A close up of a logo

Description automatically generatedA close up of a sign

Description automatically generated

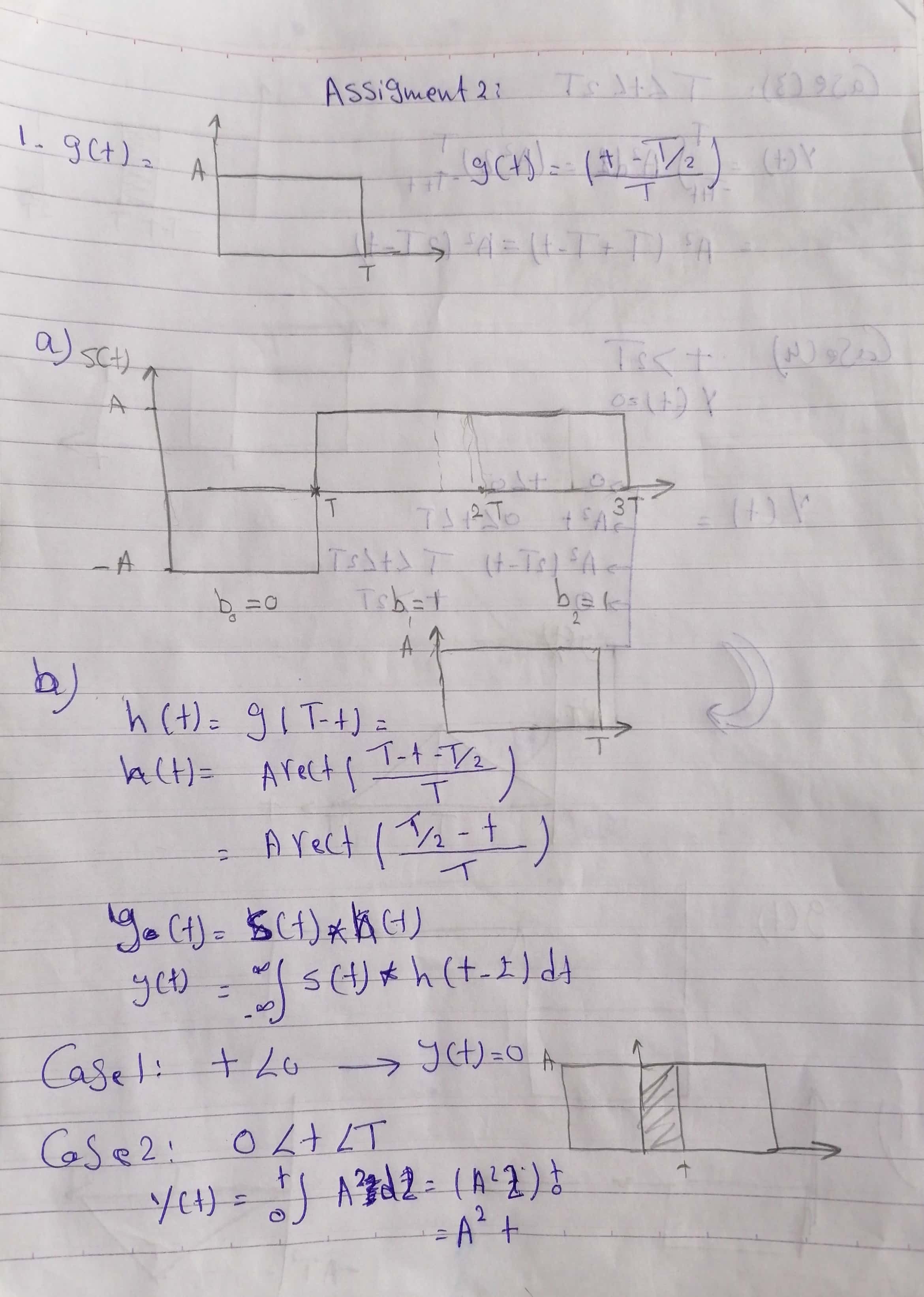
Assignment2

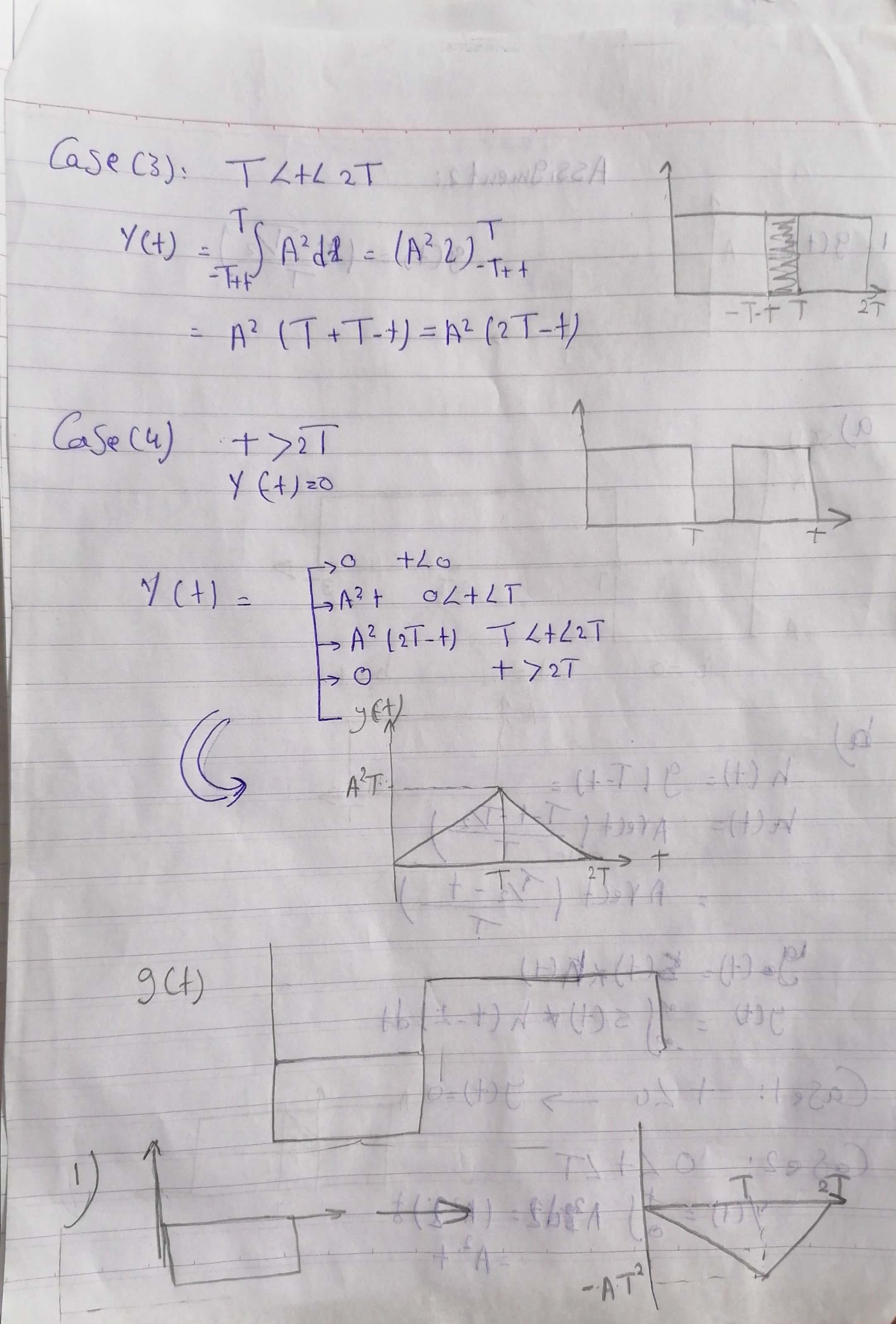
# *Team Members:*

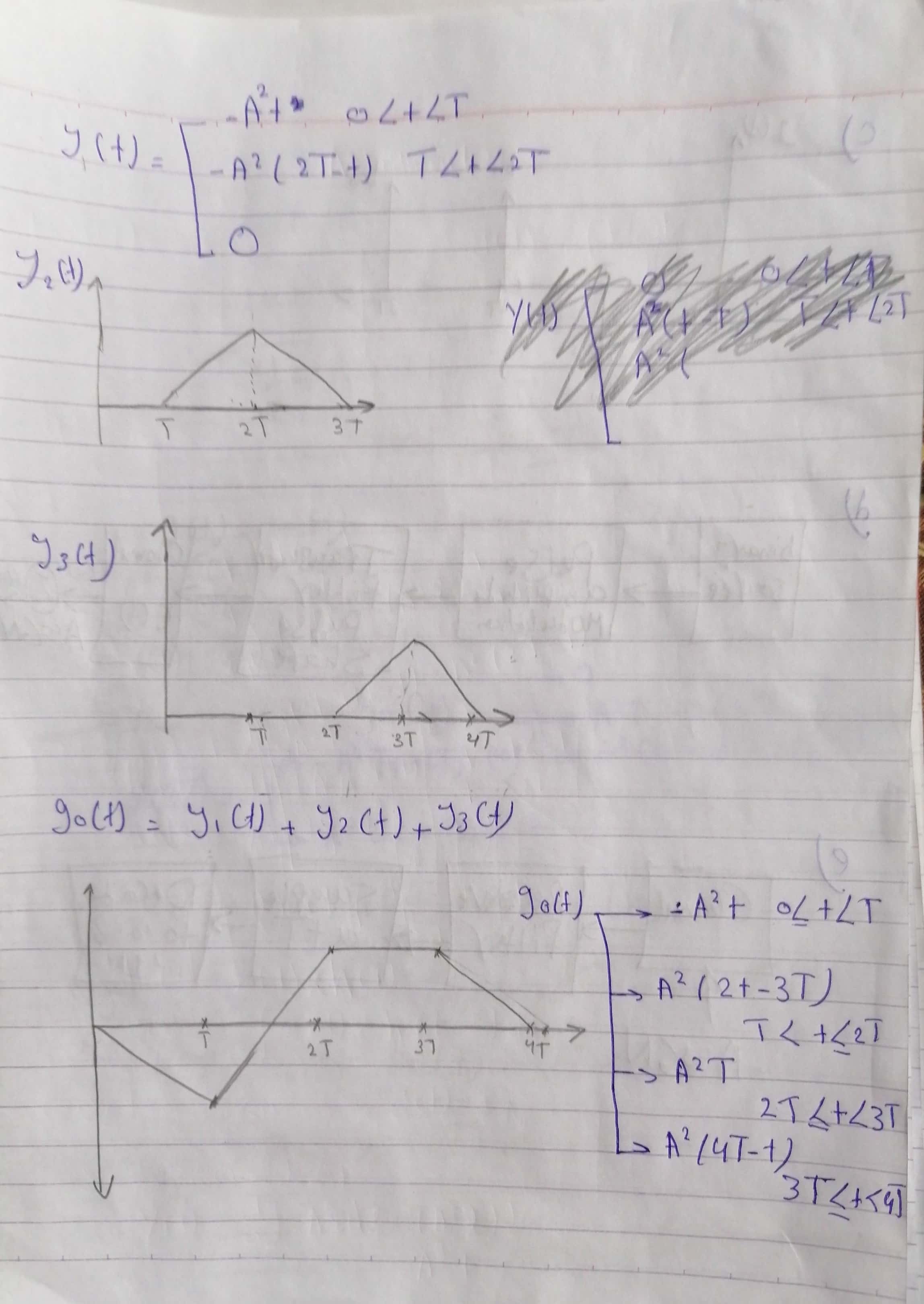
|  |  |  |
| --- | --- | --- |
| **Name** | **Sec** | **BN** |
| Zeinab Moawad Fayez Hassan | 1 | 29 |
| Basma Hatem Farid | 1 | 17 |

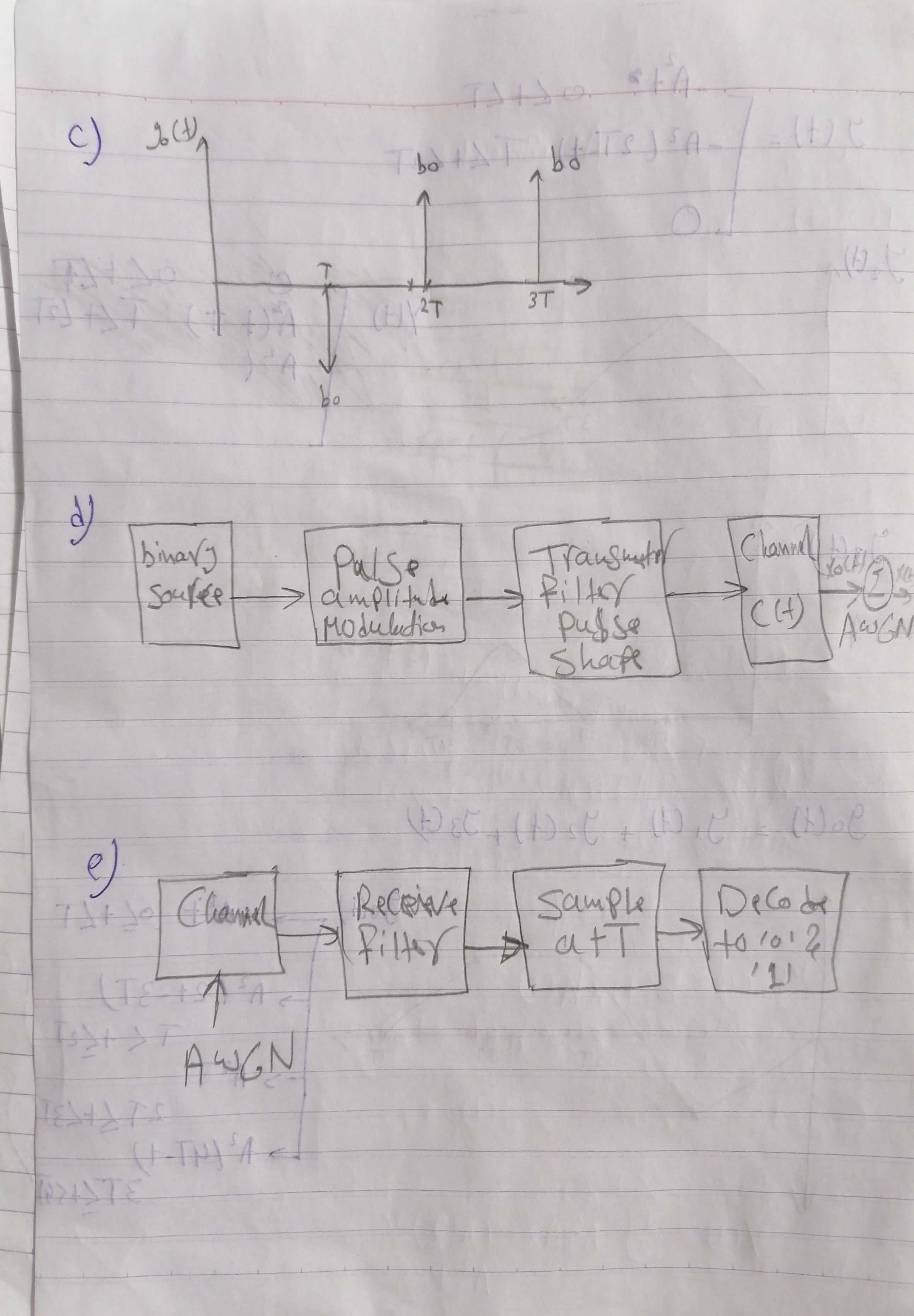
Sumbitted To:

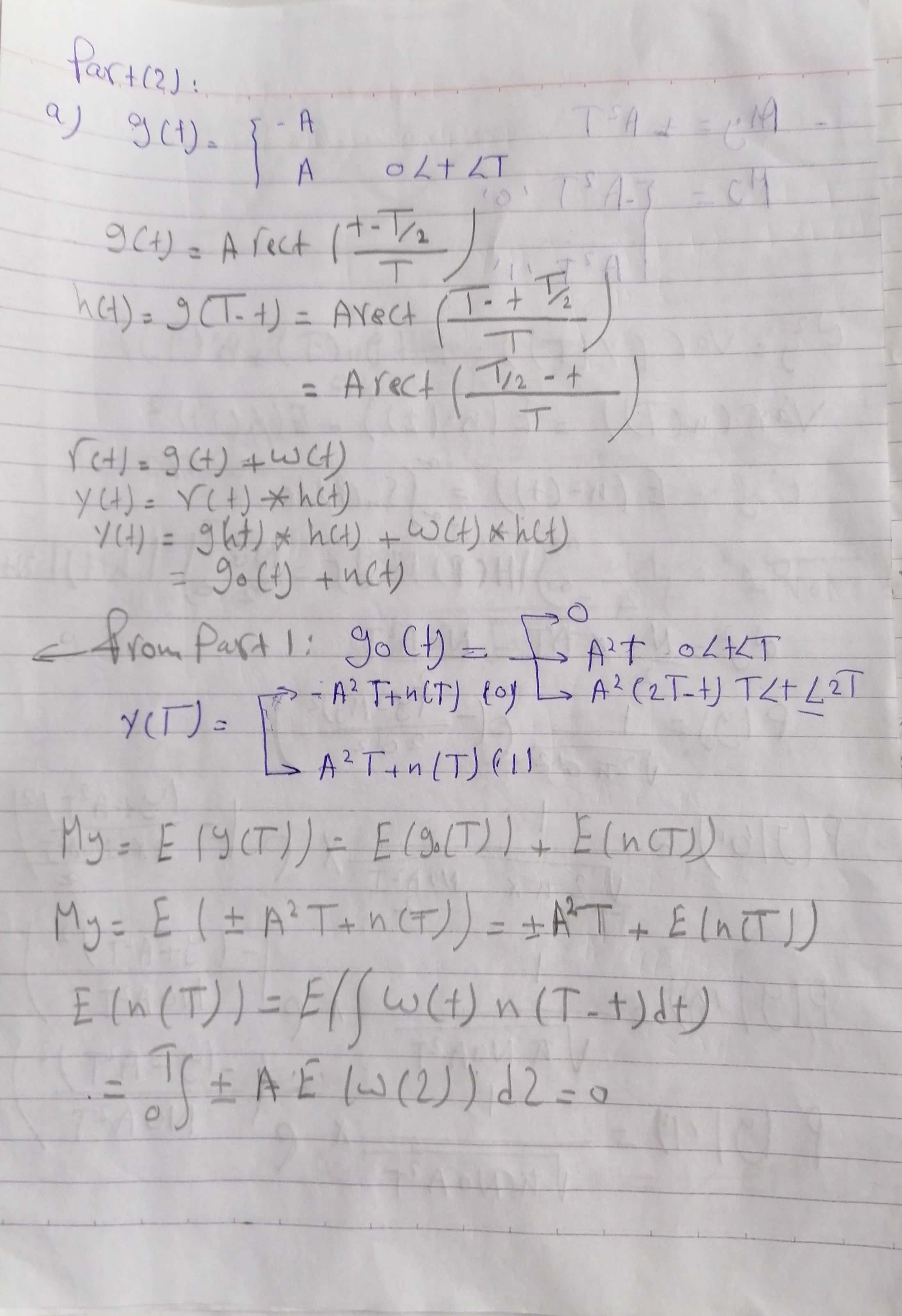
Eng: Mohamed Khaled

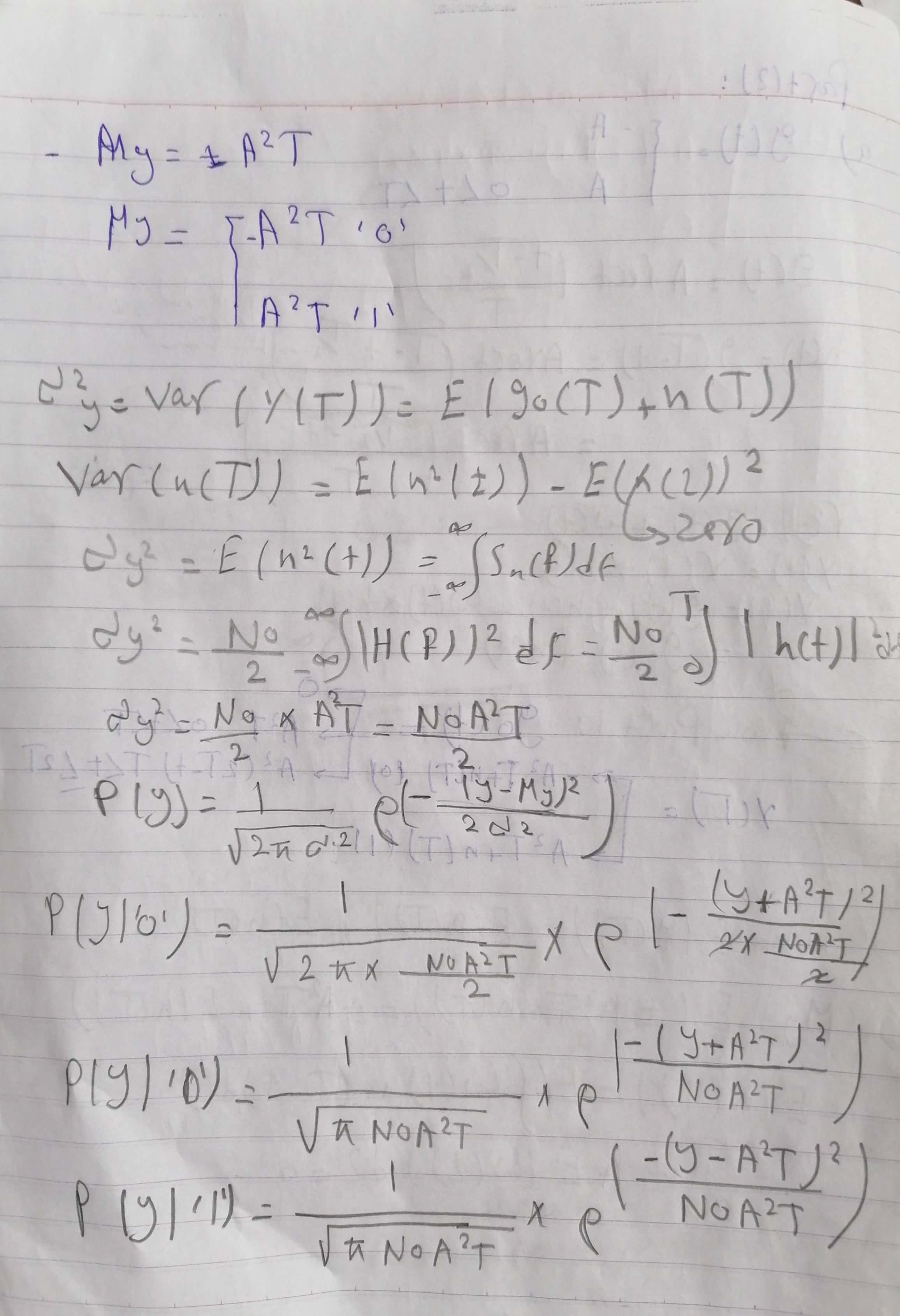


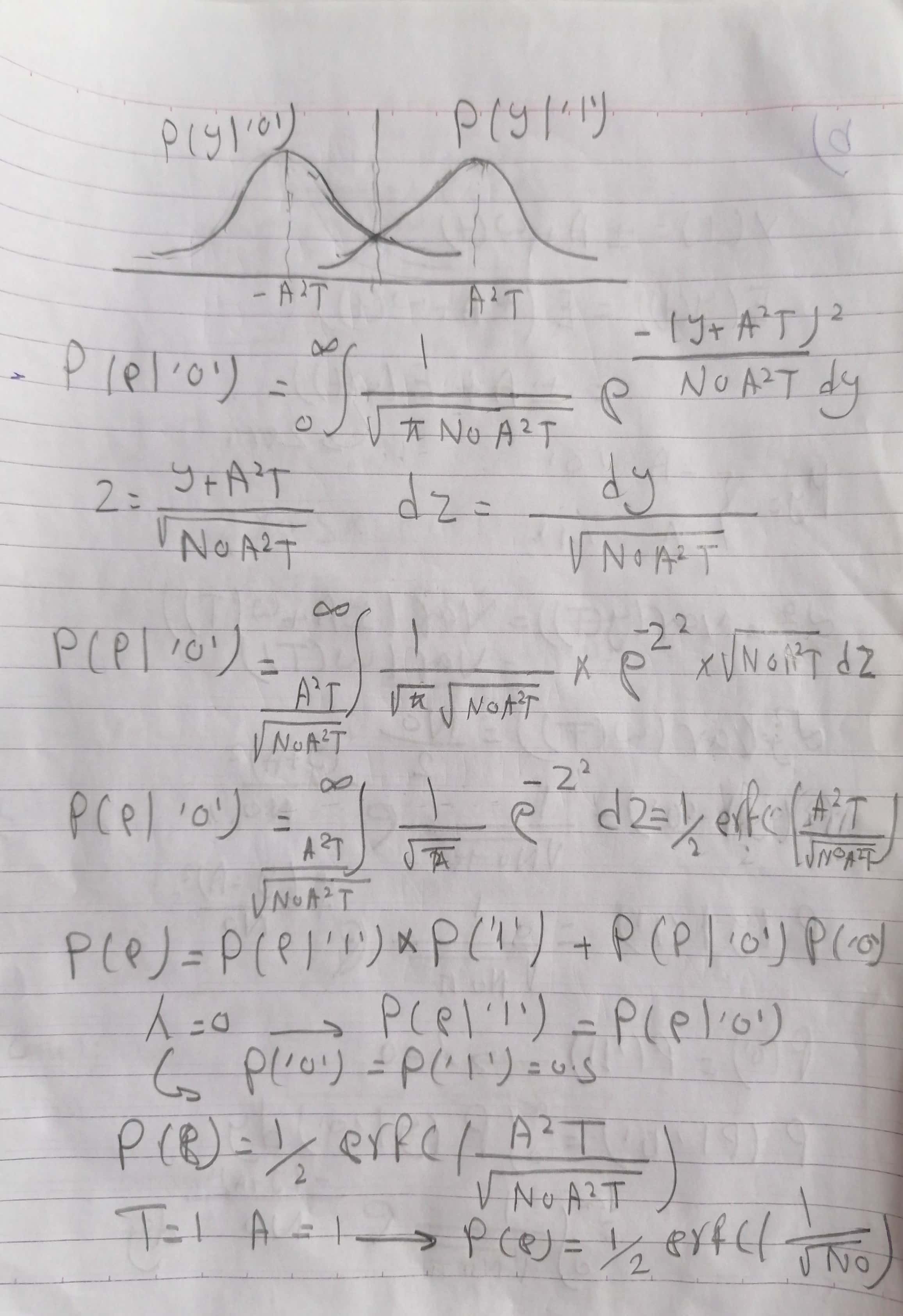


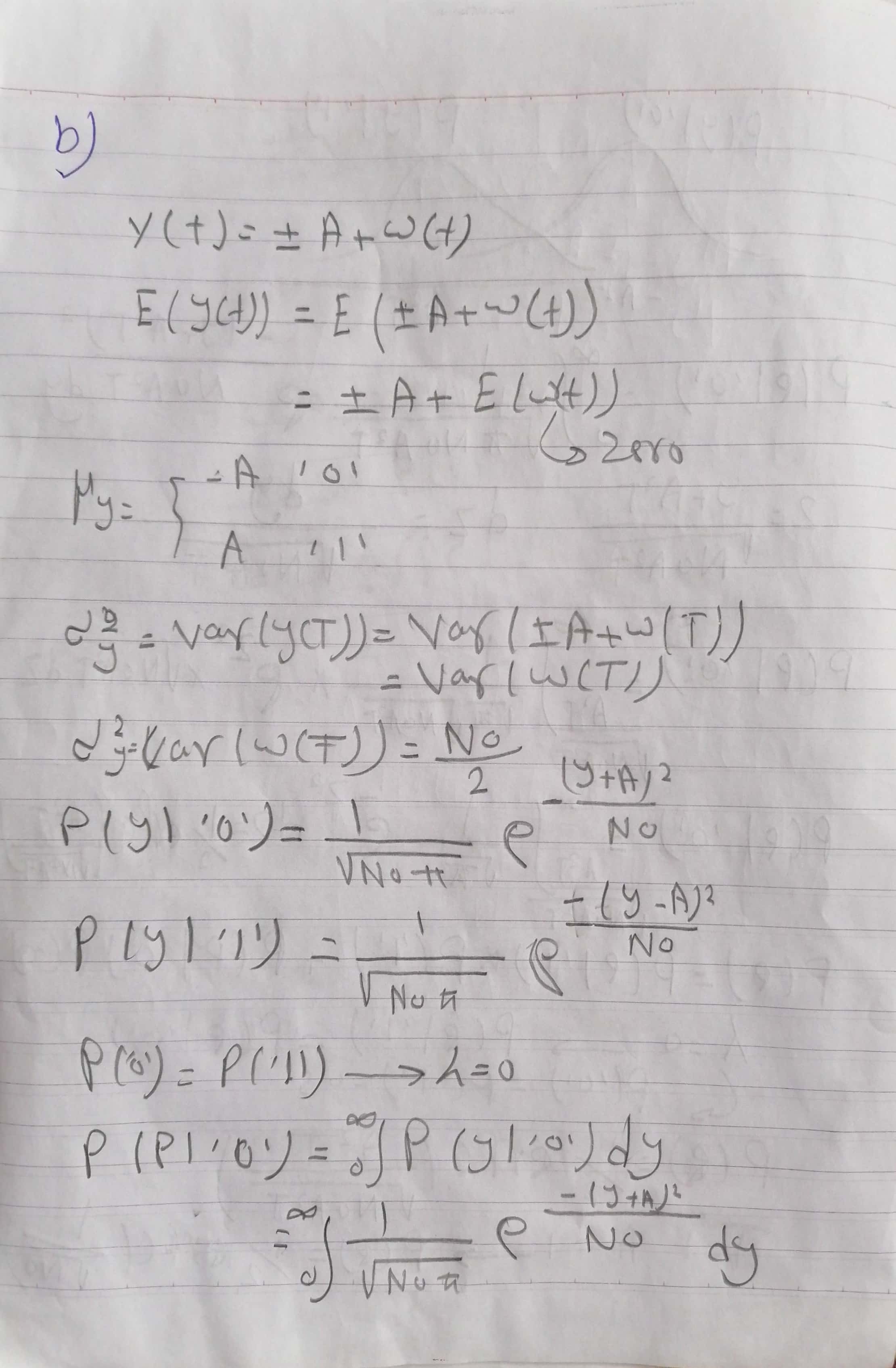


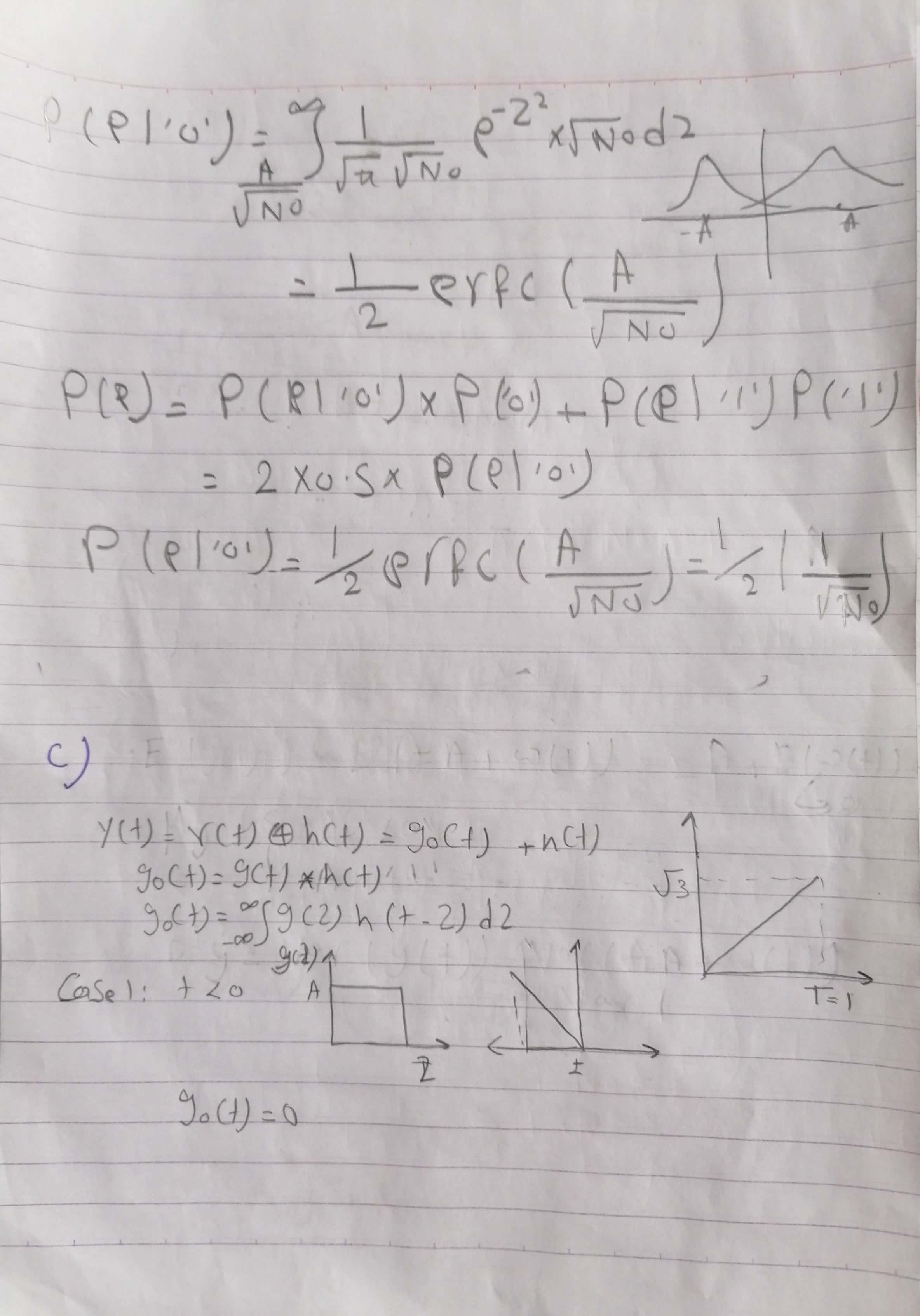


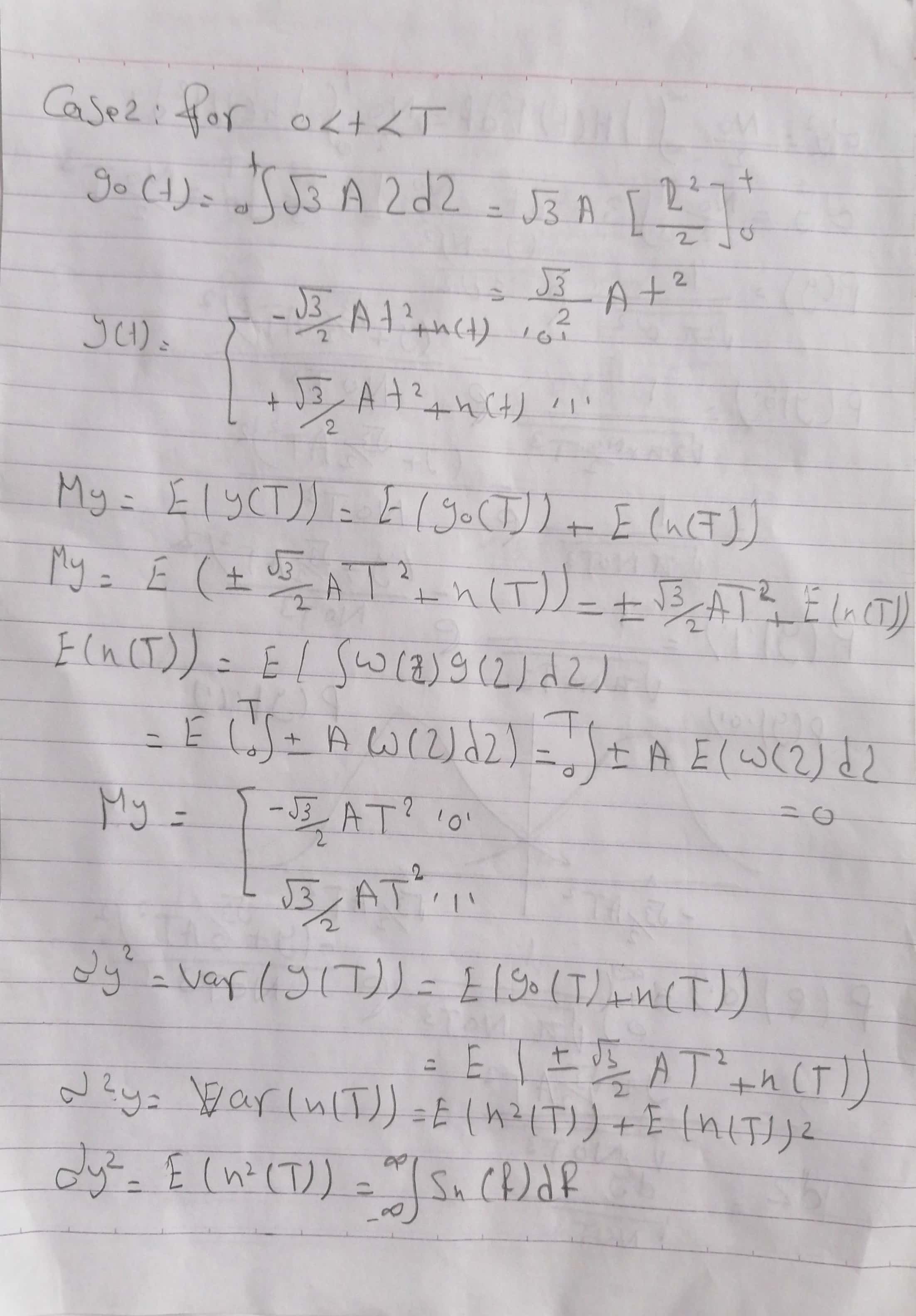


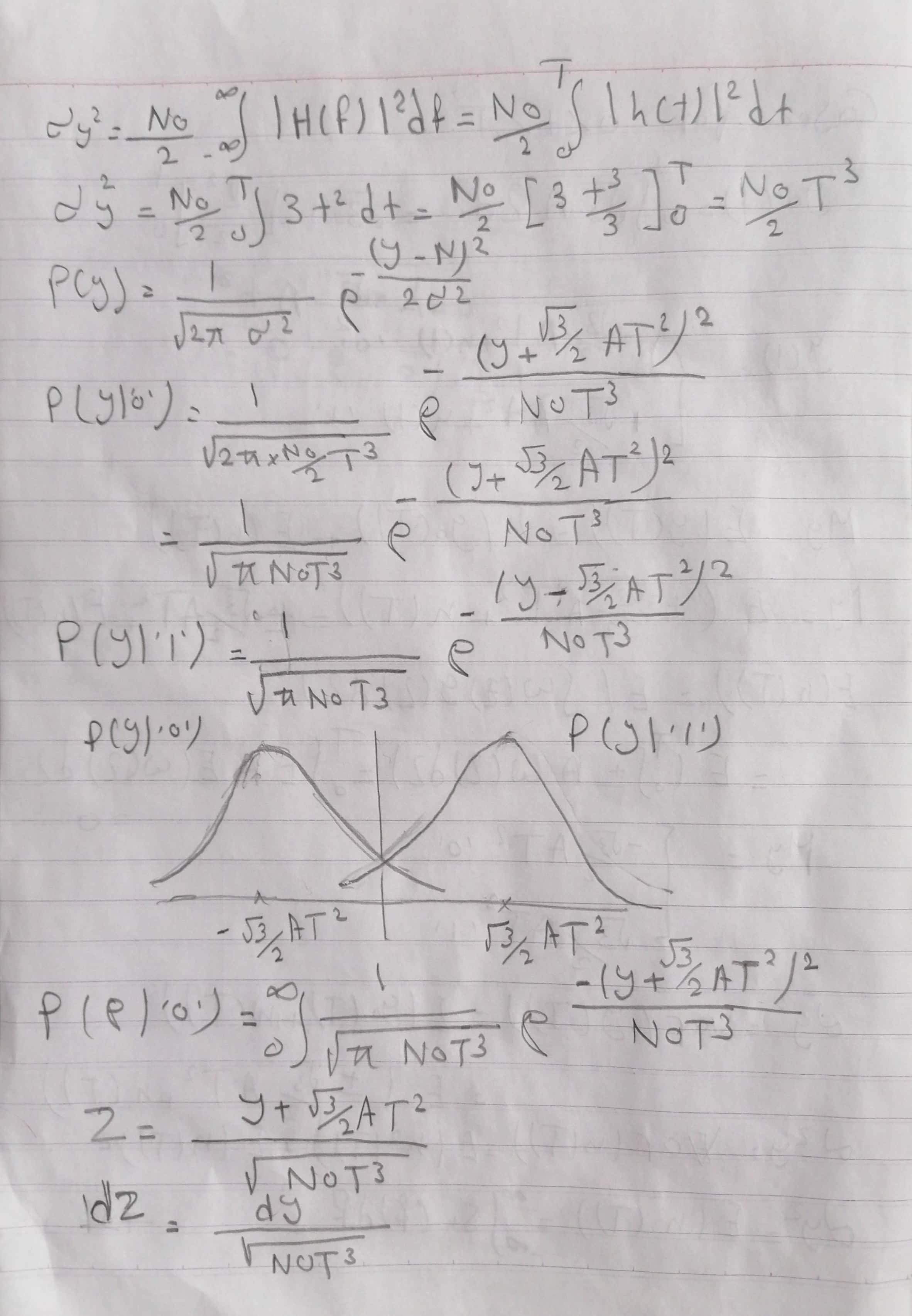


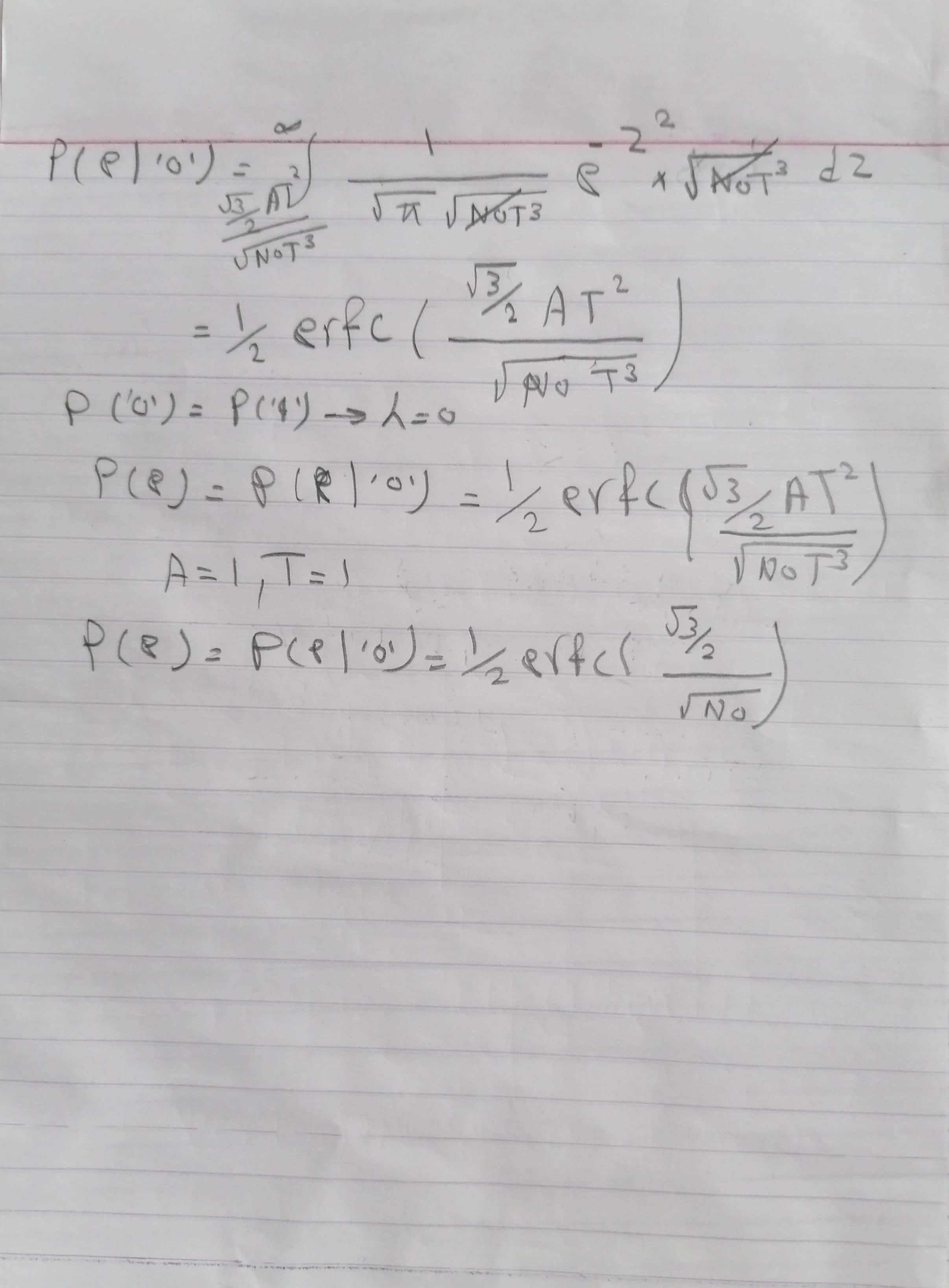






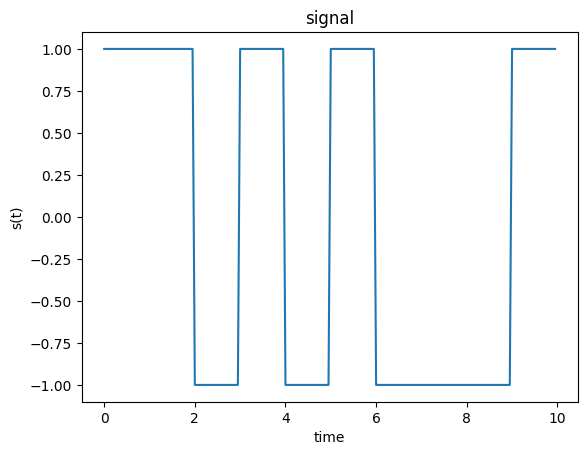


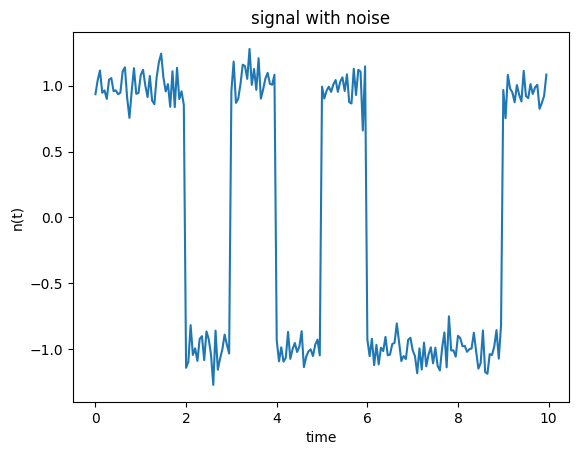




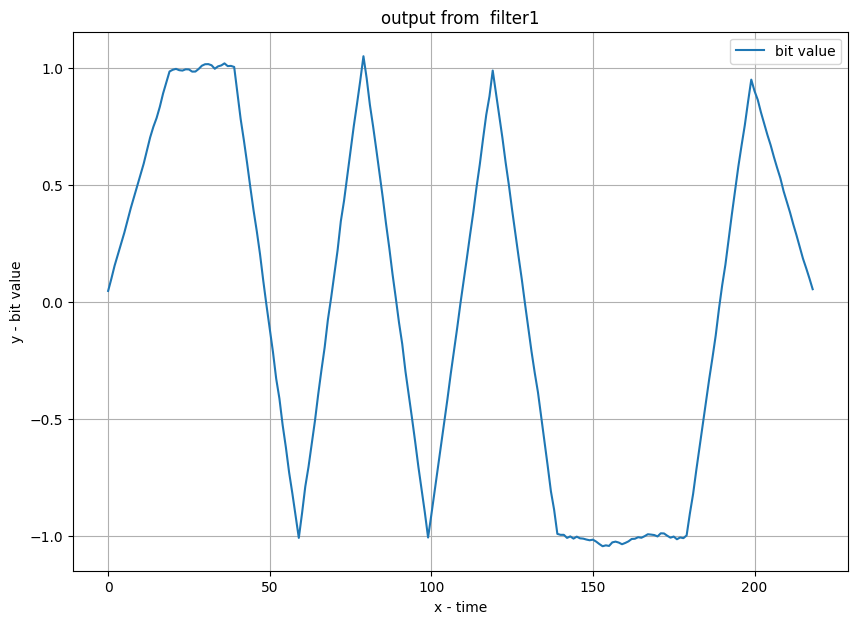
4- the required figures:

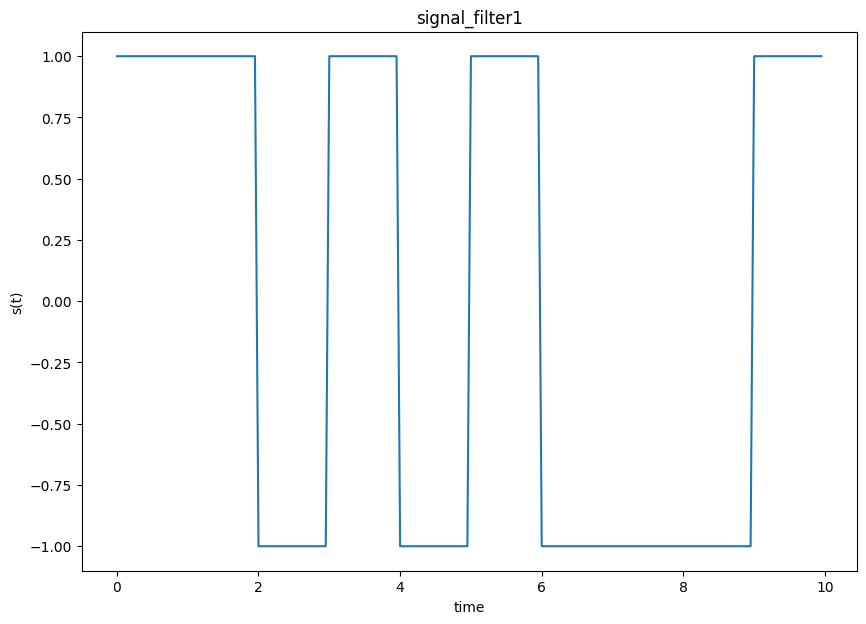
input bitstream**:** 1 1 0 1 0 1 0 0 0 1 with number of samples =20

****

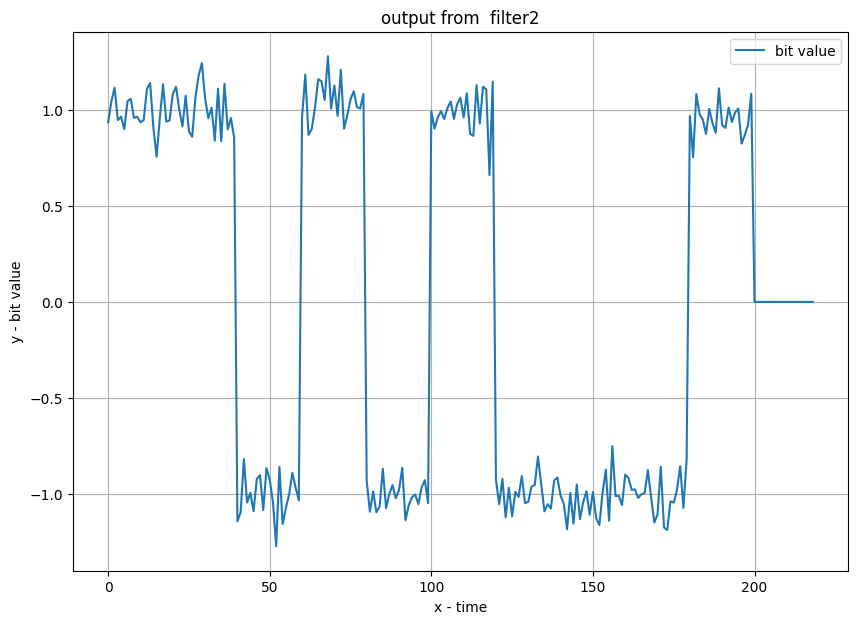


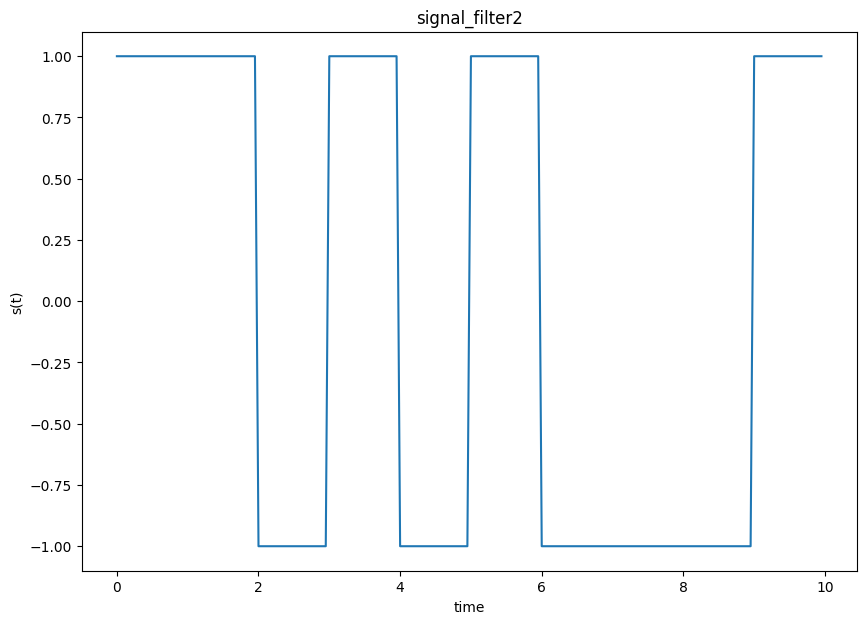
Output due to matched filter:

****

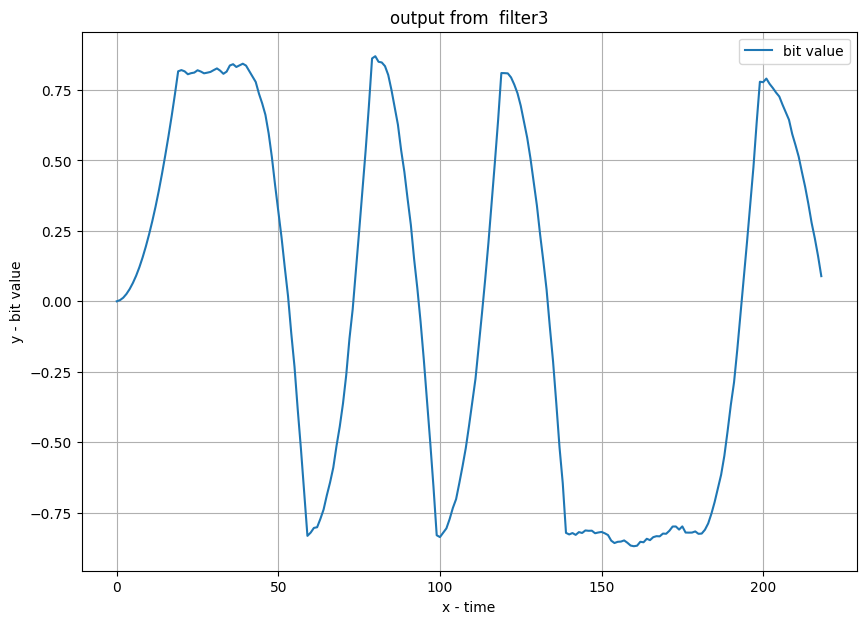
****

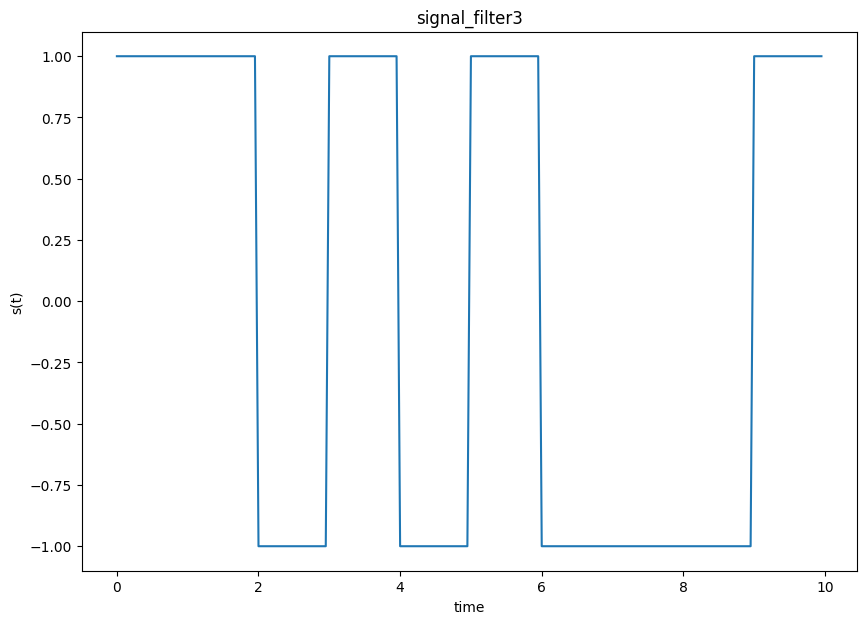
Output due to no filter:

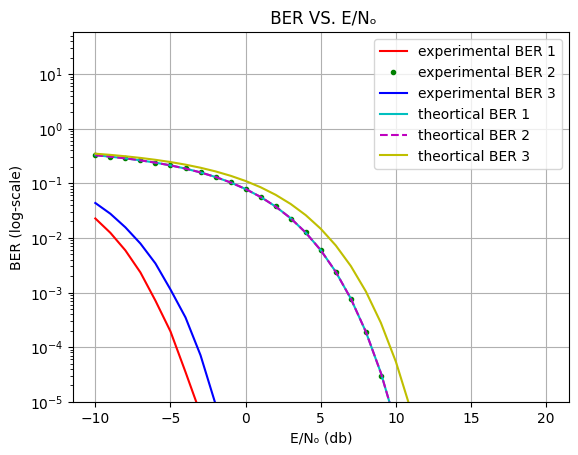




Output due to linear filter:





4-number of bits:1000000 , with number of samples =20 

Comments to 5 and 6:

5-Is the BER an increasing or a decreasing function of 𝑬/𝑵𝒐? Why?

Ans:The BER is decreasing as a function of E/No as in the plot. Due to two reasons:

1 - As E/No increases (here E is constant) No decreases and thus sigma which means the added AWGN involves less variations (corresponds to a thinner Gaussian distribution) and thus any noise added corresponds to small values close to zero which do not affect the signal that much (hence BER decreases)

2 -The relationship between E/No ratio and BER is not linear, but logarithmic

   BER decreases exponentially as the E/No ratio increases

6-Which case has the lowest BER? Why?

Ans:The matched filter case is the lowest BER since it uses a filter matched to the pulse to minimize the probability of error so it is maximizes the peak pulse SNR at the samples.

And it is designed to match the characteristics of the transmitted signal

and this makes the filter provides the best detection strategy

by correlating the received signal with a replica of the transmitted signal

so as a result the BER will be reduced and the SNR will be maximized

Note : Theoritcally using a filter or not yields the same expression due to our assumptions on variance and PSD.

Code:

first try each filter:

\*initial parameter:

#simulation parameters

n=10                       #number of bits

step=0.05                  #simulation time step (one pulse of duration 1 has 20 samples)

t=np.arange(0,n,step)

sigma\_noise = 0.1

1. Get binary bits and change to pulses:
2. bitstream = np.random.randint(0,2,10)
3. print("Bitstream: ",bitstream)
4. signal=binarycode\_to\_signal(bitstream,step)
5. plt.plot(t,signal)
6. plt.xlabel('time')
7. plt.ylabel('s(t)')
8. plt.title('signal')
9. plt.show()

2- Generate noise and add it to signal:

#generate the noise

signal\_noise=add\_AWGN\_noise(signal,len(signal),sigma\_noise)

plt.plot(t,signal\_noise)

plt.xlabel('time')

plt.ylabel('n(t)')

plt.title('signal with noise')

plt.show()

3- try 3 recieve filters

signal\_noise\_filter\_1=receive\_filter(signal\_noise, 1,step)

signal\_noise\_filter\_2=receive\_filter(signal\_noise, 2, step)

signal\_noise\_filter\_3=receive\_filter(signal\_noise, 3, step)

#ploting

plt.figure(figsize=(10,7))

plt.plot(range(0, signal\_noise\_filter\_1.flatten().shape[0]), signal\_noise\_filter\_1.flatten(), label = "bit value")

plt.xlabel('x - time')

plt.ylabel('y - bit value')

plt.title('output from  filter1')

plt.legend()

plt.grid()

plt.show()

#ploting

plt.figure(figsize=(10,7))

plt.plot(range(0, signal\_noise\_filter\_2.flatten().shape[0]), signal\_noise\_filter\_2.flatten(), label = "bit value")

plt.xlabel('x - time')

plt.ylabel('y - bit value')

plt.title('output from  filter2')

plt.legend()

plt.grid()

plt.show()

#ploting

plt.figure(figsize=(10,7))

plt.plot(range(0, signal\_noise\_filter\_3.flatten().shape[0]), signal\_noise\_filter\_3.flatten(), label = "bit value")

plt.xlabel('x - time')

plt.ylabel('y - bit value')

plt.title('output from  filter3')

plt.legend()

plt.grid()

plt.show()

4- Reconstructing bits

#sample the filtered signal

sampling\_period = int(1/step)

samples\_1 = sampling(sampling\_period, signal\_noise\_filter\_1)

samples\_2 = sampling(sampling\_period, signal\_noise\_filter\_2)

samples\_3 = sampling(sampling\_period, signal\_noise\_filter\_3)

#decode the samples

reconstructed\_bitstram\_1 = signal\_to\_binarycode(samples\_1, 0)

reconstructed\_bitstram\_2 = signal\_to\_binarycode(samples\_2, 0)

reconstructed\_bitstram\_3 = signal\_to\_binarycode(samples\_3, 0)

print('Reconstructed Bitstram 1:',reconstructed\_bitstram\_1)

print('Reconstructed Bitstram 2:',reconstructed\_bitstram\_2)

print('Reconstructed Bitstram 3:',reconstructed\_bitstram\_3)

#ploting

#generate the binary signal

signal\_filter1=binarycode\_to\_signal(reconstructed\_bitstram\_1,step)

#generate the binary signal

plt.figure(figsize=(10,7))

plt.plot(t,signal\_filter1)

plt.xlabel('time')

plt.ylabel('s(t)')

plt.title('signal\_filter1')

plt.show()

#ploting

#generate the binary signal

signal\_filter2=binarycode\_to\_signal(reconstructed\_bitstram\_1,step)

plt.figure(figsize=(10,7))

plt.plot(t,signal\_filter2)

plt.xlabel('time')

plt.ylabel('s(t)')

plt.title('signal\_filter2')

plt.show()

#ploting

#generate the binary signal

signal\_filter3=binarycode\_to\_signal(reconstructed\_bitstram\_1,step)

plt.figure(figsize=(10,7))

plt.plot(t,signal\_filter3)

plt.xlabel('time')

plt.ylabel('s(t)')

plt.title('signal\_filter3')

plt.show()

Second Campare BER of Different receive filter:

num\_of\_bits = 1000000

step = 0.05 #Samples per pulse of duration 1

T=1 # pulse period

# generate the binary symbols (1)

input\_bitstream = np.random.randint(0,2,num\_of\_bits)

# generate the binary signal (2)

g\_t = binarycode\_to\_signal(input\_bitstream, step)

# plot BER VS. E/No for each filter

E\_Nₒ=np.arange(-10, 21, 1)    # E\_Nₒ range

Nₒ = 1/(10\*\*(E\_Nₒ/10))

sigma = np.sqrt(Nₒ/2) # the range of sigma.

# Filter 1:

filter1\_BER, filter1\_BER\_th = np.zeros(len(sigma)), np.zeros(len(sigma))

# Filter 2:

filter2\_BER, filter2\_BER\_th = np.zeros(len(sigma)), np.zeros(len(sigma))

# Filter 3:

filter3\_BER, filter3\_BER\_th = np.zeros(len(sigma)), np.zeros(len(sigma))

for i in range(len(sigma)):

    # generate the noise

    s\_t = add\_AWGN\_noise(g\_t,len(g\_t), sigma[i])

    # apply the filter to the signal (4)

    y\_t\_1 = receive\_filter(s\_t,1, step)

    y\_t\_2 = receive\_filter(s\_t,2, step)

    y\_t\_3 = receive\_filter(s\_t,3, step)

    # sample the filtered signal (5)

    sampling\_period = int(T/step)

    y\_iT\_1 = sampling(sampling\_period, y\_t\_1,num\_of\_bits)

    y\_iT\_2 = sampling(sampling\_period, y\_t\_2,num\_of\_bits)

    y\_iT\_3 = sampling(sampling\_period, y\_t\_3,num\_of\_bits)

    # decode the samples (6)

    λ = 0      # due to Assumation

    bitstream\_output\_1 = signal\_to\_binarycode(y\_iT\_1, λ)

    bitstream\_output\_2 = signal\_to\_binarycode(y\_iT\_2, λ)

    bitstream\_output\_3 = signal\_to\_binarycode(y\_iT\_3, λ)

    filter1\_BER[i] = calc\_simulated\_BER(input\_bitstream,bitstream\_output\_1)

    filter1\_BER\_th[i] = calc\_theoretical\_BER(1/sigma[i])

    filter2\_BER[i] = calc\_simulated\_BER(input\_bitstream,bitstream\_output\_2)

    filter2\_BER\_th[i] = calc\_theoretical\_BER(1/sigma[i])

    filter3\_BER[i] = calc\_simulated\_BER(input\_bitstream,bitstream\_output\_3)

    filter3\_BER\_th[i] = calc\_theoretical\_BER(np.sqrt(3)/2\*1/sigma[i])

plotting:

#ploting BER

E\_Nₒ=np.arange(-10, 21, 1)

plt.semilogy(E\_Nₒ, filter1\_BER, 'r',label = "experimental BER 1")

plt.semilogy(E\_Nₒ, filter2\_BER, 'g.',label = "experimental BER 2")

plt.semilogy(E\_Nₒ, filter3\_BER, 'b',label = "experimental BER 3")

plt.semilogy(E\_Nₒ, filter1\_BER\_th, 'c',label = "theortical BER 1")

plt.semilogy(E\_Nₒ, filter2\_BER\_th, 'm--',label = "theortical BER 2")

plt.semilogy(E\_Nₒ, filter3\_BER\_th, 'y',label = "theortical BER 3")

plt.xlabel('E/Nₒ (db)')

plt.ylabel('BER (log-scale)')

plt.title(' BER VS. E/Nₒ')

plt.ylim(10\*\*(-5))

plt.legend()

plt.grid()

plt.show()

Functions Used:

def binarycode\_to\_signal(bitstream, step):

    T = 1

    A = 1

    pulse = np.ones(int(T/step))

    pulse = pulse\*A

    signal = np.zeros(len(bitstream)\*len(pulse))

    for i in range(len(bitstream)):

        if bitstream[i] == 1:

            signal[i\*len(pulse):(i+1)\*len(pulse)] = 1\*pulse

        else:

            signal[i\*len(pulse):(i+1)\*len(pulse)] = -1\*pulse

    return signal

# add noise to signal

def add\_AWGN\_noise(signal,n, sigma):

     w\_t =np.random.normal(0,sigma,n)

     s\_t = signal + w\_t

     return s\_t

# received filter

def receive\_filter(signal\_noise, filter\_num, step):

   filter\_num-=1

   filters = [np.ones(int(1/step)), np.ones(1), np.sqrt(3)\*np.arange(0, 1, step)]

   filter = filters[filter\_num]

   filter = np.concatenate((filter, np.zeros(int(1/step)-len(filter))))

   signal\_noise\_filter=np.convolve(signal\_noise, filter)

   if (filter\_num==0 or filter\_num==2):

      signal\_noise\_filter=signal\_noise\_filter\*step

   return signal\_noise\_filter

# sampling filter

def sampling(sampling\_period, signal\_noise\_filtered, n=10):

   samples = np.zeros(n)

   for i in range(len(samples)):

      samples[i] = signal\_noise\_filtered[sampling\_period-1+i\*sampling\_period]

   return samples

def signal\_to\_binarycode(samples, λ):

   return (samples>λ)\*1

def calc\_simulated\_BER(input\_bitstream , bitstream\_output):

    sim\_BER=np.sum(input\_bitstream != bitstream\_output)/len(input\_bitstream)

    return sim\_BER

def calc\_theoretical\_BER(erfc\_parameter):

    return 0.5 \* math.erfc(erfc\_parameter/math.sqrt(2))