INDEX

Experiment No.	Name of the Experiment	Remarks
1	Bisection Method	
2	Regula Falsi Method	
3	Newton Forward Interpolation Method	
4	Newton Backward Interpolation Method	
5	Gauss Elimination Method	
6	Gauss-Jordan Method	
7	Jacobi Iteration Method	
8	Gauss Seidel Method	

Objectives:

The objective of the sessional is to solve numerical algebraic equation by using Bisection Method.The main objective is to find the root of a function.

Theory:

Let a function f(x) be continuous between a and b.For definiteness,let f(a) be negative and f(b) be positive.Then there is a root of f(x)=0,lying between a and b.Let the first approximation be x_1=$\frac{1}{2}$(a+b).

Now if f(x_1)=0 then x_1 is a root of f(x)=0.Otherwisxe, the root will lie between a and x_1 or x_1 and b depending on whether f(x_0) is positive or negative.

Then,as before,we bisect the interval and and continue the process til the root is found to the desired accuracy.

The method is simple.It is also called as Botzano method or Interval having method.

Input & Output :

INPUT	OUTPUT
3 1 0 -1 -11 2 3	The root of the equation is : 2.37365

Discussion:

Bisection method is a method to solve an algebraic equation and find out the root of the given function .From that sessional class we learn the process of finding the root in a given interval.

We also learn that, untill the difference between the given interval is less than or equal to DOA(Degree of Accuracy e.g.=0.001) this process will continue.

Objectives:

The objective of the sessional is to solve numerical algebraic equation by using Regula Falsi Method.The main objective is to find the root of a function.

Theory:

Consider the equation f(x)=0.let a and b (a<b) be two values of x such that f(a) and f(b) are opposite signs.Then the graph of y=f(x) crosses the x-axis at some points between a and b.

Therefore,the equation of the chord joining the two points A[a,f(a)] and B[b,f(b)] is

y-f(a) = $\frac{f(b)-f(a)}{b-a}$ (x-a)……(1)

Now in the interval,(a,b) the graph of the function can be considered as a straight line.So the intersection of the line given by eqn-1 with the x-axis will give an approximate value of the root.

Putting y=0 in eqn(1) we get

f(a) = $\frac{f(b)-f(a)}{b-a}$ (x-a)

⇨ x = $\frac{af(a)-bf(a)}{f(b)-f(a)}$

Hence the first approxximate to the root is given by

x_1 = $\frac{af(b)-bf(a)}{f(b)-f(a)}$

Now if f(x_1) and f(a) are of opposite sign then the root lies between a and x_1.So, we replace b by x_1 in eqn(2) and get the next approximation x_2.

Input & Output :

INPUT	OUTPUT
3 1 0 -1 -11 2 3	The root of the equation is : 2.37365

Discussion:

Regula Falsi method is a process of solving numerical algebraic and trancendental equations and find out the roots of the given function.we,mainly find the root from given interval.

We also learn that sessional is as the functions of two interval must be in opposite sign and their multiplication must be less than zero. We also check the differnce between the interval.

Objectives :

Our main objective of this report is to solve problem using Newton's Forward Interpolation.

Theory :

Let $y = f(x)$ be a finction which takes value $y_0, y_1, \dots \dots . y_n$ for $(n+1)$ values $x_0, x_1, \dots \dots \dots . x_n$ of the independent variable x (argument). Let the values of x be equidistant, i.e. $xi = x0 + ih$ where $i = 0,1,2, \dots \dots \dots . n$ and let $y(x)$ be the polynomial of the nth degree –

$$y(x) = A_0 + A_1(x - x_0) + A_2(x - x_0)(x - x_1) + A_3(x - x_0)(x - x_1)(x - x_2) + \dots \dots \dots \dots + A_n(x - x_0)(x - x_1) \dots \dots \dots (x - x_{n-1}) \quad \dots \dots \dots \dots \dots \dots . (1)$$

Putting $x = x_0$:

$$y_0 = A_0 + 0 + 0 + 0 \dots$$

$$\therefore y_0 = A_0$$

Putting $= x_1$:

$$y_1 = A_0 + A_1(x_1 - x_0) + A_2(x_1 - x_0)(x_1 - x_1) + \dots \dots \dots \dots$$

$$y_1 = A_0 + A_1(x_1 - x_0) + 0 + \dots \dots \dots \dots$$

$$y_1 = y_0 + A_1(x_1 - x_0)$$

$$\therefore A_1 = \frac{y_1 - y_0}{x_1 - x_0}$$

$$\therefore A_1 = \frac{\Delta y}{h}$$

Similarly putting $x = x_2, x_3, \dots . x_3$

$$A_2 = \frac{\Delta^2 y_0}{2!\ h^2}$$

$$A_3 = \frac{\Delta^3 y_0}{3!\ h^3}$$

……

Putting values in equation (1)

$$y_{(x)} = y_0 + \frac{\Delta y}{h}(x - x_0) + \frac{\Delta^2 y_0}{2!\,h^2}(x - x_0)(x - x_1) + \frac{\Delta^3 y_0}{3!\,h^3}(x - x_0)(x - x_1)(x - x_2) + \dots \dots \dots$$

$$y_{(x)} = y_0 + p\Delta y_0 + \frac{\Delta^2 y_0}{2!}p(p-1) + \frac{\Delta^3 y_0}{3!}p(p-1)(p-2) + \dots \dots \dots$$

where $p = \frac{x - x_0}{h}$

So, the final equation is –

$$y_{(x)} = y_0 + p\Delta y_0 + \frac{\Delta^2 y_0}{2!}p(p-1) + \frac{\Delta^3 y_0}{3!}p(p-1)(p-2) + \ldots\ldots\ldots\ldots + \frac{p(p-1)(p-2)\ldots\{p-(n-1)\}}{h!}\Delta^n y_0$$

We used the formula in our code.

Input & Output :

INPUT	OUTPUT
5 8 1000 10 1900 12 3250 14 5400 16 8950 9	The desired y is : 1405.8594

Discussion :

In this experiment we learned to find the y for a given x from a table of given values. This method only works if the values are equidistance.

Objective:

The main objective of this report is to find value of function for any intermediate values of the independent variable b using Gregory-Newton backward interpolation formula.

Theory:

Let y=f(x) be a function which takes the values y0,y1,y2,…yn for the (n+1) values of x0,x1,…..xn of independent variable x0 let, x1=x0+ih,i=0,1,2…n and y(x) be the polynomial in x of nth degree. We can assume that,

$$y_p = y_0 + p\Delta y_0 + \frac{p(p-1)}{2!}\Delta^2 y_0 + \frac{p(p-1)(p-2)}{3!}\Delta^3 y_0 + \ldots\ldots\ldots + \frac{p(p-1)(p-2)\ldots[p-(n-1)]}{n!}\Delta^n y_0$$

Where, p=x-x0; p is real number

Input & Output:

INPUT	OUTPUT
4 2 94.8 5 87.9 8 81.3 11 75.1 9	The result = 79.18395

Discussion:

From our sessional, we learn the process to find the value of a function for any intermediate values which is located at the first sides of given functions.

Objectives:

Our main objective of this sessional is to code the solution of linear equation by the method of Gauss Elimination using MATLAB.

Theory:

Let the linear equation be,

$a_1x + b_1y + c_1z = d_1$

$a_2x + b_2y + c_2z = d_2$

$a_3x + b_3y + c_3z = d_3$

Where x, y, z are unknown. The system in matrix form is

AX = B where

A is given below

$$\begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{pmatrix}$$

X is given below

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

B is given below

$$\begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix}$$

Then the argumented matrix will be

[A | B]

$$\left(\begin{array}{ccc|c} a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \end{array}\right)$$

The above matrix is reduced to a upper traingluer matrix, then,

R_2' -> R_2 - $\frac{a_2}{a_1} R_1$

R_3' -> R_3 - $\frac{a_2}{a_1} R_1$

$$\begin{pmatrix} a_1 & b_1 & c_1 & d_1 \\ 0 & b_2' & c_2' & d_2' \\ 0 & b_3' & c_3' & d_3' \end{pmatrix}$$

Later

R_3' -> $R_3 - \frac{b'_3}{b'_2} R_2$

$$\begin{pmatrix} a_1 & b_1 & c_1 & d_1 \\ 0 & b_2' & c_2' & d_2' \\ 0 & 0 & c_3'' & d_3'' \end{pmatrix}$$

If c_3" is not equal zero then the given system of linear equations will be equivalent to

$$a_1 x + b_1 y + c_1 z = d_1$$

$$b_2 + c_2' z = d_2'$$

$$c_3' z = d_3''$$

Using substitution

$$z = \frac{d_3''}{c_3''}$$

$$y = \frac{d_2' c_3'' + c_2' d_3''}{b_2' c_3''}$$

$$x = \frac{d_1 b_2' c_3 + b_1 c_2' d_3 + b_2 c_2' d_3''}{a_1 b_2' c_3''}$$

There we used all these formula to code the program using MATLAB.

Input & Output :

INPUT	OUTPUT
3 3 1 -1 3 2 -8 1 -5 1 -2 9 8	1 1 1

Discussion :

In this lab we learned to use the Gauss-Elimination method to solve simultaneous linear equations. This method fails if the pivot becomes zero.

Objectives:

Our main objective of this sessional is to code the solution of linear equation by the method of Gauss-Jordan using MATLAB.

Theory:

This method is a modified form of Gauss elimination method.In this method,the co-efficient matrix A of $AX = B$ is reduced to a diagonal matrixor unit matrix by making all the elements above and below to the principle diagonal of A as zero.The labour of back substitution is saved here even though it involves additional computations.

Input & Output :

INPUT	OUTPUT
3 10 1 1 12 2 10 1 13 1 1 5 7	1 1 1

Discussion:

In this lab we learned to use the Gauss-Jordan method to solve simultaneous linear equations. This method fails if the pivot becomes zero.

Objective :

Our main objective is to code a solution for linear algebraic equation by using Jacobi Iteration method.

Theory :

This method is also known as Gauss-Jacobi method.

If we consider the following system of equations –

$a_1x + b_1y + c_1z = d_1$ ..... (1)

$a_2x + b_2y + c_2z = d_2$ ..... (2)

$a_3x + b_3y + c_3z = d_3$ ..... (3)

Then to apply Jacobi Iteration method, following conditions should be fulfilled –

$|a_1| > |b_1| + |c_1|$

$|b2| > |a2| + |c2|$

$|c3| > |a3| + |b3|$

That is, in each equation the coefficients of the diagonal terms are large. Solving for x, y and z respectively

$x = \frac{1}{a_1}(d_1 - b_1y - c_1z)$..... (4)

$y = \frac{1}{b_2}(d_2 - a_2x - c_2z)$..... (5)

$z = \frac{1}{c_3}(d_3 - a_3x - b_3y)$..... (6)

1st iteration :

Let, $x_0 = 0$, $y_0 = 0$ and $z_0 = 0$

So,

$$x_1 = \frac{1}{a_1}(d_1 - b_1y_0 - c_1z_0)$$

$$y_1 = \frac{1}{b_2}(d_2 - a_2x_0 - c_2z_0)$$

$$z_1 = \frac{1}{c_3}(d_3 - a_3x_0 - b_3y_0)$$

2nd iteration :

Putting the values of x_1, y_1 and z_1 in equation (4), (5) and (6) –

$$x_2 = \frac{1}{a_1}(d_1 - b_1 y_1 - c_1 z_1)$$

$$y_2 = \frac{1}{b_2}(d_2 - a_2 x_1 - c_2 z_1)$$

$$z_2 = \frac{1}{c_3}(d_3 - a_3 x_1 - b_3 y_1)$$

Proceeding the same way, if x_r, y_r, z_r are the rth iterates then –

$$x_{r+1} = \frac{1}{a_1}(d_1 - b_1 y_r - c_1 z_r)$$

$$y_{r+1} = \frac{1}{b_2}(d_2 - a_2 x_r - c_2 z_r)$$

$$z_{r+1} = \frac{1}{c_3}(d_3 - a_3 x_r - b_3 y_r)$$

The process is continued till convergence is secured.

Input & Output :

INPUT	OUTPUT
8 -3 2 20 4 11 -1 33 6 3 12 35	3.0167 1.9858 0.9118

Discussion :

In this experiment we learned to solve simultaneous linear equations using Jacobi Iteration method. This method is only applicable if the condition holds.

Objectives:

Our main objective of this sessional is to code the solution of linear equation by the method of Gauss Seidel using MATLAB.

Theory:

This is a modification of Gauss – Jacobi method. Let, the system of the linear equations,

$a_1x + b_1y + c_1z = d_1$

$a_2x + b_2y + c_2z = d_2$

$a_3x + b_3y + c_3z = d_3$

Where x, y,z are unknown.

The equation can be written as,

$$x = \frac{1}{a_1}(d_1 - b_1y - c_1z) \dots\dots\dots\dots\dots\dots\dots\dots (1)$$

$$y = \frac{1}{b_2}(d_2 - a_2x - c_2z) \dots\dots\dots\dots\dots\dots\dots\dots (2)$$

$$z = \frac{1}{c_3}(d_3 - a_3x - b_3y) \dots\dots\dots\dots\dots\dots\dots\dots (3)$$

And we start with the initial approximation x_0, y_0, z_0. Substituting y_0, z_0 in equation (1), we get,

$$x_1 = \frac{1}{a_1}(d_1 - b_1y_0 - c_1z_0)$$

Now substituting $x = x_1, z = z_0$ in equation (2), we get,

$$y_1 = \frac{1}{b_2}(d_2 - a_2x_1 - c_2z_0)$$

Now, substituting $x = x_1$, $y = y_1$ in equation (3)

$$z_1 = \frac{1}{c_3}(d_3 - a_3x_1 - b_3y_1)$$

This process is continued till the values of x, y, z are obtained to the desire degree of accuracy. The general algorithm is as follow:

If the x_r, y_r, z_r are the r th iteration, then

$$x_{r+1} = \frac{1}{a_1}(d_1 - b_1 y_r - c_1 z_r)$$

$$y_{r+1} = \frac{1}{b_2}(d_2 - a_2 x_{r+1} - c_2 z_r)$$

$$z_{r+1} = \frac{1}{c_3}(d_3 - a_3 x_{r+1} - b_3 y_{r+1})$$

Since the current values of the unknowns at each stage of iteration are used in proceeding to the next stage of iteration, this method is more rapid in convergence than Gauss – Seidel method.

Input & Output :

INPUT	OUTPUT
28 4 -1 32 2 17 4 35 1 3 10 24	0.9936 1.5069 1.8485

Discussion:

In this experiment we learned to solve simultaneous linear equations using the Gauss-Seidel Method. This method is only applicable when the condition holds.

D:\PROGRAMMING TASK\Neumerical\Bisection.cpp

```cpp
#include<stdio.h>
#include<math.h>
const double ACC=0.001;
double low,high,mid,fa,fb,fc;
int i,deg,arr[15];
int main()
{
    //freopen("input.txt","r",stdin);//input.in
    //freopen("output.txt","w",stdout);//output.out
    while(scanf("%d",&deg)==1){
        for(i=deg;i>=0;i--) scanf("%d",&arr[i]);
        scanf("%lf%lf",&low,&high);
        fa=fb=0.0;
        for(i=deg;i>=0;i--){
            fa+=arr[i]*pow(low,i);
            fb+=arr[i]*pow(high,i);
        }
        if((int)(fa*fb)>0) printf("No solution.\n");
        else{
            while(fabs(low-high)>ACC){
                mid=(low+high)/2;
                for(fc=0.0,i=deg;i>=0;i--){
                    fc+=arr[i]*pow(mid,i);
                }
                if(fc<0.0) low=mid;
                else high=mid;
            }
            printf("root: %.3lf\n",mid);
        }
    }
    return 0;
}

```

D:\PROGRAMMING TASK\Neumerical\Regule Falsi.cpp

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

const double ACC=0.001;
double low,high,mid,fa,fb,fc;
int i,deg,arr[15];

double func(double a,double c=0.0){
    for(int i=deg;i>=0;i--)
        c+=arr[i]*pow(a,i);
    return c;
}
int main()
{
    //freopen("input.txt","r",stdin);
    //freopen("output.txt","w",stdout);
    while(scanf("%d",&deg)==1){
        for(i=deg;i>=0;i--) scanf("%d",&arr[i]);
        scanf("%lf%lf",&low,&high);
        fa=func(low);
        fb=func(high);
        if(fa*fb>0){
            printf("No solution.\n");
            continue;
        }
        while(fabs(low-high)>ACC){
            mid=(low*fb-high*fa)/(fb-fa);
            fc=func(mid);
            if(fc<0.0) low=mid;
            else high=mid;
        }
        printf("root: %.3lf\n",mid);
    }
    return 0;
}

```

```cpp
#include<stdio.h>
#include<vector>
using namespace std;

int fact[15]={1};
double X[15],Y[15],arr[15][15];
vector<double>needy;
int N,i,j,size;
double p,x,sum;

void facto(){
    int i;
    for(i=1;i<11;i++){
        fact[i]=fact[i-1]*i;
    }
    return;
}

double p_mult(int x){
    double xx=1.0;
    for(i=0;i<x;i++){
        xx*=p-i;
    }
    return xx;
}

int main()
{
    //freopen("input.txt","r",stdin);
    facto();
    while(scanf("%d",&N)==1){
        for(i=0;i<N;i++){
            scanf("%lf",&X[i]);
        }
        for(i=0;i<N;i++){
            scanf("%lf",&Y[i]);
            arr[0][i]=Y[i];
        }
        needy.push_back(Y[0]);
        scanf("%lf",&x);
        p=(x-X[0])/(X[1]-X[0]);
        size=i;
        for(i=0;size!=1;i++){
            for(j=0;j<size-1;j++){
                arr[i+1][j]=arr[i][j+1]-arr[i][j];
            }
            size=j;
            needy.push_back(arr[i+1][0]);
        }
        size=needy.size();
        sum=needy[0];
        for(i=1;i<size;i++){
            sum+=p_mult(i)*needy[i]/fact[i];
        }
        printf("%.3lf\n",sum);
        needy.clear();
    }
    return 0;
}

```

D:\PROGRAMMING TASK\Numerical\Gauss Elimination.m

```
clear
row=input('Input dimension');
col=row+1;
for i=1:row
    for j=1:col
        arr(i,j)=input('input value');
    end
end
for k=2:row
    pivot=arr(k-1,k-1);
    for i=k:row
        up=arr(i,k-1);
        for j=1:col
            arr(i,j)=arr(i,j)-(up/pivot)*arr(k-1,j);
        end
    end
end
for i=row:-1:1
    sol(i)=arr(i,col-1);
    for j=col-2:-1:1
        if(j==i) continue;
        end
        sol(i)=sol(i)-arr(i,j)*sol(j);
    end
    sol(i)=sol(i)/arr(i,i);
end
sol
```

```cpp
#include<stdio.h>
#include<vector>
using namespace std;

int fact[15]={1};
double X[15],Y[15],arr[15][15];
vector<double>needy;
int N,i,j,size;
double p,x,sum;

void facto(){
    int i;
    for(i=1;i<11;i++){
        fact[i]=fact[i-1]*i;
    }
    return;
}

double p_mult(int x){
    double xx=1.0;
    for(i=0;i<x;i++){
        xx*=p+i;
    }
    return xx;
}

int main()
{
    //freopen("input.txt","r",stdin);
    facto();
    while(scanf("%d",&N)==1){
        for(i=0;i<N;i++){
            scanf("%lf",&X[i]);
        }
        for(i=0;i<N;i++){
            scanf("%lf",&Y[i]);
            arr[0][i]=Y[i];
        }
        needy.push_back(Y[N-1]);
        scanf("%lf",&x);
        p=(x-X[N-1])/(X[1]-X[0]);
        size=i;
        for(i=0;size!=1;i++){
            for(j=0;j<size-1;j++){
                arr[i+1][j]=arr[i][j+1]-arr[i][j];
            }
            size=j;
            needy.push_back(arr[i+1][size-1]);
        }
        size=needy.size();
        sum=needy[0];
        for(i=1;i<size;i++){
            sum+=p_mult(i)*needy[i]/fact[i];
        }
        printf("%.3lf\n",sum);
        needy.clear();
    }
    return 0;
}

```

D:\PROGRAMMING TASK\Neumerical\gauss jordan.m

```
clear
row=input('input dimension');
for i=1:row
    for j=1:row
        arr(i,j)=input('input value ');
    end
    arrb(i)=input('input value ');
end
for i=1:row
    c=abs(arr(i,i));
    d=i;
    for j=i:row
        if abs(arr(j,i))>c
            c=abs(arr(j,i));
            d=j;
        end
    end
    if(c==0)
        break
    end
    for j=1:row
        e=arr(i,j);
        arr(i,j)=arr(d,j);
        arr(d,j)=e;
    end
    e=arrb(i);
    arrb(i)=arrb(d);
    arrb(d)=e;
    c=arr(i,i);
    e=(c-1)/c;
    for j=1:row
        arr(i,j)=arr(i,j)-e*arr(i,j);
    end
    arrb(i)=arrb(i)-e*arrb(i);
    c=arr(i,i);
    for j=1:row
        if(j==i)
            continue
        end
        d=arr(j,i)/c;
        for k=1:row
            arr(j,k)=arr(j,k)-d*arr(i,k);
        end
        arrb(j)=arrb(j)-d*arrb(i);
    end
    arr
end
if (c==0)
    disp('Not Possible')
else
    b
end
```

D:\PROGRAMMING TASK\Neumerical\jacobi_iteration.m

```
clear
a(1,1)=input('Enter a1: ');
a(1,2)=input('Enter b1: ');
a(1,3)=input('Enter c1: ');
a(1,4)=input('Enter d1: ');
a(2,1)=input('Enter a2: ');
a(2,2)=input('Enter b2: ');
a(2,3)=input('Enter c2: ');
a(2,4)=input('Enter d2: ');
a(3,1)=input('Enter a3: ');
a(3,2)=input('Enter b3: ');
a(3,3)=input('Enter c3: ');
a(3,4)=input('Enter d3: ');
ACC=0.0001;
if ( (abs(a(1,1))>abs(a(1,2))+abs(a(1,3))) && (abs(a(2,2))>abs(a(2,1))+abs(a(2,3)))
 && (abs(a(3,3))>abs(a(3,1))+abs(a(3,2))))
    prevx=0;
    prevy=0;
    prevz=0;
    while(1)
        x=(a(1,4)-a(1,2)*prevy-a(1,3)*prevz)/a(1,1);
        y=(a(2,4)-a(2,1)*prevx-a(2,3)*prevz)/a(2,2);
        z=(a(3,4)-a(3,1)*prevx-a(3,2)*prevy)/a(3,3);
        if(abs(x-prevx)<ACC && abs(y-prevy)<ACC && abs(z-prevz)<ACC)
            break;
        end
        prevx=x;
        prevy=y;
        prevz=z;
    end
    x
    y
    z
else
    disp('Not possible');
end
```

D:\PROGRAMMING TASK\Neumerical\gauss_seidel.m

```
clear
a(1,1)=input('Enter a1: ');
a(1,2)=input('Enter b1: ');
a(1,3)=input('Enter c1: ');
a(1,4)=input('Enter d1: ');
a(2,1)=input('Enter a2: ');
a(2,2)=input('Enter b2: ');
a(2,3)=input('Enter c2: ');
a(2,4)=input('Enter d2: ');
a(3,1)=input('Enter a3: ');
a(3,2)=input('Enter b3: ');
a(3,3)=input('Enter c3: ');
a(3,4)=input('Enter d3: ');
ACC=0.0001;
if ( (abs(a(1,1))>abs(a(1,2))+abs(a(1,3))) && (abs(a(2,2))>abs(a(2,1))+abs(a(2,3)))
 && (abs(a(3,3))>abs(a(3,1))+abs(a(3,2))))
    prevx=0;
    prevy=0;
    prevz=0;
    x=0;
    y=0;
    z=0;
    while(1)
        x=(a(1,4)-a(1,2)*y-a(1,3)*z)/a(1,1);
        y=(a(2,4)-a(2,1)*x-a(2,3)*z)/a(2,2);
        z=(a(3,4)-a(3,1)*x-a(3,2)*y)/a(3,3);
        if(abs(x-prevx)<ACC && abs(y-prevy)<ACC && abs(z-prevz)<ACC)
            break;
        end
        prevx=x;
        prevy=y;
        prevz=z;
    end
    x
    y
    z
else
    disp('Not possible');
end
```