

Delphi and Lazarus module RK_Method v.4.00 beta

The module contains a function for solving ordinary differential equations and their systems (of virtually unlimited size) using the 4th rank Runge-Kutta method with a constant step. All types needed to work with it are available from the module.

The method is intended for ODE systems of the type:

$$\begin{aligned}\dot{X} &= f(t, X, Y, \dots), \\ \dot{Y} &= g(t, X, Y, \dots), \\ \text{etc.},\end{aligned}$$

having a solution:

$$\begin{aligned}X &= X(t), \\ Y &= Y(t), \\ \text{etc.},\end{aligned}$$

where t – independent variable (usually time);

X, Y , etc. are the desired functions (variables dependent on t).

The functions f, g , etc. must be given. The initial conditions, i.e. the values of the sought functions at the initial moment, must also be given.

One differential equation is a special case of a system with one eq.

The method can also be useful for solving differential equations of higher (second, etc.) order, since these equations can be represented by a system of differential equations of the first order.

The Runge-Kutta method involves the use of the following formulas:

$$\begin{aligned}X_{k+1} &= X_k + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4), \\ Y_{k+1} &= Y_k + \frac{1}{6}(m_1 + 2m_2 + 2m_3 + m_4), \dots,\end{aligned}$$

where

$$\begin{aligned}k_1 &= f(t_k, X_k, Y_k, \dots) \Delta t, \\ m_1 &= g(t_k, X_k, Y_k, \dots) \Delta t, \dots, \\ k_2 &= f\left(t_k + \frac{\Delta t}{2}, X_k + \frac{k_1}{2}, Y_k + \frac{m_1}{2}, \dots\right) \Delta t, \\ m_2 &= g\left(t_k + \frac{\Delta t}{2}, X_k + \frac{k_1}{2}, Y_k + \frac{m_1}{2}, \dots\right) \Delta t, \dots,\end{aligned}$$

$$k_3 = f\left(t_k + \frac{\Delta t}{2}, X_k + \frac{k_2}{2}, Y_k + \frac{m_2}{2}, \dots\right) \Delta t,$$

$$m_3 = g\left(t_k + \frac{\Delta t}{2}, X_k + \frac{k_2}{2}, Y_k + \frac{m_2}{2}, \dots\right) \Delta t, \dots,$$

$$k_4 = f(t_k + \Delta t, X_k + k_3, Y_k + m_3, \dots) \Delta t,$$

$$m_4 = g(t_k + \Delta t, X_k + k_3, Y_k + m_3, \dots) \Delta t, \dots$$

The module introduces the following types:

- 1) **TFloat** type is simply a **Double** type (you can replace it with **Extended**, for example):

type

TFloat = Double;

- 2) Types **TFloatVector** and **TFloatMatrix** – one-dimensional and two-dimensional dynamic arrays of elements of the **TFloat**:

type

TFloatVector = array of TFloat;

TFloatMatrix = array of array of TFloat;

- 3) Type **TSystem** - procedural:

type

**TSystem = procedure(t: TFloat; const X: TFloatVector;
var RP: TFloatVector);**

Procedures of this type are used as input to a method and serve to describe a system of equations. Although Delphi and Lazarus allow changing elements of a constant dynamic array, the body of the procedure should use elements of the array X as **read-only**!

Function **rk4fixed()** for solution of ODEs and ODE systems

The function prototype is as follows:

**function rk4fixed(
Syst: TSystem;
const InitConds : TFloatVector;
First : TFloat;
Last : TFloat;
PointsCnt : Cardinal;
out tPoints : TFloatVector;
out XPoints : TFloatMatrix;**

StepsFact : Cardinal = 1
) : Word;

Function parameters:

- 1) **Syst** – an input parameter of the procedural type **TSystem**, describing the right-hand sides of the system of equations (i.e., functions **f**, **g**, etc.; see above). The system equations and sought variables are numbered starting with index zero. The parameter will be discussed in the example below.
- 2) **InitConds** – input parameter, vector of initial values of the sought variables (initial conditions). The numbering is consistent with the numbering of the system equations, i.e. for the first equation of the system, which has index zero, the initial condition also has index zero, etc.
- 3) **First**, **Last** – input parameters, the initial (for which the initial conditions are specified) and end points of the calculation interval.
- 4) **PointsCnt** – input parameter, the number of points of the calculation interval for the output data.
- 5) **tPoints** – an output parameter, a one-dimensional array in which the values of the independent variable will be output. The size n of the array is equal to the number of points in the calculation interval for saving. The numbering of points starts from zero. Index zero corresponds to the starting point of the calculation interval. The maximum index (equal to $n - 1$) corresponds to end of calculation interval.
- 6) **XPoints** – output parameter, a two-dimensional array (matrix) of the results of calculating the variables (excluding the independent one). The matrix has a size of m by n , where m – the number of variables (not counting the independent one), and n is the same as for **tPoints**. For example, **XPoints**[1,0] – the value of the variable with index 1 at the initial point of the calculation interval.
- 7) **StepsFact** – an input parameter used (along with **PointsCnt**) to regulate the accuracy and time of calculation by increasing the number of steps in the calculation interval by **StepsFact** times inside the method. For example, if the specified number of points to save is 11 (corresponding to 10 steps), then with the **StepsFact** parameter equal to 1 thousand, the calculation will be performed with a total number of steps of 10 thousand. Intermediate calculation points are discarded from the result. This parameter is set to one by default.

The function returns error codes. The calculation results are available through the variables **tPoints** and **XPoints**.

List of error codes:

Error Code 1000: Invalid data.

Error Code 2000: The end of an interval must be no earlier than its beginning on the axis.

Error Code 3000: Unable to allocate memory.
Error Code 0: No errors found .

Example of solving a system of differential equations using the `rk4fixed()` function

Let the problem be stated as follows:

$$\begin{aligned}dX_0/dt &= -1/t^2 \\ dX_1/dt &= -2X_0/t^2\end{aligned}$$

$$\begin{aligned}X_0(1) &= 1 \\ X_1(1) &= 1\end{aligned}$$

It has the following analytical solution (later you can compare the calculation results with it):

$$\begin{aligned}X_0(t) &= 1/t \\ X_1(t) &= 1/t^2\end{aligned}$$

First of all, we write down the procedure `Syst`, which defines the system of equations:

```
procedure Syst(t: TFloat; const X: TFloatVector;  
               var RP: TFloatVector);  
begin  
    RP[0] := -1.0/(t*t);      // means dX0/dt = -1/t^2  
    RP[1] := -2.0*X[0]/(t*t); // means dX1/dt = -2X0/t^2  
end;
```

And an array of initial conditions (there are 2 of them) `InitConds`:

```
var InitConds : TFloatVector;  
...  
SetLength(InitConds, 2); // memory allocation  
...  
InitConds[0] := 1.0; // X0(1) = 1  
InitConds[1] := 1.0; // X1(1) = 1  
// the same, but shorter: InitConds := [1,1];
```

Let's say we want to get the calculated values at points $t = 1, 2, \dots, 11$. These are 11 points, starting with the initial one (we have `First = 1`) and ending with 11. Then we set the parameter `Last` value 11 and the `PointsCnt` parameter meaning 11. `StepsFact` is selected for the desired calculation accuracy. Let's set it, for example,

equal to 10. Let's call the result arrays in the calling program, for example, `tOut` and `XOuts`.

```
var
  tOut   : TFloatVector;
  XOuts  : TFloatMatrix;
  ...
```

To calculate, we call the method function:

```
Code := rk4fixed(Syst, InitConds, 1., 11., 11, tOut, XOuts, 10);
```

After execution, the function will return the error code `Code` of the `Word` type (this variable must be declared in advance). If everything is OK, then this code is zero. If successful, the `tOut` and `XOuts` arrays will contain the results of the system solution. The calculation results are interpreted as follows:

```
tOut[I]      → t[I]
XOuts[0,I]   → X0[I]
XOuts[1,I]   → X1[I]
...
```

where `I` is the point number, starting from zero.

See also the example included with the module (for Lazarus).