

Project 1

Question 1

```
87 #Question 1
88 white_noise = [None] * 16
89 random.seed(16128198)
90 for i in range(0, 16):
91     white_noise[i] = BoxMueller.randn()
92     print("white_noise", white_noise)
93     white_noise_squared = 0
94     for i in range(0, 16):
95         white_noise_squared += white_noise[i]**2
96     print("2-norm_of_white_noise", math.sqrt(white_noise_squared))
97     mean_white_noise = 0
98     for i in range(0, 16):
99         mean_white_noise += white_noise[i]/16
100    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 $v = [[[None]*16]*16]$

```
113 #Question 3
114 for i in range(1, 9):
115     for j in range(0, 16):
116         v[i][j] = math.cos(2*math.pi*(i)*(j+0.5)/16)
117 for i in range(9, 17):
118     for j in range(0, 16):
119         v[i][j] = math.sin(2*math.pi*(i-8)*(j+0.5)/16)
120 print("initial 17 sine and cosine vectors", v)
121 v.pop(7)
122 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 16π which will cancel out with the $(i+0.6)/16$ section of our function. The resulting formula is $\cos(0.5\pi + i\pi)$ where i is equal to any number within the range 1-16.

Question 4

```
125 #Question 4
126 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]
127 for i in range(0, 16):
128     for j in range(0, 16):
129         x[i] += ((v[i][j])**2)
130 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]
131 for i in range(0, 16):
132     y[i] = math.sqrt(x[i])
133 print("2-norm of final 16-dimensional sine and cosine vectors", y)
134 for i in range(0, 16):
135     for j in range(0, 16):
136         v[i][j] = v[i][j]/y[i]
137 print("normalized 16-dimensional 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

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

```
157 #Question 7, f=1, A=20, phase_shift=+0.9197057831 for Q7 vector, Coefficients
158 Q7B = [0.0]*16
159 Coefficient = [0.0]*16
160 for i in range(0, 16):
161     Q7B[i] = 20*math.sin(2*math.pi*1*((i+0.5)/16)+0.9197057831)
162 for i in range(0, 16):
163     for j in range(0, 16):
164         Coefficient[i] += Q7B[j]*v[i][j]
165 print("Coefficient Vector", Coefficient)
166 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,

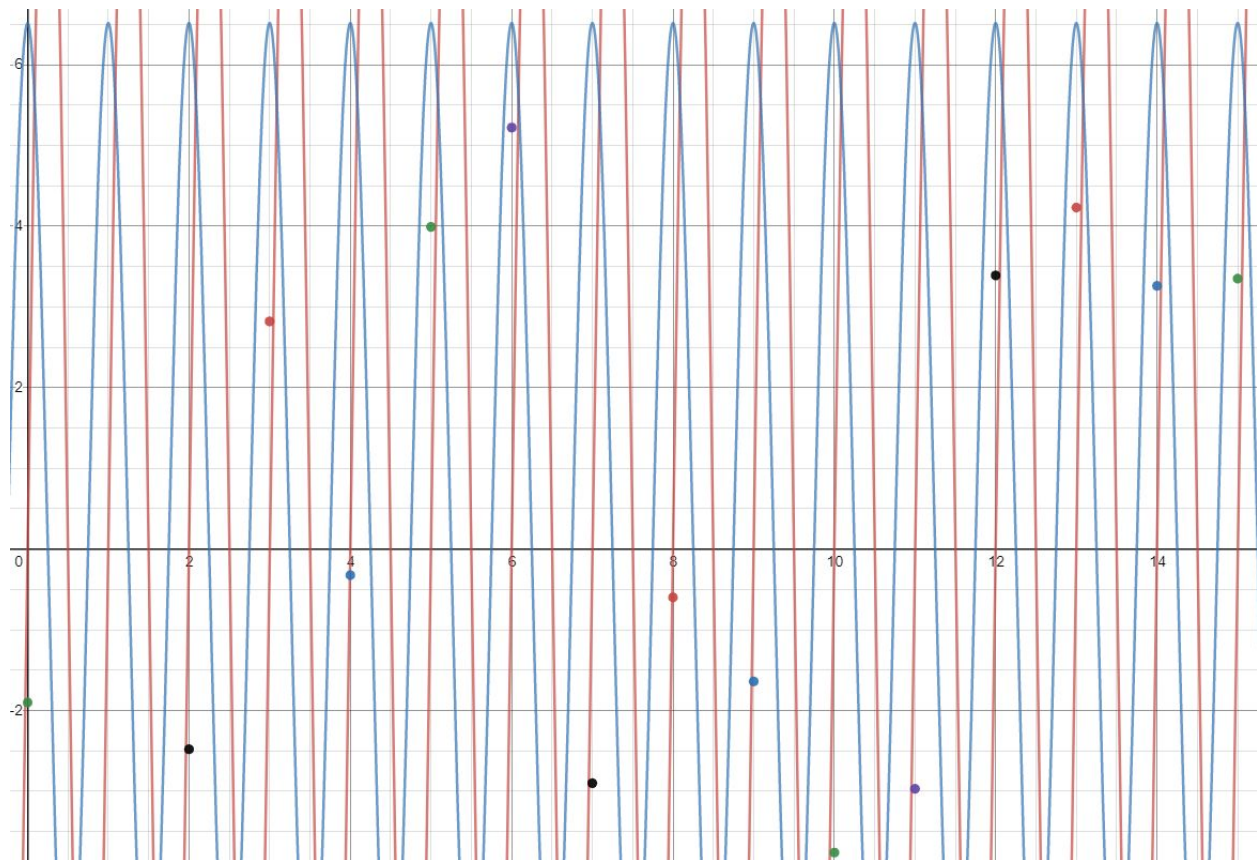
$$2\text{-norm}^2 = \text{coefficient}(\sin)^2 + \text{coefficient}(\cos)^2.$$

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 $f = \text{float}$ and not when $f = \text{int}$ 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.

```

176  @#Question 9
177  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]
178  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]
179  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]
180  for i in range(0, 16):
181      NormalizedQ9[i] += ((Q9V[i])**2)
182  for i in range(0, 16):
183      TwoNormQ9[i] = math.sqrt(NormalizedQ9[i])
184  for i in range(0, 16):
185      NormalizedQ9[i] = Q9V[i] / TwoNormQ9[i]
186  for i in range(0, 16):
187      Q9V[i] = numpy.dot(Q9V, NormalizedQ9)
188  for i in range(1, 8):
189      print("Frequency", i, ":", Q9V[i], Q9V[i + 8])
190  AmplitudeFetus = math.sqrt(Q9V[2]**2 + Q9V[10]**2)
191  AmplitudeMother = math.sqrt(Q9V[5]**2 + Q9V[13]**2)
192  print("Amplitude of Fetus", AmplitudeFetus)
193  print("Amplitude of Mother", AmplitudeMother)
194  BPMFetus = 2 / 16 * 60
195  BPMMother = 5 / 16 * 60
196  print("BPM of Fetus", BPMFetus)
197  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

```
178 #Question 10
179 Q10V = [None]*16
180 for i in range(0, 16):
181     Q10V[i] = math.cos(2 * math.pi * 13 * ((i+0.5)/16) + 2)
182 z1 = [1] * 16
183 z2 = [None] * 16
184 z3 = [None] * 16
185 z4 = [None] * 16
186 z5 = [None] * 16
187 z6 = [None] * 16
188 z7 = [None] * 16
189 z8 = [None] * 16
190 z9 = [None] * 16
191 z10 = [None] * 16
192 z11 = [None] * 16
193 z12 = [None] * 16
194 z13 = [None] * 16
195 z14 = [None] * 16
196 z15 = [None] * 16
197 z16 = [None] * 16
198 z17 = [None] * 16
199 for i in range(0, 16):
200     z2[i] = math.cos(2*math.pi*((i+0.5)/16))
201 for i in range(0, 16):
202     z3[i] = math.cos(2*math.pi*2*((i+0.5)/16))
203 for i in range(0, 16):
204     z4[i] = math.cos(2*math.pi*3*((i+0.5)/16))
205 for i in range(0, 16):
206     z5[i] = math.cos(2*math.pi*4*((i+0.5)/16))
207 for i in range(0, 16):
208     z6[i] = math.cos(2*math.pi*5*((i+0.5)/16))
209 for i in range(0, 16):
210     z7[i] = math.cos(2*math.pi*6*((i+0.5)/16))
211 for i in range(0, 16):
212     z8[i] = math.cos(2*math.pi*7*((i+0.5)/16))
213 for i in range(0, 16):
214     z9[i] = math.cos(2*math.pi*8*((i+0.5)/16))
215 for i in range(0, 16):
216     z10[i] = math.sin(2*math.pi*((i+0.5)/16))
217 for i in range(0, 16):
218     z11[i] = math.sin(2*math.pi*2*((i+0.5)/16))
```


1: Projects

2: Structure

3: Favorites

```
219 for i in range(0, 16):
220     z12[i] = math.sin(2*math.pi*3*((i+0.5)/16))
221 for i in range(0, 16):
222     z13[i] = math.sin(2*math.pi*4*((i+0.5)/16))
223 for i in range(0, 16):
224     z14[i] = math.sin(2*math.pi*5*((i+0.5)/16))
225 for i in range(0, 16):
226     z15[i] = math.sin(2*math.pi*6*((i+0.5)/16))
227 for i in range(0, 16):
228     z16[i] = math.sin(2*math.pi*7*((i+0.5)/16))
229 for i in range(0, 16):
230     z17[i] = math.sin(2*math.pi*8*((i+0.5)/16))
231 a1 = (numpy.linalg.norm(z1))
232 a2 = (numpy.linalg.norm(z2))
233 a3 = (numpy.linalg.norm(z3))
234 a4 = (numpy.linalg.norm(z4))
235 a5 = (numpy.linalg.norm(z5))
236 a6 = (numpy.linalg.norm(z6))
237 a7 = (numpy.linalg.norm(z7))
238 a8 = (numpy.linalg.norm(z8))
239 a10 = (numpy.linalg.norm(z10))
240 a11 = (numpy.linalg.norm(z11))
241 a12 = (numpy.linalg.norm(z12))
242 a13 = (numpy.linalg.norm(z13))
243 a14 = (numpy.linalg.norm(z14))
244 a15 = (numpy.linalg.norm(z15))
245 a16 = (numpy.linalg.norm(z16))
246 a17 = (numpy.linalg.norm(z17))
247 normalized1 = (z1/a1)
248 normalized2 = (z2/a2)
249 normalized3 = (z3/a3)
250 normalized4 = (z4/a4)
251 normalized5 = (z5/a5)
252 normalized6 = (z6/a6)
253 normalized7 = (z7/a7)
254 normalized8 = (z8/a8)
255 normalized10 = (z10/a10)
256 normalized11 = (z11/a11)
257 normalized12 = (z12/a12)
258 normalized13 = (z13/a13)
259 normalized14 = (z14/a14)
260 normalized15 = (z15/a15)
```

```

260 normalized15 = (z15/a15)
261 normalized16 = (z16/a16)
262 normalized17 = (z17/a17)
263 print(numpy.dot(Q10V, normalized1))
264 print(numpy.dot(Q10V, normalized2))
265 print(numpy.dot(Q10V, normalized3))
266 print(numpy.dot(Q10V, normalized4))
267 print(numpy.dot(Q10V, normalized5))
268 print(numpy.dot(Q10V, normalized6))
269 print(numpy.dot(Q10V, normalized7))
270 print(numpy.dot(Q10V, normalized8))
271 print(numpy.dot(Q10V, normalized10))
272 print(numpy.dot(Q10V, normalized11))
273 print(numpy.dot(Q10V, normalized12))
274 print(numpy.dot(Q10V, normalized13))
275 print(numpy.dot(Q10V, normalized14))
276 print(numpy.dot(Q10V, normalized15))
277 print(numpy.dot(Q10V, normalized16))
278 print(numpy.dot(Q10V, normalized17))
279

```

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.