RF – Finding Real Roots of Equations

© 2019 Valentín Albillo

Abstract
RF is a program written in 1980 for the HP-41C to find real roots of an arbitrary user-supplied equation \( f(x) = 0 \) using Newton’s method and a user-given initial guess. Interactive and non-interactive versions provided. Five worked examples are included.

Keywords: root finder, solving equations, Newton’s method, programmable calculator, RPN, HP-41C, HP-41CV, HP-41CX, HP42S

1. Introduction
RF is a short (42 steps) RPN program that I wrote in 1980 for the HP-41C programmable calculator (will also run as-is in the HP-41CV/CX and the HP42S), which will try to find a real root of an user-supplied equation \( f(x) = 0 \) using Newton’s method and some user-provided initial guess.

The procedure is as follows: given an equation \( f(x) = 0 \) and an initial guess for the root, \( x_0 \), Newton’s method produces a hopefully improved guess \( x_1 \), computed this way:

\[
x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}
\]

where \( f'(x) \) is the derivative of \( f(x) \), which is numerically approximated like this:

\[
f'(x) \approx \frac{f(x + h) - f(x)}{h}
\]

where \( h \) is a suitably small value (\( \sim 0.0003 \) is used here). The process is iterated with \( x_1 \) replacing \( x_0 \) to produce a further improved guess \( x_2 \) and so on until it either converges to the root or else 50 iterations elapse without achieving convergence. This 50-iteration max. limit prevents endless loops and guarantees termination.

2. Program Listing

```
01 LBL "RF" 12 50 23 XEQ IND 00 34 RND - 42 steps, 75 bytes
02 "NAME?" 13 STO 03 24 X=0? 35 X=0? - requires at least SIZE 004
03 AON 14 LBL 00 25 GTO 01 36 GTO 01 - uses flag 00 and Alpha register
04 PROMPT 15 RCL 02 26 ST- 01 37 DSE 03 - does not alter display settings
05 AOFF 16 1 27 RCL 01 38 GTO 00 42 END
06 ASTO 00 17 D-R 28 X=0? 39 SF 00
07 "X0?" 18 D-R 29 SIGN 40 LBL 01 - to get I press the [+] key
08 PROMPT 19 + 30 / 41 RCL 02 - the symbols ♦ and ▶ are purely cosmetic, to indicate branching
09 LBL "RFP" 20 XEQ IND 00 31 D-R 42 END
10 CF 00 21 STO 01 32 D-R
11 STO 02 22 RCL 02 33 ST- 02
```

3. Usage Instructions
The program can be used both interactively and programmatically, as follows:

1) **Interactively:** in RUN mode, set the precision you need (see below) and call “RF” (Root Finder). The program will prompt for the name of the program which defines \( f(x) \) and for the initial guess \( x_0 \). Once
provided, the program will proceed to compute the root and the result will be:

- if the process **converges**, the root will be in the display (stack register X) and flag 00 will be **clear**.
- if the process **does not converge** after 50 iterations, the latest guess \((x_{50})\) will be in the display and flag 00 will be **set** to indicate nonconvergence. In that case you can then either rerun the program using a different initial guess \(x_0\), or else decide that no real root exists.

2) **Programmatically**: your program must set the precision needed (see below) and call “RFP” (**Root Finder Programmable**), which assumes that the name of the program which defines \(f(x)\) is stored in register \(R_00\) and the initial guess \(x_0\) is in stack register \(X\), so your program must place them there before making the call. Upon returning, your program must check the outcome by testing flag 00:

- if flag 00 is **clear**, a root was found and it will be in stack register \(X\) (and also in \(R_02\)).
- if flag 00 is **set**, the process didn’t converge and no root was found. You’ll find the latest guess, \(x_{50}\), in stack register \(X\). Your program must then decide what to do next (i.e.: reporting the failure to the user, try another initial guess and call “RFP” again, call “RFP” once more to perform additional iterations continuing from \(x_{50}\) (which is already in \(X\)), try another approach, etc.)

Apart from being called programmatically, “RFP” can also be useful when searching for roots (multiple, elusive) after the very first attempt, as the name of \(f(x)\) is already stored so just simply key in your new initial guess and call “RFP”, thus avoiding all the prompts. See Example 1 below.

In both cases you need to write a program to define \(f(x)\), the equation to solve. It must be an independent program under its own global label, must assume that the argument \(x\) is in stack register \(X\) upon being called, and must compute and leave the corresponding value of \(f(x)\) in stack register \(X\).

The **accuracy** depends on the display setting, **FIX n**. The greater \(n\), the better the accuracy and the longer the time required to achieve it, though most times the computed root will be **more** accurate than specified. As a useful **rule of thumb**, if you need just 2 or 3 places, set **FIX 2**. Conversely, if you need full accuracy set **FIX 7**.

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 two nearby roots of the transcendental equation: \(e^x - 5x + 3 = 0\)

In **PRGM** Mode, enter the following 9-step program to define \(f(x)\):

```
01 LBL "FX1" 05 * 09 END
02 E↑X 06 -
03 LASTX 07 3
04 5 08 +
```

In **RUN** Mode, first set **FIX 7** for maximum accuracy and then first call the interactive version of **Root Finder (RF)** with initial guess 1, then call the non-prompting one (**RFP**) with guess 2 to find the second nearby root:

```
FIX 7
XEQ "RF" NAME? "FX1" [R/S] X0? 1 [R/S] 1.4688293 (1st root, internally accurate to 9 places)
2 XEQ "RFP" 1.7437520 (2nd root, also internally accurate to 9 places; calling RFP saves the unneeded prompts)
```
4.2 Example 2
Find a double root of the quadratic equation: \(x^2 - 4x + 4 = 0\)

In **PRGM** Mode, enter the following 9-step program to define \(f(x)\):

```
01 LBL "FX2" 05  *  09  END
02 X↑2  06  -
03 LASTX  07  4
04 4  08  +
```

In **RUN** Mode, still using **FIX 7** for maximum accuracy, call the interactive version of **Root Finder** with initial guess 1 to find the double root:

```
XEQ "RF" NAME? "FX2" [R/S] X0? 1 [R/S] 1.9999908 (double root, accurate to 6 places)
```

Double roots can usually be found to only 5-6 places in 10-digit machines, but \(f(1.9999908)\) is 0 to 10 places.

4.3 Example 3
Find a real root of the quadratic equation: \(x^2 + 1 = 0\)

In **PRGM** Mode, enter the following 5-step program to define \(f(x)\):

```
01 LBL "FX3" 04  +
02 X↑2  05  END
03 1
```

In **RUN** Mode, still using **FIX 7** for maximum accuracy, call the interactive version of **Root Finder** with initial guess 0 to attempt to find a real root:

```
XEQ "RF" NAME? "FX3" [R/S] X0? 0 [R/S] -3.3380759 and flag 00 is set.
```

This means that the process did not converge after 50 iterations, so no root was found, flag 00 was set and the latest guess (-3.3380759) was returned. Actually, this equation has no real roots.

4.4 Example 4
Find a root of the equation \(e^x - 2 = 0\), using 90 as the (very bad) initial guess, to illustrate what happens:

In **PRGM** Mode, enter the following 5-step program to define \(f(x)\):

```
01 LBL "FX4" 04  -
02 E↑X  05  END
03 2
```

In **RUN** Mode, still using **FIX 7** for maximum accuracy, call the interactive version of **Root Finder** with the (very poor) initial guess 90 to attempt to find the root:
This means that the process did not converge after 50 iterations, so no root was found. flag 00 was set and the latest guess (40.0080372) was returned, but there is a root and the reason it wasn’t found is because the initial guess (90) is very far from the root and the exponential function is nearly vertical at those values. However, with the (also very poor) latest guess still in the display, calling now “RFP” (the non-prompting version) succeeds:

```
XEQ "RFP" 0.6931472 and flag 00 is clear so the root (correct to 10 digits) was indeed found this time.
```

4.5 Example 5

Write a program to compute for \( x \geq 1 \) the function \( y = \text{LambertW}(x) \), which is defined implicitly as: \( ye^y = x \)

**In PRGM Mode**, enter the following programs which define \( \text{LambertW} \) and the equation to solve, respectively:

```
01 \textbf{LBL} "LambW" 08 \textbf{FSIC} 00 01 \textbf{LBL} "Yey"  
02 \textbf{STO} 04 09 \textbf{AVIEW} 02 \textbf{ENTER} \uparrow  
03 "Yey" 10 \textbf{END} 03 \textbf{E} \uparrow x  
04 \textbf{ASTO} 00 04 *  
05 \textbf{LN} 05 \textbf{RCL} 04  
06 \textbf{XEQ} "RFP" 06 -  
07 "NOT FOUND" 07 \textbf{END}  
```

**LAMBW** simply stores \( x \) in \( R_{44} \), the name of the equation in \( R_{46} \) (“Yey”), places the initial guess (\( \ln(x) \) does fine) in stack register \( X \), then calls **RFP** (the non-prompting version) to compute the root, \( y \). Upon returning, it checks whether no root was found (flag 00 is set), in which case it shows the message “NOT FOUND”; else, it simply stops with the value of the root \( y \) in the display (i.e.: stack register \( X \)).

**In RUN Mode**, using **FIX 7** for maximum accuracy, compute \( \text{LambertW} \) for \( x = 1, 2, 3, 10000 \):

<table>
<thead>
<tr>
<th>( x )</th>
<th>( \text{LambertW} )</th>
<th>Check</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5671433</td>
<td>( [\text{ENTER}] [e^x] \times ) 1.000000</td>
</tr>
<tr>
<td>2</td>
<td>0.8526055</td>
<td>( [\text{ENTER}] [e^x] \times ) 2.000000</td>
</tr>
<tr>
<td>3</td>
<td>1.0499089</td>
<td>( [\text{ENTER}] [e^x] \times ) 3.000000</td>
</tr>
<tr>
<td>10000</td>
<td>7.2318460</td>
<td>( [\text{ENTER}] [e^x] \times ) 10,000.000</td>
</tr>
</tbody>
</table>

**Notes**

1. If a real root exists Newton’s method usually converges quadratically to it, i.e. once the convergence starts the number of correct digits doubles after each iteration, unless the root’s multiplicity is \( \geq 1 \) in which case the convergence reduces to linear.
2. Once a root is found, displayed, and execution stops, it’s also stored in \( R_{42} \) so that it can be reused in further calculations.
3. If evaluating the derivative \( f’(x) \) ever results in \( 0 \), a division by 0 error would ensue, but the program avoids it by using the value \( I \) instead so that no error arises and the search moves on to another place. This usually happens at a minimum of \( f(x) \).
4. If you can use synthetic instructions, you may replace registers \( R_{44}, R_{42} \) and \( R_{43} \) by registers \( M, N, \) and \( O \) respectively (i.e.: \( \text{STO} 01 \) becomes \( \text{STO} M \) and so on), and insert step 42 CLA just before \( \text{END} \) to clear the Alpha register before the program ends. After this, program length will be 43 steps (83 bytes) and min. SIZE 001, thus saving 3 registers for other uses at no cost.
5. This program (RF, RFP) was duly submitted for inclusion in the PPC ROM but it wasn’t accepted.

**References**


**Copyrights**

Copyright for this paper and its contents is retained by the author. Permission to use it for non-profit purposes is granted as long as the contents aren’t modified in any way and the copyright is acknowledged.