Zusammenfassung Notebooks Zeitreihenanalyse

June 19, 2016

Sammlung der Notebooks zur Vorlesung Zeitreihenanalyse.

1 Zeitreihenanalyse

Sommersemester 2016, B.Sc. Geophysik/Ozeanographie
lars.kaleschke@uni-hamburg.de

1.1 Vorläufiger Zeitplan

1. Mi, 6. Apr. Einführung Zeitreihen Klassifikation Trends
2. Mi, 13. Apr. Zufallsvariablen, statistische Parameter
3. Mi, 20. Apr. Least-Squares Methode, Lineare Regression
4. Mi, 27. Apr. Zentraler Grenzwertsatz, Konfidenzintervalle
5. Mi, 4. Mai Hypothesentests,
6. Mi, 11. Mai Spektralanalyse, FFT, Fenster (Klaus-Werner Gurgel)
7. Mi. 25. Mai Stochastische Prozesse, Autoregressive Modelle, Faltung
8. Mi, 1. Jun. Filter
9. Mi, 8. Jun. Vorhersagen, SVD, multilineare Regression
10. Mi, 15. Jun. Signifikanz von Korrelationen, Pearson, Spearman Rangkorrelation
11. Mi, 22. Jun. Zusammenfassung und Wiederholung
12. Mi, 29. Jun. -
13. Mi, 6. Jul. Klausur

1.2 Struktur

- Vorlesung (Raum ZMAW 022/23)
- Praktische Übungen (Geomatikum R. 1536a)

Bewertung: Klausur
Notwendig:

- Regelmäßige aktive Teilnahme an praktischen Übungen.

- Schriftliche Ausarbeitungen der Aufgaben (Gruppenarbeit möglich).
- Mindestens zweimal mündliche Präsentation von Ergebnissen.

1.3 Literatur

1.3.1 Theorie Zeitreihen und statistische Datenanalyse

- Statistische Analyse von Zeitserien, Scriptum zur Vorlesung 63-618, Detlef Stammer, Version 2.4, 6. Januar 2014 (Stine)

- Bendat, J.S. and A. G. Piersol, Random Data Analysis and Measurement Procedures.John Wiley and Sons, 2010
- Praktische Statistik für Meteorologen und Geowissenschaftler, Christian-Dietrich Schönwiese, 4. Auflage 2006
- v. Storch and Zwiers, 1999: Statistical Analysis in Climate Research
- Jenkins and Watts, 1968: Spectral Analysis and its Applications. Holden-Day, 525 pp.
- Fouriertransformation für Fußgänger, T. Butz, Springer 2011
- Data Analysis Methods in Physical Oceanography, Richard E. Thomson and William J. Emery, Elsevier B.V. 2014
- Claude Duchon, Robert Hale, Time Series Analysis in Meteorology and Climatology: An Introduction, Wiley 2012

1.4 Software

- http://jupyter.org/ Open source, interactive data science and scientific computing across over 40 programming languages.
- https://www.scipy.org/ SciPy is a Python-based ecosystem of open-source software for mathematics, science, and engineering
- https://github.com/winpython Winpython Distribution
- http://continuum.io/downloads Anaconda Distribution
- https://www.gnu.org/software/octave/ Octave, freie Matlab Variante
- https://www.r-project.org/ R Statistik

1.5 Cloud Service

- SageMathCloud Interaktive Zusammenarbeit an Worksheets bzw. Notebooks.
- Seit Juni 2016 gibt es Jupyter Notebooks in der Microsoft Cloud Azure

Die Internetoption der SageMathCloud ist kostenpflichtig. Die meisten Übungen sind ohne diese Option zu bearbeiten. Datensätze können mit Copy und Paste im Textformat übertragen werden und werden darum als Textfile bereitgestellt.

1.5.1 Tutorials

Eine kleine Auswahl frei verfügbarer Anleitungen

- https://docs.python.org/2/tutorial/ Python-Grundlagen
- http://www.codecademy.com/tracks/python Interaktiver Kurs
- http://docs.scipy.org/doc/scipy/reference/tutorial/ SciPy-Tutorial
- http://wiki.scipy.org/Tentative_NumPy_Tutorial NumPy (Arrays)
- http://matplotlib.org/1.3.1/users/pyplot_tutorial.html Plotting
- http://www.loria.fr/~rougier/teaching/matplotlib/ Plotting
- Digitale Signalverarbeitung mit Python NumPy, SciPy und Matplotlib

1.5.2 Bücher zu Python und Daten/Zeitreihenanalyse

- Python for data analysis : [agile tools for real-world data] / Wes McKinney. - 1. ed. [Online-Ausg.]. - Beijing [u.a.] : O'Reilly, 2012
- Python for Signal Processing : Featuring IPython Notebooks / José Unpingco. - [Online-Ausg.]. - Dordrecht : Springer, 2014

- A Primer on Scientific Programming with Python / Hans Petter Langtangen. - 3rd ed. 2012. [Online-Ausg.]. - Berlin, Heidelberg : Springer, 2012
- Statistical Analysis of Climate Series : Analyzing, Plotting, Modeling, and Predicting with R / Helmut Pruscha. - [Online-Ausg.]. - Berlin : Springer, 2013

2 Einführung in die Zeitreihenanalyse

2.1 Klassifikation von Zeitserien, Beispiele, Begrifflichkeiten

Folgende Eigenschaften dienen zur Klassifikation von Zeitserien:

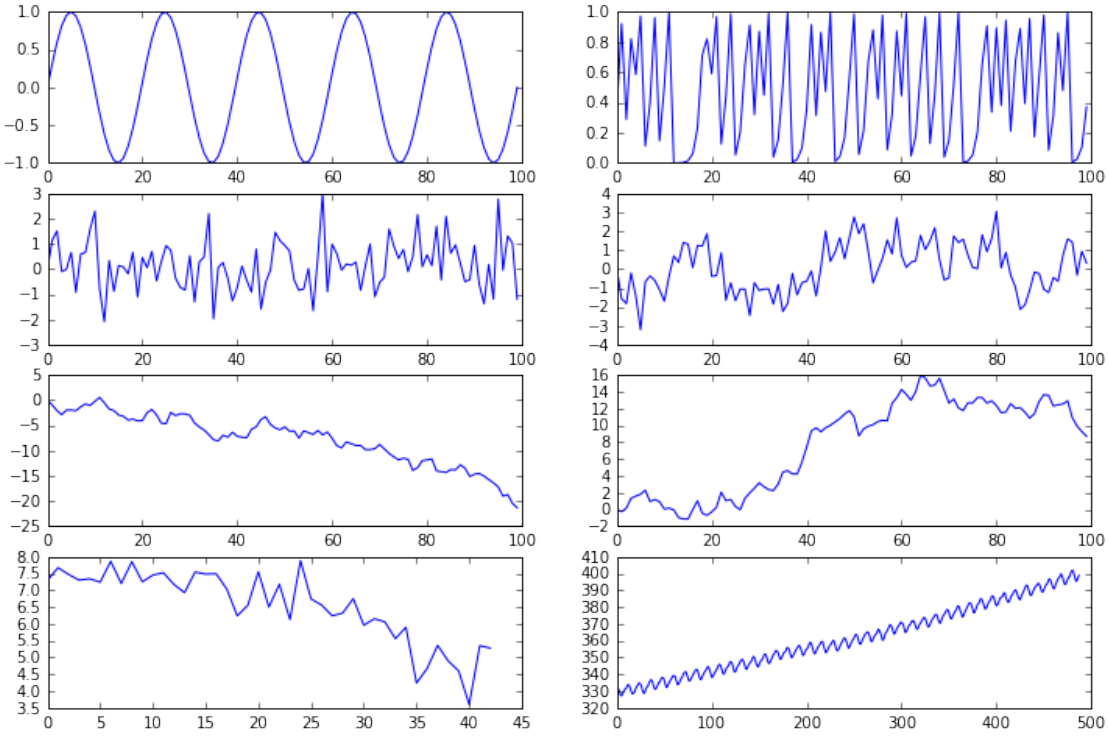
- **Periodisch**
- **Deterministisch**
- **Stochastisch**
- **Stationär**
- **Ergodisch**

Aufgabe: Versuchen Sie die folgenden Zeitserien anhand des visuellen Eindrucks zu klassifizieren. Versuchen Sie den Ursprung der Daten zu erraten. Erkennen Sie charakteristische geophysikalische Zeitreihen wieder?

Es folgt zunächst der Python-Programmcode zur Ausgabe der Plots. Weiter unten ist des Rätsels Lösung.

```
In [1]: # Aktivieren der Inline-Ausgabe von Grafiken
        %pylab inline
        import glob

        file_list=glob.glob('beispiele/*.txt') # Such nach passenden Dateien mit Platzhaltern (Stern, F
        file_list.sort() # Files in aufsteigender Reihenfole sortieren

        figure(figsize=(12,8)) # Öffne neues Fenster zur graphischen Ausgabe
        for i,f in enumerate(file_list): # Schleife über alle gefundenen Dateien, Zählervariable i,f
            subplot(4,2,i+1)
            plot(loadtxt(f)) # Lese und plotte Inhalt der Datei f
        savefig('beispiele_klassifikation.png',dpi=150)
```

```
Populating the interactive namespace from numpy and matplotlib
```



3 Deterministische Zeitserien

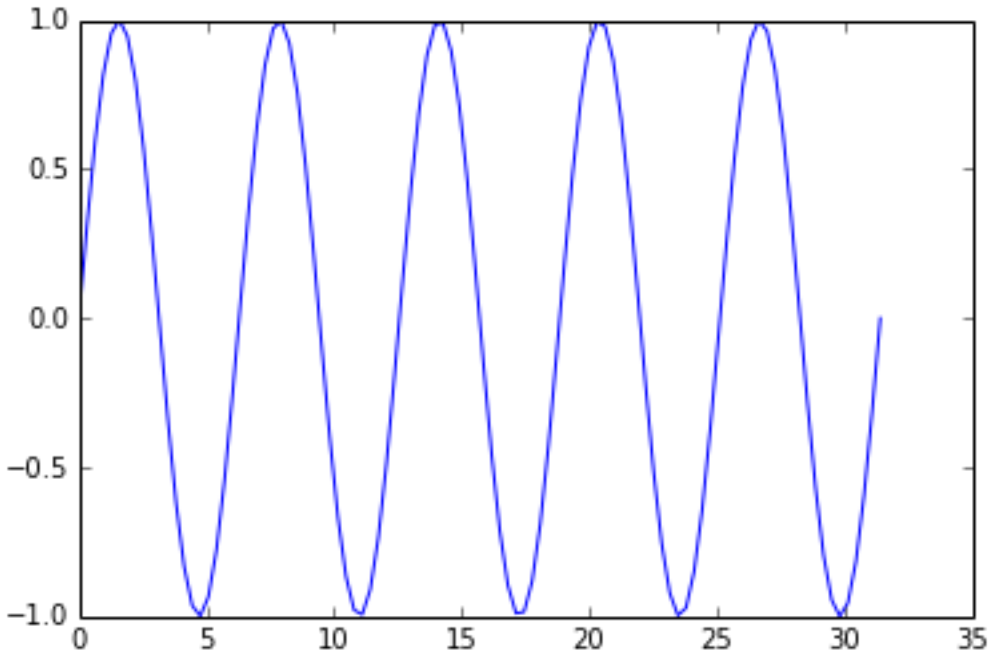
Deterministische Zeitserien können durch einen mathematischen Zusammenhang exakt beschrieben werden. Sie sind entweder periodisch oder nicht-periodisch. Ein wichtiger Spezialfall ist das deterministische Chaos, welches irregulär erscheint.

3.1 Periodische Zeitserien

Periodische Zeitreihen wiederholen sich nach einem regulären Intervall (der Periode T_p)

$$x(t) = x(t \pm nT_p)$$

```
In [2]: N=100
        #X=zeros(N)
        t=linspace(0,10*pi,N)
        Y=sin(t)
        plot(t,Y)
        Y.tofile('beispiele/beispiel_01.txt',sep=' ',format='%s')
```



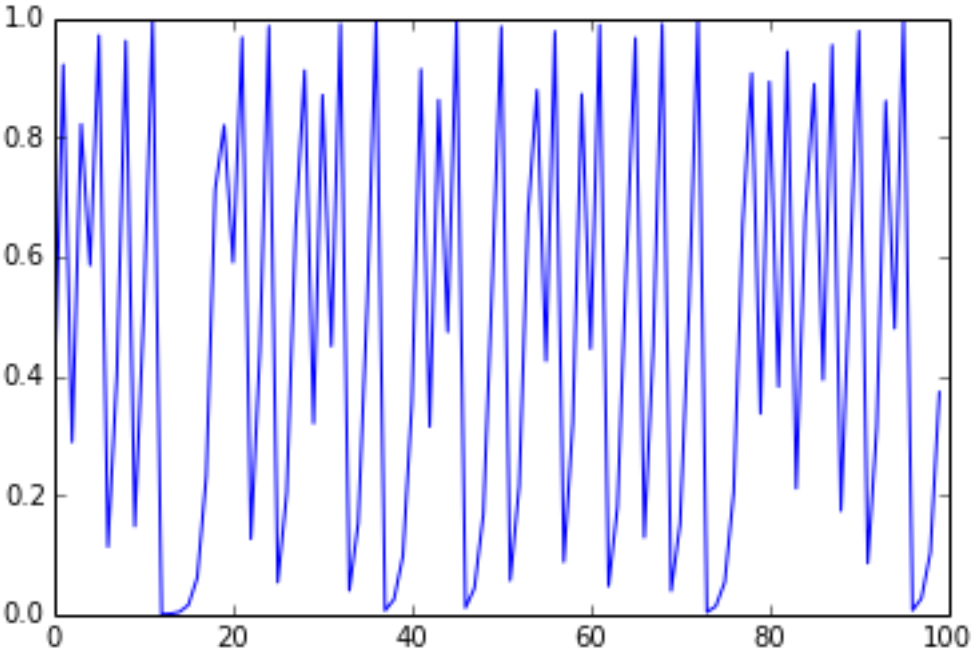
4 Logistische Gleichung

Die Logistische Gleichung ist ein Beispiel für eine deterministische Zeitserie

$$x_{n+1} = r x_n (1 - x_n)$$

Die Lösung dieser einfachen nicht-linearen Gleichung zeigt ein komplexes chaotisches Verhalten. Kleine Änderungen im Anfangszustand führen zu großen Abweichungen nach wenigen Zeitschritten. Trotz dieses chaotischen Verhaltens erhält man für exakt dieselben Anfangsbedindungen und Parameter exakt die gleiche Zeitreihe. Die Zeitserie ist deterministisch, da keine zufälligen Elemente vorhanden sind.

In [3]:
```
N=100
X=zeros(N)
x=0.1
r=4
for i in range(N):
    x=r*x*(1-x)
    X[i]=x
plot(X)

X.tofile('beispiele/beispiel_02.txt',sep=' ')
```

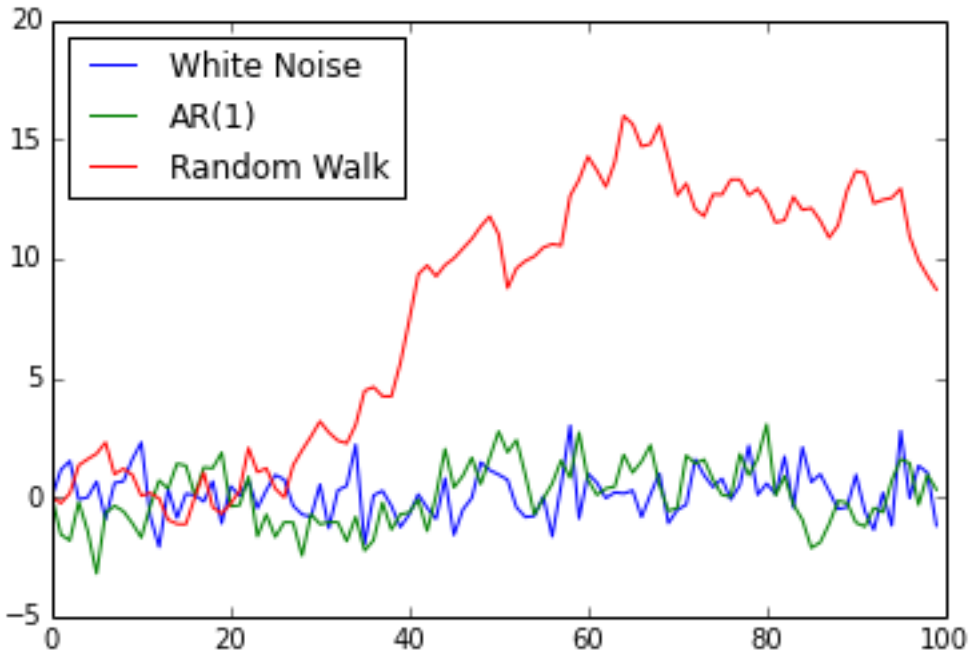


5 Stochastische Zeitreihen

Stochastische Zeitreihen bestehen aus Folgen von Zufallsvariablen. Bleiben dabei (für einen Zeitraum) stochastische Momente erhalten, spricht man von Stationarität (bezüglich dieser Momente und des Zeitraums). Sind Mittelwert über Zeit und Ensemble identisch, spricht man von einem ergodischen System.

Als stochastische Prozesse werden alle Rechenvorschriften bezeichnet, die eine stochastische Zeitreihe erzeugen.

```
In [4]: def stochastischer_prozess(N,alpha,beta):
            Y=zeros(N)
            for i in range(1,N):
                Y[i]=Y[i-1]*alpha+beta*randn(1)
            return Y

        N=100
        Y_white_noise=stochastischer_prozess(N,0.0,1.0)
        Y_AR1=stochastischer_prozess(N,0.7,1.0)
        Y_random_walk=stochastischer_prozess(N,1.0,1.0)

        plot(Y_white_noise, label='White Noise')
        plot(Y_AR1,label='AR(1)')
        plot(Y_random_walk,label='Random Walk')
        legend(loc=2)

        Y_white_noise.tofile('beispiele/beispiel_03.txt',sep=' ')
        Y_AR1.tofile('beispiele/beispiel_04.txt',sep=' ')
        Y_random_walk.tofile('beispiele/beispiel_06.txt',sep=' ') # 05.txt ist zufaellig aehnlich wie m
```



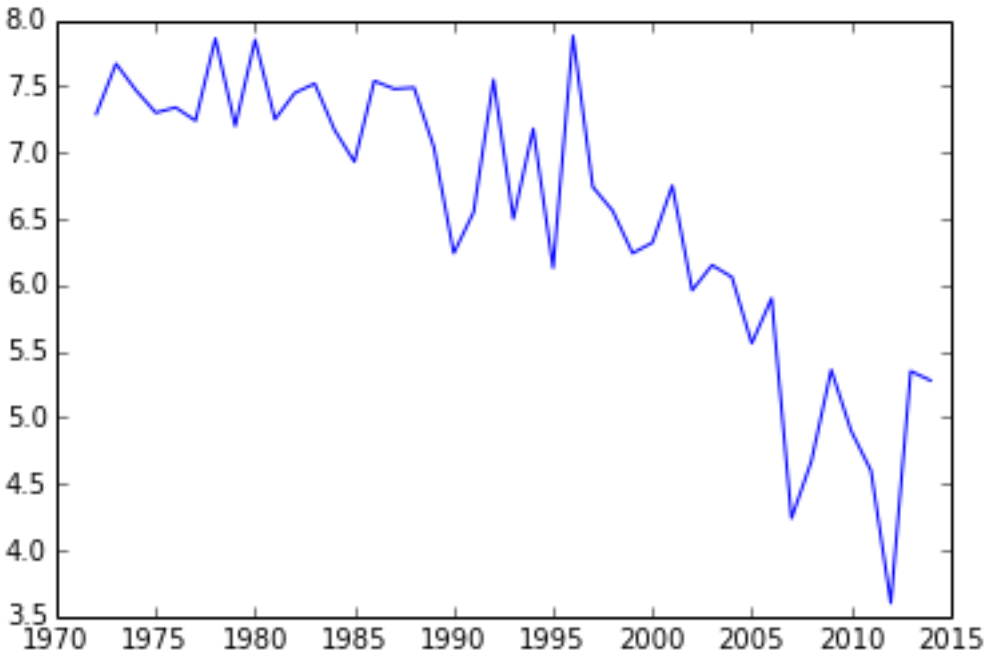
6 Geophysikalische Parameter

Unter geophysikalischen Parametern verstehen wir im englischen Sprachgebrauch Variablen, die in den Erdwissenschaften (inklusive Ozean, Atmosphäre etc.) vorkommen, nicht nur die der "Geophysik".

Geophysikalische Zeitreihen sind aufgrund der Messungenauigkeit und der Zufälligkeit immer stochastisch. Sie können aber manchmal durch deterministische Gleichungen und einen zufälligen Anteil beschrieben werden. Im einfachsten Fall ist die deterministische Gleichung ein lineares Modell zur Beschreibung des Trends. Oft ist der Trend von einem periodischen Jahresgang überlagert, welcher ebenfalls durch ein deterministisches Modell beschrieben werden kann.

6.1 Meereis-Ausdehnung im September

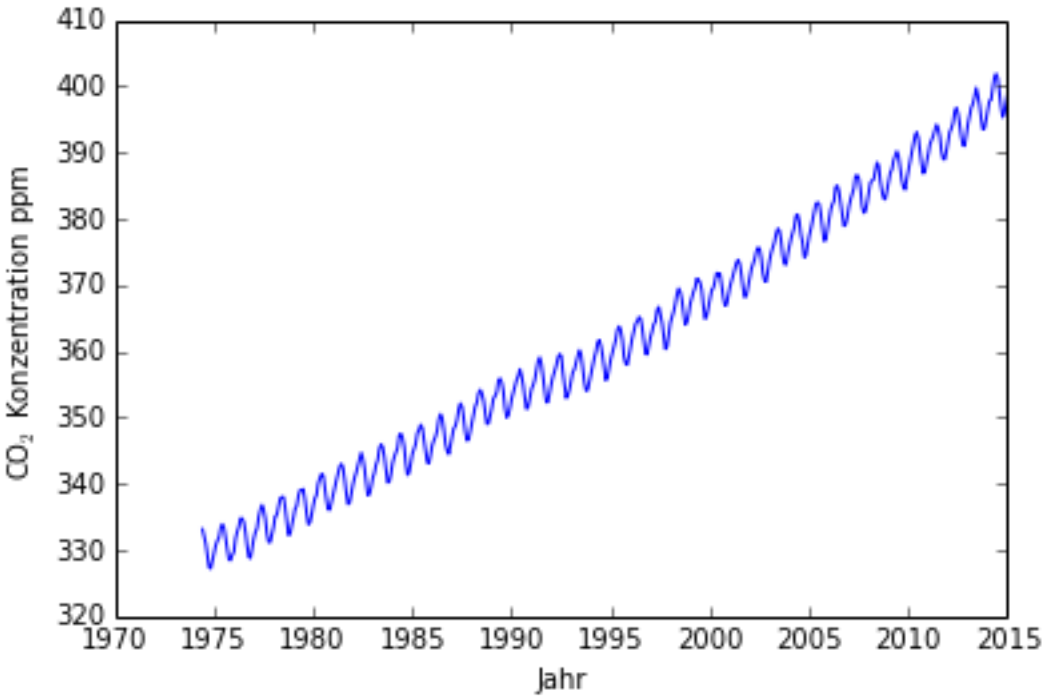
```
In [5]: D=loadtxt('data/september_extent_1972_2014.txt')
        t=D[:,0]
        X=D[:,1]
        plot(t,X)
        X.tofile('beispiele/beispiel_07.txt',sep=' ')
```



6.2 Mauna Loa CO_2

```
In [9]: #!wget  ftp://aftp.cmdl.noaa.gov/data/greenhouse_gases/co2/in-situ/surface/mlo/co2_mlo_surface-
        #!mv co2_mlo_surface-insitu_1_ccgg_MonthlyData.txt data
```

```
In [12]: D=loadtxt('data/co2_mlo_surface-insitu_1_ccgg_MonthlyData.txt',skiprows=128,usecols=(1,2,7))
         t=D[4:,0]+D[4:,1]/12.0 # Die ersten vier Einträge sind nicht gültig -999
         CO2=D[4:,2]
         plot(t,CO2)
         xlabel('Jahr')
         ylabel('CO$_2$ Konzentration ppm')
         CO2.tofile('beispiele/beispiel_08.txt',sep=' ')
```



7 Trend

Trend (vom englischen trend; aus mittelhochdeutsch: trendeln “kreiseln”, “nach unten rollen”) steht für:

- eine besonders tiefgreifende und nachhaltige Entwicklung (Soziologie)
- eine Entwicklung im mathematischen, statistischen, messtechnischen und wirtschaftlichen Bereich (Statistik)
- Kurstrend – eine Entwicklung von Börsenkursen

7.1 Was liegt im Trend?

Zwei Anfragen an Google-Trends:

- Beschreiben Sie die Kurven in Worten
- Was sind wohl die zugehörigen Suchanfragen?

http://www.tiobe.com/tiobe_index

8 Prognosen

- Wie macht man eine Prognose?

Prognose aus der Studie Die Grenzen des Wachstums veröffentlicht im Jahr 1972

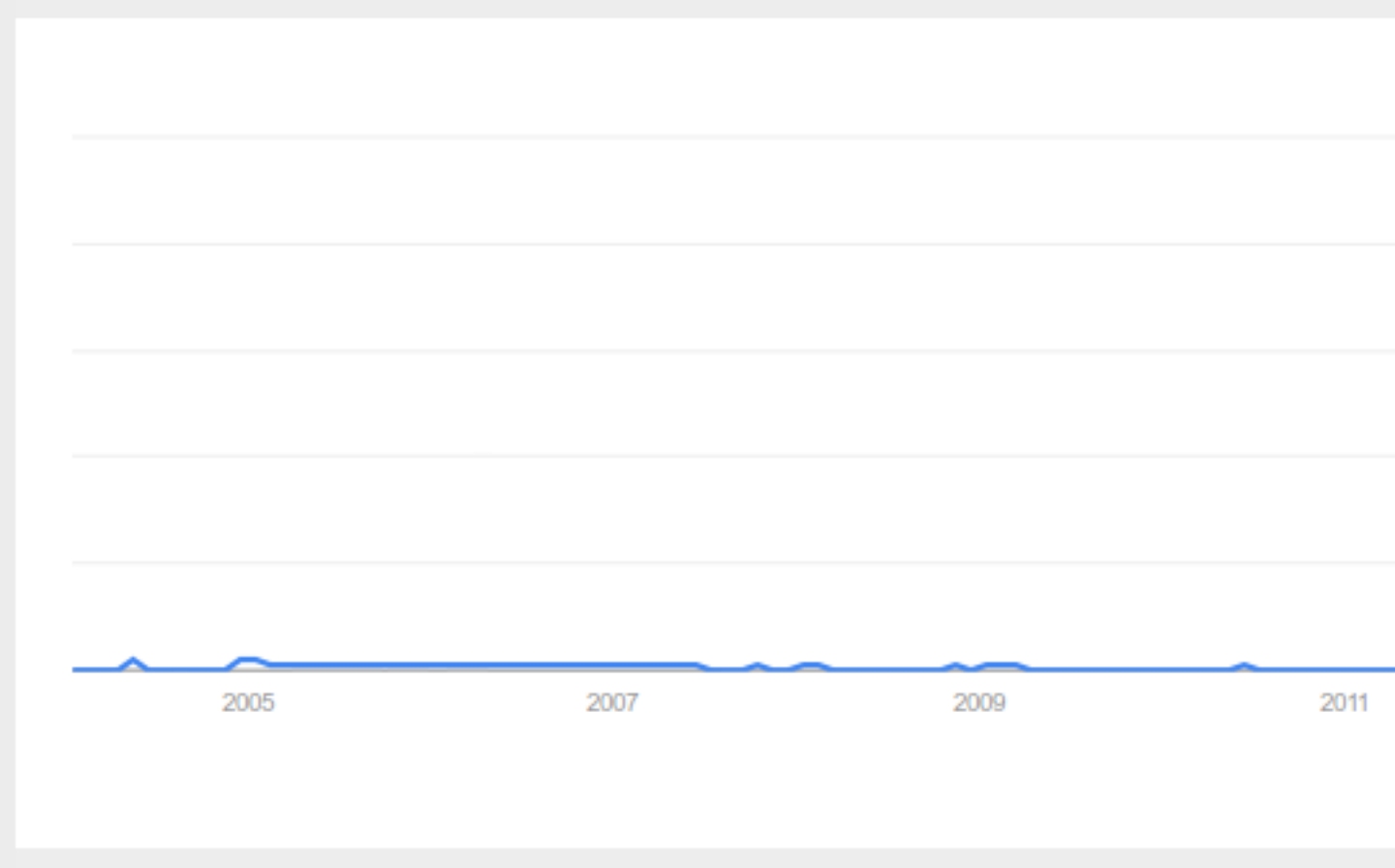


Figure 1: Jupyter

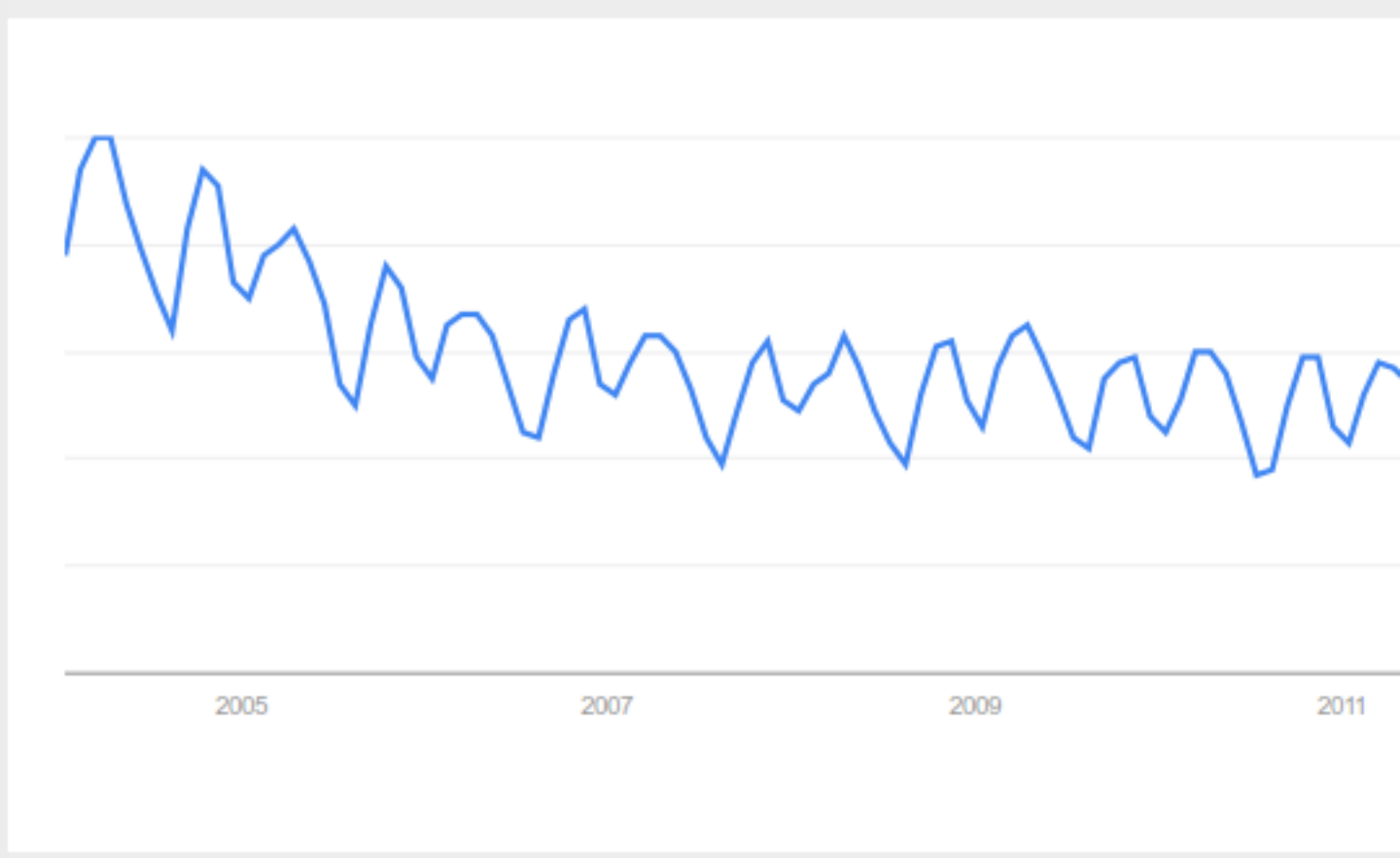
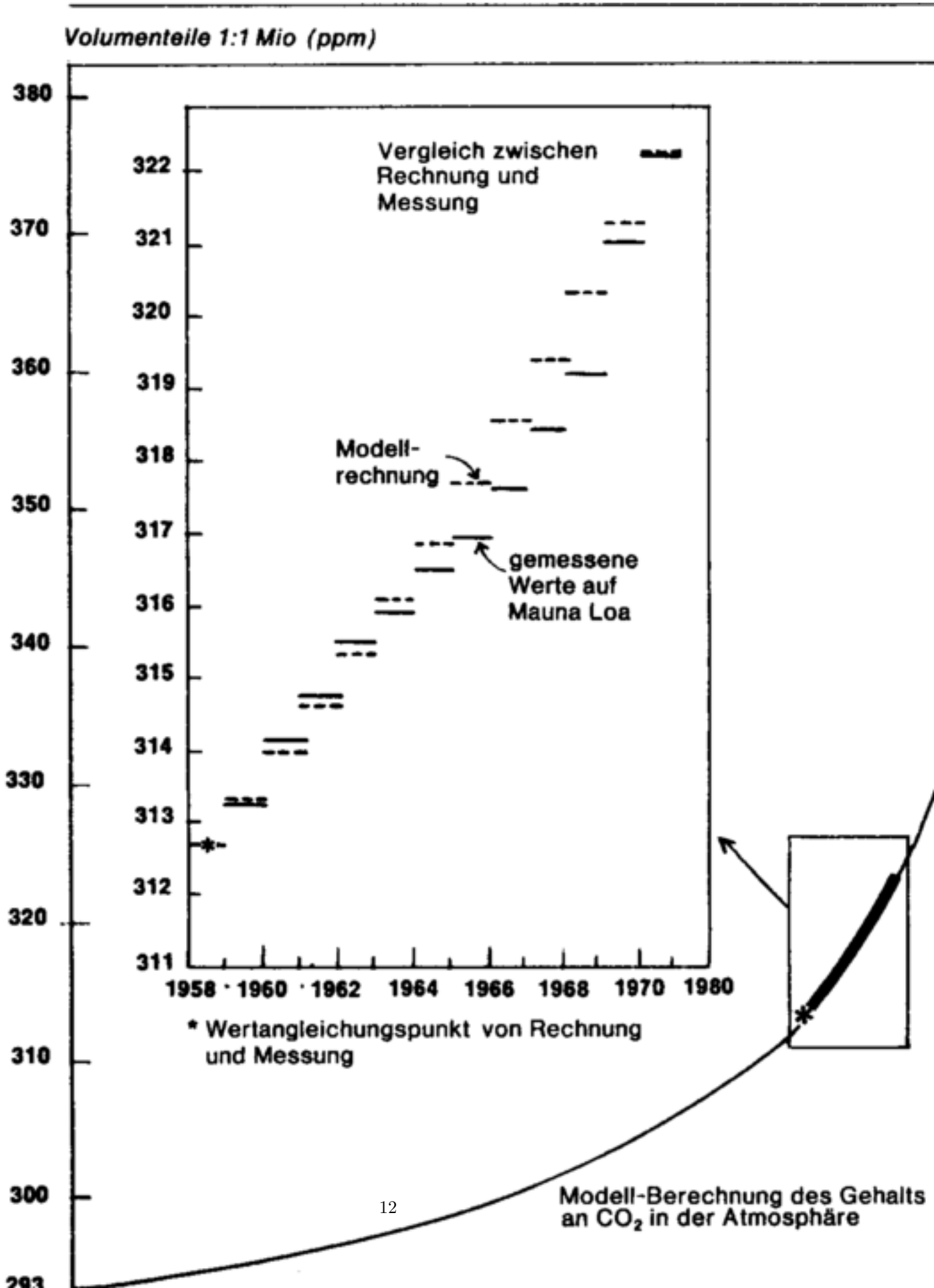


Figure 2: Matlab

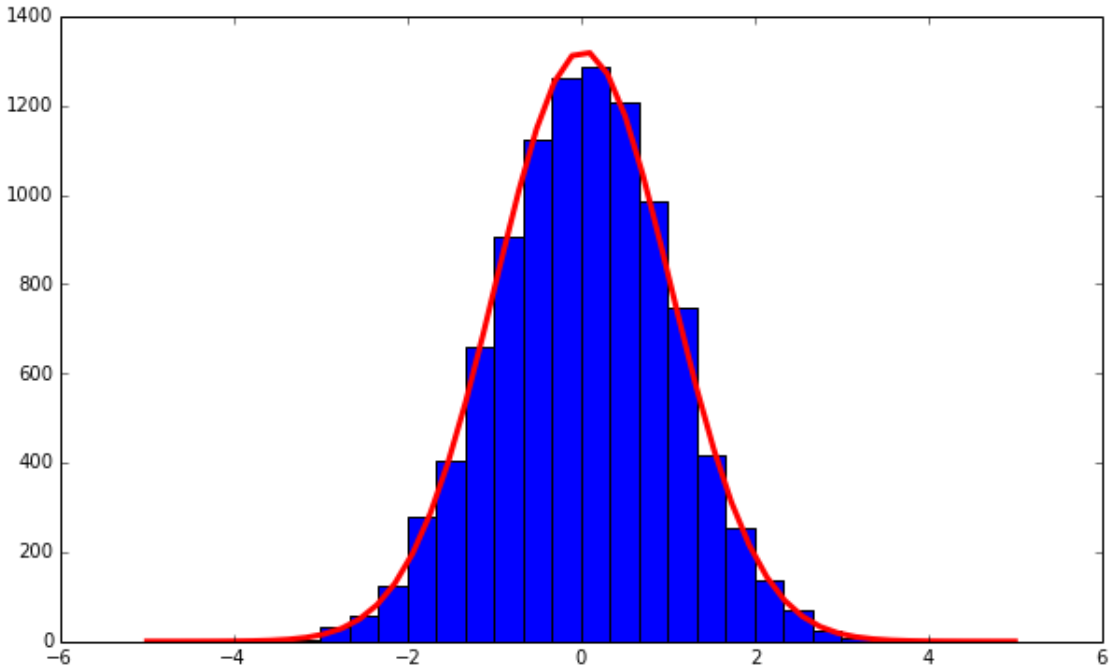

Volumenteile 1:1 Mio (ppm)
380
370
360
350
340
330
320
310
300
293
322
321
320
319
318
317
316
315
314
313
312
311
1958
1960
1962
1964
1966
1968
1970
1980
Vergleich zwischen Rechnung und Messung
Modell-rechnung
gemessene Werte auf Mauna Loa
* Wertangleichungspunkt von Rechnung und Messung
Modell-Berechnung des Gehalts an CO_2 in der Atmosphäre

9 Normalverteilung

```
In [2]: %pylab inline
        def Normalverteilung(x,mu,sigma):
            return 1.0/(sigma*sqrt(2*pi))*exp(-(1.0/2)*(x-mu)**2/sigma**2)

        N=10000
        x=randn(N)
        figure(figsize=(10,6))
        hist(x,bins=30,range=[-5,5])

        xn=linspace(-5,5)
        sigma=std(x)
        mu=mean(x)
        y=Normalverteilung(xn,mu,sigma)*N/(30/10)
        plot(xn,y,'r-',linewidth=3)
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
Out[2]: [<matplotlib.lines.Line2D at 0x7fee48099510>]
```



10 Übung 1

Wie viele Datenpunkte (in Prozent) liegen im Bereich $\pm 1\sigma$?

```
In [5]: print(100-(sum(x>sigma)+sum(x<-sigma))/N*100)
```

```
68.6
```

11 Mittelwert und Varianz

Berechnen sie Mittelwert $\mu = \frac{1}{N}\sum d_i$ und Varianz $\sigma^2 = \frac{1}{N-1}\sum(d_i - \mu)^2$ der drei Reihen in der Tabelle. Wie unterscheiden sich die Daten d_i in den Reihen?

```
In [3]: %pylab inline

        for i in range(3):
            d=loadtxt('data.txt')[i,:]
            N=size(d)
            mu=1.0/N*sum(d)
            v=1.0/(N-1)*sum((d-mu)**2)
            print 'Reihe ',i+1,': Mittelwert=', mu, ', Varianz=', v

Populating the interactive namespace from numpy and matplotlib
Reihe  1 : Mittelwert= 7.31428571429 , Varianz= 0.159285714286
Reihe  2 : Mittelwert= 7.31428571429 , Varianz= 0.159285714286
Reihe  3 : Mittelwert= 7.31428571429 , Varianz= 0.157285714286

In [4]: mean(d) # numpy-Funktion mean()

Out[4]: 7.3142857142857141

In [5]: var(d) # numpy-Funktion var() rechnet mit 1/N, nicht mit 1/(N-1)
        # 1/N ist die "unkorregierte" Stichprobenvarianz
        # 1/(N-1) ist die sogenannte "korregierte" (empirische) Stichprobenvarianz
        # Für große N  ist der Unterschied vernachlässigbar

Out[5]: 0.1497959183673469

In [ ]:
```

12 Grundlagen der Wahrscheinlichkeitsrechnung und Statistik

Dieser Abschnitt befasst sich mit den Grundlagen der Wahrscheinlichkeitstheorie in Hinblick auf die Zeitreihenanalyse. Zufallsvariablen sind darin Platzhalter für Messwerte eines Experiments. Wahrscheinlichkeitsdichteverteilungen beschreibt die theoretische Verteilungsfunktion. Realisierungen eines Experimentes werden als Stichproben bezeichnet. Aus einer Anzahl von Realisierungen ergibt sich die empirische Häufigkeitsverteilung.

12.1 Zufallsvariable

Sei $x(k)$ eine Menge von Zufallsvariablen. Die Zählvariable k bezeichnet ein bestimmtes Ereignis. Die Zufallsvariable $x(k)$ beschreibt eine Messung mit einem zufälligen Ergebnis. Der Ausgang eines N-mal wiederholten Experiments ist eine Reihe von Punkten bzw. Messwerten. Die Messwerte werden Stichproben (Samples) bzw. Realisierungen genannt. Die Anzahl der Stichproben wird hier mit N bezeichnet.

12.2 Wahrscheinlichkeitsverteilungs- und dichtefunktion

Die Wahrscheinlichkeitsverteilungsfunktion $P(x)$ wird definiert als die Wahrscheinlichkeit dafür, dass ein Ereignis $x(k)$ den Wert $x(k) \leq x$ annimmt.

$$P(x) = \text{Prob}(x(k) \leq x)$$

Es gilt

$$P(a) \leq P(b)$$

wenn $a \leq b$.

Der Wertebereich von P ist $[0, 1]$ weil gilt

$$P(-\infty) = 0, P(+\infty) = 1$$

Wenn der Wertebereich der Zufallsvariable $x(k)$ kontinuierlich ist, dann wird die Wahrscheinlichkeitsdichtefunktion $p(x)$ wie folgt definiert

$$p(x) = \lim_{\Delta x \to 0} \left(\frac{\text{Prob}(x < x(k) \leq x + \Delta x)}{\Delta x} \right)$$

Es gilt

$$p(x) \geq 0$$

$$\int_{-\infty}^{\infty} p(x)dx = 1$$

$$P(x) = \int_{-\infty}^{x} p(\zeta)d\zeta$$

$$\frac{dP(x)}{dx} = p(x)$$

Die Wahrscheinlichkeitsdichtefunktionen diskreter Zufallsvariablen, wie z.B. der Würfelfunktion, müssen mit Hilfe von Delta-Funktionen $\delta(x)$ oder als Summe beschrieben werden.

$$\text{Prob}(x < x(k) \leq x + \Delta x) = \sum_{x}^{x+\Delta x} p_k(x)$$

mit $p_k = \text{Prob}(x(k) = x_k)$.

Die Wahrscheinlichkeitsverteilungsfunktion $P(x)$ wird auch kumulative Verteilungsfunktion genannt. Sie ist definiert als das Integral über die Wahrscheinlichkeitsdichtefunktionen $p(x)$. Oft genutzte Abkürzungen sind PDF und CDF für Probability Density Function $p(x)$ und Cumulative (Probability) Density Function $P(x)$.

12.3 Erwartungswert

Der Erwartungswert einer Zufallsvariablen entspricht dem Mittelwert μ_x bei unendlicher Wiederholung eines Experiments. Der Erwartungswert errechnet sich aus der Zufallsvariablen mittels Gewichtung mit der Wahrscheinlichkeit

$$\text{E}(x(k)) = \int_{-\infty}^{\infty} xp(x)dx = \mu_x$$

Für diskrete Zufallsvariablen ist das Integral durch eine Summe zu ersetzen.

$$\text{E}(x) = \sum_{k} xp_k(x)$$

Die Varianz ist definiert als

$$\text{E}(x(k) - \text{E}(x(k))^2) = \sigma_x^2$$

12.3.1 Regeln

Der Erwartungswert ist ein linearer Operator

$$\mathrm{E}(aX_1 + bX_2) = a\,\mathrm{E}(X_1) + b\,\mathrm{E}(X_2)$$

12.3.2 Beispiel Zufallsvariable Würfel

Sei die Zufallsvariable $x(k)$ der Ausgang eines Würfelexperimentes.
Beim idealen Würfel sind alle sechs Seiten gleichwahrscheinlich.

$$P(x=1) = P(x=2) = P(x=3) = P(x=4) = P(x=5) = P(x=6) = \frac{1}{6}$$

Damit ergibt sich für den Erwartungswert

$$\mathrm{E}(x) = \sum_{k=1}^{6} x_k p_k(x_k)$$

mit $p_k(x) = \frac{1}{6}$ für alle $x = 1, 2, .., 6$

$$\mathrm{E}(x) = \frac{1}{6}(1+2+3+4+5+6) = 3.5$$

12.3.3 Übung

Berechnen Sie Mittelwert und Varianz der Zufallsvariable $x(k)$ "Würfel" theoretisch und experimentell mit einem Zufallsgenerator. Skizzieren Sie die kumulative Wahrscheinlichkeitsverteilungsfunktion und $P(x_k)$ Wahrscheinlichkeitsdichtefunktionen $p(x_k)$.

13 Kovarianz und Korrelation

Der Korrelationskoeffizient r beschreibt, wie eng zwei Zufallsvariablen in Raum oder Zeit zusammenhängen. Für zwei Zufallsvariablen $x = (x_1, x_2, ..., x_n)$ und $y = (y_1, y_2, ..., y_n)$ berechnet sich der Korrelationskoeffizient $r_{x,y}$ als

$$r_{x,y} = \frac{1}{N-1} \sum_{i=1}^{N} \frac{(x_i - \bar{x})(y_i - \bar{y})}{\sigma_x \sigma_y}$$

oder ausgedrückt durch die Kovarianz $C_{x,y}$

$$r_{x,y} = \frac{C_{x,y}}{\sigma_x \sigma_y}$$

dabei gilt

$$C_{x,y} = \mathrm{cov}(x,y) = \sigma_{x,y}^2 \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \bar{x})(y_i - \bar{y})$$

Der Korrelationskoeffizient errechnet sich aus der Kovarianz durch Normierung mit den Standardabweichungen σ_x und σ_y definiert durch

$$\sigma_x^2 = \sigma_x^2 = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \bar{x})^2$$

Äquivalent ist die Definition der Kovarianz durch den Erwartungswert

$$\mathrm{cov}(x,y) = \mathrm{E}\big[(x - \mathrm{E}(x)) \cdot (y - \mathrm{E}(y))\big]$$

Für statistisch unabhängige Zufallsvariablen x und y gilt $\mathrm{cov}(x,y) = 0$.

14 Normalverteilung (z-Verteilung)

Die Gaußverteilung ist gegeben durch

$$p(x) = \frac{1}{\sigma_x\sqrt{2\pi}} e^{-\frac{(x-\mu_x)^2}{2\sigma_x^2}}$$

Es gilt

$$E[x] = \mu_x$$

und

$$E[(x-\mu_x)^2] = \sigma_x^2$$

Substitution von $z = \frac{x-\mu_x}{\sigma_x}$ liefert die standardisierte Normalverteilung

$$p(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}}$$

für die gilt $E[x] = \mu_z = 0$ und $E[(x-\mu_z)^2] = \sigma_z^2 = 1$
Die Wahrscheinlichkeit P berechnet sich aus dem Integral

$$P(z_\alpha) = \int_{-\infty}^{z_\alpha} p(z)dz = Prob[z < z_\alpha] = 1 - \alpha$$

bzw.

$$1 - P(z_\alpha) = \int_{z_\alpha}^{\infty} p(z)dz = Prob[z > z_\alpha] = \alpha$$

14.1 Gaußsche Fehlerfunktion

Als Fehlerfunktion oder gaußsche Fehlerfunktion bezeichnet man in der Theorie der Speziellen Funktionen das Integral

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-\tau^2}\,\mathrm{d}\tau$$

Aus der Fehlerfunktion errechnet sich, wieviele Werte bei einer gegebenen Normalverteilung innerhalb eines Wertebereichs, z.B. $\pm 1\sigma$ zu erwarten sind.

14.1.1 Beispiel erf

```
In [9]: erf(1/sqrt(2)) % Octave/Matlab
```

```
Out[9]: 0.68268949213708585
```

```
In [5]: from scipy.special import erf
        from numpy import sqrt
        erf(1/sqrt(2))
```

```
Out[5]: 0.68268949213708585
```

```
In [6]: erf(2/sqrt(2))
```

```
Out[6]: 0.95449973610364158
```

```
In [7]: erf(3/sqrt(2))
```

```
Out[7]: 0.99730020393673979
```

Im Intervall $\pm\sigma$ sind 68,27% aller Zufallswerte zu erwarten, in $\pm 2\sigma$ sind es 95,45%, in $\pm 3\sigma$ erwarten wir 99,73%.

14.2 Parameter-Schätzungen

Das Grundproblem der statistischen Analyse besteht darin, dass die Stichprobenanzahl begrenzt ist und die theoretischen Verteilungsfunktionen nicht bekannt sind. Stattdessen müssen statistische Parameter aus einer begrenzten Anzahl Stichproben möglichst genau geschätzt werden.

Gegeben sei eine Zufallsvariable x_i, z.B. eine stochastische Zeitserie. Im Folgenden wird der Stichproben-Index i weggelassen, um eine kompaktere Darstellung zu erhalten. Die beiden statistischen Parameter Mittelwert μ und Varianz σ^2

$$\mu_x = E[x] = \int_{-\infty}^{\infty} xp(x)dx$$

$$\sigma_x^2 = E[(x-\mu_x)^2] = \int_{-\infty}^{\infty} (x-\mu_x)^2 p(x)dx$$

werden mit Hilfe der Wahrscheinlichkeitsdichtefunktion $p(x)$ theoretisch berechnet. Der Operator $E[]$ bezeichnet den Erwartungswert. Im Allgemeinen ist die Anzahl der Stichproben begrenzt und die Wahrscheinlichkeitsdichtefunktion ist nicht bekannt.

Eine (von vielen) Möglichkeiten um Mittelwert und Varianz von x aus N unabhängigen Messungen zu schätzen, ist gegeben durch

$$\bar{x} = \hat{\mu_x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$s_b^2 = \hat{\sigma_x}^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2$$

Der Unterschied von $\hat{\mu_x}$ und μ_x besteht darin, dass $\hat{\mu_x}$ den geschätzen Wert und μ_x den wahren bzw. theoretisch richtigen Wert bezeichnet.

Wir wollen nun untersuchen, ob die Schätzung von Mittelwert und Varianz unsere Erwartungen erfüllt. Der Erwartungswert des Mittelwertes berechnet sich aus (Bendat und Piersol, 1986):

$$E[\bar{x}] = E\left[\frac{1}{N}\sum_{i=1}^{N} x_i\right] = \frac{1}{N}E\left[\sum_{i=1}^{N} x_i\right] = \frac{1}{N}(N\mu_x) = \mu_x$$

und erfüllt unsere Erwartungen $\hat{\mu_x} = \bar{x}$ (erwartungstreu).

Der mittlere quadratische Fehler der Schätzung des Mittelwertes ergibt sich aus

$$E[(\bar{x}-\mu_x)^2] = E\left[\left(\frac{1}{N}\sum_{i=1}^{N} x_i - \mu_x\right)^2\right] = \frac{1}{N^2}E\left[\left(\sum_{i=1}^{N}(x_i-\mu_i)\right)^2\right]$$

Die Messungen x_i sind unabhängig, daher folgt

$$E[(\bar{x}-\mu_x)^2] = \frac{1}{N^2}E\left[\sum_{i=1}^{N}(x_i-\mu_i)^2\right] = \frac{1}{N^2}(N\sigma_x^2) = \frac{\sigma_x^2}{N}$$

Der Erwartungswert der Varianz s_b errechnet sich zu

$$E[s_b^2] = E[\frac{1}{N}\sum_{i=1}^{N}(x_i-\bar{x})^2] = \frac{1}{N}E[\sum_{i=1}^{N}(x_i-\bar{x})^2] =$$

mit $\sum_{i=1}^{N}(x_i-\bar{x})^2 = ... = \sum_{i=1}^{N}(x_i-\bar{x})^2 - N(x_i-\bar{x})^2$ und $E[(x_i-\mu_x)^2] = \sigma_x^2$ und $E[(\bar{x}-\mu_x)^2] = \frac{\sigma_x^2}{N}$ folgt (Herleitung siehe Bendat und Piersol, 1986)

$$E[s_b^2] = \frac{N-1}{N}\sigma_x^2$$

Die Varianz-Schätzung s_b bzw. $\hat{\sigma_x}$ ist offensichtlich nicht erwartungstreu, da $\hat{\sigma_x}^2 < \sigma_x^2$. Die Schätzung nennt sich darum verzerrt oder biased. Eine erwartungstreue/unverzerrte Schätzung der Varianz ist gegeben durch

$$s^2 = \hat{\sigma_x}^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2$$

der sogenannten empirischen Varianz oder Stichprobenvarianz. Der Unterschied macht sich insbesondere für eine kleine Anzahl von Stichproben bemerkbar und kann für eine große Anzahl meist vernachlässigt werden. Vorsicht ist jedoch geboten bei Hypothesentests, die auf einem Vergleich der Varianz basieren.

15 Hypothesenprüfungen

Bei einem statistischen Prüfverfahren wird einer sogenannten Nullhypothese eine (oder mehrere) Alternativhypothese gegenübergestellt. Im Sinne eines mathemathischen Widerspruchsbeweises wird die Nullhypothese statistisch widerlegt, um die Alternativhypothese zu beweisen.

Bei der Auswahl eines Prüfverfahrens muss die theoretische Wahrscheinlichkeitsverteilung und der Stichprobenumfang beachtet werden.

Das Ergebnis einer statistischen Prüfung ist immer im Zusammenhang mit dem sog. Signifikanzniveau zu nennen. Die zur Signifikanz komplementäre Größe ist die Irrtumswahrscheinlichkeit. Es gilt

$$Signifikanz = 1 - Irrtumswahrscheinlichkeit$$

oder

$$Si = 1 - \alpha$$

Wahrscheinlichkeiten werden entweder in Prozent oder als Zahl zwischen 0-1 angeben. Anstatt α wird üblicherweise auch die Bezeichnung p verwendet. Die Regel besagt, dass die Signifikanz groß ist, wenn p klein ist.

Aus der Angabe einer Wahrscheinlichkeit ergibt sich auch ein sog. Vertrauensbereich (auch Konfidenzintervall oder Mutungsbereich), der den Wertebereich für eine spezifische Wahrscheinlichkeit angibt.

15.1 Beschreibung von Wahrscheinlichkeitsbereichen (IPCC-Terminologie)

Bei einer Einschätzung der Unsicherheit bestimmter Ergebnisse mittels fachkundiger Beurteilung und statistischer Analyse eines Beweises (z.B. Beobachtungen oder Modellergebnisse) werden folgende Wahrscheinlichkeitsbereiche verwendet, um die geschätzte Eintrittswahrscheinlichkeit auszudrücken:

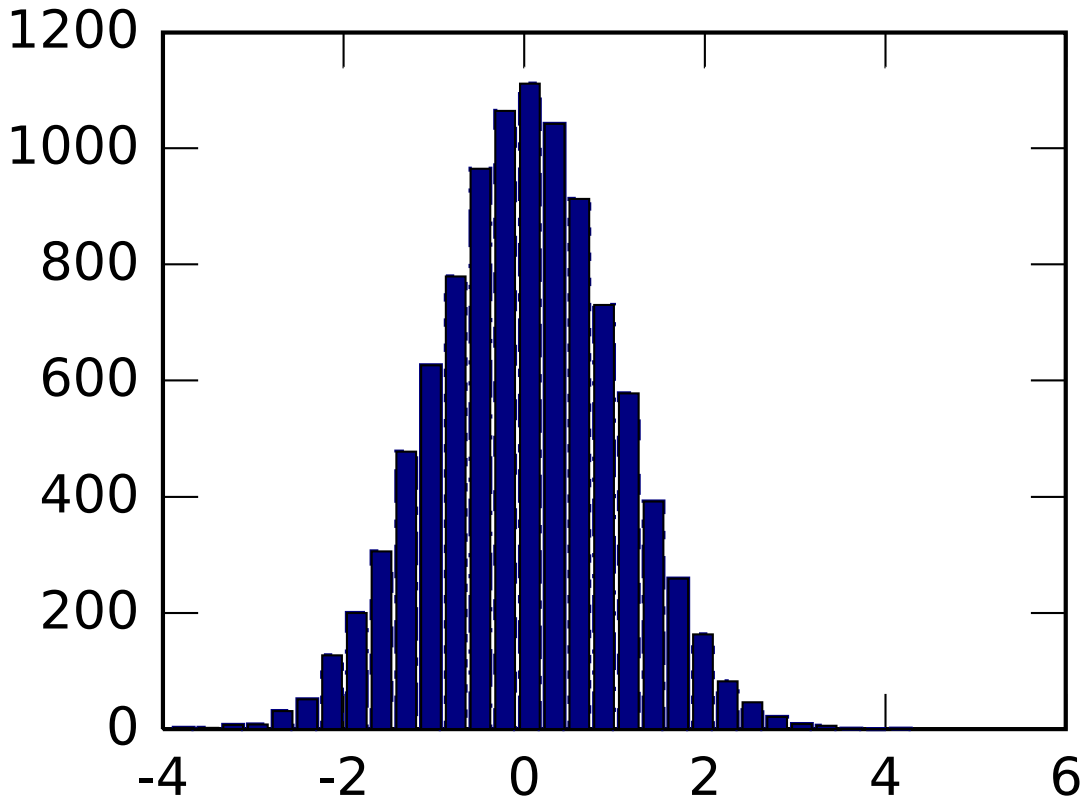
- praktisch sicher >99%
- höchst wahrscheinlich >95%
- sehr wahrscheinlich >90%
- wahrscheinlich >66%
- wahrscheinlicher als nicht >50%
- etwa so wahrscheinlich wie nicht 33% bis 66%
- unwahrscheinlich <33%
- sehr unwahrscheinlich <10%
- höchst unwahrscheinlich <5%
- außergewöhnlich unwahrscheinlich <1%

Übernommen aus IPCC AR4 (2007)

16 Histogramm

Ein Histogramm ist eine graphische Darstellung der Häufigkeitsverteilung.

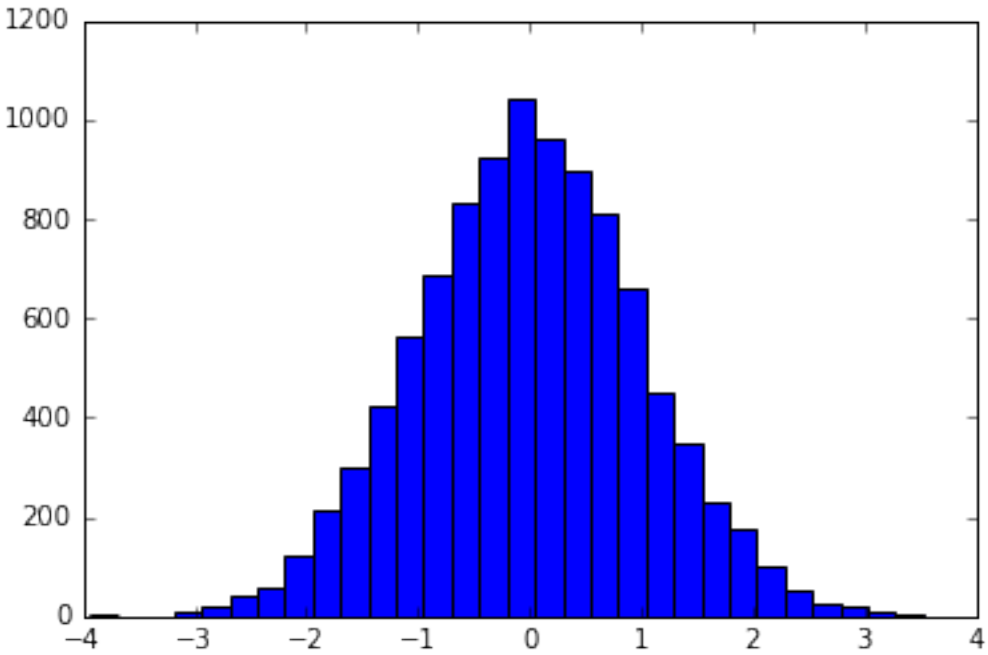
```
In [3]: N=10000
        x=randn(N, 1);
        bins=30;
        hist(x,bins);
```




```
Out[3]: N =  10000
```

```
In [3]: %pylab inline
        N=10000
        x=randn(N, 1)
        bins=30
        h=hist(x,bins)
```

```
Populating the interactive namespace from numpy and matplotlib
```



17 Gaußsche Fehlerfunktion

Als Fehlerfunktion oder gaußsche Fehlerfunktion bezeichnet man in der Theorie der Speziellen Funktionen das Integral

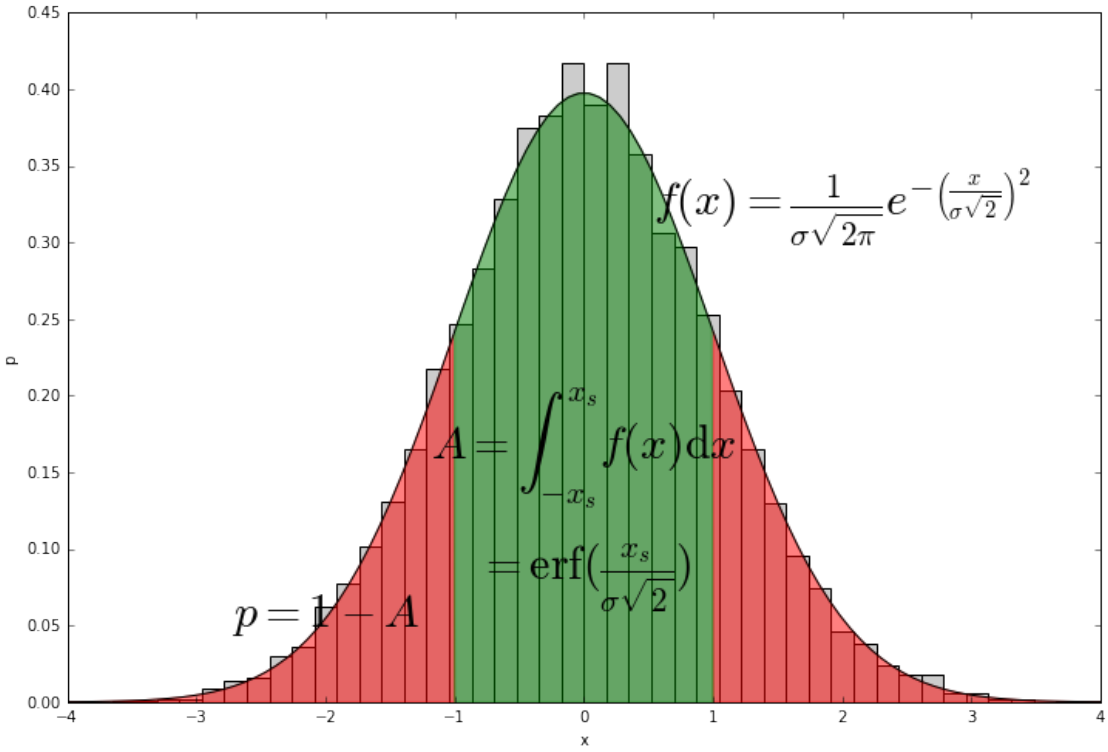
$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-\tau^2} \, \mathrm{d}\tau$$

In [9]:
```
%pylab inline

from scipy.special import erf # Gaußsche Fehlerfunktion (Integralfunktion)

def gaussian(x,sigma):
    return 1.0/(sqrt(2*pi)*sigma)*exp(-(x/sigma)**2/2)


N=10000

rd=randn(N)
sig=rd.std()

bi=int(5*log(N))
rrange=4

fig=figure(figsize=(12,8))
ax=fig.add_subplot(1,1,1)

n, bins, patches = hist(rd, bins=bi, range=[-rrange,rrange], normed=True, histtype='bar',color=
```

```
xg=linspace(-rrange,rrange,100)

G=gaussian(xg,sig)
plot(xg,G,'k',linewidth=1)

S=1

# make the shaded region
a=-rrange
b=-S
ix = arange(a, b, 0.01)
iy = gaussian(ix,sig)
verts = [(a,0)] + list(zip(ix,iy)) + [(b,0)]
poly = Polygon(verts, facecolor='r', edgecolor='r',alpha=0.5)
ax.add_patch(poly)
text(-2, 0.05,
     r"$p=1-A$", horizontalalignment='center',
     fontsize=30)
a=S
b=rrange
ix = arange(a, b, 0.01)
iy = gaussian(ix,sig)
verts = [(a,0)] + list(zip(ix,iy)) + [(b,0)]
verts = [(a,0)] + list(zip(ix,iy)) + [(b,0)]

poly = Polygon(verts, facecolor='r', edgecolor='r',alpha=0.5)
ax.add_patch(poly)


a=-S
b=S
ix = arange(a, b, 0.01)
iy = gaussian(ix,sig)
verts = [(a,0)] + list(zip(ix,iy)) + [(b,0)]
poly = Polygon(verts, facecolor='g', edgecolor='g',alpha=0.5)
ax.add_patch(poly)
rmax=G.max()
text(0.5 * (a + b), 0.4*rmax,
     r"$A=\int_{-x_s}^{x_s} f(x)\mathrm{d}x$", horizontalalignment='center',
     fontsize=30)

text(0.5 * (a + b), 0.2*rmax,
     r"$=\mathrm{erf}(\frac{x_s}{\sigma \sqrt{2}})$", horizontalalignment='center',
     fontsize=30)


text(2.0,0.8*rmax,
     r"$f(x)=\frac{1}{\sigma\sqrt{2\pi}} e^{ -\left( \frac{x}{\sigma\sqrt{2}} \right)^2 }$", ho
     fontsize=30)

xlabel('x')
ylabel('p')

print('Auswertung numerisches Experiment (Zufallsgenerator ) N=',N)
```

```
        print("-------------------------------------------------------------------------------------------
        s=1
        print("sigma=",s,",  theoretische Anzahl Werte innerhalb +-sigma: ",erf(s/sqrt(2))*100, "%")
        print("sigma=",s,",  experimentelle Anzahl Werte innerhalb +-sigma: ", 100-float(sum(rd>s)+sum(
        print("-------------------------------------------------------------------------------------------
        savefig('Normalverteilung.png',dpi=150)
```

```
Populating the interactive namespace from numpy and matplotlib
Auswertung numerisches Experiment (Zufallsgenerator ) N= 10000
-----------------------------------------------------------------------------------------------------
sigma= 1 ,  theoretische Anzahl Werte innerhalb +-sigma:  68.2689492137 %
sigma= 1 ,  experimentelle Anzahl Werte innerhalb +-sigma:  68.59 %
-----------------------------------------------------------------------------------------------------

WARNING: pylab import has clobbered these variables: ['poly']
'%matplotlib' prevents importing * from pylab and numpy
```



In [8]: `!convert -trim Normalverteilung.png Normalverteilung_plot.png`

In [ ]:

18 Wie überprüft man die Signifikanz?

Zur Überprüfung der Signifikanz von Trends und Korrelationen wird üblicherweise versucht, die Nullhypothese zu widerlegen. Die Nullhypothese besagt, dass sich der Trend bzw. die Korrelation mit einer

gewissen Wahrscheinlichkeit $1 - p$ aus einer zufälligen Zeitreihe ergibt. Nur wenn es unwahrscheinlich ist, dass die Nullhypothese zutrifft, ist die Beziehung als signifikant zu betrachten.

Als Monte-Carlo Methode bezeichnet man die Erzeugung von zufälligen Eingangs-Daten zur numerischen Datenanalyse. Mittels Monte-Carlo Methode können statistische Hypothesen-Tests durchgeführt werden. Indem ein numerisches Experiment wiederholt durchgeführt wird, kann die Anzahl von Ereignissen (z.B. die Korrelation liegt über einem bestimmten Wert) gezählt werden. Diese Vorgehensweise erlaubt einen intuitiven Zugang zur Statistik ohne den Gebrauch theoretischer Verteilungsfunktionen.

18.1 Monte-Carlo Methode

```
In [22]: %pylab inline
         d=loadtxt('september_extent_1972_2014.txt')[0:21,1] # 21 Jahre von 1972-1992
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
In [15]: d
```

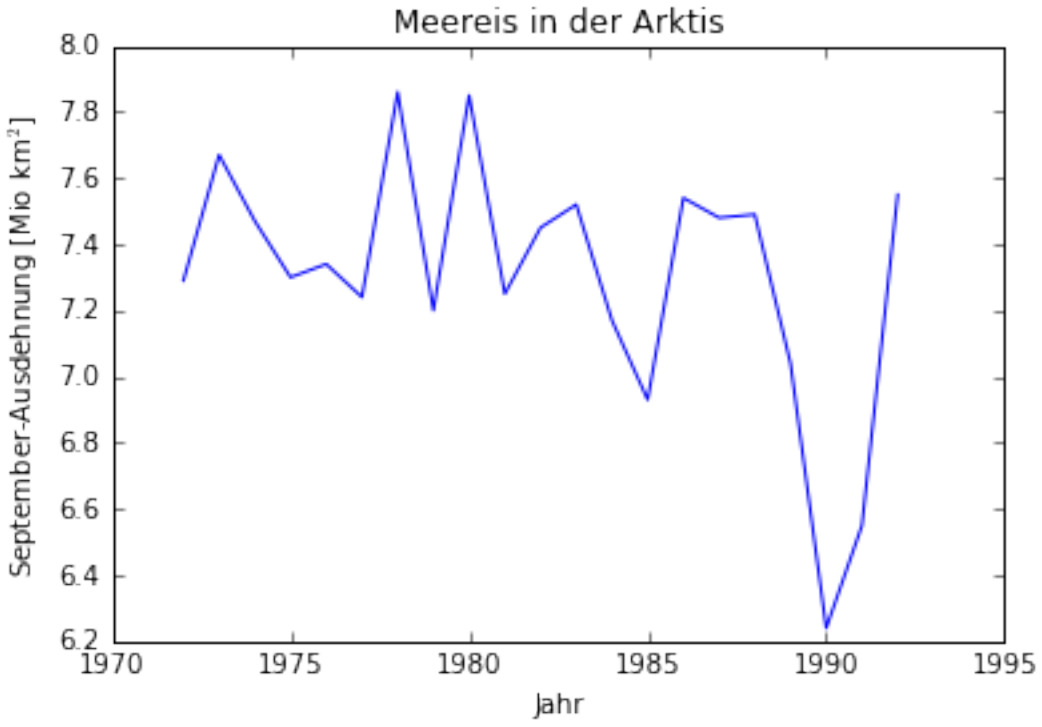
```
Out[15]: array([ 7.29,  7.67,  7.47,  7.3 ,  7.34,  7.24,  7.86,  7.2 ,  7.85,
                 7.25,  7.45,  7.52,  7.17,  6.93,  7.54,  7.48,  7.49,  7.04,
                 6.24,  6.55,  7.55])
```

```
In [12]: size(d)
```

```
Out[12]: 21
```

```
In [16]: x=linspace(1972,1992,size(d))
         plot(x,d)
         xlabel('Jahr')
         ylabel('September-Ausdehnung [Mio km$^2$]')
         title('Meereis in der Arktis')
```

```
Out[16]: <matplotlib.text.Text at 0x7f234e48dac8>
```



```
In [17]: print(d)
         print(mean(d),var(d))
```

```
[ 7.29  7.67  7.47  7.3   7.34  7.24  7.86  7.2   7.85  7.25  7.45  7.52
  7.17  6.93  7.54  7.48  7.49  7.04  6.24  6.55  7.55]
7.30619047619 0.141233106576
```

```
In [18]: shuffle(d) # Zufällige Umsortierung, Mittelwert und Varianz bleiben erhalten
         print(d)
         print(mean(d),var(d))
```

```
[ 7.24  7.25  7.04  7.45  7.52  7.55  7.54  7.17  6.24  7.86  7.3   7.29
  7.2   7.47  7.48  7.49  6.55  7.85  6.93  7.67  7.34]
7.30619047619 0.141233106576
```

```
In [19]: d2=randn(21)
         d2=d2*std(d)/std(d2) # Normiere mit Standardabweichung
         d2=d2+mean(d-d2) # Anpassen von Mittelwert
         print('Ursprüngliche Daten:')
         print(d)
         print(mean(d),var(d))
         print('-------------------------------------------------')
         print('Zufällige Daten mit gleichen Mittelwert und Varianz')
         print(d2)
         print(mean(d2),var(d2))
         d2=(d2*10.0).astype(int)/10.0
         print('-------------------------------------------------')
         print('Zufällige Daten (gerundet) mit ähnlichen Mittelwert und Varianz')
         print(d2)
         print(mean(d2),var(d2))
         print('-------------------------------------------------')
```

```
Ursprüngliche Daten:
[ 7.24  7.25  7.04  7.45  7.52  7.55  7.54  7.17  6.24  7.86  7.3   7.29
  7.2   7.47  7.48  7.49  6.55  7.85  6.93  7.67  7.34]
7.30619047619 0.141233106576
-------------------------------------------------
Zufällige Daten mit gleichen Mittelwert und Varianz
[ 6.99945432  7.44317147  6.892913    7.33905714  7.00154378  7.25316664
  7.57096758  7.34487733  6.8197436   7.50342683  7.42523574  7.17170962
  6.43933415  7.56857043  7.4546292   7.87374163  7.60624502  7.66340624
  6.93263709  7.03746452  8.08870466]
7.30619047619 0.141233106576
-------------------------------------------------
Zufällige Daten (gerundet) mit ähnlichen Mittelwert und Varianz
[ 6.9  7.4  6.8  7.3  7.   7.2  7.5  7.3  6.8  7.5  7.4  7.1  6.4  7.5  7.4
  7.8  7.6  7.6  6.9  7.   8. ]
7.25714285714 0.13768707483
-------------------------------------------------
```

18.2 Sind zwei Mittelwerte verschieden?

Um zu überprüfen, ob möglicherweise ein Trend vorliegt, vergleichen wir zunächst zwei Mittelwerte. Sind die beiden Zeitabschnitte signifikant verschieden, könnte ein Trend vorliegen:

- 1972-1982
- 1982-1992

```
In [23]: print(mean(d[0:10]))
         print(mean(d[10:20]))
```

```
7.447
7.141
```

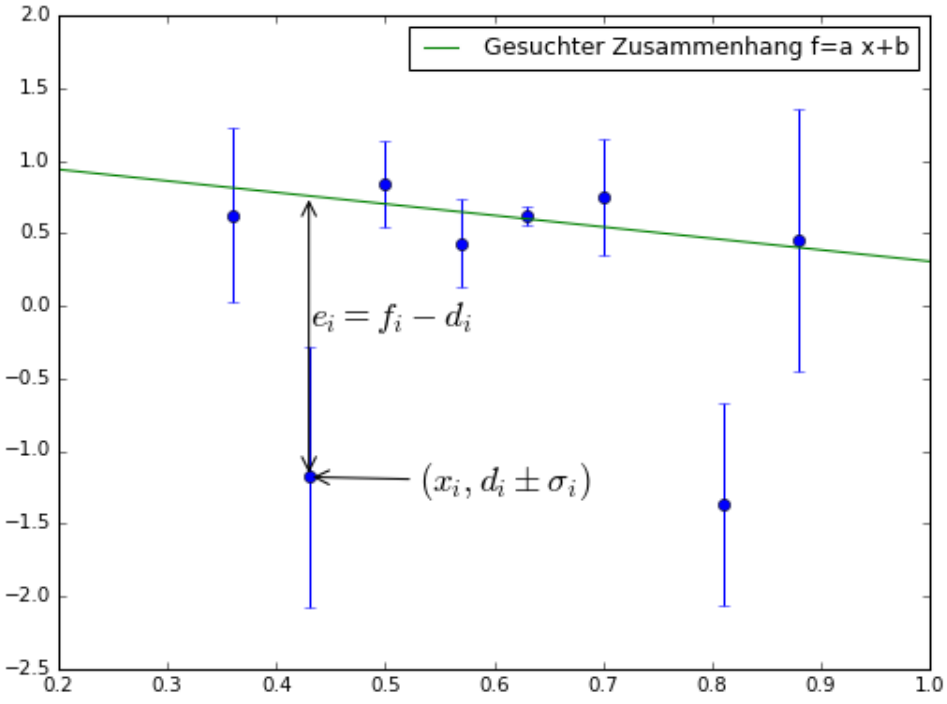
18.2.1 Aufgabe

Die zwei Mittelwerte sind verschieden. Doch ist das Ergebnis auch signifikant? Könnte es rein zufällig sein? Testen Sie die Signifikanz des Unterschieds. Formulieren Sie eine Nullhypothese. Setzen sie die Monte-Carlo Methode ein und beschreiben Sie den Unterschied quantitativ.

19 Lineare Regression

```
In [5]: from IPython.display import Image
        Image(filename=('Linreg_Beispiel.png'))
```

```
Out[5]:
```



19.1 Problemstellung:

Für gegebene Daten $d_i \pm \sigma_i$ an den Stellen x_i $(i = 1..N)$ ist die lineare Funktion f ("Ausgleichsgerade")

$$f = ax + b$$

zu bestimmen, die die Daten "möglichst gut" repräsentiert. Dabei soll die Kenntnis über die Fehler berücksichtigt werden, d.h. "schlechtere" Daten weniger berücksichtigt werden.

Der Fehler e_i für die einzelnen Datenpunkte d_i ist gegeben durch die Abweichung zur Modellgeraden

$$e_i = f_i - d_i$$

Es liegt ein überbestimmtes Gleichungssystem vor mit den zwei unbekannten Parametern a und b und $N > 2$ bekannten Variablen.

19.2 Formalisierung des Problems (Least-Squares Methode):

Die Methode der Summe der quadratischen Abweichungen (Least-Squares-Methode) ist eine mathematische Optimierung. Bei einer Optimierung geht es um das Finden von Minima oder Maxima von Zielfunktionen. Die Zielfunktion wird auch als Fehlerfunktion oder Kostenfunktion bezeichnet.

Die Güte eines Fits (die Fehlerfunktion F) ergibt sich aus der Summe der quadratischen Abweichungen zwischen den Daten und der zu optimierenden Funktion

$$F(a,b) = \sum_{i=1}^{N} \frac{e_i^2}{\sigma_i^2}$$

19.2.1 Ziel

Gesucht sind die speziellen Parameter $\hat{a}, \hat{b}$ für die die Fehlerfunktion $F(\hat{a}, \hat{b})$ minimal wird. Diese gesuchten Parameter führen zu einer optimalen Anpassung an die Daten.

19.2.2 Methode

Im Minimum von F muss gelten:

$$\frac{\partial F}{\partial a} = 0$$

$$\frac{\partial F}{\partial b} = 0$$

$$F(a,b) = \sum_{i=1}^{N} \frac{e_i^2}{\sigma_i^2} = \sum_{i=1}^{N} \frac{(ax_i + b - d_i)^2}{\sigma_i^2} = \sum_{i=1}^{N} \frac{1}{\sigma_i^2}(a^2x_i^2 + 2abx_i - 2ad_ix_i + b^2 - 2bd_i + d_i^2)$$

19.2.3 Bestimmungsgleichungen für Minimum der quadratischen Abweichung (optimaler Fit)

Aus den Ableitungen folgen die Gleichungen für die Bestimmung der optimalen Ausgleichsgeraden $f = \hat{a}x + \hat{b}$

$$\frac{\partial F}{\partial a} = 0 = \underbrace{(\sum_{i=1}^{N} \frac{x_i^2}{\sigma_i^2})}_{\alpha}\hat{a} + \underbrace{(\sum_{i=1}^{N} \frac{x_i}{\sigma_i^2})}_{\beta}\hat{b} - \underbrace{(\sum_{i=1}^{N} \frac{x_i d_i}{\sigma_i^2})}_{\gamma}$$

$$\frac{\partial F}{\partial b} = 0 = \underbrace{(\sum_{i=1}^{N} \frac{x_i}{\sigma_i^2})}_{\beta}\hat{a} + \underbrace{(\sum_{i=1}^{N} \frac{1}{\sigma_i^2})}_{\epsilon}\hat{b} - \underbrace{(\sum_{i=1}^{N} \frac{d_i}{\sigma_i^2})}_{\delta}$$

Vereinfachen wir mit $\alpha = \sum \frac{x_i^2}{\sigma_i^2}$, $\beta = \sum \frac{x_i}{\sigma_i^2}$, $\gamma = \sum \frac{x_i d_i}{\sigma_i^2}$, $\delta = \sum \frac{d_i}{\sigma_i^2}$ und $\epsilon = \sum \frac{1}{\sigma_i^2}$, so folgt

$$\alpha\hat{a} + \beta\hat{b} = \gamma$$

$$\beta\hat{a} + \epsilon\hat{b} = \delta$$

Durch Umformen und Einsetzen erhalten wir die Berechnungsformeln für $\hat{a}$ und $\hat{b}$

$$\hat{b} = \frac{\alpha\delta - \beta\gamma}{\alpha\epsilon - \beta^2}$$

$$\hat{a} = \frac{\gamma - \beta\hat{b}}{\alpha}$$

20 Aufgabe lineare Regression

Gegeben sind die folgenden Datenpunkte $d_i \pm \sigma_i$ an den Stellen x_i $(i = 1..8)$

Programmieren Sie eine Funktion ausgleichsgerade(x,d,sigma), welche die Regressionsparameter $\hat{a}$ und $\hat{b}$ berechnet.

In [2]:
```
%pylab inline
from ausgleichsgerade import ausgleichsgerade

# Zufällige Beispieldaten
x=array([ 0.63,  0.81,  0.36,  0.43,  0.70, 0.57 ,  0.50,  0.88])
d=array([ 0.62, -1.37,  0.62, -1.18,  0.75, 0.43 ,  0.84,  0.45])
sigma=array([0.06,  0.7,  0.6,  0.9,  0.4, 0.3,  0.3 ,  0.9])

# Diese Funktion gilt es zu programmieren
a,b=ausgleichsgerade(x,d,sigma)

# Erzeuge neue X und Y Werte zum Plotten der Ausgleichsgerade
X=linspace(0.2,1)
Y=a*X+b

# Graphische Ausgabe, siehe oben
figure(figsize=(8,6))
errorbar(x,d,yerr=sigma,fmt='o')
plot(X,a*X+b,'g-',label='Gesuchter Zusammenhang f=a x+b')
i=3
annotate('$(x_i,d_i\pm \sigma_i)$',xy=(x[i], d[i]), \
         arrowprops=dict(arrowstyle='->'), xytext=(x[i]+0.1, d[i]-0.1),fontsize=18)
annotate('',xy=(x[i], d[i]), arrowprops=dict(arrowstyle='<->'), \
         xytext=(x[i], x[i]*a+b),fontsize=18)
text(x[i], x[i]*a+b-sigma[i],'$e_i=f_i-d_i$',fontsize=18)
axis([0.2,1.0,-2.5,2.0])
legend()
savefig('Linreg_Beispiel.png',dpi=75) # Speichern als Bild
close()
```

```
Populating the interactive namespace from numpy and matplotlib
```

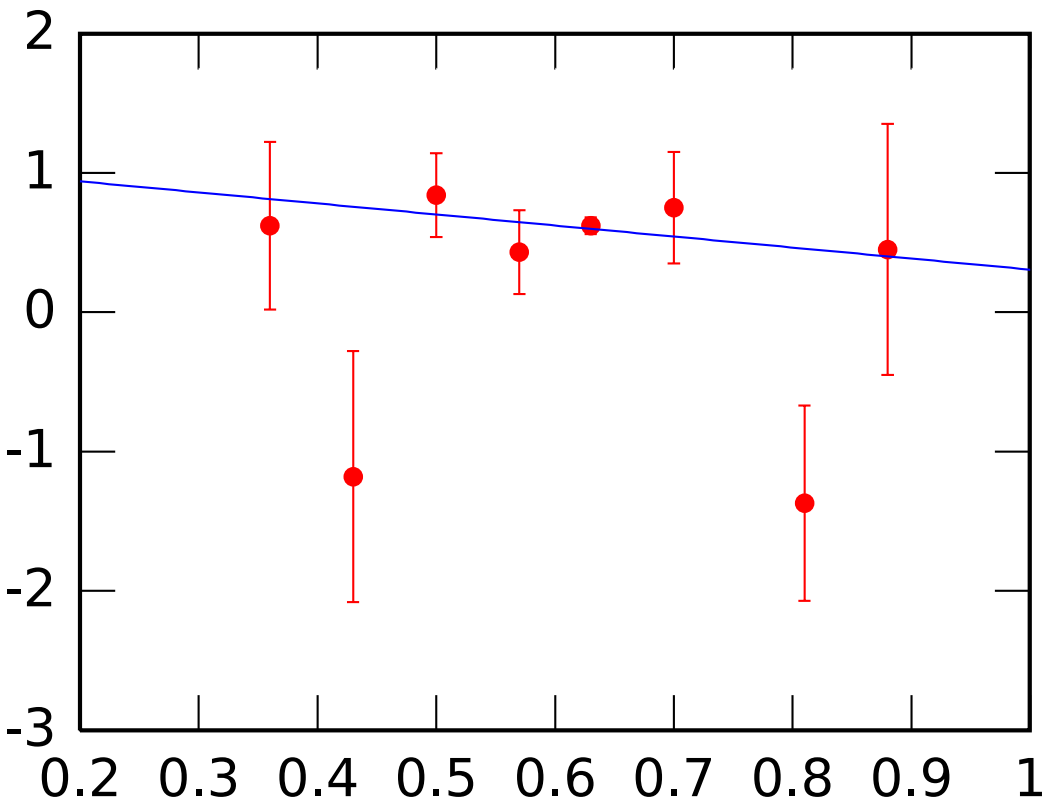
20.1 Lösungshilfe Python

Ergänzen Sie die fehlenden Berechnungsvorschriften ...

```
In [12]: %%file ausgleichsgerade.py
         from numpy import sum

         def ausgleichsgerade(x,d,s):
             """Berechne die Ausgleichsgerade f = a * x + b fuer Datenpunkte, die durch die Vektoren
                 x_i, d_i +- s_i gegeben sind.

                 Eingabe:
                     x X-Achsenabschnitt
                     d Y-Achsenabschnitt
                     s Fehler
                 Ausgabe:
                     a_fit, b_fit
             """
             alpha=sum(x**2/s**2)
             ...
             ...
             b_fit=...
             a_fit=...
             return a_fit, b_fit
```

20.2 Lösungshilfe Matlab/Octave

```
In [3]: x=[ 0.63,  0.81,  0.36,  0.43,  0.70, 0.57 ,  0.50,  0.88];
        d=[ 0.62, -1.37,  0.62, -1.18,  0.75, 0.43 ,  0.84,  0.45];
        sigma=[0.06,  0.7,  0.6,  0.9,  0.4, 0.3,  0.3 ,  0.9];

In [16]: %%file ausgleichsgerade.m

         function [a_fit, b_fit]=ausgleichsgerade(x,d,s)
             alpha=sum(x.^2./s.^2);
             ...
         end

Created file '/home/lars/sync/Zeitserien/Zeitreihenanalyse/stunde3/ausgleichsgerade.m'.

In [21]: [a,b]=ausgleichsgerade(x,d,sigma)

Out[21]: a = -0.79157
         b =  1.0966


In [28]: X=linspace(0.2,1);
         Y=a*X+b;

         errorbar(x,d,sigma,".r")
         hold()
         plot(X,a*X+b)
```



21 Beispiel Jahresgang und Anomalien

In diesem Beispiel verwenden wir meteorologische Daten, um einen mittleren Jahresgang und Anomalien zu berechnen. Die verwendeten Daten vom DWD sind hier verfügbar: http://icdc.zmaw.de/daten/atmosphere/dwd-station.html

Datenzitat: Freie Klimadaten von Messstationen für Deutschland, Deutscher Wetterdienst

21.1 Pandas

Im Folgenden wird das Pandas-Modul zum Einlesen und Prozessieren der Daten genutzt. Einführungen dazu gibt es z.B. hier:

- http://dx.doi.org/10.1007/978-1-4842-0958-5 Python Data Analytics
- http://proquest.safaribooksonline.com/9781449323592 Python for data analysis

Importieren des Moduls

```
In [2]: %pylab inline
        import pandas as pd
```

```
Populating the interactive namespace from numpy and matplotlib
```

21.2 Einlesen der Daten

```
In [3]: fn='dwdstation_hamburg_fuhlsbuettel_kl_10147.csv'
        DF=pd.read_csv(fn,encoding = "ISO-8859-1",usecols=[6,7,8,15],parse_dates=[[0,1,2]])
        DF.columns=['Date','T'] # Wir nutzen nur diese beiden Spalten
        DF.head() # Zeigt den "Kopf" der Tabelle
```

```
Out[3]:         Date     T
        0 1891-01-01 -10.7
        1 1891-01-02  -7.6
        2 1891-01-03 -10.3
        3 1891-01-04  -1.3
        4 1891-01-05  -4.1
```

21.3 Pandas Zeitserien Objekt

```
In [3]: dates=pd.to_datetime(DF['Date'].values) # Create a separate DatetimeIndex for selections
        T=pd.Series(DF['T'].values,index=dates) # Create a time series with date index
```

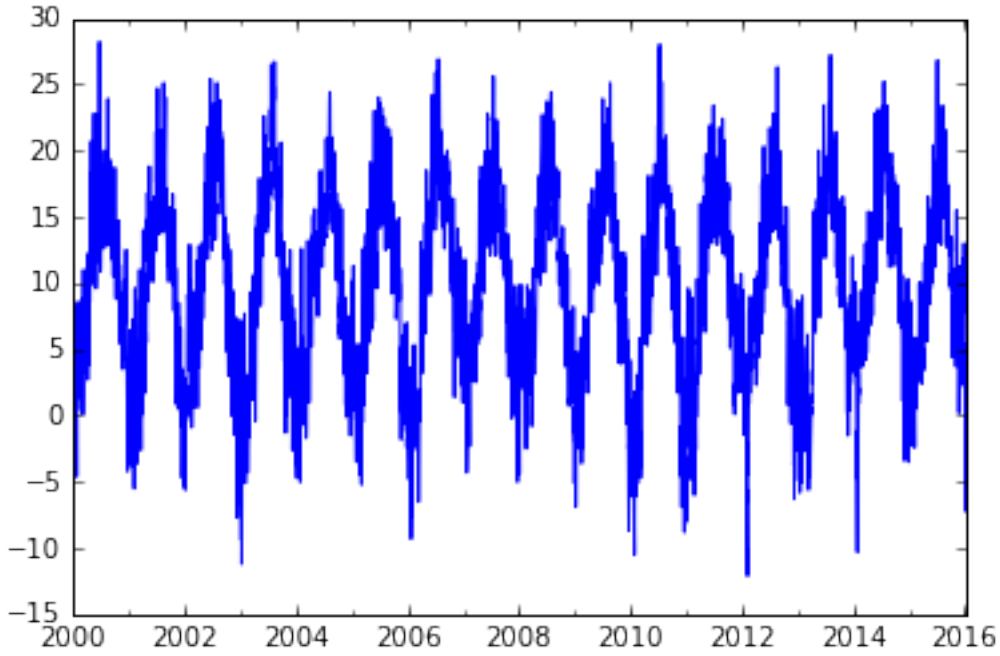
21.4 Display and indexing of time series

```
In [9]: T['2015-12-24':] # Show all entries since 24th december
```

```
Out[9]: 2015-12-24     8.5
        2015-12-25     7.8
        2015-12-26    13.0
        2015-12-27    10.2
        2015-12-28     3.7
        2015-12-29     5.4
        2015-12-30     4.5
        2015-12-31     2.1
        2016-01-01     3.5
        2016-01-02     0.1
        2016-01-03    -6.0
        2016-01-04    -7.2
        2016-01-05    -5.7
        2016-01-06    -5.4
        2016-01-07    -4.6
        2016-01-08     2.3
        2016-01-09     1.3
        2016-01-10     1.9
        2016-01-11     2.0
        2016-01-12     4.3
        dtype: float64
```

```
In [53]: T['2000-01-01':'2016-01-12'].plot()
```

```
Out[53]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb206f88d68>
```



21.5 Berechnung von wöchentlichen Mittelwerten

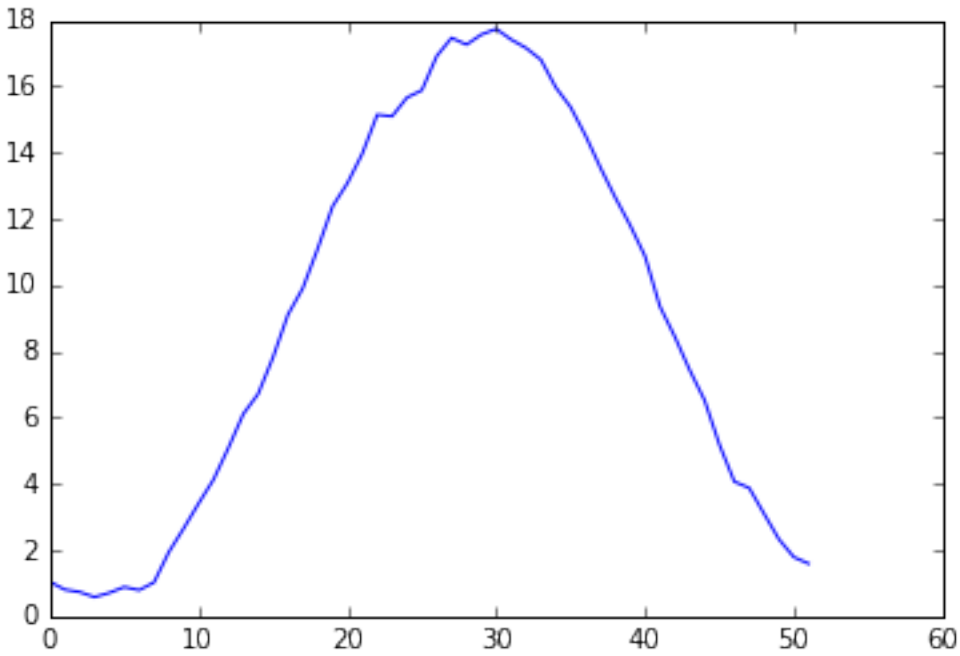
Pandas bietet vielfältige Methoden zur Auswahl von sich wiederholenden Perioden. In der folgenden Tabelle sind einige Codes zu finden.

Code	Offset	Description
D	Day	Calendar daily
H	Hour	Hourly
T or min	Minute	Minutely
S	Second	Secondly
L or ms	Milli	Millisecond
U	Micro	Microsecond
M	MonthEnd	Last calendar day of month
MS	MonthBegin	First calendar day of month
W-MON, W-TUE, ...	Week	Weekly on given day of week
A-JAN, A-FEB, ...	YearEnd	Annual dates anchored on last calendar day of given month
AS-JAN, AS-FEB, ...	YearBegin	Annual dates anchored on first day of given month

```
In [13]: week_index=dates.to_period(freq='W-THU') # Index für Wochenmittel, zentriert auf Donnerstag
         W=52
         weekly_mean=zeros(W)
         for i in range(1,W+1):
             weekly_mean[i-1]=T[week_index.weekofyear==i].mean()
```

```
In [15]: plot(weekly_mean)
         weekly_mean.tofile('T_Jahresgang_Hamburg.csv',sep=',')
         weekly_mean
```

```
Out[15]: array([  1.04303653,   0.80715909,   0.73588571,   0.58114286,
                  0.70971429,   0.88525714,   0.79131429,   1.0256    ,
                  1.94377143,   2.65931429,   3.41062857,   4.1288    ,
                  5.08548571,   6.10777143,   6.71497143,   7.82091429,
                  9.10502857,   9.90457143,  11.09748571,  12.3752    ,
                 13.07634286,  13.96822857,  15.1416    ,  15.09257143,
                 15.65805714,  15.87885714,  16.91154286,  17.46011429,
                 17.25485714,  17.56125714,  17.72937143,  17.4104    ,
                 17.15554286,  16.80125714,  15.96662857,  15.36765714,
                 14.51165714,  13.54308571,  12.64148571,  11.80365714,
                 10.86845714,   9.35348571,   8.44102857,   7.43131429,
                  6.52331429,   5.18114286,   4.06605714,   3.87554286,
                  3.09588571,   2.32502857,   1.78731429,   1.58994286])
```




```
In [11]: DF.head()
```

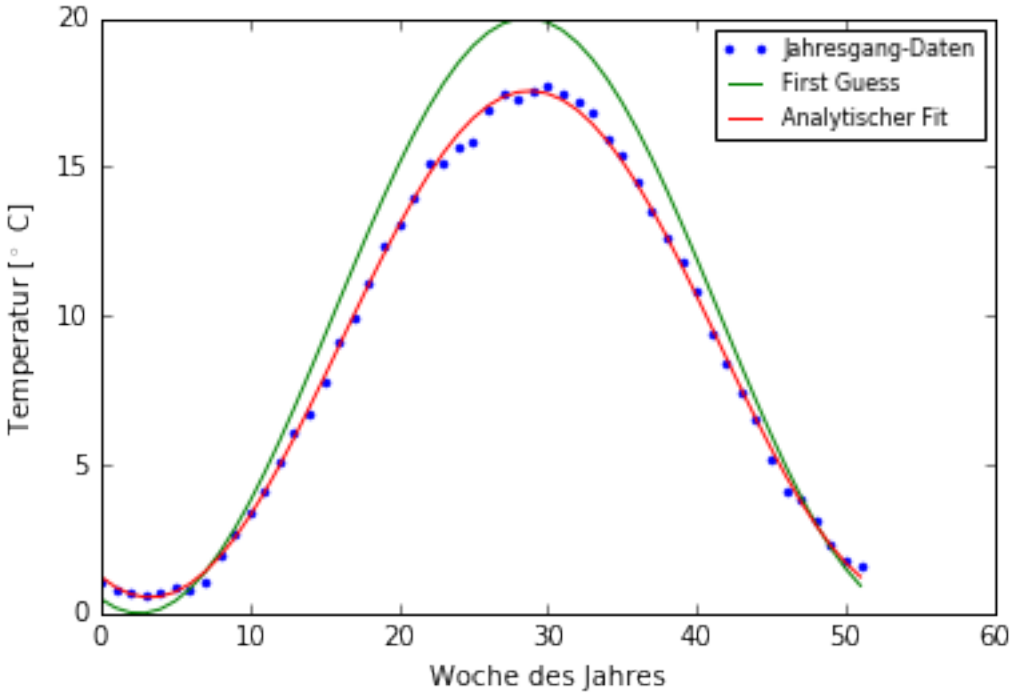
```
Out[11]:         Date     T  Wochen  Wochenmittelwert
         0 1891-01-01 -10.7       1               NaN
         1 1891-01-02  -7.6       2          1.043037
         2 1891-01-03 -10.3       2          0.807159
         3 1891-01-04  -1.3       2          0.735886
         4 1891-01-05  -4.1       2          0.581143
```

21.6 Jahresgang-Analyse

Im Folgenden versuchen wir eine analytische Funktion (Sinus) an den Jahresgang anzupassen. Dazu verwenden wir eine Optimierungsmethode.

In [27]:
```
from scipy.optimize import curve_fit

# Analytische Representation des Jahresgangs
def my_seasonal(x, freq, amplitude, phase, offset):
    return sin(x * freq + phase) * amplitude + offset

p0=[2*pi/52,10,-0.6*pi,10] # First guess parameter
first_guess=my_seasonal(x,*p0) # first guess
x=arange(52)

fit = curve_fit(my_seasonal, x, weekly_mean, p0=p0)
seasonal_fit = my_seasonal(x, *fit[0])

plot(weekly_mean, '.',label='Jahresgang-Daten')
plot(first_guess, label='First Guess')
plot(seasonal_fit, label='Analytischer Fit')
legend(fontsize=8)
xlabel('Woche des Jahres')
ylabel('Temperatur [$^\circ$ C]')
```

Out[27]: <matplotlib.text.Text at 0x7fb206f430b8>



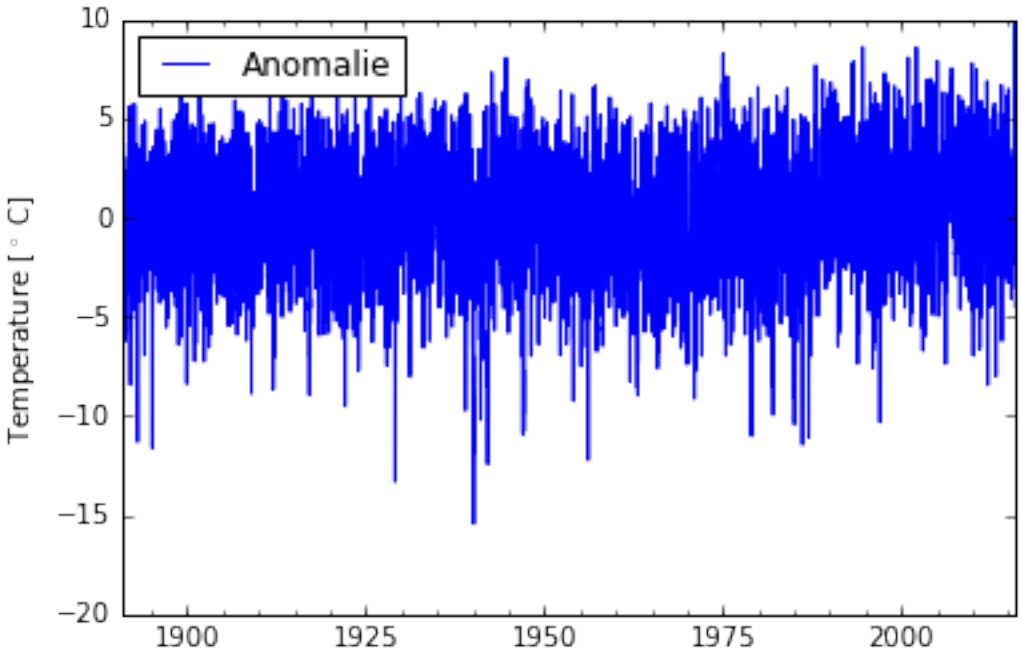
21.7 Berechnung der Anomalien

Die Anomalie ist die Abweichung zum klimatologischen Mittelwert (Jahresgang). Zur weiteren Bearbeitung/Analyse speichern wir die Daten in einer CSV-Datei.

In [35]:
```
# Analytische Funktion der Klimatologie
T_climfit=my_seasonal(T.index.weekofyear,*fit[0])
```

```
T_climfit_dates=T.index
T_climfit_series=pd.Series(T_climfit,index=T_climfit_dates)

# Berechnung der Anomalien
T_weekly_resampled=T.resample('W-THU',how='mean')
T_anom=(T_weekly_resampled-T_climfit_series).dropna()
T_anom.plot(label='Anomalie')
legend(loc=2)
ylabel('Temperature [$^\circ$ C]')
```

Out[35]: <matplotlib.text.Text at 0x7fb207038940>



In [46]:
```
TABELLE=pd.DataFrame()
TABELLE['Wochenmittel']=T_weekly_resampled
TABELLE['Jahresgang']=T_climfit_series
TABELLE['Anomalie']=T_anom
TABELLE.to_csv('DWD_Hamburg_Wochenmittel_Jahresgang_Anomalien.csv')
```

In [1]:
```
!head DWD_Hamburg_Wochenmittel_Jahresgang_Anomalien.csv
```

```
,Wochenmittel,Jahresgang,Anomalie
1891-01-01,-10.7,0.881145149061247,-11.581145149061246
1891-01-08,-6.485714285714285,0.6484336986350101,-7.134147984349295
1891-01-15,-4.957142857142857,0.5431151330989668,-5.500257990241824
1891-01-22,-6.042857142857143,0.5667872246401657,-6.609644367497308
1891-01-29,1.7857142857142858,0.7190908475025619,1.066623438211724
1891-02-05,2.257142857142857,0.9977154262300836,1.2594274309127735
1891-02-12,-0.3857142857142858,1.3984339890608757,-1.7841482747751614
1891-02-19,0.5857142857142856,1.9151672946829565,-1.329453008968671
1891-02-26,0.5,2.540076059493823,-2.040076059493823
```

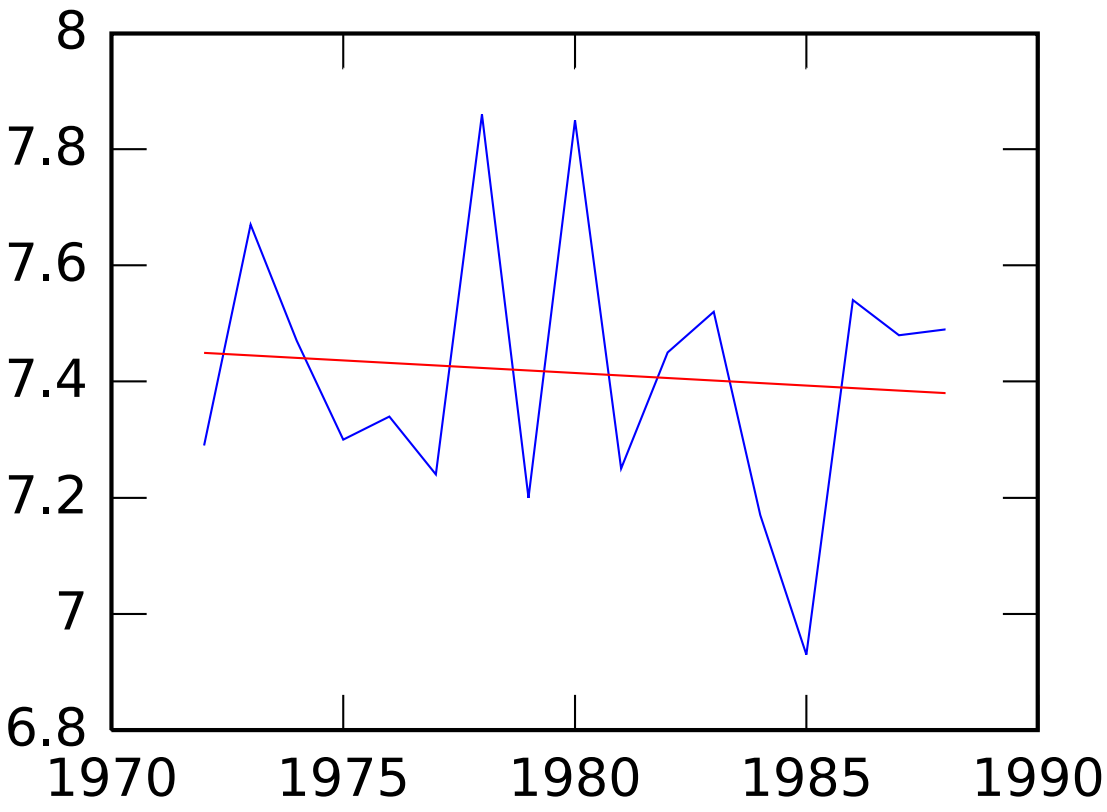
In [ ]:

22 Signifkanz von Trends

Sind die Trends signifikant?

22.1 Beispiel Meereis Ausdehnung

```
In [78]: N=17; %nehme nur die ersten 17 Jahre des Datensatzes
         Y=transpose(textread("september_extent_1972_2015.txt"));
         X=1972:1:2015;
         Y=Y(1:N);
         X=X(1:N);

         plot(X,Y);
         hold;
         p = polyfit(X,Y,1);
         y_line=p(1)*X+p(2);
         plot(X,y_line,"r")
```



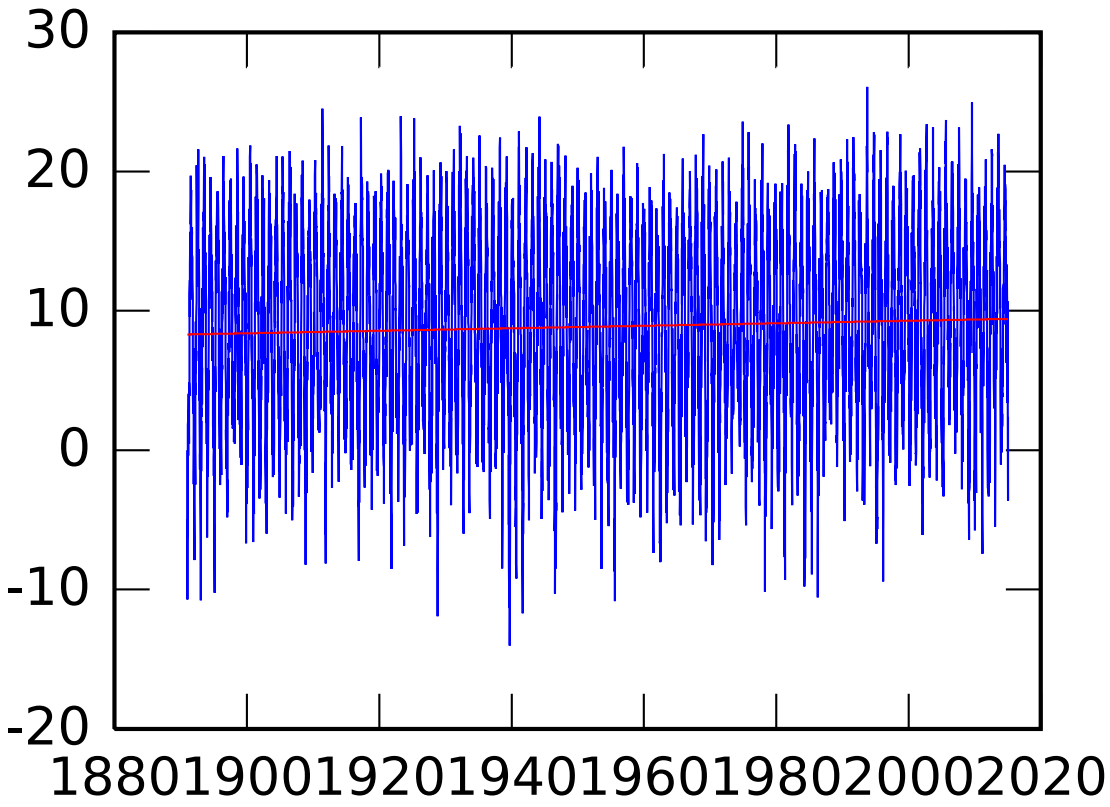
22.2 Beispiel Lufttemperatur in Hamburg

```
In [59]: !head DWD_Hamburg_Wochenmittel_Jahresgang_Anomalien.csv
```

```
,Wochenmittel,Jahresgang,Anomalie
1891-01-01,-10.7,0.881145149061247,-11.581145149061246
1891-01-08,-6.485714285714285,0.6484336986350101,-7.134147984349295
```

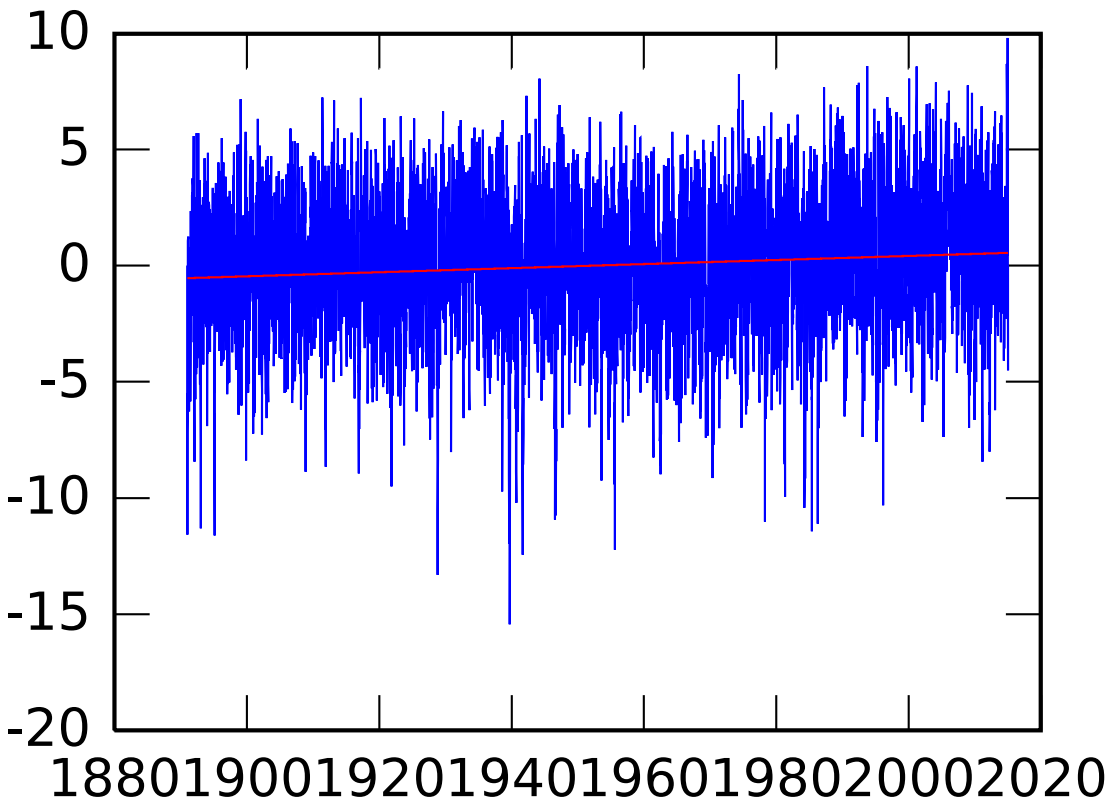
```
1891-01-15,-4.957142857142857,0.5431151330989668,-5.500257990241824
1891-01-22,-6.042857142857143,0.5667872246401657,-6.609644367497308
1891-01-29,1.7857142857142858,0.7190908475025619,1.066623438211724
1891-02-05,2.257142857142857,0.9977154262300836,1.2594274309127735
1891-02-12,-0.3857142857142858,1.3984339890608757,-1.7841482747751614
1891-02-19,0.5857142857142856,1.9151672946829565,-1.329453008968671
1891-02-26,0.5,2.540076059493823,-2.040076059493823
```

```
In [75]: TAB=csvread("DWD_Hamburg_Wochenmittel_Jahresgang_Anomalien.csv");
         T=transpose(TAB(:,2));
         t=linspace(1891,2015,6526);
         plot(t,T);
         hold;
         p = polyfit(t,T,1);
         y_line=polyval(p,t);
         plot(t,y_line,"r")
```



22.3 Anomalie der Lufttemperatur

```
In [76]: A=transpose(TAB(:,4));
         plot(t,A);
         hold;
         p = polyfit(t,A,1);
         y_line=polyval(p,t);
         plot(t,y_line,"r")
```



23 Fehlerfortpflanzung

Es sei $z = f(u, v, ..)$ eine Größe, die aus Zufallsvariablen abgeleitet ist. Für jede Realisierung $(u_i, v_i, ..)$ kann man z_i ausrechnen

$$z_i = f(u_i, v_i, ..)$$

Hat man viele Realisierungen zur Verfügung, so kann man die Varianz von z abschätzen

$$\sigma_z^2 = \frac{1}{N-1} \sum_{i=1}^{N} (z_i - \bar{z})^2$$

Näherungsweise gilt

$$z_i - \bar{z} \approx (u_i - \bar{u}) \frac{\partial f}{\partial u} + (v_i - \bar{v}) \frac{\partial f}{\partial v} + ...$$

damit erhält man

$$\sigma_z^2 \approx \frac{1}{N-1} \sum_{i=1}^{N} \left[(u_i - \bar{u})^2 (\frac{\partial f}{\partial u})^2 + (v_i - \bar{v})^2 (\frac{\partial f}{\partial v})^2 + (u_i - \bar{u})(v_i - \bar{v})(\frac{\partial f}{\partial u})(\frac{\partial f}{\partial v}) + ... \right]$$

oder

$$\sigma_z^2 \approx (\frac{\partial f}{\partial u})^2 \sigma_u^2 + (\frac{\partial f}{\partial v})^2 \sigma_v^2 + 2 \frac{\partial f}{\partial u} \frac{\partial f}{\partial v} \sigma_{u,v}^2 + \cdots$$

Für unabhängige (unkorrelierte) u, v ist dies das allgemein bekannte Fehlerfortpflanzungsgesetz. Für korrelierte Daten ($\sigma^2_{u,v} \neq 0$) sind die gemischten Terme zu beachten.

24 Kovarianzmatrix

Gegeben ist ein Vektor von Zufallsvariablen (Meßgrößen)

$$\vec{d} = [d_1, .., d_N]^T$$

z.B. d_1 = Temperatur(Zeit) an Position 1 (Zeitreihe 1) und d_2 = Temperatur(Zeit) an Position 2 (Zeitreihe 2) dann ist die Kovarianzmatrix $\mathrm{cov}(\vec{d})$ definiert als

$$\mathrm{cov}(\vec{d}) = (\mathrm{cov}(d_i, d_j))_{ij=1,..,N} = \begin{pmatrix} \mathrm{cov}(d_1, d_1) & \cdots & \mathrm{cov}(d_1, d_N) \\ \vdots & \ddots & \vdots \\ \mathrm{cov}(d_N, d_1) & \cdots & \mathrm{cov}(d_N, d_N) \end{pmatrix}$$

Beachte $\mathrm{cov}(\vec{d})$ ist quadratisch und weil $\mathrm{cov}(d_i, d_j) = \sigma^2_{d_i d_j} = \sigma^2_{d_j d_i}$ auch symmetrisch.

- Auf der Diagonalen stehen die Varianzen der einzelnen Meßgrößen, auf den Nebendiagonalen stehen die Kovarianzen
- Sind die einzelnen Meßgrößen unabhängig, so ist die Kovarianzmatrix diagonal

25 Fehlerfortpflanzung mit korrelierten Zufallsvariablen

Es sei nun $y = g(u, v, ..)$ eine zweite Größe, die aus Zufallsvariablen abgeleitet ist. Dann folgt wie oben

$$\sigma^2_y \approx (\frac{\partial g}{\partial u})^2 \sigma^2_u + (\frac{\partial g}{\partial v})^2 \sigma^2_v + 2\frac{\partial g}{\partial u}\frac{\partial g}{\partial v}\sigma^2_{u,v} + \cdots$$

Die Kovarianz $\sigma^2_{y,z} = cov(y, z)$ errechnet sich aus

$$\sigma^2_{y,z} = \frac{1}{N-1}\sum_{i=1}^{N}(y_i - \bar{y})(z_i - \bar{z}).$$

mit

$$y_i - \bar{y} \approx (u_i - \bar{u})\frac{\partial g}{\partial u} + (v_i - \bar{v})\frac{\partial g}{\partial v} + ...$$

$$z_i - \bar{z} \approx (u_i - \bar{u})\frac{\partial f}{\partial u} + (v_i - \bar{v})\frac{\partial f}{\partial v} + ...$$

folgt

$$cov(y, z) = \sigma^2_{y,z} = (\frac{\partial g}{\partial u}\frac{\partial f}{\partial u})\sigma^2_u + (\frac{\partial g}{\partial v}\frac{\partial f}{\partial v})\sigma^2_v + (\frac{\partial f}{\partial u}\frac{\partial g}{\partial v} + \frac{\partial f}{\partial v}\frac{\partial g}{\partial u})\sigma^2_{u,v} + \cdots$$

26 Anwendung auf lineare Gleichungssysteme

Sei $\mathbf{x}$ gegeben als lineare Transformation

$$\mathbf{x} = \mathbf{G}\mathbf{d}$$

dann gilt für die Kovarianzmatrix $cov(\mathbf{x})$

$$\mathrm{cov(x)} = \mathbf{G}\mathrm{cov}(\mathbf{d})\mathbf{G}^{\mathrm{T}}$$

.

26.1 Beweis für 2x2 Spezialfall

Es sei

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}$$

entsprechend

$$x = g_{11}u + g_{12}v = f(u,v)$$
$$y = g_{21}u + g_{22}v = g(u,v)$$

daraus ergibt sich

$$\sigma_x^2 = (\frac{\partial f}{\partial u})^2\sigma_u^2 + (\frac{\partial f}{\partial v})^2\sigma_v^2 + 2\frac{\partial f}{\partial u}\frac{\partial f}{\partial v}\sigma_{u,v}^2$$

$$= g_{11}\sigma_u^2 + g_{12}^2\sigma_v^2 + 2g_{11}g_{12}\sigma_{u,v}^2$$

und

$$\sigma_{x,y}^2 = g_{11}g_{21}\sigma_u^2 + g_{12}g_{22}\sigma_v^2 + (g_{11}g_{22} + g_{12}g_{21})\sigma_{u,v}^2$$

Es gilt

$$\mathrm{cov}(\mathbf{x}) = \mathbf{G}\mathrm{cov}(\mathbf{d})\mathbf{G}^{\mathrm{T}} = \begin{pmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{pmatrix} \begin{pmatrix} \sigma_u^2 & \sigma_{u,v} \\ \sigma_{u,v} & \sigma_v^2 \end{pmatrix} \begin{pmatrix} g_{11} & g_{21} \\ g_{12} & g_{22} \end{pmatrix}$$

$$\mathrm{cov}(\mathbf{x}) = \begin{pmatrix} g_{11}\sigma_u^2 + g_{12}\sigma_{u,v}^2 & g_{11}\sigma_{u,v}^2 + g_{12}\sigma_v^2 \\ g_{21}\sigma_u^2 + g_{22}\sigma_{u,v}^2 & g_{21}\sigma_{u,v}^2 + g_{22}\sigma_v^2 \end{pmatrix} \begin{pmatrix} g_{11} & g_{21} \\ g_{12} & g_{22} \end{pmatrix}$$

$$\vdots$$

27 Einschub: symbolische Algebra mit Sympy

Die Berechnung ist etwas länglich. Zeit für einen Ausflug in die Möglichkeiten der Computeralgebra mit dem Modul sympy.

```
In [8]: import sympy as sp
        sp.init_printing() # Fuer LaTex-Ausgabe im Notebook
        g11,g12,g21,g22=sp.symbols('g_{11} g_{12} g_{21} g_{22}')
        g=sp.Matrix(((g11,g12),(g21,g22)))
        su,sv,suv=sp.symbols('\sigma_{u} \sigma_{v} \sigma_{uv}')
        s=sp.Matrix(((su**2,suv**2),(suv**2,sv**2)))
```

```
In [2]: g
```

Out[2]:

$$\begin{bmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{bmatrix}$$

```
In [3]: s
```

Out[3]:

$$\begin{bmatrix} \sigma_u^2 & \sigma_{uv}^2 \\ \sigma_{uv}^2 & \sigma_v^2 \end{bmatrix}$$

In [11]: g*s

Out[11]:

$$\begin{bmatrix} \sigma_{uv}^2 g_{12} + \sigma_u^2 g_{11} & \sigma_{uv}^2 g_{11} + \sigma_v^2 g_{12} \\ \sigma_{uv}^2 g_{22} + \sigma_u^2 g_{21} & \sigma_{uv}^2 g_{21} + \sigma_v^2 g_{22} \end{bmatrix}$$

In [5]: g*s*g.T

Out[5]:

$$\begin{bmatrix} g_{11}\left(\sigma_{uv}^2 g_{12} + \sigma_u^2 g_{11}\right) + g_{12}\left(\sigma_{uv}^2 g_{11} + \sigma_v^2 g_{12}\right) & g_{21}\left(\sigma_{uv}^2 g_{12} + \sigma_u^2 g_{11}\right) + g_{22}\left(\sigma_{uv}^2 g_{11} + \sigma_v^2 g_{12}\right) \\ g_{11}\left(\sigma_{uv}^2 g_{22} + \sigma_u^2 g_{21}\right) + g_{12}\left(\sigma_{uv}^2 g_{21} + \sigma_v^2 g_{22}\right) & g_{21}\left(\sigma_{uv}^2 g_{22} + \sigma_u^2 g_{21}\right) + g_{22}\left(\sigma_{uv}^2 g_{21} + \sigma_v^2 g_{22}\right) \end{bmatrix}$$

In [12]: sp.expand((g*s*g.T)[0,0])

Out[12]:

$$2\sigma_{uv}^2 g_{11} g_{12} + \sigma_u^2 g_{11}^2 + \sigma_v^2 g_{12}^2$$

Identisch mit σ_x^2 (vergleiche mit oben)

In [7]: sp.expand((g*s*g.T)[0,1])

Out[7]:

$$\sigma_{uv}^2 g_{11} g_{22} + \sigma_{uv}^2 g_{12} g_{21} + \sigma_u^2 g_{11} g_{21} + \sigma_v^2 g_{12} g_{22}$$

Identisch mit $\sigma_{x,y}^2$. Damit haben wir gezeigt, dass die Gleichungen für den Spezialfall 2x2 äquivalent sind.

28 Fehler der Modellgeraden

Gesucht sei die Ausgleichsgerade $y = ax + b$ und deren Fehler σ_y des linearen Regressionsproblems für $N > 2$ Datenpunkte (x_i, d_i).

Der Lösungsvektor $\mathbf{x} = [a, b]^T$ ergibt sich aus der Matrixformulierung des linearen Regressionsproblems

$$ax_1 + b = d_1 ax_2 + b = d_2 ..ax_N + b = d_N$$

$$\underbrace{\mathbf{A}}_{N\times 2}\underbrace{\mathbf{x}}_{2\times 1} = \underbrace{\mathbf{d}}_{N\times 1}$$

Im Allgemeinen ist es unmöglich das Gleichungssystem $\mathbf{A} \cdot \mathbf{x} = \mathbf{d}$ exakt zu lösen. Wir suchen die Lösung im Sinne der kleinsten quadratischen Abweichung

$$\mathbf{F} = \mathbf{e}^T \cdot \mathbf{e} = (\mathbf{A} \cdot \mathbf{x} - \mathbf{d})^T (\mathbf{A} \cdot \mathbf{x} - \mathbf{d})$$

.

als

$$\mathbf{x} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{d}$$

.

Die Kovarianzmatrix errechnet sich aus $\mathrm{cov}(\mathbf{x}) = [(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T]\mathrm{cov}(\mathbf{d})[(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T]^T$ für unkorrelierte Daten mit Fehler σ_d^2 und Kovarianz $\mathrm{cov}(\mathbf{d}) = \sigma_d \mathbf{1}$ zu

$$\mathrm{cov}(\mathbf{x}) = \sigma_d^2 (\mathbf{A}^T \mathbf{A})^{-1}$$

.

Fassen wir y als lineare Abbildung $\mathbf{G} = [x, 1]$ von $\mathbf{u} = [a, b]^T$ auf

$$\underbrace{y}_{1\times1} = \underbrace{\mathbf{G}}_{2\times1}\underbrace{\mathbf{u}}_{1\times2}$$

dann ist die Kovarianz gegeben durch

$$\mathrm{cov(y)} = \mathbf{G}\mathrm{cov}(\mathbf{u})\mathbf{G}^{\mathrm{T}}$$

.

Mit $\mathrm{cov}(\mathbf{u}) = \begin{pmatrix} \sigma_a^2 & \sigma_{ab}^2 \\ \sigma_{ab}^2 & \sigma_b^2 \end{pmatrix}$ ergibt sich der gesuchte Fehler

$$\sigma_y^2 = \sigma_a^2 x^2 + 2\sigma_{ab} x + \sigma_b^2$$

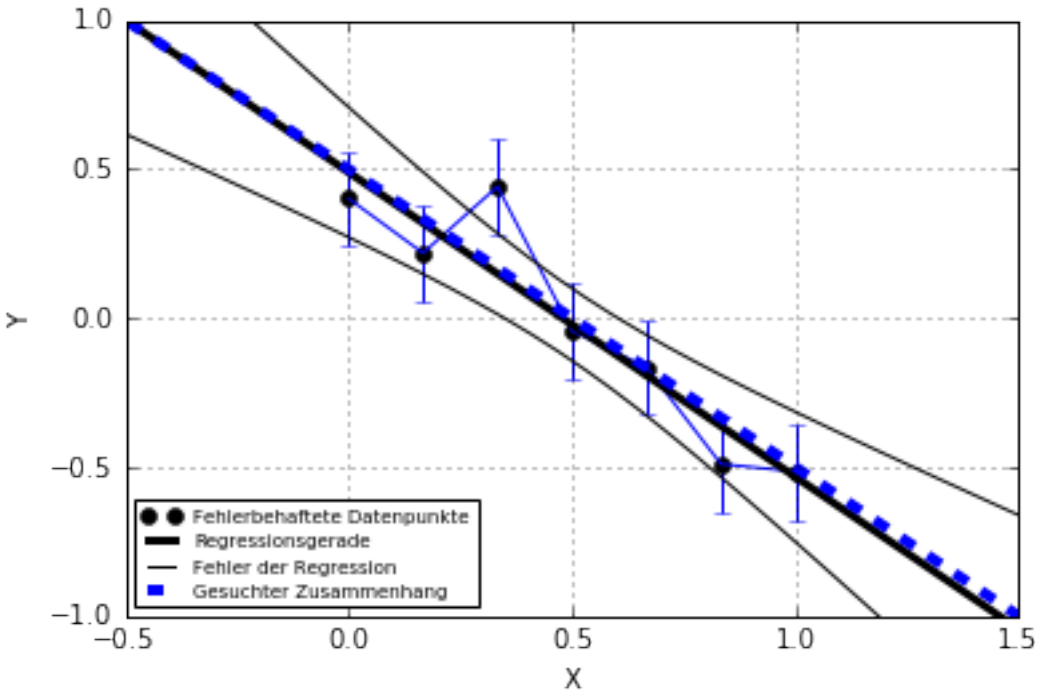
Der Fehler ist minimal im Schwerpunkt $x = \bar{x}$ der Daten. Für sehr große und sehr kleine x ist der Fehler proportional zum Fehler der Steigung σ_a.

28.1 Python-Beispiel Lineare-Regression

```
In [1]: %pylab inline
        # Erzeuge Zufallswerte
        N=7
        a_wahr=-1.0
        b_wahr=0.5

        s=0.4#  Messfehler

        t=linspace(0,1,N)
        data=a_wahr*t+b_wahr+randn(N)*s**2
        data=array([ 0.40152768,0.2184287,0.44117141,-0.04235238,-0.16512194,-0.4892583, -0.51473881])

        print(data)
        data_wahr=a_wahr*t+b_wahr

        # Lineare Regression

        A=array((t/s,ones(N)/s)).T
        d=array(data/s).T

        x=dot(dot(inv(dot(A.T,A)),A.T),d)  # Least-Squares Loesung

        # Fehler der Loesung (Daten unkorreliert)
        # Kovarianzmatrix
        sigma_x = s**2 * inv(dot(A.T,A))
        sigma_a2=sigma_x[0,0]
        sigma_ab2=sigma_x[0,1]
        sigma_b2=sigma_x[1,1]

        a,b=float(x[0]),float(x[1])
        X=linspace(-1,2)
        Y=a*X+b

        print('\nWahre vorgegebene Werte:')
        print('a=%3.2f'%a_wahr)
```

```
print('b=%3.2f'%b_wahr)

print('\nDurch Regression von Y=a*X+b bestimmte Werte:')
print('a=%3.2f'%a,' +-%3.2f'%sqrt(sigma_a2))
print('b=%3.2f'%b,' +-%3.2f'%sqrt(sigma_b2))

# 1-sigma Konfidenzintervalle fuer Y
sigma_Y=sqrt(sigma_a2*X**2+2*sigma_ab2*X+sigma_b2)

figure(1)
plot(t,data,'ko',label='Fehlerbehaftete Datenpunkte')
errorbar(t,data,yerr=s**2)
plot(X,Y,'k-',label='Regressionsgerade',lw=3)
plot(X,Y-sigma_Y*2,'k-',label='Fehler der Regression')
plot(X,Y+sigma_Y*2,'k-')
plot(X,X*a_wahr+b_wahr,'b--',linewidth=4,label='Gesuchter Zusammenhang')
xlabel('X')
ylabel('Y')
legend(loc=3,fontsize=7)
axis([-0.5,1.5,-1,1])
grid()
```

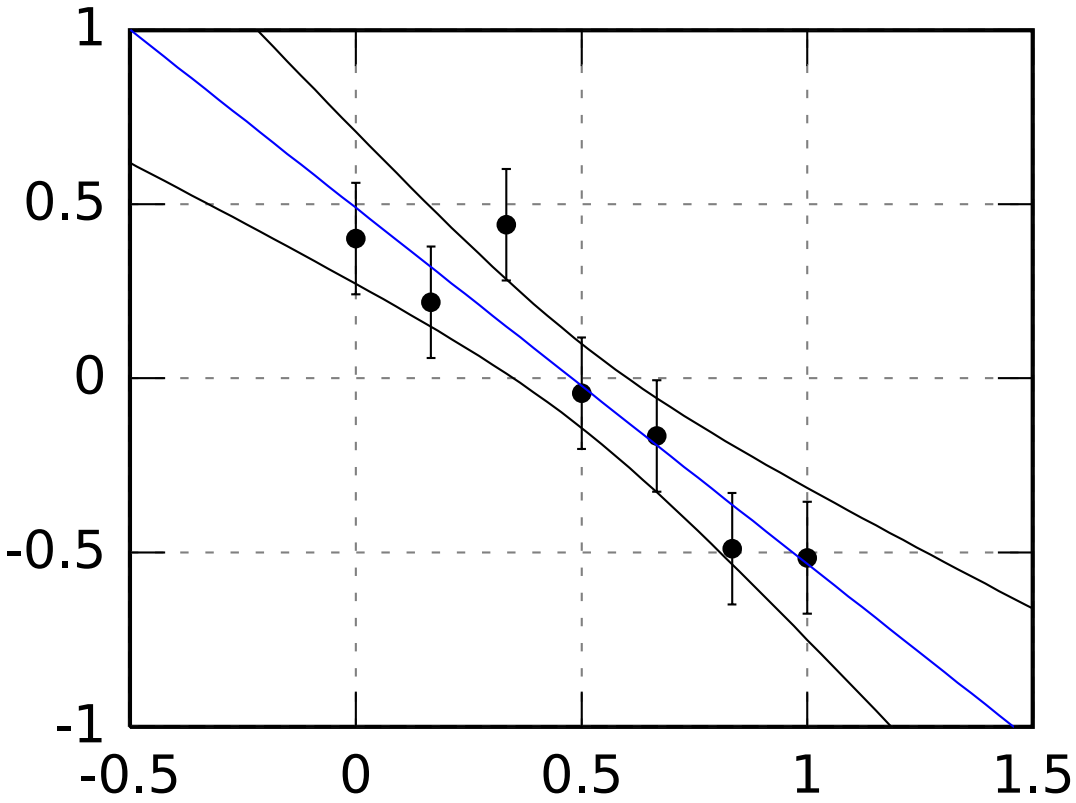
```
Populating the interactive namespace from numpy and matplotlib
[ 0.40152768  0.2184287   0.44117141 -0.04235238 -0.16512194 -0.4892583
 -0.51473881]

Wahre vorgegebene Werte:
a=-1.00
b=0.50

Durch Regression von Y=a*X+b bestimmte Werte:
a=-1.02  +-0.18
b=0.49  +-0.11
```



28.2 Octave-Beispiel

In [15]:
```
% Erzeuge Zufallswerte
N=7;
a_wahr=-1.0;
b_wahr=0.5;
s=0.4;%  Messfehler
t=linspace(0,1,N);
data=a_wahr.*t+b_wahr+randn(1,N)*s.^2;
% Gleiche Werte wie Python-Beispiel oben
data=[ 0.40152768  0.2184287   0.44117141 -0.04235238 -0.16512194 -0.4892583 -0.51473881]
data_wahr=a_wahr.*t+b_wahr;

A=[transpose(t)./s,ones(N,1)./s];
d=[transpose(data./s)];
x=inv(transpose(A)*A)*transpose(A)*d ;
sigma_x = s.^2 .* inv(transpose(A)*A);
sigma_a2=sigma_x(1,1);
sigma_ab2=sigma_x(1,2);
sigma_b2=sigma_x(2,2);
a=x(1);
b=x(2);

disp('Durch Regression von Y=a*X+b bestimmte Werte:')
disp(['a=',num2str(a,'%3.2f'),' +-',num2str(sqrt(sigma_a2),'%3.2f')])
disp(['b=',num2str(b,'%3.2f'),' +-',num2str(sqrt(sigma_b2),'%3.2f')])

X=linspace(-1,2);
```

```
Y=a*X+b;
sigma_Y=sqrt(sigma_a2.*X.^2+2.*sigma_ab2.*X+sigma_b2);

errorbar(t,data,s.^2,'k.');
hold;
plot(X,Y);
plot(X,Y-sigma_Y*2,'k-');
plot(X,Y+sigma_Y*2,'k-');
axis([-0.5,1.5,-1,1]);
grid();
```



Out[15]: data =

```
   0.401528   0.218429   0.441171  -0.042352  -0.165122  -0.489258  -0.514739
```

```
Durch Regression von Y=a*X+b bestimmte Werte:
a=-1.02 +-0.18
b=0.49 +-0.11
```

In [12]: sigma_x

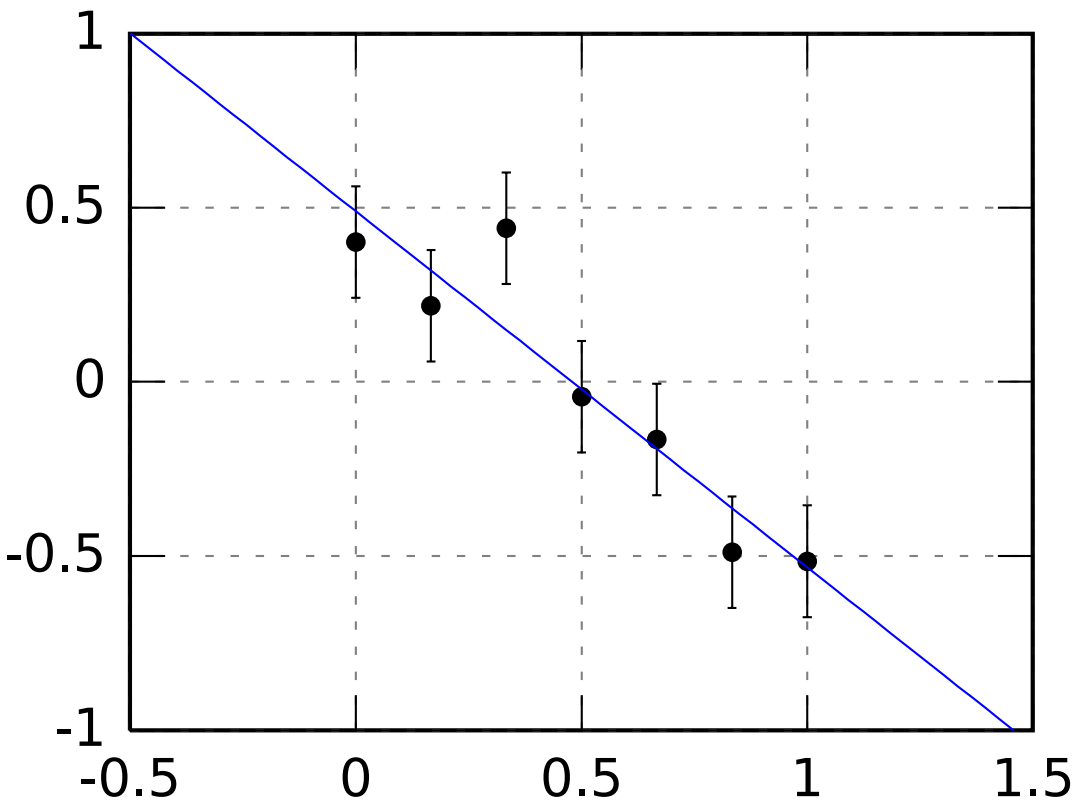
Out[12]: sigma_x =

```
   0.032914  -0.016457
```

```
   -0.016457   0.011886
```

28.3 Octave Regress-Funktion

In [17]:
```
% Erzeuge Zufallswerte
N=7;
a_wahr=-1.0;
b_wahr=0.5;
s=0.4;%  Messfehler
t=linspace(0,1,N);
data=a_wahr.*t+b_wahr+randn(1,N)*s.^2;
data=[ 0.40152768  0.2184287   0.44117141 -0.04235238 -0.16512194 -0.4892583 -0.51473881]

data_wahr=a_wahr.*t+b_wahr;

X=[ones(N,1),transpose(t)];
Y=transpose(data);
ALPHA=0.31; % 1 sigma Konfidenz, Default ist 0.05= 2 Sigma
[B, BINT, R, RINT, STATS] = regress (Y, X, ALPHA);
a=B(2);
b=B(1);
x=linspace(-1,2);
y=a*x+b;

sigma_a=B(2)-BINT(2,1);
sigma_b=B(1)-BINT(1,1);

disp('Durch Regression von Y=a*X+b bestimmte Werte:')
disp(['a=',num2str(a,'%3.2f'),' +-',num2str(sigma_a,'%3.2f')])
disp(['b=',num2str(b,'%3.2f'),' +-',num2str(sigma_b,'%3.2f')])

R2=STATS(1);
disp(['R^2=',num2str(R2,'%3.2f')])

p=STATS(3);
disp(['p=',num2str(p,'%3.5f')])

errorbar(t,data,s.^2,'k.');
hold();
plot(x,y);

axis([-0.5,1.5,-1,1]);
grid();
```




```
Out[17]: data =

            0.401528   0.218429   0.441171  -0.042352  -0.165122  -0.489258  -0.514739

         Durch Regression von Y=a*X+b bestimmte Werte:
         a=-1.02 +-0.20
         b=0.49 +-0.12
         R^2=0.87
         p=0.00215
```

```
In [6]: regress?
```

```
In [14]: corrcoef(t,data)**2
```

```
Out[14]: ans =  0.87045
```

```
In [ ]:
```

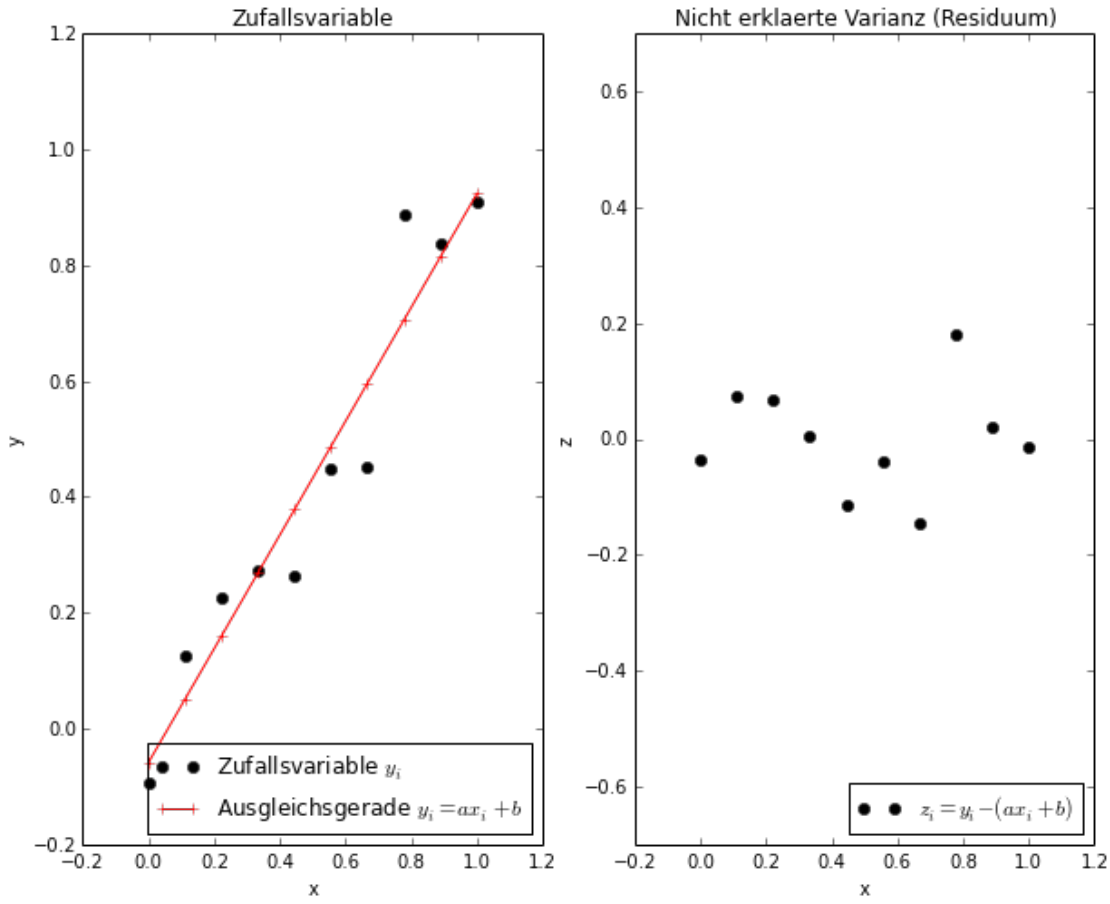
29 Erklärte Varianz

Der quadrierte Korrelationskoeffizient R^2 hat eine besondere Bedeutung. Er gibt an, welchen Anteil der Variabilität einer Zufallsvariablen durch ein lineares Modell erklärt wird. Das folgende Programm verdeutlicht den Zusammenhang.

http://de.wikipedia.org/wiki/Bestimmtheitsmaß

In [5]:
```
# Importieren von Modul für lineare Regression
import scipy.stats as stats

%pylab inline
# Ausgabe in Ipython Notebook

N=10 # Anzahl der Datenpunkte i=0,1,..,N

x=linspace(0,1,N)
y=1*x+randn(N)*0.1 # Generiere Zufallsvariable

# Lineare Regression
a, b, r_value, p_value, std_err = stats.linregress(x,y)

figure(1, figsize=(10,8))
subplot(1,2,1) # Zwei Plots in einer Abbildung

# Abbildung 1 - Ursprüngliche Varianz
plot(x,y,'ko',label='Zufallsvariable $y_i$')
plot(x,a*x+b, 'r+-',label='Ausgleichsgerade $y_i=a x_i+b$')
s=0.2 # Rand
axis([-s,1+s,-s,1+s])
title('Zufallsvariable')
xlabel('x')
ylabel('y')
legend(loc=4)

z=y-a*x-b # Subtraktion der Ausgleichgeraden

# Abbildung 2 - Erklärte Varianz
subplot(1,2,2)

plot(x,z,'ko',label='$z_i=y_i-(ax_i+b)$')
axis([-s,1+s,-s-0.5,0.5+s])
title('Nicht erklaerte Varianz (Residuum)')
xlabel('x')
ylabel('z')
legend(loc=4)

print "Quadrierter Korrelationskoeffizient R^2:", r_value**2
print '"Erklaerte Varianz" in Prozent', (1-var(z)/var(y))*100
```

```
Populating the interactive namespace from numpy and matplotlib
Quadrierter Korrelationskoeffizient R^2: 0.92463522237
"Erklaerte Varianz" in Prozent 92.463522237
```



30 Wichtige Verteilungsfunktionen

30.1 Normalverteilung (z-Verteilung)

Die Gaußverteilung ist gegeben durch

$$p(x) = \frac{1}{\sigma_x \sqrt{2\pi}} e^{-\frac{(x-\mu_x)^2}{2\sigma_x^2}}$$

Es gilt

$$E[x] = \mu_x$$

und

$$E[(x-\mu_x)^2] = \sigma_x^2$$

Substitution von $z = \frac{x-\mu_x}{\sigma_x}$ liefert die standardisierte Normalverteilung

$$p(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}}$$

für die gilt $E[x] = \mu_z = 0$ und $E[(x-\mu_z)^2] = \sigma_z^2 = 1$
Die Wahrscheinlichkeit P berechnet sich aus dem Integral

$$P(z_\alpha) = \int_{-\infty}^{z_\alpha} p(z)dz = Prob[z < z_\alpha] = 1 - \alpha$$

bzw.

$$1 - P(z_\alpha) = \int_{z_\alpha}^{\infty} p(z)dz = Prob[z > z_\alpha] = \alpha$$

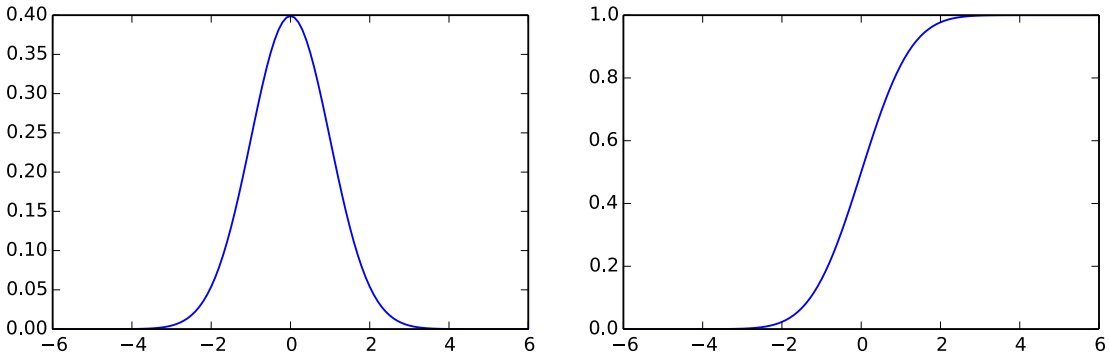
30.2 Normalverteilung: scipy.stats.norm(x)

```
In [3]: %pylab inline
        %config InlineBackend.figure_format = 'svg'
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
In [4]: import scipy.stats as stats

        x=linspace(-6,6,121)
        figure(figsize=(10,3))
        subplot(1,2,1)
        plot(x,stats.norm.pdf(x))
        subplot(1,2,2)
        plot(x,stats.norm.cdf(x))
```

```
Out[4]: [<matplotlib.lines.Line2D at 0x7fa546ba5f90>]
```




```
In [24]: #Wieviele Werte liegen innerhalb +-1, +-2 und +-3 Standardabweichungen?
         #beidseitig: *2

         z_cdf=stats.norm.cdf

         print (1-z_cdf(-1.0)*2)*100
         print (1-z_cdf(-2.0)*2)*100
         print (1-z_cdf(-3.0)*2)*100
```

```
68.2689492137
95.4499736104
99.7300203937
```

30.3 χ^2-Verteilung

Gegeben sind n unabhängige z-verteilte Zufallsvariablen $z_1, z_2, ..., z_n$. Die Summe bildet die Zufallsvariable

$$\chi^2 = z_1^2 + z_2^2 + ... + z_n^2$$

mit n-Freiheitsgraden. Die Wahrscheinlichkeitsdichteverteilung für $\chi^2 > 0$ ist gegeben durch

$$p(\chi) = \frac{1}{2^{n/2}\Gamma(n/2)} \chi^{(n/2-1)} e^{-\frac{x}{2}}$$

mit der Gamma-Funktion $\Gamma(n/2)$.

Es gilt

$$E[\chi_n^2] = \mu_{\chi^2} = n$$

und

$$E[(\chi_n^2 - \mu_{\chi^2})^2] = \sigma_{\chi^2}^2 = 2n$$

Die Gamma-Funktion ist definiert als

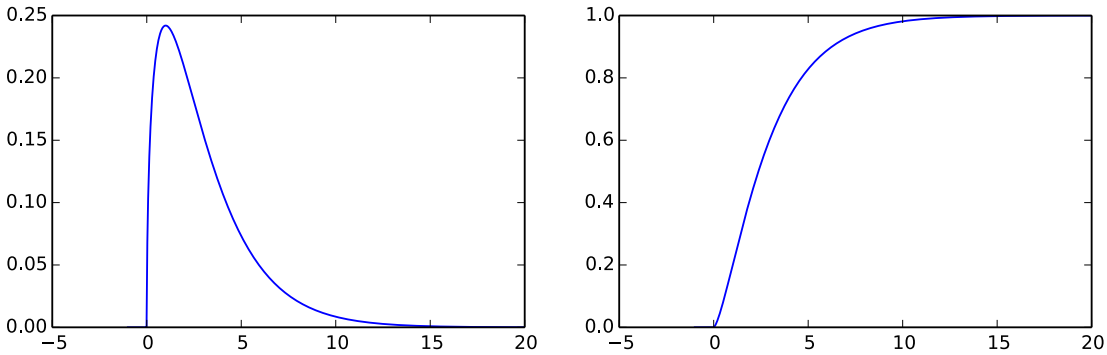
$$\Gamma(z) = \int_0^\infty t^{n-1} e^{-t}\, dt$$

Für positive Integer n ist das Ergebnis $\Gamma(n) = (n-1)!$.

30.4 χ^2-Verteilung: scipy.stats.chi2(x,df)

```
In [3]: x=linspace(-1,20,1000)
        n=3# Freiheitsgrade
        figure(figsize=(10,3))
        subplot(1,2,1)
        plot(x,stats.chi2.pdf(x,n))
        subplot(1,2,2)
        plot(x,stats.chi2.cdf(x,n))
```

```
Out[3]: [<matplotlib.lines.Line2D at 0x7f9fa2dcc050>]
```



30.5 Γ-Funktion: scipy.special.gamma(x)

```
In [45]: from scipy.special import gamma
         for i in range(1,20):
             print gamma(i)
```

```
1.0
1.0
2.0
6.0
24.0
120.0
720.0
5040.0
40320.0
362880.0
3628800.0
39916800.0
479001600.0
6227020800.0
87178291200.0
1.307674368e+12
2.0922789888e+13
3.55687428096e+14
6.40237370573e+15
```

30.6 F-Verteilung

Gegeben seien die Zufallsvariablen y_1 und y_2 mit χ^2-Verteilung und n_1 bzw. n_2 Freiheitsgraden. Die F-Verteilung ergibt sich aus dem Verhältnis

$$F_{n_1,n_2} = \frac{y_1/n_1}{y_2/n_2} = \frac{y_1 n_2}{y_2 n_1}$$

30.7 Student t-Verteilung

Gegeben sind zwei unabhängige Zufallsvariablen y und z. y folgt einer χ^2_n-Verteilung und z ist normalverteilt mit Freiheitsgrad n. Die Student $t-$Verteilung errechnet sich aus

$$t_n = \frac{z}{\sqrt{y/n}}$$

Es gilt

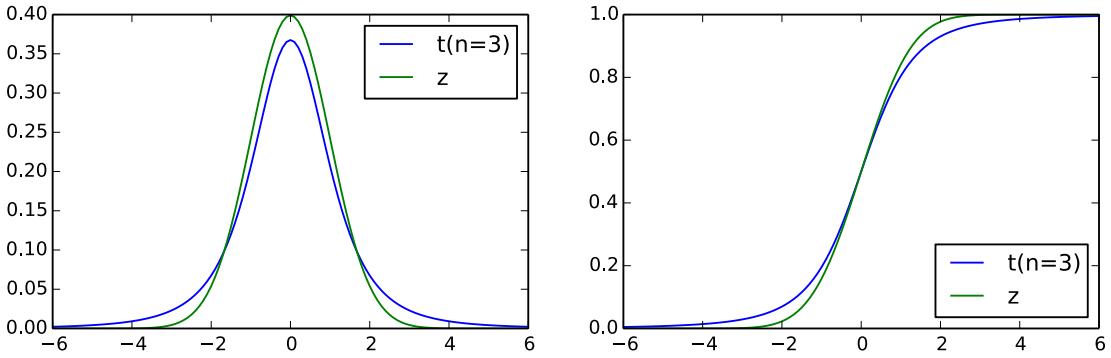
$$E[t_n] = \mu_{\chi^2} = 0$$

und

$$E[(t_n - \mu_t)^2] = \sigma_t^2 = \frac{n}{n-2}$$

für $n > 2$.

```
In [4]: x=linspace(-6,6,121)
        n=3 # Freiheitsgrade
        figure(figsize=(10,3))
        subplot(1,2,1)
        plot(x,stats.t.pdf(x,n),label='t(n='+str(n)+')')
        plot(x,stats.norm.pdf(x),label='z')
        legend()
        subplot(1,2,2)
        plot(x,stats.t.cdf(x,n),label='t(n='+str(n)+')')
        plot(x,stats.norm.cdf(x),label='z')
        legend(loc=4)

Out[4]: <matplotlib.legend.Legend at 0x7f9f9be8b5d0>
```



31 Stichproben-Verteilungsfunktionen

31.1 Stichproben-Mittel bei bekannter Varianz

N unabhängige Messungen x (normalverteilt) mit Mittelwert μ_x und Varianz σ_x^2

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

Der Mittelwert beträgt

$$\mu_{\bar{x}} = \mu_x$$

mit Varianz

$$\sigma_{\bar{x}}^2 = \frac{\sigma_x^2}{N}$$

Der sogenannte Standardfehler (gegeben durch die Standardabweichung σ) reduziert sich proportional zu $\frac{1}{\sqrt{N}}$

31.2 Stichproben-Varianz

Die Stichproben-Varianz ist gegeben als

$$s^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2$$

Für N unabhängige Messungen x (normalverteilt) mit Mittelwert μ_x und Varianz σ_x^2 gilt

$$\sum_{i=1}^{N}(x_i - \bar{x})^2 = \sigma_x^2 \chi_n^2$$

mit der χ_n^2-Verteilung mit $n = N - 1$ Freiheitsgraden. Es folgt für die Wahrscheinlichkeit

$$Prob[s^2 > \frac{\sigma_x^2 \chi_{n,\alpha}^2}{n}] = \alpha$$

32 Auswahl spezieller Prüfverfahren

32.1 Vergleich zweier Mittelwerte

Die Stichproben-Mittelwerte $\bar{a}$ und $\bar{b}$ sollen auf signifikanten Unterschied geprüft werden. Bei gleichem Stichprobenumfang $n_a = n_b$ errechnet sich die Wahrscheinlichkeit aus den Quantilen der t-Verteilung

$$\hat{t} = \frac{|\bar{a} - \bar{b}|\sqrt{n}}{s_a^2 + s_b^2}$$

mit der Anzahl der Freiheitsgrade beträgt $\Phi = 2n - 2$ $(n = n_a = n_b)$

32.2 Vergleich Stichproben-Mittel mit bekanntem Mittelwert

Der Stichproben-Mittelwert $\bar{a}$ soll auf signifikanten Unterschied hinsichtlich des bekannten Mittelwertes μ geprüft werden.

$$\hat{t} = |\bar{a} - \mu|\frac{\sqrt{n}}{s^2}$$

Die Anzahl der Freiheitsgrade beträgt $\Phi = n - 1$.

32.2.1 Beispiel

Beispiel aus Schönwiese Seite 138.

Ein geowissenschaftlicher zu untersuchender Prozess folge in guter Näherung der Normalverteilung mit μ=10 und σ=2.7 Maßeinheiten. Es ist zu prüfen, ob die Stichprobe mit $\bar{a} = 7.5$ und s=1.8 damit vereinbar ist, wobei der Stichproben-Umfang $n = 50$ betragen soll. Der Tabellenwert für das t-Quantil beträgt $t(\Phi = 49, \alpha = 0.05) \approx 1.68$.

```
In [14]: mu=10.0
         s=2.7

         a=7.5
         sa=1.8

         n=50.0
         df=n-1

         t=abs(a-mu)*sqrt(n)/s
         print t
```

```
6.54728501099
```

```
In [16]: a=0.05
         df=n-1
         x=linspace(-10,0,10000)
         print abs(x[argmax(stats.t.cdf(x,df)>a)])
         print abs(x[argmax(stats.norm.cdf(x)>a)]) # Für große n ist das t-Quantil ähnlich der Normalve
```

```
1.67616761676
1.64416441644
```

32.2.2 Folgerung

Es ist 6.54> 1.67

Daraus folgt, dass der Messwert nicht mit dem Prozess vereinbar ist.

33 Quantile

In [102]:
```
# Quantile (Verteilunfsfunktion) der Student-Verteilung (tV)
# Vergleiche Schönwiese A.3 (S. 287)
# oder http://de.wikipedia.org/wiki/Studentsche_t-Verteilung

a=0.05
x=linspace(-10,0,10000)
for df in range(1,10): #Freiheitsgrade
    print df," %1.3f " % abs(x[argmax(stats.t.cdf(x,df)>a)])
print '...'
df=49
print df," %1.3f " % abs(x[argmax(stats.t.cdf(x,df)>a)])
print '-------------------------------------------------------------------------------'
print 'Vergleich von t-Verteilung und Normalverteilung für große Anzahl Freiheitsgrade'
df=10000
print df," %1.3f " % abs(x[argmax(stats.t.cdf(x,df)>a)])

print df," %1.3f " % abs(x[argmax(stats.norm.cdf(x)>a)])
```

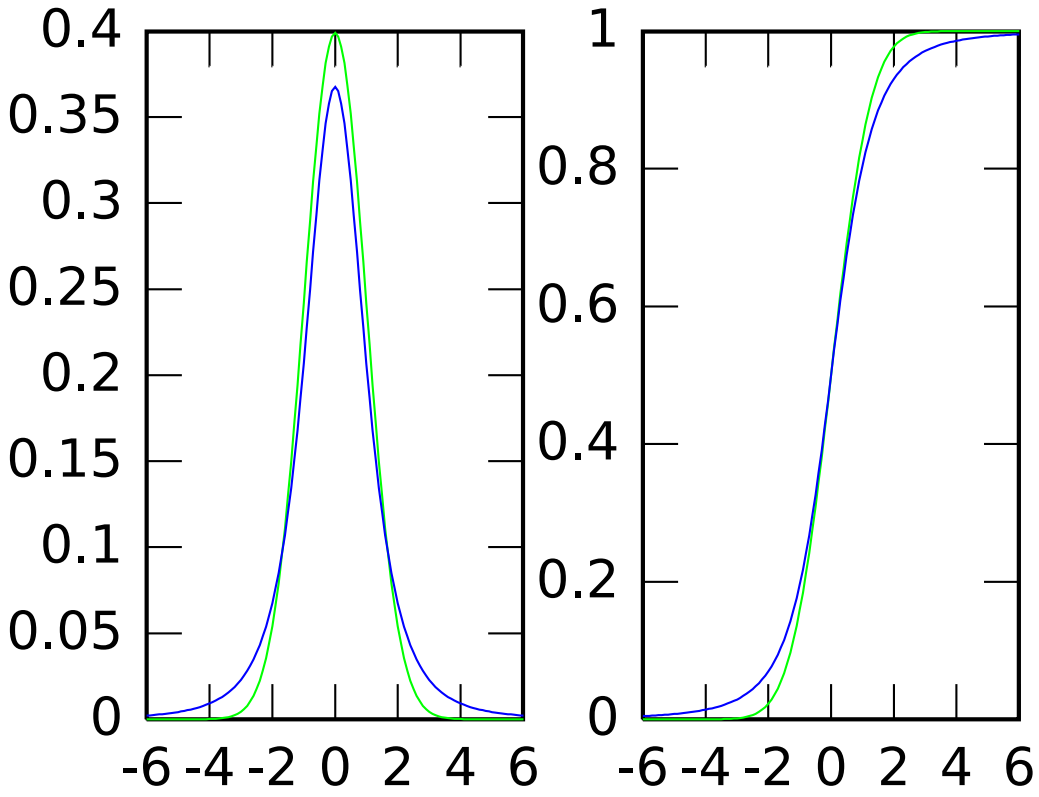
```
1  6.314
2  2.919
3  2.353
4  2.131
5  2.014
6  1.942
7  1.894
8  1.859
9  1.832
...
49  1.676
-------------------------------------------------------------------------------
Vergleich von t-Verteilung und Normalverteilung für große Anzahl Freiheitsgrade
10000  1.644
10000  1.644
```

33.1 Prüfung auf Daten-Unabhängigkeit

Vorsicht ist geboten, wenn die Daten nicht unabhängig sind. Dies ist der Fall, wenn z.B. Trends oder Autokorrelationen vorliegen. Weisen die Daten Autokorrelationen auf, sind sie nicht mehr unabhängig! Die Anzahl der Freiheitsgrade reduziert sich. Dies muss bei der Berechnung der Wahrscheinlichkeit berücksichtigt werden.

33.2 Octave-Beispiele

In [12]:
```
x=linspace(-6,6,121);
n=3 # Freiheitsgrade
subplot(1,2,1);
plot(x,normpdf(x,0,1),'g-');
hold;
plot(x,tpdf(x,n),'b-');
subplot(1,2,2);
plot(x,normcdf(x,0,1),'g-');
hold;
plot(x,tcdf(x,n),'b-');
```



Out[12]: n = 3

Quellen

- Bendat und Piersol, Random Data, Wiley 1986
- Schönwiese, Praktische Statistik, 2013

34 Zentraler Grenzwertsatz

34.1 Verbundwahrscheinlichkeit

Die Verbundwahrscheinlichkeit $P(x, y)$ ist die Wahrscheinlichkeit für das gleichzeitige Auftreten der Bedingung $x(k) \leq x$ und $y(k) \leq y$

$$P(x, y) = \text{Prob}(x(k) \leq x \quad \text{und} \quad \text{y(k)} \leq \text{y})$$

Die Verbundwahrscheinlichkeitsdichte $p(x, y)$ ist definiert als

$$p(x, y) = \lim_{\substack{\Delta x \to 0 \\ \Delta y \to 0}} \left(\frac{\text{Prob}(x < x(k) \leq x + \Delta x \quad \text{und} \quad \text{y} < \text{y(k)} \leq \text{y} + \Delta \text{y})}{\Delta x \Delta y} \right)$$

Es gilt

$$p(x) = \int_{-\infty}^{\infty} p(x, y) dy$$

$$p(y) = \int_{-\infty}^{\infty} p(x,y)dx$$

34.2 Summe zweier gleichverteilter Zufallsvariablen

Gegeben seien zwei gleichverteilte (auch rechteckverteilt genannt), unabhängige Zufallsvariablen x und y

$$p_1(x) = \begin{cases} \frac{1}{a} & 0 \leq x \leq a \\ 0 & \text{sonst} \end{cases}$$

$$p_2(y) = \begin{cases} \frac{1}{a} & 0 \leq y \leq a \\ 0 & \text{sonst} \end{cases}$$

Für die Summe der Zufallsvariablen $z(k) = x(k) + y(k)$ ergibt sich die Verbundwahrscheinlichkeitsdichte $p(x,y)$

$$p(x,y) = p(x, z-x)$$

und die Wahrscheinlichkeitsdichteverteilung $p(z)$

$$p(z) = \int_{-\infty}^{\infty} p(x, z-x)dx$$

Für unabhängige Zufallsvariablen x, y gilt

$$p(x,y) = p_1(x)p_2(y) = p_1(x)p_2(z-x)$$

und somit

$$p(z) = \int_{-\infty}^{\infty} p_1(x)p_2(z-x)dx$$

Es folgt

$$p(z) = \begin{cases} \frac{z}{a^2} & 0 \leq z \leq a \\ \frac{2a-z}{a^2} & a \leq z \leq 2a \\ 0 & \text{sonst} \end{cases}$$

Die Summe zweier gleichverteilter, unabhängiger Variablen gehorcht also einer Dreiecksverteilung.

34.3 Zentraler Grenzwertsatz

Es lässt sich zeigen, dass sich die Verteilungsfunktion für eine Summe unabhängiger Zufallsvariablen einer Gaußverteilung nähert. Dies ist die Aussage des Zentralen Grenzwertsatzes:

Die Summe unabhängiger Zufallsvariablen (mit endlicher Varianz) folgt asmptotisch einer Gaußverteilung

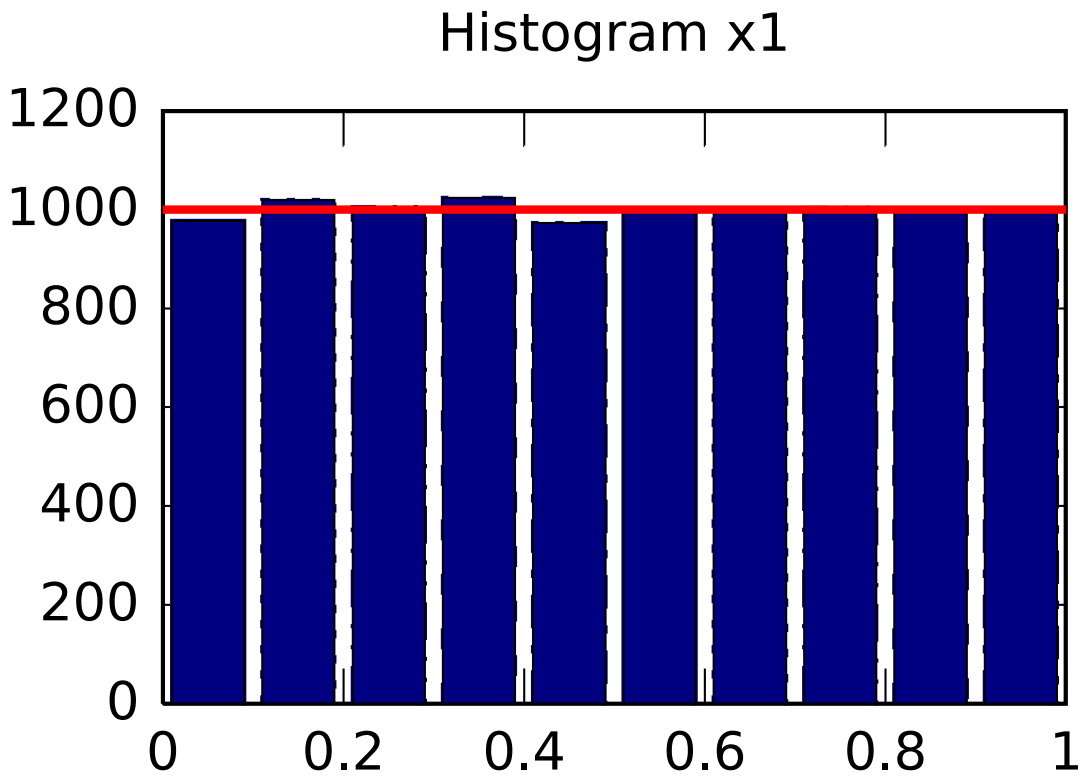
Im Folgenden zeigen wir exemplarisch die Gültigkeit anhand von numerischen Experimenten und verzichten auf einen mathematischen Beweis. Der Beweis kann mit Hilfe von charakteristische Funktionen (inverse Fouriertransformation der Verteilung) geführt werden.

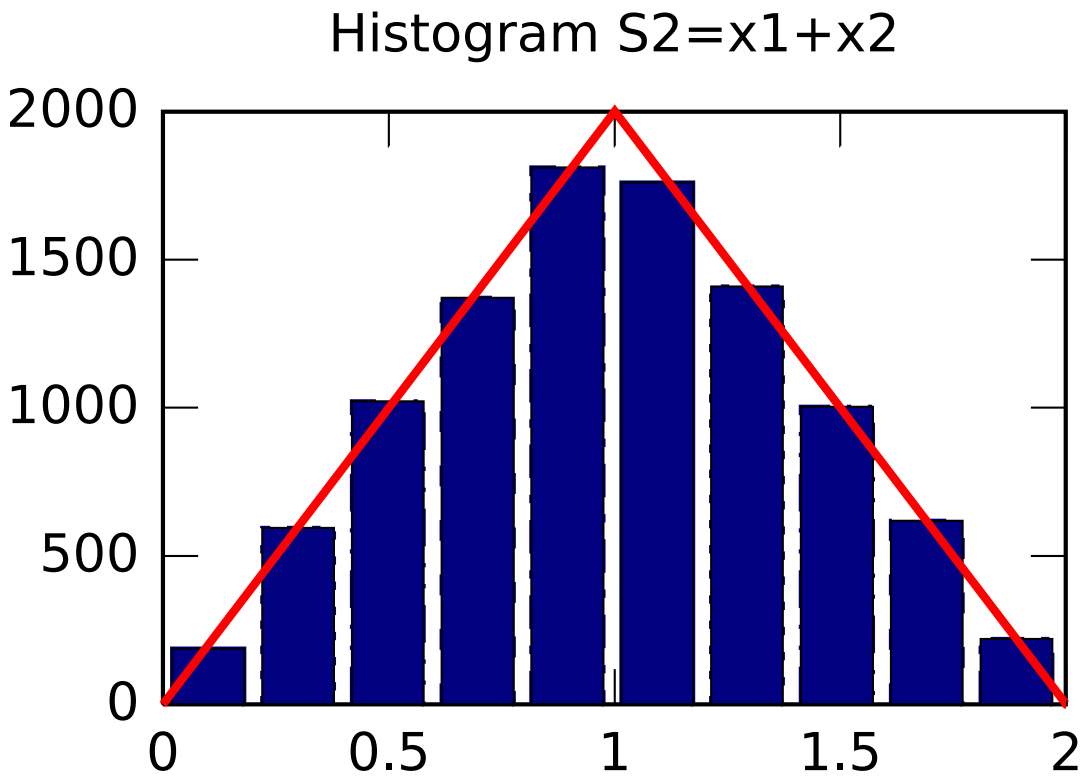
34.4 Beispiel Summe zweier gleichverteilter Zufallsvariablen

Seien x_1 und x_2 zwei im Wertebereich [0,1] gleichverteilte Zufallsvariablen. Es sei S_2 die Summe aus den Zufallsvariablen x_1 und x_2.

Die Zufallsvariable S_2 weist eine Dreiecksverteilung auf, wie das folgende Exeriment belegt.

```
In [210]: N=10000; % Anzahl der Realisationen
          a=1;
          x1=rand(N,1)*a;
          x2=rand(N,1)*a;

          hist(x1,10);
          hold();
          title('Histogram x1')
          plot([0,a],[N/10,N/10],'r-','LineWidth',2)
          figure()
          hist(x1+x2);
          hold();
          title('Histogram S2=x1+x2')
          plot([0,a,2*a],[0,2*N/10,0],'r-','LineWidth',2)
```





34.5 Beispiel Summe N gleichverteilter Zufallsvariablen

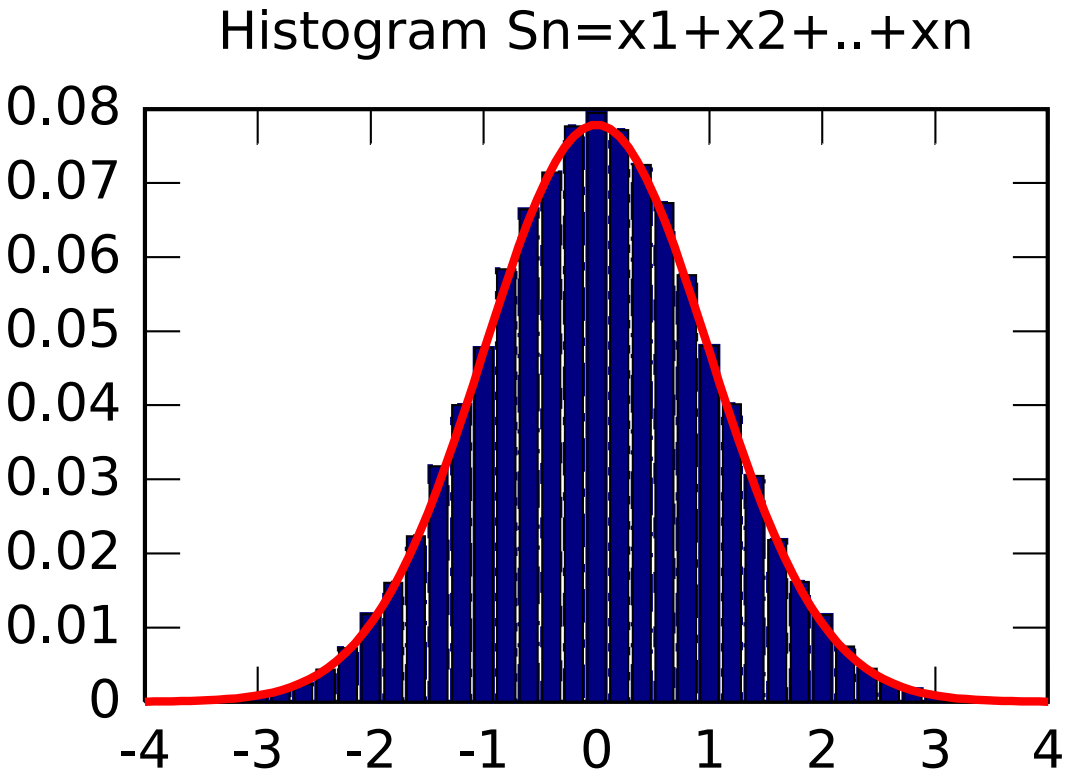
Die Summe $S_N = \sum_{i=1}^{N} x_i$ einer Reihe gleichverteilter Zufallsvariablen nähert sich mit steigendem N schnell einer Gausverteilung. Die standardisierte Zufallsvariable Z

$$Z = \frac{S_N - \mu_{S_N}}{\sigma_{S_N}}$$

folgt einer Normalverteilung. Dieses Beispiel zeigt die Bedeutung der Normalverteilung für die Statistik, denn jede Summe von Zufallsvariablen folgt im Grenzfall für große N einer Gauß- bzw. nach Standardisierung einer Normalverteilung.

```
In [192]: N=100000; % Anzahl Realisierungen von x
          M=12; % Anzahl Summation

          % Summe
          S=rand(N,1)*2-1;
          for i=1:M
              S=S+rand(N,1)*2-1;
          end

          %Standardisierung
          Z=(S-mean(S))/std(S);

          Nbins=41;
```

```
Xbins=linspace(-4,4,Nbins);
hist(Z,Xbins,1);
x=linspace(-4,4,100);
y=normpdf(x,0,1)/Nbins*8;
hold()
plot(x,y,'r-','LineWidth',2)
xlim([-4,4])
title('Histogram Sn=x1+x2+..+xn')
```



35 Hypothesentest: Verteilungsfunktionen

Oft ist die theoretische Verteilungsfunktion einer beobachteten Messgröße bzw. Zufallsvariablen nicht bekannt. Dann werden Annahmen über die zugrundeliegende statistische Verteilungsfunktion gemacht. Wir behandeln im folgenden, wie man diese Annahmen überprüft. Diese Art von Hypothesentest wird auch Anpassungstest genannt.

Es gibt verschiedene Hypothesentests, um zu untersuchen, ob eine beobachtete Häufigkeitsverteilung mit einer theoretischen Verteilungsfunktion in statistischer Übereinstimmung ist. Im Gegensatz zu anderen Hypothesentests ist die Annahme der Nullhypothese erwünscht, d.h. dass die Verteilungsanpassung als statistisch vertretbar gewertet wird.

Zwei wichtige Anpassungstests sind der Chi-Quadrat und der Kolomogorow-Smirnow-Test (KS-Test). Im Gegensatz zum Chi-Quadrat-Test ist der KS-Test auch für kleine Stichproben geeignet und robuster gegenüber Extremwerten. Der KS-Test kann auch zur Prüfung verwendet werden, ob zwei Zufallsvariablen die gleiche Verteilung besitzen.

Literatur C.D. Schönwiesse, “Praktische Statistik”, Kapitel 8.2.10

35.1 Problemstellung Anpassungstest: Vergleich einer empirischen Stichprobe (SP) mit einer theoretischen Grundgesamtheit (GG)

Gegeben sei eine beobachtete Häufigkeitsverteilung für eine beliebige Anzahl Stichproben.

Frage: sind beobachtete und theoretische Häufigkeitsverteilung im Einklang? Oft reicht ein Test nicht aus, um die Frage zu beantworten. Der Chi-Quadrat-Test ist anfällig gegenüber kleinen Abweichungen in den Randbereichen der Häufigkeitsverteilung. Daher sollte eine Ablehnung der Nullhypothese noch durch einen alternativen Test (z.B. KS-Test) verifiziert werden.

35.1.1 χ^2-Test

Für eine hinreichend große Stichprobe liefert der χ^2-Test eine Antwort.Gegeben sei $H_k(SP)$, die Häufigkeitsverteilung der Stichprobe (SP), die Häufigkeitsverteilung $H_k(GG)$ der Grundgesamtheit (GG: theoretische Verteilung). Die Testgröße χ^2 berechnet sich aus

$$\bar{\chi}^2 = \sum_{k=1}^{K} \frac{[H_k(SP) - H_k(GG)]^2}{H_k(GG)}$$

mit Freiheitsgrad $\Phi = K - Z$, wobei K die Klassenanzahl und Z die Anzahl der geschätzten Parameter der Verteilungsfunktion ist. Wird z.B. der Mittelwert μ und die Standardabweichung σ aus der Stichprobe geschätz, so wird $Z = 2$ und der Freiheitsgrad reduziert sich um zwei.

35.2 Kolmogoroff-Smirnoff-Test (KS-Test)

Die Prüfgröße $\hat{P}$ des KS-Tests berechnet sich aus

$$\hat{P} = \frac{|Max(KH_k(SP) - KH_k(GG))|}{n}$$

im Gegensatz zum χ^2-Test wird beim KS-Test mit den kumulativen Häufigkeiten gerechnet. Die Prüfgröße $\hat{P}$ wird mit einem Tabellenwert $P_{n,\alpha}$ verglichen, dabei gibt n den Stichprobenumfang und α die Irrtumswahrscheinlichkeit an.

Für eine Stichprobenanzahl $n > 35$ gilt die Näherung

$$P_{n,\alpha} = \frac{\sqrt{-\frac{1}{2}\ln\frac{\alpha}{2}}}{\sqrt{n}}$$

```
In [9]: %pylab inline
        sqrt(-0.5*log(0.2/2))
```

```
Populating the interactive namespace from numpy and matplotlib
```

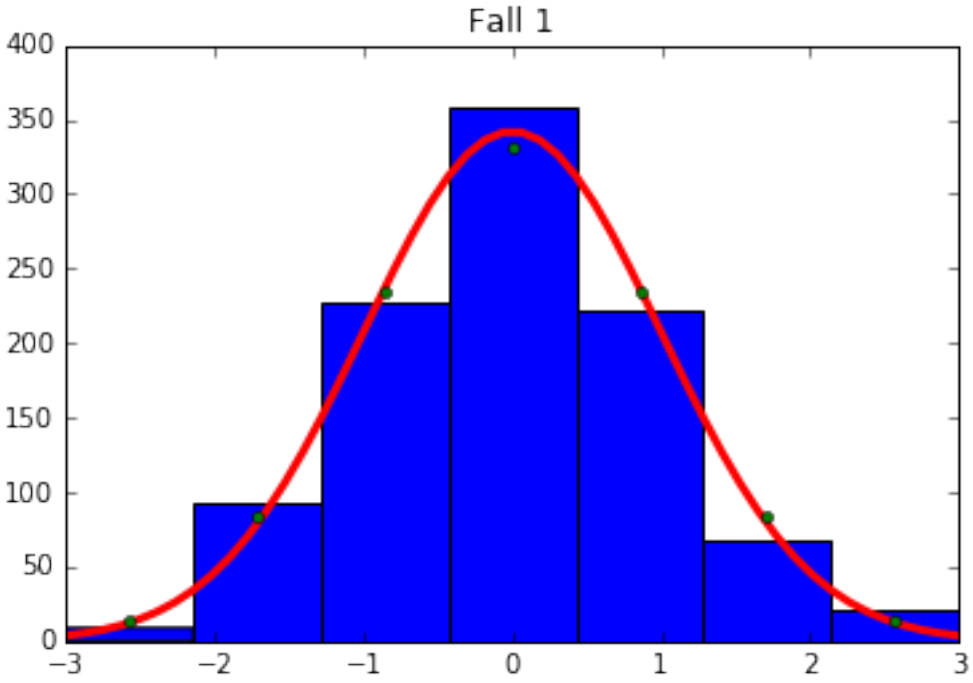
```
Out[9]: 1.0729830131446736
```

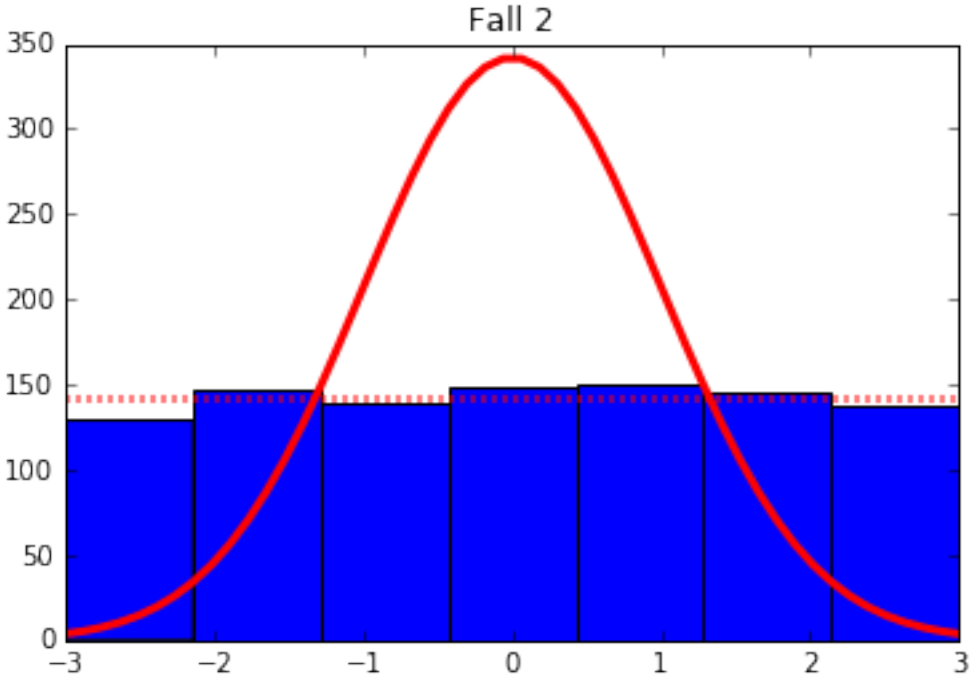
```
In [33]: %pylab inline
         import scipy.stats as stats

         N=1000 # Anzahl Stichproben
         K=7 # Anzahl Klassen
         r=3.0 # +- Rand des Histograms

         z1=randn(N)
         z2=rand(N)*r*2-r

         figure(1)
         H1=hist(z1,range=[-r,r],bins=K)
```

```
x=linspace(-r,r)
plot(x,stats.norm.pdf(x)*N/K*(2*r),'r-',linewidth=3)

x1=H1[1][1:]
x2=H1[1][:-1]
plot(x1+(x2-x1)/2,(stats.norm.cdf(x1)- stats.norm.cdf(x2))*N,'go',markersize=4)
title('Fall 1')
# Die theoretische Haeufigkeit wird durch das Integral unter der Kurve (=CDF) berechnet

figure(2)
H2=hist(z2,range=[-r,r],bins=K)
plot(x,stats.norm.pdf(x)*N/K*(2*r),'r-',linewidth=3)
plot(x,ones(len(x))*float(N)/(K),'r:',linewidth=3)
title('Fall 2')
Hk_SP1=H1[0]
Hk_GG1=(stats.norm.cdf(x1)- stats.norm.cdf(x2))*N

################################################
print("#### Fall 1 ####")
#print("Stichprobe: \n",Hk_SP1)
#print("GG:  \n", Hk_GG1)

chi2=sum((Hk_SP1-Hk_GG1)**2/Hk_GG1)
chi2_scipy=stats.chisquare(Hk_SP1,Hk_GG1) # Alternative Berechnung mit scipy.stats

print("Chi2: ",chi2)
print("Chi2_scipy: ",chi2_scipy)
print("--------------------")

Hk_SP2=H2[0]
Hk_GG2=(stats.norm.cdf(x1)- stats.norm.cdf(x2))*N

# Richtige Verteilung
#Hk_GG2=ones(len(x1))*float(N)/(K)

print("#### Fall 2 ####")
#print("Stichprobe: \n",Hk_SP2)
#print("GG:  \n", Hk_GG2)

chi2=sum((Hk_SP2-Hk_GG2)**2/Hk_GG2)
chi2_scipy=stats.chisquare(Hk_SP2,Hk_GG2)

print("Chi2: ",chi2)
print("Chi2_scipy: ",chi2_scipy)
print("--------------------")
```

```
Populating the interactive namespace from numpy and matplotlib
#### Fall 1 ####
Chi2:  11.8139627074
Chi2_scipy:  (11.813962707443345, 0.066250102156389018)
--------------------
#### Fall 2 ####
Chi2:  2186.14210402
Chi2_scipy:  (2186.1421040180235, 0.0)
```

```
--------------------

WARNING: pylab import has clobbered these variables: ['f']
`%matplotlib` prevents importing * from pylab and numpy
```





35.2.1 Was bedeutet der Wert von χ^2 ?

Je größer der Wert von χ^2 ist, desto schlechter ist die Anpassung der beobachteten Häufigkeitsverteilung mit der theoretischen Verteilung.

35.2.2 Wie sind die statistische Folgerungen?

Folgerungen können nur im statistischen Sinne gemacht werden. Wir können aus einer Tabelle der Quantile der χ^2-Verteilung die Wahrscheinlichkeiten in Abhängigkeit der Freiheitsgerade $\Phi = K - Z$ bestimmen.

Im obigen Fall testen wir, ob die Verteilung einer Standard-Normalverteilung folgt. Die Standard-Normalverteilung hat keine freien Parameter. Die Anzahl der Klassen K (Bins des Histogramms) beträgt $K = 7$. Daher folgt für unser Beispiel $Z = 0$ und $\Phi = K - Z$, d.h. $\Phi = 7$. Der Wert χ^2 ist kleiner als der tabellierte Wert $\chi^2 = 12.017$ bei einem Signifikanzniveau von 90%.

Wir können als die Aussagen formulieren:

- Fall1: Die beobachtete Häufigkeitsverteilung folgt einer Standard-Normalverteilung. Die Nullhypothese wird nicht zurückgewiesen.
- Fall2: Die beobachtete Häufigkeitsverteilung ist nicht normalverteilt. Die Nullhypothese wird zurückgewiesen.

35.3 Quantile der χ^2-Verteilung

Vergleiche Schönwiese A.4 (S. 288) oder Tabelle der Quantile der Chi-Quadrat-Verteilung

```
In [24]: stats.chi2.ppf(0.9,7)

Out[24]: 12.017036623780527

In [44]: F=[1,2,3,4,5,6,7,8,9,10,20,50,100]
         alpha=[0.1,0.9,0.95,0.99]
         print('f   1-a\t0.1\t0.9\t0.95\t0.99')
         for f in F:
             L=str(f)+'\t'
             for a in alpha:
                 L+=str('%3.2f'%stats.chi2.ppf(a,f))+'\t'
             print(L)
```

```
f   1-a        0.1        0.9        0.95        0.99
1        0.02        2.71        3.84        6.63
2        0.21        4.61        5.99        9.21
3        0.58        6.25        7.81        11.34
4        1.06        7.78        9.49        13.28
5        1.61        9.24        11.07        15.09
6        2.20        10.64        12.59        16.81
7        2.83        12.02        14.07        18.48
8        3.49        13.36        15.51        20.09
9        4.17        14.68        16.92        21.67
10        4.87        15.99        18.31        23.21
20        12.44        28.41        31.41        37.57
50        37.69        63.17        67.50        76.15
100        82.36        118.50        124.34        135.81
```

36 Bedeutung von Si und α ?

In der Tabelle werden die χ^2-Werte von links nach rechts stetig größer. Was bedeutet das für die Wahl des Signifikanzniveaus?

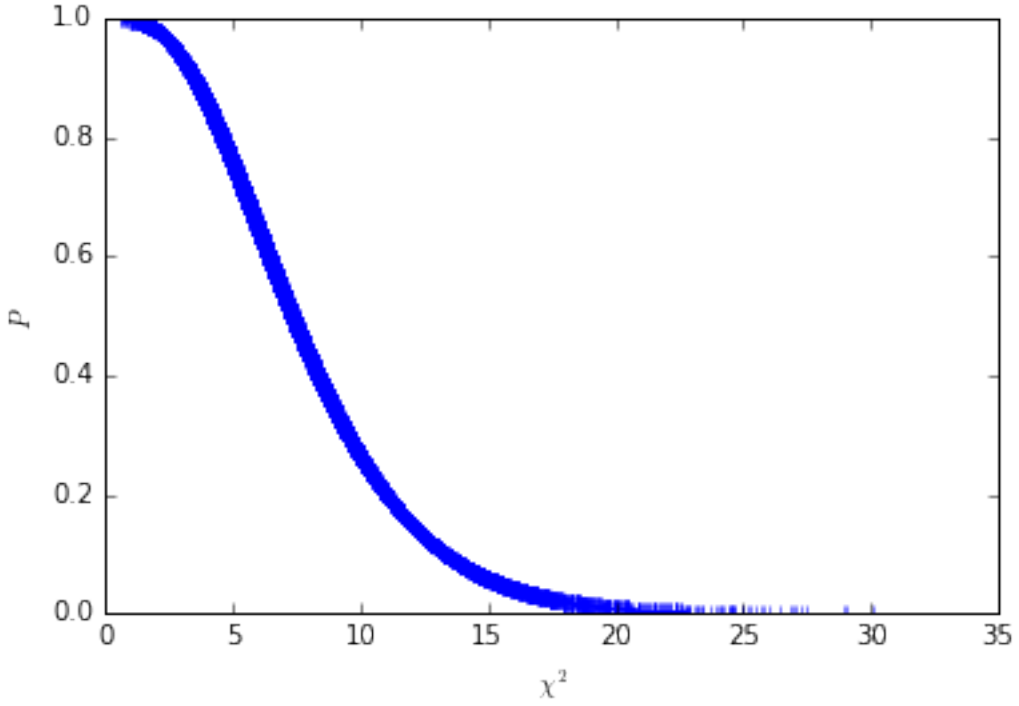
Beim Anpassungstest möchte man i.A. zeigen, dass eine Verteilung geeignet ist, d.h. die Nullhypothese bestätigen und nicht widerlegen. Die Nullhypothese ist, dass theoretische und beobachtete Verteilungsfunktionen übereinstimmen.

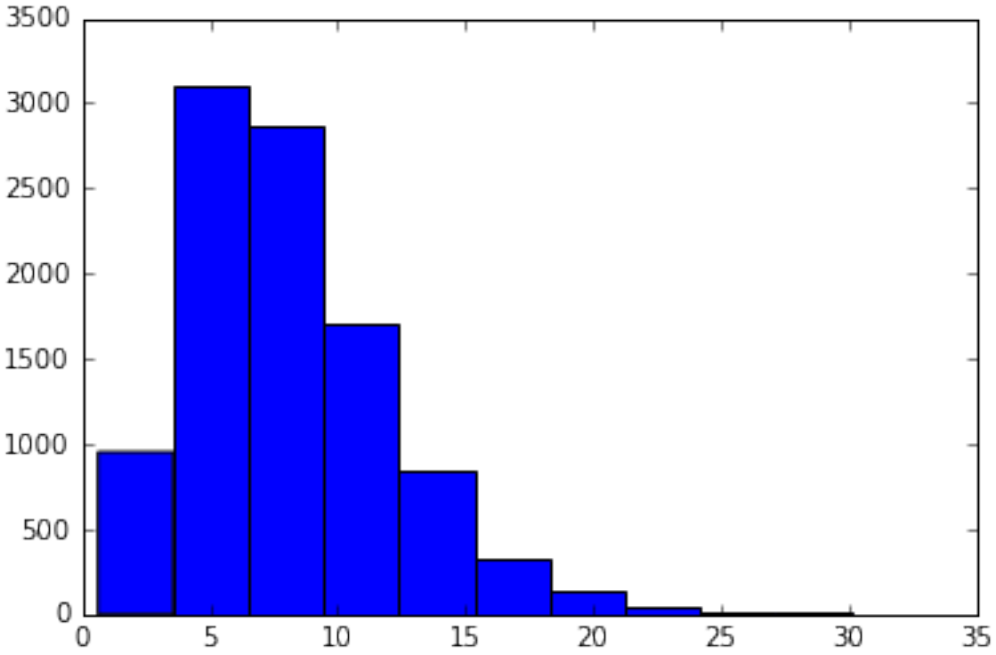
Beträgt nun z.B. Si=95%, so ist die Irrtumswahrscheinlichkeit 5% (0.05), d.h. die Nullhypothese wird mit einer Irrtumswahrscheinlichkeit von 5% akzeptiert. Für Si=50% beträgt die Irrtumswahrscheinlichkeit 50% (0.5). Es ist genauso wahrscheinlich, wie unwahrscheinlich, dass die Nullhypothese akzeptiert bzw. abgelehnt wird. Das entsprechende χ^2 ist aber kleiner! Ist dies eine logische Inkonsistenz? - Nein

Das Beispiel unten zeigt, dass bei einer Irrtumswahrscheinlichkeit von 99% etwa 99% der Nullhypothesen zurückgewiesen wird, obwohl die Stichprobe aus der theoretisch richtigen Verteilungsfunktion. Die Irrtumsrate kann also als "False-Hit-Rate" bzw. fehlerhafte Warnung bezeichnet werden. Nur die fast perfekte Übereinstimmung wird bei einer Irrtumswahrscheinlichkeit von 99% akzeptiert.

In [39]:
```
M=10000 # 10000 Monte-Carlo Experimente

N=1000 # Anzahl Stichproben
K=9 # Anzahl Klassen
r=1.5 # Rand des Histograms

CHI2=zeros(M)
P=zeros(M)

for i in range(M):
    z1=randn(N)
    z2=rand(N)*r*2-r
    H1=histogram(z1,range=[-r,r],bins=K)

    x1=H1[1][1:]
    x2=H1[1][:-1]
    H2=histogram(z2,range=[-r,r],bins=K)

    Hk_SP1=H1[0]
    Hk_GG1=(stats.norm.cdf(x1)- stats.norm.cdf(x2))*N

    (chi2_scipy,p)=stats.chisquare(Hk_SP1,Hk_GG1)#,ddof=K-2-1)
    CHI2[i]=chi2_scipy
    P[i]=p

alpha=0.99
print('Alpha= ',alpha)
print('Nullhypothese wird abgelehnt -> CHI2=', min(CHI2[P<alpha]))
print('Nullhypothese wird abgelehnt, Anzahl Fälle: ', sum(P<alpha), ' insgesamt:',M)
print('Nullhypothese wird abgelehnt, Anzahl Fälle: [%] ', sum(P<alpha)*100.0/M)

figure(3)
plot(CHI2,P,'+')
xlabel('$\chi^2$')
ylabel('$P$')

figure(4)
h=hist(CHI2)
```

```
Alpha=  0.99
Nullhypothese wird abgelehnt -> CHI2= 1.6531976213
Nullhypothese wird abgelehnt, Anzahl Fälle:  9918  insgesamt: 10000
Nullhypothese wird abgelehnt, Anzahl Fälle: [%]  99.18
```





36.1 Kolmogoroff-Smirnoff-Anpassungstest

36.1.1 Beispiel Octave/Matlab

Wir testen, ob eine Gleichverteilung vorliegt.

```
In [22]: alpha=0.05;
         for i=1:10 % 10x Wiederholen
             x=rand(20,1);
             p_ks=kolmogorov_smirnov_test(x, "unif", 0, 1);
             disp(p_ks)
             if p_ks<alpha
                 disp('Ablehnung der Nullhypothese')
             endif
         end
```

```
Out[22]:  0.14132
          0.63298
          0.037375
         Ablehnung der Nullhypothese
          0.42045
          0.65971
          0.22831
          0.46875
          0.66596
          0.46248
          0.30763
```

36.2 χ^2-Verteilung Octave

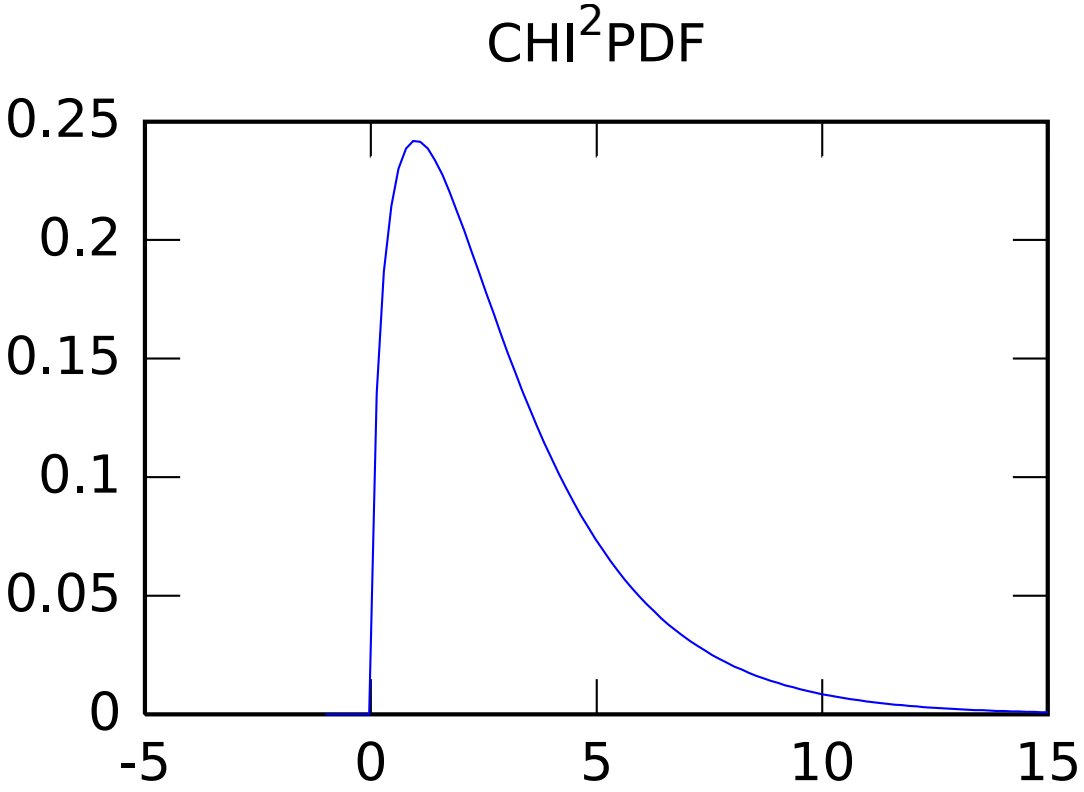
36.2.1 Quantile

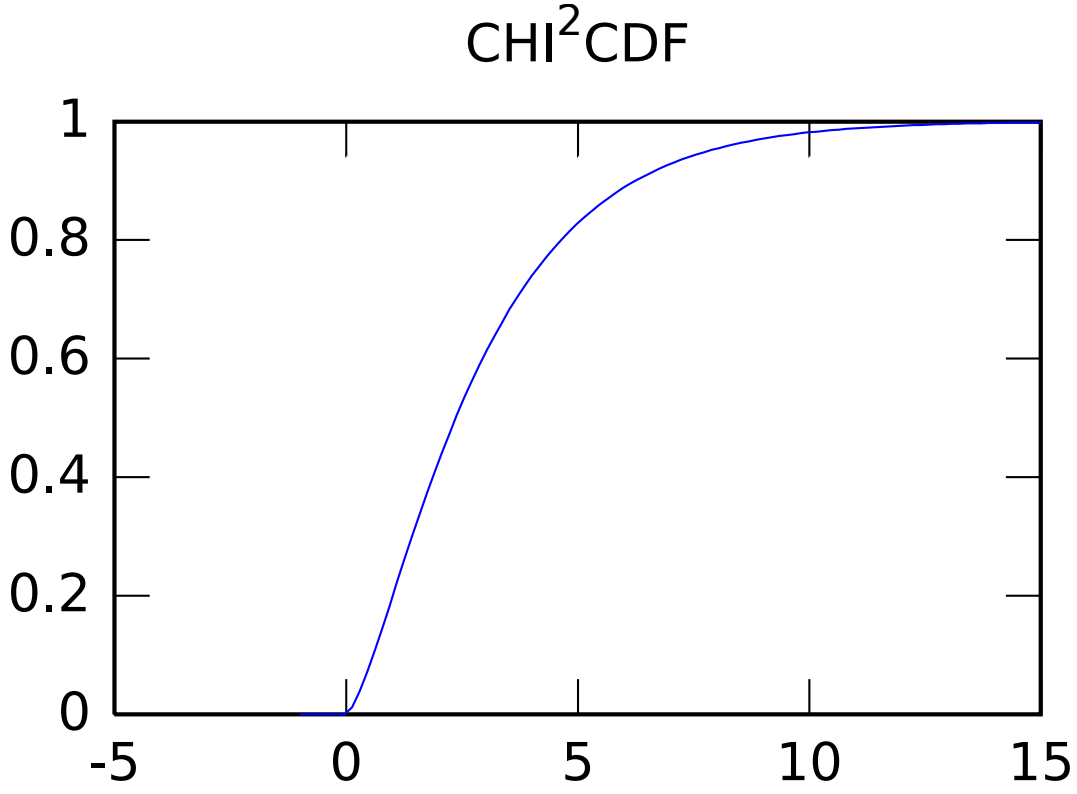
```
In [4]: chi2inv(0.9,7)
```

```
Out[4]: ans =  12.017
```

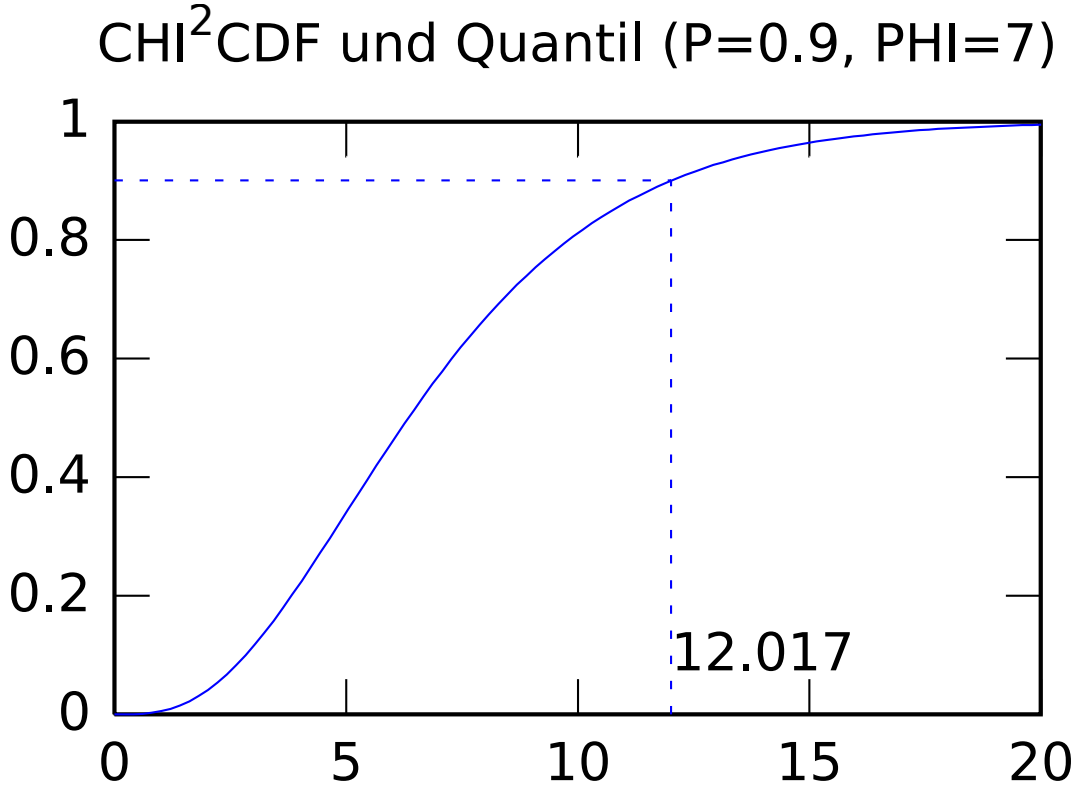
36.2.2 PDF und CDF

```
In [14]: x=linspace(-1,15);
         plot(x,chi2pdf(x,3));
         title('CHI^2 PDF');
         figure()
         plot(x,chi2cdf(x,3));
         title('CHI^2 CDF');
```


CHI2PDF
0.25
0.2
0.15
0.1
0.05
0
-5
0
5
10
15



In [28]:
```
figure()
x=linspace(0,20);
phi=7;
plot(x,chi2cdf(x,phi));
hold()
y=0.9;
q=chi2inv(y,phi);
x1=[q,q];
y1=[0,y];
plot(x1,y1,':');
x2=[0,q];
y2=[y,y];
plot(x2,y2,':');
text(q,0.1,num2str(q))
title('CHI^2 CDF und Quantil (P=0.9, PHI=7)');
```

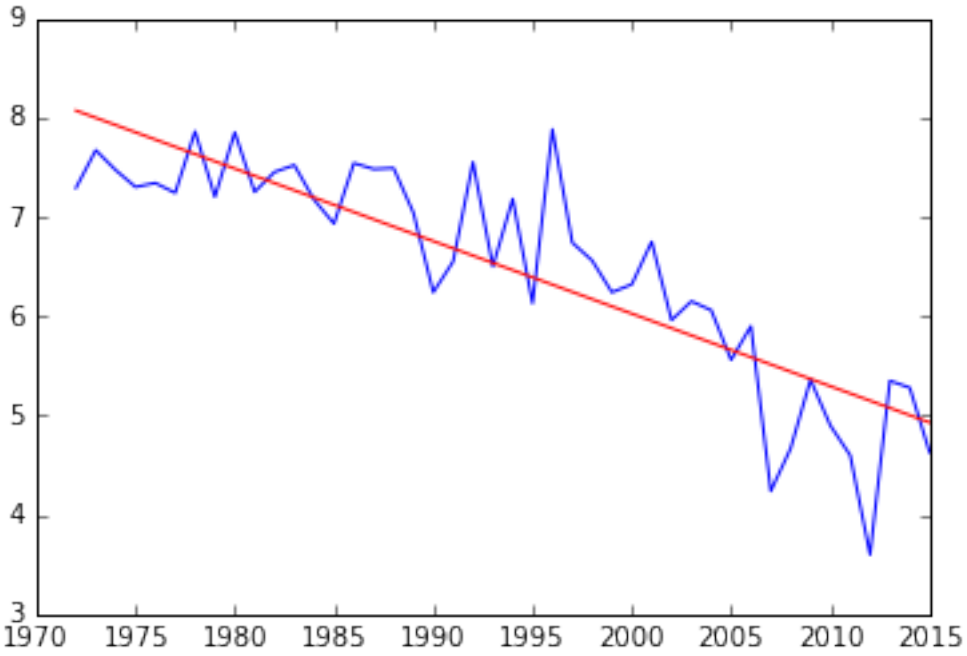


37 Signifikanz von Trends

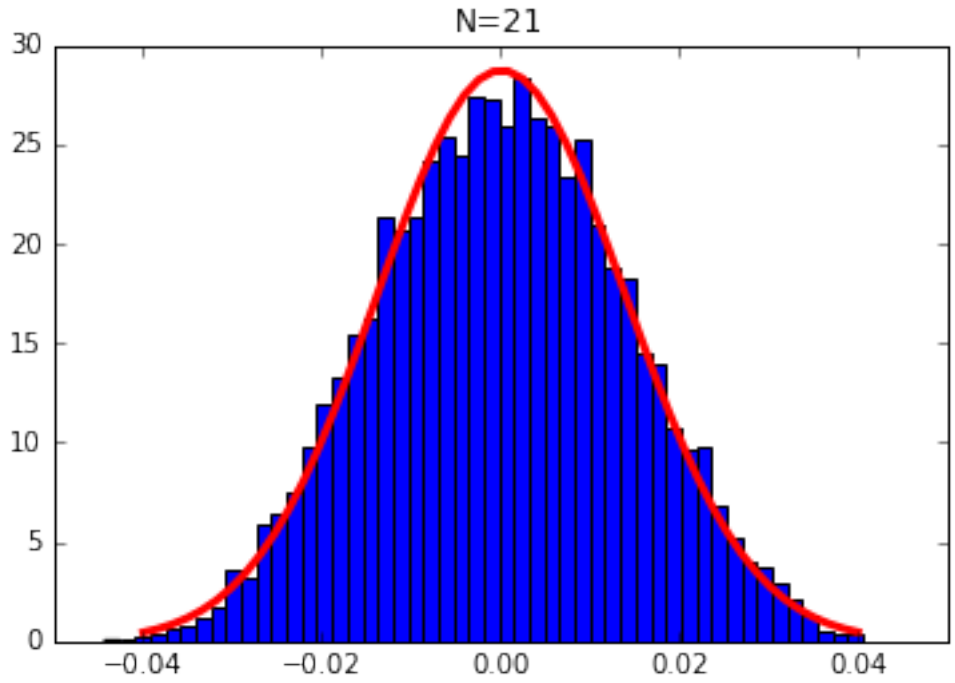
37.1 Meereis-Ausdehnung

```
In [32]: %pylab inline
         N=44
         t=linspace(1972,1972+N-1,N)
         Y=loadtxt("../stunde3/september_extent_1972_2015.txt")[0:N]
         plot(t,Y)
         p = polyfit(t,Y,1)
         y_line=p[0]*t+p[1]
         plot(t,y_line,"r")

         Y_o=Y.copy()

         M=10000
         a=zeros(M)
         for i in range(M):
             shuffle(Y)
             p = polyfit(t,Y,1)
             a[i]=p[0]
```

```
Populating the interactive namespace from numpy and matplotlib
```




```
In [13]: m=mean(a)
         s=std(a)
         x=linspace(-0.04,0.04)
         figure()
         h=hist(a,bins=50,normed=True)
         y=normpdf(x,m,s)
         plot(x,y,'r',linewidth=3)
         title('N=21')

Out[13]: <matplotlib.text.Text at 0x7fd85ef34898>
```




```
In [23]: p = polyfit(t,Y_o,1)
         p

Out[23]: array([ -2.62597403e-02,   5.93529957e+01])

In [15]: p[0]/s

Out[15]: -1.8902327505001972

In [17]: import scipy.stats as stats
         stats.norm.cdf(-1.89)

Out[17]: 0.029378980040409432
```

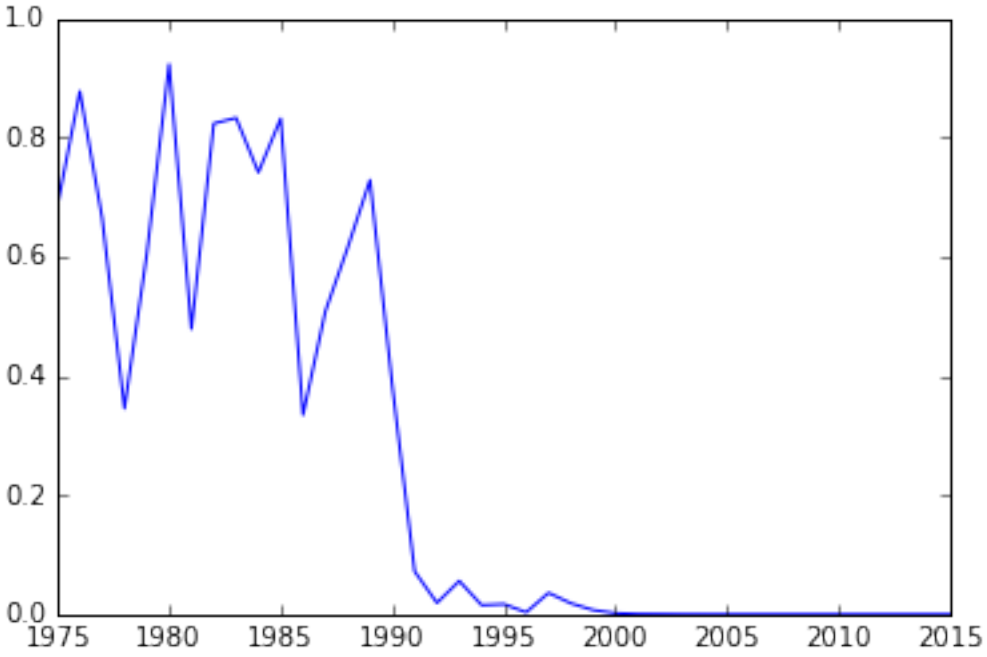
Der beobachete Trend liegt -1.89 σ vom Nullpunkt der Monte-Carlo-Gesamtheit entfernt. Dies entspricht einer Warscheinlichkeit von etwa 3%, dass es sich um einen rein zufälligen Trend handelt. Die Nullhypothese (kein Trend) wird zurückgewiesen.

37.2 Ab wann ist der Trend signifikant?

Betrachte p-Wert als Funktion der Zeit.

```
In [30]: import scipy.stats as stats
         M0,M1=3,44
         P=zeros(M1)
         P[:]=nan
         for n in range(M0,M1):
             slope, intercept, r_value, p_value, std_err = stats.linregress(t[0:n],Y_o[0:n])
             P[n]=p_value

         plot(t,P)
```

Out[30]: [<matplotlib.lines.Line2D at 0x7fd857fa5128>]



In [ ]:

38 Pandas für Zeitreihenanalyse

38.1 Beispiel Seismische Daten - Bestimmung der Wiederholrate

38.1.1 Motivation

Gibt es einen Zusammenhang zwischen Gasförderung und Erdbeben?
Literatur: http://www.nature.com/news/energy-production-causes-big-us-earthquakes-1.13372

38.1.2 Daten

Datenquelle: U.S. Geological Survey http://earthquake.usgs.gov/earthquakes/search/
Region: US-Bundesstaat Oklahamo
Andere Datenquellen: GEOFON (Archiv in Potsdam)

38.1.3 Methode

- Bestimmung der zeitlichen Abstände zwischen zwei Erdbeben der Stärke 3.
- Berechnung der Wiederholperiode aus aufeinanderfolgenden Ereignissen.

Inspirationsquelle für die folgende Datenanalyse: https://tamino.wordpress.com/2015/04/24/oklahoma-not-ok/

39 Beispiel Python-Code

39.1 Einlesen

Importieren des Pandas-Moduls, Einlesen, Umsortieren und Darstellen des Tabellenkopfes.

```
In [1]: %pylab inline
        import pandas as pd
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
In [2]: d=pd.read_csv("oklahoma_usgs.csv",parse_dates=[0]) # Einlesen
        d=d[::-1] # Umsortieren: älteste Daten an den Anfang
        d.head()
```

```
Out[2]:                        time  latitude  longitude  depth  mag magType  nst  \
        5364 1974-12-16 02:30:21.400    35.330    -97.480     10  2.6      ml  NaN
        5363 1980-11-02 10:00:49.300    35.472    -97.777      8  3.0     NaN  NaN
        5362 1981-07-11 21:09:22.540    34.884    -97.677      5  3.5    mblg  NaN
        5361 1984-01-06 17:14:49.800    36.160    -95.580      5  2.6      md  NaN
        5360 1984-03-03 11:42:02.400    35.510    -96.300      5  2.6    mblg  NaN

              gap  dmin  rms    ...                      updated  \
        5364  NaN   NaN  NaN    ...     2014-11-06T23:21:27.851Z
        5363  NaN   NaN  NaN    ...     2014-11-06T23:24:27.049Z
        5362  NaN   NaN  NaN    ...     2014-11-07T00:29:46.469Z
        5361  NaN   NaN  NaN    ...     2014-11-07T00:34:21.755Z
        5360  NaN   NaN  NaN    ...     2014-11-07T00:34:35.307Z

                                        place        type horizontalError  \
        5364  Oklahoma City urban area, Oklahoma  earthquake             NaN
        5363                            Oklahoma  earthquake             NaN
        5362                            Oklahoma  earthquake             NaN
        5361                            Oklahoma  earthquake             NaN
        5360                            Oklahoma  earthquake             NaN

             depthError  magError  magNst    status  locationSource magSource
        5364        NaN       NaN     NaN  reviewed               m       tul
        5363        NaN       NaN     NaN  reviewed               g       tul
        5362        NaN       NaN     NaN  reviewed             tul       tul
        5361        NaN       NaN     NaN  reviewed             tul       tul
        5360        NaN       NaN     NaN  reviewed             tul       tul

        [5 rows x 22 columns]
```

39.2 Pandas Zeitserien-Objekte

Ein Pandas-Zeitserien-Objekt besteht aus Werten und einem (Zeit-)Index. Der Index ermöglicht eine einfache Weiterbearbeitung.

```
In [3]: E=pd.Series(d['mag'].values,index=d['time']) # Generiere Zeitserienobjekt
        E.head()
```

```
Out[3]: time
        1974-12-16 02:30:21.400    2.6
        1980-11-02 10:00:49.300    3.0
```

```
1981-07-11 21:09:22.540    3.5
1984-01-06 17:14:49.800    2.6
1984-03-03 11:42:02.400    2.6
dtype: float64
```

```
In [4]: P=E['1980':'1984'] # Selektiere Zeitraum
        P=P[P.values>=3.0] # Nur Ereignisse >= 3
        P
```

```
Out[4]: time
        1980-11-02 10:00:49.300    3.0
        1981-07-11 21:09:22.540    3.5
        dtype: float64
```

```
In [5]: dt=P.index[1]-P.index[0] # Zeitliche Differenz aus dem Index
        #ergibt eine datetime.timedelta-Objekt
        dt.days # Uns interessieren nur die Tage
```

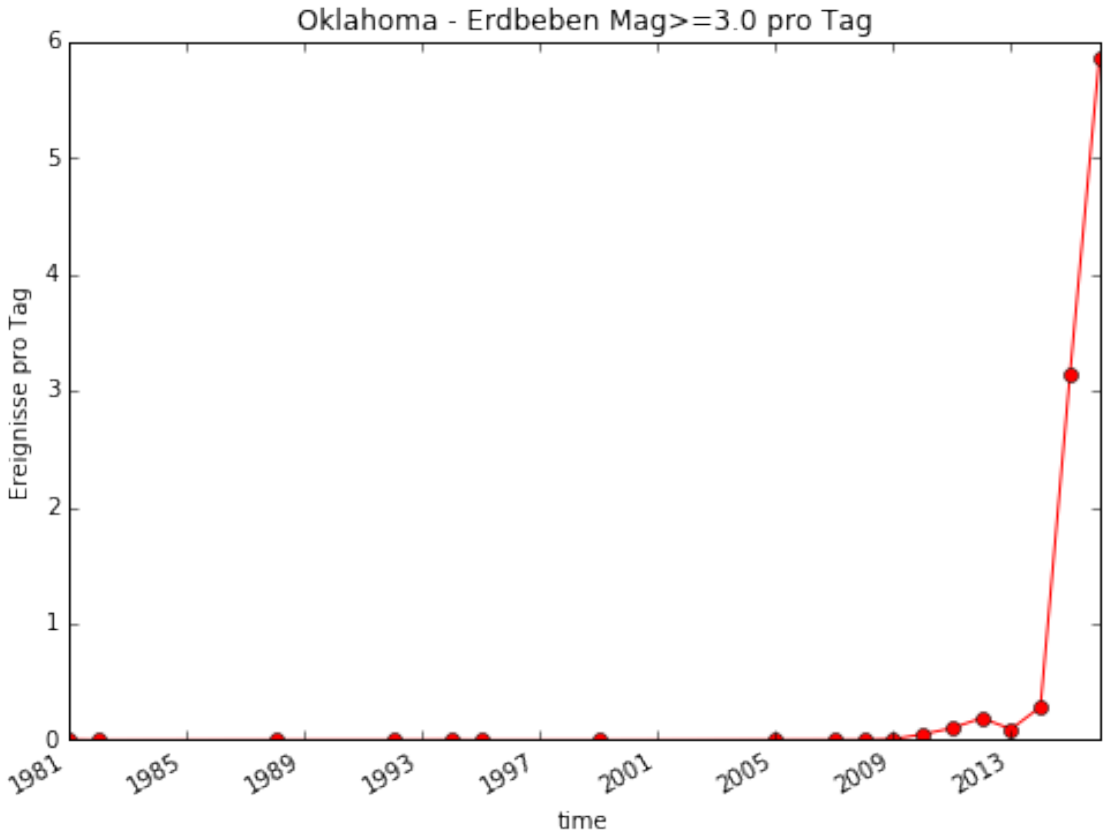
```
Out[5]: 251
```

39.3 Das vollständige Programm

Einlesen, Berechnung, Darstellen in weniger als 20 Zeilen.

```
In [6]: #Einlesen
        d=pd.read_csv("oklahoma_usgs.csv",parse_dates=[0]) # Einlesen
        d=d[::-1] #Umsortieren
        E=pd.Series(d['mag'].values,index=d['time'])

        # Berechnung
        Mag=3.0
        E=E[E.values>=Mag]
        E1, E2=E.index[1:], E.index[:-1]
        dt = [(E1[i]-E2[i]).days for i in range(len(E1))] # -> List comprehension!
        DT=pd.Series(dt,index=E2) # Generiere neues Zeitserienobjekt aus Zeitdifferenzen
        DT_mean=DT.resample('A-DEC',how='mean').dropna() # Annual Mean bis Dezember / entferne NaN
        F=1/DT_mean # Frequenz bzw. Wiederholrate

        # Darstellen
        figure(1,figsize=(8,6))
        F['1975':'2015'].plot(style='ro-')
        title('Oklahoma - Erdbeben Mag>='+str(Mag)+' pro Tag')
        ylabel('Ereignisse pro Tag')
```

```
Out[6]: <matplotlib.text.Text at 0x7fc2bf5c19b0>
```

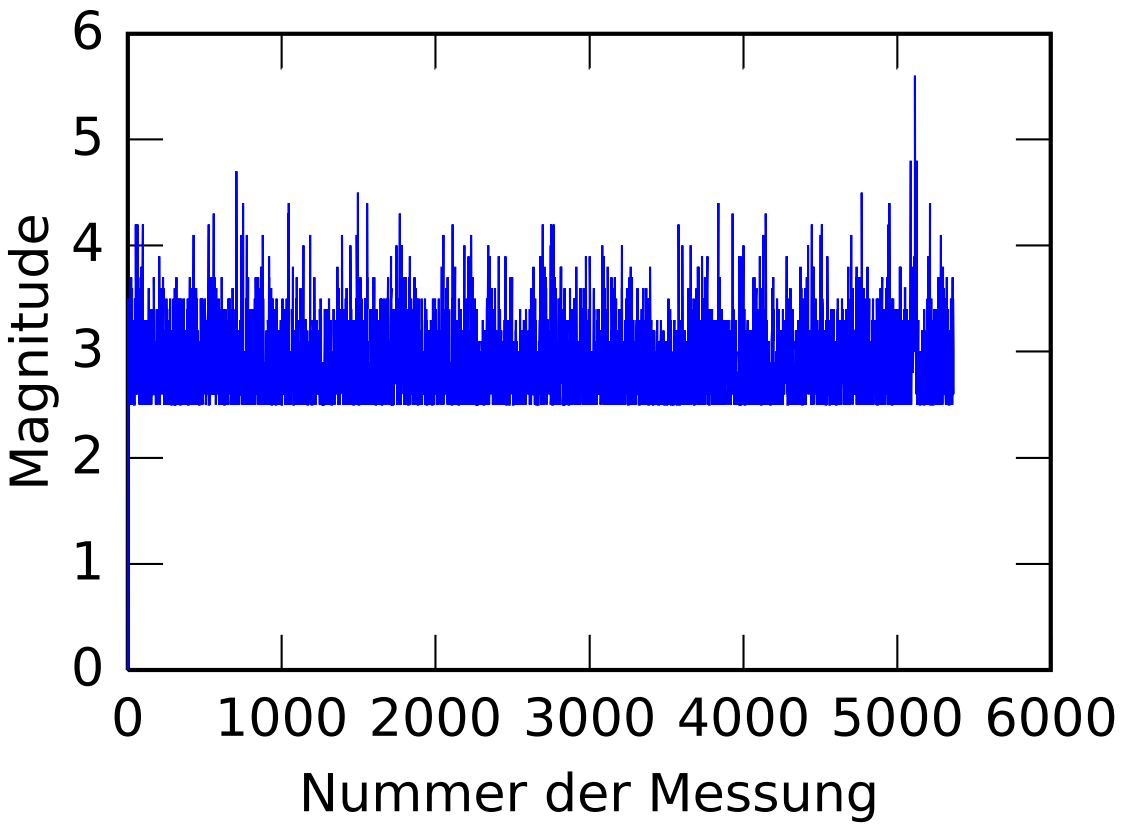


39.4 Octave-Version

Datumsfunktionen und Zeitserienobjekte sind in Octave/Matlab so wie oben nicht verfügbar. Die Umsetzung der obigen Methode bleibt als Übung.

```
In [21]: d=csvread("oklahoma_usgs.csv");
```

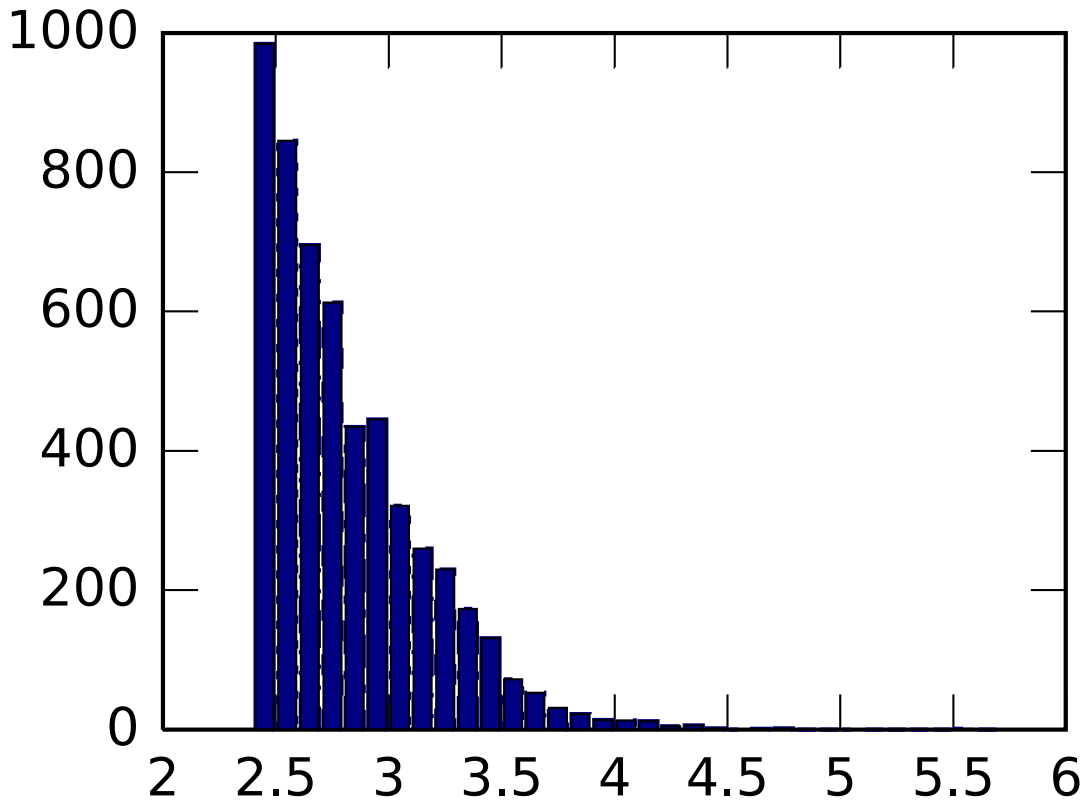
```
In [4]: plot(d(:,5));
        xlabel('Nummer der Messung');
        ylabel('Magnitude');
```




```
In [23]: m=d(2:5366,5);

In [13]: size(m)

Out[13]: ans =

            5366      1


In [24]: unique(m)

Out[24]: ans =

            2.5000
            2.6000
            2.7000
            2.8000
            2.9000
            3.0000
            3.1000
            3.2000
            3.3000
            3.4000
            3.5000
```

```
3.6000
3.7000
3.8000
3.9000
4.0000
4.1000
4.2000
4.3000
4.4000
4.5000
4.7000
4.8000
5.6000
```

In [29]:
```
hb=2.45:0.1:5.65;
hist(m,hb)
```



In [ ]:

40 Wahrscheinlichkeitsrechnung - Wiederholperiode

Sei P die Wahrscheinlichkeit für das Auftreten eines Ereignisses in einem Zeitraum ΔT, dann berechnet sich die Wiederholperiode τ aus

$$\tau = \frac{\Delta T}{P}$$

```
In [2]: %pylab inline

Populating the interactive namespace from numpy and matplotlib

In [3]: p=0.5

In [4]: n=4
        p**n

Out[4]: 0.0625

In [6]: n=5
        p**n

Out[6]: 0.03125

In [8]: n=7
        p**n

Out[8]: 0.0078125

In [11]: n=30
         p30=p**n

In [12]: p5=p**5

In [13]: 1/p5

Out[13]: 32.0

In [14]: 32*5

Out[14]: 160
```

Wenn es 5 Sekunden für 5 Würfe dauert, ist zu erwarten, dass nach 160 Sekunden (=32 Versuche a 5 Sekunden) 1x das Ereignis 5* Wappen auftritt

```
In [17]: 1/p30*30/60/60/24/365

Out[17]: 1021.443896499239
```

40.1 Jahrtausendereignis

30 * Wappen dauert > 1000 Jahre

```
In [ ]:
```

41 Spektralanalyse

Die Betrachtung von Zeitreihen im Frequenzraum ermöglicht die Bestimmung von periodischen Zyklen, z.B. dem Gezeitensignal. Die Zerlegung der Zeitreihe in ihre Frequenzanteile wird auch Harmonische Analyse genannt.

41.1 Fourier-Reihen

Gegeben sei eine periodische Funktion mit der Grundperiode T bzw. Grundfrequenz $f_0 = \frac{1}{T}$

$$x(t) = x(t \pm nT)$$

mit $n = 1, 2, 3, ...$

Mit einigen Ausnahmen lassen sich solche periodischen Daten in Fourier-Reihen entwickeln

$$x(t) = \frac{a_o}{2} + \sum_{n=1}^{\infty}(a_n \cos(2\pi n f_0 t) + b_n \sin(2\pi n f_0 t))$$

Die Fourier-Koeffizienten a_n, b_n erhält man durch Integration über ein Periode T, z.B. $-\frac{T}{2}$ bis $\frac{T}{2}$ oder von 0 bis T

$$a_n = \frac{2}{T}\int_0^T x(t)\cos(2\pi n f_0 t)dt$$

$$b_n = \frac{2}{T}\int_0^T x(t)\sin(2\pi n f_0 t)dt$$

Zu beachten ist, dass $\frac{a_o}{2} = \int_0^T x(t)dt$ der Mittelwert μ_x von $x(t)$ ist. Die Gleichungen können auch in anderer Form, z.B. mit der Kreisfrequenz $\omega = 2\pi f$ und $d\omega = 2\pi df$ geschrieben werden.

41.1.1 Amplituden- und Phasendarstellung

Durch trigonometrische Umformung lässt sich die Fourier-Reihe auch formulieren als

$$x(t) = \frac{a_o}{2} + \sum_{n=1}^{\infty}(\underbrace{A_n}_{=\sqrt{a_n^2+b_n^2}} \cos(2\pi n f_0 t - \underbrace{\Phi_n}_{=\tan^{-1}(\frac{b_n}{a_n})})$$

41.1.2 Darstellung mit komplexen Zahlen

Mittels der Euler-Beziehung $e^{-\imath\Theta} = \cos\Theta - \imath\sin\Theta$ folgt die Darstellung

$$x(t) = \frac{a_o}{2} + \sum_{n=-\infty}^{\infty} A_n e^{\imath\pi n f_0 t}$$

mit $A_n = \frac{1}{2}(a_k - \imath b_k)$ und

$$A_n = \frac{1}{T}\int_0^T x(t)e^{\imath\pi n f_0 t}$$

wobei $n = \pm1, \pm2, \pm3, ...$ nun auch negative Werte annimmt.

Auch wenn $x(t)$ eine reellwertige Zeitreihe ist, kann sie durch komplexwertige negative und positive Frequenzkomponenten beschrieben werden. Dabei gilt der Zusammenhang

$$A_n = |A_n|e^{-\imath\Theta_n}$$

mit $|A_n| = \frac{1}{2}\sqrt{a_n^2 + b_n^2} = \frac{x_n}{2}$ und $\Theta_n = \tan^{-1}(\frac{b_n}{a_n})$.

Für reelle $x(t)$ gelten die Symmetrien

$$|A_{-n}| = |A_n|$$

$$\Theta_{-n} = -\Theta_n$$

$$A_{-n} = |A_{-n}|e^{-\imath\Theta_{-n}} = |A_n|e^{\imath\Theta_n} = A_n^*$$

41.2 Fourier-Transformation

Gegeben sei eine Zeitreihe $x(t)$. Die Fourier-Reihe lässt sich erweitern zu dem Fourier-Integral

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-\imath 2\pi f t} dt$$

$X(f)$ wird auch direkte Fourier-Transformierte oder (Amplituden-)Spektrum genannt und existiert für die Bedingung

$$\int_{-\infty}^{\infty} |x(t)| dt < \infty$$

.

Die inverse Fourier-Transformation

$$x(t) = \int_{-\infty}^{\infty} X(f) e^{\imath 2\pi f t} df$$

ist i.A. eine komplexwertige Funktion

$$X(f) = X_R(f) - \imath X_I(f)$$

mit Real- und Imaginärteil X_R und X_I

$$X_R(f) = |X(f)| \cos(\Theta(f)) = \int_{-\infty}^{\infty} x(t) \cos(2\pi f t) dt$$

$$X_I(f) = |X(f)| \sin(\Theta(f)) = \int_{-\infty}^{\infty} x(t) \sin(2\pi f t) dt$$

bzw.

$$X(f) = \underbrace{|X(f)|}_{Magnitudenspektrum} \quad e^{\overbrace{-\imath \quad \Theta(f)}^{Phasenspektrum}}$$

Das quadrierte Magnitudenspektrum wird auch Leistungsspektrum genannt. Die graphische Darstellung, meist in logarithmischer Form, wird als Periodogramm bezeichnet.

41.3 Diskrete Fourier-Transformation

Für eine stationäre Zeitreihe von theoretisch unendlicher Länge existiert die Fourier-Transformation nicht, denn es gilt

$$\int_{-\infty}^{\infty} |x(t)| dt = \infty$$

Allerdings liegen tatsächlich gemessene Zeitreihen nur über ein endliches Zeitintervall T vor, und die bestimmte Fourier-Transformation

$$X_T(f) = X(f, T) = \int_0^T x(t) e^{-\imath 2\pi f t} dt$$

existiert immer. Für diskrete Frequenzen $f_n = \frac{n}{T}$ mit $n = \pm 1, \pm 2, \pm 3, ...$ ergibt sich die diskrete Fourier-transformation zu

$$X(f_n, T) = T A_n$$

mit

$$A_n = \frac{1}{T}\int_0^T x(t)e^{-\imath 2\pi f_n t}dt$$

Die Fourier-Transformation für diskrete Frequenzen f ist tatsächlich eine Fourier-Reihe.

41.3.1 Nyquist-Frequenz

Wird die Zeitreihe $x(t)$ an N-Punkten im Abstand Δt gemessen (abgetastet), so beträgt die längste Periode

$$T = N\Delta t$$

Dies führt zu der Grundfrequenz $f_0 = \frac{1}{T}$ und der Nyquist-Frequenz (Grenzfrequenz)

$$f_{Nyquist} = \frac{1}{2\Delta t}$$

Bei der Berechnung wird die Zeitreihe so behandelt, als wäre es eine zyklische Zeitreihe mit der Periode T.

41.4 Rechenregeln für die Fouriertransformation

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-\imath 2\pi f t}dt$$

$$x(t) = \int_{-\infty}^{\infty} X(f)e^{\imath 2\pi f t}df$$

Eigenschaft	Zeit-Raum	Frequenz-Raum
Linearität	$ax(t) - by(t)$	$aX(f) - bY(f)$
Zeitversatz	$x(t - t_0)$	$X(f)e^{2\pi\imath f t_0}$
Frequenzversatz	$x(t)e^{2\pi\imath f_0 t}$	$X(f - f_0)$
Differentiation	$\frac{dx(t)}{dt}$	$2\pi\imath f X(f)$
n-mal Differentiation	$\frac{d(x(t))^n}{dt^n}$	$(2\pi\imath f)^n X(f)$

41.4.1 Beispiel: Lösung einer DGL im Frequenzraum

Die bekannte Differentialgleichung für ein eindimensionales mechanisches System

$$m\frac{d^2y(t)}{dt^2} + c\frac{dy(t)}{dt} + ky(t) = F(t)$$

lässt sich für die anregende Kraft $F(t) = \delta t$ im Frequenzraum einfach lösen. Durch Fouriertransformation auf beiden Seiten erhält man

$$[-(2\pi)^2 m + \imath 2\pi f c + k]Y(f) = 1$$

$$Y(f) = \frac{1}{k - (2\pi)^2 m + \imath 2\pi f c}$$

42 Periodogramm

Gegeben sei eine Zeitreihe $x(t)$. Das Fourier-Integral

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-\imath 2\pi f t} dt$$

wird auch Amplitudenspektrum genannt. Das quadratische Amplitudenspektrum

$$P_{xx}(f) = |X(f)|^2$$

wird auch das Leistungsspektrum genannt. Die graphische Darstellung des Leistungsspektrums wird als Periodogramm bezeichnet.

Für zeitlich diskrete Zeitreihen sind Fourier-Integral und Fourier-Reihe identisch. Die Berechnung der Fourier-Reihe geschieht üblicherweise durch einen schnellen Algorithmus, der FFT (Fast Fourier Transform).

43 Abtasttheorem und Aliasing

Gegeben sei ein Signal welches Informationen mit einer maximalen Frequenz f_{max} enthält. Dieses Signal soll zeitdiskret gemessen (Sampling) und aus den diskreten Messpunkten rekonstruiert werden.

Das Abtastttheorem (Nyquist-Theorem) besagt, dass dieses Signal mindestens mit der doppelten Frequenz $f_N = 2f_{max}$ abgetastet werden muss, um eine exakte Rekonstruktion zu ermöglichen. In der praktischen Anwendung liefert eine Abtastung mit etwa 3 bis 6 Abtastwerten pro Wellenlänge gute Ergebnisse.

Ist das Abtasttheorem nicht erfüllt (Unterabtastung), gibt es Fehler. Diese werden als Aliasing-Fehler oder Aliasing-Effekte bezeichnet.

44 FFT Algorithmus

44.1 Diskrete Fouriertransformation (DFT)

DFT eines Vektors $\hat{x}$ der Länge N

$$\hat{x}(k) = \sum_{j=0}^{N-1} X(j) \underbrace{W_N^{jk}}_{e^{\frac{2\pi\imath}{N}}}$$

Inverse DFT

$$x(j) = \frac{1}{N} \sum_{j=0}^{N-1} \hat{x}(k)_j W_N^{-jk}$$

Beachte: Inverse kann als DFT der Funktion $\frac{1}{N}\hat{x}(-k)$ berechnet werden.

44.2 Ansatz für schnelle Lösung

Wenn $N = N_1 N_2$ ist, dann kann die DFT 1D-Gleichung als 2D-Gleichung beschrieben werden mit einer Umbenennung der Variablen

$$j = j(a,b) = aN_1 + b, 0 \leq a < N_2, 0 \leq b < N_1$$
$$k = k(c,d) = cN_2 + d, 0 \leq c < N_2, 0 \leq d < N_1$$

Nun wird die Eigenschaft $W_N^{m+n} = W_N^m W_N^n$ ausgenutzt und wir erhalten

$$\hat{x}(c,d) = \sum_{b=0}^{N_1-1} W_N^{b(cN_2+d)} \sum_{a=0}^{N_2-1} X(a,b) W_{N_2}^{ad}$$

Zur Berechnung benötigt man zwei Schritte, zunächst wird die innere Summe berechnet (für alle d)

$$\tilde{x}(b,d) = \sum_{a=0}^{N_2-1} X(a,b) W_{N_2}^{ad}$$

Dazu sind maximal $N_1 N_2^2$ arithmetische Rechenoperationen notwendig. Danach wird die Transformation

$$\sum_{b=0}^{N_1-1} W_N^{b(cN_2+d)} \tilde{x}(b,d)$$

berechnet, was mit $N_1 N_1^2$ Rechenschritten möglich ist.

Der FFT-Algorithmus kann effizient durch in-place Berechnungen kodiert werden, um Rechenschritte und Speicherplatz einzusparen.

44.2.1 Literatur zur FFT

- Cooley, James W., and John W. Tukey, 1965, An algorithm for the machine calculation of complex Fourier series, Math. Comput. 19: 297-301
- Computing in Science and Engineering Jan/Feb 2000, The FFT: An Algorithm the Whole Family Can Use, Daniel N. Rockmore
- E. Oran Brighman, FFT - Schnelle Fourier-Transformation, Einführung in die Nachrichtentechnik, Oldenbourg Verlag, 1982

45 Beispiele

45.1 Amplitudenspektrum

Wir erzeugen ein Test-Signal bestehend aus zwei überlagerten Sinusschwingungen und berechnen das Amplitudenspektrum mittels FFT.

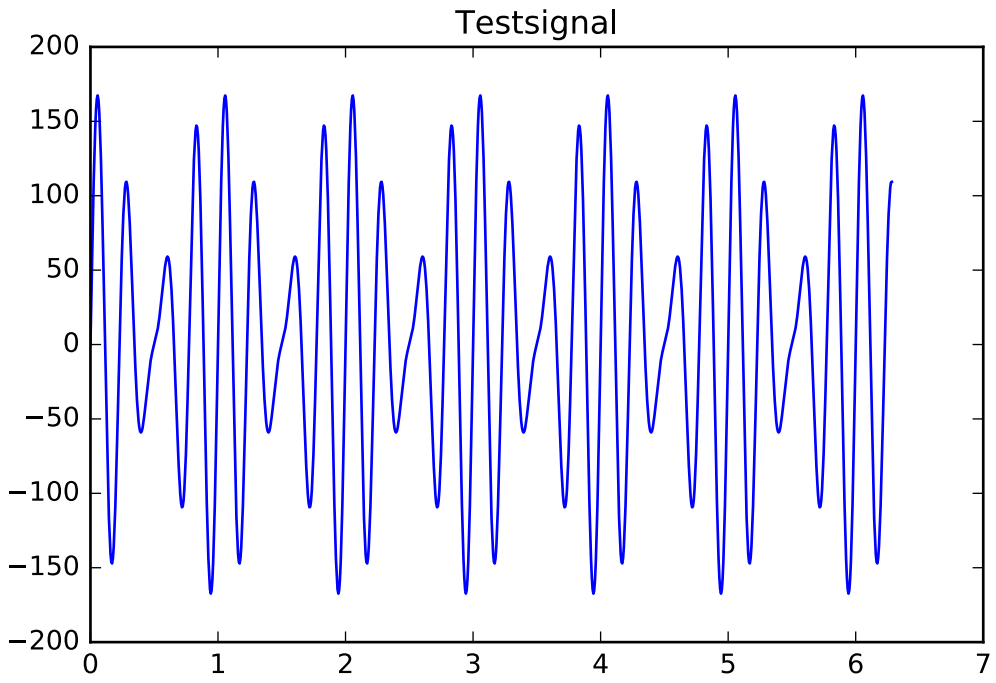
Eine "Frequenz-Verschmierung" ergibt sich dadurch, dass das Signal nicht exakt periodisch ist und Sprünge an den Rändern auftreten.

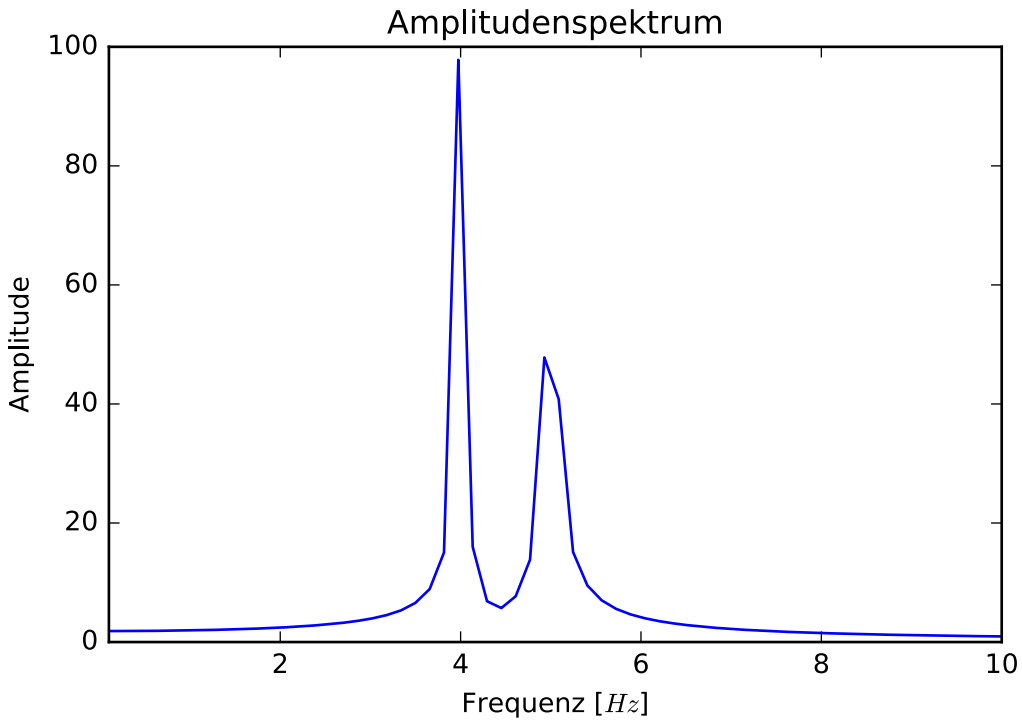
```
In [3]: %pylab inline
        %config InlineBackend.figure_format = 'svg'

        M=1000

        t = linspace(0, 2*pi, M, endpoint=True)
        f1 = 4.0 # Frequenz in Hz
        f2 = 5.0 # Frequenz in Hz

        A1 = 100.0 # Amplitude
        A2 = 70.0 # Amplitude

        y = A1 * sin(2*pi*f1*t)+  A2 * sin(2*pi*f2*t)# Signal

        plot(t,y)
        title('Testsignal')
        ######################################################
        dt = t[1] - t[0] # Abtast-Periode
        fa = 1.0/dt # Abtast-Frequenz

        Y = fft.fft(y)
        N = len(y)/2+1
```

```
X = linspace(0, fa/2, N, endpoint=True)

figure()
plot(X,abs(Y[:N])/N)
xlabel('Frequenz [$Hz$]')
ylabel('Amplitude')
xlim([0.1,10])
title('Amplitudenspektrum')
```

```
Populating the interactive namespace from numpy and matplotlib

/usr/local/lib/python3.4/dist-packages/ipykernel/__main__.py:27: DeprecationWarning: using a non-integer
```

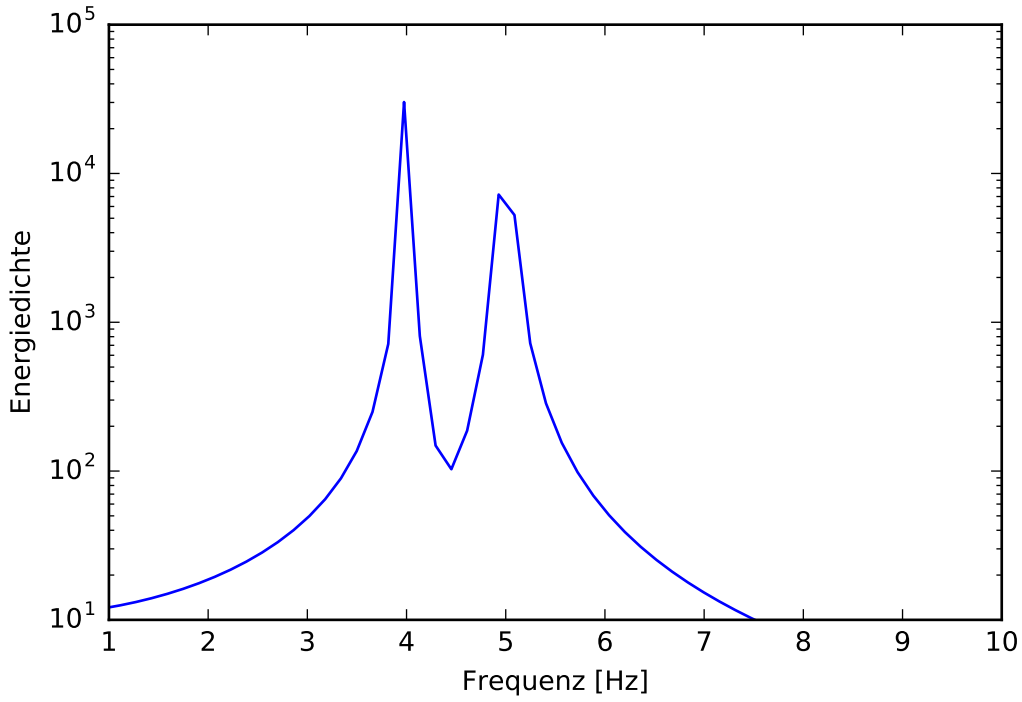
Out[3]: <matplotlib.text.Text at 0x7fea2a51a400>





45.2 Periodogramm / Leistunsspektrum

```
In [6]: from scipy.signal import periodogram
        f, Pxx = periodogram(y, fa)
        figure()
        semilogy(f, Pxx)
        xlabel('Frequenz [Hz]')
        ylabel('Energiedichte')

        ylim([1e1, 1e5])
        xlim([1, 10])

Out[6]: (1, 10)
```

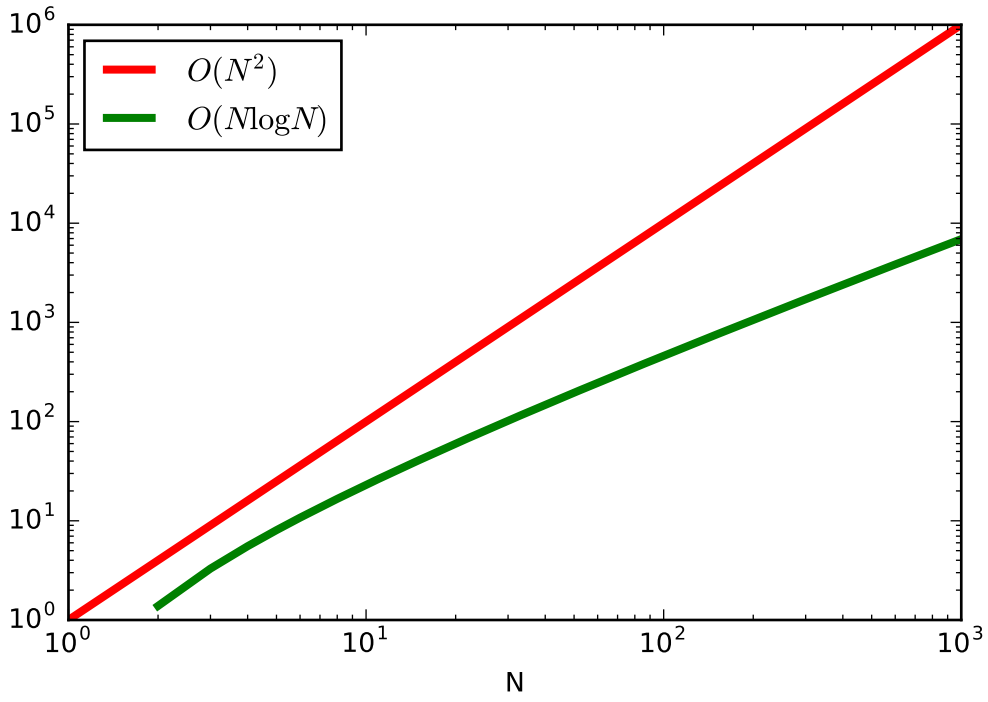


45.3 Komplexität

Der große Vorteil der FFT wird deutlich, wenn man die Anzahl der notwendigen Rechenschritte als Funktion der Vektorlänge N betrachtet.

```
In [51]: N=1000
         n=linspace(1,N,N)
         loglog(n,n**2,'r-',linewidth=3,label='$O(N^2)$')
         loglog(n,n*log(n),'g-',linewidth=3,label='$O(N \log N)$')
         xlabel('N')
         legend(loc=2)
```

```
Out[51]: <matplotlib.legend.Legend at 0x7fc1a078a0f0>
```

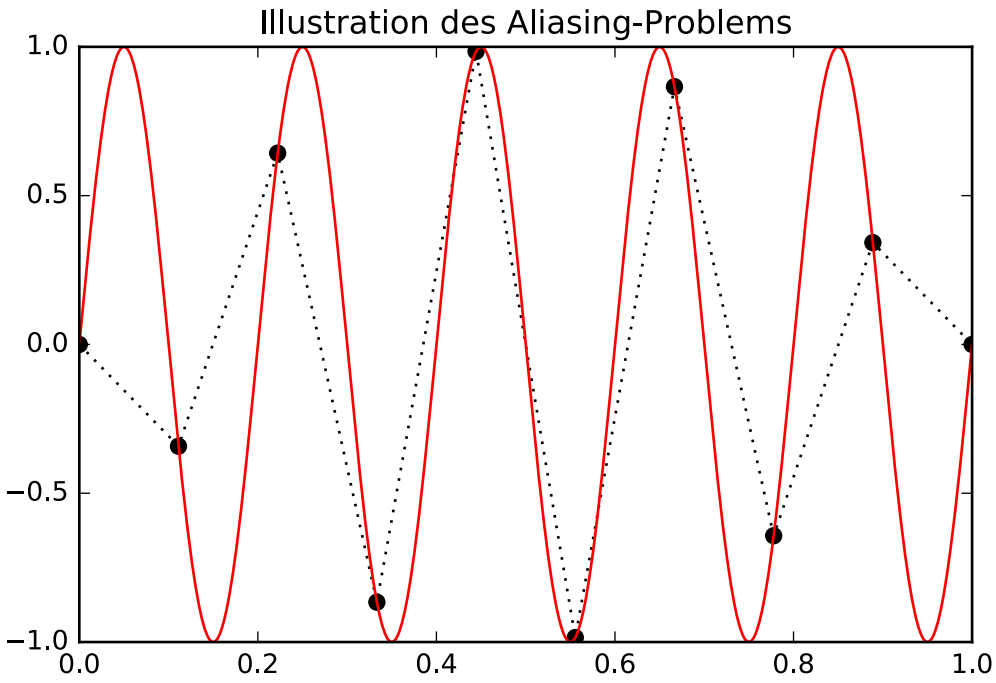


45.4 Alias-Problem

In [52]:
```
T=1.0
f=5.0/T
fN=f*2
print(fN)

N1=10 # Anzahl Abtast-Punkte
N2=1000
###################

t1=linspace(0,T,N1)
y1=sin(2*pi*f*t1)

t2=linspace(0,T,N2)
y2=sin(2*pi*f*t2)

plot(t1,y1,'ko:')
plot(t2,y2,'r-')
title('Illustration des Aliasing-Problems')
```

```
10.0
```

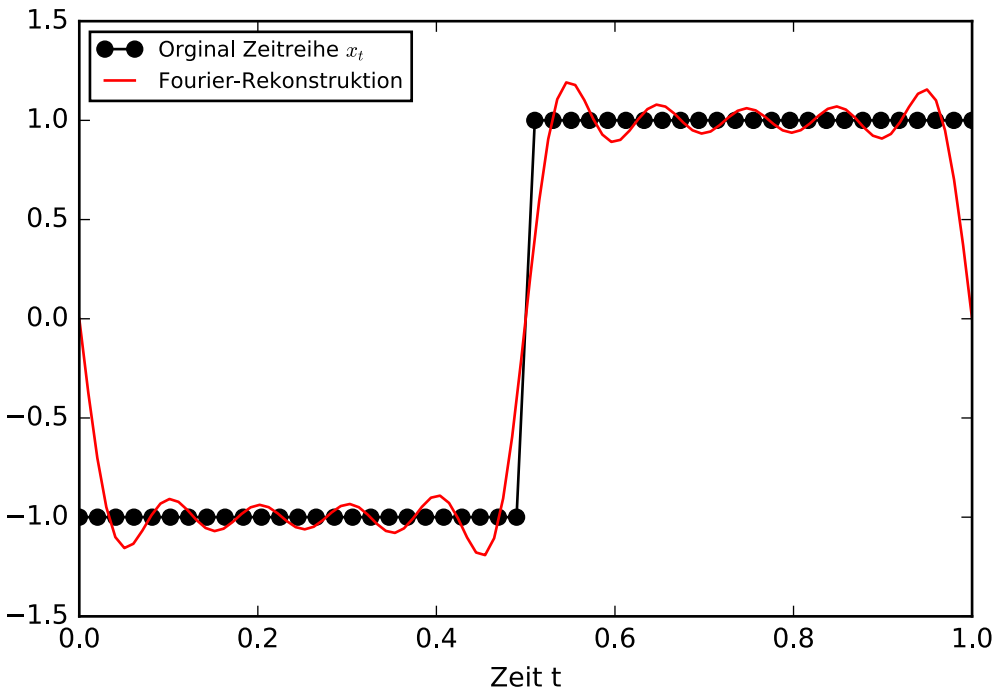
Out[52]:
```
<matplotlib.text.Text at 0x7fc1a08f5a90>
```

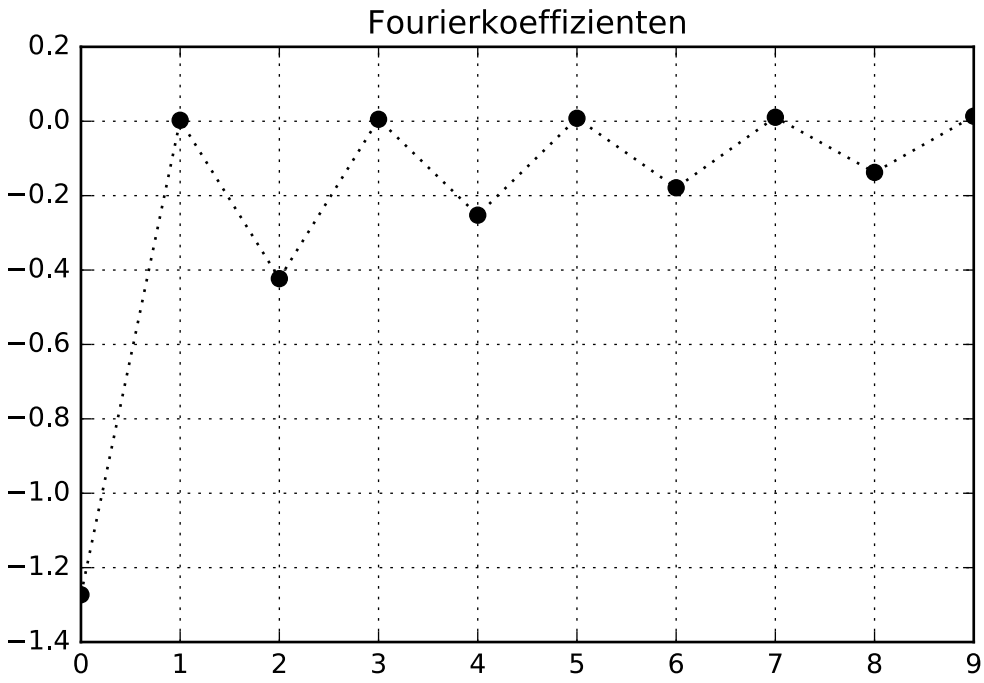


45.5 Berechnung der Fourier-Reihe zu Fuss

```
In [12]: def x(t):
             case=3 # Beispiele
             if case==1:
                 y=sin(t*2*pi) # Reiner Sinus
             if case==2:
                 y=sin(t*2*pi) + 0.5*sin(t*8*pi)
             if case==3: # Sprungfunktion
                 y=ones(t.size)-2
                 y[t>0.5]=1
             return y


         N=50
         T=1.0

         t=linspace(0,T,N)
         k=arange(1,N+1)
         f_k=k/T

         k_max=10 # Nutze Anzahl k_max Fourier-Koeffizienten

         ak=zeros(k_max)
         bk=zeros(k_max)

         # Berechnung der Fourier-Koeffizienten
         for i in range(k_max):
```

```
    ak[i]=2/T*sum(x(t)*cos(2*pi*f_k[i]*t))
    bk[i]=2/T*sum(x(t)*sin(2*pi*f_k[i]*t))

figure()
plot(t,x(t),'ko-',label='Orginal Zeitreihe $x_t$')

# Berechnung der Fourier-Reihe (Rekonstruktion)
for i in range(k_max):
    ak[i]=2/T*sum(x(t)*cos(2*pi*f_k[i]*t))/(N-1) # Integral durch Summation approximiert
    bk[i]=2/T*sum(x(t)*sin(2*pi*f_k[i]*t))/(N-1)

t2=linspace(0,T,100) # Neue (größere) Anzahl von Zeitschritten (=Interpolation)

N2=t2.size
x2=zeros(N2)
x2=x2+ak[0]/2
for i in range(k_max):
    x2=x2+ak[i]*cos(2*pi*f_k[i]*t2)+bk[i]*sin(2*pi*f_k[i]*t2)

plot(t2,x2,'r-',label='Fourier-Rekonstruktion')
xlabel('Zeit t')

legend(loc=2,fontsize=8)
figure()

plot(bk,'ko:')
title('Fourierkoeffizienten')
grid()
```

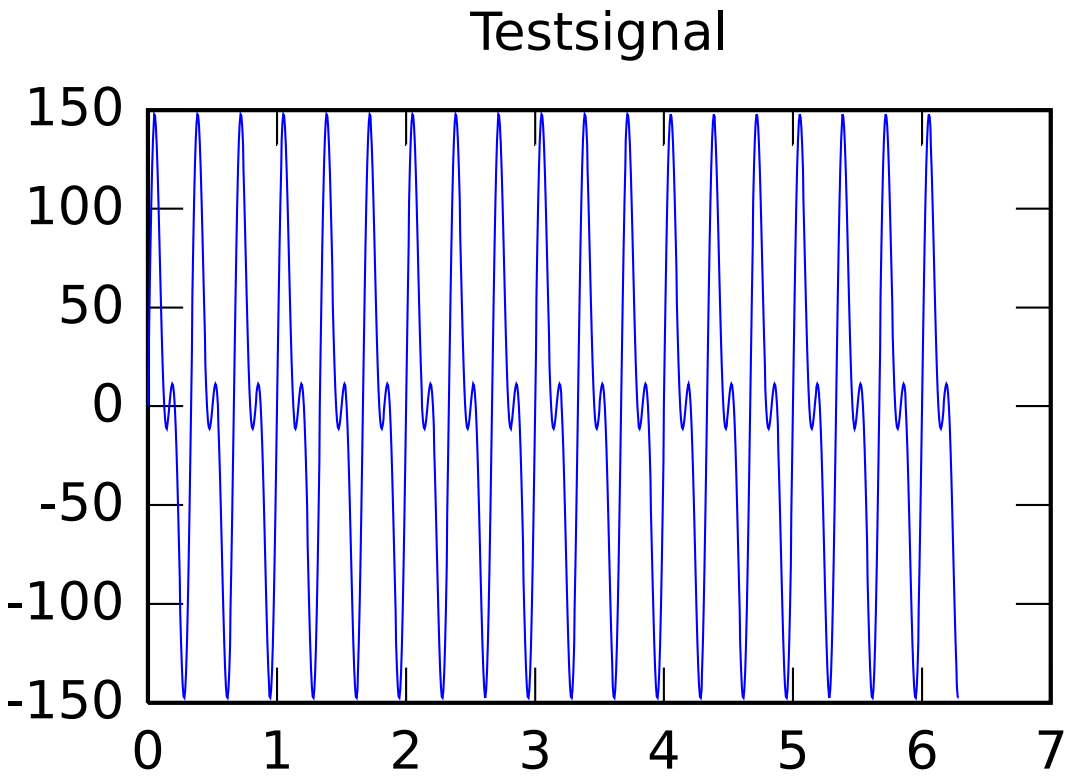


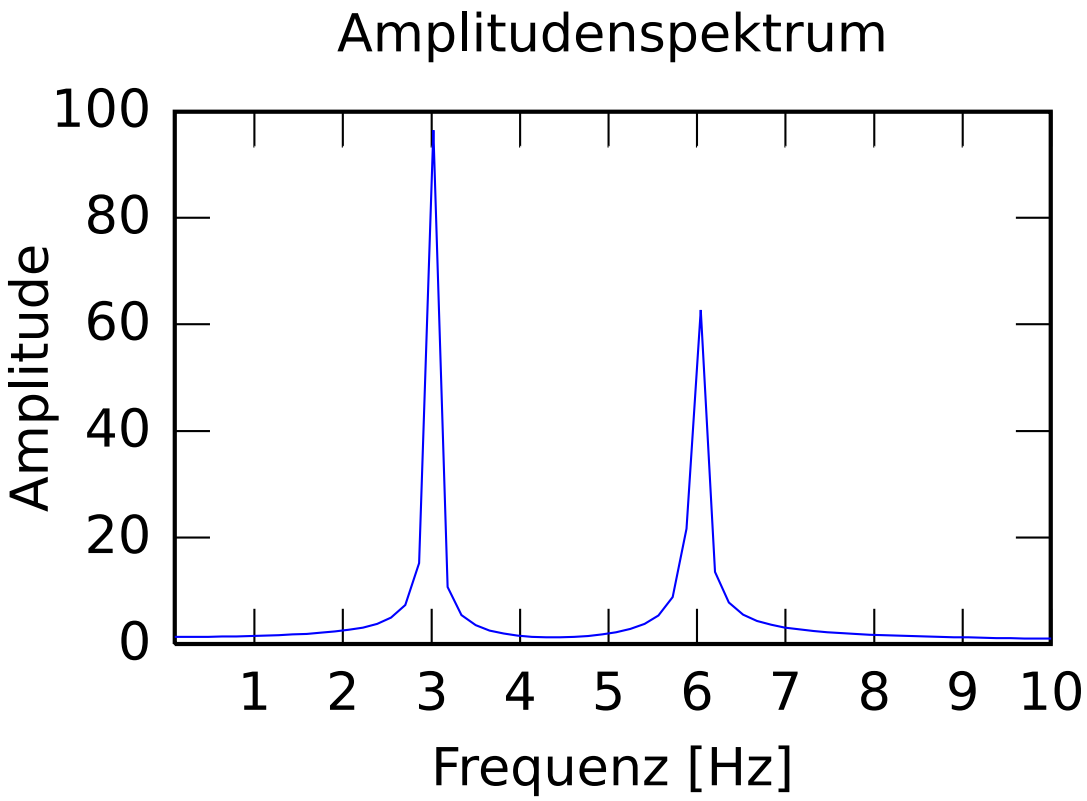


46 Matlab/Octave-Versionen

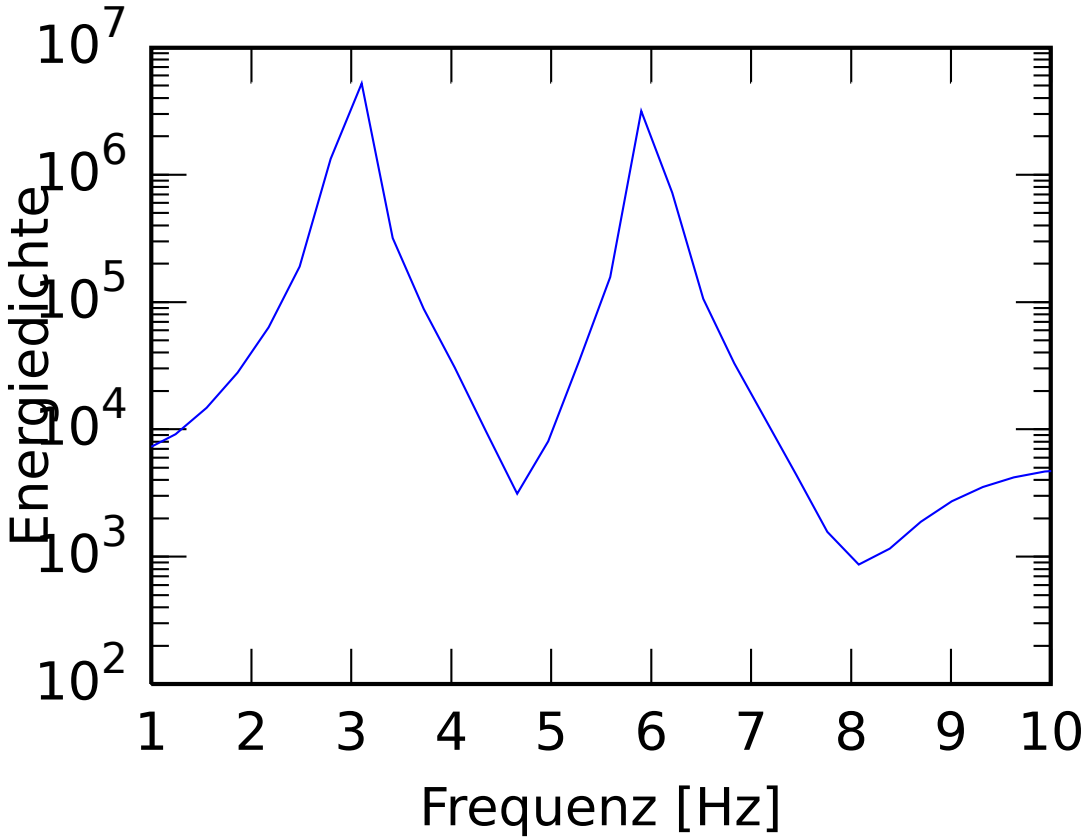
In [1]:
```
M=1000;

t = linspace(0, 2*pi, M);
f1 = 3.0; % Frequenz in Hz
f2 = 6.0; % Frequenz in Hz

A1 = 100.0; % Amplitude
A2 = 70.0; % Amplitude

y = A1 * sin(2*pi*f1.*t)+  A2 * sin(2*pi*f2.*t);% Signal

plot(t,y);
title('Testsignal');

dt = t(2) - t(1); % Abtast-Periode
fa = 1.0/dt; % Abtast-Frequenz

Y = fft(y);
N = size(y)(2)/2+1;

X = linspace(0, fa/2, N);

figure();
plot(X,abs(Y(1,1:N)/N));
xlabel('Frequenz [Hz]');
```

```
ylabel('Amplitude');
xlim([0.1,10]);
title('Amplitudenspektrum');
```






```
In [2]: [Pxx,f] = periodogram(y,1,512,fa);
        figure();
        semilogy(f, Pxx);
        xlabel('Frequenz [Hz]');
        ylabel('Energiedichte');

        ylim([1e2, 1e7]);
        xlim([1, 10]);
```



47 Faltung

Die Faltung zweier Funktion $x(t)$ und $h(t)$ ergibt eine neue Funktion $y(t)$ und wird definiert als das Integral

$$\int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau := x(t) * h(t) = (x * h)(t) = y(t)$$

oder im diskreten Fall durch

$$(x * h)(t) = \sum_k x(k)h(t-k)$$

Die Faltungsfunktion h wird auch Gewichtsfunktion genannt. Der Name wird deutlich bei der Interpretation als Filtermaske.

48 Faltungstheorem

(Herleitung gemäß Fouriertransformation für Fußgänger, Tilman Butz, Springer 2011)

Die Fouriertransformierte der Faltung ergibt

$$Y(f) = \int_{-\infty}^{\infty} y(t)e^{-\imath 2\pi ft}dt$$

Einsetzen von $y(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau$ liefert

$$= \int_{-\infty}^{\infty} \left[\int_{-\infty}^{\infty} x(\tau) h(t-\tau) d\tau \right] e^{-\imath 2\pi f t} dt$$

erweitert mit $e^{-\imath 2\pi f \tau} e^{\imath 2\pi f \tau} = 1$

$$= \int_{-\infty}^{\infty} x(\tau) e^{-\imath 2\pi f \tau} \left[\int_{-\infty}^{\infty} h(t-\tau) e^{-\imath 2\pi f (t-\tau)} dt \right] d\tau$$

mit Variablentransformation $t' = t - \tau$ erhalten wir

$$= \int_{-\infty}^{\infty} x(\tau) e^{-\imath 2\pi f \tau} H(f)$$

$$Y(f) = X(f) H(f)$$

Aus dem Faltungsintegral $*$ wird durch Fouriertransformation ein einfaches Produkt! Dies ist die wichtige Aussage des Faltungstheorems, welche die Implementierung von schnellen Faltungs-Filtern mittels FFT ermöglicht.

$$y(t) = x(t) * h(t)$$

$$Y(f) = X(f) H(f)$$

Die Korrelation $\int_{-\infty}^{\infty} x(t) y(t+\tau) d\tau$ ergibt sich analog aus dem Produkt mit der komplex konjugierten Fouriertransformation $X^*(f) Y(f)$.

49 Wiener-Chintschin-Theorem

Das Wiener-Chintschin-Theorem, besagt, dass die spektrale Leistungsdichte eines stationären Zufallsprozesses die Fourier-Transformierte der korrespondierenden Autokorrelationsfunktion ist. Im Falle diskreter Zeitserien hat das Wiener-Chintschin-Theorem die Form:

$$S(f) = \sum_k r(k) e^{-\imath 2\pi k f}$$

mit der Autokorrelationsfunktion $r(k)$ und der spektralen Leistungsdichte $S(f)$.

50 Parsevalsches Theorem

Die Energie eines Signals im Zeitbereich ist gleich seiner Energie im Frequenzbereich. Für diskrete Zeitserien gilt nach dem Parsevalschen Theorem

$$\sum_{t=0}^{N-1} |x(t)|^2 = \frac{1}{N} \sum_{f=0}^{N-1} |X(f)|^2$$

51 Beispiel: Mittelwertfilter implementiert als Faltung

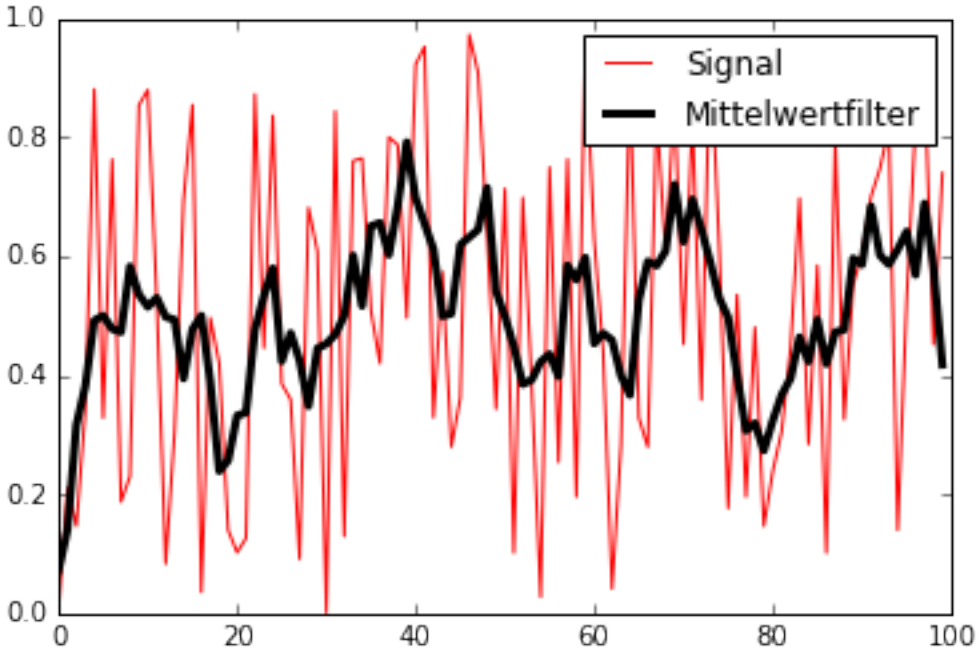
51.1 Python-Version

```
In [1]: %pylab inline

        x=rand(100)
        w=5
        h=ones(w)/w # Definiere Rechteckmaske mit Fläche=1
```

```
y=convolve(x,h,mode='same') # Faltungsoperation
plot(x,'r-',label='Signal')
plot(y,'k-',lw=3,label='Mittelwertfilter')
legend()
```

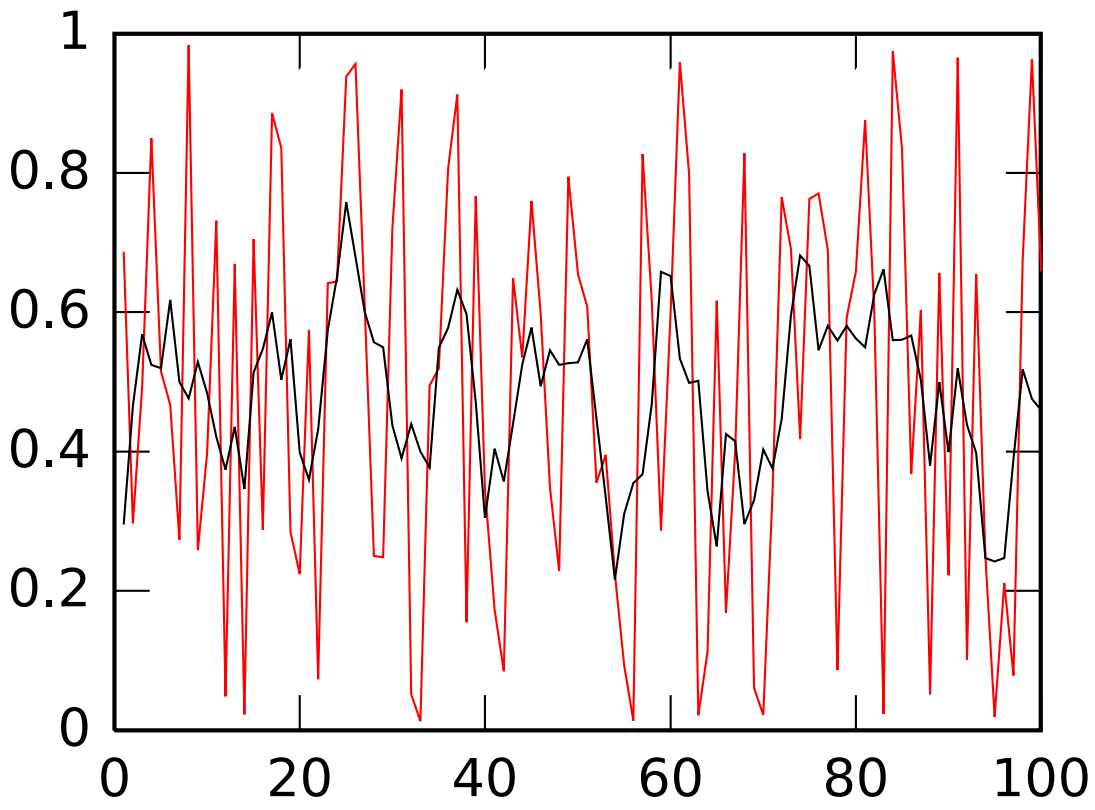
```
Populating the interactive namespace from numpy and matplotlib
```

```
Out[1]: <matplotlib.legend.Legend at 0x7f497b1e6dd8>
```



52 Beispiel Octave Version

```
In [18]: x=rand(100,1);
         w=5;
         h=ones(w,1)./w; % Definiere Rechteckmaske mit Fläche=1
         y=conv(x,h,'SAME'); % Faltungsoperation
         plot(x,'r-')
         hold();
         plot(y,'k-')
```



In [ ]:

53 Stochastische Prozesse

53.1 Stochastischer Prozess

Ein stochastischer Prozess ist eine Folge von Zufallsvariablen x_t.

53.2 White-Noise Prozess

Ein White-Noise-Prozess oder reiner Zufallsprozess ist eine identisch verteilte Zufallsvariable ϵ_t, die unabhängig von vorherigen Zeitschritten $t - \Delta t$ ist.

Beispiel Normalverteilung $\epsilon_t \sim N(\mu, \sigma)$

53.3 Autoregressive Prozesse

Beim autoregressiven Prozess (AR-Prozess) hängt die Zufallsvariable von vorherigen Zeitschritten ab. Im Fall der ersten Ordnung (AR1-Prozess) hängt der Zeitschritt t nur vom vorherigen Zeitschritt $t - 1$ ab:

$$x_t = \alpha x_{t-1} + \epsilon_t$$

Der Koeffizient α wird Feedback-Parameter genannt.

53.4 Random-Walk Prozess

Der Spezialfall des AR1-Prozesses mit $\alpha = 1$ wird Random Walk oder auch Zufallsbewegung genannt.

$$x_t = x_{t-1} + \epsilon_t$$

53.5 Korrelation und Autokorrelation

Der Korrelationskoeffizient r zwischen N Beobachtungspaaren zweier Variabler x und y ist gegeben als

$$r(x,y) = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}}$$

Die Autokorrelation $r(x,l)$ ist die Korrelation der Zeitserie x_t mit der um die Distanz (Lag) l verschobenen gleichen Zeitreihe x_{t+l}. Besondere Bedeutung hat der Autokorrelationskoeffizient für $l = 1$.

53.6 Filter

53.6.1 Lineare Filter

Ein linearer Filter L transformiert eine Zeitreihe x_t in einer andere Zeitreihe y_t mittels der Berechnung

$$y_t = \sum_i a_i x_{t-i}$$

$$y_t = Lx_t$$

Die Folge der Koeffizienten (a_i) des linearen Filteroperators L wird auch als Impulsantwortfunktion bezeichnet.

53.6.2 Gleitender Mittelwert

Der gleitende Mittelwert oder auch Moving Average (MA) berechnet sich aus

$$y_t = \frac{1}{3}(x_{t-1} + x_t + x_{t+1})$$

Die sogenannte Fensterlänge (Window Size) beträgt in diesem Fall $W = 3$ (Anzahl der Koeffizienten).

53.6.3 Kausale Filter

Wenn das Filter nur von seinen gegenwärtigen und vergangenen Eingangswerten abhängt, wird es kausales Filter genannt.

Das Mittelwertfilter $y_t = \frac{1}{3}(x_{t-1} + x_t + x_{t+1})$ ist nicht-kausal, da der gegenwärtige Ausgangswert y_t vom zukünftigen Eingangswert x_{t+1} abhängt.

53.7 Stationarität

Die Funktionen Mittelwert $\mu(t) = E[x_t]$, Varianz $\sigma^2(t) = Var[x_t]$ und Kovarianz $\gamma(s,t) = Cov[x_s, x_t]$ seien jeweils über einen endlichen Fensterbereich W zu berechnen.

Die stochastische Zeitreihe x_t wird mittelwert- bzw. varianzstationär genannt, wenn der Mittelwert $\mu(t)$ bzw. die Varianz $\sigma^2(t)$ zeitlich konstant ist. Die Zeitreihe heißt kovarianzstationär, wenn $\gamma(s,t)$ nur von $s - t$ abhängt. Ist eine Zeitreihe mittelwert-, varianz und kovarianzstationär, so wird sie schwach stationär genannt.

54 AR1-Spektrum

Das korrespondierende Spektrum $G(\omega)$ (spektrale Dichte) eines Markov-Prozesses erster Ordnung ist gegeben als (Jenkins und Watts, 1968)

$$G(\omega) = \frac{F}{1 + \alpha^2 - 2\alpha \cos(\omega \Delta)}$$

mit der Frequenz ω und der (konstanten) spektralen Energiedichte F des Rauschantriebs.

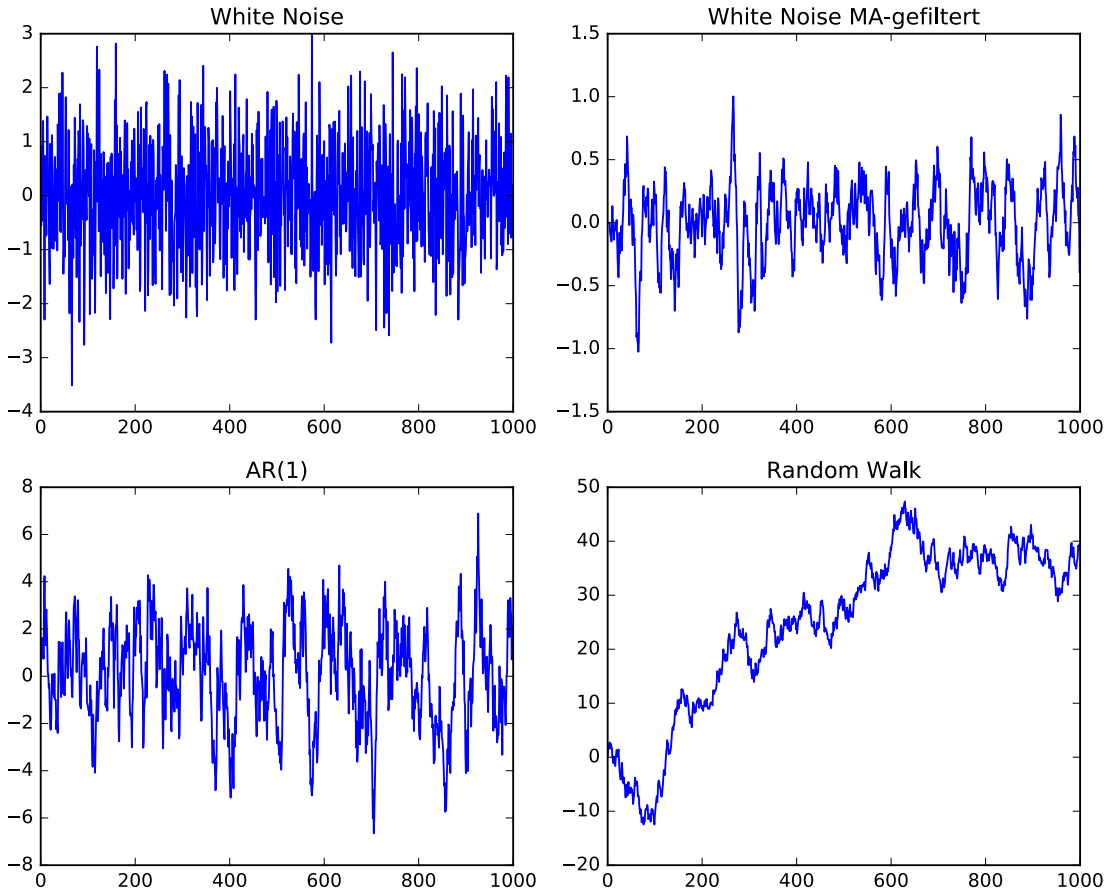
55 Beispiele

55.1 Stochastische Prozesse

In [24]:
```
%pylab inline
%config InlineBackend.figure_format = 'svg'

def AR1(N,a,b):
    Y=zeros(N)
    for i in range(1,N):
        Y[i]=Y[i-1]*a+b*randn(1)
    return Y

def MA_filter(Y,v):
    Y_in=Y
    Y=Y.copy()
    for i in range(1,N-1):
        Y[i]=1.0/(2*v+1)*(sum(Y_in[i-v:i+v+1]))
    return Y


N=1000
Y_white_noise=AR1(N,0,1)
Y_L=MA_filter(Y_white_noise,5)
Y_AR1=AR1(N,0.9,1)
Y_random_walk=AR1(N,1.0,1)

#####################################
figure(figsize=(10,8))
subplot(2,2,1)
plot(Y_white_noise)
title('White Noise')
subplot(2,2,2)
plot(Y_L)
title('White Noise MA-gefiltert')
subplot(2,2,3)
plot(Y_AR1)
title('AR(1)')
subplot(2,2,4)
plot(Y_random_walk)
title('Random Walk')
```

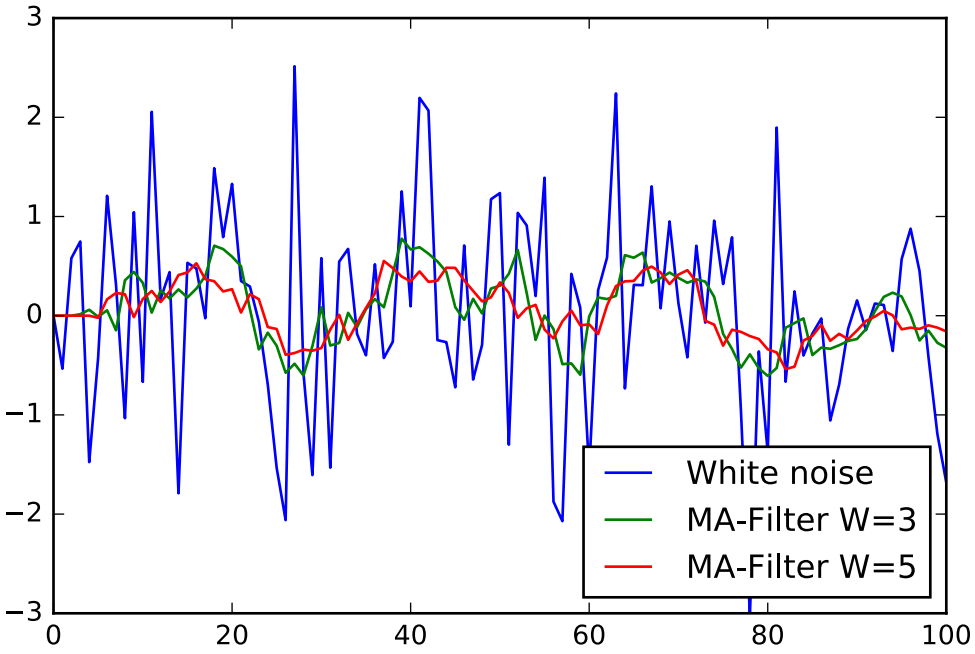
```
Populating the interactive namespace from numpy and matplotlib
```

Out[24]:
```
<matplotlib.text.Text at 0x7f47a7246b38>
```



55.2 MA-Filter

```
In [25]: N=1000
         Y_white_noise=AR1(N,0,1)
         Y_L_W3=MA_filter(Y_white_noise,3)
         Y_L_W5=MA_filter(Y_white_noise,5)

         plot(Y_white_noise,label='White noise')
         plot(Y_L_W3,label='MA-Filter W=3')
         plot(Y_L_W5,label='MA-Filter W=5')
         legend(loc=4)
         axis([0,100,-3,3])

Out[25]: [0, 100, -3, 3]
```



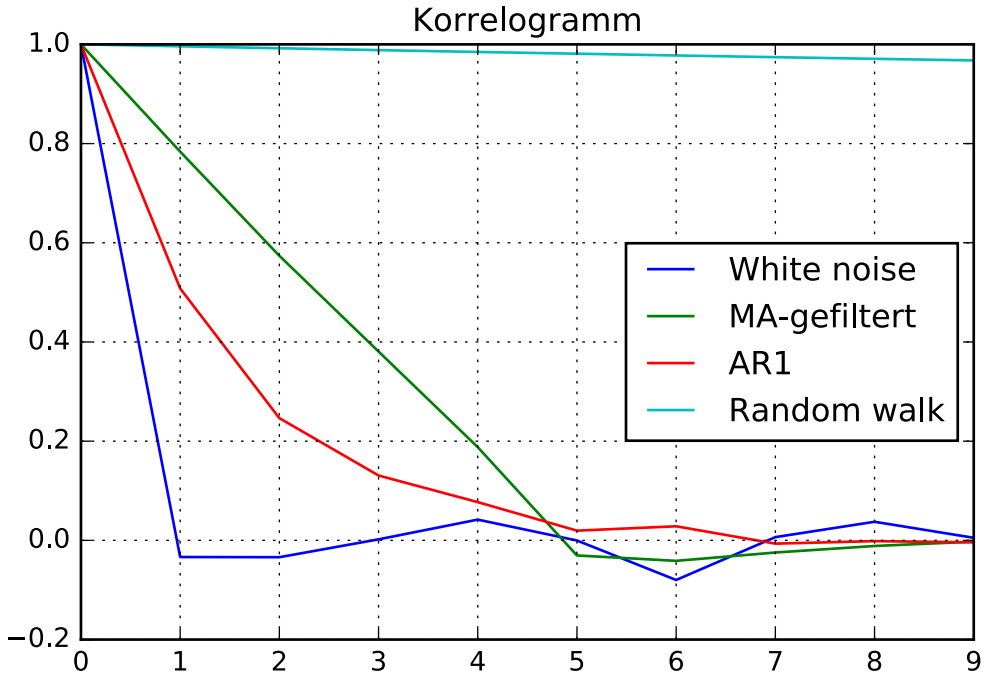
55.3 Autokorrelation

```
In [26]: def korrelation(x,y):
             mx=mean(x)
             my=mean(y)
             return sum( (x-mx)*(y-my) )/sqrt(sum( (x-mx)**2)* sum((y-my)**2) )

         print(korrelation(Y_white_noise,Y_L_W3))
         print(korrelation(Y_white_noise,Y_white_noise))
         print(korrelation(Y_L,Y_AR1))
```

```
0.347522360468
1.0
-0.019946912798
```

```
In [6]: def autokorrelation(x,l):
            r=ones(l)
            for i in range(1,l):
                x1=x[i:]
                x2=x[:-i]
                r[i]=korrelation(x1,x2)
            return r
        L=10
        plot(autokorrelation(Y_white_noise,L),label='White noise')
        plot(autokorrelation(Y_L_W5,L),label='MA-gefiltert')
        plot(autokorrelation(Y_AR1,L),label='AR1')
        plot(autokorrelation(Y_random_walk,L),label='Random walk')
        legend(loc='center right')
        grid()
        title('Korrelogramm')
```

Out[6]: <matplotlib.text.Text at 0x7f47ad9b3630>



55.4 Vorsicht bei Autokorrelationen!

Weisen die Daten Autokorrelationen auf, sind sie nicht mehr unabhängig! Die Anzahl der Freiheitsgrade reduziert sich. Dies muss bei der Hypothesentests berücksichtigt werden.

Siehe Matlab-Skript corr2.m von Ian Eisenmann http://eisenman.ucsd.edu/

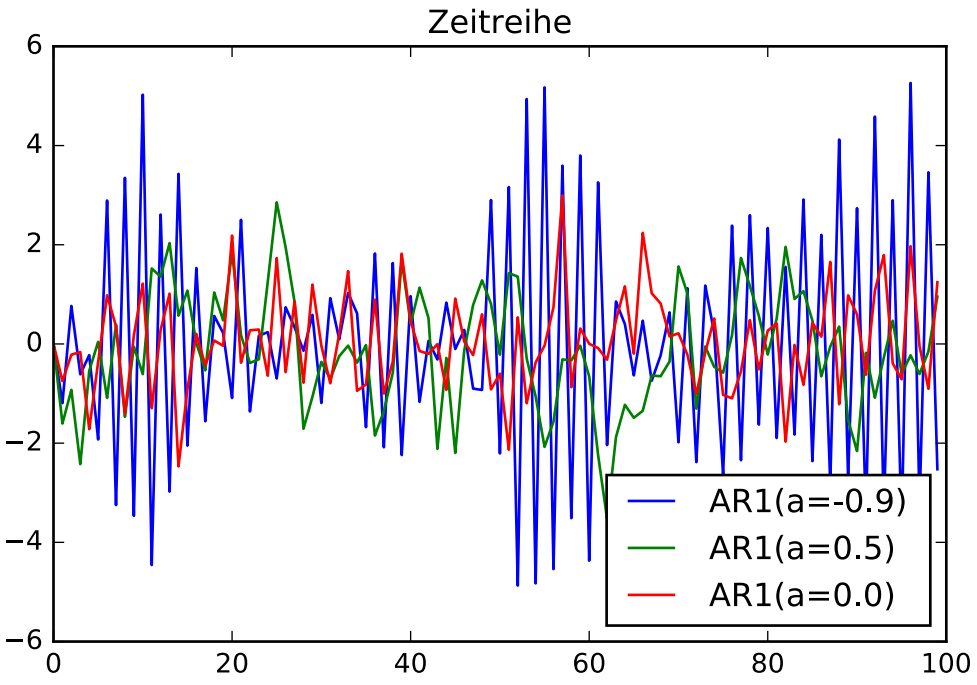
```
% Linear correlation (RHO) between X and Y in which the p-value (PVAL)
% accounts for autocorrelation in both records by using AR(1) processes as
% the null hypothesis rather than white noise as in the standard corr()
% function. Uses common method for estimating effective degrees of freedom
% in autocorrelated data (Bartlett, J. Roy. Stat. Soc. 98, 536-543, 1935;
% Mitchell et al., Climatic Change, WMO Technical Note 79, 1966; Bretherton
% et al., J. Climate 12, 1990-2009, 1999).
```

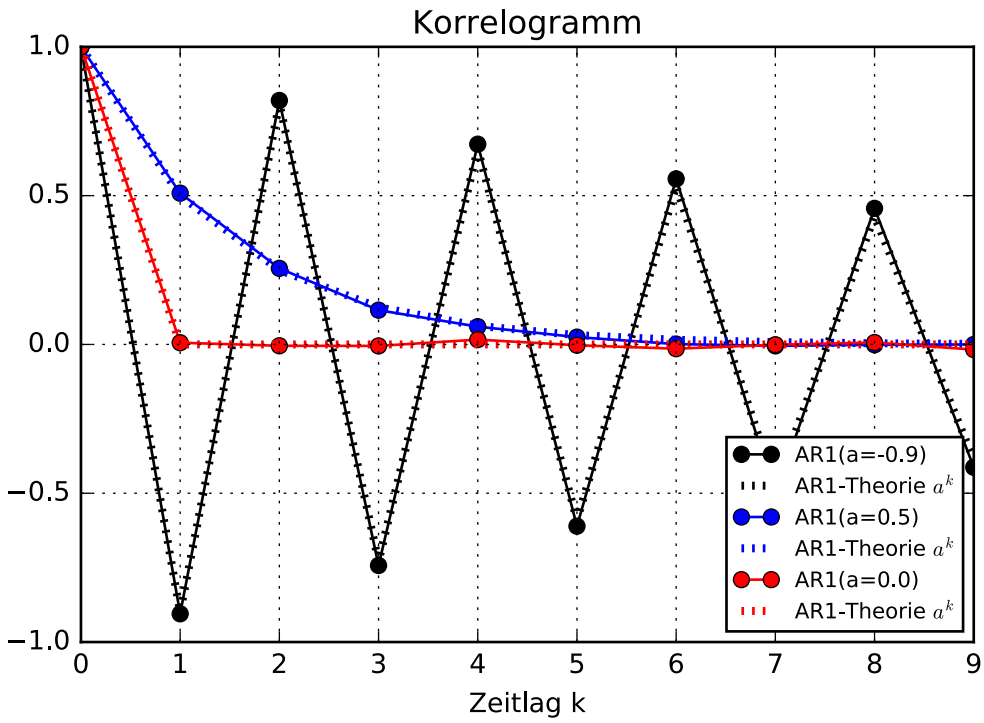
56 Beispiel AR1-Spektrum

In [27]:
```
%pylab inline
%config InlineBackend.figure_format = 'svg'


close('all')

figure(1)
figure(2)
N=10000
L=10
```

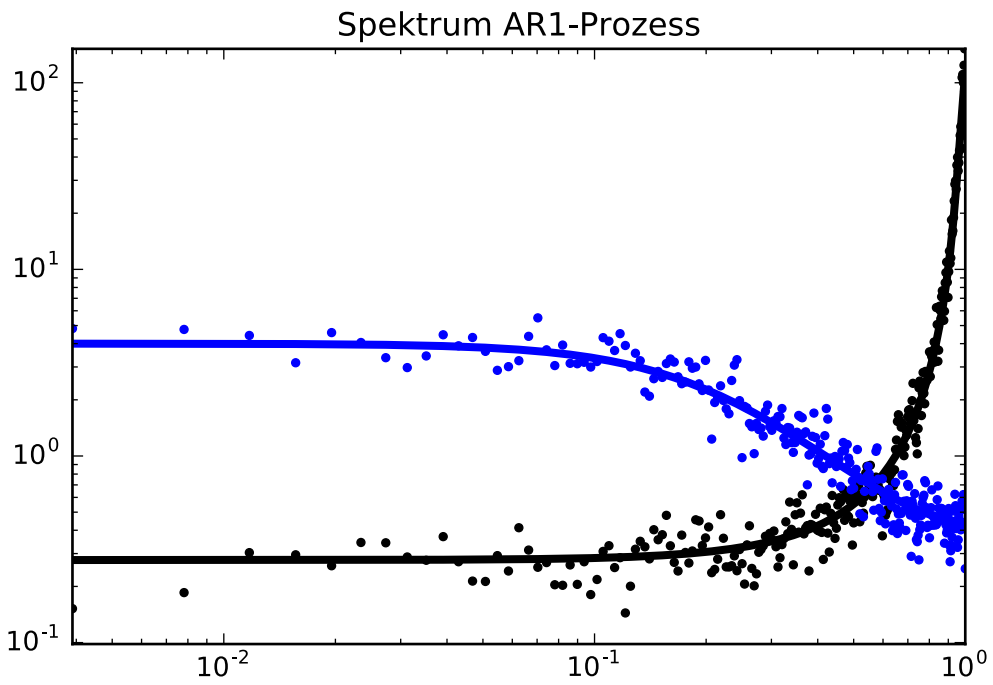
```
k=arange(0,L)
b=1.0
an=[-0.9,0.5,0.0]
Na=len(an)
Ya=zeros((Na,N))

c='kbrcy'
for i,a in enumerate(an):
    Y=AR1(N,a,b)
    Ya[i]=Y
    figure(1)
    plot(Y[0:100],label='AR1(a='+str(a)+') ')
    figure(2)
    plot(k,autokorrelation(Y,L),'o-',label='AR1(a='+str(a)+') ',color=c[i])
    plot(k,a**k,':',lw=3,label='AR1-Theorie $a^k$',color=c[i])
figure(1)
legend(loc='lower right')
title('Zeitreihe')
figure(2)
legend(loc='lower right',fontsize=8)
title('Korrelogramm')
xlabel('Zeitlag k')
grid()
```

```
Populating the interactive namespace from numpy and matplotlib
```






```
In [29]: def AR1_Spektrum(a,F,f):
             G=F/(1+a**2-2*a*cos(f*pi))
             return G

         figure()
         Pxx1,freqs=psd(Ya[0],NFFT=512)
         Pxx2,freqs=psd(Ya[1],NFFT=512)

         close()

         G1=AR1_Spektrum(an[0],b**2,freqs)
         loglog(freqs,G1,'-',linewidth=3,color=c[0])
         loglog(freqs,Pxx1,'.',color=c[0])

         G2=AR1_Spektrum(an[1],b**2,freqs)
         loglog(freqs,G2,'-',linewidth=3,color=c[1])
         loglog(freqs,Pxx2,'.',color=c[1])


         title('Spektrum AR1-Prozess')
         axis('tight')

Out[29]: (0.00390625, 1.0, 0.098249035829564429, 151.96640043431412)
```



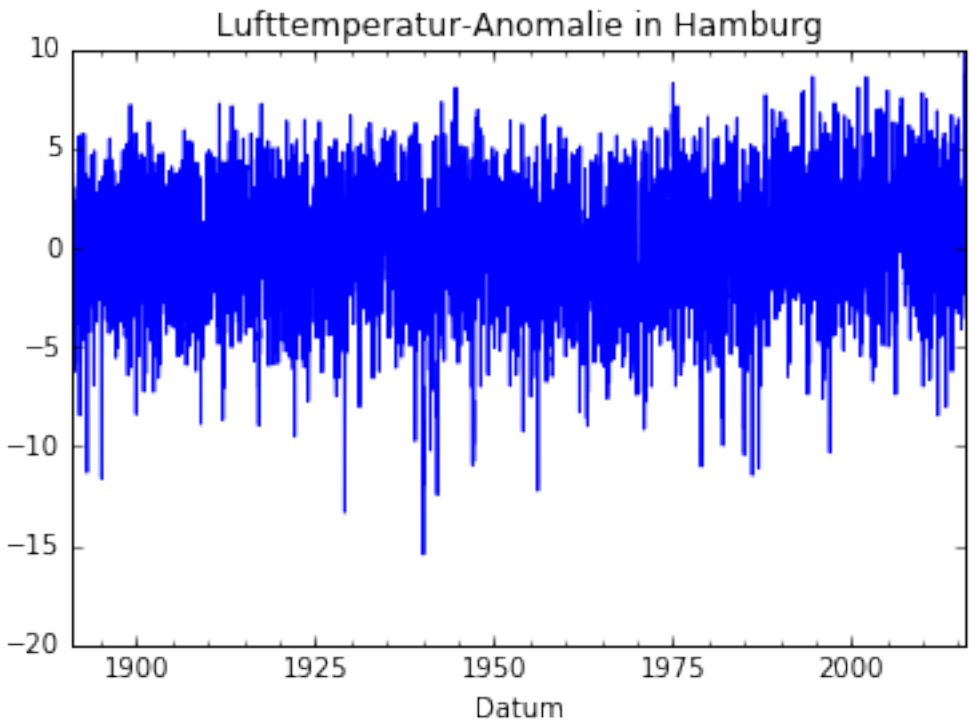
57 Beispiel: Spektrum der Lufttemperatur-Anomalie

Wir verwenden meteorologische Daten vom Deutschen Wetterdienst, aus welchen wir zuvor Wochenmittelwerte und Anomalien berechnet haben, siehe Beispiel aus Stunde 3

Fragestellung: Wir gut können die Anomalien durch einen AR1-Prozess repräsentiert werden?

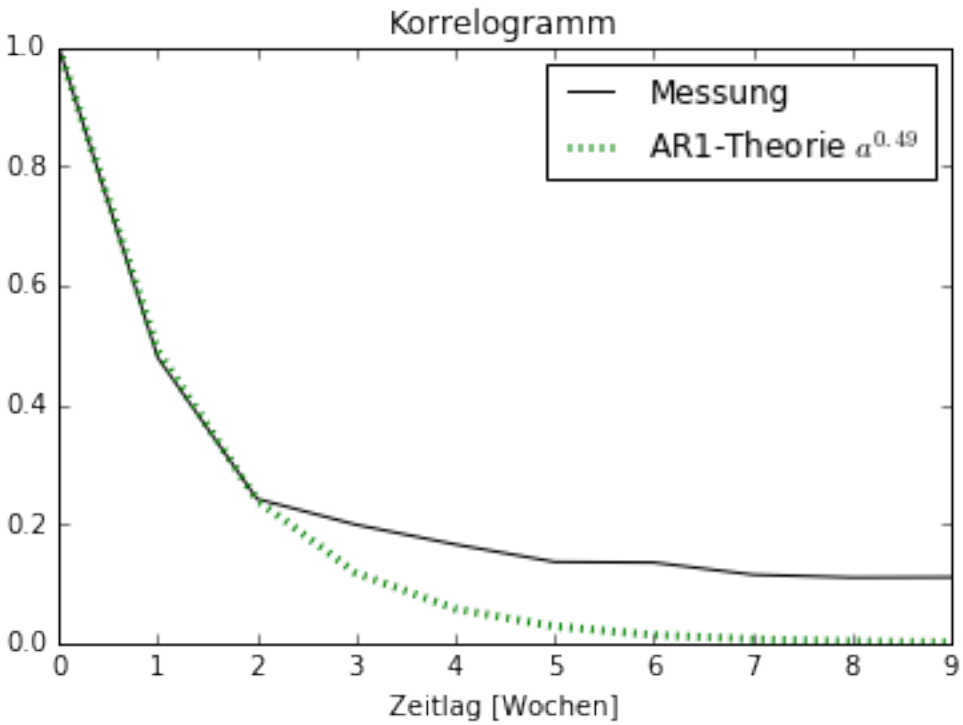
57.1 Python-Version

```
In [1]: %pylab inline
        import pandas as pd
        DF=pd.read_csv('Anomalien.csv',parse_dates=[0])
        DF.head()

Populating the interactive namespace from numpy and matplotlib

Out[1]:        Datum  Wochenmittel  Jahresgang   Anomalie
        0 1891-01-01    -10.700000    0.881145 -11.581145
        1 1891-01-08     -6.485714    0.648434  -7.134148
        2 1891-01-15     -4.957143    0.543115  -5.500258
        3 1891-01-22     -6.042857    0.566787  -6.609644
        4 1891-01-29      1.785714    0.719091   1.066623

In [2]: TA=pd.Series(DF['Anomalie'].data,index=DF['Datum'])
        TA.plot()
        title('Lufttemperatur-Anomalie in Hamburg')

Out[2]: <matplotlib.text.Text at 0x7f4b70f5ac18>
```



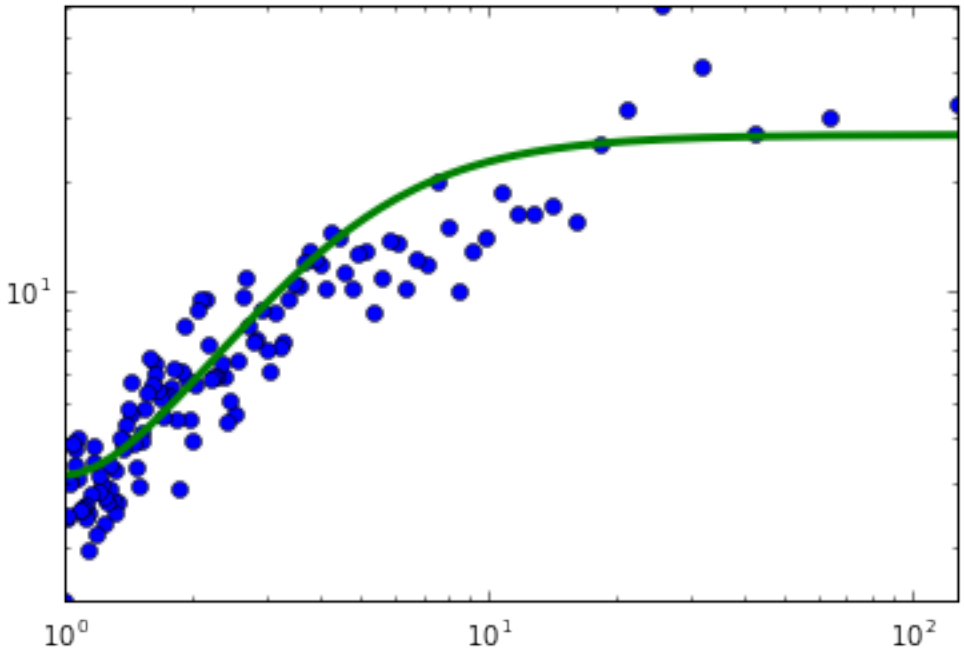
57.2 Korrelogramm

```
In [3]: def korrelation(x,y):
            mx=mean(x)
            my=mean(y)
            return sum( (x-mx)*(y-my) )/sqrt(sum( (x-mx)**2)* sum((y-my)**2) )
        def autokorrelation(x,l):
            r=ones(l)
            for i in range(1,l):
                x1=x[i:]
                x2=x[:-i]
                r[i]=korrelation(x1,x2)
            return r
        L=10
        k=arange(0,L)
        AK=autokorrelation(TA.data,L)
        plot(k,AK,'k-',label='Messung')
        a=0.49
        plot(k,a**k,'g:',lw=3,label='AR1-Theorie $a^{'+str(a)+'}$')
        xlabel('Zeitlag [Wochen]')
        title('Korrelogramm')
        legend()

Out[3]: <matplotlib.legend.Legend at 0x7f4b71536780>
```



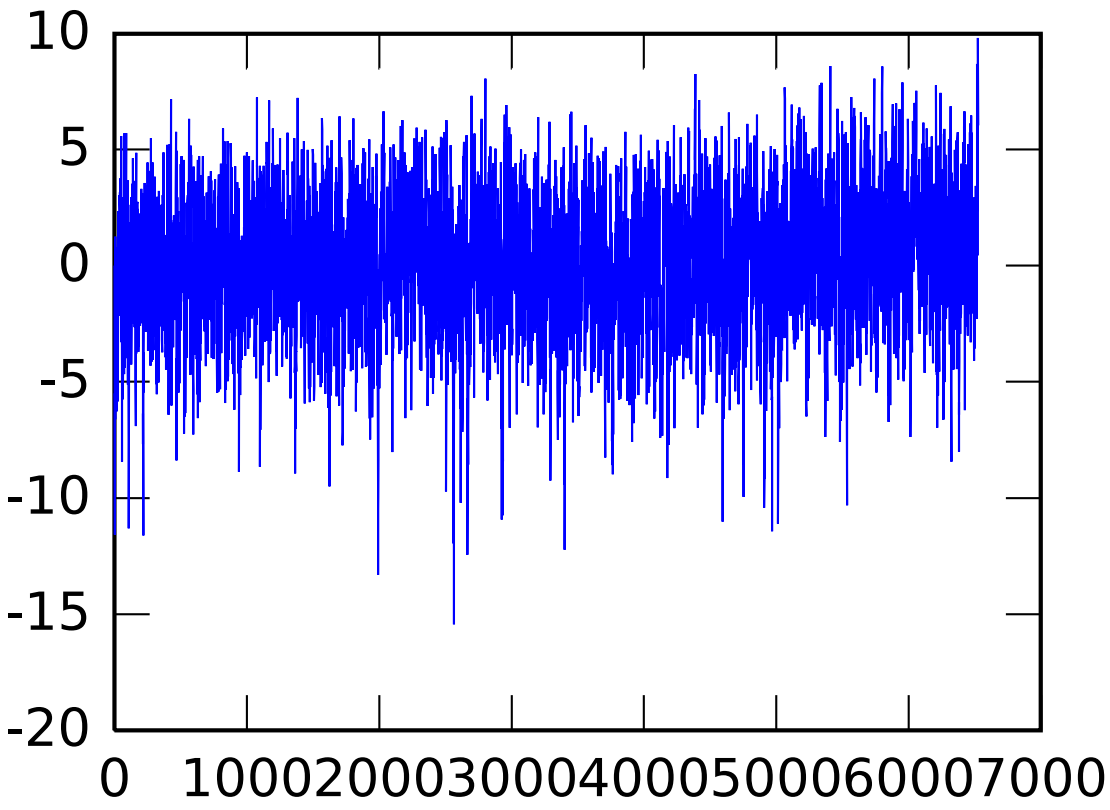
57.3 Spektrum

```
In [5]: def AR1_Spektrum(a,F,f):
            G=F/(1+a**2-2*a*cos(f*pi))
            return G

        figure()
        Pxx1,freqs=psd(TA.data)
        close()
        G1=AR1_Spektrum(0.49,7,freqs)
        T=1/freqs
        #loglog(freqs,Pxx1,'o')
        #loglog(freqs,G1,'-',linewidth=3)
        loglog(T,Pxx1,'o')
        loglog(T,G1,'-',linewidth=3)

        axis('tight')

Out[5]: (1.0, 128.0, 1.4151486616128028, 60.782831024320878)
```

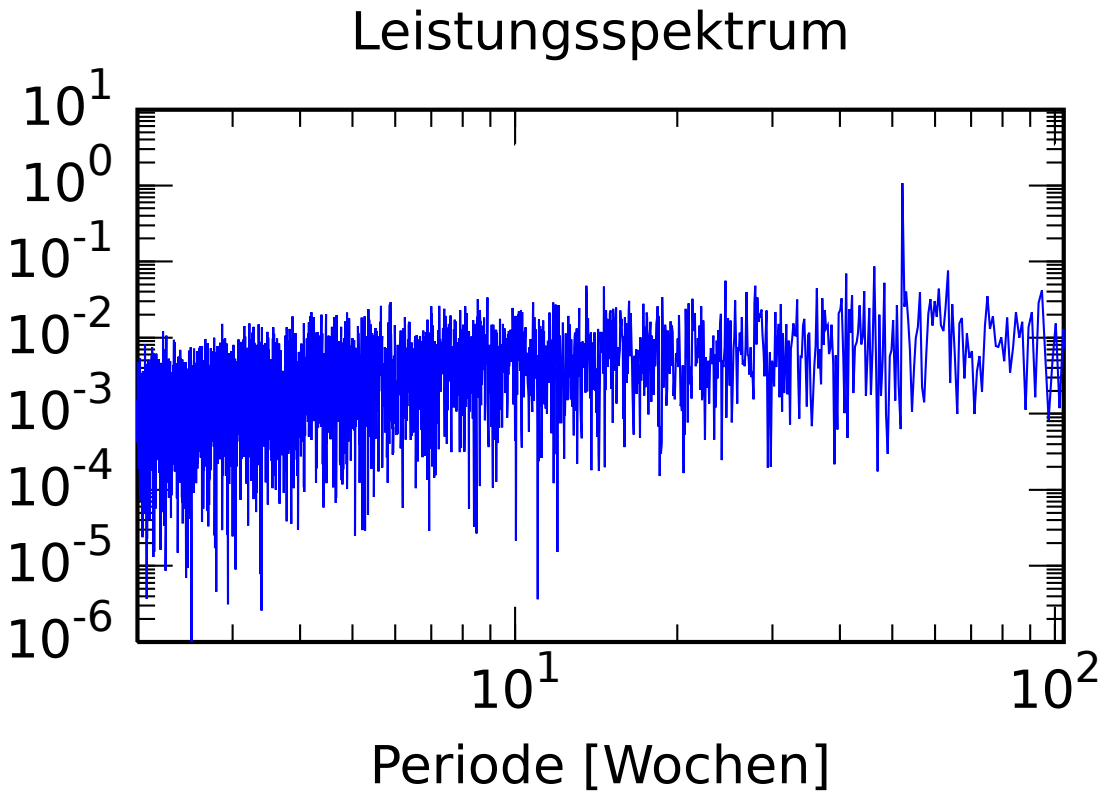


57.4 Octave Beispiele

```
In [2]: TAB=csvread('Anomalien.csv');
        y=transpose(TAB(:,4));
        plot(y)
```



In [3]:
```
fa = 1.0; % Abtast-Frequenz

Y = fft(y);
N = size(y)(2)/2+1;
X = linspace(0, fa/2, N);

T=1./X;
figure();
Amp=abs(Y(1,1:N)/N);
loglog(T,Amp.^2);

xlabel('Periode [Wochen]');
ylim([1e-6,10]);
xlim([2,104]);
title('Leistungsspektrum')
```



In [ ]:

58 Filter

Digitale Filter sind wichtige Werkzeuge für die Signalverarbeitung bzw. Zeitserienanalyse. Mit Filtern lässt sich Rauschen unterdrücken und das zu untersuchende Signal hervorheben. Desweiteren werden Filter für Interpolation und Vorhersagen verwendet.

Man unterscheided zwischen Tiefpass-, Hochpass-, Bandpass- und Bandblockfiltern.

- **Tiefpassfilter** werden zur Rauschunterdrückung und Glättung eines Signals verwendet. Der Rechen-Prozess entspricht einer Summation bzw. Integration.
- **Hochpassfilter** verstärken das Rauschen bzw. Änderungen im Signal. Hochpassfilter sind z.B. geeignet um Kanten zu detektieren (2D-Anwendung: Bildverarbeitung). Die Rechenoperation entspricht einer Differenz bzw. Ableitung.
- **Bandpass bzw. Bandblockfilter** heben bestimmte Frequenzbänder hervor bzw. blockieren sie. Die Konstruktion erfolgt aus einer Kombination von Hoch- und Tiefpassfiltern.

58.1 Literatur

1. Digitale Signalverarbeitung : Filterung und Spektralanalyse mit MATLAB-Übungen von Karl-Dirk Kammeyer, Kristian Kroschel http://dx.doi.org/10.1007/978-3-663-09805-8
2. https://de.wikipedia.org/wiki/Filter_mit_endlicher_Impulsantwort

58.2 Definition und Eigenschaften der Delta-Funktion

Die Delta-Funktion hat die Eigenschaft

$$\int x(t)\delta(t-t_0)dt = x(t_0)$$

Die Delta-Funktion wird mittels Fourier-Transformation definiert

$$x(t) = \int e^{\imath 2\pi f t} df = \delta(t)$$

Die Fouriertransformierte der Delta-Funktion ist daher gleich eins

$$X(f) = \int \delta(t) e^{-\imath 2\pi f t} dt = 1$$

Die Integralgrenzen seien von $-\infty$ bis ∞.

59 Filter mit endlicher Impulsantwort

Ein Finite Impule Responsing (FIR) Filter ist ein diskretes, nichtrekursives Filter.

$$x(t) \rightarrow \boxed{\text{H(f)}} \rightarrow y(t)$$

Im Ortsraum wird ein Filter durch eine Faltungsmaske $h(\tau)$ beschrieben, welche die Zeitserie $x(t)$ in einer Zeitserie $y(t)$ überführt

$$y(t) = \int h(\tau) x(t-\tau) d\tau$$

Für diskrete Werte ergibt sich

$$y_t = \sum_i h_i x_{t-i}$$

Ein gleitender Mittelwert mit Fensterlänge $l = 3$ wird realisiert über die Faltungsmaske

$$h = [h_{-1}, h_0, h_1] = \frac{1}{3}[1,1,1]$$

Der Filter $h(\tau)$ wird auch Impulsantwort genannt, weil h die "Antwort" auf das Impuls-Signal $x(t) = \delta(t-\tau)$ ist.

Im Frequenzraum gilt

$$Y(f) = X(f) \cdot H(f)$$

mit der Übertragungsfunktion

$$H(f) = \int e^{-\imath 2\pi\tau} h(\tau) d\tau$$

Die Übertragungsfunktion wird auch Transferfunktion genannt.

Kennt man $x(t)$ und $y(t)$ kann die Übertragunsfunktion aus dem Spektrum bestimmt werden

$$H(f) = \frac{Y(f)}{X(f)}$$

59.1 Beispiel Übertragungsfunktion

Wir untersuchen Filter anhand der Übertragung des Signals “Weisses Rauschen”. Das Leistungsspektrum des Eingangssignals ist konstant. Die Impulsantwort lässt sich daher direkt im Spektrum erkennen.

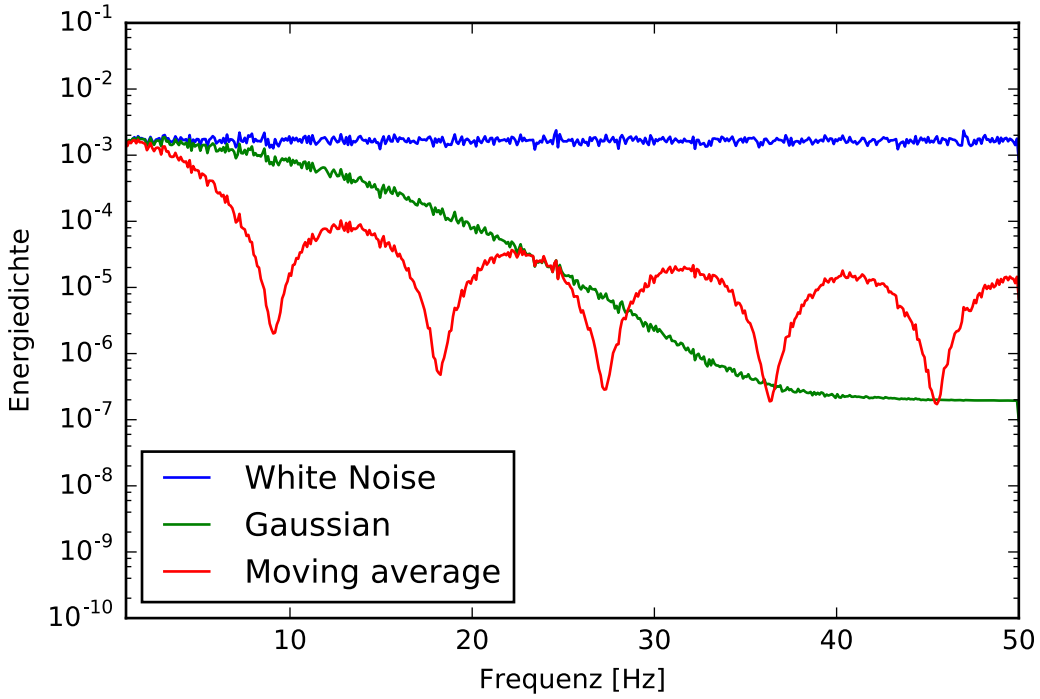
Wir nutzen im Folgenden ein Ensemble-Mittel zum Schätzen des Spektrums. Die Einzelspektren sind sehr verrauscht, daher gibt der Mittelwert aus vielen Spektren eine glattere Kurve. Ensemble-Mittelwert und Standardabweichung können auch zur Abschätzung der Signifikanz von Merkmalen im Spektrum (z.B. Resonanz-Spitzen) genutzt werden.

```
In [29]: %pylab inline
         %config InlineBackend.figure_format = 'svg'

         import scipy.signal as sig

         N=1024
         M=100
         X=rand(N*(M+1))


         V=11
         window = sig.gaussian(V,V/8)
         window=window/sum(window)
         X_filter1=convolve(X, window, mode='valid')
         X_filter2=convolve(X, ones((V,))/V, mode='valid') # Einfacher laufender Mittelwert mittels Fal

         # Test: laufender Mittelwert programmiert als Schleife = identisches Ergebnis wie Faltung
         #v=1
         #for i in range(1,N*M-1):
         #    X_filter2[i]=1.0/(2*v+1)*(sum(X[i-v:i+v+1]))

         Pxx_est=zeros(N/2+1)
         Pxx_est_filter1=zeros(N/2+1)
         Pxx_est_filter2=zeros(N/2+1)

         for i in range(M):
             x=X[i*N:(i+1)*N]
             fs=100 # Sampling Frequenz in Hz
             f, Pxx = sig.periodogram(x, fs)
             Pxx_est+=Pxx

             x_filter1=X_filter1[i*N:(i+1)*N]
             f, Pxx = sig.periodogram(x_filter1, fs)
             Pxx_est_filter1+=Pxx

             x_filter2=X_filter2[i*N:(i+1)*N]
             f, Pxx = sig.periodogram(x_filter2, fs)
             Pxx_est_filter2+=Pxx



         Pxx_est=Pxx_est/M
         Pxx_est_filter1=Pxx_est_filter1/M
         Pxx_est_filter2=Pxx_est_filter2/M
```

```
figure()
#semilogy(f, Pxx)
semilogy(f, Pxx_est,label='White Noise')
semilogy(f, Pxx_est_filter1,label='Gaussian')
semilogy(f, Pxx_est_filter2,label='Moving average')

legend(loc=3)
ylim([1e-10, 1e-1])
xlim([1, 50])

xlabel('Frequenz [Hz]')
ylabel('Energiedichte')
```

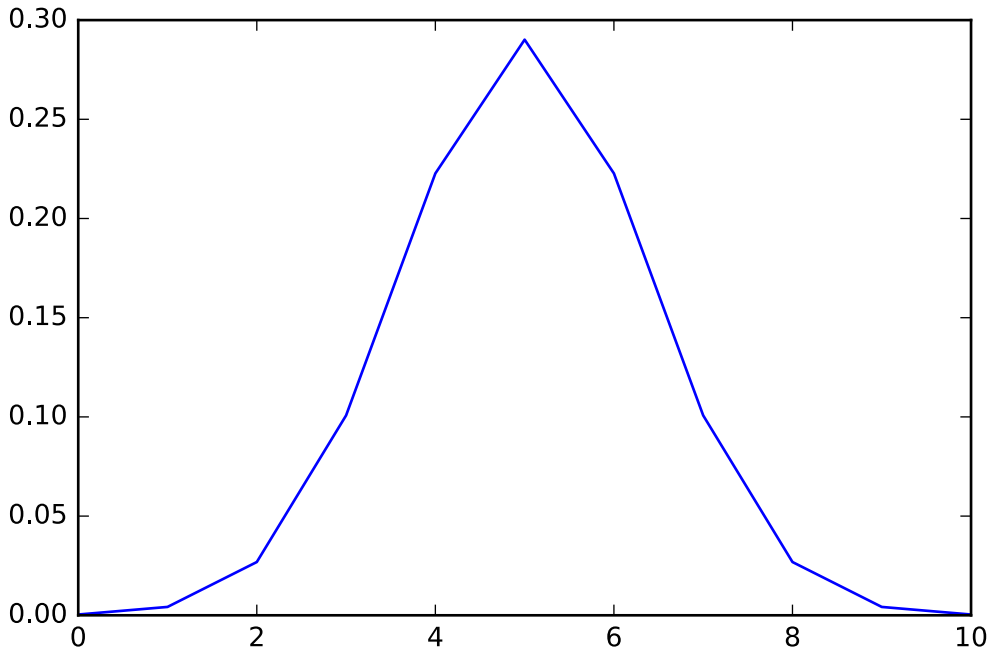
```
Populating the interactive namespace from numpy and matplotlib

WARNING: pylab import has clobbered these variables: ['f']
`%matplotlib` prevents importing * from pylab and numpy
/usr/local/lib/python3.4/dist-packages/ipykernel/__main__.py:22: DeprecationWarning: using a non-integer
/usr/local/lib/python3.4/dist-packages/ipykernel/__main__.py:23: DeprecationWarning: using a non-integer
/usr/local/lib/python3.4/dist-packages/ipykernel/__main__.py:24: DeprecationWarning: using a non-integer
```

```
Out[29]: <matplotlib.text.Text at 0x7f0dc05d7c50>
```




```
In [30]: plot(window)
```

```
Out[30]: [<matplotlib.lines.Line2D at 0x7f0dbbef6e80>]
```



59.2 Diskussion

Die Fouriertransformierte der Rechteckfunktion (genutzt für gleitenden Mittewert)

$$\mathrm{rect}(t) = \Pi(t) = \begin{cases} 0 & \text{if } |t| > \frac{1}{2} \\ \frac{1}{2} & \text{if } |t| = \frac{1}{2} \\ 1 & \text{if } |t| < \frac{1}{2} \end{cases}$$

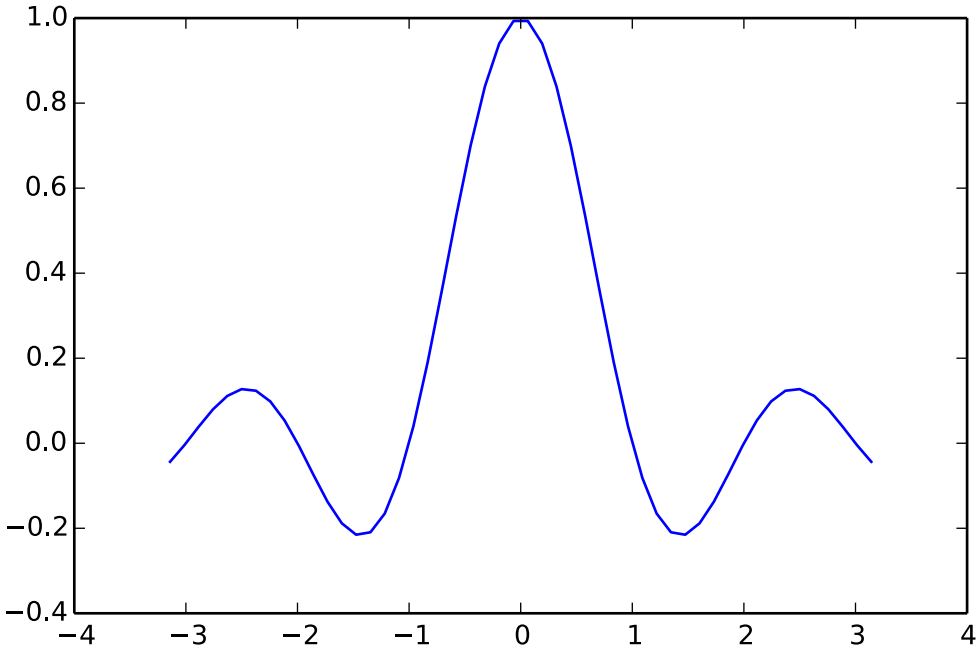
ist die sinc-Funktion

$$H(f) = sinc(f) = \frac{\sin \pi f}{\pi f}$$

https://de.wikipedia.org/wiki/Sinc-Funktion

```
In [2]: f=linspace(-pi,pi)
        plot(f,sin(pi*f)/(pi*f))
```

```
Out[2]: [<matplotlib.lines.Line2D at 0x7f06b90ce190>]
```



Die Nebenmaxima der sinc-Funktion sind deutlich in der Übertragungsfunktion erkennbar. Die Fouriertransformierte der Gauß-Funktion hat wieder die Form einer Gaußfunktion. Daher hat das Gauß-Fenster besondere Glättungseigenschaften.

60 Fenster Funktionen

Dokumentationen aus Modul scipy.signal

60.1 References

1. Blackman, R.B. and Tukey, J.W., (1958) The measurement of power spectra, Dover Publications, New York.
2. E.R. Kanasewich, "Time Sequence Analysis in Geophysics", The University of Alberta Press, 1975, pp. 109-110.
3. Wikipedia, "Window function", http://en.wikipedia.org/wiki/Window_function
4. W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, "Numerical Recipes", Cambridge University Press, 1986, page 425.

60.2 Hamming window

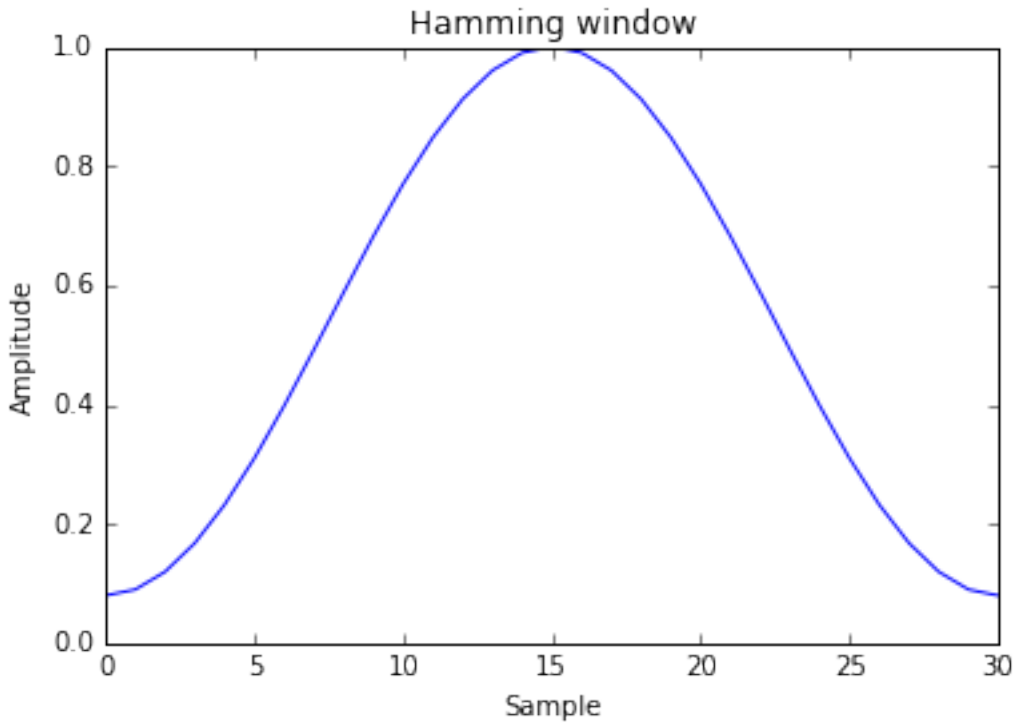
The Hamming window is defined as

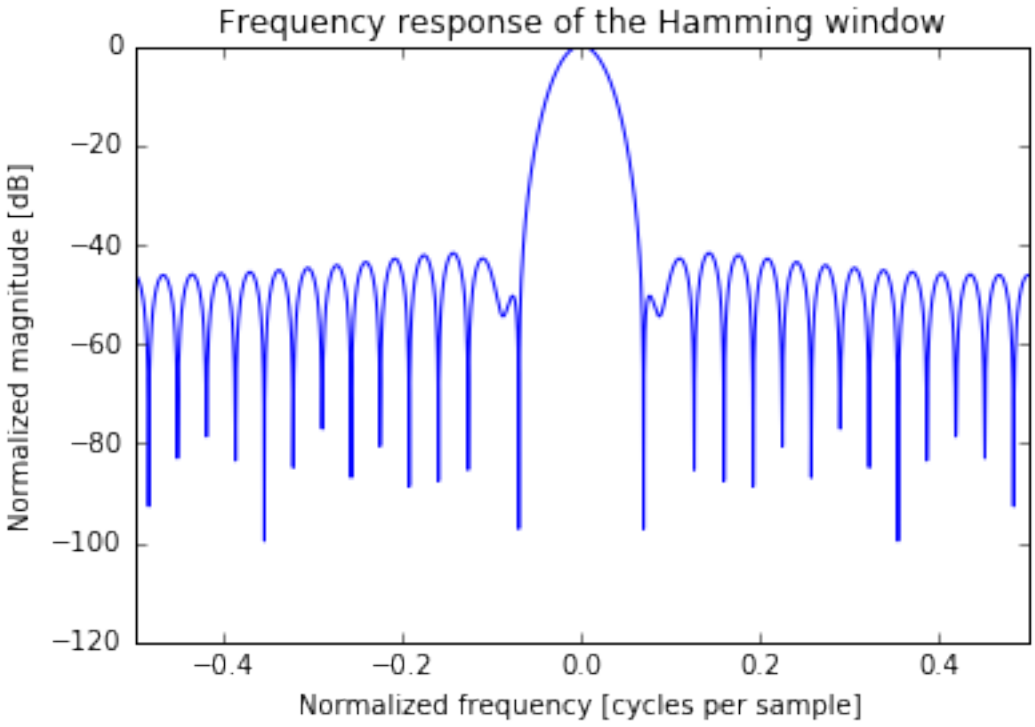
$$w(n) = 0.54 - 0.46\cos\left(\frac{2\pi n}{M-1}\right) \qquad 0 \leq n \leq M-1$$

The Hamming was named for R. W. Hamming, an associate of J. W. Tukey and is described in Blackman and Tukey. It was recommended for smoothing the truncated autocovariance function in the time domain. Most references to the Hamming window come from the signal processing literature, where it is used as one of many windowing functions for smoothing values. It is also known as an apodization (which means "removing the foot", i.e. smoothing discontinuities at the beginning and end of the sampled signal) or tapering function.

In [12]:
```
%pylab inline
%config InlineBackend.figure_format = 'svg'

from scipy import signal
from scipy.fftpack import fft, fftshift

def plot_windows_response(window,name):
    figure()
    plot(window)
    title(name+" window")
    ylabel("Amplitude")
    xlabel("Sample")
    figure()
    A = fft(window, 2048) / (len(window)/2.0)
    freq = np.linspace(-0.5, 0.5, len(A))
    response = 20 * np.log10(np.abs(fftshift(A / abs(A).max())))
    plot(freq, response)
    axis([-0.5, 0.5, -120, 0])
    title("Frequency response of the "+name+" window")
    ylabel("Normalized magnitude [dB]")
    xlabel("Normalized frequency [cycles per sample]")

window = signal.hamming(31)
plot_windows_response(window,"Hamming")
```

```
Populating the interactive namespace from numpy and matplotlib

WARNING: pylab import has clobbered these variables: ['fft']
`%matplotlib` prevents importing * from pylab and numpy
```

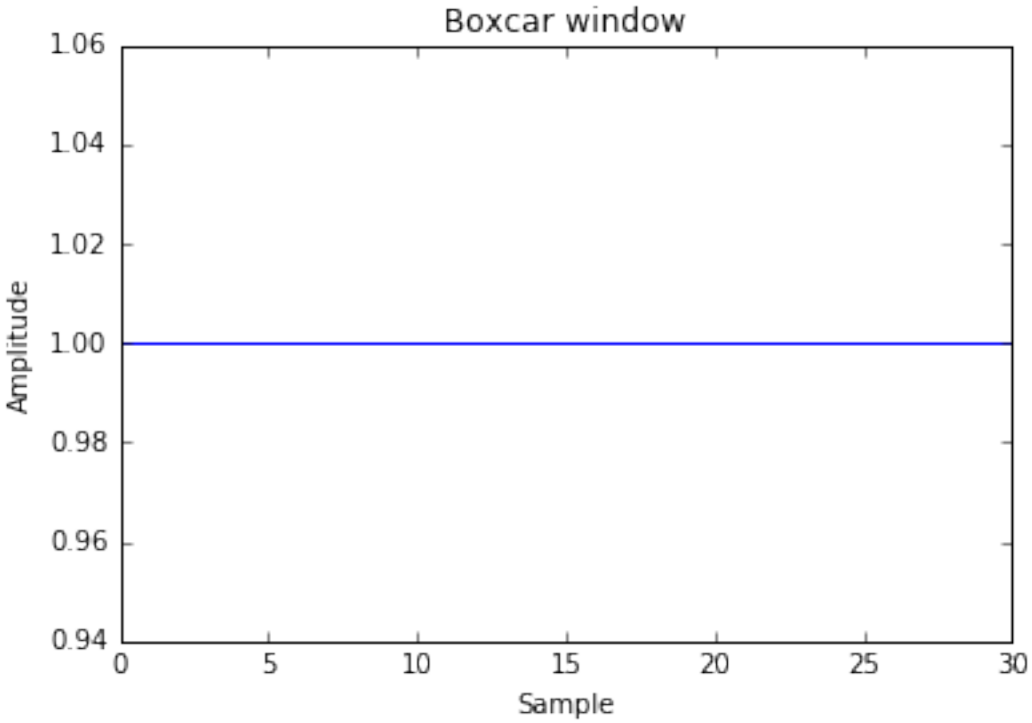


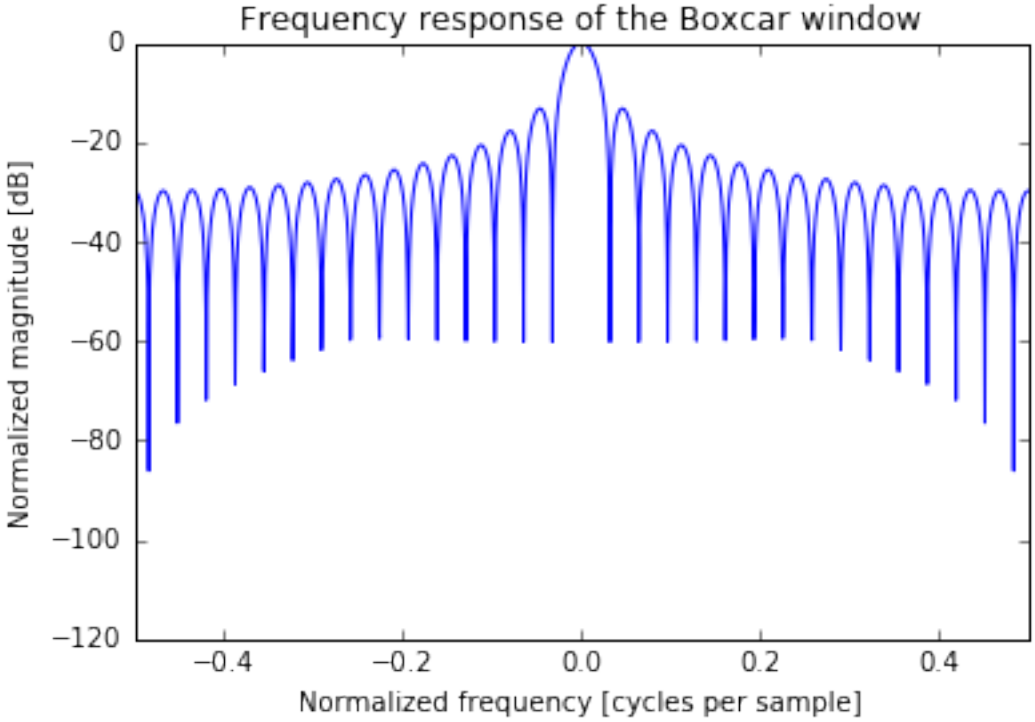


61 Boxcar

Kein Fenster bzw. Gleichgewichtung aller Werte.

```
In [13]: window = signal.boxcar(31)
         plot_windows_response(window,"Boxcar")
```

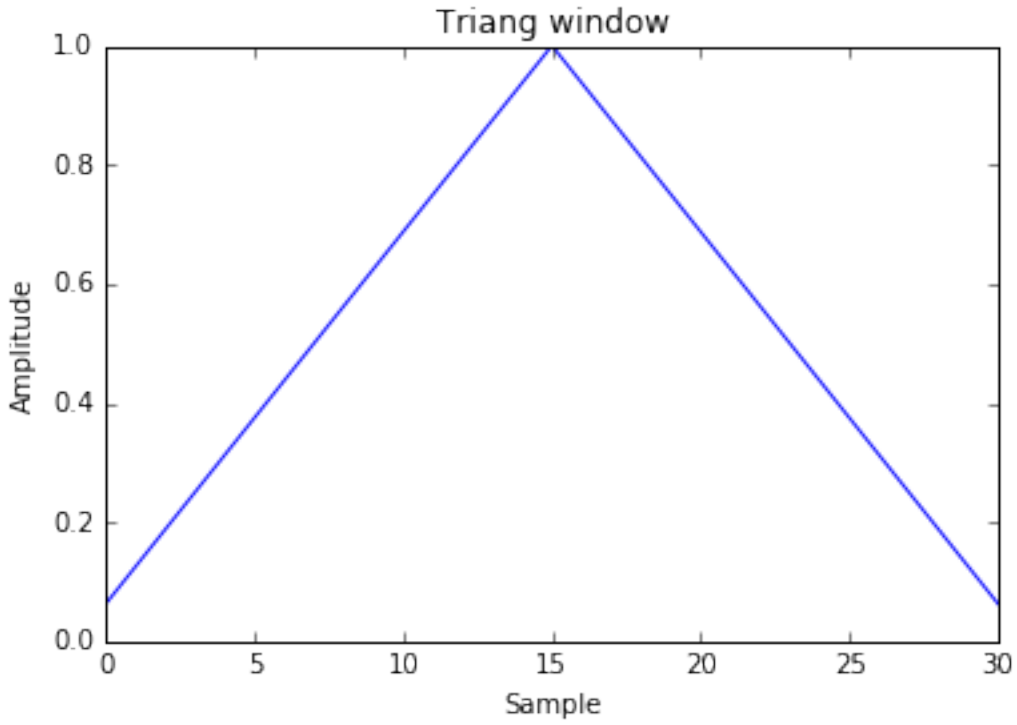

Boxcar window
1.06
1.04
1.02
1.00
0.98
0.96
0.94
Amplitude
0
5
10
15
20
25
30
Sample

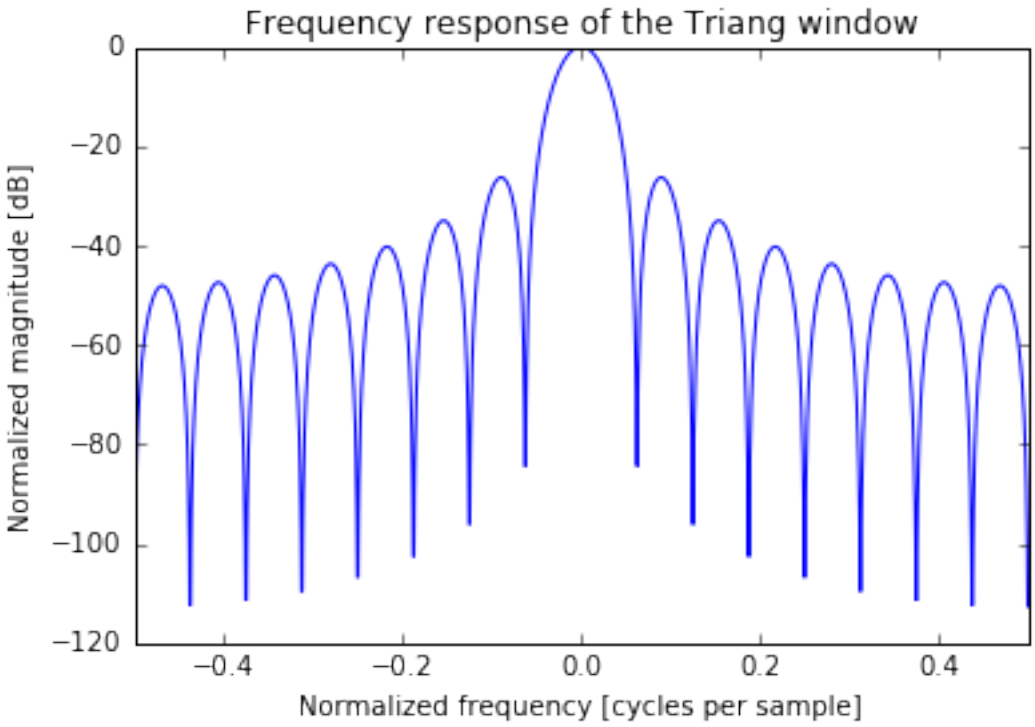

Frequency response of the Boxcar window
0
−20
−40
−60
−80
−100
−120
Normalized magnitude [dB]
−0.4
−0.2
0.0
0.2
0.4
Normalized frequency [cycles per sample]

61.1 Dreieck

```
In [50]: window = signal.triang(31)
         plot_windows_response(window,"Triang")
```

```
/usr/local/lib/python3.4/dist-packages/ipykernel/__main__.py:14: RuntimeWarning: divide by zero encounte
```





61.2 Hanning

The Hann window is defined as

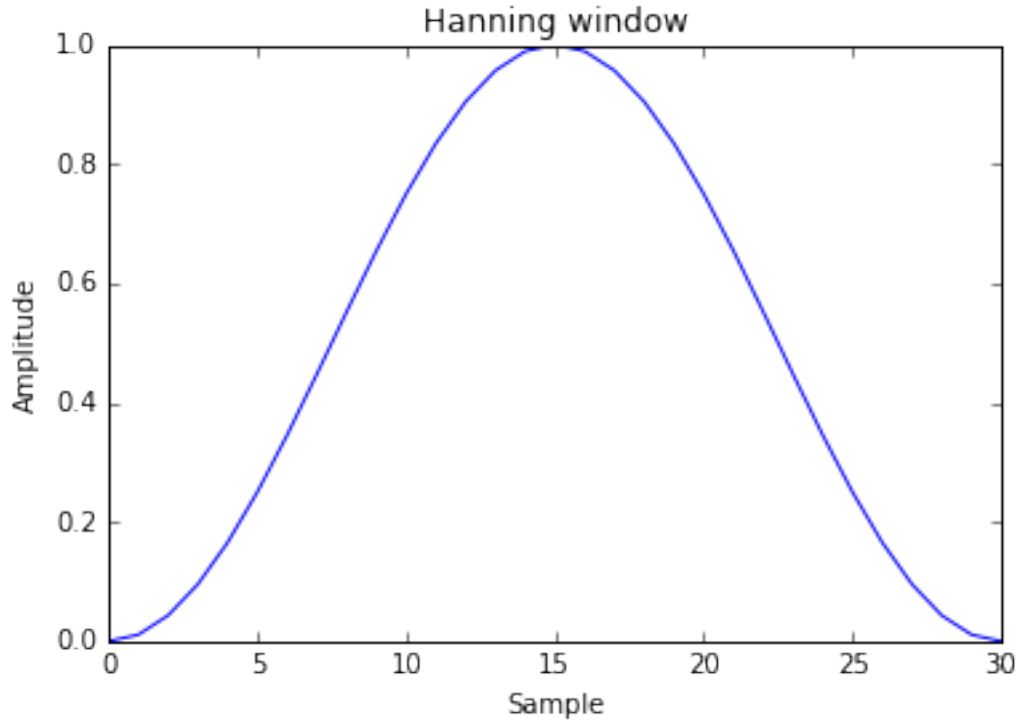
$$w(n) = 0.5 - 0.5\cos\left(\frac{2\pi n}{M-1}\right) \qquad 0 \leq n \leq M-1$$

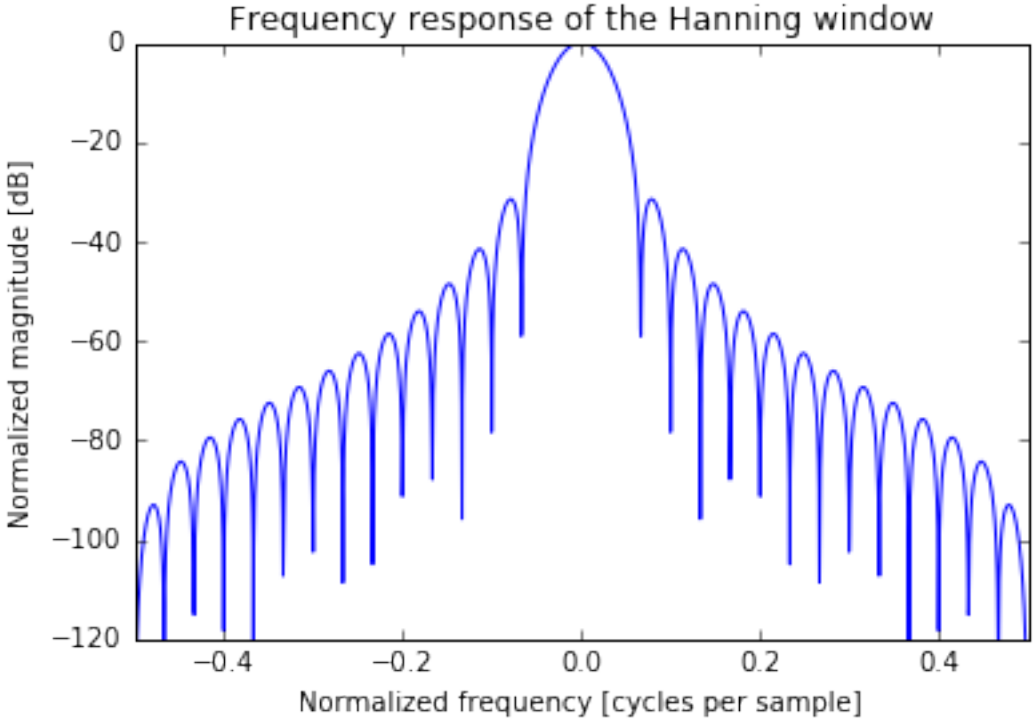
The window was named for Julius van Hann, an Austrian meterologist. It is also known as the Cosine Bell. It is sometimes erroneously referred to as the "Hanning" window, from the use of "hann" as a verb in the original paper and confusion with the very similar Hamming window.

Most references to the Hann window come from the signal processing literature, where it is used as one of many windowing functions for smoothing values. It is also known as an apodization (which means "removing the foot", i.e. smoothing discontinuities at the beginning and end of the sampled signal) or tapering function.

```
In [21]: window = signal.hanning(31)
         plot_windows_response(window,"Hanning")
```

```
/usr/local/lib/python3.4/dist-packages/ipykernel/__main__.py:14: RuntimeWarning: divide by zero encounter
```


Hanning window
Amplitude
Sample
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30

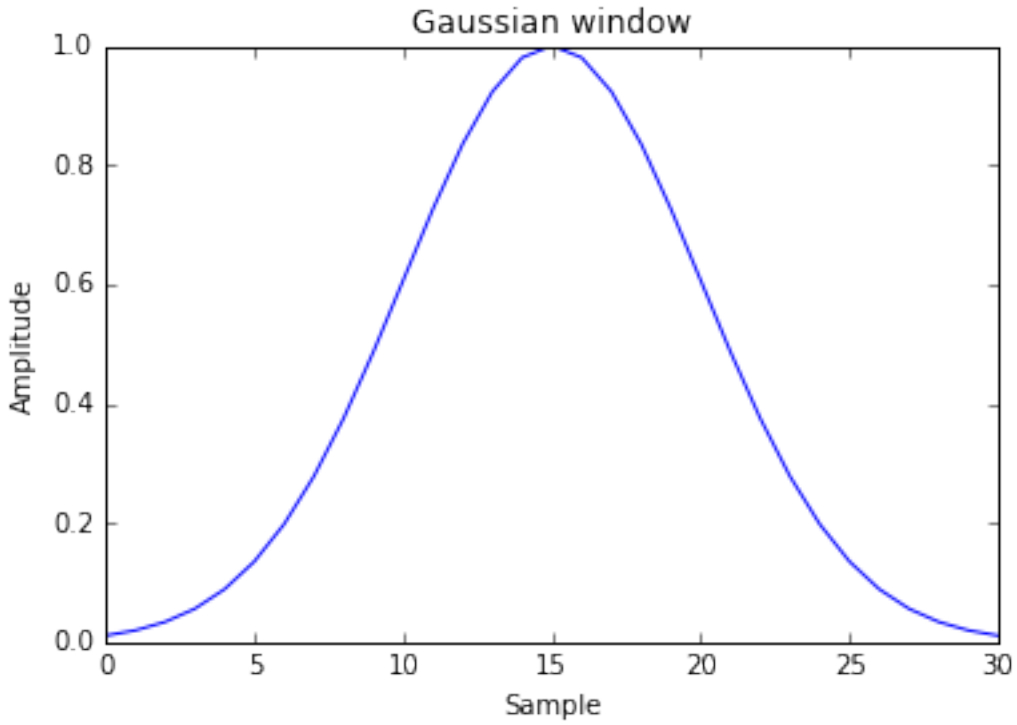

Frequency response of the Hanning window
Normalized magnitude [dB]
Normalized frequency [cycles per sample]
0
−20
−40
−60
−80
−100
−120
−0.4
−0.2
0.0
0.2
0.4

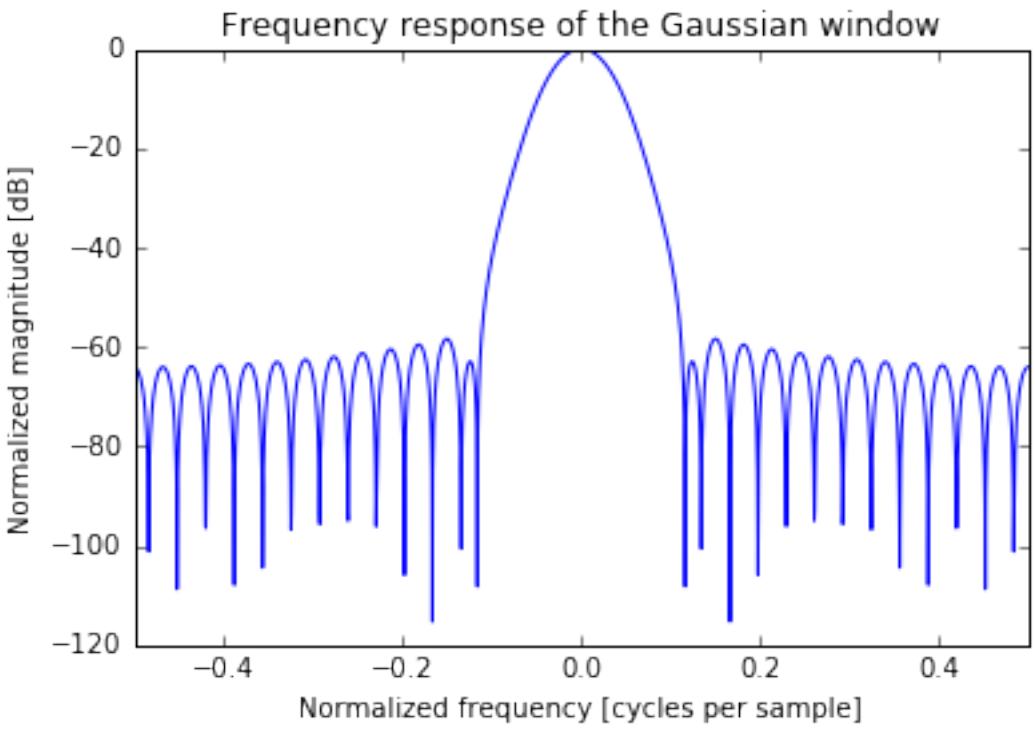
61.3 Gauss

The Gaussian window is defined as

$$w(n) = e^{-\frac{1}{2}\left(\frac{n}{\sigma}\right)^2}$$

```
In [25]: window = signal.gaussian(31,5)
         plot_windows_response(window,"Gaussian")
```






```
In [ ]: window = signal.triang(31)
        plot_windows_response(window,"Triang")
```

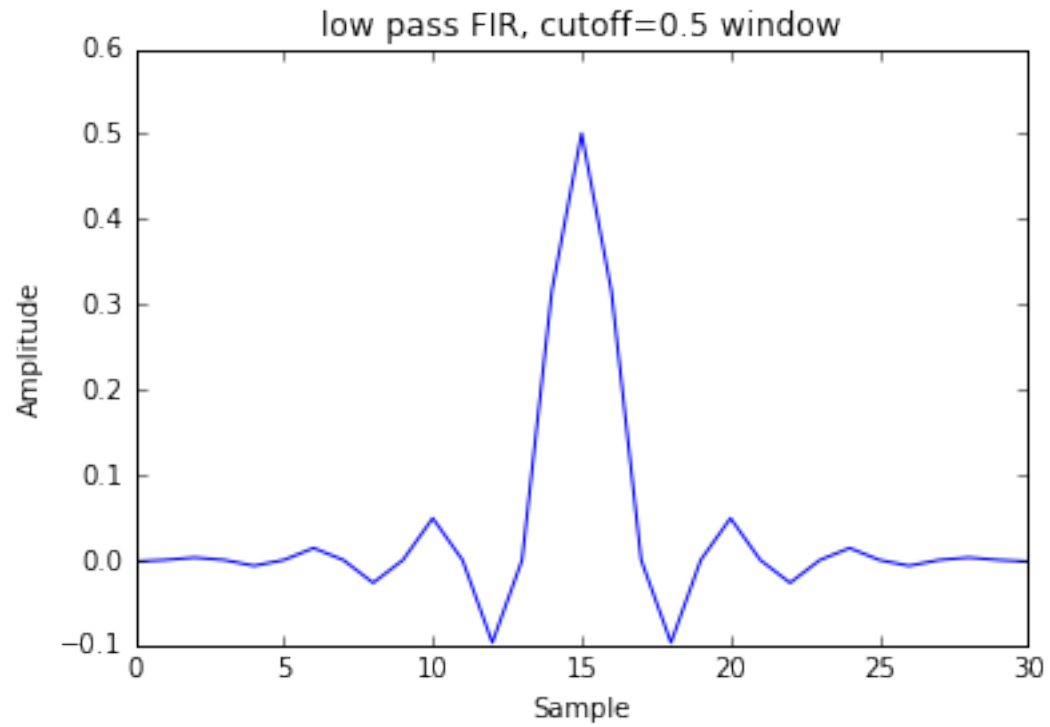
61.4 Filter Design

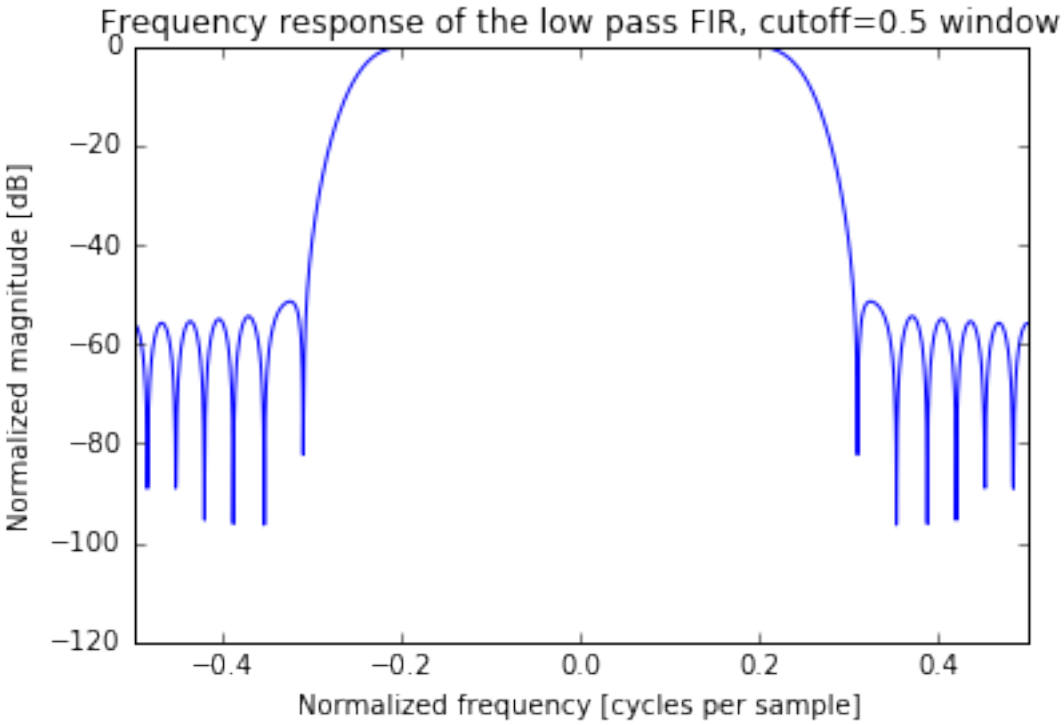
Die Koeffizienten für ein einfaches Filter mit endlicher Impulsantwort können mit der Funktion firwin berechnet werden.

https://de.wikipedia.org/wiki/Filter_mit_endlicher_Impulsantwort

61.4.1 Tiefpass

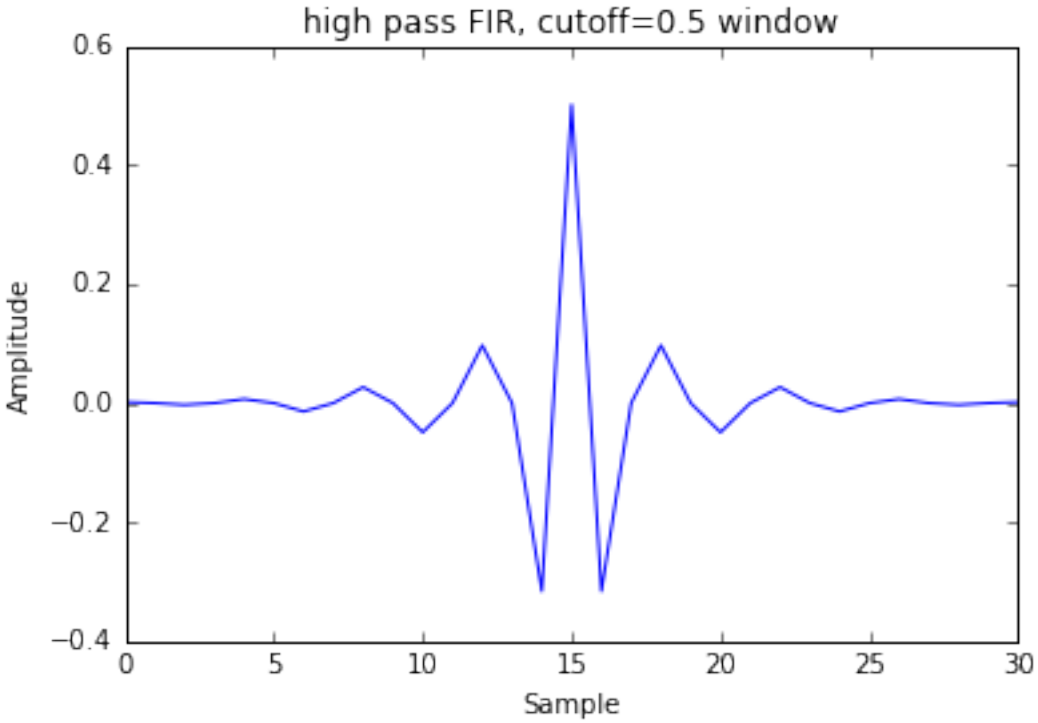
```
In [55]: numtaps=31
         window=signal.firwin(numtaps,cutoff = 0.5, pass_zero=True)
         plot_windows_response(window,"low pass FIR, cutoff=0.5")
```

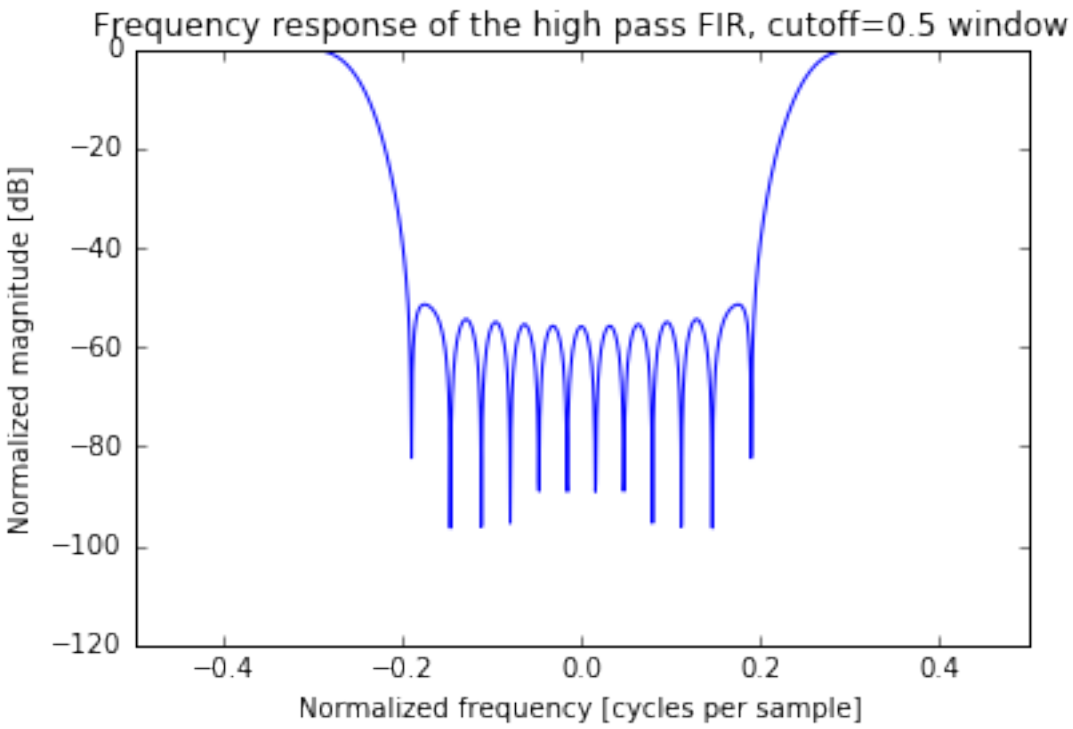

low pass FIR, cutoff=0.5 window
Amplitude
Sample
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
0
5
10
15
20
25
30


Frequency response of the low pass FIR, cutoff=0.5 window
Normalized magnitude [dB]
Normalized frequency [cycles per sample]
0
−20
−40
−60
−80
−100
−120
−0.4
−0.2
0.0
0.2
0.4

61.4.2 Hochpass

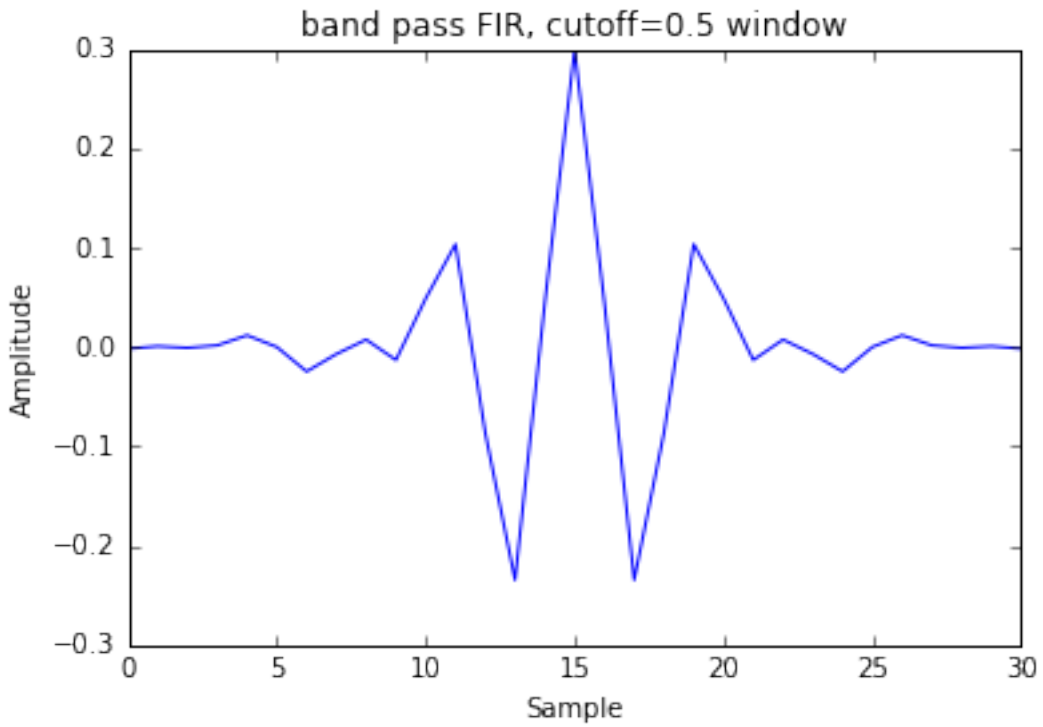
```
In [63]: numtaps=31
         window=signal.firwin(numtaps,cutoff = 0.5, pass_zero=False)
         plot_windows_response(window,"high pass FIR, cutoff=0.5")
```

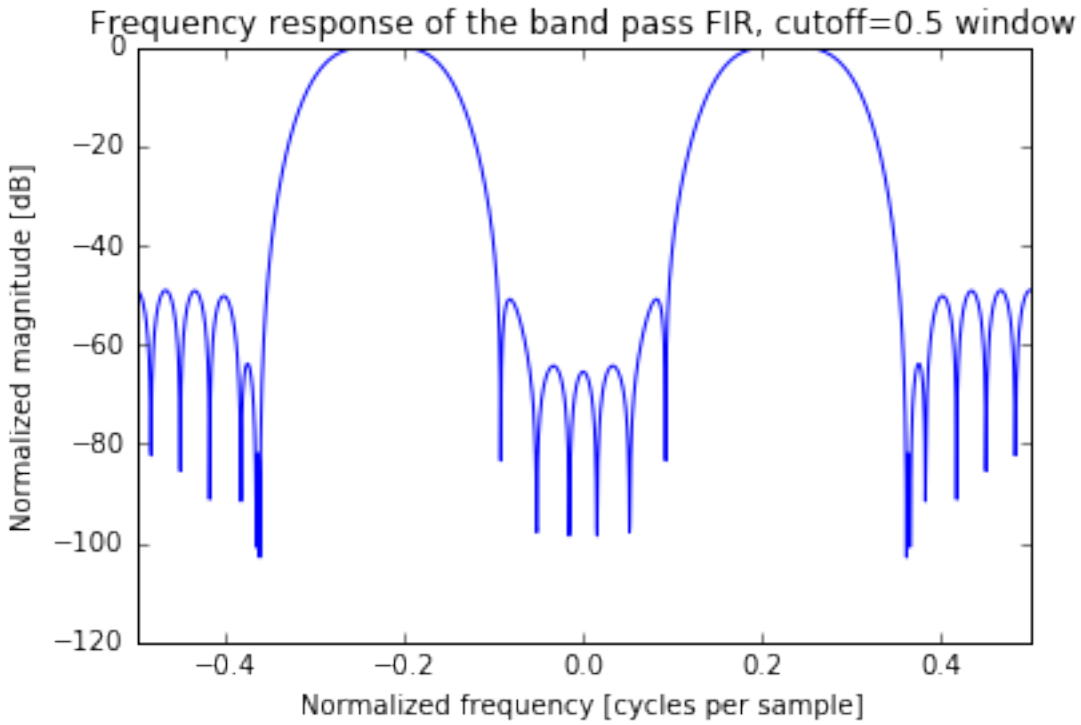




61.4.3 Bandpass

```
In [64]: numtaps=31
         window=signal.firwin(numtaps,cutoff = [0.3,0.6], pass_zero=False)
         plot_windows_response(window,"band pass FIR, cutoff=0.5")
```


band pass FIR, cutoff=0.5 window
Amplitude
Sample
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0
5
10
15
20
25
30


Frequency response of the band pass FIR, cutoff=0.5 window
Normalized magnitude [dB]
Normalized frequency [cycles per sample]
0
−20
−40
−60
−80
−100
−120
−0.4
−0.2
0.0
0.2
0.4

In [ ]:

62 Spektralanalyse Driftbojendaten

Im Folgenden nutzen wir die Daten einer GPS-Boje, um mittels Spektralanalyse die Periode von Bewegungs-Oszillationen zu bestimmen.

Hintergrund zu Feldexperiment: Kaleschke, L., et al., SMOS sea ice product: Operational application and validation in the Barents Sea marginal ice zone, Remote Sensing of Environment (2015), http://dx.doi.org/10.1016/j.rse.2016.03.009

```
In [22]: %pylab inline
         %config InlineBackend.figure_format = 'svg'
         import pandas as pd
         import scipy.ndimage as ndimage
         import scipy.signal as signal
         from mpl_toolkits.basemap import Basemap, addcyclic
```

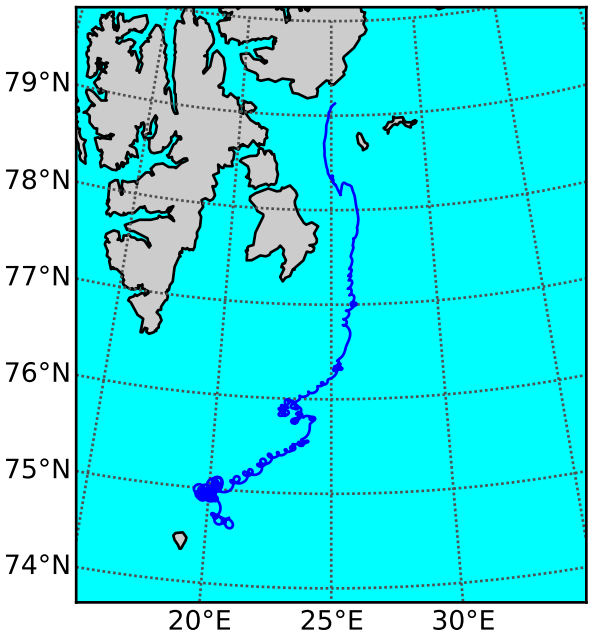
```
Populating the interactive namespace from numpy and matplotlib
```

```
In [12]: D=pd.read_csv('CliSAP_Boje_16.csv')
         Lon=array(D['Lon'])
         Lat=array(D['Lat'])
         D.head()
```

```
Out[12]:              Date(GMT)      Lat      Lon
         0  2014-03-15 00:00:00  74.7080  20.8926
         1  2014-03-15 00:15:00  74.7050  20.8706
         2  2014-03-15 00:30:00  74.7014  20.8508
         3  2014-03-15 00:45:00  74.6974  20.8320
         4  2014-03-15 01:00:00  74.6928  20.8152
```

```
In [49]: m = Basemap(width=600000,height=700000,resolution='i',projection='laea',lat_ts=80,lat_0=77,lon
         m.drawmapboundary(fill_color='aqua')
         m.drawparallels(range(-90,100,1), color='#505050',labels=[1,0,0,0])
         m.drawmeridians(range(-180,180,5), color='#505050',labels=[0,0,0,1])
         m.drawcoastlines(linewidth=1.0, color='#000000')
         m.fillcontinents()
         x,y=m(Lon,Lat)
         m.plot(x,y)
```

```
Out[49]: [<matplotlib.lines.Line2D at 0x7f9cf84c8a58>]
```

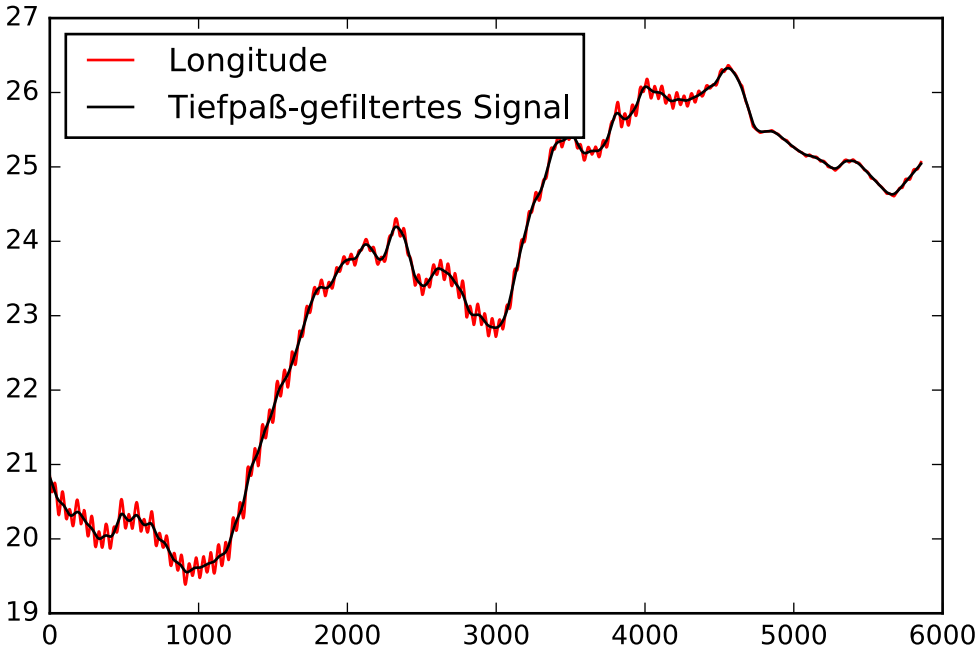


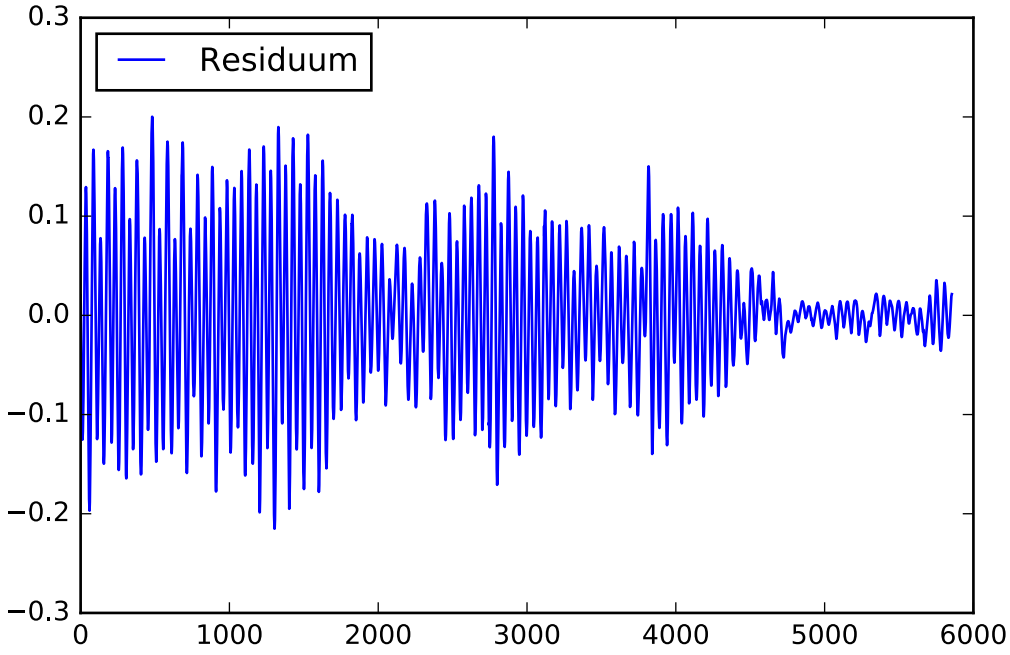
62.1 Tiefpaßfilter zur Berechnung von Residuuen

Zunächst entfernen wir die langsame Drift mittels eines Glättungs-Filters

In [82]:
```
y=Lon

numtaps=2*24*2+1
window=signal.hamming(numtaps)
window=window/sum(window)
y_tiefpass=convolve(y, window, mode='valid')

edge=int(numtaps/2)
y_residuum=y[edge:-edge]-y_tiefpass
y=y[edge:-edge]
figure()
plot(y,'r-',label='Longitude')
plot(y_tiefpass,'k-',label='Tiefpaß-gefiltertes Signal')
legend(loc=2)
figure()
plot(y_residuum,label='Residuum')
legend(loc=2)
```

Out[82]:
```
<matplotlib.legend.Legend at 0x7f9cf8340cf8>
```





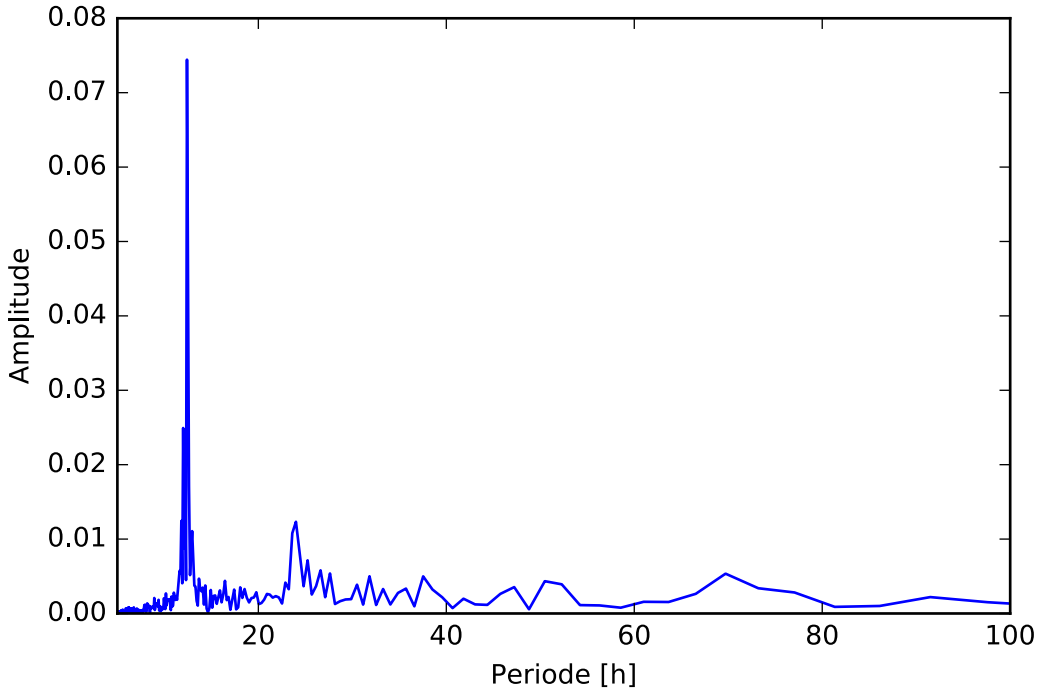
62.2 Frequenzspektrum

```
In [83]: dt=15*60.0
         fa=1/dt # Abtastfrequenz
```

```
Y=fft.fft(y_residuum)
N = int(len(Y)/2)+1 # Halbe Länge reicht (gespiegelten Teil ignorieren)

Th=1/(linspace(0, fa/2, N)*60*60) # Umrechnen in Periode, Einheit Stunden
index_max=argmax(Y[:N])
print('Periode max ', Th[index_max],'h')
plot(Th,abs(Y[:N])/N)
xlabel('Periode [h]')
ylabel('Amplitude')
xlim([5,100])
```

```
Periode max  12.406779661 h
```

Out[83]: (5, 100)



62.3 Gezeiten oder Trägheitsschwingung?

Gezeit M2: 12.421 h

Die Periode der Trägheitsschwingung ist abhängig von geographischer Breite ϕ

$$f = 2\Omega \sin(\phi)$$

$$T = \frac{2\pi}{f}$$

Literatur: Pease et al. (1995), Barents Sea tidal and inertial motions from Argos ice buoys during the Coordinated Eastern Arctic Experiment, http://onlinelibrary.wiley.com/doi/10.1029/95JC03014/abstract

```
In [89]: phi=deg2rad(77.0)
         w=2*pi/86164.09
         f=2*w*sin(phi)
         T=2*pi/f
         print(T/60/60)
```

```
12.2820221543
```

62.4 Frequenzauflösung

Ist die Frequenzauflösung ausreichend, um zwischen Gezeit und Trägheitsschwingung zu unterscheiden?

```
In [85]: Th[index_max]
```

```
Out[85]: 12.40677966101695
```

```
In [87]: Th[index_max+1]
```

```
Out[87]: 12.302521008403362
```

63 Singulärwertzerlegung

Jede $m \times n$ Matrix A läßt sich darstellen als

$$\underbrace{A}_{m\times n} = \underbrace{U}_{m\times m}\underbrace{s}_{m\times n}\underbrace{V^T}_{n\times n}$$

wobei U und V orthogonal sind und s diagonal ist. Aus der Orthogonalität folgt, dass die Spaltenvektoren u_i eine Basis des $m-$dimensionalen "Datenraumes" und v_i eine Basis des n-dimensionalen "Parameterraumes" bilden. Die Diagonalelemente s nennt man die singulären Werte von A. Diese sind nicht negativ und der Größe nach geordnet

$$s_1 \geq s_2 \geq .. \geq s_m \geq 0$$

Ist p die Anzahl (=Rang) der von 0 verschiedenen singulären Werte von A, so kann A geschrieben werden als

$$\underbrace{A}_{m\times n} = \underbrace{U_p}_{m\times p}\underbrace{s_p}_{p\times p}\underbrace{V_p^T}_{p\times n}$$

dabei werden die U_p aus U durch weglassen der Spalten $p+1, .., m$ gebildet (V_p analog).
Jeder Vektor x des Parameterraumes kann dargestellt werden als

$$x = \sum_{i=1}^{p} \alpha_i v_i + \sum_{j=p+1}^{n} \beta_j v_j$$

oder als

$$\mathbf{x} = \mathbf{x}^{\mathbf{part}} + \mathbf{x}^{\mathbf{null}}$$

mit $\mathbf{A}\mathbf{x}^{\mathbf{null}} = \mathbf{0}$.

63.1 Unterbestimmte Gleichungssystemte

Sei $Ax = d$ ein zu lösendes Gleichungssystem. Ist

$$\text{rang}(A) = p < n$$

dann ist das Gleichungssystem nicht eindeutig lösbar. Denn hat man eine partikuläre Lösung $\mathbf{x^{part}}$ gefunden, die $F = ||\mathbf{Ax^{part}} - \mathbf{d}||$ minimiert, so minimiert auch $\mathbf{x} = \mathbf{x^{part}} + \mathbf{x^{null}}$ die Funktion F

Strategie: Wähle aus der unendlichen Anzahl von Lösungen die spezielle Lösung mit dem "kleinsten" Betrag $\min(||x||)$. Die "kleinste" Optimallösung ist gegeben durch

$$x^{part} = V_p S_p^{-1} U_p^T d$$

Für überbestimmte Gleichungssysteme ist x^{part} die eindeutige Optimallösung

63.2 Fazit

Die Singulärwertzerlegung der Matrix gibt uns Informationen über den "Zustand" des Gleichungssystems. Im Fall eines unterbestimmten Gleichungssystems liefert $x^{part} = V_p S_p^{-1} U_p^T d$ die betragskleinste Optimallösung, im Falle eines überbestimmten Systems ist diese Lösung eindeutig.

63.3 Kondition einer Matrix

Sei A eine beliebige $m \times n$ Matrix und

$$\underbrace{A}_{m\times n} = \underbrace{U_p}_{m\times p}\underbrace{s_p}_{p\times p}\underbrace{V_p^T}_{p\times n}$$

ihre Singulärwertzerlegung mit $p \leq \min(m, n)$. Dann ist die Kondition der Matrix A definiert als Quotient des größten und kleinsten Singulärwert

$$\text{cond}(A) = \frac{S_1}{S_p}$$

VORSICHT bei großen Werten der Kondition! Eine Matrix A mit $\text{cond}(A) >> 1$ nennt man "schlecht konditioniert" (ill-conditioned). Es gibt zwischen den Matrix-Zeilen "fast" lineare Abhängigkeiten.

Die Kondition ist ein Maß für die Änderung des Lösungsvektors δx bei Änderung der Daten δd.

$$||\delta x|| \sim \text{cond}(A)||\delta d||$$

64 Beispiel: schlecht konditioniertes Gleichungssystem

```
In [36]: from pylab import *
         A=array([(1.0,1.0),(1.01,1.)],dtype=float)
         A

Out[36]: array([[ 1.  ,  1.  ],
                [ 1.01,  1.  ]])

In [37]: cond(A)

Out[37]: 402.00751248429634

In [39]: U,s,V=svd(A)

In [35]: sqrt(2)

Out[35]: 1.4142135623730951
```

```
In [6]: s[0]/s[1]

Out[6]: 4002.0007501252289

In [40]: s

Out[40]: array([ 2.0050125,  0.0049875])

In [18]: d=array([2.0,2.001]).T
         # Optimal-Lösung
         x=dot(V,dot(inv(diag(s)),dot(U.T,d)))
         x

Out[18]: array([-0.1  ,  2.101])

In [19]: d=array([2.0,2.02]).T # kleine Änderung, z.B. durch Meßfehler
         # Optimal-Lösung
         x=dot(V,dot(inv(diag(s)),dot(U.T,d)))
         x

Out[19]: array([-2.  ,  4.02])
```

65 VORSICHT!!!

Bei schlecht konditionierten Gleichungssystemen ist größte Vorsicht geboten! Kleine Änderungen in den Datenvektoren, z.B. durch Messungenauigkeiten, führen zu großen Änderungen in der Lösung für die zu bestimmenden Parameter.

```
In [46]: A=array([(2.0,3.01),(1.001,-1.)],dtype=float)
         print cond(A)
         A=array([(2.0,3.01),(2.001,3.)],dtype=float)
         print cond(A)
         A=array([(1,.01),(.01,0.0001)],dtype=float)
         print cond(A)

2.62342047842
1132.72840879
7.3798044618e+19

In [ ]:
```

66 Empirische Orthogonalfunktionen

Jede (stetige) Funktion f lässt sich darstellen als Linearkombination von orthogonalen Basisfunktionen

$$f(x) = \sum_{i}^{\infty} \alpha_i o_i(x)$$

Beispiele: Fourierreihen ($o_i(x) = \sin(\nu_i x)$), Polynome ($o_i(x) = x^{i-1}$), Kugelfunktionen etc.

Im Gegensatz zu diesen allgemeinen Zerlegungen sind die sogennanten empirischen Orthogonalfunktionen (EOFs) auf ein spezielles Problem zugeschnitten.

66.1 Anwendungen der EOF-Zerlegung

- Luftdruckmuster: Arktische Oszillation (AO)
- Muster von Ozeanoberflächentemperaturen-Zeitreihen: PDO
- Analyse von Altimeter-Daten (Meeresspiegel-Anstieg)
- Füllen von Datenlücken
- Singular Spectrum Analysis

66.2 Beispiel Argo-Profile: Temperatur und Salzgehaltsprofile

Argo-Floats (http://en.wikipedia.org/wiki/Argo_%28oceanography%29) messen Temperatur und Salzgehalt.
Seien an n Punkten Temperatur- oder Salzgehaltsprofile $d_i (i = 1, .., n)$ gemessen.

$$d_i = [S_i(z_1), S_i(z_2), .., S_i(z_m)]^T$$

(m Messungen in den Tiefen z_1, z_2,..,z_m)
Fasst man die Daten d_i in der $n \times m$-Matrix $\mathbf{D}$ zusammen

$$\mathbf{D} = \mathbf{O}\alpha$$

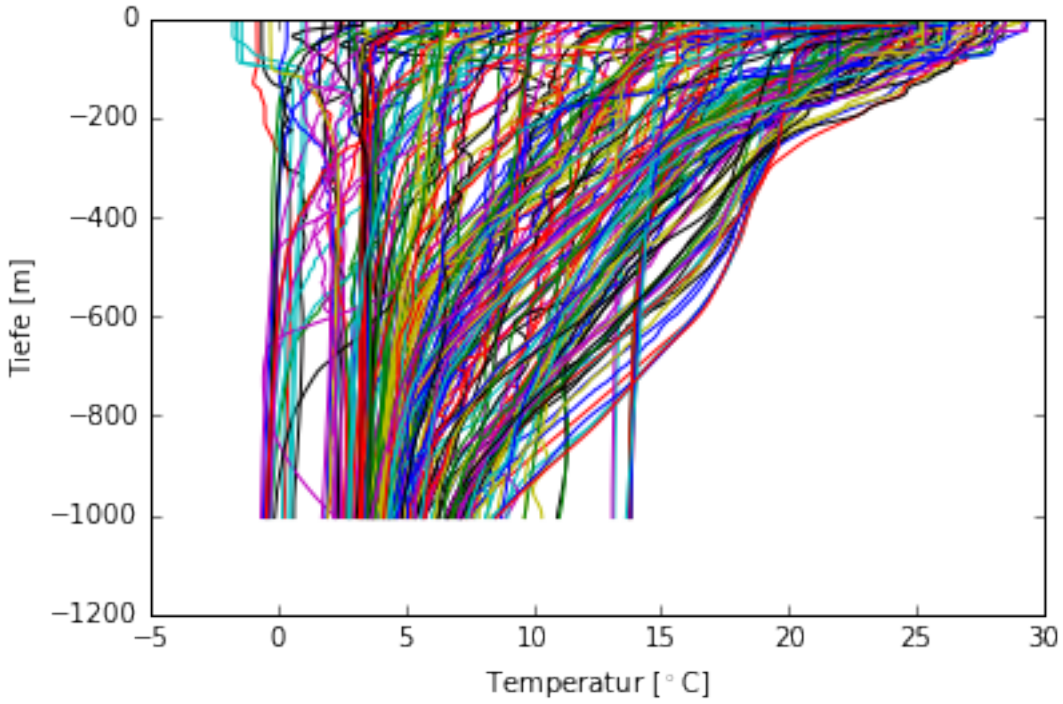
so erhält man die Koeffizienten α und Orthogonalvektoren $\mathbf{O}$ aus der Singulärwertzerlegung (SVD) der Datenmatrix

$$\mathbf{D} = \mathbf{USV}^T$$

als $\alpha = \mathbf{U}$ und $\mathbf{O} = \mathbf{SV}^T$.

In [57]:
```
N=236 # Anzahl Profile, Ausgabe
DATA=reshape(fromfile('T_S_profiles.dat',dtype=float,sep=','),(200,2,N))

Z=linspace(5,1005,200)
T=DATA[:,0,:].T
S=DATA[:,1,:].T

D=T # wir nehmen als Datenvektor den Salzgehalt/ die Temperatur
for i in range(N):
    plot(D[i],-Z)
xlabel('Temperatur [$^\circ$C]')
ylabel('Tiefe [m]')
```

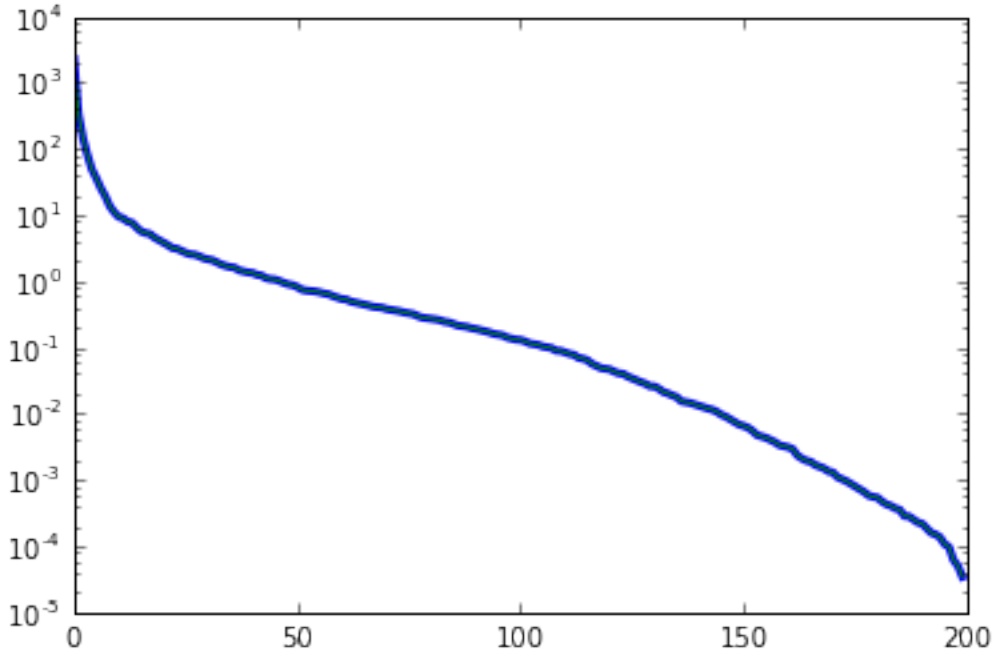
Out[57]:
```
<matplotlib.text.Text at 0x7f80d41c99b0>
```



66.3 SVD-Zerlegung

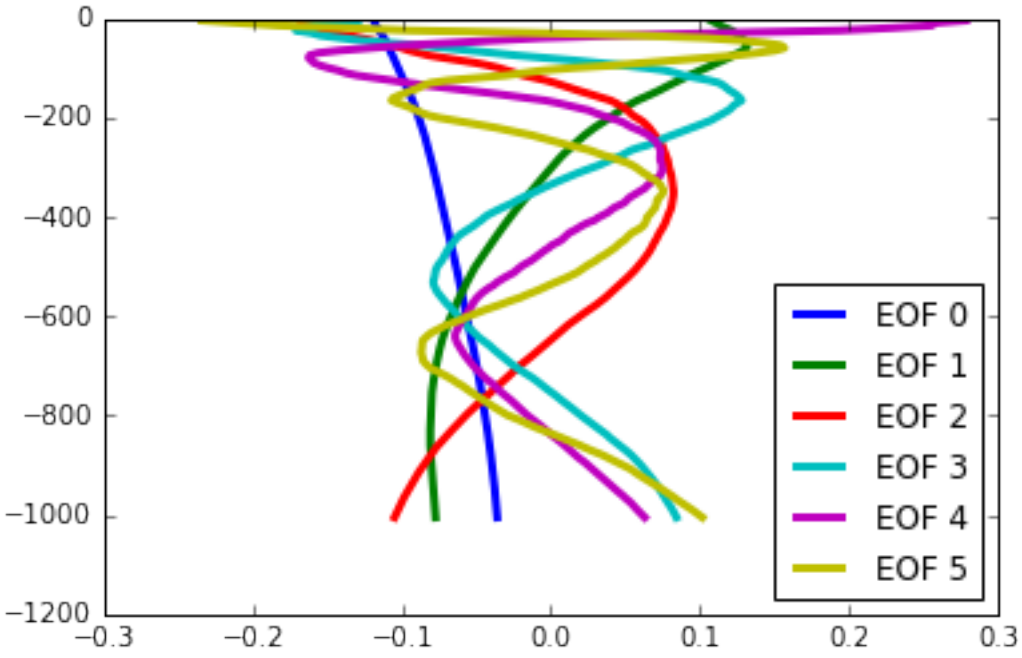
```
In [58]: U, s, V = svd(D, full_matrices=False)
         semilogy(s,linewidth=3)
         plot(s)
```

```
Out[58]: [<matplotlib.lines.Line2D at 0x7f80a8414278>]
```



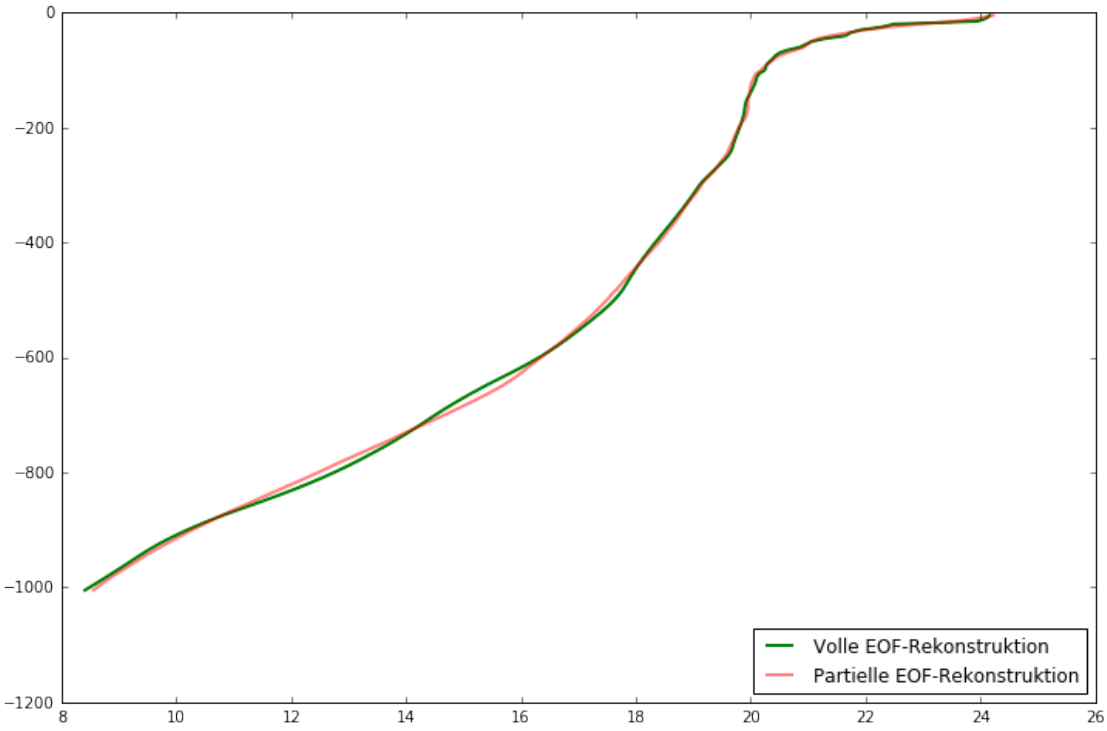
In [59]:
```
for i in range(6):
    plot(V[i,:],-Z,label='EOF '+str(i),linewidth=3)
legend(loc=4)
```

Out[59]: <matplotlib.legend.Legend at 0x7f80d438bcc0>



```
In [65]: i=3# Index Profil für Beispielplot
         D_r=dot(U,dot(diag(s),V)) # Volle Rekonstruktion mit allen Singulärwerten

         O=zeros((200,200))
         k=10# Nutze EOF 0-k zur Rekonstruktion
         O[0:k,:]=dot(diag(s[0:k]),V[0:k,:])
         D_k=dot(U,O)
         figure(figsize=(12,8))
         plot(D[i,:],-Z,'k-',linewidth=1)
         plot(D_r[i,:],-Z,'g-',linewidth=2,label='Volle EOF-Rekonstruktion')
         plot(D_k[i,:],-Z,'r-',linewidth=2,alpha=0.5,label='Partielle EOF-Rekonstruktion')
         legend(loc=4)
         print('Mittlere quadratische Abweichung: ',rms_flat(D[i,:]-D_k[i,:]))
```

```
Mittlere quadratische Abweichung:  0.131615953735
```



66.4 EOF-Rekonstruktion

Die EOF-Rekonstruktion mit $k << m$ Koeffizienten liefert eine brauchbare Näherung an das Orginalprofil. Das Orginal-Profil enthält 200 Werte, die durch nur wenige Koeffizienten und die entsprechenden Orthogonalfunktionen repäsentiert wird.

66.5 Übung

Führen Sie die EOF-Zerlegung für die N Salzgehaltsprofile durch. Beurteilen Sie die Qualität der EOF-Rekonstruktion für einige Stichproben. Wie viele Koeffizienten k sind notwendig, um eine gute Näherung zu bekommen?

66.6 Argo-Prozessierung

Zunächst müssen wir die Daten etwas aufbereiten, bevor wir uns der EOF-Zerlegung widmen.

- Download vom FTP-Server
- Einlesen und Darstellen
- Interpolation auf äquidistante Arrays zwischen 5-1005 m in 5 m Abstand
- Speichern der Zwischnenergebnisse für die weitere Bearbeitung

66.6.1 Download vom FTP-Server

```
In [1]: #!/usr/bin/env python
        import os,os.path
        from ftplib import FTP

        tmp_dir='data/'
        os.system('mkdir '+tmp_dir)

        ftp_adr='ftp.ifremer.fr'
        ftp_dir='/ifremer/argo/geo/atlantic_ocean/2016/06/'

        ftp = FTP(ftp_adr)   # connect to host, default port
        ftp.login()          # user anonymous, passwd anonymous@
        ftp.cwd(ftp_dir)

        file_list=ftp.nlst()

        for f in file_list:
            urlfile='ftp://'+ftp_adr+ftp_dir+f
            localfile=tmp_dir+f

            if not(os.path.exists(localfile)):
                print('Getting file '+f)
                 #We use curl instead of ftp.retrbinary for download
                os.system("curl -s -k -o "+localfile+"  "+urlfile)
            else:
                print('file '+f+' exists')

        ftp.close()
```

```
Getting file 20160601_prof.nc
Getting file 20160602_prof.nc
Getting file 20160603_prof.nc
Getting file 20160604_prof.nc
Getting file 20160605_prof.nc
Getting file 20160606_prof.nc
Getting file 20160607_prof.nc
```

66.6.2 Vorprozessierung

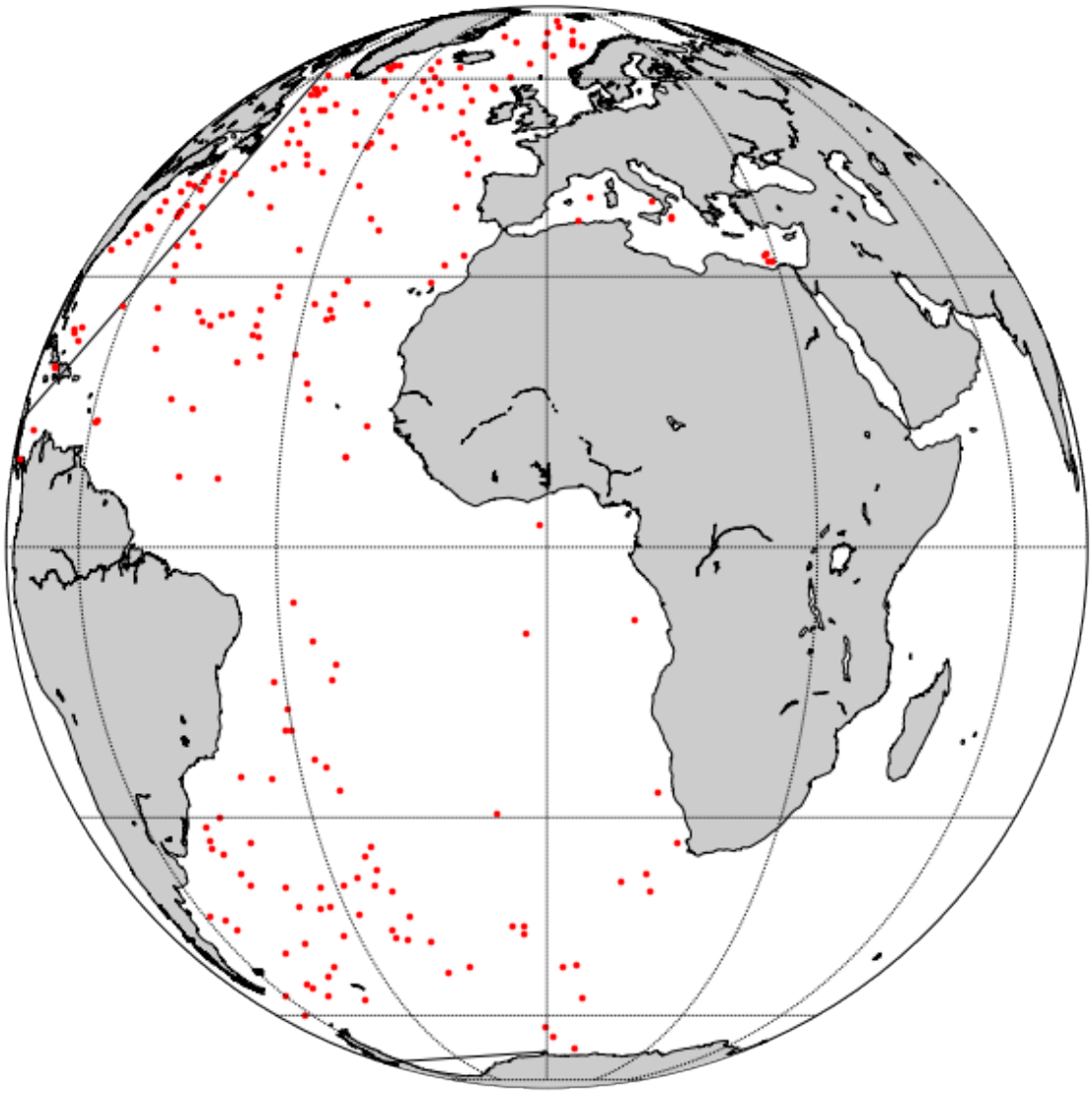
```
In [28]: %pylab inline
         import scipy.io as io
         import glob
         from pylab import *
         from mpl_toolkits.basemap import Basemap
```

```
import scipy.interpolate as interpolate

tmp_dir='data/'

file_liste=glob.glob(tmp_dir+'*.nc')

D={}# Empty dictionary to store selected profiles
for f in file_liste:# Loop over all data
    # Open netcdf data file
    fid=io.netcdf_file(f,'r')

    # Read content into variables
    lat=fid.variables['LATITUDE'][:].copy()
    lon=fid.variables['LONGITUDE'][:].copy()
    T=fid.variables['TEMP_ADJUSTED'][:].copy()
    S=fid.variables['PSAL_ADJUSTED'][:].copy()
    P=fid.variables['PRES_ADJUSTED'][:].copy()

    #T=fid.variables['TEMP'][:].copy()
    #S=fid.variables['PSAL'][:].copy()
    #P=fid.variables['PRES'][:].copy()


    T[T>=99999]=nan # Set 99999.0 to "Not a Number"
    P[P>=99999]=nan
    S[S>=99999]=nan

    (nr_profs,Z)=T.shape # Get dimension

    nm=nanmax(P,axis=1)
    cantdoit=0
    for i in range(nr_profs):
        if nm[i]>1000:
            #Spline interpolation

            x=P[i,:]
            y=T[i,:]
            ind=isfinite(y)
            Z=linspace(10,1010,200)
            try:
                tck=interpolate.splrep(x[ind],y[ind],k=2) # Calculate coefficients at the supp
                Ti=interpolate.splev(Z,tck)

                x=P[i,:]
                y=S[i,:]
                ind=isfinite(y)

                tck=interpolate.splrep(x[ind],y[ind],k=2) # Calculate coefficients at the supp
                Si=interpolate.splev(Z,tck)
                if sum(isnan(Si))==0:
                    if max(Ti)<35.0:
                        D[(lon[i],lat[i])]=(Ti,Si,Z)
            except:
                cantdoit+=1
```

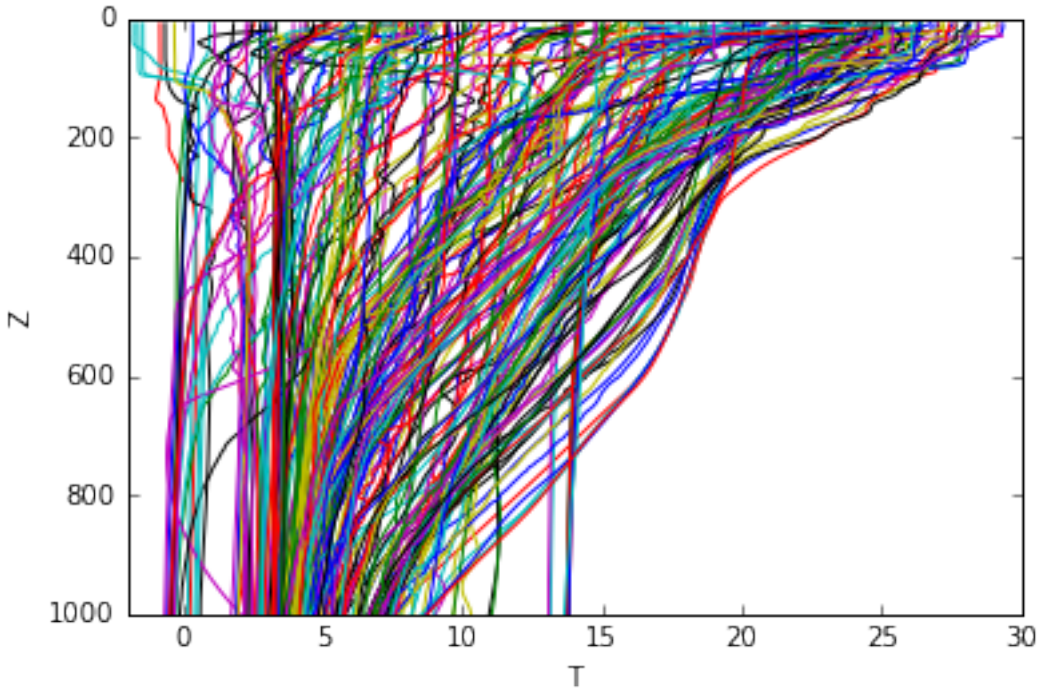
```
    #print(cantdoit)
    # Close data file
    fid.close()

#fid=open('lat_lon_T.tab','w')
#for k in D.keys():# write position (lat,lon), surface temperature to file
#    fid.write(str(k[0])+'\t'+str(k[1])+'\t'+str(D[k][0][0])+'\n')
#fid.close()


# Draw map of positions
fig=figure(figsize=(10,10))
m = Basemap(projection='ortho',lon_0=0,lat_0=0,resolution='l')

for lon,lat in D.keys():
    x,y=m(lon,lat) # Coordinate transfer
    m.plot(x,y,'r.')
m.fillcontinents()
m.drawcoastlines()
m.drawmeridians(arange(0, 360, 30))
m.drawparallels(arange(-90, 90, 30))
show()
savefig('argo_position.png',dpi=100)

# Plot profiles
figure()
for k in D.keys():
#    print k
    plot(D[k][0][:],D[k][2][:])
axis([-2,30,1000,0])
xlabel('T')
ylabel('Z')

savefig('T_plot.png',dpi=75)

# Plot profiles
figure()
for k in D.keys():
#    print k
    plot(D[k][1][:],D[k][2][:])
axis([32,40,1000,0])
xlabel('S')
ylabel('Z')

savefig('S_plot.png',dpi=75)

N=len(D.keys())
print(N)
DATA=zeros((200,2,N))
for i,k in enumerate(D.keys()):

    DATA[:,0,i]=D[k][0][:] # T
    DATA[:,1,i]=D[k][1][:] # S
```

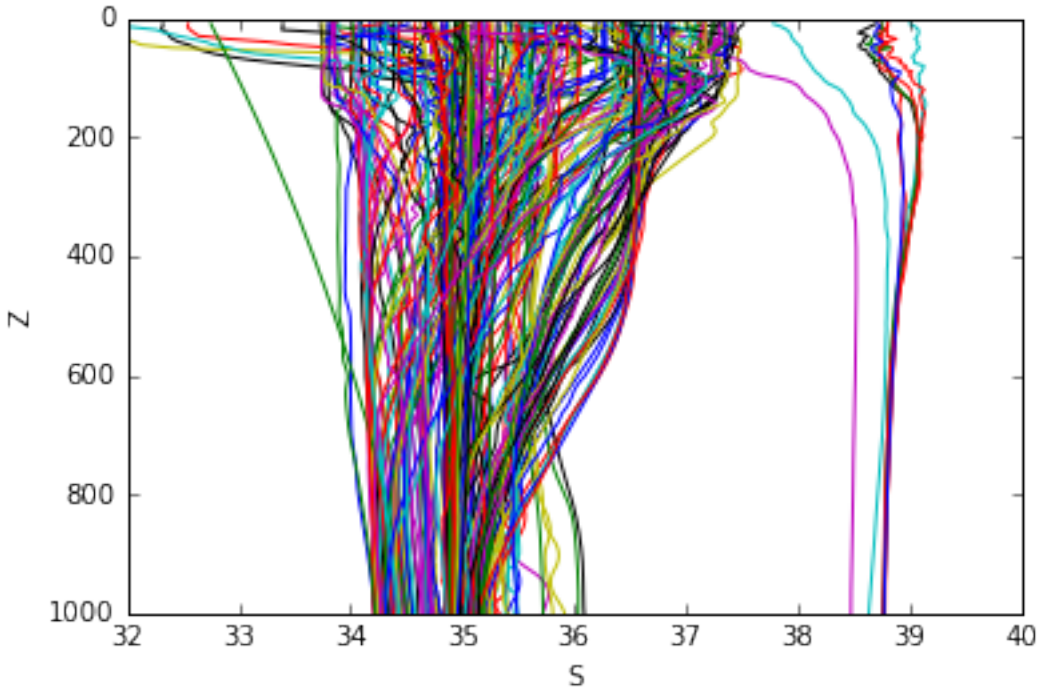
```
DATA.tofile('T_S_profiles.dat',sep=',')
```

```
Populating the interactive namespace from numpy and matplotlib

WARNING: pylab import has clobbered these variables: ['info', 'f', 'fft', 'power', 'random', 'linalg',
'%matplotlib' prevents importing * from pylab and numpy
/usr/local/lib/python3.4/dist-packages/numpy/lib/nanfunctions.py:326: RuntimeWarning: All-NaN slice enc
  warnings.warn("All-NaN slice encountered", RuntimeWarning)
```



```
236

<matplotlib.figure.Figure at 0x7f80d6237630>
```


0
200
400
600
800
1000
Z
0
5
10
15
20
25
30
T


0
200
400
600
800
1000
Z
32
33
34
35
36
37
38
39
40
S

67 EOF-Muster: Klimaindizes

- http://www.cpc.ncep.noaa.gov/products/precip/CWlink/daily_ao_index/loading.html

```
In [4]: %pylab inline
        import pandas as pd
        from datetime import datetime, timedelta
        from dateutil.parser import parse

        from mpl_toolkits.basemap import Basemap, addcyclic
        import gzip,os
        import scipy.stats as stats
        import scipy.linalg as la
        import scipy.io as io
```

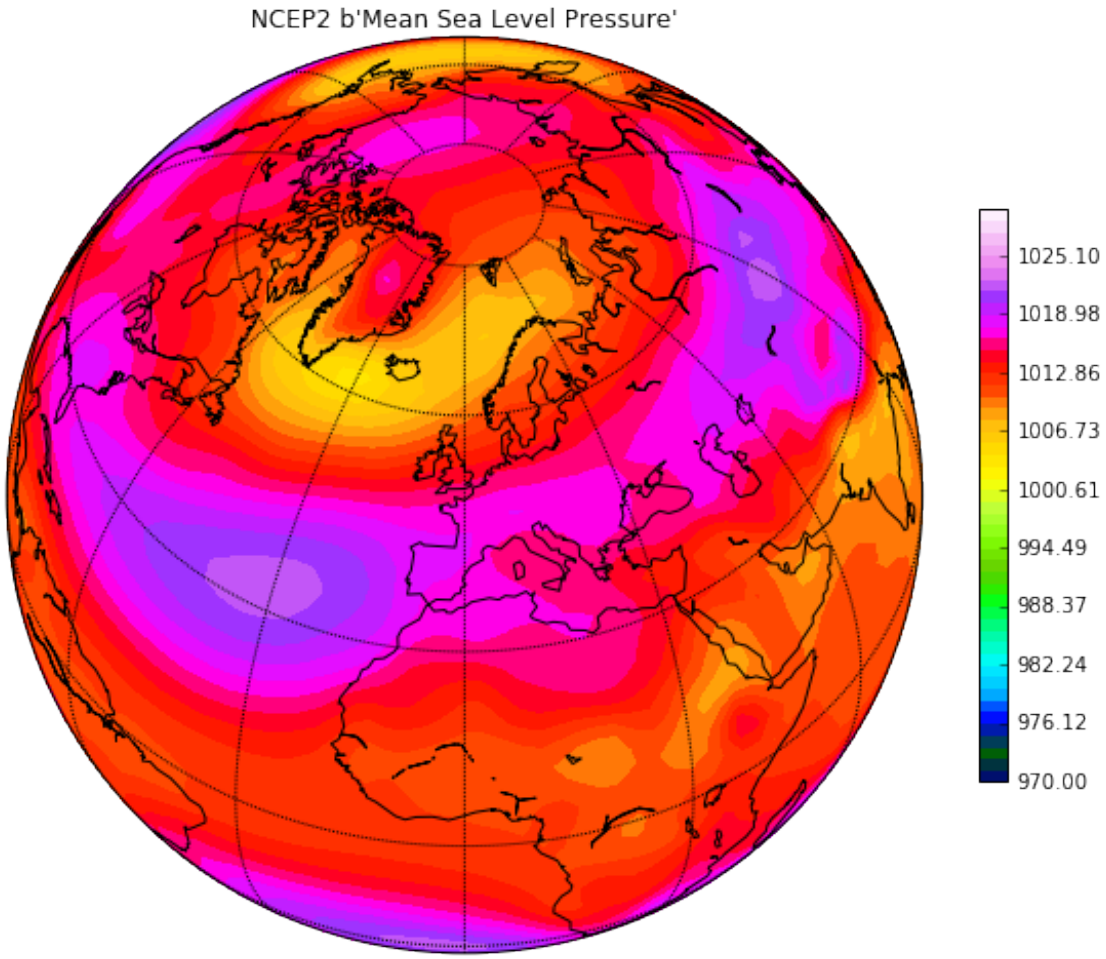
```
Populating the interactive namespace from numpy and matplotlib

WARNING: pylab import has clobbered these variables: ['datetime']
`%matplotlib` prevents importing * from pylab and numpy
```

```
In [1]: #!wget ftp://ftp.cdc.noaa.gov/Datasets/ncep.reanalysis2.derived/surface/mslp.mon.mean.nc
```

```
In [19]: fid=io.netcdf_file('mslp.mon.mean.nc','r') # monthly values for 1979/01/01 to 2013/12/31
         print(fid.variables)
         mslp=fid.variables['mslp']
         lat=fid.variables['lat'][:]
         lon=fid.variables['lon'][:]
         time=fid.variables['time']

         P=mslp.data*mslp.scale_factor+mslp.add_offset
         print(mslp.units)

         P0=mean(P[:,:,:],axis=0)/100.0 # Mean Sea Level in hPa
         Pc, lonc = addcyclic(P0, lon)

         fig=figure(figsize=(10,10))
         m= Basemap(resolution='c',projection='ortho',lon_0=0,lat_0=50)

         x,y=m(*meshgrid(lonc,lat))
         ps=linspace(970,1030)

         m.contourf(x,y,Pc,ps,cmap=cm.gist_ncar)
         m.drawcoastlines()
         m.drawmeridians(arange(0, 360, 30))
         m.drawparallels(arange(-90, 90, 30))
         title('NCEP2 '+str(mslp.var_desc))
         colorbar(shrink=0.5)
```

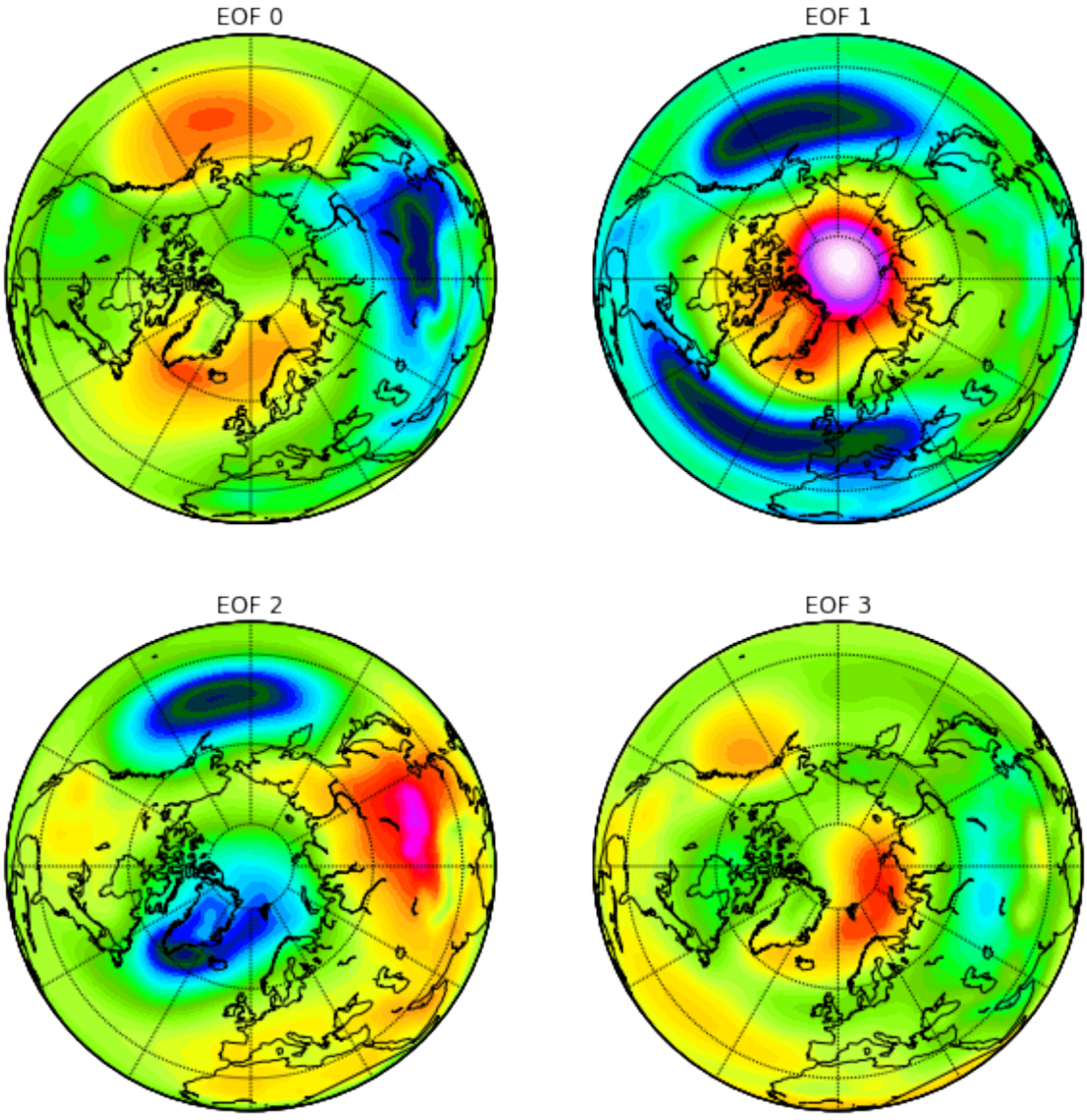
```
{'mslp': <scipy.io.netcdf.netcdf_variable object at 0x7fcec80da4a8>, 'lon': <scipy.io.netcdf.netcdf_vari
b'Pascals'
```

```
Out[19]: <matplotlib.colorbar.Colorbar at 0x7fced09e3d30>
```



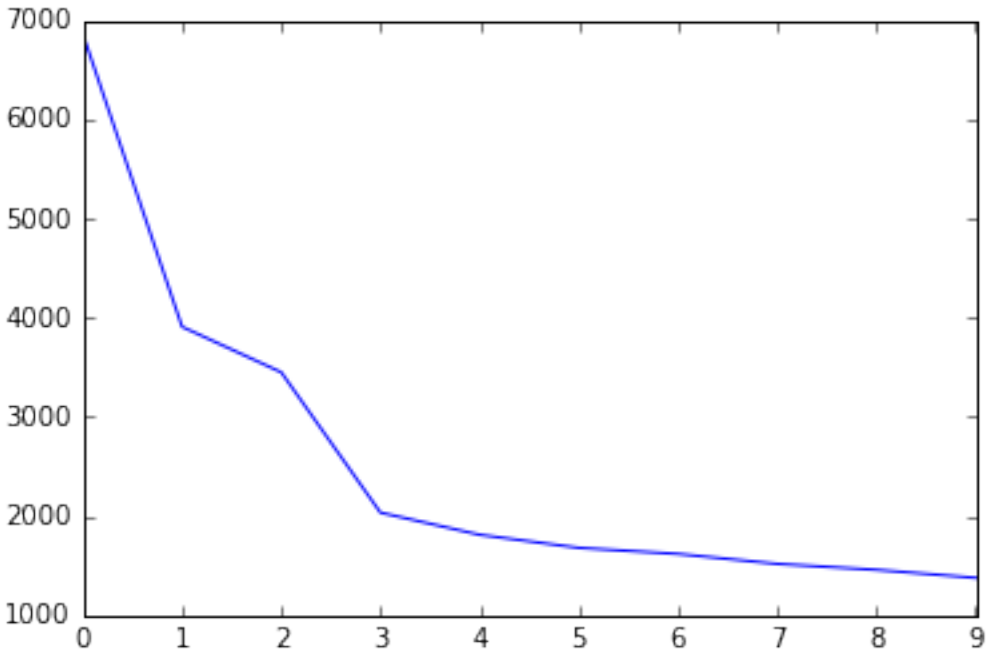

```
In [12]: P_anom.shape

Out[12]: (439, 73, 144)

In [13]: P_anom=P/100-P0

         D=reshape(P_anom,(439,73*144))

         U, s, V = svd(D, full_matrices=False)

         EOF=reshape(V[0,:],(73,144))

In [17]: EOF=reshape(V[0,:],(73,144))
         EOF.max()

Out[17]: 0.036911882

In [20]: fig=figure(figsize=(10,10))
         for k in range(4):
             subplot(2,2,k+1)
```

```
    EOF=reshape(V[k,:],(73,144))
    Pc, lonc = addcyclic(EOF, lon)

    m= Basemap(resolution='c',projection='ortho',lon_0=0,lat_0=90)

    x,y=m(*meshgrid(lonc,lat))
    ps=linspace(EOF.min(),EOF.max())

    m.contourf(x,y,Pc,ps,cmap=cm.gist_ncar)
    m.drawcoastlines()
    m.drawmeridians(arange(0, 360, 30))
    m.drawparallels(arange(-90, 90, 30))
    title('EOF '+str(k))

#colorbar(shrink=0.5)
```



```
In [28]: plot(s[0:10]) # Singulärwerte
```

```
Out[28]: [<matplotlib.lines.Line2D at 0x7fced0dce128>]
```




```
In [ ]:
```

68 Interpolation

Ein übliches Problem der Zeitserien- bzw. Signalanalyse ist der Übergang vom diskreten Signal zu einer kontinuierlichen Beschreibung mittels analytischer Funktionen. Dieser Übergang ist im Allgemeinen notwendig, um Werte geeignet zu interpolieren oder Daten zu filtern. Die sogenannte Interpolierende (oder Interpolante) ist eine stetige Funktion die Werte zwischen diskreten Mess- bzw. Abtastpunkten generiert. Wenn eine theoretische analytische Beschreibung für das Signal nicht existiert, dann besteht das Interpolationsproblem darin, die Funktion zu bestimmen, die einen optimalen Fit durch die Daten erzeugt.

68.1 Polynominterpolation

https://de.wikipedia.org/wiki/Polynominterpolation

Für $n+1$ gegebene Wertepaare $(x_i,\ f_i)$ mit paarweise verschiedenen Stützstellen x_i wird ein Polynom P maximal n-ten Grades gesucht, das alle Gleichungen

$$P(x_i) = f_i, \quad i = 0, \ldots, n$$

erfüllt.

68.2 Beispiel lineare Interpolation

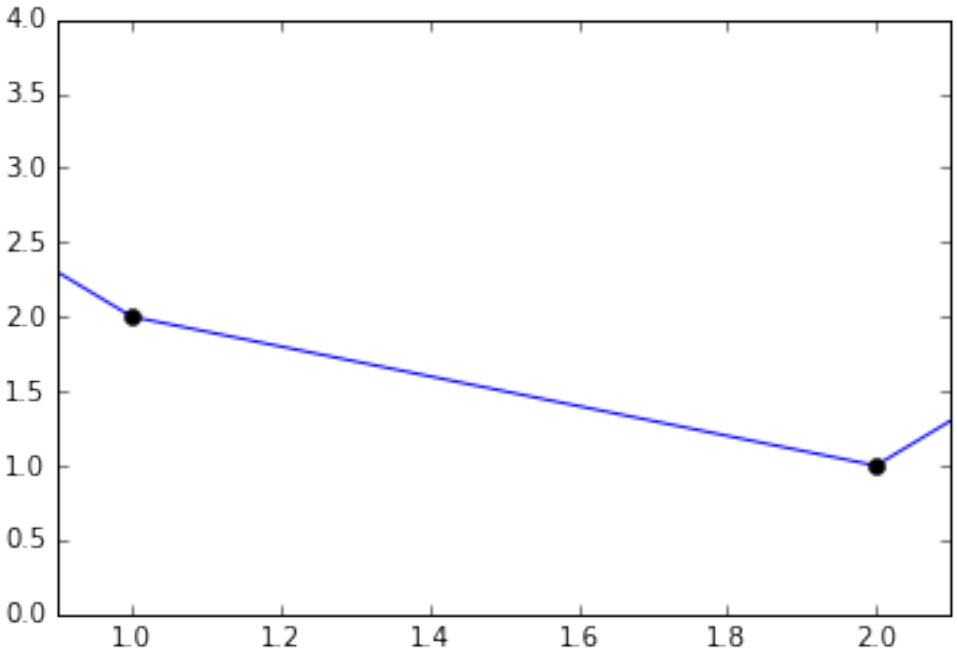
Gegeben sind zwei Beobachtungen y_1(x_1) und y_2(x_2) an unterschiedlichen Orten x_1 und x_2. Wir benötigen eine Schätzung der Beobachtungsvariable an einem Ort r dazwischen. Ein erster Ansatz ist die lineare Interpolation:

$$y(r) = y_1(x_1)\frac{|x_2 - r|}{|x_2 - x_1|} + y_2(x_2)\frac{|x_1 - r|}{|x_2 - x_1|}$$

```
In [43]: %pylab inline
         def linear_interpol(x,y,r):
             y_i=y[0]*abs(x[1]-r)/abs(x[1]-x[0])+y[1]*abs(x[0]-r)/abs(x[1]-x[0])
             return y_i

         X=array([1,2])
         Y=array([2,1])
         r=linspace(0,3,1000)
         y_i=linear_interpol(X,Y,r)
         plot(r,y_i)
         plot(X,Y,'ko')
         axis([0.9,2.1,0,4])
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
Out[43]: [0.9, 2.1, 0, 4]
```



68.3 Lagrange-Interpolation

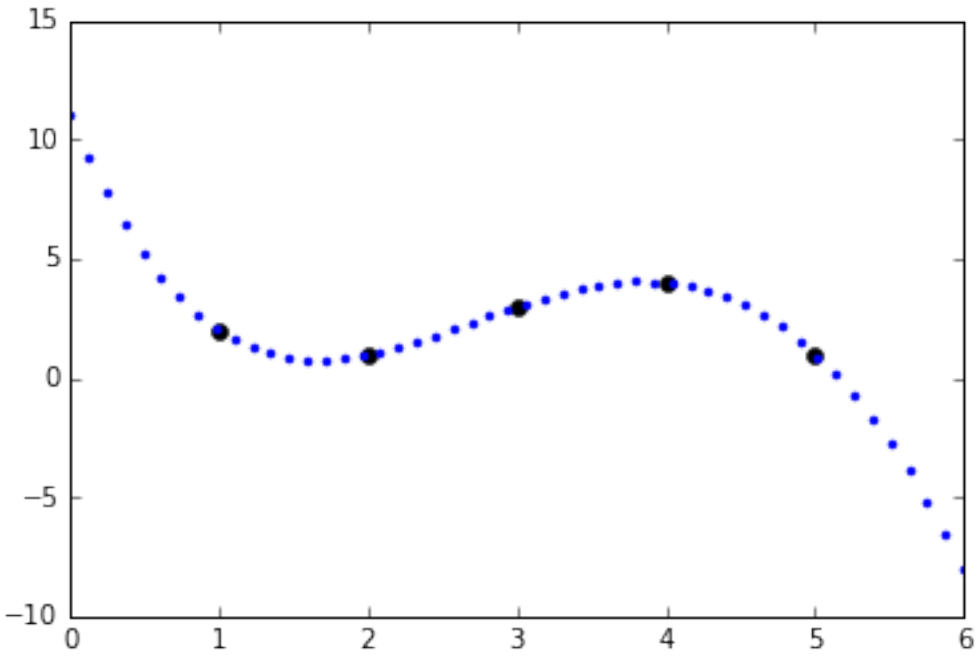
Im Falle von M Datenpunkten kann der Ansatz erweitert werden auf die sogenannte Lagrange-Interpolation mit den Lagrange-Polynomen l_i als Basisfunktionen

$$y(r) = \sum_i^M l_i(r) y_i(x_i)$$

mit

$$l_i = \prod_{j,i\neq j}^{M} \frac{r - x_j}{x_i - x_j}$$

```
In [42]: def lagrange_interpolation(r,X,Y):
             M=len(X)
             f=0
             for i in range(M):
                 P=[]
                 for j in range(M):
                     if i!=j:
                         P.append((r-X[j])/(X[i]-X[j]))
                 L=product(array(P))
                 f=f+L*Y[i]
             return f


         X=array([1,2,3,4,5])
         Y=array([2,1,3,4,1])

         r=linspace(0,6)
         plot(X,Y,'ko')
         for ri in r:
             y=lagrange_interpolation(ri,X,Y)
             plot([ri],[y],'b.')
```



Ein Nachteil der Lagrange-Darstellung ist jedoch, dass alle Basisvektoren bei Hinzunahme einer einzelnen Stützstelle komplett neu berechnet werden müssen, weshalb dieses Verfahren für die meisten praktischen Zwecke zu aufwendig ist.

69 Upsampling und FFT-Interpolation

https://de.wikipedia.org/wiki/Upsampling
https://en.wikipedia.org/wiki/Upsampling

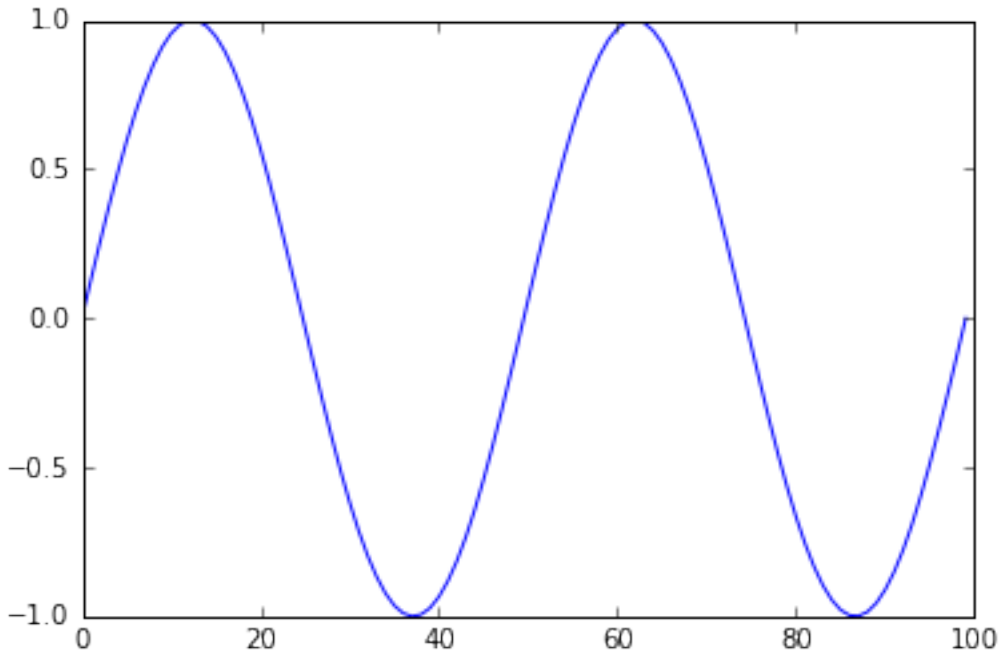
69.1 Beispiel: Verdopplung der Abtastrate

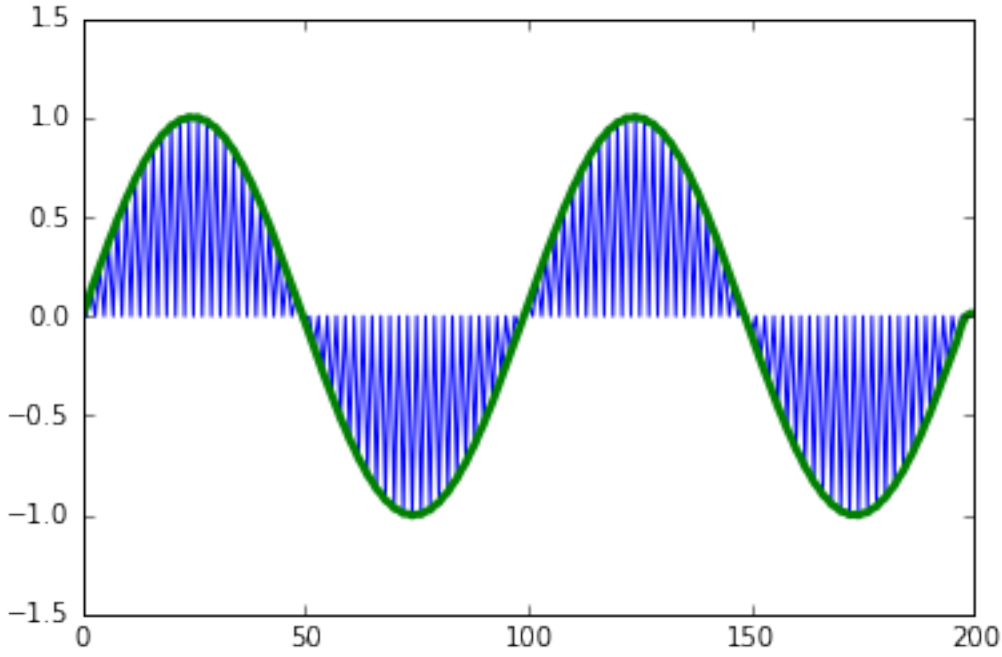
Im folgenden Beispiel verdoppeln wir die Abtastrate durch Einfügen von Nullen hinter jedem Messwert. Mit einem nachträglich angewandten Tiefpaßfilter erhalten wir ein interpoliertes Signal.

```
In [76]: from scipy import signal
         from scipy.fftpack import fft, fftshift

         numtaps=15
         window=signal.firwin(numtaps,cutoff = 0.5, pass_zero=True)

         #signal.convolve

         N=100
         m=N*2
         x=linspace(0,4*pi,N)
         y=sin(x)
         Y=zeros(m)
         Y[::2]=y # Jedes zweite Element wird aus Array y übernommen, der Rest bleibt Null
         figure()
         plot(y)
         figure()
         plot(Y)

         Y_tiefpass=signal.convolve(Y,window,mode='same')
         plot(Y_tiefpass*2,'g-',lw=3)
```

```
/usr/local/lib/python3.4/dist-packages/numpy/core/fromnumeric.py:2645: VisibleDeprecationWarning: 'rank
  VisibleDeprecationWarning)
```

```
Out[76]: [<matplotlib.lines.Line2D at 0x7f002b0490f0>]
```





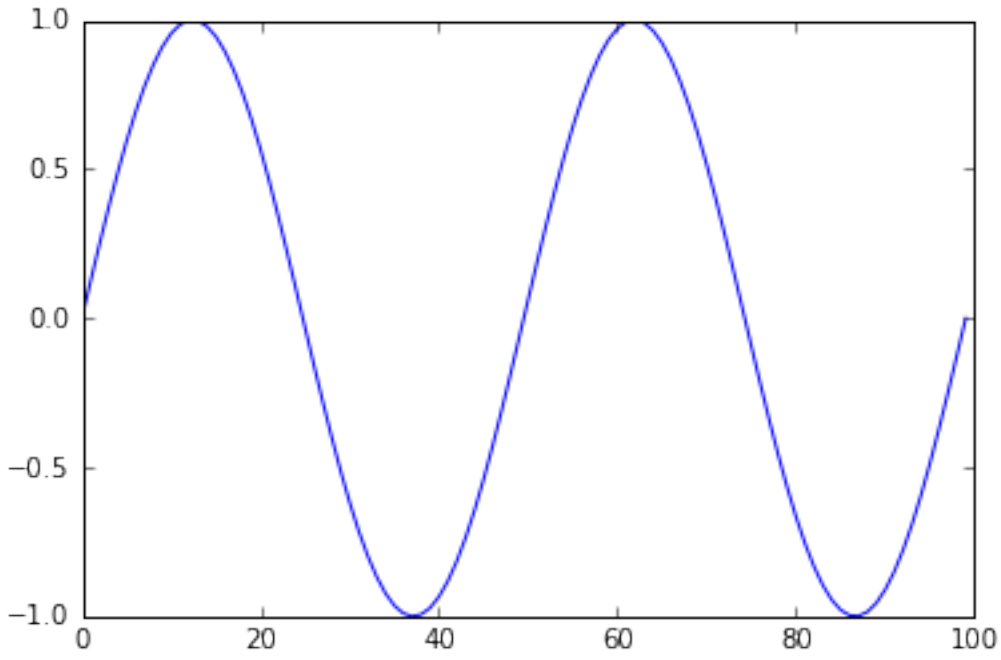
69.2 FFT-Interpolation

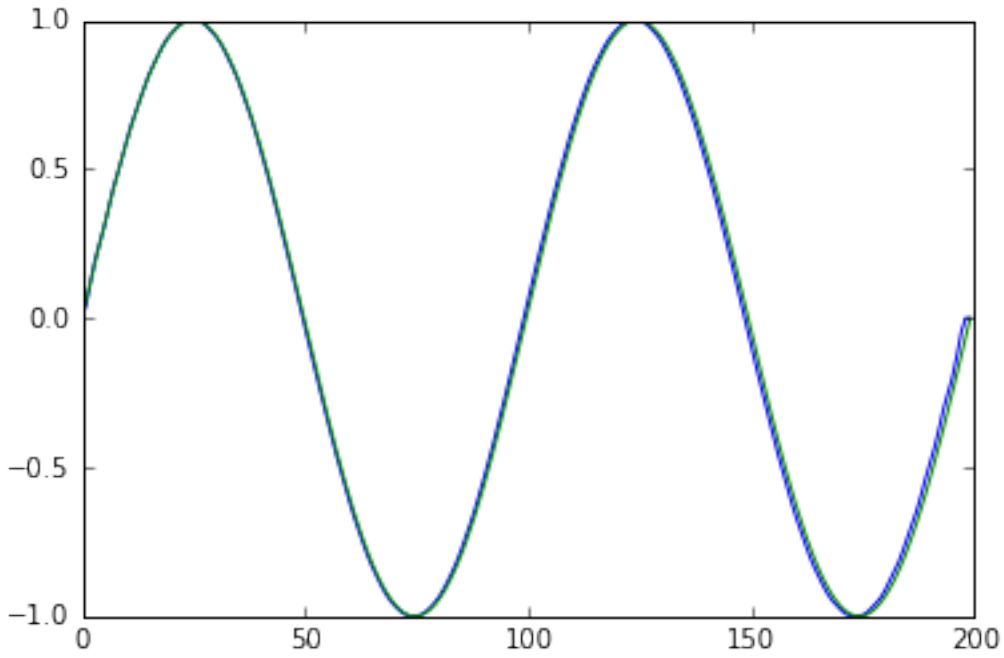
Das Einfügen von Nullen kann man auch im Spektrum durchführen. Die Rücktransformation (Inverse FFT) erzeugt ein interpoliertes Signal. Das Einfügen von Nullen wird als “Zero padding” bezeichnet.

http://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.fft.irfft.html
http://dspguru.com/dsp/howtos/how-to-interpolate-in-time-domain-by-zero-padding-in-frequency-domain

In [51]:
```
N=100
m=N*2
x=linspace(0,4*pi,N)
a=sin(x)
a=a#+randn(N)*0.3
X=linspace(0,4*pi,m)
A=sin(X)

a_resamp = irfft(rfft(a), m)
figure()
plot(a)
figure()
plot(a_resamp*2)
plot(A)
```

Out[51]: [<matplotlib.lines.Line2D at 0x7f002fc85dd8>]





69.3 Interpolation mit Splines

Splines sind stückweise Polynome und besonders zur Interpolation geeignet. Normale Polynome hoher Ordnung (>5) sind üblicherweise instabil und nicht für die Interpolation von vielen Datenpunkten geeignet. Daher wird die Interpolation stückweise mit Polynomen niedriger Ordnung vorgenommen. An den Datenpunkten, auch Knotenpunkte oder Stützstellen genannt, sollen gewisse Bedingungen (Stetigkeit und Differenzierbarkeit) sichergestellt werden. Daraus ergibt sich eine Berechnungsvorschrift für die Polynom-Koeffizienten.

Eine wichtige Funktionengruppe sind die sogenannten B-Splines (Basis-Splines). Die B-Spline-Zerlegung lautet

$$s(x) = \sum_{k \in Z} c(k) \beta^n (x - k)$$

mit den Koeffizienten $c(k)$ und den um k-verschobenen B-Splines β^n der Ordnung n. Die B-Splines ergeben sich aus der rekursiven Faltung eines Rechtecksignals

$$\beta^0(x) = \begin{cases} 1, & -\frac{1}{2} < x < \frac{1}{2} \\ \frac{1}{2}, & x = \frac{1}{2} \\ 0, & sonst \end{cases}$$

$$\beta^n(x) = \underbrace{\beta^0 * \beta^0 * ... * \beta^0(x)}_{\times n+1}$$

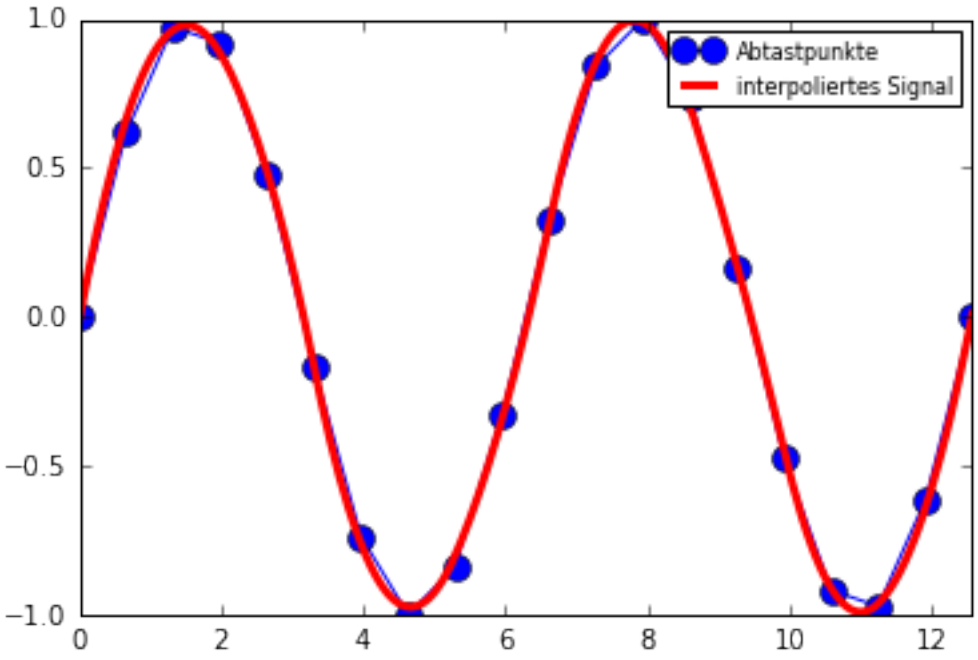
Die Module scipy.interpolate und scipy.ndimage.interpolate stellen Funktionen bereit, um die Spline-Zerlegung von ein- und mehr-dimensionalen Signalen bzw. Bildern zu berechnen. Es sind zwei Schritte notwendig. Im ersten Schritt wird mit der Funktion interpolate.splrep die Spline-Repräsentation (Koeffizienten) berechnet. Die Funktion interpolate.splev evaluiert die Koeffizienten an neuen Stützstellen und liefert so den zweiten Schritt für eine Interpolation.

69.4 References

- Bernd Jähne, Digitale Bildverarbeitung, ZMAW-Bilbliothek, Signatur: DAT-SIG J 2 (4 Exemplare vorhanden)
- Unser, M. (1999), Splines: a perfect fit for signal and image processing, IEEE Signal Processing Magazine, http://bigwww.epfl.ch/publications/unser9902.pdf

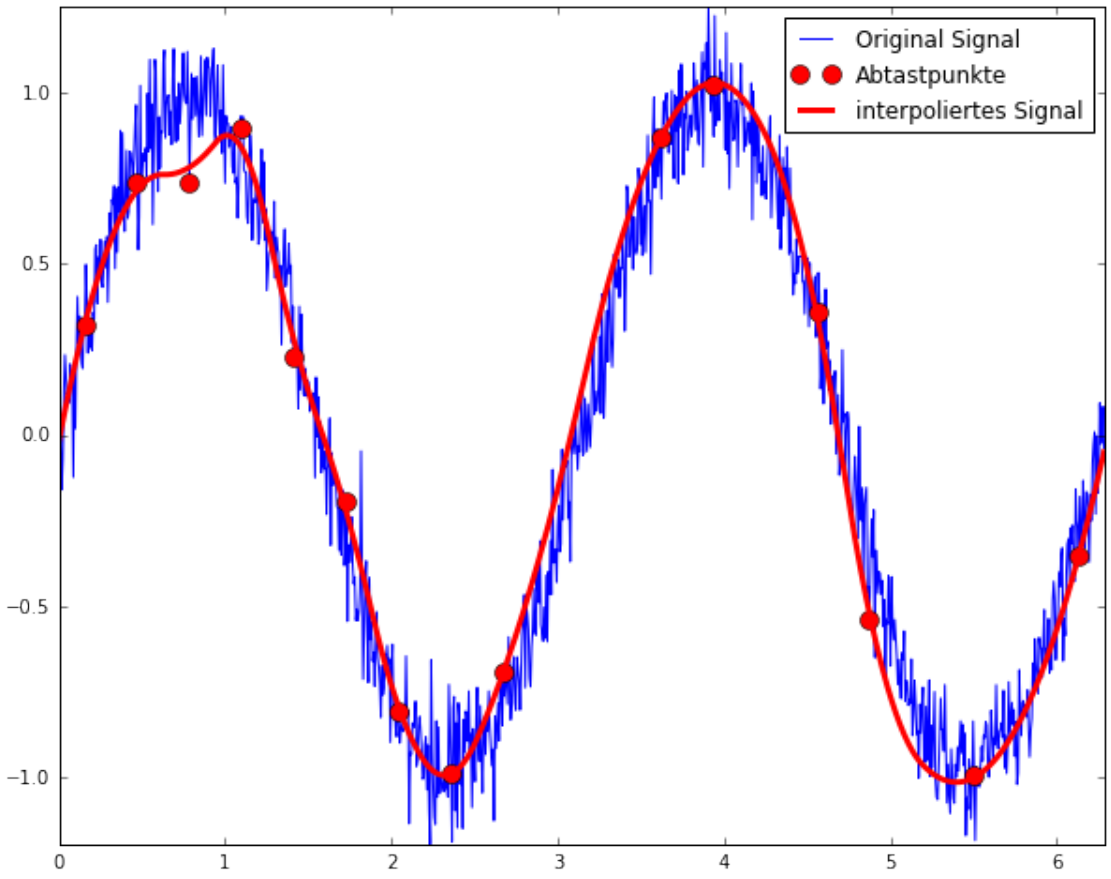
In [3]:
```
%pylab inline
from scipy import interpolate
N=20
m=200
x=linspace(0,4*pi,N)
a=sin(x)
a=a#+randn(N)*0.3
X=linspace(0,4*pi,m)
A=sin(X)


#Spline Interpolation
tck=interpolate.splrep(x,a,k=2,s=0.01) # Berechne Koeffizienten an den Datenpunkten Ordnung k=2
spline_interp=interpolate.splev(X,tck) # Berechne Spline-Interpolation auf feinem Gitter
figure()
plot(x,a,'bo-',markersize=10,label='Abtastpunkte')
plot(X,spline_interp,'r-',linewidth=3,label='interpoliertes Signal')
#title('n='+str(n)+' w='+str(w))
axis('tight')
legend(loc=1,fontsize=8)
show()
```

```
Populating the interactive namespace from numpy and matplotlib
```



69.5 Beispiel: Datenlücken, irreguläre Abtastung

In [4]:
```
N=1000 # Nahezu kontinuierliche Abtastung mit N Punkten (feines Gitter)
n=20 # Anzahl von Diskreten Datenpunkten mit zufälligen Datenlücken
w=2 # Anzahl von Sinusschwingungen im Interval

s=N/n# Skalierung

X=linspace(0,w*pi,N)
Y=sin(1*X*w)+randn(N)*0.1

i=arange(0,n)
x=X[(i*s+s/2).astype(int)]
y=Y[(i*s+s/2).astype(int)]

# Zufällige Datenlücken
y[randint(n,size=5)]=nan
index=isfinite(y)
y=y[index]
x=x[index]

#Spline Interpolation
tck=interpolate.splrep(x,y,k=2,s=0.01) # Berechne Koeffizienten an den Datenpunkten Ordnung k=2
spline_interp=interpolate.splev(X,tck) # Berechne Spline-Interpolation auf feinem Gitter
figure(figsize=(10,8))
plot(X,Y,'b-',label='Original Signal')
plot(x,y,'ro',markersize=10,label='Abtastpunkte')
plot(X,spline_interp,'r-',linewidth=3,label='interpoliertes Signal')
axis('tight')
legend()
show()
```

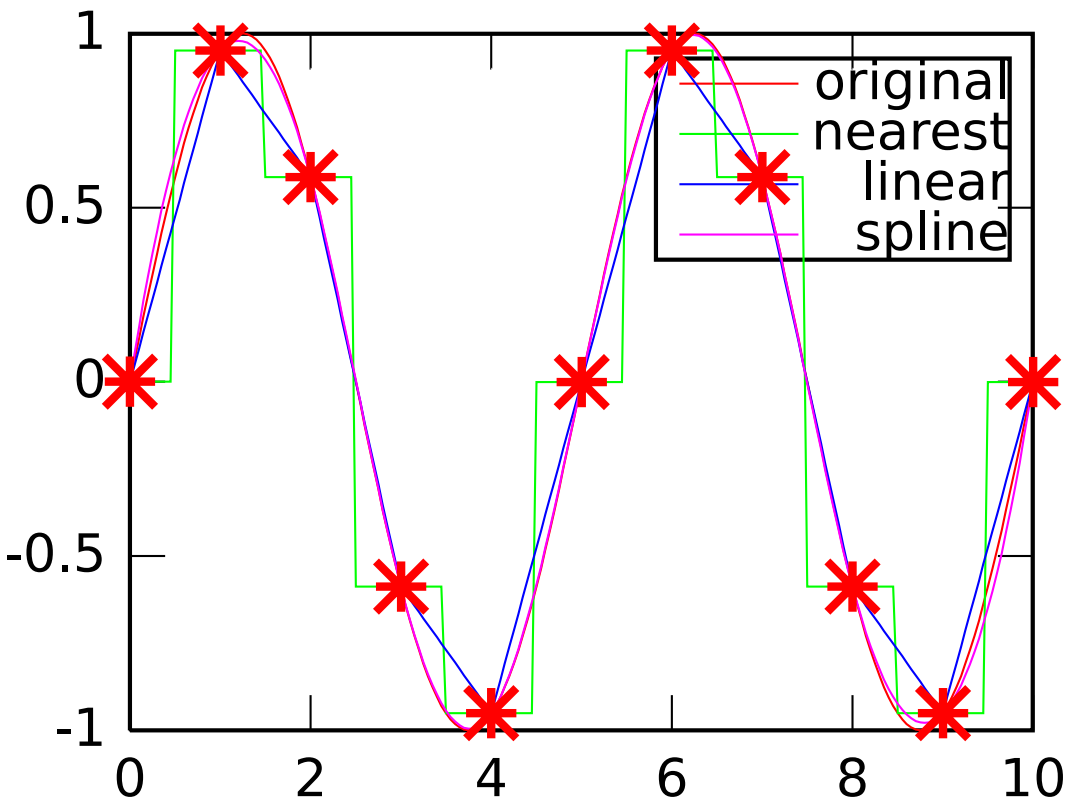


69.6 Octave Beispiele

69.6.1 1D

https://www.gnu.org/software/octave/doc/v4.0.0/One_002ddimensional-Interpolation.html

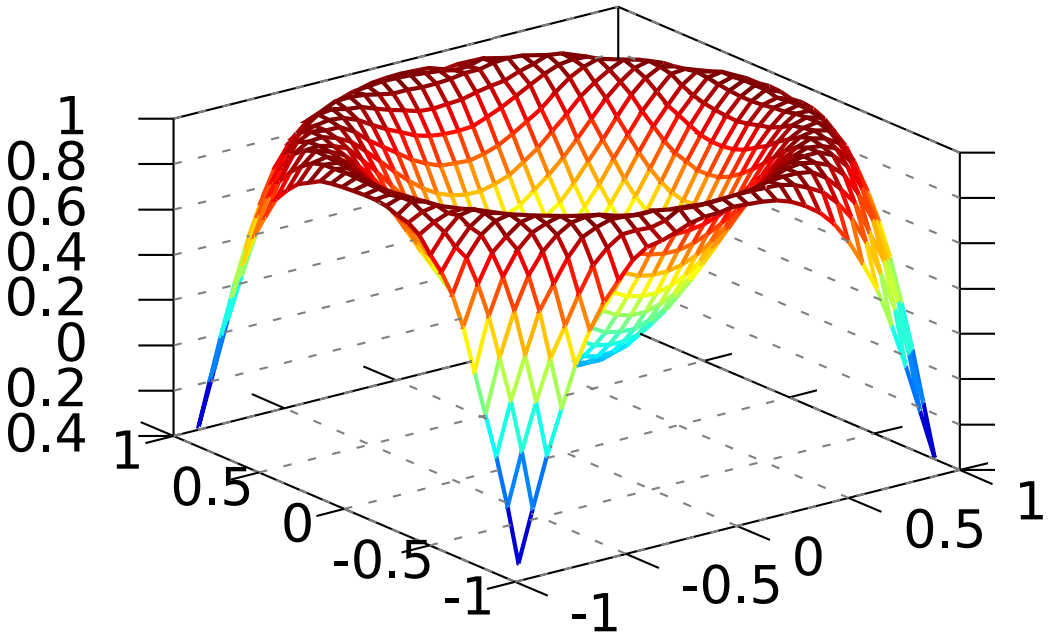
```
In [15]: xf = [0:0.05:10];
         yf = sin (2*pi*xf/5);
         xp = [0:10];
         yp = sin (2*pi*xp/5);
         lin = interp1 (xp, yp, xf);
         near = interp1 (xp, yp, xf, "nearest");
         spl = interp1 (xp, yp, xf, "spline");
         plot (xf,yf,"r", xf,near,"g", xf,lin,"b", xf,spl,"m",xp,yp,"r*");
         legend("original", "nearest", "linear", "spline");
```




```
Out[15]: error: invalid value = northeast
         error: set: invalid value for radio property "location" (value = northeast)
         error: called from:
         error:   /usr/share/octave/3.8.1/m/plot/appearance/legend.m at line 995, column 11
```

69.6.2 2D

```
In [7]: rand ("state", 1);
        x = 2*rand (1000,1) - 1;
        y = 2*rand (size (x)) - 1;
        z = sin (2*(x.^2+y.^2));
        [xx,yy] = meshgrid (linspace (-1,1,32));
        griddata (x,y,z,xx,yy);
```



In [ ]:

70 Korrelationen und Signifikanz

70.1 Korrelationskoeffizient

Der Korrelationskoeffizient $\rho_{X,Y}$ zwischen zwei Zufallsvariablen X und Y mit dem Erwartungswert $E[X] = \mu_X$ und μ_Y und Standardabweichung σ_X und σ_Y ist definiert als

$$\rho_{X,Y} = \frac{\operatorname{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}$$

Dabei bezeichnet $\operatorname{cov}(X,Y)$ die Kovarianz der Variablen X und Y. Die Korrelation ist also die mit der Standardabweichung normierte Kovarianz. Der "normale" Korrelationskoeffizient wird auch als Pearson-Korrelation bezeichnet. Auf den Unterschied zur sogenannten Rangkorrelation (Spearman-Korrelation), gehen wir weiter unten ein.

70.1.1 Beispiele Pearson-Korrelationen

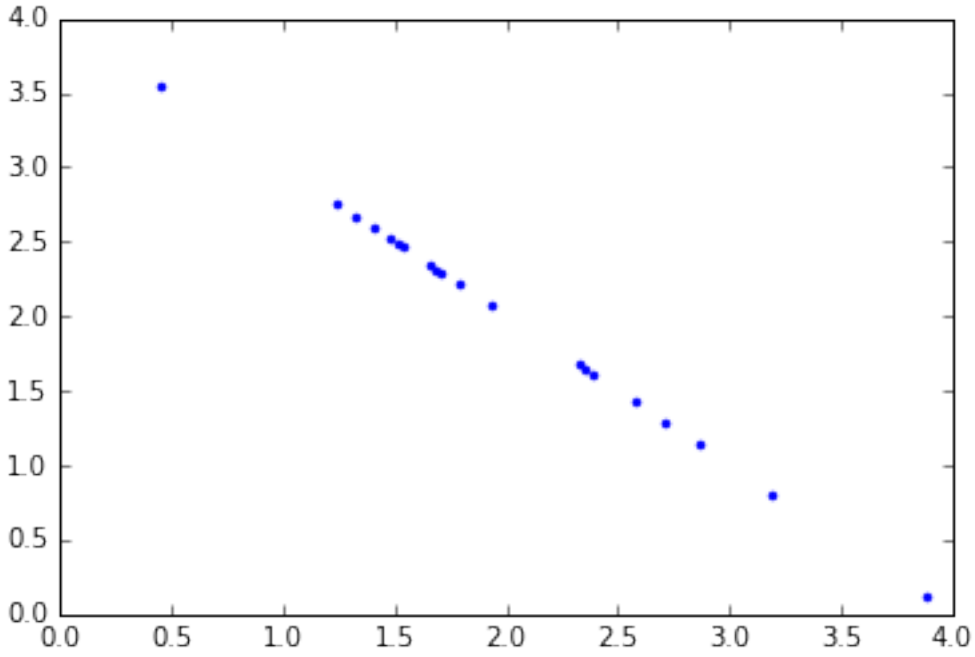
Die folgenden Übungen sind in Python. In Matlab/Octave gibt es die Funktion mvnrnd(mu,Sigma,n), die der Funktionalität der Numpy-Funktion multivariate_normal entspricht. Die Matlab/Octave-Funktion corrcoef berechnet Korrelationskoeffizienten verschiedener Art.

```
In [7]: %pylab inline
        import scipy.stats as st
        import numpy.random
```

```
Populating the interactive namespace from numpy and matplotlib
```

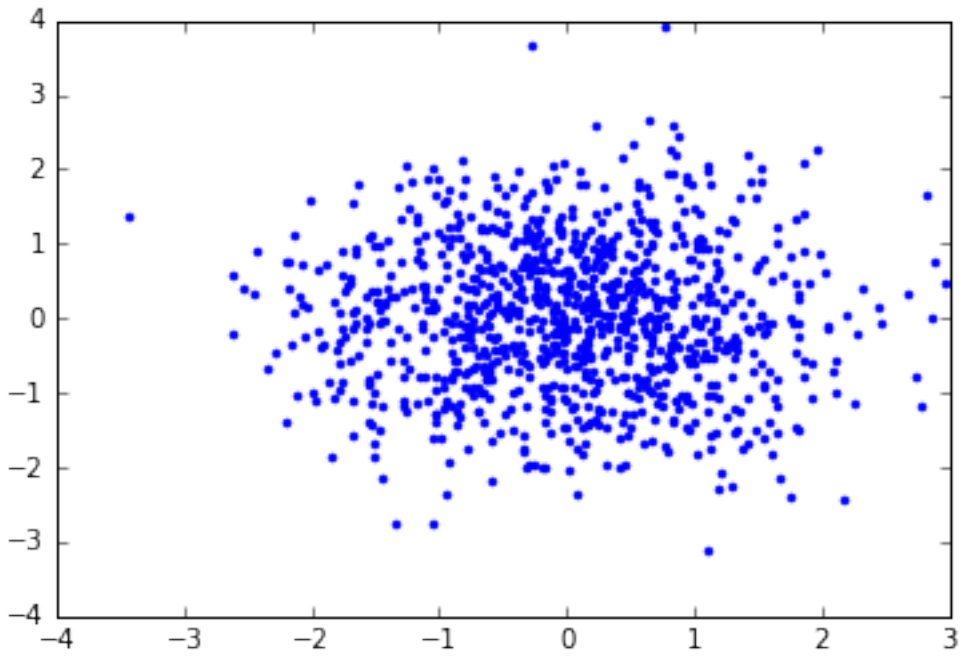
```
In [23]: N=20
         m1=[2, 2] # Mittelwerte
         c1=[[1, -1.0], [-1.0, 1]] # Covarianz
         x1, y1 = np.random.multivariate_normal( m1, c1, N).T
         plot(x1,y1,'.')
         r,p=st.pearsonr(x1,y1)
         print('Pearson r:',r)
```

```
Pearson r: -1.0
```




```
In [18]: N=1000
         m1=[0, 0] # Mittelwerte
         c1=[[1, 0.0], [0.0, 1]] # Covarianz
         x1, y1 = np.random.multivariate_normal( m1, c1, N).T
         plot(x1,y1,'.')
         r,p=st.pearsonr(x1,y1)
         print('Pearson r:',r)
```

```
Pearson r: -0.0179886687098
```



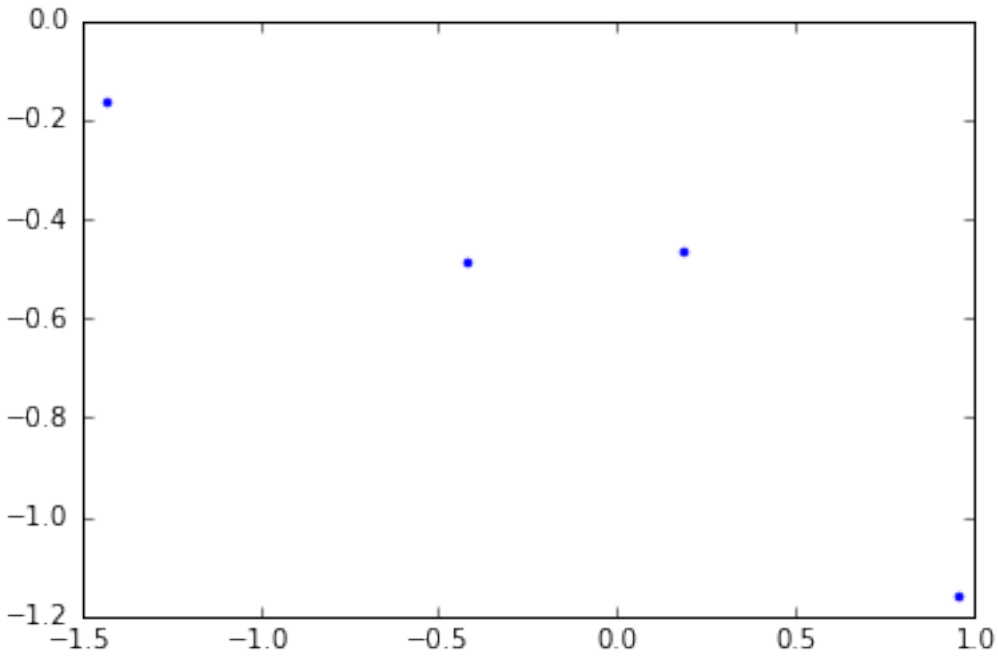
Sind die Daten X und Y unabhängig, so ist der Korrelationskoeffizient null. Der Umkehrschluss ist aber nicht immer gültig.

70.2 Fischer-Transformation für Signifikanztest

Ein übliches wissenschaftliches Problem ist die Bestimmung der Signifikanz einer gegebenen Korrelation, um die Hypothese eines Zusammenhangs zwischen zwei Variablen zu überprüfen.

```
In [29]: N=4
         m1=[0, 0] # Mittelwerte
         c1=[[1, 0.0], [0.0, 1]] # Covarianz
         x1, y1 = np.random.multivariate_normal( m1, c1, N).T
         plot(x1,y1,'.')
         r,p=st.pearsonr(x1,y1)
         print('Pearson r:',r)
```

```
Pearson r: -0.911666720126
```



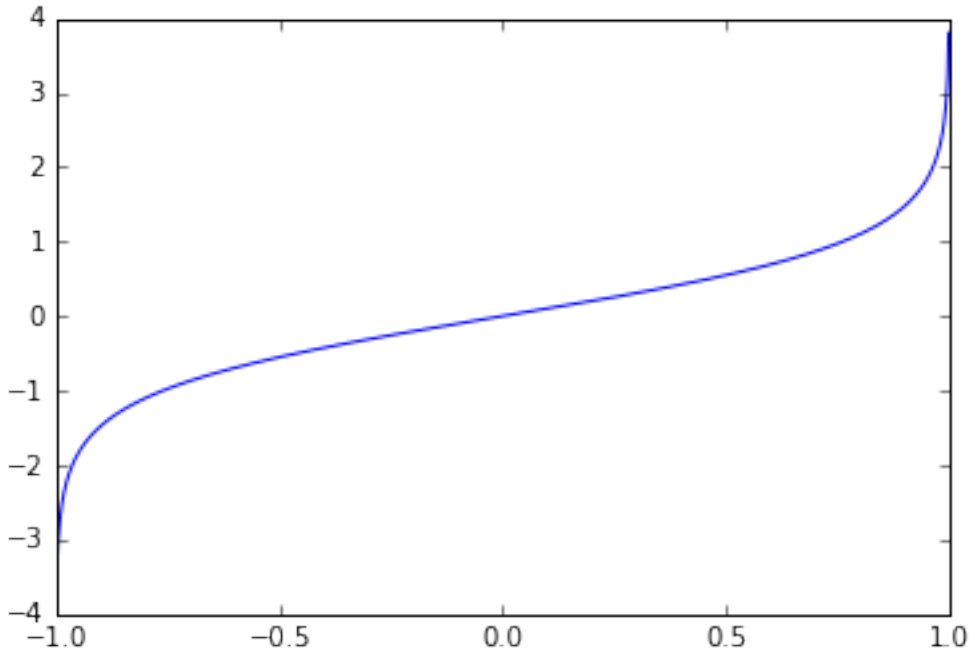
Im Beispiel oben beträgt die Korrelation -0.91, ein Wert vom Betrag her nahe bei eins. Ist dieser Wert signifikant? Das Ergebnis ist rein zufällig aufgetreten. Die Signifikanz hängt offensichtlich von der Anzahl der Realisationen N ab.

Um die Signifikanz zu testen führen wir eine besondere Transformation ein, die sogenannte z-Transformation nach Fischer

$$z = \frac{1}{2} \ln \left(\frac{1+\rho}{1-\rho} \right) = \operatorname{arctanh}(r)$$

```
In [41]: r=linspace(-0.999,0.999,1000)
         z=arctanh(r)
         plot(r,z)
```

```
Out[41]: [<matplotlib.lines.Line2D at 0x7f4410df2748>]
```

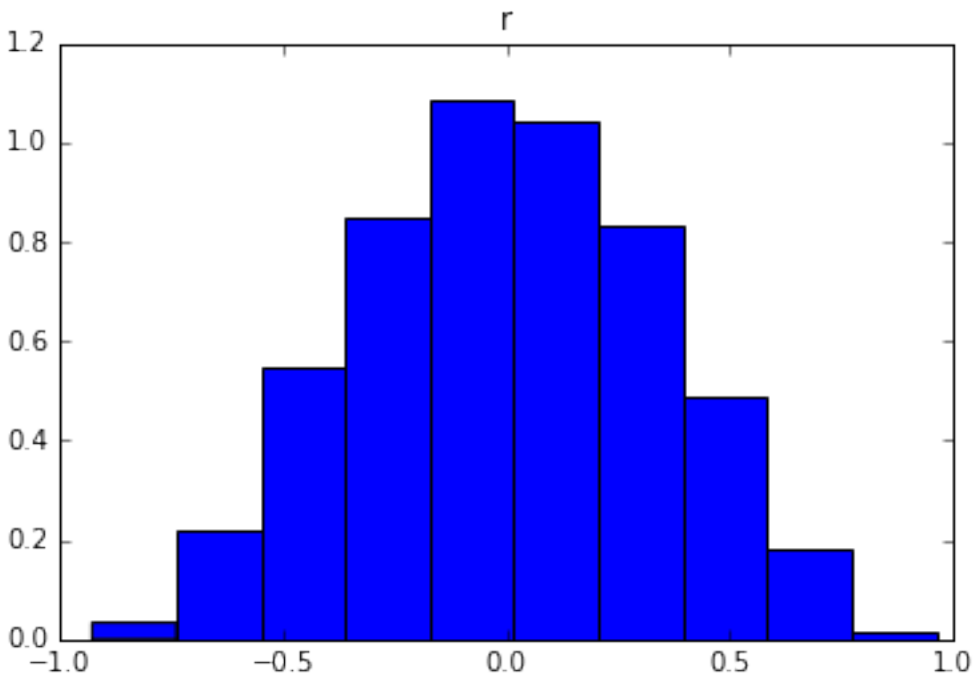


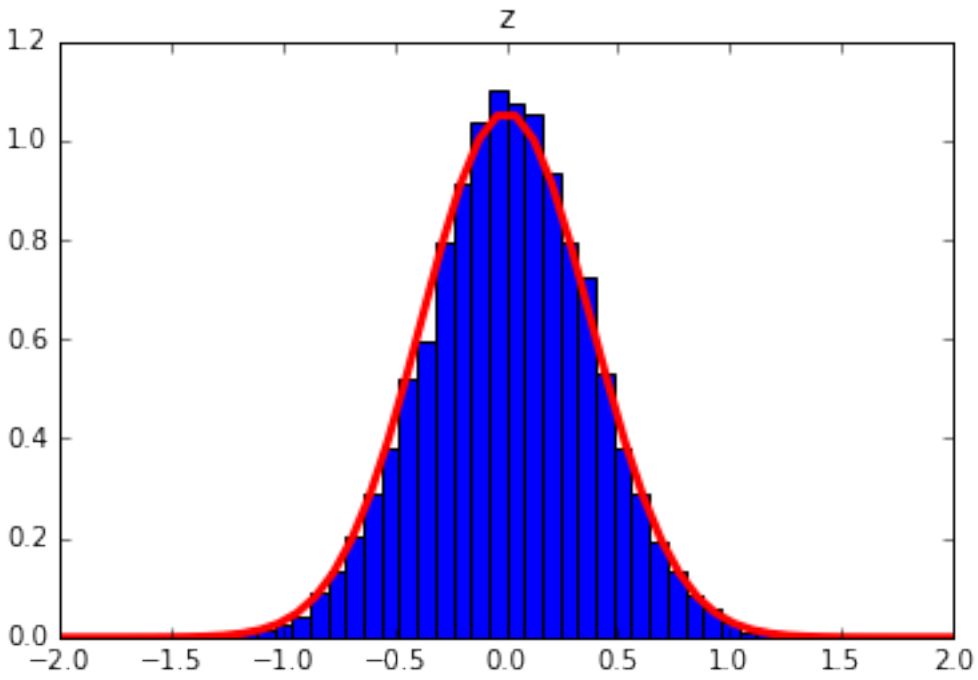
Wenn X und Y unabhängig und bivariat normalverteilt sind, so ist z nahezu (für $N > 10$) normalverteilt mit dem Standardfehler

$$\frac{1}{\sqrt{N-3}}$$

In [114]:
```
# Monte-Carlo Test
M=10000 # Wiederhole M mal und berechne Korrelationskoeffizienten
N=10
m1=[0, 0] # Mittelwerte
c1=[[1, 0.0], [0.0, 1]] # Kovarianz
R=[]
for i in range(M):
    x1, y1 = np.random.multivariate_normal( m1, c1, N).T
    r,p=st.pearsonr(x1,y1)
    R.append(r)
R=array(R)

figure()
h=hist(R,normed=True)
title('r')
Z=arctanh(R)
figure()
title('z')

h=hist(Z,bins=50,range=[-2,2],normed=True)
r=linspace(-2,2)
y=st.norm.pdf(r,0,1.0/sqrt(N-3))
plot(r,y,'r-',linewidth=3)
```

Out[114]:
```
[<matplotlib.lines.Line2D at 0x7f440ef222b0>]
```





Die z-Transformation und ihre Inverse

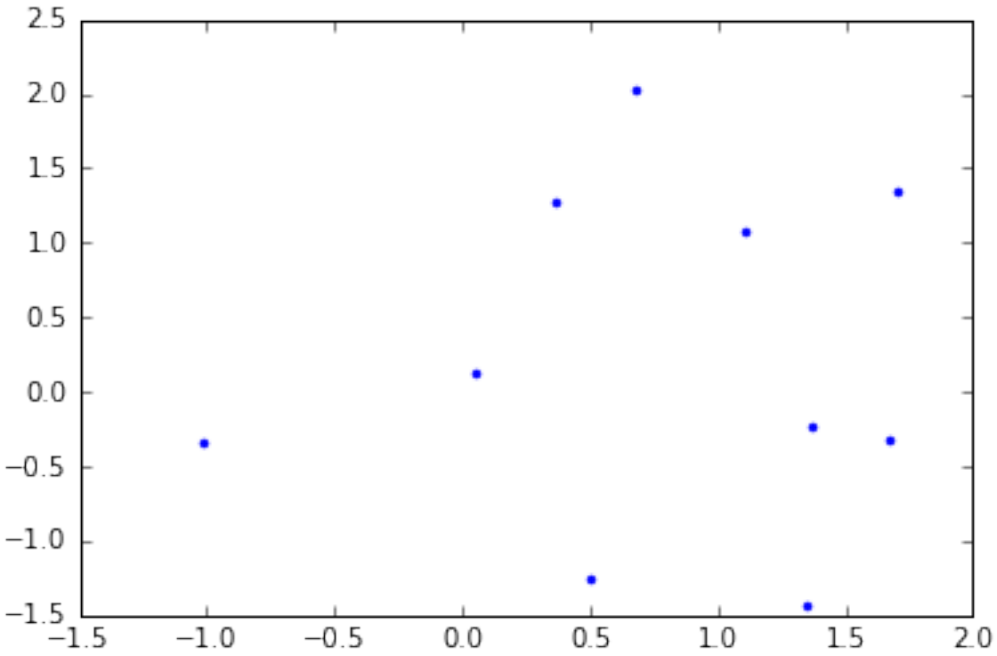
$$r = \frac{\exp(2z) - 1}{\exp(2z) + 1} = \tanh(z)$$

kann nun mittels der bekannten Theorie über Normalverteilungen bzw. mit einem t-Test in Wahrscheinlichkeiten bzw. Signifikanz überführt werden.

70.3 Signifikanztest für Korrelation

In [167]:
```
N=10
m1=[0, 0] # Mittelwerte
c1=[[1, 0.02], [0.02, 1]] # Covarianz
x1, y1 = np.random.multivariate_normal( m1, c1, N).T
plot(x1,y1,'.')
r,p=st.pearsonr(x1,y1)
print('Pearson r:',r,p)
```

```
Pearson r: 0.056988830339 0.875741016876
```



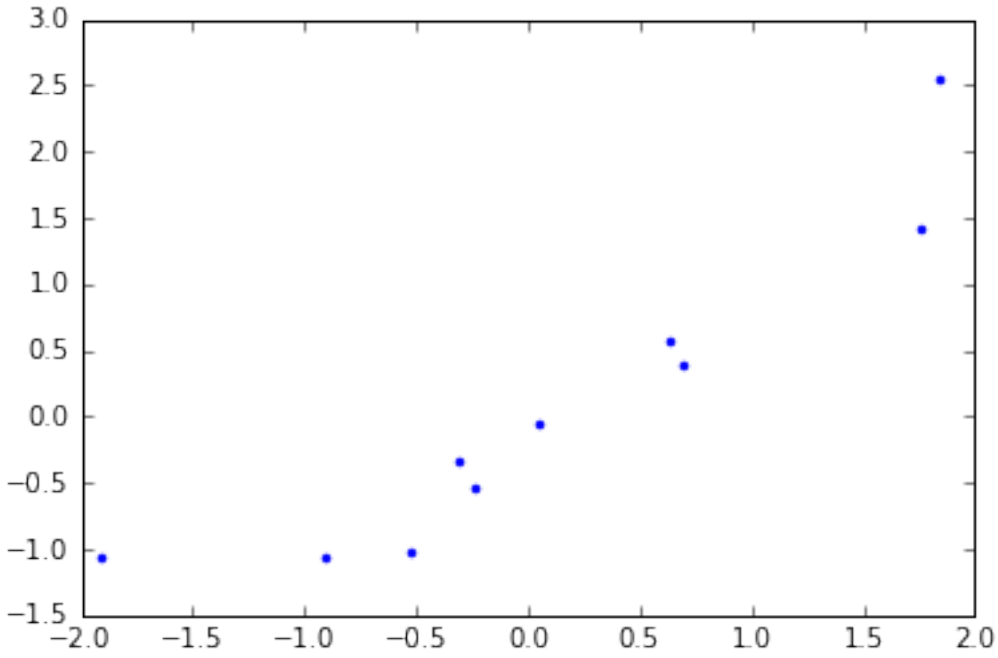
In [169]:
```
t=abs(r)*sqrt( (N-2)/(1-r**2))
p=2*st.t.cdf( -t,N-2 )
print(p)
```

```
0.875741016876
```

Der Wert r=0.05 (N=10) besagt, dass es wahrscheinlich bis sehr wahrscheinlich ist (p=0.87), dass die Daten unabhängig sind.

In [175]:
```
N=10
m1=[0, 0] # Mittelwerte
c1=[[1, 0.9], [0.9, 1]] # Covarianz
x1, y1 = np.random.multivariate_normal( m1, c1, N).T
plot(x1,y1,'.')
r,p=st.pearsonr(x1,y1)
print('Pearson r:',r,p)
```

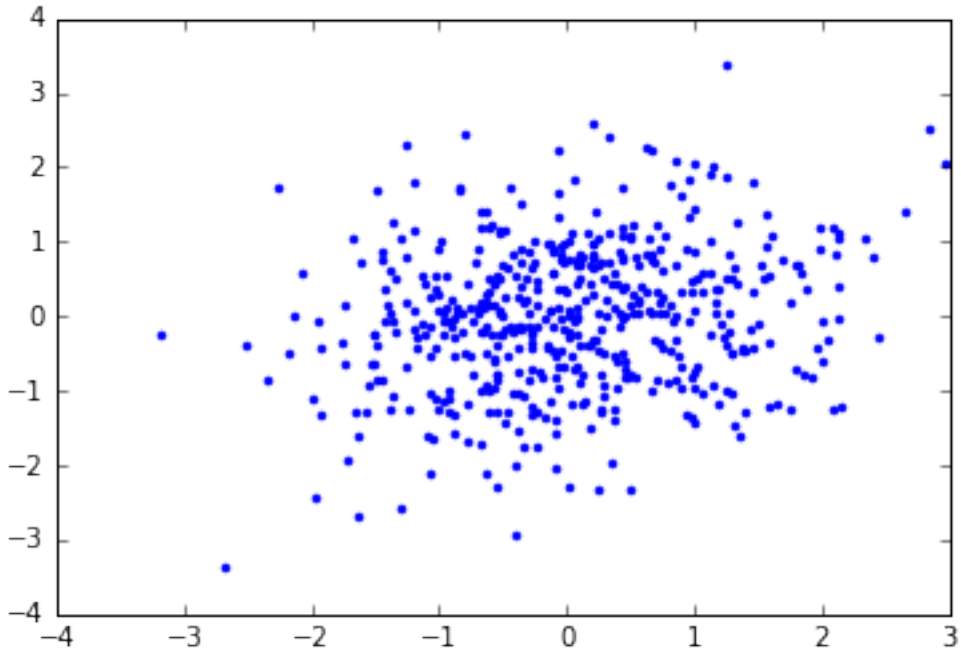
```
Pearson r: 0.926195060408 0.000118666253853
```



Der Wert r=0.92 (N=10) besagt, dass es praktisch sicher ist (p=0.001), dass die Daten abhängig sind.

```
In [180]: N=500
          m1=[0, 0] # Mittelwerte
          c1=[[1, 0.1], [0.1, 1]] # Covarianz
          x1, y1 = np.random.multivariate_normal( m1, c1, N).T
          plot(x1,y1,'.')
          r,p=st.pearsonr(x1,y1)
          print('Pearson r:',r,p)
```

```
Pearson r: 0.201209343088 5.77750469266e-06
```



Der Wert r=0.2 (N=500) besagt, dass es praktisch sicher ist (p=0.001), dass die Daten abhängig sind.

71 Rangkorrelationskoeffizient (Spearman)

71.1 Nichtlineare Zusammenhänge und Ausreißer

Ausreißer in den Daten oder nichtlineare Zusammenhänge beeinflussen den Korrelationskoeffizient. Der Rangkorrelationskoeffizient ist robust gegenüber Ausreißern.

Bei der Rangkorrelation werden die Daten in Ränge konvertiert, also der Größe nach geordnet und durchnummeriert, bevor der Korrelationskoeffizient berechnet wird.

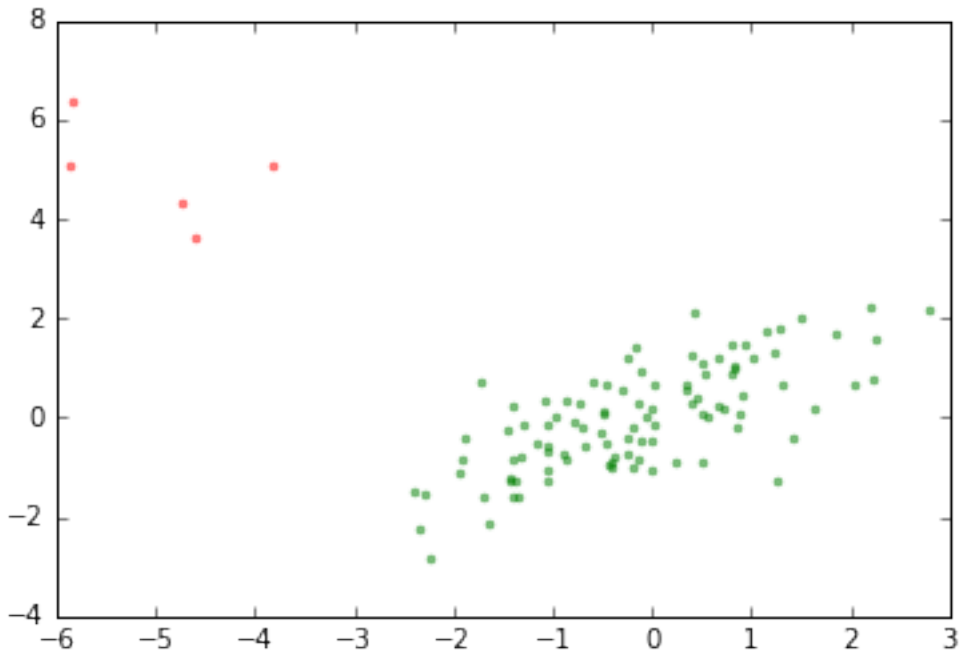
https://de.wikipedia.org/wiki/Rangkorrelationskoeffizient

71.1.1 Beispiel Datenausreißer

```
In [188]: N=10
          m1=[0, 0]
          c1=[[1, 0.7], [0.7, 1]]
          x1, y1 = np.random.multivariate_normal( m1, c1, 100).T

          m1=[-5, 5]
          c1=[[1, -0.5], [-0.5, 1]]
          x2, y2 = np.random.multivariate_normal( m1, c1, 5).T

          x=concatenate((x1,x2))
          y=concatenate((y1,y2))

          plot(x1,y1,'g.',alpha=0.5)
          plot(x2,y2,'r.',alpha=0.5)

          r,p=st.pearsonr(x,y)
```

```
            print('Pearson r,p')
            print(r,p)
            print('Spearman r,p')
            r,p=st.spearmanr(x,y)
            print(r,p)
```

```
Pearson r,p
-0.122371108693 0.213646687614
Spearman r,p
0.45235330707 1.26957197399e-06
```

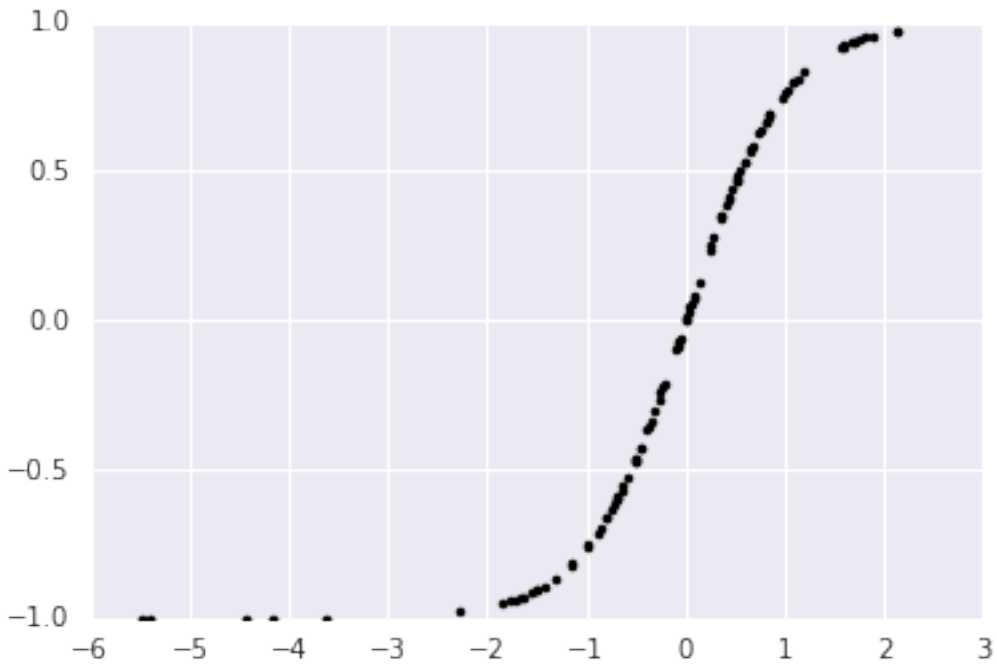


Obiges Beispiel zeigt, der lineare Korrelationskoeffizient (Pearson) zeigt eine nicht eindeutig signifikante Antikorrelation an. Die Rangkorrelation deutet darauf hin, dass ein Zusammenhang praktisch sicher ist.

71.1.2 Beispiel nichtlinearer Zusammenhang

```
In [191]: y=tanh(x)
          plot(x,y,'k.')
          r,p=st.pearsonr(x,y)
          print('Pearson r,p')
          print(r,p)
          print('Spearman r,p')
          r,p=st.spearmanr(x,y)
          print(r,p)
```

```
Pearson r,p
0.873591743256 5.44603325806e-34
Spearman r,p
1.0 0.0
```

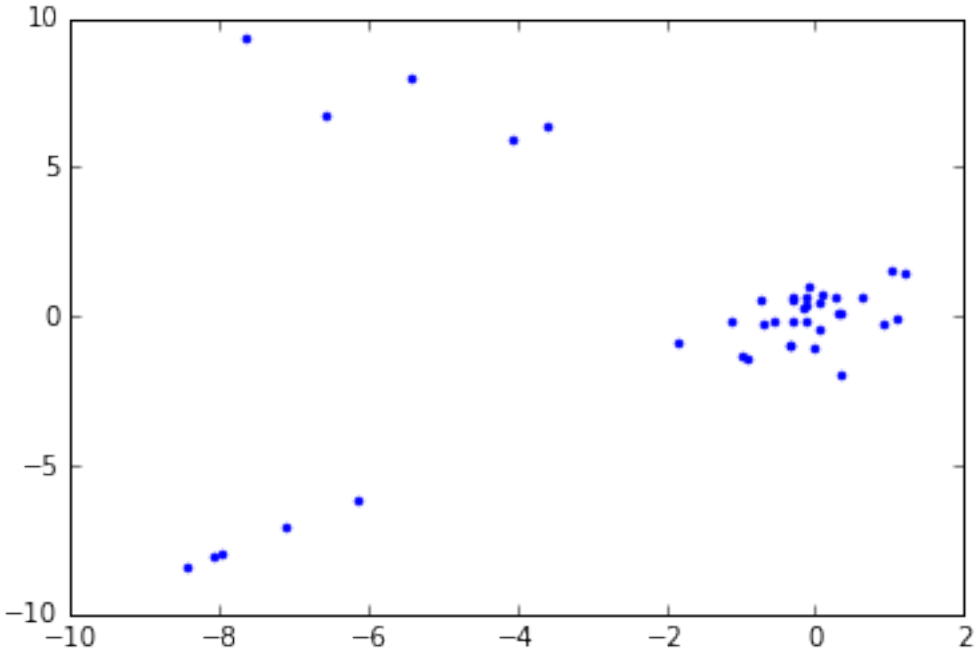


Gibt es einen streng monotonen Zusammenhang zwischen X und Y, so liefert die Spearman-Korrelation den Wert 1.

72 Clusteranalyse

https://de.wikipedia.org/wiki/Hierarchische_Clusteranalyse

```
In [225]: m1=[0, 0]
          c1=[[1, 0.7], [0.7, 1]]
          x1, y1 = np.random.multivariate_normal( m1, c1, 30).T


          m1=[-5, 7]
          c1=[[1, -0.5], [-0.5, 1]]
          x2, y2 = np.random.multivariate_normal( m1, c1, 5).T

          m1=[-8, -8]
          c1=[[1, -0.5], [-0.5, 1]]
          x3, y3 = np.random.multivariate_normal( m1, c1, 5).T


          x=concatenate((x1,x2,x3))
          y=concatenate((y1,y2,x3))

          plot(x,y,'.')

          X=array([x,y]).T
```



In [226]:
```
z=scipy.cluster.hierarchy.linkage(X)
d=scipy.cluster.hierarchy.dendrogram(z)
```