Institute for Theoretical Physics
University of Cologne

Prof. Dr. S. Trebst
Jan Attig

Computational Many-Body Physics

Exercise Sheet 0

Summer Term 2018

Due date: discussed on Wednesday, **18th April** 2018

Website: www.thp.uni-koeln.de/trebst/Lectures/2018-CompManyBody.shtml

With this very first exercise sheet, which will be discussed during the first tutorial on April 18th, we want to **refresh your coding skills** in the programming language **julia**, which we will be using throughout the course. Please carefully work through these warm up problems, as most of the technical aspects covered on this sheet will be necessary for solving the first homework assignment (out later this week). Most of you who have participated in the undergraduate computational physics course will find that the warm up problems of exercise 1 below are pretty straight-forward[1].

Setting up your Julia environment

Before you can start coding in julia, some basic installation has to take place. In this part, we will guide you through the process of **installing julia** on your computer as well as setting up all required packages.

1) Installing Julia

The installation of the julia language on your couputer depends on the operating system that you are using. The most up-to-date instructions can be found on the julia website.

To install julia on a **Windows** machine, go to the downloads section of the julia website and download the latest binaries (".exe"-file) for your Windows version. Then, install the file as you are used to installing other programs before.

To install julia on a **Mac**, go to the downloads section of the julia website and download the latest macOS package. Then, double click the file to add it to the list of programs.

To install julia on a **Linux** system, go to the downloads section of the julia website and download the latest binaries for the linux distribution you are using. Extract them to a folder of your choice on your local hard drive. Finally you need to do one of two things: Either link the julia binary file with a symbolic link into your /usr/bin folder or add the julia bin/ folder to your path variable.

In any of the cases you should be able to open the julia interpreter by first opening a terminal, typing julia and hitting enter. Note that on Windows a terminal can be opened by hitting Windows-Key + R, enter cmd and press enter.

[1] For those of you who might not have enjoyed such an undergraduate course these exercises give you a pointer as to what level of programming skills we expect you to have already. If you are not familiar with the julia programming language, but do have prior programming experience, we recommend that you install julia and then visit one of the recommended tutorials as described in the "Learning julia syntax" section below and subsequently go through this exercise sheet.

2) First time setting up packages

Most of the julia content has been parceled into libraries (called packages) that you can add to your code whenever you need it. To make use of these packages, you have to install them into your package directory.

If you installed julia on your computer for the first time, you have to first initialize the package directory before using it by typing

```
Pkg.init()
```

into the interpreter. This command will create and initialize your package directory.

For adding packages, you can use the syntax

```
Pkg.add("NAME")
```

which adds the package with the name NAME to your packages. Since the exercises will use various packages over and over, you can already start adding the following packages: PyPlot, IJulia, JLD, Optim, DifferentialEquations by typing:

```
Pkg.add("PyPlot")
Pkg.add("IJulia")
Pkg.add("JLD")
Pkg.add("Optim")
Pkg.add("DifferentialEquations")
```

3) Maintaining your packages

Once your package directory is properly set up, it needs little to no attention. Packages can be used in your code by employing the syntax

```
using NAME
```

to import the package with name NAME into your code and use its functions.

Packages are under ongoing bug-fixing and developement, so to stay up to date with the most up to date version, you can use the package update syntax of julia:

```
Pkg.update()
```

which automatically looks up all your installed packages and searches for newer versions. So run this command from time to time to stay up to date with your versions!

Some remarks on coding with julia

In this section we will give you some very basic advice concerning how you should code with julia and where to find further information on how to code with julia.

1) How and where to write julia code

In principle, there are three very different scenarios, which allow you to program some code in julia: (i) the julia **interpreter**, (ii) generic julia text **files**, and (iii) julia **notebooks**. Every scenario has certain advantages over the others, so it is best to understand which one to work with for a given task.

Julia Interpreter a.k.a. REPL

Let us start with the concept of the julia **interpreter** (also known as REPL). The interpreter can be opened from the terminal by typing "julia" and pressing enter. Inside the interpreter, you can enter a line of code and let it be executed immediately.
The interpreter is a useful tool for quickly coding only few lines, such as small scripts, plotting small datasets or even using julia as a calculator. Nevertheless, as soon as the code starts exceeding about 10 lines, you will want to switch to one of the other methods explained below.

Julia text files

Any serious amount of code can be written in julia **text files** which can be executed using the julia interpreter. The text files themselves, which should end with ".jl" to indicate that they contain julia code, can be written using an editor of your choice.

Executing a julia file with filename "filename.jl" can be done by typing "julia filename.jl" into the terminal.

In general, you should use julia text files for any larger amount of code. They are the most reliable way to store, distribute, and versionize your code and should be your first tool of choice.

Julia notebooks

The third option you have is to develop code in a julia **notebook**. This tool can be accessed via the IJulia package from within the julia interpreter. You can type

```
using IJulia
notebook()
```

into the interpreter to open a new julia notebook server in your browser in which you can create individual notebooks.

Inside a julia notebook, you can have multiple cells with code or markdown formatted text which can be executed by the julia interpreter individually.

Julia notebooks function as a mixture between the direct execution of code within the interpreter and the ability to save code similar to text files. Nevertheless there are certain issues which disqualify notebooks as the tool of choice (which we will cover in the first tutorial session in greater detail). However, they can be used in combination with julia files to construct well documented and clearly readable code as described below.

Julia notebooks in combination with julia text files

The method of choice for handing in solutions to the exercises is from our perspective a **hybrid solution** of julia notebooks and julia text files. The text files can be used for defining functions or global variables in great detail whereas the calling of functions in small scripts will be well documented with additional text inside a julia notebook. In this way, one can use the predefined functions in many different settings and various notebooks whereas the executed code itself can be documented using markdown.
Note: Including a text file called "juliatextfile.jl" into your code can be done by using the syntax

```
include("juliatextfile.jl")
```

2) Learning julia syntax

To get started learning about the julia language and syntax, we suggest that you visit the section of the julia website which contains a list of recommended julia tutorials.

Exercise 1: Julia warm up (Homework)

To test your basic julia knowledge we have compiled a very small first exercise – available as julia notebook download on the course website.

Your task is to provide/complete the source codes for the warm up problems. Go through the notebook, preferably exercise by exercise, and complete the code in the predefined cells. As you will see, topics include control flow and how to define functions as well as lists and arrays.

Exercise 2: Plotting in Julia (To be discussed in tutorial on April 18th)

In the second exercise, we want to show you the basics of plotting in julia. For any plotting to work you have to include a package which provides the plotting functions for julia. There are, however, many different packages available, all of which have their pros and cons.

- **PyPlot** - great for starting, relies on python, extensive documentation via matplotlib, bad for making animation movies
- **Plots.jl** - unifies syntax for many different backends (like PyPlot, GR, PlotlyJS, ...), can quickly construct animations and movies, tricky to find documentation
- **Gadfly** - rendering to SVG or PDF, integration with DataFrames.jl
- **GR** - fast plotting, support of many file formats
- and many others ...

In the tutorial, we will familiarize you with some of them so that you can choose the appropriate one for the exercises throughout the semester. We will start with an introductory sheet (available by April 18th) for the plotting module PyPlot.

Exercise 3: Twin prime numbers (Discussed in tutorial / Homework)

As a last step in the tutorial, we want to show you how to **properly document** the solutions you are handing in for the homework assignments. We suggest that you hand in the hybrid solution of julia notebooks and julia text files, which will be discussed in the tutorials.

Your task in this exercise is to implement a function for testing whether a number n is a prime number. Consecutively we want you to identify all pairs of prime numbers $(n, n+4)$ between 2 and $2,000,00$ and provide an answer to how many there are.

The goal of this exercise goes beyond the simple completion of the task described above. You should try to implement the function in a julia text file and include this text file in a notebook where you conclusively discuss your overall solution. For handing in, compress the two files into a .zip folder which you can send to the tutor.