Newton's Amazing Method: Estimate π to *How* Many Places?

Introduction

OBJECTIVE: Use *Mathematica* to implement Newton's Method.

To how many places can you estimate π or *e* or $\sqrt{2}$? In this module, Isaac Newton and *Mathematica* team up to help you in this endeavor. Estimates to thousands of places are at your fingertips when you have such an awesome pair to help you. You will also encounter some of the hazards in using Newton's Method.

Technology Guidelines

NOTE: If you have just finished a module, restart *Mathematica* or close the *Kernel* before executing a new module.
TO OPEN CELLS, put your cursor on the right cell bracket and double click.
INITIALIZATION CELLS
When asked if you want to ". . . automatically evaluate all the initialization cells in the notebook . . . ," respond by pressing the "Yes" button.
TO STOP AN EXECUTION
Select the *Kernel* pull-down menu and click on *Abort Evaluation.*
ORDER OF EXECUTION
Execute cells in the order given. Do not skip any Input cells within a given notebook.
SAVING NOTEBOOKS
You can save anytime to any directory you choose, and it is wise to save often.
However, before you do your final save, delete all your output by selecting the
Delete All Output selection under the *Kernel* pull-down menu.
EXPERIENCING MAJOR PROBLEMS
Save if appropriate, then shut down *Mathematica* and start it up again.

Part I: Using Newton's Method to Estimate π

Chapter 4, Section 7, Exercise 20

Setting Up the Problem

For those of us who enjoy mathematical challenges, estimating irrational numbers like π, *e*, and $\sqrt{2}$ to a large number - no, to a very large number of decimal places - is an exercise of enjoyment. Let's see if we can crash the computer!

To estimate π, we will use Newton's method to estimate the zero of the function $f(x) = \tan x$, for $\pi/2 < x < 3\pi/2$. As you know already, the exact answer is π. First, we define the function **f[x_]**. Clear the symbol **x** so that there is no conflict in defining the Newton iteration function. Later on you will want to change the function **f[x_]** and clearing **x** now will avoid complications when you redefine the iteration function for the new **f[x_]**.

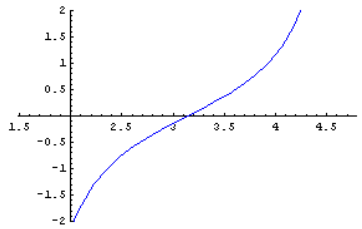
In[1]:=

```
Clear[f, x];

f[x_] = Tan[x];

"> About Mathematica
```

Now we plot f(x) over the specified domain of interest to determine the approximate location of an *x*-value that makes the function zero.

In[3]:=

```
Plot[f[x], {x, Pi / 2, 3 * Pi / 2}, PlotRange → {-
 PlotStyle → {RGBColor[0, 0, 1]}];
```



The plot window is cut down by specifying **PlotRange → {-2, 2}** so that we can see the root clearly.

▪ Applying Newton's Method

We set up Newton's iteration scheme thus.

In[4]:=

```
fprime[x_] = f'[x];

digits = 100;

newt[x_] := SetPrecision[x - f[x]/fprime[x], digit
```

We have set the precision at 100 digits for the estimate of the zero. After we go through this once, you will want to come back and change this value, but leave it at 100 for now.

```
"> About Mathematica
```

Using the graph of the function, we specify a starting value of x=3 for the iterations. We iterate until two consecutive calculated values of *x* are equal to the precision specified by **digits**. The symbol *n* is used to count the number of iterations required to achieve the specified precision. For an explanation of the following commands, click on the "About *Mathematica*" button below the cell.

In[7]:=

```
Clear[x, n, xiterates, y]
x = 3;
n = 0;
xiterates = {{n, x}};
y = x;
x = newt[x];
n++;
xiterates = Append[xiterates, {n, SetPrecisio
While[SetPrecision[y, digits] ≠ SetPrecision[
  y = x; x = newt[x]; n++;
  xiterates = Append[xiterates, {n, SetPrecisi
TableForm[xiterates, TableHeadings -> {None,
Print["The number of iterations is ", n,
  " and the zero is at x = ", SetPrecision[x
```

Out[16]//TableForm=

n	x
0	3
1	3.139707749099462936405777723305947379813997432159102159241675684826655577029322167427334469226421123811504868189588 7`100.
2	3.141592649125255694479438144065764909921239977651270564854383622650175288727148743051796063583604324927448003831971 5`100.
3	3.141592653589793238462643323954443812183769337015590476505602206898517079118787998448851562596003727177309124187390 7`100.
4	3.141592653589793238462643383279502884197169399375105820974944592307816406286069803742263001630626632762744791636834 4`100.
5	3.141592653589793238462643383279502884197169399375105820974944592307816406286208998628034825342117067982148086513282 3`100.
6	3.141592653589793238462643383279502884197169399375105820974944592307816406286208998628034825342117067982148063984269 5`100.

```
The number of iterations is
 6 and the zero is at x =
 3.141592653589793238462643383279502884197
  169399375105820974944592307816406286208
  986280348253421170 68
```

"> About Mathematica

In the cell above, we first initialize the iteration counter, **n**, to 0. We want to store each iteration count and the corresponding value of **x** in a list called **xiterates**, so we initialize the list with the first pair of values, **n** (=0) and **x** (=initial guess). Then we assign the current value of **x** to the symbol **y** for later comparison with the next computed value of **x**. We use the **newt[ ]** function to calculate the next value of **x** and increment the iteration count **n** by 1 with **n++** (at least one iteration will always be performed). Now we enter the **While[ ]** command. Inside the **While[ ]** command, the values of **x** and **y** are tested to see if they are equal. If they are not, the other commands are executed in order: the current value of **x** is again assigned to **y** for the next test; a new value of **x** is computed; **n++** increments the iteration counter **n** by 1; and the new values for **n** and **x** are appended to the list of **xiterates**. This procedure is repeated until the condition is not satisfied, that is, until two consecutive estimates of **x** are equal to the precision specified in **digits**.

From the previous results, we estimate π to 100 decimal places.

In[18]:=

```
ourestimateofπ = SetPrecision[x, digits]
```

Out[18]=

```
3.14159265358979323846264338327950288419\
 69399375105820974944592307816406286208\
 98628034825342117068
```

Testing the Results

How do we know that Newton's method is working correctly and that our estimate of π is correct? We can compare it with *Mathematica*'s estimate of π to the same number of digits.

In[19]:=

```
π - SetPrecision[ourestimateofπ, 100]
```

Out[19]=

$$0.\times 10^{-100}$$

If *Mathematica*'s estimate of π is correct to the specified number in **digits**, then so is ours. Not bad!

In[20]:=

```
π - SetPrecision[ourestimateofπ, 114]
```

Out[20]=

$$2.252901\times 10^{-107}$$

What is truly amazing is the rate at which Newton's method converges to the root. We can achieve 100 digits of accuracy with so few iterations - Wow! And because it takes so few iterations, it is also very fast. But wait, it gets even better. Go back up and change **digits** to 1000 or even 10000. See what happens. (Be aware, however, that at some point you will run up against the limitations of your computer, specifically, RAM and cache storage on your hard drive, and it may **crash**.) Also, when you increase the precision, you have two choices for the initial value of **x**. If you don't reset it with **x=***initial value*, the iteration will start with the last value calculated (i.e., *x* to 100 digits). If you reset it, then the first 100 iterations will be repeated on the way to 1000, 10000, 1000000, ...

You Try It: Part I

More About π

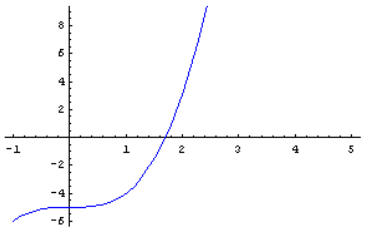
You may also want to do some research to find out what the current best estimate of π is. Research the history of the efforts that mathematicians have put into this problem.

Check out $\sqrt{2}$

See if you can construct functions so that you can use the above code to estimate $\sqrt{2}$. Begin by defining a function that is zero when $x=\sqrt{2}$. We suggest that you try a polynomial. Replace the terms in red with the appropriate function, and change the bounds over which you plot f(x), if you wish.

In[21]:=

```
Clear[f, x];
f[x_] = x^3 - 5;
Plot[f[x], {x, -1, 5}, PlotStyle → {RGBColor[0
```



Now set up Newton's iteration scheme. You can set **digits** to whatever precision you wish.

In[22]:=

```
fprime[x_] = f'[x];
digits = 50;
newt[x_] := SetPrecision[x - f[x]/fprime[x], digit
```

Using the graph of your function, specify a starting value of **x** (in red) for the iterations. We iterate until two consecutive calculated values of **x** are equal to the precision specified by **digits**, and we use the symbol **n** to count the number of iterations required to achieve the specified precision.

In[23]:=

```
Clear[x, n, xiterates, y]
x = 2;
n = 0;
xiterates = {{n, x}};
y = x;
x = newt[x];
n++;
xiterates = Append[xiterates, {n, SetPrecisio
While[SetPrecision[y, digits] ≠ SetPrecision[
  y = x; x = newt[x]; n++;
  xiterates = Append[xiterates, {n, SetPrecis
TableForm[xiterates, TableHeadings -> {None,
Print["The number of iterations is ", n,
  " and the zero is at x = ", SetPrecision[x

The number of iterations is
 7 and the zero is at x =
 1.709975946676696989353108872543860109868
  0551105431
```

Out[23]=

```
Null
```

n	x
0	2
1	1.75000000000000000000000000000000000000
2	1.71088435374149659863945578231292517
3	1.70997642891697479253088511483046297
4	1.70997594667683298869189597018863307
5	1.70997594667669698935310888336028137
6	1.70997594667669698935310887254386017
7	1.70997594667669698935310887254386017

The output from the code above renders your estimate as the following. Compare it to the $\sqrt{2}$, and see if it has given the desired precision. It clearly does not if you did not change the function **f[x_]** (in red).

In[24]:=

```
ourestimate = SetPrecision[x, digits]

√2 - ourestimate
```

Out[24]=

```
1.709975946676696989353108872543860109868
 0551105431
```

Out[25]=

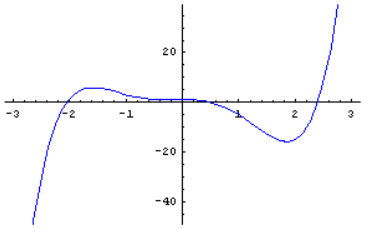
```
-0.29576238430360194055142014833416203129
  83832351661
```

Part II: Hazards in Applying Newton's Method

Let's select a polynomial that we can't factor and begin by looking at its graph.

In[26]:=

```
Clear[f, x];

f[x_] = x^5 - 5 x^3 - 2 x^2 + 1;

pf = Plot[f[x], {x, -3, 3}, PlotStyle → RGBColo
```



We set up Newton's iteration scheme and specify 20-digit precision.

In[29]:=

```
fprime[x_] = f'[x];

digits = 20;

newt[x_] := SetPrecision[x - f[x]/fprime[x], digit
```

This function has three zeros in the domain over which the plot extends. What happens when we specify a starting value of 2?

We iterate until two consecutive calculated values of **x** are equal to the precision specified by **digits**. The symbol *n* is used to count the number of iterations required to achieve the specified precision. For an explanation of the following commands, click on the "About *Mathematica*" button below the cell.

In[32]:=

```
Clear[x, n, xiterates, y]

x = 2;

n = 0;

xiterates = {{n, x}};

y = x;

x = newt[x];

n++;

xiterates = Append[xiterates, {n, SetPrecisio

While[SetPrecision[y, digits] ≠ SetPrecision[
 y = x; x = newt[x]; n++;
 xiterates = Append[xiterates, {n, SetPrecisi

TableForm[xiterates, TableHeadings -> {None,

Print["The number of iterations is ", n,
 " and the zero is at x = ", SetPrecision[x,
```

Out[41]//TableForm=

n	x
0	2
1	3.25`20.
2	2.80789955214978214058250856778967691697`20.
3	2.53718280431355258049702600951693577511`20.
4	2.42174162330835861020965218020243064071`20.
5	2.40083733782435009331664440660881704023`20.
6	2.40019741845712916528138454326242828143`20.
7	2.40019683087362685607455054105314563877`20.
8	2.40019683087313178531413208638383139015`20.
9	2.40019683087313178527317722443790160014`20.

```
The number of iterations is
 9 and the zero is at x =
 2.4001968308731317853
```

What if we had started at x=1.5?

In[43]:=

```
Clear[x, n, xiterates, y]

x = 1.5;

n = 0;

xiterates = {{n, x}};

y = x;

x = newt[x];

n++;

xiterates = Append[xiterates, {n, SetPrecisio

While[SetPrecision[y, digits] ≠ SetPrecision[
 y = x; x = newt[x]; n++;
 xiterates = Append[xiterates, {n, SetPrecisi

TableForm[xiterates, TableHeadings -> {None,

Print["The number of iterations is ", n,
 " and the zero is at x = ", SetPrecision[x,
```

Out[52]//TableForm=

n	x
0	1.5`
1	0.61471861471861466430510745340143330395`20.
2	0.50283191084360848768813435217893920492`20.
3	0.48291779156501070408890623055014971927`20.
4	0.48228571347382140444169766416310027375`20.
5	0.48228508357646991850221795836066800277`20.
6	0.48228508357584461930303916760655870169`20.
7	0.48228508357584461930303916760520624444`20.

```
The number of iterations is
 7 and the zero is at x =
 0.4822850835758446193
```

This time the iterations converged to a different value of **x**. Why did that happen? Can you tell precisely where the break would occur? Does it have anything to do with the derivative of the function?

What if we had started at x=0?

In[54]:=

```
Clear[x, n, xiterates, y]
```

```
x = 0;

n = 0;

xiterates = {{n, x}};

y = x;

x = newt[x];

n++;

xiterates = Append[xiterates, {n, SetPrecisio

While[SetPrecision[y, digits] ≠ SetPrecision[
 y = x; x = newt[x]; n++;
 xiterates = Append[xiterates, {n, SetPrecisi

TableForm[xiterates, TableHeadings -> {None,

Print["The number of iterations is ", n,
 " and the zero is at x = ", SetPrecision[x,
```

```
Power::infy :
 Infinite expression 1/0 encountered. More…

∞::indet : Indeterminate expression
   1 + ComplexInfinity + ComplexInfinity +
    ComplexInfinity encountered. More…

∞::indet : Indeterminate expression
   ComplexInfinity + ComplexInfinity +
    ComplexInfinity encountered. More…
```

Out[63]//TableForm=

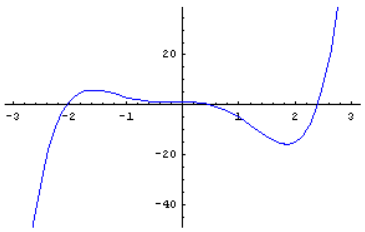
n	x
0	0
1	ComplexInfinity
2	Indeterminate

```
The number of iterations is 2
  and the zero is at x = Indeterminate
```

This time Newton's method didn't work at all. What is it that has given the $\frac{1}{0}$ expression? It might help to look at the graph again.

In[65]:=

```
Show[pf];
```



Let's try x = -1.5.

In[66]:=

```
Clear[x, n, xiterates, y]

x = -1.5;
```

```
n = 0;

xiterates = {{n, x}};

y = x;

x = newt[x];

n++;

xiterates = Append[xiterates, {n, SetPrecisio

While[SetPrecision[y, digits] ≠ SetPrecision[
 y = x; x = newt[x]; n++;
 xiterates = Append[xiterates, {n, SetPrecisi

TableForm[xiterates, TableHeadings -> {None,

Print["The number of iterations is ", n,
 " and the zero is at x = ", SetPrecision[x,
```

Out[75]//TableForm=

n	x
0	-1.5`
1	0.871794871794871806258697688463144004 34`20.
2	0.594330051452036724127638995073343306 29`20.
3	0.497622329462858066500249037984400939 34`20.
4	0.482642084144975446107915659706441269 32`20.
5	0.482285284253765960836135512102067418 9`20.
6	0.482285083575908086175576064350848496 05`20.
7	0.482285083575844619303038557751546670 99`20.
8	0.482285083575844619303038557746306748 93`20.

```
The number of iterations is
 8 and the zero is at x =
 0.48228508357584461930
```

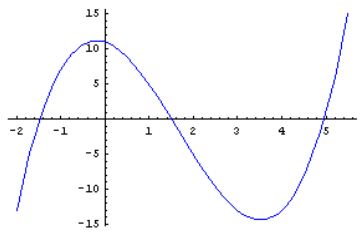
Why does this converge to the zero to the right instead of to the closer zero to the left?

You Try It: Part II

Here is a cubic polynomial. Study its graph.

In[77]:=

```
Clear[f, x];

f[x_] = x^3 - 5 x^2 - 2 x + 11;

pf = Plot[f[x], {x, -2, 5.5}, PlotStyle → RGBC

fprime[x_] = f'[x];

digits = 20;

newt[x_] := SetPrecision[x - f[x]/fprime[x], digit
```



You decide where you should begin your iterations in order to compute each of the zeros of this polynomial. By changing the term in red, you determine which starting values converge to which zero. Check it out with the given starting point of x = 0.

In[83]:=

```
Clear[x, n, xiterates, y]

x = 0;
n = 0;
xiterates = {{n, x}};
y = x;
x = newt[x];
n++;
xiterates = Append[xiterates, {n, SetPrecisio
While[SetPrecision[y, digits] ≠ SetPrecision[
 y = x; x = newt[x]; n++;
 xiterates = Append[xiterates, {n, SetPrecisi
TableForm[xiterates, TableHeadings -> {None,

Print["The number of iterations is ", n,
 " and the zero is at x = ", SetPrecision[x,
```

Out[84]//TableForm=

n	x
0	0
1	5.5`20.
2	5.05185185185185185185185185185540464763`20.
3	4.95954124047054114462870755343416766942`20.
4	4.95567782195552451670163524316267277 18`20.
5	4.95567115855749833026146705825486947813`20.
6	4.95567115853769190583509779841672475535`20.
7	4.95567115853769190578258577173365040 38`20.

```
The number of iterations is
 7 and the zero is at x =
 4.9556711585376919058
```

What happened? It seems to have completely skipped over a zero! Why did this happen?

About *Mathematica*

Whenever you define a new function like **f[x_]** in the preceding input cell, you should **Clear[f, x]** so that all previous definitions of the function and the independent variable are deleted. It is very important here because later we will want to redefine **f[x_]** with a new formula in terms of **x**. If numeric values were previously assigned to **x** and we did not clear them, our new **f[x_]** would be defined with the numeric value in place of **x**, giving a constant function. Clearing **x** ensures that the intended formula is assigned to **f[x_]** rather than a constant value. Sometimes you want *Mathematica* to remember a previous definition of a symbol, but many times you do not. Managing symbol definitions in *Mathematica* is very important because mismanaging them can produce some very misleading and confusing results. Learn more about defining symbols in *Mathematica* in "Overview of *Mathematica*: Assignment Commands in *Mathematica*," included in this supplement. You can also go to the Help Browser and type **Clear**. Go back.

The **SetPrecision[ ]** command allows us to specify the number of digits for the numbers in an expression. To learn more about precision and the **SetPrecision [ ]** command, go to the Help Browser and type **Precision**. Go back.

The **While[ ]** command is useful for performing an operation repeatedly as long as a specified condition is met. In the example above, the condition is inequality of consecutive estimates of **x** within the precision specified in **digits**. This test condition is the first argument of the **While[ ]** command. (Many numerical routines on calculators and computers work this way. They keep improving the numerical estimate of the value until consecutive approximations are equal to each other within the working precision limits of the calculator or computer processor.) The second argument in the **While[ ]** command is a string of operations to perform while the condition specified in the first argument is satisfied. Semicolons are used to separate the operations in the string. Go back.

Created by *Mathematica* (August 9, 2005)