Jupyter Notebook

December 8, 2021

[161]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
```

[162]:
```
def gaussian_pdf(x, mu=0, sigma=1):
        u = (x - mu) / abs(sigma)
        y = (1 / (np.sqrt(2 * np.pi) * abs(sigma)))
        return y * np.exp(-0.5 * u**2)
```

[163]:
```
def uniform_pdf(x):
    return 1 * ((x >= 0) & (x <= 1))
```

[164]:
```
def kernel_smoothed_density(x_values, samples, width=0.3,␣
 ↪kernel_function=gaussian_pdf):
    # Generate an array of kernel values centred on the samples
    kernel_values = [kernel_function(x_value, samples, width) for x_value in␣
 ↪x_values]
    return np.average(kernel_values, axis=1)
```

[165]:
```
# General global variables
color1 = 'steelblue'
color2 = 'firebrick'
color3 = 'seagreen'
color4 = 'darkorange'
```

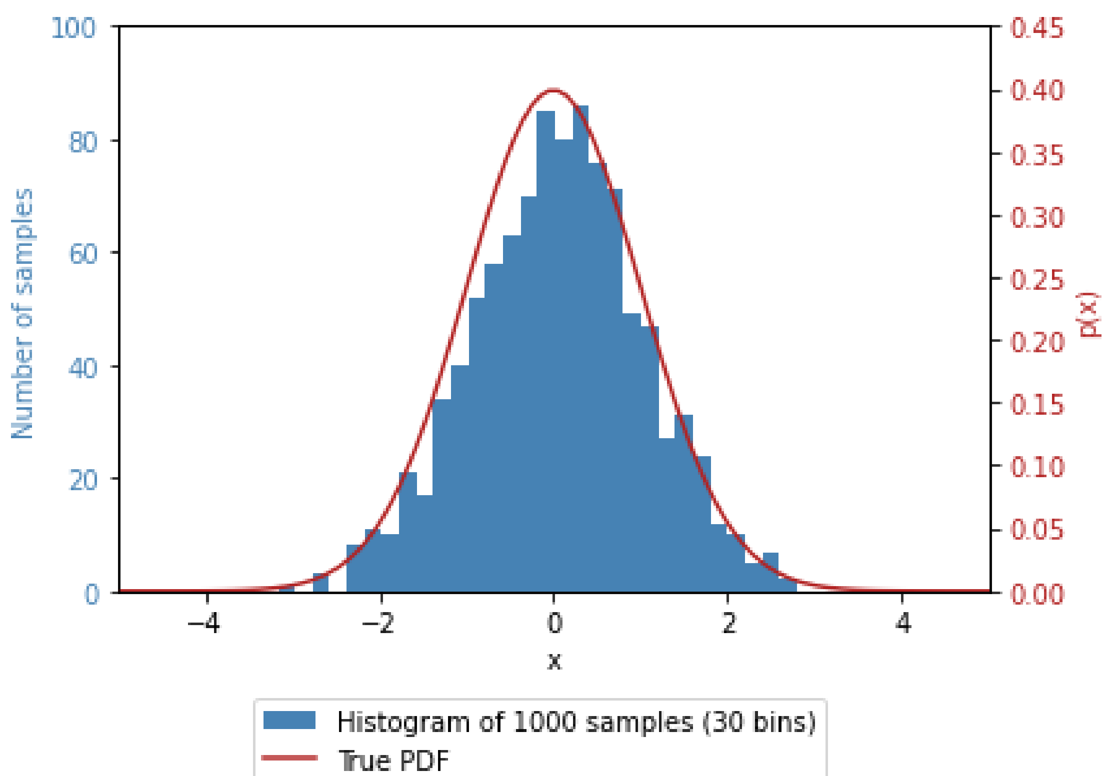
0.0.1 Section 1: Uniform and normal random variables

[166]:
```
X_gaussian = np.random.randn(10000)
X_uniform = np.random.rand(10000)
```

Histograms of random samples compared with true pdf

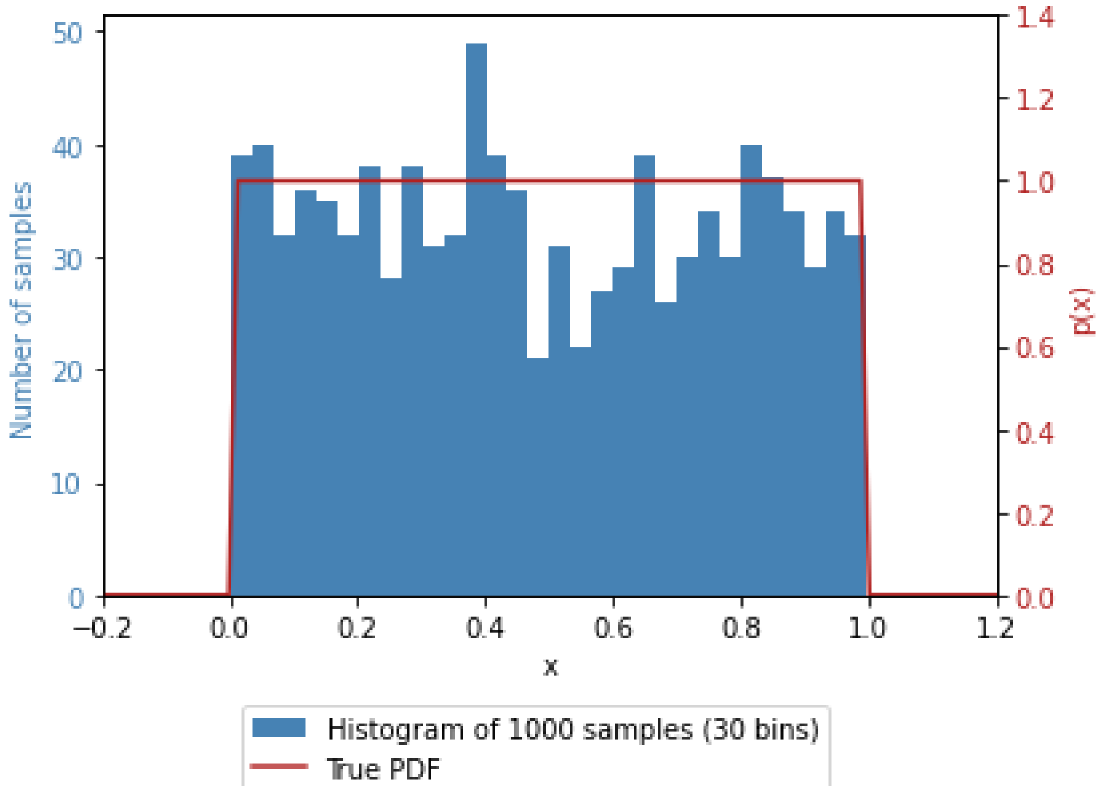
[167]:
```
fig, ax1 = plt.subplots()

ax1.hist(X_gaussian[:1000], bins=30, color=color1, label='Histogram of 1000␣
 ↪samples (30 bins)')
ax1.set_xlabel('x')
```

```
ax1.set_ylabel('Number of samples', color=color1)
ax1.tick_params(axis='y', labelcolor=color1)
ax1.set_ylim(0, 100)


ax2 = ax1.twinx()
x = np.linspace(-5, 5, 100)
ax2.plot(x, gaussian_pdf(x), color=color2, label='True PDF')
ax2.set_ylabel('p(x)', color=color2)
ax2.tick_params(axis='y', labelcolor=color2)
ax2.set_ylim(0, 0.45)
ax2.set_xlim(-5, 5)

fig.legend(loc='upper center', bbox_to_anchor=(0.5, 0))

plt.savefig('figures/gaussian_histogram_and_pdf.png', bbox_inches='tight')
```



[168]:
```
fig, ax1 = plt.subplots()

x = np.linspace(-0.2, 1.2, 100)
```

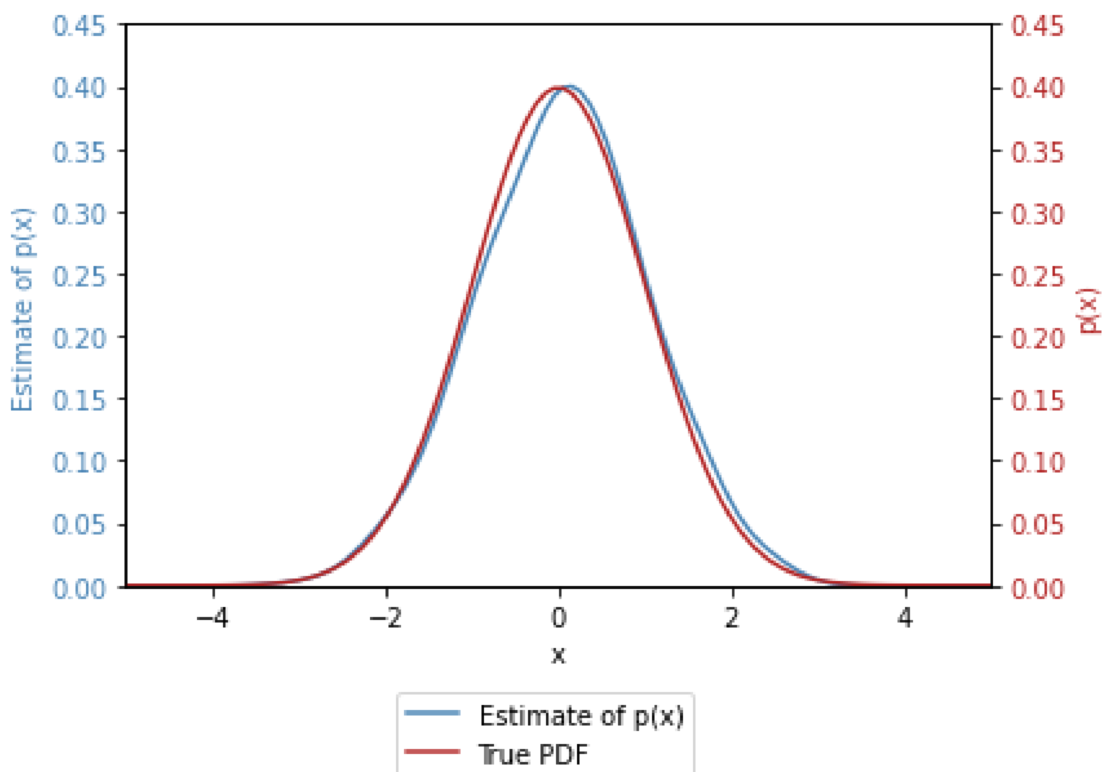
```
ax1.hist(X_uniform[:1000], bins=30, color=color1, label='Histogram of 1000␣
↪samples (30 bins)')
ax1.set_xlabel('x')
ax1.set_ylabel('Number of samples', color=color1)
ax1.tick_params(axis='y', labelcolor=color1)

ax2 = ax1.twinx()
ax2.plot(x, uniform_pdf(x), color=color2, label='True PDF')
ax2.set_ylabel('p(x)', color=color2)
ax2.tick_params(axis='y', labelcolor=color2)
ax2.set_ylim(0, 1.4)
ax2.set_xlim(-0.2, 1.2)

fig.legend(loc='upper center', bbox_to_anchor=(0.5, 0))

plt.savefig('figures/uniform_histogram_and_pdf.png', bbox_inches='tight')
```



Kernel smoothing

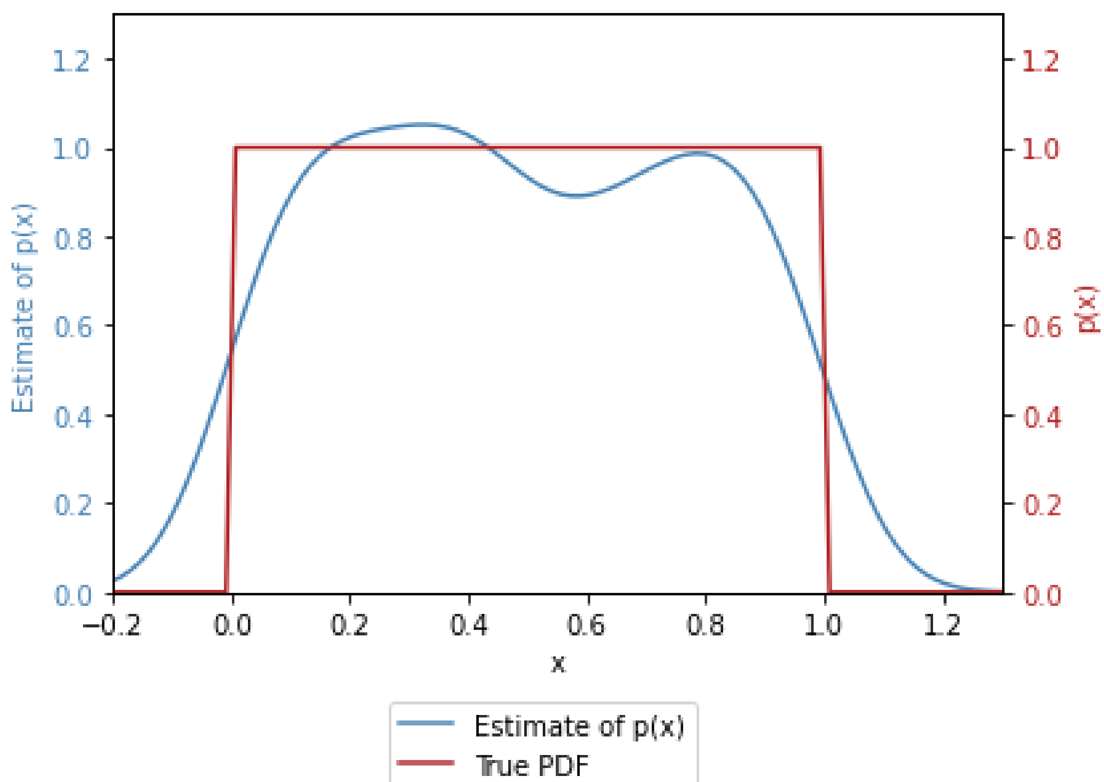
[169]:
```
fig, ax1 = plt.subplots()
```

```
x = np.linspace(-5, 5, 100)

ax1.plot(x, kernel_smoothed_density(x, X_gaussian[:1000]), color=color1,␣
 ↪label='Estimate of p(x)')
ax1.set_xlabel('x')
ax1.set_ylabel('Estimate of p(x)', color=color1)
ax1.tick_params(axis='y', labelcolor=color1)
ax1.set_ylim(0, 0.45)

ax2 = ax1.twinx()
ax2.plot(x, gaussian_pdf(x), color=color2, label='True PDF')
ax2.set_ylabel('p(x)', color=color2)
ax2.tick_params(axis='y', labelcolor=color2)
ax2.set_ylim(0, 0.45)
ax2.set_xlim(-5, 5)

fig.legend(loc='upper center', bbox_to_anchor=(0.5, 0))
plt.savefig('figures/gaussian_kernel_smoothed.png', bbox_inches='tight')
```



[170]:
```
fig, ax1 = plt.subplots()

x = np.linspace(-0.3, 1.3, 100)
```

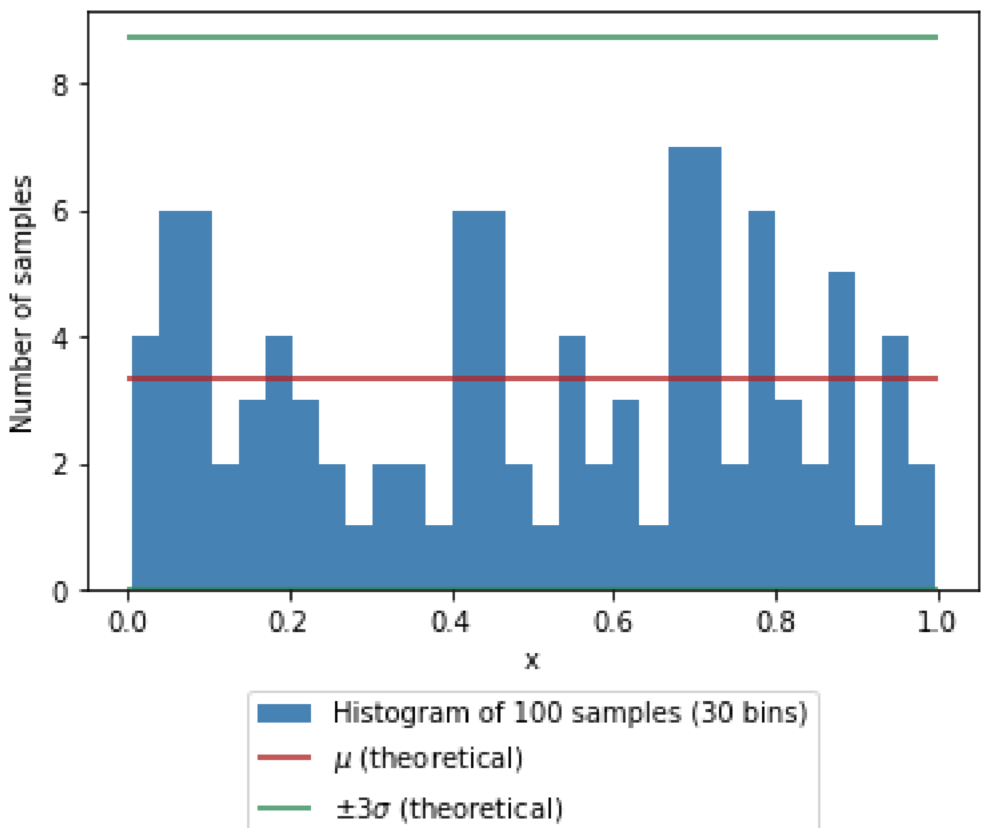
```
ax1.plot(x, kernel_smoothed_density(x, X_uniform[:1000], width=0.1),␣
↪color=color1, label='Estimate of p(x)')
ax1.set_xlabel('x')
ax1.set_ylabel('Estimate of p(x)', color=color1)
ax1.tick_params(axis='y', labelcolor=color1)
ax1.set_ylim(0, 1.3)


ax2 = ax1.twinx()
ax2.plot(x, uniform_pdf(x), color=color2, label='True PDF')
ax2.set_ylabel('p(x)', color=color2)
ax2.tick_params(axis='y', labelcolor=color2)
ax2.set_ylim(0, 1.3)
ax2.set_xlim(-0.2, 1.3)

fig.legend(loc='upper center', bbox_to_anchor=(0.5, 0))
plt.savefig('figures/uniform_kernel_smoothed.png', bbox_inches='tight')
```

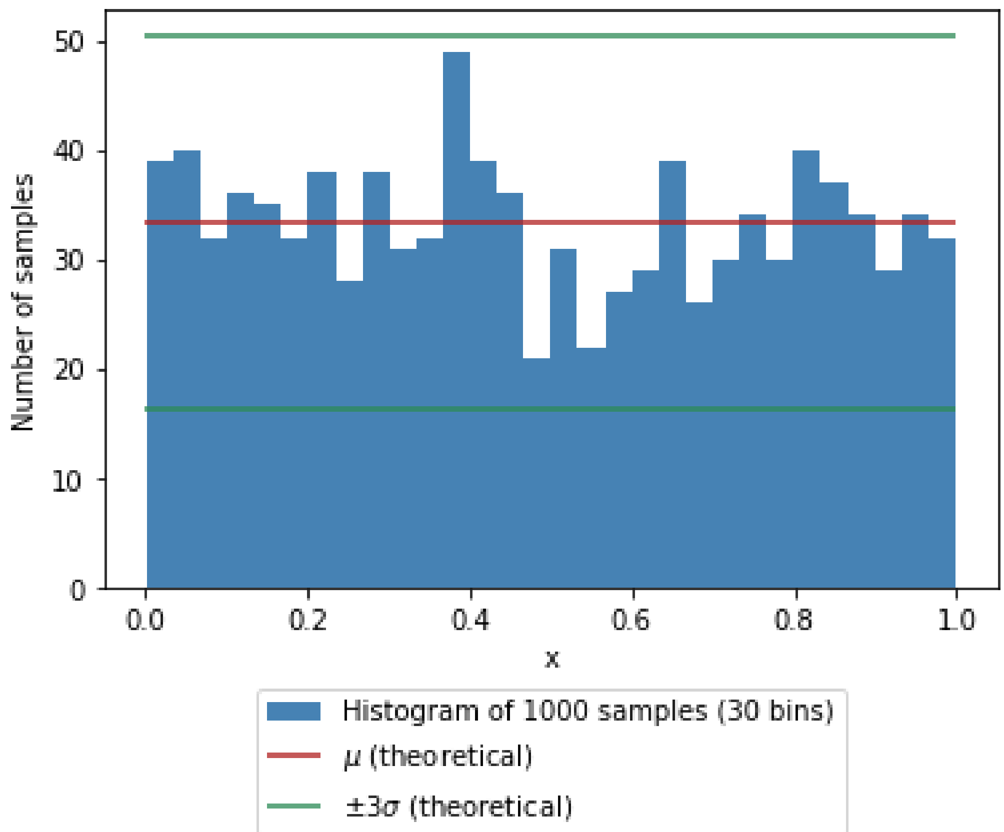


Multinomial theory: Uniform distribution

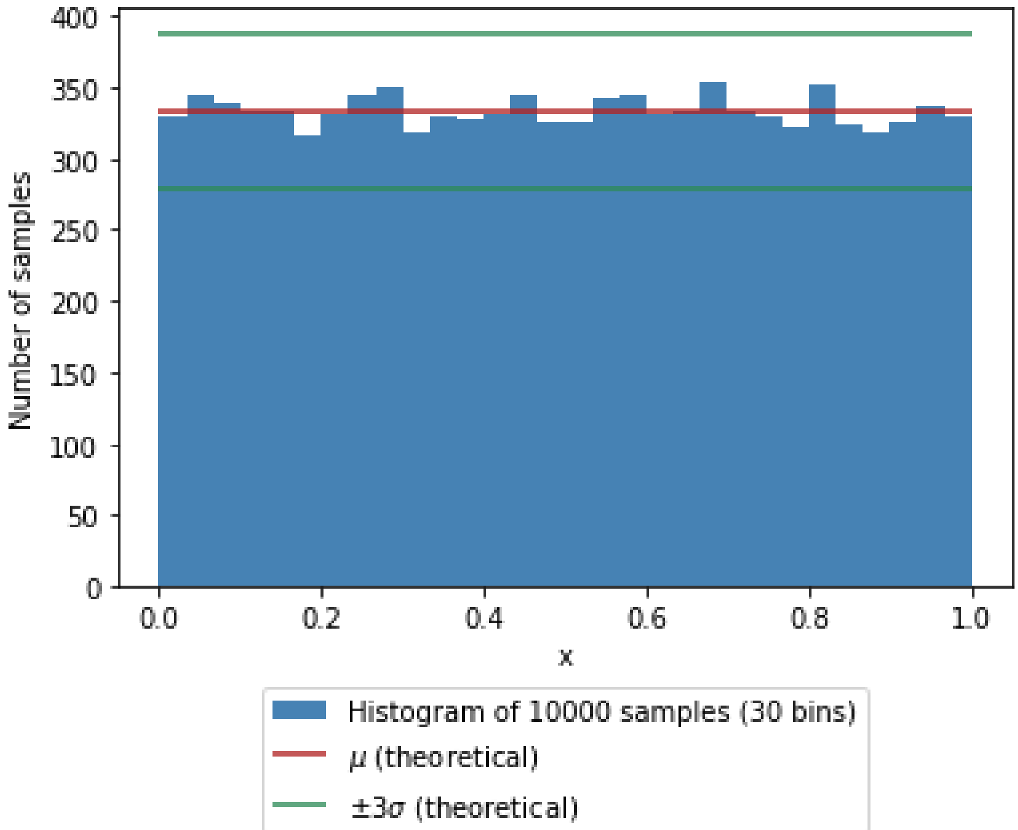
[171]:
```
def plot_uniform_histogram_mean_sd(N, nbins=30):
    plt.figure()

    plt.hist(X_uniform[:N], bins=nbins, color=color1, label=f'Histogram of {N}␣
↪samples ({nbins} bins)')

    bin_width = 1 / nbins
    mean = N * bin_width
    sd = np.sqrt(N * bin_width * (1 - bin_width))
    plt.hlines(y=mean, xmin=0, xmax=1, color=color2, label=r'$\mu$␣
↪(theoretical)')
    plt.hlines(y=[mean + 3 * sd, max(0, mean - 3 * sd)], xmin=0, xmax=1,␣
↪color=color3, label='$\pm 3 \sigma$ (theoretical)')

    plt.xlabel('x')
    plt.ylabel('Number of samples')
    plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.15))
    plt.savefig(f'figures/uniform_histogram_{N}.png', bbox_inches='tight')
```

[172]:
```
plot_uniform_histogram_mean_sd(100)
plot_uniform_histogram_mean_sd(1000)
plot_uniform_histogram_mean_sd(10000)
```





50
40
30
20
10
0
Number of samples
0.0
0.2
0.4
0.6
0.8
1.0
x
Histogram of 1000 samples (30 bins)
μ (theoretical)
$\pm 3\sigma$ (theoretical)

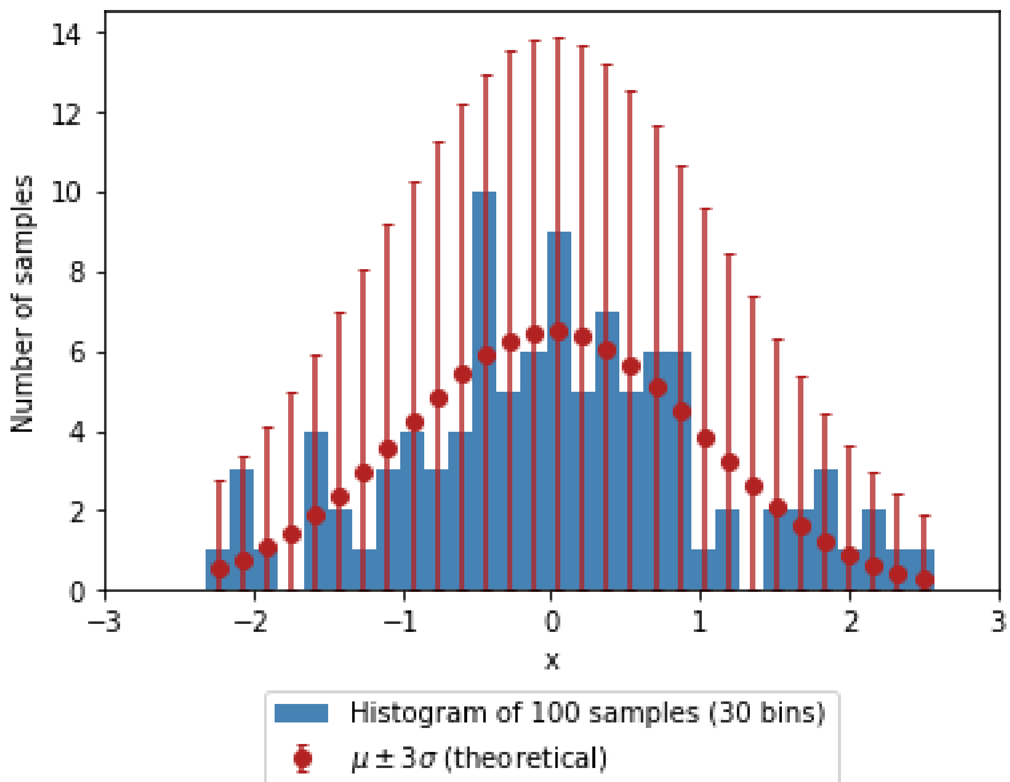


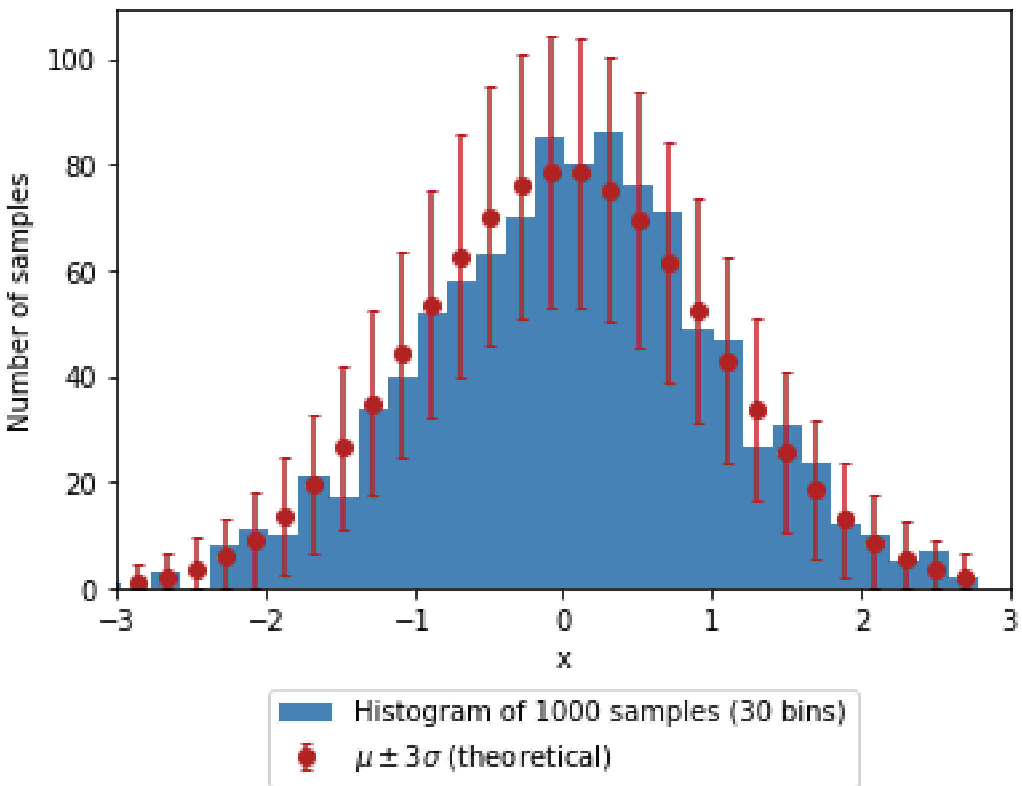
Multinomial theory: Gaussian distribution

[173]:
```
def plot_gaussian_histogram_mean_sd(N, nbins=30):
    plt.figure()

    bin_counts, bin_edges, patches = plt.hist(X_gaussian[:N], bins=nbins,
                                              color=color1, label=f'Histogram␣
 ↪of {N} samples ({nbins} bins)')

    bin_centres = (bin_edges[:-1] + bin_edges[1:]) / 2
    bin_probabilities = norm.cdf(bin_edges[1:]) - norm.cdf(bin_edges[:-1])

    bin_means = N * bin_probabilities
    bin_sds = np.sqrt(N * bin_probabilities * (1 - bin_probabilities))
    upper_errorbar = 3 * bin_sds
    lower_errorbar = upper_errorbar
    lower_errorbar = np.minimum(3*bin_sds, bin_means)
    plt.errorbar(x=bin_centres, y=bin_means,
                 yerr=[lower_errorbar, upper_errorbar],
                 color=color2, fmt='o', capsize=2,
```

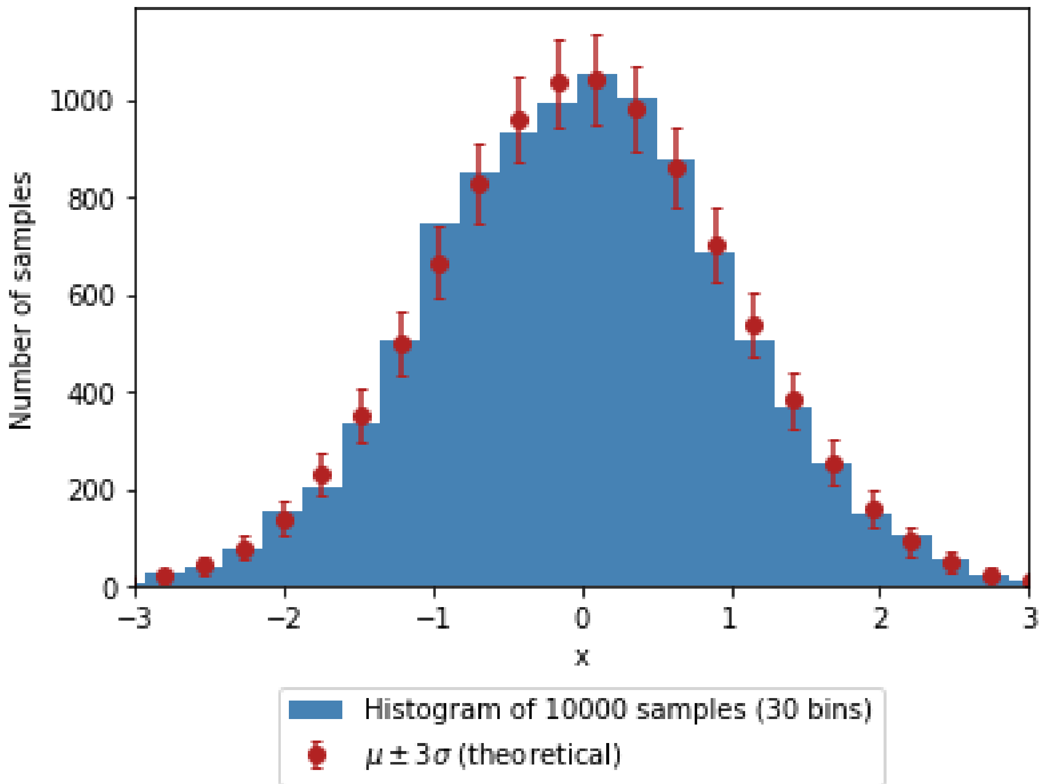
```
                 label=r'$\mu \pm 3\sigma$ (theoretical)')

    plt.xlim(-3, 3)
    plt.xlabel('x')
    plt.ylabel('Number of samples')
    plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.15))
    plt.savefig(f'figures/gaussian_histogram_{N}.png', bbox_inches='tight')
```

[174]:
```
plot_gaussian_histogram_mean_sd(100)
plot_gaussian_histogram_mean_sd(1000)
plot_gaussian_histogram_mean_sd(10000)
```

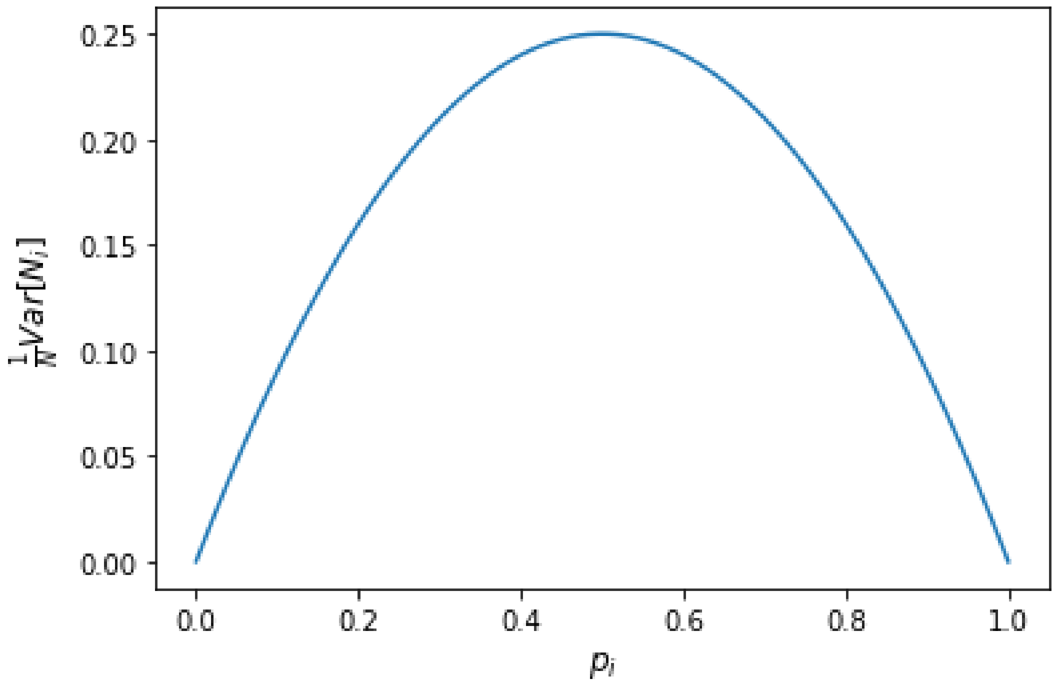




100
80
60
40
20
0
Number of samples
−3
−2
−1
0
1
2
3
x
Histogram of 1000 samples (30 bins)
$\mu \pm 3\sigma$ (theoretical)



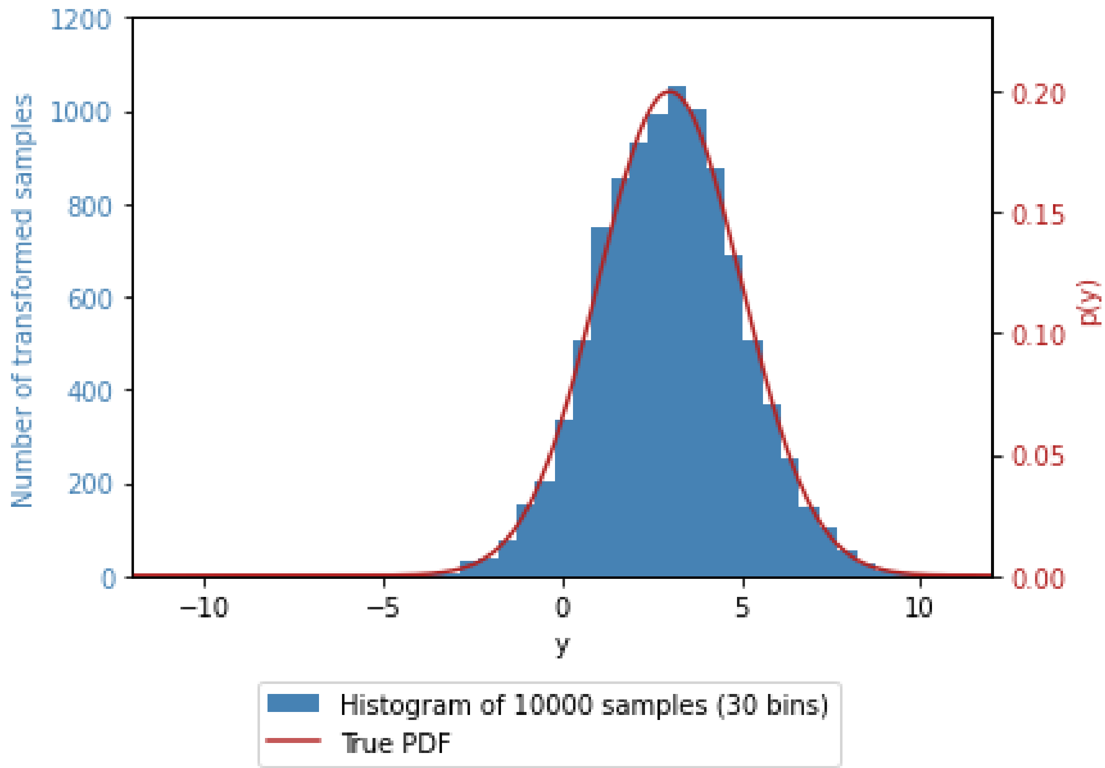
[175]:
```
plt.figure()
x = np.linspace(0, 1, 1000)
plt.plot(x, x*(1 - x))
plt.xlabel(r'$p_i$', fontsize=12)
plt.ylabel(r'$\frac{1}{N}Var[N_i]$', fontsize=12)
plt.savefig(f'figures/histogram_variance.png', bbox_inches='tight')
```



0.0.2 Section 2: Functions of Random Variables

f(x) = ax + b

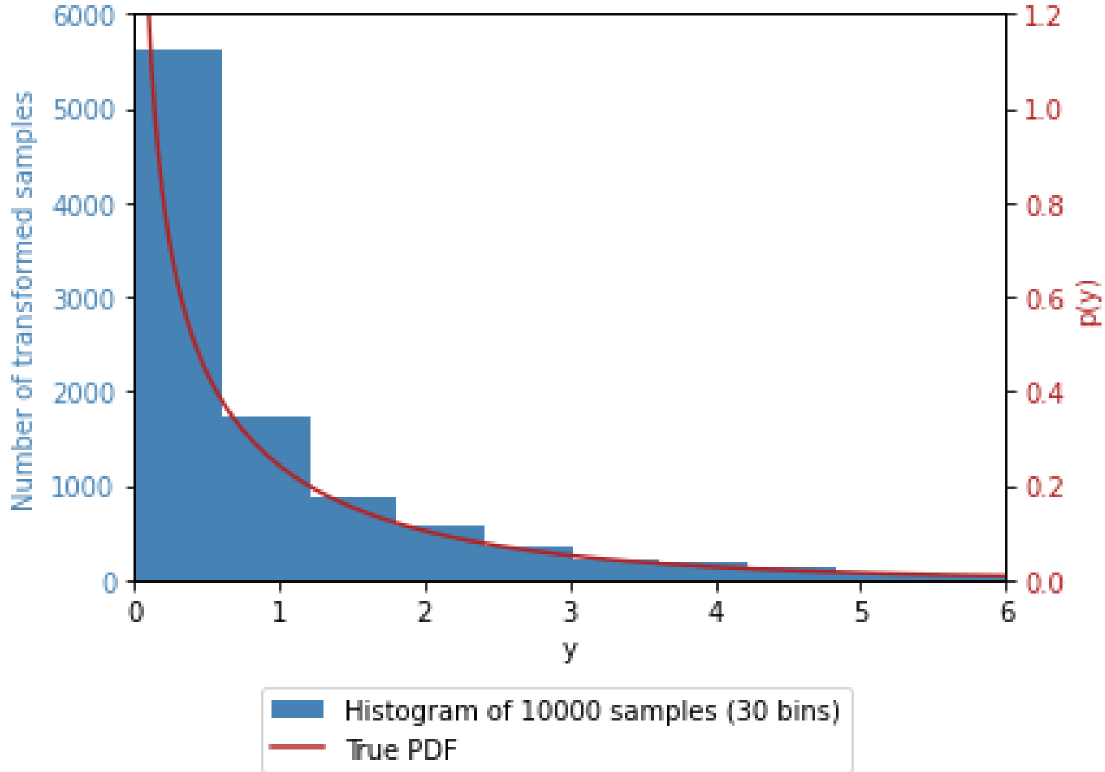
[176]:
```
fig, ax1 = plt.subplots()

Y = 2 * X_gaussian[:10000] + 3
ax1.hist(Y, bins=30, color=color1, label='Histogram of 10000 samples (30 bins)')
ax1.set_xlabel('y')
ax1.set_ylabel('Number of transformed samples', color=color1)
ax1.tick_params(axis='y', labelcolor=color1)
ax1.set_ylim(0, 1200)


ax2 = ax1.twinx()
x = np.linspace(-12, 12, 1000)
ax2.plot(x, gaussian_pdf(x, mu=3, sigma=2), color=color2, label='True PDF')
ax2.set_ylabel('p(y)', color=color2)
ax2.tick_params(axis='y', labelcolor=color2)
ax2.set_ylim(0, 0.23)
ax2.set_xlim(-12, 12)

fig.legend(loc='upper center', bbox_to_anchor=(0.5, 0))

plt.savefig('figures/linear_function_of_gaussian.png', bbox_inches='tight')
```



0.0.3 f(x) = x^2

[177]:
```
fig, ax1 = plt.subplots()

Y = X_gaussian[:10000] ** 2
ax1.hist(Y, bins=30, color=color1, label='Histogram of 10000 samples (30 bins)')
ax1.set_xlabel('y')
ax1.set_ylabel('Number of transformed samples', color=color1)
ax1.tick_params(axis='y', labelcolor=color1)
ax1.set_ylim(0, 6000)


ax2 = ax1.twinx()
x = np.linspace(0.01, 12, 1000)
pdf = np.exp(-0.5*x) / np.sqrt(2*np.pi*x)
ax2.plot(x, pdf, color=color2, label='True PDF')
ax2.set_ylabel('p(y)', color=color2)
ax2.tick_params(axis='y', labelcolor=color2)
ax2.set_ylim(0, 1.2)
ax2.set_xlim(0, 6)

fig.legend(loc='upper center', bbox_to_anchor=(0.5, 0))
```

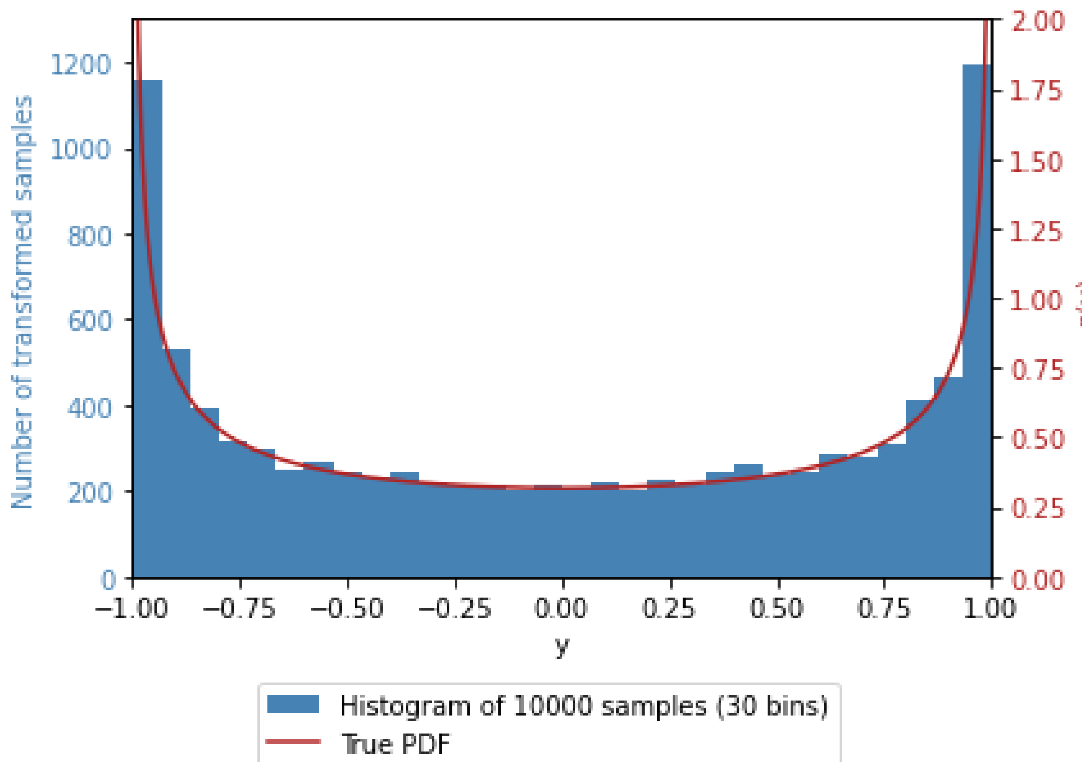
```
plt.savefig('figures/quadratic_function_of_gaussian.png', bbox_inches='tight')
```



0.0.4 f(x) = sin(x)

[178]:
```
fig, ax1 = plt.subplots()

Y = np.sin(X_uniform[:10000]*2*np.pi)
ax1.hist(Y, bins=30, color=color1, label='Histogram of 10000 samples (30 bins)')
ax1.set_xlabel('y')
ax1.set_ylabel('Number of transformed samples', color=color1)
ax1.tick_params(axis='y', labelcolor=color1)
ax1.set_ylim(0, 1300)


ax2 = ax1.twinx()
y = np.linspace(-0.99, 0.99, 1000)
pdf = 1 / (np.pi * np.sqrt(1 - y**2))
ax2.plot(y, pdf, color=color2, label='True PDF')
ax2.set_ylabel('p(y)', color=color2)
ax2.tick_params(axis='y', labelcolor=color2)
```

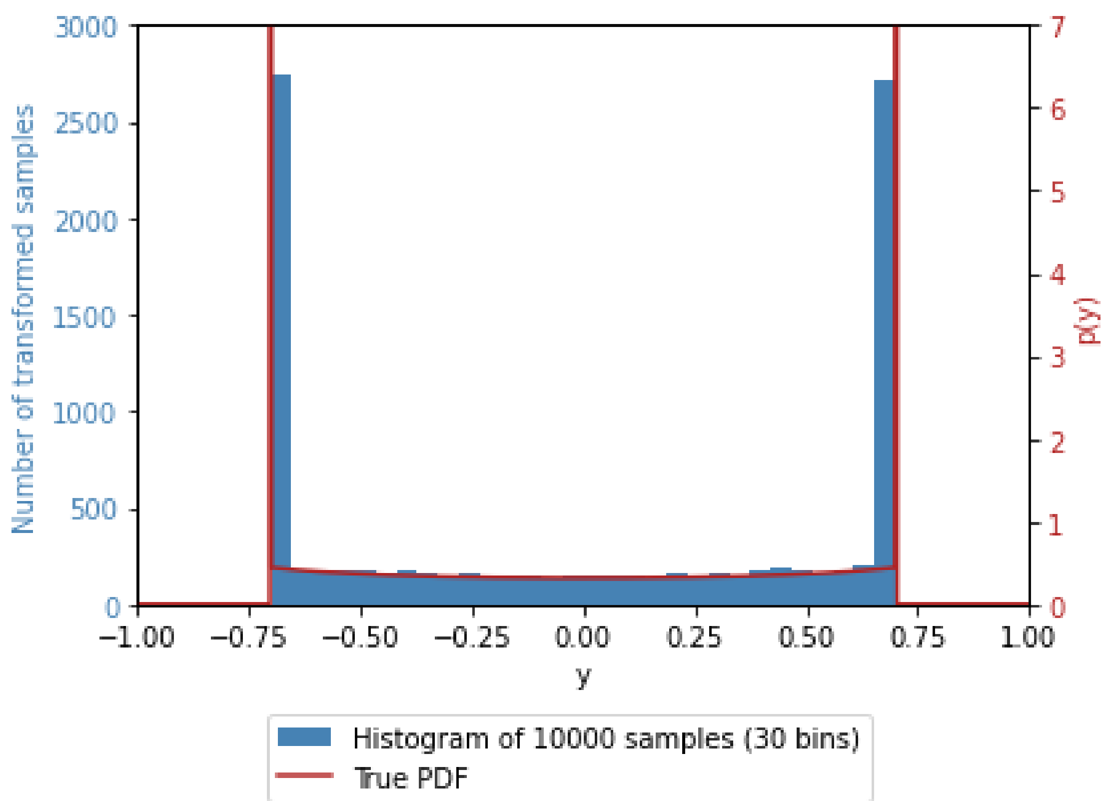
```
ax2.set_ylim(0, 2)
ax2.set_xlim(-1, 1)

fig.legend(loc='upper center', bbox_to_anchor=(0.5, 0))

plt.savefig('figures/sinusoidal_function_of_uniform.png', bbox_inches='tight')
```



0.0.5 f(x) = limited(sin(x))

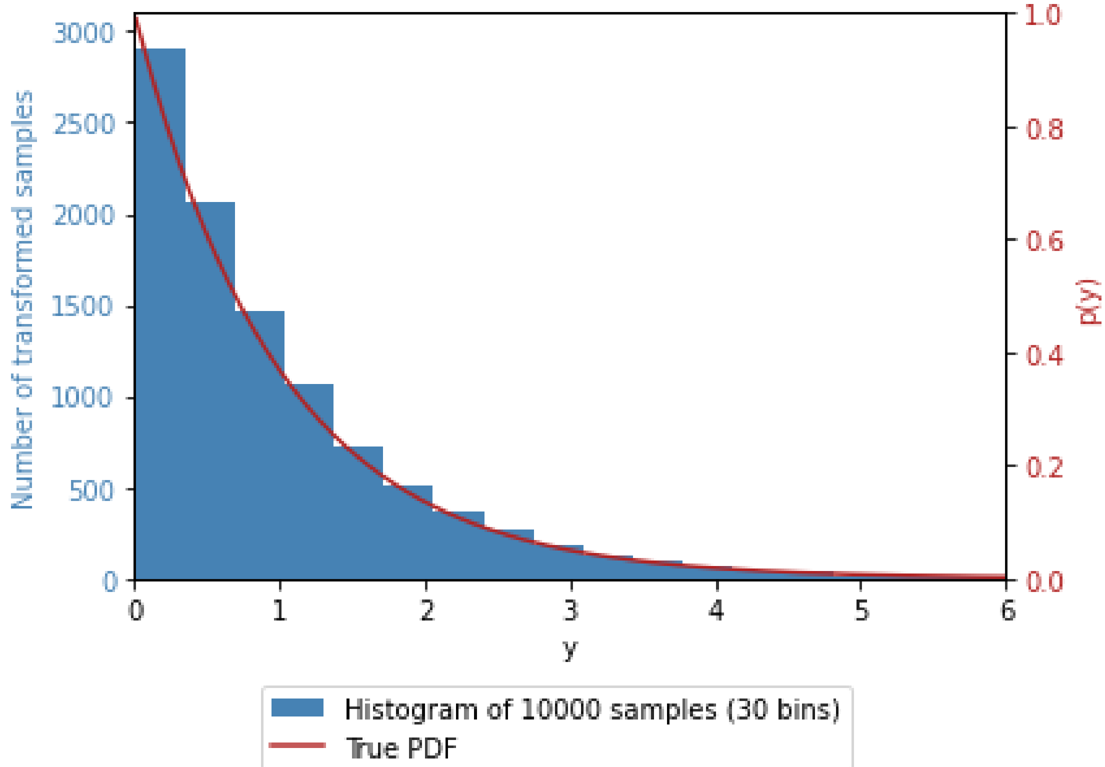
[179]:
```
fig, ax1 = plt.subplots()

Y = np.clip(np.sin(X_uniform[:10000]*2*np.pi), -0.7, 0.7)
ax1.hist(Y, bins=30, color=color1, label='Histogram of 10000 samples (30 bins)')
ax1.set_xlabel('y')
ax1.set_ylabel('Number of transformed samples', color=color1)
ax1.tick_params(axis='y', labelcolor=color1)
ax1.set_ylim(0, 3000)


ax2 = ax1.twinx()
y = np.linspace(-1, 1, 1000)
```

```
def pdf(y):
    return 1 / (np.pi * np.sqrt(1 - y**2))

ax2.plot(y,
         np.piecewise(y,
                      [np.abs(y) > 0.7,
                       np.abs(np.abs(y) - 0.7) < 0.002,
                       np.abs(y) < 0.7],
                      [0,
                       9e99,
                       pdf]),
         color=color2, label='True PDF')
ax2.set_ylabel('p(y)', color=color2)
ax2.tick_params(axis='y', labelcolor=color2)
ax2.set_ylim(0, 7)
ax2.set_xlim(-1, 1)

fig.legend(loc='upper center', bbox_to_anchor=(0.5, 0))

plt.savefig('figures/limited_sinusoidal_function_of_uniform.png',␣
 ↪bbox_inches='tight')
```



0.0.6 Section 3: iCDF method

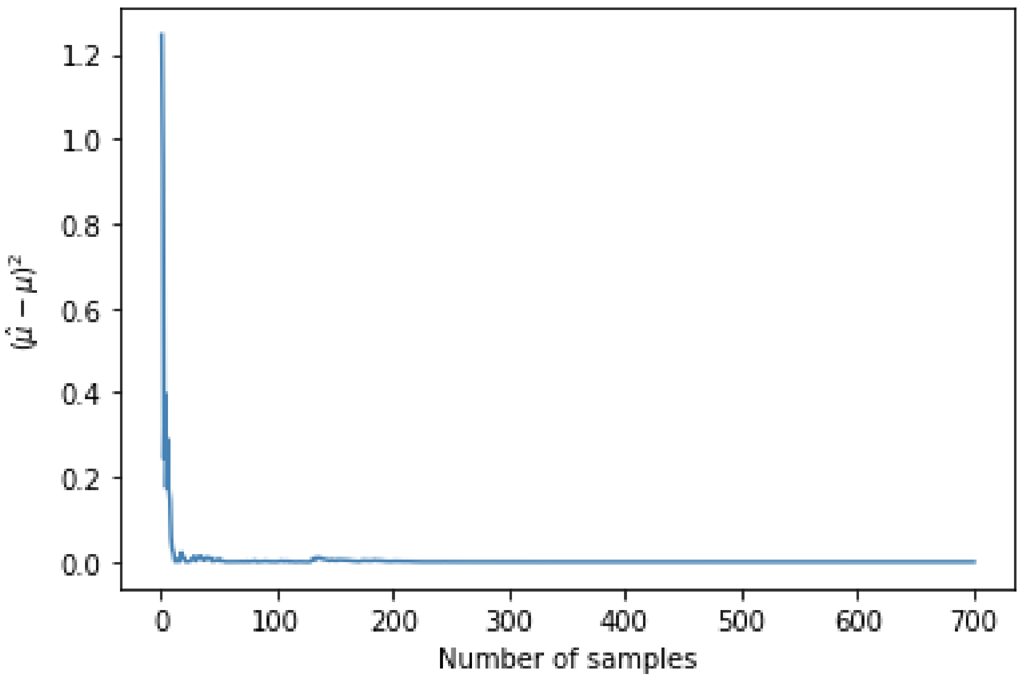
[180]:
```
fig, ax1 = plt.subplots()

Y = -np.log(1 - X_uniform)

ax1.hist(Y, bins=30, color=color1, label=f'Histogram of {len(Y)} samples (30␣
↪bins)')
ax1.set_xlabel('y')
ax1.set_ylabel('Number of transformed samples', color=color1)
ax1.tick_params(axis='y', labelcolor=color1)
ax1.set_ylim(0, 3100)


ax2 = ax1.twinx()
x = np.linspace(0.01, 12, 1000)
pdf = np.exp(-x)
ax2.plot(x, pdf, color=color2, label='True PDF')
ax2.set_ylabel('p(y)', color=color2)
ax2.tick_params(axis='y', labelcolor=color2)
ax2.set_ylim(0, 1)
ax2.set_xlim(0, 6)

fig.legend(loc='upper center', bbox_to_anchor=(0.5, 0))

plt.savefig('figures/icdf_exponential.png', bbox_inches='tight')
```

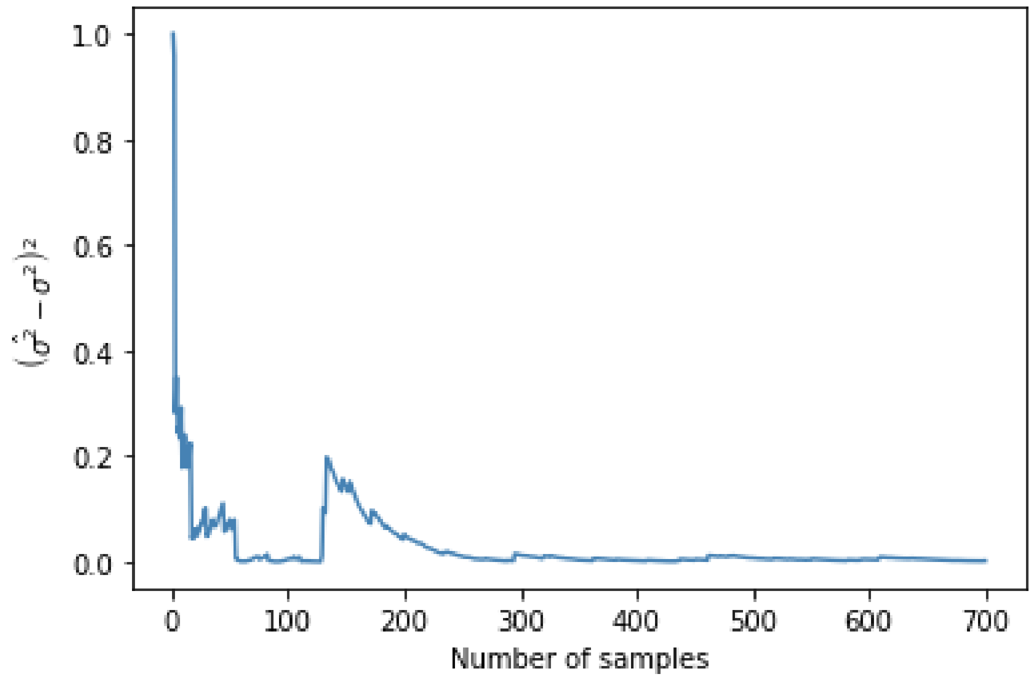


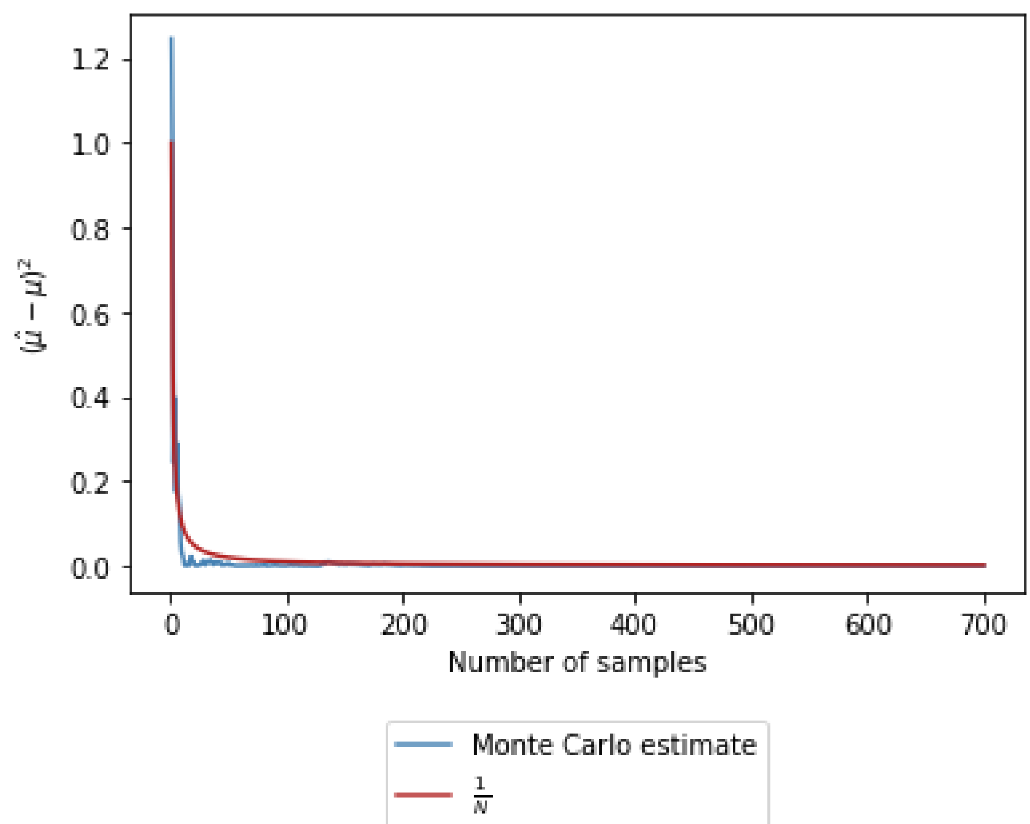
[181]:
```
def estimate_mean(N):
    return np.mean(Y[:N])

def estimate_variance(N):
    return np.mean(Y[:N] * Y[:N]) - estimate_mean(N)**2

print(f"Mean: {estimate_mean(10000)}")
print(f"Variance: {estimate_variance(10000)}")

x = np.linspace(1, 700, 700)

plt.figure()
plt.plot(x, [(estimate_mean(int(n)) - 1)**2 for n in x], color=color1)
# plt.plot(x, 1/x)
plt.xlabel("Number of samples")
plt.ylabel(r'$(\hat{\mu} - \mu)^2$')
plt.savefig("figures/monte_carlo_mean.png")

plt.figure()
plt.plot(x, [(estimate_variance(int(n)) - 1)**2 for n in x], color=color1)
plt.xlabel("Number of samples")
plt.ylabel(r'$\left(\hat{\sigma^2} - \sigma^2\right)^2$')
```

```
plt.savefig("figures/monte_carlo_variance.png")

plt.figure()
plt.plot(x, [(estimate_mean(int(n)) - 1)**2 for n in x], color=color1,␣
 ↪label='Monte Carlo estimate')
plt.plot(x, 1/x, color=color2, label=r'$\frac{1}{N}$')
plt.xlabel("Number of samples")
plt.ylabel(r'$(\hat{\mu} - \mu)^2$')
plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.2))
plt.savefig("figures/monte_carlo_mean_best_fit.png")
```

```
Mean: 0.9944135469954328
Variance: 0.9748765893086265
```




$(\hat{\sigma}^2 - \sigma^2)^2$
Number of samples

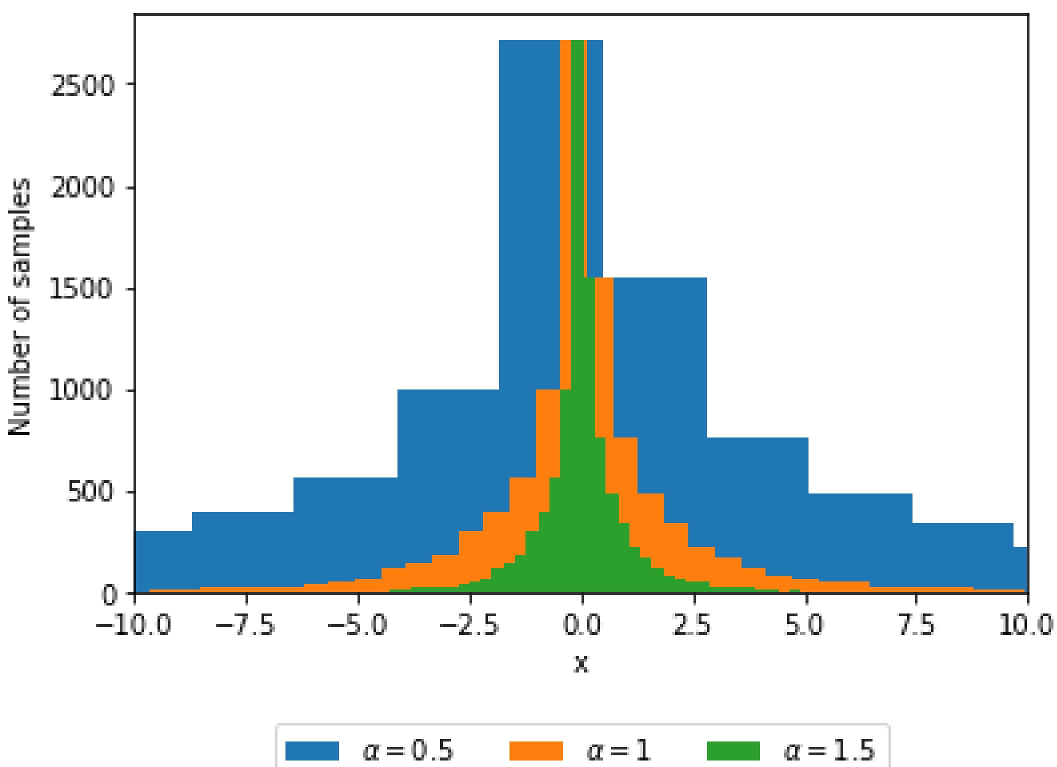

$(\hat{\mu} - \mu)^2$
Number of samples
Monte Carlo estimate
$\frac{1}{N}$

0.0.7 Section 4: Scaled mixture of Gaussians

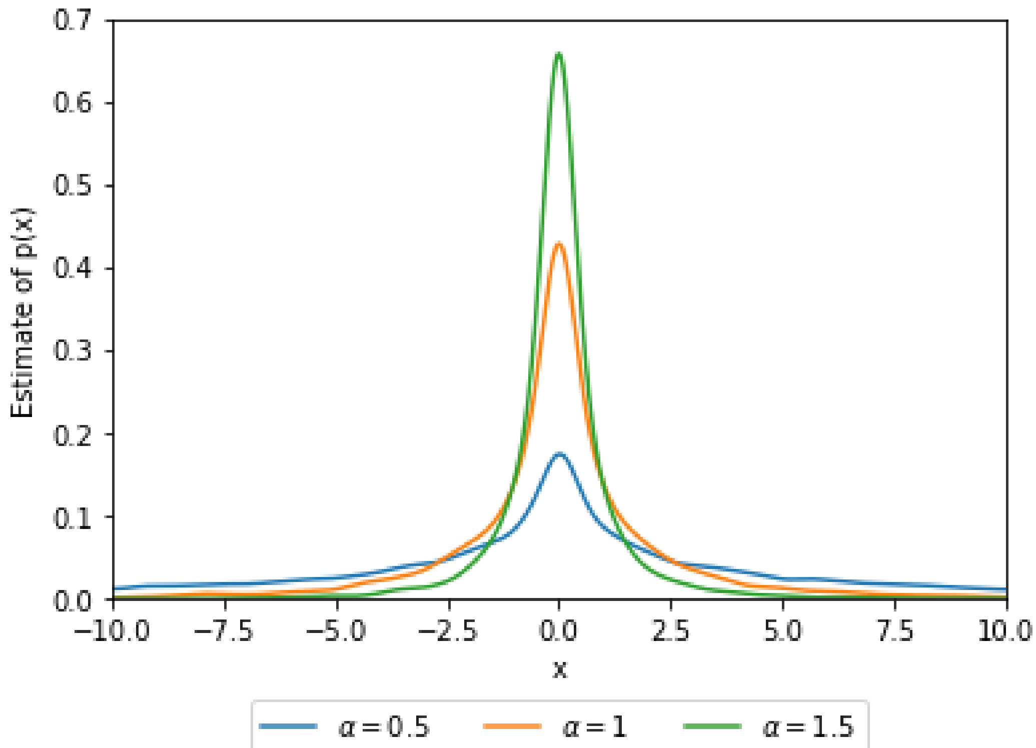
Exponential sampling

[182]:
```
def exponential_sampled_gaussian(N, alpha=1.5):
    exponential_samples = (-2 / (alpha ** 2)) * np.log(1 - X_uniform[:N])
    return X_gaussian[:N] * exponential_samples
```

[183]:
```
plt.figure()
N = 100000
for alpha in [0.5, 1, 1.5]:
    plt.hist(exponential_sampled_gaussian(N, alpha), bins=100, label=r'$\alpha␣
↪= {}$'.format(alpha))

plt.xlabel('x')
plt.ylabel('Number of samples')
plt.xlim(-10, 10)
plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.2), ncol=3)
plt.savefig(f'figures/exponential_sampled_gaussian.png', bbox_inches='tight')
```

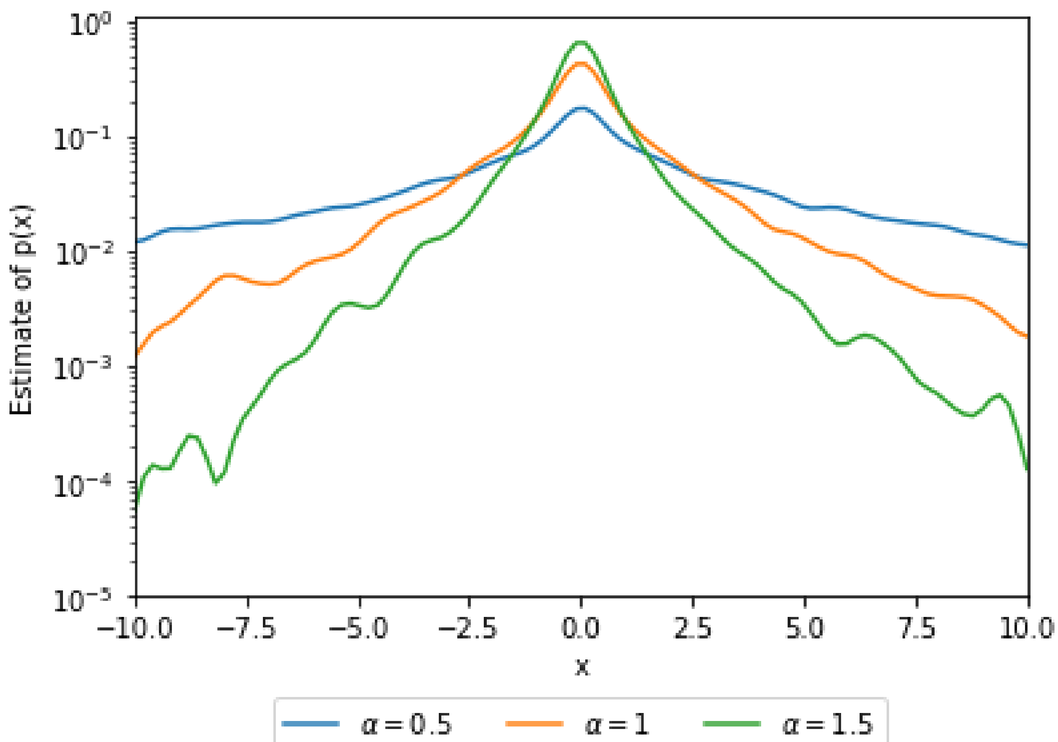


[184]:
```
plt.figure()

N = 100000
x = np.linspace(-10, 10, 1000)

for alpha in [0.5, 1, 1.5]:
    plt.plot(x, kernel_smoothed_density(x, exponential_sampled_gaussian(N,␣
 ↪alpha)), label=r'$\alpha = {}$'.format(alpha))

plt.xlabel('x')
plt.ylabel('Estimate of p(x)')
plt.xlim(-10, 10)
plt.ylim(0, 0.7)
plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.15), ncol=3)
plt.savefig(f'figures/exponential_sampled_gaussian_kernel_smoothed.png',␣
 ↪bbox_inches='tight')
```



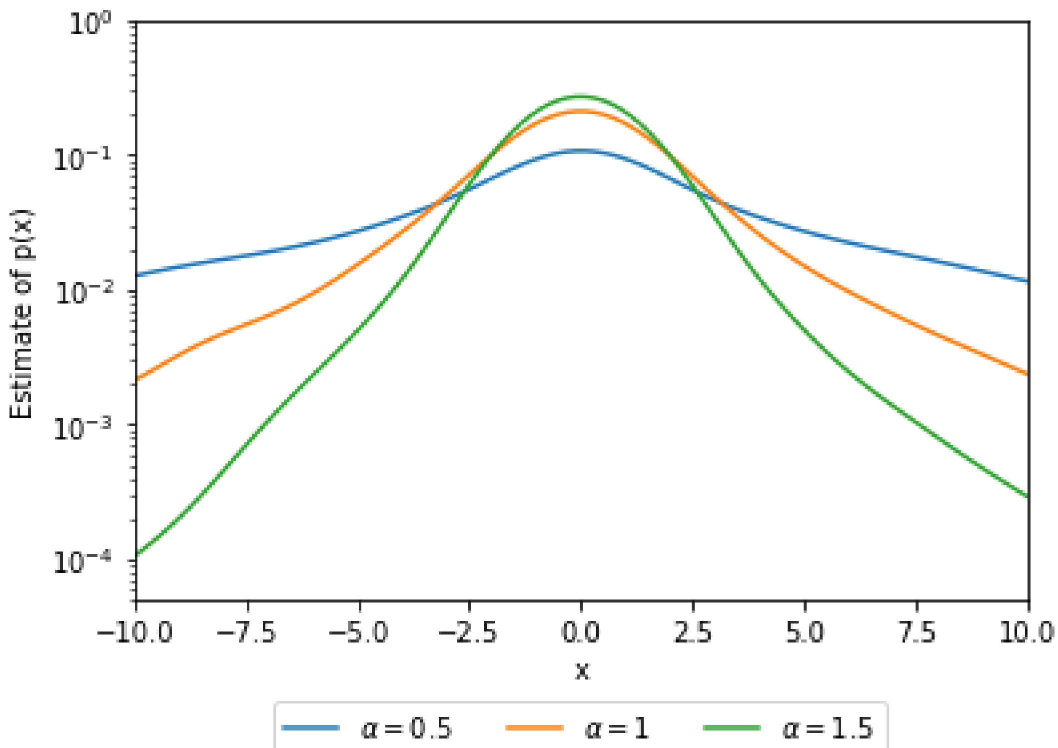
[185]:
```
plt.figure()

N = 100000
x = np.linspace(-10, 10, 100)
```

```
for alpha in [0.5, 1, 1.5]:
    plt.semilogy(x, kernel_smoothed_density(x, exponential_sampled_gaussian(N,␣
 ↪alpha)), label=r'$\alpha = {}$'.format(alpha))

plt.xlabel('x')
plt.ylabel('Estimate of p(x)')
plt.xlim(-10, 10)
plt.ylim(1e-5, 1.1)
plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.15), ncol=3)
plt.savefig(f'figures/exponential_sampled_gaussian_kernel_smoothed_log_narrow.
 ↪png', bbox_inches='tight')
```



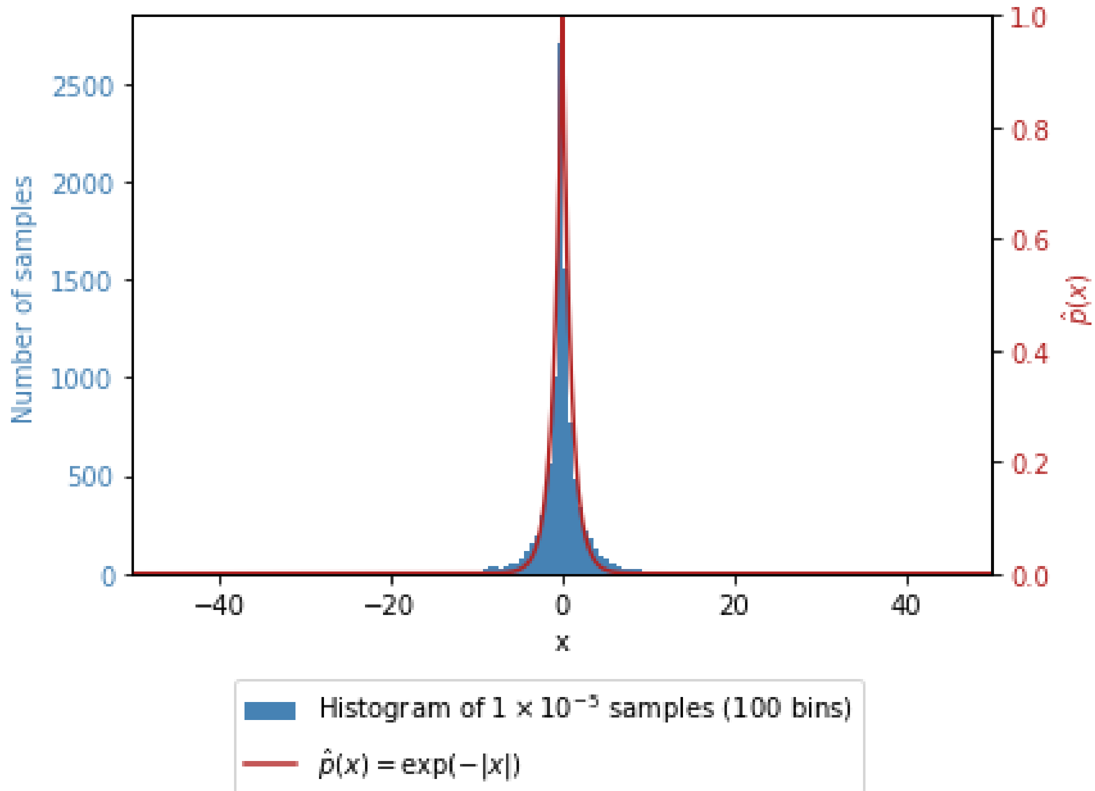
[186]:
```
plt.figure()

N = 100000
x = np.linspace(-10, 10, 100)

for alpha in [0.5, 1, 1.5]:
    plt.semilogy(x,
                 kernel_smoothed_density(x, exponential_sampled_gaussian(N,␣
 ↪alpha),
                                         width=1.2),
```

```
                 label=r'$\alpha = {}$'.format(alpha))

plt.xlabel('x')
plt.ylabel('Estimate of p(x)')
plt.xlim(-10, 10)
plt.ylim(.5e-4, 1)
plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.15), ncol=3)
plt.savefig(f'figures/exponential_sampled_gaussian_kernel_smoothed_log_wide.
↪png', bbox_inches='tight')
```



[187]:
```
fig, ax1 = plt.subplots()

N = 100000
ax1.hist(exponential_sampled_gaussian(N, 1),
         bins=100,
         label=r'Histogram of $1 \times 10^{-5}$ samples (100 bins)',
         color=color1)
ax1.set_xlabel('x')
ax1.set_ylabel('Number of samples', color=color1)
ax1.tick_params(axis='y', labelcolor=color1)

ax2 = ax1.twinx()
```

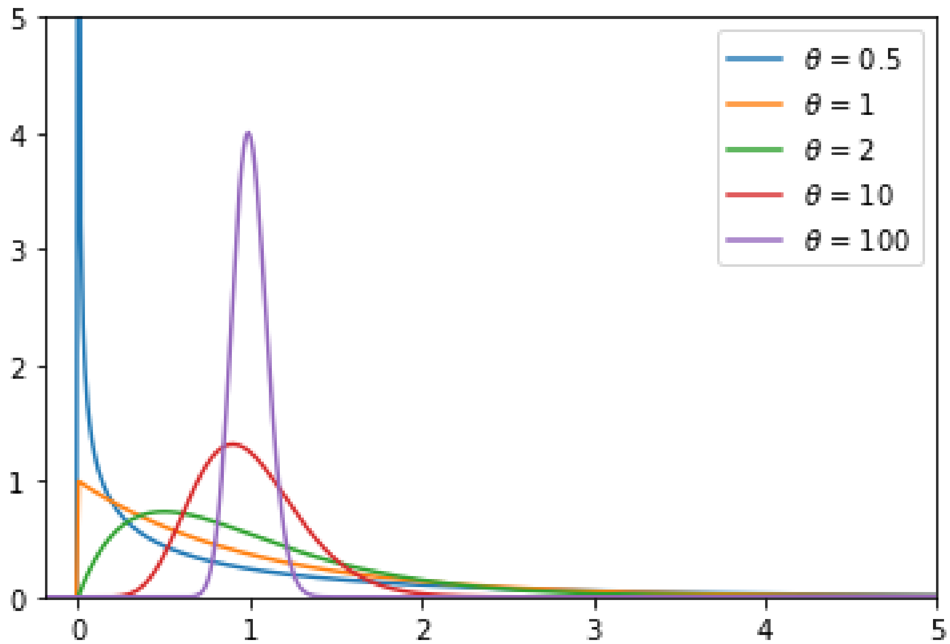
```
x = np.linspace(-50, 50, 10000)
ax2.plot(x, np.exp(-np.abs(x)), label=r'$\hat{p}(x) = \exp(-|x|)$',␣
 ↪color=color2)
ax2.set_ylabel(r'$\hat{p}(x)$', color=color2)
ax2.tick_params(axis='y', labelcolor=color2)
ax2.set_ylim(0, 1)
ax2.set_xlim(-50, 50)

fig.legend(loc='upper center', bbox_to_anchor=(0.5, 0))
plt.savefig(f'figures/exponential_sampled_gaussian_with_pdf.png',␣
 ↪bbox_inches='tight')
```



Gamma sampling

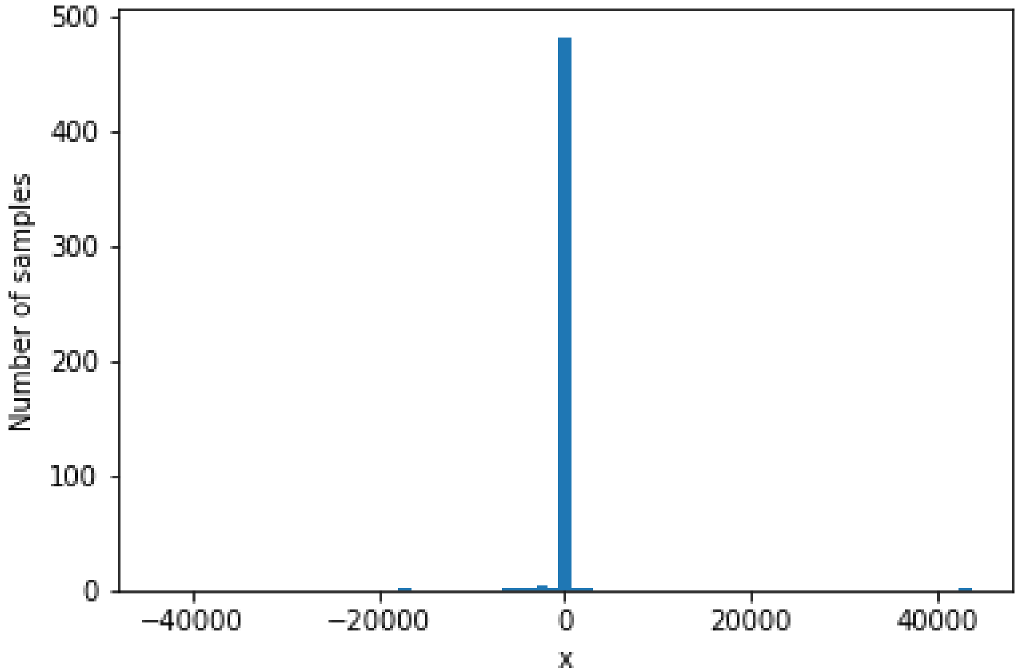
[188]:
```
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt

x = np.linspace(-1, 12, 1000)

plt.figure()
```

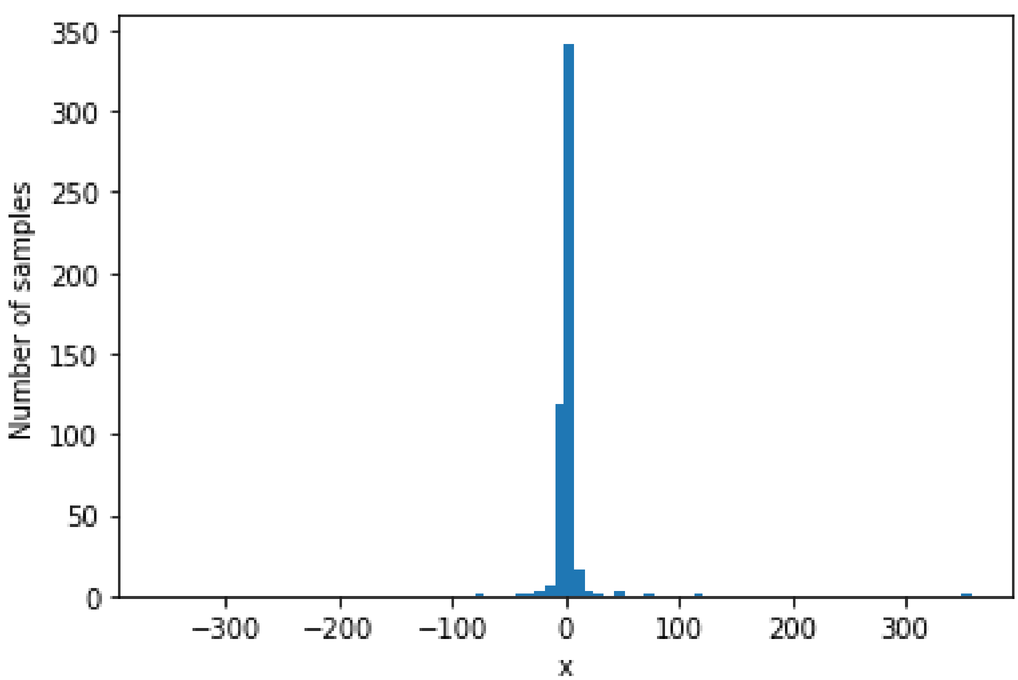
```
for t in [0.5,1,2,10,100]:
    plt.plot(x, stats.gamma.pdf(x, t, scale=1/t), label=r'$\theta$ = {}'.
 ↪format(t))
plt.xlim(-0.2,5)
plt.ylim(0,5)
plt.legend()
plt.savefig(f'figures/gamma_theta.png', bbox_inches='tight')
```

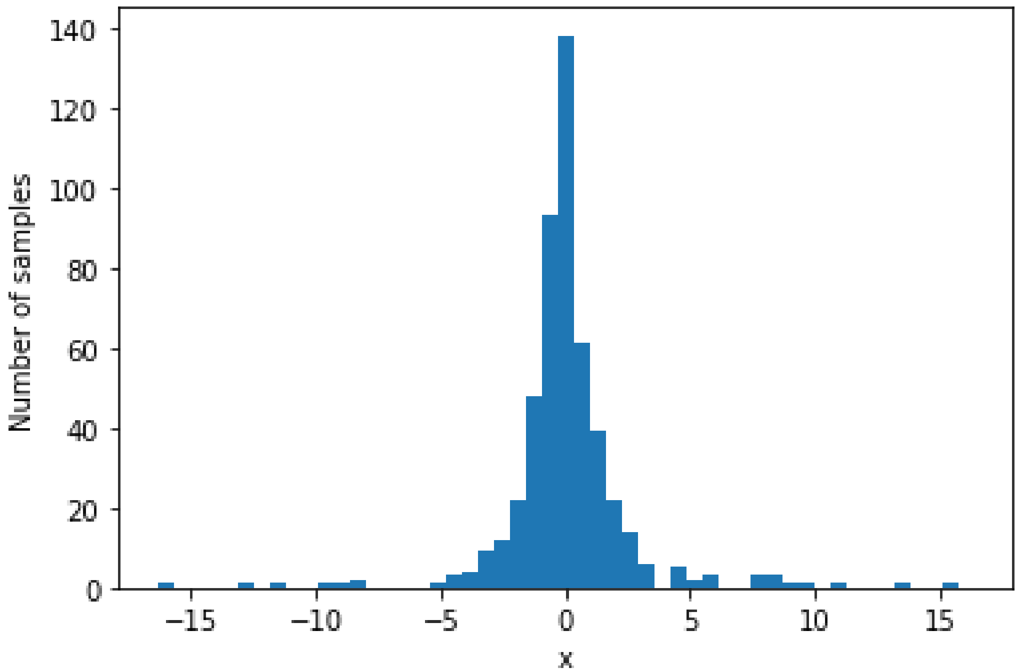


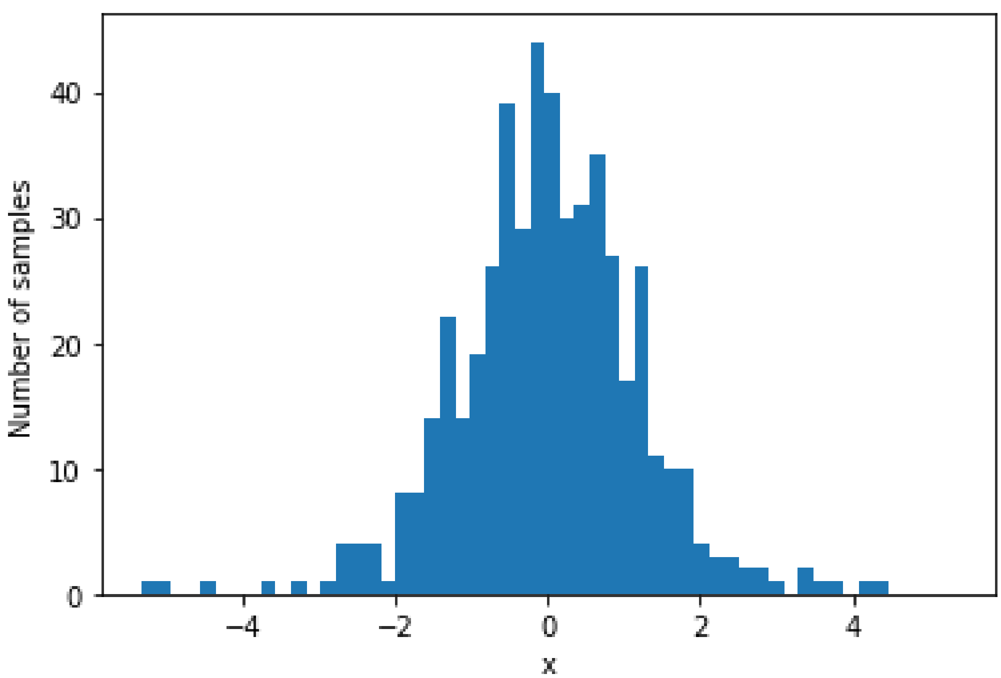
[189]:
```
def gamma_sampled_gaussian(N, theta):
    v = stats.gamma.rvs(a=theta, scale=1/theta, size=N)
    u = 1 / v
    return np.random.normal(loc=0, scale=u)

N=500
nbins=50

samples = {theta: gamma_sampled_gaussian(N, theta) for theta in [0.5, 1, 2, 10,␣
 ↪100]}

for theta, s in samples.items():
    plt.figure()
    values, bins, patches = plt.hist(s, bins=nbins)
    xlim = 1.1*max(abs(min(bins)), abs(max(bins)))
    plt.xlim(-xlim, xlim)
    plt.xlabel('x')
```

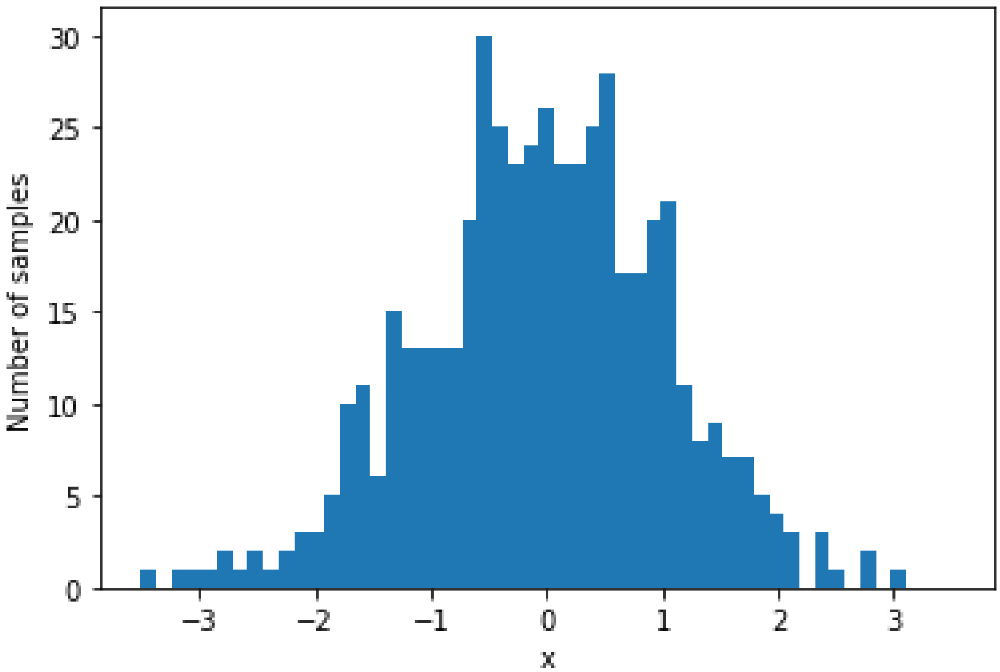
```
    plt.ylabel('Number of samples')
    plt.savefig(f"figures/gamma_sampled_gaussian_histogram_{theta}.png")
```



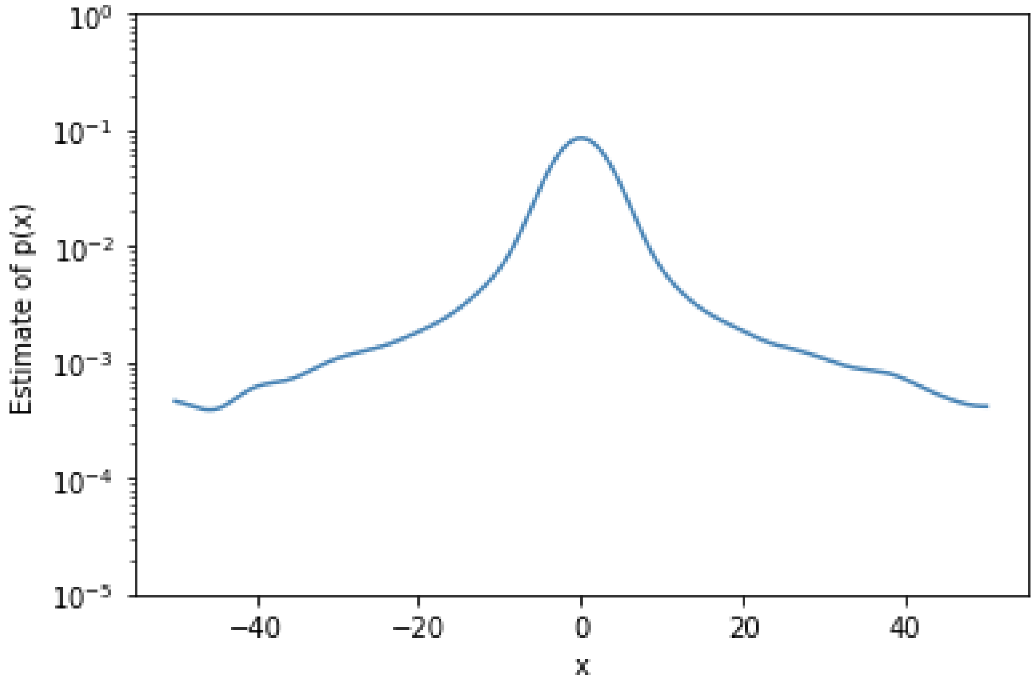



Number of samples
140
120
100
80
60
40
20
0
−15
−10
−5
0
5
10
15
x

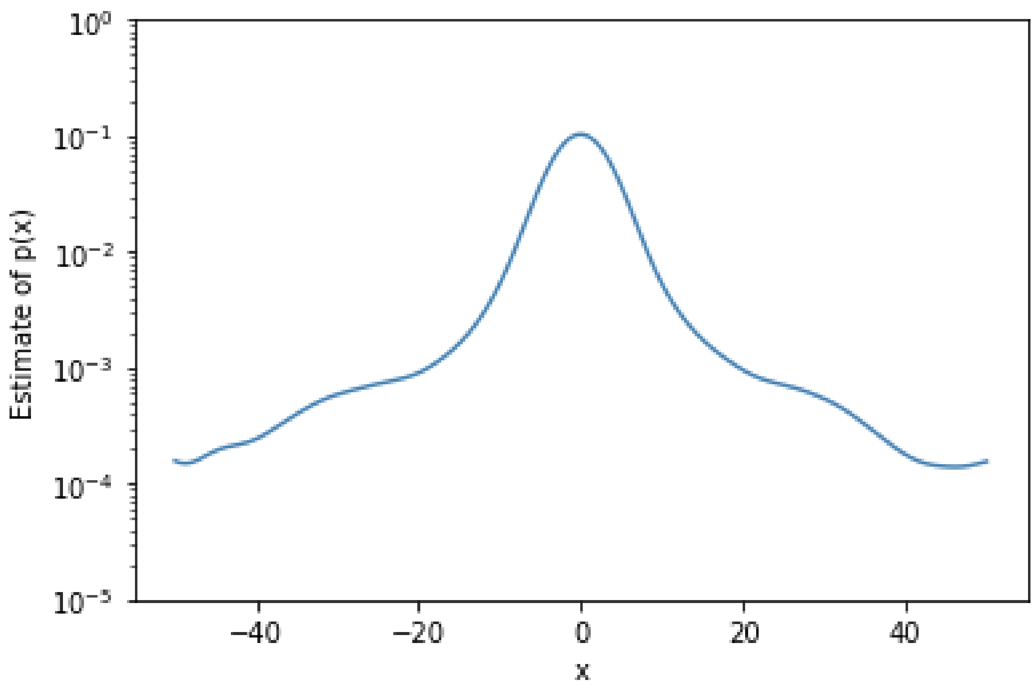

Number of samples
40
30
20
10
0
−4
−2
0
2
4
x



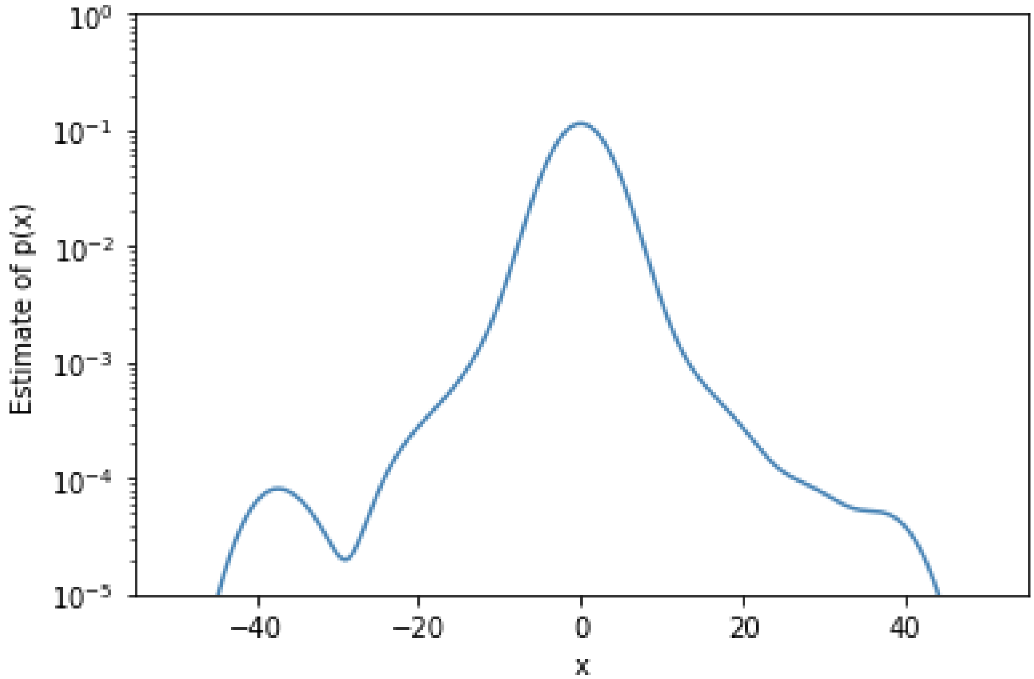
[190]:
```
N = 10000
x = np.linspace(-50, 50, 1000)

for theta in [0.5, 1, 2, 10, 100]:
    fig, ax1 = plt.subplots()

    ax1.semilogy(x,
                 kernel_smoothed_density(x,
                                         gamma_sampled_gaussian(N, theta),
                                         width=3),
                 label=r'$\theta = {}$'.format(theta),
                 color=color1)

    ax1.set_ylim(1e-5, 1)
    ax1.set_xlabel('x')
    ax1.set_ylabel('Estimate of p(x)')

    plt.savefig(f'figures/gamma_sampled_gaussian_smoothed_{theta}.png',␣
 ↪bbox_inches='tight')
```



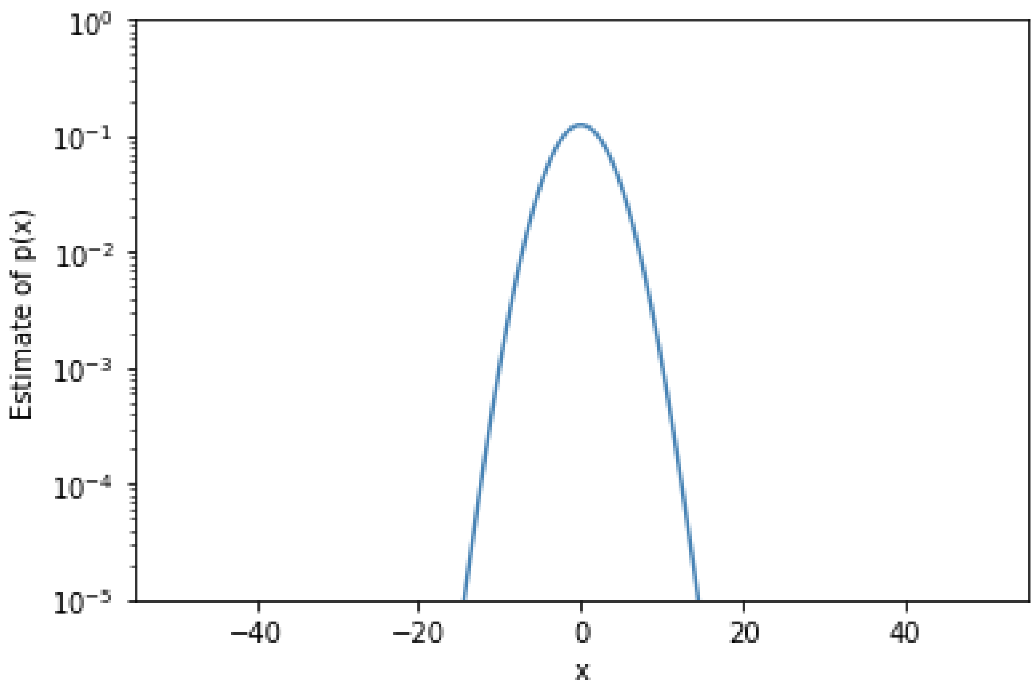
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
Estimate of p(x)
−40
−20
0
20
40
x



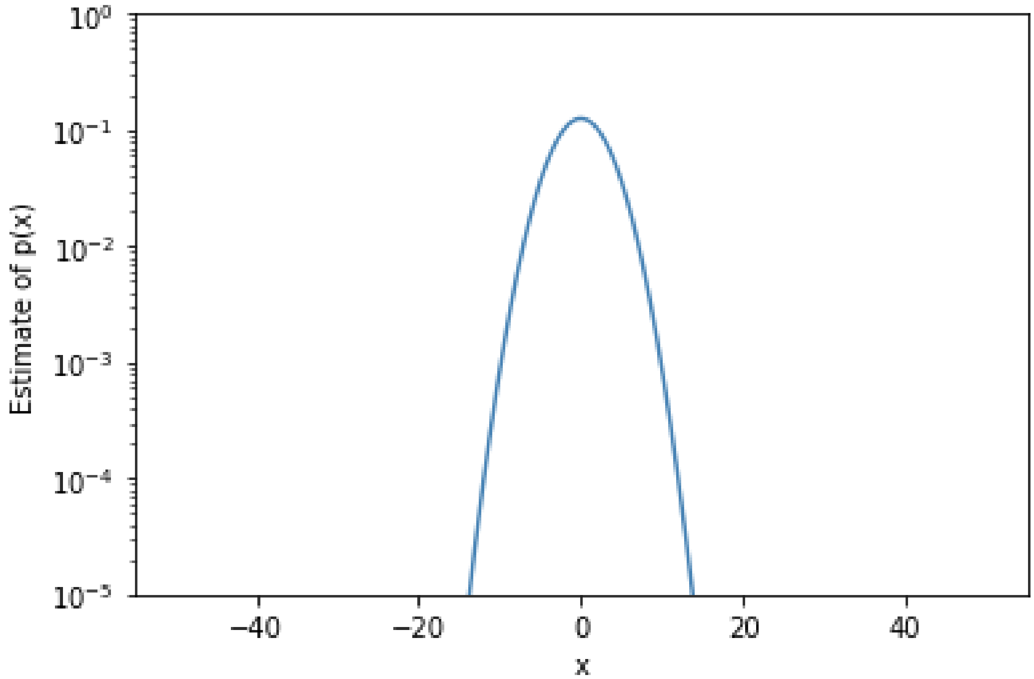
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
Estimate of p(x)
−40
−20
0
20
40
x



Estimate of p(x)
x



Estimate of p(x)
x



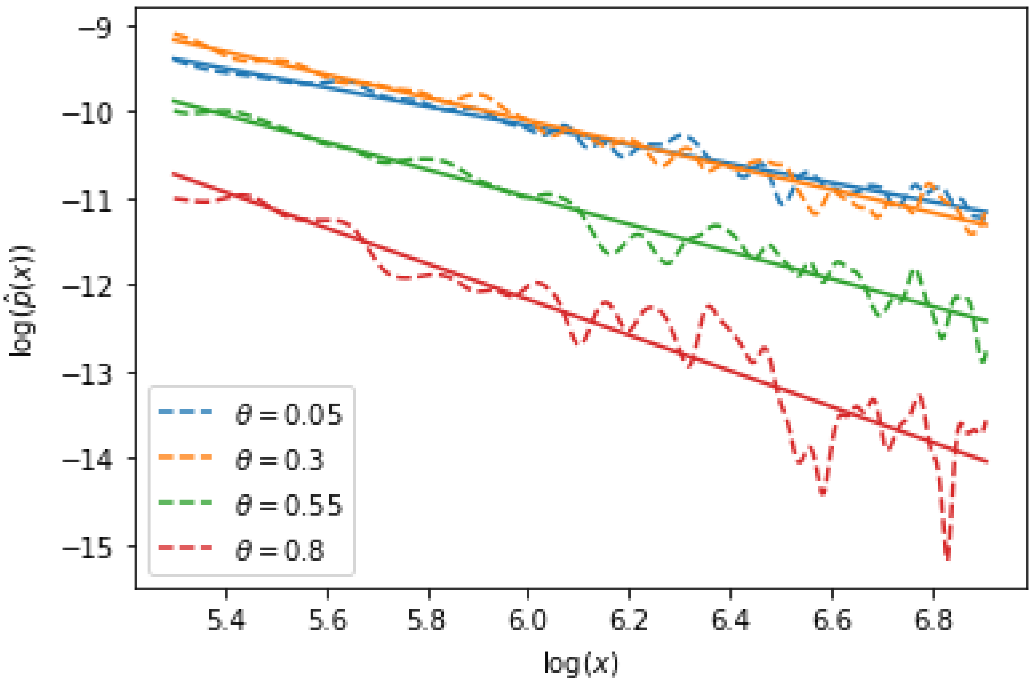
[191]:
```
N = int(1e5)
samples = {theta: gamma_sampled_gaussian(N, theta) for theta in np.linspace(0.
↪05, 0.8, 10)}
```

[192]:
```
width = 10
x = np.linspace(0, int(1e3), int(1e3))
smoothed_samples = {theta: kernel_smoothed_density(x, samples_theta, width)
                    for theta, samples_theta in samples.items()}
```

[193]:
```
from scipy.optimize import curve_fit

fit = {}
for theta, smoothed in smoothed_samples.items():
    xdata = np.log(x[200:])
    ydata = np.log(smoothed[200:])
    np.nan_to_num(ydata, neginf=-9e99)
    fit[theta] = curve_fit(lambda x, a, b: a*x+b, xdata=xdata, ydata=ydata)[0]
```

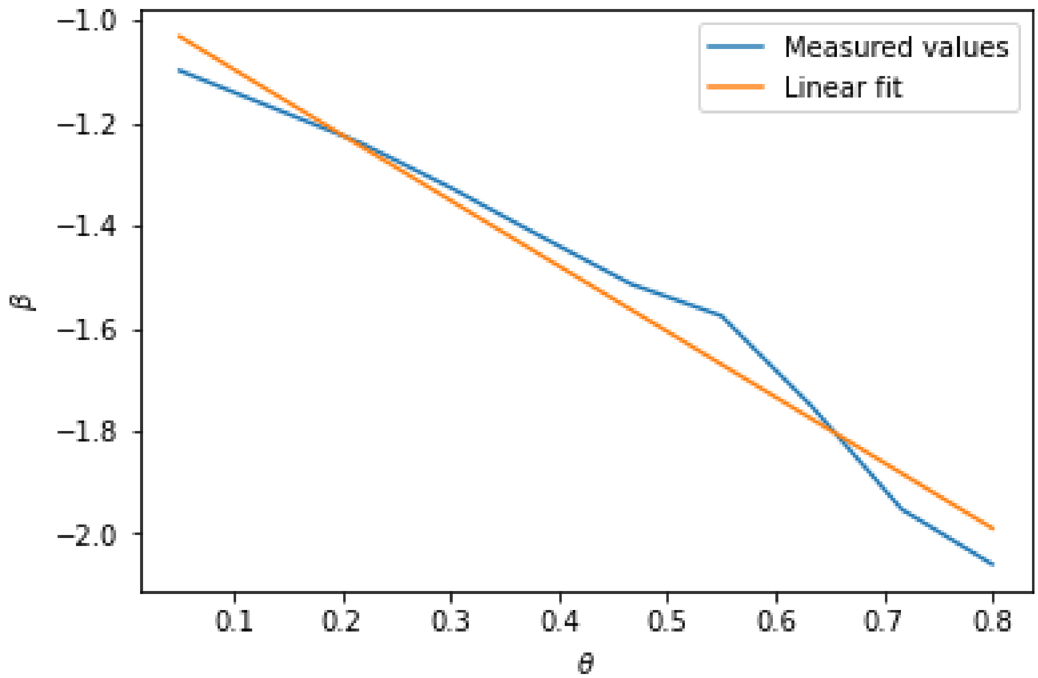
[194]:
```
plt.figure()
for i, theta in enumerate([0.05, 0.3, 0.55, 0.8]):
    m, c = fit[theta]
    logx = np.log(x[200:])
    plt.plot(logx, np.log(smoothed_samples[theta][200:]),
```

```
            label=r'$\theta={}$'.format(theta), linestyle='dashed',␣
↪color=f'C{i}')
    plt.plot(logx, m*logx+c)

plt.xlabel(r'$\log(x)$')
plt.ylabel(r'$\log(\hat{p}(x))$')
plt.legend()
plt.savefig('figures/gamma_sampled_gaussian_tail_fit.png')
```



[197]:
```
plt.figure()
thetas = [theta for theta in smoothed_samples.keys()]
gradients = [fit[theta][0] for theta in thetas]
plt.plot(thetas, gradients, label='Measured values')
m, c = curve_fit(lambda x, a, b: a*x+b, xdata=thetas, ydata=gradients)[0]
plt.plot(np.linspace(0.05, 0.8), m*np.linspace(0.05, 0.8) + c, label='Linear␣
↪fit')
# plt.scatter(0.8, fit[0.8][0], label=r'$\theta=0.8$ (discarded)')
plt.xlabel(r'$\theta$')
plt.ylabel(r'$\beta$')
plt.legend()
plt.savefig('figures/beta_vs_theta.png')
print(f"B = {m}T + {c}")
```

```
B = -1.2760084202219186T + -0.9683391662146092
```



Measured values
Linear fit
−1.0
−1.2
−1.4
−1.6
−1.8
−2.0
β
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
θ