Slope-Oriented ODE Solution Methods

by

Namir Shammas

Introduction

Numerical integration methods that solve initial-values ODE problems usually calculate the value of Y at regular steps of X. This paper presents an adaptive approach for numerical solution for ODE problems. I will call this new method the Slope-Oriented ODE Solution (or SOODES for short). The new algorithm focuses on estimating the increment in X such that the change in the value of Y does not exceed a preselected limit. This scheme allows the adaptive method to move fast in small absolute slopes sections of Y and conversely move slow through high absolute slope values.

The Basic Algorithm

The basic algorithm is very simple. Here is the pseudo-code for the basic algorithm:

```
Given dY/dX = f(X,Y), we need to integrate from X1 (and Y1) to Xmax, with
YMaxDelta maximum change in Y:
   • Repeat
         ○ d = dY/dX_at_(X1,Y1)
         ○ Adjust d if it is zero or close to zero.
         ○ h = 2 * |YMaxDelta / d|
         ○ Repeat
               ▪ h = h / 2
               ▪ X2 = X1 + h
         ○ Until |Y2 - Y1| <= YMaxDelta
         ○ If X2 > Xmax then
               ▪ h = Xmax- X1
               ▪ X2 = Xmax
         ○ End
         ○ Calculate Y2 for X2 using an algorithm of your choice.
         ○ X1 = X2
         ○ Y1 = Y2
   • Until X2 >= Xmax
```

The above pseudo-code shows that the step that updates the value of Y is a general one. This means that you can use any algorithm to calculate the next value of Y going from X1 to X2. You can use the Euler method or any version of the Runge-

Kutta methods, just to name a few. Using more accurate methods yields better Y values.

The implementation of the above pseudo-code should include a scheme that allows you to obtain/access the values of Y at regular intervals of X. The VBA code that comes next shows the details for such scheme.

Testing with Excel VBA Code

I tested the new algorithms using Excel taking advantage of the application's worksheet for easy input and the display of intermediate calculations. The following listing shows the Excel VBA code used for testing. It implements the SOODES algorithm use a variety of basic numerical integration methods:

```
Option Explicit
' Version 2. Handles slopes that are zero or close to zero1
Const EPSILON = 0.00000001

Function MyFx(ByVal sFx As String, ByVal X As Double, ByVal Y As Double) As
Double
  Dim Res As Double, SignofDerive As Integer

  sFx = UCase(sFx)
  sFx = Replace(sFx, " ", "")
  sFx = Replace(sFx, "X", "(" & X & ")")
  sFx = Replace(sFx, "Y", "(" & Y & ")")
  MyFx = Evaluate(sFx)
End Function

Function RK4(ByVal sFx As String, ByVal X As Double, ByVal Y As Double, _
             ByVal h As Double) As Double

  Dim k1 As Double, k2 As Double, k3 As Double, k4 As Double
  Dim hby2 As Double

  hby2 = h / 2
  k1 = MyFx(sFx, X, Y)
  k2 = MyFx(sFx, X + hby2, Y + hby2 * k1)
  k3 = MyFx(sFx, X + hby2, Y + hby2 * k2)
  k4 = MyFx(sFx, X + h, Y + h * k3)
  RK4 = Y + h / 6 * (k1 + 2 * (k2 + k3) + k4)

End Function

Function RK4L(ByVal sFx As String, ByVal X As Double, ByVal Y As Double, _
             ByVal h As Double, ByVal Lambda As Integer) As Double

  Dim k1 As Double, k2 As Double, k3 As Double, k4 As Double
  Dim hby2 As Double

  hby2 = h / 2
  k1 = MyFx(sFx, X, Y)
```

```
  k2 = MyFx(sFx, X + hby2, Y + hby2 * k1)
  k3 = MyFx(sFx, X + hby2, Y + (1 / 2 - 1 / Lambda) * h * k1 + h / Lambda *
k2)
  k4 = MyFx(sFx, X + h, Y + (1 - Lambda / 2) * h * k2 + Lambda / 2 * h * k3)
  RK4L = Y + h / 6 * (k1 + (4 - Lambda) * k2 + Lambda * k3 + k4)

End Function


Function RK2(ByVal sFx As String, ByVal X As Double, ByVal Y As Double, _
             ByVal h As Double) As Double

  Dim k1 As Double, k2 As Double
  Dim g As Double

  g = 2 * h / 3
  k1 = MyFx(sFx, X, Y)
  k2 = MyFx(sFx, X + g, Y + g * k1)
  RK2 = Y + h / 4 * (k1 + 3 * k2)

End Function

Sub DoEuler()
  Dim X1 As Double, Y1 As Double, Xmax As Double
  Dim X2 As Double, Y2 As Double, Deriv As Double
  Dim h As Double, YMaxDelta As Double, Xnext As Double, XnextIncr As Double
  Dim sFx As String, R As Integer

  X1 = [B1].Value
  Y1 = [B2].Value
  Xmax = [B3].Value
  YMaxDelta = [B4].Value
  sFx = [B5].Value
  XnextIncr = [B6].Value
  Xnext = X1 + XnextIncr
  R = 2
  Do
    Deriv = MyFx(sFx, X1, Y1)
    If Deriv < EPSILON Then Deriv = 0.0001
    h = 2 * Abs(YMaxDelta / Deriv)
    Do
      h = h / 2
      X2 = X1 + h
      If X2 > Xnext Then
        h = Xnext - X1
        X2 = Xnext
      End If
      Y2 = Y1 + h * MyFx(sFx, X1, Y1)
    Loop Until Abs(Y2 - Y1) <= YMaxDelta

    If X2 = Xnext Then
      Cells(R, 3) = X2
      Cells(R, 4) = Y2
      R = R + 1
      Xnext = Xnext + XnextIncr
      If Xnext > Xmax Then Xnext = Xmax
    End If
```

```
    X1 = X2
    Y1 = Y2
  Loop Until X1 >= Xmax
End Sub

Sub DoRK4()
  Dim X1 As Double, Y1 As Double, Xmax As Double
  Dim X2 As Double, Y2 As Double, Deriv As Double
  Dim h As Double, YMaxDelta As Double, Xnext As Double, XnextIncr As Double
  Dim sFx As String, R As Integer

  X1 = [B1].Value
  Y1 = [B2].Value
  Xmax = [B3].Value
  YMaxDelta = [B4].Value
  sFx = [B5].Value
  XnextIncr = [B6].Value
  Xnext = X1 + XnextIncr
  R = 2
  Do
    Deriv = MyFx(sFx, X1, Y1)
    If Deriv < EPSILON Then Deriv = 0.0001
    h = 2 * Abs(YMaxDelta / Deriv)
    Do
      h = h / 2
      X2 = X1 + h
      If X2 > Xnext Then
        h = Xnext - X1
        X2 = Xnext
      End If
      Y2 = RK4(sFx, X1, Y1, h)
    Loop Until Abs(Y2 - Y1) <= YMaxDelta

    If X2 = Xnext Then
      Cells(R, 3) = X2
      Cells(R, 4) = Y2
      R = R + 1
      Xnext = Xnext + XnextIncr
      If Xnext > Xmax Then Xnext = Xmax
    End If

    X1 = X2
    Y1 = Y2
  Loop Until X1 >= Xmax
End Sub

Sub DoRK4Lambda()
  Dim X1 As Double, Y1 As Double, Xmax As Double
  Dim X2 As Double, Y2 As Double, Deriv As Double
  Dim h As Double, YMaxDelta As Double, Xnext As Double, XnextIncr As Double
  Dim sFx As String, R As Integer, Lambda As Integer

  X1 = [B1].Value
  Y1 = [B2].Value
  Xmax = [B3].Value
  YMaxDelta = [B4].Value
```

```
  sFx = [B5].Value
  XnextIncr = [B6].Value
  Xnext = X1 + XnextIncr
  Lambda = [B7].Value

  R = 2
  Do
    Deriv = MyFx(sFx, X1, Y1)
    If Deriv < EPSILON Then Deriv = 0.0001
    h = 2 * Abs(YMaxDelta / Deriv)
    Do
      h = h / 2
      X2 = X1 + h
      If X2 > Xnext Then
        h = Xnext - X1
        X2 = Xnext
      End If
      Y2 = RK4L(sFx, X1, Y1, h, Lambda)
    Loop Until Abs(Y2 - Y1) <= YMaxDelta

    If X2 = Xnext Then
      Cells(R, 3) = X2
      Cells(R, 4) = Y2
      R = R + 1
      Xnext = Xnext + XnextIncr
      If Xnext > Xmax Then Xnext = Xmax
    End If

    X1 = X2
    Y1 = Y2
  Loop Until X1 >= Xmax
End Sub

Sub DoRK2()
  Dim X1 As Double, Y1 As Double, Xmax As Double
  Dim X2 As Double, Y2 As Double, Deriv As Double
  Dim h As Double, YMaxDelta As Double, Xnext As Double, XnextIncr As Double
  Dim sFx As String, R As Integer

  X1 = [B1].Value
  Y1 = [B2].Value
  Xmax = [B3].Value
  YMaxDelta = [B4].Value
  sFx = [B5].Value
  XnextIncr = [B6].Value
  Xnext = X1 + XnextIncr
  R = 2
  Do
    Deriv = MyFx(sFx, X1, Y1)
    If Deriv < EPSILON Then Deriv = 0.0001
    h = 2 * Abs(YMaxDelta / Deriv)
    Do
      h = h / 2
      X2 = X1 + h
      If X2 > Xnext Then
        h = Xnext - X1
        X2 = Xnext
```

```
      End If
      Y2 = RK2(sFx, X1, Y1, h)
    Loop Until Abs(Y2 - Y1) <= YMaxDelta

    If X2 = Xnext Then
      Cells(R, 3) = X2
      Cells(R, 4) = Y2
      R = R + 1
      Xnext = Xnext + XnextIncr
      If Xnext > Xmax Then Xnext = Xmax
    End If

    X1 = X2
    Y1 = Y2
  Loop Until X1 >= Xmax
End Sub
```

The VBA function **MyFX** calculates the slope value based on a string that contains the function's expression. This expression must use **X** and **Y** as the variable names for X and Y, respectively.

The following subroutines tests the various versions of SOODES:

- The subroutine **DoEuler** applies the Euler's method.
- The subroutine **DoRK2** applies the second-order Runge-Kutta method. This subroutine calls function **RK2** to calculate each subsequent value of Y.
- The subroutine **DoRK4** applies the fourth-order Runge-Kutta method. This subroutine calls function **RK4** to calculate each subsequent value of Y.
- The subroutine **DoRK2L** applies the general fourth-order Runge-Kutta method (developed by Tan Delin and Chen Zheng[3]). This subroutine calls function **RK4L** to calculate each subsequent value of Y.

Figure 1 shows a sample Excel sheet that contains the input and output data

X0	1	X	Y	Exact	Error
Y0	0	1.1	0.095699	0.09531018	-0.00038894
Xmax	2	1.2	0.182986	0.182321557	-0.0006642
Max Delta y	0.01	1.3	0.263229	0.262364264	-0.00086478
dy/dx	1/X	1.4	0.337542	0.336472237	-0.00106984
X Next Incr	0.1	1.5	0.406726	0.405465108	-0.00126126
		1.6	0.471449	0.470003629	-0.00144526
		1.7	0.532252	0.530628251	-0.00162383
		1.8	0.589562	0.587786665	-0.001775
		1.9	0.643783	0.641853886	-0.00192939
		2	0.695209	0.693147181	-0.00206207
				Sqrt of Sum Squares	0.004463916

Figure 1. The Excel spreadsheet used to test Euler's method with the SOODES algorithm.

The Input Cells

The VBA code relies on the following cells to obtain data:

- Cells B1 and B2 supply the initial ODE value (X_0,Y_0).
- Cell B3 contains the maximum value of X.
- Cell B4 contains the value for the maximum change in Y for each step.
- Cell B5 contains the expression for dY/dX = f(X,Y). Notice that the expression in cell B5 uses **X** and **Y** as the variable names. The expression is case insensitive.
- Cell B5 contains the value of the increment in X that is used to display values of Y.
- In the case of the general fourth-order Runge-Kutta method, cell B6 supplies the value for the lambda factor use by this method.

Output

The output appears in the following columns:

- Column C displays the value of X.
- Column D displays the values of Y.
- Column E displays the calculated exact values of Y.

- Column D displays the error in Y. The bottom-most cell in this column calculates the square root of the sum of error squared. This value gives a measure of the overall errors.

Columns E and D serve to examine the errors associated with the particular version of SOODES.

The Results

I tested the various versions of the SOODES.

Euler's Method

I tested dY/dX=1/X for integrating from (1, 0) to 2. I specified a maximum change in function value equal to 0.01 and the results to be displayed at steps of 0.1. The results appear in Figure 1. The value of Y at X=2 was accurate to two decimals.

Second-Order Runge-Kutta Method

I tested dY/dX=1/X for integrating from (1, 0) to 2. I specified a maximum change in function value equal to 0.01 and the results to be displayed at steps of 0.1. The results appear in Figure 1. The value of Y at X=2 was accurate to six decimals.

X0	1	X	Y	Exact	Error
Y0	0	1.1	0.09531018	0.09531018	-2.4845E-09
Xmax	2	1.2	0.18232156	0.182321557	-4.7495E-09
Max Delta y	0.01	1.3	0.26236427	0.262364264	-6.9308E-09
dy/dx	1/X	1.4	0.33647225	0.336472237	-8.8504E-09
X Next Incr	0.1	1.5	0.40546512	0.405465108	-1.0694E-08
		1.6	0.47000364	0.470003629	-1.2345E-08
		1.7	0.53062827	0.530628251	-1.3981E-08
		1.8	0.58778668	0.587786665	-1.5428E-08
		1.9	0.6418539	0.641853886	-1.6801E-08
		2	0.6931472	0.693147181	-1.8164E-08
				Sqrt of Sum Squares	3.83081E-08

Figure 2. The Excel spreadsheet used to test the second-order Runge-Kutta method with the SOODES algorithm.

Fourth-Order Runge-Kutta Method

I tested dY/dX=1/X for integrating from (1, 0) to 2. I specified a maximum change in function value equal to 0.01 and the results to be displayed at steps of 0.1. The results appear in Figure 3. The value of Y at X=2 was accurate to nine decimals.

X0	1	X	Y	Exact	Error
Y0	0	1.1	0.09531018	0.09531018	-7.3685E-12
Xmax	2	1.2	0.182321557	0.182321557	-1.4057E-11
Max Delta y	0.01	1.3	0.262364264	0.262364264	-2.056E-11
dy/dx	1/X	1.4	0.336472237	0.336472237	-2.6265E-11
X Next Incr	0.1	1.5	0.405465108	0.405465108	-3.1719E-11
		1.6	0.470003629	0.470003629	-3.6618E-11
		1.7	0.530628251	0.530628251	-4.1495E-11
		1.8	0.587786665	0.587786665	-4.5745E-11
		1.9	0.641853886	0.641853886	-4.9821E-11
		2	0.693147181	0.693147181	-5.3885E-11
				Sqrt of Sum Squares	1.1363E-10

Figure 3. The Excel spreadsheet used to test the fourth-order Runge-Kutta method with the SOODES algorithm.

General Fourth-Order Runge-Kutta Method

I tested dY/dX=1/X for integrating from (1, 0) to 2. I specified a maximum change in function value equal to 0.01 and the results to be displayed at steps of 0.1. I set the lambda factor to be 3. The results appear in Figure 4. The value of Y at X=2 was accurate to nine decimals. The results as just as accurate as the regular fourth-order Runge-Kutta method.

X0	1	X	Y	Exact	Error
Y0	0	1.1	0.09531018	0.09531018	-7.3685E-12
Xmax	2	1.2	0.182321557	0.182321557	-1.4057E-11
Max Delta	0.01	1.3	0.262364264	0.262364264	-2.056E-11
dy/dx	1/X	1.4	0.336472237	0.336472237	-2.6265E-11
X Next Incr	0.1	1.5	0.405465108	0.405465108	-3.1719E-11
Lambda	3	1.6	0.470003629	0.470003629	-3.6618E-11
		1.7	0.530628251	0.530628251	-4.1495E-11
		1.8	0.587786665	0.587786665	-4.5745E-11
		1.9	0.641853886	0.641853886	-4.9821E-11
		2	0.693147181	0.693147181	-5.3885E-11
				Sqrt of Sum Squares	1.1363E-10

Figure 4. The Excel spreadsheet used to test the general fourth-order Runge-Kutta method with the SOODES algorithm.

Conclusion

The SOODES algorithm provides a mechanism to solve ODE problems in a manner that has the Y values calculated at approximately regular intervals.

References

1. William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery, *Numerical Recipes: The Art of Scientific Computing*, 3rd edition, Cambridge University Press; 3rd edition, September 10, 2007.
2. Richard L. Burden, J. Douglas Faires, *Numerical Analysis*, Cengage Learning, 9th edition, August 9, 2010.
3. Wikipedia, *Runge-Kutta Methods*

Document Information

Version	*Date*	*Comments*
1.0.0	3/12/2014	Initial release.
1.1.0	3/15/2014	Adjusted the VBA code to handle slopes of zero or close to zero.