

HOW TO:

INSTALL JULIA & THE OWCF

(ON WINDOWS)

BY HENRIK JÄRLEBLAD

HOW TO:

INSTALL JULIA & THE OWCF (ON WINDOWS)

BY HENRIK JÄRLEBLAD

Alright, so this is a guide on how to successfully install the Julia programming language and the Orbit Weight Computational Framework (OWCF), on Windows (any new Windows operating system).

This guide will most likely get updated continuously, as the programming languages and packages evolve. Also, as of this version, **the estimated amount of space you need on your hard drive (Julia language + packages) is approx 1GB. Please make sure you have that available (preferably more).**

Please check the date below, to find out if you need to acquire a more recent version of this guide.

This version was written on 14th of October 2022.

Henrik Järleblad
+45 3 11 55 767
henrikj@dtu.dk
Fysikvej, Building 309
2800 Kgs. Lyngby
DTU
www.henrikjarleblad.se

A CODE EDITING PROGRAM

So, there are many programs which you can use to edit your code. And many integrated development environments (IDE) also offer the possibility of executing the code directly after writing it in the editor. Such as Eclipse, Atom (for Julia, Juno), IntelliJ and many, many more.

I suggest that you start by doing some googling, to find out if the IDE you are currently using is compatible with the Julia programming language. If not, then I suggest you download visual studio code (VSCode), the most basic but still sophisticated code-editing program in the world (in my humble opinion). Just google “visual studio code download”, and you should find your way. What you will then do, when working with the Julia orbit tomography packages, is writing the code in VSCode, and executing it using the Windows powershell. I will show you how to do this. However, you could also execute code and code blocks directly in VSCode using the Julia VSCode extension. You can read about that here <https://www.julia-vscode.org/docs/stable/gettingstarted/> and here <https://www.julia-vscode.org/docs/stable/userguide/runningcode/>.

You can of course use your own IDE. Just make sure that it is compatible with Julia.

1.

DOWNLOADING JULIA

To download the Julia programming language, please go to <https://julialang.org/downloads/> and choose the latest stable release for Windows, and then the 64-bit version.

Go ahead and download that file. PLEASE NOTE! It is ok if it's a newer version than v1.6.2. In fact, the newer the better.

If you want to be really safe that you have got the correct file, distributed by the officials of the Julia programming language (and that no-one has hacked their website, and replaced the downloadable files with a virus or something similar), you should verify that you get the same SHA-256 hash as they claim. The risk that the official website has been hacked is of course very low. But still, it is not zero. Please refer to the last section of this document, to find a short guide on how to verify the downloaded installation file using SHA-256.

Current stable release: v1.6.2 (July 14, 2021)

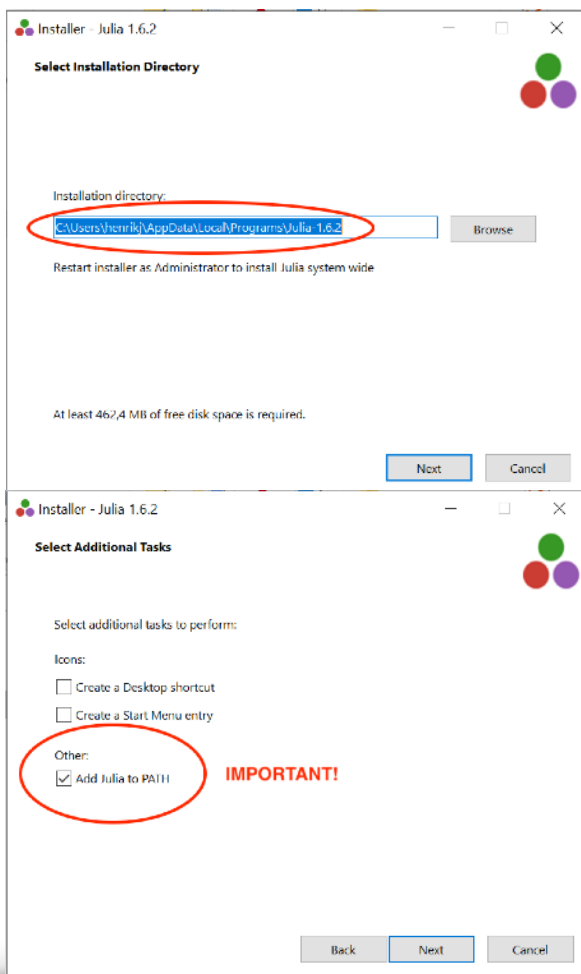
Checksums for this release are available in both MD5 and SHA256 formats.

Windows [help]	64-bit (installer), 64-bit (portable)	32-bit (installer), 32-bit (portable)
macOS [help]	64-bit	
Generic Linux on x86 [help]	64-bit (GPG), 64-bit (musl) ^[1] (GPG)	32-bit (GPG)
Generic Linux on ARM [help]	64-bit (AArch64) (GPG)	32-bit (ARMv7-a hard float) (GPG)
Generic Linux on PowerPC [help]	64-bit (little endian) (GPG)	
Generic FreeBSD on x86 [help]	64-bit (GPG)	
Source	Tarball (GPG)	Tarball with dependencies (GPG)
Source	Tarball (GPG)	Tarball with dependencies (GPG)
Generic Linux on x86 [help]	64-bit (GPG)	
Generic Linux on PowerPC [help]	64-bit (little endian) (GPG)	

Alright, you should now have the correct Julia .exe file in your

Downloads folder. Go ahead and install it, follow the instructions. **Please note**, it can be a good idea to note the installation directory where Julia will be installed on your computer. You might need it later. **Also**, make sure that you tick the 'Add Julia to PATH' option at the '**Select Additional Tasks**' window.

That way, when your IDE or your Windows Powershell tries



to execute "julia", they will find the correct file to execute. Now, please go ahead and proceed with the installation.

2.

ADDING JULIA TO PATH

(IF YOU DIDN'T TICK THE 'ADD JULIA TO PATH' BOX DURING INSTALLATION)

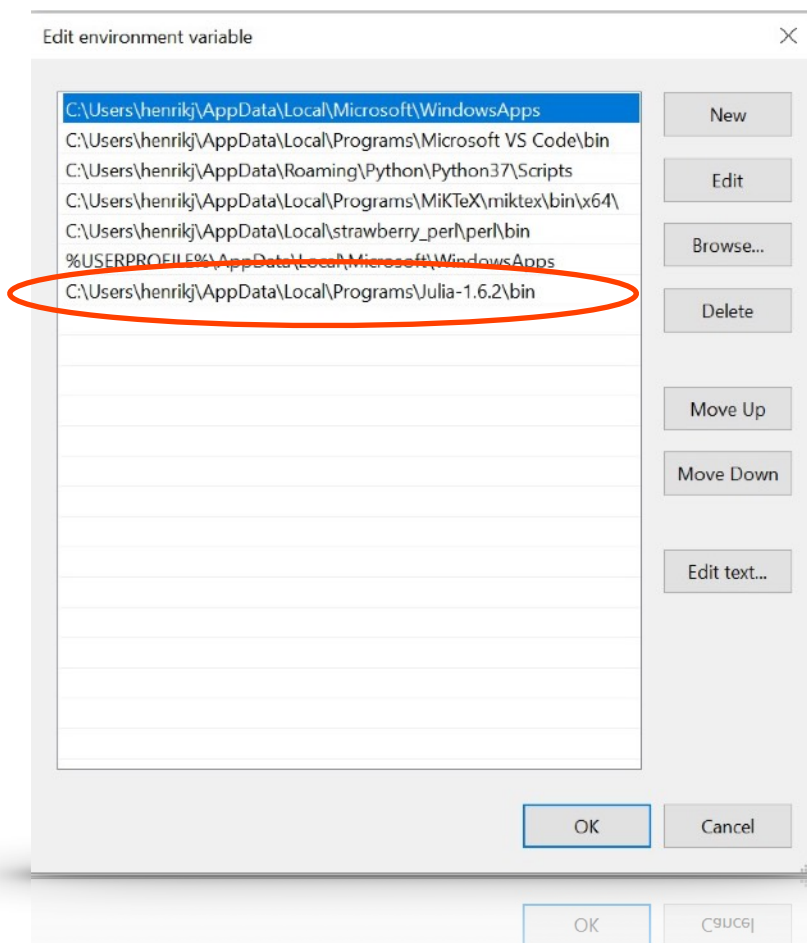
So you didn't tick the 'Add Julia to PATH' box during installation? No worries, we can fix this! I hope that you took note of the directory in which Windows installed Julia. If not, it should be somewhere similar to the path shown in the figure on the previous page (but with your Windows username instead of mine naturally).

Now, in the search bar in the bottom left corner of your Windows Desktop, type 'environmental variables'. You should see something similar to the screenshot below reveal itself as the top search result (or equivalent in your selected Windows language). Click on that.

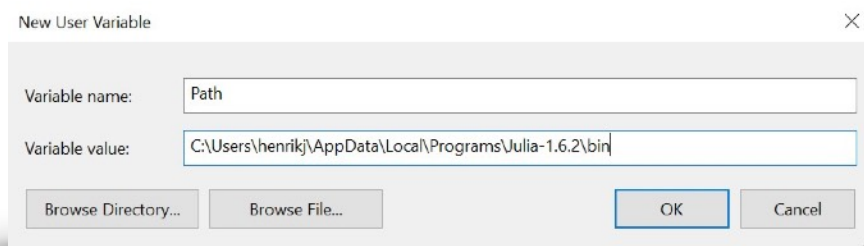


You should see many different variables under 'User variables for [YOUR USERNAME]'. If you see a 'Path' variable, double-click on that one. If not, create a 'Path' variable by clicking on 'New...' under the 'User variables...' section (NOT 'System variables').

If you double-clicked on 'Path', you should see a list of 'C:\...' paths and/or '%...%' paths. Click on 'New' and then write (or paste) the path to the bin folder of your installed version of Julia. Click 'Ok' and then 'Ok' again.



If you needed to create a new 'Path' variable by clicking on 'New...' under the 'User variables...' section, give it the variable name 'Path' (without the ' of course) and the variable value of the path to the bin folder of your installed version of Julia. Click 'Ok' and then 'Ok' again.



Congratulations! You should now be able to use the Julia programming language in the Windows Powershell. Let's start installing some Julia packages!

3.

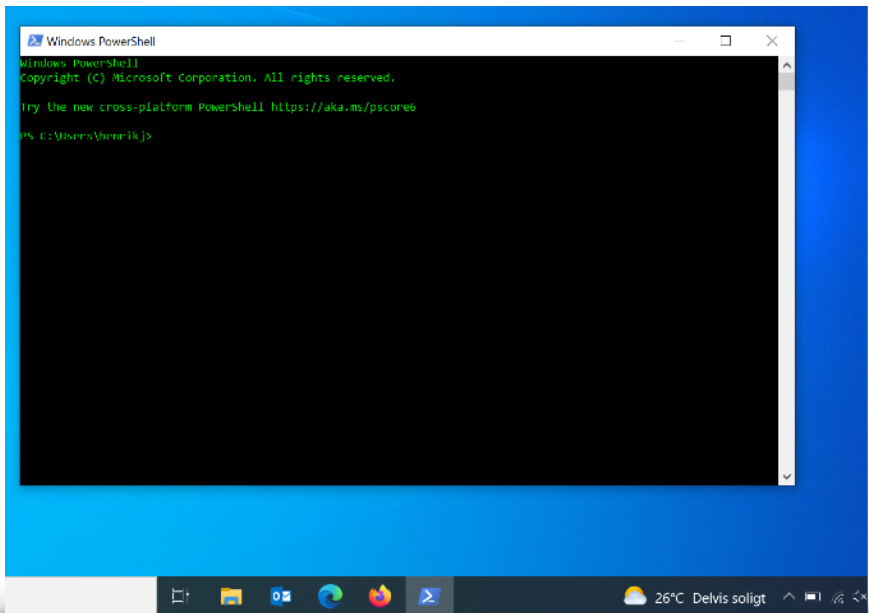
INSTALLING PACKAGES

Here is where we start using the Windows Powershell. Even if you are planning on using Julia and the OWCF packages in an IDE, you can still benefit from reading this section, because all the packages should be installed and ready-to-use when you start up your IDE later. However, Julia has an environment functionality. So, for every project you work on, Julia has the possibility of loading different packages. What I want to say is, that even if you follow the steps in this section and use your IDE afterwards, you might still run into some problems. Unfortunately, I do not know which IDE you are using, so I have no way of helping you there. Please turn to the almighty google (or duck-duck-go) if that should be the case.

Alright, let's start. Go to your Windows desktop (or start a new one, if you're a multi-tasker like me) and search for 'Powershell' in the bottom-left search bar. Hit enter. You should now have opened a Powershell window.

Don't worry if your Powershell doesn't look exactly like mine (colors, text etc). I (think that I) customised mine. You can too, if you want to.

Anyway, make sure the little underscore '_' is blinking (it means the Powershell is waiting for your input, if you are



completely new to this. If it is not blinking, just click on the Powershell window). Make sure that you have unzipped your download of the OWCF.zip folder. It's a good idea to save it in a useful folder on your computer, because you will be using it a lot. Use the 'cd' command to navigate to the OWCF folder, and start Julia by typing 'julia' and hit enter, as the screenshot below shows. **PLEASE NOTE!** Your OWCF folder will **not** be "G:\My Drive\DTU\codes\OWCF". It is only an example. Please find out where you have saved and unzipped your OWCF-folder, and use the 'cd' command to navigate there!

Now, type 'using Pkg', hit enter, 'Pkg.activate(".")' and hit enter. Press ']' to confirm that you have indeed activated the OWCF environment. As the screenshot below shows. Press backspace to go back to normal Julia mode. The '[' key in Julia activates package mode. Read more about that by Googling/Duck-duck-going 'Julia Pkg.jl'.

Here we pause for a bit, because I need to tell you something **before you proceed further and start installing any**

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\henrik> cd "G:\My Drive\DTU\codes\OwCF"
PS G:\My Drive\DTU\codes\OwCF> julia

      _       _       _
     / \   / \   / \   \
    / _ \ / _ \ / _ \  \
   / ___// ___// ___ \
  /____//____//____ \
                 \

Documentation: https://docs.julialang.org
Type "j" for help, "j?" for pkg help.
Version 1.6.2 (2021-07-14)
Official https://julialang.org/ release

julia> using Pkg

julia> Pkg.activate(".")
Activating environment at "G:\My Drive\DTU\codes\OwCF\Project.toml"

julia>
```

packages (more specifically, before you install the PyCall.jl package). If you already have Python installed on your Windows, you should let Julia know of the path to the Python executable prior to installing the PyCall.jl package. You do this by typing

`ENV["PYTHON"] = "... path of the python executable ..."`

into the Julia prompt. You can find out the path of your Python executable by starting Python, type `'import sys'`, hit enter, type `'print(sys.exec_prefix)'`, hit enter, and note the path. You then add `'\python.exe'` to the end of the path when specifying the `ENV["PYTHON"]` Julia information. In my case (most likely **not in your case**), this means

`ENV["PYTHON"] = "C:\Program Files (x86)\Python37-32\python.exe"`

You might have to change all `'\'` to `'/'`.

Also, remember, if you want to write Python code directly in Julia (yep, that can be done with Julia) you have to make

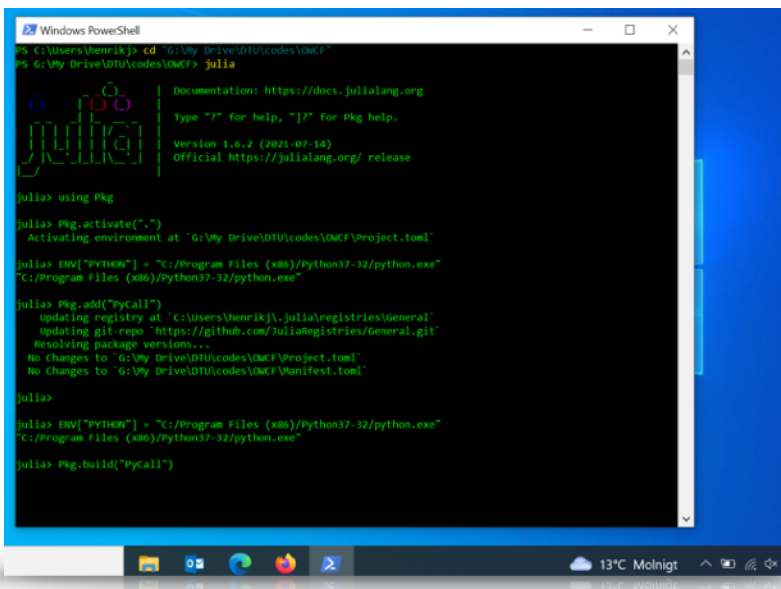
sure that the Python packages are installed in your Python (so for example, if you want to write Python code in Julia that uses the *h5py* Python package, you have to have it installed in Python first). The OWCF actually uses the following Python packages

- numpy
- scipy
- netCDF4
- h5py

so please make sure that you have those installed in Python before trying to use the OWCF. To connect Julia with your Windows pre-installed Python executable, after you have done the ‘ENV[“PYTHON”]=“...”’ step, do the following

- Pkg.add(“PyCall”)
- ENV[“PYTHON”] = “... path of the Python executable ...”
- Pkg.build(“PyCall”)

The second step is often redundant (it’s exactly the same as the earlier step). But it’s good to perform anyway, just in case. As Figure A below shows (**log messages are likely to differ**).



```
PS C:\Users\henrikj> cd "G:\My Drive\DTU\codes\OWCF"
PS G:\My Drive\DTU\codes\OWCF> julia

Documentation: https://docs.julialang.org
Type "j?" for help, "j?" for pkg help.
Version 1.6.2 (2021-07-14)
Official https://julialang.org/ release

julia> using Pkg
julia> Pkg.activate(".")
  Activating environment at "G:\My Drive\DTU\codes\OWCF\Project.toml"

julia> ENV["PYTHON"] = "C:\Program Files (x86)\Python37-32\python.exe"
"C:\Program Files (x86)\Python37-32\python.exe"

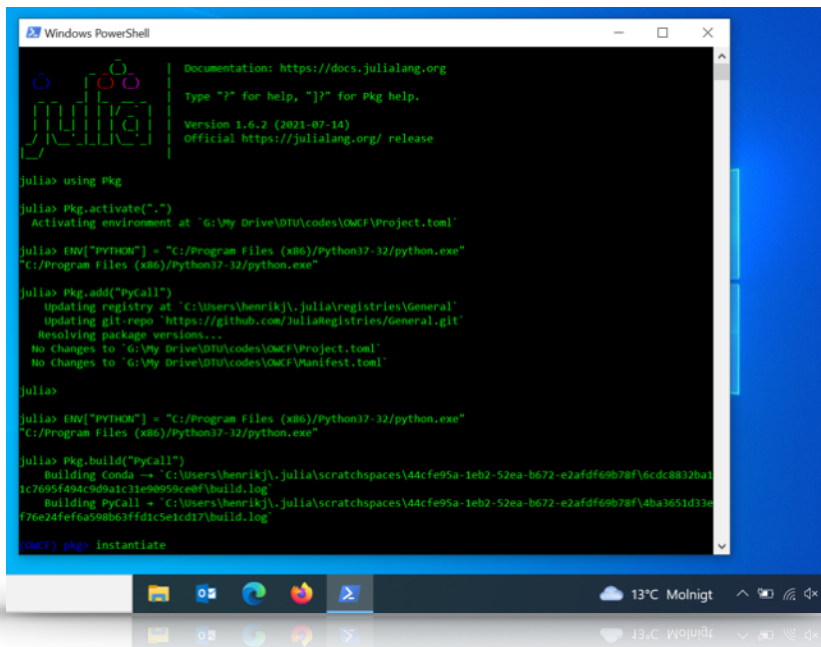
julia> Pkg.add("PyCall")
  Updating registry at "C:\Users\henrikj\.julia\registries\General"
  Updating git-repo "https://github.com/JuliaRegistries/General.git"
  Resolving package versions...
  No changes to "G:\My Drive\DTU\codes\OWCF\Project.toml"
  No changes to "G:\My Drive\DTU\codes\OWCF\Manifest.toml"

julia>
julia> ENV["PYTHON"] = "C:\Program Files (x86)\Python37-32\python.exe"
"C:\Program Files (x86)\Python37-32\python.exe"

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

Figure A

Now, back to installing Julia packages and the OWCF. For the next step, simply press ']', type 'instantiate' and hit enter. As Figure B below shows.



```
Documentation: https://docs.julialang.org
Type "?>" for help, "]]?" for Pkg help.
Version 1.6.2 (2021-07-14)
Official https://julialang.org/ release

julia> using Pkg

julia> Pkg.activate(" ")
Activating environment at "G:\My Drive\DTU\codes\OWCF\Project.toml"

julia> ENV["PYTHON"] = "C:/Program Files (x86)/Python37-32/python.exe"
"C:/Program Files (x86)/Python37-32/python.exe"

julia> Pkg.add("PyCall")
  updating registry at "C:\Users\henrik\.julia\registries\General"
  updating git-repo "https://github.com/JuliaRegistries/General.git"
  Resolving package versions...
  No changes to "G:\My Drive\DTU\codes\OWCF\Project.toml"
  No changes to "G:\My Drive\DTU\codes\OWCF\Manifest.toml"

julia>

julia> ENV["PYTHON"] = "C:/Program Files (x86)/Python37-32/python.exe"
"C:/Program Files (x86)/Python37-32/python.exe"

julia> Pkg.build("PyCall")
  Building Conda -> "C:\Users\henrik\.julia\scratchspaces\44cfe95a-1eb2-52ea-b672-e2afdf69678f\6cdc8832ba1
1c7695f494c80a1c31e90959c0f\build.log
  Building PyCall -> "C:\Users\henrik\.julia\scratchspaces\44cfe95a-1eb2-52ea-b672-e2afdf69678f\4ba3651d33a
776e24fe6a29863ffdc5e1cd17\build.log

(OWCF) pkg> instantiate
```

Figure B

This will be a good time to go grab a coffee, or even go for lunch. Julia is installing and pre-compiling all the packages needed for the OWCF (yes, **pre**-compiling. It is unique to Julia. It optimises the efficiency and speed of the Julia language). This long process might instead have happened during the previous Julia-Python connection steps.

When you come back, you should now have all the packages you need and be ready to start using the OWCF. Congratulations!

If you have not used Julia before. I would advice you to at least read up on the basics on the www.julialang.org website. For example, how do you close Julia down now that you have installed everything?

4.

WRITING AND EXECUTING YOUR VERY FIRST JULIA FILE

You can write code in so many different ways and using so many different code editing programs. Actually, if you want to use the basic Notepad text editing program, go ahead. However, I will show you how to use the VSCode code editing program to write the code and the Windows Powershell to execute it. VSCode now also supports execution of Julia code and plotting directly in the IDE, so I will show you how to do that as well. Let's get started.

First, open an empty document and save it as 'test.jl' (note the '.jl' file extensions). Next, write these lines of code to change directory to the OWCF/ folder and activate the environment.

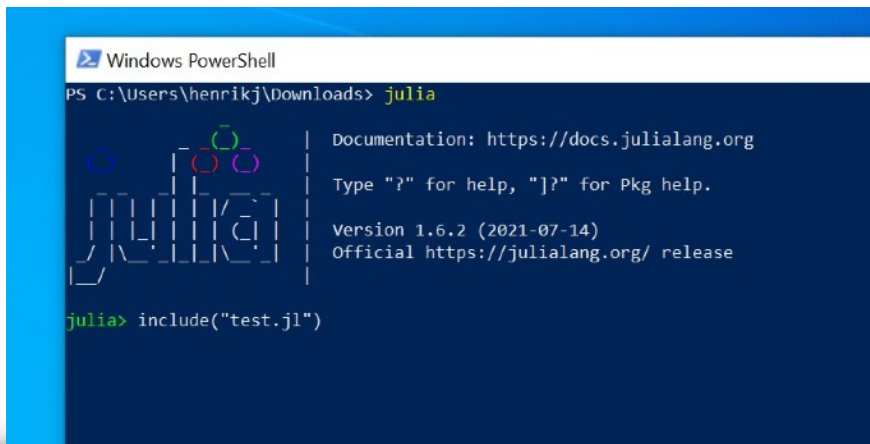
```
1  folderpath_OWCF = "/the/path/to/folder/of/your/OWCF/"
2  cd(folderpath_OWCF)
3  using Pkg
4  Pkg.activate(".")
```

Next, write the following lines of code to load the *Plots.jl* package, generate a 1D array of integers (x) between 1 and 50, a 1D array of 50 random floats (y) and plot y as a function of x (please see start of next page).

```
5 using Plots
6 x = collect(1:50)
7 y = rand(50)
8 Plots.plot(x,y)
8 bplot2.bplot(x,λ)
```

In Julia, you can generate a range without having to allocate memory for all the elements. This is done using the ‘:’ and via the syntax ‘*start:increment:stop*’. If you don’t provide an increment, Julia will simply assume evenly spaced elements. Then, if you want Julia to actually allocate memory for the range as an array, you use the *collect()* function. This will convert the type of the variable from a *Range* to an *Array*. Use the function *typeof()* anytime you want to find out what type a variable is. Now, let’s execute your first script!

Open a Powershell window and navigate to the folder

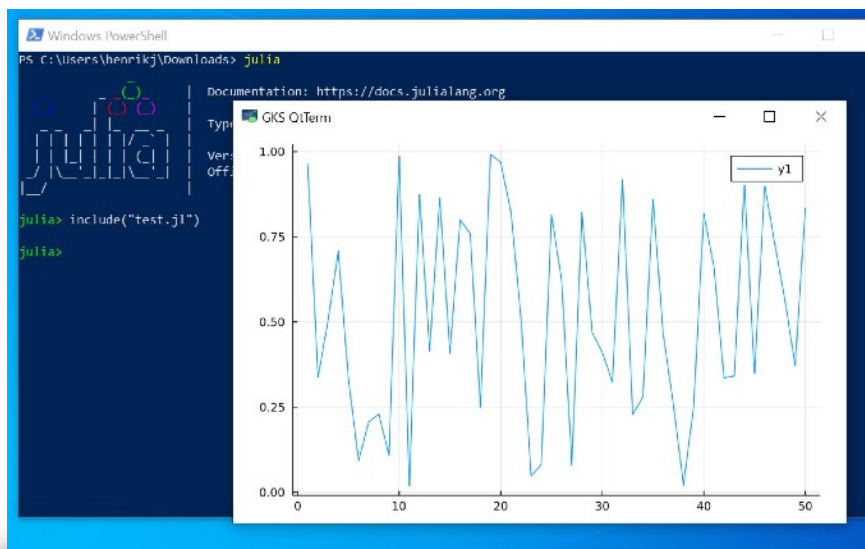


How to start a Julia session and execute a .jl script

where you have saved your ‘test.jl’ Julia script (you navigate using the ‘cd [TARGET FOLDER]’ command). Then, you can either start a Julia session with ‘julia’ and execute the script in Julia with the command ‘include(“test.jl”)’ or simply typing

'julia test.jl' directly in the Powershell prompt. The former way of doing it might be more suitable for plotting. So go ahead and try it!

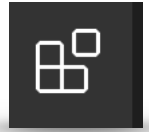
If you did it correctly, after a couple of seconds a pop-up window should appear (powered by the GKS QtTerm program) showing you the plotted data. The reason that it takes a couple of seconds the first time is that Julia loads the whole *Plots.jl* package with the 'using Plots' code line. If you have started an active Julia session, these needs to be done only once in the beginning. So go ahead and try executing the script with 'include("test.jl")' and you should see the plot window immediately appearing. When you're done, don't forget to type 'exit()' into the Julia session REPL (prompt) to close Julia and the GKS QtTerm to close the plotting functionality.



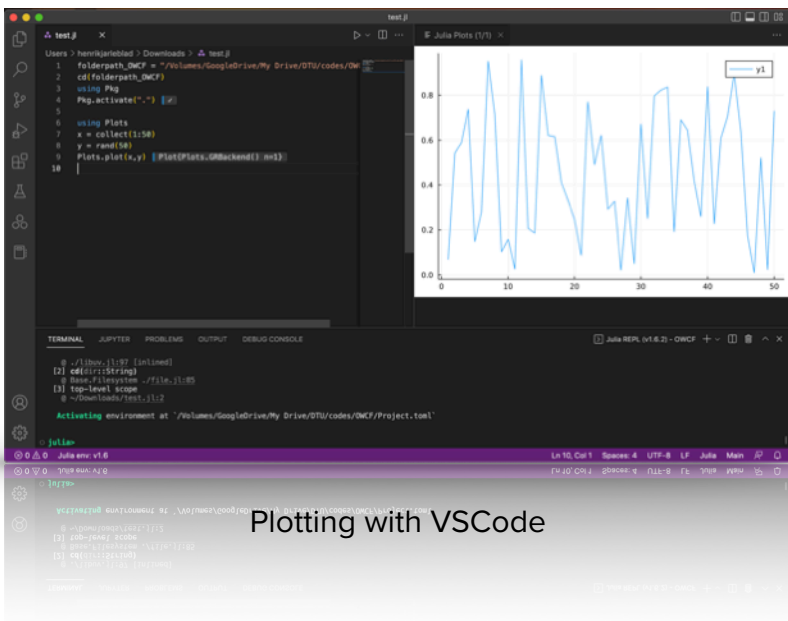
Now, there is also a way of plotting directly in VSCode, using the Julia extension and integrated plotting functionality. Let's take a look at how to do that.

Open your 'test.jl' script using VSCode. If you look to the left, you should see a symbol that resembles three connected

boxes and a fourth disconnected box. These are your VSCode extensions. Click on the symbol. Search for 'julia' in the search bar and download the Julia VSCode extension (Julia Language support). After it has installed, you might need to restart VSCode.



Then, as explained here <https://www.julia-vscode.org/docs/stable/userguide/runningcode/>, put the cursor at the end of your first line (so right after the second 's' in the 'using Plots' line, before the line break) and hit ctrl + enter. A terminal will open up in VSCode and execute the code line. Then, as explained on the website linked above, you can use different delimiters ('##') and different commands to execute your Julia code in VSCode. For now, just hit shift + enter and watch how the plot window appears to the right, integrated right into VSCode.



Plotting with VSCode

That is it! Good luck with your Julia programming!

EXTRA: USE SHA-256 TO VERIFY THE DOWNLOADED JULIA FILE

To verify the installation file with SHA-256, start by downloading the SHA-256 file, just above the Julia programming language installation files (<https://julialang.org/downloads/>)

Current stable release: v1.5.1 (Aug 25, 2020)

Checksums for this release are available in both MD5 and SHA256 formats.

Windows [help]	64-bit (installer), 64-bit (portable)	32-bit (installer), 32-bit (portable)
macOS [help]	64-bit	
Generic Linux on x86 [help]	64-bit (GPG), 64-bit (musl) ^[1] (GPG)	32-bit (GPG)
Generic Linux on ARM [help]	64-bit (AArch64) (GPG)	
Generic FreeBSD on x86 [help]	64-bit (GPG)	
Source	Tarball (GPG)	Tarball with dependencies (GPG) GitHub

Make sure you have both the Julia installation file and the .sha256 file in the same folder (presumably your Downloads folder?). Open a Powershell window. You should be in your home folder. Type “cd Downloads” (or navigate to the folder where you put the files. If you put them on your desktop, type “cd Desktop”). Hit enter. Now, you need to perform SHA-256 on the installation file to generate a bunch of letters and numbers. In your Powershell, type

```
get-filehash .\julia-1.6.2-win64.exe -algorithm sha256
```

where you change the “julia-1.6.2-win64.exe” filename to whatever version you’re currently attempting to install. (So if you want to install Julia 1.6.1, you write “julia-1.6.1-win64.exe”

instead and so on). Hit enter. This should generate a bunch of letters and numbers, as shown below (could be different).

```
PS C:\Users\henrikj\Downloads> get-filehash .\julia-1.6.2-win64.exe -algorithm sha256
```

Algorithm	Hash	Path
SHA256	480111d380f2f09f8f1330e430b67cf9330005722f50608495962d3400e02977	C:\Users\henrikj\Downloads\julia-1.6.2-win64.exe
SHA512	380115080c2f00cde10b5800f67cf9330005722f50608495962d3400e02977	C:\Users\henrikj\Downloads\julia-1.6.2-win64.exe

Your job is now to make sure that these letters and numbers match those that the Julia language officials have written in the .sha256 file. So, open the .sha256 using Notepad (or a text editor of your choice). Compare the two sets of letters and numbers. Make sure that they are identical. If they are not, you might have just downloaded a virus. Don't open it.

```
PS C:\Users\henrikj\Downloads> get-filehash .\julia-1.6.2-win64.exe -algorithm sha256
```

Algorithm	Hash	Path
SHA256	380115080c2f00cde10b5800f67cf9330005722f50608495962d3400e02977	C:\Users\henrikj\Downloads\julia-1.6.2-win64.exe

```
PS C:\Users\henrikj\Downloads>
```

julia-1.6.2.sha256 - Notepad

File Edit Format View Help

2f832283e7549d0a8c8b62c7d834b032940051eafccae01c17d70c0070d4c517 julia-1.6.2-freebsd-x86_64.tar.gz

fc48785202e960698de457741330e075a89f80b063096c3145186583f348cd julia-1.6.2-linux-aarch64.tar.gz

76229a04fc259c3d70b720f80c248f9891bd85c154cf7cc67bcbdc3350c4f julia-1.6.2-linux-armv7l.tar.gz

360f6c0abb3ceafda135a0ec943f9c336cf843445c14a06d053433362e0c86 julia-1.6.2-linux-i686.tar.gz

84c1c0ea3bb3c29192e17016d5da9eb0bdaa791a78d50a80b09528f750648a julia-1.6.2-linux-ppc64le.tar.gz

3eb4b57750bd1ed38f6c409e989501ab445c95bcb01ab02bd60f5bd1e823240 julia-1.6.2-linux-x86_64.tar.gz

6fb5e110a10c7c715b3e82442c9226b9cda3947cc7d7f15900c19c14d46f437c9 julia-1.6.2-macosx64.dmg

023b0f7c34f978190e94596a93403f09561b3b0758a2d57c73865a0c233aad julia-1.6.2-macosx64.tar.gz

5ff279bc733a99a9582f0db59eb3d18a3fa77b0d3c2733030279a258c8c5d49c julia-1.6.2-musl-x86_64.tar.gz

7ce362630ebc4c44a45592873e8347dae2b372d77dc7664df97d8d9e53502a julia-1.6.2-win32.exe

c3376063ac58515f1f9cf42e434884dc3d3f5782a06574a9f1450f3c72256bd5 julia-1.6.2-win32.zip

380115080c2f00cde10b5800f67cf9330005722f50608495962d3400e02977 julia-1.6.2-win64.exe

25f8533076cd275c85c395cf70c0ff6b731245501f795177807040bc7c326735 julia-1.6.2-win64.tar.gz

e32a29eddac7a84bd9e9617913dcba21da6f01e7eb7499674218e464fc3 julia-1.6.2-win64.zip

They match!
(lowercase/
uppercase
does not matter)

does not matter

A FEW EXTRA THOUGHTS

When you use a package for the first time, it might need to do something unique (sort of) to Julia. It might need to pre-compile. This takes a long time (especially for large packages) but the result is that many things that would need to be compiled in other languages is already compiled to machine code. This is one of the things that makes Julia so fast (in some cases, as fast as C). So don't worry if pre-compiling some packages takes a long time. It's normal. Just wait and take a coffee or similar. Julia will be super-fast afterwards. As always. Except for plotting. Julia is super slow sometimes when it comes to plotting. At least for now (Julia v1.6-v1.7).