```
import matplotlib.pyplot as plt
import numpy as np
import scipy.optimize as scp
import scipy.special as ss
import os

# Vi starter med at importere, de andre scripts som  skal bruge til
# henholdsvis kalibrering og databehandling

exec(open('Kalibrering/kalibrering.py').read())
exec(open('../Scripts/Statistik.py').read())
exec(open('../Scripts/data_renser.py').read())

grader = 15
theta = grader*(2*np.pi/360)

fig, ax = plt.subplots(2, 3, figsize = (20, 10))
ax = ax.ravel()

# Her defineres fitte funktionen. Vi fitter efter en andengradsparabel,
# t0-parametren giver mulighed for at rykke pÃ¥ parablens toppunkt, mens
# heaviside-funktionen (indikator-funktion) giver funktionen for at vÃ¦re lig 0
# i starten.

def fit(t,*p):
    a = p[0]
    t0 = p[1]
    c = p[2]
    return np.heaviside(t-t0,1)*(1/2*a*(t-t0)**2)+c

# Den fÃ¸lgende funktion plotter data.

def plot_data(data, ax, labels, title, kali):

 # Data() er en class defineret i data_renser.py

    sol1 = Data(data)
    x = func(sol1.points, *kali)
    t = sol1.t

 # .rinse() er en metode defineret i data_renser.py, som leder efter
 # outliers.

    mask = sol1.rinse([[-1, 0.1], [0.4, 0.3], [0.6, 0.4]])

# Outliers markeres med blÃ¥, de resterende punkter med rÃ¸d.

    ax.scatter(t[~mask], x[~mask], color = 'blue', label = 'outliers')
    ax.scatter(t[mask], x[mask], color = 'red', label = 'data points')

    guess_params = [1,-0.17,0.05]

# Funktionen fittes, estimatet af t0 er lidt fÃ¸lsomt, sÃ¥ der sÃ¦ttes nogle
# bounds for den.

    popt,pcov = scp.curve_fit(fit, t[mask], x[mask],
                              guess_params, bounds = ((-10, -0.3, -10),(10, -0.1, 10)))



    t_fit = np.linspace(t[mask][0], t[mask][-1], 1000)
    ax.plot(t_fit, fit(t_fit, *popt), color = 'k', linewidth = 2,
            label = 'fitted function')
```

```
# spredningen pÃ¥ parametrene hives ud covariansmatricen, den mÃ¥lte acceleration
# gemmes ogsÃ¥

    var_a = round(np.sqrt(np.diag(pcov)[0]), 2)
    eksp_a = round(popt[0], 3)

# Propagation_function, er en function defineret i Statistik.py. Den
# implementerer fejlpropagering via. den funktionelle metode.

    error = propagation_function(t_fit, fit, list(popt), pcov)
    ax.fill_between(t_fit, fit(t_fit, *popt) + error,
                    fit(t_fit, *popt) - error, alpha = 0.3)

    ax.set_ylim(-0.2,0.7)
    ax.set_ylabel('x/m')
    ax.set_xlabel('t/s')
    ax.set_title(title)
    ax.legend()

    print( "Teoretisk a = {}, ".format(round(np.sin(theta)*9.82*0.66, 3))+
          "Eksperimentel a = {} $\pm$ {}".format(eksp_a, var_a))


plot_data("Sol1", ax[0], labels = None, title = 'solid cylinder 1',
          kali = kali)

plot_data("Sol2", ax[1], labels = None, title = 'solid cylinder 2',
          kali = kali)

plot_data("Sol3", ax[2], labels = None, title = 'solid cylinder 3',
          kali = kali)

plot_data("Sol4", ax[3], labels = None, title = 'solid cylinder 4',
          kali = kali)

plot_data("Sol5", ax[4], labels = None, title = 'solid cylinder 5',
          kali = kali)

ax[5].remove()
plt.show()

    ###############

fig.savefig('Plots/solid_ruller.png')
```