1 Matplotlib

Matplotlib[1] ist das komplementäre Plotting-Interface zu den wissenschaftlichen Python-Bibliotheken *Numpy*[2] und *SciPy*[3]. Gegenüber z.B. gnuplot ist der Vorteil, dass matplotlib als Modul für Python implementiert ist, also man die kompletten Sprachfeatures von Python bei der Erstellung der Plots (so natürlich auch NumPy und SciPy) nutzen kann[4]. Die Benutzung von matplotlib ist sehr simpel, wobei Dank der Objektorientierung auch eine effiziente Erweiterung von matplotlib-Routinen möglich ist. Zudem ist die Benutzung der meisten Plot-Routinen stark an die Semantik von *MATLAB* angelehnt, um desen Benutzern einen leichten Einstieg zu ermöglichen. Ein großer Nachteil ist (momentan) der Zustand der offiziellen Dokumentation und Beispiel-Galerie, die leider häufig die Simplizität von matplotlib durch unnötig umständliche Vorgehensweisen verdecken. Daher ist die Motivation dieses Handouts eine Einführung in die wichtigsten Funktionen zu bieten, mit denen sich die häufigst auftretenden Plot-Problemstellungen lösen lassen.

Im Folgenden wird der Vereinfachung halber davon ausgegangen, dass iPython als interaktive Python-Shell benutzt wird, und NumPy und SciPy wie folgt importiert wurden. Zum Plotten benutzen wir das Plotting-Modul *matplotlib.pyplot*

```
import numpy as np                # Import des Modules numpy als np
import scipy as sp                # Import des Modules scipy als sp
import matplotlib.pyplot as plt # Import als plt
```

1.1 Einfaches Plotten

Um einen Datensatz von (jeweils gleichvielen) x- und y-Werten gegeneinander darzustellen, genügen wenige Zeilen Python-Codes:

```
x = np.arange(20)
y = x**2
plt.plot(x,y)
plt.show()
```

plt.plot(x,y) trägt wie zu erwarten war das zweite Argument gegen das erste auf. Dieses erzeugt ein *<matplotlib.lines.Line2D*-Objekt (ein 2D-Liniengraphen), und implizit ein axes-Objekt und ein figure-Objekt. Dieses wird mit *plt.show()* auf den Bildschirm ausgegeben. Es wird ein Fenster erzeugt mit weiteren Kontrollsrukturen, um z.B. reinzuzoomen, den Bildausschnitt zu verschieben, oder den Graphen in eine Datei zu speichern. Außerdem wird durch diesen Befhel der interaktive Modus aktiviert, wodurch nach jedem Plot-Befehl die Fensterausgabe aktualisiert wird (s. Abb. 1).

Den interaktiven Modus verlässt man mit dem Befehl *plt.ioff()*. Um den Graphen jetzt stattdessen direkt in eine Datei auszugeben, verwende man die Funktion *plt.savefig*. Es ist zu beachten, dass der plot erneut generiert werden muss:

```
plt.ioff() # verhindert, dass das interaktive Fenster erscheint
plt.plot(x,y)
plt.savefig("ploty.png") # speichere Plot in Datei plotxy.png
plt.show() # erzeugt das Ausgabefenster und aktivietrt den interaktiven Modus
```

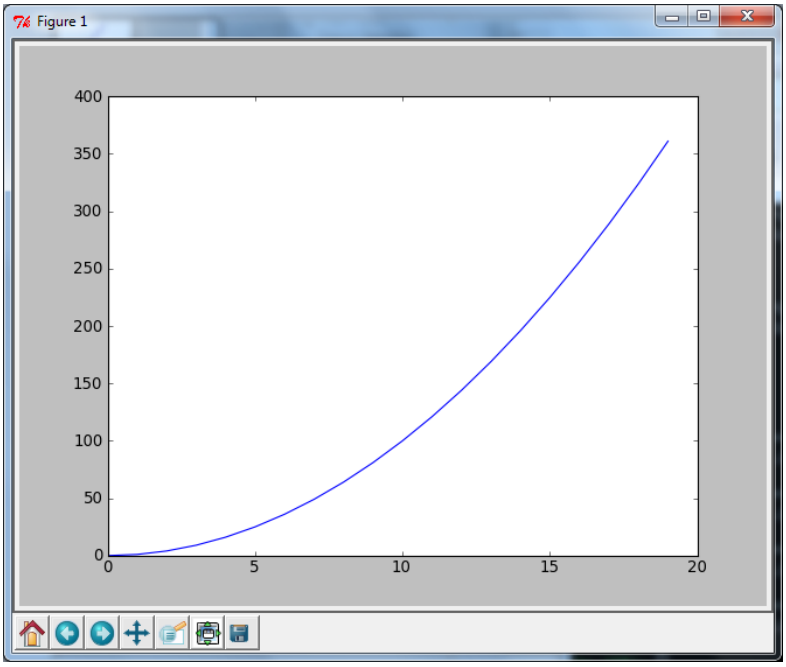

Abbildung 1: Ausgabefenster von plt.show()

Die Beschriftung des Plots lässt sich kontrollieren über diverse Funktionen, von denen ich die wichtigsten hier kurz skizziere (s. Abb. 2):

```
plt.xlabel("x / m") # Beschrifte X-Achse
plt.ylabel("y / $m^2$") # Beschrifte Y-Achse inkl. LaTeX-Code
plt.title("Beschrifteter Plot") # Titel des Plots
plt.plot(x,y,label="$x^2$") # plotte x**2 gegen x
plt.plot(x,y,linestyle=' ',marker='+',label="$x^2$ (Markers)") # plotte x**2 gegen mit Markern
plt.legend() # Füge die Legende zum Plot hinzu
```

Auf weitere Möglichkeiten der Gestaltung der Plots, z.B. das Ändern der Achsenskalierungen und Achsen-"TickBeschriftungen gehe ich hier nicht weiter ein, dieses ist bitte aus der Online-Dokumentation zu entnehmen. Erwähnenswert ist noch die Funktion *plt.xlim* bzw *plt.ylim*, mit Hilfe derer man den x- bzw. y-Wertebereich einschränken kann.

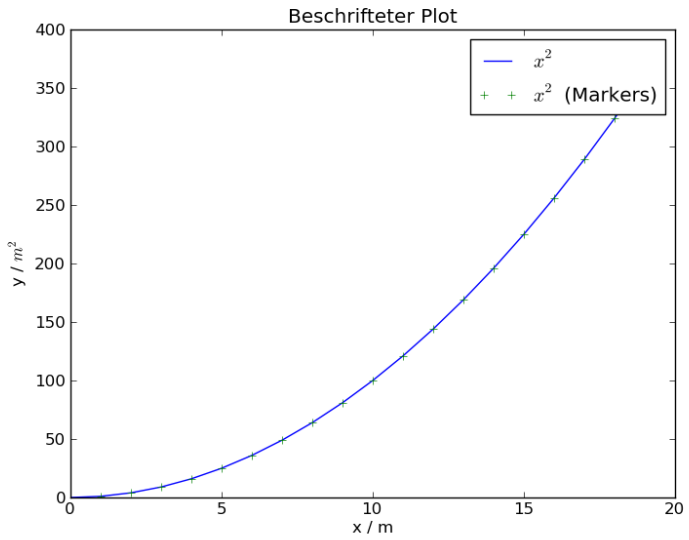

Abbildung 2: Der beschriftete Beispielplot

1.2 Histogramme

Da Matplotlib eng mit NumPy verbunden ist, benutzt es für einige seiner Fuktionen NumPy/SciPy-Routinen. Das bietet den Vorteil, dass einige Funktionen extrem einfach zu benutzen sind, so z.B. das Erstellen und Plotten von Histogrammen (s. Abb. 3):

Zuerst löschen wir den alten Plot mit Hilfe von *plt.clf()*

```
plt.clf() # lösche aktuelle Abbildung
```

```
g = np.random.randn(3000) # Ziehe 3000 normalverteilte Zufallszahlen
plt.hist(g) # erzeuge einen Histogrammplot
h,b,p = plt.hist(g,bins=30) # erzeuge ein Histogramm mit 30 bins
print h # gib die Counts der Bins aus
print b # gib die Bins aus
```

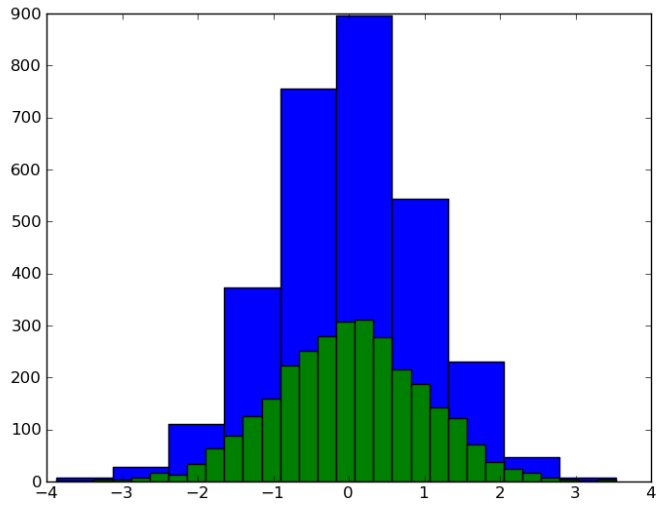

Abbildung 3: Histogramme einer Normalverteilung

Ebenso lassen sich einfach 2D-Histogramme in Falschfarben darstellen (s. Abb. 4). Zu beachten ist wegen der einer MATLAB-Konvention, dass bei pcolor die Histogramm-countmatrix transponiert werden muss[5]:

```
x = np.random.randn(3000) # Ziehe 3000 normalverteilte Zufallszahlen
y = np.random.randn(3000) # Ziehe 3000 normalverteilte Zufallszahlen
h,bx,by = np.histogram2d(x,y,bins=30) # erzeuge 2D-Histogramm
plt.pcolor(bx,by,h.T) # erzeuge Falschfarben-Plot, transponieren wegen MATLAB-Konvention
```

Für hexagonales Binning kann man direkt die Matplotlib-Routine *hexbin* nutzen:

```
plt.hexbin(x,y,gridsize=30) # erzeuge Falschfarben-Plot mit hexbins
```

Weitere Optionen zur Darstellung der Histogramme sind z.B. *plt.contour* Zum Plotten der Konturlinien, oder *plt.contourf* zum Plotten von gefüllten Konturflächen.

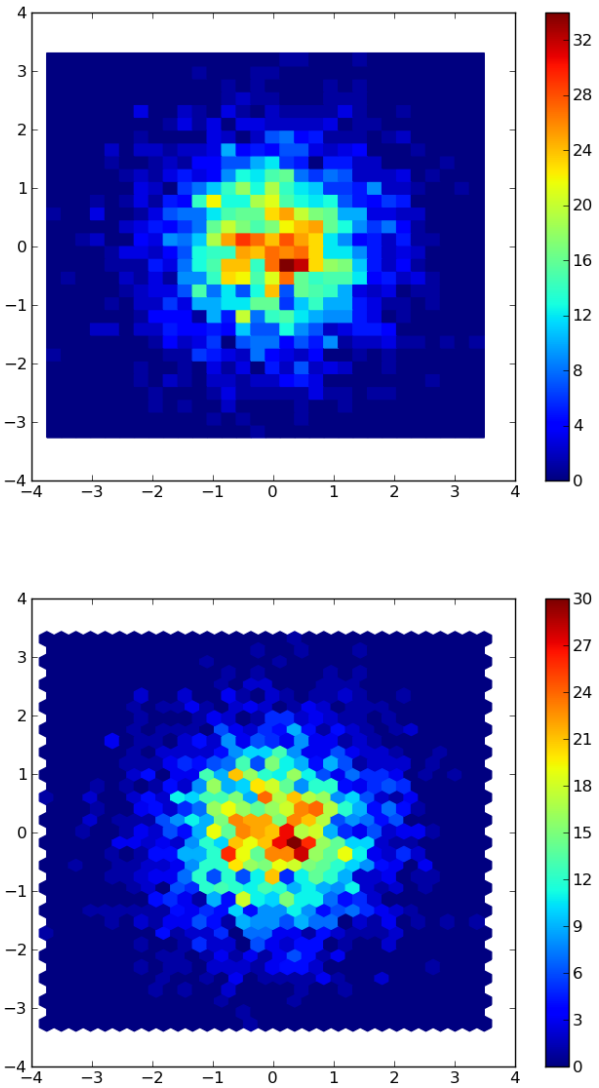

Abbildung 4: 2D-Histogramme einer Normalverteilung

1.3 Multiplots

Es ist natürlich auch möglich, mit Hilfe von Matplotlib Mutliplots, d.h. mehrerer Plots pro Abbildung zu erzeugen. Anhand der Vorgehensweise um dies zu erreichen zeige ich ein wenig die Objekt-Struktur eines Matplotlib-Plots auf.

Wir beginnen damit, ein *figure*-Objekt zu erzeugen. Eine figure ist sozusagen die gesamte Abbildung aller Plots. Im interaktiven Modus entspricht eine figure einem Plot-Fenster:

```
fig = plt.figure() # Erzeuge eine neue Abbildung
```

Sollten wir uns im interaktiven Modus befinden, wird damit ein neues Fenster erzeugt. Um in diesem Fenster einen neuen Plot anzeigen zu können, müssen wir der Abbildung, dem figure-Objekt, ein neues *axes*-Objekt hinzufügen:

```
axes = fig.add_subplot(111) # erzeuge einen einzelnen Plot
```

Die Semantik der Argumente an *add_ subplot* ist dabei gleich derer in MATLAB, d.h. der obige Befehl gibt den ersten Subplot in einer 1-spaltigen und 1-zeiligen Anordnungsmuster zurück. Allg. gibt die erste Ziffer die Anzahl Zeilen, die zweite die Anzahl Spalten, und die dritte die Auswahl des Plots an, wobei die Plots zeilenweise durchnummeriert werden[6]. Unsere figure erhält damit einen einzelnen (Sub-)Plot, um Plots anzeigen zu können. Wir speichern uns diesen Plot in die Variable axis. Mit Hilfe der gespeicherten Variablen können wir jetzt Plots in unsere axes ausgeben:

```
x = np.arange(20)
axes.plot(x,x**2)
```

Es wird der gleiche Plot wie im allerersten Beispiel ausgegeben. Diese Vorgehensweise mag umständlich erscheinen, aber ermöglicht dem Benutzer eine direktere Kontrolle über die Beschriftung und Anordnung der einzelnen Objekte der figure, da man hierbei sich explizit die figure und axes gespeichert hat. Einige erweiterte Formatierungsoptionen funktionieren nämlich nur auf den figure- und axes-Objekten. Sollte man den Weg über das manuelle erzeugen der figure und axes nicht machen wollen, kann man auf die implizit erzuegte figure und axis mit *plt.gcf()* bzw *plt.gca()* zugreifen.

Wir können jetzt mit Hilfe der axes-Objekte z.B. einfach Multiplots generieren (s. Abb. 5):

```
fig = plt.figure() # neue figure
ax1 = f.add_subplot(211) # füge obere axes hinzu
ax2 = f.add_subplot(212) # füge untere axes hinzu
x = np.arange(20)
y1 = x
y2 = x**2
ax1.plot(x,y1)
ax2.plot(x,y2)
```

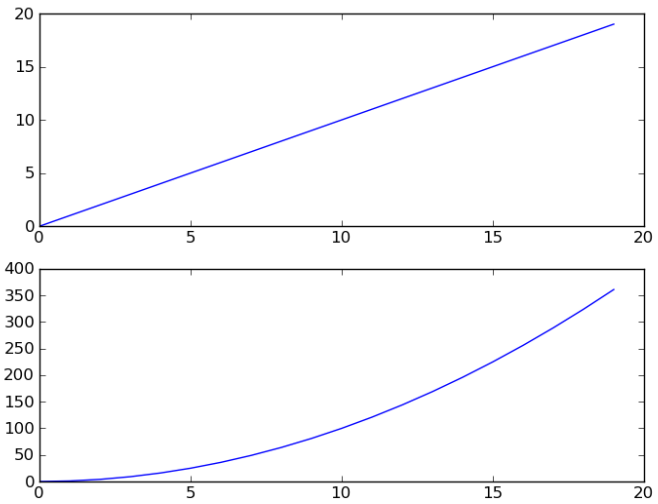

Abbildung 5: Multiplot

1.4 Referenzen

Literatur

[1] http://matplotlib.sourceforge.net/

[2] http://docs.scipy.org/doc/numpy/reference/

[3] http://docs.scipy.org/doc/scipy/reference/

[4] http://shreevatsa.wordpress.com/2010/03/07/matplotlib-tutorial/

[5] http://matplotlib.sourceforge.net/api/pyplot_api.html#axes-pcolor-grid-orientation

[6] http://www.mathworks.co.uk/help/techdoc/ref/subplot.html