Packages in Julia

- Downloading Packages
- A Word of Caution
- Sawtooth, Revisited

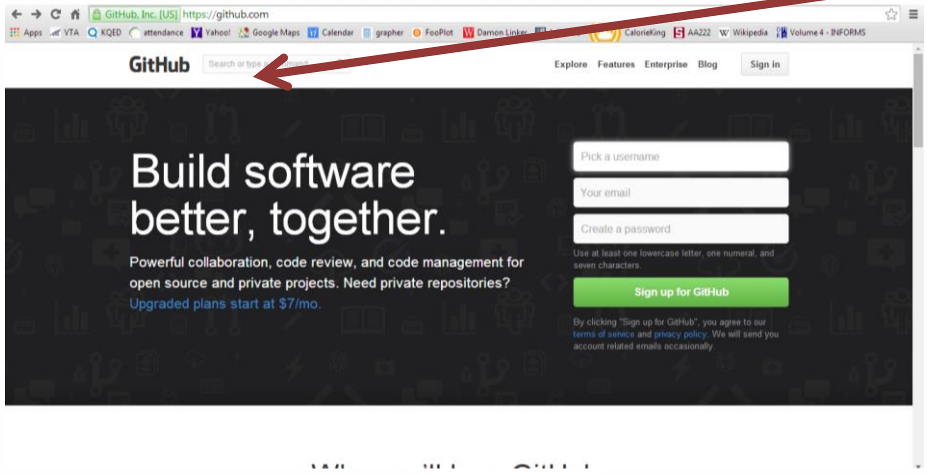
Downloading Packages

Because Julia is an open-source language, there are a ton of packages available online that enable such things as graphics capabilities, specialized programs or functionality, and yes… even calculus.

The main repository for most of these packages is GitHub, https://github.com.

Downloading Packages

The main screen of GitHub will make you think you have to sign up. You don't. The search window is here.



Build software better, together.

Powerful collaboration, code review, and code management for open source and private projects. Need private repositories? Upgraded plans start at $7/mo.

Use at least one lowercase letter, one numeral, and seven characters.

By clicking "Sign up for GitHub", you agree to our terms of service and privacy policy. We will send you account related emails occasionally.

Everything's free. GitHub is a great place to research available packages, though if you need something specific combine GitHub with a google search to narrow things down. There is a lot of stuff on this site, not just for Julia but for all languages.

Downloading Packages

If you already know what package you want to download, you can instruct Julia to take care of it for you. It's easiest to type it into the Julia console (the command-screen of the main program), but it should work in most IDEs as well.

To see how it works, type this in:

```
Pkg.add("Calculus")
```

…then wait.

Downloading Packages

After a while Julia will let you know it is done. Now you have the Calculus package installed.

Next, try this:

```
using Calculus
```

(this loads up the Calculus package, must be done every time you open a new session)

```
f(x) = x^2                f(generic…)
f'(5)                     9.99999…
f(x) = cos(x)             f(generic…)
f'(pi/2)                  -0.99999…
f(x) = 4x^3               f(generic…)
f'(3)                     108.0000…
f''(3)                    72.000069…
f'''(3)                   22.37032…
```

(this is supposed to be 24)

But WAIT!

The Calculus package can do **symbolic differentiation**. It will give you an exact answer.

Try this:

```
differentiate("4x^3", :x):(4*(3*x^2))
```

Whoa, that's awesome! You just got an exact derivative.

The : (colon) means that Julia is treating `(4*(3*x^2))` as a symbol that you can evaluate. Try:

```
x = 3
eval(differentiate("4x^3", :x))     108
```

But WAIT!

There's more! Try a second derivative:

```
eval(differentiate(differentiate("4x^3",
     :x)))                        72
```

…And the third derivative, which went terribly wrong when we tried `f'''(x)`:

```
eval(differentiate(differentiate
    (differentiate("4x^3", :x))))   24
```

(this is supposed to be 24)

But WAIT!

Why would we ever use `f'` when `eval(derivative(` gives an exact answer?

- Symbolic differentiation is more accurate than numerical differentiation, but it is also **much slower**. This makes a difference in programs with lots of loops.
- Also, `derivative(` requires a string in quotes as its first argument, and you can't pass in a variable. This is extremely limiting.

A Word of Caution

More about the functionality of the Calculus package in a moment. First, a caution.

You just downloaded onto your computer, from the internet, a segment of code that communicates directly with the brains of your computer.

Thank you for trusting me.

A Word of Caution

Although it is very simple and looks so innocent, it is never a good idea to randomly download stuff from the internet (or app store, or whatever) without doing proper research first.

This particular program, "Calculus.jl", was written by John Myles White. (That information is available on GitHub, or by googling "julia calculus".)

Take a moment to google John Myles White. Find at least 4 pieces of evidence that he's a good guy.

You should *always* do this type of search.

always.

always!

Now, Back to Calculus

There are several other functions in the Calculus package, including:

```
integrate(f, a, b)
```

Full documentation is available on the GitHub page for Calculus, at github.com/johnmyleswhite/calculus.jl.

The main functions we'll be using are derivatives.

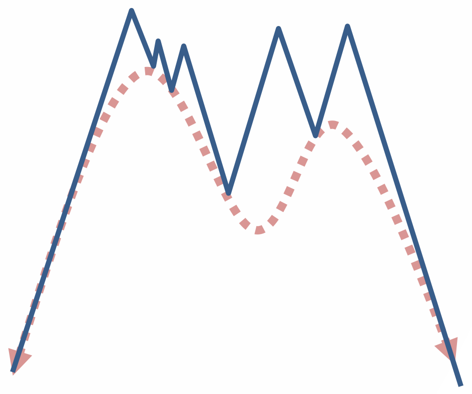
Practice Problem 1

a. Find the derivative of $y = 3x^2 + 2\sin x$ at $x = 4$ using your derivative program, using the Calculus package, and by hand. Compare the answers.
b. Find the integral of $y = e^x + 3x - 2$ from 3 to 5 using your integral program, using the Calculus package, and by hand. Compare the answers.

Sawtooth, Revisited

Another application for calculus, which we used in the second unit, involves the Sawtooth Method.

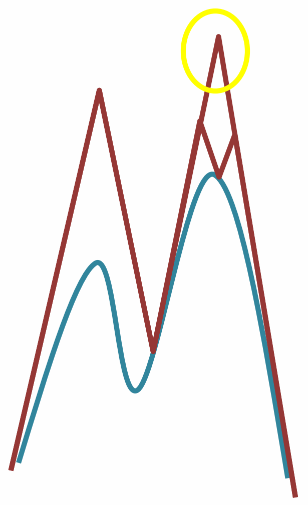
This method attempted to find a global maximum by drawing a series of lines of slope ±m from various points on the function, with m being a slope such that the function never got steeper than that slope.



Sawtooth, Revisited

The basic procedure was this:

1. Draw lines of slope ±m from the function at the midpoint and endpoints of the interval.
2. Find the intersection points of the lines.
3. Find the highest y-value among the intersection points.
4. Draw lines of slope ±m from the function at the y-value corresponding to that x-value.
5. Repeat 2 – 5.



Sawtooth, Revisited

What we didn't do, at that time, was calculate the limiting slope, m. The method will not work if the slope of the function is steeper than the value of m (or –m).

Without calculus, you can guess the value of m (as long as you guess too high) or use a graph to estimate it; but with calculus you can find the limiting slope using derivatives.

Practice Problem 2

a. Find, by hand, the steepest slope attained by the function $f(x) = 5x^3 + 2x^2$ on the interval $[-2, 5]$. This slope may be either negative or positive.

b. Repeat for $f(x) = 3x^4 - 2x^2$ on $[-0.5, 0.5]$

Sawtooth, Revisited

As you (hopefully) recalled on problem 2, the maximum and minimum values of a function must occur either at the endpoints, or where its derivative equals 0.

If your function is a derivative, you are looking for where the second derivative equals 0.

This assumes that you know the equation of the second derivative, which is fine if you're doing it by hand; but a computer does not derive by hand.

Sawtooth, Revisited

In pairs or groups, discuss how you could find the maximum or minimum slope values with the following tools:

1. The original equation
2. The ability to find f'(a) values of the derivative
3. The ability to find f''(a) values of the second derivative
4. Any existing programs you have from earlier units

Remember, you may not use the *equation* of the derivative or second derivative!

Practice Problem 3

Write a program that, given a function and endpoints, returns the limiting value of the slope for use in a sawtooth program.

If you plan to use the Calculus package, you will need to type the line

```
using Calculus
```

before the opening function line.

Test and document and save your code!