Модуль Delphi и Lazarus RK_Method v.3.00 beta2

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

Метод предназначен для систем ОДУ вида:

$\dot{X} = f(t, X, Y, \ldots)$,
$\dot{Y} = g(t, X, Y, \ldots)$,
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 заданны 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-Кутта заключается в применении следующих формул:

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

где

$k_1 = f(t_k, X_k, Y_k, \ldots)\Delta t$,
$m_1 = g(t_k, X_k, Y_k, \ldots)\Delta t$, …,
$k_2 = f(t_k + \frac{\Delta t}{2}, X_k + \frac{k_1}{2}, Y_k + \frac{m_1}{2}, \ldots)\Delta t$,
$m_2 = g(t_k + \frac{\Delta t}{2}, X_k + \frac{k_1}{2}, Y_k + \frac{m_1}{2}, \ldots)\Delta t$, …,

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

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

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

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

Функция rk4fixed() решения ОДУ и систем ОДУ

Прототип функции следующий:

```
function rk4fixed(
  Syst: TSystem;
  const InitConds: TFloatVector;
  First: TFloat;
  Last : TFloat;
  var tPoints: TFloatVector;
  var XPoints: TFloatMatrix;
  StepsFact: Cardinal = 1
              ):Word;
```

Функция использует типы:

1) Тип TFloat — просто тип Double:

```
type
  TFloat = Double;
```

Вы можете его переопределить, например, в Extended.

2) Типы TFloatVector и TFloatMatrix — одномерный и двумерный массивы элементов типа TFloat:

```
type
  TFloatVector = array of TFloat;
  TFloatMatrix = array of array of TFloat;
```

3) Тип TSystem — процедурный. Определяется так:

```
type
  TSystem = procedure (var t: TFloat; var X: TFloatVector;
                       var RP: TFloatVector);
```

Параметры функции:

1) Syst - параметр процедурного типа TSystem, описывает правые части системы уравнений (т.е. функции f, g и т.д., см. начало). Уравнения системы и переменные нумеруются с индекса ноль. Параметр будет рассмотрен на примере ниже.
2) InitConds - 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 - начальная (для которой заданы начальные условия) и конечная точки расчетного интервала.
4) tPoints – одномерный массив, в который будут выводиться значения независимой переменной. Размер *n* массива должен быть равен числу точек по расчетному интервалу, которое нужно сохранить. Нумерация точек – с нуля. Индексу ноль соответствует начальная точка расчетного интервала. Максимальный индекс (равный *n*-1) – конец расчетного интервала.
5) XPoints — двумерный массив (матрица) результатов расчета переменных (без учета независимой). Матрица должна иметь размер *m* на *n*, где *m* — число переменных (не считая независимую), а *n* — то-же, что и для tPoints. Например, XPoints[1,3] - значение переменной с индексом один после трех шагов.
6) StepsFact - параметр, служащий для поднятия точности расчетов путем увеличения числа шагов на расчетном интервале в StepsFact раз. Например, если заданное число точек для сохранения равно 11 (соответствует 10 шагам), то с параметром StepsFact равным 1 млн. расчет будет вестись с общим числом шагов 10 млн. Это позволяет сильно экономить память, отбрасывая промежуточные расчетные точки. Данный параметр по умолчанию установлен в единицу.

Функция возвращает коды ошибок. Результаты расчетов 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 4000: Крах метода.
Error Code 5000: Неизвестная ошибка.
Error Code 0: Ошибок не обнаружено.

Пример решения системы диф. уравнений при помощи функции rk4fixed()
Пусть задача поставлена так:

$dx_0/dt = 2t$
$dx_1/dt = 3x_0$

$x_0(0)=0$
$x_1(0)=0$

Она имеет следующее аналитическое решение (позже можно сравнить его с расчетом):

$x_0(t)=t^2$
$x_1(t)=t^3$

Прежде всего запишем процедуру Syst, определяющую систему наших уравнений:

```
procedure Syst (var t: TFloat; var X: TFloatVector;
                var RP: TFloatVector);
begin
  RP[0] := 2*t;        //означает dx0/dt = 2t
  RP[1] := 3*X[0];  //означает dx1/dt = 3x0
end;
```

И массив начальных условий InitConds :

```
var InitConds: TFloatVector;
....
SetLength(InitConds, 2); //число начальных условия по числу уравнений, т.е. 2
     InitConds[0]:=0.0;    //означает x0(0)=0
     InitConds[1]:=0.0;    //означает x1(0)=0
```

Допустим, мы хотим получить расчетные значения в точках t = 0,1,2,..10. Это 11 точек, начиная с исходной (у нас First = 0) и заканчивая 10. Тогда зададим параметр Last = 10 и зададим массивам результатов (назовем их tOut и XOuts) подходящие размеры:

```
var
   tOut: TFloatVector;
   XOuts: TFloatMatrix;
....
SetLength(tOut, 11); //для хранения 11 точек по t
SetLength(XOuts, 2, 11); //для x0 и x1 в 11 точках по t
```

Точность расчетов по 10 шагам (что соответствует нашим 11 точкам) обычно невелика. Поэтому при вызове функции зададим множитель числа шагов при расчете (StepsFact), например, 1 млн.

Для расчета вызываем функцию метода с параметрами:

Code:= rk4fixed(Syst, InitConds, 0.0, 10.0, tOut, XOuts, 1000000);

После выполнения функция вернет код ошибки Code (эту переменную надо описать заранее как Word). 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.

В конце в вызывающей программе не забываем очистить память:

XOuts := **Nil**; tOut := **Nil**; InitConds := **Nil**;

См. также прилагаемые к модулю примеры.