NUMEROV – Solving differential equations ***y" = f(x,y)***

Abstract

NUMEROV is a program written in 1975 for the HP-25 / 25C calculators to numerically solve 2nd-order differential equations of the form y"=f(x,y) subject to initial conditions, using the 5th-order predictor-corrector Numerov's method. One worked example included.

Keywords: *2nd-order differential equations, Numerov's method, HP calculators, HP-25, HP-25C*

1. Introduction

NUMEROV is a small *(just 37 steps)* program that I wrote in 1975 for the *HP-25 / HP-25C* calculators, which numerically solves 2nd-order differential equations of the form ***y" = f(x, y)***, subject to two initial conditions. Notice that *y'* doesn't appear in *f*'s definition, this is a special case of the more general equation ***y" = f(x, y, y')***.

The program uses the 5th-order *predictor-corrector Numerov's method*, which goes like this:

$$\text{- predictor:} \quad \hat{y}_{k+1} = 2y_k - y_{k-1} + \frac{h^2}{12}(10\, f_k + f_{k-1})$$

$$\text{- corrector:} \quad y_{k+1} = \hat{y}_{k+1} + \frac{h^2}{12} f_{k+1}$$

where $f_n = f(x_n, y_n)$. Two initial values are required, (x_0, y_0), (x_1, y_1) , and ***h*** is the spacing.

2. Program Listing

```
01  RCL 2      23  STO 7      37  STO 2
02  RCL 1      24  R↓         38  STO+6
▶03 f(x,y)     25  STO 6      39  STO+6
..   ...       26  RCL 4      40  RCL 0
..{GTO 15}*    27  RCL 3      41  STO+3
..   ...       28  STO 1      42  RCL 6
▶15 RCL 5      29  GTO 03     43  RCL 3
16   x         ▶30 10^x       44  GTO 03
17  RCL 7      31  STO-7      ▶45 STO-7
18  X<0        32   x         46  R↓
19  GTO 45     33  STO+6      47  RCL 6
20  X#0        34  RCL 2      48   +
21  GTO 30     35  STO-6      49  STO 4
22  e^x        36  RCL 4
```

Register contents **

R_0	h
R_1	x_{k-1}
R_2	y_{k-1}
R_3	x_k
R_4	y_k
R_5	$h^2/12$
R_6	$\hat{y}_{k+1}$
R_7	*flag*

* *use up to 12 steps to define* ***f(x, y)****; if it needs less, end it with* `GTO 15`

** *all storage registers used, none are available to define* ***f(x, y)***

3. Usage Instructions

To numerically solve a differential equation ***y" = f(x, y)*** given two initial conditions, follow these three *steps*:

1) *Store all needed constants and initial conditions:*

h `STO 0` *(just once, doesn't change; see* ***Note 3****)*

$h^2/12$ `STO 5` *(ditto)*

0 `STO 7` *(just once, automatically reset to 0 after each iteration)*

x_0 `STO 1` y_0 `STO 2`

x_1 `STO 3` y_1 `STO 4`

2) *Define* ***f(x, y)***:

- in RUN mode, press: `GTO 02`
- *switch to PRGM mode*
- enter the steps that define ***f(x,y)*** where ***x,y*** are assumed to be in the *X,Y* stack registers, resp.
- *switch to RUN mode*
- press `[f][PRGM]` to reset the program pointer to the beginning of the program.

3) *Compute subsequent values of the solution:* $y_2, y_3, y_4, \ldots$:

`[R/S]` → y_2 `[R/S]` → y_3 ... `[R/S]` → y_n

The corresponding x_k values are: $x_1=x_0+h$, $x_2=x_1+h$, and the computed y_k are of course $y_k= y(x_k)$ (after computing each y_k, the corresponding value of x_k is in storage register R_3)

4. Examples

The following example may be useful to check that the program is correctly entered and to practice how to use it.

4.1 Example 1

Given $\boldsymbol{y'' = x + y}$ with initial conditions $\boldsymbol{y(0) = 1}$, $\boldsymbol{y'(0) = 1}$, find $\boldsymbol{y(1)}$ using as spacing $\boldsymbol{h=0.1}$

We have $\boldsymbol{(x_0, y_0)} = (0, y(0)) = \boldsymbol{(0, 1)}$ and $\boldsymbol{x_1} = x_0+h = 0+0.1 = \boldsymbol{0.1}$, but we still need $\boldsymbol{y_1} = y(x_1) = y(0.1)$.
Using the differential equation itself we get:

$\boldsymbol{y''(0)} = x_0 + y_0 = 0 + 1 = \boldsymbol{1}$

$\boldsymbol{y'''(0)} = D(y'') = D(x+y) = D(x)+D(y) = 1+y'(0) = 1 + 1 = \boldsymbol{2}$

$\boldsymbol{y''''(0)} = D(y''') = D(1+y') = y''(0) = \boldsymbol{1}$

and inserting these values in a *5-term Taylor Series expansion around* $\boldsymbol{x=0}$, we finally get:

$y(x) = y(0) + y'(0)\, x + y''(0)\, x^2/2! + y'''(0)\, x^3/3! + y''''(0)\, x^4/4! + \ldots$ *(generic 5-term Taylor Series expansion)*

i.e.:

$\boldsymbol{y_1} = y(0.1) = \boldsymbol{1} + \boldsymbol{1}(0.1) + \boldsymbol{1}(0.1)^2/2 + \boldsymbol{2}(0.1)^3/6 + \boldsymbol{1}(0.1)^4/24 = \boldsymbol{1.10534}$, *rounded to 5 places.*

Now we have everything we need, so we proceed to store the initial conditions and required constants:

0.1	STO 0	*(h)*	[X²]12[÷]	STO 5	*(h^2/12)*	***0***	STO 7	
0	STO 1	*(x_0)*	***1***	STO 2	*(y_0)*			
0.1	STO 3	*(x_1)*	***1.10534***	STO 4	*(y_1)*			

Now, to define ***f(x, y)***, we do the following. In RUN mode:

GTO 02	*(set the program pointer at step 02, the beginning of the **f(x, y)** definition)*
... switch to PRGM mode ...	
+	*(definition of **f(x, y) = x + y**, where **x** is in stack reg. X and **y** is in stack reg. Y)*
GTO 15	*(the definition took less than 12 steps [actually just <u>one</u>], so GTO 15 ends it)*
... switch to RUN mode ...	
[f][PRGM]	*(sets the program pointer at the beginning of the program)*

Once everything is stored and defined, we finally proceed to compute the solution:

[FIX 5] [R/S] → *1.22274* *(y_2)* [R/S] → *1.35438* *(y_3)* ... [R/S] → ***2.89344*** *(y_{10})*

and the results are summarized in this table:

	⌜ *(initial conditions)* ⌝								*y_{10} = y(1)*
x	***0***	***0.1***	*0.2*	*0.3*	*0.4*	*0.5*	*0.6*	*...*	***1***
y	***1***	***1.10534***	*1.22274*	*1.35438*	*1.50258*	*1.66982*	*1.85877*	*...*	***2.89344***

The exact solution is: $y = \frac{3\,e^{x} - e^{-x}}{2} - x$ so *y(1)* = ***2.89348*** and we've got almost *6 correct digits*.

Notes

1. ***h*** is the spacing, a user-defined increment between computed values. A small ***h*** increases the accuracy but also the running time. As a *rule-of-thumb*, using ***h=0.1*** will usually give about 5 correct places for moderately sized intervals.

2. The error accumulates after each increment and if their number to reach a given value is too big the final error might be unacceptable. In that case use a smaller ***h*** initially. Using first some ***h*** and then ***h/2*** will give an idea of the accumulated error.

3. You *can't* change the value of ***h*** once the program stops after computing some value $\boldsymbol{y_k}$. If you need a smaller ***h*** you must begin anew from *Step 1* above (but you should skip *Step 2* as you don't need to define ***f(x, y)*** again). This might be necessary if the $\boldsymbol{y_k}$ values do not vary much (increase ***h***) or vary too quickly (decrease ***h***) in the interval being computed.

4. You *can* perform computations while the program is stopped between one $\boldsymbol{y_k}$ and the next; using the stack does *not* affect the computation of the next $\boldsymbol{y_{k+1}}$ as long as you change neither program memory nor the contents of the storage registers.

References

Francis Scheid (1988). *Schaum's Outline of Theory and Problems of Numerical Analysis, 2nd Edition*.

Copyrights