Programi v Matlabu za predmet numerične metode

18. 04 2002

1 Reševanje nelinearnih enačb

Napisali bomo program za reševanje nelinearnih enačb z uporabo posameznih metod.
Rešujete nelinearne enačbe oblike $f(x) = 0$. Program zapišete na funkcijsko `m`-datoteko, natanko tako kot piše v navodilih pri vsaki metodi. Natančno morate upoštevati navodila, kakšni so vhodni in kakšni so izhodni parametri. Funkcijo podamo v obliki niza, kot na primer:

```
>> f='x^2*sin(x)-x';
```

in jo prenesemo v program kot spremenljivko `f`. Neodvisno spremenljivko funkcije vedno označite z `x`.

1. **Metoda bisekcije**
   Pri metodi *bisekcije* moramo poleg funkcije podati še krajišči začetnega intervala x_0, x_1 in natančnost ϵ. Funkcijski vrednosti v krajiščih morata biti različnega znaka. Natančnost izračuna korena enačbe po n iteracijah metode je

   $$|\hat{x} - x_n| < \left| \frac{x_1 - x_0}{2^n} \right|$$

   kjer smo z $\hat{x}$ označili točno rešitev enačbe z x_n pa izračunano vrednost po n iteracijah.

   Koraki metode bisekcije:

   (a) Izračunamo vrednosti $f_0 = f(x_0)$ in $f_1 = f(x_1)$. Če sta različnega znaka nadaljujemo, sicer pa javimo napako: `error('bisekcija: napaka (1)')`.

   (b) Vstopimo v zanko.

   (c) Izračunamo vrednosti

   $$x_2 = \frac{x_0 + x_1}{2} \quad \text{in} \quad f_2 = f(x_2)$$

   (d) Če je $|f_2| < \epsilon$, zapustimo zanko in ponudimo rešitev x_2.

   (e) V primeru, ko sta f_0 in f_2 različnega znaka postavimo $x_1 = x_2$ in $f_1 = f_2$, sicer pa $x_0 = x_2$ in $f_0 = f_2$.

   (f) Če je $|x_0 - x_1| < \epsilon$, zapustimo zanko in ponudimo rešitev $x_2 = \frac{x_0 - x_1}{2}$, sicer nadaljujemo s točko (1b).

```
%[x,X]=bisekcija(f,a,e)
% vhod:
% f funkcija,
% a(1) in a(2) krajisci zacetnega intervala,
% e natancnost.
% izhod:
% x resitev enacbe,
% X matrika krajisc vmesnih intervalov.
function [x,X] = bisekcija(f,a,e)
- vasa koda -
return
```

2. **Sekantna metoda:**
   Podani sta krajišči začetnega intervala x_0 in x_1, natančnost ϵ in maksimalno število iteracij n. Pri sekantni metodi ni nujno, da se koren enačbe $f(x) = 0$ nahaja na začetnem intervalu $[x_0, x_1]$, zato tudi ni nujno, da postopek konvergira. Da se ne bi znašli v neskončni zanki, podamo na začetku maksimalno število iteracij n.

(a) Če je $|f(x_0)| < |f(x_1)|$, zamenjamo vlogi x_0 in x_1. S tem zahtevamo, da je funkcijska vrednost v desnem krajišču absolutno manjša kot v levem. V splošnem ni nujno, da je desno krajišče bližje korenu enačbe, vendar pa statistično gledano metoda v povprečju hitreje konvergira.

(b) Vstopimo v zanko, ki ima lahko največ n prehodov.

(c) Izračunamo približek korenu

$$x_2 = x_1 - f(x_1)\frac{x_0 - x_1}{f(x_0) - f(x_1)}$$

(d) in postavimo $x_0 = x_1$ in $x_1 = x_2$.

(e) Če je $|f(x_2)| < \epsilon$ ali $|x_1 - x_0| < \epsilon$, potem zapustimo zanko in vrnemo rešitev x_2,

(f) sicer pa nadaljujemo z (2b).

```
%function [x,X] = sekantna(f,a,e,n)
% vhod:
% f funkcija,
% a(1) in a(2) krajisci zacetnega intervala,
% e natancnost,
% n maksimalno stevilo iteracij.
% izhod:
% x resitev enacbe,
% X matrika krajisc vmesnih intervalov.
function [x,X] = sekantna(f,a,e,n)
- vasa koda -
return
```

3. **Metoda regula falsi:**
Metoda je podobna sekantni, le da v tem primeru zahtevamo, da sta funkcijski vrednosti v krajiščih začetnega intervala različnega znaka, kar nam zagotavlja, da se na intervalu nahaja vsaj ena ničla. Podamo krajišči začetnega intervala x_0 in x_1, funkcijo $f(x)$ ter natančnost ϵ.

(a) Preverimo, če sta funkcijski vrednosti $f(x_0)$ in $f(x_1)$ na krajiščih začetnega intervala $[x_0, x_1]$ različnega znaka, če nista javimo napako `error('regula: napaka (1)')`.

(b) Vstopimo v zanko.

(c) Izračunamo približek

$$x_2 = x_1 - f(x_1)\frac{x_0 - x_1}{f(x_0) - f(x_1)}$$

(d) Če je $f(x_2)$ je nasprotnega znaka kot $f(x_0)$, potem postavimo $x_1 = x_2$, sicer pa $x_0 = x_2$.

(e) Če je $|f(x)| < \epsilon$ ali $|x_0 - x_1| < \epsilon$, zapustimo zanko in ponudimo rešitev x_2, sicer pa nadaljujemo z (3b).

```
%[x,X]=regula(f,a,e)
% vhod:
% f funkcija,
% a(1) in a(2) krajisci zacetnega intervala,
% e natancnost.
```

```
% izhod:
% x resitev enacbe,
% X matrika krajisc vmesnih intervalov.
function [x,X] = regula(f,a,e)
- koda -
return
```

4. **Newtonova metoda:**
   Na začetku je podan približek korena x_0, natančnost ϵ in maksimalno število iteracij n. Poleg funkcijskega predpisa $f(x)$ moramo podati še odvod $f'(x)$. Konvergenca Newtonove metode je kvadratična.

   (a) Vstopimo v zanko, ki ima lahko največ n prehodov.

   (b) Če je $|f'(x_0)| < \epsilon$, javimo napako `error('newton: (1)')`, sicer pa izračunamo nov približek,
   $$x_1 = x_0 - f(x_0)/f'(x_0)$$

   (c) Če je $|f(x_1)| < \epsilon$ ali $|x_1 - x_0| < \epsilon$, zapustimo zanko in vrnemo približek x_1, sicer pa postavimo $x_0 = x_1$ in nadaljujemo z (4a).

```
%[x,X] = newton(f,df,a,e,n)
% vhod:
% f(x) funkcija,
% df(x) odvod funkcije,
% a zacetni priblizek,
% e natancnost,
% maksimalno stevilo iteracij.
% izhod:
% x resitev enacbe
% X vektor zaporednih priblizkov.
function [x,X] = newton(f,df,a,e,n)
- koda -
return
```

5. **Haleyeva metoda:**
   Na začetku je podan približek korena x_0, natančnost ϵ in maksimalno število iteracij n. Poleg funkcijskega predpisa $f(x)$ in odvoda $f'(x)$, podamo še $f''(x)$.

   (a) Vstopimo v zanko, ki ima lahko največ n prehodov.

   (b) Če je $|f'(x_0)^2 - \frac{1}{2}f(x_0)f''(x_0)| < \epsilon$, javimo napako `error('haley: (1)')`, sicer pa izračunamo nov približek,
   $$x_1 = x_0 - \frac{f(x_0)f'(x_0)}{f'(x_0)^2 - \frac{1}{2}f(x_0)f''(x_0)}$$

   (c) Če je $|f(x_1)| < \epsilon$ ali $|x_1 - x_0| < \epsilon$, zapustimo zanko in vrnemo približek x_1, sicer pa postavimo $x_0 = x_1$ in nadaljujemo z (5a).

```
%[x,X] = haley(f,df,ddf,a,e,n)
% vhod:
```

```
% f(x) funkcija,
% df(x) odvod funkcije,
% ddf(x) drugi odvod funkcije,
% a zacetni priblizek,
% e natancnost,
% maksimalno stevilo iteracij.
% izhod:
% x resitev enacbe
% X vektor zaporednih priblizkov.
function [x,X] = newton(f,df,ddf,a,e,n)
- koda -
return
```

6. **Newtonova metoda za sisteme enačb:**
   Rešujemo sistem nelinearnih enačb $\vec{F}(\vec{x}) = 0$. Podana je vektorska funkcija $\vec{F}(\vec{x})$, njena Jacobijeva matrika $J(\vec{x})$, začetni približek $\vec{x}_0$, natančnost ϵ in maksimalno število iteracij n. Ne piši posebnega programa prilagodi program `newton.m` tako, da bo znal reševati tako enačbe z eno neznanko kot z več neznankami, v realnem ali v kompleksnem.

   (a) Vstopimo v zanko, ki nima več kot n prehodov.

   (b) Če je matrika $\mathcal{J}$ singularna javimo napako `error('newton: (1)')` in zapustimo program.

   (c) Izračunamo naslednji približek

   $$\vec{x}_1 = \vec{x}_0 - \mathcal{J}^{-1}(\vec{x}_0) \cdot \vec{F}(\vec{x}_0)$$

   (d) Če je $\|\vec{F}(\vec{x})\| < \epsilon$ ali $\|\vec{x}_1 - \vec{x}_0\| < \epsilon$, končaj program in ponudi rešitev $\vec{x}_1$, sicer postavi $\vec{x}_0 = \vec{x}_1$ in nadaljuj z (6a).

```
%[x,X] = newton(f,df,a,e,n)
% vhod:
% f(x) funkcija,
% df(x) je Jacobijeva matrika,
% a zacetni priblizek,
% e natancnost,
% maksimalno stevilo iteracij.
% izhod:
% x resitev enacbe,
% X matrika zaporednih priblizkov.
function [x,X] = newton(f,df,a,e,n)
- vasa koda -
return
```

Primer 1.1. Zapišimo primer programa za reševanje nelinearnih enačb. Izberimo eno od metod, na primer Newtonovo. Zapišimo na datoteko `newton.m` naslednjo kodo:

```
%[x,X] = newton(f,df,a,e,n)
% vhod:
% f(x) funkcija,
% df(x) je Jacobijeva matrika,
```

```
% a zacetni priblizek,
% e natancnost,
% n maksimalno stevilo iteracij.
% izhod:
% x resitev enacbe,
% X matrika zaporednih priblizkov.
function [x,X] = newton(f,df,a,e,n)
    X=a; % shranimo zacetni priblizek
    x=a; % postavimo vrednost spremenljivke x.
    for k=1:n
        fx=eval(f);
        if abs(fx) < e return, end;
        dfx = eval(df);
        if abs(dfx) < e,
            error('newton: (1)');
        end;
        h = fx/dfx;
        a = x;
        x = x - h;
        X = [X,x];
        if abs(h) < e return, end;
    end;
return
```

Koda še ni prilagojena za reševanje sistemov nelinearnih enačb. Potreben je droben popravek. Poiščite kje. Poglejmo še nekaj primerov reševanja enačb.
Naj bo funkcija podana s predpisom

$$f(x) = x^2 \sin(x) - x$$

Njen odvod je

$$f'(x) = 2x \sin(x) + x^2 \cos(x) - 1$$

Vzemimo začetni približek $x_0 = -4$.

```
>> f='x^2*sin(x)-x';
>> df='2*x*sin(x)+x^2*cos(x)-1';
>> [x,X]=newton(f,df,-4,1e-15,10)
x =

   -2.7726


X =

  Columns 1 through 7

   -4.0000   -4.0000   -3.0802   -2.8327   -2.7759   -2.7726   -2.7726

  Column 8

   -2.7726
```

```
>>
```

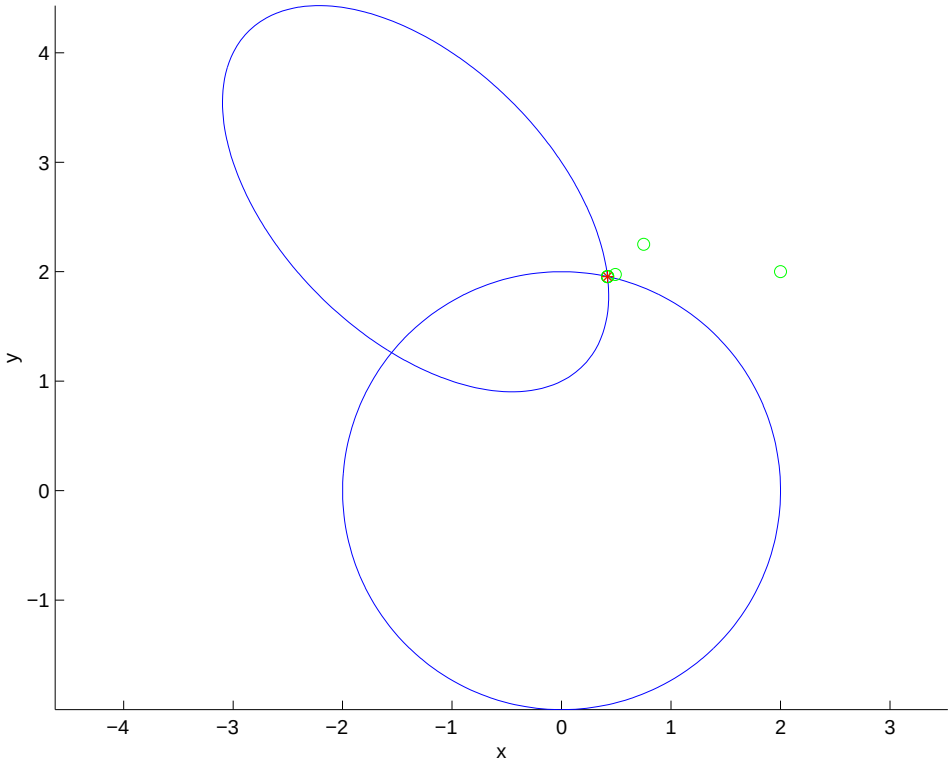
Primer 1.2. Vzemimo primer najpreprostejšega možnega sistema enačb, sistem linearnih enačb $x - 3 = 0$ in $y + 2 = 0$.

```
>> [x,X]=newton(f,df,[4;1],1e-15,10)
>> f='[x(1)-3;x(2)+2]';
>> df='[1,0;0,1]';
x =

     3
    -2


X =

     4     3
     1    -2

>>
```

Ker je sistem linearen, zadošča ena sama iteracija. Sedaj poglejmo še "pravi" sistem nelinearnih enačb. Poiščimo presečišče krožnice $x^2 + y^2 - 4 = 0$ z elipso $x^2 + x\,y + (y-2)^2 = 1$.

```
>> f='[x(1)^2+x(1)*x(2)+(x(2)-2)^2-1;x(1)^2+x(2)^2-4]';
>> df='[2*x(1)+x(2),x(1)+2*x(2)-4;2*x(1),2*x(2)]';
>> [x,X]=newton(f,df,[2;2],1e-15,10)

x =

    0.4201
    1.9554


X =

    2.0000    0.7500    0.4917    0.4223    0.4201    0.4201    0.4201
    2.0000    2.2500    1.9750    1.9562    1.9554    1.9554    1.9554

>>
```

Narišimo še na graf.


4
3
2
1
0
−1
y
−4
−3
−2
−1
0
1
2
3
x

2 Izrek o negibni točki

Bodi funkcija $f(x)$ zvezna in zvezno odvedljiva, kjer je to potrebno. Iščemo rešitve enačbe

$$x = f(x)$$

Rešitve gornje enačbe imenujemo tudi *negibne točke* funkcije $f(x)$. Poizkusimo poiskati negibno točko s pomočjo iteracije. Izberemo začetni približek x_0 in sprožimo iteracijo:

$$x_{n+1} = f(x_n)$$

Zaporedje $\{x_n\}_{n\in\mathbb{N}}$ lahko konvergira k eni od negibnih točk, lahko pa tudi divergira. V primeru, da ima funkcija $f(x)$ več negibnih točk lahko, z izbiro začetnega približka, do določene mere, vplivamo na to, h kateri od njih bo zaporedje konvergiralo.
Pri nekaterih negibnih točkah ne moremo izbrati začetnega približka tako, da bi zaporedje $\{x_n\}_{n\in\mathbb{N}}$ konvergiralo k njim, četudi ga postavimo v njeno bližino.
Izrek o negibni točki govori o lastnostih negibnih točk. Kdaj zaporedje približkov konvergira k dani negibni točki, če postavimo začetni približek dovolj blizu nje.

Izrek 2.1. *Izrek o negibni točki Naj bo funkcija $f(x)$ zvezna in zvezno odvedljiva na nekem intervalu (a, b), ki vsebuje negibno točko $\hat{x}$. Če je na vsem intervalu (a, b) odvod funkcije $f(x)$, po absolutni vrednosti pod 1, potem bo zaporedje*

$$\{x_n\}_{n\in\mathbb{N}}, \quad x_{n+1} = f(x_n)$$

konvergiralo k tej negibni točki za poljubni začetni približek x_0 iz intervala (a, b).

$$\hat{x} = \lim_{n\to\infty}(x_n)$$

Pogoj v izreku je zadosten ni pa potreben.
Funkcija $f(x)$ zvezna in zvezno odvedljiva. V primeru, ko je v negibni točki odvod absolutno pod ena, lahko najdemo odprt interval z negibno točko in odvodom absolutno pod ena. Tako smo v pogojih izreka o negibni točki. Torej velja, da v primeru, ko je v negibni točki odvod pod ena ,lahko najdemo začetni približek tak, da bo zaporedje konvergiralo k tej točki.
Negibne točke v katerih je odvod absolutno pod ena imenujemo *privlačne* tiste, kjer je odvod po absolutno nad ena pa *odbojne*.

1. Napiši program, ki bo implementiral metodo negibne točke za reševanje nelinearnih enačb oblike $f(x) = x$.
   (a) Na začetku imamo podano funkcijo $f(x)$ začetni približek x_0, natančnost ϵ in maksimalno število iteracij n.
   (b) Vstopimo v zanko, ki ima lahko največ n prehodov.
   (c) Izračunamo $x_1 = f(x_0)$.
   (d) Če je $|x_1 - x_0| < \epsilon$, zapustimo zanko in ponudimo rešitev x_1,
   (e) sicer pa postavimo $x_0 = x_1$ in nadaljujemo s točko (1b).

```
%[x,X]=negibna(f,a,e,n)
% vhod:
% f funkcija,
% a zacetni priblizek,
```

```
% e natancnost.
% n maksimalno stevilo iteracij
% izhod:
% x resitev enacbe x=f(x),
% X vektor vmesnih priblizkov.
function [x,X] = negibna(f,a,e,n)
- vasa koda -
return
```

2. Konvergenca metode je linearna, če je odvod v privlačni negibni točki različen od nič. Po Lagrangevem izreku je
$$(x_{n+1} - x_n) < f(\xi)(x_n - x_{n-1})$$
Aitken je predlagal naslednjo metodo za pospešitev konvergence.
   (a) Najprej izračunamo dve običajni iteraciji $x_1 = f(x_0)$ in $x_2 = f(x_1)$.
   (b) Nato izračunamo razliki
$$\delta_1 = x_0 - x_1, \quad \delta_2 = x_2 - 2x_1 + x_0$$
   (c) in popravljen približek
$$\overline{x} = x_0 - \delta_1^2/\delta_2 = (x_0x_2 - x_1^2)/(x_2 - 2x_1 + x_0)$$
   Približek $\overline{x}$ je v splošnem boljši kot bi bil $x_3 = f(x_2)$.
   (d) Nato sledijo dve novi iteraciji z začetnim približkom $\overline{x}$, popravek rezultata in tako naprej.
   (e) Postopek končamo, ko se dve zaporedni iteraciji razlikujeta za manj kot ϵ ali pa je bilo izčrpano maksimalno število iteracij.

   Konvergenca Aitkenove metode je kvadratična.

3. Kjer se δ_2 približa 0 dobimo po (2c) lahko zelo nenatančen rezultat, zato v tem primeru ne bomo računali približka po gornji metodi ampak naredili običajno iteracijo. Zapišimo popravljen algoritem, ki to upošteva.
   (a) Izračunamo nekaj iteracij. Označimo zadnji dve z $x_1 = f(x_0)$ in $x_2 = f(x_1)$.
   (b) Če je razlika $|x_1 - x_0| > \epsilon$, izračunamo kvocient $\gamma = (x_2 - x_1)/(x_1 - x_0)$, sicer pa postavimo $\gamma = 0$.
   (c) Izračunamo popravljen približek
$$\overline{x} = x_2 + \gamma(x_2 - x_1)/(1 - \gamma)$$
   (d) Nato sledijo dve novi iteraciji z začetnim približkom $\overline{x}$, popravek rezultata in tako naprej.
   (e) Postopek končamo, ko se dve zaporedni iteraciji razlikujeta za manj kot ϵ ali pa je bilo izčrpano maksimalno število iteracij.

```
%[x,X]=aitken(f,a,e,n)
% vhod:
% f funkcija,
% a zacetni pribli"zek,
% e natancnost.
% n maksimalno stevilo iteracij
```

```
% izhod:
% x resitev enacbe x=f(x),
% X vektor vmesnih priblizkov.
function [x,X] = aitken(f,a,e,n)
- vasa koda -
return
```

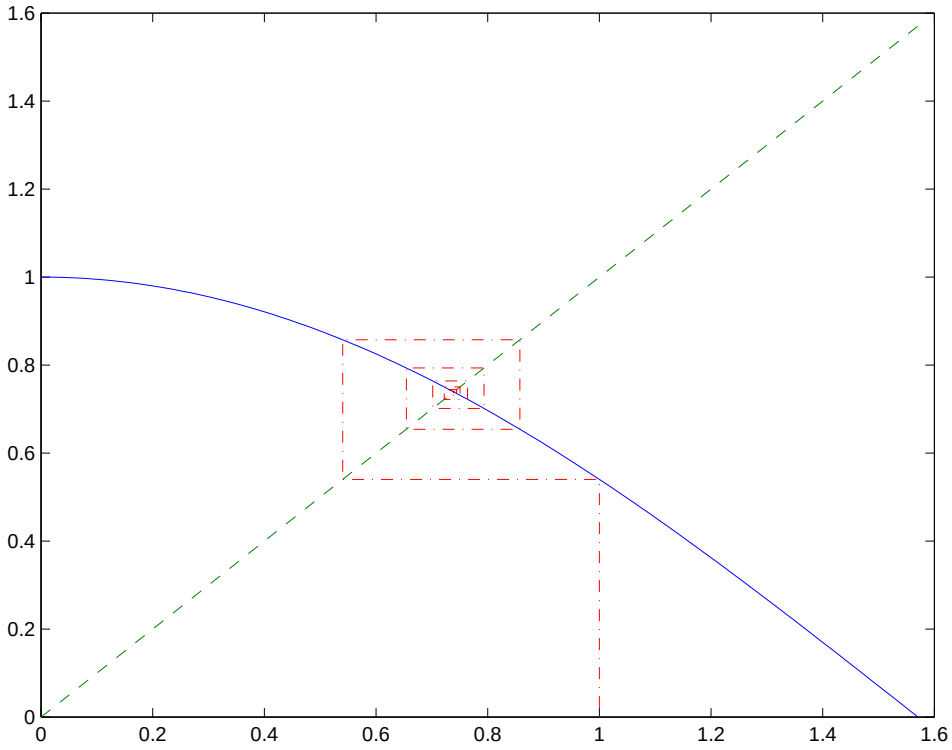
4. Poglejmo primer. Iščemo negibno točko funkcije $\cos(x)$, oziroma rešujemo enačbo $x = \cos(x)$. Uporabili bomo običajno iteracijo negibne točke in Aitkenovo pospešitev metode. Pogledali bomo koliko iteracij je potrebnih v obeh primerih, da izračunamo negibno točko na 4 mesta natančno. Vzeli bomo začetni približek $x_0 = 1$.

```
>> [x,X]=negibna('cos(x)',1,1e-4,100)

x =

    0.7391


X =

  Columns 1 through 7

    1.0000    0.5403    0.8576    0.6543    0.7935    0.7014    0.7640

  Columns 8 through 14

    0.7221    0.7504    0.7314    0.7442    0.7356    0.7414    0.7375

  Columns 15 through 21

    0.7401    0.7384    0.7396    0.7388    0.7393    0.7389    0.7392

  Columns 22 through 23

    0.7390    0.7391


>> [x,X]=aitken('cos(x)',1,1e-4,100)

x =

    0.7391


X =

  Columns 1 through 7

    1.0000    0.5403    0.8576    0.7280    0.7465    0.7341    0.7391
```

```
>>
```

V prvem primeru je bilo potrebnih 23 iteracij, medtem ko v drugem primeru le 7.

5. Predstavimo iteracijo s fiksno točko še grafično. Iščemo presečišče grafa funkcije $y = \cos(x)$ s premico $y = x$. Vzemimo začetni približek $x_0 = 1$. Naslednja slika prikazuje, kako se iteracije približujejo fiksni točki.



3 Eulerjeva transformacija

Na tem mestu omenimo *Eulerjevo transformacijo*, ki pospeši, pod določenimi pogoji, konvergenco slabo konvergentnih alternirajočih vrst. Dana je alternirajoča vrsta

$$s = \sum_{k=0}^{\infty} (-1)^k a_k = a_0 - a_1 + a_2 - \ldots, \quad a_k > 0$$

zapišimo vrsto, ki ima enako vsoto in v splošnem konvergira mnogo hitreje kot prvotna.

$$s = \sum_{k=0}^{\infty} \frac{(-1)^k \Delta^k a_0}{2^{k+1}}$$

kjer je

$$\Delta a_0 = a_1 - a_0, \quad \Delta^2 a_0 = \Delta a_1 - \Delta a_0 = a_2 - 2a_1 + a_0, \quad \ldots$$

oziroma splošno

$$\Delta^k a_0 = \sum_{m=0}^{k} (-1)^m \binom{k}{m} a_{k-m}$$

1. Napiši program, ki bo implementiral Eulerjevo transformacijo dane alternirajoče vrste in izračunal približek za vsoto.

   (a) Podan je predpis za izračun absolutne vrednosti splošnega člena $a_k = f(k)$, $k = 0, 1, \ldots$ in število členov vrste n.

   (b) Izračunamo Eulerjevo transformacijo vrste.

   (c) Seštejemo n členov transformirane vrste in vrnemo vsoto transformirane in prvotne vrste.

```
%[e,p]=eulersum(f,n)
% vhod:
% f predpis ak=f(k),
% n stevilo clenov.
% izhod:
% e vsota n clenov eulerjeve transformacije,
% p vsota n clenov prvotne vrste.
function [e,p] = eulersum(f,a,e,n)
- vasa koda -
return
```

2. Poglejmo nekaj primerov in se prepričajmo v učinkovitost metode. Primerjajmo rezultat s pravo vrednostjo. Najprej poglejmo alternirajočo harmonično vrsto.

$$s = 1 - \frac{1}{2} + \frac{1}{3} - \cdots = \sum_{k=0}^{\infty} (-1)^{k+1} \frac{1}{k+1}$$

   Vsota te vrste je $\log(2)$.

```
>> [e,p]=eulersum('1./k',20)

e =

   0.69314713705103

p =

   0.66877140317543

>> log(2)

ans =

   0.69314718055995

>> [e,p]=eulersum('1./k',40)
```

```
e =

   0.69314718055992

p =

   0.68080338179269

>> [e,p]=eulersum('1./k',50)

e =

   0.69314718055995

p =

   0.68324716057592

>>
```

3. Poglejmo še alternirajočo vsoto recipročnih kvadratov naravnih števil.

$$s = 1 - \frac{1}{4} + \frac{1}{9} - \cdots = \sum_{k=0}^{\infty} (-1)^{k+1} \frac{1}{(k+1)^2}$$

```
>> [e,p]=eulersum('1./k.^2',20)

e =

   0.82246687306889

p =

   0.82127937832975

>>  pi^2/12

ans =

   0.82246703342411

>> [e,p]=eulersum('1./k.^2',40)

e =

   0.82246703342402

p =

   0.82216234105042
```

>>

Vsota te vrste je $\frac{\pi^2}{12}$.

4 Tridiagonalni sistemi in robni problemi

Pri reševanju linearnih robnih problemov naletimo na sisteme enačb, z redko matriko, to je matriko, ki ima malo elementov različnih od nič. Običajno so od nič različni elementi razporejeni na glavni diagonali in na vzporednicah le-te. Pri reševanju robnih problemov v eni dimenziji naletimo na tridiagonalne sisteme. Od nič različni elementi so razporejeni na glavni diagonali in njej sosednjima vzporednicama.

4.1 Tridiagonalni sistemi

1. Tridiagonalna matrika

$$\begin{bmatrix} a_{1,1} & a_{1,2} & 0 & \dots & 0 \\ a_{2,1} & a_{2,2} & a_{2,3} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{m,m-1} & a_{m,m} \end{bmatrix}$$

Definicija 4.1. Matrika je diagonalno dominantna, če velja

$$|a_{i,i}| > \sum_{i \neq j} |a_{i,j}| \quad i,j = 1 \dots m$$

V vsaki vrstici je vsota absolutnih vrednosti izven diagonalnih členov manjša od absolutne vrednosti diagonalnega člena.

Izrek 4.1. *Če ima tridiagonalni sistem enačb diagonalno dominantno matriko, potem se bo Gaussova eliminacija brez pivotiranja vedno uspešno zaključila.*

2. Zapiši program, ki bo reševal tridiagonalne sisteme s pomočjo Gaussove eliminacije brez pivotiranja. Označimo elemente matrike tridiagonalnega sistema takole:

$$\begin{bmatrix} d_1 & u_1 & 0 & \dots & 0 & b_1 \\ l_2 & d_2 & u_2 & \dots & 0 & b_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & l_m & d_m & b_m \end{bmatrix}$$

   (a) Dani so štirje vektorji: vektor z elementi na na vzporednici nad glavno diagonalo u, vektor z elementi na glavni diagonali d, vektor z elementi na vzporednici pod glavno diagonalo l in vektor svobodnih členov b. Vektor z neznankami označimo z x.

   (b) Z Gaussovo eliminacijo prevedemo sistem na zgornjetrikotnega

$$\begin{bmatrix} \delta_1 & u_1 & 0 & \dots & 0 & \beta_1 \\ 0 & \delta_2 & u_2 & \dots & 0 & \beta_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & \delta_m & \beta_m \end{bmatrix}$$

   in ga rešimo z vzvratnim vstavljanjem.

   (c) Gaussova eliminacija:

$$\delta_1 = d_1; \quad \delta_k = d_k - u_{k-1}(l_k/\delta_{k-1}), \quad k = 2, \dots, m$$

$$\beta_1 = b_1; \quad \beta_k = b_k - \beta_{k-1}(l_k/\delta_{k-1}), \quad k = 2, \dots m$$

(d) Vzvratno vstavljanje:

$$x_m = \beta_m/\delta_m; \quad x_k = (\beta_k - u_k x_{k-1})/\delta_k, \quad k = m-1, \ldots, 1$$

```
function x = trisis(u,d,l,b)
% vhod:
% u vektor elementov na vzporednici nad glavno diagonalo. Dolzina je m-1.
% l vektor elementov na vzporednici pod glavno diagonalo. Dolzina je m-1.
% d je glavna diagonala, dolzina je m.
% b je stolpec svobodnih clenov, dolzina je m.
% izhod:
% x je resitev sistema, dolzina je n.
- koda -
return
```

3. Reši tridiagonalni sistem z Jacobievo iteracijo. Program napiši tako, da matrike ne prevedeš na polno obliko.

```
% function x = trijacobi(u,d,l,b,e,n)
% vhod:
% u vektor elementov na vzporednici nad glavno diagonalo. Dolzina je m-1.
% l vektor elementov na vzporednici pod glavno diagonalo. Dolzina je m-1.
% d je glavna diagonala, dolzina je m.
% b je stolpec svobodnih clenov, dolzina je m.
% e natancnost
% n maksimalno stevilo iteracij
% izhod:
% x je resitev sistema, dolzina je n.
- koda -
return
```

4. Reši tridiagonalni sistem z Gauss-Seidlovo iteracijo. Program napiši tako, da matrike ne prevedeš na polno obliko.

```
% function x = trigaussseidel(u,d,l,b,e,n)
% vhod:
% u vektor elementov na vzporednici nad glavno diagonalo. Dolzina je m-1.
% l vektor elementov na vzporednici pod glavno diagonalo. Dolzina je m-1.
% d je glavna diagonala, dolzina je m.
% b je stolpec svobodnih clenov, dolzina je m.
% e natancnost
% n maksimalno stevilo iteracij
% izhod:
% x je resitev sistema, dolzina je m.
- koda -
return
```

4.2 Robni problemi

1. Napiši proram, ki bo rešil linearni robni problem v eni dimenziji.

$$y'' + p(x)y' + q(x)y = r(x), \quad y(a) = y_a, \quad y(b) = y_b$$

Rešujemo tako, odvode nadomestimo s končnimi razlikami in tako prevedemo problem na tridiagonalni sistem linearnih enačb, ki ga rešujemo s proceduro `trisis`.

(a) Razdelimo interval $[a, b]$ na $n + 1$ enakih podintervalov.

(b) Postavimo $h = \frac{b-a}{n+1}$, $y_a = y_0$ in $y_b = y_{n+1}$.

(c) Neznane količine so y_i, $i = 1, \dots n$.

(d) Nadomestimo odvode s končnimi razlikami

$$y'(x_i) \approx \frac{y(x_{i+1}) - y(x_{i+1})}{2h} = \frac{y_{i+1} - y_{i-1}}{2h}$$

in

$$y''(x_i) \approx \frac{y(x_{i+1}) - 2y(x_i) + y(x_{i-1})}{h^2} = \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2}$$

(e) Dobimo sistem enačb

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + p_i \frac{y_{i+1} - y_{i-1}}{2h} + q_i y_i = r_i$$

na levem robu je

$$\frac{y_2 - 2y_1}{h^2} + p_1 \frac{y_2}{2h} + q_1 y_1 = -\frac{y_0}{h^2} + \frac{y_0}{2h} + r_1$$

in na desnem robu

$$\frac{-2y_n + y_{n-1}}{h^2} + p_n \frac{-y_{n-1}}{2h} + q_n y_n = -\frac{y_{n+1}}{h^2} - \frac{y_{n+1}}{2h} + r_n$$

(f) Sestavimo vektorje `u`, `d`, `l` in stolpec svobodnih členov `b`. Sistem rešimo s programom `trisis`.

```
% function [y,x]=robniproblem(p,q,r,ya,a,n)
% vhod:
% p, q in r so funkcije
% ya(1), ya(2) vrednost v levem in desnem krajiscu in intervala
% [a(1),a(2)]
% n stevilo notranjih tock intervala
% izhod:
% y vektor vrednosti funkcije y(x) v notranjih tockah intervala in na robu
% x vektor vrednosti neodvisne spremenljivke x.
- vasa koda -
return
```

Primer 4.1. (a) Rešite naslednji robni problem. Problem porazdelitve toplote v cevi. Zunanji radij cevi je 3 notranji radij pa je 1. Na zunanji površini je konstantna temperatura 40^o na notranji pa 80^o. Diferencialna enačba, ki opisuje porazdelitev toplote v cevi je

$$\frac{d^2T}{d\rho^2} + \frac{1}{\rho}\frac{dT}{d\rho} = 0, \qquad T(3) = 40, \quad T(1) = 80$$

Laplaceova enačba $\Delta T = 0$ zapisana v polarnih koordinatah. Porazdelitev ni odvisna od kota in se s časom ne spreminja, zato so odvodi na kot in na čas enaki nič. Razdeli interval [1,3] na 100 delov in nariši graf. Funkcija $p(r) = \frac{1}{\rho}$, medtem ko sta funkciji $q(\rho) = 0$ in $r(\rho) = 0$.

```
>> [y,x]=robni('1/x','0','0',[1,3],[80,40],100); plot(x,y);
>>
```

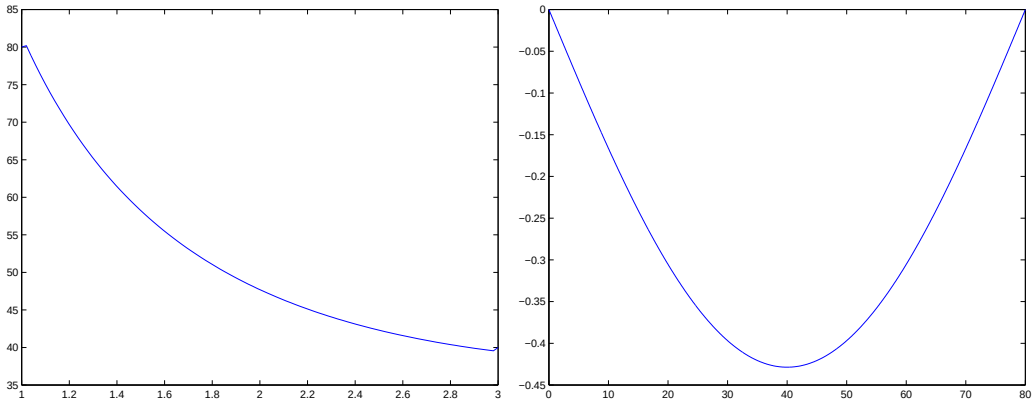
(b) Opazujmo upogib tankega na konceh pritrjenega nosilca. Teža je enakomerna razporejena po nosilcu. Upogib se podreja zakonu

$$-u'' + p\,u = q\,x(L-x), \quad 0 < x < L, \quad u(0) = u(L) = 0$$

Funkciji $p(x)$ in $q(x)$ določata fizikalne lastnosti nosilca oziroma težo, ki jo nosi, medtem ko je L je njegova dolžina. Vzemimo, da je $p = 7 \times 10^{-6}$, $q = 4 \times 10^{-7}$ in dolžina $L = 80$. Koliko je maksimalni odmik nosilca.

```
>> [y,x]=robni('0','(7e-6)','(4e-7)*x*(80-x)',[0,80],[0,0],1000);
>> plot(x,y);
>> abs(min(y))

ans =

  0.42861868643133

>>
```

(c) Naslednja grafa prikazujeta rešitvi v obeh primerih.



5 Aproksimacija funkcij po metodi najmanjših kvadratov

1. Napiši program, ki bo poiskal koeficiente linearne kombinacije danih funkcij tako, da se bo prilegala merjenim podatkom po metodi najmanjših kvadratov.

   Dane so funkcije $f_i(x),\ i = 1, \ldots, m$, in merjene vrednosti $(x_j, y_j),\ j = 1, \ldots, n$, kjer je $m \leq n$. Poišči nabor koeficientov $a_i,\ i = 1, \ldots, m$ linearne kombinacije

   $$f(x) = \sum_{i=1}^{m} a_i f_i(x)$$

   tako, da bo funkcija

   $$F(a_1, \ldots, a_m) = \sum_{j=1}^{n} (f(x_j) - y_j)^2$$

   zavzela minimalno vrednost. Nabor koeficientov ustreza sistemu

   $$\frac{\partial F}{\partial a_i} = 0, \quad i = 1, \ldots, m$$

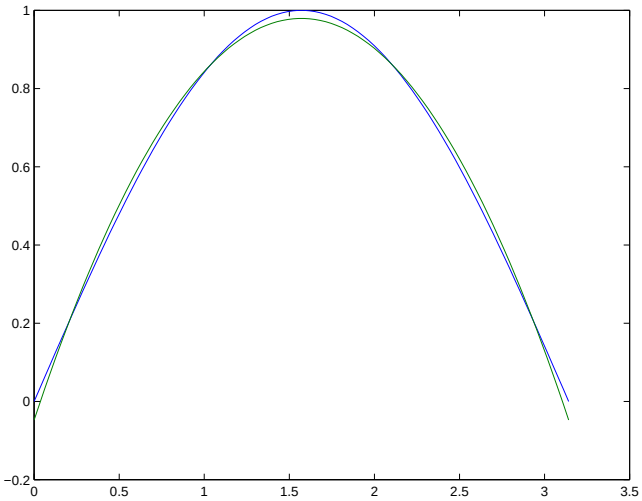
   Rešitev zgornjega sistema je optimalna rešitev predoločenega sistema

   $$\sum_{i=1}^{m} a_i f_i(x_j) = y_j, \quad j = 1, \ldots, n$$

```
%a=linlsq(h,x,y)
function a=linlsq(h,x,y)
% vhod:
% h je seznam funkcij na primer: h='[ena(x),x,x^2]'
% x y abscise in ordinate aproksimacijskih tock
% izhod:
% a vektor optimalnih vrednosti parametrov
- vasa koda -
```

   Primer 5.1. Poglejmo primer. Poiščimo aproksimacijo funkcije $\sin x$ s polinomom druge stopnje na intervalu $[0, \pi]$.

```
>> x=linspace(0,pi);
>> y=sin(x);
>> x=linspace(0,pi)';
>> y=sin(x);
>> f='[ena(x),x,x^2]';
>> a=linlsq(f,x,y);
>> yy=eval(vectorize(f))*a;
>> plot(x,y,x,yy);
```



2. Rešujemo nelinearni problem najmanjših kvadratov tako, da ga prevedemo na linearnega. Razložimo na primeru. Namesto nelinearnega problema aproksimacije funkcije

$$f(x) = a_1 e^{a_2 x}$$

rešujemo lineariziran problem

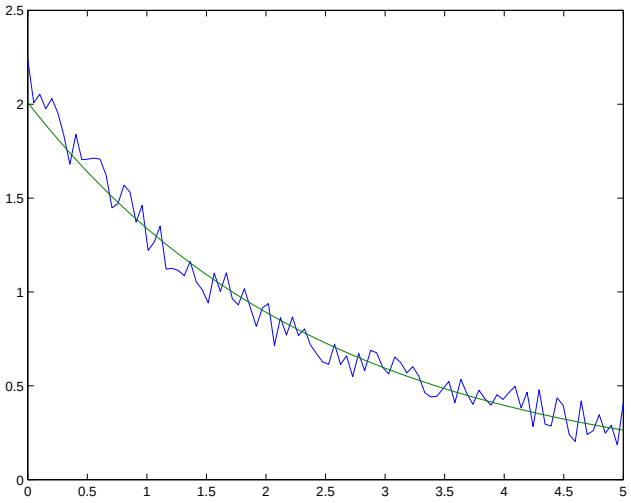
$$\log(f(x)) = \log(a_1) + a_2 x$$

Pri tem smo naredili naslednje transformacije $x := x$ in $y := \log(y)$. Rešitev moramo transformirati takole $[a_1, a_2] := [e^{a_1}, a_2]$. Procedura bo sprejemala seznam funkcij `h`, ta je v našem primeru enak `h='[ena(x),x]'`, transformacijo koordinat `fxy`, v našem primeru `fxy='[x,log(y)]'`, in transformacijo rešitve `fa` v našem primeru `fa='[exp(a(1)),a(2)]'`.

```
%a=lnslsq(h,fxy,fa,x,y)
function a=lnslsq(h,fxy,fa,y)
% h je seznam funkcij na primer
$ fxy je transformacija koordinat
$ fa je transformacija resitve
% x y abscise in ordinate aproksimacijskih tock
- vasa koda -
```

Primer 5.2. Aproksimirajmo funkcijo $f(x) = a_1 e^{a_2 x}$. Pripravimo vrednosti x in y, ki jih bomo aproksimirali.

```
>> f='a(1)*exp(a(2)*x)';
>> a=[2,-1/2];
>> x=linspace(0,5)';
>> y=eval(f)+rand(size(x))/4;
>> a=linslsq('[ena(x),x]','[x,log(y)]','[exp(a(1)),a(2)]',x,y)

a =

    2.0092   -0.4061

>> yy=eval(f)
>> plot(x,y,x,yy)
```



3. Napiši program, ki bo reševal nelinearni problem najmanjših kvadratov s pomočjo Newtonove iteracije.

   Dana je funkcija $f(x, a_1, \ldots, a_m)$ in merjene vrednosti $(x_j, y_j),\ j = 1, \ldots, n$, kjer je $m \leq n$. Poiščimo nabor parametrov $a_i,\ i = 1, \ldots m$ tako, da bo funkcija

   $$F(a_1, \ldots, a_m) = \sum_{j=1}^{n} (f(x_j, a_1, \ldots, a_m) - y_j)^2$$

   zavzela minimalno vrednost. Potreben pogoj za nastop ekstrema je

   $$\frac{\partial F}{\partial a_i} = 0, \quad i = 1, \ldots, m$$

   Sistem enačb bomo reševali kot sistem nelinearnih enačb po Newtonovi metodi, zato moramo podati tudi začetni približek.

```
% a=nolinlsq(h,u,a,x,y,e,n)
function a=nolinlsq(h,u,a,x,y,e,n)
% vhod:
% h funkcija
% u ime vektorja parametrov
% a zacetni priblizek
% x y abscise in ordinate aproksimacijskih tock
% e natancnost
% n maksimalno stevilo iteracij
% izhod:
% a vektor optimalnih vrednosti parametrov
- koda -
return
```

   Primer 5.3. Poglejmo primer. Vzemimo aproksimacijko funkcijo iz zgornjega primera $f(x) = a_1 e^{a_2 x}$ in točke, ki jih nekoliko raztresemo s pomočjo funkcije `rand`, da dobimo bolj realističen primer.

```
>> f='a(1)*exp(a(2)*x)';
>> a=[2,-1/2];
>> x=linspace(0,4,5)

x =

     0     1     2     3     4


>> y=fix(10*(eval(f)+rand(size(x))/3))/10

y =

    2.1000    1.5000    1.1000    0.5000    0.4000
```

Nato pokličemo program

```
>> nolinlsq(f,'a',a,x,y,1e-10,100)

ans =

    2.1490
   -0.4000

>>
```

Za realizacijo programa `nolinlsq` potrebujemo program `F=lsqr(f,x,y)`, ki vrne funkcijo $F(a_1, \ldots, a_m)$, katere minimum moramo poiskati. V našem primeru je ta

```
F =

    (a(1)-21/10)^2+(a(1)*exp(a(2))-3/2)^2+(a(1)*exp(2*a(2))-11/10)^2+...
    (a(1)*exp(3*a(2))-1/2)^2+(a(1)*exp(4*a(2))-2/5)^2
```

Nato pokličemo program `a=minimum(F,'a',a,1e-10,100)`, ki sprejme funkcijo `F` in vrne optimalno vrednost vektorja parametrov `a`. Program `minimum` reši sistem `jacobian(F,'a') = 0` z Newtonovo metodo. Program `minimum` morate napisati sami.

```
>> nolinlsq(f,'a',a,x,y,1e-10,100)

ans =

    2.1490
   -0.4000

>>
```

6 Navadne diferencialne enačbe in robni problemi

1. Napiši program, ki bo rešil sistem navadnih diferencialnih enačb prvega reda

$$y' = f(x, y), \quad y(x_0) = y_0$$

z modificirano Eulerjevo metodo.

$$k_1 = f(x_i, \eta_i); \quad k_2 = f(x_i + \alpha h, \eta_i + \beta h k_1)$$

$$\eta_{i+1} = \eta_i + h(\gamma_1 k_1 + \gamma_2 k_2)$$

Kjer je

$$\gamma_1 = \gamma_2 = \frac{1}{2}, \quad \alpha = \beta = 1$$

Z η_i smo označili približno vrednost za rešitev y v točki x_i, $y_i = y(x_i)$ Program morate napisati tako, da bo reševal tudi sisteme enačb prvega reda, kar pomeni, da je odvisna spremenljivka y lahko tudi vektor.

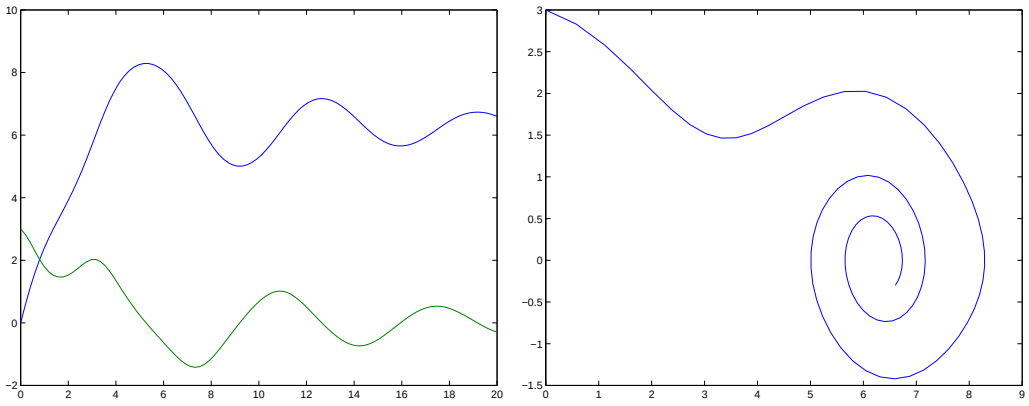
```
% function [y,x]=modeuler(f,y,a,n)
% vhod:
% f je funkcija f(x,y), kot desna stran diferencialne enacbe y'=f(x,y)
% a=[a(1),a(2)] interval za x;
% y je vrednost funkcije v a(1)
% n stevilo notranjih tock xi intervala za x
% izhod:
% y matrika približnih vrednosti funkcije y(x) v notranjih tockah intervala in na robu
% x ustrezne vrednosti neodvisne spremenljivke
- vasa koda -
return
```

Primer 6.1. Rešimo diferencialo enačbo za fizično nihalo na intervalu $[0, t]$

$$y'' + \beta y' + \omega^2 sin(y) = 0, \quad y(0) = a, \quad y(t) = b$$

kjer je $y(0) = a$ začeten odmik nihala in $y'(0) = b$ je začetna hitrost. Vzemimo primer: $t = 20$, $a = 0$, $b = 3$, $\beta = 0.2$ in $\omega = 1$. Število vozlišč naj bo $n = 100$. Naredimo graf odmika in hitrosti v odvisnosti od časa, in `fazni diagram` oziroma graf hitrosti v odvisnosti od odmika.

```
>> [y,x]=modeuler('[y(2),-0.2*y(2)-sin(y(1))]',[0,20],[0,3],100);
>> plot(x,y)
>> figure; plot(y(1,:),y(2,:));
```



2. Napiši program, ki bo rešil robni problem v eni dimenziji s strelsko metodo. Rešujemo diferencialno enačbo $y'' = f(x, y, y')$ na intervalu $[a, b]$. Dane so robne vrednosti $y_a = y(a)$ in $y_b = y(b)$. Sestavimo funkcijo $y_b(t)$, ki z modificirano Eulerjevo metodo določi vrednost v desnem krajišču, če sta pogoja v levem krajišču $y(a) = y_a$ in $y'(a) = t$. Določimo dve začetni vrednosti za parameter t, t_0 in t_1 in rešimo enačbo $y_b(t) - y_b = 0$ s sekantno metodo.

```
% function [y,x]=strelska(f,a,ya,t,n,e,m)
% vhod:
% f(x,y(1),y(2)) je desna stran enacbe y'' = f(x, y, y')
% ya(1), ya(2) vrednost v levem in desnem krajiscu in intervala
% a = [a(1),a(2)]
% t dve zacetne vrednosti y'(1) za strelasko metodo
% n stevilo notranjih tock intervala
% e natancnost sekantne metode.
% m
% izhod:
% y vektor vrednosti funkcije y(x) in y'(x) v notranjih tockah intervala in na robu
% x neodvisna spremenljivka
- vasa koda -
return
```

Primer 6.2. Rešimo robna prioblema iz primera 4.1

```
>> [y,x]=strelska('-1/x*y(2)',[1,3],[80,40],[1,2],100);
>> plot(x,y(1,:));
>> [y,x]=strelska('(7e-6)+(4e-7)*x*(80-x)',[0,80],[0,0],[-0.5,-0.1],240,1e-10,100);
>> plot(x,y(1,:));
>> abs(min(y(1,:)))

ans =

   0.42869860987213

>>
```