SYSNLEQ – Nonlinear Equations Systems

Abstract

SYSNLEQ is a program written in 1979 for the HP-34C programmable calculator to solve a system of two arbitrary nonlinear equations in two variables. Will also work as-is in the HP-67/97 and in the HP-15C and HP-41C series with minimal changes. Two worked examples are included.

Keywords: *system, nonlinear, equation, Newton's method, programmable calculator, RPN, HP-34C, HP-67, HP-97, HP-41C, HP-15C*

1. Introduction

SYSNLEQ is a *102-step* (plus the steps required to compute both user-defined functions) *RPN* program written in 1979 for the *HP-34C* calculator (will also run *as-is* in the *HP-67/97* and with minor changes *(see **Note 4**)* in other RPN models such as the *HP-15C* and *HP-41C*), which solves systems of two arbitrary nonlinear equations in two variables by iteratively using *Newton's method*. We proceed like this:

Consider a system of two nonlinear equations in two variables $\boldsymbol{x}$, $\boldsymbol{y}$, of the generic form

$$f(x, y) = 0, \quad g(x, y) = 0$$

where $\boldsymbol{f}$ and $\boldsymbol{g}$ are arbitrary user-defined functions. Starting from an initial guess ($\boldsymbol{x_0}$, $\boldsymbol{y_0}$) this program attempts to find a real solution ($\boldsymbol{x}$, $\boldsymbol{y}$) using *Newton's method* iteratively, as follows:

$$x_{i+1} = x_i + \delta x_i \quad , \qquad y_{i+1} = y_i + \delta y_i \qquad \text{where } \delta x_i = \frac{\begin{vmatrix} -f & f_y \\ -g & g_y \end{vmatrix}}{\begin{vmatrix} f_x & f_y \\ g_x & g_y \end{vmatrix}} \text{ , } \delta y_i = \frac{\begin{vmatrix} f_x & -f \\ g_x & -g \end{vmatrix}}{\begin{vmatrix} f_x & f_y \\ g_x & g_y \end{vmatrix}}$$

and $f = f(x_i, y_i)$, $g = g(x_i, y_i)$,

$$f_x = \frac{\partial f(x,y)}{\partial x}\Big] x_i, y_i \approx \frac{f(x_i,y_i) - f(x_i - \Delta x, y_i)}{\Delta x}$$

$\Delta x = \Delta y = 8.10^{-4}$, $\varepsilon = 10^{-8}$

$$f_y = \frac{\partial f(x,y)}{\partial y}\Big] x_i, y_i \approx \frac{f(x_i,y_i) - f(x_i, y_i - \Delta y)}{\Delta y}$$

$$g_x = \frac{\partial g(x,y)}{\partial x}\Big] x_i, y_i \approx \frac{g(x_i,y_i) - g(x_i - \Delta x, y_i)}{\Delta x}$$

$$g_y = \frac{\partial g(x,y)}{\partial y}\Big] x_i, y_i \approx \frac{g(x_i,y_i) - g(x_i, y_i - \Delta y)}{\Delta y}$$

iterating until $\dfrac{|\delta x_i| + |\delta y_i|}{|x_{i+1}| + |y_{i+1}|} < \varepsilon$.

As in the case of *Newton's method* applied to a single equation in one variable, the procedure may fail to converge for a number of reasons, including a bad initial guess (you might want to try another guess, see ***Usage Instructions***), hitting stationary points, looping between two values, zero or near-zero partial derivatives, divergence, etc., as well as unduly slow convergence if the system has roots with multiplicity greater than one, or even the system just not having any real solutions. See ***Note 2***.

These cases happen more frequently than when solving a single equation (a good guess is difficult to come by) and dealing with them is beyond the scope of this paper but you can consult other sources, see ***References***.

2. Program Listing

```
01 ♦LBL A      22 STO 5       43 GSB 0 ▸     64  x          85 ABS
02 SF 0        23  -          44 ST0- 7      65  -          86 RCL 0
03 STO 1       24 STO 4       45 RCL 4       66 STO 5       87 ABS
04 x↔y         25 GSB 0 ▸     46 RCL 0       67  ÷          88  +
05 STO 0       26 STO- 6      47 GSB 1 ▸     68 STO+ 0      89  ÷
06 GSB 0 ▸     27 RCL 1       48 STO- 9      69 ABS         90 EEX
07 STO 2       28 RCL 4       49 RCL 5       70 STO 4       91 CHS
08 STO 6       29 GSB 1 ▸     50 STO÷ 7      71 RCL 2       92  8
09 STO 7       30 STO- 8      51 STO÷ 9      72 RCL 8       93 x>y
10 RCL 1       31 RCL 5       52 RCL 3       73  x          94 CF 0
11 RCL 0       32 STO÷ 6      53 RCL 7       74 RCL 3       95 RCL 0
12 GSB 1 ▸     33 STO÷ 8      54  x          75 RCL 6       96 PSE
13 STO 3       34 RCL 1       55 RCL 2       76  x          97 RCL 1
14 STO 8       35  8          56 RCL 9       77  -          98 PSE
15 STO 9       36 EEX         57  x          78 RCL 5       99 F? 0
16 RCL 1       37  4          58  -          79  ÷         100 GTO A ▸
17 RCL 0       38 CHS         59 RCL 6       80 STO+ 1     101 x↔y
18  8          39 STO 5       60 RCL 9       81 ABS        102 RTN
19 EEX         40  -          61  x          82 STO+ 4
20  4          41 STO 4       62 RCL 7       83 RCL 4     103 ♦LBL 0
21 CHS         42 RCL 0       63 RCL 8       84 RCL 1     ... ♦LBL 1
```

- *102 steps + f(x,y), g(x,y)*
- *uses registers R_0 - R_9*
- *uses labels A,0,1 and flag 0*
- *FIX 4 or SCI 4 recommended*
- *RAD mode recommended*
- *the symbols ♦ and ▸ are purely cosmetic, to indicate branching*
- *define f(x,y) under this label*
- *define g(x,y) under this label*

3. Usage Instructions

Step 1: In **PRGM** Mode, enter under ♦LBL 0 the sequence of steps which defines ***f(x,y)***, ending the definition with RTN , and immediately following it enter under ♦LBL 1 the sequence of steps which defines ***g(x,y)***, ending the definition with either RTN or just the end of program memory. For the *HP-34C*, you have a maximum of 38 program steps available for defining both functions.

For definition purposes, the value of ***x*** is both in stack register ***X*** and in $\boldsymbol{R_0}$, while the value of ***y*** is both in stack register ***Y*** and in $\boldsymbol{R_1}$. The definitions can use all available storage registers after $\boldsymbol{R_9}$.

Step 2: In **RUN** Mode, input the initial guess (x_0, y_0) and press **A** to compute the solution:

x_0 **ENTER** y_0 **A** → x_1 → y_1 *(first iteration, paused for a second)*
→ x_2 → y_2 *(second iteration, ditto)*
... ...
→ x_n → y_n *(last iteration, ditto)*
→ x *(the program ends with the final value of **x** in the display and $\boldsymbol{R_0}$)*
x ↔ y → y *(the final value of **y**, left both in stack register **Y** and in $\boldsymbol{R_1}$)*

In case of the iterations not converging, stop the program by pressing **R/S** and go to *Step 3*.

Step 3: To search for another solution or to change the initial guess in case of non-convergence, go to *Step 2* above to try another initial guess, or else decide that the system doesn't have a real solution.

To solve another system, go to *Step 1* above to define another ***f(x,y)*** and ***g(x,y)*** but before entering the new definitions you must *delete* the previous definitions from program memory, if any.

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

Find the intersection point between the circle $x^2 + y^2 - 5x + y - 6 = 0$ and the hyperbola $x^2 - y^2 + 5x - y - 2 = 0$ near the point ***(1, 1)***.

In **PRGM** Mode enter this sequence of steps which defines the circle's equation under ♦LBL 0 and the hyperbola's equation under ♦LBL 1 :

```
103 ♦LBL 0        117 ♦LBL 1
104  x²           118  x²
105 x↔y           119 x↔y
106  x²           120  x²
107  +            121  -
108 RCL 0         122 RCL 0
109  5            123  5
110  x            124  x
111  -            125  +
112 RCL 1         126 RCL 1
113  +            127  -
114  6            128  2
115  -            129  -
116 RTN           130 RTN
```

In **RUN** *Mode,* **FIX 4***, enter the initial guess* $x_0=1$, $y_0=1$ *and compute the solution:*

1 **ENTER** 1 **A**	→	*2.5006*	→	*3.5010*	*(1st iteration)*
	→	*2.0500*	→	*3.3217*	*(2^{nd} iteration)*
	→	*2.0006*	→	*3.0048*	*(3rd iteration)*
	→	*2.0000*	→	*2.9997*	*(4th iteration)*
	→	*2.0000*	→	*3.0000*	*(5th iteration)*
		. . .		. . .	*(five more iterations)*
	→	*2.0000*			*(program halts, this is* ***x****)*

FIX 9	→	***2.000000000*** *(the computed value of* ***x****)*
x ↔ ***y***	→	***2.999999993*** *(the computed value of* ***y****)*

So, ignoring the negligible rounding errors, the intersection is at the point ***(2, 3)***.

If desired, check that the computed values are actually a solution by evaluating ***f(x,y)*** and ***g(x,y)*** as follows:

RCL 1 **RCL 0** **GSB 0**	→	***-0.000000047***	*(the value of* ***f(x,y)****, which is very close to 0, as expected)*
RCL 1 **RCL 0** **GSB 1**	→	***0.000000049***	*(the value of* ***g(x,y)****, ditto)*

4.2 Example 2

Find a complex root of the equation $2^z - 3z + 2+4i = 0$.

Let's consider a generic complex root $z = x+y\,i$. Substituting in the equation, we have:

$$2^{x+y\,i} - 3(x+y\,i) + 2+4\,i = 0$$

As $2^{x+y\,i} = 2^x(\cos(y \operatorname{Ln} 2)+i \operatorname{sen}(y \operatorname{Ln} 2))$, the equation becomes:

$$2^x(\cos(y \operatorname{Ln} 2)+i \operatorname{sen}(y \operatorname{Ln} 2)) - 3x - 3y\,i + 2+4\,i = 0$$

Now, separating the real and imaginary parts and rearranging a little we get this nonlinear system of two equations in the two real variables x, y :

$$3x - 2^x \cos(y \operatorname{Ln} 2) - 2 = 0$$
$$3y - 2^x \operatorname{sen}(y \operatorname{Ln} 2) - 4 = 0$$

In **PRGM** Mode, first make sure to *delete* all steps remaining from previous definitions (if any), then enter this sequence of steps which defines both equations under ♦LBL 0 and ♦LBL 1 , respectively:

```
103 ♦LBL 0
104   3
105   x
106   2
107  RCL 0
108   y^x
109  RCL 1
110   2
111  LN
112   x
113  COS
114   x
115   -
116   2
117   -
118  RTN
```

```
119 ♦LBL 1
120   x↔y
121   3
122   x
123   2
124  RCL 0
125   y^x
126  RCL 1
127   2
128  LN
129   x
130  SIN
131   x
132   -
133   4
134   -
135  RTN
```

In **RUN** *Mode,* **FIX 4**, **RAD**, *use as initial guess* $x_0=1$, $y_0=1$ *and find the solution:*

1 **ENTER** 1 **A**	→	*1.1795*	→	*1,7593*	*(1st iteration)*
	→	*0.9265*	→	*2.0422*	*(2^{nd} iteration)*
	→	*0.7646*	→	*1.9593*	*(3rd iteration)*
	→	*0.7862*	→	*1.8869*	*(4th iteration)*
	→	*0.8161*	→	*1.8884*	*(5th iteration)*
		. . .		. . .	*(... slow convergence ...)*
	→	*0.8136*			*(program halts, this is* ***x****)*
FIX 9	→	***0.813591019***			*(this is* ***x*** *, the real part)*
x ↔ ***y***	→	***1.900471318***			*(this is* ***y*** *, the imaginary part)*

So, the complex root is ***z = 0.813591019 + 1.900471318 i*** *, let's check the result:*

RCL 1 **RCL 0** **GSB 0**	→	***0.000000017***	*(* ***f(x,y)*** *is very close to 0)*
RCL 1 **RCL 0** **GSB 1**	→	***0.000000009***	*(* ***g(x,y)*** *is very close to 0 too)*

Notes

*1.*This program will run *as-is* on the *HP-67/97*, no changes required. It will also run in the *HP-41C* series by just changing all **GSB** instructions to **XEQ**. In the *HP-15C* it can be optimized further by using **RCL** arithmetic.

*2.*As stated in the ***Introduction***, the algorithm used may fail to produce a solution for many reasons, resulting in divergence, or the convergence might be very slow. It's usually not easy to come by with a good initial guess that will result in convergence (never mind fast convergence) despite *Newton's method* theoretical quadratic convergence. There's little that can be done in a RAM-limited, relatively slow calculator such as the *HP-34C* and nevertheless, as one authority put it, the solution of nonlinear systems of equations is one the most difficult problems in scientific computation.

*3.*This *102+ step* program effectively demonstrates the benefits of the *HP-34C*'s *expandable* program memory. For instance, though the *HP-29C* has more RAM, it can't be allocated flexibly and it's limited to 98 steps, so this program won't fit in, let alone the definitions for *f(x,y)* and *g(x,y)*.

*4.*This program is published in the *Hewlett-Packard's Solution Book "HP-34C Matemática Avanzada" (Spanish)*. The program listing there includes keycodes for all steps and fully commented source code.

References

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

Valentin Albillo (1980). *Hewlett-Packard's Solution Book "HP-34C Matemática Avanzada"*

Copyrights