se odpre uporabniški vmesnik - Linear Simulation Tool. Uporabniški vmesnik za-

hteva vnos vhodnega signala in začetnih stanj, nakar lahko poženemo simulacijo.

Lahko pa navedemo tudi spremenljivke na levi strani enačaja:

```
y = lsim(sys,u,t)
[y,t] = lsim(sys,u,t)
[y,t,x] = lsim(sys,u,t)
[y,t,x] = lsim(sys,u,t,x0)
```

`y` je odziv LTI sistema `sys`. Če na levi strani enačaja navedemo `t`, potem se v določenih primerih lahko dogodi, da se vektor `t` po simulaciji spremeni (npr. v primeru, če je opisan z malo elementi). `x` je matrika stanj (vsaka kolona opisuje časovni potek enega stanja), `x0` pa matrika začetnih stanj. Slednja argumanta imata pomen predvsem pri zapisu modela v prostoru stanj.

Primer 5.4 Primer prikazuje določitev odziva sistema 2. reda z ojačenjem 2, lastno frekvenco 1 in dušilnim koeficientom 1 na sinusni vhodni signal. Čas opazovanja je 10s. Z uporabo programa v jeziku Matlab

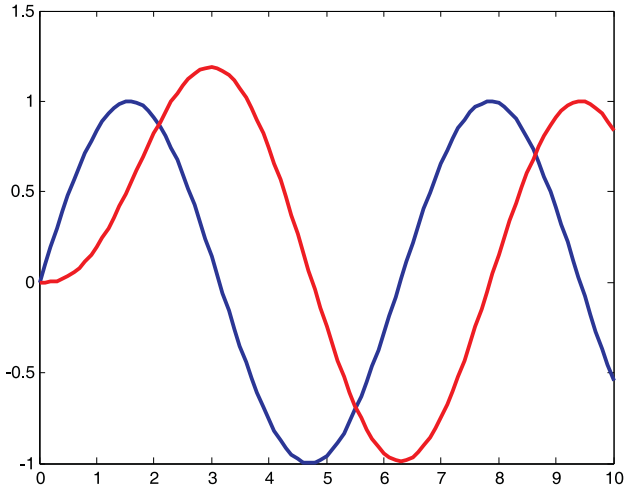
```
G=tf([2],[1 2 1]);
t=0:0.1:10;
u=sin(t);
y=lsim(G,u,t);
plot(t,u,t,y)
```

□

dobimo sliko 5.22.

5.5.3 Simulacija s pomočjo vgrajenih funkcij za numerično integracijo

Splošnejši način simulacije, ki se ne omejuje na linearne in časovno nespremenljive sisteme, poteka preko vgrajenih funkcij za numerično integriranje. Bolj matematično imenujemo postopek začetnovrednostni problem. Koncept je najlaže razumljiv, če si predstavljamo, da je nelinearni dinamični sistem zapisan v



Slika 5.22: Odziv s pomočjo funkcije `lsim`

prostoru stanj, torej z diferencialnimi enačbami 1. reda, oz. da so vsi odvodi stanj sistema odvisni od stanj, vhodnega signala in neodvisne spremenljivke simulacije. Včasih je zelo ilustrativen način, da za nelinearni sistem razvijemo simulacijsko (bločno) shemo, v njej označimo stanja in odvode stanj in iz sheme napišemo enačbe stanj.

Sintakso opisujejo naslednje vrstice:

```
[T,Y] = solver(@odefun,tspan,y0)
[T,Y] = solver(@odefun,tspan,y0,options)
```

kjer je `solver` ena izmed integracijskih funkcij `ode45`, `ode23`, `ode113`, `ode15s`, `ode23s`, `ode23t` ali `ode23tb`.

Vhodni argumenti imajo naslednji pomen:

`odefun`
Funkcija ovrednoti trenutne vrednosti odvodov (`dx`), to je levo stran enačb stanj, iz trenutnih stanj (`x`) in trenutne vrednosti neodvisne spremenljivke (`t`).

```
function dx=odefun(t,x)
dx(1)=...
dx(2)=...
dx(3)=...
```

```
dx=[dx(1) dx(2) dx(3) ...]'
...
end
```

`tspan`
Če ima vektor `tspan` 2 elementa, potem opisuje simulacijski (integracijski) interval `[t0,tf]`. Integracijska metoda vsili začetne pogoje v trenutku `t0` in nato izvrši integracijo do trenutka `tf`. Vmesne trenutke določa integracijska metoda. Če pa želimo dobiti rezultate simulacije v predpisanih trenutkih, navedemo te trenutke v vektorju `tspan`: `tspan = [t0,t1,...,tf]`. V tem primeru mora torej vektor `tspan` imeti več kot dva elementa. Vmesni trenutki pa so le trenutki, v katerih dobimo rezultate. Integracijske metode sicer določijo svoj korak, ki omogoča, da napaka pri simulaciji ni večja od dopustne.

`y0`
Vektor začetnih pogojev.

`options`
Integracijska metoda omogoča številne nastavitve, ki imajo sicer privzete vrednosti. Nastavitve definiramo s funcijo `odeset`. Glavne nastavitve zadevajo dopustno relativno `RelTol` (`1e-3` je privzeta vrednost) in absolutno napako `AbsTol` (`1e-6` je privzeta vrednost) ter razne omejitve v zvezi z računskim korakom.

Izhodni argumenti imajo naslednji pomen:

`T`
Časovni trenutki, v katerih dobimo rezultate.

`Y`
Matrika rezultatov. V kolikor je rezultet en signal, je to vektor, sicer vsaka kolona pomeni en signal.

Integracijski algoritmi

`ode45`
Eksplicitna enokoračna metoda Runge-Kutta, ki za sprotno vrednotenje napake kombinira metodo 4 in 5 reda (Dormand Prince). Je najbolj univerzalna metoda in zato s to metodo najprej poskusimo rešiti problem.

`ode23`

Eksplicitna enokoračna metoda Runge-Kutta, ki za sprotno vrednotenje napake kombinira metodo 2 in 3 reda (Bogacki Shampine). Je bolj učinkovita, če se ne zahteva velika natančnost. Uporabna tudi za srednje toge sisteme.

`ode113`
je večkoračna metoda Adams-Bashforth-Moulton spremenljivega reda. Je učinkovitejša od ode45 pri strogih zahtevah za točnost, pa tudi v primerih kompleksnih izrazov za izračun odvodov. Ker je večkoračna metoda, uporaba ni priporočljiva, če nastopajo nezveznosti v signalih. V tem primeru so primernejše enokoračne metode (npr. `ode45, ode23`).

Zgornji algoritmi se predvsem uporabljajo za netoge sisteme. Če se čas simulacije zelo poveča, je verjetno, da gre za togi sistem (zelo različni velikostni razred časovnih konstant) in v tem primeru se izplača poizkusiti z eno od naslednjih metod: `ode15s, ode23s, ode23t, ode23tb`. S temi metodami dosežemo primerno hitrost simulacije, ne pa tudi velike natančnosti. Običajno med temi metodami največjo natančnost zagotavlja metoda `ode15s`.

Več o integracijskih metodah opisuje poglavje 6.1.

Primer 5.5 Model ekološkega sistema, v katerem nastopajo roparji in žrtve, smo predstavili v primeru 1.3. Matematični model ima obliko

$$\begin{aligned} \dot{x}_1 &= a_{11}x_1 - a_{12}x_1x_2 \qquad x_1(0) = x_{10} \\ \dot{x}_2 &= a_{21}x_1x_2 - a_{22}x_2 \qquad x_2(0) = x_{20} \end{aligned} \tag{5.2}$$

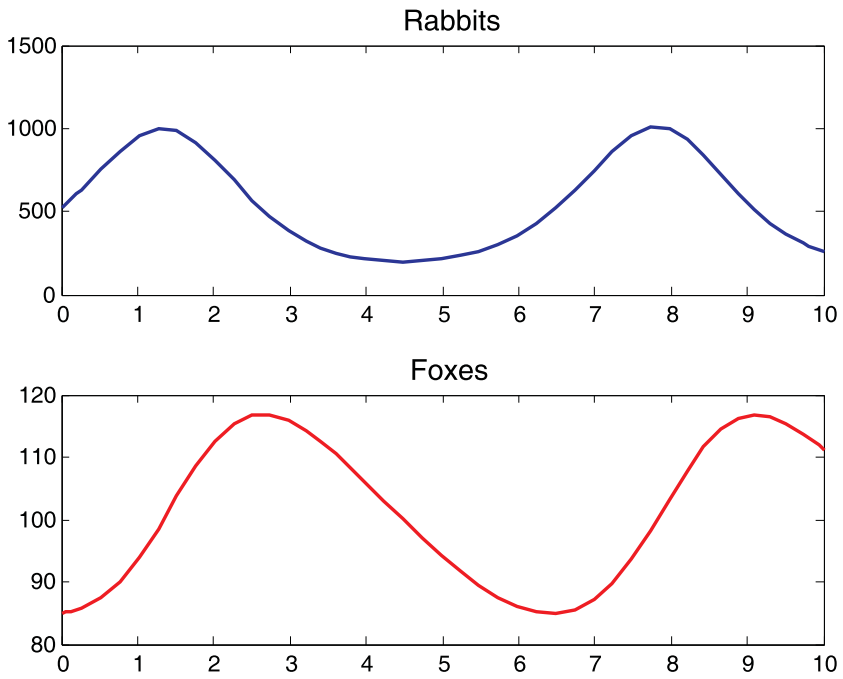
Sedaj izvedimo simulacijo v programskem okolju Matlab. Glavni program ima obliko:

```
rab0=520; % initial no. of rabbits
fox0=85;  % initial no. of foxes
tfin=10;
[T,Y]=ode45(@lhotka,[0 tfin], [rab0 fox0]);
subplot(2,1,1)
plot(T,Y(:,1))
title('Rabbits')
subplot(2,1,2)
plot(T,Y(:,2))
title('Foxes')
```

Funkcija `lhotka` pa je naslednja:

```
function dx=lhotka(t,x)
  a11=5;
  a12=0.05;
  a21=0.0004;
  a22=0.2;
  % calculation of derivatives
  dx(1)=a11.*x(1)-a12.*x(1).*x(2);
  dx(2)=a21.*x(1).*x(2)-a22.*x(2);
  dx=[dx(1) dx(2)]'
end
```

Rezultate simulacije prikazuje slika 5.23. □



Slika 5.23: Rezultati simulacije

5.5.4 Simulacija s funkcijo `sim`

S funkcijo `sim` zaženemo model, ki je opisan s Simulink shemo. Tako lahko v okolju Matlab programiramo kompleksne eksperimente, ki vključujejo simulacijske teke modela v Simulinku. Ponavadi je koristno, da modelu v Simulinku z out bloki iz knjižnice Sinks označimo izhode oz. signale, ki jih opazujemo.

`sim('model')`
ukaz simulira Simulink model `model.mdl`. Veljajo krmilni parametri, kot so nastavljeni v Simulink shemi.

`[t,x,y]=sim('model', timespan)`
`t` je vektor, ki vsebuje trenutke, v katerih se izračunajo rezultati simulacije. `x` je matrika stanj (vsaka kolona opisuje eno stanje)- stanja so izhodi integratorjev. `y` je matrika izhodov (vsaka kolona opisuje en izhod, izhodi so določeni z out bloki v Simulink shemi). `timespan` je v splošnem vektor. Če ima vektor en element, je to končni čas simulacije `tf`, če ima dva elementa, sta to začetni in končni čas simulacije `[t0 tf]`, če pa je več elementov, so to trenutki, v katerih želimo dobiti rezultete simulacije `[t0 t1 t2 ... tf]`.

`[t,x,y]=sim('model', timespan, options)`
options vsebuje vse mogoče nastavitve, ki se sicer lahko nastavijo v Simulink shemi, npr. tolerance, integracijsko metodo, minimalni in maksimalni dopustni računski korak,... Za nastavitev parametra `options` se uporablja posebna fukcija `simset`. Če želimo izbrati integracijsko metodo `ode45`, uporabimo ukaz

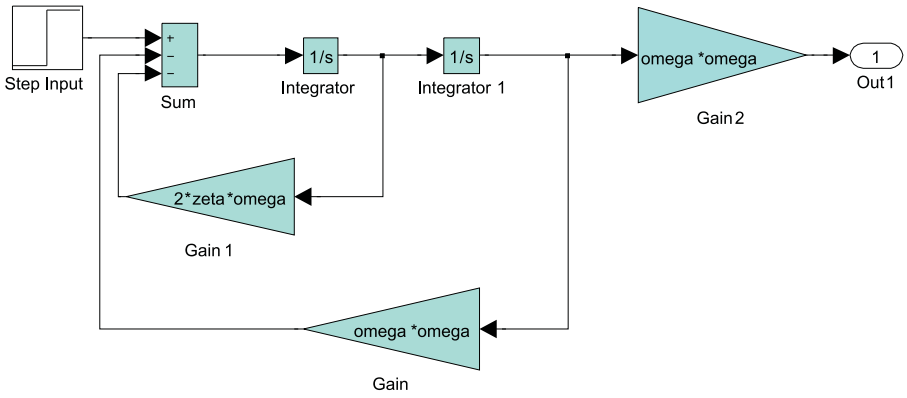
`options=simset('solver','ode45').`

Primer 5.6 Primer prikazuje parametrizacijo sistema 2. reda pri različnih vrednostih parametra ζ. Parameter ζ spreminjamo od 0 do 2 s korakom 0.1. Odziv sistema določimo s pomočjo funkcije `step` in s pomočjo funkcije `sim`. Slednja požene model `secor1.mdl` v Simulinku. Prepričamo se, da dobimo po obeh metodah enake rezultate. Program v jeziku Matlab

```
omega=1;
tf=20;
for zeta=0:0.1:2;
    [t,x,y]=sim('secor1',tf);
```

```
    Gs=tf(1,[1 2*zeta*omega omega*omega]);
    [y1,t1]=step(Gs,tf);
    subplot(2,1,1);
    hold on
    plot(t,y,'r');
    subplot(2,1,2);
    plot(t1,y1,'b');
    hold on;
end
```

ki uporabi Simulink shemo `secor1.mdl`, ki jo prikazuje slika 5.24.



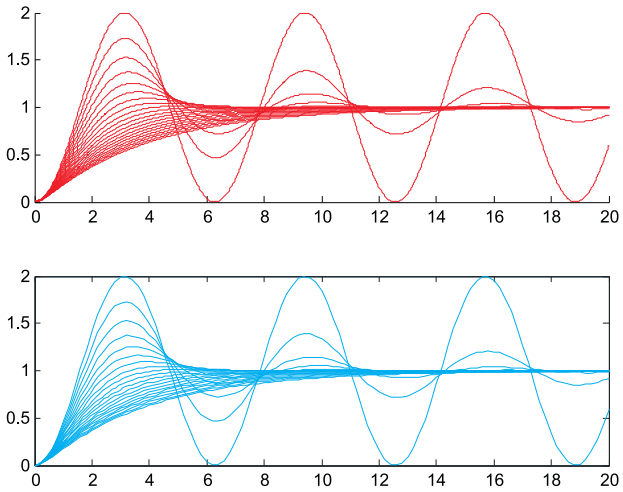
Slika 5.24: Simulink shema sistema 2.reda

nariše sliko 5.25.

□

5.6 Simulacija s splošnonamenskimi programskimi jeziki

Sliki 3.23 in 3.24 prikazujeta strukturo simulacijskega programa, ki temelji na obravnavanem konceptu digitalnih simulacijskih sistemov (glej podpoglavje 3.8). Za izvedbo lahko uporabimo katerikoli splošnonamenski programski jezik oz.



Slika 5.25: Parametrizacija sistema 2. reda

okolje: Matlab, Basic, Visual Basic, C++, Fortran, Java script, Java, ASP, Phyton, Visual Studio, itd.

Primer 5.7 Simulacija ekološkega sistema žrtev in roparjev v jeziku Matlab

Ekološki problem žrtev in roparjev smo uvedli v primeru 1.3. Opisujeta ga diferencialni enačbi

$$\begin{aligned} \dot{x}_1 &= a_{11}x_1 - a_{12}x_1x_2 \qquad x_1(0) = x_{10} \\ \dot{x}_2 &= a_{21}x_1x_2 - a_{22}x_2 \qquad x_2(0) = x_{20} \end{aligned} \tag{5.3}$$

Glede na splošno strukturo simulacijskega programa(slika 3.23) definiramo najprej podatkovno bazo modela in krmilne parametre simulacije. Podatkovna baza modela sestoji iz štirih konstant $(a_{11}, a_{12}, a_{21}, a_{22})$ in dveh začetnih pogojev $(x_1(0), x_2(0))$. Krmilni parametri simulacije pa so: začetna vrednost neodvisne spremenljivke, t.j. začetni čas, računski korak (t.j. prirastek neodvisne spremenljivke med simulacijo) in končni čas simulacije.

Upoštevali bomo naslednjo relacijo med računalniškimi in problemskimi spremenljivkami:

x_1	`rab`	$\dot{x}_1$	`rabdot`
x_2	`fox`	$\dot{x}_2$	`foxdot`
a_{11}	`a11`	a_{12}	`a12`
a_{21}	`a21`	a_{22}	`a22`

Konstante modela lahko uvedemo direktno v enačbe, ki opisujejo matematični model. Toda če želimo konstante med simulacijskimi teki spreminjati, jim je potrebno dati ime, oz. uvesti ustrezne spremenljivke. Spremenljivke inicializiramo s prireditvenimi stavki

```
a11=5;
a12=0.05;
a21=0.0004;
a22=0.2;
```

Na enak način definiramo začetni čas simulacije `t0`, računski korak `dt` in trajanje simulacijskega teka `tfin`

```
t0=0;
dt=0.01;
tfin=10;
```

`dt` moramo izbrati glede na karakteristične dinamične lastnosti modela tako, da se izognemo numerični problematiki in zagotovimo predpisano točnost.

Nato definiramo začetne vrednosti stanj

```
rab=520;
fox=85;
```

Simulacija poteka v zanki, v kateri se neodvisna spremenljivka spreminja od začetne do končne vrednosti z zahtevanim prirastkom

```
%casovna zanka
  for t=0:dt:tfin;
  ...
  end;
```

Glede na osnovni koncept digitalne simulacije, ki ga prikazuje slika 3.22, sta pri vsakem obhodu zanke dve temeljni operaciji: ovrednotenje odvodov spremenljivk

stanja v nekem trenutku t in integracija, ki določi izhode vseh integratorjev v trenutku $t + \Delta t$. Razen teh temeljnih operacij pa moramo dodati še programske vrstice za prikaz rezultatov in za končanje simulacijskega teka. Glede na kompleksnost integracijske metode so omenjene operacije precej prepletene, v primeru najenostavnejše Eulerjeve metode pa velja naslednji vrstni red:

1. **Ovrednotenje odvodov spremenljivk stanja:** Odvoda oz. izhoda integratorjev izračunamo z dvema prireditvenima stavkoma

   ```
   %DERIV
     rabdot=a11*rab-a12*rab*fox;
     foxdot=a21*rab*fox-a22*fox;
   ```

   Stavka sta sicer povsem enaka, kot v jeziku SIMCOS (primer 4.1), v splošnem pa je bistvena razlika v tem, da je pri uporabi simulacijskega jezika vrstni red teh stavkov poljuben, pri simulaciji s splošnonamenskim programskim jezikom pa mora uporabnik paziti na ustrezni vrstni red. Torej če se spremenljivka v nekem stavku n1 na desni strani enačaja izračuna v nekem drugem n2 (se nahaja levo od enačaja), mora biti stavek n2 pred stavkom n1. V našem primeru pa je vrstni red stavkov poljuben, saj so na desni strani obeh stavkov le konstante modela in izhoda integratorjev (stanji sistema).

2. **Prikaz rezultatov:** Rezultate simulacije shranimo v ustrezne indeksirane spremenljivke, ki jih bo možno po simulaciji uporabiti za poljubne prikaze ali obdelave

   ```
   % OUTPUT
     zajci(i)=rab;
     lisice(i)=fox;
     cas(i)=t;
   ```

3. **Integracija:** Uporabimo najenostavnejšo in najbolj ilustrativno Eulerjevo integracijsko metodo, ki jo določa enačba 3.43. Podrobneje pa so metode opisane v poglavju 6.1. Integracijo realiziramo s pomočjo naslednjih stavkov

   ```
   % INTEG
     rab=rab+rabdot*dt;
     fox=fox+foxdot*dt;
   ```

Integracijski postopek izračuna vrednosti `rab` in `fox` že za naslednji računski korak, zato je nujno, da je shranjevanje trenutnih rezultatov opravljeno pred integriranjem. Po izvršitvi integracije je potrebno celotni postopek ponoviti, t.j. narediti je potrebno naslednji prehod po zanki. Če je izpolnjen pogoj za končanje simulacijskega teka, pa se izvrši Matlabov ukaz za izris rezultatov

```
% prikaz rezultatov po simulaciji
  plot(cas,lisice,cas,zajci);
```

Celotni program za simulacijo ekološkega sistema žrtev in roparjev v jeziku Matlab je naslednji:

```
% inicializacija
  a11=5;
  a12=0.05;
  a21=0.0004;
  a22=0.2;
  dt=0.01;
  tfin=10;
  rab=520;
  fox=85;
  i=1;

% casovna zanka
  for t=0:dt:tfin;

     % DERIV
       rabdot=a11*rab-a12*rab*fox;
       foxdot=a21*rab*fox-a22*fox;

     % OUTPUT
       zajci(i)=rab;
       lisice(i)=fox;
       cas(i)=t;

     % INTEG
       rab=rab+rabdot*dt;
       fox=fox+foxdot*dt;
    i=i+1;
```

```
    end;

  % prikaz rezultatov po simulaciji
    plot(cas,lisice);
    plot(cas,zajci);
```

Rezultati simulacije so identični rezultatom, ki jih prikazuje slika 4.5. □

Primer 5.8 Simulacija ekološkega sistema žrtev in roparjev v jeziku BASIC

Spremenljivke lahko inicializiramo v jeziku BASIC s stavkoma `DATA` in `READ`, lahko pa uporabimo kar ustrezne prireditvene stavke:

```
20 A11=5: A12=0.05: A21=0.0004: A22=0.2
```

Več stavkov v eni vrstici ločimo z dvopičjem.

Na enak način definiramo začetni čas simulacije `T`, računski korak `DT` in trajanje simulacijskega teka `TFIN`

```
40 T=0: DT=0.01: TFIN=10
```

Računski korak `DT` ustreza glede na terminologijo CSSL'67 komunikacijskemu intervalu (ki ga običajno označimo z `CINT`) deljeno s številom računskih korakov znotraj komunikacijskega intervala (`NSTEPS`) (`DT = CINT/NSTEPS`). `DT` moramo izbrati glede na karakteristične dinamične lastnosti modela tako, da se izognemo numerični problematiki in zagotovimo predpisano točnost. Izbrana `DT` in `TFIN` odgovarjata izbiri `CINT = 0.01, NSTEPS = 1` in `TFIN = 10` v primeru simulacije istega problema z jezikom SIMCOS (primer 4.1).

Z naslednjo vrstico definiramo začetne vrednosti stanj, t.j. v našem primeru izhodov integratorjev

```
60 RAB=520: FOX=85
```

Nato izpišemo glavo za izpis rezultatov

```
80 PRINT "T            RAB          FOX"
```

1. **Ovrednotenje odvodov spremenljivk stanja:** Odvoda oz. izhoda integratorjev izračunamo z dvema prireditvenima stavkoma:

   ```
   110 RABDOT=A11*RAB-A12*RAB*FOX
   120 FOXDOT=A21*RAB*FOX-A22*FOX
   ```

2. **Prikaz rezultatov:** Najenostavnejši je prikaz rezultatov v obliki izpisa na zaslon:

   ```
   140 PRINT T,RAB,FOX
   ```

   V primerjavi s stavkom `OUTPUT` v primeru 4.1 sta dve razliki:

   - Neodvisna spremenljivka `T` se v jeziku SIMCOS izpisuje avtomatično, v BASIC programu pa jo moramo navesti v stavku `PRINT`.
   - Število 100 v stavku `OUTPUT` pomeni, da želimo izpis na zaslon le v vsakem stotem komunikacijskem koraku. Ker se v programu v jeziku BASIC izpis izvrši ob vsakem klicu stavka `PRINT`, je potrebno zagotoviti, da se stavek `PRINT` izvrši le ob vsakem stotem prehodu po zanki, če želimo, da ima enak učinek, kot stavek `OUTPUT`.

   Če želimo izpis vsak `N`-ti komunikacijski interval, potem uvedemo spremenljivki `N` in `NCOUNT`, ki ju inicializiramo pred simulacijsko zanko

   ```
   45 N=100
   46 NCOUNT=0
   ```

   Namesto stavka št.`140` pa uvedemo naslednje programske vrstice

   ```
   135 IF NCOUNT>0 GOTO 145
   140 PRINT T,RAB,FOX
   141 NCOUNT=N
   145 NCOUNT=NCOUNT-1
   ```

   Stavek `PRINT` se torej izvrši le ob vsakem stotem prehodu po zanki, torej v celotnem simulacijskem teku desetkrat.

3. **Pogoj za končanje simulacijskega teka:** Pri simulaciji v jeziku SIMCOS se tek konča, ko neodvisna spremenljivka simulacije postane večja od predpisane dolžine teka. V BASIC-u realiziramo ta pogoj s stavkom

```
160 IF T>=TFIN GOTO 210
```

znotraj simulacijske zanke. Pri izpolnjenem pogoju torej program zapusti simulacijsko zanko in simulacija se konča.

4. **Integracija:** Uporabimo najenostavnejšo in najbolj ilustrativno Eulerjevo integracijsko metodo, ki jo določa enačba 3.43. Podrobneje pa so metode opisane v poglavju 6.1. Integracijo realiziramo s pomočjo naslednjih stavkov v jeziku BASIC

```
180 T=T+DT: RAB=RAB+RABDOT*DT: FOX=FOX+FOXDOT*DT
```

Integracijski postopek izračuna vrednosti `T, RAB` in `FOX` že za naslednji računski korak, zato je nujno, da je stavek št. `180` na koncu programa. Po izvršitvi integracije je potrebno celotni postopek ponoviti, t.j. narediti je potrebno naslednji prehod po zanki

```
200 GOTO 110
```

Če je izpolnjen pogoj za končanje simulacijskega teka, pa se izvršita stavka

```
210 REM konec simulacijskega programa
220 END
```

Celotni program za simulacijo ekološkega sistema žrtev in roparjev v jeziku BASIC je naslednji:

```
10 REM konstante modela
20 A11=5: A12=.05: A21=.0004: A22=0.2
30 REM racunski korak in
31 REM trajanje simulacijskega teka
40 DT=.01: TFIN=10
45 N=100
46 NCOUNT=0
50 REM zacetni pogoji
60 T=0: RAB=520: FOX=85
70 REM glava za izpis rezultatov
```

```
80 PRINT "T            RAB          FOX"
90 REM zacetek simulacijske zanke
100 REM izracun odvodov
110 RABDOT=A11*RAB - A12*RAB*FOX
120 FOXDOT=A21*RAB*FOX-A22*FOX
130 REM izpis rezultatov
135 IF NCOUNT>0 THEN GOTO 145
140 PRINT T,RAB,FOX
141 NCOUNT=N
145 NCOUNT=NCOUNT-1
150 REM pogoj za koncanje simulacijskega teka
160 IF T>=TFIN THEN GOTO 210
170 REM integracija z Eulerjevo metodo
180 T=T+DT: RAB=RAB+RABDOT*DT: FOX=FOX+FOXDOT*DT
190 REM konec zanke
200 GOTO 110
210 REM konec simulacijskega programa
220 END
```

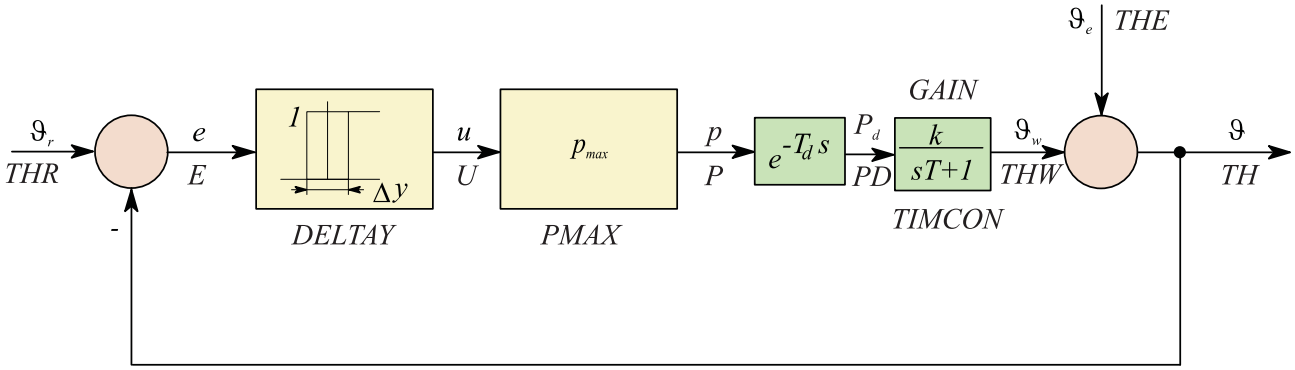
□

Pri uporabi splošnonamenskih jezikov mora torej uporabnik sam določiti vrstni red stavkov, ki opisujejo model (odvode spremenljivk stanja). To sicer v tem primeru ni bilo pomembno, pomembno pa bo v naslednjem primeru, ki prikazuje simulacijo regulacije ogrevanja.

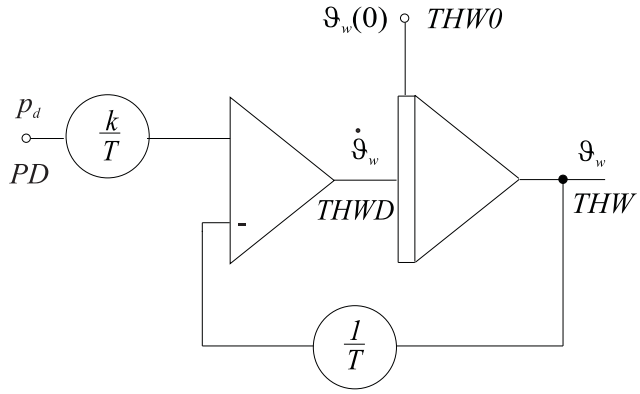
Primer 5.9 Simulacija regulacije ogrevanja v jeziku BASIC

Model regulacije ogrevanja opisuje primer 1.2, ustrezno simulacijo v jeziku SIMCOS pa prikazuje primer 4.2. V modelu uporabljamo le en integrator, katerega izhod je spremenljivka `THW`. To spremenljivko obravnavamo kot znano oz. izhodiščno pri razvrščanju blokov (stavkov). Najprej je potrebno izračunati referenčno temperaturo (`THR`) in absolutno temperaturo (`TH`). Vrstni red izračuna teh dveh spremenljivk ni pomemben, saj je povezava med njima ločena z integratorjem. Nato izračunamo pogrešek, izhod dvopoložajnega regulatorja, izhodni signal grela, njegov zakasnjeni signal in končno še odvod `THWD`. Določitev tega vrstnega reda zelo olajša uporaba bločnega diagrama 5.26 in simulacijske sheme procesa na sliki 5.27.

Simulacijski program v jeziku BASIC je naslednji:



Slika 5.26: Bločni diagram regulacijskega sistema



Slika 5.27: Simulacijska shema temperaturnega procesa

```
10 REM konstante modela
20 DELTAY=1:  PMAX=5:  GAIN=2:  TIMCON=1
30 THE=15: TDELAY =.1:  THW0=1
40 REM opis funkcijskega generatorja
50 DIM REFX(5):  REFX(1)=0:  REFX(2)=6:
51 REFX(3)=9:  REFX(4)=15:  REFX(5)=21
60 DIM REFY(5):  REFY(1)=15:  REFY(2)=20:
61 REFY(3)=18:  REFY(4)=20:  REFY(5)=15
70 REM krmilni parametri simulacije
90 DT=.02:  TFIN=24:  N=10
100 REM inicializacija polja za realizacijo mrtvega casa
110 INDEX=TDELAY/DT+1: DIM WORK(INDEX)
120 FOR I=1 TO INDEX:  WORK(I)=0: NEXT I
130 REM zacetni pogoji
140 T=0:  THW=THW0:  U=0:  NCOUNT=0
```

```
150 REM glava za prikaz rezultatov
160 PRINT "T            THR          P            TH"
170 REM zacetek simulacijske zanke
180 REM funkcijski generator za referencno temperaturo
190 FOR I=1 TO 5: IF T>=REFX(I) THEN THR=REFY(I): NEXT I
210 TH=THW+THE:  E=THR-TH: REM izracun pogreska
230 REM histereza
240 IF E>DELTAY/2 THEN  U=1
250 IF E<-DELTAY/2 THEN  U=0
270 P=PMAX*U: REM preracun v moc grela
280 REM realizacija mrtvega casa
290 FOR I=INDEX TO 2 STEP -1:  WORK(I)=WORK(I-1): NEXT I
300 WORK(1)=P
310 PD=WORK(INDEX)
320 REM izracun odvoda
330 THWD=(-1/TIMCON)*THW+(GAIN/TIMCON)*PD
340 REM izpis rezultatov
350 IF NCOUNT>0 THEN GOTO 380
360 PRINT T,THR,P,TH:  NCOUNT=N
380 NCOUNT=NCOUNT-1
390 REM pogoj za koncanje simulacije
400 IF T>=TFIN THEN GOTO 450
420 T=T+DT:  THW=THW+THWD*DT: REM Eulerjeva integrac. metoda
440 GOTO 170: REM konec simulacijske zanke
450 END
```

Struktura programa je podobna, kot v prejšnjem primeru. Vsi podatki in oznake spremenljivk pa so enake kot pri simulaciji z jezikom SIMCOS v primeru 4.2. Dodatno pojasnilo zahtevata le dva bloka, funkcijski generator, ki ga opisuje tabela 4.2 v primeru 4.2 in pa blok za realizacijo mrtvega časa.

Funkcijski generator v jeziku BASIC definiramo s poljema `REFX` in `REFY`, ki hranita vrednosti lomnih točk neodvisne in odvisne spremenljivke. Polji inicializiramo med operacijami pred simulacijskim tekom:

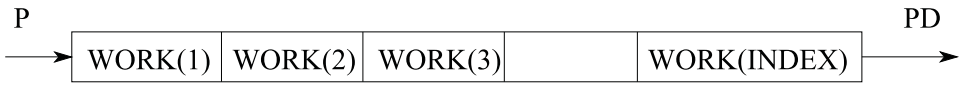
```
50 DIM REFX(5):  REFX(1)=0:  REFX(2)=6
51 REFX(3)=9:    REFX(4)=15: REFX(5)=21
60 DIM REFY(5):  REFY(1)=15: REFY(2)=20
61 REFY(3)=18:   REFY(4)=20: REFY(5)=15
```

Ti programski stavki torej ustrezajo stavku `TABLE` v jeziku SIMCOS. Referenčno temperaturo pa generiramo z naslednjimi programskimi vrsticami:

```
190 FOR I=1 TO 5: IF T>=REFX(I) THEN THR=REFY(I): NEXT I
```

Te programske vrstice sicer niso ekvivalent funkcijskega generatorja v jeziku SIMCOS, saj ne vsebujejo linearne interpolacije. Toda za odsekovno konstantni referenčni signal dobimo ustrezno delovanje.

Mrtvi čas (časovno zakasnitev) realiziramo s poljem `WORK`, ki mora imeti dimenzijo `TDELAY/DT+1`, kar je v programu označeno s spremenljivko `INDEX`. Ustrezen sistem za realizacijo mrtvega časa prikazuje slika 5.28.



Slika 5.28: Sistem za realizacijo mrtvega časa

Polje `WORK` je deklarirano in inicializirano s stavki

```
110  INDEX=TDELAY/DT+1: DIM WORK(INDEX)
120 FOR I=1 TO INDEX:  WORK(I)=0: NEXT I
```

Pri vsakem obhodu po zanki moramo elemente polja `WORK` premakniti za eno mesto proti višjemu indeksu. Vhod v blok za zakasnitev je povezan s prvim elementom polja `WORK`, izhod pa je element z indeksom `INDEX`:

```
290 FOR I=INDEX TO 2 STEP -1:  WORK(I)=WORK(I-1): NEXT I
300  WORK(1)=P
310  PD=WORK(INDEX)
```

Z opisano realizacijo lahko proučujemo tudi model brez mrtvega časa `(TDELAY=0)`.
□

Primer 5.10 Simulacija prenosne funkcije

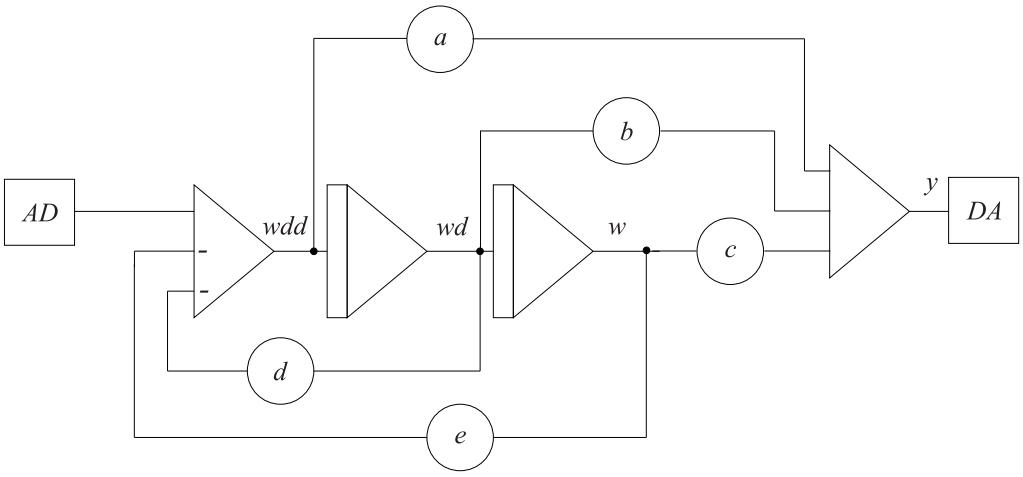
Chebyshev filter 2. reda z mejno frekvenco $f_m = 50Hz$ in dušenjem vsaj 30db v zapornem delu podaja prenosna funkcija

$$G(s) = \frac{as^2 + bs + c}{s^2 + ds + e} \tag{5.4}$$

s konstantami

$$\begin{aligned} a &= 0.03162 \\ b &= 0 \\ c &= 6242 \\ d &= 110 \\ e &= 6242 \end{aligned}$$

Filter želimo realizirati na digitalnem računalniku, zato signal, ki ga je potrebno filtrirati, vzorčimo z A/D pretvornikom, filtrirani signal pa vračamo iz računalnika v obliki analognega signala na D/A pretvorniku. Ustrezno simulacijsko shemo prikazuje slika 5.29.



Slika 5.29: Shema za simulacijo (realizacijo) Chebyshev -ega filtra

Program za realizacijo filtra v jeziku BASIC je naslednji:

```
10  a=0.03162:c=6242:d=110:e=6242:w=0:wd=0:t=0:dt=0.002
20  rem sinhronizacija z realnim casom (dt)
30  rem   u ... iz A/D pretvornika
40  wdd=u-d*wd-e*w
50  y=a*wdd+c*w
```

```
 60  rem   y ... na D/A pretvornik
 70  t=t+dt
 80  w=w+wd*dt
 90  wd=wd+wdd*dt
100  goto 20
110  end
```

V vrstici 10 so definirane vse konstante prenosne funkcije, začetni vrednosti integratorjev, začetni čas in velikost računskega koraka. Izbrali smo 2ms, saj ima filter mejno frekvenco 50Hz, torej je perioda najvišjih frekvenc, ki jih prepušča, približno 20ms. Torej smo dosegli približno 10 računskih korakov na eno periodo. Vse vrstice med stavkoma 20 in 100 so v neskončni zanki, saj mora filter delovati poljubno dolgo v realnem času. Zato je potrebno vsak nov prehod po zanki natančno sinhronizirati z realnim časom, kar naj bi zagotovil programski modul v vrstici 20 (prehodi na 2ms). V vrstici 30 se izvrši A/D pretvorba vhodnega signala, vrstice od 40 do 90 pa simulirajo (realizirajo) filter z uporabo Eulerjeve integracijske metode. Pri enačbah je zelo pomemben vrstni red. Najprej se mora izračunati `wdd` in šele nato `y`. Odvoda `wd` ni potrebno izračunavati, saj je to hkrati stanje, ki ga izračunava integracijska metoda. Zaradi zaporedno povezanih integratorjev je tudi važen vrstni red obeh enačb integratorjev. Najprej je potrebno iz trenutne vrednosti `wd` izračunati `w` za naslednji računski korak in šele nato `wd` za naslednji računski korak iz trenutne vrednosti spremenljivke `wdd`. Programski moduli za A/D in D/A pretvorbo ter za sinhronizacijo z realnim časom so odvisni od specifične materialne opreme, zato so v zgornjem programu le ustrezni komentarski stavki. Procesor računalnika mora biti zadosti hiter, da v času 2ms izvrši vse operacije v zanki. Vprašljiva pa je numerična stabilnost in natančnost Eulerjeve metode. □

Razvrstitev stavkov (blokov), ki opisujejo model pa ni vedno možna. To se dogodi, kadar imamo v simulacijski shemi zanke brez t.i. spominskih blokov (blokov z zakasnitvenim atributom, npr. integratorjev). Vrstic v obliki

```
X2=X1+X2+...
```

ali

```
X1=X2+ ...
X2=X1+ ...
```

ni možno razvrstiti. Takim strukturam pravimo "algebrajske zanke", modele, ki jih vsebujejo, pa je možno simulirati le s posebnimi metodami (poglavje 6.3). Vendar so algebrajske zanke mnogokrat posledica slabega modeliranja.

Pri eksperimentiranju z nekim simulacijskim modelom lahko pričakujemo, da bo potrebno večkrat izvesti simulacijski tek z različnimi parametri (npr. različni začetni pogoji, konstante modela,...). V jeziku BASIC, ki je običajno interpreterskega tipa, to spreminjanje ni problematično, saj le spremenimo ustrezne programske vrstice in takoj spet izvedemo simulacijo. V primeru prevajalniških jezikov (FORTRAN, PASCAL) pa je potrebno po vsaki spremembi programa le tega ponovno prevajati in povezati s knjižnicami, kar je lahko (predvsem na manjših računalnikih) kar zamudno. Zato v tem primeru običajno vse simulacijske parametre zberemo v datoteki, ki jo mora simulacijski program prebrati na začetku med operacijami pred simulacijskim tekom. Med simulacijskimi teki lahko to datoteko enostavno editiramo.

BASIC spada med slabo strukturirane splošnonamenske programske jezike, zato smo celoten program realizirali v enem programskem modulu. Modernejši jeziki (PASCAL, C, FORTRAN) so dobro strukturirani višji splošnonamenski jeziki in omogočajo bolj pregledno in modularno realizacijo koncepta digitalnih simulacijskih jezikov, ki ga prikazujeta sliki 3.22 in 3.24. Zato tak program sestoji iz glavnega programa, ki poskrbi za vse potrebne inicializacije (npr. čitanje konstant modela) in za klic osrednje operacije, t.j. integracijskega podprograma (`INTEG`). V tem podprogramu je realizirana simulacijska zanka, ki poteka od začetne vrednosti neodvisne spremenljivke simulacije do izpolnitve pogoja za končanje simulacije. Vmes pa podprogram `INTEG` kliče podprogram za izračun odvodov spremenljivk stanja (`DERIV` - t.j. podprogram, ki vsebuje enačbe modela) in v zahtevanih trenutkih podprogram za komunikacijo z uporabnikom (`OUTPUT` - podprogram za prikaz rezultatov). Čeprav bomo v zvezi s podprogramom `DERIV` običajno vedno govorili kot o podprogramu za izračun odvodov (v povezavi z zapisom v prostoru stanj so to enačbe stanj $\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u}$), pa se v tem podprogramu ovrednotijo tudi razne druge spremenljivke kot funkcije spremenljivk stanja. Gre za enačbe, ki za sam integracijski postopek niso pomembne (v povezavi z zapisom sistema v prostoru stanj so to izhodne enačbe ($\boldsymbol{y} = \boldsymbol{C}\boldsymbol{x} + \boldsymbol{D}\boldsymbol{u}$). Slednje bi bilo možno vključiti tudi v podprogram `OUTPUT`, kar bi povečalo hitrost simulacije zlasti pri kompleksnejših integracijskih postopkih.

Omenimo naj, da je v simulacijskih sistemih dostikrat realiziran tudi nekoliko spremenjeni koncept, v katerem se simulacijska zanka realizira v glavnem programu. V tej zanki glavni program kliče podprogram za izpis rezultatov in integracijski podprogram, ki pa integrira le na enem komunikacijskem intervalu.

Simulirajmo v naslednjem primeru ekološki model žrtev in roparjev v jeziku PASCAL.

Primer 5.11 Simulacija ekološkega sistema žrtev in roparjev v jeziku PASCAL

Definirajmo datoteko `database` v naslednji obliki:

```
5      0.05  0.0004  0.2
0.01   10    100
0      520   85
```

V prvi vrstici so konstante modela (`a11, a12, a21` in `a22`), v drugi vrstici so krmilni parametri simulacije, to so računski korak (`dt`), končni čas simulacije (`tfin`) in število računskih korakov med dvema izpisoma (`n`). V tretji vrstici pa so začetni čas simulacije (`t`) ter začetni vrednosti obeh stanj (`rab` in `fox`).

Program v PASCAL-u za simulacijo ekološkega sistema žrtev in roparjev je naslednji:

```
Program Prey_predator;
var t,dt,tfin,a11,a12,a21,a22:real;
   rab,fox,rabdot,foxdot:real;
   n,ncount:integer;
   data_base:text;
procedure deriv;
  (* opis strukture modela oz. enacbe za izracun odvodov *)
  begin
     rabdot:=a11*rab-a12*rab*fox;foxdot:=a21*rab*fox-a22*fox;
  end;  (* deriv *)
procedure output;
(* izpis rezultatov ob vsakem n-tem prehodu po zanki *)
  begin
    if ncount<=0 then begin
                       writeln(t,rab,fox);ncount:=n;
                      end;
   ncount:=ncount-1;
  end;  (*output *)
procedure integ;
```

```
 (* generacija simulacijske zanke in Eulerjeva integracijska metoda *)
  begin
    repeat begin
             deriv; output;
             t:=t+dt;rab:=rab+rabdot*dt;fox:=fox+foxdot*dt;
           end
    until t>=tfin;
  end;  (* integ *)
begin  (* glavni program *)
 (* citanje konstant modela in krmilnih parametrov simulacije
 iz datoteke *)
 assign(data_base,'database');reset(data_base);
 readln(data_base,a11,a12,a21,a22);
 readln(data_base,dt,tfin,n);readln(data_base,t,rab,fox);
 ncount:=0; writeln('        T        RAB       FOX');
 integ;
end.
```

Ker so oznake enake kot v programu v jeziku BASIC, dodatni komentarji niso potrebni. □

Priporočljivo je, da za integracijo uporabljamo profesionalne in komercialno dostopne podprograme. Le-ti uporabljajo numerično bolj stabilne in natančnejše integracijske metode (npr. Runge-Kutta, Adams-Bashforth, Gear-stiff,...), ki običajno vključujejo tudi postopke, ki s prilagajanjem velikosti računskega koraka ali s prilagajanjem same metode sproti nadzirajo absolutni ali relativni pogrešek med simulacijo. Pri uporabi takih podprogramov, ki jih običajno nimamo v izvorni obliki, pa se srečamo z naslednjimi problemi:

1. Podprogram za integracijo mora dobiti podatek o številu stanj (integratorjev) v našem problemu.
2. Podprogram za integracijo mora imeti povezavo z imeni spremenljivk, ki nastopajo v modelu.
3. Podprogram za integracijo mora dobiti podatek o izpolnjenem pogoju za končanje simulacijskega teka.
4. Podprogram za integracijo mora vedeti za imeni podprogramov za ovrednotenje odvodov spremenljivk stanj in za izpis rezultatov.

Prva dva problema rešimo tako, da vse vhode integratorjev združimo v polje odvodov, vse izhode integratorjev pa v polje stanj. Ti polji in število integratorjev (število elementov polj) so parametri za prenos v integracijski podprogram. S tem, da integracijski podprogram uporablja polji poljubnih dimenzij, postane univerzalno uporaben. Zato pa je treba tudi vse enačbe modela zapisati z elementi polj.

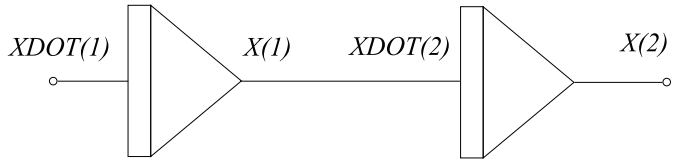
Tretji problem rešimo tako, da pogoj za končanje simulacije ni v integracijskem podprogramu ampak v enem od podprogramov, ki ju napiše uporabnik, t.j. ali v podprogramu za ovrednotenje odvodov ali pa v podprogramu za izpis rezultatov. Eden od teh programov torej določi konec simulacijskega teka in to s posebno zastavico (flag) sporoči integracijskemu podprogramu, da le ta dejansko konča s simulacijsko zanko. Ker se pri zahtevnejših integracijskih metodah podprogram za izpis rezultatov kliče precej redkeje kot podprogram za ovrednotenje odvodov, je zaradi hitrosti simulacije smiselno pogoj za končanje simulacije vgraditi v podprogram za izpis rezultatov.

Četrti problem je možno elegantno rešiti v jeziku FORTRAN. Jezik omogoča, da navedemo imena podprogramov kot parametre v klicnem stavku. Imena pa moramo navesti tudi v stavku EXTERNAL, ki se nahaja med deklarativnimi stavki glavnega programa. Pri programiranju v PASCAL-u pa moramo v izvorno integracijsko proceduro vpisati ustrezna imena, če pa nimamo izvorne kode, pa moramo za imeni procedur za ovrednotenje odvodov in izpis rezultatov uporabiti predpisani oz. v integracijski proceduri uporabljeni imeni.

Ker moramo v podprogramu za izračun odvodov po omenjenem konceptu uporabljati elemente polj namesto imen problemskih spremenljivk, tak program izgubi preglednost in ne predstavlja osnove za dobro dokumentacijo modela (npr. v PASCAL-u uporabljamo spremenljivki `x[1]` in `xdot[1]` namesto `RAB` in `RABDOT` pri simulaciji modela žrtev in roparjev). V tem primeru programski jezik FORTRAN nudi elegantno rešitev s pomočjo stavka `EQUIVALENCE`. Ta stavek (npr. `EQUIVALENCE (X(1), RAB)`) zagotovi, da imajo navedene spremenljivke isto lokacijo, torej je vseeno, katero uporabljamo (ali `X(1))` ali `RAB)`. Ustrezne `EQUIVALENCE` stavke definiramo v vseh programskih modulih, ki jih napiše uporabnik (v glavnem programu, v podprogramu za izračun odvodov, v podprogramu za izpis rezultatov). V teh modulih nato operiramo s problemskimi imeni, medtem ko integracijski podprogram, do katerega nimamo dostopa (predpostavljamo, da je to profesionalni podprogram, za katerega običajno niti nimamo izvorne kode ali pa ga ne želimo spreminjati) operira z ustreznimi elementi polj oz. z vektorjema stanj in odvodov. Pri prenosu spremenljivk v podprograme pa je potrebno upoštevati, da spremenljivke v stavkih `EQUIVALENCE` ne smejo biti

med klicnimi parametri podprograma.

Pri takem modularnem programiranju je treba paziti, da v primeru, če je nek signal v shemi hkrati stanje in odvod, le tega označimo z dvema spremenljivkama, kar prikazuje slika 5.30.



Slika 5.30: Označevanje spremenljivk pri zaporedni povezavi integratorjev

Primer 5.12 Simulacija ekološkega sistema žrtev in roparjev v jeziku FORTRAN

Postopek simulacije bomo začeli z integracijskim podprogramom in čeprav bomo nakazali koncept pri uporabi profesionalnih programov, bomo zaradi kontinuitete in večje ilustrativnosti ostali pri Eulerjevi integracijski metodi. Programski jezik FORTRAN ne pozna globalnih spremenljivk, zato poteka prenos običajno preko klicnih parametrov podprogramov (subroutine ali function) včasih pa preko spremenljivk, ki jih navedemo v stavku `COMMON`. Minimalni nabor podatkov, ki ga običajno zahteva nek profesionalni integracijski podprogram, je: začetni čas simulacije, velikost računskega koraka, spremenljivka kot zastavica (flag) za končanje simulacije (vse tri spremenljivke so običajno združene v vektorju (polju)), vektor stanj (izhodov integratorjev), ki jih integracijski podprogram izračunava med simulacijo, vektor odvodov spremenljivk stanja, dolžina obeh vektorjev oz. število stanj oz. integratorjev ter imeni podprogramov za izračun odvodov in izpis rezultatov. Enostavni integracijski podprogram je naslednji:

```
      SUBROUTINE INTEG(PRMT,X,XDOT,NDIM,MODEL,RESULTS)
      DIMENSION PRMT(1),X(1),XDOT(1)
      EXTERNAL MODEL, RESULTS
      T=PRMT(1)
10    CALL MODEL(T)
      CALL RESULTS(T,PRMT)
      IF(PRMT(3).NE.0) RETURN
      T=T+PRMT(2)
      DO 20 I=1,NDIM
```

```
20    X(I)=X(I)+XDOT(I)*PRMT(2)
      GO TO 10
      END
```

Pomen klicnih parametrov podprograma je naslednji:

`PRMT` - polje z dimenzijo tri ali več z naslednjimi parametri:
`PRMT(1)` - začetna vrednost neodvisne spremenljivke
`PRMT(2)` - velikost računskega koraka
`PRMT(3)` - zastavica za končanje simulacijskega teka
Vsi trije parametri morajo biti definirani ob klicu integracijskega podprograma. `PRMT(3)` je na začetku enak nič, ko pa je izpolnjen pogoj za končanje simulacijskega teka, ga podprogram za prikaz rezultatov postavi na vrednost različno od nič

`X` - polje stanj (izhodov integratorjev). Ob klicu podprograma `INTEG` mora vsebovati začetne vrednosti stanj

`XDOT` - polje odvodov spremenljivk stanja (vhodov v integratorje)

`NDIM` - število stanj (integratorjev)

`MODEL` - podprogram za izračun odvodov spremenljivk stanja

`RESULTS` - podprogram za prikaz rezultatov

Podprogram `INTEG` je v praksi neki profesionalni podprogram in ga zato ne želimo spreminjati. Običajno tudi nimamo izvorne kode in poznamo le pomen parametrov, ki jih je potrebno prenašati. Uporabnik pa mora seveda napisati glavni program, podprogram za izračun odvodov (predpostavimo ime `DERIV`) in podprogram za izpis rezultatov (predpostavimo ime `OUTPUT`).

Povsem ekvivalenten glavni program programu v jeziku PASCAL v primeru 5.11 je naslednji:

```
      PROGRAM PREY_PREDATOR
      DIMENSION X(2),XDOT(2),PRMT(3)
      COMMON X,XDOT,A11,A12,A21,A22,TFIN,N,NCOUNT
      EXTERNAL DERIV,OUTPUT
      EQUIVALENCE (X(1),RAB),(X(2),FOX)
```

```
EQUIVALENCE (PRMT(2),DT)
OPEN(1,FILE='DATABASE',FORM='FORMATTED',STATUS='OLD')
READ(1,*)A11,A12,A21,A22
READ(1,*)DT,TFIN,N
READ(1,*)T,RAB,FOX
CLOSE(1)
NCOUNT=0
PRMT(1)=T
PRMT(3)=0.
CALL INTEG(PRMT,X,XDOT,2,DERIV,OUTPUT)
STOP
END
```

Ustrezni podprogram za izračun odvodov spremenljivk stanja `DERIV` pa je

```
SUBROUTINE DERIV(T)
DIMENSION X(2),XDOT(2)
COMMON X,XDOT,A11,A12,A21,A22,TFIN,N,NCOUNT
EQUIVALENCE (X(1),RAB),(X(2),FOX)
EQUIVALENCE (XDOT(1),RABDOT),(XDOT(2),FOXDOT)
RABDOT=A11*RAB-A12*RAB*FOX
FOXDOT=A21*RAB*FOX-A22*FOX
RETURN
END
```

Podprogram za izpis rezultatov `OUTPUT` pa je naslednji:

```
SUBROUTINE OUTPUT(T,PRMT)
DIMENSION PRMT(1)
DIMENSION X(2),XDOT(2)
COMMON X,XDOT,A11,A12,A21,A22,TFIN,N,NCOUNT
EQUIVALENCE (X(1),RAB),(X(2),FOX)
IF (NCOUNT.LE.0) THEN
                   WRITE(*,*)T,RAB,FOX
                   NCOUNT=N
                   ENDIF
NCOUNT=NCOUNT-1
IF(T.GE.TFIN) PRMT(3)=1.
RETURN
```

```
      END
```

Stavki `EQUIVALENCE` v podprogramih `DERIV` in `OUTPUT` omogočajo, da v teh dveh uporabniških modulih uporabljamo problemska imena (npr. `RAB, RABDOT`) hkrati pa se v integracijskem podprogramu vrši integracija z uporabo polj `X` in `XDOT` (`RAB` in `RABDOT` imata isto lokacijo kot `X(1)` in `XDOT(1)`). Konstante modela (`A11`, `A12`, `A21`, `A22`) ter nekateri krmilni parametri (čas simulacije `(TFIN)`, podatki za izpis rezultatov `(N, NCOUNT)` pa se prenašajo v podprograma `DERIV` in `OUTPUT` preko t.i. `COMMON` bloka. Vektor stanj `X` in vektor odvodov `XDOT` pa se prenašata med podprogramom `INTEG` in podprogramoma `DERIV` oz. `OUTPUT` indirektno preko glavnega programa. Med podprogramom `INTEG` in glavnim programom namreč obstoja povezava preko parametrov klicnega stavka, med glavnim programom in podprograma `DERIV` in `OUTPUT` pa preko `COMMON` bloka. Neodvisno spremenljivko `T` in vektor krmilnih parametrov `PRMT` pa prenašamo preko parametrov v klicnih stavkih. Struktura bi bila nekoliko enostavnejša, če bi se vektorja `X` in `XDOT` med vsemi moduli prenašala preko parametrov, vendar FORTRAN ne dovoljuje, da je ista spremenljivka, ki se prenaša v podprogram (`DERIV` oz. `OUTPUT`) preko parametrov hkrati tudi v `EQUIVALENCE` stavku znotraj tega podprograma. Lahko pa bi med vsemi moduli stanja in odvode prenašali preko `COMMON` bloka.

Opisana struktura je torej bolj kompliciran, kot pri programu v jeziku PASCAL. Vendar smo z njo dosegli, da uporabnik lahko uporablja nek profesionalni integracijski podprogram. Pri pisanju podprogramov `DERIV` in `OUTPUT` uporablja problemske spremenljivke, tema dvema podprogramoma pa da lahko tudi poljubno ime ne glede na ime, ki ga uporablja podprogram `INTEG`.

Pri prikazu rezultatov smo vseskozi omenjali izpis na zaslon, čeprav tudi izpis v datoteko ali grafična predstavitev med simulacijo ne spremeni koncepta. V podprogramu `OUTPUT` je potrebno le uporabiti ustrezne izhodne stavke ali klicati ustrezne podprograme. □

V naslednjem primeru bomo nakazali uporabnost tranputerskih paralelnih procesorskih sistemov v simulaciji. Osnovne lastnosti smo opisali v podpoglavju 4.1.2.

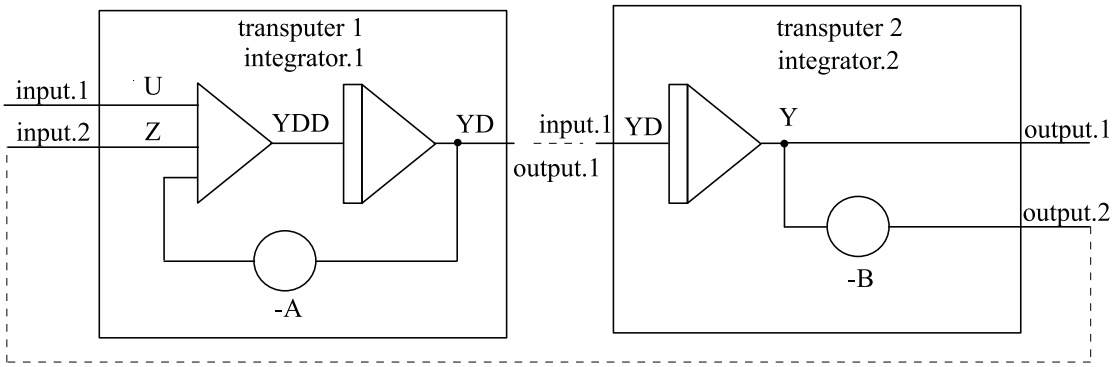
Primer 5.13 Simulacija sistema drugega reda v OCCAM-u

Simulirajmo sistem, ki ga opisuje diferencialna enačba

$$\ddot{y} + a\dot{y} + by = u \tag{5.5}$$

z ničnimi začetnimi pogoji in s konstantama $a = 1.4, b = 1$ na paralelnem transputerskem sistemu. Za integracijo uporabimo Eulerjevo metodo z računskim korakom $\Delta t = 0.01$ s.

Predpostavljamo, da ima računalnik dva transputerja. Najprej moramo določiti, kaj bo računal prvi in kaj drugi transputer. Smiselno je, da vsak transputer računa eno spremenljivko stanj. Simulacijsko shemo in razdelitev operacij prikazuje slika 5.31.



Slika 5.31: Simulacijska shema sistema 2. reda

Računalniške spremenljivke `U, Z, YDD, YD, Y, A, B, DT` smo izbrali za označitev problemskih spremenljivk $u, z, \ddot{y}, \dot{y}, y, a, b, \Delta t$. Črtkane črte predstavljajo fizične povezave med transputerjema. Prvi transputer potrebuje tri serijske priključke: dva vhoda `input.1, input.2` in en izhod `output.1`. Drugi transputer potrebuje enako število priključkov: en vhod `input.1` in dva izhoda `output.1` in `output.2`. Spremenljivki stanj `Y,YD` sta znani na začetku vsakega računskega koraka. Zato transputerja najprej postavita od stanj odvisne izhodne signale, nato sprejmeta vhodne signale, izračunata odvoda spremenljivk stanj (`YD, YDD`) in nato izračunata z Eulerjevo integracijsko metodo (enačba (3.43)) stanji (`Y, YD`) za naslednji računski korak. Postopek izmenjavanja podatkov in izračunavanja se nato ponavlja v vsakem računskem koraku. Vsak transputer razen podatkov, ki jih sprejme preko serijske povezave, potrebuje tudi lokalne spremenljivke, ki jih hrani v lastnem pomnilniku.

Slika 5.32 prikazuje ustrezni program v jeziku OCCAM. Program sestoji iz dveh t.i. procesov (`PROC integrator.1` in `PROC integrator.2`), ki se izvajata paralelno. Znotraj vsakega procesa nastopajo sekvenčni bloki (`SEQ` - izvajanje operacij druga za drugo) in paralelni bloki (`PAR` - poljubni vrstni red operacij znotraj takih blokov)). Stavek `PROC` določa fizične povezave. Nato so definirane konstante in deklaracije spremenljivk. V prvem sekvenčnem bloku je podan začetni pogoj in

```
PROC integrator.1 (CHAN input.1, input.2, output.1)
  VAL A IS      1.4 (REAL32):
  VAL DT IS    0.01 (REAL32):

  REAL32 U,YDD,YD,Z,DT:

  SEQ
   YD:=0.0(REAL32)
   WHILE TRUE
     SEQ
       PAR
         output.1!YD
         input.1?U
         input.2?Z
       YDD:=(U+Z)-(A*YD)
       YD:=YD+(DT*YDD)

PROC integrator.2 (CHAN input.1, output.1, output.2)
  VAL B IS   1.0(REAL32):
  VAL DT IS  0.01(REAL32):

  REAL32 YD, Y, Z, DT

  SEQ
   Y:=0.0(REAL32)
   WHILE TRUE
     SEQ
       Z:-B*Y
       output.2!Z
       PAR
          output.1!Y
          input.1?YD
        Y:=Y+(DT*YD)
```

Slika 5.32: Program v jeziku OCCAM

izvedena simulacijska zanka (s pomočjo `WHILE` stavka). Z naslednjim sekvenčnim blokom, ki je vgnezden v prvega (torej v simulacijsko zanko), pa izvajamo operacije v vsakem računskem koraku. Izračunavanja, ki se lahko izvajajo paralelno (v poljubnem vrstnem redu), smo navedli v `PAR` bloku. Vrstice s klicajem pošiljajo

podatke preko serijske povezave, vrstice z vprašajem pa sprejemajo podatke, pri čemer čakajo na veljavnost podatkov na povezavah (s tem je zagotovljeno tudi usklajeno delovanje obeh transputerjev). Ko se v paralelnih blokih pridobijo vsi potrebni podatki, se v obeh procesih nadaljuje izvajanje sekvenčnih blokov, v katerih se po Eulerjevi integracijski metodi izračunajo stanja za naslednji računski korak. Konce blokov ne določajo stavki `end`, ampak ustrezni zamiki pri pisanju vrstic.

V tem primeru nismo nič govorili o vhodni spremenljivki `u`. Dobili bi jo lahko tako, da bi vzorčili nek realni signal z A/D pretvornikom, ki bi ga povezali na transputer. □

Poglavje 6

Numerični postopki v simulaciji

Numerične integracijske metode predstavljajo srce vsakega digitalnega simulacijskega sistema. Za reševanje numerično nezahtevnih problemov ne potrebujejo uporabniki praktično nobenega znanja o integracijskih postopkih. Pri numerično zahtevnih problemih pa je zelo pomembno poznavanje osnovnih lastnosti integracijskih metod in še predvsem, kaj so prednosti in slabosti posameznih postopkov. Pregled integracijskih postopkov je namenjen tistim, ki uporabljajo simulacijo za praktično reševanje problemov in ne strokovnjakom s področja numeričnih metod. Opisujemo pa tudi problem integracije preko nezveznosti v signalih.

Poseben numerični problem predstavlja algebrajska zanka. V simulaciji jo redko srečamo, ponavadi pa nastane zaradi neustreznega modeliranja. Pogosto se pojavi zlasti neizkušenim modelerjem in programerjem. Zato podajamo tudi osnove numeričnega reševanja algebrajske zanke.

Z osnovnim znanjem o integracijskih metodah ter o reševanju algebrajske zanke bo dobil uporabnik bolj zanesljive rezultate, prihranil pa bo tudi veliko računalniškega časa.

6.1 Numerične integracijske metode

Simulacijska orodja, ki so nam na voljo, nudijo bogat izbor različnih integracijskih metod. Običajno pa uporabnik ne ve, katera metoda je za njegov problem na-

jprimernejša. Včasih pa sploh ne ve, da obstajajo različne metode, saj uporablja kar privzeto (default) metodo (običajno metoda Runge - Kutta 4. reda). Le-ta dobro deluje za ne preveč zahtevne probleme. Razlog je seveda jasen: uporabnik ne pozna osnovnih lastnosti, ki vplivajo na izbiro metode. Zato ignorira možnost izbire in čez čas celo pozabi na to možnost. Idealna bi bila seveda metoda, ki bi reševala vse numerične probleme z enako učinkovitostjo. Toda pričakovanja, da bi nekdo razvil tako metodo, so nestvarna. Veliko obetajo le t.i. ekspertni sistemi, ki bi uporabniku pomagali izbrati optimalno metodo za simulacijo njegovega problema.

Uporabnik, ki pa si želi pridobiti bolj poglobljeno znanje, naj uporabi kakšno od naslednjih referenc: (Hairer, Wanner, 1990), (Gustaffson, 1990),
(Hairer, Lubich, 1988), (Gustaffson, 1988), (Press in ostali, 1986), (Korn, Wait, 1978), (Hall, Watt, 1976), (Lapidus, Seinfeld, 1971), (Gear, 1971).

6.1.1 Splošna oblika numeričnega integracijskega algoritma

Začenši od poznega osemnajstega stoletja, ko je Euler razvil svoj algoritem, je numerična integracija za reševanje navadnih diferencialnih enačb predstavljala pomembno področje numerične matematike. Mi se bomo osredotočili predvsem na metode, ki jih je možno uporabiti za širši spekter problemov in pa na metode, ki imajo sicer manjšo praktično vrednost, a so zaradi enostavnosti razumljivejše in tako pedagoško učinkovitejše.

Ker je model vsakega realnega sistema skoraj vedno zapisan z diferencialnimi enačbami višjih redov, je potrebno te enačbe transformirati (ročno ali avtomatsko) v sistem enačb prvega reda z ustreznim definiranjem novih spremenljivk. Tako lahko reševanje predstavimo kot začetnovrednostni problem z vektorsko diferencialno enačbo

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{x}) \tag{6.1}$$

in začetnim pogojem $\boldsymbol{x}(t_0) = \boldsymbol{x}_0$, kjer je $\boldsymbol{x}$ vektor stanj in $\boldsymbol{f}$ odvodna funkcija. Pri simulaciji je potrebno enačbo (6.1) integrirati od začetnega časa t_0 do končnega časa t_{max}. Ob uporabi numerične integracijske metode moramo celotni čas simulacije razdeliti na ustrezno število računskih korakov. Predpostavimo, da

je računski korak h konstanten, tako je potrebno izračunati rešitev v trenutkih $t_k = t_0 + k \cdot h$, $k = 0, 1, 2, \ldots, k_{max}$. Točna rešitev v trenutku t_{k+1} je

$$\boldsymbol{x}(t_{k+1}) = \boldsymbol{x}(t_0) + \int_{t_0}^{t_{k+1}} \boldsymbol{f}(t, \boldsymbol{x})dt = \boldsymbol{x}(t_k) + \int_{t_k}^{t_{k+1}} \boldsymbol{f}(t, \boldsymbol{x})dt \tag{6.2}$$

Z diskretizacijo enačbe (6.2) dobimo izraz

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{I}_k \tag{6.3}$$

kjer je $\boldsymbol{I}_k$ približna vrednost integrala

$$\boldsymbol{I}_k \approx \int_{t_k}^{t_{k+1}} \boldsymbol{f}(t, \boldsymbol{x})dt \tag{6.4}$$

Zaradi te aproksimacije je $\boldsymbol{x}_{k+1}$ samo približek prave vrednosti $\boldsymbol{x}(t_{k+1})$. V vsakem računskem koraku torej nastane lokalna napaka, ki se lahko med simulacijo tudi akumulira. Akumulirano napako imenujemo globalna napaka.

Znani so številni numerični algoritmi, ki rešijo enačbo (6.4). Delimo jih v

- enokoračne metode (implicitne in eksplicitne),
- večkoračne metode (implicitne in eksplicitne),
- ekstrapolacijske metode in
- metode za simulacijo togih sistemov.

6.1.2 Vrste numeričnih integracijskih napak

Če postopek integracije izvedemo z numeričnim algoritmom na digitalnem računalniku, se pojavita dve vrsti napak:

- *napaka zaradi končnega reda numerične metode* in

- *napaka zaradi končne dolžine besede* (oz. omejene natančnosti), s katero deluje aritmetika določene programske opreme na računalniku.

Napaka numerične metode

Napaka numerične metode (numerical approximation or truncation error) je torej pogojena z omejeno natančnostjo integracijskega postopka in ni odvisna od natančnosti računalnika oz. njegove aritmetike. Če bi bila aritmetika računalnika povsem točna, bi določena integracijska metoda povzročila v enem računskem koraku *lokalno napako*

$$\begin{aligned} \boldsymbol{e}_{k+1} &= \boldsymbol{x}_{k+1} - \boldsymbol{x}(t_{k+1}) = \\ &= \boldsymbol{x}_{k+1} - \left(\boldsymbol{x}_k + \int_{t_k}^{t_{k+1}} \boldsymbol{f}(t, \boldsymbol{x}) dt \right) = \\ &= \boldsymbol{I}_k - \int_{t_k}^{t_{k+1}} \boldsymbol{f}(t, \boldsymbol{x}) dt \end{aligned} \tag{6.5}$$

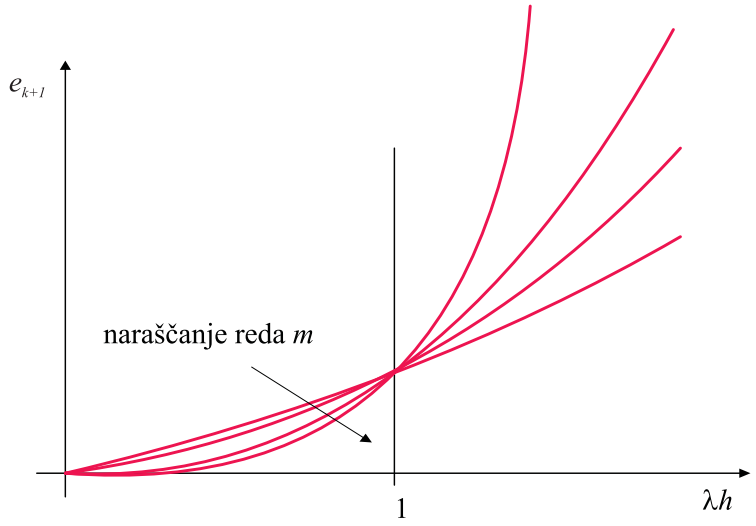
rezultirajoče vrednosti $\boldsymbol{x}_{k+1}$, pri čemer je t_k začetni trenutek opazovanja. Predpostavljamo, da imamo v tem trenutku točen rezultat. Često lahko $\boldsymbol{e}_{k+1}$ ocenjujemo med potekom simulacije. Vendar pa je za vrednotenje rezultata pomembnejša *globalna napaka*

$$\boldsymbol{e}_{T,k+1} = \boldsymbol{x}_{k+1} - \left(\boldsymbol{x}_0 + \int_{t_0}^{t_{k+1}} \boldsymbol{f}(t, \boldsymbol{x}) dt \right) \tag{6.6}$$

Pri tem je $t = t_0$ začetni čas simulacije. Običajno je globalna napaka večja od lokalne napake. V določenih primerih lahko postane neskončna. Toda globalne napake ni možno oceniti med simulacijo. Omejena lokalna napaka tudi ne zagotavlja omejene globalne napake. Vendar predstavlja lokalna napaka edino možnost za nadzor točnosti med simulacijo. Za večino integracijskih metod velja, da je lokalna napaka zaradi numerične metode sorazmerna $m + 1$ potenci računskega koraka

$$\boldsymbol{e}_{k+1} \propto h^{m+1} \tag{6.7}$$

kjer je h računski korak in m red integracijske metode. Torej je potrebno za natančnejše rezultate zmanjšati računski korak ali pa povečati red integracijske metode (glej sliko 6.1).



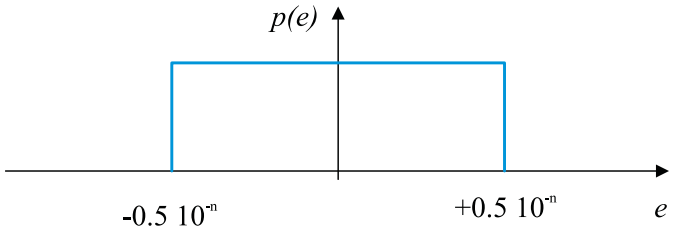
Slika 6.1: Vpliv reda metode na napako numerične metode

Napaka zaradi končne dolžine besede

Numerični integracijski postopki se seveda realizirajo s pomočjo digitalnega računalnika, ki ima zaradi omejene dolžine besede aritmetike, ki jo uporablja simulacijsko orodje, tudi omejeno natančnost. To vodi do napake, ki jo bomo imenovali napaka zaradi končne dolžine besede (roundoff error). Ta vrsta napake je zelo podvržena akumuliranju. Odvisna je

- od časa integriranja (oz. od dolžine simulacijskega teka) $t_{max} - t_0$,
- od reda integracijske metode m (zaradi zahtevnejšega izračunavanja),
- narašča pa tudi pri manjšanju računskega koraka h, saj manjši h pomeni, da imamo več računskih korakov v času integracije $t_{max} - t_0$.

Napako zaradi končne dolžine besede je težko natančno izračunati. Če računalniška aritmetika zaokrožuje rezultate, lahko predpostavimo, da so napake enakomerno (uniformno) porazdeljene med plus in minus $\frac{1}{2}10^{-n}$, kjer je n število decimalnih

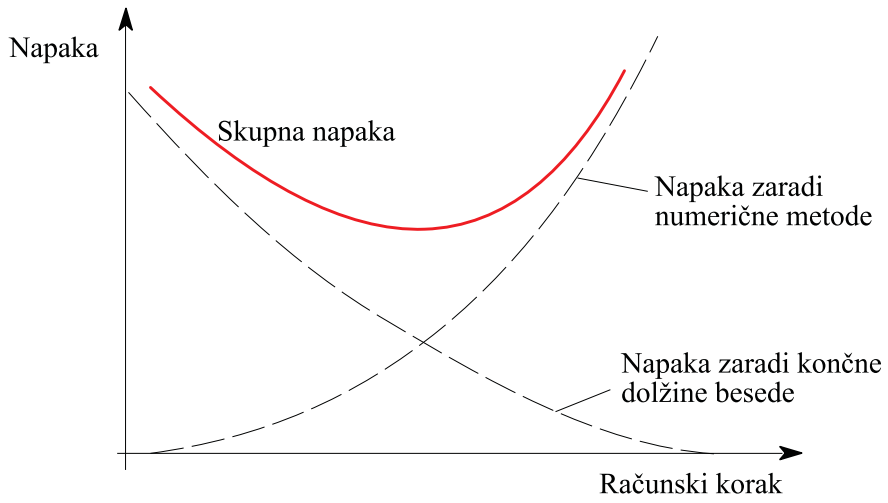


Slika 6.2: Enakomerna porazdelitev napake zaradi končne dolžine besede

digitov (glej sliko 6.2). Ob tej predpostavki ocenimo napako zaradi končne dolžine besede z izrazom (Korn, Wait, 1978)

$$\boldsymbol{e}_{T,k+1} \approx \frac{10^{-n}}{2} \sqrt[2]{\frac{m \cdot (t_{max} - t_0)}{12 \cdot h}} \tag{6.8}$$

Slika 6.3 prikazuje vpliv računskega koraka h na obe vrsti obravnavanih napak ter na skupno napako. Opazimo, da obstaja optimalna dolžina računskega koraka h_{opt}, pri katerem ima skupna napaka minimalno vrednost. Vrednost h_{opt} je seveda težko določiti, saj je odvisna od sistema diferencialnih enačb ($\boldsymbol{f}(t, \boldsymbol{x})$), od integracijske metode in od uporabljenega računalnika oz. njegove aritmetike.



Slika 6.3: Vpliv računskega koraka na celotno napako

6.1.3 Enokoračne integracijske metode

Enokoračne metode so numerični postopki, v katerih se rešitev v trenutku t_{k+1} oceni iz vrednosti le enega predhodnega trenutka t_k. V splošnem dobimo algoritem z razvojem enačbe (6.2) v Taylorjevo vrsto v okolici točke $\boldsymbol{x}(t_k)$

$$\boldsymbol{x}(t_k + h) = \boldsymbol{x}(t_k) + h\dot{\boldsymbol{x}}(t_k) + \frac{h^2}{2!}\ddot{\boldsymbol{x}}(t_k) + \frac{h^3}{3!}\dddot{\boldsymbol{x}}(t_k) + \cdots \tag{6.9}$$

Enokoračna metoda je m - tega reda, če uporabimo $m + 1$ členov v Taylorjevi vrsti. Ker člene višjih redov zanemarimo, lahko ocenimo lokalno napako z izrazom

$$\boldsymbol{e}(t_k) \approx \frac{h^{m+1}}{(m+1)!}\boldsymbol{x}^{(m+1)}(t_k) \tag{6.10}$$

Če uporabimo le dva člena Taylorjeve vrste, dobimo najpreprostejšo enokoračno metodo - Eulerjevo metodo

$$\boldsymbol{x}(t_{k+1}) \approx \boldsymbol{x}(t_k) + h\dot{\boldsymbol{x}}(t_k) \tag{6.11}$$

oziroma

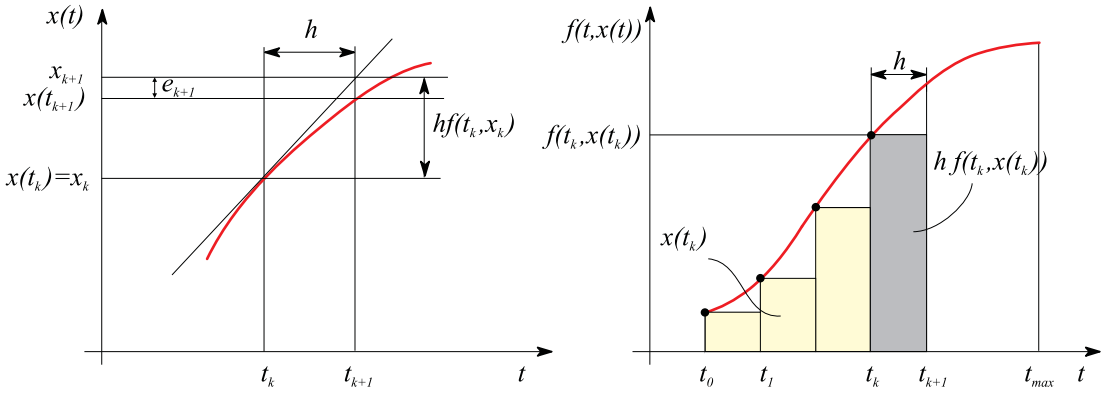
$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + h\boldsymbol{f}(t_k, \boldsymbol{x}_k) \tag{6.12}$$

Enačba (6.12) predstavlja eksplicitno Eulerjevo metodo. Le-to lahko zelo jasno predstavimo s sliko 6.4.

Ker je $\boldsymbol{x}(t_k) = \boldsymbol{x}_k$, predpostavimo, da je rezultat točen v trenutku t_k, oz. da trenutek t_k predstavlja začetni čas integracije. Zaradi lokalne napake numerične metode pa je vrednost $\boldsymbol{x}_{k+1}$ le ocena, ki bolj ali manj odstopa od prave vrednosti $\boldsymbol{x}(t_{k+1})$.

Eulerjev integracijski postopek je intuitivno zelo razumljiv in zahteva izračun le enega odvoda v računskem koraku. Vendar daje sprejemljivo točne rezultate običajno le pri dovolj majhnem računskem koraku h. Manjši računski korak pa seveda pomeni

- večjo porabo računalniškega časa



Slika 6.4: Grafična predstavitev Eulerjeve integracijske metode

- pa tudi večjo napako zaradi končne dolžine besede.

Zato resnejše simulacijske študije zahtevajo numerično bolj izpopolnjene postopke.

Boljšo natančnost seveda dosežemo z uporabo več členov Taylorjeve vrste. Toda to nas vodi do potrebe po ovrednotenju višjih odvodov, kar pa ni enostavno. Runge je prvi pokazal, kako se je v Taylorjevi vrsti možno znebiti višjih odvodov, ne da bi pri tem poslabšali natančnost. Del , ki v Taylorjevi vrsti vsebuje višje odvode, je nadomestil z delom, ki vsebuje nedoločene koeficiente in prve odvode, t.j. funkcijo $\boldsymbol{f}(t, \boldsymbol{x})$ v več točkah intervala med $(t_k, \boldsymbol{x}_k)$ in $(t_{k+1}, \boldsymbol{x}_{k+1})$. Na ta način lahko zapišemo splošno obliko enokoračne metode

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \sum_{i=1}^{v} w_i \boldsymbol{k}_i \tag{6.13}$$

kjer so w_i utežnostni koeficienti, ki jih je potrebno izračunati, v je število izračunov odvodne funkcije v enem računskem koraku, $\boldsymbol{k}_i$ pa določimo iz eksplicitne oblike

$$\boldsymbol{k}_i = h\boldsymbol{f}(t_k + c_i h, \boldsymbol{x}_k + \sum_{j=1}^{i-1} a_{ij}\boldsymbol{k}_j) \qquad i = 1, 2, \cdots, v \tag{6.14}$$

pri čemer je $\boldsymbol{k}_i$ potrebno računati rekurzivno in so c_i in a_{ij} ustrezni koeficienti. Takim algoritmom pravimo tudi eksplicitne enokoračne metode, znane tudi pod

imenom metode Runge - Kutta. Red metode m ni direktno razviden iz enačb (6.13) in (6.14) vendar je enak redu prototipne Taylorjeve vrste.

Če izpeljemo enačbi (6.13) in (6.14) za določen red m, je možno dobiti večje število algoritmov z ustrezno izbiro nekaterih prostih parametrov. Za

- rede $1 \leq m \leq 4$ je število potrebnih izračunanih odvodov na računski korak v kar enako redu metode m,
- za $m > 4$ pa je število potrebnih izračunov vedno večje kot je red.

Postopki za izpeljavo koeficientov c_i, a_{ij} in w_i so opisani v literaturi (Gear, 1971 ali Press in ostali, 1986). Na tem mestu bomo navedli predstavitev enokoračnih metod v skrčeni ali t.i. Butcherjevi obliki (Butcher, 1964). To je nekakšna matrična oblika, ki vsebuje koeficiente enačb (6.13) in (6.14):

$$\begin{array}{c|cccc|c} 0 & & & & & w_1 \\ c_2 & a_{21} & & & & w_2 \\ c_3 & a_{31} & a_{32} & & & w_3 \\ \vdots & \vdots & \vdots & \ddots & & \vdots \\ c_v & a_{v1} & a_{v2} & \cdots & a_{vv-1} & w_v \end{array} \qquad \boldsymbol{c} \mid \boldsymbol{A} \mid \boldsymbol{w}$$

Tipične eksplicitne enokoračne metode (Runge-Kutta) so: Euler (1,1), izboljšana Eulerjeva metoda (2,2), Heun (2,2), Nystrom (3,3), Heun (3,3), klasična metoda Runge-Kutta (4,4), England (4,4), Runge-Kutta-Fehlberg (4,5), Runge-Kutta-Fehlberg (5,6). Številke v oklepajih predstavljajo dvojico (m, v), kjer je (m) red in (v) število izračunov odvodne funkcije v enem računskem koraku. Koeficienti klasične metode Runge - Kutta so prikazani v tabeli 6.1.

Tabela 6.1: Koeficienti klasične metode Runge - Kutta

$$\begin{array}{c|cccc} 0 & & & & \frac{1}{6} \\ \frac{1}{2} & \frac{1}{2} & & & \frac{1}{3} \\ \frac{1}{2} & 0 & \frac{1}{2} & & \frac{1}{3} \\ 1 & 0 & 0 & 1 & \frac{1}{6} \end{array}$$

Z uporabo tabele 6.1 lahko napišemo algoritem v obliki

$$\begin{aligned} x_{k+1} &= x_k + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4 \\ k_1 &= hf(t_k, x_k) \qquad (6.15) \\ k_2 &= hf(t_k + \frac{1}{2}h, x_k + \frac{1}{2}k_1) \\ k_3 &= hf(t_k + \frac{1}{2}h, x_k + \frac{1}{2}k_2) \qquad (6.16) \\ k_4 &= hf(t_k + h, x_k + k_3) \end{aligned}$$

Razen eksplicitnih enokoračnih metod pa so zelo znane tudi implicitne enokoračne metode. Ustrezne izraze dobimo iz enačb (6.13) in (6.14), če zgornjo mejo vsote v enačbi (6.14) spremenimo iz $i-1$ v v

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \sum_{i=1}^{v} w_i \boldsymbol{k}_i \qquad (6.17)$$

$$\boldsymbol{k}_i = h\boldsymbol{f}(t_k + c_i h, \boldsymbol{x}_k + \sum_{j=1}^{v} a_{ij}\boldsymbol{k}_j) \qquad i = 1, 2, \cdots, v \qquad (6.18)$$

Matrika $\boldsymbol{A}$ v Butcherjevi obliki ni več trikotniška, ampak ima elemente tudi nad diagonalo. Glavni problem pa je seveda v tem, ker je koeficient $\boldsymbol{k}_i$ odvisen od koeficientov $\boldsymbol{k}_j$ (za $j = 1, 2, \cdots, v$) tako, da ga ni možno računati z navadnim rekurzivnim postopkom. V tem je glavna težava implicitnih postopkov, saj je potreben poseben iterativni algoritem za izračun $\boldsymbol{k}_i$. Vendar pa v primerjavi z eksplicitnimi metodami dosežemo

- višje rede pri enakem številu izračunanih odvodov, kar pomeni tudi manjšo lokalno napako in
- boljšo numerično stabilnost.

Predstavniki teh metod so naslednji: Gauss (2,1), Gauss (4,2), Radau (5,3), Milne (4,3) in Lobatto (6,4).

Polimplicitne metode so nekaj vmesnega med eksplicitnimi in implicitnimi. S temi metodami skušamo obdržati dobre lastnosti obeh prej opisanih postopkov,

- predvsem dobro numerično stabilnost implicitnih metod in
- računsko učinkovitost eksplicitnih metod (ker ni potrebno iterativno računanje).

Vendar pa je namesto iteracij potrebno računati Jacobijevo matriko

$$\boldsymbol{J} = \frac{\partial \boldsymbol{f}(t, \boldsymbol{x})}{\partial \mathbf{x}} \tag{6.19}$$

in njeno inverzno vrednost za izračun koeficienta $\boldsymbol{k}_i$. Kljub temu pa to zahteva precej manj računalniškega časa kot implicitne iteracije, čeprav je treba v vsakem računskem koraku v krat izračunati Jacobijevo matriko in njeno inverzno vrednost. Zelo znani predstavniki polimplicitnih metod so Rosenbrock-ove metode.

Ocena lokalne napake

Vemo, da je lokalna napaka zaradi numerične metode sorazmerna izrazu h^{m+1}, kjer je h velikost računskega koraka in m red metode. Za zanesljive rezultate je potrebno to napako med simulacijo izračunavati. Najenostavneje to napako ocenimo tako, da izvršimo proces integracije na intervalu $2h$ z dvema različnima računskima korakoma: h in $2h$. Iz razlike lahko ocenimo lokalno napako numeričnega postopka. Če s to oceno izboljšamo rešitev v vsakem računskem koraku, potem seveda izboljšamo tudi celotno rešitev. Pri metodi Runge - Kutta četrtega reda je ocena lokalne napake

$$\boldsymbol{e}_{k,h} \approx \frac{1}{15}(\boldsymbol{x}_{k,h} - \boldsymbol{x}_{k,2h}) \tag{6.20}$$

kjer sta $\boldsymbol{x}_{k,h}$ in $\boldsymbol{x}_{k,2h}$ rešitvi, ki smo jih dobili z računskima korakoma h in $2h$. Izboljšana rešitev je torej

$$\boldsymbol{x}^*_{k,h} = \boldsymbol{x}_{k,h} + \boldsymbol{e}_{k,h} = \frac{1}{15}(16\boldsymbol{x}_{k,h} - \boldsymbol{x}_{k,2h}) \tag{6.21}$$

Opisani postopek pa seveda zahteva precej dodatnega izračunavanja.

Drugi način za ocenitev lokalne napake temelji na osnovi razlike integralov, ki jih izračunamo na računskem koraku z metodama reda m in $m-1$. Pri tem ni

potrebno povsem ločeno izvajati oba postopka, ampak so nekateri vmesni rezultati uporabni v obeh metodah. Zelo znani metodi, ki na ta način preverjata napako, sta metodi Runge-Kutta-Merson in Runge-Kutta-Fehlberg. Pri slednji je potrebno za ovrednotenje integrala po metodi petega reda izračunati le en dodatni odvod glede na metodo četrtega reda.

Stabilnost enokoračnih metod

Znano je, da je zvezni dinamični sistem $\dot{x} = f(t, x(t))$ stabilen, če imajo lastne vrednosti njegove Jacobijeve matrike

$$\boldsymbol{J} = \frac{\partial \boldsymbol{f}(t, \boldsymbol{x})}{\partial \boldsymbol{x}} \tag{6.22}$$

ki jih dobimo z rešitvijo enačbe

$$det(\lambda_i \boldsymbol{I} - \boldsymbol{J}) = 0 \tag{6.23}$$

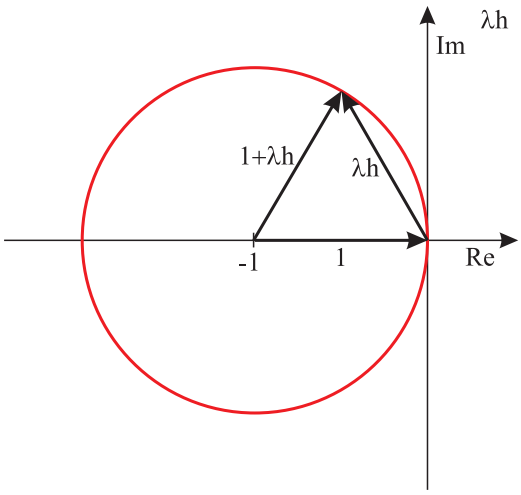
negativne realne dele. Numerični integracijski postopek pa pretvori sistem zapisan z diferencialnimi enačbami v sistem zapisan z diferenčnimi enačbami. Slednji lahko postane nestabilen tudi v primeru, če je originalni zvezni sistem stabilen. Nestabilnost pa pomeni, da se lahko majhne numerične napake med simulacijo izdatno ojačujejo. Poglejmo, kaj predstavlja uporaba Eulerjevega algoritma na enostavni diferencialni enačbi sistema prvega reda:

$$\begin{aligned} \frac{dx}{dt} &= f(t, x) = \lambda x \\ x_{k+1} &= x_k + hf(t_k, x_k) = x_k + h\lambda x_k = x_k(1 + \lambda h) \end{aligned} \tag{6.24}$$

Ker je λ lastna vrednost zveznega sistema, je sistem stabilen za $\lambda < 0$, saj rešitev diferencialne enačbe $x(t) = x(0)e^{-\lambda t}$ upada. Rešitev diferenčne enačbe $x_k = x(0)(1 + \lambda h)^k$ pa upada, če velja

$$|1 + \lambda h| < 1 \tag{6.25}$$

Enačba 6.25 predstavlja enotin krog s središčem v točki -1 ravnine λh (slika 6.5). Notranjost tega kroga predstavlja stabilno področje Eulerjeve integracijske metode (glej krivuljo za $m = 1$ na sliki 6.6). Vidimo, da lahko dosežemo stabilnost z dovolj majhno vrednostjo računskega koraka h.



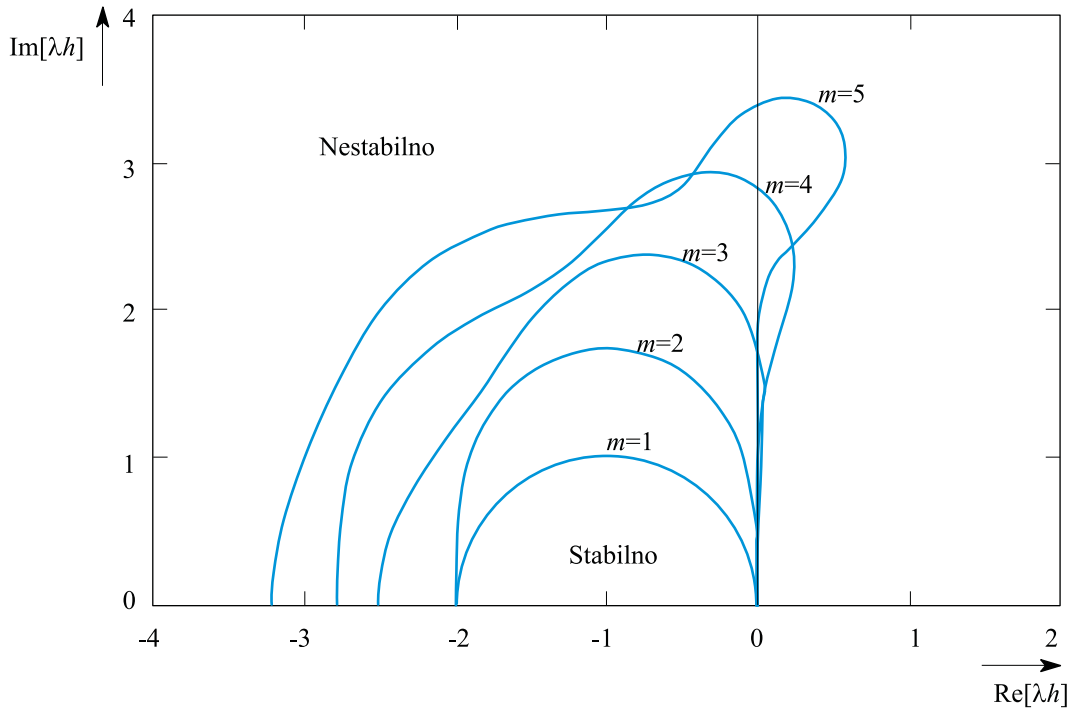
Slika 6.5: Stabilnostno področje Eulerjeve metode

Enake omejitve veljajo pri nelinearnih sistemih, če so le ti opisani s sistemom diferencialnih enačb prvega reda. Za razliko od linearnih sistemov pa so lastne vrednosti Jacobijeve matrike pri nelinearnih sistemih časovno spremenljive, tako da je lahko integracijski algoritem v določenem področju stabilen, v določenem področju pa nestabilen.

Podobno kot za Eulerjevo metodo določimo stabilnostna področja v ravnini λh tudi pri eksplicitnih metodah Runge - Kutta. Slika 6.6 prikazuje stabilnostna področja za Eulerjevo metodo ter za različne metode tipa Runge - Kutta (redi 2–5). Tako na tej kot na naslednjih slikah podajamo le zgornji del λh ravnine, saj so krivulje simetrične glede na realno os. Vse metode so stabilne znotraj zaključenih krivulj.

Čeprav metode Runge-Kutta višjih redov povečajo natančnost, pa se velikost stabilnostnih področij bistveno ne spremeni. Lahko zaključimo, da so metode Runge-Kutta stabilne, če izbrani računski korak h približno zadovoljuje neenačbo

$$|\lambda_{max}|h < 3 \tag{6.26}$$



Slika 6.6: Stabilnostna področja za Eulerjevo metodo (m=1) in za metode Runge-Kutta (m=2,3,4,5)

kjer je λ_{max} maksimalna lastna vrednost Jacobijeve matrike. Ker je λ_{max} obratno sorazmerna minimalni časovni konstanti T_{min} simuliranega sistema, velja tudi naslednja ocena stabilnosti:

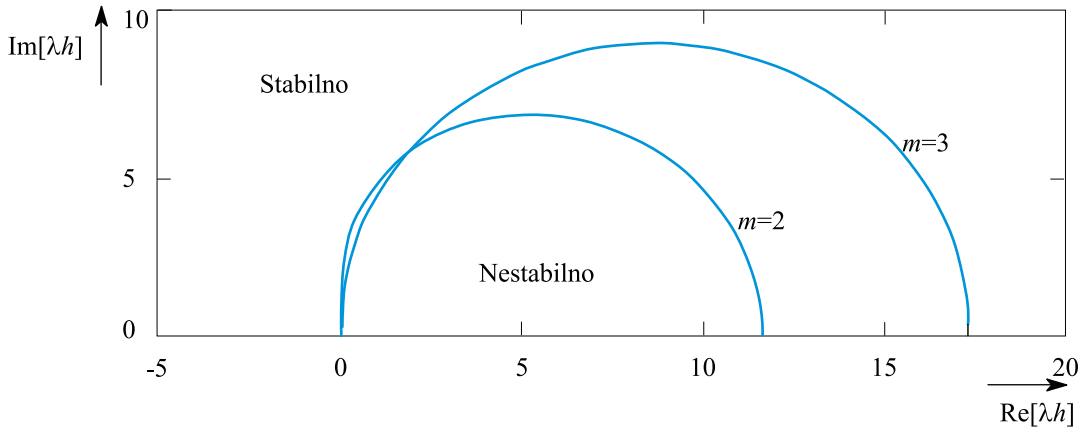
$$h < 3T_{min} \tag{6.27}$$

Toda enačba 6.27 predstavlja le stabilnostni pogoj. Sicer je dobro znano pravilo, da mora biti računski korak nekajkrat manjši od najmanjše časovne konstante, da dosežemo zadovoljivo natančnost (glej sliko 6.3).

Implicitne enokoračne metode imajo večja stabilnostna področja. Le-ta določimo podobno kot pri eksplicitnih metodah. Gaussove metode prvega, drugega (trapezoidno pravilo) in tretjega reda so stabilne v celotnem levem delu ravnine λh. Metodi Radau in Lobatto imata manjši stabilnostni področji toda vseeno večji, kot pri eksplicitnih metodah ($|\lambda_{max}|h < 5$).

Dobre stabilnostne lastnosti odlikujejo tudi polimplicitne enokoračne metode.

Slika 6.7 prikazuje stabilnostna področja Rosenbrock-ovih polimplicitnih metod drugega in tretjega reda. Metodi sta nestabilni znotraj zaključenih krivulj.



Slika 6.7: Stabilnostna področja Rosenbrock-ovih metod (drugega in tretjega reda)

Implicitne in polimplicitne metode omogočajo predvsem z vidika numerične stabilnosti relativno velike računske korake. Toda računalniški čas, ki ga prihranimo z velikim računskim korakom, delno plačamo z iterativnim postopkom za reševanje implicitnih odvisnosti ali za izračun Jacobijeve matrike in njene inverzne vrednosti v primeru polimplicitnih metod. Metode pa so predvsem učinkovite pri simulaciji sistemov z zelo različnimi časovnimi konstantami (togi (stiff) sistemi).

6.1.4 Večkoračne integracijske metode

Enokoračne metode so predvsem pri višjih redih računsko precej potratne, saj zahtevajo večje število izračunov odvodne funkcije v enem računskem koraku. Računsko manj potratne metode je možno razviti, če upoštevamo rezultate več predhodnih izračunov in ne le zadnjega, kot je to primer pri enokoračnih metodah. Tako dobimo večkoračne postopke, pri katerih potrebujemo bistveno manjše število izračunov odvodne funkcije in s tem precej pridobimo na hitrosti.

Večkoračna integracijska metoda ($p+1$ koračna, potrebuje $p+1$ preteklih vrednosti) je definirana z enačbo

$$\boldsymbol{x}_{k+1} = a_0\boldsymbol{x}_k + a_1\boldsymbol{x}_{k-1} + \cdots + a_p\boldsymbol{x}_{k-p} +$$

$$
\begin{aligned}
&+ \quad h[b_{-1}\boldsymbol{f}(t_{k+1}, \boldsymbol{x}_{k+1}) + \cdots + b_p \boldsymbol{f}(t_{k-p}, \boldsymbol{x}_{k-p})] = \\
&= \quad \sum_{i=0}^{p} a_i \boldsymbol{x}_{k-i} + h \sum_{i=-1}^{p} b_i \boldsymbol{f}(t_{k-i}, \boldsymbol{x}_{k-i}) \qquad (6.28)
\end{aligned}
$$

in zahteva podatke o vektorju stanj in odvodov v $p+1$ preteklih vrednostih ($p+1$ koračna metoda).

- Če je $b_{-1} = 0$ potem je metoda eksplicitna ali prediktorska, saj ne potrebuje bodoče vrednosti $\boldsymbol{f}(t_{k+1}, \boldsymbol{x}_{k+1})$.
- Za $b_{-1} \neq 0$ je metoda implicitna ali korektorska, saj je potrebno v vsakem računskem koraku izvesti iterativni postopek za rešitev enačbe (6.28).

Ker so pretekle vrednosti $\boldsymbol{x}_k$ in $\boldsymbol{f}(t_k, \boldsymbol{x}_k)$ shranjene, zahteva večkoračna metoda izračun le enega odvoda v enem računskem koraku.

Koeficienti a_i in b_i so izbrani tako, da je enačba (6.28) veljavna, če izrazimo stanje $\boldsymbol{x}$ s polinomom reda m ($m+1$ koeficientov). Na ta način je možno

- $m+1$ koeficientov od $2p+3$ v enačbi (6.28) izraziti z metodo nedoločenih koeficientov.
- Ostale koeficiente pa določimo tako, da minimiziramo napake in da dosežemo čim večja stabilnostna področja.

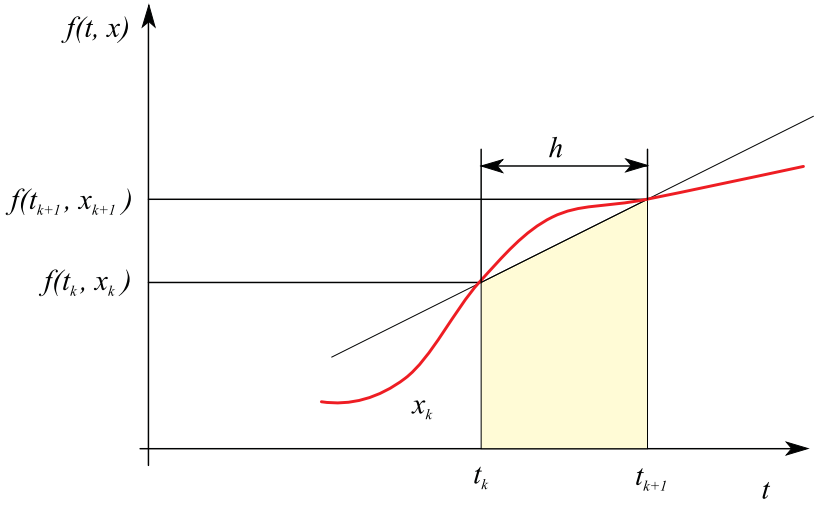
Večkoračne metode predstavljajo problem na začetku, ko pretekle vrednosti $\boldsymbol{x}_k$ in $\boldsymbol{f}(t_k, \boldsymbol{x}_k)$ še niso na voljo. Ta problem se reši tako, da se s pomočjo enokoračne metode predhodno izračunajo potrebni začetni podatki. Uporaba večkoračnih metod je problematična tudi pri nezveznostih v funkciji $\boldsymbol{x}$ ali v njenih odvodih. V takem primeru je aproksimacija (6.28) precej nenatančna, saj stanja $\boldsymbol{x}$ ni možno zadovoljivo natančno izraziti s polinomom m-tega reda. V takih primerih bi natančne rezultate dobili le tako, da bi v trenutku nastopa nezveznosti na novo sprožili večkoračno metodo.

Lokalna napaka numerične metode je odvisna od reda polinoma m, ki izraža stanje sistema in je sorazmerna h^{m+1}.

Če izberemo $p = 0$, $a_0 = 1$, $b_{-1} = \frac{1}{2}$, $b_0 = \frac{1}{2}$ dobimo zelo znano in priljubljeno trapezoidno pravilo

$$\boldsymbol{x}_{k+1} \quad = \quad \boldsymbol{x}_k + \frac{h}{2}(\boldsymbol{f}(t_k, \boldsymbol{x}_k) + \boldsymbol{f}(t_{k+1}, \boldsymbol{x}_{k+1})) \tag{6.29}$$

Princip tovrstne integracije prikazuje slika 6.8.



Slika 6.8: Integracija ob uporabi trapezoidnega pravila

Integral (ploščino) odvodne funkcije na intervalu t_k, t_{k+1} aproksimiramo s ploščino trapezoida.

Ker potrebuje metoda eno predhodno vrednost odvoda $\boldsymbol{f}(t_k, \boldsymbol{x}_k)$ $(p = 0)$, je pravzaprav implicitna enokoračna metoda in kot taka poseben primer večkoračne metode. Metodo lahko učinkovito uporabljamo pri simulaciji enostavnejših problemov, saj porabi malo računskega časa in omogoča dobro numerično stabilnost.

Najbolj znane večkoračne metode so Adamsove metode. Poznamo

- eksplicitne (metode Adams-Bashforth) in
- implicitne (metode Adams-Moulton).

Adams-Bashforth-ovo metodo dobimo, če izberemo

$$p = m - 1 \qquad a_0 = 1 \quad a_1 = a_2 = a_3 = \cdots = a_{m-1} = 0 \qquad b_{-1} = 0 \tag{6.30}$$

Da dobimo metodo m-tega reda, moramo uporabiti $p+1=m$ predhodnih vrednosti. To je torej m koračna metoda.

Adams-Moulton-ovo metodo dobimo, če izberemo

$$p = m-2 \qquad a_0 = 1 \quad a_1 = a_2 = a_3 = \cdots = a_{m-2} = 0 \tag{6.31}$$

Da dobimo metodo m-tega reda, moramo uporabiti le $p+1=m-1$ predhodnih vrednosti. Taka metoda je $m-1$ koračna. Vendar pa je potrebno v vsakem računskem koraku izvršiti iterativni postopek, saj je metoda implicitna. Da zagotovimo enotino ojačenje pri integraciji konstante, mora veljati

$$\sum_{i=-1}^{p} b_i = 1 \tag{6.32}$$

Tabela 6.2 prikazuje koeficiente često uporabljenih Adamsovih metod. Nekatere že znane enokoračne metode lahko obravnavamo kot posebne primere večkoračnih metod.

Tabela 6.2: Koeficienti pri uporabi Adamsovih metod ($a_0 = 1$, vsi ostali $a_i = 0$)

m	b_{-1}	b_0	b_1	b_2	b_3	Ime	Št. korakov
1	0	1	0	0	0	eksplicitna Eulerjeva metoda	1
2	0	$\frac{3}{2}$	$\frac{-1}{2}$	0	0	eksplicitno trapezoidno pravilo	2
3	0	$\frac{23}{12}$	$\frac{-16}{12}$	$\frac{5}{12}$	0	Adams-Bashforth 3. reda	3
4	0	$\frac{55}{24}$	$\frac{-59}{24}$	$\frac{37}{24}$	$\frac{-9}{24}$	Adams-Bashforth 4. reda	4
1	1	0	0	0	0	implicitna Eulerjeva metoda	0
2	$\frac{1}{2}$	$\frac{1}{2}$	0	0	0	implicitno trapezoidno pravilo	1
3	$\frac{5}{12}$	$\frac{8}{12}$	$\frac{-1}{12}$	0	0	Adams-Moulton 3. reda	2
4	$\frac{9}{24}$	$\frac{19}{24}$	$\frac{-5}{24}$	$\frac{1}{24}$	0	Adams-Moulton 4. reda	3

Metode prediktor-korektor

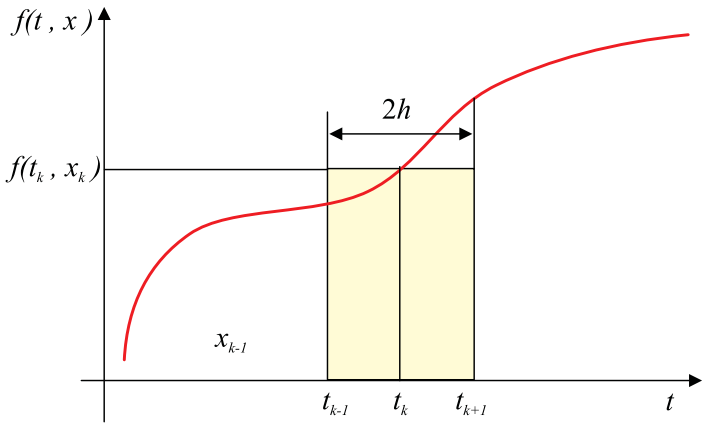
Pri implicitnih enokoračnih ali večkoračnih metodah predstavlja glavni problem uporaba iterativnega algoritma (npr. Newton-Raphson) pri reševanju enačb (6.17), (6.18) in (6.28). Lahko pa uporabimo eksplicitno enokoračno ali večkoračno metodo, da ocenimo $\boldsymbol{x}_{k+1}$. Oceno $\widehat{\boldsymbol{x}}_{k+1}$ nato upoštevamo pri ovrednotenju odvoda $\boldsymbol{f}(t_{k+1}, \widehat{\boldsymbol{x}}_{k+1})$, ki ga potrebuje implicitna metoda. Le-ta torej služi kot korektor za izračun vrednosti $\boldsymbol{x}_{k+1}$. Take metode so poznane pod imenom prediktor - korektor (PC). Red korektorske metode mora biti vedno večji ali enak redu prediktorske metode.

Enostavno metodo prediktor - korektor dobimo, če uporabimo eksplicitno Nystrom-ovo metodo (pravilo srednje točke - midpoint rule) kot prediktorsko metodo in implicitno trapezoidno pravilo kot korektorsko metodo (enačba (6.29)):

$$\widehat{\boldsymbol{x}}_{k+1} = \boldsymbol{x}_{k-1} + 2h\boldsymbol{f}(t_k, \boldsymbol{x}_k) \tag{6.33}$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \frac{h}{2}(\boldsymbol{f}(t_k, \boldsymbol{x}_k) + \boldsymbol{f}(t_{k+1}, \widehat{\boldsymbol{x}}_{k+1})) \tag{6.34}$$

Nystrom-ovo metodo ilustrira slika 6.9.



Slika 6.9: Nystrom-ova metoda, sredinsko pravilo

Vidimo, da sta potrebna dva izračuna odvodov v enem računskem koraku.

Metode prediktor - korektor imajo dobre stabilnostne lastnosti, in relativno majhno lokalno napako numerične metode. Lokalno napako lahko ocenimo iz razlike

med prediktorsko vrednostjo $\widehat{\boldsymbol{x}}_{k+1}$ in korektorsko vrednostjo $\boldsymbol{x}_{k+1}$. Če ta napaka ni znotraj predpisanih toleranc, lahko korektorsko metodo iterativno uporabljamo. Vsaka iteracija zahteva en izračun odvoda in uporabo korektorske enačbe. Postopek običajno konvergira pri dovolj majhnem računskem koraku. Če ne, je potrebno zmanjšati računski korak. Običajno dosežemo dovolj natančne rezultate z le nekaj iteracijami.

Pogosto v prediktor - korektor postopkih uporabljamo metode Adams-Bashforth in Adams-Moulton četrtega reda.

Ocena lokalne napake numerične metode

Postopek za oceno lokalne napake večkoračnih metod je enak kot pri enokoračnih metodah. Za metode Adams-Bashforth in Adams-Moulton četrtega reda podajata oceno napake in izboljšano vrednost rezultata naslednji enačbi:

$$\boldsymbol{e}_{k,h} \approx \frac{1}{15}(\boldsymbol{x}_{k,h} - \boldsymbol{x}_{k,2h}) \tag{6.35}$$

$$\boldsymbol{x}^*_{k,h} = \frac{1}{15}(16\boldsymbol{x}_{k,h} - \boldsymbol{x}_{k,2h}) \tag{6.36}$$

$\boldsymbol{x}_{k,h}$ je rešitev pri računskem koraku h, $\boldsymbol{x}_{k,2h}$ pa je rešitev pri uporabi računskega koraka $2h$. $\boldsymbol{x}^*_{k,h}$ je izboljšana vrednost rezultata.

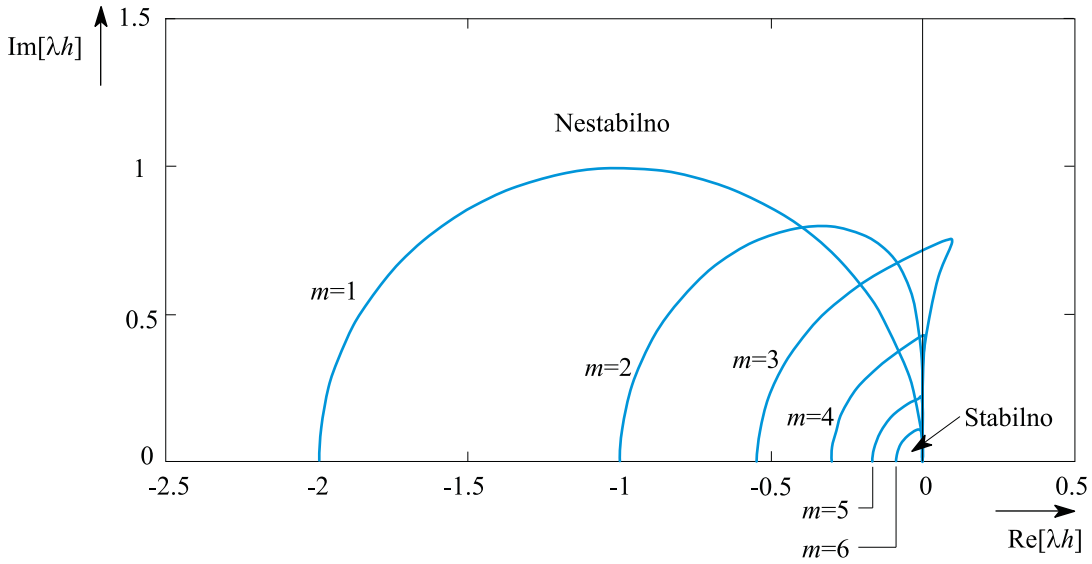
Pri metodah prediktor - korektor je potrebno za ovrednotenje lokalne napake precej manj računalniškega časa, saj dobimo oceno iz razlike med rešitvijo prediktorja in korektorja. Če uporabimo Adams-Bashforth-ovo metodo četrtega reda kot prediktorsko in Adams-Moulton-ovo metodo kot korektorsko, sta ocena napake in izboljšana vrednost podani z enačbama

$$\boldsymbol{e}_k \approx -\frac{1}{14}(\boldsymbol{x}_{k,C} - \boldsymbol{x}_{k,P}) \tag{6.37}$$

$$\boldsymbol{x}^*_k = \frac{1}{14}(13\boldsymbol{x}_{k,C} + \boldsymbol{x}_{k,P}) \tag{6.38}$$

Stabilnost večkoračnih metod

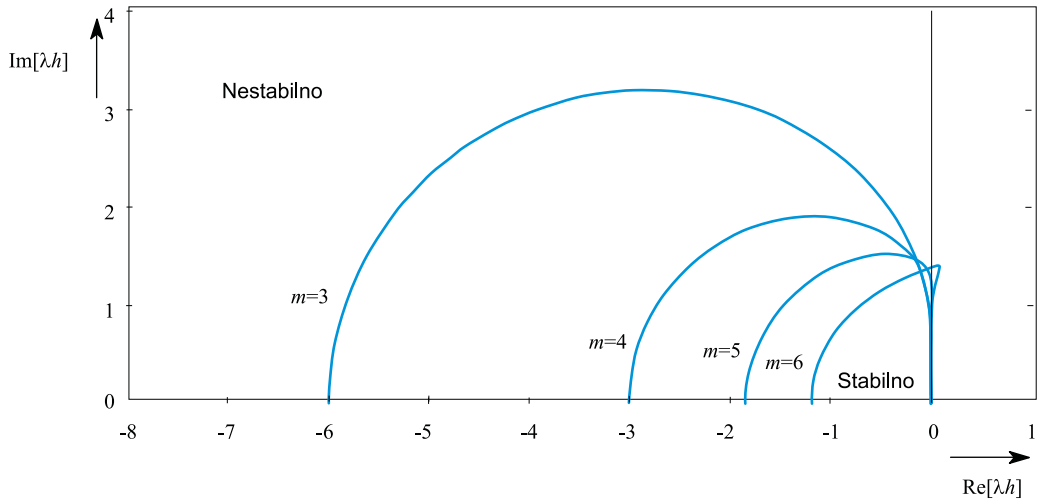
Eksplicitne večkoračne metode imajo majhna stabilnostna področja v λh ravnini. Z večanjem reda metode postajajo ta področja celo manjša. Slika 6.10 prikazuje stabilnostna področja za Adams-Bashforth-ove metode. Metode so stabilne znotraj zaključenih krivulj.



Slika 6.10: Stabilnostna podpodročja metod Adams-Basforth

Implicitne večkoračne metode pa imajo precej večja stabilnostna področja, tako da omogočajo izbiro večjega računskega koraka. Implicitna Eulerjeva metoda in implicitno trapezoidno pravilo sta stabilna v celotni levi polovici ravnine λh. Slika 6.11 prikazuje stabilnostna področja Adams-Moulton-ovih metod. Le-te so stabilne znotraj zaključenih krivulj.

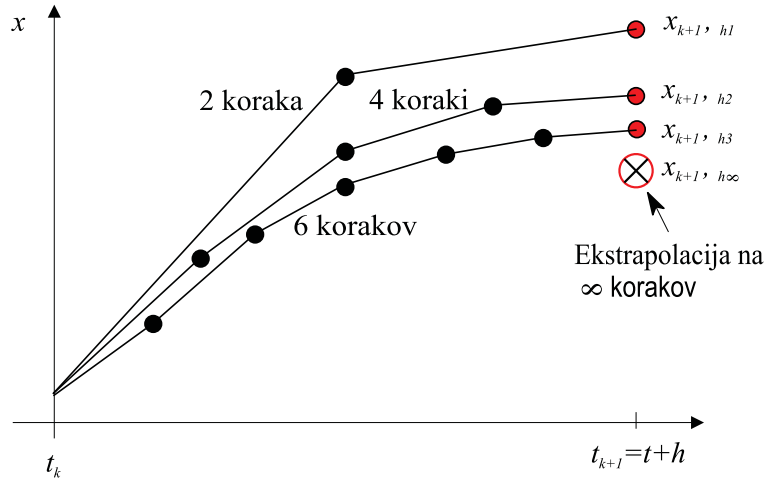
Stabilnostne lastnosti metod prediktor-korektor so odvisne od prediktorske in korektorske metode ter od števila korektorskih iteracij. Stabilnostna področja so večja kot pri prediktorskih metodah (eksplicitne metode) in manjša kot pri korektorskih metodah (implicitne metode).



Slika 6.11: Stabilnostna področja metod Adams-Moulton

6.1.5 Ekstrapolacijske metode

V primerjavi z dosedaj obravnavanimi metodami so ekstrapolacijske metode specifične in se tudi redkeje uporabljajo. Glavna ideja tega postopka je v tem, da se integracija zaporedno vrši na vsakem računskem koraku z različno dolgimi in vedno krajšimi računskimi koraki. Označimo rezultate integracije na intervalu od t_k do t_{k+1} z $\boldsymbol{x}_{k+1,h_1}$, $\boldsymbol{x}_{k+1,h_2}$, $\boldsymbol{x}_{k+1,h_3}$, $\cdots$, $\boldsymbol{x}_{k+1,h_r}$.Postopek prikazuje slika 6.12.



Slika 6.12: Osnovni princip ekstrapolacijske metode

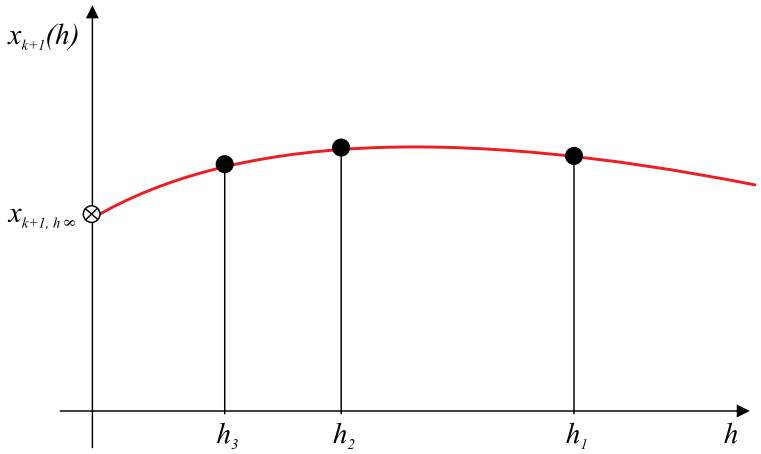
Zaporedje rezultatov $\boldsymbol{x}_{k+1,h_i}$ in zaporedje računskih korakov $h_i, i = 1, 2, \cdots, r$

uporabimo za določitev analitične funkcije $\boldsymbol{x}_{k+1}(h)$. S pomočjo te odvisnosti je možno z ekstrapolacijo dobiti rešitev $\boldsymbol{x}_{k+1,h_\infty}$. Pri tem h_∞ pomeni računski korak dolžine nič, kar teoretično vodi do točne rešitve.

Običajno uporabimo integracijsko metodo nižjega reda in enega od naslednjih ekstrapolacijskih postopkov:

- Richardson-ovo ekstrapolacijo in
- racionalno ekstrapolacijo.

Richardson-ovo ekstrapolacijo dobimo, če izrazimo analitično funkcijo $\boldsymbol{x}_{k+1}(h)$ v obliki polinoma (slika 6.13)



Slika 6.13: Richardson-ova ekstrapolacija

$$\boldsymbol{x}_{k+1}(h) = \boldsymbol{x}_{k+1,h_\infty} + \boldsymbol{\alpha}_1 h + \boldsymbol{\alpha}_2 h^2 + \cdots + \boldsymbol{\alpha}_r h^r \tag{6.39}$$

kjer so $\boldsymbol{\alpha}_i$ koeficienti. V številnih primerih pa dobimo bolj natančne rezultate, če funkcijo $\boldsymbol{x}_{k+1}(h)$ analitično izrazimo kot ulomljeno racionalno funkcijo. Taki ekstrapolaciji pa pravimo racionalna ekstrapolacija in jo za eno komponento vektorja $\boldsymbol{x}_{k+1}(h)$ podaja enačba

$$\begin{aligned} x_{k+1}(h) &= \frac{p_0 + p_1 h + p_2 h^2 + \cdots + p_r h^r}{q_0 + q_1 h + q_2 h^2 + \cdots + q_r h^r} \\ x_{k+1,h_\infty} &= \frac{p_0}{q_0} \end{aligned} \tag{6.40}$$

in so p_i in q_i ustrezne konstante.

V ekstrapolacijskem postopku se uporabljajo različna zaporedja računskih korakov. Literatura najbolj priporoča zaporedje $\frac{h}{2}, \frac{h}{3}, \frac{h}{4}, \frac{h}{6}, \frac{h}{8}, \frac{h}{12}, \cdots$.

Celoten postopek je naslednji:

- Po vsaki integraciji na nekem računskem koraku se izvede ekstrapolacija. Postopek izračuna ekstrapolirano vrednost in oceno lokalne napake.
- Če je napaka večja od dopustne, se izvede nova integracija z manjšim računskim korakom.
- Ko napaka postane manjša od dopustne, se nadaljuje integracija na naslednjem računskem koraku.

Ena od osnovnih ekstrapolacijskih metod je Euler Romberg-ova metoda. V tej metodi se za integracijo na posameznih računskih korakih uporablja Eulerjeva metoda. Boljšo natančnost in ugodnejše stabilnostne lastnosti lahko dosežemo z uporabo trapezoidnega pravila, vendar moramo v tem primeru reševati implicitne enačbe. Obe možnosti uporabljata polinomsko ekstrapolacijo.

Zelo kvalitetne ekstrapolacijske metode so metode Bulirsch-Stoer-Gragg. Nekatere od njih uporabljajo integracijsko metodo s sredinskim pravilom in racionalno ekstrapolacijo.

Natančnost in stabilnost ekstrapolacijskih integracijskih metod je odvisna od uporabljenega integracijskega in ekstrapolacijskega algoritma. Za neko povprečno natančnost potrebujejo ekstrapolacijske metode običajno večje število izračunov odvodne funkcije kot npr. metode Runge-Kutta. Postanejo pa zelo učinkovite pri visokih zahtevah glede točnosti. Nekateri avtorji jih priporočajo tudi za simulacijo sistemov, v katerih nastopajo nezveznosti.

6.1.6 Integracijske metode za toge sisteme

Modeli mnogih fizikalnih sistemov lahko vsebujejo zelo različne lastne vrednosti oz. časovne konstante. Znani so primeri modelov

- dinamike tekočin,

- sistemov vodenja,
- električnih sistemov,
- kemičnih reakcij, itd.

Takim sistemom pravimo togi (stiff) sistemi. Dobro digitalno simulacijsko orodje mora vsebovati vsaj eno integracijsko metodo za učinkovito obravnavo tovrstnih problemov.

Za sistem pravimo, da je tog, če velja neenačba

$$\frac{\max Re[\lambda_i]}{\min Re[\lambda_i]} > 100 \tag{6.41}$$

Pri tem je $\max Re[\lambda_i]$ največja in $\min Re[\lambda_i]$ najmanjša vrednost realnih delov lastnih vrednosti Jacobijeve matrike. Togi sistemi povzročajo problematiko tako glede

- natančnosti kot
- glede numerične stabilnosti.

Pri simulaciji togih sistemov prihaja do naslednjih problemov:

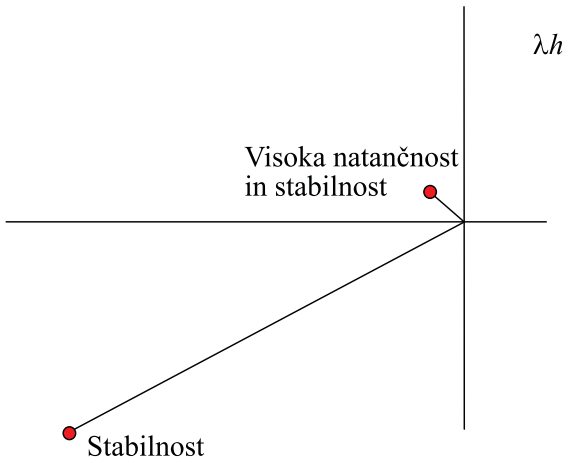
- Če uporabimo metodo z omejenim stabilnostnim področjem, (npr. metoda Runge-Kutta), bode velike vrednosti realnih delov nekaterih lastnih vrednosti povzročile, da bo potrebno za dosego numerične stabilnosti izbrati zelo majhen računski korak. Taka simulacija je s stališča potrebnega računalniškega časa zelo potratna, majhen računski korak pa pomeni tudi povečano napako zaradi končne natančnosti računalnika oz. dolžine besede njegove aritmetike.
- Če pa uporabimo metodo, ki je stabilna na celotnem levem delu ravnine λh (npr. trapezoidno pravilo), se sicer izognemo problemom stabilnosti, toda komponente, ki pripadajo velikim lastnim vrednostim, bodo pri sprejemljivi velikosti računskega koraka vnašale velike napake.

- Tudi iterativni postopki pri implicitnih integracijskih metodah so konvergentni le pri dovolj majhnih računskih korakih. Lastnosti konvergentnosti so seveda odvisne od uporabljene iterativne metode.

Zaradi ugodnih stabilnostnih lastnosti predstavljajo implicitne in polimplicitne metode osnovo integracijskih metod za toge sisteme. Čeprav so take metode računalniško potratne, pa se to poplača z možnostjo povečanja računskega koraka. Na žalost so v celotnem levem delu ravnine λh stabilne le metode nižjih redov.

Gear pa je uspel razviti povsod uveljavljene metode, ki dajejo ugodne stabilnostne lastnosti tudi za rede večje od dve. Uvedel je posebne postopke, ki zagotavljajo

- visoko natančnost in stabilnost za dominantne (majhne) lastne vrednosti in
- samo stabilnost za relativno nepomembne kratke časovne konstante oz. velike lastne vrednosti (glej sliko 6.14).



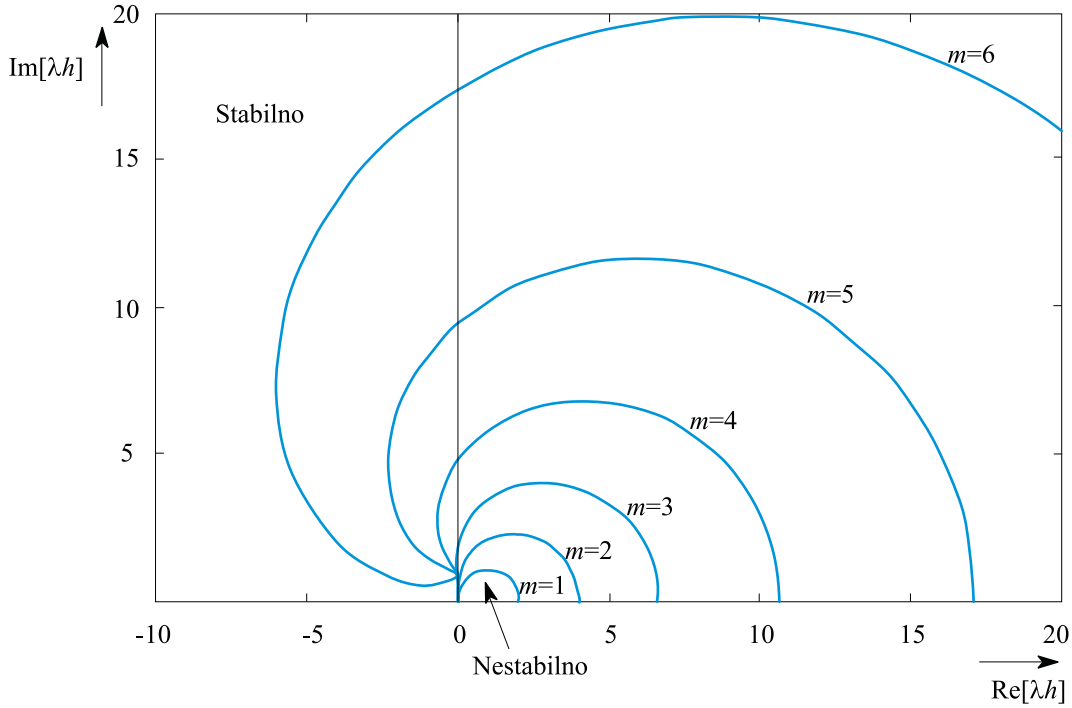
Slika 6.14: Osnovna ideja Gearovih metod

Tako je razvil metode od prvega do šestega reda z naslednjimi značilnostmi:

- Implicitne enačbe se rešujejo z Newton-Raphson-ovim algoritmom.
- Jacobijeva matrika pa se ponovno izračuna le, če poseben test pokaže, da sedanja vrednost ni več ustrezna.

- Lokalna napaka se nadzira med simulacijo z metodami spreminjanja računskega koraka in reda metode.

Slika 6.15 prikazuje stabilnostna področja Gear-ovih metod prvega do šestega reda. Metode so stabilne zunaj zaključenih krivulj.



Slika 6.15: Stabilnostna področja Gear-ovih metod

Poznamo še druge metode, ki so primerne za simulacijo togih sistemov. Ena od njih je polimplicitna Rosenbrock-Wanner-jeva metoda. Veliko pa na tem področju obetajo ekstrapolacijske metode z implicitnim sredinskim pravilom.

6.1.7 Izbira računskega koraka in postopki za njegovo avtomatsko nastavljanje

Izbira računskega koraka

V predhodni obravnavi smo ugotovili, da ima zmanjšanje računskega koraka naslednje posledice:

- zmanjša se lokalna (in s tem tudi globalna) napaka numerične metode,
- izboljšajo se lastnosti v zvezi z numerično stabilnostjo,
- izboljša se konvergenca iterativnih postopkov pri implicitnih metodah,
- poveča se napaka zaradi končne dolžine besede in
- poveča se porabljeni računalniški čas.

Na srečo imajo danes vsa sodobna simulacijska orodja vgrajene algoritme za avtomatsko nastavljanje velikosti računskega koraka. Tako je lahko le-ta relativno velik, če se odvodi spreminjajo počasi, če pa le-ti postanejo živahnejši, se temu ustrezno zmanjša tudi računski korak. Postopek zagotavlja, da je med celotno simulacijo napaka znotraj dopustnih meja. Tak način je posebej učinkovit v primerih, če se med simulacijo spreminja dinamika sistema ali dinamika vhodnih signalov (nelinearni sistemi, časovno spremenljive lastne vrednosti, nezveznosti vhodnih spremenljivk, odvodov,...).

Tako se za uporabnika problem določitve primernega računskega koraka spremeni v problem izbire smiselne tolerance za napako. Toda tudi to ni povsem enostavna naloga. Za smiselno izbiro mora imeti uporabnik nekaj znanja o modelu, ki ga simulira oz. predvsem o tem, kako natančen je model. To pa je v veliki meri povezano z natančnostjo podatkov, ki so bili uporabljeni pri razvoju modela.

Razen tolerance lahko uporabnik poda tudi začetno vrednost računskega koraka. Le-tega izbere tako, da je enak intervalu opazovanja (komunikacijski interval) rezultatov simulacije. S smiselo izbranim začetnim računskim korakom je včasih tudi možno prihraniti nekaj računalniškega časa. Zato pa mora imeti uporabnik nekaj znanja (informacij)

- o dinamiki modela (lastne vrednosti, časovne konstante,...) in
- o dinamiki vhodnih signalov (npr. frekvenčni spekter).

Priporočljivo je izbrati vsaj tako majhno začetno vrednost računskega koraka,

- da je izpolnjen pogoj za numerično stabilnost (npr. $\mid \lambda_{max} \mid h < 3$ za metode Runge-Kutta).

- Razen tega moramo zagotoviti, da na eno časovno konstanto pride vsaj nekaj računskih korakov (velja najmanjša časovna konstanta za metode, ki niso predvidene za toge sisteme in največja pri metodah za toge sisteme).
- Glede na dinamiko vhodnih signalov pa moramo upoštevati Shannon-ov teorem (teoretično naj bosta vsaj dva računska koraka na periodo maksimalne frekvence, praktično pa pet do deset).

Metode za avtomatsko nastavljanje računskega koraka

Kot smo že omenili, imajo sodobna simulacijska orodja algoritme za avtomatsko nastavljanje računskega koraka med simulacijo. Le-ta se običajno lahko giblje med maksimalno vrednostjo, ki je podana s komunikacijskim intervalom in med minimalno vrednostjo, ki jo lahko poda uporabnik. Vse metode avtomatskega nastavljanja temeljijo na sprotnem ocenjevanju lokalne napake numerične metode, ki je za metodo m-tega reda

$$\boldsymbol{e} = \boldsymbol{\Phi} \cdot h^{m+1} \tag{6.42}$$

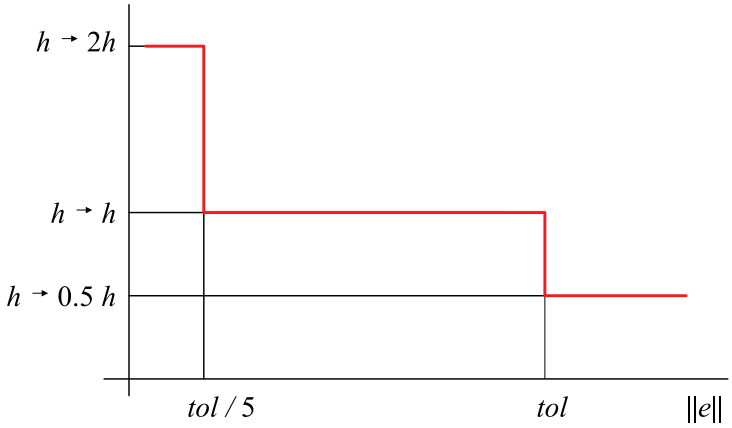
pri čemer je $\boldsymbol{\Phi}$ vektor, ki zavisi od rešitve diferencialne enačbe. Vektor ocene $\boldsymbol{e}$ se izračuna v vsakem računskem koraku na način, ki je odvisen od vrste integracijske metode. Iz vektorja ocene je potrebno izračunati skalarno veličino (normo napake), saj jo moramo v algoritmu za avtomatsko nastavljanje računskega koraka primerjati s toleranco. Uporabljajo se različne norme, ki navadno kombinirajo tudi relativno in absolutno napako. Tipično normo podaja enačba

$$\|\boldsymbol{e}\| = \max_i \left| \frac{e_i}{|x_i| + \eta_i} \right| \tag{6.43}$$

kjer je e_i i-ta komponenta $\boldsymbol{e}$ in η_i skalirni faktor za i-to komponento (običajno je ena). Za $|x_i| \gg \eta_i$ predstavlja norma relativno napako, za $|x_i| \ll \eta_i$ pa absolutno napako. Ob uporabi take norme ima potem tudi toleranca, ki jo podaja uporabnik, absolutno - relativni značaj.

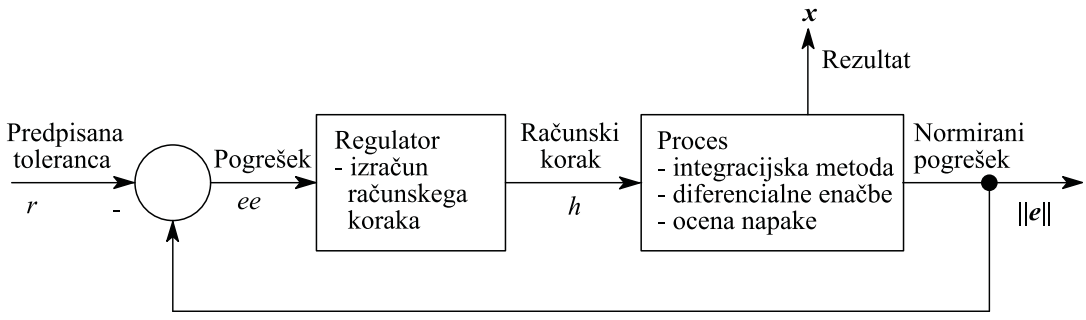
Klasična metoda za avtomatsko nastavljanje računskega koraka temelji na razpolavljanju in podvajanju računskega koraka. Postopek, ki ga ilustrira tudi slika 6.16, je naslednji:

- Če norma napake v nekem trenutku presega podano toleranco, se računski korak razpolovi in integracija se ponovi.
- Če pa norma pomnožena s celoštevilčno konstanto (npr. 5) postane manjša od tolerance, se računski korak podvoji.
- Konstanta večja od ena vnese v algoritem mrtvo cono, kar prepreči morebitna nihanja računskega koraka h. Napake v področju mrtve cone namreč ne spremenijo računskega koraka.



Slika 6.16: Tropoložajni regulator za nastavitev računskega koraka

Opisan postopek je v teoriji avtomatskega vodenja poznan kot tropoložajna nelinearna regulacija. Iz te teorije pa tudi vemo, da je primernejše vedenje regulacijskega sistema možno dobiti z zveznim regulatorjem. Postopek prikazuje slika 6.17.



Slika 6.17: Regulacijski sistem za avtomatsko nastavljanje računskega koraka

Norma lokalne napake $\|\boldsymbol{e}\|$, ki je izhod "procesa", se primerja s predpisano toleranco r. Razliko vodimo v regulator, ki izračuna novo vrednost računskega koraka

h. Regulator mora zagotoviti, da je $\|\boldsymbol{e}\|$ približno enak r. Dober regulator mora zagotoviti ne preveč nihajoči potek računskega koraka.

Na žalost izsledki moderne regulacijske teorije v preteklosti niso bili uporabljeni za obravnavani problem. Mnoge sodobne integracijske metode (npr. Gear-ova metode za toge sisteme (Gear, 1971) uporabljajo za avtomatsko nastavljanje računskega koraka integrirni regulator. Vemo pa, da tak regulator predvsem v smislu stabilnosti slabo vpliva na delovanje regulacijskega sistema, zato gotovo ne predstavlja optimalne rešitve. Nihajoči potek računskega koraka je zlasti očiten, če uporabljamo metode za netoge sisteme (npr. Runge-Kutta) za simulacijo togih sistemov.

Gustaffson (Gustaffson, 1988, Gustaffson, 1990) je med prvimi poskušal uporabiti sodobne regulacijske algoritme za avtomatsko nastavljanje računskega koraka. V eksplicitni metodi Runge-Kutta je uporabil diskretni proporcionalno-integrirni (PI) regulator. Razvil je tudi diskretni model "procesa", ki ga je uporabil pri nastavitvi parametrov regulatorja. PI regulator je omogočil boljše delovanje pri le majhnem povečanju potrebnega računskega časa.

Veljajo naslednje ugotovitve:

- Avtomatsko nastavljanje računskega koraka je relativno enostavno vgraditi v enokoračne integracijske metode.
- Pri večkoračnih metodah pa je problem v tem, da po spremembi računskega koraka pretekle vrednosti, ki se uporabljajo v algoritmu, niso več direktno uporabne, ampak jih je potrebno z interpolacijskimi postopki preračunati na novo vrednost računskega koraka, kar seveda zahteva dodatni računski čas.
- Metode z avtomatskim nastavljanjem računskega koraka se ne uporabljajo za simulacijo v realnem času zaradi računalniške potratnosti in zaradi preveč spremenljive porabe računalniškega časa na posameznih intervalih.

6.1.8 Izbira integracijske metode

Primerno izbrana integracijska metoda mora

- zagotoviti rezultate v okviru predpisanih toleranc

- pri čim manjši porabi računalniškega časa.

Zavedati se moramo, da ni metode, ki bi bila enako primerna za vse probleme. Enostavni problem je običajno sicer možno simulirati s katerokoli metodo, toda brž ko imamo opraviti z realnimi in kompleksnimi problemi, lahko prihranimo veliko računalniškega časa z izbiro primerne metode. V tem podpoglavju bomo podali nekaj nasvetov v tem smislu (Cellier, 1979). Iz problema, ki ga simuliramo, moramo izluščiti nekaj informacij, ki so odločilne pri izbiri integracijske metode. To so

- zahteve po natančnosti,
- pogostost nezveznosti,
- togost in
- oscilatornost

Zahteve po natančnosti

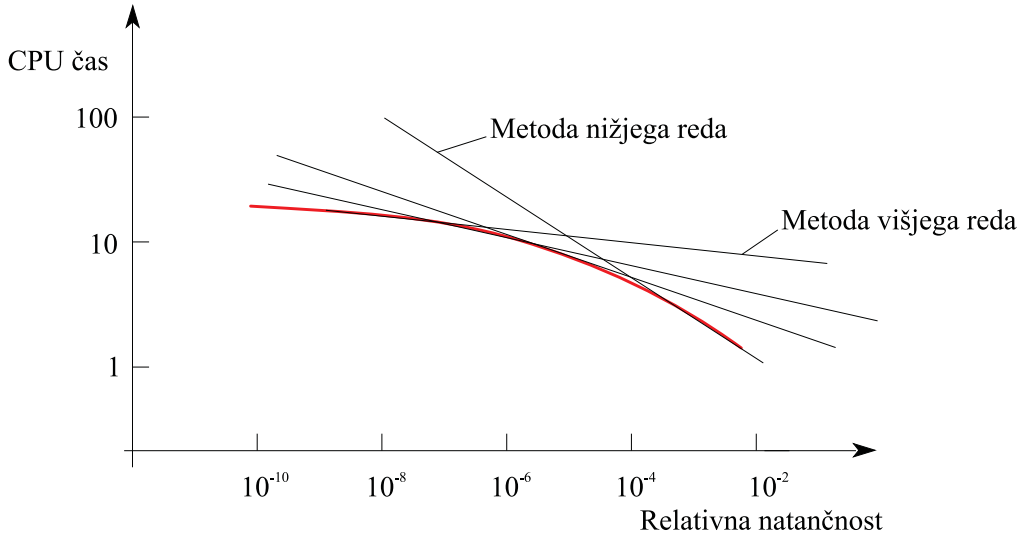
Odvisnost med porabo računalniškega časa in relativno natančnostjo pri različnih redih metod prikazuje slika 6.18.

Velja:

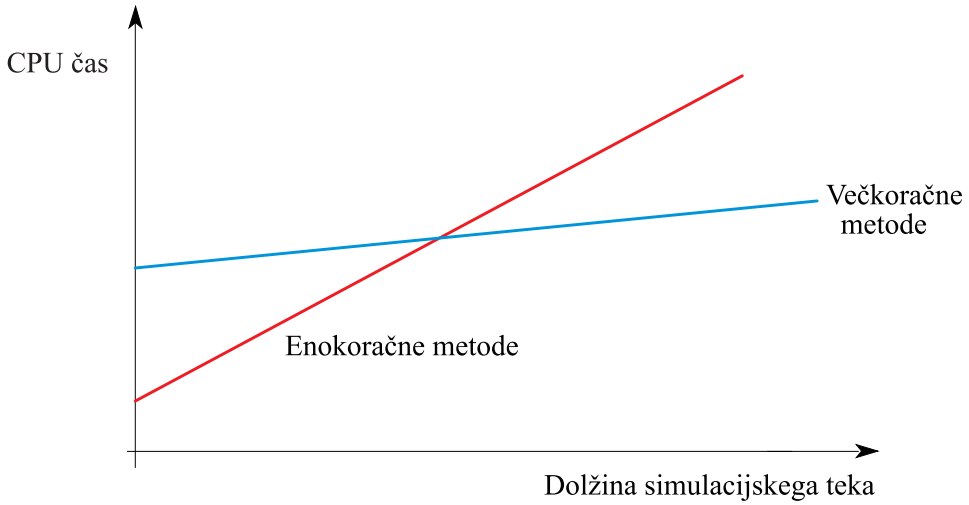
- Uporabimo metodo nizkega reda, če ne potrebujemo velike natančnosti (npr. pri večjih zanemaritvah pri modeliranju, pri nenatančnih meritvah,...)
- in metodo višjega reda pri večji zahtevani natančnosti, saj le-te zagotavljajo ob enakem računalniškem času bolj natančne rezultate.
- Poenostavljeno pravilo pravi, da je za relativno natančnost 10^{-m} potrebno uporabiti metodo m-tega reda.

Pogostost nezveznosti

Slika 6.19 prikazuje odvisnost porabe računalniškega časa od dolžine simulacijskega teka pri enokoračnih in pri večkoračnih metodah. Večkoračne metode



Slika 6.18: Odvisnost potrebnega računalniškega časa od zahtevane relativne natančnosti



Slika 6.19: Odvisnost potrebnega računalniškega časa od dolžine simulacijskega teka

so v primerjavi z enokoračnimi metodami bolj potratne v začetnem delu simulacije, zato pa so bolj ekonomične pri daljših simulacijah. To razložimo s tem, da enokoračne metode (npr. Runge-Kutta) ne potrebujejo nikakršne inicializacije, ker ne potrebujejo predhodnih vrednosti, večkoračne metode (npr. Adams-Moulton) pa morajo na začetku izračunati manjkajoče predhodne vrednosti. To dejstvo je zlasti pomembno, če imajo spremenljivke v modelu pogoste nezvezne

prehode. Druga potrebna informacija, ki jo je treba izluščiti je torej, ali imamo opravka s *pogostimi nezveznostmi*. Za integracijo preko nezveznosti kvalitetni algoritmi detektirajo trenutek njegovega nastopa. Pri enokoračnih metodah je potrebno le spremeniti zadnjo vrednost računskega koraka, tako da se integracija zaključi natančno v trenutku nastopa nezveznosti. Pri večkoračnih metodah pa je treba v vsaki točki nezveznosti ponoviti postopek inicializacije. Zato te metode niso primerne, če imamo med simulacijo pogosto opraviti z nezveznostmi (npr. dvopoložajna regulacija, funkcijski generatorji,...).

Togost

Tretja informacija, ki vpliva na izbiro integracijske metode, je *togost* modela. Le-to določimo z izračunom lastnih vrednosti Jacobijeve matrike (npr. z orodjem MATLAB, če simulacijsko orodje tega ne omogoča). Če je razmerje med maksimalno in minimalno vrednostjo realnih delov lastnih vrednosti večje kot 100, je priporočljivo izbrati integracijsko metodo za toge sisteme (npr. Gear-ovo metodo).

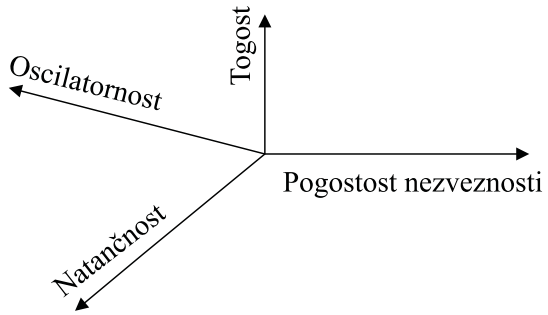
Oscilatornost

Končno predstavljajo poseben problem tudi *oscilatorni sistemi*, to so sistemi, ki imajo dominantne pole blizu imaginarne osi v ravnini λh. V tem primeru moramo nujno uporabiti metodo, ki ima stabilno področje vsaj v celotni levi polovici ravnine λh (implicitne ali polimplicitne metode nizkega reda).

Podatki o omenjenih štirih lastnostih (natančnost, pogostost nezveznosti, togost, oscilatornost, glej sliko 6.20) običajno zadostujejo za ustrezno izbiro integracijske metode. Zato zaključimo z zbranimi nasveti, pri tem pa poudarimo, da je uporabnik seveda omejen z zmožnostmi, ki mu jih daje uporabljeno simulacijsko orodje.

Končna priporočila

- Uporabimo metodo višjega reda za večjo natančnost in metodo nižjega reda, če se zahteva manjša natančnost.



Slika 6.20: Značilnosti modela, ki vplivajo na izbiro integracijske metode

- Uporabimo metodo Runge-Kutta-Fehlberg 4,5, če ne vemo ničesar o numerični problematiki v simulaciji, če je simulirani sistem netog ali če nastopajo pogoste nezveznosti. V okolju Matlab-Simulink sta uporabni metodi ode45 Dormand Prince in ode23 Bogacki-Shampine.

- Uporabimo Gear-ovo metodo za toge sisteme, ekstrapolacijsko metodo za toge sisteme ali kakšno drugo implicitno ali polimplicitno metodo (trapezoidno pravilo, implicitna Eulerjeva metoda,...) pri togih sistemih (v okolju Matlab-Simulink uporabljamo ode15s večkoračno metodo in ode23s enokoračno metodo).

- Uporabimo Adamsovo večkoračno metodo, če ne nastopajo nezveznosti in če sistem ni tog (v okolju Matlab-Simulink je to metoda ode113-prediktor korektor metoda iz Adams-Bashforth in Adams-Moulton).

- Uporabimo ekstrapolacijsko metodo, če se zahtevajo izredno natančni rezultati.

- Uporabimo implicitno metodo nizkega reda pri simulaciji oscilatornih sistemov.

Metode, ki morajo med simulacijo izračunavati Jacobijevo matriko, postanejo z naraščajočim redom diferencialnih enačb izredno računalniško potratne. V tem primeru postanejo nekateri od zgoraj navedenih nasvetov vprašljivi.

6.2 Numerična problematika pri simulaciji sistemov z nezveznostmi

Realni fizikalni sistemi so po naravi strogo gledano zvezni. Zaradi zanemaritev in poenostavljanj med modeliranjem pa njihovi modeli pogosto vsebujejo nezveznosti. Standard CSSL'67 vključuje uporabo gradnikov, ki vsebujejo nezveznosti. Tako imajo orodja v svojih knjižnicah številne nezvezne nelinearne funkcije (npr. mrtva cona, nasičenje, mrtvi hod, histereza, ...). Toda le najsodobnejša orodja te nezvezne funkcije tudi numerično pravilno obdelajo.

Nezveznosti delimo v dve skupini:

1. Nezveznosti, ki nastopajo v vnaprej znanih časovnih trenutkih.
2. Nezveznosti, ki jih proži stanje sistema oz. spremenljivke sistema; trenutek nastopa ni znan vnaprej.

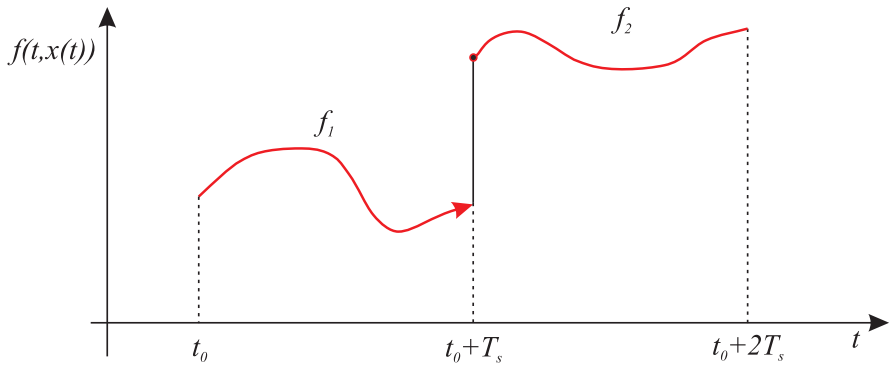
6.2.1 Nezveznosti, ki nastopajo v vnaprej znanih časovnih trenutkih

Ta problem je znan pri diskretnem (računalniškem) vodenju zveznih procesov, ko regulator enkrat na periodo vzorčenja preračuna novo vrednost regulirne veličine, vmes pa je vrednost konstantna. Numerično problematiko je v tem primeru relativno enostavno rešiti. Ker je trenutek nastopa nezveznosti znan vnaprej, mora postopek nastavljanja računskega koraka med simulacijo zadnji računski korak spremeniti tako, da bo zaključek časovno sovpadal s trenutkom vzorčenja. Pri večkoračnih metodah se mora ob vsakem vzorčnem trenutku metoda inicializirati.

Vendar je potrebno tudi v integracijski postopek uvesti nekatere spremembe. Osnovna enačba, ki opisuje enokoračne integracijske metode, je

$$\boldsymbol{x}(t_0 + T_s) = \boldsymbol{x}(t_0) + \int_{t_0}^{t_0+T_s} \boldsymbol{f}(t, \boldsymbol{x}(t))dt \tag{6.44}$$

Naj bo funkcija $\boldsymbol{f}(t, \boldsymbol{x}(t))$ funkcija z nezveznostjo v trenutku $t_0 + T_S$. Prikazuje jo slika 6.21.



Slika 6.21: Nezvezna funkcija $f(t, x(t))$

Funkcija $f(t, x(t))$ je torej

$$f(t, x(t)) = \left\{ \begin{array}{ll} f_1 & t_0 \leq \quad t < t_0 + T_s \\ f_2 & t_0 + T_s \leq \quad t < t_0 + 2T_s \end{array} \right\} \tag{6.45}$$

Modifikacijo, ki je potrebna v enačbi 6.44, opisuje naslednja enačba

$$\boldsymbol{x}(t_0 + T_s) = \boldsymbol{x}(t_0) + \int_{t_0}^{(t_0+T_s)_-} \boldsymbol{f}(t, \boldsymbol{x}(t))dt \tag{6.46}$$

Iz enačb 6.44 in 6.45 ter iz slike 6.21 vidimo, da je potrebno integrirati na intervalu tako, da se zaključi integracija na tem intervalu prej, preden pride do spremembe odvodne funkcije v trenutku $t = t_0 + T_s$. Torej je pravilni vrstni red operacij

- integracija do $t = t_0 + T_s$,
- sprememba odvodne funkcije (izhodov diskretnih blokov v primeru diskretnih regulatorjev) v $t = t_0 + T_s$,
- prikaz rezultatov v $t = t_0 + T_s$.

Zato sta potrebna dva klica (izračuna) odvodne funkcije v trenutku $t = t_0 + T_s$.

6.2.2 Nezveznosti, ki jih proži stanje sistema oz. spremenljivke sistema; trenutek nastopa ni znan vnaprej

Ta vrsta nezveznosti povzroča resnejše probleme. Orodja, ki numerično pravilno obdelajo take vrste nezveznosti, morajo:

- odkriti (detektirati) nezveznost,
- ugotoviti točko nezveznosti (lokacijo), t.j. določiti vrednost neodvisne spremenljivke (običajno čas),
- numerično pravilno integrirati.

Odkrivanje nezveznosti

Gear in Osterby (Gear, Osterby, 1984) predlagata, da ima sistem nezveznost v primeru, če metoda prilagodljivega računskega koraka ugotovi, da je nova vrednost računskega koraka manjša od polovice prejšnjega računskega koraka. Tak način omogoča avtomatsko odkrivanje.

V večini simulacijski orodij pa mora uporabnik definirati t.i. preklopno funkcijo oz. funkcijo nezveznosti (switching function, discontinuity function). To je spremenljivka modela ali linearna funkcija več spremenljivk in ima eno od naslednjih oblik:

$$\begin{aligned} \phi(t) &= x(t) \\ \phi(t) &= x(t) - y(t) \\ \phi(t) &= \sum_{i=1}^{n} [k_i x_i(t) + n_i] \end{aligned} \tag{6.47}$$

Ko ta funkcija postane enaka vrednosti nekega praga (običajno nič), pomeni to indikacijo integracijskemu postopku, da je prišlo do nezveznosti. Nezveznost nastopi pri prehodu iz pozitivne v negativno vrednost ali obratno.

Do nezveznosti pride, ko je $x(t) = 0$ oz. $x(t) = y(t)$, t.j. $\phi(t) = 0$. Torej modeler pomaga simulacijskemu sistemu v postopku detekcije in lokacije nezveznosti. Zelo

znan simulacijski problem je modeliranje skakajoče žoge. Sistem modeliramo tako, da v trenutku, ko višina $h(t)$ postane nič (žoga se dotakne tal)

$$h(t) = 0$$

spremenimo spremenljivko stanja modela- hitrost

$$v = kv \qquad k > -1$$

V tem primeru je torej preklopna funkcija

$$\phi(t) = h(t)$$

V vsakem računskem koraku se razen enačb modelnih spremenljivk ovrednoti tudi preklopna funkcija. Če med dvema zaporednima računskima trenutkoma (angl. mesh points) preklopna funkcija spremeni predznak, pomeni, da je nastopila nezveznost, nakar je treba natančno določiti trenutek nezveznosti.

Problem pa je, če ima preklopna funkcija več prehodov skozi nič na enem računskem intervalu. Če je teh prehodov sodo število, seveda sploh ne odkrijemo nezveznosti, saj se predznak ne spremeni. Do danes še ni povsem zanesljive metode oz. dokaza za detekcijo nezveznosti. V praksi pa opisani postopek deluje zadovoljivo. Če je $\phi(t)$ linearna funkcija spremenljivk stanj, se redko dogodi, da bi imela več prehodov skozi nič.

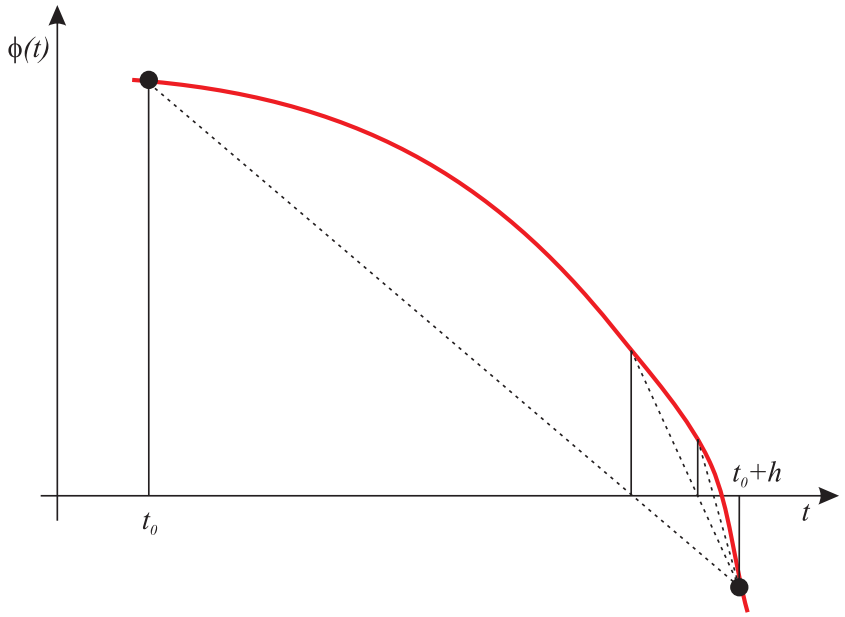
Problem je tudi takrat, če nastopa več preklopnih funkcij, ki imajo ničlo na istem računskem koraku.

Funkcija $\phi(t)$ sicer v ničemer ne vpliva na enačbe, ki opisujejo matematični model.

Ugotavljanje trenutka nezveznosti

Po detektiranju nezveznosti je potrebno določiti trenutek nastopa nezveznosti. Uporabljamo iterativne postopke s hitro konvergenco in širokim konvergenčnim področjem. Itertivni postopek prikazuje slika 6.22.

Pravilno numerično integriranje



Slika 6.22: Iterativni postopek za lokacijo nezveznosti

Ko je algoritem odkril in lociral nezveznost, je nadaljnji postopek enak kot pri nezveznostih, pri katerih vnaprej poznamo trenutek nastopa.

Izvedba preklopne funkcije v okolju Matlab-Simulink

Omenjena preklopna funkcija ima v okolju Matlab-Simulink ime Explicit zero crossing function in daje modelerju možnost, da 'pomaga' integracijskemu algoritmu pri ugotavljanju trenutka nezveznosti. Izvedena je z blokom `Hit crossing` v menuju `Sinks`. Na vhod v ta blok pripeljemo spremenljivko-preklopno funkcijo, ki se primerja z pragom, ki je parameter bloka. Pri prehodu razlike med preklopno funkcijo in pragom skozi nič (v pozitivni, negativni ali v obeh smereh) se sproži postopek za določitev trenutka nezveznosti.

Nekateri bloki, ki sicer povzročajo nezveznosti, imajo opisane mehanizme vgrajene v sam blok, saj je znotraj bloka jasno, kaj povzroča nezveznost. Pravimo, da imajo bloki vgrajeno preklopno funkcijo (Intrinsic zero crossing function). Tabela 6.3 vsebuje bloke, ki imajo vgrajeno preklopno funkcijo.

Primer 6.1 Skakajoča žoga

Primer najdemo v številnih priročnikih simulacijskih orodij. Žogo spustimo z

Tabela 6.3: Bloki v Simulinku, ki imajo vgrajen mehanizem detekcije nezveznosti

`Abs`	`Backlash`
`Compare To Constant`	`Compare To Zero`
`Dead Zone`	`Enable`
`From Workspace`	`If`
`Integrator`	`MinMax`
`Relational Operator`	`Relay`
`Saturation`	`Sign`
`Signal Builder`	`Step`
`Switch`	`Switch Case`
`Trigger`	`Enabled and Triggered Subsystem`

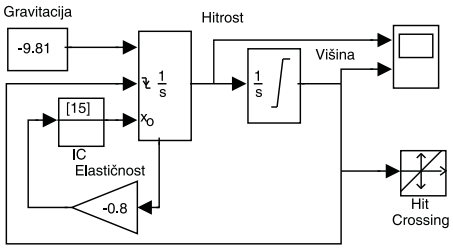
določene začetne višine in z začetno hitrostjo. Ko doseže tla, se seveda odbije. Odboj modeliramo tako, da v trenutku spremenimo hitrost. Hitrost po odboju ima obratno in zaradi izgube energije nekoliko nižjo vrednost. Za modeliranje uporabimo Newtonove zakone

$$\begin{aligned} \sum F_i &= ma \\ -mg &= ma = mh'' \\ h'' &= -g = -9.81\ [m/s^2] \\ h(0) &= 15\ [m] \\ v(0) &= 10\ [m/s] \\ h(t) = 0 \longrightarrow v &= kv \quad k = -0.8 \end{aligned} \tag{6.48}$$

Simulacijsko shemo prikazuje slika 6.23.

Za realizacijo nezveznosti uporabljamo proženi integrator. Ko gre hitrost iz pozitivne v negativno področje, spremenimo vrednost hitrosti v skladu z enačbo $v = -0.8v$. Uporabimo stanje integratorja, ki je po vrednosti enako izhodu integratorja, vendar omogoča, da signal povežemo na vhod za proženje. Uporabimo tudi blok za določitev začetne vrednosti signala - hitrosti (`IC`).

Preklopna funkcija je v tem primeru kar signal $h(t)$, zato na ta signal priključimo blok `Hit Crossing` in na ta način omogočimo določitev trenutka odboja žoge. V tem primeru uporaba tega bloka niti ni potrebna, saj za ustrezni numerični postopek poskrbi tudi proženi integrator, ki spada med bloke z vgrajenim meha-



Slika 6.23: Simulacijska shema v Simulinku za model skakajoče žoge

nizmom za določitev nezveznosti.

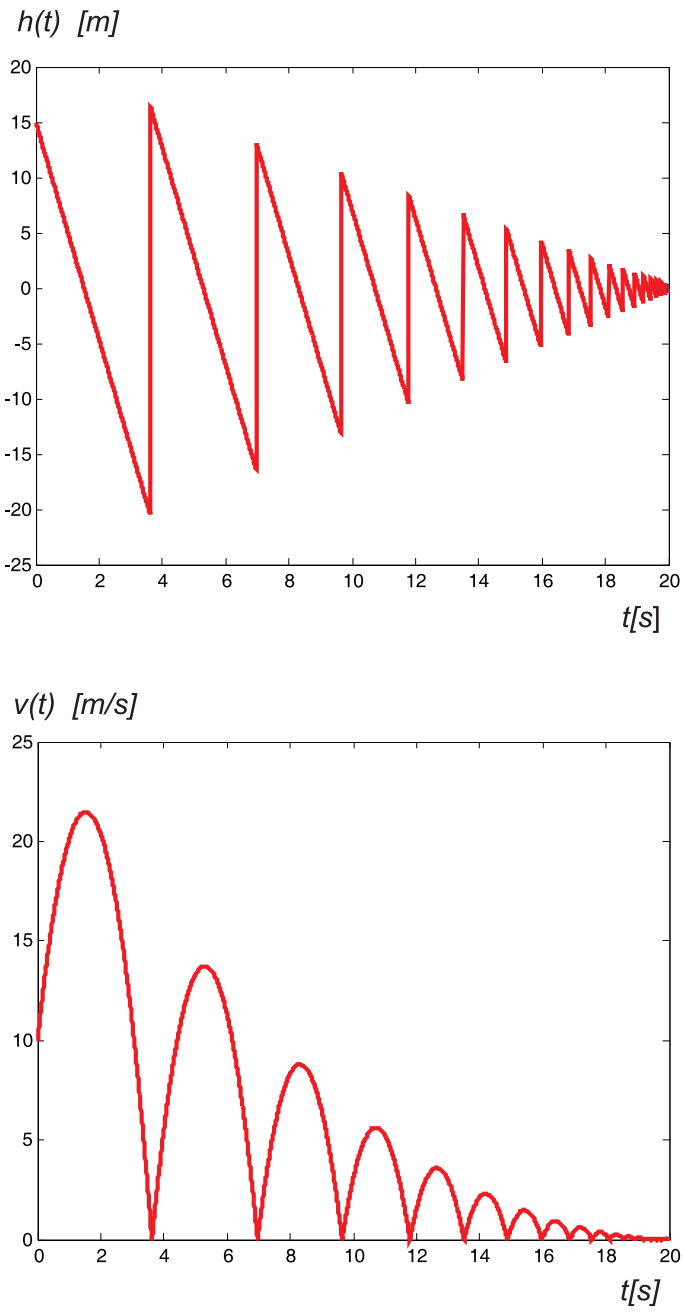
Slika 6.24 prikazuje potek višine $h(t)$ in hitrosti $v(t)$.

□

6.3 Algebrajske zanke

Pred izvajanjem simulacije je potrebno v fazi procesiranja modela razvrstiti stavke oz. bloke, ki opisujejo model. Ko v fazi izračunavanja odvodov nek stavek (blok) pride na vrsto, morajo biti vse spremenljivke desno od enačaja predhodno ovrednotene.

- Pri dobro modeliranem realnem problemu je možno vse spremenljivke modela izračunati iz spremenljivk stanja (iz izhodov integratorjev) in iz vhodnih signalov. V takem primeru vrstni algoritem lahko razvrsti stavke (bloke) iz simulacijskega programa v pravilen proceduralni vrstni red za izračun odvodov (npr. podprogram DERIV).
- Vsak problem, v katerem je možno stavke razvrstiti na opisani način, ima lastnost, da je v vsaki zanki simulacijske sheme vsaj en blok z zakasnitvenim atributom (integrator, diskretna zakasnitev, zakasnilni člen,...).
- Če so modeli v t.i. kanoničnih oblikah, potem ni problemov v razvrščanju.
- Zaradi
  - napak v programiranju (nedefinirane konstante, vhodni signali, tipkarske napake v imenih spremenljivk,...),



Slika 6.24: Potek višine $h(t)$ in hitrosti $v(t)$

– zaradi slabega modeliranja,

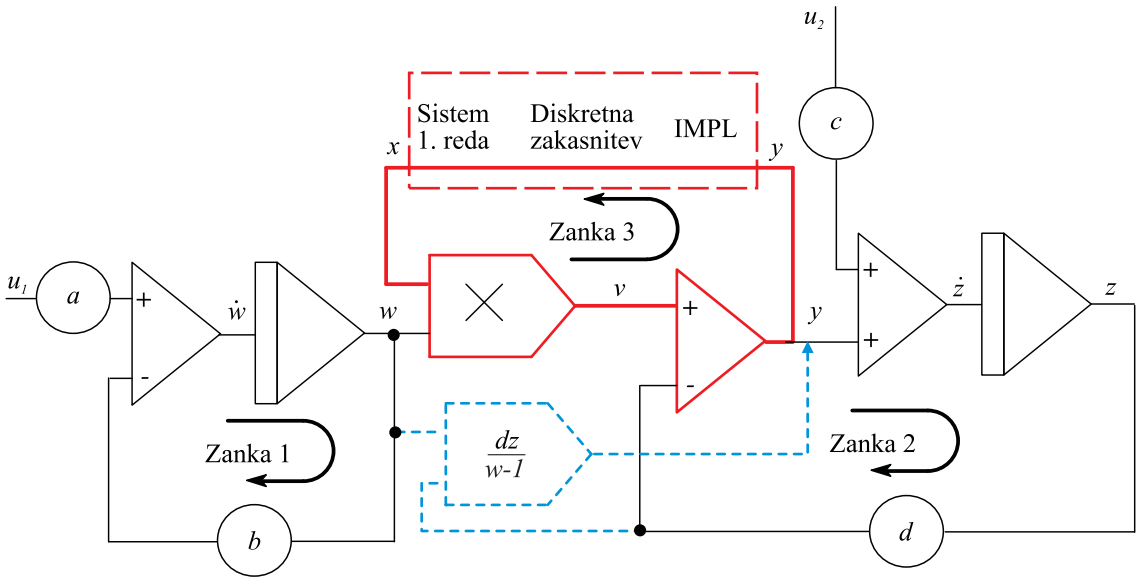
– zaradi kakšnih drugih razlogov, ki zahtevajo bolj eksotično modeliranje ali

– pa zaradi komponent, ki ne vsebujejo zakasnitev (npr. regulator, električno vezje),

vrstni algoritem ne uspe razvrstiti blokov. Eden od razlogov je lahko, da je vhod nekega bloka algebraično odvisen od lastnih izhodov. Taki zanki pravimo *algebrajska zanka*. Pomeni, da ima model vsaj eno zanko, v kateri nastopajo le bloki brez zakasnitvenega atributa (običajno so to statični bloki kot npr. sumatorji, množilniki,...). Take zanke lahko povzročijo zaradi neidealnih komponent probleme celo pri simulaciji na analognem računalniku, pri digitalni simulaciji pa je taka zanka še posebej neprijetna.

V številnih primerih lahko algebrajske zanke odpravimo z algebraično preureditvijo enačb, ki opisujejo model oz. njegove podmodele. Taka rešitev omogoča znaten prihranek potrebnega računalniškega časa, precej pa se poveča tudi natančnost rezultatov. Zato je preureditev enačb (algebraična eliminacija zanke) vedno prvo, kar je vredno poizkusiti. Nekatera orodja, ki ne rešujejo le problema simulacije, ampak nudijo tudi močno podporo pri modeliranju, izvršijo ta postopek avtomatsko (npr. DYMOLA, Dymola, 2008).

Slika 6.25 prikazuje simulacijsko shemo nekega sistema drugega reda. V tej



Slika 6.25: Simulacijska shema nekega sistema drugega reda

shemi se nahajajo tri zanke. Prva in tretja zanka vsebujeta integratorja, druga zanka, ki je poudarjeno označena pa je algebrajska zanka, saj velja

$$y = f(y) \tag{6.49}$$

V tem primeru je algebrajsko zanko možno izločiti z algebraično preureditvijo. Ker je

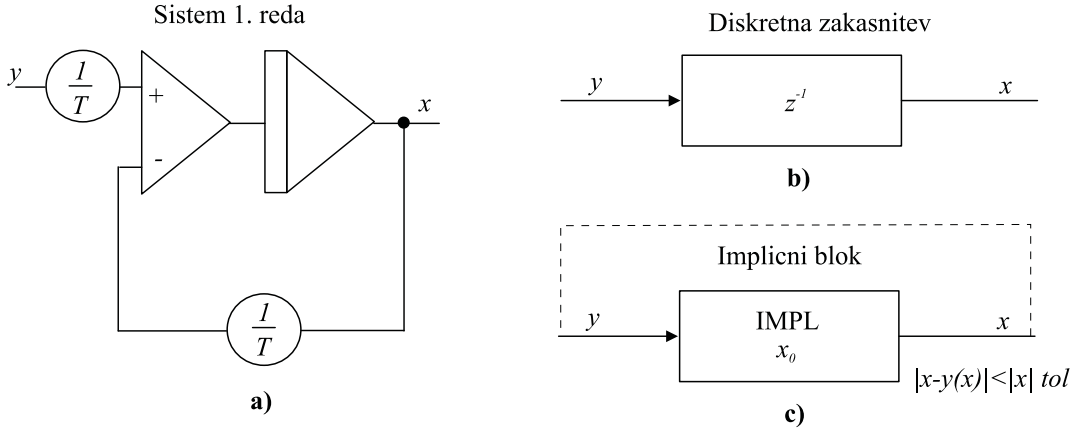
$$y = r - dz = wy - dz \tag{6.50}$$

lahko y izračunamo iz enačbe

$$y = \frac{dz}{w - 1} \tag{6.51}$$

Na sliki 6.25 je postopek označen s črtkano črto.

Včasih pa algebrajske zanke ni možno odpraviti na tako enostaven način. Če simulacijsko shemo na sliki 6.25 pretvorimo v simulacijski program, bo simulacijski prevajalnik verjetno sporočil, da ne more razvrstiti blokov. In če ne vsebuje algoritma za t.i. implicitno reševanje algebrajske zanke, ima uporabnik le to možnost, da zanko umetno prekine, tako da doda zakasnilni člen (sistem prvega reda s časovno konstanto T in ojačenjem ena), ali pa diskretno zakasnitev. Možnosti so prikazane na sliki 6.26.



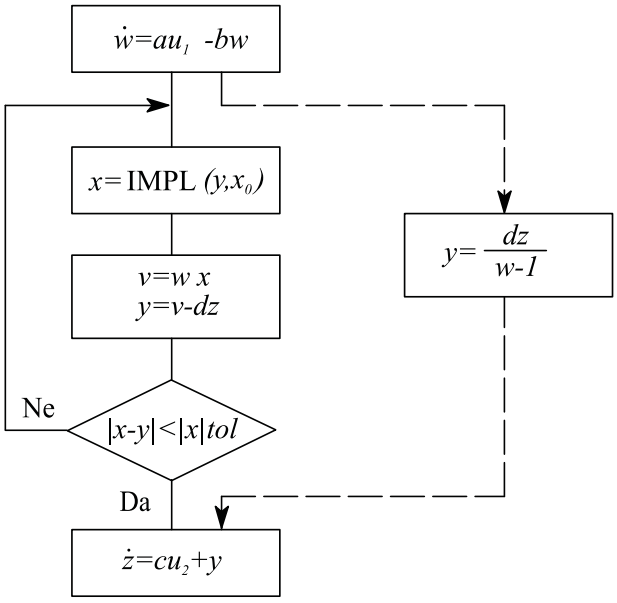
Slika 6.26: Možnosti za prekinitev algebrajske zanke

Če uporabimo zvezni sistem, mora biti časovna konstanta T precej manjša od ostalih časovnih konstant modela. Diskretna zakasnitev pa je običajno enaka komunikacijskemu intervalu. Teoretično bi seveda morali izbrati čim manjšo časovno

konstanto in čim krajšo zakasnitev, praktično pa nas to pripelje do numeričnih problemov v zvezi s togimi sistemi.

Mnogi simulacijski jeziki pa omogočajo reševanje algebrajske zanke na implicitni način. V tem primeru se zanka prekine s t.i. implicitnim blokom (operator IMPL, ki ga vstavi uporabnik), kakor prikazujeta sliki 6.25 in 6.26. Ta blok jemlje prevajalnik kot blok z zakasnitvenim atributom. Zato ga vrstni algoritem postavi pred vse druge bloke v zanki. Med računanjem odvodov postavi IMPL blok na svoj izhod začetno vrednost spremenljivke x. S pomočjo te vrednosti se izračuna vrednost $y(x)$, kakor zahteva zanka. S to vrednostjo se običajno s pomočjo Newton-Raphsonove iterativne metode znotraj IMPL bloka izračuna nova ocena spremenljivke x. Postopek se nato ponavlja, dokler razlika $\mid x - y(x) \mid$ ne postane znotraj tolerance $\mid x \mid tol$ (tol je relativna toleranca).

Za učinkovito izračunavanje je potrebno v iterativnem računanju izvajati le stavke (bloke), ki se nahajajo znotraj algebrajske zanke. Vrstni red izračunavanja prikazuje slika 6.27. Na isti sliki je s črtkano povezavo označen tudi postopek algebraične preureditve enačb.



Slika 6.27: Implicitno računanje algebrajske zanke

Stavki, ki jih je možno razvrstiti tudi brez IMPL operatorja, se izvršijo na začetku. Nato sledijo stavki, ki tvorijo algebrajsko zanko in se iterativno izračunavajo. Na koncu pa se izvajajo stavki, ki za izračunavanje potrebujejo rezultate implicitnega postopka, vendar se ne nahajajo v algebrajski zanki.

Kot smo že omenili, se pri implicitnem računanju običajno uporablja Newton-Raphson-ova iterativna metoda. Postopek se začne izvajati z začetnimi vrednostmi

$$\begin{aligned} x_0 &\ldots \text{definira uporabnik} \\ x_1 &= x_0 + 0.0001x_0 \end{aligned}$$

opisujeta pa ga enačbi

$$c_n = \frac{y(x_n) - y(x_{n-1})}{x_n - x_{n-1}} \tag{6.52}$$

$$x_{n+1} = \begin{cases} \frac{y(x_n) - c_n x_n}{1 - c_n} & c_n \neq 1 \\ y(x_n) & c_n = 1 \end{cases} \tag{6.53}$$

za $n = 1, 2, 3, \ldots$. Začetno izbrana vrednost x_0 se uporablja le na začetku simulacijskega teka. Na naslednjih računskih korakih se izbere začetna vrednost, ki je enaka rešitvi v prejšnjem računskem koraku ($x_0(t_k) = x(t_{k-1})$). Razen začetne vrednosti x_0 mora uporabnik običajno podati tudi relativno toleranco (tol) kot pogoj za končanje iterativnega postopka, maksimalno število iteracij (to je pomembno, če iterativni algoritem ne konvergira) in vrednost, ki se uporabi pri inkrementiranju spremenljivke x (0.0001 v našem primeru).

Pomembno je, da ima uporabljeni iterativni algoritem čim boljše konvergenčne lastnosti. Nobeden pa ne zagotavlja absolutne konvergentnosti.

Implicitni operator pri reševanju algebrajske zanke močno poveča porabo računalniškega časa. Tudi če algoritem naredi le nekaj iteracij, se morajo le-te izvršiti nekajkrat v vsakem računskem koraku (odvisno od uporabljene integracijske metode). Zato je vredno že med modeliranjem vložiti več truda v to, da se po možnosti izognemo algebrajskim zankam.

Poglavje 7

Začetki simulacije z analogno-hibridnimi sistemi

Kot smo opisali v zgodovinskem pregledu (podpoglavje 1.8) so v v letih 1950 do 1980 zvezne dinamične sisteme pretežno simulirali na analogno - hibridnih računalnikih. Kljub temu, da jih danes že skoraj povsod izpodrinili digitalni simulacijski sistemi, pa so še vedno problemi, ki jih je možno zadovoljivo (hitro) simulirati le na analogno-hibridnih sistemih. Ker tudi najbolj neposredno prikazujejo koncepte zveznega modeliranja in simulacije, jim bomo posvetili poglavje. Ob tem naj omenimo, da se nekateri postopki, ki izvirajo iz konceptov analogno-hibridne simulacije (npr. skaliranje), s pridom uporabljajo tudi na drugih področjih.

Najprej na kratko opišimo bistvene značilnosti in bistvene razlike med analognimi in digitalnimi računalniki v smislu simulacije zveznih dinamičnih sistemov.

Digitalni računalniki - Aritmetične in logične operacije se med simulacijo računajo sekvenčno (če imamo v mislih klasično procesno enoto). Nobenih problemov ni pri pomnenju večje količine podatkov (spremenljivke, rezultati, delovni vektorji za realizacijo zakasnitev, točke funkcijskih generatorjev,...). Direktna matematična integracija ni možna, numerični postopki pa so lahko tudi delikatni. Čeprav se z današnjo materialno opremo dosegajo izredne procesne zmožnosti, je simulacija v realnem času na digitalnem računalniku še vedno lahko vprašljiva. Čas, ki ga procesna enota potrebuje za simulacijo, namreč zavisi tako od kompleksnosti simulacijskega modela kot tudi od zahtev za natančnost simulacijskih rezultatov.

Analogno-hibridni računalniki - Analogni računalniki so naprave, ki delujejo

na osnovi analogije veličin, ki jih generirajo in veličin v realnem problemu. Ker so rezultati zvezne veličine (npr. električna napetost pri elektronskem analognem računalniku), jim pravimo tudi zvezni računalniki (Jackson, 1960). Ideja analognega računanja morda izvira iz logaritmičnega računala, ki omogoča matematične operacije s seštevanjem in odštevanjem razdalj na okvirju in na drsniku. Te razdalje torej predstavljajo analogijo problemskim spremenljivkam. Seveda so najučinkovitejši splošnonamenski elektronski analogni računalniki, ki so se razvili iz nekaterih naprav za posebne namene (npr. analizatorji električnih vezij, posebno namenski električni simulatorji).

Zaradi pravega paralelizma operacij in zaradi tega, ker so matematične operacije realizirane z analognimi elektronskimi vezji, predstavljajo dandanes analogni računalniki še vedno najhitrejše simulacijsko orodje. Ob tem je hitrost simulacije neodvisna od kompleksnosti simulacijskega modela. Omogoča simulacijo v realnem, skrčenem in raztegnjenem času. Delo z njem nudi dovolj interaktivnosti in ilustrativnosti.

Seveda pa ima analogni računalnik tudi vrsto slabosti. V prvi vrsti je to gotovo visoka cena, kar velja predvsem za večje računalnike, kajti vsaka operacija zahteva svoj računski element. Pri starejših računalnikih so tudi slabe možnosti za shranjevanje in dokumentiranje simulacijskih modelov in simulacijskih rezultatov. Problematične so tudi nekatere operacije, kot npr. analogni spomin in generacija funkcij. Omeniti pa je seveda treba tudi uporabniško neprijazno programiranje (povezovanje elementov analognega računalnika z žicami) in omejeno točnost računanja zaradi ustreznih lastnosti elektronskih komponent. Razen tega je potrebno skoraj vsak simulacijski problem skalirati, kar pomeni, da je potrebno enačbe preurediti tako, da bodo za predvidena področja spremenljivk realnega problema tudi analogne veličine (t. j. napetosti na elektronskih elementih) znotraj dopustnega območja (npr. od -10 V do 10 V).

Hibridni sistem dobimo s povezavo analognega in digitalnega računalnika, ob simulaciji pa izkoriščamo prednosti obeh. Moderni hibridni računalniki omogočajo programiranje kot v primeru digitalnih simulacijskih jezikov. Glavni problem pri tem predstavlja prenos velike množice podatkov med obema računalnikoma, glavno slabost pa ekstremno visoka cena.

7.1 Operacijski ojačevalnik

Operacijski ojačevalnik je osnovna elektronska komponenta analognega računalnika (EAI Handbook, 1967, Blum, 1969, Tomović, Karplus, 1962) oz. bistvena sestavina večine simulacijskih komponent.

Pri obravnavi vezij z operacijskim ojačevalnikom predpostavimo:

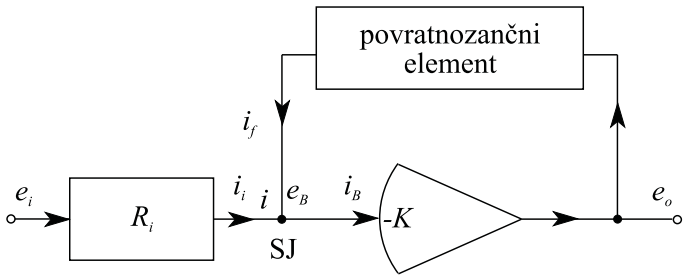
- neskončno ojačenje k (tipično razmerje med izhodno in vhodno napetostjo je sicer velikostnega reda 10^8),
- neskončno pasovno širino (enako ojačenje za vse frekvence vključno s frekvenco nič),
- neskončno vhodno ter nično izhodno impedanco,
- ni lezenja (t.j. nični drift, izhod je nič pri vhodu nič),
- fazni zasuk nič na celotnem frekvenčnem področju (ojačenje $k = -\infty$, kar bi lahko interpretirali tudi kot ojačenje $k = \infty$ in fazni zasuk -180^o).

Zaradi omenjenih lastnosti vsaka komponenta, ki vsebuje operacijski ojačevalnik, obrača predznak vhodne napetosti. Vhodna in izhodna napetost pa sta definirani kot napetosti proti skupni masi.

Ker je izhodna napetost $e_o(t)$ omejena s konstrukcijskimi lastnostmi ojačevalnika največkrat (pri t.i. $10V$ računalnikih) na območje približno $\pm 13V$, mora vhodna napetost $e_i(t)$ pri upoštevanju omenjenega ojačenja ostati manjša od $0.1\mu V$. Zato točki, kjer deluje vhodna napetost, pravimo tudi virtualna masa. To lastnost uporabimo kot še eno predpostavko pri analizi vezja, ki ga prikazuje slika 7.1. Simbol, ki smo ga izbrali za operacijski ojačevalnik, je nekoliko specifičen zaradi simbolov, ki smo jih definirali pri uvedbi splošne simulacijske sheme (glej poglavje 3.1).

Obravnavajmo najprej primer, ko je povratnozančni element na sliki 7.1 upor R_f. Ker je e_B približno enak nič (virtualna masa), je tudi $i_B \approx 0$. Kirchoffov zakon za sumacijsko točko (summing junction SJ) je torej

$$i_i + i_f = 0 \tag{7.1}$$



Slika 7.1: Vezje z operacijskim ojačevalnikom

in

$$\frac{e_i - e_B}{R_i} + \frac{e_o - e_B}{R_f} = 0 \tag{7.2}$$

Iz izraza

$$e_o = -ke_B$$

lahko izrazimo ustrezno napetost e_B

$$e_B = -\frac{e_o}{k} \tag{7.3}$$

Z upoštevanjem enačb (7.2) in (7.3) dobimo izraz

$$\frac{e_o}{R_f} = -\frac{e_i}{R_i} - \frac{e_o}{kR_i} - \frac{e_o}{kR_f} \tag{7.4}$$

Ker je $k \approx 10^8$, lahko zanemarimo drugi in tretji člen desne strani enačbe (7.4), kar daje poenostavljeno končno enačbo

$$e_o = -(\frac{R_f}{R_i})e_i \tag{7.5}$$

Torej relacijo med izhodno in vhodno napetostjo podaja faktor $\frac{R_f}{R_i}$, ki je v posebnem primeru lahko tudi ena. V tem posebnem primeru ustrezno vezje imenujemo invertor. Če ima operacijski ojačevalnik več vhodnih uporov, predstavlja sumator. Če pa je povratnozančni element kondenzator, velja enačba

$$\frac{e_i - e_B}{R_i} + C\frac{d}{dt}(e_o - e_B) = 0 \tag{7.6}$$

Če vstavimo enačbo (7.3) v enačbo (7.6), dobimo izraz

$$C\frac{de_o}{dt} = -\frac{e_i}{R_i} - \frac{e_o}{kR_i} - \frac{C}{k}\frac{de_o}{dt} \tag{7.7}$$

Ker iz enakega razloga kot prej tudi tu lahko zanemarimo zadnja dva člena, velja enačba

$$\frac{de_o}{dt} = -\frac{e_i}{R_i C} \tag{7.8}$$

oz. v integralski obliki

$$e_o = -\frac{1}{R_i C}\int e_i dt \tag{7.9}$$

Torej smo prišli do komponente, katere izhodna veličina je integral časovno spremenljive vhodne veličine in predstavlja elektronski integrator.

Napake, ki se pojavljajo v vezjih z operacijskim ojačevalnikom, so v glavnem posledica neidealnih karakteristik operacijskega ojačevalnika (končno ojačenje, omejena pasovna širina, lezenje, parazitne kapacitivnosti,...(Tomović, Karplus, 1962)).

7.2 Osnovne komponente analognega računalnika

V tem podpoglavju bomo podali kratek opis najpomembnejših komponent, ki jih ima analogni računalnik na razpolago za programiranje. Razdelimo jih lahko na *linearne* in *nelinearne* (EAI Handbook, 1967). Med linearne prištevamo naslednje matematične operacije:

- množenje s konstanto,
- invertiranje,
- algebraično seštevanje,
- zvezna integracija,

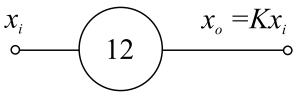
medtem ko so osnovne nelinearne operacije:

- množenje in deljenje spremenljivk,
- generiranje poljubnih funkcij.

Omenjene operacije oziroma ustrezne komponente analognega računalnika s še nekaterimi drugimi, ki bodo omenjene pozneje, omogočajo simulacijo kompleksnih dinamičnih sistemov.

Potenciometer

Potenciometer je element, ki množi določen signal v simulacijski shemi s konstantno vrednostjo, ki je manjša od ena. Ustrezno ikono prikazuje slika 7.2.



Slika 7.2: Ikona za potenciometer

V notranjost ikone običajno vpišemo adreso uporabljene fizične komponente. To velja tudi za druge elemente, ki jih bomo obravnavali.

Ne da bi si podrobneje ogledali elektronsko zgradbo, povejmo (Blum, 1969), da obremenitev potenciometra vpliva tudi na njegovo nastavitev. Zato ga je potrebno nastaviti pod enako obremenitvijo, kot jo čuti kasneje med simulacijo. Če se kasneje zaradi morebitne spremembe v simulacijski shemi (npr. obremenitev z dodatnim elementom, sprememba ojačenja kakšnega elementa) spremeni obremenitev, je potrebno potenciometer ponovno nastaviti.

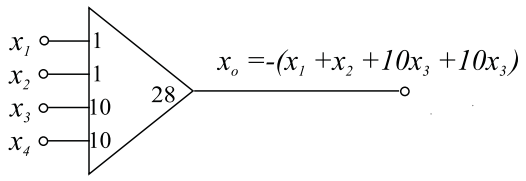
Splošnonamenski analogni računalniki imajo ročno in avtomatsko nastavljive potenciometre, nastavljamo pa jih v za to predvidenem stanju (stanje SP - set potentiometer) med obremenitvijo. Avtomatsko nastavljivi potenciometri se pri starejših računalnikih nastavljajo s servomotorji (zato jim pravimo tudi servopotenciometri) (EAI-580, 1968), novejši računalniki pa imajo digitalno nastavljive potenciometre. Slednji lahko realizirajo množenje s pozitivno in negativno konstanto, tako da iz simulacijske sheme odpadejo številni invertorji, ki so sicer potrebni pri starejših računalnikih (EAI-2000, 1982).

Nekateri potenciometri v shemi vsebujejo le skalirne konstante. Take običajno izberemo kot digitalno nastavljive. Ročni potenciometri pa so primerni na tistih mestih, kjer so vsebovane konstante modela. Na ta način lahko zelo učinkovito proučujemo vpliv teh konstant na simulacijske rezultate v hitrem ponavljalnem delovanju analognega računalnika.

Sumator

Sumator je komponenta, katere izhodna veličina je negativna vsota ustrezno uteženih vhodnih spremenljivk. Uteži definirajo ustrezna razmerja uporov $\frac{R_f}{R_i}$.

Ikono, ki jo uporabljamo pri risanju analognih simulacijskih shem, prikazuje slika 7.3 za eno od možnih konfiguracij vhodov.

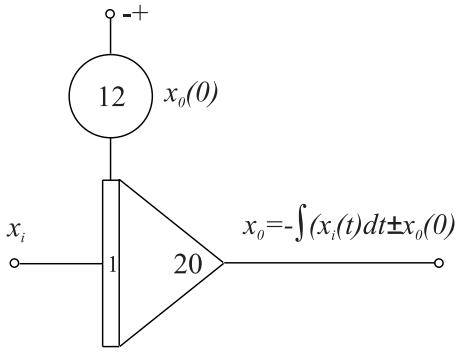


Slika 7.3: Ikona za sumator

Ustrezne uteži (ojačenja) vpišemo v notranjost ikone v bližini vhodnih nožic, adreso sumatorja pa vpišemo v bližini izhoda. Starejši analogni računalniki imajo običajno sumatorje s štirimi vhodi (EAI-580, 1968), medtem ko imajo novejši računalniki zelo različne možnosti (običajno štiri do sedem vhodov, nekateri imajo utež 1, nekateri 10) (EAI-2000, 1982).

Integrator

Integrator je seveda osrednja komponenta analognega računalnika. Ikono prikazuje slika 7.4.



Slika 7.4: Ikona za integrator

Začetni pogoj definiramo preko posebnega vhoda, na katerega je preko potenciometra priključena referenčna napetost analognega računalnika (običajno $\pm 10V$), vhod pa priključimo na spremenljivko, ki jo je potrebno integrirati. Zavedati se moramo, da tudi integrator obrača predznak tako vhodnemu signalu kot začetnemu pogoju. Realizira matematično operacijo

$$x_o(t) = -(\frac{1}{R_i C} \int_0^T x_i(t)dt \pm x_o(0)) \tag{7.10}$$

Starejši tipi analognih računalnikov imajo tudi integratorje z več vhodi, torej vsebujejo sumator in integrator v eni komponenti (EAI-580, 1968), medtem ko imajo novejši tipi običajno le integratorje z enim vhodom (EAI-2000, 1982). Konstanta $\frac{1}{R_i C}$ določa ojačenje (integrirno časovno konstanto) vhoda in s tem hitrost integracije. Če vsem integratorjem v shemi spremenimo to konstanto, problem časovno preskaliramo, kar pomeni, da vse oblike spremenljivk ostanejo enake, le da je na časovni osi drugo merilo.

Med delovanjem se lahko integratorji nahajajo v različnih stanjih:

- stanje začetnih pogojev (IC - initial conditions),
- stanje integriranja (OP - operate),
- stanje držanja (HD - hold).

V stanju začetnih pogojev se na izhodu pojavi napetost, ki je določena s posebnim vhodom za začetni pogoj. V stanju integriranja integrator integrira vhodni signal, v stanju držanja pa integrator drži tisto napetost, ki je bila na njegovem izhodu v trenutku preklopa v to stanje.

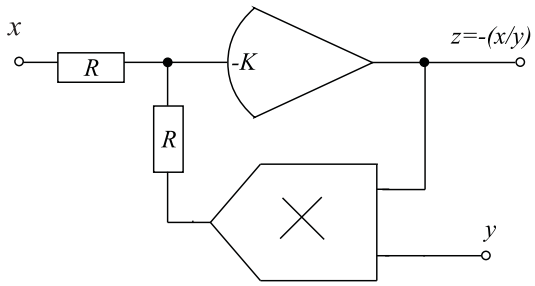
Stanja se krmilijo s posebnimi logičnimi signali. Ti signali lahko enotno krmilijo vse integratorje v analogni simulacijski shemi (običajno pri t. i. ponavljalnem delovanju, ko računalnik ponavlja simulacijske teke s preklapljanjem med stanjema IC in OP), možno pa je tudi ločeno krmiliti vsak integrator posebej (bolj zahtevne sheme pri hibridnem računanju).

Nelinearne komponente

V primerjavi z digitalno simulacijo so nelinearne komponente dosti bolj problematične tako s stališča realizacije ustreznih analognih vezij, ki naj bi zagotovila ustrezno natančnost kot tudi s stališča uporabe teh komponent.

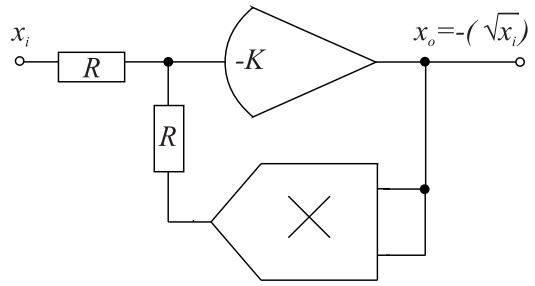
Eden od pomembnejših nelinearnih komponent je *množilnik*. Uporabljajo se zelo različne realizacije, različne ikone in različni načini povezovanja pri vključitvi množilnika v analogno simulacijsko shemo. Mi bomo uporabljali ikono, ki smo jo definirali pri obravnavanju simulacijske sheme. Dodatno moramo upoštevati le obračanje predznaka.

Delilnik je realiziran z množilnikom v povratni zanki ojačevalnika z visokim ojačenjem. Realizacijo prikazuje slika 7.5.



Slika 7.5: Realizacija delilnika

Na podoben način realiziramo komponento za kvadratni koren. Množilnik v povratni zanki mora v tem primeru realizirati kvadrat izhodne veličine operacijskega ojačevalnika. Realizacijo prikazuje slika 7.6.



Slika 7.6: Realizacija kvadratnega korena

Splošnonamenski računalniki lahko vsebujejo tudi komponente za realizacijo posebnih funkcij, kot npr. $sin, cos, arcsin, log$ itd. kot tudi *generatorje poljubnih funkcijskih odvisnosti* pri katerih je potrebno definirati ustrezne lomne točke, vmes pa se signali linearno interpolirajo. Pri starejših analognih računalnikih se za realizacijo uporabljajo diodna vezja (za opis funkcijskega generatorja je potrebno vnesti vrednosti neodvisne spremenljivke v lomnih točkah in naklone proti naslednjim lomnim točkam). Pri novejših analognih računalnikih pa so funkcijski generatorji digitalno nastavljivi, kar pomeni, da preko tipkovnice nastavimo vrednost neodvisne in odvisne spremenljivke v lomnih točkah.

7.3 Programiranje na analognem računalniku

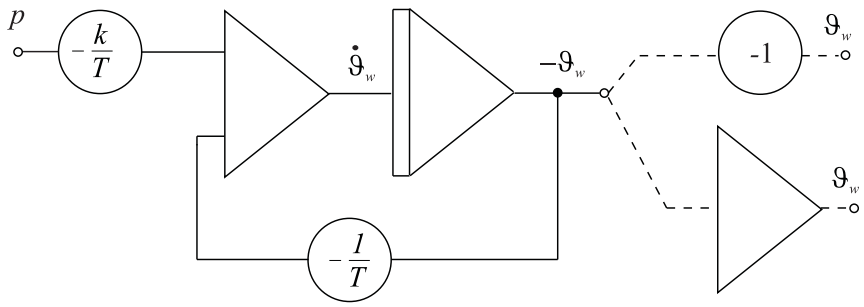
Osnova za programiranje analognega računalnika je *analogna simulacijska shema*, ki jo seveda tesno povezana z indirektnim pristopom pri reševanju diferencialnih enačb s simulacijo. Postopek smo obravnavali v podpoglavju 3.2. Osnovne razlike, ki jih je treba upoštevati so v tem, da analogne komponente obračajo predznak, da posamezni proizvajalci v uporabniškem priročniku navajajo različne oblike ikon za funkcionalno enake komponente in da se računalniki ločijo v naborih (predvsem nestandardnih) komponent. Postopek, po katerem pridemo do analogne simulacijske sheme, bomo prikazali na nekaterih primerih.

Primer 7.1 Model ogrevanja

Enačbo, ki jo dobimo v prvem koraku indirektnega pristopa (primer 3.1, enačba (3.4)), preoblikujemo tako, da ima celotna desna stran izpostavljen negativni predznak

$$\dot{\vartheta}_w = -(\frac{1}{T}\vartheta_w - \frac{k}{T}p) \tag{7.11}$$

Izpostavljeni negativni predznak bomo uporabili za realizacijo obračanja predznaka ustreznega sumatorja, ki izračuna najvišji odvod. S podobnim postopkom, kot v podpoglavju 3.2, dobimo simulacijsko shemo, ki jo prikazuje slika 7.7.

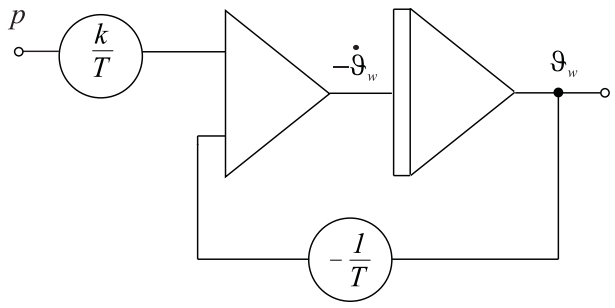


Slika 7.7: Analogna simulacijska shema modela ogrevanja

Ker tudi integrator obrača predznak, dobimo na njegovem izhodu spremenljivko $-\vartheta_w$. Zato potrebujemo še en potenciometer z ojačenjem -1 ali pa invertor. Temu pa se izognemo, če pomnožimo enačbo (7.11) z -1 in dobimo enačbo

$$-\dot{\vartheta}_w = -(-\frac{1}{T}\vartheta_w + \frac{k}{T}p) \tag{7.12}$$

Zgornja enačba predstavlja sumator, ki generira spremenljivko $-\dot{\vartheta}_w$, z ustrezno integracijo pa dobimo spremenljivko ϑ_w. Torej prihranimo eno analogno komponento. Analogno simulacijsko shemo prikazuje slika 7.8.



Slika 7.8: Modificirana analogna simulacijska shema modela ogrevanja

Obe shemi upoštevata, da lahko potenciometri množijo tudi z negativnimi konstantami. V primeru starejših analognih računalnikov pa je potrebno uporabiti invertorje.

Ko narišemo analogno simulacijsko shemo, je smiselno pogledati, ali imajo vse povratne zanke rezultirajoče negativne predznake (liho število negativnih predznakov v zanki). V nasprotnem primeru bodo časovni poteki verjetno nestabilni, kar pomeni, da smo pri razvoju sheme najbrž naredili napako. □

Primer 7.2 Avtomobilsko vzmetenje

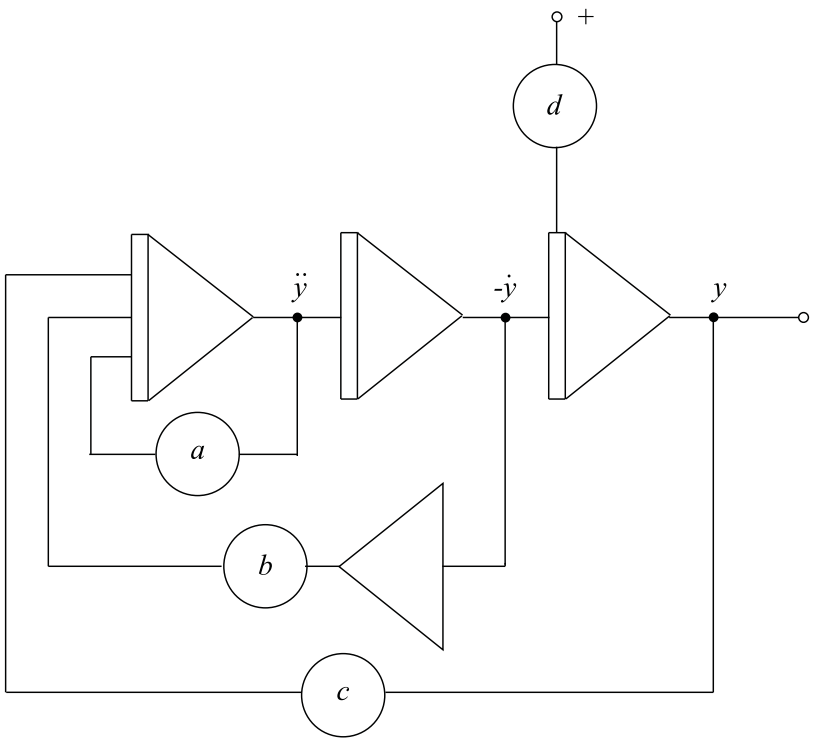
V tem primeru bomo za model avtomobilskega vzmetenja, ki ga opisuje diferencialna enačba (enačba (1.9), primer 1.1)

$$\dddot{y} + a\ddot{y} + b\dot{y} + cy = 0 \qquad y(0) = -d \tag{7.13}$$

razvili simulacijsko shemo za starejše tipe analognih računalnikov (sumacijski integratorji, potenciometri, ki so lahko le pozitivni, invertorji, ...). Enačbo (7.13) preuredimo v obliko

$$\frac{d}{dt}(\ddot{y}) = -(a\ddot{y} + b\dot{y} + cy) \qquad y(0) = -d \tag{7.14}$$

Namesto spremenljivke $\dddot{y}$ na levi strani izrazimo odvod spremenljivke $\ddot{y}$. Spremenljivka $\ddot{y}$ predstavlja izhod sumacijskega integratorja, njen odvod pa ni dostopen. Analogno simulacijsko shemo prikazuje slika 7.9.



Slika 7.9: Analogna simulacijska shema modela avtomobilskega vzmetenja

Za spremembo predznaka spremenljivke $-\dot{y}$ smo uporabili invertor, kajti potenciometri lahko množijo le s pozitivno konstanto. Ker je začetni pogoj negativen in ker integrator obrača tudi predznak začetnega pogoja, moramo le tega generirati z uporabo pozitivne reference. □

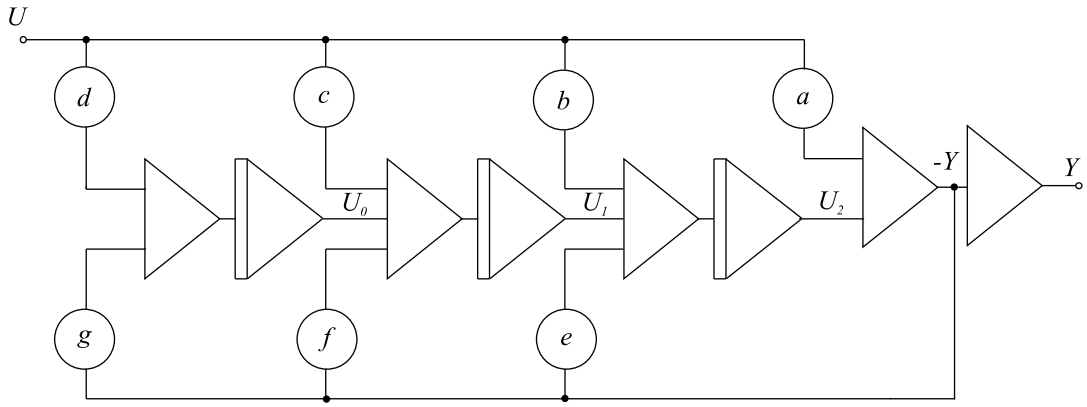
Primer 7.3 Metode za simulacijo prenosnih funkcij

V podpoglavju 3.5 smo si ogledali *vgnezdeno* in *delitveno* metodo za simulacijo prenosne funkcije

$$\frac{Y(s)}{U(s)} = \frac{as^3 + bs^2 + cs + d}{s^3 + es^2 + fs + g} \tag{7.15}$$

Splošno simulacijsko shemo pri uporabi vgnezdene metode prikazuje slika 7.10. Iz sheme je razvidno, da je realizacija pomožnih spremenljivk (enačbe (3.20)) v analogni simulacijski shemi povsem enaka, saj integratorji vedno nastopajo v parih s sumatorji, torej ta kombinacija ne obrača predznaka. Ker pa izhod realizira enačba (3.21), je očitno, da je edina razlika med splošno in analogno simulacijsko shemo v predznaku spremenljivke na izhodu zadnjega (oz. na sliki

7.10 predzadnjega) sumatorja. Torej na tem mestu dobimo spremenljivko $-Y$, zato je potrebno zamenjati tudi predznake potenciometrov e, f in g. Ustrezno analogno simulacijsko shemo prikazuje slika 7.10.



Slika 7.10: Analogna simulacijska shema za simulacijo prenosne funkcije po vgnezdeni metodi

Če potrebujemo tudi spremenljivko y, moramo uporabiti še en invertor.

Več sprememb glede na splošno simulacijsko shemo nastane pri uporabi delitvene metode. V tem primeru je potrebno enačbo (3.25) spremeniti v obliko

$$s^3W = -(es^2W + fsW + gW - U) \tag{7.16}$$

enačbo (3.26) pa v obliko

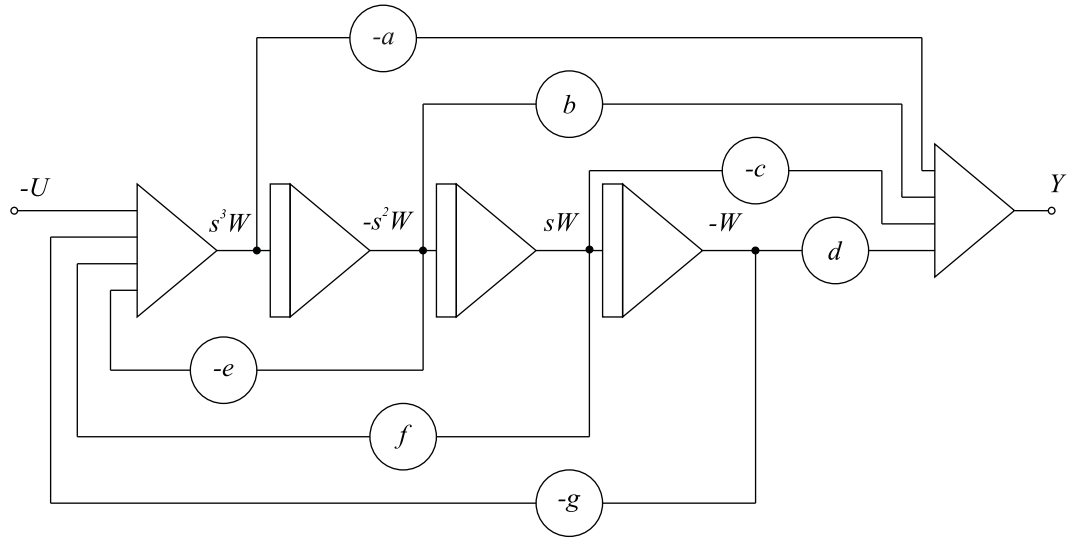
$$Y = -(-as^3W - bs^2W - csW - dW) \tag{7.17}$$

Splošna simulacijska shema, ki jo prikazuje slika 3.14, se spremeni v shemo, ki jo prikazuje slika 7.11.

□

Dobljene analogne simulacijske sheme pa še ne predstavljajo popolnega programa za analogni računalnik. Kot smo že omenili, moramo *problemske spremenljivke* prilagoditi analognemu računalniku oz. jih pretvoriti v t.i. *računalniške spremenljivke*. Postopek, ki to omogoči, imenujemo *amplitudno in časovno skaliranje*, dobljeno shemo pa *skalirano analogno simulacijsko shemo*.

Skalirana analogna simulacijska shema vsebuje adrese uporabljenih komponent, vsi izhodi komponent pa morajo biti opremljeni z normiranimi računalniškimi



Slika 7.11: Analogna simulacijska shema za realizacijo prenosne funkcije po delitveni metodi

spremenljivkami (to so imena problemskih spremenljivk, deljena z ocenjenimi maksimalnimi vrednostmi, kvocient je v oglatem oklepaju). Torej če med simulacijo določena problemska spremenljivka zavzame vrednost ocenjene maksimalne vrednosti, je ustrezna normirana računalniška spremenljivka v tistem trenutku enaka ena (oz. $10V$ pri $10V$ računalniku). V splošnem pa lahko iz znane ocenjene maksimalne vrednosti spremenljivke in trenutne vrednosti normirane računalniške spremenljivke določimo velikost problemske spremenljivke. Razen tako opremljene sheme pa program za analogni računalnik sestavlja tudi tabela nastavitve potenciometrov. Pregledna tabela naj vsebuje adrese, pomen, vrednosti potenciometrov ter ojačenja (uteži) vhodov, na katere so priključeni izhodi potenciometrov. S tem je vse pripravljeno za ustrezno povezavo komponent analognega računalnika (ročno ali avtomatsko) in za ustrezne nastavitve (npr. potenciometrov). Če smiselno izbiramo potrebne komponente, dobimo na programski plošči pregleden program, ki poveča ilustrativnost analogne simulacije.

7.4 Amplitudno in časovno skaliranje

Do sedaj se nismo ukvarjali s problemom velikosti simulacijskih spremenljivk in z njihovimi enotami. V praksi pa so žal območja problemskih spremenljivk zelo različna, npr. od velikostnega reda $10^{-10}m$, če simuliramo dogajanja v atomu,

do $10^{10}m$, če simuliramo gibanja planetov. Ni pa problematična le velikost spremenljivk, zelo različna je tudi njihova časovna dinamika, ki se lahko spreminja od mikrosekund, če simuliramo elektromagnetne pojave, do več tisoč let, če npr. simuliramo nekatere radioaktivne pojave.

Na drugi strani pa analogni računalnik deluje v omejenem napetostnem območju (običajno $\pm 10V$) in s časovno dinamiko, ki zavisi od uporabljenih električnih (in morebitnih mehanskih) komponent. Zato je potrebno vsak realni problem, ki ga želimo simulirati na analognem računalniku, poprej ustrezno amplitudno (veličinsko) in časovno skalirati. Enačb pa ni potrebno skalirati le pri simulaciji na analognem računalniku, ampak tudi pri realizaciji raznih algoritmov v posebnonamenski materialni opremi, če želimo doseči optimalno hitrost izračunavanja in ustrezno natančnost (npr. digitalni filtri, diskretni regulatorji,...). Pri digitalni simulaciji skaliranje ni potrebno, razen če velikosti spremenljivk in časovne konstante modela prekoračijo minimalna in maksimalna števila, ki jih lahko predstavimo v ustrezni aritmetiki. Amplitudno in časovno skaliranje je potrebno tudi v primeru, če uporabljamo celoštevilčno aritmetiko ali aritmetiko s fiksno decimalno vejico.

Amplitudno in časovno skaliranje je torej na analogno - hibridnih računalnikih potrebno zato, da prilagodimo področje problemskih spremenljivk področju računalniških spremenljivk. Vemo pa že tudi, da so ojačenja potenciometrov omejena na ± 1 in če uporabimo še vhode z višjimi ojačenji (največ 10 ali 100, odvisno od vrste računalnika), še vedno ne moremo realizirati konstant, ki so večje od 10 oz. 100. Zato lahko ustrezne vrednosti konstant upoštevamo le s skaliranjem enačb.

Problem pa prav tako nastane, če imajo med simulacijo izhodi ojačevalnikov zelo majhne napetosti. V tem primeru postane vpliv šuma na rezultate simulacije znaten. Problem predstavljajo tudi zelo majhne nastavitve potenciometrov. Najbolj natančni analogni računalniki omogočajo nastavitve potenciometrov z natančnostjo 10^{-4}, kar predstavlja relativni pogrešek 0.02%, če nastavimo potenciometer na 0.5 in relativni pogrešek 10% pri nastavitvi na 0.001. Nastavitev na 0.00005 pa sploh ni možna. Povsem podobni so problemi, če na digitalnem računalniku želimo uporabiti celoštevilčno aritmetiko. Rešitev za vso omenjeno problematiko je v amplitudnem in časovnem skaliranju.

7.4.1 Amplitudno skaliranje

Amplitudno skaliranje je postopek algebraične transformacije sistema diferencialnih enačb z namenom, da transformiramo problemske spremenljivke v računalniške. Poznamo nekaj različnih metod, opisali pa bomo le metodo *normiranih spremenljivk*, ki je po našem mnenju najbolj sistematična in konsistentna.

Osnovna ideja metode normiranih spremenljivk je v deljenju problemske spremenljivke x z njeno maksimalno absolutno vrednostjo x_{max}. Rezultat tega deljenja je normirana problemska spremenljivka $[\frac{x}{x_{max}}]$. Tudi moderni analogno - hibridni računalniki uporabljajo v smislu programiranja in spremljanja rezultatov normirane spremenljivke, ki jih dobimo z deljenjem izhodne napetosti ojačevalnika e z referenčno napetostjo e_{max}. Torej je normirana računalniška spremenljivka $[\frac{e}{e_{max}}]$.

Normirane problemske in računalniške spremenljivke so torej brez enot in v območju ± 1. Relacijo med problemsko in računalniško spremenljivko pa dobimo z izenačitvijo normirane problemske in normirane računalniške spremenljivke

$$[\frac{x}{x_{max}}] = [\frac{e}{e_{max}}] \tag{7.18}$$

Torej izračunamo problemsko spremenljivko iz izraza

$$x = e\frac{x_{max}}{e_{max}} = [\frac{e}{e_{max}}]x_{max} \tag{7.19}$$

Čeprav je dejanski skalirni faktor $\frac{x_{max}}{e_{max}}$, pa le tega ni potrebno uporabljati pri preračunavanju, saj vsa odčitavanja računalniških spremenljivk potekajo že v normirani obliki $[\frac{e}{e_{max}}]$ in je torej za izračun problemske spremenljivke potrebno normirano (odčitano) računalniško spremenljivko le pomnožiti z maksimalno vrednostjo problemske spremenljivke.

Dobra ocenitev maksimalnih vrednosti problemskih spremenljivk je osnovni pogoj uspešnega skaliranja. Če so ocenjene vrednosti manjše od dejanskih, potem nekateri ojačevalniki med simulacijo pridejo v nasičenje. Če pa so ocenjene vrednosti bistveno prevelike (npr. za faktor večji od deset), potem lahko računamo s slabšo natančnostjo simulacijskih rezultatov, saj v tem primeru šum ojačevalnikov postane nezanemarljiv.

Ocenitev maksimalnih vrednosti

Za ocenitev maksimalnih vrednosti problemskih spremenljivk poznamo več postopkov:

- Osnovna metoda temelji na dobrem poznavanju fizikalnega ozadja problema.
- Kadar je model opisan z diferencialno enačbo

$$a_0 x^{(n)} + a_1 x^{(n-1)} + \ldots + a_n x = f(t) \tag{7.20}$$

ali z ustreznim zapisom v prostoru stanj, obstajajo nekatere metode, po katerih lahko ocenimo maksimalne vrednosti, ne da bi predhodno rešili diferencialno enačbo.

Po *pravilu enakih koeficientov* (Jackson, 1960) izberemo maksimalne vrednosti spremenljivke $x(t)$ in njenih odvodov tako, da so vsi koeficienti v skalirani enačbi približno iste velikosti. Obstoja pa omejitev, da morajo tako določene maksimalne vrednosti tvoriti monotono naraščajoče ali monotono padajoče zaporedje. Če imamo avtonomni sistem 1. reda

$$a_0 \dot{x} + a_1 x = 0 \tag{7.21}$$

dobimo po amplitudnem skaliranju enačbo

$$a_0 \dot{x}_{max} [\frac{\dot{x}}{\dot{x}_{max}}] + a_1 x_{max} [\frac{x}{x_{max}}] = 0 \tag{7.22}$$

Maksimalni vrednosti moramo izbrati tako, da sta koeficienta $a_0 \dot{x}_{max}$ in $a_1 x_{max}$ približno enaka.

Pravilo povprečne lastne frekvence določi maksimalne vrednosti z upoštevanjem časovnih konstant in lastnih frekvenc sistema, ki ga opisuje enačba (7.20). Za avtonomni stabilni sistem prvega in drugega reda ($f(t) = 0$)

$$a_0 \dot{x} + a_1 x = 0 \tag{7.23}$$

oz.

$$a_0 \ddot{x} + a_1 \dot{x} + a_2 x = 0 \tag{7.24}$$

je možno enostavno pokazati, da maksimalne vrednosti spremenljivke x in njenih odvodov tvorijo geometrično zaporedje. Za sistem prvega reda (enačba (7.23)) je rešitev

$$x(t) = x(0) e^{-\frac{a_1}{a_0} t} = x_{max} e^{-\frac{a_1}{a_0} t} \tag{7.25}$$

Maksimalna vrednost spremenljivke $x(t)$ je torej kar enaka začetnemu pogoju $x(0)$. Maksimalna vrednost spremenljivke $\dot{x}(t)$ pa je

$$\dot{x}_{max} = \frac{a_1}{a_0} x_{max} \tag{7.26}$$

Za sistem drugega reda moramo pri ocenitvi maksimalnih vrednosti upoštevati najbolj neugodno možnost, t.j. nedušeni sistem ($a_1 = 0$). V tem primeru dobimo za spremenljivko $x(t)$ rešitev

$$x(t) = x_{max} \sin(\sqrt{\frac{a_2}{a_0}} t + \varphi) \tag{7.27}$$

oz. maksimalni vrednosti prvega in drugega odvoda spremenljivke $x(t)$

$$\dot{x}_{max} = \sqrt{\frac{a_2}{a_0}} x_{max} = \omega x_{max} \tag{7.28}$$

in

$$\ddot{x}_{max} = \sqrt{\frac{a_2}{a_0}} \dot{x}_{max} = \frac{a_2}{a_0} x_{max} = \omega^2 x_{max} \tag{7.29}$$

Za avtonomne in stabilne sisteme višjih redov vpeljemo povprečno lastno frekvenco (Carlson in ostali, 1967)

$$\bar{\omega} = \sqrt[n]{\frac{a_n}{a_0}} \tag{7.30}$$

Ta frekvenca predstavlja geometrično srednjo vrednost vseh korenov karakteristične enačbe sistema n-tega reda in določa razmerje maksimalnih vrednosti prvega odvoda in spremenljivke ali pa razmerje dveh zaporednih odvodov

$$\bar{\omega} = \frac{\dot{x}_{max}}{x_{max}} = \frac{\ddot{x}_{max}}{\dot{x}_{max}} = \cdots = \frac{x^{(n)}_{max}}{x^{(n-1)}_{max}} \tag{7.31}$$

Torej so ocene maksimalnih vrednosti odvodov

$$\dot{x}_{max} = \bar{\omega} x_{max} \tag{7.32}$$

$$\ddot{x}_{max} = \bar{\omega} \dot{x}_{max} = \bar{\omega}^2 x_{max} \tag{7.33}$$

$$\vdots$$

Pri obravnavani metodi torej dobimo le ustrezna razmerja med spremenljivko in odvodom oz. med višjimi odvodi. Zato je potrebno oceniti maksimalno vrednost spremenljivke ali kakšnega njenega odvoda. V

primeru, če ima avtonomni sistem le en nenični začetni pogoj, le ta predstavlja tudi maksimalno vrednost tiste spremenljivke oz. odvoda. Če pa je sistem vzbujan s konstantnim signalom $f(t) = c$, je ustaljena vrednost spremenljivke x enaka c/a_n. Če so vsi začetni pogoji enaki nič in če predpostavimo 100% prevzpon, potem lahko ocenimo maksimalno vrednost spremenljivke $x(t)$ z izrazom $x_{max} = 2c/a_n$. V tem primeru so ob upoštevanju pravila enakih koeficientov ostale ocenjene maksimalne vrednosti $x^{(i)}_{max} = \frac{c}{a_{n-i}}$ ali $x^{(i)}_{max} = \frac{2c}{a_{n-i}}$, če predpostavimo večje prevzpone.

Če je sistem vzbujan z vhodnimi signali in tudi z več začetnimi pogoji, moramo oceniti maksimalne vrednosti z ustreznim kombiniranjem (superponiranjem).

Z določenimi poenostavitvami in linearizacijami je možno obravnavano metodo uporabiti tudi pri nelinearnih sistemih.

- Pri statičnih komponentah (npr. sumator, množilnik, delilnik, ...) določimo običajno maksimalno vrednost izhodne spremenljivke ob upoštevanju situacij, ki povzročijo največjo možno vrednost izhoda (npr. pri sumatorju ali množilniku vhodni signali istočasno dosežejo maksimalne vrednosti, torej je maksimalna vrednost izhoda vsota maksimalnih vrednosti vhodov pri sumatorju oz. produkt pri množilniku, pri delilniku hkrati nastopi maksimalna vrednost deljenca in minimalna vrednost delitelja, torej je maksimalna vrednost izhoda kvocient maksimalne vrednosti deljenca in minimalne vrednosti delitelja.

- Pri modernih hibridnih sistemih lahko uporabimo digitalno simulacijo za ocenitev maksimalnih vrednosti.

- Maksimalne vrednosti lahko določimo (ocenimo) iz rezultatov simulacijskega teka, ki ga izvedemo s konservativnim skaliranjem, ko izberemo tako velike maksimalne vrednosti, da nobeden od ojačevalnikov gotovo ne pride v nasičenje.

Če se pri nekem problemu izkaže, da je slabo skaliran, ga je potrebno ponovno skalirati tako, da uporabimo maksimalne vrednosti, ki smo jih dobili iz simulacije pri slabem skaliranju.

Postopek amplitudnega skaliranja

Pri obravnavi se bomo omejili na linearne bloke (npr. sumator, invertor, integrator) ter na množilnik in delilnik. Postopek amplitudnega skaliranja lahko opišemo v dveh korakih:

1. Za vse bloke (komponente) analogne simulacijske sheme (razen za potenciometre) napišemo originalne neskalirane enačbe v obliki

$$izhod = -f(vhod) \tag{7.34}$$

   f predstavlja določen funkcijski simbol oz. blok.

2. Problemske spremenljivke zamenjamo z normiranimi računalniškimi spremenljivkami, vendar na tak način, da ostanejo enačbe veljavne. To naredimo tako, da problemske spremenljivke delimo in množimo z njihovimi maksimalnimi vrednostmi. Enačbe preuredimo v obliko, kjer so normirane spremenljivke, nastavitve potenciometrov in vhodna ojačenja eksplicitno razvidna. Končno dobimo skalirano enačbo v naslednji obliki:

$$[norm.\ izhod] = -f\{(nastavitev\ potenciometra) \cdot ojačenje \cdot [norm.\ vhod]\} \tag{7.35}$$

V primeru integratorja funkcijski simbol $\int\{\}dt + (začetni\ pogoj)$ običajno zamenjamo z operatorjem odvajanja $\frac{d}{dt}$ na drugi, t.j. levi strani enačbe, tako da dobimo izraz

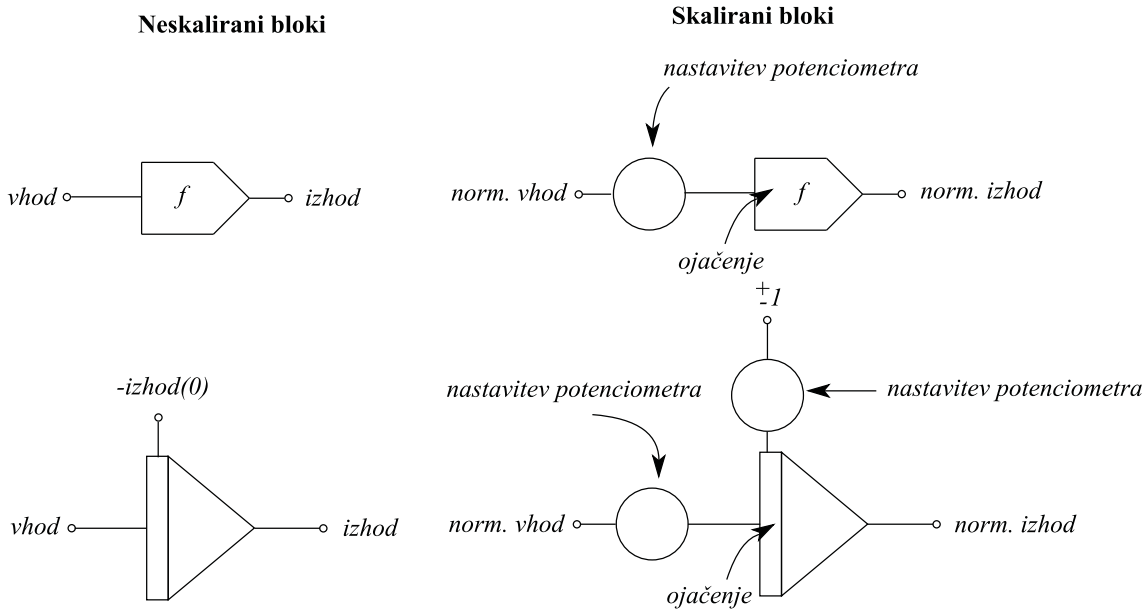
$$\frac{d}{dt}[norm.\ izhod] = -\{(nastavitev\ potenciometra) \cdot ojačenje \cdot [norm.\ vhod]\} \tag{7.36}$$

Začetni pogoj pa opišemo z ločeno enačbo

$$[norm.\ izhod(0)] = -\{(nastavitev\ potenciometra) \cdot [\pm 1]\} \tag{7.37}$$

kjer $[\pm 1]$ predstavlja normirano referenčno napetost. Torej integrator obrača predznak tudi začetnemu pogoju. Slika 7.12 prikazuje enačbam (7.34), (7.35), (7.36) in (7.37) ustrezne dele analogne simulacijske sheme.

Če se problemski parametri (koeficienti diferencialne enačbe) spreminjajo znotraj določenega območja, je smiselno, da ustrezne potenciometre opišemo kot spremenljivke. Ta postopek bo prikazan v primeru 7.4.



Slika 7.12: Skaliranje splošnega analognega bloka in integratorja

Pri opisanem postopku amplitudnega skaliranja smo predpostavili, da negativne minimalne vrednosti ležijo v enakem velikostnem področju, kot maksimalne vrednosti, tako da izkoristimo celotno področje. Če to ni res (npr. v primeru ekološkega modela žrtev in roparjev populacije ne morejo biti negativne), moramo normirane spremenljivke definirati kot odmike problemskih spremenljivk od njihovih nominalnih (delovnih) točk. To najlaže dosežemo tako, da preuredimo enačbe, da le te opisujejo odmike od delovne točke. Tak postopek je prikazan v primeru 7.5.

7.4.2 Časovno skaliranje

Pri amplitudnem skaliranju smo obravnavali le odvisne spremenljivke nekega problema. Enako pomembno pa je, da vzpostavimo relacijo med problemsko in računalniško neodvisno spremenljivko. Neodvisna spremenljivka na analogno - hibridnih računalnikih je seveda čas. Njegova dinamika mora biti v skladu z dinamičnimi zmožnostmi računalnika in perifernih naprav. Po drugi strani pa želi uporabnik čim hitreje priti do rezultatov, razen v primerih, ki zahtevajo simulacijo v realnem času (ob priključenih zunanjih napravah). V splošnem ni nujno, da je problemska neodvisna spremenljivka čas, če pa je, je običajno v drugem dinamičnem področju kot računalniška neodvisna spremenljivka. Nek realni pro-

ces namreč lahko poteka ure ali leta in ne bi bilo praktično tudi pri simulaciji čakati tako dolgo. Zato uvedemo časovno skaliranje in relacijo med problemsko in računalniško neodvisno spremenljivko opišemo z enačbo

$$\tau = nt \tag{7.38}$$

kjer je τ računalniška neodvisna spremenljivka (čas v sekundah, kadar analogni računalnik deluje pri nižji hitrosti), t je problemska neodvisna spremenljivka in n *skalirni faktor*. Če je problemska neodvisna spremenljivka čas (v sekundah!), potem je n brezdimenzijska veličina, sicer pa ima dimenzije sekunda/(enota problemske spremenljivke). V prvem primeru imamo tri možnosti

- $n > 1$
  potek rešitve na računalniku je počasnejši kot pri realnem problemu,
- $n < 1$
  potek rešitve na računalniku je hitrejši kot pri realnem problemu,
- $n = 1$
  izvaja se simulacija v realnem času.

Kemične, termične, nuklearne, farmakokinetične, socio-ekonomske idr. sisteme simuliramo na analognem računalniku hitreje kot v realnem času, električne sisteme običajno upočasnimo. Mehanske sisteme pa simuliramo blizu realnega časa (n je približno 1).

Kako pa časovno skaliranje vpliva na amplitudno skalirane enačbe? Analiza vseh komponent analognega računalnika pokaže, da je integrator edini časovno odvisni element. Zato je potrebno dodatno spremeniti le tiste amplitudno skalirane enačbe, ki opisujejo integratorje. Časovno skaliranje izvedemo tako, da zaradi veljavnosti enačbe (7.38) operator $\frac{d}{dt}$ na levi strani enačbe integratorja v enačbi (7.36) zamenjamo z operatorjem $n \cdot \frac{d}{d\tau}$

$$n\frac{d}{d\tau}[norm.\ izhod] = -\{(nastavitev\ potenciometra) \cdot ojačenje \cdot [norm.\ vhod]\} \tag{7.39}$$

Po deljenju s konstanto n dobimo končno obliko amplitudno in časovno skaliranega integratorja

$$\frac{d}{d\tau}[norm.\ izhod] = -\{(nova\ nast.\ potenciometra) \cdot ojačenje \cdot [norm.\ vhod]\} \tag{7.40}$$

Skalirni faktor n vpliva torej na vse nastavitve potenciometrov na desnih straneh enačb integratorjev. Nastavitve potenciometrov neskaliranih enačb je treba s skalirnim faktorjem n deliti ($nova\ nast. = nastavitev/n$). Enačbe začetnih pogojev integratorjev ostanejo nespremenjene.

Pri izbiri skalirnega faktorja n skušamo doseči nastavitve vseh potenciometrov (skupaj z ojačenji) pred integratorji v velikostnem razredu med 0.1 in 10 (če seveda ni kakšnih drugačnih zahtev, npr. po simulaciji v realnem času). To pa vedno ni možno. Če je ob pravilnem amplitudnem in časovnem skaliranju razmerje med največjim in najmanjšim potenciometrom večje od 1000, govorimo o t.i. togem (stiff) sistemu. Z drugimi besedami to pomeni, da imamo opravka s problemom z zelo različnimi časovnimi konstantami. Takega simulacijskega problema ne moremo odpraviti s skaliranjem.

Omenili pa smo že, da časovno skaliranje ni potrebno le zaradi narave problemske neodvisne spremenljivke, ampak tudi zaradi vrste naprave za prikaz rezultatov. Povsem različni velikostni razred neodvisne računalniške spremenljivke namreč zahteva prikaz rezultatov na koordinatnem risalniku kot npr. na osciloskopu. Vendar se to potrebno preskaliranje na analogno - hibridnih računalnikih izvede avtomatsko s pritiskom na gumb ali preko tipkovnice, torej uporabniku ni treba preoblikovati enačb in spremeniti nastavitev v programu.

Primer 7.4 Skaliranje sistema avtomobilskega vzmetenja

Namen tega primera je, da ustrezno skaliramo enačbo, ki opisujejo model avtomobilskega vzmetenja

$$\dddot{y} + 70\ddot{y} + 300\dot{y} + 1000y = 0 \quad y(0) = -0.05 \tag{7.41}$$

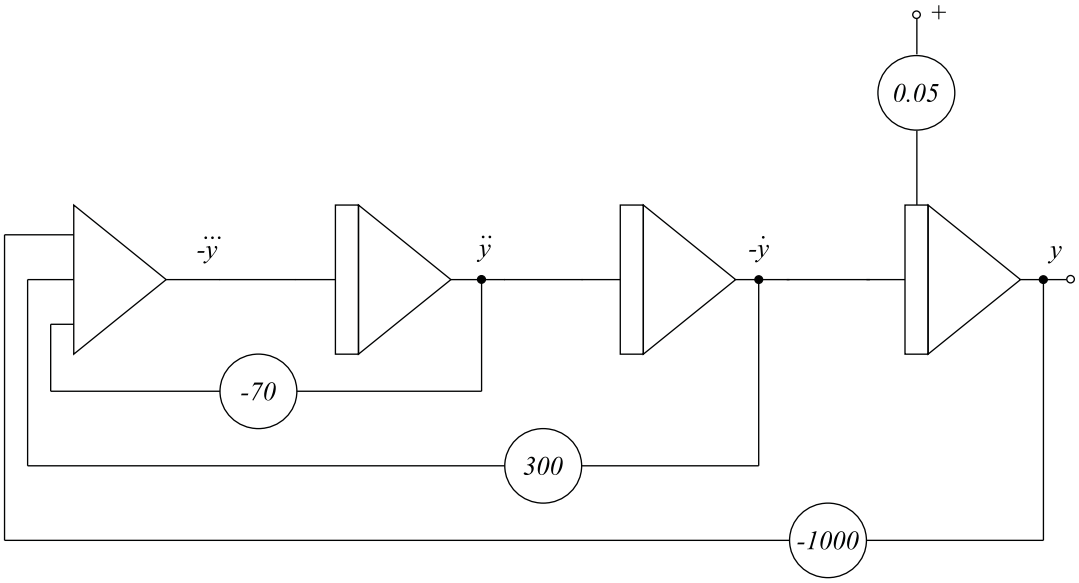
Preden izvedemo postopek skaliranja, narišemo analogno simulacijsko shemo. Zato izrazimo negativno vrednost najvišjega odvoda, kajti odločili smo se, da se bodo na analognem računalniku direktno generirale spremenljivke $-\dddot{y}, \ddot{y}, -\dot{y}$ in y.

$$-\dddot{y} = -(-70\ddot{y} + 300(-\dot{y}) - 1000y) \tag{7.42}$$

Analogno simulacijsko shemo prikazuje slika 7.13.

Ker je v postopku skaliranja potrebno skalirati vse bloke, napišimo dodatno neskalirane enačbe integratorjev. Namesto integralskega operatorja na desni strani enačbe običajno pišemo operator odvajanja na levi strani

$$\frac{d}{dt}(\ddot{y}) = -(-\dddot{y}) \tag{7.43}$$



Slika 7.13: Neskalirana analogna simulacijska shema za model avtomobilskega vzmetenja

$$\frac{d}{dt}(-\dot{y}) = -(\ddot{y}) \tag{7.44}$$

$$\frac{d}{dt}(y) = -(-\dot{y}) \qquad y(0) = -(0.05) \tag{7.45}$$

V postopku skaliranja zahteva prvi korak ocenitev maksimalnih vrednosti. Ker pričakujemo dušen prehodni pojav, je seveda ocenjena maksimalna vrednost spremenljivke $y(t)$ enaka absolutni vrednosti začetnega pogoja, t.j. 0.05. Sedaj enačbo (7.41) preuredimo v obliko

$$\dddot{y}_{max}\left[\frac{\dddot{y}}{\dddot{y}_{max}}\right] + 70\ddot{y}_{max}\left[\frac{\ddot{y}}{\ddot{y}_{max}}\right] + 300\dot{y}_{max}\left[\frac{\dot{y}}{\dot{y}_{max}}\right] + 50\left[\frac{y}{0.05}\right] = 0 \tag{7.46}$$

Ker je koeficient pred izrazom $\left[\frac{y}{0.05}\right]$ enak 50, določimo po pravilu enakih koeficientov maksimalne vrednosti $\dddot{y}_{max}, \ddot{y}_{max}$ in $\dot{y}_{max}$ tako, da so tudi ostali koeficieni enaki 50. Ustrezne ocene so zbrane v prvi koloni tabele 7.1.

Z uporabo pravila povprečne lastne frekvence je le ta

$$\bar{\omega} = \sqrt[3]{1000} = 10 \tag{7.47}$$

Ustrezne ocenjene maksimalne vrednosti so zbrane v drugi koloni tabele 7.1.

Tabela 7.1: Ocene maksimalnih vrednosti sistema avtomobilskega vzmetenja

	Ocenjene maksimalne vrednosti			
	Pravilo enakih koeficientov	*Pravilo povprečne lastne frekvence*	*Prave vrednosti*	*Vrednosti za reskaliranje*
y_{max}	0.05	0.05	0.05	0.05
$\dot{y}_{max}$	0.1666	0.50	0.101	0.125
$\ddot{y}_{max}$	0.71	5.0	0.49	0.50
$\dddot{y}_{max}$	50.0	50.0	50.0	50.0

Za skaliranje uporabimo bolj konservativne ocene (druga kolona v tabeli 7.1). Preden pa začnemo s preoblikovanjem enačb, pa napišemo še t.i. tabelo skaliranja, ki vsebuje problemske spremenljivke, ki nastopajo v neskalirani simulacijski shemi, konstante skaliranja (to so običajno navzgor na okrogle vrednosti zaokrožene maksimalne vrednosti) in normirane računalniške spremenljivke v oglatih oklepajih. Ustrezne vrednosti prikazuje tabela 7.2.

Tabela 7.2: Tabela skaliranja

Spremenljivka	*Konstanta skaliranja*	*Normirana računalniška spremenljivka*
y	0.05	$[\frac{y}{0.05}]$
$-\dot{y}$	0.5	$[-\frac{\dot{y}}{0.5}]$
$\ddot{y}$	5	$[\frac{\ddot{y}}{5}]$
$-\dddot{y}$	50	$[-\frac{\dddot{y}}{50}]$

Skalirati je potrebno vse bloke. Za sumator dobimo ob ustrezni preureditvi enačbe 7.42 enačbo

$$50\left[-\frac{\dddot{y}}{50}\right] = -\left\{-70\cdot 5\cdot\left[\frac{\ddot{y}}{5}\right] + 300\cdot 0.5\cdot\left[-\frac{\dot{y}}{0.5}\right] - 1000\cdot 0.05\cdot\left[\frac{y}{0.05}\right]\right\} \quad (7.48)$$

ki jo preuredimo v končno skalirano obliko

$$\left[-\frac{\dddot{y}}{50}\right] = -\left\{(-0.7)\cdot 10\cdot\left[\frac{\ddot{y}}{5}\right] + (0.3)\cdot 10\cdot\left[-\frac{\dot{y}}{0.5}\right] + (-1.0)\cdot 1\left[\frac{y}{0.05}\right]\right\} \quad (7.49)$$

Ob upoštevanju skalirnega faktorja $\tau = nt$ so skalirane enačbe za vse tri integratorje

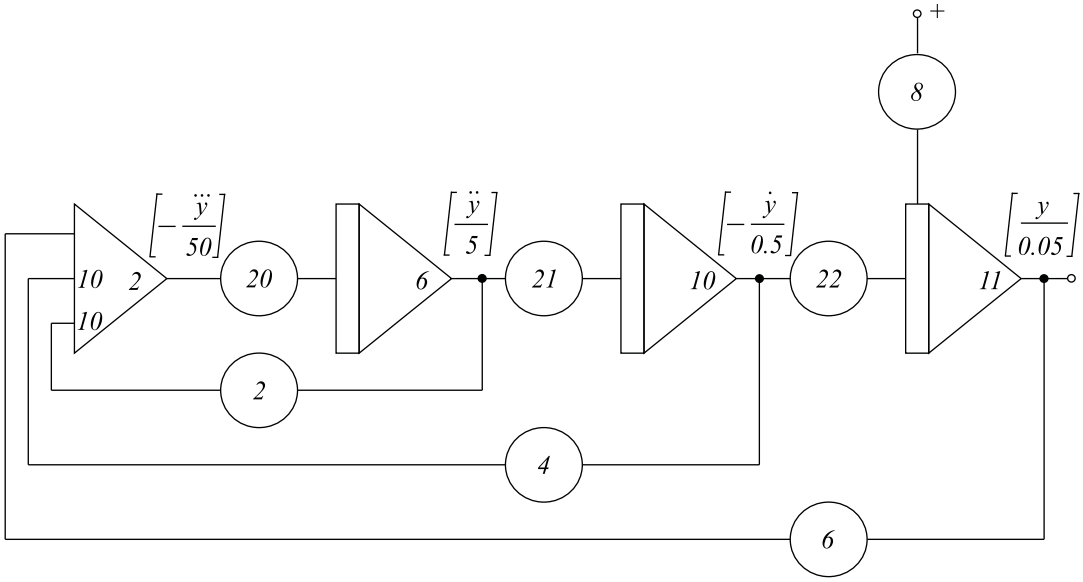
$$\frac{d}{d\tau}\left[\frac{\ddot{y}}{5}\right] = -\left\{\frac{10}{n}\left[-\frac{\ddot{y}}{50}\right]\right\} \tag{7.50}$$

$$\frac{d}{d\tau}\left[-\frac{\dot{y}}{0.5}\right] = -\left\{\frac{10}{n}\left[\frac{\ddot{y}}{5}\right]\right\} \tag{7.51}$$

$$\frac{d}{d\tau}\left[\frac{y}{0.05}\right] = -\left\{\frac{10}{n}\left[-\frac{\dot{y}}{0.5}\right]\right\} \quad \left[\frac{y(0)}{0.05}\right] = -\{(1.0)\cdot[+1]\} \tag{7.52}$$

Če izberemo $n = 10$, postanejo vse nastavitve potenciometrov pred integratorji enake 1.

Skalirano analogno simulacijsko shemo prikazuje slika 7.14. Adrese potenciometrov, pomeni, nastavitve in ojačenja pa so zbrana v tabeli 7.3.



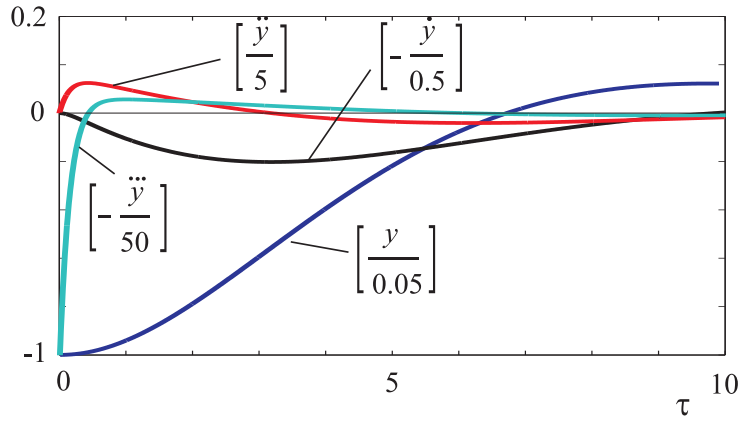
Slika 7.14: Skalirana analogna simulacijska shema za model avtomobilskega vzmetenja

Rezultati simulacije so prikazani na sliki 7.15 in iz nje (oz. iz numerično dobljenih vrednosti, ki jih lahko odčitamo med simulacijo z digitalnim voltmetrom) smo dobili prave maksimalne vrednosti, ki so prikazane v tretji koloni tabele 7.1.

Da dosežemo natančnejšo simulacijo, reskaliramo enačbe z maksimalnimi vrednostmi, ki so vpisane v četri koloni tabele 7.1. V tem primeru je pri skaliranju

Tabela 7.3: Nastavitve potenciometrov in ojačenja

Adresa	*Pomen*	*Vrednost*	*Ojačenje*
2	$-a \cdot \frac{\dddot{y}_{max}}{\ddot{y}_{max}} = -\frac{k_1+k_2}{f} \cdot \frac{\dddot{y}_{max}}{\ddot{y}_{max}}$	-0.7	10
4	$b \cdot \frac{y_{max}}{\ddot{y}_{max}} = \frac{k_2}{m} \cdot \frac{\dot{y}_{max}}{\ddot{y}_{max}}$	0.3	10
6	$-c \cdot \frac{y_{max}}{\ddot{y}_{max}} = -\frac{k_1 k_2}{mf} \cdot \frac{y_{max}}{\ddot{y}_{max}}$	-1.0	1
8	$-\frac{y_0}{y_{max}}$	1.0	/
20	$\frac{\dddot{y}_{max}}{n\ddot{y}_{max}}$	1.0	1
21	$\frac{\ddot{y}_{max}}{n\dot{y}_{max}}$	1.0	1
22	$\frac{\dot{y}_{max}}{ny_{max}}$	1.0	1



Slika 7.15: Rezultati analogne simulacije avtomobilskega vzmetenja

potrebno upoštevati skalirno tabelo 7.4.

Če predvidevamo, da bomo med različnimi simulacijskimi teki spreminjali parametre matematičnega modela ($a = 70, b = 300, c = 1000$), je smiselno namesto ustreznih konstant potenciometrov v le te uvesti ustrezne spremenljivke a, b in c. Tako dobimo ob upoštevanju skalirnega faktorja $n = 10$ naslednje skalirane enačbe

$$\begin{aligned} \left[-\frac{\dddot{y}}{50}\right] &= -\left\{(-\frac{a}{100}) \cdot \left[\frac{\ddot{y}}{0.5}\right] + (\frac{b}{400}) \cdot \left[-\frac{\dot{y}}{0.125}\right] + (-\frac{c}{1000}) \cdot \left[\frac{y}{0.05}\right]\right\} \\ \frac{d}{d\tau}\left[\frac{\ddot{y}}{0.5}\right] &= -\left\{(1.0) \cdot 10 \cdot \left[-\frac{\ddot{y}}{50}\right]\right\} \end{aligned} \tag{7.53}$$

Tabela 7.4: Tabela skaliranja

Spremenljivka	*Konstanta skaliranja*	*Normirana računalniška spremenljivka*
y	0.05	$[\frac{y}{0.05}]$
$-\dot{y}$	0.125	$[-\frac{\dot{y}}{0.125}]$
$\ddot{y}$	0.50	$[\frac{\ddot{y}}{0.50}]$
$-\dddot{y}$	50	$[-\frac{\dddot{y}}{50}]$

$$\begin{aligned} \frac{d}{d\tau}\left[-\frac{\dot{y}}{0.125}\right] &= -\left\{(0.4)\cdot 1\cdot\left[\frac{\ddot{y}}{0.5}\right]\right\} \\ \frac{d}{d\tau}\left[\frac{y}{0.05}\right] &= -\left\{(0.25)\cdot 1\cdot\left[-\frac{\dot{y}}{0.125}\right]\right\} \quad \left[\frac{y(0)}{0.05}\right] = -\{(1.0)\cdot[+1]\} \end{aligned}$$

Izbira skalirnega faktorja $n = 10$ seveda ni edina možnost. Izbrali smo jo zaradi lažjega računanja. Tudi izbira $n = 25$ bi omogočila, da bi bili vsi koeficienti v enačbah integratorjev v želenem področju. □

Primer 7.5 Skaliranje ekološkega modela žrtev in roparjev

V tem primeru bomo pokazali postopek ocenjevanja maksimalnih vrednosti in skaliranje nelinearnega sistema.

Ekološki model žrtev in roparjev opisujeta nelinearni diferencialni enačbi

$$\begin{aligned} \dot{x}_1 &= 5x_1 - 0.05x_1x_2 \qquad x_1(0) = 520 \\ \dot{x}_2 &= 0.0004x_1x_2 - 0.2x_2 \quad x_2(0) = 85 \end{aligned} \tag{7.54}$$

Rezultati simulacije na sliki 4.5 v primeru 4.1 kažejo, da ima sistem v ustaljenem stanju limitni cikel, t.j. nelinearno periodično vedenje okoli nekih srednjih delovnih vrednosti. Torej v tem primeru ni smiselno predpostaviti, da so minimalne vrednosti spremenljivk enake negativnim ocenjenim maksimalnim vrednostim. Ker je postopek skaliranja ob upoštevanju minimalnih in maksimalnih vrednosti kompliciran, bomo raje transformirali spremenljivke tako, da bo novo dobljeni model opisoval razmere okoli delovne točke. Smiselne vrednosti delovne točke

(singularne točke ali točke kvazi ustaljenega stanja), ki ju označimo z $(\bar{x}_1, \bar{x}_2)$, dobimo iz enačb (7.54) ob upoštevanju

$$\dot{x}_1 = \dot{x}_2 = 0 \tag{7.55}$$

kar da ustrezni vrednosti

$$\bar{x}_1 = 500 \qquad \bar{x}_2 = 100 \tag{7.56}$$

Definiramo novi spremenljivki

$$\tilde{x}_1 = x_1 - \bar{x}_1 = x_1 - 500 \tag{7.57}$$
$$\tilde{x}_2 = x_2 - \bar{x}_2 = x_2 - 100 \tag{7.58}$$

Sistem enačb (7.54) se s tem transformira v obliko

$$\begin{aligned} \dot{\tilde{x}}_1 &= -0.05\tilde{x}_1\tilde{x}_2 - 25\tilde{x}_2 \qquad \tilde{x}_1(0) = 20 \\ \dot{\tilde{x}}_2 &= 0.0005\tilde{x}_1\tilde{x}_2 + 0.04\tilde{x}_1 \qquad \tilde{x}_2(0) = -15 \end{aligned} \tag{7.59}$$

Izberimo spremenljivke $(-\tilde{x}_1)$, $\tilde{x}_2$, $\dot{\tilde{x}}_1$,$(-\dot{\tilde{x}}_2)$ in $\tilde{x}_{12}$ kot izhodne spremenljivke blokov v simulacijski shemi. Pri tem smo torej uvedli novo spremenljivko $\tilde{x}_{12} = \tilde{x}_1 \cdot \tilde{x}_2$, ki jo bomo dobili na izhodu množilnika. Na ta način lahko zapišemo naslednje enačbe v neskalirani obliki:

$$\begin{aligned} \tilde{x}_{12} &= -\{(-\tilde{x}_1) \cdot \tilde{x}_2\} \\ \dot{\tilde{x}}_1 &= -\{0.05\tilde{x}_{12} + 25\tilde{x}_2\} \\ (-\dot{\tilde{x}}_2) &= -\{0.0005\tilde{x}_{12} + 0.04\tilde{x}_1\} \\ \frac{d}{dt}(-\tilde{x}_1) &= -\{\tilde{x}_1\} \qquad -\tilde{x}_1(0) = -\{20\} \\ \frac{d}{dt}(\tilde{x}_2) &= -\{(-\dot{\tilde{x}}_2)\} \qquad \tilde{x}_2(0) = -\{15\} \end{aligned} \tag{7.60}$$

V naslednjem koraku je potrebno oceniti maksimalne vrednosti. Ker že vemo, da poteki nihajo okoli delovnih točk in ker hkrati spremenljivke v originalnem koordinatnem sistemu ne morejo biti negativne, je smiselno za maksimalni vrednosti izbrati $\tilde{x}_{1max} = 500$ in $\tilde{x}_{2max} = 100$.

Maksimalni vrednosti za spremenljivki $\dot{\tilde{x}}_1$ in $\dot{\tilde{x}}_2$ bomo ocenili z obema obravnavanima praviloma. S pravilom enakih koeficientov dosežemo približno enake koeficiente skaliranih enačb z vrednostima $\dot{\tilde{x}}_{1max} = 2500$ in $\dot{\tilde{x}}_{2max} = 20$. To

enostavno pokažemo, če vstavimo maksimalni vrednosti spremenljivk $\tilde{x}_1$ in $\tilde{x}_2$ v enačbi (7.59).

Če pa hočemo uporabiti pravilo povprečne lastne frekvence, moramo sistem nelinearnih enačb najprej linearizirati okoli delovne točke ($\tilde{x}_1 = \tilde{x}_2 = 0$). Linearizacijo izvedemo tako, da zanemarimo prva člena na desni strani enačb (7.59). Če nato obe enačbi preuredimo tako, da v prvi nastopa le spremenljivka $\tilde{x}_1$, v drugi pa le spremenljivka $\tilde{x}_2$, dobimo dve ločeni enačbi drugega reda

$$\begin{aligned} \ddot{\tilde{x}}_1 + \tilde{x}_1 &= 0 \\ \ddot{\tilde{x}}_2 + \tilde{x}_2 &= 0 \end{aligned} \tag{7.61}$$

Lastna frekvenca je torej $\bar{\omega} = 1$. Zato so ocenjene vrednosti spremenljivk $\dot{\tilde{x}}_1$ in $\dot{\tilde{x}}_2$ enake $\dot{\tilde{x}}_{1max} = \tilde{x}_{1max} = 500$ in $\dot{\tilde{x}}_{2max} = \tilde{x}_{2max} = 100$. V postopku skaliranja uporabimo iz obeh postopkov bolj konservativni vrednosti ($\dot{\tilde{x}}_{1max} = 2500$ in $\dot{\tilde{x}}_{2max} = 100$). Za ocenitev maksimalne vrednosti izhoda množilnika $\tilde{x}_{12}$ pa bomo predpostavili najbolj konservativno vrednost, t.j. produkt maksimalnih vrednosti obeh spremenljivk ($\tilde{x}_{12max} = \tilde{x}_{1max}\tilde{x}_{2max}$). Skalirane enačbe imajo obliko

$$\begin{aligned} \left[\frac{\tilde{x}_{12}}{50000}\right] &= -\left\{\left[-\frac{\tilde{x}_1}{500}\right] \cdot \left[\frac{\tilde{x}_2}{100}\right]\right\} \\ \left[\frac{\dot{\tilde{x}}_1}{2500}\right] &= -\left\{\left[\frac{\tilde{x}_{12}}{50000}\right] + \left[\frac{\tilde{x}_2}{100}\right]\right\} \\ \left[-\frac{\dot{\tilde{x}}_2}{100}\right] &= -\left\{(0.2) \cdot \left[\frac{\tilde{x}_{12}}{50000}\right] + (0.2) \cdot \left[-\frac{\tilde{x}_1}{500}\right]\right\} \\ \frac{d}{dt}\left[-\frac{\tilde{x}_1}{500}\right] &= -\left\{(0.5) \cdot 10 \cdot \left[\frac{\dot{\tilde{x}}_1}{2500}\right]\right\} \quad \left[-\frac{\tilde{x}_1(0)}{500}\right] = -\{(0.04) \cdot [+1]\} \\ \frac{d}{dt}\left[\frac{\tilde{x}_2}{100}\right] &= -\left\{\left[-\frac{\dot{\tilde{x}}_2}{100}\right]\right\} \quad \left[\frac{\tilde{x}_2(0)}{100}\right] = -\{(0.15) \cdot [+1]\} \end{aligned} \tag{7.62}$$

Ker so vsi koeficienti v območju med 0.1 in 10, problema ni potrebno časovno skalirati. Skalirni faktor n je torej 1, kar pomeni da ena enota računalniškega časa (sekunda pri nižji hitrosti analognega računalnika) predstavlja eno enoto problemske neodvisne spremenljivke, to pa je 1 leto. Skalirni faktor n torej v tem primeru ni brezdimenzijski faktor ampak ima dimenzijo $[\frac{s}{leto}]$.

Simulacija je pokazala, da so dejanske maksimalne vrednosti $\tilde{x}_1 = 507, \tilde{x}_2 = 16.7, \dot{\tilde{x}}_1 = 520$ in $\dot{\tilde{x}}_2 = 20.5$. Spremenljivka $\tilde{x}_1$ je torej za malenkost prekoračila maksimalno ocenjeno vrednost, vendar to operacijski ojačevalniki na analognih računalnikih običajno dopuščajo (v našem primeru gre za napetost 10.14V na 10V računalniku). □

Opisan postopek skaliranja lahko uporabljamo tudi pri načrtovanju posebnonamenskih analognih vezij, kar bo prikazal naslednji primer.

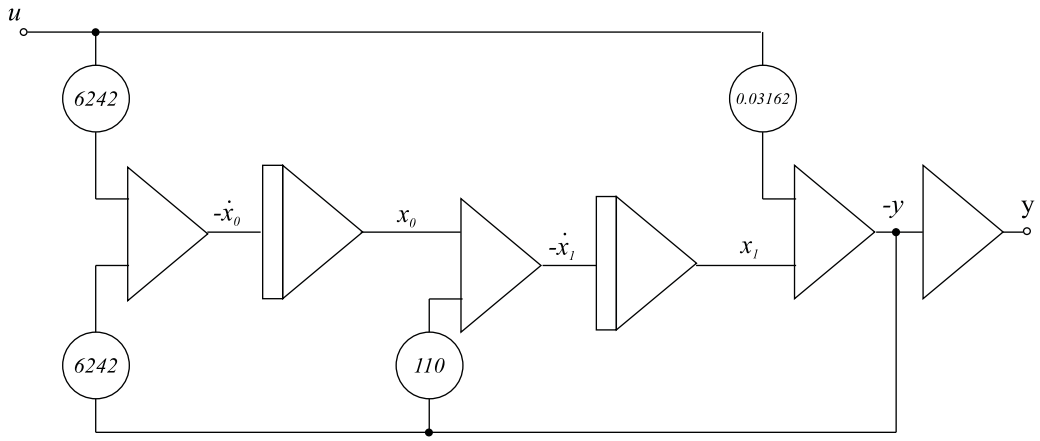
Primer 7.6 Uporaba skaliranja pri načrtovanju analognega filtra

Čeprav se danes vedno bolj uporabljajo digitalno realizirani filtri, so na nekaterih mestih še vedno potrebni analogni filtri. Na vhodih analogno - digitalnih pretvornikov taki filtri preprečijo prekrivanje pasov v frekvenčni karakteristiki vzorčenega signala, do česar pride, če frekvenca vzorčenja ni vsaj dvakrat višja od maksimalne frekvence, ki nastopa v signalu.

Chebyshev filter drugega reda s pasovno propustno karakteristiko, z mejno frekvenco 50 Hz in z dušenjem vsaj 30dB v zapornem delu, ima prenosno funkcijo

$$G(s) = \frac{0.03162s^2 + 6242}{s^2 + 110s + 6242} \tag{7.63}$$

Analogno simulacijsko shemo v vgnezdeni obliki prikazuje slika 7.16.



Slika 7.16: Analogna simulacijska shema Chebyshev-ega filtra

Vrednosti koeficientov v sliki 7.16 kažejo, da je potrebno uporabiti skaliranje.

Najprej je potrebno oceniti maksimalne vrednosti. Običajno ne želimo ojačenja ali dušenja v propustnem delu, zato lahko predpostavimo enaki maksimalni vrednosti za spremenljivki u in y. Brez izgube na splošnosti lahko izberemo ti vrednosti ena (kar npr. pomeni $\pm 10V$ vhodno in izhodno področje).

$$u_{max} = 1 \qquad\qquad y_{max} = 1 \tag{7.64}$$

Ob tem pa se moramo zavedati, da lahko pride do določenih problemov v prehodnem pojavu. Če bi se želeli izogniti temu problemu, bi lahko predpostavili 100% možni prevzpon, kar bi zahtevalo dvakrat večjo maksimalno vrednost za spremenljivko y.

Maksimalne vrednosti preostalih spremenljivk iz slike 7.16 bomo ocenili z dvema metodama:

1. V najneugodnejšem primeru, ko u_{max} in y_{max} nastopita hkrati, je ocenjena maksimalna vrednost spremenljivke $\dot{x}_0$

$$\dot{x}_{0max} =\mid 6242 \cdot u_{max} \mid + \mid 6242 \cdot y_{max} \mid= 12484 \tag{7.65}$$

   kar je razvidno iz slike 7.16. Ker je lastna frekvenca $\bar{\omega} = \sqrt{6242} = 79$, je ocenjena maksimalna vrednost spremenljivke x_0

$$x_{0max} = \frac{\dot{x}_{0max}}{\bar{\omega}} = 158 \tag{7.66}$$

   Z enakim postopkom lahko nadaljujemo po shemi 7.16 z leve proti desni in ocenimo maksimalni vrednosti $\dot{x}_{1max}$ in x_{1max}. Toda izkaže se, da je tako ocenjevanje preveč črnogledo. Zato raje ocenimo ustrezne maksimalne vrednosti s postopkom iz desnega dela sheme proti levi

$$x_{1max} =\mid y_{max} \mid + \mid 0.03162 \cdot u_{max} \mid= 1.03162 \tag{7.67}$$

$$\dot{x}_{1max} = \bar{\omega} x_{1max} = 81.5 \tag{7.68}$$

   Vse ocenjene maksimalne vrednosti so zbrane v drugi koloni tabele 7.5.

2. Če je pomembno le delovanje filtra v ustaljenem stanju pri sinusnem vhodnem signalu, lahko uporabimo *pravilo upoštevanja frekvenčne karakteristike* (Saucedo,Schirring, 1986, Terrel, 1988). Maksimalne vrednosti ocenimo tako, da izračunamo prenosne funkcije med vhodnim signalom u in vsemi ostalimi spremenljivkami iz slike 7.16. Iz prenosnih funkcij $G_i(s)$ izračunamo ustrezne absolutne vrednosti frekvenčnih karakteristik $\mid G_i(j\omega) \mid$. Maksimumi teh karakteristik določajo maksimalne možne amplitude sinusnih signalov. Tako dobljene vrednosti so zbrane v tretji koloni tabele 7.5.

Podatki iz tabele 7.5 kažejo, da so ocene, ki smo jih dobili s pravilom povprečne lastne frekvence, bolj črnoglede kot tiste, ki smo jih dobili s pravilom upoštevanja frekvenčne karakteristike. In ker je pri filtrih običajno pomembno le delovanje

Tabela 7.5: Maksimalne vrednosti pri Chebyshev-em filtru

Spremenljivka	*Pravilo povprečne lastne frekvence*	*Pravilo upoštevanja frekvenčne karakteristike*
u	1.00	1.00
$\dot{x}_0$	12484	7870
x_0	158.0	114.0
$\dot{x}_1$	81.5	55.00
x_1	1.03	0.9684
y	1.00	1.00

v ustaljenem stanju, skaliramo enačbe z upoštevanjem tretje kolone v tabeli 7.5. Tako dobimo izraze

$$\left[-\frac{\dot{x}_0}{7870}\right] = -\left\{(0.7931)\cdot\left[\frac{u}{1}\right] + (0.7931)\cdot\left[-\frac{y}{1}\right]\right\} \tag{7.69}$$

$$\frac{d}{dt}\left[\frac{x_0}{114}\right] = -\left\{(0.6904)\cdot 100\cdot\left[-\frac{\dot{x}_0}{7870}\right]\right\} \tag{7.70}$$

$$\left[-\frac{\dot{x}_1}{55}\right] = -\left\{(0.2073)\cdot 10\cdot\left[\frac{x_0}{114}\right] + (0.2)\cdot 10\cdot\left[-\frac{y}{1}\right]\right\} \tag{7.71}$$

$$\frac{d}{dt}\left[\frac{x_1}{0.9684}\right] = -\left\{(0.5679)\cdot 100\cdot\left[-\frac{\dot{x}_1}{55}\right]\right\} \tag{7.72}$$

$$\left[-\frac{y}{1}\right] = -\left\{(0.03162)\cdot\left[\frac{u}{1}\right] + (0.9684)\cdot\left[\frac{x_1}{0.9684}\right]\right\} \tag{7.73}$$

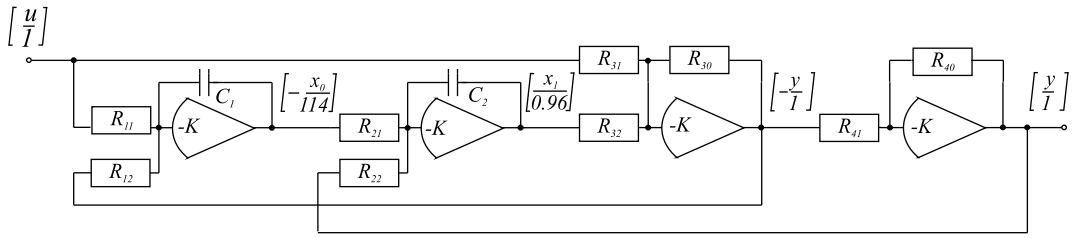
Filter mora seveda delovati v realnem času, zato ne smemo uporabiti časovnega skaliranja. Lahko pa uporabimo časovno skaliranje, če želimo filter frekvenčno preskalirati. Z $n = 30$ bi postal filter 30 krat počasnejši, kar pa v zvezi s frekvenčno karakteristiko pomeni, da smo mejno frekvenco zmanjšali za enak faktor (iz 50 Hz na 1.66 Hz).

Pri realizaciji vezja moramo upoštevati tudi ekonomske faktorje in filter realizirati s čim manj komponentami. Zato dva sumatorja in dva integratorja zamenjamo z dvema sumacijskima integratorjema

$$\frac{d}{dt}\left[-\frac{x_0}{114}\right] = -\left\{54.75\cdot\left[\frac{u}{1}\right] + 54.75\cdot\left[-\frac{y}{1}\right]\right\} \tag{7.74}$$

$$\frac{d}{dt}\left[\frac{x_1}{0.9684}\right] = -\left\{117.7\cdot\left[-\frac{x_0}{114}\right] + 113.6\cdot\left[\frac{y}{1}\right]\right\} \tag{7.75}$$

Realizacijo analognega filtra prikazuje slika 7.17. Koeficienti, ki nastopajo v skaliranih enačbah (v primeru simulacije na analognem računalniku bi bili to potenciometri in ojačenja analognih komponent) so realizirani z upori in kondenzatorji. Njihove vrednosti prikazuje tabela 7.6.



Slika 7.17: Vezje za realizacijo Chebyshev-ega filtra

Tabela 7.6: Vrednosti uporov in kondenzatorjev

Element	*Sumacijski integrator* 1	*Sumacijski integrator* 2	*Sumator*	*Invertor*
povratnozančni element	$C_1 = 1\mu F$	$C_2 = 1\mu F$	$R_{30} = 1k\Omega$	$R_{40} = 10k\Omega$
koeficient prvega vhoda	54.75	117.7	0.03162	1
upor prvega vhoda	$R_{11} = 18.26k\Omega$	$R_{21} = 8.4966k\Omega$	$R_{31} = 31.63k\Omega$	$R_{41} = 10k\Omega$
koeficient drugega vhoda	54.75	113.6	0.9684	–
upor drugega vhoda	$R_{12} = 18.26k\Omega$	$R_{22} = 8.803k\Omega$	$R_{32} = 1.033k\Omega$	–

□

7.5 Statični test

Amplitudno in časovno skaliranje vpliva na točnost simulacijskih rezultatov. Zanesljivost rezultatov simulacije pa je seveda odvisna tudi od tega, ali smo vse potrebne postopke izvedli brez napake. Predpostavimo, da v samem matematičnem modelu ni napak, kajti ne bomo proučevali napak modeliranja

ampak le napake simulacije. Le te so lahko zelo raznovrstne. Največkrat naredimo napako v skaliranju in v povezovanju komponent analognega računalnika.

Statični test je sistematični postopek za ugotavljanje raznih vrst napak. Čeprav ideja izvira iz analogno - hibridne simulacije, pa je uporabnost bolj splošna in jo lahko uporabljamo tudi v digitalni simulaciji.

Pri statičnem testu predpostavimo neke poljubne vendar primerne vrednosti na izhodih integratorjev. Iz teh podatkov iz originalnih enačb izračunamo izhode vseh komponent (blokov) simulacijskega modela in vse odvode, t.j. vhode v integratorje. Enake izračune naredimo tudi s pomočjo skalirane analogne simulacijske sheme oz. s pomočjo skaliranih enačb. Ustrezne izhode pa izmerimo tudi na analognem računalniku. Vsako neujemanje pomeni napako.

Sistematični postopek je naslednji:

1. Izberemo poljubne vrednosti vseh izhodov integratorjev. Vendar naj pri tem noben ojačevalnik ne pride v nasičenje (statično gledano), izogibati pa se moramo tudi pretirano majhnim signalom v analogni shemi.

2. Izračunamo izhode vseh blokov in vse odvode (vhode v integratorje) iz originalnih enačb.

3. Iz skalirane analogne simulacijske sheme ob upoštevanju nastavitev potenciometrov in pod točko 1 izbranih začetnih vrednosti izhodov integratorjev izračunamo vse izhode blokov in odvode.

4. Primerjamo rezultate, ki smo jih dobili pod točko 2 in 3. Kakršnokoli neujemanje pomeni napako v programiranju ali skaliranju. Mesto napake odkrijemo z izračuni v simulacijski shemi v obrnjenem vrstnem redu.

5. Izmerimo izhode vseh blokov in vse odvode, t.j. vhode v integratorje.

6. Primerjamo meritve z izračuni. Vsako neujemanje pomeni, da smo naredili napako v povezovanju ali pa kaže na napako v delovanju komponente analognega računalnika. Napako odkrijemo tako, da kontroliramo povezave oz. delovanje komponent v obratni smeri, kot smo računali enačbe.

Princip statičnega testa je možno uporabiti tudi pri digitalni simulaciji. Na ta način lahko odkrijemo napake pri pisanju programa (npr. tipkarske napake).

Primer 7.7 Uporaba statičnega testa pri digitalni simulaciji

Predpostavimo, da smo odtipkali naslednji program za simulacijo ekološkega modela žrtev in roparjev:

```
PROGRAM PREY AND PREDATOR
CONSTANT A11=5,A12=0.05,A21=0.0004,A22=0.2
CONSTANT RAB0=520,FOX0=85
RABDOT=A11*RAB-A12*RAB*FOX
FOXDOT=A12*RAB*FOX-A22*FOX
RAB=INTEG(RABDOT,RAB0)
FOX=INTEG(FOXDOT,FOX0)
CONSTANT TFIN=10
TERMT(T.GE.TFIN)
CINTERVAL CI=0.01
OUTPUT 100, RABDOT,RAB,FOXDOT,,FOX
END
```

Kot statično točko za preizkus lahko uporabimo kar točko v trenutku $t = 0$, ko so na izhodih integratorjev začetni pogoji. Rezultati simulacije v trenutku T=0 so

```
T        RABDOT      RAB         FOXDOT      FOX
0.         390.      520.         2193.       85.
```

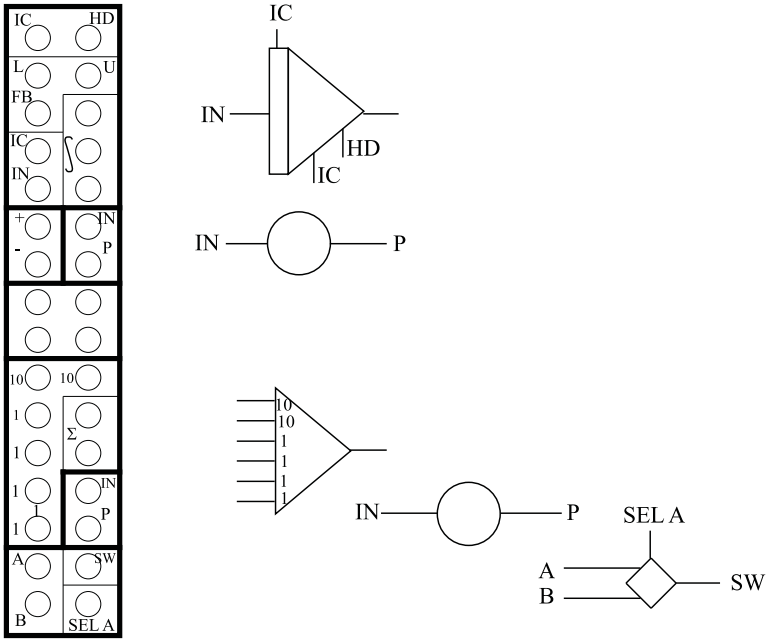
Če pa ob upoštevanju istih začetnih pogojev izračunamo spremenljivki `RABDOT` in `FOXDOT` iz originalnih enačb (1.27), dobimo vrednosti 390 in 0.68. Napaka je torej pri spremenljivki `FOXDOT`. Ker kontroliramo shemo (program) v obratnem vrstnem redu, kot se sicer izračunavajo enačbe, najprej pregledamo enačbo za izračun spremenljivke `FOXDOT`. Peta vrstica programa pokaže, da smo namesto `A21` pri pisanju programa odtipkali `A12`. □

7.6 Delo z analognimi računalniki

Zgradba analognega računalnika je modularna, tako da lahko ob nakupu izberemo poljubne kombinacije matematičnih in logičnih komponent. Sicer so po številu

matematičnih blokov relativno omejeni. Najdražji računalniki imajo nekaj deset integratorjev, potenciometrov in sumatorjev ter nekaj funkcijskih generatorjev.

Analogne programe v obliki analogne simulacijske sheme in tabele nastavitev potenciometrov realiziramo na programski (povezovalni) plošči analognega računalnika. Na tej plošči so vhodi in izhodi elektronskih vezij, ki realizirajo ustrezne matematične in logične operacije. Matematični in logični bloki so jasno vidni in smiselno razporejeni. Vsak blok ima svojo adreso. Slika 7.18 prikazuje del analogne programske plošče računalnika EAI 2000. Ta vsebuje integrator, sumator,dva potenciometra in elektronsko stikalo. Dosegljivi pa sta tudi pozitivna in negativna referenčna napetost.



Slika 7.18: Del analogne programske plošče računalnika EAI 2000

S pomočjo žic povežemo bloke, kakor zahteva analogna simulacijska shema.

Programsko ploščo lahko zamenjamo ter na ta način hranimo simulacijski program. S tem pa shranimo le povezovalni del programa (strukturo modela), ne pa tudi raznih nastavitev.

Potem, ko zvežemo program, nastavimo parametre modela ter razne krmilne parametre simulacije. Pri starejših računalnikih to omogoča posebna programska konzola na samem računalniku, na novejših mikroračunalniško krmiljenih računalnikih pa opravimo omenjene nastavitve s pomočjo tipkovnice in zaslona.

Nastavitve običajno opravimo v zato predvidenem stanju računalnika, t.j. *stanje nastavitve potenciometrov*.

Po omenjenih nastavitvah lahko izvedemo statično testiranje programa. Za to imamo posebno stanje, t.j. *stanje statičnega testiranja*.

Glavni načini delovanja analognega računalnika pa se razlikujejo po različnih stanjih integratorjev. Temeljni način delovanja je način *integriranja* (operate OP), v katerem integratorji in drugi elementi generirajo dinamično rešitev analognega programa. Razen tega je še način *začetnih pogojev* (initial conditions IC), v katerem integratorji na svojih izhodih izkazujejo napetosti začetnih pogojev in način *držanja* (hold HD), v katerem vsi integratorji zadržijo tiste napetosti, ki so jih imeli na izhodu v trenutku preklopa v to stanje.

S pomočjo vgrajenega časovnika lahko v t.i *ponavljalnem načinu* dosežemo, da se simulacijski teki izredno hitro ponavljajo (npr. nekaj sto simulacijskih tekov v sekundi). Ponavljalni način je pravzaprav menjavanje načina začetnih pogojev in načina integriranja (IC OP, IC OP, IC OP,...). Na časovniku nastavimo dolžini obeh načinov (perioda IC in perioda OP, t.j. dolžina simulacijskega teka). Ponavljalni način daje simulaciji na analognem računalniku veliko ilustrativnost. V tem načinu lahko učinkovito spremljamo vpliv nekega parametra na rezultate simulacije. S spreminjanjem ročnega potenciometra, ki predstavlja parameter modela, praktično istočasno na osciloskopu že dobimo rezultate. Pri digitalni simulaciji je potrebno po koncu simulacijskega teka spremeniti parameter in nato ponoviti simulacijski tek.

Časovnik analognega računalnika omogoča v povezavi z ustreznim krmiljenjem integratorjev tudi več različnih hitrosti simulacije, ne da bi bilo potrebno časovno preskalirati enačbe. Ena hitrost je v milisekundnem področju in omogoča učinkovito spremljanje rezultatov na osciloskopu (enota računalniške neodvisne spremenljivke je npr. $1ms$). Druga hitrost je v sekundnem področju in omogoča uporabo koordinatnega risalnika (enota računalniške neodvisne spremenljivke je npr. $1s$).

Simulacija v ponavljalnem načinu predstavlja običajno simulacijo v skrčenem ali v raztegnjenem času, redkeje tudi v realnem času. Če pa je analogni računalnik priključen na nek realni proces (HIL, hardware in the loop), potem mora delovati v realnem času, kar pomeni, da je računalniška neodvisna spremenljivka enaka problemski neodvisni spremenljivki. V tem primeru računalnik ne deluje v ponavljalnem načinu, ampak ga moramo postaviti za stalno v način integriranja (npr. analogni računalnik ima funkcijo regulatorjev v jedrskem reaktorju). V

takem primeru seveda ne nastavljamo dolžine simulacijskega teka.

Razen rezultatov, ki jih spremljamo s pomočjo digitalnega voltmetra, osciloskopa ali koordinatnega risalnika, pa lahko med simulacijo spremljamo tudi razne druge statuse. Eden od pomembnejših je status za prekoračitev območja ojačevalnikov (overload).

Na koncu še enkrat povzemimo korake, ki jih je potrebno izvesti pri neki simulacijski študiji na analognem računalniku:

- Iz matematičnega modela razvijemo analogno simulacijsko shemo. Ob tem uporabimo indirektno metodo ali metode za simulacijo prenosne funkcije.
- S postopkom amplitudnega in časovnega skaliranja razvijemo skalirano analogno simulacijsko shemo. Le ta skupaj s tabelo potenciometrov (in morebitnih drugih parametrov, npr. podatkov za funkcijske generatorje) omogoča realizacijo programa na analognem računalniku.
- Na analogni programski plošči zvežemo program, t.j. povežemo potrebne matematične (in morebitne logične) bloke.
- Nastavimo potenciometre (in morebitne druge parametre modela). Nastavimo tudi parametre časovnika (čas stanja začetnih pogojev (IC) in čas simulacijskega teka (OP)).
- S statičnim preizkusom odkrijemo morebitne napake pri razvoju analogne sheme, pri skaliranju, pri povezovanju ali pa napake zaradi okvare komponent analognega računalnika.
- Nato izberemo način, v katerem bo analogni računalnik posredoval rešitev. Običajno je to ponavljalni način delovanja. Ob spremljanju rezultatov simulacije se lahko pokaže potreba po reskaliranju skaliranih enačb.
- Ko dobimo zanesljiv in dobro delujoči simulacijski program, se lahko lotimo prave problematike obravnavane študije. Spremembe parametrov pa tudi strukture lahko izvajamo hitro, zelo ilustrativno in interaktivno.

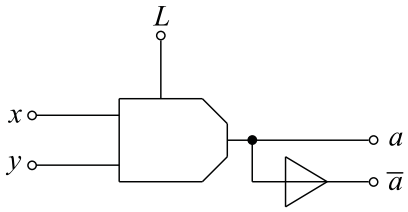
7.7 Analogno-hibridna simulacija

Če analognemu računalniku dodamo le nekatere digitalne logične komponente, pravimo takemu sistemu *analogno-hibridni računalnik*.

Digitalni logični elementi imajo priključne sponke na analogni programski plošči ali na posebni logični programski plošči. Logični programski elementi zelo povečajo sposobnosti eksperimentiranja z analogno - hibridnimi računalniki. S pomočjo logičnih elementov lahko sprogramiramo optimizacijo, parametrizacijo, metodo optimalnega prileganja odziva modela neki krivulji, ...itd. Razen povsem logičnih elementov so tudi elementi, ki povezujejo analogni in logični del programa. Ti elementi so lahko krmiljeni z logičnimi signali ali pa dajejo logične signale glede na vrednosti analognih vhodov.

V nadaljevanju bomo predstavili le bistvene elemente za povezavo analognih in logičnih signalov ter nekatere povsem logične elemente.

- *Komparator.* Komparator je element, ki na osnovi relacije med dvema analognima signaloma (x, y) generira logični signal a ($a =$ "1", če $x > y$ in $a =$ "0", če $x < y$) in negirani logični signal $\overline{a}$. Signal y imenujemo običajno prag komparatorja. To je vedno spodnji signal pri orientaciji iz leve proti desni. Ikono prikazuje slika 7.19.

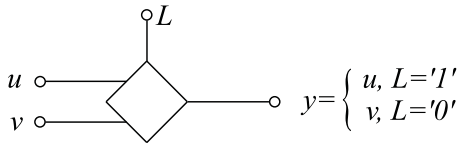


Slika 7.19: Komparator

  Komparator ima vgrajeno tudi manjšo histerezo, ki prepreči nezaželeno in nekontrolirano preklapljanje, kadar sta vhodna signala skoraj enaka. Razen analognih vhodov ima običajno tudi logični vhod za omogočanje delovanja (s tem vhodom lahko komparator preklopimo v nekakšen način držanja (hold)).

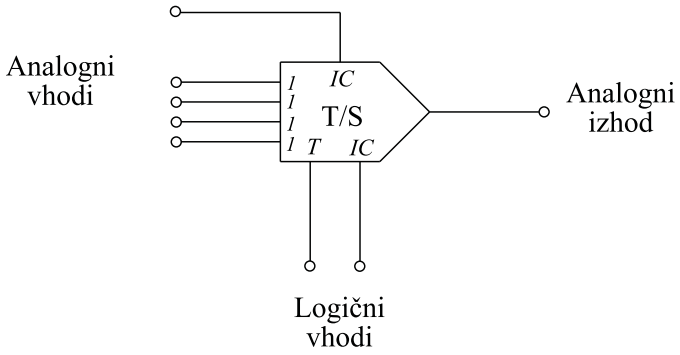
- *Stikala in releji.* Zasledimo različne vrste elektronskih stikal pa tudi relejev pri starejših računalnikih. Elektronsko stikalo je krmiljeno z logičnim signalom in v odvisnosti od vrednosti tega signala je analogni izhod enak enemu ali drugemu analognemu vhodu. Če damo signal le na en analogni vhod, lahko smatramo, da je drugi analogni vhod analogna masa. Ikono za elektronsko stikalo prikazuje slika 7.20.

- *Sledilno-shranjevalni element (track-store).* To je element, ki predstavlja nekakšen analogni spomin. V stanju sledenja sledi negativni vsoti vhod-



Slika 7.20: Elektronsko stikalo

nih napetosti (deluje kot sumator), v stanju držanja pa zadrži napetost, ki je bila na izhodu v trenutku preklopa iz stanja sledenja v stanje držanja (podobno kot pri integratorju način držanja - hold). Preklapljanje med stanjema sledenja in shranjevanja omogoča eden od dveh logičnih vhodov (T). Drugi logični vhod (IC) pa omogoča preklapljanje med sledilno shranjevalnim načinom in stanjem začetnih pogojev. V stanju začetnih pogojev se na izhodu enote pojavi začetni pogoj, t.j. napetost na posebnem analognem vhodu (podobno kot pri integratorju). Sledilno shranjevalno enoto prikazuje slika 7.21.

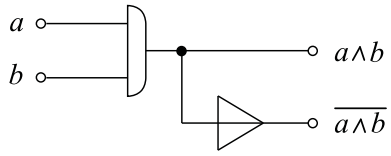


Slika 7.21: Ikona sledilno shranjevalne enote

Nabor in oblika ikon povsem logičnih elementov pa se za različne tipe računalnikov zelo razlikujeta. Omenili bomo naslednje elemente:

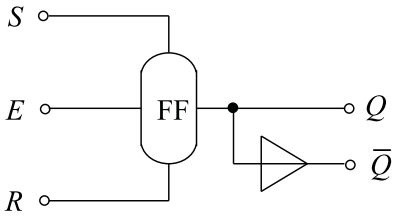
- *Logična vrata.* Večina računalnikov ima le IN vrata z dvema ali s štirimi vhodi $(a, b,...)$, z izhodom in z negiranim izhodom. Ikono prikazuje slika 7.22. Velja naslednja pravilnostna tabela:

a	b	$a \wedge b$	$\overline{a \wedge b}$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0



Slika 7.22: Ikona za IN vrata

- *Bistabilni element.* Bistabilna vezja (flip-flop) so spominski logični elementi. Delovanje je sinhronizirano z urinimi impulzi, katerih periodo je možno nastaviti. S takim sinhronskim delovanjem se izognemo problemom zaradi različnih zakasnitev pri razširjanju signalov. Odločitev o delovanju in izvršitev časovno ne sovpadata. Torej se bistabilno vezje najprej odloči, kaj narediti in nato v natančno določenem trenutku to tudi naredi. Ikono SR bistabilnega vezja prikazuje slika 7.23.

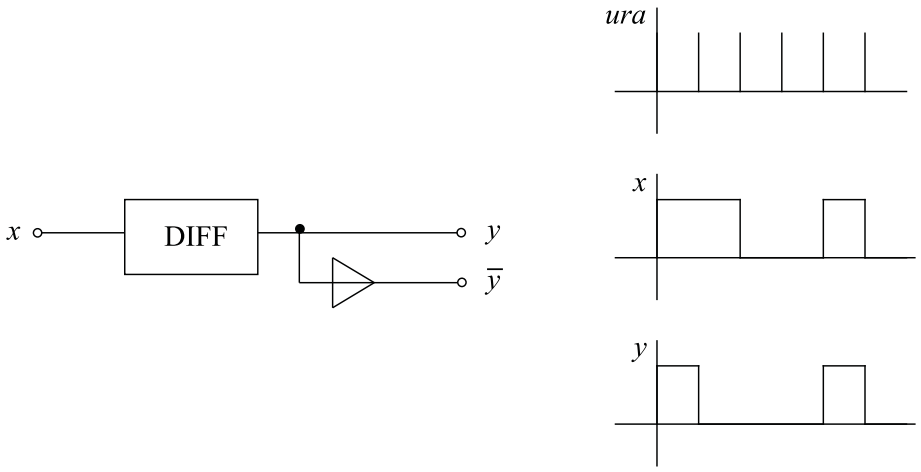


Slika 7.23: Ikona za bistabilno vezje

Vezje ima logična vhoda za setiranje (S) in resetiranje (R), logični vhod za omogočanje (E) ter izhod (Q) in negirani izhod ($\overline{Q}$). Velja naslednja pravilnostna tabela:

		$Q_N = 0$		$Q_N = 1$	
S	R	Q_{N+1}	$\overline{Q}_{N+1}$	Q_{N+1}	$\overline{Q}_{N+1}$
0	0	0	1	1	0
0	1	0	1	0	1
1	0	1	0	1	0
1	1	1	0	0	1

- *Logični diferenciator.* Vezje generira logični impulz dolžine ene urine periode pri prehodu vhodnega signala iz "0"v "1"(diferenciator prve fronte). S pomočjo dodatnih logičnih elementov je seveda zelo enostavno narediti tudi diferenciator zadnje fronte ali pa obeh front. Ikono diferenciatorja in primer signalov prikazuje slika 7.24.



Slika 7.24: Ikona logičnega diferenciatorja in primer signalov

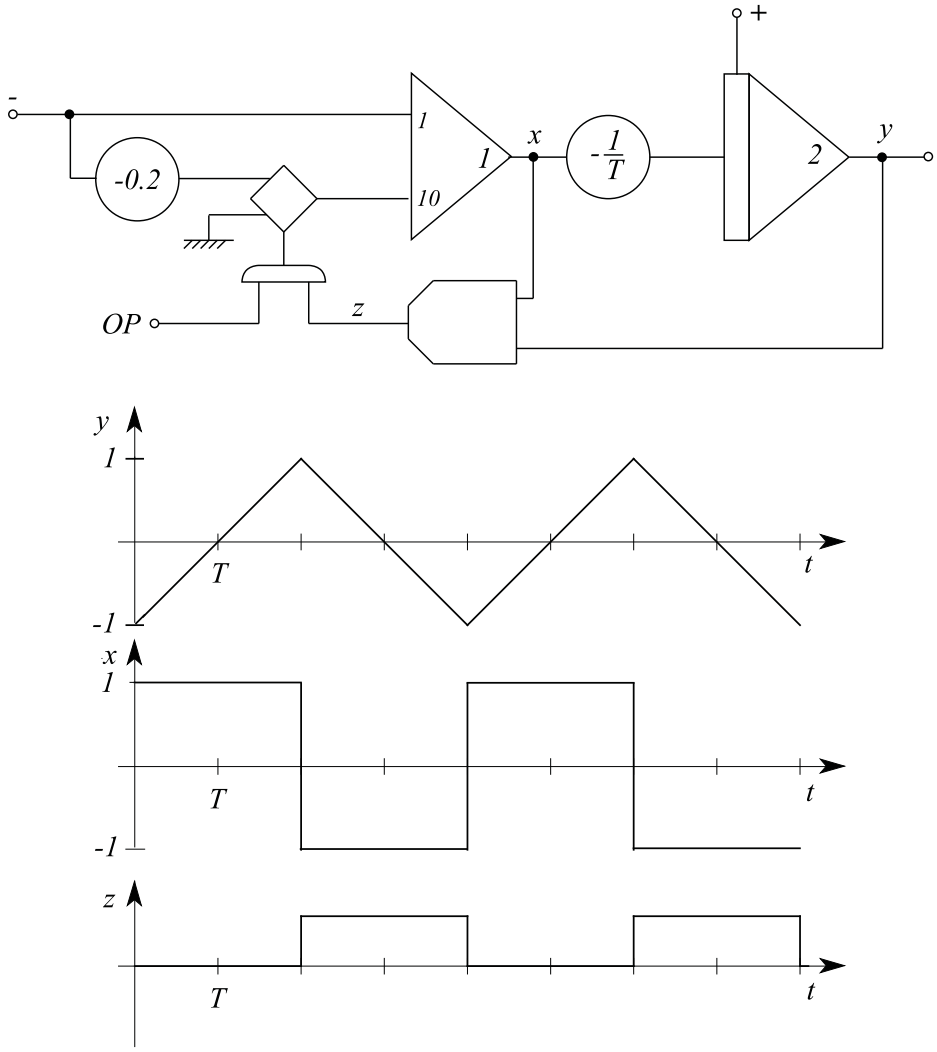
- *Splošnonamenski registri.* Te enote vsebujejo štiri do osem bistabilnih vezij, ki jih je možno uporabiti posamezno ali skupinsko. Običajno jih uporabljamo kot dvojiške števce ter premikalne ali krožne registre.

- *Števci.* Enote omogočajo dvojiško ali dekadno štetje vhodnih impulzov. Ob vsakem impulzu se vrednost števca dekrementira za ena, torej se vsebina spreminja od neke začetne vrednosti do vrednosti nič. Ob tem se ustrezno spremeni izhod števca. Števce je možno setirati ali resetirati z logičnimi signali (programa) ali pa z določenim ukazom (preko tipk ali tipkovnice).

Primer 7.8 Analogno-hibridna simulacijska shema za generacijo trikotnih impulzov

Analogno-hibridni shemi za generacijo vlaka trikotnih impulzov z ustreznimi signali prikazujeta sliki 7.25 in 7.26.

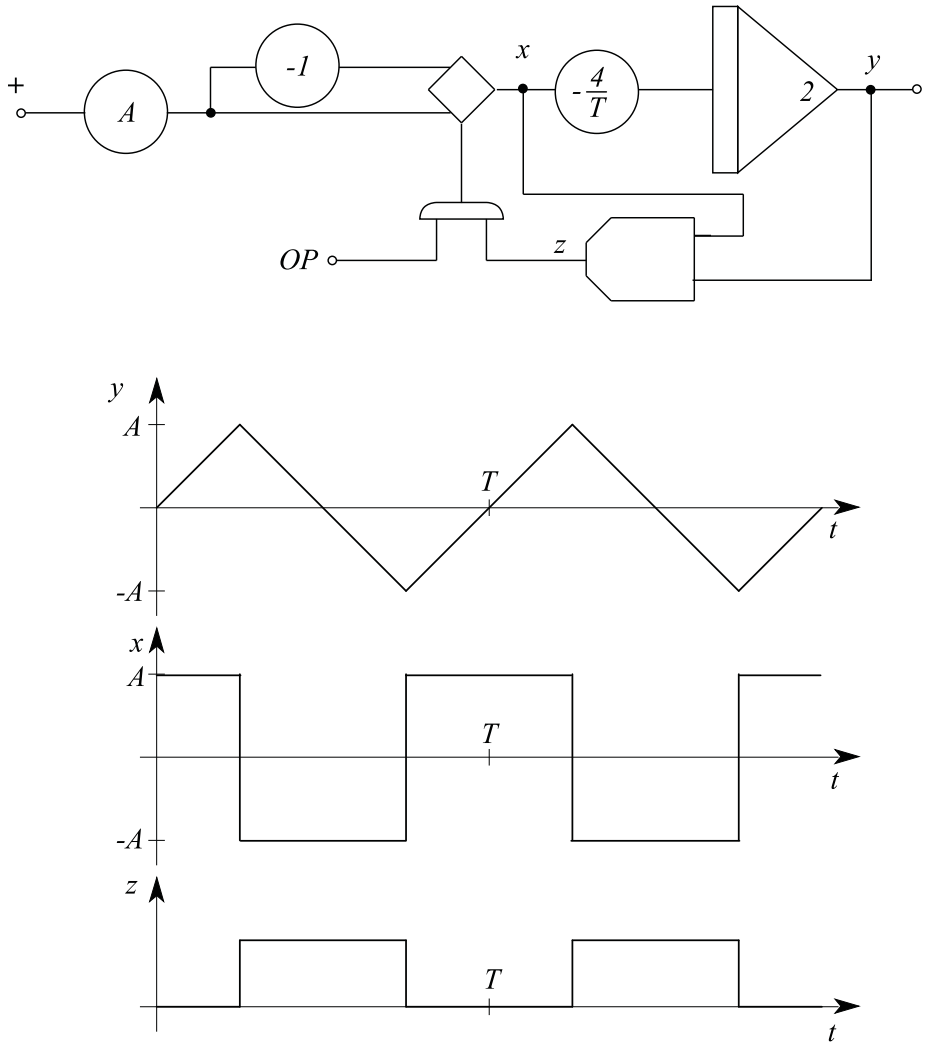
Prva shema generira trikotne impulze s periodo $4T$ z enosmerno vrednostjo nič in z začetno vrednostjo -1. Na začetku je izhod komparatorja logična "0"in elektronsko stikalo prevaja spodnji signal, ki je sklenjen na maso. Zato je izhod sumatorja



Slika 7.25: Generacija trikotnih impulzov - 1. možnost

pozitivna referenčna napetost, izhod integratorja pa narašča. Ko doseže vrednost pozitivne referenčne napetosti, gre izhod komparatorja v logično "1". Zato začne elektronsko stikalo prevajati zgornji signal in izhod sumatorja dobi negativno referenčno napetost. Izhod integratorja začne upadati in ko upade do negativne referenčne napetosti, gre izhod komparatorja spet v "0". Zaradi potenciometra z vrednostjo $-1/T$ pred integratorjem je perioda signala $4T$. Logični signal OP je signal, ki določa stanje računaj (operate).

Druga shema pa generira trikotne impulze s periodo T in z začetno vrednostjo nič. Tudi v tem primeru je izhod komparatorja v začetnem del na logični "0",

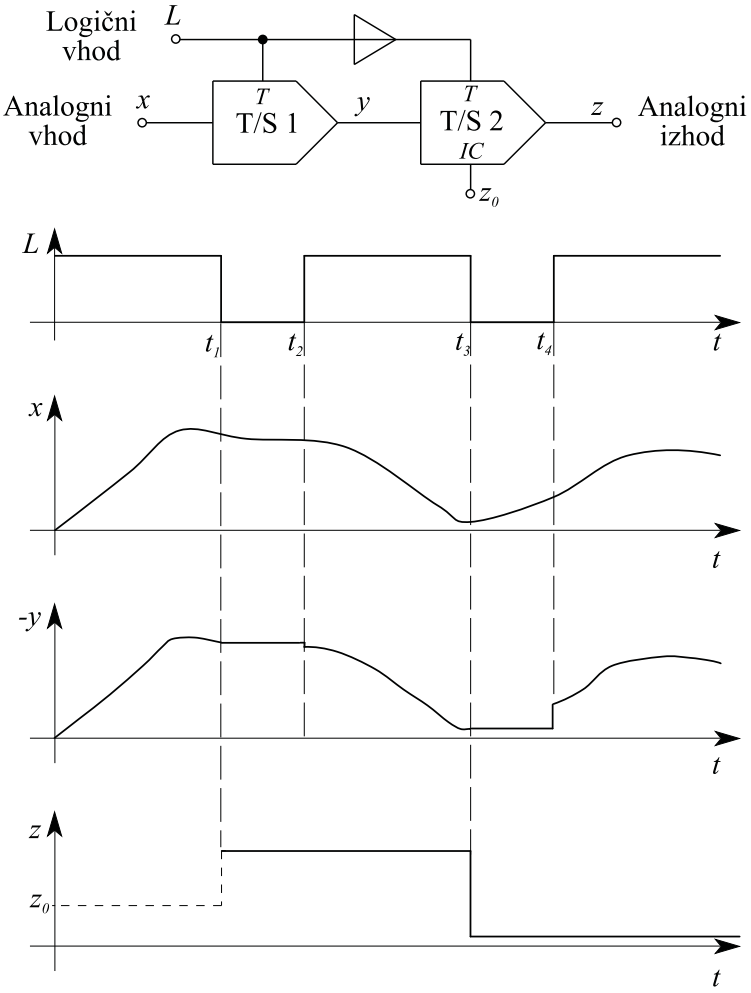


Slika 7.26: Generacija trikotnih impulzov - 2. možnost

elektronsko stikalo pa prepušča spodnji signal. Zato je na vhodu integratorja negativna napetost in izhod integratorja začne naraščati. Ko doseže vrednost A, gre izhod komparatorja v "1", elektronsko stikalo pa zamenja predznak napetosti na vhodu integratorja. Zato začne izhod integratorja upadati. □

Primer 7.9 Analogni spomin s parom sledilno-shranjevalnih enot

Analogni spomin (pomnenje analognih vrednosti v določenih trenutkih in zadrževanje teh vrednosti za določen čas) realiziramo s povezavo dveh sledilno-shranjevalnih enot. Shemo in značilne signale prikazuje slika 7.27.

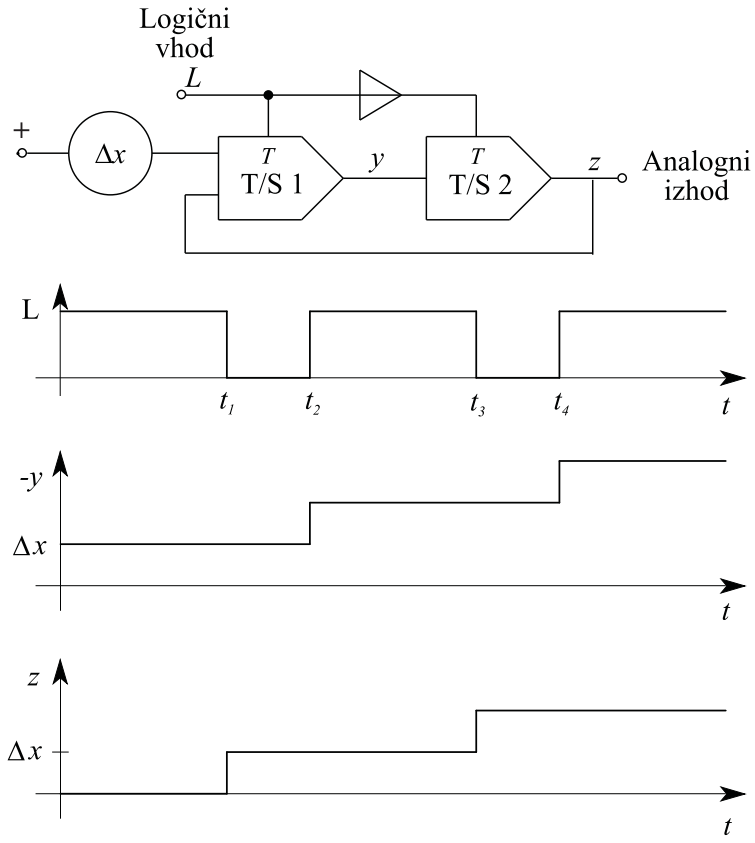


Slika 7.27: Analogni spomin

Na časovnem intervalu $[0, t_1]$ prva enota sledi analognemu vhodnemu signalu, druga enota pa drži shranjeno vrednost, ki je določena z njenim začetnim pogojem. Na intervalu $[t_1, t_2]$ prva enota drži vrednost iz trenutka t_1, druga pa sledi tej vrednosti in je zato tudi konstantna. Na intervalu $[t_2, t_3]$ prva enota sledi analognemu vhodu, druga enota pa drži svojo vrednost iz trenutka t_2, torej se njen konstantni izhodni signal ne spremeni. Na intervalu $[t_3, t_4]$ prva enota drži svojo izhodno vrednost iz trenutka t_3, druga enota pa sledi tej konstantni vrednosti. Izhod druge sledilno-shranjevalne enote je torej odsekovno konstantni signal, njegove konstantne vrednosti pa so vzorci analogne vhodne napetosti v trenutkih $t_1, t_3, ...$ Da zagotovimo vrednost izhoda na prvem intervalu, uporabimo na drugi enoti analogni vhod z_0. Torej tak spominski par omogoča nekakšno analogno vzorčenje z zadrževalnikom. □

Primer 7.10 Analogni akumulator

Shemo, ki jo prikazuje slika 7.28, imenujemo analogni akumulator.



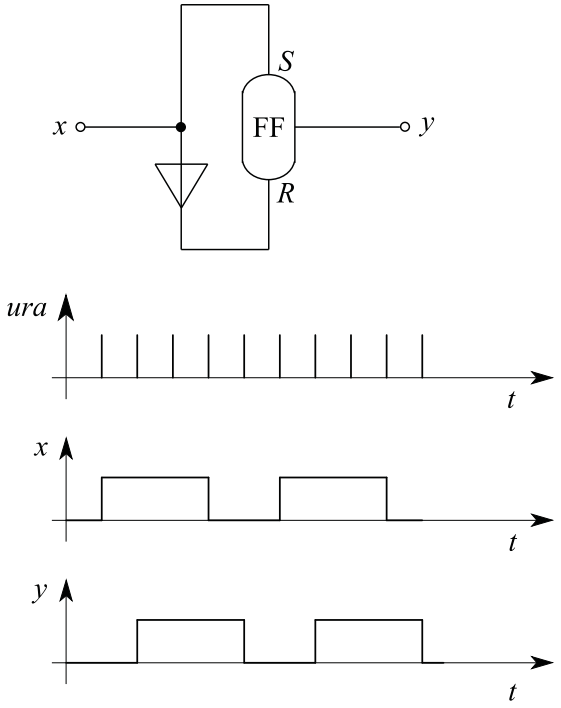
Slika 7.28: Analogni akumulator

Z njo realiziramo stopničasto naraščajoči analogni izhodni signal s prirastkom Δx. Torej shema med delovanjem akumulira (prišteva) prirastek, ki je določen z nastavitvijo potenciometra. Pri tem smo predpostavili, da je začetni pogoj druge sledilno shranjevalne enote enak nič. Če pa bi bila začetna vrednost z_0, pa bi imel signal $-y$ prvo konstantno vrednost $z_0 + \Delta x$, signal z pa z_0.

Analogni akumulator lahko koristimo za parametersko študijo, v kateri ponavljamo simulacijske teke pri spreminjajočem se parametru (vrednosti npr. 0, Δx, $2\Delta x$, $3\Delta x$,...). Logični signal L mora biti v tem primeru kar logični signal sistemskega časovnika in krmili računalnik med stanjema IC in OP. □

Primer 7.11 Kasnilnik logičnih signalov

S pomočjo sheme, ki jo prikazuje slika 7.29, zakasnimo logični signal za čas periode urinih impulzov.



Slika 7.29: Kasnilnik logičnih signalov

□

Primer 7.12 Logični diferenciator

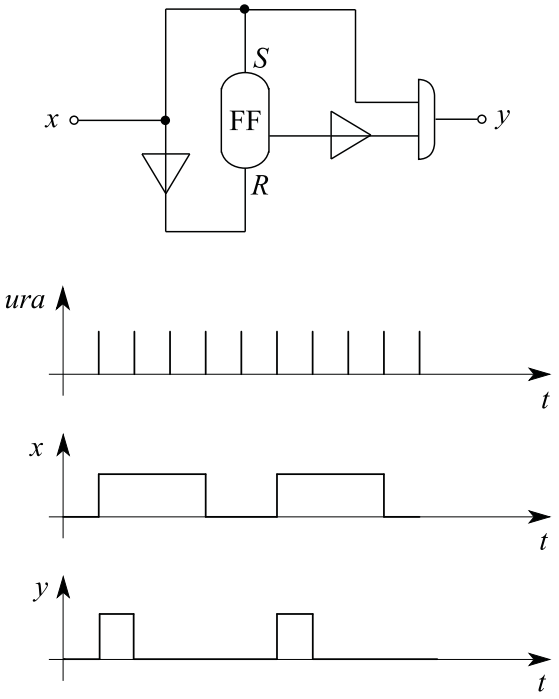
S pomočjo sheme, ki jo prikazije slika 7.30, realiziramo diferenciranje logičnega signala. Ob vsaki prvi fronti vhodnega signala x dobimo impulz dolžine ene urine periode.

□

Primer 7.13 Šumni generator

Slika 7.31 prikazuje analogno-hibridno shemo šumnega generatorja.

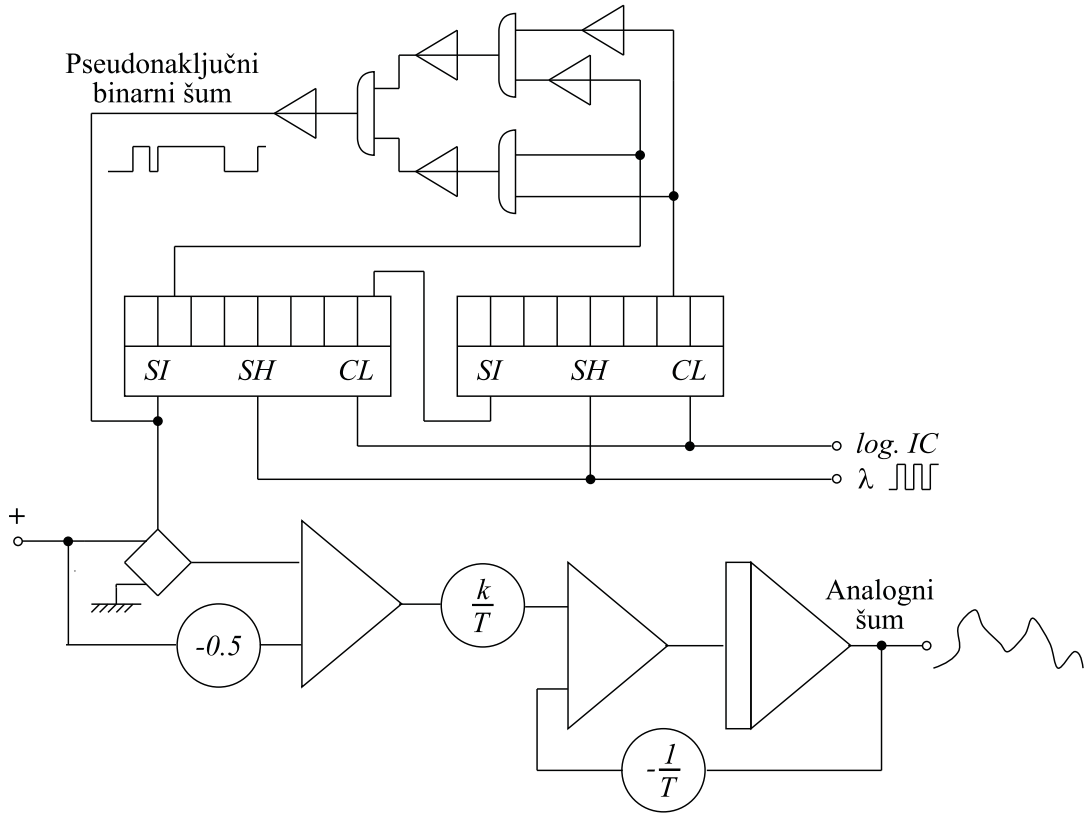
Uporabljamo dva povezana osem bitna premikalna registra. Zadnji bit prvega registra je povezan na vhod SI(serial input) drugega registra, to pa je vhod,



Slika 7.30: Logični diferenciator

ki definira prvi bit pri pomiku v desno. Drugi bit prvega in sedmi bit drugega registra peljemo v logično vezje, katerega izhod povežemo na vhod SI prvega premikalnega registra. To je hkrati tudi pseudonaključni binarni šum (PRBN - pseudo random binary noise). Z različnimi logičnimi vezji (v primeru na sliki 7.31 uporabljamo logično ekvivalenco, ki je ugodna zato, ker omogoča začetek delovanja s praznima registroma) in z uporabo različnih bitov iz premikalnih registrov generiramo različne binarne šume, ki se med seboj predvsem razlikujejo po dolžini ponavljanja sekvence (zato pseudonaključni šum). Različne med seboj dokaj nekorelirane šume dobimo z različnimi začetnimi vsebinami v registrih. Na vhod SH pa priključimo vlak impulzov, ki določa premikalne trenutke. Frekvenca teh impulzov vpliva na pasovno širino šuma. Čim večja je frekvenca, večja je pasovna širina in bolj se karakteristika šuma približuje karakteristiki belega šuma. Vhoda CL (clear) pa sta povezana na logični signal za periodo začetnih pogojev (IC). To je potrebno pri ponavljalnem delovanju računalnika, saj morajo v stanju začetnih pogojev tudi registri vedno dobiti enake vsebine (v tem primeru 0).

Dodatni analogni del v sliki 7.31 pa omogoča generacijo analognega šuma. Analogno vezje je nizkopasovni filter (sistem 1. reda), ki s filtriranjem višjih frekvenc pseudonaključnega binarnega šuma le tega preoblikuje v analogni šum.



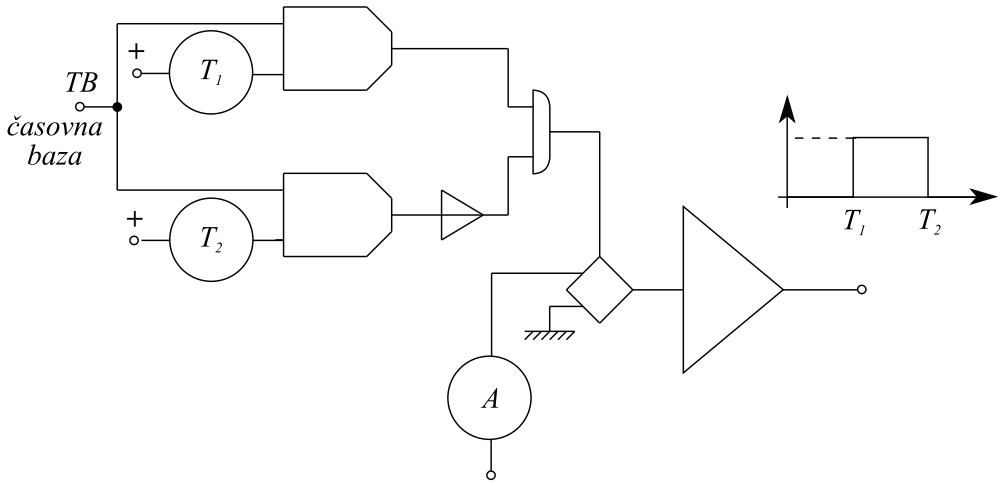
Slika 7.31: Šumni generator

S konstanto k izbiramo amplitudo šuma, s konstanto T pa njegovo pasovno širino.
□

Primer 7.14 Generiranje impulza

Slika 7.32 prikazuje analogno - hibridno shemo za generiranje enkratnega impulza, ki ima vrednost A na intervalu $[T_1, T_2]$, drugje pa nič.

Na prva vhoda obeh komparatorjev pripeljemo računalnikovo časovno bazo. Le-ta se spreminja od nič do pozitivne reference (od $0V$ do $+10V$). Na intervalu $[0, T_1]$ sta izhoda obeh komparatorjev "0", zato elektronsko stikalo prepušča spodnji signal, torej je izhod nič. Ko časovna baza doseže vrednost T_1, se izhod zgornjega komparatorja postavi v "1", izhod IN vezja postane tudi "1", zato je izhod na intervalu $[T_1, T_2]$ enak A. V trenutku, ko časovna baza doseže vrednost T_2, postane izhod spodnjega komparatorja "1", izhod IN vezja pa "0", zato postane izhod enak nič.

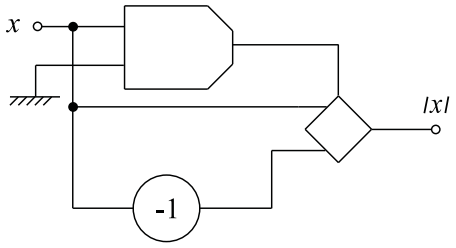


Slika 7.32: Generiranje impulza

S potenciometri torej določimo trenutek nastopa, dolžino in amplitudo impulza. □

Primer 7.15 Absolutna vrednost

Slika 7.33 prikazuje analogno - hibridno shemo, katere izhod je absolutna vrednost vhoda.



Slika 7.33: Shema za absolutno vrednost

Če je vhodni signal pozitiven, je izhod komparatorja "1"in elektronsko stikalo prepušča na izhod kar vhodni signal. Če pa je vhodni signal negativen, potem elektronsko stikalo prepušča njegovo nasprotno vrednost.

□

7.8 Hibridna simulacija

Če analognemu računalniku dodamo digitalnega in med simulacijo izkoriščamo procesne zmožnosti obeh, pravimo takemu simulacijskemu sistemu *hibridni računalnik*. Omenili pa smo že, da v primeru, če analognemu računalniku dodamo le nekatere digitalne logične bloke, takemu sistemu pravimo *analogno-hibridni računalnik*.

Vsak hibridni sistem je sestavljen iz treh delov: iz analognega računalnika, iz digitalnega računalnika in iz ustreznega vmesnika (Bekey, Karplus, 1968). Oba računalnika lahko delujeta samostojno ali pa v pravem hibridnem delovanju. Vmesnik je bil zlasti kompleksen (in seveda drag) pri starejših hibridnih sistemih, saj je moral razen analogno - digitalnih in digitalno - analognih pretvorb omogočati tudi krmiljenje nekaterih komponent in načinov delovanja analognega računalnika (npr. nastavljanje servopotenciometrov, funkcijskih generatorjev, ...). Pri novejših hibridnih sistemih je vmesnik enostavnejši, saj je analogni računalnik mikroračunalniško krmiljen.

Vloga in zgradba digitalnega računalnika in vmesnika v hibridnem sistemu je določena z vrsto operacij, ki jih mora izvrševati hibridni sistem. Le te razdelimo v časovno nekritične in v časovno kritične operacije (EAI-2000, 1982).

Med *časovno nekritične* štejemo operacije programiranja oz. ustreznih nastavitev analognega računalnika pred simulacijskim tekom ali med dvema zaporednima simulacijskima tekoma. Take operacije lahko opravi digitalni računalnik s pomočjo dvosmerne serijske povezave (RS 232) s krmilno - nastavitvenim mikroračunalnikom v analognem delu. Tipične tovrstne operacije so: nastavljanje potenciometrov analogne simulacijske sheme, nastavljanje funkcijskih generatorjev, nekatere nastavitve logičnih komponent, nastavitve časovne skale, krmiljenje stanj analognega računalnika, odčitavanje izhodov ojačevalnikov, itd. S tako konfiguracijo je možno na avtomatizirani način izvajati tudi eksperimente, ki vključujejo osnovni eksperiment t.j. simulacijski tek, a med simulacijskim tekom ne zahtevajo obsežnejšega prenosa podatkov med analognim in digitalnim računalnikom.

Med *časovno kritične* pa štejemo zlasti operacije prenosa podatkov med analognim in digitalnim računalnikom med simulacijo. Vmesnik mora v tem primeru omogočati paralelni prenos podatkov med digitalnim računalnikom in krmilno - nastavitvenim mikroračunalnikom, vsebovati mora prekinitvene linije, krmilne linije itd. Omogočati mora izredno hitre analogno - digitalne in digitalno -

analogne pretvorbe (z velikim številom kanalov), prenos podatkov z direktnim dostopom do pomnilnika (DMA) itd. Tak sistem potem omogoča pravo hibridno simulacijo, v kateri se lahko določeni deli modela simulirajo na analognem, določeni pa na digitalnem računalniku. Tipičen je primer simulacije regulacijskega sistema, kjer digitalni računalnik simulira delovanje diskretnega (ali zveznega) regulatorja, zvezni proces pa simuliramo na analognem računalniku. Taka konfiguracija hibridnega sistema omogoča tudi učinkovito spremljanje, dokumentiranje in obdelovanje rezultatov simulacije pa tudi izvrševanje kompleksnih eksperimentov s prenosom podatkov med analognim in digitalnim računalnikom tudi med simulacijskim tekom (npr. izračunavanje kriterijske funkcije med optimizacijo).

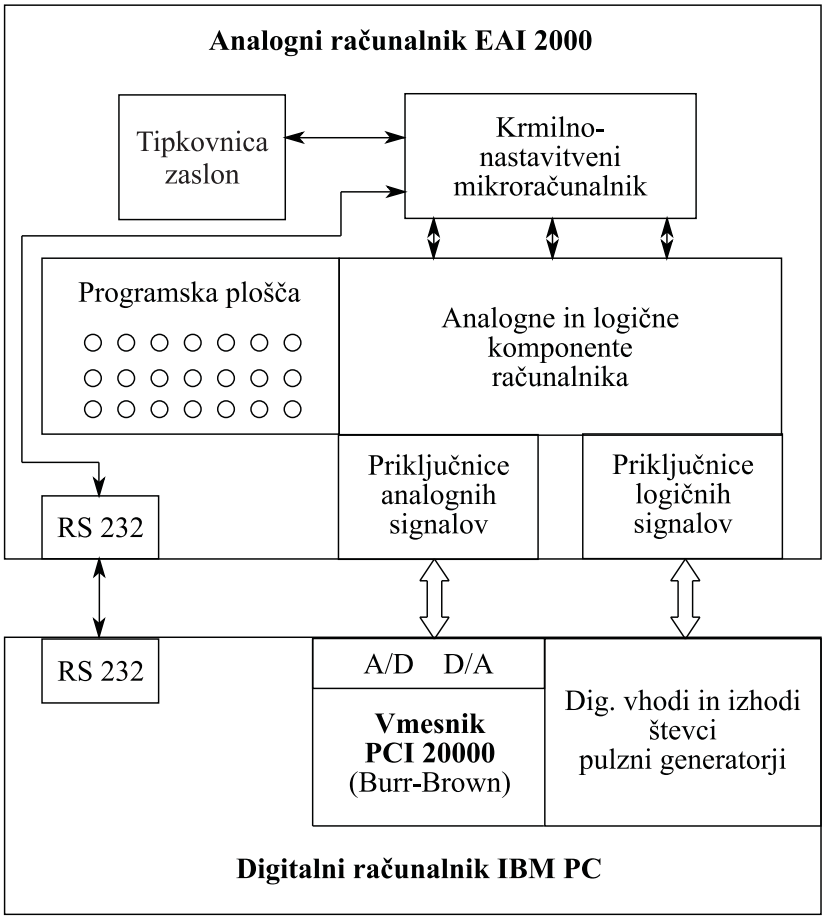
Najsposobnejši hibridni sistemi vsebujejo točno določen tip analognega in digitalnega računalnika in ustreznega vmesnika. Zaradi izredno visoke cene takih sistemov pa lahko v primeru, če nimamo tako velikih zahtev, shajamo s cenenimi standardnimi digitalnimi računalniki (tudi osebnimi) in s standardnimi izvedbami procesnih vmesnikov. Eno od takih cenenih razširitev analognega računalnika EAI 2000 v hibridni sistem, prikazuje slika 7.34.

V tej izvedbi uporabljamo za povezavo osebnega in analognega računalnika procesni vmesnik PCI 20000 (proizvajalec Burr-Brown). Vmesnik vsebuje A/D in D/A pretvornike, digitalne vhodno izhodne linije, števce in pulzne generatorje. Vhodi in izhodi iz vmesnika so povezani na analogne in logične priključne sponke analognega računalnika (t.i. tranki). Preko serijske linije pa se prenašajo časovno nekritični podatki med osebnim računalnikom in krmilno nastavitvenim mikroračunalnikom analognega računalnika.

Bolj kot materialna oprema je seveda kritična programska oprema, ki jo za take nestandardne konfiguracije ni možno kupiti. Osnovne funkcije programiranja je sicer mogoče relativno enostavno in hitro sprogramirati, bolj problematična pa je programska oprema, ki naj omogoča pravo hibridno simulacijo. V ta namen lahko zelo dobro služi simulacijski jezik, ki je prirejen za delo v realnem času z možnostjo komuniciranja preko enot procesnega vmesnika.

Končajmo kratek opis hibridnih sistemov z navedbo tistih simulacijskih nalog in karakteristik simuliranih procesov, za katere je smiselno uporabiti hibridno simulacijo:

- izredno hitra simulacija,
- simulacija v realnem času ali hitreje,



Slika 7.34: Cenena izvedba hibridnega sistema

- načrtovanje sistemov vodenja, vrednotenje in preizkušanje,
- simulacija, ki zahteva direktno interakcijo z uporabnikom,
- simulacija s priključenimi zunanjimi napravami (H-I-L),
- kompleksni, časovno spremenljivi, nelinearni sistemi,
- prisotnost nezveznosti,
- togi sistemi,
- sistemi, ki so opisani z zelo sklopljenimi diferencialnimi in algebrajskimi enačbami itd.

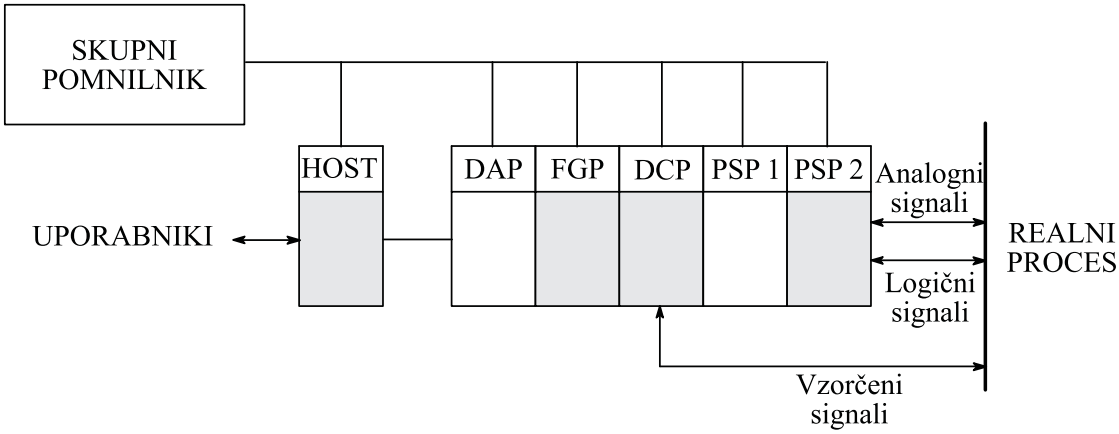
Primer 7.16 Hibridni sistem SIMSTAR

Večprocesorski hibridni sistem SIMSTAR je najpomembnejši predstavnik nove generacije hibridnih sistemov (Landauer, 1988) in predstavlja na področju simulacije danes v svetu eno najsposobnejših orodij. Njegova glavna prednost pred drugimi hibridnimi sistemi je v tem, da poteka celotno programiranje na nivoju simulacijskega jezika. Povezovanje komponent in skaliranje sta povsem avtomatizirana.

Pri opisu materialne in programske opreme računalnika SIMSTAR bomo razen že ustaljenih uporabili tudi nekaj novih ali spremenjenih izrazov, ki jih uporablja proizvajalec v priročniku.

Materialna oprema

Slika 7.35 prikazuje zgradbo večprocesorskega sistema SIMSTAR.



Slika 7.35: Zgradba večprocesorskega sistema SIMSTAR

Minimalna osnovna konfiguracija sestoji iz digitalnega aritmetičnega procesorja (DAP - digital arithmetic processor) in iz paralelnega simulacijskega procesorja - analogno-hibridnega računalnika (PSP - parallel simulation processor), ki je lahko preko zveznih in logičnih signalov priključen na realni proces. Osnovni konfiguraciji se lahko doda procesor za hitro generiranje funkcij (FGP - function generation processor), pretvorniški procesor (DCP - data conversion processor) za hitro izvrševanje pretvorb A/D in D/A ter glavni računalnik (HOST), katerega bistvena naloga je generacija simulacijskih programov za procesorje. Tej

osnovni konfiguraciji se lahko doda še en paralelni simulacijski procesor. V maksimalni izvedbi pa je možno na en glavni računalnik povezati tri take konfiguracije. Skupni pomnilnik in podatkovno vodilo omogočajo komunikacijo med procesorji. Nekateri procesorji so povezani tudi interno, kar omogoča zelo hitro prenašanje podatkov.

Digitalni aritmetični procesor (DAP) je lahko katerikoli procesor iz družine GOULD, CONCEPT/32 (npr. GOULD 32/87). Med simulacijo deluje DAP v realnem času preko periodičnih prekinitev. Lahko izvaja vse simulacijske operacije. Tipične operacije, ki jih DAP izvaja ob vsaki prekinitvi, so:

- prečita določene vrednosti iz pomnilnika, kamor jih je shranil pretvorniški procesor (DCP),
- ustrezno pripravi podatke za morebitno obdelavo s procesorjem za generacijo funkcij (FGP) ter ustrezno inicializira FGP, zbira podatke, ki jih generira FGP in po potrebi interpolira med točkami,
- obdeluje podatke, ki prihajajo iz paralelnega simulacijskega procesorja (PSP),
- detektira napako v primeru, če DAP ne zmore operacij znotraj intervala med dvema prekinitvama.

Pretvorniški procesor (DCP) omogoča izredno hiter prenos podatkov med DAP in PSP ter realnim procesom. Hitra pretvorniška sistema A/D in D/A omogočata vzorčenje do cca 300 kHz na vsakem kanalu in se avtomatično prožita periodično z izbrano frekvenco vzorčenja (frame rate). Po izvršenih pretvorbah A/D DCP sproži prekinitev DAP, ki sprejme podatke, jih ustrezno obdela in pošlje v pretvorniški sistem D/A. Le-ta jih prenese na PSP ali v realni proces. Kompletna procedura znotraj DCP se izvede s pomočjo materialne opreme brez intervencije programske opreme v DAP.

Generator funkcij (FGP) je sam po sebi večprocesorski sistem z lokalnim pomnilnikom. S pomočjo tabel omogoča izredno hitro generiranje pseudo zveznih funkcij do štirih neodvisnih spremenljivk ter linearno interpolacijo. Število možnih funkcij in število lomnih točk je odvisno od velikosti uporabljenega pomnilnika. Izvršitev operacij v FGP sproži procesor DAP. Ob tem se v FGP izvršijo naslednje operacije:

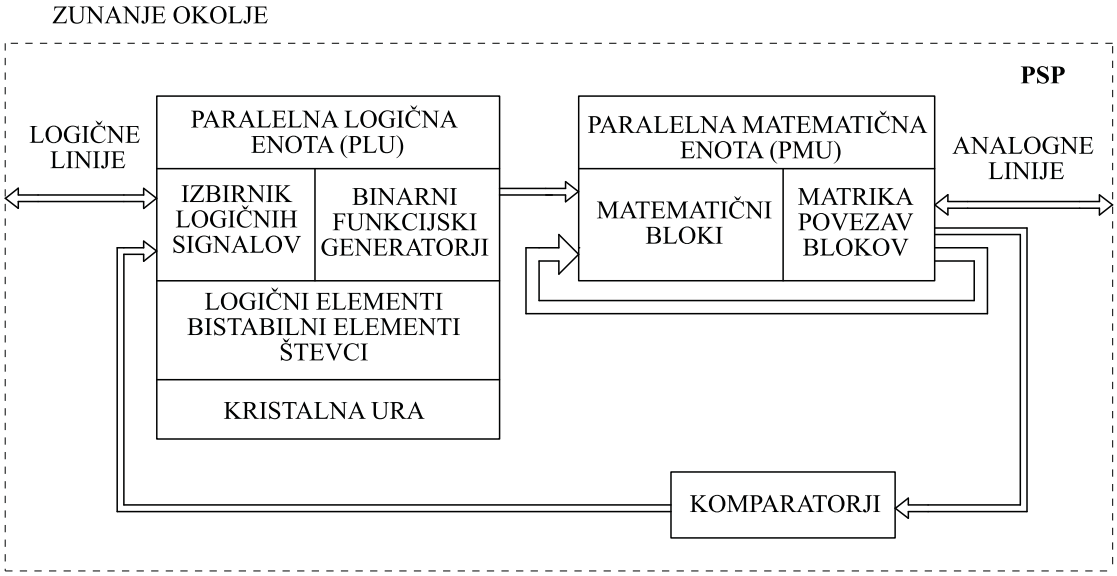
- vrednosti vhodnih argumentov se prenesejo iz pomnilnika,

- izračuna se funkcijska odvisnost,
- generirana vrednost se zapiše v pomnilnik.

Operacije v FGP se izvršujejo avtonomno (v materialni opremi brez programske opreme), medtem pa se v DAP lahko izvršujejo operacije, ki ne potrebujejo funkcijskih vrednosti.

Paralelni simulacijski procesor (PSP) (analogni del hibridnega sistema SIMSTAR) je zgrajen na podlagi dolgoletnih izkušenj podjetja EAI. Uporablja se za simulacijo časovno kritičnih delov simulacijskih modelov (reševanje diferencialnih enačb, kompleksne logične operacije).

Slika 7.36 prikazuje zgradbo paralelnega simulacijskega procesorja.



Slika 7.36: Zgradba paralelnega simulacijskega procesorja

Paralelni simulacijski procesor sestoji iz paralelne logične in paralelne matematične enote. Obe enoti povezujejo komparatorji. Paralelna matematična enota je sestavljena iz do dvesto matematičnih blokov, ki izvršujejo operacije analognih računalnikov. Vhodi in izhodi matematičnih blokov so interno povezani z matriko povezav blokov. Le-ta s pomočjo simulacijskega programa poveže ustrezne matematične bloke, komparatorje in analogne linije priključenih signalov v simulacijski model. Podobno se povežejo tudi logični elementi na paralelni logični enoti.

Programska oprema

Programska oprema omogoča, da delamo na večprocesorskem sistemu SIMSTAR na nivoju simulacijskega jezika. Celotna programska oprema je sestavljena iz GOULD MPX-32 sistemske programske opreme, ki vsebuje operacijski sistem, editor, simulacijski jezik ACSL za nesprotno testiranje, prevajalnik FORTRAN 77 s potrebnimi knjižnicami ter iz simulacijskega jezika STARTRAN, ki omogoča razvoj in izvajanje modela.

Simulacijski jezik STARTRAN temelji na izredno razširjenem simulacijskem jeziku tipa CSSL - ACSL (Mitchel & Gauthier, 1981). Uporablja koncept prevajanja v jezik FORTRAN. Uporabnik poda model v strukturnih sekcijah INITIAL, DYNAMIC, TERMINAL, dodati pa mora tudi t.i. ciljne direktive, s katerimi pove, na katerem procesorju naj se izvaja določena operacija. Za opis modela je bistvena sekcija DYNAMIC, ki lahko vsebuje do pet sekcij DERIVATIVE. Vsaka sekcija DERIVATIVE ima lahko drugačne krmilne parametre (npr. različne metode integracije). Vendar mora prva sekcija vsebovati podsisteme z najvišjimi frekvencami, zadnja pa podsisteme z najnižjimi frekvencami. Podsisteme, ki vsebujejo nezveznosti in signale iz realnega procesa, je potrebno po možnosti vključiti v prvo sekcijo DERIVATIVE z najvišjo prioriteto.

Bistvo učinkovite simulacije kompleksnih modelov na večprocesorskem sistemu SIMSTAR je ustrezna razdelitev nalog med procesorje. Ker moramo zato že relativno dobro poznati model je priporočljivo problem predhodno simulirati v jeziku ACSL. Za razdelitev dela med procesorji moramo oceniti predvsem:

- maksimalne frekvence podsistemov,
- območje spremenljivk stanja,
- zahteve po natančnosti in
- možnost nastopa nezveznosti.

Na osnovi teh ugotovitev izvedemo na digitalnem aritmetičnem procesorju celotno inicializacijo in dele simulacije, ki niso časovno kritični. Če je potrebno, lahko uporabimo tudi programske funkcijske generatorje. Paralelni aritmetični procesor pa čim bolj izkoristimo za simulacijo podsistemov, ki vsebujejo visoke naravne frekvence, majhne časovne konstante, lastnosti togih sistemov, nezveznosti, kompleksne logične operacije, algebrajske zanke ter zunanje analogne in logične signale (velja torej isto kot za prvo sekcijo DERIVATIVE). V praksi skušamo najprej

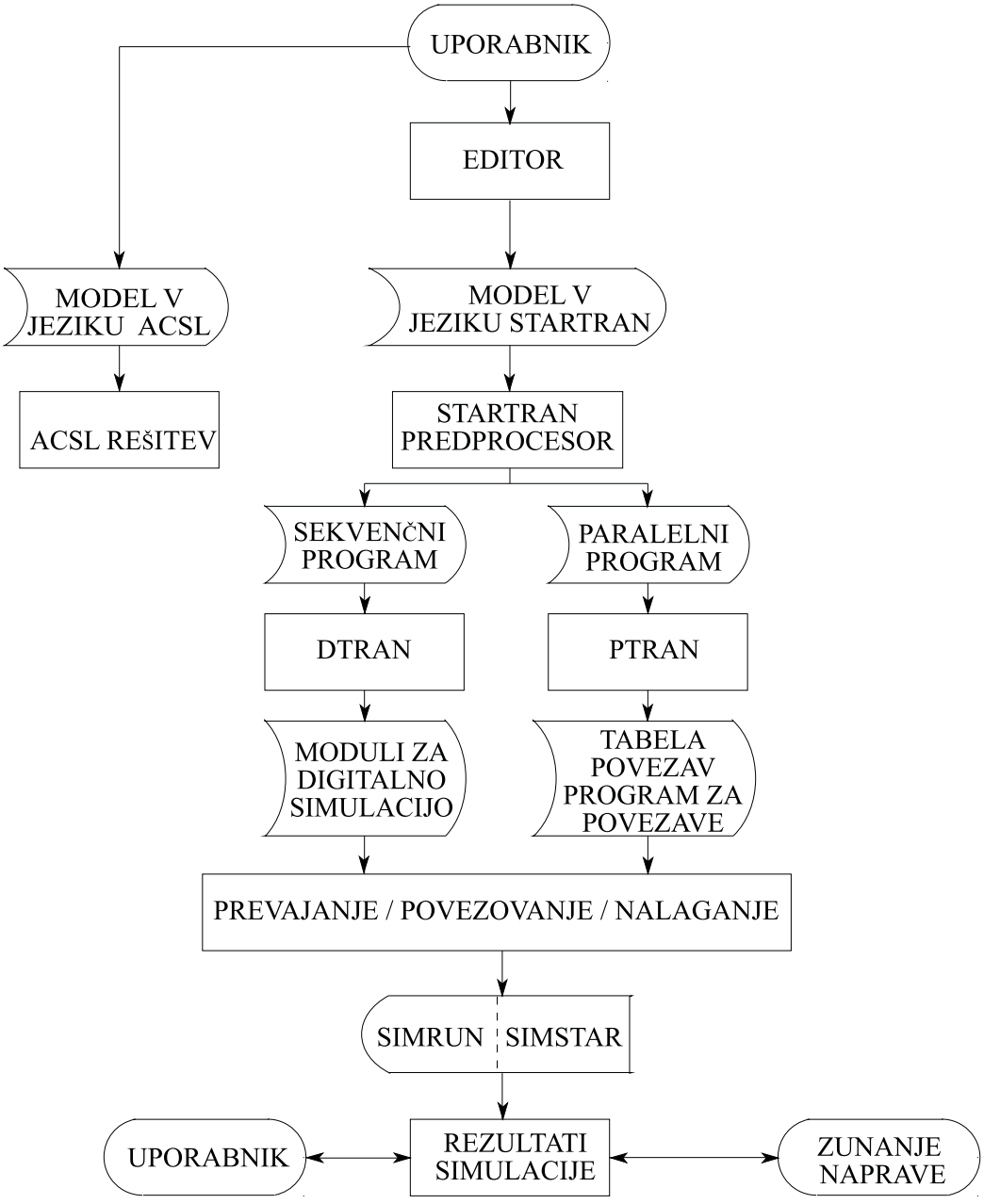
uporabiti elemente paralelnega simulacijskega procesorja in šele nato realiziramo določene operacije na digitalnih procesorjih. Uporabnik z ciljnimi direktivami v simulacijskem programu definira, na katerem procesorju naj se izvršujejo določene operacije.

Slika 7.37 prikazuje procesiranje simulacijskega programa na sistemu SIMSTAR.

Procesiranje se izvaja na digitalnem aritmetičnem procesorju, v primeru, če je v konfiguraciji tudi glavni računalnik, pa na njem. Uporabnik s pomočjo editorja opiše model. Model je možno procesirati z jezikoma STARTRAN ali v ACSL (v fazi razvoja). V primeru uporabe jezika ACSL ciljne direktive nimajo pomena. Predprocesor STARTRAN na osnovi ciljnih direktiv generira sekvenčni program za digitalni del in paralelni program (določene tabele) za paralelni del. Sekvenčni program obdela procesor DTRAN, ki predstavlja pravzaprav prirejeni jezik ACSL za delo v realnem času in zgradi ustrezne module (v jeziku FORTRAN), ki omogočajo digitalni del simulacije. Paralelni program pa obdela procesor PTRAN, ki minimizira enačbe, določi območja spremenljivk in koeficientov in izbere ustrezne matematične bloke. PTRAN generira tabelo povezav in program (v jeziku FORTRAN) za ustrezno povezovanje in nastavitve. Po prevajanju in povezovanju se v procesorje naložijo ustrezni simulacijski programi. Nadzor med simulacijo ima interpreter SIMRUN.

Pri analiziranju modela je vedno možno izvesti simulacijo v jeziku ACSL in tako uporabnik lahko verificira rezultate.

□



Slika 7.37: Procesiranje v jeziku STARTRAN

Literatura

Aburdene, M.F. (1988), *Digital Continuous System Simulation*, Wm.C. Brown Publishers, Dubuque, Iowa, USA.

Annino, J. S., E.C. Russel (1979), "The ten most frequent causes of simulation analysis failure - and how to avoid them!" *Simulation*, 137-140.

Araki, M. (1985), *Industrial applications in Japan of computer aided design packages for control systems.* Preprints of the 3 IFAC/IFIP International symposium CADCE '85, Copenhagen, Denmark, 71-76.

Åström, K.J. (1985a), *A Simnon tutorial*. Department of Automatic Control, Lund Institute of Technology, CODEN: LUTFD2/(TFRT-3168)/ (1982), Sweden.

Åström, K.J. (1985b), "Computer aided tools for control system design", in M.J.Jamshidi and C.J. Herget (eds.), *Computer-aided control system engineering*, North-Holland, Amsterdam, Netherlands, 3-40.

Atherton, D.P. (1986), *Simulation in control system design.* Preprints of the IFAC/IMACS International symposium on simulation of control systems, Wien, Austria, 53-59.

Baker, N.J.C., P.J. Smart (1983), "The SYSMOD simulation language". *Proceedings of the* 2^{st} *European simulation congress*, Aachen, W. Germany, 281-286.

Baker, N.J.C., P.J. Smart (1985), "Elements of the SYSMOD simulation language". *Proceedings of the* 11^{th} *IMACS world congress*, Oslo, Norwege, vol.2.

Barker, H.A., P. Townsend, M. Chen, J. Harvey (1988b), "CES - a workstation environment for computer aided design in control systems". *Preprints of the* 4^{th} *IFAC Symposium on Computer aided design in control systems CADCS '88*, China, Beijing, 248-251.

Bausch - Gall, I. (1987), "Continuous system simulation languages". *Veh. Syst. Dyn. (Netherlands)*, vol.16, 347-366.

Bekey, G.A. and W.J. Karplus (1968). *" Hybrid Computation"*, John Wiley, New York, USA.

Blum, J.J. (1969), *Introduction to Analog Computation*, Harcourt, brace & world, inc., New York, USA.

Boullard, L., L.D. Coen (1985), "A multi - microprocessor system for parallel computing in simulation". *Proceedings of the* 11th *IMACS world congress*, Oslo, Norwege, vol.3, 163-166.

Breitenecker, F., Solar, D. (1986), "Models, methods and experiments - modern aspects of simulation languages". *Proceedings of the* 2nd *European simulation congress*, Antwerpen, Belgium, 195-199.

Bosch, P.P.J. Van den, A.J.W. Van den Boom (1985), "Industrial applications in the Netherlands of computer - aided design packages for control systems; trends and practice". *Preprints of the* 3rd *IFAC/IFIP International symposium CADCE '85*, Copenhagen, Denmark, 58-61.

Bosch, P.P.J. Van den (1987), *Simulation program PSI (manual).*Delft university of technology, Netherlands.

Breitenecker, F. (1989). "Need for Hybrid Simulation?", *Proceedings of the* 3rd *European Simulation Congress*, (D. Murray - Smith, J. Stephenson and R.N. Zobel Eds.), Edinburgh, 421-425.

Britt, J.I, J. S. Wareck, J.A. Smith (1991), "A computer - aided process synthesis and analysis environment".In J.J. Siirola, I.E. Grossmann, G. Stephanopoulos (Eds.), *Foundations of Computer - Aided Process Design*, Elsevier, Amsterdam, 381-307

Bruijn, M. A. de, F.P.J. Soppers (1986), "The Delft parallel processor and software for continuous system simulation". *Proceedings of the* 2nd *European simulation congress*, Antwerpen, Belgium, 777-783.

Butcher, J.C. (1964), "Implicit Runge-Kutta processes", *Math. Comp. 18*, 50.

Carlson A., G. Hannauer, T. Carey , P.J. Holsberg (1967), *Handbook of Analog Computation.* Electronic Associates, Inc. Princeton, N.J., USA.

Cellier, F.E. (1979), *Combined continuous / discrete system simulation by use of digital computers, Techniques and tools.* Ph.D. Swiss Federal Institute of Technology, Zürich, Switzerland.

Cellier, F.E. (1983), "Simulation software - today and tomorrow." *Proceedings of the IMACS conference*, Nantes, France.

Cellier, F.E. (1991), *Continuous System Modeling*, Springer - Verlag, New York, 1991, USA.

Chen, S. (1989), *Toward the future. In Supercomputers: directions and technology and applications*, National Academy Press, Washington D.C.,USA, 51-65

Colijn, A.W., P.D. Ariel (1986), "Continuous system simulation languages on supercomputers". *Proceedings of the 1986 Summer Computer Simulation Conference*, Reno, USA, 3-7.

Crosbie, R.E., F.E. Cellier (1982), "Progres in simulation language standards"; *An activity report for Technical Committee TC3 on simulation software. Proceedings of the* 10^{th} *IMACS world congress*, Montreal, Canada, vol.1. 411-412.

Crosbie, R.E.(1984), "Simulation on microcomputer - experiences with ISIM". *Proceedings of the 1984 Summer Computer Simulation Conference*, Boston, Massachusetts, USA, vol.1, 13-17.

Crosbie, R.E., S. Javey, J.S. Hay, J.G. Pearce (1985a), "ESL - a new continuous system simulation language". *Simulation* , vol.44, no.5, 242-246.

Crosbie, R.E. (1986), "Developments in ESL and other's for the 1990's". *Proceedings of the 1986 Summer Computer Simulation Conference*, USA, 313-314.

Cser J., P.M. Brujn, Verbruggen (1986), "Music - a tool for simulation and real - time control". 4^{th} *IFAC/IFIP symposium on the software for computer control, SOCOCO'86*, Graz, Austria, 58-62.

Dabney J.B., Harman T.L. (2004), *Mastering SIMULINK* , Prentice Hall, Upper Saddle River, N.J., USA.

Dymola (2008), Multi-engineering modeling and simulation, Users manual, ver 7.0. Dessault System, Dynasim AB, Sweden, Lund.

Delebeque, F. and S. Steer (1986), "Some remarks about the design of an interactive CACSD package: The BLAISE experience", in K.L.Lineberry (ed.),*Proc. IEEE* 3^{rd} *Symp. on computer aided control system design*, New York, USA, 48-51.

Hindmarsh, A. C. (1983), "ODEPACK, a systematized collection of ODE solvers", *IMACS Transactions on Scientific Computing*, 1, 55-64.

D'Hollander, E. H. (1985), "A multiprocessor for dynamic simulation". *Proceedings of the 1985 Summer Computer Simulation Conference*, USA, 112-117.

Divjak, S. (1975), *Synthesis of time optimal digital simulation system for continuous dynamical systems*. Ph.D. Faculty of electrical engineering, University of Ljubljana, Yugoslavia.

Duncan, R. (1990), "A survey of parallel computer architectures". *IEEE Computer*, February 1990, 5-16.

EAI Handbook of Analog Computation, (A Carlson, G. Hannauer, T. Carey and P.J. Holsburg Eds.)(1967), Electronic Associates, Inc., West Long Branch, New Jersey, USA.

EAI-580 Analog-Hybrid Computing System - Reference Handbook(1968), West Long Branch, New Jersey, USA.

EAI-2000 Analog Reference Handbook(1982), West Long Branch, New Jersey, USA.

Eckelmann, P. (1987), "The transputer - component for the 5 generation systems". *Proceedings VLSI and computers. First international conference on computer technology, systems and applications*, Hamburg, West Germany, 977.

Elmqvist, H. (1975), *SIMNON, an interactive simulation program for nonlinear systems*. Users manual. Departement of Automatic Control, Lund Institute of Technology, Sweden, Report 7502.

Elmqvist, H. (1977), "SIMNON: an interactive simulation program for nonlinear system". *Proceedings Simulation'77*, Montreux, France.

Elmqvist, H. (1978), *A structured model language for large continuous systems*, Ph. D. diss. Rep. CODEN: LUTFD2/(TFRF-1015), Departement of Automatic Control, Lund Institute of Technology, Sweden.

Elmqvist, H., S.E. Matson (1986), "A simulator for dynamical systems using graphics and equations for modeling". *Proceedings of the IEEE Control System Society. Third symposium on computer aided control system design*, Arlington, USA, 134-139.

Elmqvist, H. (1994), *Dymola - Dynamic Modeling Language*, User's Manual, Dynasim AB, Lund, Sweden.

Fadden E.J (1987), "SYSTEM 100: Time - critical simulation of continuous systems". *Multiprocessor and Array Processor Conference, publishing number 87 02mT15*, San Diego, USA.

Fischlin, A., M. Mansour, M. Rimvall, W. Schaufelberger, (1986), "Simulation and computer aided control system design in engineering education", *IFAC/IMACS International symposium on simulation of control systems*, Wien, Austria, 61-73.

Fishman, G.S. (1973), *"Concepts and methods in discrete event digital simulation"*, John Wiley, New York, USA.

Fritzson P. (2004), *Principles of Object Oriented Modeling and Simulation with Modelica 2.1*, IEEE Press, John Wiley&Sons Inc., Publication, USA.

Gauthier, J.E. (1987), "ACSL and simulators". *Proceedings of the 1987 Summer Computer Simulation Conference*, Orlando, USA, 73-77.

Gear, C.W. (1971), *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice Hall.

Gear, C.W. (1984), "Efficient step size control for output and discontinuities", *Trans. Soc. Comput. Sim. 1*, 27-31.

Gear, C.W. (1986), "The potential for parallelism in ordinary differential equations". Proc. Int. Conf. *Numerical Mathematics*. 33-48.

Gear, C.W., O. Osterby (1984), "Solving ordinary differential equations with discontinuities". *ACM Trans. Math. Software, 10*, 23-44.

Giloi, W.K. (1975), *Principles of Continuous System Simulation* B.G. Teubner, Stuttgart.

Goforth, R.R., R.M. Crisp (1988), "Simulation with microcomputers: an overview". *Acces, a journal of microcomputer applications*, feb. 1988, vol.7, no.1, 4-14.

Golden, D.G. (1985), "Software engineering considerations for the design of simulation languages". *Simulation*, vol.45, no.4, 169-178.

Grierson, W.O. (1986), "Perspectives in simulation hardware and software architecture". *Modeling, Identification and Control (norwegian research bulletin)*, vol.6 ,no. 4, 249-255.

Gustaffson, K. (1988), *Stepsize control in ODE-Solvers-Analysis and Synthesis*, Licentiate Thesis TFRT-3199, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden

Gustaffson, K. (1990), "Using control theory to improve stepsize selection in numerical integration of ODE", *Preprints of* 11^{th} *IFAC World Congress*, Tallin, Estonia, USSR, vol. 10, 139-144.

Hairer, E., C. Lubich (1988), "Extrapolation at stiff differential equations", *Numer. Math.*, Vol. 52, 377-400.

Hairer, E., G. Wanner (1990), *Solving Ordinary Differential Equations II. Stiff and Differential–Algebraic Problems* , Springer Verlag.

Hall, G., J.M. Watt (1976), *Modern numerical methods for ordinary differential equations*, Clarendon Press, Oxford.

Hallin, H.J., S.A.R. Hepner (1984), "Solving benchmark problems with PSCSP and other simulation languages". *Proceedings of the 1984 Summer Computer Simulation Conference*, Boston, Massachusetts, USA, vol.1, 3-8.

Hallin, H.J (1988), "Simulation im Zeitalter von Supercomputern und Mini-Supercomputern". *Proceedings 5. Symposium Simulationstechnik*, Aachen, W. Germany, 2-13.

Hamblen, J.O. (1987), "Parallel continuous system simulation using transputer". *Simulation, vol.49, no.6*, 249-253.

Havranek, W.A. (1983), "Simulation in the 80s". *Proceedings of the 1883 Summer Computer Simulation Conference*, Vencouver, Canada, vol.1, 523-528.

Hay, J.L., R.E. Crosbie (1981), *Outline proposal for a new standard for continuous system simulation language (CSSL 81)*. Computer Simulation Centre, Departement of Electrical Engineering, University of Salford, USA.

Hay, J.L., R.E. Crosbie (1984), "ISIM - a simulation language for microprocessors". *Simulation* , sep. 1984, USA, 133-136.

Hay, J.L., R.E. Crosbie (1986), "Parallel processor simulation with ESL". *Proceedings of the 1986 Summer Computer Simulation Conference*, Reno, USA, 959-964.

Heinrich, R. (1986), "Simulation in control engineering". *Proceedings of the* 2^{nd} *European simulation congress*, Antwerpen, Belgium, 10-13.

Herget, C.J. (1988), "Survey of existing computer-aided design in control systems packages in the United States of America". *Preprints of the* 4^{th} *IFAC Symposium on Computer aided design in control systems CADCS '88*, China, Beijing, 46-50.

Hooper, J.W. (1987), "Strategy- related characteristics of discrete event languages and models". *Simulation*, vol. 46, no. 4, 153-158.

Huber, R.M., A. Guasch (1985), "Towards a specification of a structure for continuous system simulation languages". *Proceedings of the* 11^{th} *IMACS world congress*, Oslo, Norwege, 109-113.

Jackson, A.S. (1960), *Analog Computation*, McGraw Hill, London.

Karba, R., B. Zupančič, F. Bremšak, A. Mrhar and S. Primožič (1990), "Simulation tools in pharmacokinetic modelling", *Acta Pharm. Jugosl.*, Vol. 40, 247-262.

Karplus, W.J. (1984a), "Selection criteria and performance evaluation methods for peripheral array processors". *Simulation*, vol.43, no.3, 125-131.

Karplus, W.J. (1984b), "The changing role of peripheral array processors in continuous systems simulation", *Proceedings of the 1984 Summer Computer Simulation Conference*, Boston, Massachusetts, USA, vol.1, 1-13.

Kettenis, D.L. (1986), "COSMOS: a member of a new generation simulation languages". *Proceedings of the* 2^{nd} *European simulation congress*, Antwerpen, Belgium, 263-269.

Kleinert, W., D. Solar, F. Berger (1983), "Status report on TU Vienna's hybrid time sharing system". *Proceedings of the* 2^{st} *European simulation congress*, Aachen, W. Germany, 193-200.

Kleinert, W., M. Graff, R. Karba, B. Zupančič (1988), "Simulation einer Destillationskolonne - Modellierung mit SIMCOS und Vergleich der Ergebnisse von ACSL und SIMSTAR Simulationen". *Proceedings of the* 5^{th} *symposium Simulationstechnik*, Aachen, W. Germany, 254-259.

Korn, G.A. and J.V. Wait (1978), *Digital Continuous System Simulation*, Prentice Hall, Englewood Clifs, N.J., USA.

Korn, G.A. (1983a), "A wish for simulation - language specifications". *Simulation* ,jan. 1983, USA.

Korn, G.A. (1983b), "Direct - executing languages for interactive simulation and computer - aided experiments." *Proceedings of the* 2nd *European simulation congress*, Aachen, W. Germany, 225-233.

Koskossidis, D.A. and C.J. Brennan (1984), "A Review of Simulation Techniques and Modelling", *Proceedings of the 1984 Summer Computer Simulation Conference (W.D. Wade ed.)*, Vol.1, Boston, USA, 55-58.

Landauer, J, P. (1988), *EAI STARTRAN environment*. Users manual, Electronic Associates, Inc., West Long Branch, New Jersey, USA.

Landauer, J.P. (1988b), "Real-time simulation of the Space Shuttle main engine on the SIMSTAR multiprocessor". *Proceedings of the SCS Multiconference on Aerospace Simulation III*, San Diego, USA.

Lapidus, L., J.H. Seinfeld (1971), *Numerical Solution of Ordinary Differential Equations*, Academic Press New York and London.

Lincoln, A. (1988), A modular parallel computer system for high perfomance real-time simulation. *Proceedings of the* 12th *IMACS world congress*, Paris, France, vol.2., 619-621.

Marquardt, W. (1991), "Dynamic process simulation - recent progress and future challenges." *Preprints of CPC IV, Forth International Conference on Chemical Process Control*, South Padre Island, Texas, USA.

Matko, D., B. Zupančič, R. Karba (1992), *Simulation and Modelling of Continuous Systems - A Case Study Approach.* Prentice Hall.

Matko, D., B. Zupančič, S. Divjak (1989), "New concepts in control system simulation". *Proceeding of the* 3rd *European Simulaton Congress*, Edinburgh, Scotland, 444-446.

$PC - MATLAB^{TM}$ for MS-DOS personal computers, The MathWorks Inc. SouthNatick, USA (1989).

Mc Leod, J.P.E. (1986a), "Computer modelling and simulation: the changing chalenge." *Simulation*, vol.46, no.3, 114-118.

Mc Leod, J.P.E. (1987b), "What is simulation?" (Simulation in the service of society). *Simulation*, 219-221.

Modelica (2007), A Unified Object-Oriented Language for Physical Systems Modeling, Language Specification, Version 3.0, Modelica Association, http://www.modelica.org/documents/ModelicaSpec30.pdf

Mitchel & Gauthier, Assoc. (1981), *ACSL: advanced continuous simulation language.* (user guide / reference manual).

Myers, W. (1990), "Parallel programming: views from the trenches." *IEEE Software, jan. 1990.*

Neelamkavil, F. (1987), *Computer Simulation and Modelling*, John Willey, New York, USA.

Nilsen, N.R. (1984), *The CSSL IV simulation language (reference manual).* Simulation Service, Chatsworth, California, USA.

Nilsen, N.R. (1985), "Recent advances in CSSL IV." *Proceedings of the* 11th *IMACS world congress*, Oslo, Norwege, vol.3, 101-103.

Oblak, S., Škrjanc, I. (2008), *Matlab s Simulinkom, priročnik za laboratorijske vaje* , Univerza v Ljubljani, Založba FE in FRI, Slovenija.

Ogata K. (2010), *Modern Control Engineering*, Fifth edition, Prentice Hall, USA.

Ören, T.I. (1974),"A bibliography of bibliographies on modelling, simulation and programing", *Simulation*, Vol.23, No.3, 90-95 and Vol.23, No.4, 115-116.

Ören, T.I., B.P. Ziegler (1979), "Concepts for advanced simulation methodologies." *Simulation*, vol.32, no.3.

Pearce, J.G., P. Holliday, J.O. Gray (1985), "Survey of parallel processing in simulation." *Proceedings of the* 2nd *European workshop on parallel processing techniques for simulation*, Manchester, United Kingdom, 183-202.

Press, W.H., B.P., Flannery, S.A.,Teukolsky, W.T. Vetterling (1986), *"Numerical recipes - The Art of Scientific Computing"*, Cambridge University Press, UK.

Pritsker, A.A.B (1979), "Comparisons of definitions of simulation." *Simulation* ,vol.33, no.2, 61-63.

Rimvall, M., F.E. Cellier (1985), "The matrix environment as enhancement to modelling and simulation." *Proceedings of the* 11th *IMACS world congress*, Oslo, Norwege, vol.3., 93-96.

Rimvall, M., F.E. Cellier (1986), "Evaluation and perspectives of simulation languages following the CSSL standard." *Modeling, Identification and Control (norwegian research bulletin), vol.6 ,no. 4*, 181-199.

Saucedo R., Schirring E. E. (1986), *Introduction to Continuous and Digital Control Systems*, Mcmillan Applied System Science, New York, USA.

Schmid, C. (1988), "Tehniques and tools of CACDS." *Preprints of the* 4^{th} *IFAC Symposium on Computer aided design in control systems CACDS '88*, China, Beijing, 67-75.

Schmidt, G. (1980), *Simulationstechnik*. R. Oldenbourg, München, W. Germany.

Schmidt, B. (1986), "Classification of simulation software." *Syst. Anal. Model. Simul. (Benelux journal), 3 (1986) 2*, 133-140.

Schmidt, B. (1987), "What does simulation do? Simulation's place in the scientific method." *Syst. Anal. Model. Simul.* (Benelux journal), 4 (1987) 3, 193-211.

Schmidt, A., F. Scheider (1988), "Erfahrungen mit Hardware in-the-loop Simulation an der Workstation XANALOG XA-1000". *Proceedings of the* 5^{th} *symposium Simulationstechnik*, Aachen, W. Germany, 2-13.

Schrage, M.H., D.F. Mc Ardle (1986), " New array processor continuous simulation system uses tactile sensing icon programming." *Proceedings of the 1986 Summer Computer Simulation Conference*, Reno, USA, 138-142.

Schumann A., D. Matko and B. Zupančič (1991), "Simulation of a diesel engine using a digital simulation language," MELECON '91, Ljubljana, Slovenia.

Shah, M., J. (1988), *Engineering simulation: tools and applications using the IBM PC family*. Prentice Hall, Englewood Cliffs, New Jersey, USA.

Shneiderman, B. (1987), *Designing the user interface.* Addison - Wesley Publishing Company, USA.

Simulink, Dynamic System Simulation Software (2009), Users manual, R2009b, Math Works, Inc., Natick, MA, ZDA.

Smith, J.M. (1977), *Mathematical modeling and digital simulation for engineers and scientists.* John Wiley & Sons, Inc.

Smith, D.J.M (1995), *Continuous System Simulation*, Chapman & Hall, London, UK.

Sodja, A, B. Zupančič (2009), *Modelling thermal processes in buildings using an object-oriented approach and Modelica*. Simulation Modelling Practice and Theory, vol. 17, no. 6, str. 1143-1159.

Spriet, J.A. and G.C. Vansteenkiste (1982), *Computer Aided Modelling and Simulation*, Academic Press, London, U.K.

Steppard, S. (1983), "Applying software engineering to simulation." *Simulation*, 13-19.

Strauss, J.C. (1967), "The SCi continuous system simulation language." *Simulation*, no.9, 281-303.

Syn, W.M., H. Dost (1985), "On the dynamic simulation language (DSL/VS) and its use in the IBM corporation." *Proceedings of the* 11th *IMACS world congress*, Oslo, Norwege, vol.3, 115-118.

Šega, M., S. Strmčnik, R.Karba and D.Matko (1985), "Interactive program package ANA for system analysis and control design," *Prep.* 3rd *IFAC int. Symp. CADCE'85*, Copenhagen, 95-98.

Taylor, H., D.K. Friderick, C.M. Rimvall, H.A. Sutherland (1990), "Computer - aided control engineering environments: architecture, user interface, data - base management, and expert aiding." *Preprints of* 11th *IFAC World Congress*, Tallinn, USSR, vol. 10, 55-65.

Terrel, J.T. (1988), *Introduction to digital filters*. Macmillan Education, Houndmills.

Tesler, L.G. (1989), "Achieving a pioneering outlook with supercomputing." *Supercomputers: directions in technology and applications*, National Academy Press, Washington D.C., USA, 90-95.

Tomovic, R. and W.J. Karplus (1962), *High Speed Analog Computers*, John Wiley, New York, USA.

TUTSIM user's manual for IBM PC computers (1983). Twente University of Technology, Netherlands.

Tyso, A. (1985), "Simulation as a tool in operational safety, reliability and control." *Modeling, Identification and Control* (norwegian research bulletin), vol.6 ,no. 3, 127-140.

Worlton, J. (1989), *Existing conditions. In Supercomputers: directions in technology and applications*, National Academy Press, Washington D.C., 21-50

Ziegler, B.P. (1976), *Theory of modelling and simulation.* John Wiley & Sons, Inc., New York, USA.

Ziegler, B.P. (1987), *Simulation objectives: experimantal frames and validity. System & Control Encyklopedia*, Pergamon Press, 4388-4392.

Zupančič, B., D. Matko, M. Šega, P. Tramte (1986), "Simulation in the program package ANA." *Proceedings of the* 2nd *European simulation congress*, Antwerpen, Belgium, 314-318.

Zupančič B. (1989), *Sinteza simulacijskega jezika pri računalniško podprtem načrtovanju sistemov avtomatskega vodenja*, Doktorska disertacija, Fakulteta za elektrotehniko in računalništvo, Univerza v Ljubljani, Ljubljana, Slovenija.

Zupančič, B., D. Matko, R. Karba, M. Atanasijević, Z. Šehić (1991), "Extensions of the simulation language SIMCOS towards continuous - discrete complex experimentation system." *Preprints of the IFAC Symposium CADCS '91* , University of Wales, Swansea, U.K., 351-356.

Zupančič B. (1992), *SIMCOS- jezik za simulacijo zveznih in diskretnih dinamičnih sistemov*, Fakulteta za elektrotehniko in računalništvo, Univerza v Ljubljani, Slovenija.

Učbenik **Simulacija dinamičnih sistemov** je namenjen študentom Avtomatike na Fakulteti za elektrotehniko pri predmetu Simulacije, vendar je po vsebini nekoliko širši in je primeren za vse, ki se kakorkoli ukvarjajo s proučevanjem dinamičnih sistemov. Kajti vsak model realnega procesa je običajno nelinearen, časovno spremenljiv in v takem primeru so analitični postopki neuspešni. Edino simulacija nas pripelje v relativno kratkem času in brez zamotane matematike do zanesljivih in uporabnih rezultatov.
Delo pretežno obravnava zvezno simulacijo. Obdelani sta predvsem indirektna metoda in simulacija prenosnih funkcij kot osnovni simulacijski metodi. Koncept digitalne simulacije pa daje znanja za načrtovanje simulacijskih programov v splošnonamenskih programskih orodjih. Precejšen del je posvečen simulacijskim orodjem, predvsem jeziku po standardu CSSL in okolju Matlab-Simulink. Podrobno pa so obdelani numerični simulacijski postopki. Zadnji del prikazuje zgodovino simulacije, ki se je začela z uporabo analogno hibridnih sistemov.

SIMULACIJA DINAMIČNIH SISTEMOV

Simulacija dinamičnih sistemov, modeliranje dinamičnih sistemov, zvezna simulacija, simulacijski jeziki, numerični simulacijski postopki.

KLJUČNA GESLA

Borut Zupančič je redni profesor na Fakulteti za elektrotehniko Univerze v Ljubljani, predstojnik Laboratorija za modeliranje, simulacijo in vodenje in predstojnik Katedre za sisteme, avtomatiko in kibernetiko. V letih 2004-2007 je bil predsednik evropske federacije EUROSIM, v letih 2010-2013 pa je njen sekretar. Predava predmete iz področja modeliranja, simulacije in vodenja procesov na dodiplomskem in podiplomskem študiju. Raziskovalno se ukvarja z modeliranjem in vodenjem hibridnih sistemov, z več domenskim objektno orientiranim modeliranjem, ter z modeliranjem, simulacijo in vodenjem toplotnih in svetlobnih tokov v stavbah in v drugih termičnih procesih. Za rezultate na področju računalniško podprtega načrtovanja sistemov vodenja prejel nagrado sklada Borisa Kidriča. Je avtor približno 200 domačih in mednarodnih konferenčnih prispevkov ter 50 člankov v revijah.

B. ZUPANČIČ

ISBN 978-961-243-144-0

9 789612 431440

ZALOŽBA FAKULTETE ZA ELEKTROTEHNIKO in FAKULTETE ZA RAČUNALNIŠTVO IN INFORMATIKO