Assignment 1

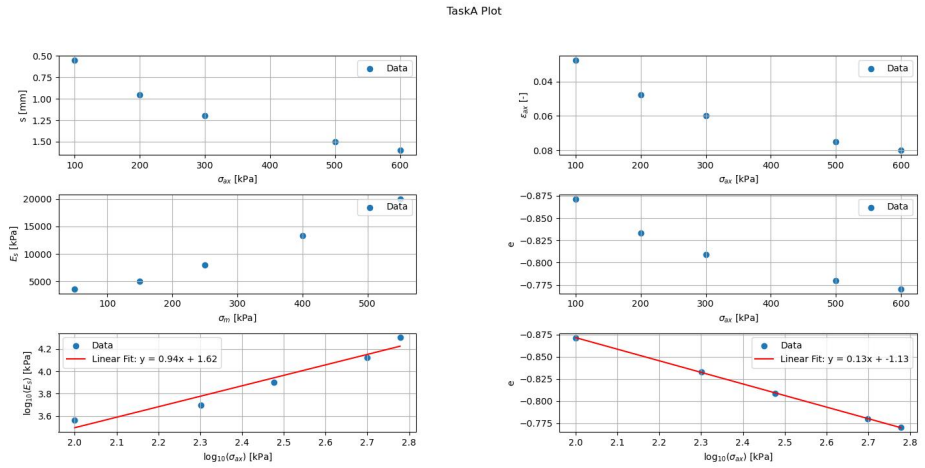
Task A: Compression behaviour of soil

1. Calculating results

recordData.txt

	sig_ax[kPa]	settlement[mm]	eps_ax[-]	Es[kPa]	log_Es[kPa]	log_sig_ax[kPa]	e	sig_m[kPa]
1	100.00000	0.55000	0.02750	3636.36364	3.56067	2.00000	-0.87153	50.00000
2	200.00000	0.95000	0.04750	5000.00000	3.69897	2.30103	-0.83304	150.00000
3	300.00000	1.20000	0.06000	8000.00000	3.90309	2.47712	-0.80898	250.00000
4	500.00000	1.50000	0.07500	13333.33333	4.12494	2.69897	-0.78012	400.00000
5	600.00000	1.60000	0.08000	20000.00000	4.30103	2.77815	-0.77050	550.00000

2. Plot results



from above figure (5) _linear regression for log_sig_ax-log_Es, the soil parameter/α = 0.94

from above figure (6)_linear regression for log_sig_ax-e, the compression index/C_c = 0.13

3. Source code

Remark: The below source code contains the calculation step.

"""

Task A : Compression behaviour of soil

"""

```
import numpy as np
import matplotlib.pyplot as plt


#================================
# Task A
#================================


# Experimental data from the odemetric test
sig_ax = [100, 200, 300, 500, 600]
settlement = [0.55, 0.95, 1.20, 1.50, 1.60]
density_s = 2.65
h0 = 20.0
d = 50.7
m_d = 55.6
```

```
# Calculation data recording
def recordData(sig_ax, settlement):
    eps_ax = [s / h0 for s in settlement]        # eps_ax

    sig_m = [0.5 * (sig_ax[i+1] + sig_ax[i]) for i in range(0, len(sig_ax)-1)]
    sig_m.insert(0, 0.5 * (sig_ax[0] + 0))
    Es = [(sig_ax[i+1] - sig_ax[i]) / (eps_ax[i+1] - eps_ax[i]) for i in range(0, len(sig_ax)-1)]
    Es.insert(0, (sig_ax[0] + 0) / (eps_ax[0] - 0))      # odemetric modulus
    log_Es = np.log10(Es)
    log_sig_ax = np.log10(sig_ax)

    V_solid = m_d / density_s
    V_total = [np.pi * ((d * 0.1 / 2) ** 2) * (h0 * 0.1 - s * 0.1) for s in settlement]
    V_void = [V_t - V_solid for V_t in V_total]
    e = [-1 * V_v / V_solid for V_v in V_void]      # void ratio

    data = np.array([sig_ax, settlement, eps_ax, Es, log_Es, log_sig_ax, e, sig_m]).T

    return data

# output to a text file
def printResults(data):
    with open('recordData.txt', 'w') as outputfile:
        outputfile.write('{:<17} {:<17} {:<17} {:<17} {:<17} {:<17} {:<17} {:<17}\n'.format(
            'sig_ax[kPa]', 'settlement[mm]', 'eps_ax[-]', 'Es[kPa]', 'log_Es[kPa]', 'log_sig_ax[kPa]', 'e',
'sig_m[kPa]'))
        for row in data:
            sig_ax, settlement, eps_ax, Es, log_Es, log_sig_ax, e, sig_m = row
            outputfile.write('{:<17.5f} {:<17.5f} {:<17.5f} {:<17.5f} {:<17.5f} {:<17.5f} {:<17.5f}
{:<17.5f}\n'.format(
                sig_ax, settlement, eps_ax, Es, log_Es, log_sig_ax, e, sig_m))

def data_plot(data):
    # create the figure and the subplots
    fig, axs = plt.subplots(3, 2, figsize=(17, 8))
    fig.subplots_adjust(hspace=0.4, wspace=0.4)

    # Plot for sigma_ax-s
    axs[0, 0].scatter(data[:, 0], data[:, 1], label='Data')
    axs[0, 0].set_xlabel('$\sigma_{ax}$ [kPa]')
    axs[0, 0].set_ylabel('s [mm]')
    axs[0, 0].grid(which='both')
    axs[0, 0].legend()
    axs[0, 0].invert_yaxis()
```

```
# Plot for sigma_ax-eps_ax
axs[0, 1].scatter(data[:, 0], data[:, 2], label='Data')
axs[0, 1].set_xlabel('$\sigma_{ax}$ [kPa]')
axs[0, 1].set_ylabel('$\epsilon_{ax}$ [-]')
axs[0, 1].grid(which='both')
axs[0, 1].legend()
axs[0, 1].invert_yaxis()

# Plot for sigma_m-Es
axs[1, 0].scatter(data[:, 7], data[:, 3], label='Data')
axs[1, 0].set_xlabel('$\sigma_{m}$ [kPa]')
axs[1, 0].set_ylabel('$E_s$ [kPa]')
axs[1, 0].grid(which='both')
axs[1, 0].legend()

# Plot for sigma_ax-e
axs[1, 1].scatter(data[:, 0], data[:, 6], label='Data')
axs[1, 1].set_xlabel('$\sigma_{ax}$ [kPa]')
axs[1, 1].set_ylabel('e')
axs[1, 1].grid(which='both')
axs[1, 1].legend()
axs[1, 1].invert_yaxis()

# Plot and linear regression for log_sig_ax-log_Es
axs[2, 0].scatter(data[:, 5], data[:, 4], label='Data')
axs[2, 0].set_xlabel('$\log_{10}(\sigma_{ax})$ [kPa]')
axs[2, 0].set_ylabel('$\log_{10}(E_s)$ [kPa]')
axs[2, 0].grid(which='both')
axs[2, 0].legend()

# Perform linear regression
slope, intercept = np.polyfit(data[:, 5], data[:, 4], 1)
fit_x = np.array([min(data[:, 5]), max(data[:, 5])])
fit_y = slope * fit_x + intercept
axs[2, 0].plot(fit_x, fit_y, color='red', label=f'Linear Fit: y = {slope:.2f}x + {intercept:.2f}')
axs[2, 0].legend()

# Plot and linear regression for log_sig_ax-e
axs[2, 1].scatter(data[:, 5], data[:, 6], label='Data')
axs[2, 1].set_xlabel('$\log_{10}(\sigma_{ax})$ [kPa]')
axs[2, 1].set_ylabel('e')
axs[2, 1].grid(which='both')
axs[2, 1].legend()
```

```
    axs[2, 1].invert_yaxis()

    # Perform linear regression
    slope, intercept = np.polyfit(data[:, 5], data[:, 6], 1)
    fit_x = np.array([min(data[:, 5]), max(data[:, 5])])
    fit_y = slope * fit_x + intercept
    axs[2, 1].plot(fit_x, fit_y, color='red', label=f'Linear Fit: y = {slope:.2f}x + {intercept:.2f}')
    axs[2, 1].legend()

    fig.suptitle('TaskA Plot')

    # show figure
    plt.show()

data = recordData(sig_ax, settlement)
printResults(data)
data_plot(data)
```

Task B: Extend the Python routines and run simulations

1. Output data

(1) output_iso_ohde

eps_p	eps_q	p	q	eps_1	eps_2	sigma_1	sigma_2	epor	stateVar2
0.00000	0.00000	100.0	0.0	0.00000	0.00000	100.0	100.0	0.92400	0.00000
0.00300	0.00000	106.8	0.0	0.00100	0.00100	106.8	106.8	0.91823	0.00000
0.00600	0.00000	113.9	0.0	0.00200	0.00200	113.9	113.9	0.91247	0.00000
0.00900	0.00000	121.6	0.0	0.00300	0.00300	121.6	121.6	0.90674	0.00000
0.01200	0.00000	129.7	0.0	0.00400	0.00400	129.7	129.7	0.90102	0.00000
0.01500	0.00000	138.3	0.0	0.00500	0.00500	138.3	138.3	0.89531	0.00000
0.01800	0.00000	147.4	0.0	0.00600	0.00600	147.4	147.4	0.88963	0.00000
0.02100	0.00000	157.2	0.0	0.00700	0.00700	157.2	157.2	0.88396	0.00000
0.02400	0.00000	167.5	0.0	0.00800	0.00800	167.5	167.5	0.87831	0.00000
0.02700	0.00000	178.4	0.0	0.00900	0.00900	178.4	178.4	0.87267	0.00000
0.03000	0.00000	190.0	0.0	0.01000	0.01000	190.0	190.0	0.86705	0.00000
0.03300	0.00000	202.4	0.0	0.01100	0.01100	202.4	202.4	0.86145	0.00000
0.03600	0.00000	215.4	0.0	0.01200	0.01200	215.4	215.4	0.85587	0.00000
0.03900	0.00000	229.3	0.0	0.01300	0.01300	229.3	229.3	0.85030	0.00000
0.04200	0.00000	244.0	0.0	0.01400	0.01400	244.0	244.0	0.84475	0.00000
0.04500	0.00000	259.6	0.0	0.01500	0.01500	259.6	259.6	0.83921	0.00000
0.04800	0.00000	276.1	0.0	0.01600	0.01600	276.1	276.1	0.83370	0.00000
0.05100	0.00000	293.6	0.0	0.01700	0.01700	293.6	293.6	0.82820	0.00000
0.05400	0.00000	312.1	0.0	0.01800	0.01800	312.1	312.1	0.82271	0.00000
0.05700	0.00000	331.7	0.0	0.01900	0.01900	331.7	331.7	0.81724	0.00000

0.06000	0.00000	352.5	0.0	0.02000	0.02000	352.5	352.5	0.81179	0.00000
0.06300	0.00000	374.5	0.0	0.02100	0.02100	374.5	374.5	0.80636	0.00000
0.06600	0.00000	397.7	0.0	0.02200	0.02200	397.7	397.7	0.80094	0.00000
0.06900	0.00000	422.4	0.0	0.02300	0.02300	422.4	422.4	0.79553	0.00000
0.07200	0.00000	448.4	0.0	0.02400	0.02400	448.4	448.4	0.79015	0.00000
0.07500	0.00000	476.0	0.0	0.02500	0.02500	476.0	476.0	0.78478	0.00000
0.07800	0.00000	505.1	0.0	0.02600	0.02600	505.1	505.1	0.77942	0.00000
0.08100	0.00000	535.9	0.0	0.02700	0.02700	535.9	535.9	0.77408	0.00000
0.08400	0.00000	568.4	0.0	0.02800	0.02800	568.4	568.4	0.76876	0.00000
0.08700	0.00000	602.9	0.0	0.02900	0.02900	602.9	602.9	0.76346	0.00000
0.09000	0.00000	639.2	0.0	0.03000	0.03000	639.2	639.2	0.75817	0.00000
0.09300	0.00000	677.6	0.0	0.03100	0.03100	677.6	677.6	0.75289	0.00000
0.09600	0.00000	718.2	0.0	0.03200	0.03200	718.2	718.2	0.74763	0.00000
0.09900	0.00000	761.0	0.0	0.03300	0.03300	761.0	761.0	0.74239	0.00000
0.10200	0.00000	806.2	0.0	0.03400	0.03400	806.2	806.2	0.73716	0.00000
0.10500	0.00000	854.0	0.0	0.03500	0.03500	854.0	854.0	0.73195	0.00000
0.10800	0.00000	904.4	0.0	0.03600	0.03600	904.4	904.4	0.72676	0.00000
0.11100	0.00000	957.5	0.0	0.03700	0.03700	957.5	957.5	0.72157	0.00000
0.11400	0.00000	1013.6	0.0	0.03800	0.03800	1013.6	1013.6	0.71641	0.00000

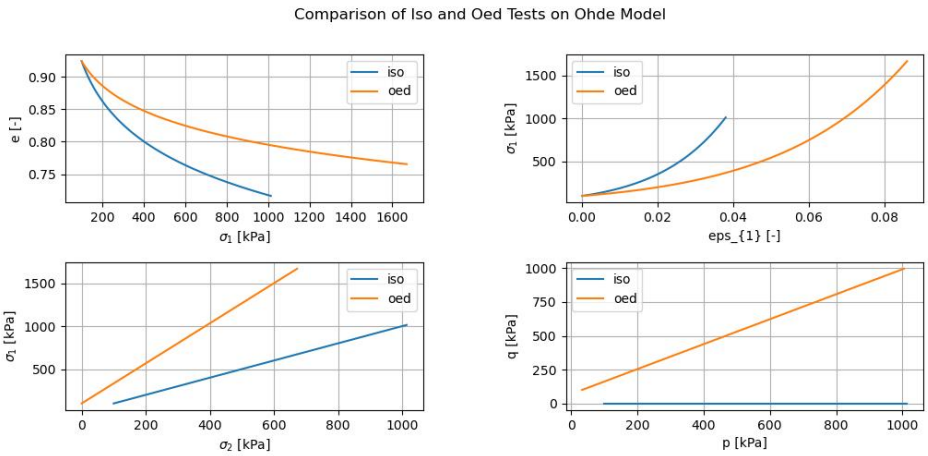
(2) output_oed_ohde

eps_p	eps_q	p	q	eps_1	eps_2	sigma_1	sigma_2	epor	stateVar2
0.00000	0.00000	33.3	100.0	0.00000	0.00000	100.0	0.0	0.92400	0.00000
0.00100	0.00067	35.6	102.1	0.00100	0.00000	103.6	1.6	0.92208	0.00000
0.00200	0.00133	37.9	104.2	0.00200	0.00000	107.4	3.2	0.92015	0.00000
0.00300	0.00200	40.3	106.4	0.00300	0.00000	111.3	4.8	0.91823	0.00000
0.00400	0.00267	42.8	108.7	0.00400	0.00000	115.3	6.6	0.91632	0.00000
0.00500	0.00333	45.4	111.1	0.00500	0.00000	119.5	8.3	0.91440	0.00000
0.00600	0.00400	48.0	113.6	0.00600	0.00000	123.8	10.2	0.91248	0.00000
0.00700	0.00467	50.8	116.1	0.00700	0.00000	128.2	12.1	0.91057	0.00000
0.00800	0.00533	53.6	118.7	0.00800	-0.00000	132.8	14.1	0.90866	0.00000
0.00900	0.00600	56.6	121.4	0.00900	0.00000	137.5	16.1	0.90675	0.00000
0.01000	0.00667	59.6	124.2	0.01000	0.00000	142.4	18.2	0.90485	0.00000
0.01100	0.00733	62.7	127.1	0.01100	0.00000	147.5	20.4	0.90294	0.00000
0.01200	0.00800	66.0	130.1	0.01200	0.00000	152.7	22.6	0.90104	0.00000
0.01300	0.00867	69.3	133.2	0.01300	0.00000	158.1	24.9	0.89914	0.00000
0.01400	0.00933	72.8	136.4	0.01400	0.00000	163.7	27.3	0.89724	0.00000
0.01500	0.01000	76.4	139.7	0.01500	0.00000	169.5	29.8	0.89534	0.00000
0.01600	0.01067	80.0	143.1	0.01600	0.00000	175.5	32.3	0.89345	0.00000
0.01700	0.01133	83.9	146.6	0.01700	0.00000	181.6	35.0	0.89155	0.00000
0.01800	0.01200	87.8	150.3	0.01800	0.00000	188.0	37.7	0.88966	0.00000
0.01900	0.01267	91.9	154.0	0.01900	0.00000	194.5	40.5	0.88777	0.00000
0.02000	0.01333	96.1	157.9	0.02000	0.00000	201.3	43.4	0.88588	0.00000
0.02100	0.01400	100.4	161.9	0.02100	0.00000	208.3	46.4	0.88400	0.00000
0.02200	0.01467	104.9	166.0	0.02200	0.00000	215.6	49.5	0.88211	0.00000

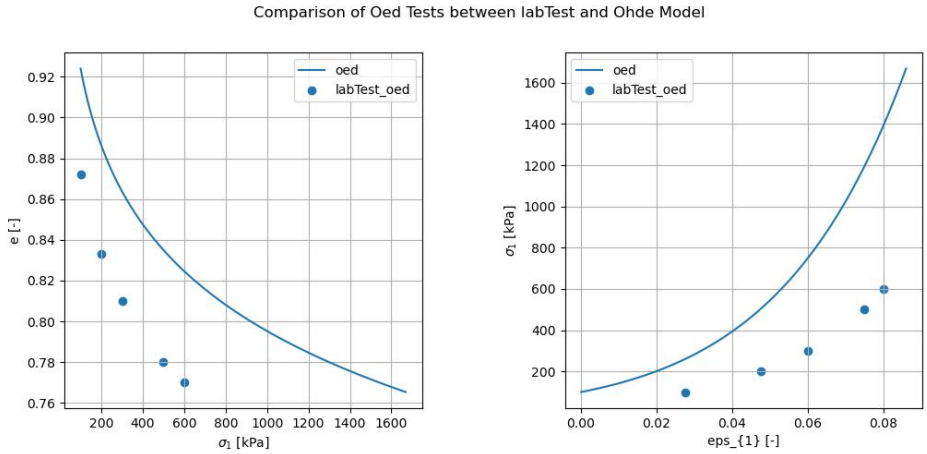
0.02300	0.01533	109.5	170.3	0.02300	0.00000	223.0	52.7	0.88023	0.00000
0.02400	0.01600	114.3	174.7	0.02400	0.00000	230.7	56.0	0.87835	0.00000
0.02500	0.01667	119.2	179.3	0.02500	0.00000	238.7	59.4	0.87647	0.00000
0.02600	0.01733	124.3	184.0	0.02600	0.00000	246.9	63.0	0.87460	0.00000
0.02700	0.01800	129.5	188.8	0.02700	0.00000	255.4	66.6	0.87272	0.00000
0.02800	0.01867	135.0	193.8	0.02800	0.00000	264.2	70.4	0.87085	0.00000
0.02900	0.01933	140.5	199.0	0.02900	0.00000	273.2	74.2	0.86898	0.00000
0.03000	0.02000	146.3	204.3	0.03000	0.00000	282.5	78.2	0.86711	0.00000
0.03100	0.02067	152.3	209.8	0.03100	0.00000	292.1	82.3	0.86524	0.00000
0.03200	0.02133	158.4	215.5	0.03200	0.00000	302.1	86.6	0.86338	0.00000
0.03300	0.02200	164.8	221.3	0.03300	0.00000	312.3	91.0	0.86151	0.00000
0.03400	0.02267	171.3	227.4	0.03400	0.00000	322.9	95.5	0.85965	0.00000
0.03500	0.02333	178.1	233.6	0.03500	0.00000	333.8	100.2	0.85779	0.00000
0.03600	0.02400	185.0	240.0	0.03600	0.00000	345.0	105.0	0.85593	0.00000
0.03700	0.02467	192.2	246.7	0.03700	0.00000	356.6	110.0	0.85408	0.00000
0.03800	0.02533	199.6	253.5	0.03800	0.00000	368.6	115.1	0.85222	0.00000
0.03900	0.02600	207.3	260.6	0.03900	0.00000	381.0	120.4	0.85037	0.00000
0.04000	0.02667	215.1	267.8	0.04000	0.00000	393.7	125.9	0.84852	0.00000
0.04100	0.02733	223.3	275.3	0.04100	0.00000	406.8	131.5	0.84667	0.00000
0.04200	0.02800	231.7	283.1	0.04200	0.00000	420.4	137.3	0.84483	0.00000
0.04300	0.02867	240.3	291.1	0.04300	0.00000	434.3	143.3	0.84298	0.00000
0.04400	0.02933	249.2	299.3	0.04400	0.00000	448.7	149.5	0.84114	0.00000
0.04500	0.03000	258.4	307.8	0.04500	0.00000	463.6	155.8	0.83930	0.00000
0.04600	0.03067	267.9	316.5	0.04600	0.00000	478.9	162.4	0.83746	0.00000
0.04700	0.03133	277.6	325.5	0.04700	0.00000	494.7	169.1	0.83562	0.00000
0.04800	0.03200	287.7	334.8	0.04800	0.00000	510.9	176.1	0.83379	0.00000
0.04900	0.03267	298.1	344.4	0.04900	0.00000	527.7	183.3	0.83195	0.00000
0.05000	0.03333	308.8	354.3	0.05000	0.00000	545.0	190.7	0.83012	0.00000
0.05100	0.03400	319.8	364.4	0.05100	0.00000	562.8	198.3	0.82829	0.00000
0.05200	0.03467	331.2	374.9	0.05200	0.00000	581.1	206.2	0.82646	0.00000
0.05300	0.03533	342.9	385.7	0.05300	0.00000	600.1	214.3	0.82463	0.00000
0.05400	0.03600	355.0	396.9	0.05400	0.00000	619.5	222.7	0.82281	0.00000
0.05500	0.03667	367.4	408.4	0.05500	0.00000	639.6	231.3	0.82099	0.00000
0.05600	0.03733	380.2	420.2	0.05600	0.00000	660.3	240.1	0.81917	0.00000
0.05700	0.03800	393.4	432.4	0.05700	0.00000	681.6	249.3	0.81735	0.00000
0.05800	0.03867	407.0	444.9	0.05800	0.00000	703.6	258.7	0.81553	0.00000
0.05900	0.03933	421.0	457.8	0.05900	0.00000	726.2	268.4	0.81371	0.00000
0.06000	0.04000	435.4	471.2	0.06000	0.00000	749.5	278.4	0.81190	0.00000
0.06100	0.04067	450.3	484.9	0.06100	0.00000	773.5	288.7	0.81009	0.00000
0.06200	0.04133	465.6	499.0	0.06200	0.00000	798.3	299.3	0.80828	0.00000
0.06300	0.04200	481.4	513.6	0.06300	0.00000	823.7	310.2	0.80647	0.00000
0.06400	0.04267	497.6	528.6	0.06400	-0.00000	850.0	321.4	0.80466	0.00000
0.06500	0.04333	514.3	544.0	0.06500	-0.00000	877.0	333.0	0.80286	0.00000
0.06600	0.04400	531.5	559.9	0.06600	-0.00000	904.8	344.9	0.80106	0.00000

0.06700	0.04467	549.3	576.2	0.06700	-0.00000	933.4	357.2	0.79926	0.00000
0.06800	0.04533	567.5	593.1	0.06800	-0.00000	962.9	369.8	0.79746	0.00000
0.06900	0.04600	586.3	610.4	0.06900	-0.00000	993.3	382.8	0.79566	0.00000
0.07000	0.04667	605.7	628.3	0.07000	-0.00000	1024.5	396.2	0.79386	0.00000
0.07100	0.04733	625.6	646.7	0.07100	-0.00000	1056.7	410.0	0.79207	0.00000
0.07200	0.04800	646.1	665.6	0.07200	-0.00000	1089.8	424.2	0.79028	0.00000
0.07300	0.04867	667.2	685.1	0.07300	-0.00000	1123.9	438.8	0.78849	0.00000
0.07400	0.04933	688.9	705.1	0.07400	-0.00000	1159.0	453.9	0.78670	0.00000
0.07500	0.05000	711.3	725.8	0.07500	-0.00000	1195.1	469.3	0.78491	0.00000
0.07600	0.05067	734.3	747.0	0.07600	-0.00000	1232.3	485.3	0.78313	0.00000
0.07700	0.05133	758.0	768.9	0.07700	-0.00000	1270.6	501.7	0.78134	0.00000
0.07800	0.05200	782.3	791.4	0.07800	-0.00000	1309.9	518.5	0.77956	0.00000
0.07900	0.05267	807.4	814.5	0.07900	-0.00000	1350.4	535.9	0.77778	0.00000
0.08000	0.05333	833.2	838.3	0.08000	-0.00000	1392.1	553.8	0.77600	0.00000
0.08100	0.05400	859.8	862.8	0.08100	-0.00000	1435.0	572.1	0.77423	0.00000
0.08200	0.05467	887.1	888.1	0.08200	-0.00000	1479.1	591.0	0.77245	0.00000
0.08300	0.05533	915.2	914.0	0.08300	-0.00000	1524.5	610.5	0.77068	0.00000
0.08400	0.05600	944.1	940.7	0.08400	-0.00000	1571.2	630.5	0.76891	0.00000
0.08500	0.05667	973.8	968.1	0.08500	-0.00000	1619.2	651.1	0.76714	0.00000
0.08600	0.05733	1004.4	996.4	0.08600	-0.00000	1668.6	672.3	0.76538	0.00000

2. Comparison of Iso and Oed Tests on Ohde Model



3. Comparison of Oed Tests between labTest and Ohde Model



4. Source code

(1) functions_elementTest

```
# functions for the element test programme

import numpy

# transformation of principal stresses into invariants
def sigma2pq(sigma_1,sigma_2):
    p = (sigma_1+2*sigma_2)/3
    q = sigma_1-sigma_2
    stresspq = numpy.array([p,q])
    return stresspq

# transformation of invariants into principal stresses
def pq2sigma(p,q):
    sigma_1 = p+2*q/3
    sigma_2 = p-q/3
    stress12 = numpy.array([sigma_1,sigma_2])
    return stress12

# transformation of principal strains into invariants
def eps2pq(eps_1,eps_2):
    eps_p = eps_1+2*eps_2
    eps_q = 2*(eps_1-eps_2)/3
    strainpq = numpy.array([eps_p,eps_q])
    return strainpq

# transformation of invariants into principal strains
def pq2eps(eps_p,eps_q):
    eps_1 = eps_p/3+eps_q
    eps_2 = eps_p/3-eps_q/2
    strain12 = numpy.array([eps_1,eps_2])
```

```
    return strain12

def testControl(M,testType):
    """
    Calculates the strain and strain increments for a given testType

    Parameters
    ----------
        M : 2x2 array of floats
            stiffness matrix

        testtype : string
            type of the element test, the options are
            'oed' for oedometric compression,
            'txd' for drained triaxial compression,
            'txu' for undrained triaxial compression

    Returns
    -------
        dstress : 2x1 array
            stress increment in volumetric and deviatoric invariants
            order: dp, dq

        dstrain : 2x1 array
            strain increment in volumetric and deviatoric invariants
            order: deps_p, deps_q
    """

    M_pp = M[0,0]; M_pq = M[0,1]
    M_qp = M[1,0]; M_qq = M[1,1]

    #strain increment according to test type
    if testType=='oed':
        deps_1 = 1
        deps_2 = 0
        deps_p, deps_q = eps2pq(deps_1,deps_2)
    elif testType=='txd':
        deps_q = 1
        deps_p = (M_qq-3*M_pq)*deps_q/(3*M_pp-M_qp)
    elif testType=='txu':
        deps_1 = 1
        deps_2 = -0.5
        deps_p, deps_q = eps2pq(deps_1,deps_2)
    elif testType == 'iso':
```

```
            deps_1 = 1
            deps_2 = 1
            deps_p, deps_q = eps2pq(deps_1,deps_2)
        else:
            return
        dstrain=numpy.array([deps_p,deps_q])
        # stress increment
        dstress=numpy.dot(M,dstrain)

        return dstress, dstrain

def integration(stress,strain,dstress,dstrain,stateVar,dStateVar,dt):
    """
    Explicit numerical integration of the state variables

    Parameters
    ----------
        stress, dstress : 2x1 array
            actual stress and stress increment in volumetric and
            deviatoric invariants (p, q)

        strain, dstrain : 2x1 array
            actual strain and strain increment in volumetric and
            deviatoric invariants (eps_p, eps_q)

        stateVar, dStateVar : 2x1 array
            actual state variables additional to stress and strain
            and their increments

        dt : float
            time step

    Returns
    -------
        stress : 2x1 array
            updated stress in volumetric and deviatoric invariants (p, q)

        strain : 2x1 array
            updated strain in volumetric and deviatoric invariants (eps_p, eps_q)

        stateVar : 2x1 array
            updated additional state variables
    """
```

```
    stress = stress+dstress*dt
    strain = strain+dstrain*dt
    deps_p, deps_q = dstrain
    stateVar[0] = stateVar[0]-deps_p*(1+stateVar[0])*dt
    stateVar[1] = stateVar[1]+dStateVar[1]*dt

    return stress, strain, stateVar

#================================
# output
#================================

# save state variables to a numpy array
def recordData(data,stress,strain,stateVar):
    """
    Records the state variables in a numpy array

    Parameters
    ----------
        data : Nx10 array of floats
            array for saving the state variables in the following order
            #eps_p #eps_q #p #q #eps_1 #eps_2 #sigma_1 #sigma_2 #epor #stateVar2

        stress : 2x1 array
            volumetric and deviatoric stress invariants
            order: p, q

        strain : 2x1 array
            volumetric and deviatoric strain invariants
            order: eps_p, eps_q

        stateVar : 2x1 array
            actual state variables, stateVar[0] is usually void ratio

    Returns
    -------
        a textfile with the output stored in the same order as written in the data array
    """

    p, q = stress
    eps_p, eps_q = strain
    sigma_1, sigma_2 = pq2sigma(p,q)
    eps_1, eps_2 = pq2eps(eps_p,eps_q)
    data = numpy.append(data,[[eps_p, eps_q, p, q, eps_1, eps_2, sigma_1, sigma_2, stateVar[0], stateVar[1] ]],
```

```
axis=0)

    return data

# output to a text file
def printResults(filename,data):
    """
    Records the state variables in a text file

    Parameters
    ----------
        filename : string
            Name of the outputfile

        data : Nx10 array of floats
            data as specified in the print_results() function
            strains, stresses in various invariants
            #eps_p #eps_q #p #q #eps_1 #eps_2 #sigma_1 #sigma_2 #epor #stateVar2

    Returns
    -------
        a textfile with the output stored in the same order as written in the data array
    """

    outputfile = open(filename+'.txt','w')
    outputfile.write('{:<8} {:<8} {:<8} {:<8} {:<8} {:<8} {:<8} {:<8} {:<8} {:<8}\n'.format(
        'eps_p', 'eps_q', 'p', 'q', 'eps_1', 'eps_2', 'sigma_1', 'sigma_2', 'epor', 'stateVar2'))
    for i in range(numpy.shape(data)[0]):
        outputfile.write('{:<8.5f} {:<8.5f} {:<8.1f} {:<8.1f} {:<8.5f} {:<8.5f} {:<8.1f} {:<8.1f} {:<8.5f}
{:<8.5f}\n'.format(
            data[i,0], data[i,1], data[i,2], data[i,3], data[i,4], data[i,5], data[i,6], data[i,7], data[i,8], data[i,9]))
    outputfile.close()

    return
```

(2) functions_matModels

```
# functions for the element test programme
# especially for the material models

import numpy, math
from functions_elementTest import *


def matModel(stress,strain,stateVar,model,modelParam):
```

```
"""
Calculates the stiffness matrix for a given material model
This function only deals with elastic models

Parameters
----------
    stress : 2x1 array
        stress in volumetric and deviatoric invariants (p, q)

    strain : 2x1 array
        strain in volumetric and deviatoric invariants (deps_p, deps_q)

    stateVar : 2x1 array
        actual state variables, stateVar[0] is usually void ratio

    model : string
        type of the material model
        'linelast' for a linear elastic model,
        'hyperb' for a strain dependent hyperbolic model

    modelParam : dictionary
        parameters for the material models
        such as: E, nu, G0, phi, psi, ...

Returns
-------
    M : 2x2 array of floats
        stiffness matrix
"""

p, q = stress
eps_p, eps_q = strain
# linear elasticity
if model=='linelast':
    E    = modelParam["E"]
    nu = modelParam["nu"]
    K = E/3/(1-2*nu)    # bulk modulus
    G = E/2/(1+nu)        # shear modulus
# non-linear elasticity (strain-dependent hyperbolic stiffness)
elif model=='hyperb':
    G0      = modelParam["G0"]
    nu      = modelParam["nu"]
    qmax = modelParam["qmax"]
    bhyp = 1/G0
```

```
            ahyp = 1/qmax
            G = 1/(bhyp+ahyp*eps_q)    #tangent stiffness
            E = 2*G*(1+nu)
            K = E/3/(1-2*nu)
        elif model=='mcc':
            kappa = modelParam["kappa"]
            nu    = modelParam["nu"]
            K = (1 + stateVar[0])*p/kappa
            G = 3*(1-2*nu)/(2*(1+nu))*K
        elif model=='ohde':
            alpha = modelParam["alpha"]
            nu = modelParam["nu"]
            E = ohde_Es(stress,3636.364,100,alpha)*(1+nu)*(1-2*nu)/(1-nu)
            K = E/3/(1-2*nu)    # bulk modulus
            G = E/2/(1+nu)      # shear modulus
        else:
            return
        M_pp, M_qq = K, 3*G
        M_pq, M_qp = 0, 0
        M = numpy.array([[M_pp,M_pq],[M_qp,M_qq]])
        return M

# define odemetric modulus
def ohde_Es(stress,Es0,sigma0,alpha):
        p,q = stress
        Es = Es0 * math.pow(pq2sigma(p,q)[0]/sigma0, alpha)
        return Es
```

(3) plotting_routines

```
import numpy as np
import matplotlib.pyplot as plt

def plotting_testData(data_iso, testType_iso, data_oed, testType_oed, data_lab, p_type, outputName=None):
        """
        plotting routine for the data produced by the element test script

        Parameters
        ----------
            data_iso : Nx9 array of floats
                data as specified in the print_results() function
                strains, stresses in various invariants for iso test

            testType_iso : string
                type of the element test that is simulated (iso in this case)
```

```
        data_oed : Nx9 array of floats
            data as specified in the print_results() function
            strains, stresses in various invariants for oed test

        testType_oed : string
            type of the element test that is simulated (oed in this case)

        outputName : string, optional
            name of the output file if the figure is to be saved
            default is None which will results in an immediate plot

    Returns
    -------
        either an output figure or an immediate plot
    """

    if p_type == "1":

        # create the figure and the subplots
        fig, axs = plt.subplots(2, 2, figsize=(12, 5))
        fig.subplots_adjust(hspace=0.4, wspace=0.4)

        # Plot for e-sigma_1
        axs[0, 0].plot(data_iso[:, 6], data_iso[:, 8], label=testType_iso)
        axs[0, 0].plot(data_oed[:, 6], data_oed[:, 8], label=testType_oed)
        axs[0, 0].set_xlabel('$\sigma_{1}$ [kPa]')
        axs[0, 0].set_ylabel('e [-]')
        axs[0, 0].grid(which='both')
        axs[0, 0].legend()

        # Plot for eps_1-sigma_1
        axs[0, 1].plot(data_iso[:, 4], data_iso[:, 6], label=testType_iso)
        axs[0, 1].plot(data_oed[:, 4], data_oed[:, 6], label=testType_oed)
        axs[0, 1].set_xlabel('eps_{1} [-]')
        axs[0, 1].set_ylabel('$\sigma_{1}$ [kPa]')
        axs[0, 1].grid(which='both')
        axs[0, 1].legend()

        # plot for sigma_1-sigma_2
        axs[1, 0].plot(data_iso[:, 7], data_iso[:, 6], label=testType_iso)
        axs[1, 0].plot(data_oed[:, 7], data_oed[:, 6], label=testType_oed)
        axs[1, 0].set_xlabel('$\sigma_{2}$ [kPa]')
        axs[1, 0].set_ylabel('$\sigma_{1}$ [kPa]')
```

```
        axs[1, 0].grid(which='both')
        axs[1, 0].legend()

        # plot for q-p
        axs[1, 1].plot(data_iso[:, 2], data_iso[:, 3], label=testType_iso)
        axs[1, 1].plot(data_oed[:, 2], data_oed[:, 3], label=testType_oed)
        axs[1, 1].set_xlabel('p [kPa]')
        axs[1, 1].set_ylabel('q [kPa]')
        axs[1, 1].grid(which='both')
        axs[1, 1].legend()

        fig.suptitle('Comparison of Iso and Oed Tests on Ohde Model')

    elif p_type == "2":
        # Convert data_lab to a NumPy array if it's not already
        data_lab = np.array(data_lab)

        # create the figure and the subplots
        fig, axs = plt.subplots(1, 2, figsize=(12, 5))
        fig.subplots_adjust(hspace=0.4, wspace=0.4)

        # Plot for e-sigma_1
        axs[0].plot(data_oed[:, 6], data_oed[:, 8], label=testType_oed)
        axs[0].scatter(data_lab[0, :], data_lab[2, :], label='labTest_oed', marker='o')
        axs[0].set_xlabel('$\sigma_{1}$ [kPa]')
        axs[0].set_ylabel('e [-]')
        axs[0].grid(which='both')
        axs[0].legend()

        # Plot for eps_1-sigma_1
        axs[1].plot(data_oed[:, 4], data_oed[:, 6], label=testType_oed)
        axs[1].scatter(data_lab[1, :], data_lab[0, :], label='labTest_oed', marker='o')
        axs[1].set_xlabel('eps_{1} [-]')
        axs[1].set_ylabel('$\sigma_{1}$ [kPa]')
        axs[1].grid(which='both')
        axs[1].legend()

        fig.suptitle('Comparison of Oed Tests between labTest and Ohde Model')

    else:
        print("No or wrong testType is specified. Aborting...")
        return

    # export figure
```

```
    if outputName is not None:
        plt.savefig(outputName + '.pdf', format='pdf', bbox_inches='tight')
    else:
        # show figure
        plt.show()
```

(4) taskB

```
"""
Task B: Extend the Python routines and run simulations
"""

import numpy
# import additional functions
from functions_elementTest import *
from functions_matModels import *
from plotting_routines import *

#================================
# Task B
#================================

# initial state
def ini_state(testType):
    if testType == 'oed':
        epor = 0.924    # void ratio
        sigma_1 = 100    # [kPa]
        sigma_2 = 0    # [kPa]
        eps_1 = 0    # [-]
        eps_2 = 0    # [-]
    elif testType == 'iso':
        epor = 0.924    # void ratio
        sigma_1 = 100    # [kPa]
        sigma_2 = 100    # [kPa]
        eps_1 = 0    # [-]
        eps_2 = 0    # [-]
    else:
        return

    return epor, sigma_1, sigma_2, eps_1, eps_2

# material models for the elastic part
# "linelast" ... linear elasticity
# "hyperb"      ... non-linear elasticity (strain-dependent hyperbolic stiffness)
model_el = 'ohde'
```

```
# define the parameters for the material models
alpha = 0.937          # soil parameter
nu = 0.3                # poisson ratio
# save them to a dictionary
modelParam={"alpha":alpha, "nu":nu}

#    stop criterion for calculation
p_max = 1000          # [kPa]
epsq_max = 0.2    # [-]
i_max = 1000

# numerical parameter
dt = 0.001                        # time step (numerical integration)

# Function to run the test
def run_test(testType):
    # inital stress and strain in volumetric and deviatoric invariants
    epor = ini_state(testType)[0]
    sigma_1 = ini_state(testType)[1]
    sigma_2 = ini_state(testType)[2]
    eps_1 = ini_state(testType)[3]
    eps_2 = ini_state(testType)[4]
    stress = sigma2pq(sigma_1, sigma_2)
    strain = eps2pq(eps_1, eps_2)

    # these arrays are only for later use with more complex constitutive models
    stateVar = numpy.array([epor, 0])
    dStateVar = numpy.array([0, 0])

    # record the state variables in a numpy array in the following order
    # column: eps_p, eps_q, p, q, eps_1, eps_2, sigma_1, sigma_2, epor
    # every row represents a time step
    data = numpy.array([[strain[0], strain[1], stress[0], stress[1], eps_1, eps_2, sigma_1, sigma_2, epor,
stateVar[1]]])

    # initialise the loop
    step = 1
    while (stress[0] < p_max) and (strain[1] < epsq_max) and (step < i_max):
        M = matModel(stress, strain, stateVar, model_el, modelParam)
        dstress, dstrain = testControl(M, testType)
        stress, strain, epor = integration(stress, strain, dstress, dstrain, stateVar, dStateVar, dt)
        data = recordData(data, stress, strain, stateVar)
        step += 1
```

```
        # open output file for saving the data and the plot
        filename = 'output_' + testType + '_' + model_el
        # save results to a text file
        printResults(filename, data)

        return data

# Run the tests
data_iso = run_test('iso')
data_oed = run_test('oed')
data_lab = [[100, 200, 300, 500, 600], [0.0275, 0.0475, 0.06, 0.075, 0.08], [0.872, 0.833, 0.81, 0.78, 0.77]]

# Plot the results
print("The plot type you want to realize:\n"
          "1:Comparison of Iso and Oed Tests on Ohde Model\n"
          "2:Comparison of Oed Tests between labTest and Ohde Model"
      )
p_type = input("which kind of plot you want to realize: ")
if p_type == "1":
      plotting_testData(data_iso, 'iso', data_oed, 'oed', data_lab, "1")
elif p_type == "2":
      plotting_testData(data_iso, 'iso', data_oed, 'oed', data_lab, "2")
```