1. Diferența

function val = dif1(f, x, h)

val = (f(x+h) - f(x)) ./ h;

end

1. Diferența 2

function val = dif2(f, x, h)

val = (f(x+h) - f(x)) ./ h;

end

1. Diferența 3

function val = dif3(f, x, h)

val = (f(x+h) - f(x-h)) ./ (2\*h);

end

1. Regula trapezului compusă

function int = trapez\_comp(f, a, b, m)

y0 = f(a);

ym = f(b);

s = 0;

h = (b-a)/m;

for i = 1:m-1

s = s + f(a + i\*h);

end

int = h/2\*(y0 + ym + 2\*s);

1. Regula lui Simpson compusă

function int = simpson\_comp(f, a, b, m)

y0 = f(a);

ym = f(b);

s = 0;

h = (b-a) / (2\*m);

for i = 1:m-1

s = s + f(a + 2\*i\*h);

end

s1 = 0;

for i = 1:m

s1 = s1 + f(a + (2\*i-1)\*h);

end

int = h/3\*(y0 + ym + 4\*s1 + 2\*s);

1. Regula mijlocului compusă

function int = mijloc\_comp(f, a, b, m)

y0 = f(a);

ym = f(b);

s = 0;

h = (b-a) / m;

for i = 1:m-1

s = s + f(a + ((2\*i-1)\*h)/2);

end

int = h\*s;

1. Integrarea Romberg

function r = romberg(f, a, b, n)

h = (b-a) ./ (2.^(0:n-1));

r(1,1) = (b-a)\*(f(a)+f(b)) / 2;

for j = 2:n

subtotal = 0;

for i = 1:2^(j-2)

subtotal = subtotal + f(a+(2\*i-1)\*h(j));

end

r(j,1) = r(j-1,1)/2 + h(j)\*subtotal;

for k = 2:j

r(j,k) = (4^(k-1)\*r(j,k-1)-r(j-1,k-1))/(4^(k-1)-1);

end

end

1. Regula trapezului

function int = trapez(f, x0, x1)

y0 = f(x0);

y1 = f(x1);

h = x1 - x0;

int = h/2\*(y0+y1);

1. Cuadratura adaptativă cu regula trapezului

function int = adapquad(f, a0, b0, tol0)

int = 0;

n = 1;

a(1) = a0;

b(1) = b0;

tol(1) = tol0;

app(1) = trapez(f, a, b);

while n>0

c = (a(n) + b(n)) / 2;

oldapp = app(n);

app(n) = trapez(f, a(n), c);

app(n+1) = trapez(f, c, b(n));

if abs(oldapp - (app(n) + app(n+1))) < 3\*tol(n)

int = int + app(n) + app(n+1);

n = n-1;

else

b(n+1) = b(n);

b(n) = c;

a(n+1) = c;

tol(n) = tol(n)/2;

tol(n+1) = tol(n);

n = n+1;

end

end

1. Cuadratura adaptivă cu regula lui Simpson

function int = adapquad(f, a0, b0, tol0)

int = 0;

n = 1;

a(1) = a0;

b(1) = b0;

tol(1) = tol0;

app(1) = simpson\_comp(f, a, b);

while n>0

c = (a(n) + b(n)) / 2;

oldapp = app(n);

app(n) = simpson\_comp(f, a(n), c);

app(n+1) = simpson\_comp(f, c, b(n));

if abs(oldapp - (app(n) + app(n+1))) < 10\*tol(n)

int = int + app(n) + app(n+1);

n = n-1;

else

b(n+1) = b(n);

b(n) = c;

a(n+1) = c;

tol(n) = tol(n)/2;

tol(n+1) = tol(n);

n = n+1;

end

end

1. Cuadratura gausiannă

function int = quad\_gauss(f, x, c, n)

sum = 0;

for i = 1:n

sum = sum + c(i)\*f(x(i));

end

int = sum;

1. Metoda lui Euler 1

function [t,y] = euler1(inter, y0, n)

t(1) = inter(1);

y(1) = y0;

h = (inter(2) - inter(1)) / n;

for i = 1:n

t(i+1) = t(i) + h;

y(i+1) = eulerstep(t(i),y(i),h);

end

plot(t,y)

function y = eulerstep(t,y,h)

y = y + h\*ydot(t,y);

function z = ydot(t,y)

z=t\*y + t^3;

1. Metoda lui Euler 2

function [t,y] = euler2(inter, y0, n)

t(1) = inter(1);

y(1,:) = y0;

h = (inter(2) - inter(1)) / n;

for i = 1:n

t(i+1) = t(i) + h;

y(i+1,:) = eulerstep(t(i),y(i,:),h);

end

plot(t,y(:,1),t,y(:,2));

function y = eulerstep(t,y,h)

y = y + h\*ydot(t,y);

function z = ydot(t,y)

z(1) = y(2)^2 - 2\*y(1);

z(2) = y(1) - y(2) - t\*y(2)^2;

1. Metoda trapezului explicită

function [t, y] = trapez\_exp(inter, y0, n)

t(1) = inter(1);

y(1) = y0;

h = (inter(2) - inter(1)) / n;

for i = 1:n

t(i+1) = t(i) + h;

y(i+1) = eulerstep(t(i), y(i), h);

end

plot(t, y);

function y = eulerstep(t, y, h)

y = y + h/2\*(ydot(t, y) + ydot(t+h, y+h\*ydot(t, y));

function z = ydot(t, y)

z = 1/(y\*y);

1. Metoda mijlocului

function [t, y] = mijloc(inter, y0, n)

t(1) = inter(1);

y(1) = y0;

h = (inter(2) - inter(1)) / n;

for i = 1:n

t(i+1) = t(i) + h;

y(i+1) = mijstep(t(i), y(i), h);

end

plot(t, y);

function y = mijstep(t, y, h)

y = y + h\*ydot(t+h/2, y+h/2\*ydot(t,y));

function z = ydot(t, y)

z = 1/(y\*y);

1. Metoda Runge-Kutta de ordinul 4 (RK4)

function [t, y] = rk4(inter, y0, n)

t(1) = inter(1);

y(1) = y0;

h = (inter(2) - inter(1)) / n;

for i = 1:n

t(i+1) = t(i) + h;

y(i+1) = rk4step(t(i),y(i),h);

end

plot(t,y);

function y = rk4step(t,y,h)

s1 = ydot(t,y);

s2 = ydot(t+h/2,y+h/2\*s1);

s3 = ydot(t+h/2,y+h/2\*s2);

s4 = ydot(t+h,y+h\*s3);

y = y + h/6\*(s1+2\*s2+2\*s3+s4);

function z = ydot(t,y)

z = t^3/y^2;

1. Metoda Adams-Bashford cu doi pași

function [t, y] = exmultistep9(inter, y0, n, s)

h = (inter(2) - inter(1)) / n;

y(1,:) = y0;

t(1) = inter(1);

for i = 1:s-1

t(i+1) = t(i) + h;

y(i+1,:) = trapstep(t(i),y(i,:),h);

f(i,:) = ydot(t(i),y(i,:));

end

for i = s : n

t(i+1) = t(i) + h;

f(i,:) = ydot(t(i),y(i,:));

y(i+1,:) = ab2step(t(i),i,y,f,h);

end

plot(t,y)

function y = trapstep(t,x,h)

z1 = ydot(t,x);

g = x + h\*z1;

z2 = ydot(t+h,g);

y = x + h\*(z1+z2)/2;

function z = ab2step(t,i,y,f,h)

z = y(i,:) + h\*(3\*f(i,:)/2-f(i-1,:)/2);

function z = unstable2step(t,i,y,f,h)

z = -y(i,:) + 2\*y(i-1,:) + h\*(5\*f(i,:)/2+f(i-1,:)/2);

function z = weaklystable2step(t,i,y,f,h)

z = y(i-1,:) + h\*2\*f(i,:);

function z = ydot(t,y)

z = t\*y + t^3;

1. Metoda Adams-Bashfort cu trei pași

function [t, y] = exmultistep11(inter,y0,n,s)

h = (inter(2)-inter(1))/n;

y(1,:) = y0;

t(1) = inter(1);

for i = 1:s-1

t(i+1) = t(i) + h;

y(i+1,:) = rk4step(t(i),y(i,:),h);

f(i,:) = ydot(t(i),y(i,:));

end

for i = s:n

t(i+1) = t(i)+h;

f(i,:) = ydot(t(i),y(i,:));

y(i+1,:) = ab3step(t(i),i,y,f,h);

end

plot(t,y)

function y = rk4step(t,y,h)

s1 = ydot(t,y);

s2 = ydot(t+h/2,y+h\*s1/2);

s3 = ydot(t+h/2,y+h\*s2/2);

s4 = ydot(t+h,y+h\*s3);

y = y + h\*(s1+2\*s2+2\*s3+s4)/6;

function z = ab3step(t,i,y,f,h)

z = y(i,:) + h/12\*(23\*f(i,:) - 16\*f(i-1,:) + 5\*f(i-2,:));

function z = unstable2step(t,i,y,f,h)

z = -y(i,:)+2\*y(i-1,:) + h\*(5\*f(i,:)/2+f(i-1,:)/2);

function z = weaklystable2step(t,i,y,f,h)

z = y(i-1,:) + h\*2\*f(i,:);

function z = ydot(t,y)

%z = 1/(y\*y);

%z=2\*(t+1)\*y;

z=(t\*t\*t)/(y\*y);

1. Metoda Adams-Bashfort cu patru pași

function [t, y] = exmultistep11(inter,y0,n,s)

h = (inter(2)-inter(1))/n;

y(1,:) = y0;

t(1) = inter(1);

for i = 1:s-1

t(i+1) = t(i) + h;

y(i+1,:) = rk4step(t(i),y(i,:),h);

f(i,:) = ydot(t(i),y(i,:));

end

for i = s:n

t(i+1) = t(i)+h;

f(i,:) = ydot(t(i),y(i,:));

y(i+1,:) = ab3step(t(i),i,y,f,h);

end

plot(t,y)

function y = rk4step(t,y,h)

s1 = ydot(t,y);

s2 = ydot(t+h/2,y+h\*s1/2);

s3 = ydot(t+h/2,y+h\*s2/2);

s4 = ydot(t+h,y+h\*s3);

y = y + h\*(s1+2\*s2+2\*s3+s4)/6;

function z = ab4step(t,i,y,f,h)

z = y(i,:) + h/24\*(55\*f(i,:) - 59\*f(i-1,:) + 37\*f(i-2,:) - 9\*f(i-3, :));

function z = unstable2step(t,i,y,f,h)

z = -y(i,:)+2\*y(i-1,:) + h\*(5\*f(i,:)/2+f(i-1,:)/2);

function z = weaklystable2step(t,i,y,f,h)

z = y(i-1,:) + h\*2\*f(i,:);

function z = ydot(t,y)

%z = 1/(y\*y);

%z=2\*(t+1)\*y;

z=(t\*t\*t)/(y\*y);

1. Metoda stabilă cu doi pași

function [t,y] = exmultistep10(inter,y0,n,s)

h = (inter(2)-inter(1))/n;

y(1,:) = y0;

t(1) = inter(1);

for i = 1:s-1

t(i+1) = t(i) + h;

y(i+1,:) = trapstep(t(i),y(i,:),h);

f(i,:) = ydot(t(i),y(i,:));

end

for i = s:n

t(i+1) = t(i) + h;

f(i,:) = ydot(t(i),y(i,:));

y(i+1,:) = unstable2step(t(i),i,y,f,h);

end

plot(t,y)

function y = trapstep(t,x,h)

z1 = ydot(t,x);

g = x + h\*z1;

z2 = ydot(t+h,g);

y = x + h\*(z1+z2)/2;

function z = ab2step(t,i,y,f,h)

z = y(i,:) + h\*(3\*f(i,:)/2-f(i-1,:)/2);

function z = unstable2step(t,i,y,f,h)

z = -y(i,:) + 2\*y(i-1,:) + h\*(5\*f(i,:)/2+f(i-1,:)/2);

function z = weaklystable2step(t,i,y,f,h)

z = y(i-1,:) + h\*2\*f(i,:);

function z = ydot(t,y)

%z = 1/(y\*y);

%z = 2\*(t+1)\*y;

%z = (t\*t\*t)/(y\*y);

1. Metoda Adams-Moulton cu doi pași

function [t,y] = predcorr(inter,y0,n,s)

h = (inter(2)-inter(1))/n;

y(1,:) = y0;

t(1) = inter(1);

for i = 1:s-1

t(i+1) = t(i) + h;

y(i+1,:) = trapstep(t(i),y(i,:),h);

f(i,:) = ydot(t(i),y(i,:));

end

for i = s:n

t(i+1) = t(i) + h;

f(i,:) = ydot(t(i),y(i,:));

y(i+1,:) = ab2step(t(i),i,y,f,h);

f(i+1,:) = ydot(t(i+1),y(i+1,:));

y(i+1,:) = am1step(t(i),i,y,f,h);

end

plot(t,y)

function y = trapstep(t,x,h)

z1 = ydot(t,x);

g = x + h\*z1;

z2 = ydot(t+h,g);

y =x + h\*(z1+z2)/2;

function z = ab2step(t,i,y,f,h)

z = y(i,:) + h\*(3\*f(i,:)-f(i-1,:))/2;

function z = am1step(t,i,y,f,h)

z = y(i,:) + h\*(f(i+1,:)+f(i,:))/2;

function z = ydot(t,y)

%z = 1/(y\*y);

%z = 2\*(t+1)\*y;

%z = (t\*t\*t)/(y\*y);

1. Metoda Adams-Moulton cu trei pași

function [t, y] = predcorr\_ex13(inter, y0, n, s)

h = (inter(2) - inter(1)) / n;

y(1,:) = y0;

t(1) = inter(1);

for i = 1:s-1

t(i+1) = t(i) + h;

y(i+1,:) = trapstep(t(i),y(i,:),h);

f(i,:) = ydot(t(i),y(i,:));

end

for i = s:n

t(i+1) = t(i) + h;

f(i,:) = ydot(t(i),y(i,:));

y(i+1,:) = ab4step(t(i),i,y,f,h);

f(i+1,:) = ydot(t(i+1),y(i+1,:));

y(i+1,:) = am3step(t(i),i,y,f,h);

end

plot(t,y);

function y = trapstep(t,x,h)

z1 = ydot(t,x);

g = x + h\*z1;

z2 = ydot(t+h,g);

y = x + h\*(z1+z2)/2;

function z = ab4step(t,i,y,f,h)

z = y(i,:) + h/24\*(55\*f(i,:)-59\*f(i-1,:)+37\*f(i-2,:)-9\*f(i-3,:));

function z = am3step(t,i,y,f,h)

z = y(i,:) + h/24\*(9\*f(i+1,:)+19\*f(i,:)-5\*f(i-1,:)+f(i-2,:));

function z = ydot(t,y)

z=1/y^2;

1. Funcția de interpolare trigonometrică

function xp = dftinterp(inter, x, n, p)

c = inter(1);

d = inter(2);

t = c + (d-c)\*(0:n-1)/n;

tp = c + (d-c)\*(0:p-1)/p;

y = fft(x);

yp = zeros(p,1);

yp(1:n/2+1) = y(1:n/2+1);

yp(p-n/2+2:p) = y(n/2+2:n);

xp = real(ifft(yp))\*(p/n);

plot(t,x,’o’,tp,xp)

1. Funcția de aproximare trigonometrică

function xp = dftfilter(inter, x, m, n, p)

c = inter(1);

d = inter(2);

t = c + (d-c)\*(0:n-1)/n;

tp = c + (d-c)\*(0:p-1)/p;

y = fft(x);

yp = zeros(p,1);

yp(1:m/2) = y(1:m/2);

yp(m/2+1) = real(y(m/2+1));

if(m<n)

yp(p-m/2+1) = yp(m/2+1);

end

yp(p-m/2+2:p) = y(n-m/2+2:n);

xp = real(ifft(yp))\*(p/n);

plot(t,x,’o’,tp,xp)

1. Aproximarea TCD de tip cele mai mici pătrate

function xp = tcd(y, t, m, n)

y0 = 1/sqrt(n)\*y(1);

sum = 0;

for k = 2 : m

sum = sum + sqrt(2/n) .\* (y(k) .\* cos((k-1) .\* (2.\*t + 1)\*pi)/(2\*n));

end

xp = y0 + sum;

1. Funcția de compresie

function out = compresie (img, p)

n = 8;

for i=1:n

for j=1:n

C(i,j)=cos((i-1)\*(2\*j-1)\*pi/(2\*n));

end

end

C=sqrt(2/n)\*C;

C(1,:)=C(1,:)/sqrt(2);

Q=p\*8./hilb(8);

dim = length(img);

out = zeros (dim);

for i=1:8:dim

for j=1:8:dim

X=img(i:i+7 , j:j+7);

Xd=double(X);

Xc=Xd -128;

Y = C\*Xc\*C';

Yq=round(Y./Q);

Ydq=Yq.\*Q;

Xdq=C'\*Ydq\*C;

Xe=Xdq+128;

Xf=uint8(Xe);

out(i:i+7 , j:j+7) = Xf;

end

end

imshow(out, [0 255])

1. Iterația de putere

function [lambda, u] = powerit(A, x, k)

for j = 1:k

u = x/norm(x);

x = A\*u;

lambda = u'\*x;

end

u = x/norm(x);

1. Iterația de putere inversă

function [lambda, u] = invpowerit(A, x, s, k)

As = A - s\*eye(size(A));

for j = 1:k

u = x/norm(x);

x = As\u;

lambda = u'\*x;

end

lambda = 1/lambda + s;

u = x/norm(x);

1. Iterația câtului Rayleigh

function [lambda, u] = rqi(A, x, k)

for j = 1:k

u = x/norm(x);

lambda = u'\*A\*u;

x = (A-lambda\*eye(size(A)))\u;

end

u = x/norm(x);

lambda = u'\*A\*u;

1. Iterația simultană normalizată

function lambda = nsi(A, k)

[m,n] = size(A);

Q = eye(m,m);

for j = 1:k

[Q,R] = qr(A\*Q);

end

lambda = diag(Q'\*A\*Q);

1. Algoritmul QR nedeplasat

function lambda = unshiftedqr(A, k)

[m,n] = size(A);

Q = eye(m,m);

R = A;

for j = 1:k

[Q,R] = qr(R\*Q);

end

lambda = diag(R\*Q);

1. Algoritmul QR deplasat

function lambda = shiftedqr(A)

tol = 1e-14;

m = size(A,1);

lambda = zeros(m,1);

n = m;

while n>1

while max(abs(A(n,1:n-1))) > tol

s = A(n,n);

[Q,R] = qr(A-s\*eye(n));

A = R\*Q + s\*eye(n);

end

lambda(n) = A(n,n);

n = n-1;

A = A(1:n,1:n);

end

lambda(1) = A(1,1);

1. Căutarea secțiunii de aur (GSS)

function xmin = gss(f,a,b,k)

g = (sqrt(5)-1)/2;

x1 = a + (1-g)\*(b-a);

x2 = a + g\*(b-a);

f1 = f(x1);

f2 = f(x2);

for i = 1:k

if f1 < f2

b = x2;

x2 = x1;

x1 = a + (1-g)\*(b-a);

f2 = f1;

f1 = f(x1);

else

a = x1;

x1 = x2;

x2 = a + g\*(b-a);

f1 = f2;

f2 = f(x2);

end

end

xmin = (a+b)/2;

1. Interpolarea parabolică succesivă (SPI)

function xmin = spi(f,r,s,t,k)

x(1) = r;

x(2) = s;

x(3) = t;

fr = f(r);

fs = f(s);

ft = f(t);

for i = 4:k+3

x(i) = (r+s)/2-(fs-fr)\*(t-r)\*(t-s)/(2\*((s-r)\*(ft-fs)-(fs-fr)\*(t-s)));

t = s;

s = r;

r = x(i);

ft = fs;

fs = fr;

fr = f(r);

end

xmin = x(k+3);

1. Metoda lui Newton

function x = newton(Df,Hf,x0,k)

x = x0;

for i = 1:k

x = x - Hf(x)\Df(x);

end

1. Metoda gradientului

function x = mgradient(f,Df,x0,k)

x = x0;

for i = 1:k

v = Df(x);

fun = @(s) f(x-s\*v);

s = fminbnd(fun,0,1);

x = x - s\*v;

end

1. Căutarea gradienţilor conjugaţi

function x = gradconj(f,df,x0,k)

d = -df(x0);

r = -df(x0);

x = x0;

for i = 1:k

fun = @(alfa) f(x+alfa\*d);

alfa = fminbnd(fun,0,1);

x = x + alfa\*d;

r1 = r;

r = -df(x);

beta = (r'\*r)/(r1'\*r1);

d = r + beta\*d;

end