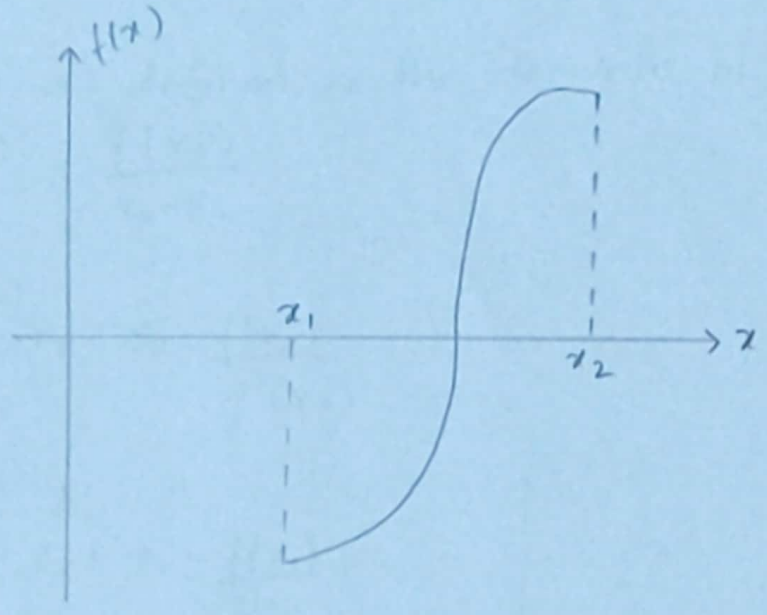
Bisection Method

If $f(x)$ is continuous in the interval (x_1, x_2) and $f(x_1)$ & $f(x_2)$ have different sign {i.e $f(x_1) * f(x_2) < 0$, then the equation $f(x) = 0$ has at least one root in the interval (x_1, x_2).

Now, the next point for next iteration,

$$x_3 = \frac{x_1 + x_2}{2}$$



Different cases arises:

ⓐ If $f(x_3) = 0$, then x_3 is the root of the equation

ⓑ If $f(x_1) * f(x_3) < 0$, root lies in the interval $[x_1, x_3]$.

ⓒ If $f(x_2) * f(x_3) < 0$, root lies in the interval $[x_3, x_2]$.

Newton Raphson's Method

Let $f(x)$ be a real and continuous function.

- x_0 be the initial approximation

Let's draw a tangent to the curve at point x_0.
The point at which tangent crosses the x-axis will be the improved estimate of the root.
Now, the slope of the tangent,

$$\tan\alpha = \frac{f(x_0)}{x_0 - x_1}$$

Slope is defined as the derivate of the function

$$f'(x_0) = \frac{f(x_0)}{x_0 - x_1}$$

Now,

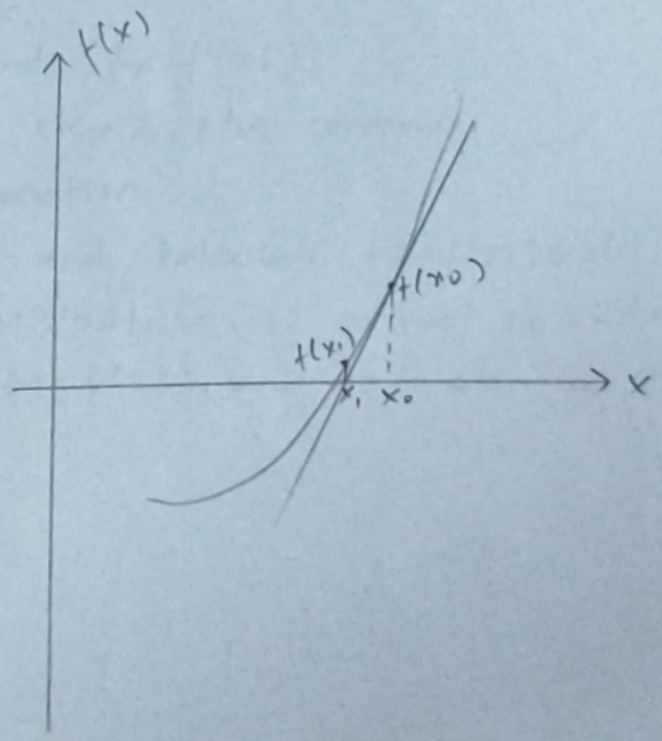
$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Similarly,

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$

So, in general,

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



Algorithm:

Bisection Method

1. Define a function $f(x)$
2. Input the tolerable error E.
3. Input initial guesses x1 and x2
4. Calculate $f1 = f(x1)$ and $f2 = f(x2)$
5. Check if $f1 * f2 < 0$, if so continue otherwise go to step 3.
6. Compute $x3 = (x1 + x2)/2$ and calculate $f(3) = f(x3)$
7. If $f1 * f3 < 0$, set $x2 = x3$ and $f2 = f3$.
   else set $x1 = x3$ and $f1 = f3$.
8. Check $|f3| <= E$, if so root is x3 and continue
   else go to step 5.
9. Stop.

Newton Raphson's Method

1. Define a function $f(x)$ and $f'(x)$.
2. Input the tolerable error E.
3. Input initial guesses x_1.
4. Calculate $f1 = f(x1)$ and $fd = f'(x1)$
5. Check if $fd = 0$, go to step 3, else continue
6. Compute next approximation
   $x_2 = x_1 - f1/fd$ and calculate $f2 = f(x2)$ & $fd = f'(x_2)$
7. Check ~~f(2) <E~~ $|(x2 - x1)/x2| <= e$, if so root is x2 & stop.
   else $x1 = x2$, $f1 = f2$, $fd = f(x2)$, and go to step 6.

Program:

Bisection Method

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

float calc (float a){
    return (a*a-16);
}

int main(){
    float x1,x2,x3,f1,f2,f3,e=0.001,root;
l1: printf ("Enter x1 and x2:");
    scanf ("%f%f",&x1,&x2);
    f1=calc(x1);
    f2=calc(x2);
l2: if ((f1*f2)<0)
    {
        x3=(x1+x2)/2;
        f3=calc(x3);
        if ((f1*f3)<0){
            x2=x3;
            f2=f3;
        }
        else{
            x1=x3;
            f1=f3;
        }
        if (fabs(f3)<=e){
            root=x3;
            printf ("%.2f",root);
            return 0;
        }else{
            goto l2;
        }
    }
    else{
        goto l1;
    }
}
```

Theory:

Discussion

Newton Raphson's Method

```
#include <stdio.h>
#include <math.h>
float function(float x){
    return x*x+4*x-9;
}
float derivative(float x){
    return 2*x+4;
}
int main(){
    float x1,x2,f1,fd,f2,e=0.1,root;
l1: printf("Enter the initial guess:");
    scanf("%f",&x1);
    f1=function(x1);
    fd=derivative(x1);
    if (fd==0){
        printf("Determinant is 0. So, root can't be converged.\n");
        goto l1;
    }
l2: x2=x1-f1/fd;
    f2=function(x2);
    fd=derivative(x2);
    if (fabs((x2-x1)/x2)<=e){
        root=x2;
        printf("%f",root);
        return 0;
    } else {
        x1=x2;
        f1=f2;
        fd=derivative(x2);
        goto l2;
    }
}
```

Theory:

Discussion

Bisection Method

In this program, I firstly created 'calc' function calculates the value of given function. Then, in 'main' function, the program asks user for initial values and calculates function values at these points, and then enters a loop where it checks if the signs of the function values are opposite. If true, it enters another loop where it calculates the midpoint, adjusts the interval based on the signs of the function values, and repeats the processes until the absolute value of the function at the midpoint is within an acceptable error ('e'). Once the condition is met, the ~~problem~~ program prints the root. The use of 'goto' statements, is used to redirect the flow of code to the required place.

Newton Raphson's Method

Firstly, the user is prompted to provide an initial guess and the program calculates it's functional value and the derivative value. If the derivative is found to be zero at the initial guess, the program informs the user that convergence is not possible and prompts for a new initial guess. Otherwise, it enters a loop it ~~#~~ iteratively applies the Newton Raphson's formula to update the approximation. The loop continues until the relative error between consecutive approximations is within an acceptable tolerance. Once this condition is met, we get the root.

CamScanner

Theory:

Conclusion

Hence, the program development of Lagrange and Newton Interpolation method is concluded.

Theory:

Secant Method

The Newton Raphson method formula is:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad \ldots \text{(i)}$$

The derivative of function $f(x)$ can be approximated as:

$$f'(x_n) = \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}} \quad \ldots \text{(ii)}$$

From (i) & (ii)

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

This is the equation of secant method.

Fixed Point Method

In fixed point iteration method, we rearrange the function $f(x) = 0$ such that x is on the left hand side. This can be expressed as below:

$$f(x) = 0$$

is written as

$$x = g(x)$$

If x_0 is the initial guess to the root, next approximated value of root can be calculated as:

$$x_1 = g(x_0)$$

Similarly,

$$x_2 = g(x_1)$$

After generalizing,

$$x_{n+1} = g(x_n)$$

CamScanner

Algorithm

Secant Method

1. Define a function $f(x)$
2. Input the tolerable error(e)
3. Input initial guesses as x1 & x2.
4. Calculate f1: f(x1) & f2: f(x2)
5. Compute x3: (x1* f2 - x2* f1)/ (f2-f1)
6. Calculate f3: f(x3).
7. Check |(x3-x2)/x3| <= E, if so root is x3 and stop.
8. Else set x1: x2, x2: x3 & f1: f2, f2: f3.
9. Go to step 5.
10. End

Fixed Point Method

1. Define function $f(x)$
2. Define function $g(x)$ from $f(x)=0$ such that $x=g(x)$ and $|g'(x)<1|$.
3. Input tolerable error e.
4. Input initial guess x1.
5. Calculate x2 = g(x1).
6. If |f(x2)| > E then
   x1 = x2
   go to step 5.
7. Else root is x2
8. End

Program

Secant Method

```
#include <stdio.h>
#include <math.h>
float function (float x){
    return 2*x*x+4*x-10;
}
int main(){
    float x1,x2,x3,f1,f2,f3,root,e=0.05;
    printf("Enter two initial guesses:");
    scanf("%f%f",&x1,&x2);
    f1=function(x1);
    f2=function(x2);
l1: x3=(x1*f2-x2*f1)/(f2-f1);
    f3=function(x3);
    if (fabs(x3-x2)/x3<=e){
        root=x3;
        printf("The required root is %f",root);
        return 0;
    }else{
        x1=x2;
        x2=x3;
        f1=f2;
        f2=f3;
        goto l1;
    }
}
```

Program

Fixed Point Method

```
#include <stdio.h>
#include <math.h>
float fx (float x){
    return x*x+x-2;
}
float gx (float x){
    return 2/(x+1);
}
int main(){
    float x1, x2, root, e=0.0001;
    printf ("Enter initial guess:");
    scanf ("%f", &x1);
    if (fabs (-2/((x1+1)*(x1+1)))>1){
        printf ("Since the root can't be converged.
        Enter a different initial guess.");
        return 0;
    }
l1: x2 = gx(x1);
    if (fabs (fx(x2))>e){
        x1 = x2;
        goto l1;
    }else {
        root = x2;
        printf (" The required root is %f", root);
        return 0;
    }
}
```

Discussion

Secant Method

The program begins by prompting the user to enter two initial guesses for the root. It then iteratively refines these guesses by constructing secant lines that intersect the function's graph. The point of intersection in each iteration becomes the new guess, and the process continues until a sufficiently accurate approximation is found. The code uses a tolerance level to determine the level of accuracy required for the approximation.

Fixed Point Method

This program is beginned by defining two function f(x) itself and a function g(x) related to f(x) that will be used in the fixed-point ~~a~~ iteration. The main part of the code first prompts the user to provide an initial guess for the root. It then checks a condition related to the derivative of g(x) to ensure the convergence of the method. If the condition is not met, it asks for a different initial guess. Else, it repeatedly applies g(x) to the current approximation and checking if the value of f(x) at the new approximation is sufficiently close to zero. If so, it reports the approximated root, else it continues the iteration until convergence is reached or a maximum number of iterations is exceeded.

CamScanner

Conclusion

Hence, the program for secant method and fixed point method is ~~calculated~~. were developed in c.

CamScanner

Lagrange Interpolation Method

Let us consider a second order polynomial of the form

$$P_2(x) = b_1(x-x_0)(x-x_1) + b_2(x-x_1)(x-x_2) + b_3(x-x_2)(x-x_0) \quad \text{— (A)}$$

Let (x_0, f_0), (x_1, f_1) and (x_2, f_2) be three interpolation points.

Now, find the polynomial in different values of x

$$P_2(x_0) = b_2(x_0-x_1)(x_0-x_2)$$

$$f_0 = b_2(x_0-x_1)(x_0-x_2)$$

$$\therefore b_2 = \frac{f_0}{(x_0-x_1)(x_0-x_2)}$$

$$P_2(x_1) = f_1 = b_3(x_1-x_2)(x_1-x_0)$$

$$b_3 = \frac{f_1}{(x_1-x_2)(x_1-x_0)}$$

Similarly, for $P_2(x_2) = f_2 = b_1(x_2-x_0)(x_2-x_1)$

$$\therefore b_1 = \frac{f_2}{(x_2-x_0)(x_2-x_1)}$$

<u>Now,</u>

Putting the value of b_1, b_2 & b_3 in eqⁿ (A)

$$P_2(x) = f_0\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} + f_1\frac{(x-x_2)(x-x_0)}{(x_1-x_2)(x_1-x_0)} + f_2\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} \quad \text{— (B)}$$

Eqⁿ (B) can be represented as

$$P_2(x) = f_0 l_0(x) + f_1 l_1(x) + f_2 l_2(x)$$

where $l_0(x), l_1(x), l_2(x)$ are lagrange basis polynomial.

$$\therefore P_2(x) = \sum_{i=0}^{2} f_i l_i(x) \text{ where } l_i(x) \prod_{j=0, j\neq i}^{2} \frac{x-x_j}{x_i-x_j}$$

Generalizing

$$P_n(x) = \sum_{i=0}^{n} f_i l_i(x) \quad \text{— (C)}$$

This is Lagrange Interpolation Polynomial

Newton Interpolation Method

The Newton's form of polynomial is expressed as

$$P_n(x) = a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1) + \dots a_n(x-x_0)(x-x_1)\dots(x-x_{n-1})$$

Let $(x_1,f_0),(x_1,f_1),(x_2,f_2)\dots(x_{n-1},f_{n-1})$ be the interpolation points

i.e

For $x = x_0$ $P_n(x_k) = f_k$ where $k = 0, 1, \dots n-1$

For $x = x_1$, $a_0 = f_0$

$$P_n(x_1) = a_0 + a_1(x-x_0)$$

i.e $f_1 = a_0 + a_1(x_1 - x_0)$

$$a_1 = \frac{f_1 - f_0}{x_1 - x_0}$$

At $x = x_2$

$$P_n(x_2) = a_0 + a_1(x_2 - x_0) + a_2(x_2 - x_0)(x_2 - x_1)$$

$$f_2 = a_0 + a_1(x_2 - x_0) + a_2(x_2 - x_0)(x_2 - x_1)$$

$$a_2 = \frac{\frac{(f_2 - f_1)}{(x_2 - x_1)} - \frac{(f_1 - f_0)}{(x_1 - x_0)}}{x_2 - x_0}$$

Let us define a representation

$$f[x_k] = f_k$$

$$f[x_k, x_{k+1}] = \frac{f[x_{k+1}] - f[x_k]}{x_{k+1} - x_k}$$

$$\vdots$$

$$f[x_k, x_{k+1} \dots x_i, x_{i+1}] = \frac{f[x_{k+1} \dots x_{i+1}] - f[x_k \dots x_i]}{x_{i+1} - x_k}$$

In terms of a_i

$$a_0 = f_0$$

$$a_1 = \frac{f_1 - f_0}{x_1 - x_0} = f[x_0 - x_1]$$

Similarly, $a_n = f[x_0, x_1 \dots x_n]$

Thus, the interpolating polynomial

$$P_n(x) = f_0 + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x_0 - x_0)(x - x_1) + \dots$$
$$f[x_0, x_1 \dots x_n](x_1 - x_0)(x_2 - x_1)\dots(x - x_{n+1})$$

which can be represented as

Algorithm:

Lagrange Interpolation Method

1. Enter the number of data available(n)
2. Input data X_i and Y_i for i=1 to n
3. Input the value of x (xint) for which y (yint) is to be calculated.
4. Initialize yint = 0
5. For i=1 to n
   Set l=1
   For j=1 to n
   If i≠j to n
   Calculate l=l * (xint-xj)/(xi-xj)
   End if
   Next j
   Calculate yint = yint + l * Yi
   Next i
6. Output yint as calculated interpolated value.
7. End.

Newton Interpolation Method

1. Enter the number of data available(n)
2. Input data X_i and Y_i
3. Input the value of x (xint) for which y (yint) is to be calculated.
4. define array d[i][j] for the difference table
5. d[i][1] = y[i]
6. for j=2 to less than equal to ~~for (n-j+1)~~ n;
   for i=1 to less than equal to (n-j+1)
   d[i][j] = (d[i+1][j-1] - d[i][j-1])/(x[i+j-1] - x[i]);
   Next j
   Next i
7. Assign a[j] = d[1][j]
8. Initialize p=1, yint = y1
   for j=1 to less than equal to i-1
   p = p * (xint - x[j]);
   next j
   yint = yint + p * a[j]
   next i
10. Output yint as calculated interpolated value
11. End.

Newton Interpolation Method

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

Program

Lagrange Interpolation Method

```
#include <stdio.h>
#include <math.h>
int main(){
    int i,j;
    float n,l,xint,yint,x[10],y[10];
    printf("Enter the number of data available:");
    scan("%f",&n);
    printf("Enter the values of x:");
    for(i=0;i<n;i++){
        scanf("%f",&x[i]);
    }
    printf("Enter the values of y:");
    for(i=0;i<n;i++){
        scanf("%f",&y[i]);
    }
    printf("Enter the value to be calculated:");
    scanf("%f",&xint);
    for (i=0;i<n;i++){
        l=1
        for (j=0;j<n;j++){
            if (i!=j){
                l=l*(xint-x[j])/x[i]-x[j]);
            }
        }
        yint=yint+l*y[i];
    }
    printf("The required value is %.2f",yint);
    return 0;
}
```

Newton Interpolation Method

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

int main(){
    int i,j;
    float n,p, xint,yint,x[10],y[10],a[10],d[10][10];
    printf("Enter the number of data available:");
    scanf("%f",&n);
    printf("Enter the values of x:");
    for (i=0;i<n;i++){
        scanf("%f",&x[i]);
    }
    printf("Enter the values of y:");
    for (i=0;i<n;i++){
        scanf("%f",&y[i]);
    }
    printf("Enter the value to be calculated:");
    scanf("%f",&xint);
    for (i=0;i<n;i++){
        d[i][0]=y[i];
    }
    for (j=1;j<n;j++){
        for (i=0;i<n-j;i++){
            d[i][j]=(d[i+1][j-1]-d[i][j-1])/(x[i+j]-x[i]);
    }   }
    yint=y[0];
    for (i=1;i<n;i++){
        p=1;
        for (j=0;j<i;j++){
            p*=xint-x[j];
        }
        yint+=p*d[0][i];
    }
    printf("The required value is %.2f\n",yint);
    return 0;
}
```

Discussion

Lagrange Interpolation Method

- The program performs lagrange interpolation to estimate the value of a function at a given point based on set of data points provided by a user. The user is prompted to enter number of data points ('n') and the 'x' & 'y' values for each data point. Then the user inputs the value at which they want to estimate the function ('value').

The program iterates through each data point and calculate the lagrange polynomial using the given formula:

$$l_i(x) = \prod_{j=0, j \neq i}^{n-1} \frac{x - x_j}{x_i - x_j}$$

where x_i and y_i are the i-data points x & y values

It then calculates the interpolated value at the specified point ~~su~~ by summing up the products for each lagrange polynomial and the corresponding y value.

Finally, it prints the result, which is the estimated value of the function at the given points.

Newton Interpolation Method

- The program performs Newton's divided difference ~~inter~~ interpolation to estimate the value of a function at a given set of point and datas provided by the user. The user is prompted to enter the no. of data points(n) and the x & y values for each data point. The user inputs the value at which they want to interpolate (xint).

The program calculates the divided difference table(d) using formula $d[i][0] = y[i]$, $d[i][j] = \frac{d[i+1][j-1] - d[i][j]}{x[i+j] - x[i]}$

It computes the coefficient [a] of the interpolation polynomial from the first row of the table. It calculates (yint) at the specified point (xint) using the Newton divided difference interpolation formula.

Finally, the program prints the interpolated value.

Conclusion

Hence, the program development of Lagrange and Newton Interpolation method was completed.

CamScanner

Discussion

Forward Difference Quotient

Firstly, the program defines the function. Then, it asks the ...er to input the value of x & the step size h. Using the ... derivative

Linear Curve ($y = a + bx$)

It is the process of finding the best-fitting curve or line of best fit for a set of data points by reducing the sum of the squares of the offsets of the points from the curve

Formula: $b = \frac{n\Sigma xy - \Sigma x . \Sigma y}{n\Sigma x^2 - (\Sigma x)^2}$ $\quad a = \frac{\Sigma y}{n} - b\frac{\Sigma x}{n}$

$y = a + bx$

Transcendental Curve ($y = ax^b$)

It is a mathematical method used to approximate the relationship betn data points by fitting them to a curve that represents a transcendental function.

Formula ?

Algorithm:

Linear Curve ($y=a+bx$)

1. Enter the number of data available(n)
2. Input data x[i] and y[i] for i=1 to n
3. Initialize sumx=0, sumx2=0, sumy=0, sumxy=0
4. For i=1 to n:
   - sumx = sumx + x[i]
   - sumx2 = sum2 + x[i]* x[i]
   - sumy = sumy + y[i]
   - sumxy + x[i]* y[i]
   - Next i
5. Calculate deno=n* sumx2-sumx * sumx
6. If den=0 end
7. else
8. b=(n*sumxy-sumx*sumy)/deno
9. a=(sumy-b*sumx)/n
10. display the equation of line y=a+bx
11. end.

Transcendental Curve ($y=ax^b$)

1. Enter the number of data available(n).
2. Input data x[i] and y[i] for i=1 to n.
3. x[i]=ln(x[i])
   y[i]=ln(y[i])
4. Initialize sumx=0, sumx2=0, sumy=0, sumxy=0
5. For i=1 to n
   - sumx=sumx+x[i]
   - sumx2=sumx2+x[i]*x[i]
   - sumy=sumy+y[i]
   - sumxy=sumxy+x[i]*y[i]
   - Next i
6. Calculate
   - b=(n*sumxy-sumx*sumy)/(n*sumx2-sumx*sumx)
   - a=(sumy-b*sumx)/n
   - a=exp(a)
7. Display the equation of line y=ax^b
8. end.

Discussion

Forward Difference Quotient

Firstly, the program defines the function. Then, it asks the user to input the value of x & the step size h. Using the

Program

Linear Curve

```
#include <stdio.h>
int main(){
    int i,j,n;
    float x[10],y[10],sumx=0,sumx2=0,sumy=0,sumxy=0,a,b;
    float deno,xint;
    printf("Enter the number of data:");
    scanf("%d",&n);
    printf("Enter the value of x:");
    for (i=0;i<n;i++)
        scanf("%f",&x[i]);
    printf("Enter value of y:");
    for (i=0;i<n;i++)
        scanf("%f",&y[i]);
    for (i=0;i<n;i++){
        sumx=sumx+x[i];
        sumx2=sumx2+x[i]*x[i];
        sumy=sumy+y[i];
        sumxy=sumxy+x[i]*y[i];
    }
    deno=n*sumx2-sumx*sumx;
    if (deno==0){
        printf("Solution doesn't exist.");
        return 0;
    }else{
        b=(n*sumxy-sumx*sumy)/deno;
    }
    a=(sumy-b*sumx)/n;
    printf("The required eqn of the line is y=%.2f+%.2fx",a,b);
    return 0;
}
```

Discussion

Forward Difference Quotient

Firstly, the program defines the function. Then, it asks the user to input the [illegible]

Transcendental curve

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

int main(){
    int i,j,n;
    float x[10],y[10], sumx=0, sumx2=0, sumy=0, sumxy=0, a,b;
    printf("Enter the number of data:");
    scanf("%d",&n);
    printf("Enter value of x:");
    for(i=0; i<n; i++)
        scanf("%f",&x[i]);
    printf("Enter value of y:");
    for(i=0;i<n;i++)
        scanf("%f",&y[i]);
    for(i=0;i<n;i++){
        x[i]=log(x[i]);
        y[i]=log(y[i]);
    }
    for(i=0;i<n;i++){
        sumx = sumx + x[i];
        sumx2 = sumx2 + x[i]*x[i];
        sumy= sumy+y[i];
        sumxy= sumxy + x[i]*y[i];
    }
    b= (n*sumxy-sumx*sumy)/(n*sumx2-sumx*sumx);
    a= (sumy-b*sumx)/n;
    a=exp(a);
    printf("The required eqn of the line is y= %.2f x^%.2f",
            a,b);
    return 0;
}
```

Discussion

Linear Curve

This program efficiently determines the equation of a straight line that best fits a given data set. It starts by collecting the x and y values of the data points from the user. Then, it calculates the necessary sums and uses them to determine the slope and intercept of the line. The program ensures a valid solution by checking for a non-zero denominator. Finally, it represents the equation of the line in the standard form y=a+bx, where 'a' is the intercept & 'b' is the slope.

Transcendental Curve

This program determines the equation of a transcendental curve, which follows the form y=ax^b, based on a given dataset. It begins by collecting the x & y values from the user. Then it transforms the x & y values by taking their natural logarithms, preparing them for linear regression. After computing the necessary sums, it determines curve's parameters 'a' & 'b'. By exponentiating 'a' back to its original scale, the program presents the equation of the curve in the desired format.

Discussion

Forward Difference Quotient

Firstly, the program defines the function. Then, it asks the

Forward Difference Quotient

It is a method used for approximating the derivative of a function at a particular point by considering the values of the function at that point & a small increment forward.

$$f'(x) = \frac{f(x+h) - f(x)}{h}$$

Central Difference Quotient

It is a method used to approximate the derivative of a function at a particular point. It is a finite difference approximation that calculates the derivative by taking the average of the slopes on either side of the point

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h}$$

CamScanner

Discussion

Forward Difference Quotient

Firstly, the program defines the function. Then, it asks the [illegible] value of x & the step size h. Using the

Algorithm

Forward Difference Quotient

1. Start
2. Define the function
3. Enter value of 'x' & 'h'.
4. Calculate

   fx = (function(x+h) - function(x)) / h;

5. Display the result or calculated derivative
6. End.

~~Backward~~ Central Difference Quotient

1. Start
2. Define the function
3. Enter the value of 'x' & 'h'.
4. Calculate

   fx = (function(x+h) - function(x-h)) / (2*h);

5. Display the result or calculated derivative
6. End.

CamScanner

Program

Forward Difference ~~Me~~ Quotient

```
#include <stdio.h>

float function(float x){
    return x*x;
}
int main(){
    float x,h,fx;
    printf("Enter the value of x:");
    scanf("%f",&x);
    printf("Enter the value of h:");
    scanf("%f",&h);
    fx=(function(x+h)-function(x))/h;
    printf("The required value is %.2f",fx);
    return 0;
}
```

Central ~~Backward~~ Difference Quotient

```
#include <stdio.h>
float function(float x){
    return x*x;
}
int main(){
    float x,h,fx;
    printf("Enter the value of x:");
    scanf("%f",&x);
    printf("Enter the value of h:");
    scanf("%f",&h);
    fx=(function(x+h)-function(x-h))/(2*h);
    printf("The required value is %.2f",fx);
    return 0;
}
```

CamScanner

Discussion

Forward Difference Quotient

Firstly, the program defines the function. Then, it asks the user to input the value of x & the step size h. Using the forward difference formula, it calculates the derivative value & displays it. Using the formula $\frac{f(x+h)-f(x)}{h}$.

Central Difference Quotient

This program defines a functions. Then it prompts user for input value of x & step size h. Using central difference formula, it calculates the derivative value & displays it using the formula $\frac{f(x+h)-f(x-h)}{2h}$.

CamScanner

Composite Trapezoidal Rule

The integration interval is divided to k segments of equal width to improve accuracy of trapezoidal rule. The equations are called the multiple-segment or composite integration formulae.

$$A \approx \frac{h}{2}\left[y_0 + 2(y_1 + y_2 + \cdots + y_{n-1}) + y_n\right]$$

Simpson's 1/3 Rule

It is an expansion of trapezoid rule where the integration is approximated using 2nd order polynomial.

$$\int_{x_0}^{x_n} f(x)\,dx = \frac{h}{3}\left[f_0 + f_n + 4(f_1 + f_5 + \ldots) + 2(f_2 + f_4 + f_6 + \ldots)\right]$$

Simpson's 3/8 Rule

It is also expansion of trapezoid rule where the integration is approximated using 3rd order polynomial.

$$\int_{x_0}^{x_n} f(x)\,dx = \frac{3h}{8}\left[f_0 + f_n + 3(f_1 + f_2 + f_4 + f_5 + \ldots) + 2(f_3 + f_6 + f_9 + \ldots)\right]$$

CamScanner

Discussion

Trapezoidal Rule

~~Algorithm~~ Program

Algorithm

Composite Trapezoidal Rule

1. Start
2. Define the function
3. Enter value of b, a, n.
4. Calculate $h = \frac{b-a}{n}$
5. Compute ans = f(b) + f(a)
6. For i = 1 to n-1, do
   - Calculate k = a + i * h
   - Add 2 * f(k) to ans
7. Multiply ans by h & divide by 2.
8. Display the output
9. End.

Simpson's 1/3 Rule

1. Start
2. Define the function f(x)
3. Enter value of b, a, n.
4. Calculate h = (b-a)/n
5. Compute ans = f(b) + f(a).
6. For i = 1 to n-1, do:
   - Calculate k = a + i * h.
   - If i is even, add 2 * f(k) to ans; otherwise, add 4 * f(k)
7. Multiply ans by h/3 to get the result
8. Display the result
9. End.

Simpson's 3/8 Rule

1. Start
2. Define the function f(x)
3. Enter value of b, a, n.
4. Calculate h = (b-a)/n
5. Compute ans = f(b) + f(a)
6. For i = 1 to n-1 do:
   - Calculate k = a + i * h
   - If i is a multiple of 3, add 2 * f(k) to ans, else add 3 * f(k)
7. Multiply ans by 3h/8 to get the final result
8. Display the output
9. End.

CamScanner

Discussion

Composite Trapezoidal Rule

~~Algorithm~~ Program

Composite Trapezoidal Rule

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

float fx(float x){
    return log(x);
}
int main(){
    float a,b,n,h,i,k,ans;
    printf("Enter upper limit for integration(b):");
    scanf("%f", &b);
    printf("Enter lower limit for integration(a):");
    scanf("%f",&a);
    printf("Enter number of intervals:");
    scanf("%f",&n);
    h=(b-a)/n;
    ans=fx(b)+fx(a);
    for (i=1;i<=n-1;i++){
        k=a+i*h;
        ans += 2*fx(k);
    }
    ans=(h*ans)/2;
    printf("Area = %f",ans);
}
```

Simpson's 1/3 Rule

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

float fx(float x){
    return 1/(1+x*x);
}
int main(){
    float a,b,n,h,k,ans;
    int i;
    printf("Enter upper limit for integration(b):");
    scanf("%f",&b);
    printf("Enter lower limit for integration(a):");
    scanf("%f",&a);
    printf("Enter number of intervals:");
    scanf("%f",&n);
```

Discussion

```
ans=fx(b)+fx(a);
for (i=1;i<=n-1;i++){
    k=a+i*h;
    if (i%2==0){
        ans += 2 * fx(k);
    } else {
        ans += 4* fx(k);
    }
    ans *= ~~3*h/8~~ h/3;
    printf("Area=%f",ans);
}
```

Simpson's 3/8 Rule

```
#include <stdio.h>
#include <math.h>
float fx(float x){
    return 1/(1+x*x);
}
int main(){
    float a,b,n,h,k,ans;
    int i;
    printf("Enter upper limit for integration(b):");
    scanf("%f",&b);
    printf("Enter lower limit for integration(a):");
    scanf("%f",&a);
    printf("Enter number of intervals:");
    scanf("%f",&n);
    h=(b-a)/n;
    ans=fx(b)+fx(a);
    for (i=1; i<=n-1; i++){
        k=a+i*h;
        if(i%3==0){
            ans += 2 * fx(k);
        } else {
            ans += 3* fx(k);
    }
    ans *= 3*h/8;
    printf("Area=%f",ans);
}
```

Discussion

Composite Trapezoidal Rule

Firstly, the program defines the function. Then, it asks the user to input upper limit (b) and lower limit (a) and ~~step size~~ intervals (n). Using the composite trapezoidal rule, it then calculates the value using the formula and displays the output

Simpson's 1/3 Rule

The function is defined first. Then, it prompts the user to input upper & lower limit and number of intervals. Then it calculates the ~~interval~~ step size and the loop is done to multiply even with 2 and odd with 4 and multiplies with h/3 and displays ~~with~~ the output

Simpson's 3/8 Rule

Initially, the function is defined. Then it prompts the user to input upper & lower limit and number of intervals. Then, it calculates the step size and the loop is run to multiply 2 with multiples of 3 and others with 3 and adds it with to a fn. Then it is multiplied with 3h/8 & displays the output.

Gauss Elimination Method

In gauss elimination method

- first matrix is converted to upper triangular matrix using forward elimination
- then the unknown variables are calculated using backward substitution method.

The diagonal element is considered as pivot element in each column.

Gauss Seidel Method

Here, the given equation should be diagonally dominant. The next approximation of unknown variables are calculated using the latest calculated values of unknowns.

$$x_1 = \frac{b_1 - a_{12}b - a_{13}c}{a_{11}}$$

$$x_2 = \frac{b_2 - a_{21}a - a_{23}c}{a_{22}}$$

$$x_3 = \frac{b_3 - a_{31}a - a_{32}b}{a_{33}}$$

From Gauss seidal method (Assuming Initial Values as 0)

$$x_1^{(1)} = \frac{b_1 - a_{12}b - a_{13}c}{a_{11}}$$

$$x_2^{(1)} = \frac{b_2 - a_{21}x_1^{(1)} - a_{23}c}{a_{22}}$$

$$x_3^{(1)} = \frac{b_3 - a_{31}x_1^{(1)} - a_{32}x_2^{(1)}}{a_{33}}$$

CamScanner

Algorithm

Gauss Elimination Method

1. Enter the order of matrix: n
2. Read Augmented Matrix (a) of n x n+1 size
   for i=1 to n
   for j=1 to n+1
   input a[i][j]
3. Transform Augmented Matrix (a) to Upper Triangular Matrix by Row Operations.
   for j=1 to n
   if (a[j][j] == 0) error and exit
   for i=1 to n
   if (i>j)
   ratio = a[i][j]/a[j][j]
   for k=1 to n
   a[i][k] = a[i][k] - ratio * a[j][k]
   next k
   end if
   next j
   next i
4. Obtain Solution by Back Substitution
   x[n] = a[n][n+1]/a[n][n]
   for i=n-1 to 1
   x[i] = a[i][n+1]
   for j=i+1 to n
   x[i] = x[i] - a[i][j] * x[j]
   next j
   x[i] = x[i]/a[i][i]
   next i
5. Display Result x[i] ~~for~~ from 1 to n.
6. end

CamScanner

Gauss Seidal Method

1. Arrange given system of linear equation in diagonally dominant form
2. Input tolerable error value
3. Re arrange the first equation in terms of first variable, second equation in terms of second variable & so on
4. Set initial guesses for x0, y0, z0 as zero.
5. Substitute value of y0, z0 in equations obtained from step 3 to calculate new value of x1.
6. Use x1, z0, in second equation obtained from step 3 to calculate new value of y1
7. Similarly, use x1, y1, to find new z1 and so on.
8. If $|x0-x1|>e$ and $|y0-y1|>e$ and $|z0-z1|>e$ and so on then go to step 9.
   else set x0 = x1, y0 = y1, z0 = z1 and so on & goto step 5
9. Print value of x1, y1, z1 and so on

CamScanner

Program

Gauss Elimination Method

```
#include <stdio.h>
int main(){
    float a[10][10],x[10],ratio;
    int i,j,k,n;
    printf("Enter the order of matrix:");
    scanf("%d",&n);
    printf("Enter the value of:\n");
    for (i=1;i<=n;i++){
        for (j=1;j<=n+1;j++){
            printf("a[%d][%d]:",i,j);
            scanf("%f",&a[i][j]);
        }
    }
    for (j=1;j<=n;j++){
        if (a[j][j]==0){
            return 1;
        }
        for (i=1;i<=n;i++){
            if (i>j){
                ratio = a[i][j]/a[j][j];
                for (k=1;k<=n+1;k++){
                    a[i][k] = a[i][k] - ratio * a[j][k];
                }
            }
        }
    }
    x[n] = a[n][n+1]/a[n][n];
    for(i=n-1;i>=1;i--){
        x[i] = a[i][n+1];
        for (j=i+1;j<=n;j++){
            x[i] = x[i] - a[i][j] * x[j];
        }
        x[i] = x[i]/a[i][i];
    }
    for (i=1;i<=n;i++){
        printf("x[%d] = %f\n",i,x[i]);
    }return 0;
}
```

Gauss Seidel Method

```
#include <stdio.h>
#include <math.h>
float first(float y, float z){
    return (17-y+2*z)/20;
}
float second(float x, float z){
    return (-18-3*x+z)/20;
}
float third (float x, float y){
    return (25-2*x+3*y)/20;
}
int main(){
    float e, x0=0, y0=0, z0=0, x1, y1, z1;
    printf("Enter tolerable error:");
    scanf("%f", &e);
repeat:
    x1 = first(y0, z0);
    y1 = second(x1, z0);
    z1 = third(x1, y1);
    if ((fabs(x0-x1)>e) && (fabs(y0-y1)>e) &&
        (fabs(z0-z1)>e)){
        x0 = x1;
        y0 = y1;
        z0 = z1;
        goto repeat;
    } else {
        goto print;
    }
print:
    printf("x = %f\ny = %f\n z = %f", x1, y1, z1);
    return 0;
}
```

Discussion

Gauss Elimination Method

Initially, the user is prompted to input the elements of the matrix after it is asked for it's order. If zero diagonal element is encountered during forward elimination, it displays error & terminates. Otherwise, it proceeds with forward elimination & back substitution steps to obtain the solution vector. Finally, the result is printed

Gauss Seidel Method

Firstly, 3 functions are defined to represent equations. Then, the program iteratively computes the values of x, y, z using these functions until the error reaches the user specified tolerance. Once, the convergence criteria is met, it displays the solution of the equation

CamScanner

Euler's Method

It is a numerical technique used to solve ordinary differential equation (ODE) with initial conditions. It is a simple & straightforward method that approximation the solution of an ODE by discretizing the independent variable into small intervals & approximating the derivative with a finite difference

$$y_{i+1} = y_i + h f(x_i, y_i)$$

RK-4 Method

The RK-4 method is a fourth order method which approximates the curve of the function f betn two points by a 4th degree polynomial.

$$y_{i+1} = y_i + \frac{h}{6}(m_1 + 2m_2 + 2m_3 + m_4)$$

CamScanner

Algorithm

Euler's Method

1. Input x0, y0, h and xp; xp is the calculation point
2. n = (xp-x0)/h
3. for i=0 to n
4. yi+1 = yi + h * f(xi, yi)
5. xi+1 = x0+h
6. Print values of xi & yi
7. next i
8. end

RK-4 Method

1. Define function f(x,y)
2. Input x0, y0, h and xp; xp is the calculation point
3. n = (xp - x0)/h
4. for i=0 to n-1
5. xi+1 = xi+h
6. m1 = f(xi, yi)
7. m2 = f(xi + h/2, yi+m1h/2)
8. m3 = f(xi+h/2, yi+m2h/2)
9. m4 = f(xi+h, yi+m3h)
10. yi+1 = yi+h/6 (m1+2m2+2m3+m4)
11. print xi+1, yi+1
12. next i
13. end

CamScanner

Program

Eulers Method

```
#include <stdio.h>
float f(float x,float y){
    return x+y;
}
int main(){
    float x[10],y[10], h,xp,n;
    printf("Enter value of x0 & y0:");
    scanf("%f%f",&x[0],&y[0]);
    printf("Enter value of h:");
    scanf("%f",&h);
    printf("Enter value of xp:");
    scanf("%f",&xp);
    n=(xp-x[0])/h;
    for(int i=0;i<=n;i++){
        y[i+1]=y[i]+h*f(x[i],y[i]);
        x[i+1]=x[i]+h;
        printf("x[%d],y[%d]=%.2f\n",i,i,x[i],y[i]);
    }
    return 0;
}
```

RK-4 Method

```
#include <stdio.h>
float f(float x, float y){
    return x+y;
}
int main(){
    float x[10],y[10],h,xp,n,m1,m2,m3,m4;
    printf("Enter value of x0 & y0:");
    scanf("%f %f",&x[0],&y[0]);
    printf("Enter value of h:");
    scanf("%f",&h);
    printf("Enter value of xp:");
    scanf("%f",&xp);
    n=(xp-x[0])/h;
    for(int i=0;i<=n-1;i++){
        x[i+1]=x[i]+h;
        m1=f(x[i],y[i]);
        m2=f(x[i]+h/2,y[i]+m1*h/2);
        m3=f(x[i]+h/2,y[i]+m2*h/2);
        m4=f(x[i]+h,y[i]+m3*h);
        y[i+1]=y[i]+h/6*(m1+2*m2+2*m3+m4);
        printf("x[%d],y[%d]=%.2f,%.2f\n",i+1,
        i+1,x[i+1],y[i+1];
    }
    return 0;
}
```

Discussion

Euler's Method

The given program uses Euler's method to approximately calculate the solution of first order ordinary differential equation (ODE) with given initial conditions. It prompts the user to input the initial value & step size and the calculated point. The program then computes the solution iteratively within a loop, updating the value of y at each step using formula of Euler's forward difference

RK-4 Method

The given program implements the Runge-Kutta Fourth Order (RK-4) method to approximate the solution of a first-order ordinary differential equation (ODE) of the form $\frac{dy}{dx} = f(x, y)$ with given initial conditions. It prompts the user to input the initial values step size & the calculation point. Then, in a loop, it computes four intermediate slopes to approximate change. Finally, it displays the values.

CamScanner