**ECON 442: Quantitative Trade Models** Professor Jack Rossbach

**Solving Models in R**

In this class we have been using R to solve nonlinear systems of equations using the *multiroot* function from the *rootSolve* library. We are going to retrace our steps to cover the basics of of this process.

**Getting Set Up**

The first step is to open both R and a text editor. We've been writing our code in Notepad++ and pasting it into the R console. For our purposes, Notepad++ is just a plain text editor with syntax highlighting. If you are uncomfortable with the Notepad++/R console setup, an alternative is RStudio. Rstudio provides both a plain text editor with syntax highlighting and the R console in a single interface, which may be easier for some students. Another alternative is R-fiddle (http://www.r-fiddle.org), which also combines a text editor and R console in a single interface, and allows you write, run, and share R code online, again. The downside of R-fiddle is that you cannot run only selections of your code, which may make finding typos and debugging your code difficult.

**Math Underlying Nonlinear Root Solving**

We can gain intuition for what R is doing by reviewing Newton's method for root solving. Newton's method is a way to approximate the root of real valued function $f(x)$. Roots of $f(x)$ are values of $x$ for which $f(x) = 0$. If the function does not have any roots (in which case Newtons method won't converge).

Newton's method works by picking an initial guess $x_0$ and then iterating according to

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

Where $f'(x)$ is the derivative of $f(x)$ with respect to $x$. [i.e. $f'(x) \equiv \partial f(x)/\partial x$]. We then continue the iterative process, using the current guess $x_n$ to find the next guess $x_{n+1}$

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

When the function $f(x)$ is "nice" Newtons method will quickly converge to a root. If the function is not so nice, or if it has multiple roots, the initial guess may be important in determining whether Newton's method converges and which root it converges to.

Note that root finding is equivalent to nonlinear equation solving. Suppose we have two functions of $x$; $h(x)$ and $g(x)$. If we want to find the $x$ such that $h(x) = g(x)$ we can use Newton's method to find this solution by defining $f(x) = h(x) - g(x)$ and then finding the root for $f(x)$. This is why, when we put our equilibrium equations into R we put all the variables on one side and equate it to zero.

That is instead of writing

$$\text{Expenditures} = \text{Income}$$

we write the equivalent statement in R, where we leave off the $= 0$ since it is implied

$$\text{Income} - \text{Expenditures} = 0$$

**Example of Newtons Method for a Single Variable**

Consider the function $f(x) = x^2 - 4$. Then $f'(x) = 2x$. We could easily solve for $f(x) = 0$ algebraically, but let's do with with Newton's method and an initial guess of **1**

| Iteration | Current Guess $(x_n)$ | $f(x_n)$ | $f'(x_n)$ | $x_n - \frac{f(x_n)}{f'(x_n)}$ |
|---|---|---|---|---|
| 0 | **1** | -3 | 2 | 2.5 |
| 1 | 2.5 | 2.25 | 5 | 2.05 |
| 2 | 2.05 | 0.2025 | 4.1 | 2.000609756 |
| 3 | 2.000609756 | 0.002439396 | 4.001219512 | 2.000000093 |
| 4 | 2 | 3.72E-07 | 4 | 2 |

**Nonlinear Systems of Equations with Multiple Variables**

Finding the roots of systems of equations is similar to a single variable, except we use vector and matrix notation. See here for a simple write up: https://www.math.ohiou.edu/courses/math3600/lecture13.pdf

The algorithm we use in R to solve nonlinear systems of equations is similar to Newton's method, except that you do not need to provide the derivatives of the functions since it approximates them numerically.

**Using R code to find solution to equation $2x^2 = 4$.**

**Step 1: Define the $f(x)$ function in R**

Solving $2x^2 = 4$ is equivalent to finding the root of $f(x) = 2x^2 - 4$.

The following R code defines this function

```
example_fun <- function(x) {
      2*x^2-4
}
```

We named the function "example_fun" but we could have named it anything. We also did not need to use the variable $x$, we could have instead written the equivalent code

```
example_fun <- function(potato) {
      2*potato^2-4
}
```

Note that regardless of what we call the variable in the function, it only exists inside of the loop. This is because we are working with R as a local variable. If we give $x$ a value outside of the function, that is kept completely separate from the $x$ inside the function. We can return valuesThat will return $f(2) = 2(2)^2 - 4 = 4$. Note that when we defined the function,

```
example_fun(2)
```

That will return $f(2) = 2(2)^2 - 4 = 4$. Note that when we defined the function,

**Step 2: Use the multiroot function**

We need to load the rootSolve library and then use the multiroot function. Similarly to Newton's method, we need to provide an initial guess for the algorithm to use.

```
library("rootSolve")
multiroot(f = example_fun, start = 2)
```

The first line of code loads the library, which contains the prewritten multiroot function. The second line of code finds the root of our example function using a starting value of 2. It will return the root of $f(x) = 2x^2 - 4$, the value of $f(x)$ at that value, how many iterations the algorithm took to converge, and the estimated precision. When we work with a system of equations, the precision is the distance from zero of the equation the furthest away from zero. We only have one equation, so it is equal to the value of that function. Note that the precision is not exactly zero, so we did not find an exact solution but rather an approximation that is very good.

We can store the results of the multiroot algorithm in a separate variable. That way we can access parts of the solution later if we want. Suppose we only care about the root of the function. We can access that with the following code:

```
answer <- multiroot(f = example_fun, start = 2)
answer$root
```

The first line stores the result of the multiroot algorithm in the variable *answer.* The second line of code displays just the root. If you forget what a function does, you can always access documentation by typing help(function) where function is what you want documentation on. For functions in libraries, you must load the library first, otherwise it well not recognize the function. For example,

```
help(multiroot)
```

will bring you to a local version of http://finzi.psych.upenn.edu/library/rootSolve/html/multiroot.html if you have the library rootSolve installed on your computer and loaded in R.

**Example: Use R to solve a simple partial equilibrium model**

Consider the question from online assignment 4

In the home country supply and demand are given by

$$S^H = 20 + 20P^H$$

$$D^H = 100 - 20P^H$$

How can we find the free trade equilibrium for the home market in Autarky market? The answer is to set Supply equal to Demand in the Home country.

**Step 1: Define the $f(x)$ function in R**

The first step is to figure out what the function is that we want to set equal to zero. In our case, that function is $f(x) = \text{Supply} - \text{Demand}$.

Instead of solving for the right hand side ourselves, we can simply

The following R code defines this function excess world supply

```
excess_Home_supply <- function(P_H) {
      S_H = 20 + 20*P_H
      D_H = 100 - 20*P_H

      result <- S_H - D_H

      return(result)
}
```

We named the function "excess_Home_supply" this time, but again we could have named it anything (no spaces, special characters, or starting with a number). We also used the variable $P\_H$, which we will use to represent $P^H$. What you call the variable isn't important, what's important is that you remember what the variable represents so you write the equations correctly.

The way functions work in R, they automatically return the last line where there is no assignment (no "=" or "<-"), since R guesses that this is what you want it to return. If you do not want R to guess what we want the function to return, we can specify it by using the return() function. R will then return whatever we put in the function. Above, we put the variable "result", which is excess supply, or $S^H - D^H$.

**Step 2: Use the multiroot function**

If you previously loaded the rootSolve library, you don't need to do it again, however it doesn't hurt to do so. We then use the multiroot function, starting it off with an initial guess.

```
library("rootSolve")
multiroot(f = excess_Home_supply, start = (some initial guess) )
```

The above should report the value of $P^H$ that sets $S^H = D^H$ after putting in your initial guess.

**Exercise 1: Use R to solve a partial equilibrium model**

Now consider the version of the question with trade.

In the home country supply and demand are still given by

$$S^H = 20 + 20P^H$$

$$D^H = 100 - 20P^H$$

In the foreign country, supply and demand are given by

$$S^F = 40 + 20P^F$$

$$D^F = 80 - 20P^F$$

How can we find the free trade equilibrium for this market? The answer is to set Export Supply equal to Import Demand for the world as a whole. We know that the foreign market will export since it has higher supply $(40 > 20)$ and lower demand $(80 < 100)$ than the foreign country (slopes are same).

**Step 1: Define the $f(x)$ function in R**

The first step is to figure out what the function is that we want to set equal to zero. In our case, that function is $f(x) = \text{Export Supply} - \text{Import Demand}$.

***Fill out the R code to define the function for $f(x)$***

```
excess_world_supply <- function(P_H) {

      ##define home supply and demand
      ##define foreign supply and demand
      ##then define XS and MD

      result = XS - MD

      return(result)
}
```

It's important to remember not to use variables (e.g. $P^F$) that you haven't defined within the loop in terms of function inputs ($P^H$). This may be relevant when you defined $S^F$ and $S^H$.

**Step 2: Use the multiroot function**

After defining the multiroot function you can find the world equation.

```
multiroot(f = excess_world_supply, start = (some initial guess) )
```

**Exercise 2: Add a Tariff**

Go back to your code above, and make it so that there is a 50 cent *specific* tariff. This means

$$P^H = 0.50 + P^F$$

Find the new Home price. Rearrange the above so $P^F$ is on the LHS, so you can put it in your code.

**Exercise 3: Find the Tariff such that total Tariff Revenue is equal to 3**

Tariff revenue for the Home country is equal to Imports times the value of the tariff (since this is a specific, or flat tariff, and not an ad valorem tariff). I.e. $\text{Tariff Revenue} = \text{MD} * \text{Tariff}$

To solve this problem, we now need the function to have multiple inputs, as we're solving for both the Tariff that gives us Tariff Revenue equal to 3, and the Home Price, $P^H$, that sets $XS = MD$.

We can define a vector variable $\text{Inputs} = [P^H, \text{Tariff}]$ and use this in our function.

```
equilibrium <- function(Inputs) {

      P_H <- Inputs[1]
      Tariff <- Inputs[2]

      ##The rest of your code from before, need P_F = P_H - Tariff

      ##Need to Define Tariff Revenue
      Rev = MD * Tariff

      ##The result of the function will now be a vector.
      #The first element will say how far we are from XS=MD and the
      #second element will say how far we are from Tariff Revenue of 3.
      result <- c(XS-MD, Rev-3)

      return(result)
}
```

This tells the code that $P^H$ is the first element of Inputs and Tariff is the second element of Inputs. When R executes our code, it automatically guesses that since we are trying to access the first two elements of the variable Inputs that it is a vector with 2 total elements. This is a nice thing about R, in other programming languages you often need to declare what everything is before you're able to work with it. R guesses what we want without us having to tell it explicitly.

When you use the multiroot function now, you'll need to put in an initial guess for the vector Inputs. You create a vector in $R$ using the "c()" function. For example to create the vector $[1, 0.05]$ and store it in the variable "vec", you would just type:

```
vec <- c(1,0.05)
```

We can then find the equilibrium for the specific tariff that gives the Home country total Tariff revenue equal to 3 by again using the multiroot function

```
multiroot(f = equilibrium, start = c((Price Guess),(Tariff Guess)) )
```

**Finished Code for All Exercises**

Example Code: http://www.r-fiddle.org/#/fiddle?id=fFBpZJAE

Exercise 1 Code: http://www.r-fiddle.org/#/fiddle?id=dk8WCZlX

Exercise 2 Code: http://www.r-fiddle.org/#/fiddle?id=MHuXbdoP

Exercise 3 Code: http://www.r-fiddle.org/#/fiddle?id=lNjMUv0a

*Copy and Paste Link into Browser if Link Doesn't Work*

I strongly suggest not looking at the finished code until you think you've figured it out yourself or have completely stopped making progress. It's important to remember that there are multiple ways of writing the code to solve these problems. Most ways will be fairly similar since this is a straightforward problem, but other ways aren't wrong if they work.