Continuum Mechanics (BMEGEMMNWCM) II. Homework

Szász Zsolt
KRCH5Q

November 26, 2024

Homework 2

BMEGEMMNWCM, Continuum Mechanics

We use the 2nd-order incompressible Ogden's isotropic hyperelastic constitutive model to represent the mechanical behavior of an incompressible rubber-like material. The strain energy potential of the model is defined as

$$W = \sum_{k=1}^{2} \frac{2\mu_k}{\alpha_k^2}\left(\lambda_1^{\alpha_k} + \lambda_2^{\alpha_k} + \lambda_3^{\alpha_k} - 3\right).$$

The model contains four independent material parameters $\left(\mu_1, \mu_2, \alpha_1, \alpha_2\right)$. We have performed the following experimental tests on the material: Uniaxial extension, Equibiaxial extension, Planar Extension. The measured data are given in tabular form.

TASKS:

General: Determine the material parameters of the 2nd-order Ogden's model by performing a parameter-fitting task. Use the "*Root Mean Squared Relative Error*" with the engineering stress to define the quality function for all experiments. Use *constrained minimization* with constraint $\mu_1 + \mu_2 > 0$ in order to ensure positive ground-state Young's modulus.

• **Task 1.** Plot the experimental data points for each test in the same coordinate system. Use stretch for the horizontal axis and the nominal stress for the vertical axis. The plot range for stretch must be $\lambda = 1 \ldots 4$. The plot range for the nominal stress must be $P = 0 \ldots P_{\max}$, where $P_{\max}$ is the maximum measured nominal stress in the equibiaxial test. You can connect the data point with lines, but use dots to indicate the particular data points.

• **Task 2.** Obtain the material parameters by fitting the model only to the uniaxial data. Consequently, in the calculation of the quality function use only the uniaxial data. The parameter-fitting is acceptable if $Q_U < 5$ %. Report the values of the fitted material parameters and Q_U.

• **Task 3.** Compute the quality functions (numerical values) of the fitted model (in Task 2) for the equibiaxial and the planar extensions. Report the values!

• **Task 4.** Visualize the model accuracy by plotting the model solutions for the stresses in each test using the fitted material parameters. Use separate figures for the tests. Use the plot range defined in Task 1. Write a few sentences (your personal conclusions and observation) about the accuracy of the fitted model.

Task 5. Obtain the material parameters by fitting the model simultaneously to all measurements. Consequently, the quality function is the mean of the individual quality values: $Q = (Q_U + Q_B + Q_P)/3$. The parameter-fitting is acceptable if $Q < 5$ %. Report the values of the fitted parameters and Q, Q_U, Q_B, Q_P.

• **Task 6.** Visualize the model accuracy by plotting the model solutions for the stresses in each test using the fitted material parameters in Task 5. Use separate figures for the tests. Use the plot range defined in Task 1. Write a few sentences (your personal conclusions and observation) about the accuracy of the fitted model.

• **Task 7.** The homogeneous displacement field in the material is given with the components of the material displacement field. The displacement components are given in [mm] if the coordinates are substituted in [mm]. The material is in a plane stress state, i.e. $\sigma_z = \tau_{xz} = \tau_{yz} = 0$. Determine the unknown scalar k included in the displacement field using the incompressibility constraint!

• **Task 8.** Determine the principal stretches and the normalized principal Eulerian directions.

• **Task 9.** Compute the components of the matrix of the Cauchy stress tensor using the fitted material parameters in Task 5. Determine the Mises equivalent stress.

Your data can be downloaded from the following link:

https://www.mm.bme.hu/~kossa/NEPTUN-HW2.pdf

The string **NEPTUN** has to be replaced with the Student's NEPTUN code. The link above is case-sensitive! Use uppercase letters for the file-name! Students must solve the problem corresponding to their NEPTUN's code!

Contents

Data

Experimental data:

Eng. Strain %	Eng. Stress (Uniaxial) MPa	Eng. Stress (Equibiaxial) MPa	Eng. Stress (Planar) MPa
0	0	0	0
30	6	10	7
60	8	17	10
90	9	22	12
120	11	28	14
150	12	37	16
180	14	44	17
210	15	52	19
240	16	60	21
270	17	70	23
300	19	82	24

Deformation:

$$\boldsymbol{U} = \begin{bmatrix} 5 - 3kX \\ 1 + 0.3X + 0.5Y \\ 3 \end{bmatrix}$$

1 Task: Plotting experimental data

Using the provided data, we can plot the experiments:

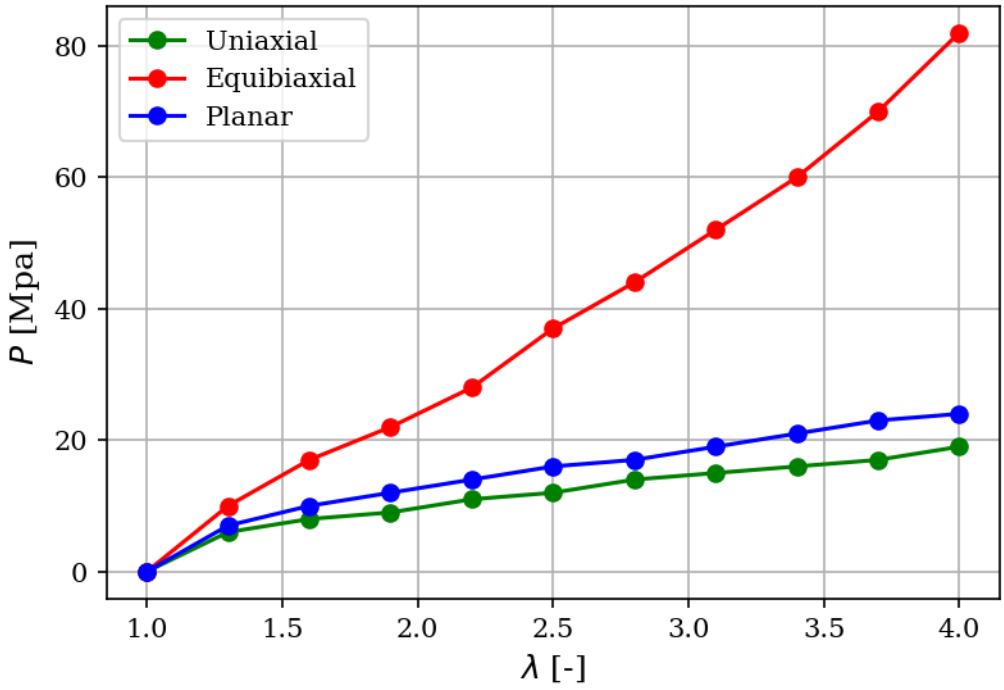


Figure 1: Experimental data

2 Task: Fitting model only to the unaxial data

At the lecture, we saw the solution of the general Ogden's model for the uniaxial tension, which in our 2nd order case looks like

$$P_{OG}^{U} = \sum_{k=1}^{2} \frac{2\mu_k}{\alpha_k}(\lambda^{\alpha_k - 1} - \lambda^{-\frac{\alpha_k}{2} - 1}).$$

Based on the instruction we can define a quality function calculated from the given $n = 11$ measured point, which looks like

$$Q_U = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{P_i^U - P_{OG}^U(\lambda_i)}{P_i^U}\right)^2}.$$

Choosing this quality function as the objective function of an optimization algorithm, we can fit the model parameters to the provided data points. I will use the `scipy.optimize.minimize` function, with BFGS (Broyden-Fletcher-Goldfarb-Shanno) method. I will also apply the given $\mu_1 + \mu_2 > 0$ constraint. The optimialization results the following parameters:

Table 1: Parameters of model fitted only on uniaxial data

μ_1 [MPa]	μ_2 [MPa]	α_1 [-]	α_2 [-]
10.71992575	0.04956515	−3.2772214	4.57758998

The quality functions value is

$$Q_U = 3.2700839\ \% < 5\ \%$$

3 Task: Equibiaxial and planar quality functions

The model response for the equibiaxial and planar loading was also shown at the lecture as

$$P_{OG}^{E} = \sum_{k=1}^{2} \frac{2\mu_k}{\alpha_k}(\lambda^{\alpha_k - 1} - \lambda^{-2\alpha_k - 1}) \quad \& \quad P_{OG}^{P} = \sum_{k=1}^{2} \frac{2\mu_k}{\alpha_k}(\lambda^{\alpha_k - 1} - \lambda^{-\alpha_k - 1}).$$

With them we can define the quality functions regarding to equibiaxial and planar experiments too, like previously

$$Q_E = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{P_i^E - P_{OG}^E(\lambda_i)}{P_i^E}\right)^2} \quad \& \quad Q_P = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{P_i^P - P_{OG}^P(\lambda_i)}{P_i^P}\right)^2}.$$

Which can be evaluated with the model fitted to the uniaxial data

$$Q_E = 7703.9478\ \%,$$
$$Q_P = 306.5815\ \%.$$

We can observe the overfitting of the model to the uniaxial data points.

4 Task: Plotting solutions

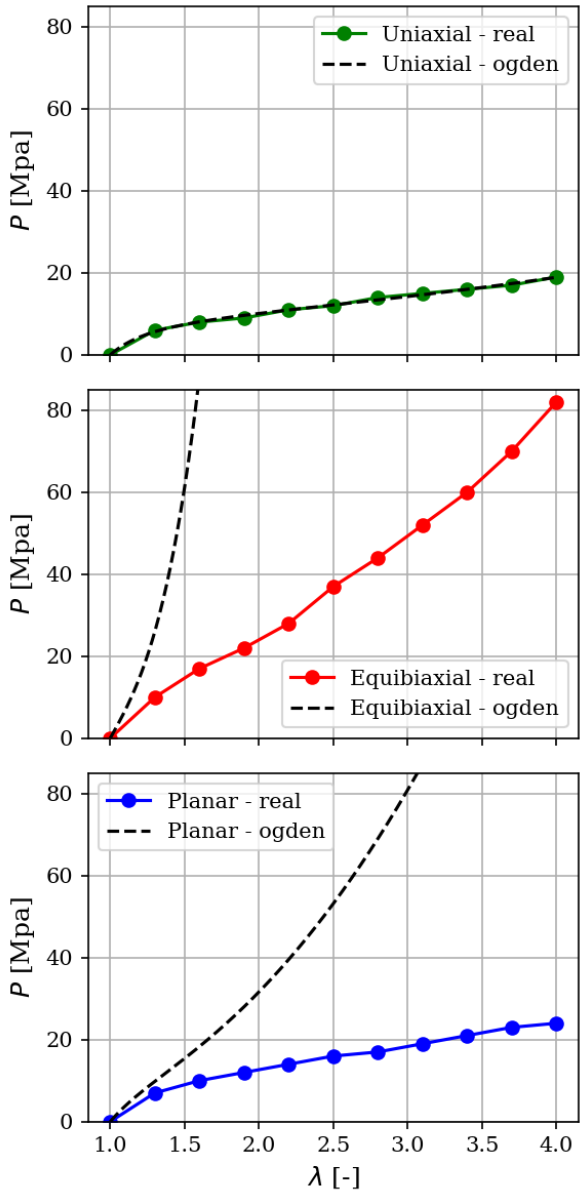


Figure 2: Uniaxially overfitted model's response for uniaxial, equibiaxial and planar loads

We can observe how the model overfits to the uniaxial measurement data. The reason is that, we didn't take into account the error of the equibiaxial and planar quality functions during the optimization procedure.

5 Task: Simultaneous fitting to all measurements

To take into account the equibiaxial and planar measurements too, we will use the suggested

$$Q = \frac{Q_U + Q_B + Q_P}{3}$$

objective function during the optimization. The procedure itself is tha same as the one used in the 2nd task. The resulting parametrs are

Table 2: Parameters of model fitted simultaneously to all measurements

μ_1 [MPa]	μ_2 [MPa]	α_1 [-]	α_2 [-]
3.18596983	4.62213918	−1.48228961	1.77455216

The values of the objective and separate three quality functions looks like

$$Q = 4.204376\ \% < 5\ \% \qquad Q_U = 6.414038\ \%$$
$$Q_E = 2.196516\ \% \qquad Q_P = 4.002574\ \%$$

6 Task: Plotting solutions

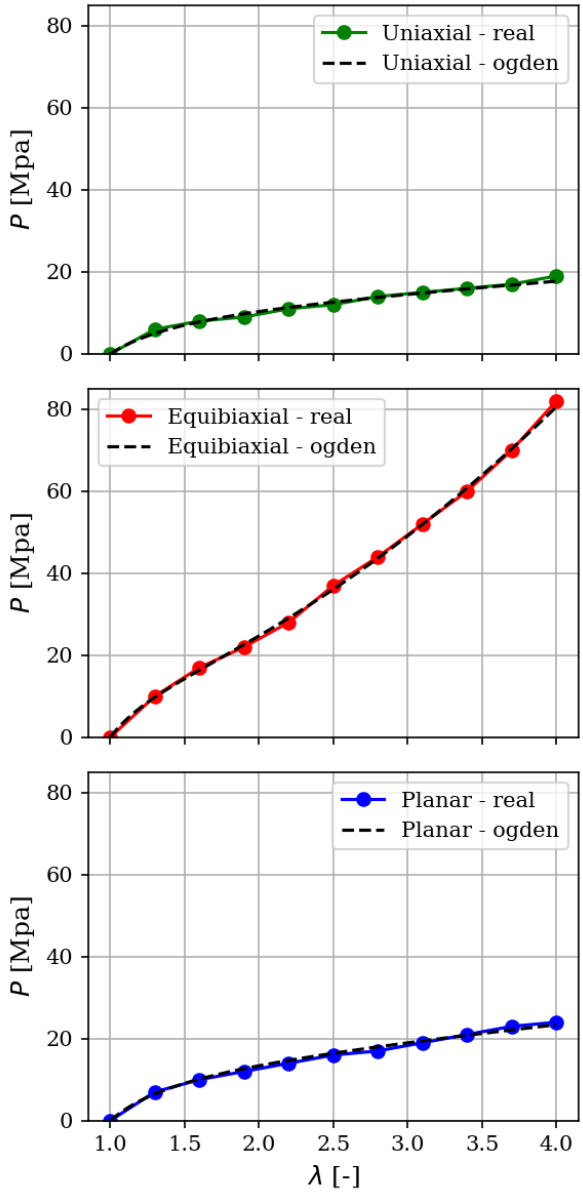


Figure 3: Simultaneously fitted model's response for uniaxial, equibiaxial and planar loads

We can see that taking into account all of the measurement series in the objective, provide us a model which fit into all of the measuremet datas.

7 Task: Determination of k

The displacement field is given in the with the components of the material displacement field, like

$$\boldsymbol{U} = \begin{bmatrix} 5 - 3kX_1 \\ 1 + 0.3X_1 + 0.5X_2 \\ 3 \end{bmatrix}.$$

From here we can determine the displacement gradient as

$$\boldsymbol{K} = \frac{\mathrm{d}\boldsymbol{U}}{\mathrm{d}\boldsymbol{X}} = \begin{bmatrix} -3k & 0 & 0 \\ 0.3 & 0.5 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

The deformation gradient can be calculated as

$$\boldsymbol{F} = \boldsymbol{K} + \boldsymbol{I} = \begin{bmatrix} 1 - 3k & 0 & 0 \\ 0.3 & 1.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

The volume change can be calculated as the determinant of the deformation gradient

$$J = \det\boldsymbol{F} = (1 - 3k) \cdot 1.5$$

.
If there is incompressible case, the volume change should be equal with 1, therefore:

$$(1 - 3k) \cdot 1.5 = 1,$$
$$k = \frac{1}{9}.$$

8 Task: Principal streches

We know that by the deformation gradient we can determine the left Cauchy-Green strain tensor as

$$\boldsymbol{b} = \boldsymbol{F}\boldsymbol{F}^T = \begin{bmatrix} 0.444 & 0.2 & 0 \\ 0.2 & 2.34 & 0 \\ 0 & 0 & 1.0 \end{bmatrix}.$$

The eigendecomposition of the Cauchy-Green tensor looks like

$$\boldsymbol{b} = \sum_{i=1}^{3} \mu_i \cdot \boldsymbol{n_i} \otimes \boldsymbol{n_i} = \sum_{i=1}^{3} \lambda_i^2 \cdot \boldsymbol{n_i} \otimes \boldsymbol{n_i},$$

where λ_i are the principal streches and $\boldsymbol{n_i}$ are the principal Eulerian directions. They numerically looks like

$$\begin{array}{ll} \lambda_1 = 0.65082 & \boldsymbol{n_1} = \begin{bmatrix} -0.994599 & -0.103797 & 0 \end{bmatrix}^T \\ \lambda_2 = 1.53651 & \boldsymbol{n_2} = \begin{bmatrix} 0.103797 & -0.994599 & 0 \end{bmatrix}^T \\ \lambda_3 = 1 & \boldsymbol{n_3} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T \end{array}.$$

9 Task: Cauchy stress tensor and Mises Equivalent stress

When we're talking about plane stress problems, the Cauchy stress tensors reduces to

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_x & \tau_{xy} & 0 \\ \tau_{xy} & \sigma_y & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

First we should mention the decomposition of the stress tensor into deviatoric and spherical part, which looks like

$$\boldsymbol{\sigma} = \boldsymbol{s} + \boldsymbol{p}$$

because, we know that in hyperelasticity the deformation only determine the deviatoric part of the stress tensor, which looks like

$$\boldsymbol{s} = \mathrm{dev}\left(\sum_{i=1}^{3} \sigma_i \cdot \boldsymbol{n_i} \otimes \boldsymbol{n_i}\right) = \sum_{i=1}^{3} \sigma_i \cdot \boldsymbol{n_i} \otimes \boldsymbol{n_i} - \frac{1}{3}\mathrm{tr}\left(\sum_{i=1}^{3} \sigma_i \cdot \boldsymbol{n_i} \otimes \boldsymbol{n_i}\right),$$

where the principal stress components can be calculated from the principal streches as

$$\sigma_i = \frac{1}{J}\lambda_i \frac{\partial W}{\partial \lambda_i} \overset{\mathrm{J=1}}{=} \lambda_i \frac{\partial W}{\partial \lambda_i}.$$

The second term of the product can be expressed from the provided strain energy potential function as

$$\frac{\partial W}{\partial \lambda_i} = \sum_{k=1}^{2} \frac{2\mu_k}{\alpha_k^2} \alpha_k \lambda_i^{\alpha_k - 1} = \sum_{k=1}^{2} \frac{2\mu_k}{\alpha_k} \lambda_i^{\alpha_k - 1}$$

Now we can calculate $\boldsymbol{s}$ with the previously fitted model parameters.

$$\boldsymbol{s} = \begin{bmatrix} -6.9058 & -1.5056 & 0 \\ -1.5056 & 7.3637 & 0 \\ 0 & 0 & -0.45787 \end{bmatrix}$$

The spherical part of the stress tensor can be determined from the plane stress boundary condition, Which means us that

$$p + s_{33} = \sigma_z = 0 \quad \Rightarrow \quad p = -s_{33} = 0.45787$$

Therefore, we can provide the spherical stress and the Cauchy stress tensors too, like

$$\boldsymbol{p} = \begin{bmatrix} 0.45787 & 0 & 0 \\ 0 & 0.45787 & 0 \\ 0 & 0 & 0.45787 \end{bmatrix} \quad \& \quad \boldsymbol{\sigma} = \begin{bmatrix} -6.4479 & -1.5056 & 0 \\ -1.5056 & 7.8215 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

Using the deviatoric stress tensor, we can define and calculate the von Mises equivalent stress, like

$$\sigma_{vM} = \sqrt{\frac{3}{2}\boldsymbol{s} : \boldsymbol{s}} = 12.648525 \text{ [MPa]}$$

Defines

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import rcParams
from scipy.optimize import minimize
import sympy as sp
rcParams['font.family'] = 'serif'

# Create the DataFrame
data = {
    "Eng. Strain [%]": [0, 30, 60, 90, 120, 150, 180, 210, 240, 270, 300],
    "Eng. Stress [MPa] (Uniaxial)": [0, 6, 8, 9, 11, 12, 14, 15, 16, 17, 19],
    "Eng. Stress [MPa] (Equibiaxial)": [0, 10, 17, 22, 28, 37, 44, 52, 60, 70, 82],
    "Eng. Stress [MPa] (Planar)": [0, 7, 10, 12, 14, 16, 17, 19, 21, 23, 24],
}

df = pd.DataFrame(data)
df
```

Out[1]:

	Eng. Strain [%]	Eng. Stress [MPa] (Uniaxial)	Eng. Stress [MPa] (Equibiaxial)	Eng. Stress [MPa] (Planar)
0	0	0	0	0
1	30	6	10	7
2	60	8	17	10
3	90	9	22	12
4	120	11	28	14
5	150	12	37	16
6	180	14	44	17
7	210	15	52	19
8	240	16	60	21
9	270	17	70	23
10	300	19	82	24

In [2]:
```
print(df.to_latex(index=False))
```

```
\begin{tabular}{rrrr}
\toprule
Eng. Strain [%] & Eng. Stress [MPa] (Uniaxial) & Eng. Stress [MPa] (Equibiaxial) & Eng. Stress
[MPa] (Planar) \\
\midrule
0 & 0 & 0 & 0 \\
30 & 6 & 10 & 7 \\
60 & 8 & 17 & 10 \\
90 & 9 & 22 & 12 \\
120 & 11 & 28 & 14 \\
150 & 12 & 37 & 16 \\
180 & 14 & 44 & 17 \\
210 & 15 & 52 & 19 \\
240 & 16 & 60 & 21 \\
270 & 17 & 70 & 23 \\
300 & 19 & 82 & 24 \\
\bottomrule
\end{tabular}
```

In [3]:
```python
def W(λ1, λ2, λ3, mu1, mu2, a1, a2):
    ret  = 2*mu1/a1**2 * (λ1**a1 + λ2**a1 + λ3**a1 - 3)
    ret += 2*mu2/a2**2 * (λ1**a2 + λ2**a2 + λ3**a2 - 3)
    return ret

def P_OG_uniaxial(λ, mu1, mu2, a1, a2):
    ret  = 2*mu1/a1 * (λ**(a1-1) - λ**(-a1/2 - 1))
    ret += 2*mu2/a2 * (λ**(a2-1) - λ**(-a2/2 - 1))
    return ret

def P_OG_equibiaxial(λ, mu1, mu2, a1, a2):
    ret  = 2*mu1/a1 * (λ**(a1-1) - λ**(-2*a1 - 1))
    ret += 2*mu2/a2 * (λ**(a2-1) - λ**(-2*a2 - 1))
    return ret

def P_OG_planar(λ, mu1, mu2, a1, a2):
    ret  = 2*mu1/a1 * (λ**(a1-1) - λ**(-a1 - 1))
    ret += 2*mu2/a2 * (λ**(a2-1) - λ**(-a2 - 1))
    return ret
```

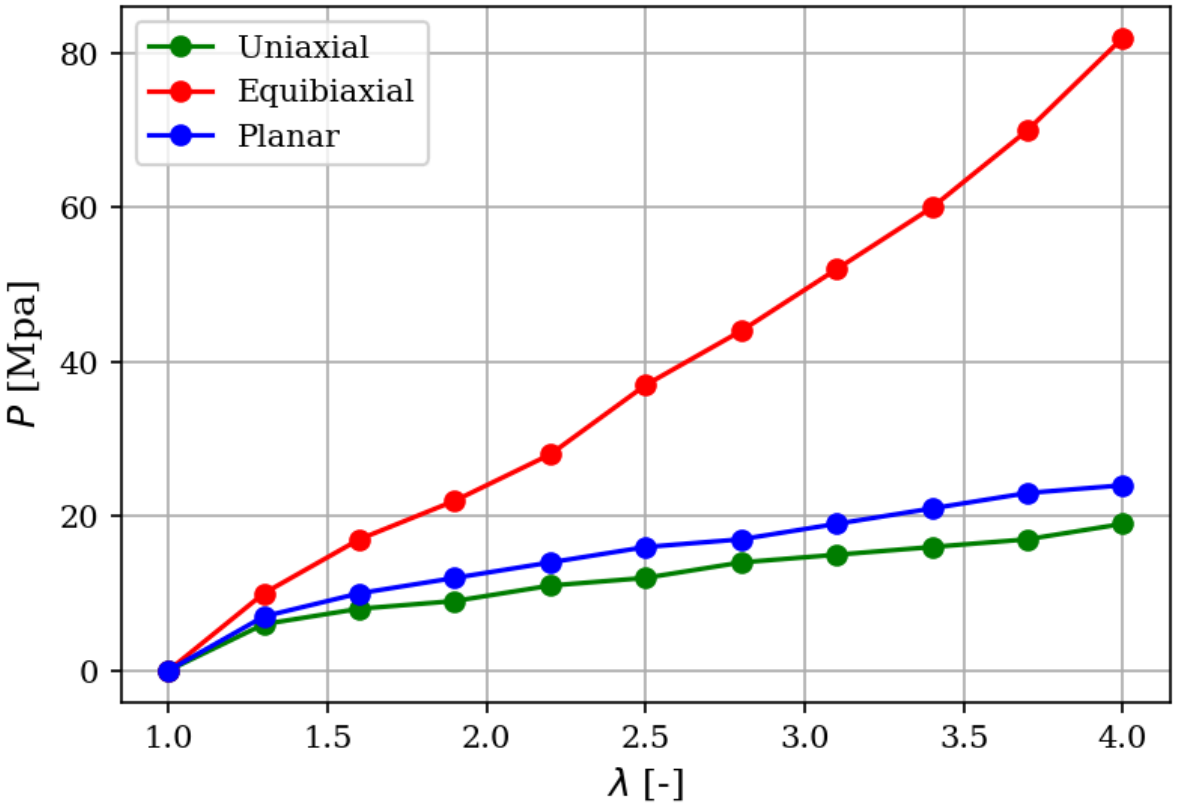
Tasks

Task 1. - Plotting experimental data

In [4]:
```python
ε = df["Eng. Strain [%]"].to_numpy()/100
λ = ε + 1

P_uniaxial    = df["Eng. Stress [MPa] (Uniaxial)"   ].to_numpy()
P_equibiaxial = df["Eng. Stress [MPa] (Equibiaxial)"].to_numpy()
P_planar      = df["Eng. Stress [MPa] (Planar)"     ].to_numpy()


plt.figure(figsize=(6,4),dpi=150)
plt.plot(λ,P_uniaxial,marker='o',label='Uniaxial',color='green')
plt.plot(λ,P_equibiaxial,marker='o',label='Equibiaxial',color='red')
plt.plot(λ,P_planar,marker='o',label='Planar',color='blue')
plt.grid()
plt.xlabel(r'$\lambda$ [-]',fontsize=12)
```

```
plt.ylabel(r'$P$ [Mpa]',fontsize=12)
plt.legend()
```

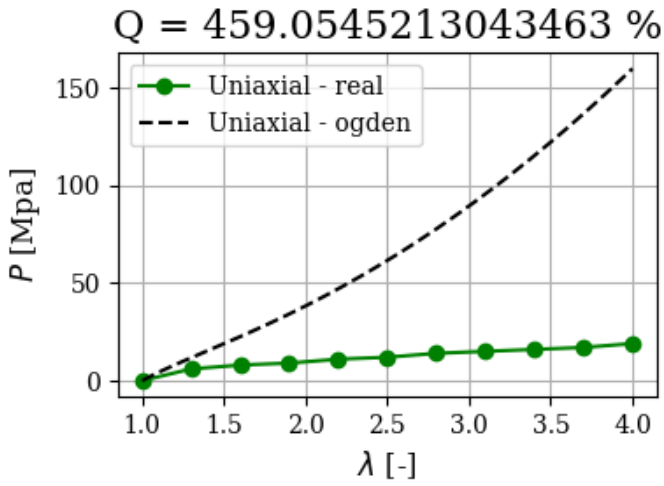
Out[4]: <matplotlib.legend.Legend at 0x1e8eceaab50>



Task 2. - Fitting model only to the unaxial data

In [5]:
```
def Q_uniaxial(params):
    mu1, mu2, a1, a2 = params
    # [:1] to avoid 0 divide
    # the model provide P=0 for λ=0
    pred = P_OG_uniaxial(λ[1:],mu1, mu2, a1, a2)
    real = P_uniaxial[1:]
    n = P_uniaxial.shape[0]

    return (np.sum(((real - pred)/real)**2)/n)**(1/2)
```

In [6]:
```
params = 10, 5, 3, 3

plt.figure(figsize=(4,3),dpi=100)
plt.plot(λ,P_uniaxial,marker='o',label='Uniaxial - real',color='green')
λ_range = np.linspace(1,4,500)
plt.plot(λ_range,P_OG_uniaxial(λ_range,*params),color='black',label='Uniaxial - ogden',ls='--'
plt.grid()
plt.xlabel(r'$\lambda$ [-]',fontsize=12)
plt.ylabel(r'$P$ [Mpa]',fontsize=12)
plt.legend()
plt.title(f'Q = {Q_uniaxial(params)*100} %',fontsize=16)
plt.tight_layout()
plt.show()
```

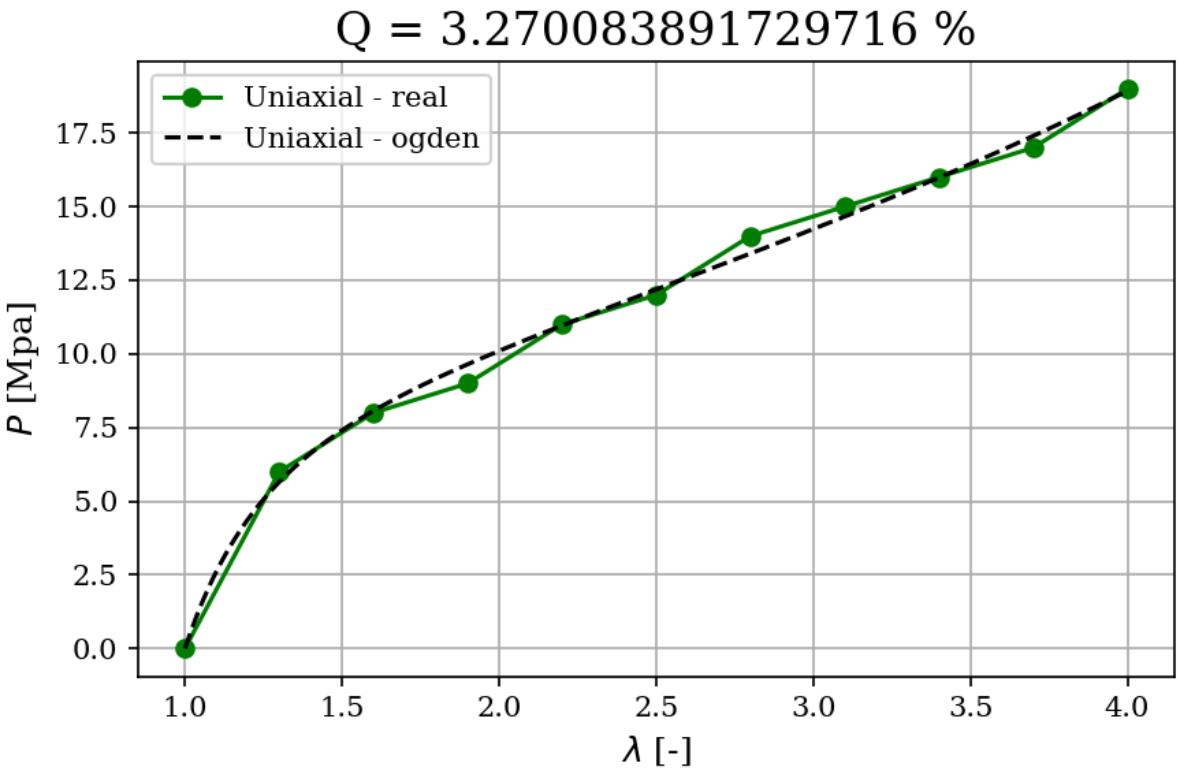


In [7]:
```
def constraint(params):
    mu1, mu2, a1, a2 = params
    return mu1 + mu2 - 1

constraints = [
    {'type': 'ineq', 'fun': constraint},
]

result = minimize(
    Q_uniaxial,
    params,
    method='BFGS',
    constraints=constraints)

print(f'Q_uniaxial = {result.fun*100} [%]')
print(f'params = {result.x}')
```

```
Q_uniaxial = 3.270083891729716 [%]
params = [10.71992575  0.04956515 -3.2772214   4.57758998]
```

```
C:\Users\Zsoci\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\scipy\optimize\_minimize.py:576: RuntimeWarning: Method BFGS cannot handle constraints.
  warn('Method %s cannot handle constraints.' % method,
```

In [8]:
```
params = result.x

plt.figure(figsize=(6,4),dpi=150)
plt.plot(λ,P_uniaxial,marker='o',label='Uniaxial - real',color='green')
λ_range = np.linspace(1,4,500)
plt.plot(λ_range,P_OG_uniaxial(λ_range,*params),color='black',label='Uniaxial - ogden',ls='--'
plt.grid()
plt.xlabel(r'$\lambda$ [-]',fontsize=12)
plt.ylabel(r'$P$ [Mpa]',fontsize=12)
plt.legend()
plt.title(f'Q = {Q_uniaxial(params)*100} %',fontsize=16)
plt.tight_layout()
plt.show()
```



Task 3. - Equibiaxial and planar quality functions

In [9]:
```
def Q_equibiaxial(params):
    mu1, mu2, a1, a2 = params
    # [:1] to avoid 0 divide
    # the model provide P=0 for λ=0
    pred = P_OG_equibiaxial(λ[1:],mu1, mu2, a1, a2)
    real = P_equibiaxial[1:]
    n = P_equibiaxial.shape[0]

    return (np.sum(((real - pred)/real)**2)/n)**(1/2)

def Q_planar(params):
    mu1, mu2, a1, a2 = params
    # [:1] to avoid 0 divide
    # the model provide P=0 for λ=0
    pred = P_OG_planar(λ[1:],mu1, mu2, a1, a2)
    real = P_planar[1:]
    n = P_planar.shape[0]

    return (np.sum(((real - pred)/real)**2)/n)**(1/2)
```

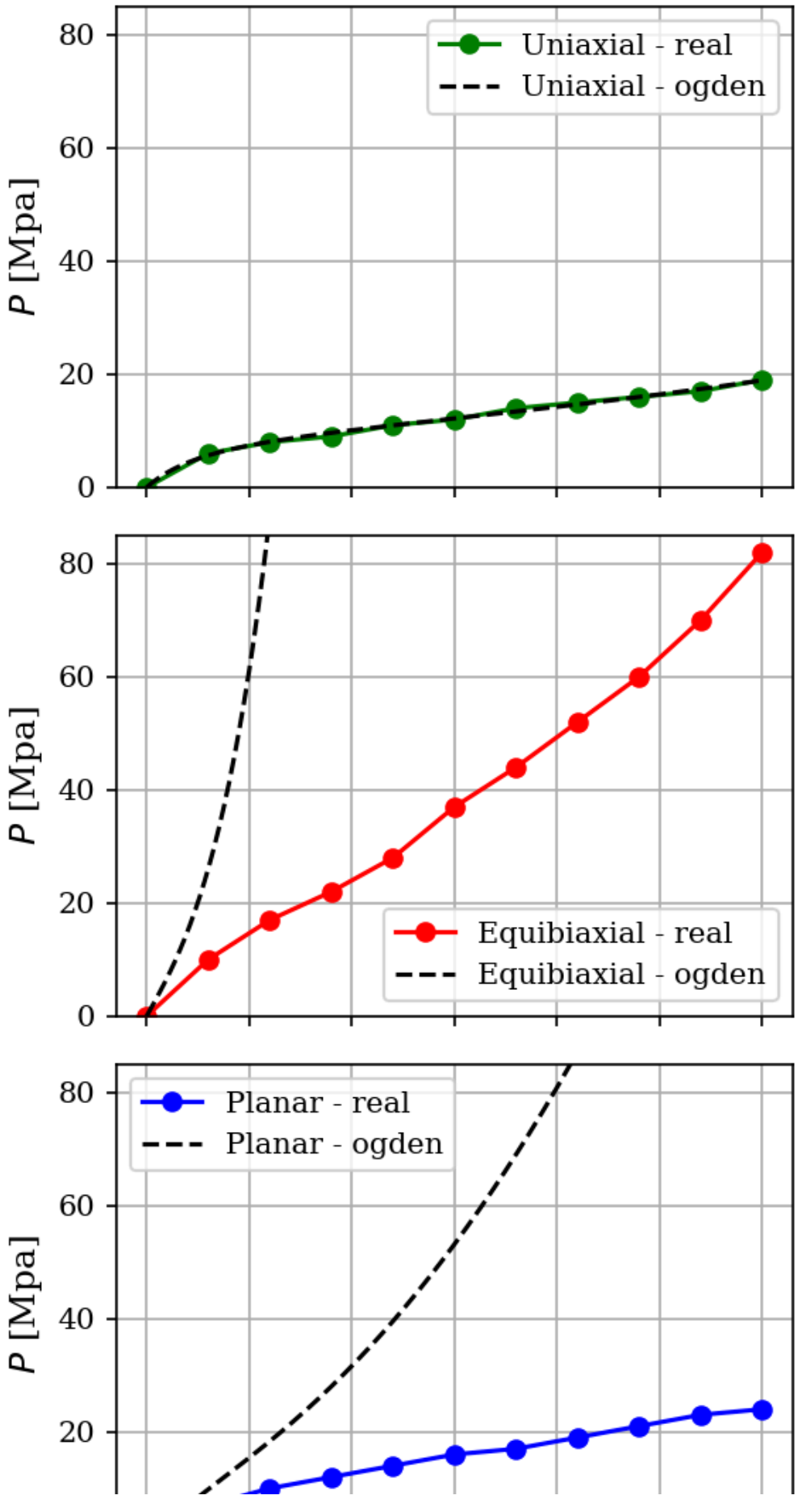
In [10]:
```
print(f'Q_equibiaxial = {Q_equibiaxial(params)*100} %')
print(f'Q_planar = {Q_planar(params)*100} %')
```

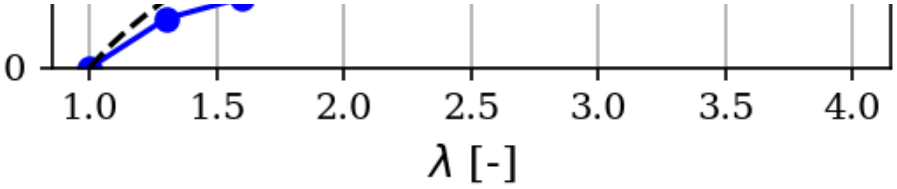
```
Q_equibiaxial = 7703.947754246242 %
Q_planar = 306.5815214494574 %
```

Task 4. - Plotting solutions

In [11]:
```
fig, ax = plt.subplots(3,figsize=(4,8),dpi=150,sharex=True,sharey=True)
```

```
λ_range = np.linspace(1,4,500)
ax[0].plot(λ,P_uniaxial,marker='o',label='Uniaxial - real',color='green')
ax[0].plot(λ_range,P_OG_uniaxial(λ_range,*params),color='black',label='Uniaxial - ogden',ls='-
ax[0].grid()
ax[0].set_ylim(0,85)
ax[0].legend()
ax[0].set_ylabel(r'$P$ [Mpa]',fontsize=12)
ax[1].plot(λ,P_equibiaxial,marker='o',label='Equibiaxial - real',color='red')
ax[1].plot(λ_range,P_OG_equibiaxial(λ_range,*params),color='black',label='Equibiaxial - ogden'
ax[1].grid()
ax[1].set_ylim(0,85)
ax[1].legend()
ax[1].set_ylabel(r'$P$ [Mpa]',fontsize=12)
ax[2].plot(λ,P_planar,marker='o',label='Planar - real',color='blue')
ax[2].plot(λ_range,P_OG_planar(λ_range,*params),color='black',label='Planar - ogden',ls='--')
ax[2].grid()
ax[2].set_ylim(0,85)
ax[2].legend()
ax[2].set_ylabel(r'$P$ [Mpa]',fontsize=12)
ax[2].set_xlabel(r'$\lambda$ [-]',fontsize=12)
plt.tight_layout()
plt.show()
```


Uniaxial - real
Uniaxial - ogden
P [Mpa]
0
20
40
60
80
Equibiaxial - real
Equibiaxial - ogden
P [Mpa]
0
20
40
60
80
Planar - real
Planar - ogden
P [Mpa]
20
40
60
80



Task 5. - Simultanious fitting to all measurements

In [12]:
```
def Q(params):
    return (Q_uniaxial(params) + Q_equibiaxial(params) + Q_planar(params))/3
```

In [13]:
```
def constraint(params):
    mu1, mu2, a1, a2 = params
    return mu1 + mu2 - 1

constraints = [
    {'type': 'ineq', 'fun': constraint},
]

result = minimize(
    Q,
    params,
    method='BFGS',
    constraints=constraints,
)

print(f'Q = {result.fun*100} [%]')
params = result.x
print(f'params = {params}')
```

```
Q = 4.204375992236135 [%]
params = [ 3.18596983  4.62213918 -1.48228961  1.77455216]
```

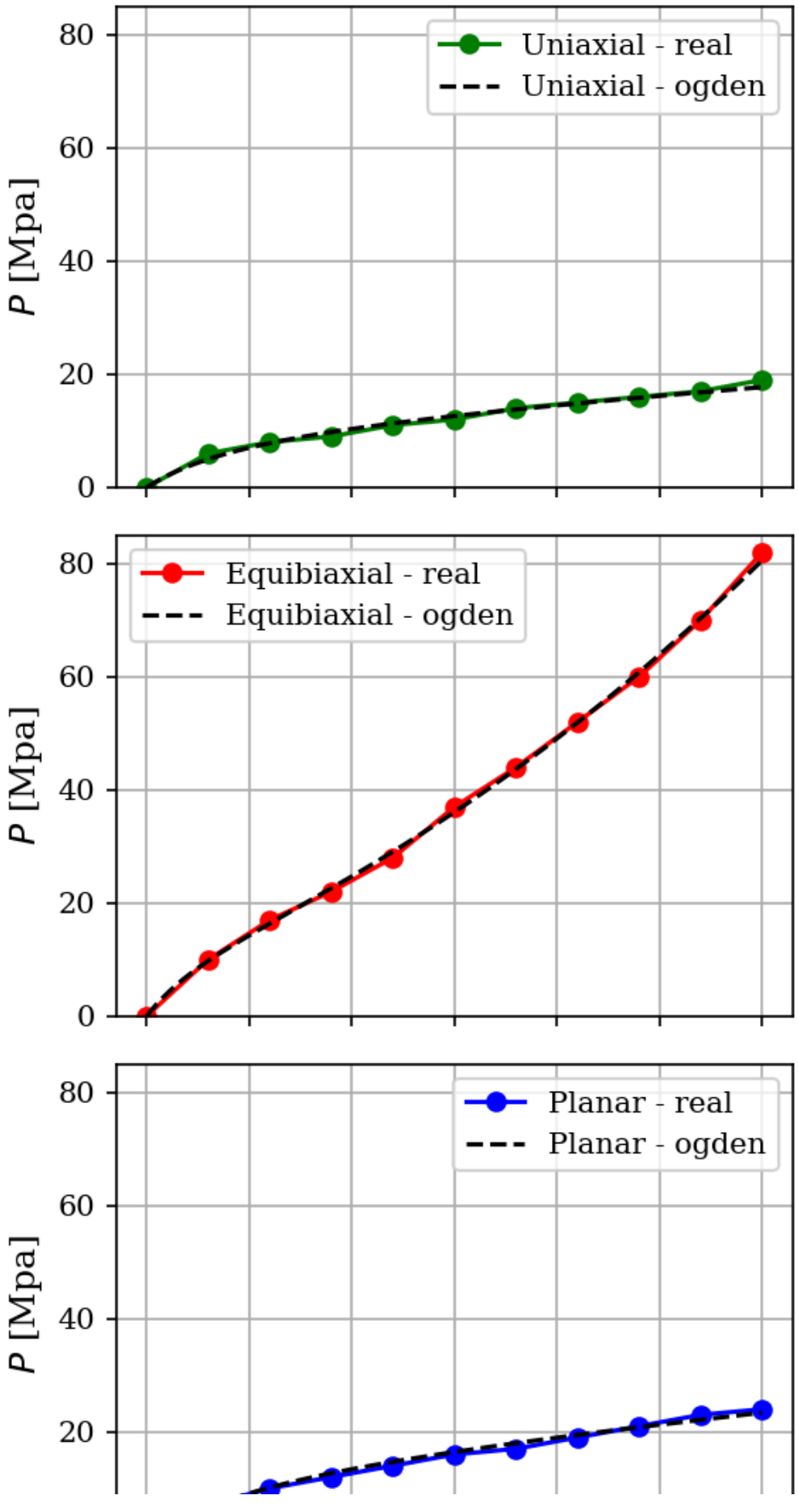
In [14]:
```
print(Q_uniaxial(params)*100,'[%]')
print(Q_equibiaxial(params)*100,'[%]')
print(Q_planar(params)*100,'[%]')
```

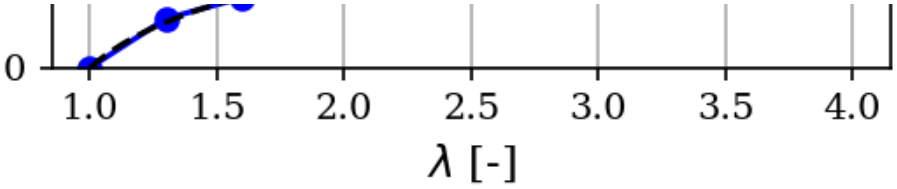
```
6.414037738579216 [%]
2.196516422902834 [%]
4.002573815226355 [%]
```

Task 6. - Plotting solutions

In [15]:
```
fig, ax = plt.subplots(3,figsize=(4,8),dpi=150,sharex=True,sharey=True)
λ_range = np.linspace(1,4,500)
ax[0].plot(λ,P_uniaxial,marker='o',label='Uniaxial - real',color='green')
ax[0].plot(λ_range,P_OG_uniaxial(λ_range,*params),color='black',label='Uniaxial - ogden',ls='-
ax[0].grid()
ax[0].set_ylim(0,85)
ax[0].legend()
ax[0].set_ylabel(r'$P$ [Mpa]',fontsize=12)
ax[1].plot(λ,P_equibiaxial,marker='o',label='Equibiaxial - real',color='red')
ax[1].plot(λ_range,P_OG_equibiaxial(λ_range,*params),color='black',label='Equibiaxial - ogden'
ax[1].grid()
ax[1].set_ylim(0,85)
ax[1].legend()
ax[1].set_ylabel(r'$P$ [Mpa]',fontsize=12)
ax[2].plot(λ,P_planar,marker='o',label='Planar - real',color='blue')
ax[2].plot(λ_range,P_OG_planar(λ_range,*params),color='black',label='Planar - ogden',ls='--')
```

```
ax[2].grid()
ax[2].set_ylim(0,85)
ax[2].legend()
ax[2].set_ylabel(r'$P$ [Mpa]',fontsize=12)
ax[2].set_xlabel(r'$\lambda$ [-]',fontsize=12)
plt.tight_layout()
plt.show()
```


Uniaxial - real
Uniaxial - ogden
P [Mpa]
0
20
40
60
80
Equibiaxial - real
Equibiaxial - ogden
P [Mpa]
0
20
40
60
80
Planar - real
Planar - ogden
P [Mpa]
20
40
60
80



Task 7. - Determine of k

The displacement field is given in the with the components of the material displacement field, like

$$\boldsymbol{U} = \begin{bmatrix} 5 - 3kX_1 \\ 1 + 0.3X_1 + 0.5X_2 \\ 3 \end{bmatrix}$$

.

From this we can determine the displacement gradient as

$$\boldsymbol{K} = \frac{\mathrm{d}\boldsymbol{U}}{\mathrm{d}\boldsymbol{X}} = \begin{bmatrix} -3k & 0 & 0 \\ 0.3 & 0.5 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

.

The deformation gradient can be calculated as

$$\boldsymbol{F} = \boldsymbol{K} + \boldsymbol{I} = \begin{bmatrix} 1 - 3k & 0 & 0 \\ 0.3 & 1.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

.

The volume change can be calculated as the determinant of the deformation gradient

$$J = \det\boldsymbol{F} = (1 - 3k) \cdot 1.5$$

If there is incompressible case, the volume change should be equal with 1, therefore:

$$(1 - 3k) \cdot 1.5 = 1$$

$$k = \frac{1}{9}$$

In [16]:
```
k = 1/9

F = np.array([
    [1-3*k,   0, 0],
    [  0.3, 1.5, 0],
    [    0,   0, 1]
])

print(F)
```

```
[[0.66666667 0.         0.        ]
 [0.3        1.5        0.        ]
 [0.         0.         1.        ]]
```

We can see that $\boldsymbol{F}$ is dependent from the material coordinates, which means the deformation is isotropic.

Task 8. - Principal streches

We know that by the deformation gradient we can determine the left Cauchy-Green deformation tensor as

$$\boldsymbol{b} = \boldsymbol{F}\boldsymbol{F}^T$$

In [17]:
```
b = F@F.T

sp.print_latex(sp.Matrix(b))
```

```
\left[\begin{matrix}0.444444444444445 & 0.2 & 0\\0.2 & 2.34 & 0\\0 & 0 & 1.0\end{matrix}\right]
```

The eigendecomposition of the Cauchy-Green tensor looks like

$$\boldsymbol{b} = \sum_{i=1}^{3} \mu_i \cdot \boldsymbol{n_i} \otimes \boldsymbol{n_i} = \sum_{i=1}^{3} \lambda_i^2 \cdot \boldsymbol{n_i} \otimes \boldsymbol{n_i}$$

,

where λ_i are the principal streches and $\boldsymbol{n_i}$ are the principal Eulerian directions

In [18]:
```
(μ1,μ2,μ3), (n1,n2,n3) = np.linalg.eig(b)

(λ1,λ2,λ3) = (μ1**(1/2),μ2**(1/2),μ3**(1/2))


print(λ1,λ2,λ3)
sp.print_latex(sp.Matrix(n1).T.evalf(6))
sp.print_latex(sp.Matrix(n2).T.evalf(6))
sp.print_latex(sp.Matrix(n3).T.evalf(6))
```

```
0.650824305915314 1.5365129896210752 1.0
\left[\begin{matrix}-0.994599 & -0.103797 & 0\end{matrix}\right]
\left[\begin{matrix}0.103797 & -0.994599 & 0\end{matrix}\right]
\left[\begin{matrix}0 & 0 & 1.0\end{matrix}\right]
```

Task 9. - Cauchy stress tensor and Mises Equivalent stress

In the coordinate system of the principal Eulerian directions, the Cauchy stress tensor can be expressed as

$$\boldsymbol{\sigma} = \sum_{i=1}^{3} \sigma_i \cdot \boldsymbol{n_i} \otimes \boldsymbol{n_i}$$

where the principal stress components can be calculated from the principal streches as

$$\sigma_i = \frac{1}{J}\lambda_i \frac{\partial W}{\partial \lambda_i} = \lambda_i \frac{\partial W}{\partial \lambda_i}.$$

The second term of the product can be calculated from the given strain energy potential function as

$$\frac{\partial W}{\partial \lambda_i} = \sum_{i=1}^{2} \frac{2\mu_k}{\alpha_k^2}\alpha_k \lambda_i^{\alpha_k - 1} = \sum_{i=1}^{2} \frac{2\mu_k}{\alpha_k}\lambda_i^{\alpha_k - 1}$$

In [19]:
```
mu1, mu2, a1, a2 = params

dWdλ1 = 2*mu1/a1 * λ1**(a1-1) + 2*mu2/a2 * λ1**(a2-1)
dWdλ2 = 2*mu1/a1 * λ2**(a1-1) + 2*mu2/a2 * λ2**(a2-1)
dWdλ3 = 2*mu1/a1 * λ3**(a1-1) + 2*mu2/a2 * λ3**(a2-1)

# tmp = np.array([
#     [λ1*dWdλ1,0,0],
#     [0,λ2*dWdλ2,0],
#     [0,0,λ3*dWdλ3]
# ])

tmp = λ1*dWdλ1 * np.outer(n1,n1) + λ2*dWdλ2 * np.outer(n2,n2) + λ3*dWdλ3 * np.outer(n3,n3)


s = tmp - 1/3*np.trace(tmp)*np.eye(3)
p = -s[2,2] * np.eye(3)
sig = s + p

sp.print_latex(sp.Matrix(s).evalf(5))
sp.print_latex(sp.Matrix(p).evalf(5))
sp.print_latex(sp.Matrix(sig).evalf(5))
```

```
\left[\begin{matrix}-6.9058 & -1.5056 & 0\\-1.5056 & 7.3637 & 0\\0 & 0 & -0.45787\end{matrix}\r
ight]
\left[\begin{matrix}0.45787 & 0 & 0\\0 & 0.45787 & 0\\0 & 0 & 0.45787\end{matrix}\right]
\left[\begin{matrix}-6.4479 & -1.5056 & 0\\-1.5056 & 7.8215 & 0\\0 & 0 & 0\end{matrix}\right]
```

The Mises equivalent stress can be expressed as

$$\sigma_{vM} = \sqrt{\frac{3}{2}\boldsymbol{s} : \boldsymbol{s}}$$

In [20]:
```
sig_vM = np.sqrt(3/2 * np.sum(s**2))

sig_vM
```

Out[20]:
```
12.648525122821695
```

In [21]:
```
params
```

Out[21]:
```
array([ 3.18596983,  4.62213918, -1.48228961,  1.77455216])
```