Data Mining

Tamás Budavári - budavari@jhu.edu
Class 3

- Sampling from Distributions
- Density Estimation

Samples, PDFs in 1-D and 2-D

Descriptive Statistics

- Characterization of location, dispersion, etc.

Characterization	Sample Estimates	Probabilisty Density Functions
Average	$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i = \left\langle x_i \right\rangle_{i=1}^{N}$	$\mu = \mathbb{E}[X] = \int x\, p(x)\, dx$
Variance	$s^2 = \frac{1}{N-1}\sum_{i=1}^{N} \left(x_i - \bar{x}\right)^2$	$\mathbb{V}\mathrm{ar}[X] = \int (x-\mu)^2 p(x)\, dx$

- Useful connection to sampling

Sampling from Distributions

- Uniform between a and b: scale and shift

$$U_{ab} = a + (b-a)\, U_{01}$$

- Inverse transform sampling in $\mathbb{R}$

$$X = \mathrm{CDF}^{-1}(U_{01})$$

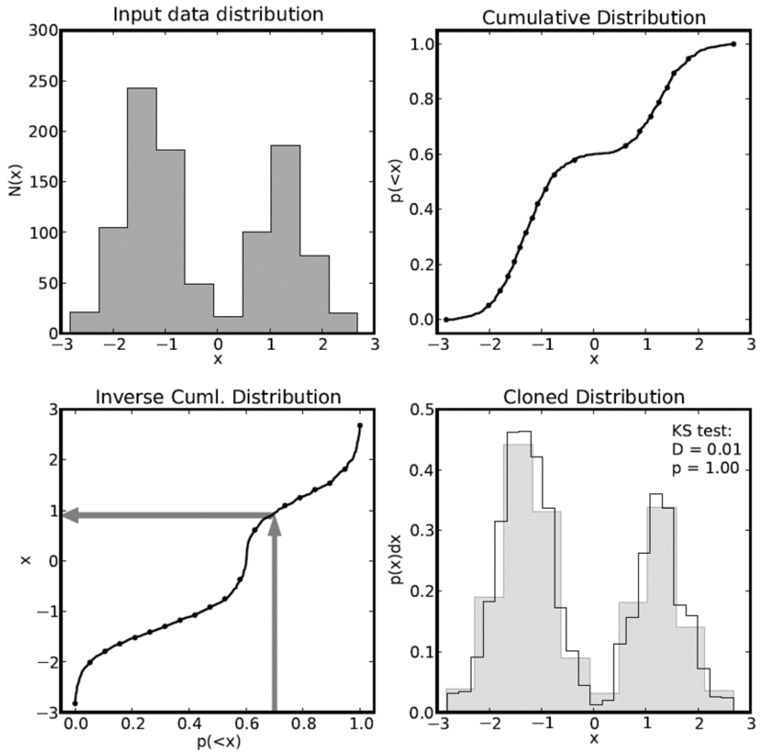
Unhomework: prove it!

Pros: 100% efficient: every random seed generated will become the random sample of the target distribution.

Cons: require the knowledge of the CDF and its inverse function, which are not always available in analytical expression (including the normal distribution) or may be computationally inefficient.

The inverse transform sampling method works as follows:

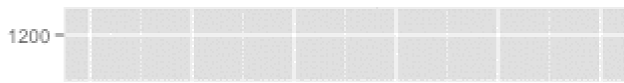
- Generate a random number u from the standard uniform distribution in the interval [0,1], e.g. from $U \sim \mathrm{Unif}[0, 1]$.
- Find the inverse of the desired CDF, e.g. $F_X^{-1}(x)$.
- Compute $X = F_X^{-1}(u)$. The computed random variable X has distribution $F_X(x)$.



Input data distribution
N(x)
x
Cumulative Distribution
p(<x)
x
Inverse Cuml. Distribution
x
p(<x)
Cloned Distribution
p(x)dx
x
KS test:
D = 0.01
p = 1.00

- Rejection sampling - also works in $\mathbb{R}^N$

> Pros: only require PDF of the target distribution (no need for CDF)
>
> Cons:
>
> - Normally the efficiency will be less than 100%.
> - The algorithm becomes inefficient and impractical as the dimensions of the problem get larger.
> - The algorithm becomes unstable (extremely large variance) with improper proposal density.
>
> The algorithm to obtain a sample from X with target density f using samples from Y with proposal density g is as follows:
>
> - Obtain a sample y from distribution Y and a sample u from $\mathrm{Unif}(0, 1)$.
> - Check whether or not $u < f(y)/Mg(y)$. If this holds, accept y as a sample drawn from f; if not, reject the value of y and return to the sampling step.
>
> The algorithm will take an average of M iterations to obtain a sample.
>
> **Note**: the proposal density must majorize the target density, i.e., $g(y) = 0$ implies $f(x) = 0$.



Numerical Methods

If the $\{x_i\}$ set is sampled from the probability density function $p(\cdot)$, the following will be true:

- Average

$$\mathbb{E}[X] = \int x\,p(x)\,dx \approx \frac{1}{N}\sum_{i=1}^{N} x_i$$

- Variance

$$\mathbb{E}[(X-\mu)^2] = \int (x-\mu)^2\,p(x)\,dx \approx \frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)^2$$

compare to

unbiased sample variance: $s^2 = \dfrac{1}{N-1}\displaystyle\sum_{i=1}^{N}\left(x_i-\bar{x}\right)^2$ (when μ is unknown)

Bessel correction: the use of $N-1$ instead of N in the formula of unbiased sample variance.

Intuition for dividing by $N-1$: in general, the values of $\{x_i-\bar{x}\}$ tend to be smaller than the values of $\{x_i-\mu\}$, since $\bar{x}$ is more likely to be closer to $\{x_i\}$. Hence, dividing by $N-1$ can compensate the small difference to make an unbiased variance estimate.

In [1]:
```
%pylab inline
from scipy.stats import norm as gaussian
```

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

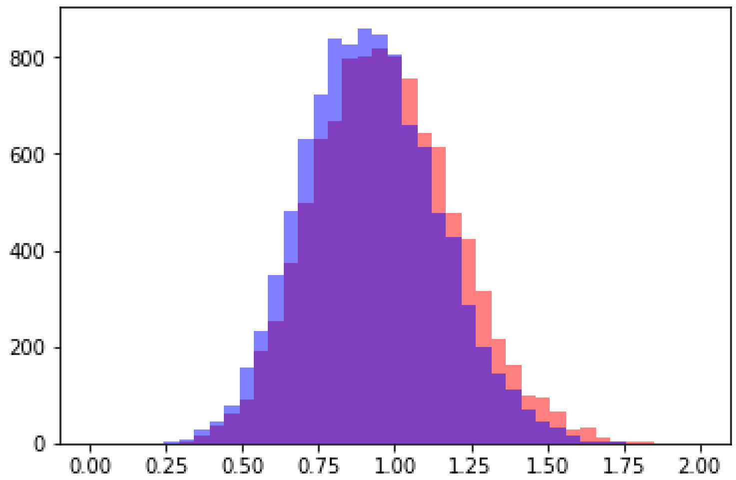
In [2]:
```
# Generate sample with size N
mu, sigma, N = 0, 1, 10
x = gaussian.rvs(mu, sigma, N)

# Mean estimate
avg = np.mean(x)

# Variance estimates
s2   = np.sum( (x-avg)**2 ) /(N-1)  # correct
s2n  = np.sum( (x-avg)**2 ) / N     # biased
s2k  = np.sum( (x- mu)**2 ) / N     # known mean

# Standard deviation estimates
sqrt(s2), sqrt(s2n), sqrt(s2k)
```

Out[2]:
```
(0.8080517700618420З, 0.76658521821782377, 0.7666577301421913)
```

In [3]:

```
# Generate M runs with N samples each
mu, sigma, N, M = 0, 1, 10, 10000
X = gaussian.rvs(loc=mu, scale=sigma, size=(N,M))
avg = np.mean(X, axis=0)
print("Shape of X:", X.shape) # 10000 runs with 10 samples each (each column is a run)
print("Shape of the average:", avg.shape) # 10000 value with each being a sample mean of 10 values

# Variance estimates - check out broadcasting in X-avg
s2   = np.sum( (X-avg)**2, axis=0) /(N-1) # correct
s2n  = np.sum( (X-avg)**2, axis=0) / N    # biased
s2k  = np.sum( (X- mu)**2, axis=0) / N    # known mean

print ("Shape of variance:", s2.shape) # 10000 value with each being a variance of 10 values

# Standard deviation estimates (true value = 1)
s, sn, sk = np.sqrt(s2), np.sqrt(s2n), np.sqrt(s2k)
print ("Unbiased:", mean(s), "Biased:", mean(sn), "Known mean:", mean(sk))

print("Variance estimates: known mean > unbiased > biased")

# Plot to compare the difference
hist(s , 41, range=[0,2], color='r', alpha=0.5);
hist(sn, 41, range=[0,2], color='b', alpha=0.5);
```

```
Shape of X: (10, 10000)
Shape of the average: (10000,)
Shape of variance: (10000,)
Unbiased: 0.970621960988 Biased: 0.92081284311 Known mean: 0.971755426699
Variance estimates: known mean > unbiased > biased
```



Density Estimation

- Histograms
  - Width of bins, h
  - Start of bin boundary, x_0

$$\text{Hist}(x) = \frac{1}{N}\sum_i \mathbf{1}_{\text{bin}(x_i;x_0,h)}(x)$$

- Kernel Density Estimation (KDE)
  - Bandwidth h

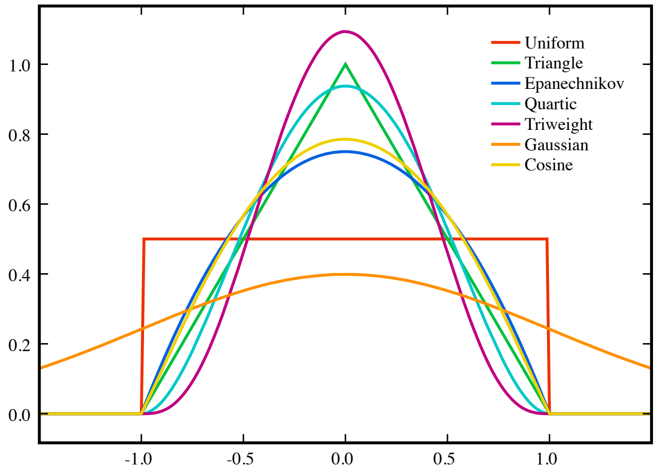
$$\text{KDE}(x) = \frac{1}{N}\sum_i K_h(x-x_i) = \frac{1}{Nh}\sum_i K\left(\frac{x-x_i}{h}\right)$$

  - Can use different $K(\cdot)$ kernel functions
    - E.g., Uniform, Triangular, Gauss, Epanechnikov
  - Essentially, we are putting a density (kerneal funciton) on every data point and then add them together.

See animations at http://www.mglerner.com/blog/?p=28 (http://www.mglerner.com/blog/?p=28)

Kernel Function

- Finite vs Infinite support
- Numerical evaluations
- Frequently used kernels



Learn more about KDE here (https://jakevdp.github.io/blog/2013/12/01/kernel-density-estimation/) and also check out Bayesian Blocks here (https://jakevdp.github.io/blog/2012/09/12/dynamic-programming-in-python/) — tutorials by Jake Vanderplas

Detour: Dirac delta

- In the limit of $h \to 0$, the kernel will become strange:

> **Dirac's** δ "function" is 0 everywhere except at 0 such that
>
> $$\int \delta(x)\, dx = 1$$

- Interesting properties, e.g.,

> $$\int f(x)\, \delta(x-a)\, dx = f(a)$$

- See **distribution theory** and **functionals** for more background
- **Note**: that Dirac delta function is different than the indicator function. Let $A = \{0\}$ and an indicator function on A be I_A, then $\int I_A(x)\, dx = 0$.

An interesting result

- Bad density estimation but if...

$$p(x) = \frac{1}{N}\sum_{i=1}^{N} \delta(x - x_i)$$

- The expectation value

$$\mathbb{E}[X] = \int x \frac{1}{N}\sum_{i=1}^{N} \delta(x - x_i)\, dx$$

$$\mathbb{E}[X] = \frac{1}{N}\sum_{i=1}^{N} \int x\, \delta(x - x_i)\, dx$$

$$\mathbb{E}[X] = \frac{1}{N}\sum_{i=1}^{N} x_i$$

Unhomework

1. Sample from a mixture of two Gaussians using uniform random numbers in the [0,1) interval. Try different (μ_1, σ_1) and (μ_2, σ_2) values!
2. Build different density estimators and compare to the original PDF. Try histogramming and KDE with different parameters.

A Little Bit More about Plot in Python

Basic Plot

- **Example**

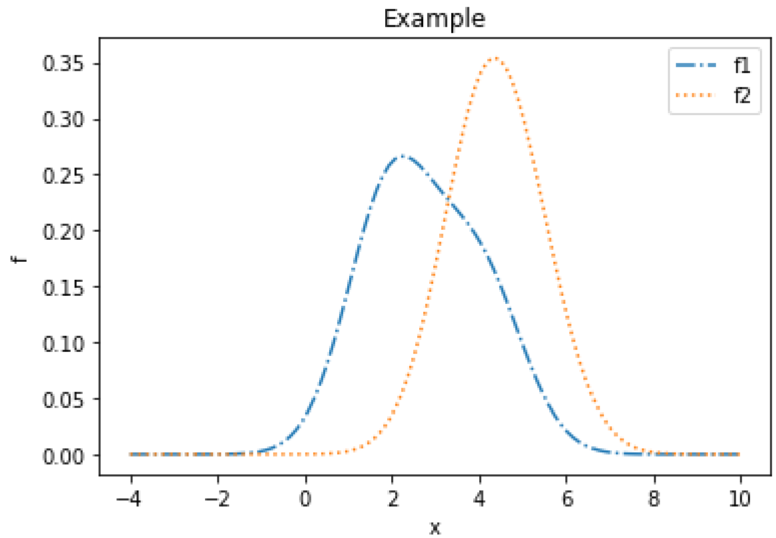
$$f_1(x) = \frac{3}{5} \cdot \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{(x-2)^2}{2}\right] + \frac{2}{5} \cdot \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{(x-4)^2}{2}\right]$$

$$f_2(x) = \frac{3}{5} \cdot \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{(x-4)^2}{2}\right] + \frac{2}{5} \cdot \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{(x-6)^2}{2}\right]$$

In [4]:

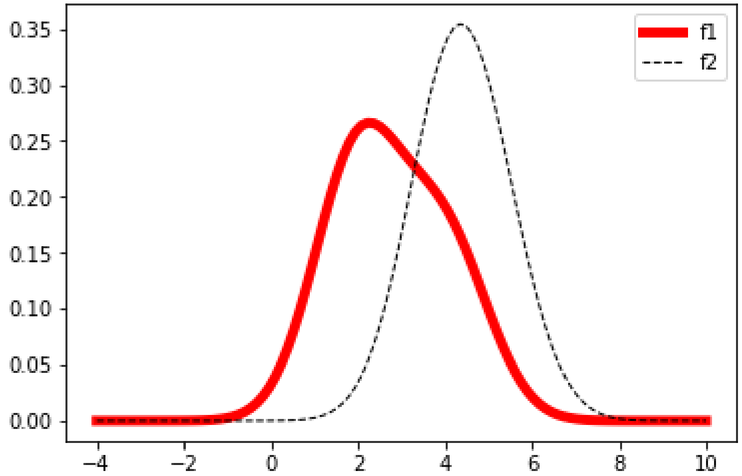
```
# Generate data
x = np.linspace(-4,10,100)
f1 = (3/5) * (1/np.sqrt(2*np.pi)) * np.exp(-(x-2)**2/2) + (2/5) * (1/np.sqrt(2*np.pi)) * np.exp(-(x-4
)**2/2)
f2 = (3/5) * (1/np.sqrt(2*np.pi)) * np.exp(-(x-4)**2/2) + (2/5) * (1/np.sqrt(2*np.pi)) * np.exp(-(x-5
)**2/2)
```

In [5]:
```
# Plot (in one plot command)
plot(x, f1, '-.', x, f2, ':');
xlabel('x');
ylabel('f');
title('Example');
legend(['f1', 'f2']);
```



In [6]:

```
# Plot (in two plot commands)
plot(x, f1, 'r-', linewidth = 5, label='f1');
plot(x, f2, '--k', linewidth = 1, label='f2');
legend();
```



- **Line Style or Marker**

character	description
'-'	solid line style
'--'	dashed line style
'-.'	dash-dot line style
':'	dotted line style
'.'	point marker
','	pixel marker
'o'	circle marker
'v'	triangle_down marker
'^'	triangle_up marker
'<'	triangle_left marker
'>'	triangle_right marker
'1'	tri_down marker

'1'	tri_down marker
'2'	tri_up marker
'3'	tri_left marker
'4'	tri_right marker
's'	square marker
'p'	pentagon marker
'*'	star marker
'h'	hexagon1 marker

- **Color**

character	color
‘b’	blue
‘g’	green
‘r’	red
‘c’	cyan
‘m’	magenta
‘y’	yellow
‘k’	black
‘w’	white

Histogram

In [7]:
```
from scipy import stats
```

Syntax:

- matplotlib.pyplot.hist(x, bins=None, range=None, density=None, orientation='vertical', color=None, normed=None, **kwargs)

Parameters:

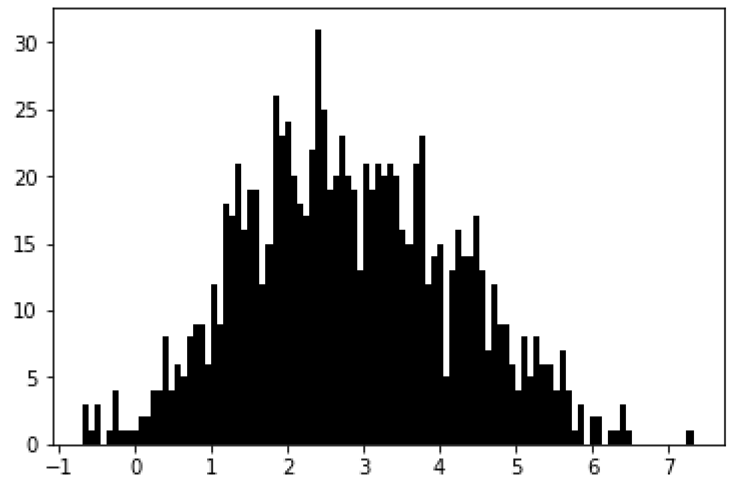
- **x** : (n,) array or sequence of (n,) arrays
  Input values, this takes either a single array or a sequency of arrays which are not required to be of the same length.
- **bins** : integer or sequence or 'auto', optional
  The number of bins.
- **range** : tuple or None, optional
  The lower and upper range of the bins. Lower and upper outliers are ignored
- **normed** : boolean, optional
  Count or percentage.

- **Example**

$$f(x) = \frac{3}{5} \cdot \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{(x-2)^2}{2}\right] + \frac{2}{5} \cdot \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{(x-4)^2}{2}\right]$$

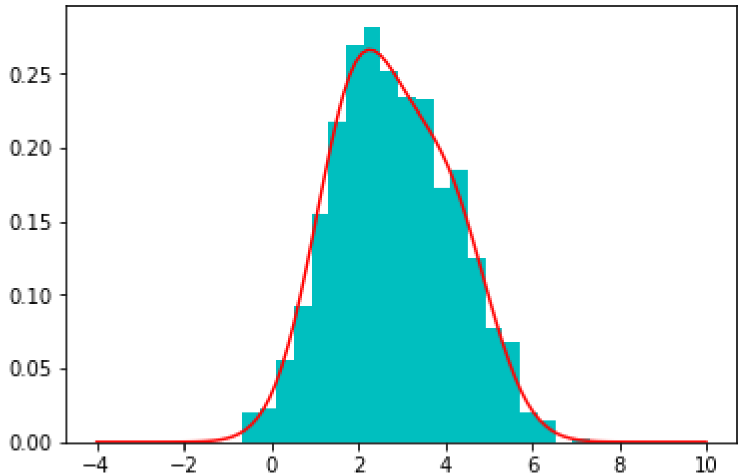
In [8]:
```
# Generate data
np.random.seed(seed=2018)
s1 = stats.norm.rvs(loc=2, scale=1, size=600)
s2 = stats.norm.rvs(loc=4, scale=1, size=400)
s = np.hstack([s1,s2])
```

In [9]:
```
# Plot
hist(s, bins=100, color='k');
```



In [10]:

```
# Plot to compare the sampled and theoretical distribution
hist(s, normed=True, bins=20, color='c');      # normed = True to compare the density
plot(x, f1, 'r');
```



Box Plot

Syntax:

- matplotlib.pyplot.boxplot(x, sym=None, **kwargs)

Parameters:

- **x** : Array or a sequence of vectors
  The input data.
- **sym** : str, optional
  The default symbol for flier points. Enter an empty string ('') if you don't want to show fliers. If None, then the fliers default to 'b+' If you want more control use the flierprops kwarg.

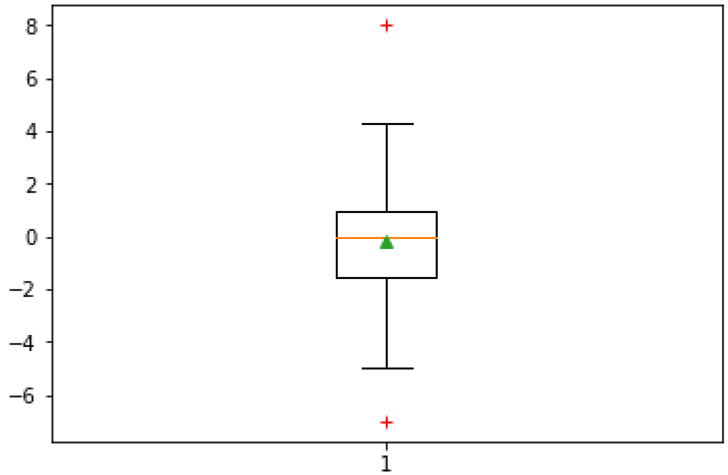
Returns

- **result** : dict
  A dictionary mapping each component of the boxplot to a list of the matplotlib.lines.Line2D instances created. That dictionary has the following keys (assuming vertical boxplots):
  - boxes: the main body of the boxplot showing the quartiles and the median's confidence intervals if enabled.
  - medians: horizontal lines at the median of each box.
  - whiskers: the vertical lines extending to the most extreme, non-outlier data points.
  - caps: the horizontal lines at the ends of the whiskers.
  - fliers: points representing data that extend beyond the whiskers (fliers).
  - means: points or lines representing the means.

- **Example**

In [11]:
```
# Genrate data and add some outliers
np.random.seed(seed=2018)
x = stats.norm.rvs(loc=0, scale=2, size=100)
x[20] = 8
x[60] = -7
```

In [12]:
```python
# Plot
result = boxplot(x, sym='r+', showmeans=True); # whis = 1.5 (default)
```



In [13]:
```python
result
```

Out[13]:
```
{'boxes': [<matplotlib.lines.Line2D at 0x1a1246c940>],
 'caps': [<matplotlib.lines.Line2D at 0x1a124794a8>,
  <matplotlib.lines.Line2D at 0x1a12479908>],
 'fliers': [<matplotlib.lines.Line2D at 0x1a124df6a0>],
 'means': [<matplotlib.lines.Line2D at 0x1a124df208>],
 'medians': [<matplotlib.lines.Line2D at 0x1a12479d68>],
 'whiskers': [<matplotlib.lines.Line2D at 0x1a1246cb70>,
  <matplotlib.lines.Line2D at 0x1a12479048>]}
```

In [14]:
```python
# Find the outliers based on box plot
outliers = result['fliers']
for outlier in outliers:
    print(outlier.get_data())
```

```
(array([ 1.,  1.]), array([-7.,  8.]))
```

Subplot

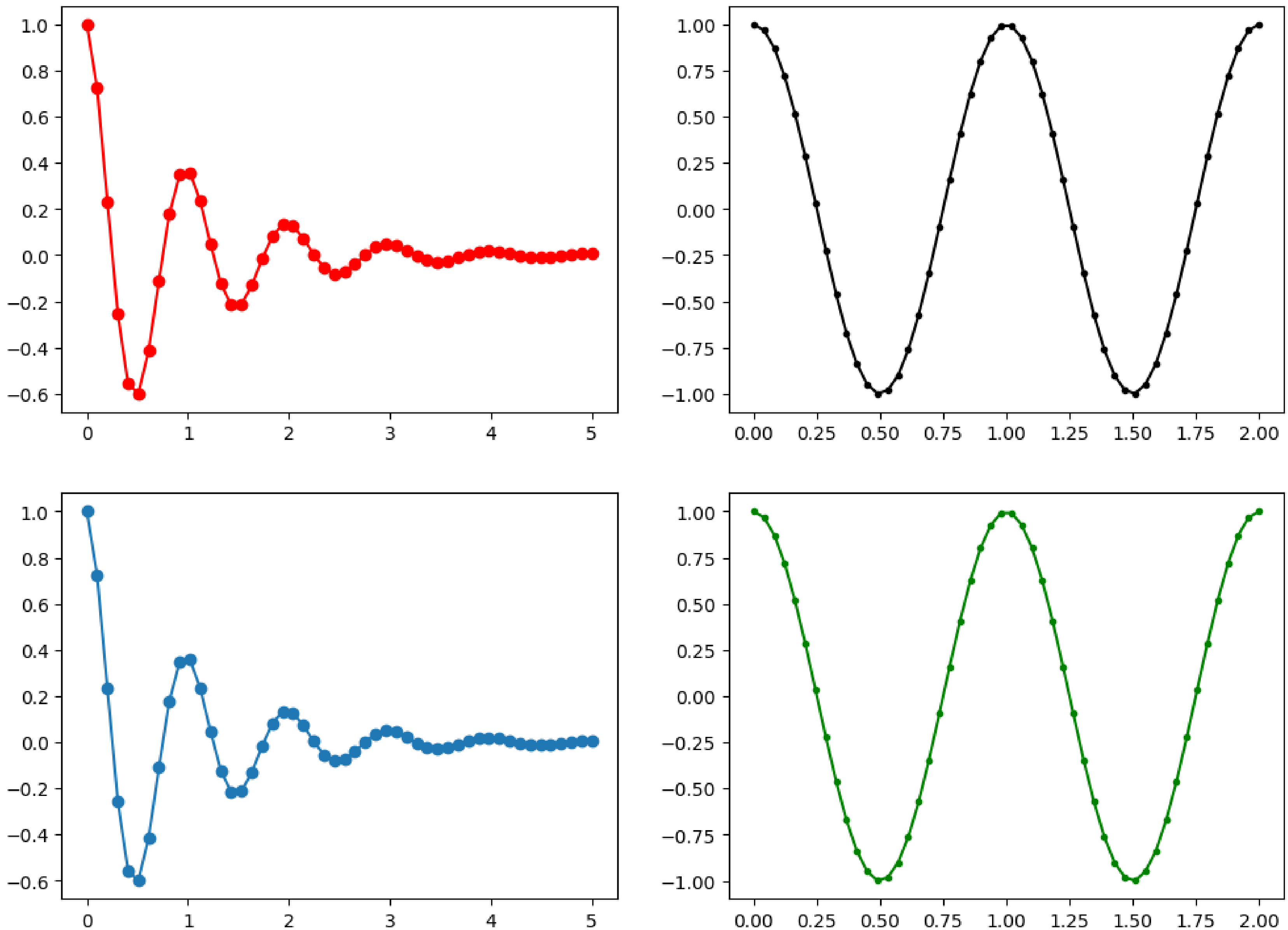
Syntax:

- matplotlib.pyplot.subplot(nrows, ncols, index, **kwargs)

- **Example**

In [15]:
```
# Generate data
x1 = np.linspace(0.0, 5.0)
x2 = np.linspace(0.0, 2.0)

y1 = np.cos(2 * np.pi * x1) * np.exp(-x1)
y2 = np.cos(2 * np.pi * x2)
```

In [16]:
```
# Subplot 2 x 2
figure(figsize=(12, 9), dpi=100, facecolor='w');

# One way to specify the location of the plot
subplot(2, 2, 1);
plot(x1, y1, 'ro-');

subplot(2, 2, 2);
plot(x2, y2, 'k.-');

# Another way to specify the location of the plot
subplot(223);
plot(x1, y1, 'o-');

subplot(224);
plot(x2, y2, 'g.-');
```



References

- [Matplotlib (https://matplotlib.org/tutorials/index.html)](https://matplotlib.org/tutorials/index.html)