RV Educational Institutions ®
RV College of Engineering ®
Autonomous Institution Affiliated to Visvesvaraya Technological University, Belagavi
Approved by AICTE, New Delhi,

Go, change the world

Computational Advanced Numerical Methods (18G5B16)

Experiential Learning Report

Submitted By

Ananya G (1RV20IS007)
Dileep Sharma (1RV20IS017)
Karthik Pai (1RV20IS020)
Mayank Somani (1RV20IS069)

Under The Guidance of

Prof. Rajashekhar P.L.

Department of Mathematics

2022-2023

RV COLLEGE OF ENGINEERING®, BENGALURU-59

(Autonomous Institution Affiliated to VTU, Belagavi)

DEPARTMENT OF MATHEMATICS

CERTIFICATE

Certified that the Matlab Code execution Experiential Learning is successfully carried out by Ananya G (1RV20IS007), Dillep Sharma (1RV20IS017), Karthik Pai (1RV20IS020) and Mayank Somani (1RV20IS069) in partial fulfilment of the completion of the course Computational Advanced Numerical Methods (18G5B16) of the V Sem, during the academic year 2022-2023. It is certified that all corrections/suggestions indicated for the Internal Assessment have been incorporated in the project report and duly approved by the faculty.

Signature of Faculty

1.Power Method

The power method is an iterative algorithm used to find the dominant eigenvector and eigenvalue of a matrix. The dominant eigenvector is the eigenvector corresponding to the largest eigenvalue of the matrix. The power method works by repeatedly multiplying the matrix by a vector, normalizing the result, and using it as the new vector in the next iteration. The dominant eigenvalue can be calculated as the ratio of the dot product of the resulting eigenvector and the matrix with the resulting eigenvector.

Algorithm:

Inputs:

- $Square\ Matrix\ A\ of\ size\ n\ *\ n\ (We\ have\ considered\ 3\ *\ 3\ )$
- $Initial\ Guess\ vector\ X\ of\ size\ n\ *\ 1\ (Here\ 3\ *\ 1)$

Steps:

1. $Start$
2. $Choose\ value\ of\ \ tolérance$
3. $Define\ Matrix\ \ Y\ =\ A\ *\ X$
4. $Define\ m1, m2\ =\ max(All\ values\ of\ Y)$
5. $Define\ err\ =\ Absolute(m1\ -\ m2)$
6. $while\ err\ >\ \ tolerance\ do$:
    a. $Y\ =\ A\ *\ X$
    b. $m2\ =\ max(All\ values\ of\ Y)$
    c. $X\ =\ Y/m2$
    d. $err\ =\ Absolute(m1\ -\ m2)$
    e. $m1\ =\ m2$
7. $Display\ X, m2$
8. *Stop*

Program:

Input: Predefined array of input with initial approximation of Eigenvector

Output: Largest Eigenvalue and vector with approx tolerance of 10^-6

Class: Contains functions utilised in finding largest Eigenvalue and vector

```
classdef PowerMethod
  properties
    A,tolerance,eig,X,n
  end
  methods (Access=public)
    function this=PowerMethod()
      this.A=[6 -2 2;-2 3 -1; 2 -1 3];
      this.X=[1;1;1];
      this.n=1;
      this.tolerance=10^-6;
    end
    function run(this)
```

```
        this=this.power();
        fprintf("The value of the largest eigen value is ")
        disp(this.eig)
        fprintf("The corresponding eigen vector is\n\n")
        disp(this.X(:,this.n))
    end
  end
  methods (Access=private)
    function this=power(this)
        A=this.A; %#ok<*PROP>
        X=this.X;
        tolerance=this.tolerance;
        m1=1;
        Y=A*X;
        m2=max(abs(Y));
        err=abs(m1-m2);
        while err>tolerance
            Y=vpa(A*X);
            m2=max(abs(Y));
            X=Y/m2;
            err=abs(m1-m2);
            m1=m2;
            this.n=this.n+1;
            this.X=[this.X,X];
        end
        this.eig=m2;
    end
  end
end
```

Syntax to run:

```
powerMethod=PowerMethod();
powerMethod.run()
```

Sample Output:

```
>> runPowerMethod
The value of the largest eigen value is 7.999999821186071002898484382078 5

The correspoding eigen vector is

                                    1.0
-0.499999966472387563642952117003 86
 0.500000011175870812119015960998 71
```

2.Fixed Point Iteration Method

The Fixed Point Iteration Method is an open and simple method for finding the real root of a non-linear equation by successive approximation. It requires one initial guess to start. Since it is an open method its convergence is not guaranteed. This method is also known as the Iterative method.

Algorithm:

Inputs:

- $Non\ -\ linear\ function\ f(x)$
- $Initial\ value\ x0$

Steps:

1. $Start$
2. $Define\ function\ f(x)$
3. $Define\ function\ g(x)\ which\ is\ obtained\ from\ f(x)\ =\ 0\ such\ that\ x\ =\ g(x)\ and\ |g'(x)\ <\ 1|$
4. $Choose\ an\ initial\ value\ x0\ and\ tolerance\ e$
5. $Calculate\ x1\ =\ g(x0)$
6. $Set\ x0\ =\ x1\ for\ next\ iteration$
7. $If\ |f(x1)|\ >\ e\ then\ goto\ step\ (5)\ otherwise\ goto\ step\ (8)$
8. $Display\ x1\ as\ root$
9. $Stop$

Program:

Input: Predefined function f(x) and task is to find value of f at some given value x

Output: Value of f at given value x with tolerance of 10^-6

Code: Contains a class which has utility functions for doing the same

```
classdef FixedPointIteration
  properties(Access=private)
    tolerance,f,g,x0,X1,n
  end
  methods(Access=public)
     function this=FixedPointIteration()
        this.tolerance=10^-6;
        this.f=@(x)x^4-x-10;
        this.g=@(x)(x+10)^(1/4);
        this.x0=2;
        this.n=1;
     end
     function run(this)
```

```
        this=this.fixedPointIteration();
        fprintf("The value of x1 calculate is ")
        disp(this.X1(this.n))
    end

  end
  methods(Access=private)
    function this=fixedPointIteration(this)
        tolerance=this.tolerance;
        x0=this.x0; %#ok<*PROP,*PROPLC>
        g=this.g;
        x1=g(x0);
        error=abs(x1-x0);
        X=x1;
        while (error>tolerance)
            x0=x1;
            x1=vpa(g(x0));
            error=abs(x1-x0);
            X=[X x1]; %#ok<AGROW>
            this.n=this.n+1;
        end
        this.X1=X;
    end
  end
end
```

Syntax to run:

```
fixedPointIteration=FixedPointIteration();
fixedPointIteration.run()
```

Sample Output:

```
>> runFixedPointIteration
The value of x1 calculate is 1.855584541824958623231450164800 1
```

3.Cubic Spline Method

A cubic spline is a piecewise polynomial function that passes through a set of control points and has some smoothness conditions1. Cubic spline solutions to two-point boundary value problems are numerical methods that use cubic splines to approximate the solution of a differential equation with given boundary conditions

Algorithm:

Inputs:

- $f(x, y)$: $a\ function\ defining\ the\ differential\ equation$
- α, β: $boundary\ values$
- $a,\ b$: $interval\ boundaries$
- n: $number\ of\ subintervals$
- $Boundary\ value\ problem\ of\ the\ type$: $y'' + f(x)y'(x) + g(x)y = r(x)$

Steps:

1. $Start$
2. $Define\ h = (b - a)/n$
3. $Define\ a\ set\ of\ n + 1\ points\ \{x_i\}\ on\ the\ interval\ [a, b],\ where$

   $$x_i = a + ih\ for\ i = 0,\ 1,\ \ldots,\ n$$

4. $for\ i\ from\ 0\ to\ n\ do$:

   $$S'(x_i^-) = \frac{h}{3!}(2 * M_i + M_{i-1}) + \frac{1}{h}(y_i - y_{i-1})\ for\ i = 1, 2, 3 \ldots\ldots n$$

   $$S'(x_i^+) = \frac{-h}{3!}(2 * M_i + M_{i+1}) + \frac{1}{h}(y_{i+1} - y_i)\ for\ i = 0, 1, 2, 3 \ldots\ldots n - 1$$

5. $Change\ the\ given\ equation\ to$:

   $$M_i + f_i * S'(x_i) + g_i * y_i = r_i$$

   $where\ S(x) is\ the\ cubic\ spline\ approximating\ y = y(x),\ M_i = S''(x_i)$

   $$and\ S'(x_i) = S'(x_i^+) \cup S'(x_i^-)$$

6. $for\ i\ from\ 1\ to\ n - 1\ do$:

   $$M_{i-1} + 4 * M_i + M_{i+1} = \frac{6}{h^2}(y_{i-1} - 2 * y_i + y_{i+1})$$

7. $Solve\ the\ equations\ obtained\ from\ step\ 4\ and\ y\ to\ obtain\ the\ value\ of$ $M_0, M_1, M_2, \ldots\ldots M_n\ and\ y_1, y_2 \ldots\ldots y_{n-1}\ and\ display\ the\ solutions$
8. $Stop$

Program:

Input: Predefined differential equation and an interval in which we need to find value of y

Output: Approximated value of y using Cubic Spline Method

Code: Contains a class which has utility functions for doing the same

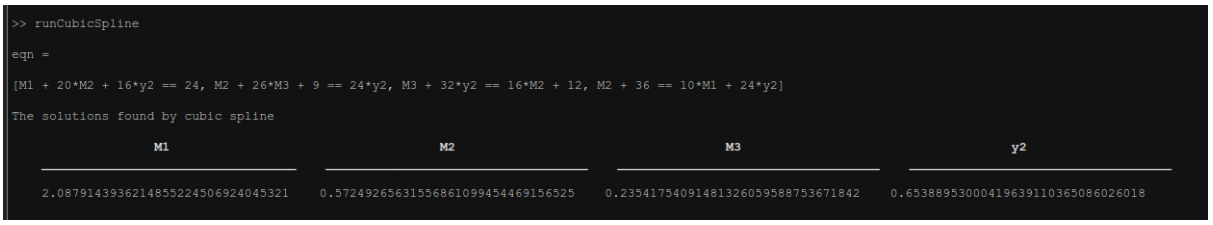
```
classdef CubicSpline
  properties (Access = private)
     x,y,h,n,output;
  end
  methods (Access=public)
     function this=CubicSpline()
        intervals=2;
%           intervals=3;
        x0=1;xn=2;
%           x0=0;xn=1;
        this.h = (xn-x0)/intervals;
        this.n=intervals+1;
        n=this.n;
        this.x = linspace(x0,xn,n);
        syms y [1 n]
        y(1)=1;y(n)=0.5;

        this.y=y;

     end
     function run(this)
        this=this.cubicSpline();
        fprintf("The solutions found by cubic spline\n\n")
        disp(this.output)
     end
  end
  methods (Access=private)
     function this=cubicSpline(this)
        n=this.n; %#ok<*PROP>
        y=this.y;
        x=this.x;
        syms M [1 n]
        syms dydx [n 2]
        eqn=[];
        for i=1:n
           dydx(i,1:2)=this.equationDyDx(i,M);
           eqn=[eqn,M(i)*x(i)^2+x(i)*dydx(i,1:2)-y(i)==0];

        end
        if(size(eqn)<2*n-2)
           for i=1:n-2
              eqn=[eqn,this.equationM(i+1,M)]; %#ok<*AGROW>
           end
        end
```

```
        eqn=unique(simplify(eqn));
        display(eqn)
        solutions=vpasolve(eqn,[M,y(2:n-1)]);
        this.output=struct2table(solutions);
    end
    function dydx=equationDyDx(this,i,M)
        dydx=[];
        n=this.n;
        y=this.y;
        h=this.h;
        if(i~=1)
            dydx=h/6*(2*M(i)+M(i-1))+(y(i)-y(i-1))/h;
        end
        if(i~=n)
            dydx=[dydx,-h/6*(2*M(i)+M(i+1))+(y(i+1)-y(i))/h];
        end
    end
    function equation=equationM(this,i,M)
        h=this.h; %#ok<*PROPLC>
        y=this.y;
        equation=simplify(M(i-1)+4*M(i)+M(i+1)==6*(y(i-1)-2*y(i)+y(i+1))/h^2);
    end
  end
end
```

Syntax to run:

```
cubicSpline=CubicSpline();
cubicSpline.run()
```

Sample Output:



```
>> runCubicSpline

eqn =

[M1 + 20*M2 + 16*y2 == 24, M2 + 26*M3 + 9 == 24*y2, M3 + 32*y2 == 16*M2 + 12, M2 + 36 == 10*M1 + 24*y2]

The solutions found by cubic spline

                M1                                 M2                                 M3                                 y2
    ___________________________________    ___________________________________    ___________________________________    ___________________________________

    2.0879143936214855224506924045321    0.57249265631556861099454469156525    0.23541754091481326059588753671842    0.65388953000419639110365086026018
```

4.Runge-Kutta-Fehlberg Method (RKF45)

The Runge-Kutta-Fehlberg method (RKF45) is a numerical method for solving ordinary differential equations (ODEs). It is an adaptive step-size method, which means that the step size is adjusted during the integration process to maintain a desired level of accuracy. The method is based on a fourth-order Runge-Kutta method (RK4) and a fifth-order Runge-Kutta method (RK5), both of which are used to estimate the solution at the next time step.

Algorithm:

Inputs:

- $x0,\ y0$
- $tolerance\ TOL$
- $step\ size\ h$

Steps:

1. $Start.$
2. $Set\ t\ =\ x0;$
   $w\ =\ y0.$
3. $Set\ K1\ =\ h\ *\ f(t,\ w);$
   $K2\ =\ h\ *\ f(t\ +\ \frac{h}{4},\ w\ +\ \frac{K1}{4});$
   $K3\ =\ h\ *\ f(t\ +\ \frac{3h}{8},\ w\ +\ \frac{3K1}{32}\ +\ \frac{9K2}{32});$
   $K4\ =\ h\ *\ f(t\ +\ \frac{12h}{13},\ w\ +\ \frac{1932K1}{2197}\ -\ \frac{7200K2}{2197}\ +\ \frac{7296K3}{2197});$
   $K5\ =\ h\ *\ f(t\ +\ h,\ w\ +\ \frac{439K1}{216}\ -\ 8\ *\ K2\ +\ \frac{3680K3}{513}\ -\ \frac{845K4}{4104});$
   $K6\ =\ h\ *\ f(t\ +\ \frac{h}{2},\ w\ -\ \frac{8K1}{27}\ +\ 2K2\ -\ \frac{3544K3}{2565}\ +\ \frac{1859K4}{4104}\ -\ \frac{11K5}{40}).$
4. $Set\ R\ =\ \frac{1}{h}\ *\ |\frac{K1}{360}\ -\ \frac{128K3}{4275}\ -\ \frac{2197K4}{75240}\ +\frac{K5}{50}\ +\frac{2K6}{55}|$
5. $Set\ \delta\ =\ 0.84(\frac{TOL}{R})¼.$
6. $if\ \delta\ >\ 1\ Approximation\ is\ not\ accepted,\ Stop.$
7. $else\ Set\ t\ =\ t\ +\ h;$
   $w\ =\ w\ +\ \frac{25K1}{216}\ +\ \frac{1408K3}{2565}\ +\ \frac{2197K4}{4104}\ -\ \frac{K5}{5};$
   $h\ =\ h\ *\ q.$
8. $Output\ (t,\ w,\ h).$
9. $Stop.$

Program:

Input: Predefined differential equation and a initial value provided at some point x
Output: Approximated value of y using RKF method with 4th ,5ht order method
Code: Contains a class which has utility functions for doing the same

```
classdef RangeKuttaFehlberg
  properties(Access=private)
```

```
    f,h,x,y,n,y1,q,tolerance
  end
  methods (Access=public)
    function this=RangeKuttaFehlberg()
      this.f=@(x,y) y-x^2+1;
%       this.f=@(x,y)1+y*y;
      this.h=0.25;
%       this.h=0.2;
      this.tolerance=10^-5;
      n=1;
      syms X [1 n+1]
      syms Y [1 n+1]
      X(1)=0;
%       Y(1)=0;
      Y(1)=0.5;
      this.x=X;
      this.y=Y;
      this.y1=Y;
      this.n=n;
    end
    function run(this)
      this=this.rangeKuttaFehlberg();
      q=this.q;
      fprintf("The value of q found is\n\n")
      disp(q)
      h=this.h*q;
      if(this.q<1)
        fprintf("Since q<1 we accept the value of y\n\n")
        fprintf("The value of y predicted through 4th order is\n\n")
        disp(this.y1(2))
        fprintf("The value of y predicted through 5th order is\n\n")
        disp(this.y(2))
        fprintf("The new value of h is\n\n")
        disp(h)
      else
        fprintf("Value of q > 1 we reject the value of y we reject the choice of h")
      end
    end
  end
  methods (Access=private)
    function this=rangeKuttaFehlberg(this)
      n=this.n; %#ok<*PROP>
      syms k [n 6]
      syms q [n,1]
```

```
        for i=1:n
            k(i,1:6)=this.findK(i);
            this=this.findY(i,k(i,1:6));
            q(i)=this.findQ(k(i,1:6));
        end
        this.q=q;
    end
    function k=findK(this,i)
        f=this.f; %#ok<*PROPLC>
        xi=this.x(i);
        yi=this.y(i);
        h=this.h;
        syms k [1 6]
        k(1) =h*f(xi,yi);
        k(2)=h*f(xi+h/4,yi+k(1)/4);
        k(3)=h*f(xi+3*h/8,yi+3*k(1)/32+9*k(2)/32);
        k(4)=h*f(xi+12*h/13,yi+1932/2197*k(1)-7200/2197*k(2)+7296/2197*k(3));
        k(5)=h*f(xi+h,yi+439/216*k(1)-8*k(2)+3680/513*k(3)-845/4104*k(4));
        k(6)=h*f(xi+h/2,yi-8/27*k(1)+2*k(2)-3544/2565*k(3)+1859/4104*k(4)-11/40*k(5));
        k=vpa(k);
    end
    function this=findY(this,i,k)
        xi=this.x(i);
        yi=this.y(i);
        y1i=this.y1(i);
        this.x(i+1)=xi+this.h;
        this.y1(i+1)=simplify(y1i+25/216*k(1)+1408/2565*k(3)+2197/4104*k(4)-k(5)/5);

this.y(i+1)=simplify(yi+16/135*k(1)+6656/12825*k(3)+28561/56430*k(4)-9/50*k(5)+2/55*
k(6));
    end
    function q=findQ(this,k)
        h=this.h;
        tolerance=this.tolerance;
        R=abs(k(1)/360-128*k(3)/4275-2197*k(4)/75240+k(5)/50+2*k(6)/55)/h;
        q=0.84*(tolerance/R).^(1/4);
    end
  end
end
```

Syntax to run:

```
rangeKuttaFehlberg=RangeKuttaFehlberg();
rangeKuttaFehlberg.run()
```

Sample Output:

```
The value of q found is

0.946208809139595967170931861115 89

Since q<1 we accept the value of y

The value of y predicted through 4th order is

0.92048860207582131410256410256 41

The value of y predicted through 5th order is

0.92048704929840870392628205128205

The new value of h is

0.23655220228489899179273296527897
```