Homework 4

1)

$$\int_0^{V_{LSB}} \frac{\left(x - \frac{0+V_{LSB}}{2}\right)^2}{V_{LSB} - 0}\,dx = \left.\frac{\left(x - \frac{V_{LSB}}{2}\right)^3}{3\,V_{LSB}}\right]_0^{V_{LSB}}$$

$$= \frac{\left(V_{LSB} - \frac{V_{LSB}}{2}\right)^3}{3\,V} - \frac{\left(0 - \frac{V_{LSB}}{2}\right)^3}{3\,V}$$

$$= \frac{\left(V_{LSB} - \frac{V_{LSB}}{2}\right)^3}{3\,V} + \frac{\left(\frac{V_{LSB}}{2}\right)^3}{3\,V}$$

$$= \frac{\left(V_{LSB} - \cancel{\frac{V_{LSB}}{2}} + \cancel{\frac{V_{LSB}}{2}}\right)\left(\cancel{V_{LSB}^2} - \cancel{V_{LSB}^2} + \cancel{\frac{V_{LSB}^2}{4}} - \cancel{\frac{V_{LSB}^2}{2}} + \cancel{\frac{V_{LSB}^2}{4}} + \frac{V_{LSB}^2}{4}\right)}{3\,V_{LSB}}$$

$$= \frac{\cancel{V_{LSB}}\ \frac{V_{LSB}^2}{4}}{3\,\cancel{V_{LSB}}} = \frac{V_{LSB}^2}{12}$$

$$V_{LSB} = \frac{V_{ref}}{2^n}$$

$$\sigma = \sqrt{\frac{V_{LSB}^2}{12}} = \sqrt{\frac{1}{12}\left(\frac{V_{ref}}{2^n}\right)^2} = \frac{1}{\sqrt{12}}\ \frac{V_{ref}}{2^n}$$

2) a) $\frac{2}{38} = \frac{1}{19}$ chance that the House will win

b) $\lambda = pN$

$\Rightarrow N = \frac{\lambda}{p} = \frac{2}{\frac{1}{19}} = 38$ (times)

c) the probability that the House wins nothing:

$$\binom{38}{0}\left(\frac{2}{38}\right)^{0}\left(\frac{36}{38}\right)^{38} = 0.128$$

d) $1000 \times \frac{2}{38} \times \$100 = \$5263.16$

e) $\sigma = \sqrt{1000\left(\frac{2}{38}\right)\left(1 - \frac{2}{38}\right)} = 7.06$

f) $$\int_{-\infty}^{26.5} \frac{1}{7.06\sqrt{2\pi}} e^{\frac{-(k-53)^2}{2.(7.06)^2}} dk = 0.00008$$

5) a) $e^{-0.37} = 0.69$

probability: $1 - 0.69 = 0.31$

this is the probability that there will be at least one event

b) $P(k) = \frac{1}{k!} \lambda^k e^{-\lambda} < 0.000001$

$k = 6$, $P(6) = 2.4 \times 10^{-6} > 0.000001$

$k = 7$, $P(7) = 1.3 \times 10^{-7}$

$\Rightarrow k = 7$

```python
frac = .01
# If I want to see how many sigma I need to go away from the mean so that a fraction 'frac' is outside of it - either
# above or below.  Because the function is symmetric, I'll look for the point that frac/2 falls below, which will
# be a negative value
dev = -stats.norm.ppf(frac/2,loc=mu,scale=sigma)
print('A fraction %f of the distribution will lie outside of %.2f sigma.'%(frac,dev))
```

```
A fraction 0.010000 of the distribution will lie outside of 2.58 sigma.
```

```python
frac = .001
# If I want to see how many sigma I need to go away from the mean so that a fraction 'frac' is outside of it - either
# above or below.  Because the function is symmetric, I'll look for the point that frac/2 falls below, which will
# be a negative value
dev = -stats.norm.ppf(frac/2,loc=mu,scale=sigma)
print('A fraction %f of the distribution will lie outside of %.2f sigma.'%(frac,dev))
```

```
A fraction 0.001000 of the distribution will lie outside of 3.29 sigma.
```

```python
frac = .0001
# If I want to see how many sigma I need to go away from the mean so that a fraction 'frac' is outside of it - either
# above or below.  Because the function is symmetric, I'll look for the point that frac/2 falls below, which will
# be a negative value
dev = -stats.norm.ppf(frac/2,loc=mu,scale=sigma)
print('A fraction %f of the distribution will lie outside of %.2f sigma.'%(frac,dev))
```

```
A fraction 0.000100 of the distribution will lie outside of 3.89 sigma.
```

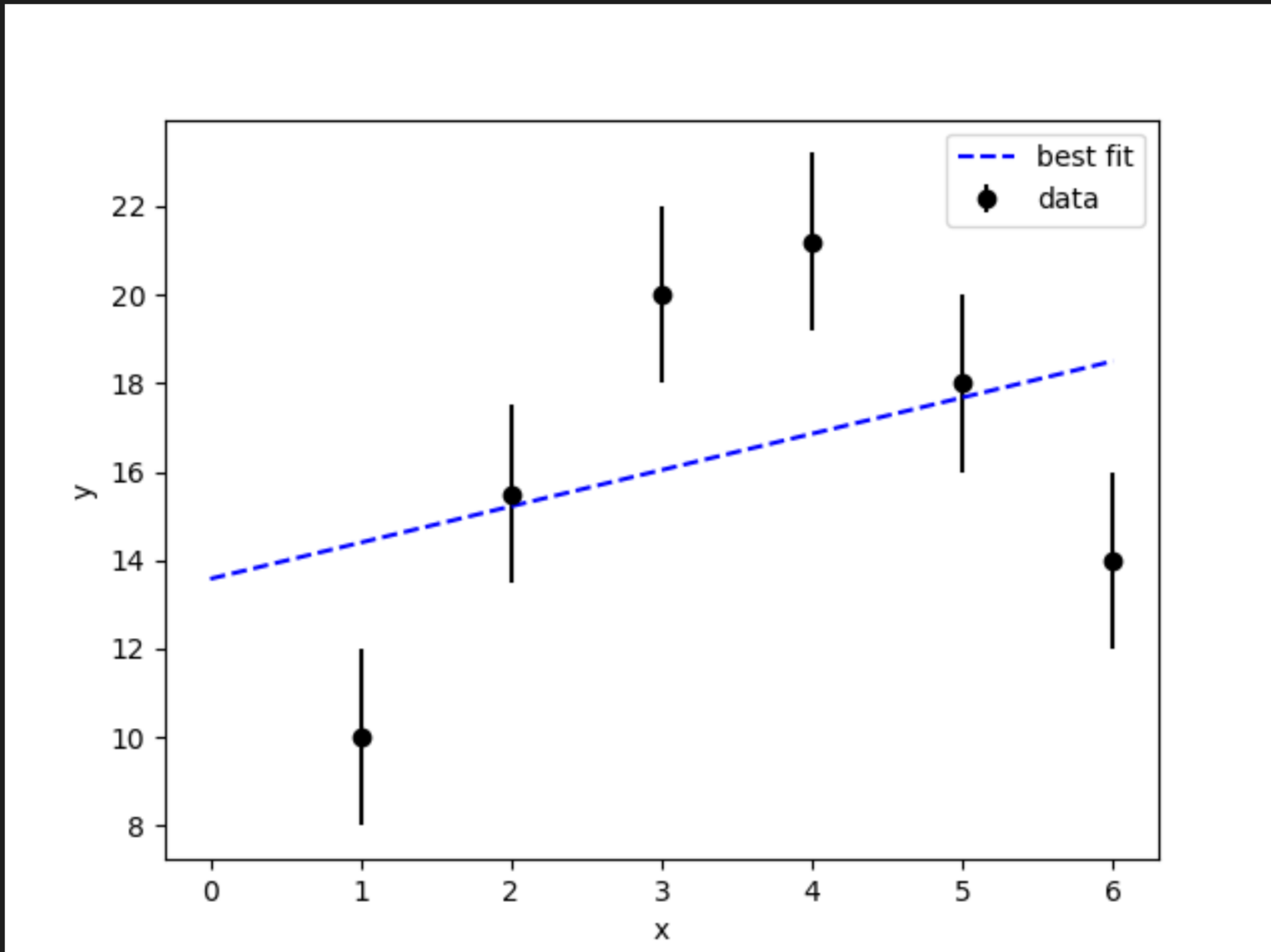
```python
frac = .00001
# If I want to see how many sigma I need to go away from the mean so that a fraction 'frac' is outside of it - either
# above or below.  Because the function is symmetric, I'll look for the point that frac/2 falls below, which will
# be a negative value
dev = -stats.norm.ppf(frac/2,loc=mu,scale=sigma)
print('A fraction %f of the distribution will lie outside of %.2f sigma.'%(frac,dev))
```

```
A fraction 0.000010 of the distribution will lie outside of 4.42 sigma.
```

```python
frac = .000001
# If I want to see how many sigma I need to go away from the mean so that a fraction 'frac' is outside of it - either
# above or below.  Because the function is symmetric, I'll look for the point that frac/2 falls below, which will
# be a negative value
dev = -stats.norm.ppf(frac/2,loc=mu,scale=sigma)
print('A fraction %f of the distribution will lie outside of %.2f sigma.'%(frac,dev))
```

```
A fraction 0.000001 of the distribution will lie outside of 4.89 sigma.
```

```
best fit value of a:   0.8199999999996088
best fit value of b:   13.580000000027429
```



```python
from scipy import optimize
# define the fitting function, in this case, a straight line:
# return y = a*x + b for parameters a and b
def fit_func(x, a, b, c):
    return a*(x**2)+b*x+c

# fill np arrays with the data to be fit:
x_data = np.array([1.0, 2.0, 3.0, 4.0, 5.0, 6.0])
y_data = np.array([10.0, 15.5, 20., 21.2, 18., 14.])
y_unc  = np.array([2.0, 2.0, 2.0, 2.0, 2.0, 2.0])

# plot the raw data
plt.errorbar(x_data, y_data,yerr=y_unc,fmt="ko",label="data")

# calculate best fit curve
par, cov = optimize.curve_fit(fit_func, x_data, y_data, sigma=y_unc)  # include error
# retrieve and print the fitted values of a ;and b:
fit_a = par[0]
fit_b = par[1]
fit_c = par[2]
print("best fit value of a:  ", fit_a)
print("best fit value of b:  ", fit_b)
print("best fit value of c:  ", fit_c)

# plot the best fit line:
xf    = np.linspace(0.0,6.0,100)
yf    = fit_func(xf,fit_a,fit_b, fit_c)
plt.plot(xf,yf,"b--",label="best fit")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.show()
```

```
best fit value of a:   -1.3982142917016842
best fit value of b:   10.60750004314891
best fit value of c:   0.5299999397876907
```