18.337 Project Proposal: Julia Documentation

Dragos Velicanu

December 17, 2013

Abstract

For the final project I have written an alternate documentation system that builds on top of the current framework and greatly expands the capabilities and usefulness of trying to get help while working at the Julia terminal. The system I created has a standard format for documenting the funnction, which has a Description, Input, Output, two URLs, instructions where to find the source code, a See Also, and unlimited examples. This content is displayed with nice colored formatting in the Julia command line terminal but still needs some work to look more than plain text in IJulia. In this report I show how a documentation entry is created, appended, and displayed. Some issues and proposed improvements are discussed. In conclusion this documentation system might not be goood enough to be used by the Julia community for all time, however since it's much better than what's currently available and it's already working and very fast to use, add, and understand, there is merit in adopting it until the final improved documentation is in place.

1 Introduction to Current Julia Documentation Status

Julia currently has some documentation online [1] which has an extensive manual, a search function, and a command and function index. The manual and search part of the documentation is pretty good already. What is currently lacking is how to use specific functions with examples that work out of the box. The builtin help(...) macro currently gives only a very brief description of the function with not a single example and is generally not very helpful for new Julia users from my personal experience trying to understand how to get plotting to work, how to get arbitrary precision to work, and even how the help itself works. In absence of better documentation, the best strategy so far for figuring how something works is to look inside the source code and trial and error.

The improvements to the documentation system described in this paper were partially motivated by what I personally find the most useful when learning a new language from scratch: examples; seeing them, running them, and modifing them. Additionally other well written documentation systems, like that of

Matlab and Mathematica, served as inspiration for the structure and content of what is to be displayed to the user.

2 The improved documentation system

My documentation function is built on top of the current Julia help, i.e. doc(function_name) calls help and in addition displays any new things that may have been written. I didn't modify the help function itself since it was not very transparent how it works (and there was no documentation about how it works!). I found the helpdb.jl that is loaded and read by help but the current framework only displays text help, so I wrote my own side documentation function that is more extensive in what it can show. The documentation system works from three functions: docwrite(), doc(), and docaddex() described below.

2.1 doc(fname)

doc() takes one input which is the name of the function for which the user wishes to display the documentation and outputs a display of text that has been written for the function. In addition, it also executes any examples that may have been written in the command line itself. This function is very simple, it calls help on the function and then it executes the documentation macro that has been written for the function. If no such custom documentation macro exists, it still calls help and informs the user that no custom documentation has been written for this yet.

```
julia> doc(doc)
# methods for generic function doc
doc(func_name) at C:\Users\velicanu\Downloads\julia-d6f7c7c781\bin\mydocs\docmacros.jl:50


Shows the extended documentation of a pre-existing function.

Input : function name
Output: executes the .jl documentation macro

Julia specific help is located here online: http://docs.julialang.org/en/latest/manual/
A good source of information about this is: null

See Also: docwrite docaddex help methods

Source Code: to see where the source code is located call the function methods(doc)

Examples:

julia> doc(abs)

This doc can be found at: mydocs/doc_doc.jl
```

Figure 1: The documentation entry for doc

2.2 docwrite(fname,desc,input,output,url1,url2,seealso,examples...)

docwrite() is the tool that helps in creating new documentation. As inputs, it takes a valid function name, a definition for that function, a description of the inputs the function accepts, a description of the outputs the function throws, a URL linking to the IJulia help page, another URL that contains other useful information regarding the function, the location of the source code, see also

which delineates other functions of potential interest to the user and lastly a set of infinite examples that are executable. The inputs are taken in the above order. What docwrite() does is that it creates a .jl file of the given specification which gets run when doc() on that function is called.

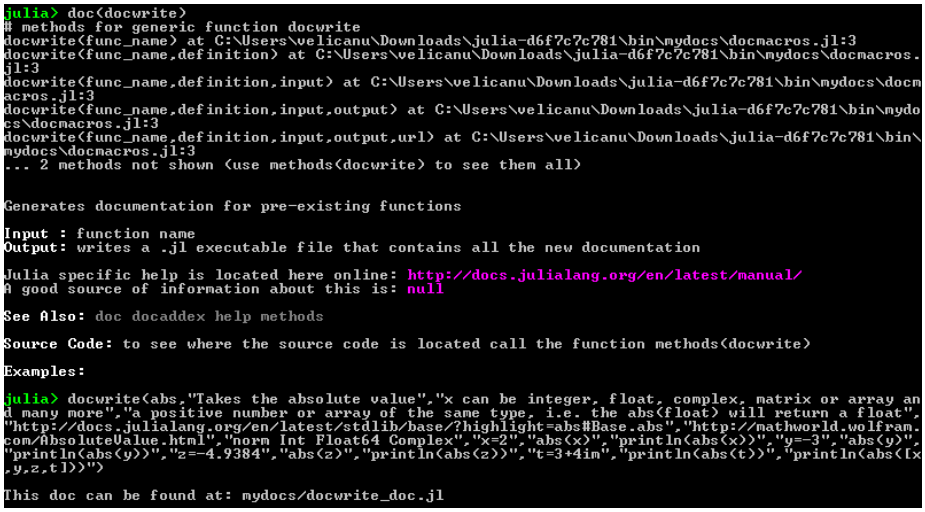


Figure 2: The documentation entry for docwrite

2.3 docaddex(fname,examples...)

docaddex() is there so that new examples could be added to existing documentation by different users. I decided to incorporate this function in addition to docwrite since it's likely that situations will arise where somoene wants to add a cool new example to an existing documentation without overwriting what is already there.

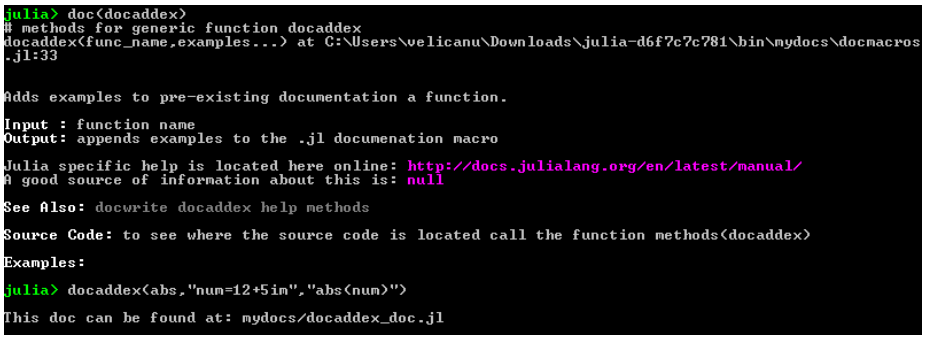
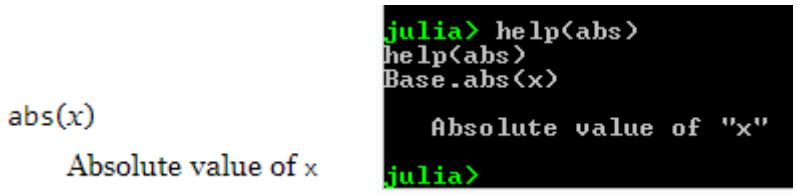


Figure 3: The documentation entry for docaddex

2.4 Before and After: Documentation structure

As can be seen from Figures 1-3 and the comparison below, the improved documentation shows the current help(...) entry, which defaults to calling methods(...) on the function if no help exists, in addition shows the custom description written for the doc, the input and output, the two relevant URL's for julia and general help, some functions to see also, how to get the source code, and the list of examples. In Figure 4 below is shown the "before" of what is currently available as documentation for an arbitrary function I chose as my example, and Figure 5 shows the "after" of what is possible for documentation for the same function with the new documentation system.



(a) Online help entry (b) Terminal help entry

Figure 4: Current help entry documentation both online and in Julia

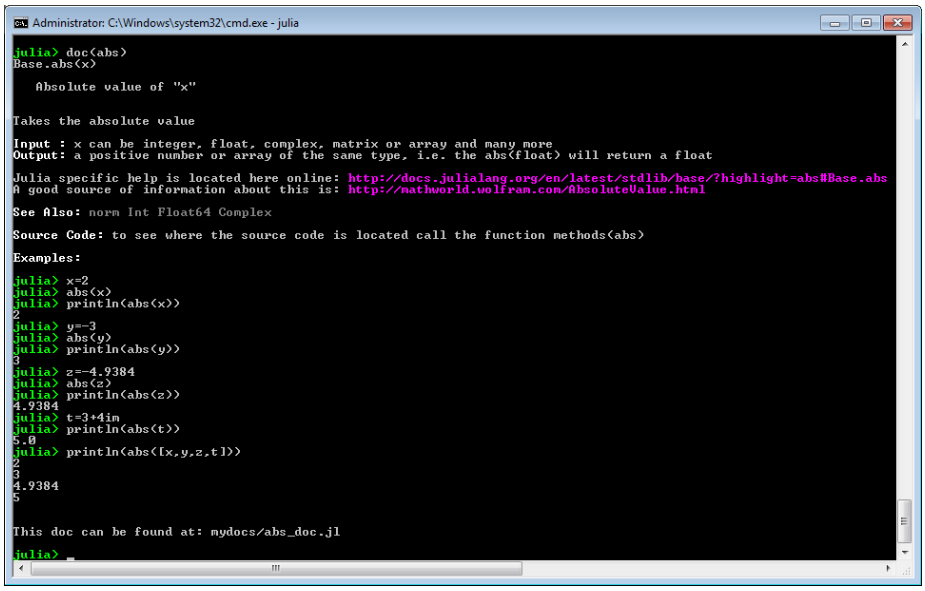


Figure 5: The new documentation entry of abs(example)

The code to create the new abs documentation using docwrite is repro-

duced below, and also happens to be the example that is shown when you ask doc(docwrite) shown in Figure 2. A peek inside the documenation .jl files shows that the command that was used to create them is stored at the top in a comment for educational purpose as well as a quick way to remake the documentation file with a small change if needed.

Listing 1: Code that generates the abs documentation

```
docwrite(abs,"Takes the absolute value","x can be integer,
    float, complex, matrix or array and many more","a
    positive number or array of the same type, i.e. the abs(
    float) will return a float","http://docs.julialang.org/en
    /latest/stdlib/base/?highlight=abs#Base.abs","http://
    mathworld.wolfram.com/AbsoluteValue.html","norm Int
    Float64 Complex","x=2","abs(x)","println(abs(x))","y
    =-3","abs(y)","println(abs(y))","z=-4.9384","abs(z)","
    println(abs(z))","t=3+4im","println(abs(t))","println(abs
    ([x,y,z,t]))")
```

The design and workflow of the entire system is shown in Figure 6. In short, docwrite creates/overrides a doc_functionName.jl file when executed with the appropriate parameters. Inside that .jl file is the information to be displayed in a series of println statements, some with the color that is shown on the terminal, followed by an arbitrarily long list of examples which get both printed and executed line by line. The macro docadd just sticks more examples in the same manner at the end of a pre-existing .jl documentation macro, leaving the rest untouched. Lastly, the macro doc just executes the .jl file matching the pattern doc_functionName.jl after calling help.

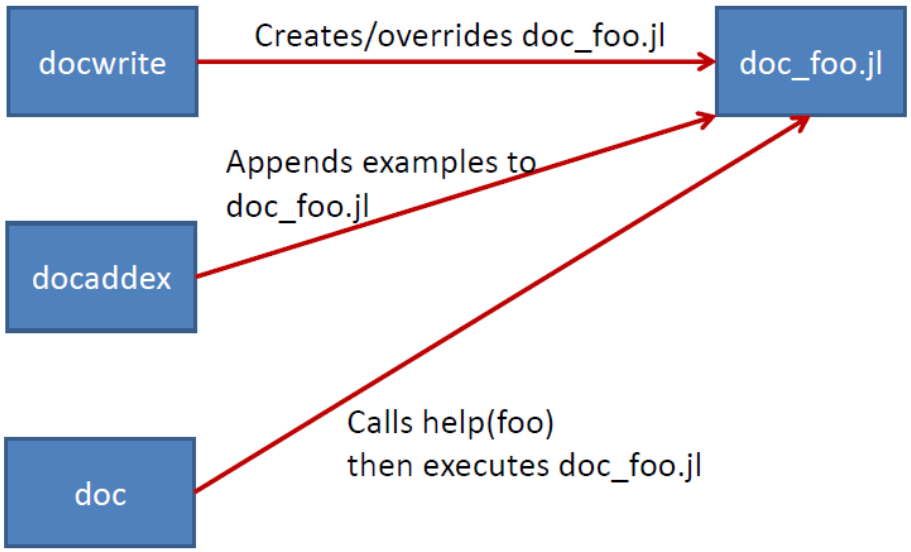


Figure 6: The workflow of the documentation system.

2.5 A neat example: plot documentation

Since my implementation thinks of all Julia documentation as executable .jl macros, the capability of what the documentation can do or show is far greater than what is possible with just text formatted in special ways. This last example shows a documentation for plotting in Julia that I wrote to show the power of this way of doing documentation. Documenting plot is a good example because currently no documentation whatsoever exists within the Julia help function, secondly plot comes from a package and raises the question of how to document and integrate functions from packages, and lastly plot is one of the hardest things to get working, especially if you can't find the documentation for it (the current best place is on github [4]). The documentation in Figure 7 not only adds documentation for an undocumented function, but also leaves little doubt for a novice user how to get it working and in addition displays the plot in the example. This shows the real power of thinking of documentation as macros instead of just text since now doc(someFunction) can show plots, images, SVDs... basically anything that can be done in Julia can be shown by the documentation, far beyond simple formatted text.

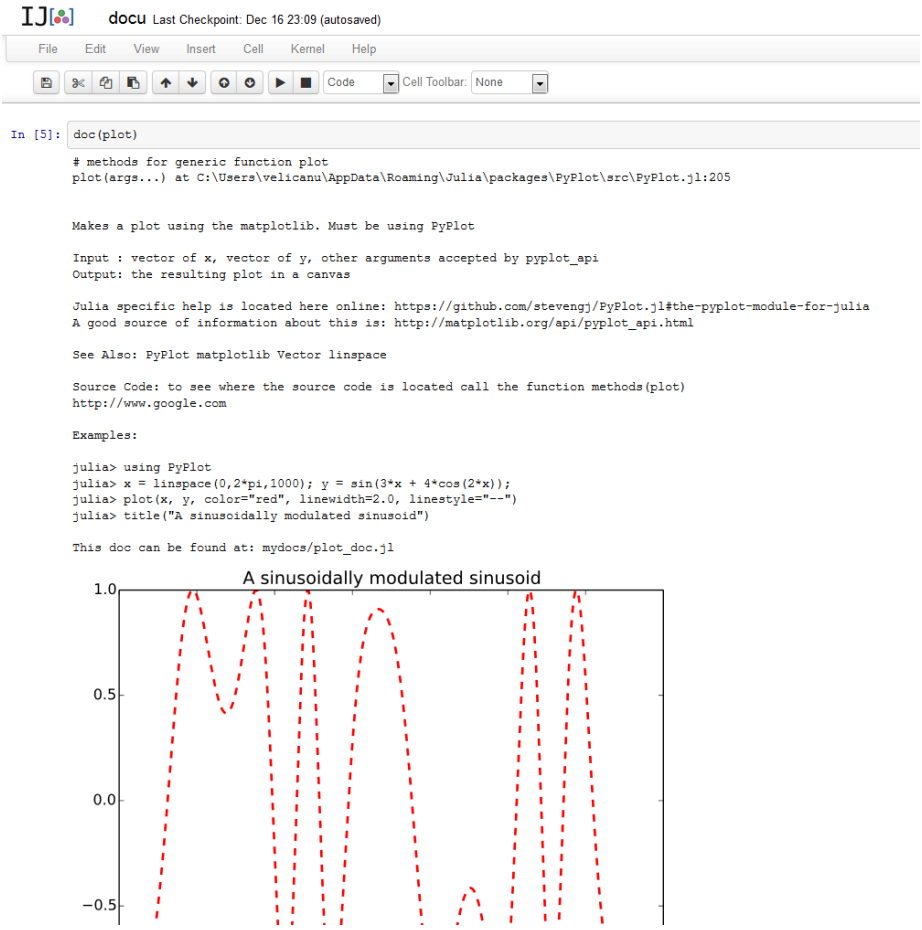


Figure 7: Documentation entry for plot as seen in IJulia.

3 Issues / Future work

While it is very neat to have working examples in the documentation that run when you call help, it could also be very dangerous at the same time. For instance, if the documentation for addprocs runs as an example addprocs(4) , when run by the user seeking that help processes could be added without his knowledge when that could be contrary to what he wants. Also if the help for plot is called for instance and if the user already has some plots displayed that he was working on, the documentation plots would erase his then current work which could be frustrating. These problems show the importance of writing "safe" documentation when all documentation are executable macros. Thus when writing the documentation for state-changing functions like addprocs or plot one should be careful to just print, and not execute, the examples which

could be disruptive to the user seeking help. Another perhaps more elegant solution would be to run the documentation macro in a sandboxed shell that doesn't affect ones environment. This could perhaps be implemented in docwrite or doc while leaving the rest of the framework as well as currently written documenation macros intact.

Another criticism that I received for my project was that my documentation function has a set structure that is not very flexible. It is inflexible on two levels, one that its structure is unchanging and centrally determined i.e. {definition, links, examples etc} and secondly that it's a .jl file which my not scale in the way we wish to user interfaces beyond the terminal and IJulia it was designed for. The formatting should be decoupled from the content of the documentation since different places that use Julia can have different ways to ideally display the content especially for ways we haven't imagined yet i.e. beyond the terminal and IJulia. The current system doesn't allow for that. It is also hard to make changes to the current documentation system without rewriting the current documentation files. Thus it will also be hard to extract the content from the current documentation system to be used in a different interface since one would effectively have to parse the .jl files to retrieve the content.

As a technical note of improvement, the examples need better quotation parsing since passing an example that takes a "string" as input will run into errors when trying to println the quotes and currently have to be escaped by hand. Additionally the text color formatting that is seen in the terminal is absent when used in IJulia, so a future step would be to either modify print_with_color to also work in IJulia in addition to the terminal, or to modify the documentation system to do this. It would also be very nice if a way is found to make the URL links clickable in IJulia

Ultimately one should decouple the content of the documentation from how to display the documentation. A very good proposal on how to do this has been discussed here [2] . Doing so would create a documentation system that is lasting and reusable since ways we interact with the code may evolve and the way to display documentation can evolve with them while the content itself can remain static and reusable.

Despite all these issues that writing a state-of the art documentation which is flexible as well as unbreakable will take a lot of time as well as work by many developers. In the meantime, the system of documentation I have developed is a good starting point for the accumulation of content. From my experience using Julia, I have realized that very little useful documentation exists and I found myself looking at the source code every time even for trivial things like how to set arbitrary precision in Julia. In order to make a good documentation system we need content in addition to functionality. My system is a very good way to collect a lot of content since it is so simple to use, documentation can be added very quickly by any user, and the system is already functional.

4 Conclusions

Julia is an extremely versitile and powerful language that is easy to understand and learn from simple examples. The current documentation is very limited, which is probably good enough for experienced Julia users or experienced coders used to styles similar to Julia when seeking help, but is a major roadblock to new users or novice programmers seeking to learn the language due to the lack of easily accessible examples. I have designed and implemented a useful help template for Julia, inspired by documentation seen in Matlab and Mathematica, in a backwards compatible way that is full of examples for users to learn from. I have documented a few functions in this framework (abs, plot, and the everything I wrote: doc, docwrite, docaddex).

While my implementation is not the final solution to providing the best documentation system for Julia, it does offer significant improvements over the current help functionality and can be an effective solution until the real documentation system is written. Since Julia is becoming easier to set up and is used by more and more MIT students and other aspiring coders, some trying programming for the first time, it's important to have a working system of useful documentation filled with examples before they become too frustrated with the learning steps. I have no doubt the final system of documentation will be much better than what I have put together and this work will certainly become obsolete at that point. However that system is still being talked about and no one is writing the code to get it working, it could be a long time before it gets done.

In conclusion, until the perfect documenation system designed to stand the test of time for Julia's long life ahead is created by the experts, the simple system shown here that vastly adds in quality to the current help functionality, can, and perhaps should, be used, so that the beginners like myself have something to guide us when we need it most. This system is publicly available on Github [3] and is extremely easy to expand the database of documented functions and to add to any Julia version on any computer.

References

[1] "Julia Documentation." Julia Documentation — Julia Language 0.3.0-dev Documentation. Web. 17 Dec. 2013. <http://docs.julialang.org/en/latest/>.

[2] "Associating Data with Functions, Modules, and Globals - Comment Thread." GitHub. Web. 23 Nov. 2013. <https://github.com/JuliaLang/julia/issues/3988#issuecomment-29141667>.

[3] Velicanu, Dragos. "Julia Documentation Project Github." GitHub. Web. 17 Dec. 2013. <https://github.com/velicanu/doc_18337_final_project>.

[4] Johnson, Steven G. “The PyPlot Module for Julia.” GitHub. Web. 17 Dec. 2013. <https://github.com/stevengj/PyPlot.jl>.