This chapter will introduce what the Julia Language is and explain why I love it. More importantly, this chapter will teach you how to obtain Julia and install it in your machine. Well, at this moment, the most challenging task for using Julia in computing would probably be installing the language and other libraries and programs correctly in your own machine. I will go over every step with fine details with screenshots for both Windows and Mac machines. I assumed that Linux users can handle the installation process well enough without much help from this book by reading online manuals and googling. Perhaps the Mac section could be useful to Linux users.

1.1 What is Julia and Why Julia?

The Julia Language is a very young language. As of March 5, 2016, the latest stable version is 0.4. The primary target of Julia is technical computing. It is developed for making technical computing more fun and more efficient. There are many good things about the Julia Language from the perspective of computer scientists and software engineers; you can read about the language at the official website\textsuperscript{1}.

Here is a quote from the creators of Julia from their first official blog article “Why

\textsuperscript{1}http://julialang.org
1.1. What is Julia and Why Julia?

We Created Julia:\(^2\):

“We want a language that’s open source, with a liberal license. We want the speed of C with the dynamism of Ruby. We want a language that’s homoiconic, with true macros like Lisp, but with obvious, familiar mathematical notation like Matlab. We want something as usable for general programming as Python, as easy for statistics as R, as natural for string processing as Perl, as powerful for linear algebra as Matlab, as good at gluing programs together as the shell. Something that is dirt simple to learn, yet keeps the most serious hackers happy. We want it interactive and we want it compiled.

(Did we mention it should be as fast as C?)”

So this is how Julia is created, to serve all above greedy wishes.

Let me tell you my story. I used to be a Java developer for a few years before I joined a graduate school. My first computer codes for homework assignments and course projects were naturally written in Java; even before then, I used C for my homework assignments for computing when I was an undergraduate student. Later, in the graduate school, I started using MATLAB, mainly because my fellow graduate students in the lab were using MATLAB. I needed to learn from them, so I used MATLAB.

I liked MATLAB. Unlike in Java and C, I don’t need to declare every single variable before I use it; I just use it in MATLAB. Arrays are not just arrays in the computer memory; arrays in MATLAB are just like vectors and matrices. Plotting computation results is easy. For modeling optimization problems, I used GAMS and connected with solvers like CPLEX. While the MATLAB-GAMS-CPLEX chain suited my purpose well, I wasn’t that happy with the syntax of GAMS—I couldn’t fully understand—and the slow speed of the interface between GAMS and MATLAB. While CPLEX provides complete connectivities with C, Java, and Python, it was very basic with MATLAB.

When I finished with my graduate degree, I seriously considered Python. It was—and still is—a very popular choice for many computational scientists. CPLEX also has a better support for Python than MATLAB. Unlike MATLAB, Python is free and open source software. However, I didn’t go with Python and decided to

\(^2\)http://julialang.org/blog/2012/02/why-we-created-julia
stick with MATLAB. I personally don’t like 0 being the first index of arrays in C and Java. In Python, it is also 0. In MATLAB, it is 1. For example, if we have a vector like:

\[
v = \begin{bmatrix} 1 \\ 0 \\ 3 \\ -1 \end{bmatrix}
\]

it may be written in MATLAB as:

```matlab
v = [1; 0; 3; -1]
```

The first element of this vector should be accessible by \(v(1)\), not \(v(0)\). The \(i\)-th element must be \(v(i)\), not \(v(i-1)\). So I stayed with MATLAB.

Later in 2012, the Julia Language was introduced and it looked attractive to me, since at least the array index begins with 1. After some investigations, I didn’t move to Julia at that time. It was ugly in supporting optimization modeling and solvers. I kept using MATLAB.

In 2014, I came across several blog articles and tweets talking about Julia again. I gave it one more look. Then I found a package for modeling optimization problems in Julia, called JuMP—Julia for Mathematical Programming. After spending a few hours, I felt in love with JuMP and decided to go with Julia, well more with JuMP. Here is a part of my code for solving a network optimization problem:

```julia
m = Model()
@defVar(m, 0<= x[links] <=1)

@setObjective(m, Min, sum{c[(i,j)] * x[(i,j)], (i,j) in links} )

for i=1:no_node
    @addConstraint(m, sum{x[(ii,j)], (ii,j) in links; ii==i } - sum{x[(j,ii)], (j,ii) in links; ii==i } == b[i])
end

solve(m)
```
This is indeed a direct “translation” of the following mathematical language:

\[
\min \sum_{(i,j) \in A} c_{ij}x_{ij}
\]

subject to

\[
\sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = b_i \quad \forall i \in \mathcal{N}
\]

\[
0 \leq x_{ij} \leq 1 \quad \forall (i,j) \in A
\]

I think it is a very obvious translation. It is quite beautiful, isn’t it?

CPLEX and its competitor Gurobi are also very smoothly connected with Julia via JuMP. Why should I hesitate? After two years of using Julia, I still love it—I even wrote a book, which you are reading now, about Julia!

1.2 Julia in the Cloud: JuliaBox

You can enjoy many features of the Julia Language on the web at \texttt{http://juliabox.org}. Log in with your Google account and create a “New Notebook”.

First, install the \texttt{Clp.jl} and \texttt{JuMP.jl} packages.

\begin{verbatim}
Pkg.add("Clp")
Pkg.add("JuMP")
\end{verbatim}

and press \texttt{Shift+Enter} or click the “play” button to run your code. \texttt{Clp} provides an open source LP solver, and \texttt{JuMP} provides a nice modeling interface.

Copy this code to your screen:

\begin{verbatim}
using JuMP
m = Model()
@defVar(m, 0 <= x <= 40)
@defVar(m, y <= 0)
@defVar(m, z <= 0)
@setObjective(m, Max, x + y + z)
\end{verbatim}
Chapter 1. Introduction and Installation

```julia
@addConstraint(m, const1, -x + y + z <= 20)
@addConstraint(m, const2, x + 3y + z <= 30)
solve(m)
println("Optimal Solutions: ")
println("x = ", getValue(x))
println("y = ", getValue(y))
println("z = ", getValue(z))
```

and press `Shift+Enter` or click the “play” button to run your code. The result will look like:

![Jupyter Notebook output](image)

If you want to use commercial solvers CPLEX or Gurobi, you have to install Julia in your computer. Please follow the instruction in the next section.

### 1.3 Installing Julia

Graduate students and researchers are strongly recommended to install Julia in their local computers. In this guide, we will first install the Gurobi optimizer and
1.3. Installing Julia

then Julia.

1.3.1 Installing Gurobi

First, install Gurobi Optimizer. Gurobi is a commercial optimization solver package for solving LP, MILP, QP, MIQP, etc, and it is free for students, teachers, professors, or anyone else related to educational organizations.

1. **Download Gurobi Optimizer** and install in your computer. (You will need to register as an academic user, or purchase a license.)

2. **Request a free academic license** and follow their instruction to activate it.

(Note to Windows users: The version you select, either 32-bit or 64-bit, needs to be consistent. That is, if you choose 64-bit Gurobi Optimizer, you will need to install 64-bit Julia in the next step. After installation, you must reboot your computer.)

If you are not eligible for free licenses for Gurobi, please go ahead and install Julia. There are open-source solvers available.

The following two sections provide steps with screenshots to install the Julia Language, the JuMP package, and the Gurobi package. Windows users go to section 1.3.2, and Mac users go to Section 1.3.3.

1.3.2 Installing Julia in Windows

- **Step 1.** Download Julia from the official website. (Select an appropriate version: 32-bit or 64-bit, same as your Gurobi Optimizer version.)

<table>
<thead>
<tr>
<th>Julia (command line version)</th>
<th>32-bit</th>
<th>64-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows Self-Extracting Archive (.exe)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mac OS X Package (.dmg)</td>
<td>10.7+ 64-bit</td>
<td></td>
</tr>
<tr>
<td>Ubuntu packages (.deb)</td>
<td>32/64-bit</td>
<td></td>
</tr>
<tr>
<td>Fedora/RHEL/CentOS/SLE packages (.rpm)</td>
<td></td>
<td>32/64-bit</td>
</tr>
<tr>
<td>Generic Linux binaries</td>
<td>32-bit (GPG)</td>
<td>64-bit (GPG)</td>
</tr>
<tr>
<td>Source</td>
<td>Tarball (GPG)</td>
<td>Full Tarball (GPG)</td>
</tr>
</tbody>
</table>

3[http://user.gurobi.com/download/gurobi-optimizer](http://user.gurobi.com/download/gurobi-optimizer)
4[http://user.gurobi.com/download/licenses/free-academic](http://user.gurobi.com/download/licenses/free-academic)
• **Step 2.** Install Julia in \texttt{C:\julia}.

![Julia Installation Screen](image)

• **Step 3.** Open a Command Prompt and enter the following command:

\texttt{setx PATH "\%PATH\%;C:\julia\bin"}

![Command Prompt Window](image)

If you don’t know how to open a Command Prompt, see [this link]^{6}(choose your...)

1.3. Installing Julia

Windows version, and see “How do I get a command prompt”).

- **Step 4.** Open a **NEW** command prompt and type

```
echo %PATH%
```

The output must include `C:\julia\bin` in the end. If not, you must have something wrong.

- **Step 5.** Run `julia`.

You have successfully installed the Julia Language on your Windows computer. Now its time for installing additional packages for mathematical programming.
Chapter 1. Introduction and Installation

- **Step 6.** If you have not installed Gurobi yet in your system, please install it first. On your julia prompt, type

  ```
Pkg.add("JuMP")
Pkg.add("Gurobi")
  ```

  (If you are ineligible to use a free license of Gurobi, use the Cbc solver: `Pkg.add("Cbc")`)

- **Step 7.** Open Notepad (or any other text editor, for example Atom\(^7\)) and type the following, and save the file as `script.jl` in some folder of your choice.

\(^7\)http://atom.io
1.3. Installing Julia

```julia
using JuMP, Gurobi
m = Model(solver=GurobiSolver())

@defVar(m, 0 <= x <= 2 )
@defVar(m, 0 <= y <= 30 )

@setObjective(m, Max, 5x + 3*y )
@addConstraint(m, 1x + 5y <= 3.0 )

print(m)

status = solve(m)

println("Objective value: ", getObjectiveValue(m))
println("x = ", getValue(x))
println("y = ", getValue(y))
```

If you are ineligible to use a free license of Gurobi, replace the first two lines by

```julia
using JuMP, Cbc
m = Model(solver=CbcSolver())
```

- **Step 8.** Press and hold your Shift Key and right-click the folder name, and choose “Open command window here.”
• **Step 9.** Type `dir` to see your script file `script.jl`.
1.3. Installing Julia

If you see a filename such as `script.jl.txt`, use the following command to rename:

```
ren script.jl.txt script.jl
```

- **Step 10.** Type `julia script.jl` to run your Julia script.
Chapter 1. Introduction and Installation

After a few seconds, the result of your julia script will be printed. Done.

Please proceed to Section 1.3.4.

1.3.3 Installing Julia in Mac OS X

- **Step 1.** Download Julia from the official website.⁸ (Select an appropriate OS X version.)

![Image of command prompt output]

<table>
<thead>
<tr>
<th>Julia (command line version)</th>
<th>32-bit</th>
<th>64-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows Self-Extracting Archive (.exe)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mac OS X Package (.dmg)</td>
<td>10.7+ 64-bit</td>
<td></td>
</tr>
<tr>
<td>Ubuntu packages (.deb)</td>
<td>32/64-bit</td>
<td></td>
</tr>
<tr>
<td>Fedora/RHEL/CentOS/SL packages (.rpm)</td>
<td>32/64-bit</td>
<td></td>
</tr>
<tr>
<td>Generic Linux binaries</td>
<td>32-bit (GPG)</td>
<td>64-bit (GPG)</td>
</tr>
<tr>
<td>Source</td>
<td>Tarball (GPG)</td>
<td>Full Tarball (GPG)</td>
</tr>
</tbody>
</table>

⁸http://julialang.org/downloads/
1.3. Installing Julia

- **Step 2.** In your download folder, double-click the downloaded .dmg file to mount it. Drag “Julia-0.x.x.app” file to the “Applications” folder.

- **Step 3.** Open “Terminal.app” from your Applications folder. (If you don’t know how to open it, see this video.⁹ It is convenience to place “Terminal.app” in your dock.

⁹https://www.youtube.com/watch?v=zw7Nd67_aFw
Step 4. In your terminal, enter the following commands:

```bash
touch ~/.bash_profile
open -e ~/.bash_profile
```

It will open a TextEdit window, enter the following line somewhere:
1.3. Installing Julia

```bash
export PATH=/Applications/Julia-0.4.2.app/Contents/Resources/julia/bin/:
```

Change 0.4.2 to reflect your Julia version.

Save the file and close the TextEdit window. In your terminal, enter the following command:

```bash
source ~/.bash_profile
```

- **Step 5.** In your terminal, enter `julia`. 
• **Step 6.** If you have not installed **Gurobi** in your system yet, go back and install it first. Then, on your Julia prompt, type

```
Pkg.add("JuMP")
Pkg.add("Gurobi")
```

(If you are ineligible to use a free license of Gurobi, use the **Cbc** solver: `Pkg.add("Cbc")`)
Step 7. Open TextEdit (or any other text editor, for example Atom\textsuperscript{10}) and type the following, and save the file as script.jl in some folder of your choice.

```julia
using JuMP, Gurobi
m = Model(solver=GurobiSolver())
@defVar(m, 0 <= x <= 2)
@defVar(m, 0 <= y <= 30)
@setObjective(m, Max, 5x + 3*y)
@addConstraint(m, 1x + 5y <= 3.0)
print(m)
status = solve(m)
println("Objective value: ", getObjectiveValue(m))
println("x = ", getValue(x))
println("y = ", getValue(y))
```

If you are ineligible to use a free license of Gurobi, replace the first two lines by

\textsuperscript{10}http://atom.io
Chapter 1. Introduction and Installation

``` julia
using JuMP, Cbc
m = Model(solver=CbcSolver())
```

- **Step 8.** Open a terminal window\(^\text{11}\) at the folder that contains your `script.jl`.

- **Step 9.** Type `ls -al` to check your script file.

- **Step 10.** Type `julia script.jl` to run your script.

\(^\text{11}\): To do this, you can drag the folder to the Terminal.app icon in your dock, or see [http://osxdaily.com/2011/12/07/open-a-selected-finder-folder-in-a-new-terminal-window/](http://osxdaily.com/2011/12/07/open-a-selected-finder-folder-in-a-new-terminal-window/)
1.3. Installing Julia

After a few seconds, the result of your julia script will be printed. Done.

Please proceed to Section 1.3.4.

1.3.4 Running Julia Scripts

When you are ready, there are basically two methods to run your Julia script:

- In your Command Prompt or Terminal, enter `C:> julia your-script.jl`
- In your Julia prompt, enter `julia> include("your-script.jl")`

1.3.5 Installing CPLEX

Instead of Gurobi, you can install and connect the CPLEX solver, which is also free to academics. Installing CPLEX is a little more complicated task.

CPLEX in Windows

You can follow this step by step guide to install:
Chapter 1. Introduction and Installation

1. Log in at the academic initiative page\textsuperscript{12} at the IBM website.

2. Follow the instructions on the page.

3. Download an appropriate version to your system:
   - \texttt{cplex\_studio126\_win\_x86\_32.exe} for 32-bit systems
   - \texttt{cplex\_studio126\_win\_x8664.exe} for 64-bit systems.

4. Reboot.

5. Run the downloaded exe file You may need to right-click the exe file and “Run as Administrator.”

6. Run Julia and add the \texttt{CPLEX.jl} package:

   \begin{verbatim}
   julia> Pkg.add("CPLEX")
   \end{verbatim}

7. Ready. Test the following code:

   \begin{verbatim}
   using JuMP, CPLEX
   m = Model(solver=CplexSolver())
   @defVar(m, x <= 5)
   @defVar(m, y <= 45)
   @setObjective(m, Min, x + y)
   @addConstraint(m, 50x + 24y <= 2400)
   @addConstraint(m, 30x + 33y <= 2100)

   status = solve(m)
   println("Optimal objective: ",getObjectiveValue(m))
   println("x = ", getValue(x), " y = ", getValue(y))
   \end{verbatim}

**CPLEX in Mac OS X**

The instruction includes how to deal with .bin file on Mac OS X:

\textsuperscript{12}https://www-304.ibm.com/ibm/university/academic/pub/jsps/assetredirector.jsp?asset_id=1070
1.3. Installing Julia

1. Log in at the academic initiative page at the IBM website.
2. Follow the instructions on the page.
3. Download an appropriate version to your system: cplex_studio126.osx.bin.
4. Place the file in your home directory: /Users/[Your User Name]. Copying and pasting from the Download directory should work here.
5. To install, open Terminal.
6. At the prompt, type in

```
/bin/bash ~/cplex_studio126.osx.bin
```

and hit enter. Follow all the prompts.

To add the CPLEX.jl package to Julia, follow:

1. Open your ~/.bash_profile file to edit:

```
open -e ~/.bash_profile
```

2. Add the following line to your ~/.bash_profile file: (change [USER NAME])

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:
/Users/[USER NAME]/Applications/IBM/ILOG/CPLEX_Studio126/plex/bin/x86-64_osx/
```

Note that the above code needs to be a single line.

3. Reload your profile:
Chapter 1. Introduction and Installation

source ~/.bash_profile

4. Run Julia and add the CPLEX.jl package:

```julia
julia> Pkg.add("CPLEX")
```

5. Ready. Test the following code:

```julia
using JuMP, CPLEX
m = Model(solver=CplexSolver())
@defVar(m, x <= 5)
@defVar(m, y <= 45)
@setObjective(m, Min, x + y)
@addConstraint(m, 50x + 24y <= 2400)
@addConstraint(m, 30x + 33y <= 2100)
status = solve(m)
println("Optimal objective: ",getObjectiveValue(m))
println("x = ", getValue(x), " y = ", getValue(y))
```

1.4 Installing IJulia

As you have seen in Section 1.2, JuliaBox provides a nice interactive programming environment. You can also use such an interactive environment in your local computer. JuliaBox is based on Jupyter Notebook\(^\text{14}\). Well, at first there was IPython notebook that was an interactive programming environment for the Python language. It has been popular, and now it is extended to cover many other languages such as R, Julia, Ruby, etc. The extension became the Jupyter Notebook project. For Julia, it is called IJulia, following the naming convention of IPython.

To use IJulia, you need Python and Jupyter installed in you computer. The Anaconda Python is an easy-to-install distribution of Python and Jupyter (and many other python packages). In Section 3.10, you will need to install the Anaconda Python anyway to use plotting.

\(^{14}\text{http://jupyter.org}\)
1.4. Installing IJulia

1. Download and install the Anaconda Python 2.7 from https://www.continuum.io/downloads.

2. Open a new terminal window and run Julia. Install IJulia:

```
 julia> Pkg.add("IJulia")
```

3. To open the IJulia notebook in your web browser:

```
 julia> using IJulia
 julia> notebook()
```

It will open a webpage in your browser that looks like the following screenshot:

![IJulia Notebook](image)

The current folder will be your home folder. You can move to another folder and also create a new folder by clicking the “New” button on the top-right corner of the screen. After locating a folder you want, you can now create a new IJulia notebook by clicking the “New” button again and select the julia version of yours, for example “Julia 0.4.1”. See Figure 1.1.

It will basically open an interactive session of the Julia Language. If you have used Mathematica or Maple, the interface will look familiar. You can test basic Julia commands. When you need to evaluate a block of codes, press Shift+Enter, or press the “play” button. See Figure 1.2.
Chapter 1. Introduction and Installation

Figure 1.1: Creating a new notebook

Figure 1.2: Some basic Julia codes.
1.5 Package Management

If you properly install a plotting package like PyPlot.jl (details in Section 3.10.1), you can also do plotting directly within the IJulia notebook as shown in Figure 1.4.

Personally, I prefer the REPL for most tasks, but I occasionally use IJulia, especially when I need to test some simple things and need to plot the result quickly, or when I need to share the result of Julia computation with someone else. (IJulia can export the notebook in various formats, including HTML and PDF.)

What is the REPL? It stands for read-eval-print loop. It is the Julia session that runs in your terminal; see Figure 1.3, which must look familiar to you already.

1.5 Package Management

There are many useful packages in Julia and we rely many parts of our computations on packages. If you have followed my instructions to install Julia, JuMP, Gurobi, and CPLEX, you have already installed a few packages. There are some more commands that are useful in managing packages.
Chapter 1. Introduction and Installation

Figure 1.4: Plotting in IJulia
1.5. Package Management

```
 julia> Pkg.add("PackageName")
```

This installs a package, named `PackageName`. To find its online repository, you can just google the name `PackageName.jl`, and you will be directed to a repository hosted at GitHub.com.

```
 julia> Pkg.rm("PackageName")
```

This removes the package.

```
 julia> Pkg.update()
```

This updates all packages that are already installed in your machine to the most recent versions.

```
 julia> Pkg.status()
```

This displays what packages are installed and what their versions are. If you just want to know the version of a specific package, you can do:

```
 julia> Pkg.status("PackageName")
```

```
 julia> Pkg.build("PackageName")
```

Occasionally, installing a package will fail during the `Pkg.add("PackageName")` process, usually because some libraries are not installed or system path variables are not configured correctly. Try to install some required libraries again and check the system path variables first. Then you may need to reboot your system or restart your Julia session. Then `Pkg.build("PackageName")`. Since you have downloaded package files during `Pkg.build("PackageName")`, you don’t need to download them again; you just build it again.
1.6 Helpful Resources

Readers can find codes and other helpful resources in the author’s website at

http://www.chkwon.net/julia

which also includes a link to a Facebook page of this book for discussion and communication.

This book does not teach everything of the Julia Language—only a very small part of it. When you want to learn more about the language, the first place you need to visit is

http://julialang.org/learning/

where many helpful books, tutorials, videos, and articles are listed. Also, you will need to visit the official documentation of the Julia Language at

http://docs.julialang.org/

which I think serves as a good tutorial as well.

When you have a question, there will be many Julia enthusiasts ready for you. Relevant mailing lists are

- julia-users\footnote{https://groups.google.com/forum/#!forum/julia-users}: Any general questions about the Julia Language.
- julia-opt\footnote{https://groups.google.com/forum/#!forum/julia-opt}: Questions about mathematical optimization.

You can also ask questions at http://stackoverflow.com with tag julia-lang.

The webpage of JuliaOpt is worth visiting. JuliaOpt is a group of people who develop and use optimization related packages in the Julia Language. The website provides a nice overview of the available packages and well-tailored examples. The website is

http://www.juliaopt.org
1.6. Helpful Resources