Plotting & visualization

Human perception is much better adapted to visual representation than to numbers

Purposes of plotting:

- Visualization of research results for publications & presentations
- Debugging + developing algorithms
- "In-situ visualization" of evolving computations
- Investigation of data
- 1D, 2D, 3D, 4D data
- Similar tasks in CAD, Gaming, Virtual Reality . . .

Processing steps in visualization

High level tasks:

- Representation of data using elementary primitives: points,lines, triangles . . .
- Very different depending on purpose

Low level tasks

- Coordinate transformation from "world coordinates" of a particular model to screen coordinates
- Transformation 3D → 2D, visibility computation
- Coloring, lighting, transparency
- Rasterization: turn smooth data into pixels

Software implementation of low level tasks

- Software: rendering libraries, e.g. Cairo, AGG
- Software for vector based graphics formats, e.g. PDF, postscript, svg
- Typically performed on CPU

Hardware for low level tasks

- Low level tasks are characterized by huge number of very similar operations
- Well adaped to parallelism "Single Instruction, Multiple Data" (SIMD)
- Dedicated hardware: *Graphics Processing Unit* (GPU) can free CPU from these taks
- Multiple parallel pipelines, fast memory for intermediate results

(wikimedia)

GPU Programming

- Typically, GPUs are processing units which are connected via bus interface to CPU
- GPU Programming:
    - Prepare low level data for GPU
    - Send data to GPU
    - Process data in rendering pipeline(s)
- Modern visualization programs have a CPU part and GPU parts a.k.a. *shaders*
    - Shaders allow to program details of data processing on GPU
    - Compiled on CPU, sent along with data to GPU
- Modern libraries: Vulkan, modern OpenGL/WebGL, DirectX
- Possibility to "mis-use" GPU for numerical computations

GPU Programming in the "old days"

- "Fixed function pipeline" in OpenGL 1.1 fixed one particular set of shaders
- Easy to program

```
glClear()
glBegin(GL_TRIANGLES)
glVertex3d(1,2,3)
glVertex3d(1,5,4)
glVertex3d(3,9,15)
glEnd()
glSwapBuffers()
```

- Not anymore: now everything works through shaders leading to highly complex programs

Library interfaces to GPU useful for Scientific Visualization

- **vtk** (backend of **Paraview**)
- **three.js** (for WebGL in the browser)
- Alternatively, work directly with OpenGL...
- very few . . .
    - Money seems to be in gaming, battlefield rendering . . .
    - Problem regadless of julia, python, C++, . . .
- Common approach:
    - Write data into "vtk" files, use paraview for visualization.

Graphics in Julia

- **Plots.jl** General purpose plotting package with different backends
    - GPU support via default `gr` backend (based on "old" OpenGL)
- **Plotly.jl** Interface to javascript library plotly.js
    - plots in the browser or electron window
    - also as backend for Plots.jl
    - some WebGL functionality
- **Makie.jl**
    - GPU based plotting using modern OpenGL
    - good plot performance, some precompilation time
    - essentially still under development
- **WGLMakie.jl** maps Makie API to three.js, can be used from the browser
- **WriteVTK.jl** vtk file writer for files to be used with paraview - so this is not a plotting library.
- **PyPlot.jl**: Interface to **python/matplotlib**
    - realization via PyCall.jl

  - also as backend for Plots.jl

PyPlot

During this course we will use PyPlot, but feel free to try some of the other packages.

- It has all the functionality we need (including plots on triangular meshes not available in Plots.jl)
- Python users instantly will recognize the interfaces
- Knowledge obtained here can also be used in python
- Low precompilation time (as opposed to e.g. Makie)

Drawback: Plotting performance - it does not use the GPU, large parts of the logic are in python

PyPlot resources:

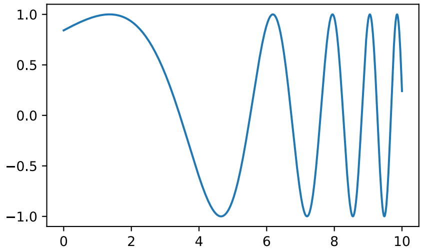
- **Julia package**
- **Julia examples**
- **Matplotlib documentation**

We can choose the way the plot is created: in the browser it can make sense to create it as a vector graphic in svg format. The alternatice is png, a pixel based format.

```
true
```

```
PyPlot.svg(true)
```

How to create a plot ?

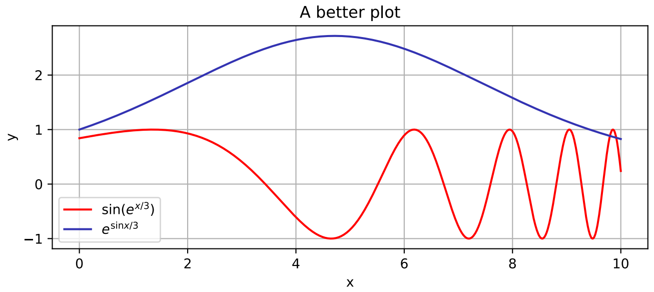


```
let
    X=collect(0:0.01:10)
    PyPlot.clf() # Clear the figure
    PyPlot.plot(X,sin.(exp.(X/3))) # call the plot function
    figure=PyPlot.gcf() # return figure to Pluto
end
```

Instead of a `begin/end` block we used a `let` block. In a let block, all new variables are local and don't interfer with other pluto cells.

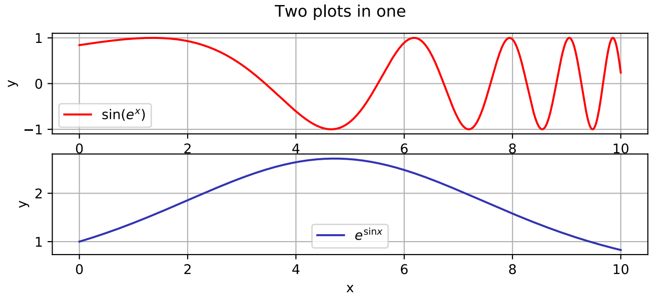
This plot is not nice. It lacks:

- orientation lines ("grid")
- title
- axis labels
- label of the plot
- size adjustment



```
let
    X=collect(0:0.01:10)
    PyPlot.clf()
    PyPlot.plot(X,sin.(exp.(X/3)),
        label="\$\\sin(e^{x/3})\$", color=:red) # Plot with label
    PyPlot.plot(X,exp.(sin.(X/3)),
        label="\$e^{\\sin x/3}\$",color=(0.2,0.2,0.7)) # Plot with label
    PyPlot.legend(loc="lower left") # legend placement
    PyPlot.title("A better plot") # The plot title
    PyPlot.grid() # add grid lines to the plot
    PyPlot.xlabel("x") # x axis label
    PyPlot.ylabel("y") # y axis label
    figure=PyPlot.gcf()
    figure.set_size_inches(8,3) # adjust size
    PyPlot.savefig("myplot.png") # save figure to disk
    figure # return figure
end
```

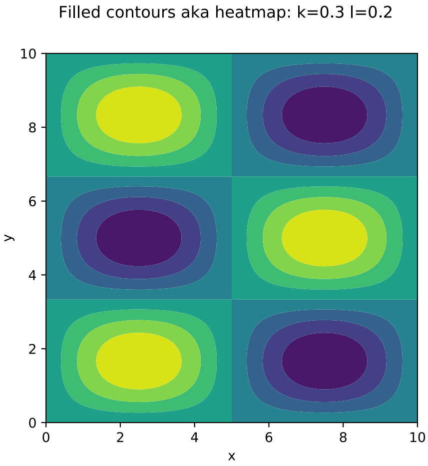
We can use $\LaTeX$ math strings in plot labels here, we just need to escape the `$` symbols with `\` !



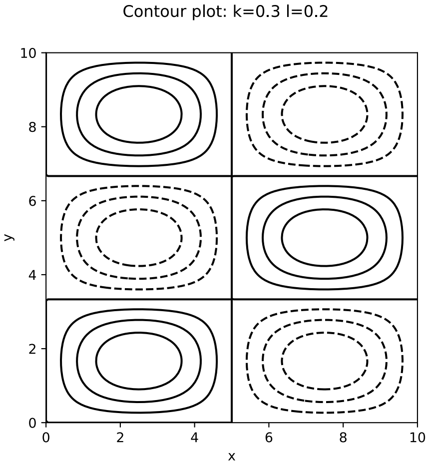
```
let
    X=collect(0:0.01:10)
    PyPlot.clf()
    PyPlot.suptitle("Two plots in one") # Title of compound plot

    PyPlot.subplot(211) # Subplot: 2 rows, 1 column, 1st plot
    PyPlot.plot(X,sin.(exp.(X/3)),
        label="\$\\sin(e^x)\$", color=:red)
    PyPlot.grid()
    PyPlot.xlabel("x")
    PyPlot.ylabel("y")
    PyPlot.legend(loc="lower left")

    PyPlot.subplot(212) # Subplot: 2 rows, 1 column, 2nd plot
    PyPlot.plot(X,exp.(sin.(X/3)),
        label="\$e^{\\sin x}\$",color=(0.2,0.2,0.7))
    PyPlot.legend(loc="lower center")
    PyPlot.grid()
    PyPlot.xlabel("x")
    PyPlot.ylabel("y")

    figure=PyPlot.gcf()
    figure.set_size_inches(8,3)
    figure
end
```

Plotting 2D data

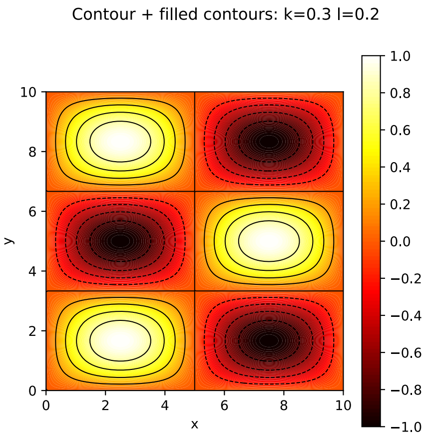
k: l:



```
let
    clf()
    X=collect(0:0.05:10)
    Y=X
    suptitle("Filled contours aka heatmap: k=$(k) l=$(l)")
    F=[sin(k*π*X[i])*sin(l*π*Y[j]) for i=1:length(X), j=1:length(Y)]
    contourf(X,Y,F) # plot filled contours
    xlabel("x")
    ylabel("y")
    figure=gcf()
    figure.set_size_inches(5,5)
    figure
end
```



```
let
    clf()
    X=collect(0:0.05:10)
    Y=X
    suptitle("Contour plot: k=$(k) l=$(l)")
    F=[sin(k*π*X[i])*sin(l*π*Y[j]) for i=1:length(X), j=1:length(Y)]
    contour(X,Y,F,colors=:black)
    xlabel("x")
    ylabel("y")
    gcf()
end
```

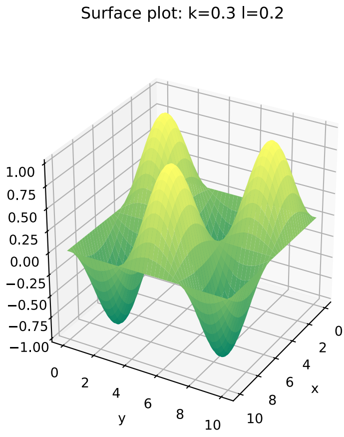


```
let
    clf()
    X=collect(0:0.05:10)
    Y=X
    suptitle("Contour + filled contours: k=$(k) l=$(l)")
    F=[sin(k*π*X[i])*sin(l*π*Y[j]) for i=1:length(X), j=1:length(Y)]
    fmin=minimum(F)
    fmax=maximum(F)
    number_of_isolines=10
    isolines=collect(fmin:(fmax-fmin)/number_of_isolines:fmax)
    cnt=contourf(X,Y,F,cmap="hot",levels=100)
    if fix_moire
        for c in cnt.collections
            c.set_edgecolor("face")
        end
    end
    axes=gca()
    axes.set_aspect(1)
    colorbar(ticks=isolines)
    contour(X,Y,F,colors=:black,linewidths=0.75,levels=isolines)
    xlabel("x")
    ylabel("y")
    gcf()
end
```

Remove the moire in the plot: ☐

This occurs in `contourf` when we use many colors to make a smooth impression.

α: β:



```
let
    clf()
    X=collect(0:0.05:10)
    Y=X
    suptitle("Surface plot: k=$(k) l=$(l)")
    F=[sin(k*π*X[i])*sin(l*π*Y[j]) for i=1:length(X), j=1:length(Y)]

    surf(X,Y,F,cmap=:summer) # 3D surface plot
    ax=gca(projection="3d")  # Obtain 3D plot axes
    ax.view_init(α,β) # Adjust viewing angles


    xlabel("x")
    ylabel("y")
    gcf()
end
```

... all movements could be much faster if we would use the GPU...

There are analogues for `contour` `contourf` and `surf` on triangular meshes which will be discussed once we get there in the course.

Feel free watch my **vizcon2 talk** about using vtk from Julia - just to show what could be possible. Unfortunately, these things currently work only on Linux...