GR: Plotting with Python or Julia

July 11th – 17th, 2016

Scientific Computing with Python • Austin, Texas | Josef Heinen | @josef_heinen

http://goo.gl/XQsXhi
The main ideas behind GR

✓ procedural graphics backend (completely written in C)
  ➡ presentation of continuous data streams

✓ builtin support for 2D plotting and OpenGL (GR3)
  ➡ coexistent 2D and 3D world

✓ support for different programming languages including Python, Julia and JavaScript

✓ backend support for Matplotlib (Python) and Plots (Julia)

✓ interoperability with GUIs (Qt, wx, Gtk) and web applications (Jupyter)
Use GR as a backend for Matplotlib

✓ combine the power of Matplotlib and GR
  ➟ select GR as a backend by setting the MPLBACKEND environment variable
  
  export MPLBACKEND="module://gr.matplotlib.backend_gr"

✓ significantly improve the performance of existing Matplotlib applications

✓ create plots containing both 2D and 3D graphics elements (from different packages)

✓ produce video contents on the fly by adding a single line of code
  ➟ no need to import an animation module or write extra code
Matplotlib using the GR backend
... with Matplotlib’s new colormaps
Use GR as a MATLAB–like plot interface

```matlab
using GR
figure(figsize=(6,4))

tevent = 1126259462.422
d = readdlm("strain.dat")
time, H1_strain, L1_strain = d[:,1], d[:,2], d[:,3]
d = readdlm("nr_waveform.dat")
NRtime, NR_H1_whitenbp = d[:,1], d[:,2]

xlim([-0.1, +0.05])
ylim([-4, +4])
legend("H1 strain", "L1 strain", "matched NR waveform")
xlabel(@sprintf("time (s) since %.3f", tevent))
ylabel("whitented strain")
title("Advanced LIGO WHITENED strain data near GW150914")
plot(time-tevent, H1_strain, "r
time-tevent, L1_strain, "g"
NRtime+0.002, NR_H1_whitenbp, "k")
```

Advanced LIGO WHITENED strain data near GW150914
... for 2D

using LsqFit

\[
x = -16:0.5:35 \\
y = 19.4/((x - 7)^2 + 15.8) + \text{randn(size}(x))/10;
\]

\[
\text{model}(x, p) = p[1]/((x-p[2])^2+p[3])
\]

\[
\text{fit} = \text{curve_fit(model, x, y, [25.,10.,10.])}
\]

\[
p = \text{fit.param}
\]

\[
f = p[1]/((x-p[2])^2+p[3])
\]

using GR

\[
\text{plot}(x, y, "d", x, f, "-", \text{linewidth}=2)
\]
... for 2D (line, scatter)

using GR

```matlab
x = linspace(0,20,200)
for order in 0:3
    plot(x, besselj(order, x))
    hold(true)
end
legend("J_0", "J_1", "J_2", "J_3")
```

using GR

```matlab
n = 500
x = rand(n)
y = rand(n)
area = pi .* (15 .* rand(n)).^2
scatter(x, y, area, 1:n, alpha=0.5)
```
... or 2D (filled contours)

using GR

```matlab
figure(size=(1000,500))

z = readdlm("TerrainHeight.txt")

contourf(max(z',-1500),
    xlim=(0,1440), ylim=(0,720),
    colormap=GR.COLORMAP_TERRAIN)
```
... or 3D (surface)

using GR
inline("mov")

figure(size=(800,800))

z = Z = readdlm("sans.dat")
G = [ exp(-x^2 -y^2) for 
x in linspace(-1.5, 1.5, 128),
y in linspace(-1.5, 1.5, 128) ]

for t = 0:500
    surface(z, title="Time: $t$ s")
    z += 0.05 * Z .* G .* rand((128, 128))
end
... or 3D (iso surface)

using GR
inline("mov")

Ψv = calculate_electronic_density(3, 2, 0)

for alpha in 30:210
    isosurface(Ψv, isovalue=0.25, rotation=alpha)
end
Mix 2D and 3D scenes and create video content on the fly

```python
import PyCall
@PyCall.pyimport moqll1
molecules = moqll1.read("data/700K.xyz");

using GR
inline("mov")
angles = readdlm("data/700K.txt");

lens = []
for t in 1:100
    x = angles[t, 1]
    x = x[!isnan(x)]
    clears()
    subplot(1, 3, 3)
    histogram(x, xlim=(-200, 200), ylim=(0, 2000))
    setviewport(0, 0.7, 0.05, 0.75)
    setwindow(0, 1, 0, 1)
    moqll1.draw(molecules[t])
    settextalign(GR.TEXT_HALIGN_CENTER, GR.TEXT_ALIGN_HALF)
    text(0.35, 0.7, "sprintf("700K (%.1f ps) # of bonds: %.d", t / 10.0, length(x)))
    lens = [lens; length(x)]
    if t > 1
        setwindow(0, 10, 3500, 5000)
        setviewport(0.1, 0.6, 0.05, 0.1)
        axes(1, 0, 3500, 2, 0, 0.005)
        polyline([1:t] / 10.0, lens)
    end
    updatews();
end
```
Use GR with Interact

using Interact
using GR
inline()

z = peaks()
maps = Dict("gnuplot"=>39,
            "viridis"=>44,
            "inferno"=>45)

@manipulate for θ=20:1:70, Φ=20:1:70,
cmap=radiobuttons(["gnuplot",
                  "viridis",
                  "inferno"])
surface(z, rotation=θ, tilt=Φ,
colormap=maps[cmap])
end
… or with JavaScript

GR can be transpiled to JS ➟ gr.js
(Emscripten: LLVM–to–JavaScript compiler)

➤ call GR functions from JS

Use cases:
✓ embed JS code in Jupyter
✓ interpret GR display list in the browser

```javascript
<canvas id="canvas" width="500" height="500"></canvas>
<script type="text/javascript" src="gr.js"></script>
<script type="text/javascript">
GR.ready(function() {
  var gr = new GR();
  var t = 0;
  var x = new Array(629);
  var y = new Array(629);
  var draw = function() {
    gr_clearws();
    for (i = 0; i < 629; i++) {
      x[i] = i / 630.0 * 2 * Math.PI;
      y[i] = Math.sin(x[i] + t / 10.0);
    }
    gr_setviewport(0.1, 0.95, 0.1, 0.95);
    gr_setwindow(0, 8, -1, 1);
    gr_setcharheight(0.020);
    gr_grid(0.5, 0.1, 0, -1, 4, 5);
    gr_axes(0.5, 0.1, 0, -1, 4, 5, 0.01);
    gr_polyline(629, x, y);
    gr_updatews();
    t = t + 1;
    if (t < 200) {
      setTimeout(draw, 1);
    }
    draw();
  };
  draw();
</script>
```
Use GR with Gtk ...

function paint(w)
    ctx = Gtk.getgc(w)
    Gtk.show_text(ctx, "Contour Plot using Gtk ...")

    ENV["GKS_WSTYPE"] = "142"
    ENV["GKSconid"] = @sprintf("%lu", UInt64(ctx.ptr))
    ...

    x, y, z = gridit(xd, yd, zd, 200, 200)
    h = linspace(-0.5, 0.5, 20)
    surface(x, y, z, 5)
    contour(x, y, h, z, 0)
    polymarker(xd, yd)
    axes(0.25, 0.25, -2, -2, 2, 2, 0.01)
end

win = @Window("Gtk")
canvas = @Canvas(600, 450)
Gtk.push!(win, canvas)

Gtk.draw(paint, canvas)
Gtk.showall(win)
...
Gtk.gtk_main()
... or with PyQt / PySide

class GrWidget(QGui.QWidget):
    def __init__(self):
        QGui.QWidget.__init__(self)

        self.setup_ui(self)
        self.rubberband = QGui.QRubberBand(QGui.QRubberBand.Rectangle, self)
        self.setMouseTracking(True)

        environ['GKS_WSTYPE'] = "381"

    def paintEvent(self, ev):
        self.painter.begin(self)
        if have_pyside:
            environ['GKSconid'] = "%x!%x" % (long(shiboken.getCppPointer(self)), long(shiboken.getCppPointer(self.painter)))
        else:
            environ['GKSconid'] = "%x!%x" % (unwrapinstance(self), unwrapinstance(self.painter))

        draw()

        self.painter.end()

app = QGui.QApplication(argv)
win = GrWidget()
win.show()
exit(app.exec_())
Use GR in Atom …

✓ powerful IDE
✓ syntax highlighting
✓ command completion
✓ inline graphics
… or in iTerm2

✓ terminal emulator for macOS
✓ syntax highlighting
✓ command completion
✓ inline graphics
Render molecules ...

```
gr.setviewport(0, 1, 0, 1)
gr3.setbackgroundcolor(1, 1, 1, 1)

for angle in range(360):
gr.clearws()

g3.clear()
g3.drawmolecule('data/dna.xyz', bond_delta=2, tilt=45, rotation=angle)
g3.drawimage(0, 1, 0, 1, 1000, 1000, gr3.GR3_Drawable.GR3_DRAWABLE_GKS)

g.text(0.15, 0.95, "DNA rendered using gr3.drawmolecule")
g.textext(0.15, 0.875, "$\alpha=%d$" % angle)

g.updatews()
```
... and export a POV–Ray scene
Render spins

```
gr.setviewport(0, 1, 0, 1)

gr3.setbackgroundcolor(1, 1, 1, 1)
gr3.cameralookat(0, -60, 35, 0, -2, 0, 0, 2, 0)

for positions, directions, colors in ...
gr.clearws()

g3.clear()
g3.drawspins(positions, directions, colors)
g3.drawimage(0, 1, 0, 1, 1000, 1000,
           gr3.GR3_Drawable.GR3_DRAWABLE_GKS)

g3.updatews()

g3.export('spins.html', 1000, 1000)
```
Demos

✓ GR interoperability example
✓ interactive widget demo for a simple surface plot
✓ visualizing atomic orbitals in a Jupyter notebook
✓ use GR to visualize MRI data in an interactive notebook
Programs using GR: pyMolDyn
... GR as a backend for Plots.jl

```
julia> using Plots
julia> gr()
Plots.GRBackend()
```
Current Activities

✓ near completion of a “fully–featured” GR distribution for Windows based on the MinGW–w64 toolchain

✓ improve the build and deployment process (self–contained GR packages)

✓ provide more (MATLAB–like) convenience functions

✓ proof–of–concept for an in–browser JavaScript–based GR interactive renderer (based on gr.js)
What’s next?

QtTerm
✓ stand-alone viewer (C++, Qt 5.7)
✓ platform-independent
✓ interactive (zoom, pan, pick data, …)

EGS
✓ based on ModernGL
✓ extensibility using plugins
✓ combines 2D and 3D
✓ high-performance rendering
✓ GKS driver
Why EGS? … because performance matters!  
E.g. for the visualization of Micro Swimmers …
… or for the visualization of Skyrmion formations
Thank you for your attention

Questions?

Contact:

j.heinen@fz-juelich.de
@josef_heinen

Thanks to:

Florian Rhiem, Ingo Heimbach, Christian Felder, David Knodt, Jörg
Winkler, Fabian Beule, Philip Klinkhammer, Marvin Goblet, Daniel
Kaiser, et al.