3. *Netiesinių lygčių ir jų sistemų sprendimo metodai*

Šioje skyriuje nagrinėsime vieną iš pagrindinių skaičiavimo matematikos uždavinių – lygčių sprendimo uždavinį. Tai yra vienas iš seniausių matematikos uždavinių. Babilone rastos molio plokštelės, datuojamos 1700 m. prieš Kristaus gimimą, kuriose aiškinama, kaip apskaičiuoti $\sqrt{2}$, t.y., kaip spręsti lygtį: $x^2 - 2 = 0$.

Lygčių bei jų sistemų sprendimo uždavinys sutinkamas fizikoje, inžinerijoje, matematikoje, chemijoje ir kitose srityse.

Gimnazijose ir vidurinėse mokyklose mokiniai mokomi spręsti pirmo laipsnio ir kvadratines algebrines lygtis, rodiklines, logaritmines ir trigonometrines lygtis. Visais tais atvejais išvedamos formulės, pagal kurias analitiškai apskaičiuojamos šaknų reikšmės. Sudėtingesnės lygtys yra pertvarkomos į tokias lygtis, kurių sprendiniai yra žinomi, t.y., kurias mokame spręsti. Kitaip tariant, mokyklose lygtims spręsti naudojamas „arbatinuko“ principas*).

Lygčių $f(x) = 0$, kurių šaknis galima išreikšti lygties koeficientais naudojant elementarias operacijas (aritmetines, kėlimo laipsniu, šaknies traukimo, logaritmavimo ir pan.), yra nedaug. Todėl, sprendžiant lygtis, šaknų ieškojimo uždavinys paprastai suskaidomas į du:

1) šaknų atskyrimo uždavinį,

2) šaknies tikslinimo uždavinį, kai žinomas tos šaknies izoliacijos intervalas arba apytikslė šaknies reikšmė.

Iš pradžių aptarsime, kaip sprendžiama viena lygtis, o lygčių sistemos sprendinių ieškojimo uždavinį nagrinėsime vėliau.

3.1 Lygties f(x)=0 sprendimas

Uždavinio formulavimas.

Sakykime, duota lygtis $f(x) = 0$. Reikia rasti tokias x reikšmes s, su kuriomis $f(s)$ reikšmė būtų lygi nuliui. Kitaip tariant, reikia rasti taškų, kuriuose funkcijos $y = f(x)$ grafikas kerta x-o ašį, abscises (x reikšmes).

---

*) Ši sąvoka atsirado iš tokio pasakojimo.

Fizikas ir matematikas sprendžia uždavinį. Kambaryje yra vandens čiaupas, arbatinukas ir dujinė viryklė. Kaip užvirinti vandenį?

Ir fizikas, ir matematikas pateikia tą patį uždavinio sprendimo metodą:

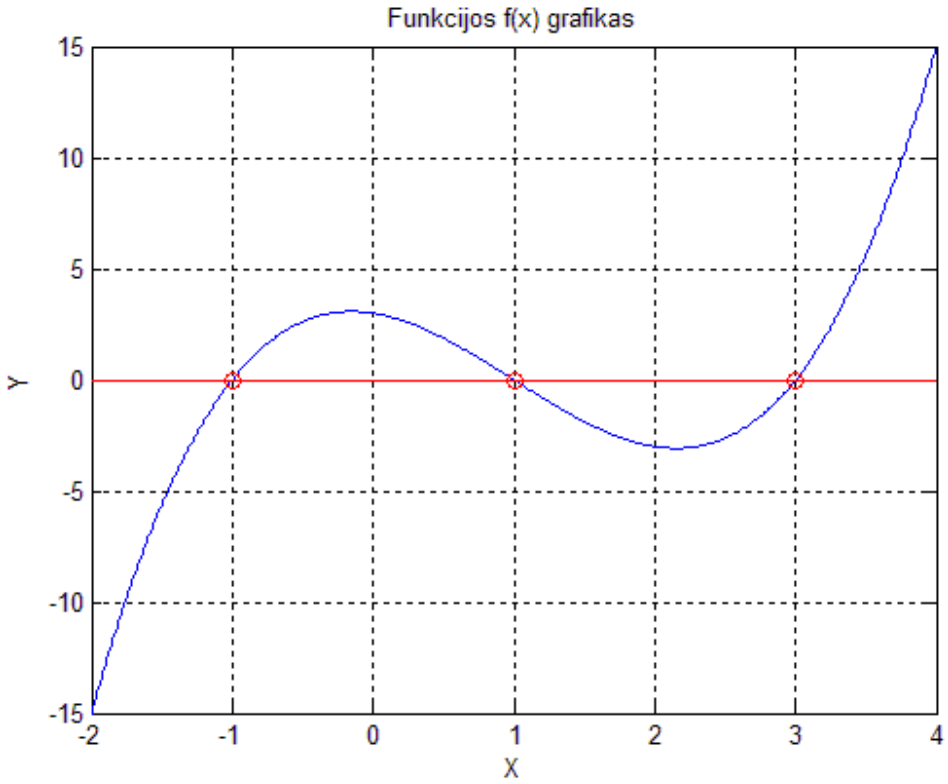
1. Pripilti į arbatinuką vandens,
2. Arbatinuką uždėti ant dujinės viryklės,
3. Užkurti ugnį.

Dabar jiems pateikiamas kitas uždavinys. Kambaryje yra vandens čiaupas, arbatinukas ir dujinė viryklė. Arbatinukas su vandeniu uždėtas ant dujinės viryklės. Kaip užvirinti vandenį?

Fizikas sako, kad tai dar lengviau,- tik reikia užkurti ugnį. Tačiau matematikas pasipiktina. Nieko panašaus,- sako jis. Reikia arbatinuką nukelti nuo viryklės, išpilti vandenį ir turėsime uždavinį, kurį jau mokame spręsti.

Todėl, kai uždavinys suvedamas į uždavinį, kurį mokame spręsti, dažnai juokais sakoma, kad sprendžiant uždavinį panaudojamas „arbatinuko“ principas.

3.1 paveikslėlyje pavaizduotas funkcijos $f(x) = x^3 - 3x^2 - x + 3$ grafikas ir raudonais apskritimėliais pažymėtos lygties $x^3 - 3x^2 - x + 3 = 0$ šaknys.



3.1 pav. Lygties $f(x) = x^3 - 3x^2 - x + 3 = 0$ šaknys

3.1.1 Šaknų atskyrimo uždavinys

Apibrėžimas. Jei lygtis $f(x) = 0$ intervale $(a; b)$ turi vienintelę šaknį, tai intervalas $(a; b)$ yra šaknies izoliacijos intervalas.

Šaknų atskyrimo uždavinys formuluojamas taip: duota lygtis $f(x) = 0$; reikia rasti jos šaknų izoliacijos intervalus arba šaknų apytiksles reikšmes.

Bendrų šio uždavinio sprendimo metodų nėra, todėl aptarkime dažniausiai taikomus jo sprendimo būdus.

1. ***Grafinis būdas.*** Lygtį $f(x) = 0$ parašykime išraiška $f_1(x) = f_2(x)$ ir tarkime, kad mokame nubraižyti funkcijų $f_1(x)$ bei $f_2(x)$ grafikus. Aišku, kad tų grafikų sankirtos taškų abscisės bus lygties $f(x) = 0$ šaknys. Šis uždavinys lengvai sprendžiamas MATLAB'o aplinkoje, kuri įgalina lengvai nubrėžti funkcijų $f_1(x)$ ir $f_2(x)$ grafikus ir, pasinaudojant funkcija ***ginput***, aproksimuoti susikirtimo taškų koordinates.

Funkcija ginput. Funkcija ***ginput*** yra skirta pelės pagalba įvesti grafiko, išbrėžto konkrečioje koordinačių sistemoje, pažymėtus taškus.

Kreipinys į funkciją yra:

[x,y]=ginput(n),

čia *n* – natūralusis skaičius, žymintis pažymėtų taškų skaičių,

x ir y – *n*-elemenčiai vektoriai, nusakantys pažymėtų taškų koordinates.

Žemiau pateikiama funkcija ***plotap2***, realizuojanti grafinį šaknų ieškojimo būdą.

```
function approx=plotap2(func1,func2,x0,h,x1)
% PLOTAP2 įgalina rasti funkcijų y=f1(x) ir y=f2(x) susikirtimo
% tašką.
% Įėjimo parametrai
% func1    - vardas funkcijos, apskaičiuojančios y=f1(x) reikšmes,
% func2    - vardas funkcijos, apskaičiuojančios y=f2(x) reikšmes,
% x0 ir x1 - galai intervalo, kuriame brėžiami funkcijų grafikai,
% h        - intervalo [x0; x1],diskretizavimo žingsnis.
% Išėjimo parametrai
% approx   - (approx(1);approx(2)) pažymėto susikirtimo taško
%            koordinatės.

approx=[]; x=x0:h:x1; y1=feval(func1,x);
%  Brėžiamas funkcijos y=f1(x) grafikas
plot(x,y1,'r','LineWidth',1.5);
hold on;
xlabel('x'); ylabel('y');
hold on;
%  Brėžiamas funkcijos y=f2(x) grafikas
y2=feval(func2,x);
plot(x,y2,'b','LineWidth',1.5);
title('Pažymėkite susikirtimo tašką ir paspauskite kairįjį klavišą');
grid on;
% Susikirtimo tašką aproksimuoja MATLAB'o funkcija "ginput"
approx=ginput(1); hold off;
```

Pavyzdys. *Pasinaudodami funkcija* ***plotap2****, grafiškai raskime lygties* $2x\cos 2x-(x+1)^2=0$ *vieną (mažesniąją) šaknį.*

Lygtį perrašykime pavidalu $2x\cos 2x=(x+1)^2$, ir komandų lange surinkime komandą:

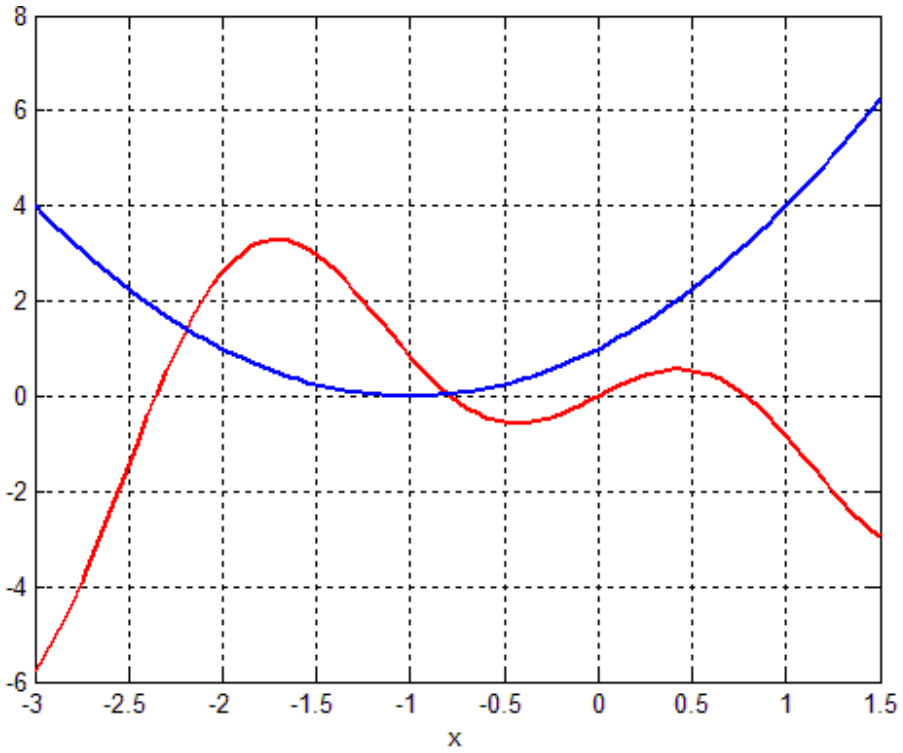
```
>> approx=plotap2(inline('2*x.*cos(2*x)'),...
                  inline('(x+1).^2'),-3,0.05,1.5)
```

Ekrane pasirodys brėžinys (žr. 3.2 pav.). Tada, pele pažymėję kairįjį susikirtimo tašką, turėsime:

```
approx = -2.1964   1.4298.
```

Tas rezultatas reiškia, kad apytikslė mažesniosios šaknies reikšmė $x0$ yra:

$x0=$`approx(1)= -2.1964.`



3.2 pav. Funkcijų $y_1 = 2x\cos 2x$ *(raudona) ir* $y_2 = (x+1)^2$ *(mėlyna) grafikai*

2 . Izoliacijos intervalo apskaičiavimas pagal uždavinio taikomąją prasmę.
Sprendžiama lygtis $f(x)=0$ apibūdina kurį nors fizikinį reiškinį. Remdamiesi juo, galime nurodyti arba šaknų izoliacijos intervalus, arba intervalą $(\alpha;\beta)$, kuriame yra sprendžiamos lygties šaknys.

Tarkime, kad $f(x)$ yra tolydžioji funkcija. Jei $f(a)\cdot f(b)<0$ ir $f'(x)$ turi pastovų ženklą, kai $x\in[a;b]$. Tada $(a;b)$ yra šaknies izoliacijos intervalas.

Kai intervalas $(\alpha;\beta)$ yra žinomas, remiantis ką tik pateikta teorema, galima rasti šaknų izoliacijos intervalus.

Tarkime, kad žingsnis h yra mažesnis nei dviejų gretimų lygties šaknų mažiausias skirtumas. Aišku, kad bendrų tokio žingsnio apskaičiavimo metodų nėra. Tačiau dažnai, žinant lygtimi aprašyto fizikinio reiškinio savybes, galima rasti tokio žingsnio įvertį. Tada, žingsniu h „peržingsniavę“ intervalą $(\alpha;\beta)$, t.y., kiekviename intervale $(x,x+h)$ (aišku pradinė x reikšmė yra α) tikriname sąlygą: „ar $f(x)*f(x+h)\le 0$“.

Jei $f(x)*f(x+h)<0$, tai intervalas $(x,x+h)$ yra šaknies izoliacijos intervalas, o jei $f(x)*f(x+h)=0$, tai šaknis yra vienas iš intervalo $(x,x+h)$ galų (arba abu galai).

Žemiau pateikta funkcija ***interval***, apskaičiuojanti duotos lygties šaknies izoliacijos intervalus tarp nurodytų rėžių.

Pavyzdys. *Pasinaudodami funkcija „**interval**“ ir imdami žingsnį h=0.1, raskime lygties* $2x\cos 2x-(x+1)^2=0$ *šaknies izoliacijos intervalus intervale* **(***-3; 2***)**.

```
function [a,b,s]=interval(fun,alfa,beta,h);
% INTERVAL funkcija randa lygties fun(x)=0 šaknų
% izoliacijos intervalus ir šaknis intervale (alfa; beta).
% Įėjimo parametrai
% fun  - vardas funkcijos, apskaičiuojančios lygties kairiąją pusę,
% alfa, beta - intervalo galai,
% h  - šaknies izoliacijos intervalo ilgis.
% Išėjimo parametrai
% a  - šaknų izoliacijos intervalu kairieji galai,
% b  - šaknų izoliacijos intervalu dešinieji galai,
% s  - rastos šaknų reikšmės.

x=alfa:h:beta;
y=feval(fun,x);
s=x(abs(y)<= eps);
z=y(1:end-1).*y(2:end);
a=x(z < 0);
b=a+h;
```

Komandų lauke surinkę komandą

```
>> [a,b,s]=interval(inline('2*x.*cos(2*x)-(x+1).^2'),-3,2,0.1)
```

turėsime:

```
a = -2.2000  -0.8000
b = -2.1000  -0.7000
s = Empty matrix: 1-by-0
```

Tai reiškia, kad, lygtis $2x\cos 2x-(x+1)^2=0$ intervale (-3; 2) turi dvi šaknis ir jų izoliacijos intervalai yra: (-2.2; -2.1) ir (-0.8; -0.7). Tuo nesunkiai galime įsitikinti pažiūrėję į funkcijos $y=2x\cos 2x-(x+1)^2$ grafiką (3.3 pav.). Kadangi nei vienas diskretizacijos taškas nepataikė į šaknį, tai šaknų masyvas *s* yra tuščias.

3. Polinomo šaknų rėžiai. Sprendžiant fizikos, technikos ir matematikos uždavinius, dažnai tenka skaičiuoti polinomo šaknis. Nors yra sukurti specialūs polinomo šaknų ieškojimo metodai, tačiau praktikoje svarbu žinoti kurioje srityje yra polinomo šaknys.

Tarkime, kad $f(x)$ yra *n*-tojo laipsnio polinomas:

$$f(x)=a_0x^n+a_1x^{n-1}+\ldots+a_{n-1}x+a_n\quad (a_0>0). \qquad (3.1)$$

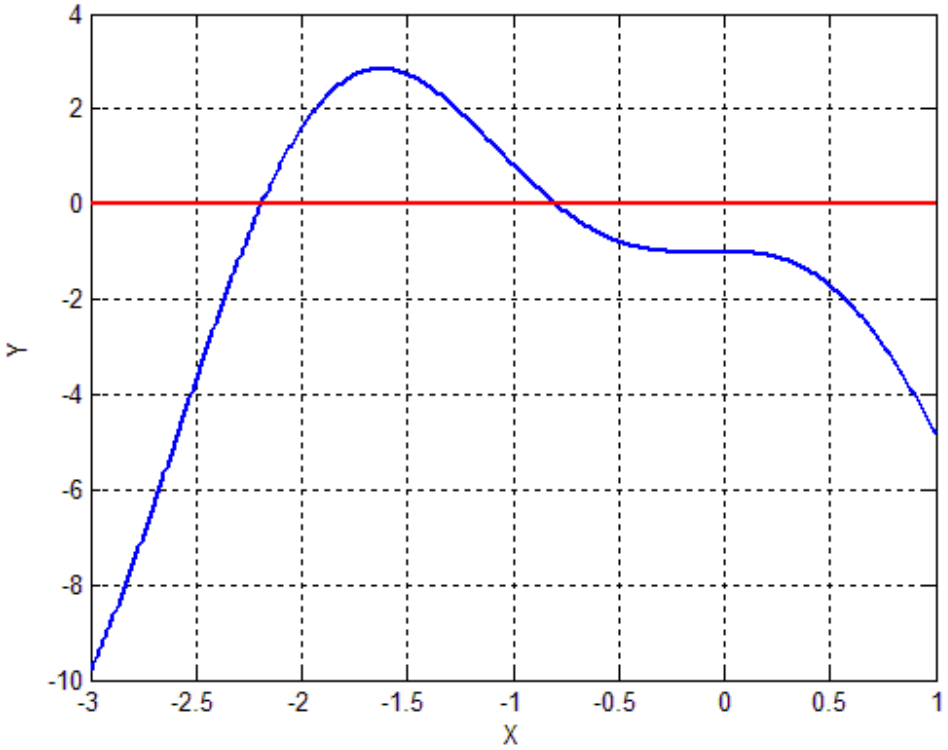
1 teorema. Jei $x\ge 1+\dfrac{A}{a_0}$ (čia $A=\max\limits_{1\le i\le n}|a_i|$), tai

$$|a_0x^n|>|a_1x^{n-1}+\ldots+a_{n-1}x+a_n|\ .$$

Teorema pagrįsta tuo, kad galima nurodyti tokią mažiausią *x* reikšmę, kad vyriausias polinomo narys visada būtų didesnis už likusių narių sumą.

1 teoremos išvada. Visos (3.1) polinomo šaknys tenkina sąlygą $|x|<1+\dfrac{A}{a_0}$.

Šią sąlygą tenkina ne vien realiosios šaknys, bet ir kompleksinių šaknų moduliai.



3.3 pav. Funkcijos $y = 2x\cos 2x - (x+1)^2$ *grafikas intervale* [*-3; 1*]

Ieškant tik realiųjų polinomo šaknų rėžių, galima remtis 3 teorema.

2 teorema. Tarkime, kad a_k yra pirmasis iš kairės neigiamas (3.1) polinomo koeficientas. Tada (3.1) polinomo teigiamųjų realiųjų šaknų viršutinis rėžis bus $R = 1 + \sqrt[k]{\frac{B}{a_0}}$; čia $B = \max_{1 \le i \le n} \left(|a_i|, a_i < 0 \right)$.

Teoremos teisingumas pagrįstas tuo, kad galima nurodyti tokią mažiausią x reikšmę, kad polinomo teigiamų narių suma visada būtų didesnė už neigiamų narių sumos modulį.

Norėdami sužinoti polinomo neigiamųjų realiųjų šaknų apatinį rėžį, turime:

1) rasti polinomo $f(-x)$ teigiamųjų realiųjų šaknų viršutinį rėžį, jei n — lyginis, arba $-f(-x)$, jei n — nelyginis;

2) gautą rėžį paimti su minuso ženklu — tai bus polinomo $f(x)$ neigiamųjų realiųjų šaknų apatinis rėžis.

Žemiau pateikta funkcija ***sakreziai***, apskaičiuojanti polinomo tikrųjų šaknų rėžius.

```
function [alfa,beta]=sakreziai(n,a);
% SAKREZIAI apskaičiuoja polinomo tikrųjų šaknų rėžius.
% Įėjimo parametrai
% n   - polinomo laipsnis,
% a   - polinomo koeficientai pradedant vyriausiuoju.
% Išėjimo parametrai
% (alfa, beta) - polinomo tikrųjų šaknų rėžiai.

% Teigiamu tikrųjų šaknų viršutinio rėžio apskaičiavimas
```

```
if a(1) < 0
    b=-a;
else
    b=a;
end;
ind=find(b<0);
if numel(ind)==0
    beta=0;
else
    k=ind(1)-1;
    d=max(abs(b(b<0)));
    beta=1+(d/b(1))^(1/k);
end;
% Neigiamų tikrųjų šaknų rėžio apskaičiavimas
b=a;
b(end-1:-2:1)=-b(end-1:-2:1);
if b(1) < 0
    b=-b;
end;
ind=find(b<0);
if numel(ind)==0
    alfa=0;
else
    k=ind(1)-1;
    d=max(abs(b(b<0)));
    alfa=-(1+(d/b(1))^(1/k));
end;
```

Pavyzdys. *Pasinaudodami funkcijomis*: ***sakreziai*** ir ***interval*** *apskaičiuokime polinomo* $f(x)=x^5+4x^4-9x^3+14x^2+50x-25$ *tikrųjų šaknų izoliacijos intervalus.*

Komandų lange surinkę komandas:

```
a=[1 4 -9 14 50 -25];
n=numel(a);
[alfa,beta]=sakreziai(n,a)
h=0.1;
[aa,bb,s]=interval(inline('x.^5+4*x.^4-9*x.^3+14*x.^2+50*x-25'),...
                   alfa,beta,h)
roots(a),
```

turėsime

```
alfa = -15
beta =  6
aa = -5.8000   -1.8000    0.4000
bb = -5.7000   -1.7000    0.5000
s =  Empty matrix: 1-by-0
ans = -5.7127
       1.5050 + 1.7949i
       1.5050 - 1.7949i
      -1.7524
       0.4551
```

Parametras *ans* nusako nagrinėjamo polinomo šaknis, apskaičiuotas su vidine MATLAB'o funkcija ***roots.***

Kaip matome, polinomas $f(x)=x^5+4x^4-9x^3+14x^2+50x-25$ turi trys realiąsias ir dvi jungtines kompleksines šaknis. Funkcija ***sakreziai***, pirmiausia apskaičiuoja polinomo

realiųjų šaknų rėžius: *alfa=-15 ir beta=6,* o toliau, funkcija ***interval*** apskaičiuoja jų intervalus: (-5.8; -5.7), (-1.8; -1.7) ir (0.4; 0.5).

3.1.2. Šaknų tikslinimas.

Uždavinio formulavimas. Duota lygtis $f(x)=0$ ir šaknies izoliacijos intervalas $(a;b)$ arba apytikslė šaknies reikšmė x_0. Reikia rasti šaknį $s \in (a;b)$ tikslumu ε.

Šį uždavinį galima spręsti keliais metodais:
- pusiaukirtos metodu,
- kirstinių metodu,
- liestinių (Niutono) metodu,
- Miulerio (parabolių) metodu
- kombinuotuoju metodu,
- paprastųjų iteracijų metodu,
- pagreitintų iteracijų metodu (Vegsteino metodu) ir kt.

1. Pusiaukirtos metodas

Metodo idėja labai paprasta. Šaknies izoliacijos intervalą $(a;b)$ dalijame pusiau. Tarkime, kad *c* yra intervalo $(a;b)$ vidurys. Tada šaknis yra arba intervale $(a;c)$, arba intervale $(c;b)$, arba taškas *c* yra norimo tikslumo lygties šaknis (šiuo atveju lygties sprendimas baigtas). Vadinasi, po kiekvienos iteracijos pradinis šaknies izoliacijos intervalas sutrumpėja dvigubai. Skaičiuoti baigiama, kai šaknies intervalo ilgis pasidaro mažesnis už ε arba $abs(f(c)) < \varepsilon$. Tada, pirmuoju atveju, bet kuri reikšmė iš šio intervalo, o antruoju atveju, reikšmė *c* yra norimo tikslumo šaknis. Pirmuoju atveju šakniai priskiriama intervalo vidurinio taško reikšmė.

Pusiaukirtos metodas visada konverguoja, jei tik *f*(*x*) intervale $(a;b)$ yra tolydžioji funkcija ir $f(a) * f(b) < 0$, tačiau konvergavimo greitis mažas. Pavyzdžiui, jei $|b-a|=1$, tai, norėdami rasti šaknį šiame intervale 10^{-6} tikslumu, iteracijų skaičių galime apskaičiuoti iš nelygybės

$$\frac{|b-a|}{2^n} < 10^{-6}.$$

Iš čia turime, kad $n = 20$.

Dėl šios priežasties pusiaukirtos metodas paprastai taikomas pradiniam šaknies izoliacijos intervalui siaurinti, paskui šaknis tikslinama greitesniu, pavyzdžiui, liestinių, metodu.

Pastaba. *Visoms iteracinėms procedūroms būtina numatyti maksimalų leistiną iteracijų skaičių. Priešingu atveju, jei iteracinis procesas nekonverguoja, kompiuteris vykdys amžintąjį ciklą.*

Pusiaukirtos algoritmas

```
function [s,it]=pskrt(func,a,b,e,itmax);
% PSKRT funkcija tikslina lygties f(x)=0 šaknį pusiaukirtos metodu.
% Įėjimo parametrai
% func  - vardas funkcijos, apskaičiuojančios lygties kairąją pusę,
% a,b   - šaknies izoliacijos intervalas,
% e     - šaknies apskaičiavimo tikslumas,
% itmax - maksimalus leistinas iteracijų skaičius
```

```
% Išėjimo parametrai
% s     - šaknies reikšme,
% it    - atliktų iteracijų skaičius.

it=0;
p=true;
pa=feval(func,a);
while (it <= itmax) & (abs(b-a) > e) & p
    c=a+(b-a)/2;% randame atkarpos (a;b) vidurio tašką
        pc=feval(func,c);
    if sign(pa)*sign(pc) > 0 % sign yra ženklo funkcija
        a=c;
        pa=pc;
    elseif abs(pc)<=e
            p=false;
            s=c;% c yra norimo tikslumo šaknis
        else
            b=c;
    end
    it=it+1;
end
    if p
        s=a+(b-a)/2;
    end
```

Pavyzdys.

Pusiaukirtos metodu 10^{-4} tikslumu raskime lygties $x^3+4x^2-10=0$ realiąją šaknį s, kai žinomas jos izoliacijos intervalas (1;2).

Skaičiavimo rezultatai patalpinti 3.1 lentelėje.

Kaip matome iš skaičiavimo rezultatų, kad nors intervalo (*a,b*) ilgis yra didesnis už 10^{-4}, bet $abs(f(c))< 10^{-4}$. Todėl norimo tikslumo šaknimi galima laikyti reikšmę *c.*

2. Kirstinių metodas

Tikslindami lygties $f(x)=0$ šaknį pusiaukirtos metodu, neatsižvelgiame į funkcijos *f(x)* kitimo pobūdį. Tarkime, kad $|f(a)|<|f(b)|$. Tada galime tikėtis, kad lygties šaknis yra arčiau taško *a* negu taško *b.* Vadinasi, geresnis taško *c* pasirinkimas būtų ne atkarpos (*a;b*) vidurio taškas, bet taškas, kuriame tiesė (kirstinė), einanti per taškus $(a, f(a))$ ir $(b, f(b))$, kerta *X*-o ašį. Toks šaknies tikslinimo metodas vadinamas kirstinių metodu.

Kirstinių metodas būtų aprašomas tokiu būdu.

Tarkime, kad x_i ir x_{i+1} yra du gretimi šaknies artiniai. Tada nauja patikslinta artinio reikšmė apskaičiuojama pagal formulę

$$x_{i+2}=x_{i+1}-\frac{f(x_{i+1})(x_{i+1}-x_i)}{f(x_{i+1})-f(x_i)}.$$

Aišku, kad kiekviename žingsnyje kirstinė vedama remiantis dviem paskutiniais šaknies artiniais. Šaknies tikslinimą baigiame, pagal įprastas iteracinio proceso pabaigos sąlygas, t.y. 1) $|x_{i+2}-x_{i+1}|<\varepsilon$ arba 2) $|f(x_{i+2})|<\varepsilon$.

Kirstinių metodas, skirtingai nuo pusiaukirtos metodo, neturi šaknies apskliaudimo savybės. Vadinasi, kirstinių metodas ne visada konverguoja. Tačiau konvergavimo atveju jis konverguoja greičiau nei pusiaukirtos metodas.

3.1 lentelė. Lygties $x^3 + 4x^2 - 10 = 0$ *šaknies apskaičiavimas pusiaukirtos metodu*

it	*a*	*b*	*c*	*f(c)*
1	1	2	1.5	2.375
2	1	1.5	1.25	-1.796875
3	1.25	1.5	1.375	0.162109375
4	1.25	1.375	1.3125	-0.8483886719
5	1.3125	1.375	1.34375	-0.3509826660
6	1.34375	1.375	1.359375	-0.0964088440
7	1.359375	1.375	1.3671875	0.0323557854
8	1.359375	1.3671875	1.36328125	-0.0321499705
9	1.36328125	1.3671875	1.365234375	0.0000720248

Dažnai vietoje kirstinių metodo naudojamas pusiaukirtos ir kirstinių metodų hibridas – ***stygų metodas(method of False Position).*** Tai atskiras kirstinių metodo atvejis, kai yra fiksuotas (nejudantis) vienas kirstinės taškas. Šis metodas, jei žinomas šaknies izoliacijos intervalas, visada konverguoja, tačiau jo konvergavimo greitis yra lėtesnis nei kirstinių metodo konvergavimo greitis.

Stygų metodas. Funkciją $y = f(x)$ intervale $(a;b)$ keičiame styga, einančia per taškus $(a;f(a))$ ir $(b;f(b))$. Randame tos stygos ir X ašies sankirtos tašką c. Nustatome, ar lygties šaknis yra intervale $(c;b)$ (šiuo atveju $a=c$), ar intervale $(a;c)$ (šiuo atveju $b = c$), ar ji yra taškas c (šiuo atveju sprendimas baigtas). (žr. 3.4 pav.).

Po šio žingsnio turime patikslintą šaknies intervalą $(a;b)$ ir skaičiavimo procesą kartojame iš naujo.

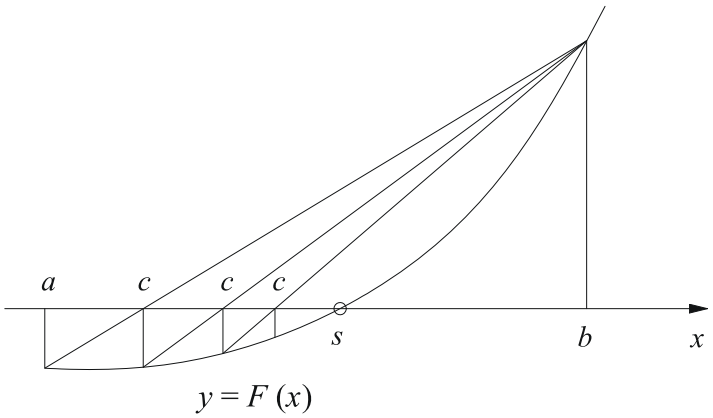
Skaičiavimo pabaigos sąlyga. Skaičiavimą baigiame, kai arba 1) $|f(c)| < \varepsilon$, arba 2) $\left|\frac{c - c1}{c}\right| < \varepsilon$ (čia c ir $c1$ — dvi gretimos c reikšmės), arba 3) $\max\left(|f(c)|, \left|\frac{c - c1}{c}\right|\right) < \varepsilon$.

Tada šaknies s reikšmė tikslumu ε yra paskutinė c reikšmė.

Reikia pasakyti, kad galime rasti „patologinių“ lygčių, kurioms nurodytos pabaigos sąlygos netinka.

Taškas, kuriame styga kerta X ašį, dalija intervalą $(a;b)$ santykiu $\lambda = -\frac{F(a)}{F(b)}$, todėl c reikšmę patogu skaičiuoti pagal formulę $c = \frac{a + \lambda \cdot b}{1 + \lambda}$.

Kadangi stygų metodo algoritmas iš esmės sutampa su pusiaukirtos metodo algoritmu, todėl čia jo detaliau nenagrinėsime.



3.4 pav. Stygų metodas

3. Miulerio (parabolių) metodas

Natūralus kirstinių metodo tęsinys yra Miulerio arba parabolių metodas. Miulerio metodas remiasi tuo, kad šaknies izoliacijos intervale (šis reikalavimas nėra būtinas) funkcija $y = f(x)$ aproksimuojama kvadratine parabole, einančia per tris tos funkcijos taškus. Vienas parabolės sankirtos su *X* ašimi taškas laikomas apytiksle šaknies reikšme. Tarkime, kad x_0, x_1 ir x_2 yra trys iš eilės einančios lygties $f(x) = 0$ apytikslės šaknies reikšmės. Tada funkciją $y = f(x)$ aproksimuokime kvadratine parabole

$$P(x) = a(x - x_2)^2 + b(x - x_2) + c,$$

einančia per taškus $(x_0;\ f(x_0))$, $(x_1;\ f(x_1))$ bei $(x_2;\ f(x_2))$, ir raskime taškus, kuriuose ta parabolė kerta *X* ašį.

Suprastinkime simboliką: $f(x_i)$ toliau žymėsime y_i, čia $i = 0, 1, 2$.

Parabolės $P(x)$ konstantas *a*, *b* ir *c* apskaičiuosime iš sąlygų:

$$y_0 = a(x_0 - x_2)^2 + b(x_0 - x_2) + c,$$

$$y_1 = a(x_1 - x_2)^2 + b(x_1 - x_2) + c,$$

$$y_2 = c.$$

Aišku, kad parabolė $P(x)$ kirs *X* ašį taškuose x_3, kurie nustatomi pagal formulę

$$x_3 = x_2 - \frac{2c}{b \pm \sqrt{b^2 - 4ac}}.$$

Iš šių dviejų x_3 reikšmių pasirinksime tą, kuri artimesnė reikšmei x_2.

Vadinasi, ***Miulerio metodas*** apibūdinamas taip:

1) turint reikšmes x_0, x_1 ir x_2, reikšmė x_3 apskaičiuojama pagal formulę

$$x_3 = x_2 - \frac{2c}{b + sign(b)\sqrt{b^2 - 4ac}};$$

čia

$$c = y_2,$$

$$b=\frac{(x_0-x_2)^2(y_1-y_2)-(x_1-x_2)^2(y_0-y_2)}{(x_0-x_1)(x_0-x_2)(x_1-x_2)},$$

$$a=\frac{(x_1-x_2)(y_0-y_2)-(x_0-x_2)(y_1-y_2)}{(x_0-x_1)(x_0-x_2)(x_1-x_2)}.$$

2) reikšmės pernumeruojamos, t. y. $x_0 := x_1$, $x_1 := x_2$ ir $x_2 := x_3$.

Skaičiavimo pabaigos sąlyga. Skaičiuoti pagal 1 ir 2 punktą baigiama tada, kai $|f(x_3)| < \varepsilon$. Tuomet x_3 yra apytikslė lygties $f(x)=0$ šaknies reikšmė.

Kadangi, apskaičiuojant reikšmę x_3, reikia rasti $\sqrt{b^2-4ac}$, tai Miulerio metodas tinka kompleksinėms šaknims tikslinti. Aišku, kad šiuo atveju taikoma kompleksinių skaičių aritmetika, kuri labai patogiai realizuota MATLAB‘o terpėje.

Kita labai svarbi Miulerio metodo savybė yra ta, kad šis metodas paprastai konverguoja su bet kokiomis pradinėmis x_0, x_1 ir x_2 reikšmėmis. Tačiau svarbu parinkti tokias x_0, x_1 ir x_2 reikšmes, kad Miulerio metodas konverguotų prie norimos šaknies reikšmės.

Miulerio metodui būdingas didelis konvergavimo greitis net ir tuomet, kai reikšmės x_0, x_1 ir x_2 nėra artimos lygties šakniai. Kaip nurodyta literatūroje, (i + 1)-osios iteracijos paklaida apytiksliai proporcinga i-tosios iteracijos paklaidos 1,85 laipsniui, taigi Miulerio metodas turi beveik kvadratinį konvergavimo greitį. Tą galime pastebėti iš toliau pateikto pavyzdžio.

Miulerio metodo funkcija polinomams.

```
function [s,it]=miuler(n,a,x0,x1,x2,e,itmax);
% MIULER apskaičiuoja polinomo šaknį Miulerio metodu
% Įėjimo parametrai
% n         - polinomo eile,
% a         - polinomo koeficientai pradedant vyriausiuoju,
% e         - šaknies apskaičiavimo tikslumas,
% x0,x1,x2  - pradiniu taškų reikšmės,
% itmax     - maksimalus leistinas iteracijų skaičius.
% Išėjimo parametrai
% s         - šaknies reikšmė,
% it        - atliktų iteracijų skaičius.

it=0;
fx2=horn(n,a,x2);%horn - Hornerio schemos funkcija (žr.6.1 paragrafą)
while (it <= itmax) & (abs(fx2) > e)
    fx0=horn(n,a,x0); fx1=horn(n,a,x1);
  % Kvadratinio interpoliacinio polinomo koeficientų apskaičiavimas
    c=fx2;
    b=((x0-x2)^2*(fx1-fx2)-(x1-x2)^2*(fx0-fx2))/...
       ((x0-x2)*(x1-x2)*(x0-x1));
    aa=((x1-x2)*(fx0-fx2)-(x0-x2)*(fx1-fx2))/...
       ((x0-x2)*(x1-x2)*(x0-x1));
 % Vardiklio apskaičiavimas
    tp=b+sqrt(b^2-4*aa*c); tm=b-sqrt(b^2-4*aa*c); v=tp;
    if abs(tm) > abs(tp)
        v=tm;
    end;
% Kvadratines lygties sprendinio apskaičiavimas
```

```
    x3=x2-2*c /v;
% Taškų perstumimas
    x0=x1; x1=x2; x2=x3;
    fx2=horn(n,a,x2); it=it+1;
end
    if it <= itmax
        s=x2;
    else
        s=nan;
    end
```

Pavyzdys. Apskaičiuokime polinomo $f(x)=16x^4-40x^3+5x^2+20x+6$ šaknis 10^{-5} tikslumu. Šio polinomo šaknų reikšmės nurodytu tikslumu yra 1,241677, 1,970446 ir $-0{,}356062\pm 0{,}162758i$.

Šaknų tikslinimas Miulerio metodu pateiktas 3.2, 3.3 ir 3.4 lentelėje.

3.2 lentelė. Polinomo $f(x)=16x^4-40x^3+5x^2+20x+6$ kompleksinių šaknų tikslinimas, kai $x_0=0{,}5,\ x_1=-0{,}5,\ x_2=0$

i	x_i	$F(x_i)$
3	–0,555556 + 0,598352i	–29,4007 – 3,89872i
4	–0,435450 + 0,102101i	1,33223 – 1,19309i
5	–0,390631 + 0,141852i	0,375057 – 0,670164i
6	–0,357699 + 0,169926i	–0,146746 – 0,00744629i
7	–0,356051 + 0,162856i	$-0{,}183868\cdot10^{-2}+0{,}539780\cdot10^{-3}i$
8	–0,356062 + 0,162758i	$0{,}286102\cdot10^{-5}+0{,}953674\cdot10^{-6}i$

3.3 lentelė. Polinomo $f(x)=16x^4-40x^3+5x^2+20x+6$ pirmosios realiosios šaknies tikslinimas, kai $x_0=2{,}5,\ x_1=2{,}0,\ x_2=2{,}25$

i	x_i	$F(x_i)$
3	1,96059	$-0{,}611255$
4	1,97056	$0{,}748825\cdot10^{-2}$
5	1,97044	$-0{,}295639\cdot10^{-4}$
6	1,97044	$-0{,}295639\cdot10^{-4}$

3.4 lentelė. Polinomo $f(x)=16x^4-40x^3+5x^2+20x+6$ antrosios realiosios šaknies tikslinimas, kai $x_0=0{,}5,\ x_1=1{,}0,\ x_2=1{,}5$

i	x_i	$F(x_i)$
3	1,28785	–1,37624
4	1,23746	0,126941
5	1,24160	$0{,}219440\cdot10^{-2}$
6	1,24168	$0{,}257492\cdot10^{-4}$
7	1,24168	$0{,}257492\cdot10^{-4}$

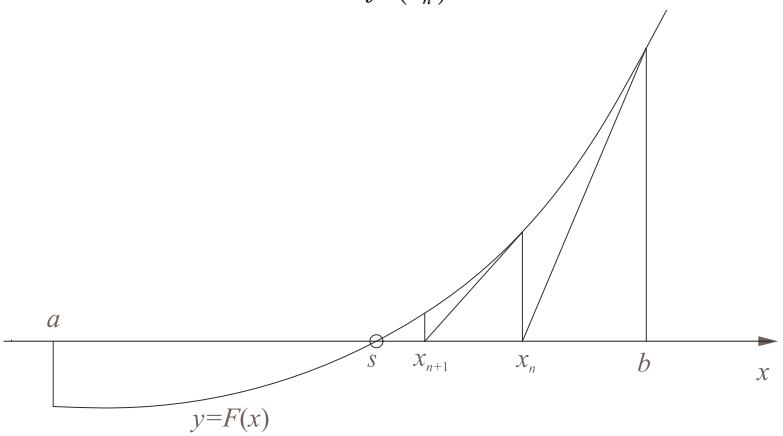
4. Liestinių (Niutono) metodas

Šis metodas literatūroje dažnai vadinamas Niutono arba Niutono ir Rafsono metodu. Tikslinant šaknį liestinių metodu, funkcija $y = f(x)$ intervale $(a;b)$ keičiama liestine, nubrėžta iš taško $(x_n; f(x_n))$.

Tarkime x_n yra šaknies artinys, priklausantis intervalui $(a;b)$. Tada randamas liestinės, einančios per tašką $(x_n, f(x_n))$, ir *X* ašies sankirtos taškas. Liestinės lygtis yra

$y - f(x_n) = f'(x_n)(x - x_n)$, todėl liestinės sankirtos su *X* ašimi taško abscisė apskaičiuojama pagal formulę:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$



3.5 pav. Liestinių metodas

Jei skaičiavimo pabaigos sąlyga netenkinama, tai x_n suteikiama x_{n+1} reikšmė ir skaičiuojama toliau. Priešingu atveju $s = x_{n+1}$.

Pradinis artinys. Pradinė x_n reikšmė, kai žinomas šaknies izoliacijos intervalas $(a;b)$, apskaičiuojama pagal formulę: jei $f(a) \cdot f''(a) > 0$, tai $x_n = a$, priešingu atveju $x_n = b$. 3.5 paveiksle pavaizduota funkcija, kurios $f(b) \cdot f''(b) > 0$.

Kaip galima sužinoti pradinę x_n reikšmę, neskaičiuojant $f(x)$ antrosios išvestinės? Tam tikslui raskime stygos, einančios per taškus $(a; f(a))$ ir $(b; f(b))$, sankirtos su *X* ašimi taško abscisę: $c = \frac{a + san \cdot b}{1 + san}$; čia $san = \quad = -\frac{f(a)}{f(b)}$. Tada jei $f(a) \cdot f(c) > 0$, tai $x_n = b$, priešingu atveju $x_n = a$.

Jei lygtį sprendžiame MATLAB'o terpėje, tai pradinį šaknies artinį patogu nustatyti grafiškai, naudojant funkciją ***plotap2*** (žr. 3.1.1 paragrafą).

Skaičiavimo pabaigos sąlyga. Liestinių metodu, kaip ir kirstinių metodu, skaičiuoti baigiama tada, kai arba 1) $|f(x_{n+1})| < \varepsilon$, arba 2) $\left|\frac{x_{n+1}-x_n}{x_{n+1}}\right| < \varepsilon$, arba 3) $\max\left(|f(x_{n+1})|, \left|\frac{x_n - x_{n+1}}{x_{n+1}}\right|\right) < \varepsilon$.

Žemiau pateikta funkcija ***liest***, tikslinanti lygties šaknį liestinių metodu, o taip pat lygties $2x\cos 2x - (x+1)^2 = 0$ mažesniosios šaknies tikslinimas, kai pradinis šaknies artinys nustatomas funkcijos ***plotap2*** pagalba.

```
function [s,it]=liest(func,funci,x,e,itmax);
% LIEST funkcija tikslina šaknį liestinių metodu.
% Įėjimo parametrai
% func - vardas funkcijos, apskaičiuojančios lygties kairiąją pusę,
% funci- vardas funkcijos, apskaičiuojančios lygties kairiosios
%         puses išvestinę,
 % x   - šaknies apytikslė reikšmė,
% e   - šaknies apskaičiavimo tikslumas,
% itmax - maksimalus leistinas iteracijų skaičius.
% Išėjimo parametrai
% s     - šaknies reikšmė,
% it    - atliktu iteracijų skaičius.

it=0; xn=x;
fxn=feval(func,xn);
while (it <= itmax) & (abs(fxn) > e)
    xn1=xn-fxn/feval(funci,xn);
    xn=xn1;
    fxn=feval(func,xn);
    it=it+1;
end
    if it <= itmax
        s=xn1;
    else
        fprintf('atliktų iteracijų skaičius yra: %8.5f\n',it);
        fprintf('norimas tikslumas nepasiektas');
        s=xn1;
    end
```

Lygties $2x\cos 2x - (x+1)^2 = 0$ ***sprendimo rezultatai***

```
x0 = -2.18605990783410% pradinis artinys

x1 = -2.19132924020914 % patikslinta reikšmė
x2 = -2.19130801213979 % patikslinta reikšmė
x3 = -2.19130801179725 % patikslinta reikšmė
s  = -2.191308011797247 % apskaičiuota šaknies reikšmė
it = 3 % atliktų iteracijų skaičius
f(s) = 1.332267629550188e-015 % netiktis
```

Niutono metodo konvergavimo greitis. Lygtys dažnai sprendžiamos Niutono metodu, nes jis greitai konverguoja. Konvergavimo greičio klausimą aptarkime plačiau.

Sakykime, iteracijų metodas, nusakomas formule $x_{n+1} = g(x_n)$, generuoja skaičių $x_0, x_1, x_2, \ldots, x_n, \ldots$ seką, kurią žymėsime $\{x_n\}$.

Apibrėžimas. Tarkime, kad $a \in \boldsymbol{R}$, $x_n \in \boldsymbol{R}$, $n = 0, 1, \ldots$. Sakome: seka $\{x_n\}$ konverguoja prie a, jei $\lim\limits_{n\to\infty} |x_n - a| = 0$.

Apibrėžimas. Jei egzistuoja konstanta $c \in [0; 1)$ ir sveikasis $k \geq 0$, kad su visais $n \geq k$ galioja nelygybė $|x_{n+1} - a| \leq c\,|x_n - a|$, tai sakoma, kad seka $\{x_n\}$ konverguoja prie a tiesiškai.

Apibrėžimas. Jei egzistuoja seka $\{c_n\}$, konverguojanti prie 0, ir galioja nelygybė $|x_{n+1} - a| \leq c_n\,|x_n - a|$, tai sakoma, kad seka $\{x_n\}$ konverguoja prie a greičiau nei tiesiškai.

Apibrėžimas. Jei $\{x_n\}$ konverguoja prie a ir egzistuoja tokie $p > 1$, $c \geq 0$, $k \geq 0$, kad nelygybė

$$|x_{n+1} - a| \leq c\,|x_n - a|^p$$

galioja su visais $n \geq k$, tai sakoma, kad seka $\{x_n\}$ konverguoja prie a eile, ne mažesne nei p. Kai $p = 2$, sakoma, kad konvergavimo greitis yra kvadratinis.

Kaip nustatyti konvergavimo greitį, kai žinoma iteracinį procesą aprašanti formulė?

Tarkime iteracinis procesas, aprašomas formule $x_{n+1} = g(x_n)$, generuoja seką: $x_0, x_1, \cdots, x_n, \cdots$, ir šios sekos nariai konverguoja prie reikšmės a, t.y. $\lim\limits_{n\to\infty} |x_n - a| = 0$.

Pažymėkime $\varepsilon_n = |x_n - a|$.

4 teorema. Tarkime, kad funkcijos $g(x)$ visos išvestinės iki $(p-1)$-osios eilės imtinai taške $x = a$ yra lygios nuliui. Tada

$$\varepsilon_{n+1} = (\varepsilon_n)^p \frac{g^{(p)}(t)}{p!}, \text{ čia } t \in [x_n; a].$$

Remdamiesi 4 teorema, apskaičiuosime liestinių metodo konvergavimo greitį.

Liestinių metodas aprašomas formule:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Vadinasi, $g(x) = x - \dfrac{f(x)}{f'(x)}$.

Apskaičiuokime $g(x)$ išvestinę:

$$g'(x) = 1 - \frac{(f'(x))^2 - f(x)f''(x)}{(f'(x))^2} = \frac{f(x)f''(x)}{(f'(x))^2}.$$

Jei a yra lygties $f(x) = 0$ šaknis, tai matome, kad $g(a) = 0$ ir $g'(a) = 0$. Tačiau, jei užrašytume $g(x)$ antrąją išvestinę ir apskaičiuotume jos reikšmę kai $x = a$, tai turėtume, kad $g''(a) \neq 0$.

Vadinasi, remdamiesi 4-ta teorema, galima parašyti

$$|x_{n+1} - a| = \frac{|g''(t)|}{2}|x_n - a|^2 \text{, čia } t \in [x_n; a].$$

Tai reiškia, kad liestinių metodas turi kvadratinį konvergavimo greitį.

Taikymo ypatumai.

1. Nors liestinių (Niutono) metodas turi kvadratinį konvergavimo greitį, tačiau, jei lygtis $f(x) = 0$ turi kartotinę šaknį, tai, ją tikslinant liestinių metodu, konvergavimo greitis yra tiesinis. Norint pagreitinti skaičiavimą, galimi du būdai.

- Taikyti Šrioderio metodą, kuris aprašomas tokia iteracine formule:

$$x_{n+1} = x_n - m\frac{f(x_n)}{f'(x_n)},$$

čia *m* – šaknies kartotinumas.

- Niutono metodu spręsti lygtį: $\varphi(x) = \frac{f(x_n)}{f'(x_n)} = 0$. Ši lygtis turi tą pačią šaknį, kaip ir lygtis $f(x) = 0$, tačiau ji nėra kartotinė . Kitaip tariant, taikyti iteracinę formulę: $x_{n+1} = x_n - \frac{\varphi(x_n)}{\phi'(x_n)}$. Aišku, kad šis būdas yra geresnis nei pirmasis, nes nereikia žinoti šaknies kartotinumo.

2. Pagrindinis Niutono metodo trūkumas yra tas, kad reikia nurodyti pradinę šaknies reikšmę, artimą tiksliajai reikšmei. Jei pradinis šaknies artinys nėra tikslus, tai metodas dažnai diverguoja arba generuoja ciklišKai pasikartojančias reikšmes. Jei, pavyzdžiui, spręstume lygtį $arctg(x) = 0$ ir šaknies artinys $x_0 \in [1{,}35; 1{,}40]$, tai Niutono metodas generuotų ciklišKai pasikarojančias artinių reikšmes.

Gali pasitaikyti lygčių, kurioms $f'(x_n)$ yra artima nuliui. Šiuo atveju metodas gali nekonverguoti arba gali būti apskaičiuota kita šaknis.

Dėl šių priežasčių Niutono metodas paprastai taikomas kartu su pusiaukirtos arba stygų metodu. Pirmiausia pusiaukirtos arba stygų metodu susiaurinamas šaknies izoliacijos intervalas, o po to šaknis tikslinama liestinių metodu.

5. *Funkcija fzero*

MATLAB'o funkcija ***fzero*** yra skirta spręsti lygtį $f(x) = 0$, kai žinomas šaknies izoliacijos intervalas arba šaknies artinys. Ši funkcija realizuoja hibridinį metodą, kurį sudaro pusiaukirtos, parabolių ir kirstinių metodas.

Jei žinomas tik šaknies artinys, tai pirmiausia apskaičiuojamas šaknies izoliacijas intervalas, kuriam priklauso nurodytas artinys. Jei tokio intervalo nepavyksta rasti, tai šakniai suteikiama ***nan*** reikšmė ir skaičiavimas baigiamas. Priešingu atveju, tarkime, kad žinome tris funkcijos $y = f(x)$ taškus: $(x_0; f(x_0))$, $(x_1; f(x_1))$ ir $(x_2; f(x_2))$. Taikydami parabolių metodą, apskaičiuojame tašką x_3 ir procesą tęsiame toliau imdami taškus: $(x_1; f(x_1))$, $(x_2; f(x_2))$ ir $(x_3; f(x_3))$.

Jei $x_0 \approx x_1$ arba $x_1 \approx x_2$, tai vietoje parabolių metodo taikome kirstinių metodą.

Jei parabolių arba kirstinių metodu apskaičiuotas šaknies artinys nepriklauso šaknies izoliacijos intervalui, tai taikomas pusiaukirtos metodas.

Aišku, kad algoritmas yra sudėtingesnis: organizuotas taip, kad skaičiavimai nesikartotų ir apvalinimo paklaidos būtų galimai mažiausios.

Kreipinio į funkciją ***fzero*** sintaksė gali keistis priklausomai nuo MATLAB'o versijos. Norėdami gauti išsamesnę informaciją apie šią funkciją, surinkite komandą: ***help fzero.*** Aptarkime du dažniausiai naudojamus kreipinius.

```
s=fzero(fun,x0);
s=fzero(fun,x0,options);
```

Čia ***fun*** yra vardas funkcijos (m-failo), apskaičiuojančios funkcijos $y = f(x)$ reikšmę, o ***x0*** – šaknies izoliacijos intervalas (dviejų komponenčių vektorius) arba pradinis šaknies artinys; ***s*** – tikslumu ***eps*** ($eps \approx 2.2 * 10^{-16}$) apskaičiuota šaknies reikšmė. Jei šaknies apskaičiuoti nepavyko, tai ***s*** įgauna reikšmę ***nan***.

Parametras ***options*** yra speciali duomenų struktūra, kurią sukuria funkcija ***optimset***. Ši funkcija yra paslaugų priemonė, leidžianti nusakyti parametrus įvairioms MATLAB'o optimizavimo funkcijoms (pvz., ***fminbnd, fzero, fsolve, fminsearch*** ir kt.). Plačiau apie funkciją ***optimset*** galima sužinoti surinkus komandą: ***help optimset.*** Čia aptarsime šios funkcijos parametrus, kurie valdo funkcijos ***fzero*** darbą.

Funkcijos ***optimset*** sintaksė yra

```
options = optimset('parametro1Vardas', parametro1reikšmė,
'parametro2Vardas', parametro2reikšmė ,…),
```

čia `parametro1Vardas` yra vardas parametro, kuriam priskiriama reikšmė `parametro1reikšmė` ir t.t.

Funkcijos ***fzero*** darbą valdo parametrai: „***TolX***" ir „***Display***". Pirmasis parametras nurodo šaknies tikslumą, o antrasis – kokia informacija bus išvedama į ekraną.

Parametro ***TolX*** reikšmė yra skaičius. Pagal nutylėjimą ***TolX*** = ***eps.***

Parametro ***Display*** reikšmės yra simbolių eilutės ir gali būti: „***iter***", "***off***" arba „***final***". Pagal nutylėjimą ***Display*** reikšmė yra ***final.***

Jei ***Display*** reikšmė yra ***iter***, tai spausdinami kiekvienos iteracijos rezultatai; jei reikšmė yra ***off***, nėra spausdinamas joks rezultatas, o jei reikšmė yra ***final***, tai atspausdinamas pradinis artinys ir suskaičiuota šaknies reikšmė.

1-as pavyzdys. *Raskime lygties* $2x\cos 2x - (x+1)^2 = 0$ šaknį, kai žinomas jos artinys $x0 = -0.77592$.

Surinkę komandas:

```
options=optimset('Display','iter');
[root,froot]=fzero(inline('2*x.*cos(2*x)-(x+1).^2'),-0.77592,options)
```

turėsime:

```
Search for an interval around -0.77592 containing a sign change:
   Func-count    left        f(left)           right      f(right)
Procedure
      1        -0.77592    -0.0796273       -0.77592    -0.0796273
initial interval
      3       -0.753974      -0.15524      -0.797866   -0.00107042
search
      5       -0.744883      -0.185668      -0.806957     0.0323003
search
```

```
Search for a zero in the interval [-0.74488, -0.80696]:
 Func-count      x          f(x)         Procedure
     5       -0.806957     0.0323003        initial
     6       -0.797758   -0.00146434        interpolation
     7       -0.798157 -1.01744e-005        interpolation
     8        -0.79816  6.71889e-011        interpolation
     9        -0.79816  4.85723e-017        interpolation
    10        -0.79816  4.85723e-017        interpolation
Zero found in the interval [-0.744883, -0.806957]
(Nulis rastas intervale)
root = (šaknis)
  -0.79815996140580;
```

čia žodis „interpoliation“ reiškia, kad buvo taikyta parabolių metodo procedūra.

2-as pavyzdys. *Raskime lygties* $\sin x = 0$ *šaknį, kai žinomas jos izoliacijos intervalas:* $[pi/4; 3*pi/2]$.

Surinkę komandas:

```
options=optimset('Display','iter');
[root,froot]=fzero('sin',[pi/4 3*pi/2],options)
```

turėsime:

```
Func-count      x          f(x)         Procedure
     2        0.785398     0.707107        initial
     3         2.41201     0.666558        interpolation
     4          3.5622    -0.408315        bisection
     5         3.12527     0.0163175        interpolation
     6         3.14206  -0.000471667        interpolation
     7         3.14159  2.03284e-008        interpolation
     8         3.14159 -7.65714e-016        interpolation
     9         3.14159  5.66554e-016        interpolation

Zero found in the interval [0.785398, 4.71239]
(Nulis rastas intervale)
root = (šaknis)
   3.14159265358979
froot = (netiktis)
    5.665538897647980e-016;
```

čia žodis „bisection“ reiškia, kad buvo taikytas pusiaukirtos metodas.

6. Polinomo šaknys

Svarbią vietą tarp fizikos ir technikos uždavinių užima polinomo šaknų ieškojimo uždavinys, kuris formuluojamas taip: reikia rasti lygties

$$f(x) = a_0 x^n + a_1 x^{n-1} + \ldots + a_{n-1} x + a_n = 0$$

šaknis.

Nagrinėsime polinomus, kurių koeficientai yra realieji skaičiai, nes praktikoje su tokiais polinomais susiduriama dažniausiai.

Ieškant polinomo šaknų, naudinga žinoti kai kurias tų šaknų savybes.

1. *n*-tojo laipsnio polinomas kompleksinių skaičių lauke turi *n* šaknų.

2. Jei polinomo koeficientai a_i yra realieji skaičiai, tai kompleksinės šaknys sudaro poras.

Polinomo šaknims ieškoti tinka visi anksčiau nagrinėti lygčių sprendimo metodai. Tačiau, remiantis uždavinio specifika, yra sukurta eilė specialų metodų, skirtų tik

polinomo šaknims ieškoti. Tai Lobačevskio ir Grefės, Berstou, Bernulio, QD, QR ir kt. Čia panagrinėsime liestinių (Niutono) metodą, pritaikytą tikslinti polinomo šaknis, o taip pat MATLAB'o funkciją ***roots***. Ši funkcija polinomo šaknų ieškojimo uždavinį redukuoja į matricos tikrinių reikšmių radimo uždavinį ir yra skirta rasti visas polinomo šaknis.

6.1. Hornerio schema

Tikslinant šaknį Niutono metodu, kiekvienoje iteracijoje reikia apskaičiuoti polinomo ir jo išvestinės reikšmes. Šias reikšmes patogu apskaičiuoti taikant Hornerio schemą. Pirmiausia aptarkime, kaip apskaičiuojama polinomo reikšmė.

Uždavinio formulavimas. *Duotas polinomas*

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0 .$$

Reikia rasti jo reikšmę, kai $x = x_0$.

Polinomą $f(x)$ padalijus iš $x - x_0$, gaunamas dalmuo

$$p(x) = b_n x^{n-1} + b_{n-1} x^{n-2} + ... + b_2 x + b_1$$

ir liekana $r = b_0$, todėl

$$f(x) = (x - x_0) \cdot p(x) + r . \qquad (3.2)$$

Iš čia matyti, kad $f(x_0) = r$, taigi polinomo reikšmė, kai $x = x_0$, lygi liekanai, gautai polinomą dalijant iš $x - x_0$ (Bezu teorema).

Sulyginę (6.1.1) formulės koeficientus prie tų pačių x laipsnių ir išreiškę koeficientus $b_k \left(k = \overline{n, 0}\right)$, turime:

$$b_n = a_n ,$$
$$b_k = a_k + b_{k+1} \cdot x_0 , \; k = \overline{n-1, 0} . \qquad (3.3)$$

Šios formulės vadinamos ***Hornerio schema***. Remdamiesi jomis, galime apskaičiuoti ir $f'(x_0)$.

Iš (3.2) lygybės išplaukia, kad

$$f'(x) = p(x) + (x - x_0) \cdot p'(x).$$

Tada $f'(x_0) = p(x_0)$. Vadinasi, $f'(x_0)$ lygi polinomo $p(x)$ reikšmei, kai $x = x_0$. Šią polinomo reikšmę apskaičiuosime taikydami Hornerio schemą:

$$p(x) = (x - x_0)(c_n x^{n-2} + c_{n-1} x^{n-3} + ... + c_3 x + c_2) + c_1 ;$$

čia $c_n = b_n$,

$$c_k = b_k + c_{k+1} \cdot x_0 , \; k = \overline{n-1, 1} , \qquad (3.4)$$
$$p(x_0) = f'(x_0) = c_1 .$$

Pagal (3.3) ir (3.4) formules polinomo ir jo išvestinės reikšmes patogu skaičiuoti kartu.

Pavyzdys. Duotas polinomas $f(x) = 3x^5 - 2x^2 + 5x - 1$. Apskaičiuokime $f(2)$ ir $f'(2)$.

3.2 lentelėje eilutė *a* yra polinomo koeficientų masyvas, o eilutės *b* ir *c* rodo, kaip skaičiuojant keičiasi *b* ir *c* reikšmės.

Taigi $f(2) = 97$, o $f'(2) = 237$.

3.2 lentelė. Hornerio schema polinomo $3x^5 - 2x^2 + 5x - 1$ *reikšmei apskaičiuoti*

a		3	0	0	–2	5	–1
b	0	3	6	12	22	49	97
c	0	3	12	36	94	237	

Algoritmas polinomo ir jo išvestinės reikšmėms apskaičiuoti

```
function [b,c]=hs(a,x);
% HS pagal Hornerio schemą apskaičiuoja polinomo ir jo
% išvestinių reikšmes, kai argumento reikšmės
% patalpintos masyve "x".
% Įėjimo parametrai:
% n  - polinomo eile,
% a  - polinomo koeficientu masyvas, pradedant vyriausio
%      nario koeficientu,
% x  - argumento reikšmių masyvas.
% Išėjimo parametrai
% b  -  polinomo reikšmės,
% c  -  polinomo pirmos eiles išvestinės reikšmės.

    n=numel(a)-1; m=numel(x);
    b=zeros(1,m); c=zeros(1,m);
       for k=1:n
        b=a(k)+b.*x; c=b+c.*x;
       end;
    b=a(n+1)+b.*x;
```

6.2. Niutono metodas polinomams

Atsižvelgdami į polinomų specifiką, modifikuosime 4-me paragrafe išnagrinėtą Niutono (liestinių) metodą. Matėme, kad tiek polinomo, tiek ir jo išvestines reikšmes patogiai galime apskaičiuoti Hornerio schemos pagalba, remiantis tik polinomo koeficientais. Vadinasi, anksčiau pateiktą Niutono metodo algoritmą galima pakeisti taip, kad nereikėtų įėjimo parametrų **`func`** ir **`funci`**.

Žemiau pateikiama funkcija ***lstpol***, realizuojanti Niutono metodą polinomams.

```
function [s,it]=lstpol(a,x,e,itmax);
% LSTPOL apskaičiuoja polinomo šaknį Niutono(liestinių) metodu.
% Įėjimo parametrai:
% a     - polinomo koeficientai pradedant vyriausiuoju,
% x     - apytikslė šaknies reikšmė,
% e     - šaknies apskaičiavimo tikslumas,
% itmax - maksimalus leistinas iteracijų skaičius.
% Įšėjimo parametrai:
% s     - šaknies reikšmė,
% it    - atliktų iteracijų skaičius.

n=numel(a)-1; it=0; xn=x; [b,c]=hs(a,xn);
```

```
while (it <= itmax) & (abs(b) > e)
    xn1=xn-b/c; xn=xn1; [b,c]=hs(a,xn);
    it=it+1;
end
    if it <= itmax
        s=xn;
    else
        s=nan; disp('nekonverguoja')
    end
```

Pavyzdys. *Apskaičiuokime polinomo* $f(x) = 2x^3 - 3x^2 - 12x - 5$ *didžiausią realiąją šaknį, jei žinomas jos artinys x0=2.3646.*

Komandų lauke surinkę komandas:

```
>> a=[2 -3 -12 -5];
>> x0=2.3646;
>> [s,it]=lstpol(a,x0,1e-13,10),
```

turėsime

```
s = 3.4495,
it = 7 .
```

Kam lygi netiktis?

```
>> hs(a,s)
ans = -1.0658e-014
```

6.3. *MATLAB'o funkcijos polyval* ir *roots*

Funkcija ***polyval*** skirta apskaičiuoti polinomo $y = f(x)$ reikšmei, kai x yra duotas vektorius arba matrica. Į šią funkciją galima kreiptis keliais būdais, paprasčiausias iš kurių yra *y= polyval*(*p,x*), čia *p* polinomo koeficientų masyvas, pradedant vyriausiuoju koeficientu. Plačiau apie ***polyval*** funkciją galima sužinoti surinkus komandą:
help polyval.

Pavyzdys. Apskaičiuokime polinomo $f(x) = 2x^3 - 3x^2 - 12x - 5$ reikšmę, kai $x = 2;4 ir 9$.

```
>> x=[2 4 9];
>> p=[2 -3 -12 -5];
>> y=polyval(p,x)
y =
     -25   27   1102
```

Funkcija ***roots*** yra skirta apskaičiuoti polinomo šaknis. Polinomo šaknų apskaičiavimą galime paversti matricos tikrinių reikšmių (žr. 4.6 paragrafą) apskaičiavimu. Kadangi yra žinomi „galingi" matricos tikrinių reikšmių apskaičiavimo metodai, tai jie taikomi polinomo šaknims apskaičiuoti.

Matricos

$$P = \begin{pmatrix} p_1 & p_2 & p_3 & \cdots & p_{n-1} & p_n \\ 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 & 0 \end{pmatrix}$$

(ši matrica yra viršutinė Hesenbergo matrica) charakteristinis polinomas yra

$$\det(P - \lambda E) = (-1)^n \left(\lambda^n - p_1\lambda^{n-1} - p_2\lambda^{n-2} - \dots - p_{n-1}\lambda - p_n\right).$$

Vadinasi, norint rasti polinomo

$$f(x) = a_0 x^n + a_1 x^{n-1} + \ldots + a_n$$

šaknis, reikia ieškoti matricos

$$P = \begin{pmatrix} -\frac{a_1}{a_0} & -\frac{a_2}{a_0} & -\frac{a_3}{a_0} & \ldots & -\frac{a_{n-1}}{a_0} & -\frac{a_n}{a_0} \\ 1 & 0 & 0 & \ldots & 0 & 0 \\ 0 & 1 & 0 & \ldots & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & 1 & 0 \end{pmatrix}$$

tikrinių reikšmių.

Vienas iš pagrindinių viršutinės Hesenbergo matricos tikrinių reikšmių skaičiavimo metodų yra 1961 metais sukurtas QR metodas. Tokią polinomo šaknų apskaičiavimo schemą ir realizuoja MATLAB'o funkcija ***roots.***

Kreipinys į funkciją yra: ***polsaknys = roots(a)***, čia *a* – polinomo koeficientų, pradedant vyriausiuoju, masyvas.

Pavyzdys. *Apskaičiuokime polinomo* $f(x) = 2x^3 - 3x^2 - 12x - 5$ *šaknis.*

Komandų lauke surinkę komandas:

```
>> a=[2 -3 -12 -5];
>> polsaknys=roots(a)
```

turėsime

```
polsaknys = 3.4495
           -1.4495
           -0.5000
```

Komandą `>> polsaknys=roots(a)` galima pakeisti tokia komandų seka:

```
>> n=numel(a)-1;
>> p=diag(ones(1,n-1),-1);
>> p(1,:)=-a(2:end)/a(1)
>> polsaknys=eig(p),
```

čia ***eig*** matricos tikrinių reikšmių apskaičiavimo MATLAB'o funkcija.

Turėsime:

```
p = 1.5000    6.0000    2.5000
    1.0000         0         0
         0    1.0000         0,
polsaknys =
    3.4495
   -1.4495
   -0.5000.
```

Reikia pastebėti, kad funkcija ***roots*** netiksliai sprendžia polinomus, kurie turi kartotinių šaknų.

Pavyzdys. *Išspręskime lygtį* $x^6 - 19x^5 + 127x^4 - 381x^3 + 692x^2 - 1220x + 800 = 0$ (*lygtis paimta iš literatūros R. Čiegis, V. Būda. Skaičiuojamoji matematika, TEV, Vilnius,* 1997)

Tikslios šios lygties šaknys yra: $x_1 = 1; x_2 = 2i; x_3 = -2i; x_4 = 5; x_5 = 5; x_6 = 8$.

Funkcija ***roots*** apskaičiuoja šaknis:

```
8.00000000000003,
4.99999999999998 + 0.00000030772607i,
4.99999999999998 - 0.00000030772607i,
0.00000000000000 + 2.00000000000000i,
0.00000000000000 - 2.00000000000000i,
```

```
1.00000000000000.
```

Jei šią lygtį spręstume Berstou metodu turėtume:

```
4.99999998027732,
1.00000000000000,
0.00000000000000 + 2.00000000000000i,
0.00000000000000 - 2.00000000000000i,
8.00000000000000,
5.00000001972268.
```

Matome, kad Berstou metodas šaknis apskaičiavo tiksliau nei funkcija ***roots***.

3.2 *Netiesinių lygčių sistemų sprendimas*

Uždavinio formulavimas. Duota lygčių sistema

$$\begin{cases} f_1(x_1, x_2, \ldots, x_n) = 0, \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ f_n(x_1, x_2, \ldots, x_n) = 0; \end{cases} \qquad (3.5)$$

čia $f_i(x_1, x_2, \ldots, x_n),\ i = \overline{1, n}$ — funkcijos, nusakytos n-matės vektorinės erdvės srityje D. (3.5) sistemą galima užrašyti viena vektorine lygtimi $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$; čia $\boldsymbol{f}(\boldsymbol{x})$ — vektorinė funkcija, kurios koordinatės yra funkcijos $f_i(x_1, x_2, \ldots, x_n),\ i = \overline{1, n}$ ir kuri n-matę vektorinę erdvę transformuoja pačią į save.

Reikia rasti vektorių $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^t$, tenkinantį (3.5) sistemą.

Visi šio uždavinio sprendimo metodai yra iteraciniai. Kaip nurodyta literatūroje, daugelis šio uždavinio vienažingsnių iteracijų metodų gali būti užrašyti tokia apibendrinta formule:

$$B_{k+1}\frac{\boldsymbol{x}_{k+1} - \boldsymbol{x}_k}{\tau_{k+1}} + \boldsymbol{f}(\boldsymbol{x}_k) = \boldsymbol{0},\ k = 0, 1, \ldots;$$

čia $\boldsymbol{x}_0 = \left(x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}\right)^t$ — duotas vektorius (pradinis sistemos sprendinys), k — iteracijos numeris, $\boldsymbol{x}_k = \left(x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)}\right)^t$ — (3.5) sistemos sprendinys po k iteracijų, B_{k+1} — n-tosios eilės kvadratinė neišsigimusi matrica, τ_{k+1} — skaitiniai parametrai.

Iš šios lygties išplaukia, kad

$$B_{k+1}\boldsymbol{x}_{k+1} = \boldsymbol{g}(\boldsymbol{x}_k); \qquad (3.6)$$

čia $\boldsymbol{g}(\boldsymbol{x}_k) = B_{k+1}\boldsymbol{x}_k - \tau_{k+1}\boldsymbol{f}(\boldsymbol{x}_k)$.

Jei $B_{k+1} = E$ su visais $k = 0, 1, 2, \ldots$, tai (3.6) formule nusakytas metodas vadinamas ***išreikštiniu***, priešingu atveju — ***neišreikštiniu***.

Jei B ir τ nepriklauso nuo iteracijos numerio k, tai (3.2.2) formulė apibūdina stacionarųjį metodą.

Toliau aptarsime keletą (3.5) sistemos sprendimo metodų.

3.2.1 Niutono metodas

Aiškinant Niutono metodą, naudosime Jakobio matricos sąvoka. Todėl pirmiausia ją ir apibrėšime.

Apibrėžimas. Vektorinės funkcijos $\boldsymbol{f}(\boldsymbol{x})$: $R^n \to R^n$ išvestine, jeigu ji egzistuoja, vadinama Jakobio matricą

$$J(\boldsymbol{x}) = \boldsymbol{f}'(\boldsymbol{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \dots & \dots & \dots & \dots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \dots & \frac{\partial f_n}{\partial x_n} \end{pmatrix}.$$

Netiesinių lygčių sistemas patogu spręsti Niutono metodu, kuris pasižymi kvadratiniu konvergavimo greičiu.

Pirmiausia išsiaiškinkime, kaip Niutono metodas taikomas dviejų lygčių sistemai spręsti. Tarkime, kad turime sistemą

$$\begin{cases} f_1(x_1,\, x_2) = 0, \\ f_2(x_1,\, x_2) = 0, \end{cases} \tag{3.7}$$

pradinį jos sprendinį $\boldsymbol{x}_0 = \left(x_1^{(0)};\, x_2^{(0)}\right)$, be to, tokias Δx_1 ir Δx_2 reikšmes, su kuriomis $x_1^{(0)} + \Delta x_1$ ir $x_2^{(0)} + \Delta x_2$ yra (3.7) sistemos sprendiniai. Kaip galima rasti tokias Δx_1 ir Δx_2 reikšmes? Funkcijas $f_1(x_1,\, x_2)$ ir $f_2(x_1, x_2)$ išskleidę Teiloro eilute taško $\left(x_1^{(0)};\, x_2^{(0)}\right)$ aplinkoje, turime lygtį

$$f_1\left(x_1^{(0)} + \Delta x_1,\quad x_2^{(0)} + \Delta x_2\right) = f_1 + \frac{\partial f_1}{\partial x_1}\Delta x_1 + \frac{\partial f_1}{\partial x_2}\Delta x_2 +$$

$$+ \frac{1}{2}\left(\frac{\partial^2 f_1}{\partial x_1^2}\Delta x_1^2 + 2\frac{\partial^2 f_1}{\partial x_1\, \partial x_2}\Delta x_1 \Delta x_2 + \frac{\partial^2 f_1}{\partial x_2^2}\Delta x_2^2 \right) + \ldots = 0$$

ir analogišką lygtį, apibūdinančią funkciją $f_2\left(x_1^{(0)} + \Delta x_1,\, x_2^{(0)} + \Delta x_2\right)$. Palikę pirmuosius tris eilutės narius, turime lygčių sistemą

$$\begin{cases} \frac{\partial f_1}{\partial x_1}\Delta x_1 + \frac{\partial f_1}{\partial x_2}\Delta x_2 = -f_1, \\ \frac{\partial f_2}{\partial x_1}\Delta x_1 + \frac{\partial f_2}{\partial x_2}\Delta x_2 = -f_2, \end{cases} \tag{3.8}$$

kurią galima užrašyti matricine išraiška:

$$J \cdot \Delta\mathbf{x} = -\mathbf{f}\,; \tag{3.9}$$

čia J — antrosios eilės Jakobio matrica, $\Delta\boldsymbol{x}$ ir $\boldsymbol{f}$ — vektoriai stulpeliai, kurių komponentės yra atitinkamai Δx_1 ir Δx_2 bei f_1 ir f_2.

Išsprendę (3.9) lygčių sistemą, aišku, kad negausime tokių pataisų, kurias pridėję prie pradinio sprendinio, rastume tikslų sprendinį, tačiau tikimės, kad $\left(x_1^{(0)} + \Delta x_1;\, x_2^{(0)} + \Delta x_2\right)$ bus tikslesnis sprendinys nei $\left(x_1^{(0)};\; x_2^{(0)}\right)$.

Jei spręstume (3.5) lygčių sistemą, tai (3.9) lygtyje J būtų *n*-tosios eilės Jakobio matrica, o $\Delta\boldsymbol{x}$ ir $\boldsymbol{f}$ — *n*-mačiai vektoriai stulpeliai.

Niutono metodo algoritmas

Duota: netiesinių lygčių sistema $\boldsymbol{f}(\boldsymbol{x}) = \mathbf{0}$,

$\boldsymbol{x}_0$ — pradinis sistemos sprendinys.

Reikia rasti: sprendinį $\boldsymbol{x}$ tikslumu ε.

k-toji iteracija
$(k = 0, 1, 2, \ldots)$

Sprendinio ieškoma tokia seka:

1) apskaičiuojama $\boldsymbol{f}(\boldsymbol{x}_k)$ ir nusprendžiama, ar skaičiavimą reikia baigti, ar tęsti;

2) apskaičiuojama Jakobio matrica $J(\boldsymbol{x}_k)$;

3) išsprendžiama lygtis $J(\boldsymbol{x}_k)\Delta\boldsymbol{x}_k = -\boldsymbol{f}(\boldsymbol{x}_k)$;

4) apskaičiuojamas patikslintas sistemos sprendinys $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta\boldsymbol{x}_k$.

Skaičiavimo pabaigos sąlyga. Skaičiavimas baigiamas, kai arba 1) $\max\limits_{1\le i\le n}\left|\dfrac{\Delta \mathbf{x}_i}{\mathbf{x}_i}\right| < \varepsilon$, arba 2) $\max\limits_{1\le i\le n}\left|\mathbf{f}_i(\mathbf{x})\right| < \varepsilon$.

Įrodyta, jei $\boldsymbol{x}_0$ — pradinis sistemos sprendinys priklauso metodo konvergavimo sričiai, tai Niutono metodo konvergavimo greitis yra kvadratinis.

Metodo trūkumai.

1. Praktiškai sunku rasti pradinį sistemos sprendinį, kad metodas konverguotų.
2. Kiekvienoje iteracijoje reikia apskaičiuoti Jakobio matricą: n^2 funkcijų dalinių išvestinių, o taip pat n funkcijų reikšmių. Pavyzdžiui, jei turime dešimties lygčių sistemą (n=10), tai vartotojas turės apibrėžti 110 funkcijų. Aišku, kad dalines išvestines galima aproksimuoti skaitinio diferencijavimo būdu. Šiuo atveju vartotojui nereikės apibrėžti dalinių išvestinių funkcijų, tačiau skaičiavimo apimtis kiekvienoje iteracijoje nuo to nesumažės.

Žemiau pateikta funkcija ***niutmet*** , realizuojanti Niutono metodą.

```
function [s,it,p]=niutmet(func,fjc,x0,e,itmax);
% NIUTMET apskaičiuoja netiesinių lygčių sistemos sprendinį Niutono
% metodu.
% Įėjimo parametrai
% func   - vardas funkcijos, apskaičiuojančios sistemos kaire pusę,
% fjc    - vardas funkcijos, apskaičiuojančios Jakobio matricą,
% x0     - pradinis sistemos sprendinys,
% e      - sprendinio apskaičiavimo tikslumas,
% itmax  - maksimalus leistinas iteracijų skaičius.
% Išėjimo parametrai
% s      - sistemos sprendinys,
% it     - atliktu iteracijų skaičius,
% p      - požymis, rodantis ar sprendinys geras:
%          p=true, jei rastas norimo tikslumo sistemos sprendinys,
%          p=false,- priešingu atveju.

it=0; s=x0' p=true;
```

```
ff=feval(func,s);
z=max(abs(ff));
while (it <= itmax) & (z > e)
     it=it+1;
     J=feval(fjc,s);
     s=s-J\ff;
     ff=feval(func,s);
     z=max(abs(ff));
end
    if it > itmax
       p=false;
    end
```

Pavyzdys. *Apskaičiuokime sistemos* $\begin{cases} tg(x_1x_2+0.4)-x_1^2=0, \\ 0.6x_1^2+2x_2^2-1=0, \end{cases}$ *sprendinį* 10^{-10} *tikslumu, kai žinomas pradinis artinys* $x_0=(1.0;0.5)$.

$J=\begin{pmatrix} \dfrac{x_2}{\cos^2(x_1x_2+0.4)}-2x_1, & \dfrac{x_1}{\cos^2(x_1x_2+0.4)} \\ 1.2\mathrm{x}_1 \quad , & 4\mathrm{x}_2 \end{pmatrix}$ - sistemos Jakobio matrica.

```
s =  % pradinis artinys
    1.0000
    0.5000
```

Pirmoji iteracija.

```
ff = % f(s) reikšmės
   0.26015821755034
   0.10000000000000
J = % Jakobio matrica
  -0.70600063337018   2.58799873325965
   1.20000000000000   2.00000000000000
s = % pirmosios iteracijos metu patikslintas sprendinys
   1.05788838685156
   0.41526696788906
```

Antroji iteracija.

```
ff = % f(s) reikšmės
  -0.00505183228133
   0.01637001266079
J = % Jakobio matrica
  -1.18509483475850   2.37090279553621
   1.26946606422188   1.66106787155625
s = % antrosios iteracijos metu patikslintas sprendinys
   1.04840661375915
   0.41265827045681
```

Trečioji iteracija.

```
ff = % f(s) reikšmės
  1.0e-004 *
   0.75655003357822
   0.67553017171651
J = % Jakobio matrica
  -1.18553535873002   2.31520803776059
   1.25808793651099   1.65063308182724
s = % trečiosios iteracijos metu patikslintas sprendinys
   1.04840014084294
   0.41262227849710
```

```
Ketvirtoji iteracija.
ff = % f(s) reikšmės
  1.0e-008 *
   0.44353305472811
   0.26159816535198
J = % Jakobio matrica
  -1.18568282841217   2.31498810441310
   1.25808016901153   1.65048911398842
s = % ketvirtosios iteracijos metu patikslintas sprendinys
   1.04840014110262
   0.41262227671419
ff = % netiktis
  1.0e-015 *
  -0.22204460492503
                   0
sprendinys = % norimo tikslumo sprendinys
   1.048400141103
   0.412622276714
```

3.2.2 Broideno (kirstinių) metodas

Kaip minėjome kiekvienoje Niutono metodo iteracijoje reikia apskaičiuoti $n^2 + n$ funkcijos reikšmių ir išspręsti tiesinę lygčių sistemą. Be to, reikia žinoti gana tikslų sistemos sprendinio pradinį artinį. Šiems trūkumams išvengti siūloma įvairių Niutono metodo modifikacijų, kurios vadinamos bendru ***kvaziniutono metodo*** vardu.

Viena iš plačiai naudojamų modifikacijų yra anksčiau aptarto kirstinių metodo apibendrinimas. Kiekvienoje Niutono metodo iteracijoje Jakobio matrica nėra skaičiuojama remiantis funkcijų dalinėmis išvestinėmis arba jų aproksimacijomis, o yra atnaujinama remiantis jau turimomis funkcijų $\boldsymbol{f}(\boldsymbol{x})$ reikšmėmis. Vienas iš sėkmingiausių kirstinių metodo apibendrinimų yra ***Broideno* (*Broyden C. G.*)** metodas.

Broideno metodo algoritmas pilnai sutampa su Niutono metodo algoritmu išskyrus Jakobio matricos apskaičiavimą.

Klasikinis Broideno metodas. Tarkime, kad $\boldsymbol{x}_0$ yra netiesinių lygčių sistemos $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$ sprendinio pradinis artinys, o $\boldsymbol{A}_0$ yra Jakobio matrica $\boldsymbol{J}(\boldsymbol{x}_0)$ arba jos aproksimacija. Remiantis $\boldsymbol{A}_0$, patiksliname sprendinį pagal formulę

$$x_1 = x_0 - A_0^{-1} * f(x_0).$$

Aptarsime, kaip remiantis $\boldsymbol{A}_0$, $\boldsymbol{x}_0$, $\boldsymbol{x}_1$ ir $\boldsymbol{f}(\boldsymbol{x}_0)$, $\boldsymbol{f}(\boldsymbol{x}_1)$, apskaičiuoti $A_1 \approx J(x_1)$. Matrica $\boldsymbol{A}_1$ bus naudojama apskaičiuojant patikslintą sistemos sprendinį $\boldsymbol{x}_2$.

Kirstinių metodas vienai lygčiai iš esmės yra Niutono metodas, kai tiksli $f'(x_i)$ reikšmė aproksimuojama pagal formulę

$$f'(x_i) \approx \frac{f(x_1) - f(x_0)}{x_1 - x_0}.$$

Netiesinių lygčių sistemos atveju $x_1 - x_0$ yra vektorius ir santykis $\dfrac{f(x_1) - f(x_0)}{x_1 - x_0}$ yra neapibrėžtas. Tačiau galime elgtis taip: Niutono metode Jakobio matricą $J(x_1)$ pakeiskime nežinoma matrica $\boldsymbol{A}_1$, tenkinančią sąlygą:

$$A_1(x_1 - x_0) = f(x_1) - f(x_0).$$

Ši sąlyga matricą A_1 apibrėžia nevienareikšmiškai: egzistuoja $n(n-1)$ matavimo matricų perdvis, kurio kiekviena matrica tenkina minėtą sąlygą. Broidenas pasiūlė imti tokią matricą A_1, kuri tenkintų papildomas sąlygas:

$A_1 z_i = J(x_0) z_i, i = 1,2,\ldots,n-1$, čia z_i tiesiškai nepriklausomi vektoriai, ortogonalūs vektoriui $x_1 - x_0$, t.y. vektoriai, tenkinantys sąlygą: $(x_1 - x_0) z_i = 0, i = \overline{1, n-1}$. Šis reikalavimas reiškia, kad nei vienas vektorius z_i, kuris yra ortogonalus vektoriui $x_1 - x_0$, nebus pakeistas keičiant matricą $A_0 = J(x_0)$, kuri buvo naudojama apskaičiuojant x_1, į matricą A_1, kuri bus naudojama apskaičiuojant x_2. Žodis „keičiant“ čia reiškia, kad mes neaproksimuojame pilnai matricos $J(x_1)$, pavyzdžiui, skaitinio diferencijavimo metodu, o perskaičiuojame matricos $J(x_0)$ aproksimaciją A_0 į matricą A_1, kuri aproksimuoja $J(x_1)$.

Minėtos papildomos sąlygos vienareikšmiškai apibrėžia matricą A_1 ir tolesnių iteracijų matricas A_i, i=0,1,2, 3,...:

$$A_{i+1} = A_i + \frac{(y_i - A_i s_i) s_i^t}{s_i^t s_i}, \text{ čia } y_i = f(x_{i+1}) - f(x_i), \text{ o } s_i = x_{i+1} - x_i,\ i = 0,1,2,... \quad (3.10)$$

Aišku, kad dabar sistemos sprendinys bus tikslinamas pagal formulę:

$$x_{i+2} = x_{i+1} - A_{i+1}^{-1} f(x_{i+1}),\ i = 0,1,2,... . \quad (3.11)$$

Kaip parodyta literatūroje, jei pradinis artinys x_0 yra pakankamai artimas tiksliam sprendiniui x_*, Jakobio matrica $J(x_*)$ - neišsigimusi ir matrica $A_0 \approx J(x_0)$, tai Broideno metodu apskaičiuota seka $\{x_i\}$ konverguoja prie sprendinio x_* greičiau nei tiesiškai.

Funkcija, realizuojanti Broideno metodą.

```
function [s,it,p]=broiden(func,fjc,x0,e,itmax);
% BROIDEN apskaičiuoja netiesinių lygčių sistemos sprendinį
% Broideno metodu.
% Šios funkcijos parametrai pilnai sutampa su NIUTMET funkcijos
% parametrais.

it=0; p=true;
xi=x0';
ffi=feval(func,xi);
a=feval(fjc,xi);
xi1=xi-inv(a)*ffi;
ffi1=feval(func,xi1);
z=max(abs(ffi1));
while (it <= itmax) & (z > e)
     it=it+1;
     s=xi1-xi; y=ffi1-ffi;
     a=a+(y-a*s)*s'/(s'*s);
     xi2=xi1-a\ffi1;
     ffi2=feval(func,xi2);
     xi=xi1; xi1=xi2;
     ffi=ffi1; ffi1=ffi2;
     z=max(abs(ffi1));
end
    if it > itmax
```

```
       p=false;
    else
        s=xi1;
    end
```

Pavyzdys. A*pskaičiuokime sistemos* $\begin{cases} tg(x_1x_2+0.4)-x_1^2=0, \\ 0.6x_1^2+2x_2^2-1=0, \end{cases}$ *sprendinį Broideno metodu* 10^{-7} *tikslumu, kai žinomas pradinis artinys* $x_0=(1.0;0.5)$.

```
x0=  % pradinis artinys
    1.0000
    0.5000
ff0 = % f(x0) reikšmė
   0.26015821755034
   0.10000000000000
A0 = % A0=J(x0)
  -0.70600063337018   2.58799873325965
   1.20000000000000   2.00000000000000
x1 = % patikslintas sprendinys
   1.05788838685156
   0.41526696788906
Pirmoji iteracija.
ff1 = % f(x1) reikšmė
  -0.00505183228133
   0.01637001266079
A1 = % A1 yra Jakobio matricos J(x1) aproksimacija
  -0.73377095971170   2.62864702355768
   1.28998726966552   1.86828283486815
x2= % patikslintas sprendinys remiantis matrica A1
   1.04686963457109
   0.41411298710459
Antroji iteracija.
ff2 = % f(x2) reikšmė
   0.00526038940219
   0.00054075124959
A2 = % A2 yra Jakobio matricos J(x2) aproksimacija
  -1.20599491294451   2.57919157214853
   1.24144415369576   1.86319897222913
x3 = % patikslintas sprendinys remiantis matrica A2
   1.04841240532746
   0.41279481603313
Trečioji iteracija.
ff3 = % f(x3) reikšmė
 1.0e-003 *
   0.38497345563959
   0.30026327435939
A3 = % A3 yra Jakobio matricos J(x3) aproksimacija
  -1.06175822793975   2.45595315527263
   1.35394276545148   1.76707813575085
x4 = % patikslintas sprendinys remiantis matrica A3
   1.04840141712567
   0.41263331447173
Ketvirtoji iteracija.
ff4 = %  f(x4) reikšmė
  1.0e-004 *
   0.24039703420575
```

```
   0.19823282520148
A4 = % A4 yra Jakobio matricos J(x4) aproksimacija
  -1.07183906307469   2.30778781889761
   1.34563004880010   1.64490011752549
x5 = % patikslintas sprendinys remiantis matrica A4
   1.04840014260108
   0.41262230575227
s= % norimo tikslumo sprendinys
   1.048400143
   0.412622306
ff = % f(s) reikšmė - netiktis
  1.0e-007 *
   0.65446127406332
   0.49812219460321
```

Šermano-Morisono (Sherman-Morrison) modifikacija. Iš pateikto nagrinėjimo matyti, kad vietoje n^2+n funkcijos reikšmių skaičiavimo, kuris reikalingas sprendžiant sistemą Niutono metodu, Broideno metode šis skaičius sumažėjo iki *n,* tačiau abiem atvejais kiekvienoje iteracijoje reikia išspręsti tiesinių lygčių sistemą: $A_{i+1}s_{i+1} = -f(x_{i+1})$. Kad to išvengtume, galime pasinaudoti Šermano-Morisono formule, kuri, žinant atvirkštinę matricą A_i^{-1}, įgalina tiesiogiai apskaičiuoti atvirkštinę matricą A_{i+1}^{-1}.

Šermano-Morisono formulė. Jeigu A yra neišsigimusi matrica, o x ir y – vektoriai, tai matrica $A+xy^t$ yra neišsigimusi tada ir tiktai tada, kai $1+y^tA^{-1}x \neq 0$. Be to

$$\left(A+xy^t\right)^{-1} = A^{-1} - \frac{A^{-1}xy^tA^{-1}}{1+y^tA^{-1}x} \quad .$$

Pritaikę Šermano-Morisono formulę (3.10) formulei, t.y. laikydami, kad $A = A_i$, $x = (y_i - A_i s_i)/(s_i^t s_i)$, $y^t = s_i^t$, turėsime:

$$A_{i+1}^{-1} = A_i^{-1} + \frac{\left(s_i - A_i^{-1}y_i\right)s_i^tA_i^{-1}}{s_i^tA_i^{-1}y_i} \qquad (3.12).$$

Vadinasi, Šermano-Morisono formulė įgalina kirstinių metodą modifikuoti taip, kad kiekvienoje iteracijoje skaičiavimai būtų atliekami pagal formules (3.11) ir (3.12), nesprendžiant tiesinių lygčių sistemos $A_{i+1}s_{i+1} = -f(x_{i+1})$.

Kiti kvaziniutono metodai. Išnagrinėtas Broideno metodas kiekvienoje iteracijoje sumažina skaičiavimų apimtį lyginant su Niutono metodu, tačiau tiek Niutono, tiek ir Broideno metodai yra lokalieji, t.y. jie sparčiai konverguoja, jei žinomas pakankamai tikslus pradinis sistemos sprendinio artinys. Todėl kiti kvaziniutono metodai susiję su tikslo funkcijos $F(\mathbf{x}) = \frac{1}{2}(\mathbf{f}(\mathbf{x}))^t\mathbf{f}(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{n}\left(f_i(x_1, x_2, \ldots, x_n)\right)^2$ minimizavimu, nes šios funkcijos globaliojo minimumo taškai sutampa su (3.5) sistemos šaknimis.

Kvaziniutono metodai — tai tokios Niutono metodo modifikacijos, kurioje lėta globaliojo konvergavimo strategija, susijusi su funkcijos $F(x)$ minimizavimu, derinama su greita lokaliojo konvergavimo strategija. Jei apytikslis sprendinys $\boldsymbol{x}_k$ yra už konvergavimo srities ribų, tai $\boldsymbol{x}_{k+1}$ apskaičiuojamas pagal globaliojo konvergavimo strategiją, o jei $\boldsymbol{x}_k$ priklauso konvergavimo sričiai — Niutono metodu (pagal lokaliojo konvergavimo strategiją).

Kvaziniutono metodo algoritmas

Uždavinio sąlyga formuluojama taip pat, kaip ir taikant Niutono metodą:

k-toji iteracija ($k = 0,1,2,\ldots$)

1) apskaičiuojama $\boldsymbol{f}(\boldsymbol{x}_k)$ ir nusprendžiama, ar skaičiavimą reikia baigti, ar tęsti;

2) apskaičiuojama funkcijos $F(x)$ mažėjimo kryptis p_k: tai tokia kryptis, kuria einant funkcijos $F(x)$ reikšmė mažėja. Matematiškai tai galima apibrėžti taip: jei funkcijos $F(x)$ išvestinė kryptimi p_k yra neigiama, t. y. $\nabla\mathbf{F}(\mathbf{x}_k)^t \cdot \mathbf{p}_k < 0$, tai kryptis p_k yra mažėjimo kryptis, čia simbolis $\nabla\mathbf{F}(\mathbf{x}_k)$ žymi funkcijos $F(x)$ gradientą: $grad(F(x_k)) = \nabla\mathbf{F}(\mathbf{x}_k) = J(x_k)^t f(x_k)$.

Pastaba. Funkcijos $F(x)$ išvestinė kryptimi p_k yra lygi gradiento $\nabla\mathbf{F}(\mathbf{x}_k)$ projekcijai į vektorių p_k, t. y. $\nabla\mathbf{F}(\mathbf{x}_k)^t \cdot \mathbf{p}_k / |p_k|$. Tačiau, norint patikrinti ar $F(x)$ išvestinė kryptimi p_k yra neigiama, patogiau taikyti anksčiau pateiktą formulę $\nabla\mathbf{F}(\mathbf{x}_k)^t \cdot \mathbf{p}_k < 0$.

Pavyzdžiui, Niutono kryptis yra mažėjimo kryptis:

- apskaičiuojama Jakobio matrica $J(\boldsymbol{x}_k)$;
- išsprendžiama sistema $J(\boldsymbol{x}_k)\Delta\boldsymbol{x}_k = -\boldsymbol{f}(\boldsymbol{x}_k)$;
- apskaičiuojama Niutono kryptis $p_k = -J^{-1}(x_k)f(x_k)$.

Kadangi $\nabla\mathbf{F}(\mathbf{x}_k)^t \cdot \mathbf{p}_k = -f^t(x_k)J(x_k)J^{-1}(x_k)f(x_k) = -f^t(x_k)f(x_k) < 0$, tai einant Niutono kryptimi, funkcijos $F(x)$ reikšmė mažėja.

Tačiau tai nėra vienintelė funkcijos $F(x)$ mažėjimo kryptis. Yra kitų funkcijos $F(x)$ mažėjimo krypčių nustatymo metodų, pavyzdžiui, greičiausio nusileidimo metodas, Pauelo (Powell M. J. D.) (tikėtinos srities su dvigubo lūžio žingsniu) metodas ir kt.

3) mažėjimo kryptimi apskaičiuojamas toks žingsnis $\Delta x_k = \alpha \cdot p_k, 0 < \alpha < 1$, kad $F(x_x + \Delta x_k) < F(x_k)$.

4) apskaičiuojamas naujas sprendinio artinys $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta\boldsymbol{x}_k$.

Pavyzdžiui, tarkime, kad, žengus Niutono žingsnį, iš taško $\boldsymbol{x}_k$ gaunamas taškas $\boldsymbol{x}_{k+1}$, t. y. $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \left(J(\boldsymbol{x}_k)\right)^{-1}\boldsymbol{f}(\boldsymbol{x}_k)$.

Ar $\boldsymbol{x}_{k+1}$ yra priimtinas? Aišku, kad jis bus priimtinas tik tada, kai $F(\boldsymbol{x}_{k+1}) < F(\boldsymbol{x}_k)$. Jei ši sąlyga netenkinama, tai žingsnis mažinamas.

3.2.3 Greičiausio nusileidimo metodas.

Niutono ir Broideno metodų privalumas yra didelis jų konvergavimo greitis, kai žinomas pakankamai tikslus pradinis sistemos sprendinio artinys. Tačiau nurodyti tokį artinį nėra paprasta. Todėl panagrinėkime ***greičiausio nusileidimo metodą***, kurio konvergavimo greitis yra tiesinis, tačiau kuris globalus pagal prigimtį. Šis metodas yra derinamas su Niutono metodu panašiai, kaip vienos lygties atveju pusiaukirtos metodas derinamas su liestinių metodu. Taikant greičiausio nusileidimo metodą, pirmiausia randamas pakankamai tikslus pradinis artinys, kuris toliau tikslinamas Niutono metodu.

Greičiausio nusileidimo metodo algoritmas pilnai sutampa su anksčiau aptartu kvaziniutono metodo algoritmu, kai nusileidimo kryptis yra priešinga gradiento krypčiai: $p_k = -grad\ (F(x_k))/|grad\ (F(x_k))|$ ir $grad(F(x_k)) \neq 0$.

Kaip apskaičiuoti tokią α reikšmę α^*, kad $F(x_k + \alpha^* p_k) < F(x_k)$?

Tam tikslui reikia rasti funkcijos $h(\alpha) = F(x_k + \alpha p_k)$ mažiausią reikšmę.

Norint tiksliai apskaičiuoti α^* reikšmę, reikėtų rasti funkcijos $h(\alpha)$ išvestinę, o po to išspręsti lygtį $h'(\alpha) = 0$.

Šis kelias yra per daug sudėtingas, todėl minėtas uždavinys sprendžiamas apytiksliai:

- remiantis trimis taškais $(\alpha_1, h(\alpha_1)), (\alpha_2, h(\alpha_2)), (\alpha_3, h(\alpha_3))$, tenkinančiais sąlygą: $\alpha_1 = 0$, $0 < \alpha_3 \leq 1$ ir $h(\alpha_3) < h(\alpha_1)$, o $\alpha_2 = \alpha_3 / 2$, funkciją $h(\alpha)$ aproksimuojame kvadratiniu polinomu $P(\alpha) = a(\alpha - \alpha_2)^2 + b(\alpha - \alpha_0) + c$;
- apskaičiuojame polinomo $P(\alpha)$ ekstremumo taško abscisė: $\alpha_{\min} = \alpha_2 - \frac{b}{2a}$;
- jei $0 < \alpha_{\min} < \alpha_3$, tai randame α reikšmę $\alpha^* \in \{\alpha_{\min}, \alpha_3\}$, esant kuriai funkcijos $h(\alpha) = F(x_k + \alpha p_k)$ reikšmė yra mažiausia; priešingu atveju $\alpha^* = \alpha_3$.

Kaip apskaičiuoti α_3 reikšmę, tenkinančią sąlygą; $h(\alpha_3) < h(\alpha_1)$?

Ši reikšmė apskaičiuojama taip:

$\alpha_3 = 1$;

while $h(\alpha_3) > h(\alpha_1)$ *do* $\alpha_3 = \alpha_3 / 2$;

Koeficientų a ir b apskaičiavimo formulės.

Pažymėkime: $h(\alpha_1) = h_1$, $h(\alpha_2) = h_2$, $h(\alpha_3) = h_3$. Kadangi anksčiau išvardinti taškai:

$$(\alpha_1, h(\alpha_1)), (\alpha_2, h(\alpha_2)), (\alpha_3, h(\alpha_3))$$

turi tenkinti polinomą

$$P(\alpha) = a(\alpha - \alpha_2)^2 + b(\alpha - \alpha_0) + c,$$

tai koeficientai *a* ir *b* apskaičiuojami pagal formules:

$$a = \frac{(h_1 - h_2)(\alpha_3 - \alpha_2) - (h_3 - h_2)(\alpha_1 - \alpha_2)}{(\alpha_1 - \alpha_2)(\alpha_3 - \alpha_2)(\alpha_1 - \alpha_3)},$$

$$b = \frac{(h_3 - h_2)(\alpha_1 - \alpha_2)^2 - (h_1 - h_2)(\alpha_3 - \alpha_2)^2}{(\alpha_1 - \alpha_2)(\alpha_3 - \alpha_2)(\alpha_1 - \alpha_3)}.$$

Žemiau pateikta greičiausio nusileidimo metodą realizuojanti funkcija ***grnm***, kurioje gradientas apskaičiuojamas tiksliai pagal formulę: $grad(F(x_k)) = \nabla \mathbf{F}(\mathbf{x}_k) = J(x_k)^t f(x_k)$.

Funkcija, realizuojanti greičiausio nusileidimo metodą.

```
function [s,it,p]=grnm(func,fjc,x0,e,itmax);
% GRNM apskaičiuoja netiesinių lygčių sistemos sprendinį
% greičiausio nusileidimo metodu.
```

```
% Šios funkcijos parametrai pilnai sutampa su NIUTMET funkcijos
% parametrais.

it=0; p=true;
x1=x0'; ff=feval(func,x1); z=sum(ff.*ff)/2;
while (it <= itmax) & (z > e)
     it=it+1; J=feval(fjc,x1);
     grad=J'*ff; grad=-grad/norm(grad,2); % antigradiento kryptis
     a1=0; h1=z;
     a3=1; x3=x1+a3*grad; ff=feval(func,x3); h3=sum(ff.*ff)/2;
     while h3 > h1
         a3=a3/2; x3=x1+a3*grad; ff=feval(func,x3);
         h3=sum(ff.*ff)/2;
     end
     a2=a3/2; x2=x1+a2*grad; ff=feval(func,x2);
     h2=sum(ff.*ff)/2;
% Kvadratinė aproksimacija
     v=(a1-a2)*(a3-a2)*(a1-a3);
     a=((h1-h2)*(a3-a2)-(h3-h2)*(a1-a2))/v;
     b=((h3-h2)*(a1-a2)^2-(h1-h2)*(a3-a2)^2 )/v;
     alfa=a2-b/(2*a);
     if (alfa > 0) & (alfa < a3)
         x1=x1+alfa*grad; ff=feval(func,x1);
         halfa=sum(ff.*ff)/2;
         if halfa > h3
             x1=x3;
         end
     else
         x1=x3;
     end
     ff=feval(func,x1); z=sum(ff.*ff)/2;
end
    if it > itmax
       p=false; s=nan;
    else
       s=x1;
    end
```

Pavyzdys. Greičiausio nusileidimo metodu raskime sistemos

$$\begin{cases} 3x_1 - \cos(x_2 x_3) - 0.5 = 0, \\ x_1^2 - 81(x_2 + 0.1)^2 + \sin x_3 + 1.06 = 0, \\ e^{-x_1 x_2} + 20x_3 + \dfrac{10\pi - 3}{3} = 0. \end{cases}$$

sprendinio artinį, kai paieška pradedama iš taško $x_0 = (0,0,0)^t$.

Skaičiavimai patalpinti 3.3 lentelėje.

3.3 *lentelė. Sistemos sprendinio artinio skaičiavimas greičiausio nusileidimo metodu.*

it	x_1	x_2	x_3	*F(x)*
0	0.0	0.0	0.0	55.9874

1	0.0112182	0.0100964	-0.522741	1.16381
2	0.1378597	-0.2054528	-0.522059	0.63703
3	0.2669594	0.0055110	-0.558494	0.53407
4	0.2727338	-0.0081175	-0.522006	0.23415
5	0.3086893	-0.0204026	-0.533112	0.19054

Toliau sistemos sprendinys buvo tikslinamas Niutono metodu. Po keturių iteracijų buvo apskaičiuotas sprendinys 10^{-13} tikslumu:

$x = (0.5, 0, -0.52359877559830)^t$.

Pastaba. *Pateiktoje funkcijoje* ***grnm*** *gradientą galima apskaičiuoti skaitiniu būdu. Šiuo atveju vietoj įėjimo funkcijų* ***func*** *ir* ***fjc*** *reikėtų imti funkciją* ***F****, kuri apskaičiuotų sistemos tikslo funkciją* $F(\mathbf{x}) = \frac{1}{2}(\mathbf{f}(\mathbf{x}))^t \mathbf{f}(\mathbf{x})$.

Žemiau pateikta taip modifikuota funkcija ***grnm1***, ankstesnio pavyzdžio funkcija ***F*** ir lentelėje 3.4 ankstesnės sistemos sprendimo rezultatai.

```
function [s,it,p]=grnm1(F,x0,e,itmax);
% GRNM1 apskaičiuoja netiesinių lygčių sistemos sprendinį
% greiciausio nusileidimo metodu.
% Įėjimo parametrai
% F      - vardas funkcijos, apskaičiuojančios sistemos tikslo funkciją,
% x0     - pradinis sistemos sprendinys,
% e      - sprendinio apskaičiavimo tikslumas,
% itmax  - maksimalus leistinas iteracijų skaičius.
% Išėjimo parametrai
% s      - sistemos sprendinys,
% it     - atliktu iteracijų skaičius,
% p      - požymis, rodantis ar sprendinys geras:
%          p=true, jei rastas sistemos norimo tikslumo sprendinys,
%          p=false,- priešingu atveju.

n=numel(x0); delta=0.0001; it=0; p=true; x1=x0; z=feval(F,x1);
while (it <= itmax) & (z > e)
     it=it+1;
% Gradiento apskaičiavimas
     for i=1:n
         xd=x1; xd(i)=xd(i)+delta;
         grad(i)=(feval(F,xd)-feval(F,x1))/delta;
     end
     grad=-grad/norm(grad,2);% antigradiento kryptis
% Interpoliavimo taškų antigradiento kryptimi apskaičiavimas
     a1=0; h1=z; a3=1; x3=x1+a3*grad;  h3=feval(F,x3);
     while h3 > h1
         a3=a3/2; x3=x1+a3*grad; h3=feval(F,x3);
     end
     a2=a3/2; x2=x1+a2*grad; h2=feval(F,x2);
% Kvadratinė aproksimacija
     v=(a1-a2)*(a3-a2)*(a1-a3);
     a=((h1-h2)*(a3-a2)-(h3-h2)*(a1-a2))/v;
     b=((h3-h2)*(a1-a2)^2-(h1-h2)*(a3-a2)^2 )/v;
     alfa=a2-b/(2*a);
     if (alfa > 0) & (alfa < a3)
         x1=x1+alfa*grad; halfa=feval(F,x1);
         if halfa > h3
             x1=x3;
         end
     else
         x1=x3;
     end
     z=feval(F,x1);
    end
```

```
        if it > itmax
           p=false; s=nan;
        else
           s=x1;
        end
```

```
function fv=F(x);
% F funkcija, apskaičiuojanti sistemos tikslo funkcijos reikšmę.

fv=((3*x(1)-cos(x(2)*x(3))-0.5)^2+...
    (x(1)^2-81*(x(2)+0.1)^2+sin(x(3))+1.06)^2+...
    (exp(-x(1)*x(2))+20*x(3)+(10*pi-3)/3)^2)/2;
```

***3.4** lentelė. Sistemos sprendinio artinio skaičiavimas greičiausio nusileidimo metodu, kai gradientas apskaičiuojamas skaitiniu būdu.*

it	x_1	x_2	x_3	*F(x)*
0	0.0	0.0	0.0	55.9874
1	0.01121597	0.01006775	-0.52274336	1.16360357
2	0.13774529	-0.20554838	-0.52262054	0.63746706
3	0.26709452	0.00649043	-0.55105152	0.44407969
4	0.27304018	-0.00805584	-0.52140295	0.23402224
5	0.29772886	-0.01664804	-0.53215331	0.20096686

3.2.4 Funkcija fsolve

MATLAB‘o funkcija ***fsolve*** yra skirta netiesinių lygčių sistemai spręsti, kai žinomas pradinis sistemos sprendinio artinys. Ši funkcija realizuoja kvaziniutono metodą, kai mažėjimo kryptis vidutinės apimties sistemoms nustatoma Pauelo metodu .

Norėdami gauti išsamesnę informaciją apie šią funkciją ir jos parametrus, surinkite komandą: ***help fsolve.***

Aptarkime du dažniausiai naudojamus kreipinius.

```
s=fsolve(fun,x0);
s=fsolve(fun,x0,options);
```

Čia ***fun*** yra vardas funkcijos (*m*-failo), apskaičiuojančios sistemos kairiąją pusę, t.y. sistemos funkcijų reikšmių stulpelį, o ***x0*** – sistemos sprendinio artinys.

Parametras ***options*** yra speciali duomenų struktūra, kurią sukuria funkcija ***optimset***. Plačiau apie šią funkciją žiūrėk 3.1.2 paragrafo 5 poskyrį.

Funkcija ***fsolve*** turi daug jos darbą reguliuojančių parametrų.

Pagrindiniai fsolve parametrai. Parametras ***LargeScale*** nusako didelės ar vidutinės apimties sistemos yra sprendžiamos (jos sprendžiamos skirtingais metodais). Parametro reikšmės gali būti: „***off***“ arba „***on***“. Jei parametro reikšmė yra „***off***“, tai sprendžiama vidutinės apimties sistema; priešingu atveju – didelės apimties. Pagal nutylėjimą parametro reikšmė yra „***off***“.

Parametras ***Display*** nurodo kokie sprendimo rezultatai rodomi ekrane. Parametro reikšmės gali būti: „***off***“, „***iter***“ arba „***final***“. Jei parametro reikšmė yra „***off***“, tai jokie rezultatai nerodomi; jei - „***iter***“, tai rodomi kiekvienos iteracijos rezultatai, jei „***final*** “ (ši reikšmė priskiriama pagal nutylėjimą) rodomas tik galutinis sistemos sprendinys.

Parametras ***Jacobian*** nurodo, ar Jakobio matrica skaičiuojama pagal vartotojo sudarytą funkciją, ar skaitiniu būdu. Parametro reikšmės gali būti: „***on***“, arba „***off***“ (ši reikšmė

priskiriama pagal nutylėjimą). Jei parametro reikšmė yra „***on***“, tai naudojama vartotojo sudaryta Jakobio matrica, kuri aprašoma anksčiau paminėtos funkcijos ***fun*** kūne.

Parametras ***MaxFunEvals*** nurodo didžiausią leistiną skaičiuotų funkcijos reikšmių skaičių. Pagal nutylėjimą šis skaičius lygus sistemos kintamųjų skaičiui padaugintam iš šimto.

Parametras ***MaxIter*** nurodo didžiausią leistiną iteracijų skaičių. Pagal nutylėjimą parametro reikšmė lygi 400.

Parametras ***TolFun*** nurodo mažiausią sistemos tikslo reikšmę, kurią pasiekus skaičiavimas baigiamas. Pagal nutylėjimą ***TolFun*** reikšmė yra 10^{-6}.

Parametras ***TolX*** nurodo sistemos sprendinio tikslumą. Pagal nutylėjimą ***TolX*** $=10^{-6}$.

Pavyzdys. Pasinaudodami funkcija ***fsolve,*** išspręskime sistemą

$$\begin{cases} 2x_1 - x_2 - 0.0625e^{0.25x_1} - 1 = 0, \\ -x_1 + 2x_2 - x_3 - 0.0625e^{0.5x_2} = 0, \\ -x_2 + 2x_3 - 0.0625e^{0.75x_3} - e = 0, \end{cases}$$

kai Jakobio matrica skaičiuojama analitiškai.

Sudarykime funkciją ***fjx***, kuri apskaičiuotų sistemos kairiąją pusę ir Jakobio matricą.

```
function [F,J] = fjx(x);
F =[2*x(1)-x(2)-0.0625*exp(0.25*x(1))-1;
   -x(1)+2*x(2)-x(3)-0.0625*exp(0.5*x(2));
   -x(2)+2*x(3)-0.0625*exp(0.75*x(3))-exp(1)];
if nargout > 1   % du išėjimo argumentai
  a=0.0625;
J=[2-a*0.25*exp(0.25*x(1)) -1 0;
    -1 2-a*0.5*exp(0.5*x(2)) -1;
    0 -1 2-a*0.75*exp(0.75*x(3))];
end
```

Atlikę komandas:

```
x0=[0 0 0]
options=optimset('Jacobian','on');
[root,fval]=fsolve('fjx',x0,options)
```

turėsime:

```
x0 = 0     0     0
root =
   1.73109963035545   2.36585368602258   2.79661316703612
fval =
  1.0e-008 *
  -0.00068332006720
  -0.02240637952955
  -0.19848194199312
```