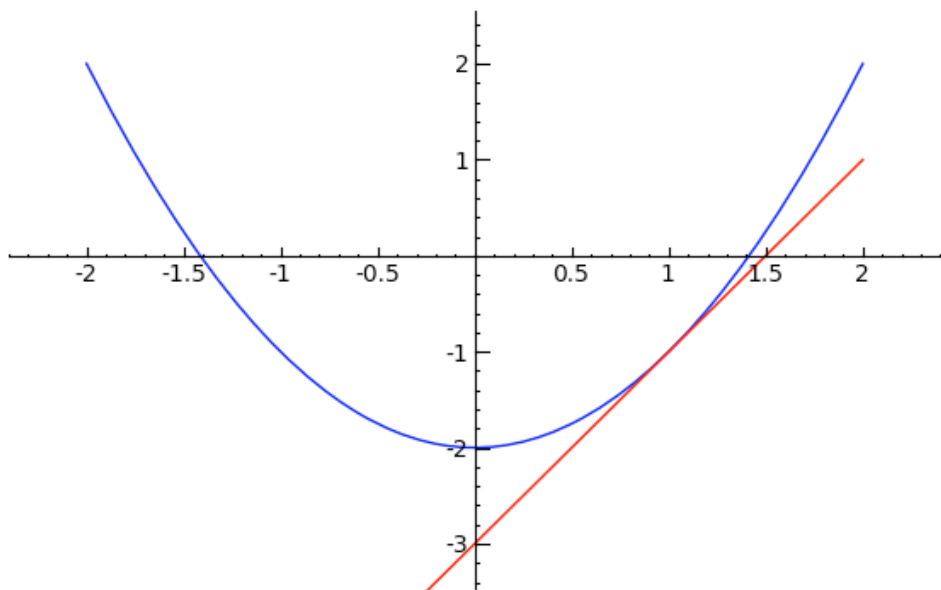


Calc A Lab 4 Solutions

Problem 1

The slope of the tangent line will be $y'(1) = 2(1) = 2$. The tangent line goes through the point $(1, -1)$; so its equation will be $y + 1 = 2(x - 1)$, or $y = 2x - 3$. This line is sketched below. It seems clear from the picture that the point where the tangent line crosses the x -axis is a better approximation to $\sqrt{2}$ (the point where the parabola crosses the axis) than is $x = 1$. This crossing point of the tangent line happens where $0 = 2x - 3$, i.e., at $x = 3/2 = 1.5$.

```
p1 = plot(x^2-2, (x, -2, 2), color='blue')
p2 = plot(2*x-3, (x, -2, 2), color='red')
show(p1+p2, ymin=-3, ymax=2)
```



The difference between x_1 and $\sqrt{2}$ is therefore

```
n(1.5-sqrt(2))
```

```
0.0857864376269049
```

Problem 2

The tangent line touches the curve $y = x^2 - 2$ at the point $(\frac{3}{2}, \frac{1}{4})$. The slope of the tangent line will be $y'(1.5) = 2(1.5) = 3$. The equation of the tangent line will therefore be $y - \frac{1}{4} = 3(x - \frac{3}{2})$, or $y = 3x - \frac{17}{4}$. This line crosses the x -axis at the point where $0 = 3x - \frac{17}{4}$, i.e., at $x = \frac{17}{12} \approx 1.4167$. This is closer to $\sqrt{2}$ than the previous estimate of 1.5.

```
p1 = plot(x^2-2, (x, 0, 1.7), color='blue')
p2 = plot(2*x-3, (x, 0, 1.7), color='red')
p3 = plot(3*x-17/4, (x, 0, 1.7), color='red')
p4 = line([[1,0],[1,-1]],rgbcolor='black')
p5 = line([[1.5,0],[1.5,0.25]],rgbcolor='black')
```

```
show(p1+p2+p3+p4+p5, ymin=-2, ymax=1)
```



```
2.859283843333951225327771682329396813996269230689024810345288803495\
40615971192416486966994977356554757441891385575818973e-49
```

x7 - RealField(400)(sqrt(2))

```
2.890477193215364553280018738096114864083684518002198364659182137891\
58110855633485944411474386994507822046463593698410449e-98
```

What these differences seem to show is that the number of correct digits seems roughly to double every time we do Newton: x_6 is accurate to 49 digits, and x_7 is accurate to 98 digits. Newton's method therefore converges on the root with astounding rapidity.

I'd be content with people just noticing this empirical fact, but those of you with some mathematical background might be interested in trying to prove this result. Here's one way to do it. Suppose that $x_0 - \sqrt{2} = \epsilon$ for some small number ϵ . I claim that $0 < x_1 - \sqrt{2} < \epsilon^2/2^{3/2}$. Thus, knowing that $\epsilon < 3 \times 10^{-49}$ (as happens for x_6) guarantees that the error at the next step will be at most $9 \times 10^{-98}/2^{3/2} \approx 3 \times 10^{-98}$ (as it is for x_7).

So let's see:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = x_0 - \frac{x_0^2 - 2}{2x_0} = \frac{x_0}{2} + \frac{2}{x_0} = \frac{1}{2} \left[x_0 + \frac{2}{x_0} \right].$$

Now, this is an interesting formula in its own right. It says that x_1 is the average of two numbers, x_0 and $2/x_0$. These two numbers multiply to give 2; so if $x_0 > \sqrt{2}$, then $2/x_0 < \sqrt{2}$. We might therefore expect their average to be a better approximation to $\sqrt{2}$ than is x_0 . In other words, we could easily have decided to approximate $\sqrt{2}$ using this formula for x_1 even if we had never heard of Newton.

OK, so let's keep going. We know that $x_0 = \sqrt{2} + \epsilon$; so

$$x_1 = \frac{1}{2} \left[\sqrt{2} + \epsilon + \frac{2}{\sqrt{2} + \epsilon} \right] = \frac{1}{2(\sqrt{2} + \epsilon)} \left[(\sqrt{2} + \epsilon)(\sqrt{2} + \epsilon) + 2 \right] = \frac{1}{2(\sqrt{2} + \epsilon)} \left[4 + 2\sqrt{2}\epsilon + \epsilon^2 \right].$$

Subtract $\sqrt{2}$, and you get

$$\begin{aligned} x_1 - \sqrt{2} &= \frac{1}{2(\sqrt{2} + \epsilon)} \left[4 + 2\sqrt{2}\epsilon + \epsilon^2 \right] - \sqrt{2} \\ &= \frac{1}{2(\sqrt{2} + \epsilon)} \left[4 + 2\sqrt{2}\epsilon + \epsilon^2 - 2(\sqrt{2} + \epsilon)\sqrt{2} \right] \\ &= \frac{\epsilon^2}{2(\sqrt{2} + \epsilon)} < \frac{\epsilon^2}{2\sqrt{2}}, \end{aligned}$$

as desired.

See? If you just have confidence that you can do the algebra, then proving stuff like this doesn't require anything you don't already know.

The problem also asked how many times you would need to apply Newton if you wanted 1000 digit accuracy. After 7 uses of Newton, we have 98 digits. The number is slightly more than doubling each time, so we'd need 11 applications of Newton to get to 1000 digit accuracy, and then 10 more (21 total) to get 1,000,000 digit accuracy. Amazing, isn't it?

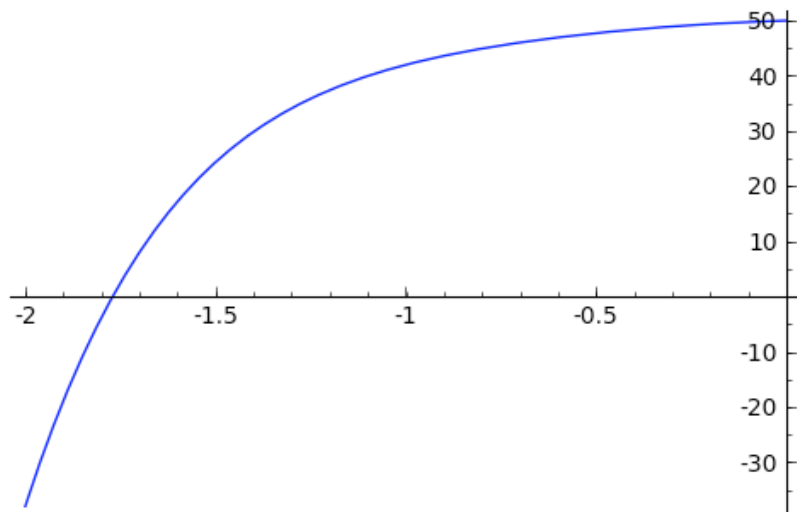
Problem 5

Using out function `newton`, we can make quick work of this.

```
f(x) = -x^6-5*x^2+2*x+50
f(x)
```

$$-x^6 - 5x^2 + 2x + 50$$

plot(f(x),(x,-2,0))



newton(f, -2.0)

-1.82242990654206

newton(f, _)

-1.77353161687131

newton(f, _)

-1.77038234813877

newton(f, _)

-1.77037011162506

newton(f, _)

-1.77037011144116

newton(f, _)

-1.77037011144116

So it looks like the root is roughly at $x = -1.77037011144116$.

Near this root, the graph of f is increasing and concave down. One needs only sketch tangent lines in one's mind to see that these lines will always strike the x -axis to the left of the actual root.

Problem 6

The function $f(x) = x^2 - 2$ that we have looked at for most of this lab is increasing and concave up near its root at $\sqrt{2}$. Thinking about the geometry makes it pretty clear that for any function having those properties, Newton's method will produce approximations that are larger than the actual root. The same thing will happen if f is decreasing and concave down. In the other two cases, where f is increasing and concave down or where f is decreasing and concave up, Newton will produce estimates smaller than the actual root.

Problem 7

What happens is shown algebraically and graphically below

$$f(x) = 4x^5 - 61x^3 - 480$$

$f(x)$

$$4x^5 - 61x^3 - 480$$

`newton(f, -3.0)`

`4.22222222222222`

`newton(f, _)`

`4.12656783456478`

`newton(f, _)`

`4.11914583564285`

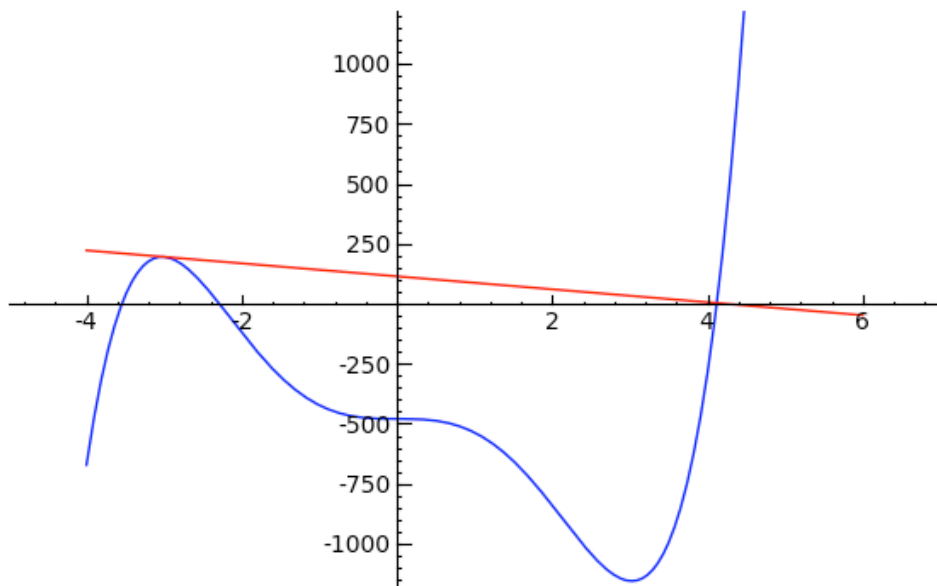
`newton(f, _)`

`4.11910324568429`

`p1 = plot(f(x), (x, -4, 6), color='blue')`

`p2 = plot(f(-3)+diff(f,x)(-3)*(x+3), (x, -4, 6), color='red')`

`show(p1+p2, ymin=-1000, ymax=1000)`



Newton's method started by following the tangent line to the curve at the point $x = -3$. Unfortunately, $x = -3$ is close to a maximum for f ; so the tangent line ends up intersecting the x -axis not near the root between $x = -3$ and $x = -2$, but near another root at about $x = 4$. Repeating Newton closes in rapidly upon this second root.

So what would we do if we wanted to locate the root between -3 and -2 ? Well, we'd need to start with our first estimate closer to actual root. For instance, here's what would happen if our starting point were $x = -2.5$:

`newton(f, -2.5)`

`-2.27241379310345`

```

newton(f, _)
-2.28839601949867

newton(f, _)
-2.28843083516269

newton(f, _)
-2.28843083534186

newton(f, _)
-2.28843083534186

```

The root we were looking for is at about $x = -2.28843083534186$.

So one way Newton's method can give us trouble is if we start too far from the root we want to approximate. If we wanted to write code to use Newton in root finding, we would therefore have to have some means for assuring that we were close enough to the root before we started using Newton.

Problem 8

Another much rarer but far more catastrophic way that Newton's method can give us trouble is shown by the function $f(x) = x^{1/3}$. Here's what happens if you start at $x = 1$ and try to find the root at $x = 0$.

```

f(x) = x^(1/3)
f(x)
x1/3

newton(f, 1.0)
-2.000000000000000

newton(f, _)
4.000000000000000 - 2.22044604925031 × 10-15i

newton(f, _)
-8.000000000000000 + 4.44089209850062 × 10-15i

newton(f, _)
16.000000000000000 - 2.22044604925031 × 10-14i

```

Two things happened here. One, which is insignificant, is that in approximating the cube roots, *Sage* is putting in tiny imaginary parts. We can just ignore these. The more critical problem is that the numbers we're getting aren't moving closer to $x = 0$, but farther away. This time, starting closer to the root doesn't help. Newton always drives us out, not in:

```

def f(x):
    return(x.cube_root())

diff(x.cube_root(),x)

Traceback (click to the left for traceback)
...
AttributeError: 'sage.symbolic.expression.Expression' object has no

```


Just to show how subtle the problem caused by $x^{1/3}$ is, here's the plot of $x^{3/5}$ (whose shape looks qualitatively like that of $x^{1/3}$) along with some evidence that Newton converges to its root at $x = 0$. Even though Newton converges, though, the convergence is much slower than the convergence we got for all the nice functions above; so the funny twist in the graph is still giving Newton some trouble.

```
f(x)=x^(3/5)
```

```
newton(f, 1.0)
```

```
-0.6666666666666667
```

```
newton(f, _)
```

```
0.4444444444444445 - 1.11022302462516 × 10-16i
```

```
newton(f, _)
```

```
-0.296296296296297 + 7.40148683083438 × 10-17i
```

```
newton(f, _)
```

```
0.197530864197531 - 9.25185853854294 × 10-18i
```

```
def g(x): return(RR(x^3).nth_root(5))
```

```
plot(g, -4, 4)
```

