## Question 1:

1. <http://users.ece.gatech.edu/mrichard/Ch1_FRSP2e.pdf>

This link provides useful info about radar.

Radar development started in the middle and late 1930’s independently by countries like the United States, Britain, Japan, Russia, France, Germany and Italy spurred by threat of war and military necessity. Radars now exhibit a much wider range of applications from traffic control to space borne environmental mapping, most uses of the radar can be classified as detection, tracking or imaging.

Radar imaging uses fast Fourier transform techniques (FFT). Target detection in radars uses linear filtering and statistical detection theory. Target identification also uses pattern recognition techniques.

Radar signals have a very high dynamic ranges of tens of decibels thus gain control schemes are common. Moreover, sildelobe control is critical to avoid having weak signals masked by stronger signals.

As for signal processing hardware, compared to most DSP applications, radar signal bandwidths are large with instantaneous bandwidths of individual pulses reaching several hundred megahertz and as high as 1 GHz. This in turn has several implications for digital signal processing. Primarily, very fast analog-to-digital converters (ADC’s) are needed. The high data rate also meant the need to design custom hardware for the digital processor to obtain adequate throughput.

The purpose of signal processing in radar is to improve figures of merit such as signal to interference ratio (SIR), resolution, accuracy and sidelobe behaviour. Pulse compression and other waveform design techniques, such as frequency agility can improve SIR and resolution. SIR can also be improved using pulse integration. Windowing techniques used in applications of signal processing can improve sidelobe behaviour.

1. Both FPGA’s and ASIC’s are available hardware platforms for signal processing techniques with their advantages.

http://www.xilinx.com/fpga/asic.htm

The advantages of FPGA’s:

* FPGA’s are cheaper in short term because they require no upfront non-recruiting expenses (NRE) such as engineers to develop designs for signal processing techniques which are costs associated with ASIC designs.
* They are reprogrammable, allowing programs to be uploaded remotely.
* Make designs easier to design since software is used to handle timing, placement and routing.
* They require less time to develop since no manufacturing of specific wafers is needed which gives a faster time to market.

The advantages of ASIC’s:

* They are faster since there isn’t the extra hardware at FPGA’s that provides programmability.
* They are more power efficient consuming less power with less hardware.
* More robust than FPGA’s since they don’t have to be flashed to program them.
* ASIC’s are cheaper when produced in very large quantities to compensate for cost of extra development time.

1. FPGA technology has its advantages over ASIC because of being reprogrammable. However, FPGA presents some crucial disadvantages if we consider space applications. For high telecom processing and high volume measurement applications (such as measuring data of different environments) arrays of ASICs are used instead of FPGA, because they can give the required processing capacity at a much better power consumption. FPGAs also have a much larger configuration and compilation time compared to ASICs, which make them preferable in dealing with applications requiring faster boot-up, configuration and compilation. TOCITE: <https://amstel.estec.esa.int/tecedm/website/stag_ygt/Boada.pdf>

## Question 2:

a) In order to prove that dmin = min wt(**c**) over all non-zero codewords for binary linear block codes, two inequalities that are shown below must be first proves:

1. dmin ≤ min wt(**c**)
2. dmin ≥ min wt(**c**)

where wt(**c**) equals the number of non-zero symbols for a codeword c, in codebook, where c is non-zero.

The proof for the two inequalities is shown below:

1. Knowing that dmin is the minimum distance, dH between two codewords, c­i and cj as shown in this equation:

dmin = dH(ci, cj)

Also knowing that the minimum weight is achieved by cmin, which is the minimum non-zero codeword:

min wt(**c**) = wt(**cmin**)

The weight of cmin is equal to the distance dH between the all-zero codeword and cmin:

wt(**cmin**) = dH (0 0 0 0 0 0, cmin)

Thus, this proves that dmin is less than or equal to minimum weight codeword:

dmin ≤ min wt(**c**)

1. In order to prove the second inequality, we will revisit some properties. Knowing that codeword, c, is generated using the generator matrix, G, and message, m:

c = m×G

Using the linearity property of matrices, if c1 and c2 are two codewords and c3 is given by the equation:

c1 + c2 = c3

This can be rewritten as given below:

m1 ×G + m2 ×G = (m1 + m2) ×G = m3 ×G

This gives a new message vector, m3­, which means that c3 is also a codeword.

Taking the weight of the addition of two binary linear codewords, wt(c1 + c2), gives the distance between these two codewords, since the addition of binary is a modulus :

wt(c1 + c2) = dH(c1, c2­)

The addition of these codewords gives a new codeword c3:

wt(c3) = wt(c1 + c2) = dH(c1, c2­)

Knowing that dmin is the minimum distance between two codewords:

dmin = dH(ci, cj)

Then we can prove that the weight of c3 is also less than or equal to dmin:

dmin ≥ min wt(**c**)

These two proofs conclude that dmin= min wt(**c**) for any binary linear block codes that are non-zero.

b) To prove that dmin = min wt(**c**) over all non-zero codewords for binary non-linear block codes, two inequalities shown below need to be proved:

1. dmin ≤ min wt(**c**)
2. dmin ≥ min wt(**c**)

Where wt(**c**) equals the number of non-zero symbols for a codeword c, in codebook, where c is non-zero.

The first proof is similar to the proof used in linear block codes in question 2 (a), however the proof for the second inequality is different. It is different because in binary linear codes, addition and subtraction are similar, however non-binary codes subtraction and addition operations are no longer similar. Thus, the distance between two codewords is given by subtraction instead of addition.

The proof of the two inequalities is shown below:

1. Knowing that dmin is the minimum distance, dH between two codewords, c­i and cj as shown in this equation:

dmin = dH(ci, cj)

Also knowing minimum weight is achieved by cmin, which is the minimum non-zero codeword:

min wt(**c**) = wt(**cmin**)

The weight of cmin is equal to the distance dH between the all-zero codeword and cmin:

wt(**cmin**) = dH (0 0 0 0 0 0, cmin)

Thus, this proves that dmin is less than or equal to minimum weight codeword:

dmin ≤ min wt(**c**)

1. In order to prove the second inequality, we will revisit some properties. Knowing that codeword, c, is generated using the generator matrix, G, and message, m:

c = m×G

Using the linearity property of matrices, if c1 and c2 are two codewords and c3 is given by the equation:

c1 - c2 = c3

This can be rewritten as given below:

m1 ×G - m2 ×G = (m1 - m2) ×G = m3 ×G

This gives a new message vector, m3­, which means that c3 is also a codeword.

Taking the weight of the subtraction of two binary linear codewords, wt(c1 - c2), gives the distance between these two codewords, since the subtraction and addition of non-binary codes are not the same:

wt(c1 - c2) = dH(c1, c2­)

As shown earlier, the subtraction of these codewords gives a new codeword c3:

wt(c3) = wt(c1 - c2) = dH(c1, c2­)

Knowing that dmin is the minimum distance between two codewords:

dmin = dH(ci, cj)

Then we can prove that the weight of c3 is also less than or equal to dmin:

dmin ≥ min wt(**c**)

These two proofs conclude that dmin= min wt(**c**) for any non-binary linear block codes that are non-zero.

c) The Hamming bound theorem states that any (n, k, dmin) binary code must satisfy the following formula:

2k ( + ( + + … + () ≤ 2n

Where 2k corresponds to the total number of spheres, which are multiplied by all possible points in each sphere (for example, (n 1) corresponds to all points which differ from 0 codeword by 1 symbol). 2n corresponds to all possible points.

For (24, 12, 9) binary code, need to check that the Hamming bound formula is valid:

212 ( + + + + ) ≤ 224

+ + + + ≤ 212

1 + 24 + 276 + 2024 + 10626 ≤ 4096

12951 /≤ 4096

The Hamming Bound on the (24, 12, 9) does not exist meaning that this is an invalid binary code.

d) Knowing that a codebook for any binary code is formed via b \* G, where b = all possible binary vectors for the given code, and G = generator matrix. In order for each codeword in the codebook to be decodable, each codeword has to differ by at most dmin – 1. Hence, the binary code can always correct up to dmin – 1 erasures.

Consider two codewords, c1 and c2 in the figure below. Each circle around that codeword indicates all possible symbols which differ from the codeword by d, where d is the difference between the codeword and the symbol. C1 and c2 differ by dmin.

d = 2

d = 1

d = 2

d = 1

X – received vector

dmin

For X to be decoded as c1, it has to fall within the sphere of c1 (as shown above). Hence the maximum radius of a sphere is (dmin - 1 )/2, so the code can always correct (dmin - 1 )/2 binary errors in the codeword.

e) The table below lists all codewords for (6,3,3) code

|  |  |
| --- | --- |
| Codewords | wt(c) |
| 000000 | 0 |
| 100110 | 3 |
| 010101 | 3 |
| 110011 | 4 |
| 001011 | 3 |
| 101101 | 4 |
| 011110 | 4 |
| 111000 | 3 |

We can see from above that the minimum distance between any two codewords in the table above is 3. So dmin = 3.

f) There are 16 codeword pairs which differ by dmin distance (3) for this code (a matlab script was used to determine this value). Wt(c) was used for each codeword for this code (from the table in 2e) to plot its spectrum, as shown below.



g) The codewords for (8,4,4) is given in the table below

|  |  |
| --- | --- |
| Codewords | wt(c) |
| 0000 0000 | 0 |
| 1000 1110 | 4 |
| 0100 1101 | 4 |
| 1100 0011 | 4 |
| 0010 1011 | 4 |
| 1010 0101 | 4 |
| 0110 0110 | 4 |
| 1110 1000 | 4 |
| 0001 0111 | 4 |
| 1001 1001 | 4 |
| 0101 1010 | 4 |
| 1101 0100 | 4 |
| 0011 1100 | 4 |
| 1011 0010 | 4 |
| 0111 0001 | 4 |
| 1111 1111 | 8 |

There are 112 codeword pairs which differ by dmin distance (4) for this code (a matlab script was used to determine this value). Wt(c) was used for each codeword for this code (from the table in 2g) to plot its spectrum, as shown below.



h)

## Question 3:

d) The performance of the decoders were measured with a range of probability of erasures, starting from Pe = 0.1 and ending at Pe = 0.8, with Pe incrementing with 0.05. Random messages were generated 100 times for each probability. The bit-error-rate (BER) was recorded against each probability, taking the average BER for all 100 messages. The results were plotted in the graphs shown below.





e) **Time complexity for Exhaustive decoder:**

The exhaustive decoder is essentially divided into two steps. The first step is to form the codebook for the given code. This is achieved by m\*G, where m consists of all possible binary messages for the give code, the size of m being (2^k x k). G is the generator matrix for the code.

For (6,3,3) code generating codebook takes about 0.0002 seconds. So we can assume that the operational time increments linearly and the for (10000, 5000) code it takes approximately 0.0002\* 5000 = 1 seconds.

The next step is to loop over all the vectors from the received matrix, then loop over all the codewords in the codebook and compute difference between all the codewords in the codebook, and the vector under consideration from the received matrix. The codeword with the minimum distance is also found in the outer loop. The dimensions of codebook are (2^k x n) and the received vector is (k x n). Therefore, the complexity will be O(k)\*O(2^k), which is almost equivalent to O(2^k). For (6,3,3) code, considering the worst case scenario (where the entire received matrix contains erasures) the execution time takes about 0.001 seconds. So the time for (10000, 5000) will almost take 0.001\*25000-3, which is almost 101500 seconds.

So the total complexity for (10000, 5000) is almost O(2^5000) or about 101500 seconds.

**Time complexity for Gaussian decoder:**

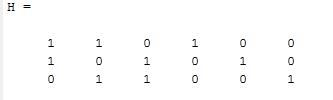
The Gaussian decoder begins by looping over the received matrix and replacing all 0.5 values with e-symbols, denoting erasure unknowns. Next, k equations are formed, which the solved by Matlab’s “linsolve” utility. The “linsolve” utility has about O(n) time complexity. For (6,3,3) code the “linsolve” takes about 0.07 seconds to solve for each row. Each row in this code has 6 columns. So for (10000, 5000) the time taken would be about 0.07\*10000 = 700 seconds.

The outer loop loops over all the rows of the received matrix, which is k. This is an O(k) operation. For (6,3,3) this takes about 0.4 seconds.

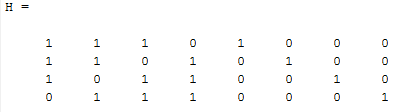
The overall time complexity for this algorithm is O(n) + O(k) which is approximately 700 + (0.4\*500) = 900 seconds for (10000, 5000) code.

f) Parity check matrices, H, are formed by making the parity symbols for an identity matrix, such that, H = (P | I).

For (6,3,3) code the H matrix is given below:



For (8,4,4) code the H matrix is given below:

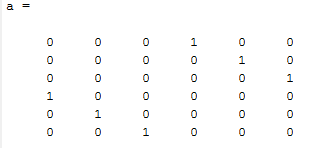


g) It can be observed that H = (P | I) while G = (I | P). So in order to get H from G for any system, we need to swap the P and I sections from G matrix. This is achieved by multiplying G with another matrix, say A, to get H. So we have:

H = G \* A

For any system (n, k, dmin), matrix A is an n\*n matrix. It consists of 2 identity matrices of k\*k dimensions, and the other entries of matrix A are padded with 0’s to make it n\*n. The first identity matrix starts at k+1 column and goes up to n columns for the first k rows. The next identity matrix starts at 0th column and goes up to n-k columns for the remaining n-k rows.

For (6,3,3) code the A matrix looks like:



3h) The graph below shows the performance of syndrome decoder, where the test was conducted for the following probabilities of error (bit flip): 0.2, 0.15, 0.1, 0.05 and 0.01. Random messages were generated for each probability along with random generation of errors, 100 times. The average result was taken for each probability and plotted.



## Question 4:

1. The finite state machine representation for ½ convolutional code used in the 2G, 3G, and 4G cellphones is shown below: The boxes are registers/states, input and outputs c1 and c2 are indicated on **figure 4.1**.

c2

Input

c1

If the current state of registers is 0001 the input 1, then the finite state machine representation is shown in **figure 4.2** and the trellis of this stage is shown in **figure 4.3**.

1

0

0

0

c2

Input

c1

1

0 0 0 0

1 0 0 0

0 / 01

0 0 0 1

1 / 10

1. For the (6,3,3) binary linear code, the parity check matrix is:

[1 0 0 : 1 1 0

0 1 0 : 1 0 1

0 0 1 : 0 1 1]

Table 4.1 of the message bits and parity bits can be used to aid in the formation of trellis.

|  |  |
| --- | --- |
| **Message Bits (m1, m2, m3)** | **Parity Bits (p1, p2, p3)** |
| 0 0 0 | 0 0 0 |
| 0 0 1 | 0 1 1 |
| 0 1 0 | 1 0 1 |
| 0 1 1 | 1 1 0 |
| 1 0 0 | 1 1 0 |
| 1 0 1 | 1 0 1 |
| 1 1 0 | 0 1 1 |
| 1 1 1 | 0 0 0 |

One stage of the trellis diagram can be sketched as shown in **figure 4.4**.

**p1 p2 p3**

**m3 / m3**

**p1 p2 p3**

**m2 / m2**

**p1 p2 p3**

**m1 / m1**

- / 1 1 0

- / 1 0 1

- / 0 1 1

- / 0 0 0

1 / 1

0 / 0

1/ 1

0 / 0

1 / 1

0 / 0

0 0 0

0 1 1

1 0 1

1 1 0 0 0

0 0 0

0 / 0

1 / 1

0 / 0

0 0 0

1 1 0

1 1 0

1 0 1

0 0 0

0 0 0

0 / 0

1 / 1

1 / 1

0 / 0

0 1 1

0 / 0

1. The trellis of one full stage of intersymbol echo interference channel described by the equation, yn = xn + 2\*xn-1 + xn-2 with channel inputs +1/-1, is shown in **figure 4.5.**

0 0

-1/0

1/2

1/1

-1/-3

1/0

1/-2

-1/-2

-1/-4

-1/2

1/4

-1/1

-1 -1

1/3

1/1

-1/-1

0 0

1 0

-1 0

1 1

-1 1

1 -1

-1 -1

1 -1

-1 1

1 1

1. For (5,3) linear block code over alphabet {-1, 0, +1), the codebook with message bits parity check bits will be as shown in **table 4.2**.

|  |  |
| --- | --- |
| **Message Bits (m1, m2, m3)** | **Parity Bits (p1, p2)** |
| 0 0 0 | 0 0 |
| 0 0 1 | 1 1 |
| 0 0 -1 | -1 -1 |
| 0 1 0 | 1 -1 |
| 0 1 1 | -1 -1 |
| 0 1 -1 | 0 1 |
| 0 -1 0 | -1 1 |
| 0 -1 1 | 0 -1 |
| 0 -1 -1 | 1 0 |
| 1 0 0 | 1 1 |
| 1 0 1 | -1 -1 |
| 1 0 -1 | 0 0 |
| 1 1 0 | -1 0 |
| 1 1 1 | 0 1 |
| 1 1 -1 | 1 -1 |
| 1 -1 0 | 0 -1 |
| 1 -1 1 | 1 0 |
| 1 -1 -1 | -1 1 |
| -1 0 0 | -1 -1 |
| -1 0 1 | 0 0 |
| -1 0 -1 | 1 1 |
| -1 1 0 | 0 1 |
| -1 1 1 | 1 -1 |
| -1 1 -1 | -1 0 |
| -1 -1 0 | 1 0 |
| -1 -1 1 | -1 1 |
| -1 -1 -1 | 0 -1 |

The trellis can be shown in **figure 4.6** with at most 3 states at each state.

To determine dmin, the formula stated in question 2 a) is used:

dmin = min wt(c)

By looking at the code book, the min wt(c) is two, thus dmin = 2.

The generator matrix is given by:

G = [1 0 0 1 1 1; 0 1 0 1 -1 1; 0 0 1 0 0 0]

The parity check matrix is given by [PT: - I n–k × n–k]

Thus H = [1 1 0 -1 0 0; 1 -1 0 0 -1 0; 1 1 0 0 0 -1]

1. Convolutional codes are called “convolutional” because they use a sliding application of boolean polynomial algebra. The sliding application represents the 'convolution' of the encoder over the data. This sliding nature facilitates using trellis for decoding over time-invariant trellis.

The output/input convolution equation for code in part (a), where c1n and c2n are the outputs of codewords c1 and c2 and xn is the input at stage n is and their impulse responses are:

c1n = xn + xn-3 + xn-4

hc1[n] = δ[n] + δ[n-3] + δ[n-4]

c2n = xn + xn-1 + xn-3 + xn-4

hc2[n] = δ[n] + δ[n-1] + δ[n-3] + δ[n-4]

The output/input convolution equation for code in part (c), where yn is the output and xn is the input at stage n is:

yn = xn + 2\*xn-1 + xn-2

The impulse response of code in part (c) is:

hy[n] = δ[n] + 2×δ[n-1] + δ[n-2]