Leibniz-Rechenzentrum
der Bayerischen Akademie der Wissenschaften

numpy, pandas and matplotlib

Ferdinand.Jamitzky@LRZ.de

scipy

NumPy
Base N-dimensional array package

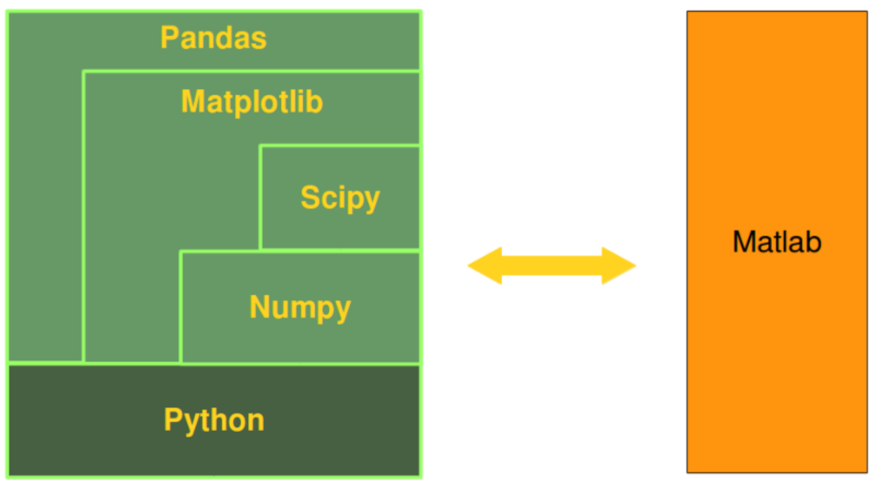
SciPy library
Fundamental library for scientific computing

Matplotlib
Comprehensive 2D Plotting

IPython
Enhanced Interactive Console

Sympy
Symbolic mathematics

pandas
Data structures & analysis



lrz

Leibniz-Rechenzentrum
der Bayerischen Akademie der Wissenschaften

Numerical Computations

numpy

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities
- for comparison to other array languages (Numpy vs MATLAB, R, IDL) see:

http://mathesaurus.sourceforge.net/

Numpy in a nutshell

- NumPy’s main object is the homogeneous multidimensional array. It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. In NumPy dimensions are called axes.

```
>>> A = np.array([[ 1., 0., 0.],[ 0., 1., 2.]])
>>> A.ndim
>>> A.shape
>>> A.size
>>> A.dtype
>>> A.itemsize
```

Array Creation

```
>>> import numpy as np
>>> a = np.array([2,3,4])
>>> a
array([2, 3, 4])
>>> a.dtype
dtype('int64')
>>> b = np.array([1.2, 3.5, 5.1])
>>> b.dtype
dtype('float64')
```

Array Creation

```
>>> np.zeros((3,4))
>>> np.ones((3,4), dtype=np.int16)
>>> np.empty((2,3))
>>> np.arange(10,30,5)
>>> np.arange(0,2,0.3)
>>> np.linspace(0,2,9)
>>> b = np.arange(12).reshape(4,3)
```

Basic Operations

- Vector Operations on Arrays:
  - elementwise add, substract, multiply, divide, power
  - special functions: sin, cos, ...
  - elementwise comparison
  - Matrix Product `A@B`
  - in place operations A+=3
  - `A.sum(), A.cumsum(), A.min(), A.max()`

Universal Functions

- these functions operate elementwise on an array, producing an array as output

all, any, apply_along_axis, argmax, argmin, argsort, average, bincount, ceil, clip, conj, corrcoef, cov, cross, cumprod, cumsum, diff, dot, floor, inner, inv, lexsort, max, maximum, mean, median, min, minimum, nonzero, outer, prod, re, round, sort, std, sum, trace, transpose, var, vdot, vectorize, where

Indexing, Slicing, Iterating

- indexing and slicing like for python lists

```
>>> a[2:5]
>>> a[ : :-1]
>>> b[1:3, : ]
>>> b[-1]
```

Stacking Arrays

```
>>> np.vstack((a,b))
array([[ 8.,  8.],
       [ 0.,  0.],
       [ 1.,  8.],
       [ 0.,  4.]])
>>> np.hstack((a,b))
array([[ 8.,  8.,  1.,  8.],
       [ 0.,  0.,  0.,  4.]])
```

Copy and views

- Simple assignments make **no** copy of array objects or of their data.

```
>>> a = np.arange(12)
>>> b = a              # no new object is created
>>> b is a             # a and b are two names for the
same object
True
>>> d = a.copy()         # a new array object with new
data is created
>>> d is a
False
```

Random Numbers

Numpy has a plentitude of random number distributions

uniform:

```
>>> A = np.random.random(2,3))
>>> A = np.random.uniform(size=10)
```

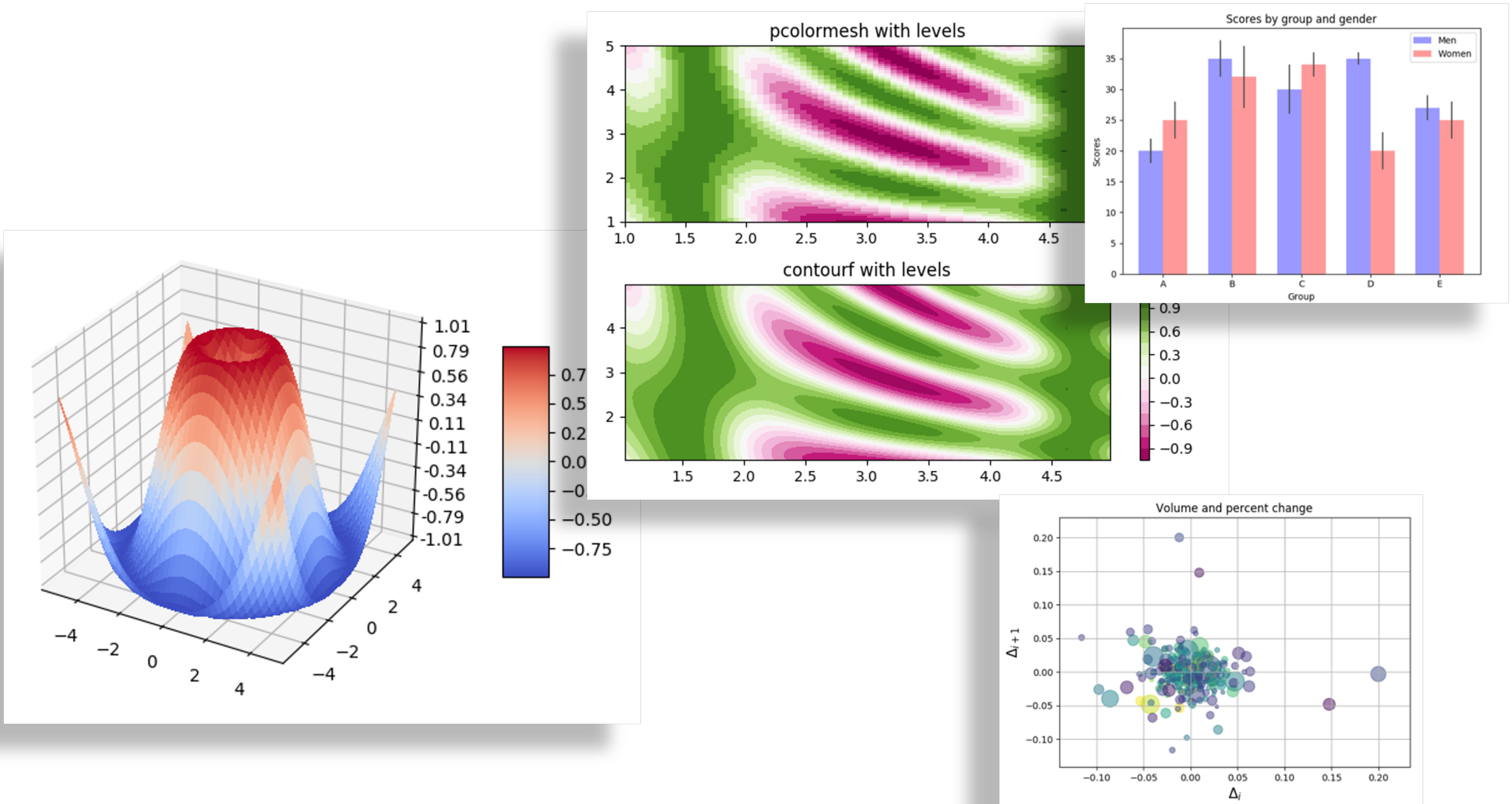
others are:

beta, binomial, chisquare, dirichlet, exponential, F, gamma, geometric, gumbel, hypergeometric, laplace, logistic, lognormal, logseries, multinormal, normal, pareto, poisson, power, Rayleigh, Cauchy, standard_t, triangular, uniform, vonmises, wald, weibul, zipf

Leibniz-Rechenzentrum der Bayerischen Akademie der Wissenschaften

matplotlib

matplotlib



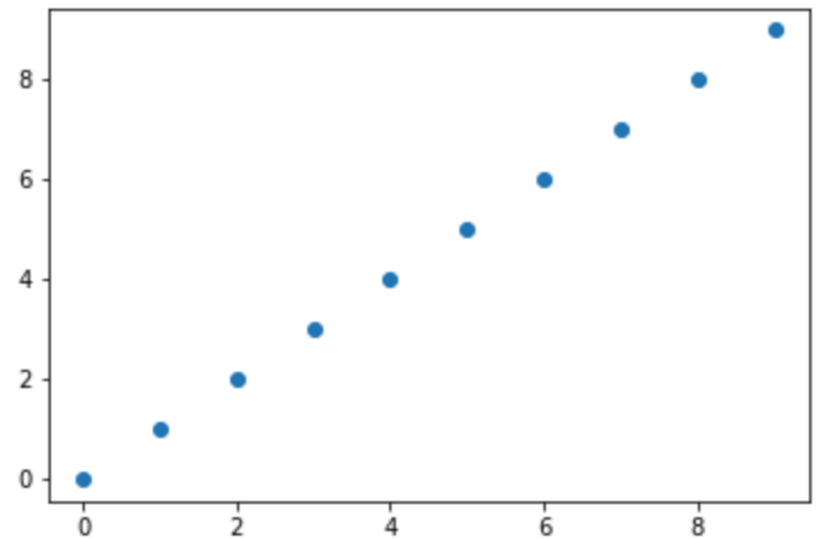
pcolormesh with levels
contourf with levels
Scores by group and gender
Men
Women
Scores
Group
Volume and percent change
Δ_i
Δ_{i+1}

matplotlib

matplotlib can be used as standalone library or together with jupyter, numpy and pandas

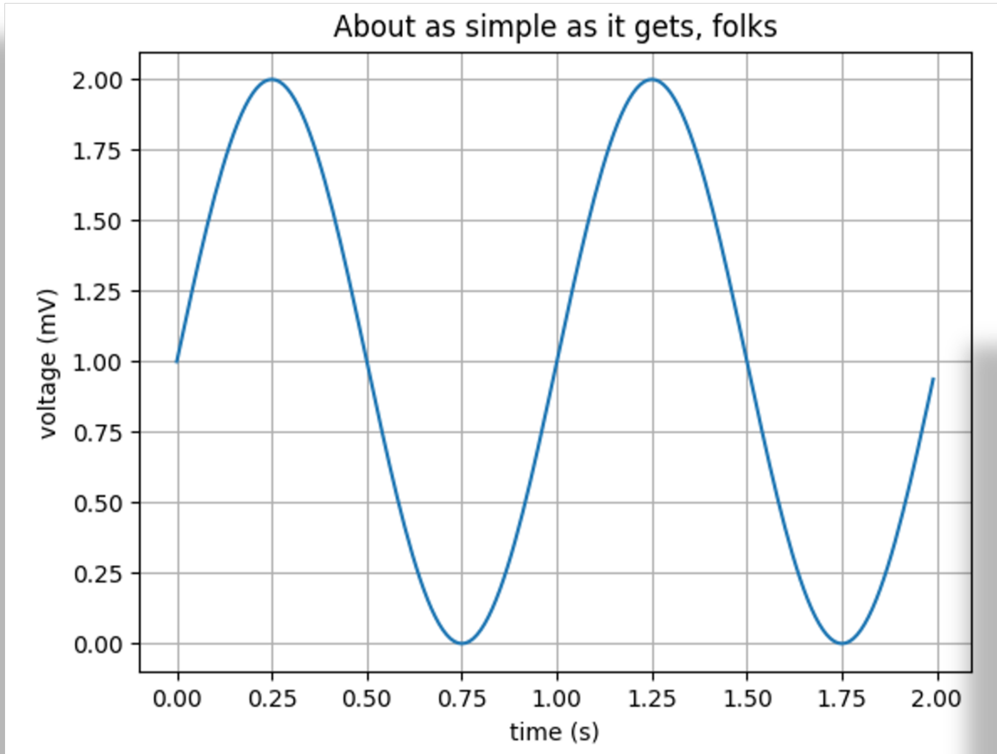
```
[3]: %matplotlib inline
     import matplotlib.pyplot as p
```

```
[5]: p.scatter(range(10),range(10))
     p.show()
```




```
<Figure size 432x288 with 0 Axes>
```

matplotlib




```python
import matplotlib.pyplot as plt
import numpy as np

# Data for plotting
t = np.arange(0.0, 2.0, 0.01)
s = 1 + np.sin(2 * np.pi * t)

# Note that using plt.subplots below is equivalent to using
# fig = plt.figure() and then ax = fig.add_subplot(111)
fig, ax = plt.subplots()
ax.plot(t, s)

ax.set(xlabel='time (s)', ylabel='voltage (mV)',
       title='About as simple as it gets, folks')
ax.grid()

fig.savefig("test.png")
plt.show()
```

matplotlib gallery

Use the gallery for plotting templates:

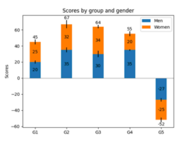
https://matplotlib.org/stable/gallery/index.html

Gallery

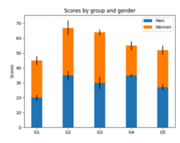
This gallery contains examples of the many things you can do with Matplotlib. Click on any image to see the full image and source code.

For longer tutorials, see our tutorials page. You can also find external resources and a FAQ in our user guide.

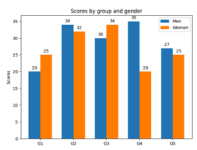
Lines, bars and markers



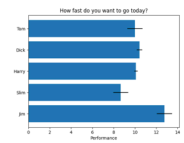
Bar Label Demo



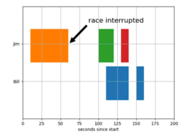
Stacked bar chart



Grouped bar chart with labels

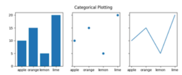


Horizontal bar chart

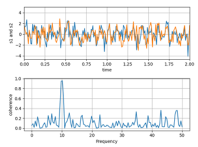


Broken Barh

CapStyle



Plotting categorical variables



Plotting the coherence of two signals

Table of Contents

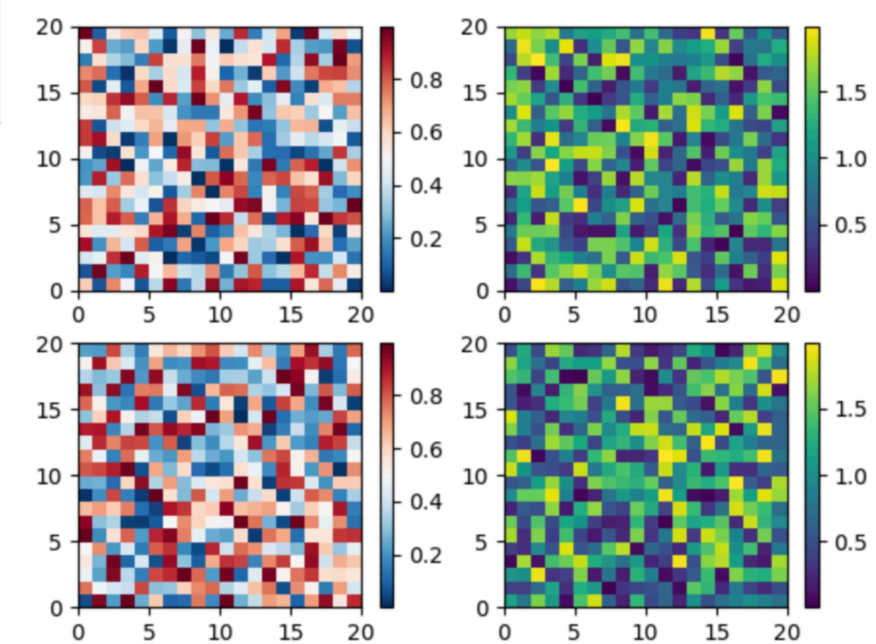
lrz

subplots

matplotlib

```
import numpy as n
import matplotlib.pyplot as p

fig, axs = p.subplots(2, 2)
cmaps = ['RdBu_r', 'viridis']
for col in range(2):
    for row in range(2):
        ax = axs[row, col]
        pcm = ax.pcolormesh(np.random.random((20, 20)) * (col + 1),
                            cmap=cmaps[col])
        fig.colorbar(pcm, ax=ax)
p.show()
```



Leibniz-Rechenzentrum
der Bayerischen Akademie der Wissenschaften

Data Analysis

pandas

- DataFrame object for data manipulation with integrated indexing.
- Tools for reading and writing data between in-memory data structures and different file formats.
- Data alignment and integrated handling of missing data.
- Reshaping and pivoting of data sets.
- Label-based slicing, fancy indexing, and subsetting of large data sets.
- Data structure column insertion and deletion.
- Group by engine allowing split-apply-combine operations on data sets.
- Data set merging and joining.
- Hierarchical axis indexing to work with high-dimensional data in a lower-dimensional data structure.
- Time series-functionality: Date range generation and frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging.

Pandas in a nutshell

The two primary data structures of pandas

- Series (1-dimensional)
- DataFrame (2-dimensional)

handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering.

For R users:

- DataFrame provides everything that R’s data.frame provides
- pandas is built on top of NumPy and is intended to integrate well within a scientific computing environment with many other 3rd party libraries.

Dataframes and Series

DataFrame is a container for Series, and Series is a container for scalars.

```
for col in df.columns:
    series = df[col]
    # do something with series

s = pd.Series([1, 3, 5, np.nan, 6, 8])
```

pandas

- Object Creation
- Viewing Data
- Selection
- Missing Data
- Operations
- Merge
- Grouping
- Reshaping
- Time Series
- Categorials
- Plotting
- Data I/O

Object Creation

Creating a Series by passing a list of values, letting pandas create a default integer index:

```
s = pd.Series([1, 3, 5, np.nan, 6, 8])
```

Creating a DataFrame by passing a NumPy array, with a datetime index and labeled columns:

```
df = pd.DataFrame(np.random.randn(6, 4),
index=dates, columns=list('ABCD'))
```

Viewing Data

```
df.head()
df.tail(3)
df.index
df.columns
df.to_numpy()
df.describe()
```

Selection

```
df['A']
df[0:3]
df.loc[:, ['A', 'B']]
df.iloc[3:5, 0:2]
df[df.A > 0]
df[df > 0]
df2[df2['E'].isin(['two', 'four'])]
df.loc[:, 'D'] = np.array([5] * len(df))
df2[df2 > 0] = -df2
```

Missing Data

```
df1 = df.reindex(index=dates[0:4],
columns=list(df.columns) + ['E'])
df1.dropna(how='any')
df1.fillna(value=5)
pd.isna(df1)
```

Operations

```
df.mean()
df.mean(1)
df.apply(np.cumsum)
df.apply(lambda x: x.max() - x.min())
s.value_counts()
s.str.lower()
```

Grouping

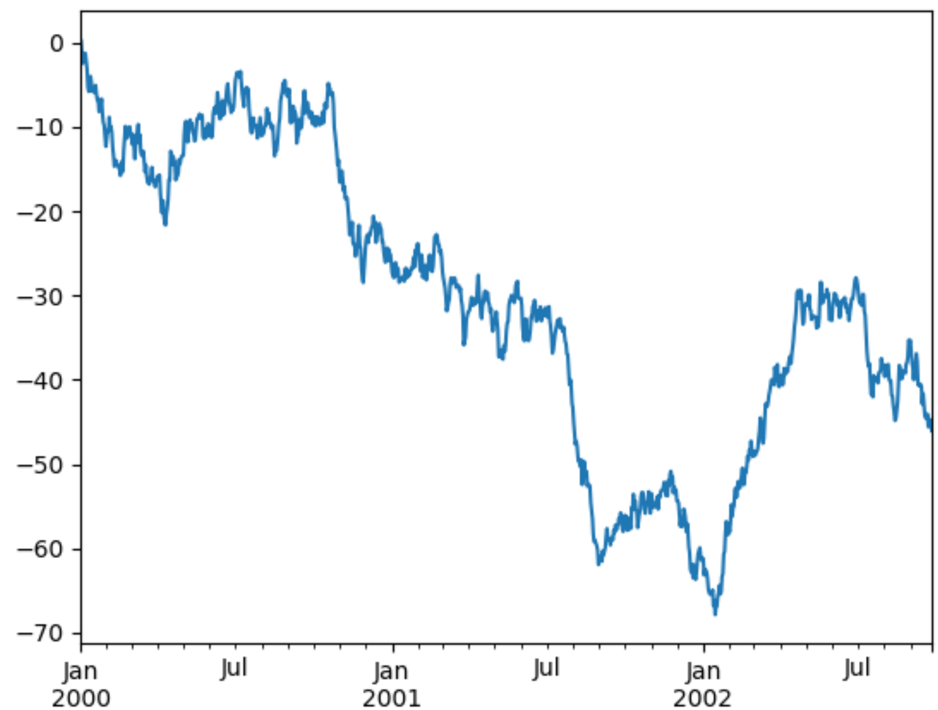
By “group by” we are referring to a process involving one or more of the following steps:

- Splitting the data into groups based on some criteria
- Applying a function to each group independently
- Combining the results into a data structure

```
>>> df.groupby('A').sum()
>>> df.groupby(['A', 'B']).sum()
```

Plotting

```
>>> ts = pd.Series(np.random.randn(1000),
index=pd.date_range('1/1/2000',
periods=1000))
>>> ts = ts.cumsum()
>>> ts.plot()
```



Getting Data In/Out

- CSV

```
>>> pd.read_csv('foo.csv')
>>>  df.to_csv('foo.csv')
```

- Excel

```
>>> pd.read_excel('foo.xlsx', 'Sheet1',
index_col=None, na_values=['NA'])
>>> df.to_excel('foo.xlsx', sheet_name='Sheet1')
```

- HDF5

```
>>>  pd.read_hdf('foo.h5', 'df')
>>> df.to_hdf('foo.h5', 'df')
```

jupyter+scipy+matplotlib+latex

IP[y]: Notebook spectrogram Last Checkpoint: a few seconds ago (autosaved) IPython (Python 3)

File Edit View Insert Cell Kernel Help

Code Cell Toolbar: None

Simple spectral analysis

An illustration of the [Discrete Fourier Transform](#) using windowing, to reveal the frequency content of a sound signal.

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \qquad k = 0, \ldots, N-1$$

We begin by loading a datafile using SciPy's audio file support:

In [1]:
```
from scipy.io import wavfile
rate, x = wavfile.read('test_mono.wav')
```

And we can easily view its spectral structure using matplotlib's builtin `specgram` routine:

In [2]:
```
%matplotlib inline
from matplotlib import pyplot as plt
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
ax1.plot(x); ax1.set_title('Raw audio signal')
ax2.specgram(x); ax2.set_title('Spectrogram');
```