MATLAB PROGRAMS FOR SOME NUMERICAL METHODS AND ALGORITHMS (I)

D.A.GISMALLA
Department of Mathematics & Computer Studies
Faculty of Science & Technology
Gezira University, Wadi medani,
Sudan

Abstract: In[1],We have listed and described some numerical methods and techniques for the reader so that he can be acquainted with them and then a description program flow chart is mentioned without the Matlab Software Program . Here , We list the Matlab Software Programs for each Interpolation Method mentioned in [1]. These Methods are

1. Taylor's Interpolation stored in M-file "Taylor.m"
2. Lagrange Polynomial stored in M-file " lagran.m"
3. Newton's Forward divided _difference formula stored in M-file " newton_table.m"
4. Newton' divided polynomial stored in M-file " newton_poly.m"
5. NEWTON'S divided_difference stored in M-file " newton_backdiagonal.m"
6. NEWTON'S forward difference stored in M-file " newtonforward.m"
7. NEWTON'S backward difference stored in M-file " newton_backward.m"

The reader must distinguishes between when the symbol the semicolon ; and the percentage % can be removed or not in the MATLAB statement . Each one has a different meaning to be carefully supper pressed or not , specially % when one COPIES and PASTES any M_file. Further , the name of each M_file for a particular described method that We have mentioned can be found within the FUNCTION name in the MATLAB PROGRAM . Hence , there is no need to mention or state the corresponding file name for each method or algorithm in the second paper (II) as this is similar for having the same title as above to complete the job. Furthermore , the Command Instructions Window must be typed and submitted carefully so that the execution of the corresponding M_file runs safely.

Key words: Interpolations, Taylors, Lagrange, Newton's Tables

1. INTRODUCTION

1.1 Taylor's Interpolation

```
% Taylor's Interpolation concentrates all
% information around one fixed given
% point x0. The Expansion Series with its
% error bound is given by
% Suppose a function f(x) and its first nth
% derivatives are continuous in a closed
% interval [a, b], and that f(n+1)(x) exists
% on this interval. Then there exists some
% c ∈(a, b) such that
% It's clear from the fact that n! grows
% rapidly as n increases , that is for
% sufficiently differentiable functions f(x)
% Taylor polynomials become more
% accurate as n increases.
% Example 2.1 Find the Taylor polynomials
% of orders 1, 3, 5, and 7 near x = 0
% for f(x) = sin x. (Even orders are omitted
% because Taylor polynomials for
% sin(x) have no even order terms.)
% The MATLAB command for a Taylor
% polynomial is taylor(f,n+1,a), where f is
% the function, a is the point around which
% the expansion is made, and n is the
% order of the polynomial.
% We can use the following code:
% >>syms x
% >>f=inline('sin(x)')
% >>taylor(f(x),2,0)
% ans = x
% >>taylor(f(x),4,0)
% ans = x-1/6*x^3
% >>taylor(f(x),6,0)
% ans = x-1/6*x^3+1/120*x^5
% >>taylor(f(x),8,0)
% ans = x-1/6*x^3+1/120*x^5-1/5040*x^7
```

1.2 Lagrange Polynomial

```
% Example 2.2(a) Given the tabulated values
% find Lagrange Polynomial and use it to
% approximate f(x)= 1/x at x=3
% In Matlab command window
% X = [2.00 2.5 4.00 ]
% Y = [0.50 0.40 0.25 ]
% [C L] = lagran(X,Y)
% To approximate 1/x at x = 3
% TYPE in the Command Window
% polyval(C,3)
function [C,L]=lagran(X,Y)
w=length(X);
n=w-1;
L=zeros(w,w);
% Form the Lagrange coefficient polynomials
for k=1:n+1
V=1;
for j=1:n+1
if k~=j
```

```
      V=conv(V,poly(X(j)))/(X(k)-X(j));
    end
  end
  L(k,:)=V;
 end
 %  Determine the coefficients of the Lagrange
 %  interpolator polynomial C=Y*L;
 %  Example 2.2(b) Given the  tabulated values find Lagrange
 %  Polynomial and  use it to  approximate f(1.5)
 %  In Matlab command window
 %   X = [1.00 1.3 1.6 1.9  2.2]
  %   Y = [0.7651977  0.6200860   0.4554022  0.2818186
0.1103623]
  %   [C L] = lagran(X,Y)
 %       polyval(C, 1.5)
```

1.3 Newton's Forward divided difference formula

1.3.1 Example 1.3.1(a)

```
%  Example 2.3(a)
%  In Matlab command window
%  newton_table_test.m
%  x = [1.00 1.3 1.6 1.9  2.2];
%  y = [0.7651977  0.6200860   0.4554022  0.2818186
0.1103623];
%  F = newton_table(x, y)

function [F] = newton_table(x,y)
% newton_table generate 1-based newton table
% x = [x0,x1,...,xn-1] is the vector of nodes
% y = [y0,y1,...,yn-1] is the vector of data values
% F = n by n matrix of divided differences.
%  Diagonal elements are the coefficients of newton polynomial

% x0   x(1)  F(1,1)
% x1   x(2)  F(2,1) F(2,2)
% x2   x(3)  F(3,1) F(3,2) F(3,3)
% x3   x(4)  F(4,1) F(4,2) F(4,3) F(4,4)
% ...  ...   ...    ...    ...    ...
% ...  ...   ...    ...    ...    ...
% xn-1 x(n)  F(n,1) F(n,2) F(n,3) F(n,4) ... ... F(n,n)

n = length(x);
F = zeros(n,n);

% Construct first column using data values in y

 for i = 1:n
   F(i,1) = y(i);
 end

 % Construct divided diffreence table
  % loop over column j
  for j = 2:n
 % loop down column j from the diagonal
    for i = j:n
    F(i,j) = (F(i,j-1) - F(i-1,j-1)) / (x(i) - x(i-j+1));
    end
  end
```

1.3.2 Example 1.3.2(b)

```
%  Example 1.3.2(b)
%  In Matlab command window
%  newton_table_test.m and
 %  newton_poly_test.m is
%  the Newton'divided polyn
%  The following data are for
 %  BESSEL'S function J0(x)
%   to approximate J0(1.5)
%  x = [1.00 1.3 1.6 1.9  2.2];
%  y = [0.7651977  0.6200860   0.4554022
              0.2818186 0.1103623];
%  x0 =1.5 ;
%  [F,A , Valpol] = newton_poly(x, y,x0)

function [F,A,Valpol] = newton_poly(x,y,x0)
% newton_table generate 1-based newton table
% x = [x0,x1,...,xn-1] is the vector of nodes
% y = [y0,y1,...,yn-1] is the vector of
 %  data values
% F = n by n matrix of divided differences.
% Diagonal elements are the coefficients
% of newton polynomial A = [a0,a1,...,an-1]
% is vector of polynomial coefficients
% Valpol is the value of the Newton's divided
% difference polynomail at x0

% x0   x(1)  F(1,1)
% x1   x(2)  F(2,1) F(2,2)
% x2   x(3)  F(3,1) F(3,2) F(3,3)
% x3   x(4)  F(4,1) F(4,2) F(4,3) F(4,4)
% ...  ...   ...    ...    ...    ...
% ...  ...   ...    ...    ...    ...
% xn-1 x(n)  F(n,1) F(n,2) F(n,3)  ... ... F(n,n)

n = length(x);
F = zeros(n,n);
A(1)= y(1);

% Construct first column using data values in y

 for i = 1:n
   F(i,1) = y(i);
 end

 % Construct divided diffreence table

  for j = 2:n % loop over column j
  % loop down column j from the diagonal
    for i = j:n
  F(i,j) = (F(i,j-1) - F(i-1,j-1)) / (x(i) - x(i-j+1));
    end
  %  A is Diagonal elements are the coefficients
  %  of newton polynomial
    A(j) = F(j,j);
  end
  % Valpol gives the value of NEWTON'S
  % divided_difference polynomail at
  % x=x0 using the  diagonal elements of A(j)
  % from top to bottom
  Valpol=A(1);
  for j=2:n
    v=1;
    for k=1:j-1
```

```
      v=v*(x0-x(k));
    end
  Valpol=Valpol + v*A(j);
 end
```

1.3.3 Example 1.3.3(c)

```
  %  Example 1.3.3(c)
%  In Matlab command window
%  newton_table_test.m and
 %  newton_poly_test.m
%   is the Newton'divided polyn
%  The following data are for
%   BESSEL'S function
%  J0(x) to approximate J0(1.5)
%  x = [1.00 1.3 1.6 1.9  2.2];
%  y = [0.7651977  0.6200860  0.4554022
       0.2818186 0.1103623];
%  x0 = 1.5;
%  [F , A , Valpol] =
 %  newton_backdiagonal(x, y, x0)

function [F,A,Valpol] =
       newton_backdiagonal(x,y,x0)
% newton_table generate 1-based newton table
% x = [x0,x1,...,xn-1] is the vector of nodes
% y = [y0,y1,...,yn-1] is the
 %  vector of data values
% F = n by n matrix of divided differences.
% Diagonal elements are the coefficients of
% newton polynomial  A = [a0,a1,...,an-1]
% is vector of polynomial coefficients
% Valpol is the value of the Newton's divided
% difference polynomail at x0

% x0   x(1)  F(1,1)
% x1   x(2)  F(2,1) F(2,2)
% x2   x(3)  F(3,1) F(3,2) F(3,3)
% x3   x(4)  F(4,1) F(4,2) F(4,3) F(4,4)
% ...  ...   ...    ...    ...    ...
% ...  ...   ...    ...    ...    ...
% xn-1 x(n)  F(n,1) F(n,2) F(n,3) ... ... F(n,n)

n = length(x);
F = zeros(n,n);
A(1)= y(n);

% Construct first column using data values in y

 for i = 1:n
   F(i,1) = y(i);
 end

 % Construct divided difference table
 % loop over column j
  for j = 2:n
 % loop down column j from the diagonal
  for i = j:n
   F(i,j) = (F(i,j-1) - F(i-1,j-1)) / (x(i) - x(i-j+1));
  end
  %  A is Diagonal elements are the coefficients
  % of  newton polynomial
   A(j) = F(n ,j);
  end
  % Valpol gives the value of
  %  NEWTON'S divided
  %  difference polynomial at x=x0  using the
  %  diagonal elements of A(j) from
  %   bottom to top
   Valpol=A(1);
   for j=2:n
    v=1;
     for k=1:j-1
       v=v*(x0-x(n-k+1));
     end
    Valpol=Valpol + v*A(j);
   End
```

1.3.4 Example 1.3.4(d)

```
%  Example 1.3.4(d)
%  In Matlab command window
%  newton_table_test.m and
%  newton_poly_test.m
%  is the Newton'divided polyn
%  The following data are for
%  BESSEL'S function J0(x)
%  to approximate J0(1.1)
%  x = [1.00 1.3 1.6 1.9  2.2];
%  y = [0.7651977  0.6200860   0.4554022
     0.2818186 0.1103623];
%  x0 = 1.1 ;
%  h= 0.3 ;
%  [F,A ,Valpol] = newtonforward(x, y,x0,h)

function [F,A,Valpol] =     newtonforward(x,y,x0 ,h)
 % newton_table generate 1-based Newton table
 % x = [x0,x1,...,xn-1] is the vector of nodes
 % y = [y0,y1,...,yn-1] is the
  %  vector of data values
 % F = n by n matrix of divided differences.
 % Diagonal elements are the coefficients of
 % Newton polynomial  A = [a0,a1,...,an-1] is
 % vector of polynomial coefficients
 % Valpol is the value of the Newton's forward
 %  difference polynomial at x0

 % x0   x(1)  F(1,1)
 % x1   x(2)  F(2,1) F(2,2)
 % x2   x(3)  F(3,1) F(3,2) F(3,3)
 % x3   x(4)  F(4,1) F(4,2) F(4,3) F(4,4)
 % ...  ...   ...    ...    ...    ...
 % ...  ...   ...    ...    ...    ...
 % xn-1 x(n)  F(n,1) F(n,2) F(n,3) ... ... F(n,n)

 n = length(x);
 F = zeros(n,n);
 A(1)= y(1);

 % Construct first column using data values in y

  for i = 1:n
    F(i,1) = y(i);
  end

  % Construct divided difference table
   for j = 2:n % loop over column j
% loop down column j from the diagonal
```

```
  for i = j:n
 F(i,j) = (F(i,j-1) - F(i-1,j-1)) / (x(i) - x(i-j+1));
  end
 %  A is Diagonal elements are the coefficients
 %  of newton polynomial
   A(j) = F(j,j);
 end
 % Valpol gives the value of NEWTON'S
 % divided_difference polynomail at
 % x=x0 using the  diagonal elements of A(j)
 % from top to bottom calculate s
  s=(x0-x(1))/h ;
 Valpol=A(1);
 for j=2:n
    v=1;
    for k=1:j-1
        v=v*(s-k+1)*h;
    end
  Valpol=Valpol + v*A(j);
 End
```

1.3.5 Example 1.3.5(e)

```
%  Example 1.3.5(e)
%  In Matlab command window
%  newton_table_test.m
 % and newton_poly_test.m is
%  the Newton'divided polyn
%  The following data are for
%   BESSEL'S function
%  J0(x) to approximate J0(1.5)
%  x = [1.00 1.3 1.6 1.9  2.2];
%  y = [0.7651977  0.6200860   0.4554022
                        0.2818186 0.1103623];
%  x0 = 2.0;
%  h =  0.3 ;
%  [F , A , Valpol] =
 %    newton_backward(x, y, x0 ,h)

function [F,A,Valpol] =
                    newton_backward(x,y,x0,h)
% newton_table generate 1-based newton table
% x = [x0,x1,...,xn-1] is the vector of nodes
% y = [y0,y1,...,yn-1] is the
%  vector of data values
% F = n by n matrix of divided differences.
% Diagonal elements are the coefficients
% of Newton polynomial A = [a0,a1,...,an-1]
% is vector of polynomial coefficients
% Valpol is the value of the Newton's divided
%  difference polynomial at x0

% x0   x(1)  F(1,1)
% x1   x(2)  F(2,1) F(2,2)
% x2   x(3)  F(3,1) F(3,2) F(3,3)
% x3   x(4)  F(4,1) F(4,2) F(4,3) F(4,4)
% ...  ...   ...    ...    ...    ...
% ...  ...   ...    ...    ...    ...
% xn-1 x(n)  F(n,1) F(n,2) F(n,3)  ... ... F(n,n)
n = length(x);
F = zeros(n,n);
A(1)= y(n);
% Construct first column using data values in y
 for i = 1:n
    F(i,1) = y(i);
 end

 % Construct divided difference table

  for j = 2:n % loop over column j
 % loop down column j from the diagonal
 for i = j:n
  F(i,j) = (F(i,j-1) - F(i-1,j-1)) / (x(i) - x(i-j+1));
   end
  %  A is Diagonal elements are the coefficients
  %  of newton polynomial
   A(j) = F(n ,j);
  end
  % Valpol gives the value of NEWTON'S
  %  forward_difference polynomail
  % at x=x0 using the  diagonal elements of A(j)
  % from bottom to top
    s=(x0-x(n))/h ;
   Valpol=A(1);
   for j=2:n
     v=1;
      for k=1:j-1
        v=v*(s+k-1)*h;
      end
    Valpol=Valpol + v*A(j);
   end
```

REFERENCES

[1] International Journal of Algorithms, Computing and Mathematics, Volume 5 , number 1 pp.58-68 , Feb. 2012 India www.eashwarpublications.com

[2] Richard L. Burden, J. Douglas Faires, Numerical Analysis Prindle ,Weber & Schmidt

[3] Birkhoff,G and G. Rota(1978)Ordinary Differential Equation John Wiley & Sons, New York; 342 PP.QA372,B58

[4] www.swarthmore.edu/NatSci/echeeve1/Ref/NumericInt/RK2.html

