HW2

July 5, 2019

1 EE 263: HW2 Notebook

Author: Luis Perez **Last Updated**: July 3, 2019

1.1 Imports

[1]:
```
import numpy as np
from scipy import linalg

from typing import List, Optional
```

1.2 Problem 5: Single sensor failure detection and identification

[2]:
```
"""
Define A and ytilde.
"""
A = np.array([
   [1.164953510500657, -0.360029625711573, 0.375041023696104, -0.
 ↪557093642241282, -0.507700386669636,0.924159404893175,0.039884852845797, -0.
 ↪620214209475792,0.438705097860831,0.670291996969230],
   [0.626839082632431, -0.135576294466487, 1.125161817875028, -0.
 ↪336705699002853,0.885299448191509, -1.814114702851241, -2.482842514256541,0.
 ↪237148765008739, -1.247344316401997,0.420146041651794],
   [0.075080154677683, -1.349338480385175, 0.728641591773905,0.415227462723156,␣
 ↪-0.248093553237236,0.034973320285167,1.158654705247901, -1.586846990031033,0.
 ↪324666916936102, -2.872751269668520],
   [0.351606902768522, -1.270449896283403, -2.377454293765433,1.557813537123208,␣
 ↪-0.726248999742084, -1.807862060321251, -1.026279466693260, -0.
 ↪401484809800359,0.390070410090458,1.685874080406989],
   [-0.696512535163682,0.984570272925253, -0.273782415743900, -2.
 ↪444298897865560, -0.445040300996161,1.028192546045777,1.153486988237923, -0.
 ↪770692268923938, -0.405138316773605,0.027924553523994],
   [1.696142480747077, -0.044880613828856, -0.322939921204497, -1.
 ↪098195387799324, -0.612911120338436,0.394600308811932, -0.786456613020222,␣
 ↪-0.262680506066512,0.292314877283450, -0.902030581228208],
  [0.059059777981351, -0.798944516671106,0.317987915650739,1.122647857944875,␣
↪-0.209144084593638,0.639405642088516,0.634808587961936,0.976489543659970,2.
↪565910242123806, -2.053257491526201],
  [1.797071783694818, -0.765172428787515, -0.511172207780700,0.
↪581667258045274,0.562147834450359,0.874212894863609,0.820409761532064,0.
↪977815041129280, -0.457815643580367,0.089086297675464],
  [0.264068528817227,0.861734897324192, -0.002041345349433, -0.271354295524753,␣
↪-1.063922887881042,1.752401730329559, -0.176026510455600,1.170021110265055,␣
↪-1.610827014289158,2.087099131649750],
  [0.871673288690637, -0.056225124358975,1.606510961119237,0.414191307229504,0.
↪351588948379816, -0.320050826432138,0.562473874646301,0.159310862415417, -2.
↪669523782410902,0.365118460310679],
  [-1.446171539339335,0.513478173674302,0.847648634500925, -0.977814227461400,1.
↪132999926008681, -0.137413808144866, -0.127442875395491,0.499520851464531,␣
↪-0.759696648513815,0.846105526166482],
  [-0.701165345682908,0.396680865935824,0.268100811901575, -1.021466173866152,0.
↪149994248007729,0.615769628086716,0.554171560978313, -1.055375070659330, -0.
↪674720856431937, -0.184537657075523],
  [1.245982120437819,0.756218970285488, -0.923489085784077,0.317687979852042,0.
↪703144053247466,0.977894069845197, -1.097344319221644, -0.450743202815186,␣
↪-1.171687194533551,1.030714423869546],
  [-0.638976995013557,0.400486023191097, -0.070499387778694,1.516107798150034,␣
↪-0.052411584998689, -1.115347712205141, -0.731301400074801,1.
↪270378242169987,2.032930016155204, -1.527622652429381],
  [0.577350218771609, -1.341380722378574,0.147891351014747,0.749432452588256,2.
↪018496124007770, -0.550021448804486,1.404731919616814,0.898693600923036,0.
↪968481047964462,0.964938959209115]])
ytilde = np.array([
  [0.293700010391366],
  [-0.548030198505630],
  [0.003532110461013],
  [1.375859546156174],
  [-6.752682998496523],
  [1.190484875889765],
  [8.782196150345506],
  [1.911972855063559],
  [-1.462868211077097],
  [-4.433624854460799],
  [-1.723404706120404],
  [-4.547493026790328],
  [-0.109245813786955],
  [8.033526684801210],
  [1.782619515709060]])
```

[3]:
```
'''
To calculate whether ytilde is achievable from A, we compute the rank
of [A ytilde] and see if this matches the rank of A.
```

```
If it does, this means ytilde is a linear combination of A and therefore
no sensors failed (or if they did, it is impossible to tell).

If the rank incrases by one, this means ytilde is linearly independent
and
'''
def isInSpan(mat: np.array, v: np.array) -> bool:
    """Returns true if v is in the span of the columns of mat."""
    assert mat.shape[0] == v.shape[0]
    return (np.linalg.matrix_rank(mat) ==
            np.linalg.matrix_rank(np.concatenate((mat,v), axis=1)))
```

[4]:
```
def findFailingSensor(stateMatrix: np.array, measurement: np.array) ->␣
 ↪Optional[int]:
    for i in range(0, stateMatrix.shape[0]):
        # Drop the i-th row.
        newStateMatrix = np.delete(stateMatrix, i, axis=0)
        newMeasurement = np.delete(measurement, i, axis=0)
        if isInSpan(newStateMatrix, newMeasurement):
            return i
    return None
```

[5]:
```
if isInSpan(A, ytilde):
    print("No sensors failed")
else:
    print("The ranks are not equal. The failing sensor is sensor "
          "%i." % findFailingSensor(A, ytilde))
```

```
The ranks are not equal. The failing sensor is sensor 10.
```

1.3 Proble 6: Reverse engineering a smoothing filter

1.3.1 Part b

[6]:
```
"""
Load the provided y and u vectors.
"""
# Data for reverse engineering smoothing filter problem
n = 150
# Input vector u
u = np.array([
   -2.755845e+00,
   -9.234193e+00,
   -5.290089e-01,
   1.284874e-01,
   -7.076616e+00,
   4.712192e+00,
    4.948501e+00,
    -8.001869e-01,
    1.537274e+00,
    1.392752e+00,
    2.815396e-01,
    5.581623e+00,
    -2.472801e-01,
    1.431712e+01,
    3.354382e+00,
    5.136780e+00,
    1.030113e+01,
    5.515362e+00,
    4.835518e+00,
    1.091770e+00,
    6.522234e+00,
    -1.955776e+00,
    7.879493e+00,
    1.195192e+01,
    -1.153684e-01,
    7.167036e+00,
    8.744386e+00,
    -5.797513e+00,
    -5.208255e+00,
    4.827247e+00,
    1.074556e-01,
    5.852822e+00,
    6.926662e+00,
    6.982320e+00,
    1.054629e+01,
    8.170432e+00,
    1.153113e+01,
    2.858415e-01,
    6.847011e+00,
    6.695114e+00,
    -1.598211e-01,
    9.350795e+00,
    2.790135e+00,
    1.495523e+01,
    3.466091e+00,
    9.570246e+00,
    7.309861e+00,
    7.801549e-01,
    -6.352747e+00,
    3.299334e+00,
    -2.329165e+00,
    5.007705e+00,
    3.811868e+00,
    9.235628e+00,
    3.419012e+00,
    -2.862788e+00,
    2.354979e+00,
    -4.299488e+00,
    1.112873e+00,
    1.571023e+00,
    2.508189e+00,
    1.658975e+00,
    9.454843e+00,
    -4.720811e+00,
    7.346612e+00,
    1.008322e+01,
    9.466253e+00,
    8.689694e+00,
    5.761411e+00,
    8.476640e+00,
    7.253836e+00,
    2.263539e+00,
    6.397091e-01,
    -6.685378e-02,
    -7.123909e+00,
    -2.069139e+00,
    -1.392026e+00,
    -1.377567e+00,
    3.461468e+00,
    -6.116954e+00,
    -1.629606e+00,
    -9.011609e-01,
    -1.124921e-01,
    -9.484350e+00,
    -2.986092e+00,
    -2.234419e+00,
    -7.743486e+00,
    -5.652527e+00,
    4.219030e+00,
    -1.167148e+00,
    2.005359e+00,
    9.016095e-01,
    -2.502557e+00,
    -2.123466e+00,
    2.671089e+00,
    -4.733731e+00,
    3.288058e+00,
    1.427455e+00,
    -6.450545e+00,
    -4.671544e+00,
   -1.030808e+01,
   -1.637650e+01,
   -1.346730e+00,
   -9.655710e+00,
   -6.041011e+00,
   -6.924759e+00,
   -8.189228e+00,
   -1.329723e+01,
   -1.278159e+01,
   -9.493896e+00,
   -1.296561e+01,
   -1.160846e+01,
   1.816272e+00,
   -4.458492e+00,
   -1.003858e+01,
   -3.443663e+00,
   -8.277031e+00,
   -8.925164e+00,
   -3.196868e+00,
   2.982124e+00,
   -1.217341e+00,
   1.176362e+00,
   -8.338238e+00,
   -5.311512e+00,
   -4.907473e+00,
   -1.028190e+01,
   -1.142310e+01,
   -7.243588e-01,
   -5.783221e+00,
   -9.394667e+00,
   -2.314705e+00,
   -5.188887e+00,
   -1.112385e+01,
   8.644223e-01,
   -3.891328e+00,
   2.677864e+00,
   1.689869e+00,
   -6.648518e+00,
   -8.900073e+00,
   -2.043615e+00,
   -2.634938e+00,
   -4.600642e+00,
   2.582165e+00,
   7.870772e+00,
   -2.117235e+00,
   -3.577158e+00,
   -6.661662e-01,
    3.678589e+00,
    -4.191429e+00,
    -6.258337e+00,
])
# output vector y
y = np.array([
    -3.853013e+00,
    -3.288960e+00,
    -2.463849e+00,
    -1.626045e+00,
    -8.032515e-01,
    1.655397e-01,
    8.512323e-01,
    1.221120e+00,
    1.580551e+00,
    2.018439e+00,
    2.619256e+00,
    3.417308e+00,
    4.226382e+00,
    5.044730e+00,
    5.382827e+00,
    5.531889e+00,
    5.488654e+00,
    5.116571e+00,
    4.639678e+00,
    4.249766e+00,
    4.129210e+00,
    4.155957e+00,
    4.435427e+00,
    4.470849e+00,
    4.016698e+00,
    3.432931e+00,
    2.600209e+00,
    1.688483e+00,
    1.440320e+00,
    1.965514e+00,
    2.849269e+00,
    4.015022e+00,
    5.139932e+00,
    6.036621e+00,
    6.599342e+00,
    6.689755e+00,
    6.403762e+00,
    5.843154e+00,
    5.503495e+00,
    5.310380e+00,
    5.298029e+00,
    5.622181e+00,
    5.907113e+00,
    6.085899e+00,
    5.699612e+00,
    5.002385e+00,
    3.905815e+00,
    2.648394e+00,
    1.737788e+00,
    1.537719e+00,
    1.727586e+00,
    2.225558e+00,
    2.588475e+00,
    2.602124e+00,
    2.078894e+00,
    1.361129e+00,
    8.654841e-01,
    6.170401e-01,
    8.306103e-01,
    1.315588e+00,
    1.955570e+00,
    2.677537e+00,
    3.460578e+00,
    4.167363e+00,
    5.210131e+00,
    6.137828e+00,
    6.637930e+00,
    6.645558e+00,
    6.213789e+00,
    5.445778e+00,
    4.270059e+00,
    2.782248e+00,
    1.271225e+00,
    -5.846890e-02,
    -1.041890e+00,
    -1.445095e+00,
    -1.515877e+00,
    -1.483501e+00,
    -1.529056e+00,
    -1.823789e+00,
    -2.074439e+00,
    -2.390006e+00,
    -2.827248e+00,
    -3.294471e+00,
    -3.406231e+00,
    -3.292033e+00,
    -2.960117e+00,
    -2.236495e+00,
    -1.317574e+00,
    -6.799411e-01,
    -2.896032e-01,
    -2.039481e-01,
    -3.089077e-01,
    -4.159420e-01,
    -5.532013e-01,
    -9.214742e-01,
    -1.439791e+00,
    -2.429201e+00,
    -3.817791e+00,
    -5.203525e+00,
    -6.408893e+00,
    -7.115080e+00,
    -7.229261e+00,
    -7.395205e+00,
    -7.606486e+00,
    -8.017846e+00,
    -8.591427e+00,
    -9.132330e+00,
    -9.312988e+00,
    -9.058948e+00,
    -8.462551e+00,
    -7.500571e+00,
    -6.442345e+00,
    -5.873745e+00,
    -5.588280e+00,
    -5.235828e+00,
    -4.842011e+00,
    -4.192608e+00,
    -3.317359e+00,
    -2.632168e+00,
    -2.545724e+00,
    -2.998718e+00,
    -3.881429e+00,
    -4.722590e+00,
    -5.449484e+00,
    -5.978213e+00,
    -6.076545e+00,
    -5.796759e+00,
    -5.589399e+00,
    -5.354286e+00,
    -4.949179e+00,
    -4.548334e+00,
    -4.013921e+00,
    -3.199966e+00,
    -2.575441e+00,
    -2.208767e+00,
    -2.325768e+00,
    -2.738253e+00,
    -2.892306e+00,
    -2.545969e+00,
    -1.929059e+00,
    -1.141584e+00,
    -3.007348e-01,
    1.524805e-01,
    -9.174533e-03,
    -3.656063e-01,
    -7.463907e-01,
    -1.303474e+00,
    -2.208574e+00,
    -3.191773e+00,
])
```

[7]:
```
def makeMatrix(start):
    """Makes a 3x3 submatrix B as defined in the homework handout."""
    assert start > 2
    res = np.zeros((3,3))
    for i in range(3):
        res[i][0] = y[start + i]
        res[i][1] = (- y[start + i + 1]
                     + 2*y[start + i]
                     - y[start + i - 1])
        res[i][2] = (y[start + i + 2]
                     - 4*y[start + i + 1]
                     - 6*y[start + i]
                     - 4*y[start + i - 1]
                     + y[start + i - 2])
    return res
```

[8]:
```
avgResult = np.zeros(3, dtype='float')
count = 0
for START in range(3,y.shape[0] - 4):
    C = makeMatrix(START)
    count += 1
    avgResult += np.dot(np.linalg.inv(C), (u - y)[START:START + 3])
avgResult /= count
avgResult
```

[8]:
```
array([120.09865793,   2.00010201,   9.99988969])
```

1.4 Problem 9: Zeroing out the board

1.4.1 Part (a)

[9]:
```
def getAllVectors(dimension: int) -> List[np.array]:
    """
    Returns all 'action' vectors for the given dimension.
    """
    basis = []
    for i in range(dimension):
        for j in range(dimension):
            M = np.zeros((dimension, dimension))
            M[i][j] = 1.0
            if i > 0: M[i - 1][j] = -1.0
            if j > 0: M[i][j - 1] = -1.0
            if i < dimension - 1: M[i + 1][j] = -1.0
            if j < dimension - 1: M[i][j + 1] = -1.0
            basis.append(M.flatten())
    return basis
```

[10]:
```
# Since the rank is 36, this means that this vectors span the
# entire space, so any initial vector can be zero-ed out.
A = np.vstack(getAllVectors(6)).T
np.linalg.matrix_rank(A)
```

[10]:
```
36
```

1.4.2 Part (b)

[11]:
```
# We now figure out how to zero it out by solving the
# equation x = A^{-1}y
y = np.zeros((A.shape[1], 1))
y[0] = 1
x = np.dot(np.linalg.inv(A), y)
```

[12]:
```
x
```

[12]:
```
array([[ 1.30769231e+00],
       [ 1.53846154e-01],
       [-6.92307692e-01],
       [-4.61538462e-01],
       [ 7.69230769e-02],
       [ 2.30769231e-01],
       [ 1.53846154e-01],
       [-4.61538462e-01],
       [-3.84615385e-01],
       [ 1.53846154e-01],
       [ 3.07692308e-01],
       [ 1.53846154e-01],
       [-6.92307692e-01],
       [-3.84615385e-01],
       [ 6.15384615e-01],
       [ 6.92307692e-01],
       [-7.69230769e-02],
       [-3.84615385e-01],
       [-4.61538462e-01],
       [ 1.53846154e-01],
       [ 6.92307692e-01],
       [-2.31957311e-16],
       [-6.92307692e-01],
       [-4.61538462e-01],
       [ 7.69230769e-02],
       [ 3.07692308e-01],
       [-7.69230769e-02],
       [-6.92307692e-01],
       [-1.53846154e-01],
       [ 6.15384615e-01],
       [ 2.30769231e-01],
       [ 1.53846154e-01],
       [-3.84615385e-01],
       [-4.61538462e-01],
       [ 6.15384615e-01],
       [ 1.23076923e+00]])
```

1.4.3 Part (c)

[13]:
```
# We expect a rank of 81, but only receive a rank of 79.
# This means that there exists a nullspace of dimension 2.
# Any vectors in this null-space cannot be zeroed out.
B = np.vstack(getAllVectors(9)).T
np.linalg.matrix_rank(B)
```

[13]:
```
79
```

[14]:
```
kernel = linalg.null_space(B)
# Explicitly set a few values to zero.
kernel[kernel < 0.00001] = 0
```

[15]:
```
# Our first kernel basis. This is not solvable by Reza.
(kernel[:,0] / kernel[:,0][6]).reshape((9, 9))
```

[15]:
```
array([[0., 0., 0., 0., 0., 2., 1., 1., 2.],
       [0., 2., 2., 0., 0., 1., 0., 0., 1.],
       [0., 2., 2., 0., 0., 1., 0., 0., 1.],
       [0., 0., 0., 0., 0., 2., 1., 1., 2.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [2., 1., 1., 2., 0., 0., 0., 0., 0.],
       [1., 0., 0., 1., 0., 0., 2., 2., 0.],
       [1., 0., 0., 1., 0., 0., 2., 2., 0.],
       [2., 1., 1., 2., 0., 0., 0., 0., 0.]])
```

[16]:
```
# Our second kernel basis, also not solvable by Reza.
(kernel[:,1] / kernel[:,1][1]).reshape((9, 9))
```

[16]:
```
array([[0., 1., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0., 0., 1.],
       [0., 0., 0., 0., 0., 1., 0., 0., 1.],
       [0., 1., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 1., 0.],
       [1., 0., 0., 1., 0., 0., 0., 0., 0.],
       [1., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 1., 0.]])
```