1 Getting started with Julia

Julia is a freely available, open-source programming language aimed at technical computing.

As it is open source, indeed with a liberal MIT license, it can be installed for free on many types of computers (though not phones or tablets).

lauch binder

We recommend taking advantage of Binder, which provides a web-based interface to `Julia` built around `Jupyter`, a wildly succesful platform for interacting with different open-source software programs. Clicking the launch button above will open a web page which provides a blank notebook, save for a package used by these notes.

COCALC offers free access to Julia through the web. The free version can be slow due to limits on the resources used, but the monthly price for paid version is modest and the performance is quite reasonable.

1.0.1 Installing Julia locally

Binder is great and convenient, but it may not be the best option. Fortunately, installing `Julia` is a not-so-difficult option.

Binaries of `Julia` are provided at julialang.org. Julia has an official released version and a developmental version. Unless there is a compelling reason, the latest released version should be downloaded and installed for use.

The base `Julia` provides a *command-line interface*, or REPL (read-evaluate-parse).

1.0.2 Basic interactive usage

Once installed, `Julia` can be started by clicking on an icon or typing `julia` at the command line. Either will open a *command line interface* for a user to interact with a `Julia` process. The basic workflow is easy: commands are typed then sent to a `Julia` process when the "return" key is pressed for a complete expression. Then the output is displayed.

A command is typed following the *prompt*. An example might be `2 + 2`. To send the command to the `Julia` interpreter the "return" key is pressed. A complete expression or expressions will then be parsed and evaluated (executed). If the expression is not complete, `julia`'s prompt will still accept input to complete the expression. Type `2 +` to see. (The expression `2 +` is not complete, as the infix operator `+` expects two arguments, one on its left and one on its right.)

```
_
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.0.0 (2018-08-08)
 _/ |\__'_|_|_|\__'_|  |  Official https://julialang.org/ release
|__/                   |

julia> 2 + 2
```

```
4
```

Above, `julia>` is the prompt. These notes will not include the prompt, so that copying-and-pasting can be more easily used. Input and output cells display similarly, though with differences in coloring. For example:

```
2 + 2
```

4

Other interfaces to `Julia` are described briefly in Julia interfaces. The notebook interface provided through `IJulia` most closely matches the style of the notes.

1.1 Add-on packages

`Julia` has well over a 1000 external, add-on packages that enhance the offerings of base `Julia`. We refer to one, `CalculusWithJulia`, that is designed to accompany these notes. This package installs several other packages that provide the needed functionality. The package (and its dependencies) can be installed through:

```
using Pkg
Pkg.add("CalculusWithJulia")
```

(Or the one liner `] add CalculusWithJulia`. Some additional details on packages is provided here.)

Installation only needs to be done once, but to use a package it must be loaded into each new session. This can be done with this command:

```
using CalculusWithJulia
```

1.2 The basics of working with IJulia

The **very** basics of the Jupyter notebook interface provided by `IJulia` are covered here.

An `IJulia` notebook is made up of cells. Within a cell a collection of commands may be typed (one or more).

When a cell is executed (by the triangle icon or under the Cell menu) the contents of the cell are evaluated by the `Julia` kernel and any output is displayed below the cell. Typically this is just the output of the last command.

```
2 + 2
3 + 3
```

6

If the last commands are separated by commas, then a "tuple" will be formed and each output will be displayed, separated by commas.

```
2 + 2, 3 + 3
```

```
(4, 6)
```

Comments can be made in a cell. Anything after a `#` will be ignored.

```
2 + 2  # this is a comment, you can ignore me...
```

4

Graphics are provided by external packages. There is no built-in graphing. We use the `Plots` package and its default backend. The `Plots` package provides a common interface to several different backends, so this choice is easily changed. The `GR` and `plotly` backends are used in these notes, when possible; the `PyPlot` backend is used for some surface plots.

```
using CalculusWithJulia
using Plots
```

With that in hand, to make a graph of a function over a range, we follow this pattern:

```
plot(sin, 0, 2pi)
```

```
Plot{Plots.PlotlyBackend() n=1}
```

A few things:

- Cells are numbered in the order they were evaluated.
- This order need not be from top to bottom of the notebook.
- The evaluation of a cell happens within the state of the workspace, which depends on what was evaluated earlier.
- The workspace can be cleared by the "Restart" menu item under "Kernel". After restarting the "Run All" menu item under "Cell" can be used to re-run all the commands in the notebook - from top to bottom. "Run all" will also reload any packages.