Math 284: Introduction to Root-Finding

If you are new to working with MATLAB-like software, I encourage you to work through tutorial at `http://www.cyclismo.org/tutorial/matlab/`. We will also build skills as we go along, learning by imitating code in class and from the text book. You should also work through the MATLAB examples that follow each section in the textbook as another great way to get some guided help. Always let me know if you have any questions as you go along! I'm happy to help both during class and in office hours (and via email for simpler questions), to make your computing experience enhance the course and enable exploration, which is what makes the subject fun.

Today let's try out some of the root-finding methods, as a way of learning some coding as well as exploring those methods. If you haven't done much coding before, no worries–work through the examples to see what kinds of things can be done and start to get a feel for how the program works, but I certainly don't expect everyone to start whipping up for-loops with fancy index finding techniques.

Bisection and brackets

The simplest method conceptually is to bracket a root, that is, find an interval containing the root. We compute the function at each end of the bracket and at the middle to see where a sign change occurs, then update the bracket to the left or right half, according to where the sign change happened. Repeat until desired accuracy is reached, for instance, the first 12 decimals places are no longer changing.

To create a bracket for seeking the square root of, say, 5:

```
format long
bracket=[2 2.5 3];
bracket.^2-5
```

You should see the change in sign on the left half, so we proceed by updating the bracket to [2 2.25 2.5] and repeat a few times. This process gets tedious repeating by hand in the command line, so let's be more efficient in automating it.

Open a new script in the editor and type `clear` or `clear all` at the top. Unless you're creating a function script, clearing the memory is generally a good idea to

avoid bugs due to previously stored values interfering with new computations. Save the script in some suitable folder, ensuring the name ends in `.m`. Now type in the following loop, which runs the process 10 times (or however many you specify), by identifying where the sign changes:

```
bracket=[2 2.5 3];
for count=1:10
ind=find(abs(diff(sign(bracket.^2-5)))>0);
bracket=[bracket(ind) (bracket(ind)+bracket(ind+1))/2 bracket(ind+1)];
end
```

If you are working in FreeMat, the `sign` function is not built in. To fix this, create a new script called `sign.m` saved in the same folder with your root-finding script, and type in the following two lines:

```
function s=sign(x)
s=(2*(x>0)-1).*(x~=0);
```

We can make this code a bit more elegant by defining a function to evaluate the bracket values:

```
f=inline('x.^2-5');
bracket=[2 2.5 3];
for count=1:10
ind=find(abs(diff(sign(f(bracket))))>0);
bracket=[bracket(ind) (bracket(ind)+bracket(ind+1))/2 bracket(ind+1)];
end
```

One further nice update would be to have the loop itself determine how long to run, rather having to specify a particular number of repetitions. Let's use the condition that we repeat until the x-value in the middle of the bracket yields $|f(x)| < 10^{-8}$. We fix a max number of repetitions for safety's sake, to avoid infinite loops. The counter will also let you see how many repetitions it took to reach the desired condition.

```
f=inline('x.^2-5');
bracket=[2 2.5 3];
count=0;
while abs(f(bracket(2)))>1e-8 && count<100
ind=find(abs(diff(sign(f(bracket))))>0);
bracket=[bracket(ind) (bracket(ind)+bracket(ind+1))/2 bracket(ind+1)];
count=count+1;
end
```

You can simply type `bracket` and `count` in the command window to check the results. If you want to get fancy, you can have the script output the results in a formatted manner:

```
disp(['Estimated value:  '  num2str(bracket(2),'%0.12f')])
disp([' Actual value:  '  num2str(sqrt(c),'%0.12f')])
disp([' Iterations:  '  num2str(count)])
```

An iterative method

Now let's try implementing Heron's rule for finding the square root of c, which involves rewriting the equation $x^2 - c = 0$ to the form $x = \frac{1}{2}(x + \frac{c}{x})$ (check that these are equivalent). We'll use $c = 5$ again as an example, but you should try out different numbers to explore the method's performance.

```
g=inline('(x+5/x)/2');
x=1;
x=g(x)
```

To iterate multiple times without having to enter `x=g(x)` over and over, you can do a for-loop:

```
x=1;
for count=1:5
x=g(x);
end
```

Or a while-loop to automate the iterations until a desired condition is met:

```
x=1;
count=0;
while abs(f(x))>1e-8 && count<100
x=g(x);
count=count+1;
end
```

How do the methods compare? Which seems to require fewer repetitions to reach the desired accuracy? Which seems like a more elegant method to implement?