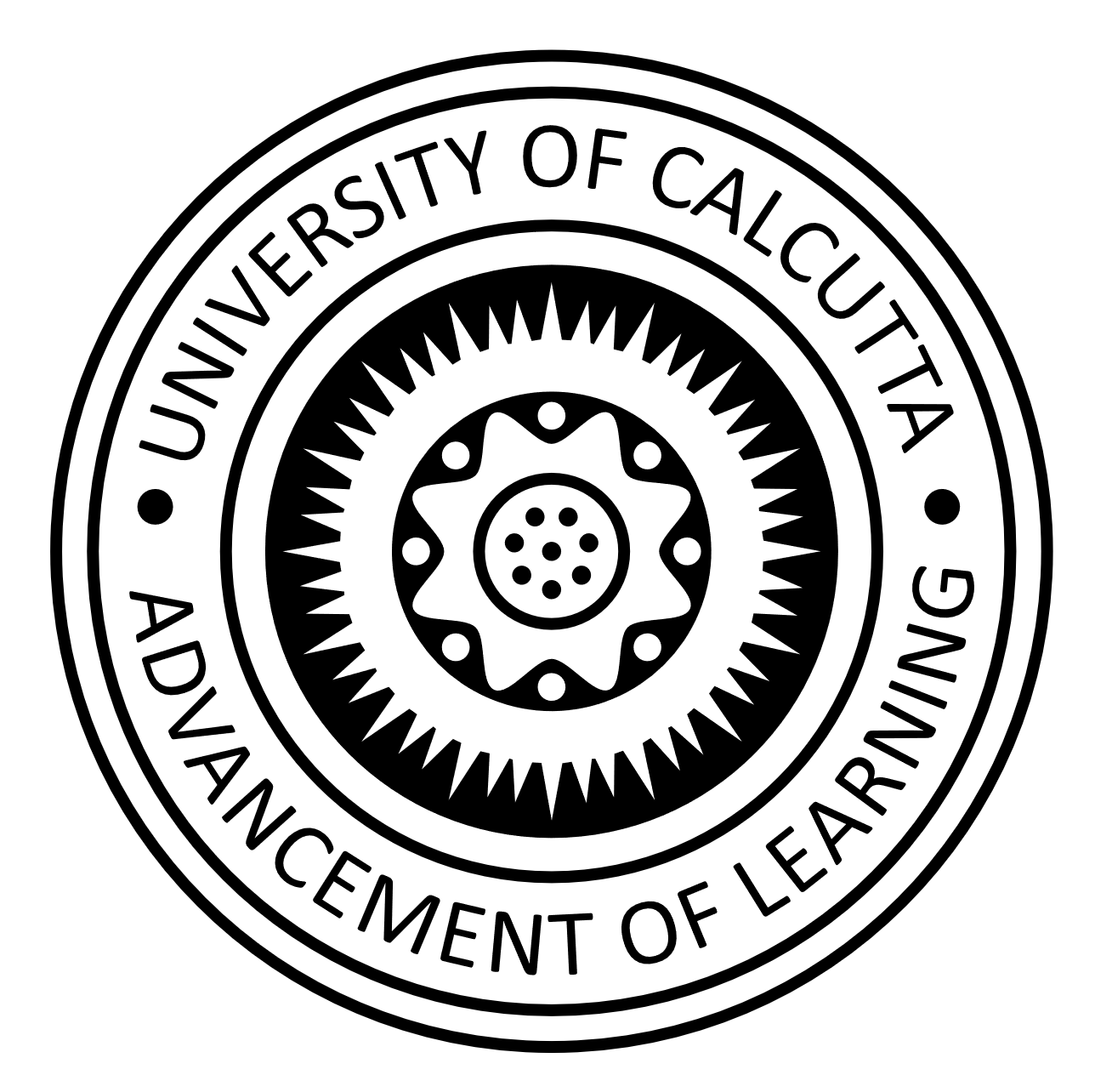
B.A./B.Sc. Semester 6 HONOURS Examination 2024 (Under CBCS)

MTMA CC14 Practical Notebook



**CU Roll Number:** 223-1111-0461-21

**CU Registration Number:** 213223-21-0115

# Contents

1. Calculate the sum
2. Enter 100 integers into an array and sort them in an ascending order.
3. Solution of transcendental and algebraic equations by
4. Bisection method
5. Newton Raphson method (Simple root, multiple roots, complex roots)
6. Secant method
7. Regula-Falsi method
8. Solution of system of linear equations
9. LU decomposition method
10. Gaussian elimination method
11. Gauss-Jacobi method
12. Gauss-Seidel method
13. Interpolation
14. Lagrange Interpolation
15. Newton’s forward, backward and divided difference interpolations
16. Numerical Integration
    * 1. Trapezoidal Rule
      2. Simpson’s one third rule
      3. Weddle’s Rule
      4. Gauss Quadrature
17. Method of finding Eigenvalue by Power method (up to 4 × 4)
18. Fitting a Polynomial Function (up to third degree)
19. Solution of ordinary differential equations
    * 1. Euler method
      2. Modified Euler method
      3. Runge-Kutta method (order 4)
      4. The method of successive approximations (Picard)

# Assignment 1

Write a C program to calculate the sum correct up to 4 decimal places of

where is the last digit of your university roll number.

## Program

#include <stdio.h>

int main()

{

int i, R = 4, N;

double sum = 0.0;

printf("Input the value of R: ");

scanf("%d", &R);

N = 10 + R;

for (i = 0; i < N; i++)

sum = sum + 1.0/(i+1);

printf("The sum of the series correct upto 4D is %.4f", sum);

return 0;

}

## Output

Input the value of R: 4

The sum of the series correct upto 4D is 3.2516

# Assignment 2

Write a C program to enter 10 integers into an array and sort them in ascending order.

## Program

#include <stdio.h>

#define N 10

int main()

{

int a[N] = {0};

int i, j, temp;

printf("Input %d integers:\n", N);

for (i = 0; i < N; i++)

scanf("%d", &a[i]);

for (i = 0; i < N-1; i++)

{

for (j = i+1; j < N; j++)

{

if (a[i] > a[j])

{

temp = a[i];

a[i] = a[j];

a[j] = temp;

}

}

}

printf("The integers in ascending order are:\n");

for (i = 0; i < N; i++)

printf("%d \t", a[i]);

printf("\n");

return 0;

}

## Output

Input 10 integers: 12 -13 56 0 1 -34 0 8 9 11

The integers in ascending order are:

-34 -13 0 0 1 8 9 11 12 56

# Assignment 3

Find a root of the following equation correct to 5D by bisection method

where and is the last digit of your university roll number.

## Program

#include <stdio.h>

#include <math.h>

#define R 4

double f(double x)

{

double a = (R / 10.0) + 0.2;

return exp(-a\*x) - 10\*a\*log(x) - 0.8;

}

int main()

{

double a0 = 0, b0 = 1, a, b, x, error = 0.00001;

printf("Root finding by Bisection Method\n");

printf("Input the root containing interval\n");

printf("The lower bound: ");

scanf("%lf", &a0);

printf("The upper bound: ");

scanf("%lf", &b0);

printf("\n");

if (f(a0) \* f(b0) > 0)

{

printf("The interval (%f, %f) contains no root.\n", a0, b0);

return 0;

}

a = a0;

b = b0;

printf("a \t\t b \t\t x \t\t f(x)\n");

do

{

x = (a + b) / 2;

printf("%f \t %f \t %f \t %f\n", a, b, x, f(x));

if (f(a) \* f(x) > 0)

a = x;

else

b = x;

} while (fabs(f(x)) >= error);

printf("\nThe root in the interval (%f,%f) correct to 5D ");

printf("is %.5f\n", a0, b0, x);

return 0;

}

## Output

Root finding by Bisection Method

Input the root containing interval

The lower bound: 0

The upper bound: 2

a b x f(x)

0.000000 2.000000 1.000000 -0.251188

0.000000 1.000000 0.500000 4.099701

0.500000 1.000000 0.750000 1.563721

0.750000 1.000000 0.875000 0.592744

0.875000 1.000000 0.937500 0.157014

0.937500 1.000000 0.968750 -0.050309

0.937500 0.968750 0.953125 0.052521

0.953125 0.968750 0.960938 0.000902

0.960938 0.968750 0.964844 -0.024754

0.960938 0.964844 0.962891 -0.011939

0.960938 0.962891 0.961914 -0.005522

0.960938 0.961914 0.961426 -0.002311

0.960938 0.961426 0.961182 -0.000705

0.960938 0.961182 0.961060 0.000098

0.961060 0.961182 0.961121 -0.000303

0.961060 0.961121 0.961090 -0.000102

0.961060 0.961090 0.961075 -0.000002

The root in the interval (0.000000,2.000000) correct to 5D is 0.96107

# Assignment 3

1. Find a root of the above equation which lies in .

## Program

#include <stdio.h>

#include <math.h>

double f(double x)

{

return pow(x,5) + 0.7\*pow(x,4) - 7.77\*pow(x,3) + 22.041\*pow(x,2)

- 17.6824\*x - 276.46048;

}

double df(double x)

{

return 5\*pow(x,4) + 4\*0.7\*pow(x,3) - 3\*7.77\*pow(x,2)

+ 2\*22.041\*x - 17.6824;

}

int main()

{

double x = 0, error = 0.0000001;

printf("Root finding by Newton-Raphson Method\n");

printf("Enter initial approximation of the root: ");

scanf("%lf", &x);

printf("\nx \t\t f(x)\n");

printf("%f \t %f\n", x, f(x));

while (fabs(f(x)) >= error)

{

x = x - f(x)/df(x);

printf("%f \t %f\n", x, f(x));

}

printf("\n");

printf("The root of the equation correct upto 6D is %.6f\n", x);

return 0;

}

## Output

Root finding by Newton-Raphson Method

Enter initial approximation of the root: 2.7

x f(x)

2.700000 -135.771040

3.238257 66.620656

3.111487 5.094133

3.100086 0.037737

3.100000 0.000002

3.100000 0.000000

The root of the equation correct upto 6D is 3.100000

1. Find a double root of the above equation which lies in .

## Program

#include <stdio.h>

#include <math.h>

double f(double x)

{

return pow(x,5) + 0.7\*pow(x,4) - 7.77\*pow(x,3) + 22.041\*pow(x,2)

- 17.6824\*x - 276.46048;

}

double df(double x)

{

return 5\*pow(x,4) + 4\*0.7\*pow(x,3) - 3\*7.77\*pow(x,2)

+ 2\*22.041\*x - 17.6824;

}

int main()

{

int m = 2;

double x = 0, error = 0.0000001;

printf("Root finding by Newton-Raphson Method\n");

printf("Enter initial approximation of the root: ");

scanf("%lf", &x);

printf("Enter the multiplicity of the root: ");

scanf("%d", &m);

printf("\nx \t\t f(x)\n");

printf("%f \t %f\n", x, f(x));

while (fabs(f(x)) >= error)

{

x = x - m \* f(x)/df(x);

printf("%f \t %f\n", x, f(x));

}

printf("\n");

printf("The root of the equation upto 5D is %.5f with ");

printf("multiplicity %d\n", x, m);

return 0;

}

## Output

Root finding by Newton-Raphson Method

Enter initial approximation of the root: -3.4

Enter the multiplicity of the root: 2

x f(x)

-3.400000 -16.965000

-3.119800 -0.064618

-3.100095 -0.000001

-3.100000 0.000000

The root of the equation upto 5D is -3.10000 with multiplicity 2

1. Find a pair of complex roots of the above equation, one of them has the initial value .

## Program

#include <stdio.h>

#include <math.h>

#include <complex.h>

double complex f(double complex z)

{

return cpow(z,5) + 0.7\*cpow(z,4) - 7.77\*cpow(z,3)

+ 22.041\*cpow(z,2) - 17.6824\*z - 276.46048;

}

double complex df(double complex z)

{

return 5\*cpow(z,4) + 4\*0.7\*cpow(z,3) - 3\*7.77\*cpow(z,2)

+ 2\*22.041\*z - 17.6824;

}

int main()

{

double x = 0, y = 0;

double complex z;

double error = 0.0000001;

printf("Root finding by Newton-Raphson Method\n");

printf("Enter initial approximation of the root\n");

printf("Enter the real part: ");

scanf("%lf", &x);

printf("Enter the imaginary part: ");

scanf("%lf", &y);

z = x + I\*y;

printf("\nz \t\t\t f(z)\n");

printf("%f + i\*%f \t %f + i\*%f\n", creal(z), cimag(z),

creal(f(z)), cimag(f(z)));

while (cabs(f(z)) >= error)

{

z = z - f(z)/df(z);

printf("%f + i\*%f \t %f + i\*%f\n", creal(z), cimag(z),

creal(f(z)), cimag(f(z)));

}

printf("\n");

printf("The roots of the equation upto 5D are ");

printf("%.5f + i\*%.5f) and (%.5f - i\*%.5f)\n",

creal(z), cimag(z), creal(z), cimag(z));

return 0;

}

## Output

Root finding by Newton-Raphson Method

Enter initial approximation of the root

Enter the real part: 1.4299

Enter the imaginary part: 2.1520

z f(z)

1.429900 + i\*2.152000 -225.021076 + i\*-62.451576

0.728407 + i\*2.939392 -52.493383 + i\*243.583173

1.111315 + i\*2.682717 -61.025890 + i\*27.969082

1.213853 + i\*2.807117 4.713991 + i\*-6.243556

1.200148 + i\*2.799944 -0.014576 + i\*-0.077478

1.200000 + i\*2.800000 -0.000008 + i\*0.000002

1.200000 + i\*2.800000 -0.000000 + i\*-0.000000

The roots of the equation upto 5D are (1.20000 + i\*2.80000) and (1.20000 - i\*2.80000)

# Assignment 3

Find a positive root of the following equation correct upto 6D by secant method

Where is the last digit of your university roll number.

## Program

#include <stdio.h>

#include <math.h>

#define R 4

double f(double x)

{

return pow(x,2) \* tanh(x) - exp((1+R/20)\*sin(x)) - 3;

}

int main()

{

double x0 = 0, x1 = 1, x, error = 0.0000001;

printf("Root finding by Secant Method\n");

printf("Input two initial approximations of the root: ");

scanf("%lf%lf", &x0, &x1);

printf("\n");

printf("x \t\t f(x)\n");

do

{

x = x1 - f(x1)\*(x1-x0) / (f(x1)-f(x0));

printf("%f \t %f\n", x, f(x));

x0 = x1;

x1 = x;

} while (fabs(f(x)) >= error);

printf("\nThe root correct to 6D is %.6f\n", x);

return 0;

}

## Output

Root finding by Secant Method

Input two initial approximations of the root: 1 5

x f(x)

1.696634 -3.005450

2.099890 -1.091674

2.329919 0.261049

2.285527 -0.011541

2.287407 -0.000102

2.287424 0.000000

The root correct to 6D is 2.287424

# Assignment 3

Find a positive root of the following equation correct upto 5D by Regula Falsi method

where and is the last digit of your university roll number.

## Program

#include <stdio.h>

#include <math.h>

#define R 4

double f(double x)

{

double d = 1 + R/10.0;

return d\*pow(x,2) + x\*log(1+x) - 2;

}

int main()

{

double a0 = 0, b0 = 1, a, b, x, error = 0.0000001;

printf("Root finding by Regula-Falsi Method\n");

printf("Input the root containing interval\n");

printf("The lower bound: ");

scanf("%lf", &a0);

printf("The upper bound: ");

scanf("%lf", &b0);

printf("\n");

if (f(a0) \* f(b0) > 0)

{

printf("The interval (%f, %f) contains no root.\n", a0, b0);

return 0;

}

a = a0;

b = b0;

printf("a \t\t b \t\t x \t\t f(x)\n");

do

{

x = b - f(b) \* (b-a) / (f(b) - f(a));

printf("%f \t %f \t %f \t %f\n", a, b, x, f(x));

if (f(a) \* f(x) > 0)

a = x;

else

b = x;

} while (fabs(f(x)) >= error);

printf("\nThe root in the interval (%f,%f) correct to 5D is "

%.5f\n", a0, b0, x);

return 0;

}

## Output

Root finding by Regula-Falsi Method

Input the root containing interval

The lower bound: 0

The upper bound: 1

a b x f(x)

0.000000 1.000000 0.955499 -0.081029

0.955499 1.000000 0.976201 -0.000877

0.976201 1.000000 0.976424 -0.000009

0.976424 1.000000 0.976426 -0.000000

The root in the interval (0.000000,1.000000) correct to 5D is 0.97643

# Assignment 4

Find the solution of the following system of linear equations by LU decomposition correct to 4D.

Where and is the last digit of your university roll number.

## Program

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#define N 10

/\*

Decomposes matrix a into lower and upper triangular matrix l

and u respectively. The diagonal elements of matrix u are 1.

exit(EXIT\_FAILURE) if l[i][i] is zero.

\*/

void lu\_decompose(double a[][N], double l[][N], double u[][N],

int n)

{

for (int i = 0; i < n; i++)

u[i][i] = 1;

for (int i = 0; i < n; i++)

{

// compute ith row of l

for (int j = 0; j <= i; j++)

{

double sum = 0;

for (int k = 0; k < i; k++)

sum += l[i][k] \* u[k][j];

l[i][j] = a[i][j] - sum;

}

// compute ith row of u

for (int j = i+1; j < n; j++)

{

double sum = 0;

for (int k = 0; k < i; k++)

sum += l[i][k] \* u[k][j];

if (l[i][i] == 0)

{

printf("l[%d][%d] is zero. Cannot divide by "

"zero\n", i, i);

exit(EXIT\_FAILURE);

}

u[i][j] = (a[i][j] - sum) / l[i][i];

}

}

}

/\*

a must be a lower triangular matrix.

exit(EXIT\_FAILURE) if a[i][i] == 0

\*/

void forward\_substitute(double a[][N], double b[], double x[],

int n)

{

for (int i = 0; i < n; i++)

{

double root = b[i];

for (int j = 0; j < i; j++)

root = root - a[i][j]\*x[j];

if (a[i][i] == 0)

{

printf("The diagonal elements must be numerically

largest\n");

printf("a[%d][%d] is zero\n", i, i);

exit(EXIT\_FAILURE);

}

x[i] = root / a[i][i];

}

}

/\*

a must be an upper triangular matrix.

exit(EXIT\_FAILURE) if a[i][i] == 0

\*/

void back\_substitute(double a[][N], double b[], double x[], int n)

{

for (int i = n-1; i >= 0; i--)

{

double root = b[i];

for (int j = i+1; j < n; j++)

root = root - a[i][j]\*x[j];

if (a[i][i] == 0)

{

printf("The diagonal elements must be numerically"

"largest\n");

printf("a[%d][%d] is zero\n", i, i);

exit(EXIT\_FAILURE);

}

x[i] = root / a[i][i];

}

}

int main()

{

int n = 4;

double a[N][N] = {0}, b[N] = {0};

double l[N][N] = {0}, u[N][N] = {0};

double x[N] = {0}, z[N] = {0};

printf("Enter the number of equations present: ");

scanf("%d", &n);

printf("\n");

if (n > N)

{

printf("Too many equations\n");

exit(EXIT\_FAILURE);

}

printf("The diagonal elements must be numerically largest\n");

printf("Enter the coefficients of the system:\n");

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

scanf("%lf", &a[i][j]);

printf("\nEnter the right-hand side of the system: ");

for (int i = 0; i < n; i++)

scanf("%lf", &b[i]);

printf("\n");

lu\_decompose(a, l, u, n);

printf("The lower triangular matrix is:\n");

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

printf("%.6f\t", l[i][j]);

printf("\n");

}

printf("\n");

printf("The upper triangular matrix is:\n");

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

printf("%.6f\t", u[i][j]);

printf("\n");

}

printf("\n");

forward\_substitute(l, b, z, n);

back\_substitute(u, z, x, n);

printf("The solution for the given system correct to 6D is:\n");

for (int i = 0; i < n; i++)

printf("Root %d: %.6f\n", i+1, x[i]);

printf("\n");

return 0;

}

## Output

Enter the number of equations present: 4

The diagonal elements must be numerically largest

Enter the coefficients of the system:

1.5161 0.1254 0.1397 0.1490

0.1582 1.5675 0.1768 0.1871

0.1968 0.2071 1.6168 0.2271

0.2368 0.2471 0.2568 1.6671

Enter the right-hand side of the system: 1.5471 1.6471 1.7471 1.8471

The lower triangular matrix is:

1.516100 0.000000 0.000000 0.000000

0.158200 1.554415 0.000000 0.000000

0.196800 0.190822 1.578751 0.000000

0.236800 0.227514 0.211236 1.593738

The upper triangular matrix is:

1.000000 0.082712 0.092144 0.098278

0.000000 1.000000 0.104363 0.110365

0.000000 0.000000 1.000000 0.118257

0.000000 0.000000 0.000000 1.000000

The solution for the given system correct to 6D is:

Root 1: 0.781478

Root 2: 0.871437

Root 3: 0.874727

Root 4: 0.878007

# Assignment 4

Solve the following system of linear equations by Gaussian elimination method correct to 6D. where and ;

Where and is the last digit of your university roll number.

## Program

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <math.h>

#define N 10

/\*

Converts a n\*n matrix into an upper triangular matrix

exit(EXIT\_FAILURE) if the diagonal elements are zero.

\*/

void to\_upper\_triangular(double a[][N+1], int n)

{

for (int k = 0; k < n; k++)

{

for (int i = k+1; i < n; i++)

{

if (a[k][k] == 0)

{

printf("The diagonal elements must be numerically "

"largest\n");

printf("a[%d][%d] is zero\n", k, k);

exit(EXIT\_FAILURE);

}

double m = a[i][k]/a[k][k];

for (int j = k; j < n+1; j++)

a[i][j] = a[i][j] - m \* a[k][j];

}

}

}

/\*

a must be augmented upper triangular matrix.

exit(EXIT\_FAILURE) if a[i][i] == 0

\*/

void back\_substitute(double a[][N+1], double x[], int n)

{

for (int i = n-1; i >= 0; i--)

{

double root = a[i][n];

for (int j = i+1; j < n; j++)

root = root - a[i][j]\*x[j];

if (a[i][i] == 0)

{

printf("The diagonal elements must be numerically"

"largest\n");

printf("a[%d][%d] is zero\n", i, i);

exit(EXIT\_FAILURE);

}

x[i] = root / a[i][i];

}

}

int main()

{

int n = 4;

double a[N][N+1] = {0}, b[N] = {0}, x[N] = {0};

printf("Solution of system of linear equations by Gaussian "

"Elimination\n");

printf("Enter the number of equations present: ");

scanf("%d", &n);

printf("\n");

if (n > N)

{

printf("Too many equations\n");

exit(EXIT\_FAILURE);

}

printf("The diagonal elements must be numerically largest\n");

printf("Enter the coefficients of the system:\n");

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

scanf("%lf", &a[i][j]);

printf("\nEnter the right-hand side of the system: ");

for (int i = 0; i < n; i++)

scanf("%lf", &b[i]);

printf("\n");

// augmented matrix

for (int i = 0; i < n; i++)

a[i][n] = b[i];

// upper triangular matrix

to\_upper\_triangular(a, n);

printf("The augmented upper triangular matrix is:\n");

for (int i = 0; i < n; i++)

{

for (int j = 0; j <= n; j++)

printf("%.6f\t", a[i][j]);

printf("\n");

}

printf("\n");

// back substitution

back\_substitute(a, x, n);

printf("The solution for the given system correct to 6D is:\n");

for (int i = 0; i < n; i++)

printf("Root %d: %.6f\n", i+1, x[i]);

printf("\n");

return 0;

}

## Output

Solution of system of linear equations by Gaussian Elimination

Enter the number of equations present: 4

The diagonal elements must be numerically largest

Enter the coefficients of the system:

8.97 1.99 1.04 -2.02

1.64 8.03 2.29 0.82

2.90 0.86 9.55 0.96

0.70 -2.00 1.82 7.89

Enter the right-hand side of the system: 3.49 1.90 -4.00 2.55

The augmented upper triangular matrix is:

8.970000 1.990000 1.040000 -2.020000 3.490000

0.000000 7.666165 2.099855 1.189320 1.261918

0.000000 0.000000 9.154430 1.579458 -5.163976

0.000000 0.000000 0.000000 7.980138 3.946321

The solution for the given system correct to 6D is:

Root 1: 0.516771

Root 2: 0.265773

Root 3: -0.649417

Root 4: 0.494518

# Assignment 4

Solve the following system of linear equations by Gauss Jacobi method correct to 6D.

and . Here and is the last digit of your university roll number.

## Program

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <math.h>

#define N 10

int main()

{

int n = 4;

double a[N][N] = {0}, b[N] = {0};

double x0[N] = {0}, x1[N] = {0};

double error = 0.0000001;

bool flag = false;

printf("Solution of system of linear equations by Gauss Jacobi"

"Method\n");

printf("Enter the number of equations present: ");

scanf("%d", &n);

printf("\n");

if (n > N)

{

printf("Too many equations\n");

exit(EXIT\_FAILURE);

}

printf("The system must be diagonally dominant.\n");

printf("Enter the coefficients of the system:\n");

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

scanf("%lf", &a[i][j]);

printf("\nEnter the right-hand side of the system: ");

for (int i = 0; i < n; i++)

scanf("%lf", &b[i]);

printf("\nEnter initial approximation of the roots:\n");

for (int i = 0; i < n; i++)

{

printf("Root %d: ", i+1);

scanf("%lf", &x0[i]);

}

printf("\n");

while (flag == false)

{

for (int i = 0; i < n; i++)

{

x1[i] = b[i];

for (int j = 0; j < n; j++)

{

if (i != j)

x1[i] = x1[i] - a[i][j]\*x0[j];

}

if (a[i][i] == 0)

{

printf("The coefficient matrix must be ");

printf("diagonally dominant\n A[%d][%d] is zero\n,

i, i);

exit(EXIT\_FAILURE);

}

x1[i] /= a[i][i];

}

for (int i = 0; i < n; i++)

{

if (fabs(x1[i]-x0[i]) < error)

flag = true;

x0[i] = x1[i];

}

}

printf("The solution for the given system correct to 6D is:\n");

for (int i = 0; i < n; i++)

printf("Root %d: %.6f\n", i+1, x0[i]);

printf("\n");

return 0;

}

## Output

Solution of system of linear equations by Gauss Jacobi Method

Enter the number of equations present: 4

The system must be diagonally dominant.

Enter the coefficients of the system:

7.60 1.28 1.34 -1.70

2.20 8.94 1.31 0.84

2.24 -0.75 5.96 7.76

2.12 1.84 -2.55 9.64

Enter the right-hand side of the system: -1.65 3.21 -8.44 31.17

Enter initial approximation of the roots:

Root 1: 0

Root 2: 0

Root 3: 0

Root 4: 0

The solution for the given system correct to 6D is:

Root 1: 0.821157

Root 2: 0.579464

Root 3: -4.078072

Root 4: 1.863470

# Assignment 4

Solve the following system of linear equations by Gauss Siedel method correct to 4D.

where and ;

Here and is the last digit of your university roll number.

## Program

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <math.h>

#define N 10

int main()

{

int n = 4;

double a[N][N] = {0}, b[N] = {0};

double x0[N] = {0}, x1[N] = {0};

double error = 0.0000001;

bool flag = false;

printf("Solution of system of linear equations by Gauss Siedel "

"Method\n");

printf("Enter the number of equations present: ");

scanf("%d", &n);

printf("\n");

if (n > N)

{

printf("Too many equations\n");

exit(EXIT\_FAILURE);

}

printf("The system must be diagonally dominant.\n");

printf("Enter the coefficients of the system:\n");

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

scanf("%lf", &a[i][j]);

printf("\nEnter the right-hand side of the system: ");

for (int i = 0; i < n; i++)

scanf("%lf", &b[i]);

printf("\nEnter initial approximation of the roots:\n");

for (int i = 0; i < n; i++)

{

printf("Root %d: ", i+1);

scanf("%lf", &x0[i]);

}

printf("\n");

while (flag == false)

{

for (int i = 0; i < n; i++)

{

x1[i] = b[i];

for (int j = 0; j < n; j++)

{

if (j < i)

x1[i] = x1[i] - a[i][j]\*x1[j];

else if (j > i)

x1[i] = x1[i] - a[i][j]\*x0[j];

}

if (a[i][i] == 0)

{

printf("The coefficient matrix must be diagonally");

printf(" dominant\n A[%d][%d] is zero\n", i, i);

exit(EXIT\_FAILURE);

}

x1[i] /= a[i][i];

}

for (int i = 0; i < n; i++)

{

if (fabs(x1[i]-x0[i]) < error)

flag = true;

x0[i] = x1[i];

}

}

printf("The solution for the given system correct to 4D is:\n");

for (int i = 0; i < n; i++)

printf("Root %d: %.4f\n", i+1, x0[i]);

printf("\n");

return 0;

}

## Output

Solution of system of linear equations by Gauss Siedel Method

The system must be diagonally dominant.

Enter the coefficients of the system:

9.31 2.34 1.42 -0.81

2.52 10.66 -2.22 -0.12

1.14 0.35 10.98 2.98

0.23 -2.38 0.59 8.24

Enter the right-hand side of the system: 4.58 12.44 -10.36 12.78

Enter initial approximation of the roots:

Root 1: 0

Root 2: 0

Root 3: 0

Root 4: 0

The solution for the given system correct to 4D is:

Root 1: 0.7115

Root 2: 0.6988

Root 3: -1.5399

Root 4: 1.8432

# Assignment 5

Compute the values of f(x) at and at by Lagrange’s interpolation formula from the following table.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

where is the last digit of your university roll number.

## Program

#include <stdio.h>

#include <stdlib.h>

#define N 10

int main()

{

double x[N] = {0}, y[N] = {0};

double x\_inter = 0, y\_inter = 0;

int n = 8;

printf("Interpolation using Lagrange's formula\n");

printf("Enter the number of points: ");

scanf("%d", &n);

if (n > N)

{

printf("Too many points\n");

exit(EXIT\_FAILURE);

}

printf("\n");

printf("Enter the points:\n");

for(int i = 0; i < n; i++)

{

printf("%d x: ", i+1);

scanf("%lf", &x[i]);

printf(" y: ");

scanf("%lf", &y[i]);

}

printf("\n");

printf("Enter the value for which interpolation is required: ");

scanf("%lf", &x\_inter);

for (int i = 0; i < n; i++)

{

double prod = 1;

for (int j = 0; j < n; j++)

{

if (i != j)

{

if (x[i] == x[j])

{

printf("The values of x\_%d and x\_%d cannot be"

" same!\n", i+1, j+1);

exit(EXIT\_FAILURE);

}

prod \*= (x\_inter - x[j]) / (x[i] - x[j]);

}

}

y\_inter += prod \* y[i];

}

printf("The functional value at x = %f is %f\n", x\_inter,

y\_inter);

return 0;

}

## Output

### Output 1

Interpolation using Lagrange's formula

Enter the number of points: 8

Enter the points:

1 x: 1.12

y: 0.307961

2 x: 1.16

y: 0.311448

3 x: 1.20

y: 0.321976

4 x: 1.26

y: 0.334217

5 x: 1.32

y: 0.342368

6 x: 1.37

y: 0.357905

7 x: 1.43

y: 0.370674

8 x: 1.49

y: 0.381982

Enter the value for which interpolation is required: 1.2249999

The functional value at x = 1.225000 is 0.328428

### Output 2

Interpolation using Lagrange's formula

Enter the number of points: 8

Enter the points:

1 x: 1.12

y: 0.307961

2 x: 1.16

y: 0.311448

3 x: 1.20

y: 0.321976

4 x: 1.26

y: 0.334217

5 x: 1.32

y: 0.342368

6 x: 1.37

y: 0.357905

7 x: 1.43

y: 0.370674

8 x: 1.49

y: 0.381982

Enter the value for which interpolation is required: 1.415

The functional value at x = 1.415000 is 0.370010

# Assignment 5

Compute the values of f(x) at from the following table by Newton’s forward interpolation formula.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

where is the last digit of your university roll number.

## Program

#include <stdio.h>

#include <stdlib.h>

#define N 10

int main()

{

double x[N] = {0}, y[N][N] = {0};

double x\_inter = 0, y\_inter = 0, u = 0, h = 0, prod = 1;

int n = 10;

printf("Interpolation using Newton's Forward Interpolation"

" formula\n");

printf("Enter the number of points: ");

scanf("%d", &n);

if (n > N)

{

printf("Too many points\n");

exit(EXIT\_FAILURE);

}

printf("\n");

printf("The x points must be equispaced\n");

printf("Enter the points:\n");

for(int i = 0; i < n; i++)

{

printf("%d x: ", i+1);

scanf("%lf", &x[i]);

printf(" y: ");

scanf("%lf", &y[i][0]);

}

printf("\n");

printf("Enter the value for which interpolation is required: ");

scanf("%lf", &x\_inter);

// difference table

for (int j = 1; j < n; j++)

{

for (int i = 0; i < n-j; i++)

y[i][j] = y[i+1][j-1] - y[i][j-1];

}

h = x[1] - x[0];

u = (x\_inter - x[0]) / h;

y\_inter = y[0][0];

for (int j = 1; j < n; j++)

{

prod \*= (u - (j-1)) / j;

y\_inter += prod \* y[0][j];

}

printf("The functional value at x = %f is %.10f\n", x\_inter,

y\_inter);

return 0;

}

## Output

Interpolation using Newton's Forward Interpolation formula

Enter the number of points: 10

The x points must be equispaced

Enter the points:

1 x: 0.20

y: 1.2922071606

2 x: 0.35

y: 1.3397750591

3 x: 0.50

y: 1.3890939964

4 x: 0.65

y: 1.4402284308

5 x: 0.80

y: 1.4932451930

6 x: 0.95

y: 1.5482135742

7 x: 1.10

y: 1.6052054161

8 x: 1.25

y: 1.6642952050

9 x: 1.40

y: 1.7255601691

10 x: 1.55

y: 1.7890803797

Enter the value for which interpolation is required: 0.25

The functional value at x = 0.250000 is 1.3078724509

# Assignment 5

Compute the value of at from the following table using Newton’s backward interpolation formula.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Where is the last digit of your university roll number.

## Program

#include <stdio.h>

#include <stdlib.h>

#define N 10

int main()

{

double x[N] = {0}, y[N][N] = {0};

double x\_inter = 0, y\_inter = 0, u = 0, h = 0, prod = 1;

int n = 10;

printf("Interpolation using Newton's Backward Interpolation"

" formula\n");

printf("Enter the number of points: ");

scanf("%d", &n);

if (n > N)

{

printf("Too many points\n");

exit(EXIT\_FAILURE);

}

printf("\n");

printf("The points must be equispaced\n");

printf("Enter the points:\n");

for(int i = 0; i < n; i++)

{

printf("%d x: ", i+1);

scanf("%lf", &x[i]);

printf(" y: ");

scanf("%lf", &y[i][0]);

}

printf("\n");

printf("Enter the value for which interpolation is required: ");

scanf("%lf", &x\_inter);

// difference table

for (int j = 1; j < n; j++)

{

for (int i = 0; i < n-j; i++)

y[i][j] = y[i+1][j-1] - y[i][j-1];

}

h = x[1] - x[0];

u = (x\_inter - x[n-1]) / h;

y\_inter = y[n-1][0];

for (int j = 1, i = n-2; j < n; j++, i--)

{

prod \*= (u + (j-1)) / j;

y\_inter += prod \* y[i][j];

}

printf("The functional value at x = %f is %.10f\n", x\_inter,

y\_inter);

return 0;

}

## Output

Interpolation using Newton's Backward Interpolation formula

Enter the number of points: 10

The x points must be equispaced

Enter the points:

1 x: 0.20

y: 1.2922071606

2 x: 0.35

y: 1.3397750591

3 x: 0.50

y: 1.3890939964

4 x: 0.65

y: 1.4402284308

5 x: 0.80

y: 1.4932451930

6 x: 0.95

y: 1.5482135742

7 x: 1.10

y: 1.6052054161

8 x: 1.25

y: 1.6642952050

9 x: 1.40

y: 1.7255601691

10 x: 1.55

y: 1.7890803797

Enter the value for which interpolation is required: 1.45

The functional value at x = 1.450000 is 1.7464789516

# Assignment 5

Compute the value of using Divided Difference interpolation formula from the following table.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

where is the last digit of your university roll number.

## Program

#include <stdio.h>

#include <stdlib.h>

#define N 10

int main()

{

double x[N] = {0}, y[N][N] = {0};

double x\_inter = 0, y\_inter = 0, prod = 1;

int n = 10;

printf("Interpolation using Divided Difference Interpolation"

" formula\n");

printf("Enter the number of points: ");

scanf("%d", &n);

if (n > N)

{

printf("Too many points\n");

exit(EXIT\_FAILURE);

}

printf("\n");

printf("Enter the points:\n");

for(int i = 0; i < n; i++)

{

printf("%d x: ", i+1);

scanf("%lf", &x[i]);

printf(" y: ");

scanf("%lf", &y[i][0]);

}

printf("\n");

printf("Enter the value for which interpolation is required: ");

scanf("%lf", &x\_inter);

// divided difference table

for (int j = 1; j < n; j++)

{

for (int i = 0; i < n-j; i++)

y[i][j] = (y[i+1][j-1] - y[i][j-1]) / (x[i+j] - x[i]);

}

y\_inter = y[0][0];

for (int j = 1; j < n; j++)

{

prod \*= x\_inter - x[j-1];

y\_inter += prod \* y[0][j];

}

printf("The functional value at x = %f is %f\n", x\_inter,

y\_inter);

return 0;

}

## Output

Interpolation using Divided Difference Interpolation formula

Enter the number of points: 8

Enter the points:

1 x: 0.12

y: 0.29751

2 x: 0.16

y: 0.31145

3 x: 0.22

y: 0.31848

4 x: 0.29

y: 0.32960

5 x: 0.34

y: 0.33774

6 x: 0.42

y: 0.34904

7 x: 0.49

y: 0.35729

8 x: 0.53

y: 0.36976

Enter the value for which interpolation is required: 0.425

The functional value at x = 0.425000 is 0.349649

# Assignment 6

Evaluate the following integral by Trapezoidal rule correct to 5D using 13 ordinates

where is the last digit of your university roll number.

## Program

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#define MAX\_ORD 2000

#define R 4

double f(double x)

{

double c = (2 + R)/10.0;

return sqrt(12.3 \* sin(c\*x) + 3.2 \* cos(c\*x));

}

double to\_radians(double angle)

{

double PI = 4 \* atan(1);

return angle \* PI / 180.0;

}

int main()

{

int num\_ordinates = 13;

double x[MAX\_ORD] = {0}, y[MAX\_LEN] = {0}, sum = 0, h = 0;

printf("Integration by Trapezoidal Rule\n");

printf("Enter the number of ordinates (including end-points):");

scanf("%d", &num\_ordinates);

if (num\_ordinates > MAX\_ORD)

{

printf("Too many points.\n");

exit(EXIT\_FAILURE);

}

h = (to\_radians(45) - to\_radians(0)) / (num\_ordinates - 1);

x[0] = 0;

y[0] = f(x[0]);

for (int i = 1; i < num\_ordinates; i++)

{

x[i] = x[0] + i\*h;

y[i] = f(x[i]);

}

for (int i = 0; i < num\_ordinates-1; i++)

sum += y[i] + y[i+1];

sum \*= h/2;

printf("The integration correct upto 5D is %.5f\n", sum);

return 0;

}

## Output

Integration by Trapezoidal Rule

Enter the number of ordinates (including end-points): 13

The integration correct upto 5D is 1.89533

# Assignment 6

Compute the value of the integral correct to 5D by Simpson’s one third rule taking 13 ordinates

where is the last digit of your university roll number.

## Program

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#define MAX\_ORD 200

#define R 4

double f(double x)

{

return pow(1.5 + (R+1)/20.0 \* pow(sin(x), 3), 3/2.0);

}

double to\_radians(double angle)

{

double PI = 4 \* atan(1);

return angle \* PI / 180.0;

}

int main()

{

int num\_ordinates = 13;

double x[MAX\_ORD] = {0}, y[MAX\_LEN] = {0}, sum = 0, h = 0;

printf("Integration by Simpson's Rule\n");

printf("Enter the number of ordinates (including end-points):");

scanf("%d", &num\_ordinates);

if (num\_ordinates > MAX\_ORD)

{

printf("Too many points.\n");

exit(EXIT\_FAILURE);

}

h = (to\_radians(60) - to\_radians(15)) / (num\_ordinates - 1);

x[0] = to\_radians(15);

y[0] = f(x[0]);

for (int i = 1; i < num\_ordinates; i++)

{

x[i] = x[0] + i\*h;

y[i] = f(x[i]);

}

for (int i = 0; i < num\_ordinates-2; i += 2)

sum += y[i] + 4\*y[i+1] + y[i+2];

sum \*= h/3;

printf("The integration correct upto 5D is %.5f\n", sum);

return 0;

}

## Output

Integration by Simpson's Rule

Enter the number of ordinates (including end-points): 13

The integration correct upto 5D is 1.53960

# Assignment 6

Compute the value of the following integral correct to 5D by Weddle’s rule using 13 ordinates

where is the last digit of your university roll number.

## Program

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#define MAX\_ORD 200

#define R 4

double f(double x)

{

double q = (6 + R)/60.0;

return (q + x \* pow(cos(q\*x), 2)) / (sqrt(x + sin(q\*x)));

}

double to\_radians(double angle)

{

double PI = 4 \* atan(1);

return angle \* PI / 180.0;

}

int main()

{

int num\_ordinates = 13;

double x[MAX\_ORD] = {0}, y[MAX\_LEN] = {0}, sum = 0, h = 0;

printf("Integration by Weddle's Rule\n");

printf("Enter the number of ordinates (including end-points):");

scanf("%d", &num\_ordinates);

if (num\_ordinates > MAX\_ORD)

{

printf("Too many points.\n");

exit(EXIT\_FAILURE);

}

h = (to\_radians(40) - to\_radians(10)) / (num\_ordinates - 1);

x[0] = to\_radians(10);

y[0] = f(x[0]);

for (int i = 1; i < num\_ordinates; i++)

{

x[i] = x[0] + i\*h;

y[i] = f(x[i]);

}

for (int i = 0; i < num\_ordinates-2; i += 6)

sum += y[i] + 5\*y[i+1] + y[i+2] + 6\*y[i+3] + y[i+4]

+ 5\*y[i+5] + y[i+6];

sum \*= 3\*h/10;

printf("The integration correct upto 5D is %.5f\n", sum);

return 0;

}

## Output

Integration by Weddle's Rule

Enter the number of ordinates (including end-points): 13

The integration correct upto 5D is 0.44192

# Assignment 6

Compute the following integration using six-point Gauss quadrature rule.

## Program

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#define MAX\_ORD 10

double f(double x)

{

return exp(0.03 \* sin(x)) / (pow(x,2) + 0.0009);

}

int main()

{

int n = 6, m = 0, i;

double u[MAX\_ORD] = {0}, w[MAX\_ORD] = {0};

double I = 0, a = 1.1, b = 3.3;

printf("Integration by Gauss Quadrature Rule\n");

printf("Enter limits of integration\n");

printf("Lower limit: ");

scanf("%lf", &a);

printf("Upper limit: ");

scanf("%lf", &b);

printf("Enter the number of points: ");

scanf("%d", &n);

printf("\n");

m = (n%2 == 0) ? n/2 : (n+1)/2;

if (m > MAX\_ORD)

{

printf("Too many points.\n");

exit(EXIT\_FAILURE);

}

for (i = 0; i < m; i++)

{

printf("Give the non-negative value of u[%d]: ", i);

scanf("%lf", &u[i]);

printf("Give the corresponding value of w[%d]: ", i);

scanf("%lf", &w[i]);

printf("\n");

}

if (n % 2 == 0)

{

I = 0;

i = 0;

}

else

{

I = w[0] \* f((u[0]\*(b - a) + (a + b)) / 2);

i = 1;

}

for ( ; i < m; i++)

I = I + w[i] \* (f((u[i]\*(b - a) + (a + b)) / 2)

+ f((-u[i]\*(b - a) + (a + b)) / 2));

I = (b - a) \* I / 2;

printf("The integration value is %f\n", I);

return 0;

}

## Output

Integration by Gauss Quadrature Rule

Enter limits of integration

Lower limit: 1.1

Upper limit: 3.3

Enter the number of points: 6

Give the non-negative value of u[0]: 0.2386191861

Give the corresponding value of w[0]: 0.4679139346

Give the non-negative value of u[1]: 0.6612093865

Give the corresponding value of w[1]: 0.3607615730

Give the non-negative value of u[2]: 0.9324695142

Give the corresponding value of w[2]: 0.1713244924

The integration value is 0.620976

# Assignment 7

Compute the dominant eigenvalue of the following matrix correct to 6D by power method:

where ; is the last digit of your university roll number.

## Program

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#include <stdbool.h>

#define N 10

/\*

Computes the maximum of the absolute values of the

`n` entries in `x`.

\*/

double abs\_max(double x[], int n) {

double l = fabs(x[0]);

for (int i = 0; i < n; i++)

if (l < fabs(x[i]))

l = fabs(x[i]);

return l;

}

/\*

Returns the dominant eigenvalue of the matrix `a(n\*n)`

Pass the initial approximation of eigenvector in `x`

Put `x\_i=1, i=1...n` for the approximation eigenvector

\*/

double dom\_eigenvalue(double a[][N], double x[], int n) {

double x1[N], l, error = 1E-10;

bool accurate;

do {

accurate = true;

// a\*x = x1

for (int i = 0; i < n; i++) {

x1[i] = 0;

for (int j = 0; j < n; j++)

x1[i] += a[i][j] \* x[j];

}

l = abs\_max(x1, n);

for (int j = 0; j < n; j++)

x1[j] = x1[j]/l;

for (int j = 0; j < n; j++) {

// keeps accurate to true only if all x[j]'s are

// close to x1[j]

if (fabs(x1[j] - x[j]) > error)

accurate = false;

x[j] = x1[j];

}

} while (!accurate);

return l;

}

int main()

{

int n = 4;

double a[N][N] = {0}, x[N] = {1};

printf("Calculating the dominant eigenvalue using "

"power method\n");

printf("Enter the order of the matrix: ");

scanf("%d", &n);

printf("\n");

if (n > N) {

printf("The order is too large\n");

exit(EXIT\_FAILURE);

}

printf("Enter the matrix row wise:\n");

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

scanf("%lf", &a[i][j]);

printf("\nEnter the intial approximation of the "

"dominant eigenvector: ");

for (int i = 0; i < n; i++)

scanf("%lf", &x[i]);

printf("\n");

printf("The dominant eigenvalue is %f\n",

dom\_eigenvalue(a, x, n));

return 0;

}

## Output

Calculating the dominant eigenvalue using power method

Enter the order of the matrix: 4

Enter the matrix row wise:

10.11 -1.15 1.55 -3.08

-1.15 16.56 -3.14 2.11

1.55 -3.14 10.12 -1.18

-3.08 2.11 -1.18 10.65

Enter the intial approximation of the dominant eigenvector: 1 1 1 1

The dominant eigenvalue is 19.296029

# Assignment 7

Evaluate the least (in magnitude) eigenvalue of the following matrix correct to 4D by Power method:

where ; is the last digit of your university roll number.

## Program

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#include <stdbool.h>

#define N 10

double abs\_max(double x[], int n) {

double l = fabs(x[0]);

for (int i = 0; i < n; i++)

if (l < fabs(x[i]))

l = fabs(x[i]);

return l;

}

/\*

Converts a n\*n matrix into an upper triangular matrix

The matrix a must be an augmented matrix

exit(EXIT\_FAILURE) if the diagonal elements are zero.

\*/

void to\_upper\_triangular(double a[][N+1], int n) {

for (int k = 0; k < n; k++) {

for (int i = k+1; i < n; i++) {

if (a[k][k] == 0) {

printf("The diagonal elements must be "

"numerically largest\n");

printf("a[%d][%d] is zero\n", k, k);

exit(EXIT\_FAILURE);

}

double m = a[i][k]/a[k][k];

for (int j = k; j < n+1; j++)

a[i][j] = a[i][j] - m \* a[k][j];

}

}

}

/\*

a must be augmented upper triangular matrix.

exit(EXIT\_FAILURE) if a[i][i] == 0

\*/

void back\_substitute(double a[][N+1], double x[], int n) {

for (int i = n-1; i >= 0; i--) {

double root = a[i][n];

for (int j = i+1; j < n; j++)

root = root - a[i][j]\*x[j];

if (a[i][i] == 0) {

printf("The diagonal elements must be "

"numerically largest\n");

printf("a[%d][%d] is zero\n", i, i);

exit(EXIT\_FAILURE);

}

x[i] = root / a[i][i];

}

}

/\*

Computes the inverse of a(n\*n) by Gaussian elimination

and stores it in b

exit(EXIT\_FAILURE) if the diagonal elements are zero.

\*/

void inverse\_matrix(double a[][N], double b[][N], int n) {

double x[N];

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++)

x[j] = 0;

x[i] = 1;

// forming augmented matrix a1 = [a|x]

double a1[N][N+1];

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

a1[i][j] = a[i][j];

for (int i = 0; i < n; i++)

a1[i][n] = x[i];

to\_upper\_triangular(a1, n);

back\_substitute(a1, x, n);

for (int k = 0; k < n; k++)

b[k][i] = x[k];

}

}

/\*

Returns the dominant eigenvalue of the matrix a(n\*n)

with initial approximation of eigenvector in x

Put i=1..n(x\_i=1) for the approximation eigenvector

\*/

double dom\_eigenvalue(double a[][N], double x[], int n) {

double x1[N], l, error = 1E-10;

bool accurate;

do {

accurate = true;

// a\*x = x1

for (int i = 0; i < n; i++) {

x1[i] = 0;

for (int j = 0; j < n; j++)

x1[i] += a[i][j] \* x[j];

}

l = abs\_max(x1, n);

for (int j = 0; j < n; j++)

x1[j] = x1[j]/l;

for (int j = 0; j < n; j++) {

// keeps accurate to true only if all x[j]'s

// are close to x1[j]

if (fabs(x1[j] - x[j]) > error)

accurate = false;

x[j] = x1[j];

}

} while (!accurate);

return l;

}

int main()

{

int n = 4;

double a[N][N] = {0}, b[N][N], x[N] = {1};

printf("Calculating the least eigenvalue using "

"power method\n");

printf("Enter the order of the matrix: ");

scanf("%d", &n);

printf("\n");

if (n > N) {

printf("The order is too large\n");

exit(EXIT\_FAILURE);

}

printf("Enter the matrix row wise:\n");

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

scanf("%lf", &a[i][j]);

printf("\nEnter the intial approximation of the "

"least eigenvector: ");

for (int i = 0; i < n; i++)

scanf("%lf", &x[i]);

printf("\n");

inverse\_matrix(a, b, n);

printf("The least eigenvalue is %.4f\n",

1/dom\_eigenvalue(b, x, n));

return 0;

}

## Output

Calculating the least eigenvalue using power method

Enter the order of the matrix: 4

Enter the matrix row wise:

1.27 1.99 1.04 -2.55

1.99 9.33 0.86 -2.00

1.04 0.86 3.85 1.82

-2.55 -2.00 1.82 -0.81

Enter the intial approximation of the least eigenvector: 1 1 1 1

The least eigenvalue is 1.9122

# Assignment 7

Evaluate the dominant and least (in magnitude) eigenvalues of the following matrix correct to 4D by power method:

where ; is the last digit of your university roll number.

## Program

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#include <stdbool.h>

#define N 10

double abs\_max(double x[], int n) {

double l = fabs(x[0]);

for (int i = 0; i < n; i++)

if (l < fabs(x[i]))

l = fabs(x[i]);

return l;

}

/\*

Converts a n\*n matrix into an upper triangular matrix

The matrix a must be an augmented matrix

exit(EXIT\_FAILURE) if the diagonal elements are zero.

\*/

void to\_upper\_triangular(double a[][N+1], int n) {

for (int k = 0; k < n; k++) {

for (int i = k+1; i < n; i++) {

if (a[k][k] == 0) {

printf("The diagonal elements must be "

"numerically largest\n");

printf("a[%d][%d] is zero\n", k, k);

exit(EXIT\_FAILURE);

}

double m = a[i][k]/a[k][k];

for (int j = k; j < n+1; j++)

a[i][j] = a[i][j] - m \* a[k][j];

}

}

}

/\*

a must be augmented upper triangular matrix.

exit(EXIT\_FAILURE) if a[i][i] == 0

\*/

void back\_substitute(double a[][N+1], double x[], int n) {

for (int i = n-1; i >= 0; i--) {

double root = a[i][n];

for (int j = i+1; j < n; j++)

root = root - a[i][j]\*x[j];

if (a[i][i] == 0) {

printf("The diagonal elements must be "

"numerically largest\n");

printf("a[%d][%d] is zero\n", i, i);

exit(EXIT\_FAILURE);

}

x[i] = root / a[i][i];

}

}

/\*

Computes the inverse of a(n\*n) by Gaussian elimination

and stores it in b

exit(EXIT\_FAILURE) if the diagonal elements are zero.

\*/

void inverse\_matrix(double a[][N], double b[][N], int n) {

double x[N];

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++)

x[j] = 0;

x[i] = 1;

// forming augmented matrix a1 = [a|x]

double a1[N][N+1];

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

a1[i][j] = a[i][j];

for (int i = 0; i < n; i++)

a1[i][n] = x[i];

to\_upper\_triangular(a1, n);

back\_substitute(a1, x, n);

for (int k = 0; k < n; k++)

b[k][i] = x[k];

}

}

/\*

Returns the dominant eigenvalue of the matrix a(n\*n)

with initial approximation of eigenvector in x

Put i=1..n(x\_i=1) for the approximation eigenvector

\*/

double dom\_eigenvalue(double a[][N], double x[], int n) {

double x1[N], l, error = 1E-10;

bool accurate;

do {

accurate = true;

// a\*x = x1

for (int i = 0; i < n; i++) {

x1[i] = 0;

for (int j = 0; j < n; j++)

x1[i] += a[i][j] \* x[j];

}

l = abs\_max(x1, n);

for (int j = 0; j < n; j++)

x1[j] = x1[j]/l;

for (int j = 0; j < n; j++) {

// keeps accurate to true only if all

// x[j]'s are close to x1[j]

if (fabs(x1[j] - x[j]) > error)

accurate = false;

x[j] = x1[j];

}

} while (!accurate);

return l;

}

int main()

{

int n = 4;

double a[N][N] = {0}, b[N][N], x[N] = {1};

printf("Calculating the dominant and least eigenvalue "

"using power method\n");

printf("Enter the order of the matrix: ");

scanf("%d", &n);

printf("\n");

if (n > N) {

printf("The order is too large\n");

exit(EXIT\_FAILURE);

}

printf("Enter the matrix row wise:\n");

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

scanf("%lf", &a[i][j]);

printf("\nEnter the intial approximation of the "

"dominant eigenvector: ");

for (int i = 0; i < n; i++)

scanf("%lf", &x[i]);

printf("\n");

printf("The dominant eigenvalue is %.4f\n",

dom\_eigenvalue(a, x, n));

printf("\nEnter the intial approximation of the "

"least eigenvector: ");

for (int i = 0; i < n; i++)

scanf("%lf", &x[i]);

printf("\n");

inverse\_matrix(a, b, n);

printf("The least eigenvalue is %.4f\n",

1/dom\_eigenvalue(b, x, n));

return 0;

}

## Output

Calculating the dominant and least eigenvalue using power method

Enter the order of the matrix: 4

Enter the matrix row wise:

9.14 -2.34 1.04 1.67

-2.34 7.38 1.86 1.56

1.04 1.86 10.65 0.98

1.67 1.56 0.98 6.99

Enter the intial approximation of the dominant eigenvector: 1 1 1 1

The dominant eigenvalue is 12.0639

Enter the intial approximation of the least eigenvector: 1 1 1 1

The least eigenvalue is 3.8492

# Assignment 8

Fit a curve of the form to the following data using Least Square method correct to 4D

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

where is the last digit of your university roll number.

## Program

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#define N 15

#define M 10

/\*

Converts a n\*n matrix into an upper triangular matrix

exit(EXIT\_FAILURE) if the diagonal elements are zero.

\*/

void to\_upper\_triangular(double a[][M+1+1], int n)

{

for (int k = 0; k < n; k++)

{

for (int i = k+1; i < n; i++)

{

if (a[k][k] == 0)

{

printf("The diagonal elements must be numerically"

"largest\n");

printf("a[%d][%d] is zero\n", k, k);

exit(EXIT\_FAILURE);

}

double m = a[i][k]/a[k][k];

for (int j = k; j < n+1; j++)

a[i][j] = a[i][j] - m \* a[k][j];

}

}

}

/\*

a must be augmented upper triangular matrix.

exit(EXIT\_FAILURE) if a[i][i] == 0

\*/

void back\_substitute(double a[][M+1+1], double x[], int n)

{

for (int i = n-1; i >= 0; i--)

{

double root = a[i][n];

for (int j = i+1; j < n; j++)

root = root - a[i][j]\*x[j];

if (a[i][i] == 0)

{

printf("The diagonal elements must be numerically"

"largest\n");

printf("a[%d][%d] is zero\n", i, i);

exit(EXIT\_FAILURE);

}

x[i] = root / a[i][i];

}

}

int main()

{

double x[N] = {0}, y[N] = {0}, a[M+1][M+1+1] = {0};

double coeff[M+1] = {0};

int n = 8, m = 1, num\_eqn;

printf("Curve Fitting using Least Square\n");

printf("Enter degree of fitting polynomial: ");

scanf("%d", &m);

if (m > M)

{

printf("Degree of polynomial too high.\n");

exit(EXIT\_FAILURE);

}

printf("Enter number of points: ");

scanf("%d", &n);

if (n > N)

{

printf("Too many points.\n");

exit(EXIT\_FAILURE);

}

printf("\n");

printf("Enter %d values of x: ", n);

for (int i = 0; i < n; i++)

scanf("%lf", &x[i]);

printf("Enter %d values of y: ", n);

for (int i = 0; i < n; i++)

scanf("%lf", &y[i]);

num\_eqn = m+1;

for (int k = 0; k < n; k++)

{

for (int i = 0; i < num\_eqn; i++)

{

for (int j = 0; j < num\_eqn; j++)

a[i][j] = a[i][j] + pow(x[k], i+j);

}

for (int i = 0; i < num\_eqn; i++)

a[i][num\_eqn] = a[i][num\_eqn] + pow(x[k], i)\*y[k];

}

// finding coefficients using Gaussian elimination

to\_upper\_triangular(a, num\_eqn);

back\_substitute(a, coeff, num\_eqn);

printf("The polynomial is: ");

for (int i = 0; i < m+1; i++)

{

if (i == 0)

printf("%.4f", coeff[i]);

else

printf(" %c %.4f\*x^%d", (coeff[i]<0 ? '-':'+'),

fabs(coeff[i]), i);

}

printf("\n");

return 0;

}

## Output

Curve Fitting using Least Square

Enter degree of fitting polynomial: 1

Enter number of points: 8

Enter 8 values of x: 2.5 3.5 4.5 5.5 6.5 7.5 8.5 9.5

Enter 8 values of y: 1.465 1.685 1.715 1.765 1.775 1.855 1.885 1.975

The polynomial is: 1.4079 + 0.0595\*x^1

# Assignment 8

Fit a curve of the form to the following data using Least Square method correct to 4D

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

where is the last digit of your university roll number.

## Program

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#define N 15

#define M 10

/\*

Converts a n\*n matrix into an upper triangular matrix

exit(EXIT\_FAILURE) if the diagonal elements are zero.

\*/

void to\_upper\_triangular(double a[][M+1+1], int n)

{

for (int k = 0; k < n; k++)

{

for (int i = k+1; i < n; i++)

{

if (a[k][k] == 0)

{

printf("The diagonal elements must be numerically"

"largest\n");

printf("a[%d][%d] is zero\n", k, k);

exit(EXIT\_FAILURE);

}

double m = a[i][k]/a[k][k];

for (int j = k; j < n+1; j++)

a[i][j] = a[i][j] - m \* a[k][j];

}

}

}

/\*

a must be augmented upper triangular matrix.

exit(EXIT\_FAILURE) if a[i][i] == 0

\*/

void back\_substitute(double a[][M+1+1], double x[], int n)

{

for (int i = n-1; i >= 0; i--)

{

double root = a[i][n];

for (int j = i+1; j < n; j++)

root = root - a[i][j]\*x[j];

if (a[i][i] == 0)

{

printf("The diagonal elements must be numerically"

"largest\n");

printf("a[%d][%d] is zero\n", i, i);

exit(EXIT\_FAILURE);

}

x[i] = root / a[i][i];

}

}

int main()

{

double x[N] = {0}, y[N] = {0}, a[M+1][M+1+1] = {0}, coeff[M+1] = {0};

int n = 8, m = 1, num\_eqn;

printf("Curve Fitting using Least Square\n");

printf("Enter degree of fitting polynomial: ");

scanf("%d", &m);

if (m > M)

{

printf("Degree of polynomial too high.\n");

exit(EXIT\_FAILURE);

}

printf("Enter number of points: ");

scanf("%d", &n);

if (n > N)

{

printf("Too many points.\n");

exit(EXIT\_FAILURE);

}

printf("\n");

printf("Enter %d values of x: ", n);

for (int i = 0; i < n; i++)

scanf("%lf", &x[i]);

printf("Enter %d values of y: ", n);

for (int i = 0; i < n; i++)

scanf("%lf", &y[i]);

num\_eqn = m+1;

for (int k = 0; k < n; k++)

{

for (int i = 0; i < num\_eqn; i++)

{

for (int j = 0; j < num\_eqn; j++)

a[i][j] = a[i][j] + pow(x[k], i+j);

}

for (int i = 0; i < num\_eqn; i++)

a[i][num\_eqn] = a[i][num\_eqn] + pow(x[k], i)\*y[k];

}

// finding coefficients using Gaussian elimination

to\_upper\_triangular(a, num\_eqn);

back\_substitute(a, coeff, num\_eqn);

printf("The polynomial is: ");

for (int i = 0; i < m+1; i++)

{

if (i == 0)

printf("%.4f", coeff[i]);

else

printf(" %c %.4f\*x^%d", (coeff[i]<0 ? '-':'+',

fabs(coeff[i]), i);

}

printf("\n");

return 0;

}

## Output

Curve Fitting using Least Square

Enter degree of fitting polynomial: 2

Enter number of points: 8

Enter 8 values of x: 1.2 2.2 3.2 4.2 5.2 6.2 7.2 8.2

Enter 8 values of y: 3.75 5.75 8.55 11.35 14.55 18.75 22.35 27.55

The polynomial is: 1.8066 + 1.3707\*x^1 + 0.2131\*x^2

# Assignment 8

Fit a curve of the form to the following data using Least Square method correct to 4D

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

where is the last digit of your university roll number.

## Program

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#define N 15

#define M 10

/\*

Converts a n\*n matrix into an upper triangular matrix

exit(EXIT\_FAILURE) if the diagonal elements are zero.

\*/

void to\_upper\_triangular(double a[][M+1+1], int n)

{

for (int k = 0; k < n; k++)

{

for (int i = k+1; i < n; i++)

{

if (a[k][k] == 0)

{

printf("The diagonal elements must be numerically"

"largest\n");

printf("a[%d][%d] is zero\n", k, k);

exit(EXIT\_FAILURE);

}

double m = a[i][k]/a[k][k];

for (int j = k; j < n+1; j++)

a[i][j] = a[i][j] - m \* a[k][j];

}

}

}

/\*

a must be augmented upper triangular matrix.

exit(EXIT\_FAILURE) if a[i][i] == 0

\*/

void back\_substitute(double a[][M+1+1], double x[], int n)

{

for (int i = n-1; i >= 0; i--)

{

double root = a[i][n];

for (int j = i+1; j < n; j++)

root = root - a[i][j]\*x[j];

if (a[i][i] == 0)

{

printf("The diagonal elements must be numerically"

"largest\n");

printf("a[%d][%d] is zero\n", i, i);

exit(EXIT\_FAILURE);

}

x[i] = root / a[i][i];

}

}

int main()

{

double x[N] = {0}, y[N] = {0}, a[M+1][M+1+1] = {0};

double coeff[M+1] = {0};

int n = 8, m = 1, num\_eqn;

printf("Curve Fitting using Least Square\n");

printf("Enter degree of fitting polynomial: ");

scanf("%d", &m);

if (m > M)

{

printf("Degree of polynomial too high.\n");

exit(EXIT\_FAILURE);

}

printf("Enter number of points: ");

scanf("%d", &n);

if (n > N)

{

printf("Too many points.\n");

exit(EXIT\_FAILURE);

}

printf("\n");

printf("Enter %d values of x: ", n);

for (int i = 0; i < n; i++)

scanf("%lf", &x[i]);

printf("Enter %d values of y: ", n);

for (int i = 0; i < n; i++)

scanf("%lf", &y[i]);

num\_eqn = m+1;

for (int k = 0; k < n; k++)

{

for (int i = 0; i < num\_eqn; i++)

{

for (int j = 0; j < num\_eqn; j++)

a[i][j] = a[i][j] + pow(x[k], i+j);

}

for (int i = 0; i < num\_eqn; i++)

a[i][num\_eqn] = a[i][num\_eqn] + pow(x[k], i)\*y[k];

}

// finding coefficients using Gaussian elimination

to\_upper\_triangular(a, num\_eqn);

back\_substitute(a, coeff, num\_eqn);

printf("The polynomial is: ");

for (int i = 0; i < m+1; i++)

{

if (i == 0)

printf("%.4f", coeff[i]);

else

printf(" %c %.4f\*x^%d", (coeff[i]<0 ? '-':'+'),

fabs(coeff[i]), i);

}

printf("\n");

return 0;

}

## Output

Curve Fitting using Least Square

Enter degree of fitting polynomial: 3

Enter number of points: 8

Enter 8 values of x: 3.1 4.1 5.1 6.1 7.1 8.1 9.1 10.1

Enter 8 values of y: 2.8 4.3 7.8 11.7 14.4 18.3 23.1 27.6

The polynomial is: -1.6589 + 0.5124\*x^1 + 0.2869\*x^2 - 0.0051\*x^3

# Assignment 8

Fit a curve of the form to the following data using Least Square method correct to 4D

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

where is the last digit of your university roll number.

## Program

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#define N 15

#define M 1

/\*

Converts a n\*n matrix into an upper triangular matrix

exit(EXIT\_FAILURE) if the diagonal elements are zero.

\*/

void to\_upper\_triangular(double a[][M+1+1], int n)

{

for (int k = 0; k < n; k++)

{

for (int i = k+1; i < n; i++)

{

if (a[k][k] == 0)

{

printf("The diagonal elements must be numerically"

"largest\n");

printf("a[%d][%d] is zero\n", k, k);

exit(EXIT\_FAILURE);

}

double m = a[i][k]/a[k][k];

for (int j = k; j < n+1; j++)

a[i][j] = a[i][j] - m \* a[k][j];

}

}

}

/\*

a must be augmented upper triangular matrix.

exit(EXIT\_FAILURE) if a[i][i] == 0

\*/

void back\_substitute(double a[][M+1+1], double x[], int n)

{

for (int i = n-1; i >= 0; i--)

{

double root = a[i][n];

for (int j = i+1; j < n; j++)

root = root - a[i][j]\*x[j];

if (a[i][i] == 0)

{

printf("The diagonal elements must be numerically"

"largest\n");

printf("a[%d][%d] is zero\n", i, i);

exit(EXIT\_FAILURE);

}

x[i] = root / a[i][i];

}

}

int main()

{

double x[N] = {0}, y[N] = {0}, a[M+1][M+1+1] = {0};

double coeff[M+1] = {0};

int n = 8, m = 1, num\_eqn;

printf("Curve Fitting using Least Square to the polynomial "

"a + b\*x^2\n");

printf("Enter number of points: ");

scanf("%d", &n);

if (n > N)

{

printf("Too many points.\n");

exit(EXIT\_FAILURE);

}

printf("\n");

printf("Enter %d values of x: ", n);

for (int i = 0; i < n; i++)

scanf("%lf", &x[i]);

printf("Enter %d values of y: ", n);

for (int i = 0; i < n; i++)

scanf("%lf", &y[i]);

num\_eqn = m+1;

for (int k = 0; k < n; k++)

{

for (int i = 0; i < num\_eqn; i++)

{

for (int j = 0; j < num\_eqn; j++)

a[i][j] = a[i][j] + pow(x[k], 2\*(i+j));

}

for (int i = 0; i < num\_eqn; i++)

a[i][num\_eqn] = a[i][num\_eqn] + pow(x[k], 2\*i)\*y[k];

}

// finding coefficients using Gaussian elimination

to\_upper\_triangular(a, num\_eqn);

back\_substitute(a, coeff, num\_eqn);

printf("The polynomial is: ");

for (int i = 0; i < m+1; i++)

{

if (i == 0)

printf("%.4f", coeff[i]);

else

printf(" %c %.4f\*x^%d", (coeff[i]<0 ? '-':'+'),

fabs(coeff[i]), 2\*i);

}

printf("\n");

return 0;

}

## Output

Curve Fitting using Least Square to the polynomial a + b\*x^2

Enter number of points: 8

Enter 8 values of x: 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0

Enter 8 values of y: 6.65 9.38 12.03 16.23 19.7 22.93 28.21 31.67

The polynomial is: 8.3795 + 0.3910\*x^2

# Assignment 9

Solve the following initial value problem by Euler’s method to find the values of for correct to 3D.

with , is the last digit of your university roll number.

## Program

#include <stdio.h>

#include <math.h>

#define R 4

double f(double x, double y) {

double num = pow(1 + pow(x\*y,3) + pow(x\*y,2), 2/3.0);

double denom = pow(1 + x\*x + y\*y, 1/3.0);

return num / denom;

}

int main()

{

double x0 = 0.0, y0 = 1.1 + R/100.0;

double xn = 0.5, h = 0.1;

printf("Solving differential equation by Euler's method\n");

printf("Enter values for x:\n");

printf("Initial value: ");

scanf("%lf", &x0);

printf("Final value: ");

scanf("%lf", &xn);

printf("Step length: ");

scanf("%lf", &h);

printf("\nEnter value of y(%.1f): ", x0);

scanf("%lf", &y0);

printf("\n");

printf("Solution of the differential equation correct to 3D: ");

printf("\n");

double x\_i = x0, y\_i = y0;

int n = round((xn - x0) / h);

printf("y(%0.1f) = %0.3f\n", x0, y0);

for (int i = 1; i <= n; ++i) {

x\_i = x0 + (i-1)\*h;

y\_i = y\_i + h\*f(x\_i, y\_i);

printf("y(%0.1f) = %0.3f\n", x\_i+h, y\_i);

}

return 0;

}

## Output

Solving differential equation by Euler's method

Enter values for x:

Initial value: 0.0

Final value: 0.5

Step length: 0.1

Enter value of y(0.0): 1.14

Solution of the differential equation correct to 3D:

y(0.0) = 1.140

y(0.1) = 1.216

y(0.2) = 1.290

y(0.3) = 1.366

y(0.4) = 1.446

y(0.5) = 1.536

# Assignment 9

Solve the following initial value problem by Modified Euler’s method to find the values of for correct to 5D.

with , is the last digit of your university roll number.

## Program

#include <stdio.h>

#include <math.h>

#define R 4

double f(double x, double y) {

double num = pow(1 + x\*y + pow(x\*y,2) + pow(x\*y,3), 3/2.0);

double denom = pow(1 + x\*y + pow(x\*y,2), 1/2.0);

return num / denom;

}

int main()

{

double x0 = 0.0, y0 = 1.1 + R/10.0;

double xn = 0.5, h = 0.1, error = 1E-7;

printf("Solving differential equation by Modified Euler's");

printf(" method\n");

printf("Enter values for x:\n");

printf("Initial value: ");

scanf("%lf", &x0);

printf("Final value: ");

scanf("%lf", &xn);

printf("Step length: ");

scanf("%lf", &h);

printf("\nEnter value of y(%.1f): ", x0);

scanf("%lf", &y0);

printf("\n");

printf("Solution of the differential equation correct to 5D:");

printf("\n");

double x\_i = x0, y\_i = y0;

int n = round((xn - x0) / h);

printf("y(%0.1f) = %0.5f\n", x0, y0);

for (int i = 1; i <= n; ++i) {

x\_i = x0 + (i-1)\*h;

double y\_prev, y = y\_i + h\*f(x\_i, y\_i);

do {

y\_prev = y;

y = y\_i + h/2 \* (f(x\_i, y\_i) + f(x\_i+h, y));

} while (fabs(y - y\_prev) >= error);

y\_i = y;

printf("y(%0.1f) = %0.5f\n", x\_i+h, y\_i);

}

return 0;

}

## Output

Solving differential equation by Modified Euler's method

Enter values for x:

Initial value: 0.0

Final value: 0.5

Step length: 0.1

Enter value of y(0.0): 1.5

Solution of the differential equation correct to 5D:

y(0.0) = 1.50000

y(0.1) = 1.60966

y(0.2) = 1.74609

y(0.3) = 1.93371

y(0.4) = 2.23640

y(0.5) = 2.93189

# Assignment 9

Solve the following initial value problem by 4th order Runge Kutta method and tabulate the values of for correct to 5D.

with , is the last of your university roll number.

## Program

#include <stdio.h>

#include <math.h>

#define R 4

double f(double x, double y) {

double num = 1 + log(pow(x,3) + pow(y,3));

double denom = 1.5 + 2.5\*pow(x,2) + 2.5\*pow(y,2);

return num / denom;

}

int main()

{

double x0 = 0.0, y0 = 1 + R/10.0;

double xn = 1, h = 0.1;

printf("Solving differential equation by "

"4th order Runge Kutta");

printf(" method\n");

printf("Enter values for x:\n");

printf("Initial value: ");

scanf("%lf", &x0);

printf("Final value: ");

scanf("%lf", &xn);

printf("Step length: ");

scanf("%lf", &h);

printf("\nEnter value of y(%.1f): ", x0);

scanf("%lf", &y0);

printf("\n");

printf("Solution of the differential equation correct to 5D:");

printf("\n");

printf("x\ty\n");

double x\_i = x0, y\_i = y0;

int n = round((xn - x0) / h);

printf("%0.1f \t%0.5f\n", x0, y0);

for (int i = 1; i <= n; ++i) {

x\_i = x0 + (i-1)\*h;

double k1 = h \* f(x\_i, y\_i);

double k2 = h \* f(x\_i + h/2, y\_i + k1/2);

double k3 = h \* f(x\_i + h/2, y\_i + k2/2);

double k4 = h \* f(x\_i + h, y\_i + k3);

y\_i = y\_i + 1/6.0 \* (k1 + 2\*k2 + 2\*k3 + k4);

printf("%0.1f \t%0.5f\n", x\_i+h, y\_i);

}

return 0;

}

## Output

Solving differential equation by 4th order Runge Kutta method

Enter values for x:

Initial value: 0

Final value: 1

Step length: 0.1

Enter value of y(0.0): 1.4

Solution of the differential equation correct to 5D:

x y

0.0 1.40000

0.1 1.43134

0.2 1.46240

0.3 1.49298

0.4 1.52294

0.5 1.55218

0.6 1.58065

0.7 1.60831

0.8 1.63515

0.9 1.66117

1.0 1.68639

# Assignment 9

Use Picard’s method and find three approximation values of for

## Program

#include <stdio.h>

#include <math.h>

#define R 4

double f1(double x) {

return x + pow(x,2)/2;

}

double f2(double x) {

return x + pow(x,2)/2 + pow(x,3)/3 + pow(x,4)/8;

}

double f3(double x) {

return x + pow(x,2)/2 + pow(x,3)/3 + pow(x,4)/8

+ pow(x,5)/15 + pow(x,6)/48;

}

int main()

{

double x0 = 0.1, y\_initial = 1.0;

double xn = 1.0, h = 0.1;

printf("Solving differential equation by Picard's method\n");

printf("Enter values for x:\n");

printf("Initial value: ");

scanf("%lf", &x0);

printf("Final value: ");

scanf("%lf", &xn);

printf("Step length: ");

scanf("%lf", &h);

printf("\nEnter initial value of y: ");

scanf("%lf", &y\_initial);

printf("\n");

printf("Solution of the differential equation: \n");

int n = round((xn - x0) / h);

for (int i = 0; i <= n; ++i)

{

double x\_i = x0 + i\*h;

printf("x = %f\n", x\_i);

printf("First approximation value of y(%.1f) = %f\n",

x\_i, y\_initial + f1(x\_i));

printf("Second approximation value of y(%.1f) = %f\n",

x\_i, y\_initial + f2(x\_i));

printf("Third approximation value of y(%.1f) = %f\n",

x\_i, y\_initial + f3(x\_i));

printf("\n");

}

return 0;

}

## Output

Solving differential equation by Picard's method

Enter values for x:

Initial value: 0.1

Final value: 1.0

Step length: 0.1

Enter initial value of y: 1

Solution of the differential equation:

x = 0.100000

First approximation value of y(0.1) = 1.105000

Second approximation value of y(0.1) = 1.105346

Third approximation value of y(0.1) = 1.105347

x = 0.200000

First approximation value of y(0.2) = 1.220000

Second approximation value of y(0.2) = 1.222867

Third approximation value of y(0.2) = 1.222889

x = 0.300000

First approximation value of y(0.3) = 1.345000

Second approximation value of y(0.3) = 1.355013

Third approximation value of y(0.3) = 1.355190

x = 0.400000

First approximation value of y(0.4) = 1.480000

Second approximation value of y(0.4) = 1.504533

Third approximation value of y(0.4) = 1.505301

x = 0.500000

First approximation value of y(0.5) = 1.625000

Second approximation value of y(0.5) = 1.674479

Third approximation value of y(0.5) = 1.676888

x = 0.600000

First approximation value of y(0.6) = 1.780000

Second approximation value of y(0.6) = 1.868200

Third approximation value of y(0.6) = 1.874356

x = 0.700000

First approximation value of y(0.7) = 1.945000

Second approximation value of y(0.7) = 2.089346

Third approximation value of y(0.7) = 2.103002

x = 0.800000

First approximation value of y(0.8) = 2.120000

Second approximation value of y(0.8) = 2.341867

Third approximation value of y(0.8) = 2.369173

x = 0.900000

First approximation value of y(0.9) = 2.305000

Second approximation value of y(0.9) = 2.630013

Third approximation value of y(0.9) = 2.680450

x = 1.000000

First approximation value of y(1.0) = 2.500000

Second approximation value of y(1.0) = 2.958333

Third approximation value of y(1.0) = 3.045833