Numerical Analysis
Exercises

Christian Rossi

Academic Year 2023-2024

Abstract

The topics of the course are:

- Floating-point arithmetic: different sources of the computational error; absolute vs relative errors; the floating point representation of real numbers; the round-off unit; the machine epsilon; floating-point operations; over- and under-flow; numerical cancellation.

- Numerical approximation of nonlinear equations: the bisection and the Newton methods; the fixed-point iteration; convergence analysis (global and local results); order of convergence; stopping criteria and corresponding reliability; generalization to the system of nonlinear equations (hints).

- Numerical approximation of systems of linear equations: direct methods (Gaussian elimination method; LU and Cholesky factorizations; pivoting; sparse systems: Thomas algorithm for tridiagonal systems); iterative methods (the stationary and the dynamic Richardson scheme; Jacobi, Gauss-Seidel, gradient, conjugate gradient methods (hints); choice of the preconditioner; stopping criteria and corresponding reliability); accuracy and stability of the approximation; the condition number of a matrix; over- and under-determined systems: the singular value decomposition (hints).

- Numerical approximation of functions and data: Polynomial interpolation (Lagrange form); piecewise interpolation; cubic interpolating splines; least-squares approximation of clouds of data.

- Numerical approximation of derivatives: finite difference schemes of the first and second order; the undetermined coefficient method.

- Numerical approximation of definite integrals: simple and composite formulas; midpoint, trapezoidal, Cavalieri-Simpson quadrature rules; Gaussian formulas; degree of exactness and order of accuracy of a quadrature rule.

- Numerical approximation of ODEs: the Cauchy problem; one-step methods (forward and backward Euler and Crank-Nicolson schemes); consistency, stability, and convergence (hints).

Contents

Chapter 1

Introduction to MATLAB

1.1 Main MATLAB operators

Assignment operator:

```matlab
% Print output
a = 1
% Does not print output
b = 2;
```

The active variables can be found in the workspace and the value can be checked on the command window with:

```matlab
% Value of all variables
whos
% Value of a
whos a
```

If you want to save the file:

```matlab
% Save the command history
diary file_name.txt
% Save the whole workspace
save file_name
% Save only the variable a
save file_name_only_a a
% Load only the variable a
load file_name_only_a
% Load the whole workspace
load file_name
```

It is possible to clear variables with the following commands:

```matlab
% Clear only the variable a
clear a
% Clear the whole workspace
clear all
```

1.2   Vector and matrices

Most of the entities in MATLAB are matrices, even real numbers. The matrices can be defined in the following ways:

```matlab
% Row vector definition
c = [1 2 3]
% Column vector definition
c = [1; 2; 3]
% Vectorn transposition
c = [1 2 3]'
% 2D matrix definition
D = [ 1 2 3;
      4 5 6;
      7 8 9 ]
```

It is also possible to define various types of matrices:

```matlab
% Zeros vector/matrix
A = zeros(row_length,column_length)
% Ones vector/matrix
A = ones(row_length,column_length)
% Identity matrix
A = eye(row_length,column_length)
% Diagonal matrix
d = [1:4]
D = diag(d)
% Set a not principal diagonal
D = diag(d, diagonal_index)
% Select only upper o lower trinagular
Ml = tril(M)
Mu = triu(M)
% Access an element in vector
C(1)
% Access an element in matrix
```

```matlab
C([2,3]);
% Access a part of the matrix
Q(rows,columns)
% Access the element in position (n,m)
Q(end, end)
% Dimension of a matrix
length(a);
numel(b);
size(a);
```

The operations on vectors are done in the following way:

```matlab
% Given two row vectors a and b
% Vector sum
a + b
% Vector difference
a - b
% Scalar product
a * b'
dot(a,b)
% Tensor product
a' * b
% Elementwise product
a .* b
% Elementwise division
a ./ b
% Elementwise exponentiation
a .^ 2
```

The operations on matrices are done in the following way:

```matlab
% Givcen two matrices A and B (both 3x2)
% Matrix sum
A + B
% Matrix difference
A - B
% Matrix product
K * L'
% Elementwise product
A .* B
% Elementwise division
A ./ B
% Elementwise exponentiation
A .^ 2
```

```
% Power matrix (useful only square)
A ^ 2
% Other useful values of the matrices
% Determinant
det(A)
% Trace
trace(A)
% Inverse of small matrix
inv(A)
% Given a column vector b ths olutio of Ax=b
A \ b
```

The function used to plot a graph are the following:

```
% To plot y=f(x) in [a,b]
x = a:step_length:b;
y = f(x);
figure
plot(x,y,color)
% To add y2=f2(x) in [c,d]
hold on
x2 = c:step_length:d;
y2 = f2(x);
plot(x2,y2,color)
% Show graph's grid
grid on
% Set the axis limit
axis([xmin xmax ymin ymax])
% Set the same scaling for both axis
axis equal
```

To handle functions the commands are:

```
% Define a function handle to g(x)
f = @g(x);
% Evaluation of f in a
f(a)
% Define an anonymous function
% It is useful to modify other functions
f = @(argument-list) expression
```

The operators that u logical values are:

```
% Smaller than
```

```
a < b
% Greater than
a > b
% Smaller or equal than
a <= b
% Equal to
a == b
% Different from
a ~= b
% And
(a < b) & (b > c)
% Or
(a < b) | (b > c)
```

The control-flow statement are:

```
% if-then-else statements
if (condition1)
    block1
elseif (condition2)
    block2
else
    block3
end
% for loops
for (index=start:step:end)
    instruction block
end
% while loops
while (condition)
    instruction block
end
```

There are two categories of m-files:

- Scripts: these files contain instructions that are executed in sequence in the command line if the script file is called. The variables are saved in the current workspace.

- Functions: they take some input arguments and return some outputs after a series of instructions are performed. The variables defined in the function are local to the scope of the function itself.

Chapter 2

Laboratory I

Exercise 1

Define the row vector:

$$\bar{v}_k = [1, 9, 25, \dots, (2k+1)^2] \in \mathbb{R}$$

with $k = 8$ using the following strategies:

1. A for loop to define one by one each element of the vector.

2. The vector syntax to build it in just one shot.

Answer of exercise 1

```
k = 8;
% For loop strategy
vk = zeros(1, k+1);
for (ii = 0:k)
    vk(ii+1) = (2*ii + 1)^2;
end
vk
% Vector syntax
vk = [1:2:2*k+1].^2;
```

Exercise 2

Define a function which, for an input value k, returns the corresponding vector v_k as defined in the previous exercise.

Answer of exercise 2

```
function vk = ex_1_2(k)
vk = [1:2:2*k+1].^2;
end
```

Exercise 3

Using the function of the previous exercise write another function that returns, for a generic value k, the $2(k+1) \times 2(k+1)$ matrix.

$$
m_k = \begin{bmatrix}
2 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\
0 & \sqrt[2]{2} & 0 & 0 & 0 & 0 & \cdots & 0 & 1 \\
0 & 0 & \sqrt[3]{2} & 0 & 0 & 0 & \cdots & 0 & 0 \\
0 & 0 & 0 & \sqrt[4]{2} & 0 & 0 & \cdots & 0 & 9 \\
0 & 0 & 0 & 0 & \sqrt[5]{2} & 0 & \cdots & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \sqrt[6]{2} & \cdots & 0 & 25 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & 0 & 0 & 0 & \cdots & \sqrt[(2k+1)]{2} & 0 \\
1 & 1 & 9 & 9 & 25 & 25 & \cdots & (2k+1)^2 & (2k+1)^2
\end{bmatrix}
$$

Answer of exercise 3

```
function Mk = ex_1_3(k)
% Create a diagonal matrix with the right elements and dimension
Mk = diag(2.^(1./[1:2*(k+1)]))
% The bottom right element will be overwritten
% Last column
Mk(2:2:end, end) = ex_1_2(k)
% Last line
Mk(end, 1:2:end) = ex_1_2(k)
Mk(end, 2:2:end) = ex_1_2(k)
end
```

Exercise 4

Find the machine epsilon by implementing an ad hoc procedure. Comment and justify the obtained results.

Answer of exercise 4

```
k = 0;
EPS = 1/2;
while (1 + EPS) > 1
    EPS_old = EPS;   % keep track of the value
    EPS = EPS / 2;
    k = k + 1;
end
format long
EPS_old
(1 + EPS_old) > 1
EPS
(1 + EPS) > 1
k        % Number of iterations, which is also the numer of digits
    in the mantissa, according to the standard
eps
```

Exercise 5

Consider the following function:

$$f(x) = \frac{e^x - 1}{x}$$

1. Evaluate $f(x)$ for values of x around zero (try with $x_k = 10^{-k}$, $k \in [1, 20]$). What do you obtain? Explain the results.

2. Propose an approach for fixing the problem. (Hint: Use Taylor expansions to get an approximation of $f(x)$ around $x = 0$).

3. How many terms in the Taylor expansion are needed to get double precision accuracy (16 decimal digits) $\forall x \in \left[0, \frac{1}{2}\right]$?

Answer of exercise 5

```
%  Evaluate f(x) for values of x around zero (try with xk =
   10^{-k}, k in [1,20]). What do you obtain? Explain the results.
k = [1:20]';
x = 10.^(-k);
f = @(x) (exp(x) - 1) ./ x;
format long
[k x f(x)]
figure;
plot(x, f(x), '*')
f_taylor_5 = @(x) 1 + 1/2*x + 1/6*x.^2 + 1/24*x.^3 + 1/120*x.^4;
[k x f(x) f_taylor_5(x)]
format short e
n = [1:20]';
err = 1./factorial(n+2) .* (0.5).^(n+1).*exp(0.5);
[n err]
%  Therefore n* = 13$.
```

Exercise 6

The sequence:
$$1, \frac{1}{3}, \frac{1}{9}, \ldots, \frac{1}{3^n}, \ldots$$

can be generated with the following recursive relations:

$$\begin{cases} p_n = \dfrac{10}{3}p_{n-1} = p_{n-2} \\ p_1 = \dfrac{1}{3}, \; p_0 = 1 \end{cases}$$

$$\begin{cases} q_n = \dfrac{1}{3}q_{n-1} \\ q_0 = 1 \end{cases}$$

1. Implement the two relations in order to generate the first 100 terms of the sequence.

2. Study the stability of the two algorithms and justify the obtained results.

Answer of exercise 6

```
p(1) = 1;
p(2) = 1/3;
for i = 2:100
    p(i+1) = 10/3*p(i) - p(i-1);
end
figure
subplot(2,1,1)
plot(0:100, p, 'LineWidth',3)
% gca return the current axes
% setting the fontsize on axes
set(gca,'FontSize',16)
xlabel('n','FontSize',16)
ylabel('p_n','FontSize',16)
% The sequence explodes!

q(1) = 1;
for i=1:100
    q(i+1) = 1/3*q(i);
end
subplot(2,1,2)
plot(0:100, q, 'LineWidth',3)
```

```
set(gca,'FontSize',16)
xlabel('n','FontSize',16)
ylabel('q_n','FontSize',16)
% The sequence is ok.
```

Chapter 3

Laboratory session II

Exercise 7

Consider the following function

$$f(x) = x^3 - (2 + e)x^2 + (2e + 1)x + (1 - e) - \cosh(x - 1) \ \ x \in [0.5, 5.5]$$

- Plot the function f and determine two intervals that contain its roots.

- Implement the bisection method:

```
function [x,x_iter]=bisection(f,a,b,tol)
```

where x is the solution, x_iter is the vector of the approximations at each iteration, f is the function, defined as handle function, a, b are the end points of the interval, tol is the required tolerance.

- For which roots the bisection method can be used? Compute the number of needed iterations for the bisection method to converge with a tolerance of 10^{-3}, when the interval $[3, 5]$ is chosen as starting interval.

Answer of exercise 7

```
1.  f= @(x) x.^3-(2 + exp(1))*x.^2 + (2*exp(1) + 1)*x + (1 -
    exp(1)) - cosh(x - 1);
    a=0.5;
    b=5.5;
    x_plot=linspace(a,b,1000);
    plot(x_plot,f(x_plot));
    grid on
```

2.
```
function [x,x_iter]=bisection(f,a,b,tol)
    Nmax = ceil(log((b-a)/tol)/log(2));
    for i=1:Nmax
        x_iter(i)=(b+a)/2;
        if f(x_iter(i))*f(a)<0
            b=x_iter(i);
        else
            a=x_iter(i);
        end
    end
    x=x_iter(end);
end
```
3.
```
a=3;
b=5;
tol=1.e-3;
[x,x_iter]=bisection(f,a,b,tol);
x
x_plot=linspace(a,b,1000);
figure
plot(x_plot,f(x_plot));
title("Iterations required to reach the tollerance:",
    length(x_iter))
grid on
hold on
```

Exercise 8

Consider the following function in the interval $[-0.5, 1.5]$

$$f(x) = sin(x)(1-x)^2$$

1. Plot f in order to find some intervals containing the roots.

2. Implement the Newton method by using a stopping criterion based on the error estimator $\left| x^k - x^{k-1} \right|$. The signature of the function is:

```
function [x,x_iter]=newton(f,df,x0,tol,Nmax)
```

    where x is the approximate, x_iter is the vector of the approximations at each iteration, f, df are the function and its first derivative, defined as handle functions, $x0$ is the initial guess, tol is the tolerance demanded by user and $Nmax$ is the maximum number of allowed iterations.

3. Use Newton method to find the roots with a tolerance equal to 10^{-6}, by considering as initial guess $x0 = 0.3$ and $x0 = 0.5$.

4. Compute an estimate of the convergence rate.

Answer of exercise 8

```
1.  f=@(x) sin(x).*(1-x).^2;
    df=@(x) cos(x).*(1-x).^2-2*sin(x).*(1-x);
    x_plot=linspace(-0.5,1.5,1000);
    plot(x_plot,f(x_plot));
    grid on
2.  function [x,x_iter]=newton(f,df,x0,tol,Nmax)
        i=1;
        err=1+tol;
        x_iter(i)=x0;
        while i<=Nmax && err>tol
            if(abs(df(x_iter(i))) < 1e-8)
                break;
            end
            x_iter(i+1)=x_iter(i)-f(x_iter(i))/df(x_iter(i));
            err=abs(x_iter(i+1)-x_iter(i));
            i=i+1;
        end
        x=x_iter(end);
    end
3.  [x1,x1_iter]=newton(f,df,0.3,1.e-6,100);
```

```matlab
    err1=abs(x1_iter-0)
4.  function p = conv_order(e) % e is the error
        p = log(e(3:end)./e(2:end-1))./log(e(2:end-1)./e(1:end-2));
    end
    p = conv_order(err1)
    figure;
    plot(p);
    title('convergence order, root x1=0');
    [x2,x2_iter]=newton(f,df,0.5,1.e-6,100);
    err2=abs(x2_iter-1);
    p = conv_order(err2)
    figure;
    plot(p);
    title('convergence order, root x2=1');
```

Exercise 9

1. Implement the Newton method for systems. The signature of the function is:

```
function
    [x,res,niter,difv,x_vect]=newtonsys(Ffun,Jfun,x0,tol,
    kmax,normtype,varargin)
```

---

where $Ffun$ is the function handle to vector function f(x), $Jfun$ is Jacobian handle matrix, $x0$ is the initial value for iterative process, $nmax$ is the maximum number of allowed iterations, tol is the absolute error tolerance, $normtype$ is the type of norm used in the error estimation, and $varargin$ is an input variable in a function definition statement that enables the function to accept any number of input arguments.

2. Use Newton method for the given system and the given parameters.

Answer of exercise 9

```
1.  function [x,res,niter,difv,x_vect] =
    newtonsys(Ffun,Jfun,x0,tol, kmax,normtype,varargin)
    k = 0;
    x_vect = x0;
    err = tol + 1; difv=[ ];
    x = x0;
    if normtype == 2
        nor = 2;
    else
        nor = inf;
    end
    while err >= tol && k < kmax
        J = Jfun(x,varargin{:});
        F = Ffun(x,varargin{:});
        delta = - J\F;
        x = x + delta;
        err = norm(delta,nor);
        difv=[difv; err];
        x_vect = [x_vect x];
        k = k + 1;
    end
    res = norm(Ffun(x,varargin{:}));
    if (k==kmax && err> tol)
```

```matlab
        fprintf(['The method does not converge'],F);
    end
    niter=k;
2.  F = @(x) [x(1).^2+x(2).^2-1; sin(pi*x(1)/2)+x(2).^3];
    J = @(x) [2*x(1), 2*x(2);  cos(pi*x(1)/2)*pi/2, 3*x(2).^2];
    nmax = 200;
    tol = 1e-10;
    x0 = [-1;-1];
    [x,res,niter,difv,x_vect] = newtonsys(F,J,x0,tol,nmax,2);
    format long
    x
    niter
```

Chapter 4

Laboratory session III

Exercise 10

Given $\xi = \sqrt{5}$, using the fixed point iteration method:

1. Check if $\phi(x) = 5 + x - x^2$ converges in ξ.

2. Check if $\phi(x) = 5/x$ converges in ξ.

3. Check if $\phi(x) = 1 + x - (1/5) * x^2$ converges in ξ.

4. Check if $\phi(x) = (1/2) * (x + 5/x)$ converges in ξ and the order of convergence.

5. Apply the fixed point iteration method in all previous cases.

Answer of exercise 10

```
1.  xi = sqrt(5);
    phi1 = @(x) 5 + x - x.^2;
    dphi1 = @(x) 1 - 2*x;
    abs(dphi1(xi))
    % The absolute value of the derivative of phi at xi
    % is greater than 1: the method will not converge.
2.  xi = sqrt(5);
    phi2 = @(x) 5./x;
    dphi2 = @(x) -5./x.^2;
    abs(dphi2(xi))
    % The absolute value of the derivative of phi at xi
    % is equal to 1: no theoretical conclusion can be stated in
       this case.
3.  xi = sqrt(5);
```

```matlab
    phi3 = @(x) 1 + x - 1/5*x.^2;
    dphi3 = @(x) 1 - 2/5*x;
    abs(dphi3(xi))
    % The absolute value of the derivative of phi at xi
    % is less than 1: the method will converge provided that the
        initial guess x{(0)} is close enough to xi (local
        convergence).
4.  xi = sqrt(5);
    phi4 = @(x) 1/2*(x + 5./x);
    dphi4 = @(x) 1/2 - 5./(2*x.^2);
    abs(dphi4(xi))
    % The absolute value of the derivative of phi at xi
    % is zero (i.e. less than 1): the method will converge provided
        that the initial guess x{(0)} is close enough to xi (local
        convergence).
    d2phi4 = @(x) 5./x.^3;
    abs(d2phi4(xi))
    % The second derivative is different from zero,
    % so method 4 is expected to be of second order.
5.  function [xi, x_iter] = fixed_point(phi, x0, tol, maxit)
        x_iter(1) = x0;
        for (iter = 1:maxit)
        x_iter(iter+1) = phi(x_iter(iter));
        if (abs (x_iter(iter+1) - x_iter(iter)) < tol)
            break;
        end
        end
        xi = x_iter(end);
    end

    tol = 1e-6;
    maxit = 1000;
    x0 = xi + 0.001;
    [xi1, x1] = fixed_point(phi1, x0, tol, maxit);
    xi1
    iter1 = numel(x1) - 1
    [xi1, x1] = fixed_point_FV(phi1, x0, tol, maxit);
    xi1
    iter1 = numel(x1) - 1
    % The approximation xi is incorrect and the number of performed
        iterations is the maximum: as expected, the method did not
        converge.
```

```
x0 = 3;
[xi2, x2] = fixed_point(phi2, x0, tol, maxit);
xi2
% number of iterations of method 2
iter2 = numel(x2) - 1
% different implementation of fixed point method
[xi2, x2] = fixed_point_FV(phi2, x0, tol, maxit);
xi2
iter2 = numel(x2) - 1
% The approximation xi is incorrect and the number of performed
    iterations is the maximum: the method did not converge.

x0 = 4;
[xi3, x3] = fixed_point(phi3, x0, tol, maxit);
xi3
iter3 = numel(x3) - 1
[xi3, x3] = fixed_point_FV(phi3, x0, tol, maxit);
xi3
iter3 = numel(x3) - 1
% The method converged locally to xi.

x0 = 10;
[xi3, x3] = fixed_point(phi3, x0, tol, maxit);
xi3
iter3 = numel(x3) - 1
[xi3, x3] = fixed_point_FV(phi3, x0, tol, maxit);
xi3
iter3 = numel(x3) - 1
% With a different initial guess, the method may not converge
    to xi.

x0 = 4;
[xi4, x4] = fixed_point(phi4, x0, tol, maxit);
xi4
iter4 = numel(x4) - 1
[xi4, x4] = fixed_point_FV(phi4, x0, tol, maxit);
xi4
iter4 = numel(x4) - 1
% The method converged to xi.
```

Exercise 11

Consider the following function in the interval $[-1, 6]$

$$f(x) = \arctan\left[7\left(x - \frac{\pi}{2}\right)\right] + \sin\left[\left(x - \frac{\pi}{2}\right)^3\right]$$

1. Plot f in order to find an interval containing a root. What is the multiplicity of the root?

2. Use the Newton method to find the root with a tolerance of 10^{-10} and initial guess $x^{(0)} = 1.5$. Compute the error.

3. Use the Newton method to find the root with a tolerance of 10^{-10} and initial guess $x^{(0)} = 4$. Compute the error.

4. If possible, apply the bisection method on the interval $[a, b] = [-1, 6]$ and tolerance $\dfrac{b - a}{20^{30}}$. Compute the error.

5. Write a function $bisection\_newton.m$ to find ξ using the Newton method starting from an initial guess obtained after few iterations of a bisection method. Test with $[a, b] = [-1, 6]$, 5 iterations of the bisection method and tolerance 10^{-10} for the Newton method.

Answer of exercise 11

```
1. function rootfinding_function_plot(f, a, b, new_figure)
       if ((nargin < 4) || new_figure)
       figure
       end
       hold on, box on
       x_plot = linspace(a, b, 1000);
       plot(x_plot, f(x_plot), 'LineWidth', 2)
       plot(x_plot, 0*x_plot, 'k-', 'LineWidth', 1)
       xlabel('x','FontSize', 16)
       ylabel('f(x)','FontSize', 16)
       set(gca,'FontSize', 16)
       set(gca,'LineWidth', 1.5)
   end

   a = -1;
   b = 6;
   f = @(x) atan(7*( x - pi/2)) + sin((x-pi/2).^3);
   rootfinding_function_plot(f, a, b, true);
```

```matlab
    xi_ex = pi/2;
    df = @(x) 7 ./ ( 1 + 49 * ( x-pi/2 ).^2 ) + 3 * (x-pi/2).^2 .*
        cos( (x-pi/2).^3 );
    df(xi_ex);
2.  x0 = 1.5;
    tol = 1e-10;
    maxit = 1000;
    [xi1, x_iter1] = newton(f, df, x0, tol, maxit);
    xi1
    iter1 = numel(x_iter1) - 1
    err1 = abs( xi1 - xi_ex)
    % Newton method converges to xi
3.  x0 = 4;
    tol = 1e-10;
    maxit = 1000;
    [xi2, x_iter2] = newton(f, df, x0, tol, maxit);
    xi2
    iter2 = numel(x_iter2) - 1
    err2 = abs( xi2 - xi_ex)
    % Newton method does not converge to xi
4.  tol = (b-a)/(2^30);
    [xi3, x_iter3] = bisection(f, a, b, tol);
    xi3
    iter3=numel(x_iter3)
    tol_bisection = (b-a)/(2^5);
    tol_newton = 1e-10;
    maxit_newton = 1000;
    [xi4, x_iter4_bisection, x_iter4_newton] = bisection_newton(f,
        df, a, b, tol_bisection, tol_newton, maxit_newton);
    xi4
    iter4_bisection=numel(x_iter4_bisection)
    iter4_newton=numel(x_iter4_newton)
    err4 = abs( xi4 - xi_ex)
5.  function [xi, x_iter_bisection, x_iter_newton] =
    bisection_newton(f, df, a, b, tol_bisection, tol_newton,
    maxit_newton, multiplicity)
        if (nargin < 8)
        multiplicity = 1;
        end
        [xi_bisection, x_iter_bisection] = bisection(f, a, b,
            tol_bisection);
        [xi_newton, x_iter_newton] = newton(f, df, xi_bisection,
            tol_newton, maxit_newton, multiplicity);
```

```matlab
        xi = xi_newton;
    end

function [xi, x_iter] = bisection(f, a, b, tol)
    max_iterb = ceil( log((b - a)/tol)/log(2) );

    for (iter = 1:max_iterb)
    x_iter(iter) = a + (b-a)/2;
    f_iter(iter) = f(x_iter(iter));

    if ( f(b)*f_iter(iter) < 0 )
        a = x_iter(iter);
    elseif ( f(a)*f_iter(iter) < 0 )
        b = x_iter(iter);
    else % f(x) = 0
        break;
    end
    end

    xi = x_iter(end);
end

function [xi, x_iter] = newton(f, df, x0, tol, maxit, ...
    multiplicity)
    if (nargin < 6)
    multiplicity = 1;
    end
    x_iter(1) = x0;
    for (iter = 1:maxit)
    newton_method = @(x) x - multiplicity*f(x)/df(x);
    x_iter(iter+1) = newton_method(x_iter(iter));

    if (abs (x_iter(iter+1) - x_iter(iter)) < tol)
        break;
    end
    end
    xi = x_iter(end);
end
```

Chapter 5

Laboratory session IV