SYSDIFEQ – Systems of First-order Differential Equations

Abstract

SYSDIFEQ is a program written in 1980 for the HP-41C programmable calculator to obtain an approximate numerical solution for a system of N simultaneous first-order differential equations using a fourth-order Runge-Kutta method. Two worked examples included.

Keywords: *numerical solution, first-order differential equations, system, Runge-Kutta, programmable calculator, HP-41C, HP42S*

1. Introduction

SYSDIFEQ is a *137-step RPN* program that I wrote in 1980 for the *HP-41C* programmable calculator (will also run in the *HP-41CV/CX* and the *HP42S* with trivial or no changes, see ***Note 1***), to obtain an approximate numerical solution (using a 4th-order Runge-Kutta method) for a system of *N* simultaneous first-order differential equations of the general form:

$$\begin{aligned} y_1' &= f_1(x, y_1, y_2, \ldots, y_n) \\ y_2' &= f_2(x, y_1, y_2, \ldots, y_n) \\ &\ldots \quad \ldots \quad \ldots \\ y_n' &= f_n(x, y_1, y_2, \ldots, y_n) \end{aligned}$$

with the initial conditions: $x = x_0, \quad y_1 = y_1(x_0), \quad \ldots, \quad y_n = y_n(x_0)$

where the f_i are user-defined functions of the variables $x, y_1, y_2, \ldots, y_n$ and $y_1, y_2, \ldots, y_n$ are functions of x.

We seek to compute the values $y_i(x_0 + h)$, where h is an arbitrary increment of the independent variable x and the index $i = 1, 2, \ldots, n$. The input consists of:

a) the equations: $y_i' = f_i(x, y_1, y_2, \ldots, y_n)$
b) the increment: h *(generally small)*
c) the initial values: $y_i(x_0)$

and the output is $y_i(x_0 + h)$, $y_i(x_0 + 2h), \ldots$, for $i = 1, 2, \ldots, n$.

To this purpose, the 4th-order *Runge-Kutta* method of *Gill* is applied as shown in the following pseudocode:

for j = 1 to 4
  for i = 0 to n
    $k_{i,j} = f_i(y_{0,j-1}, y_{1,j-1}, \ldots, y_{n,j-1})$
  next i
  for i = 0 to n
    $y_{i,j} = y_{i,j-1} + h\,(a_j(k_{i,j} - b_j\, q_{i,j-1}))$
    $q_{i,j} = q_{i,j-1} + 3\,(a_j(k_{i,j} - b_j\, q_{i,j-1})) - c_j\, k_{i,j}$
  next i
next j
output y_i *for i = 0 to n*

where the constants are:

$a_1 = c_1 = c_4 = 1/2$
$a_2 = c_2 = 1 - 1/\sqrt{2}, \; a_3 = c_3 = 1 + 1/\sqrt{2}$
$a_4 = 1/6, \; b_1 = 2b_2 = 2b_3 = b_4 = 2$

This procedure is repeated as often as desired to yield $y_i(x_0 + h)$, $y_i(x_0 + 2h), \ldots$

The notation $y_0 = x$ is used for notational convenience and to simplify the form of the process.

Initially $q_{i,0} = 0$, $y_{i,0} = y_i(x_0)$, thereafter $q_{i,0}(x_i) = q_{i,4}(x_{i-1})$

1.1 N-th order differential equations

A single differential equation of orden ***n***:

$$y^{(n)} = f\,(x, y, y', y'', \ldots , y^{(n-1)}) \text{ , subject to initial conditions } \quad x = x_0, y = y_0, y' = y'_0, \ldots, y^{(n-1)} = y_0^{(n-1)}$$

can be reduced to a system of ***n*** first-order differential equations and solved using this program.

For example, consider the 3rd-order differential equation:

$$y''' = 6y'' + 7y' - 8y - 4x + 7, \quad \text{subject to initial conditions} \quad x_0=1, y(1) = 2, y'(1) = 3, y''(1) = 4$$

and substitute: $y = y_1$, $y' = y_2$, $y'' = y_3$ and thus $y''' = y_3'$, so the equation may be written as:

$$\begin{aligned} y_3' &= 6y_3 + 7y_2 - 8y_1 - 4x + 7 \\ y_2' &= y_3 \\ y_1' &= y_2 \end{aligned} \qquad \text{subject to initial conditions} \quad x_0=1, y_1(1) = 2, y_2(1) = 3, y_3(1) = 4$$

which is a system of 3 first-order differential equations that can be solved using this program.

Using four memory modules, in theory equations up to the 80th order might be solved because after the last one all the others take only 5 bytes each to define.

1.2 Program characteristics

This program (labeled as "***RK***" for *Runge-Kutta*) is 32-register, 224 bytes long so it *exactly* fits into a single magnetic card as long as it doesn't have a final **END** instruction, let the final **.END.** of program memory do the task (else it won't fit into a single card, a second card would be required.)

The program solves a system of *N* first-order differential equations, where *N* depends on the available memory:

Number of modules	***Number of equations***	***Average size eq.***
none	2	18 bytes
1	6	63 bytes
2	10	74 bytes
3	15	72 bytes
4	20	71 bytes
4	30	41 bytes
4	40	25 bytes

Of course, the more equations in the system the more bytes it will probably take to define each. The program is also fast, the loops are designed to run as fast as possible by duplicating indirect addresses. Computing times depend mostly on the number and complexity of the functions you define but the following times are typical:

Number of equations	1	2	3	5
Time per point	12 sec.	22 sec.	32 sec.	62 sec.

2. Program Listing

```
♦LBL "RK"
CLRG
FIX 0
CF 29
SF 21
"N=?"
PROMPT
STO 04
 3
 *
22
 +
STO 05
SF 25
ARCL IND X
 1
STO 11
STO 07
STO 08
 +
FS?C 25
GTO 00 ►
"SIZE"
ARCL X
PROMPT
♦LBL 00
 2
STO 06
STO 09
STO 00
ENTER↑
SQRT
LASTX
 +
STO 01
 /
STO 02
RCL 04
ST- 05
20
 +
ST+ 11
1 E3
 /
20
 +
STO 14
STO 10
♦LBL 01
"Y"
ARCL 12
├"=?"
PROMPT
STO IND 14
ISG 12
X<>Y
ISG 14
GTO 01 ►
"INC=?"
PROMPT
STO 12
♦LBL 14
.002
STO 13
 6
STO 04
STO 03
♦LBL 11
RCL 10
STO 14
RCL 11
STO 15
RCL 05
STO 16
♦LBL 12
XEQ IND 14 ►
RCL 12
 *
STO IND 15
ISG 15
X<>Y
ISG 14
GTO 12 ►
RCL 11
STO 15
RCL 10
STO 14
♦LBL 13
RCL IND 15
RCL IND 13
 /
 3
FS? 29
ST* Y
RCL IND 16
RCL IND 04
 *
RCL IND 15
 -
RCL IND 13
 /
ST- IND 14
 *
 +
ST- IND 16
 1
ST+ 15
ST+ 16
ISG 14
GTO 13 ►
FS?C 29
GTO 10 ►
ST+ 04
ISG 13
GTO 11 ►
SF 29
GTO 11 ►
♦LBL 10
RCL 10
 0
ADV
♦LBL 02
"Y"
FIX 0
ARCL X
├"="
FIX 7
ARCL IND Y
AVIEW
ISG X
X<>Y
ISG Y
GTO 02 ►
GTO 14 ►
♦LBL 20
 1
RTN
.END.
```

The symbols ♦ and ► are purely cosmetic, to indicate branching; ├ is the Append alpha function

2.1 Contents of the storage registers

Register	Contents	Register	Contents	Register	Contents
00	$a_1 = 2$	10	add.f,y	20	$y_0 = x$
01	$a_2 = 3.41+$	11	add.k	...	...
02	$a_3 = 0.58+$	12	h	N+20	y_n
03	$a_4 = 6$	13	add.a	N+21	k_0
04	N, add.b	14	add.f,y var.	...	...
05	add q = 2N+22	15	add k var.	2N+21	k_n
06	$b_1 = 2$	16	add q var.	2N+22	q_0
07	$b_2 = 1$	17	*free*	...	...
08	$b_3 = 1$	18	*free*	3N+22	q_n
09	$b_4 = 2$	19	*free*		

3. Usage Instructions

To solve a system of N first-order differential equations perform the following *Steps*:

Step 1: in **RUN** Mode: **GTO .138** → *{ switch to* **PRGM** *Mode }.*

If there are definitions left from previously solved systems, execute:

DEL 999 → *{ you should see:* `137 RTN` *}*

Step 2: still in **PRGM** Mode, define the equations. Every equation f_i must be defined under a *numerical label* whose number is $i+20$. This is: f_1 must be defined under `LBL 21`, f_7 under `LBL 27`, f_{14} under `LBL 34`, and so on. Each definition must be terminated with a `RTN` instruction.

To define each equation, y_i can be found stored in register R_{i+20} and the convention $y_0 = x$ is used. This is: x (= y_0) can be found in R_{20}, y_1 in R_{21}, ..., y_{14} in R_{34} , and so on.

You may use R_{17}, R_{18} and R_{19} as scratch registers if you wish but do not disturb the contents of any other registers from register R_{00} up to R_{3N+22} , both included. See ***Examples***.

Step 3: once the equations have been defined in program memory, switch back to **RUN** Mode and execute the program, as follows:

	XEQ "RK"	→	*N=?*	*{ asks for the number of equations in the system}*
N	**R/S**	→	*Y0=?*	*{ remember the convention* $y_0 = x_0$ *, so it's asking for* x_0 *}*
x_0	**R/S**	→	*Y1=?*	*{ asks for* $y_1 (x_0)$ *}*
y_1	**R/S**	→	*Y2=?*	*{ asks for* $y_2 (x_0)$ *}*
...	...	...	...	...
y_n	**R/S**	→	*INC=?*	*{ asks for the increment [step]* h *}*

Note: the increment (step) h should be adequately small. The smaller is h, the greater is the accuracy and number of stages required to reach a given value of x (and so the longer it will take and the more rounding errors will accumulate). The error per stage is approximately proportional to h^5 so as a rule of thumb $h=0.1$ should give about 5 correct places at the end of 10 stages or so.

h	**R/S**	→	*{ computation begins }*

After some time, the values of $y_i (x_0 + h)$ will be output. If there's a printer attached and *On*, all values will be printed and spaced and computation will proceed to the following stage automatically. Otherwise, simply press **R/S** after each displayed value to see the next one or resume the procedure, like this:

	→	*Y0=its value*	*{ remember,* $y_0 = x_0$ *, so this is* $x = x_0 + h$ *in this stage}*
R/S	→	*Y1=its value*	*{ outputs* $y_1 (x_0 + h)$ *}*
R/S	→	*Y2=its value*	*{ outputs* $y_2 (x_0 + h)$ *}*
... ...	...	...	
R/S	→	*YN=its value*	*{ outputs* $y_n (x_0 + h)$*, the last value in this stage }*
R/S	→	*Y0=its value*	*{ proceeds to compute the next stage,* $y_i (x_0 + 2h)$ *}*

and so on. To stop the procedure at any time, press **R/S**. For another system, go to *Step 1*.

Notes:

- if after introducing N a message *SIZE nnn* does show up this means the current register allocation is insufficient to run the program, so simply execute **SIZE** *nnn* and then press **R/S** to resume.
  In general, if your system has N equations, a minimum **SIZE** of *3N+23* registers is required.

- *do not disturb the stack while the output is taking place*. If you want to simply view (not write down) each value, without having to press **R/S** each time to continue (you have no printer attached), then at any moment stop the computation by pressing **R/S** and execute **CF 21** , then **R/S** again to resume. Subsequent values will be displayed without stopping program execution.

- if you want to change the spacing (step) ***h*** during the process, wait for the program to stop or output something, then stop (if not yet halted) and input the new ***h*** like this:

  new value of ***h*** **STO 12** **R↓** **R/S** .

  the procedure will resume from where it left but this time using the new ***h***.

- to change the number of decimals in the output, change line ***127*** **FIX 7** to the desired display setting.

4. Examples

The following examples can be useful to check that the program is correctly entered and to understand its usage.

4.1 Example 1

Solve the system:

$y_1' = \sin x - y_2$		$x_0 = 0.230253487$
$y_2' = e^x + y_3$	with initial conditions:	$y_1 = -0.258919089$
$y_3' = 1 - y_1 - y_2$		$y_2 = 1.487143417$
		$y_3 = 0.973608574$

using a step $h = -0.102342187$.

Define the equations: in **RUN** Mode, **GTO .138** , *switch to* **PRGM** *Mode,* **DEL 999** → see: ***137*** **RTN**
Enter the definitions below, and then switch to **RUN** Mode, execute **PACK** , **RAD** and run the program:

138 ♦LBL 21	144 ♦LBL 22	150 ♦LBL 23
139 RCL 20	145 RCL 20	151 1
140 SIN	146 E↑X	152 RCL 21
141 RCL 22	147 RCL 23	153 -
142 -	148 +	154 RCL 22
143 RTN	149 RTN	155 -
		156 RTN

XEQ "RK"		→	*N=?*
3	**R/S**	→	*Y0=?*
0.230253487	**R/S**	→	*Y1=?*
-0.258919089	**R/S**	→	*Y2=?*
1.487143417	**R/S**	→	*Y3=?*
0.973608574	**R/S**	→	*INC=?*
-0.102342187	**R/S**	→	*computing proceeds and then ...*

... the output begins	→	*Y0=0.1279113*	*{ remember, $y_0 = x_0$, so this is $x = x_0 + h$ in this first stage}*
`R/S`	→	*Y1=-0.1364522*	*{ outputs $y_1 (x_0 + h)$ }*
`R/S`	→	*Y2=1.2640152*	*{ outputs $y_2 (x_0 + h)$ }*
`R/S`	→	*Y3=0.9918304*	*{ outputs $y_3 (x_0 + h)$, the last value in this stage }*
`R/S`	→	*Y0=0.0255691*	*{ this is $x = x_0 + 2h$ in this second stage}*
`R/S`	→	*Y1=-0.0258989*	*{ outputs $y_1 (x_0 + 2h)$ }*
`R/S`	→	*Y2=1.0514656*	*{ outputs $y_2 (x_0 + 2h)$ }*
`R/S`	→	*Y3=0.9996729*	*{ outputs $y_3 (x_0 + 2h)$, the last value in this stage, and so on ... }*

To check the accuracy obtained, the exact solution is:

$y_1 = 1 - e^x$		$y_1 (x_0 + h) = -0.136452195$	
$y_2 = e^x + \sin x$	thus:	$y_2 (x_0 + h) = 1.264014981$	, so we got 7-8 correct places
$y_3 = \cos x$		$y_3 (x_0 + h) = 0.991830497$	

4.2 Example 2

Solve the system:

$y_1' = y_1 - y_2 + e^x - y_4 - x$		$x_0 = 0$
$y_2' = y_1 - \sin x + e^x$		$y_1 = 1$
$y_3' = \cos x - y_3 - y_4 - x$	with initial conditions:	$y_2 = 1$
$y_4' = y_3 - e^{-x} - 1$		$y_3 = 2$
$y_5' = (y_5 + \sin x - y_4)^2$		$y_4 = y_5 = 0$

Using a step $h = 0.1$, find the values of $y_1 .. y_5$ for $x = 1$.

Define the equations: in **RUN** Mode, `GTO .138` , *switch to* **PRGM** *Mode,* `DEL 999` → see: `137 RTN`
Enter the 5 definitions below, and then switch to **RUN** Mode, execute `PACK` , `RAD` and run the program:

138 ♦LBL 21	150 ♦LBL 22	159 ♦LBL 23	169 ♦LBL 24	178 ♦LBL 25
139 RCL 21	151 RCL 21	160 RCL 20	170 RCL 23	179 RCL 25
140 RCL 22	152 RCL 20	161 COS	171 RCL 20	180 RCL 20
141 -	153 SIN	162 RCL 23	172 CHS	181 SIN
142 RCL 20	154 -	163 -	173 E↑X	182 +
143 E↑X	155 RCL 20	164 RCL 24	174 -	183 RCL 24
144 +	156 E↑X	165 -	175 1	184 -
145 RCL 24	157 +	166 RCL 20	176 -	185 X↑2
146 -	158 RTN	167 -	177 RTN	186 RTN
147 RCL 20		168 RTN		
148 -				
149 RTN				

`XEQ`	"RK"	→	*N=?*
5	`R/S`	→	*SIZE 38* [*]
`SIZE 38`	`R/S`	→	*Y0=?*
0	`R/S`	→	*Y1=?*
1	`R/S`	→	*Y2=?*
1	`R/S`	→	*Y3=?*
2	`R/S`	→	*Y4=?*
0	`R/S`	→	*Y5=?*
0	`R/S`	→	*INC=?*
0.1	`R/S`	→	*computation proceeds and then ...*

[*] The example assumes there were less than 38 registers allocated so it prompts for the user to allocate them right now, before resuming.

... the output begins	→	*Y0=0.1*	{ *remember,* $\boldsymbol{y_0 = x_0}$ *, so this is* $\boldsymbol{x = x_0 + h}$ *= 0.1 in this first stage*}
R/S	→	*Y1=1.0948375*	{ *outputs* $\boldsymbol{y_1}$ *(0.1)* }
R/S	→	*Y2=1.2050044*	{ *outputs* $\boldsymbol{y_2}$ *(0.1)* }
R/S	→	*Y3=1.8998416*	{ *outputs* $\boldsymbol{y_3}$ *(0.1)* }
R/S	→	*Y4=-0.0001665*	{ *outputs* $\boldsymbol{y_4}$ *(0.1)* }
R/S	→	*Y5=0.0003345*	{ *outputs* $\boldsymbol{y_5}$ *(0.1), the last value in this stage* }
R/S	→	...	{ *proceeds to compute the next stage,* $\boldsymbol{y_i}$ *(0.2)* }

After 10 such stages you should get:

$x = 1.0000000$			
$y_1 = 1.3817719$		$y_1 = \sin x + \cos x$	
$y_2 = 3.5597527$	and the exact solution is:	$y_2 = \sin x + e^{x}$	so we got 5-6 correct places
$y_3 = 0.9081817$		$y_3 = \cos x + e^{-x}$	
$y_4 = -0.1585284$		$y_4 = \sin x - x$	
$y_5 = 0.5573977$		$y_5 = \tan x - x$	

Notes

*1.*This program will also run in the *HP42S* but as it has a wider numeric range and greater precision (12-digit instead of 10-digit), the results might differ slightly. Consult the *HP42S Owner's Manual* for other minor differences (e.g.: naming).

*2.*This program was published in *PPC Melbourne Chapter Technical Notes V1N2 pp23-28 (September 1980).* However, there's a missing minus sign ("–") between e^{x} and y_4 in the definition of the first equation of the second example.

3. It was also published in *PPC Calculator Journal V8N1 pp17-19 (Jan./Feb. 1981).* However, there's also a missing minus sign ("–") between e^{x} and y_4 in the definition of the first equation of the second example.

References

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

Copyrights