Easyviz Documentation Documentation

Release 1.0

H. P. Langtangen and J. H. Ring

September 09, 2010

CONTENTS

Contents:

CHAPTER

ONE

EASYVIZ

Easyviz is a unified interface to various packages for scientific visualization and plotting. The Easyviz interface is written in Python with the purpose of making it very easy to visualize data in Python scripts. Both curve plots and more advanced 2D/3D visualization of scalar and vector fields are supported. The Easyviz interface was designed with three ideas in mind: 1) a simple, Matlab-like syntax; 2) a unified interface to lots of visualization engines (called backends later): Gnuplot, Matplotlib, Grace, Veusz, Pmw.Blt.Graph, PyX, Matlab, VTK, VisIt, OpenDX; and 3) a minimalistic interface which offers only basic control of plots: curves, linestyles, legends, title, axis extent and names. More fine-tuning of plots can be done by invoking backend-specific commands.

Easyviz was made so that one can postpone the choice of a particular visualization package (and its special associated syntax). This is often useful when you quickly need to visualize curves or 2D/3D fields in your Python program, but haven't really decided which plotting tool to go for. As Python is gaining popularity at universities, students are often forced to continuously switch between Matlab and Python, which is straightforward for array computing, but (previously) annoying for plotting. Easyviz was therefore also made to ease the switch between Python and Matlab.

If you encounter problems with using Easyviz, please visit the *Troubleshooting* chapter and the *Installation* chapter at the end of the documentation.

1.1 Guiding Principles

First principle. Array data can be plotted with a minimal set of keystrokes using a Matlab-like syntax. A simple

```
t = linspace(0, 3, 51)     # 51 points between 0 and 3
y = t**2*exp(-t**2)
plot(t, y)
```

plots the data in (the NumPy array) `t` versus the data in (the NumPy array) `y`. If you need legends, control of the axis, as well as additional curves, all this is obtained by the standard Matlab-style commands

```
y2 = t**4*exp(-t**2)
# pick out each 4 points and add random noise:
t3 = t[::4]
y3 = y2[::4] + random.normal(loc=0, scale=0.02, size=len(t3))

plot(t, y1, 'r-')
hold('on')
plot(t, y2, 'b-')
plot(t3, y3, 'bo')
legend('t^2*exp(-t^2)', 't^4*exp(-t^2)', 'data')
title('Simple Plot Demo')
axis([0, 3, -0.05, 0.6])
xlabel('t')
```

```
ylabel('y')
show()

hardcopy('tmp0.ps')  # this one can be included in LaTeX
hardcopy('tmp0.png') # this one can be included in HTML
```

Easyviz also allows these additional function calls to be executed as a part of the `plot` call:

```
plot(t, y1, 'r-', t, y2, 'b-', t3, y3, 'bo',
     legend=('t^2*exp(-t^2)', 't^4*exp(-t^2)', 'data'),
     title='Simple Plot Demo',
     axis=(0, 3, -0.05, 0.6),
     xlabel='t', ylabel='y',
     hardcopy='tmp1.ps',
     show=True)

hardcopy('tmp0.png') # this one can be included in HTML
```

A scalar function $f(x, y)$ may be visualized as an elevated surface with colors using these commands:

```
x = linspace(-2, 2, 41)  # 41 point on [-2, 2]
xv, yv = ndgrid(x, x)    # define a 2D grid with points (xv,yv)
values = f(xv, yv)       # function values
surfc(xv, yv, values,
      shading='interp',
      clevels=15,
      clabels='on',
      hidden='on',
      show=True)
```

Second princple. Easyviz is just a unified interface to other plotting packages that can be called from Python. Such plotting packages are referred to as backends. Several backends are supported: Gnuplot, Matplotlib, Grace (Xmgr), Veusz, Pmw.Blt.Graph, PyX, Matlab, VTK, VisIt, OpenDX. In other words, scripts that use Easyviz commands only, can work with a variety of backends, depending on what you have installed on the machine in question and what quality of the plots you demand. For example, switching from Gnuplot to Matplotlib is trivial.

Scripts with Easyviz commands will most probably run anywhere since at least the Gnuplot package can always be installed right away on any platform. In practice this means that when you write a script to automate investigation of a scientific problem, you can always quickly plot your data with Easyviz (i.e., Matlab-like) commands and postpone to marry any specific plotting tool. Most likely, the choice of plotting backend can remain flexible. This will also allow old scripts to work with new fancy plotting packages in the future if Easyviz backends are written for those packages.

Third principle. The Easyviz interface is minimalistic, aimed at rapid prototyping of plots. This makes the Easyviz code easy to read and extend (e.g., with new backends). If you need more sophisticated plotting, like controlling tickmarks, inserting annotations, etc., you must grab the backend object and use the backend-specific syntax to fine-tune the plot. The idea is that you can get away with Easyviz and a plotting package-independent script "95 percent" of the time - only now and then there will be demand for package-dependent code for fine-tuning and customization of figures.

These three principles and the Easyviz implementation make simple things simple and unified, and complicated things are not more complicated than they would otherwise be. You can always start out with the simple commands - and jump to complicated fine-tuning only when strictly needed.

CHAPTER

TWO

TUTORIAL

This tutorial starts with plotting a single curve with a simple `plot(x,y)` command. Then we add a legend, axis labels, a title, etc. Thereafter we show how multiple curves are plotted together. We also explain how line styles and axis range can be controlled. The next section deals with animations and making movie files. More advanced topics such as fine tuning of plots (using plotting package-specific commands) and working with Axis and Figure objects close the curve plotting part of the tutorial.

Various methods for visualization of scalar fields in 2D and 3D are treated next, before we show how 2D and 3D vector fields can be handled.

2.1 Plotting a Single Curve

Let us plot the curve $y = t^2 \exp(-t^2)$ for t values between 0 and 3. First we generate equally spaced coordinates for t, say 51 values (50 intervals). Then we compute the corresponding y values at these points, before we call the `plot(t,y)` command to make the curve plot. Here is the complete program:

```
from scitools.std import *

def f(t):
    return t**2*exp(-t**2)

t = linspace(0, 3, 51)     # 51 points between 0 and 3
y = zeros(len(t))          # allocate y with float elements
for i in xrange(len(t)):
    y[i] = f(t[i])

plot(t, y)
```

The first line imports all of SciTools and Easyviz that can be handy to have when doing scientific computations. In this program we pre-allocate the `y` array and fill it with values, element by element, in a Python loop. Alternatively, we may operate on the whole `t` array at once, which yields faster and shorter code:

```
from scitools.std import *

def f(t):
    return t**2*exp(-t**2)

t = linspace(0, 3, 51)     # 51 points between 0 and 3
y = f(t)                   # compute all f values at once
plot(t, y)
```

The `f` function can also be skipped, if desired, so that we can write directly