УНИВЕРЗИТЕТ „СВ. КИРИЛ И МЕТОДИЈ“
СКОПЈЕ

ФАКУЛТЕТ ЗА ЕЛЕКТРОТЕХНИКА
И ИНФОРМАЦИСКИ ТЕХНОЛОГИИ

Настава на магистерски студии на студиската програма
ЕЛЕКТРОЕНЕРГЕТСКИ СИСТЕМИ

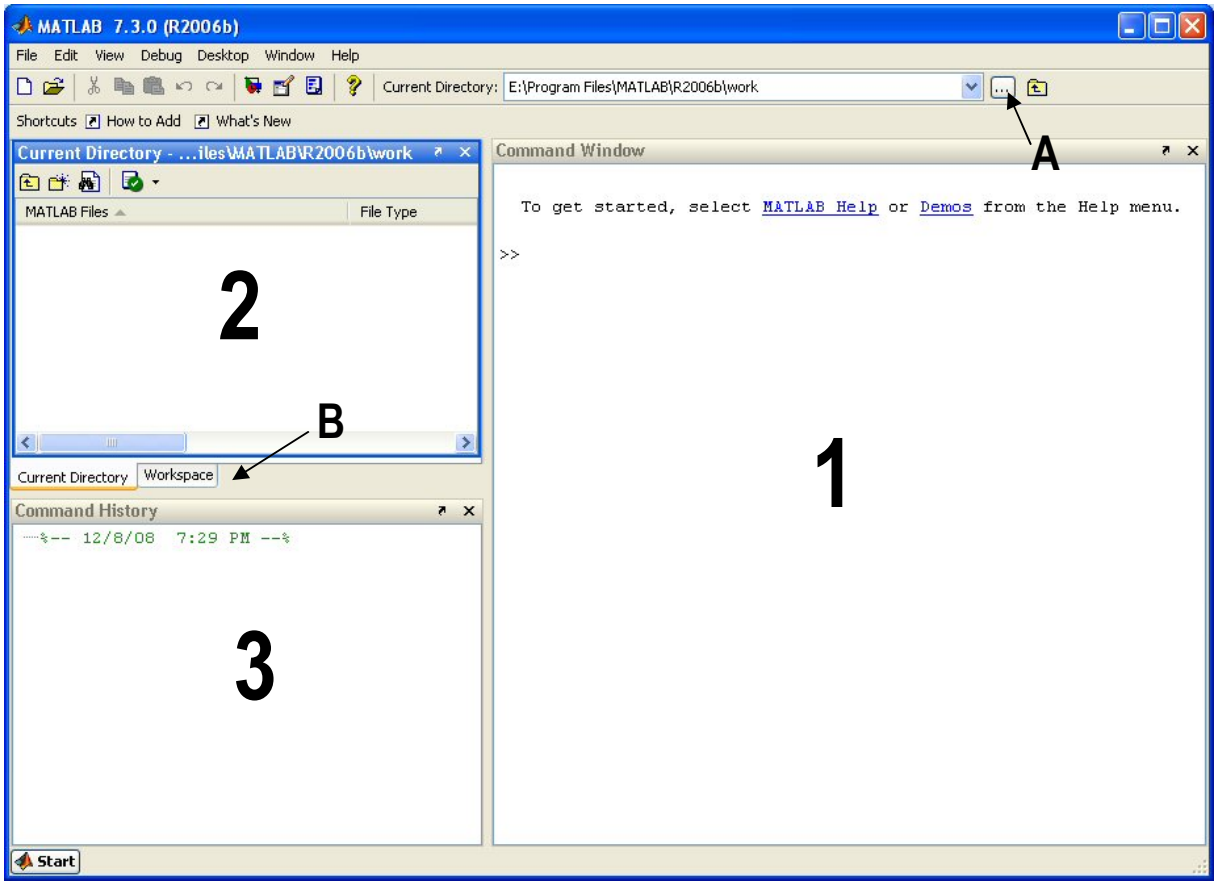
М. Тодоровски

Упатство за работа со Matlab

Скопје, јануари 2009 г.

1. ОСНОВНИ КОМАНДИ

Со активирање на Matlab се добива главниот прозорец кој што вообичаено го има изгледот прикажан на сликата 1.1. Главниот прозорец се состои од три други помали прозорци означени со броевите 1, 2 и 3. Првиот прозорец е Command Window и тој служи за пишување на командите кои што сакаме да бидат извршени. Вториот прозорец е Current Directory (тековен директориум) и во него ги гледаме датотеките кои што се содржани во директориумот наведен на лентата со алатки и кој по потреба можеме да го промениме така што кликнеме на копчето означено со A и ќе одбереме нов директориум. Во третиор прозорец Command History (историја со команди), кој што сега е празен, се запишуваат командите кои што ги задаваме во прозорецот Command Window (команден прозорец) така што доколку некогаш повторни ни затребаат истите команди можеме од него да ги ископираме. Покрај овој прозорец можеме да одбереме да го гледаме и прозорецот Workspace (работен простор) со кликнување на јазичето означено со B. Во работниот простор се прикажани променливите кои во моментот се користат во Matlab заедно со нивните вредности.



Слика 1.1. Изглед на главниот прозорец во Matlab

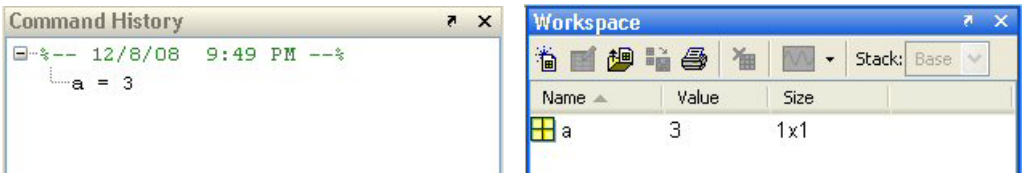
Доколку во командниот прозорец ја напишеме наредбата `a = 3` и притиснеме Enter ќе го добиеме следниот резултат

```
>> a = 3
a =
     3
```

На таков начин на променливата *a* сме и доделиле вредност 3, при што тоа ни е прикажано во командниот прозорец со изразот

```
a =
     3
```

а истовремено е променета содржината на Command History и на Workspace така што тие сега изгледаат како на сликата 1.2.



Слика 1.2. Изглед на Command History и на Workspace по извршување на една команда

На сликата 1.2 гледаме дека командата `a = 3` е запишана во историјата со команди, а во работниот простор е меморирана променливата за која што таму е наведено дека има димензии 1 × 1. Тоа е така затоа што во Matlab сите променливи се третираат како матрици. Во овој случај, бидејќи внесовме само еден број, програмот сфатил дека сме сакале да внесеме матрица со една редица и една колона.

Доколку истата команда ја напишевме со знакот ; на крајот, т.е. доколку напишевме

```
>> a = 3;
```

ќе ги добиевме наполно истите промени само што во командниот прозорец ништо немаше да ни се испише.

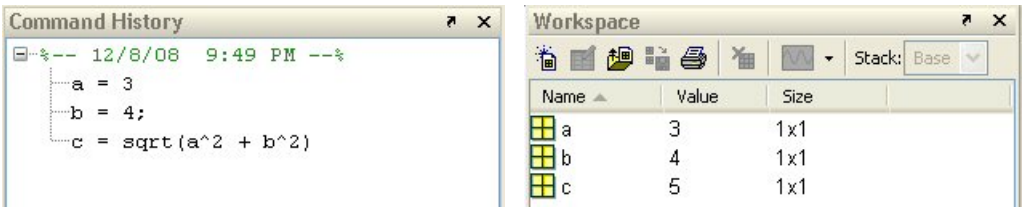
Ако понатаму напишеме две други команди на следниот начин

```
>> b = 4;
>> c = sqrt(a^2 + b^2)
```

ќе го добиеме следниот одговор

```
c =
     5
```

Воедно ќе дојде до промене на содржината на Command History и на Workspace така што тие сега ќе изгледаат како на сликата 1.3.



Слика 1.3. Изглед на Command History и на Workspace по извршување на повеќе команди

Со наредбата

```
>> p = [2 -4 7]
```

го добиваме следниот резултат

```
p =
     2    -4     7
```

со што всушност сме дефинирале променлива ***p*** која што е матрица со една редица и три колони.

Доколку сакаме да дефинираме матрица колона ***q*** тогаш можеме да напишеме

```
>> q = [
-1
2
9
]
```

со што го добиваме следниот резултат

```
q =
    -1
     2
     9
```

а до истиот резултат можевме да дојдеме и ако ја напишевме следната наредба

```
>> q = [-1; 2; 9]
```

што значи дека знакот ; при внесувањето на матрици означува дека после него треба да започне нова редица.

Дводимензионална матрица ***A*** можеме да внесеме на еден од следните два начини

```
>> A = [
1 3 6
8 9 2
0 3 1
]
```

или

```
>> A = [1 3 6; 8 9 2; 0 3 1]
```

со што го добиваме следниот идентичен резултат

```
A =
     1     3     6
     8     9     2
     0     3     1
```

Множењето на две матрици во Matlab се изведува на многу едноставен начин, исто како да множиме две обични променливи. На пример, можеме да напишеме

```
>> B = A * q
```

со што го добиваме векторот ***B*** кој што е производ на матрицата ***A*** и векторот ***q***, т.е.

```
B =
    59
    28
    15
```

При множењето на матрици треба да водиме сметка нивните димензии да се такви што ќе овозможат да се изврши нивното множење. На пример, можеме да пресметаме

```
>> p * q
ans =
    53
```

или пак

```
>> q * p
ans =
    -2     4    -7
     4    -8    14
    18   -36    63
```

но следната операција генерира грешка

```
>> q * A
??? Error using ==> mtimes
Inner matrix dimensions must agree.
```

затоа што бројот на колони на матрицата ***q*** не е еднаков со бројот редици на матрицата ***A***.

Следната операција

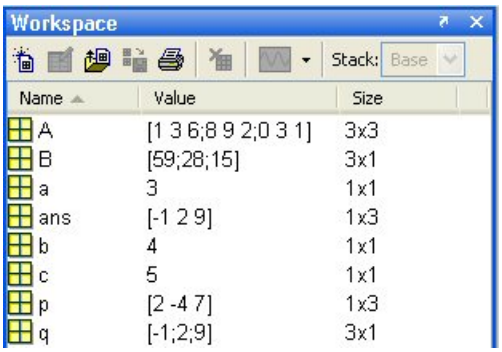
```
>> q' * A
ans =
    15    42     7
```

повторно е изводлива затоа што со наредбата `q'` правиме транспонирање на матрицата ***q*** со што се добива

```
>> q'
ans =
    -1     2     9
```

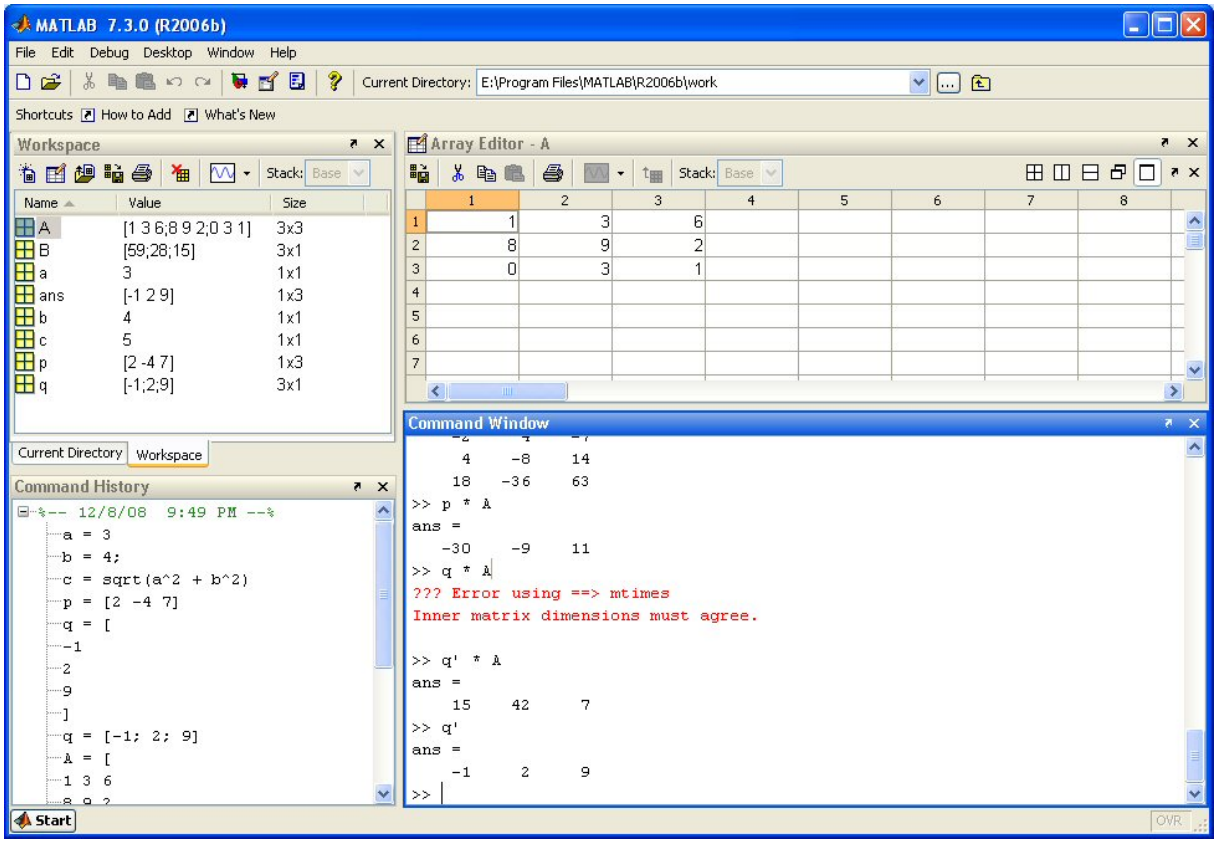
Како што може да се забележи во претходните неколку примери доколку напишеме наредба во која што нема промелнива од левата страна на наредбата, по која што следува знакот = а потоа и самата наредба, резултатот се запишува во променливата `ans`.

По извршувањето на сите претходни наредби прозорецот со Command History е пополнет со сите нив, додека работниот простор го има изгледот прикажан на сликата 1.4.



Слика 1.4. Повеќе променливи во работниот простор

Доколку кликнеме два пати на која било променлива од работниот простор, тогаш ќе ни се отвори нов прозорец во кој што ќе ја видиме содржината на одбраната променлива. Така на пример, ако ја одбереме променливата A го добиваме изгледот на Matlab прикажан на сликата 1.5 каде што ја гледаме содржината на променливата A над командниот прозорец.



Слика 1.5. Содржина на една променлива во Matlab

На крајот од воведниот дел ќе илустрираме како со досегашното познавање на Matlab и само на уште една едноставна наредба ќе можеме да решиме релативно комплициран математички проблем.

Пример 1.1. Да се одредат сите нули на полиномот

$$P(x) = x^4 - 10x^3 + 35x^2 - 50x + 24. \qquad (1.1)$$

Решение

Дефинираме матрица ***P*** со една редица во која што ќе ги сместиме коефициентите од полиномот (1.1). Тоа го правиме со следната наредба

```
>> P = [1   -10    35   -50    24];
```

Потоа ги одредуваме нулите на полиномот со помош на наредбата

```
>> nuli = roots(P)
nuli =
    4.0000
    3.0000
    2.0000
    1.0000
```

со што ја добиваме променливата `nuli` во која што е запишан резултатот. Од неа гледаме дека нулите на полиномот (1.1) се 4, 3, 2 и 1. Со тоа задачата е решена.

Ако сакаме да провериме, на пример, дали бројот 3 е навистина нула на полиномот (1.1) можеме да напишеме

```
>> proverka = 3^4 - 10*3^3 + 35*3^2 - 50*3 + 24
proverka =
     0
```

од каде што забележуваме дека бројот 3 е навистина нула на дадениот полином.

Проверката можевме да ја направиме и со помош на наредбата `polyval` на следниот начин

```
>> proverka = polyval(P, 3)
proverka =
     0
```

каде всушност сме напишале дека сакаме да ја пресметаме вредноста на полиномот кој што е зададен со матрицата $\boldsymbol{P}$ за вредност на $x = 3$.

Доколку пак напишеме

```
>> proverka = polyval(P, nuli)
proverka =
  1.0e-012 *
    0.1457
    0.0462
    0.0142
         0
```

ја пресметуваме вредноста на полиномот кој што е зададен со матрицата $\boldsymbol{P}$ за сите вредности кои се содржат во векторот `nuli`. Од резултатот забележуваме дека резултатот (сите елементи од променливата `proverka`) се помали од 10^{-12}.

Ако во полиномот некој од членовите на постои, како на пример во следниот случај

$$Q(x) = x^4 - 10x^3 - 50x + 24, \tag{1.2}$$

тогаш во матрицата со која што го дефинираме тој полином на соодветното место ставаме елемент еднакон на нула. Постапувајќи така, нулите на полиномот (1.2) ги добиваме на следниот начин

```
>> Q = [1 -10 0 -50 24];
>> nuli2 = roots(Q)
nuli2 =
  10.4378
  -0.4496 + 2.1869i
  -0.4496 - 2.1869i
   0.4613
```

од каде што забележуваме дека две од нулите се комплексни броеви.

□ □ □

2. РЕШАВАЊЕ НА СИСТЕМИ ЛИНЕАРНИ РАВЕНКИ

Во општ случај еден систем линеарни равенки може да се напише во матричен облик на следниот начин

$$\boldsymbol{A} \cdot \boldsymbol{x} = \boldsymbol{b}\,, \tag{2.1}$$

каде што $\boldsymbol{A}$ е матрицата на системот која што е квадратна со димензии $n \times n$ (за систем со n равенки) и ги содржи коефициентите пред промеливите од левата страна на системот равенки. Векторот $\boldsymbol{b}$ има димензии $n \times 1$ и тој ги содржи слободните членови од десната страна на системот равенки, додека векторот $\boldsymbol{x}$ кој што исто така има димензии $n \times 1$ треба да се определи.

Постојат повеќе начини за решавање на системот равенки (2.1), но веројатно најелегантниот начин за определување на непознатиот вектор $\boldsymbol{x}$ е со помош на инверзната матрица на системот на следниот начин

$$\boldsymbol{x} = \boldsymbol{A}^{-1} \cdot \boldsymbol{b}\,. \tag{2.2}$$

За почеток, тука, ќе го користиме само овој начин кој што ќе го илустрираме преку неколку примери.

Пример 2.1. Да се реши системот линеарни равенки

$$\begin{aligned} -x_1 + 3x_2 + 7x_3 &= 64 \\ 3x_1 - 2x_2 + x_3 &= 4 \\ 4x_1 + x_2 - 8x_3 &= -38 \end{aligned}\,, \tag{2.3}$$

Решение

На почетокот ги внесуваме матрицата на системот $\boldsymbol{A}$ и векторот со слободните членови $\boldsymbol{b}$ на следниот начин

```
>> A = [-1 3 7; 3 -2 1; 4 1 -8];
>> b = [64; 4; -38];
```

а потоа го пишуваме матричниот израз (2.2) кој што во Matlab гласи

```
>> x = A^-1 * b
x =
    3.0000
    6.0000
    7.0000
```

со што е добиено следното решение $x_1 = 3$, $x_2 = 6$ и $x_3 = 7$, со што зададениот проблем е решен.

Претходно опишаната постапка за решавање на системи линеарни равенки во Matlab е многу едноставна и единствено нешто на кое што треба да внимаваме е правилно да ги внесеме матрицата $\boldsymbol{A}$ и векторот $\boldsymbol{b}$. На пример, ако системот равенки беше

$$\begin{aligned} -x_1 + 3x_2 + 7x_3 &= 64 \\ 3x_1 + x_3 &= 4 \\ 4x_1 + x_2 - 8x_3 &= -38 \end{aligned}\,, \tag{2.4}$$

решението ќе го добиевме на следниот начин

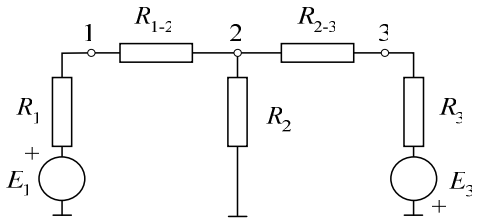
```
>> A = [-1 3 7; 3 0 1; 4 1 -8];
```

```
>> b = [64; 4; -38];
>> x = A^-1 * b
x =
   -0.5094
    8.2642
    5.5283
```

каде што се забележува дека вториот елемент од втората редица на матрицата A е еднаков на нула затоа што во втората равенка од системот (2.4) го нема членот со x_2.

□ □ □

Пример 2.1. На сликата 2.1 е прикажано едно електрично коло кое што се состои од отпорници и временски константни напонски генератори. Со примена на методот на независни напони (јазлови потенцијали) да се одредат вредностите на напоните на јазлите 1, 2 и 3.



Слика 2.1.

Бројни вредности:
$R_1 = 4\ \Omega$; $R_2 = 0{,}5\ \Omega$; $R_3 = 10\ \Omega$;
$E_1 = 12$ V; $E_3 = 9$ V;
$R_{1\text{-}2} = 20\ \Omega$; $R_{2\text{-}3} = 10\ \Omega$.

Решение

Со помош на методот на независни напони (јазлови потенцијали) за едно електрично коло може да се напише следниот матричен систем равенки

$$\boldsymbol{Y} \cdot \boldsymbol{U} = \boldsymbol{I}\,, \tag{2.5}$$

кој што дава врска помеѓу напоните на независните јазли $\boldsymbol{U}$ (сите јазли освен референтниот) и инјектираните струи $\boldsymbol{I}$.

Матрицата $\boldsymbol{Y}$ која што се нарекува матрица на адмитанции на јазлите може да се добие на различни начини. Доколку во разгледуваното коло нема индуктивни спреги помеѓу одделни елементи (што е овде исполнето), матрицата $\underline{\boldsymbol{Y}}$ едноставно се определува со инспекција на мрежата:

- Членот Y_{kk} е еднаков на сумата на адмитанциите на сите елементи на кои им припаѓа јазелот k.
- Членот Y_{kl} е еднаков на сумата на адмитанциите на сите елементи што директно ги поврзуваат јазлите k и l, со променет знак.

Според наведените правила, за колото од сликата 2.1 матрицата $\boldsymbol{Y}$ ќе биде

$$\boldsymbol{Y} = \left[\begin{array}{c|c|c} \frac{1}{R_1}+\frac{1}{R_{1-2}} & -\frac{1}{R_{1-2}} & 0 \\ \hline -\frac{1}{R_{1-2}} & \frac{1}{R_2}+\frac{1}{R_{1-2}}+\frac{1}{R_{2-3}} & -\frac{1}{R_{2-3}} \\ \hline 0 & -\frac{1}{R_{2-3}} & \frac{1}{R_3}+\frac{1}{R_{2-3}} \end{array}\right] = \left[\begin{array}{c|c|c} \frac{1}{4}+\frac{1}{20} & -\frac{1}{20} & 0 \\ \hline -\frac{1}{20} & \frac{1}{0{,}5}+\frac{1}{20}+\frac{1}{10} & -\frac{1}{10} \\ \hline 0 & -\frac{1}{10} & \frac{1}{10}+\frac{1}{10} \end{array}\right], \tag{2.6}$$

што во Matlab се пишува на следниот начин

```
Y = [1/4 + 1/20, -1/20, 0
    -1/20, 1/0.5 + 1/20 + 1/10, -1/10
     0, -1/10, 1/10 + 1/10]
```

со што го добиваме следниот резултат

```
Y =
    0.3000   -0.0500         0
   -0.0500    2.1500   -0.1000
         0   -0.1000    0.2000
```

Векторот на инјектираните струи во јазлите $\boldsymbol{I}$ за колото од сликата 2.1 е следниот

$$\boldsymbol{I} = \begin{bmatrix} \frac{E_1}{R_1} \\ \hline 0 \\ \hline \frac{-E_3}{R_3} \end{bmatrix} = \begin{bmatrix} \frac{12}{4} \\ \hline 0 \\ \hline \frac{-9}{10} \end{bmatrix}, \tag{2.7}$$

или во Matlab тоа е

```
I = [12/4; 0; -9/10]
```

со што се добива

```
I =
    3.0000
         0
   -0.9000
```

Користејќи ја релацијата (2.5) за непознатите напони на јазлите можеме да напишеме

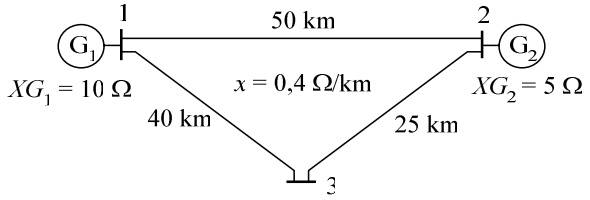
$$\boldsymbol{U} = \boldsymbol{Y}^{-1} \cdot \boldsymbol{I}, \tag{2.8}$$

односно

```
>> U = Y^-1 * I
U =
   10.0040
    0.0239
   -4.4880
```

□ □ □

Пример 2.2. На сликата 2.2 е прикажано мал ЕЕС за кој што се познати параметрите на сите елементи за директен систем. Со помош на матрицата на импеданции на јазлите $\boldsymbol{Z}$ да се пресмета струјата на трифазна куса врска во јазелот 3, како и напоните на останатите јазли и струите во сите гранки. Номиналниот напон изнесува 110 kV.



Слика 2.2.

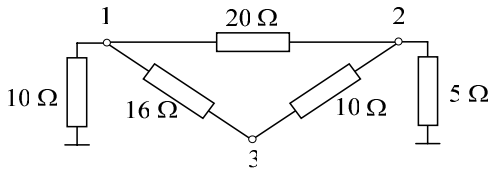
Решение

Во едно електрично коло матрицата на импеданции на јазлите $\boldsymbol{Z}$ се добива како инверзна матрица на матрицата на адмитанции на јазлите т.е. важи

$$\boldsymbol{Z} = \boldsymbol{Y}^{-1}, \tag{2.9}$$

што значи дека за да го решиме проблемот е потребно прво да ја формираме матрицата $\boldsymbol{Y}$.

Најнапред го формираме еквивалентното коло за системот од сликата 2.2 кое што е прикажано на сликата 2.3 каде што секој од елементите е заменет со својата реактанција за директен систем.



Слика 2.3. Еквивалентно коло за системот од сликата 2.2

Според сликата 2.3 и правилата за формирање на матрицата $\boldsymbol{Y}$ наведени во претходниот пример во Matlab можеме веднаш да напишеме

```
Y = [1/10 + 1/20 + 1/16, -1/20, -1/16
    -1/20, 1/20 + 1/10 + 1/5, -1/10
    -1/16, -1/10, 1/10 + 1/16];
```

а потоа да ја добиеме и матрицата $\boldsymbol{Z}$ на следниот начин

```
>> Z = Y^-1
Z =
    6.1983    1.9008    3.5537
    1.9008    4.0496    3.2231
    3.5537    3.2231    9.5041
```

Нека ја дефинираме променливата

```
>> k = 3;
```

која што го содржи индексот на јазелот за кој што сакаме да ја пресметаме струјата на куса врска.

Во тој случај, за струјата на куса врска можеме да напишеме

```
>> Ik = 110/sqrt(3) / Z(k,k)
Ik =
    6.6822
```

со што првиот дел од задачата е решен.

Напоните на јазлите можеме да ги добиеме на следниот начин

$$U_i = \frac{U_\text{n}}{\sqrt{3}} - Z_{ik} \cdot I_k \,,\; i = 1, 2, 3. \tag{2.10}$$

Бидејќи во Matlab сите променливи се матрици и сите операции се прават врз матрици равенката (2.10) ќе ја напишеме во матрична форма на следниот начин

$$\boldsymbol{U} = \begin{bmatrix} U_1 \\ U_2 \\ U_3 \end{bmatrix} = \frac{110}{\sqrt{3}} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - I_k \cdot \begin{bmatrix} Z_{1k} \\ Z_{2k} \\ Z_{3k} \end{bmatrix} , \tag{2.11}$$

што значи дека за да ги пресметаме напоните на сите јазли покрај веќе пресметаната струја I_k потребна ни е и колоната k од матрицата $\boldsymbol{Z}$. Неа во Matlab можеме да ја добиеме на следниот начин

```
>> Zkolona = Z(:,k)
Zkolona =
    3.5537
    3.2231
    9.5041
```

каде што знакот : во изразот `Z(:,k)` значи дека од матрицата $\boldsymbol{Z}$ сакаме да ги земеме елементите од сите нејзини редици и тоа само од колоната k затоа што за вториот индекс имаме внесено k. Доколку сакавме да ја земеме редицата k тогаш требаше да напишеме `Z(k,:)`.

На крајот останува уште да се напише изразот (2.11) во Matlab

```
>> U = 110/sqrt(3) * ones(3,1) - Ik * Zkolona
U =
   39.7619
   41.9709
   -0.0000
```

со што ги добивме напоните на сите јазли. Во последната команда со `ones(3,1)` дефинираме матрица со три редици и една колона која што се состои само од единици.

Струите во гранките можеме да ги пресметаме така што ќе ги одземеме напоните на нивните два крај и таа разлика ќе ја поделиме со реактанцијата на соодветната гранка. Тоа го правиме на следниот начин

```
>> I12 = (U(1) - U(2))/20
I12 =
   -0.1104
>> I13 = (U(1) - U(3))/16
I13 =
    2.4851
>> I23 = (U(2) - U(3))/10
```

```
I23 =
    4.1971
```

Ќе наведеме уште еден начин за пресметка на струите во гранките каде што претходното ќе го направиме на поелегантен начин. Нека ја дефинираме следната матрица

```
>> MREZA = [1 2 20; 1 3 16; 2 3 10]
MREZA =
     1     2    20
     1     3    16
     2     3    10
```

во која што бројот на редици е еднаков со бројот на гранки во мрежата, а во секоја редица се запишани по три броја кои што ги даваат индексите на почетните и крајните јазли на гранките и нивната реактанција. Така на пример првата редица од оваа матрица треба да ја толкуваме на следниот начин: првата гранка тргнува од јазелот 1, завршува кај јазелот 2 и има реактанција еднаква на 20 Ω.

Потоа ги дефинираме следните матрици

```
>> IP = MREZA(:,1)
IP =
     1
     1
     2
>> IQ = MREZA(:,2)
IQ =
     2
     3
     3
>> XGR = MREZA(:,3)
XGR =
    20
    16
    10
```

кои што всушност ги содржат елементите од првата, втората и третата колона од матрицата MREZA соодветно.

Со помош на матрицата `IP` можеме многу едноставно да ги добиеме напоните на почетните јазли од сите гранки ако напишеме

```
>> U(IP)
ans =
   39.7619
   39.7619
   41.9709
```

Овој резултат треба да го толкуваме на следниот начин: напонот на почетниот јазел од првата гранка изнесува 39,7619 kV, напонот на почетниот јазел од втората гранка изнесува 39,7619 kV и напонот на почетниот јазел од третата гранка изнесува 41,9709 kV.

Според тоа за струите во сите гранки можеме да напишеме

```
>> IGR = (U(IP) - U(IQ)) ./ XGR
IGR =
   -0.1104
    2.4851
    4.1971
```

од каде што се гледа дека го добивме истиот резултат како и претходно. Во последната команда операцијата `./` значи дека операцијата делење нема да се прави врз матрици (што и не може да се направи) туку соодветните елементите од векторите ќе се делат меѓусебно. На пример ако имаме два вектори

```
>> a = [1; 2; 3];
>> b = [2; 8; 10];
```

резултатот ќе биде

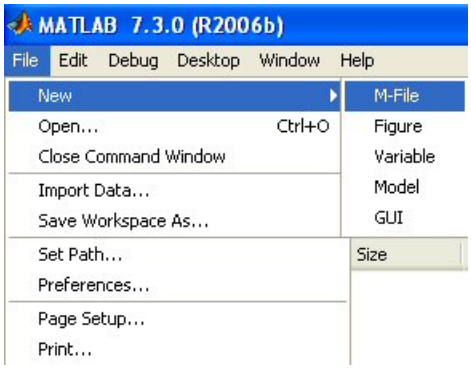
```
ans =
    0.5000
    0.2500
    0.3000
```

□ □ □

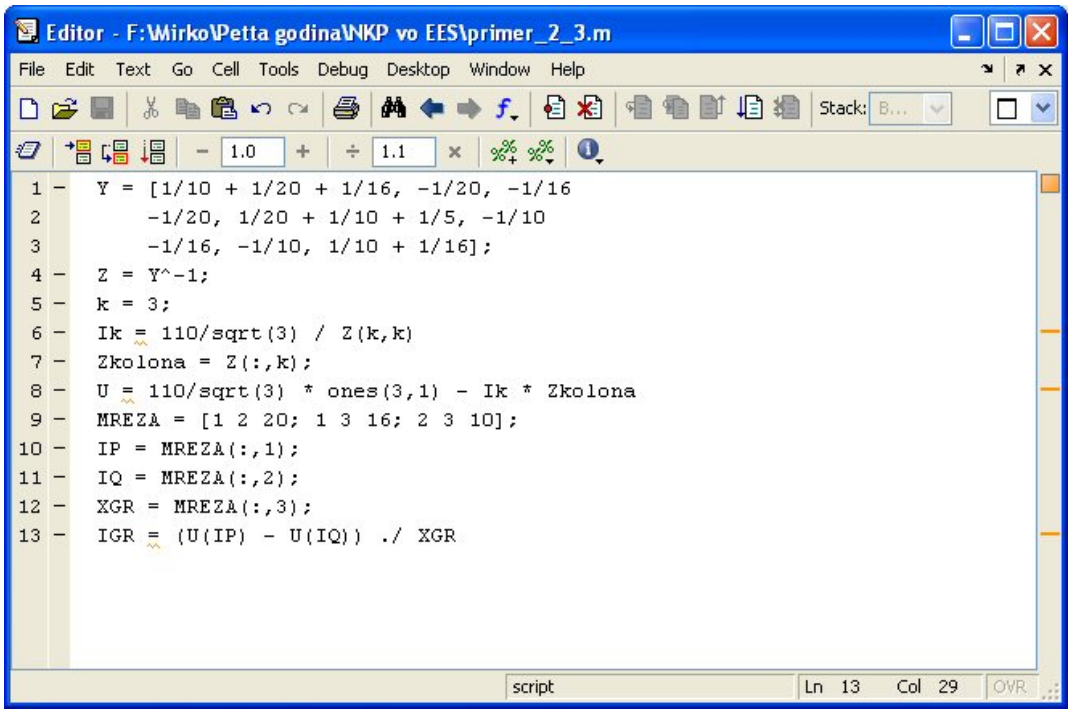
Пример 2.3. Командите од примерот 2.2 да се сместат во една m-датотека која што ќе биде активирана како една комплетна програма за пресметка на куси врски.

Решение

Програмите во Matlab се сместуваат во m-датотеки кои што можат да се пишуваат со едиторот кој што се активира од главното мени како што тоа е прикажано на сликата 2.4 со што се добива едиторот од сликата 2.5. Во него едноставно ги пишуваме сите команди кои што ги наведовме во примерот 2.2 така што неговата содржина ќе биде како на сликата 2.5. На крајот напишаната m-датотека ја снимаме на дискот под името primer_2_3.m. Тука треба да се нагласи дека во името не смее да има празни места ниту знаци кои што означуваат оператори како што се +, -, /, * итн.



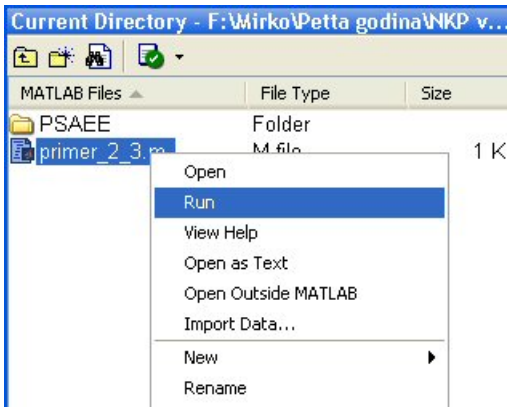
Слика 2.4. Избирање на нова m-датотека



Слика 2.5. Едитор за m-датотеки

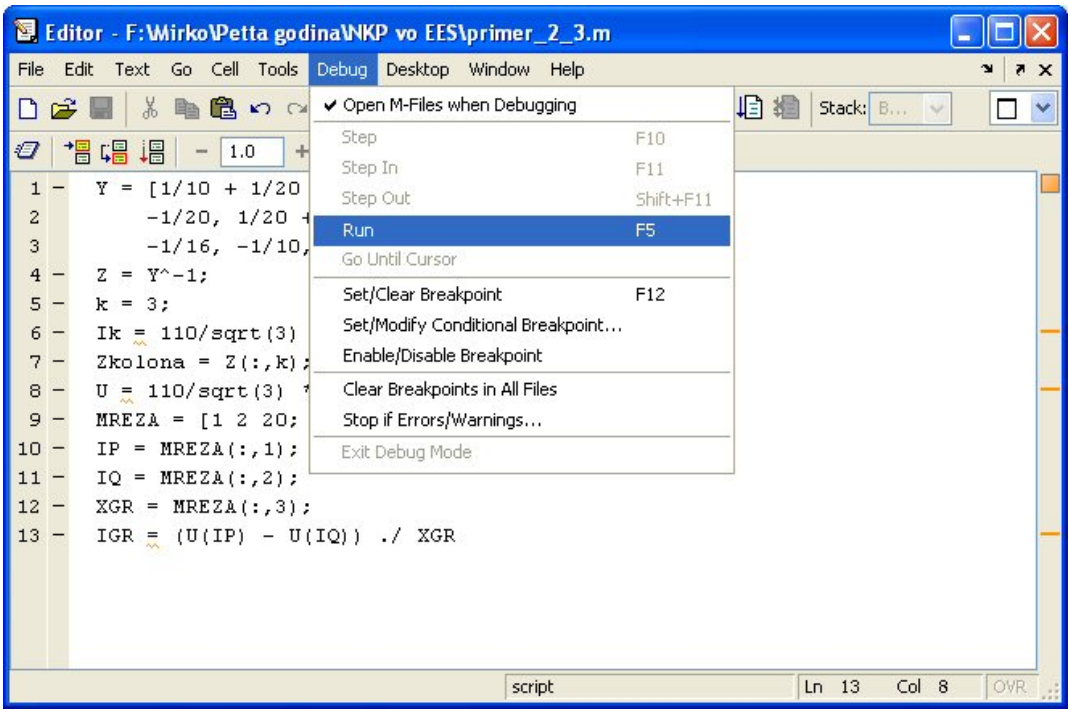
За да можеме да ја активираме програмата primer_2_3.m во Current Directory (означено со A на сликата 1.1) да ја напишеме (или избереме) патеката на дискот до местото на кое што сме ја снимиле таа програма.

Првиот начин за активирање на програмата е да кликнеме врз неа со десното копче од глушецот и да го одбереме Run од мените како што тоа е прикажано на сликата 2.6.



Слика 2.6. Активирање на програмата primer_2_3.m со помош на глушецот

Вториот начин за активирање на програмата е преку менито во едиторот за m-датотеки како што тоа е прикажано на сликата 2.7. Додека третиот начин е со пишување на primer_2_3 (без .m на крајот) во командниот прозорец.



Слика 2.7. Активирање на програмата primer_2_3.m преку едиторот за m-датотеки

Кој било начин да го избереме како резултат од активирањето на програмата го добиваме следниот резултат

```
Ik =
    6.6822
U =
   39.7619
   41.9709
   -0.0000
IGR =
   -0.1104
    2.4851
    4.1971
```

□ □ □

3. ЧИТАЊЕ НА ПОДАТОЦИ ОД НАДВОРЕШНИ ДАТОТЕКИ

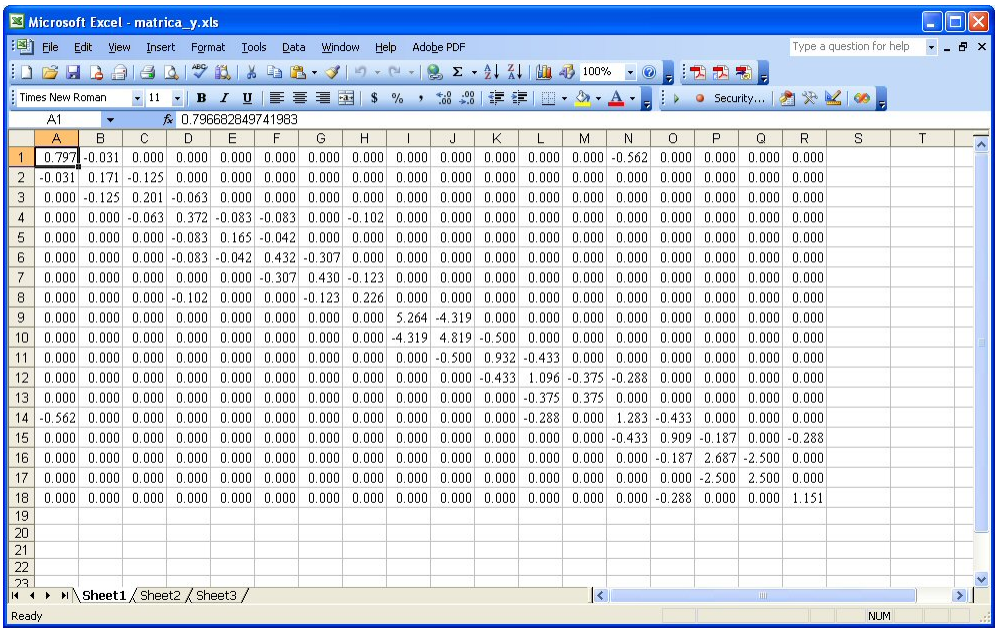
Често пати може да се јави потреба да се прочитаат податоци кои што не се директно напишани во Matlab туку се генерирани во некој друг софтвер. На пример голем број програми, вклучувајќи ги и професионалните програми за анализа на ЕЕС, имаат можност за снимање на податоците кои што тие ги користат во форма на датотеки во Excel или пак обични такстуални датотеки. Со помош на Matlab таквите податоци можеме дополнително да ги обработиме или, пак, со нивно користење да направиме дополнителни пресметки кои што во останатите софтвери ги нема. За таа цел прво треба да се запознаеме со командите на Matlab кои што овозможуваат читање и запишување на податоци во Excel или текстуален формат. Постапката ќе ја илустрираме преку примери.

Пример 3.1. Нека е дадена датотеката matrica_y.xls во која што во првиот работен лист почнувајќи од ќелијата A1 е сместена матрицата на адмитанции $\boldsymbol{Y}$ на еден EEC која што е добиена со помош на друг софтвер (слика 3.1). Со помош на Matlab да се пресмета матрицата на импеданции $\boldsymbol{Z}$, а потоа таа да се смести на следните локации

а) во нова датотека со име matrica_z.xls;

б) во вториот работен лист во датотека matrica_y.xls;

в) во нов работен лист со име matica_z во датотека matrica_y.xls;



Слика 3.1. Матрица на адмитанции на еден EEC дадена во Excel

Решение

За случајот под а) ќе направиме програма во m-датотека која што ќе ја снимиме на дискот под име primer_3_1a.m. Нејзината содржина е следната

```
Y = xlsread('matrica_y.xls');
Z = Y^-1;
xlswrite('matrica_z.xls', Z);
```

Како што се забележува читањето на податоци од Excel се прави со помош на наредбата `xlsread` во која што како параметар се задава името на датотеката од која што сакаме да ги прочитаме податоците, при што бидејќи тоа е текст се става во единечни наводници. Потоа со наредбата `Z = Y^-1` ја пресметуваме матрицата $\boldsymbol{Z}$, која што на крајот ја запишуваме во Excel со помош на наредбата `xlswrite`. Во наредбата `xlswrite` прво во единечни наводници се пишува името на датотеката во која што сакаме да го запишеме резултатот, а потоа се задава променливата која што сакаме да ја запишеме.

За случајот под б) програмата е дадена во датотеката primer_3_1b.m од каде што се забележува дека таа е скоро иста како програмата во случајот под а) само што тука во последната команда по променливата `Z` сме го напишале името на работниот лист во кој што сакаме да биде снимен резултатот во датотеката matrica_y.xls. Содржината на програмата е

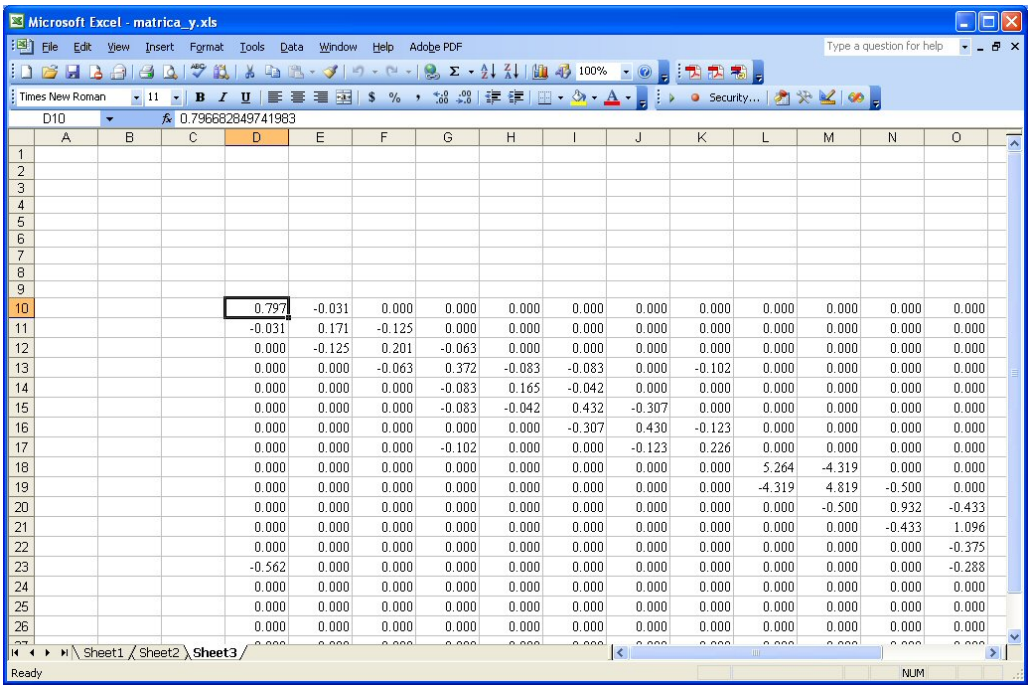
```
Y = xlsread('matrica_y.xls');
Z = Y^-1;
xlswrite('matrica_y.xls', Z, 'Sheet2');
```

За случајот под в) програмата е дадена во датотеката primer_3_1v.m и таа се разликува од програмата во случајот под б) по тоа што тука името на работниот лист во кој што сакаме да биде снимен резултатот е matica_z. Бидејќи таков работен лист не постоел пред активирањето на програмата, тој ќе биде креиран и потоа ќе биде запишан резултатот. Содржината на програмата е

```
Y = xlsread('matrica_y.xls');
Z = Y^-1;
xlswrite('matrica_y.xls', Z, 'matica_z');
```

□ □ □

Пример 3.2. Нека податоците за матрицата на адмитанции ***Y*** се сместени во третиот работен лист од matrica_y.xls опсегот D10:U27 како што тоа е прикажано на сликата 3.2. Со помош на Matlab да се пресмета матрицата на импеданции ***Z***, а потоа таа да се смести во истата датотека во нов работен лист со име matica_z во опсегот F4:W21.



Слика 3.2. Матрица на адмитанции на еден ЕЕС дадена во Excel во третиот работен лист

Решение

Програмата е дадена во датотеката primer_3_2.m и го има следниот изглед

```
Y = xlsread('matrica_y.xls', 3, 'D10:U27');
Z = Y^-1;
xlswrite('matrica_y.xls', Z, 'matica_z', 'F4:W21');
```

Тука читањето го правиме на тој начин што во командата `xlsread` прво го задаваме името на датотеката од која што сакаме да читаме, потоа го задаваме редниот број на работниот лист и на крајот опсегот од кој што сакаме да ги прочитаме податоците.

Запишувањето на податоците го правиме така што во командата `xlswrite` прво го задаваме името на датотеката во која што сакаме да го запишеме резултатот, потоа ја задаваме променливата која што сакаме да ја запишеме, името на работниот лист и на крајот опсегот во кој што сакаме да биде запишано решението.

Доколку во било која команда го погрешиме опсегот и напишеме на пример

```
Y = xlsread('matrica_y.xls', 3, 'D10:T26');
```

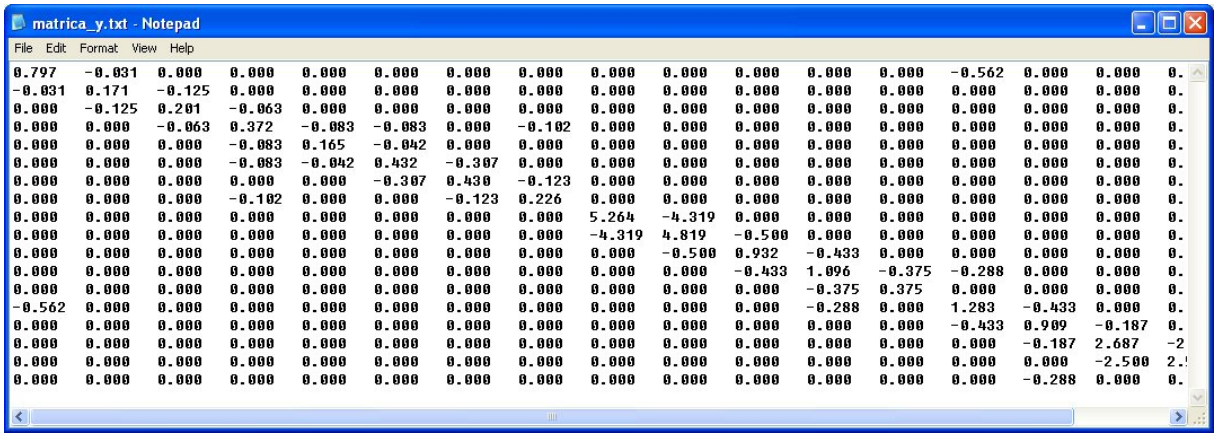
или

```
xlswrite('matrica_y.xls', Z, 'matica_z', 'F4:V20');
```

тогаш во првиот случај погрешно ќе ја прочитаме матрицата, односно ќе прочитаме 17 редици и колони наместо 18, а во вториот случај ќе запишеме 17 редици и колони со што ќе имаме погрешни резултати.

□ □ □

Пример 3.3. Нека е дадена текстуалната датотека matrica_y.txt во која што е сместена матрицата на адмитанции ***Y*** на еден ЕЕС (слика 3.3). Со помош на Matlab да се пресмета матрицата на импеданции ***Z***, а потоа таа да се сними во текстуалната датотека matrica_z.txt.



Слика 3.3. Матрица на адмитанции на еден ЕЕС дадена текстуална датотека

Решение

Програма е дадена во датотеката primer_3_3.m и нејзината содржина е следната

```
Y = load('matrica_y.txt');
Z = Y^-1;
save 'matrica_z.txt' Z -ascii
```

Како што се забележува читањето на податоци од текстуални датотеки се прави со помош на наредбата `load` во која што како параметар се задава името на датотеката од која што сакаме да ги прочитаме податоците. Потоа со наредбата `Z = Y^-1` ја пресметуваме матрицата ***Z***, која што на крајот ја запишуваме во текстуалната датотека matrica_z.txt со помош на наредбата `save` во која што по името на датотеката се задава променливата која што сакаме да ја снимиме и клучниот збор `-ascii` со што сме одбрале снимањето да се направи во обичен текстуален формат.

Наместо наредбата

```
save 'matrica_z.txt' Z -ascii
```

можеме да ја користиме и наредбата

```
csvwrite('matrica_z_csv.txt', Z)
```

со тоа што резултатот пак ќе биде запишан во текстуален формат во датотеката matrica_z_csv.txt но броевите ќе бидат одделени со запирки помеѓу себе.

□ □ □

4. ДЕФИНИРАЊЕ НА ФУНКЦИИ – НУЛИ НА ФУНКЦИИ

Во Matlab постојат два начина за дефинирање на функции од страна на корисникот и тоа со пишување во една командна линија или со дефинирање на функција во посебна m-датотека. Двата пристапа ќе ги илустрираме преку примери.

Пример 4.1. Дадена е функцијата $f(x)$ од една променлива

$$f\left(x\right) = xe^{-x} - 0,2\,; \tag{4.1}$$

за која што е потребно да извршат следните работи:

а) да се напише функција во Matlab во една линија во која што ќе биде дефинирана фукцијата (4.1), а потоа таа да се нацрта во интервалот $0 \le x \le 5$.

б) да се повтори задачата под а) со тоа што функција ќе се напише во посебна m-датотека.

Решение

а) За дефинирање на функции во една линија во Matlab се користи наредбата inline, со која што во овој случај ќе ја дефинираме функцијата (4.1) на следниот начин

```
>> f = inline('x .* exp(-x) - 0.2')
f =
     Inline function:
     f(x) = x .* exp(-x) - 0.2
```

каде што се гледа по наредбата inline во единечни загради се пишува функцијата како обичен текст затворен со единечни наводници. Во функцијата наместо обично множење имаме ставено точкасто множење (.*) со кое што операцијата множење помеѓу елементите x и exp(x) се прави за секој елемент од x посебно. За се увериме дека нашата нова функција работи можеме да напишеме

```
>> f(1)
ans =
    0.1679
```

што навистина одговара на вредноста на изразот $1 \cdot e^{-1} - 0,2$.

За да ја нацртаме функцијата (4.1) прво ќе ја дефинираме променлива ***x*** како вектор кој што ги содржи броевите од 0 до 5 со чекор од 0,1. Тоа го правиме на следниот начин:

```
>> x = 0:0.1:5;
```

а потоа ја пресметуваме вредноста на функцијата во сите точки од векторот ***x*** со помош на наредбата

```
>> y = f(x);
```

и на крајот функцијата ја цртаме со помош на наредбата

```
>> plot(x, y)
>> grid
```

со што го добиваме резултатот прикажан на сликата 4.1.

Наредбата `plot` се користи за цртање на функции при што како прв параметар се задава вектор кој што содржи координати од хоризонталната оска, а потоа вектор со координати од вертикалната оска. Со наредбата `grid` ги цртаме хоризонталните и вертикални исцрткани линии кои што се гледаат на сликата 4.1. До истиот резултат ќе

дојдевме и доколку наместо `plot(x, y)` напишевме `plot(x, f(x))`. Во тој случај претходната команда `y = f(x);` е непотребна.

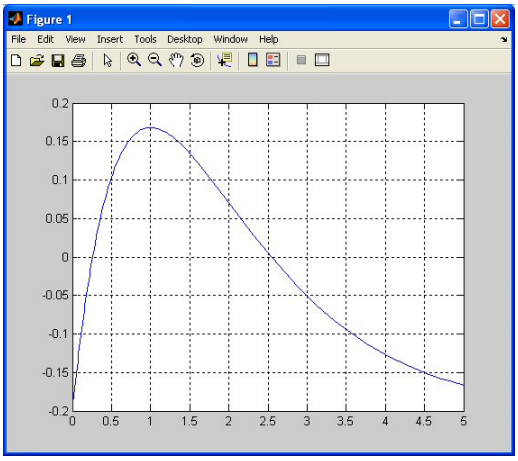
Покрај овој начин, графикот на функцијата (4.1) можеме да го добиеме и со помош на следната команда

```
>> fplot(f, [0 5])
>> grid
```

каде што ја користиме наредбата `fplot` во која што се внесуваат два параметри. Првиот параметар е името на функцијата која што сакаме да ја цртаме, а вториот параметар е матрица со една редица и две колони која што ги содржи почетната и крајната вредност на интервалот во кој што сакаме да ја нацртаме функцијата.

Тоа значи дека за случајот под а) можеме да направиме една програма (primer_4_1a.m) чија што содржина е

```
f = inline('x .* exp(-x) - 0.2');
fplot(f, [0 5])
grid
```



Слика 4.1. Изглед на функцијата 4.1

б) Во овој случај функцијата (4.1) ќе ја дефинираме во посебна m-датотека на следниот начин

```
function y = funkcija_4_1(x)
y = x .* exp(-x) - 0.2;
```

при што оваа функција ќе ја снимиме во датотека со име funkcija_4_1.m кое што мора да биде исто со името на функцијата кое претходно сме го задале. Од овој пример се гледа дека функциите се дефинираат така што прво се пишува клучниот збор `function` а потоа следуваат листа на променливите кои што дадената функција ги пресметува (во овој случај тоа е само променливата `y`, доколку имаше повеќе променливи тие ќе беа ставени во средни загради одделени со запирки), знакот за еднаквост, името на функцијата и на крајот во мали загради листа на промеливи одделени со запирки кои што се влезни параметри во функцијата (во овој случај тоа е само променливата `x`). И овде, како проверка за исправноста на нашата функција можеме да напишеме

```
>> funkcija_4_1(1)
ans =
    0.1679
```

За случајот под б) целата програма за цртање на функцијата е дадена во датотеката primer_4_1b.m и таа го има следниот изглед

```
fplot(@funkcija_4_1, [0 5])
```

```
grid
```

Тука забележуваме дека пред името на функцијата која влегува како параметар во наредбата `fplot` го имаме ставено знакот `@` со сме кажале дека предметната функција е дефинирана во m-датотека со име funkcija_4_1.m.

□ □ □

Пример 4.2. Да се нацрта функцијата $f(x)$ од една променлива дефинирана на следниот начин

$$f(x)=\begin{cases} xe^{-x}-0,2; & x<3, \\ \dfrac{x-8}{100}; & x\geq 3. \end{cases} \tag{4.2}$$

во интервалот $0\leq x\leq 10$.

Решение

За решавање на овој пример задолжително треба да дефинираме функција во посебна m-датотека затоа што е невозможно функцијата дефинирана со изразот (4.2) да ја напишеме во една линија. Поради тоа ја дефинираме следната функција

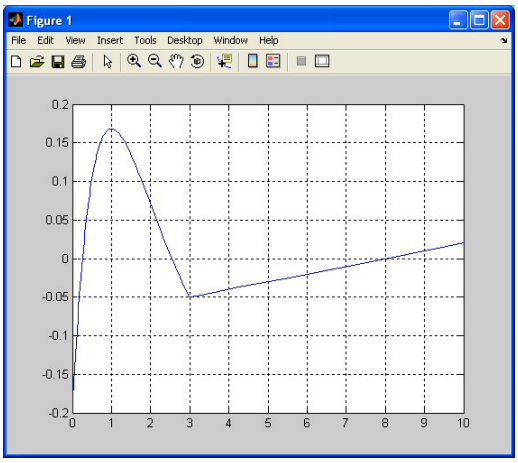
```
function y = funkcija_4_2(x)
if x < 3
    y = x .* exp(-x) - 0.2;
else
    y = (x - 8)/100;
end
```

која што е снимена во датотеката funkcija_4_2.m.

Во неа ја користиме наредбата if со која се дефинира извршување на една или друга команда во зависност од тоа дали е исполнет условот кој што е даден со неа. Ако условот е исполнет (`x < 3`) тогаш се извршува командата `y = x .* exp(-x) - 0.2;`, а во спротивно (`else`) се извршува командата `y = (x - 8)/100;`. На крајот од наредбата if секогаш следува end. Програмата за цртање на оваа функција е дадена во primer_4_2.m и го има следниот изглед

```
fplot(@funkcija_4_2, [0 10])
grid
```

Со активирање на програмата primer_4_2.m ја добиваме сликата 4.2.



Слика 4.1. Изглед на функцијата 4.2

□ □ □

Пример 4.3. Да се најдат нулите на фукцијата (4.1) во интервалот $0 \le x \le 5$.

Решение

Од графикот на функцијата (4.1) кој што е даден на сликата 4.1 забележуваме дека таа во интервалот $0 \le x \le 5$ има две нули. Едната нула се наоѓа околу 0,25 додека другата се наоѓа околу 2,5.

Точната вредност на првата нула ќе ја одредиме со помош на наредбата fsolve на следниот начин

```
>> f = inline('x .* exp(-x) - 0.2');
>> fsolve(f, 0.25)
Optimization terminated: first-order optimality is less
than options.TolFun.
ans =
    0.2592
```

од каде што се гледа дека во наредбата `fsolve` како параметар прво се пишува името на функцијата за која што сакаме да ја пресметаме нулата, а потоа почетното решение за самата нула.

До истиот резултат ќе дојдевме ако напишевме

```
>> fsolve(@funkcija_4_1, 0.25)
Optimization terminated: first-order optimality is less
than options.TolFun.
ans =
    0.2592
```

затоа што претходно веќе ја имеме дефинирано функцијата `funkcija_4_1`. Бидејќи таа е дефинирана во посебна m-датотека пред нејзиното име го ставаме знакот `@`.

Точната вредност на втората нула ќе ја одредиме со помош на наредбата fsolve на следниот начин

```
>> fsolve(f, 2.5)
Optimization terminated: first-order optimality is less
than options.TolFun.
ans =
    2.5426
```

□ □ □

5. ИЗВОДИ И ИНТЕГРАЛИ

Иако во математиката за пресметувањето на изводи и интеграли на функции постојат аналитички постапки кои што во Matlab се вградени во електротерхниката ни е сосема доволно доколку вредностите на интергралите и изводите ги пресметаме нумерички. На крајот од каква било анализа која што ја прават инженерите резултатите се сведуваат на добивање на решение во нумеричка форма со конкретни вредности за одредени величини како што се напони, струи, моќности, должини, висини итн. додека аналитичките изрази за сите тие величини се без никакво дополнително значење. Поради тоа тука ќе се задржиме само на нумеричките методи.

Пример 5.1. Да се пресмета вредноста на првиот извод на функцијата $f(x) = x^2 - 3x + 1$ за вредност на $x = 1$.

Решение

По дефиниција првиот извод на една функција е даден со изразот

$$f'(x) = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon) - f(x)}{\varepsilon}, \tag{5.1}$$

кој што ние тука ќе го искористиме за нумерички да го пресметаме изводот.

Прво ќе ја дефинирама функцијата $f(x)$ на следниот начин

```
>> f = inline('x^2 - 3*x + 1')
f =
     Inline function:
     f(x) = x^2 - 3*x + 1
```

а потоа ќе го пресметаме изводот

```
>> izvod = (f(1 + 0.001) - f(1))/0.001
izvod =
   -0.9990
```

со што сме добиле резултат кој што е многу близок до точната вредност (која што изнесува -1) и сме претпоставиле дека е $\varepsilon = 0{,}001$.

□ □ □

Пример 5.2. Да се пресмета вредноста на интегралот $\int_0^1 \left(x^2 - 3x + 1\right) dx$ со помош на трапезното правило и со помош на функцијата quadl.

Решение

Кај трапезното правило интервалот на интегрирање се дели на n точки рамномерно оддалечени една од друга на растојание Δx. Може да се покаже дека во таков случај вредноста на интегралот може да се пресмета со помош на изразот

$$I = \left(\frac{y_1 + y_n}{2} + \sum_{i=2}^{n-1} y_i \right) \cdot \Delta x, \tag{5.2}$$

каде што $y_1, y_2, \ldots y_n$ се вредности на функцијата која што се интегрира во n-те точки на кои што е поделен интервалот на интегрирање.

Програмата за пресметка на интеграл со помош на трапезното правило е дадена во датотеката primer_5_2.m и нејзиниот изглед е следниот

```
 1. f = inline('x.^2 - 3*x + 1');
 2. dx = 0.1;
 3. x = 0:dx:1;
 4. y = f(x);
 5. n = size(x, 2);
 6. I = (y(1) + y(n))/2;
 7. for i = 2:n-1
 8.     I = I + y(i);
 9. end
10. I = I * dx
```

во неа во првиот ред е дефинирана подинтегралната функција, а во вториот ред е дефинирано растојанието помеѓу точките на интеграција `dx`. Во третиот ред е дефиниран векторот `x` кој што е една редица која ги содржи броевите од 0 до 1 со чекор од 0,1. Во четвртио ред е пресметана вредноста на подинтегралната функција за сите точки кои што се содржат во `x` и при тоа резултатот е сместен во променливата `y`. Во петтиот ред е одреден бројот на колони во променливата `n`, односно бројот на точки на кои што е поделен интервалот на итеграција. Во редовите од 6 до 9 пресметан е изразот во заградата од (5.2). Најпрво е пресметан првиот член, а потоа во еден `for` циклус во кој што бројачот `i` оди од 2 до n-1 додадени се останатите членови при што во деветиот ред на крајот од циклусот стои командата `end`. Во последниот, десети ред, пресметаната сума е помножена со Δx според зададеното од изразот (5.2).

Со активирање на програмата го добиваме следниот резултат

```
>> primer_5_2
I =
   -0.1650
```

кој што е многу блиску до точниот резултат од $-\frac{1}{6}$. Доколк во програмата во вториот ред напишеме `dx = 0.05;` тогаш ќе добиеме резултат -0.1663.

Истиот интеграл можеме да го решиме и со помош на наредбата quadl на следниот начин

```
>> f = inline('x.^2 - 3*x + 1');
>> quadl(f, 0, 1)
ans =
   -0.1667
```

од каде што се забележува дека со само две команди во командниот прозор го добивме точниот резултат. Наредбата `quadl` се повикува така што како параметри се задаваат подинтегралната функција и долната и горната граница на интегралот.

□ □ □

Пример 5.3. Да се пресмета вредноста на интегралот $\int_0^1 \frac{xe^x dx}{(1+x)^2}$ и да се провери дали решението е исто со вредноста $\frac{e}{2}-1$ која што е најдена од табела со решени интеграли.

Решение

```
>> f = inline('x .* exp(x) ./ (1 + x).^2');
>> moe_resenie = quadl(f, 0, 1)
moe_resenie =
    0.3591
>> tablicno_resenie = exp(1)/2 - 1
tablicno_resenie =
    0.3591
```

□ □ □

6. КОМПЛЕКСНИ БРОЕВИ

Комплексните броеви во Matlab се пишуваат со користење на имагинарните единици `i` и `j` на еден од следните начини

```
>> Z = 2.3 + 4.8 * i
Z =
   2.3000 + 4.8000i
>> U = 231 * exp(0.2 * i)
U =
  2.2640e+002 +4.5893e+001i
```

при што првиот број е напишан во правоаголни координати, а вториот во експоненцијален обли, но резултатот секогаш се добива во правоаголни координати.

При користењето на комплексни броеви треба да се внимава на променливите `i` и `j` да не им се задаваат вредности затоа што во спротивен случај тие нема да бидат имагинарни единици. На пример, ако во некоја програма сме напишале `i = 4;` тогаш бројот `Z` од претходно ќе стане

```
>> Z = 2.3 + 4.8 * i
Z =
   21.5000
```

т.е. тој добил вредност 2.3 + 4.8 * 4. За ова да го исправиме треба да ја избришеме претходната променлива `i` или пак пред да пишуваме комплексен број да напишеме

```
>> i = sqrt(-1)
i =
         0 + 1.0000i
```

Врз комплексните броеви во Matlab може да се применуваат сите операции кои се применуваат и кај реалните броеви, а истото тоа важи и за матриците.

Пример 6.1. Во еден ЕЕС е приклучен потрошувач на собирници чиј што напон изнесува $\underline{U} = 110,3 \cdot e^{j4,1^0}$ kV, додека струјата на потрошувачот изнесува $\underline{I} = 0,23 \cdot e^{j14,9^0}$. Да се пресметаат активната и реактивната моќност на потрошувачот како и факторот на моќност.

Решение

На почетокот ги дефинираме напонот и струјата како комплексни броеви

```
>> U = 110.3 * exp(4.1/180*pi * i)
U =
  1.1002e+002 +7.8862e+000i
>> I = 0.23 * exp(14.9/180*pi * i)
I =
   0.2223 + 0.0591i
```

каде што дадените агли од степени сме ги претвориле во радијани.

Привидната моќност на потрошувачот ја пресметуваме според изразот $\underline{S} = \sqrt{3} \cdot \underline{U} \cdot \underline{I}^*$ што во Matlab гласи

```
>> S = sqrt(3) * U * conj(I)
```

```
S =
  43.1621 - 8.2336i
```

Активната и реактивната моќност се еднакви на реалниот односно имагинарниот дел од привидната моќност

```
>> P = real(S)
P =
   43.1621
>> Q = imag(S)
Q =
   -8.2336
```

додека модулот на привидната моќност го пресметуваме на следниот начин

```
>> abs(S)
ans =
   43.9404
```

Факторот на моќност можеме да го пресметаме според изразот $\cos\varphi = \frac{P}{S}$ или во Matlab тоа е

```
>> cos_fi = P/abs(S)
cos_fi =
    0.9823
```

До истиот резултат ќе дојдевме ако прво го пресметавме аголот φ, а потоа косинусот од него

```
>> fi = angle(S)
fi =
   -0.1885
>> cos_fi = cos(fi)
cos_fi =
    0.9823
```

Во претходниот случај аголот е пресметан во радијани. Доколку сакаме да го добиеме резултатот во степени треба да напишеме

```
>> fi_stepeni = angle(S)/pi*180
fi_stepeni =
  -10.8000
```

□ □ □

7. ДИФЕРЕНЦИЈАЛНИ РАВЕНКИ

Во електроенергетските системи постојат две големи области на изучување во кои што проблемите се сведуваат на решавање на системи диференцијални равенки од прв ред. Тие области се решавањето на преодните процеси при комутационите и атмосферските пренапони како и решавањето на проблемите од статитчката и динамичката стабилост на ЕЕС.

Решенијата на диференцијалните равенки, кај нашите проблеми, се функции со кои што е дадена временската промена на одредена големина како што е напон, струја,

агол на роторот на генератор итн. Егзактното решавање на диференцијалните равенки, кон кое секогаш се тежнее во математиката, ги дава овие функции во аналитички облик, односно дава сложени изрази од независната променлива (време) со која што е опишана промената на одредена величина. Инженерскиот пристап кон добивањето на решението ќе биде поинаков затоа што нас ни е сосема доволно да го познаваме решението дадено во нумерички облик од кој што можеме да ги извлечеме сите потребни заклучоци. Нумеричкото решение е дадено во еден вектори кои што содржат нумерички вредности на функциите кои што ги бараме за одредени вредности на независната променлива (одредени временски моменти). Од тие вектори можеме да заклучиме дали одредена величина (напон, струја, агол ...) има осцилаторна или апериодична промена, дали нејзината максимална вредност претставува опасност за некоја опрема или уред (пренапони или ударни струи), дали одреден систем е стабилен итн.

Во Matlab постојат повеќе методи за нумеричко решавање на системи диференцијани равенки од прв ред како што се ode23, ode45, ode113, ode15s, ode23s, ode23t и ode23tb. Тука, преку примери, ќе ја разгледаме примената на функцијата ode45 за решавање на системи диференцијални равенки од прв ред од следниот облик

$$\begin{aligned}
&\frac{dy_1}{dt} = f_1\left(t, y_1, y_2, \ldots y_n\right);\\
&\frac{dy_2}{dt} = f_2\left(t, y_1, y_2, \ldots y_n\right);\\
&\vdots\\
&\frac{dy_n}{dt} = f_n\left(t, y_1, y_2, \ldots y_n\right);
\end{aligned} \tag{7.1}$$

од каде што се забележува дека диференцијалните равенки треба да бидат напишани на таков начин што од левата страна ќе бидат дадени првите изводи на зависните променливи y_1, y_2, … y_n, додека од десната страна може да има изрази (функции) од произволен облик (не само линеарен) кој што може да ги содржи сите зависни променливи како и за независната променлива t.

Пример 7.1. Да се реши следната диференцијална равенка

$$\frac{dy}{dt} = t^2 - y \cdot t, \tag{7.2}$$

за $0 \le t \le 5$ ако е познато дека почетната вредност на зависната променлива изнесува $y(0) = 3$.

Решение

На почетокот дефинираме функција со која што ќе биде дадена десната страна на равенката (7.2), односно со неа ќе биде дефиниран првиот извод на зависната променлива y

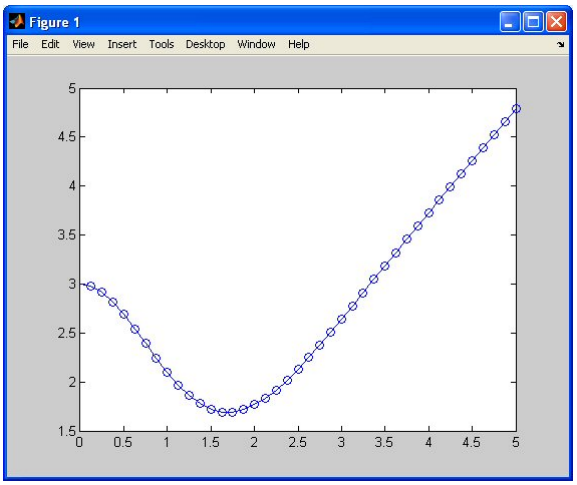
```
>> f = inline('t^2 - y*t')
f =
     Inline function:
     f(t,y) = t^2 - y*t
```

а потоа ја повикуваме функцијата ode45 на следниот начин

```
>> ode45(f, [0 5], 3)
```

каде што се забележува дека прво се задава името на функцијата со која што е дефиниран изводот на зависната променлива y (десната страна на дадената диференцијална равенка), потоа се задава интервалот за независната променлива t за кој што сакаме да го добиеме решението (матрица со една редица и две колони во која што се содржани почетната и крајната вредност за t) и на крајот се задава почетната вредност на зависната променлива y.

Со активирање на претходната команда функцијата ode45 нумерички ги пресметува вредностите на t и y и веднаш ја црта промената на функцијата $y(t)$ како што тоа е прикажано на сликата 7.1.



Слика 7.1. Временски тек на функцијата $y(t)$ која што е решение на равенката (7.1)

Доколку сакаме да ги добиеме бројните вредности на t и y за кои што е нацртана сликата 7.1 тогаш во командниот прозорец треба да напишеме

```
>> [t, y] = ode45(f, [0 5], 3);
```

со што во работниот простор ќе добиеме два вектори со по 41 елемент кои ќе ги содржат координатите на 41-та точка од сликата 7.1.

Во овој случај сликата 7.1 нема автоматски да биде нацртана, туку ние сами ќе треба да ја генерираме со помош на следните наредби

```
>> plot(t, y)
>> grid
```

□ □ □

Пример 7.2. Да се реши следниот систем диференцијални равенки

$$\begin{aligned} \frac{dy_1}{dt} &= \sin t - y_2, \\ \frac{dy_2}{dt} &= y_1 - \sin t, \end{aligned} \tag{7.2}$$

за $0 \le t \le 10$ ако е познато дека почетните вредности на зависните променливи изнесуваат $y_1(0) = 1$ и $y_2(0) = 2$.

Решение

За разлика од примерот 7.1 тука функцијата со која што ќе ги зададеме десните страни од двете диференцијални равенки од системот (7.2) ќе ја дефинираме во m-датотека со име izvodi_primer_7_2.m чија што содржина е

```
function y = izvodi_primer_7_2(t, y)
y = [
    sin(t) - y(2)
    y(1) - sin(t)
    ];
```

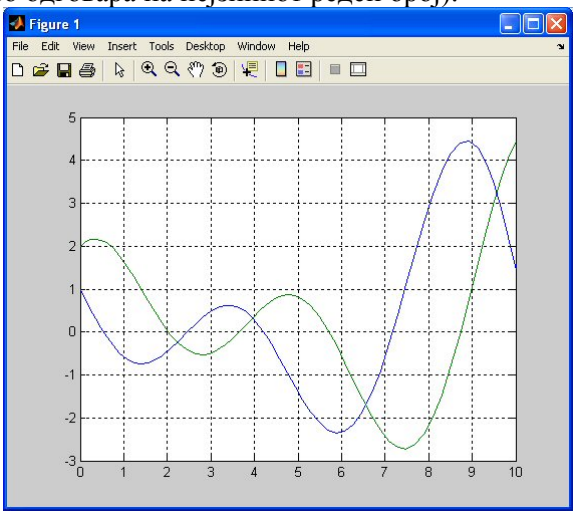
што значи сме дефинирале функција која што како резултат дава еден вектор `y` со два елементи кои се еднакви на десните страни од двете равенки од системот (7.2).

Понатаму, во датотеката primer_7_2.m ја имаме напишано следната програма

```
[t, y] = ode45(@izvodi_primer_7_2, [0 10], [1 2]);
plot(t, y)
grid
```

со која што го добиваме решението на системот диференцијални равенки (пред името на функцијата стои знакот @ затоа што таа е дефинирана во посебна m-датотека). Временските промена на функциите $y_1(t)$ и $y_2(t)$ кои што се исцртуваат со активирањето на претходната програма е дадена на сликата 7.2.

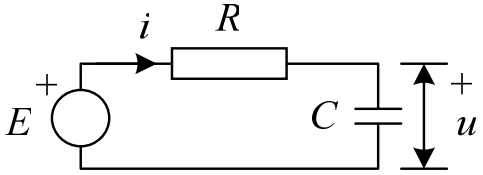
(*забелешка*: доколку сакаме да нацртаме само една од кривите, на пример втората, тогаш треба да напишеме `plot(t, y(:,2))` бидејќи `y` е матрица чиј што број на колони е еднаков на бројот на равенки во зададениот проблем и решението за секоја од функциите е сместено во соодветна колона која што одговара на нејзиниот реден број).



Слика 7.2. Временски тек на функциите $y_1(t)$ и $y_2(t)$ кои што се решение на системот равенки (7.2)

□ □ □

Пример 7.3. Го посматраме RC колото на сликата 7.3 кое што е приклучено на генератор со константен напон E. Потребно е да го определиме и нацртаме временскиот тек на напонот $u(t)$ за следните бројни вредности $R = 1000\ \Omega$; $C = 100\ \mu F$ и $E = 12$ V. Кондензаторот на почетокот бил празен.



Слика 7.1. RC – коло

Решение

За прикажаното коло можеме да ги напишеме следните равенки

$$E = u + Ri, \tag{7.3}$$

$$i = C\frac{du}{dt}, \tag{7.4}$$

од каде што следува

$$E = u + RC\frac{du}{dt}, \tag{7.5}$$

$$\frac{du}{dt} = \frac{E-u}{RC}. \tag{7.6}$$

Равенката (7.6) е диференцијална равенка со која што е опишано колото од сликата 7.1 и неа ќе ја решиме на идентичен начин како што ги решивме равенките во претходните два примера.

Најнапред во ќе дефинираме функција RC напишана во m-датотека со име RC.m. Таа функција ќе ги има следните параметри: *t*, *u*, *R*, *C* и *E*, при што нивниот редоследот е следниот:

1. прва доаѓа независната променлива, која во нашиот случај е времето *t*. Таа во случајов е вектор во кој се сместени временските моменти за кои се прави пресметката на непознатата *u* при решавањето на диференцијалната равенка,
2. потоа доаѓа векторот на бараното решение (или матрицата на решението кај проблемите со повеќе променливи) кој ги содржи вредностите на променливата (променливите) пресметани во временските моменти кои се дадени во векторот *t*,
3. понатаму се наведуваат променливите кои се специфични за дадениот проблем како што се тоа параметрите на колото *R*, *C* и *E* во нашиот случај.

Функцијата RC треба да врати број (вектор со броеви за случаите со повеќе променливи) кој е еднаков на вредноста на изводот на променливата (променливите) пресметани со актуелните вредности на промеливата (променливите). Таа треба да биде направена така што во неа треба да биде опишана десната страна од релацијата (7.6), па според тоа таа ќе го има следниот изглед

```
function dudt = RC(t, u, R, C, E)
dudt=(E-u)/(R*C);
```

Диференцијалната равенка (7.6) ќе ја решиме со помош на програмата primer_7_3.m која го има следниот изглед

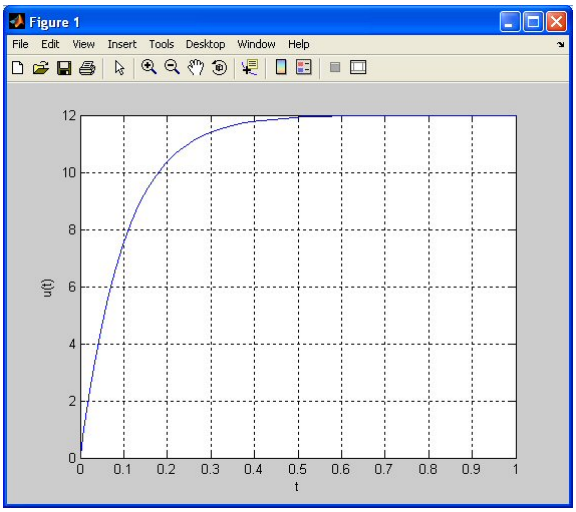
```
R = 1000; C = 100e-6; E = 12;
[t,u] = ode45(@RC, [0 1], 0, [], R, C, E);
plot(t,u); xlabel('t'); ylabel('u(t)');
grid
```

Во неа функцијата ode45 се повикува така што по името на функцијата @RC, е даден временскиот интервал [0 1] за кој сакаме да имаме пресметани вредности на промеливите (во овој случај од 0 до 1 s), потоа почетната вредност на *u* (вредност 0), празен вектор [] со што на функцијата ode45 и задаваме default опции со кои ќе ја решава диференцијалната равенка и на крајот се дадени променливите кои се специфични за дадениот проблем.

Дијаграмот $u(t)$ го цртаме со командата

```
plot(t,u); xlabel('t'); ylabel('u(t)');
```

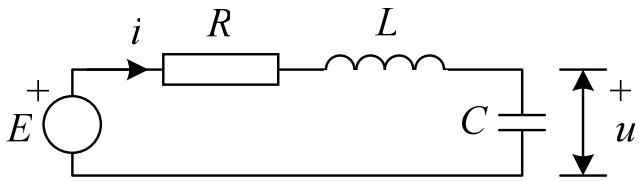
во која што за разлика од досегашното нејзино користење имаме додадено уште две наредби со кои се пишуваат ознаки под хоризонталната и лево од вертикалната оска како што тоа е прикажано на сликата 7.3.



Слика 7.3. Временска промена на напонот *u*(*t*) во RC колото

□ □ □

Пример 7.4. Го посматраме RLC колото на сликата 7.4 кое што е приклучено на генератор со константен напон E. Потребно е да ги определиме и нацртаме временскиот тек на напонот $u(t)$ и струјата $i(t)$ за следните бројни вредности $R = 3{,}6\ \Omega$; $L = 0{,}57$ H; $C = 100\ \mu$F и $E = 12$ V. Кондензаторот на почетокот бил празен, а струјата во калемот била еднаква на нула.



Слика 7.4. RLC – коло

Решение

Состојбата во колото ќе ја опишеме со помош на две дифренцијални равенки од прв ред при што за непознати ќе ги прогласиме струјата $i(t)$ и напонот на кондензаторот $u(t)$. Во тој случај врз основа на сликата 7.4, со примена на II Кирхофов закон можеме да напишеме

$$E = u + Ri + L\frac{di}{dt}, \tag{7.7}$$

додека за струјата во колото важи

$$i = C \cdot \frac{du}{dt}. \tag{7.8}$$

Понатаму, системот диференцијални равенки ќе го напишеме така што на неговата лева страна се наоѓаат првите изводи по непознатите $i(t)$ и u (t) а на десната изразите со кои што тие изводи се пресметуваат. На тој начин, од изразите (7.7) и (7.8) добиваме

$$\frac{di}{dt} = \frac{E - Ri - u}{L}, \tag{7.9}$$

$$\frac{du}{dt} = \frac{i}{C}. \tag{7.10}$$

Сега ќе дефинираме фукција која што ќе ја наречеме RLC и која ќе ја снимиме во датотеката RLC.m

```
function dydt = RLC(t, y, R, L, C, E)
dydt = [
    (E - R*y(1) - y(2)) / L
    y(1) / C
];
```

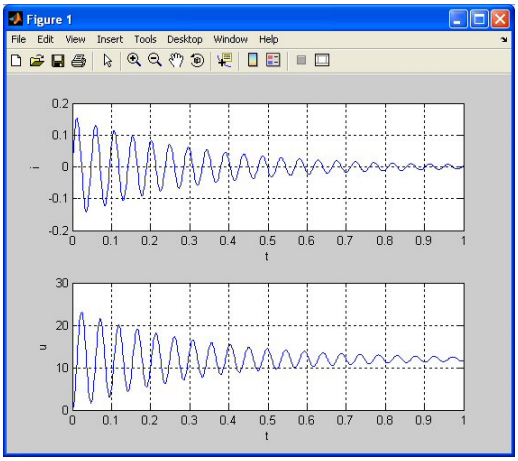
Во неа независна променлива е времето t додека непозната е променливата y која е всушност матрица со две колони. Првата колона од y е променливата i, втората колона е променливата u.

Потоа ќе формираме друга m-датотека со назив primer_7_4.m која ќе ја сместиме на дискот во истиор фолдер со датотеката RLC.m. Изгледот на m-датотеката primer_7_4.m е следниот

```
R = 3.6; L = 0.57; C = 100e-6; E = 12;
[t,y] = ode45(@RLC, [0 1], [0 0], [], R, L, C, E);
subplot(2, 1, 1), plot(t, y(:,1)); xlabel('t'); ylabel('i'); grid
subplot(2, 1, 2), plot(t, y(:,2)); xlabel('t'); ylabel('u'); grid
```

Проблемот ќе го решиме со повикување на програмата primer_7_4 со што како резултат ќе го добиеме векторот t со вкупно $n = 633$ дискретни моменти (големината на n ја одлучува алгоритмот) во коишто се пресметувани вредностите на променливите y_1 и y_2 т.е. струјата i и напонот u. Променливите ќе бидат сместени во матрицата y којшто претставува матрица со n редици и со две колони. Притоа во првата колона на матрицата y се сместени дискретните вредности за струјата i, додека во втората – дискретните вредности за напонот u. Овие вредности можеме да ги видиме ако на работниот простор (Workspace) кликнеме два пати на иконата со која е опишана матрицата y.

Како резултат се добиваат и два графика на ист екран, т.е. иста страница. На едниот е прикажана променливата y_1, т.е. зависноста $i(t)$, а на другата променливата y_2, т.е. зависноста $u(t)$ како што тоа е прикажано на сликата 7.5.

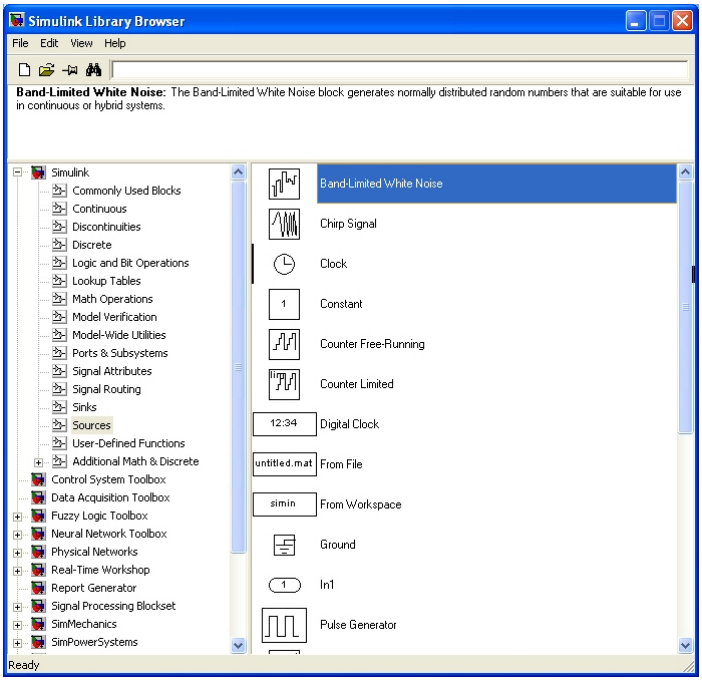


Слика 7.5 Струја во калемот и напон на кондензаторот во RLC колото

□ □ □

8. РЕШАВАЊЕ НА ЕЛЕКТРИЧНИ КОЛА СО ПОМОШ НА SIMULINK[1]

Simulink е модул во Matlab наменет за симулација на динамички процеси во разни системи со примена на графички начин на работа. Решавање на електрични кола се прави со формирање на модел на колото со користење на готови блокови (графички симболи) со кои се моделираат одделните елементи на колото како што се: напонски и струјни генератори, отпорници, калеми, кондензатори и нелинеарни елементи. Основната предност на работата со Simulink е едноставноста во претставувањето на проблемот кој што треба да се реши и можноста за прикажување на резултатите на повеќе начини. Тој се активира со помош на командата simulink која што треба да се напише во командниот простор во Matlab, со што се добива прозорецот прикажан на сликата 8.1.



Слика 8.1. Прозор со основни блокови во Simulink

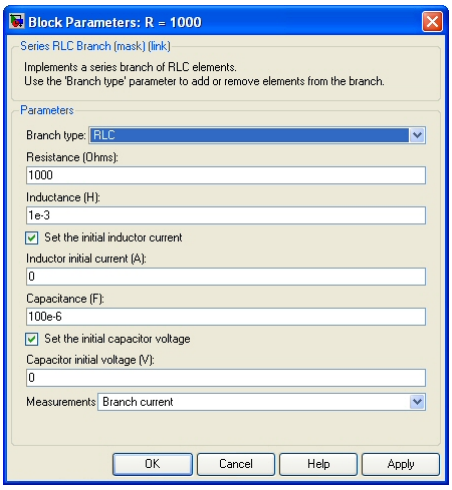
Примената на Simulink за решавање на електрични кола ќе ја илустрираме преку примери.

Пример 8.1. Да се реши RC колото од примерот 7.3 со користење на елементи од множеството SimPowerSystems.

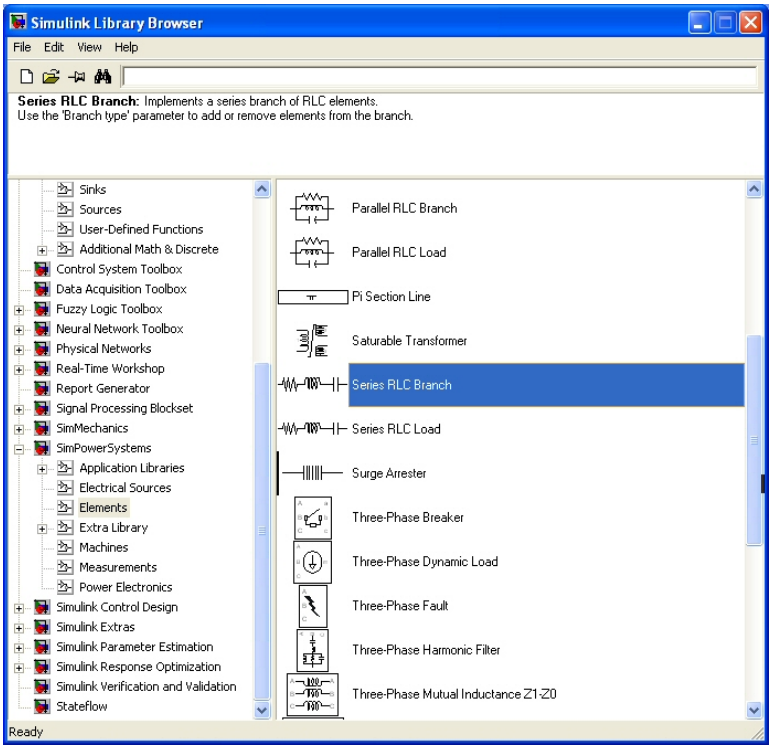
Решение

На сликата 8.3 на левата страна е селектирано множеството блокови SimPowerSystems од каде што го бираме подмножеството Elements и во него го бираме блокот Series RLC Branch кој што претставува редна RLC врска. Доколку во некој модел вметнеме ваков блок (со влечење на блокот врз прозорецот каде што ја цртаме сликата на колото), а потоа двојно кликнеме врз него ќе се појави дијалогот од сликата 8.2.

[1] Повеќе детали за примената на Simulink за решавање на покомплицирани проблеми во ЕЕС може да се видат на http://pees.etf.ukim.edu.mk/predmeti/kapp/index.htm



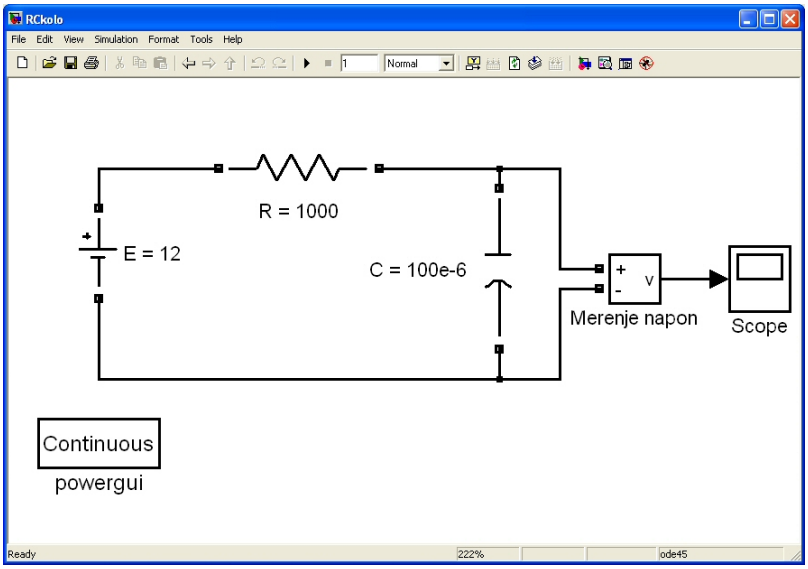
Слика 8.2. Параметри на блокот RLC



Слика 8.3. Избор на блок Series RLC Branch од множеството SimPowerSystems

Како што може да се забележи на сликата 8.2 можно е да се одбере типот на серискатa RLC гранка со селекција на една од опциите од паѓачкото мени Branch type при што може да се одбере RLC, R, L, C, RC, RL, LC и Open circuit. Тоа значи дека нашето RC коло можеме да го решиме така што во моделот ќе вметнеме една сериска RLC гранка и ќе одбереме таа да биде од типот RC. Но тоа нема да го направиме така затоа што во тој случај нема да имаме пристап до средишната точка на RC гранката (спојот помеѓу R и C) со што нема да имаме можност да го измериме напонот на кондензаротор кој се бара како решение од колото. Поради тоа, во моделот ќе внесеме две RLC гранки при што едната ќе биде од типот R, а другата од типот C и ќе ги поврземе редно. Во дијалогот за параметрите на двете гранки ќе ги внесеме вредноста на отпорноста и капацитетот, а кај кондензаторот ќе го означиме полето Set initial capacitor voltage и ќе внесеме вреднос 0 (кондензаторот е празен пред почетокот на

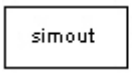
симулацијата). Изгледот на комплетниот модел е даден на сликата 8.4 и е снимен под името RCkolo.mdl.

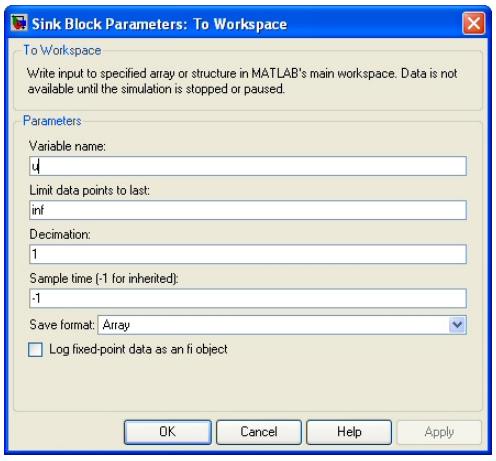


Слика 8.4. Изглед на моделот RCkolo.mdl

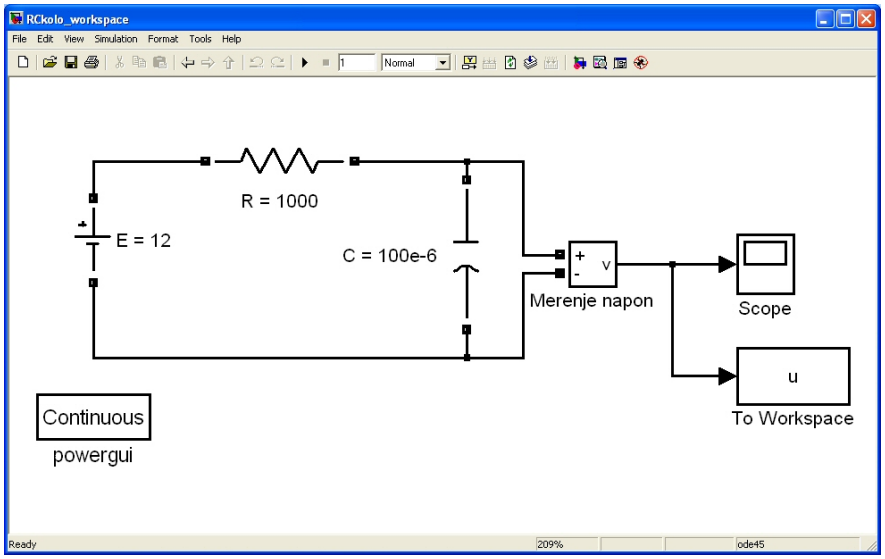
Елементот  DC Voltage Source се наоѓа во Electrical Sources од множеството блокови SimPowerSystems и тој претставува напонски генератор со константен напон (во случајов 12 V што се внесува преку дијалогот кој се добива до двојно кликнување врз симболот), додека елеметот  Voltage Measurement се наоѓа во Measurements во SimPowerSystems и тој е наменет за мерења напон во некој дел од колото и негово претворање во сигнал кој може да се прикаже на осцилоскоп. Како што може да се види на сликата 1.21 кај овие елементи стои текстот E = 12 и Merenje napon наместо DC Voltage Source и Volate Measurement. Промената на овие текстови може да се направи откако елементот ќе се внесе во моделот со едноставно пишување врз текстот кој што е автоматски внесен со внесувањето на елементот (DC Voltage Source и Voltage Measurement).

Со активирање на симулацијата резултатот на осцилоскопот е напонот на кондензаторот и тој го има истиот изглед како на сликата 7.3.

Освен тоа е можно резултатот да се добие и како вектор од броеви доколку во моделот се вметне елементот  To Workspace кој што се наоѓа во Sinks под од множеството Simulink. Со двојно кликнување врз овој елемент се добива дијалогот од сликата 8.5 во кој што во полето Variable name имаме внесено u, а во полето Save format имаме одбрано Array. Изгледот на вака дополнетиот модел е даден на сликата 8.6, а тој е снимен под името RCkolo_workspace.mdl. Со активирање на симулацијата го добиваме истиот приказ на осцилоскопот, но сега резултатот го добваме како вектор со броеви под името u кој што се наоѓа во Workspace (работната меморија на Matlab) и можеме дополнително да го обработуваме или префрламе во други програми како на пример во Excel заради цртање на график кој можеме да го форматираме многу полесно и подобро отколку во Matlab.



Слика 8.5. Параметри на блокот To Workspace



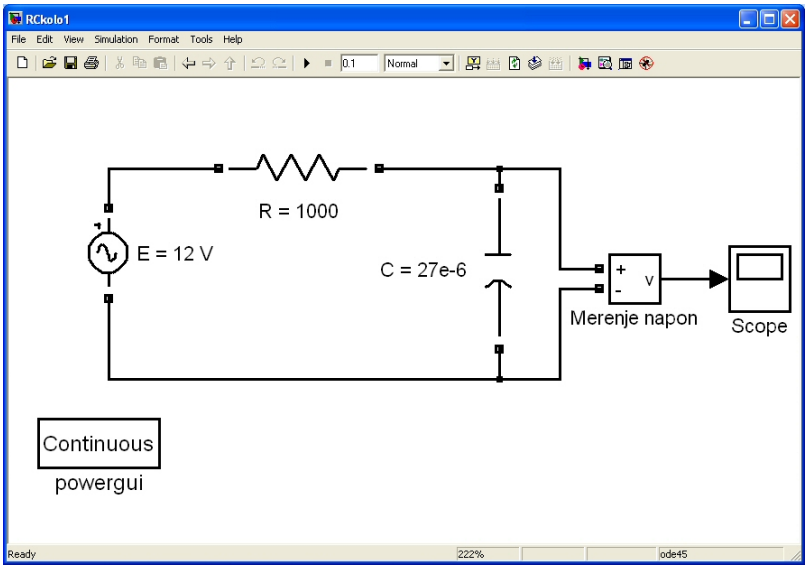
Слика 8.6. Изглед на моделот RCkolo_workspace.mdl

□ □ □

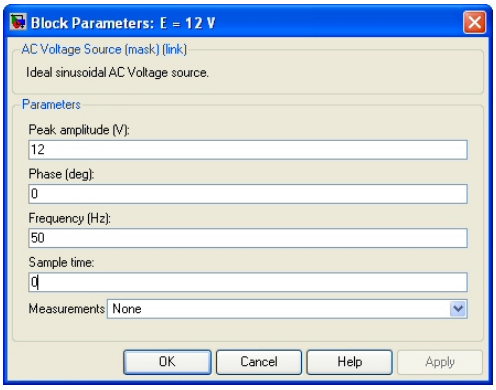
Пример 8.2. Да се реши RC колото за случајот кога е $R = 1000\,\Omega$; $C = 27\,\mu F$ и напонскиот генератор има максимална вредност на напонот $E = 12$ V при фреквенција $f = 50\ Hz$. Нека е потребно решението за бараниот напон u да го добиеме во интервалот $0 \leq t \leq 0{,}1$.

Решение

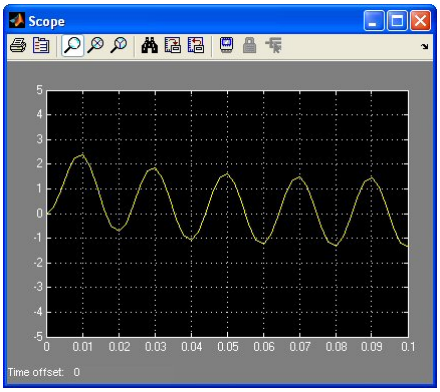
Моделот на колото за овој случај е даден во RCkolo1.mdl чиј што изглед е прикажан на сликата 8.7 каде што може да се види дека тука е вметнат напонски генератор со синусен облик кој што се наоѓа во Electrical Sources од множеството блокови SimPowerSystems. Параметрите на овој генератор се задаваат преку дијалогот прикажан на сликата 8.8 каде што за амплитудата на напонот е внесено 12, а за фревенцијата 50. Со активирање на симулацијата се добива резултат како на сликата 8.9.



Слика 8.7. Изглед на моделот RCkolo1.mdl



Слика 8.8. Параметри на блокот за синусен напонски генератор



Слика 8.9. Напон на кондензаторот во RC колото напојувано со наизменичен напон

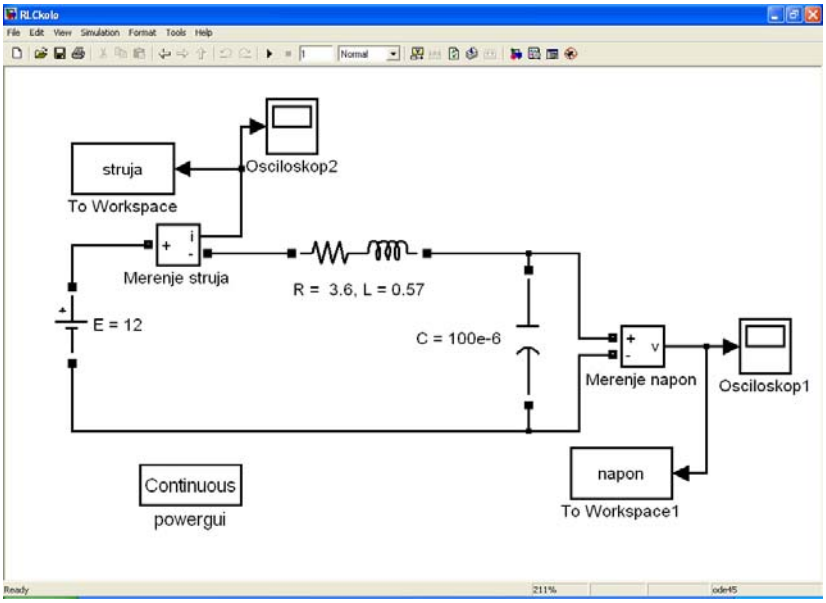
□ □ □

Пример 8.3. Едно RLC коло е поврзано на напонски генератор со константен напон од 12 V. Параметрите на колото се $R = 3{,}6\ \Omega$; $L = 0{,}57$ H и $C = 100\ \mu$F. Потребно е да се одреди обликот на струјата во калемот и напонот на кондензаторот во интервалот

$0 \leq t \leq 1$. Потоа да се одредат максималната вредност на струјата и напонот како и вкупната енергија која ќе се ослободи во вид на топлина во отпорникот.

Решение

Моделот на колото за овој случај е даден во RLCkolo.mdl чиј што изгед е прикажан на сликата 8.10. Колото е составено од две сериски RLC гранки од кои едната е од типот RL, а другата од типот C. Тука повторно не е користена само една RLC гранка за да имаме пристап до двата краја од кондензаторот за да можеме да го измериме бараниот напон. На сликата 8.10 можеме да забележиме дека е вметнат блок за мерење на струја заедно со уште еден осцилоскоп кој што ја прикажува измерената струја. Блокот за мерење на струја се се наоѓа во Measurements во SimPowerSystems. Сигналите од двата осцилоскопи се поврзани и за блокови од типот To Workspace со што во работниот простор во Matlab (Workspace) се формираат два вектори struja и napon кои што ги содржат бројните вредности на струјата и напонот.

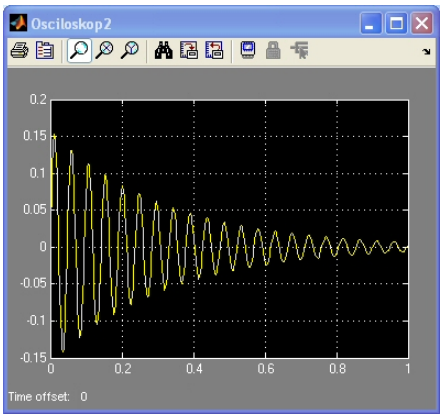


Слика 8.10. Изглед на моделот RLCkolo.mdl

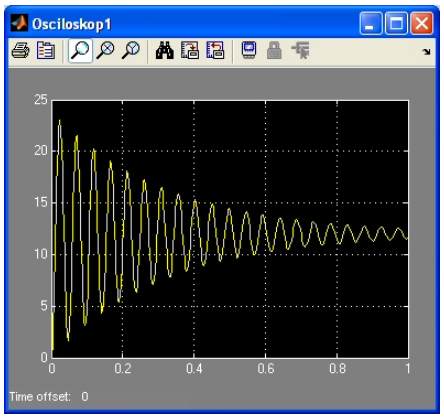
По активирањето на симулацијата на осцилоскопите ги добиваме резултатите кои се прикажани на сликата 8.11. Во исто време бројните вредности на струјата и напонот се запишани во векторите struja и napon, така што бараните максимални вредности можеме да ги добиеме на со пишување на следните наредби во командниот простор во Matlab:

```
>> max(abs(struja))
ans =
    0.1532
>> max(abs(napon))
ans =
   23.0639
```

што значи дека максималната вредност на струјата во колото изнесува 0,1532 A; додека максималната вредност на напонот на кондензаторот изнесува 23,0639 V.



а) Струја во калемот



б) Напон на кондензаторот

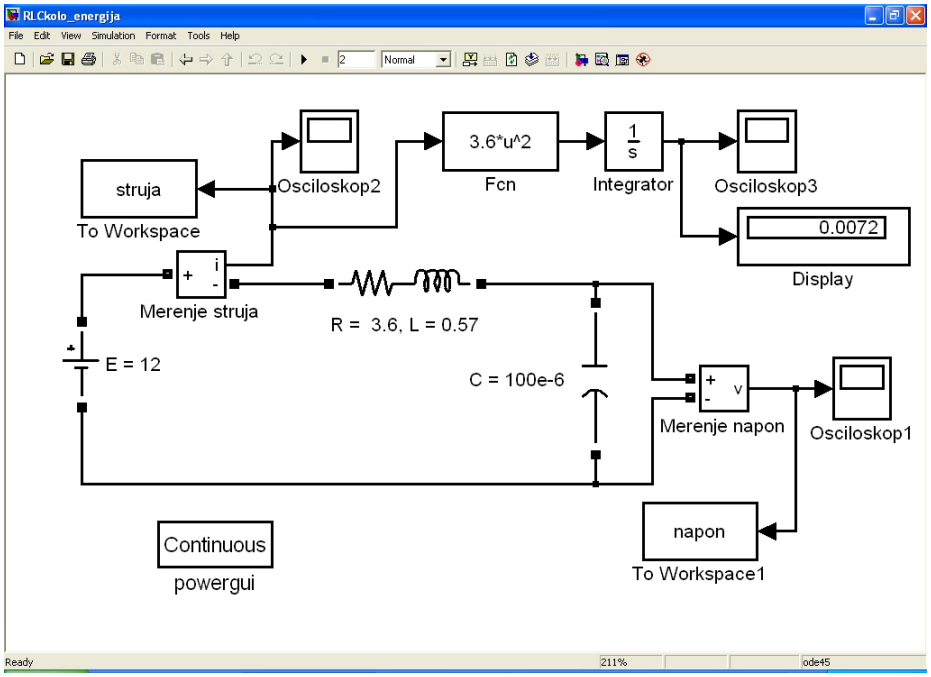
Слика 8.11. Резултати од моделот RLCkolo.mdl

Топлинската енергија ослободена во отпорникот може да се пресмета со следниот израз

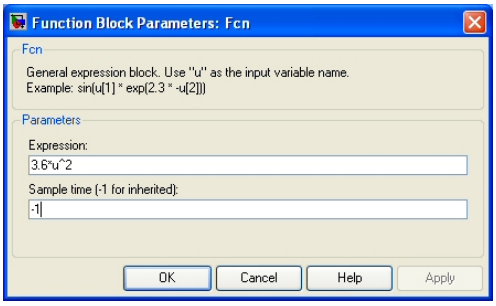
$$W = R \cdot \int_0^1 i^2(t)\,dt = 3,6 \cdot \int_0^1 i^2(t)\,dt\,, \tag{8.1}$$

кој што ние тука нема да го решаваме аналитички или нумерички, туку ќе користиме блокови од Simulink кои ќе ги направат бараните операции. Моделот кој што тоа го прави е даден во RLCkolo_energija.mdl и е прикажан на сликата 8.12.

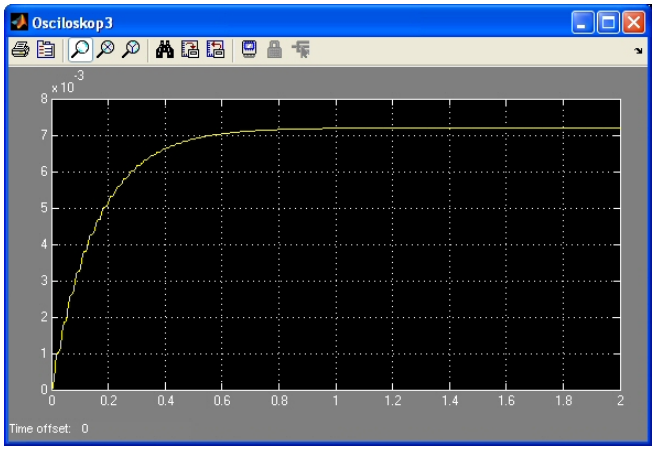
Сигналот кој што излегува од блокот за мерење на струјата прво се праќа во блокот Fcn (User-Defined Functions од библиотеката Simulink) во кој што е дефинирана математичка функција 3.6*u^2 како што тоа е прикажано на сликата 8.12 која е добиена со двојно кликнување врз блокот Fcn. Фукнцијата која што таму се пишува секогаш како параметар треба да ја има буквата u и во нашиот случај таа ја пресметува величината која што влегува во интегралот во (8.1). Потоа, сигналот кој што излегува од блокот Fcn се интегрира со блокот Integrator кој што можеме да го најдеме во Continuous од библиотеката Simulink. Вредноста на интегралот ја прикажуваме на блокот Display кој што се можеме да го најдеме во Sinks од библиотеката Simulink. Покрај тоа резултатот од интегрирањето го прикажуваме и на третиот осцилоскоп каде што ја добиваме вредноста на ослободената енергија со текот на времето, при што на крајот од симулациониот период ја добиваме вкупната ослободена топлина во отпорникот (слика 8.14). Вредноста на таа топлина е 0,0072 J и таа може да се отчита од дисплејот.



Слика 8.12. Изглед на моделот RLCkolo_energija.mdl



Слика 8.13. Дефинирање на фукција во блокот Fcn



Слика 8.14. Временска зависност на ослободената топлина во отпорникот

□ □ □

Пример 8.4. Во овој пример ќе прикажеме неколку дополнителни начини за обработка на мерните сигнали, како и за групирањето на одделни елементи во едно електрично коло користејќи го RLC колото од претходниот пример 8.3.

Решение

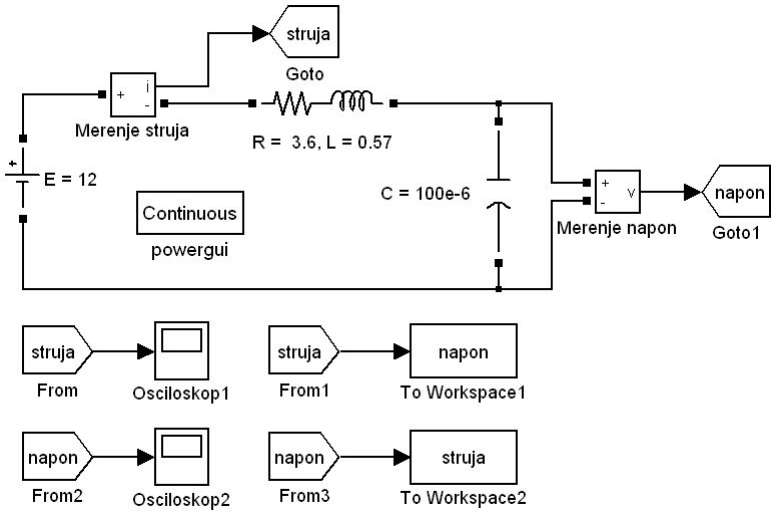
Ќе започнеме од моделот RLCkolo.mdl (чиј што изглед е прикажан на сликата 8.10) така што ќе го снимиме под името RLCkolo_goto_from.mdl. Новиот модел, кој што е прикажан на сликата 8.15, содржи два нови елементи:

- елементот Goto  кој што се наоѓа во библиотеката Signal Routing која што е дел од библиотеката Simulink,
- елементот From  кој што се наоѓа во истата библиотека како и елементот Goto.

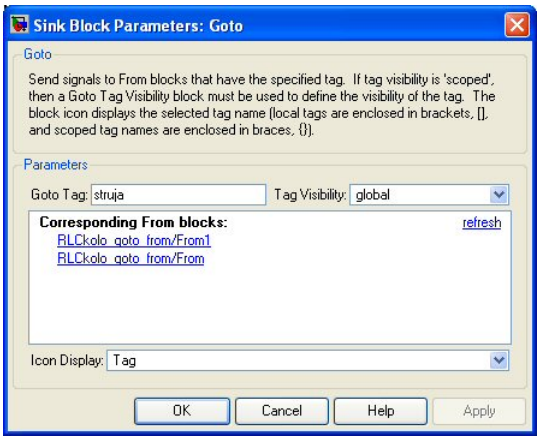
Од сликата 8.15 се забележува дека сега сигналите од измерената струја и напон не одат директно на осцилоскопите или во работниот простор туку тие се пренесуваат преку двата нови елементи. Доколку кликнеме двојно на елемент од типот Goto ќе го добиеме дијалогот од сликата 8.16, додека за елементот од типот From таквиот дијалог е прикажан на сликата 8.17.

Кај елементот од типот Goto треба да се внесе неговото име во полето Goto Tag како и областа во која што сакаме тој блок да има важност односно да биде видлив. Под Tag Visibilty можеме да избереме global, local или scoped со што можеме мерниот сигнал кој што овој блок „бежично“ го испраќа во моделот да биде видлив само локално (онаму каде што е блокот) или пак глобално во сите делови од моделот без разлика дали тој во себе содржи и подсистеми (што се тоа подсистеми ќе видиме во понатамошниот текст). Доколку сакаме измерениот сигнал да ни биде достапен насекаде во моделот, што обично е случај, тогаш се препорачува кај сите блокови од типот Goto во полето Tag Visibilty да избереме global.

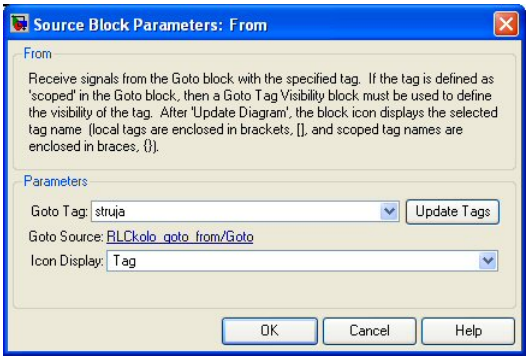
Кај елементот од типот From само се внесува името на блокот од типот Goto од кој што сакаме да го примиме сигналот. На пример, во колото од сликата 8.15 измерената струја „бежично“ сме ја испратиле насекаде во моделот под името struja, а потоа со два блока од типот From, во кои што исто така сме го навеле името struja, сме до примиле испратениот сигнал и потоа сме го проследиле до еден осцилоскоп и во во променливата struja во работниот простор. На сличен начин сме постапиле и кај мерењето на напонот, само што во тој случај сигналот е испратен под името napon. На ваков начин го добиваме истиот резултат како во колото 8.10 само што тука немаме дополнителни видливи линии со кои ги испраќаме измерените сигнали со што сме го поедноставиле целиот цртеж. Ова е од особено значење за модели во кои што има повеќе мерења чии што сигнали се користат за понатамошна обработка или приказ на повеќе места.



Слика 8.15. Изглед на моделот RLCkolo_goto_from.mdl (примена на Goto и From)



Слика 8.16. Дијалог за нагодување на блокот Goto



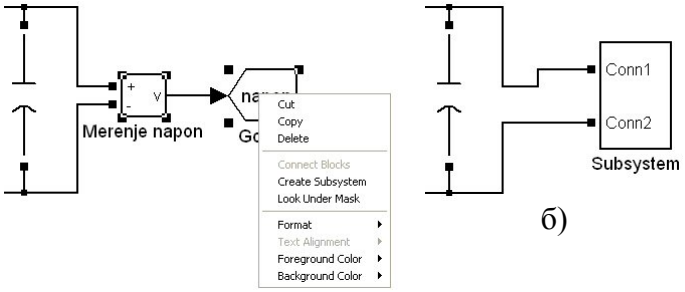
Слика 8.17. Дијалог за нагодување на блокот From

Приказот на моделот од сликата 8.15 можеме уште да го упростиме со групирање на одделни елементи во подсистеми. Доколку ги селектираме елементите Merenje napon и Goto1 и потоа кликнеме на десниот тастер од глушецот ќе добиеме едно мени како што тоа е прикажано на сликата 8.18а. Од менито ја избираме опцијата Create Subsystem со што во моделот се креира еден подсистем и тој го добива изгледот прикажан на сликата 8.18б. Доколку кликнеме двојно врз новиот подсистем ќе влеземе во него и ќе ја видиме неговата содржина како што тоа е прикажано на сликата 8.19а. Во подсистемот покрај елеметите Merenje napon и Goto1, кои ги одбравме пред да го

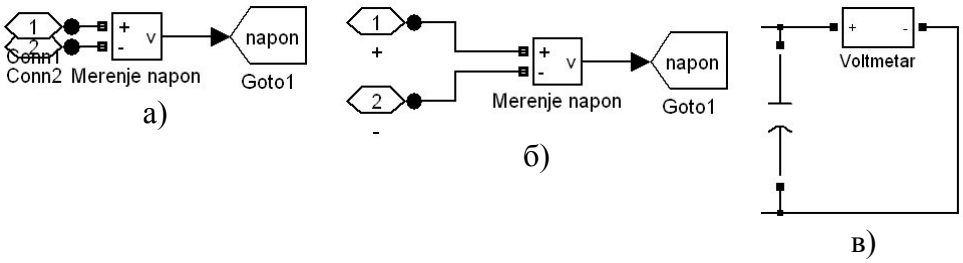
креираме, постојат и два приклучоци означени со шестоаголници и броеви во нив. Приклучоците имаат и свои имиња кои што се Conn1 и Conn2, кои што како што гледаме од сликата 8.18б, се појавуваат како имиња во надворешниот правоаголник со кој што е заграден подсистемот. Доколку двојно кликнеме на приклучокот 2 ќе го добиеме дијалогот од сликата 8.20 каде што во полето Port location on parent subsystem избираме Right. Освен тоа, имињата Conn1 и Conn2 ги менуваме во + и – со што сакаме да означиме дека тоа позитивниот и негативниот приклучок на блокот за мерење напон. Со мало преуредување на позицијата на блоковите во подсистемот го добиваме изгледот прикажан на сликата 8.19б.

На крајот, во главниот модел, ја менуваме големината на правоаголникот на подсистемот и го менуваме неговото име со што ја добиваме состојбата прикажана на сликата 8.19в.

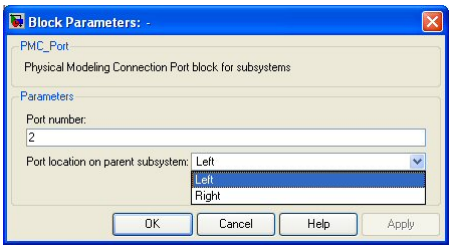
Доколку постапиме на сличен начин со блоковите Merenje struja и Goto ќе добиеме еден подсистем кој сме го означиле со името Ampermetar како што тоа е прикажано на сликата 8.21. На таа слика се гледа и подсистемот Rezultati во кои што се вметнати сите блокови поврзани со приказот на резултатите на осцилоскопи или пак во работниот простор. Изгледот на подсистемите Ampermetar и Rezultati е прикажан на сликата 8.22.



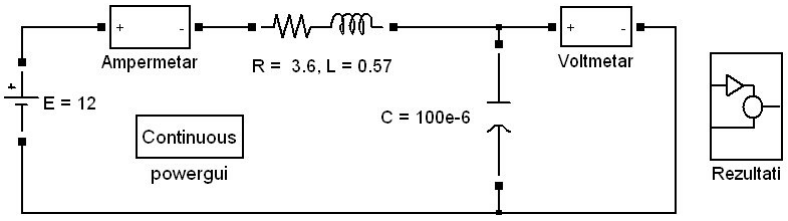
Слика 8.18. Креирање на подсистем



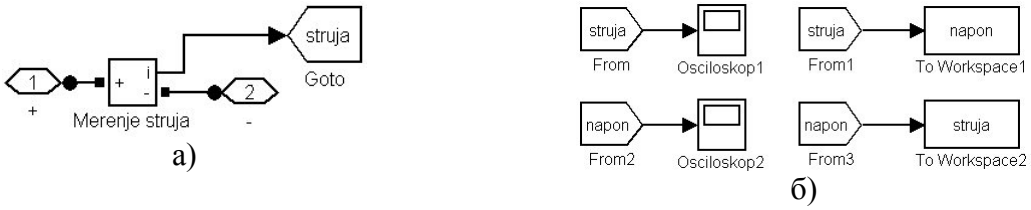
Слика 8.19. Модификација на подсистем



Слика 8.20. Параметри на приклучок во подсистем

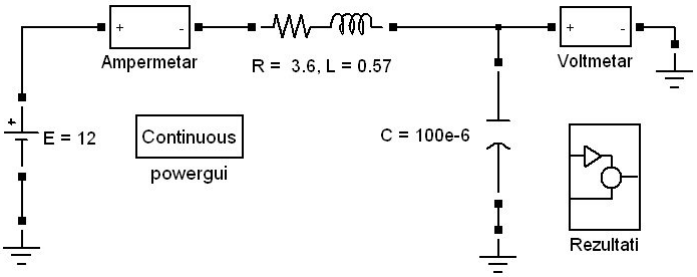


Слика 8.21. Модел на RLC коло кој што содржи подсистеми



Слика 8.22. Изглед на подсистемите а) Ampermetar и б) Rezultati

На крајот, моделот од сликата 8.21 ќе го измениме на таков начин што долната заедничка линија која што поврзува неколку елементи ќе ја замениме така што секој од тие елементи ќе го заземјиме како што тоа е прикажано на сликата 8.23 (елементот за зазамјување се наоѓа во библиотеката Elements од SimPowerSystems). Во овој случај е можно заземјувањето да го воведеме и во подсистемот Voltmetar со тоа што во него приклучокот 2 ќе го избришеме и негативниот крај од блокот за мерење на напон ќе го поврземе со земјата. Со тоа овој подсистем би имал само еден надворешен приклучок. Тоа тука не е направено и е оставено читателот сам да го проба. Конечниот модел на RLC колото е даден во RLCkolo_goto_from_podsistemi_zemja.mdl.



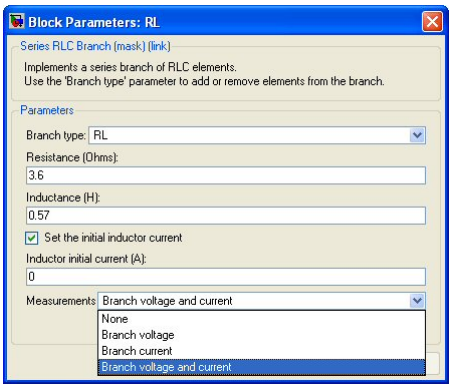
Слика 8.23. Модел на RLC коло кој што содржи подсистеми и заземјување

□ □ □

Пример 8.5. Дадено е RLC коло со параметри како во примерот 8.3. Со помош на блокот Multimeter да се изврши мерење на напоните и струите во елементите RL и C. Мерењата да се прикажат на еден 4-канален осцилоскоп, како и на еден едноканален осцилоскоп на кој што на ист екран ќе бидат прикажани напоните на двата елемента.

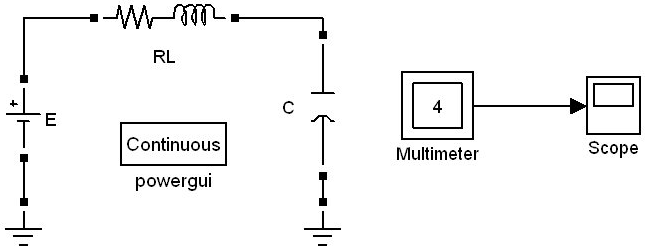
Решение

Блокот Multimeter се наоѓа во библиотеката Measurements од SimPowerSystems и тој во себе ги содржи временските промени на одредени величини од колото кои што претходно имаме избрано да се запомнат. Тоа го правиме на следниот начин: со двојно кликнување на елеметот RL го добиваме дијалогот од сликата 8.24 во кој што долниот дел од полето Measurements сме ја избрале опцијата Branch voltage and current со што на моделот сме му дале задача напонот и струјата на гранката RL да си запомни во блокот Multimeter. На сличен начин истата опција ја имаме избрано и кај елементот C.

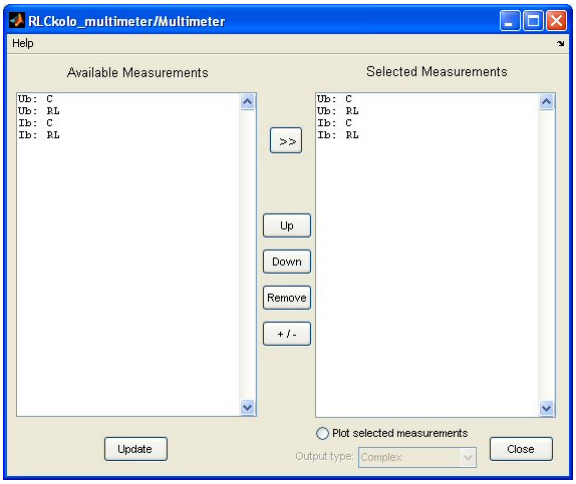


Слика 8.24. Избор на величини кои ќе бидат запомнети во блокот Multimeter

Со додавање на еден Multimeter и негово поврзување со осцилоскоп го добиваме моделот RLCkolo_multimeter.mdl кој што е прикажан на сликата 8.25. Со двојно кликнување на блокот Multimeter до добиваме прозорецот прижан на сликата 8.26 кој што се состои од два дела. Во левиот дел од прозорецот се дадени сите сигнали кои претходно сме избрале да бидат мерени, тоа во овој случај се: Ub: C (напон на гранката C), Ub: RL (напон на гранката RL), Ib: C (струја во гранката C) и Ib: RL (струја во гранката RL). Со кликнување на тастерот означен со >> можеме овие сигнали да ги префрлиме на десната страна од прозорецот со што сме избрале тие да излезат од мултиметарот. Во овој случај сме ги избрале сите сигнали, со што на мултиметарот ни се појавил бројот 4 со што ни е кажано колку сигнали излегуваат од него. Редоследот на излезните сигнали можеме да го менуваме со кликнување на тастерите Up или Down откако претходно ќе селектираме еден излезен сигнал.



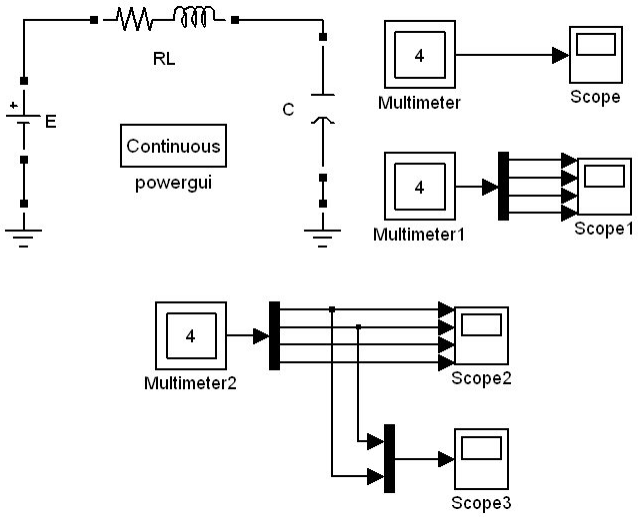
Слика 8.25. Модел на RLC коло кој што содржи Multimeter



Слика 8.26. Избор на сигнали од блокот Multimeter

За моделот од сликата 8.25 на осцилоскопот се добива приказот кој што е даден на сликата 8.27 каде што гледаме дека сите 4 сигнали се прикажани на еден екран кој што не е многу прегледен. За да ги одделиме сигналите и да ги прикажеме на 4-канален осцилоскоп, како што се бараше во примерот, користиме демултиплексер како што е прикажано на сликата 8.28. Демултиплексерот (Demux) се наоѓа во библиотеката Signal Routing во Simulink и тој на сликата 8.28 е поврзан помеѓу блоковите Multimeter1 и Scope1. Со двојно кликнување на демултиплексерот добиваме дијалог во кој што можеме да го внесеме бројот на излезните сигнали кои што сакаме да се појава по него. Во овој случај сме избрале 4 со што сигналот кој што излегува од мултиметарот е „расцепен“ на 4 сигнали кои што одделно ги прикажуваме на еден 4-канален осцилоскоп чиј приказ е даден на сликата 8.29.

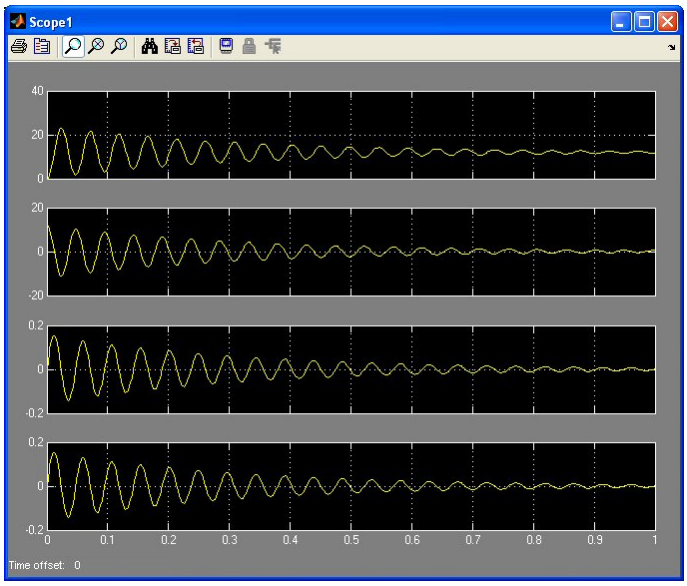
Слика 8.27. Четири сигнали на еден екран



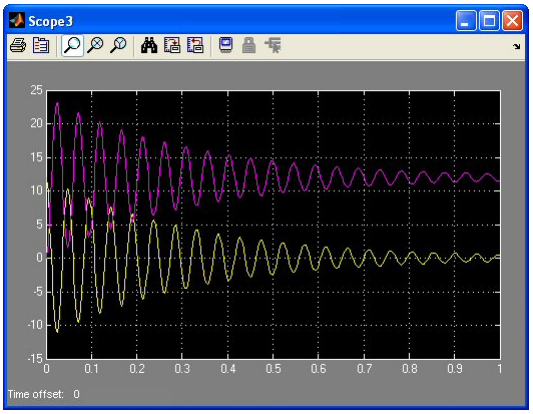
Слика 8.28. Демултиплексирање и мултиплексирање на сигнали

Бидејќи се бараше и прикажување на напоните на двата елемента на еден едноканален осцилоскоп на ист екран потребно е два од четирите сигнали кои што излегуваат од демултиплексерот да бидат мултиплексирани во еден сигнал. Тоа ќе го направиме со помош на елементот мултиплексер (Mux) кој што се наоѓа во библиотеката Signal Routing во Simulink и тој на сликата 8.28 е поврзан помеѓу блоковите Multimeter2 и Scope3. Со двојно кликнување на мултиплексерот добиваме дијалог во кој што можеме да го внесеме бројот на влезните сигнали кои што сакаме да се појават како еден заеднички сигнал на неговиот излез. Во овој случај сме ги избрале првите два сигнали затоа што тие се напоните на елементите според редоследот на

сигнали во мултиметарот. Едноканалниот осцилоскоп има приказ кој што е даден на сликата 8.30.



Слика 8.29. Четири сигнали на четири одделни екрани (Scope 1 и Scope2)



Слика 8.30. Напони на гранките RL и C прикажани на еден екран (Scope3)

□ □ □

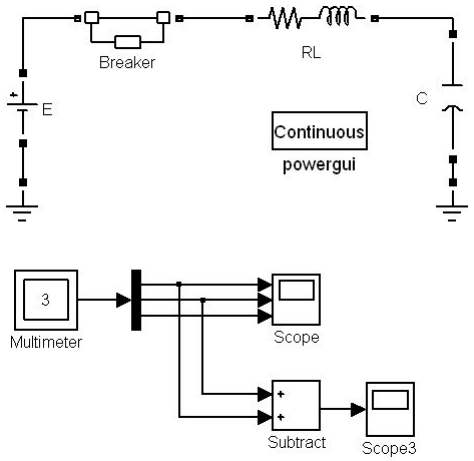
Пример 8.6. Во колото од примерот 8.5 редно со гранката RL да се постави прекинувач кој што во почетокот е вклучен, а потоа се исклучува во временскиот момент 0,3 s и повторно се вклучува во 0,4 s. Да се прикажат облиците на струјата и напоните на гранките RL и C на еден 3-канален осцилоскоп. Исто така на друг едноканален осцилоскоп да се прикаже збирот од напоните на гранките RL и C.

Решение

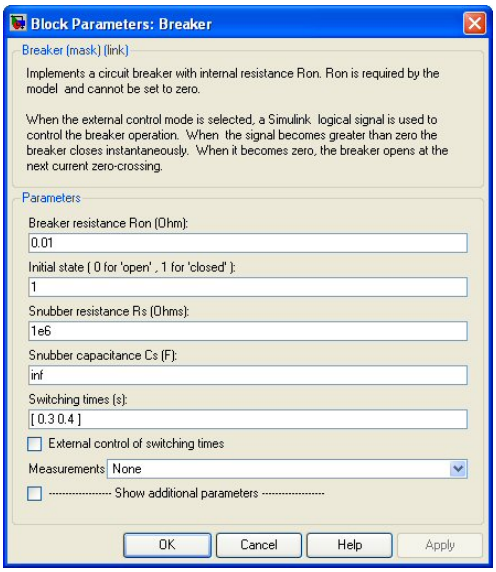
Моделот на колото според кое што ќе го симулираме исклучувањето и вклучувањето на прекинувачот во RLC колото е RLCkolo_prekinuvac.mdl и неговиот изглед е даден на сликата 8.31. Моделот е идентичен како моделот во примерот 8.5 со тоа што сега редно со гранката RL имаме додадено прекинувач.

Прекинувачот кој што е вметнат во колото се наоѓа во библиотеката Elements во SimPowerSystems. Тој во овој случај мора да има паралелен отпорник со него затоа што се наоѓа во серија со калем. Вредноста на тој отпорник се внесува во полето Snubber

resistance и таа обично е голема вредност која што тука е еднаква на 1 MΩ. Со двојно кликнување врз него се добива дијалогот од сликата 8.32 каде што се задени параметрите на прекинувачот. Како што се гледа на сликата опцијата Externer control of switching times не е селектирана со што ни се дава можност сами да зададеме времиња во кои што прекинувачот ќе ја промени својата состојба. Почетната состојба на прекинувачот се задава во полето Initial state (во овој случај тоа е 1, што значи дека тој е вклучен) додека временските моменти во кои што тој ќе ја промени состојбата се дадени во полето Switching times каде што се задава вектор со временски моменти во секунди. Тука е зададен векторот [0.3 0.4] што значи дека прекинувачот два пати ќе ја промени својата состојба и тоа во временските моменти $t_1 = 0{,}3$ s и $t_2 = 0{,}4$ s.



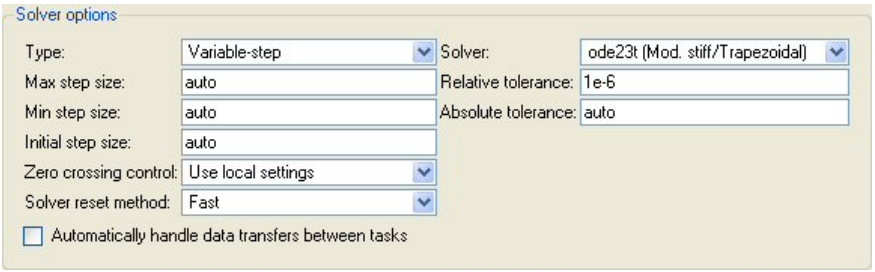
Слика 8.31. Модел на RLC коло кој што содржи прекинувач



Слика 8.32. Параметри на блокот за прекинувач

Во колата кои што содржат прекинувачи препорачливо е да се користи методата od23t за решавање на преодниот процес. За тоа да го направиме е потребно на прозорецот RLCkolo_prekinuvac.mdl од менито Simulation да избереме Configuration parameters со што ја добиваме сликата 8.33 каде што во полето Solver избираме ode23t. Во спротивно во командниот простор во MATLAB ќе ја добиеме следната порака:

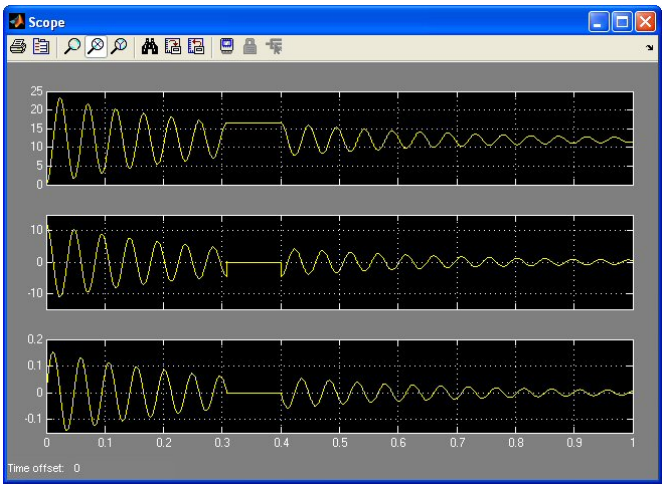
Warning: You have required continuous-time simulation of a system containing switches or nonlinear elements. The ode23t variable-step stiff solver is recommended in order to get best accuracy and simulation performance. ...



Слика 8.33. Избор на метод за решавање на преодниот процес

Резултатите од симулацијата ги добиваме со помош на елементот Multimeter кој што се користи на начин опишан во претходниот пример 8.5. Во овој случај од него излегуваат три сигнали кои и тоа струјата во колото и напоните на гранките RL и C. Нивниот изглед е прикажан на сликата 8.34, при што прво е даден напонот на гранката C, потоа напонот на гранката RL и на крајот е дадена струјата.

На сликата 8.34 јасно се гледа акцијата на прекинувачот во колото, каде во одреден временски период кога тој делувал има нагла промена во изгледот на сигналите. Тука е потребно да се нагласи дека иако на прекинувачот му дадовме налог да ја исклучи струјата во моментот t_1 = 0,3 s таа не е прекината во тој момент туку симулацијата продолжила со вклучен прекинувач до првиот нареден момент во кој што струјата добила вредност нула. Тоа е во согласност со начинот на работа на прекинувачите кои што не можат да ја прекинат струјата се додека таа самата не стане еднаква на нула. Во спротивниот случај, кога е даден налог за вклучување на прекинувачот тоа се случило моментално точно во моментот кога тоа сме го побарале (t_2 = 0,4 s).

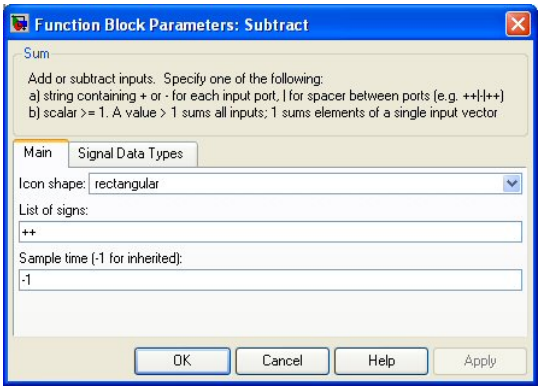


Слика 8.34. Облици на напоните и струјата во RLC коло кое што содржи прекинувач

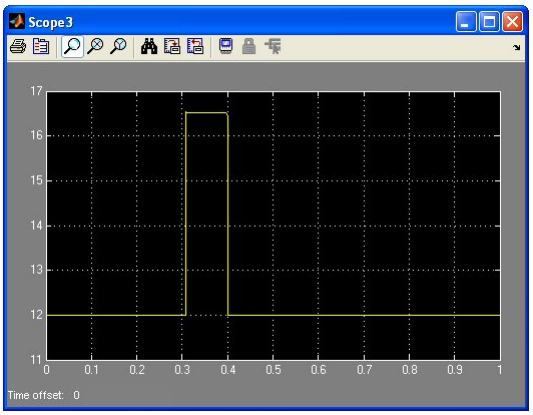
Сумирањето на напоните го правиме со помош на блокот Substract кој што се наоѓа во библиотеката Math Operations во Simulink. Со двојно кликнување врз овој блок го добиваме дијалогот од сликата 8.35 каде што во полето List of signs имаме внесено ++ со што на блокот сме му кажале дека сакаме да имаме два влезни сигнали кои што тој треба да ги помножи со +1 и да ги собере, при што резултатот ќе го даде на својот излез (доколку сакавме од вториот сигнал да го одземеме првиот сигнал тогаш во во полето List of signs ќе требаше да внесеме –+). На влезот од овој блок се дадени првиот

и вториот сигнал од мултиметарот затоа што во него претходно по тој редослед сме ги задале напонот на гранката C и напонот на гранката RL. Излезниот збирен сигнал е прикажан на едноканален осцилоскоп чиј што приказ е даден на сликата 8.36.

Од обликот на сигналот на сликата 8.36 се забележува дека збирот на двата напони е еднаков на 12 V, што е еднакво на напонот на генераторот, освен во периодот со отворен прекинувач кога тој збир е околу 16,5 V. На читателот се остава да го протолкува ова зголемување во збирот на двата напона.



Слика 8.35. Нагодување на блокот Substract



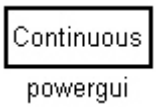
Слика 8.36. Сума на напоните на гранките C и RL во RLC коло со прекинувач

□ □ □

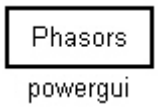
Пример 8.7. Дадено е RLC коло со параметри како во примерот 8.3 во кое што напонскиот генератор е заменет со друг кој што е простопериодичен и има напон со ефективна вредност од 220 V и фреквенција 50 Hz. Да се реши колото во простопериодичен принуден режим (со помош на фазори) и да се пресметаат привидните моќности на сите елементи.

Решение

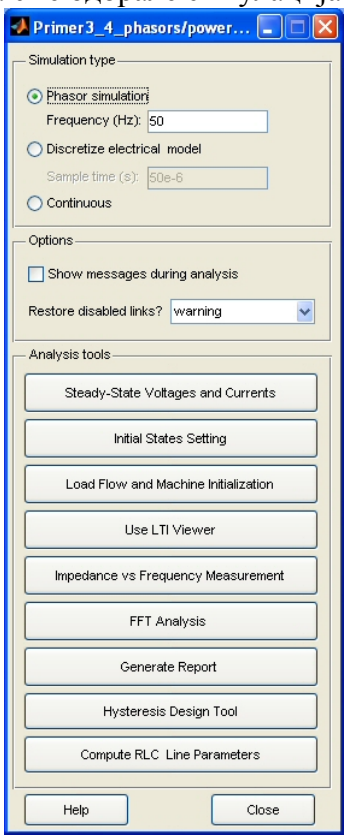
Во овој случај симулацијата нема да ја правиме во временски домен туку во доменот на фазори (решавање на коло за наизменична струја со комплексни броеви). За тоа да го напрвиме кликнуваме два пати врз објектот



со што го добиваме дијалогот од сликата 8.37 каде што ја одбираме опцијата Phasor simulation и внесуваме фреквенција од 50 Hz. Со тоа во моделот го добиваме следниот објект



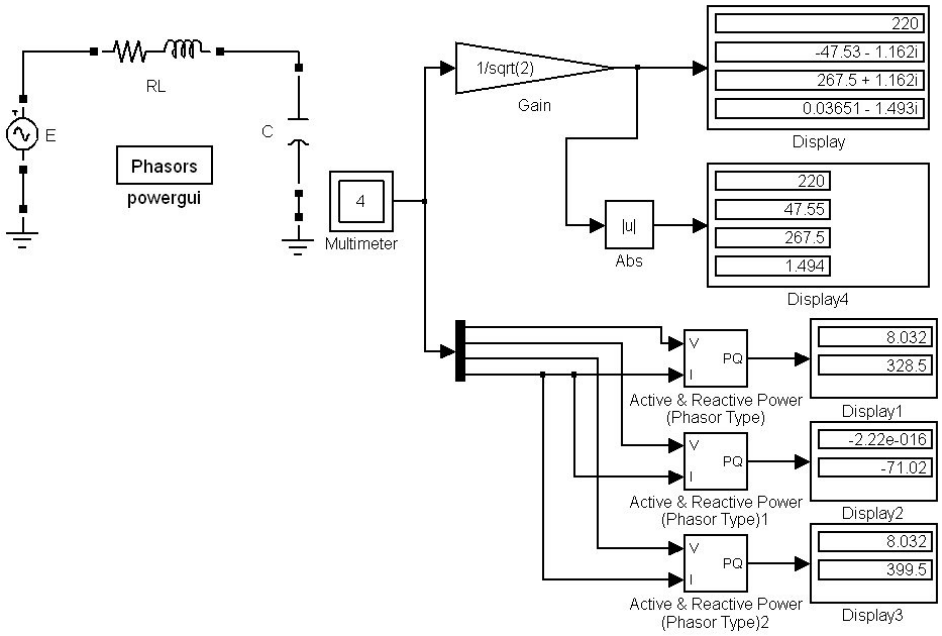
со што ни е сигнализирано дека сме одбрале симулација со помош на фазори.



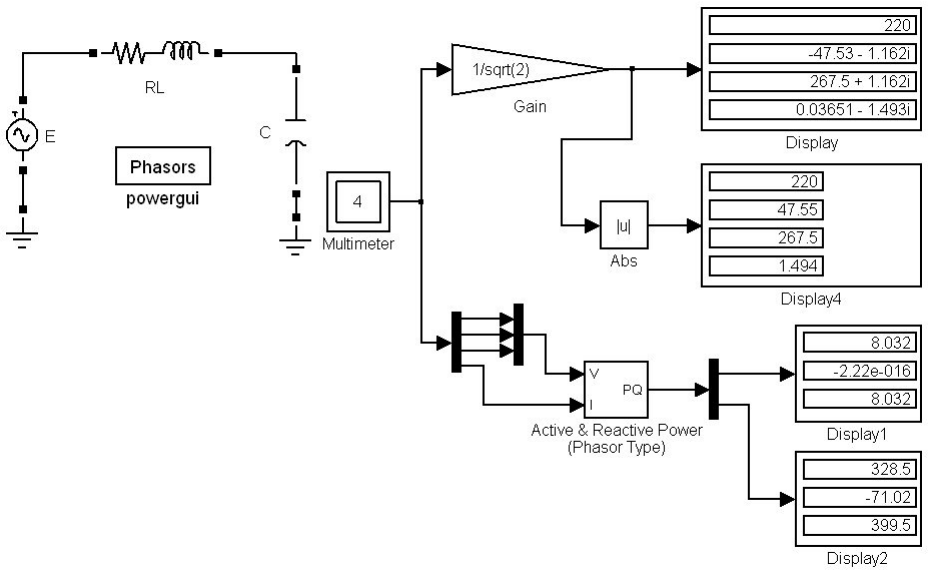
Слика 8.37. Избор на видот на симулација

Моделот на колото е даден во датотеката RLCkolo_fazori.mdl и тој е прикажан на сликата 8.38. Во колото напонскиот извор е заменет со AC Voltage Source чии што параметри се дефинираат како во примерот 8.2 само што тука за максималната вредност на напонот на изворот задаваме 220*sqrt(2), а освен тоа сме избрале да се прави мерење на неговиот напон кој што потоа можеме да го отчитаме од мултиметарот. Во мултиметарот имаме селектирано четири сигнали при што нивниот редослед е: 1) напон на генераторот, напон гранката C, напон на гранката RL и струја во колото. Овие четири сигнали, по излезот од мултиметарот, ги множиме со коефициент $1/\sqrt{2}$ со помош на блокот Gain кој што го наоѓаме во библиотеката Math Operations во Simulink. Ова е неопходно затоа што во Simulink фазорите се однесуваат на максималните вредност на наизменичните величини, а не на ефективните на кои што ние сме навикнати. Резултатот е прикажан на дисплеј каде што се прикажани четири комплексни вредности (дисплејот е истегнат по вертикала за да покаже повеќе од една вредност) за сите четири величини одделно. Освен тоа, четирите комплексни величини се пратени преку блокот Abs на друг дисплеј каде што се прикажани нивните апсолутни вредности, т.е. нивните ефективни вредности (блокот Abs се наоѓа во библиотеката Math Operations во Simulink).

Четирите излезни сигнали од мултиметарот ги демултиплексираме со што добиваме четири одделни сигнали (три напони и една струја) кои што понатаму ги користиме за да ги пресметаме активните и реактивните моќности на елементите во колото. Тоа го правиме со помош на блокот Active & Reactive Power (Phasor Type) кој што се наоѓа во библиотеката Phasor Library која што е дел од библиотеката Extra Library од SimPowerSystems. Тој блок има два влезни комплексни сигнали за напон и струја и на излезот дава два реални броеви кои што се едкави на активната и реактивната моќност пресметани со помош на влезниот напон и струја. Со помош на три такви блока ги пресметуваме активната и реактивната моќност на генераторот, на кондензаторот и на калемот. Тие се прикажани на три одделни дисплеја.



Слика 8.38. Симулација со помош на фазори



Слика 8.39. Симулација со помош на фазори (втор начин на обработка на излезните сигнали)

На сликата 8.39 е прикажан моделот RLCkolo_fazori1.mdl во кој што на поинаков начин се пресметани моќностите на елементите. Таму најпрво четирите мерни сигнали од мултиметарот се разделени на четири одделни сигнали, а потоа првите три

(напоните) повторно се мултиплексирани во еден сигнал со три компоненти. На таков начин имаме два сигнали од кои што едниот содржи три напони, а другиот содржи една струја. Овие два сигнали ги носиме до блокот за пресметка на активните и реактивните моќности кој што во овој случај пресметува три активни и три реактивни моќности затоа што влезниот сигнал за напони содржи три вредности. На крајот излезот од него го демултиплексираме на два сигнали од кои првиот ги содржи трите активни моќности на елементите, а вториот сигнал ги содржи трите реактивни моќности.

□ □ □

Пример 8.8. Да се направи модел на идеален напонски генератор со двоен експоненцијален облик кој што се користи за моделирање на атмосферските празнења. За ваквите генератори е познато дека можат да се опишат со следната релација

$$u(t) = U_{\mathrm{M}} \cdot \left(e^{-at} - e^{-bt} \right), \tag{8.2}$$

при што најчесто наместо коефициентите a и b се користат номиналното време на траењето на челото T_{C} и номиналното време на траење на грбот T_{Z} кои може приближно да се пресметаат со помош на релациите (8.3) и (8.4)

$$T_{\mathrm{C}} = \frac{3,24}{b}, \tag{8.3}$$

$$T_{\mathrm{Z}} = \frac{0,69}{a}. \tag{8.4}$$

Максималната вредност на напонот даден со изразот (8.2) изнесува

$$U_{\mathrm{m}} = U_{\mathrm{M}} \cdot \left[\left(\frac{a}{b} \right)^{\frac{-a}{a-b}} - \left(\frac{a}{b} \right)^{\frac{-b}{a-b}} \right]. \tag{8.5}$$

При изработката на моделот да се земе дека се познати следните параметри U_{M} = 100 kV; T_{C} = 1,2 μs и T_{Z} = 50 μs. Нека ваквиот напонски генератор напојува едно RC коло со следните параметри: R = 100 Ω и C = 10 nF. На еден осцилоскоп да се прикажат облиците на напоните на самиот генератор и на кондензаторот.

Потоа моделот да се модифицира така што ќе биде можно наведените параметри да му се зададат интерактивно од страна на корисникот.

Решение

Користејќи ги изразиот (8.3) и (8.4) со помош на зададените параметри пресметуваме

$$a = \frac{0,69}{T_{\mathrm{Z}}} = \frac{0,69}{50 \cdot 10^{-6}} = 1,38 \cdot 10^{4},$$

$$b = \frac{3,24}{T_{\mathrm{C}}} = \frac{3,24}{1,2 \cdot 10^{-6}} = 2,7 \cdot 10^{6},$$

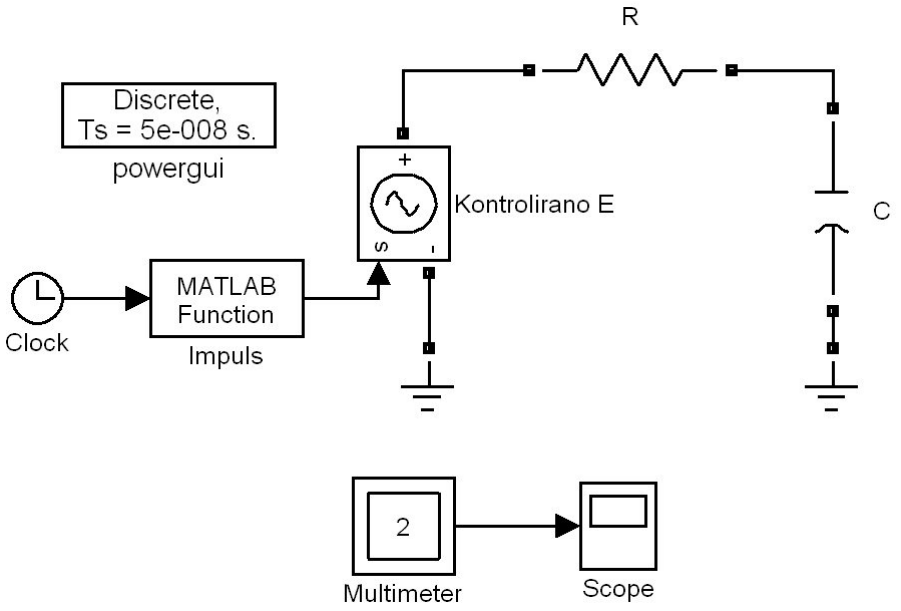
што значи во конкратниот случај е потребно да направиме напонски генератор чиј што облик на напонот е даден со изразот

$$u(t) = 100 \cdot 10^{3} \cdot \left(e^{-1,38 \cdot 10^{4} \cdot t} - e^{-2,7 \cdot 10^{6} \cdot t} \right). \tag{8.6}$$

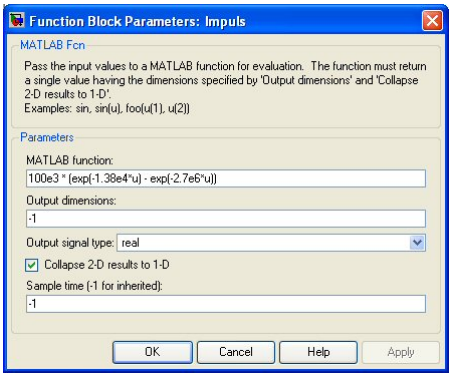
Моделот на колото е даден во датотеката dvojno_exp.mdl а неговиот изглед е прикажан на сликата 8.40. Напонскиот генератор во колото е контролиран со сигнал кој што зависи од времето и тој може да се најде во библиотеката Electrical Sources од библиотеката SimPowerSystems под името Controlled Voltage Source.

Како што се гледа на самата слика, ваквиот генератор има три приклучоци. Два од нив, означени со знаците + и –, ја имаат истата намена како и кај другите напонски генератори кои што до сега сме ги користеле, односно тоа се краеви на напонскиот генератор со кои што тој се приклучува во колото и низ кои тече струја одредена со состојбата во самото коло.

Третиот приклучок е приклучок преку кој може програмски да се задава вредноста на напонот на генераторот. Тука тој ќе го користиме за дефинирање на релацијата (8.6). Најнапред во моделот вметнуваме часовник кој што се наоѓа во библиотеката Sources во Simulink. Часовникот на својот излез го дава симулационото време кое што го испраќаме во блокот MATLAB Function (тој се наоѓа во библиотеката User-Defined Functions во Simulink). Со двојно кликнување врз него го добиваме дијалогот од сликата 8.41 каде што во полето MATLAB Function го имаме внесено изразот 100e3 * (exp(-1.38e4*u) - exp(-2.7e6*u)) кој што е идентичен со изразот (8.6) со таа разлика што t е заменето со u затоа што влезниот сигнал во Simulink е секогаш означен со буквата u. Излезниот сигнал од блокот MATLAB Function оди во контролниот влез од контролираниот напонски генератор.

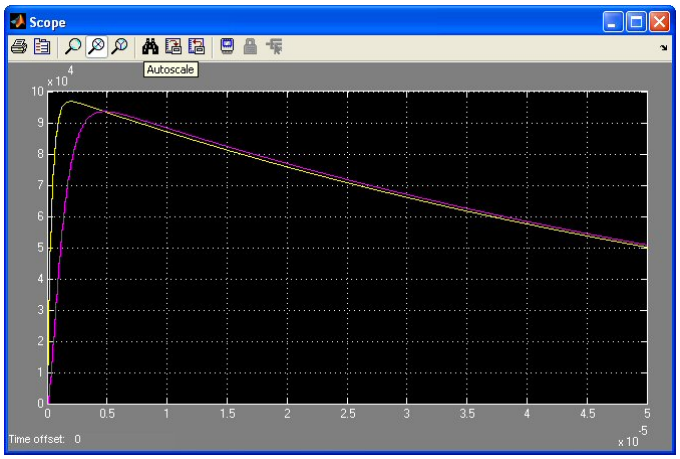


Слика 8.40. Модел на двојно-експоененцијален напонски генератор



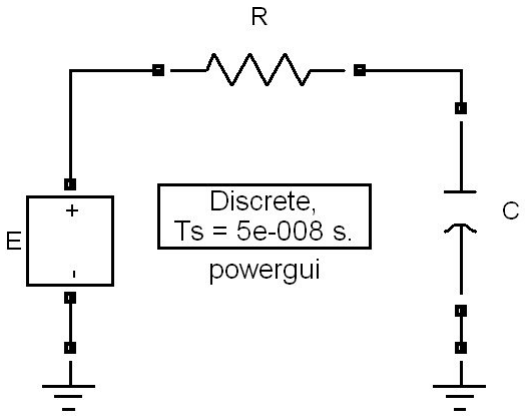
Слика 8.41. Задавање на двојно-експоненцијалниот облик на напонот

Во колото е додадена и една RC гранка (составена од посебен отпорник и кондензатор), како и мултиметар во кој што на излезот се одбрани напонот на генераторот и напонот на кондензаторот (за мерења со помош на мултиметар видете го примерот 8.5). Изгледот на сигналите прикажани на осцилоскопот е прикажан на сликата 8.42.

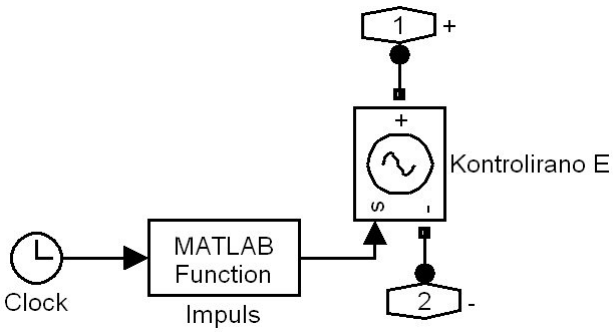


Слика 8.42. Напони на двојно-експоненцијалниот генератор и на кондензаторот

Доколку од моделот на сликата 8.40 направиме подсистем кој што ќе ги содржи болковите „Clock“, „Impuls“ и „Kontrolirano E“ моделот ќе го добие изгледот прикажан на сликата 8.43, а содржината на формираниот подсистем е дадена на сликата 8.44.



Слика 8.43. Модел на двојно-експоененцијален напонски генератор сместен во подсистем

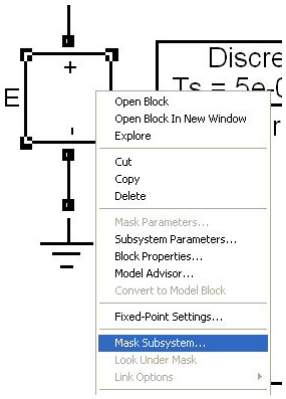


Слика 8.44. Подсистем кој што го содржи двојно-експоненцијалниот напонски генератор

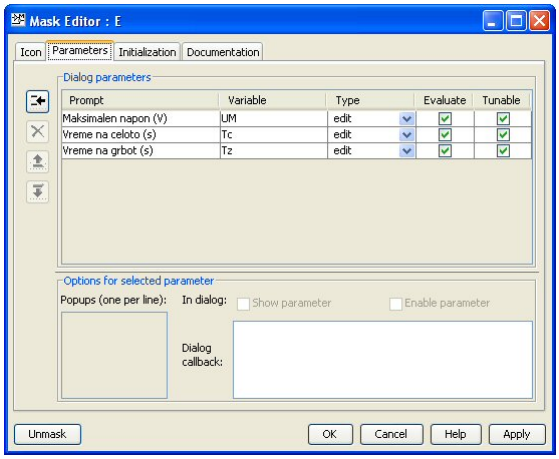
Доколку го селектираме подсистемот од сликата 8.43 и потоа над него кликнеме со десниот тастер од глушецот ќе го добиеме менито од сликата 8.45 од кое ја избираме

опцијата Mask Subsystem со што на дадениот подсистем креираме маска т.е. прозорец-форма преку кој можеме да задаваме некои параметри за самиот подсистем. Маската на подсистемот можеме да ја форматираме преку прозорецот прикажан на сликите 8.46-8.48.

На сликата 8.46 е прикажан делот со наслов Parameters каде што гледаме дека имаме три редици со кои што сме дефинирале дека сакаме преку маската да внесеме три параметри. Секој параметар има свој опис и име на променливата во која што тој ќе биде внесен. Описот на параметрите е даден во колоната Prompt додека името на променливата е дадено во колоната Variable. На сликата гледаме дека како параметри сме ги дефинирале максималниот напон, времето на челото и времето на грбот кои што ќе бидат внесени во променливите UM, Tc и Tz.



Слика 8.45. Креирање на маска

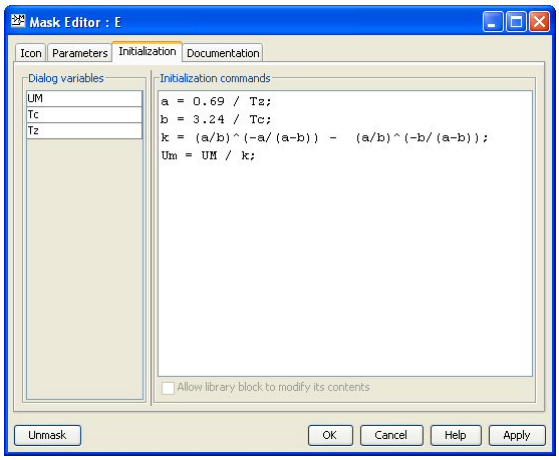


Слика 8.46. Параметри на маската

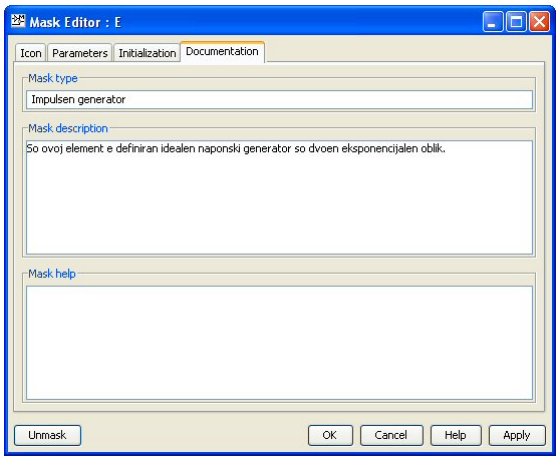
На сликата 8.47 е прикажан делот со наслов Initialization каде што има еден прозорец во кој што може да се внесуваат команди на ист начин како што тоа се прави во командниот простор во самиот Matlab. На левата страна од тој прозорец имаме листа на променливи кои ни се на располагање и кои можеме да ги сфатиме како локален работен простор сличен со главниот работен простор во Matlab. Тука всушност се дадени променливите кои ние претходно ги дефиниравме. Во прозорецот со команди најпрво ги пресметуваме a и b со помош на релациите (8.3) и (8.4), а потоа ја пресметуваме величината Um со помош на релацијата (8.5).

Описот на маската го внесуваме во делот со наслов Documentation (слика 8.48) каде што можеме да напишеме текст со кој што на корисникот накратко ќе му

објасниме која е намената на блокот кој што е вметнат во подсистемот за кој што е дефинирана предметната маска.



Слика 8.47. Пресметки при иницијализација на маската

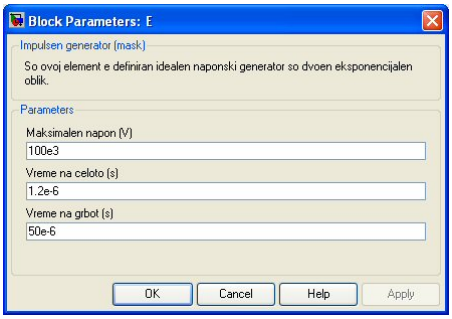


Слика 8.48. Опис на маската

Од кога сме ја дефинирале маската со двојно кликнување врз подсистемот ќе го добиеме прозорецот прикажан на сликата 8.49 каде што ги гледаме параметрите на двојно-експоненцијалниот генератор. Доколку, пак, сакаме да влеземе во самиот подсистем, односно да ја добиеме сликата 8.44, треба да го селектираме подсистемот а потоа да кликнеме на десниот тастер од глушецот и од менито да ја избереме опцијата Look Under Mask. Повторното уредување на веќе дефинирана маска се прави така што од менито ќе ја избереме опцијата Edit Mask.

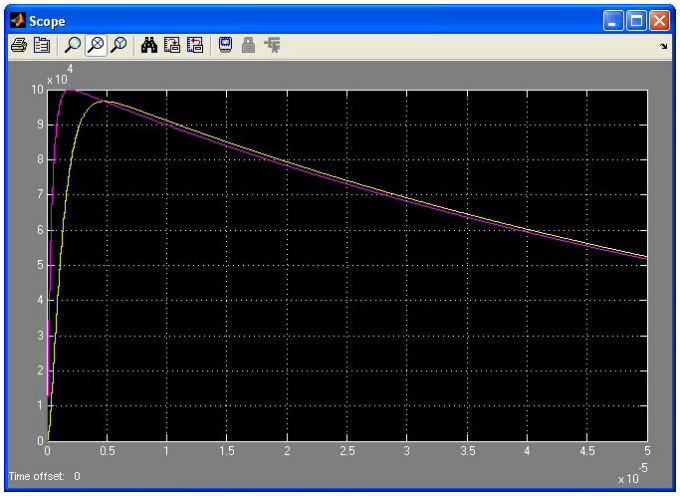
Во овој случај во полето MATLAB Function од блокот Impuls во подсистемот го имаме внесено изразот Um * (exp(-a*u) - exp(-b*u)) каде што фигурираат променливите Um, a и b кои претходно сме ги пресметале во самата маска.

Комплетниот модел со сите погоре наведени нагодувања е даден во датотеката dvojno_exp_maska.mdl.



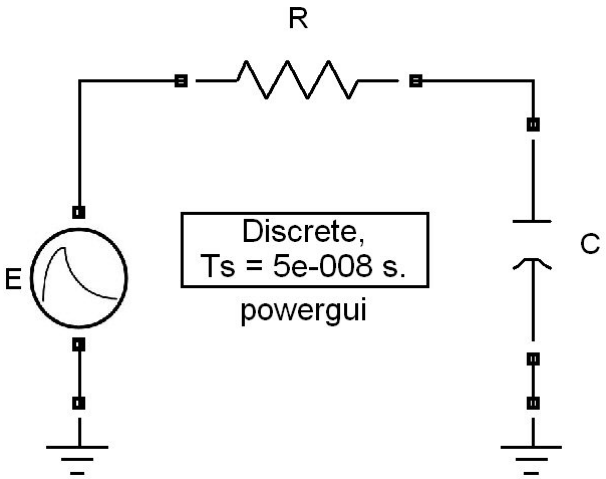
Слика 8.49. Конечен изглед на маската за двојно-експоненцијалниот генератор

Со активирање на симулацијата сигналите на осцилоскопот го добиваат обликот кој што е прикажан на сликата 8.50. На оваа слика, за разлика од сликата 8.42, максималната вредност на напонот на генератор изнесува точно 100 kV затоа што во овој случај во самата маска ја искористивме и релацијата (8.5) со која што се пресметува максималната вредност на напонот.

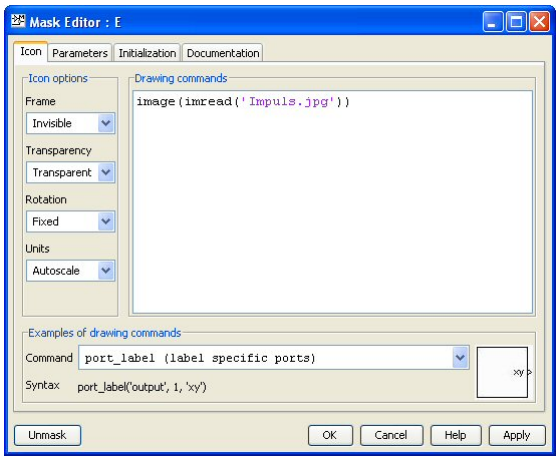


Слика 8.50. Напони на двојно-експоненцијалниот генератор и на кондензаторот (влезот е дефиниран преку маска)

На крајот, во датотеката dvojno_exp_maska_slika.mdl, е прикажан истиот модел само што на подсистемот од претходниот модел му е поставена посебна сликичка како што тоа е прикажано на сликата 8.51. Сликичната е претходно нацртана и сместена во датотеката Impuls.jpg, а таа е вчитана во маската на начин прикажан на сликата 8.52 каде што се забележува дека во делот со наслов Icon е напишана командата image(imread('Impuls.jpg')) а лево од неа во Frame е одбрано Invisible.



Слика 8.51. Модел на двојно-експоененцијален напонски генератор сместен во подсистем со посебна сликичка



Слика 8.52. Дефинирање на сликата на маската

□ □ □

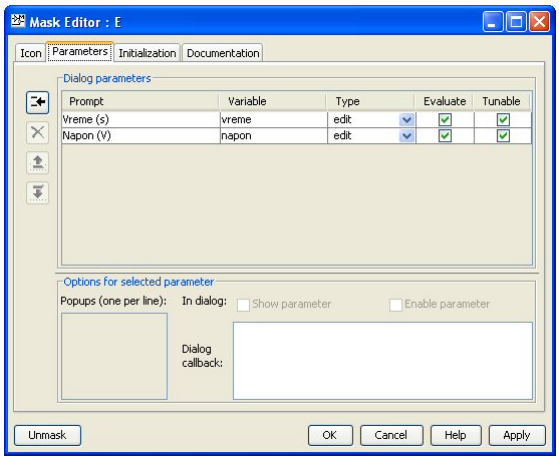
Пример 8.9. Во моделот од примерот 8.8 напонскиот генератор да се замени со друг кој што има напон во форма на рампа. При тоа да се земе дека напонот линеарно расте од 0 до 100 kV за време од 5 μs, а потоа има константна вредност од 100 kV до крајот на симулацијата која што трае 20 μs. Внесувањето на обликот на напонот да се направи преку маска во која што ќе се внесуваат дискретни вредности за напонот и за одредени временски моменти кои што ќе бидат дефинирани со вектори.

Решение

Моделот на колото е даден во датотеката rampa.mdl која што има изглед како на сликата 8.43. Маската на подсистемот ја формираме и форматираме на начинот опишан во примерот 8.8. Во овој случај во делот Initialization нема ништо, додека во делот Documentation имаме внесено соодветен опис.

На сликата 8.53 е прикажан делот со наслов Parameters каде што гледаме дека имаме две редици со кои што сме дефинирале дека сакаме преку маската да внесеме два параметра. Тие параметри се два вектори во кои што ги внесуваме временските моменти и вредностите на напоните во тие моменти со кои што го дефинираме обликот на напонот. Внесените вредности, заедно со изгледот на целата маска, се прикажани на

сликата 8.54 каде што гледаме дека сме внеле по три вредности за времето и напонот со кои што наполно сме го опишале напонот во форма на рампа.

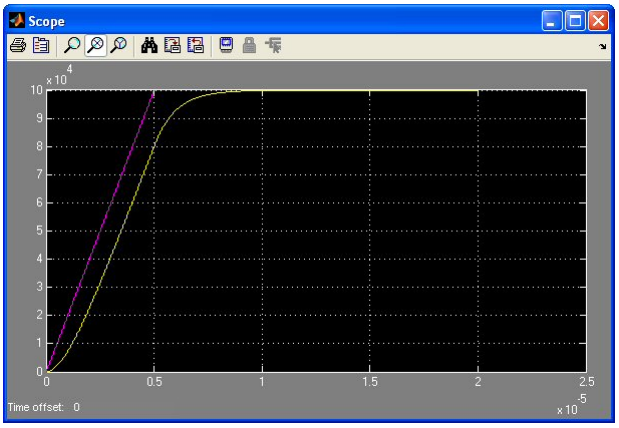


Слика 8.53. Параметри на маската

Слика 8.54. Параметри на напонот со форма на рампа

Во овој случај во полето MATLAB Function од блокот Impuls во подсистемот го имаме внесено изразот interp1(vreme, napon, u) каде што ја користиме функцијата interp1 која што прави линеарна интерполација. Како влезни вектори во оваа функција ги користиме векторите vreme и napon кои ги имаме внесено преку маската, а потоа интерполацијата ја правиме во временскиот момент u кој што го добиваме на излезот од часовникот во самиот подсистем.

Со активирање на симулацијата сигналите на осцилоскопот го добиваат обликот кој што е прикажан на сликата 8.55.



Слика 8.55. Напони на генераторот-рампа и на кондензаторот

□ □ □

Пример 8.10. Да се направи модел на диода ако е познато дека нејзината U-I карактеристика е дадена со Шоклиевиот израз кој што гласи

$$I = I_S \cdot \left(e^{U/U_T} - 1 \right), \tag{8.7}$$

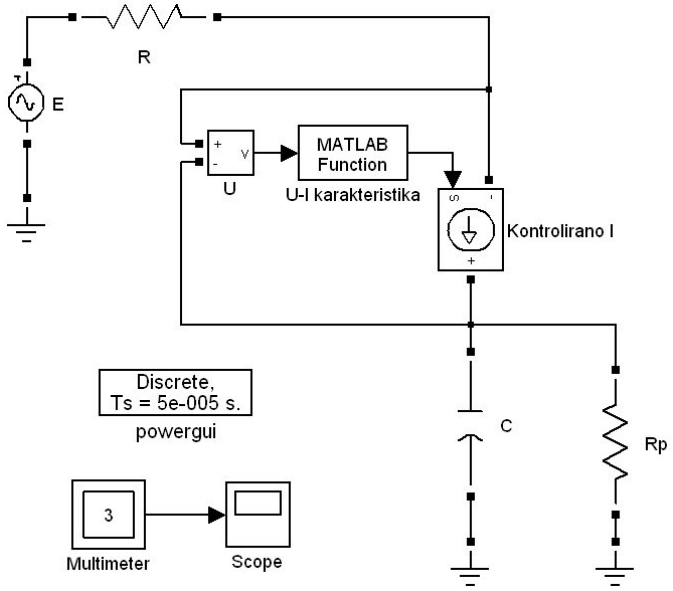
каде што I_S е струја во непропусната насока на диодата, а U_T е термичкиот напон кој што на собна температура изнесува 26 mV. Во моделот да се земе дека струјата во непропусната насока изнесува 30 μA.

Решение

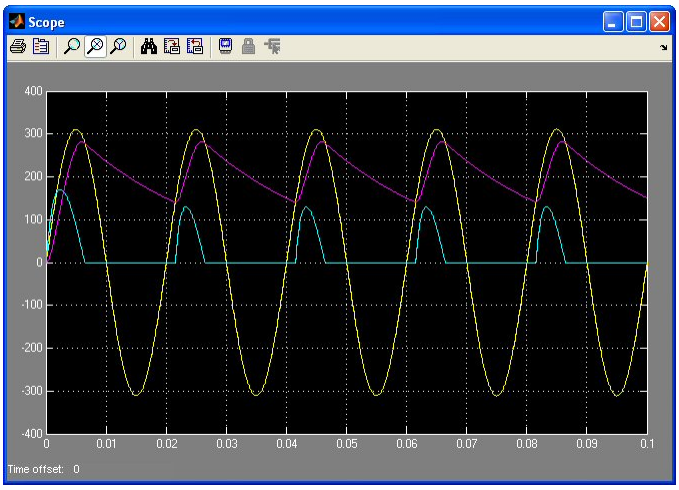
Моделот на колото е даден во датотеката dioda.mdl која што има изглед како на сликата 8.56. Во колото има еден напонски генератор со синусен напон чија што максимална вредност изнесува 220*sqrt(2) и има фреквенција од 50 Hz. Сериски со него е поставен отпорник R = 0,5 Ω кој што ја претставува неговата внатрешна отпорност. Генераторот преку диодата, која што е составена од три елементи, полни кондензатор со капацитет C = 2200 μF од кој што се напојува потрошувач преставен со отпорникот R_p = 10 Ω.

Диодата е моделирана со контролиран струен генератор кој што се наоѓа во библиотеката Electrical Sources од библиотеката SimPowerSystems под името Controlled Current Source. И тој слично како контролираниот напонски генератор од примерот 8.8 има три приклучоци, при што на третиот приклучок се доведува сигналот со кој што се контролира неговата струја. Контролниот сигнал во овој случај се добива така што прво на краевите од струјниот генератор се мери напонот и потоа тој се пропушта преку блокот MATLAB Function (кој тука има име U-I karakteristika). Во овој блок, во полето MATLAB Function (слика 8.41) е внесен изразот 30e-6 * (exp(u/26e-3) - 1), односно со него е пресметана струјата според изразот (8.7) за вредноста на измерениот напон кој што е доведен на неговиот влез. На ваков начин диодата ја третираме како нелинеарен отпорник.

Преку мултиметарот на осцилоскопот се прикажани напонот на напонскиот генератор, напонот на отпорникот R_p и струјата во контролираниот струен генератор. Нивниот изглед е прикажан на сликата 8.57 од каде што се гледа дека нашиот модел на диода функционира правилно.



Слика 8.56. Модел на диода



Слика 8.57. Напони и струја во колото со моделот на диода

□ □ □

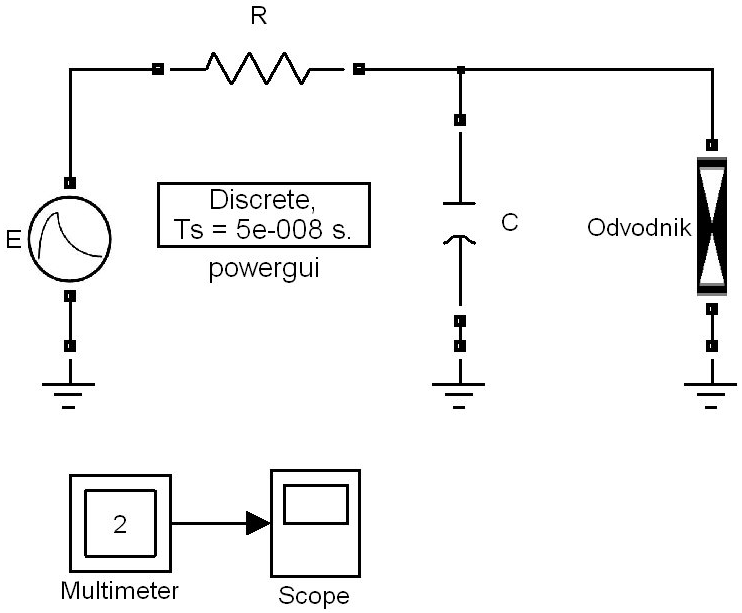
Пример 8.11. За еден ZnO одводник на пренапони со мерење е добиена неговата U-I карактеристика која што е дадена во табелата 8.1. Да се направи модел на одводникот при што неговата карактеристика ќе се внесува преку маска, а потоа тој да се вметне во колото од примерот 8.8. Максималната вредност на напонот на двојно-експоненцијалниот генератор изнесува 30 kV.

Табела 8.1. U-I карактеристика на ZnO одводник

U (V)	0	8000	8500	9000	9500	10000
I (A)	0	0	3	50	300	1000
U (V)	10500	11000	11500	12000	12500	13000
I (A)	2700	5800	12400	25600	51300	100000

Решение

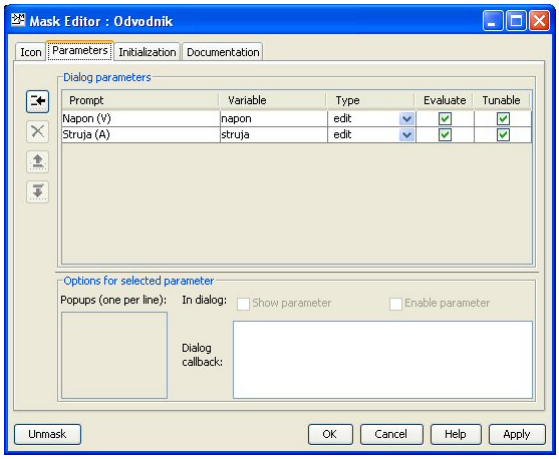
Моделот на колото е даден во датотеката dvojno_exp_odvodnik.mdl која што има изглед како на сликата 8.58 каде што одводникот е сместен во подсистем за кој што е дефинирана маска и е вметната слика во неа.



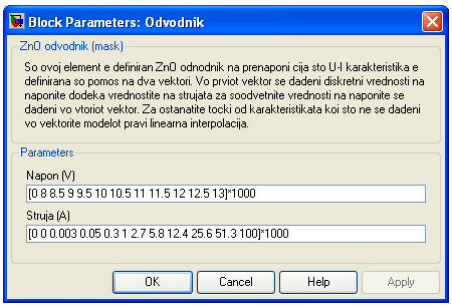
Слика 8.58. RC коло со двојно-експоненцијален напонски генератор и одводник

Маската на подсистемот ја формираме и форматираме на начинот опишан во примерот 8.8. Во овој случај во делот Initialization нема ништо, во делот Documentation имаме внесено соодветен опис, додека во делот Icon е напишана командата image(imread('Odvodnik.jpg')).

На сликата 8.59 е прикажан делот со наслов Parameters каде што гледаме дека имаме две редици со кои што сме дефинирале дека сакаме преку маската да внесеме два параметра. Тие параметри се два вектори во кои што ги внесуваме вредностите на напоните и струјата за кои што е снимена U-I карактеристиката. Внесените вредности, заедно со изгледот на целата маска, се прикажани на сликата 8.60 каде што гледаме дека сме внеле сите вредности од табелата 8.1.



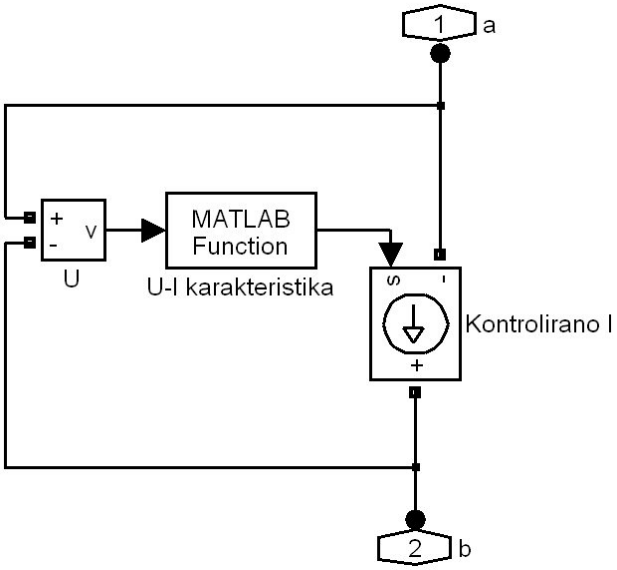
Слика 8.59. Параметри на маската



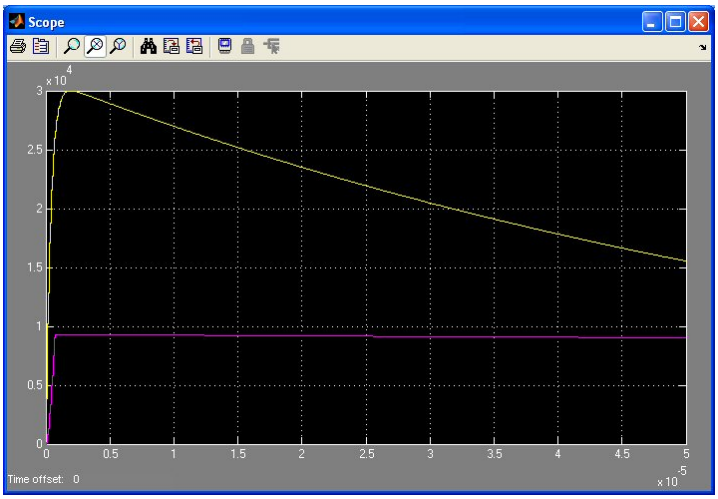
Слика 8.60. Параметри U-I карактеристиката на ZnO одводник

Изгледот на подсистемот со кој е моделиран самиот одводник е даден на сликата 8.61 од каде што се забележува дека тој се моделира на потполно ист начин како и диодата во претходниот пример, односно и него го третираме како нелинеарен отпорник. Во овој случај, како на сликата 8.41, во полето MATLAB Function на блокот U-I karakteristika во подсистемот го имаме внесено изразот interp1(napon, struja, u) каде што повторно ја користиме функцијата interp1 која што прави линеарна интерполација. Како влезни вектори во оваа функција ги користиме векторите napon и struja кои ги имаме внесено преку маската, а потоа интерполацијата ја правиме за вресноста на напонот која што е измерена со блокот за мерење на напон и која што како величина u е дадена на влезот во блокот MATLAB Function. На ваков начин, струјниот генератор ќе добие соодветна вредност на струјата која одговара на напонот измерен на неговите краеви.

Со активирање на симулацијата сигналите на осцилоскопот го добиваат обликот кој што е прикажан на сликата 8.62.



Слика 8.61. Подсистем со модел на нелинеарен отпорник (ZnO одводник)



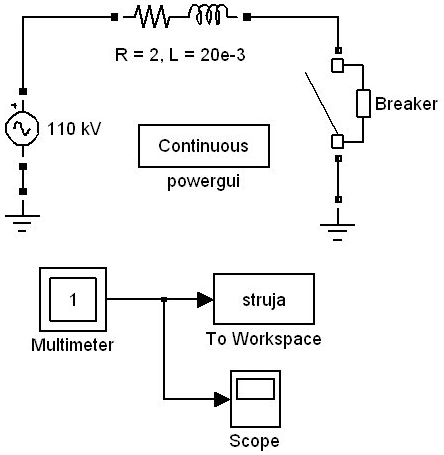
Слика 8.62. Напони на двојно-експоненцијалниот генератор и на кондензаторот во случајот со ZnO одводник на пренапони

□ □ □

Пример 8.12. Разгледуваме едно RL коло кое што се вклучува со прекинувач на напонски генератор со синусен облик. Параметрите на елементите на колото се R = 2 Ω и L = 20 mH, додека максималната вредност на напонот на напонскиот генератор изнесува $U_{\mathrm{m}} = \sqrt{2} \cdot \dfrac{110}{\sqrt{3}}$ kV а неговата фреквенција е f = 50 Hz. Потребно е да се одреди максималната (ударна) вредност на струјата во колото за различни временски моменти на вклучување на прекинувачот почнувајќи од моментот t_1 = 0 до моментот t_2 = 10 ms со чекор од Δt = 0,5 ms.

Решение

Моделот на колото е даден во датотеката Primer_8_12.mdl а неговиот изглед е прикажан на сликата 8.63. Во колото се користени елементи кои што и порано ги имаме користено во другите примери, па поради тоа тука нема детално да ги објаснуваме.

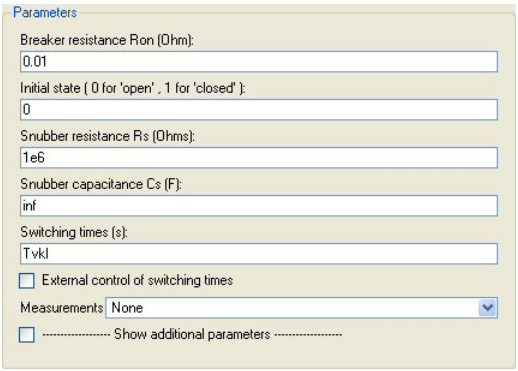


Слика 8.69. Изглед на моделот Primer_8_12.mdl

Со знаењето за Simulink кое што до сега го имаме проблемот во овој пример би можеле да го решиме на следниот начин: на прекинувачот ќе му зададеме време на вклучување 0, ќе ја активираме симулацијата и потоа во командниот простор ќе

напишеме max(abs(struja)) со што ќе ја добиеме максималната струја во колото ако прекинувачот се вклучи во моментот $t_1 = 0$. Вака добиениот резултат ќе го запомниме некаде, а потоа на прекинувачот ќе му зададеме време на вклучување 0,5 ms и ќе ја повториме постапката со што ќе добиеме втора вредност за максималната струја во колото. Оваа постапка можеме да ја повториме 21 пат со што ќе ти опфатиме сите временски моменти од $t_1 = 0$ до моментот $t_2 = 10$ ms со чекор од $\Delta t = 0{,}5$ ms но за тоа ќе потрошиме многу време и има голема веројатност дека нешто ќе погрешиме.

За среќа, во Simulink постои начин погоре опишаната постапка да ја извршиме со помош на програма која самата ќе ги потворува пресметките потребен број пати. Најпрво на прекинувачот ќе му зададеме временски момент за негово вклучување дефиниран со помош на променлива Tvkl како што тоа е прикажано на сликата 8.70.



Слика 8.70. Дефинирање на моментот на вклучување на прекинувачот преку променливата Tvkl

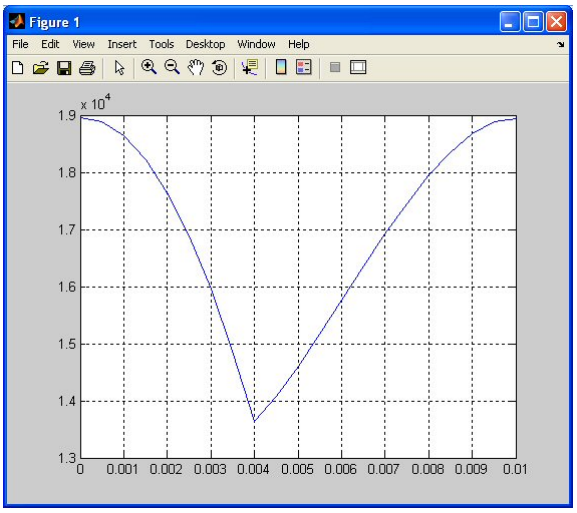
Променливата Tvkl ќе биде контролирана од програмата дадена во датотеката Primer_8_12_povekekratno.m чиј што изглед е следниот

```
T = []; Imax = [];
for Tvkl = 0:0.5e-3:10e-3
    T = [T; Tvkl];
    sim('Primer_8_12');
    Imax = [Imax; max(abs(struja))];
end
plot(T,Imax); grid;
```

На почетокот дефинираме две празни матрици `T` и `Imax` во кои што понатаму ќе ги запомниме временските моменти на вклучување на прекинувачот и соодветните максимални вредности на струјата во колото. Целата постапка се одвива во еден for циклус во кој што променливата `Tvkl` се менува од 0 до $10 \cdot 10^{-3}$ со чекор од $0{,}5 \cdot 10^{-3}$. Во самиот циклус со наредбата `T = [T; Tvkl];` во матрицата `T` додаваме нова редица во која што новиот елемент е еднаков на тековната вредност на променливата `Tvkl`. Потоа ја активираме симулацијата на моделот Primer_8_12.mdl со помош на наредбата `sim('Primer_8_12');` и на крајот со наредбата `Imax = [Imax; max(abs(struja))];` во матрицата `Imax` додаваме нова редица во која што новиот елемент е еднаков на `max(abs(struja))` односно е еднаков на максималната вредност на струјата добиена со симулацијата (во моделот на колото струјата се праќа во работниот простор во променливата `struja`).

На крајот од програмата со помош на наредбата `plot(T,Imax); grid;` ја цртаме зависноста на максималната струја во колото од временскиот момент на вклучување на прекинувачот со што ја добиваме сликата 8.71. Од сликата забележуваме дека најголема ударна струја се добива ако прекинувачот се вклучи во

временскиот момент 0, а најмала се добива ако тој се вклучи во моментот 4 ms. Вредностите на струите можеме да ги видиме во работниот простор во променливата `Imax` и тие изнесуваат 18951 A и 13640 A соодветно.



Слика 8.71. Зависност на максималната вредност на струјата од временскиот момент на вклучување на прекинувачот

□ □ □

9. ОПТИМИЗАЦИЈА ВО MATLAB

9.1. ЛИНЕАРНИ ФУНКЦИИ

Проблемите во кои што се бара минимизација на една линеарна функција од повеќе променливи, при што решението треба да задоволи низа линеарни ограничувања прикажани со помош на равенки или неравенки, се решаваат со помош на линеарното програмирање кое што во Matlab е дадено во функцијата **linprog**. Таа е наменета за решавање на следниот математички проблем:

$$\min_{x} \boldsymbol{C}^{\mathrm{T}} \cdot \boldsymbol{x}, \tag{9.1}$$

при што треба да се задоволени следните ограничувања:

$$\boldsymbol{A} \cdot \boldsymbol{x} \le \boldsymbol{b}, \tag{9.2}$$

$$\boldsymbol{Aeq} \cdot \boldsymbol{x} = \boldsymbol{beq}, \tag{9.3}$$

$$\boldsymbol{lb} < \boldsymbol{x} < \boldsymbol{ub}. \tag{9.4}$$

Во општ случај функцијата linprog се повикува на следниот начин:

```
[x,fval,exitflag] = linprog(C,A,b,Aeq,beq,lb,ub,x0,options)
```

каде што

C Вектор во кој што се дадени коефициентите на линеарната функција дефинирана со (9.1),

A Матрица која се користи на левата страна на ограничувањето (9.2),

b Вектор кој се користи на десната страна на во ограничувањето (9.2),

Aeq Матрица која се користи на левата страна на во ограничувањето (9.3),

beq Вектор кој се користи на десната страна на во ограничувањето (9.3),

lb Вектор со вредности за долните граници на променливите во ограничувањето (9.4),

ub Вектор со вредности за горните граници на променливите во ограничувањето (9.4),

x0 Вектор со почетно решение за променливите,

options Вектор во кој се дефинирани опциите со кои сакаме да работи функцијата **linprog** при одредување на минимумот на функцијата (9.1)**.** Тие се дефинират со користење на командата **optimset**,

Како резултат од функцијата linprog се добиваат следните променливи

x Оптимални вредности на променливите,

fval Вредност на функцијата пресметана со оптималните вредности на променливите,

exitflag Променлива со која што се сигнализира каков е исходот од решавањето на проблемот. Ако оваа променлива има вредност 1 алгоритамот успеал да најде оптимално решение кое што ги задоволува сите ограничувања. Во спротивно при решавањето имало некој проблем кој не довел до конечно решение на проблемот. За значењето на останатите вредности на променливата може повеќе информации да се добијат со пишување на наредбата help linprog.

За да го прикажеме начинот на примена на функција на linptog ќе разгледаме неколку проблеми.

Пример 9.1. Да се одреди минимумот на функцијата

$$y = -5x_1 - 4x_2 - 6x_3,$$

со следните ограничувања

$$x_1 - x_2 + x_3 \le 20,$$

$$3x_1 + 2x_2 + 4x_3 \le 42,$$

$$3x_1 + 2x_2 \le 30,$$

$$x_1 \ge 0,\ x_2 \ge 0,\ x_3 \ge 0.$$

Решение

Програмата со која што ќе го решиме наведениот проблем е дадена во датотеката primer_9_1.m чија што содржина е

```
C = [-5; -4; -6];
A = [1 -1 1; 3 2 4; 3 2 0];
b = [20; 42; 30];
lb = [0; 0; 0];
[x,fval,exitflag] = linprog(C,A,b,[],[],lb)
```

Во неа прво сме ги дефинирале векторот ***C*** , како и матрицата ***A*** и векторот ***b*** со кои што се дефинирани ограничувањата од типот (9.2). Трите долните граници на променливите се сместени во векторот ***lb***, додека векторот ***ub*** не е дефиниран затоа што

во овој проблем немаме зададени горни граници за промениливите. Исто така, тука не ги користиме матрицата ***Aeq*** и векторот ***beq*** затоа што нема ограничувања од типот на равенство (9.3).

При повикувањето на функцијата linprog на местото на променливите ***Aeq*** и ***beq*** имаме ставено празни матрици со што сме и дале до знаење дека немаме ограничувања од типот (9.3). Освен тоа, променливите десно од променливата ***lb*** воопшто не ги внесуваме затоа што тие се небитни за разгледуваниот проблем.

Со активирање на програмата го добиваме следниот резултат

```
Optimization terminated.
x =
    0.0000
   15.0000
    3.0000
fval =
  -78.0000
exitflag =
     1
```

Бидејќи променливата exitflag има вредност 1 решението е оптимално и ги задоволува сите ограничувања.

□ □ □

Пример 9.2. Да се одреди минимумот на функцијата

$y = -5x_1 - 4x_2 - 6x_3$,

со следните ограничувања

$x_1 - x_2 + x_3 \le 20$,

$3x_1 + 2x_2 + 4x_3 \le 42$,

$3x_1 + 2x_2 \ge 30$,

$x_1 + x_2 = 20$,

$x_1 \ge 0,\ x_2 \ge 0,\ x_3 \ge 0, x_2 \le 18$.

Решение

Пред да започнеме да го решаваме проблемот треба сите ограничувања да ги напишеме во облик (9.2) или (9.3). Од сите нив, ограничувањето $3x_1 + 2x_2 \ge 30$ не е напишано во бараниот облик, но тоа лесно можеме да го надминеме со едноставно множење со -1 со што добиваме $-3x_1 - 2x_2 \le -30$. Програмата со која што ќе го решиме наведениот проблем е дадена во датотеката primer_9_2.m чија што содржина е

```
C = [-5; -4; -6];
A = [1 -1 1; 3 2 4; -3 -2 0];
b = [20; 42; -30];
Aeq = [1 1 0];
beq = [20];
lb = [0; 0; 0];
ub = [inf; 18; inf];
[x,fval,exitflag] = linprog(C,A,b,Aeq,beq,lb,ub)
```

Таа е доста слична со програмата од примерот 9.1 со тоа што тука сега ги користиме и матрицата ***Aeq*** заедно со векторот ***beq*** со кои што го дефинираме ограничувањето $x_1 + x_2 = 20$. Освен тоа, тука е воведен и векторот ***ub*** во кој што го дефинираме ограничувањето од горна страна $x_2 \leq 18$. Бидејќи кај другите две променливи нема ограничување од горната страна во тој вектор имаме внесено inf што во Matlab означува бесконечност.

Со активирање на програмата го добиваме следниот резултат

```
Optimization terminated.
x =
    2.0000
   18.0000
    0.0000
fval =
  -82.0000
exitflag =
     1
```

Бидејќи променливата exitflag има вредност 1 решението е оптимално и ги задоволува сите ограничувања.

□ □ □

Пример 9.3. Некој оператор на пазарот на електрична енергија треба да набави вкупно 300 MW од тројца продавачи на електрична енергија. Првиот продавач ја нуди својата електрична енергија по цена од 70 €/MWh и нуди моќност во опсегот од 70 до 200 MW. Понудената цена на вториот продавач изнесува 75 €/MWh и тој нуди моќност во опсегот од 50 до 150 MW. Третиот продавач нуди електрична енергија по цена од 65 €/MWh во опсегот од 30 до 100 MW. Да се одреди колкава моќност треба да се земе од секој продавач за да се постигне минимална вредност на трошоците.

Решение

Во овој пример, за разлика од претходните два, проблемот не е дефиниран експлицитно со математички изрази туку само описно. Затоа на почетокот прво треба да го напишеме проблемот во облик даден со изразите (9.1) – (9.4).

Функцијата која што треба да се минимизира овде ќе биде еднаква на сумата од производите на цената на електричната енергија и моќноста која што треба да се купи од одделните продавачи, т.е.

$$F = 70 \cdot P_1 + 75 \cdot P_2 + 65 \cdot P_3,$$

при што треба да бидат задоволени следните ограничувања

$$P_1 + P_2 + P_3 = 300,$$

$$70 \leq P_1 \leq 200,$$

$$50 \leq P_2 \leq 150,$$

$$30 \leq P_3 \leq 100.$$

Програмата со која што ќе го решиме вака дефинираниот проблем е дадена во датотеката primer_9_3.m а нејзиниот изглед е

```
C = [70; 75; 65];
Aeq = [1 1 1];
beq = [300];
lb = [70; 50; 30];
ub = [200; 150; 100];
[P,F,exitflag] = linprog(C,[],[],Aeq,beq,lb,ub)
```

Со активирање на програмата го добиваме следниот резултат

```
Optimization terminated.
P =
  150.0000
   50.0000
  100.0000
F =
  2.0750e+004
exitflag =
     1
```

Бидејќи променливата exitflag има вредност 1 решението е оптимално и ги задоволува сите ограничувања, што значи дека од првиот продавач треба да купиме 150 MW, од вториот 50 MW и од третиот 100 MW. Вкупните трошоци во тој случај ќе изнесуваат 20750 €/h.

□ □ □

Пример 9.4. Го разгледуваме проблемот од примерот 9.3 во кој што поради определени технички проблеми дополнително се бара вкупната купена моќност од првиот и третиот продавач да не надмине 220 MW. Да се одреди колкава моќност треба да се земе од секој продавач во оваа ситуација за да се постигне минимална вредност на трошоците.

Решение

Во овој случај треба да се најде минимун на функцијата

$$F = 70 \cdot P_1 + 75 \cdot P_2 + 65 \cdot P_3,$$

при што треба да бидат задоволени следните ограничувања

$$P_1 + P_2 + P_3 = 300,$$

$$P_1 + P_3 \le 220,$$

$$70 \le P_1 \le 200,$$

$$50 \le P_2 \le 150,$$

$$30 \le P_3 \le 100.$$

Програмата со која што ќе го решиме вака дефинираниот проблем е дадена во датотеката primer_9_4.m а нејзиниот изглед е

```
C = [70; 75; 65];
A = [1 0 1];
b = [220];
Aeq = [1 1 1];
beq = [300];
lb = [70; 50; 30];
ub = [200; 150; 100];
[P,F,exitflag] = linprog(C,A,b,Aeq,beq,lb,ub)
```

Со активирање на програмата го добиваме следниот резултат

```
Optimization terminated.
P =
  120.0000
   80.0000
  100.0000
F =
  2.0900e+004
exitflag =
     1
```

Бидејќи променливата exitflag има вредност 1 решението е оптимално и ги задоволува сите ограничувања, што значи дека од првиот продавач треба да купиме 120 MW, од вториот 80 MW и од третиот 100 MW. Вкупните трошоци во тој случај ќе изнесуваат 20900 €/h и се нешто поголеми од трошоците пресметани во примерот 9.3.

□ □ □

9.2. НЕЛИНЕАРНИ ФУНКЦИИ

За минимизација на произволна функција со ограничувања во Matlab се користи функцијата fmincon. Таа е наменета за решавање на следниот математички проблем:

$$\min_{x} f(\boldsymbol{x}), \tag{9.5}$$

при што треба да се задоволени следните ограничувања

$$c(\boldsymbol{x}) \le 0, \tag{9.6}$$

$$ceq(\boldsymbol{x}) = 0, \tag{9.7}$$

$$\boldsymbol{A} \cdot \boldsymbol{x} \le \boldsymbol{b}, \tag{9.8}$$

$$\boldsymbol{Aeq} \cdot \boldsymbol{x} = \boldsymbol{beq}, \tag{9.9}$$

$$\boldsymbol{lb} < \boldsymbol{x} < \boldsymbol{ub}. \tag{9.10}$$

Во општ случај функцијата fmincon се повикува на следниот начин:

```
[x,fval,exitflag] = …
fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options,p1,p2,…)
```

каде што:

fun Функција која треба да се минимизира и која како параметар прима вектор $\boldsymbol{x}$, а враќа скалар f кој е еднаков на вредноста на функцијата на пресметана за вредноста на променливите зададени во векторот $\boldsymbol{x}$. Функцијата **fun** се задава во посебна m-датотека,

x0 Вектор со почетно решение за променливите,

A Матрица која се користи на левата страна на ограничувањето (9.8),

b Вектор кој се користи на десната страна на во ограничувањето (9.8),

Aeq Матрица која се користи на левата страна на во ограничувањето (9.9),

beq Вектор кој се користи на десната страна на во ограничувањето (9.9),

lb Вектор со вредности за долните граници на променливите во ограничувањето (9.10),

ub Вектор со вредности за горните граници на променливите во ограничувањето (9.10),

nonlcon Функција која ги пресметува нелинераните ограничувања во вид на неравенство (9.6) и равенство (9.7). Функција како параметар прима вектор ***x***, а враќа скалар два вектори ***c*** и ***ceq***. Векторот ***c*** ги содржи вредностите на нелинеарните неравенки (9.6) пресметани за вредноста на променливите зададени во векторот ***x***, а ***ceq*** ги содржи вредностите на нелинарните равенки (9.7) пресметани за вредноста на променливите зададени во векторот ***x***. Функцијата **nonlcon** се задава во посебна m-датотека,

options Вектор во кој се дефинирани опциите со кои сакаме да работи функцијата **fmincon** при одредување на минимумот на функцијата **fun.** Тие се дефинират со користење на командата **optimset**,

p1,p2,… Вектори или матрици со кои се дефинираат некои параметри кои се специфични за пресметување на функцијата **fun**.

Како резултат од функцијата fmincon се добиваат променливите **x**, **fval** и **exitflag** кои што имаат исто значење како и кај функцијата linprog.

Пример 9.5. Да се одреди минимумот на функцијата

$y = 5 - x_1^2 - 2x_2^2 - x_1$,

со следните ограничувања

$-x_1^2 + x_2^2 \le 4$,

$2x_1 + x_2 \le 6$.

Решение

Пред да можеме да ја употребиме функцијата fmincon е потребно сите ограничувања да ги напишеме во облик во кој на десната страна од ограничувањето ќе имаме 0. Така добиваме:

$-x_1^2 + x_2^2 - 4 \le 0$,

$2x_1 + x_2 - 6 \le 0$.

За да ја дефинираме функцијата на цел во Matlab пишуваме функција која ја снимаме во датотеката funkcija_9_5.m која што го има следниот изглед

```
function y = funkcija_9_5(x)
y = 5 - x(1)^2 - 2*x(2)^2 - x(1);
```

Нелинеарното ограничување го дефинираме со друга функција која ја ја снимаме во датотеката nelin_ogr_9_5.m со следниот изглед:

```
function [c, ceq] = nelin_ogr_9_5(x)
c = [-x(1)^2 + x(2)^2 - 4];
ceq = [];
```

каде што матицата ***ceq*** е празна затоа што во дадениот проблем нема нелинеарно ограничување од вид на равенство.

Линеарното ограничување $2x_1 + x_2 \le 6$ матрично можеме да го напишеме во облик

$$\begin{bmatrix} 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \le 6,$$

кој што одговара на релацијата (9.9) за која важи $\boldsymbol{A} = \begin{bmatrix} 2 & 1 \end{bmatrix}$ и $\boldsymbol{b} = \begin{bmatrix} 6 \end{bmatrix}$.

Во датотеката primer_9_5.m ја запишуваме програмата со која ја повикуваме функцијата fmincon:

```
x0 = [0 0];
A = [2 1]; b = 6;
%opcija za prikaz na iteraciite vo tekot na optimizacijata
options = optimset('Display','iter');
[x,fval,exitflag] = ...
    fmincon('funkcija_9_5',x0,A,b,[],[],[],[],'nelin_ogr_9_5',options)
```

Векторот `x0` го содржи почетното решение за променливите x_1 и x_2 кои во овој случај е земено дека се еднакви на 0. Со активирањето на програмата го добиваме следното решение

```
x =
    6.3094   -6.6188
fval =
 -128.7350
exitflag =
     1
```

кое што е оптимално и ги исполнува сите ограничувања затоа што променливата exitflag има врдност еднаква на 1.

□ □ □

Пример 9.6. Да се одреди минимумот на функцијата

$$y = 5 - x_1^2 - 2x_2^2 - x_1,$$

со следните ограничувања

$$3 \le -x_1^2 + x_2^2 \le 4,$$

$$x_1 + x_2 = 5.$$

Решение

Бидејќи се работи за истата функција како во претходниот пример и тука ќе ја користиме функцијата funkcija_9_5 која што е дадена во датотеката funkcija_9_5.m.

Првото ограничување ќе го поделиме на два дела на следниот начин

$$3 \le -x_1^2 + x_2^2,$$

$$-x_1^2 + x_2^2 \le 4,$$

а потоа ќе ги напишеме во облик во кој од десната страна ќе имаме нула:

$$3 + x_1^2 - x_2^2 \le 0,$$

$$-x_1^2 + x_2^2 - 4 \le 0.$$

Нелинеарните ограничувања ги дефинираме со функција која ја снимаме во датотеката nelin_ogr_9_6.m со следниот изглед:

```
function [c, ceq] = nelin_ogr_9_6(x)
c = [
    3 + x(1)^2 - x(2)^2
    -x(1)^2 + x(2)^2 - 4
    ];
ceq = [];
```

Линеарното ограничување $x_1 + x_2 = 5$ матрично можеме да го напишеме во облик

$$\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 5,$$

кој одговара на релацијата (9.9) за која важи $\boldsymbol{Aeq} = \begin{bmatrix} 1 & 1 \end{bmatrix}$ и $\boldsymbol{beq} = \begin{bmatrix} 5 \end{bmatrix}$.

Во датотеката primer_9_6.m ја запишуваме програмата со која ја повикуваме функцијата FMINCON:

```
x0 = [0 0];
Aeq = [1 1]; beq = 5;
%opcija za prikaz na iteraciite vo tekot na optimizacijata
options = optimset('Display','iter');
[x,fval,exitflag] = ...
    fmincon('funkcija_9_5',x0,[],[],Aeq,beq,[],[],'nelin_ogr_9_6',options)
```

Со активирањето на програмата го добиваме следното решение

```
x =
    2.1000    2.9000
fval =
  -18.3300
exitflag =
     1
```

кое што е оптимално и ги исполнува сите ограничувања затоа што променливата exitflag има вредност еднаква на 1.

□ □ □

Пример 9.7. Во табелата 9.1 се дадени податоците за $n = 3$ генератори при што за секој генератор i се познати неговата минимална моќност $PG_i^{\min}$, максималната

моќност $PG_i^{\max}$, како и неговата карактеристика на производни трошоци која што е дадена во облик на квадратна функција $f_i = a_i + b_i PG_i + c_i PG_i^2$. Моќноста на сите потрошувачи во системот изнесува P = 270 MW. Да се одредат моќностите на генераторите така што вкупните трошоци во системот да бидат минимални и да биде задоволен билансот на моќности во системот како и техничките ограничувања на генераторите.

Табела 9.1. Карактеристики на генераторите

Бр.	$PG_{\min}$(MW)	$PG_{\max}$(MW)	a (€/h)	b (€/MWh)	c (€/MW^{2}h)
1	40	100	10	2,0	0,010
2	40	80	10	1,5	0,012
3	50	120	20	1,5	0,004

Решение

Решението на проблемот ќе го добиме со барање на минимум на фукцијата дефинирана со (9.11) која што се состои од вкупните трошоци на сите n генератори

$$F = \sum_{i=1}^{n} f_i = \sum_{i=1}^{n} \left(a_i + b_i PG_i + c_i PG_i^2 \right), \tag{9.11}$$

при што треба да биде задоволено ограничувањето за билансот на моќности

$$\sum_{i=1}^{n} PG_i = PP_{\Sigma}\,, \tag{9.12}$$

како и техничките ограничувања за моќностите на генераторите

$$PG_i^{\min} \le PG_i \le PG_i^{\max}\,. \tag{9.13}$$

Програмата со која што ќе го решиме проблемот дефиниран со изразите (9.11) – (9.13) користејќи ги податоците од табелата 9.1 е дадена во датотеката primer_9_7.m. Во неа најнапред ја дефинираме вкупната моќност на потрошувачите

```
P = 270;
```

потоа ги дефинираме податоците за генераторите со следната матрица

```
GEN = [
    1 40 100 10 2.0 0.010
    2 40  80 10 1.5 0.012
    3 50 120 20 1.5 0.004
];
```

каде што во секоја редица, за еден генератор, ги задаваме редниот број на генераторот, минималната моќност на генераторот, максималната моќност на генераторот и коефициентите a, b и c од карактеристиката на производни трошоци на генераторот.

Потоа ги формираме векторите `PGmin` и `PGmax` кои всушност се долната и горната граница на независните променливи (тие одговараат на променливите `lb` и `ub` дефинирани погоре кога го опишувавме начинот на кој се повикува функцијата `fmincon`), како и матрицата `ABC` која што ги содржи коефициентите од карактеристиката на производни трошоци за сите генератори. Тоа го правиме со командите

```
PGmin = GEN(:,2); PGmax = GEN(:,3); ABC = GEN(:,4:6);
```

при што наведените променливи ги добиваат вредностите

```
PGmin =          PGmax =          ABC =
    40               100              10.0000    2.0000    0.0100
    40                80              10.0000    1.5000    0.0120
    50               120              20.0000    1.5000    0.0040
```

На крајот бројот на генератори го одредуваме со командата `NG = size(GEN,1);`

Функцијата на цел (9.11) која што треба да се минимизира ја дефинираме во датотеката trosoci.m која го има следниот изглед

```
function f = trosoci(PG,ABC)
f = 0;
for i = 1:size(PG)
    f = f + ABC(i,1) + ABC(i,2)*PG(i) + ABC(i,3)*PG(i)^2;
end
```

Функцијата `trosoci` како параметар прима вектор `PG`, а враќа скалар `f` кој што е еднаков на вредноста на функцијата (9.11) т.е. на сумата на трошоци за сите генератори пресметана со вредностите на променливите зададени во векторот `PG` кој ги содржи вредностите на моќностите на генераторите. Првиот параметар на функцијата `trosoci` секогаш мора да биде векторот кој ги содржи независните променливи т.е. векторот `PG`, а потоа може да следуваат останатите параметри како што е тоа тука матрицата `ABC` која ги содржи коефициентите од кривата на трошоци за генераторите.

Изразот `ABC(i,1) + ABC(i,2)*PG(i) + ABC(i,3)*PG(i)^2` е всушност изразот $a_i + b_i P_i + c_i P_i^2$ напишан на начин кој се користи во Matlab, додека пак со користењето на тој израз во циклусот `for i=1:size(PG)` се пресметува изразот $\sum_{i=1}^{n}\left(a_i + b_i P_i + c_i P_i^2\right)$.

На крајот останува уште да ја повикаме функцијата fmincon која што ќе ни го даде оптималното решение. Тоа го правиме на следниот начин

```
PG0 = (PGmin + PGmax) / 2;
[PG,F,exitflag] = ...
    FMINCON('trosoci',PG0,[],[],ones(1,NG),P,PGmin,PGmax,[],[],ABC)
```

со што сме ги дефинирале следните параметри на функцијата:

`'trosoci'`	име на функцијата која што сакаме да ја минимизираме,
`PG0`	вектор со почетно решение за променливите, за кој овде сме ја задале средната вредност од интервалот на дозволени вредности на моќности за секој генератор,
`[],[]`	два празни параметри со што на функцијата `fmincon` и даваме до знаење дека ограничувањата од типот (9.8) во нашиот случај не постојат,
`ones(1,NG)`	редица со `NG` единици која всушност ја претставува матрицата `Aeq` која се користи на левата страна на ограничувањето (9.9),
`P`	број (моќност на потрошувачите) кој всушност го претставува вектор `beq` кој се користи на десната страна на ограничувањето (9.9),

користејќи редица со единици ние всушност матрично сме ја напишале следната равенка: $PG_1 + PG_2 + ... + PG_{NG} = P$, т.е. на функцијата `fmincon` сме и „кажале“ дека нашето ограничување (9.9) може матрично да се напише на следниот начин

$$\begin{bmatrix} 1 & 1 & \dots & 1 \end{bmatrix} \cdot \begin{bmatrix} PG_1 \\ PG_2 \\ \dots \\ PG_{NG} \end{bmatrix} = P .$$

`PGmin` вектор со минимални вредности на моќностите на генераторите (9.10),

`PGmax` вектор со максимални вредности на моќностите на генераторите (9.10),

`[],[]` два празни параметри со што на функцијата `fmincon` и даваме до знаење дека немаме ограничувањата од типот (9.6) и (9.7), односно дека не и задаваме функција `nonlcon` и дека не и задаваме никакви опции, т.е. оставаме самата да одлучи на кој начин ќе го одреди оптималното решение (default опции),

`ABC` матрица која што ги содржи коефициентите од кривата на трошоци на генераторите.

По извршувањето на програмата од датотеката primer_9_7.m, чиј што комплетен изглед е следниот

```
P = 270;
GEN = [
    1 40 100 10 2.0 0.010
    2 40  80 10 1.5 0.012
    3 50 120 20 1.5 0.004
];
PGmin = GEN(:,2); PGmax = GEN(:,3); ABC = GEN(:,4:6); NG = size(GEN,1);
PG0 = (PGmin + PGmax) / 2;
[PG,F,exitflag] = ...
    FMINCON('trosoci',PG0,[],[],ones(1,NG),P,PGmin,PGmax,[],[],ABC)
```

како резултат добиваме

```
PG =
   70.4545
   79.5455
  120.0000
F =
  663.3955
exitflag =
     1
```

што значи дека моќностите на генераторите треба да бидат P_1 = 70,4545 MW; P_2 = 79,5455 MW и P_3 = 120 MW при што трошоците за работа на системот изнесуваат 663,4 €/h.

□ □ □

10. СИСТЕМИ НЕЛИНЕАРНИ РАВЕНКИ

За решавање на системи нелинеарни равенки во Matlab се користи функцијата fsolve која што е наменета за решавање на следниот математички проблем

$$F(\boldsymbol{x}) = 0, \tag{10.1}$$

каде што $\boldsymbol{x}$ е вектор кој што ги содржи вредностите на променливите, а F е произволна нелинеарна функција која што враќа вектор со иста димензија како и векторот $\boldsymbol{x}$, односно таа пресметува вредности на повеќе нелинеарни функции за вредностите дадени во векторот $\boldsymbol{x}$.

Во општ случај функцијата fsolve се повикува на следниот начин

```
[x,fval,exitflag] = fsolve(fun,x0,options,p1,p2,…)
```

каде што:

fun Функција на која што треба да и се одредат нулите и која како параметар прима вектор $\boldsymbol{x}$, а враќа вектор $\boldsymbol{f}$ кој што е еднаков на вредноста на функцијата пресметана за вредноста на променливите зададени во векторот $\boldsymbol{x}$. Функцијата **fun** се задава во посебна m-датотека,

x0 Вектор со почетно решение за променливите,

options Вектор во кој се дефинирани опциите со кои сакаме да работи функцијата **fsolve** при одредување на нулите на функцијата **fun.** Тие се дефинират со користење на командата **optimset**,

p1,p2,… Вектори или матрици со кои се дефинираат некои параметри кои се специфични за пресметување на функцијата **fun**.

Како резултат од функцијата fmincon се добиваат променливите **x**, **fval** и **exitflag** кои што имаат исто значење како и кај функцијата linprog.

Пример 10.1. Да се одредат нулите на следниот систем нелинеарни равенки

$$x_1^2 - x_2 = -1,$$

$$x_2^2 - x_1 = 3,5,$$

при што е познато дека почетното решение е $x_1 = 1$, $x_2 = 2$.

Решение

Најпрво е потребно системот линеарни равенки да го напишеме во обликот (10.1) каде што од десната страна стои само една нула, со што добиваме

$$x_1^2 - x_2 + 1 = 0,$$

$$x_2^2 - x_1 - 3,5 = 0.$$

За горниот систем равенки ја дефинираме функцијата funkcija_10_1 која што е дадена во датотеката funkcija_10_1.m и чија што содржина е

```
function f = funkcija_10_1(x)
f = [
    x(1)^2 - x(2) + 1
    x(2)^2 - x(1) - 3.5
    ];
```

Програмата за решавање на системот линеарни равенки е дадена во датотеката primer_10_1.m која што содржи само две линии во кои е дефинирано почетното решение и е повикана функцијата fsolve:

```
x0 = [1; 2];
x = fsolve(@funkcija_10_1,x0)
```

Со активирањето на програмата го добиваме следниот резултат

```
Optimization terminated: first-order optimality is less than
options.TolFun.
x =
    1.0662
    2.1369
fval =
  1.0e-009 *
    0.3407
    0.1596
exitflag =
     1
```

од каде што забележуваме дека решенијата се е x_1 = 1,0662; x_2 = 2,1369; двете нелинеарни равенки имаат вредности практично еднакви на нула и се исполнети условите за конвергенција (exitflag = 1).

□ □ □

Пример 10.2. Да се одредат нулите на следниот систем нелинеарни равенки

$$\begin{aligned} x_1 + 7x_3 &= \frac{9}{x_1} \\ 2x_1 - x_2 + 3x_3 &= \frac{5}{x_2}, \\ x_2 - 2x_3 &= \frac{-1}{x_3} \end{aligned}$$

при што е познато дека во почетното решение сите променливи се еднакви на 1.

Решение

Дадениот систем равенки е можно да се напише во Matlab со користење на матрици на следниот начин

```
A * x = b ./ x
```

или

```
A * x - b ./ x = 0
```

каде што за матриците $\boldsymbol{A}$ и $\boldsymbol{b}$ важи

$$\boldsymbol{A}=\begin{bmatrix}1 & 0 & 7\\ 2 & -1 & 3\\ 0 & 1 & -2\end{bmatrix},$$

$$\boldsymbol{b}=\begin{bmatrix}9\\ 5\\ -1\end{bmatrix}.$$

За горниот систем равенки ја дефинираме функцијата funkcija_10_2 која што е дадена во датотеката funkcija_10_2.m и чија што содржина е

```
function f = funkcija_10_2(x,A,b)
f = A * x - b ./ x;
```

каде што во листата со аргументи на функцијата, покрај аргументот `x` кој што е прв, се додадени и дополнителни аргументи `A` и `b`.

Програмата за решавање на системот нелинеарни равенки е дадена во датотеката primer_10_2.m чиј што изглед е следниот

```
x0 = [1; 1; 1];
A = [1  0  7; 2 -1  3; 0  1 -2];
b = [9; 5; -1];
[x, fval, exitflag] = fsolve(@funkcija_10_2,x0,[],A,b)
```

Во програмата прво се дефинирани почетното решение и матриците $\boldsymbol{A}$ и $\boldsymbol{b}$, а потоа при повикувањето на функцијата fsolve по параметарот со почетното решение е зададена една празна матрица (со која што сме дефинирале дека сакаме да работиме со default опции), а потоа се додадени параметрите `A` и `b` кои што се специфични за разгледуваниот проблем.

Со активирањето на програмата го добиваме следниот резултат

```
Optimization terminated: first-order optimality is less than
options.TolFun.
x =
    1.0501
    1.2179
    1.0744
fval =
  1.0e-010 *
   -0.0218
   -0.1323
    0.0061
exitflag =
     1
```

од каде што забележуваме дека решенијата со даденото решение се трите нелинеарни равенки имаат вредности практично еднакви на нула и се исполнети условите за конвергенција (exitflag = 1).

□ □ □

Пример 10.3. Да се одредат нулите на следниот систем комплексни нелинеарни равенки

$$(1+j)\underline{Z}_1 + 3\underline{Z}_2 = \frac{4}{\underline{Z}_1},$$

$$(2-j)\underline{Z}_1 + (-1+j3)\underline{Z}_2 = \frac{1+j2}{\underline{Z}_2},$$

при што е познато дека решението е такво што двете променливи имаат реални делови приближно еднакви на 1 и имагинарни делови приближно еднакви на 0.

Решение

Дадениот систем равенки, како и во претходниот пример, е можно да се напише во Matlab со користење на матрици на следниот начин

```
A * Z - b ./ Z = 0
```

каде што за матриците ***A*** и ***b*** важи

$$\boldsymbol{A} = \left[\begin{array}{c|c} 1+j & 3 \\ \hline 2-j & -1+j3 \end{array}\right],$$

$$\boldsymbol{b} = \begin{bmatrix} 4 \\ 1+j2 \end{bmatrix}.$$

Бидејќи функцијата fsolve решава системи нелинеарни равенки со реални броеви треба нашиот комплексен систем равенки да го поделиме на реален и имагинарен дел. Прво дефинираме вектор ***D*** кој што е еднаков на

```
D = A * Z - b ./ Z
```

при што целта е да се одреди векторот ***Z*** на таков начин што сите елементи од векторот ***D*** ќе бидат еднакви на нула. Еден комплексен број ќе биде еднаков на нула ако истовремено неговиот реален и имагинарен дел е еднаков на нула. Според тоа, елементите на комплексниот вектор ***D*** ќе бидат еднакви на нула ако елементите на следниот реален вектор ***F*** се еднакви на нула

```
F = [
    real(D)
    imag(D)
    ];
```

Векторот ***F*** има два пати поголема должина од векторот ***D*** и тој на првите две локации ги содржи реалните делови од векторот ***D***, а на вторите две локации ги содржи имагинарните делови. Со тоа проблемот сме го свеле на решавање на систем од нелинеарни равенки со реални броеви, но е потребо, на сличен начин, да дефирниграме и вектор со решение на системот равенки кој што го има следниот изглед

```
Zrx = [
    real(Z)
    imag(Z)
    ];
```

Според условот поставен во задачата очигледно е дека почетното решение за вака дефинираниот вектор ќе биде

```
Zrx0 = [1; 1; 0; 0];
```

За разгледуваниот систем равенки ја дефинираме функцијата funkcija_10_3 која што е дадена во датотеката funkcija_10_3.m и чија што содржина е

```
function F = funkcija_10_3(Zrx,A,b)
Z = Zrx(1:2) + j * Zrx(3:4);
D = A * Z - b ./ Z;
F = [
    real(D)
    imag(D)
    ];
```

каде што во листата со аргументи на функцијата, покрај аргументот `Zrx` кој што е прв, се додадени и дополнителни аргументи `A` и `b`. Во оваа функција најпрво од реалниот вектор со двојна должина (4 елементи за 2 равенки) формираме комплексен вектор `Z` чии што реални делови се еднакви на првите 2 елемента од векторот `Zrx`, а имагинарните делови се еднакви на вторите 2 елемента од векторот `Zrx`. Потоа го пресметуваме векторот `D` и на крајот неговите реални и имагинарни делови ги сместуваме во векторот `F`.

На почетокот овие вектори ја имаат следната содржина

$$\boldsymbol{Zrx} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix},\ \underline{\boldsymbol{Z}} = \begin{bmatrix} 1+j0 \\ 1+j0 \end{bmatrix},\ \underline{\boldsymbol{D}} = \begin{bmatrix} 0+j1 \\ 0+j0 \end{bmatrix},\ \boldsymbol{F} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix},$$

од каде што се гледа дека решението не е доволно точно затоа што не се сите елементи на векторот $\boldsymbol{F}$ еднакви на нула.

Уточнувањето на решението се прави во итеративна постапка со помош на програмата fsolve чие што користење за решавање на системот комплексни нелинеарни равенки е дадена во датотеката primer_10_3.m чиј што изглед е следниот

```
Zrx0 = [1; 1; 0; 0];
A = [1 + j, 3; 2 - j, -1 + 3*j];
b = [4; 1 + 2*j];
[Zrx, fval, exitflag] = fsolve(@funkcija_10_3,Zrx0,[],A,b)
Z = Zrx(1:2) + j * Zrx(3:4)
```

Во програмата прво се дефинирани почетното решение и матриците $\boldsymbol{A}$ и $\boldsymbol{b}$, а потоа при повикувањето на функцијата fsolve по параметарот со почетното решение е зададена една празна матрица (со која што сме дефинирале дека сакаме да работиме со default опции), а потоа се додадени параметрите `A` и `b` кои што се специфични за разгледуваниот проблем. На крајот, од конечното решение, формираме комплексен вектор `Z` на ист начин како што тоа го направивме во функцијата funkcija_10_3.

Со активирањето на програмата, таа во неколку итерации ги одредува новите вредности на елементите од векторите $\boldsymbol{Zrx}$, $\underline{\boldsymbol{Z}}$, $\underline{\boldsymbol{D}}$ и $\boldsymbol{F}$ при што се добива

итерација 1:

$$\boldsymbol{Zrx} = \begin{bmatrix} 0,9392 \\ 1,0497 \\ -0,1547 \\ -0,0552 \end{bmatrix},\ \underline{\boldsymbol{Z}} = \begin{bmatrix} 0,9392 - j0,1547 \\ 1,0497 - j0,0552 \end{bmatrix},\ \underline{\boldsymbol{D}} = \begin{bmatrix} 0,0968 - j0,0641 \\ -0,0102 + j0,0058 \end{bmatrix},\ \boldsymbol{F} = \begin{bmatrix} 0,0968 \\ -0,0102 \\ -0,0641 \\ 0,0058 \end{bmatrix},$$

итерација 2:

$$\boldsymbol{Zrx} = \begin{bmatrix} 0,9331 \\ 1,0408 \\ -0,1389 \\ -0,0561 \end{bmatrix},\ \underline{\boldsymbol{Z}} = \begin{bmatrix} 0,9331 - j0,1389 \\ 1,0408 - j0,0561 \end{bmatrix},\ \underline{\boldsymbol{D}} = \begin{bmatrix} 0,0005 - j0,0013 \\ -0,0000 + j0,0002 \end{bmatrix},\ \boldsymbol{F} = \begin{bmatrix} 0,0005 \\ 0,0000 \\ -0,0013 \\ 0,0002 \end{bmatrix},$$

итн.

Од горните вредности, пресметани во различни итерации, забележуваме како програмата fsolve одредува такви вредности на елементите на векторот ***Zrx*** за кои што елементите на векторот ***F*** постепено се приближуваат до нула. За да го видиме текот на решавање на системот нелинеарни равенки е потребно да ги избришеме знаците ; на крајот од секој ред во функцијата funkcija_10_3 со што пресметаните вредности ќе бидат прикажани во командниот простор. На крајот од програмата го добиваме следниот резултат

```
Optimization terminated: first-order optimality is less than
options.TolFun.
Zrx =
    0.9329
    1.0408
   -0.1391
   -0.0562
fval =
  1.0e-006 *
    0.0217
   -0.0100
   -0.1824
    0.0180
exitflag =
     1
Z =
   0.9329 - 0.1391i
   1.0408 - 0.0562i
```

од каде што забележуваме дека со даденото решение четирите реални нелинеарни равенки равенки имаат вредности практично еднакви на нула и се исполнети условите за конвергенција (exitflag = 1).

За да се увериме дека ова е навистина точно решение во комадниот простор пишуваме

```
>> A*Z - b./Z
  1.0e-006 *
   0.0217 - 0.1824i
  -0.0100 + 0.0180i
```

од каде што гледаме дека добивме два комплексни броја чии што реални и имагинарни делови се практично еднакви на нула.

□ □ □

Пример 10.4. Да се одредат нулите на следниот систем комплексни нелинеарни равенки

$$(1+j)\underline{Z}_1+(7-j2)\underline{Z}_3=\frac{9}{\underline{Z}_1}$$

$$(2-j)\underline{Z}_1+(-1+j3)\underline{Z}_2+(3-j2)\underline{Z}_3=\frac{4-j0,5}{\underline{Z}_2},$$

$$(1-j)\underline{Z}_2+(-2+j3)\underline{Z}_3=\frac{-1+j2}{\underline{Z}_3}$$

при што е познато дека решението е такво што сите променливи имаат реални делови приближно еднакви на 1 и имагинарни делови приближно еднакви на 0.

Решение

Дадениот систем равенки, како и во претходниот пример, е можно да се напише во Matlab со користење на матрици на следниот начин

```
A * Z - b ./ Z = 0
```

каде што за матриците ***A*** и ***b*** важи

$$\boldsymbol{A}=\left[\begin{array}{c|c|c} 1+j & 0 & 7-j2 \\ \hline 2-j & -1+j3 & 3-j2 \\ \hline 0 & 1-j & -2+j3 \end{array}\right],$$

$$\boldsymbol{b}=\begin{bmatrix} 9 \\ 4-j0,5 \\ -1+j2 \end{bmatrix}.$$

Во овој пример, векторот со решението ќе има должина 6. На првите три локации тој ќе ги содржи реалните делови на комплексните броеви, а на вторите три локации ќе ги содржи имагинарните делови на комплексните броеви. Почетното решение за овој вектор, според условот поставен во задачата, ќе биде

```
Zrx0 = [1; 1; 0; 0];
```

За разгледуваниот систем равенки ја дефинираме функцијата funkcija_10_4 која што е дадена во датотеката funkcija_10_4.m и чија што содржина е

```
function F = funkcija_10_3(Zrx,A,b)
Z = Zrx(1:3) + j * Zrx(4:6);
D = A * Z - b ./ Z;
F = [
    real(D)
    imag(D)
    ];
```

од каде што се гледа дека таа практично е иста со функцијата funkcija_10_4 само што сега векторот `Zrx` има должина 6. Во првиот ред од фунцијата повторно се формира комплексен вектор `Z` за кој што сега важи дека реални делови се еднакви на првите 3 елементи од векторот `Zrx`, а имагинарните делови се еднакви на вторите 3 елементи од

векторот Zrx. Понатаму функцијата funkcija_10_4 е наполно иста со функцијата funkcija_10_3.

Програмата за решавање на системот комплексни нелинеарни равенки е дадена во датотеката primer_10_4.m и таа го има следниот изглед

```
Zrx0 = [1; 1; 1; 0; 0; 0];
A = [1 + j, 0, 7 - 2*j; 2 - j, -1 + 3*j, 3 - 2*j; 0, 1 - j, -2 + 3*j];
b = [9; 4 - 0.5*j; -1 + 2*j];
[Zrx, fval, exitflag] = fsolve(@funkcija_10_3,Zrx0,[],A,b)
Z = Zrx(1:3) + j * Zrx(4:6)
```

со нејзино активирање активирањето го добиваме следниот резултат

```
Zrx =
    1.1420
    0.8716
    0.9737
    0.1230
   -0.0649
   -0.0224
fval =
  1.0e-012 *
    0.0435
   -0.0249
    0.0004
   -0.3717
   -0.0181
         0
exitflag =
     1
Z =
   1.1420 + 0.1230i
   0.8716 - 0.0649i
   0.9737 - 0.0224i
```

Претходната програма која што функционира за дадениот пример 10.4 можеме да ја направиме да биде применлива за решавање на кој било друг проблем зададен со помош на матричната релација A * Z - b ./ Z = 0, каде што само се менуваат елементите на матрицата A и векторот b.

Ако бројот на равенки го означиме со N, тогаш векторот Z ќе го формираме така што неговиот реален дел ќе биде еднаков на векторот составен од елементите од 1 до N од векторот Zrx, додека неговиот имагинарен дел ќе биде еднаков на векторот составен од елементите од од N+1 до 2N од векторот Zrx. Соодветно на тоа фукцијата funkcija_10_4 ја со што ја добиваме функцијата funkcija_10_4a којашто го има следниот изглед

```
function F = funkcija_10_4a(Zrx,N,A,b)
Z = Zrx(1:N) + j * Zrx(N+1:2*N);
D = A * Z - b ./ Z;
F = [
    real(D)
    imag(D)
    ];
```

Потоа ја формираме функцијата resavanje_ravenki којашто го има следниот изглед

```
function [Z, exitflag] = resavanje_ravenki(A,b)
N = size(A,1);
Zrx0 = [ones(N,1); zeros(N,1)];
[Zrx, fval, exitflag] = fsolve(@funkcija_10_4a,Zrx0,[],N,A,b);
Z = Zrx(1:N) + j * Zrx(N+1:2*N);
```

Во неа прво го одредуваме бројот на равенки кој што е еднаков со бројот на редици во матрицата `A`, а потоа го формираме векторот `Zrx0` којшто, во општ случај, се состои од $2N$ елементи од кои првите N се единици додека останатите се нули. Повикувањето на функцијата fsolve е идентично како и претходно, со тоа што тука се користи функцијата funkcija_10_4a којашто е дефинирана за општ случај со N равенки. На крајот комплексниот вектор `Z` го формираме на ист начин како во функцијата funkcija_10_4a.

Користејќи ја функцијата resavanje_ravenki примерот 10.4 можеме да го решиме на следниот начин (даден во датотеката primer_10_4a.m.

```
A = [1 + j, 0, 7 - 2*j; 2 - j, -1 + 3*j, 3 - 2*j; 0, 1 - j, -2 + 3*j];
b = [9; 4 - 0.5*j; -1 + 2*j];
[Z, exitflag] = resavanje_ravenki(A,b)
```

□ □ □