Contents

Q1 : bisection 2
Q2: regula false 3
Q3 : NR 4
Q4 : gauss eleminetion 5
Q5 : lagranges 7
Q7: RK 2 / Heun's Method 8
Q8 : RK4 9
Q9 : trapezoidal: 10
Q10 : invarse of matrix 11
Q11 : Quadratic Curve Fitting for Regression 14
Q12:succissive approximation 16
Q13 : simson 1/3 17
Q14: linear regression 18

Q1 : bisection

```
#include <stdio.h>
#include <math.h>

// Function to evaluate the polynomial
double f(double x) {
    return pow(x, 3) - 4*x + 1;
}

// Function implementing Bisection Method
void bisection(double a, double b, double tol) {
    double c;
    if (f(a) * f(b) >= 0) {
        printf("Incorrect initial guesses.\n");
        return;
    }

    printf("Iter\t\t a\t\t b\t\t c\t\t f(c)\n");
    for (int iter = 1; (b - a) / 2 > tol; iter++) {
        c = (a + b) / 2;
        printf("%d\t\t %.6lf\t %.6lf\t %.6lf\t %.6lf\n", iter, a, b, c, f(c));

        if (f(c) == 0.0)
            break;
        else if (f(c) * f(a) < 0)
            b = c;
        else
            a = c;
    }

    printf("The root is: %.6lf\n", c);
}

int main() {
    double a = 0, b = 1, tol = 0.000001; // Initial guesses
    bisection(a, b, tol);
    return 0;
}
```

Q2: regula false

```
#include <stdio.h>
#include <math.h>

// Function to evaluate the polynomial
double f(double x) {
    return pow(x, 3) - 4*x - 36;
}

// Function implementing Regula Falsi Method
void regulaFalsi(double a, double b, double tol) {
    double c;
    if (f(a) * f(b) >= 0) {
        printf("Incorrect initial guesses.\n");
        return;
    }

    printf("Iter\t\t a\t\t b\t\t c\t\t f(c)\n");
    for (int iter = 1; fabs(b - a) > tol; iter++) {
        c = (a*f(b) - b*f(a)) / (f(b) - f(a));
        printf("%d\t\t %.6lf\t %.6lf\t %.6lf\t %.6lf\n", iter, a, b, c, f(c));

        if (fabs(f(c)) < tol)
            break;
        else if (f(c) * f(a) < 0)
            b = c;
        else
            a = c;
    }

    printf("The root is: %.6lf\n", c);
}

int main() {
    double a = 3, b = 4, tol = 0.000001; // Initial guesses
    regulaFalsi(a, b, tol);
    return 0;
}
```

Q3 : NR

```
#include <stdio.h>
#include <math.h>

// Function to evaluate the polynomial
double f(double x) {
    return pow(x, 3) - 4*x - 36;
}

// Function implementing Regula Falsi Method
void regulaFalsi(double a, double b, double tol) {
    double c;
    if (f(a) * f(b) >= 0) {
        printf("Incorrect initial guesses.\n");
        return;
    }

    printf("Iter\t\t a\t\t b\t\t c\t\t f(c)\n");
    for (int iter = 1; fabs(b - a) > tol; iter++) {
        c = (a*f(b) - b*f(a)) / (f(b) - f(a));
        printf("%d\t\t %.6lf\t %.6lf\t %.6lf\t %.6lf\n", iter, a, b, c, f(c));

        if (fabs(f(c)) < tol)
            break;
        else if (f(c) * f(a) < 0)
            b = c;
        else
            a = c;
    }

    printf("The root is: %.6lf\n", c);
}

int main() {
    double a = 3, b = 4, tol = 0.000001; // Initial guesses
    regulaFalsi(a, b, tol);
    return 0;
}
```

Q4 : gauss elemination

```
#include <stdio.h>

void gaussElimination(int n, double mat[][n+1]) {
    for (int i = 0; i < n; i++) {
        // Search for maximum in this column
        double maxEl = fabs(mat[i][i]);
        int maxRow = i;
        for (int k = i+1; k < n; k++) {
            if (fabs(mat[k][i]) > maxEl) {
                maxEl = fabs(mat[k][i]);
                maxRow = k;
            }
        }

        // Swap maximum row with current row
        for (int k = i; k < n+1; k++) {
            double tmp = mat[maxRow][k];
            mat[maxRow][k] = mat[i][k];
            mat[i][k] = tmp;
        }

        // Make all rows below this one 0 in current column
        for (int k = i+1; k < n; k++) {
            double c = -mat[k][i] / mat[i][i];
            for (int j = i; j < n+1; j++) {
                if (i == j) {
                    mat[k][j] = 0;
                } else {
                    mat[k][j] += c * mat[i][j];
                }
            }
        }
    }

    // Solve equation Ax=b for an upper triangular matrix A
    double x[n];
    for (int i = n-1; i >= 0; i--) {
        x[i] = mat[i][n] / mat[i][i];
        for (int k = i-1; k >= 0; k--) {
            mat[k][n] -= mat[k][i] * x[i];
        }
    }

    // Print solution
    printf("Solution:\n");
    for (int i = 0; i < n; i++) {
        printf("x%d = %.6lf\n", i+1, x[i]);
    }
}

int main() {
    int n = 4;
    double mat[4][5] = {
        {1, 1, 2, -1, 2},
```

```
        {1, -2, -3, 4, 0},
        {2, 1, -3, 4, 9},
        {3, 1, 2, 3, 7}
    };
    gaussElimination(n, mat);
    return 0;
}
```

Q5 : lagranges

```
#include <stdio.h>


double lagrangeInterpolation(double x[], double y[], int n, double xi) {
    double result = 0.0;
    for (int i = 0; i < n; i++) {
        double term = y[i];
        for (int j = 0; j < n; j++) {
            if (j != i) {
                term *= (xi - x[j]) / (x[i] - x[j]);
            }
        }
        result += term;
    }
    return result;
}


int main() {
    int n = 6;
    double x[] = {-3, -1, 0, 3, 5};
    double y[] = {-30, -22, -12, 330, 3458};
    double xi1 = -2.5;
    double xi2 = 5.5;


    printf("f(%.2lf) = %.6lf\n", xi1, lagrangeInterpolation(x, y, n, xi1));
    printf("f(%.2lf) = %.6lf\n", xi2, lagrangeInterpolation(x, y, n, xi2));


    return 0;
}
```

Q7: RK 2 / Heun's Method

```
#include <stdio.h>

// Function to evaluate the differential equation
double dydx(double x, double y) {
    return -x * y;
}

// Function implementing Runge-Kutta Second Order Method
void rungeKutta2(double x0, double y0, double x, double h) {
    int n = (int)((x - x0) / h);
    double k1, k2;

    printf("x\t\t y\n");
    for (int i = 0; i <= n; i++) {
        printf("%.2lf\t %.6lf\n", x0, y0);
        k1 = h * dydx(x0, y0);
        k2 = h * dydx(x0 + 0.5 * h, y0 + 0.5 * k1);
        y0 = y0 + k2;
        x0 = x0 + h;
    }
}

int main() {
    double x0 = 0, y0 = 1, x = 0.25, h = 0.05;
    rungeKutta2(x0, y0, x, h);
    return 0;
}
```

Q8 : RK4

```
#include <stdio.h>

// Function to evaluate the differential equation
double dydx(double x, double y) {
    return -x * y;
}

// Function implementing Runge-Kutta Fourth Order Method
void rungeKutta4(double x0, double y0, double x, double h) {
    int n = (int)((x - x0) / h);
    double k1, k2, k3, k4;

    printf("x\t\t y\n");
    for (int i = 0; i <= n; i++) {
        printf("%.2lf\t %.6lf\n", x0, y0);
        k1 = h * dydx(x0, y0);
        k2 = h * dydx(x0 + 0.5 * h, y0 + 0.5 * k1);
        k3 = h * dydx(x0 + 0.5 * h, y0 + 0.5 * k2);
        k4 = h * dydx(x0 + h, y0 + k3);
        y0 = y0 + (k1 + 2*k2 + 2*k3 + k4) / 6;
        x0 = x0 + h;
    }
}

int main() {
    double x0 = 0, y0 = 1, x = 0.25, h = 0.05;
    rungeKutta4(x0, y0, x, h);
    return 0;
}
```

Q9 : trapezoidal:

```
#include <stdio.h>

// Function to perform Trapezoidal integration
double trapezoidal(double x[], double y[], int n) {
    double integral = 0.0;
    for (int i = 0; i < n-1; i++) {
        integral += 0.5 * (x[i+1] - x[i]) * (y[i] + y[i+1]);
    }
    return integral;
}

int main() {
    int n = 10;
    double x[] = {-0.6, -0.5, -0.4, -0.3, -0.2, -0.1, 0.0, 0.1, 0.2, 0.3};
    double y[] = {4, 2, 3, 8, 4, -2, 2, 3, 5, 8};

    double integral = trapezoidal(x, y, n);
    printf("The integral is: %.6lf\n", integral);

    return 0;
}
```

Q10 : invarse of matrix

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define N 3  // Size of the matrix

void swap_row(float *a, float *b, int n) {
    float temp;
    for (int i = 0; i < n; i++) {
        temp = a[i];
        a[i] = b[i];
        b[i] = temp;
    }
}

void print_matrix(float A[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++)
            printf("%f\t", A[i][j]);
        printf("\n");
    }
}

void gauss_elimination(float A[N][N], float B[N][N]) {
    int i, j, k, max_row;
    float max_element, temp;

    for (i = 0; i < N; i++) {
        // Initialize B as identity matrix
        for (j = 0; j < N; j++) {
            B[i][j] = (i == j) ? 1.0 : 0.0;
        }
    }
```

```
for (i = 0; i < N; i++) {
    // Find pivot
    max_element = fabs(A[i][i]);
    max_row = i;
    for (k = i + 1; k < N; k++) {
        if (fabs(A[k][i]) > max_element) {
            max_element = fabs(A[k][i]);
            max_row = k;
        }
    }

    // Swap maximum row with current row
    if (max_row != i) {
        swap_row(A[i], A[max_row], N);
        swap_row(B[i], B[max_row], N);
    }

    // Make all rows below this one 0 in current column
    for (k = i + 1; k < N; k++) {
        temp = -A[k][i] / A[i][i];
        for (j = 0; j < N; j++) {
            A[k][j] += temp * A[i][j];
            B[k][j] += temp * B[i][j];
        }
    }
}

// Solve equation Ax=b using back substitution
for (i = N - 1; i >= 0; i--) {
    for (j = i + 1; j < N; j++) {
        temp = A[i][j];
        for (k = 0; k < N; k++) {
            B[i][k] -= temp * B[j][k];
        }
```

```
        }
        temp = 1 / A[i][i];
        for (k = 0; k < N; k++) {
            B[i][k] *= temp;
        }
    }
}

int main() {
    float A[N][N] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 10}
    };
    float B[N][N];

    printf("Original Matrix:\n");
    print_matrix(A);

    gauss_elimination(A, B);

    printf("\nInverse Matrix:\n");
    print_matrix(B);

    return 0;
}
```

Q11 : Quadratic Curve Fitting for Regression

```
#include <stdio.h>

void solveLinearEquations(double a[][3], double b[], int n) {
    for (int i = 0; i < n; i++) {
        for (int k = i + 1; k < n; k++) {
            double t = a[k][i] / a[i][i];
            for (int j = 0; j <= n; j++)
                a[k][j] = a[k][j] - t * a[i][j];
        }
    }

    double x[n];
    for (int i = n - 1; i >= 0; i--) {
        x[i] = a[i][n];
        for (int j = i + 1; j < n; j++) {
            x[i] = x[i] - a[i][j] * x[j];
        }
        x[i] = x[i] / a[i][i];
    }

    printf("Solution:\n");
    for (int i = 0; i < n; i++) {
        printf("x%d = %.6lf\n", i, x[i]);
    }
}

void quadraticFit(double x[], double y[], int n) {
    double X[2*n+1];
    for (int i = 0; i < 2*n+1; i++) {
        X[i] = 0;
        for (int j = 0; j < n; j++) {
            X[i] = X[i] + pow(x[j], i);
        }
    }

    double B[n+1][n+2], Y[n+1];
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= n; j++) {
            B[i][j] = X[i+j];
        }
    }

    for (int i = 0; i < n+1; i++) {
        Y[i] = 0;
        for (int j = 0; j < n; j++) {
            Y[i] = Y[i] + pow(x[j], i) * y[j];
        }
    }

    for (int i = 0; i <= n; i++) {
        B[i][n+1] = Y[i];
    }

    solveLinearEquations(B, Y, n+1);
```

```
}

int main() {
    int n = 9;
    double x[] = {-4, -3, -2, -1, 0, 1, 2, 3, 4, 5};
    double y[] = {21, 12, 4, 1, 2, 7, 15, 30, 45, 67};
    quadraticFit(x, y, 2);
    return 0;
}
```

Q12:succissive approximation

```
#include <stdio.h>
#include<math.h>
#include<stdlib.h>
double f(double x){
    return x*x*x +x*x - 1;
}
double g(double x) {
    return pow((x+1),-0.5);
}

double successiveApproximation(double initGuess, double error,int iteration) {
    double x0=initGuess;
    double x1=g(x0);
    for (int i = 0; i < iteration; i++)
    {
        x0=x1;
        x1=g(x0);
        if(fabs((x1-x0)/x1)<=error){
            printf("Converge for a root %f in %d interation",x1,i);
            exit(0);
        }
    }
    printf("Does't Converge");

    return x1;
}

int main() {
    double root;
    double initGuess = 0.5; // Initial Guess
    double error = 0.00001; // Tolerance level

    root = successiveApproximation(initGuess, error,10);

    printf("Root is approximately: %.6f\n", root);

    return 0;
}
```

Q13 : simson 1/3

```
#include <stdio.h>

// Define the function to be integrated
double f(double x) {
    // Example function: f(x) = x^2
    return x * x;
}

// Function to calculate the integral using Simpson's 1/3 rule
double simpson(double a, double b, int n) {
    // Check if n is even
    if (n % 2 != 0) {
        printf("Number of intervals n must be even.\n");
        return -1;
    }

    double h = (b - a) / n;
    double sum = f(a) + f(b);

    for (int i = 1; i < n; i++) {
        double x = a + i * h;
        if (i % 2 == 0) {
            sum += 2 * f(x);
        } else {
            sum += 4 * f(x);
        }
    }

    return (h / 3) * sum;
}

int main() {
    double a = 0;  // Lower limit of integration
    double b = 1;  // Upper limit of integration
    int n = 10;    // Number of intervals (must be even)

    double result = simpson(a, b, n);
    if (result != -1) {
        printf("The integral of the function from %.2lf to %.2lf is: %.6lf\n", a, b,
result);
    }

    return 0;
}
```

Q14: linear regression

```c
#include <stdio.h>

// Function to calculate the coefficients a and b for the best fit line y = ax + b
void linearRegression(double x[], double y[], int n, double *a, double *b) {
    double sumX = 0, sumY = 0, sumXY = 0, sumX2 = 0;

    for (int i = 0; i < n; i++) {
        sumX += x[i];
        sumY += y[i];
        sumXY += x[i] * y[i];
        sumX2 += x[i] * x[i];
    }

    // Calculating the coefficients
    *a = (n * sumXY - sumX * sumY) / (n * sumX2 - sumX * sumX);
    *b = (sumY - (*a) * sumX) / n;
}

int main() {
    // Example data points
    int n = 10; // Number of data points
    double x[] = {-4, -3, -2, -1, 0, 1, 2, 3, 4, 5};
    double y[] = {21, 12, 4, 1, 2, 7, 15, 30, 45, 67};

    double a, b;
    linearRegression(x, y, n, &a, &b);

    printf("The best fit line is y = %.6lfx + %.6lf\n", a, b);

    return 0;
}
```