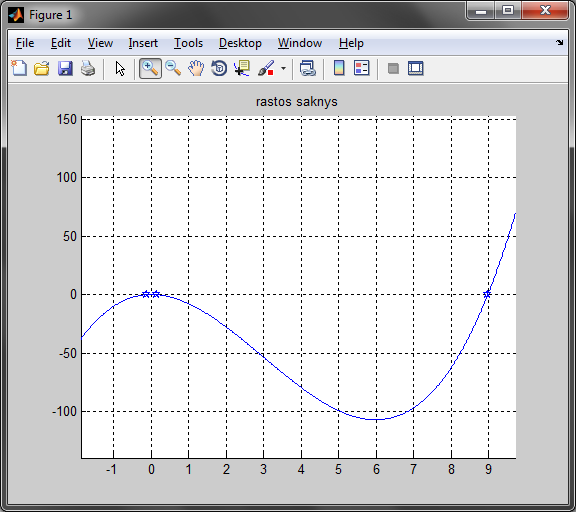
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Grupė** | **Pavardė Vardas** | **Savarankiško darbo Nr.** | **Lygčių Nr.** | **Sprendimo metodas** | |
| **daugianario** | **sistemos** |
| IFF-0 | Rubavičius Osvaldas | 2 | 2 | Kvazi-Niutono | Paprastųjų iteracijų |

1. *Vienos lygties sprendimo algoritmai*. **Duota** daugianario lygtis f(x)=0.

* Daugianario grafikas su pažymėtomis šaknimis.



Pav 1. Rastos šaknys Kvazi-Niutono metodu

* Rezultatų lentelė

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Metodas** | | **Lygtis** | | | **Šaknų intervalo įverčiai** | |
| Kvazi-Niutono | |  | | | Grubus  Tikslesnis | |
| **Šaknis** | **Intervalas**  ***arba* artinys** | | **Tikslumas** | **Iteracijų skaičius** | **Iteracijų pabaigos sąlygos** | ***roots* funkcijos rezultatas** |
| 8.9809 | 9.0009 | | 5.82077e-11 | 4 | prec < 1e-9 | 8.9809 |
| 0.1351 | 0.1551 | | 1.96648e-14 | 6 | prec < 1e-9 | 0.1351 |
| -0.1159 | -0.0959 | | 1.10207e-14 | 7 | prec < 1e-9 | -0.1159 |

* Programos kodas

function daugianaris

clc; clear all; close all;

syms f x;

% apsibreziame funkcija:

f = (64\*x.^4)-576\*(x.^3)+10\*(x.^2)+9\*x;

% ------------------------Grafinis atvaizdavimas-----------------------

% ribos ir tasku skaicius

xmin = -10;

xmax = 10;

points = 1000;

firstBoundary = xmin :(xmax-xmin)/points : xmax;

plotFunction(firstBoundary, f);

pause;

close all;

% ------------------------Funkcijos f intervalu radimas----------------

% pakeisti x-> -x ir iskleisti funkcija

fNegative = expand(subs(f,x,-x));

% funkcijos koeficientai ir ju tvarka

[coefficients, orders] = coeffs(f);

[coefficientsNegative, ordersNegative] = coeffs(fNegative);

% koeficientas prie auksciausio x laipsnio turi buti teigiamas:

positiveFunctionCoeffs = eval(coefficients/coefficients(1));

negativeFunctionCoeffs = eval(coefficientsNegative/coefficientsNegative(1));

% ------------- Grubus ivertis-----------------------------------------

disp('Grubaus ivercio reiksme:');

roughBoundary = roughBoundaries(positiveFunctionCoeffs)

% grafinis funkcijos, saknu ir grubaus ivercio intervalo pavaizdavimas:

roughInterval = -roughBoundary:roughBoundary/500:roughBoundary;

figure(1);

grid on;

hold on;

plot(roughInterval, fnk(positiveFunctionCoeffs, roughInterval), 'b-');

plot([-roughBoundary,roughBoundary], [0 0],'r\*');

% ------------ Tikslesnis ivertis--------------------------------------

disp('Tikslesnio ivercio reiksme:');

[lowerBoundary, higherBoundary] = accurateBoundaries(positiveFunctionCoeffs, negativeFunctionCoeffs)

% pavaizduojame tikslesni iverti:

plot(min(roughBoundary,lowerBoundary),0,'bp');

plot(min(roughBoundary,higherBoundary),0,'bp');

legend('kreive f(x)','grubus saknu intervalo ivertis','tikslesnis saknu intervalo ivertis');

title([char(f),'=0']);

pause;

close all;

%------------------Pradiniu artiniu nustatymas-------------------------

disp('Pradinius artinius paimame is teisingu rezultatu aibes rastos roots su roots funkcija, pridejus nedideli nuokrypi kad matytusi skaiciavimu iteracijos:');

saknys = roots(positiveFunctionCoeffs);

saknys = saknys + 0.02

%--------------- Kvazi-Niutono metodas---------------------------------

results = kvaziNewton(f, saknys);

close all;

figure(1);

title('rastos saknys');

grid on;

hold on;

plot(roughInterval, fnk(positiveFunctionCoeffs, roughInterval), 'b-');

for i = 1 : size(results, 2)

plot(results(i),0,'bp');

end

pause;

disp('Roots funkcijos rezultatas:');

roots(positiveFunctionCoeffs)

end

function p=fnk(CF,x)

% Apskaiciuoja daugianario reiksmes, kai argumentas yra x

% Kai x yra reiksmiu vektorius, p taip pat yra atitinkamu funkcijos reiksmiu vektorius

p=0; n=length(CF)-1;

for i=1:length(CF), p=p+CF(i)\*x.^(n-i+1); end

return

end

function plotFunction(array, func)

x = array;

figure();

plot(x, eval(func), 'b-'); grid on; hold on;

title([char(func),'=0']);

end

function roughBoundary = roughBoundaries(coeffs)

roughBoundary = max(abs(coeffs(2:end)))/coeffs(1)+1;

return

end

function [lowerAccurateBoundary, higherAccurateBoundary] = accurateBoundaries(coeffs, negativeCoeffs)

% teigiamoms saknims:

neig\_ind = find(coeffs(2:end) < 0);

if ~isempty(neig\_ind)

B = max(abs(coeffs(neig\_ind+1)));

k = neig\_ind(1);

higherAccurateBoundary = 1+(B/coeffs(1))^(1/k);

else

higherAccurateBoundary = 0;

end

% neigiamoms saknims:

neig\_ind1 = find(negativeCoeffs(2:end) < 0);

if ~isempty(neig\_ind1)

B = max(abs(negativeCoeffs(neig\_ind1+1)));

k = neig\_ind1(1);

lowerAccurateBoundary = 1+(B/negativeCoeffs(1))^(1/k);

lowerAccurateBoundary = -lowerAccurateBoundary;

else

lowerAccurateBoundary = 0;

end

return

end

function results = kvaziNewton(f, intervals)

close all;

results = [];

for i = 1 : size(intervals, 1)

x0 = intervals(i, 1); % parenkame pradini artini

deltax=0.01; % parenkame pradine zingsnio reiksme (reikalinga tik kirstiniu metodui)

nitmax=100; % parenkame didziausia leistina iteraciju skaiciu

if x0 ~= 0, range=3\*[-abs(x0),abs(x0)]; % parenkame intervala vaizdavimui

else, range=[-3,0];

end

eps=1e-9; % Parenkame tiksluma

x=x0;fxn=eval(f);x=x0+deltax;fxn\_1=eval(f);dfxn=(fxn\_1-fxn)/deltax

% Taikant kirstiniu metoda reiks isvestines apytiksles

% reiksmes pradinio artinio taske

% braizomas funkcijos grafikas:

npoints=1000; xrange=range(1): (range(2)-range(1))/(npoints-1) :range(2);

figure(1); grid on; hold on; axis equal; title('Kvazi-Niutono metodas');

x=xrange; % simbolinis x keiciamas reiksmemis is parinkto funkcijos vaizdavimo intervalo

plot(x,eval(f),'r-');plot(range,[0 0],'b-');plot(x0,0,'mp');h = findobj(gca,'Type','line');h1=h(1);

input('Press Enter'), figure(1); % skaiciavimas stabdomas iki bus paspaustas Enter klavisas

%------------------------ SPRENDIMAS -----------------------------------

xn=x0;prec=1;nit=0;

while prec > eps % iteracijos

nit=nit+1;

if nit > nitmax, fprintf('Virsytas leistinas iteraciju skaicius');break;end

xn1=xn-fxn/dfxn;

plot([xn,xn,xn1],[0,fxn,0],'g-');

delete(h1);plot(xn1,0,'mp');h = findobj(gca,'Type','line');h1=h(1);

x=xn1;fxn1=eval(f);dfxn=(fxn1-fxn)/(xn1-xn);

xn=xn1;

fxn=fxn1;

x=xn;fxn=eval(f);prec=abs(fxn);

fprintf(1,'iteracija %d x= %g prec= %g \n',nit,xn,prec);

end

plot(xn,fxn,'k\*');plot(xn,fxn,'ko');

disp('Saknis xn:')

results = [results xn]

nit

end

disp('Kvazi-Niutono metodo rezultatai:')

results

end

1. *Tiesinių lygčių sistemų sprendimo algoritmai*. **Duota** tiesinių lygčių sistema [A][X]=[B].
2. Rezultatų lentelė

|  |  |  |  |
| --- | --- | --- | --- |
| **Sprendinys [X]** | **Daugikliai ir iteracijų skaičius (iteraciniams metodams)** | **Sprendinio patikrinimas** | |
| **[A][X]-[B]** | **Lygčių sistemos sprendimo funkcija linsolve** |
| -17.8881 | alpha=[1.7; 1.7; 1.7; 1.7; 1.7]  iteraciju sk = 136 | 1.0e-09 \*  0.4267  0.1381  -0.1505  -0.5119  -0.0678 | -17.8881 |
| 5.0651 | 5.0651 |
| 2.0319 | 2.0319 |
| 33.3596 | 33.3596 |
| -3.1267 | -3.1267 |

1. Programos kodas

function sistema

clc; close all; clear all;

%\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Paprastosios iteracijos\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

disp('\*\*\*Paprastuju iteraciju algoritmas\*\*\*');

disp('Kintamuju koeficientu matrica:');

A = [ 5 1 2 4 1;

1 6 1 2 2;

2 1 15 1 1;

4 2 1 7 1;

1 2 1 1 5 ]

disp('Laisvieji nariai:');

b = [ 50; 75; 30; 171; 12 ]

n=size(A,1)

Aprad=A;

disp('Alpha reiksmes:');

alpha=[1.7; 1.7; 1.7; 1.7; 1.7] % laisvai parinkti metodo parametrai

Atld=diag(1./diag(A))\*A-diag(alpha)

btld=diag(1./diag(A))\*b

nitmax=1000;

eps=1e-12;

x=zeros(n,1);

for it=1:nitmax

x1=(btld-Atld\*x)./alpha;

prec(it)=norm(x1-x)/(norm(x)+norm(x1));

fprintf(1,'iteracija Nr. %d, tikslumas %g\n',it,prec(it))

if prec(it) < eps, break, end

x=x1;

end

x

disp('patikrinimas A\*x-b')

Aprad\*x-b

disp('linsolve patikrinimas');

linsolve(A, b)

semilogy([1:length(prec)],prec,'r.');grid on,hold on

end