Optimization Module Preassignment

Contact: jackdunn@mit.edu

Purpose

This document aims to get you set up with

1. **Julia** - the programming language.
2. **IJulia** - a popular environment for writing and sharing Julia code.
3. **Gurobi** – a powerful commercial optimization solver
4. **JuMP** - a package for Julia that lets you express optimization models (together with a number of other Julia packages which we will use).

Julia (https://julialang.org/) is a relatively new programming language that is aimed at technical computing. We'll be using Julia through the **IJulia** notebook interface, which lets you mix code, documentation, mathematics, graphics, etc.

JuMP (https://github.com/JuliaOpt/JuMP.jl) is a modeling language for optimization problems, like AMPL, GAMS, and YALMIP.

Note: some of the formatting in this document may cause issues if you try to copy/paste code. If you get any errors, please try entering the commands manually rather than pasting them.

1. Install Julia

You can download Julia from the official Julia downloads page: http://julialang.org/downloads/. You should download the "**Current Release (v0.6.0)".** (If you have an older version of Julia that is 0.5 that should be fine, but it is best to upgrade to the current version. If you have Julia 0.4 you will need to upgrade). Accept the default setup options.

- If you are on Windows you should probably choose the 64-bit version. If you have a very old Windows computer (>4 years old) there is a chance you will have a 32-bit version of Windows. To check, go to the "System Information" page (start typing "system information" in the Start menu and it should show up).
- If you have an old Mac and haven't upgraded to OS X 10.7 or higher, you won't be able to run Julia. You can check this by clicking the Apple logo in the top-left of your screen and clicking "About This Mac"
- Up-to-date versions of Julia are available for some distributions. Please read instructions on the download page. If you are on Ubuntu, **do not** install the version of Julia that is in the main repository - it is out-of-date.

2. Install IJulia

IJulia is a browser-based environment for writing and running Julia code. It lets you combine code, text, graphics, and equations all in one place. You may have heard of IPython, which is something similar for the Python programming language.
Note: If you are on the MIT campus, **do not use the MIT Guest wifi network**. Please connect to “MIT” or “MIT Secure” - downloads can have issues on MIT Guest.

Windows

- Open Julia by double-clicking the icon or from the Start menu.
- A window with a `julia>` prompt will appear.
- Type `Pkg.add(“IJulia”)` and wait for the installation to finish.

Apple

- Open Julia from your Applications.
- A window with a `julia>` prompt will appear.
- Type `Pkg.add(“IJulia”)` and wait for the installation to finish.

Linux

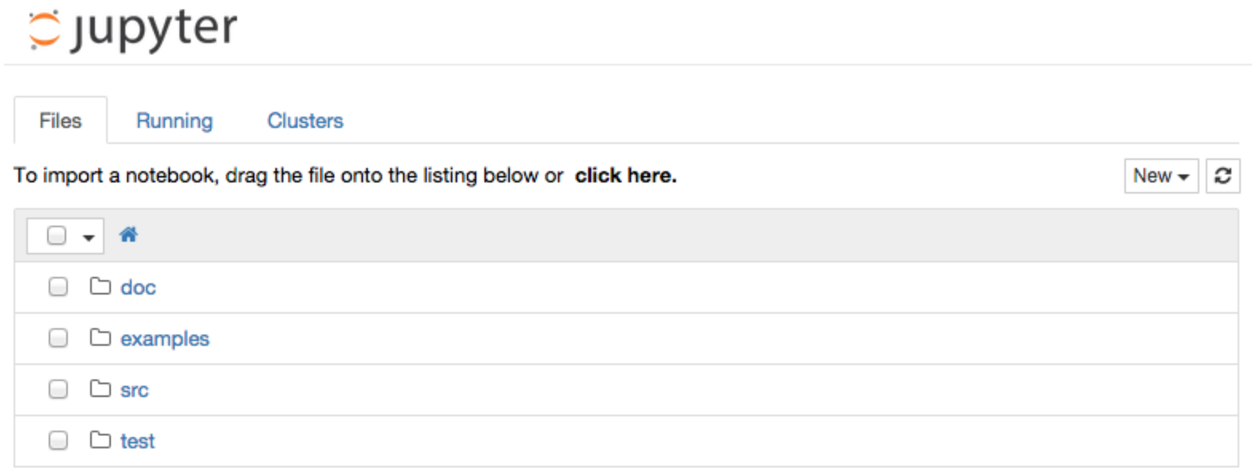
- Open Julia and run `Pkg.add(“IJulia”)` at the `julia>` prompt.

After installing IJulia

The easiest way to check if your installation was successful is to open Julia and run

```
using IJulia
notebook()
```

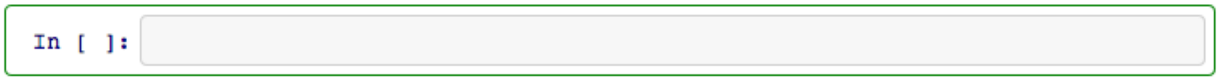
A web browser window should open, with something that looks like:



To import a notebook, drag the file onto the listing below or **click here.**

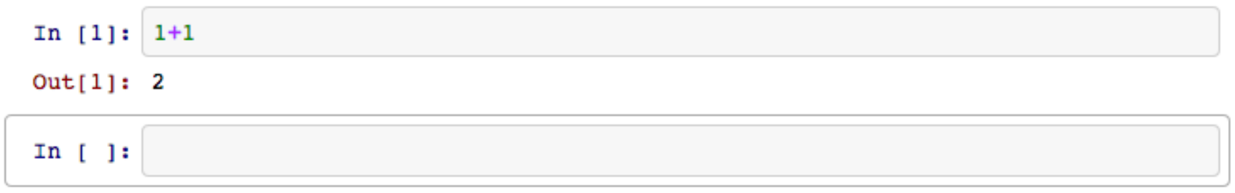
(the list of files will be different).

Click “New” and select “Julia 0.6.0” - a new tab will open. In the “In [ ]” box:



```
In [ ]:
```

type 1+1 and press shift and enter/return at the same time. The result should appear like



```
In [1]: 1+1
Out[1]: 2
In [ ]:
```

3. Install Gurobi

Gurobi is a commercial mixed integer linear and mixed integer second-order conic solver which we will be using during the class. If you don't have Gurobi installed already, please follow their installation guide (http://www.gurobi.com/documentation/) for your platform.

The main steps are:

- Go to gurobi.com and sign up for an account
- Get an academic license from the website (section 2.1 of the quick-start guide)
- Download and install the Gurobi optimizer (section 3 of the quick-start guide)
- Activate your academic license (section 4.1 of the quick-start guide)
  - you need to do the activation step while connected to the MIT network. If you are off-campus, you can use the MIT VPN (https://ist.mit.edu/vpn) to connect to the network and then activate (get in touch if you have trouble with this).
- Test your license (section 4.6 of the quick-start guide)

4. Add Julia packages

Now we will install the main Julia packages that we will use during the course.

- Open an IJulia notebook like described above.
- In the first cell enter the following commands:

```
Pkg.add("JuMP")
Pkg.add("Gurobi")
Pkg.add("Cbc")
Pkg.add("RDatasets")
Pkg.add("DataFrames")
Pkg.add("DataFramesMeta")
Pkg.add("Plots")
Pkg.add("StatPlots")
Pkg.add("PyPlot")
```

Then press shift-enter to run the cell.

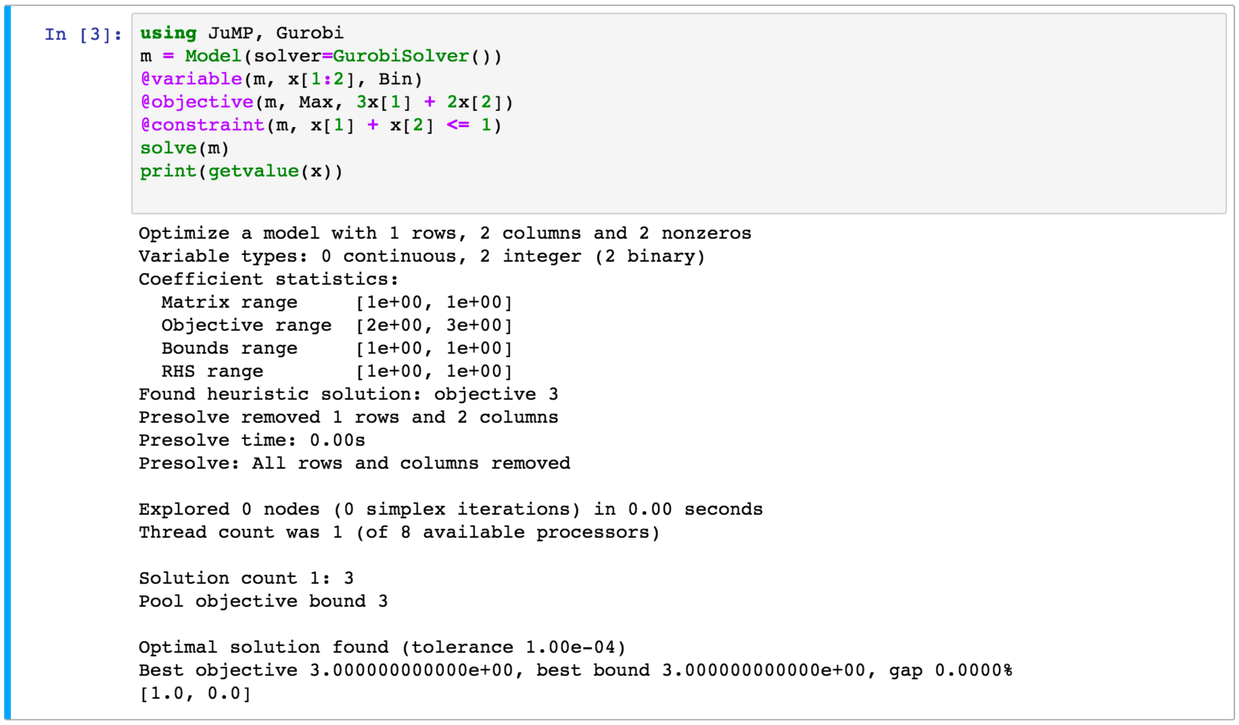
- This will install the JuMP package, and the interface to the Gurobi solver for linear and mixed-integer programming. It will also install some other packages we'll be using during the class

5. Testing JuMP

We can test out if everything's set up correctly by running the following code in a notebook:

```
using JuMP, Gurobi
m = Model(solver=GurobiSolver())
@variable(m, x[1:2], Bin)
@objective(m, Max, 3x[1] + 2x[2])
@constraint(m, x[1] + x[2] <= 1)
solve(m)
print(getvalue(x))
```

Remember to press shift-enter to execute it.
You should see something like



```
In [3]: using JuMP, Gurobi
        m = Model(solver=GurobiSolver())
        @variable(m, x[1:2], Bin)
        @objective(m, Max, 3x[1] + 2x[2])
        @constraint(m, x[1] + x[2] <= 1)
        solve(m)
        print(getvalue(x))
```

```
Optimize a model with 1 rows, 2 columns and 2 nonzeros
Variable types: 0 continuous, 2 integer (2 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [2e+00, 3e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Found heuristic solution: objective 3
Presolve removed 1 rows and 2 columns
Presolve time: 0.00s
Presolve: All rows and columns removed

Explored 0 nodes (0 simplex iterations) in 0.00 seconds
Thread count was 1 (of 8 available processors)

Solution count 1: 3
Pool objective bound 3

Optimal solution found (tolerance 1.00e-04)
Best objective 3.000000000000e+00, best bound 3.000000000000e+00, gap 0.0000%
[1.0, 0.0]
```