STATISTICS 579
R Tutorial: More on Writing Functions

1. **Iterative Methods:**

   Three kinds of looping constructs in R: the `for` loop, the `while` loop, and the `repeat` loop were discussed previously. In order to understand the need for looping in a programming language, some background knowledge in numerical computing is useful.

   What follows is a short introduction to `iterative methods`. A variety of problems exist where the solution to a problem can be improved by applying the same calculation repeatedly, each time using the previous solution as the starting value. Such a mathematical process is said to form a convergent sequence, if the solution converges to a value such that the error becomes uniformly smaller in each iteration. The problem of finding the solution to the equation $f(x) = 0$ where f is a nonlinear function of a single variable x, is used in this class as an example of application of this method.

   Example: Find the root of $f(x) = x^3 - 3x + 1 = 0$ contained in the interval $[0, 1]$ by an iterative method.

   There are several approaches to obtaining iterative algorithms for this problem. A very brief look at a few methods follows below:

   Bisection Algorithm: The class of methods where only the value of the function at different points are calculated is known as *search methods*. The best known iterative search methods are called *bracketing methods* since they attempt to reduce the width of the interval where the root is located while keeping the successive iterates bracketed within that interval. The simplest bracketing method is the *bisection algorithm* where the interval of uncertainty is halved at each iteration, and from the two resulting intervals, the one that *straddles* the root is chosen as the next interval. Consider the function $f(x)$ to be continuous in the interval $[x_0, x_1]$ where $x_0 < x_1$ and that it is known that a root of $f(x) = 0$ exists in this interval.

   `Bisection Algorithm:`

   Select starting values $x_0 < x_1$ such that $f(x_0) * f(x_1) < 0$, ϵ_x, and ϵ_f,

   Repeat
   (a) set $x_2 = (x_0 + x_1)/2$
   (b) if $f(x_2) * f(x_0) < 0$

   set $x_1 = x_2$

   (c) else

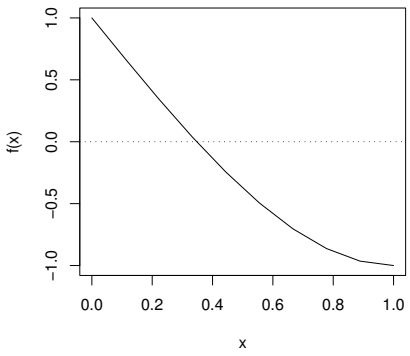
   set $x_0 = x_2$

   until $|x_1 - x_0| < \epsilon_x$ or $|f(x_2)| < \epsilon_f$

   Return x_2

   To use the bisection algorithm, it is easily verified by plotting the function that the $f(x) = x^3 - 3x + 1 = 0$ has a single root in the interval $[0, 1]$:

```
> f=function(x){x^3 - 3x + 1}
> x=seq(0,1,,10)
> plot(x,f(x),type="l")
> abline(h=0,lty=3)
```



The results of the bisection iteration are:

i	$x_{(2)}$	$f(x_{(0)})$	$f(x_{(1)})$	$f(x_{(2)})$	Error
1	0.5	1	-1	-0.375	0.1527036447
2	0.25	1	-0.375	0.265625	-0.0972963553
3	0.375	0.265625	-0.375	-0.07226563	0.0277036447
4	0.3125	0.265625	-0.07226563	0.09301758	-0.0347963553
5	0.34375	0.09301758	-0.07226563	0.009368896	-0.0035463553
6	0.359375	0.009368896	-0.07226563	-0.03171158	0.0120786447
7	0.3515625	0.009368896	-0.03171158	-0.01123571	0.0042661447
8	0.34765625	0.009368896	-0.01123571	-0.0009493232	0.0003598947
9	0.345703125	0.009368896	-0.0009493232	0.00420583	-0.0015932303
10	0.3466796875	0.00420583	-0.0009493232	0.001627262	-0.0006166678
11	0.3471679688	0.001627262	-0.0009493232	0.000338721	-0.00012838655
12	0.3474121094	0.000338721	-0.0009493232	-0.0003053632	0.000115754075
13	0.3472900391	0.000338721	-0.0003053632	1.666335e-05	-6.3162375e-06
14	0.3473510742	1.666335e-05	-0.0003053632	-0.0001443538	5.471891875e-05
15	0.3473205566	1.666335e-05	-0.0001443538	-6.38462e-05	2.420134062e-05
16	0.3473052979	1.666335e-05	-6.38462e-05	-2.359167e-05	8.942551562e-06

It is obvious that the convergence is very slow, the error reducing extremely slowly and even changing sign. However, the bisection eventually converges and is not affected by features of the function such as its gradient.

Fixed Point Iteration: By re-expressing $f(x) = 0$ in the form $x = g(x)$, one can obtain an iterative formula $x_{(i+1)} = g(x_{(i)})$ for finding roots of nonlinear equations. Any such function g, called the mapping function, must satisfy conditions so that the sequence $x_{(1)}, x_{(2)}, \ldots$ converges to a unique solution in a specified interval $[a, b]$.

To use the fixed point method for $f(x) = x^3 - 3x + 1 = 0$, rewrite it in the form $x = (x^3 + 1)/3$ giving $g(x) = (x^3 + 1)/3$. It can be verified that this choice of g satisfies the necessary conditions for the iteration to produce a convergent sequence. To obtain a starting value, we use the plot of $f(x)$ in the interval $[0, 1]$. The plot clearly shows that a root exists is the interval between .3 and .4. In practice, several intervals may be searched for the existence of roots by trial and error, and is called a *grid search*. The results of the iteration using $x_{(0)} = .4$ as the starting value are:

i	$x_{(i)}$	Error
1	0.3546667	0.007370267
2	0.3482043	0.0009079228
3	0.3474062	0.0001097562
4	0.3473096	1.320312e-05
5	0.3472980	1.553274e-06
6	0.3472965	1.480699e-07
7	0.3472964	2.141932e-08

Newton-Raphson Iteration: This iteration is based on the following. Consider a starting value $x_{(0)}$ for obtaining an itertive formula to solve $f(x) = 0$. If the next iterate $x_{(1)}$ is very close to $x_{(0)}$ then the first derivative of f at $x_{(0)}$ can be approximated as $f'(x_{(0)}) \approx \frac{f(x_{(0)})}{x_{(0)}-x_{(1)}}$. This gives an approximation for the next iterate $x_{(1)}$ as $x_{(1)} \approx x_{(0)} - \frac{f(x_{(0)})}{f'(x_{(0)})}$, which leads to the Newton-Raphson iterative formula

$$x_{(i+1)} = x_{(i)} - \frac{f(x_{(i)})}{f'(x_{(i)})}$$

Newton-Raphson iteration produces a convergent sequence as long as the starting value $x_{(0)}$ is chosen to be close to a root of $f(x) = 0$. A good property of this method is that it has *quadratic convergence*, i.e., the error is reduced quadratically at each iteration, the fastest rate of convergence for an iteration for root finding.

Since the the first derivative of $f(x) = x^3 - 3x + 1$, the example used previously, is $f'(x) = 3x^2 - 3$ the Newton-Raphson iterative formula to solve $x^3 - 3x + 1 = 0$ is given by

$$x_{(i+1)} = x_{(i)} - \frac{x_{(i)}^3 - 3x_{(i)} + 1}{3x_{(i)}^2 - 3}$$

Applying this iteration with the starting value of $x_{(0)} = .4$ to find the root gives the following sequence:

i	$x_{(i)}$	Error
1	0.3460317	0.001264654
2	0.3472957	6.740933e-07
3	0.3472964	4.46663e-08
4	0.3472964	4.466614e-08

When iteration schemes such as above are implemented in computer programs, a condition for termination of the iteration, called the *stopping rule* must be specified. In the above two examples the iteration was terminated when the absolute difference between two successive iterates became smaller than 1.e-8. It has been suggested that more than a single rule be used for determining whether convergence is achieved. For example, in a root-finding problem one could also check if the function value has become close enough to zero to conclude that convergence has been reached and thus terminate the iteration.

2. **Programming Iterative Methods in R:**

If the fixed point iteration is to be used to solve an equation $f(x) = 0$, it must be first expressed in the form $x = g(x)$, where $g(x)$ must satisfy certain conditions so that the iteration $x_{(i+1)} = g(x_{(i)})$ produces a convergent sequence. It will be assumed that this part has been done and $g(x)$ is available. The following function, to be copied from the file `simple.R` in the `StatLabs\Stat579\R` folder on your desktop, is an implementation of the fixed point iteration (also called simple iteration). Source this function as usual into R:

```
simple=function(x0, nlim, eps)
{
        iterno = 0
        repeat {
            iterno = iterno + 1
            if(iterno > nlim) {
                cat(" Iteration Limit Exceeded: Current = ",iterno, fill = T)
                x1=NA
                break
              }
            else
            {
            x1=g(x0)
            cat("****Iter. No: ", iterno, "    Current Iterate = ", x1, fill = T)
            if(abs(x1-x0) < eps||abs(x1-g(x1))<.1e-10)
                        break
            x0=x1
            }
        }
        return(x1)
}
```

Note that the arguments are, respectively, `g`, the mapping function, `x0`, the starting value, `nlim` , a limiting value for the number of iterations for the iteration to be terminated if convergence is not reached before that number, and `eps`, the tolerance value to determine if convergence has been reached. To find the root of $f(x) = x^3 - 3x + 1 = 0$ contained in the interval $[0, 1]$ using the mapping function $g(x) = (x^3 + 1)/3$, first define g and call the above function in R:

```
(Assume the function simple() has been sourced into R)
> g = function(x){(x^3+1)/3}
> simple(.4,100,.000001)
****Iter. No:  1     Current Iterate =  0.3546667
****Iter. No:  2     Current Iterate =  0.3482043
****Iter. No:  3     Current Iterate =  0.3474062
****Iter. No:  4     Current Iterate =  0.3473096
****Iter. No:  5     Current Iterate =  0.3472980
****Iter. No:  6     Current Iterate =  0.3472965
****Iter. No:  7     Current Iterate =  0.3472964
[1] 0.3472964
```

Note carefully that the R function `simple()` calls another R function `g()`. Both these functions are user-written. However, if the function `g()` were to be renamed as, say `g1()`, `simple()` will not execute because it will look for the function named `g()`. To correct this problem, the function `simple()` is modified by including `g()`, as an argument:

```
(The modified version of simple() is  sourced into R)
```

```
> simple
function(g, x0, nlim, eps)
{
        iterno = 0
        repeat {
            iterno = iterno + 1
            if(iterno > nlim) {
                cat(" Iteration Limit Exceeded: Current = ",iterno, fill = T)
                x1=NA
                break
              }
            else
            {
            x1=g(x0)
            cat("****Iter. No: ", iterno, "    Current Iterate = ", x1, fill = T)
            if(abs(x1-x0) < eps||abs(x1-g(x1))<.1e-10)
                        break
            x0=x1
            }
        }
        return(x1)
}
> simple(g,.4,100,.000001)
****Iter. No:  1     Current Iterate =  0.3546667
****Iter. No:  2     Current Iterate =  0.3482043
****Iter. No:  3     Current Iterate =  0.3474062
****Iter. No:  4     Current Iterate =  0.3473096
****Iter. No:  5     Current Iterate =  0.3472980
****Iter. No:  6     Current Iterate =  0.3472965
****Iter. No:  7     Current Iterate =  0.3472964
[1] 0.3472964
```

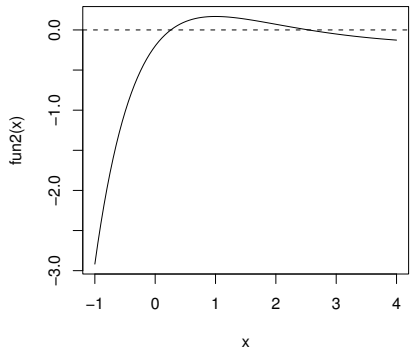
In the following implementation of the Newton-Raphson algorithm to solve $f(x) = 0$, two R functions `fun()` and `derf()` that represent the function f and its derivative f' are passed into the function `newton()` as arguments. The function `newton()` is in the file `newton.R` available in the `StatLabs\Stat579\R` folder on your desktop. Copy this file and source it into R. Before calling `newton()`, two R functions `fun()` and `derf()` need to be written:

```
newton=function(fun, derf, x0, eps)
{
        iter=0
        repeat {
                iter= iter+1
                x1 = x0 - fun(x0)/derf(x0)
                if(abs(x0 - x1) < eps||abs(fun(x1))<.1e-10)
                        break
                x0 = x1
                cat("****** Iter. No: ", iter, "    Current Iterate = ", x1,fill=T)
        }
        return(x1)
}
```

```
> fun = function(x){x^3 - 3*x + 1}
> derf = function(x) { 3*x^2 -3 }
> newton(fun,derf,.4,.000001)
****** Iter. No:  1     Current Iterate =  0.3460317
****** Iter. No:  2     Current Iterate =  0.3472957
[1] 0.3472964
```

Below, the entire sequence of steps needed to solve an equation using `newton()` is repeated, to find the roots of the equation $xe^{-x} = 0.2$. First sketch the function:

```
> fun2 =function(x) {x*exp(-x)-0.2}
> x= seq(-2,2,,20)
> fun2(x)
 [1] -14.97811220 -10.91232802  -7.85765477  -5.57672048  -3.88583912  -2.64318178
 [7]  -1.73950426  -1.09088663  -0.63305522  -0.31694768  -0.10525393   0.03027777
[13]   0.11093553   0.15266956   0.16735168   0.16374898   0.14827479   0.12556636
[19]   0.09892812   0.07067057
> x=seq(-1,4,,100)
> plot(x,fun2(x),type="l")
> abline(h=0,lty=2)
```



It is obvious that there is one root near 0.0 and another near 2.0. Write a function `derf2()` representing f' and call `newton()` with appropriate starting values:

```
> derf2=function(x) {(1-x)*exp(-x)}
> newton(fun2,derf2,0.0,.000001)
****** Iter. No:  1     Current Iterate =  0.2
****** Iter. No:  2     Current Iterate =  0.2553507
****** Iter. No:  3     Current Iterate =  0.2591540
****** Iter. No:  4     Current Iterate =  0.2591711
[1] 0.2591711
> newton(fun2,derf2,2.0,.000001)
****** Iter. No:  1     Current Iterate =  2.522189
****** Iter. No:  2     Current Iterate =  2.542569
****** Iter. No:  3     Current Iterate =  2.542641
[1] 2.542641
```

Finally, for completeness, an R function named `bisect()` for implement the bisection method is available in the file `bisect.R` in the `StatLabs\Stat579\R` folder on your desktop. Recall that the bisection method is a bracketing method; thus two values (x_0, x_1) that bracket the requiredroot uniquely must be available as starting values:

```
bisect=function(fun, x0, x1 , eps)
{
        iterno = 0
        repeat {
                iterno = iterno + 1
                x2 = (x0+x1)/2
                if (fun(x2)*fun(x0)<0)
                       x1=x2
                else
                       x0=x2
                if(abs(x1-x0) < eps||abs(fun(x2))<.1e-10)
                        break
                cat("****** Iter. No: ", iterno, "    Current Iterate = ",
                    x2, fill = T)
        }
        return(x2)
}
> f=function(x){x^3 - 3x + 1}
> bisect(f,0,1,.00001)
****** Iter. No:  1     Current Iterate =  0.5
****** Iter. No:  2     Current Iterate =  0.25
****** Iter. No:  3     Current Iterate =  0.375
****** Iter. No:  4     Current Iterate =  0.3125
****** Iter. No:  5     Current Iterate =  0.34375
****** Iter. No:  6     Current Iterate =  0.359375
****** Iter. No:  7     Current Iterate =  0.3515625
****** Iter. No:  8     Current Iterate =  0.3476563
****** Iter. No:  9     Current Iterate =  0.3457031
****** Iter. No:  10     Current Iterate =  0.3466797
****** Iter. No:  11     Current Iterate =  0.347168
****** Iter. No:  12     Current Iterate =  0.3474121
****** Iter. No:  13     Current Iterate =  0.3472900
****** Iter. No:  14     Current Iterate =  0.3473511
****** Iter. No:  15     Current Iterate =  0.3473206
****** Iter. No:  16     Current Iterate =  0.3473053
[1] 0.3472977
```