3. Numerical analysis I

1. Root finding: Bisection method
2. Root finding: Newton-Raphson method
3. Interpolation
4. Curve fitting: Least square method
5. Curve fitting in MATLAB
6. Summary

Text

A. Gilat, MATLAB: An Introduction with Applications, 4th ed., Wiley
3.1. Root finding: Bisection method

- Formulation of the problem
- Idea of the bisection method
- MATLAB code of the bisection method
- Root finding with build-in MATLAB function fzero

Reading assignment

Gilat 7.9, 9.1
http://en.wikipedia.org/wiki/Bisection_method
3.1. Root finding: Bisection method

Problem statement

We need to find real roots $x_*$ of an equation

$$f(x_*) = 0 \quad (3.1.1)$$

in the interval $a < x < b$, where $f(x)$ is the continuous function.

Root of Eq. (3.1.1) is the (real) number that turns this equation into identity.

In general, a non-linear equation can have arbitrary number of roots in a fixed interval $(a, b)$.

Examples:

- Linear equation
  $$px_* = q, \quad f(x) = px - q. \quad \text{Only one root } x_* = q/p.$$

- Quadratic equation
  $$px_*^2 + qx_* + r = 0, \quad f(x) = px^2 + qx + r. \quad \text{Can have 0, 1, or 2 real roots.}$$

- Transcendental equation
  $$\sin x_* = a, \quad f(x) = \sin x - a. \quad \text{Multiple roots, Can not be solved algebraically.}$$
3.1. Root finding: Bisection method

Example: Roots finding in thermo-physical calculations

- The temperature dependence of the material properties is given by empirical equations. The specific heat $C$ (J/kg/K) as a function of temperature $T$ (K) of some material:
  \[ C(T) = C_0 + C_1 T + C_2 T^2 + C_3 T^3 \]

Then the specific internal (thermal) energy $u$ (J/kg) at temperature $T$ is
  \[ u(T) = \int_0^T C(T')dT = C_0 T + \frac{C_1}{2} T^2 + \frac{C_2}{3} T^3 + \frac{C_3}{4} T^4 \]

Let's assume that

1. We consider some body of that material of mass $M$ (kg) with initial temperature $T_1$. Then the thermal energy of that body is equal to
   \[ U_1 = M u(T_1) \]

2. We heat the body by a laser and add energy $\Delta Q$ (J).

3. What is the body temperature $T_2$ after heating?
   In order to answer this question we must find a root $T_2$ of the equation:
   \[ M u(T_2) = M u(T_1) + \Delta Q \]

\[ T_2 = ? \]
3.1. Root finding: Bisection method

**Algebraic** solution \( x_* \) is:

- An equation (formula) that defines the root of the equation \( f(x_*) = 0 \).
- An accurate solution.

**Numerical** solution \( x_*^{(num)} \):

- A **numerical** value which turns equation \( f(x_*) = 0 \) into identity.
- An approximate solution. It means that \( f(x_*^{(num)}) \neq 0 \), but \(|f(x_*^{(num)})|\) is small.

The numerical methods for root finding of non-linear equations usually use iterations for successive approach to the root:

We find \( x_*^{(1)}, x_*^{(2)}, x_*^{(3)}, \ldots \) such that \( x_*^{(i)} \to x_* \), i.e. \( \varepsilon_i = |x_*^{(i)} - x_*| \to 0 \).

After finite number of iterations, we will be able to find the root with finite numerical error \( \varepsilon_i \).
3.1. Root finding: Bisection method

**Bisection method**

- Let's assume that we localize a single root in an interval \((a, b)\) and \(f(x)\) changes sign in the root. If the interval \((a, b)\) contains one root of the equation, then \(f(a)f(b) < 0\).
- Let's iteratively shorten the interval by bissections until the root will be localized in the sufficiently short interval. For every bisection at the central point \(c = (a + b)/2\), we replace either \(a\) or \(b\) by \(c\) providing \(f(a)f(b) < 0\) after the replacement.

One **iteration** of the bisection method:

1. Assume the root is localized in the interval \(a_i < x < b_i\).
2. Calculate middle point \(c_i = (a_i + b_i)/2\). This is the \(i^{th}\) approximation to the root \(x_{*(i+1)} = c_i\).
3. If \(b_i - a_i < \varepsilon\), then stop iterations. The root is found with tolerance \(\varepsilon\).
4. If \(f(a_i)f(c) < 0\) then \(a_{i+1} = a_i\), \(b_{i+1} = c\) or \(a_{i+1} = c\), \(b_{i+1} = b_i\) otherwise.
3.1. Root finding: Bisection method

MATLAB code for the bisection method

Example: Solving equation $\sin x = 1/2$.

```matlab
function [ x, N ] = Bisection ( a, b, Tol )
    N = 0;
    fa = Equation ( a ) ;
    while b – a > Tol
        c = 0.5 * ( a + b ) ;
        fc = Equation ( c ) ;
        if fa * fc > 0
            a = c;
        else
            b = c ;
        end
        N = N + 1 ;
    end
    x = c ;
end

function f = Equation ( x )
    f = sin ( x ) – 0.5;
end
```

Notes:

1. Calculation of $f(x)$ is the most computationally "expensive" part of the algorithm. It is important to calculate $f(x)$ only once per pass of the loop.

2. **Advantage** of the bisection method: If we are able to localize a single root, the method allows us to find the root of an equation with any continuous $f(x)$ that changes its sign in the root. No any other restrictions applied.

3. **Disadvantage** of the bisection method: It is a slow method. Finding the root with small tolerance $\varepsilon$ requires a large number $N$ of bisections. Example: Let's assume $\Delta x = b - a = 1$, $\varepsilon = 10^{-8}$. Then the $N$ can be found from equation $\varepsilon = \Delta x / 2^N$:

$$
N = \frac{\log(\Delta x / \varepsilon)}{\log 2} = \frac{\log 10^8}{\log 2} \approx 27.
$$
3.1. Root finding: Bisection method

Summary on root finding with build-in MATLAB function fzero

The MATLAB build-in function `fzero` allows one to find a root of a nonlinear equation:

\[ x = \text{fzero}( \text{@fun, x0} ) \]

Example:

\[
\sin(x) = \frac{1}{2} \quad \Rightarrow \quad f(x) = \sin x - \frac{1}{2} = 0
\]

```
function [f] = fun(x)
    f = sin(x) - 0.5;
end
```

\[ x = \text{fzero}( \text{@fun, 0.01} ) \]
3.1. Root finding: Bisection method

- The MATLAB build-in function `fzero` allows one to find a root of a nonlinear equation:
  - ✓ `x = fzero ( @fun, x0 ).`
  - ✓ `fun` is the (user-defined) function that calculates the LHS `f(x)` of the equation.
  - ✓ `x0` can be either a single real value or a vector of two values.
- If `x0` is a single real number, then it is used as the initial approximation to the root. In this case the `fzero` function automatically finds another boundary of the interval `x1` such that `f(x1) * f(x0) < 0` and then iteratively shrinks that interval.
- If `x0` is a vector of two numbers, then `x0(1)` and `x0(2)` are used as the boundaries of the interval, where the root is localized, such that `f( x0(1) ) * f ( x0(2) ) < 0`.
- The function works only if `f(x)` changes its sign in the root (not applicable for `f(x) = x^2`).
- The function utilizes a complex algorithm based on a combination of the bisection, secant, and inverse quadratic interpolation methods.

- **Example**: Roots of equation `sin(x) = 0.5`

  ```matlab
def function [ f ] = SinEq ( x )
    f = sin ( x ) - 0.5 ;
end

x = fzero ( @SinEq, [ 0, pi / 2 ] )
x = fzero ( @SinEq, 0.01 )
```
3.2. Root finding: Newton-Raphson method

- Idea of Newton-Raphson method: Linearization
- Graphical form of the root finding with Newton-Raphson method
- Examples: When Newton-Raphson method does not work
- MATLAB code for Newton-Raphson method
- MATLAB function function

Reading assignment

http://en.wikipedia.org/wiki/Newton's_method
Gilat 7.9, 9.1
3.2. Root finding: Newton-Raphson method

**Problem statement**
We need to find a real root \( x_* \) of a non-linear equation

\[
(3.2.1) \quad f(x_*) = 0
\]

in an \( a < x < b \) interval, where \( f(x) \) is the differentiable function with continuous derivative \( f'(x) \).

**Newton-Raphson method**

- In the framework of Newton-Raphson (Newton's) method we start calculations from some initial approximation for the root, \( x_*(1) \), and then iteratively increase the accuracy of this approximation, i.e. successively calculate \( x_*(2), x_*(3), \ldots \) such that \( x_*(i) \to x_* \) and \( \varepsilon_i = |x_*(i) - x_*| \to 0 \).

- In order to find the next approximation to the root, \( x_*(i) \), based on the previous approximation, \( x_*(i-1) \), we use the idea of linearization: For one iteration, we replace non-linear Eq. (3.2.1) by a linear equation that is as close to Eq. (3.2.1) as possible.

All other functions in this example are not differentiable if \( (a, b) \) includes point \( P \).
3.2. Root finding: Newton-Raphson method

- Linearization is based on the **Taylor series**. The Taylor series is the approximation of \( f(x) \) in a vicinity of point \( x = a \) by a polynomial:

\[
f(x) = f(a) + f'(a)(x - a) + \frac{1}{2} f''(a)(x - a)^2 + \cdots
\]

- Let's apply the Taylor series in order to find \( x^*_i \) based on \( x^*_i-1 \), i.e. represent \( f(x) \) in Eq. (3.2.1) in the form of the Taylor series at \( x = x^*_i \) and \( a = x^*_i-1 \)

\[
f(x^*_i) = 0
\]

\[
f(x^*_i-1) + f'(x^*_i-1)(x^*_i - x^*_i-1) + \frac{1}{2} f''(x^*_i-1)(x^*_i - x^*_i-1)^2 + \cdots = 0
\]

- then drop all non-linear terms

\[
f(x^*_i-1) + f'(x^*_i-1)(x^*_i - x^*_i-1) = 0 \quad (3.2.2)
\]

- and use this equation to find the next approximation to the root:

\[
x^*_i = x^*_i-1 - \frac{f(x^*_i-1)}{f'(x^*_i-1)} \quad (3.2.3)
\]
3.2. Root finding: Newton-Raphson method

Graphical representation of the Newton-Raphson method

- The plot of the function \( y = f(x_{*i-1}) + f'(x_{*i-1})(x - x_{*i-1}) \) is the straight line that is tangent to the plot of the function \( f(x) \) in the point \( x_{*i-1} \).
- When we find the root of Eq. (3.2.2), we find a point, where the tangent crosses the axis \( Ox \).
- The iterative process of Newton-Raphson method can be graphically represented as follows:

\[
y = f(x_{*(1)}) + f'(x_{*(1)})(x - x_{*(1)})
\]

- **Advantages** of Newton-Raphson method:
  - It is the fast method. Usually only a few iterations are required to obtain the root.
  - It can be generalized for systems of non-linear equations.
3.2. Root finding: Newton-Raphson method

- **Disadvantage** of the Newton-Raphson method: There are lot of situations, when the method does not work. Conditions that guarantee the convergence of \( x_{*}(1), x_{*}(2), \ldots \) to \( x_{*} \), i.e. \( |x_{*}(i) - x_{*}| \to 0 \), are complicated. Roughly, the Newton-Raphson method converges if

- In some interval around the root \( x_{*} \), \( f(x) \) has the first and second derivatives (first derivative is continuous), \( f'(x) \neq 0 \), \( f''(x) \) is finite.

**Example**: \( f(x) = \sqrt[3]{x} \) is the function that does not satisfy these properties and the root of equation \( \sqrt[3]{x} = 0 \) can not be find with the Newton-Raphson method.

- Initial approximation, \( x_{*}(1) \), is chosen to be "sufficiently close" to the root \( x_{*} \).

**Examples**: Newton-Raphson method does not work when the initial point is too "far" from the root or enters a cycle.
3.2. Root finding: Newton-Raphson method

MATLAB code for Newton-Raphson method

Example: Solving equation \( \sin x = \frac{1}{2} \).

```matlab
function [ x, N ] = NewtonMethod ( a, Tol )
    N = 0 ;
    x = a ;
    [ f, dfdx ] = Equation ( x ) ;
    while abs ( f ) > Tol
        x = x - f / dfdx ;
        [ f, dfdx ] = Equation ( x ) ;
        N = N + 1 ;
    end
end

function [ f, dfdx ] = Equation ( x )
    f = sin ( x ) - 0.5 ;
    dfdx = cos ( x ) ;
end
```

Notes:

1. Calculation of \( f(x) \) and \( f'(x) \) is the most computationally "expensive" part of the algorithm. It is important to calculate \( f(x) \) and \( f'(x) \) only once per pass of the loop.

2. Disadvantage of the current version of the code: For solving different equations we need to prepare different versions of the NewtonMethod function. They will be different only by the name of the function (Equation) that calculates \( f(x) \) and \( f'(x) \).

We can make NewtonMethod universal (capable of solving different equations) by programming the MATLAB `function function`.

- Only 3 iterations is necessary to get the root with tolerance \( \epsilon = 10^{-8} \).
3.2. Root finding: Newton-Raphson method

MATLAB function function

- **Function function** is a function that accepts the name of another function as an input argument.
- Definition of the function function:
  
  ```matlab
  function [ ... ] = Function1 ( Fun, .... ) : Here Fun the name of input function argument
  ```
- Use of the function function:
  
  ```matlab
  [ ... ] = Function1 ( @Fun1, ... ) : Here Fun1 is the name of a MATLAB function
  ```

MATLAB code for the Newton-Raphson method based on function function

**File NewtonMethodFF.m**

```matlab
function [ x, N ] = NewtonMethodFF ( Eq, a, Tol )
    N = 0;
    x = a;
    [ f, dfdx ] = Eq ( x );
    while abs ( f ) > Tol
        x = x - f / dfdx ;
        [ f, dfdx ] = Eq ( x );
        N = N + 1;
    end
end
```

**File SinEq.m**

```matlab
function [ f, dfdx ] = SinEq ( x )
    f = sin ( x ) - 0.5 ;
    dfdx = cos ( x );
end
```

In the MATLAB command window:

```matlab
[ x, N ] = NewtonMethodFF ( @SinEq, 0.01, 1e-08 )
```
3.3. Interpolation

- Interpolation problem
- Reduction of the interpolation problem to the solution of a SLE
- Polynomial interpolation
- Example: Interpolations of smooth and non-smooth data

Reading assignment
3.3. Interpolation

Interpolation problem

Let's assume that a functional dependence between two variables \( x \) and \( y \) is given in the tabulated form: We know values of the function, \( y_i = y(x_i) \), for some discrete values of the argument \( x_i, i = 1, ..., N \).

\[
\begin{array}{cccccccccc}
\text{Arg.} & x_1 & x_2 & \ldots & x_{i-1} & x_i & x_{i+1} & \ldots & x_{N-1} & x_N \\
\text{Fun.} & y_1 & y_2 & \ldots & y_{i-1} & y_i & y_{i+1} & \ldots & y_{N-1} & y_N \\
\end{array}
\]  

(3.3.1)

Such tabulated data can be produced in experiments. Example: \( x = t \) is time and \( y = T \) is temperature, in the experiment we measure the temperature \( T_i \) at a discrete times \( t_i \).

We are interested in the question: How can we predict the values of the function \( y(x) \) (and its derivatives \( y'(x) \), etc.) at arbitrary \( x \) which does not coincide with any of \( x_i \)?

There are two major of approaches to introduce \( y(x) \) based on tabulated data in the form (3.3.1). We will consider two major methods:

1. **Interpolation**.

2. **Fitting** (will be considered later).

Interpolation implies that we introduce a continuous interpolation function \( f(x) \) such that

\[
f(x_i) = y_i, \quad i = 1, ..., N.
\]  

(3.3.2)

This means that the interpolation function goes through every point \((x_i, y_i)\) on the plane \((x, y)\).
3.3. Interpolation

- We assume that all $x_i$ are given in ascending order: $x_{i-1} < x_i$
- **Interpolation** is the process of constructing new data points within the observation interval:
  $$x_1 \leq x \leq x_N: \quad y = f(x) \text{ is the interpolated value of the function}$$
- **Extrapolation** is the process of constructing new data points beyond the observation interval:
  $$x < x_1 \text{ or } x > x_N: \quad y = f(x) \text{ is the extrapolated value of the function}$$
- Both interpolation and extrapolation can be performed only approximately, but extrapolation is subject to greater uncertainty and higher risk of producing meaningless results.
3.3. Interpolation

Solution of the interpolation problem

Let's introduce a system of \( N \) known functions

\[
f_1(x), \ f_2(x), \ f_3(x), \ f_4(x), \ \ldots
\]

Usually these functions are assumed to be smooth (have continuous derivatives of any order).

Now, let's look for the interpolation function in the following form:

\[
f(x) = C_1f_1(x) + C_2f_2(x) + \cdots + C_Nf_N(x) = \sum_{i=1}^{N} C_if_i(x) \tag{3.3.3}
\]

where \( C_i \) are unknown coefficients. In order to be an interpolation function, \( f(x) \) should satisfy conditions (3.3.2), i.e.

\[
\begin{align*}
C_1f_1(x_1) + C_2f_2(x_1) + \cdots + C_Nf_N(x_1) &= y_1 \\
C_1f_1(x_2) + C_2f_2(x_2) + \cdots + C_Nf_N(x_2) &= y_2 \\
&\ \vdots \\
C_1f_1(x_N) + C_2f_2(x_N) + \cdots + C_Nf_N(x_N) &= y_N \tag{3.3.4}
\end{align*}
\]

Eqs. (3.3.4) is the linear system of \( N \) equations with respect to \( N \) coefficients \( C_i \). It can be rewritten in the matrix form as follows:

\[
\begin{bmatrix}
f_1(x_1) & \cdots & f_N(x_1) \\
\vdots & \ddots & \vdots \\
f_1(x_N) & \cdots & f_N(x_N)
\end{bmatrix}
\begin{bmatrix}
C_1 \\
\vdots \\
C_N
\end{bmatrix}
= 
\begin{bmatrix}
y_1 \\
\vdots \\
y_N
\end{bmatrix} \tag{3.3.5}
\]

Thus solution of the interpolation problem reduces to solution of a SLE.
3.3. Interpolation

The interpolation function in the form
\[ f(x) = C_1 x^{N-1} + \cdots + C_{N-2} x^2 + C_{N-1} x + C_N \] (3.3.6)
is called the interpolation polynomial.

- In order to find the interpolation polynomial one needs to solve the SLE given by Eqs. (3.3.5):

\[ f_1(x) = x^{N-1}, \quad f_2(x) = x^{N-2}, \quad \ldots, \quad f_{N-1}(x) = x, \quad f_N(x) = 1. \] (3.3.7)

\[
\begin{bmatrix}
    x_1^{N-1} & \cdots & 1 \\
    \vdots & \ddots & \vdots \\
    x_N^{N-1} & \cdots & 1
\end{bmatrix}
\begin{bmatrix}
    C_1 \\
    \vdots \\
    C_N
\end{bmatrix}
=
\begin{bmatrix}
    y_1 \\
    \vdots \\
    y_N
\end{bmatrix}.
\] (3.3.8)

Elements of the matrix of coefficients \( A \) are equal to \( a_{ij} = x_i^{N-j} \)

- If the interpolation data includes \( N \) points \( (x_i, y_i) \), then we can find the interpolation polynomial of degree \( N - 1 \).

- The chosen order of functions in Eqs. (3.3.7) and (3.3.8) \( (C_1 \) is the coefficient at the highest degree of \( x \)) allows us to use the MATLAB \texttt{polyval} function in order to calculate value of the interpolation polynomial.
### 3.3. Interpolation: General approach

#### Problem 3.3.1: Interpolation of various functions

**File InterpolationProblem.m**

```matlab
function [ C ] = InterpolationProblem ( x_i, y_i )
    N = length ( x_i );
    A = zeros ( N, N );
    for i = 1 : N % i is the row index
        for j = 1 : N % j is the column index
            A(i,j) = x_i(i)^(N-j);
        end
    end
    C = inv ( A ) * y_i';
end
```

**File Interpolation.m**

```matlab
function [ C ] = Interpolation ( Fun, a, b, N, NN )
    x_i = linspace ( a, b, N );
    y_i = arrayfun ( Fun, x_i );
    C = InterpolationProblem ( x_i, y_i );
    x = linspace ( a, b, NN );
    f = polyval ( C, x ); % Interpolation polynomial
    y = arrayfun ( Fun, x ); % Original function
    plot ( x, y, 'r', x_i, y_i, 'bx', x, f, 'g' )
end
```

**File Problem_3_3_1**

```matlab
C = Interpolation ( @TriangleFun, -1, 3, 5, 101 )
```

These functions can be used to generate data points:

**File PolyFun.m**

```matlab
function [ y ] = PolyFun ( x )
    Coeff = [ 1 2 3 ];
    y = polyval ( Coeff, x );
end
```

**File SinFun.m**

```matlab
function [ y ] = SinFun ( x )
    y = sin ( pi * x / 2 );
end
```

**File TriangleFun.m**

```matlab
function [ y ] = TriangleFun ( x )
    if x < 0
        y = 0 ;
    elseif x < 1
        y = x ;
    elseif x < 2
        y = 2 - x;
    else
        y = 0; 
    end
end
```
3.3. Interpolation

**Example 1:** Smooth data \( y = \sin(\pi x/2) \), \( N \) is the number of interpolation points

A. Symmetric interpolation interval (-1,1)

B. Non-symmetric interpolation interval (0.5, 1)

- In general, it is difficult to build and calculate interpolation polynomials at large \( N (>10-20) \) due to strong enhancement of **round-off errors.** We are limited by small \( N! \)
3.3. Interpolation

Example 2: Non-smooth data in the form of a triangle pulse

Symmetric interpolation interval (-1,3)

- For non-smooth data, an increase in the number of data points $N$ (and degree of the polynomial) can deteriorate the accuracy.

- The values of the interpolation polynomial for non-smooth data are subject to "oscillations."
3.4. Curve fitting: Least square method

- Fitting problem
- When is interpolation not a viable approach?
- Least square method: General approach
- Least square method: Polynomial fitting

Reading assignment
Gilat 8.2, 8.4, 8.5
3.4. Curve fitting: Least square method

Curve fitting is the process of constructing a curve, or mathematical function, that has the best fit to a series of discrete data points.

Curve fitting implies that

1. We choose a form of the fitting function (e.g. linear fitting function \( f(x, C_1, C_2) = C_1 + C_2x \)) with some number of unknown coefficients \((C_1, C_2)\). In general, the choice of the fitting function is *arbitrary* and the number of unknown coefficients is *much smaller* than the number \( N \) of data points.
2. We introduce a *measure of difference*, \( R \), between the data points \((x_i, y_i)\) and the fitting function \( f(x) \).
3. We find such unknown coefficients \((C_1, C_2)\) that allow us to *minimize* the value of \( R \).
Curve fitting is an *alternative to interpolation*.

**Difference between interpolation and fitting functions:**

- **Interpolation function** passes precisely through every data point. Fitting function goes closely to data points and follows the general trend in data behavior.

- **Interpolation function** has $N$ coefficients, where $N$ is the number of data points. Fitting function has $M$ coefficients, usually $M \ll N$. 

**Fitting function**

\[ f(x, C_1, C_2) = C_1 + C_2 x \]
3.4. Curve fitting: Least square method

- Curve fitting can (and must!) be used instead of interpolation if
  - There are too many data points in order to build an interpolating function \((N > \sim 10)\).
  - Input data are noisy.
  - We are interested in revealing general trends in the data behavior (Curve fitting can be used as a tool for data analysis).

Example: Fitting vs. interpolation of noisy data (see solution in FittingVsInterpolation.m)
Data points are given by the law \(y(x) = 1 + 2x + \text{random noise}\)
3.4. Curve fitting: Least square method

**Least square method:** General idea

Least square method for finding coefficients of fitting functions is based on the general conditions that allow one to find a minimum of a function

Minimum of a function \( f(x) \)

\[
\frac{\partial f}{\partial x}(x_{\text{min}}) = 0
\]

In the least square method, the same conditions of a minimum are applied to the mean-square difference \( R \) between the fitting function and tabulated data.

Example: Fitting function with two coefficients:

\[
R(C_1, C_2) = \frac{1}{N} \sum_{i=1}^{N} (f(x_i, C_1, C_2) - y_i)^2
\]

Conditions of minimum of \( R(C_1, C_2) \):

\[
\frac{\partial R}{\partial C_1} = 0, \quad \frac{\partial R}{\partial C_2} = 0
\]

These are two equations with respect to \( C_1 \) and \( C_2 \).
3.4. Curve fitting: Least square method

Least square method: Polynomial fitting

- Assume that we have $N$ data points $(x_k, y_k), k = 1, \ldots, N$.
- Consider the fitting function in the form of a polynomial of degree $M$ ($M \ll N$)

$$f(x) = f(x, C_1, C_2, \ldots, C_M) = C_1 x^{M-1} + \ldots + C_{M-2} x^2 + C_{M-1} x + C_M = \sum_{j=1}^{M} C_j x^{M-j} \quad (3.4.1)$$

- Introduce the mean square difference $R$

$$R(C_1, C_2, \ldots, C_M) = \frac{1}{N} \sum_{k=1}^{N} (f(x_k) - y_k)^2$$

- Apply conditions of a minimum of $R(C_1, C_2, \ldots, C_M)$:

$$\frac{\partial R}{\partial C_i} = 0 \quad \Rightarrow \quad \frac{2}{N} \sum_{k=1}^{N} (f(x_k) - y_k) \frac{\partial f}{\partial C_i}(x_k) = 0, \quad i = 1, \ldots, M \quad (3.4.2)$$

Eq. (3.4.1) $\Rightarrow$ $\frac{\partial f}{\partial C_i}(x_k) = x_k^{M-i} \Rightarrow \sum_{k=1}^{N} x_k^{M-i} f(x_k) = \sum_{k=1}^{N} y_k x_k^{M-i}, \quad (3.4.2)$

$$\sum_{k=1}^{N} x_k^{M-i} \sum_{j=1}^{M} C_j x_k^{M-j} = \sum_{k=1}^{N} y_k x_k^{M-i},$$
3.5. Curve fitting: Least square method

\[ \sum_{j=1}^{M} \left( \sum_{k=1}^{N} x_k^{M-j} x_k^{M-i} \right) C_j = \sum_{k=1}^{N} y_k x_k^{M-i} \]

\[ \sum_{j=1}^{M} a_{ij} C_j = b_i, \quad i = 1, \ldots, M \quad \Rightarrow \quad \begin{bmatrix} a_{11} & \cdots & a_{1M} \\ \vdots & \ddots & \vdots \\ a_{M1} & \cdots & a_{MM} \end{bmatrix} \begin{bmatrix} C_1 \\ \vdots \\ C_M \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_M \end{bmatrix} \tag{3.4.3} \]

Eq. (3.4.3) is the SLE with respect to coefficients \( C_1, C_2, \ldots, C_M \), where the matrix of coefficients and the RHS are

\[ a_{ij} = \sum_{k=1}^{N} x_k^{2M-(i+j)}, \quad b_i = \sum_{k=1}^{N} y_k x_k^{M-i}. \tag{3.4.4} \]

Solution of the polynomial fitting problem reduces to a SLE given by Eq. (3.4.3) with respect to unknown coefficients \( C_i \) \((i = 1, \ldots, M)\) of the fitting polynomial.

Once coefficients are found, values of the fitting polynomial can be calculated with the MATLAB \texttt{polyval} function.
3.5. Curve fitting in MATLAB

- Polynomial curve fitting with the MATLAB build-in functions
- Other fitting functions
- Data analysis based on the curve fitting
- MATLAB basic fitting interface

**Reading assignment**

Gilat 8.2, 8.4, 8.5
3.5. Curve fitting in MATLAB

Least square method: Polynomial fitting in the MATLAB

- Assume that we have \( N \) data points \((x_k, y_k), k = 1, \ldots, N\).
- Consider the fitting polynomial of degree \( K = M - 1 \) \((M \ll N)\)

\[
f(x) = f(x, C_1, C_2, \ldots, C_M) = C_1 x^{M-1} + C_{M-2} x^2 + \ldots + C_{M-1} x + C_M
\]

- In the MATLAB, coefficients of the fitting polynomial, \( C_1, C_2, \ldots, C_M \), can be calculated with the build-in `polyfit` function.
- This function implements the solution of the SLE given by Eq. (3.4.3).
- Syntax:

\[
C = \text{polyfit} (x_i, y_i, K)
\]

- \( x_i \) \((= [1:N])\) is a 1D array of x-coordinates of the data points
- \( y_i \) \((= [1:N])\) is a 1D array of y-coordinates of the data points
- \( K = M-1 \) is the degree of the fitting polynomial
- \( C = [C[1], C[2], \ldots, C[M]] \) is an array of \( M \) coefficients of the fitting polynomial
- \( f = \text{polyval} (C, x) \) can be used in order to calculate the value of the fitting polynomial
3.5. Curve fitting in MATLAB

Problem 3.5.1: Fitting of polynomial data. Initial data points are obtained with the polynomial \( y(x) = (x - 0.1)(x - 0.4)(x - 0.75)(x - 0.8)(x - 0.9) \) in the interval \([0,1]\) at \( N = 10 \).

File Fitting.m

```matlab
function [ C ] = Fitting ( Fun, a, b, N, K, NN )
    % Preparation of tabulated data
    x_i = linspace ( a, b, N );
    y_i = arrayfun ( Fun, x_i );
    % Solving the fitting problem
    C = polyfit ( x_i, y_i, K );
    % Now we plot the function, fitting polynomial, and data points
    x = linspace ( a, b, NN );
    f = polyval ( C, x );  % Fitting polynomial
    y = arrayfun ( Fun, x );  % Original function
    plot ( x, y, 'r', x_i, y_i, 'bx', x, f, 'g' )
end
```

File PolyFun.m

```matlab
function [ y ] = PolyFun ( x )
    y = ( x - 0.1 ) .* ( x - 0.4 ) .* ( x - 0.75 ) .* ( x - 0.8 ) .* ( x - 0.9 );
end
```

File Problem_3_5_1.m

```matlab
C = Fitting ( @PolyFun, 0.0, 1.0, 10, 3, 101 )
```
3.5. Curve fitting in MATLAB

Solution of problem 3.5.1:

Fitting polynomial

Fitting, interpolation, and original polynomials of degree 5 coincide with each other here, \( N > K = 5 \)
3.5. Curve fitting in MATLAB

Curve fitting with functions other than polynomials

- Theoretically, any function can be used to model data within some *short range* of $x$.
- For a given problem, some particular function provide a better fit than others (better fit in a *broader range* of $x$).
- The choice of the fitting function for the experimental data points is often based on preliminary theoretical consideration of scaling laws governing the dependence $y = y(x)$.
- Curve fitting with power, exponential, logarithmic, and reciprocal functions are of particular importance since these functions often occur in science and engineering.

\[
\begin{align*}
  f &= bx^m & \text{: Power function} \\
  f &= be^{mx} & \text{: Exponential function} \\
  f &= b + m \log x & \text{: Logarithmic function} \\
  f &= 1/(b + mx) & \text{: Reciprocal function}
\end{align*}
\]

- Fitting with these functions can be reduced to fitting with a polynomial of the first degree and, thus, can be performed with the *polyfit* function.
3.5. Curve fitting in MATLAB

For this purpose, one needs to rewrite any of such functions in a form that can be fitted with a linear polynomial:

\[ \log f = \log b + m \log x \quad : \text{Power function} \quad : \text{Linear relation between } \ln x \text{ and } \ln f \]
\[ \log f = \ln b + mx \quad : \text{Exponential function} \quad : \text{Linear relation between } x \text{ and } \ln f \]
\[ f = b + m \log x \quad : \text{Logarithmic function} \quad : \text{Linear relation between } \ln x \text{ and } f \]
\[ 1/f = b + mx \quad : \text{Reciprocal function} \quad : \text{Linear relation between } x \text{ and } 1/f \]

Then the fitting problem can be solved in three steps:

1. Transform tabulated data \((x_i, y_i)\) to \((t_i, z_i)\) such that for the chosen fitting function relationship between \(t\) and \(z\) is linear.

   **Example**: For the power fit, \(t_i = \log x_i, z_i = \log y_i\).

2. Apply \(C = \text{polyfit}(t, z, 1)\) to \((t_i, z_i)\) and obtain the coefficients \(C_2\) and \(C_1\) in the linear fitting function for transformed data

   \[ z = C_1 t + C_2 \]
   \[ \log y = m \log x + \log b \]

3. Obtain coefficients \(m\) and \(b\) in the original fitting function

   **Example**: For the power fit, \(m = C_1, b = \exp C_2\).
3.5. Curve fitting in MATLAB

**Problem 3.5.2:** Fitting data with the power function: Data points are given by the equation \( y = (1 + 2\sqrt{x})x^2 \) in the interval \([0.1, 10]\) at \( N = 10 \).

The part of the code that prepares coefficients of the fitting functions:

**File PowerFitting.m**

```matlab
function [ m1, b1 ] = PowerFitting ( Fun, a, b, N, NN )

% Preparation of tabulated data
x_i = linspace ( a, b, N );
y_i = arrayfun ( Fun, x_i );

% Solving the fitting problem
C = polyfit ( log ( x_i ), log ( y_i ), 1 );

% Coefficients of the power fitting function
m1 = C(1);  
b1 = exp ( C(2) );

% Now we plot the function, power fitting function, and
x = linspace ( a, b, NN );
f = b1 * x.^m1;  

% Original function
y = arrayfun ( Fun, x );

% Plotting
loglog ( x, y, 'r', x_i, y_i, 'bx', x, f, 'g' )

end

**File QuasiPowerFun.m**

```matlab
function [ y ] = QuasiPowerFun ( x )

y = ( 1 + 2.0 * sqrt ( x ) ) * x^2;

end

**File Problem_3_5_2.m**

```matlab
[m1, b1] = PowerFitting ( @QuasiPowerFun, 0.1, 10.0, 10, 101 )
```

---

*Results in the double logarithmic scale*

**Power fitting function**

\( f(x) = bx^m \)

**Fitting polynomial of degree 3 (\( K = 3 \))**
3.5. Curve fitting in MATLAB

Data analysis based on the curve fitting

- "Basic" fitting functions (linear, power, exponential, logarithmic, and reciprocal) are specific for many engineering problems since many fundamental physical laws are described in terms of these functions.
- We can visually judge about the best shape of the fitting function by plotting data in different scales (normal, semi logarithmic, double logarithmic) or by plotting reciprocal ($1/y$) data points.
- If the data points follow to one of the "basic" functions (or scaling laws), there will be a scale type when the data points fall on a line plot.
- Four data sets are plotted using different plot scales.
  - $f = bx^m$
  - $f = be^{mx}$
  - $f = b + m \ln x$
  - $f = 1/(b + mx)$
  - $\ln f = \ln b + m \ln x$
  - $\ln f = \ln b + mx$
  - $f = b + m \ln x$
  - $1/f = b + mx$
- See DataAnalysis.m
3.5. Curve fitting in MATLAB

**Basic fitting tool of the MATLAB figure window**

- Fitting functions can be added to the MATLAB figure window by using the **Basic fitting tool**.
- For this purpose we need to do only two steps:
  - Plot data points using `plot`, `semilogx`, `semilogy`, or `loglog` commands.
  - In the opened figure window, go to menu Tools->Basic Fitting
- The Basic fitting panel allows us to
  - Add fitting functions to the figure window.
  - See coefficients of fitting functions.
  - Plot residuals.
- Similar interactive fitting tools are build in MS Excel and other data processing software.
3.6. Summary

For the exam we must know how

- To implement and use the bisection method for finding roots of a non-linear equation.
- To implement and use the Newton-Raphson method for finding roots of a non-linear equation.
- To use the build-in *fzero* function for finding roots of an individual non-linear equation.
- To find coefficients of an interpolation polynomial by solving a SLE.
- To understand the basic idea of the least square method and how to reduce the fitting problem to the solution of a SLE.
- To use *polyfit* function in order to find coefficients of the fitting polynomial.
- To use *polyfit* function to fit data to power, exponential, logarithmic, and reciprocal functions.
- To chose the best shape of the fitting function by changing plot scales.