pylab 3

1. Fitting

cleaning data

```
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

xdata, ydata = np.loadtxt(r"C:\Users\lenovo\Desktop\courses\PHY224\assignments\PyLab 3\decay.txt", unpac
xdata, background = np.loadtxt(r"C:\Users\lenovo\Desktop\courses\PHY224\assignments\PyLab 3\background.t

mean_background = background.mean()
mean_background = mean_background/20 # convert to rates
ydata = ydata/20 # convert to rates
ydata = ydata - mean_background# subtract the mean background radiation


sigma_y = np.sqrt(ydata + mean_background)
xdata = (xdata-1)*20    # change the starting time to 0, and scale it up by 20s
```

define function f and g and the theoretical formula

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

def g(x, a, b):
    return b * np.exp(a*x)

def theoretical(I_0, t):
    half_life = 2.6 *60
    return I_0 *(0.5**(t/half_life))
```

linear model fitting

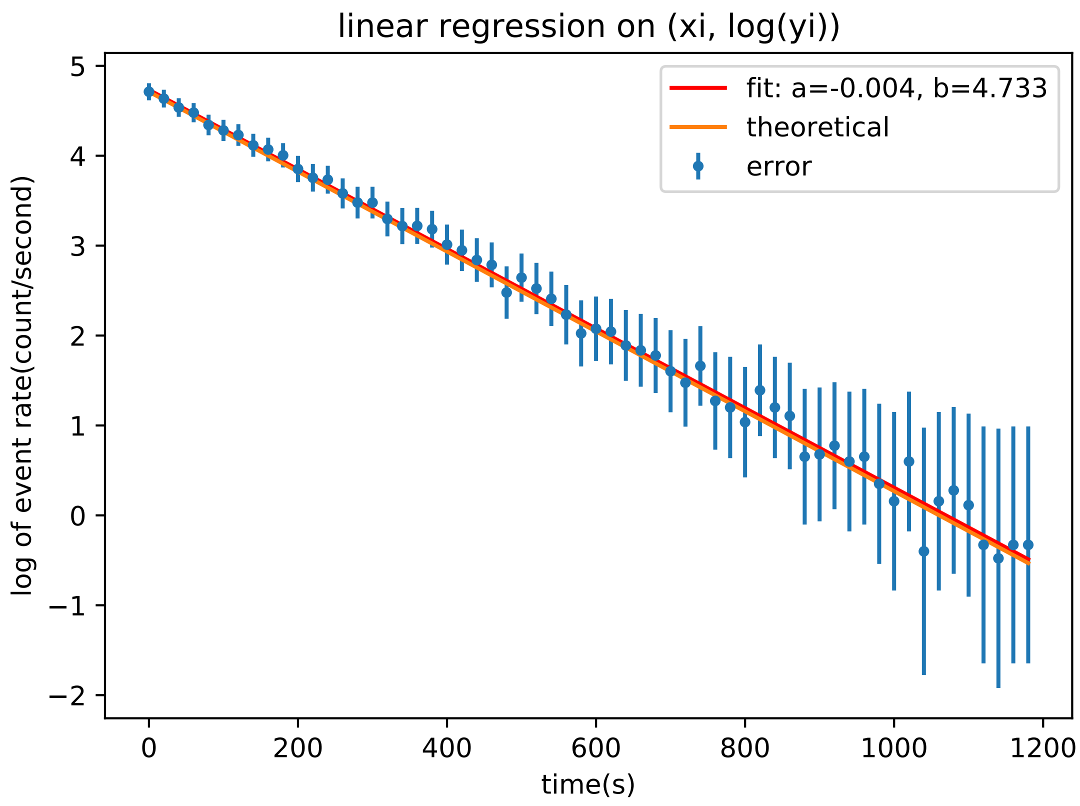
```
# linear regression on (xi, log(yi)) using f
z =np.log(ydata)
sigma_z = sigma_y/ydata
p_opt, p_cov = curve_fit(f, xdata, z, (1, 0), sigma_y, True)

fig, ax = plt.subplots()
ax.plot(xdata, f(xdata, *p_opt), 'r-',
        label='fit: a=%5.3f, b=%5.3f' % tuple(p_opt))
ax.errorbar(xdata, z, yerr=sigma_z, linestyle='None', label="error", marker=".")
```

```
## <ErrorbarContainer object of 3 artists>
```

```
ax.plot(xdata, np.log(theoretical(ydata[0], xdata)), label="theoretical")

ax.set_title('linear regression on (xi, log(yi))')
ax.set_xlabel('time(s)')
ax.set_ylabel('log of event rate(count/second)')
ax.legend()
plt.show()
```



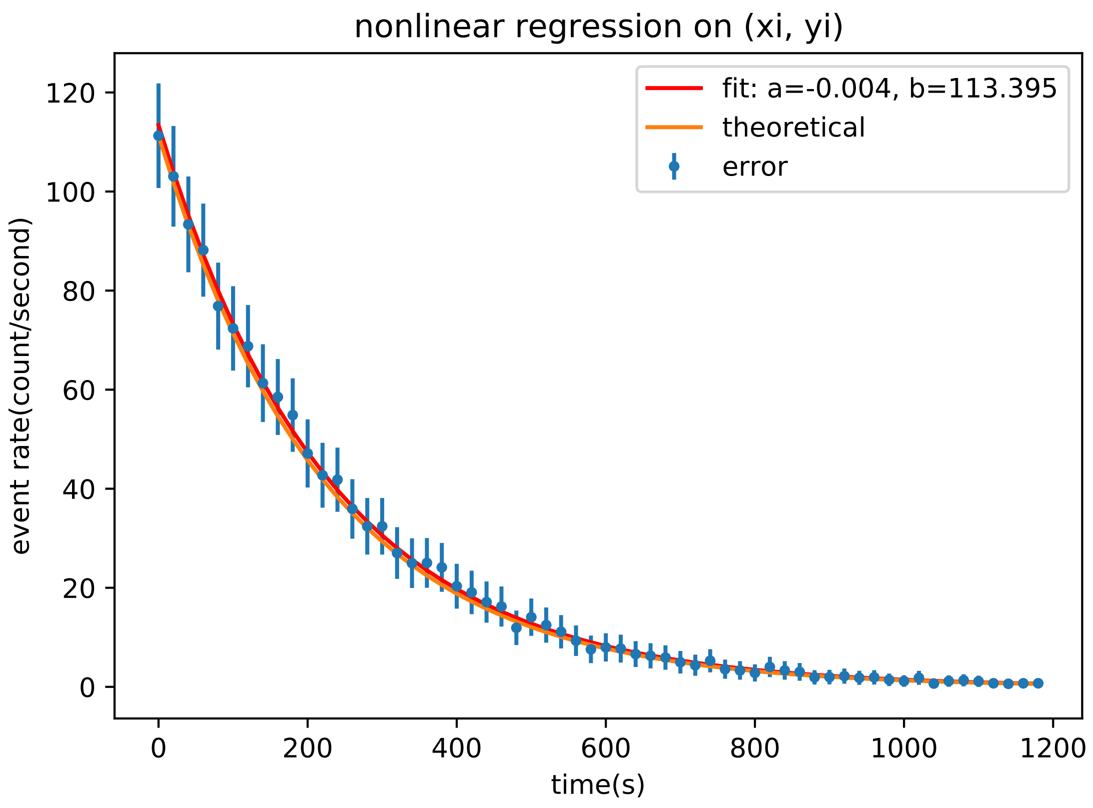
non-linear model fitting

```
# nonlinear regression on (xi, yi) using g
p_opt, p_cov = curve_fit(g, xdata, ydata, (-0.004, 100), sigma_y, True)

fig, ax = plt.subplots()
ax.plot(xdata, g(xdata, *p_opt), 'r-',
        label='fit: a=%5.3f, b=%5.3f' % tuple(p_opt))
ax.errorbar(xdata, ydata, yerr=sigma_y, linestyle='None', label="error", marker=".")
```

```
## <ErrorbarContainer object of 3 artists>
```

```
ax.plot(xdata, theoretical(ydata[0], xdata), label="theoretical")
ax.set_title('nonlinear regression on (xi, yi)')
ax.set_xlabel('time(s)')
ax.set_ylabel('event rate(count/second)')
ax.legend()
plt.show()
```

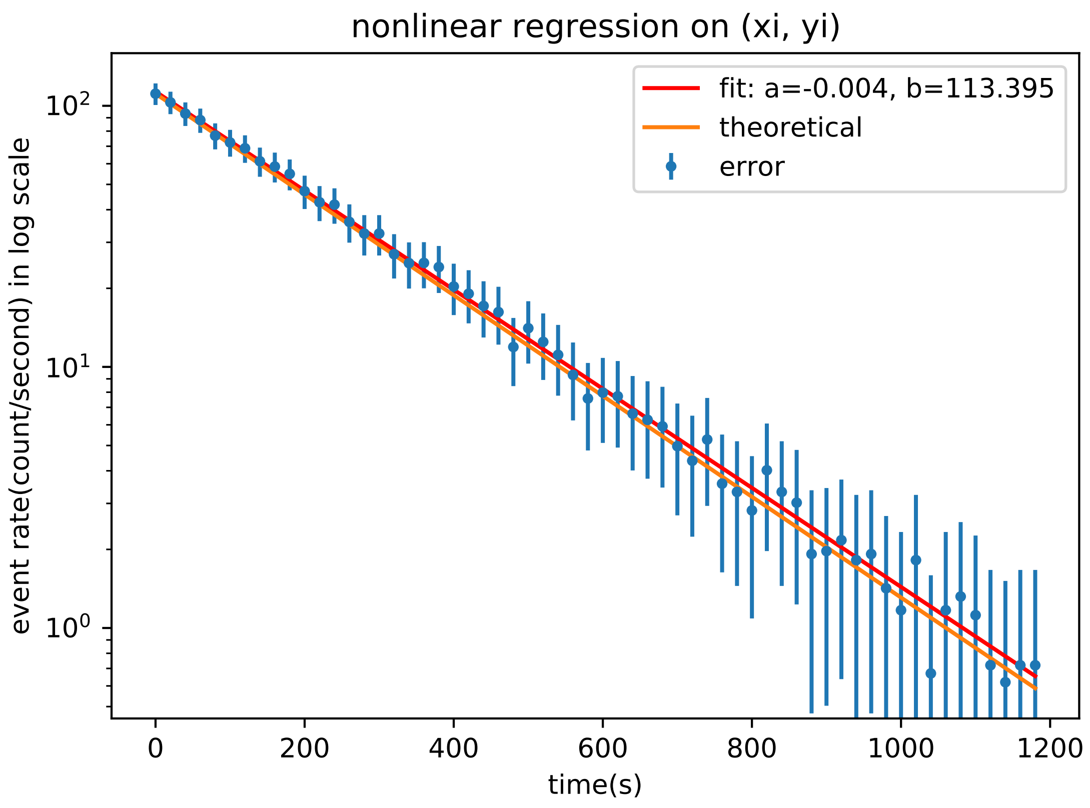


another plot in log scale for the nonlinear model

```
fig, ax = plt.subplots()
ax.semilogy(xdata, g(xdata, *p_opt), 'r-',
        label='fit: a=%5.3f, b=%5.3f' % tuple(p_opt))
ax.errorbar(xdata, ydata, yerr=sigma_y, linestyle='None', label="error", marker=".")
```

```
## <ErrorbarContainer object of 3 artists>
```

```
ax.plot(xdata, theoretical(ydata[0], xdata), label="theoretical")
ax.set_title('nonlinear regression on (xi, yi)')
ax.set_xlabel('time(s)')
ax.set_ylabel('event rate(count/second) in log scale')
ax.legend()
plt.show()
```



2. Quality of Fit

a) Variance of parameters

for linear fitting

```
p_opt, p_cov = curve_fit(f, xdata, z, (1, 0), sigma_y, True)
print("standard deviation of the half-life:{:.3f}".format(np.sqrt(p_cov[0,0])/p_opt[0]**2))
```

```
## standard deviation of the half-life:58.587
```

```
print("sample_half_life: {:.3f}".format(1/p_opt[0]*np.log(0.5)))
```

```
## sample_half_life: 156.619
```

```
print("normal_half_life: {:.3f}".format(2.6*60))
```

```
## normal_half_life: 156.000
```

The experimental half life is 156.619 ± 58.587. The nominal half-life (156 seconds) falls in the range.

for nonlinear fitting

```
p_opt, p_cov = curve_fit(g, xdata, ydata, (-0.004, 100), sigma_y, True)
print("standard deviation of the half-life:{:.3f}".format(np.sqrt(p_cov[0,0])/p_opt[0]**2))
```

```
## standard deviation of the half-life:6.838
```

```
print("sample_half_life: {:.3f}".format(1/p_opt[0]*np.log(0.5)))
```

```
## sample_half_life: 158.661
```

```
print("normal_half_life: {:.3f}".format(2.6*60))
```

```
## normal_half_life: 156.000
```

The experimental half life is 158.661 ± 6.838. The nominal half-life (156 seconds) falls in the range.

Both linear and non-linear models produce confidence interval that contains the nominal half-life (156 seconds).But, based on variance of parameters, the nonlinear model is better. The nonlinear model produces a much smaller confidence interval.

b) Reduced χ^2

```
def chi2(xdata, ydata, measurement_error, model_function):
    if model_function == f:
        p_opt, p_cov = curve_fit(model_function, xdata, ydata, (1, 0), measurement_error, True)
    if model_function == g:
        p_opt, p_cov = curve_fit(model_function, xdata, ydata, (-0.004, 100), measurement_error, True)
    n = len(p_opt)
    N = len(xdata)
    v = N - n
    expected = model_function(xdata, *p_opt)
    statistics = sum(((ydata - expected) / measurement_error) ** 2) / v
    return statistics
```

for linear fitting

```
z =np.log(ydata)
sigma_z = sigma_y/ydata
chi2(xdata, z, sigma_z, f)
```

```
## 0.06139646016646158
```

for nonlinear fitting

```
chi2(xdata, ydata, sigma_y, g)
```

```
## 0.06504453860755562
```

Based on chi2, the linear model is the better model.