1 Questions to be handed in on Newton's Method:

Read about this material here: Newton's Method.

Begin by loading our package for plotting and the Roots package

```
using Gadfly
using Roots
```

1.0.1 Quick background Read the notes for more detail

Symbolic math is pretty nice. For so many problems it can easily do what is tedious work. However, for some questions, only numeric solutions are possible. For example, there is no general formula to solve a fifth order polynomial, the way there is a quadratic formula. Even an innocuous polynomial like $f(x) = x^5 - x - 1$ has no easy algebraic solution:

```
using SymPy
x = Sym("x")
solve(x^5 - x - 1)
```

We see that `SymPy` basically punts on this question.

Numeric solutions are available. As this is a polynomial, we could do:

```
using Polynomial
x = Poly([1, 0])        # 1x + 0
roots(x^5 - x - 1)
```

We see 5 roots – as expected from a fifth degree polynomial – with one real root (the one with `0.0im`) that is approximately 1.1673. Finding such a value usually requires some root-finding algorithm.

Newton's method is a root-finding algorithm like the bisection method discussed earlier. As an *algorithm* it starts with some *guess* for a *root* to an equation $f(x) = 0$. If this guess is called x_0, then the algorithm gives a *new (and improved)* guess x_1. It is expected that x_1 is a better guess, but may not be the best that can be. The algorithm is then repeated *again* to produce x_2. This is done until some guess x_n is as close as we can get or the algorithm fails for some reason. The *approximate root* is taken to be x_n.

What is the algorithm? It is simple. If we start with some x_i, then x_{i+1} is given by the intersection point of the x-axis of the tangent line of $f(x)$ at x_i. See a figure [here](http://mth229.github.io/newton.html#_basic_i
Mathematically then we can equate our two means to compute the slope of a tangent line:

$$f'(x_i) = \frac{f(x_i) - 0}{x_i - x_{i+1}}$$

Or, solving for x_{i+1}:

$$x_{i+1} = x_i - f(x_i)/f'(x_i)$$

Let's see this algorithm for `f(x) = x^32x5`, a function that Newton considered. He was looking for a solution near 2, so let's start there:

```
x = 2
f(x) = x^3 - 2x -5
fp(x) = 3x^2 - 2        # done by hand
```

We don't need to track the index (x_0, x_1, ...) as when we write the following expression, the next value is just assigned to `x` using the *current* value of `x` when computed:

```
x = x - f(x) / fp(x)
x, f(x)             # display both the new guess, x,  and the value f(x)
```

The value of 2.1 is a better guess, but not near the actual answer. We simply repeat to (hopefully) get a better guess:

```
x = x - f(x) / fp(x)
x, f(x)
```

Here are a few repeats:

```
x = x - f(x) / fp(x)
x, f(x)

x = x - f(x) / fp(x)
x, f(x)
```

The value of `f(x)` is now *basically* 0, and any further updates to `x` do not change its value. We see that the algorithm has converged to an answer, `x`, and the fact that it is a zero is confirmed by the value of `f(x)`.

Repeating steps in `IJulia` can be a bit of a chore, so here we define a *macro* to repeat some expression 5 times and then show how to use it. (A macro does not evaluate its expression immediately, unlike functions.)

```
macro take5(body) quote [$(esc(body)) for _ in 1:5] end end # take5 macro
```

Macros are prefaced with a `@` in their name and are called without parentheses:

```
@take5     x = x - f(x) / fp(x)
```

We now show how we can get the above in a more efficient manner:

```
f(x) = x^3 - 2x -5
x = 2               # initial guess
fp(x) = 3x^2 - 2

@take5  x = x - f(x)/fp(x)
(x, f(x))           # has converged, as f(x) is basically 0
```

1.0.2 Questions

- Apply Newton's Method to the function $f(x) = \sin(x)$ with an initial guess 3. (This was historically used to compute many digits of π efficiently.) What is the answer after 5 iterations? What is the value of `sin` at the answer?

- Use Newton's method to find a zero for the function $f(x) = x^5 - x - 1$. Start at $x = 1.6$. What is the approximate root after 5 iterations? What is the value of $f(x)$ for your answer? If you do one or two more iterations, will your guess be better?

- Use Newton's method to find a zero of the function $f(x) = \cos(x) - x$. Make a graph to identify an initial guess.

1.0.3 Using `D` for the derivative

If the function `f(x)` allows it, the `D` operator from the `Roots` package can simplify the algorithm, as the derivative need not be computed by hand. In this case, the algorithm in `julia` becomes `x = x - f(x)/D(f)(x)`.

- Use Newton's method to find an intersection point of $f(x) = e^{-x^2}$ and $g(x) = x$. (Look at $h(x) = f(x) - g(x) = 0$.) Start with a guess of 0.

- Use Newton's method to find *both* positive intersection point of $f(x) = e^x$ and $g(x) = 2x^2$. Make a graph to identify good initial guesses.

1.0.4 using `newton` and `fzero` from the `Roots` package

The `newton` function in the `Roots` package will compute newton's method. For example:

```
f(x) = sin(x)
fp(x) = cos(X)
x = 3
newton(f, fp, x)
```

However, the `fzero` function – that we have seen before – will use a derivative-free algorithm, similar to Newton's method to find a zero. So, the above could be achieved with:

```
fzero(sin, 3)
```

(That is right, `fzero` can be used two different ways – at least. Above it is called with an initial guess. Previously, we called it with a bracketing interval, as in `fzero(sin, [3,4])`. If you specify a bracketing interval, `fzero` will use an algorithm guaranteed to converge. If you just specify an initial guess, the convergence may not happen.)

- find a zero of $f(x) = x \cdot (2 + \ln(x))$ starting at 1. What is your answer? How small is the function for this value?

- Use `fzero` to find all zeros of the function $f(x) = 2\sin(x) - \cos(2x)$ in $[0, 2\pi]$. (Graph first to see approximate answers.)

- The default algorithm for `fzero` is to use Steffensen's method, which replaces $f'(x)$ with an approximation: $(f(x + f(x)) - f(x))/f(x)$. It looks complicated, but is just the secant line approximation with a small `h` given by `f(x)`, so the initial guess is quite important.

  For the function

$$f(x) = 5\frac{\sin(x)}{\cos^2(x)} - 7\frac{\cos(x)}{\sin^2(x)}$$

  Compare the approximate derivative above with ($h = f(x)$) to that given by `D(f)(x)` when `x=pi/4`. Are they close?

- The above question came from trying to find when the derivative of $f(x) = 5/\cos(x)+7/\sin(x)$ is 0 in the interval $(0, \pi/2)$. The basic `fzero` call for `D(f)` fails if `x=pi/4`. (Try it: `fzero(D(f), pi/4)`.) One can try closer guesses, *or* a different algorithm. The `fzero` function has an argument `order` which can be either 2, 5, 8 or 16. Try it with `order=8` and see if it converges. If it does, what is the root? (An order 8 algorithm should converge faster mathematically than the default order 2 algorithm, but may not once implemented on the computer.)

1.0.5 When Newton's method fails

Newton's method can fail due to various cases:

1) the initial guess is not close to the zero
2) the derivative, $|f'(x)|$ is too small
3) the second derivative $|f''(x)|$ is too big

- Let $f(x) = x^5 - x - 1$. Try Newton's method with an initial guess of $x_0 = 0$. Why does this fail? (You can look graphically. Otherwise, you could look at the output of `newton` with this extra argument: `newton(f, fp, x0, verbose=true)`.

- Let `f(x) = abs(x)^(1/3)`. Starting at `x=1`, Newton's method will fail to converge. What happens? Are any of the above 3 reason's to blame?

1.0.6 Quadratic convergence

When Newton's method converges to a *simple zero* it is said to have *quadratic convergence*. A simple zero is one with multiplicity 1 and quadratic convergence says basically that the error at the $i + 1$st step is like the error for ith step squared. In particular, if the error is like 10^{-3} on one step, it will be like 10^{-6}, then 10^{-12} then 10^{-24} on subsequent steps. (Which is typically beyond the limit of a floating point approximation.) This is why one can *usually* take just 5 steps to get to an answer.

Not so for multiple roots.

- For the function `f(x) = (8x*exp(-x^2) -2x - 3)^8`, starting with `x=-2.0` Newton's method will converge, but it will take many steps to get to an answer that has $f(x)$ around 10^{-16}. How many? Roughly how many iterations do you need? (A single call of `@take5 x = x-f(x)/D(f)(x)` gives an answer with `f(x) = 0.00028` only.)

- Repeat the above with `f(x) = 8x*exp(-x^2) -2x - 3` and again, starting with `x=-2.0`. Roughly how many iterations are needed now?