Quarto Installation Notes

Darren Irwin

2023-05-13

These are notes on how I set up Quarto for use with Julia. I am taking these notes both to try out Quarto and to record the steps involved. I hope this will be of use for myself, students, and maybe others.

I am assuming you already have both Julia and Visual Studio Code (i.e., VS Code) installed, with the Julia extension in VS Code. If not, follow the instructions here: https://code.visualstudio.com/docs/languages/julia . (Another option is to use some other text editor, but I highly recommend VS Code and assume it below.)

The following is for MacOS:

First, I got set up with Juliaup and the newest version of Julia:

Install Juliaup (Julia version manager)

First, install Juliaup using this command in the Terminal:

```
curl -fsSL https://install.julialang.org | sh
```

You can then check whether you have installed Julia versions using:

```
juliaup status
```

If you have no Julia version yet, or do have a version but want to upgrade to the latest, get the latest release version using:

```
juliaup update release
```

At the time of writing this, the latest version of Julia is 1.9.0. The version installed on my machine is `1.9.0+0.x64.apple.darwin14`.

If you also want an older version as an option, you can install it using something like `juliaup add 1.8.5`. (Not needed for most users, since newer versions of Julia run older code.)

Install Quarto (and dependencies)

This is a somewhat lengthy process, because running Quarto relies on several other software packages, including Jupyter and iJulia. But the results are worth it!

I learned much of the below from: https://quarto.org/docs/computations/julia.html

Download and install Quarto from this site: https://quarto.org/docs/get-started/ . The version I have installed is `Quarto CLI 1.3.353 (Mac OS)`.

Install the Quarto extension for VS Code (using the standard way of choosing extensions in VS Code).

IJulia

This is a Julia package that is needed to render your Quarto files. To install, start a Julia REPL either in VS Code (using View > Command Palette > Julia: Start REPL) or in a terminal window by typing Julia. Then type these commands:

```
using Pkg
Pkg.add("IJulia")
```

This takes a couple minutes to install, with all the dependencies.

Jupyter

Jupyter notebooks can be used with Julia, Python, and R (and the name is sort of a combination of those). Quarto uses Jupyter as an intermediate step in processing the code into output. To install Jupyter, we simply need to call an IJulia command (see below) and this will trigger a prompt asking you whether to install Jupyter. Say yes (unless perhaps you already have Jupyter installed).

```
using IJulia
notebook()
```

Then the computer will ask `install Jupyter via Conda, y/n?` to which you can respond "y".

After some processing, a Jupyter browser window will open. You can choose quit to close, and the Julia prompt returns in the REPL. After this initializing of Jupyter, you don't need to open Jupyter in this way again (Quarto will simply use Jupyter directly).

Revise.jl

This package enables Quarto to interact more efficiently with Julia when you change code. In a Julia REPL, type:

```
using Pkg
Pkg.add("Revise")
```

To set up Revise to launch automatically within IJulia, create a text file containing the following:

```
try
  @eval using Revise
catch e
  @warn "Revise init" exception=(e, catch_backtrace())
end
```

Save the above file as `.julia/config/startup_ijulia.jl` (the dot at the start means this directory is hidden–the computer will probably ask you to confirm that you want to do this).

Jupyter-cache

This enables smart and efficient re-execution of parts of your code (only the parts dependent on changes you've made). In the Julia REPL, execute:

```
using Pkg
Pkg.add("Conda")
using Conda
Conda.add("jupyter-cache")
```

Almost there!

After the above, Quarto is essentially installed and should work well for rendering in HTML or DOCX. However, I found a few more things are needed for a satisfying experience rendering in PDF:

librsvg and BasicTeX

```
brew install librsvg homebrew/cask/basictex
```

For some reason, the download of this was really slow. Administrator password was then needed, then after a while it finished successfully.

TinyTeX

```
curl -sL "https://yihui.org/tinytex/install-bin-unix.sh" | sh
```

JuliaMono font

This font works well with monocode symbols used in Julia (without it, many of these symbols were missing from PDF output).

```
brew tap homebrew/cask-fonts
```

```
brew install --cask font-juliamono
```

Try a demo Quarto file

OK we are now ready to do a little prayer to the Quarto and Julia gods and try this all out. In VS Code make a new file called `QuartoDemo.qmd`. The `.qmd` ending is important as it tells VS Code this is a Quarto file. Put this in the file:

````
---
title: "Demo Quarto File"
author: "Darren Irwin"
date: "5/13/2023"
execute:
  echo: true
format:
  pdf:
    keep-tex: true
    monofont: "JuliaMono"
  html:
    code-fold: true
jupyter: julia-1.9
---

## Demo plot produced by Julia in Quarto

Plot function pair (x(u), y(u)).
See @fig-parametric for an example.

Unicode test: ` α  β `
(To get those symbols in Julia REPL, type \\alpha then tab, or \\beta then tab)

```{julia}
````
````

```
#| label: fig-parametric
#| fig-cap: "Parametric Plots"

using Plots

plot(sin,
     x->sin(2x),
     0,
     2π,
     leg=false,
     fill=(0,:lavender))
```

Now, click `Render HTML` in the upper right of your VS Code window. The resulting html version of your document may appear as a preview to the right, and the html file is automatically saved. Then try `Render DOCX` for a Word document. Then try `Render PDF` for a PDF. (Check the unicode symbols carefully to see if they are there. I think they should be if the above installations were done.)

The rendered document should display the output shown below.

Demo plot produced by Julia in Quarto

Plot function pair (x(u), y(u)). See Figure 1 for an example.

Unicode test: `α  β` (To get those in Julia REPL, type \alpha then tab, or \beta then tab)

```
using Plots

plot(sin,
     x->sin(2x),
     0,
     2π,
     leg=false,
     fill=(0,:lavender))
```

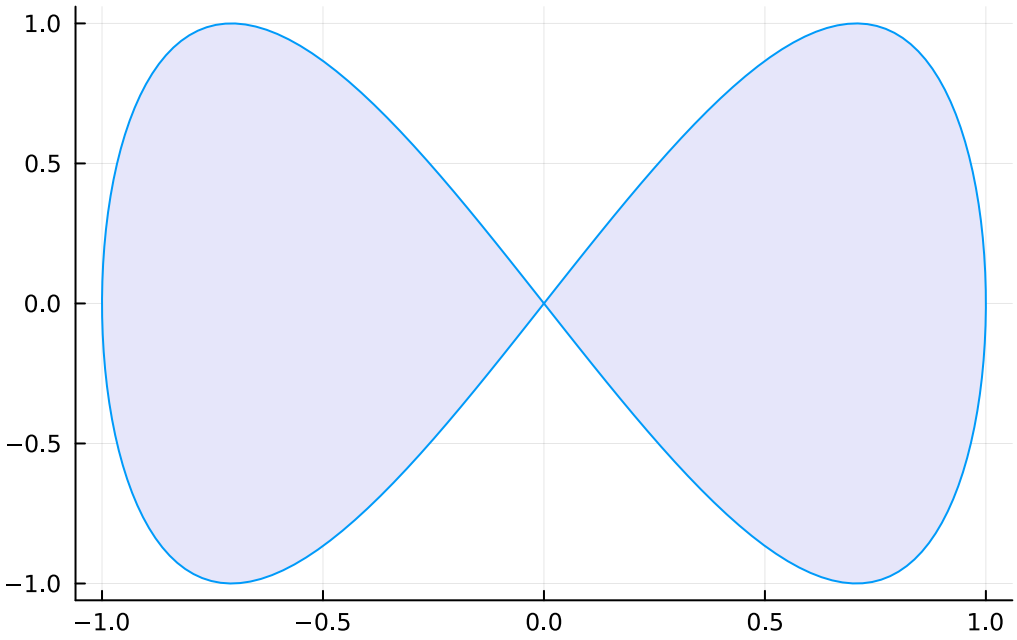

Figure 1: Parametric Plots

The code for the example plot above came from the Quarto website: https://quarto.org/docs/computations/julia.html