World Academy of Science, Engineering and Technology
International Journal of Mechanical and Mechatronics Engineering
Vol:7, No:1, 2013

Determination Of Extreme Shear Stresses In Teaching Mechanics Using Freely Available Computer Tools

Rado Flajs

Abstract—In the present paper the extreme shear stresses with the corresponding planes are established using the freely available computer tools like the `Gnuplot`, `Sage`, `R`, `Python` and `Octave`. In order to support these freely available computer tools, their strong symbolical and graphical abilities are illustrated.

The nature of the stationary points obtained by the Method of Lagrangian Multipliers can be determined using freely available computer symbolical tools like `Sage`.

The characters of the stationary points can be explained in the easiest way using freely available computer graphical tools like `Gnuplot`, `Sage`, `R`, `Python` and `Octave`. The presented figures improve the understanding of the problem and the obtained solutions for the majority of students of civil or mechanical engineering.

Keywords—engineering, continuum mechanics, extreme shear stresses, `Gnuplot`, `Sage`, `R`, `Python`, `Octave`

I. Introduction

In the study of continuum mechanics in engineering and physics a great attention is devoted to the consideration of the normal and shear stresses. Shear stress $\tau \equiv \sigma_{nt}$ is defined by the projection of the stress vector $\vec{\sigma}_n$ to the corresponding plane. The physical meaning of the stress vector, normal and shear stresses are seen from Fig. 1. The square of the shear

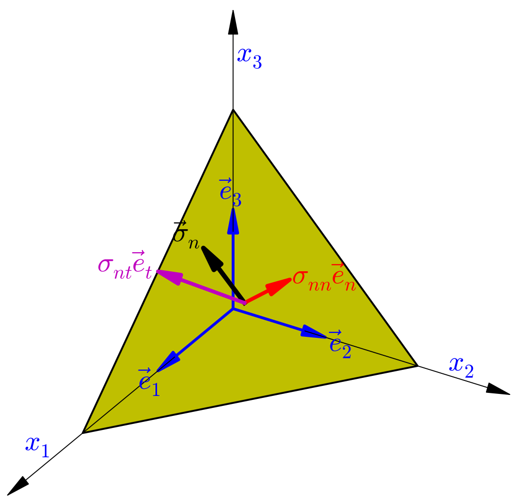


Fig. 1. The physical meaning of the components of the stress vector i.e. the normal and shear stresses.

R. Flajs is with the Chair of Mechanics of Faculty of Civil and Geodetic Engineering, University of Ljubljana, Slovenia, e-mail: (rado.flajs@fgg.uni-lj.si

stress can be obtained by Pitagora theorem

$$\sigma_{nt}^2 = \vec{\sigma}_n \cdot \vec{\sigma}_n - (\vec{\sigma}_n \cdot \vec{e}_n)^2 . \tag{1}$$

Using the Lagrange identity

$$(\vec{a} \times \vec{b}) \cdot (\vec{c} \times \vec{d}) = (\vec{a} \cdot \vec{c})\,(\vec{b} \cdot \vec{d}) - (\vec{b} \cdot \vec{c})\,(\vec{a} \cdot \vec{d})$$

we get

$$\begin{aligned} \sigma_{nt}^2 &= \vec{\sigma}_n \cdot \vec{\sigma}_n - (\vec{\sigma}_n \cdot \vec{e}_n)^2 \\ &= (\vec{\sigma}_n \cdot \vec{\sigma}_n)\,(\vec{e}_n \cdot \vec{e}_n) - (\vec{\sigma}_n \cdot \vec{e}_n)^2 \\ &= (\vec{\sigma}_n \times \vec{e}_n) \cdot (\vec{\sigma}_n \times \vec{e}_n) = \|\vec{\sigma}_n \times \vec{e}_n\|^2 . \end{aligned} \tag{2}$$

Applying the principal coordinate system (x_1, x_2, x_3) and referring to the principal basis $\vec{e}_1$, $\vec{e}_2$, $\vec{e}_3$ the stress vector σ_n can be expressed by the linear combination

$$\vec{\sigma}_n = \sigma_{11}\, e_{n1}\, \vec{e}_1 + \sigma_{22}\, e_{n2}\, \vec{e}_2 + \sigma_{33}\, e_{n3}\, \vec{e}_3,$$

where σ_{11},σ_{22},σ_{33} are principal stresses and e_{n1},e_{n2},e_{n3} are directional cosines of the unit normal at the arbitrary plane [1], [2]. According to that we can write

$$\vec{\sigma}_n \times \vec{e}_n = \begin{vmatrix} \vec{e}_1 & \vec{e}_2 & \vec{e}_3 \\ \sigma_{11}\, e_{n1} & \sigma_{22}\, e_{n2} & \sigma_{33}\, e_{n3} \\ e_{n1} & e_{n2} & e_{n3} \end{vmatrix}$$

and obtain

$$\begin{aligned} \sigma_{nt}^2 &= (\sigma_{11} - \sigma_{22})^2\; e_{n1}^2\, e_{n2}^2 + (\sigma_{33} - \sigma_{11})^2\; e_{n1}^2\, e_{n3}^2 \\ &\quad + (\sigma_{22} - \sigma_{33})^2\; e_{n2}^2\, e_{n3}^2 . \end{aligned} \tag{3}$$

It is worth to note that Eq. (3) presents the component form of the equation $|\sigma_{nt}| = \|\vec{\sigma}_n \times \vec{e}_n\|$ in the principal coordinate system.

One of the interesting problems is to find the extreme shear stresses with the corresponding planes. The extreme shear stresses play the important role in Tresca yielding criteria [1], [2], [3], [4], [5], [6], [7].

Using Eq. (3) we have to mathematically determine the directional cosines e_{n1},e_{n2},e_{n3} of unit normal to the planes where the extreme shear stresses appear. Since the sum of directional cosines squares equals one, we will search the maximum of the square of the shear stresses under the condition $g(e_{n1}, e_{n2}, e_{n3}) = e_{n1}^2 + e_{n2}^2 + e_{n3}^2 - 1 = 0$. Using the Method of Lagrangian Multipliers [8] we can find the stationary points of the Lagrangian function $L(e_{n1}, e_{n2}, e_{n3}, \lambda) = \sigma_{nt}^2(e_{n1}, e_{n2}, e_{n3}) + \lambda\, g(e_{n1}, e_{n2}, e_{n3})$. This method requires the usage of the differential calculus. The classification of the stationary points can be quite cumbersome and is usually omitted in the undergraduate study of mechanics. The Method of Lagrangian Multipliers can be naturally explained by the usage of the freely available graphics tools like `Gnuplot` [9], `Sage` [10], `R` [11], `Python` [12] and `Octave` [13]. The characters of the obtained stationary points (saddle points or Extrema), which usually cause a bit of confusion, can be determined immediately from the obtained pictures.

II. Classification of the stationary points, obtained by the Method of Lagrangian Multipliers, using `Sage` symbolic tools

The type of the stationary point (saddle points or Extrema) should be determined from the eigenvalues of Hessian second derivative matrix of Lagrangian function L as follows:

$$H = \begin{bmatrix} \dfrac{\partial^2 L}{\partial e_{n1}^2} & \dfrac{\partial^2 L}{\partial e_{n1}\partial e_{n2}} & \dfrac{\partial^2 L}{\partial e_{n1}\partial e_{n3}} \\ \dfrac{\partial^2 L}{\partial e_{n2}\partial e_{n1}} & \dfrac{\partial^2 L}{\partial e_{n2}^2} & \dfrac{\partial^2 L}{\partial e_{n2}\partial e_{n3}} \\ \dfrac{\partial^2 L}{\partial e_{n3}\partial e_{n1}} & \dfrac{\partial^2 L}{\partial e_{n3}\partial e_{n2}} & \dfrac{\partial^2 L}{\partial e_{n3}^2} \end{bmatrix} \qquad (4)$$

at stationary points

$$(e_{n1}, e_{n2}, e_{n3}, \lambda) = \left(\pm\frac{\sqrt{2}}{2}, \pm\frac{\sqrt{2}}{2}, 0, -\sigma_{11}\,\sigma_{22}\right), \qquad (5a)$$

$$(e_{n1}, e_{n2}, e_{n3}, \lambda) = \left(\pm\frac{\sqrt{2}}{2}, 0, \pm\frac{\sqrt{2}}{2}, -\sigma_{11}\,\sigma_{33}\right), \qquad (5b)$$

$$(e_{n1}, e_{n2}, e_{n3}, \lambda) = \left(0, \pm\frac{\sqrt{2}}{2}, \pm\frac{\sqrt{2}}{2}, -\sigma_{22}\,\sigma_{33}\right). \qquad (5c)$$

The calculation can be performed using the following procedure in `Sage 4.6.2` as follows:

```
# attach('Hessian.sage')

# function to minimizes
def funF(sig11,sig22,sig33,x,y,z,lam):
    snn = sig11 * x^2 + sig22 * y^2 + sig33 * z^2
    sn2 = sig11^2 * x^2 + sig22^2 * y^2 + sig33^2 * z^2
    phi = 1 - x^2 - y^2 - z^2
    snt2 = sn2 - snn^2
    F = snt2 + lam * phi
    return F

# first and second derivatives
def D(F,x):
    return derivative(F,x)
def D2(F,x,y):
    return derivative(F,x,y)

# Hessian matrix
def Hesse(F,x,y,z):
    return matrix([[D2(F,x,x), D2(F,x,y), D2(F,x,z)],
                   [D2(F,y,x), D2(F,y,y), D2(F,y,z)],
                   [D2(F,z,x), D2(F,z,y), D2(F,z,z)]])



# simple test
sig11,sig22,sig33 = var('s11,s22,s33')
x,y,z,lam = var('x,y,z,lam')
F = funF(sig11,sig22,sig33, x,y,z,lam)

# all stationary points
S = solve([D(F,x)==0,D(F,y)==0,D(F,z)==0,D(F,lam)==0],\
          x,y,z,lam,solution_dict=True)

# stationary points analysis
n = len(S)
for (k,s) in zip(range(1,n+1),S):
    # function value
    f = F.subs(x=s[x],y=s[y],z=s[z],lam=s[lam])
    # Hessian matrix
    h = Hesse(F,x,y,z).subs(x=s[x],y=s[y],z=s[z],lam=s[lam])
    # eigenvalues of Hessian matrix
    l = h.eigenvalues()
    # eigenvectors of Hessian matrix
    e = h.eigenvectors_right()
    # simplify
    if f != 0:
        f = f.factor()
    for i in range(len(l)):
        if l[i] != 0:
            l[i] = l[i].factor()
    print k, s, f, l
#    print k, s, f, l, e
```

Applying this program one can obtain the following results, where for each obtained solution the values in the first, the second, the third and the fourth rows present index of the obtained solution, the stationary point, the value of Lagrangian function L and the eigenvalues of Hessian matrix at those point, respectively:

```
1
{y: 0, lam: -s11^2, x: -1, z: 0}
0
[2*(s11 - s33)^2, 2*(s11 - s22)^2, -8*s11^2]

2
{y: 0, lam: -s11^2, x: 1, z: 0}
0
[2*(s11 - s33)^2, 2*(s11 - s22)^2, -8*s11^2]

3
{y: 1, lam: -s22^2, x: 0, z: 0}
0
[2*(s22 - s33)^2, -8*s22^2, 2*(s11 - s22)^2]

4
{y: -1, lam: -s22^2, x: 0, z: 0}
0
[2*(s22 - s33)^2, -8*s22^2, 2*(s11 - s22)^2]

5
{y: -1/2*sqrt(2), lam: -s11*s22, x: -1/2*sqrt(2), z: 0}
1/4*(s11 - s22)^2
[2*(s22 - s33)*(s11 - s33), -4*s11^2 - 4*s22^2, 0]

6
{y: 1/2*sqrt(2), lam: -s11*s22, x: -1/2*sqrt(2), z: 0}
1/4*(s11 - s22)^2
[2*(s22 - s33)*(s11 - s33), -4*s11^2 - 4*s22^2, 0]

7
{y: -1/2*sqrt(2), lam: -s11*s22, x: 1/2*sqrt(2), z: 0}
1/4*(s11 - s22)^2
[2*(s22 - s33)*(s11 - s33), -4*s11^2 - 4*s22^2, 0]

8
{y: 1/2*sqrt(2), lam: -s11*s22, x: 1/2*sqrt(2), z: 0}
1/4*(s11 - s22)^2
[2*(s22 - s33)*(s11 - s33), -4*s11^2 - 4*s22^2, 0]

9
{y: 0, lam: -s33^2, x: 0, z: -1}
0
[-8*s33^2, 2*(s22 - s33)^2, 2*(s11 - s33)^2]

10
{y: 0, lam: -s33^2, x: 0, z: 1}
0
[-8*s33^2, 2*(s22 - s33)^2, 2*(s11 - s33)^2]

11
{y: 0, lam: -s11*s33, x: -1/2*sqrt(2), z: -1/2*sqrt(2)}
1/4*(s11 - s33)^2
[-4*s11^2 - 4*s33^2, -2*(s22 - s33)*(s11 - s22), 0]

12
{y: 0, lam: -s11*s33, x: -1/2*sqrt(2), z: 1/2*sqrt(2)}
1/4*(s11 - s33)^2
[-4*s11^2 - 4*s33^2, -2*(s22 - s33)*(s11 - s22), 0]
```

```
13
{y: 0, lam: -s11*s33, x: 1/2*sqrt(2), z: -1/2*sqrt(2)}
1/4*(s11 - s33)^2
[-4*s11^2 - 4*s33^2, -2*(s22 - s33)*(s11 - s22), 0]

14
{y: 0, lam: -s11*s33, x: 1/2*sqrt(2), z: 1/2*sqrt(2)}
1/4*(s11 - s33)^2
[-4*s11^2 - 4*s33^2, -2*(s22 - s33)*(s11 - s22), 0]

15
{y: -1/2*sqrt(2), lam: -s22*s33, x: 0, z: -1/2*sqrt(2)}
1/4*(s22 - s33)^2
[-4*s22^2 - 4*s33^2, 2*(s11 - s33)*(s11 - s22), 0]

16
{y: -1/2*sqrt(2), lam: -s22*s33, x: 0, z: 1/2*sqrt(2)}
1/4*(s22 - s33)^2
[-4*s22^2 - 4*s33^2, 2*(s11 - s33)*(s11 - s22), 0]

17
{y: 1/2*sqrt(2), lam: -s22*s33, x: 0, z: -1/2*sqrt(2)}
1/4*(s22 - s33)^2
[-4*s22^2 - 4*s33^2, 2*(s11 - s33)*(s11 - s22), 0]

18
{y: 1/2*sqrt(2), lam: -s22*s33, x: 0, z: 1/2*sqrt(2)}
1/4*(s22 - s33)^2
[-4*s22^2 - 4*s33^2, 2*(s11 - s33)*(s11 - s22), 0]
```

We can assume

$$\sigma_{11} \leq \sigma_{22} \leq \sigma_{33}. \tag{6}$$

Using this assumption it is easy to see, that the stationary points in solutions no. 1–10 and 15–18 are saddle points, since the eigenvalues are generally of different sign, while the stationary points in solutions no. 11–14 are Extrema, since the all eigenvalues are generally of the same sign. So, from the obtained results we can conclude that the second points in Eq. (5b) are Extrema, while the first and the last points in Eqs. (5a) and (5c) are the saddle points. It is also worth to emphasize that the solutions no. 1–4 and 9–10 present the principal planes.

III. Classification of the stationary points, obtained by the Method of Lagrangian Multipliers, using Gnuplot, Sage, R, Python and Octave graphics tools

The Method of Lagrangian Multipliers has a nice graphical explanation [8]. In order to explain the nature of the stationary points obtained by the presented computer graphics tools, the similar ideas will be used.

A. *Graphical explanation of the Method of Lagrangian Multipliers on a two dimensional example*

In order to make the presentation easy to understand, let us begin with the simple two dimensional example. We will search Extrema of the function

$$f(x,y) = \begin{cases} x^2 - (2\,y)^2, & y \geq 0 \\ x^2 + (2\,y)^2, & y \leq 0 \end{cases} \tag{7}$$

fulfilling the condition

$$g(x,y) = x^2 + y^2 - 1 = 0. \tag{8}$$

Using the Method of Lagrangian Multipliers we can find four stationary points i.e. $T_1(-1,0)$, $T_2(0,-1)$, $T_3(1,0)$ and $T_4(0,1)$. From Fig. 2 it is easy to see that at these four

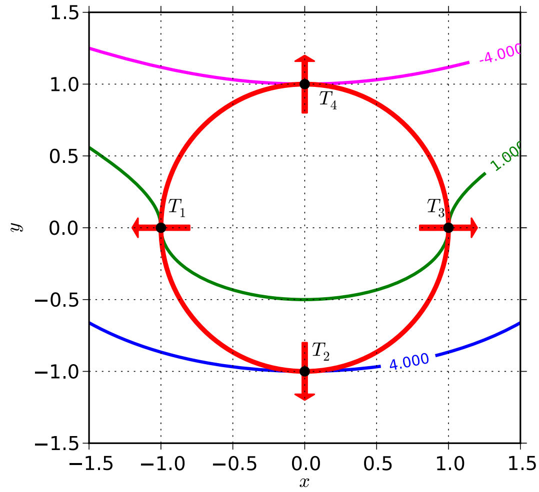


Fig. 2. Graphical explanation of the characters of the stationary points, saddle points or Extrema, obtained by the Method of Lagrangian Multipliers.

points the function gradients ∇f and ∇g, in Fig 2 denoted by red arrows, coincide, what in fact confirms the existence of constant λ such that $\nabla f + \lambda \nabla g = 0$. However, it is also clearly seen from Fig. 2 that only two of four points described above, namely T_2 and T_4, are Extrema. We have to emphasize that at these Extrema the unit circle touches the curves $f(x,y) = c$ for $c = -4$ and $c = 4$, meanwhile at the saddle points T_1 and T_3 unit circle intersects the curve $f(x,y) = 1$.

B. *Graphical explanation of the character of the stationary shear stress points obtained by the Method of Lagrangian Multipliers using graphical tools on two dimensional example*

A nice explanation of the two dimensional stress state is presented in [14].

Let us consider the case $\sigma_{11} = \sigma_{22} < \sigma_{33}$. Equation (3) simplifies into

$$\begin{aligned} \sigma_{nt}^2 &= (\sigma_{11} - \sigma_{22})^2 \; e_{n1}^2 \, e_{n2}^2 + (\sigma_{33} - \sigma_{11})^2 \; e_{n1}^2 \, e_{n3}^2 \\ &\quad + (\sigma_{22} - \sigma_{33})^2 \; e_{n2}^2 \, e_{n3}^2 \\ &= (\sigma_{11} - \sigma_{33})^2 \; \left(e_{n1}^2 + e_{n2}^2\right) \; e_{n3}^2. \end{aligned} \tag{9}$$

Consequently we get

$$\sigma_{nt} = \pm \left(\sigma_{11} - \sigma_{33}\right) \, e_{n3} \sqrt{e_{n1}^2 + e_{n2}^2}. \tag{10}$$

Employing the abbreviation $x = e_{n1}$, $y = e_{n2}$, $z = e_{n3}$, $r = \sqrt{x^2 + y^2}$ we maximize the function $f(r,z) = (\sigma_{33} - \sigma_{11}) \, r \, z$ under the condition $g(r,z) = r^2 + z^2 - 1 = 0$. Using the Method of Lagrangian Multipliers, we can find four stationary points $(r,z) = \left(\pm\frac{\sqrt{2}}{2}, \pm\frac{\sqrt{2}}{2}\right)$ (in the Fig. 3 denoted by grey bullets) where the gradients ∇f and ∇g coincide. The gradients are perpendicular to the curves $g = 0$ and $f = \tau_{\max}$. All stationary points are Extrema, since the unit circle (graph of the function $g = 0$) only touches (not intersects) the graph of the function $f = \tau_{\max}$ at these points.

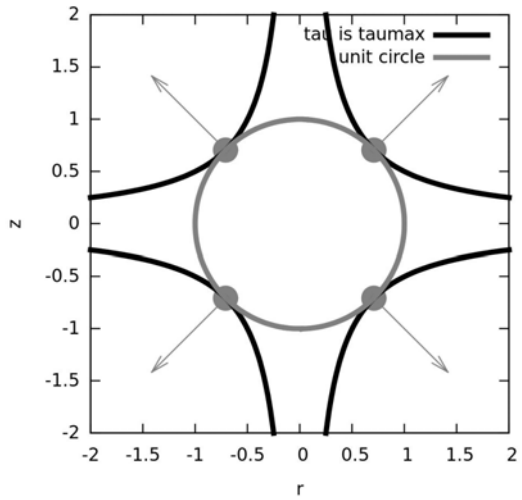


Fig. 3. Graphical explanation of the characters of stationary (grey) points obtained by the Method of Lagrangian Multipliers. The all grey points are Extrema, since unit circle touches the graph of the function $f = \tau_{\max}$ at these points. The corresponding programming code in case $\sigma_{11} = 1$, $\sigma_{33} = 4$ written in the `Gnuplot 4.4` is presented in the Appendix.

C. *Graphical explanation of the character of the stationary shear stress points obtained by the Method of Lagrangian Multipliers using graphical tools on three dimensional example*

In three dimensional case, where $\sigma_{11} < \sigma_{22} < \sigma_{33}$, the situation is a bit more complicated, so we will use the computer graphical tools in order to classify the character of the stationary points directly from the obtained picture. Employing the principal coordinate system (x, y, z) and using the Method of Lagrangian Multipliers one can obtain 12 stationary points, namely $\left(\pm\frac{\sqrt{2}}{2}, \pm\frac{\sqrt{2}}{2}, 0\right)$, $\left(\pm\frac{\sqrt{2}}{2}, 0, \pm\frac{\sqrt{2}}{2}\right)$, $\left(0, \pm\frac{\sqrt{2}}{2}, \pm\frac{\sqrt{2}}{2}\right)$. We will construct the unit sphere, i.e. the graph of the function $g(x, y, z) = x^2 + y^2 + z^2 - 1 = 0$, and three surfaces corresponding the equations:

$$\tau(x, y, z) \equiv \sigma_{nt}(x, y, z) = \frac{\sigma_{33} - \sigma_{22}}{2}, \tag{11a}$$

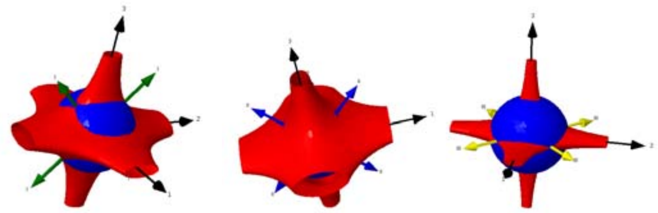
$$\tau(x, y, z) \equiv \sigma_{nt}(x, y, z) = \frac{\sigma_{33} - \sigma_{11}}{2}, \tag{11b}$$

$$\tau(x, y, z) \equiv \sigma_{nt}(x, y, z) = \frac{\sigma_{22} - \sigma_{11}}{2}. \tag{11c}$$

Since the unit sphere intersects the graph of the functions (11a) and (11c), the corresponding stationary points $\left(\pm\frac{\sqrt{2}}{2}, \pm\frac{\sqrt{2}}{2}, 0\right)$ and $\left(0, \pm\frac{\sqrt{2}}{2}, \pm\frac{\sqrt{2}}{2}\right)$ are saddle points. In the second case the unit sphere touches the graph of the function (11b), so the corresponding stationary points $\left(\pm\frac{\sqrt{2}}{2}, 0, \pm\frac{\sqrt{2}}{2}\right)$, are Extrema.

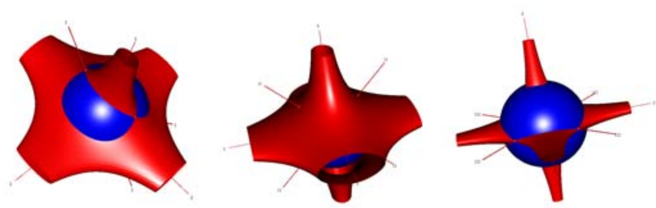
All these facts can be seen easily from Figs. 4, 5, 6 and 7 and can obtained from graphical programming tools like `Sage` [10], `R` [11], `Python` [12] and `Octave` [13] using the corresponding programming codes presented in the Appendix. The principal axes in Figs. 4, 5, 6 and 7, are denoted with indices 1, 2 and 3.

The stationary points in Figs 4(b), 5(b), 6(b), and 7(b), are Extrema, since the unit ball touches the graph of the function $f = \tau_{\max}$ at these points. The stationary points in Figs. 4(a), 5(a), 6(a), 7(a) and 4(c), 5(c), 6(c), 7(c) are saddle points, since the unit ball intersect the graph of the function $f = c$ for $c = \frac{\sigma_{33} - \sigma_{22}}{2}$ and $c = \frac{\sigma_{22} - \sigma_{11}}{2}$, respectively. The corresponding programming codes written in `Sage 4.6.2`, `R 2.10.1`, `Python` and `Octave 3.2.4` are presented in the Appendix.



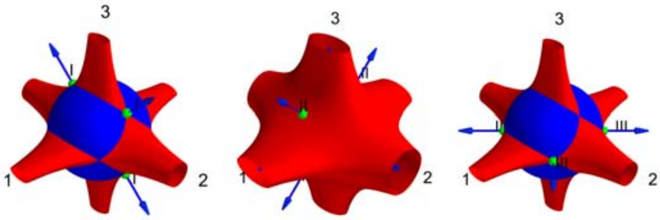
(a) Saddle points (b) Extrema (c) Saddle points

Fig. 4. Saddle points and Extrema obtained by `Sage`.



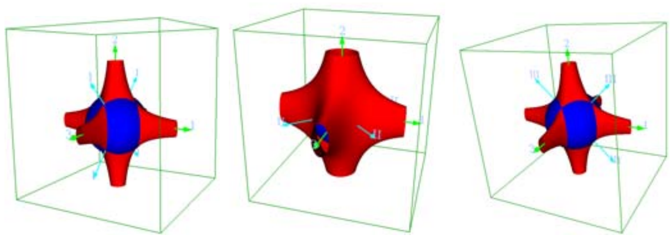
(a) Saddle points (b) Extrema (c) Saddle points

Fig. 5. Saddle points and Extrema obtained by `R`.



(a) Saddle points (b) Extrema (c) Saddle points

Fig. 6. Saddle points and Extrema obtained by `Python`.



(a) Saddle points (b) Extrema (c) Saddle points

Fig. 7. Saddle points and Extrema obtained by `Octave`.

IV. Conclusion

In the paper the extreme shear stresses with the corresponding planes were determined. At first step, the nature of the stationary points obtained by the Method of Lagrangian Multipliers is established symbolically using the freely available symbolic tool `Sage` [10]. Next, the new simple graphical procedure to establish the character of the stationary points is presented, employing the freely available graphical tools like `Sage` [10], `R` [11], `Python` [12] and `Octave` [13]. The same quality of graphical presentation could be achieved as with the commercial ones like `MATLAB` and `Mathematica` using the same length of programming code. The authors found all these tools helpful in their teaching process. The presented figures help the students of civil or mechanical engineering to improve their understanding of the problem and the obtained solutions.

V. Appendix: Applied graphics programming codes

The surfaces shown in Figs. 3, 4, 5, 6 and 7 were obtained by the usage of the `Gnuplot` [9], `Sage` [10], `R` [11], `Python` [12] and `Octave` [13] program codes presented below. The `Octave` graphics is principally based on `Gnuplot` graphics, which is well suitable for two dimensional presentation, but less suitable for three dimensional plots. In order to avoid this inconvenience, the `Octave vrml` package [15] was included. Applying this package, high quality three dimensional graphics plot can be constructed in standardized `vrml` or `wrl` format [16], by readable using well known programming tools like `view3dscene` [17] or `FreeWRL` [18].

A. `Gnuplot 4.4`

```
# notation according to article: x = r, y = z
reset
# pricipal stresses
sig1 = 1.0 # sig2 = 1.0
sig3 = 4.0
taumax = abs(sig3 - sig1)/2 # extreme shear stress
circle(x, y) = sqrt(x**2 + y**2) - 1
# shear stresses as a function of
# radial coordinate x and vertical coordinate y
tau(x,y,sig1,sig3)=sqrt((sig1-sig3)**2 * x**2 * y**2)-taumax
set terminal wxt
set view map
unset surface
set contour # contour plot
set xrange [-2:2]
set yrange [-2:2]
set isosample 100, 100
set cntrparam levels discrete 0
# contour plot of the unit circle
# onto file ’circle.dat’
set table "circle.dat"
splot circle(x,y)
unset table
# contour plot of function tau
# is taumax onto file ’tau.dat’
set table "tau.dat"
splot tau(x,y,sig1,sig3)
unset table

plot "stationarypoints.dat" u 1:2 w p pt 7 ps 3 lc 2 notitle, \
 "" u 1:2:1:2 w vectors notitle, \
     "tau.dat" u 1:2 w l lw 4 lt 1 lc 1 t "tau is taumax", \
     "circle.dat" u 1:2 w l lw 4 lt 1 lc 3 t "unit circle"
set xlabel "r"; set ylabel "z"
set size square
replot
```

B. `Sage 4.6.2`

```
def ball(x,y,z):
    return sqrt(x^2+y^2+z^2)
def tau(x,y,z, sig1=1,sig2=2,sig3=3):
    return ball((sig1-sig2)*x*y,(sig1-sig3)*x*z,(sig2-sig3)*y*z)
def plotUnitBall(c):
    [x,y,z] = var(’x,y,z’)
    return implicit_plot3d(ball(x,y,z),\
              (x,-1.5,1.5), (y,-1.5,1.5), (z,-1.5,1.5),  \
              plot_points=50, contour=1, color=c, opacity=1)
def plotTauIsConst(c, sig1=1,sig2=2,sig3=3, isp=0):
    tausp = map(abs,[(sig3-sig2)/2,(sig3-sig1)/2,(sig2-sig1)/2])
    [x,y,z] = var(’x,y,z’)
    return implicit_plot3d(tau(x,y,z,sig1,sig2,sig3), \
               (x,-2,2), (y,-2,2), (z,-2,2), \
               plot_points=100, contour=tausp[isp], \
               color=c, opacity=1)
def plotStationaryPoints(p, c):
    return point(p, rgbcolor=c, pointsize=150)
def plotStationaryArrow(p1, p2, c):
    return arrow(p1,p2, rgbcolor=c, width=2, arrowsize=5)
def plotStationaryText(t, p):
    return text3d(t,p*1.05, fontsize=80, color=’black’)
def plotCoordinateSystem(c):
    COOR = matrix([[1,0,0],[0,1,0],[0,0,1]])*3
    return (sum(arrow((0,0,0),COOR[i], rgbcolor=c, \
                width=2, arrowsize=5) for i in [0..2]) + \
            sum(text3d(i+1,COOR[i]*1.05,fontsize=12, \
                color=c) for i in [0..2]))

def tauplot(sig1=1,sig2=2,sig3=3, isp=0):
#  stationary point colors
   col = [’green’, ’blue’, ’yellow’]
#  stationary points annotations
   txt = ["I", "II", "III"]
#  plot of the unit ball isosurface
   B1 = plotUnitBall(’blue’)
#  plot of the isosurface tau = const
   T = plotTauIsConst(’red’,sig1,sig2,sig3, isp)
#  stationary points
#  stationary points for isp=1
   SP1 = matrix([[0,-1,-1],[0,-1,1],[0,1,-1],[0,1,1]])\
                   *sqrt(2)/2
#  stationary points for isp=2
   SP2 = matrix([[-1,0,-1],[-1,0,1],[1,0,-1],[1,0,1]])\
                   *sqrt(2)/2
#  stationary points for isp=3
   SP3 = matrix([[-1,-1,0],[-1,1,0],[1,-1,0],[1,1,0]])
                   *sqrt(2)/2
#  all stationary points
   SP = [SP1, SP2, SP3]
#  stationary points annotation
#  points
   S  = plotStationaryPoints(SP[isp],col[isp])
#  text
   TS = sum(plotStationaryText(txt[isp],SP[isp][i]*2)\
           for i in [0..3])
#  gradients
   AS = sum(plotStationaryArrow(SP[isp][i],SP[isp][i]*2,\
                           x col[isp])\
           for i in [0..3])
#  plot Cartesian coordinate system
#  with basis e1, e2 and e3
#  axes + text
   CS = plotCoordinateSystem(’black’)
#  join all figure parts
   P = T+B1+S+AS+TS+CS # tau is const + unit sphere \
       + stationary points + coor system
   return P
```

C. `R 2.10.1`

```
# source(’tauplot.R’); r = tauplot(1,2,4,1)
library(rgl)    # base 3d graphics library
library(misc3d) # miscelaneous 3d graphics library

ball = function(x,y,z){ sqrt(x^2+y^2+z^2) }
tau = function (x,y,z,sig1=1,sig2=2,sig3=3){
    ball((sig1-sig2)*x*y,(sig1-sig3)*x*z,(sig2-sig3)*y*z)
    }

plotUnitBall = function(ballc=’blue’){
    x = seq(-2,2,len=200)
    contour3d(ball,1,x,x,x,color=ballc,alpha=1,add=T)
    }
plotTauIsConstant =
function(sig1=1,sig2=2,sig3=3,tauc=’red’,isp=1){
    tausp = abs(c(sig3,sig1,sig2)-c(sig2,sig3,sig1))/2
    f = function(x,y,z){ tau(x,y,z,sig1,sig2,sig3) }
    x = seq(-2,2,len=200)
    contour3d(f,tausp[isp],x,x,x,color=tauc,alpha=1,add=T)
    }
```

```
plotStationaryPoints = function(spat){
    plot3d(spat$x,spat$y,spat$z,spat$col,
           size=2,type='s',add=T)
    }
plotStationaryArrows = function(spat){
    d = 1+(0:100/100)
    for (i in 1:4){
        plot3d(spat$x[i]*d,spat$y[i]*d,spat$z[i]*d,
               spat$col,size=1,type='s',add=T)
    } }
plotStationaryText = function(spat){
    for (i in 1:4){
        text3d(spat$x[i]*2.1,spat$y[i]*2.1,
               spat$z[i]*2.1,spat$t[i],add=T)
    } }
plotCoordinateSystem = function(coor){
    d = 1.5+(0:100/100)
    for (i in 1:3){
        plot3d(coor$x[i]*d,coor$y[i]*d,
               coor$z[i]*d,coor$col[i],
               size=0.6,type='s',add=T)
        text3d(coor$x[i]*2.6,coor$y[i]*2.6,
               coor$z[i]*2.6,coor$t[i],add=T)
    } }
tauplot = function(sig1=1,sig2=2,sig3=3,isp=1){
#   stationary points color
    spc = list('green', 'blue', 'yellow')
#   stationary points numbering
    spt = list('I', 'II', 'III')
#   plot of the unit ball
    pball = plotUnitBall('blue')
#   plot of the surface tau = const
    ptau = plotTauIsConstant(sig1,sig2,sig3,'red',isp)
#   stationary points
    s = sqrt(2)/2;
    rep4 = function(x) { rep(x, times=4) }
#   stationary points for isp=1
    sp1 = data.frame(x=rep4(0),
                     y=c(s,-s,s,-s),z=c(s,s,-s,-s),
                     col=rep4(1),t=rep4('I'))
#   stationary points for isp=2
    sp2 = data.frame(y=rep4(0),
                     x=c(s,-s,s,-s),z=c(s,s,-s,-s),
                     col=rep4(2),t=rep4('II'))
#   stationary points for isp=3
    sp3 = data.frame(z=rep4(0),
                     x=c(s,-s,s,-s),y=c(s,s,-s,-s),
                     col=rep4(3),t=rep4('III'))
    spat = list(sp1,sp2,sp3) # all stationary points
#   stationary points annotation
    pspat = plotStationaryPoints(spat[[isp]])
    pspt =  plotStationaryText(spat[[isp]])
    pspa =  plotStationaryArrows(spat[[isp]])
#   plot Cartesian coordinate system
#   with basis e1, e2 and e3
    coor = data.frame(x=c(1,0,0),y=c(0,1,0),z=c(0,0,1),
                      col=c(4,4,4),t=c('1','2','3'))
    pcoor = plotCoordinateSystem(coor)
    }
```

D. Octave 3.2.4

```
function is =  tauplot2(sig11, sig22, sig33, isp)
% call: tauplot(sig11, sig22, sig33, isp)
% function plots the surface tau =
% taumax with the unit ball (inserted)
% input parameters:
%   sig11, sig22, sig33 are the principal stresses
%   isp is the index of the plot group of
%       the stationary points
    if nargin < 3,
        sig11 = 1; sig22 = 2; sig33 = 3;
    end
    sig = [sig11 sig22 sig33];
    if nargin < 4,
        isp=1;
    end
% ball
    R = @(x,y,z) sqrt(x.^2 + y.^2 + z.^2);
% tau_{xi eta}
    tau = @(x,y,z, sig)
             R((sig11-sig22)*x.*y, ...
             (sig11-sig33)*x.*z, ...
             (sig22-sig33)*y.*z);
% stationary (extreme) shear stresses
% stationary shear stresses
    tausp = abs(sig([2 3 1]) - sig([3 1 2]))/2;
% max shear stress
    taumax = max(abs(tausp));
    if taumax == 0,
       error('Hidrostatic stress state.');
    end
% domain plot points
    xmax = 2; ymax = 2; zmax = 2; d = -1:0.01:1;
    x = d*xmax; y = d*ymax; z = d*zmax;
    [X,Y,Z] = meshgrid(x,y,z);
% vrml plot of the unit ball
    r = R(X,Y,Z);
    iso = 1;
    [F, V, col] = isosurface(X, Y, Z, r, iso, r);
    s1  = vrml_faces(V', F', "col", [0 0 1],'tran',0.0);
% vrml plot of the surface tau(x1,y1,z1) = const
    T = tau(X,Y,Z, sig);
    [F, V, col] = isosurface(X,Y,Z,T,tausp(isp),T);
    s2 = vrml_faces(V', F', "col", [1 0 0],'tran',0.0);
% vrml plot of the principal coordinate axes
    scale = 2.5;
    a = [0 0 0];
    b = {[scale 0 0],[0 scale 0],[0 0 scale]};
    txt = {'1','2','3'};
    for i = 1:3
        eg{i} = vrml_arrowAB(a,b{i});
        egx{i} = vrml_textT(txt{i},b{i});
    end
% vrml plot of the stationary/extreme shear directions
    s22 = sqrt(2)/2;
    switch isp
        case 1
            spt = {[0 -s22 -s22],[0 -s22 s22],...
                   [0 s22 -s22],[0 s22 s22]};
        case 2
            spt = {[-s22 0 -s22],[-s22 0 s22],...
                   [s22 0 -s22],[s22 0 s22]};
        case 3
            spt = {[-s22 -s22 0],[-s22 s22 0],...
                   [s22 -s22 0],[s22 s22 0]};
    end
    stxt = {'I','II','III'};
    for i = 1:4
        sp{i} = vrml_arrowAB(spt{i}*0,2*spt{i},[0 1 1]);
        spx{i} = vrml_textT(stxt{isp},2*spt{i});
    end
% vrml background
    bkg = vrml_Background ("skyColor",[1 1 1]);
% join vrml plots together
    fr = vrml_frame([0 0 0],[0 0 pi],'scale',0.5);
    save_vrml('tauplot.wrl','nobg',s1,s2, ...
              eg{1},eg{2},eg{3}, egx{1},egx{2},egx{3}, ...
              sp{1},sp{2},sp{3}, sp{4}, ...
              spx{1},spx{2},spx{3}, spx{4}, bkg, fr);
end


function s = vrml_arrowAB(A, B, col)
% s = vrml_arrowAB(A, B, col)
% s = vrml plot of the arrow with endpoints A and B
    if nargin < 3
       col = [0 1 0];
    end
    s   = vrml_arrow([1 1/10 1/32 1/128],col);
    ey  = [0 1 0]';
    a   = B-A;
    R   = rotation_matrix(ey, a(:));
    scale = norm(a);
    s   = vrml_transfo(s,A,R,scale,'');
end


function s = vrml_textT(t, T)
% s = vrml_textT(t, T)
% s = vrml plot of the text t at endpoint T
    s  = vrml_text(t,'size',0.5);
    s  = vrml_transfo(s,T,eye(3),1,'');
end


function R = rotation_matrix(a, b)
% R = rotation_matrix(a, b)
% rotation matrix, which rotates the
% unit vector ea into the unit vector eb
% angle between vectors
    phi = acos(dot(a,b)/norm(a)/norm(b));
```

```
    n = cross(a,b);
    if norm(n)==0,
        R = eye(3);
    else
        n = n/norm(n); % rotation direction
        N = [0 -n(3) n(2); n(3) 0 -n(1); -n(2) n(1) 0];
        R = eye(3) + sin(phi)*N + (1-cos(phi))*N*N;
        R = R';
    end
end
```

E. Python

```
import numpy as np
#from enthought.mayavi import mlab
from mayavi import mlab
NA,NM, sqrt,sin = np.array,np.matrix, np.sqrt,np.sin
N = 100j
NN = 100

# sphere
def R(x, y, z):
    x,y,z = NA(x),NA(y),NA(z)
    return sqrt(x*x + y*y + z*z)

# tau_xieta
def tau(x, y, z, sig11,sig22,sig33):
    return R((sig11-sig22)*x*y, \
             (sig11-sig33)*x*z, \
             (sig22-sig33)*y*z)

# domain plot points
def domainplotpoints(xmax=2,ymax=2,zmax=2,n=N):
    x, y, z = np.ogrid[-xmax:xmax:n,\
                       -ymax:ymax:n,\
                       -zmax:ymax:n]
    return x,y,z

# plot unit sphere
def plotunitsphere(col=(0.0, 0.0, 1.0)):
    x,y,z = domainplotpoints()
    r     = R(x,y,z);
    src1  = mlab.pipeline.scalar_field(r)
    mlab.pipeline.iso_surface(src1, contours=[1, ],\
                              opacity=1,  color=col)
# plot sphere
def plotsphere((x0,y0,z0),r0,col=(0.0, 1.0, 0.0)):
    x,y,z = domainplotpoints()
    r     = R(x-x0,y-y0,z-z0);
    src   = mlab.pipeline.scalar_field(r)
    mlab.pipeline.iso_surface(src, contours=[r0, ], \
                              color=col)

# plot of the surface tau(x,y,z) = c
def plotconstau(sig11=1,sig22=2,sig33=3,c=1,\
                col=(1.0, 0.0, 0.0)):
    x,y,z = domainplotpoints()
    t     = tau(x,y,z, sig11,sig22,sig33);
    src   = mlab.pipeline.scalar_field(t)
    mlab.pipeline.iso_surface(src, contours=[c, ], \
                              opacity=1, color=col)

# plot of the disk
def posevnidisk((x,y,z),(x0,y0,z0),(a,b,c),r):
    a = abs(a*(x-x0)+b*(y-y0)+c*(z-z0))
    b = np.sign(R(x-x0,y-y0,z-z0)-r) + 1
    return a + b
def plotposevnidisk((x0,y0,z0),(a,b,c),r):
    x,y,z = domainplotpoints()
    pd    = posevnidisk((x,y,z),(x0,y0,z0),(a,b,c),r)
    src   = mlab.pipeline.scalar_field(pd)
    mlab.pipeline.iso_surface(src, contours=[0.01, ],\
                              opacity=0.8)

def centerline((x,y,z),(x0,y0,z0),d):
    a = abs(x*y0-y*x0) + abs(x*z0-z*x0) + abs(y*z0-z*y0)
    b = np.sign(R(x-x0,y-y0,z-z0)-d) + 1
    return  a + b
def plotcenterline((x0,y0,z0),d):
    # plot of the surface tau(x,y,z) = c
    x,y,z = domainplotpoints(2,2,2)
    pd = centerline((x,y,z),(x0,y0,z0),d)
    src = mlab.pipeline.scalar_field(pd)
    mlab.pipeline.iso_surface(src, contours=[0.04, ],\
                              opacity=0.8)

# plot pline
def pline((x,y,z),(x0,y0,z0),(x1,y1,z1)):
    a = (x-x0)/(x1-x0)
    b = (y-y0)/(y1-y0)
    c = (z-z0)/(z1-z0)
    f = abs(a-b) + abs(a-c) + abs(b-c) + abs(np.sign(a)-1)
    return  f
def plotpline((x0,y0,z0),(x1,y1,z1)):
    x,y,z = domainplotpoints()
    pd    = pline((x,y,z),(x0,y0,z0),(x1,y1,z1))
    src   = mlab.pipeline.scalar_field(pd)
    mlab.pipeline.iso_surface(src, contours=[0.02, ],\
                              opacity=0.8)

# plot cone
def cone((x,y,z),(x0,y0,z0),(x1,y1,z1),s=1):
    # distance between points T0(x0,y0,z0) and T1(x1,y1,z1)
    d = sqrt((x1-x0)**2 +(y1-y0)**2 +(z1-z0)**2)
    # unit vector between points T0 and T1
    nx,ny,nz = (x1-x0)/d,(y1-y0)/d,(z1-z0)/d
    # point T(x,y,z) distance to the plane
    # trough the point T0(x0,y0,z0)
    d1 = abs(nx*(x-x0)+ny*(y-y0)+nz*(z-z0))
    # point T''(xcc,ycc,zcc) is the projection
    # of the point T onto the plane
    xcc,ycc,zcc = x-d1*nx,y-d1*ny,z-d1*nz
    # distance d2
    d2 = R(xcc-x0,ycc-y0,zcc-z0)
    return d1 + 5 * s * d2 - d
def plotcone((x0,y0,z0),(x1,y1,z1),s=1):
    x,y,z = domainplotpoints()
    pd    = cone((x,y,z),(x0,y0,z0),(x1,y1,z1),s)
    src   = mlab.pipeline.scalar_field(pd)
    mlab.pipeline.iso_surface(src, contours=[0.02, ],\
                              opacity=0.8)

# plot arrow
def plotarrow((x0,y0,z0),(x1,y1,z1)):
    a = 0.35
    b = 1.0-a
    xm = a*x0 + b*x1
    ym = a*y0 + b*y1
    zm = a*z0 + b*z1
    d = R(xm-x0,ym-y0,zm-z0)
    plotcenterline((x0,y0,z0),d)
    plotcone((xm,ym,zm),(x1,y1,z1))

# plot text
def textcoor((x,y,z),xmax=2,ymax=2,zmax=2,n=NN):
    xmin,ymin,zmin = -xmax,-ymax,-zmax
    b = n/float(xmax-xmin); a = -b*xmin; xt = a+b*x;
    b = n/float(ymax-ymin); a = -b*ymin; yt = a+b*y;
    b = n/float(zmax-zmin); a = -b*zmin; zt = a+b*z;
    return (xt,yt,zt)
def plottext3d((a,b,c),t):
    x,y,z = domainplotpoints()
    mlab.text3d(a,b,c,t,scale=5)


def tauplot(sig11=1, sig22=2, sig33=3, isp=1):
    # call: is = tauplot(sig11, sig22, sig33, isp)
    # function plots the surface tau = taumax
    # with the unit ball (inserted)
    # input parameters:
    #    sig11, sig22, sig33 are the principal stresses
    #    isp is the index of the plot group
    #       of the stationary points

    # stationary shear stresses
    isp = isp-1
    s = sqrt(2)/2
    tausp = abs(NA([sig22-sig33, \
                    sig11-sig33, \
                    sig11-sig22]))/2.0
    taumax = max(abs(tausp))  # max shear stress
    if taumax == 0: # hydrostatic stress state
        return 0

    mlab.figure(bgcolor=(1,1,1),size=(1600,1200))
    # stationary points
    txtSP = ['I', 'II', 'III']
    txtG = ['1', '2', '3']
    SP = [[(0,s,s),(0,-s,s),(0,s,-s),(0,-s,-s)],\
          [(s,0,s),(-s,0,s),(s,0,-s),(-s,0,-s)],\
          [(s,s,0),(-s,s,0),(s,-s,0),(-s,-s,0)]]
    E = [(1,0,0),(0,1,0),(0,0,1)]
```

```
    G = [(2.0,0,0),(0,2.0,0),(0,0,2.0)]
    plotunitsphere()
    plotconstau(sig11,sig22,sig33, tausp[isp])
    for sp in SP[isp]:
        plotsphere(sp,0.1)
        plottext3d(textcoor(tuple(1.2*NA(sp))),\
                   txtSP[isp])
        plotarrow(sp,tuple(2*NA(sp)))
    for (e,g,txt) in zip(E,G,txtG):
        plottext3d(textcoor(tuple(1.2*NA(g))),txt)
        plotarrow(e,g)

#     mlab.show()
    mlab.draw()
    name = ['Fig1.png', 'Fig2.png', 'Fig3.png' ]
    mlab.savefig(name[isp])
    mlab.close()
    return 1
```

ACKNOWLEDGMENT

The work was partially supported by the Slovenian Research Agency through the grant P2-0260. The support is gratefully acknowledged.

REFERENCES

[1] Y. C. Fung, *A First Course in Continuum Mechanics*. Englewood Cliffs, New Jersey: Prentice-Hall, 1965.
[2] G. E. Mase, *Continuum Mechanics*. New York: McGraw-Hill, 1970.
[3] J. Brnić, *Elastomehanika i Plastomehanika (Elasticity and Plasticity)*. Zagreb: Školska knjiga, 1996.
[4] Y. C. Fung and P. Tong, *Classical and Computational Continuum Mechanics*. Singapore: World Scientific Publishing Co. Pte. Ltd., 2001.
[5] S. Srpčič, *Mehanika Trdnih Teles (Mechanics of Solids)*. Ljubljana: Univerza v Ljubljani, Fakulteta za gradbeništvo in geodezijo, 2004.
[6] M. Stanek and G. Turk, *Osnove Mehanike Trdnih Teles (An Introduction to the Mechanics of Solids)*. Ljubljana: Univerza v Ljubljani, Fakulteta za gradbeništvo in geodezijo, 1998.
[7] B. Štok, *Mehanika Deformabilnih Teles, Zbirka Rešenih Problemov, I. in II. del (Continuum Mechanics, Problems with Solutions, I. and II. part)*. Ljubljana: Univerza v Ljubljani, Fakulteta za strojništvo, 1988.
[8] S. Jensen. (2004) An Introduction to Method of Lagrange Multipliers. [Online]. Available: http://www.slimy.com/ steuard/teaching/tutorials/Lagrange.html
[9] T. Williams, C. Kelley, R. Lang, D. Kotz, J. Campbell, G. Elber, and A. Woo. (1986) Gnuplot. [Online]. Available: http://www.gnuplot.info/
[10] W. Stein. (2005, February) Sage. [Online]. Available: http://www.sagemath.org/
[11] R. Ihaka and R. Gentleman. (1993) R. [Online]. Available: http://www.r-project.org/
[12] G. van Rossum. (1991) Python. [Online]. Available: http://www.python.org/
[13] J. Eaton, W. (1988) Octave. [Online]. Available: http://www.gnu.org/software/octave/
[14] L. Wei-Pin, C. Chang-Hsuan, H. Chung-Li, and M. John, “Digital simulation of the transformation of plane stress,” *Computer Applications in Engineering Education*, vol. 17, no. 1, pp. pp. 25–33, 2009.
[15] E. Grossmann. (2012) vrml (octave package). [Online]. Available: http://octave.sourceforge.net/vrml/overview.html
[16] W. Consortium. (1994, November) Vrml. [Online]. Available: http://www.web3d.org/x3d/vrml/
[17] M. Kamburelis. view3dscene. [Online]. Available: http://castle-engine.sourceforge.net/view3dscene.php
[18] T. J. Lukka and J. Stewart. (1998) Freewrl. [Online]. Available: http://freewrl.sourceforge.net/