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

Note: You may notice differences between this Maple worksheet and the equivalent Mathematica notebook. These differences were introduced to preserve the content of these modules and were necessary because of major functional differences between Maple and Mathematica.

Introduction

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

To how many decimal places can you estimate π or $\sqrt{2}$? In this module, Isaac Newton and

Maple team up to help you in this endeavor. Estimates to thousands of decimal 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 worksheet, **restart** Maple before executing a new worksheet.
TO OPEN SECTIONS,
Click on the **PLUS** sign at the left hand side of the screen or select **Expand All Sections** from the **View** drop down menu.
TO STOP AN EXECUTION
Click on **STOP** button from the toolbar.
ORDER OF EXECUTION
Execute commands in the order given. Do not skip any Maple Input lines within a given worksheet
Alternatively, you can execute the entire worksheet by selecting the **Execute Worksheet** command from the **Edit** drop down menu.
SAVING WORKSHEETS.
You can save anytime to any directory you choose, and it is wise to save often.
EXPERIENCING MAJOR PROBLEMS
Save if appropriate, and then sut down Maple 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 of decimal places is an exercise of enjoyment.

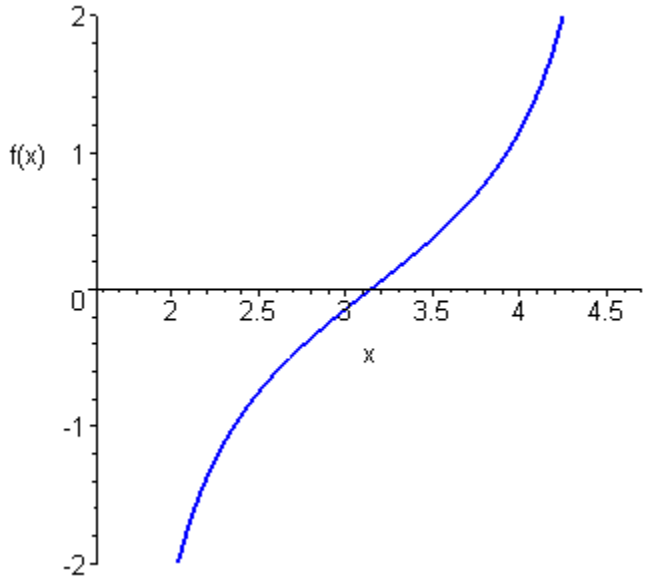
To estimate Pi, we will use Newton's method to estimate the 0 of the function $f(x) = \tan(x)$, for $\frac{\pi}{2} < x < \frac{3\pi}{2}$, since the exact value of this root is π. First we define the function f(x).

```
> restart;
  NumericEventHandler(division_by_zero=exception):

  f(x):=tan(x);
```

$$f(x) := \tan(x)$$

We start by plotting $\tan(x)$ over the specified domain to determine the approximate location of an *x*-value that makes the value of the function equal to 0.

```
> plot(f(x), x=Pi/2..3*Pi/2, -2..2, color=blue, thickness=2, labels=["x","f(x)"]);
```



The plot window is cut down by specifying the plot range to **-2..2,** so that we can see the root clearly.

Applying Newton's Method

We will now write a *Maple* procedure which implements Newton's Method. It takes as

input a function *f*, the variable *x*, and a starting guess for the zero. At each iteration, it prints out the iteration number *n* and the calculated value of *x* for that iteration. When two consecutive calculated values of *x* are equal to the precision specified by **Digits**, the procedure terminates and returns the value of the root.

```
> Newton := proc(f, x, start_value)
   local formula, current_value, next_value, residual, n:
   formula := x - f / diff(f,x):
   current_value := evalf(start_value):
   residual := 1:
   n := 0:
   while residual <> 0 do
    next_value := evalf(eval(formula, x = current_value),Digits+2):
    residual := evalf(next_value - current_value):
    printf("n = %a    %a = %a \n", n, x, current_value):
    n := n + 1:
    current_value := next_value:
   od:
  end:
```

We 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.

```
> Digits := 100:
```

Using the graph of the function, we specify a starting value of $x = 3$ for the iterations.

```
> our_estimate := Newton(tan(x),x,3);
```

```
n = 0    x = 3.

n = 1    x =
3.139707749099462936405777723305947379813997432159102159241675684826655577(

n = 2    x =
3.141592649125255694479438144065764909921239977651270564854383622650175288

n = 3    x =
3.141592653589793238462643323954443812183769337015590476505602206898517079

n = 4    x =
3.141592653589793238462643383279502884197169399375105820974944592307816406

n = 5    x =
3.141592653589793238462643383279502884197169399375105820974944592307816406
```

$$our\_estimate := 3.14159265358979323846264338327950288419716939937510582097494\cdot 40628620899862803482534211706798$$

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 *Maple's* estimate of π to the same number of digits.

> **evalf(Pi - our_estimate);**

$$0.$$

If Maple's estimate of Pi is correct to the specified number of digits (which it is), then so is ours. Not bad! What is truly amazing is the rate at which Newton's method converges to the zero. Six iterations for 100 digits of accuracy - 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. As you increase **Digits**, you may want to change your initial guess to be the value returned by your previous run of Newton (i.e. $our\_estimate := \mathrm{Newton}(\tan(x), x, our\_estimate)$) to avoid recomputing the same initial iterations.

You Try It - Part I

More About π

Do some research to find out what the current best estimate of π is, and what method was used to compute it. Research the history of the efforts that mathematicians have put into this problem.

Check out $\sqrt{2}$

Construct a function **f** so that the Newton procedure can be used to estimate $\sqrt{2}$. Begin by defining a function that is 0 when $x = \sqrt{2}$. It is suggested that you try a polynomial.

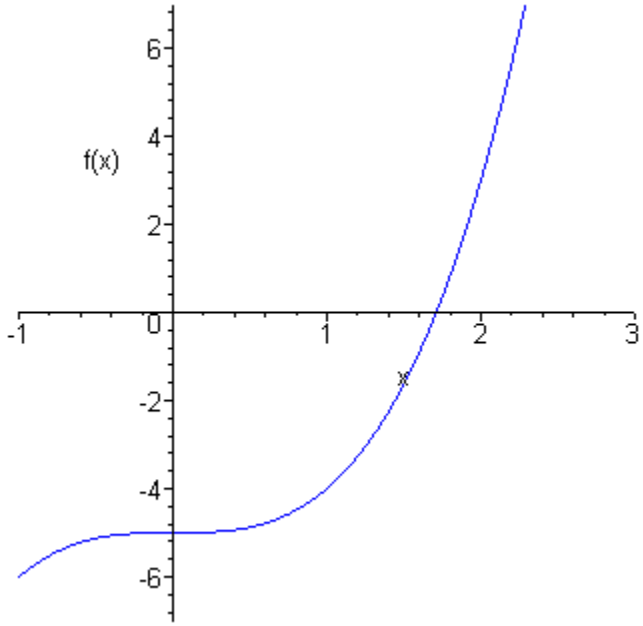
Replace the polynomial given below with your function.

> **f := x^3 - 5;**

$$f := x^3 - 5$$

Change the bounds over which you plot your function **f** if desired.

> **plot(f, x=-1..3, -7..7, color=blue, labels=["x","f(x)"]);**



Change the precision to 50 digits.

> **Digits:=50:**

Use the graph above to pick an initial estimate of the zero.

> **our_estimate := Newton(x^3-5,x,2);**

n = 0 x = 2.

n = 1 x = 1.7500000000000000000000000000000000000000000000000000

n = 2 x = 1.7108843537414965986394557823129251700680272108843537

n = 3 x = 1.7099764289169747925308851148304629067312269434384800

n = 4 x = 1.7099759466768329886918959701886330858146021596408530

n = 5 x = 1.7099759466766969893531088833602813504545357482234640

n = 6 x = 1.7099759466766969893531088725438601098680551105430550

our_estimate := 1.7099759466766969893531088725438601098680551105430550

Compare your result to $\sqrt{2}$, and see if it has given the desired precision. It clearly does not if you haven't changed **f.**

```
> evalf( sqrt(2) - our_estimate );
```

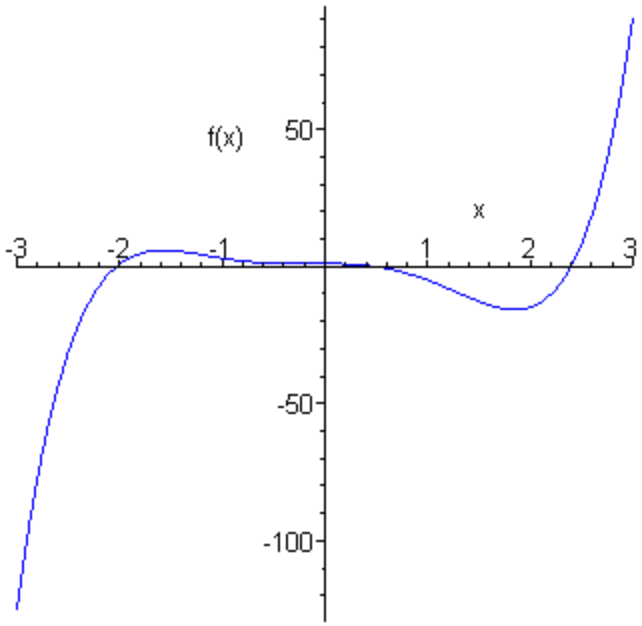
-0.29576238430360194055142014833416203129838323516 62

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.

```
> f:=x^5-5*x^3-2*x^2+1;
  plot(f, x=-3..3, color=blue, labels=["x","f(x)"]);
```

$$f := x^5 - 5x^3 - 2x^2 + 1$$



From the graph we see that there is a zero after $x = 2$. Use this as our initial guess to Newton's method.

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.

```
> Digits:=20:
  Newton(f,x,2);
```

n = 0 x = 2.

n = 1 x = 3.250000000000000000000

n = 2 x = 2.807899552149782140582

n = 3 x = 2.537182804313552580496

n = 4 x = 2.421741623308358610209

n = 5 x = 2.400837337824350093316

n = 6 x = 2.400197418457129165281

n = 7 x = 2.400196830873626856075

n = 8 x = 2.400196830873131785314

2.400196830873131785314

So we have found the zero close to $x = 2$. What if we had instead made an initial guess of $x = 1.5$?

```
> Newton(f,x,1.5);
```

n = 0 x = 1.5

n = 1 x = .614718614718614718614718

n = 2 x = .502831910843608502275 5

n = 3 x = .482917791565010704965 0

n = 4 x = .482285713473821404443 4

n = 5 x = .482285083576469918502 3

n = 6 x = .482285083575844619303 0

0.482285083575844619303 1

This time the iterations converged to a different value of *x*. Why did that happen? Can you tell

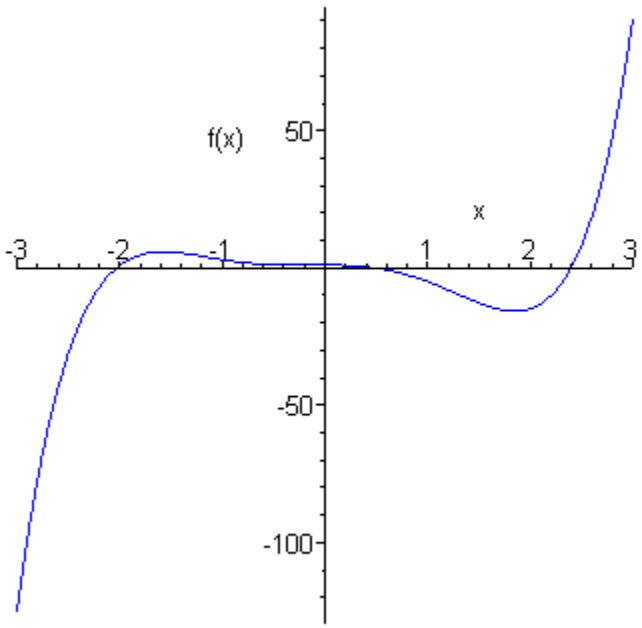
precisely where the break would occur? Does it have anything to do with the derivative of the function? What if we had started at $x = 0$?

> **Newton(f,x,0);**

Error, (in Newton) numeric exception: division by zero

This time Newton's method didn't work at all. Why did this happen? It might help to look at the graph again. How might we modify the procedure Newton so that this is avoided?

> **plot(f,x=-3..3, color=blue, labels=["x","f(x)"]);**



Let's try an initial guess of $x = -1.5$.

> **Newton(f,x,-1.5);**

n = 0 x = -1.5

n = 1 x = .871794871794871794872

n = 2 x = .594330051452036719677 4

n = 3 x = .497622329462858065424 3

n = 4 x = .482642084144975446058 8

n = 5 x = .482285284253765960836 2

n = 6 x = .482285083575908086 1756

n = 7 x = .482285083575844619 3030

0.48228508357584461931

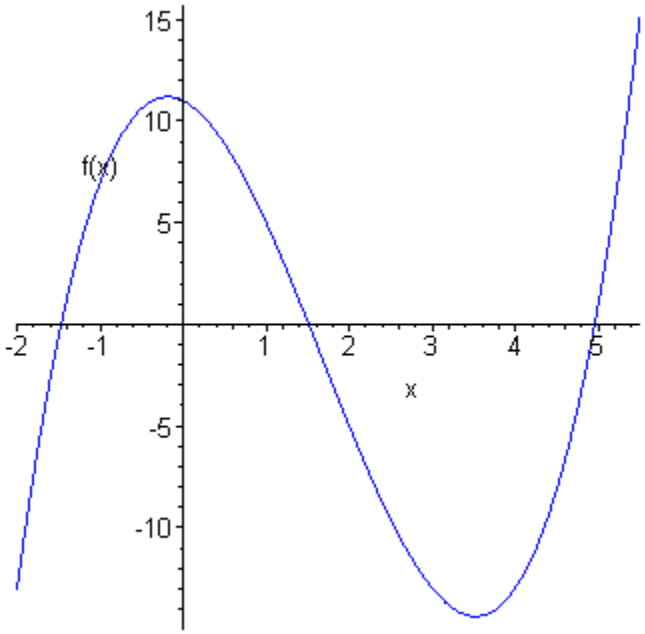
Why does this converge to the zero to the right instead of the closer zero to the left?

You Try It - Part II

Here is a cubic polynomial. Study its graph.

```
> f:=x^3-5*x^2-2*x+11;
  plot(f,x=-2..5.5, color=blue,labels=["x","f(x)"]);
```

$$f := x^3 - 5x^2 - 2x + 11$$



You decide where you should begin your iterations in order to compute each of the zeros of this polynomial. By changing the initial value, you determine which starting values converge to which zero. We start with $x = 0$.

```
> Newton(f,x,0);
```

n = 0 x = 0.

n = 1 x = 5.500000000000000000 0

n = 2 x = 5.051851851851851851 52

n = 3 x = 4.959541240470541144629

n = 4 x = 4.955677821955524516702

n = 5 x = 4.955671158557498330263

n = 6 x = 4.955671158537691905836

4.955671158537691905830

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

Animation of Newton's Method

This Maple procedure shows Newton's method in action.

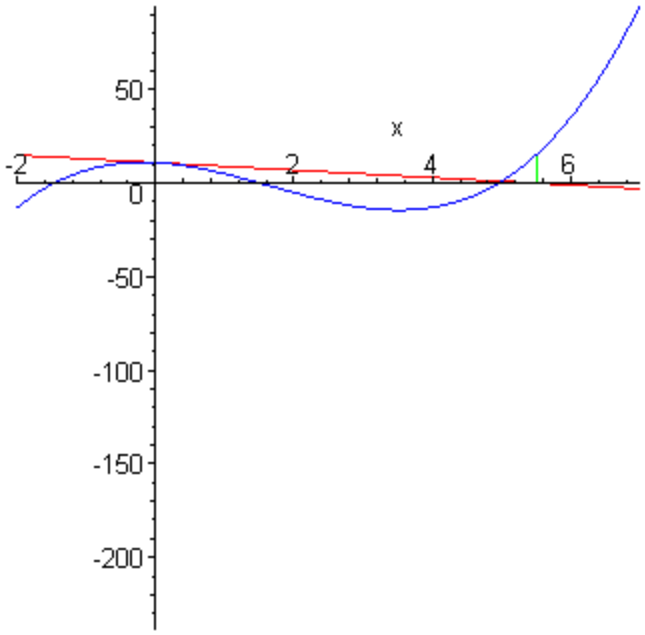
```
> NewtonPlot := proc(f, x, start_value, xrange)
   local p, p1, p2, formula, current_value, next_value, residual, n, MAX_FRAMES:
   MAX_FRAMES := 20:
   formula := x - f / diff(f,x):
   current_value := evalf(start_value):
   residual := 1:
   n := 0:
   p1 := plot(f,x=xrange, args[5..nargs], color=blue):
   p:= NULL:
   while residual <> 0 do
    if n > MAX_FRAMES then break fi:
    next_value := evalf(eval(formula, x = current_value),Digits+2):
    residual := evalf(next_value - current_value):
    if residual = 0 then break fi:
    p2 := plots[pointplot]({[next_value,0],[next_value,eval
  (f,x=next_value)]},connect=true,color=green):
    p := p, plots[display](p1,
  p2,          plot(eval(f,x=current_value)/(current_value-next_value)*(x-next_value),x=xrang
    n := n + 1:
    current_value := next_value:
   od:
   plots[display](p,insequence=true):
  end:

> Digits:=20:
```

Try different starting values (the third argument in **NewtonPlot( )**) so that Newton's method will converge on each of the three zeros. Also, try to find the break-over points where Newton's method will converge to one root if the starting value is on one side of the break-over point, and to another root if the starting value is on the other side.

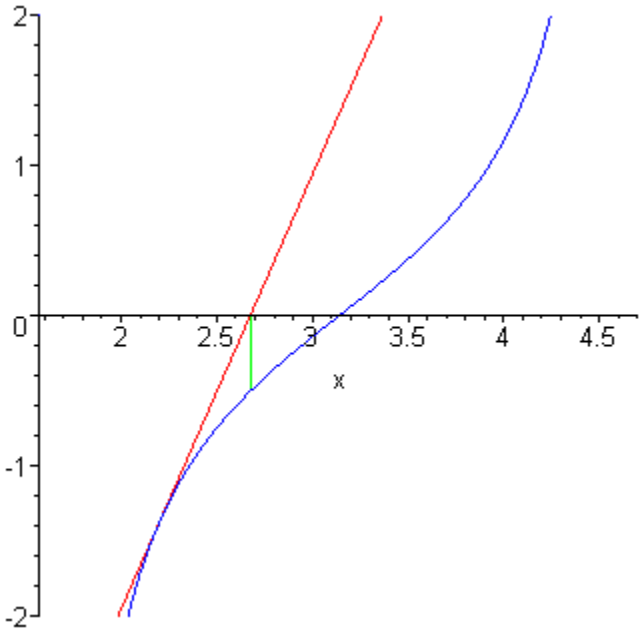
To animate, click on the graph below to bring up the animation toolbox at the top of the worksheet. Press the **play** button (the large arrow) to play the animation. Change the speed by

clicking on the **double arrow** buttons, or show one frame at a time by pressing the **arrow followed by a vertical bar**.

> **NewtonPlot(x^3-5*x^2-2*x+11,x,0,-2..7);**



> **NewtonPlot(tan(x),x,2.2,Pi/2..3*Pi/2,-2..2);**



>