In [6]:
```
import numpy as np
```

In [7]:
```
def helper_1(v, C):
    C1 = np.linalg.inv(
        np.matrix([[C[0], C[5], C[4]], [C[5], C[1], C[3]], [C[4], C[3], C[2]]]))
    return [C1[0, 0], C1[1, 1], C1[2, 2], C1[1, 2], C1[0, 2], C1[0, 1]], v
```

In [8]:
```
def helper_2(v, C):
    I1 = C[0] + C[1] + C[2]
    I2 = (-C[0]**2 - C[1]**2 - C[2]**2 - 2*C[3]**2 - 2*C[4]
          ** 2 - 2*C[5]**2 + (C[0] + C[1] + C[2]) ** 2)/2
    I3 = C[0]*C[1]*C[2] - C[0]*C[3]**2 - C[1] * \
        C[4]**2 - C[2]*C[5]**2 + 2*C[3]*C[4]*C[5]
    invariants_list = [I1, I2, I3]
    return invariants_list, v
```

In [9]:
```
def test_many_vectors(vs, C):
    accumulator = np.zeros((6))

    # both helper function takes the same input, input stored in tuple
    input_bundle = (np.random.rand(3), C)
    trace, _, det = helper_2(*input_bundle)[0]
    C_inv = helper_1(*input_bundle)[0]

    #calculation of each elemet of Matrix-D
    D1 = np.array([1, 1, 1, 0, 0, 0])
    D2 = trace*D1-C
    D3 = det*np.array(C_inv)
    D_mat = np.array([D1, D2, D3]).transpose()

    # for-loop has been replaced by efficient list comprehension
    accumulator = sum([np.dot(D_mat, v) for v in vs])

    return accumulator / len(vs)
```

In [10]:
```
if __name__ == '__main__':
    np.random.seed(5)
    shear = np.random.uniform(0.01, 0.2)
    C = np.array([1 + shear**2, 1, 1, 0, 0, shear])

    vs = [np.random.rand(3) for i in range(100000)]

    print(test_many_vectors(vs, C))
```

```
[ 2.00407174  2.00680091  2.00543787  0.          0.         -0.05230429]
```