Matthias Bernhard
D. Harder, NE 112, University of Waterloo
November 16, 2020

Project 1

Question 1

```
#Question 1
white_noise = [None] * 16
random.seed(16128198)
for i in range( 0, 16 ):
        white_noise[i] = BoxMueller.randn();
print("white_noise", white_noise )
white_noise_squared = 0
for i in range(0, 16):
        white_noise_squared += white_noise[i]**2
print("2-norm_of_white_noise", math.sqrt(white_noise_squared))
mean_white_noise = 0
for i in range(0, 16):
        mean_white_noise += white_noise[i]/16
print("average of white noise", mean_white_noise)
```

Using the Box Mueller and random seed source code I wrote these lines of code to determine the random vector, the 2-norm, and average of the white noise. Using different random seeds here were my outputs,

Seeds,

1. 420420
2. 696969
3. 666666
4. 8008135
5. 123456

2-Norm of White Noise

1. 5.003519381665708
2. 4.750598659601192
3. 3.756284227819451
4. 4.74932372951922
5. 2.4660228620809193

Average of White Noise

1. 0.5432224973009461
2. -0.2049400528266051
3. 0.5648001166830745
4. -0.502345440967396
5. 0.07250025244722494

The 2-norm of the white noise will always be significantly larger than the average since you are squaring and adding together positive and negative values whereas the average is compiled of positive and negative values and will typically result in a decimal number in this case.

Question 2

In terms of COVID-19 testing, reporting the number of positive tests on a day to day basis rather than over an averaged ten-day period would produce misinformed and inaccurate representations of exactly how volatile and dangerous the virus truly is. The number of reported cases on an individual day could be affected by a variety of different unforeseeable variables such as traffic, day of the week, or recent large mass gatherings such as Halloween. Due to this, if news outlets reported the number of cases on a daily basis the public might interpret a day with relatively low cases reported as a good sign and that COVID-19 is starting to slow. Conversely, if on a single day the number of cases reported was high in comparison to previous days the public might panic and believe that the pandemic is worse than it truly is. When reporting cases on a day to day basis this variance in reported cases is much more likely and will have a greater change in the number of cases. Furthermore, when analyzing an average over the course of ten days trendlines can be observed and compared to previous ten-day averages rather than comparing cases on individual days. This would be a better indicator of whether or not the cases in a given area are increasing or decreasing.

Question 3

Note v = [[[None]*16]*16]

```python
#Question 3
for i in range(1, 9):
        for j in range(0,16):
                v[i][j] = math.cos(2*math.pi*(i)*(j+0.5)/16)
for i in range(9, 17):
        for j in range(0, 16):
                 v[i][j] = math.sin(2*math.pi*(i-8)*(j+0.5)/16)
print("initial 17 sine and cosine vectors", v)
v.pop(7)
print("final 16 sine and cosine vectors without zero vector", v)
```

In regards to the cosine function, when f is equal to 8 the resulting vector is 0 (or zero vector), therefore, that vector contains practically no useful information. This is because when f is equal to 8 we now have 16pi which will cancel out with the (i+0.6)/16 section of our function. The resulting formula is cos(0.5pi + i*pi) where i is equal to any number within the range 1-16.

Question 4

```python
#Question 4
x = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
for i in range(0, 16):
        for j in range(0, 16):
                x[i] += ((v[i][j])**2)
y = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
for i in range(0, 16):
        y[i] = math.sqrt(x[i])
print("2-norm of final 16-dimensional sine and cosine vectors", y)
for i in range(0, 16):
        for j in range(0, 16):
                v[i][j] = v[i][j]/y[i]
print("normalized 16-dimenesional sine and cosine vectors", v)
```

Using the seed 420420 (Which I will also use for all subsequent questions and answers), the 2-norm for each of the 16 vectors are as follows,

1. 4.0
2. 2.8284271247461903
3. 2.8284271247461903
4. 2.82842712474619
5. 2.828427124746188
6. 2.82842712474619
7. 2.82842712474619

8. 7.728839736468457e-15
9. 2.8284271247461903
10. 2.82842712474619
11. 2.8284271247461903
12. 2.828427124746192
13. 2.8284271247461903
14. 2.8284271247461903
15. 2.8284271247461903
16. 4.0

Question 5

```python
#Question 5
Q5A = [[0.0]*16, [0.0]*16, [0.0]*16, [0.0]*16, [0.0]*16, [0.0]*16, [0.0]*16, [0.0]*16, [0.0]*16, [0.0]*16, [0.0]*16
for i in range(0, 16):
        for j in range(i, 16):
                for k in range(0, 16):
                        Q5A[i][j] += v[i][k]*v[j][k]
print("Inner product of every possible pair combination of the 16-dimensional sine and cosine vectors", Q5A)
```

Output in console

```
Inner product of every possible pair combination of the 16-dimensional vectors [[1.0, -6.938893903907228e-17, -6.938893903907228e-17, -2.914335439641036e-16, -4.163336342344337e-17, 4.85722573273506e-17, -3
.122502256758253e-16, -0.6637361936714414, 1.1102230246251565e-16, 2.7755575615628914e-17, 1.3877787807814457e-17, -5.551115123125783e-17, -2.3592239273284576e-16, 2.0816681711721685e-16, 2.498001805406602e-16, 0.0], [0.0,
0.9999999999999998, -1.249000902703301e-16, -2.636779683484747e-16, -1.3877787807814457e-16, -1.3877787807814457e-16, -3.7470027081099033e-16, 0.14671196876668707, -2.7755575615628914e-17, -6.245004513516506e-17, -1
.5265566588595902e-16, -5.551115123125783e-17, 8.326672684688674e-17, -4.0245584642661925e-16, 2.914335439641036e-16, 8.326672684688674e-17], [0.0, 0.0, 0.9999999999999999, -2.914335439641036e-16, -1.6653345369377348e-16,
-6.938893903907228e-17, -5.065392549852277e-16, -0.12358286362318155, -1.1449174941446927e-16, -3.469446951953614e-17, -4.3021142204224816e-16, -2.0816681711721685e-16, 3.191891195797325e-16, 5.551115123125783e-17,
-3.469446951953614e-16, 6.938893903907228e-17], [0.0, 0.0, 0.0, 0.9999999999999999, 1.6653345369377348e-16, -1.5959455978986625e-16, -4.3021142204224816e-16, -0.08188684434987076, 6.591949208711867e-17, 4.85722573273506e-17,
 1.8041124150158794e-16, -9.71445146547012e-17, 1.249000902703301e-16, 3.0531133177191805e-16, -7.355227538141662e-16, 8.326672684688674e-17], [0.0, 0.0, 0.0, 0.0, 1.0, 2.0816681711721685e-17, -5.75928194024З3e-16, -0
.005921912132931778, 3.469446951953614e-17, -1.5959455978986625e-16, -4.0245584642661925e-16, 0.0, -6.938893903907228e-17, -3.0531133177191805e-16, 3.469446951953614e-16, 6.938893903907228e-17], [0.0, 0.0, 0.0, 0.0, 0.0,
1.0, -3.9898639947466563e-16, -0.0993679197969420, 9.194034422677078e-17, -2.949029909160572e-16, -6.245004513516506e-17, 2.42861286636753e-16, -1.5265566588595902e-16, 1.6653345369377348e-16, 4.718447854656915e-16,
-3.5388358909926865e-16], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0000000000000002, -0.1482085639469374, -1.2836953722228372e-16, -9.020562075079397e-17, 3.157196726277789e-16, 2.42861286636753e-16, 4.510281037539698e-16,
-9.71445146547012e-17, 9.020562075079397e-17, 4.0245584642661925e-16], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.9999999999999999, 0.2879572085232336, 0.18124329467731196, 0.005566771557325378, 0.057458716313476134, 0
.09400964528256761, 0.012880564368006414, 0.02557260172082898, 0.5645037438847402], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, -4.683753385137379e-17, -9.71445146547012e-17, -2.7755575615628914e-17, 2
.0469737016526324e-16, -1.3530843112619095e-16, -3.8163916471489756e-17, -2.7755575615628914e-17], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 3.469446951953614e-17, 4.85722573273506e-17, 1.1796119636642288e-16,
3.7470027081099033e-16, -5.065392549852277e-16, 4.85722573273506e-17], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0000000000000002, -1.3877787807814457e-17, -1.942890293094024e-16, -5.551115123125783e-17,
1.3877787807814457e-16, 2.0816681711721685e-16], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.9999999999999999, -8.326672684688674e-17, 0.0, 7.632783294297951e-16, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 1.0000000000000002, 9.71445146547012e-17, 2.914335439641036e-16, 2.7755575615628914e-17], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 4.85722573273506e-16, 5
.412337245047638e-16], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0000000000000002, 4.85722573273506e-16], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0]]
```

Whenever one of the 16-dimensional vectors is multiplied onto itself it returns the one vector, all other cases return the 0-vector since the vectors are all orthogonal to each other. All returned vectors after running this code produce a variety of extremely small entries to e-16 or e-17. This is a result of python trying its best to calculate these values but for all intents and purposes, we can assume them to be close enough to 0 that we consider them the 0-vector.

Question 6

```python
#Question 6
InnerProduct = [0.0]*16
for i in range(0, 16):
        for j in range(0, 16):
                        InnerProduct[i] += v[i][j]*white_noise[j]
print("Inner Product", InnerProduct)
```

Before I programmed this question I had somewhat expected the one-vector coefficient and white noise to be significantly larger than the other but after calculating with python all of the individual vector coefficients, I have come to the conclusion that none of the coefficients are consistently significantly larger than the others. This, of course, doesn't line up with my original hypothesis. Coefficients would be closer in mean when applying vectors of the same or similar magnitude although not in a significantly measurable way.

Question 7

```
#Question 7, f=1, A=20, phase shift=+0.9197057831 for Q7 vector, Coefficients
Q7B = [0.0]*16
Coefficient = [0.0]*16
for i in range(0, 16):
    Q7B[i] = 20*math.sin(2*math.pi*1*((i+0.5)/16)+0.9197057831)
for i in range(0, 16):
        for j in range(0, 16):
                Coefficient[i] += Q7B[j]*v[i][j]
print("Coefficient Vector", Coefficient)
print("2-norm of Coefficient Vector", numpy.linalg.norm(Coefficient))
```

All coefficients are very close to zero which we can assume to practically be equal to zero. The one exception being when the inner products hold the same frequency of the vector just created. Furthermore, the 2-norm is dependant on the magnitude and where the vector is multiplied by amplitude. Using the Pythagorean Theorem,

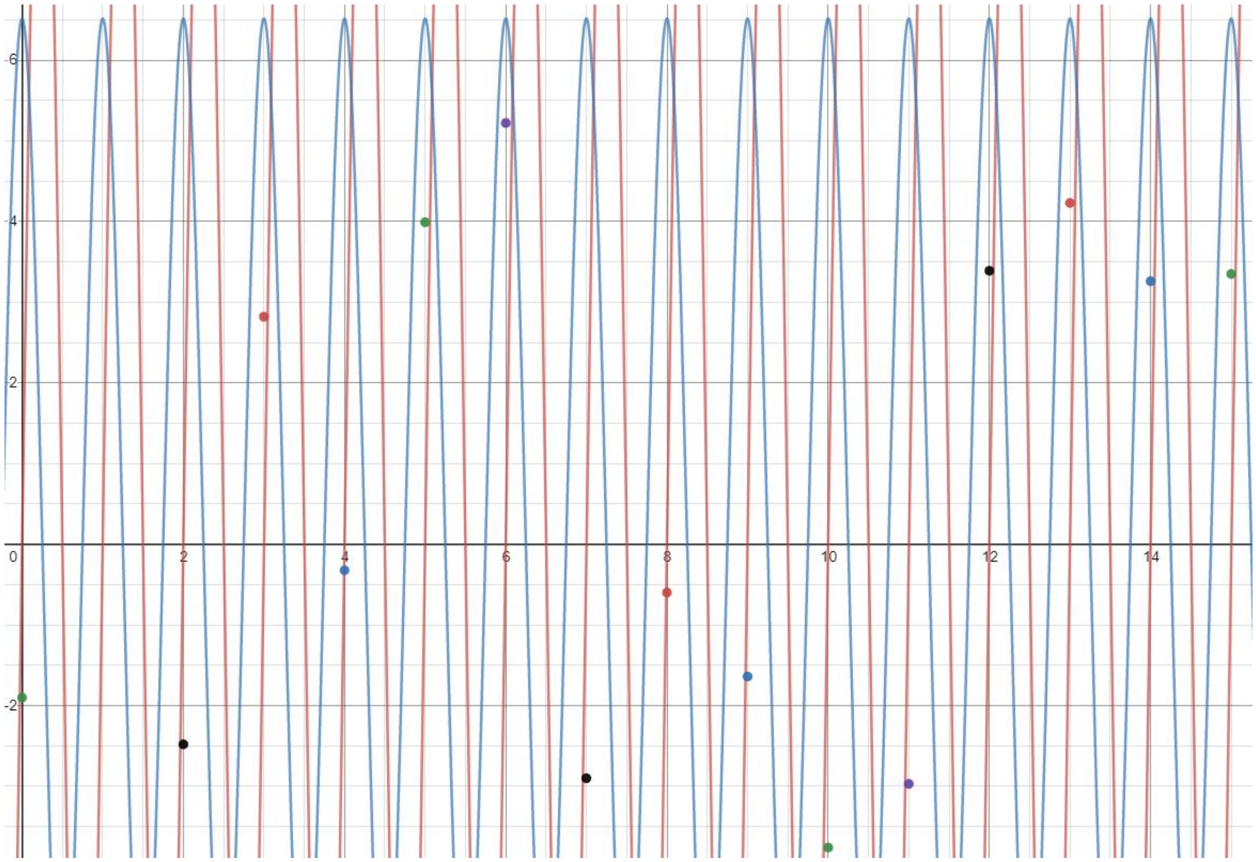
2-norm^2 = coefficient(sin)^2 + coefficient(cos)^2.

Note, the coefficients in this case only apply to those when not 0. Lastly, multiplying the 2-norm by sin(phi) to return a cos vector of the same frequency. This also works in reverse.

Question 8

The returned coefficient vector given from when f = float and not when f = int gives us values that don't represent exact values pertaining to our exact frequencies. This resulting code is less useful in determining and containing information regarding our original 16-dimensional vector which leads to it being more difficult to know which coefficient to use in our calculations. Without knowing the exact coefficient to use in our calculations, determining our original 16-dimensional frequency, frequency, and amplitude will become measurable more challenging. We could always use estimations and approximations but even that becomes increasingly harder as our floats approach midpoints of two integers such as 3.5 or 4.5.

Question 9



Desmos graph of the heartbeat values taken every second and the two sinusoidal waves. The red function is that of the fetus's heartbeat and the blue function is that of the mother's heartbeat.

```
#Question 9
Q9V = [-1.90, -6.05, -2.48, 2.82, -.324, 3.99, 5.22, -2.90, -.599, -1.64, -3.76, -2.97, 3.39, 4.23, 3.26, 3.35]
NormalizedQ9 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
TwoNormQ9 = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
for i in range(0, 16):
    NormalizedQ9[i] += ((Q9V[i])**2)
for i in range(0, 16):
    TwoNormQ9[i] = math.sqrt(NormalizedQ9[i])
for i in range(0, 16):
    NormalizedQ9[i] = Q9V[i] / TwoNormQ9[i]
for i in range(0, 16):
    Q9V[i] = numpy.dot(Q9V, NormalizedQ9)
for i in range(1,8):
    print("Frequency", i, ":", Q9V[i], Q9V[i + 8])
AmplitudeFetus = math.sqrt(Q9V[2]**2 + Q9V[10]**2)
AmplitudeMother = math.sqrt(Q9V[5]**2 + Q9V[13]**2)
print("Amplitude of Fetus", AmplitudeFetus)
print("Amplitude of Mother", AmplitudeMother)
BPMFetus = 2 / 16 * 60
BPMMother = 5 / 16 * 60
print("BPM of Fetus", BPMFetus)
print("BPM of Mother", BPMMother)
```

```
frequency 1 : 0.7976341611353694        and     0.023534121097750993
frequency 2 : -1.7827771199462985       and     -10.839124392891181
frequency 3 : -0.4996645117487344       and     -2.8713031619971496
frequency 4 : -0.4757499999999971       and     -0.3607499999999968
frequency 5 : 6.079684236415877         and     -2.3416022146588045
frequency 6 : -0.2830571772557464       and     2.130468967253787
frequency 7 : -1.2561982153167883       and     -1.8370717501922178

Amplitude
fetus :      10.984758161378547
mother :     6.515033487728112

BPM (frequency)
fetus :      7.5
mother :     18.75
```

A few assumptions I made to complete this question are as follows,

1. The fetus will have a larger amplitude
2. Resulting in the fetus having a lower BPM than the mother
    a. This isn’t the case for real-world pregnancies

Question 10

```
#Question 10
Q10V = [None]*16
for i in range(0, 16):
    Q10V[i] = math.cos(2 * math.pi * 13 * ((i+0.5)/16) + 2)
z1 = [1] * 16
z2 = [None] * 16
z3 = [None] * 16
z4 = [None] * 16
z5 = [None] * 16
z6 = [None] * 16
z7 = [None] * 16
z8 = [None] * 16
z9 = [None] * 16
z10 = [None] * 16
z11 = [None] * 16
z12 = [None] * 16
z13 = [None] * 16
z14 = [None] * 16
z15 = [None] * 16
z16 = [None] * 16
z17 = [None] * 16
for i in range(0, 16):
    z2[i] = math.cos(2*math.pi*((i+0.5)/16))
for i in range(0, 16):
    z3[i] = math.cos(2*math.pi*2*((i+0.5)/16))
for i in range(0, 16):
    z4[i] = math.cos(2*math.pi*3*((i+0.5)/16))
for i in range(0, 16):
    z5[i] = math.cos(2*math.pi*4*((i+0.5)/16))
for i in range(0, 16):
    z6[i] = math.cos(2*math.pi*5*((i+0.5)/16))
for i in range(0, 16):
    z7[i] = math.cos(2*math.pi*6*((i+0.5)/16))
for i in range(0, 16):
    z8[i] = math.cos(2*math.pi*7*((i+0.5)/16))
for i in range(0, 16):
    z9[i] = math.cos(2*math.pi*8*((i+0.5)/16))
for i in range(0, 16):
    z10[i] = math.sin(2*math.pi*((i+0.5)/16))
for i in range(0, 16):
    z11[i] = math.sin(2*math.pi*2*((i+0.5)/16))
```

```
for i in range(0, 16):
    z12[i] = math.sin(2*math.pi*3*((i+0.5)/16))
for i in range(0, 16):
    z13[i] = math.sin(2*math.pi*4*((i+0.5)/16))
for i in range(0, 16):
    z14[i] = math.sin(2*math.pi*5*((i+0.5)/16))
for i in range(0, 16):
    z15[i] = math.sin(2*math.pi*6*((i+0.5)/16))
for i in range(0, 16):
    z16[i] = math.sin(2*math.pi*7*((i+0.5)/16))
for i in range(0, 16):
    z17[i] = math.sin(2*math.pi*8*((i+0.5)/16))
a1 = (numpy.linalg.norm(z1))
a2 = (numpy.linalg.norm(z2))
a3 = (numpy.linalg.norm(z3))
a4 = (numpy.linalg.norm(z4))
a5 = (numpy.linalg.norm(z5))
a6 = (numpy.linalg.norm(z6))
a7 = (numpy.linalg.norm(z7))
a8 = (numpy.linalg.norm(z8))
a10 = (numpy.linalg.norm(z10))
a11 = (numpy.linalg.norm(z11))
a12 = (numpy.linalg.norm(z12))
a13 = (numpy.linalg.norm(z13))
a14 = (numpy.linalg.norm(z14))
a15 = (numpy.linalg.norm(z15))
a16 = (numpy.linalg.norm(z16))
a17 = (numpy.linalg.norm(z17))
normalized1 = (z1/a1)
normalized2 = (z2/a2)
normalized3 = (z3/a3)
normalized4 = (z4/a4)
normalized5 = (z5/a5)
normalized6 = (z6/a6)
normalized7 = (z7/a7)
normalized8 = (z8/a8)
normalized10 = (z10/a10)
normalized11 = (z11/a11)
normalized12 = (z12/a12)
normalized13 = (z13/a13)
normalized14 = (z14/a14)
normalized15 = (z15/a15)
```

```
normalized15 = (z15/a15)
normalized16 = (z16/a16)
normalized17 = (z17/a17)
print(numpy.dot(Q10V, normalized1))
print(numpy.dot(Q10V, normalized2))
print(numpy.dot(Q10V, normalized3))
print(numpy.dot(Q10V, normalized4))
print(numpy.dot(Q10V, normalized5))
print(numpy.dot(Q10V, normalized6))
print(numpy.dot(Q10V, normalized7))
print(numpy.dot(Q10V, normalized8))
print(numpy.dot(Q10V, normalized10))
print(numpy.dot(Q10V, normalized11))
print(numpy.dot(Q10V, normalized12))
print(numpy.dot(Q10V, normalized13))
print(numpy.dot(Q10V, normalized14))
print(numpy.dot(Q10V, normalized15))
print(numpy.dot(Q10V, normalized16))
print(numpy.dot(Q10V, normalized17))
```

The 16 coefficients that output from the code above is as follows,

1. 3.0531133177191805e-16
2. -1.1102230246251565e-16
3. -1.7208456881689926e-15
4. 1.1770410003672587
5. 1.2490009027033011e-15
6. 0.0
7. 2.6367796834847468e-15
8. 1.887379141862766e-15
9. -1.3877787807814457e-15
10. -1.1102230246251565e-16
11. -2.5718815064956697
12. 1.942890293094024e-15
13. -8.326672684688674e-16
14. 1.3461454173580023e-15
15. -3.552713678800501e-15
16. -3.1363800445660672e-15

The frequency that the data above suggests is that f = 6. In order to more accurately find the activity within a cycle, you can sample the sensor more often than the original function for example 3-4x more samples per period. This would change the 16 values in the original function into 48, or 64 respectively. This allows for the activity to be more accurately recorded by sampling at a higher rate, resulting in a large data and sample base.