Chapter 4
Numerical methods for ODEs

4.1. Numerical methods for solution of IVP for ODEs. Basic concepts
4.2. Euler method
4.3. Convergence, approximation, and stability
4.4. Methods of higher orders of approximation
4.5. Runge-Kutta methods
4.6. Numerical solution of IVP for systems of ODEs
4.7. Explicit, implicit and predictor-corrector methods
4.8. Linear multistep methods (LMMs). Adams family of LMMs (optional)

Reading:
Kreyszig, *Advanced Engineering Mathematics, 10th Ed.,* 2011
Selection from chapter 21

Prerequisites:
Kreyszig, *Advanced Engineering Mathematics, 10th Ed.,* 2011
Interpolation: Section 19.3

Topics for self-studying: 4.7

4.1. Numerical methods for solution of IVPs for ODEs. Basic concepts

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

$$y' = f(x, y) \tag{4.1.1}$$

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 Eq. (4.1.1).

On the other hand, virtually any Eq. (4.1.1) can be solved **numerically** with the help of a computer that performs basic algebraic operations ($\pm, \times, \div$) with integer and real numbers.

In order to solve Eq. (4.1.1) 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 Eq. (4.1.1), we can solve numerically only the Cauchy problem with the initial condition

$$y' = f(x, y), \qquad y(a) = y_0, \qquad a \le x \le b \tag{4.1.2}$$

4.1. Numerical methods for solution of IVPs for ODEs. Basic concepts

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, …, etc. of x, where $x_{n+1} > x_n$. In other words, we are looking for a solution represented in the tabulated form:

x	$x_0 = a$	x_1	…	x_i	…	$x_N = b$
y	y_0	y_1	…	y_i	…	y_N

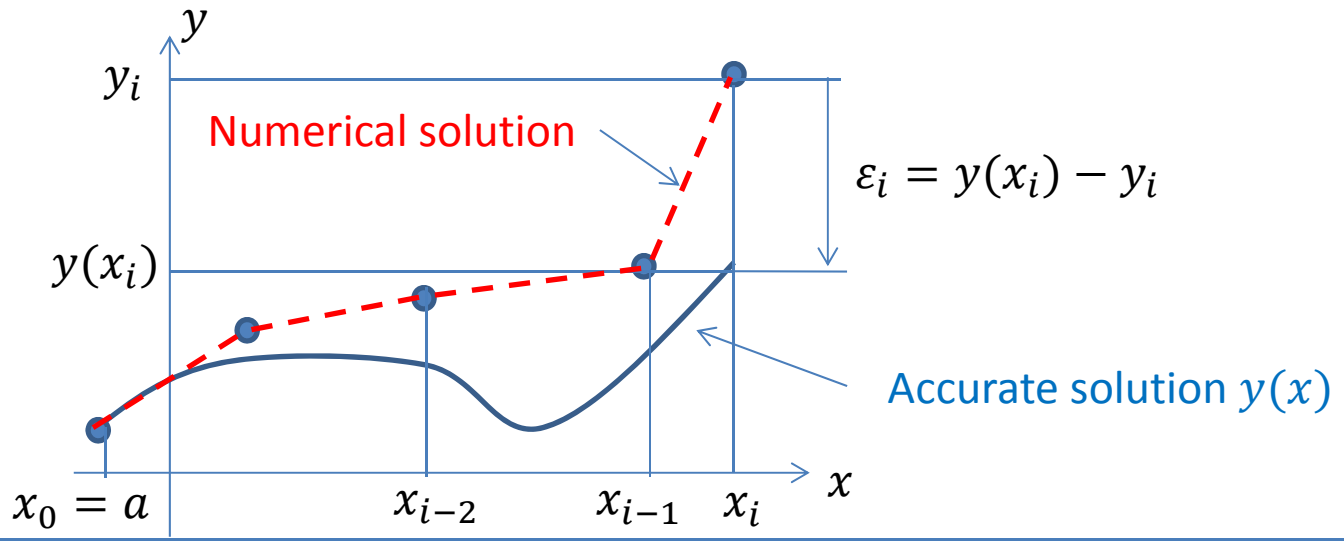
(4.1.3)

This table is called the **numerical solution** of problem (4.1.2).

3. In general, the numerical solution is **approximate** in a sense that $y(x_i) \neq y_i$ (Here $y(x)$ is the presumed accurate solution of the Cauchy problem (4.1.2)). The difference

$$\varepsilon_i = y(x_i) - y_i \tag{4.1.4}$$

is called the numerical (or **global truncation**) **error of the numerical solution** in point x_i.



4.2. Euler method

The initial value problem (IVP)

$$\frac{dy}{dx} = f(x, y), \quad y(a) = y_0, \quad a \le x \le b \tag{4.2.1}$$

The Euler method of numerical integration for the IVP

1. Let's introduce a mesh of nodes x_n with the spacing Δx: $x_{i+1} = x_i + \Delta x$.
2. Let's approximate ODE in the IVP (4.2.1) in node x_i with forward difference of the first order of approximation

$$\frac{y_{i+1} - y_i}{\Delta x} = f(x_i, y_i) \tag{4.2.2}$$

3. We start from the point $x = x_0 = a$ and $y = y_0$ given by the initial condition and apply Eq. (4.2.2) in order to find y_1:

$$x_1 = x_0 + \Delta x, \qquad y_1 = y_0 + f(x_0, y_0)\Delta x$$

3. Now we can repeat step 2 recursively

$$\boxed{x_{i+1} = x_i + \Delta x, \qquad y_{i+1} = y_i + f(x_i, y_i)\Delta x} \tag{4.2.3}$$

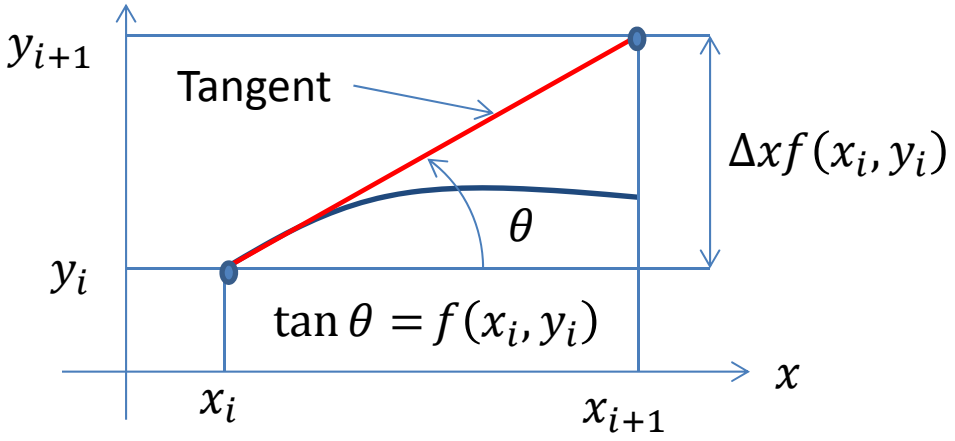
until $x_{i+1} = x_0 + (i+1)\Delta x > b$.

The numerical method given by Eqs. (4.2.3) is called the **(explicit) Euler method**

4.2. Euler method

Graphically, the one integration step of the numerical algorithm according to the Euler method can be shown as follows:

We follow to the tangent in point (x_i, y_i) in order to obtain y_{i+1}.



Eq. (4.2.2)

$$\frac{y_{i+1} - y_i}{\Delta x} = f(x_i, y_i) = f_i$$

resembles the initial ODE in (4.2.1), since according to the definition of the derivative,

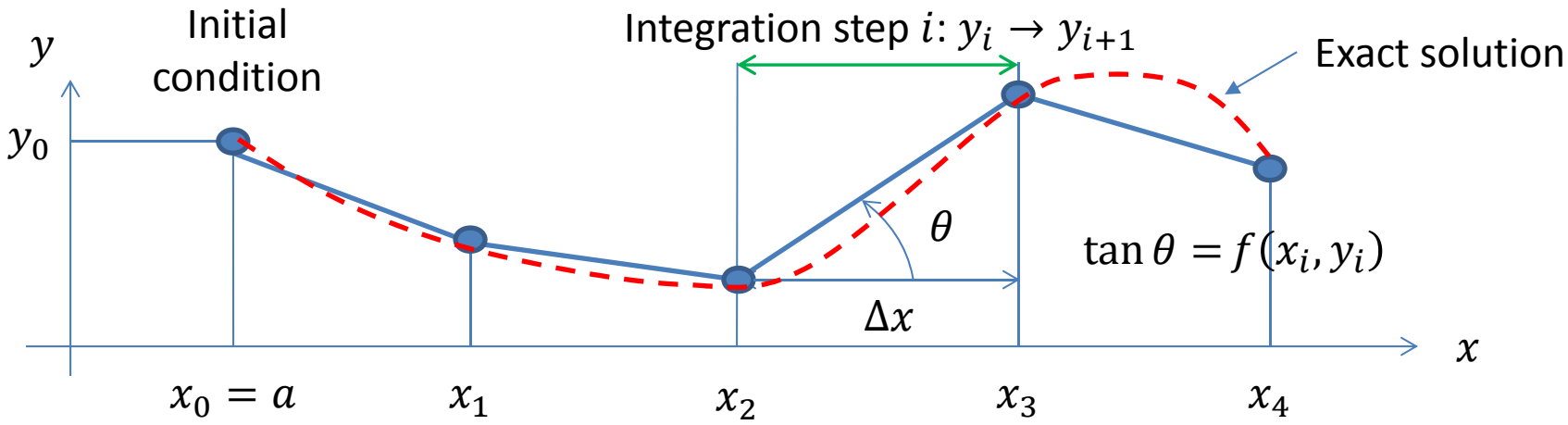
$$\lim_{\Delta x \to 0} \frac{y(x_i + \Delta x) - y(x_i)}{\Delta x} = y'(x_i)$$

The value $\Delta y(x_i) = y(x_i + \Delta x) - y(x_i)$ is called the **finite difference** (as an antipode of the *infinitesimal* difference $dy = y'dx$) and *algebraic* equations formulated in terms of finite differences are called the **finite difference equations**.

- Finite differences are used to approximate individual derivatives.
- Finite difference equation replaces a *differential* equation with an *algebraic* equation.

4.2. Euler method

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

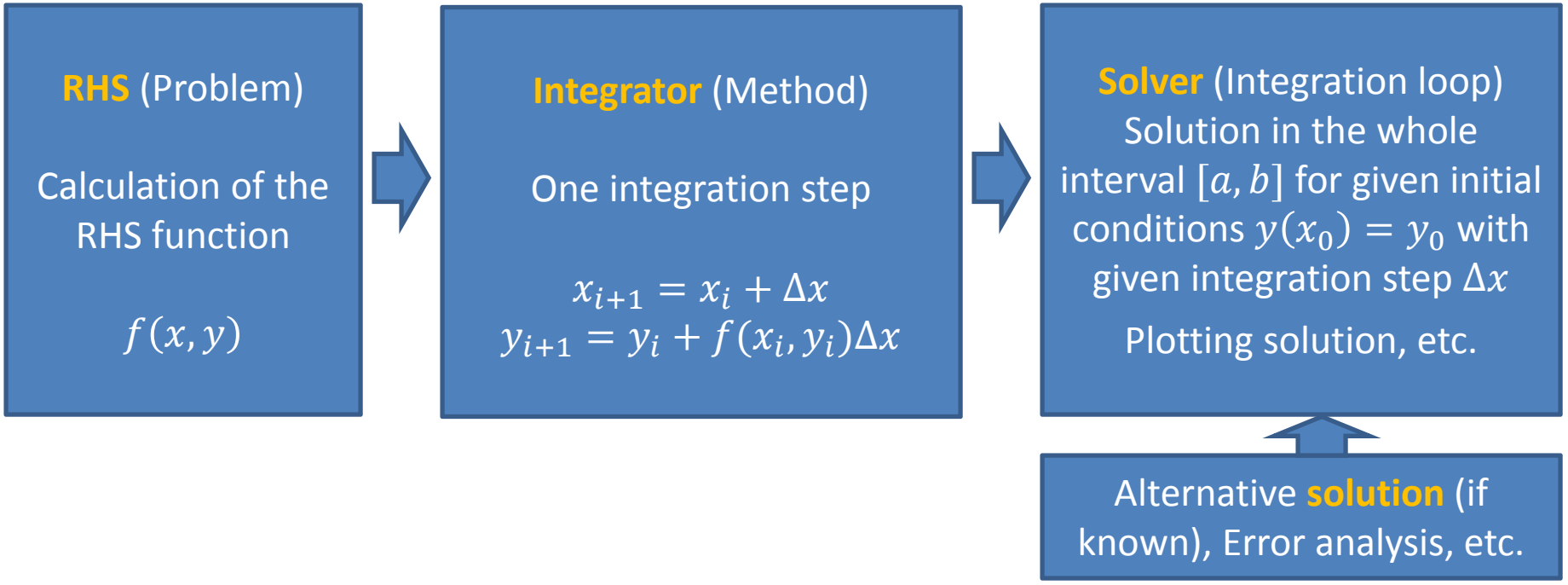


- Value $\Delta x_n = x_n - x_{n-1}$ (increment of the independent variable during one integration step) is called the **integration step size**. In our case $\Delta x_i = \Delta x = const$, but in general the integration step does not need to be constant at every n.
- Any numerical method (finite difference equation) is an *algebraic* equation (formulated in terms of numbers) and, thus, its properties are different from the properties of the original *differential* equation (formulated in terms of functions).
- 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** ε_i.
- 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.

4.2. Euler method

Implementation of the typical IVP solver

$$y' = f(x, y), \qquad y(x_0) = y_0, \qquad x_0 = a \le x \le b$$

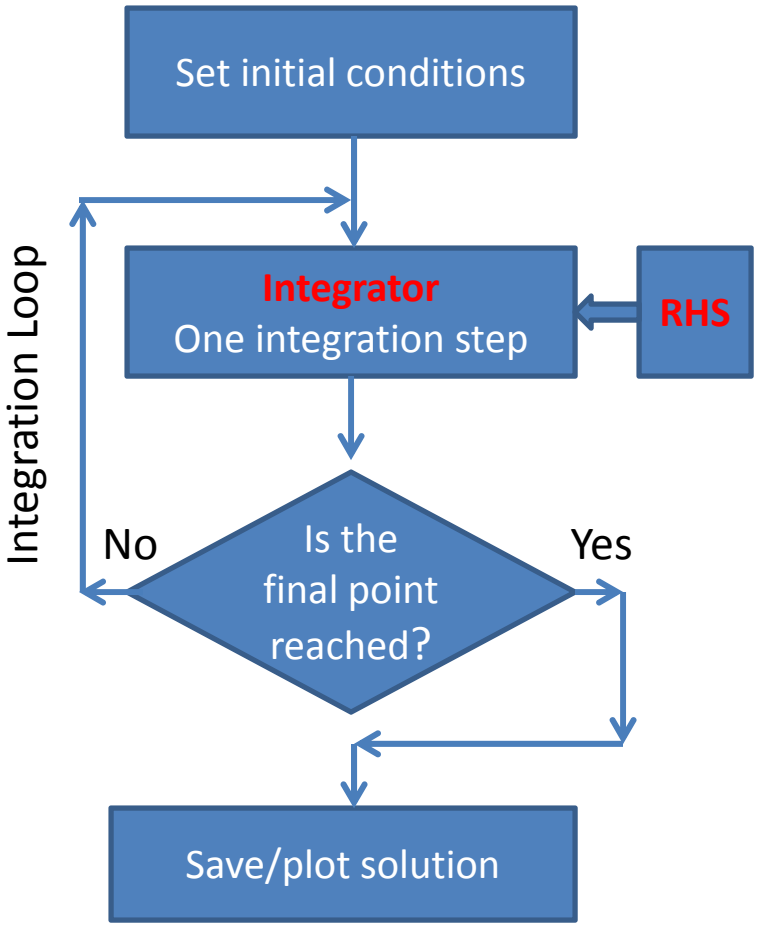


- The RHS, Integrator, and Solver are usually implemented in the form of separate functions.
- It allows one to use the same Integrator and Solver to solve different IVPs or use different Integrators to solve the same IVP.

4.2. Euler method

Flowchart of a typical numerical solver for IVP

Flowchart



Euler method

$$x_0 = a$$
$$y_0 = y_a$$
$$i = 0$$

$$RHS = f(x_i, y_i)$$
$$x_{i+1} = x_i + \Delta x,$$
$$y_{i+1} = y_i + \Delta x RHS$$
$$i = i + 1$$

$$x_n > b\ ?$$

MATLAB Code

```
X(1) = A;
Y(1) = YA;
i=0;

while
  [X(i+1),Y(i+1)] = Int(X(i),Y(i),DX,RHS);
  i=i+1;


  if ( X(i) > B )
     break;
  end

end

plot ( X, Y );
```

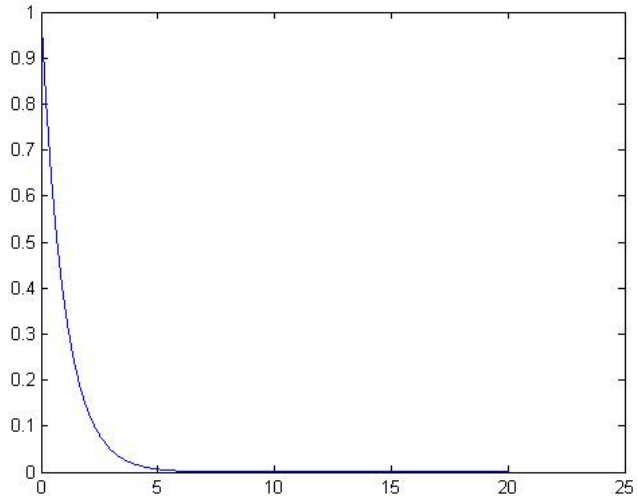
4.2. Euler method

Example: Solve the IVP for the radioactive decay problem with the Euler method

$$y' = -y, \quad y(0) = 1, \quad 0 \le x < 20.$$

File ODESolver1.m

```
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
end
```



File DecayProblem_Euler.m:

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

File RHSDecay.m

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

File Euler.m

```
function [X,Y] = Euler ( X0, Y0, DX, RHS )
      F = RHS ( X0, Y0 );
      Y = Y0 + DX * F;
      X = X0 + DX;
end
```

4.2. Euler method

This script allows us to compare the numerical and exact solutions and calculate the global truncation error

File DecayProblem_Euler_Err.m

```
[ X, Y ] = ODESolver1 ( @Euler, @RHSDecay, 0.0, 10.0, 2.0, 0.1 );
Yexact = SolDecay ( X, 0.0, 2.0 );
E = abs ( ( Yexact- Y ) ./ Yexact ); % Error
figure ( 1 ); % First figure is the solution
plot ( X, Y, 'r-', X, Yexact, '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');
```

Relative numerical error

$$\bar{\varepsilon}_i = \left| \frac{y(x_i) - y_i}{y(x_i)} \right|$$

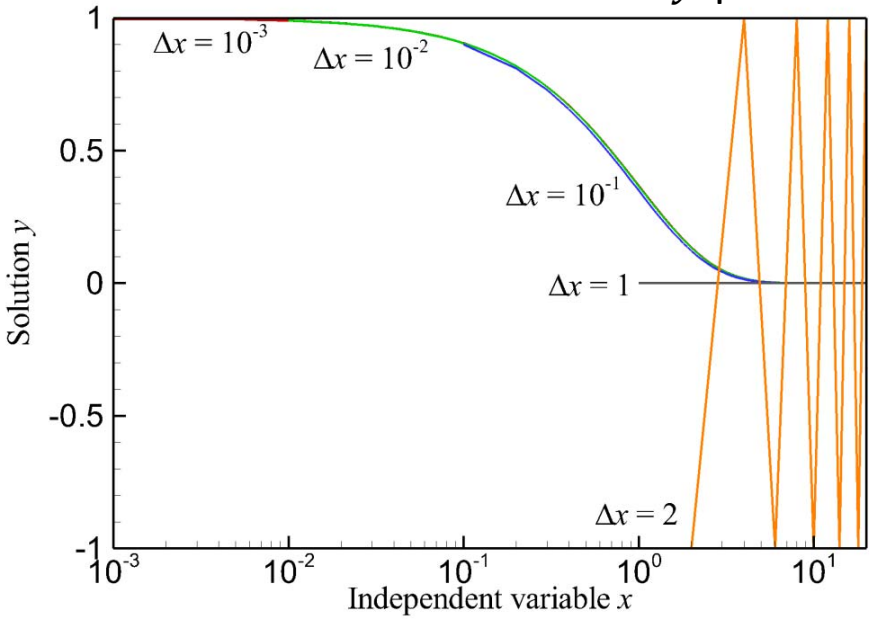
File SolDecay.m

```
function Y = SolDecay ( X, A, YA )
        Y = YA * exp ( A - X );
end
```

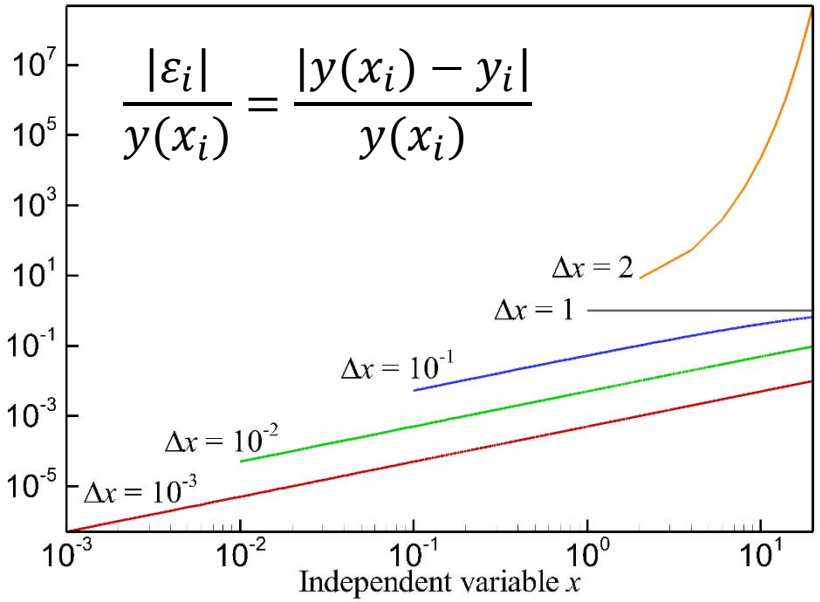
4.2. Euler method

Problem: $y' = -y, \;\; y(0) = 1, \;\; 0 \le x < 20$. Accurate solution: $y(x) = \exp(-x)$.

Numerical solution at the xy-plane



Relative numerical error $|\varepsilon_i / y(x_i)|$ vs. x_i



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 Δx.
4. There is a some "critical value" Δx_{crit}. if $\Delta x \ge \Delta x_{crit}$, then the numerical solution has no physical sense and useless.

Questions:

1. Why is the numerical error proportional to Δx?
2. Why does the "critical value" Δx_{crit} exist?

4.3. Convergence, approximation, and stability

Local and global truncation errors

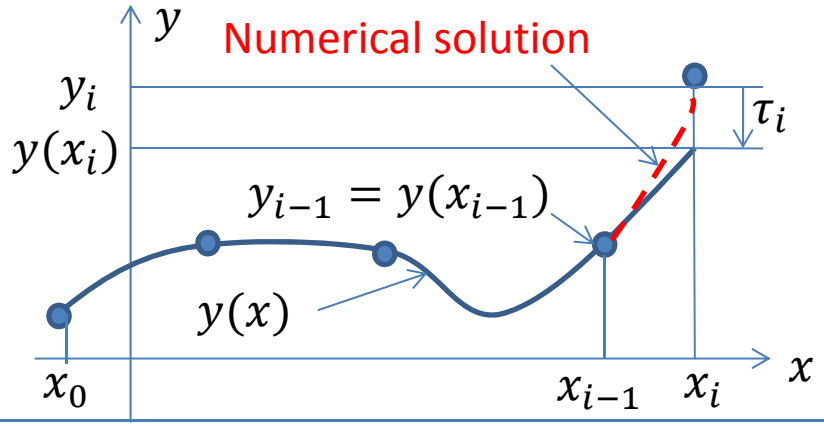
The answers on the questions formulated in the end of section 4.2. can be obtained by studying the convergence, approximation, and stability of solutions of finite-difference equations.

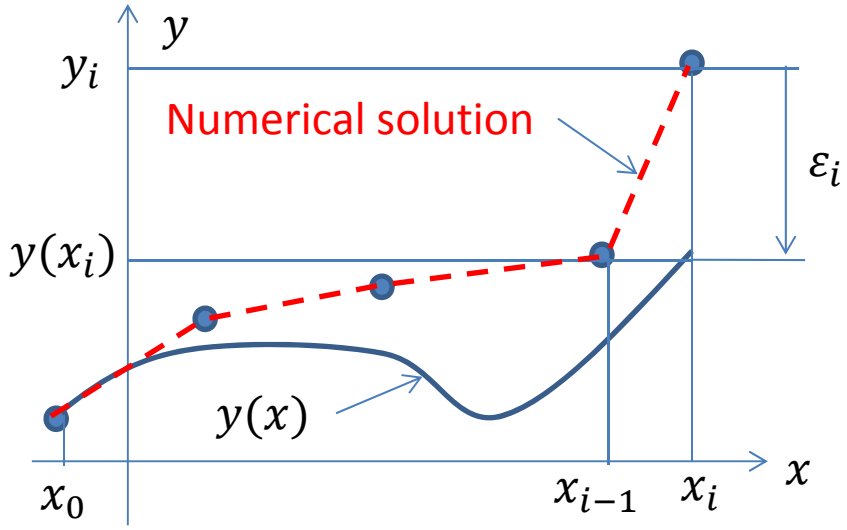
- We say that the numerical solution y_i **converges** to the solution $y(x)$ of the original ODE if the numerical (**global truncation**) error $\varepsilon_i = y(x_i) - y_i \to 0$ as $\Delta x \to 0$.
- **Local truncation error** of the numerical method is the error

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

calculated *assuming that all previous numerical values are accurate*, i.e. $y_j = y(x_j)$ for $j < i$.

- *Local truncation error characterizes the error we make at one integration step*. Total numerical error ε_i usually increases with the number of the integration steps i and can be much larger than τ_i.





4.3. Convergence, approximation, and stability

Order of approximation

- We say that the numerical method (finite difference equation) **approximates** ODE if $\tau_i \to 0$ as $\Delta x \to 0$. If $\tau_i = O(\Delta x^{k+1})$ then we say that the method has k^{th} **order of approximation**.
- Why we say that a method has k^{th} order of approximation if $\tau_i = O(\Delta x^{k+1})$?
  Let's estimate the global error at the end of the interval, i.e. after $N = [(b-a)/\Delta x]$ integration steps. At every step we introduce error $\tau_i = O(\Delta x^{k+1})$, then after N steps

$$\varepsilon_N \sim N\tau_i = \frac{b-a}{\Delta x} O(\Delta x^{k+1}) \sim O(\Delta x^k)$$

  Thus, the method of k^{th} order has the global error $O(\Delta x^k)$.
- **The order of approximation is the simple measure of accuracy** (the larger order of approximation, the better accuracy).
- The general approach to study the approximation is to take the Taylor series of $y(x_{i+1})$ in the point $x = x_i$. Euler method has 1st order of approximation :

$$y(x_{i+1}) = y(x_i + \Delta x) = y(x_i) + y'(x_i)\Delta x + \frac{1}{2}y''(x_i)\Delta x^2 + O(\Delta x^3)$$

$$y_{i+1} = y_i + f(x_i, y_i)\Delta x$$

$$\tau_{i+1} = y(x_{i+1}) - y_{i+1} = \frac{1}{2}y''(x_i)\Delta x^2 + O(\Delta x^3) = O(\Delta x^2)$$

The approximation is not sufficient for the convergence.

4.3. Convergence, approximation, and stability

Stability

Now let's consider how the error evolves with $n \to \infty$ if previous data are also erroneous.

Let's consider the **model equation**

$$y' = -\lambda y, \qquad \lambda = const > 0$$

For $f(x, y) = -\lambda y$, the Euler method results in

$$y_{i+1} = y_i - \lambda y_i \Delta x = (1 - \lambda \Delta x) y_i = (1 - \lambda \Delta x)^2 y_{i-1} = (1 - \lambda \Delta x)^{i+1} y_0$$

We see that $y_i \to 0$ at $n \to \infty$ (only such solutions have physical meaning) if $|1 - \lambda \Delta x| < 1$ or $0 < \lambda \Delta x < 2$.

If $y_i \to 0$ at $i \to \infty$ for our model equation, then we say that the numerical method (scheme) is **stable**, otherwise **unstable**. **Unstable methods are practically useless**.

- The Euler method is **conditionally stable**, i.e. it is stable only in the finite **stability range** $0 < \lambda \Delta x < 2$. This is in agreement with results of our numerical experiments.
- The stability a numerical method depends on properties of the ODE (λ in our case), so that Δx available for numerical integration depends on the problem under consideration.

There is a **theorem** that roughly reads that

If a numerical method approximates the initial ODE and the method is stable, then the numerical solution converges, so that **the approximation and stability imply convergence**.

4.4. Methods of higher orders of approximation

- The Euler method is not suitable for practical calculations, since the numerical error increases fast with the number of integration steps.
- In practical calculations, methods of al least 2nd order of approximation are used, but methods of the 4th are applied in the majority of applications.
- Some applications, e.g., **molecular dynamics (MD) simulations**, require methods with increased accuracy, typically, methods of 6th-8th order of approximation.

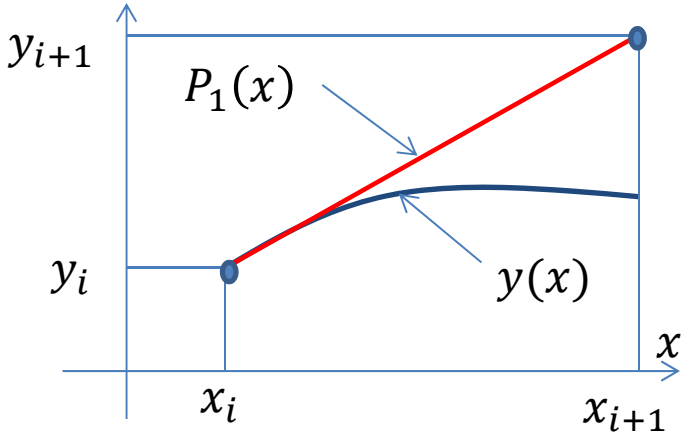
Question: How can we improve the accuracy (increase order of approximation)?

Let's consider again the Euler method (see slide 11)

$$y_{i+1} = y_i + f_i \Delta x = y_i + f_i (x_i - x_{i-1})$$

$$y(x_{i+1}) = y(x_i) + y'(x_i)\Delta x + \frac{1}{2} y''(x_i)\Delta x^2 + O(\Delta x^3)$$

$$\tau_{i+1} = y(x_{i+1}) - y_{i+1} = \frac{1}{2} y''(x_i)\Delta x^2 + O(\Delta x^3) = O(\Delta x^2)$$



- Euler method means that we approximate solution at an integration step by a linear function – linear **interpolation polynomial**

$$y(x) \approx y_i + f_i\,(x - x_i)$$

- If solution is a linear function (first-order polynomial)

$$y(x) = a + b(x - x_i) = P_1(x)$$

then the method is accurate (all derivatives of order 2 and higher are equal to zero).

4.4. Methods of higher orders of approximation

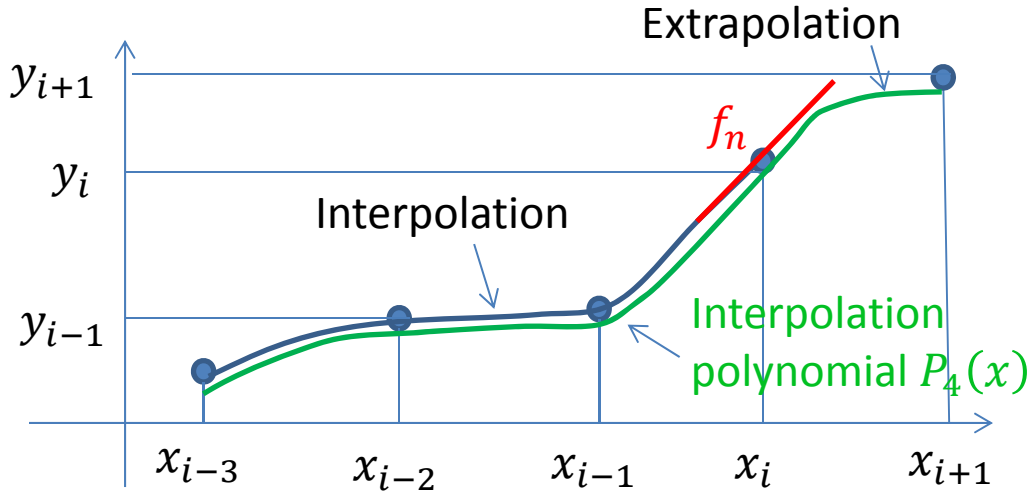
The general approach for development of numerical methods with higher order of approximation is *to approximate $y(x)$ within the integration step size with the interpolation polynomial $P_n(x)$ of degree n*. One can show then that

- The obtained numerical method has n^{th} order of approximation.
- The method is accurate (zero numerical error) if the solution is a polynomial of degree n.

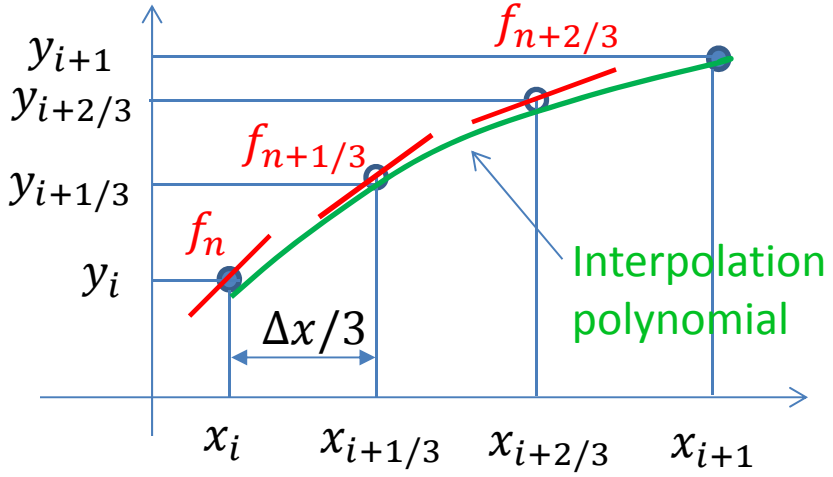
There are many ways how the interpolation polynomial of degree n can be introduced. Two most important practical approaches are

- To use information about solution at previous integration steps, $y_{i-1}, f_{i-1}, y_{i-2}, f_{i-2}$, …(**Linear multistep methods, LMM**).
- To define additional values of y and/or f within the integration step (**Runge-Kutta methods, RK**).

Linear multistep methods



Runge-Kutta methods



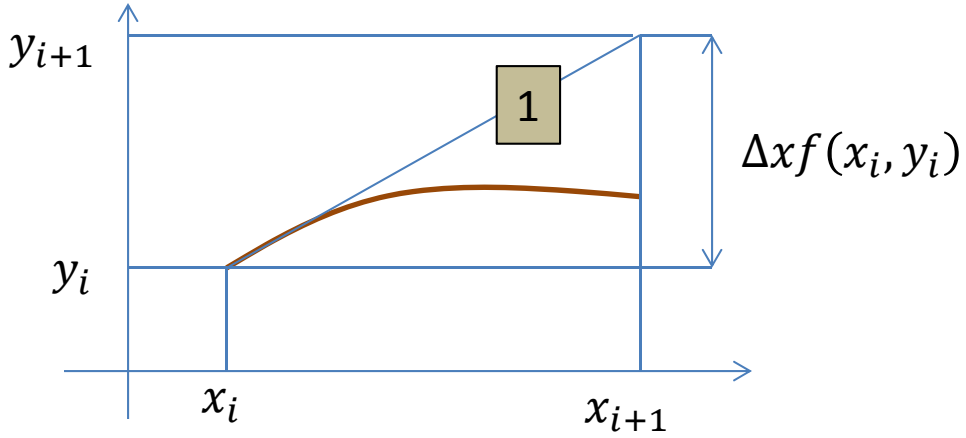
4.5. Runge-Kutta methods

We are going to develop numerical methods for a 1st order ODE in the explicit form

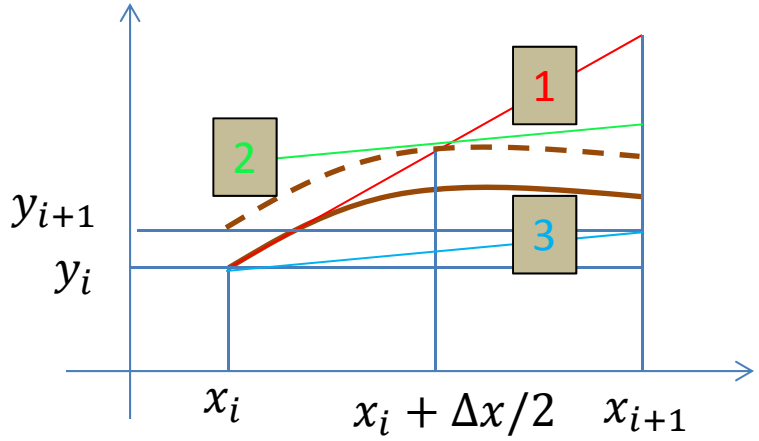
$$y' = f(x, y) \tag{4.5.1}$$

The idea of the **Runge-Kutta methods** is to use additional values of the RHS at the interval $x_n \le x \le x_{n+1}$ in order to increase the order of approximation. Let's consider how in can be done in the case of the 2nd order method.

1st order Euler method



2nd order Runge-Kutta



The **Runge-Kutta method of the 2nd order** (RK2) (**improved Euler method**) can be formulated as:

$$k_1 = \Delta x f(x_i, y_i)$$

$$k_2 = \Delta x f(x_i + 0.5\Delta x, y_i + 0.5k_1) \tag{4.5.2}$$

$$y_{i+1} = y_i + k_2, \quad x_{i+1} = x_i + \Delta x$$

4.5. Runge-Kutta methods

Note: The order of approximation of the method given by Eqs. (4.5.2) can be found by taking the Taylor expansions of both accurate and numerical solutions.

Implementation of the RK2 method in the MATLAB

Integrator: 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
```

In the MATLAB Command window:

```
ODESolver1 ( @RK2, @RHSDecay, 0.0, 10.0, 2.0, 0.1 );
```

4.5. Runge-Kutta methods

The development of the RK methods of higher orders requires complex algebra.

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

$$
\begin{aligned}
k_1 &= \Delta x f(x_i, y_i) \\
k_2 &= \Delta x f(x_i + 0.5\Delta x, y_i + 0.5k_1) \\
k_3 &= \Delta x f(x_i + 0.5\Delta x, y_i + 0.5k_2) \\
k_4 &= \Delta x f(x_i + \Delta x, y_i + k_3) \\
y_{i+1} &= y_i + \frac{1}{6}(k_2 + 2k_2 + 2k_3 + k_4) \\
x_{i+1} &= x_i + \Delta x
\end{aligned}
\tag{4.5.3}
$$

Notes:

1. RK methods do not require any information about the numerical solution in previous points. Value of y_{i+1} is completely defined by y_i. This is convenient for programming and use.
2. Every individual integration step, $i \to i+1$, can be performed with the individual integration step size Δx_i. The integration step size is easy to change in the course of integration. It can be used for the **adaptive step size control**, when Δx_i is chosen based on the analysis of error of then numerical integration.
3. RK4 method require 4 calculations of the RHS per time step while the Adams-Moulton method (Sect. 4.6) of the same order require only 2 calculations of the RHS. On the other hand, RK4 has larger stability range, and allows to solve the same problem with more than twice large Δx, so it can be more computationally effective than the Adams-Moulton LMM.

4.5. Runge-Kutta methods

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

- MATLAB has a lot of build-in solvers (integrators) for IVPs for first-order ODEs that implements numerical methods with the adaptive step size control.
- These solvers implements different numerical methods, but have the same syntax.

Syntax:

[x, y] = Solver (@Fun, xspan, ya)

- 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 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.

Example:

```
[ X, Y ] = ode45 ( @RHSDecay, [ 0.0,10.0 ], 2.0 ) ;
Yexact = SolDecay ( X, 0, 1 );
E = abs ( ( Yexact - Y ) ./ Yexact );
loglog ( x, E );
```

4.5. Runge-Kutta methods

The following solvers are available in the MATLAB:

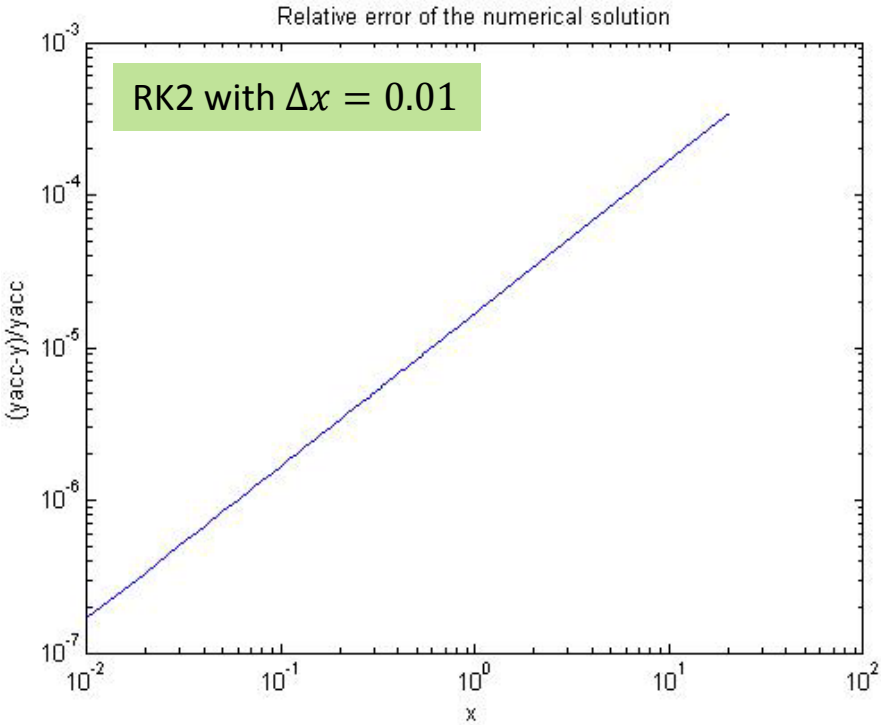
ODE Solver Name	Description
`ode45` RK4-RK5 with the adaptive step size control	For nonstiff problems, one-step solver, best to apply as a first try for most problems. Based on explicit Runge-Kutta method.
`ode23` RK2-RK3 with the adaptive step size control	For nonstiff problems, one-step solver. Based on explicit Runge-Kutta method. Often quicker but less accurate than `ode45`.
`ode113`	For nonstiff problems, multistep solver.
`ode15s`	For stiff problems, multistep solver. Use if `ode45` failed. Uses a variable order method.
`ode23s`	For stiff problems, one-step solver. Can solve some problems that `ode15s` cannot.
`ode23t`	For moderately stiff problems.
`ode23tb`	For stiff problems. Often more efficient than `ode15s`.

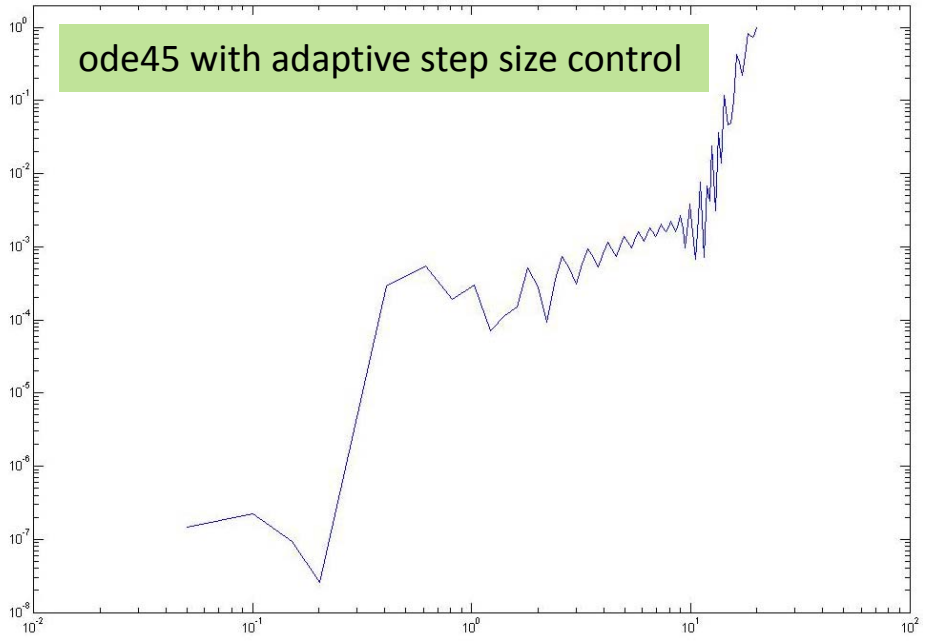
This should be the first solver you try.

4.5. Runge-Kutta methods

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" (slide 9)





- Adaptive step size control algorithm is not absolutely universal and sometimes leads to unsatisfactory results.
- Constant step size integration 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 Δx).

4.6. Numerical solution of IVP for systems of ODEs

Any numerical method developed for single 1st order ODE can be applied for the n-dimensional fundamental system

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

$$\ldots$$

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

Let's rewrite this system using the vector notation:

$$\mathbf{Y}' = \mathbf{F}(x, \mathbf{Y}) \qquad \mathbf{Y} = \begin{pmatrix} y_1 \\ \ldots \\ y_n \end{pmatrix}, \qquad \mathbf{F} = \begin{pmatrix} f_1 \\ \ldots \\ f_n \end{pmatrix} = \mathbf{F}(x, \mathbf{Y}), \tag{4.6.1}$$

Then any numerical method designed for a single ODE in the explicit form can be reformulated for the system by changing scalar quantities to vector ones. For instance, the Runge-Kutta method of 4th order

For single equation (4.5.1):

$$k_1 = \Delta x f(x_i, y_i)$$
$$k_2 = \Delta x f(x_i + 0.5\Delta x, y_i + 0.5k_1)$$
$$k_3 = \Delta x f(x_i + 0.5\Delta x, y_i + 0.5k_2)$$
$$k_4 = \Delta x f(x_i + \Delta x, y_i + k_3)$$
$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$
$$x_{i+1} = x_i + \Delta x$$

For system (4.6.1):

$$\mathbf{K}_1 = \Delta x \mathbf{F}(x_i, \mathbf{Y}_i)$$
$$\mathbf{K}_2 = \Delta x \mathbf{F}(x_i + 0.5\Delta x, \mathbf{Y}_i + 0.5\mathbf{K}_1)$$
$$\mathbf{K}_3 = \Delta x \mathbf{F}(x_i + 0.5\Delta x, \mathbf{Y}_i + 0.5\mathbf{K}_2)$$
$$\mathbf{K}_4 = \Delta x \mathbf{F}(x_i + \Delta x, \mathbf{Y}_i + \mathbf{K}_3)$$
$$\mathbf{Y}_{i+1} = \mathbf{Y}_i + \frac{1}{6}(\mathbf{K}_1 + 2\mathbf{K}_2 + 2\mathbf{K}_3 + \mathbf{K}_4)$$
$$x_{i+1} = x_i + \Delta x$$

4.6. Numerical solution of IVP for systems of ODEs

MATLAB implementation of a solver for a fundamental system of ODEs

$$\mathbf{Y}' = \mathbf{F}(x, \mathbf{Y}) \qquad \mathbf{Y} = \begin{pmatrix} y_1 \\ \dots \\ y_n \end{pmatrix}, \qquad \mathbf{F} = \begin{pmatrix} f_1 \\ \dots \\ f_n \end{pmatrix} = \mathbf{F}(x, \mathbf{Y})$$

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

y = [y1, y2, ..., yn], F = [f1, f2, ..., fn]

- 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 $y_j(x)$.

	y_1	y_2	…	y_n
x_0	$y_1(x_0)$	$y_2(x_0)$	…	$y_n(x_0)$
x_1	$y_1(x_1)$	$y_2(x_1)$		$y_n(x_1)$
x_2	$y_1(x_2)$	$y_2(x_2)$	…	$y_n(x_2)$
	…	…	…	…
x_N	$y_1(x_N)$	$y_2(x_N)$	…	$y_n(x_N)$

Array Y

4.6. Numerical solution of IVP for systems of ODEs

Implementation of the RK2 method for a fundamental system in the MATLAB

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
end
```

Integrator: File RK2.m

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

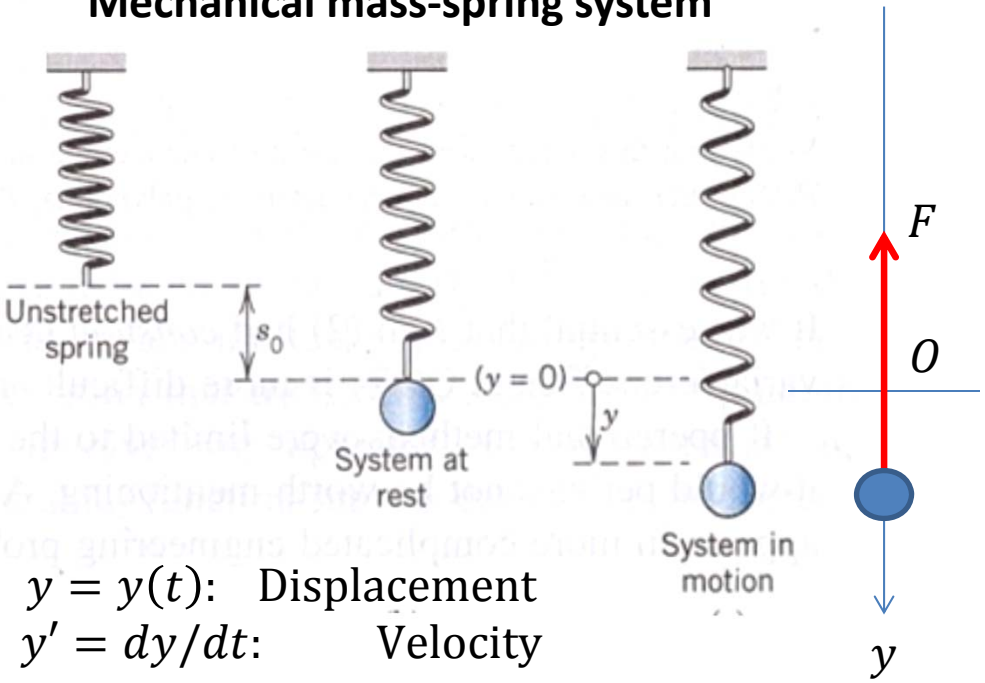
Precisely the same integrator RK2 can be used for both individual equation and system!

This is power of MATLAB.

4.6. Numerical solution of IVP for systems of ODEs

Example 1: Mechanical mass-spring system

Mechanical mass-spring system



$y = y(t)$: Displacement

$y' = dy/dt$: Velocity

Second-order ODE

$$my'' + cy' + ky = r(t)$$

Harmonic driving force

$$r(t) = F_0 \cos \omega t$$

Equivalent two-dimensional system

$$y_1 = y$$
$$y_2 = y'$$
$$my_2' + cy_2 + ky_1 = r(t)$$

$$y_1' = y_2$$
$$y_2' = (r(t) - cy_2 - ky_1)/m$$

Two-dimensional system in vector notation

$$\mathbf{Y}' = \mathbf{F}(t, \mathbf{Y})$$

$$\mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

$$\mathbf{F} = \begin{bmatrix} y_2 \\ (F_0 \cos \omega t - cy_2 - ky_1)/m \end{bmatrix}$$

MATLAB implementation:

RHS: 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
```

In the MATLAB Command window:

```
ODESolverN ( @RK2, @Equation2, 2, 0, 300, [ 1, 1 ], 0.1 )
```

4.6. Numerical solution of IVP for systems of ODEs

In the case of **forced oscillations** ($F_0 \neq 0$), the behavior of the solution drastically depends on the relation between the **input angular frequency** (angular frequency of the driving force) ω and **natural angular frequency** of the mass-spring system $\omega_0 = \sqrt{k/m}$.

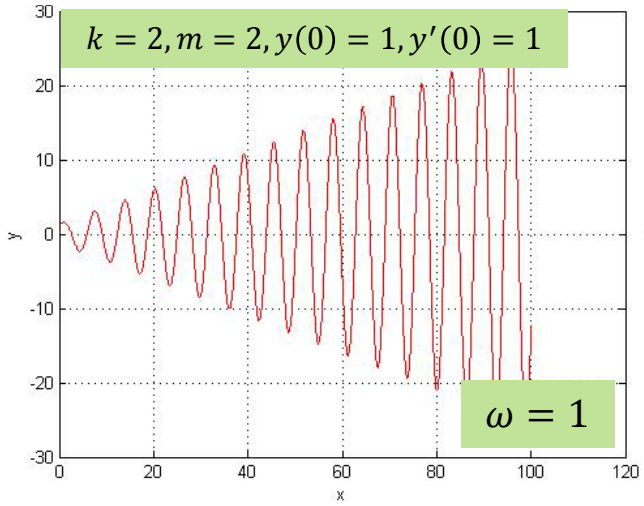
Resonance, beats, and practical resonance

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 $|\omega_0 - \omega| \ll \omega$.

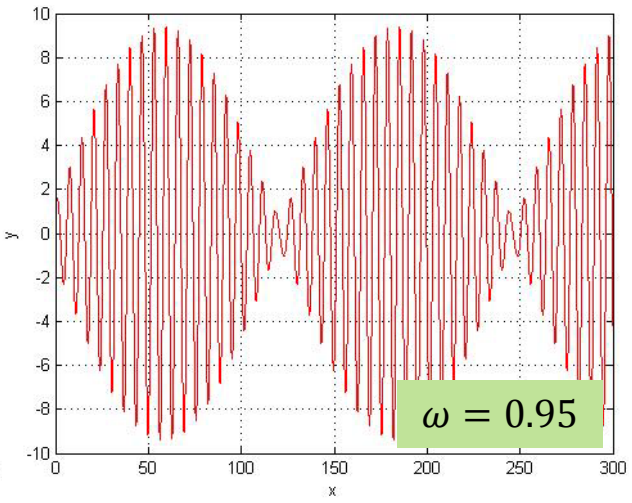
Practical resonance in the damped system ($c \neq 0$): Maximum amplification of oscillations by the driving force with $\omega^2 = \omega_{max}^2 = {\omega_0}^2 - c^2/(2m^2)$

Resonance



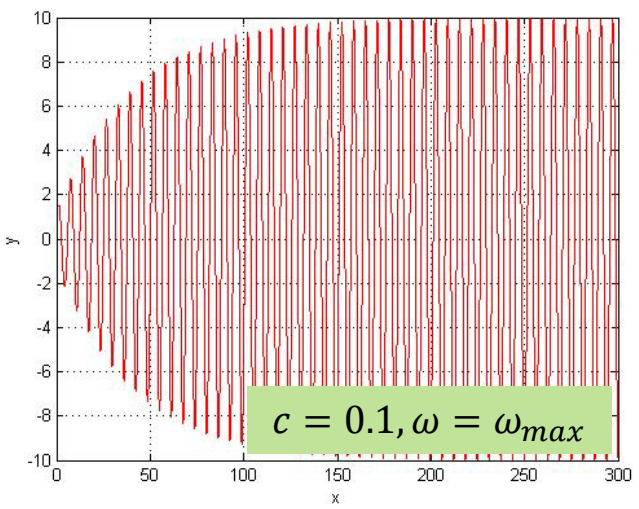
Linear increase of magnitude with time

Beats



Temporal modulation of magnitude

Practical resonance



Strong amplification to a finite level

4.6. Numerical solution of IVP for systems of ODEs

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:

$$\frac{dy_1}{dx} = f_1(x, y_1, y_2), \frac{dy_2}{dx} = f_2(x, y_1, y_2), y_1(x_0) = y_{10}, y_2(x_0) = y_{20}, x_0 = a \le x \le b$$

Name of a function - RHS $f(x, y)$

Initial condition

```
[ x, y ] = ode45 ( @fun, [ a, b ], [ y10, y20 ] )
```

Tables: 1D array of x and 2D array of y

Integration limits

Example:

$$\frac{dy_1}{dx} = xy_1 - y_2, \qquad \frac{dy_2}{dx} = -2y_1, \qquad y_1(0) = 2, \qquad y_2(0) = -2, \qquad 0 \le x \le 4$$

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

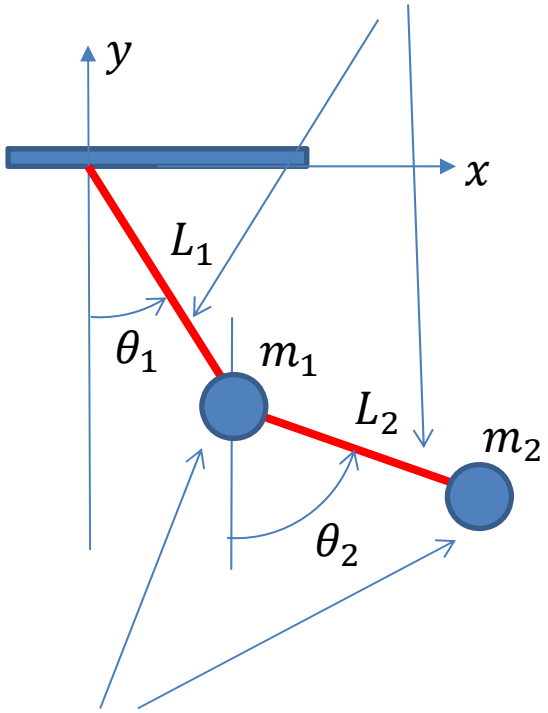
f must be a column vector !!!!

```
[ 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 )
```

4.6. Numerical solution of IVP for systems of ODEs

Example 2: Double pendulum

Rigid massless strings



Bobs

Equations of motion for the double pendulum reduces to the fundamental systems (see Eq. (3.7.3)):

$$z_1 = \theta_1, \qquad z_2 = \dot{\theta}_1, \qquad z_3 = \theta_2, \qquad z_4 = \dot{\theta}_2$$

$$\dot{z}_1 = z_1$$

$$\dot{z}_2 = \frac{A_{22}B_1 - A_{12}B_2}{A_{11}A_{22} - A_{12}A_{21}}$$

$$\dot{z}_3 = z_4$$

$$\dot{z}_4 = \frac{-A_{21}B_1 + A_{11}B_2}{A_{11}A_{22} - A_{12}A_{21}}$$

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

$$\begin{pmatrix} B_1 \\ B_2 \end{pmatrix} = \begin{pmatrix} -g \sin z_1 - \mu_2 L_2 \sin(z_1 - z_3)\, z_4^2 \\ -g \sin z_3 + L_1 \sin(z_1 - z_3)\, z_2^2 \end{pmatrix}$$

4.6. Numerical solution of IVP for systems of ODEs

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
```

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

$$\begin{pmatrix} B_1 \\ B_2 \end{pmatrix} = \begin{pmatrix} -g \sin z_1 - \mu_2 L_2 \sin(z_1 - z_3)\, z_4^2 \\ -g \sin z_3 + L_1 \sin(z_1 - z_3)\, z_2^2 \end{pmatrix}$$

$$\dot{z}_1 = z_1$$

$$\dot{z}_2 = \frac{A_{22}B_1 - A_{12}B_2}{A_{11}A_{22} - A_{12}A_{21}}$$

$$\dot{z}_3 = z_4$$

$$\dot{z}_4 = \frac{-A_{21}B_1 + A_{11}B_2}{A_{11}A_{22} - A_{12}A_{21}}$$

4.6. Numerical solution of IVP for systems of ODEs

Double pendulum: MATLAB script

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_3 / 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:

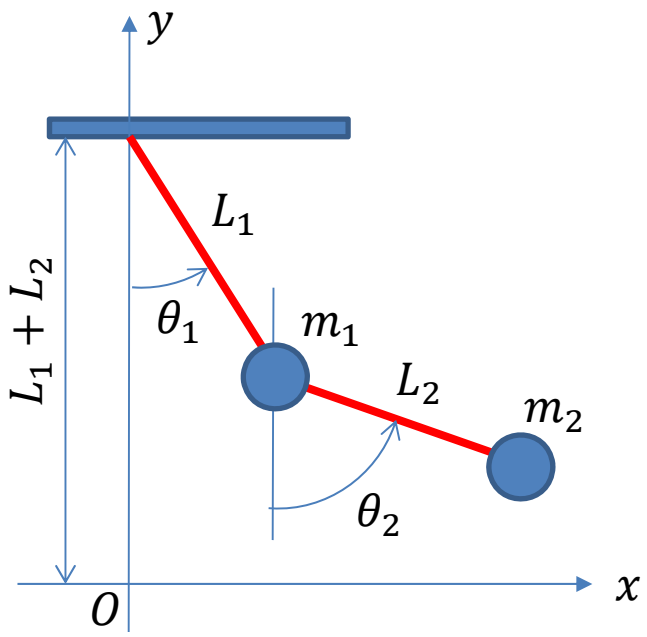
$$\theta_1(0) = \theta_2(0) = 90^0, \dot{\theta}_1(0) = \dot{\theta}_2(0) = 0$$

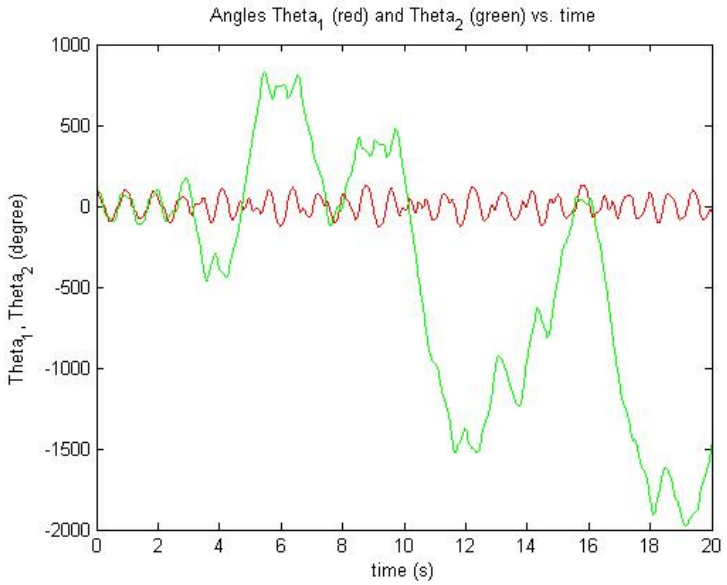
Eq. (7)-1(10):

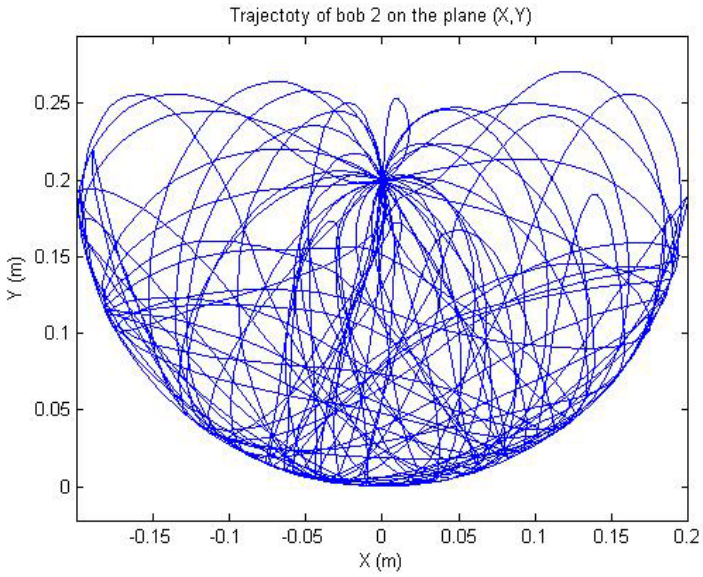
$$x_1 = L_1 \sin\theta_1$$
$$y_1 = L_2 + L_1 - L_1 \cos\theta_1$$
$$x_2 = L_1 \sin\theta_1 + L_2 \sin\theta_2$$
$$y_2 = L_2 + L_1 - L_1 \cos\theta_1 - -L_2 \cos\theta_2$$

4.6. Numerical solution of IVP for systems of ODEs

Double pendulum: Numerical solution







$m_1 = m_2 = 1$ kg, $\quad L_1 = L_2 = 10$ cm

Initial conditions:

$$\theta_1(0) = \theta_2(0) = 90^0$$
$$\dot{\theta}_1(0) = \dot{\theta}_2(0) = 0$$

4.7. Explicit, implicit, and predictor-corrector methods (optional)

Question: How can we improve the stability (enlarge the stability range)?

Let's consider the IVP (4.2.1) and use forward finite difference in order to approximate the derivative in the equation:

$$\frac{y_{i+1} - y_i}{\Delta x} = f(x_{i+1}, y_{i+1})$$

or

$$\boxed{x_{i+1} = x_i + \Delta x, \qquad y_{i+1} = y_i + f(x_{i+1}, y_{i+1})\Delta x} \tag{4.7.1}$$

The method given by Eq. (4.7.1) is called the **implicit Euler method**.

One can show (by taking the Taylor expansion) that the implicit Euler method has the 1st order of approximation.

Let's study the stability of the implicit Euler method. For $f(x, y) = -\lambda y$, the implicit Euler method results in

$$y_{i+1} = y_i - \lambda y_{i+1}\Delta x \Longrightarrow \ (1 + \lambda\Delta x)y_{i+1} = y_i \Longrightarrow y_{i+1} = \frac{y_i}{(1 + \lambda\Delta x)} = \frac{y_0}{(1 + \lambda\Delta x)^{i+1}}$$

We see that $y_i \to 0$ at $i \to \infty$ if $|1 + \lambda\Delta x| < 1$, or if $\lambda > 0$ or $\lambda\Delta x < -2$. Thus, we see that

The implicit Euler method is stable at positive λ for any Δx!

4.7. Explicit, implicit and predictor-corrector methods

Let's compare explicit and implicit Euler methods:

$$\begin{aligned} &\text{Explicit:} && x_{i+1} = x_i + \Delta x, && y_{i+1} = y_i + f(x_i, y_i)\Delta x \\ &\text{Implicit:} && x_{i+1} = x_i + \Delta x, && y_{i+1} = y_i + f(x_{i+1}, y_{i+1})\Delta x \end{aligned}$$

- Both methods have the same order of approximation and, thus provide roughly the same level of accuracy.
- The explicit method is stable only if $0 < \lambda\Delta x < 2$, while the implicit method is stable at arbitrary Δx, and this is the great advantage of the implicit method.
- On the other hand, y_{i+1} can be immediately found with the explicit method, while for the implicit method we must solve the equation (4.7.1) with respect to y_{i+1}. If the original ODE is non-linear, the equation with respect to y_{i+1} can be solved only iteratively.

We call a numerical method **explicit**, if finite difference equation with respect to y_{i+1} is explicit, i.e. does not contain $f(x_{i+1}, y_{i+1})$. Otherwise, we call the method **implicit**.

General properties of the explicit and implicit methods are the same as for the Euler methods:

- Explicit methods have a limit range of stability for the model problem.
- Implicit methods are unconditionally stable, but require more calculations per integration step.

4.7. Explicit, implicit and predictor-corrector methods

Predictor-corrector methods

We combine positive sides of both explicit and implicit methods in one approach using the so-called **predictor-corrector methods** which are composed of two successive steps:

Predictor step: We use some explicit method in order to roughly predict the value of y_{n+1}, e.g.

$$\text{Predictor:}\quad y_{i+1(*)} = y_i + f(x_i, y_i)\Delta x \tag{4.7.2a}$$

Corrector step: We use some implicit method in order to correct prediction $y_{i+1(*)}$, but $f(x_{i+1}, y_{i+1})$ is calculated as $f(x_{i+1}, y_{i+1(*)})$, e.g.

$$\text{Corrector:}\quad x_{i+1} = x_i + \Delta x, \qquad y_{i+1} = y_i + f(x_{i+1}, y_{i+1(*)})\Delta x \tag{4.7.2b}$$

The method given by Eqs. (4.7.2) is called the **predictor-corrector Euler method**.

Predictor-corrector methods remain conditionally stable (i.e. they have a limited range of stability), but

- Using the corrector step, we are able to enlarge the stability range with respect to the purely explicit predictor method.

- Using the predictor we avoid iterations which required for the purely implicit corrector method.

4.8. Linear multistep methods (LMMs). Adams family of LMMs

We are going to develop numerical methods for a 1st order ODE in the explicit form

$$y' = f(x, y) \tag{4.8.1}$$

The Euler method, e.g., the explicit Euler method of approximation of Eq. (4.8.1)

$$\frac{y_{i+1} - y_n}{\Delta x} = f(x_i, y_i) \tag{4.8.2}$$

has the 1st order of approximation. In order increase the order of approximation, we must increase the accuracy of approximation of the derivative in the original ODE. Eq. (4.8.2) implies that at $x_i \le x \le x_{i+1}$:

$$y(x) = y_i \frac{x_{i+1} - x}{\Delta x} + y_{i+1} \frac{x - x_i}{\Delta x}$$
$$f(x, y) = y' = f(x_i, y_i) = const$$

since we know that Eq. (4.8.2) is accurate for any solution in the form of a polynomial of the 1st degree. From this point, we see that in order to obtain methods of higher orders, we need to represent either $y(x)$ or $f(x, y)$ (or both of them) at the interval $x_i \le x \le x_{i+1}$ in the form of polynomials of higher degrees

$$y(x) = \sum_{i=0}^{k} b_i x^i \tag{4.8.3}$$

$$f(x, y(x)) = \sum_{i=0}^{k} a_i x^i \tag{4.8.4}$$

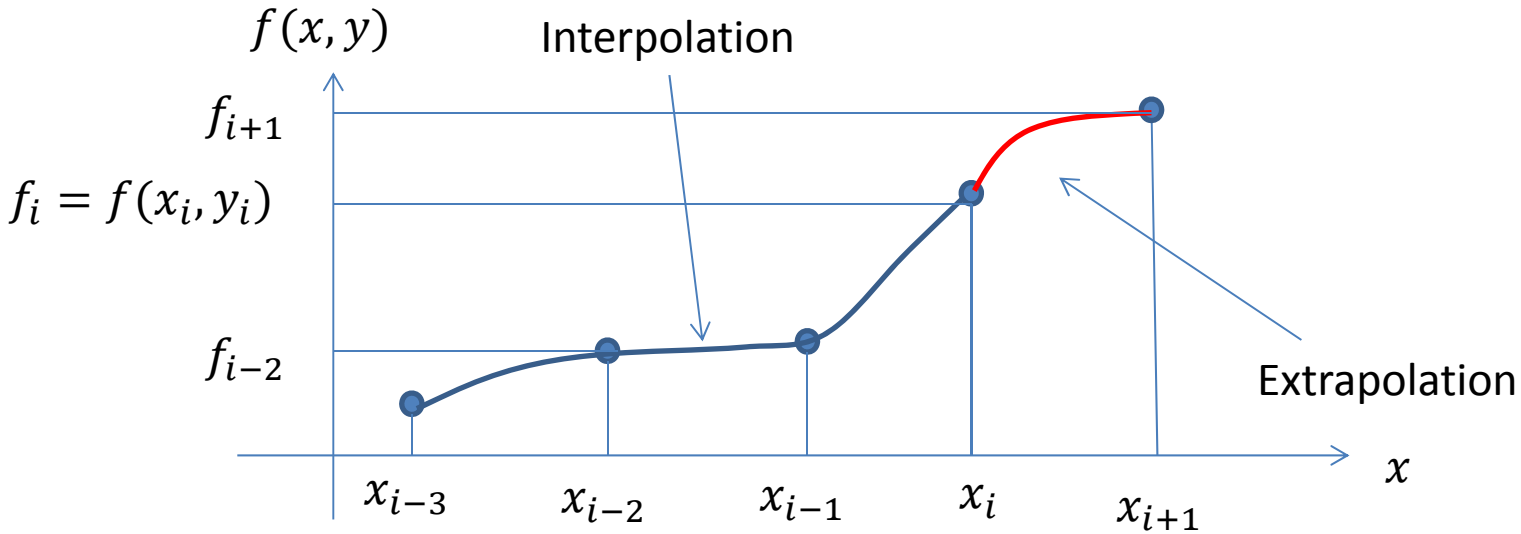
4.8. Linear multistep methods (LMMs). Adams family of LMMs

Question: How can we determine coefficients in Eq. (4.8.2) or (4.8.3)?

We need additional information about the numerical solution in some additional points.

Various high-order numerical methods for ODEs are different mostly by the approach that is used in order to determine these coefficients.

Linear multistep methods (LMMs) are methods, where coefficients in Eq. (4.8.3) and (4.8.4) are determined based on the values of y_i and $f_i = (x_i, y_i)$ in previous integration points, $i < n$.



LMMs are introduced for the constant integration step, $\Delta x = x_i - x_{i-1} = const$ for any i.

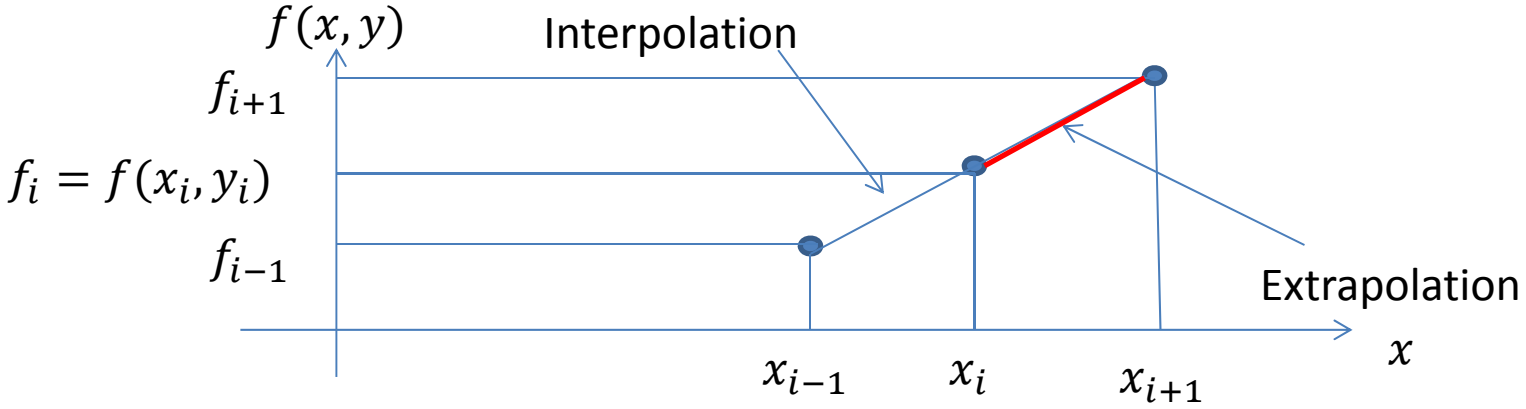
In particular, in the **Adams family** of LMMs, it is assumed that $f(x)$ is an interpolation polynomial of degree k following through points (x_m, f_m), where

For explicit methods: $m = i, i-1, \ldots, i-k$

For implicit methods: $m = i+1, i, \ldots, i-(k-1)$

4.8. Linear multistep methods (LMMs). Adams family of LMMs

Let's first consider how we can introduce an explicit LMM of the second order:



$$r = \frac{x - x_i}{\Delta x}$$
$$x = x_i + \Delta x\, r$$

Let's introduce the linear polynomial going through points (x_{i-1}, f_{i-1}) and (x_i, f_i)

$$f(x, y(x)) = f_i \frac{x - x_{i-1}}{x_i - x_{i-1}} + f_{i-1} \frac{x_i - x}{x_i - x_{i-1}}$$

and integrate this polynomial in the interval $x_i \le x \le x_{i+1}$

$$y_{i+1} - y_i = \int_{x_i}^{x_{i+1}} y' dx = \int_{x_i}^{x_{i+1}} f\big(x, y(x)\big) dx = f_i \int_{x_i}^{x_{i+1}} \frac{x - x_{i-1}}{x_i - x_{i-1}} dx + f_{i-1} \int_{x_i}^{x_{i+1}} \frac{x_i - x}{x_i - x_{i-1}} dx$$

$$\int_{x_i}^{x_{i+1}} \frac{x - x_{i-1}}{x_i - x_{i-1}} dx = \Delta x \int_0^1 (r+1) dr = \frac{3}{2}\Delta x, \qquad \int_{x_i}^{x_{i+1}} \frac{x_i - x}{x_i - x_{i-1}} dx = \Delta x \int_0^1 (-r) dr = -\frac{1}{2}\Delta x$$

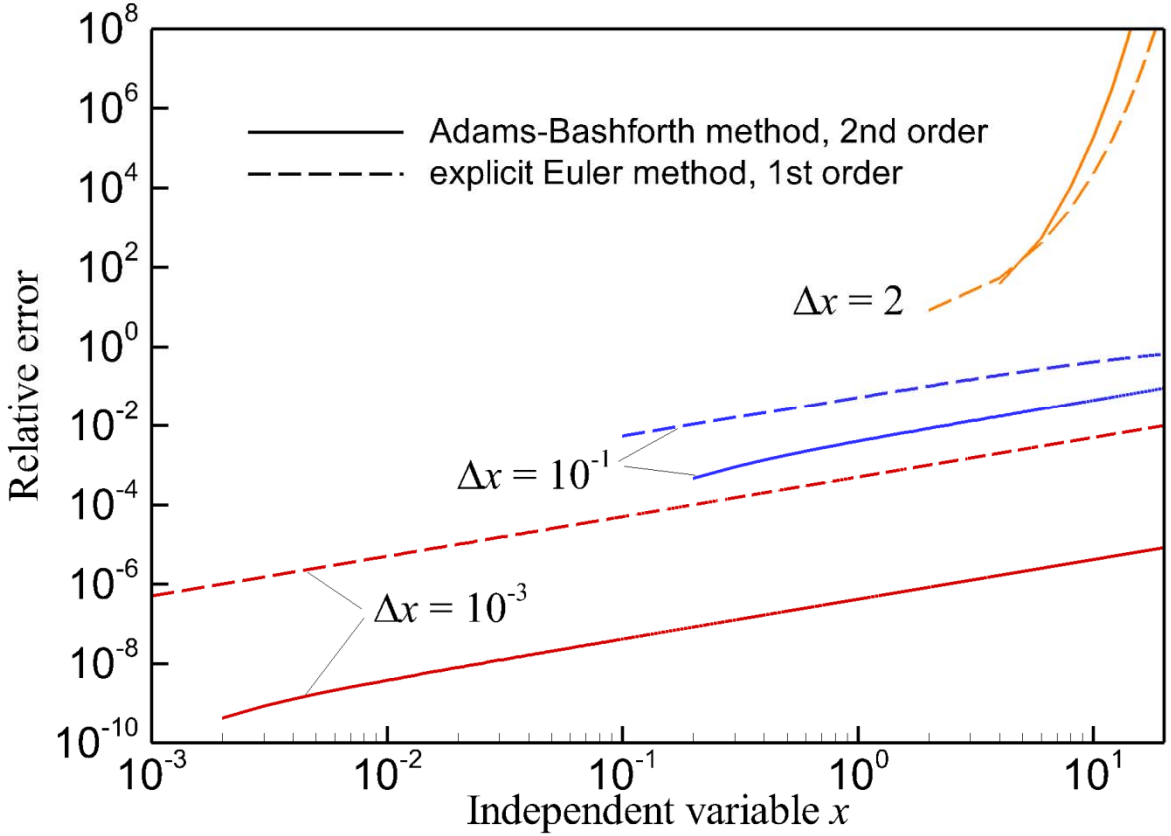
$$\boxed{y_{i+1} = y_i + \frac{\Delta x}{2}(3f_i - f_{i-1})} \qquad (4.8.5)$$

Eq. (4.8.5) is the explicit **Adams-Bashforth method of 2nd order**.

Comparison of the 1st order Euler and 2nd order Adams-Bashforth methods:

Example: $y' = -y, \;\; y(0) = 1, \;\; 0 \le x < 20$. Accurate solution: $y(x) = \exp(-x)$.

Relative numerical error $|\varepsilon_i / y(x_i)|$ vs. x_i

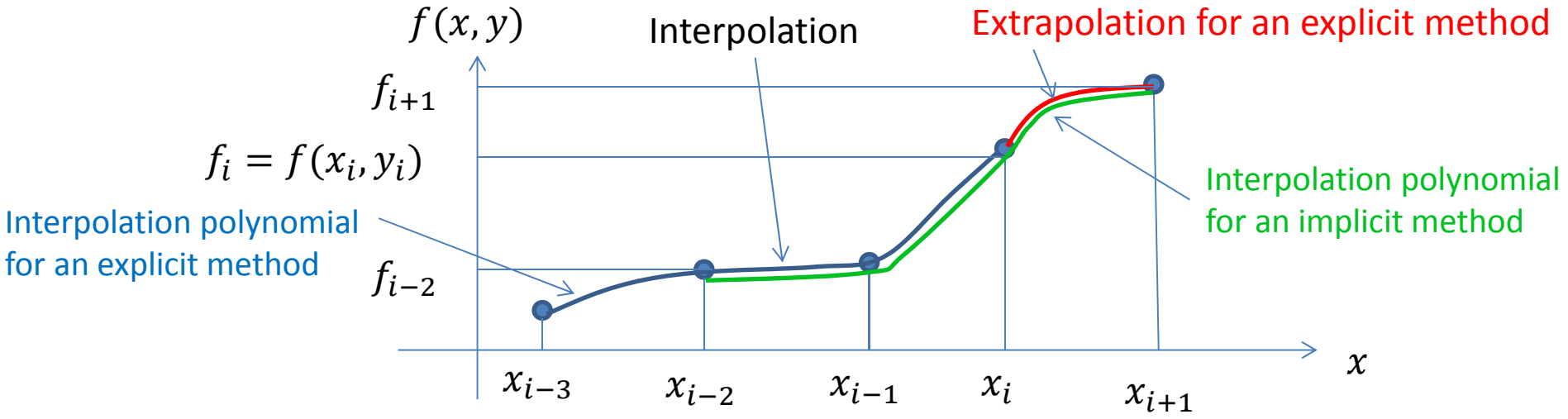


$$\left|\frac{\varepsilon_i}{y(x_i)}\right| = \left|\frac{y(x_i) - y_i}{y(x_i)}\right|$$

- Method of the 2nd order provides superior accuracy with respect to the 1st order method.
- Method of the 2nd order is less stable than the 1st order method.
- Implementation of the Adams-Bashforth method: see ODE1_AdamsBashforth2.cpp.

4.8. Linear multistep methods (LMMs). Adams family of LMMs

Now let's consider how this approach can be used to introduce the explicit LMM of k^{th} order:



Let's first consider explicit methods and represent the interpolation polynomial for $f(x,y)$ in the form of the **Lagrange interpolation polynomial**:

$$f(x, y(x)) = \sum_{m=0}^{k} f_{i-m} L_{i-m}(x) \qquad (4.8.6)$$

where

$$L_{i-m}(x) = \prod_{j=0, j\neq m}^{k} \frac{x - x_{i-j}}{x_{i-m} - x_{i-j}}, \qquad L_{i-m}(x_{i-l}) = \begin{cases} 1 & l = m \\ 0 & l \neq m \end{cases}$$

Now we can substitute Eq. (4.8.6) into the RHS of the differential equation and integrate from $x = x_i$ to $x = x_{i+1}$

4.8. Linear multistep methods (LMMs). Adams family of LMMs

$$y_{i+1} - y_i = \int_{x_i}^{x_{i+1}} y' dx = \sum_{m=0}^{k} f_{i-m} \int_{x_i}^{x_{i+1}} L_{i-m}(x) dx \tag{4.8.7}$$

If we introduce the notation

$$a_m = \frac{1}{\Delta x} \int_{x_i}^{x_{i+1}} L_{i-m}(x) dx \tag{4.8.8}$$

then Eq. (4.8.6) reduces to an explicit finite-difference equation with respect to y_{i+1}:

$$y_{i+1} = y_i + \Delta x \sum_{m=0}^{k} a_m f_{i-m} = \Delta x (a_0 f_i + a_1 f_{i-1} + \cdots + a_k f_{i-k}) \tag{4.8.9}$$

Eqs. (4.8.7) and (4.8.8) give the **explicit Adams or Adams-Bashforth method of k^{th} order**.

Let's calculate coefficients a_i for the popular Adams-Bashforth method of the 4th order ($k = 3$):

$$a_0 = \frac{1}{\Delta x} \int_{x_i}^{x_{i+1}} L_i(x) dx = \int_0^1 L_i(x_i + \Delta x\, r) dr$$

where the new integration variable $r = (x - x_i)/\Delta x$ is introduced and

$$L_i(x) = \prod_{j=0, j\neq 0}^{2} \frac{x - x_{i-j}}{x_i - x_{i-j}} = \frac{(x - x_{i-1})(x - x_{i-2})(x - x_{i-3})}{(x_i - x_{i-1})(x_i - x_{i-2})(x_i - x_{n-3})}$$

$$\tilde{L}_i(r) = L_i(x_i + \Delta x\, r) = \frac{\big((r+1)\Delta x\big)\big((r+2)\Delta x\big)\big((r+3)\Delta x\big)}{(\Delta x)(2\Delta x)(3\Delta x)} = \frac{(r+1)(r+2)(r+3)}{6}$$

$$a_0 = \int_0^1 \tilde{L}_n(r)dr = \frac{1}{6}\int_0^1 (r^3 + 6r^2 + 11r + 6)dr = \frac{55}{24}$$

All other coefficients can be calculated in the same way. It finally gives the following equation for the **Adams-Bashforth method of the 4th order**:

$$y_{i+1} = y_i + \frac{\Delta x}{24}(55f_i - 59f_{i-1} + 37f_{i-2} - 9f_{i-3}) \qquad (4.8.10)$$

For the implicit Adams methods, the interpolation polynomial includes the point (x_{i+1}, f_{i+1}) :

$$f(x, y) = \sum_{m=-1}^{k-1} f_{i-m} L_{i-m}(x)$$

(compare with Eq. (4.8.5): Only limits for m are changed!). Now, by inserting (4.8.10) into (4.8.1) and integrating, one can obtain

$$y_{i+1} - y_i = \Delta x \sum_{m=-1}^{k-1} a_m f_{i-m} = \Delta x(a_{-1} f(x_{i+1}, y_{i+1}) + a_0 f_i + a_1 f_{i-1} + \cdots + a_k f_{i-k+1}) \qquad (4.8.11)$$

4.8. Linear multistep methods (LMMs). Adams family of LMMs

All coefficients can be calculated in the same way as it is done for the explicit method. if $k = 3$, It gives the following equation for the **implicit Adams-Moulton method of the 4th order**:

$$y_{i+1} = y_i + \frac{\Delta x}{24}\left(9f_{i+1} + 19f_i - 5f_{i-1} + f_{i-2}\right) \tag{4.8.12}$$

Adams-Moulton predictor-corrector method of the 4th order:

$$\text{Predictor: } y_{i+1(*)} = y_i + \frac{\Delta x}{24}\left(55f_i - 59f_{i-1} + 37f_{i-2} - 9f_{i-3}\right)$$
$$\text{Corrector: } y_{i+1} = y_i + \frac{\Delta x}{24}\left(9f\left(x_i + \Delta x, y_{i+1(*)}\right) + 19f_i - 5f_{i-1} + f_{i-2}\right) \tag{4.8.13}$$

Notes:

1. When we start numerical integration, the Adams-Moulton PC method can be applied only after 3 successive steps, i.e. only for calculation of y_4. Values y_1, y_2, and y_3 should be obtained with other methods: either LMM of smaller order of approximation, or Runge-Kutta methods considered in the next section.
2. LMMs in the classical formulation can be used only for constant value of the integration step size Δx. Thus, adaptive changing Δx for the error control is problematic for classic LMMs.