

## Модуль Delphi и Lazarus RK\_Method v.4.00 beta

Модуль содержит функцию для решения обыкновенных дифференциальных уравнений и их систем (практически неограниченного размера) методом Рунге-Кутты 4-го ранга с постоянным шагом. Из модуля доступны все типы, которые нужны для работы с ним.

Метод предназначен для систем ОДУ вида:

$$\dot{X}=f(t,X,Y,\dots),$$

$$\dot{Y}=g(t,X,Y,\dots),$$

и т.д.,

имеющих решение:

$$X=X(t),$$

$$Y=Y(t),$$

и т.д.,

где  $t$  – независимая переменная (обычно время);

$X$ ,  $Y$  и т.д. – искомые функции (зависимые от  $t$  переменные).

Функции  $f$ ,  $g$  и т.д. должны быть заданы. Также должны быть заданы и начальные условия, т.е. значения искомых функций в начальный момент.

Одно диф. уравнение – частный случай системы с одним элементом.

Метод может быть полезен и для решения диф. уравнений высшего (второго и т.д.) порядка, т.к. эти уравнения могут быть представлены системой диф. уравнений первого порядка.

Метод Рунге-Кутты заключается в применении следующих формул:

$$X_{k+1}=X_k+\frac{1}{6}(k_1+2k_2+2k_3+k_4),$$

$$Y_{k+1}=Y_k+\frac{1}{6}(m_1+2m_2+2m_3+m_4), \dots,$$

где

$$k_1=f(t_k,X_k,Y_k,\dots)\Delta t,$$

$$m_1=g(t_k,X_k,Y_k,\dots)\Delta t, \dots,$$

$$k_2=f\left(t_k+\frac{\Delta t}{2},X_k+\frac{k_1}{2},Y_k+\frac{m_1}{2},\dots\right)\Delta t,$$

$$m_2=g\left(t_k+\frac{\Delta t}{2},X_k+\frac{k_1}{2},Y_k+\frac{m_1}{2},\dots\right)\Delta t, \dots,$$

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

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

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

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

Модуль вводит следующие типы:

- 1) Тип **TFloat** – просто тип **Double** (вы можете его заменить, например, на **Extended**):

**type**

**TFloat = Double;**

- 2) Типы **TFloatVector** и **TFloatMatrix** – одномерный и двумерный динамические массивы элементов типа **TFloat**:

**type**

**TFloatVector = array of TFloat;**

**TFloatMatrix = array of array of TFloat;**

- 3) Тип **TSystem** – процедурный:

**type**

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

Процедуры данного типа используются как входные в методе и служат для описания системы уравнений. Хотя Delphi и Lazarus позволяют изменять элементы константного динамического массива, в теле процедуры следует использовать элементы массива X только для чтения!

Функция **rk4fixed()** решения ОДУ и систем ОДУ

Прототип функции следующий:

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

StepsFact : Cardinal = 1  
) : Word;

Параметры функции:

- 1) **Syst** – входной параметр процедурного типа **TSystem**, описывающий правые части системы уравнений (т.е. функции **f**, **g** и т. д.; см. выше). Уравнения системы и искомые переменные нумеруются с индекса ноль. Параметр будет рассмотрен на примере ниже.
- 2) **InitConds** – входной параметр, вектор начальных значений искомых переменных (начальные условия). Нумерация согласована с нумерацией уравнений системы, т.е. для первого уравнения системы, имеющего индекс ноль, начальное условие имеет тоже индекс ноль и т.д.
- 3) **First, Last** – входные параметры, начальная (для которой заданы начальные условия) и конечная точки расчётного интервала.
- 4) **PointsCnt** – входной параметр, количество точек расчётного интервала для выходных данных.
- 5) **tPoints** – выходной параметр, одномерный массив, в который будут выводиться значения независимой переменной. Размер *n* массива равен числу точек по расчетному интервалу для сохранения. Нумерация точек – с нуля. Индексу ноль соответствует начальная точка расчетного интервала. Максимальный индекс (равный *n*-1) соответствует концу расчётного интервала.
- 6) **XPoints** – выходной параметр, двумерный массив (матрица) результатов расчёта переменных (без учета независимой). Матрица имеет размер *m* на *n*, где *m* – число переменных (не считая независимую), а *n* – то-же, что и для **tPoints**. Например, **XPoints[1,0]** – значение переменной с индексом 1 в начальной точке расчётного интервала.
- 7) **StepsFact** – входной параметр, служащий (наряду с **PointsCnt**) для регулирования точности и времени расчёта путем увеличения числа шагов на расчетном интервале в **StepsFact** раз внутри метода. Например, если заданное число точек для сохранения равно 11 (соответствует 10 шагам), то с параметром **StepsFact**, равным 1 тыс., расчёт будет вестись с общим числом шагов 10 тыс. Промежуточные расчётные точки из результата отбрасываются. Данный параметр по-умолчанию установлен в единицу.

Функция возвращает коды ошибок. Результаты расчётов доступны через переменные **tPoints** и **XPoints**.

Список кодов ошибок:

Error Code 1000: Неверные данные.

Error Code 2000: Конец интервала должен быть по оси не ранее его начала.

Error Code 3000: Невозможно выделить память.

Error Code 0: Ошибок не обнаружено.

### Пример решения системы диф. уравнений при помощи функции rk4fixed()

Пусть задача поставлена так:

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

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

Она имеет следующее аналитическое решение (позже можно сравнить с ним результаты расчёта):

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

Прежде всего запишем процедуру Syst, определяющую систему уравнений:

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

И массив начальных условий (их 2) InitConds:

```
var InitConds : TFloatVector;
...
SetLength(InitConds, 2); // выделение памяти
...
InitConds[0] := 1.0;      // X0(1) = 1
InitConds[1] := 1.0;      // X1(1) = 1
// то-же, но короче: InitConds := [1,1];
```

Допустим, мы хотим получить расчетные значения в точках  $t = 1, 2, \dots, 11$ . Это 11 точек, начиная с исходной (у нас First = 1) и заканчивая 11. Тогда зададим параметру Last значение 11 и параметру PointsCnt значение 11. StepsFact подбирается под желаемую точность расчёта. Зададим его, например, равным 10. Массивы результатов назовем в вызывающей программе, например, tOut и XOuts.

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

Для расчета вызываем функцию метода:

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

После выполнения функция вернет код ошибки **Code** типа **Word** (эту переменную надо объявить заранее). Если все нормально, то этот код – ноль. При успешном выполнении в массивах **tOut** и **XOuts** будут содержаться результаты решения системы. Результаты расчета трактуются так:

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

где **I** – номер точки, начиная с нуля.

См. также прилагаемый к модулю пример (для Lazarus).