4. Numerical analysis II

1. Numerical differentiation
2. Numerical integration
3. Ordinary differential equations (ODEs). First-order ODEs
4. Numerical methods for initial value problems (individual ODEs)
5. Second- and higher order ODEs. Systems of ODEs
6. Numerical methods for initial value problems (systems of ODEs)
7. Numerical solution of IVPs with build-in MATLAB solvers
8. Summary

Text

4.1. Numerical differentiation

- Formulation of the problem
- General approach to numerical differentiation
- Finite-different approximation of the first-order derivatives
- Finite-different approximation of the second-order derivatives
- Problems
4.1. Numerical differentiation

Formulation of the problem and general approach to numerical differentiation
Let's assume that a functional dependence between two variables $x$ and $y$ is given in the tabulated form: We know $y_i = y(x_i)$, for discrete values of the argument $x_i$, $i = 1, ..., N$. Our goal is to find numerically (i.e. approximately) the derivative $y'(x)$

The general approach to the numerical differentiation involves only two steps:

- We need to find an interpolation polynomial for the given data points
  \[ f(x) = C_1 + C_2 x + C_3 x^2 + \cdots + C_N x^{N-1} \]  
  (4.1.1)
- Then $y'(x)$ can be find approximately as a derivative of the interpolation polynomial
  \[ y'(x) \approx f'(x) = C_2 + 2C_3 x + 3C_4 x^2 + \cdots + (N - 1)C_N x^{N-2} \]
4.1. Numerical differentiation

Nodal values of derivatives at a uniform mesh

- In many applications, e.g. for numerical solution of differential equations, we are interested in finding derivatives

  ✓ For data points given on a mesh with equal spacing $\Delta x$ when $x_i = a + \Delta x(i - 1)$.
  ✓ Only in data points, i.e. finding only nodal values of derivatives $y_i' = y'(x_i)$.

- In this case, the explicit solution of the interpolation problem (finding coefficients $C_i$ of the interpolation polynomial in Eq. (4.1.1)) is not necessary.
- One can obtain very simple and compact formulas for nodal values $y_i' = y'(x_i)$ in terms of $y_i$ and $\Delta x$. Such formulas are known as finite-difference formulas (approximations, equations).
4.1. Numerical differentiation

Taylor series on a uniform mesh

One can use the Taylor series in order to find derivatives on a mesh with equal spacing

\[ y(x) = y(a) + y'(a)(x - a) + \frac{1}{2} y''(a)(x - a)^2 + \frac{1}{6} y'''(a)(x - a)^3 \ldots \]

Let’s use this equation with \( a = x_i \) and different \( x = x_{i-1}, x_{i+1}, \) etc.

\[ y_{i-1} = y_i + y'_i(x_{i-1} - x_i) + \frac{1}{2} y''_i(x_{i-1} - x_i)^2 + \frac{1}{6} y'''_i(x_{i-1} - x_i)^3 + \ldots \]

\[ y_{i+1} = y_i + y'_i(x_{i+1} - x_i) + \frac{1}{2} y''_i(x_{i+1} - x_i)^2 + \frac{1}{6} y'''_i(x_{i+1} - x_i)^3 + \ldots \]

But \( x_{i-1} - x_i = -\Delta x, x_{i+1} - x_i = \Delta x. \) Then

\[ y_{i-1} = y_i - y'_i\Delta x + \frac{1}{2} y''_i\Delta x^2 - \frac{1}{6} y'''_i\Delta x^3 + \ldots \]

\[ y_{i+1} = y_i + y'_i\Delta x + \frac{1}{2} y''_i\Delta x^2 + \frac{1}{6} y'''_i\Delta x^3 + \ldots \]
4.1. Numerical differentiation

First-order derivatives

\[ y_{i-1} = y_i - y_i'\Delta x + \frac{1}{2} y_i''\Delta x^2 - \frac{1}{6} y_i'''\Delta x^3 + \cdots \]  
(4.1.2)

\[ y_i = y_i \]  
(4.1.3)

\[ y_{i+1} = y_i + y_i'\Delta x + \frac{1}{2} y_i''\Delta x^2 + \frac{1}{6} y_i'''\Delta x^3 + \cdots \]  
(4.1.4)

Let’s subtract (4.1.3) from (4.1.2)

\[ y_{i-1} - y_i = -y_i'\Delta x + \frac{1}{2} y_i''\Delta x^2 - \frac{1}{6} y_i'''\Delta x^3 + \cdots \]

and divide the equation by \(-\Delta x\)

\[ y_i' = \frac{y_i - y_{i-1}}{\Delta x} + O(\Delta x) \]

If \(\Delta x\) is small, then

\[ y_i' \approx f_i' = \frac{y_i - y_{i-1}}{\Delta x} \]  
(4.1.5)

- \(f_i'\) in Eq. (4.1.5) is the backward-difference approximation of the first-order derivative.
- \(O(\Delta x^m)\) denotes the rest of terms in the Taylor series. It means that \(O(\Delta x^m) \sim C\Delta x^m \rightarrow 0\) when \(\Delta x \rightarrow 0\). Value of \(m\) characterizes the error of approximation and is called the order of approximation of finite-difference formula (4.5.1).
4.1. Numerical differentiation

\[ y_{i-1} = y_i - y'_i \Delta x + \frac{1}{2} y''_i \Delta x^2 - \frac{1}{6} y'''_i \Delta x^3 + \ldots \]
\[ y_i = y_i \]
\[ y_{i+1} = y_i + y'_i \Delta x + \frac{1}{2} y''_i \Delta x^2 + \frac{1}{6} y'''_i \Delta x^3 + \ldots \]

Let’s subtract (4.1.3) from (4.1.4) and divide the equation by \( \Delta x \)

\[ y'_i = \frac{y_{i+1} - y_i}{\Delta x} + O(\Delta x) \]

If \( \Delta x \) is small, then

\[ y'_i \approx f'_i = \frac{y_{i+1} - y_i}{\Delta x} \]  \hspace{1cm} (4.1.6)

\( f'_i \) in Eq. (4.1.6) is the forward-difference approximation of the first-order derivative.

\( f'_i \) in Eq. (4.1.7) is the central-difference approximation of the first-order derivative.

Let’s subtract (4.1.2) from (4.1.4) and divide the equation by \( \Delta x \)

\[ y'_i = \frac{y_{i+1} - y_{i-1}}{2 \Delta x} + O(\Delta x^2) \]

If \( \Delta x \) is small, then

\[ y'_i \approx f'_i = \frac{y_{i+1} - y_{i-1}}{2 \Delta x} \]  \hspace{1cm} (4.1.7)

\( f'_i \) in Eq. (4.1.7) is the central-difference approximation of the first-order derivative.

Central-difference approximation is the approximation of the second order.
4.1. Numerical differentiation

The same derivative can be approximated with different finite-difference formulas.

If \( y(x) \) is smooth, the higher the order of approximation, the more accurate the formula.
4.1. Numerical differentiation

Second-order derivatives

\[
y_{i-1} = y_i - y_i' \Delta x + \frac{1}{2} y_i'' \Delta x^2 - \frac{1}{6} y_i''' \Delta x^3 + \frac{1}{24} y_i^{(IV)} \Delta x^4 + \ldots
\]

\[
y_i = y_i
\]

\[
y_{i+1} = y_i + y_i' \Delta x + \frac{1}{2} y_i'' \Delta x^2 + \frac{1}{6} y_i''' \Delta x^3 + \frac{1}{24} y_i^{(IV)} \Delta x^4 + \ldots
\]

Let’s sum (4.1.2) with (4.1.4) and subtract 2 x (4.1.3)

\[
y_{i+1} - 2y_i + y_{i-1} = y_i'' \Delta x^2 + \frac{1}{12} y_i^{(IV)} \Delta x^4 + \ldots
\]

and divide the equation by \( \Delta x^2 \)

\[
y_i'' = \frac{y_{i+1} - 2y_i + y_{i-1}}{\Delta x^2} + O(\Delta x^2)
\]

If \( \Delta x \) is small, then

\[
y_i'' \approx f_i'' = \frac{y_{i+1} - 2y_i + y_{i-1}}{\Delta x^2}
\]  \(4.1.8\)

- \( f_i'' \) in Eq. (4.1.8) is the central-difference approximation of the second-order derivative.
- This is the approximation of the second order.
4.1. Numerical differentiation

Summary on numerical differentiating with MATLAB

Central finite differences of the second order of approximation for function $y(x)$:

$$y'_i \approx \frac{y_{i+1} - y_{i-1}}{2\Delta x}, \quad y''_i \approx \frac{y_{i+1} - 2y_i + y_{i-1}}{\Delta x^2}$$

$$y'(x) \approx \frac{y(x + \Delta x) - y(x - \Delta x)}{2\Delta x}, \quad y''(x) \approx \frac{y(x + \Delta x) - 2y(x) + y(x - \Delta x)}{\Delta x^2}$$

Example:

$$y(x) = x^2$$

\[
x = 1.0;
dx = 0.01;
dydx = ( \text{Fun}(x + dx) - \text{Fun}(x - dx) ) / (2.0 * dx);
d2ydx2 = ( \text{Fun}(x + dx) - 2.0 * \text{Fun}(x) + \text{Fun}(x - dx) ) / dx^2;
\]

\[
\text{function} \ [ y ] = \text{Fun}(x)
\]

\[
y = x^2;
\]

end
4.1. Numerical differentiation

Problem 4.1.1: Calculate numerically the first-order derivative of the function

\[ y(x) = \sin(\log(x + 1)) \]

at point \( x_i = 2 \) with \( \Delta x = 0.1 \) and \( \Delta x = 0.01 \) using backward and central finite differences and compare then with accurate values by calculating the relative numerical error:

\[ \varepsilon = \left| \frac{y_i' - y'(x_i)}{y'(x_i)} \right| \]

Accurate derivative is equal to

\[ y'(x) = -\frac{\cos(\log(x + 1))}{x + 1}. \]

File Problem_4_1_1.m:

```matlab
x0 = 2.0;
dx = 0.1;
x = [x0 - dx, x0, x0 + dx];
y = sin(log(x + 1));
dydx = cos(log(x0 + 1.0)) / (x0 + 1.0); % Accurate derivative
dydx_b = (y(2) - y(1)) / (x(2) - x(1)); % Backward difference
err_b = abs((dydx_b - dydx) / dydx) * 100.0 % Relative error of backward difference
dydx_c = (y(3) - y(1)) / (x(3) - x(1)); % Central difference
err_c = abs((dydx_c - dydx) / dydx) * 100.0 % Relative error of central difference
```

- With decreasing \( \Delta x \) in 10 times, \( \text{err\_b} \) decreases in 10 time, while \( \text{err\_c} \) decreases in 100 times!
4.1. Numerical differentiation

**Problem 4.1.2**: Calculation of velocity and acceleration in points of a trajectory given in a tabulated form

\[
\begin{array}{cccccccccc}
 t & t_1 & t_2 & t_3 & \ldots & t_{i-1} & t_i & t_{i+1} & \ldots & t_N \\
 x & x_1 & x_2 & x_3 & \ldots & x_{i-1} & x_i & x_{i+1} & \ldots & x_N \\
y & y_1 & y_2 & y_3 & \ldots & y_{i-1} & y_i & y_{i+1} & \ldots & y_N \\
\end{array}
\]

where \( t_i = (i - 1)\Delta t \) and \( \Delta t \) is the time step. Our goal is to find numerically and plot the body velocity and force affecting the body at times \( t_i \) using central finite differences.

```matlab
function [] = VelocityAndAcceleration ( Dt, X, Y, N )
V = zeros ( N, 2 ); A = zeros ( N, 2 );
XX = zeros ( 2, 1 ); YY = zeros ( 2, 1 );
plot ( X, Y, 'r' );
hold on
for i = 2 : N - 1
    XX(1) = X(i); YY(1) = Y(i);
    V(i,1) = ( X(i+1) - X(i-1) ) / ( 2.0 * Dt );
    V(i,2) = ( Y(i+1) - Y(i-1) ) / ( 2.0 * Dt );
    XX(2) = XX(1) + V(i,1);
    YY(2) = YY(1) + V(i,2);
    plot ( XX, YY, 'b' );
    A(i,1) = ( X(i+1) - 2 * X(i) + X(i-1) ) / ( Dt * Dt );
    A(i,2) = ( Y(i+1) - 2 * Y(i) + Y(i-1) ) / ( Dt * Dt );
    XX(2) = XX(1) + A(i,1);
    YY(2) = YY(1) + A(i,2);
end
end
```

Let's solve the problem assuming that trajectory is given by equations

\[
\begin{align*}
x(t) &= \sin t \\
y(t) &= 2 \cos t - 1
\end{align*}
\]

File Problem_4_1_2:

```matlab
t = [ 0 : 0.1 : 2 ];
X = sin ( t );
Y = 2 * cos ( t ) - 1;
VelocityAndAcceleration ( 0.1, X, Y, 21 );
```
4.2. Numerical Integration

- Formulation of the problem
- Elementary approach for the numerical integration
- Rectangle rule
- Trapezoidal rule
- Numerical integration in the MATLAB

**Reading assignment**

Gilat 9.3
4.2. Numerical integration

Formulation of the problem

Let’s assume that we want to find the value of definite integral $I$ of the integrand $y(x)$ in the interval $[a, b]$

$$I = \int_{a}^{b} y(x)\,dx$$  \hspace{1cm} (4.2.1)

- The integrand $y(x)$ can be given in either the explicit function form (e.g., $y(x) = \exp(-x)\sin(x)$) or tabulated form, i.e. $y_i = y(x_i)$ for discrete $x_i, i = 1, \ldots, N$.
- In many practical cases, we cannot evaluate the integral algebraically, i.e. to find antiderivative $F(x)$, such that $F' = y, I = F(b) - F(a)$.

Example:

$$I = \int_{a}^{b} e^{-x^2}\,dx$$

- Our goal is to find the value of the integral in Eq. (4.2.1) numerically (approximately), i.e. perform numerical integration (find numerical quadrature) of the integrand $y(x)$ in the interval $[a, b]$.
- Quadrature is a historical mathematical term which means determining area.
- The rules of the numerical integration are called quadrature rules.
4.2. Numerical integration

The geometrical meaning of an interval

The integral is the signed area of the shape bounded by the plot of integrand $y(x)$ and axis $Ox$

$$I = A_+ - A_-$$

Major property of definite integral: Additivity on intervals

$$I = \int_{a}^{b} y(x) \, dx = \int_{a}^{c} y(x) \, dx + \int_{c}^{b} y(x) \, dx \quad (4.2.2)$$
4.2. Numerical integration

Elementary approach to the numerical integration

Elementary approach to the numerical integration is based on two ideas:

- We can divide interval \([a, b]\) into many short intervals by a mesh of nodes \(x_i, i = 1, ..., N\).

Then the integral over \([a, b]\) is equal to the sum of intervals for every short interval

\[
\int_{a}^{b} y(x) \, dx = \int_{x_1=a}^{x_2} y(x) \, dx + \int_{x_2}^{x_3} y(x) \, dx \ldots + \int_{x_{N-1}}^{x_N=b} y(x) \, dx = \sum_{i=1}^{N-1} \int_{x_i}^{x_{i+1}} y(x) \, dx \tag{4.2.3}
\]

- For a short interval, integrand can be approximated by a constant value or a linear function.
### 4.2. Numerical integration

**Rectangle (mid-point) rule**

At every mesh interval \( x_i < x < x_{i+1} \), the value of the signed area is estimated as an area of a corresponding *rectangle*.

\[
y(x) \mid_{x_i}^{x_{i+1}} = \int_{x_i}^{x_{i+1}} y(x) \, dx \\
\approx \sum_{i=1}^{N-1} y(x_{i+1/2}) (x_{i+1} - x_i)
\]

(4.2.4)

For a mesh with equal spacing \( \Delta x = x_{i+1} - x_i \), \( x_{i+1/2} = x_i + \Delta x / 2 \):

\[
\int_{a}^{b} y(x) \, dx \approx \Delta x \sum_{i=1}^{N-1} y(x_i + \Delta x / 2)
\]

(4.2.5)
Problem 4.2.1: Application of the rectangle rule for calculation of the position of the center of mass

The position of the center of mass

\[ x_C = \frac{\int_a^b xy(x) \, dx}{\int_a^b y(x) \, dx} \]

Find center of mass for \( y(x) = \sqrt{x - 1} \)
\( a = 2 \text{ in}, \ b = 4 \text{ in} \)

The center of mass is a point with respect to which the torque of gravity force is equal to zero. The body will not spin if it is hanged (pinned) at the center of mass.
4.2. Numerical integration

Trapezoidal rule

At every mesh interval $x_i < x < x_{i+1}$, the value of the signed area is estimated as a signed area of a corresponding trapezium.

For a mesh with equal spacing $\Delta x = x_{i+1} - x_i = \text{const}$:

\[
\int_{a}^{b} y(x)dx \approx \frac{\Delta x}{2} \sum_{i=1}^{N-1} \left[ y_i + y_{i+1} \right] = \Delta x \left( \frac{y_1 + y_N}{2} + \sum_{i=2}^{N-1} y_i \right)
\]
4.2. Numerical integration

Problem 4.2.2: Application of the trapezoidal rule for numerical evaluation of the flow rate

The flow rate $Q$ (volume of fluid per second) in a round pipe can be calculated by:

$$Q = \int_0^r 2\pi v r dr$$

For turbulent flow the velocity profile can be estimated by: $v = v_{max} \left(1 - \frac{r}{R}\right)^{1/n}$. Determine $Q$ for $R = 0.25$ in., $n = 7$, $v_{max} = 80$ in./s.

File TrapezoidalRule.m

```matlab
function [ I ] = TrapezoidalRule ( Fun, a, b, N )
dx = ( b - a ) / ( N - 1 );
x = [ a : dx : b ];
y = Fun ( x );
I = dx * ( 0.5 * ( y(1) + y(N) ) + sum ( y(2:(N-1)) ) );
end
```

File FlowRateInt.m

```matlab
function [ F ] = FlowRateInt ( r )
Vmax = 80;
NFluid = 7;
Rmax = 0.25;
V = Vmax .* ( 1.0 - r / Rmax ).^((1/NFluid) ;
F = 2.0 * pi * r .* V;
end
```

File Problem_4_2_2:

FlowRate = TrapezoidalRule ( @FlowRateInt, 0, 0.25, 101 )

The trapezoidal rule is more accurate (provides smaller numerical error) than the rectangle rule (compare the shadowed areas in two previous figures).
4.2. Numerical integration

Build-in MATLAB functions for numerical integration

- In order to calculate numerically the integrals of the form \( I = \int_a^b y(x)dx \) one can use the build-in `quad` function.

- It implements **adaptive integration algorithms**, when the mesh spacing \( \Delta x_i = x_{i+1} - x_i \) varies in the course of integration in order to ensure that error is within given tolerance.

- Syntax: \( I = \text{quad}( \text{@Fun}, a, b, \text{Tol} ) \)
  
  - `Fun` is the (user-defined) function that implements calculation of the integrand \( f(x) \)
  
  - `a` and `b` are the left- and right-hand side limits of the integration interval \([a, b]\).
  
  - `Tol` in the optional tolerance parameter for the adaptive integration method. Default Tol=1.0e-6.

- **This is important:** In order to be used in the `quad` function, the integrand should be written for an input argument that is a **vector** of mesh values \( x_i \) and the output argument that is a **vector** of the mesh values of \( y_i \). We must use the **elementwise** operations.
Problem 4.2.3: Calculation of the perimeter of Pluto’s orbit with MATLAB quad function

The orbit of Pluto is elliptical in shape, with \( a = 5.9065 \times 10^9 \text{ km} \) and \( b = 5.7208 \times 10^9 \text{ km} \). The perimeter of an ellipse can be calculated by

\[
P = 4a \int_0^{\pi/2} \sqrt{1-k^2 \sin^2 \theta} \, d\theta
\]

where \( k = \frac{\sqrt{a^2-b^2}}{a} \). Determine the distance Pluto travels in one orbit.

File PlutoOrbitInt.m:

```matlab
function [ Y ] = PlutoOrbitInt ( Theta )

a = 5.9065e+9;
b = 5.7208e+9;
k = sqrt ( a^2 - b^2 ) / a;
Y = 4 * a * sqrt ( 1 - k^2 * sin ( Theta ) .^2 );
end
```

File Problem_4_2_3.m:

```matlab
P = quad ( @PlutoOrbitInt, 0, pi / 2 )
P = 3.653056964855344e+10
```
4.2. Numerical integration

Summary on numerical integration with build-in MATLAB function quad

The MATLAB build-in function `quad` allows one to find a definite integral:

\[ I = \int_{a}^{b} y(x) dx \]

Example:

\[ y(x) = x^2, \quad a = -2, \quad b = 2 \]

```matlab
function [ y ] = fun ( x )
    y = x .^ 2;
end
I = quad ( @fun, -2.0, 2.0 )
```

Element-by-element operations must be used !!!
4.3. Ordinary differential equations (ODEs). First-order ODEs

- A concept of an ordinary differential equation. Order of an ODE
- General and particular solution of an ODE
- Initial conditions. Initial value problem for a 1st order ODE
- Example: Radioactive decay
4.3. Ordinary differential equations (ODEs). First-order ODEs

**Equation** is a way to formulate a mathematical problem. The solution of the problem (unknown) can be a number, a function, etc.

**Differential equation** is an equation

- where the unknown is a function of one or a few independent variables.
- which contains derivatives of the unknown function.

**Ordinary differential equation (ODE)** is a differential equation where unknown is a function of a single independent variable.

**Example:**

\[
y' = \frac{dy}{dx} = -\lambda y \quad : \text{differential equation}
\]

\[
x: \text{ Independent variable}
\]

\[
y(x): \text{ Unknown function}
\]

The general form of an ODE is

\[
f (x, y, y', y'', y''', ..., y^{(n)}) = 0 \quad (4.3.1)
\]

where \( y^{(n)} = \frac{d^n y}{dx^n} \) is the derivative of \( n^{\text{th}} \) order.

**Order** of an ODE is the highest order of derivatives in Eq. (4.3.1).

**Examples:**

\[
a(x) y' + b(x) y = c(x) \quad : \text{Linear differential equation of the 1}^{\text{st}} \text{ order}
\]

\[
a(x)y'' + b(x) y' + c(x) y = d(x) \quad : \text{Linear differential equation of the 2}^{\text{nd}} \text{ order}
\]
4.3. Ordinary differential equations (ODEs). First-order ODEs

- To **solve** an ODE means to find all functions which turn the equation into identity. Any such function \( y = g(x) \) is called the **particular solution** of the ODE. The process of solving ODEs is called **integration**.

- **Any ODE has multiple solutions**, i.e. a solution of an ODE is non-unique. The main reason for this is that different functions can have the same derivatives.

**Example:**

\[ y' = x/y. \]

This is separable equation with the solution

\[ x^2 - y^2 = c. \]

Check: \( 2x - 2yy' = 0. \)

The general form of an **ODE of the 1st order** is

\[ f(x, y, y') = 0. \quad (4.3.2) \]

A solution of Eq. (4.3.2) is not unique. A set of different solutions of Eq. (4.3.2) can be written in the form

\[ G(x, y, c) = 0. \quad (4.3.3) \]

where \( c \) is an arbitrary real number. Solution in the form (4.3.3) is called **general solution**, since this equation includes a lot of solutions for different \( c \). A **particular solution** of Eq. (4.3.2) can be obtained from its general solution if variable \( c \) is replaced by some particular real number.
4.3. Ordinary differential equations (ODEs). First-order ODEs

Initial value problem

In many engineering applications we are not interested in the general solution of an ODE, but we are interested in the particular solution that satisfies some additional condition(s). For the 1st order ODE of in the explicit form (resolved with respect to the derivative)

\[ y' = f(x, y) \]

such conditions can be formulated as a requirement that at some given point \( x = x_0 \) the solution \( y(x) \) is equal to the prescribed value \( y_0 \), i.e.

\[ y(x_0) = y_0 \quad \text{or} \quad y\big|_{x=x_0} = y_0 \]

Eq. (4.3.5) is called the initial condition for Eq. (4.3.4).

A problem given by (4.3.4) and (4.3.5) is called the initial value (or Cauchy) problem (IVP).

Example:

ODE: \[ dy/dx = x/y \]

Initial condition: \[ y(1) = 2 \]

General solution: \[ ydy = xdx \quad \Rightarrow \quad \int ydy = \int xdx + c \]

\[ x^2 - y^2 = c \]

Solution of the IVP: \[ c = 1^2 - 2^2 = -3 \quad \text{and} \quad y = \sqrt{x^2 + 3} \]

If we want to check that some formula is a solution of the IVP, we need to check that

- This formula turns the ODE into identity.
- This formula satisfies the initial condition.
### 4.3. Ordinary differential equations (ODEs). First-order ODEs

**Problem 4.3.1:** Radioactive decay of a radioactive substance

**Physical law:** Radioactive decay rate (number of nuclei exhibiting decay per unit time) is proportional to the current number of nuclei.

\[ N(t), \text{ number of nuclei at time } t \]
\[ N_0 = N(0), \text{ Initial conditions: number of nuclei at initial time } t = 0 \]
\[ N' = dN/dt < 0, \text{ decay rate with the negative sign} \]

1\textsuperscript{st} order ODE \quad dN/dt = -\lambda N

General solution: \quad N(t) = N_0 \exp(-\lambda t)

Particular solution: \quad N(t) = 10^6 \exp(-\lambda t)

Check: \quad N' = -\lambda N_0 \exp(-\lambda t) = -\lambda N

**Half-life** \( \tau \) is the time when a half of initial nuclei decayed

\[ N(\tau) = N_0/2 = N_0 \exp(-\lambda \tau) \]
\[ \tau = \ln 2 / \lambda \]

---

**File RadioactiveDecay.m**

```matlab
function Y = RadioactiveDecay ( X, Lam, A, YA )
    Y = YA * exp ( - Lam * ( X - A ) );
end
```

---

**File Problem_4_3_1.m**

- HL = 10.0;
- Lam = log ( 2.0 ) / HL;
- t = [ 0 : 0.01 : 10 0];
- N0 = 1.0e+06;
- N = RadioactiveDecay ( t, Lam, 0.0, N0 );
- ```semilogy ( t, N )```
4.4. Numerical methods for initial value problems (individual ODEs)

- Basic concepts of the numerical methods for IVPs
- The Euler method
- Implementation of the IVP solver in MATLAB
- Truncation errors and order of approximation
- The Runge-Kutta methods
- Numerical solution of IVPs in the MATLAB

**Reading assignment**

Gilat 9.4
4.4. Numerical methods for initial value problems (individual ODEs)

Let’s consider a 1st ODE in the explicit form

\[ y' = f(x, y) \]

Assume that we are not able to solve this equation algebraically, i.e. we cannot find a function \( y = g(x, c) \) or \( G(y, x, c) = 0 \) that represents the general solution of this equation.

On the other hand, virtually any ODE can be solved numerically with the help of a computer that performs basic algebraic operations (\( +, \times, \div \)) with integer and real numbers.

In order to solve an IVP numerically, we need to develop an algorithm that allows us to calculate approximate values of the unknown function in this equation with a finite number of algebraic operations. Such algorithms for solving different mathematical problems are called numerical methods or numerical schemes.

**Basic concepts of the numerical methods for IVPs**

1. Numerically we can look only for a particular solution of an ODE or a system of ODEs, i.e. we can solve only initial or boundary value problems. For the first-order ODE we can solve numerically only the Cauchy problem with the initial condition

\[ y' = f(x, y), \quad y(a) = y_0, \quad a \leq x \leq b \quad (4.1.1) \]
4.4. Numerical methods for initial value problems (individual ODEs)

2. We cannot find numerically the function \( y = y(x) \). We are looking for the numerical values of this function at some prescribed values \( x_0, x_1, \ldots \), etc. of \( x \), where \( x_{i+1} > x_i \). In other words, we are looking for a solution represented in the tabulated form:

<table>
<thead>
<tr>
<th>( x )</th>
<th>( x_0 = a )</th>
<th>( x_1 )</th>
<th>...</th>
<th>( x_i )</th>
<th>...</th>
<th>( x_N = b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y )</td>
<td>( y_0 )</td>
<td>( y_1 )</td>
<td>...</td>
<td>( y_i )</td>
<td>...</td>
<td>( y_N )</td>
</tr>
</tbody>
</table>

This table is called the numerical solution of problem (4.4.1).

3. In general, the numerical solution is approximate in a sense that \( y(x_i) \neq y_i \) (Here \( y(x) \) is the presumed exact solution of the Cauchy problem (4.4.1)). The difference

\[
\varepsilon_i = y(x_i) - y_i
\]

is called the numerical (or global truncation) error of the numerical solution in point \( x_n \).
4.4. Numerical methods for initial value problems (individual ODEs)

The Euler method

Let's develop the simplest numerical method to solve an IVP

\[ \frac{dy}{dx} = f(x, y), \quad y(x_0) = y_0, \quad x_0 = a \leq x \leq b. \]  \hspace{1cm} (4.4.1)

1. Let's introduce a mesh of nodes \( x_i \) with the spacing \( \Delta x \): \( x_{i+1} = x_i + \Delta x \).
2. Let's approximate derivative in the ODE in node \( x_i \) with the forward finite difference

\[ \frac{y(x_{i+1}) - y(x_i)}{x_{i+1} - x_i} = f(x_i, y_i) \quad \text{or} \quad \frac{y_{i+1} - y_i}{\Delta x} = f(x_i, y_i). \]  \hspace{1cm} (4.4.2)

3. We start from the point \( x = x_0 = a \) and \( y = y_0 \) given by the initial condition and apply Eq. (4.4.2) in order to find \( y_1 \):

\[ x_1 = x_0 + \Delta x, \quad y_1 = y_0 + f(x_0, y_0)\Delta x \]

3. Now we can repeat step 2 recursively

\[ x_{i+1} = x_i + \Delta x, \quad y_{i+1} = y_i + f(x_i, y_i)\Delta x \]  \hspace{1cm} (4.4.3)

until \( x_{n+1} = x_0 + (n + 1)\Delta x > b \).

The numerical method given by Eqs. (4.4.3) is called the **(explicit) Euler method**.

Graphically, the numerical algorithm of the Euler method can be shown as follows:
4.4. Numerical methods for initial value problems (individual ODEs)

- The whole process of numerical solution looks like a sequence of individual integration steps.

![Diagram](image)

- Value $\Delta x_i = x_i - x_{i-1}$ (increment of the independent variable during one integration step) is called the integration step size. In our case $\Delta x_i = \Delta x = \text{const}$, but in general the integration step does not need to be constant at every $n$.

- Numerical solution is the solution of the finite difference equation and depends on the numerical parameter(s), the integration step size, which is absent in the original IVP.

- The integration step size controls the numerical error $\varepsilon_i = y(x_i) - y_i$, where $y(x)$ is the accurate solution of the problem given by Eq. (4.4.1).

- Our obvious goal is to chose such numerical parameter that allows us to obtain a numerical solution that approximates the accurate solution of the problem with sufficient accuracy.
4.4. Numerical methods for initial value problems (individual ODEs)

Implementation of the IVP solver in MATLAB

\[ y' = f(x, y), \quad y(x_0) = y_0, \quad x_0 = a \leq x \leq b \]

- **RHS** (Problem)
  Calculation of the RHS function
  \[ f(x, y) \]

- **Integrator** (Method)
  One integration step
  \[ x_{i+1} = x_i + \Delta x \]
  \[ y_{i+1} = y_i + f(x_i, y_i)\Delta x \]

- **Solver** (Integration loop)
  Solution in the whole interval \([a, b]\) for given initial conditions \(y(x_0) = y_0\) with given integration step \(\Delta x\)
  Plotting solution, etc.

- Alternative **solution** (if known), Error analysis, etc.

- The RHS, Integrator, and Solver are usually implemented in the form of separate functions.

- It allows one to use the same Integrator and Solver to solve different IVPs or use different Integrators to solve the same IVP.
Problem 4.4.1: Solve the IVP for the radioactive decay problem with the Euler method

\[ y' = -y, \quad y(0) = 1, \quad 0 \leq x < 20. \]

File `ODESolver1.m`

```matlab
function [X, Y] = ODESolver1 (Integrator, RHS, A, B, YA, DX)

NI = int64((B - A) / DX + 1); % Number of integration steps
X = zeros(NI, 1);
Y = zeros(NI, 1);

% Initial condition
X(1) = A;
Y(1) = YA;

for i = 1 : NI - 1 % Now we can perform NI integration steps
    [X(i+1), Y(i+1)] = Integrator(X(i), Y(i), DX, RHS);
end
```

File `Problem_4_4_1.m`

```matlab
[X, Y] = ODESolver1(@Euler, @RHSDecay, 0, 10, 2, 0.1);
plot(X, Y);
```

File `RHSDecay.m`

```matlab
function F = RHSDecay (X, Y)
    F = -Y;
end
```

File `Euler.m`

```matlab
function [X,Y] = Euler (X0, Y0, DX, RHS)
    F = RHS(X0, Y0);
    Y = Y0 + DX * F;
    X = X0 + DX;
end
```
4.4. Numerical methods for initial value problems (individual ODEs)

Version of the solver that allows us to compare the numerical and accurate solutions of the problem:

File ODESolver1S.m

```matlab
function [ X, Y ] = ODESolver1S ( Integrator, RHS, A, B, YA, DX, Sol )
    NI = int64 ( ( ( B - A ) / DX ) + 1 ); % Number of integration steps
    X = zeros ( NI, 1 ); Y = zeros ( NI, 1 ); Yacc = zeros ( NI, 1 ); E = zeros ( NI, 1 );
    X(1) = A; Y(1) = YA; % Initial condition
    Yacc(1) = Sol ( X(1), A, YA ); E(1) = 0;
    for i = 1 : NI - 1 % Now we can perform NI integration steps
        [ X(i+1), Y(i+1) ] = Integrator ( X(i), Y(i), DX, RHS );
        Yacc(i+1) = Sol ( X(i+1), A, YA );
        E(i+1) = abs ( ( Yacc(i+1) - Y(i+1) ) / Yacc(i+1) ); % Error
    end
    figure ( 1 ); % First figure is the solution
    plot ( X, Y, 'r-', X, Yacc, 'g--' ) ;
    title ('Numerical (red) and accurate (green) solutions');
    figure ( 2 ); % Second figure is the numerical error
    loglog ( X, E ); title ('Relative error of the numerical solution');
end
```

File Problem_4_4_1_S.m:

```matlab
ODESolver1S ( @Euler, @RHSDecay, 0, 10, 2, 0.1, @SolDecay );
```

File SolDecay.m

```matlab
function Y = SolDecay ( X, A, YA )
    Y = YA * exp ( A - X );
end
```
4.4. Numerical methods for initial value problems (individual ODEs)

**IVP:** \( y' = -y, \ y(0) = 1, \ 0 \leq x < 20. \) Accurate solution: \( y(x) = \exp(-x). \)

**Conclusions:**
1. The smaller the integration step, the better the accuracy of the numerical solution.
2. \( |\varepsilon_i|/y(x_i) \) increases with \( i \), so that eventually the error becomes unacceptably large.
3. For the Euler method, the numerical error at given \( x \) is proportional to \( \Delta x \).
4. There is a some “critical value” \( \Delta x_{\text{crit}} \). If \( \Delta x \geq \Delta x_{\text{crit}} \), then the numerical solution has no physical sense and useless.
4.4. Numerical methods for initial value problems (IVPs) for individual ODEs

**Truncation error and order of approximation**

The difference between numerical $y_i$ and accurate solutions $y(x)$

$$\varepsilon_i = y_i - y(x_i)$$

is called the global truncation error after $i$ integration steps. Local truncation error of the numerical method is the error

$$\tau_i = y(x_i) - y_i$$
calculated assuming that all previous numerical values are accurate, i.e. assuming that $y_j = y(x_j)$ for $j < i$. Local truncation error characterizes the error we make at one integration step. Global error $\varepsilon_i$ increases with the number of steps $i$ and can be much larger than $\tau_i$.

We say that the numerical method (approximates) the initial ODE if $\tau_i \to 0$ as $\Delta x \to 0$.

If $\tau_i = O(\Delta x^{k+1})$ then we say that the numerical method has $k^{th}$ order of approximation. The higher order of approximation, the more accurate the numerical method.
The Euler method has the first order of approximation. Usually, it is insufficiently accurate to solve practical problems.

**The Runge-Kutta methods**

The idea of the Runge-Kutta methods is to use additional values of the RHS of the ODE at the interval \( x_i \leq x \leq x_{i+1} \) in order to *increase the order of approximation and accuracy*.

1\(^{\text{st}}\) order Euler method

\[
\begin{align*}
y_{i+1} & = y_i + \Delta xf(x_i, y_i) \\
y_{i+1} & = y_i + \Delta f(x_i, y_i)
\end{align*}
\]

2\(^{\text{nd}}\) order Runge-Kutta

\[
\begin{align*}
k_1 & = \Delta f(x_i, y_i) \\
k_2 & = \Delta f(x_i + 0.5\Delta x, y_i + 0.5k_1) \\
y_{i+1} & = y_i + k_2, \quad x_{i+1} = x_i + \Delta x
\end{align*}
\]

It can be proved that RK2 method has the second order of approximation.
4.4. Numerical methods for initial value problems for individual ODEs

Implementation of RK2 in MATLAB:

File RK2.m

```matlab
function [ X, Y ] = RK2 ( X0, Y0, DX, RHS )
    K1 = DX * RHS ( X0, Y0 );
    K2 = DX * RHS ( X0 + 0.5 * DX, Y0 + 0.5 * K1 );
    X = X0 + DX;
    Y = Y0 + K2;
end
```

File Problem_4_4_1_RK2.m

ODESolver1S ( @RK2, @RHSDecay, 0, 10, 1, 0.1, @SolDecay );

- Development of the RK methods of higher order requires complex algebra.

- Popular Runge-Kutta method of the 4th order (RK4) can be formulated as follows:

\[
\begin{align*}
    k_1 &= \Delta x f(x_i, y_i) \\
    k_2 &= \Delta x f(x_i + 0.5\Delta x, y_i + 0.5k_1) \\
    k_3 &= \Delta x f(x_i + 0.5\Delta x, y_i + 0.5k_2) \\
    k_4 &= \Delta x f(x_i + \Delta x, y_i + k_3) \\
    y_{i+1} &= y_i + \frac{1}{6} (k_1 + 2k_2 + 2k_3 + k_4) \\
    x_{i+1} &= x_i + \Delta x
\end{align*}
\] (4.4.5)

- See MATLAB implementation of RK4 on the blackboard.
4.4. Numerical methods for initial value problems (individual ODEs)

Problem 4.4.2: Solution of an IVP with the RK2 method

\[ y' = \frac{\cos y}{1 + x^2} e^x \]

\[ y(0) = 0, \quad 0 \leq x < 2 \]

with the integration step size 0.01.

File RHS_4_4_2.m

```matlab
function F = RHS_4_4_2 (X, Y)
    F = cos(Y) * exp(X) / (1 + X^2);
end
```

File Problem_4_4_2.m

```matlab
[X, Y] = ODESolver1 (@RK2, @RHS_4_4_2, 0.0, 2.0, 0.0, 0.01);
plot(X, Y);
```
4.4. Numerical methods for initial value problems (individual ODEs)

Build-in MATLAB functions for numerical solutions of the IVP for first-order ODEs

\[ y' = f(x, y), \quad y(x_0) = y_0, \quad x_0 = a \leq x \leq b \]

MATLAB has a lot of build-in solvers (integrators) for IVPs for first-order ODEs that implement numerical methods with the adaptive step size control, when the integration step size is changed in the course of integration in order to ensure that the numerical error is within a given tolerance.

These solvers implement different numerical methods, but have the same syntax.

**Syntax:** \([ x, y ] = \textbf{Solver} \ ( @\text{Fun}, x\text{span}, y_0 )\)

- **Solver** is the name of the solver (see the next slide).
- Fun in the (user-defined) function that implements calculation of the RHS \( f(x, y) \).
- xspan is 1D array that should contain at least two real values. The first and last elements of xspan are used as limits of the integration interval \([a, b]\).
- ya is the initial condition for \( y \) at \( x = a \).
- \( x \) and \( y \) are the column vectors with nodes values of \( x \) and \( y \) obtained as a result of integration. Values of \( x \) depend on xspan (see Gilat's textbook, p. 306).
### 4.4. Numerical methods for initial value problems (individual ODEs)

**Build-in MATLAB solvers/integrators for IVPs**

<table>
<thead>
<tr>
<th>ODE Solver Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ode45</td>
<td>For nonstiff problems, one-step solver, best to apply as a first try for most problems. Based on explicit Runge-Kutta method.</td>
</tr>
<tr>
<td>ode23</td>
<td>For nonstiff problems, one-step solver. Based on explicit Runge-Kutta method. Often quicker but less accurate than ode45.</td>
</tr>
<tr>
<td>ode113</td>
<td>For nonstiff problems, multistep solver.</td>
</tr>
<tr>
<td>ode15s</td>
<td>For stiff problems, multistep solver. Use if ode45 failed. Uses a variable order method.</td>
</tr>
<tr>
<td>ode23s</td>
<td>For stiff problems, one-step solver. Can solve some problems that ode15s cannot.</td>
</tr>
<tr>
<td>ode23t</td>
<td>For moderately stiff problems.</td>
</tr>
<tr>
<td>ode23tb</td>
<td>For stiff problems. Often more efficient than ode15s.</td>
</tr>
</tbody>
</table>

This should be the first solver to try.

**File Problem_4_4_1_ode45.m**

```matlab
[ x, y ] = ode45 ( @RHSDecay, [ 0.0, 20.0 ], 1.0 ) ;
plot ( x, y );
```
Comparison of numerical accuracy of the RK2 with fixed integration step size and **ode45** with adaptive step size control

Relative numerical error in the "exponential decay problem" (slides 28, 37)

- Adaptive step size control algorithm is not absolutely universal and sometimes leads to unsatisfactory results.
- Constant integration step size often results in a linear increase of the numerical error. The integration step size appropriate for a particular problem can be chosen by means of experimentation (obtaining a series of numerical solutions with gradually decreasing $\Delta x$).
4.4. Numerical methods for initial value problems (individual ODEs)

Problem 4.4.3: Solve the following IVP with ode45

\[ y' = -y(0.5 + 4 \tan(4t)), \quad y(0) = 2, \quad 0 \leq x < 10 \]

File RHSOsc.m

```matlab
function [ F ] = RHSOsc ( X, Y )
    F = - Y * ( 0.5 + 4 * tan ( 4 * X ) );
end
```

File Problem_4_4_3.m

```matlab
[ t, Y ] = ode45 ( @RHSOsc, [ 0, 10 ], 2.0 );
plot ( t, Y, 'r' );
[ t, Y ] = ode45 ( @RHSOsc, [ 0, 2, 4, 6, 8, 10 ], 2.0 );
```

Output:

<table>
<thead>
<tr>
<th>t</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2.000000000000000000</td>
</tr>
<tr>
<td>2</td>
<td>-0.117370766502767</td>
</tr>
<tr>
<td>4</td>
<td>-0.300818077404844</td>
</tr>
<tr>
<td>6</td>
<td>0.053507383841958</td>
</tr>
<tr>
<td>8</td>
<td>0.036993807372788</td>
</tr>
<tr>
<td>10</td>
<td>-0.011835293403053</td>
</tr>
</tbody>
</table>

- If parameter xspan of ode45 contains more than two values, then the solver will evaluate \( y(x) \) only for \( x \) from array xspan.
- In this case, after execution of \([ X, Y ] = \text{ode45} ( \ldots, \text{xspan}, \ldots )\), \( X = \text{xspan} \).
- For details, see Gilat's textbook, p. 306.
4.4. Numerical methods for initial value problems (individual ODEs)

Summary on numerical integration of ODEs with build-in MATLAB function `ode45`

The MATLAB build-in function `ode45` allows one to solve an initial value problem:

$$\frac{dy}{dx} = f(x, y), \quad y(x_0) = y_0, \quad x_0 = a \leq x \leq b.$$ 

\[
[ x, y ] = \text{ode45} \left( \@\text{fun}, [ a, b ], y0 \right)
\]

Example:

$$\frac{dy}{dx} = \sin(x) \cdot y^2, \quad y(1) = -1, \quad 1 \leq x \leq 2$$

\[
\text{function} \ [ f ] = \text{fun} \ ( x, y ) \\
f = \sin( x ) \cdot y^2;
\end{function}
\]

\[
[ x, y ] = \text{ode45} \left( \@\text{fun}, [ 1.0, 2.0 ], -1.0 \right);
\]

plot ( x, y )
4.5. Second- and higher order ODEs. Systems of ODEs

- Second order ODEs. Initial value problem (IVP)
- Example: Mass-spring mechanical system
- Systems of ODEs. Fundamental systems
- Reduction of higher order ODEs in the normal form to a fundamental system of first-order ODEs.
- Matrix notation for fundamental systems

Reading assignment
Gilat 9.4
4.5. Second- and higher order ODEs. Systems of ODEs

Second-order ODE. Initial value problem

General form of the 2\textsuperscript{nd} order ODE

\[ f(x, y, y', y'') = 0 \]  
(4.5.1)

We will consider only equations in the explicit form, resolved with respect to the highest derivative

\[ y'' = f(x, y, y') \]  
(4.5.2)

The general solution of Eq. (4.5.2) depends on two constants \( c_1 \) and \( c_2 \) and can be written as:

\[ G(x, y, c_1, c_2) = 0 \]  
(4.5.3)

Example: \( y'' = f(x) \)

\[ y'(x) = \int f(x)dx + c_1 = h(x, c_1), \quad y(x) = \int \left[ \int f(x)dx + c_1 \right] dx + c_2 = g(x, c_1, c_2) \]

Any particular choice of constants \( c_1 \) and \( c_2 \) gives a particular solution of Eq. (4.5.2).

One possible way to determine \( c_1 \) and \( c_2 \) is to specify the initial conditions

\[ y(x_0) = y_0, \quad y'(x_0) = y'_0 \]  
(4.5.4)

The problem given by Eqs. (4.5.2) and (4.5.4) is called the initial value (or Cauchy) problem (IVP) for the 2\textsuperscript{nd} order ODE.

In mechanics, if \( x = t \) is time and \( y(t) \) is coordinate, then \( y'(t) \) is velocity and \( y''(t) \) is acceleration. In order to solve the 2\textsuperscript{nd} order ODE with respect to the coordinate (Newton’s second law of motion) we need to specify the initial position of the body and its velocity.
4.5. Second- and higher order ODEs. Systems of ODEs

**Example: Mechanical mass-spring system**

Mechanical mass-spring system

Newton’s second law of motion:

\[ my'' = \sum F_i = F_1 + F_2 + F_3 \]

1. **Elastic restoring force** (Hook’s law):
   \[ F_1 = -ky \]
   
   *k* is the **spring constant** (spring stiffness)

2. **Damping (friction) force**:
   \[ F_2 = -cy' \]

3. **Input (driving) external force**
   \[ F_3 = r(t) \]

**ODE:**

\[ my'' + cy' + ky = r(t) \]

**Initial conditions:**

\[ y(t_0) = y_0, \ y'(t_0) = v_0 \]

- **Undamped oscillation**: \( c = 0 \)
- **Damped oscillation**: \( c \neq 0 \)
- **Free oscillation**: \( r(t) = 0 \)
- **Forced oscillation**: \( r(t) \neq 0 \)
4.5. Second- and higher order ODEs. Systems of ODEs

Systems of ODEs

Many scientific and engineering problems are formulated in terms of systems of ODEs. A system of ODEs is a set of ODEs, when RHSs in individual equations depend on a few unknown functions, so all equations should be solved simultaneously.

Example: 3D motion of a body, e.g. a spacecraft on a LEO

\[
\begin{align*}
mx'' &= F_{Gx} + F_{Dx} \\
my'' &= F_{Gy} + F_{Dy} \\
mz'' &= F_{Gz} + F_{Dz}
\end{align*}
\]

- \( F_G \): Gravity force
- \( F_D \): Aerodynamic drag force

\[
\begin{align*}
r &= \sqrt{x^2 + y^2 + z^2} \\
v &= \sqrt{x'^2 + y'^2 + z'^2} \\
F_{G\alpha} &= -\frac{\partial U}{\partial \alpha} = -\frac{GMm \alpha}{r^2} \\
F_{D\alpha} &= -\frac{1}{2} C_D \rho Av \alpha'
\end{align*}
\]

The RHS of every equation of motion depends on all current coordinates and velocity components of the body and, thus, \( x(t) \) can not be found independently of \( y(t) \) and \( z(t) \).
4.5. Second- and higher order ODEs. Systems of ODEs

Fundamental systems of ODEs

Among all systems of ODEs, the systems of the first-order equations in the explicit form play a fundamental role.

\[
y'_1 = f_1(x, y_1, y_2, \ldots, y_{j-1}, y_j, y_{j+1}, \ldots, y_n) \\
\vdots \\
y'_n = f_n(x, y_1, y_2, \ldots, y_{j-1}, y_j, y_{j+1}, \ldots, y_n)
\]  \hspace{1cm} (4.5.5)

This is the \textbf{n-dimensional fundamental system}, i.e. a system of \( n \) first-order ODEs written in the explicit form. The \textbf{dimension} of a fundamental system of ODEs is the number of unknown functions (equations).

**General solution of system** (4.5.5) is the system of \( n \) functions

\[
y_1 = g_1 (x, c_1, \ldots, c_n), \ldots, y_n = g_n (x, c_1, \ldots, c_n)
\]

which turn every ODE in (4.5.5) into identity. Every function includes \( n \) arbitrary constants \( c_1, \ldots, c_n \). These constants can be determined from the \textbf{initial conditions} of the \textbf{initial value problem}:

\[
y_1(x_0) = y_{1,0}, \quad y_2(x_0) = y_{2,0}, \quad y_n(x_0) = y_{n,0}.
\]  \hspace{1cm} (4.5.6)

Systems (4.5.6) are called \textbf{fundamental}, because \textit{any system (or individual equation) in the explicit form reduces to a system of form (4.5.6) by a proper change of variables.}
4.5. Second- and higher order ODEs. Systems of ODEs

**Example 1**: Transform equation \( y'' - ay = \exp x \) into a fundamental system

1. Let's re-write equation with in the *explicit* form
   \[
y'' = ay + \exp x
   \]
   It means that \( f(x, y, y') = ay + g(x) \).

2. Let's introduce new variables:
   \[
y_1 = y, \quad y_2 = y_1 = y'
   \]
   Number of variables = order of the equation.
   \[
f(x, y, y', y'') = f(x, y_1, y_2, y_3) = y_1
   \]

3. Then
   \[
   y'_1 = y_2 \\
   y'_2 = ay_1 + \exp x
   \]

*Equation of 2nd order reduces to a 2-dimensional fundamental system*
Example 2: Transform equation $y''' - y' = 0$ into a fundamental system

1. Let’s re-write equation with in the explicit form

$$y''' = y'$$

It means that $f(x, y, y', y'') = y'$.

2. Let’s introduce new variables:

$$y_1 = y, \quad y_2 = y_1' = y', \quad y_3 = y_2' = y''$$

Number of variables = order of the equation.

$$f(x, y, y', y'') = f(x, y_1, y_2, y_3) = y' = y_2$$

3. Then

$$y_1' = y_2$$
$$y_2' = y_3$$
$$y_3' = y_2$$

Equation of 3rd order reduces to a 3-dimensional fundamental system
4.5. Second- and higher order ODEs. Systems of ODEs

Example 3: Transform the system of equations

\[ y'' - 2z' + x = 0, \quad z' - xz = 0 \]

into a fundamental system

1. Let's re-write equations of the system in the explicit form:

\[ y'' = 2xz - x, \quad z' = xz \]

2. Let's introduce new variables:

\[ y_1 = y, \quad y_2 = y'_1 = y', \quad y_3 = z \]

Number of variables = sum of orders of individual equations.

3. Then

\[ y'_1 = y_2 \]
\[ y'_2 = x(2y_3 - 1) \]
\[ y'_3 = y_3 \]

System of equations of 2\textsuperscript{nd}+1\textsuperscript{st} orders reduces to a 3-dimensional fundamental system
### 4.5. Second- and higher order ODEs. Systems of ODEs

General approach for reducing of an equation/system to a fundamental systems

Let’s consider a single equation of \( n \)th order and show that it reduces to a fundamental system:

\[
y^{(n)} = f(x, y, y', y'', ..., y^{(n-1)})
\]

(4.5.7)

Let’s introduce \( n \) new functions

\[
y_1 = y, \quad y_2 = y'_1 = y', \quad y_3 = y'_2 = y'', \quad ..., \quad y_n = y'_{n-1} = y^{(n-1)}
\]

Then one can solve a fundamental system

\[
\begin{align*}
y'_1 &= y_2 \\
y'_2 &= y_3 \\
&\quad \vdots \\
y'_{n-1} &= y_n \\
y'_n &= f(x, y_1, y_2, ..., y_{j-1}, y_j, y_{j+1}, ..., y_n)
\end{align*}
\]

(4.5.8)

Once system (4.5.8) is solved, function \( y_1 = y \) gives the solution of the original equation (4.5.7).

**Note:**

- Reduction to fundamental system is often used when ODEs are solved numerically.
- Most of numerical methods are designed to solve only fundamental systems of form (4.5.5).
- After an appropriate change of variables, the majority of practical problems can be formulated in terms of fundamental systems.
4.5. Second- and higher order ODEs. Systems of ODEs

Matrix notation for fundamental systems

Fundamental systems

\[ y_1' = f_1(x, y_1, y_2, \ldots, y_{j-1}, y_j, y_{j+1}, \ldots, y_n) \]

\[ \ldots \]

\[ y_n' = f_n(x, y_1, y_2, \ldots, y_{j-1}, y_j, y_{j+1}, \ldots, y_n) \]

are often formulated and solved using the matrix notation: Let’s introduce column vectors of unknowns \( \mathbf{Y} \), their derivatives \( \mathbf{Y}' \), and RHSs \( \mathbf{F} \):

\[
\mathbf{Y} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}, \quad \mathbf{Y}' = \begin{pmatrix} y_1' \\ \vdots \\ y_n' \end{pmatrix}, \quad \mathbf{F} = \begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix} = \mathbf{F}(x, \mathbf{Y})
\]

Then the fundamental system (4.5.6) can be written as

\[ \mathbf{Y}' = \mathbf{F}(x, \mathbf{Y}) \]  \hspace{1cm} (4.5.9)

- The vector notation, Eq. (4.5.9), allows us to write an arbitrary fundamental system in the form that is similar to the single first-order ODE in the explicit form

\[ y' = f(x, y) \]

- The vector notation is especially useful in two cases:
  1. When we have many equations \( n >> 1 \) with unified RHSs.
  2. For numerical solution of fundamental systems, since it allows one to develop universal computer codes, which can be applied to systems with arbitrary number of equations.
4.5. Second- and higher order ODEs. Systems of ODEs

Example 4: Mechanical mass-spring system

**Mechanical mass-spring system**

- Displacement: \( y = y(t) \)
- Velocity: \( v = v(t) = dy/dt \)

**Equation of motion: Second-order ODE**

\[
my'' + cy' + ky = r(t)
\]

**Equivalent fundamental two-dimensional system**

\[
\begin{align*}
y_1 &= y \\
y_2 &= y_1' \\
m\dot{y}_2' + cy_2 + ky_1 &= r(t) \\
y_1' &= y_2 \\
y_2' &= (r(t) - cy_2 - ky_1)/m
\end{align*}
\]

**Two-dimensional system written using the matrix notation**

\[
Y(t) = \begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} \quad Y' = F(t, Y) \quad F = \begin{bmatrix} y_2 \\ (r(t) - cy_2 - ky_1)/m \end{bmatrix}
\]

**Initial conditions**

\[
Y(t_0) = Y_0 = \begin{bmatrix} y(t_0) \\ v(t_0) \end{bmatrix}
\]
4.6. Numerical methods for initial value problems (systems of ODEs)

- Numerical methods IVPs for fundamental systems of ODEs
- Runge-Kutta methods of the second order for systems of ODEs
- Oscillations in the mass-spring system. Resonance. Beats

Reading assignment

Gilat 9.4
4.6. Numerical methods for initial value problems (systems of ODEs)

Numerical solution of fundamental systems of ODEs

Let's consider a \( n \)-dimensional fundamental system

\[
\begin{align*}
y_1' &= f_1(x, y_1, y_2, \ldots, y_{j-1}, y_j, y_{j+1}, \ldots, y_n) \\
& \quad \ldots \\
y_n' &= f_n(x, y_1, y_2, \ldots, y_{j-1}, y_j, y_{j+1}, \ldots, y_n)
\end{align*}
\]

(4.6.1)

and rewrite this system using the vector notation:

\[
\begin{align*}
Y' &= F(x, Y) \\
y &= \begin{pmatrix}
y_1 \\
\vdots \\
y_n
\end{pmatrix}, \\
F &= \begin{pmatrix}
f_1 \\
\vdots \\
f_n
\end{pmatrix} = F(x, Y)
\end{align*}
\]

(4.6.2)

Vector equation (4.6.2) is similar to a single first-order equation in the explicit form:

\[
y' = f(x, y)
\]

(4.6.3)

- Any numerical method designed for a single ODE in the explicit form can be reformulated for the system (4.6.1) by changing scalar quantities to vector ones.

- For instance, the Runge-Kutta method of 2nd order can be formulated as follows:

For single equation (4.6.3):

\[
\begin{align*}
k_1 &= \Delta x f(x_i, y_i) \\
k_2 &= \Delta x f(x_i + 0.5 \Delta x, y_i + 0.5 k_1) \\
y_{i+1} &= y_i + k_2 \\
x_{i+1} &= x_i + \Delta x
\end{align*}
\]

For system (4.6.2):

\[
\begin{align*}
K_1 &= \Delta x F(x_i, Y_n) \\
K_2 &= \Delta x F(x_i + 0.5 \Delta x, Y_i + 0.5 K_1) \\
Y_{i+1} &= Y_i + K_2 \\
x_{i+1} &= x_i + \Delta x
\end{align*}
\]
Four steps to solve numerically an IVP for a higher-order ODE or a system of ODES

1. Re-write Equation/System in the explicit form

2. Introduce new functions and transform Equation/System to an equivalent fundamental system

3. Re-write fundamental system in the vector form

4. Apply integrators (Euler, RK2, etc.) developed for individual ODE in the vector form

\[
\begin{align*}
\frac{d^2y_1}{dx^2} - y_2 &= g(x) \\
y_1y_2' &= b \\
y_1'' &= y_2 + g(x) \\
y_2' &= \frac{b}{y_1}
\end{align*}
\]

\[
z_1 = y_1, \quad z_2 = z_1' = y_1', \quad z_3 = y_2 \\
z_1' = z_2 \\
z_2' = z_3 + g(x) \\
z_3' = \frac{b}{z_1}
\]

\[
Z' = F(x, Z) \quad Z = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} \quad F(x, Z) = \begin{pmatrix} z_2 \\ z_3 + g(x) \\ \frac{b}{z_1} \end{pmatrix}
\]

\[
K_1 = \Delta x \ F(x_i, Z_i) \\
K_2 = \Delta x \ F(x_i + 0.5\Delta x, Z_i + 0.5K_1) \\
Z_{i+1} = Z_i + K_2 \\
x_{i+1} = x_i + \Delta x
\]
**4.6. Numerical methods for initial value problems (systems of ODEs)**

**Problem 4.6.1:** Mechanical mass-spring system

**Mechanical mass-spring system**

\[ y = y(t): \quad \text{Displacement} \]
\[ y' = \frac{dy}{dt}: \quad \text{Velocity} \]

- **Second-order ODE**
  \[ my'' + cy' + ky = r(t) \]

- **Harmonic driving force**
  \[ r(t) = F_0 \cos \omega t \]

- **Equivalent two-dimensional system**
  \[ y_1 = y \]
  \[ y_2 = y' \]
  \[ my'_2 + cy_2 + ky_1 = r(t) \]

- **MATLAB implementation:**
  ```matlab
  File RHSMassSpring.m
  function [ F ] = RHSMassSpring ( X, Y )
  M = 2.0; K = 2.0; C = 0.1; Omega = 1.0; F0 = 1.0;
  F(1) = Y(2);
  F(2) = ( F0 * cos( Omega * X ) - C * Y(2) - K * Y(1) ) / M;
  end
  ```
MATLAB implementation of a solver for a fundamental system of ODEs

\[ Y' = F(x, Y) \quad Y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}, \quad F = \begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix} = F(x, Y) \]

In the development of such a solver, we assume that \( Y(x) \) and \( F \) are represented by row vectors \( F \) and \( y \):

\[ y = [y_1, y_2, \ldots, y_n], \quad F = [f_1, f_2, \ldots, f_n] \]

The whole numerical solution is represented by a two-dimensional array \( Y \), where individual columns contain individual functions \( y_i \), in other words \( Y(i, j) = y_j(x_{i+1}) \)

Array \( Y \) has \( N \) rows (number of integration steps + 1) and \( n \) columns (number of unknown functions in the system). \( Y(:,j) \) is the column containing values of the unknown function \( y_j(x) \).
4.6. Numerical methods for initial value problems (systems of ODEs)

Using the vector notation in the MATLAB, any solver developed for a single equation can be easily adopted for an arbitrary fundamental system.

File ODESolverN.m

function [ X, Y ] = ODESolverN ( Integrator, RHS, A, B, YA, DX )
    [ M, N ] = size ( YA ); % N is the dimension of the system if YA is the row vector
    NI = int64 ( ( ( B - A ) / DX ) + 1 );
    X = zeros ( NI, 1 );
    Y = zeros ( NI, N );
    % Initial conditions
    X(1) = A;
    Y(1,:) = YA;
    for i = 1 : NI - 1
        [ X(i+1), Y(i+1,:) ] = Integrator ( X(i), Y(i,:), DX, RHS );
    end

File RK2.m

function [ X, Y ] = RK2 ( X0, Y0, DX, RHS )
    K1 = DX * RHS ( X0, Y0 );
    K2 = DX * RHS ( X0 + 0.5 * DX, Y0 + 0.5 * K1 );
    X = X0 + DX;
    Y = Y0 + K2;
end

File Problem_4_6_1.m

[ X, Y ] = ODESolverN ( @RK2, @RHSMassSpring, 0, 300, [ 1, 1 ], 0.01 );
figure ( 1 );
plot ( X, Y(:,1) , 'r' );
figure ( 2 );
plot ( X, Y(:,2) , 'g' );
Free oscillations \((F_0 = 0)\)

In the case of free oscillation the undamped system \((c = 0)\) experiences harmonic oscillation
\[
y(t) = A \cos(\omega_0 t + \varphi)
\]
with the natural angular frequency of the mass-spring system \(\omega_0 = \sqrt{k/m}\) or period of oscillation \(\tau = 2\pi/\omega_0\). The magnitude and phase of oscillation, \(A\) and \(\varphi\), are determined by the initial conditions.

- The natural frequency is the system property and does not depend on initial conditions.
- The damped system \((c \neq 0)\) can exhibit oscillations with decreasing magnitude (underdamping, weak damping) and non-oscillatory solutions (overdamping, strong damping).

Undamped system

Weak damping \((c^2 < 4m^2 \omega_0^2)\)

Strong damping \((c^2 > 4m^2 \omega_0^2)\)

\[
k = 2, m = 2\pi^2, \tau = 2\pi^2, y(0) = 1, y'(0) = 1
\]
4.6. Numerical methods for IVPs for systems of ODEs

**Forced oscillations** ($F_0 \neq 0$)

In the case of **forced oscillations**, the behavior of the solution drastically depends on the relation between the **input angular frequency** (angular frequency of the driving force) $\omega$ and the **natural angular frequency** of the mass-spring system $\omega_0 = \sqrt{k/m}$.

- **Resonance** in the undamped system ($c = 0$): Excitation of large-magnitude oscillations by matching input and natural frequencies, $\omega_0 = \omega$.
- **Beats** in the undamped system ($c = 0$): Strong temporal modulation of the magnitude of oscillation at $\omega_0 \neq \omega$, but $|\omega_0 - \omega| \ll \omega$.
- **Practical resonance** in the damped system ($c \neq 0$): Maximum amplification of oscillations by the driving force with $\omega^2 = \omega_{max}^2 = \omega_0^2 - c^2/(2m^2)$.

<table>
<thead>
<tr>
<th>Resonance</th>
<th>Beats</th>
<th>Practical resonance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear increase of magnitude with time</td>
<td>Temporal modulation of magnitude</td>
<td>Strong amplification to a finite level</td>
</tr>
</tbody>
</table>

- $k = 2, m = 2, y(0) = 1, y'(0) = 1$
- $\omega = 1$
- $\omega = 0.95$
- $c = 0.1, \omega = \omega_{max}$
4.6. Numerical methods for initial value problems (systems of ODEs)

- Resonance and beats are general phenomena specific for numerous oscillating systems.
- Sometimes resonance is useful, e.g. the resonant amplification is used for registering signals of small magnitudes (radio).
- In engineering applications, resonance and beats are often dangerous phenomena that result in reduced durability and/or catastrophic failures of engineering designs.

Tacoma Narrows bridge collapse (http://en.wikipedia.org/wiki/Tacoma_Narrows_Bridges)
Opening day, July 1, 1940
Collapse, November 7, 1940

- In order to avoid resonance and beats, the oscillating systems should be designed providing large difference between natural frequency(s) of the system and input frequencies of various external forces.
4.7. Numerical solution of IVPs with build-in MATLAB solvers

- MATLAB build-in solvers for systems of ODEs
- Example: Simple pendulum
- Example: Double pendulum

Reading assignment
Gilat 9.4
4.7. Numerical solution of IVPs with build-in MATLAB solvers

MATLAB build-in solvers for systems of ODEs

- Any build-in MATLAB solver for IVPs can be used to solve fundamental systems of ODEs.
- \[ [x, y] = \text{Solver} \left( \text{@RHS, xspan, ya} \right) \]
  - RHS in the function that implements calculation of the RHS \( F(x, y) \), where \( y \) is a vector.
  - xspan is 1D array that should contain at least two real values. The first and last elements of xspan are used as limits of the integration interval \([a, b]\).
  - ya is the initial condition (vector) at \( x = a \).
  - \( x \) is the column vector with nodal values of \( x \) obtained as a result of integration.
  - \( y \) is a 2D vector. Individual columns \( y(:,1), y(:,2), ..., \) correspond to numerical solutions for \( y_1, y_2, ... \)
  - The RHS function should be programmed assuming that the \( F \) is the column vector!

Problem 4.7.1: Solution of ODEs for the mass-spring system with \texttt{ode45} solver:

File \texttt{RHSMassSpring.m}

```matlab
function [F] = MassSpringEq2 (X, Y)
    M = 2.0; K = 2.0; C = 0.1; Omega = 1.0; F0 = 1.0;
    F(1) = Y(2);
    F(2) = (F0 * cos(Omega * X) - C * Y(2) - K * Y(1)) / M;
    F = F';
end
```

File \texttt{Problem_4_7_1.m}

```matlab
[X, Y] = \texttt{ode45} (@RHSMassSpring, [0, 300], [1, 1]);
figure (1); plot (X, Y(:,1), 'r');
figure (2); plot (X, Y(:,2), 'g');
```
4.7. Numerical solution of IVPs with build-in MATLAB solvers

Summary on numerical integration of systems of ODEs with MATLAB function ode45

The MATLAB build-in function `ode45` allows one to solve an initial value problem:
\[
\frac{dy_1}{dx} = f_1(x, y_1, y_2), \quad \frac{dy_2}{dx} = f_2(x, y_1, y_2), \quad y_1(x_0) = y_{10}, \quad y_2(x_0) = y_{20}, \quad x_0 = a \leq x \leq b
\]

Example:
\[
\frac{dy_1}{dx} = x y_1 - y_2, \quad \frac{dy_2}{dx} = -2 y_1,
\]
\[
y_1(0) = 2, \quad y_2(0) = -2, \quad 0 \leq x \leq 4
\]

```matlab
function [ f ] = fun ( x, y )
    f(1) = x * y(1) - y(2);
    f(2) = -2.0 * y(1);
    f = f';
end
```

```
[ x, y ] = ode45 ( @fun, [ 0.0, 4.0 ], [ 2.0, -2.0 ]);
plot ( x, y(:,1), x, y(:,2) ) % y1(x) and y2(x)
figure ( 2 );
plot ( y(:,1), y(:,2) ) % Phase portrait: y2(y1)
```

Name of a function - RHS \( f(x, y) \)
Initial condition
Integration limits

Tables: 1D array of \( x \) and 2D array of \( y \)
Problem 4.7.2: Simple pendulum

Newton's second law of motion

\[ mL \ddot{\theta} = -mg \sin \theta \]

Second-order ODE

\[ \ddot{\theta} = -g \sin \theta /L \]

Fundamental system

\[
\begin{align*}
z_1 &= \theta, z_2 &= \dot{\theta} \\
\dot{z}_1 &= z_2 \\
\dot{z}_2 &= -g \sin z_1 /L
\end{align*}
\]

Initial conditions

\[ z_1(0) = \theta_0 = 85^0, z_2(0) = 0 \]

File RHSPendulum1.m

```matlab
function [ F ] = RHSPendulum1 ( t, Z )
g = 9.81;
L = 0.1;
F(1) = Z(2);
F(2) = - g * sind ( Z(1) ) / L;
F = F';
end
```

File Problem_4_7_2.m

```matlab
L = 0.1;
[t, Z] = ode45 ( @RHSPendulum1, [ 0 : 0.01 : 100 ], [ 85.0, 0 ] );
X = L * sind ( Z(:,1));
Y = L * - cosd ( Z(:,1));

% Trajectory on ( X, Y )
figure ( 1 );
plot ( X, Y, 'r' );
axis equal

% Angle vs. time
figure ( 2 );
plot ( t, Z(:,1), 'g' );

% Phase portrait
figure ( 3 );
plot ( Z(:,1), Z(:,2), 'b' );
```

\[ x = L \sin \theta \]
\[ y = -L \cos \theta \]
4.7. Numerical solution of IVPs with build-in MATLAB solvers

Single pendulum: Numerical solution

\[ \dot{z}_1 = z_2 \]
\[ \dot{z}_2 = -g \sin z_1 / L \]

Initial conditions
\[ z_1(0) = \theta_0 = 85^0, z_2(0) = 0 \]

$L = 0.1 \text{ m}$
Problem 4.7.3: Double pendulum

- Double pendulum demonstrates unpredictable behavior
- This is the simple example of mechanical chaotic system
4.7. Numerical solution of IVPs with build-in MATLAB solvers

Double pendulum: Equations of motion

Newton's second law of motion

\[ m_1 \ddot{x}_1 = -T_1 \sin \theta_1 + T_2 \sin \theta_2 \]
\[ m_1 \ddot{y}_1 = T_1 \cos \theta_1 - T_2 \cos \theta_2 - m_1 g \]

Newton's second law of motion

\[ m_2 \ddot{x}_2 = -T_2 \sin \theta_2 \]
\[ m_2 \ddot{y}_2 = T_2 \cos \theta_2 - m_2 g \]
### 4.7. Numerical solution of IVPs with build-in MATLAB solvers

**Double pendulum: Reduction to a fundamental system (optional slide 1)**

- We obtained 4 equations of motion, Eqs. (1)-(4).
- However, we can reduce these equations to 2 equations, because lengths of strings are constant.
- For this purpose we need to do 2 steps
  1. Exclude from equations of motion string tensions $T_1$ and $T_2$.
  2. Derive $x_1, y_1, x_2, y_2$ in terms of $\theta_1$ and $\theta_2$.
- Then we will get 2 equations with respect to $\theta_1$ and $\theta_2$.

#### Step 1

Add Eq. (1) to Eq. (3) and Eq. (2) to Eq. (4)

\[
\begin{align*}
(1) & \quad m_2 \ddot{x}_2 = -T_2 \sin \theta_2 \\
(2) & \quad m_2 \ddot{y}_2 = T_2 \cos \theta_2 - m_2 g \\
(3) & \quad m_1 \ddot{x}_1 = -T_1 \sin \theta_1 + T_2 \sin \theta_2 \\
(4) & \quad m_1 \ddot{y}_1 = T_1 \cos \theta_1 - T_2 \cos \theta_2 - m_1 g
\end{align*}
\]

\[
\begin{align*}
\text{Add Eq. (1) to Eq. (3) and Eq. (2) to Eq. (4)} & \quad m_1 \ddot{x}_1 + m_2 \ddot{x}_2 = -T_1 \sin \theta_1 \\
& \quad m_1 \ddot{y}_1 + m_2 \ddot{y}_2 = T_1 \cos \theta_1 - (m_1 + m_2)g
\end{align*}
\]

\[
\begin{align*}
\cos \theta_1 (m_1 \ddot{x}_1 + m_2 \ddot{x}_2) & \quad \times \cos \theta_1 \\
& \quad + \sin \theta_1 (m_1 \ddot{y}_1 + m_2 \ddot{y}_2) \\
& \quad = -g(m_1 + m_2) \sin \theta_1
\end{align*}
\]
4.7. Numerical solution of IVPs with build-in MATLAB solvers

Double pendulum: Reduction to a fundamental system (optional slide 2)

\[
\cos \theta_1 \left( m_1 \ddot{x}_1 + m_2 \ddot{x}_2 \right) + \sin \theta_1 \left( m_1 \ddot{y}_1 + m_2 \ddot{y}_2 \right) = -g \left( m_1 + m_2 \right) \sin \theta_1 \quad (5)
\]

Step 2

\[
\cos \theta_2 \ddot{x}_2 + \sin \theta_2 \ddot{y}_2 = -g \sin \theta_2 \quad (6)
\]

- We need to connect Cartesian coordinates of bob to angles \( \theta_1 \) and \( \theta_2 \)
  
  \[
  x_1 = L_1 \sin \theta_1 \quad (7)
  \]
  
  \[
  y_1 = L_2 + L_1 - L_1 \cos \theta_1 \quad (8)
  \]
  
  \[
  x_2 = L_1 \sin \theta_1 + L_2 \sin \theta_2 \quad (9)
  \]
  
  \[
  y_2 = L_2 + L_1 - L_1 \cos \theta_1 - L_2 \cos \theta_2 \quad (10)
  \]

- After inserting Eqs. (7)-(10) into Eqs. (5) and (6) we get equations of motion in terms of angles \( \theta_1 \) and \( \theta_2 \).

  Example: \( \ddot{x}_1 = -L_1 \left( \dot{\theta}_1 \right)^2 \sin \theta_1 + L_1 \ddot{\theta}_1 \sin \theta_1 \).

Equations of motion of double pendulum

\[
L_1 \ddot{\theta}_1 + \mu_2 L_2 \cos(\theta_1 - \theta_2) \ddot{\theta}_2 = -g \sin \theta_1 - \mu_2 L_2 \sin(\theta_1 - \theta_2) \dot{\theta}_2^2 \quad (11)
\]

\[
L_1 \cos(\theta_1 - \theta_2) \ddot{\theta}_1 + L_2 \ddot{\theta}_2 = -g \sin \theta_2 + L_1 \sin(\theta_1 - \theta_2) \dot{\theta}_1^2 \quad (12)
\]

Here \( \mu_2 = \frac{m_2}{m_2 + m_1} \) is the reduced mass of the second bob.
4.7. Numerical solution of IVPs with build-in MATLAB solvers

Double pendulum: Reduction to a fundamental system (optional slide 3)

Reduction of equations of motion to explicit form

\[ L_1 \ddot{\theta}_1 + \mu_2 L_2 \cos(\theta_1 - \theta_2) \ddot{\theta}_2 = -g \sin \theta_1 - \mu_2 L_2 \sin(\theta_1 - \theta_2) \dot{\theta}_2^2 \]

\[ L_1 \cos(\theta_1 - \theta_2) \ddot{\theta}_1 + L_2 \ddot{\theta}_2 = -g \sin \theta_2 + L_1 \sin(\theta_1 - \theta_2) \dot{\theta}_1^2 \]

These equations can be re-written in the matrix form as follows:

\[
\begin{pmatrix}
\ddot{\theta}_1 \\
\ddot{\theta}_2
\end{pmatrix}
= \begin{pmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{pmatrix}
\begin{pmatrix}
\dot{\theta}_1 \\
\dot{\theta}_2
\end{pmatrix}
= \begin{pmatrix}
L_1 & \mu_2 L_2 \cos(\theta_1 - \theta_2) \\
L_1 \cos(\theta_1 - \theta_2) & L_2
\end{pmatrix}
\begin{pmatrix}
\dot{\theta}_1 \\
\dot{\theta}_2
\end{pmatrix}
\]

\[
\begin{pmatrix}
B_1 \\
B_2
\end{pmatrix}
= \begin{pmatrix}
-g \sin \theta_1 - \mu_2 L_2 \sin(\theta_1 - \theta_2) \dot{\theta}_2^2 \\
-g \sin \theta_2 + L_1 \sin(\theta_1 - \theta_2) \dot{\theta}_1^2
\end{pmatrix}
\]

SLE given by Eq. (13) can be solved as \( \begin{pmatrix}
\ddot{\theta}_1 \\
\ddot{\theta}_2
\end{pmatrix}
= A^{-1}B \), where \( A^{-1} \) is calculated with Kramer’s rule:

\[
\begin{align*}
\ddot{\theta}_1 &= \frac{A_{22} B_1 - A_{12} B_2}{A_{11} A_{22} - A_{12} A_{21}}, & \ddot{\theta}_2 &= \frac{-A_{21} B_1 + A_{11} B_2}{A_{11} A_{22} - A_{12} A_{21}}
\end{align*}
\]
4.7. Numerical solution of IVPs with build-in MATLAB solvers

Double pendulum: Equations of motion in a forma of a fundamental system

\[
\begin{align*}
\ddot{\theta}_1 &= \frac{A_{22}B_1 - A_{12}B_2}{A_{11}A_{22} - A_{12}A_{21}} \\
\ddot{\theta}_2 &= \frac{-A_{21}B_1 + A_{11}B_2}{A_{11}A_{22} - A_{12}A_{21}} \\
\end{align*}
\]

\[
\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} L_1 & \mu_2 L_2 \cos(\theta_1 - \theta_2) \\ L_1 \cos(\theta_1 - \theta_2) & L_2 \end{pmatrix}
\]

\[
\begin{pmatrix} B_1 \\ B_2 \end{pmatrix} = \begin{pmatrix} -g \sin \theta_1 - \mu_2 L_2 \sin(\theta_1 - \theta_2) \dot{\theta}_2^2 \\ -g \sin \theta_2 + L_1 \sin(\theta_1 - \theta_2) \dot{\theta}_1^2 \end{pmatrix}
\]

\[
\begin{align*}
z_1 &= \theta_1, & z_2 &= \dot{\theta}_1, & z_3 &= \theta_2, & z_4 &= \dot{\theta}_2
\end{align*}
\]

\[
\begin{align*}
\dot{z}_1 &= z_1 \\
\dot{z}_2 &= \frac{A_{22}B_1 - A_{12}B_2}{A_{11}A_{22} - A_{12}A_{21}} \\
\dot{z}_3 &= z_4 \\
\dot{z}_4 &= \frac{-A_{21}B_1 + A_{11}B_2}{A_{11}A_{22} - A_{12}A_{21}} \\
\end{align*}
\]

\[
\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} L_1 & \mu_2 L_2 \cos(z_1 - z_3) \\ L_1 \cos(z_1 - z_3) & L_2 \end{pmatrix}
\]

\[
\begin{pmatrix} B_1 \\ B_2 \end{pmatrix} = \begin{pmatrix} -g \sin z_1 - \mu_2 L_2 \sin(z_1 - z_3) z_4^2 \\ -g \sin z_3 + L_1 \sin(z_1 - z_3) z_2^2 \end{pmatrix}
\]
4.7. Numerical solution of IVPs with build-in MATLAB solvers

Double pendulum: MATLAB function for RHS

File RHSPendulum2.m

function [ F ] = Pendulum2RHS ( t, Z )

g = 9.81;
L1 = 0.1;
L2 = 0.1;
M1 = 1.0;
M2 = 1.0;
Mu2 = M2 / ( M1 + M2 );
C13 = cos ( Z(1) - Z(3) );
S13 = sin ( Z(1) - Z(3) );
A = [ L1, ( Mu2 * L2 * C13 ); ( L1 * C13 ), L2 ];
B = [ ( - g * sin ( Z(1) ) - Mu2 * L2 * S13 * Z(4)^2 ) ; ( - g * sin ( Z(3) ) + L1 * S13 * Z(2)^2 ) ];
FF = inv ( A ) * B;
F(1) = Z(2);
F(2) = FF(1);
F(3) = Z(4);
F(4) = FF(2);
F = F';
end
4.7. Numerical solution of IVPs with built-in MATLAB solvers

**Double pendulum: MATLAB script**

```matlab
File Problem_4_7_3.m
% Here we set lengths of strings
L1 = 0.1;
L2 = 0.1;

% Here we use ode45 solver to obtain angles
[t, Z] = ode45 (@RHSPendulum2, [ 0 : 0.01 : 20 ], [ ( degtorad ( 90.0 ) ), 0, ( degtorad ( 90.0 ) ), 0 ]);%

% Now we have Z solution of equations of motions:
% Z(1) = Theta_1
% Z(2) = d Theta_1 / d t
% Z(3) = Theta_2
% Z(4) = d Theta_2 / d t

% Here we convert angles into Cartesian coordinates of bob 1
X1 = L1 * sin ( Z(:,1) );
Y1 = L1 + L2 - L1 * cos ( Z(:,1) );

% Here we convert angles into Cartesian coordinates of bob 2
X2 = L1 * sin ( Z(:,1) ) + L2 * sin ( Z(:,3) );
Y2 = L1 + L2 - L1 * cos ( Z(:,1) ) - L2 * cos ( Z(:,3) );

% Now we can choose one of two ways to visualize solutions
% Script Pendulum2Plots prepare plots
% Pendulum2Plots
% Script Pendulum2Animation animates the motion of pendulum on the plane (X,Y)
Pendulum2Animation

Initial conditions:
\[ \theta_1(0) = \theta_2(0) = 90^0, \dot{\theta}_1(0) = \dot{\theta}_2(0) = 0 \]

Eq. (7)-1(10):
\[ x_1 = L_1 \sin \theta_1 \]
\[ y_1 = L_2 + L_1 - L_1 \cos \theta_1 \]
\[ x_2 = L_1 \sin \theta_1 + L_2 \sin \theta_2 \]
\[ y_2 = L_2 + L_1 - L_1 \cos \theta_1 - -L_2 \cos \theta_2 \]
```
4.7. Numerical solution of IVPs with build-in MATLAB solvers

Double pendulum: Numerical solution

\[ m_1 = m_2 = 1 \text{ kg}, \quad L_1 = L_2 = 10 \text{ cm} \]

Initial conditions:
\[ \theta_1(0) = \theta_2(0) = 90^0 \quad \dot{\theta}_1(0) = \dot{\theta}_2(0) = 0 \]
4.8. Summary

For the exam we must know how

- To calculate numerically the first derivatives with forward, backward, and central differences of the first and second order.
- To calculate numerically the second derivatives with central-difference equation.
- To implement calculations of the above derivatives in the MATLAB code.
- To numerically integrate a function with the rectangle and trapezoidal rules.
- To implement the rectangle and trapezoidal rules in the MATLAB code.
- To numerically integrate a function with the MATLAB `quad` functions.
- To use Euler, RK2, and RK4 methods for numerical solution of individual first-order ODEs.
- To use `ode45` solver for numerical solution of individual first-order ODEs.
- To use the Euler, RK2, and RK4 methods for numerical solution of fundamental systems of ODEs.
- To use `ode45` solver for numerical solution of fundamental systems of ODEs.
- To re-write a higher-order ODE or a system in the form of a fundamental system of ODEs using the matrix notation,
- To formulate and solve numerically the equations of the mechanical mass-spring system.
- To know conditions, when resonance, beats, and practical resonance occur in the mass-spring system.
- To formulate and solve numerically the equations of the single pendulum.