(1) Overview

Title

Plots.jl – a user extendable plotting API for the julia programming language

Paper Authors

1. Christ, Simon;
2. Schwabeneder, Daniel;
3. Rackauckas, Christopher;
4. Borregaard, Michael Krabbe;
5. Breloff, Thomas

Paper Author Roles and Affiliations

1. Leibniz Universität Hannover
2. TU Wien
3. Massachusetts Institute of Technology
4. Center for Macroecology, Evolution and Climate, Globe Institute, University of Copenhagen
5. Headlands Technologies

Abstract

There are plenty of excellent plotting libraries. Each excels at a different use case: one is good for printed 2D publication figures, the other at interactive 3D graphics, a third has excellent \texttt{LaTeX} integration or is good for creating dashboards on the web.

The aim of Plots.jl is to enable the user to use the same syntax to interact with many different plotting libraries, such that it is possible to change the library "backend" without needing to touch the code that creates the content – and without having to learn yet another application programming interface (API).

This is achieved by the separation of the plot specification from the implementation of the actual graphical backend. These plot specifications may be extended by a "recipe" system, which allows package authors and users to define how to plot any new type (be it a statistical model, a map, a phylogenetic tree or the solution to a system of differential equations) and create new types of plots - without depending on the Plots.jl package. This supports a modular ecosystem structure for plotting and yields a high reuse potential across the entire julia package ecosystem. Plots.jl is publicly available at https://github.com/JuliaPlots/Plots.jl.
Keywords

visualization; julia; plotting; julia-language; user-extendable

Introduction

Julia is a programming language that achieves high performance and stellar modularity and composability by making use of multiple dispatch and just-in-time compilation. This comes at the cost of increased latency as the language compiles new machine-code the first time any function is called on new types of arguments. This is notoriously an issue for packages that call a large part of their codebase in the first call, such as plotting packages. It even coined the term "time to first plot" as a phrase for julia’s start-up latency. Indeed, the julia language survey 2020 identified "it takes too long to generate the first plot" as the biggest problem faced by Julia users.

Package authors try to minimize loading time by reducing the number of dependencies, in particular those with long loading times themselves. Thus, authors are faced with a challenge if they want to define new plotting functionality for their packages; e.g. if a package for differential equations wishes to make it possible for users to investigate different solutions visually. Depending on a plotting package drastically increases startup times, as well as limiting users to that particular plotting package (which may conflict with other plotting packages used by the project). As such, depending on plotting packages is rarely seen in the julia ecosystem.

Plots.jl has solved this problem, by introducing plotting "recipes", which allow package authors to only depend on a very lightweight package RecipesBase.jl instead of depending on Plots.jl. This package has no other effect than making specialized syntax available for the code author to define visualizations; but otherwise has no effect, until the package end user loads Plots.jl directly. Thus, Plots.jl offers a unified and powerful API with a convenient way for package authors to support visualizations for multiple plotting packages, without increasing the loading time of their package – with the definition of a single recipe. An example can be seen in listing 5.

Development

Plots.jl was created by Tom Breloff between September 2015 and 2017, with the goal of creating a plotting API for the julia language, that was powerful, intuitive, concise, flexible, lightweight and smart. In particular the recipe system helped the package gain large traction within the community, as the latency of loading large dependencies was generally recognized as one of the major factors limiting the uptake of Julia.

With time Tom moved on, and the development of Plots.jl was continued by Michael K. Borregaard and Daniel Schwabeneder. The maintenance of the project is now a joint effort of the julia community. The package has reached a very high uptake in the ecosystem. In the Julia Language Survey of both 2019 and 2020, Plots.jl was identified as the julia community’s favorite package across the entire ecosystem, with 47 percent of all julia users listing it among their favorite packages.
Usage

Plots.jl is used for visualizations in scientific publications of different fields, such as numerics, mathematics, biology, ecology and geology as well as for teaching purposes.

Many packages in the Julia ecosystem, as well as non-packaged code (e.g. for scientific projects and publications) contain Plots.jl recipes. According to recent download statistics, Plots.jl has between 500 and 2000 downloads per day, and >300 published packages in the general package registry of Julia currently have recipes for Plots.jl defined.

Comparison

Plots.jl achieves its functionality by leveraging the multiple dispatch paradigm of Julia, which allows the user to define multiple methods for the same function, with the compiler selecting the appropriate method based on the types of the input arguments. Because of the close connection to Julia’s multiple dispatch, it’s approach to plotting is fairly unique.

In Python, the library unified-plotting shares the aim of providing a unified API for multiple packages, in this case matplotlib, pyplot and javascript libraries including d3.js. However, unified-plotting is still in the beta phase and not widely used.

The authors are not aware of other package ecosystems that have a recipe system akin to that of Plots.jl, though a recipe system inspired by that of Plots.jl is presently being implemented for the Julia library Makie.jl.

Implementation and architecture

One-function API

A central design goal of Plots.jl is that the user should rarely have to consult the documentation while plotting. This is achieved by having a tightly unified syntax. Plots.jl’s main interface is simply the `plot` function, which creates a new plot object. Additionally there is the `plot!` function to modify an existing plot object, e.g. by changing axes limits or adding new elements. Any type of predefined plot (e.g. a histogram, bar plot, scatter plot, a heatmap, an image, a geographical map etc.), may be created by a call to `plot` - the exact type is defined by the keyword argument `seriesspec` and the input arguments (type and number). New seriestypes can be created with recipes (see below).

For convenience, Plots.jl also exports "shorthand" functions named after the seriestypes (see examples in listing 1).

---

1Technically the API consists of more than one function, but the vast majority is `plot/plot!` and aliases thereof.
Analysis of the iris dataset

Distribution of sepal lengths per species

Figure 1: Example plot of the iris dataset[37] to illustrate the use of different attribute types (cf. listing 2).

Listing 1: Examples of shorthands. Full list available at https://docs.juliaplots/stable/api/#Plot-specification.

```
boxplot(args...; kwargs...) = plot(args...; seriestype = :boxplot, kwargs...)
scatter(args...; kwargs...) = plot(args...; seriestype = :scatter, kwargs...)
```

All aspects of the plot are controlled by a set of plot attributes, that are controlled by keyword arguments[26]. Plots.jl distinguishes four hierarchical levels of attributes: plot attributes, subplot attributes, axis attributes and series attributes (cf. fig. 1).

Listing 2: Code corresponding to fig. 1.

```
using StatsPlots
import RDatasets
pgfplotsx() # switching to the PGFPlotsX backend
iris = RDatasets.dataset("datasets", "iris") # loading the iris data as a DataFrame
default(palette=:seaborn_colorblind) # change the default color palette
plot(
    @df(iris, scatter(
        :SepalLength,
        :SepalWidth,
        group=:Species,
        title="Sepal length vs. width",
        xlabel="Length",
        seriestype = :scatter,
    ),
    @df(iris, boxplot(
        :SepalLength,
        :SepalWidth,
        group=:Species,
        title="Sepal length vs. width",
        xlabel="Length",
        seriestype = :boxplot,
    ),
)
```


A series in a Plots.jl context is an individual plot element, such as a continuous line or a set of scatter points. A plot may contain multiple series, e.g. when adding a trend line to a scatter plot. Multiple series may be added in the same plot call by concatenating the data as columns in a row matrix (see below).

Input arguments can have many different forms like:

| Function | Description |
|----------|-------------|
| plot()   | # empty Plot with axes |
| plot(4)  | # initialize a Plot with 4 empty series |
| plot(rand(10)) | # 1 series... x = 1:10 |
| plot(rand(10,5), rand(10)) | # 5 series... y is the same for all |
| plot(sin, rand(10)) | # y = sin.(x) |
| plot((sin,cos), 0, pi) | # sin and cos lines on the range [0, pi] |
| plot(1:10, Any[rand(10), sin]) | # using an automatic adaptive grid |
| plot( plot(rand(10)), plot(rand(10)) ) | # 2 series, y is rand(10) and sin.(x) |
| @df dataset("Ecdat", "Airline") plot(:Cost) | # a layout with two equally sized subplots |
| @df iris, boxplot{:Species, :SepalLength, group=:Species, ylabel="Length", title = "Distribution of sepal lengths per species", xtickfontsize = 11, legend = false, legend=false}, plot_title="Visualization of the iris dataset", right_margin=1Plots.cm, size=(900, 600) | # save the last current Plot object as a pdf-file |

Calling the plot function returns a Plot object. The Plot object is essentially a big nested dictionary holding the plot attributes for the layout, subplots, series, segments, etc. and their values. The plot object is automatically rendered in the surrounding context when returned to an interactive session, or can be displayed explicitly by calling the display function on the object. This delayed rendering means that plot calls can be combined without unnecessary intermediate rendering.

**Pipeline**

The plotting pipeline mainly has two stages (cf. fig. 2): construction of the plot using plot/plot! calls and creating the output via savefig/display/gui calls. These calls are often called implicitly in environments like the julia REPL, notebooks or IDEs.
The very first step upon construction is to convert all inputs to form the list of plot attributes that constitute the plot specification. As shown in listing 3 Plots.jl is very flexible about possible input values. The conversion step involves defining values for all attributes based on the values input as keyword arguments. This includes replacing "aliases" of attributes (which are multiple alternatively spelled keywords, such as 'c' or 'color', encoding the same attribute), handling of missing and nothing values in the input data and attribute values, and determining the final values based on the set of defaults. The default values are organized in a hierarchical framework, based on the values of other attributes; e.g. linecolor, fillcolor and markercolor will default to seriescolor under most seriestypes. But, for instance, under the bar seriestype, linecolor will default to :black, giving bars with a black border. This allows the specification of useful plots with a minimum of specification, in contrast to the paradigm of e.g. matplotlib, where every aspect of the plot is usually defined manually by the user.

Listing 3: Examples of input preprocessing steps in Plots.jl. All these calls are equivalent.

```julia
plot(2:4, c = :steelblue) # c is the shortest alias for seriescolor
plot([2,3,4], color = 1) # :steelblue is the first color of the default palette
plot(1:3, [2,3,4], colour = :auto) # the recipe for a single input will use 1:3 as x-values
plot(1:3, [2,3,4], seriescolors = 1) # you can use singular
plot([1,2,3], [2,3,4], seriescolor = RGBA{Float64}(0.275,0.51,0.706,1.0)) # or plural version of attributes
# this is the fully expanded call
```

Afterwards recipes are applied recursively and the Plot and Subplot objects are initialized. Recipes will be explained in detail in the next section.

When an output is to be produced the layout will be computed and the backend-specific code will be executed to produce the result.
Figure 2: Plotting pipeline in Plots.jl. The separation of construction and output production enables the flexible use of different backends in the same session and helps to avoid unnecessary intermediate calculation. Created using mermaid[1].

Recipes

As mentioned in the introduction, recipes are the key mechanism in the Plots.jl pipeline to allow composable definitions of visualisations across julia packages. The composable definitions may be applied recursively, which is a major advancement for improving ecosystem support by giving a combinatoric reduction
in the amount of code required for downstream libraries to add native plotting support for their types.

Plots.jl distinguishes four types of recipes: user recipes, type recipes, plot recipes and series recipes [20]. User recipes (which define how to plot objects of a certain type) and series recipes (which define a new seriestype) are by far the most commonly used. All of them can be constructed with the @recipe macro which acts on a function definition. The type of the recipe is then determined by the signature of that function, utilizing the multiple dispatch capabilities of the julia programming language.

Listing 4: Recipe signatures

```julia
using RecipesBase
struct CustomStruct end
@recipe function f(arg::CustomStruct; custom_kw = 1) # user recipe
end
@recipe function f(::Type{CustomStruct}, val::CustomStruct) # type recipe
end
@recipe function f(::Type{Val{:recipename}}, plt::AbstractPlot) # plot recipe
end
@recipe function f(::Type{Val{:recipename}}, x, y, z) # series recipe
end
```

It is enough to depend on the RecipesBase.jl package, a small and lightweight dependency to define a recipe.

The major question with recipes is how this is an improvement over previous designs. For example, in most plotting libraries such as matplotlib[21], a downstream ODE solver library can add a new function plotSolution that will plot an ODE solution. However, the difference, and the major technological advance of the Plots.jl recipe system, is that the application of recipes is recursive and extendable via multiple dispatch. This solves a combinatoric problem for downstream support: it is possible to combine and chain recipes to support plotting on new combinations of input types without ever defining a recipe for that specific combination.

To illustrate this, consider the example of combining the recipes defined by the julia packages DifferentialEquations.jl[33] and Measurements.jl[16] (cf. fig. 3 and listing 6). In this example, a user solves a differential equation with uncertain initial conditions specified by Measurements.Measurement objects. The uncertainty encoded in the Measurement objects are automatically propagated through the ODE solver, as multiple methods for this type have been defined for the arithmetic functions. The resulting ODE solution `sol` is then already specified in terms of such Measurements.Measurements. When running the plot command `plot(sol)`, the recipe for ODE solvers will transform the ODESolution object into an array of arrays, each representing a time series to plot (using techniques like dense output to produce a continuous looking solution). This array of arrays contains number types matching the state of the solution, in this case Measurements.Measurements. Successive applications of the user recipe defined in Measurements.jl then take each state value and assign the uncertainty part of the state to the yerror attribute and pass the value part of the state to the next recipe. When used with the initial seriestype :scatter this results
in a scatter plot with proper error bars as seen in fig. 3.

Therefore, while the two packages were not developed to work together, multiple dispatch allows to efficiently solve problems containing combinations of these packages, and the Plots.jl recipe system allows the combined visualization to work automatically.

The recipe of Measurements.jl is an example of a particularly short recipe. A Measurements.Measurement is represented as a type with two fields: value and uncertainty. It can be conveniently constructed with the Unicode infix operator ±. Thus the object \( a \pm b \) has \( a \) as the value and \( b \) as the uncertainty. An array of measurement values can be converted into an array of floating point values to plot, along with having the uncertainties as error bars, via the following recipe:

Listing 5: Measurements.jl recipe

```julia
@recipe function f(x::AbstractArray, y::AbstractArray{<:Measurement})
yerror := uncertainty.(y) # := is special syntax in the @recipe block which
# sets an attribute overriding
# any present value. The alternative syntax
# is --> to give call-site values precedence.
x, value.(y)
end
```

9
Figure 3: Showcase of composing recipes. Plotting a ODESolution object from DifferentialEquations.jl containing Measurements from Measurements.jl will apply the recipe of DifferentialEquations.jl which will return vectors of Measurements, which will apply the recipe from Measurements.jl; yielding the solutions of the Lotka-Volterra system[2] with correct error bounds without the user having to change the callsite. Neither of these packages has code in their recipes for handling types of the other package. Full code available in listing 6.
Structure and interfaces

Figure 4: Overview of the Plots.jl ecosystem and its interfaces with other Julia packages. The numbers of dependents are taken from juliahub[30].

The code for Plots.jl is not located in one repository, but split into a few packages, to enhance reuse of more general parts of the code by other packages (cf. fig. 4). In the following the different packages and their use cases will be described.

**Plots.jl**: The main user facing package. Defines all default values and holds the code for layouting, conversion of input arguments, output generation, all backend code and the default recipes. This is the repository with the highest rate of change.

**StatsPlots.jl**: A drop-in replacement for Plots.jl, meaning it loads and reexports all of Plots.jl and adds recipes that are specially targeted at visualisation of statistical data (aiming to be integrated with Julia’s statistical package ecosystem under the JuliaStats organisation). Therefore it has more dependencies than Plots.jl which increases the loading time and since not all users need this functionality it is separated in its own repository.

**PlotUtils.jl**: Provides general utility routines, such as handling colors, optimizing ticks or function sampling. This package is also used by e.g. the newer plotting package Makie.jl.

**RecipesBase.jl**: A package with zero 3rd-party dependencies, that can be used by other packages to define recipes for their own types without needing to
depend on Plots.jl.

RecipesPipeline.jl: Another lightweight package that defines an API such that other plotting packages can consume recipes from RecipesBase.jl without needing to become a backend of Plots.jl.

GraphRecipes.jl: A package that provides recipes for visualisation of graphs in the sense of graph theory. These are also split out because they have some heavy dependencies.

PlotThemes.jl: Provides different themes for Plots.jl.

PlotDocs.jl: Hosts the documentation of Plots.jl.

Backends

Plots.jl currently supports seven plotting frameworks as backends. Typically these plotting frameworks themselves have different graphic libraries as backends to support different output types. The backends of Plots.jl differ in their area of expertise and have different trade-offs.

GR: The default backend. Uses the GR framework[18]. It is among the fastest backends with a good coverage of functionality.

Plotly/PlotlyJS: Is the backend with the most interactivity and best web support using the plotly javascript library[29]. One use case is to create interactive plots in documentation[31] or notebooks. The Plotly backend is a version with minimal dependencies, which doesn’t require the user to load any other julia package and displays its graphics in the browser, while PlotlyJS requires the user to load PlotlyJS.jl, but offers display of plots in a standalone window.

PyPlot: PyPlot.jl is the julia wrapper of matplotlib[21] and covers a lot of functionality at moderate speed.

PGFPlotsX: Uses the pgfplots \LaTeX package[28] and is thus the slowest of the backends, but integrates very good with \LaTeX-documents.

InspectDR: Fast backend with GUI and some interactivity that does good for 2D and handles large datasets and high refresh rates[25].

UnicodePlots: A backend that allows plotting in the terminal with unicode characters and can be used in a terminal (also on headless machines)[38]. Therefore it lacks a lot of functionality compared to the other backends.
**HDF5:** A backend that can be used to save the `Plot` object along the data in a hdf5-file using HDF5.jl[19], such that it can be recovered with any back-end. Potentially allows interfacing with Plots.jl from other programming languages.

Furthermore there are 6 deprecated backends that were used in the earlier stages of Plots.jl, but which are no longer maintained and the Gaston.jl backend which is in an early experimental stage. Gaston.jl is a julia interface for gnuplot[17]. This shows that Plots.jl can be sustained even if a maintainer of backend code leaves. Either the backend will be maintained by the community or it will be replaced by another backend.

**Quality control**

Plots.jl runs its unit tests of all backends as well as visual regression tests of the default backend against the latest version of macOS, Ubuntu and Windows using the current stable version of julia, the long term support version and the nightly version on every pull request and pushes to the default branch. Furthermore benchmarks are run to detect performance regressions. Lastly, building the documentation creates a suite of example plots for every backend, which would also detect certain errors.

**(2) Availability**

**Operating system**

Plots.jl is tested on Windows, Linux and macOS.

**Programming language**

julia 1.5

**Additional system requirements**

**Dependencies**

Plots.jl has the following direct dependencies:

- Contour.jl v0.5
- FFMPEG.jl v0.2 - v0.4
- FixedPointNumbers v0.6 - v0.8
- GR.jl v0.46 - v0.55, v0.57
- GeometryBasics.jl v0.2, v0.3.1 - v0.3
JSON.jl v0.21, v1
Latexify.jl v0.14 - v0.15
Measures.jl v0.3
NaNMath.jl v0.3
PlotThemes.jl v2
PlotUtils.jl v1
RecipesBase.jl v1
RecipesPipeline.jl v0.3
Reexport.jl v0.2, v1
Requires.jl v1
Scratch.jl v1
Showoff.jl v0.3.1 - v0.3, v1
StatsBase.jl v0.32 - v0.33

In addition it has 125 indirect dependencies all of which can be seen at [30].

List of contributors

![Code ownership over time of Plots.jl]

Figure 5: Lines of code alive of the top ten contributors of the Plots.jl repository over time. Data created with Hercules[36].
Code ownership over time of the Plots.jl ecosystem

![Graph showing code ownership over time of the Plots.jl ecosystem]

Figure 6: Lines of code alive of the top ten contributors of the Plots.jl ecosystem (fig. 4) over time. Data created with hercules[36].

Table 1: Contributors sorted by number of commits.

| name                        | affiliation                                      | role               | orcid               |
|-----------------------------|--------------------------------------------------|--------------------|---------------------|
| Tom Breloff                 | Headlands Technologies                           | Creator            | missing             |
| Daniel Schwabeneder         | TU Wien                                          | ProjectLeader      | 0000-0002-0412-0777|
| Michael Krabbe Borregaard   | GLOBE Institute, University of Copenhagen        | ProjectLeader      | 0000-0002-8146-8435|
| Simon Christ                | Leibniz Universität Hannover                    | ProjectLeader      | 0000-0002-5866-1472|
| Josef Heinen                | Forschungszentrum Jülich                         | ProjectMember      | 0000-0001-6509-1925|
| Yuval                       | missing                                          | Other              | missing             |
| Andrew Palugniok            | missing                                          | ProjectMember      | missing             |
| Simon Danisch               | @beacon-biosignals                               | Other              | missing             |
| Pietro Vertechi             | Veos Digital (https://veos.digital/)            | ProjectMember      | missing             |
| Zhanibek Omarov             | Korea Advanced Inst. of Science and Technology (KAIST) | ProjectMember | 0000-0002-8783-8791|
| Thatcher Chamberlin         | missing                                          | Other              | missing             |
| @ma-laforge                 | missing                                          | ProjectMember      | missing             |
| Christopher Rackauckas      | Massachusetts Institute of Technology            | Other              | 0000-0001-5850-0663|
| Oliver Schulz               | Max Planck Institute for Physics                 | Other              | missing             |
| Sebastian Pfützner          | @JuliaComputing                                  | Other              | missing             |
| Takafulmi Arakaki           | missing                                          | Other              | missing             |
| Amin Yahyaabadi             | University of Manitoba                           | Other              | missing             |
| Jack Devine                 | missing                                          | Other              | missing             |
| Sebastian Pech              | missing                                          | Other              | missing             |
| name                        | affiliation                  | role  | orcid                              |
|-----------------------------|------------------------------|-------|------------------------------------|
| Patrick Kofod Mogensen     | @JuliaComputing              | Other | 0000-0002-4910-1932               |
| Samuel S. Watson            | missing                      | Other | missing                            |
| Naoki Saito                 | UC Davis                     | Other | 0000-0001-5234-4719               |
| Benoît Pasquier            | University of Southern       | Other | 0000-0002-3838-5976               |
|                                           | California (USC)             |       |                                    |
| Ronny Bergmann             | NTNU Trondheim               | Other | 0000-0001-8342-7218               |
| Andy Nowacki                | University of Leeds          | Other | 0000-0001-7669-7383               |
| Ian Butterworth            | missing                      | Other | missing                            |
| David Gustavsson           | Lund University              | Other | missing                            |
| Anshul Singhvi              | Columbia University          | Other | 0000-0001-6055-1291               |
| david-macmahon             | missing                      | Other | missing                            |
| Fredrik Ekre                | missing                      | Other | missing                            |
| Maaz Bin Tahir Saeed       | missing                      | Other | missing                            |
| Kristoffer Carlsson        | missing                      | Other | missing                            |
| Will Kearney               | missing                      | Other | missing                            |
| Niklas Korsbo              | missing                      | Other | missing                            |
| Miles Lucas                | missing                      | Other | missing                            |
| @Godisemo                  | missing                      | Other | missing                            |
| Florian Oswald             | missing                      | Other | missing                            |
| Diego Javier Zea           | missing                      | Other | missing                            |
| @WillRam                   | missing                      | Other | missing                            |
| Fedor Bezrukov             | missing                      | Other | missing                            |
| Spencer Lyon               | missing                      | Other | missing                            |
| Darwin Darakananda         | missing                      | Other | missing                            |
| Lukas Hauertmann           | missing                      | Other | missing                            |
| Huckleberry Febbo          | missing                      | Other | missing                            |
| @H-M-H                     | missing                      | Other | missing                            |
| Josh Day                   | missing                      | Other | missing                            |
| @wfgra                     | missing                      | Other | missing                            |
| Sheehan Olver              | missing                      | Other | missing                            |
| Jerry Ling                 | missing                      | Other | missing                            |
| Jks Liu                    | missing                      | Other | missing                            |
| Seth Axen                  | missing                      | Other | missing                            |
| @o01eg                     | missing                      | Other | missing                            |
| Sebastian Micluta-Câmpeanu | missing                      | Other | missing                            |
| Tim Holy                   | missing                      | Other | missing                            |
| Tony Kelman                | missing                      | Other | missing                            |
| Antoine Levitt             | missing                      | Other | missing                            |
| Iblis Lin                  | missing                      | Other | missing                            |
| Harry Scholes              | missing                      | Other | missing                            |
| @djsegal                   | missing                      | Other | missing                            |
| Goran Nakerst              | missing                      | Other | missing                            |
| Felix Hagemann             | missing                      | Other | missing                            |
| Matthieu Gomez             | missing                      | Other | missing                            |
| @biggsbiggsby              | missing                      | Other | missing                            |
| Jonathan Anderson           | missing                      | Other | missing                            |
| Michael Kraus              | missing                      | Other | missing                            |
| name                  | affiliation          | role       | orcid                  |
|-----------------------|----------------------|------------|------------------------|
| Carlo Lucibello       | missing              | Other      | missing                |
| Robin Deits           | missing              | Other      | missing                |
| Misha Mkhasenko       | missing              | Other      | missing                |
| Benoît Legat          | missing              | Other      | missing                |
| Steven G. Johnson     | missing              | Other      | missing                |
| John Verzani          | missing              | Other      | missing                |
| Mattias Fält          | missing              | Other      | missing                |
| Rashika Karki         | missing              | Other      | missing                |
| Morten Pībeleht       | missing              | Other      | missing                |
| Filippo Vicentini     | missing              | Other      | missing                |
| David Anthoff         | missing              | Other      | missing                |
| Leon Wabeke           | missing              | Other      | missing                |
| Yusuke Kominami       | missing              | Other      | missing                |
| Oscar Dowson          | missing              | Other      | missing                |
| Max G                 | missing              | Other      | missing                |
| Fabian Greimel       | missing              | Other      | missing                |
| Jérémy                | missing              | Other      | missing                |
| Pearl Li              | missing              | Other      | missing                |
| David P. Sanders      | missing              | Other      | missing                |
| Asbjørn Nilsen Riseth | missing              | Other      | missing                |
| Jan Weidner           | missing              | Other      | missing                |
| @jakkor2              | missing              | Other      | missing                |
| Pablo Zubieta         | missing              | Other      | missing                |
| Hamza Yusuf Çakır     | missing              | Other      | missing                |
| John Rinhardt         | missing              | Other      | missing                |
| Martin Biel           | missing              | Other      | missing                |
| Moritz Schauer        | missing              | Other      | missing                |
| Mosè Giodano          | missing              | Other      | missing                |
| @olesgshch            | missing              | Other      | missing                |
| Leon Shen             | missing              | Other      | missing                |
| Jeff Fessler          | missing              | Other      | missing                |
| @hustf                | missing              | Other      | missing                |
| Asim H Dar            | missing              | Other      | missing                |
| @8uurg                | missing              | Other      | missing                |
| Abel Siqueira         | missing              | Other      | missing                |
| Adrian Dawid          | missing              | Other      | missing                |
| Alberto Lusiani       | missing              | Other      | missing                |
| Balázs Mezei          | missing              | Other      | missing                |
| Ben Ide               | missing              | Other      | missing                |
| Benjamin Lungwitz     | missing              | Other      | missing                |
| Bernd Riederer        | University of Graz   | Other      | 0000-0001-8390-0087   |
| Christina Lee         | missing              | Other      | missing                |
| Christof Stocker      | missing              | Other      | missing                |
| Christoph Finkensiep  | missing              | Other      | missing                |
| @Cornelius-G          | missing              | Other      | missing                |
| Daniel Hoegh          | missing              | Other      | missing                |
| Denny Biasiolli       | missing              | Other      | missing                |
| Dieter Castel         | missing              | Other      | missing                |
| name                          | affiliation     | role   | orcid            |
|-------------------------------|-----------------|--------|------------------|
| Elliot Saba                  | missing         | Other  | missing          |
| Fengyang Wang                | missing         | Other  | missing          |
| Fons van der Plas            | missing         | Other  | missing          |
| Fredrik Bagge Carlson        | missing         | Other  | missing          |
| Graham Smith                 | missing         | Other  | missing          |
| Hayato Ikoma                 | missing         | Other  | missing          |
| Hessam Mehr                  | missing         | Other  | missing          |
| @InfiniteChai               | missing         | Other  | missing          |
| Jack Dunn                    | missing         | Other  | missing          |
| Jeff Bezanson                | missing         | Other  | missing          |
| Jeff Eldredge                | missing         | Other  | missing          |
| Jinay Jain                   | missing         | Other  | missing          |
| Johan Bläbäck               | missing         | Other  | missing          |
| @jmert                       | missing         | Other  | missing          |
| Lakshya Khatri               | missing         | Other  | missing          |
| Lia Siegelmann               | missing         | Other  | missing          |
| @marekkukan-tw              | missing         | Other  | missing          |
| Mauro Werder                 | ETH Zurich      | Other  | 0000-0003-0137-9377 |
| Maxim Grechkin              | missing         | Other  | missing          |
| Michael Cawte                | missing         | Other  | missing          |
| @milesfrain                 | missing         | Other  | missing          |
| Nicholas Bauer               | missing         | Other  | missing          |
| Nicolau Leal Werneck         | missing         | Other  | missing          |
| @nilshg                     | missing         | Other  | missing          |
| Oliver Evans                 | missing         | Other  | missing          |
| Peter Gagarinov              | missing         | Other  | missing          |
| Páll Haraldsson              | missing         | Other  | missing          |
| Rik Huijzer                  | missing         | Other  | missing          |
| Romain Franconville          | missing         | Other  | missing          |
| Ronan Pigott                 | missing         | Other  | missing          |
| Roshan Shariff               | missing         | Other  | missing          |
| Scott Thomas                 | missing         | Other  | missing          |
| Sebastian Rollén             | missing         | Other  | missing          |
| Seth Bromberger              | missing         | Other  | missing          |
| Siva Swaminathan             | missing         | Other  | missing          |
| Tim DuBois                   | missing         | Other  | missing          |
| Travis DePrato               | missing         | Other  | missing          |
| Will Thompson                | missing         | Other  | missing          |
| Yakir Luc Gagnon             | missing         | Other  | missing          |
| Benjamin Chislett            | missing         | Other  | missing          |
| @hhaenscl                    | missing         | Other  | missing          |
| @improbable22                | missing         | Other  | missing          |
| Johannes Fleck               | missing         | Other  | missing          |
| Peter Czaban                 | missing         | Other  | missing          |
| @innerlee                    | missing         | Other  | missing          |
| Mats Cronqvist               | missing         | Other  | missing          |
| Shi Pengcheng                | missing         | Other  | missing          |
| @wg030                       | missing         | Other  | missing          |
The code for creating tables and figures is publicly available at https://
gitlab.uni-hannover.de/comp-bio/manuscripts/plots-paper.

Software location:

**Code repository** Github

Name: JuliaPlots/Plots.jl

Persistent identifier: https://doi.org/10.5281/zenodo.4725318

Licence: MIT

Version published: 1.13.2

Date published: 28/04/2021

The first version of Plots.jl was published on github at 11/09/2015.

**Language**

julia

(3) Reuse potential

Plots.jl can be used by people working in all fields for data visualization. In particular it is possible to define backend agnostic recipes for their domain specific data structures with minimal dependencies. These can be shared, reused and extended by peers with ease by including these recipes in their packages or published scripts. Also it is possible for other plotting software with julia bindings to take advantage of the recipe system either by contributing backend code to Plots.jl or by using RecipesPipeline.jl to become an independent consumer of RecipesBase.jl's recipes. Plotting software without julia bindings could potentially use the HDF5 backend to consume fully processed and serialized recipe data.

People interested in modifying, extending or maintaining Plots.jl can get in contact either via the github issue tracker, the julia discourse forum or the julia slack and zulip spaces. There are quarterly maintenance calls that can be joined on request.

**Acknowledgements**

We like to acknowledge the support of the julia community and the numerous contributors that keep this project alive.


Funding statement

Competing interests

The authors have no competing interests to declare.

Code examples

Listing 6: Recipes showcase

```
using Plots, Measurements, OrdinaryDiffEq
pgfplotx() # change to pgfplotx backend

# define Lotka-Volterra equations
function f(du,u,p,t)
    du[1] = p[1]*u[1] - p[2]*u[1]*u[2] # prey
    du[2] = -p[3]*u[2] + p[4]*u[1]*u[2] # predator
end

u0 = [1.0 ± 0.1 ; 1.0 ± 0.1] # define initial conditions with uncertainty

# define start and end time

p = [1.5,1.0,3.0,1.0] # define vector of parameters

# define ODEProblem
prob = ODEProblem(f,u0,tspan,p)

# solve the problem using the Tsit5 integrator. Returns a ODESolution
sol = solve(prob, Tsit5())

# plot density is a keyword of the recipe defined in OrdinaryDiffEq
pl = scatter(sol, plotdensity = 75)

# save plot as pdf-file
savefig(pl, "DiffEq<3Measurements.pdf")

# return plot to display

```

References

[1] About Mermaid. URL: https://mermaid-js.github.io/mermaid/#/README (visited on 03/02/2022).
[2] Alfred J. Lotka. *Elements of Physical Biology*. In collab. with Indian Institute Of Science, IISc Library, and Jiju. Williams and Wilkins Company, 1925. 495 pp. URL: http://archive.org/details/elementsofphysic017171mbp (visited on 05/31/2022).
[3] Justin Angevaare, Zeny Feng, and Rob Deardon. *Infectious Disease Transmission Network Modelling with Julia*. Feb. 13, 2020.
[4] Fredrik Bagge Carlson. “MonteCarloMeasurements.Jl : Propagation of Distributions by Monte-Carlo Sampling; Real Number Types with Uncertainty Represented by Particle Clouds.” In: (2019). URL: http://lup.lub.lu.se/record/8ff6a743-0ad6-4d98-bbb3-5d549c698bc1 (visited on 05/03/2021).
[5] Jeff Bezanson et al. “Julia: A Fresh Approach to Numerical Computing”. In: *SIAM Rev.* 59.1 (Jan. 2017), pp. 65–98. ISSN: 0036-1445, 1095-7200. DOI: 10.1137/141000671.
[6] Kevin S. Bonham et al. “Microbiome.Jl and BiobakeryUtils.Jl - Julia Packages for Working with Microbial Community Data”. In: *Journal of Open Source Software* 6.67 (Nov. 17, 2021), p. 3876. issn: 2475-9066. doi: 10.21105/joss.03876.

[7] Mike Bostock. *D3.Js - Data-Driven Documents*. URL: https://d3js.org/ (visited on 03/02/2022).

[8] Stephen Boyd and Lieven Vandenberghe. *Introduction to Applied Linear Algebra: Vectors, Matrices, and Least Squares*. 1st ed. Cambridge University Press, June 7, 2018. isbn: 978-1-316-51896-0 978-1-108-58366-4. doi: 10.1017/9781108583664.

[9] Allen Caldwell et al. “BAT.Jl Upgrading the Bayesian Analysis Toolkit”. In: *EPJ Web Conf.* 245 (2020), p. 06001. issn: 2100-014X. doi: 10.1051/epjconf/202024506001.

[10] Navid Constantinou et al. “GeophysicalFlows.Jl: Solvers for Geophysical Fluid Dynamics Problems in Periodic Domains on CPUs GPUs”. In: *JOSS* 6.60 (Apr. 21, 2021), p. 3053. issn: 2475-9066. doi: 10.21105/joss.03053.

[11] Matija Čufar. “Ripserer.Jl: Flexible and Efficient Persistent Homology Computation in Julia”. In: *JOSS* 5.54 (Oct. 19, 2020), p. 2614. issn: 2475-9066. doi: 10.21105/joss.02614.

[12] Simon Danisch and Julius Krumbiegel. “Make.Jl: Flexible High-Performance Data Visualization for Julia”. In: *Journal of Open Source Software* 6.65 (Sept. 1, 2021), p. 3349. issn: 2475-9066. doi: 10.21105/joss.03349.

[13] Gabriel Dansereau and Timothée Poisot. “SimpleSDMLayers.Jl and GBIF.Jl: A Framework for Species Distribution Modeling in Julia”. In: *JOSS* 6.57 (Jan. 27, 2021), p. 2872. issn: 2475-9066. doi: 10.21105/joss.02872.

[14] Tobin Driscoll. “ComplexRegions.Jl: A Julia Package for Regions in the Complex Plane”. In: *JOSS* 4.44 (Dec. 2, 2019), p. 1811. issn: 2475-9066. doi: 10.21105/joss.01811.

[15] Jamie Fairbrother et al. “GaussianProcesses.Jl: A Nonparametric Bayes Package for the Julia Language”. June 30, 2019. arXiv: 1812.09064 [stat]. URL: http://arxiv.org/abs/1812.09064 (visited on 05/03/2021).

[16] Mosè Giordano. “Uncertainty Propagation with Functionally Correlated Quantities”. In: *ArXiv e-prints* (Oct. 2016). arXiv: 1610.08716 [physics.data-an].

[17] Gnuplot Homepage. URL: http://www.gnuplot.info/ (visited on 05/31/2022).

[18] Josef Heinen. *GR Framework — GR Framework 0.64.0 Documentation*. URL: https://gr-framework.org/index.html (visited on 03/03/2022).

[19] Home - HDF5.Jl. URL: https://juliaio.github.io/HDF5.jl/stable/ (visited on 03/03/2022).

[20] How Do Recipes Actually Work? URL: https://daschw.github.io/recipes/#what_are_recipes (visited on 03/02/2022).

[21] J. D. Hunter. “Matplotlib: A 2D Graphics Environment”. In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. doi: 10.1109/MCSE.2007.55.
[22] Introduction to Computational Thinking. URL: https://computationalthinking.mit.edu/Spring21/newton_method/ (visited on 05/27/2021).

[23] C. Brenhin Keller and T. Mark Harrison. “Constraining Crustal Silica on Ancient Earth”. In: Proceedings of the National Academy of Sciences 117.35 (Sept. 2020), pp. 21101–21107. DOI: 10.1073/pnas.2009431117.

[24] Michael Lindner et al. “NetworkDynamics.Jl – Composing and Simulating Complex Networks in Julia”. Mar. 26, 2021. arXiv: 2012.12696 [physics]. URL: http://arxiv.org/abs/2012.12696 (visited on 05/03/2021).

[25] ma-laforge. InspectDR.Jl: Fast, Interactive Plots. Mar. 3, 2022. URL: https://github.com/ma-laforge/InspectDR.jl (visited on 03/02/2022).

[26] Overview · Plots. URL: https://docs.juliaplots.org/latest/attributes/ (visited on 05/11/2021).

[27] Package Download Stats for Julia. URL: https://pkgs.genieframework.com/ (visited on 03/02/2022).

[28] PGFPlots - A LaTeX Package to Create Plots. URL: http://pgfplots.sourceforge.net/ (visited on 03/03/2022).

[29] Plotly JavaScript Graphing Library. URL: https://plotly.com/javascript/ (visited on 03/03/2022).

[30] Plots · JuliaHub. URL: https://juliahub.com/ui/Packages/Plots/1d3vC/1.13.27t=1 (visited on 05/11/2021).

[31] Plotting · SpectralDistances. URL: https://baggepinnen.github.io/SpectralDistances.jl/latest/plotting/ (visited on 05/27/2021).

[32] Christopher Rackauckas and Qing Nie. “DifferentialEquations.Jl – A Performant and Feature-Rich Ecosystem for Solving Differential Equations in Julia”. In: Journal of Open Research Software 5.1 (1 May 25, 2017), p. 15. ISSN: 2049-9647. DOI: 10.5334/jors.151.

[33] Christopher Rackauckas et al. SciML/DifferentialEquations.Jl: V7.1.0. Zenodo, Jan. 11, 2022. DOI: 10.5281/zenodo.5837925.

[34] Viral B. Shah and Andrew Claster. 2020 Julia User and Developer Survey. URL: https://julialang.org/blog/2020/08/2020-julia-user-and-developer-survey/ (visited on 05/30/2022).

[35] Viral B. Shah, Andrew Claster, and Abhijith C. Julia User - Developer Survey 2019. URL: https://julialang.org/blog/2019/08/2019-julia-survey/ (visited on 05/30/2022).

[36] Src-d/Hercules. source{d}, May 19, 2021. URL: https://github.com/src-d/hercules (visited on 05/19/2021).

[37] UCI Machine Learning Repository: Iris Data Set. URL: https://archive.ics.uci.edu/ml/datasets/Iris/ (visited on 04/07/2022).

[38] UnicodePlots. JuliaPlots, Mar. 3, 2022. URL: https://github.com/JuliaPlots/UnicodePlots.jl (visited on 03/03/2022).

[39] Unified Plotting — Unified-Plotting 0.5.0rc4 Documentation. URL: https://robert-haas.github.io/unified-plotting-docs/ (visited on 08/04/2021).