SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

LABORATORIJ I VJEŠTINE - MATLAB

Uvod u SIMULINK

Zagreb, 2006

Sadržaj

1

Simulacija dinamičkih sustava

Ponašanje dinamičkih sustava opisano je skupom diferencijalnih jednadžbi koje su općenito nelinearne. U tim su diferencijalnim jednadžbama sadržane zakonitosti koje vrijede za sustav. Osnovni je problem koji se pojavljuje kod analize takvih sustava nepostojanje opće metodologije za rješavanje nelinearnih diferencijalnih jednadžbi. Kao logično rješenje problema analize ponašanja ovakvih sustava nameće se provođenje simulacije sustava na digitalnom računalu.

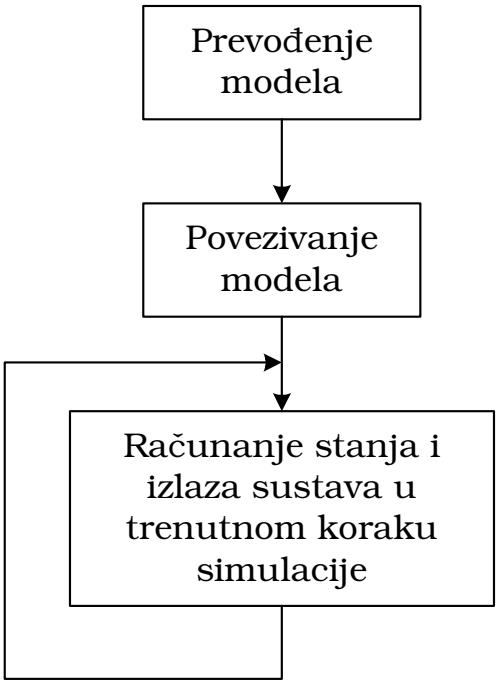
SIMULINK predstavlja grafički alat koji koristi matematičku ljusku Matlaba kako bi se provela simulacija sustava. Izgradnja simulacijskog modela unutar Simulinka obavlja se na jednostavan način korištenjem biblioteke gotovih grafičkih blokova. Osim postojećih blokova korisnik može napisati i vlastite blokove koristeći bilo Matlabove m-funkcije ili funkcije napisane u programskom jeziku C/C++ (S-funkcije).

Simulacijska shema u Simulinku sastoji se od blokova i linija tj. signala kojima se povezuju pojedini blokovi i time realiziraju jednadžbe koje opisuju sustav.

Simulacija unutar Simulinka može se podijeliti tri faze (slika 1.1) i to: (1) prevođenje modela, (2)povezivanje modela i (3) simulacija.

Prevođenje modela (eng. model compilation). U ovoj fazi model se prevodi u izvršni oblik, pri čemu se obavljaju sljedeće operacije:

- izračunavaju se parametri blokova,
- određuju se svojstva signala (dimenzija signala, tip signala)
- provodi se postupak optimizacije s ciljem reduciranja broja blokova u shemi,
- virtualni se blokovi zamjenjuju blokovima koje oni sadrže,
- formira se lista blokova koja se u fazi povezivanja modela koristi kao polazište za određivanje poretka izvršavanja blokova,
- određuju se vremena uzorkovanja (eng. sample time) blokova za koje vrijeme uzorkovanja nije eksplicitno navedeno.



Slika 1.1: Faze provedbe simulacije u simulinku

Povezivanje modela U ovoj se fazi alocira memorija potrebna za spremanje signala, stanja, izlaza i "run-time" parametara simulacije. Nadalje, na temelju liste blokova stvorene u prethodnoj fazi, određuje se najefikasniji poredak izvršavanja blokova kako eventualno ne bi došlo do pojave algebarskih petlji. Pritom se uzima u obzir i eventualna razina prioriteta koju korisnik može dodijeliti pojedinom bloku.

Simulacijska petlja. O ovoj fazi Simulink sukcesivno izračunava stanja i izlaze iz sustava u vremenskim trenucima koji mogu, ali i ne moraju biti ekvidistantni što ovisi o odabranom rješavaču diferencijalnih jednadžbi (eng. solver).

1.1 Kako radi Simulink?

Simulacija dinamičkog sustava se u Simulinku obavlja sukcesivnim računanjem stanja sustava što obavljaju rješavači tj. posebni matematički programi za rješavanje diferencijalnih jednadžbi. Rješavači unutar Simulinka dijele se u dvije kategorije i to: rješavače s konstantnim korakom i rješavače s promjenjivim korakom.

Rješavači s konstantnim korakom. Ovi rješavači računaju stanja sustava u pravilnim vremenskim intervalima. Trajanje simulacije i njezina točnost izravno ovise o izboru koraka diskretizacije. Što je taj korak manji simulacija je točnija, ali se produljuje njezino trajanje.

Rješavači s promjenjivim korakom. Ovi rješavači adaptiraju/mijenjaju korak diskretizacije tijekom simulacije kako bi se zadovoljili zahtjevi koje korisnik postavlja na apsolutni i relativni iznos pogreške simulacije.

Osim ove podjele rješavači se mogu svrstati i prema prirodi sustava koje rješavaju i to na:

- kontinuirane rješavače koji rješavaju problem proračuna stanja kontinuiranih dinamičkih sustava, koristeći postupke numeričke integracije.
- diskretne rješavače, koji zbog prirode diskretnih sustava ne zahtijevaju provođenje numeričke integracije, te su stoga znatno jednostavniji.

Spomenuti postupci numeričke integracije predstavljaju najveći problem koji kontinuirani rješavači trebaju riješiti. Obično se temelje na razvoju funkcije u Taylorov red, čime se unosi određena u pogreška u proračun stanja (ostatak beskonačnog Taylorovog reda). Kao rezultat ovog ovih postupaka diferencijalna se jednadžba nadomješta konačnom jednadžbom diferencija.

Prema formi pripadne jednadžbe diferencija postupci numeričkog rješavanja diferencijalnih jednadžbi dijele se na **eksplicitne** i **implicitne**. Najjednostavniji primjer za eksplicitni postupak je unaprijedna Eulerova metoda:

$$x_{n+1} = x_n + hf(t_n, x_n), \tag{1.1}$$

a za implicitni unazadna :

$$x_{n+1} = x_n + hf(t_{n+1}, x_{n+1}) \tag{1.2}$$

Očito je da se proračun budućeg stanja kod eksplicitnih postupaka može provesti u jednom koraku, dok je kod implicitnih postupaka proračun potrebno obavljati iterativno u više koraka.

2

Simulink

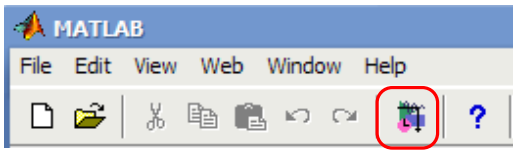
2.1 Osnovne akcije unutar Simulinka

2.1.1 Pokretanje Simulinka

Simulink se pokreće izravno iz Matlaba izvršavanjem naredbe:

```
»simulink;
```

ili klikom miša na ikonicu simulinka u Matlab-ovom osnovnom prozoru (slika 2.1).



Slika 2.1: Ikonica za pokretanje Simulinka

Kao rezultat bilo koje od ovih akcija pokreće se Simulink biblioteka blokova koja sadrži blokove potrebne za izgradnju simulacijskog modela.

2.1.2 Stvaranje novog Simulink modela

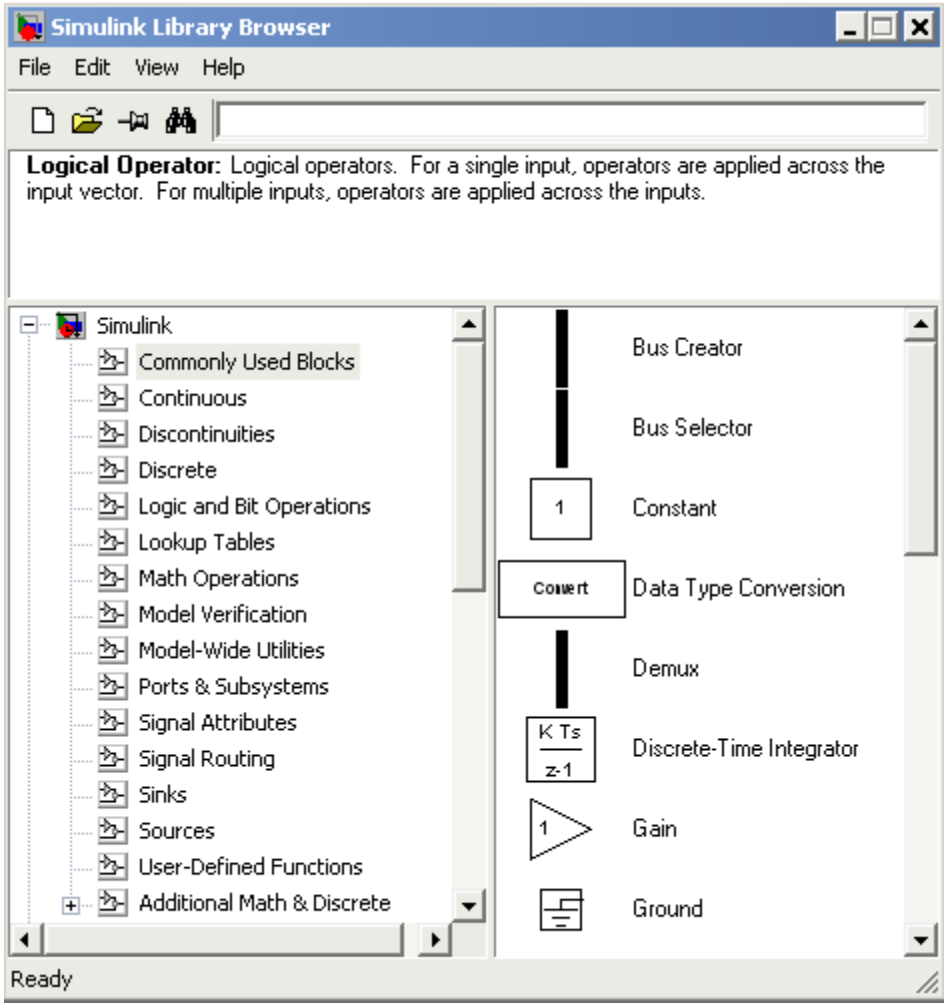
Stvaranje novog Simulink modela obavlja se

1. odabirom opcije `New` unutar `File` izbornika u Simulink biblioteci blokova, ili
2. klikom na odgovarajuću ikonicu.

2.1.3 Otvaranje postojećeg Simulink modela

Otvaranje postojećeg Simulink modela može se obaviti na jedan od sljedećih načina:

1. odabirom opcije `Open` unutar `File` izbornika u Simulink biblioteci blokova ili
2. odabirom opcije `Open` unutar `File` izbornika u Matlabovom osnovnom prozoru ili
3. utipkavanjem naredbe `»ime_modela.mdl` u Matlab komandnom prozoru.



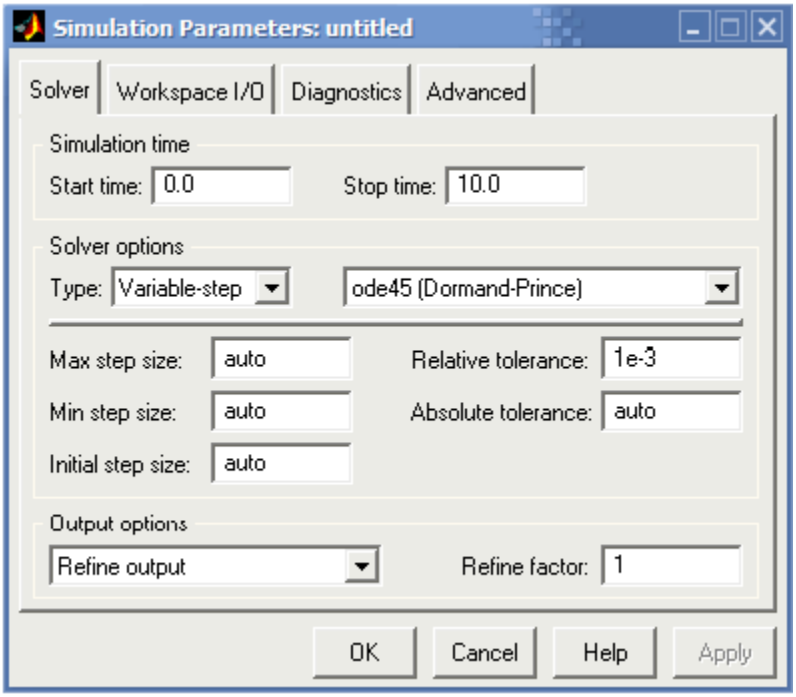
Slika 2.2: Simulink biblioteka blokova

2.1.4 Postavljanje osnovnih parametara simulacije

Osnovne postavke simulacije postavljaju se odabirom opcije "Simulation/configuration". Trajanje simulacije zadaje se izborom početnog (`start time`) i završnog (`stop time`) vremena simulacije. U dijelu "Solver options" određuju se parametri numeričkog postupka. Pritom se može odabrati konstantni odnosno promjenjivi vremenski korak kao i sami numerički postupak koji će biti korišten za simulaciju sustava. U dijelu izbornika `Output option` korisnik može odabrati jednu od opcija:

- Refine output,
- Produce additional output,
- Produce specified output only.

Ovdje je bitno naglasiti da su navedene tri opcije dostupne samo ako je odabran neki od postupaka s promjenjivim vremenskim korakom. Opcija `Refine output` omogućuje koris-



Slika 2.3: Postavljanje parametara simulacije

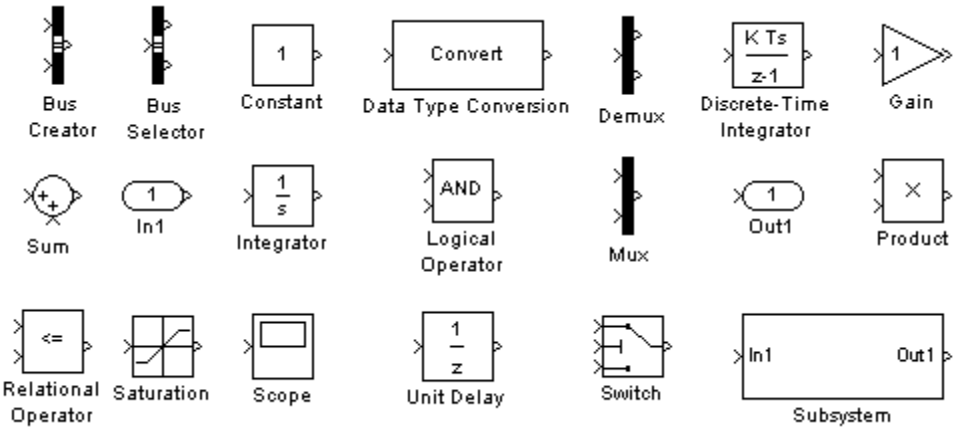
niku povećanje broja točaka u kojima se izračunava stanje sustava za faktor n, čime se dobije bolja rezlučivost stanja koje se izračunava. S druge strane opcija `Dodatni izlazi (Produce additional outputs)` izračunava dodatno stanja sustava u vremenskim trenucima definiranim kao vektor `[t1 t2 t3 ... tn]`. Konačno odabirom opcije `Produce specified output only` stanja sustava se izračunavaju samo u trenucima definiranim vremenskim vektorom, slično kao u prethodnom slučaju.

2.1.5 Simulink biblioteka blokova

Po pokretanju Simulinka otvara se biblioteka blokova koji su korisniku na raspolaganju za izgradnju simulacijske sheme. Ti su blokovi organizirani u skupine s obzirom na tip operacije koje obavljaju. Osim osnovnih Simulink blokova, u biblioteci blokova također se nalaze i dodatni blokovi, ovisno o tome koji su dodatni toolbox-i instalirani. Korisnik može dodati bilo koji od tih blokova u svoju simulacijsku shemu po načelu "Drag'n'Drop", te dodatno postaviti parametre dodanog bloka dvostrukim klikom na njega.

Na slici 2.4 prikazani su najčešće korišteni blokovi, a njihov kratak opis dan je u nastavku.

- `Bus Creator` - Blok za grupiranje više signala u jednu sabirnicu;
- `Bus Selector` - Integracija ulaznog signala $y(t) = \int_0^t u(t)dt$
- `Constant` - Blok na svojem izlazu daje konstantnu vrijednost koja se definira kao parametar bloka.

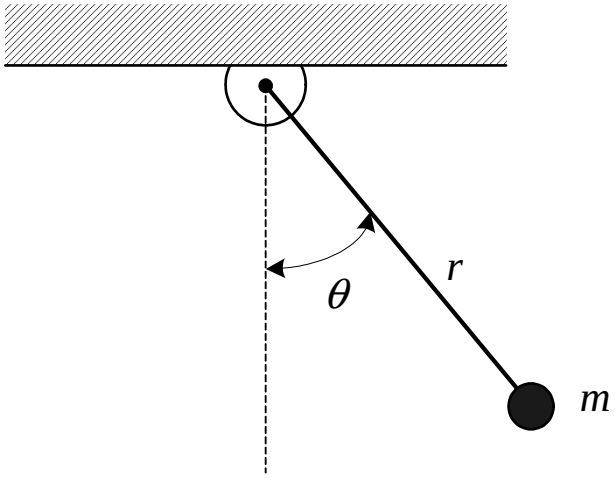


Slika 2.4: Najčešće korišteni blokovi

- `Demux` - Demultipleksor
- `Discrete-Time Integrator` - Diskretni integrator koristeći neku od metoda integracije (unaprijedna Eulerova, unazadna Eulerova ili trapezna integracija).
- `Gain` - Blok pojačanja ($y = K \cdot u$)
- `Sum` - Sumator;
- `In1` - Ulazni port (koristi se kod podsustava);
- `Integrator` - Blok obavlja operaciju integracije u kontinuiranom području ($y(t) = \int_{t_1}^{t} u(t)dt + IC(t_1)$);
- `Logical Operator` - Blok obavlja odabranu logičku operaciju između ulaznih signala;
- `Mux` - Multipleksor;
- `Product` - Blok za množenje/dijeljenje ulaznih signala;
- `Relational Operator` - Blok provjerava odabranu relacija između ulaznih signala te na izlazu daje logičku vrijednost;
- `Saturation` - Blok nelinearne operacije zasićenja;
- `Scope` - Osciloskop (za praćenje razultata simulacije);
- `Unit Delay` - Blok jediničnog kašnjenja ($y(kT) = u((k-1)T)$);
- `Switch` - Blok propušta na izlaz signal doveden na prvi ulaz ukoliko je vrijednost na drugom ulazu zadovoljava odabrani kriterij, a u suprotnom propušta na izlaz signal doveden na treći ulaz;
- `Subsystem` - Podsustav.

2.2 Primjeri korištenja Simulinka za simulaciju ponašanja sustava

Primjer 2.1. ***(Matematičko njihalo)*** *Potrebno je simulirati ponašanje matematičkog njihala prikazano na slici 2.5 pri čemu su dimenzije mase m zanemarive.*



Slika 2.5: Matematičko njihalo

Matematički opis sustava. *Iz fizike je poznato da je gibanje mase m određeno sljedećom diferencijalnom jednadžbom:*

$$J\ddot{\theta} + m \cdot g \cdot r \cdot \sin(\theta) = 0 \tag{2.1}$$

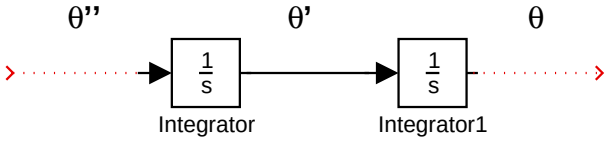
Uz pretpostavku da je cjelokupna masa koncentrirana u kuglici m moment tromosti je $J = m \cdot r^2$*, te slijedi:*

$$\ddot{\theta} + \frac{g}{r}\sin(\theta) = 0 \tag{2.2}$$

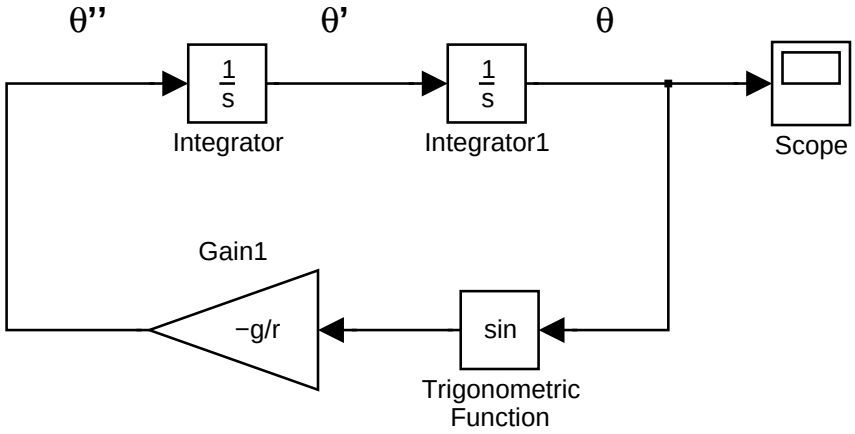
Izgradnja simulacijske sheme. *Prvi korak u izgradnji simulacijskog modela jest zapis diferencijalne jednadžbe* (2.2) *u obliku da na lijevoj strani jednadžbe bude samo derivacija najvećeg reda, a svi ostali članovi na desnoj strani jednadžbe, kako slijedi:*

$$\ddot{\theta} = -\frac{g}{r}\sin(\theta) \tag{2.3}$$

Kod izgradnje simulacijske sheme najprije se pomoću blokova realizira lijeva strana jednadžbe za što su potrebna dva bloka `integrator`*(slika 2.6). U korak se na temelju izraza s desne strane jednadžbe* (2.3) *određuju blokovi potrebni za tvorbu signala koji se spaja na ulaz* $\ddot{\theta}$ *(slika 2.7). Početne vrijednosti stanja sustava općenito se postavljaju tako da se u pojedine integratore u simulacijskoj shemi upišemo njihove početne vrijednosti. Tako u ovom primjeru u lijevi integrator upišemo* 0 *kao početnu vrijednost (početna brzina gibanja njihala) dok u desni integrator upišemo početnu vrijednost kuta* $\theta_0 = \pi/4$.



Slika 2.6: Prvi korak kod izgradnje simulacijske sheme



Slika 2.7: Simulacija shema njihala

Postavljanje parametara i pokretanje simulacije *U ovom primjeri zadržani su inicijalni (defaultni) parametri simulacije (trajanje simulacije, tip rješavača, točnost simulacije,...). Budući da su u simulacijskoj shemi korištene varijable g i r (gravitacija i duljina njihala) potrebno je prije pokretanja simulacije postaviti njihove vrijednosti u Matlabovom komandnom prostoru, kako slijedi:*

```
>>g=10;
>>r=1;
```

Po pokretanju simulacije rezultate možemo pratiti u bloku Scope (Osciloskop) što je prikazano na slici 2.8.

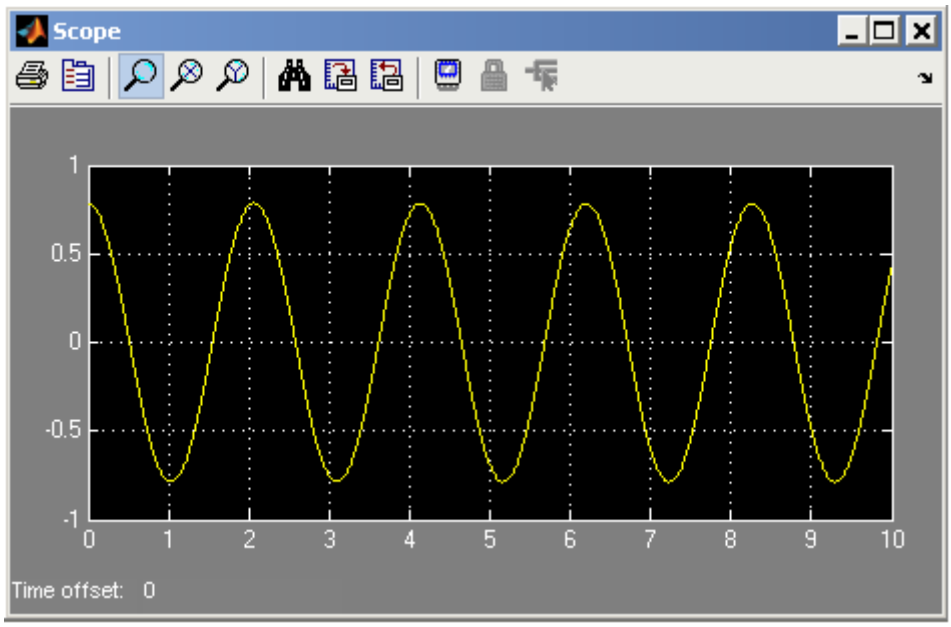
Primjer 2.2. ***(RLC strujni krug)*** *Potrebno je simulirati RLC strujni krug prikazan na slici 2.9.*

Matematički opis modela *Vladanje ovog kruga je opisano je sljedećom diferencijalnom jednadžbom drugog reda:*

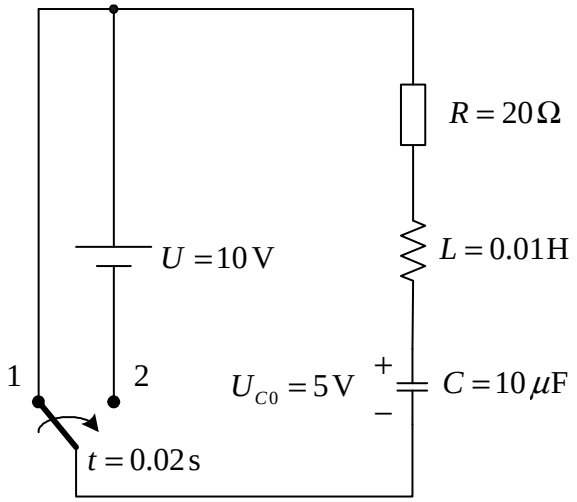
$$u(t) = R \cdot i(t) + L \cdot \frac{di}{dt} + \int_{-\infty}^{t} i(t)dt \tag{2.4}$$

odnosno, ukoliko pređemo na količinu naboja $i(t) = \frac{dq}{dt}$, slijedi:

$$u(t) = R \cdot \frac{dq}{dt} + L \cdot \frac{d^2q}{dt^2} + \frac{1}{C}q(t) \tag{2.5}$$



Slika 2.8: Rezultati simulacije matematičkog njihala uz $r = 1$



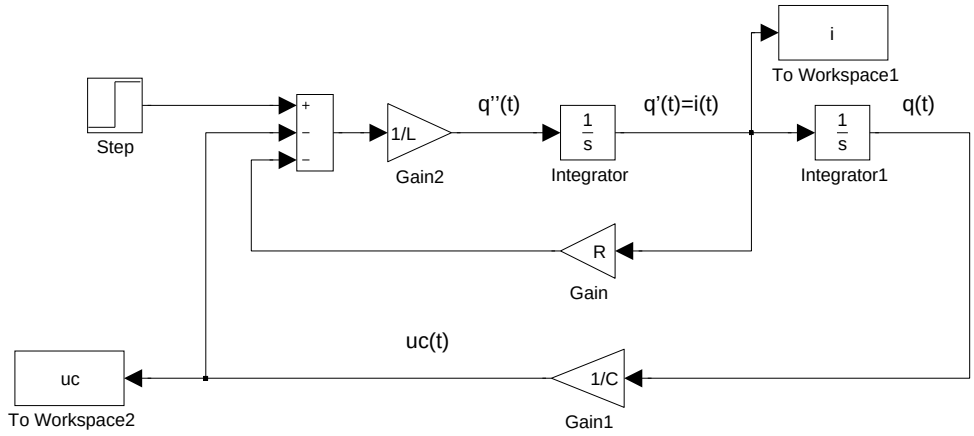
Slika 2.9: RLC krug

Izgradnja simulacijske sheme *Najprije je potrebno diferencijalnu jednadžbu* (2.5) *zapisati u obliku prikladnom za izgradnju simulacijske sheme:*

$$\frac{d^2q}{dt^2} = \frac{1}{L}\left(u(t) - R \cdot \frac{dq}{dt} - \frac{1}{C}q(t)\right) \tag{2.6}$$

Na temelju jednadžbe (2.6) *izgrađuje se simulacijska shema prikazana na slici 2.10.*

Postavljanje parametara i pokretanje simulacije *Sa slike 2.10 je vidljivo da se i u ovom primjeru umjesto upisivanja konkretnih iznosa parametara u simulacijsku koriste*



Slika 2.10: Simulacijska shema RLC kruga

varijable Matlabovog radnog prostora R,L,C i u_{C0}. Vrijednosti ovih parametara potrebno je prije provođenja simulacije unijeti u komandnom prozoru.

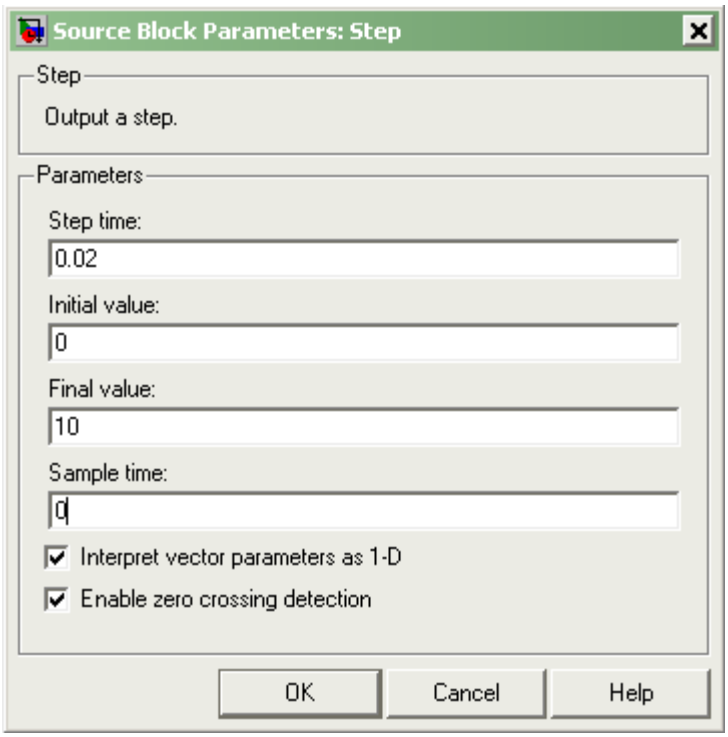
```
>>L=0.01;
>>C=10e-6;
>>R=20;
>>uc0=5;
```

Početnu vrijednost napona na kondenzatoru postavlja se tako da se u desni integrator (čiji je izlaz $q(t)$) upiše početnu vrijednost $q_0 = C \cdot u_0$. Kao ulazni signal postavljen je blok **`Step`** *kojim je ostvarena skokovita promjena ulazne vrijednosti s iznosa* 0 *na iznos* 10 *u trenutku* $t = 0.02s$. *Parametri bloka* **`Step`** *prikazani su na slici 2.11. Za razliku od prethodnog primjera umjesto praćenja signala pomoću bloka* **`Scope`** *(osciloskop), u ovom primjeru se rezultati simulacije spremaju u radni prostor Matlaba (workspace) pomoću bloka* **`To workspace`**.

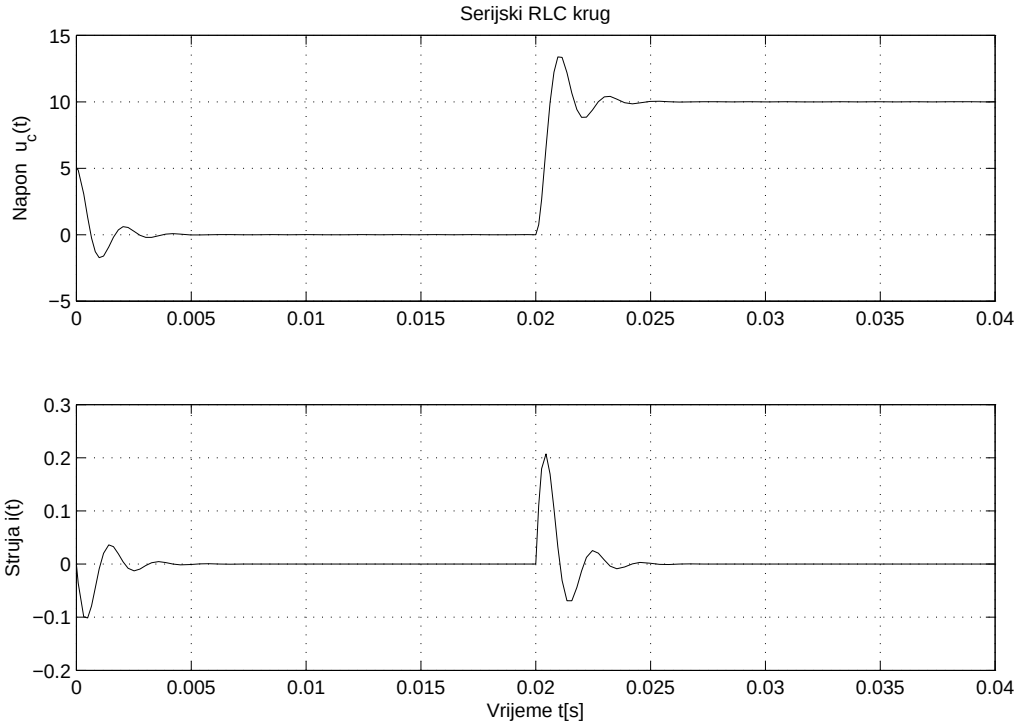
Nakon provođenja simulacije u radnom prostoru Matlaba se nalaze spremljeni vektori uc, i, te tout, koji se automatski generira (vidi postavke simulacije). Sada se rezultati simulacije mogu iscrtati nekom od odgovarajućih Matlab naredbi što je u ovom slučaju napravljeno na sljedeći način:

```
>>subplot(211),plot(tout,uc,'k'),grid;
>>subplot(212),plot(tout,i,'k'),grid;
```

čime su iscrtani odzivi prikazani aa slici 2.12.



Slika 2.11: Parametri bloka `Step`



Slika 2.12: Struja i napon na kondenzatoru

3

Napredne tehnike korištenja Simulinka

U prethodnom je poglavlju opisano osnovno korištenje Simulinka i prikazani su simulacijski postupci dvaju jednostavnih primjera. Međutim, ako se treba simulirati složene dinamičke sustave onda se trebaju koristiti napredne tehnike koje Simulink pruža. U nastavku se najprije opisuje razlaganje složenih sustava na podsustave, a zatim rješavanje problema algebarskih petlji, prolaska signala kroz nulu te simulacija tzv. krutih dinamičkih sustava. Na kraju se daju preporuke za odabir numeričkog postupka rješavanja diferencijalnih jednadžbi te mogućnosti upravljanja simulacijom iz Matlabovog komandnog prozora.

3.1 Podsustavi

Simulink omogućuje modeliranje složenih sustava kao skupa međusobno povezanih podsustava. Općenito unutar Simulinka postoje dva tipa podsustava: **virtualni** (eng. virtual) i **nedjeljivi** (eng. atomic) podsustav. Osnovna je karakteristika virtualnog podsustava da ne utječe na redoslijed proračuna signala, tj. njegovim korištenjem se samo simulacijska shema čini preglednijom. S druge strane nedjeljivi podsustav predstavlja elementarni blok koji se unutar Simulinka izvršava kao jedna cjelina.

Najjednostavniji način stvaranja podsustava je označavanje dijela simulacijske sheme i odabir opcije `Edit/Create Subsystem`. Drugi način je da se u simulacijsku shemu, iz Simulink biblioteke blokova, doda blok `Subsystem` koji inicijalno sadrži samo ulazni i izlazni port, te se u njega dodaju drugi blokovi kako bi se ostvarila željena funkcionalnost podsustava.

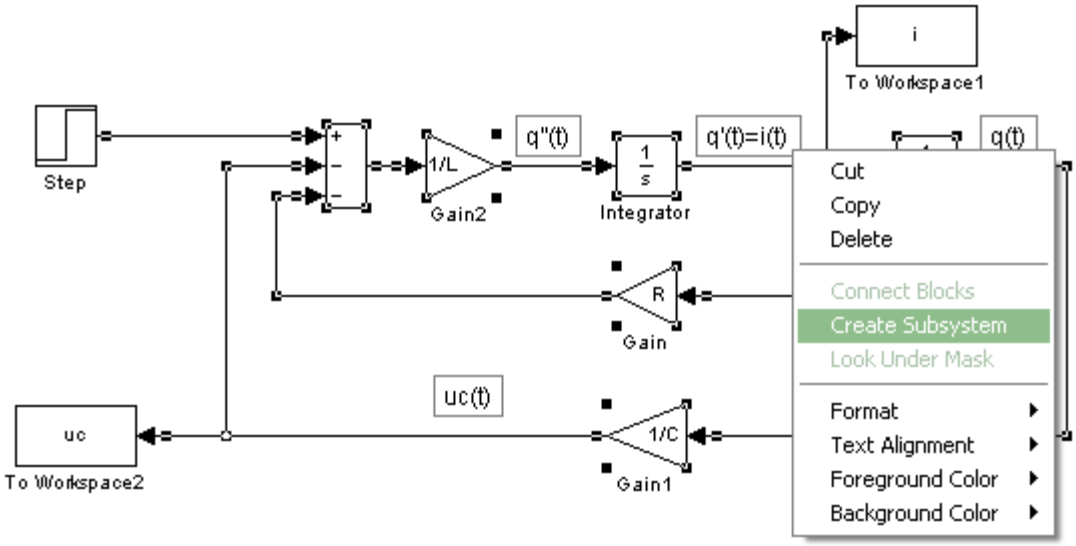
Dodatna pogodnost kod korištenja podsustava je mogućnost njihovog uvjetnog izvršavanja i to kao:

a) podsustav čije je izvršavanje uvjetovano razinom kontrolnog signala (`Enabled Subsystem`);

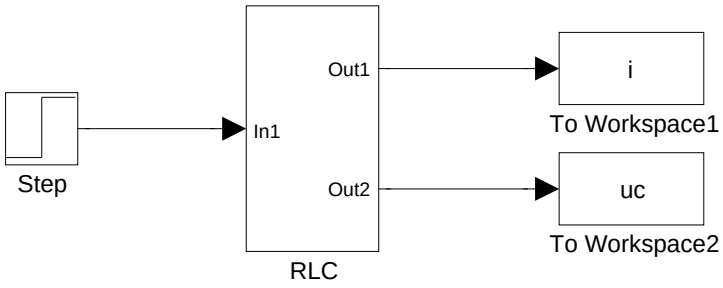
b) podsustav čije je izvršavanje uvjetovano bridom kontrolnog signala (`Triggered Subsystem`);

c) podsustav čije je izvršavanje uvjetovano istovremeno i razinom i bridom kontrolnog signala (`Enabled and Triggered Subsystem`);.

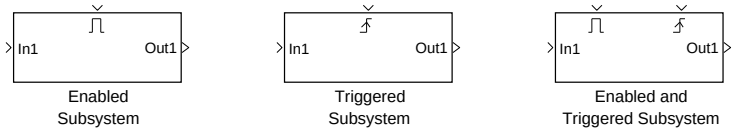
Da bi se podsustav učinio jednim od tri nabrojana tipa potrebno je u njega dodati blok `Enable` i/ili blok `Trigger`. U tom slučaju podsustav ima dodatni ulaz (ili dva ulaza ako se radi o podsustavu pod c)) na koji se dovodi kontrolni signal čije stanje/promjena određuje njegovo eventualno izvršavanje (slika 3.3).



Slika 3.1: Stvaranje podsustava - korak 1



Slika 3.2: Stvaranje podsustava - korak 2

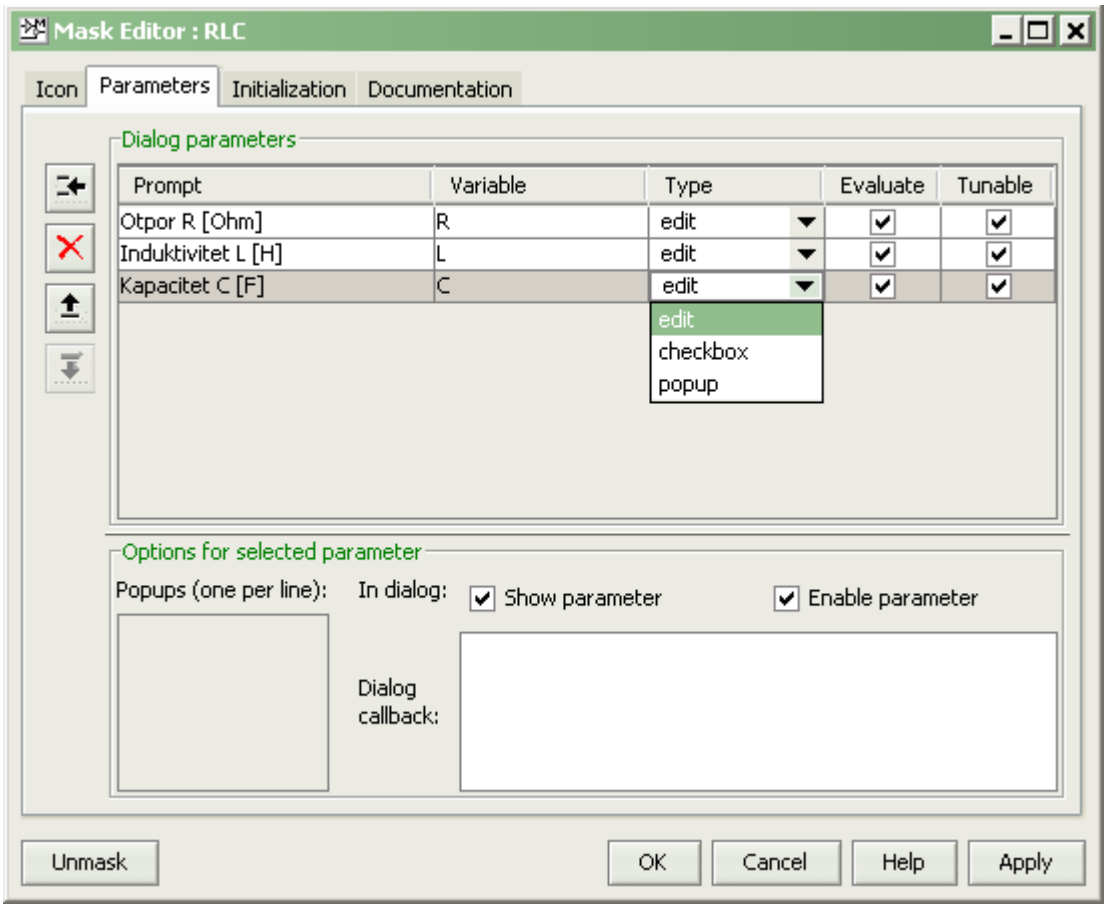


Slika 3.3: Podsustavi s uvjetnim izvršavanjem

3.1.1 Maskiranje podsustava

Nakon što korisnik stvori podsustav željene funkcionalnosti moguće je dodatno ograničiti njegove promjene samog podsustava na način da se stvori maska pomoću koje je moguće mijenjati samo točno određene parametre podsustava. Postojeći se podsustav maskira na način da se najprije označi, a zatim se odabere opcija `Edit/Mask Subsystem`, nakon čega se otvara prozor za definiciju maske.

Pod tabom `Parameters` definiraju se parametri koji unose preko maske (slika 3.4). Nakon definiranja maske, dvostrukim klikom na podsustav otvara se maska za unos parametara



Slika 3.4: Prozor za definiranje maske bloka

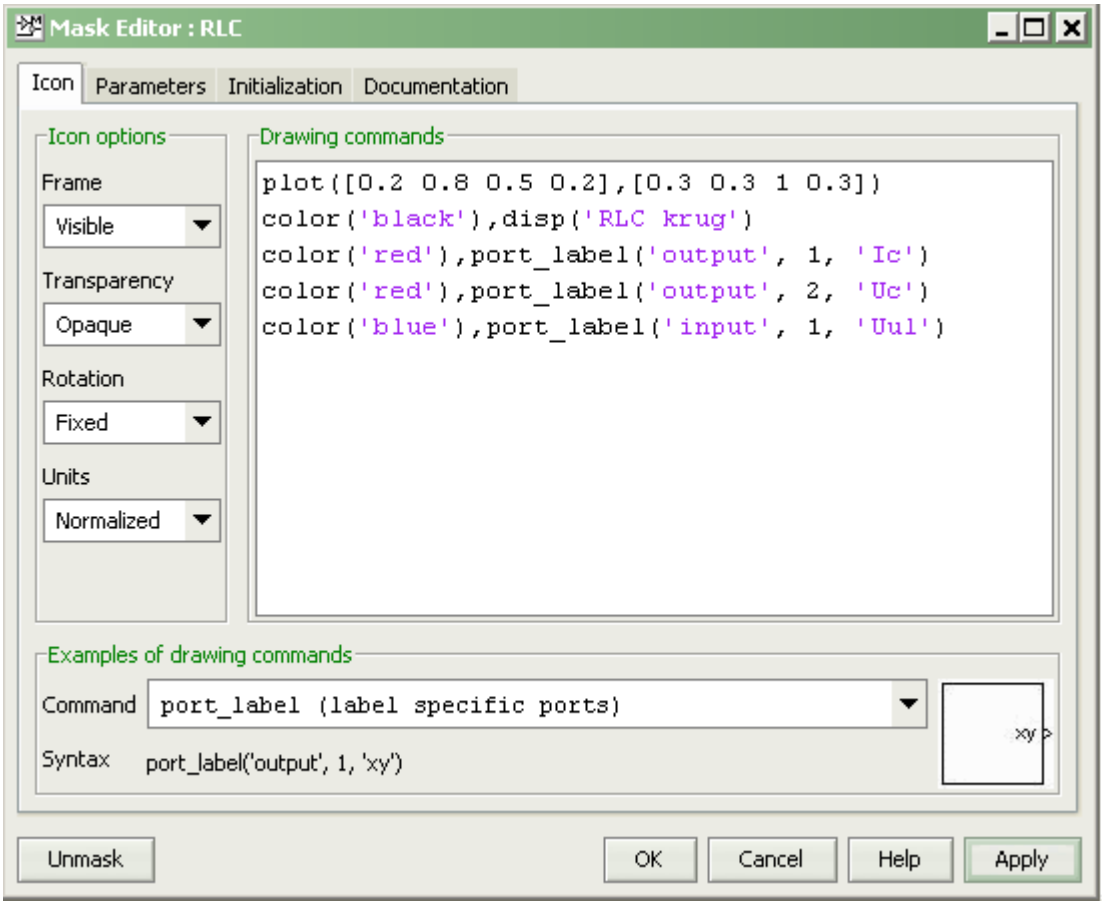
kako je to prikazano na slici 3.7.

Izgled bloka definira se pod tabom `Icon` koristeći skup naredbi koje omogućuju iscrtavanje geometrijskih likova na ikonu bloka kao i definiranje prikaza ulazno/izlaznih portova (slika 3.5).

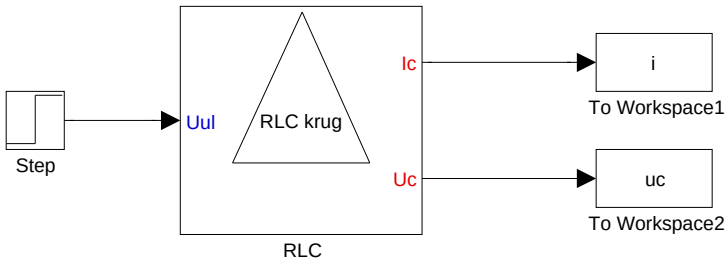
Pod tabom `Initialization` moguće je definirati slijed naredbi koje će se izvršiti kod učitavanja bloka, dok se pod tabom `Documentation` daje opis bloka (Description) i definira pomoć (Help) vezana za taj blok.

3.2 Algebarske petlje

Problem algebarskih petlji može se pojaviti u slučaju korištenja blokova koji imaju svojstvo izravnog prosljeđivanja signala bez kašnjenja. Primjerice takvi su blokovi `Gain`, `Sum` `Transfer Fcn` uz jednak stupanj polinoma u brojniku i nazivniku. Ako postoje petlje (povratne veze) u simulink blokovskoj shemi koje sadrže samo elemente sa svojstvom izravnog prosljeđivanja signala, pojavljuje se problem redoslijeda proračuna signala u toj petlji.



Slika 3.5: Maskiranje podsustava - definiranje ikone bloka

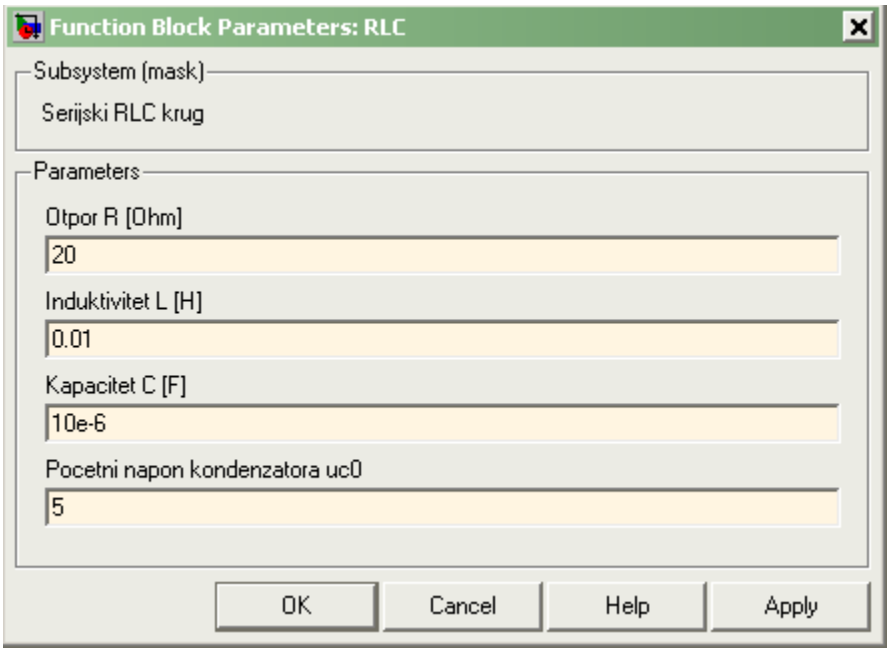


Slika 3.6: Izgled bloka nakon definicije maske

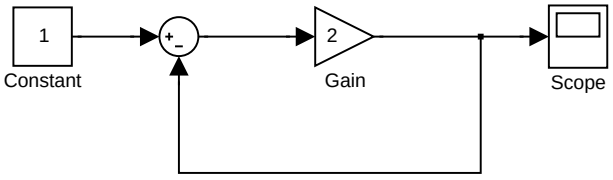
Primjer 3.1. *Razmotrimo jednostavan sustav opisan sljedećim izrazom:*

$$y = 2(u - y). \tag{3.1}$$

Blokovska shema ovog sustava prikazana je na slici 3.8. Očito da je za izračunavanje izlaza iz bloka **Sum***(zbrajalo) prethodno potrebno izračunati signale u i y. No naravno da bi se izračunao signal y prethodno je potrebno izračunati izlaz iz zbrajala, pa je prema tome problem redoslijeda obavljanja proračuna signala očit. Kao najlogičnije rješenje tog problema nameće se transformacija izraza* (3.1) *u oblik* $y = \frac{1}{3}u$*, odnosno analitičko rješavanje algebarske petlje.*



Slika 3.7: Primjer maske bloka



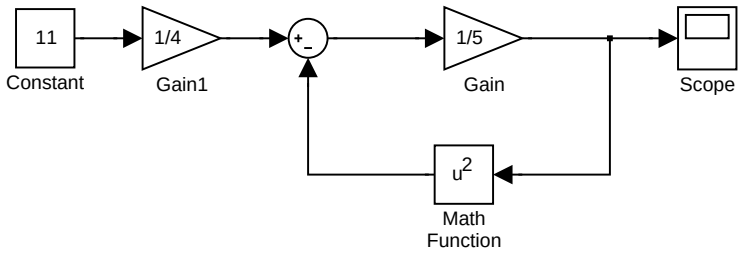
Slika 3.8: Primjer jednostavne algebarske petlje

Prethodni jednostavni primjer, osim ilustracije problema algebarskih petlji, sugerira i najefikasniji način njegova rješavanja, tj. eliminaciju algebarskih petlji njihovim analitičkim rješavanjem. Međutim, često je zbog strukture algebarske petlje (npr. prisutnosti nelinearnih funkcija u petlji) teško ili nemoguće naći analitičko rješenje tog problema. Zbog toga Simulink u slučaju postojanja algebarskih petlji u svakom koraku pokreće poseban postupak za njihovo numeričko rješavanje, ako u postavkama simulacije nije drugačije navedeno. Numerički postupak za rješavanje algebarskih petlji zasniva se na Newton-Raphsonovom postupku rješavanja implicitnih algebarskih jednadžbi oblika $x = f(x)$. Provođenjem ovog numeričkog postupka u svakom koraku dolazi do određenog usporenja simulacije. Ovdje također treba imati na umu da algebarska petlja može općenito imati više rješenja, dok se numeričkim rješavanjem algebarske jednadžbe dobije samo jedno od rješenja, ovisno o početnim uvjetima iz kojih algoritam kreće. Kako bi korisnik ipak mogao utjecati na izbor rješenja algebarske petlje potrebno je u petlju dodati blok početnih uvjeta `IC`.

Primjer 3.2. *Kao ilustraciju spomenutog problema razmotrimo nešto složeniji sustav koji*

sadrži algebarsku petlju:

$$y = 5\left(\frac{1}{4}u - y^2\right) \tag{3.2}$$

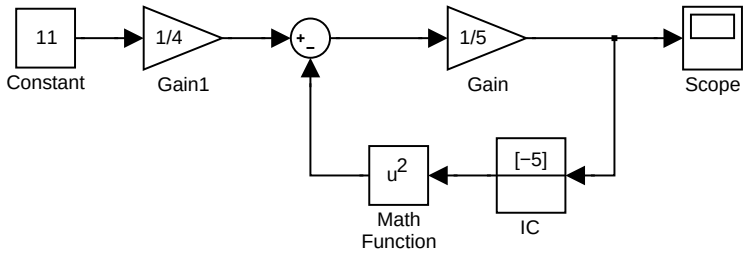


Slika 3.9: Primjer nešto složenije algebarske petlje

Vidljivo je da postoje dva rješenje algebarske jednadžbe i to:

$$y_{1,2} = \frac{-5 \pm \sqrt{25+u}}{2} \tag{3.3}$$

Ovisno o iznosu početnih vrijednosti algoritam će konvergirati jednom odnosno drugom rješenju. Kako bi ipak korisnik mogao utjecati na izbor rješenja u simulacijsku se shemu dodaje blok početnih uvjeta `IC` *(slika 3.10), čime se definiraju početni uvjeti iz kojih kreće algoritam. Ako taj blok nije dodan algoritam pretpostavlja da su početni uvjeti jednaki nuli (slika 3.9).*



Slika 3.10: Primjer nešto složenije algebarske petlje uz dodan blok `IC`

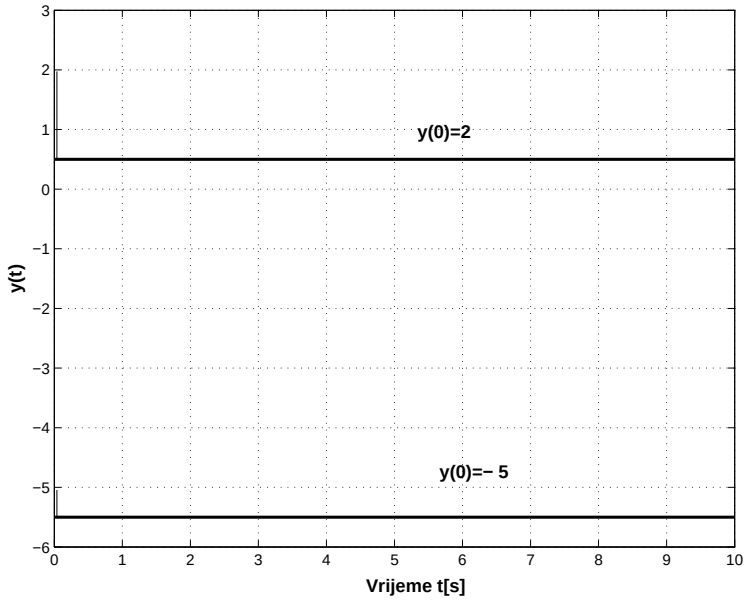
Simulacijom sustava uz ulazni signal oblika $u(t) = 11 \cdot S(t)$*, te uz početne uvjete* $y(0) = 0$ *i* $y(0) = -5$ *dobiju se odzivi prikazani na slici 3.11*

Kao što je ranije spomenuto Simulink na pojavu algebarske petlje reagira na način da:

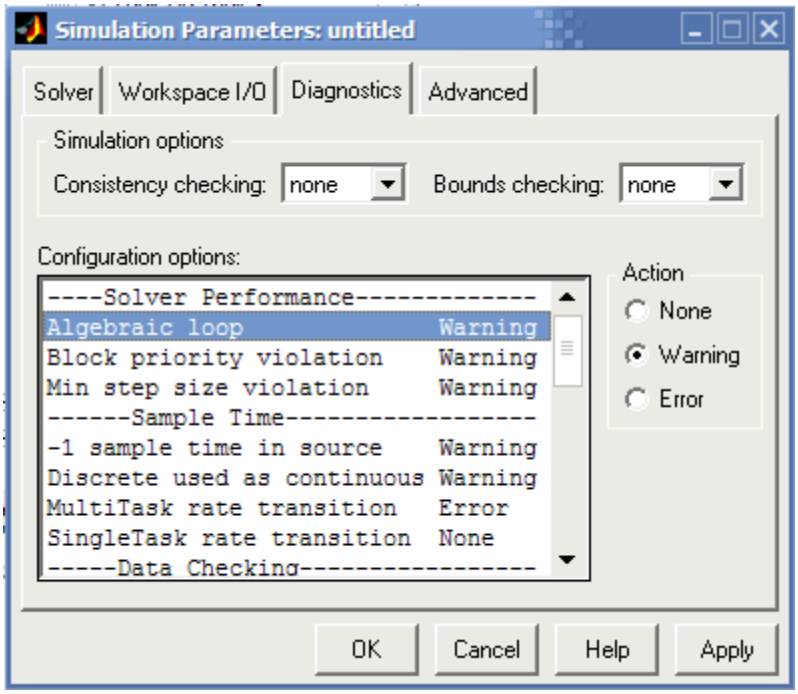
1. u svakom koraku pokrene numerički postupak za rješavanje algebarske petlje (opcije `None` i `Warning` u postavkama simulacija - slika 3.12) ili
2. zaustavi simulaciju (opcija `Error` u postavkama simulacije)

Drugi način rješavanja algebarske petlje je dodavanje memorijskog člana (blok `Memory`), koji unosi kašnjenje u sustav od jednog integracijskog koraka. Ovdje treba imati na umu da je taj način rješavanja problema algebarske petlje uvjetno primjenjiv samo u slučaju da je pojačanje otvorene petlje manje od 1. U suprotnom će sustav postati nestabilan.

Zaključno, ponovimo načine rješavanja problema algebarske petlje:



Slika 3.11: Rezultati simulacija sustava s algebarskom petljom u različite početne uvjete signala $y(t)$



Slika 3.12: Podešavanje reakcije simulinka an pojavu algebarske petlje

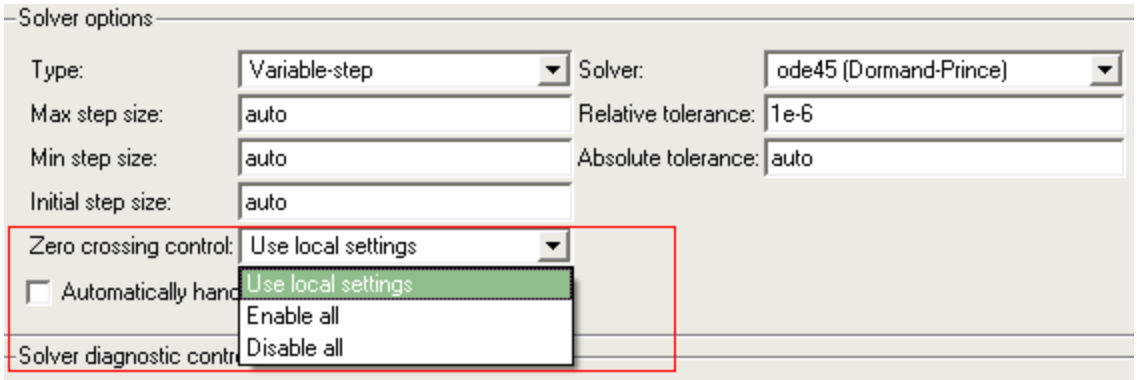
1. Transformacija simulacijske sheme na osnovi rješenja analitičkog algebarske jednadžbe;

2. dodavanje blokova početnih uvjeta (`IC`) u petlju kako bi se utjecalo na smjer traženja rješenja algebarske jednadžbe;

3. Dodavanje memorijskog člana(`Memory`) u petlju (uvjetno primjenjiv).

3.3 Detekcija prolaska kroz nulu

Prolazak stanja kroz nulu često je vezan uz pojavu diskontinuiteta u stanjima sustava, te je u nekim slučajevima od iznimne važnosti upravo odrediti trenutak kada se taj prolazak događa. Tipičan primjer takve pojave je simulacija gibanja kugle na biljarskom stolu. Kod udara kugle o rub stola dolazi do pojave diskontinuiteta u brzini kugle, tj. kugla naglo mijenja smjer gibanja. Očito je da u slučaju korištenja rješavača s konstantnim korakom nema jamstva da će biti pogođen upravo trenutak kada dolazi do pojave diskontinuiteta. Ako se koriste rješavači s promjenjivim korakom tada će zbog pojave diskontinuiteta doći do povećanja lokalne pogreške pa će se vremenski korak smanjivati kako bi se zadovoljile zadane tolerancije na pogrešku što će rezultirati određenim produljenjem trajanja simulacije. No čak i u ovom slučaju nema jamstva da će se pogoditi upravo trenutak pojave diskontinuiteta. Iz toga razloga Simulink koristi posebnu tehniku određivanja točnog trenutka prolaska stanja sustava kroz nulu (eng. Zero Crossing Detection). Tako pojedini blokovi unutar Simulink biblioteke blokova imaju dodatni parametar za odabir detekcije prolaza kroz nulu (parametar `Zero Crossing Detection`). Blokovi koji imaju tu mogućnost su: `Abs`, `Backlash`, `Dead Zone`, `From Workspace`, `Hit Crossing`, `If`, `Integrator`, `MinMax`, `Relay`, `Relational Operator`, `Saturation`, `Sign`, `Signal Builder`, `Subsystem`, `Step`, `Switch`, `Switch Case`. Ako se odabere opcija `Zero Crossing Detection` za neke od tih blokova Simulink će na kraju svakog simulacijskog koraka provjeravati je li došlo do pojave prolaza kroz nulu za njihove signale. Kada se detektira promjena predznaka nekog od signala koji se prati izračunat će se točni trenutak u kojem je došlo do prolaza kroz nulu, te će se nakon toga izračunati vrijednosti svih stanja i signala u tom trenutku.

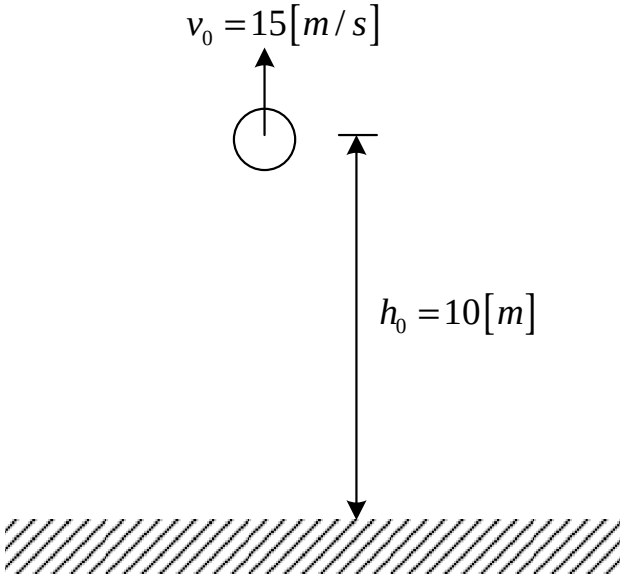


Slika 3.13: Podešavanje parametara simulacije vezano za detekciju prolaza kroz nulu

Osim odgovarajućih postavki za pojedine blokove korisnik ima mogućnost definiranja globalnih postavki simulacije vezanih za detekciju prolaska kroz nulu pomoću opcije `Simulation/ Configuration Parameters` (slika 3.13). Pritom je moguće odabrati jednu od sljedećih opcija:

- `Use local settings` - obavljanje detekcije prolaska kroz nulu određeno je postavkama pojedinih blokova koji imaju tu mogućnost.
- `Enable All` - radi se detekcija prolaza signala za svaki od blokova u simulacijskoj shemi koji imaju tu mogućnost (ignorira se odabir parametra Zero Crossing Detection za pojedine blokove),
- `Disable All` - ne obavlja se detekcija prolaza kroz nulu bez obzira na odgovarajuće postavke pojedinih blokova.

Primjer 3.3. *Kao primjer sustava gdje je neophodno raditi detekciju prolaza kroz nulu modelirat će se sljedeći sustav. Neka se elastična loptica nalazi na visini* $h = 10m$ *i neka ima početnu brzinu u smjeru okomitom na podlogu* $v_0 = 10m/s$*, kako je prikazano na slici 3.14. Kod udara u podlogu loptica gubi* 20% *količine gibanja koju je imala neposredno prije udara.*



Slika 3.14: Elastični udar loptice o podlogu

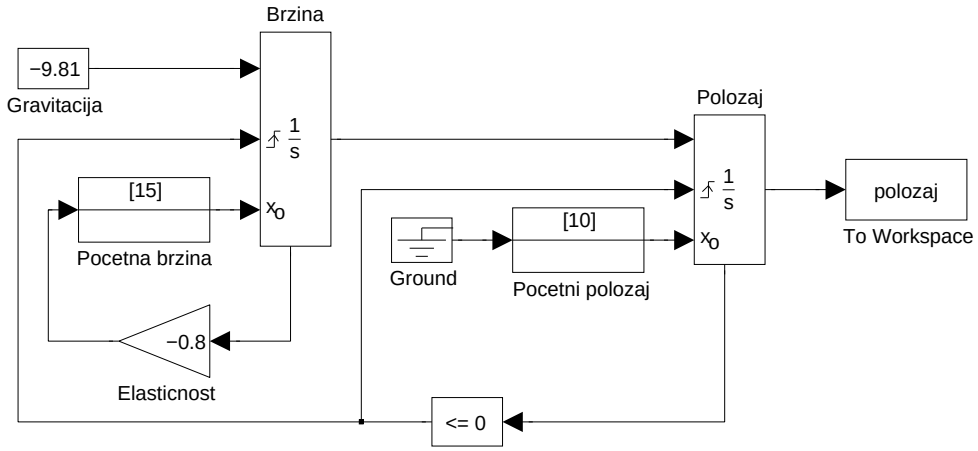
Uz pretpostavku da loptica ima početnu brzinu v_0 *možemo zapisati:*

$$m\dot{v}(t) = -mg \tag{3.4}$$

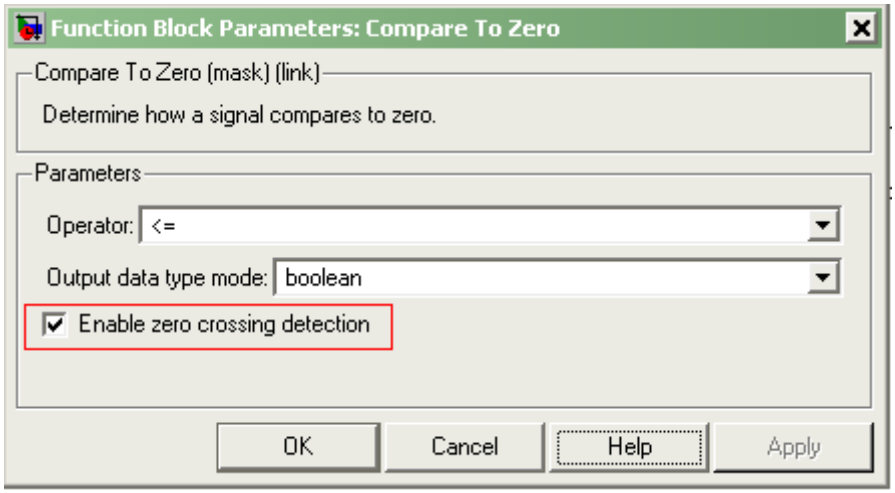
odnosno

$$\dot{v}(t) = -g, \;\; uz\; IC = v_0. \tag{3.5}$$

Bitno je primijetiti da ovaj model vrijedi između dva trenutka udara loptice o podlogu. U trenutka kada se dogodi udar loptice o podlogu vektor brzine loptice mijenja smjer za 180°*, a iznos mu se smanji za* 20%*. Zbog toga je nužno osigurati mogućnost vanjskog reseta za pojedine integratore kako bi se u trenutku kada loptica udari o podlogu u integrator upisala nova vrijednost brzine* $v_0 = -0.8 \cdot v$*, koja predstavlja početnu vrijednost za simulaciju do sljedećeg udara loptice o podlogu.*



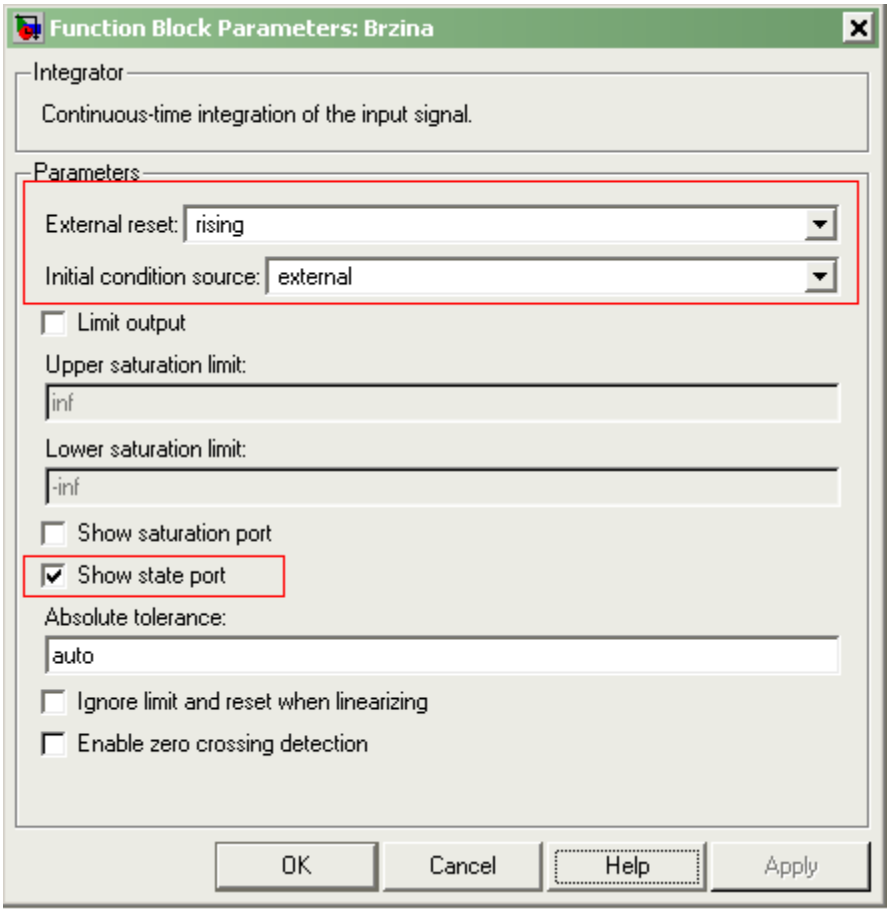
Slika 3.15: Shema simulacije udara loptice o podlogu



Slika 3.16: Parametri bloka usporedbe s nulom - odabir detekcije prolaza kroz nulu

Simulacijska je shema prikazana na slici 3.15. Kako je sam udar loptice o podlogu određen visinom odnosno položajem loptice potrebno je upravo za taj signal raditi detekciju prolaska kroz nulu. Zbog toga se u parametrima bloka `Compare to Zero`*, čiji je ulazni signal položaj loptice, odabire opcija* `Zero Crossing Detection`*(Slika 3.16). Osim toga potrebno je podesiti i parametre bloka* `Integrator` *kako bi se omogućio vanjski reset na pozitivni brid signala iz bloka usporedbe s nulom* `Compare to Zero`*(Slika 3.17). Dodatno je na oba integratora odabrana i opcija* `Show state port` *jer se ti signali (umjesto signala izlaza iz integratora) koriste za detekciju prolaza kroz nulu i upis nove početne vrijednosti. Na taj se način izbjegava mogućnost pojave algebarske petlje u trenutku kada se dogodi prolaz signala kroz nulu, zbog činjenice da je signal sa* `State` *porta bloka* `Integrator` *dostupan ranije nego signal s izlaznog porta bloka.*

U sklopu ovog primjera napravljene su dvije simulacije i to sa i bez bez detekcije prolaska kroz nulu. Za prvu je simulaciju za parametar `Zero Crossing Control` *u konfiguraciji sim-*



Slika 3.17: Parametri bloka Integrator (brzina loptice)

ulacije odabrana opcija `Use Local Setting`*, dok je za drugu simulaciju odabrana opcija* `Disable All`*.*

Radi provjere točnosti provedenih simulacija analitički su izračunati trenutci u kojima dolazi do udara loptice o podlogu:

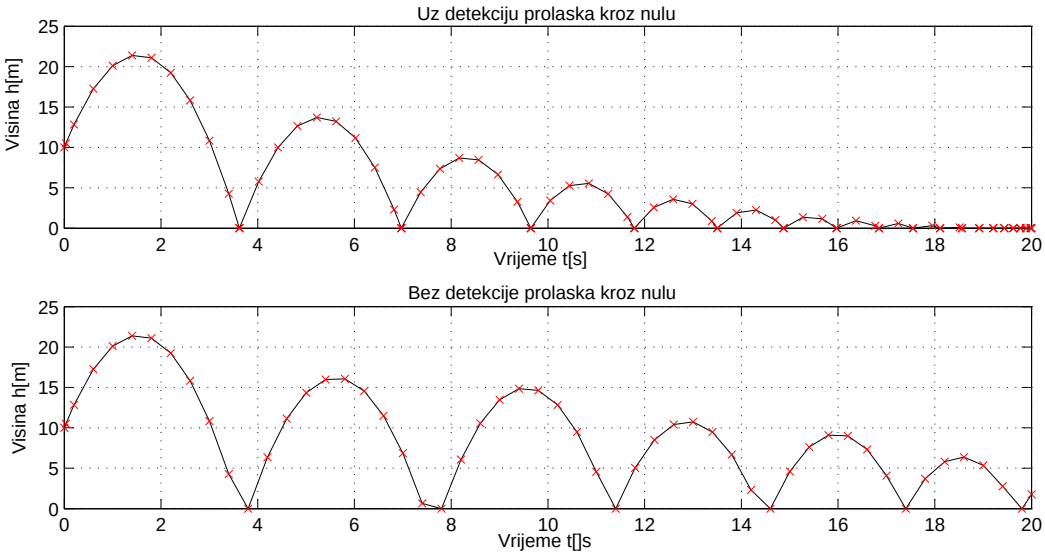
$$t_1 = 3.6211s \quad t_2 = 6.9684 \quad t_3 = 9.6462s \quad ... \tag{3.6}$$

Usporedbom odziva prikazanih na slici 3.18 s vremenima proračunatim analitičkim putem evidentno je da simulacija uz uključenu detekciju prolaska kroz nulu daje zadovoljavajuće rezultate, dok simulacija bez detekcije prolaska kroz nulu ima značajnu pogrešku.

Ovdje je bitno primijetiti i jedan problem koji se pojavljuje kod simulacije uz uključenu detekciju prolaska kroz nulu za ovaj konkretan primjer. Naime ako se pažljivo analiziraju vremena između dva udara loptice o podlogu može se zaključiti da ona čine geometrijski niz pri čemu vrijedi:

$$t_{n+1} = 0.8t_n \tag{3.7}$$

Ovo praktično znači da će, kako simulacija odmiče, biti potrebno sve češće raditi detekciju prolaska kroz nulu, što će rezultirati usporenjem, a na kraju i zaustavljanjem simulacije.



Slika 3.18: Rezultati simulacija a) uz uključenu detekciju prolaska kroz nulu, b) bez detekcije prolaska kroz nulu

3.4 Upravljanje simulacijom iz Matlabovog komandnog prozora

Unutar Matlaba implementiran je skup funkcija koje omogućuju upravljanje simulacijom iz Matlabovog komandnog prozora, te jednako tako i modifikaciju, pa čak i izgradnju samog Simulink modela. Te funkcije omogućuju da se postavljanje parametara simulacije, pokretanje simulacije te obrada samih rezultata simulacije grupiraju u m-skriptu ili m-funkciju. U nastavku se navode najvažnije funkcije koje su korisniku na raspolaganju.

`sim('model',TIMESPAN,OPTIONS,UT);` Ova funkcija pokreće simulaciju modela spremljenog pod imenom 'model'. Parametri funkcije 'TIMESPAN', 'OPTIONS', 'UT' su opcijski i pomoću njih možemo modificirati parametre simulacije. Ako se funkcija pozove bez ovih opcijskih parametara tada se simulacija izvodi uz parametre uz koje je model spremljen.

`set_param('OBJ','PARAMETER1',VALUE1,'PARAMETER2',VALUE2,...)` Funkcija `set_param` postavlja parametre 'PARAMETER1','PARAMETER2',..., objekta 'OBJ' na vrijednosti VALUE1, VALUE2, ...,. Objekt čiji se parametri mijenjaju definira se u obliku 'model/blok'.

`get_param('OBJ','PARAMETER')` Ova funkcija vraća vrijednost parametra 'PARAMETER' objekta 'OBJ'.

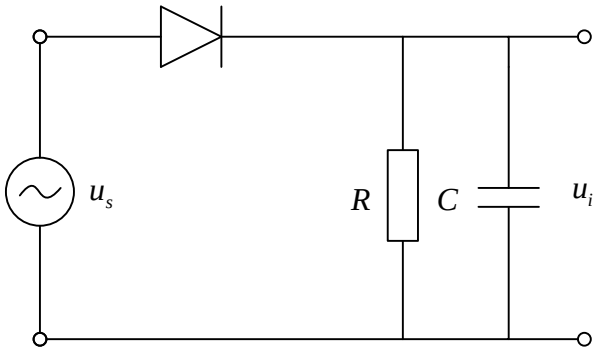
`add_block('SRC','DEST')` Ova funkcija kopira blok čija je staza 'SRC' u novi blok sa stazom 'DEST'.

`add_line('SYS','OPORT','IPORT')` Ovom se funkcijom povezuje izlazni port '`OPORT`' s ulaznim portom '`IPORT`' u simulacijskoj shemi '`SYS`'. Pritom se izlazni i ulazni portovi zadaju u obliku 'blok/port'.

Primjer 3.4. *Na slici 3.19 je prikazan poluvalni ispravljač sa stabilizatorom koji se sastoji od kondenzatora u paraleli s izlaznim otporom. Pritom je pretpostavljeno da je strujno-naponska karakteristika diode idealna i opisana Schocklyevom jednadžbom:*

$$i_d = I_s \left(e^{\frac{u_d}{U_T}} - 1 \right) \tag{3.8}$$

gdje je u_d-napon na diodi, i_d-struja kroz diodu, $U_t = 25.2mV$-termalni napon diode i $I_s = 1.34pA$-struja zasićenja diode. Pritom otpor R iznosi $R = 10k\Omega$, dok je kapacitet kondenzatora C promjenjiv.



Slika 3.19: Shema poluvalnog ispravljača sa stabilizatorom napona

Potrebno je grafički prikazati ovisnost faktora valovitosti (eng. ripple factor) izlaznog napona u_i o iznosi kapaciteta kondenzatora C u intervalu $C \in (1e-7,\ 1e-5)F$.

Faktor valovitosti definiran je kao:

$$\phi = \frac{RMS\{u_{AC}\}}{u_{DC}} \tag{3.9}$$

gdje je:

$$RMS\{u_{AC}\} = \sqrt{\frac{\sum_{i=1}^{N} u_{AC}(i)}{N}} \tag{3.10}$$

Ponašanje ispravljača prikazanog na slici 3.19 opisano je sljedećim izrazima:

$$u_s(s) = u_d(t) + u_i, \tag{3.11}$$

$$i_d(t) = f(u_d(t)), \tag{3.12}$$

$$i_d(t) = \frac{u_i(t)}{R} + C\frac{du_i}{dt}. \tag{3.13}$$

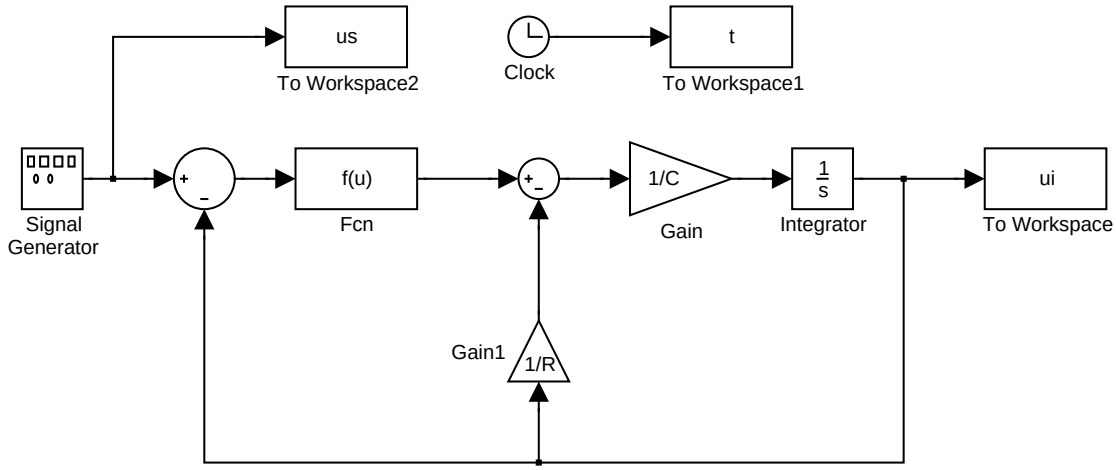
Ove se jednadžbe se zapisuju u obliku prikladnom za izgradnju simulacijske sheme:

$$u_d(s) = u_s(t) - u_i \tag{3.14}$$

$$i_d(t) = f(u_d(t)) \tag{3.15}$$

$$\frac{du_i}{dt} = \frac{1}{C}\left(i_d(t) - \frac{u_i(t)}{R}\right) \tag{3.16}$$

Na temelju jednadžbi (3.14)-(3.14) *izrađuje se simulacijska shema koja je prikazana na slici 3.20. Karakteristika diode opisana je funkcijskim blokom* **`Fcn`** *čiji su parametri prikazani na slici 3.21*

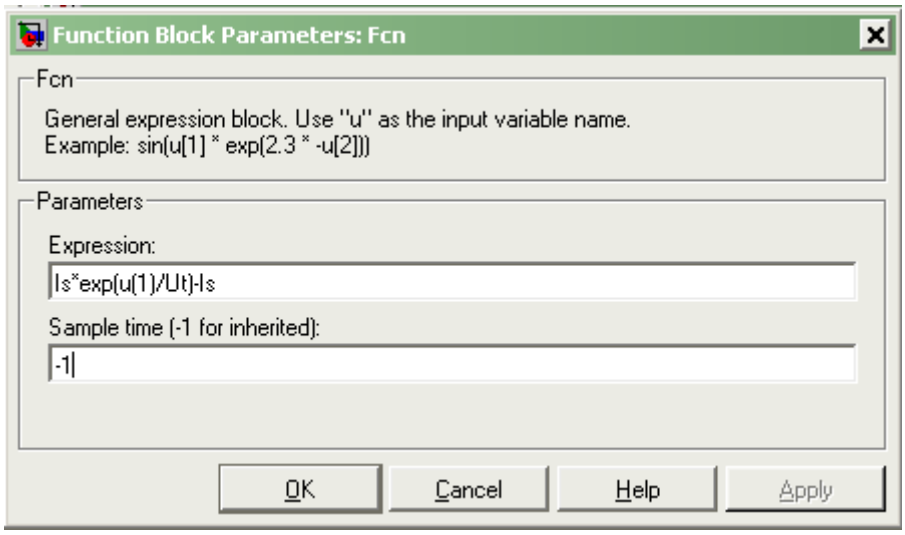


Slika 3.20: Simulacijska shema poluvalnog ispravljača *rect.mdl*

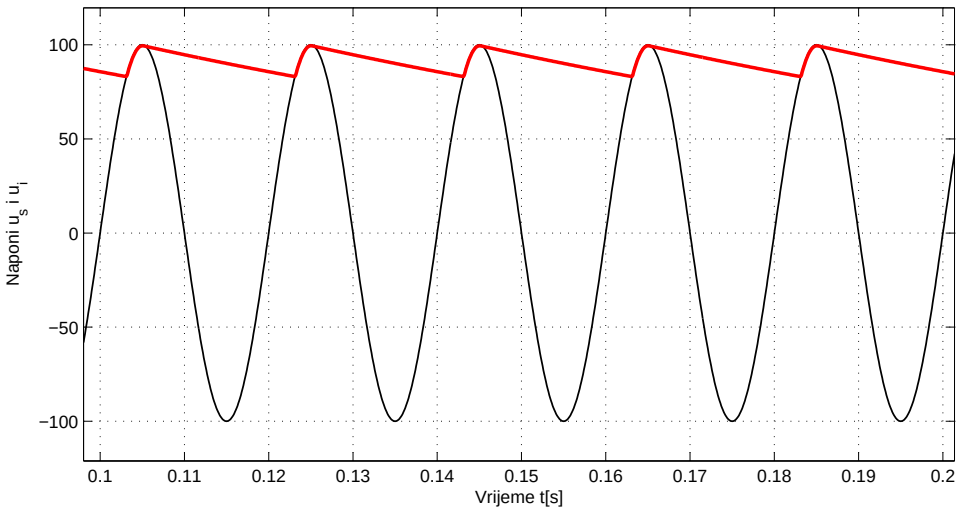
Da bi se dobila grafička ovisnost faktora valovitosti o kapacitetu kondenzatora C potrebno je najprije generirati niz vrijednosti kapaciteta kondenzatora C. Za svaku od generiranih vrijednosti potrebno je provesti simulaciju, te na temelju rezultata simulacije izračunati pripadni faktor valovitosti.

Prijedlog rješenja:

```
clear phi;
Cind=logspace(-7,-4,30); for i=1:length(Cind)
    C=Cind(i);
    sim('rect');
    N=length(ui);
    N1=floor((N-1)/2);
    U_sr=mean(ui(N1:N));
    rms=sqrt(sum((ui(N1:N)-U_sr).^2)/(N-N1));
    phi(i)=rms/U_sr;
end;
semilogx(Cind,phi);
```



Slika 3.21: Parametri bloka `Fcn` koji opisuje U-I karaktristiku diode



Slika 3.22: Usporedni prikaz ulaznog napona u_s i napona na izlazu ispravljača u_i dobiven uz $C = 1e-6\,F$

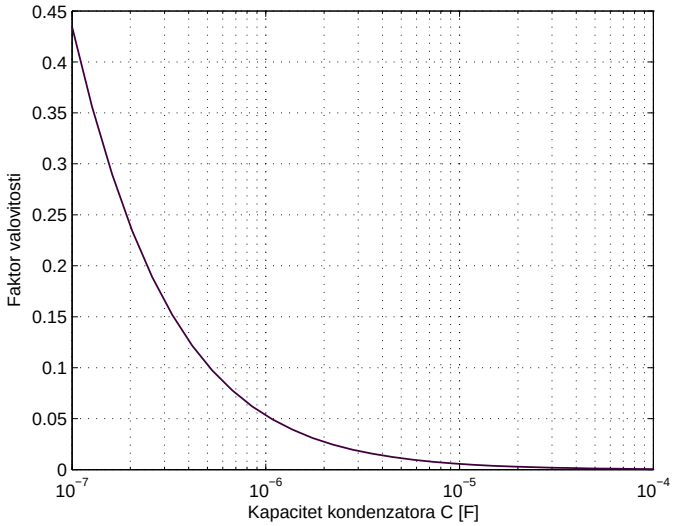
Bitno je primijetiti da se promjena vrijednosti kapaciteta kondenzatora obavljala preko radnog prostora Matlaba (eng. Matlab workspace) naredbom:

```
C=Cind(i);
```

ali se ona također mogla napraviti na način da se izravno promjeni vrijednost upisana u simulacijskoj shemi pomoću naredbe `set_param`*:*

```
set_param('rect/Gain','Gain', mat2str(1/Cind(i)));
```

Nakon izvršavanja prikazane m-skripte dobije se tražena ovisnost prikazana u logaritamskom mjerilu (slika 3.23).



Slika 3.23: Ovisnost faktora valovitosti o iznosi kapaciteta kondenzatora

3.5 Simulacija krutih dinamičkih sustava

Krutim dinamičkim sustavima (eng. stiff systems) nazivaju se sustavi koji posjeduju više po brzini jako različitih dinamika. Ako se radi o linearnom sustavu:

$$\dot{X} = \Lambda \cdot X \tag{3.17}$$

tada se on naziva krutim ako vrijedi $|\Re\{\lambda_{max}\}| >> |\Re\{\lambda_{min}\}|$, pri čemu su λ_{max} i λ_{min} maksimalna i minimalna karakteristična vrijednost matrice Λ.

Problem koji se pojavljuje kod simulacije takvih sustava posljedica je činjenice da stabilnost numeričkih postupaka za rješavanje diferencijalnih jednadžbi ovisi o vremenskom koraku h. Ta je ovisnost posebno izražena kod eksplicitnih numeričkih postupaka, koju ćemo ilustrirati na sljedećem jednostavnom primjeru.

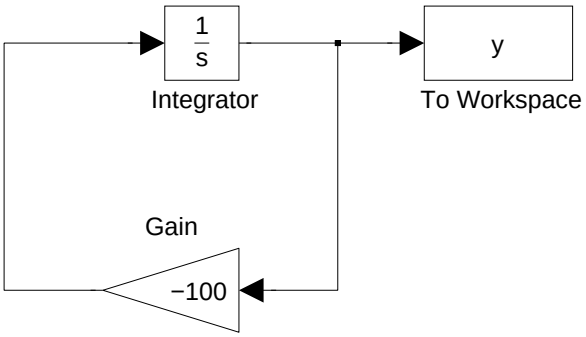
Primjer 3.5. *Neka je dan linearni sustav prvog reda opisan sljedećom diferencijalnom jednadžbom:*

$$\dot{y}(t) + \lambda y(t) = 0 \quad y(0) = y_0, \tag{3.18}$$

pri čemu je $\lambda = 100$ *i* $y_0 = 5$. *Potrebno je usporediti rezultate dobivene ode1(Euler) numeričkim postupkom uz različite iznose vremenskog koraka h sa stvarnim rješenjem diferencijalne jednadžbe* (3.18). *Analitičko rješenje ove diferencijalne jednadžbe je* $y(t) = y_0 \cdot e^{-\lambda t}$, *što je lako potvrditi korištenjem Matlab Symbolic Toolboxa.*

```
>> syms y y0 lambda t
>> sol=dsolve('Dy+lambda*y=0','y(0)=y0')

sol =
```

```
y0*exp(-lambda*t)

>> soln=subs(sol,{y0,lambda},[5,100])

soln =

5*exp(-100*t)
```



Slika 3.24: Simulacija shema sustava opisanog dif. jednadžbom $y'(t) + \lambda y(t) = 0$.

Na temelju diferencijalne jednadžbe (3.18) *izgrađena je simulacijska shema, prikazana na slici 3.24, pri čemu je u postavkama simulacije odabran numerički postupak* `ode1(Euler)` *s konstantnim vremenskim korakom* h*. U sklopu ovog primjera provedeno je 5 simulacija uz različite iznose vremenskog koraka (*$h = 1e-3,\ 5e-3,\ 15e-3,\ 20e-3,\ 21e-3$*) i usporedni rezultati tih simulacija su prikazani na slici 3.25.*

Prikazani rezultati ukazuju da se pogreška numeričkog postupka povećava s iznosom vremenskog koraka h*, da bi za određeni njegov iznos postupak postao nestabilan. Da bi se objasnilo zašto dolazi do efekta nestabilnosti u numeričkom postupku primijenit će se izraz za Eulerovu unaprijednu integraciju na polaznu diferencijalnu jednadžbu. Zapišimo najprije diferencijalnu jednadžbu* (3.18) *u standardnom obliku:*

$$\dot{y} = f(t, y) = -\lambda \cdot y. \tag{3.19}$$

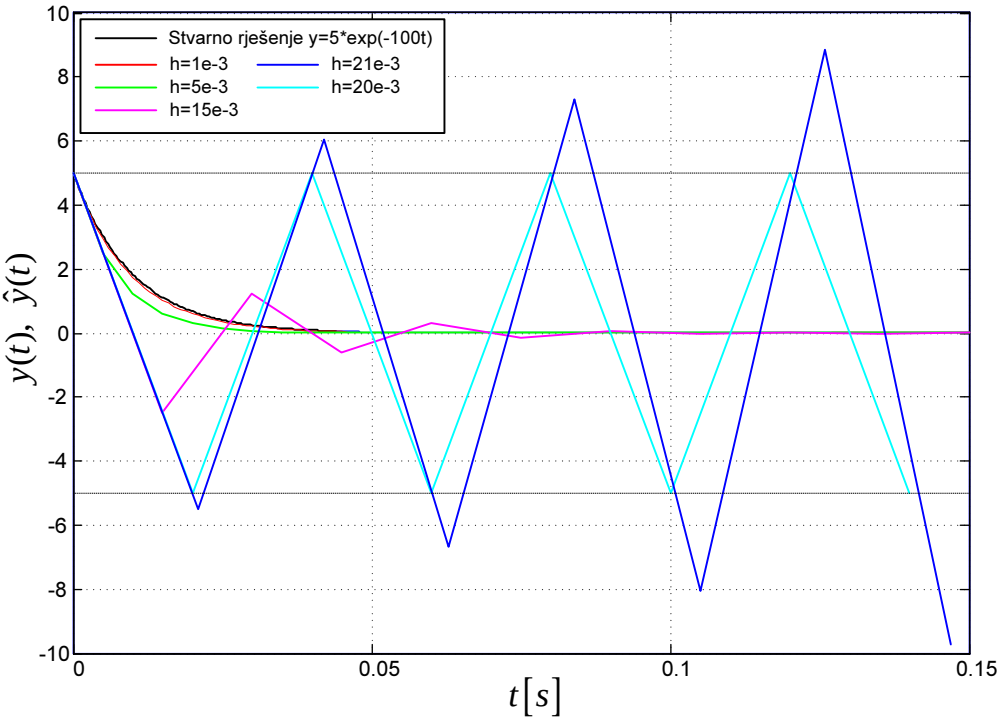
Primjenom izraza za Eulerovu unaprijednu integraciju na ovaj izraz slijedi:

$$y_{n+1} = y_n + hf(t_n, y_n) = y_n - \lambda y_n \tag{3.20}$$

$$y_{n+1} = (1 - \lambda h) y_n \tag{3.21}$$

$$y_n = (1 - \lambda h)^n y_0 \tag{3.22}$$

Predstavlja li dobiveni izraz dobru aproksimaciju rješenja polazne diferencijalne jednadžbe?



Slika 3.25: Rezultati simulacije sustava opisanog dif. jednadžbom $y'(t) + 100y(t) = 0$ korištenjem Eulerovog postupka uz različite iznose vremenskog koraka h

Da bi se odgovorilo na to pitanje potrebno je zapisati dobiveni rezultat u nešto drukčijem obliku:

$$y_n = (1 - \lambda h)^{\frac{-\lambda n h}{-\lambda h}} y_0 \tag{3.23}$$

Ako vrijedi $h \ll \lambda$ *tj.* $h\lambda \to 0$ *tada :*

$$\lim_{\lambda h \to 0} (1 - \lambda h)^{\frac{-1}{\lambda h}} = e \tag{3.24}$$

pa prema tome vrijedi:

$$\hat{y}_n = \hat{y}_0 (1 - \lambda h)^n \approx \hat{y}_0 e^{-\lambda n h} = \hat{y}_0 e^{-\lambda t_n} \tag{3.25}$$

Očito, ako je vremenski korak h *dovoljno malen tada se postiže dobra aproksimacija stvarnog rješenja diferencijalne jednadžbe. Ključnu ulogu za stabilnost postupka u ovom slučaju ima član* $\phi(h) = 1 - \lambda h$*, te ako on zadovoljava uvjet:*

$$|1 - \lambda h| < 1, \tag{3.26}$$

numerički je postupak stabilan. U slučaju da $\lambda \in \mathbb{R}$ *uvjet za stabilnost Eulerovog numeričkog postupka glasi:*

$$h < \frac{2}{\lambda} \tag{3.27}$$

Kako je u ovom konkretnom primjeru $\lambda = 100$ *slijedi da je postupak stabilan ako je* $h \leq 0.02$*, što potvrđuju i rezultati simulacija na slici 3.25.*

Razmotrimo sada slučaj da, na polaznu diferencijalnu jednadžbu umjesto formule za unaprijednu, primijenimo formulu za unazadnu Eulerovu integraciju:

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1}) = y_n - \lambda y_{n+1} \tag{3.28}$$

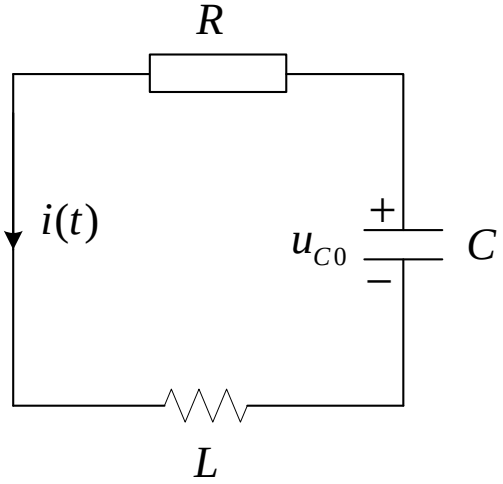
$$y_{n+1} = \frac{y_n}{1 + \lambda h} \tag{3.29}$$

$$y_n = \frac{y_0}{(1 + \lambda h)^n} \tag{3.30}$$

Očito da član koji određuje stabilnost postupka sada glasi $\phi(h) = 1/(1 + \lambda h)$. Budući da se radi o stabilnom dinamičkom sustavu tada vrijedi $\lambda > 0$, pa je u tom slučaju $\phi(h) < 1$ za svaki iznos vremenskog koraka h. Međutim, kod ovog postupka pojavljuje se drugi problem, a to je njegova implicitnost, te je u općem slučaju kod takvih algoritama nužno koristiti iterativne postupke za njihovo rješavanje.

Prethodni je primjer je zorno pokazao da točnost, a kod određenih tipova postupaka i stabilnost, numeričkog postupka izravno ovise o iznosu vremenskog koraka h. Sada pretpostavimo da rješenje diferencijalne jednadžbe ima više komponenata. Radi jednostavnosti neka se radi o zbroju dvije eksponencijalne funkcije tj. $y(t) = C_1 e^{-\lambda_1 t} + C_2 e^{-\lambda_2 t}$, pri čemu vrijedi $\lambda_1 >> \lambda_2, \lambda_1, \lambda_2 > 0$. U tom slučaju maksimalni korak koji se može koristiti u numeričkim postupku određen je bržom komponentom $C_1 e^{-\lambda_1 t}$. S druge strane sporija komponenta $C_2 e^{-\lambda_2 t}$ definira trajanje simulacije. Takva situacija može u konačnici rezultirati dugotrajnom simulacijom. Upravo takav slučaj ilustriran je sljedećim primjerom.

Primjer 3.6. *Razmotrimo primjer jednostavnog linearnog sustava II reda, prikazanog na slici 3.26, (serijski RLC krug) koji za određene iznose parametara R,L i C postaje krut.*



Slika 3.26: RLC strujni krug

Ponašanje RLC kruga opisano je sljedećim izrazom:

$$L\frac{di(t)}{dt} + Ri(t) + \frac{1}{C}\int_0^t i(t)dt = 0 \tag{3.31}$$

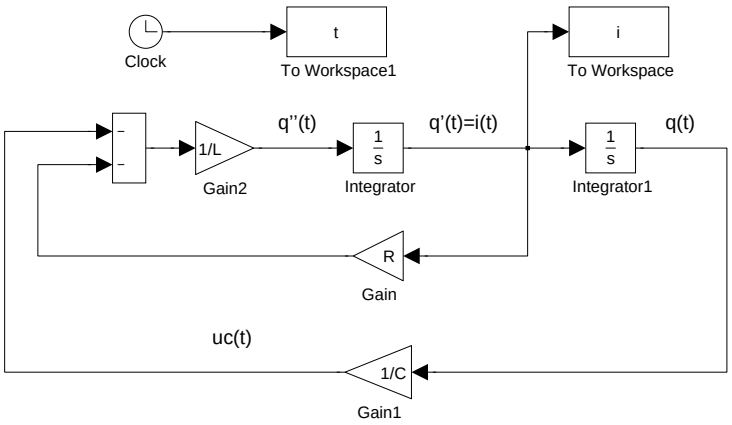
Tablica 3.1: Usporedba trajanja simulacije i broj točaka u kojima se izračunava stanje krutog sustava korištenjem ode45(Dormand-Prince) i ode15(stiff/NDF) numeričkih postupka

Algoritam	ODE45(Dormand-Prince)	ODE15s (Stiff./NDF)
Trajanje simulacije	**0.26 s**	**0.05 s**
Broj točaka	**11315**	**136**

Uvođenjem naboja $q(t) : \frac{dq(t)}{dt} = i(t)$ *slijedi:*

$$L\frac{d^2q(t)}{dt^2} + R\frac{dq(t)}{dt} + \frac{q(t)}{C},\ q(0) = C \cdot u_{c0}, \tag{3.32}$$

gdje u_{C0} *predstavlja napon na kondenzatoru u trenutku* $t = 0$*. Odgovarajuća simulacijska shema, napravljena na temelju izraza* (3.32)*, prikazana je na slici 3.27. Parametri uz koje su obavljene simulacije su sljedeći:* $R = 25\Omega$*,* $C = 200mF$*,* $L = 20mH$ *i* $u_{c0} = 12V$*.*



Slika 3.27: Simulacija shema RLC sustava

U sklopu ovog primjera obavljene su dvije simulacije uz različite numeričke postupke i to: ode45(Dormand-Prince) i ode15s(Stiff./NDF). Na temelju rezultata simulacije napravljena je usporedba trajanja simulacije i broja točaka u kojima se izračunava stanje sustava za pojedine postupke, što je prikazano u tablici 3.1. Iako je ode15s(stiff/NDF) impicitni postupak koji unutar svakog koraka zahtijeva više iteracija kako bi se izračunalo stanje sustava, trajanje simulacije uz taj postupak je približno pet puta kraće nego u slučaju korištenje ode45(Dormand-Prince) postupka. Posebno drastična razlika je u broju točaka u kojima se izračunava stanje sustava, što je izravna posljedica ograničenja maksimalnog vremenskog koraka uvjetima stabilnosti pojedinih postupaka. Ovdje je bitno napomenuti da oba postupka spadaju u skupinu postupaka s promjenjivim vremenskim korakom.

U prethodnom smo primjeru vidjeli da se implicitni numerički postupci (`ode15s`) pokazuju kao bolje rješenje za simulaciju krutih sustava. Budući da se u ovom slučaju ipak radilo o umjereno krutom sustavu, eksplicitni `ode45` postupak je uspio obaviti simulaciju u relativno

Tablica 3.2: Usporedba trajanja simulacije krutog sustava korištenjem različitih postupaka numeričke integracije

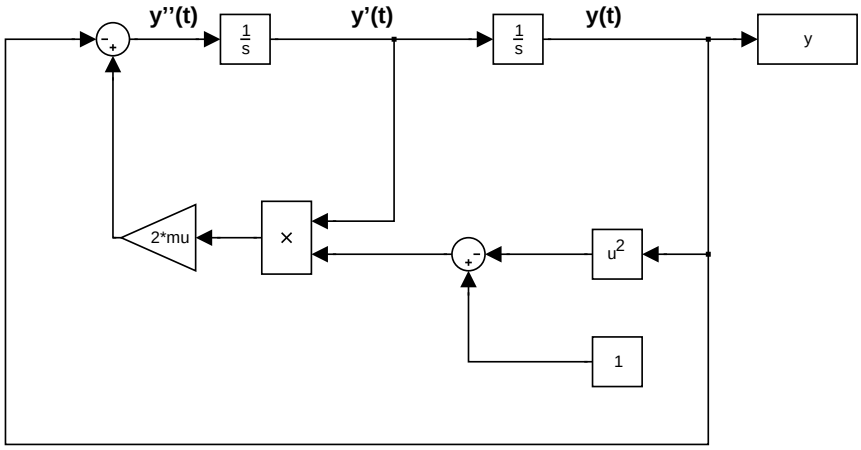
Algoritam	ODE45(Dormand-Prince)	ODE15s (Stiff/NDF)
Trajanje simulacije	**2192 s**	**0.33 s**

kratkom vremenu, tako da nismo bili prisiljeni nužno mijenjati defaultni numerički postupak. Sljedeći primjer izrazito krutog nelinearnog sustava prikazat će situaciju kada je promjena numeričkog postupka nužna.

Primjer 3.7. ***(Van der Polov oscilator)*** *Neka je dan nelinearni sustav, opisan sljedećom diferencijalnom jednadžbom:*

$$\ddot{y}(t) - 2\mu(1 - y^2(t))\dot{y}(t) + y(t) = 0 \tag{3.33}$$

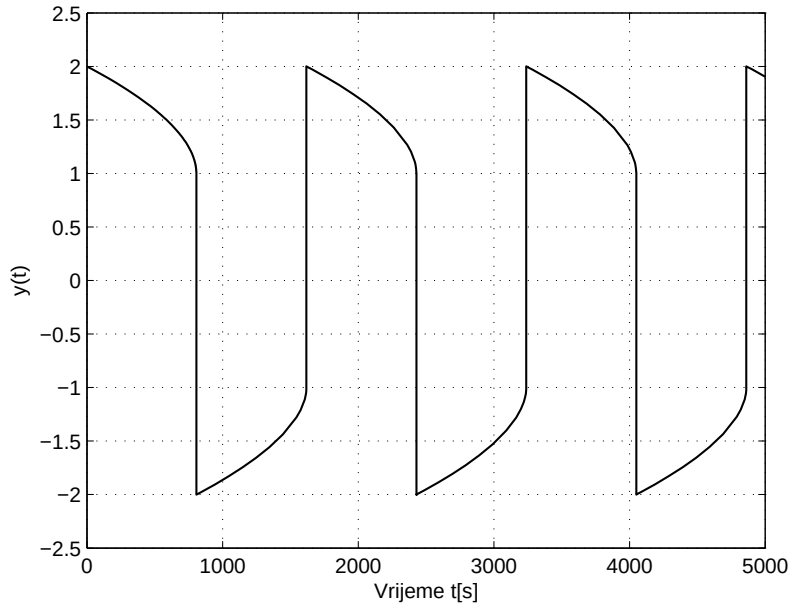
koji je u fizici poznat kao Van der Polov oscilator. Ponašanje tog sustava u znatnoj mjeri ovisi o iznosu parametra μ. Ono što je karakteristično za taj sustav je da postaje izrazito krut za veće iznose parametra μ. Odgovarajuća simulacijska shema Van der Polovog oscilatora prikazana je na slici 3.28. Rezultat simulacije dobiven uz $\mu = 1000$ koristeći ode15s postupak prikazan je na slici 3.29. U tablici 3.2 dana su vremena trajanja simulacije uz pojedine numeričke postupke, odakle zaključujemo da je u slučaju simulacije ovakvih sustava nužno prijeći na neki od postupaka prikladnih za njihovo rješavanje (ode15s,ode23s). Bitno je naglasiti da, na žalost, u općem slučaju ne postoji jednostavna procedura kojom bi se došlo do odgovora je li neki sustav krut.



Slika 3.28: Simulacija shema Van der Polova oscilatora

3.6 Numerički postupci unutar Matlab/Simulink programskog sustava

U ovom potpoglavlju je dan kratak pregled numeričkih postupaka za rješavanje diferencijalnih jednadžbi dostupnih unutar programskog sustava Matlab/Simulink, kao i određeni



Slika 3.29: Simulacija Van der Polova oscilatora uz korištenje ODE15s metode numeričke integracije

iskustveni naputci za njihov odabir.

3.6.1 Numerički postupci s promjenjivim vremenskim korakom

ode45(Dormand-Prince). Ovaj je numerički postupak zasnovan na kombinaciji Runge-Kutta metoda četvrtog i petog reda pri čemu se za proračun novog stanja koristi `RK5` metoda, dok se lokalna pogreška procjenjuje na temelju `RK4` metode. Procijenjena lokalna pogreška koristi se za adaptaciju vremenskog koraka h. Ovaj numerički postupak je defaultni postupak u novijim verzijama Simulinka.

ode23(Bogacki-Shampine). Slično kao `ode45` i ovaj postupak koristi kombinaciju Runge-Kutta metoda drugog i trećeg reda. Pritom se za proračun novog stanja koristi `RK3` metoda, dok se lokalna pogreška procjenjuje na temelju `RK2` metode. Pokazuje se da je ovaj postupak efikasniji od `ode45` metode kod grubljih tolerancija.

ode113(Adams). Ovaj postupak koristi kombinaciju dviju linearnih višekoračnih metoda (Adams-Bashforth eksplicitne metode i Adams-Moulton implicitne metode) u prediktor-korektor formi. Pritom se najprije pomoću eksplicitne AB metode radi predikcija novog stanja sustava da bi se nakon toga radila korekcija tog novog stanja koristeći implicitnu AM metodu. Unutar Matlab/Simulinka implementirana je verzija ovog postupka uz promjenjivi red samih metoda pri čemu se red metode može mijenjati od reda 1 do reda 13. Pri strogim tolerancijama ovaj postupak je često efikasniji od defaultnog ode45 postupka.

ode15s(Stiff/NDF). Implicitni numerički postupak promjenjivog reda (od 1 do 5) zasnovan na numeričkim diferencijalnim formulama. Postupak je namijenjen prije svega za rješavanje problema simulacije krutih sustava. Preporuka je koristiti ovaj postupak ako simulacija s defultnim `ode45(Dormand-Prince)` postupkom predugo traje (što ukazuje na mogućnost da se radi o krutom sustavu).

ode23s(Stiff/mod. Rosenbrok). Ovaj je numerički postupak zasnovan na modificiranoj Rosenbrokovoj formuli reda 2 i namijenjen je za simulaciju krutih dinamičkih sustava. Budući da je `ode23s` jednokoračni postupak pokazuje se da je efikasniji od `ode15s` postupka pri grubljim tolerancijama.

ode23tb. Implicitni Runge-Kutta postupak koji koristi kombinaciju trapeznog pravila i unazadne diferencijalne formule reda 2. Slično kao `ode23s`, ovaj postupak se pokazuje efikasnijim od `ode15s` postupka pri grubljim tolerancijama.

3.6.2 Numerički postupci s konstantnim korakom

ode1(Euler). Eksplicitni jednokoračni postupak prvog reda temeljen na izrazu za unaprijednu Eulerovu integraciju. Postupak se odlikuje značajnom značajnom pogreškom te daje dobre rezultate samo uz malen vremenski korak h.

ode2(Heun). Eksplicitni, jednokoračni postupak drugog reda koji predstavlja poboljšanje osnovne Eulerove metode u smislu da se nagib stanja $x(t)$ na intervalu $[t_n,\ t_n+h]$ procjenjuje kao srednja vrijednost nagiba u trenutku t_n i procjene nagiba u trenutku $t_n + h$. Time se postiže bolja točnost nego kod `ode1(Euler)` postupka.

ode3(Bogacki-Shampine). Verzija `ode23(Bogacki-Shampine)` postupka koja koristi konstantan vremenski korak. Zasnovana je na Runge-Kutta metodi trećeg reda.

ode4(Runge-Kutta). Eksplicitni, jednokoračni postupak zasnovan na Runge-Kutta metodi četvrtog reda, koji predstavlja najpoznatiju i najčešće korištenu od Runge-Kutta metoda.

ode5(Dormand-Prince). Verzija `ode45(Dormand-Prince)` postupka koja koristi konstantan vremenski korak. Zasnovana je na Runge-Kutta metodi petog reda.

ode14x(Extrapolation). Postupak predstavlja implicitnu Runge-Kutta metodu koja koristi kombinaciju iterativnog Newton-Raphsonovog algoritma za rješavanje implicitnih algebarskih jednadžbi i ekstrapolacije. Pritom korisnik može definirati maksimalni broj iteracije Newton-Raphsonovog algoritma i red ekstrapolacije što izravno utječe na točnost i složenost numeričkog postupka. Postupak se prije svega koristi za simulaciju krutih dinamičkih sustava.

3.6.3 Odabir numeričkog postupka

Kod simulacije složenih dinamičkih sustava vrlo je teško unaprijed definirati prikladan simulacijski postupak kao i parametre samog postupka. Zbog toga je često nužno sam postupak i njegove parametre odrediti kroz više koraka metodom pokušaja-i-pogreške. Tako se prilikom simulacije složenih dinamičkih sustava preporuča prvo pokušati s nekim od postupaka s promjenjivim korakom uz uključenu detekciju prolaska kroz nulu. Pritom je defaultni `ode45(Dormand-Prince)` postupak dobar izbor. Ako su rezultati simulacije očekivani može nakon toga pokušati s nekim od manje složenih postupaka kako bi simulacija kraće trajala.

Ako je s druge strane simulacija korištenjem `ode45` postupka prespora ili se ne slaže s očekivanim rezultatima, tada je moguće da se radi o krutom dinamičkom sustavu pa je u tom slučaju preporuka koristiti neki od postupaka namijenjen za rješavanje krutih sustava poput `ode15s` postupka.

Kao sljedeći korak trebalo bi provjeriti jesu li odabrane tolerancije pogreške zadovoljavajuće. U tom smislu je potrebno nakon provođenja simulacije uz defaultne postavke za tolerancije `RTOL=1e-3`, simulirati sustav uz strože tolerancije npr. `RTOL=1e-6`. Ukoliko su rezultati slični tada je prvobitna tolerancija zadovoljavajuća. Ako se pak odzivi znatno razlikuje tada je potrebno ponoviti cijeli postupak uz strože tolerancije.

Ako se koriste postupci s konstantnim vremenskim korakom tada je potrebno iterativno kroz nekoliko simulacija odrediti prikladan vremenski korak h. To se radi na način da se provodi više simulacija svaki puta s korak umanjenim za određeni faktor (npr 10) i zaustavlja se postupak kada daljnje smanjenje koraka nema bitnijeg efekta na točnost simulacije.

U slučaju da se treba simulirati kruti dinamički sustav uz konstantan vremenski korak h tada se preporuča koristiti `ode14x` implicitni postupak zbog mogućnosti korištenja većih vremenskih koraka kod simulacije takvih sustava. Postupci s konstantnim korakom ne omogućuje korištenje detekcije prolaska kroz nulu nego je u slučaju takvih sustava potrebno smanjiti vremenski korak kako bi se smanjila pogreška.

Ako simulacijska shema sadrži elemente s prekapčanjem koji mogu prouzročiti pojavu diskontinuiteta i efekta poznatog pod imenom "chattering" potrebno je razmotriti mogućnost uvođenja histereznih elemenata kako bi se smanjio utjecaj prekapčanja.