AIMS | Physics for Machine Learning | CW2

Hugo Touchette

Started: 12 April 2022

Last updated: 2 Nov 2023

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
from scipy import optimize
from scipy import stats
import warnings
```

In [ ]:
```
# Getting some unwanted warnings when using subplots
warnings.filterwarnings("ignore")
```

Q1

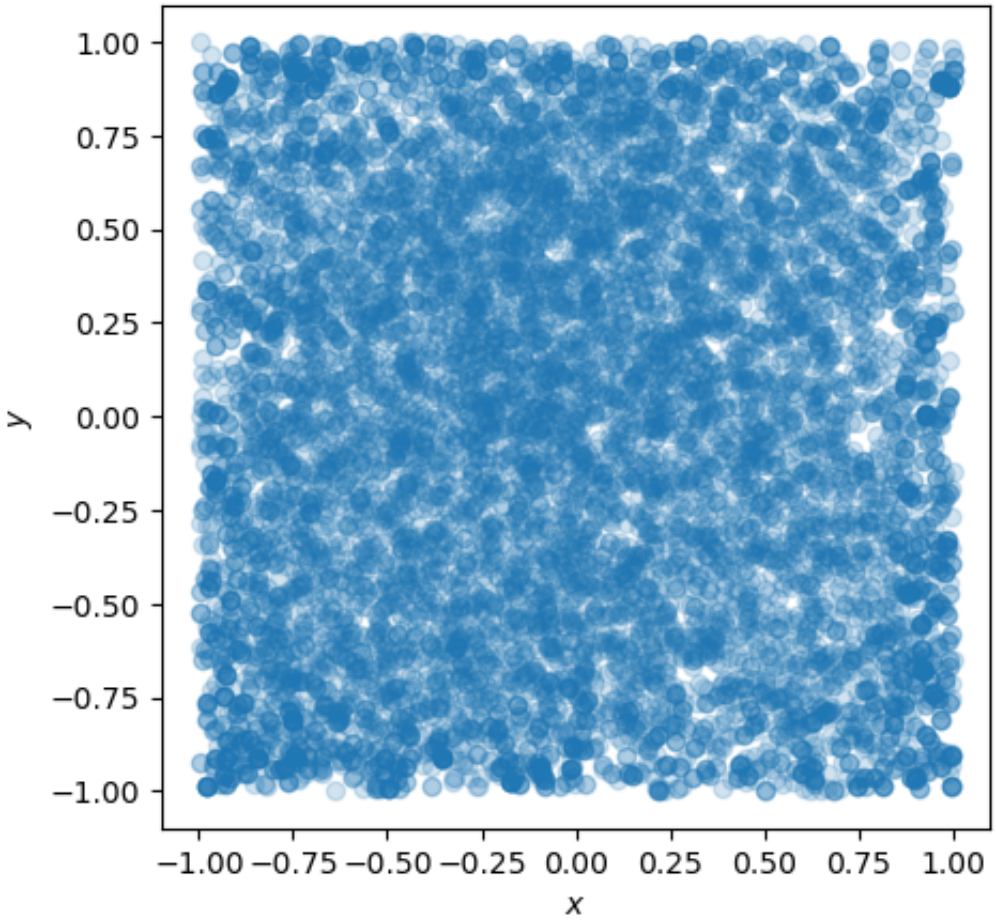
(a)

We use a small Gaussian displacement with zero mean throughout, meaning $\delta P_x \sim \delta P_y \sim \mathcal{N}(0, \sigma^2)$, with small σ.

In [ ]:
```
# Parameters
nb_steps = 10**4 # Number of steps
sigma = 0.2      # Variance for displacements
plist = []       # List of points to keep

# Initial point
p = np.array([0., 0.])

# Counter for acceptance rate
cnt = 0

# Simulation
for i in range(nb_steps):
    dp = np.random.normal(0, sigma, 2)  # Displacement
    p_try = p+dp                        # Possible change
    if p_try[0] <= 1.0 and p_try[0] >= -1.0 and p_try[1] <= 1.0 and p_try
        p = p_try                      # Accept change if in square, otherw
        cnt += 1                       # Count change accepted
    plist.append(p)                    # Keep point in list even if not cha

# Convert list of arrays into a genuine numpy array
newplist = np.vstack(plist)

# Fraction of moves accepted
```

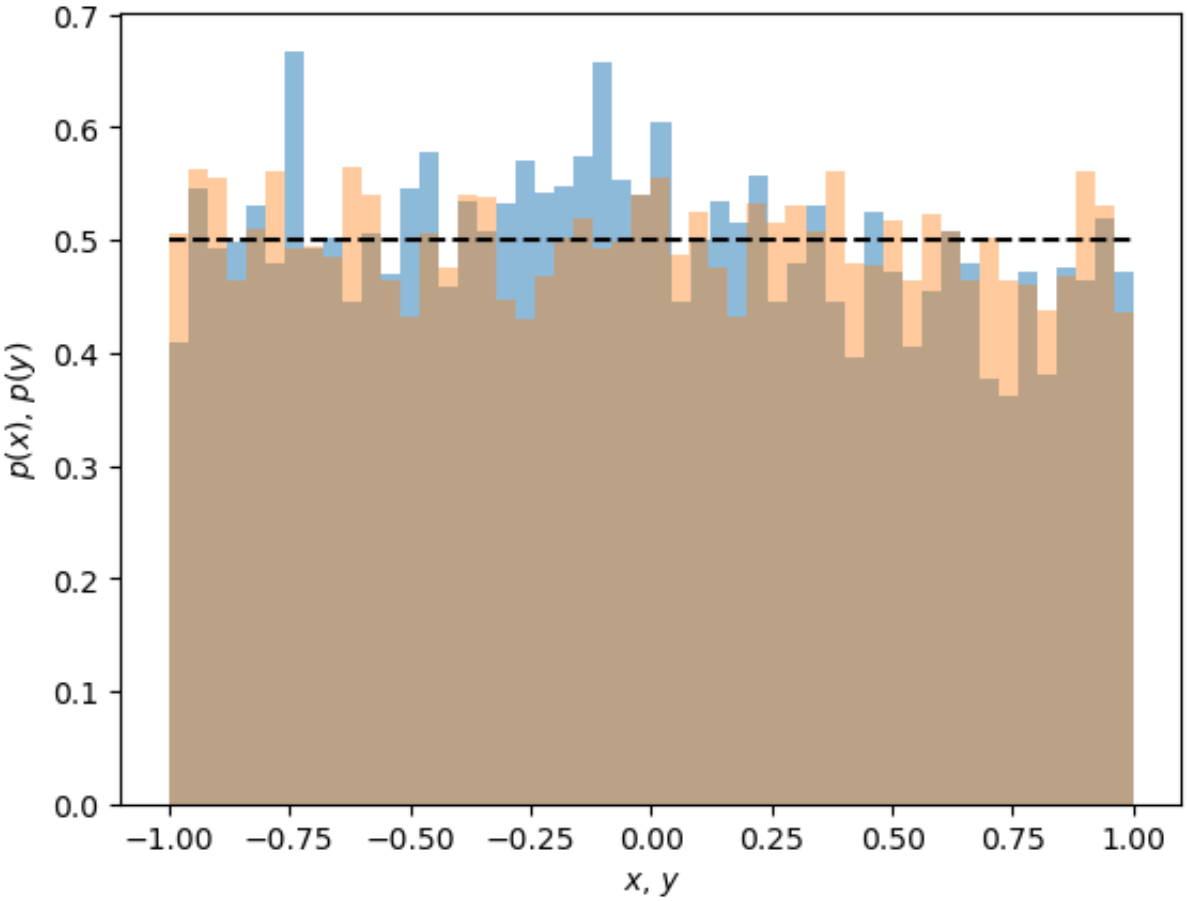
```
print(cnt/nb_steps, '% of moves accepted.')

# Plot points in (x,y) plane
plt.figure(figsize=(5, 5))
plt.plot(newplist[:, 0], newplist[:, 1], 'o', alpha=0.2)
plt.xlabel(r'$x$')
plt.ylabel(r'$y$')
plt.show()
```

```
0.8423 % of moves accepted.
```



This looks uniform. To confirm, we plot the histogram of the x and y positions:

In [ ]:
```
plt.hist(newplist[:, 0], bins=50, alpha=0.5, density=True)
plt.hist(newplist[:, 1], bins=50, alpha=0.4, density=True)
plt.plot(np.linspace(-1,1,50), 50*[0.5], 'k--')
plt.xlabel(r'$x$, $y$')
plt.ylabel(r'$p(x)$, $p(y)$')
plt.show()
```



You can play with the standard deviation of the displacements and see that the acceptance ratio is high for low σ (tiny moves will necessarily be accepted) while the acceptance ratio is low for high σ (big moves are likely to fall outside the square and are therefore not accepted).

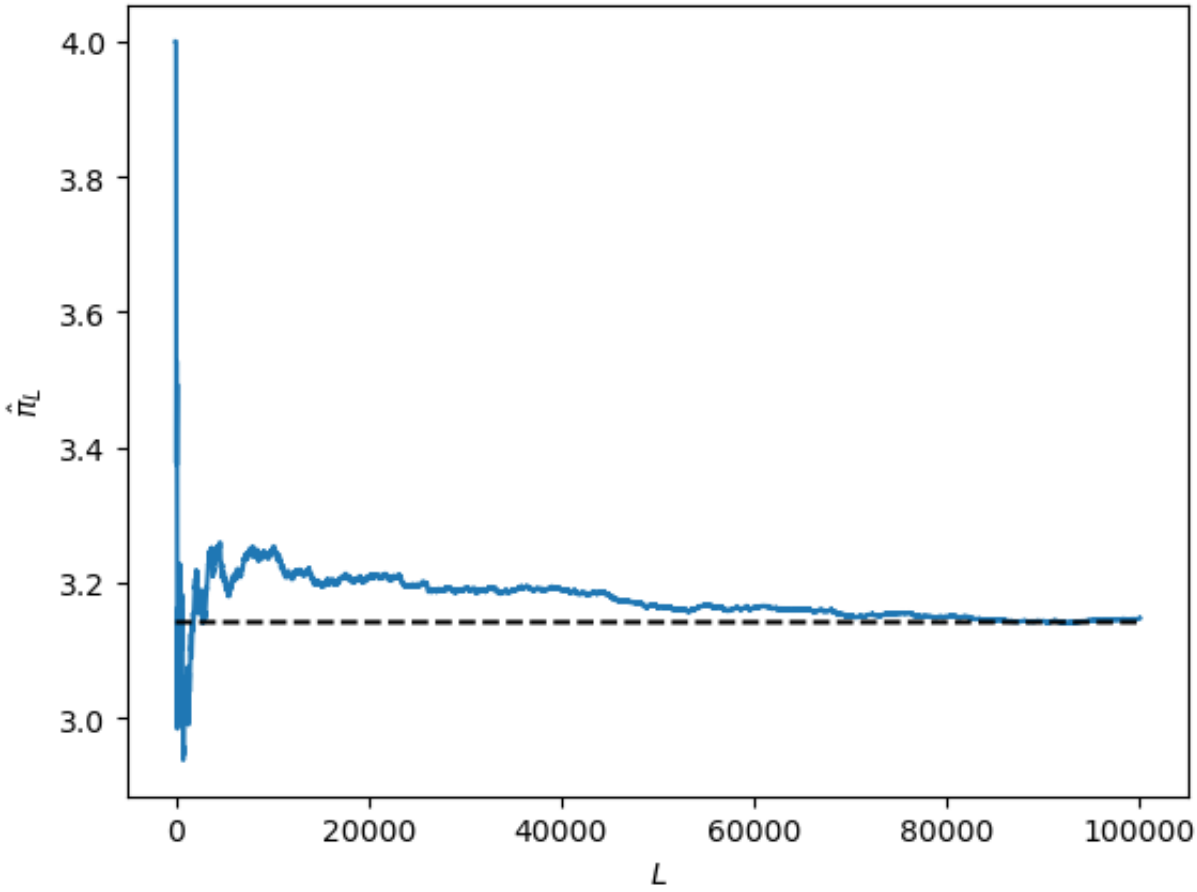
(b)

We use the code before and just add the estimator for π, known from CW1:

In [ ]:
```
# Parameters
nb_steps = 10**5
var = 0.2
pi_est = 0.0
estimate = []

# Initial point
p = np.array([0., 0.])

# Counter for acceptance rate
cnt = 0

# Simulation
for i in range(nb_steps):
    d = np.random.normal(0, var, 2)
    p_try = p+d

    # Target distribution is uniform on square, so Metropolis ratio is 1
    if np.abs(p_try[0]) <= 1.0 and np.abs(p_try[1]) <= 1.0:
```

```
        p = p_try
        cnt += 1

    # Estimator: Check if point falls in circle
    if p[0]**2+p[1]**2 <= 1:
        pi_est += 4.0

    estimate.append(pi_est/(i+1))

print('Final estimate of pi:', pi_est/nb_steps)
print('Acceptance ratio:', cnt/nb_steps)
plt.plot(range(nb_steps), estimate, range(nb_steps), np.pi*np.ones(nb_ste
plt.xlabel('$L$')
plt.ylabel('$\hat\pi_L$')
plt.show()
```

```
Final estimate of pi: 3.14752
Acceptance ratio: 0.8468
```



(c)

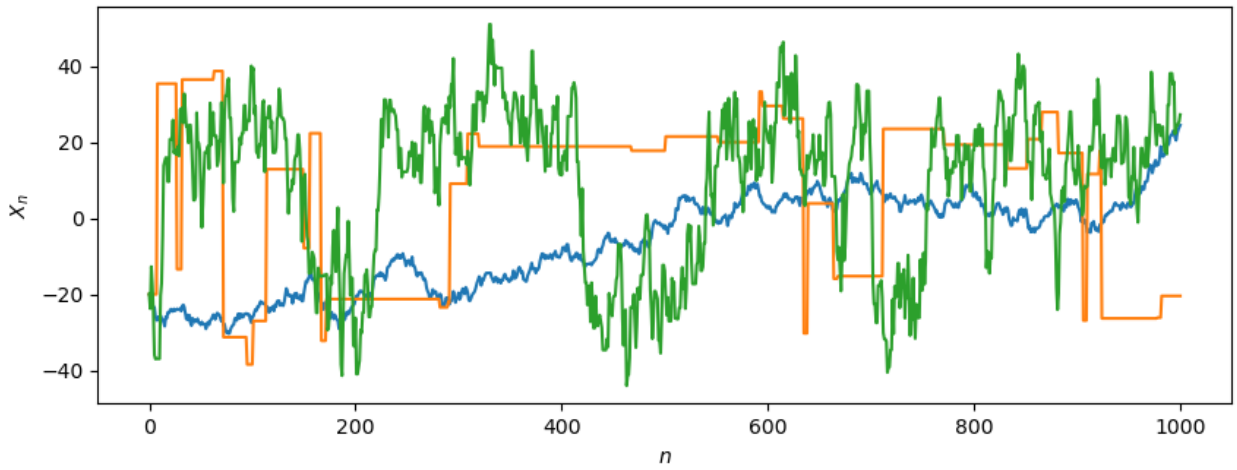
No, we can't because the series of points generated by the Markov chain are *not* independent.

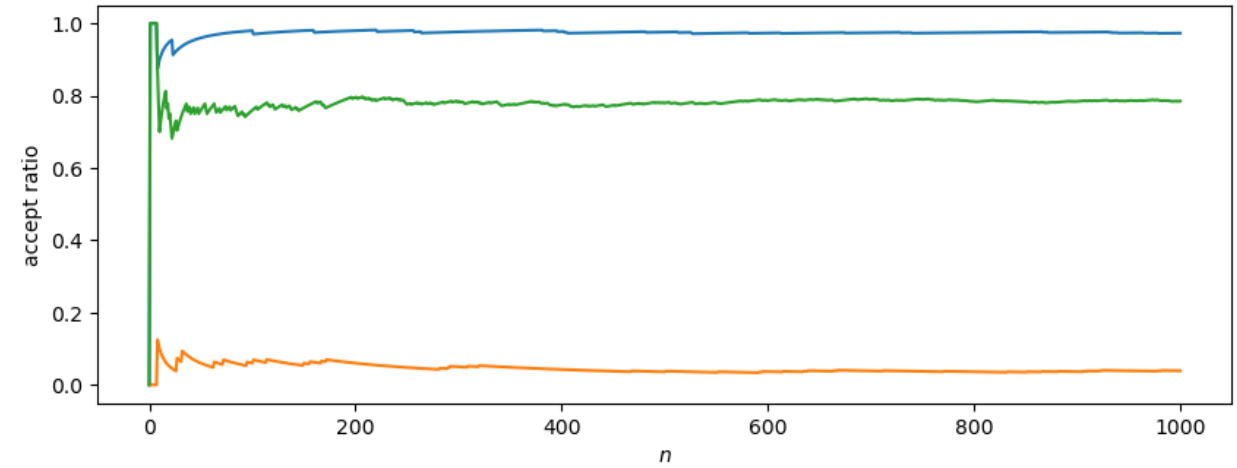
Q2

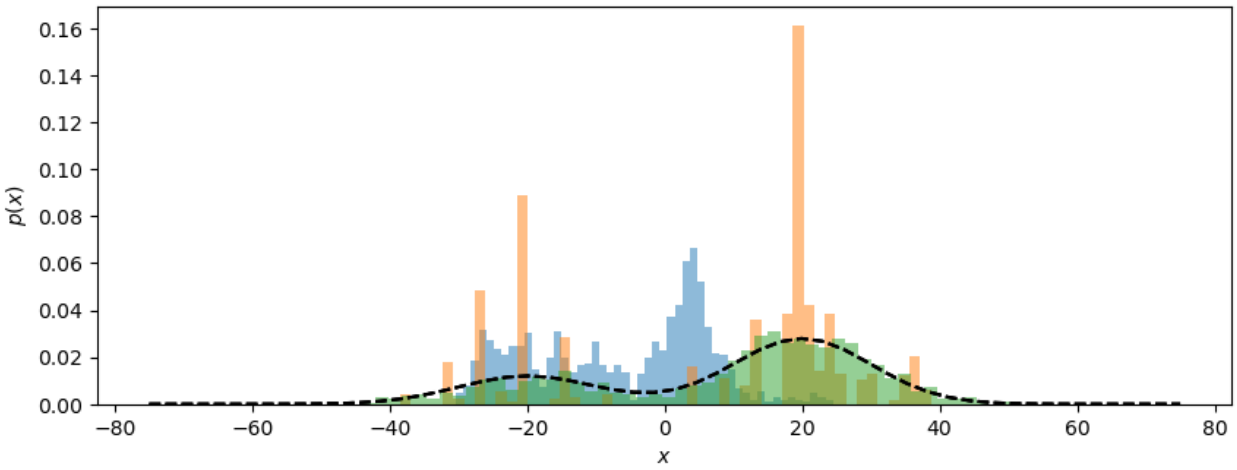
The location (loc) of the stats.normal.pdf is the mean; the scale is the standard deviation. The same applied to the np.random.normal random number generator.

In [ ]:
```
def pmix(x):
    return 0.3*stats.norm.pdf(x, loc=-20, scale=10)+0.7*stats.norm.pdf(x,
```

In [ ]:
```
niter = 10**3
sigma = [1, 500, 8]

plt.figure(figsize=(10, 12))

# Try difference move variance
for s in sigma:
    x=-20.0      # Initial value
    hit = 0      # Counter for acceptance ratio
    xlist = [x]
    hitlist = [hit]

    # Metropolis simulation
    for j in range(niter):
        xp = x + np.random.normal(0,s)
        r = np.random.random()
        rhoMR = pmix(xp)/pmix(x)
        if rhoMR>1.0 or r<rhoMR:
            x = xp
            hit+=1

        xlist.append(x)
        hitlist.append(hit/(j+1))

    # Plot moves
    plt.subplot(3,1,1)
    plt.plot(range(niter+1), xlist)
    plt.xlabel(r'$n$')
    plt.ylabel(r'$X_n$')

    # Plot acceptance ratio
    plt.subplot(3,1,2)
    plt.plot(range(niter+1), hitlist)
    plt.xlabel(r'$n$')
    plt.ylabel('accept ratio')

    # Final histograms
    xval = np.linspace(-75,75, 50)
    plt.subplot(3,1,3)
    plt.hist(xlist, bins=50, density=True, alpha=0.5)
    plt.plot(xval, pmix(xval), 'k--')
    plt.xlabel(r'$x$')
    plt.ylabel(r'$p(x)$')

plt.show()
```







Analysis:

- Blue line ($\sigma = 1$): Variance too low to allow for the exploration of the two peaks. The acceptance ratio is very close to 1, showing that most displacements (moves) are accepted because they are small.
- Orange line ($\sigma = 500$): Large displacements (moves) generated but they are rarely accepted, as shown by the very low acceptance ratio. The random walks sticks as shown in the trajectory plot.
- Green line ($\sigma = 8$): The whole of $p(x)$ is explored with small and large moves that are likely to be accepted. Good trade-off between exploration and reinforcement.

Q3

Define the linear model:

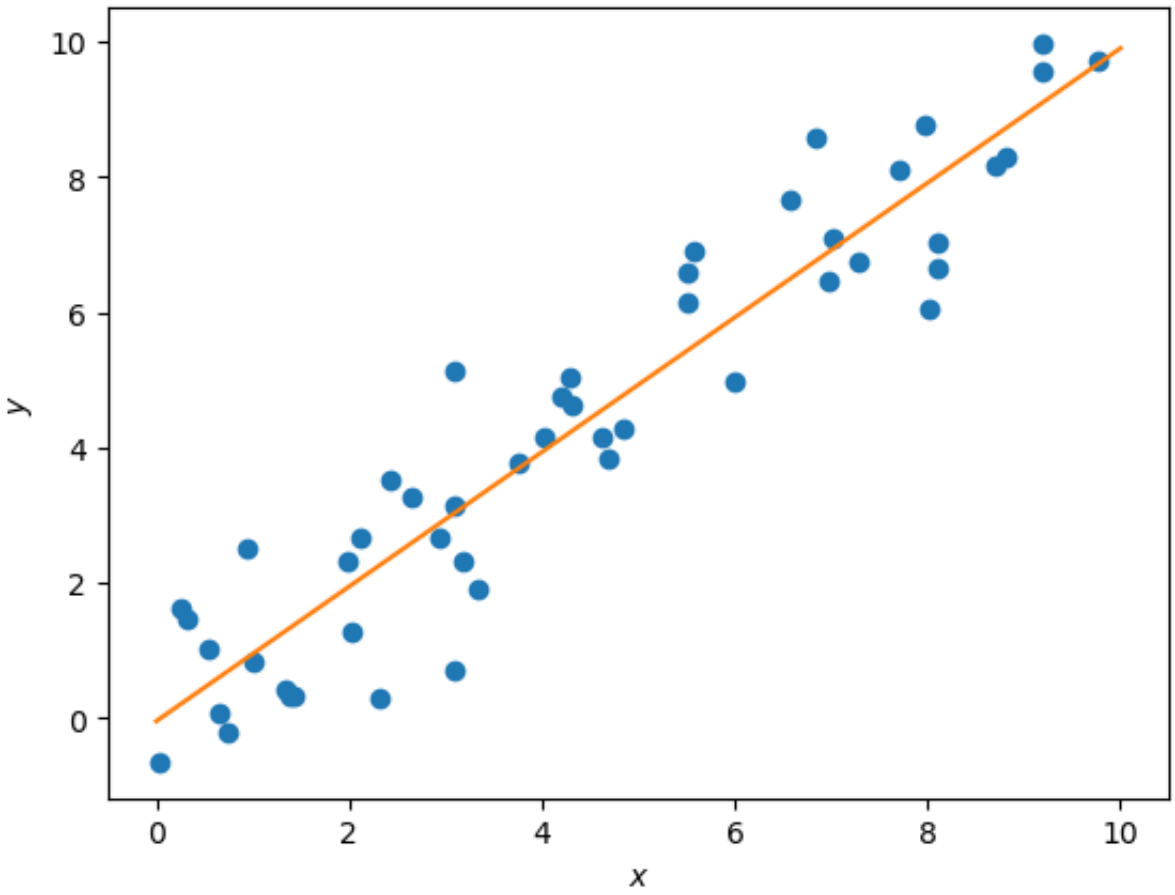
In [ ]:
```
def f(x, a, b):
    return a*x+b
```

(a)

Generate the data from the model:

In [ ]:
```
npoints = 50
areal = 1.0
breal = 0.0
sigmareal = 1.0

xdata = np.random.uniform(low=0.0, high=10.0, size=npoints)
ydata = f(xdata, areal, breal) + np.random.normal(0, sigmareal, size=npoi

slope, intercept, r, p, se = stats.linregress(xdata, ydata)
print('Parameters for linear regression:')
print('Intercept = ', intercept)
print('Slope = ', slope)

xval = np.linspace(0,10,20)
plt.plot(xdata, ydata, 'o')
plt.plot(xval, intercept+slope*xval)
plt.xlabel(r'$x$')
plt.ylabel(r'$y$')
plt.show()
```

```
Parameters for linear regression:
Intercept =  -0.04509374917306719
Slope =  0.9945525716565281
```



(b)

The posterior distribution of the parameters $\theta = (b_0, b_1, \sigma)$ given the data D is

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)}.$$

The likelihood of one data point (x_i, y_i) for the linear model with Gaussian noise is

$$p(x_i, y_i|b_0, b_1, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y_i - f(x_i))^2/2\sigma^2}.$$

The full likelihood $p(D|\theta)$ of the data is the product of this distribution over the data (naive Bayes). For the prior, we have

$$p(\theta) = p(a)p(b)p(\sigma),$$

each marginal being Gaussian.

In [ ]:
```
def logL(x, y, a, b, s):
    val = 0.0
    npts = len(x)
    for i in range(npts):
        val += (y[i]-f(x[i], a, b))**2

    return -val/(2.0*s**2) - npts*np.log(s)
```

In [ ]:
```
def logPriora(a, mua, sa):
    return stats.norm.logpdf(a, loc=mua, scale=sa)
```

In [ ]:
```
def logPriorb(b, mub, sb):
    return stats.norm.logpdf(b, loc=mub, scale=sb)
```

In [ ]:
```
def logPriors(s, mus, ss):
    return stats.norm.logpdf(s, loc=mus, scale=ss)
```

For the priors, we need to define the hyperparameters (means and variances). We can use the data for this or just set some values that we'll change when running our simulations:

In [ ]:
```
hyp_amean = slope
hyp_astddev = 1.0
hyp_bmean = intercept
hyp_bstddev = 1.0
hyp_smean = 1.0
hyp_sstddev = 1.0
```

(c)

For the Metropolis simulation, we use the Gaussian displacements specified in the CW:

In [ ]:
```
# Nb MCMC steps
niter = 10**4
ndiagnostic = 10**3

# Variance for proposals/moves
sa = 0.05
sb = 0.05
ssigma = 0.02

# Initial values
a = 0.5
b = 0.0
s = 0.8
hit = 0

# Container lists
alist = [a]
blist = [b]
slist = [s]

for i in range(niter):

    # Proposals/moves
    atry = a + np.random.normal(0, sa)
    btry = b + np.random.normal(0, sb)
    stry = s + np.random.normal(0, ssigma)

    # log of Metropolis ratio
```

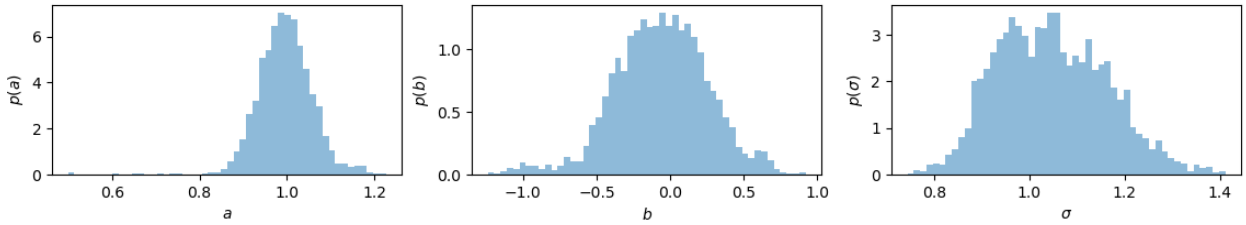
```
    logMR = logL(xdata, ydata, atry, btry, stry) - logL(xdata, ydata, a,
    + logPriora(atry, hyp_amean, hyp_astddev) - logPriora(a, hyp_amean, h
    + logPriorb(btry, hyp_bmean, hyp_bstddev) - logPriorb(b, hyp_bmean, h
    + logPriors(stry, hyp_smean, hyp_sstddev) - logPriors(s, hyp_smean, h

    if logMR>0 or np.log(np.random.random())<logMR:
        a = atry
        b = btry
        s = stry
        hit += 1

    alist.append(a)
    blist.append(b)
    slist.append(s)

    # Monitoring of acceptance ratio
    if i>0 and np.mod(i, ndiagnostic)==0:
        print(i, 'steps ->',  i/niter*100, '% done.', 'Acceptance ratio =

print('Final acceptance ratio = ', hit/niter)
```

```
1000 steps -> 10.0 % done. Acceptance ratio = 0.5114885114885115
2000 steps -> 20.0 % done. Acceptance ratio = 0.5222388805597201
3000 steps -> 30.0 % done. Acceptance ratio = 0.5204931689436855
4000 steps -> 40.0 % done. Acceptance ratio = 0.5106223444138965
5000 steps -> 50.0 % done. Acceptance ratio = 0.5144971005798841
6000 steps -> 60.0 % done. Acceptance ratio = 0.5175804032661223
7000 steps -> 70.0 % done. Acceptance ratio = 0.5186401942579631
8000 steps -> 80.0 % done. Acceptance ratio = 0.522559680039995
9000 steps -> 90.0 % done. Acceptance ratio = 0.5227196978113543
Final acceptance ratio =  0.5233
```

(d)

In [ ]:

```
plt.figure(figsize=(14, 2))

plt.subplot(1,3,1)
plt.hist(alist, bins=50, density=True, alpha=0.5)
plt.xlabel(r'$a$')
plt.ylabel(r'$p(a)$')

plt.subplot(1,3,2)
plt.hist(blist, bins=50, density=True, alpha=0.5)
plt.xlabel(r'$b$')
plt.ylabel(r'$p(b)$')

plt.subplot(1,3,3)
plt.hist(slist, bins=50, density=True, alpha=0.5)
plt.xlabel(r'$\sigma$')
plt.ylabel(r'$p(\sigma)$')
plt.show()
```



Most probable values:

In [ ]:
```
indexa = np.argmax(np.histogram(alist, 50)[0])
indexb = np.argmax(np.histogram(blist, 50)[0])
indexs = np.argmax(np.histogram(slist, 50)[0])
print('Most probable a =', np.histogram(alist, 50)[1][indexa])
print('Most probable b =', np.histogram(blist, 50)[1][indexb])
print('Most probable sigma =', np.histogram(slist, 50)[1][indexs])
print('Comparison with linear regression:')
print('a = ', slope)
print('b = ', intercept)
```

```
Most probable a = 0.979466765305771
Most probable b = -0.07261101271147874
Most probable sigma = 1.0373080056020803
Comparison with linear regression:
a =  0.9945525716565281
b =  -0.04509374917306719
```