Basic Data Analysis and more (A guided tour using `python`)

O. Melchert

Institut für Physik, Universität Oldenburg

Motivation

Data is comparatively cheap, *insight* is hard to come by!

- Data analysis based on three pillars:
  (1) *Statistics*:
    - craft of using data samples to understand “phenomena”
  (2) *Probability*:
    - study of random events
  (3) *Computation*:
    - tool for quantitative analysis
    - instrument to generate data
- Here:
  - approach (1) and (2) from a computational point of view
  - numerical experiments: consider $1D$ random walk
- Aim:
  - perform analysis as careful as possible
  - arrive at maximally “justifiable” conclusions

Outline

- Basic `python`
- Assembling data (1 D random walk)
- Descriptive statistics
  - summarizing data
  - visualizing data
- More aspects covered in the lecture notes
  - hypothesis testing
  - parameter estimation
  - object-oriented programming via `python`
  - “speed issues”

Basic `python`

- Two basic data structures:
  - Lists:

```
>>> a=[4,2]; a.append(5); print a
[4, 2, 5]
```

  - Dictionaries:

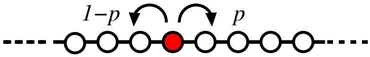
```
>>> d={'n0':[1,2]}; d['n1']=[5,6]; print d
{'n0': [1, 2], 'n1': [5, 6]}
>>> for key,val in d.items(): print key,val
n0 [1, 2]
n1 [5, 6]
```

- Facilitate data analysis and small-scale simulations:
- Many open-source libraries for scientific computing
  - `SciPy`: statistics, optimization, linear algebra, etc.
  - `Networks`: implements graphs and graph algorithms

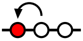
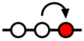
Assembling data

Random experiment:

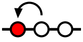
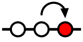
- outcome is not predictable
- e.g.: $1D$ random walk:



Sample space Ω:

- set of elementary events
- e.g.: $1D$ random walk: $\Omega = \{$  ,  $\}$

Random variable (RV):

- function $X : \Omega \to \mathbb{R}$ that relates a numerical value $x = X(\omega)$ to each elementary event $\omega \in \Omega$
- e.g.: $1D$ random walk: $X($  $) = -1$, $X($  $) = 1$

Assembling data

Combination of several RVs

- new RV $Y = f(X^{(0)}, \ldots, X^{(k)})$
- use outcomes $x^{(i)}$ to yield $y = f(x^{(0)}, \ldots, x^{(k)})$

Example: symmetric $1D$ random walk starting at $x_0 = 0$

- probability to step right: $p = 0.5$
- determine endposition x_N after N steps
- random experiment: take one step, repeat N-times
- new random variable $Y = \sum_{i=0}^{N-1} X^{(i)}$ yields endposition $x_N = \sum_{i=0}^{N-1} x^{(i)}$

Relevance of the random walk model:

- continuum limit yields diffusion equation
- simplified model for polymers

Assembling data

- 1*D* random walk – a computer scientists view:

```python
from random import seed, choice

N=100   # nbr of steps in single walk
n=10    # nbr independent walks

print '# (seed) (endPos)'
for s in range(n):
    seed(s)
    # construct single walk
    endPos = 0
    for i in range(N):
        # implement single step, update RV
        endPos += choice([-1,1])
    # dump data to stdout
    print s,endPos
```

Listing 1: `EX_1DrandWalk/1_randWalk.py`

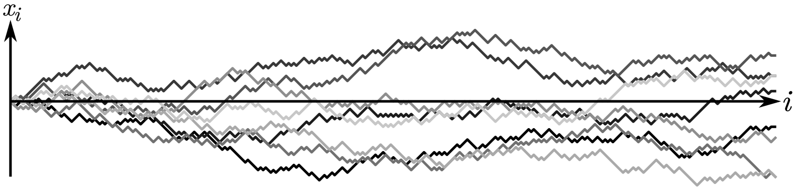
Assembling data

- calling the script yields the *raw data*:

```
$ python 1D_randWalk.py
# (seed) (endPos)
0 26
1 6
2 18
3 14
4 -8
```

Listing 2: Output of `EX_1DrandWalk/1_randWalk.py`

- more pictographic account of 1*D* random walks:



Distributions of RVs

Probability function P:

- $P(X = x)$ signifies probability to observe RV with value x

Probability mass function (PMF):

- $p_x : \mathbb{R} \rightarrow [0, 1]$, where $p_X(x) = P(X = x)$
- description of discrete RV:
  - map numerical values to probabilities
  - discrete state space: $p_X(x) = 0$ except for finite set $\{x_i\}$
  - normalized: $\sum_{x_i} p_X(x_i) = 1$

Cumulative distribution function (CDF):

- $F_X : \mathbb{R} \rightarrow [0, 1]$, where $F_X(x) = P(X \leq x)$
- properties:
  - non-decreasing: if $x_1 < x_2$, then $F_X(x_1) \leq F_X(x_2)$
  - normalized: $\lim_{x \rightarrow -\infty} F_X(x) = 0$, $\lim_{x \rightarrow \infty} F_X(x) = 1$
  - relation to PMF: $F_X(x) = \sum_{x_i < x} p_X(x_i)$

Distribution of RVs

- Represent raw data (i.e. a finite dataset) as PMF:

```
def getPmf(myList):
    """construct prob mass fct"""
    # step 1: compute frequencies
    fHist = {}
    for x in myList:
        fHist.setdefault(x,0)
        fHist[x] += 1

    # step 2: normalization
    N = len(myList)
    myPmf = {}
    for x,freq in fHist.items():
        myPmf[x]=float(freq)/N

    return myPmf
```

Listing 3: Variant of function `getPmf` in `MCS2012_lib.py`

Distribution of RVs

- Postprocess raw data to yield PMF and CDF:

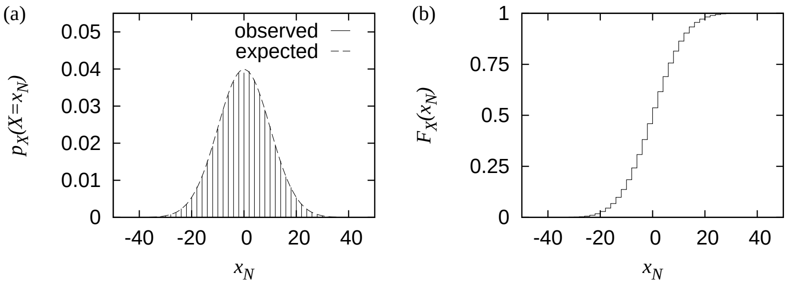
```
import sys
from MCS2012_lib import *

# parse command line arguments
fileName = sys.argv[1]
col      = int(sys.argv[2])

# construct approximate pmf from data
rawData  = fetchData(fileName,col)
pmf      = getPmf(rawData)

# dump pmf and cdf to standard outstream
FX=0.
for endpos in sorted(pmf):
    FX+=pmf[endpos]
    print endpos,pmf[endpos],FX
```

Listing 4: Script `EX_1DrandWalk/pmf.py`

Distributions of RVs

Monte Carlo simulation (discussed by HGK):

- $n = 10^5$ independent $N = 100$-step walks
- hint: store raw data in file, e.g. `N100_n100000.dat`
- determine distribution of endpoints x_N as
  `python pmf.py N100_n100000.dat > N100_n100000.pmf`
- (a) PMF (enclosing curve = Gaussian with $\mu = 0$ and $\sigma = \sqrt{N}$),
  (b) CDF (figures prepared using `gnuplot`)



Summary statistics

Features of a distribution function:

- moments of the distribution

$$E[X^k] = \begin{cases} \sum_i x_i^k \, p_X(x_i), & \text{for } X \textit{ discrete}, \\ \int_{-\infty}^{\infty} x^k \, p_X(x) \, dx, & \text{for } X \textit{ continuous}. \end{cases}$$

  $E[\cdot]$ signifies the *expectation operator*.

Here:

- sample of N iid values $x = \{x_0, \ldots, x_{N-1}\}$
- summary statistics: reduce full dataset to single number

Summary statistics

- Basic parameters related to a finite dataset:

  $\mathrm{av}(x) = \frac{1}{N}\sum_{i=0}^{N-1} x_i$ (average/mean value)

  - central tendency of sample

  $\mathrm{Var}(x) = \frac{1}{N-1}\sum_{i=0}^{N-1}[x_i - av(x)]^2$ (corrected variance)

  - unbiased estimator for the spread of the $x_i \in x$
  - proper implementation: corrected two-pass algorithm

  $\mathrm{sDev}(x) = \sqrt{\mathrm{Var}(x)}$ (standard deviation)

  $\mathrm{sErr}(x) = \frac{1}{\sqrt{N}}\mathrm{sDev}(x)$ (standard error)

  - signifies how accurate sample mean approximates the true mean

- Convergence properties of the above observables might be poor, if distribution has a broad tail!

  $\rightarrow$ more robust estimation of deviations in the sample:
  $\mathrm{aDev}(x) = \frac{1}{N}\sum_{i=0}^{N-1} |x_i - \mathrm{av}(x)|$ (absolute deviation)

Summary statistics

- Summary statistics based on $\text{av}(x)$:

```
def basicStatistics(myList):
    """compute summary statistics"""
    av=var=tiny=0.
    N=len(myList)

    for x in myList:  # 1st pass
        av += x
    av /= N

    for x in myList:  # 2nd pass
        dum   = x - av
        tiny += dum
        var  += dum*dum

    var  = (var - tiny*tiny/N)/(N-1)
    sDev = sqrt(var)
    sErr = sDev/sqrt(N)

    return av, sDev, sErr
```

Listing 5: Function `basicStatistics` in `MCS2012_lib.py`

Example 1: good convergence

- Script to compute summary statistics:

```
import sys
from MCS2012_lib import *

## parse command line arguments
fileName      = sys.argv[1]
col           = int(sys.argv[2])

## construct approximate pmf from data
rawData       = fetchData(fileName,col)
av,sDev,sErr  = basicStatistics(rawData)

print "av   = %4.3lf"%av
print "sErr = %4.3lf"%sErr
print "sDev = %4.3lf"%sDev
```

Listing 6: Script `EX_1DrandWalk/basicStats.py`

Example 1: good convergence

- Obtain summary of the raw data:

```
$ python basicStats.py N100_n100000.dat 1
av   = 0.008
sErr = 0.032
sDev = 10.022
```

Listing 7: Summary statistics for 1*D* random walk

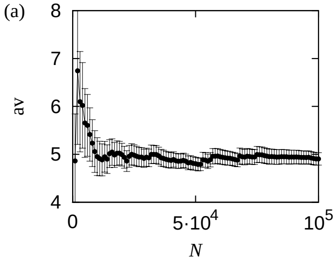
- Central limit theorem:
  - independently drawn values
  - values drawn from the same distribution with mean μ and standard deviation σ
  - sum up n values

$\rightarrow$ distribution of summed up values tends to be normal with mean $n\mu$ and variance $n\sigma^2$

Example 2: Poor convergence

Poor convergence:

- power-law distributed data: $p_X(x) \propto x^{-\alpha}$ ($N = 10^5$, $\alpha = 2.2$)
  $\rightarrow$ obtain random variates via inversion method:
  $x = x_0(1-r)^{-1/(\alpha-1)}$ ($r \in [0,1)$, $x \in [x_0, \infty)$)
- robust estimators less affected by “outliers”
  (consider also summary measures based on *median* $\rightarrow$ see exercises)
- (a) mean value, (b) standard deviation (inset: absolute deviation)



Estimators with(out) bias

Unbiased estimator:

- consider estimator $\hat{\phi}(x)$ for parameter ϕ
- estimator unbiased if $E[\hat{\phi}(x)] = \phi$
  $\rightarrow E[\cdot]$ with respect to all possible data sets

Example:

- sample $x = \{x_0, \ldots, x_{N-1}\}$, true mean μ, true variance σ^2
- estimate mean $\phi = \mu$ using $\hat{\phi}(x) = \mathrm{av}(x)$:
  $\rightarrow$ unbiased since $E[\mathrm{av}(x)] = \mu$
- associated mean square error (MSE) $E[(\hat{\phi}(x) - \phi)^2]$
  $\rightarrow$ measures variance $+$ bias
- uncorrected variance $\mathrm{uVar}(x) = \frac{1}{N}\sum_{i=0}^{N-1}[x_i - \mathrm{av}(x)]^2$:
  $\rightarrow$ biased since $E[\mathrm{uVar}(x)] = \frac{N-1}{N}\sigma^2$

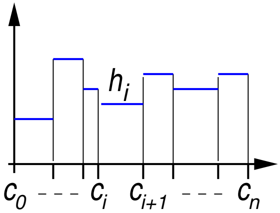
Graphical representation of data

Histogram:

- consider sample $x = \{x_0, \ldots, x_{N-1}\}$
- discrete approximation of underlying prob dens fct requires:
  (1) n distinct intervals $C_i = [c_i, c_{i+1})$, $i = 0 \ldots n-1$ (bins)
  bin-width: $\Delta c_i = c_{i+1} - c_i$
  (2) frequency density $h_i = n_i / [N \times \Delta c_i]$
  (n_i = number of elements in bin C_i)
- Histogram = set of tuples

$$H = \{(C_i, h_i)\}_{i=0}^{n-1}$$

$\rightarrow$ normalized: $\sum_i h_i \times \Delta c_i = 1$
$\rightarrow$ data binning = information loss



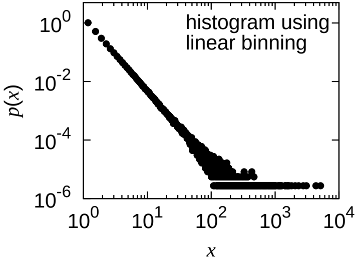
Example 3(a): linear binning

Linear binning:

- n bins of equal width $\Delta c = (x_+ - x_-)/n$
- interval bounds $c_i = x_- + i\Delta c$, $i = 0 \ldots n$
- element x_j belongs to bin C_i with $i = \lfloor x_j/\Delta c \rfloor$

Example:

- `python` example 3(a)
- power-law PDF: $p_X(x) \propto x^{-\alpha}$, $\alpha = 2.5$, $N = 10^6$
- linear binning, $n = 2 \times 10^4$ bins



Example 3(a): linear binning

- Implementation of linear binning:

```
def hist_linBinning(rawData,xMin,xMax,nBins=10):
   """construct histogram using linear binning"""
   h = [0]*nBins              # ini freqs for each bin
   dx = (xMax-xMin)/nBins # uniform bin width

   # bin id corresponding to value
   def binId(val): return int(floor((val-xMin)/dx))
   # lower + upper boundary for binId i
   def bdry(i): return xMin+i*dx, xMin+(i+1)*dx

   for value in rawData: # data binning
      if 0<=binId(value)<nBins: h[binId(value)]+=1

   N=sum(h)
   for bin in range(nBins): # dump histogram
        hRel    = float(h[bin])/N
        low,up = bdry(bin)
        print low, up, hRel/(up-low)
```

Listing 8: Function `hist_linBinning` in `MCS2012_lib.py`

Example 3(b): logarithmic binning

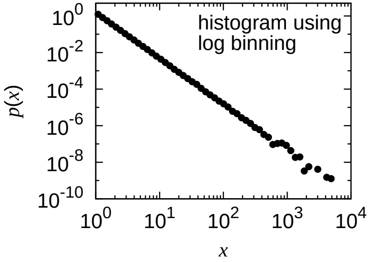
Logarithmic binning:

- interval bounds $c_i = c_0 \times \exp\{i\Delta c'\}$
- "growth factor" for bin width $\Delta c' = \log(x_+/x_-)/n$
- element x_j belongs to bin C_i with $i = \lfloor \log(x_j/x_-)/\Delta c' \rfloor$

```
dx = log(xMax/xMin)/nBins
def binId(val): return int(floor(log(val/xMin)/dx))
def bdry(i): return xMin*exp(i*dx), xMin*exp((i+1)*dx)
```

Example:

- `python` example 3(b)
- power-law PDF: $p_X(x) \propto x^{-\alpha}$, $\alpha = 2.5$, $N = 10^6$
- log-binning, $n = 55$ bins



Bootstrap resampling

Error estimation via bootstrap resampling

- given: sample $x = \{x_0, \ldots, x_{N-1}\}$ of statistically independent numbers
- aim: measure $q = f(x)$ and provide unbiased error estimate
- three-step procedure:
  (1) generate M auxiliary bootstrap data sets $\tilde{x}^{(k)}$, $k = 0 \ldots M-1$
  (2) compute $\tilde{q}_k = f(\tilde{x}^{(k)})$ to yield set of estimates $\tilde{q} = \{\tilde{q}_k\}_{k=0}^{M-1}$
  (3) obtain bootstrap error estimate
  $$\mathrm{sDev}(\tilde{q}) = \left(\frac{1}{M-1}\sum_{k=0}^{M-1}[\tilde{q}_k - \mathrm{av}(\tilde{q})]^2\right)^{1/2}$$

$\rightarrow$ basic assumption: $\tilde{q}_k$ are distributed around q similar to the way, independent estimates of q are distributed around the true quantity $q^\star$

Bootstrap resampling

- function to perform empirical bootstrap resampling of data:

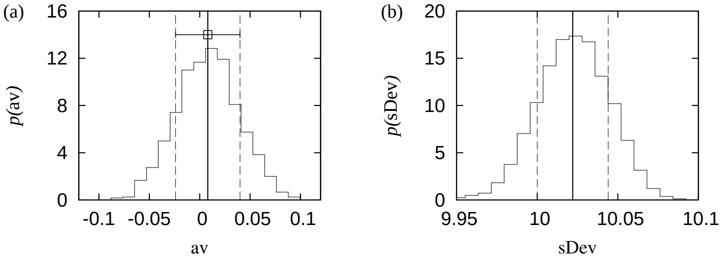
```
def bootstrap(array,estimFunc,nBootSamp=128):
        """bootstrap resampling of dataset"""
        # estimate mean value from original array
        origEstim=estimFunc(array)
        # resample data from original array
        nMax=len(array)
        h    = [0.0]*nBootSamp
        bootSamp = [0.0]*nMax
        for sample in range(nBootSamp):
           for val in range(nMax):
              bootSamp[val]=array[randint(0,nMax-1)]
           h[sample]=estimFunc(bootSamp)
        # estimate error as std deviation of
        # resampled values
        resError = basicStatistics(h)[1]
        return origEstim,resError
```

Listing 9: Function `bootstrap` in `MCS2012_lib.py`

Example 4: bootstrap resampling

Example:

- revisit endpoint data for $1D$ random walk
- $M = 1024$ bootstrap data sets
- result: $\mathrm{av} = 0.008 \pm 0.032$, $\mathrm{sDev} = 10.022 \pm 0.022$ PDF (histogram using 18 bins) of (a) resampled av, (b) resampled sDev



- Descriptive statistics
  - summarizing data
  - visualizing data
- Howto accomplish things using `python`
- More aspects covered in the lecture notes
- Tutorial: “Statistical data analysis”
  - 16:00-17:15 (today)
  - W1 0-008
- Thank you!