Shadow Paper Notes

December 15, 2022

[45]:
```
import numpy as np
from scipy.optimize import minimize
import networkx as nx
from scipy import stats
```

[46]:
```
#Let E be the set of hyperedges in a hypergraph.  We assume that the edges of E␣
 ↪are labeled canonically beginning with 0.
#E is written as a list of lists.

#Given a list of hyperedges E, returns a (0,1)-matrix with |V| rows and |E|␣
 ↪columns where [v,e] is 1 precisely when v is in e.
def incidence_matrix(E=[]):
    V = []
    for e in E:
        V.extend(e)
    V = list(set(V))
    n = len(V)
    M = np.zeros((n,len(E)))
    for i in range(len(E)):
        e = E[i]
        if len(e) > 1:
            for j in range(len(e)):
                M[e[j],i] = M[e[j],i] +1
    return M

#Given a list of hyperedges E, returns the 2-shadow graph G which is a␣
 ↪multigraph (i.e., graph with weighted edges) where each edge in E is␣
 ↪replaced with a complete graph of the same size.
def shadow_graph(E =[]):
    I = incidence_matrix(E)
    n = len(I)
    M = np.matmul(I,I.T)
    for i in range(len(M)):
        M[i,i] = 0
    G = nx.Graph()
    for i in range(n):
        for j in range(i+1,n):
            G.add_edge(i,j, weight = M[i,j])
    return G

#Given a list of hyperedges E, returns the principal eigenvector of the␣
↪2-shadow graph.  That is, the principal eigenvector of the co-occurence␣
↪matrix M which is a non-negative integer valued matrix with |V| rows and |V|␣
↪columns where [v,u] is equal to the number of edges containing u and v.
#Unit vector under the 2-norm
def principal_eigenvector_shadow(E=[]):
    G = shadow_graph(E)
    x =[0 for i in range(len(G.nodes))]
    eig_vec = nx.eigenvector_centrality(G, weight = 'weight')
    for u in G.nodes():
        x[u] = eig_vec[u]
    return x

#Given a list of hyperedges E, returns the degree vector of the edge set.
#Unit vector under the 1-norm
#With an abuse of notation, the degree vector of a hypergraph is thought of as␣
↪the principal eigenvector of its 1-shadow.
def degree_vector(E=[]):
    V = []
    for e in E:
        V.extend(e)
    V = list(set(V))
    y = [0 for i in range(len(V))]
    for v in V:
        d = 0
        for e in E:
            if v in e:
                d = d+1
        y[v] = d
    s = sum(y)
    y = [u*s**(-1) for u in y]
    return list(y)

#Given a list of hyperedges E and an approximation for the principal␣
↪eigenvector x, returns an ordered pair (lower_bound, upper_bound) which are␣
↪bounds on the spectral radius of the Hypergraph.
#Bounds are computed using the formula using Lemma 3.
def HG_Collatz_Wielandt_Lemma(E=[], x=[]):
    W = []
    V = []
    for e in E:
        V.extend(e)
    V = list(set(V))
    for v in V:
        s = 0
        for e in E:
            if v in e:
                t = 1
                for u in e:
                    if u != v:
                        t = t*x[u]
                s = s + t
        W.append(s/(x[v]**2))
    return (min(W),max(W))

#Given a list of hyperedges E and an approximation for the principal␣
 ↪eigenvector x, returns an ordered pair (lower_bound, upper_bound) which are␣
 ↪bounds on the spectral radius of the 2-shadow.
#Bounds are computed using the formula using Lemma 3.
def G_Collatz_Wielandt_Lemma(E=[], x=[]):
    G = shadow_graph(E)
    W = []
    V = []
    for e in E:
        V.extend(e)
    V = list(set(V))
    for v in V:
        s = 0
        for e in G.edges:
            if v in e:
                t = 1
                for u in e:
                    if u != v:
                        t = t*x[u]
                s = s + G.get_edge_data(e[0],e[1])['weight']*t
        W.append(s/(x[v]))
    return (min(W),max(W))
```

[47]:
```
#### 8-Pleated Bow Tie
E =␣
 ↪[[0,1,2],[1,2,3],[1,2,4],[1,2,5],[1,2,6],[1,2,7],[1,2,8,],[1,2,9],[1,2,10],[0,11,12],[0,11,

def objective(x):
    x1 = x[0]
    x2 = x[1]
    x3 = x[2]
    x4 = x[3]
    x5 = x[4]
    x6 = x[5]
    x7 = x[6]
    x8 = x[7]
    x9 = x[8]
    x10 = x[9]
    x11 = x[10]
    x12 = x[11]
    x13 = x[12]
    x14 = x[13]
    x15 = x[14]
    x16 = x[15]
    x17 = x[16]
    x18 = x[17]
    x19 = x[18]
    x20 = x[19]
    x21 = x[20]
    f = 0
    for e in E:
        f = f + x[e[0]]*x[e[1]]*x[e[2]]
    return -3*f


def constraint1(x):
    sum_eq = 1
    for i in range(21):
        sum_eq = sum_eq - x[i]**3
    return sum_eq

def constraint2(x):
    sum_eq = 1/3
    A = [0,3,4,5,6,7,8,9,10]
    for a in A:
        sum_eq = sum_eq - x[a]**3
    return sum_eq

n = 21
y0 = np.zeros(n)
for i in range(n):
    y0[i] = 1.0
b = (0.0,1.0)
bnds = tuple([b for i in range(n)])
con1 = {'type': 'eq', 'fun': constraint1}
con2 = {'type': 'eq', 'fun': constraint2}
cons = ([con1,con2])

solution = minimize(objective,y0,method='trust-constr',␣
 ↪bounds=bnds,constraints=cons)
y= solution.x
```

```
print("8-Pleated Bow Tie Hypergraph")

print('--------------------------------------------------------------------------------')

print('Approximate Principal Eigenvector of B_8')

print('ym = ' + str(y[0]))
print('yr_1 = ' + str(y[1]))
print('yr_2 = ' + str(y[2]))
print('yr_3 = ' + str(y[3]))
print('yr_4 = ' + str(y[4]))
print('yr_5 = ' + str(y[5]))
print('yr_6 = ' + str(y[6]))
print('yr_7 = ' + str(y[7]))
print('yr_8 = ' + str(y[8]))
print('yr_9 = ' + str(y[9]))
print('yr_10 = ' + str(y[10]))
print('yell_1 = ' + str(y[11]))
print('yell_2 = ' + str(y[12]))
print('yell_3 = ' + str(y[13]))
print('yell_4 = ' + str(y[14]))
print('yell_5 = ' + str(y[15]))
print('yell_6 = ' + str(y[16]))
print('yell_7 = ' + str(y[17]))
print('yell_8 = ' + str(y[18]))
print('yell_9 = ' + str(y[19]))
print('yell_10 = ' + str(y[20]))

B = HG_Collatz_Wielandt_Lemma(E, y)

print('Collatz-Wielandt Bounds: Spectral radius is in [' + str(B[0]) +','+␣
 ↪str(B[1])+']')
print('Polynomial Form Lower Bound: Spectral radius is at least ' +␣
 ↪str(-objective(y)))

d = degree_vector(E)
x = principal_eigenvector_shadow(E)

print('--------------------------------------------------------------------------------')
print('Approximate Principal Eigenvector of 2-shadow of B_8')

print('xm = ' + str(x[0]))
print('xr_1 = ' + str(x[1]))
print('xr_2 = ' + str(x[2]))
print('xr_3 = ' + str(x[3]))
print('xr_4 = ' + str(x[4]))
print('xr_5 = ' + str(x[5]))
print('xr_6 = ' + str(x[6]))
print('xr_7 = ' + str(x[7]))
print('xr_8 = ' + str(x[8]))
print('xr_9 = ' + str(x[9]))
print('xr_10 = ' + str(x[10]))
print('xell_1 = ' + str(x[11]))
print('xell_2 = ' + str(x[12]))
print('xell_3 = ' + str(x[13]))
print('xell_4 = ' + str(x[14]))
print('xell_5 = ' + str(x[15]))
print('xell_6 = ' + str(x[16]))
print('xell_7 = ' + str(x[17]))
print('xell_8 = ' + str(x[18]))
print('xell_9 = ' + str(x[19]))
print('xell_10 = ' + str(x[20]))

C = G_Collatz_Wielandt_Lemma(E, x)
print('Collatz-Wielandt Bounds: Spectral Radius is in [' + str(C[0]) +','+␣
↪str(C[1])+']')

print('----------------------------------------------------------------------------')
print('Approximate Degree Vector of B_8')
print('dm = ' + str(d[0]))
print('dr_1 = ' + str(d[1]))
print('dr_2 = ' + str(d[2]))
print('dr_3 = ' + str(d[3]))
print('dr_4 = ' + str(d[4]))
print('dr_5 = ' + str(d[5]))
print('dr_6 = ' + str(d[6]))
print('dr_7 = ' + str(d[7]))
print('dr_8 = ' + str(d[8]))
print('dr_9 = ' + str(d[9]))
print('dr_10 = ' + str(d[10]))
print('dell_1 = ' + str(d[11]))
print('dell_2 = ' + str(d[12]))
print('dell_3 = ' + str(d[13]))
print('dell_4 = ' + str(d[14]))
print('dell_5 = ' + str(d[15]))
print('dell_6 = ' + str(d[16]))
print('dell_7 = ' + str(d[17]))
print('dell_8 = ' + str(d[18]))
print('dell_9 = ' + str(d[19]))
print('dell_10 = ' + str(d[20]))

print('----------------------------------------------------------------------------')
print('Correlation Coefficients')
print('Pearson(y,x)='+str(stats.pearsonr(x,y)[0]))
print('Pearson(y,d)='+str(stats.pearsonr(y,d)[0]))
print('Pearson(x,d)='+str(stats.pearsonr(x,d)[0]))

y_round = [format(v, '.6f') for v in y]
x_round = [format(v, '.6f') for v in x]
d_round = [format(v, '.6f') for v in d]

print('Spearman(y_round,x_round)='+str(stats.spearmanr(x_round,y_round)[0]))
print('Spearman(y_round,d_round)='+str(stats.spearmanr(y_round,d_round)[0]))
print('Spearman(x_round,d_round)='+str(stats.spearmanr(x_round,d_round)[0]))

print('Shadow Length='+str(((1-stats.spearmanr(x_round,y_round)[0])*(1-stats.
 ↪spearmanr(y_round,d_round)[0])*(1-stats.spearmanr(x_round,d_round)[0]))**(1/
 ↪3)))
```

```
8-Pleated Bow Tie Hypergraph
------------------------------------------------------------------------------
Approximate Principal Eigenvector of B_8
ym = 0.5703210695671954
yr_1 = 0.5725498830384635
yr_2 = 0.5725498913828883
yr_3 = 0.26437565235267013
yr_4 = 0.26437566150830694
yr_5 = 0.26437565291272236
yr_6 = 0.2643756535475594
yr_7 = 0.2643756599985115
yr_8 = 0.2643756634566221
yr_9 = 0.2643756610482333
yr_10 = 0.26437565595007867
yell_1 = 0.5261351987745225
yell_2 = 0.25293944433204557
yell_3 = 0.2529394293326771
yell_4 = 0.25293942637942446
yell_5 = 0.252939408114653
yell_6 = 0.25293941253204644
yell_7 = 0.2529394287747368
yell_8 = 0.25293941659567865
yell_9 = 0.252939410455218
yell_10 = 0.2529394142994963
Collatz-Wielandt Bounds: Spectral radius is in
[4.690116745434148,4.690118088552746]
Polynomial Form Lower Bound: Spectral radius is at least 4.6901176253669
------------------------------------------------------------------------------
Approximate Principal Eigenvector of 2-shadow of B_8
xm = 0.5070057260865928
xr_1 = 0.4554540089458123
```

```
xr_2 = 0.4554540089458123
xr_3 = 0.07918187548913753
xr_4 = 0.07918187548913753
xr_5 = 0.07918187548913753
xr_6 = 0.07918187548913753
xr_7 = 0.07918187548913753
xr_8 = 0.07918187548913753
xr_9 = 0.07918187548913753
xr_10 = 0.07918187548913753
xell_1 = 0.4625798622653105
xell_2 = 0.08428140757444742
xell_3 = 0.08428140757444742
xell_4 = 0.08428140757444742
xell_5 = 0.08428140757444742
xell_6 = 0.08428140757444742
xell_7 = 0.08428140757444742
xell_8 = 0.08428140757444742
xell_9 = 0.08428140757444742
xell_10 = 0.08428140757444742
Collatz-Wielandt Bounds: Spectral Radius is in
[11.50399649243704,11.504145650337524]
-------------------------------------------------------------------------------
Approximate Degree Vector of B_8
dm = 0.18518518518518517
dr_1 = 0.16666666666666666
dr_2 = 0.16666666666666666
dr_3 = 0.018518518518518517
dr_4 = 0.018518518518518517
dr_5 = 0.018518518518518517
dr_6 = 0.018518518518518517
dr_7 = 0.018518518518518517
dr_8 = 0.018518518518518517
dr_9 = 0.018518518518518517
dr_10 = 0.018518518518518517
dell_1 = 0.16666666666666666
dell_2 = 0.018518518518518517
dell_3 = 0.018518518518518517
dell_4 = 0.018518518518518517
dell_5 = 0.018518518518518517
dell_6 = 0.018518518518518517
dell_7 = 0.018518518518518517
dell_8 = 0.018518518518518517
dell_9 = 0.018518518518518517
dell_10 = 0.018518518518518517
-------------------------------------------------------------------------------
Correlation Coefficients
Pearson(y,x)=0.9944911984667654
Pearson(y,d)=0.9959619922750506
Pearson(x,d)=0.9998496534425814
Spearman(y_round,x_round)=0.07116104868913857
Spearman(y_round,d_round)=0.7262282061496975
Spearman(x_round,d_round)=0.7343880736345255
Shadow Length=0.40724794129935615
```

[48]:

```
#### 1-Pleated Bow tie
E = [[0,1,2],[1,2,3],[0,4,5],[0,4,6]]

#We begin by approximating the principal eigenvector of the hypergraph as a␣
↪constrained optimisation problem using scipy.optimize.minimize
def objective(y):
    y1 = y[0]
    y2 = y[1]
    y3 = y[2]
    y4 = y[3]
    y5 = y[4]
    y6 = y[5]
    y7 = y[6]
    f = 0
    for e in E:
        f = f + y[e[0]]*y[e[1]]*y[e[2]]
    return -3*f

def constraint1(y):
    sum_eq = 1
    for i in range(7):
        sum_eq = sum_eq - y[i]**3
    return sum_eq

n = 7
y0 = np.zeros(n)
y0[0] = 1.0
y0[1] = 1.0
y0[2] = 1.0
y0[3] = 1.0
y0[4] = 1.0
y0[5] = 1.0
y0[6] = 1.0


b = (0.0,1.0)
bnds = (b, b, b, b, b,b,b)
con1 = {'type': 'eq', 'fun': constraint1}
cons = ([con1])
solution = minimize(objective,y0,method='trust-constr',␣
↪bounds=bnds,constraints=cons)
```

```
y = solution.x

print("1-Pleated Bow Tie Hypergraph")

print('-------------------------------------------------------------------------------')

print('Approximate Principal Eigenvector of B_1')
print('yc = ' + str(y[0]))
print('yr_1 = ' + str(y[1]))
print('yr_2 = ' + str(y[2]))
print('yr_3 = ' + str(y[3]))
print('yell_1 = ' + str(y[4]))
print('yell_2 = ' + str(y[5]))
print('yell_3 = ' + str(y[6]))


B = HG_Collatz_Wielandt_Lemma(E, y)


print('Collatz-Wielandt Bounds: Spectral radius is in [' + str(B[0]) +','+␣
 ↪str(B[1])+']')
print('Polynomial Form Lower Bound: Spectral radius is at least ' +␣
 ↪str(-objective(y)))

d = degree_vector(E)
x = principal_eigenvector_shadow(E)

print('-------------------------------------------------------------------------------')

print('Approximate Principal Eigenvector of 2-shadow of B_1')
print('xc = ' + str(x[0]))
print('xr_1 = ' + str(x[1]))
print('xr_2 = ' + str(x[2]))
print('xr_3 = ' + str(x[3]))
print('xell_1 = ' + str(x[4]))
print('xell_2 = ' + str(x[5]))
print('xell_3 = ' + str(x[6]))

C = G_Collatz_Wielandt_Lemma(E, x)
print('Collatz-Wielandt Bounds: Spectral Radius is in [' + str(C[0]) +','+␣
 ↪str(C[1])+']')

print('-------------------------------------------------------------------------------')

print('Approximate Degree Vector of B_1')
print('dc = ' + str(d[0]))
print('dr_1 = ' + str(d[1]))
print('dr_2 = ' + str(d[2]))
print('dr_3 = ' + str(d[3]))
print('dell_1 = ' + str(d[4]))
print('dell_2 = ' + str(d[5]))
print('dell_3 = ' + str(d[6]))



print('-------------------------------------------------------------------------------')

print('Correlation Coefficients')

print('Pearson(y,x)='+str(stats.pearsonr(x,y)[0]))
print('Pearson(y,d)='+str(stats.pearsonr(y,d)[0]))
print('Pearson(x,d)='+str(stats.pearsonr(x,d)[0]))

#We round the vectors to 6 decimal places to allow
y_round = [format(v, '.6f') for v in y]
x_round = [format(v, '.6f') for v in x]
d_round = [format(v, '.6f') for v in d]

print('Spearman(y_round,x_round)='+str(stats.spearmanr(x_round,y_round)[0]))
print('Spearman(y_round,d_round)='+str(stats.spearmanr(y_round,d_round)[0]))
print('Spearman(x_round,d_round)='+str(stats.spearmanr(x_round,d_round)[0]))

print('Shadow Length='+str(((1-stats.spearmanr(x_round,y_round)[0])*(1-stats.
 ↪spearmanr(y_round,d_round)[0])*(1-stats.spearmanr(x_round,d_round)[0]))**(1/
 ↪3)))
```

```
1-Pleated Bow Tie Hypergraph
-------------------------------------------------------------------------------
Approximate Principal Eigenvector of B_1
yc = 0.6431819309742854
yr_1 = 0.5578858587473997
yr_2 = 0.5578858582206434
yr_3 = 0.4066793184886647
yell_1 = 0.5425426698993899
yell_2 = 0.4306164018491096
yell_3 = 0.4306164017307633
Collatz-Wielandt Bounds: Spectral radius is in
[1.8818566218081243,1.8818569192366803]
Polynomial Form Lower Bound: Spectral radius is at least 1.8818568056216485
-------------------------------------------------------------------------------
Approximate Principal Eigenvector of 2-shadow of B_1
xc = 0.5591865147866596
xr_1 = 0.4117513335770121
xr_2 = 0.4117513335770121
```

```
xr_3 = 0.21255597572648038
xell_1 = 0.4189988851668115
xell_2 = 0.25247971372744615
xell_3 = 0.25247971372744615
Collatz-Wielandt Bounds: Spectral Radius is in
[3.874286123170291,3.8743128527523205]
------------------------------------------------------------------------------
Approximate Degree Vector of B_1
dc = 0.25
dr_1 = 0.16666666666666666
dr_2 = 0.16666666666666666
dr_3 = 0.08333333333333333
dell_1 = 0.16666666666666666
dell_2 = 0.08333333333333333
dell_3 = 0.08333333333333333
------------------------------------------------------------------------------
Correlation Coefficients
Pearson(y,x)=0.9954862087116865
Pearson(y,d)=0.9878099702179215
Pearson(x,d)=0.9922882993728723
Spearman(y_round,x_round)=0.8888888888888891
Spearman(y_round,d_round)=0.9428090415820635
Spearman(x_round,d_round)=0.9428090415820635
Shadow Length=0.07136261347418814
```

[49]:
```
#Red Octahedron, OR
E = [[0,1,2],[0,3,4],[1,2,5],[2,3,6],[1,4,6]]
def objective(x):
    x1 = x[0]
    x2 = x[1]
    x3 = x[2]
    x4 = x[3]
    x5 = x[4]
    x6 = x[5]
    x7 = x[6]
    f = 0
    for e in E:
        f = f + x[e[0]]*x[e[1]]*x[e[2]]
    return -3*f

def constraint1(x):
    sum_eq = 1
    for i in range(7):
        sum_eq = sum_eq - x[i]**3
    return sum_eq

n = 7
```

```
x0 = np.zeros(n)
for i in range(n):
    x0[i] = 1.0
b = (0.0,1.0)
bnds = tuple([b for i in range(n)])
con1 = {'type': 'eq', 'fun': constraint1}
cons = ([con1])


b = (0.0,1.0)
bnds = (b, b, b, b, b,b,b)
con1 = {'type': 'eq', 'fun': constraint1}
cons = ([con1])
solution = minimize(objective,x0,method='trust-constr',␣
 ↪bounds=bnds,constraints=cons)
y = solution.x

print('Red Octahedron, O_R')
print('-------------------------------------------------------------------------------')
print('Approximate Principal Eigenvector of O_R')

print('yt = ' + str(y[0]))
print('yp = ' + str(y[1]))
print('yq = ' + str(y[2]))
print('ys = ' + str(y[3]))
print('yr = ' + str(y[4]))
print('yu = ' + str(y[5]))
print('yb = ' + str(y[6]))

B = HG_Collatz_Wielandt_Lemma(E, y)

print('Collatz-Wielandt Bounds: Spectral radius is in [' + str(B[0]) +','+␣
 ↪str(B[1])+']')
print('Polynomial Form Lower Bound: Spectral radius is at least ' +␣
 ↪str(-objective(y)))

d = degree_vector(E)
x = principal_eigenvector_shadow(E)

print('-------------------------------------------------------------------------------')
print('Approximate Principal Eigenvector of the 2-shadow O_R')
print('xt = ' + str(x[0]))
print('xp = ' + str(x[1]))
print('xq = ' + str(x[2]))
print('xs = ' + str(x[3]))
print('xr = ' + str(x[4]))
print('xu = ' + str(x[5]))
print('xb = ' + str(x[6]))

C = G_Collatz_Wielandt_Lemma(E, x)
print('Collatz-Wielandt Bounds: Spectral Radius is in [' + str(C[0]) +','+␣
↪str(C[1])+']')

print('------------------------------------------------------------------------------')
print('Approximate Degree Vector of O_R')
print('dt = ' + str(d[0]))
print('dp = ' + str(d[1]))
print('dq = ' + str(d[2]))
print('ds = ' + str(d[3]))
print('dr = ' + str(d[4]))
print('du = ' + str(d[5]))
print('db = ' + str(d[6]))

print('------------------------------------------------------------------------------')

print('Pearson(y,x)='+str(stats.pearsonr(x,y)[0]))
print('Pearson(y,d)='+str(stats.pearsonr(y,d)[0]))
print('Pearson(x,d)='+str(stats.pearsonr(x,d)[0]))

y_round = [format(v, '.6f') for v in y]
x_round = [format(v, '.6f') for v in x]
d_round = [format(v, '.6f') for v in d]

print('Spearman(y_round,x_round)='+str(stats.spearmanr(x_round,y_round)[0]))
print('Spearman(y_round,d_round)='+str(stats.spearmanr(y_round,d_round)[0]))
print('Spearman(x_round,d_round)='+str(stats.spearmanr(x_round,d_round)[0]))

print('Shadow Length='+str(((1-stats.spearmanr(x_round,y_round)[0])*(1-stats.
↪spearmanr(y_round,d_round)[0])*(1-stats.spearmanr(x_round,d_round)[0]))**(1/
↪3)))
```

```
Red Octahedron, O_R
------------------------------------------------------------------------------
Approximate Principal Eigenvector of O_R
yt = 0.5159682075921611
yp = 0.5938350919681654
yq = 0.5938351002440639
ys = 0.4985601994069094
yr = 0.4985601955991514
yu = 0.3951649716633049
yb = 0.5120581964109722
Collatz-Wielandt Bounds: Spectral radius is in
[2.258264439743861,2.258264673357055]
Polynomial Form Lower Bound: Spectral radius is at least 2.2582645424939174
```

```
--------------------------------------------------------------------------------
Approximate Principal Eigenvector of the 2-shadow O_R
xt = 0.35793972301726734
xp = 0.4870931240730303
xq = 0.4870931240730303
xs = 0.3348462644886994
xr = 0.3348462644886994
xu = 0.21211988273172985
xb = 0.35793972301726734
Collatz-Wielandt Bounds: Spectral Radius is in
[4.592611588319402,4.592621095204563]
--------------------------------------------------------------------------------
Approximate Degree Vector of O_R
dt = 0.13333333333333333
dp = 0.2
dq = 0.2
ds = 0.13333333333333333
dr = 0.13333333333333333
du = 0.06666666666666667
db = 0.13333333333333333
--------------------------------------------------------------------------------
Pearson(y,x)=0.9940642111949639
Pearson(y,d)=0.9913906395904082
Pearson(x,d)=0.9949947600530733
Spearman(y_round,x_round)=0.9906974722292784
Spearman(y_round,d_round)=0.9128709291752769
Spearman(x_round,d_round)=0.9214426752509268
Shadow Length=0.03993161859690086
```

[50]:
```python
#RBlue Octahedron, O_B
E = [[0,2,3],[0,1,4],[1,2,5],[1,2,6],[3,4,6]]
def objective(x):
    x1 = x[0]
    x2 = x[1]
    x3 = x[2]
    x4 = x[3]
    x5 = x[4]
    x6 = x[5]
    x7 = x[6]
    f = 0
    for e in E:
        f = f + x[e[0]]*x[e[1]]*x[e[2]]
    return -3*f

def constraint1(x):
    sum_eq = 1
    for i in range(7):
        sum_eq = sum_eq - x[i]**3
    return sum_eq

n = 7
x0 = np.zeros(n)
for i in range(n):
    x0[i] = 1.0
b = (0.0,1.0)
bnds = tuple([b for i in range(n)])
con1 = {'type': 'eq', 'fun': constraint1}
cons = ([con1])


b = (0.0,1.0)
bnds = (b, b, b, b, b,b,b)
con1 = {'type': 'eq', 'fun': constraint1}
cons = ([con1])
solution = minimize(objective,x0,method='trust-constr',␣
 ↪bounds=bnds,constraints=cons)
y = solution.x

print('Blue Octahedron, O_B')
print('-------------------------------------------------------------------------------')
print('Approximate Principal Eigenvector of O_B')

print('yt = ' + str(y[0]))
print('yp = ' + str(y[1]))
print('yq = ' + str(y[2]))
print('ys = ' + str(y[3]))
print('yr = ' + str(y[4]))
print('yu = ' + str(y[5]))
print('yb = ' + str(y[6]))

B = HG_Collatz_Wielandt_Lemma(E, y)

print('Collatz-Wielandt Bounds: Spectral radius is in [' + str(B[0]) +','+␣
 ↪str(B[1])+']')
print('Polynomial Form Lower Bound: Spectral radius is at least ' +␣
 ↪str(-objective(y)))

d = degree_vector(E)
x = principal_eigenvector_shadow(E)

print('-------------------------------------------------------------------------------')
print('Approximate Principal Eigenvector of the 2-shadow O_B')

print('xt = ' + str(x[0]))
print('xp = ' + str(x[1]))
print('xq = ' + str(x[2]))
print('xs = ' + str(x[3]))
print('xr = ' + str(x[4]))
print('xu = ' + str(x[5]))
print('xb = ' + str(x[6]))

C = G_Collatz_Wielandt_Lemma(E, x)
print('Collatz-Wielandt Bounds: Spectral Radius is in [' + str(C[0]) +','+␣
 ↪str(C[1])+']')

print('-------------------------------------------------------------------------------')

print('Approximate Degree Vector of O_B')
print('dt = ' + str(d[0]))
print('dp = ' + str(d[1]))
print('dq = ' + str(d[2]))
print('ds = ' + str(d[3]))
print('dr = ' + str(d[4]))
print('du = ' + str(d[5]))
print('db = ' + str(d[6]))

print('-------------------------------------------------------------------------------')

print('Pearson(y,x)='+str(stats.pearsonr(x,y)[0]))
print('Pearson(y,d)='+str(stats.pearsonr(y,d)[0]))
print('Pearson(x,d)='+str(stats.pearsonr(x,d)[0]))

y_round = [format(v, '.6f') for v in y]
x_round = [format(v, '.6f') for v in x]
d_round = [format(v, '.6f') for v in d]

print('Spearman(y_round,x_round)='+str(stats.spearmanr(x_round,y_round)[0]))
print('Spearman(y_round,d_round)='+str(stats.spearmanr(y_round,d_round)[0]))
print('Spearman(x_round,d_round)='+str(stats.spearmanr(x_round,d_round)[0]))

print('Shadow Length='+str(((1-stats.spearmanr(x_round,y_round)[0])*(1-stats.
 ↪spearmanr(y_round,d_round)[0])*(1-stats.spearmanr(x_round,d_round)[0]))**(1/
 ↪3)))
```

```
Blue Octahedron, O_B
-------------------------------------------------------------------------------
Approximate Principal Eigenvector of O_B
yt = 0.5120581996319441
yp = 0.5938350997586888
yq = 0.5938350978319967
ys = 0.4985601927000609
```

```
yr = 0.49856019258397627
yu = 0.39516497555459545
yb = 0.5159682040901082
Collatz-Wielandt Bounds: Spectral radius is in
[2.258264489945013,2.2582646493350875]
Polynomial Form Lower Bound: Spectral radius is at least 2.2582645424939165
------------------------------------------------------------------------------
Approximate Principal Eigenvector of the 2-shadow O_B
xt = 0.35793972301726734
xp = 0.4870931240730303
xq = 0.4870931240730303
xs = 0.3348462644886994
xr = 0.3348462644886994
xu = 0.21211988273172985
xb = 0.35793972301726734
Collatz-Wielandt Bounds: Spectral Radius is in
[4.592611588319402,4.592621095204563]
------------------------------------------------------------------------------
Approximate Degree Vector of O_B
dt = 0.13333333333333333
dp = 0.2
dq = 0.2
ds = 0.13333333333333333
dr = 0.13333333333333333
du = 0.06666666666666667
db = 0.13333333333333333
------------------------------------------------------------------------------
Pearson(y,x)=0.994064216622603
Pearson(y,d)=0.991390641479769
Pearson(x,d)=0.9949947600530733
Spearman(y_round,x_round)=0.9906974722292784
Spearman(y_round,d_round)=0.9128709291752769
Spearman(x_round,d_round)=0.9214426752509268
Shadow Length=0.03993161859690086
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[9]:
```

```
[13]:
```

```
[14]:
```

```
[15]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[50]:
```

```
[ ]:
```

```
[1]:
```

```
[7]:
```

```
[8]:
```

```
[ ]:
```

```

[ ]:
```

```