Computational Linear Algebra Coursework 1

_

CID:01724711

Note: Code regarding question {i} is found in cw1/coursework1q{i}.py .Testing my code for question {i} can be achieved by running pytest cw1/coursework1q{i}.py

Question 1

_

1a

In [21]:

```
import numpy as np
#verified C is correct shape
C = np.loadtxt("/Users/nikolaikrokhin/Documents/comp-lin-alg-course/cw1/C.dat", delimiter = ',')
print(C.shape)
```

```
(1000, 100)
```

We load R from the householder decomposition of C, noting that R from GS_modified gives the exact save values in R.

In [78]:

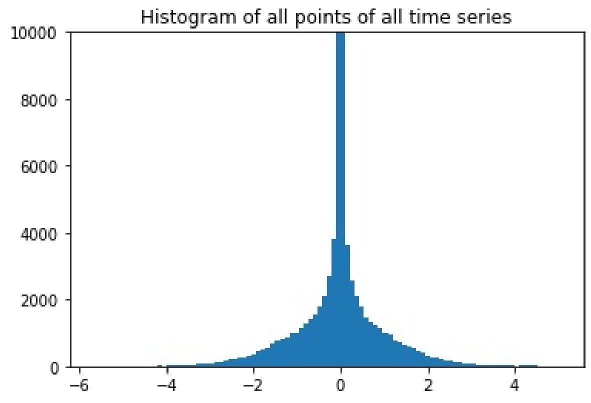
```
R = np.loadtxt("/Users/nikolaikrokhin/Documents/comp-lin-alg-course/file1.txt")
print(R)
print("-------------------------------------------")
print(np.around(R, 35)) #only first 3 three rows non zero, rounded to 35 d.p
```

```
[[ 1.82021454e-04  3.31705075e-04  5.96154073e-04 ...  5.36544596e-04
   2.99187724e-04  1.64520629e-04]
 [-0.00000000e+00  1.31010697e-06  4.80547728e-06 ... -2.22675999e-04
  -1.22782497e-04 -6.67865866e-05]
 [-0.00000000e+00 -0.00000000e+00  1.03490007e-08 ...  4.81849472e-05
   2.62795521e-05  1.41436438e-05]
 ...
 [-0.00000000e+00 -0.00000000e+00  7.52316385e-37 ... -1.17549435e-38
  -1.17549435e-38  1.17549435e-38]
 [-0.00000000e+00 -0.00000000e+00 -0.00000000e+00 ...  2.05711511e-38
   2.35098870e-38 -2.35098870e-38]
 [-0.00000000e+00 -0.00000000e+00 -0.00000000e+00 ...  1.76324153e-38
   1.46936794e-38 -1.76324153e-38]]
-------------------------------------------
[[ 1.82021454e-04  3.31705075e-04  5.96154073e-04 ...  5.36544596e-04
   2.99187724e-04  1.64520629e-04]
 [-0.00000000e+00  1.31010697e-06  4.80547728e-06 ... -2.22675999e-04
  -1.22782497e-04 -6.67865866e-05]
 [-0.00000000e+00 -0.00000000e+00  1.03490007e-08 ...  4.81849472e-05
   2.62795521e-05  1.41436438e-05]
 ...
 [-0.00000000e+00 -0.00000000e+00  0.00000000e+00 ... -0.00000000e+00
  -0.00000000e+00  0.00000000e+00]
 [-0.00000000e+00 -0.00000000e+00 -0.00000000e+00 ...  0.00000000e+00
   0.00000000e+00 -0.00000000e+00]
 [-0.00000000e+00 -0.00000000e+00 -0.00000000e+00 ...  0.00000000e+00
   0.00000000e+00 -0.00000000e+00]]
```

Firstly, we notice that only the first three rows are essentially non-zero (a tolerance of 35 decimal points). This leads to the observation that R has rank 3 (since it is upper triangular) which using the theorem $rank(AB) \leq rank(A)rank(B)$ means $rank(C) \leq 3$. We know that the rank of a matrix is equal to the number of linear independent rows in that matrix, here it is just 3 which is very small compared to the number of time series. Due to their dependency on one another we may suggest that each time series from this experiment is a 100 long sample from a single distribution. We plot a histogram of all the time series points along with individual time series histograms and clearly see that we have a sample from the same normal distribution in each one , justifying this suggestion. i.e the possible variations in the sample come from a normal distribution.

In [79]:

```
import matplotlib.pyplot as plt
plt.hist(C.reshape(100000,1), bins=100)
plt.ylim(0,10000)
plt.title("Histogram of all points of all time series")
plt.show()
```



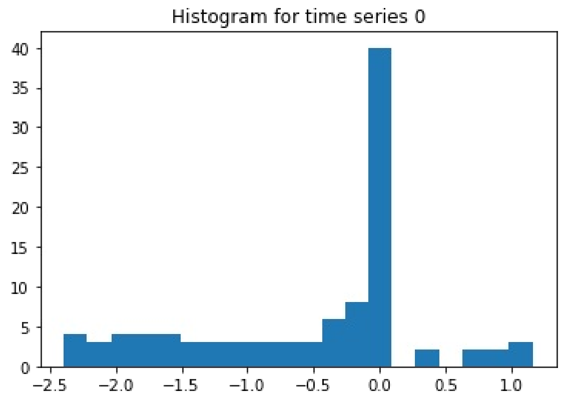
In [64]:

```
m = np.mean(C.reshape(100000,1))
v = np.var(C.reshape(100000,1))
print("Sample mean and variance", (m, v))
```

```
Sample mean and variance (-0.015615344852206963, 0.6730238484035198)
```

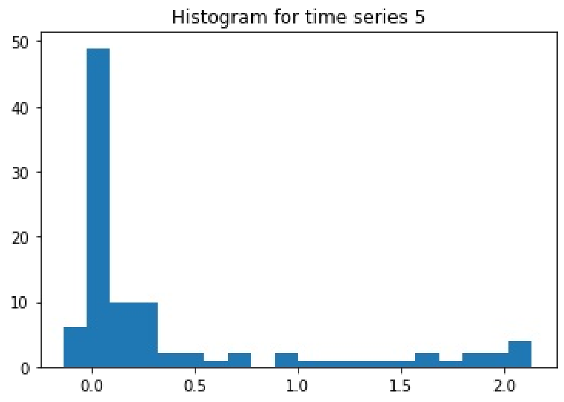
In [77]:

```
plt.hist(C[0],bins=20)
plt.title("Histogram for time series 0")
plt.show()
#as expected from calculating the sample mean and variance of the normal distribution we see large peaks around 0
```



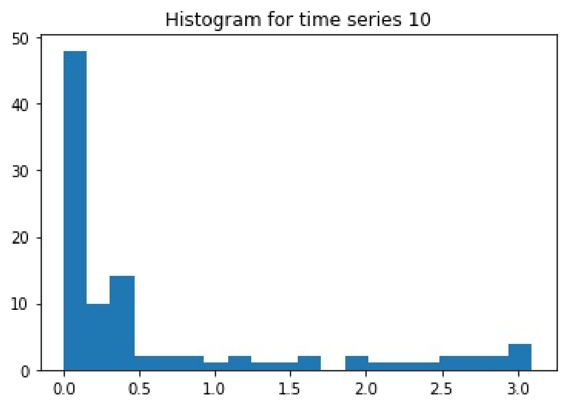
In [76]:

```
plt.hist(C[5],bins=20)
plt.title("Histogram for time series 5")
plt.show()
```



In [138]:

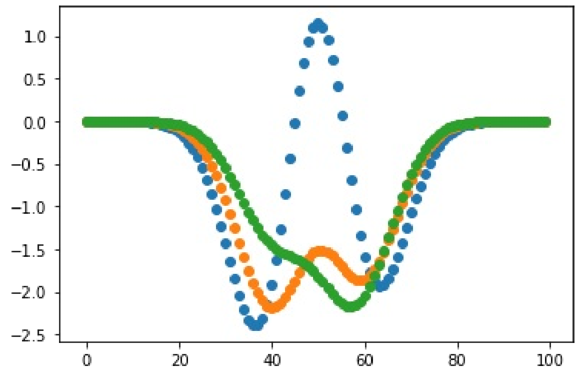
```
plt.hist(C[10],bins=20)
plt.title("Histogram for time series 10")
plt.show()
```



We also investigate the time series themselves to figure out how they we created.

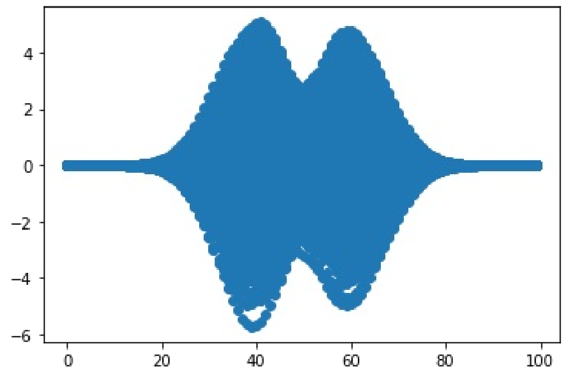
In [88]:

```
plt.scatter(y = C[0], x = [i for i in range(100)])
plt.scatter(y = C[1], x = [i for i in range(100)])
plt.scatter(y = C[2], x = [i for i in range(100)])
plt.show()
```



In [85]:

```
plt.scatter(y = C.reshape(1,100000), x = [i for i in range(100)]*1000)
plt.show()
```



We see that the the time series are themselves bounded by 4 normal looking curves. Perhaps this is because each time series is created by adding creating a curve bounded by these normal curves and then taking points equally spaced along it.

1b

Since the only non-zero rows of R are the first three it means that we can compress the 100000 entries of C into the product of the first three columns of Q (3000 entries) and the product of the first three rows of R (300) entries. Total entries are 3300 which is by far less than storing C in its normal form. Code is found in cw1/coursework1q1.py.

1c

To make it more obvious that R has three non zero rows at the top and then zeros elsewhere we may transform C by writing it as the following sum: $C = q_1 r_1^* + q_2 r_2^* + q_3 r_3^*$ where q_i are the columns of Q and r_i are the rows of R. This would not affect the compression since we are still storing 3300 elements on the rhs of the equation. It is not immediately obvious that the Q, R decomposition would be as we know it to be since we would have to check certain requirements such as orthonormality of the q_i etc. So for the compression method we would have to compute the sum of the outer products and then do the compression method to be sure that the vectors given in the sum are from the Q, R decomposition. But it would certainly make it more obvious to look at the first three rows of R and observe the rest are zero.

Question 2

–

2a

We use the vandermonde matrix to represent our interpolation equation in matrix form, noting that we replace a_{13} to a_{51} with 0s since we only seek to get a degree twelve polynomial.

$$\begin{bmatrix} 1 & x_0 & \dots & x_0^{51} \\ \vdots & \vdots & & \vdots \\ 1 & x_{51} & \dots & x_{51}^{51} \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_{12} \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

By using the column space formulation we may simplify this to

$$\begin{bmatrix} 1 & x_0 & \dots & x_0^{12} \\ \vdots & \vdots & & \vdots \\ 1 & x_{51} & \dots & x_{51}^{12} \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_{12} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

and since the matrix we are decomposing is smaller we will increase the decomposition speed.

Using the code in cw1/coursework1q2.py we obtain our coefficients in reverse degree order.

2b

In [133]:

```
print("GS",np.loadtxt("/Users/nikolaikrokhin/Documents/comp-lin-alg-course/q2file1.txt"))
print("GSM",np.loadtxt("/Users/nikolaikrokhin/Documents/comp-lin-alg-course/q2file2.txt"))
print("HH",np.loadtxt("/Users/nikolaikrokhin/Documents/comp-lin-alg-course/q2file3.txt"))
```

```
GS [ 1.20833971e+00 -9.23186760e+01  1.90321678e+03 -1.69784929e+04
  7.91606960e+04 -2.09578613e+05  3.20620268e+05 -2.66460792e+05
  8.72849283e+04  2.17027130e+04 -1.93003798e+04  1.53834341e+03
  1.99122713e+02]
GSM [ 9.41937157e-01 -5.93211599e+01  1.23068468e+03 -1.18303369e+04
  5.84556979e+04 -1.35379130e+05 -1.25374487e+04  9.42390781e+05
 -2.67375435e+06  3.85446639e+06 -3.17819533e+06  1.42614687e+06
 -2.70935320e+05]
HH [ 9.41937158e-01 -5.93211603e+01  1.23068470e+03 -1.18303373e+04
  5.84557018e+04 -1.35379151e+05 -1.25373719e+04  9.42390602e+05
 -2.67375407e+06  3.85446610e+06 -3.17819514e+06  1.42614680e+06
 -2.70935308e+05]
```

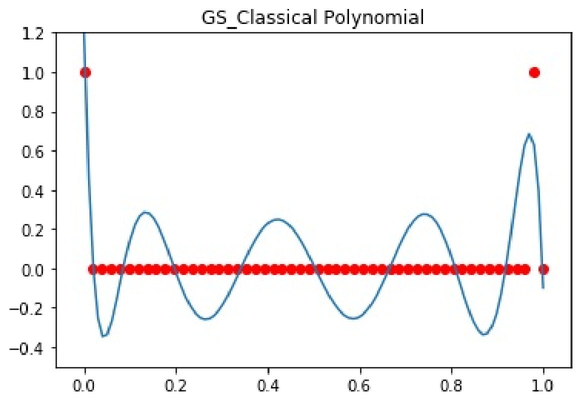
We now plot the resulting polynomials to see if we can see any visual differences.

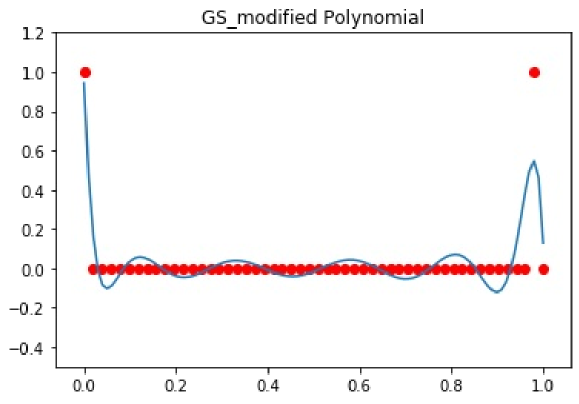
In [95]:

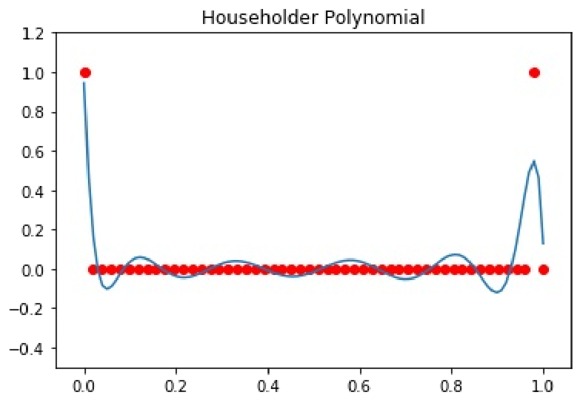
```
from matplotlib import pyplot as plt

def PolyCoefficients(x, coeffs):
    o = len(coeffs)
    y = 0
    for i in range(o):
        y += coeffs[i]*x**i
    return y

x = np.linspace(0, 1, 100)
coeffs1 = np.loadtxt("/Users/nikolaikrokhin/Documents/comp-lin-alg-course/q2file1.txt")
plt.plot(x, PolyCoefficients(x, coeffs1))
plt.scatter(np.arange(0., 1.0001, 1./51),np.array([1] + [0] * 49 + [1] + [0]), color = 'red')
plt.ylim(-0.5, 1.2)
plt.title("GS_Classical Polynomial")
plt.show()
coeffs2 = np.loadtxt("/Users/nikolaikrokhin/Documents/comp-lin-alg-course/q2file2.txt")
plt.plot(x, PolyCoefficients(x, coeffs2))
plt.scatter(np.arange(0., 1.0001, 1./51),np.array([1] + [0] * 49 + [1] + [0]), color = 'red')
plt.ylim(-0.5, 1.2)
plt.title("GS_modified Polynomial")
plt.show()
coeffs3 = np.loadtxt("/Users/nikolaikrokhin/Documents/comp-lin-alg-course/q2file3.txt")
plt.plot(x, PolyCoefficients(x, coeffs3))
plt.scatter(np.arange(0., 1.0001, 1./51),np.array([1] + [0] * 49 + [1] + [0]), color = 'red')
plt.ylim(-0.5, 1.2)
plt.title("Householder Polynomial")
plt.show()
```







We observe that the coefficients in GSM and HH are very similar with a maximum difference of 2.90995322e-01 which occurs at coefficient 10 and an average difference of -2.7115309997327586e-11, hence why the graphs look so similar. GS coefficients are quite different to the other two, when comparing GS and HH we have an average difference of -0.0175434960043416 for a coefficient and a maximum difference of -3.83276338e+06 which occurs at the 10th coefficient and the minimum difference is 2.66402555e-01 occuring at the the first coefficient. It makes sense that the first coefficient is the most similar of them all for all the coefficient sets since this is the y intercept and that has to be close to 1, which differs to the vast majority of other points and so is most influencing, since that is the point being interpolated at x = 0. GS polynomial is actually not to different in shape from the others, just more stretched out vertically and it crosses the x axis two less times. We observe that the size of the difference in the cofficients relates to the differences in the corresponding (by coefficient index) column and row of Q and R between two methods.

In [137]:

```
GSQ = np.loadtxt("/Users/nikolaikrokhin/Documents/comp-lin-alg-course/q2file4.txt")
GSMQ = np.loadtxt("/Users/nikolaikrokhin/Documents/comp-lin-alg-course/q2file6.txt")
HHQ = np.loadtxt("/Users/nikolaikrokhin/Documents/comp-lin-alg-course/q2file8.txt")
GSR = np.loadtxt("/Users/nikolaikrokhin/Documents/comp-lin-alg-course/q2file5.txt")
GSMR = np.loadtxt("/Users/nikolaikrokhin/Documents/comp-lin-alg-course/q2file7.txt")
HHR = np.loadtxt("/Users/nikolaikrokhin/Documents/comp-lin-alg-course/q2file9.txt")
#print(GSMQ[:13, :13] - HHQ[:13, :13], GSMR[:13, :13] - HHR[:13, :13])
#print(GSQ[:13, :13] - HHQ[:13, :13], GSR[:13, :13] - HHR[:13, :13])
#print(GSMQ[:13, :13] - GSQ[:13, :13], GSMR[:13, :13] - GSR[:13, :13])
#unhash to see the differences
```

Question 3

–

3a

If we are using the householder method then our Q will be of dimension mxm. But we want the reduced Q which is of dimension mxn. If $m \gg n$ then we are having to discard a large number (m - n) of columns to get from full Q to reduced Q. So it would be more efficient to use Gram-Schmidt Modified since this does not give any extra number of columns that we will then have to discard.

3b

Let our A be mxn dimensional. When we are calculating the reflectors at iteration k we are using the outer product of v where v is size m - k + 1. However after multiplying our A on the left by Q_k we introduce zeros under the diagonal of length m - k, which is not big enough to store v. But we can store $v[1:]$ in these zeros and recover the v_0 using the following. We know by construction that $R[k,k]$ = sign(x_0) ||x||, so we use the definition of the modulus to get $x_0 =$ -sign($R[k,k]$)$\sqrt{R[k,k]^2 - (x_1^2 + \dots + x_n^2)}$ = -sign($R[k,k]$)$\sqrt{R[k,k]^2 - (v_1^2 + \dots + v_n^2)}$ (since v and x only differ in the first value) which we all have stored in the k^{th} column of R. Then we have $v_0 =$ -sign($R[k,k]$)||x|| + x_0. We now can construct our v. We may use these vs for example to find Q^*b by applying the equivalent operations to b as multiplying by these householder matrices.

Question 4

–

4a

Using Lagrange Multipiers we obtain $\phi(x,\lambda) = ||\mathbf{b} - \mathbf{Ax}||^2 + \lambda(||\mathbf{x}||^2 - 1)$ We may recompose this as $\phi(x,\lambda) = (\mathbf{b^T} - \mathbf{x^T A^T})(\mathbf{b}\text{-}\mathbf{Ax}) + \lambda(\mathbf{x^T x} - 1)$ We want to minimize this potential, by finding the grad function and setting to 0.

4b

$\nabla_x \phi = 0 \Rightarrow -2\mathbf{A^T b} + \mathbf{2A^T Ax} + 2\lambda\mathbf{x} = \mathbf{0} \Rightarrow \mathbf{x} = \mathbf{(A^T A + \lambda I)^{-1} A^T b}$ given that $\mathbf{(A^T A + \lambda I)}$ inverse exists.

4c

$$x = (A^TA + \lambda I_n)^{-1}A^Tb = (R^TQ^TQR + \lambda I_n)^{-1}R^TQ^Tb = (R^TR + \lambda I_n)^{-1}R^TQ^Tb =$$

$$= (\frac{1}{\lambda}I_n - \frac{1}{\lambda}I_nR^T(I_m + \frac{1}{\lambda}RR^T)^{-1}R\frac{1}{\lambda}I_m)R^TQ^Tb =$$

$$\frac{1}{\lambda}R^TQ^Tb - \frac{1}{\lambda^2}R^T(I_m + \frac{1}{\lambda}RR^T)^{-1}RR^TQ^Tb = \frac{1}{\lambda}R^T(I_m + \frac{1}{\lambda}RR^T)^{-1}(I_m + \frac{1}{\lambda}RR^T)Q^Tb - \frac{1}{\lambda^2}R^T(I_m + \frac{1}{\lambda}RR^T)^{-1}RR^TQ^Tb =$$

$$\frac{1}{\lambda}R^T(I_m + \frac{1}{\lambda}RR^T)^{-1}(Q^Tb + \frac{1}{\lambda}RR^TQ^Tb - \frac{1}{\lambda}RR^TQ^Tb) = \frac{1}{\lambda}R^T(I_m + \frac{1}{\lambda}RR^T)^{-1}Q^Tb =$$

$\frac{1}{\lambda}R^T[Q(I_m + \frac{1}{\lambda}RR^T)]^{-1}b = \frac{1}{\lambda}R^Tx'$ when we define $A' = (Q + \frac{1}{\lambda}AR^T)$ and $A'x' = b$.

By defining in this way we can use our householder_solve rather than computing inverses which is far more efficient. Under the condition that m is smaller than n then we are transforming from a nxn inverse on the lhs to a mxm inverse on the rhs which also translates in efficiency to when we use householder_solve to do this inverse since we have a smaller matrix to work with and so quicker implementation is achieved.

4d

We know $\frac{\partial}{\partial \lambda}\phi(x, \lambda) = x^T x - 1$ and so if the norm of x is greater than 1, this partial derivative is positive and so to follow the derivative up the slope and increase $\phi(x, \lambda)$ w.r.t λ, we increase λ. We have a similar argument if the norm is less than 1. Using this convexity we may build our binary search algorithm. We create an initial large interval (-10000, 100000). We input the middle value of this interval as our lambda into the function from the previous part to find x and so the norm of x (making sure to shift the middle slightly if it is 0 to avoid division error). If the norm of x is greater than 1 then our lambda is too small and so we halve our interval to be (lambda, 100000). If the norm is less than 1 we halve the interval in the other direction. We repeat in the same fashion until our lambda gives us a norm close enough to 1 for our liking.

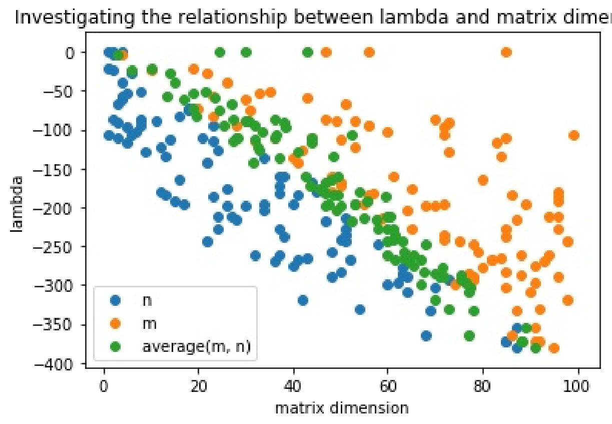
When the norm of $\mathbf{x}$ exceeds 1, this derivative is positive, so to follow the derivative uphill and increase the Lagrangian with respect to λ, we increase λ. Because the coefficient on the $\mathbf{x}T\mathbf{x}$ penalty has increased, solving the linear equation for $\mathbf{x}$ will now yield a solution with a smaller norm. The process of solving the linear equation and adjusting λ continues until $\mathbf{x}$ has the correct norm and the derivative is 0.

4e

When first implementing this code it would often break after the first iteration since my λ was so close to 0 that $\frac{1}{\lambda}$ was too large to compute the norm of this x. So to combat this I split my interval into two, one positive and one negative, and do my binary search algorithm on both of these. This does not affect the time complexity of my function. I have noticed that my λs are always negative. I think this is to do with when I call my householder function in lagrange_ls my Q and R are the negatives of Q and R when I call GS_modified. I also noticed that larger matrices would result in more negative lambda in than smaller matrices, so in the following graphs I will investigate the relation between the matrix dimensions and lambda.

In [53]:

```python
y = np.loadtxt("/Users/nikolaikrokhin/Documents/comp-lin-alg-course/q4data11.txt")
x = np.loadtxt("/Users/nikolaikrokhin/Documents/comp-lin-alg-course/q4data12.txt")
m = x.T[0]
n = x.T[1]
plt.scatter(m, y, label = "n")
plt.scatter(n, y, label = "m")
plt.scatter((m+n)/2, y, label = "average(m, n)")
plt.legend()
plt.xlabel("matrix dimension")
plt.ylabel("lambda")
plt.title("Investigating the relationship between lambda and matrix dimensions")
plt.show()
```



We clearly observe the relationship that as the dimensions of the matrix increase the lambda becomes more negative. Perhaps this is because we have the condition that norm(x) = 1 so when we compute the inverse of $(\mathbf{A}^{\mathbf{T}}\mathbf{A} + \lambda\mathbf{I})^{-1}$we will need lambda to be large in size so that the inverse is smaller in size in order to compensate for the increasing norm when multiplying by $A^T b$.