

Exercise class 1

Maximilian Holst
Toni Heugel

General information

First part:

- Discussion of the previous exercise
- Additional information about the new exercise

Second part:

- You work on the tasks
- You can ask questions

General information

- Default programming language: Julia
- You are free to use your favorite language
- Hand-in is voluntary

The procedure for handing in your results (if you want to do so) is:

1. Create a new directory named in the format `exXX_lastname_firstname` (XX is the corresponding exercise sheet number).
2. Copy the solution files into this directory. The files should be formatted in a readable way. Make sure that you include both the *source code* (e.g. `*.jl`) as well as a *scientific report* in pdf-format that includes a list of used parameters, the plots you produced (with these parameters) and additional comments/explanations.
3. Zip the directory and send it to `holstm@phys.ethz.ch`.

Hand-ins that do not follow this guideline will not be accepted! Please do not hand in files that do not need to be corrected!

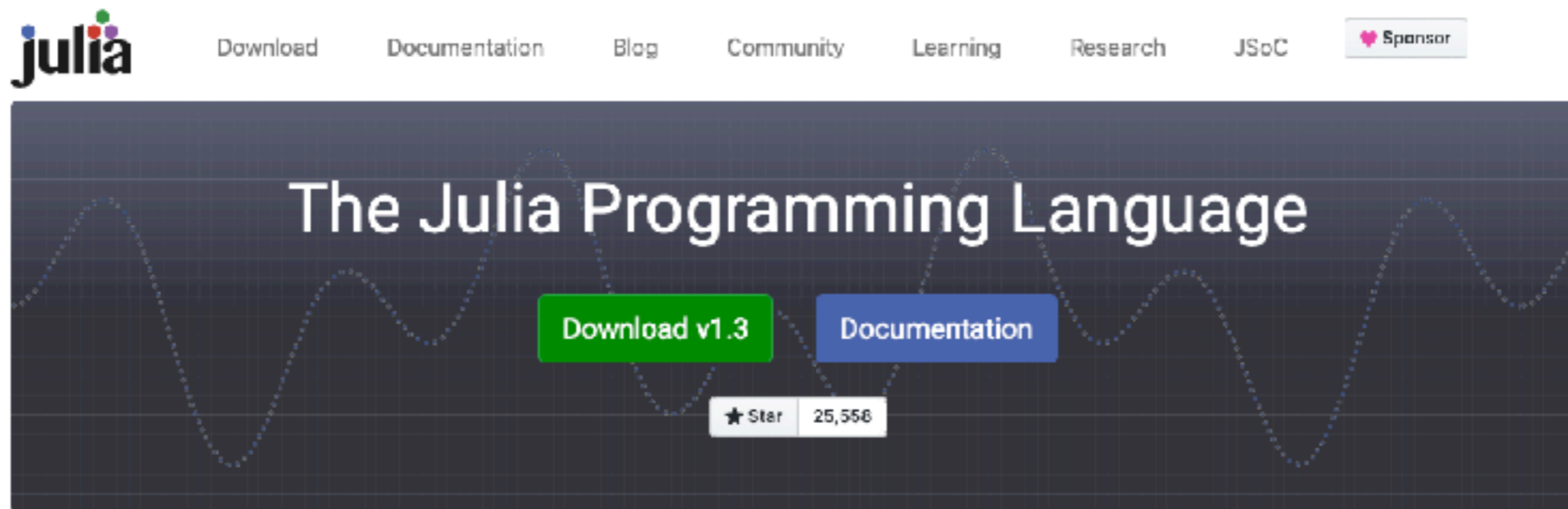
Why Julia?

It solves the tradeoff: fast coding vs. fast execution

- Code like in python
- Execution nearly as fast as C

Installing Julia

<https://julialang.org>



Julia in a Nutshell

Julia is fast!

Julia was designed from the beginning for [high performance](#). Julia programs compile to efficient native code for [multiple platforms](#) via LLVM.

General

Julia uses [multiple dispatch](#) as a paradigm, making it easy to express many object-oriented and [functional programming patterns](#). It provides [asynchronous I/O](#), [debugging](#), [logging](#), [profiling](#), a [package manager](#), and more.

Dynamic

Julia is [dynamically-typed](#), feels like a scripting language, and has good support for [interactive use](#).

Easy to use

Julia has high level syntax, making it an accessible language for programmers from any background or experience level. [Browse the Julia microbenchmarks](#) to get a feel for the language.

Optionally typed

Julia has a [rich language of descriptive datatypes](#), and type declarations can be used to clarify and solidify programs.


Open source

Julia is provided under the [MIT license](#), free for everyone to use. All [source code](#) is publicly viewable on GitHub.

Installing Julia

Download Julia

If you like Julia, please consider starring us on [GitHub](#) and spreading the word!

 Star 25,558

We provide several ways for you to run Julia:

- In the terminal using the built-in Julia command line using the binaries provided below.
- Using [Docker](#) images from [Docker Hub](#) maintained by the [Docker Community](#).
- [JuliaPro](#) by [Julia Computing](#) includes Julia and the [Juno IDE](#), along with access to a curated set of packages for plotting, optimization, machine learning, databases and much more (requires registration).

Please see [platform specific instructions](#) for further installation instructions and if you have trouble installing Julia. If the provided download files do not work for you, please [file an issue in the Julia project](#). Different OSes and architectures have varying [tiers of support](#), and are listed at the bottom of this page.

Current stable release: v1.3.1 (Dec 30, 2019)

Checksums for this release are available in both [MD5](#) and [SHA256](#) formats.

Windows (.exe) [help]	32-bit	64-bit	
macOS 10.8+ (.dmg) [help]		64-bit	
Generic Linux Binaries for x86 [help]	32-bit (GPG)	64-bit (GPG)	
Generic Linux Binaries for ARM [help]	32-bit (ARMv7-a hard float) (GPG)	64-bit (AArch64) (GPG)	
Generic FreeBSD Binaries for x86 [help]		64-bit (GPG)	
Source	Tarball (GPG)	Tarball with dependencies (GPG)	GitHub

Run Julia

- In console:

```
julia  
code...  
exit()
```
- `julia myscript.jl`
- Juno, Jupiter,...

Installing packages

Install package:

```
julia> using Pkg; Pkg.add("myPkg")
```

Include package:

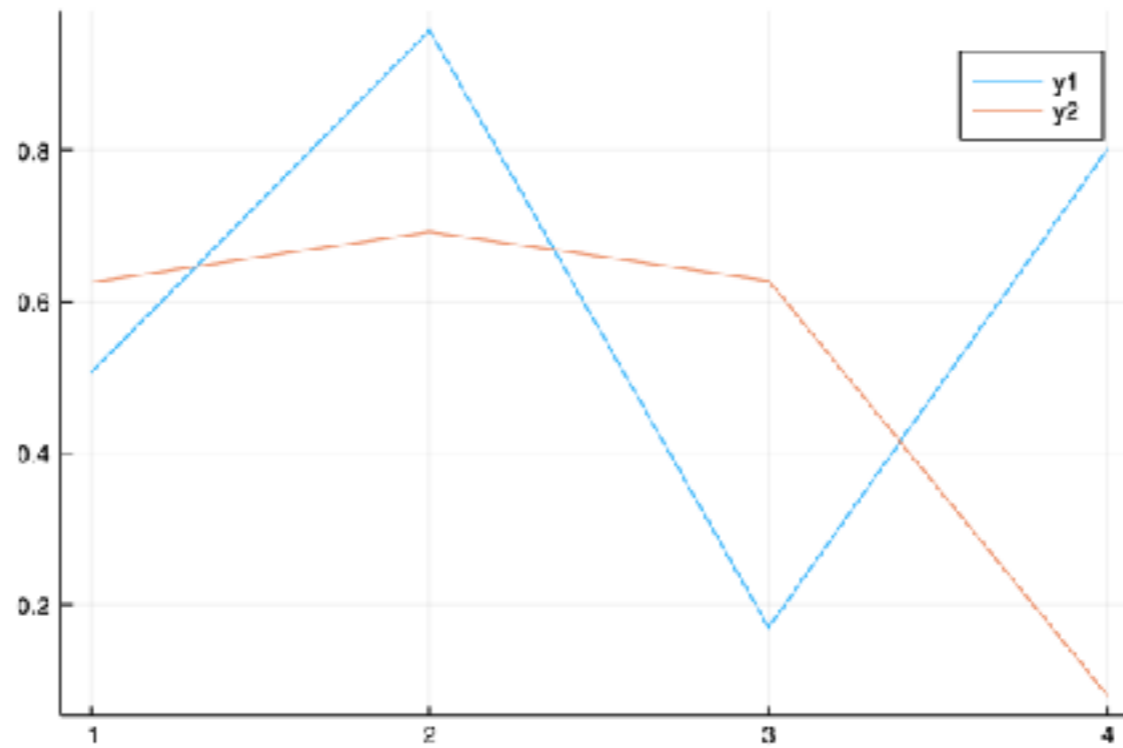
```
julia> using myPkg
```

```
julia> import myPkg
```

```
julia> myPkg.function(x)
```


Plotting

```
[julia> using Plots
[julia> gr()
Plots.GRBackend()
[julia> plot(rand(4,2))
julia> ]
```



Arrays

Empty:	<code>a = []</code>	<code>a[1]</code> index counting starts from 1!
Zeros:	<code>zeros(3)</code>	
Ones:	<code>ones(3,2)</code>	<code>a[from:step:to]</code>
Column vector:	<code>[1, 2, 3]</code>	<code>a[from:to]</code>
Row vector:	<code>[1 2 3]</code>	

Arrays

N-dimensional array:

- $A = [[1, 2, 3,] [4, 5, 6]]$
- $A = [1\ 4; 2\ 5; 3\ 6]$

$A[\text{row}, \text{col}]$

$A[4] \rightarrow 4$

Array of arrays:

$B = [[1, 2, 3,], [4, 5, 6]]$

$B[2][3]$

Functions: $\text{exp.}(A)$

Array

Example

```
1
2 a = [i+j for i=1:2, j=1:3] # construct array
3
4 N_row, N_col = size(a)
5
6 # transpose a
7 a_T = zeros(Int, N_col, N_row) # integer array
8 for i = 1:N_col, j = 1:N_row # for loop
9     a_T[i,j] = a[j,i]
10 end
```

Functions

- $f(x, y) = 2x + y$

- ```
function f(x, y)
 2x + y
end
```

```
function f(x, y)
 return 2x + y
end
```

Functions which manipulate input:

```
function g!(x)
 x[2] = 2
end
```

**Exercise 1. Ising model**

*Goal: We start by simulating the 3D Ising model using the Metropolis-based single-spin flip Monte Carlo method. For those who attended last semester's lecture it is (to some degree) a revision.*

Write a program for a Monte Carlo simulation to solve the three-dimensional Ising model with periodic boundary conditions. Implement the *single-spin flip* Metropolis algorithm for sampling. As you will have to reuse this code for upcoming exercise sheets, it might be worth to make sure that it is well-structured!

**Task 1:** Measure and plot the *energy*  $E$ , the *magnetization*  $M$ , the *magnetic susceptibility*  $\chi$  and the *heat capacity*  $C_V$  at different temperatures  $T$ .

**Task 2:** Determine the critical temperature  $T_c$ .

*Hint: You should obtain  $T_c \simeq 4.51$ .*

**Task 3:** Study how your results depend on the system size.

*Hint: Start with small systems to reduce the computation time.*

**Task 4 (OPTIONAL):** Save computation time by avoiding unnecessary reevaluations of the exponential function. To achieve this, use an array to store the possible spin-flip acceptance probabilities.

**Task 5 (OPTIONAL):** Plot the time dependence of  $M$  for a temperature  $T < T_c$ .

*Hint: For small systems you should be able to observe sign-flips in  $M$ .*

# 3D Ising model

---

- Hamiltonian: 
$$H = -J \sum_{\langle i,j \rangle} \sigma_i \sigma_j - H \sum_i \sigma_i$$
- Mean of observable: 
$$\langle O \rangle = \sum_x O(x) \frac{1}{Z} e^{-\beta H(x)}$$
- Relevant observables at zero magnetic field:

|                |                                                          |
|----------------|----------------------------------------------------------|
| Energy         | $\langle H \rangle$                                      |
| Magnetization  | $\left\langle \frac{1}{N} \sum_i \sigma_i \right\rangle$ |
| Susceptibility | $\beta(\langle M^2 \rangle - \langle M \rangle^2)$       |
| Heat capacity  | $\beta^2(\langle E^2 \rangle - \langle E \rangle^2)$     |

# Monte Carlo sampling

---

Rather than computing the observables over the exponentially large phase space, we sample them on statistically relevant configurations

1. Choose randomly a new configuration in phase space
2. Accept or reject the new configuration, depending on the strategy
3. Compute physical quantities
4. Repeat



# Metropolis algorithm

---

- Propose new configuration:  $\sigma_i \rightarrow -\sigma_i$

Probability =  $1/N$

- Accept new configuration with probability

$$\min \left( 1, \exp \left( -\beta(E_{new} - E_{old}) \right) \right)$$

✓ Ergodic

✓ Detailed balance

# Implementation

---

- Start with any spin configuration (e.g. random)
- Perform MC-steps until the system is thermalized

## Sampling:

- Calculate observables and keep them for sampling
- Do subsweep to obtain a new uncorrelated configuration ( again calculate the observables)
- Repeat

## Post processing

- Calculate mean and fluctuations

Repeat for different temperature and system size

# Hints

---

- Julia is column major
- Calculate energy and magnetization for new configuration locally

```
1 1 1 1 1 1 1 1 1 1
1 1 1 1 -1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
```

- Use a lookup table for the the Metropolis acceptance probabilities