Univerza v Mariboru

Fakulteta za gradbeništvo

Dean Korošak

Praktikum iz numeričnih metod z Mathematico

Maribor, 2011

CIP - Kataložni zapis o publikaciji
Univerzitetna knjižnica Maribor

519.6(075.8)(076)

KOROŠAK, Dean
Praktikum iz numeričnih metod z Mathematico
[Elektronski vir] / Dean Korošak. - El. učbenik. -
Maribor : Fakulteta za gradbeništvo, 2011

Način dostopa (URL):
http://dkum.uni-mb.si/IzpisGradiva.php?id=18342

ISBN 978-961-248-273-2

COBISS.SI-ID 66880001

Naslov:	Praktikum iz numeričnih metod z Mathematico
Avtor:	izr. prof. dr. Dean Korošak
Strokovna recenzenta:	red. prof. dr. Bruno Cvikl, izr. prof. dr. Rok Strašek
Lektorica:	mag. Suzana Jakoša, prof. slov.
Tehnični recenzent:	doc. dr. Andrej Tibaut
Računalniški prelom:	avtor
Oblikovanje slik:	avtor
Oblikovanje ovitka:	avtor
Urednik:	izr. prof. dr. Matjaž Šraml
Tipologija (COBISS) / vrsta publikacije:	Univerzitetni učbenik
Založnik:	Univerza v Mariboru, Fakulteta za gradbeništvo
Kraj založbe:	Maribor
Datum izida:	21.4.2011
Naklada:	
Različica (e-pub.):	R1
URL (e-pub.):	http://dkum.uni-mb.si/
Sistemske zahteve (e-pub.):	osebni računalnik
Programske zahteve (e-pub.):	PDF-bralnik, Mathematica 5+ (za uporabo programov)

ISBN 978-961-248-273-2

Kazalo

1 Osnove Mathematice

Mathematica je programski paket, namenjen simboličnemu in numeričnemu računanju. Del Mathematice, ki skrbi za komunikacijo z uporabnikom, se imenuje Front End, medtem ko dejansko računa Kernel (jedro), ki se požene, ko od Mathematice prvič zahtevamo rezultat.

Osnovna datoteka v Mathematici, s katero delamo, je beležka (angl. notebook) z običajno končnico .nb, ki jo sestavlja vrsta celic (angl. cell). Celice vsebujejo izraze, ki jih vnesemo sami ali pa jih ustvari Mathematica kot rezultat izvrednotenja celice. Celico izvrednotimo tako, da se postavimo s kurzorjem nekam v celico (ali jo označimo tako, da kliknemo oklepaj celice na robu beležke) in nato izberemo Kernel -> Evaluation -> Evaluate Cells ali pa pritisnemo Shift + Enter.

Primer :

```
1 + 1
```

```
2
```

Ko celico izvrednotimo, jo *Mathematica* opremi z `In[nn]=`, kar pomeni, da gre za t. i. input celico, katere vsebino je vnesel uporabnik, svoj odgovor pa *Mathematica* opremi z `Out[nn]=` . Številka nn v oklepaju pomeni zaporedno številko celice v beležnici, na katero se lahko nato sklicujemo. Izrazi v input celicah so izpisani **`poudarjeno`**, izrazi v t. i. output celicah pa v `normalni pisavi`, kot je razvidno iz prejšnjega primera. Obstaja še vrsta drugih tipov celic, kot so besedilna (kot na primer ta), grafična ... Z beležko lahko manipuliramo, kot to običajno počnemo z datotekami v okolju Windows, torej odpiramo (File -> Open), ustvarimo novo (File -> New), shranimo (File -> Save ali Save As), natisnemo (File -> Print) in podobno.

Mathematica ponuja ogromno vgrajenih funkcij in matematičnih operacij, ki jih lahko uporabimo pri računanju oziroma reševanju problemov. Po drugi strani pa je *Mathematica* močan programski jezik, ki omogoča programiranje lastnih funkcij, kontrolo izvajanja programov, interakcijo z drugimi programi ... Za uspešno uporabo in programiranje moramo dobro poznati sintakso in strukturo funkcij in ukazov. Primer za to je, denimo, raba oklepajev v *Mathematici,* ki je glede tega zelo stroga. Naslednja tabela prikazuje vrste oklepajev in njihovo uporabo v *Mathematici*:

( )	ločuje elemente v izrazih : (a + b) (c + d)
[ ]	ograjuje argument funkcije : Sin[x]
{ }	seznami (= vektorji, matrike) : {a, b, c, d}
[[ ]] oz. ⟦ ⟧	elementi seznama : {a, b, c, d}[[2]]
(* *)	komentarji : (* to je komentar *)

Imena vgrajenih funkcij in ukazov se začenjajo z veliko začetnico (Sin, Plot, For, While, Integrate ...), zato je dobro, da imena funkcij, izrazov in spremenljivk, ki jih definiramo sami, začenjamo z malo začetnico.

Nekaj primerov:

```
Sin[Pi / 4]
```

$$\frac{1}{\sqrt{2}}$$

Simbol Pi je definiran v Mathematici in predstavlja število π. *Mathematica* poskuša vsak izraz/rezultat podati natančno. Število 4 v zgornjem primeru pomeni naravno število 4, zato dobimo točen rezultat.

```
Sin[Pi / 4.]
```

```
0.707107
```

Z zapisom 4. smo funkciji Sin[x] kot argument podali realno število z omejeno natančnostjo, zato dobimo tudi rezultat z določeno natančnostjo. Računamo lahko tudi s kompleksnimi števili:

```
Sin[Pi + 2 I]
```

```
-i Sinh[2]
```

```
sin[Pi / 4]
```

```
General::spell1 : Possible spelling error: new
   symbol name "sin" is similar to existing symbol "Sin". More…
```

$$\sin\left[\frac{\pi}{4}\right]$$

Ob napakah pri vnosu nas *Mathematica* opozori. V tem primeru je simbol sin podoben vgrajeni funkciji Sin. Seveda bi lahko definirali svojo funkcijo sin

```
sin[x_] := Sin[x]
```

in jo uporabili na enak način kot vgrajeno

```
sin[Pi / 4]
```

$$\frac{1}{\sqrt{2}}$$

Zdaj opozorila ni več.

Opazimo, da v definiciji funkcije na levi strani nastopa vzorec x_, ki predstavlja katerikoli izraz (poimenovan x). Funkcije lahko v Mathematici uporabljamo na več načinov:

- postfiksno

```
(Pi / 4) // sin
```

$$\frac{1}{\sqrt{2}}$$

- prefiksno

```
sin@(Pi / 4)
```

$$\frac{1}{\sqrt{2}}$$

- lahko jih uporabimo na seznamih

```
sin[{0, Pi / 4, Pi / 2, 3 Pi / 4, Pi}]
```

$$\left\{0, \frac{1}{\sqrt{2}}, 1, \frac{1}{\sqrt{2}}, 0\right\}$$

Definiramo lahko čiste funkcije (pure functions), ki nimajo imena.

```
(#1^#2 &)[n, 3]
```

n^3

Od trenutka, ko se požene Kernel (jedro), *Mathematica* vodi seznam vseh simbolov funkcij ali izrazov, ki jih definiramo in uporabimo, tako da izvrednotimo celice. Tudi ko delo v neki beležki končamo in nadaljujemo v drugi, ostanejo v jedru vse definicije iz prve beležke. Iz spomina se izbrišejo, če jedro ustavimo s Kernel -> Quit Kernel -> Local (pride prav, kadar želimo preveriti, ali programi in izrazi v beležki s katero smo dolgo delali in jo spreminjali delujejo) ozirom prenehamo z delom v *Mathematici* (File -> Exit).

Mathematica ima dober in obsežen sistem pomoči (Help), kjer so posamezne funkcije in operacije natančno opisane, njihova uporaba pa pojasnjena s primeri. Opis določene funkcije lahko poiščemo v sistemu pomoči (Help -> Help browser ...) ali pa v beležnici, tako da pred imenom izraza postavimo vprašaj:

```
?Plot
```

```
Plot[f, {x, xmin, xmax}] generates a plot of f as a function of x from xmin to
   xmax. Plot[{f1, f2, ... }, {x, xmin, xmax}] plots several functions fi. More...
```

Iz opisa ukaza Plot razberemo, da moramo podati vsaj dva argumenta: funkcijo f, ki jo želimo narisati, in interval argumenta funkcije f v obliki seznama.

Osnovni elementi, s katerimi operiramo, so v *Mathematici* predstavljeni enotno kot *izrazi* f[x,y, ...]. Funkcija sin, ki smo jo definirali v prejšnjem primeru, je izraz, kot so tudi seznami, grafi, operacije med elementi ... Z uporabo FullForm[*izraz*] prikažemo strukturo izraza v *Mathematici*.

Definirajmo izraz z:

```
z = (a + b) c^2
```

$(a + b)\ c^2$

Prejšnji izraz je v *Mathematici* predstavljen tako:

```
FullForm[z]
```

```
Times[Plus[a, b], Power[c, 2]]
```

---

2 Linearno programiranje

Linearni program imenujemo optimizacijske probleme naslednjega tipa:

$$\begin{aligned} &\min\ (\max)\ \vec{c} \cdot \vec{x} \\ &A_1 \vec{x} \geq 0 \\ &A_2 \vec{x} \leq 0 \\ &A_3 \vec{x} = 0 \\ &\vec{x} \geq 0 \end{aligned}$$

Mathematica pozna več vgrajenih funkcij, s katerimi lahko poiščemo rešitve optimizacijske naloge: `Maximize[]`, `Minimize[]` in `LinearProgramming[]`.

■ 2.1 Vektorji, baza vektorskega prostora ...

Vektorje v *Mathematici* predstavimo s seznami, komponente zapišemo med zavitimi oklepaji {}. e1, e2, e3 in e4 so 4-dim vektorji:

```
e1 = {1, 0, 0, 0};
e2 = {1, 1, 0, 0};
e3 = {1, 1, 1, 0};
e4 = {1, 1, 1, 1};
```

Iz vektorjev sestavimo matriko tako, da tvorimo seznam vektorjev {e1, e2, e3, ,e4}. Če želimo, da so vektorji stolpci v matriki, sestavljeni seznam (= matrika) transponiramo z ukazom `Transpose[]`.

```
bb = Transpose[{e1, e2, e3, e4}];
```

Matriko bb prikažemo v pregledni obliki z ukazom `MatrixForm`, tukaj v postfiksni obliki:

```
bb // MatrixForm
```

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Preverimo, ali so linearno neodvisni (ali torej lahko sestavljajo bazo 4-dim prostora)

```
Det[bb]
```

```
1
```

in ali so ortogonalni:

```
e1.e2
```

```
1
```

Ortonormirano bazo lahko iz vektorjev tvorimo, recimo, z Gram-Schmidtovim ortogonalizacijskim postopkom. *Mathematicin* vgrajeni ukaz je v dodatnem paketu za linearno algebro, ki ga naložimo v jedro z naslednjim ukazom:

```
<< LinearAlgebra`Orthogonalization`
```

Zdaj lahko dobimo ortonormirano bazo:

```
GramSchmidt[{e1, e2, e3, e4}]
```

```
{{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}, {0, 0, 0, 1}}
```

Podobno lahko v tem primeru dosežemo z reducirano bazo:

```
b = LatticeReduce[{e1, e2, e3, e4}]
```

```
{{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}, {0, 0, 0, 1}}
```

Poskusimo še zamenjati en bazni vektor z drugim. Najprej odstranimo iz jedra definicije, ki smo jih do sedaj uporabili z ukazom Remove. Argument ukaza Global`* poišče vse definicije v globalnem kontekstu (na katerega se nanašajo vse uporabnikove definicije, če ne definira lastnega lokalnega konteksta):

```
Remove["Global`*"]
```

Definirajmo novo bazo in dva vektorja a in b:

```
baza1 = {e1, e2, e3, e4, e5};
b = {b1, b2, b3, b4, b5};
a = {a1, a2, a3, a4, a5};
```

Zamenjajmo bazni vektor e3 z novim vektorjem bb:

```
e3 = e3 /. Solve[b.baza1 == bb, e3][[1]]
```

$$\frac{\text{bb} - \text{b1 e1} - \text{b2 e2} - \text{b4 e4} - \text{b5 e5}}{\text{b3}}$$

in zapišimo vektor a v novi bazi:

```
anovi = a.baza1
```

$$\text{a1 e1} + \text{a2 e2} + \text{a4 e4} + \text{a5 e5} + \frac{\text{a3 (bb} - \text{b1 e1} - \text{b2 e2} - \text{b4 e4} - \text{b5 e5)}}{\text{b3}}$$

Če vpeljemo konkretno ortonormirano bazo (ukaz `IdentityMatrix[m]` tvori enotno matriko velikosti m):

```
{e1, e2, bb, e4, e5} = IdentityMatrix[5]
```

```
{{1, 0, 0, 0, 0}, {0, 1, 0, 0, 0}, {0, 0, 1, 0, 0}, {0, 0, 0, 1, 0}, {0, 0, 0, 0, 1}}
```

lahko, denimo, pogledamo, kako se transformira koeficient četrte komponente vektorja a ob prehodu v novo bazo:

```
anovi.e4
```

$$\text{a4} - \frac{\text{a3 b4}}{\text{b3}}$$

■ 2.2 Grafična metoda

Kadar je število spremenljivk linearnega problema dovolj majhno (dve, kvečjemu tri), lahko optimizacijski problem rešimo grafično. Denimo, da rešujemo naslednji linearni program:

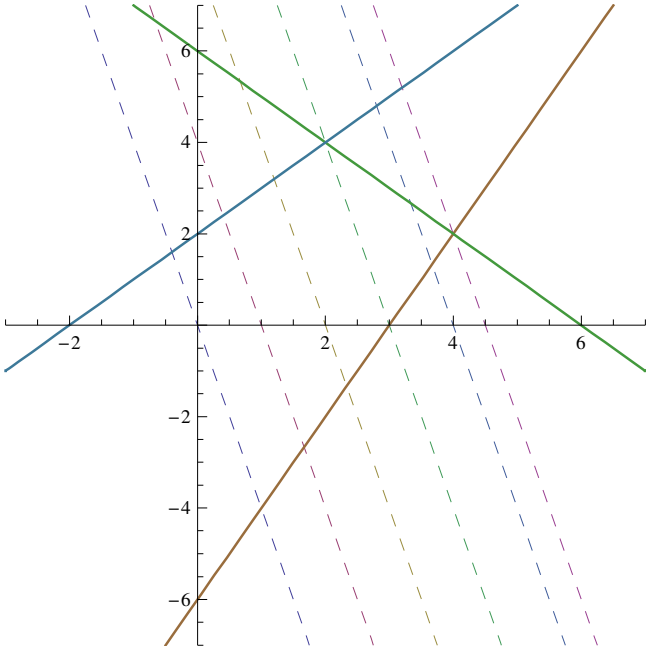
$$\begin{aligned} &\max 4x_1 + x_2 \\ &2x_1 - x_2 \le 6 \\ &x_1 + x_2 \le 6 \\ &\text{-}x_1 + x_2 \le 2 \\ &x_1,\ x_2 \ge 0 \end{aligned}$$

Vpeljimo naslednje funkcije spremenljivke x = x1, ki jih dobimo, ko izrazimo spremenljivko x2 iz zgornjih zvez:

```
g[x_, c_] := -4 x + c;
f1[x_] := 2 x - 6;
f2[x_] := 6 - x;
f3[x_] := 2 + x;
```

Definirane funkcije narišemo, namensko funkcijo narišemo pri različnih vrednostih konstante c:

```
p1 = Plot[{g[x, 0], g[x, 4], g[x, 8], g[x, 12], g[x, 16], g[x, 18], f1[x], f2[x], f3[x]},
  {x, -3, 7}, AspectRatio → 1, PlotRange → {{-3, 7}, {-7, 7}},
  PlotStyle → {Dashing[{0.02, 0.02}], Dashing[{0.02, 0.02}],
    Dashing[{0.02, 0.02}], Dashing[{0.02, 0.02}], Dashing[{0.02, 0.02}],
    Dashing[{0.02, 0.02}], Thickness[0.004], Thickness[0.004], Thickness[0.004]}]
```



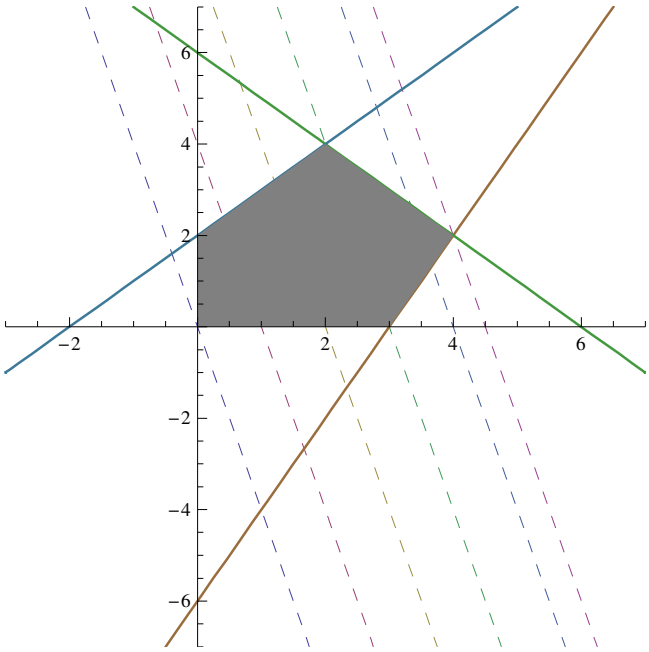
Poiščemo še presečišča funkcij, ki omejujejo območje točk množice možnih rešitev (konveksnega poligona):

```
presecisca =
 {{0, 0}, {x, f1[x]} /. Solve[f1[x] == 0, x][[1]], {x, f1[x]} /. Solve[f1[x] == f2[x], x][[1]],
  {x, f3[x]} /. Solve[f3[x] == f2[x], x][[1]], {0, f3[0]}}
```

```
{{0, 0}, {3, 0}, {4, 2}, {2, 4}, {0, 2}}
```

Prejšnji sliki dodamo še obarvano območje konveksnega poligona možnih rešitev, ki ga tvorimo z ukazom Polygon[]:

```
obmocje = Show[p1, Graphics[{GrayLevel[0.5], Polygon[presecisca]}]]
```



S slike lahko odčitamo, da ima namenska funkcija (ki jo vzporedno premikamo vstran od izhodišča) maksimalno vrednost 18 v oglišču (4,2).

Z *Mathematicino* funkcijo `Maximize[]` poiščemo rešitev tako, da kot argumenta navedemo seznam z namensko funkcijo in neenačbami ter seznam spremenljivk problema:

```
Maximize[{4 x1 + x2, 2 x1 - x2 ≤ 6, x1 + x2 ≤ 6,
  -x1 + x2 ≤ 2, x1 ≥ 0, x2 ≥ 0}, {x1, x2}]
```

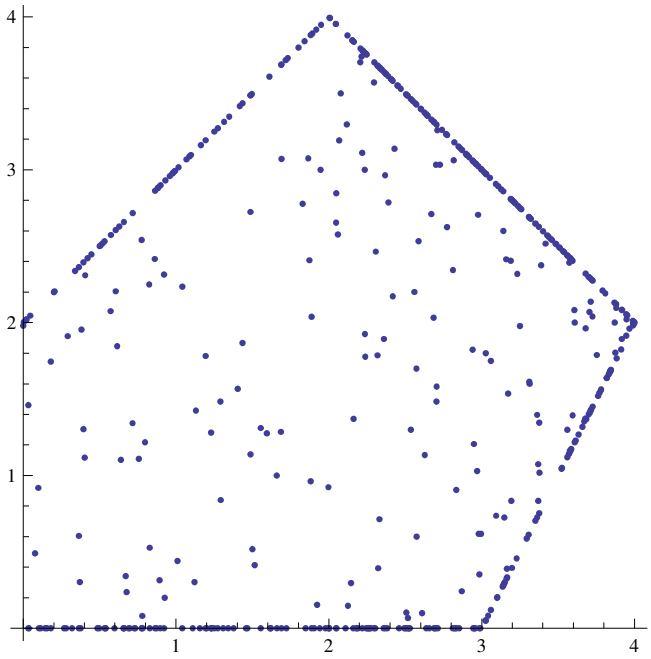
```
{18, {x1 → 4, x2 → 2}}
```

Neenačbe lahko nastopajo kot argument tudi v nekaterih drugih funkcijah *Mathematice*. Ukaz `FindInstance[]` poišče točko, ki ustreza danim pogojem. Z naslednjim ukazom poiščemo množico 500 točk, ki rešijo sistem neenačb prejšnjega problema:

```
points = {x1, x2} /. FindInstance[{2. x1 - x2 ≤ 6, x1 + x2 ≤ 6,
     -x1 + x2 ≤ 2, x1 ≥ 0, x2 ≥ 0}, {x1, x2}, 500];
```

Točke še narišemo z ukazom `ListPlot[]`:

```
ListPlot[points, AspectRatio → Automatic]
```



■ 2.3 Algoritem Simplex

Za isti linearni program kot zgoraj poiščimo rešitev z uporabo algoritma Simplex:

$$\begin{aligned} &\max 4x_1 + x_2 \\ &2x_1 - x_2 \le 6 \\ &x_1 + x_2 \le 6 \\ &\text{-}x_1 + x_2 \le 2 \\ &x_1,\ x_2 \ge 0 \end{aligned}$$

Sestavimo začetno matriko a: v vrsticah so koeficienti iz neenačb, v prvem stolpcu desna stran, nato po vrsti koeficienti spremenljivk problema, dodatnih in umetnih. Sestavimo še vrstico cj (začnemo jo z ničlo v prvem stolpcu), ki vsebuje koeficiente spremenljivk v namenski funkciji, ter vrstico ci, ki vsebuje koeficiente spremenljivk, ki jim ustrezajo začetni bazni vektorji. Na koncu izračunamo še vrstico zj.

Za zgornji problem iskanja maksimuma funkcije je konkretno:

```
a = {{6, 2, -1, 1, 0, 0}, {6, 1, 1, 0, 1, 0},
   {2, -1, 1, 0, 0, 1}};
cj = {0, 4, 1, 0, 0, 0};
ci = {0, 0, 0};
zj = ci.a - cj;
```

Z ukazom `MatrixForm[]` izpišemo začetno matriko a še v pregledni obliki:

```
MatrixForm[a]
```

$$\begin{pmatrix} 6 & 2 & -1 & 1 & 0 & 0 \\ 6 & 1 & 1 & 0 & 1 & 0 \\ 2 & -1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

To lahko storimo tudi v postfiksni obliki:

```
a // MatrixForm
```

$$\begin{pmatrix} 6 & 2 & -1 & 1 & 0 & 0 \\ 6 & 1 & 1 & 0 & 1 & 0 \\ 2 & -1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Zdaj določimo bazo, ki jo sestavljajo vektorji v zadnjih m-stolpcih matrike a, m pa je dolžina ci. Na seznam base vpišemo zaporedna števila baznih vektorjev, na seznamih baza in vektorji pa vektorje zapišemo simbolično. Števec korak postavimo na začetno vrednost 1.

```
n = Length[cj];
m = Length[ci];
korak = 1;
base = Table[(i - 1), {i, n - m + 1, n}];
baza = Table["P⃗"base[[i]], {i, 1, m}]
vektorji = Table["P⃗"i, {i, 0, n - 1}]
(*baza=Outer[NonCommutativeMultiply,{P},base][[1]];*)
```

$$\{\vec{P}_3, \vec{P}_4, \vec{P}_5\}$$

$$\{\vec{P}_0, \vec{P}_1, \vec{P}_2, \vec{P}_3, \vec{P}_4, \vec{P}_5\}$$

Sestavimo še začetno tabelo algoritma Simplex po vzoru s predavanj in z vaj, kjer začetnim vrsticam cj in ci ter matriki a ustrezno dodamo bazo in vektorje z zaporednimi ukazi `Transpose[]`, ki transponira matriko, `Prepend[]`, ki na začetek podanega seznama doda drug seznam in `Join[]`, ki združi podana seznama:

```
izpis = Transpose[Prepend[Transpose[
    Prepend[Prepend[Append[Transpose[Prepend[Transpose[a], baza]], Prepend[zj, "zj-cj"]],
      Prepend[vektorji, " "]], Prepend[cj, "cj"]]], Join[{" ", "ci"}, ci, {" "}]]]
```

$$\left\{\{\ ,\ c_j,\ 0,\ 4,\ 1,\ 0,\ 0,\ 0\},\ \{c_i,\ \ ,\ \vec{P}_0,\ \vec{P}_1,\ \vec{P}_2,\ \vec{P}_3,\ \vec{P}_4,\ \vec{P}_5\},\ \{0,\ \vec{P}_3,\ 6,\ 2,\ -1,\ 1,\ 0,\ 0\},\right.$$
$$\left.\{0,\ \vec{P}_4,\ 6,\ 1,\ 1,\ 0,\ 1,\ 0\},\ \{0,\ \vec{P}_5,\ 2,\ -1,\ 1,\ 0,\ 0,\ 1\},\ \{\ ,\ z_j\text{-}c_j,\ 0,\ -4,\ -1,\ 0,\ 0,\ 0\}\right\}$$

Z ukazom `DisplayForm[]` prikažemo začetno tabelo izpis v obliki tabele, opremljeno z okvirjem in ustreznimi črtami (ukaza `FrameBox[]` in `GridBox[]` z argumentoma `RowLines` in `ColumnLines`):

```
DisplayForm[FrameBox[GridBox[izpis,
   RowLines → {True, True, False, False, True}, ColumnLines → {False, True, True, False}]]]
```

	c_j	0	4	1	0	0	0
c_i		$\vec{P}_0$	$\vec{P}_1$	$\vec{P}_2$	$\vec{P}_3$	$\vec{P}_4$	$\vec{P}_5$
0	$\vec{P}_3$	6	2	-1	1	0	0
0	$\vec{P}_4$	6	1	1	0	1	0
0	$\vec{P}_5$	2	-1	1	0	0	1
	$z_j - c_j$	0	-4	-1	0	0	0

Novo dopustno rešitev optimizacijskega problema (geometrijsko: premaknemo se v novo oglišče konveksnega politopa) konstruiramo z zamenjavo enega izmed baznih vektorjev z nebaznim. Postopek ponavljamo, dokler so v vrstici zj-cj maksimizaciji (minimizaciji) elementi, ki so negativni (pozitivni). Zanko, v kateri zaporedno konstruiramo nove rešitve, realiziramo z ukazom `While[]`, ki ponavlja podane ukaze, dokler je pogoj (prvi argument ukaza) resničen. Transformacijo matrike a v vsakem novem koraku izvedemo z množenjem z matriko T, ki je enotna matrika (velikost matrike določa velikost baze) s transformiranim stolpcem, ki pripada novemu baznemu vektorju.

```
While[Max[Drop[-zj, 1]] > 0, (* Preverimo pogoj negativnosti elementov vrstice zj−
    cj. Funkcija Max[] vrne maksimalni element podanega seznama, funkcija Drop[] pa spusti
    elemente seznama, ki so našteti v argumentu. Torej gledamo maksimalni element seznama zj−cj,
   razen prvega, ki je trenutna vrednost namenske funkcije, zato ga spustimo. Če je,
   pomnožen z −1, večji od nič, še nismo dosegli optimalne rešitve, in telo ukaza While se izvede. *)

  Print["korak:  ", korak++]; (* Povečamo števec korakov za 1 in ga hkrati izpišemo. *)

  k = Position[Drop[zj, 1], -Max[Drop[-zj, 1]]][[1, 1]] + 1; (* Določimo vektor,
  ki gre v bazo: poiščemo položaj največjega negativnega elementa v vrstici zj−cj s funkcijo Position[],
  ki vrne položaj elementa na seznamu. *)

  theta = Table[If[a[[i, k]] > 0, a[[i, 1]] / a[[i, k]], -1], {i, 1, m}];
  r = Position[theta, Min[Complement[theta, {-1}]]][[1, 1]]; (* Določimo vektor,
  ki gre iz baze: poiščemo minimalni kvocient istoležnih komponent v k−tem in prvem stolpcu,
  s tem da upoštevamo samo pozitivne komponente,
  neustrezne pa označimo z −1 in jih izločimo s seznama kvocientov theta. *)

  t = IdentityMatrix[m]; (* t je enotna matrika reda m. *)

  For[i = 0, i < m, If[i == r, t[[i, r]] = 1 / a[[r, k]],
    t[[i, r]] = -a[[i, k]] / a[[r, k]]], i++];
  (* V matriki t zamenjamo r−ti stolpec z ustreznimi elementi. *)

  Print["T=" MatrixForm[t]]; (* Matriko t izpišemo. *)

  a = t.a; (* Transformiramo matriko a: nova matrika a je produkt transformacijske t in stare a. *)

  ci[[r]] = cj[[k]];
  base[[r]] = k - 1; (* Zamenjamo bazne vektorje in ustrezne koeficiente na seznamu ci. *)

  zj = ci.a - cj; (* Tvorimo novo vrstico zj. *)

  baza = Table["P⃗"base[[i]], {i, 1, m}]; (* Novi bazni vektorji. *)

  izpis = Transpose[Prepend[Transpose[Prepend[
       Prepend[Append[Transpose[Prepend[Transpose[a], baza]], Prepend[zj, "zj-cj"]],
        Prepend[vektorji, " "]], Prepend[cj, "cj"]]], Join[{" ", "ci"}, ci, {" "}]]];
  Print[DisplayForm[FrameBox[GridBox[izpis, RowLines → {True, True, False, False, True},
      ColumnLines → {False, True, True, False}]]]] (* Sestavimo in izpišemo novo tabelo. *)];
```

```
korak:  1
```

$$T= \begin{pmatrix} \frac{1}{2} & 0 & 0 \\ -\frac{1}{2} & 1 & 0 \\ \frac{1}{2} & 0 & 1 \end{pmatrix}$$

	c_j	0	4	1	0	0	0
c_i		$\vec{P}_0$	$\vec{P}_1$	$\vec{P}_2$	$\vec{P}_3$	$\vec{P}_4$	$\vec{P}_5$
4	$\vec{P}_1$	3	1	$-\frac{1}{2}$	$\frac{1}{2}$	0	0
0	$\vec{P}_4$	3	0	$\frac{3}{2}$	$-\frac{1}{2}$	1	0
0	$\vec{P}_5$	5	0	$\frac{1}{2}$	$\frac{1}{2}$	0	1
	$z_j - c_j$	12	0	-3	2	0	0

korak: 2

$$T=\begin{pmatrix} 1 & \frac{1}{3} & 0 \\ 0 & \frac{2}{3} & 0 \\ 0 & -\frac{1}{3} & 1 \end{pmatrix}$$

	c_j	0	4	1	0	0	0
c_i		$\vec{P}_0$	$\vec{P}_1$	$\vec{P}_2$	$\vec{P}_3$	$\vec{P}_4$	$\vec{P}_5$
4	$\vec{P}_1$	4	1	0	$\frac{1}{3}$	$\frac{1}{3}$	0
1	$\vec{P}_2$	2	0	1	$-\frac{1}{3}$	$\frac{2}{3}$	0
0	$\vec{P}_5$	4	0	0	$\frac{2}{3}$	$-\frac{1}{3}$	1
	$z_j - c_j$	18	0	0	1	2	0

Poglejmo še, kako ravnamo, kadar iščemo minimum namenske funkcije. Imamo naslednji linearni program:

$$\begin{aligned} &\min 2x_1 + 6x_2 + 8x_3 + 5x_4 \\ &4x_1 + x_2 + 2x_3 + 2x_4 \geq 80 \\ &2x_1 + 5x_2 + 4x_4 \geq 40 \\ &2x_2 + 4x_3 + 1x_4 \geq 120 \\ &x_1, x_2, x_3, x_4 \geq 0 \end{aligned}$$

Program v Mathematici je enak kot zgoraj za maksimum razen pri pogoju za funkcijo While[], ki se izvaja v korakih, dokler vrstica zj-cj vsebuje pozitivne elemente.

```
a = {{80, 4, 1, 2, 2, -1, 0, 0, 1, 0, 0},
   {40, 2, 5, 0, 4, 0, -1, 0, 0, 1, 0},
   {120, 0, 2, 4, 1, 0, 0, -1, 0, 0, 1}};
cj = {0, 2, 6, 8, 5, 0, 0, 0, 10, 10, 10};
ci = {10, 10, 10};
zj = ci.a - cj;
n = Length[cj];
m = Length[ci];
korak = 1;
```

```
base = Table[(i - 1), {i, n - m + 1, n}];
baza = Table["P⃗"base[[i]], {i, 1, m}]
vektorji = Table["P⃗"i, {i, 0, n - 1}]
(*baza=Outer[NonCommutativeMultiply,{P},base][[1]];*)
```

$$\{\vec{P}_8, \vec{P}_9, \vec{P}_{10}\}$$

$$\{\vec{P}_0, \vec{P}_1, \vec{P}_2, \vec{P}_3, \vec{P}_4, \vec{P}_5, \vec{P}_6, \vec{P}_7, \vec{P}_8, \vec{P}_9, \vec{P}_{10}\}$$

```
izpis = Transpose[Prepend[Transpose[
    Prepend[Prepend[Append[Transpose[Prepend[Transpose[a], baza]], Prepend[zj, "zj-cj"]],
      Prepend[vektorji, " "]], Prepend[cj, "cj"]]], Join[{" ", "ci"}, ci, {" "}]]]
```

$$\{\{\ ,\ c_j,\ 0,\ 2,\ 6,\ 8,\ 5,\ 0,\ 0,\ 0,\ 10,\ 10,\ 10\},$$
$$\{c_i,\ \ ,\ \vec{P}_0,\ \vec{P}_1,\ \vec{P}_2,\ \vec{P}_3,\ \vec{P}_4,\ \vec{P}_5,\ \vec{P}_6,\ \vec{P}_7,\ \vec{P}_8,\ \vec{P}_9,\ \vec{P}_{10}\},\ \{10,\ \vec{P}_8,\ 80,\ 4,\ 1,\ 2,\ 2,\ -1,\ 0,\ 0,\ 1,\ 0,\ 0\},$$
$$\{10,\ \vec{P}_9,\ 40,\ 2,\ 5,\ 0,\ 4,\ 0,\ -1,\ 0,\ 0,\ 1,\ 0\},\ \{10,\ \vec{P}_{10},\ 120,\ 0,\ 2,\ 4,\ 1,\ 0,\ 0,\ -1,\ 0,\ 0,\ 1\},$$
$$\{\ ,\ z_j-c_j,\ 2400,\ 58,\ 74,\ 52,\ 65,\ -10,\ -10,\ -10,\ 0,\ 0,\ 0\}\}$$

```
DisplayForm[FrameBox[GridBox[izpis,
   RowLines → {True, True, False, False, True}, ColumnLines → {False, True, True, False}]]]
```

	c_j	0	2	6	8	5	0	0	0	10	10	10
c_i		$\vec{P}_0$	$\vec{P}_1$	$\vec{P}_2$	$\vec{P}_3$	$\vec{P}_4$	$\vec{P}_5$	$\vec{P}_6$	$\vec{P}_7$	$\vec{P}_8$	$\vec{P}_9$	$\vec{P}_{10}$
10	$\vec{P}_8$	80	4	1	2	2	-1	0	0	1	0	0
10	$\vec{P}_9$	40	2	5	0	4	0	-1	0	0	1	0
10	$\vec{P}_{10}$	120	0	2	4	1	0	0	-1	0	0	1
	$z_j - c_j$	2400	58	74	52	65	-10	-10	-10	0	0	0

```
While[Max[Drop[zj, 1]] > 0,
  Print["korak:  ", korak++];
  k = Position[Drop[zj, 1], Max[Drop[zj, 1]]][[1, 1]] + 1;
  theta = Table[If[a[[i, k]] > 0, a[[i, 1]] / a[[i, k]], -1], {i, 1, m}];
  r = Position[theta, Min[Complement[theta, {-1}]]][[1, 1]];
  t = IdentityMatrix[m];
  For[i = 0, i < m,
   If[i == r, t[[i, r]] = 1 / a[[r, k]], t[[i, r]] = -a[[i, k]] / a[[r, k]]], i++];
  Print["T=" MatrixForm[t]];
  a = t.a;
  ci[[r]] = cj[[k]];
  base[[r]] = k - 1;
  zj = ci.a - cj;
  baza = Table["P⃗"base[[i]], {i, 1, m}];
  izpis = Transpose[Prepend[Transpose[Prepend[
       Prepend[Append[Transpose[Prepend[Transpose[a], baza]], Prepend[zj, "zj-cj"]],
        Prepend[vektorji, " "]], Prepend[cj, "cj"]]], Join[{" ", "ci"}, ci, {" "}]]];
  Print[DisplayForm[FrameBox[GridBox[izpis, RowLines → {True, True, False, False, True},
      ColumnLines → {False, True, True, False}]]]];
```

```
korak:  1
```

$$T=\begin{pmatrix} 1 & -\frac{1}{5} & 0 \\ 0 & \frac{1}{5} & 0 \\ 0 & -\frac{2}{5} & 1 \end{pmatrix}$$

	c_j	0	2	6	8	5	0	0	0	10	10	10
c_i		$\vec{P}_0$	$\vec{P}_1$	$\vec{P}_2$	$\vec{P}_3$	$\vec{P}_4$	$\vec{P}_5$	$\vec{P}_6$	$\vec{P}_7$	$\vec{P}_8$	$\vec{P}_9$	$\vec{P}_{10}$
10	$\vec{P}_8$	72	$\frac{18}{5}$	0	2	$\frac{6}{5}$	-1	$\frac{1}{5}$	0	1	$-\frac{1}{5}$	0
6	$\vec{P}_2$	8	$\frac{2}{5}$	1	0	$\frac{4}{5}$	0	$-\frac{1}{5}$	0	0	$\frac{1}{5}$	0
10	$\vec{P}_{10}$	104	$-\frac{4}{5}$	0	4	$-\frac{3}{5}$	0	$\frac{2}{5}$	-1	0	$-\frac{2}{5}$	1
	$z_j - c_j$	1808	$\frac{142}{5}$	0	52	$\frac{29}{5}$	-10	$\frac{24}{5}$	-10	0	$-\frac{74}{5}$	0

korak: 2

$$T=\begin{pmatrix} 1 & 0 & -\frac{1}{2} \\ 0 & 1 & 0 \\ 0 & 0 & \frac{1}{4} \end{pmatrix}$$

	c_j	0	2	6	8	5	0	0	0	10	10	10
c_i		$\vec{P}_0$	$\vec{P}_1$	$\vec{P}_2$	$\vec{P}_3$	$\vec{P}_4$	$\vec{P}_5$	$\vec{P}_6$	$\vec{P}_7$	$\vec{P}_8$	$\vec{P}_9$	$\vec{P}_{10}$
10	$\vec{P}_8$	20	4	0	0	$\frac{3}{2}$	-1	0	$\frac{1}{2}$	1	0	$-\frac{1}{2}$
6	$\vec{P}_2$	8	$\frac{2}{5}$	1	0	$\frac{4}{5}$	0	$-\frac{1}{5}$	0	0	$\frac{1}{5}$	0
8	$\vec{P}_3$	26	$-\frac{1}{5}$	0	1	$-\frac{3}{20}$	0	$\frac{1}{10}$	$-\frac{1}{4}$	0	$-\frac{1}{10}$	$\frac{1}{4}$
	$z_j - c_j$	456	$\frac{194}{5}$	0	0	$\frac{68}{5}$	-10	$-\frac{2}{5}$	3	0	$-\frac{48}{5}$	-13

korak: 3

$$T=\begin{pmatrix} \frac{1}{4} & 0 & 0 \\ -\frac{1}{10} & 1 & 0 \\ \frac{1}{20} & 0 & 1 \end{pmatrix}$$

	c_j	0	2	6	8	5	0	0	0	10	10	10
c_i		$\vec{P}_0$	$\vec{P}_1$	$\vec{P}_2$	$\vec{P}_3$	$\vec{P}_4$	$\vec{P}_5$	$\vec{P}_6$	$\vec{P}_7$	$\vec{P}_8$	$\vec{P}_9$	$\vec{P}_{10}$
2	$\vec{P}_1$	5	1	0	0	$\frac{3}{8}$	$-\frac{1}{4}$	0	$\frac{1}{8}$	$\frac{1}{4}$	0	$-\frac{1}{8}$
6	$\vec{P}_2$	6	0	1	0	$\frac{13}{20}$	$\frac{1}{10}$	$-\frac{1}{5}$	$-\frac{1}{20}$	$-\frac{1}{10}$	$\frac{1}{5}$	$\frac{1}{20}$
8	$\vec{P}_3$	27	0	0	1	$-\frac{3}{40}$	$-\frac{1}{20}$	$\frac{1}{10}$	$-\frac{9}{40}$	$\frac{1}{20}$	$-\frac{1}{10}$	$\frac{9}{40}$
	$z_j - c_j$	262	0	0	0	$-\frac{19}{20}$	$-\frac{3}{10}$	$-\frac{2}{5}$	$-\frac{37}{20}$	$-\frac{97}{10}$	$-\frac{48}{5}$	$-\frac{163}{20}$

Rešitev poiščimo še z vgrajeno funkcijo `Minimize[]`

```
Minimize[{2 x1 + 6 x2 + 8 x3 + 5 x4,  4 x1 + x2 + 2 x3 + 2 x4 >= 80,  2 x1 + 5 x2 + 4 x4 >= 40,
   2 x2 + 4 x3 + x4 >= 120,  x1 ≥ 0,  x2 ≥ 0,  x3 ≥ 0,  x4 ≥ 0}, {x1, x2, x3, x4}]
```

```
{262, {x1 → 5, x2 → 6, x3 → 27, x4 → 0}}
```

in še s funkcijo `LinearProgramming[]`

```
LinearProgramming[{2, 6, 8, 5}, {{4, 1, 2, 2}, {2, 5, 0, 4}, {0, 2, 4, 1}}, {80, 40, 120}]
```

```
{5, 6, 27, 0}
```

3 Nelinearne enačbe in sistemi

Osnovna funkcija, s katero lahko v večini primerov poiščemo rešitev nelinearne enačbe f(x) = 0, je `FindRoot[]`. Kot argument podamo enačbo (ali sistem enačb) ter ime spremenljivke in začetni približek. Ogledali si bomo uporabo `FindRoot[]` na nekaj primerih ter tudi, kako lahko sami sestavimo kratek program za reševanje nelinearnih enačb.

Pri iskanju rešitev nelinearnih enačb je v veliko pomoč, če lahko funkcijo narišemo in približno določimo interval, na katerem leži iskana rešitev. Ker bomo uporabili nekaj ukazov za risanje funkcij, ki niso v osnovnem paketu *Mathematice*, najprej v jedro naložimo dodatni grafični paket:

```
<< Graphics`
<< Graphics`ImplicitPlot`
```

Iščemo rešitev enačbe:

$$x+\ln(x) = 0.$$

```
koren = x /. FindRoot[x + Log[x] == 0, {x, 0.9}, WorkingPrecision → 11]
```

```
0.56714329041
```

`FindRoot` vrne rezultat v obliki pravila x→0.567..., zato uporabimo ukaz /. (ReplaceAll), s katerim spremenljivki koren priredimo rezultat funkcije. Opcija `WorkingPrecision` določi natančnost vsakega koraka. Z Accuracy, ki vrne število natančnih mest, lahko preverimo natančnost rezultata:

```
Accuracy[koren]
```

```
11.2463
```

Število decimalk pri računanju določa vrednost:

```
$MachinePrecision
```

```
15.9546
```

Opcija `EvalutationMonitor` omogoči, da spremljamo zaporedne približke:

```
priblizki = {}; FindRoot[x + Log[x] == 0, {x, 0.9},
 EvaluationMonitor :> AppendTo[priblizki, PaddedForm[x, {11, 11}]], WorkingPrecision → 11];
```

```
priblizki
```

```
{ 0.90000000000,  0.52359182321,  0.56601654173,
  0.56714257522,  0.56714329041,  0.56714329041}
```

■ 3.1 Reševanje nelinearne enačbe z iteracijo

Pri iteraciji tvorimo zaporedje približkov:

$$x^{(n+1)} = g(x^{(n)}),$$

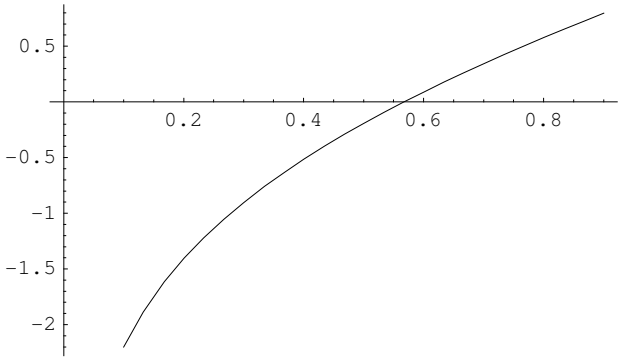
kjer dobimo funkcijo g s tranformacijo začetne enačbe f(x) = 0 v ekvivalentno x = g(x). Zaporedje konvergira na intervalu [a,b], če je $\max_{[a,b]} |g'(x)| < 1$.

Definirajmo funkciji:

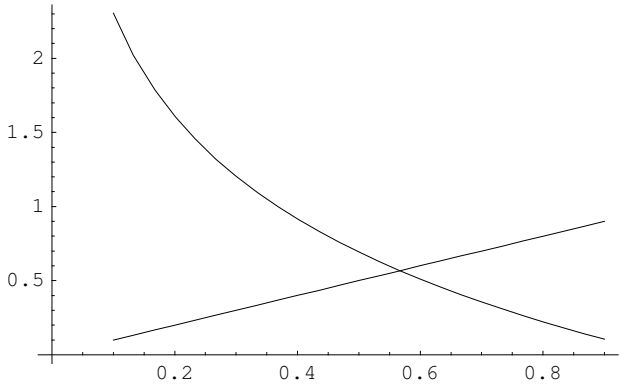
```
f[x_] := x + Log[x];
g[x_] := -Log[x];
```

in narišimo grafa f(x) in g(x) in x:

```
enacba = Plot[f[x], {x, 0.1, 0.9}];
```




```
Plot[{g[x], x}, {x, 0.1, 0.9}];
```



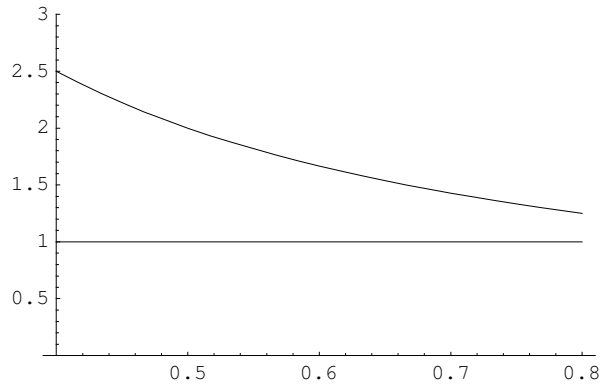
Vidimo lahko, da ima enačba x+ln(x)=0 koren med 0.55 in 0.6. Z ukazom D[] izračunamo odvod g:

```
odvod[x_] = D[g[x], x]
```

$$-\frac{1}{x}$$

Funkcija g, kot smo jo definirali zgoraj, v okolici korena ne izpolnjuje konvergenčnega kriterija:

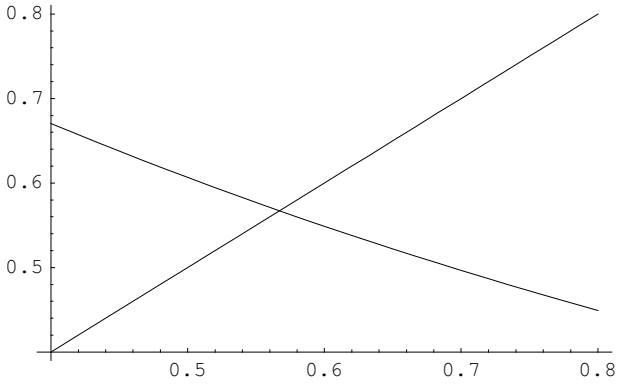
```
Plot[{Abs[odvod[x]], 1}, {x, 0.4, 0.8}, PlotRange → {All, {0, 3}}];
```



Poiščemo novo obliko g(x) in jo ponovno narišemo ter izračunamo odvod:

```
g[x_] := Exp[-x];
```

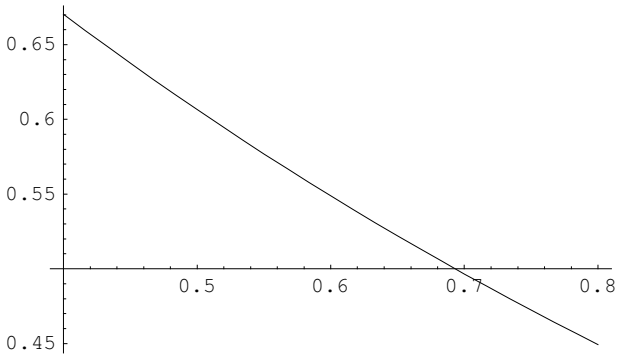
```
Plot[{x, g[x]}, {x, 0.4, 0.8}, PlotRange → All];
```




```
odvod[x_] = D[g[x], x]
```

$$-e^{-x}$$

```
Plot[Abs[odvod[x]], {x, 0.4, 0.8}];
```



Tokrat je konvergenca v okolici korena zagotovljena, zato lahko izračunamo koren z iteracijo:

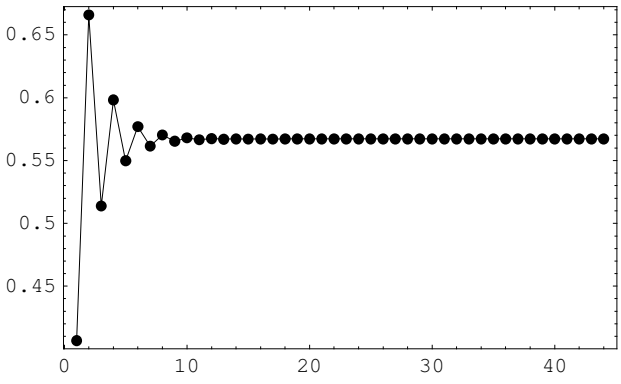
```
eps1 = eps2 = 0.2 × 10^-10; (* Zahtevana natančnost. *)
nn = 11; (* Število mest pri izpisu rezultata. *)
x1 = 0.9; (* Začetni vrednost. *)
razlika = Abs[g[x1] - x1];(* Prvi približek. *)
i = 0; (* Števec zaporednih približkov. *)
xn = x1; (* Trenutni približek. *)
(* V seznam priblizki shranimo zaporedne
 približke z ukazoma Reap in Sow znotraj zanke
 While, kjer ponavljamo iteracijo, dokler je razlika zaporednih približkov večja
 od zahtevane natančnosti eps1 oziroma dokler je vrednost f(x) večja od eps2. *)
priblizki = Reap[While[(razlika > eps1) && (Abs[f[xn]] > eps2),
    xnn = g[xn]; razlika = Abs[xnn - xn];
    (* V vsakem koraku izpišemo trenutni približek in razliko. *)
    Print[PaddedForm[i, 2], ": x", i, "=", PaddedForm[ xnn, {nn, nn}], ", ϵ=",
     ScientificForm[PaddedForm[razlika, 3]]]; xn = xnn; i++; n = i + 1; Sow[xnn]]];
```

```
 0: x0= 0.40656965974, ϵ= 4.93×10^-1
 1: x1= 0.66593070544, ϵ= 2.59×10^-1
 2: x2= 0.51379511320, ϵ= 1.52×10^-1
 3: x3= 0.59822094908, ϵ= 8.44×10^-2
 4: x4= 0.54978886895, ϵ= 4.84×10^-2
 5: x5= 0.57707163526, ϵ= 2.73×10^-2
 6: x6= 0.56154035620, ϵ= 1.55×10^-2
 7: x7= 0.57032987573, ϵ= 8.79×10^-3
 8: x8= 0.56533891635, ϵ= 4.99×10^-3
 9: x9= 0.56816755285, ϵ= 2.83×10^-3
10: x10= 0.56656268424, ϵ=  1.6×10^-3
11: x11= 0.56747267292, ϵ=  9.1×10^-4
12: x12= 0.56695651409, ϵ= 5.16×10^-4
13: x13= 0.56724922924, ϵ= 2.93×10^-4
14: x14= 0.56708321110, ϵ= 1.66×10^-4
15: x15= 0.56717736501, ϵ= 9.42×10^-5
16: x16= 0.56712396556, ϵ= 5.34×10^-5
17: x17= 0.56715425048, ϵ= 3.03×10^-5
18: x18= 0.56713707452, ϵ= 1.72×10^-5
19: x19= 0.56714681572, ϵ= 9.74×10^-6
20: x20= 0.56714129106, ϵ= 5.52×10^-6
21: x21= 0.56714442433, ϵ= 3.13×10^-6
22: x22= 0.56714264731, ϵ= 1.78×10^-6
23: x23= 0.56714365514, ϵ= 1.01×10^-6
24: x24= 0.56714308356, ϵ= 5.72×10^-7
25: x25= 0.56714340772, ϵ= 3.24×10^-7
26: x26= 0.56714322388, ϵ= 1.84×10^-7
```

```
27: x27= 0.56714332814, ϵ= 1.04×10^-7
28: x28= 0.56714326901, ϵ= 5.91×10^-8
29: x29= 0.56714330255, ϵ= 3.35×10^-8
30: x30= 0.56714328353, ϵ=  1.9×10^-8
31: x31= 0.56714329431, ϵ= 1.08×10^-8
32: x32= 0.56714328820, ϵ= 6.12×10^-9
33: x33= 0.56714329167, ϵ= 3.47×10^-9
34: x34= 0.56714328970, ϵ= 1.97×10^-9
35: x35= 0.56714329081, ϵ= 1.12×10^-9
36: x36= 0.56714329018, ϵ= 6.33×10^-10
37: x37= 0.56714329054, ϵ= 3.59×10^-10
38: x38= 0.56714329034, ϵ= 2.04×10^-10
39: x39= 0.56714329045, ϵ= 1.15×10^-10
40: x40= 0.56714329039, ϵ= 6.55×10^-11
41: x41= 0.56714329042, ϵ= 3.71×10^-11
42: x42= 0.56714329040, ϵ= 2.11×10^-11
43: x43= 0.56714329041, ϵ= 1.19×10^-11
```

Konvergenco zaporednih približkov še narišemo z `ListPlot[]`, kjer rišemo vrednosti s seznama priblizki (`Flatten[]` spusti zunanje oklepaje), graf pa oblikujemo z vrsto opcij:

```
p1 = ListPlot[Flatten[priblizki[[2]]], PlotStyle → PointSize[0.02],
  PlotJoined → False, PlotRange → All, Frame → True, DisplayFunction → Identity];
p2 = ListPlot[Flatten[priblizki[[2]]], PlotStyle → PointSize[0.02],
  PlotJoined → True, PlotRange → All, Frame → True, DisplayFunction → Identity];
Show[p1, p2, DisplayFunction → $DisplayFunction];
```



■ 3.2 Reševanje nelinearne enačbe z regula falsi in s sekantno metodo

Pri metodi regula falsi tvorimo zaporedje približkov:

$$x^{(n+1)} = x^{(n)} - \frac{f\left(x^{(n)}\right)\left(c - x^{(n)}\right)}{f(c) - f\left(x^{(n)}\right)},$$

kjer sta c in začetni približek konca intervala, na katerem leži rešitev. Pri sekantni metodi fiksno točko c nadomestimo s približkom v prejšnem koraku $x^{(n-1)}$.

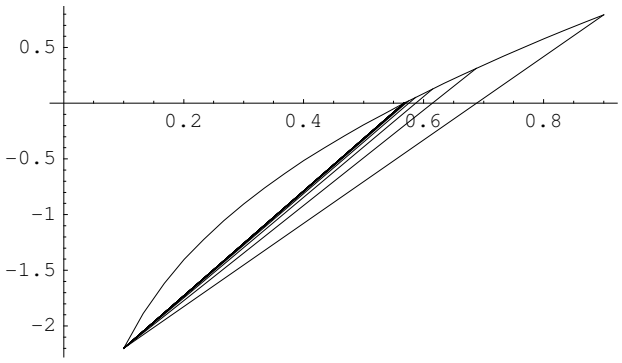
```
nn = 11; eps = .1 × 10^-10; (* Število mest pri izpisu in natančnost računanja. *)
inval = {}; (* Seznam, na katerega dodajamo vmesne približke. *)
c = 0.1; x0 = xn = .9; (* Fiksna točka in začetni približek. *)
inval = AppendTo[inval, {{c, f[c]}, {xn, f[xn]}}];
(* Zapišemo začetne vrednosti in približke. *)
razlika = Abs[c - x0]; (* Začetna razlika. *)
i = 0; (* Števec zaporednih približkov. *)
Print["n ", "xn"];
Print[i, " ", xn];
(* Zanka, v kateri računamo zaporedne približke do zahtevane vrednosti in na seznam
 inval dodajamo izračunane približke v vsakem koraku ter jih izpišemo. *)
While[razlika > eps, xnn = xn - f[xn] (c - xn) / (f[c] - f[xn]); razlika = Abs[xnn - xn];
 Print[i, " ", PaddedForm[xnn, {nn, nn}],
  ",  eps=", ScientificForm[PaddedForm[razlika, 3]]];
 xn = xnn; inval = AppendTo[inval, {{c, f[c]}, {xn, f[xn]}}]; i++]
```

```
n xn
0 0.9
0  0.68789991505,  eps= 2.12 × 10^-1
1  0.61458966766,  eps= 7.33 × 10^-2
2  0.58637144745,  eps= 2.82 × 10^-2
3  0.57503370412,  eps= 1.13 × 10^-2
4  0.57039780476,  eps= 4.64 × 10^-3
5  0.56848849887,  eps= 1.91 × 10^-3
6  0.56769979908,  eps= 7.89 × 10^-4
7  0.56737359948,  eps= 3.26 × 10^-4
8  0.56723861721,  eps= 1.35 × 10^-4
9  0.56718274939,  eps= 5.59 × 10^-5
10  0.56715962423,  eps= 2.31 × 10^-5
11  0.56715005177,  eps= 9.57 × 10^-6
12  0.56714608928,  eps= 3.96 × 10^-6
13  0.56714444901,  eps= 1.64 × 10^-6
14  0.56714377001,  eps= 6.79 × 10^-7
15  0.56714348894,  eps= 2.81 × 10^-7
16  0.56714337259,  eps= 1.16 × 10^-7
17  0.56714332443,  eps= 4.82 × 10^-8
18  0.56714330449,  eps= 1.99 × 10^-8
19  0.56714329624,  eps= 8.25 × 10^-9
20  0.56714329282,  eps= 3.42 × 10^-9
21  0.56714329141,  eps= 1.41 × 10^-9
22  0.56714329082,  eps= 5.85 × 10^-10
```

```
23  0.56714329058,  eps= 2.42×10^-10
24  0.56714329048,  eps=   1.×10^-10
25  0.56714329044,  eps= 4.15×10^-11
26  0.56714329042,  eps= 1.72×10^-11
27  0.56714329041,  eps= 7.12×10^-12
```

Na seznamu inval so začetne in končne točke intervala, na katerem je rešitev za vsak korak. Grafično prikažemo konvergenco k rešitvi z risanjem črt med koncema intervala. Uporabimo ukaz `Line[]` in vse črte zbreremo v seznam crte, ki ga v naslednjem ukazu narišemo skupaj z grafom enačbe.

```
crte = Table[Line[inval[[i]]], {i, 1, Length[inval]}];

Show[enacba, Graphics[crte]];
```



Rešitev poiščemo še s sekantno metodo:

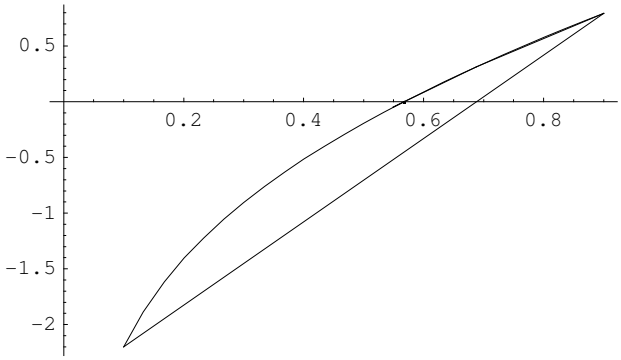
```
c = 0.1; x0 = xn = .9;
inval = {};
inval = AppendTo[inval, {{c, f[c]}, {xn, f[xn]}}];
razlika = Abs[c - x0];
i = 0;
Print["n ", "xn"];
Print[i, " ", xn];
While[razlika > eps, xnn = xn - f[xn] (c - xn) / (f[c] - f[xn]); razlika = Abs[xnn - xn];
 Print[i, " ", PaddedForm[xnn, {nn, nn}],
  ",  eps=", ScientificForm[PaddedForm[razlika, 3]]];
 c = xn; (* Fiksno točko zamenjamo s prejšnim približkom. *)
 xn = xnn; inval = AppendTo[inval, {{c, f[c]}, {xn, f[xn]}}]; i++]
```

```
n xn
0 0.9
0  0.68789991505,  eps= 2.12×10^-1
1  0.54949032185,  eps= 1.38×10^-1
2  0.56827490980,  eps= 1.88×10^-2
3  0.56715464407,  eps= 1.12×10^-3
4  0.56714328319,  eps= 1.14×10^-5
5  0.56714329041,  eps= 7.22×10^-9
6  0.56714329041,  eps= 4.62×10^-14
```

Zdaj je konvergenca veliko boljša, hitro pridemo v bližino rešitve, kar se vidi tudi iz grafične predstavitve približevanja rešitvi:

```
crte = Table[Line[inval[[i]]], {i, 1, Length[inval]}];
```

```
Show[enacba, Graphics[crte]];
```



■ 3.3 Reševanje nelinearne enačbe z Newtonovo (s tangentno) metodo

Pri Newtonovi metodi tvorimo naslednje zaporedje približkov:

$$x^{(n+1)} = x^{(n)} - \frac{f\left(x^{(n)}\right)}{f'\left(x^{(n)}\right)}.$$

Določiti moramo odvod funkcije, katere ničle iščemo:

```
df[x_] = D[f[x], x];


nn = 11; (* Število mest izpisa. *)
eps = 0.1 × 10^-10;
x0 = xn = 0.9; (* Začetni približek. *)
razlika = Abs[x0];
i = 0; (* Števec zaporednih približkov. *)
Print["n ", "xn"];
Print[i, " ", PaddedForm[xn, {nn, nn}]];
(* Zanka, v kateri računamo zaporedne približke,
dokler je razlika zaporednih približkov večja od zahtevane natančnosti. *)
While[razlika >= eps,
  i++;
  xnn = xn - f[xn] / df[xn];
  razlika = Abs[xnn - xn];
  Print[i, " ", PaddedForm[xnn, {nn, nn}],
   ",  eps=", ScientificForm[PaddedForm[razlika, 3]]];
  xn =
   xnn];
```

```
n xn
0  0.90000000000
1  0.52359182321,  eps= 3.76 × 10^-1
2  0.56601654173,  eps= 4.24 × 10^-2
3  0.56714257522,  eps= 1.13 × 10^-3
4  0.56714329041,  eps= 7.15 × 10^-7
5  0.56714329041,  eps= 2.88 × 10^-13
```

Tokrat pridemo do rešitve enako hitro kot pri vgrajeni funkciji `FindRoot[]`. Primerjajmo zaporedje izračunanih približkov s tistim, ki smo ga dobili kot rezultat uporabe vgrajene funkcije `FindRoot[]`:

```
priblizki
```

```
{ 0.90000000000,  0.52359182321,  0.56601654173,
  0.56714257522,  0.56714329041,  0.56714329041}
```

■ 3.4 Reševanje sistema nelinearnih enačb z iteracijo

S `FindRoot[]` rešujemo tudi sisteme nelinearnih enačb:

```
priblizki = {};
koren = FindRoot[{x^3 + y^3 - 6 x + 3 == 0, x^3 - y^3 - 6 y + 2 == 0}, {{x, 0.5}, {y, 0.33}},
  EvaluationMonitor :> AppendTo[priblizki, PaddedForm[{x, y}, {8, 8}]],
  WorkingPrecision → 8]
```

```
{x → 0.53237037, y → 0.35125745}
```

```
priblizki // TableForm
```

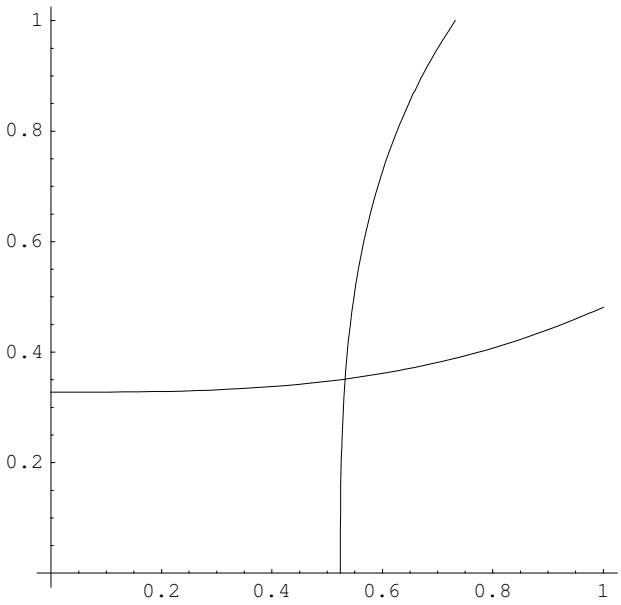
```
{ 0.50000000,  0.33000000}
{ 0.53196318,  0.35102761}
{ 0.53237031,  0.35125741}
{ 0.53237031,  0.35125741}
{ 0.53237037,  0.35125745}
```

Definirajmo funkciji f1 in f2 tako, da sta rešitvi sistema rešitvi enačb f1 = 0 in f2 = 0:

```
f1[x_, y_] := x^3 + y^3 - 6 x + 3;
f2[x_, y_] := x^3 - y^3 - 6 y + 2;
```

Ukaz ImplicitPlot[] nariše implicitno podano funkcijo. Na podlagi grafov obeh funkcij lokaliziramo rešitev in določimo začetni približek.

```
ImplicitPlot[{f1[x, y] == 0, f2[x, y] == 0}, {x, 0, 1}, {y, 0, 1}];
```



Z iteracijo pridemo do rešitve s konstrukcijo funkcij g1 in g2, ki ju tvorimo iz enačb f1 = 0 in f2 = 0 tako, da velja x = g1 in y = g2.

```
g1[x_, y_] := (x^3 + y^3 + 3) / 6;
g2[x_, y_] := (x^3 - y^3 + 2) / 6;
```

Sestavimo Jacobijevo matriko:

```
j[x_, y_] = {{D[g1[x, y], x], D[g1[x, y], y]}, {D[g2[x, y], x], D[g2[x, y], y]}};
```

```
j[x, y] // MatrixForm
```

$$\begin{pmatrix} \frac{x^2}{2} & \frac{y^2}{2} \\ \frac{x^2}{2} & -\frac{y^2}{2} \end{pmatrix}$$

in pripadajočo normo:

```
q[x_, y_] := Max[Table[{Sum[Abs[j[x, y][[i, k]]], {k, 1, 2}]}, {i, 1, 2}]];
```

s katero preverimo konveregenčni pogoj q < 1

```
q[5 / 6, 1 / 2]
```

$$\frac{17}{36}$$

Tvorimo zaporedje približkov:

```
nn = 8; (* Število mest izpisa. *)
x0 = xn = 0.5; y0 = yn = 1 / 3.; (* Začetna približka. *)
eps = 10^-8; (* Zahtevana natančnost. *)
razlika = Max[Abs[x0], Abs[y0]];
i = 0;
rl = {}; (* Seznam, na katerega dodajamo zaporedne približke. *)
Print[i, ":", xn, "  ", yn];
(* V zanki ponavljamo iteracijo,
dokler je razlika zaporednih približkov večja od zahtevane natančnosti. *)
While[razlika > eps, i++;
  xnn = g1[xn, yn]; ynn = g2[xn, yn];
  razlika = Max[Abs[xnn - xn], Abs[ynn - yn]];
  rl = Append[rl, N[razlika, nn]];
  xn = xnn; yn = ynn;
  Print[i, ":", PaddedForm[xn, {nn, nn}], "  ", PaddedForm[yn, {nn, nn}]]];
```

```
0:0.5  0.333333
1: 0.52700617   0.34799383
2: 0.53141838   0.35070440
3: 0.53220164   0.35115685
4: 0.53234027   0.35123975
5: 0.53236501   0.35125427
6: 0.53236942   0.35125688
7: 0.53237020   0.35125735
8: 0.53237034   0.35125743
9: 0.53237037   0.35125744
10: 0.53237037   0.35125745
```

```
rl
```

$$\{0.0270062,\ 0.00441221,\ 0.000783264,\ 0.000138622,\ 0.0000247493,\ 4.40292\times10^{-6},\ 7.85\times10^{-7},\ 1.39795\times10^{-7},\ 2.49113\times10^{-8},\ 4.43758\times10^{-9}\}$$

■ 3.5 Reševanje sistema nelinearnih enačb z Newtonovo metodo

Zaporedne približke po Newtonovi metodi za reševanje sistemov nelinearnih enačb tvorimo tako:

$$\vec{x}^{(n+1)} = \vec{x}^{(n)} - J\left(\vec{f}\left(\vec{x}^{(n)}\right)\right)^{-1} \vec{f}\left(\vec{x}^{(n)}\right).$$

Definiramo Jacobijevo matriko sistema:

```
j[x_, y_] = {{D[f1[x, y], x], D[f1[x, y], y]}, {D[f2[x, y], x], D[f2[x, y], y]}};
```

in njen inverz:

```
invj[x_, y_] = Inverse[j[x, y]]
```

$$\left\{\left\{\frac{-6-3\,y^2}{36-18\,x^2+18\,y^2-18\,x^2\,y^2},\ -\frac{3\,y^2}{36-18\,x^2+18\,y^2-18\,x^2\,y^2}\right\},\right.$$
$$\left.\left\{-\frac{3\,x^2}{36-18\,x^2+18\,y^2-18\,x^2\,y^2},\ \frac{-6+3\,x^2}{36-18\,x^2+18\,y^2-18\,x^2\,y^2}\right\}\right\}$$

Tvorimo zaporedje približkov in na seznam rl dodajamo razliko zaporednih približkov.

```
nn = 8; (* Število mest izpisa. *)
x0 = xn = {0.5, 0.33}; (* Začetni približek. *)
eps = 10^-8; (* Zahtevana natančnost. *)
razlika = Sqrt[x0.x0];
i = 0;
rl = {};
Print[PaddedForm[xn, {nn, nn}]];
While[(razlika > eps), i++;
  xnn = xn - invj[xn[[1]], xn[[2]]].{f1[xn[[1]], xn[[2]]], f2[xn[[1]], xn[[2]]]};
  (* . pomeni produkt matrike z vektorjem. *)
  razlika = Sqrt[(xnn - xn).(xnn - xn)];
  (* Tukaj je . skalarni produkt vektorjev približkov. *)
  rl = Append[rl, Max[Abs[xnn[[1]] - xn[[1]]], Abs[xnn[[2]] - xn[[2]]]]];
  xn = xnn;
  Print[PaddedForm[xn, {nn, nn}]]];
```

```
{ 0.50000000,  0.33000000}
{ 0.53196318,  0.35102761}
{ 0.53237031,  0.35125741}
{ 0.53237037,  0.35125745}
{ 0.53237037,  0.35125745}
```

```
rl
```

$$\{0.0319632,\ 0.000407122,\ 6.51617\times10^{-8},\ 1.66533\times10^{-15}\}$$

Zaporedje izračunanih približkov primerjamo z rezultati `FindRoot[]`:

```
priblizki // TableForm
```

```
{ 0.50000000,  0.33000000}
{ 0.53196318,  0.35102761}
{ 0.53237031,  0.35125741}
{ 0.53237031,  0.35125741}
{ 0.53237037,  0.35125745}
```

4 Sistemi linearnih enačb

Splošna naloga je poiskati rešitev sistema linearnih enačb, ki ga lahko zapišemo v matrični obliki:

$$A\vec{x}=\vec{b}, \qquad \text{kjer je } A=\begin{pmatrix} a_{11} \ldots & a_{1n} \\ & \\ a_{n1} \ldots & a_{nn} \end{pmatrix}, \quad \vec{b}=(b_1, \ldots, b_n)^T.$$

V *Mathematici* so nam na voljo številne funkcije in ukazi, s katerimi lahko rešimo zgornji problem oziroma transformiramo matriko koeficientov enačb v primernejšo obliko. Iščemo lahko tudi lastne vrednosti in lastne vektorje, rešujemo homogene sisteme. Z `LinearSolve[]` rešimo splošni linearni sistem, algoritem uporablja Gaussovo eliminacijo z delnim pivotiranjem, matriko lahko razcepimo z `LUDecomposition[]` v produkt zgornje in spodnje trikotne, z `QRDecomposition[]` v produkt ortogonalne in trikotne, `Eigenvalues[]` in `Eigenvectors[]` vrneta lastne vrednosti in lastne vektorje ...

■ 4.1 LU-dekompozicija

Najprej v jedro naložimo nekatere naprednejše funkcije za delo z matrikami:

```
<< LinearAlgebra`MatrixManipulation`
```

Definirajmo matriko a:

```
a = {{24, 12, 36, 6}, {12, 9, 15, 4},
   {8, 6, 13, 2}, {6, 5, 8, 2}};
```

in poiščimo njen razcep v zgornjo in spodnjo trikotno matriko ter rešitev sistema z:

```
b = {3, 5, 2, 3};
```

Rešitev poiščemo z `LinearSolve[]`:

```
x = LinearSolve[a, b]
```

$$\left\{-3,\ 0,\ 1,\ \frac{13}{2}\right\}$$

n je dimenzija matrike:

```
n = Dimensions[a][[1]]
```

```
4
```

Matrika l bo spodnja trikotna matrika, na začetku jo definiramo kot enotno matriko dimenzije n:

```
l = IdentityMatrix[n]
```

```
{{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}, {0, 0, 0, 1}}
```

```
a // MatrixForm
```

$$\begin{pmatrix} 24 & 12 & 36 & 6 \\ 12 & 9 & 15 & 4 \\ 8 & 6 & 13 & 2 \\ 6 & 5 & 8 & 2 \end{pmatrix}$$

```
(* LU-dekompozicijo izvedemo s tremi zankami For,
v katerih se pomikamo po vrsticah in kolonah v vsakem koraku eliminacije,
hkrati pa v matriko l vpisujemo koeficiente spodnje trikotne matrike. *)
For[r = 1, r ≤ n - 1, (* r je tekoča vrstica. *)
 For[i = r + 1, i ≤ n,
  For[k = r + 1, k ≤ n,
   a[[i, k]] = a[[i, k]] - a[[i, r]] a[[r, k]] / a[[r, r]];
   (* Matrika a bo zgornja trikotna. *)
   k++];
  l[[i, r]] = a[[i, r]] / a[[r, r]]; (* l bo spodnja trikotna matrika. *)
  a[[i, r]] = 0;
  i++];
 Print[MatrixForm[a]];
 (* Izpišemo transformirano matriko a na vsakem koraku eliminacije. *)
 r++]
```

$$\begin{pmatrix} 24 & 12 & 36 & 6 \\ 0 & 3 & -3 & 1 \\ 0 & 2 & 1 & 0 \\ 0 & 2 & -1 & \frac{1}{2} \end{pmatrix}$$

$$\begin{pmatrix} 24 & 12 & 36 & 6 \\ 0 & 3 & -3 & 1 \\ 0 & 0 & 3 & -\frac{2}{3} \\ 0 & 0 & 1 & -\frac{1}{6} \end{pmatrix}$$

$$\begin{pmatrix} 24 & 12 & 36 & 6 \\ 0 & 3 & -3 & 1 \\ 0 & 0 & 3 & -\frac{2}{3} \\ 0 & 0 & 0 & \frac{1}{18} \end{pmatrix}$$

a in l sta zdaj zgornja in spodnja trikotna matrika:

```
MatrixForm[a]
MatrixForm[l]
```

$$\begin{pmatrix} 24 & 12 & 36 & 6 \\ 0 & 3 & -3 & 1 \\ 0 & 0 & 3 & -\frac{2}{3} \\ 0 & 0 & 0 & \frac{1}{18} \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{2} & 1 & 0 & 0 \\ \frac{1}{3} & \frac{2}{3} & 1 & 0 \\ \frac{1}{4} & \frac{2}{3} & \frac{1}{3} & 1 \end{pmatrix}$$

Njun produkt pa je nerazcepljena matrika a:

```
l.a // MatrixForm
```

$$\begin{pmatrix} 24 & 12 & 36 & 6 \\ 12 & 9 & 15 & 4 \\ 8 & 6 & 13 & 2 \\ 6 & 5 & 8 & 2 \end{pmatrix}$$

Rešitev dobimo z reševanjem dveh sistemov s trikotnima matrikama:

```
y = LinearSolve[l, b]
```

$$\left\{3, \frac{7}{2}, -\frac{4}{3}, \frac{13}{36}\right\}$$

```
x1 = LinearSolve[a, y]
```

$$\left\{-3, 0, 1, \frac{13}{2}\right\}$$

```
x == x1
```

```
True
```

```
a = l.a;
```

Funkcija `LUDecomposition[]` vrne razcep matrike, permutacijski vektor, ki vsebuje informacijo o zamenjavi vrstic pri pivotiranju, ter oceno pogojenosti matrike:

```
{lu, permutacija, pogojenost} = LUDecomposition[a]
```

$$\left\{\left\{\{6, 5, 8, 2\}, \left\{\frac{4}{3}, -\frac{2}{3}, \frac{7}{3}, -\frac{2}{3}\right\}, \left\{2, \frac{3}{2}, -\frac{9}{2}, 1\right\}, \left\{4, 12, \frac{16}{3}, \frac{2}{3}\right\}\right\}, \{4, 3, 2, 1\}, 1\right\}$$

Z vektorjem permutacije vrstic tvorimo permutacijsko matriko p:

```
p = IdentityMatrix[n][[permutacija]]
```

```
{{0, 0, 0, 1}, {0, 0, 1, 0}, {0, 1, 0, 0}, {1, 0, 0, 0}}
```

Matrika, ki je bila razcepljena, je produkt permutacijske matrike in matrike a:

```
p.a // MatrixForm
```

$$\begin{pmatrix} 6 & 5 & 8 & 2 \\ 8 & 6 & 13 & 2 \\ 12 & 9 & 15 & 4 \\ 24 & 12 & 36 & 6 \end{pmatrix}$$

Z `LUMatrices[]` dobimo spodnjo in zgornjo trikotno matriko v eksplicitnem zapisu:

```
{l, u} = LUMatrices[lu]
```

$$\left\{\left\{\{1, 0, 0, 0\}, \left\{\frac{4}{3}, 1, 0, 0\right\}, \left\{2, \frac{3}{2}, 1, 0\right\}, \left\{4, 12, \frac{16}{3}, 1\right\}\right\},\right.$$
$$\left.\left\{\{6, 5, 8, 2\}, \left\{0, -\frac{2}{3}, \frac{7}{3}, -\frac{2}{3}\right\}, \left\{0, 0, -\frac{9}{2}, 1\right\}, \left\{0, 0, 0, \frac{2}{3}\right\}\right\}\right\}$$

```
MatrixForm[l]
```

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ \frac{4}{3} & 1 & 0 & 0 \\ 2 & \frac{3}{2} & 1 & 0 \\ 4 & 12 & \frac{16}{3} & 1 \end{pmatrix}$$

```
MatrixForm[u]
```

$$\begin{pmatrix} 6 & 5 & 8 & 2 \\ 0 & -\frac{2}{3} & \frac{7}{3} & -\frac{2}{3} \\ 0 & 0 & -\frac{9}{2} & 1 \\ 0 & 0 & 0 & \frac{2}{3} \end{pmatrix}$$

Z LUBackSubstitution[] rešimo sistem:

```
x2 = LUBackSubstitution[{lu, permutacija, pogojenost}, b]
```

$$\left\{-3,\ 0,\ 1,\ \frac{13}{2}\right\}$$

Za razcep permutirane matrike s prej zapisanim programom dobimo enak rezultat:

```
a = p.a; (* Začetna matrika je permutirana. *)

For[r = 1, r ≤ n - 1, (* r je tekoča vrstica. *)
 For[i = r + 1, i ≤ n,
  For[k = r + 1, k ≤ n,
   a[[i, k]] = a[[i, k]] - a[[i, r]] a[[r, k]] / a[[r, r]];
   (* Matrika a bo zgornja trikotna. *)
   k++];
  l[[i, r]] = a[[i, r]] / a[[r, r]]; (* l bo spodnja trikotna matrika. *)
  a[[i, r]] = 0;
  i++];
 Print[MatrixForm[a]];
 (* Izpišemo transformirano matriko a na vsakem koraku eliminacije. *)
 r++]
```

$$\begin{pmatrix} 6 & 5 & 8 & 2 \\ 0 & -\frac{2}{3} & \frac{7}{3} & -\frac{2}{3} \\ 0 & -1 & -1 & 0 \\ 0 & -8 & 4 & -2 \end{pmatrix}$$

$$\begin{pmatrix} 6 & 5 & 8 & 2 \\ 0 & -\frac{2}{3} & \frac{7}{3} & -\frac{2}{3} \\ 0 & 0 & -\frac{9}{2} & 1 \\ 0 & 0 & -24 & 6 \end{pmatrix}$$

$$\begin{pmatrix} 6 & 5 & 8 & 2 \\ 0 & -\frac{2}{3} & \frac{7}{3} & -\frac{2}{3} \\ 0 & 0 & -\frac{9}{2} & 1 \\ 0 & 0 & 0 & \frac{2}{3} \end{pmatrix}$$

```
l // MatrixForm
```

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ \frac{4}{3} & 1 & 0 & 0 \\ 2 & \frac{3}{2} & 1 & 0 \\ 4 & 12 & \frac{16}{3} & 1 \end{pmatrix}$$

4.2 Choleskyjeva dekompozicija

Za pozitivno definitne matrike uporabimo razcep Choleskega. Definirajmo matriko a:

```
a = {{9, 12, 6}, {12, 17, 13}, {6, 13, 65}};
```

Preverimo, ali so lastne vrednosti matrike pozitivne. Funkcija `Positive` vrne True, če je njen argument pozitivno število, `Eigenvalues[]` pa poišče lastne vrednosti matrike. Ukaz `Map[f,izraz]` deluje s funkcijo f na vsak element izraza:

```
Map[Positive, Eigenvalues[a]]
```

```
{True, True, True}
```

Ali je matrika a hermitska?

```
Conjugate[Transpose[a]] == a
```

```
True
```

u bo zgornja trikotna matrika velikosti dim(a). Na začetku je u = 0:

```
u = IdentityMatrix[Length[a]] - IdentityMatrix[Length[a]];

(* Zgornjo trikotno matriko tvorimo po razcepu Choleskega v dveh zankah For. *)
For[i = 1, i ≤ Length[a],
  u[[i, i]] = Sqrt[a[[i, i]] - Sum[u[[j, i]]^2, {j, 1, i - 1}]];
  For[k = i + 1, k ≤ Length[a],
   u[[i, k]] = (a[[i, k]] - Sum[u[[j, i]] u[[j, k]],
        {j, 1, i - 1}]) / u[[i, i]];
   k++];
  i++];
```

Rezultat dekompozicije je:

```
u // MatrixForm
```

$$\begin{pmatrix} 3 & 4 & 2 \\ 0 & 1 & 5 \\ 0 & 0 & 6 \end{pmatrix}$$

Z vgrajeno funkcijo dobimo enak rezultat:

```
CholeskyDecomposition[a] // MatrixForm
```

$$\begin{pmatrix} 3 & 4 & 2 \\ 0 & 1 & 5 \\ 0 & 0 & 6 \end{pmatrix}$$

Produkt transponirane u in u je enak nerazcepljeni matriki a:

```
Transpose[u].u
```

```
{{9, 12, 6}, {12, 17, 13}, {6, 13, 65}}
```

■ 4.3 Jacobijeva iteracija

Rešitev linearnega sistema lahko računamo iterativno z zaporednimi približki, ki konvergirajo k rešitvi. V matrični obliki razdelimo matriko na vsoto zgornje in spodnje trikotne in diagonalne matrike a=s+z+d. Jacobijevo iteracijo dobimo z zaporedjem približkov:

$$\mathrm{d}\,\vec{x}^{\,(n+1)}=\vec{b}-(\mathrm{s}+\mathrm{z})\vec{x}^{\,(n)},$$

kar lahko zapišemo tudi tako:

$$\vec{x}^{\,(n+1)}=\mathrm{m}\,\vec{x}^{\,(n)}+\vec{c}.$$

```
a = {{10, -1, -2}, {-1, 10, -3}, {-2, -3, 10}};
b = {7, 6, 5};
```

n je diagonalna matrika, ki vsebuje elemente diagonale matrike a. Funkcija `Tr[]` (trace) brez opcije List izračuna sled matrike, z vključeno opcijo List pa vrne seznam elementov na diagonali matrike:

```
n = DiagonalMatrix[Tr[a, List]]
```

```
{{10, 0, 0}, {0, 10, 0}, {0, 0, 10}}
```

Sestavimo še dve pomožni matriki:

```
p = n - a
```

```
{{0, 1, 2}, {1, 0, 3}, {2, 3, 0}}
```

```
m = Inverse[n].p
```

$$\left\{\left\{0,\ \frac{1}{10},\ \frac{1}{5}\right\},\ \left\{\frac{1}{10},\ 0,\ \frac{3}{10}\right\},\ \left\{\frac{1}{5},\ \frac{3}{10},\ 0\right\}\right\}$$

in izračunamo maksimum normo matrike m, s katero množimo zaporedni približek v vsakem koraku:

```
Norm[m, Infinity]
```

$$\frac{1}{2}$$

Ker je $\|\mathrm{m}\|_{\infty}<1$, zaporedje približkov konvergira. Izračunamo še vektor $\vec{c}$:

```
c = Inverse[n].b
```

$$\left\{\frac{7}{10},\ \frac{3}{5},\ \frac{1}{2}\right\}$$

Izberemo začetni približek

```
x0 = {0.0, 0.0, 0.0};
```

in definiramo pravilo za računanje približkov:

```
jacobi[x_] := m.x + c;
```

Zaporedje približkov tvorimo s funkcijo NestList[f,g,n], ki uporabi n + 1 krat funkcijo f na izrazu g:

```
NestList[jacobi, x0, 9]
```

```
{{0., 0., 0.}, {0.7, 0.6, 0.5}, {0.86, 0.82, 0.82},
 {0.946, 0.932, 0.918}, {0.9768, 0.97, 0.9688}, {0.99076, 0.98832, 0.98636},
 {0.996104, 0.994984, 0.994648}, {0.998428, 0.998005, 0.997716},
 {0.999344, 0.999158, 0.999087}, {0.999733, 0.99966, 0.999616}}
```

Točna rešitev:

```
LinearSolve[a, b]
```

```
{1, 1, 1}
```

■ 4.4 Gauss-Seidlova iteracija

Pri Gauss-Seidlovi iteraciji izboljšamo konvergenco približkov tako, da uporabimo že izračunane komponente novega približka. V matrični obliki to pomeni:

$$(s+d)\,\vec{x}^{(n+1)} = \vec{b} - z\,\vec{x}^{(n)},$$

```
a = {{10, -1, -2}, {-1, 10, -3}, {-2, -3, 10}};
b = {7, 6, 5};
r = Length[a];
```

postopek za računanje zaporednih približkov je enak kot prej, le da je n zdaj trikotna matrika:

```
n = Table[If[i >= j, a[[i, j]], 0], {i, 1, r}, {j, 1, r}]
```

```
{{10, 0, 0}, {-1, 10, 0}, {-2, -3, 10}}
```

```
p = n - a
```

```
{{0, 1, 2}, {0, 0, 3}, {0, 0, 0}}
```

```
m = Inverse[n].p
```

$$\left\{\left\{0, \frac{1}{10}, \frac{1}{5}\right\}, \left\{0, \frac{1}{100}, \frac{8}{25}\right\}, \left\{0, \frac{23}{1000}, \frac{17}{125}\right\}\right\}$$

```
Norm[m, Infinity]
```

$$\frac{33}{100}$$

```
c = Inverse[n].b
```

$$\left\{\frac{7}{10}, \frac{67}{100}, \frac{841}{1000}\right\}$$

```
x0 = {0.0, 0.0, 0.0};
```

```
gs[x_] := m.x + c;
```

```
NestList[gs, x0, 9]
```

```
{{0., 0., 0.}, {0.7, 0.67, 0.841}, {0.9352, 0.94582, 0.970786},
 {0.988739, 0.99011, 0.994781}, {0.997967, 0.998231, 0.999063},
 {0.999636, 0.999682, 0.999832}, {0.999935, 0.999943, 0.99997},
 {0.999988, 0.99999, 0.999995}, {0.999998, 0.999998, 0.999999}, {1., 1., 1.}}
```

```
LinearSolve[a, b]
```

```
{1, 1, 1}
```

■ 4.5 Lastne vrednosti in lastni vektorji

Lastne vrednosti matrike A so števila λ, za katera obstajajo rešitve homogenega sistema:

$$A\,\vec{x}=\lambda\,\vec{x},$$

ki je različna od nič. Rešitve so lastni vektorji.

Funkcija `Eigensystem[]` vrne lastne vrednosti in lastne vektorje, samo lastne vrednosti dobimo z uporabo `Eigenvalues[]`, samo lastne vektorje pa z `Eigenvectors[]`.

Poiščimo lastne vrednosti in lastne vektorje matrike a:

```
a = {{4.2, 0.4, 0.2}, {0.4, 5.6, 0.8}, {0.2, 0.8, 6.4}};
```

```
Eigenvalues[a] // TableForm
```

```
6.94768
5.15862
4.0937
```

```
Eigenvectors[a] // TableForm
```

```
0.138608      0.535622      0.833005
-0.219849     -0.803495     0.553229
0.965637      -0.259817     0.00638518
```

```
Eigensystem[a] // TableForm
```

```
6.94768      5.15862       4.0937
0.138608     -0.219849     0.965637
0.535622     -0.803495     -0.259817
0.833005     0.553229      0.00638518
```

Lastne vrednosti simetrične matrike računamo po Jacobijevi metodi z nizom rotacij matrike A, ki jo prevedejo v diagonalno obliko $U_m{}^T \ldots U_1{}^T A U_1 \ldots U_m$. Sestavimo funkcijo jac[mat, stevec], ki na dani matriki mat opravi števec rotacij. Kot rezultat izpiše v vsakem koraku približke lastnih vrednosti in sestavi matriko lastnih vektorjev lvek:

```
jac[mat_, stevec_] := Module[{a = mat, st = stevec}, lvek = IdentityMatrix[3];
  (* V Module sta spremenljivki a in st lokalni. *)
  For[i = 1, i <= st,
   u = IdentityMatrix[3];
   (* Definiramo rotacijsko matriko in določimo
    največji izvediagonalni element, ki določi rotacijo. *)
   el = Position[a, Max[Abs[a - DiagonalMatrix[Tr[a, List]]]]][[1]];
   (* Vrstica in stolpec elementa. *)
   k = el[[1]]; l = el[[2]];
   (* Kot rotacije. *)
   phi = ArcTan[-2 a[[k, l]] / (a[[l, l]] - a[[k, k]])] / 2;
   (* Elementi rotacijske matrike. *)
   u[[k, k]] = Cos[phi]; u[[k, l]] = -Sin[phi];
   u[[l, l]] = Cos[phi]; u[[l, k]] = -u[[k, l]];
   (* Nov približek k diagonalni matriki. *)
   a = Transpose[u].a.u;
   (* Še nov približek lastnim vektorjem. *)
   lvek = lvek.u;
   Print[TableForm[Tr[a, List]]];
   i++]]
```

V štirih korakih dobimo:

```
jac[a, 4]
```

```
4.2
5.10557
6.89443

4.14732
5.10557
6.94711

4.09372
5.15918
6.94711

4.09372
5.15862
6.94766
```

```
lvek // TableForm
```

```
0.965309      0.219733     0.141055
-0.261078     0.803526     0.534962
0.00420807    -0.55323     0.833018
```

Pri posebnih oblikah matrik je koristno uporabiti izbrane metode, ki hitreje konvergirajo. Za iskanje lastnih vrednosti tridiagonalne matrike uporabimo QR-metodo, kjer tridiagonalno matriko z vrsto zaporednih QR- dekompozicij prevedemo v diagonalno. *Mathematica* pozna funkcijo QRDecomposition[], ki matriko a razcepi v produkt ortogonalne in zgornje trikotne.

```
a = N[TridiagonalMatrix[Plus, 5]];(* tvorimo tridiagonalno matriko *)
stkor = 500; (* število zaporednih razcepov *)
For[i = 0, i < stkor, {q, r} = QRDecomposition[a]; a = Chop[r.Transpose[q]]; i++];
```

Po 500 korakih dobimo matriko a v diagonalni obliki z lastnimi vrednostmi originalne matrike na diagonali.

```
MatrixForm[a]
```

$$\begin{pmatrix} 20.0258 & 0 & 0 & 0 & 0 \\ 0 & 10.4179 & 0 & 0 & 0 \\ 0 & 0 & -4.53355 & 0 & 0 \\ 0 & 0 & 0 & 4.2264 & 0 \\ 0 & 0 & 0 & 0 & -0.136588 \end{pmatrix}$$

```
Eigenvalues[N[TridiagonalMatrix[Plus, 5]]] // TableForm
```

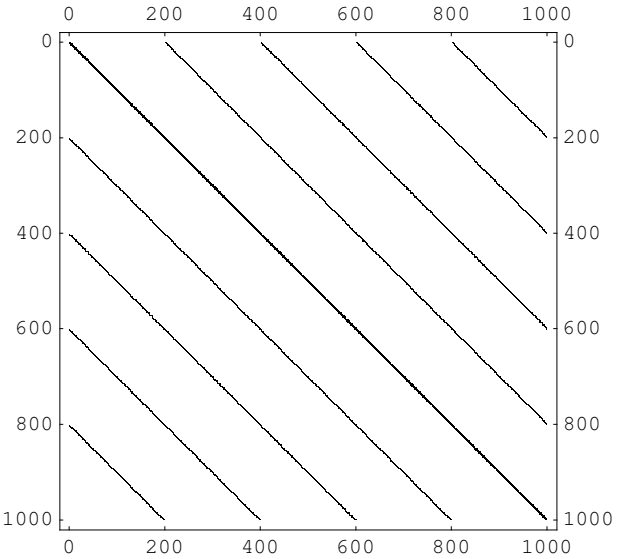
```
20.0258
10.4179
-4.53355
4.2264
-0.136588
```

■ 4.6 Reševanje velikih linearnih sistemov

Metode numeričnega modeliranja, ki temeljijo na linearizaciji in diskretizaciji enačb (npr. metoda končnih elementov), praviloma vodijo do reševanja velikih sistemov lineranih enačb. Matrike koeficientov enačb so velikokrat redke (vsebujejo relativno malo od nič različnih elementov). V takih primerih lahko v *Mathematici* definiramo redke matrike s `SparseArray` in uporabimo napredne numerične metode za rešitev sistema linearnih enačb.

```
n = 1000;(* Velikost matrike n*n. *)
pravila = {{i_, i_} :> Random[], {i_, j_} /; Mod[Abs[i - j], n / 5] == 1 → 1.}; (* Določimo
 pravila za tvorbo redke matrike: na diagonalo postavimo naključne vrednosti,
zunaj diagonale pa po trakovih vrednosti 1. *)
sp = SparseArray[pravila, {n, n}];
```

```
MatrixPlot[sp];(* Grafično predstavimo strukturo matrike. *)
```



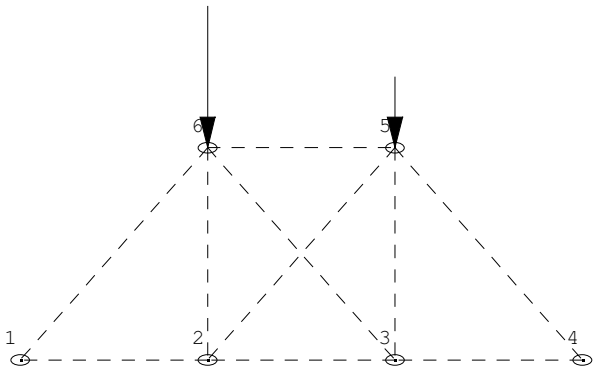

```
b = Table[Random[], {n}];
(* Tudi za vektor desne strani enačb izberemo naključna števila. *)
```

```
x = LinearSolve[ sp, b, Method → Multifrontal]; // Timing
```

```
{0.01 Second, Null}
```

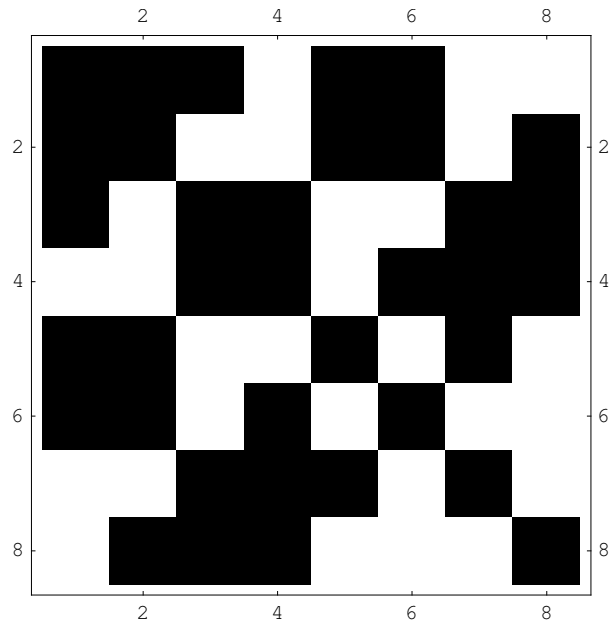
4.7 Primer: ravninsko paličje

```
<< Graphics`Arrow`

m = 10;  (* Število palic. *)
d = 2;  (* Dimenzija problema. *)
n = 6;(* Število vozlišč. *)
p = 2; (* Število vpetih vozlišč. *)
ea = 210. × 10^9 × 5 × 10^-4 10^-3; (* Modul elastičnosti x prerez,
upoštevamo se, da so sile v kN! *)


z = {{0, 0}, {2, 0}, {4, 0}, {6, 0}, {4, 2}, {2, 2}};(* Koordinate vozlišč. *)


f = {{0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, -10}, {0, -20}}; (* Sile v vozliščih. *)


id = IdentityMatrix[n d];

(* Seznam začetnih in končnih točk palic. *)
palice = {{2, 1}, {3, 2}, {4, 3}, {5, 4}, {6, 5}, {6, 2}, {5, 2}, {5, 3}, {6, 3}, {6, 1}};

(* Narišemo začetno paličje s silami, ki delujejo. *)
risisile = Table[Graphics[Arrow[z[[i]] - f[[i]] / 15, z[[i]]]], {i, 1, Length[z]}];


crte = Table[
   Graphics[{Dashing[{0.02, 0.02}], Line[{z[[palice[[i, 1]]]], z[[palice[[i, 2]]]]}]}],
   {i, 1, Length[palice]}];
vozlisca = Table[Graphics[Circle[z[[i]], {.1, .05}]], {i, 1, Length[z]}];
oznake = Table[Graphics[Text[i, z[[i]] + {-0.1, 0.2}]], {i, 1, Length[z]}];


nodef = Show[crte, vozlisca, oznake, risisile];
```




```
(* Sestavimo incidenčno matriko paličja *)
a = Table[(Flatten[z].(id[[2 palice[[stpalice, 1]] - 1]] - id[[2 palice[[stpalice, 2]] - 1]])
       (id[[2 palice[[stpalice, 1]] - 1]] - id[[2 palice[[stpalice, 2]] - 1]]) +
      Flatten[z].(id[[2 palice[[stpalice, 2]]]] - id[[2 palice[[stpalice, 1]]]])
       (id[[2 palice[[stpalice, 2]]]] - id[[2 palice[[stpalice, 1]]]])) /
    Sqrt[(z[[palice[[stpalice, 1]]]] - z[[palice[[stpalice, 2]]]]).
      (z[[palice[[stpalice, 1]]]] - z[[palice[[stpalice, 2]]]])],
   {stpalice, 1, Length[palice]}];
```

```
(* in še diagonalno matriko elastičnih konstant. *)
dd =
  DiagonalMatrix[Table[ea / Sqrt[(z[[palice[[stpalice, 1]]]] - z[[palice[[stpalice, 2]]]]).
       (z[[palice[[stpalice, 1]]]] - z[[palice[[stpalice, 2]]]])],
    {stpalice, 1, Length[palice]}]];

(* Določimo podpore,
za vsako komponento vozlišča posebej: pritrjeni sta prvo in četrto vozlišče. *)
podpore = {{1}, {2}, {7}, {8}};

(* Sestavimo reduktor, ki bo izbral samo nepritrjene komponente. *)
r = Delete[IdentityMatrix[n d], podpore];


f0 = r.Flatten[f];


a0 = a.Transpose[r];

(* redicurana togostna matrika *)
k0 = Transpose[a0].dd.a0;


k0 // MatrixPlot (* Grafično prikažemo strukturo reducirane togostne matrike. *)
```




```
- DensityGraphics -

(* Izračunamo pomike nepritrjenih
 vozlišč: zaporedoma si sledita komponenti x in y po vrstnem redu vozlišč. *)
ColumnForm[x0 = Inverse[k0].f0]
```

```
0.0000215312
-0.00101558
0.0000419609
-0.000917023
-0.000136301
-0.000854632
0.000118768
-0.00101668
```

```
(* Rekonstruiramo celotni vektor pomikov. *)
x = Transpose[r].x0;
```

```
(* Izpišemo pomike po vozliščih. *)
izpis_pomikov = TableForm[Table[{x[[2 i - 1]], x[[2 i]]}, {i, 1, Length[z]}],
  TableHeadings → {Automatic, {"x", "y"}}]
```

	x	y
1	0.	0.
2	0.0000215312	-0.00101558
3	0.0000419609	-0.000917023
4	0.	0.
5	-0.000136301	-0.000854632
6	0.000118768	-0.00101668

```
(* Togostna matrika je lahko slabo pogojena, zato raje uporabimo QR-razcep. *)
sqdd = Sqrt[dd];


{q, u} = QRDecomposition[sqdd.a0];

(* Rešitve so seveda enake kot prej. *)
x = Transpose[r].Inverse[Transpose[u].u].f0
```

```
{0., 0., 0.0000215312, -0.00101558, 0.0000419609, -0.000917023,
 0., 0., -0.000136301, -0.000854632, 0.000118768, -0.00101668}
```

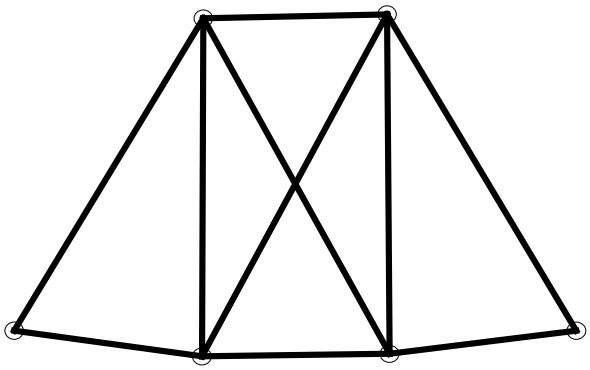
```
(* Definiramo nove koordinate vozlišč primerne za grafično predstavitev. *)
zdef = Flatten[z] + 150 x
```

```
{0., 0., 2.00323, -0.152337, 4.00629, -0.137553, 6., 0., 3.97955, 1.87181, 2.01782, 1.8475}
```

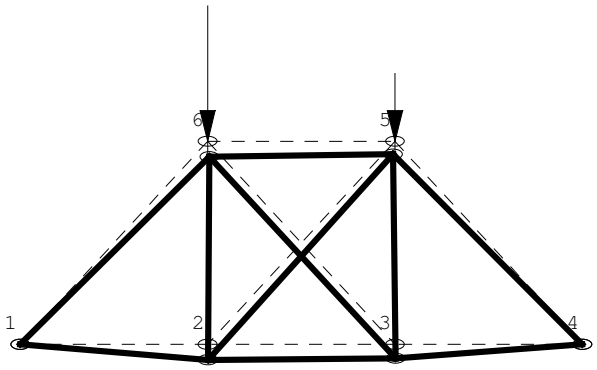
```
znovi = Table[{zdef[[i]], zdef[[i + 1]]}, {i, 1, Length[zdef], 2}]
```

```
{{0., 0.}, {2.00323, -0.152337}, {4.00629, -0.137553},
 {6., 0.}, {3.97955, 1.87181}, {2.01782, 1.8475}}
```

```
(* Narišemo premaknjeno paličje. *)
crte = Table[
   Graphics[{Thickness[0.01], Line[{znovi[[palice[[i, 1]]]], znovi[[palice[[i, 2]]]]}]},
   {i, 1, Length[palice]}];
vozlisca = Table[Graphics[Circle[znovi[[i]], {.1, .05}]], {i, 1, Length[znovi]}];
oznake = Table[Graphics[Text[i, znovi[[i]] + {-0.2, 0.2}]], {i, 1, Length[znovi]}];


def = Show[crte, vozlisca];
```



```
(* Na eni sliki prikažemo premik paličja zaradi delovanja zunanjih sil. *)
Show[nodef, def];
```




```
(* Izračunamo še sile v vseh vozliščih. *)
sile = dd.(a.x)
```

```
{1.13039, 1.07256, -2.20294, -18.8562,
 -13.3912, -0.0578324, 0.0817874, 3.2755, -4.63226, -23.5702}
```

```
(* Sestavimo seznam samo nepritrjenih vozlišč. *)
prosti = Complement[Table[{i}, {i, 1, n d}], podpore]
```

```
{{3}, {4}, {5}, {6}, {9}, {10}, {11}, {12}}
```

```
(* Reduktor, ki izbere samo nepritrjene komponente. *)
s = Delete[IdentityMatrix[n d], prosti]
```

```
{{1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, {0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
 {0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0}}
```

```
(* Izračunamo še sile v podporah. *)
k = Transpose[sqdd.a].(sqdd.a);
s.k.x
```

```
{15.5363, 16.6667, -15.5363, 13.3333}
```

5 Interpolacija, aproksimacija in numerična integracija funkcij

■ 5.1 Polinomska interpolacija

Funkcija g(x) interpolira funkcijo f(x), običajno podano z vrednostmi v diskretnih točkah x_k k=0,...,n, če velja:

$$f(x_k)=g(x_k).$$

Funkcija f(x) naj bo znana v točkah, danih s tabelo:

```
y = {{0, -0.5}, {0.1, 0}, {0.2, 0.2}, {0.3, 1}};
```

Konstruirajmo najprej Lagrangeev polinom, ki interpolira funkcijo f:

```
intpol[x_] = Sum[y[[k, 2]]
   Product[If[j == k, 1, (x - y[[j, 1]]) / (y[[k, 1]] - y[[j, 1]])], {j, 1, Length[y]}],
  {k, 1, Length[y]}]
```

```
83.3333 (-0.3 + x) (-0.2 + x) (-0.1 + x) -
 100. (-0.3 + x) (-0.1 + x) x + 166.667 (-0.2 + x) (-0.1 + x) x
```

Preglednejši izpis Lagrangeveega interpolacijskega polinoma dobimo z ukazom `Expand` (% pomeni vsebino zadnje celice `Out[]`):

```
Expand[%]
```

```
-0.5 + 9.5 x - 60. x^2 + 150. x^3
```

Interpolacijski polinom dobimo tudi z vgrajeno funkcijo `InterpolatingPolynomial[]`:

```
Expand[InterpolatingPolynomial[y, x]]
```

```
-0.5 + 9.5 x - 60. x^2 + 150. x^3
```

Pri konstrukciji Newtonovega interpolacijskega polinoma si pomagamo s tabelo razlik. Naj bo funkcija f dana z naslednjo tabelo:

```
y = {{x0, y0}, {x1, y1}, {x2, y2}, {x3, y3}};
```

```
n = Length[y]; (* dolžina tabele oziroma število točk v katerih poznamo funkcijo *)
```

Deljene diference računamo rekurzivno:

```
diferenca[j_] := diferenca[j] = Table[(diferenca[j - 1][[k + 1]] - diferenca[j - 1][[k]]) /
      (y[[k + j, 1]] - y[[k, 1]]), {k, 1, n - j}];
```

Podati moramo še začetno vrednost (prvo diferenco), da lahko rekurzivno izračunamo druge:

```
diferenca[1] = Table[(y[[k + 1, 2]] - y[[k, 2]]) / (y[[k + 1, 1]] - y[[k, 1]]), {k, 1, n - 1}];
```

Druga diferenca $\Delta^2 f$ je:

```
diferenca[2]
```

$$\left\{\frac{-\frac{-y0+y1}{-x0+x1}+\frac{-y1+y2}{-x1+x2}}{-x0+x2},\ \frac{-\frac{-y1+y2}{-x1+x2}+\frac{-y2+y3}{-x2+x3}}{-x1+x3}\right\}$$

Zdaj sestavimo še Newtonov polinom:

```
l[x_] := y[[1, 2]] + Sum[Product[(x - y[[j, 1]]), {j, 1, i}] diferenca[i][[1]], {i, n - 1}]
```

Primer: izračunajmo f(1.6) funkcije, ki je dana s tabelo:

```
y = {{1.1275, 0.11971}, {1.1503, 0.13957}, {1.1735, 0.15931}, {1.1972, 0.17902}};
```

```
Clear[diferenca]
```

```
n = Length[y];
diferenca[j_] := diferenca[j] = Table[(diferenca[j - 1][[k + 1]] - diferenca[j - 1][[k]]) /
     (y[[k + j, 1]] - y[[k, 1]]), {k, 1, n - j}];
diferenca[1] = Table[(y[[k + 1, 2]] - y[[k, 2]]) / (y[[k + 1, 1]] - y[[k, 1]]), {k, 1, n - 1}];
l[x_] := y[[1, 2]] + Sum[Product[(x - y[[j, 1]]), {j, 1, i}] diferenca[i][[1]], {i, n - 1}]

l[1.16]
```

```
0.147879
```

Podobno ravnamo pri konstruiranju Newtonovih polinomov, kadar je funkcija dana v ekvidistančnih točkah.

```
Clear[diferenca]

y = {y0, y1, y2, y3};

n = Length[y];


diferenca[j_] :=
  diferenca[j] = Table[diferenca[j - 1][[k + 1]] - diferenca[j - 1][[k]], {k, 1, n - j}];


diferenca[1] = Table[y[[k + 1]] - y[[k]], {k, 1, n - 1}];
```

Tretja diferenca je, denimo, naslednja:

```
diferenca[3]
```

```
{-y0 + 3 y1 - 3 y2 + y3}
```

Sestavimo Newtonova polinoma za interpolacijo naprej in nazaj:

```
p1 = y[[1]] + Sum[Product[q - k, {k, 0, j - 1}] diferenca[j][[1]] / j!, {j, 1, n - 1}]
```

$$y0 + q\,(-y0 + y1) + \frac{1}{2}\,(-1 + q)\,q\,(y0 - 2\,y1 + y2) + \frac{1}{6}\,(-2 + q)\,(-1 + q)\,q\,(-y0 + 3\,y1 - 3\,y2 + y3)$$

```
p2 = y[[n]] + Sum[Product[q + k, {k, 0, j - 1}] diferenca[j][[n - j]] / j!, {j, 1, n - 1}]
```

$$y3 + \frac{1}{6}\,q\,(1 + q)\,(2 + q)\,(-y0 + 3\,y1 - 3\,y2 + y3) + \frac{1}{2}\,q\,(1 + q)\,(y1 - 2\,y2 + y3) + q\,(-y2 + y3)$$

Primer: Sestavimo tabelo vrednosti integrala $\sqrt{2/\pi}\int_0^t e^{-x^2/2}\,dx$ na intervalu [1, 1.25] s korakom 0.05 in izračunajmo vrednost integrala z Newtonovim interpolacijskim polinomom v točki 1.235:

```
y = Table[Sqrt[2 / Pi] NIntegrate[Exp[-(x^2) / 2], {x, 0, t}], {t, 1, 1.25, 0.05}]
```

```
{0.682689, 0.706282, 0.728668, 0.749856, 0.769861, 0.7887}
```

Najprej izračunajmo tabelo končnih diferenc:

```
n = Length[y];
Clear[diferenca]
diferenca[j_] :=
  diferenca[j] = Table[diferenca[j - 1][[k + 1]] - diferenca[j - 1][[k]], {k, 1, n - j}];
diferenca[1] = Table[y[[k + 1]] - y[[k]], {k, 1, n - 1}];
```

Tabelo pregledno izpišemo. Diference uvrstimo v stolpce, red diference narašča od leve proti desni:

```
stolpec[i_] := Join[diferenca[i], Table[" ", {k, 1, i}]];

TableForm[{y, stolpec[1], stolpec[2], stolpec[3]},
 TableDirections → {Row, Column}, TableHeadings → {{"f", "Δf", "Δ²f", "Δ³f"}, None}]
```

f	Δf	Δ²f	Δ³f
0.682689	0.0235924	-0.0012064	8.66401×10^{-6}
0.706282	0.022386	-0.00119774	0.0000140205
0.728668	0.0211883	-0.00118372	0.0000189821
0.749856	0.0200045	-0.00116474	
0.769861	0.0188398		
0.7887			

Zdaj izračunajmo vrednost funkcije v točki 1.235. Ker je ta točka blizu desnega krajišča intervala, na katerem poznamo vrednosti funkcije, uporabimo Newtonov interpolacijski polinom za interpolacijo nazaj:

```
xi = 1.235; x0 = 1.25; h = 0.05;
q = (xi - x0) / h;
p2 = y[[n]] + Sum[Product[q + k, {k, 0, j - 1}] diferenca[j][[n - j]] / j!, {j, 1, n - 1}]
```

```
0.783169
```

```
Sqrt[2 / Pi] NIntegrate[Exp[-(u^2) / 2], {u, 0, 1.235}]
```

```
0.783169
```

Interpolacijske polinome lahko uporabimo tudi pri numeričnem odvajanju. Tabelirano funkcijo interpoliramo s polinomom in nato izračunamo odvod interpolacijskega polinoma.

Vrednosti funkcije f so dane v ekvidistančnih točkah:

```
y = {y0, y1, y2};
```

Izračunamo intepolacijski polinom:

```
intpol[x_] = Sum[y[[k]] Product[If[j == k, 1, (x - j) / (k - j)], {j, 1, Length[y]}],
  {k, 1, Length[y]}]
```

$$\frac{1}{2}(2-x)(3-x)\,y0 + (3-x)(-1+x)\,y1 + \frac{1}{2}(-2+x)(-1+x)\,y2$$

Približki za odvode funkcije, dane z zgornjo tabelo so:

```
TableForm[Table[Simplify[Expand[D[intpol[x], x]] /. x → j], {j, 1, Length[y]}]]
```

$\frac{1}{2}(-3\,y0 + 4\,y1 - y2)$

$\frac{1}{2}(-y0 + y2)$

$\frac{1}{2}(y0 - 4\,y1 + 3\,y2)$

■ 5.2 Intepolacija s kubičnimi zlepki

S kubičnimi zlepki oblike:

$$S_k(x)=\sum_{j=0}^{3} s_{k,j}(x-x_k)^j$$

odsekoma interpoliramo funkcijo na intervalu $[x_k, x_{k+1}]$. Koeficienti $s_{k,j}$ so:

$$s_{k,0}=y_k,\ s_{k,1}=d_k-h_k(2m_k+m_{k+1})/6,\ s_{k,2}=m_k/2,\ s_{k,3}=(m_{k+1}-m_k)/6h_k,$$

kjer je $d_k=(y_{k+1}-y_k)/h_k$ in $h_k=x_{k+1}-x_k$. Momenti m_k so rešitve tridiagonalnega sistema enačb:

$$h_{k-1}m_{k-1}+2(h_{k-1}+h_k)m_k+h_km_{k+1}=u_k,\ u_k=6(d_k-d_{k-1})$$

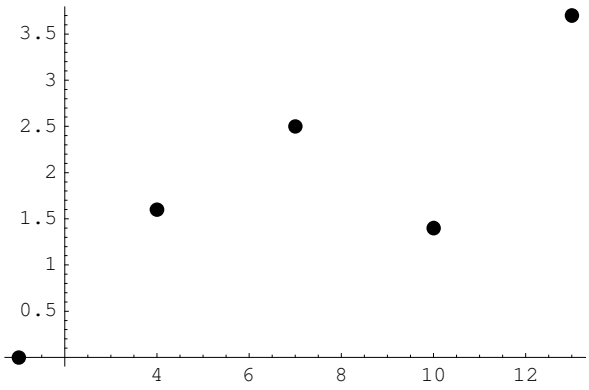
ob ustreznih robnih pogojih na krajiščih (znana prva odvoda v krajiščih, druga odvoda v krajiščih sta enaka nič (= naravni zlepek), druga odvoda na robu sta ekstrapolirana, druga odvoda sta konstantna in druga odvoda na robu sta znana).

S kubičnimi polinomi interpolirajmo funkcijo, dano s tabelo:

```
y = {{1, 0}, {4, 1.6}, {7, 2.5}, {10, 1.4}, {13, 3.7}};
```

Narišimo točke:

```
points = ListPlot[y, PlotStyle -> PointSize[0.025]];
```



Poiščimo najprej kubične zlepke ob predpisanih prvih odvodih s1, s2 na robu:

```
h = 3.;
s1 = .001;
s2 = .001;

(* Sestavimo tabelo razlik. *)
d = Table[(y[[i + 1, 2]] - y[[i, 2]]) / h, {i, 1, Length[y] - 1}]

{0.533333, 0.3, -0.366667, 0.766667}

(* Desne strani tridiagonalnega sistema. *)
u = Table[6 (d[[i + 1]] - d[[i]]), {i, 1, Length[y] - 2}]

{-1.4, -4., 6.8}

(* Linearni sistem skupaj z robnima pogojema. *)
a = {{1.5 h + 2 h, h, 0}, {h, 2 (h + h), h}, {0, h, 2 h + 1.5 h}};
b = {u[[1]] - 3 (d[[1]] - s1), u[[2]], u[[3]] - 3 (s1 - d[[4]])};
```

```
(* Momenti notranjih točk intervala. *)
m = LinearSolve[a, b]
```

```
{-0.125905, -0.558333, 1.0259}
```

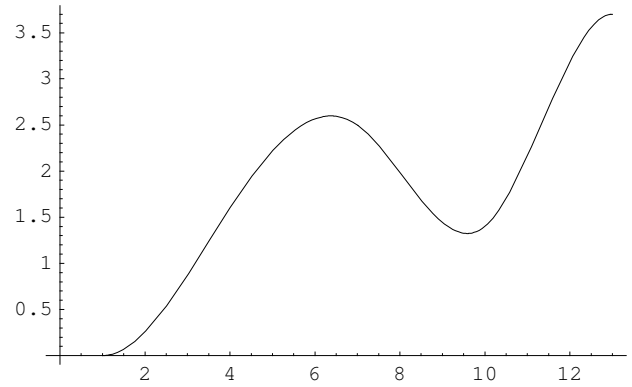
```
(* dodamo še m0 *)
m = Prepend[m, 3 (d[[1]] - s1) / h - m[[1]] / 2]
```

```
{0.595286, -0.125905, -0.558333, 1.0259}
```

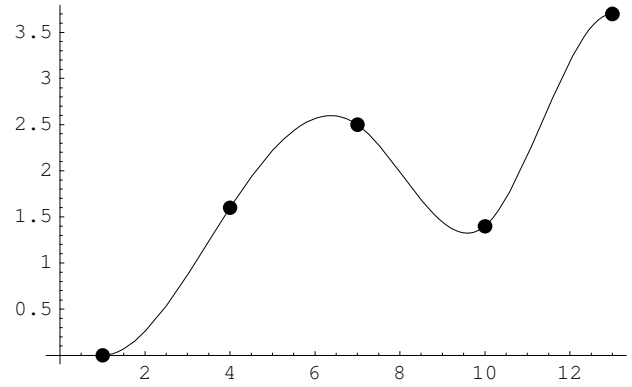
```
(* in še m4 *)
m = Append[m, 3 (s2 - d[[4]]) / h - m[[4]] / 2]
```

```
{0.595286, -0.125905, -0.558333, 1.0259, -1.27862}
```

```
(* k-ti kubični interpolacijski polinom. *)
s[k_, x_] := (y[[k, 2]] +
      (d[[k]] - h (2 m[[k]] + m[[k + 1]]) / 6) (x - y[[k, 1]]) + m[[k]] (x - y[[k, 1]])^2 / 2 +
      (m[[k + 1]] - m[[k]]) (x - y[[k, 1]])^3 / 6 / h)
   (-UnitStep[x - y[[k + 1, 1]]] + (UnitStep[x - y[[k, 1]]]));


(* Narišemo celotno interpolacijsko funkcijo - zlepek. *)
splot1 = Plot[s[1, x] + s[2, x] + s[3, x] + s[4, x], {x, 1, 13}];
```




```
(* Točke tabele in interpolacija s kubičnimi zlepki. *)
Show[points, splot1];
```



Poiščimo še naravni zlepek, ki ima druga odvoda na robu nič. Spremeni se tridiagonalni sistem:

```
a = {{2 h + 2 h, h, 0}, {h, 2 (h + h), h}, {0, h, 2 h + 2 h}};
b = {u[[1]], u[[2]], u[[3]]};
```

```
m = LinearSolve[a, b]
```

```
{0.0107143, -0.509524, 0.694048}
```

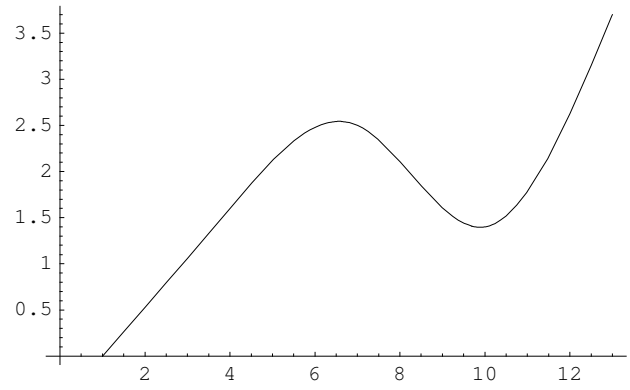
```
(* Dodamo m0 *)
m = Prepend[m, 0.]
```

```
{0., 0.0107143, -0.509524, 0.694048}
```

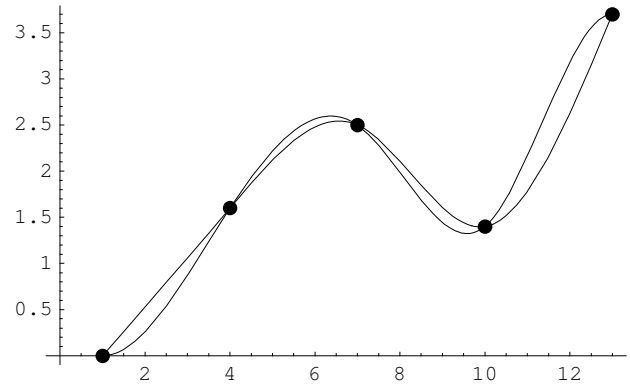
```
(* in m4. *)
m = Append[m, 0.]
```

```
{0., 0.0107143, -0.509524, 0.694048, 0.}
```

```
(* Narišemo novo interpolacijsko funkcijo. *)
splot2 = Plot[s[1, x] + s[2, x] + s[3, x] + s[4, x], {x, 1, 13}];
```




```
(* Narišemo točke in obe interpolacijski funkciji. *)
Show[points, splot2, splot1];
```



Mathematica pozna funkcijo `SplineFit[],` ki za dano tabelo tvori interpolacijsko funkcijo s kubičnimi zlepki, pri čemer sta druga odvoda na robu nič - naravni spline. Na voljo je v paketu `NumericalMath`, ki ga najprej naložimo:

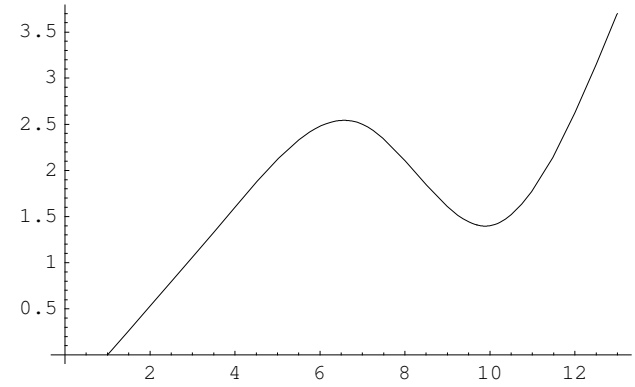
```
<< NumericalMath`SplineFit`
```

```
zlepki = SplineFit[y, Cubic]
```

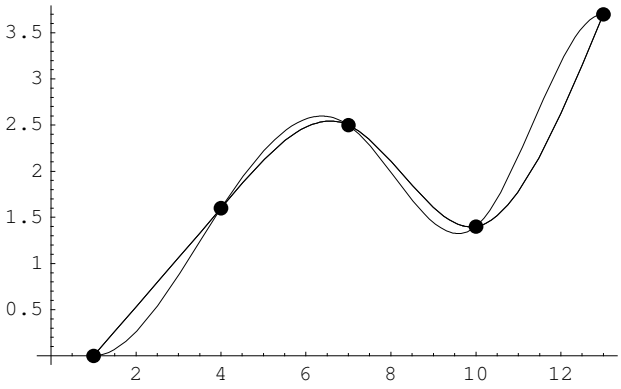
```
SplineFunction[Cubic, {0., 4.}, <>]
```

Rezultat je objekt `SplineFunction,` definiran na intervalu [0, 4]. Interpolacijsko funkcijo lahko narišemo:

```
splot3 = ParametricPlot[zlepki[x], {x, 0, 4}];
```




```
Show[points, splot2, splot1, splot3];
```



Ogledamo si lahko tudi strukturo interpolacijske funkcije, iz katere lahko preberemo koeficiente posameznih kubičnih polinomov v zlepku:

```
InputForm[zlepki]
```

```
SplineFunction[Cubic, {0., 4.}, {{1, 0}, {4, 1.6}, {7, 2.5}, {10, 1.4}, {13, 3.7}},
 {{{1, 3, 0, 0}, {0, 1.5839285714285718, 2.220446049250313*^-16,
    0.016071428571428514}}, {{4, 3, 0, 0}, {1.6, 1.632142857142857,
    0.04821428571428599, -0.780357142857143}},
  {{7, 3, 0, 0}, {2.5, -0.6125, -2.292857142857143, 1.8053571428571429}},
  {{10, 3, 0, 0}, {1.4, 0.21785714285714286, 3.1232142857142855, -1.041071428571429}}}]
```

Enak rezultat dobimo, če računamo kot prej, le za korak postavimo h = 1:

```
h = 1.;
d = Table[(y[[i + 1, 2]] - y[[i, 2]]), {i, 1, Length[y] - 1}];
u = Table[6 (d[[i + 1]] - d[[i]]), {i, 1, Length[y] - 2}];
a = {{4, 1, 0}, {1, 4, 1}, {0, 1, 4}};
b = {u[[1]], u[[2]], u[[3]]};
m = LinearSolve[a, b];
m = Prepend[m, 0.];
m = Append[m, 0.];
s[k_, x_] :=
  (y[[k, 2]] + (d[[k]] - h (2 m[[k]] + m[[k + 1]]) / 6) (x - k + 1) + m[[k]] (x - k + 1)^2 / 2 +
    (m[[k + 1]] - m[[k]]) (x - k + 1)^3 / 6 / h);
```

```
s[1, x]
```

```
1.58393 x + 0. x^2 + 0.0160714 x^3
```

```
s[2, x]
```

```
1.6 + 1.63214 (-1 + x) + 0.0482143 (-1 + x)^2 - 0.780357 (-1 + x)^3
```

```
s[3, x]
```

```
2.5 - 0.6125 (-2 + x) - 2.29286 (-2 + x)^2 + 1.80536 (-2 + x)^3
```

```
s[4, x]
```

```
1.4 + 0.217857 (-3 + x) + 3.12321 (-3 + x)^2 - 1.04107 (-3 + x)^3
```

■ 5.3 Aproksimacija z metodo najmanjših kvadratov

Aproksimacijo funkcije *f(x)*, ki je dana s tabelo, iščemo z linearno kombinacijo funkcij $q(x)=\sum_{j=0}^{m} c_j\, g_j(x)$. Koeficienti c_j so rešitve linearnega sistema:

$$\sum_{j=0}^{m} c_j\left(g_j,\, g_k\right) = (f,g_k),\ k=0,..,m$$

Skalarni produkt je definiran z vsoto:

$$(f,\, g)=\sum_{k=0}^{n} f\,(x_k)\, \mathrm{g}\,(x_k).$$

Dana je tabela točk:

```
Clear[y]
xy = {{1, 0}, {4, 1.6}, {7, 2.5}, {10, 1.4}, {13, 3.7}};


TableForm[xy, TableDirections → {Row, Column},
 TableHeadings → {None, {x, y}}]
```

```
x  1  4    7    10   13

y  0  1.6  2.5  1.4  3.7
```

Aproksimacijsko funkcijo iščemo kot linearno kombinacijo funkcij x^j.

```
g[x_] := {1, x, x^2, x^3};

(* x in y koordinate iz tabele. *)
xk = Transpose[xy][[1]]
yk = Transpose[xy][[2]]
```

```
{1, 4, 7, 10, 13}
```

```
{0, 1.6, 2.5, 1.4, 3.7}
```

```
(* Matrika koeficientov cj. *)
a = Table[Sum[g[xk[[t]]][[i]] g[xk[[t]]][[j]], {t, 1, Length[xk]}],
  {j, 1, Length[g[x]]}, {i, 1, Length[g[x]]}]
```

```
{{5, 35, 335, 3605}, {35, 335, 3605, 41 219},
 {335, 3605, 41 219, 489 125}, {3605, 41 219, 489 125, 5 948 555}}
```

```
(* Desna stran - skalarni produkti. *)
b = Table[Sum[yk[[t]] g[xk[[t]]][[j]], {t, 1, Length[xk]}], {j, 1, Length[g[x]]}]
```

```
{9.2, 86., 913.4, 10 488.8}
```

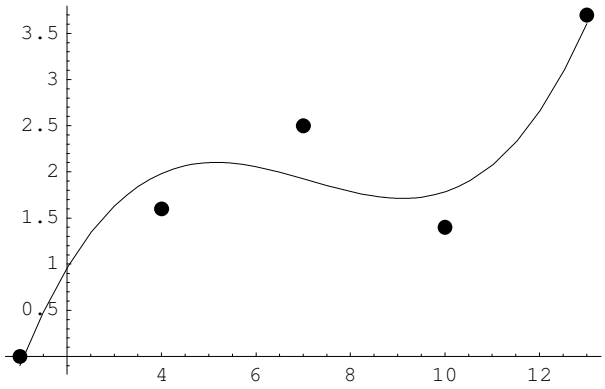
```
(* Koeficienti. *)
c = LinearSolve[a, b]
```

```
{-1.6175, 1.77963, -0.270503, 0.0126543}
```

```
(* Aproksimacijska funkcija. *)
fitfunc[x_] = c.g[x]
```

```
-1.6175 + 1.77963 x - 0.270503 x^2 + 0.0126543 x^3
```

```
(* Narišemo točke in aproksimacijsko funkcijo. *)
Show[ListPlot[xy, PlotStyle → PointSize[0.025], DisplayFunction → Identity],
  Plot[fitfunc[x], {x, 1, 13}, DisplayFunction → Identity],
  PlotRange → All, DisplayFunction → $DisplayFunction];
```



Koeficiente aproksimacijske funkcije lahko poiščemo s `FindFit[]`:

```
FindFit[xy, a0 + a1 x + a2 x^2 + a3 x^3, {a0, a1, a2, a3}, x]
```

```
{a0 → -1.6175, a1 → 1.77963, a2 → -0.270503, a3 → 0.0126543}
```

ali z `Fit[]`:

```
ft[x_] = Fit[xy, {1, x, x^2, x^3}, x]
```

```
-1.6175 + 1.77963 x - 0.270503 x^2 + 0.0126543 x^3
```

■ 5.4 Aproksimacija z ortogonalnimi polinomi

Ortogonalne polinome $p_j(x)$, ki jih konstruiramo nad množico točk tabelirane funkcije, uporabimo za aproksimacijsko funkcijo:

$$g(\mathrm{x})=\sum_{k=0}^{n} \frac{(f,p_k)}{(p_k,p_k)}\, p_k(x).$$

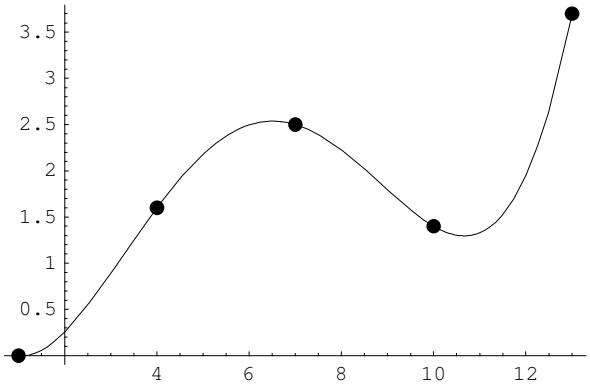
Ortogonalne polinome konstruiramo rekurzivno:

$$p_{k+1}(x)= (\mathrm{x}-\frac{(\mathrm{x}p_k,p_k)}{(p_k,p_k)})p_k(x) - \frac{(p_k,p_k)}{(p_{k-1},p_{k-1})}\, p_{k-1}(x),\ p_0(x) = 1.$$

```
Clear[p];
xy = {{1, 0}, {4, 1.6}, {7, 2.5}, {10, 1.4}, {13, 3.7}};(* Funkcijska tabela. *)
(* x in y vrednosti iz tabele. *)
xk = Transpose[xy][[1]];
yk = Transpose[xy][[2]];
n = Length[xk];
m = 3;

p[-1, x_] = 1; (* Začetni člen rekurzije. *)

(* Računanje polinomov. *)
p[k_, x_] := p[k, x] =
   (x - (Sum[xk[[i]] p[k - 1, xk[[i]]]^2, {i, 1, n}]) / (Sum[p[k - 1, xk[[i]]]^2, {i, 1, n}]))
     p[k - 1, x] - If[k > 0,
     p[k - 2, x] Sum[p[k - 1, xk[[i]]]^2, {i, 1, n}] / Sum[p[k - 2, xk[[i]]]^2, {i, 1, n}], 0];

(* Aproksimacijska funkcija. *)
g[x_] = Sum[Sum[yk[[i]] p[j, xk[[i]]], {i, 1, n}] / Sum[p[j, xk[[i]]]^2, {i, 1, n}] p[j, x],
   {j, -1, m}];

Expand[g[x]]
```

$$0.500823 - 1.02582\,x + 0.605401\,x^2 - 0.0838477\,x^3 + 0.0034465\,x^4$$

```
(* Narišemo točke in aproksimacijo z ortogonalnimi polinomi. *)
Show[ListPlot[xy, PlotStyle → PointSize[0.025], DisplayFunction → Identity],
  Plot[g[x], {x, 1, 13}, DisplayFunction → Identity],
  PlotRange → All, DisplayFunction → $DisplayFunction];
```



Mathematica pozna funkcijo `PolynomialFit[]`, ki kostruira objekt `FittingPolynomial`. Potrebujemo paket `NumericalMath`PolynomialFit``:

```
<< NumericalMath`PolynomialFit`

pf = PolynomialFit[xy, 4]
```

```
FittingPolynomial[<>, 4]
```

```
Expand[pf[x]]
```

$$0.500823 - 1.02582\,x + 0.605401\,x^2 - 0.0838477\,x^3 + 0.0034465\,x^4$$

■ 5.5 Numerično integriranje

Integrala funkcije pogosto ne moremo izraziti analitično s kombinacijo elementarnih funkcij, zato pri numeričnem računanju integral nadomestimo z vsoto primerno uteženih vrednosti funkcije v točkah na integracijskem območju. Kadar so točke znane vnaprej (po navadi ekvidistančno razdelimo interval integracije), lahko na podlagi izbire polinomske baze x^j j=1,..,n konstruiramo integracijske formule, ki so natančne za polinome stopnje j ≤ n (trapezna, Simpsonova formula ...). Pri integralskih formulah (kvadraturah) Gaussovega tipa točke, s katerimi računamo približek integralu, niso znane vnaprej in jih izračunamo v okviru konstrukcije formule.

Funkcijo

```
f[x_] := 2 Exp[-x^2] / Sqrt[Pi];
```

integrirajmo najprej z vgrajenim ukazom `Integrate[]`, s katerim iščemo analitični izraz za nedoločen integral, če ne podamo mej:

```
anint[x_] = Integrate[f[x], x]
```

```
Erf[x]
```

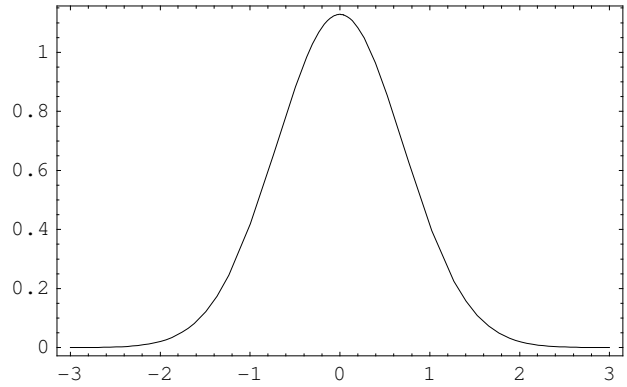
Določen integral izračunamo, če navedemo spodnjo in zgornjo mejo integrala:

```
Integrate[f[x], {x, 0, Infinity}]
```

```
1
```

Sestavimo kratek program za računanje določenega integrala s Simpsonovo formulo. Računamo integral zgornje funkcije na intervalu [-1,1], katere graf je:

```
Plot[{f[x]}, {x, -3, 3}, Frame → True, Axes → False];
```




```
(* Program, ki za vhodne parametre: spmeja (spodnja meja integrala),
zgmeja (zgornja meja integrala) in n(interval razdelimo na 2n podintervalov.)
 izračuna integral funkcije f po Simpsonovi formuli *)

simpint[spmeja_, zgmeja_, n_] := Module[{a, b, m, h}, a = spmeja; b = zgmeja;
   m = n;
   h = (b - a) / 2 / m;
   simpson =
    (h / 3.) (f[a] + f[b] + 4 Sum[f[a + (2 k - 1) h], {k, 1, m}] + 2 Sum[f[a + 2 k h], {k, 1, m - 1}])];
```

Za n = 10 dobimo:

```
PaddedForm[simpint[-1, 1, 5], 7]
```

```
 1.685431
```

Sestavimo kratek program še za računanje določenega integrala s trapezno formulo:

```
(* Podobno kot zgoraj podamo spodnjo in zgornjo mejo interval ter število delitev. *)

trapint[spmeja_, zgmeja_, n_] := (zgmeja - spmeja)
  (0.5 (f[spmeja] + f[zgmeja]) + Sum[f[spmeja + i (zgmeja - spmeja) / n], {i, 1, n - 1}]) / n


PaddedForm[trapint[-1, 1, 20], 7]
```

```
 1.684017
```

Uporabimo lahko tudi funkcijo `NIntegrate[]`, ki numerično izračuna integral dane funkcije:

```
PaddedForm[NIntegrate[f[x], {x, -1, 1}], 7]
```

```
 1.685402
```

Z `GaussianQuadrature[]` poiščemo točke, s katerimi računamo približek integralu in ustrezne uteži:

```
<< NumericalMath`GaussianQuadrature`
```

Podamo število točk in meje intervala:

```
gq = GaussianQuadratureWeights [3, -1, 1]
```

```
{{-0.774597, 0.555556}, {0., 0.888889}, {0.774597, 0.555556}}
```

Približek je:

```
PaddedForm[Transpose[gq][[2]].f[Transpose[gq][[1]]], 7]
```

```
 1.691079
```

Točke, ki jih uporabimo za računanje integralov v formulah Gaussovega tipa, so ničle ortogonalnega polinoma, ki ga konstruiramo podobno, kot smo naredili pri aproksimaciji funkcij, le da je skalarni produkt zdaj definiran tako:

$$(p_j,\ p_k) = \int_a^b w(x)\, p_j(x)\, p_k(x)\, dx,$$

kjer je w(x) utežna funkcija. Numerični približek integralu funkcije f(x) na intervalu [a,b] je:

$$s = \frac{b-a}{2} \sum_{k=1}^{n} c_k\, f(x_k),$$

kjer so x_k ničle $p_n(x)$. Koeficienti c_k so rešitve linearnega sistema:

$$\frac{b-a}{2} \sum_{k=1}^{n} c_k\, {x_k}^j = \int_a^b w(x)\, x^j\, dx.$$

Na intervalu[-1,1] tako dobimo z utežno funkcijo w(x) = 1:

```
Clear[p];
w[x_] = 1;
a = -1;
b = 1;

p[-1, x_] = 1; (* Začetek rekurzije. *)
```

```
(* Rekurzivno računanje polinomov. *)
p[k_, x_] := p[k, x] = (x - (Integrate[t w[t] p[k - 1, t]^2, {t, a, b}]) /
        (Integrate[w[t] p[k - 1, t]^2, {t, a, b}])) p[k - 1, x] - If[k > 0, p[k - 2, x]
      Integrate[w[t] p[k - 1, t]^2, {t, a, b}] / Integrate[w[t] p[k - 2, t]^2, {t, a, b}], 0];
```

Za računanje v treh točkah potrebujemo ničle polinoma $p_2(x)$:

```
Expand[p[2, x]]
```

$$-\frac{3\,x}{5}+x^3$$

```
(* Ničle polinoma sortirane po vrsti. *)
xk = x /. Sort[N[Solve[p[2, x] == 0, x]]]
```

```
{-0.774597, 0., 0.774597}
```

```
(* Matrika koeficientov linearnega sistema. *)
m = (b - a) / 2 Table[xk^j, {j, 1, 2}]
```

```
{{-0.774597, 0., 0.774597}, {0.6, 0., 0.6}}
```

```
m = Prepend[m, (b - a) / 2 {1., 1., 1.}]
```

```
{{1., 1., 1.}, {-0.774597, 0., 0.774597}, {0.6, 0., 0.6}}
```

```
(* Desna stran linearnega sistema. *)
r = Table[Integrate[w[x] x^j, {x, a, b}], {j, 0, 2}]
```

$$\left\{2,\ 0,\ \frac{2}{3}\right\}$$

```
(* Koeficienti. *)
ck = LinearSolve[m, r]
```

```
{0.555556, 0.888889, 0.555556}
```

```
(* Numerični približek integrala. *)
s = (b - a) / 2 Sum[ck[[j]] f[xk[[j]]], {j, 1, 3}]
```

```
1.69108
```

Če izračunane polinome normiramo tako, da velja $p_n(1) = 1$, vidimo, da smo konstruirali Legendrove polinome:

```
l[n_, x_] := p[n, x] / p[n, 1]
```

```
Expand[l[2, x]]
```

$$-\frac{3\,x}{2}+\frac{5\,x^3}{2}$$

```
LegendreP[3, x]
```

$$-\frac{3\,x}{2}+\frac{5\,x^3}{2}$$

Literatura

Bohte, Z. *Numerične metode*, DMFA, 1991.
Eugene, D. *Schaum's outline of theory and problems of Mathematica*, McGraw-Hill, 2001.
Grasselli, J. Vadnal A., *Linearna algebra. Linearno programiranje.* DMFA, 1994.
Hoffman, J. D. *Numerical methods for Engineers and Scientists,* 2nd ed., Marcel Dekker, 2001.
Press, W. H., Teukolsky, S. A., Vetterling, W. T., Flannery, B. P. *Numerical Recipes in C*, Cambridge University Press, 1992.
Wolfram, S. *The Mathematica Book* 5th ed., Wolfram Media, 2003.
Zakrajšek, E. *Matematično modeliranje*, DMFA, 2004.