RF – Finding Real Roots of Equations

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:

$$x_1 = x_0 - f(x_0)/f'(x_0)$$

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

$$f'(x) \sim \frac{f(x+h) - f(x)}{h}$$

where h is a suitably small value *(~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

```
♦LBL "RF"
"NAME?"
AON
PROMPT
AOFF
ASTO 00
"X0?"
PROMPT
♦LBL "RFP"
CF 00
STO 02
50
STO 03
♦LBL 00
RCL 02
1
D-R
D-R
+
XEQ IND 00
STO 01
RCL 02
XEQ IND 00
X=0?
GTO 01 ►
ST- 01
RCL 01
X=0?
SIGN
/
D-R
D-R
ST- 02
RND
X=0?
GTO 01 ►
DSE 03
GTO 00 ►
SF 00
♦LBL 01
RCL 02
END
```

- 42 steps, 75 bytes
- requires at least SIZE 004
- uses flag 00 and Alpha register
- does not alter angular mode or display settings
- to get / press the [÷] *key*
- the symbols ♦ and ► are purely cosmetic, to indicate branching

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"** *(**R**oot **F**inder)*. 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"** *(**R**oot **F**inder **P**rogrammable)*, 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 accuray 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
2   XEQ "RFP"   1.7437520
```

(1st root, internally accurate to 9 places) — for 1.4688293

(2nd root, also internally accurate to 9 places; calling RFP saves the unneeded prompts) — for 1.7437520

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:

`XEQ "RF"` *NAME?* `"FX4"` `[R/S]` *X0?* ***90*** `[R/S]` ***40.0080372*** and *flag 00* is <u>*set*</u>.

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"` ***<u>0.6931472</u>*** and *flag 00* is <u>*clear*</u> 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 = LambertW(x),*** which is defined implicitly as: $ye^y = x$

In **PRGM** Mode, enter the following programs which define *LambertW* and the equation to solve, respectively :

```
01 ♦LBL "LAMBW"      08 FS?C 00      01 ♦LBL "YEY"
02 STO 04            09 AVIEW        02 ENTER↑
03 "YEY"             10 END          03 E↑X
04 ASTO 00                           04  *
05 LN                                05 RCL 04
06 XEQ "RFP"                         06  -
07 "NOT FOUND"                       07 END
```

*LAMBW simply stores **x** in R_{04}, the name of the equation in R_{00} ("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 *LambertW* for *x = 1, 2, 3, 10000*:

1	XEQ "LAMBW"	***0.5671433***	*to check:* [ENTER] [e^x] [x]	*1.0000000*
2	XEQ "LAMBW"	***0.8526055***	*to check:* [ENTER] [e^x] [x]	*2.0000000*
3	XEQ "LAMBW"	***1.0499089***	*to check:* [ENTER] [e^x] [x]	*3.0000000*
10000	XEQ "LAMBW"	***7.2318460***	*to check:* [ENTER] [e^x] [x]	*10,000.0000*

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 >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_{02} 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 ***1*** 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_{01}, R_{02} and R_{03} by registers *M*, *N*, and *O* respectively (i.e.: `STO 01` becomes `STO M` and so on), and insert step *42* `CLA` just before `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

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

Copyrights