ME 562 - Assignment 2 - Code Output - Brandon Lampe

In [30]:
```
# from __future__ import division, print_function
import sys
from scipy import linalg as LA
import math
import numpy as np
import blfunc as bl
import scipy.optimize as optimize
import numdifftools as nd
from IPython.display import display

np.set_printoptions(precision= 2, suppress = True)
# %precision 2
```

Problem 1

Program that provides the matrix of elasticity coefficients -> Elastic()

In [31]:
```
def Elastic(Y_1, Y_2, Y_3, nu_12, nu_23, nu_31, G_44, G_55, G_66):
    r"""Function calculates the 6x6 Elasticity matrix in V-M notation in a given basis.
    Function takes engineering moduli (Young's Modulus - Y, Poission's Ratio - nu, and Shear Modulus - G).
    Produces results for an orthropic material, which has 9 independent elastic parameters.  To return elastic matrix for
    with a greater level of symmetry (e.g., tetragonal, cubic, isotropic, etc.), the input parameters must be modified
    accordingly.
    """

    EngMod = np.array([Y_1, Y_2, Y_3, nu_12, nu_23, nu_31, G_44, G_55, G_66])

    Y  = np.array([EngMod[0], EngMod[1], EngMod[2]])  #Young's Moduli: Y_1, Y_2, Y_3,
    nu = np.array([EngMod[3], EngMod[4], EngMod[5]]) #Poisson's Ratios: nu_12, nu_23, nu_31,
    G  = np.array([EngMod[6], EngMod[7], EngMod[8]])  #Shear Moduli: G_44, G_55, G_66

    # Check Values of Engineering Moduli
    import sys
    for i in range(0,3):
        if nu[i] >= 0.5:
            sys.exit("BAD VALUE: nu value greater than 0.5")
        if nu[i] <= -1:
            sys.exit("BAD VALUE: nu value less than -1.0")
        if Y[i] <= 0.0:
            sys.exit("BAD VALUE: Young's Modulus must be > 0")
        if G[i] < 0:
            sys.exit("BAD VALUE: Shear Modulus must be > 0")

    F = np.zeros((6,6)) #flexibility matrix (inverse of E)
    k = np.array([1, 2, 0])
    m = np.array([3, 4, 5])

    # not vectorized
    for i in range(0,3):
        F[i, i] = 1 / Y[i] # main diagonal components of upper left 3x3 quad in F
        F[i, k[i]] = - nu[i] / Y[k[i]] # off diagonal components of upper left 3x3 quad in F
        F[k[i], i] = - nu[i] / Y[i] # off diagonal components of upper left 3x3 quad in F
        F[m[i], m[i]] = 1 / (2 * G[i]) # main diagonal components of Lower Right 3X3 quad in F

    E = LA.inv(F)
#     print(F) # check
    return(E)
```

Verification of Elastic() using completely isotropic parameters are input. Results were checked with hand calculations.

In [32]:
```
Y = 10
nu = 0.25
G = Y/(1+nu)
print("The Stiffness Matrix:")
print(Elastic(Y, Y, Y, nu, nu, nu, G, G, G))
```

```
The Stiffness Matrix:
[[ 12.   4.   4.   0.   0.  -0.]
 [  4.  12.   4.   0.   0.  -0.]
 [  4.   4.  12.   0.   0.  -0.]
 [  0.   0.   0.  16.   0.  -0.]
 [  0.   0.   0.   0.  16.  -0.]
 [  0.   0.   0.   0.   0.  16.]]
```

Problem 3

(i) Calculate the stiffness and flexibility matrices for NaCl

In [33]:
```
# values from Roberston et al.
Y = 30.8 # GPa
nu = 0.347
G = 11.3 # GPa
E_EE = Elastic(Y, Y, Y, nu, nu, nu, G, G, G)
print("The Stiffness Matrix:")
print(E_EE)
```

```
The Stiffness Matrix:
[[ 48.79  25.93  25.93   0.     0.    -0.  ]
 [ 25.93  48.79  25.93   0.     0.    -0.  ]
 [ 25.93  25.93  48.79   0.     0.    -0.  ]
 [  0.     0.     0.    22.6    0.    -0.  ]
 [  0.     0.     0.     0.    22.6   -0.  ]
 [  0.     0.     0.     0.     0.    22.6 ]]
```

In [34]:
```
F = LA.inv(E_EE)
print("The Flexibility Matrix:")
print(F)
```

```
The Flexibility Matrix:
[[ 0.03 -0.01 -0.01  0.    0.   -0.  ]
 [-0.01  0.03 -0.01  0.    0.   -0.  ]
 [-0.01 -0.01  0.03  0.    0.   -0.  ]
 [ 0.    0.    0.    0.04  0.   -0.  ]
 [ 0.    0.    0.    0.    0.04 -0.  ]
 [ 0.    0.    0.    0.    0.    0.04]]
```

Alternate calculation using values from a different source.

In [35]:
```
# values from Meyers and Chawla
Y_2 = 32.5 # GPa
nu_2 = 0.25
G_2 = 13 # GPa
E_2 = Elastic(Y_2, Y_2, Y_2, nu_2, nu_2, nu_2, G_2, G_2, G_2)
print("The Stiffness Matrix:")
print(E_2)
```

```
The Stiffness Matrix:
[[ 39.  13.  13.   0.   0.  -0.]
 [ 13.  39.  13.   0.   0.  -0.]
 [ 13.  13.  39.   0.   0.  -0.]
 [  0.   0.   0.  26.   0.  -0.]
 [  0.   0.   0.   0.  26.  -0.]
 [  0.   0.   0.   0.   0.  26.]]
```

(ii) Function to be minimized (R)

In [36]:
```
def resid_IsoFit(x0,E_aniso):
    """ Function to be minimized when approximating an anisotropic stiffness matrix with an isotropic stiffness matrix.
    x0 = [Y_inital, nu_initial]
    E_aniso = 6x6 anisotropic stiffness matrix to be approximated
    """

    E_iso = Elastic(x0[0], x0[0], x0[0], x0[1], x0[1], x0[1], x0[0]/(1+x0[1]), x0[0]/(1+x0[1]), x0[0]/(1+x0[1]))
    numerator = np.dot(E_aniso - E_iso, E_aniso - E_iso)
    return np.trace(numerator) / np.trace(E_aniso.dot(E_aniso))
```

Call "resid_IsoFit" to determine the best isotropic approximation of E_EE

In [37]:
```
# initial guess
Y_iso = 1 #GPa
nu_iso = 0.25

InitialGuess = [Y_iso, nu_iso] # initial guess for paramters
bnds = ((0, None), (-.99, 0.49)) # bounds on Y and nu

# run minimize function
result = optimize.minimize(resid_IsoFit, InitialGuess, args = (E_EE,), bounds = bnds, options={'xtol':1e-4, 'disp':True})
print("Results from fitting:")
print result
```

```
Results from fitting:
  status: 0
 success: True
    nfev: 80
     fun: 0.01804291894308626
       x: array([ 18.25,   0.41])
 message: 'CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL'
     jac: array([-0., -0.])
     nit: 19
```

Best Estimate for E_iso

In [38]:
```
Y_iso, nu_iso = result.x

Y = Y_iso # GPa
nu = nu_iso
G = Y/(1+nu)
print("Isotropic Approximation To The Stiffness Matrix For NaCl:")
print(Elastic(Y, Y, Y, nu, nu, nu, G, G, G))
```

```
Isotropic Approximation To The Stiffness Matrix For NaCl:
[[ 42.19  29.23  29.23   0.     0.    -0.  ]
 [ 29.23  42.19  29.23   0.     0.    -0.  ]
 [ 29.23  29.23  42.19   0.     0.    -0.  ]
 [  0.     0.     0.    25.9    0.    -0.  ]
 [  0.     0.     0.     0.    25.9   -0.  ]
 [  0.     0.     0.     0.     0.    25.9 ]]
```

The Residual, difference between E and E_iso (normalized differece)

In [51]:
```
E_scalar = np.trace(E_EE)
B = Y_iso / (3 * (1 - 2 * nu_iso)) # bulk modulus
G = (Y_iso/(1 + nu_iso))/2 # shear modulus
R = (1 / E_scalar**2) * (E_scalar**2 - 3 * (3 * B)**2 - 5 * (2 *G)**2)
print("The residual:")
display(R)
```

```
The residual:


0.32
```

Problem 4

(i) Stiffness matrix in an arbitrarily orientated (wrt orthotropic axes) orthonormal basis (e_i)

The transformation matrix between e-e and E-E (a_eE)

In [52]:
```
a_eE = bl.orth_basis() #arbitrarily orientated orthonormal basis
a_Ee = np.transpose(a_eE)
check = a_eE.dot(a_Ee)
print ("a_eE:"); print(a_eE);
print ("Check for Orthonormal:")
print(check);
```

```
a_eE:
[[ 0.8   0.32  0.51]
 [ 0.26  0.58 -0.77]
 [-0.55  0.75  0.37]]
Check for Orthonormal:
[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
```

The transformation matrix for a fourth orther tesnor in V-M notation (A_eE)

In [53]:
```
A_eE = bl.tran_a_A(a_eE)
A_Ee = np.transpose(A_eE)
print("A_eE:"); print(A_eE);
```

```
A_eE:
[[ 0.63  0.11  0.26  0.37  0.23  0.57]
 [ 0.07  0.33  0.6   0.21 -0.63 -0.29]
 [ 0.3   0.56  0.14 -0.58  0.4  -0.29]
 [ 0.29  0.26 -0.56  0.54  0.04 -0.48]
 [-0.2   0.61 -0.41 -0.12 -0.37  0.52]
 [-0.61  0.34  0.27  0.42  0.5   0.02]]
```

Elasticity componenets for the istropic (cubic) material NaCl when in a basis that is not aligned with planes of symmetry

In [54]:
```
E_ee = A_eE.dot(E_EE).dot(A_Ee)
print("E_ee:"); print(E_ee)
check = A_Ee.dot(E_ee).dot(A_eE)
print("Check with transformation back to EE basis:"); print(check);
```

```
E_ee:
[[ 48.66  25.99  26.     0.02  -0.05  -0.08]
 [ 25.99  48.66  26.01  -0.06  -0.02   0.06]
 [ 26.    26.01  48.64   0.04   0.06   0.01]
 [  0.02  -0.06   0.04  22.72   0.09  -0.06]
 [ -0.05  -0.02   0.06   0.09  22.75   0.06]
 [ -0.08   0.06   0.01  -0.06   0.06  22.75]]
Check with transformation back to EE basis:
[[ 48.79  25.93  25.93   0.     0.     0.  ]
 [ 25.93  48.79  25.93   0.     0.     0.  ]
 [ 25.93  25.93  48.79   0.     0.     0.  ]
 [  0.     0.     0.    22.6    0.     0.  ]
 [  0.     0.     0.    -0.    22.6    0.  ]
 [  0.    -0.     0.     0.     0.    22.6 ]]
```

Call "resid_IsoFit" to determine the best isotropic approximation of E_ee

In [55]:
```
# initial guess
Y_iso = 1 #GPa
nu_iso = 0.25

InitialGuess = [Y_iso, nu_iso] # initial guess for paramters
bnds = ((0, None), (-.99, 0.49)) # bounds on Y and nu

# run minimize function
result = optimize.minimize(resid_IsoFit, InitialGuess, args = (E_ee,), bounds = bnds, options={'xtol':1e-8, 'disp':True})
print("Results from fitting:"); print(result);
```

```
Results from fitting:
  status: 0
 success: True
    nfev: 80
     fun: 0.017165106558853709
       x: array([ 18.29,   0.41])
 message: 'CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL'
     jac: array([ 0., -0.])
     nit: 19
```

Elastic parameters from isotropic approximation

In [56]:
```
Y_iso, nu_iso = result.x
print("{Y_iso, nu_iso}:"), display(Y_iso, nu_iso);

B = Y_iso / (3 * (1 - 2 * nu_iso)) # bulk modulus
G = (Y_iso/(1 + nu_iso))/2 # shear modulus
print("{Bulk Moduli, Shear Moduli}:"); display(B, G);
```

```
{Y_iso, nu_iso}:
18.29
0.41
 None
{Bulk Moduli, Shear Moduli}:

33.55
6.49
```

In [57]:
```
Y_iso, nu_iso = result.x

Y = Y_iso # GPa
nu = nu_iso
G = Y/(1+nu)
print("Isotropic Approximation To The Stiffness Matrix For NaCl:")
print(Elastic(Y, Y, Y, nu, nu, nu, G, G, G))
```

```
Isotropic Approximation To The Stiffness Matrix For NaCl:
[[ 42.21  29.22  29.22   0.     0.    -0.  ]
 [ 29.22  42.21  29.22   0.     0.    -0.  ]
 [ 29.22  29.22  42.21   0.     0.    -0.  ]
 [  0.     0.     0.    25.97   0.    -0.  ]
 [  0.     0.     0.     0.    25.97  -0.  ]
 [  0.     0.     0.     0.     0.    25.97]]
```

Problem 5. Find Material Axes

(i) Solve the eigenproblem for E_ee, which is the arbitrarily oriented stiffness matrix of an orthotropic material.

In [61]:
```
eigval, eigvec = LA.eig(E_ee)
eig1_E, eig2_E, eig3_E, eig4_E, eig5_E, eig6_E = eigval
A_P_e = np.transpose(eigvec)
print("eigenvalues of the E tensor:"); print(eigval);
print("6 eigenvectors of the E tensor in the e-e-e-e basis (each vector is one row):"); print(A_P_e);
```

```
eigenvalues of the E tensor:
[ 100.65+0.j   22.60+0.j   22.60+0.j   22.87+0.j   22.87+0.j   22.60+0.j]
6 eigenvectors of the E tensor in the e-e-e-e basis (each vector is one row):
[[-0.58 -0.58 -0.58  0.    0.   -0.  ]
 [ 0.72 -0.33 -0.39 -0.1   0.24  0.39]
 [-0.49  0.7  -0.2   0.29 -0.01 -0.39]
 [-0.38  0.26  0.13 -0.18  0.44  0.74]
 [ 0.08  0.21 -0.29 -0.61 -0.7   0.01]
 [ 0.02 -0.09  0.07  0.61 -0.58  0.52]]
```

This set of eigenvectors does not form an orthonormal basis, as [A][A]^T not equal [I]

In [62]:
```
A_P_e.dot(np.transpose(A_P_e))
```

Out[62]:
```
array([[ 1.  , -0.  , -0.  ,  0.  , -0.  ,  0.  ],
       [-0.  ,  1.  , -0.68,  0.  ,  0.  ,  0.02],
       [-0.  , -0.68,  1.  ,  0.  , -0.  , -0.1 ],
       [ 0.  ,  0.  ,  0.  ,  1.  , -0.21,  0.  ],
       [-0.  ,  0.  , -0.  , -0.21,  1.  , -0.  ],
       [ 0.  ,  0.02, -0.1 ,  0.  , -0.  ,  1.  ]])
```

In [63]:
```
vec1_E_eeee, vec2_E_eeee, vec3_E_eeee, vec4_E_eeee, vec5_E_eeee, vec6_E_eeee = eigvec[:,0],eigvec[:,1],eigvec[:,2],eigvec[:,3]
eigvec[:,4],eigvec[:,5],
```

(ii) Transfrom vectors from V-M notation {6x1} to typical tensorial notation [3x3]

In [64]:
```
#obtain six eigentensors (3x3)
a_P1_e_3x3 = bl.tran_vm_3x3(vec1_E_eeee)
a_P2_e_3x3 = bl.tran_vm_3x3(vec2_E_eeee)
a_P3_e_3x3 = bl.tran_vm_3x3(vec3_E_eeee)
a_P4_e_3x3 = bl.tran_vm_3x3(vec4_E_eeee)
a_P5_e_3x3 = bl.tran_vm_3x3(vec5_E_eeee)
a_P6_e_3x3 = bl.tran_vm_3x3(vec6_E_eeee)
```

(iii) Find 18 eigenvectors of 6 eigenbases assoicated with the Stiffness Tensor in the e-e basis, these are the material basis

In [67]:
```
eig1_ee, vec1_eeT = LA.eig(a_P1_e_3x3)
eig2_ee, vec2_eeT = LA.eig(a_P2_e_3x3)
eig3_ee, vec3_eeT = LA.eig(a_P3_e_3x3)
eig4_ee, vec4_eeT = LA.eig(a_P4_e_3x3)
eig5_ee, vec5_eeT = LA.eig(a_P5_e_3x3)
eig6_ee, vec6_eeT = LA.eig(a_P6_e_3x3)
```

In [68]:
```
a_M1_e = np.transpose(vec1_eeT)
a_M2_e = np.transpose(vec2_eeT)
a_M3_e = np.transpose(vec3_eeT)
a_M4_e = np.transpose(vec4_eeT)
a_M5_e = np.transpose(vec5_eeT)
a_M6_e = np.transpose(vec6_eeT)
display(a_M1_e, a_M2_e, a_M3_e,a_M4_e,a_M5_e,a_M6_e)
```

```
array([[ 1.  ,  0.  ,  0.  ],
       [-0.35,  0.48,  0.81],
       [-0.54,  0.72, -0.43]])
array([[ 0.97, -0.03,  0.23],
       [-0.2 , -0.57,  0.8 ],
       [-0.11,  0.82,  0.56]])
array([[-0.86,  0.12, -0.49],
       [ 0.48, -0.14, -0.87],
       [ 0.18,  0.98, -0.06]])
array([[ 0.8 ,  0.26, -0.55],
       [ 0.51, -0.77,  0.37],
       [ 0.32,  0.58,  0.75]])
array([[-0.51,  0.77, -0.37],
       [ 0.8 ,  0.26, -0.55],
       [ 0.32,  0.58,  0.75]])
array([[ 0.57, -0.63, -0.53],
       [-0.67, -0.73,  0.16],
       [ 0.49, -0.26,  0.83]])
```

(iv) Using two of the calculated bases, select two and find the components of the Stiffness matrix with respect to these new eigenbases

Eigenbase 1

In [72]:
```
A_M1_e = bl.tran_a_A(a_M1_e)
A_e_M1 = np.transpose(A_M1_e)

E_M1_M1_test = A_M1_e.dot(E_ee).dot(A_e_M1)
display(E_M1_M1_test);
```

```
array([[ 48.66,  28.75,  32.62, -24.04,  12.98, -37.21],
       [ 28.75,  48.66,  26.75, -23.93,  12.95, -21.98],
       [ 32.62,  26.75,  48.79, -16.09,  13.04, -37.35],
       [-24.04, -23.93, -16.09,  31.81, -17.16,  18.35],
       [ 12.98,  12.95,  13.04, -17.16,  25.29, -15.55],
       [-37.21, -21.98, -37.35,  18.35, -15.55,  44.57]])
```

Eigenbase 2

In [73]:
```
A_M2_e = bl.tran_a_A(a_M2_e)
A_e_M2 = np.transpose(A_M2_e)

E_M2_M2_test = A_M2_e.dot(E_ee).dot(A_e_M2)
display(E_M2_M2_test);
```

```
array([[ 48.62,  26.03,  26.  ,  -0.02,  -0.01,  -0.02],
       [ 26.03,  48.65,  25.97,  -0.04,  -0.02,  -0.06],
       [ 26.  ,  25.97,  48.69,   0.05,   0.02,   0.08],
       [ -0.02,  -0.04,   0.05,  22.81,  -0.09,  -0.01],
       [ -0.01,  -0.02,   0.02,  -0.09,  22.68,   0.08],
       [ -0.02,  -0.06,   0.08,  -0.01,   0.08,  22.74]])
```

Eigenbase 3

In [74]:
```
A_M4_e = bl.tran_a_A(a_M4_e)
A_e_M4 = np.transpose(A_M4_e)

E_M4_M4_test = A_M4_e.dot(E_ee).dot(A_e_M4)
display(E_M4_M4_test);
```

```
array([[ 48.79,  25.93,  25.93,   0.  ,   0.  ,   0.  ],
       [ 25.93,  48.79,  25.93,   0.  ,   0.  ,   0.  ],
       [ 25.93,  25.93,  48.79,   0.  ,   0.  ,   0.  ],
       [  0.  ,   0.  ,   0.  ,  22.6 ,   0.  ,   0.  ],
       [  0.  ,   0.  ,   0.  ,   0.  ,  22.6 ,   0.  ],
       [  0.  ,  -0.  ,  -0.  ,   0.  ,   0.  ,  22.6 ]])
```

(v) Obtain material vectors with respect to the E-E basis

In [75]:
```
a_M1_E = a_M1_e.dot(a_eE)
a_M2_E = a_M2_e.dot(a_eE)
a_M3_E = a_M3_e.dot(a_eE)
a_M4_E = a_M4_e.dot(a_eE)
a_M5_E = a_M5_e.dot(a_eE)
a_M6_E = a_M6_e.dot(a_eE)

display(a_M1_E, a_M2_E, a_M3_E, a_M4_E, a_M5_E, a_M6_E, )
```

```
array([[ 0.8 ,  0.32,  0.51],
       [-0.59,  0.77, -0.24],
       [-0.01, -0.08, -1.  ]])
```

```
array([[ 0.65,  0.47,  0.6 ],
       [-0.74,  0.21,  0.64],
       [-0.18,  0.86, -0.48]])
```

```
array([[-0.38, -0.58, -0.72],
       [ 0.82, -0.58,  0.03],
       [ 0.43,  0.58, -0.69]])
```

```
array([[ 1., -0.,  0.],
       [-0.,  0.,  1.],
       [ 0.,  1., -0.]])
```

```
array([[-0.,  0., -1.],
       [ 1., -0., -0.],
       [ 0.,  1.,  0.]])
```

```
array([[ 0.57, -0.58,  0.58],
       [-0.81, -0.52,  0.28],
       [-0.14,  0.63,  0.76]])
```

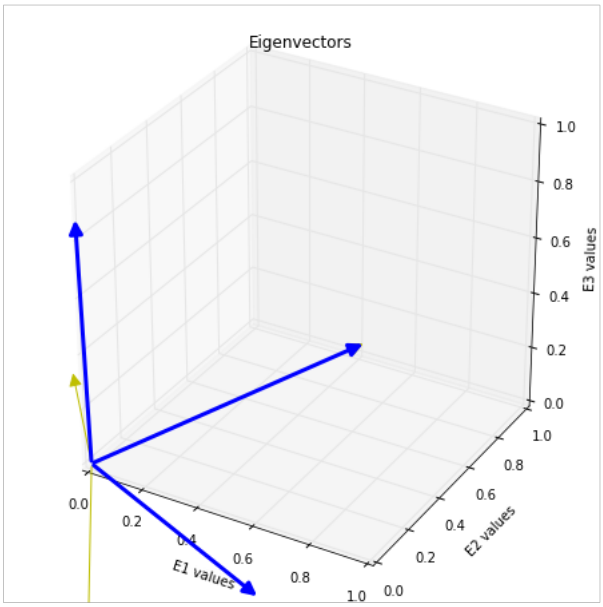
Create Plots

Plotting Function

In [77]:
```
%matplotlib inline
# %matplotlib qt

import numpy as np
from numpy import *
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.patches import FancyArrowPatch
from mpl_toolkits.mplot3d import proj3d

class Arrow3D(FancyArrowPatch):
    def __init__(self, xs, ys, zs, *args, **kwargs):
        FancyArrowPatch.__init__(self, (0,0), (0,0), *args, **kwargs)
        self._verts3d = xs, ys, zs

    def draw(self, renderer):
        xs3d, ys3d, zs3d = self._verts3d
        xs, ys, zs = proj3d.proj_transform(xs3d, ys3d, zs3d, renderer.M)
        self.set_positions((xs[0],ys[0]),(xs[1],ys[1]))
        FancyArrowPatch.draw(self, renderer)
```

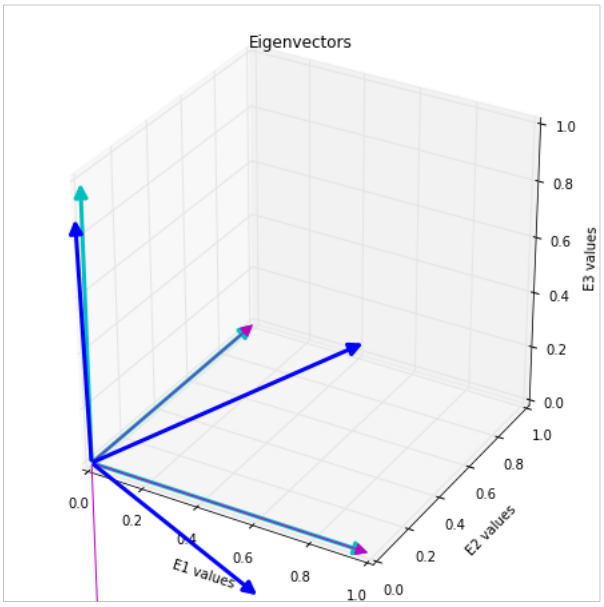
First Material Basis

In [78]:
```
##############################
#plotting eigenvectors
##############################

fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(111, projection='3d')

# Material Basis 1
for v in a_M1_E:
        a = Arrow3D([0,v[0]], [0,v[1]], [0, v[2]], mutation_scale=20, lw=1, arrowstyle="-|>", color="y")
        ax.add_artist(a)

# e basis: Blue
for v in a_eE:
    a = Arrow3D([0, v[0]], [0, v[1]], [0, v[2]], mutation_scale=20, lw=3, arrowstyle="-|>", color="b")
    ax.add_artist(a)

ax.set_xlabel('E1 values')
ax.set_ylabel('E2 values')
ax.set_zlabel('E3 values')

plt.title('Eigenvectors')

plt.draw()
plt.show()
```



Second Material Basis

In [79]:
```
##############################
#plotting eigenvectors
##############################

fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(111, projection='3d')

# 4th material basis: Cyan
for v in a_M4_E:
    a = Arrow3D([0, v[0]], [0, v[1]], [0, v[2]], mutation_scale=20, lw=3, arrowstyle="-|>", color="c")
    ax.add_artist(a)

    # Third eigenbasis: Magenta
for v in a_M5_E:
    a = Arrow3D([0, v[0]], [0, v[1]], [0, v[2]], mutation_scale=20, lw=1, arrowstyle="-|>", color="m")
    ax.add_artist(a)

# e-e basis: Blue
for v in a_eE:
    a = Arrow3D([0, v[0]], [0, v[1]], [0, v[2]], mutation_scale=20, lw=3, arrowstyle="-|>", color="b")
    ax.add_artist(a)

ax.set_xlabel('E1 values')
ax.set_ylabel('E2 values')
ax.set_zlabel('E3 values')

plt.title('Eigenvectors')

plt.draw()
plt.show()
```



Third Material basis

In [80]:
```
##############################
#plotting eigenvectors
##############################

fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(111, projection='3d')

# Fifth eigenbasis: Green
for v in a_M2_E:
    a = Arrow3D([0, v[0]], [0, v[1]], [0, v[2]], mutation_scale=20, lw=1, arrowstyle="-|>", color="g")
    ax.add_artist(a)

# Third eigenbasis: Blue
for v in a_M3_E:
    a = Arrow3D([0, v[0]], [0, v[1]], [0, v[2]], mutation_scale=20, lw=1, arrowstyle="-|>", color="r")
    ax.add_artist(a)

# Fifth eigenbasis: Black
for v in a_M6_E:
    a = Arrow3D([0, v[0]], [0, v[1]], [0, v[2]], mutation_scale=20, lw=1, arrowstyle="-|>", color="k")
    ax.add_artist(a)

# e-e basis: Blue
for v in a_eE:
    a = Arrow3D([0, v[0]], [0, v[1]], [0, v[2]], mutation_scale=20, lw=3, arrowstyle="-|>", color="b")
    ax.add_artist(a)

ax.set_xlabel('E1 values')
ax.set_ylabel('E2 values')
ax.set_zlabel('E3 values')

plt.title('Eigenvectors')

plt.draw()
plt.show()
```