Sampling Methods

Computational Methods, Dec. 2018, Kenji Doya

Some computations involve random numbers, such as simulating stochastic processes or searching in a high dimension space.

References:

- Christopher M. Bishop: Pattern Recognition and Machine Learning, Chapter 11 Sampling methods. Springer, 2006.
- Jun S. Liu: Monte Carlo Strategies in Scientific Computing. Springer, 2004.
- NumPy Reference: 3.25 Random sampling (numpy.random)

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Random Numbers

How can we generate *random* numbers from deterministic digital computers? They are, of course, *pseudo* random numbers.

The classic way of generating pseudo random numbers is *Linear Congruent Method* using a sequential dynimcs:

$$x_{i+1} = ax_i + b \mod m.$$

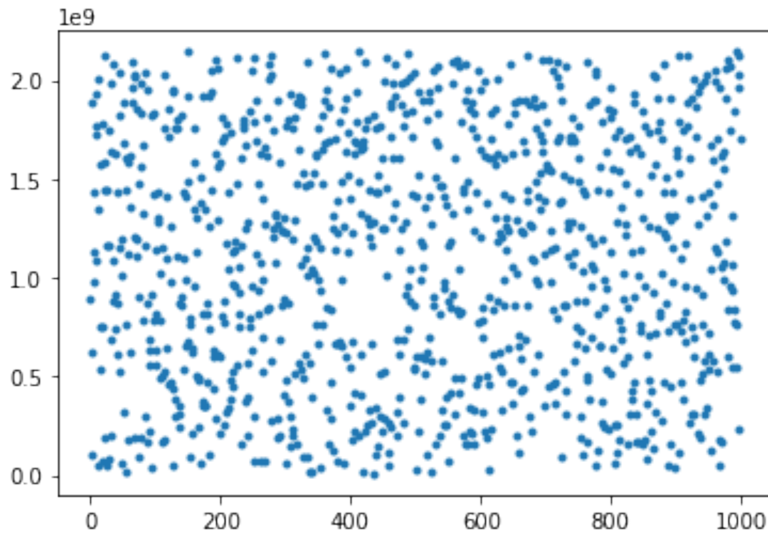
Below is what was long used in the Unix system.

In [2]:
```
def lcm(n=1, seed=-1726662223):
    """Random integers <2**31 by linear congruent method"""
    a = 1103515245
    b = 12345
    c = 0x7fffffff  # 2**31 - 1
    x = np.zeros(n+1, dtype=int)
    x[0] = seed
    for i in range(n):
        x[i+1] = (a*x[i] + b) & c  # bit-wise and
    return(x[1:])
```

In [3]:
```
x = lcm(1000)
```

In [4]:
```
1 plt.plot(x, '.')
```

Out[4]: [<matplotlib.lines.Line2D at 0x11f55d588>]



A common problem with LCM is that lower digits can fall in simple cycles.

In [5]:
```
1 x[:100]%4
```

Out[5]:
```
array([2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2
, 3,
       0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0
, 1,
       2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2
, 3,
       0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0
, 1,
       2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0, 1])
```

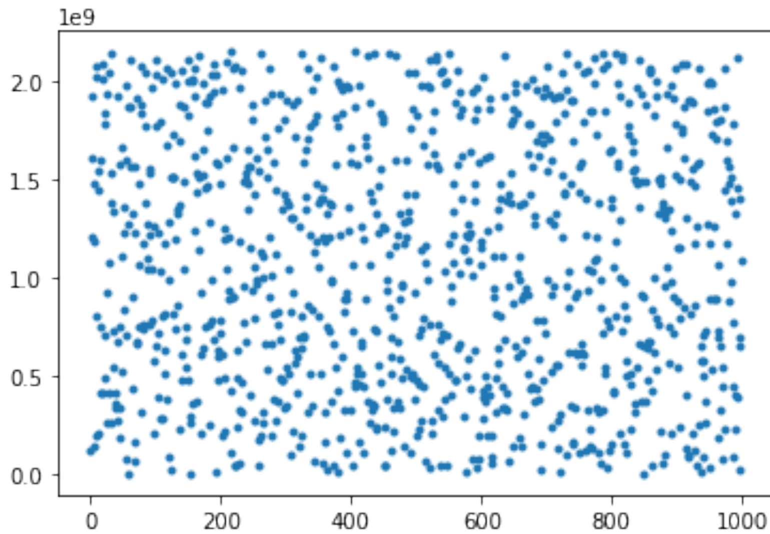
Also the samples can form crystal-like structure when consecutive numbers are taken as a vector.

`numpy.random` uses an advanced method called *Mersenne Twister* which overcomes these problems.

In [6]:
```
1 y = np.random.randint(2**31, size=1000)
```

In [7]:
```
1 plt.plot(y, '.')
```

Out[7]: [<matplotlib.lines.Line2D at 0x11f65f5c0>]



In [8]:
```
1 y[:100]%4
```

Out[8]:
```
array([0, 2, 0, 2, 1, 0, 2, 3, 3, 0, 1, 3, 0, 2, 3, 0, 0, 0, 1, 1, 0
, 1,
       2, 1, 1, 1, 2, 0, 2, 1, 0, 2, 2, 3, 0, 2, 3, 1, 3, 0, 2, 3, 1
, 1,
       3, 1, 1, 2, 0, 3, 0, 3, 0, 1, 2, 2, 1, 2, 1, 0, 3, 1, 0, 2, 3
, 3,
       1, 3, 2, 3, 1, 0, 3, 2, 3, 1, 0, 3, 3, 1, 2, 0, 0, 3, 0, 3, 0
, 2,
       0, 1, 0, 2, 0, 3, 2, 0, 2, 1, 1, 2])
```

Integrating area/volume

Uniform random numbers can be used to approximate integration

$$\int_V f(x)dx \simeq \frac{|V|}{n}\sum_{i=1}^{n} f(x_i)$$

by uniform samples x_i in a volume V

In [9]:
```
# Sphere in 3D
def sphere(x):
    """height of a half sphere"""
    h2 = 1 - x[0]**2 - x[1]**2
#     if h2>0:
#         y = np.sqrt(h2)
#     else:
#         y = 0
#     return(y)
    return(np.sqrt((h2>0)*h2))
```

In [10]:
```
sphere([0.5,0.9])
```

Out[10]: -0.0

In [11]:
```
m = 10000
x = np.random.random((2, m)) # one quadrant
v = np.sum(sphere(x))/m
8*v
```

Out[11]: 4.176319464033466

In [12]:
```
4/3*np.pi
```

Out[12]: 4.1887902047863905

In [13]:
```
# n-dimensional sphere
def nsphere(x):
    """height of a half sphere in n-dim"""
    h2 = 1 - np.sum(x**2, axis=0)
    return(np.sqrt((h2>0)*h2))
```

In [14]:
```
m = 1000000
for n in range(2,20):
    x = np.random.random((n-1, m)) # one quadrant
    v = np.sum(nsphere(x))/m
    print(2**n*v)
```

```
3.140703861752272
4.191826820215456
4.92737040600436
5.274035595313263
5.153678387157588
4.738555180211118
4.035733640959217
3.2894179115193753
2.5478728656881393
1.8589900159998796
1.3060975811715394
0.9501054471565537
0.578621375844066
0.4474387450402334
0.23188631160668186
0.28255985107146414
0.1307365045541965
0.0
```

In [ ]:
```

```

Non-uniform Distributions

How can we generate samples following a non-uniform distribution $p(x)$?

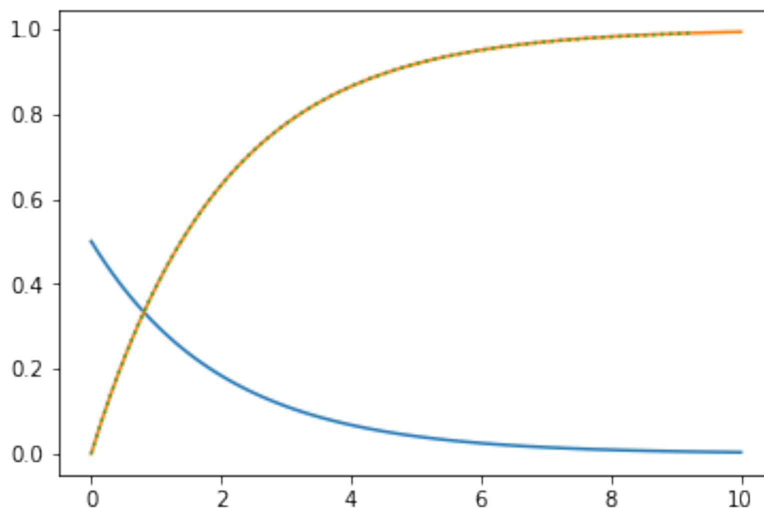
If the cumulative density function $f(x) = \int_{-\infty}^{x} p(u)du$ is known, we can map uniformly distributed samples $y_i \in [0, 1]$ to $x_i = f^{-1}(y_i)$.

Exponential distribution

$$p(x; \mu) = \frac{1}{\mu} e^{-\frac{x}{\mu}}$$
$$f(x; \mu) = 1 - e^{-\frac{x}{\mu}}$$

In [15]:
```
# Exponential distribution in [0,infinity)
def p_exp(x, mu=1):
    """density function of exponential distribution"""
    return(np.exp(-x/mu)/mu)
def f_exp(x, mu=1):
    """cumulative density function of exponential distribution"""
    return(1 - np.exp(-x/mu))
def finv_exp(y, mu=1):
    """inverse of cumulative density function of exponential distril
    return(-mu*np.log(1 - y))
x = np.linspace(0, 10, 100)
plt.plot(x, p_exp(x, 2))
plt.plot(x, f_exp(x, 2))
y = np.arange(0, 1, 0.01)
plt.plot(finv_exp(y, 2), y, ':')
```

Out[15]: [<matplotlib.lines.Line2D at 0x10f0f96a0>]



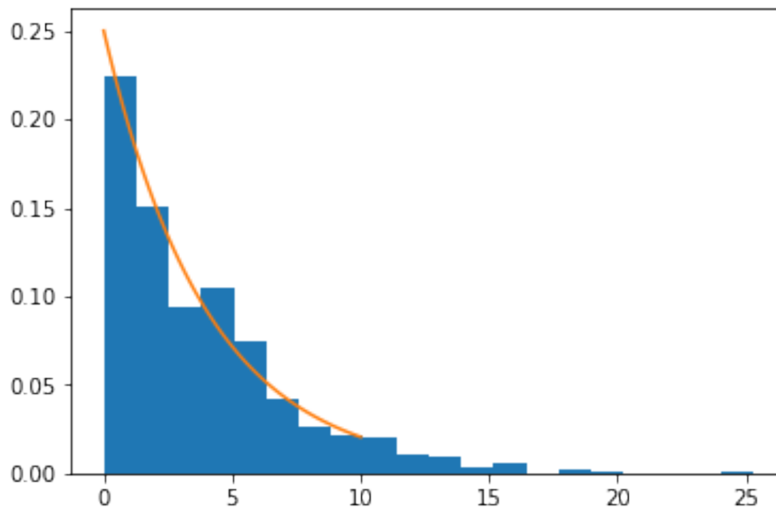
In [16]:
```
def x_exp(n=1, mu=1):
    """sample from exponential distribution"""
    ys = np.random.random(n)  # uniform in [0,1]
    return(finv_exp(ys, mu))
```

In [17]:
```
x_exp(1, 2)
```

Out[17]: array([0.63579029])

In [18]:
```
1 xs = x_exp(1000, 4)
2 plt.hist(xs, bins=20, density=True)
3 plt.plot(x, p_exp(x, 4))
```

Out[18]: [<matplotlib.lines.Line2D at 0x11f6a5c88>]



In [ ]:
```
1
```

When a stochastic variable x following the distribution $p(x)$ is transformed by $y = g(x)$, the distribution of y is given by

$$p(y) = \left|\frac{\partial x}{\partial y}\right| p(x)$$

where | | means the absolute value for a scalar derivative and the determinant for a Jacobian matrix.

Normal distribution

A common way of generating a normal (Gaussian) distribution

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

is to consider a 2D normal distribution

$$p(x_1, x_2) = p(x_1)p(x_2) = \frac{1}{2\pi} e^{-\frac{1}{2}(x_1^2 + x_2^2)}.$$

For a polar coordinate tranformation

$$\begin{aligned} x_1 &= r\cos\theta \\ x_2 &= r\sin\theta, \end{aligned}$$

the Jacobian is

$$\frac{\partial(x_1, x_2)}{\partial(r, \theta)} = \begin{pmatrix} \cos\theta & -r\sin\theta \\ \sin\theta & r\cos\theta \end{pmatrix}$$

and its determinant is

$$\det \frac{\partial(x_1, x_2)}{\partial(r, \theta)} = r\cos^2\theta + r\sin^2\theta = r.$$

Thus we have the relationship

$$p(r, \theta) = rp(x_1, x_2) = \frac{r}{2\pi} e^{-\frac{r^2}{2}}.$$

By further transforming $u = r^2$, from $\frac{du}{dr} = 2r$, we have

$$p(u, \theta) = \frac{1}{2r} p(r, \theta) = \frac{1}{4\pi} e^{-\frac{u}{2}}.$$

Thus we can sample u by exponential distribution $p(u) = \frac{1}{2} e^{-\frac{u}{2}}$ and θ by uniform distribution in $[0, 2\pi)$, and then transform them to x_1 and x_2 to generate two samples following normal distribution.

This is known as Box-Muller method.

In [19]:

```
1 def box_muller(n=1):
2     """Generate 2n gaussian samples"""
3     u = x_exp(n, 2)
4     r = np.sqrt(u)
5     theta = 2*np.pi*np.random.random(n)
6     x1 = r*np.cos(theta)
7     x2 = r*np.sin(theta)
8     return(np.hstack((x1,x2)))
```

In [20]:

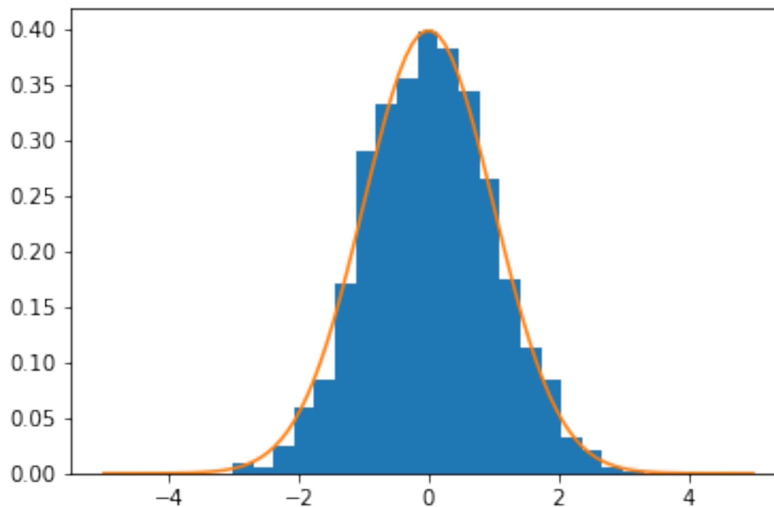
```
1 box_muller(1)
```

Out[20]:

```
array([-1.08980556,  1.90010887])
```

In [21]:
```
1 xs = box_muller(1000)
2 plt.hist(xs, bins=20, density=True)
3 # check how the histogram fits the pdf
4 x = np.linspace(-5, 5, 100)
5 px = np.exp(-x**2/2)/np.sqrt(2*np.pi)
6 plt.plot(x, px)
```

Out[21]: [<matplotlib.lines.Line2D at 0x1206bd860>]



In [ ]:
```
1
```

Rejection Sampling

What can we do when transformations from uniform distribution is not available?

Let us find a *proposal distribution* $q(x)$ for which samples are easily generated and covers the target distribution as $p(x) \leq cq(x)$ with a scaling constant $c > 0$.
Then take a sample from $q(x)$ and accept it with probability $\frac{p(x)}{cq(x)}$.

Gamma distribution

Gamma distribution is an extension of exponential distribution defined as

$$p(x; k, \theta) = \frac{1}{\Gamma(k)\theta^k} x^{k-1} e^{-\frac{x}{\theta}}$$

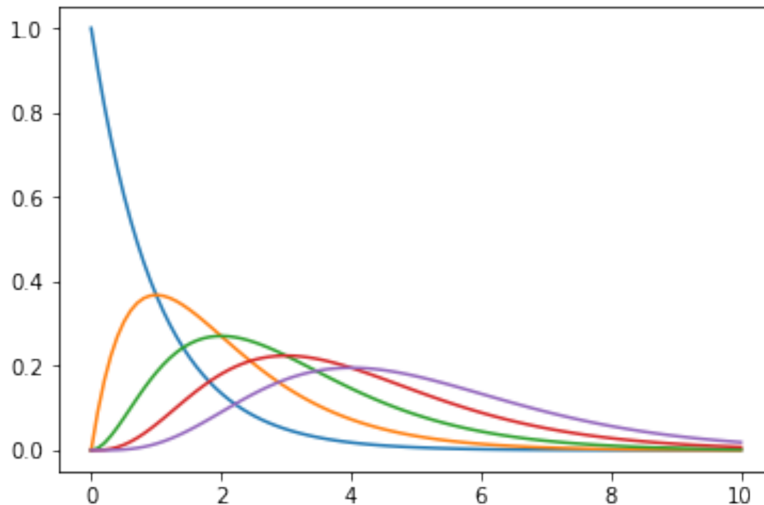
with the shape parameter $k > 0$ and the scaling parameter θ.

$\Gamma(k)$ is the *gamma function*, which is a generalization of factorial, $\Gamma(k) = (k-1)!$ for an integer k.

Let us generate samples from gamma distribution with integer k and $\theta = 1$.

In [23]:
```
def p_gamma(x, k=1):
    """gamma distribution with integer k"""
    return(x**(k-1)*np.exp(-x)/np.prod(range(1,k)))
```

In [24]:
```
x = np.linspace(0, 10, 100)
for k in range(1, 6):
    plt.plot(x, p_gamma(x, k))
```



Consider the exponential distribution $q(x;\mu)=\frac{1}{\mu}e^{-\frac{x}{\mu}}$ as the proposal distribution. For

$$\frac{p(x;k)}{q(x;\mu)}=\frac{\mu}{(k-1)!}x^{k-1}e^{-(1-\frac{1}{\mu})x},$$

we set

$$\frac{d}{dx}\frac{p(x;k)}{q(x;\mu)}=0$$

and have

$$\left((k-1)x^{k-2}+\frac{1-\mu}{\mu}x^{k-1}\right)e^{\frac{1-\mu}{\mu}x}=0,$$

$$x=\frac{\mu(k-1)}{\mu-1},$$

where $\frac{p(x;k)}{q(x;\mu)}$ takes the maximum

$$\frac{\mu^k}{(k-1)!}\left(\frac{k-1}{\mu-1}\right)^{k-1}e^{1-k}.$$

By taking $\mu=k$,

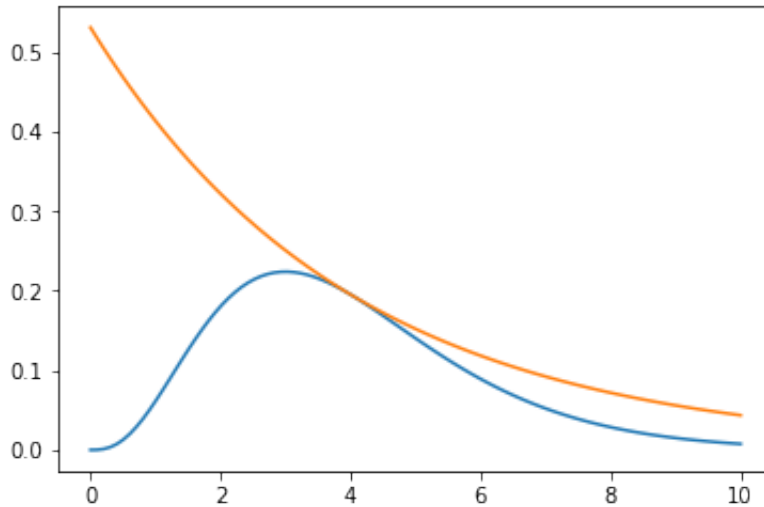
$$c=\frac{k^k}{(k-1)!}e^{1-k}$$

satisfies $p(x)\leq cq(x)$

In [25]:

```
k = 4
c = (k**k)/np.prod(range(1,k))*np.exp(1-k)
#print(c)
x = np.linspace(0, 10, 100)
plt.plot(x, p_gamma(x, k))
plt.plot(x, c*p_exp(x, k))
```

Out[25]: [<matplotlib.lines.Line2D at 0x12096c2b0>]

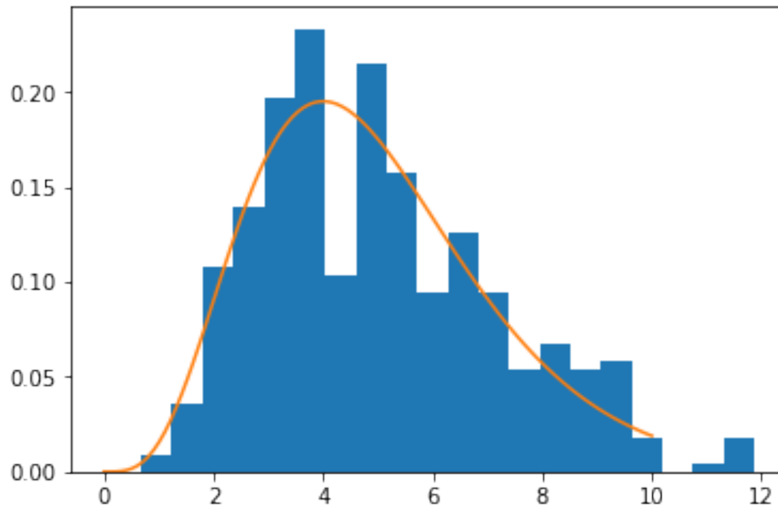


In [26]:

```
def x_gamma(n=1, k=1):
    """sample from gamma distribution by rejection sampling"""
    c = (k**k)/np.prod(range(1,k))*np.exp(1-k)
    #print("c =", c)
    xe = x_exp(n, k)
    paccept = p_gamma(xe, k)/(c*p_exp(xe, k))
    accept = np.random.random(n)<paccept
    xg = xe[accept]  # rejection sampling
    #print("c =", c, "; acceptance rate =", len(xg)/n)
    return(xg)
```

In [27]:
```
1 k = 5
2 xs = x_gamma(1000, k)
3 plt.hist(xs, bins=20, density=True)
4 plt.plot(x, p_gamma(x, k))
```

Out[27]: [<matplotlib.lines.Line2D at 0x120915390>]



In [ ]:
```
1
```

Importance Sampling

One probelm of rejection sampling is that we may end up rejecting many of the samples when we cannot find a good proposal distribution.

When the aim is not just to take samples from $p(x)$ but to take the *expectation* of a function $h(x)$ with respect to the distribution $p(x)$

$$E_p[h(x)] = \int h(x)p(x)dx,$$

we can use the ratio between the target and proposal distributions to better utilize the samples.

$$E_p[h(x)] = \int h(x)p(x)dx = \int h(x)\frac{p(x)}{q(x)}q(x)dx = E_q\left[\frac{p(x)}{q(x)}h(x)\right]$$

Mean and variance

Let us consider taking the mean

$$\mu = E_p[x] = \int x p(x) dx$$

and the variance

$$\sigma^2 = E_p[(x-\mu)^2] = \int (x-\mu)^2 p(x) dx$$

for the gamma distribution.

In [28]:
```
def mv_exp(n=1, mu=1):
    """mean and variance of exponential distribution"""
    x = x_exp(n, mu)
    mean = np.mean(x)
    var = np.var(x)
    return(mean, var)
```

In [29]:
```
mv_exp(1000, 2)
```

Out[29]: (2.0101584371883985, 3.7220432195225905)

In [30]:
```
def mv_gamma(n=1, k=1):
    """mean and variance of gamma distribution by rejection sampli
    x = x_gamma(n, k)  # by rejection sampling
    mean = np.mean(x)
    var = np.var(x)
    return(mean, var)
```

In [31]:
```
mv_gamma(100, 2)
```

Out[31]: (1.94703603666266, 1.866852743253504)

In [32]:
```
def mv_gamma_is(n=1, k=1):
    """mean and variance of gamma distribution by importance sampli
    x = x_exp(n, k)  # sample by exponential distribution
    importance = p_gamma(x, k)/p_exp(x, k)
    mean = np.dot(importance, x)/n
    var = np.dot(importance, (x-mean)**2)/(n - 1)
    return(mean, var)
```

In [33]:
```
mv_gamma_is(100, 2)
```

Out[33]: (1.8522023534867502, 2.025685261012499)

Compare the variability of the estimate of the mean.

In [34]:
```
m = 100
n = 500
k = 2
means = np.zeros((m, 2))
for i in range(m):
    means[i,0], var = mv_gamma(n, k)
    means[i,1], var = mv_gamma_is(n, k)
print("RS: ", np.mean(means[:,0]), np.var(means[:,0]))
print("IS: ", np.mean(means[:,1]), np.var(means[:,1]))
```

```
RS:  2.010376153266712 0.005855418735112626
IS:  2.0040024711068374 0.0038048718097131516
```

In [ ]:
```

```

Exercise

1. Integration

When an explicit function like $x_n = g(x_1, \ldots, x_{n-1})$ is not available, you can use a binary function that takes 1 within and 0 outside of a region to measure its volume.

1) Measure the volume of an n-dimensional sphere using a binary function

$$f(x) = \begin{cases} 1 & \text{if} \quad \sum_{i=1}^{n} x_i^2 \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

In [ ]:
```

```

2. Rejection Sampling

Let us generate samples from Gamma distribution

$$p(x; k, \theta) = \frac{1}{\Gamma(k)\theta^k} x^{k-1} e^{-\frac{x}{\theta}}$$

with the shape parameter $k > 0$ and the scaling parameter θ using the gamma function $\Gamma(k)$ available in `scipy.special`.

In [ ]:
```
from scipy.special import gamma
from scipy.special import factorial
```

See how the gamma function looks like and compare with factorial at integer values.

In [ ]:

```
1 kmax = 5
2 x = np.linspace(0, kmax, 50)
3 plt.plot(x, gamma(x))
4 plt.plot(range(1,kmax+1), [factorial(k-1) for k in range(1,kmax+1)
5 plt.xlabel("x"); plt.ylabel("f(x)")
```

1) Define the Gamma density function with arbitrary k and θ.

In [ ]:

```
1 def p_gamma(x, k=1, theta=1):
2     """density function of gamma distribution"""
3     return
```

Consider the exponential distribution $q(x;\mu) = \frac{1}{\mu}e^{-\frac{x}{\mu}}$ as the proposal distribution.

In [ ]:

```
1 def p_exp(x, mu=1):
2     """density function of exponential distribution"""
3     return
4 
5 def x_exp(n=1, mu=1):
6     """sample from exponential distribution"""
7     y = np.random.random(n)  # uniform in [0,1]
8     return
```

The ratio of the target and proposal distributions is

$$\frac{p(x;k,\theta)}{q(x;\mu)} = \frac{\mu}{\Gamma(k)\theta^k}x^{k-1}e^{(\frac{1}{\mu}-\frac{1}{\theta})x}.$$

By setting

$$\frac{d}{dx}\frac{p(x;k,\theta)}{q(x;\mu)} = 0$$

we have

$$\{(k-1)x^{k-2} + (\frac{1}{\mu}-\frac{1}{\theta})x^{k-1}\}e^{(\frac{1}{\mu}-\frac{1}{\theta})x} = 0.$$

Thus at

$$x = \frac{(k-1)\mu\theta}{\mu-\theta}$$

the ratio $\frac{p(x;k,\theta)}{q(x;\mu)}$ takes the maximum

$$\frac{\mu^k}{\Gamma(k)\theta}\left(\frac{k-1}{\mu-\theta}\right)^{k-1}e^{1-k}.$$

2) What is a good choice of μ to satisfy $p(x) \leq cq(x)$ and what is the value of c for that?

By setting $\mu = k\theta$, we have $p(x) \leq cq(x)$ with $c = \frac{k^k}{\Gamma(k)}e^{1-k}$

3) Verify that $cq(x)$ covers $p(x)$ by plotting them for some k and θ.

In [ ]:
```
k = 2
theta = 2
c = (k**k)/gamma(k)*np.exp(1-k)
x = np.linspace(0, 10, 50)
plt.plot(x, p_gamma(x, k, theta))
plt.plot(x, c*p_exp(x, k*theta))
plt.xlabel("x");
```

4) Implement a function to general samples from Gamma distribution with arbitorary k and θ using rejection sampling from exponential distribution.

In [ ]:
```
def x_gamma(n=1, k=1, theta=1):
    """sample from gamma distribution by rejection sampling"""
    c = (k**k)/gamma(k)*np.exp(1-k)
    #print("c =", c)
    xe = x_exp(n, k*theta)
    paccept = 
    accept = np.random.random(n)<paccept
    xg = xe[accept]  # rejection sampling
    #print("accept rate =", len(xg)/n)
    return(xg)
```

In [ ]:
```
k = 2
theta = 2
# sample histogram
xs = x_gamma(1000, k, theta)
plt.hist(xs, bins=20, density=True)
# compare with the density function
x = np.linspace(0, 10, 100)
plt.plot(x, p_gamma(x, k, theta))
plt.xlabel("")
```

In [ ]:
```

```

3. Importance Sampling

You have m sample values $f(x_i)$ $(i = 1, \dots m)$ at x_i following a normal distribution

$$q(x; \mu_0, \sigma_0) = \frac{1}{\sqrt{2\pi\sigma_0^2}} e^{-\frac{(x-\mu_0)^2}{2\sigma_0^2}}.$$

Consider estimating the mean of $f(x)$ for samples with a different normal distribution $p(x; \mu_1, \sigma_1)$ by importance sampling

$$E_p[h(x)] = E_q\left[\frac{p(x)}{q(x)} h(x)\right]$$

1) What is the importance weight $\frac{p(x)}{q(x)}$?

Type *Markdown* and LaTeX: α^2

2) Generate $m = 100$ samples with $\mu_0 = 100$ and $\sigma_0 = 20$, compute values of $f(x) = x$, and take the sample mean $E_q[f(x)]$.

In [ ]:

```
m = 100
mu0 = 100
sig0 = 20
xs = 
# show histogram
plt.hist(xs, bins=20, density=True)
plt.xlabel("x")
# check the sample mean
np.mean(xs)
```

3) Estimate the mean $E_p[f(x)]$ for $\mu_1 = 120$ and $\sigma_1 = 10$ by importance sampling.

In [ ]:

```
mu1 = 120
sig1 = 20
importance = 
mean1 = np.dot(importance, xs)/m
print(mean1)
```

Optional) See how the result changes with different settings of μ_1, σ_1 and sample size m.

In [ ]:

```

```

In [ ]:

```

```