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 symbolic 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 graphical tools like Sage.

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

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 stress can be obtained by Pitagora theorem

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

Using the Lagrange identity

$$\langle \vec{a} \times \vec{b} \rangle \cdot (\vec{e} \times \vec{d}) = (\vec{a} \cdot \vec{e}) (\vec{b} \cdot \vec{d}) - (\vec{b} \cdot \vec{e}) (\vec{a} \cdot \vec{d})$$

we get

$$\sigma_{nt}^2 = \vec{\sigma}_n \cdot \vec{\sigma}_n - (\vec{\sigma}_n \cdot \vec{e}_n)^2 = (\vec{\sigma}_n \cdot \vec{e}_n) (\vec{\sigma}_n \cdot \vec{e}_n) - (\vec{\sigma}_n \cdot \vec{e}_n)^2 = (\vec{\sigma}_n \times \vec{e}_n) \cdot (\vec{\sigma}_n \times \vec{e}_n) = \| \vec{\sigma}_n \times \vec{e}_n \|^2. \quad (2)$$

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 \(\sigma_n\) can be expressed by the linear combination

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

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

$$\vec{\sigma}_n \times \vec{e}_n = \left| \begin{array}{ccc} \vec{e}_1 & \vec{e}_2 & \vec{e}_3 \\ \sigma_{11} e_{11} & \sigma_{22} e_{22} & \sigma_{33} e_{33} \\ e_{11} & e_{22} & e_{33} \end{array} \right|$$

and obtain

$$\sigma_{nt}^2 = (\sigma_{11} - \sigma_{22})^2 e_{11}^2 e_{22}^2 + (\sigma_{33} - \sigma_{11})^2 e_{33}^2 e_{11}^2 + (\sigma_{22} - \sigma_{33})^2 e_{22}^2 e_{33}^2. \quad (3)$$

It is worth to note that Eq. (3) presents the component form of the equation \(\| \vec{\sigma}_n \times \vec{e}_n \| \) 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 \(e_{11}, e_{22}, e_{33}\) 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 \(g(e_{11}, e_{22}, e_{33}) = e_{11}^2 + e_{22}^2 + e_{33}^2 - 1 = 0\). Using the Method of Lagrangian Multipliers [8] we can find the stationary points of the Lagrangian function \(L(e_{11}, e_{22}, e_{33}, \lambda) = \sigma_{nt}^2(e_{11}, e_{22}, e_{33}) + \lambda(g(e_{11}, e_{22}, e_{33}) - 1)\). 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:

$$H = \begin{bmatrix}
\frac{\partial^2 L}{\partial x^2} & \frac{\partial^2 L}{\partial x \partial y} & \frac{\partial^2 L}{\partial x \partial z} \\
\frac{\partial^2 L}{\partial y \partial x} & \frac{\partial^2 L}{\partial y^2} & \frac{\partial^2 L}{\partial y \partial z} \\
\frac{\partial^2 L}{\partial z \partial x} & \frac{\partial^2 L}{\partial z \partial y} & \frac{\partial^2 L}{\partial z^2}
\end{bmatrix}$$

(4)

at stationary points

$$(e_{n1}, e_{n2}, e_{n3}), \lambda = \begin{cases}
\pm \frac{\sqrt{2}}{2}, \pm \frac{\sqrt{2}}{2}, 0, -\sigma_{11} \sigma_{22}, \\
0, \pm \frac{\sqrt{2}}{2}, \pm \frac{\sqrt{2}}{2}, -\sigma_{11} \sigma_{33}, \\
0, \pm \frac{\sqrt{2}}{2}, \pm \frac{\sqrt{2}}{2}, -\sigma_{22} \sigma_{33}
\end{cases},$$

(5a, 5b, 5c)

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

```
# function to minimize
def fun(sig11,sig22,sig33,x,y,z,lambda):
    sn1 = sig11 * x^2 + sig22 * y^2 + sig33 * z^2
    sn2 = sig11 * x^2 + sig22 * y^2 + sig33 * z^2
    phi = 1 - x^2 - y^2 - z^2
    sn2 = sn2 - snn^2
    F = sn2 + lambda * 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)

# symbolic Hessian
def Hessian(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('sig11,sig22,sig33')
x,y,z,lambda = var('x,y,z,lambda')
F = fun(sig11,sig22,sig33,x,y,z,lambda)

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

# stationary points analysis
n = len(S)
for (k,s) in zip(range(1,n+1),S):
    # eigenvectors of Hessian matrix
    l = h.eigenvalues()
    # simplify
    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 Lagrangian matrix at those point, respectively:

1. $[y: 1/2*sqrt(2), lam: -s11*s22, x: 1/2*sqrt(2), z: 0]$ 1
2. $[y: 1, lam: -s22^2, x: 0, z: 0]$ 2
3. $[y: 0, lam: -s11*s33, x: -1/2*sqrt(2), z: 1/2*sqrt(2)]$ 3
4. $[y: 1, lam: -s11*s22, x: 1/2*sqrt(2), z: 0]$ 4
5. $[y: -1/2*sqrt(2), lam: -s11*s33, x: -1/2*sqrt(2), z: 0]$ 5
6. $[y: 1/2*sqrt(2), lam: -s11*s22, x: 1/2*sqrt(2), z: 0]$ 6
7. $[y: 0, lam: -s11*s22, x: 0, z: 0]$ 7
8. $[y: 1/2*sqrt(2), lam: -s11*s22, x: 1/2*sqrt(2), z: 0]$ 8
9. $[y: 0, lam: -s33^2, x: 0, z: -1]$ 9
10. $[y: 0, lam: -s33^2, x: 0, z: 1]$ 10
11. $[y: 0, lam: -s11*s33, x: 0, z: 1/2*sqrt(2)] 11$
12. $[y: 0, lam: -s11*s22, x: 1/2*sqrt(2), z: 0]$ 12

We can assume  
\[ \sigma_{11} \leq \sigma_{22} \leq \sigma_{33}. \]  

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 all the 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  
\[ f(x, y) = \begin{cases} x^2 - (2y)^2, & y \geq 0 \\ x^2 + (2y)^2, & y \leq 0 \end{cases} \]  

fulfilling the condition  
\[ g(x, y) = x^2 + y^2 - 1 = 0. \]

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 points the function gradients \( \nabla f \) and \( \nabla g \), in Fig 2 denoted by red arrows, coincide, what in fact confirms the existence of constant \( \lambda \) 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  
\[ \sigma_{nt}^2 = (\sigma_{11} - \sigma_{22})^2 e_{n1}^2 e_{n2}^2 + (\sigma_{33} - \sigma_{11})^2 e_{n1}^2 e_{n3}^2 + (\sigma_{22} - \sigma_{33})^2 e_{n2}^2 e_{n3}^2 = (\sigma_{11} - \sigma_{33})^2 \left( e_{n1}^2 + e_{n2}^2 \right) e_{n3}^2. \]  

Consequently we get  
\[ \sigma_{nt} = \pm (\sigma_{11} - \sigma_{33}) e_{n3} \sqrt{e_{n1}^2 + e_{n2}^2}. \]  

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 \( \nabla f \) and \( \nabla g \) coincide. The gradients are perpendicular to the curves \( g = 0 \) and \( f = \tau_{\text{max}} \). All stationary points are Extrema, since the unit circle (graph of the function \( g = 0 \)) only touches (not intersects) the graph of the function \( f = \tau_{\text{max}} \) at these points.
is an extrema, since the unit ball touches 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.

In the 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 \( \left( \pm \sqrt{\frac{\sigma_{11}}{2}}, \pm \sqrt{\frac{\sigma_{22}}{2}}, 0 \right) \), \( \left( \pm \sqrt{\frac{\sigma_{22}}{2}}, 0, \pm \sqrt{\frac{\sigma_{33}}{2}} \right) \), \( (0, \pm \sqrt{\frac{\sigma_{11}}{2}}, \pm \sqrt{\frac{\sigma_{33}}{2}}) \). 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:

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

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

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

Since the unit sphere intersects the graph of the functions (11a) and (11c), the corresponding stationary points \( \left( \pm \sqrt{\frac{\sigma_{11}}{2}}, \pm \sqrt{\frac{\sigma_{33}}{2}}, 0 \right) \) and \( \left( 0, \pm \sqrt{\frac{\sigma_{22}}{2}}, \pm \sqrt{\frac{\sigma_{33}}{2}} \right) \) are saddle points. In the second case the unit sphere touches the graph of the function (11b), so the corresponding stationary points \( \left( \pm \sqrt{\frac{\sigma_{11}}{2}}, 0, \pm \sqrt{\frac{\sigma_{33}}{2}} \right) \), 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 \( f = \tau_{\text{max}} \) 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.

\[\text{IV. CONCLUSION}\]

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 Octave graphics, which is well suitable for two dimensional presentation, but less suitable for three dimensional plots. In order to avoid this inconvenience, the Octave vrm1 package [15] was included. Applying this package, high quality three dimensional graphics plot can be constructed in standardized vrm1 or wrl format [16], by using well known programming tools like view3dscene [17] or FreeWRL [18].

A. Gnuplot 4.4

# notation according to article: x = r, y = z
reset
# principal stresses
sig1 = 1.0 # sig2 = 1.0
sig3 = 4.0
tau_max = abs(sig3 - sig1)/2 # extreme shear stress
tau(y, z) = sqrt((sig1-sig3)**2 * y**2 * z**2) - tau_max

set terminal wxt
set size square
set view map
plot "stationarypoints.dat" u 1:2wpp t7p s3l c2 notitle,
unset table
splot tau(x,y,sig1,sig3)
set table "tau.dat"
# contour plot of function tau
set cntrparam levels discrete 0
set isosample 100, 100
set xrange [-2:2]
set contour
# plot of the unit circle
circle(x, y) = sqrt(x**2+y **2 )-1
taumax = abs(sig3 - sig1)/2 # extreme shear stress
sig1 = 1.0 # sig2 = 1.0
# notation according to article:x=r ,y=z

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 Octave graphics, which is well suitable for two dimensional presentation, but less suitable for three dimensional plots. In order to avoid this inconvenience, the Octave vrm1 package [15] was included. Applying this package, high quality three dimensional graphics plot can be constructed in standardized vrm1 or wrl format [16], by using well known programming tools like view3dscene [17] or FreeWRL [18].

A. Gnuplot 4.4

# notation according to article: x = r, y = z
reset
# principal stresses
sig1 = 1.0 # sig2 = 1.0
sig3 = 4.0
tau_max = abs(sig3 - sig1)/2 # extreme shear stress
tau(y, z) = sqrt((sig1-sig3)**2 * y**2 * z**2) - tau_max

set terminal wxt
set size square
set view map
plot "stationarypoints.dat" u 1:2wpp t7p s3l c2 notitle,
unset table
splot tau(x,y,sig1,sig3)
set table "tau.dat"
# contour plot of function tau
set cntrparam levels discrete 0
set isosample 100, 100
set xrange [-2:2]
set contour
# plot of the unit circle
circle(x, y) = sqrt(x**2+y **2 )-1
taumax = abs(sig3 - sig1)/2 # extreme shear stress
sig1 = 1.0 # sig2 = 1.0
# notation according to article:x=r ,y=z

B. Sage 4.6.2

def ball(x,y,z):
    return sqrt((x+y+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):

C. R 2.10.1

# source("tauplot.R"); r = tauplot(1,2,4,1)
library(rgl) # base 3d graphics library
library(misc3d) # miscellaneous 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'){
    for i in [0..3]
        points
        plots + stationary points + coor system
return P
plotStationaryPoints = function(spat){
  plot3d(spat$x, spat$y, spat$z, size=2, type='s', add=T)
}

plotStationaryArrows = function(spat){
  d = 1+(0:100/100)
  for (i in 1:4)
    text3d(spat$x[i]*d, spat$y[i]*d, spat$z[i]*d, size=0.6, type='s', add=T)
}

plotStationaryText = function(spat){
  txt = {'1','2','3'}
  egx = {}
  eg = {}
  for i = 1:4
    egx{i} = vrml_textT(txt{i},b{i})
    eg{i} = vrml_arrowAB(a,b{i})
  end

tauplot = function(sig1=1,sig2=2,sig3=3,isp=1){
  % stationary shear stresses
  % stationary (extreme) shear stresses
  taust = sig1*x.*y + sig2*y.*z + sig3*z.*x;
  switch isp
    case 1
      spt = list(s1, s2, s3, s4, sp, fr, bkg, egx, eg)
      s = vrml_plot([s1, s2, s3, s4, ...], [sp, fr, bkg, egx, eg])
    case 2
      spt = list(s1, s2, s3, s4, sp, fr, bkg, egx, eg)
      s = vrml_plot([s1, s2, s3, s4, ...], [sp, fr, bkg, egx, eg])
    case 3
      spt = list(s1, s2, s3, s4, sp, fr, bkg, egx, eg)
      s = vrml_plot([s1, s2, s3, s4, ...], [sp, fr, bkg, egx, eg])
    end

  psi = 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, ...)
  s = vrml_plot([s1, s2, s3, s4, ...], [sp, fr, bkg, egx, eg])
}

Octave 3.2.4

D. Octave 3.2.4

function is = tauplot2(sig11, sig22, sig33, isp)
  % call: tauplot(sig11, sig22, sig33, isp)
  % function plots the surface tau = const
  % input parameters:
  % sig1, sig2, sig3 are the principal stresses
  % isp is the index of the plot group of
  % the stationary points
  if nargin < 3
    sig1 = 1; sig2 = 2; sig3 = 3;
  end
  sig = [sig1 sig2 sig3];
  if nargin < 4
    isp = 1;
  end

  % ball
  R = @(x,y,z) sqrt(x.^2 + y.^2 + z.^2);
  tau = @(x,y,z, sig)
  R(sig11-sig33)*x.*y, ...
    (sig11-sig22)*y.*z);
  % stationary (extreme) shear stresses
  % stationary shear stresses
  taumax = max(abs(tausp));
  if taumax == 0,
    error('Hidrostatic stress state.');
  end

  switch isp
    case 1
      spt = list(s1, s2, s3, s4, sp, fr, bkg, egx, eg)
      s = vrml_plot([s1, s2, s3, s4, ...], [sp, fr, bkg, egx, eg])
    case 2
      spt = list(s1, s2, s3, s4, sp, fr, bkg, egx, eg)
      s = vrml_plot([s1, s2, s3, s4, ...], [sp, fr, bkg, egx, eg])
    case 3
      spt = list(s1, s2, s3, s4, sp, fr, bkg, egx, eg)
      s = vrml_plot([s1, s2, s3, s4, ...], [sp, fr, bkg, egx, eg])
    end

  psi = 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, ...)
  s = vrml_plot([s1, s2, s3, s4, ...], [sp, fr, bkg, egx, eg])
}

function s = vrml_arrowAB(A, B, col)
  % s = vrml plot of the arrow with endpoints A and B
  % if nargin < 3
  %   col = [0 0 1];
  % end
  % s = vrml_arrow([A 0 0 B],col)
def plotcenterline((x0,y0,z0),d):
    """plot of the centerline"""
    return d

def centerline((x,y,z),(x0,y0,z0),d):
    """centerline of a disk"""
    return d

def plotposevnidisk((x0,y0,z0),(a,b,c),r):
    """plot of the disk"""
    pd = posevnidisk((x,y,z),(x0,y0,z0),(a,b,c),r)
    return pd

def posevnidisk((x,y,z),(x0,y0,z0),(a,b,c),r):
    """plot of the disk"""
    return r

def plotsphere((x0,y0,z0),r0,col=(0.0, 1.0, 0.0)):
    """plot sphere"""
    src = mlab.pipeline.scalar_field(r0)
    return src

def plotconstau(sig11=1,sig22=2,sig33=3,c=1,
                isp=1):
    """plot of the disk"""
    return c

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

def RA(x,y,z):
    """sphere"""
    return x,y,z

def domainplotpoints(xmax=2,ymax=2,zmax=2,n=N):
    """domain plot points"""
    return x,y,z

def R(x, y, z):
    """sphere"""
    return R((sig11-sig22)*x*y, 
             (sig11-sig33)*x*z, 
             (sig22-sig33)*y*z)

def plotpline((x0,y0,z0),(x1,y1,z1)):
    """plot pline"""
    return f

def pline((x,y,z),(x0,y0,z0),(x1,y1,z1)):
    """plot pline"""
    return f

def textcoor((x,y,z),xmax=2,ymax=2,zmax=2,n=NN):
    """plot text"""
    return a,b,c

def plottext3d((a,b,c),t):
    """plot text"""
    return a,b,c

def plotarrow((x0,y0,z0),(x1,y1,z1)):
    """plot arrow"""
    return a,b,c

def plotcone((x0,y0,z0),(x1,y1,z1),s=1):
    """plot cone"""
    return a,b,c
def cone((x,y,z),(x0,y0,z0),(x1,y1,z1),s=1):  # plot cone
    """plot cone"""
    return a,b,c

def pline((x0,y0,z0),(x1,y1,z1)):
    """plot pline"""
    return f

def plottext3d((a,b,c),t):
    """plot text"""
    return a,b,c

def plotarrow((x0,y0,z0),(x1,y1,z1)):
    """plot arrow"""
    return a,b,c

def plotcone((x0,y0,z0),(x1,y1,z1),s=1):
    """plot cone"""
    return a,b,c
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