Tutorial III.V - Plotting Graphs

Applied Optimization with Julia

Introduction to Basic Plotting in Julia

This interactive Julia script introduces the basics of plotting in Julia using the Plots.jl package. It covers creating basic plots, customizing them to your needs, and saving them as files for further use. Follow the instructions, write your code in the designated code blocks, and validate your results with @assert statements.

Ensure you have the Plots package installed to follow along with this tutorial. If you followed the course so far, you first have to install the Plots Package before you can start using it:

```julia
import Pkg; Pkg.add("Plots")
```

```julia
using Plots
```

Section 1: Creating Basic Plots

The Plots.jl package simplifies the process of creating a wide array of plots, from simple line plots to complex 3D visualizations. The following illustrates the basic syntax of a line plot for some random data. First, we define a variable x with a range from 1 to 10 for the x axis. Then, we define y with 10 random numbers. For example:

```julia
x = 1:10
y = rand(10)
line_plot = plot(x, y, title="Simple Line Plot", xlabel="x axis", ylabel="y axis",
↪  legend=false)
display(line_plot)
```

Simple Line Plot
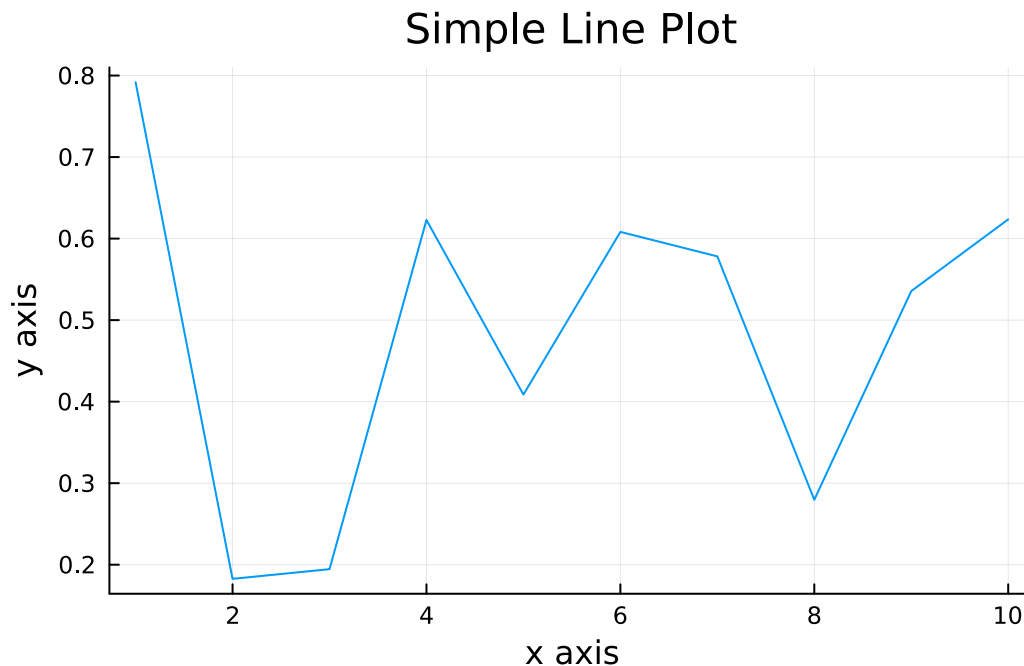
We call `plot()` and use x and y as main arguments. Furthermore, we have a variety of choices to add further information, such as the title, labels, a legend, colors, opacity and so on. More on this later.

Exercise 1.1

Create a scatter plot called `scatter_plot` for some random data on a range from 1 to 20. The arguments inside the functions can be the same as before, but we need to replace `plot` by `scatter`. Note, that the test are not so sophisticated here due to the structure of plots and their exploratory nature.

```
# Create a scatter plot called 'scatter_plot' for some random data on a range from 1 to
↳  20. The arguments inside the functions can be the same as before, but we need to
↳  replace 'plot' by 'scatter'. Note, that the test are not so sophisticated here due to
↳  the structure of plots and their exploratory nature.
# YOUR CODE BELOW

# Test your answer
@assert @isdefined scatter_plot
println("Simple scatter plot created successfully!")
display(scatter_plot)
```

Section 2: Customizing Plots

Plots.jl allows extensive customization options to tailor your plots. You can change colors, markers, line styles, and much more. For example, you could dash the line by adding `line=:dash`, change the width with `linewidth=3` and the color by adding `color=:red`.

Exercise 2.1

Customize a line plot and save the plot in the variable `custom_line_plot`.

```
# Customize a line plot and save the plot in the variable `custom_line_plot`.
# YOUR CODE BELOW

# Test your answer
@assert @isdefined custom_line_plot
println("Customized line plot created successfully!")
display(custom_line_plot)
```
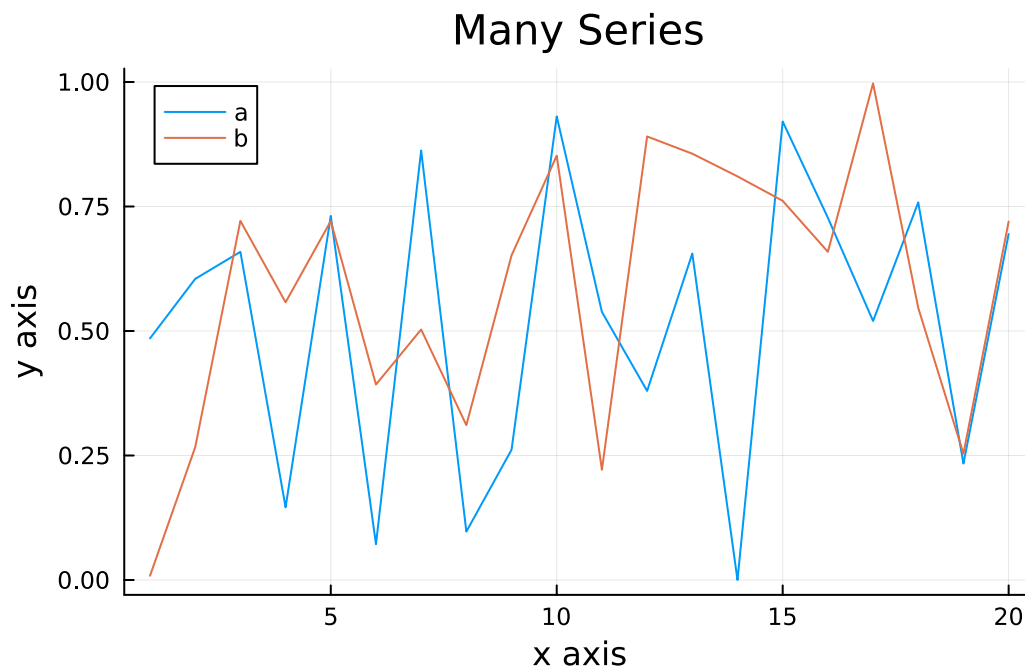
> **i** Note
>
> You can customize the appearance of a random line plot as you like.

Add multiple series to a plot. Instead of one series, we can also add multiple series. This is done, for example, by defining one plot (as before) and then calling `plot!()` while using the first plot as first argument in the function `plot!()`.

For example:

```
x = 1:20
a = rand(20)
b = rand(20)
multi_plot = plot(x, a, title="Many Series", xlabel="x axis", ylabel="y axis", label="a")
plot!(multi_plot, x, b, label="b")
display(multi_plot)
```


> **i** Note
>
> The `!` after `plot` adds to the existing plot.

Exercise 2.2

Now, create a plot called `multi_series_plot` with three data series `y1`, `y2` and `y3`.

```
# Now, create a plot called `multi_series_plot` with three data series `y1`, `y2` and
↪  `y3`.
# YOUR CODE BELOW

# Test your answer
@assert @isdefined y1
@assert @isdefined y2
@assert @isdefined y3
@assert @isdefined multi_series_plot
println("Multiple series plot created successfully!")
display(multi_series_plot)
```

Section 3: Saving Plots to Files

Plots.jl supports saving your plots to various file formats including PNG, SVG, and PDF, enabling you to use your plots outside of Julia. The function to save plots is `savefig()`, the first argument is the plot itself and the second argument is the `path/filename` format as string. Replace `path` with the path, the `filename` with the actual name and `format` with the file format, e.g. `pdf`, `png`, … . For example, if you want to save your file as PDF, you would just name it `path/filename.pdf`.

Exercise 3.1:

Save the previously created plot `multi_series_plot` to a file as PNG. Save it in the folder `ExampleData` and name it `saved_plot.png`.

```
# Save the previously created plot `multi_series_plot` to a file as PNG. Save it in the
↪  folder `ExampleData` and name it `saved_plot.png`.
# YOUR CODE BELOW

# Test your answer
@assert  isfile("ExampleData/saved_plot.png") "File does not exist yet."
println("Plot saved to 'saved_plot.pdf' successfully!")
```

Conclusion

Fantastic! You've completed the tutorial on basic plotting in Julia. You've learned how to create basic plots and customize and save them. Continue to the next file to learn more.