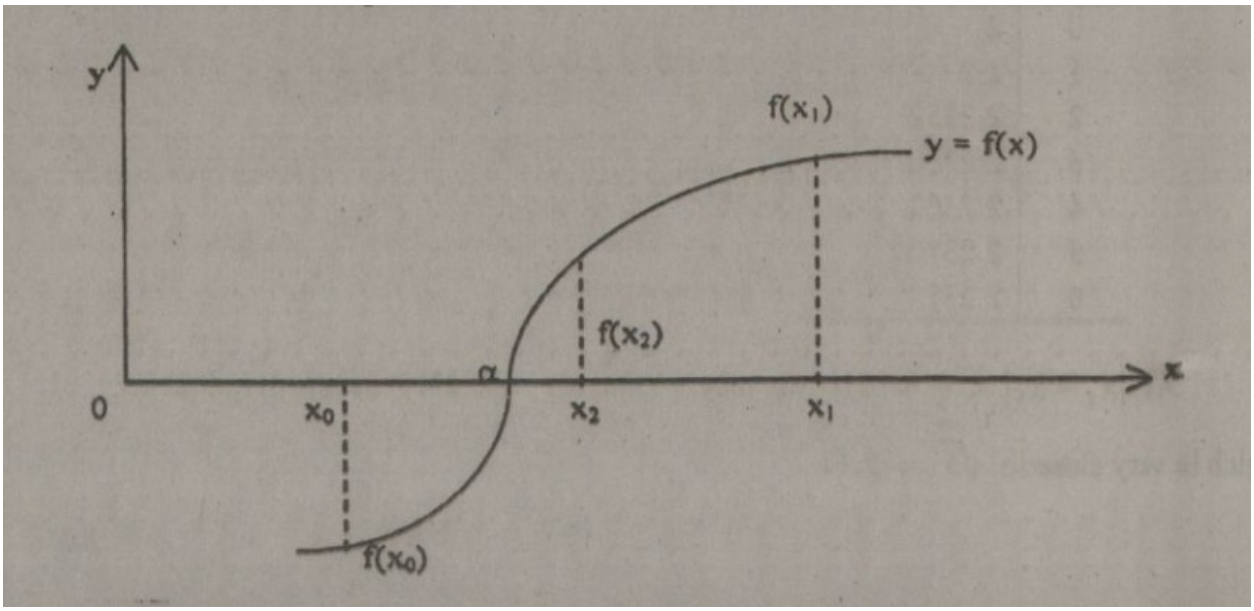
Lab:01

Bisection code in MATLAB

Introduction

Bisection method is used for finding the real root of a given function say f(x)=0. It requires two starting values x_0 and x_1 for the solution such that $f(x_0)$. $f(x_1) <0$. Then the equation f(x) = 0 has at least one root in the interval (x_0 , x_1). Graphically it can be represented as:



Procedure of bisection method rely on repeated application of the following:

1. Find x_2, the new iterate using, $x_2 = (x_0 + x_1)/2$ and evaluate $f(x_2)$.
2. If $f(x_2) = 0$ then x_2 is a root of f(x).
3. If $f(x_2) \neq 0$ then two things are possible:
   i) If $f(x_0)\ f(x_2) < 0$, we compute the new iterate x_3 as:
      $x_3 = (x_0 + x_2)/2$ and evaluate $f(x_3)$
   ii) If $f(x_1)\ f(x_2) < 0$, we compute the new iterate x_3 as:
      $x_3 = (x_1 + x_2)/2$ and evaluate $f(x_3)$

Matlab code for Bisection Method

A function f(x) = `x.^3+4*x.^2-10` is given. Find the root in interval [1 ,2]

Codes for bisection method in matlab:

```
% Solution of nonlinear equations
% Bisection method
clear all, close all, clc
format short
f = %question;
a=1; b=2;% roots
max_I=20;% maximum number of iterations
tol=10^(-5);% tolerance
fa=feval(f,a);  % Initial guess
fb=feval(f,b);
if fa*fb>0,

   disp('Wrong choice')
end


for k=1:max_I
   c=(a+b)/2;
   C(k)=c;
   fc=feval(f,c);
   Fc(k)=fc;
   if fc==0,break
   end
   if fb*fc>0;
      b=c;
      fb=fc;
   else
      a=c;
      fa=fc;
end
if abs(fc)<tol,break
end
end

fprintf('no of iteration(k), root sequence(x_k), function values fc\n')
[(1:k);C(1:k);Fc(1:k)]'
```

Example solve on matlab

```
% Solution of nonlinear equations
% Bisection method
clear all, close all, clc
format short
f = @(x) 2*x.^3*x.^2+10*x+1;
a=1; b=3;
max_I=15;% maximum number of iterations
```

```
tol=10^(-5);
fa=feval(f,a);  % Initial guess
fb=feval(f,b);
if fa*fb>0,

   disp('Wrong choice')
end



for k=1:max_I
   c=(a+b)/2;
   C(k)=c;
   fc=feval(f,c);
   Fc(k)=fc;
   if fc==0,break
   end
   if fb*fc>0;
b=c;
fb=fc;
else
a=c;
fa=fc;
end
if abs(fc)<tol,break
end
end


fprintf('no of iteration(k), root sequence(x_k), function values fc\n')
[(1:k);C(1:k);Fc(1:k)]'
```

Solution:

```
        Wrong choice
no of iteration(k), root sequence(x_k), function values fc


ans =

   1.0000    2.0000   85.0000
   2.0000    1.5000   31.1875
   3.0000    1.2500   19.6035
   4.0000    1.1250   15.8541
```

5.0000	1.0625	14.3332
6.0000	1.0312	13.6452
7.0000	1.0156	13.3175
8.0000	1.0078	13.1575
9.0000	1.0039	13.0784
10.0000	1.0020	13.0391
11.0000	1.0010	13.0196
12.0000	1.0005	13.0098
13.0000	1.0002	13.0049
14.0000	1.0001	13.0024
15.0000	1.0001	13.0012

Another example on bisection method solve on matlab:

Codes:

```
 % Solution of nonlinear equations

% Bisection method
clear all, close all, clc
format short
f = @(x) 3*x.^4-x.^3+5*x.^2-7;
a=2; b=5;
max_I=20;% maximum number of iterations
tol=10^(-3);
fa=feval(f,a); % Initial guess
fb=feval(f,b);
if fa*fb>0,
disp('Wrong choice')
end



for k=1:max_I
   c=(a+b)/2;
   C(k)=c;
   fc=feval(f,c);
   Fc(k)=fc;
   if fc==0,break
   end
```

```
    if fb*fc>0;
       b=c;
       fb=fc;
    else
       a=c;
       fa=fc;
end
if abs(fc)<tol,break
end
end


fprintf('no of iteration(k), root sequence(x_k), function values fc\n')
[(1:k);C(1:k);Fc(1:k)]'
```

Solution:

```
   Wrong choice
no of iteration(k), root sequence(x_k), function values fc

ans =

    1.0000    3.5000  461.5625
    2.0000    2.7500  181.5898
    3.0000    2.3750  103.2566
    4.0000    2.1875   75.1514
    5.0000    2.0938   63.3932
    6.0000    2.0469   58.0334
    7.0000    2.0234   55.4768
    8.0000    2.0117   54.2285
    9.0000    2.0059   53.6118
   10.0000    2.0029   53.3053
   11.0000    2.0015   53.1525
   12.0000    2.0007   53.0762
   13.0000    2.0004   53.0381
   14.0000    2.0002   53.0190
   15.0000    2.0001   53.0095
   16.0000    2.0000   53.0048
   17.0000    2.0000   53.0024
   18.0000    2.0000   53.0012
```

19.0000	2.0000	53.0006
20.0000	2.0000	53.0003

Lab: 02

Title: Newton Raphson method.

Introduction:

In numerical analysis, Newton's method, also known as the Newton–Raphson method, named after Isaac Newton and Joseph Raphson, is a root-finding algorithm which produces successively better approximations to the roots (or zeroes) of a real-valued function. The most basic version starts with a single-variable function f defined for a real variable x, the function's derivative f′, and an initial guess x_0 for a root of f. If the function satisfies sufficient assumptions and the initial guess is close, then

$$X_1 = x_0 + \frac{f(x0)}{f(x0)'}$$

is a better approximation of the root than x_0. Geometrically, (x_1, 0) is the intersection of the x-axis and the tangent of the graph of f at (x_0, f (x_0)): that is, the improved guess is the unique root of the linear approximation at the initial point. The process is repeated as

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

until a sufficiently precise value is reached. This algorithm is first in the class of Householder's methods, succeeded by Halley's method. The method can also be extended to complex functions and to systems of equations.

Codes for this method in matlab:

```
% Solution of nonlinear equations
% Newton-Raphson method
clear all, close all, clc
format short
f = @(x) x.^3+4*x.^2-10;
df= @(x) 3*x.^2+8*x;
N=20;% maximum number of iterations
tol=10^(-3);
x0=2;  % Initial guess
```

```
x(1)=x0;
for k=2:N
x(k) = x(k-1)-feval(f,x(k-1))/feval(df,x(k-1));
error=abs(x(k)-x(k-1));
if (error<tol), break;
end
end
% fprintf('no of iteration(k), root sequence(x_k) \n')
[(1:k);x(1:k)]'
```

Example on this method in matlab:

```
% Solution of nonlinear equations
% Newton-Raphson method
clear all, close all, clc
format short
f = @(x) 2*x.^3+x.^2-10;
df= @(x) 6*x.^2+2*x;
N=20;% maximum number of iterations
tol=10^(-3);
x0=2;  % Initial guess
x(1)=x0;
for k=2:N
x(k) = x(k-1)-feval(f,x(k-1))/feval(df,x(k-1));
error=abs(x(k)-x(k-1));
if (error<tol), break;
end
end
% fprintf('no of iteration(k), root sequence(x_k) \n')
[(1:k);x(1:k)]'
```

Solution in matlab:

```
ans =

    1.0000    2.0000
    2.0000    1.6429
    3.0000    1.5624
    4.0000    1.5585
    5.0000    1.5585
```

Other example:

```
% Solution of nonlinear equations
% Newton-Raphson method
clear all, close all, clc
format short
f = @(x) x.^3-x.*sin(x);
df= @(x) 3*x.^2-x.*cos(x)+sin(x);
N=20;% maximum number of iterations
tol=10^(-3);
x0=2;  % Initial guess
x(1)=x0;
for k=2:N
x(k) = x(k-1)-feval(f,x(k-1))/feval(df,x(k-1));
error=abs(x(k)-x(k-1));
if (error<tol), break;
end
end
% fprintf('no of iteration(k), root sequence(x_k) \n')
[(1:k);x(1:k)]'
```

SOLUTION:

```
ans =

    1.0000    2.0000
    2.0000    1.5502
    3.0000    1.2841
    4.0000    1.1243
    5.0000    1.0276
    6.0000    0.9688
    7.0000    0.9329
    8.0000    0.9111
    9.0000    0.8977
   10.0000    0.8896
   11.0000    0.8846
   12.0000    0.8815
   13.0000    0.8797
   14.0000    0.8785
   15.0000    0.8778
```

LAB: 03

TITLE: REGULA FALSI METHOD

INTRODUCTION:

n mathematics, the regula falsi, method of false position, or false position method is a very old method for solving an equation with one unknown, that, in modified form, is still in use. In simple terms, the method is the trial and error technique of using test ("false") values for the variable and then adjusting the test value according to the outcome. This is sometimes also referred to as "guess and check". Versions of the method predate the advent of algebra and the use of equations.

The method of false position provides an exact solution for linear functions, but more direct algebraic techniques have supplanted its use for these functions. However, in numerical analysis, double false position became a root-finding algorithm used in iterative numerical approximation techniques.

Many equations, including most of the more complicated ones, can be solved only by iterative numerical approximation. This consists of trial and error, in which various values of the unknown quantity are tried. That trial-and-error may be guided by calculating, at each step of the procedure, a new estimate for the solution. There are many ways to arrive at a calculated-estimate and regula falsi provides one of these.

CODES OR REGULA FALSI METHOD:

```
%  Solution of nonlinear equations
% Regula-Falsi
clear all,close all
f = @(x) x.^3+4*x.^2-10;
a=1; b=2;
N=20;% maximum number of iterations
tol=10^(-5);
fa=feval(f,a);  % Initial guess
fb=feval(f,b);
if fa*fb>0
   %break
   disp('Wrong choice')
end

for k=1:N
   c=b-(b-a)*fb/(fb-fa);
   C(k)=c;
   fc=feval(f,c);
   Fc(k)=fc;
   if fc==0,break
```

```
    end
    if fb*fc>0;
        b=c;
        fb=fc;
    else
        a=c;
        fa=fc;
end
if abs(fc)<tol,break
end
end

fprintf('no of iteration(k), root sequence(x_k), function values fc\n')
[(1:k);C(1:k);Fc(1:k)]'
```

EXAMPLE ON REGULA FALSI METHOD:

```
%  Solution of nonlinear equations
% Regula-Falsi
clear all,close all
f = @(x) 2*x.^4+6*x.^3+x.^2-5*x-10;
a=4; b=7;
N=20;% maximum number of iterations
tol=10^(-5);
fa=feval(f,a);  % Initial guess
fb=feval(f,b);
if fa*fb>0
    %break
    disp('Wrong choice')
end


for k=1:N
    c=b-(b-a)*fb/(fb-fa);
    C(k)=c;
    fc=feval(f,c);
    Fc(k)=fc;
    if fc==0,break
    end
    if fb*fc>0;
        b=c;
        fb=fc;
else
a=c;
fa=fc;
end
if abs(fc)<tol,break
end
```

```
end


fprintf('no of iteration(k), root sequence(x_k), function values fc\n')
[(1:k);C(1:k);Fc(1:k)]'
```

SOLUTION:

```
untitled10
Wrong choice
no of iteration(k), root sequence(x_k), function values fc

ans =

    1.0000    3.5577  575.4475
    2.0000    2.7274  216.1874
    3.0000    2.3141  115.4969
    4.0000    2.0601   72.4260
    5.0000    1.8866   49.7474
    6.0000    1.7602   36.2208
    7.0000    1.6643   27.4534
    8.0000    1.5893   21.4236
    9.0000    1.5293   17.0889
   10.0000    1.4804   13.8648
   11.0000    1.4402   11.4011
   12.0000    1.4067    9.4770
   13.0000    1.3785    7.9475
   14.0000    1.3547    6.7135
   15.0000    1.3344    5.7056
   16.0000    1.3170    4.8737
   17.0000    1.3021    4.1812
   18.0000    1.2893    3.6002
   19.0000    1.2782    3.1096
   20.0000    1.2685    2.6931
```

ANOTHER EXAMPLE:

```
% Solution of nonlinear equations
% Regula-Falsi
clear all,close all
f = @(x) x*sin(x)-1;
a=0; b=2;
N=20;% maximum number of iterations
tol=10^(-5);
fa=feval(f,a); % Initial guess
fb=feval(f,b);
if fa*fb>0
%break
disp('Wrong choice')
end


for k=1:N
   c=b-(b-a)*fb/(fb-fa);
   C(k)=c;
   fc=feval(f,c);
   Fc(k)=fc;
   if fc==0,break
   end
   if fb*fc>0;
      b=c;
      fb=fc;
   else
      a=c;
      fa=fc;
end
if abs(fc)<tol,break
end
end


fprintf('no of iteration(k), root sequence(x_k), function values fc\n')
[(1:k);C(1:k);Fc(1:k)]'
```

SOLUTION:

untitled11

no of iteration(k), root sequence(x_k), function values fc

ans =

1.0000	1.0998	-0.0200
2.0000	1.1212	0.0098
3.0000	1.1142	0.0000

LAB:04

TTLE: SECANT METHOD:

INTRODUCTON:

In numerical analysis, the secant method is a root-finding algorithm that uses a succession of roots of secant lines to better approximate a root of a function f. The secant method can be thought of as a finite-difference approximation of Newton's method.

The secant method is very similar to the bisection method except instead of dividing each interval by choosing the midpoint the secant method divides each interval by the secant line connecting the endpoints.

CODES FOR SECANT METHOD

```
%  Solution of nonlinear equations
%Secant method method


clear all,close all
f = @(x) sqrt(x)-cos(x);
format short
N=20;% maximum number of iterations
tol=10^(-5);
  % Initial guess
x(1)=1.0;x(2)=0.5;
for k=3:N
x(k) = x(k-1)-feval(f,x(k-1))*(x(k-1)-x(k-2))/(feval(f,x(k-1))-feval(f,x(k-2)));
error=abs(x(k)-x(k-1));
if (error<tol), break;
end
end
fprintf('no of iteration(k), root sequence(x_k) \n')
[(1:k);x(1:k)]'
```

EXAMPLE ON THIS METHOD IN MATLAB:

```
%  Solution of nonlinear equations
%Secant method method
```

```
clear all,close all
f = @(x) sqrt(x)-sin(x);
format short
N=20;% maximum number of iterations
tol=10^(-5);
  % Initial guess
x(1)=1.0;x(2)=0.5;
for k=3:N
x(k) = x(k-1)-feval(f,x(k-1))*(x(k-1)-x(k-2))/(feval(f,x(k-1))-feval(f,x(k-2)));
error=abs(x(k)-x(k-1));
if (error<tol), break;
end
end
fprintf('no of iteration(k), root sequence(x_k) \n')
[(1:k);x(1:k)]'
```

SOLLUTION:

untitled12

no of iteration(k), root sequence(x_k)

ans =

```
   1.0000 + 0.0000i   1.0000 + 0.0000i
   2.0000 + 0.0000i   0.5000 + 0.0000i
   3.0000 + 0.0000i   2.1462 + 0.0000i
   4.0000 + 0.0000i  -0.4409 + 0.0000i
   5.0000 + 0.0000i   1.4746 + 2.2377i
   6.0000 + 0.0000i   0.1607 + 0.0761i
   7.0000 + 0.0000i   0.2288 + 0.2424i
   8.0000 + 0.0000i   0.3421 - 0.7550i
   9.0000 + 0.0000i   1.2073 + 0.4307i
  10.0000 + 0.0000i   1.5804 + 0.6270i
  11.0000 + 0.0000i   1.0720 + 0.5687i
  12.0000 + 0.0000i   1.0906 + 0.6594i
  13.0000 + 0.0000i   1.1391 + 0.6473i
  14.0000 + 0.0000i   1.1354 + 0.6508i
  15.0000 + 0.0000i   1.1352 + 0.6506i
  16.0000 + 0.0000i   1.1352 + 0.6506i
```

ANOTHER EXAMPLE:

```
%  Solution of nonlinear equations
%Secant method method



clear all,close all
f = @(x) x-0.25*sin(x)-0.5;
format short
N=20;% maximum number of iterations
tol=10^(-5);
  % Initial guess
x(1)=0.5;x(2)=0.8;
for k=3:N
x(k) = x(k-1)-feval(f,x(k-1))*(x(k-1)-x(k-2))/(feval(f,x(k-1))-feval(f,x(k-2)));
error=abs(x(k)-x(k-1));
if (error<tol), break;
end
end
fprintf('no of iteration(k), root sequence(x_k) \n')
[(1:k);x(1:k)]'
```

SOLUTION:

untitled13

no of iteration(k), root sequence(x_k)

ans =

1.0000	0.5000
2.0000	0.8000
3.0000	0.6495
4.0000	0.6516
5.0000	0.6516
6.0000	0.6516

LAB:05

TITLE: GUASS ELIMATION METHOD

INTRODUCTION:

Gaussian elimination, also known as row reduction, is an algorithm in linear algebra for solving a system of linear equations. It is usually understood as a sequence of operations performed on

the corresponding matrix of coefficients. This method can also be used to find the rank of a matrix, to calculate the determinant of a matrix, and to calculate the inverse of an invertible square matrix.

To perform row reduction on a matrix, one uses a sequence of elementary row operations to modify the matrix until the lower left-hand corner of the matrix is filled with zeros, as much as possible. There are three types of elementary row operations:

1. Swapping two rows,
2. Multiplying a row by a nonzero number,
3. Adding a multiple of one row to another row.

Using these operations, a matrix can always be transformed into an upper triangular matrix,

CODES FOR ELIMINATION METHOD:

```
clear all, close all, clc
% Gauss Elimination method
A=[4 1 2; 2 4 -1; 1 1 -3] % is N*N matrix
B=[9;-5;-9] % is N*1 martix
[N N]= size(A);
X=zeros(N,1);
C=zeros(1, N+1);
Aug= [A B]; % augmented matrix
for p=1:N-1
[Y,j]=max(abs(Aug(p:N,p))); %partial pivoting for column p
% interchange row p and j
C=Aug(p,:);
Aug(p,:)=Aug(j+p-1,:);
Aug(j+p-1,:)=C;
if Aug(p,p)==0, break
end
% elimination of column p
for k=p+1:N
m=Aug(k,p)/Aug(p,p);
Aug(k,p:N+1)=Aug(k,p:N+1)-m*Aug(p,p:N+1);
end
end
% Back substitution
X(N)=Aug(N,N+1)/Aug(N,N);
for k=N-1:-1:1
X(k)=(Aug(k,N+1)-Aug(k,k+1:N)*X(k+1:N))/Aug(k,k);
```

```
end
```

EXAMPLE ON ELIMINATION METHOD IN MATLAB

```
clear all, close all, clc
% Gauss Elimination method
A=[4 1 2; 2 4 -1; 1 1 -3] % is N*N matrix
B=[9;-5;-9] % is N*1 martix
[N N]= size(A);
X=zeros(N,1);
C=zeros(1, N+1);
Aug= [A B]; % augmented matrix
for p=1:N-1
[Y,j]=max(abs(Aug(p:N,p))); %partial pivoting for column p
% interchange row p and j
C=Aug(p,:);
Aug(p,:)=Aug(j+p-1,:);
Aug(j+p-1,:)=C;
if Aug(p,p)==0, break
end
% elimination of column p
for k=p+1:N
m=Aug(k,p)/Aug(p,p);
Aug(k,p:N+1)=Aug(k,p:N+1)-m*Aug(p,p:N+1);
end
end
% Back substitution
X(N)=Aug(N,N+1)/Aug(N,N);
for k=N-1:-1:1
X(k)=(Aug(k,N+1)-Aug(k,k+1:N)*X(k+1:N))/Aug(k,k);
end
```

SOLUTION:

```
A =

    4    1    2
    2    4   -1
    1    1   -3


B =

    9
```

-5
-9

ANOTHER EXAMPLE:

```
clear all, close all, clc
% Gauss Elimination method
A=[1 2; 2 7; 1 1] % is N*N matrix
B=[9 6;-5 1;-9 9] % is N*N martix
[N N]= size(A);
X=zeros(N,1);
C=zeros(1, N+1);
Aug= [A B]; % augmented matrix
for p=1:N-1
[Y,j]=max(abs(Aug(p:N,p))); %partial pivoting for column p
% interchange row p and j
C=Aug(p,:);
Aug(p,:)=Aug(j+p-1,:);
Aug(j+p-1,:)=C;
if Aug(p,p)==0, break
end
% elimination of column p
for k=p+1:N
m=Aug(k,p)/Aug(p,p);
Aug(k,p:N+1)=Aug(k,p:N+1)-m*Aug(p,p:N+1);
end
end
% Back substitution
X(N)=Aug(N,N+1)/Aug(N,N);
for k=N-1:-1:1
X(k)=(Aug(k,N+1)-Aug(k,k+1:N)*X(k+1:N))/Aug(k,k);
end
```

SOLUTION:

A =

1	2
2	7
1	1

B =

9 6
-5 1
-9 9

LAB:06

TITLE: TRAPIZIODAL RULE

INTRODUCTION:

n mathematics, and more specifically in numerical analysis, the trapezoidal rule (also known as the trapezoid rule or trapezium rule—see Trapezoid for more information on terminology) is a technique for approximating the definite integral.

$$\int_a^b f(x)\,dx$$

The trapezoidal rule works by approximating the region under the graph of the function f(x) as a trapezoid and calculating its area. It follows that

$$\int_a^b f(x)\,dx = (b-a).\frac{f(a)+f(b)}{3}$$

CODES FOR THE TRAPIZIADAL RULE:

```
% Composite Trapezoidal rule
a=0;b=1;n=6;
f =@(x) x.^3;
h=(b-a)/n;
s=0;
for k=1:n-1
   x=a+h*k;


   s=s+feval(f,x);
end


s=h*(feval(f,a)+feval(f,b))/2+h*s;
s
```

EXAMPLE ON THIS METHOD:

```
% Composite Trapezoidal rule
a=0;b=3;n=6;
f =@(x) x.^2 / 1+x.^3 ;
h=(b-a)/n;
s=0;
for k=1:n-1
   x=a+h*k;


   s=s+feval(f,x);
end


s=h*(feval(f,a)+feval(f,b))/2+h*s;
s
```

SOLUTION:

```
s =

  29.9375
```

ANOTHER EXAMPLE:

```
% Composite Trapezoidal rule
a=3;b=7;n=6;
f =@(x) 1-x.^3 /4 ;
h=(b-a)/n;
s=0;
for k=1:n-1
   x=a+h*k;


   s=s+feval(f,x);
end


s=h*(feval(f,a)+feval(f,b))/2+h*s;
s
```

SOLUTION:

s =

-142.1111

LAB:07

TITLE: SIMPSON METHOD

INTRODUCTION:

In numerical integration, Simpson's rules are several approximations for definite integrals, named after Thomas Simpson (1710–1761).

The most basic of these rules, called Simpson's 1/3 rule, or just Simpson's rule, reads

$$\int_a^b f(x) = \frac{b-a}{6} ¿¿ \text{f(a)} + 4\text{f}\frac{a+b}{2} + f(b)$$

if the 1/3 rule is applied to n equal subdivisions of the integration range $[a,b]$, one obtains the composite Simpson's rule. Points inside the integration range are given alternating weights 4/3 and 2/3.

Simpson's 3/8 rule, also called Simpson's second rule requests one more function evaluation inside the integration range, and is exact if f is a polynomial up to cubic degree.

CODES FOR SIMPSON RULE:

```
a=-0.25; b=0.25; n=4;
f =@(x) (cos(x)).^2;
h=(b-a)/n;
s1=0;
s2=0;
for k=1:n
   x=a+h*(2*k-1);
   s1=s1+feval(f,x);
end
for k=1:n-1
   x=a+h*2*k;
   s2=s2+feval(f,x);
end
s=h/3*(feval(f,a)+feval(f,b)+4*s1+2*s2);
s
```

EXAMLE ON THIS METHOD:

```
a=-0; b=90; n=6;
f =@(x) x.^2/1+x.^3;
h=(b-a)/n;
s1=0;
s2=0;
for k=1:n
   x=a+h*(2*k-1);
   s1=s1+feval(f,x);
end
for k=1:n-1
   x=a+h*2*k;
   s2=s2+feval(f,x);
end
s=h/3*(feval(f,a)+feval(f,b)+4*s1+2*s2);
s
```

SOLUTION:

untitled7

```
s =

  374.4000
```

ANOTHER EXAMPLE:

```
a=-0; b=90; n=6;
f =@(x) sin(x)^0.5;
h=(b-a)/n;
s1=0;
s2=0;
for k=1:n
   x=a+h*(2*k-1);
   s1=s1+feval(f,x);
end
for k=1:n-1
   x=a+h*2*k;
   s2=s2+feval(f,x);
end
s=h/3*(feval(f,a)+feval(f,b)+4*s1+2*s2);
s
```

SOLUTION:

s =

82.3028 +56.0736i