ENGN 2020:

Solutions to Midterm #1

Brown University School of Engineering

Problem 1

Part a. The elements that go into our matrix are:

$$\tilde{k}_{ij} = \frac{k_{ij}}{\sqrt{m_i m_j}}$$

This was analyzed with code at the end of the problem to find the natural frequencies, which are:

```
[ 0.29557781  0.71803543  1.14237507  1.35085132  2.42453087]
```

with units of s^{-1}. The associated eigenvectors are the columns of the below

```
[[-0.64816503 -0.5476156  -0.36709199  0.23347628 -0.30121678]
 [-0.20881728  0.15616332  0.28313297 -0.63382245 -0.67090475]
 [ 0.57204243  0.03508771 -0.26077684  0.40526135 -0.66279333]
 [-0.43780757  0.61993618  0.30544771  0.56207913 -0.12154181]
 [ 0.13179317 -0.53868187  0.78979512  0.25216704 -0.07132925]]
```

Eigenvectors are properly dimensionless, since they can be scaled by an arbitrary factor.

Part b. This is a basis set transformation, which we can do by

$$\underline{\mathbf{q}} = c_1\underline{\mathbf{v}}_1 + c_2\underline{\mathbf{v}}_2 + c_3\underline{\mathbf{v}}_3 + c_4\underline{\mathbf{v}}_4 + c_5\underline{\mathbf{v}}_5 = \underline{\underline{\mathbf{v}}}\,\underline{\mathbf{c}} \tag{1}$$

where $\underline{\mathbf{q}}$ is our given vector, $\underline{\underline{\mathbf{v}}}$ is a matrix containing the eigenvectors on the columns, and $\underline{\mathbf{c}}$ is a vector containing our five coefficients. We can use `np.linalg.solve` to find $\underline{\mathbf{c}}$.

Different assumptions could reasonably be made on whether the presented numbers contained the displacements directly, or whether the presented numbers already contained the $\sqrt{m_i}$ correction. Either interpretation is eligible for full credit.

1. Assuming numbers need mass correction. Here, we first calculate $\underline{\tilde{\mathbf{q}}} = \underline{\mathbf{m}} \circ \underline{\mathbf{q}}$, where $\circ$ indicates the element-wise product and $\underline{\mathbf{m}}$ contains the square-roots of the masses, and use this in equation (1). This gives:

```
[ 0.2241178  -1.04391844  0.4175895  -0.12186543 -0.12678062]
```

2. Assuming numbers contain mass correction. Here, we just use $\underline{\mathbf{q}}$ directly in equation (1), giving:

```
[ 0.10134328 -0.39418793  0.1201439  -0.09726451 -0.05523108]
```

Part c. Note that our force equation, as given in the problem statement, can be re-expressed as:

$$\underline{\mathbf{F}} = -\underline{\underline{\mathbf{k}}}\,\underline{\mathbf{q}}$$

This is simple to solve, and we get:

```
[[-0.72]
 [-1.  ]
 [-0.44]
 [ 0.47]
 [-2.16]]
```

Python code. File attached here.

```python
import numpy as np

masses = [8., 2., 3., 5., 9.]

ks = np.zeros((len(masses), len(masses)))
ks[0, 0] = 8.
ks[0, 1] = 3.
ks[0, 2] = 7.
ks[0, 3] = 1.
ks[0, 4] = 0.

ks[1, 1] = 7.
ks[1, 2] = 5.
ks[1, 3] = 0.
ks[1, 4] = 1.

ks[2, 2] = 9.
ks[2, 3] = 3.
ks[2, 4] = 1.

ks[3, 3] = 5.
ks[3, 4] = 3.

ks[4, 4] = 10.

# Copy symmetric values:
ks = ks + ks.T - np.diag(ks.diagonal())
print(ks)

# Make k-tildes.
ktildes = np.zeros((len(masses), len(masses)))
for i in range(len(masses)):
    for j in range(len(masses)):
        ktildes[i, j] = ks[i, j] / np.sqrt(masses[i] * masses[j])

print(ktildes)
w, v = np.linalg.eigh(ktildes)
print(w)

print('The natural frequencies are:')
print(np.sqrt(w))
print(v)

#####################
# Part b.
#####################
q = np.array([0.1, .05, .01, -.3, .3])
# Approach 1.
qt = np.sqrt(masses) * q
print(np.linalg.solve(v, qt))
# Approach 2.
print(np.linalg.solve(v, q))
```

```python

#####################
# Part c.
#####################
F = -np.matmul(ks, q)
print(F)
```

Rubric. Total 15 points.

- Part a. 8 points (pick one from each group separated by carriage return)

  (+1) natural frequency units 1/s (rad/s OK)
  (+0) incorrect frequency units

  (+1) eigenvector units unitless or m, with correct frequency/justification
  (+0) incorrect eigenvector units

  (+3) correct five eigenvalues
  (+2) eigenvalue numbers all correct but wrong signs or squared
  (+1) one incorrect eigenvalue (not sign error)
  (+0) more than one incorrect eigenvalue (not sign error)

  (+3) correct five eigenvectors
  (+2) eigenvalues do not match eigenvectors but otherwise correct
  (+2) one incorrect eigenvector or fewer than three small typos across multiple
  (+1) eigenvector numbers correct but wrongly said they were rows not columns etc.
  (+1) eigenvectors not normalized but correct
  (+0) more than one incorrect eigenvector or 3+ typos

- Problem 1b. Max total: 5 points

  (+5) correct five coefficients
  (+3) correct numbers but coefficient order does not match basis set order, or all wrong sign
  (+3) four correct coefficients
  (+2) incorrect coefficients due to input but correct input type and attempt to solve q=vc for c
  (+1) incorrect coefficients and input type but attempt to solve q=vc for c
  (+0) incorrect values and/or incorrect procedure

- Problem 1c. Max total: 2 points

  (+2) correct five forces
  (+1) correct numbers but sign and/or ordering error
  (+1) multiple incorrect forces but correct attempt to solve F=-kq
  (+1) any incorrect input but correct attempt to solve F=-kq
  (+0) incorrect values and/or incorrect procedure

Problem 2

Part a. The key equation was given in eq (4) of the reference

$$\underline{\mathbf{x}} = (1-m)\underline{\underline{\mathbf{A}}}\underline{\mathbf{x}} + m\underline{\mathbf{s}}$$

We try this on the two examples in the paper, and one extra, with a link deleted. Note when the link is deleted, we need to be a little careful not to divide by zero. For the three values of m, we get:

```
0.15
[[ 0.41432569  0.41432569  0.09130925  0.04751391  0.03252545]]
0.25
[[ 0.35827687  0.35827687  0.14731658  0.07991615  0.05621352]]
0.35
[[ 0.30981002  0.30981002  0.19089057  0.10957189  0.07991751]]
```

File attached here.

```python
import numpy as np

def make_A(n, links):
    """Make the link matrix."""
    A = np.zeros((n, n))
    for link in links:
        A[link[1], link[0]] += 1.
    # Make each column sum to one.
    Asum = A.sum(axis=0)
    # If any columns have no entries, they should stay zero, not nan.
    for index, value in enumerate(Asum):
        if value == 0.:
            Asum[index] = 1.
    A = A / Asum
    return A

def power_method(m, iterations=10):
    """Run the power method."""
    x = np.ones((n, 1))  # Guess of x.
    for _ in range(10):
        x = (1. - m) * np.matmul(A, x) + m * s
        x /= np.linalg.norm(x, ord=1)
        print(x.T)

# Figure 2, modified.
n = 5  # number of pages
links = [[1, 2], [2, 1], [4, 3], [5, 3], [5, 4]]
# Bring to python numbering system.
for link in links:
    link[0] -= 1
    link[1] -= 1

A = make_A(n, links)
s = 1./ n * np.ones((n, 1))
```

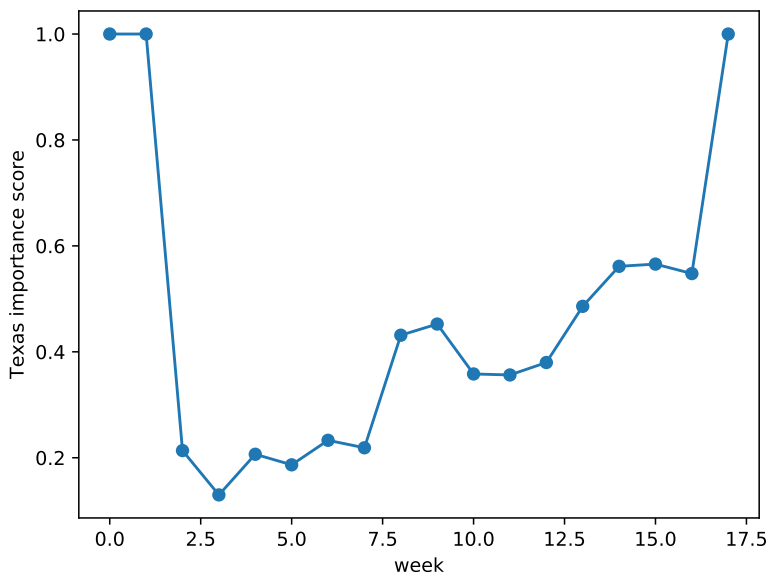

Figure 1: Texas' importance score by week.

```
for m in [0.15, 0.25, 0.35]:
    print(m)
    power_method(m)
```

Part b. The code is not shown, as it is a simplified version of that in part c. The two unique values are `(0.235606, 1.)`.

Part c. The top ten teams, according to this ranking scheme, are (with their importance scores shown):

```
1.000 Texas
0.835 LSU
0.823 Virginia Tech
0.815 Michigan
0.757 Florida State
0.700 Georgia
0.695 Penn State
0.653 Miami (Florida)
0.545 Georgia Tech
0.544 Auburn
```

Texas was the top team at the end; their week-by-week score is shown in Figure 1. Code to generate this follows. File attached here.

```python
import json
import numpy as np
from matplotlib import pyplot
np.set_printoptions(precision=2)

def make_A(n, links):
    """Make the link matrix."""
    A = np.zeros((n, n))
    for link in links:
        A[link[1], link[0]] += 1.
    # Make each column sum to one.
    Asum = A.sum(axis=0)
    # If any columns have no entries, they should stay zero, not nan.
    for index, value in enumerate(Asum):
        if value == 0.:
            Asum[index] = 1.
    A = A / Asum
    return A

def get_importance_scores(links, n, m=0.15, iterations=1000):
    A = make_A(n, links)
    m = 0.15
    S = 1. / n * np.ones((n, n))
    M = (1. - m) * A + m * S
    # Power method.
    x = np.ones((n, 1))  # Guess of x.
    for _ in range(iterations):
        x = np.matmul(M, x)
        x /= x.max()
    return x

def get_links(results):
    """Creates links list from results list."""
    links = []
    for r in results:
        if r['home_score'] > r['away_score']:
            links.append([r['away_team'], r['home_team']])
        else:
            links.append([r['home_team'], r['away_team']])
    return links

def get_results_until_week(week):
    partial_results = []
    for result in results:
        if result['week'] <= week:
            partial_results.append(result)
    return partial_results

results = json.load(open('results.json'))
team_names = json.load(open('teams.json'))
n = len(team_names)

partial_results = get_results_until_week(np.inf)
links = get_links(partial_results)
```

```python
x = get_importance_scores(links, n)
sorted_indices = np.argsort(x.flatten())
sorted_indices = np.flip(sorted_indices, axis=0)
for index in sorted_indices[:10]:
    print('{:.3f} {:s}'.format(float(x[index]), team_names[index]))

# Now make a plot of what Texas did throughout the year.
# They are team #202.
texas_scores = []
for week in range(18):
    partial_results = get_results_until_week(week)
    links = get_links(partial_results)
    x = get_importance_scores(links, n)
    texas_score = x[202]
    print(texas_score)
    texas_scores.append(texas_score)

fig, ax = pyplot.subplots()
ax.plot(range(18), texas_scores, 'o-')
ax.set_xlabel('week')
ax.set_ylabel('Texas importance score')
fig.savefig('texas.pdf')
```

Rubric. Total 15 points.

- Part a: 4 points

  4/4 all three importance scores are correct
  3/4 two of three importance scores are correct
  2/4 one of three importance scores are correct
  1/4 all importance scores are wrong but have right M
  1/4 all importance scores are wrong but code provided
  0/4 importance scores are wrong and no code provided

- Part b: 2 points

  2/2 both importance scores are correct
  1/2 importance scores are wrong but code provided
  0/2 importance scores are wrong and no code provided

- Part c, top-ten ranking: 3 points

  3/3 all top ten rankings and importance scores are correct.
  2/3 ranking are wrong but all importance score are correct.
  1/3 rankings are correct but importance scores are wrong.
  0/3 all rankings and importance score are wrong.

- Part c, plot: 2 points

  2/2 The plot is right (Texas, x-axis is week and y-axis is importance score)
  2/2 The plot is not right or not normalized, but the importance scores for every week are provided and these scores are right
  1/2 The plot is right but the team names are wrong

0/2 The plot is not right or not normalized, and no importance scores for every week provided.

- Part c, code: 4 points

  4/4 code makes sense.
  3/4 code doesnt make sense..
  0/4 no code provided.

Problem 3

Part a.

- Node 1, x direction:
$$0 = -F_{\mathrm{A}} \cos\beta + F_{\mathrm{C}} \cos\gamma$$
- Node 1, y direction:
$$0 = -F_{\mathrm{load}} - F_{\mathrm{A}} \sin\beta - F_{\mathrm{C}} \sin\gamma$$
- Node 2, x direction:
$$0 = F_{\mathrm{B}} + H_2 + F_{\mathrm{A}} \cos\beta$$
- Node 2, y direction:
$$0 = V_2 + F_{\mathrm{A}} \sin\beta$$
- Node 3, x direction:
$$0 = -F_{\mathrm{B}} - F_{\mathrm{C}} \cos\gamma$$
- Node 3, y direction:
$$0 = V_3 + F_{\mathrm{C}} \sin\gamma$$

Part b. Converting to a linear system, $\underline{\underline{\mathbf{A}}}\ \underline{\mathbf{x}} = \underline{\mathbf{b}}$, gives

$$\begin{bmatrix} -\cos\beta & 0 & \cos\gamma & 0 & 0 & 0 \\ -\sin\beta & 0 & -\sin\gamma & 0 & 0 & 0 \\ \cos\beta & 1 & 0 & 1 & 0 & 0 \\ \sin\beta & 0 & 0 & 0 & 1 & 0 \\ 0 & -1 & -\cos\gamma & 0 & 0 & 0 \\ 0 & 0 & \sin\gamma & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} F_{\mathrm{A}} \\ F_{\mathrm{B}} \\ F_{\mathrm{C}} \\ H_2 \\ V_2 \\ V_3 \end{bmatrix} = \begin{bmatrix} 0 \\ F_{\mathrm{load}} \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Part c. Using the script, this gives:

```
[[-500.        ]
 [ 433.01270189]
 [-866.02540378]
 [   0.        ]
 [ 250.        ]
 [ 750.        ]]
```

File attached here.

```python
import numpy as np

def get_A(beta, gamma):
    """Returns the A matrix given angles beta and gamma."""
    cos = np.cos
    sin = np.sin
```

```
    A = [[-cos(beta), 0., cos(gamma), 0., 0., 0.],
         [-sin(beta), 0., -sin(gamma), 0., 0., 0.],
         [cos(beta), 1., 0., 1., 0., 0.],
         [sin(beta), 0., 0., 0., 1., 0.],
         [0., -1., -cos(gamma), 0., 0., 0.],
         [0., 0., sin(gamma), 0., 0., 1.]]
    return np.array(A)

def get_b(F_load):
    """Returns the b vector given the load force."""
    b = [[0., F_load, 0., 0., 0., 0.]]
    return np.array(b).T

F_load = 1000.  # N
alpha = 90. / 180. * np.pi
beta = 30. / 180. * np.pi
gamma = 60. / 180. * np.pi

A = get_A(beta, gamma)
b = get_b(F_load)
x = np.linalg.solve(a=A, b=b)
print(x)
```

Part d. Given a value of γ, we first need a formula to find β. The constraint is that the height if fixed. If we take the distance between 2 and 3 to be 1 unit, we can work through the trigonometry to find the height at 1 is $\sqrt{3}/4$.

Writing a script to solve this, we find the maximum angle is 65.2°.

File attached here.

```
import numpy as np
from scipy.optimize import newton

def get_A(beta, gamma):
    """Returns the A matrix given angles beta and gamma."""
    cos = np.cos
    sin = np.sin
    A = [[-cos(beta), 0., cos(gamma), 0., 0., 0.],
         [-sin(beta), 0., -sin(gamma), 0., 0., 0.],
         [cos(beta), 1., 0., 1., 0., 0.],
         [sin(beta), 0., 0., 0., 1., 0.],
         [0., -1., -cos(gamma), 0., 0., 0.],
         [0., 0., sin(gamma), 0., 0., 1.]]
    return np.array(A)

def get_b(F_load):
    """Returns the b vector given the load force."""
    b = [[0., F_load, 0., 0., 0., 0.]]
    return np.array(b).T

def get_beta(gamma):
    """Returns the value of angle gamma given a value of beta."""
    h = np.sqrt(3.) / 4.
    x = h / np.tan(gamma)
```

```python
    y = 1. - x
    return np.arctan(h / y)

def get_V3(gamma):
    beta = get_beta(gamma)
    A = get_A(beta, gamma)
    b = get_b(F_load=1000.)
    x = np.linalg.solve(a=A, b=b)
    V3 = x[-1]
    return V3

def get_residual(gamma):
    gamma = float(gamma)
    V3max = 800.
    V3 = get_V3(gamma)
    return V3 - V3max


# Just check that it works for our original system.
gamma = 61. * np.pi / 180.
V3 = get_V3(gamma)
print(V3)

# Solve it!
gamma = newton(func=get_residual, x0=60. * np.pi / 180.)
print(gamma * 180 / np.pi)
V3 = get_V3(gamma[0])
print(V3)
```

Rubric

- Part a. 4 points.
  - -1 Small sign error(s) or similar. Max 2 deductions if equations otherwise sound.
- Part b. 4 points.
  - -1 Sign error.
  - -1 Missing or wrong term. (Max two deductions of missing/sign/wrong.)
  - -2 $\underline{\mathbf{b}}$ not provided.
  - -1 $\underline{\mathbf{b}} = \underline{\mathbf{0}}$ (missing F_{load})
- Part c. 4 points.
  - -1 Sign error. (Not taken twice if already taken in a/b.)
  - 2/4 Right approach, wrong numbers.
  - 1/4 Attempted (incorrect) hand solution; didn't use numerical approach.
- Part d. 8 points.
  - 4/8 Right approach, wrong numbers.
  - 2/8 Poorly documented / hard-to-follow approach with wrong numbers, but using nonlinear solver.
  - 2/8 Only solved for height correctly.
  - -1 Small sign / conversion error.