MAT 155 Project VIII Spring 2014
Newton's Method

BYUNG DO PARK

You may work on this file. Don't forget to **save it regularly** as yourname8.mws and as yourname8b.mws. Sign your name as a comment at the top of the file by backspacing in front of the prompt and typing. Also write the names of any other students you are working with. Don't forget to number your problems and to type restart at the beginning of each problem.
If you need more space to work than you are given, you can **get a new prompt** by selecting the prompt button right below the word "spreadsheet". It has the symbol "[>" on it.

Suppose we want to solve f(x)=0. We already know that we can use solve or fsolve. However, we do not really unerstand how Maple finds these solutions. When we type solve, Maple uses formulas like the quadratic formula and algebraic techniques just like you have all learned in algebra. When we type fsolve Maple uses numerical techniques similar to the ones we will learn today to find the decimal expansion of a solution. These techniques will also give you an introduction to computer programming.

Problem I: The Method of Tabulation and Loops:

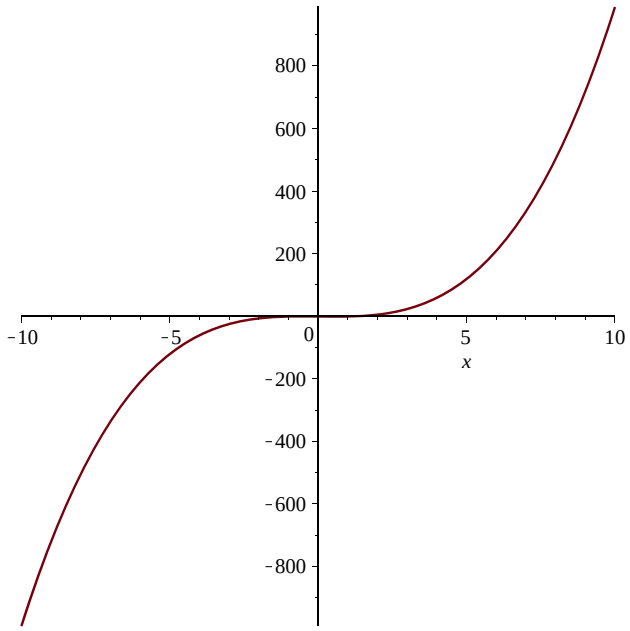
Suppose we want to find the real roots of a continuous function. We know that if a continuous function is positive at one point and negative at another point then it must be zero somewhere in between the two points. This is called the intermediate value theorem.

Let f(x)= x^3-x-1; We want to find a real solutions of f(x) = 0 using the intermediate value theorem. Hit enter on the following lines:

```
> f := x -> x^3 - x - 1;
```

$$f := x \to x^3 - x - 1 \qquad \textbf{(1)}$$

```
> plot(f(x), x=-10 .. 10);
```



The plot shows there is only one real root, in (-2, 3). It is not clear where the graph is positive or negative between -2 and 3, so we can just check:

```
> f(-2), f(-1),f(0), f(1), f(2), f(3);
```

$$-7, -1, -1, -1, 5, 23 \qquad \textbf{(2)}$$

And see that f(1)<0 and f(2)>0 so there is a solution between 1 and 2.

If we check every number 1.0, 1.1, 1.2,.. 1.9, 2.0, then we can find out what the solution is with an accuracy of one decimal place. This would take a while to work out by hand, but a computer can do it quickly. In fact we can write a "**Loop" Program** and tell the computer that **for x** = numbers starting **from 1 by** taking steps of size **0.1** until you get **to 2 do** the command **x, f(x)**. Recall that the words for, from, by, to, do and od are all commands that you can use
to write a loop program. do and od are placed on either side of the list of commands. ***Be sure to save your file before clicking enter on this loop in case it crashes the computer!***

```
> for x from 1  by  0.1  to  2  do  x, f(x)  od;
```

$$1, -1$$
$$1.1, -0.769$$
$$1.2, -0.472$$
$$1.3, -0.103$$

1.4, 0.344
1.5, 0.875
1.6, 1.496
1.7, 2.213
1.8, 3.032
1.9, 3.959
2.0, 5.000 **(3)**

The output shows that there is a solution of f(x) = 0 in (1.3, 1.4). To pinpoint the solution, test f(x) in (1.3, 1.4), taking step = 0.01:

```
> for x from 1.3 by 0.01 to 1.4 do x, f(x) od;
```

1.3, −0.103
1.31, −0.061909
1.32, −0.020032
1.33, 0.022637
1.34, 0.066104
1.35, 0.110375
1.36, 0.155456
1.37, 0.201353
1.38, 0.248072
1.39, 0.295619
1.40, 0.344000 **(4)**

Since 1.32 < c < 1.33, the solution c = 1.3 is accurate to 1 decimal place.

Now you repeat this process until you get a solution which is accurate to 3 decimal places. Write comments explaining what you are doing.

```
> for x from 1.32 by 0.001 to 1.33 do x, f(x) od;
```

1.32, −0.020032
1.321, −0.015800839
1.322, −0.011561752
1.323, −0.007314733
1.324, −0.003059776
1.325, 0.001203125
1.326, 0.005473976
1.327, 0.009752783
1.328, 0.014039552
1.329, 0.018334289
1.330, 0.022637000 **(5)**

```
> for x from 1.324 by 0.0001 to 1.325 do x, f(x) od;
```

1.324, −0.003059776
1.3241, −0.002633843
1.3242, −0.002207832
1.3243, −0.001781740
1.3244, −0.001355569
1.3245, −0.000929319
1.3246, −0.000502989
1.3247, −0.000076580
1.3248, 0.000349909
1.3249, 0.000776477
1.3250, 0.001203125 **(6)**

Problem 2: The while loop:

Now suppose you have a slow computer and you don't want it to bother doing all the computations. After all, you know that for the f in problem 1, f(1)<0 and f(2) >0. You need only check f(1), f(1.1) and

so on until you get a negative answer. So you use a "while loop" and tell the computer that **for x** = numbers starting **from 1 by** taking steps of size **0.1** until you get **to 2 while f(x)<0 do** the command **x, f (x).** This means that the computer should only keep working while f(x)<0 and stop when f(x) is bigger than or equal to 0. ***Be sure to save your file before clicking enter on the while loop in case it crashes the computer!***

```
> for x from 1  by  0.1  to  2 while (f(x) < 0) do  x, f(x)  od;
```

1, −1
1.1, −0.769
1.2, −0.472
1.3, −0.103 **(7)**

This output tells us that the computer got f(1.3)<0 but f(1.4)>0. It didn't bother to do the command 1.4, f(1.4) because the answer f(1.4) wasn't negative anymore. Now you can continue with another while loop to find the decimal expansion to two decimal places.

```
> for x from 1.3  by  0.01  to  1.4 while (f(x) < 0) do  x, f(x)
  od;
```

1.3, −0.103
1.31, −0.061909
1.32, −0.020032 **(8)**

Problem 3: Newton's Method: **Look up wikipedia article(there's a nice picture)**

Using the Method of Tabulation described above you have to run the for or while loops over and over, one time for each extra digit of accuracy. This is not very efficient because you are just dividing the intervals up into ten pieces every time and doing lots of computations. If the function you are working with is a differentiable function then you can use another technique called Newton's Method. It won't always work, but it often works faster than the method of tabulation. The idea is to use tangent lines to point towards solutions.

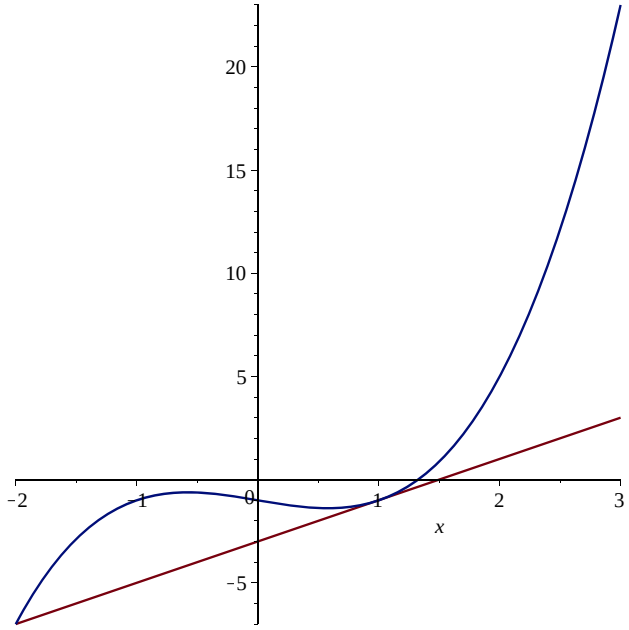
We are going to use the same function again so we won't type restart, but we do need to clear the value for x.

```
> x:='x';
```

$$x := x \quad \textbf{(9)}$$

Recall that for a function f, the slope of the tangent line at a point ***a*** is ***m:= D(f)(a)*** and the line is ***y= (D(f)(a)) (x-a) +f(a).***
So for example, if we take the function f from problem 1, we can find the tangent line at 1 and graph it with the function:

```
> plot({ f(x) , D(f)(1)*( x-1 ) + f(1) }, x=-2..3);
```



Recall from last week that the tangent line at ***a*** is the linear approximation for f at ***a***. This means that near ***a*** the tangent line and ***f*** are rather close to each other. In particular, in the graph you've just plotted, you can see that the tangent line crosses the x axis near where the function crosses the x axis. So we can find out where it crosses doing some simple algebra:
(D(f)(1))*(x-1)+f(1)=0 (D(f)(1))(x-1) = - f(1) x-1= -f(1)/D((f)(1))
so ***x = 1- f(1)/(D(f)(1))*** . We can call this x where the line crosses the x axis, "crossa" *(In computers it is common to use an entire word for a variable instead of just one letter)*:

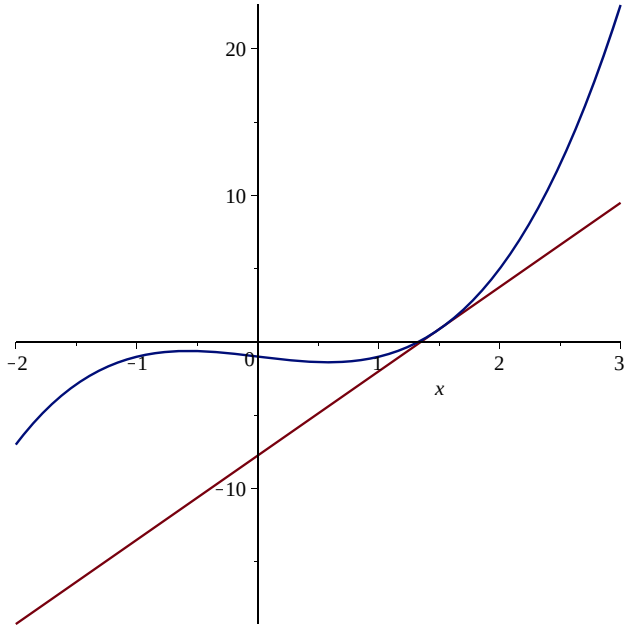
```
> crossa:= 1-f(1)/(D(f)(1));
```

$$crossa := \frac{3}{2} \qquad \textbf{(10)}$$

and find the tangent line to f at a point a = crossa:

```
> a:= evalf(crossa); plot({f(x), (D(f)(a))*(x-a)+f(a)}, x= -2..3);
```

$$a := 1.500000000$$



Now this tangent line crosses very close to the same place f does. It is easy to repeat the process and choose a new crossa where this tangent line crosses: We will call it "crossa" again. The old crossa is just called "a" now.

```
> crossa:=a-f(a)/(D(f)(a)); evalf(crossa);
```

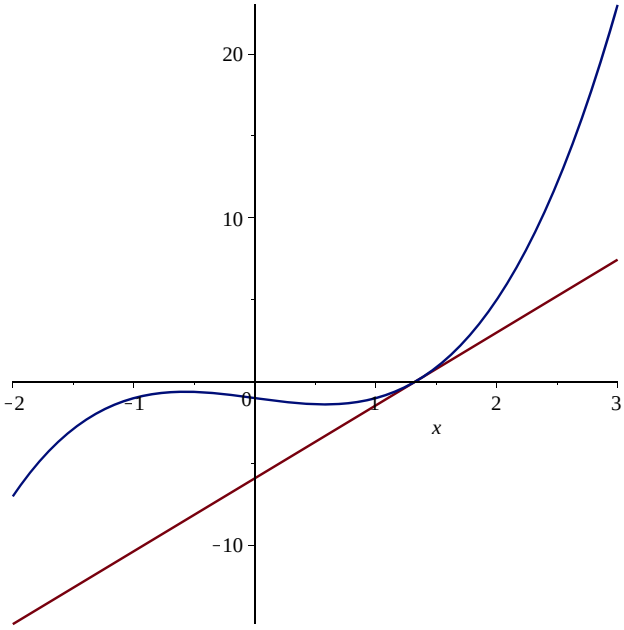
crossa := 1.347826087

1.347826087 **(11)**

Now lets find the tangent line to f at a= this new crossa.

```
> a:= evalf(crossa); plot({f(x), (D(f)(a))*(x-a)+f(a)}, x= -2..3);
```

a := 1.347826087



Now we check where it crosses, and can use the same formula as before to get a third crossa:

```
> crossa:=a-f(a)/(D(f)(a)); evalf(crossa);
```

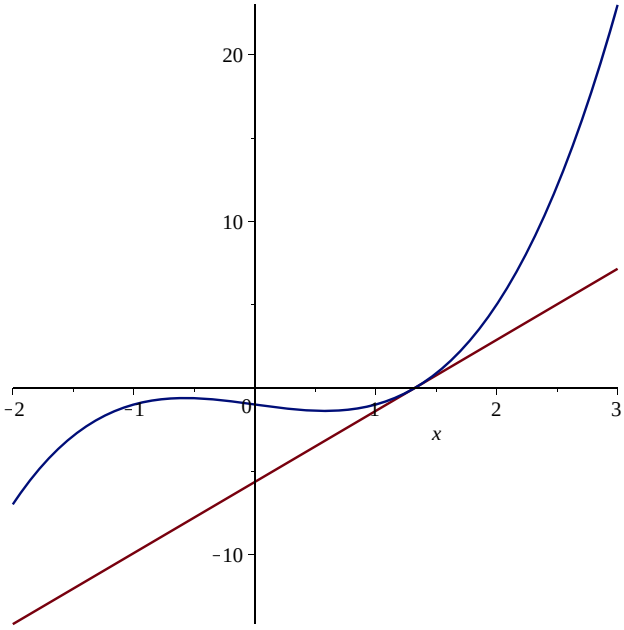
crossa := 1.325200399

1.325200399 **(12)**

Lets repeat this again using the same commands:

```
> a:= evalf(crossa); plot({f(x), (D(f)(a))*(x-a)+f(a)}, x= -2..3);
```

a := 1.325200399




```
> crossa:=a-f(a)/(D(f)(a)); evalf(crossa);
```

$$crossa := 1.324718174$$

$$1.324718174 \qquad \textbf{(13)}$$

Note that we've already found an answer which is accurate up to 2 decimal places, 1.32. To compute the error it is standard to compare the old crossa (which is now called a) with the new cross a. We need to find

|crossa -a | and since the command in Maple for the absolute value is abs we write:

```
> err:= abs(crossa-a);
```

$$err := 0.000482225 \qquad \textbf{(14)}$$

Now suppose we want an error which is less than .0000001. We need to keep repeating these commands. One way to do this would be to use a for loop and a counter j which counts the number of repetitions (up to 10):

for j from 1 by 1 to 10 do plot (tan line at a), crossa:=a-f(a)/(D(f)(a)); evalf(crossa); error:=abs(crossa-a) ; a:=evalf(crossa); od;

Note that we must start by giving the first a and error values to start with. Then the computer will graph the tan line find the cross compute the error and set the new a to be the crossa before repeating the process. However there is no reason to make the computer do j steps if the error is alread small enough. There is also no reason to keep plotting the graph. So we use the following while loop. **Be sure to save your file before clicking enter on the while loop in case it crashes the computer!**

```
> a:=1; err:=1;
```

$$a := 1$$
$$err := 1$$

(15)

```
>  for j from 1 by 1 to 10 while (err>0.0000001) do j;crossa:=a-f
  (a)/(D(f)(a)); err:=evalf(abs(crossa-a)); a:=evalf(crossa); od;
```

$$1$$
$$crossa := \frac{3}{2}$$
$$err := 0.5000000000$$
$$a := 1.500000000$$
$$2$$
$$crossa := 1.347826087$$
$$err := 0.152173913$$
$$a := 1.347826087$$
$$3$$
$$crossa := 1.325200399$$
$$err := 0.022625688$$
$$a := 1.325200399$$
$$4$$
$$crossa := 1.324718174$$
$$err := 0.000482225$$
$$a := 1.324718174$$
$$5$$
$$crossa := 1.324717957$$
$$err := 2.17\ 10^{-7}$$
$$a := 1.324717957$$
$$6$$
$$crossa := 1.324717957$$
$$err := 0.$$
$$a := 1.324717957$$

(16)

Notice that it only takes 6 steps! Now what would happen if we started with a different a:

```
> a:=0; err:=1;
>  for j from 1 by 1 to 30 while (err>0.0000001) do j;crossa:=a-f
  (a)/(D(f)(a)); err:=evalf(abs(crossa-a)); a:=evalf(crossa); od;
```

$$a := 0$$
$$err := 1$$
$$1$$
$$crossa := -1$$
$$err := 1.$$
$$a := -1.$$
$$2$$
$$crossa := -0.5000000000$$
$$err := 0.5000000000$$
$$a := -0.5000000000$$
$$3$$
$$crossa := -3.000000000$$
$$err := 2.500000000$$
$$a := -3.000000000$$
$$4$$
$$crossa := -2.038461538$$

err := 0.961538462
a := −2.038461538
5
crossa := −1.390282147
err := 0.648179391
a := −1.390282147
6
crossa := −0.9116118974
err := 0.4786702496
a := −0.9116118974
7
crossa := −0.3450284959
err := 0.5665834015
a := −0.3450284959
8
crossa := −1.427750701
err := 1.082722205
a := −1.427750701
9
crossa := −0.9424179099
err := 0.4853327911
a := −0.9424179099
10
crossa := −0.4049493526
err := 0.5374685573
a := −0.4049493526
11
crossa := −1.706904617
err := 1.301955264
a := −1.706904617
12
crossa := −1.155756341
err := 0.551148276
a := −1.155756341
13
crossa := −0.6941917921
err := 0.4615645489
a := −0.6941917921
14
crossa := 0.7424945829
err := 1.436686375
a := 0.7424945829
15
crossa := 2.781291994
err := 2.038797411
a := 2.781291994
16
crossa := 1.982722879
err := 0.798569115
a := 1.982722879
17
crossa := 1.536926216

$$err := 0.445796663$$
$$a := 1.536926216$$
$$18$$
$$crossa := 1.357262166$$
$$err := 0.179664050$$
$$a := 1.357262166$$
$$19$$
$$crossa := 1.325663076$$
$$err := 0.031599090$$
$$a := 1.325663076$$
$$20$$
$$crossa := 1.324718789$$
$$err := 0.000944287$$
$$a := 1.324718789$$
$$21$$
$$crossa := 1.324717957$$
$$err := 8.32\ 10^{-7}$$
$$a := 1.324717957$$
$$22$$
$$crossa := 1.324717957$$
$$err := 0.$$
$$a := 1.324717957 \quad \textbf{(17)}$$

This time 10 steps have gone by and the error is still large. So increase the number of steps. If you increase it to 20 steps the error is small but not small enough yet. If you increase it to 30 steps it is done in 22 steps and it says that the error is 0. The error is probably not zero but some number which has 0's for the first ten places.

Exploration: Cut and paste the Newton's Method while loop so you can use it for your work:

Problem A: Try f(x)=sin(x) and starting with a=2 use Newton's method to find the value of Pi up to 8 decimal places.

```
> f := x→sin(x);
```

$$f := x \rightarrow \sin(x) \quad \textbf{(18)}$$

```
> a:=2; err:=1;
>  for j from 1 by 1 to 30 while (err>0.0000001) do j;crossa:=a-f
  (a)/(D(f)(a)); err:=evalf(abs(crossa-a)); a:=evalf(crossa); od;
>
```

$$a := 2$$
$$err := 1$$
$$1$$
$$crossa := 2 - \frac{\sin(2)}{\cos(2)}$$
$$err := 2.185039863$$
$$a := 4.185039863$$
$$2$$
$$crossa := 2.467893675$$
$$err := 1.717146188$$
$$a := 2.467893675$$
$$3$$
$$crossa := 3.266186277$$
$$err := 0.798292602$$

$$a := 3.266186277$$
$$4$$
$$crossa := 3.140943912$$
$$err := 0.125242365$$
$$a := 3.140943912$$
$$5$$
$$crossa := 3.141592654$$
$$err := 0.000648742$$
$$a := 3.141592654$$
$$6$$
$$crossa := 3.141592654$$
$$err := 0.$$
$$a := 3.141592654 \quad \textbf{(19)}$$

>

Problem B: If you would like to design some while an for loops of your own, do them in another file and be careful to save regularly. Loops can ask too much of a computer and cause it to crash if they are accidently written incorrectly. If you want to try this at home, remember that if your computer freezes you can type Ctrl Alt Delete and you will be asked if you would like to End a Task. Choose Maple and the computer will just crash Maple and not all the other programs. Everything you did after your last save will be lost. You can open Maple again and continue working.

If you are interested in a career designing programs that perform calculations you should major in both Mathematics and Computer Science. It is not important to complete two majors but to complete all the requirements for each major. Then plan to get a masters in Numerical Methods (this can be done either in a math dept or a comp sci dept).