NumPy (Fortsetzung)

Matrizen, Vektoren, lineare Algebra

Vektoren werden als 1d Arrays dargestellt, Matrizen als 2d.

```
import numpy as np
v = np.array( (1.5, 1, 0.5) )
M = np.array( ( (3,1,2), (1,2,4), (2,4,-1) ), dtype=float)
```

Lineare Algebra

- Vektor-Vektor:
  - `np.cross()`: Kreuzprodukt
  - `np.dot()`: Skalarprodukt
  - `np.linalg.norm()`: 2-Norm/Länge
- Matrix:
  - `np.linalg.det()`: Determinante
  - `np.trace()`: Spur
  - `np.linalg.eig()`: Eigenwerte- und Vektoren
- Matrix-Matrix und Matrix-Vektor
  - `np.matmul()` Matrix * Matrix und Matrix * Vektor
  - `np.linalg.solve()`: löst das Gleichungssystem $Ax = b$ mit Matrix A und Vektoren x und b

Beispiel: Eigenwerte und Eigenvektoren

```
import numpy as np
v = np.array( (1.5, 1, 0.5) )
M = np.array( ( (3,1,2), (1,2,4), (2,4,-1) ), dtype=float)
eval, evec = np.linalg.eig(M)
print(eval)
print(evec)
# Matrix * Eigenvektor = Eigenwert * Eigenvektor
print(eval[0]*evec[:,0],  np.matmul(M, evec[:,0]))

# The matrix built from eigenvectors can be used for basis transformations.
# In its Eigenbasis, the matrix is diagonal with the Eigenvalues on the diagonal
# E^-1 M E
print(np.matmul(np.matmul(np.linalg.inv(evec),i M),  evec))
```

Zufallszahlen

- `np.random.random()` Fließkommazahl gleichverteilt zwischen 0 und 1
- `np.random.randint()` Ganzzahl zwischen 0 und einem gegebenen Wert
- `np.random.normal()` Normalverteilte (gaußverteilte) Flieskommazahl
- `np.random.permutation()` zufällige Vertauschung von Listenelementen
- `np.random.seed()` setzt den Seed (Anfangszustand) des Zufallsgenerators

Plotten mit Matplotlib

Überblick

- Ein Modul zum Erstellen von Graphen, mächtiger als Gnuplot
- 2D oder 3D, mehrere Graphen in einem
- Speichern als Bitmap oder Vektorgrafik
- Kann auch animierte Kurven darstellen
- Es gibt mehrere Arten der Nutzung. Die einfachste ist `pyplot`

2d-Plots

1. `matplotlib.pyplot` importieren
2. x- und y-Daten in NumPy-Arrays speichern
3. `pyplot.plot(x, y)` für jede zu zeichnende Kurve einmal aufrufen
4. Mit `pyplot.show()` anzeigen oder mit `pyplot.savefig()` speichern

- Linien- oder Punktstile werden mit Zeichenketten beschrieben
  - z.B. `.`, `o`, `+`, `^` für Punkte, Kreise, Kreuze, Dreiecke
  - z.B. `-`, `--`, `-.` für Linie, gestrichelte Linie, gestrichpunktete Linie
  - z.B. `k` für Schwarz, `r` für Rot, `b` für Blau
  - kombinierbar: `.k` `-r` `--b`
  - verwenden als drittes Argument zu `pyplot.plot()`

Logarithmisch und halblogarithmisch Plotten

- Warum sollte man das überhaupt wollen?
- Mittels `pyplot.semilogx()` `pyplot.semilogy()` und `pyplot.loglog()`
- Ansonsten wie vorhin

Plots aufhübschen

- Achsenbeschriftung mit `pyplot.xlabel()` und `ylabel()`
- Legende mit `pyplot.legend()`

- Titel mit `pyplot.suptitle()`
- Text an belibiger Stelle mit `pyplot.text()`
- Gezeigten Wertebereich einstellen mit `pyplot.xlim()` und `ylim()`
- Linien- und Punktstärken mit `pyplot.plot(..., linewidth=..., markersize=...)`

Scatter-Plots

- Zeichnet für jedes (x, y)-Paar einen Punkt.
- Zum Erkennen von Zusammenhängen zwischen Variablen aus einer Stichprobe
  - Alter vs. Kontostand
  - Preisänderung der Aktien von Ölkonzernen und Fluglinien
- Zur Darstellung räumlich verteilter seltener Ereignisse (Blitzschläge)
- Mittels `pyplot.scatter()`

Heatmaps und Contourplots

Heatmap

- Die räumliche Verteilung eines skalaren Wertes wird farb- oder helligkeitskodiert
  - Temperatur, Niederschlagsmenge als Funktion des Ortes
  - Skalarwertige Funktionen von zwei Variablen
- Mittels `pyplot.imshow()`

Contour-Plots

- Auch für skalarwertige Funktionen von zwei Variablen
- Aber Isolinien (Linien, auf denen der Wert gleich bleibt) werden eingezeichnet
  - Höhenlinien auf der Fahrradkarte…
- Mittels `pyplot.contour()`

Beispiel für 3d-Plots

Matplotlib kann 3d-Oberflächen und Wireframes zeichnen.

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Grab some test data.
X, Y, Z = axes3d.get_test_data(0.05)

# Plot a basic wireframe.
```

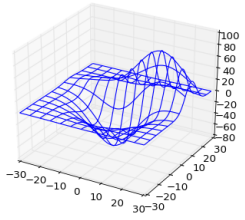

Abbildung 1: aus dem Matplotlib-Tutorial

```
ax.plot_wireframe(X, Y, Z, rstride=10, cstride=10)
plt.show()
```

SciPy

Überblick

SciPy bietet vieles rund ums wissenschaftliche Rechnen

- Optimierung, Fitten
- Statistik
- Fouriertransformationen
- Signalverarbeitung

Fitten

- Wir kennen die Gleichung, die unsere Messdaten beschreibt, wissen aber die Parameter nicht.
- Beim Fitten werden die Parameter gesucht, die am besten zu den Daten passen

1. Aus `scipy.optimize` die Funktion `curve_fit` importieren
2. Die Messdaten laden
3. Die Fitfunktion definieren, oft mittels eines `lambda`
4. Halbwegs realistische Anfangswerte für die Parameter raten
5. Grenzen für den Parameterbereich, in dem gesucht werden soll, raten
6. Fitten

```
import numpy as np
from scipy.optimize import curve_fit

# Load data
```

```
data = np.genfromtxt("damp_osc.dat")
# Split columns
t = data[:,0]
y = data[:,1]

# Fit function
f = lambda t,amplitude,damping, omega : amplitude *  np.exp(-damping*t) * np.cos(omega*t)

# Guessed initial parameters
p0 = (y[0], 1.0, 1.0)
result = curve_fit(f, t,y, p0)
print("Fit amplitude, damping, omega", result[0])
print("Estimated parameter errors" , np.sqrt(np.diag(result[1])))
```