Groepswerk cursus Communicatietheorie : opgave

I. Inleiding

In dit project dat aansluit bij de cursus *Communicatietheorie* versturen we een audiobestand over een (gemodelleerd) communicatiekanaal. Hierbij zul je de verschillende segmenten van het communicatiesysteem moeten implementeren in de softwareomgeving Matlab (beschikbaar via Athena) of in de programmeertaal Python, waarna de eigenschappen en prestaties van elk segment worden onderzocht en besproken. Als hulp zijn er op Ufora enkele Matlab-/Python-bestanden beschikbaar waarin sommige functionaliteiten al geïmplementeerd zijn. Het is de bedoeling dat je deze bestanden aanvult waar nodig en deze op het eind van het project mee indient.

Bij het project dient ook een verslag gemaakt te worden waarin je onderstaande vragen beantwoordt en jouw resultaten illustreert aan de hand van figuren. Merk op dat een verslag niet zomaar een opsomming is van antwoorden op de vragen en zorg er ook voor dat de figuren duidelijk zijn! Een elektronische versie (pdf) van het verslag moet samen met alle Matlab code en enkele audiobestanden **ingediend worden ten laatste op vrijdag 10 december 2021 vóór 17u00** en dit via de opdracht van jullie groep op Ufora in een zipbestand met de naam *groepXX.zip*. Geef in het verslag ook een overzicht van de taakverdeling binnen de groep van wie wat gedaan heeft.

II. Opmerkingen over het gebruik van Matlab

Matlab is een technische softwareomgeving en wordt gebruikt voor tal van wiskundige toepassingen. In dit project modelleren we er een communicatiekanaal in. Een uitgebreide beschrijving van de syntaxis, de verschillende functies en hoe te werken met Matlab is te vinden in de online documentatie[1]. Verder vind je op Ufora ook nog een document waarin de basis uitgelegd wordt aan de hand van voorbeelden. Met het oog op een vlotte implementatie van het project worden hier nog enkele zaken benadrukt.

- Een Matlab file bevat een script, een functie of een klasse. Een script voert een reeks van commando's uit en is voornamelijk handig om code die verschillende functies en klassen oproept in te bewaren om die zo te kunnen hergebruiken. Functies en klassen in Matlab zijn vrij gelijkaardig aan functies en klassen in andere programmeertalen. In dit project krijg je een aantal files met klassen waarin enkele statische functies moeten aangevuld worden. Om deze functies te testen en om de verschillende simulaties uit te voeren is het handig om daarvoor verschillende scripts te schrijven.
- Matlab werkt intern voornamelijk met matrices en vectoren. Het vergt dan ook vaak veel minder tijd om iets te bereken als het geïmplementeerd is aan de hand van matrices en vectoren in vergelijking met een implementatie die lussen bevat. In dit project is dit vooral het geval voor het stuk 'Kanaalcodering' en 'Modulatie en Detectie'.
- In Tabel 1 vind je enkele nuttige Matlab commando's die je kan gebruiken in het project. Wanneer er vermeld wordt dat je zelf iets moet implementeren mag je wel geen ingebouwde functies gebruiken die hetzelfde doen.
- In dit project zul je ook werken met anonieme functies. Een anonieme functie is een functie die geassocieerd wordt met een variable van het type *function_handle.* Net zoals gewone functies accepteren deze functies input variabelen en geven ze één output variabele terug. Als we bijvoorbeeld een anonieme functie willen definiëren om het kwadraat van een getal te berekenen, schrijven we

  ```
  sqr = @(x) x.^2;
  ```

  Hierbij is de variable 'sqr' van het type *function_handle.* De input variabelen bevinden zich tussen de haakjes na de @ operator. Om dan het kwadraat van 5 te bereken, gebruik je volgende regel:

[1] https://nl.mathworks.com/help/matlab/

Tabel I
ENKELE NUTTIGE MATLAB COMMANDO'S

functienaam	*omschrijving*
a./b en a.*b	Matrices a en b puntsgewijs delen of vermenigvuldigen
integral	Numeriek berekenen van een integraal
reshape, kron, repmat	Functies om matrices te vervormen of uit te breiden
de2bi, bi2de	Zet integers om naar bits en omgekeerd
sort, find	Sorteer of zoek een vector in een matrix
rand,randn	Genereren van random data
plot, histogram, bar,...	Functies om te plotten van data
mod	Bereken van de modulus
conv	Bereken van de convolutie of product van veeltermen
max, min	Bereken van minimum en maximum
ones, zeros, eye	Genereer matrix met enkel enen/nullen, eenheidsmatrix
sum, prod	Berekent som/product van alle vectorelementen

```
sqr(5)
```

wat uiteraard 25 oplevert. In Matlab hebben sommige functies, zoals *integral*, een *function_handle* als argument. Om bijvoorbeeld de integraal van x^2 te bereken van 0 tot 1 gebruik je volgend commando:

```
q = integral(sqr,0,1);
```

Verder hoef je niet altijd een variable aan te maken en kan je ook een *function_handle* gebruiken in de definitie van een andere *function_handle.* Zo kan je de integraal over x^4 van 0 tot 1, bereken op volgende twee manieren:

```
q = integral(@(x) x.^4,0,1);
```

```
q = integral(@(x) sqr(x).^2,0,1);
```

Bemerk ten slotte dat we in de definitie van de verschillende anonieme functies telkens gebruik maken van '.^' (elementsgewijs) in plaats van '^'. Dit doen we omdat de *integral* functie van Matlab verwacht dat de meegegeven anonieme functie een vector heeft als input en een vector van dezelfde grootte heeft als output.

III. OPMERKINGEN OVER HET GEBRUIK VAN PYTHON

De programmeertaal Python aangevuld met enkele uitbreidingspakket zoals 'NumPy', 'SciPy' en 'Matplotlib' vormt een goed alternatief voor Matlab. 'NumPy' laat je toe om multi-dimensionale rijen en matrices te definiëren en stelt ook veel wiskundige functies ter beschikking. In 'SciPy' zijn er verschillende basisfuncties geïmplementeerd, zoals bijvoorbeeld de fft. Tot slot zorgt de package 'Matplotlib' ervoor dat je jouw data op een overzichtelijke manier kunt voorstellen. Door deze drie uitbreidingspakketten met elkaar te combineren is er voor bijna alle functies in Matlab een alternatief in Python. Ook hier is er een uitgebreide documentatie online beschikbaar[2] en op Ufora vind je een document waarin de basis is uitgelegd. Voor de rest zijn er analoge opmerkingen geldig als bij Matlab.

- In dit project krijg je een aantal files waarbij je enkele functies binnen een klasse moet aanvullen. Hiervoor is het opnieuw aan te raden om de code tijdens het testen in een ander bestand te schrijven.

[2]https://docs.scipy.org/doc/ en https://matplotlib.org/

Tabel II
ENKELE NUTTIGE PYTHON COMMANDO'S

functienaam	*omschrijving*
a/b en a*b	Matrices a en b puntsgewijs delen of vermenigvuldigen
integrate.quad	Numeriek berekenen van een integraal
np.linspace	Maak een rij met x punten tussen twee grenzen
np.reshape, np.kron, np.matlab.repmat	Functies om matrices te vervormen of uit te breiden
np.sort, np.where	Sorteer of zoek een vector in een matrix
np.random.rand,np.random.randn	Genereren van random data
plt.plot, plt.hist,...	Functies om data te plotten
np.mod, %	Bereken van de modulus
np.convolve	Bereken van de convolutie of product van veeltermen
np.max, np.min	Bereken van minimum en maximum
np.ones, np.zeros, np.eye	Genereer matrix met enen/nullen, eenheidsmatrix
np.sum, np.prod	Berekent som/product van alle vectorelementen

- Ook Python zal sneller werken als je het aantal lussen in je code beperkt. Gebruik dus ook in Python matrices en vectoren.
- In Tabel 2 vind je enkele nuttige Python commando's die je kan gebruiken in het project. Je mag opnieuw geen ingebouwde functies gebruiken wanneer er gevraagd wordt om zelf iets te implementeren.
- Ook Python kent anonieme functies, al is de syntax lichtjes anders. Hier schrijven we

```
sqr = Lambda x: x**2
```

IV. DE COMPONENTEN VAN HET DIGITAAL COMMUNICATIESYSTEEM

Het doel van het communicatiesysteem is het versturen van een audiobestand van de zender naar de ontvanger. De originele wav-file is opgebouwd uit monsterwaarden die gekwantiseerd zijn met 16 bits. Dit betekent dat de originele data 65536 verschillende waarden kan innemen tussen 1 en -1. Door dit grote aantal, beschouwen we in dit project de input van de zender dan ook als een continue grootheid U met een waarschijnlijkheidsdichtheidsfunctie (w.d.f.) f_U.

A. Zender

De zender heeft als taak de informatie die aangeboden wordt door een bron om te zetten in een fysisch informatiesignaal dat kan verzonden worden over het kanaal. Volgende stappen worden daarvoor uitgevoerd:

- Eerst wordt het originele geluidsfragment beperkt in grootte door elke monsterwaarde te kwantiseren met slechts $M = 2^6 = 64$ waarden tussen 1 en -1. In subsectie V-A zullen we hiervoor verschillende *kwantisator*s ontwerpen en met elkaar vergelijken. Een kwantisator wordt gespecifieerd aan de hand van $M + 1$ *kwantisatiedrempels* ($-\infty = r_0 < r_1 < ... < r_M = +\infty$) en M reconstructieniveaus ($q_1 < q_2 ... < q_M$), met $q_i \in [r_{i-1}, r_i]$. Met elk *kwantisatieinterval* $[r_{i-1}, r_i]$ wordt een verschillend *bronsymbool* s_i geassocieerd. De kwantisator beeldt alle waarden u van U gelegen in $[r_{i-1}, r_i]$ af op s_i.
- De bronsymboolsequentie aan de uitgang van de kwantisator wordt vervolgens omgezet in een informatiebitsequentie. Dit gebeurt door de bronsymbolen te groeperen in *macrosymbolen* en aan elk

macrosymbool een unieke *informatiebit*sequentie toe te kennen. Deze omzetting wordt (verliesloze) *broncodering* genoemd. Elk macrosymbool bestaat uit n bronsymbolen. Er zijn dus M^n verschillende macrosymbolen. We spreken over *vectorcodering met dimensie* n als $n > 1$ en over *scalaire codering* als $n = 1$. Indien de bitsequenties die worden toegekend aan de verschillende macrosymbolen allemaal even lang zijn, dan spreken we over *vaste-lengte codering*. De sequentie bronsymbolen bevat echter meestal redundantie, dit is overtollige informatie. De te versturen sequentie van informatiebits kan dan worden ingekort of gecomprimeerd door n te laten toenemen en/of door een kortere bitsequentie toe te kennen aan veel voorkomende macrosymbolen en een langere bitsequentie aan zeldzame macrosymbolen (variabele-lengte codering). Voor een gegeven waarde van n, is het gebruik van *Huffmancodering* optimaal. Huffmancodering en -decodering is kort uitgelegd in **Appendix A**. In het geval van optimale codering gelden de volgende *entropiegrenzen*:

$$H \leq \bar{\ell}_{\min} < H + 1, \tag{1}$$

met

$$H = -\sum_{k=1}^{M^n} p_k \log_2 p_k \tag{2}$$

en

$$\bar{\ell}_{\min} = \ell_k p_k. \tag{3}$$

In bovenstaande formules stelt p_k de waarschijnlijkheid van het $k$de macrosymbool voor, en is ℓ_k de lengte van de informatiebitsequentie die gebruikt wordt om het $k$de macrosymbool voor te stellen. De opeenvolgende informatiebitsequenties worden aan elkaar geplakt tot één continue informatiebitsequentie. Broncodering wordt verder behandeld in subsectie V-B.

- Als we informatiebits versturen over een kanaal, worden onvermijdelijk fouten geïntroduceerd (deze werden tot nu toe genegeerd). Om onze informatiebitsequentie te beschermen tegen dergelijke fouten, voegen we op gecontroleerde wijze opnieuw redundantie toe. Deze procedure wordt *kanaalcodering* genoemd (subsectie V-C) en resulteert in een nieuwe, langere bitsequentie. Om te encoderen wordt de continue informatiebitsequentie (uitgang bronencoder) eerst opgesplitst in *informatiewoorden* van k bits en vervolgens worden aan elk informatiewoord $(n - k)$ redundante bits toegevoegd. Op die manier verkrijgen we een verzameling $\mathcal{C}$ van 2^k geldige *codewoorden* van n bits. In de cursus ligt de nadruk op *lineair*e blokcodes. Een blokcode is lineair als het volgende geldt: als $\boldsymbol{c}_1$ het codewoord is dat hoort bij $\boldsymbol{b}_1$, en als $\boldsymbol{c}_2$ het codewoord is dat hoort bij $\boldsymbol{b}_2$, dan is $\boldsymbol{c}_1 + \boldsymbol{c}_2$ het codewoord dat hoort bij $\boldsymbol{b}_1 + \boldsymbol{b}_2$. Het verband tussen de informatiewoorden $\boldsymbol{b}$ en de codewoorden $\boldsymbol{c}$ is lineair en wordt beschreven door de binaire (modulo-2) matrixoperatie $\boldsymbol{c} = \boldsymbol{bG}$, met $\boldsymbol{G}$ de *generatormatrix* van de code.
- Vervolgens (subsectie V-D) wordt de bitsequentie klaargemaakt voor verzending. Daarvoor wordt deze opgedeeld in blokken van m bits. Met elk van deze blokken laat men een complex symbool corresponderen (symbol *mapping*). De verzameling van mogelijke complexe symbolen vormt de *constellatie*. Deze bevat 2^m symbolen, waarvoor geldt dat (normeringsvoorwaarde)

$$\mathbb{E}[|a_k|^2] = 1. \tag{4}$$

In dit project passen we *Gray-mapping* toe: bitsequenties die gemapped worden op naburige constellatiepunten (afstanden te bekijken in het complexe vlak) verschillen in zo weinig mogelijk posities van elkaar. We zullen drie verschillende constellaties met elkaar vergelijken: ***BPSK*** of $\{-1, 1\}$, ***4QAM*** of $\left\{\frac{1+j}{\sqrt{2}}, \frac{-1+j}{\sqrt{2}}, \frac{-1-j}{\sqrt{2}}, \frac{1-j}{\sqrt{2}}\right\}$ en ***4PAM*** of $\left\{\frac{-3}{\sqrt{5}}, \frac{-1}{\sqrt{5}}, \frac{1}{\sqrt{5}}, \frac{3}{\sqrt{5}}\right\}$. De sequentie van complexe symbolen $\{a_k\}$ wordt vervolgens op pulsen $p(t - kT)$ geplaatst en over het kanaal verstuurd. Er geldt

$$\int_{-\infty}^{\infty} p\,(t)\, p\,(t - kT)\,\mathrm{d}t = \delta_k.$$

We gebruiken een *square-root raised cosine puls* in dit project. De (enkelzijdige) bandbreedte van zo een puls is $\frac{1+\alpha}{2T}$, waarbij T het *symboolinterval* en α de roll-off factor voorstelt. De zender creëert dus het volgende signaal:

$$x(t) = \sqrt{E_s} \sum_{k=0}^{K-1} a_k p(t - kT), \tag{5}$$

waarbij E_s de verzonden energie per symbool voorstelt. Later zullen we de verzonden energie per informatiebit E_b als referentie gebruiken. Er geldt dat

$$E_s = m\frac{k}{n}E_b. \tag{6}$$

In het vervolg van dit project zullen we altijd veronderstellen dat $E_s = 1$. Het complexe basisband-signaal $x(t)$ wordt in de laatste stap van de zender op een draaggolf met frequentie f_0 geplaatst. Het resulterende (reële) signaal heeft volgende vorm:

$$s(t) = \sqrt{2}\Re[x(t)e^{j2\pi f_0 t}]. \tag{7}$$

B. Kanaal

Het signaal $s\,(t)$ wordt verzonden over het kanaal, dat gemodelleerd wordt door middel van een schalingsfactor h_{ch} en reële Gaussiaanse ruis die wit is in een voldoende groot frequentie-interval. Deze ruis $w(t)$ heeft een spectrale dichtheid gelijk aan $N_0/2$. Het ontvangen signaal $r(t)$ kan geschreven worden als volgt:

$$r(t) = h_{\text{ch}}s(t) + w(t). \tag{8}$$

C. Ontvanger

Aan de ontvanger wensen we het signaal $r(t)$ te bemonsteren. Daarom wordt het ontvangen signaal eerst gefilterd door een ideaal laagdoorlaatfilter[3]. Vervolgens wordt de uitgang van dit filter bemonsterd aan een debiet $\frac{N_s}{T} = \frac{1}{T_s}$, met $N_s \in \mathbb{N}$. De monsterwaarden aan de ontvanger kunnen we dan noteren als

$$r'_l = r\Big(l\frac{T}{N_s}\Big) = h_{\text{ch}}s\left(l\frac{T}{N_s}\right) + n_l, \tag{9}$$

met

$$\mathbb{E}[n_{l_1}n_{l_2}] = \sigma^2\delta_{l_1-l_2} = \frac{N_0N_s}{2T}\delta_{l_1-l_2}. \tag{10}$$

We wensen nu de verzonden informatie te reconstrueren. Hiervoor voeren we de volgende stappen uit:

- Het bemonsterde signaal wordt gedemoduleerd met de draaggolffrequentie f_0. In ons model veronderstellen we dat de demodulator niet perfect is en een onbekende rotatie θ introduceert. We bekomen

$$r_l = \sqrt{2}r'_l e^{-j(2\pi f_0 l T_s + \theta)}. \tag{11}$$

  Deze monsterwaarden worden vervolgens door een digitaal ontvangerfilter met impuls antwoord $\{p_l\}$ gestuurd dat aangepast is aan de zenderpuls ($p_l = p\,(lT)$). De uitgang van dit filter wordt dan gegeven door

$$y_n = T_s \sum_{l=-\infty}^{\infty} p_l r_{n-l}. \tag{12}$$

[3]Om te voldoen aan het bemonsteringstheorema van Shannon kiezen we een bandbreedte gelijk aan $\frac{N_s}{2T}$

De resulterende sequentie $\{y_l\}$ bevat nog steeds N_s monsterwaarden per symboolinterval. Na decimatie met een factor N_s verkrijgen we een sequentie $\{z_l\}$. De decisievariabele u_k wordt bekomen door z_k te schalen met factor $\frac{e^{j\hat{\theta}}}{\hat{h}_{\mathrm{ch}}}$, i.e.,

$$u_k = \frac{z_k}{\hat{h}_{\mathrm{ch}}} e^{j\hat{\theta}}, \tag{13}$$

hierbij is $\hat{h}_{\mathrm{ch}}$ de geschatte schalingsfactor van het kanaal en $\hat{\theta}$ de geschatte fase van de modulator. In het algemeen zijn de samples z_k een functie zijn van alle verstuurde symbolen $(a_0, a_1 ..., a_{K-1})$. Stel dat de dominante bijdrage in z_k afkomstig is van het k-de symbool a_k, dan worden de bijdragen van alle andere symbolen in z_k aangeduid als *inter-symboolinterferentie*. Afhankelijk van het tijdstip waarop de decimatie van y_n wordt doorgevoerd is en meer of minder inter-symboolinterferentie aanwezig in z_k. Het beste tijdstip om de decimatie door te voeren zal worden bepaald in subsectie V-D. In het ideale geval, is er geen inter-symboolinterferentie en zijn de schattingen $\hat{h}_{\mathrm{ch}}$ en $\hat{\theta}$ perfect, i.e., $\hat{h}_{\mathrm{ch}} = h_{\mathrm{ch}}$ en $\hat{\theta} = \theta$. Aangezien de energie van de zenderpuls genormeerd is ($\int_{-\infty}^{+\infty} |p(t)|^2 dt = 1$), geldt in dat geval

$$u_k = a_k + \tilde{w}_k, \tag{14}$$

waarbij $\tilde{w}_k$ complexwaardige witte Gaussiaanse ruis voorstelt met $\mathbb{E}[\tilde{w}_k(\tilde{w}_k)^*] = \frac{N_0}{|\hat{h}_{\mathrm{ch}}|^2}$. Voor elke u_k zoekt de ontanger dan het dichtstbijzijnde constellatiepunt $\hat{a}_k$. Hieruit vinden we de m corresponderende bits $\hat{\beta}_l$ terug via demapping.

- Vervolgens zal de decoder de geschatte bitsequentie $\left\{\hat{\beta}_l\right\}$ opsplitsen in *ontvangen woorden* $\boldsymbol{\beta}$ van n bits. Bij het ontwerp van de kanaaldecodering gaan we uit van een *binair symmetrisch kanaal (BSC)*. Dat houdt in dat we veronderstellen dat elk informatiebit een even grote kans p heeft om foutief te worden geschat aan de ontvanger en dat de bitfouten bovendien onafhankelijk van elkaar optreden. Voor elk ontvangen woord $\boldsymbol{\beta}$ gaat de decoder na of het een geldig codewoord is (*zuivere foutdetectie*). Belangrijke prestatiematen zijn de *kans op decodeerfalen*

$$p_{f,DET} = \Pr\left[\boldsymbol{\beta} \notin \mathcal{C}\right], \tag{15}$$

met $\mathcal{C}$ de verzameling van geldige codewoorden, en de *kans op een niet-detecteerbare fout* (*missed detection*)

$$p_m = \Pr\left[(\boldsymbol{\beta} \in \mathcal{C}) \cap (\boldsymbol{\beta} \neq \boldsymbol{c})\right], \tag{16}$$

met $\boldsymbol{c}$ het verstuurde codewoord en $\boldsymbol{\beta}$ het corresponderende ontvangen woord. Indien $\boldsymbol{\beta}$ geen geldig codewoord is, kan de decoder een poging doen om de opgetreden bitfouten te corrigeren (*volledige foutcorrectie*). In het geval van foutcorrectie bepaalt de kanaaldecoder hoeveel bits (+ op welke posities) er minimaal moeten worden gecorrigeerd in $\boldsymbol{\beta}$ om een geldig codewoord te verkrijgen. Vervolgens wordt deze correctie doorgevoerd. Dit resulteert in een *geschat codewoord* $\hat{\boldsymbol{c}}$. De prestatie van een kanaalcode met volledige decodering wordt gemeten in termen van de *decodeerfoutwaarschijnlijkheid*

$$p_{e,DEC} = \Pr\left[\hat{\boldsymbol{c}} \neq \boldsymbol{c}\right], \tag{17}$$

met $\boldsymbol{c}$ het verstuurde codewoord en $\hat{\boldsymbol{c}}$ het corresponderende geschatte codewoord. De prestatie van een lineaire blokcode hangt in de eerste plaats af van de mate waarin codewoorden van elkaar verschillen. Een belangrijke parameter is dan ook de *minimale Hammingafstand* $d_{\min}$ van de code. Dit is het minimaal aantal bitfouten dat nodig is om één geldig codewoord om te zetten in een ander geldig codewoord. Voor lineaire blokcodes is d_{min} gelijk aan het minimum aantal 1-bits in een van nul verschillend codewoord. Het *gegarandeerd foutcorrigerend vermogen* van een code is t (bitfouten per codewoord), met

$$t = \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor. \tag{18}$$

Het *gegarandeerd foutdetecterend vermogen* van een code is $\mathcal{L}$ (bitsfouten per codewoord), met

$$\mathcal{L} = d_{\min} - 1. \tag{19}$$

Foutdetectie en foutcorrectie van lineaire blokcodes kan efficiënt worden geimplementeerd met behulp van een *syndroomtabel*. Het *syndroom* van een ontvangen woord $\boldsymbol{\beta}$ is een bitsequentie $\boldsymbol{s}$ van $n-k$ bits. Het syndroom van een n-bit sequentie $\boldsymbol{\beta}$ wordt berekend als

$$\boldsymbol{s} = \boldsymbol{\beta}\boldsymbol{H}^T, \tag{20}$$

met $\boldsymbol{H}$ een *checkmatrix* van de code (met als eigenschap dat $\boldsymbol{G}\boldsymbol{H}^T = \boldsymbol{0}$). Bijgevolg is het syndroom $\boldsymbol{s} = \boldsymbol{\beta}\boldsymbol{H}^T$ een nulwoord als en slechts als $\boldsymbol{\beta}$ een codewoord is. Meer details over het verband tussen de checkmatrix $\boldsymbol{H}$ en de generatormatrix $\boldsymbol{G}$ van een code, het gebruik van een *syndroomtabel* voor foutcorrectie en bij het bepalen van $p_{f,DET}$, p_m en $p_{e,DEC}$ zijn te vinden in **Appendix B**. In subsectie V-C onderzoeken we de prestaties van verschillende lineaire (14,10) blokcodes en gaan we na of we de foutprobabiliteit verder kunnen verlagen met behulp van *retransmissieprotocols*, waarbij de decoder via een NACK bericht een retransmissie kan aanvragen. De decoder doet aan zuivere foutdetectie en genereert een NACK bericht als hij een fout detecteert, waarna een retransmissie wordt aangevraagd. We veronderstellen dat het aantal retransmissies beperkt is tot $T_{\max}$. Wanneer het maximaal aantal transmissies ($T_{\max}+1$) bereikt wordt, heeft de decoder verschillende opties. Indien de decoder na $T_{\max}$ retransmissies zuivere foutdetectie toepast is de uiteindelijke decodeerfoutwaarschijnlijkheid gegeven door

$$p_e^{ARQ} = p_m + p_{f,DET}p_m + \left(p_{f,DET}\right)^2 p_m + ... + \left(p_{f,DET}\right)^{T_{\max}} p_m + \left(p_{f,DET}\right)^{T_{\max}+1} \left(p_m + p_{f,DET}\right). \tag{21}$$

- De gedecodeerde codewoorden worden door de ontvanger afgebeeld op de corresponderende informatiewoorden, waarna de continue informatiebitsequentie die ontstaat wordt gedecomprimeerd (brondecodering). Bij het ontwerp van de broncodering gaat men er van uit dat de continue informatiebitsequentie foutloos kan worden ontvangen. De broncodering wordt bovendien zo ontworpen dat deze lange informatiebitsequentie slechts op één manier decodeerbaar is.
- Uiteindelijk zal de ontvanger op basis van de bronsymbolen aan de uitgang van de brondecoder de monsterwaarden van het geluidsfragment reconstrueren. Telkens een ontvangen bronsymbool gelijk is aan s_i, genereert de ontvanger de corresponderende monsterwaarde als q_i ($i = 1, 2, ..., M$). Bij het ontwerp van kwantisatoren gaat men er van uit dat de bronsymbolen foutloos kunnen worden ontvangen. De overgang van continue monsterwaarden naar discrete reconstructieniveaus gaat echter onvermijdelijk gepaard met een verlies aan informatie. Een veelgebruikte maat voor dit informatieverlies is de gemiddelde kwadratische distorsie (GKD), die gegeven is door

$$GKD = \sigma_e^2 = \sum_{i=1}^{M} \int_{r_{i-1}}^{r_i} (q_i - u)^2 f_U(u)\,\mathrm{d}u, \tag{22}$$

met $r_0 < r_1 < ... < r_M$ de kwantisiatiedrempels en $q_1 < q_2 < ... < q_M$ de reconstructieniveaus van de gebruikte kwantisator, en $f_U(u)$ de w.d.f. van de bron. In Matlab en Python is de w.d.f $f_U(u)$ waar nodig beschikbaar als een anonieme functie.

V. Opgave

A. *Kwantisatie (*Kwantisatie.m/kwantisatie.py)

Omdat we het originele audiofragment ('input.wav') willen reduceren in grootte, zullen we het kwantiseren met een kwantisator die slechts 64 reconstructieniveaus ($M = 2^x = 2^6$) bezit. Dit wensen we uiteraard te doen met zo laag mogelijke GKD (zie (22)) .

- Maak een figuur van de distributie $f_U(u)$. Plot hierbij tevens een genormaliseerd histogram van de originele monsterwaarden om na te gaan of de gegeven anonieme functie inderdaad correct is. Doe dit in de functie *plot_distributie()*.

Beschouw nu eerst een uniforme kwantisator (*bepaal_optimale_lineaire_kwantisator()*) bepaald door symmetriepunt $x_0 = 0$ en stapgrootte Δ. In dat geval zijn de kwantisiatiedrempels zijn gegeven door $r_i = x_0 + \frac{(2i-M)\Delta}{2}$ voor $i = 1, 2, ..., M-1$ en de reconstructieniveaus door $q_i = x_0 + \left(i - \frac{M+1}{2}\right)\Delta$ voor $i = 1, 2, ..., M$. De afstand tussen de reconstructieniveaus is constant en gelijk aan Δ en het zelfde geldt voor de kwantisatiedrempels. Bovendien liggen in het *granulaire gebied* van de kwantisator $\left[x_0 - M\frac{\Delta}{2}, x_0 + M\frac{\Delta}{2}\right]$ de reconstructieniveaus precies in het midden van de kwantisatieintervallen. Zoals beschreven in de cursus, kunnen we bij een uniforme kwantisator de GKD opsplitsen in een *granulaire bijdrage* $\sigma^2_{\mathrm{e,gr}}$ en een *overlaadbijdrage* $\sigma^2_{\mathrm{e,ol}}$:

$$\sigma_e^2 = \sigma^2_{\mathrm{e,gr}} + \sigma^2_{\mathrm{e,ol}}, \tag{23}$$

met

$$\sigma^2_{\mathrm{e,gr}} = \sum_{i=1}^{M} \int_{q_i - \frac{\Delta}{2}}^{q_i + \frac{\Delta}{2}} (q_i - u)^2 f_U(u)\, \mathrm{d}u \tag{24}$$

en

$$\sigma^2_{\mathrm{e,ol}} = \int_{-\infty}^{q_1 - \frac{\Delta}{2}} (q_1 - u)^2 f_U(u)\, du + \int_{q_M + \frac{\Delta}{2}}^{\infty} (q_M - u)^2 f_U(u)\, \mathrm{d}u. \tag{25}$$

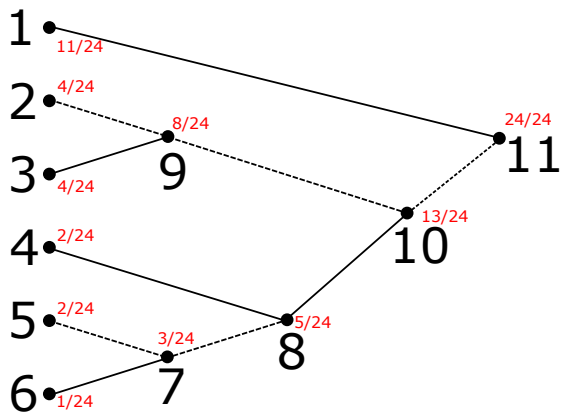
- Maak een figuur die het verloop van σ_e^2, $\sigma^2_{\mathrm{e,gr}}$ en $\sigma^2_{\mathrm{e,ol}}$ weergeeft in functie van de stapgrootte Δ. Verklaar wat je ziet en bepaal, op basis van de figuur, de optimale stapgrootte Δ_{opt} en de bijhorende minimale totale GKD. Bereken verder ook de waarschijnlijkheden p_k horend bij elke bronsymboolwaarde en geef de entropie H (2) van de bronsymboolsequentie aan de uitgang van de kwantisator.
- De *signaal-kwantisatieruis verhouding* (SQR) is gedefinieerd als de verhouding van de variantie van de bronuitgang en de GKD. De optimale SQR wordt bijgevolg dus gerealiseerd door de kwantisator met minimale GKD. Maak een figuur van de optimale SQR in dB in functie van $\alpha = [2, ..., 8]$, waarbij $M = 2^\alpha$. Bespreek de figuur en verklaar de winst van ongeveer 6dB bij de hogere waarden van α.
- Plot nu opnieuw de distributie $f_U(u)$ waarbij de bekomen kwantisatiedrempels en reconstructieniveaus duidelijk zijn aangegeven.

Bij de optimale kwantisator (minimale GKD) liggen de kwantisatiedrempels en reconstructieniveaus vaak niet op vaste afstand van elkaar. Bijgevolg is deze optimale kwantisator niet uniform. De Lloyd-Max voorwaarden zijn twee nodige voorwaarden opdat een kwantisator optimaal kan zijn. Een algoritme om een kwantisator te ontwerpen die voldoet aan de Lloyd-Max voorwaarden kan je vinden in **Appendix C**. Implementeer zelf dit algoritme in de functie *Llyod_Max_algoritme()*.

- Maak een figuur van de GKD in functie van het aantal iteraties en bespreek. Is de resulterende GKD kleiner dan de GKD bekomen met de uniforme kwantisator? Plot tevens de entropie van het gekwantiseerde signaal in functie van het aantal iteraties. Verklaar wat je ziet. Voeg de entropie van een uniform verdeelde discrete toevalsgrootheid met een alfabet van M letters toe als referentie. Hoeveel dB winst maakt de Lloyd-Max kwantisator tenslotte in termen van SQR ten opzichte van de uniforme kwantisator?
- Plot nu opnieuw de distributie $f_U(u)$ waarbij de bekomen kwantisatiedrempels en reconstructieniveaus van de Lloyd-Max kwantisator duidelijk zijn aangegeven. Komt dit overeen met de verwachtingen?

Een suboptimale techniek om niet-uniforme kwantisators te ontwerpen is compansie. Hierbij passen we uniforme kwantisatie toe op een vervormde bronuitgang $g(u)$, waarna de inverse bewerking $g^{-1}(.)$ wordt uitgevoerd op de bekomen kwantisatiedrempels en reconstructieniveaus. In dit project gebruiken we een compansiekarakteristiek $g\,(u)$ die ervoor zorgt dat de vervormde bronuitgang uniform verdeeld is in [-0.5,0.5]:

$$g(u) = \int_{-\infty}^{u} f_U(x)\, \mathrm{d}x - 0.5. \tag{26}$$



0	0
0	0
0	0
0	0
0	0
0	0
5	6
7	4
2	3
9	8
10	1

Figuur 1. Voorbeeld codeboom met genummerde knopen (links) en de bijhorende variable *boom_info* (rechts).

Implementeer deze kwantisator in *bepaal_compansie_kwantisator()* en maak hierbij gebruik van de functie *Kwantisatie.inverse()* die numeriek de inverse $g^{-1}(y)$ berekent voor een specifieke y.

- Maak een figuur van $g(u)$ in functie van u. Verklaar het verloop van deze functie.
- Geef dezelfde karakteristieken van de kwantisator als voor de uniforme kwantisator (GKD, SQR, entropie).
- Plot nu opnieuw de distributie $f_U(u)$ waarbij de bekomen kwantisatiedrempels en reconstructieniveaus van de compansiekwantisator duidelijk zijn aangegeven. Zijn de prestaties van deze kwantisator beter of slechter dan deze van de uniforme kwantisator? Is dit altijd zo?

Implementeer tenslotte de functie *kwantiseer()*, die de eigenlijke kwantisatie implementeert, en beluister het oorspronkelijke fragment en de gekwantiseerde fragmenten van de verschillende kwantisators. Gebruik hiervoor de ingebouwde functie *sound()* (Python: *Kwantisatie.save_and_play_music()).* Welke verschillen bemerk je? Sla de gekwantiseerde fragmenten ook op met behulp van de ingebouwde functie *audiowrite()* (Python: *Kwantisatie.save_and_play_music())* met als naam *'uniform.wav', 'LM.wav'* en *'compansie.wav'*. Deze files dienen tevens ingediend te worden.

*B. Broncodering (*Broncodering.m/broncodering.py*)*

De volgende stap aan de zenderzijde is het omzetten van het gekwantiseerde signaal naar een bitsequentie door middel van broncodering. We onderzoeken de prestatie van Huffmancodering en van vaste-lengte codering. Gebruik hiervoor telkens de bronsymboolsequentie aan de uitgang van de Lloyd-Max kwantisator die je bepaald hebt hierboven en vul de klasse *Broncodering* aan waar nodig.

- Implementeer eerst de functie *maak_codetabel_Huffman()* die op basis van een set macrosymbool-waarschijnlijkheden met behulp van het Huffmanalgoritme de codewoorden berekent (zie ook **Appendix A**). Verder moet ook de $(2M-1)\times 2$ matrix *boom_info* geconstrueerd worden. Deze matrix bevat de informatie over de opbouw van de *codeboom* en is van grote hulp bij het decoderen van de Huffmancode. Om deze variabele op te bouwen worden eerst alle oorspronkelijke knopen genummerd van 1 tot en met M en worden de eerste M rijen van *boom_info* gelijk gesteld aan nulrijen. Wanneer een nieuwe knoop wordt aangemaakt in het Huffmanalgoritme, wordt telkens een nieuwe rij toegevoegd aan *boom_info,* waarin het linkerelement en rechterelement gelijk zijn aan het nummer van de knoop dat respectievelijk correspondeert met de op één na kleinste en kleinste waarschijnlijkheid. Een voorbeeld hiervan is te vinden in Figuur 1.
- Vervolgens passen we scalaire Huffmancodering toe. Dit betekent concreet dat zowel het macrosymboolalfabet gelijk is aan bronsymboolalfabet (en dit betaat op zijn beurt uit de verschillende mogelijke uitgangswaarden van de kwantisator). Wat is het gemiddeld aantal codebits per monsterwaarde (let op!(3) geeft het resultaat per macrosymbool.) ? Zijn de entropiegrenzen (1) voldaan?

- We onderzoeken verder ook de prestaties van de Huffmanvectorcodering met dimensie 2. Hiervoor nemen we twee bronsymbolen (gekwantiseerde samples) samen tot één macrosymbool.
  - Hoeveel letters bevat het bronsymboolalfabet nu? Hoeveel verschillende macrosymbolen bevat het macrosymboolalfabet? Noem dit M_2. Deze macrosymbolen zullen we eenvoudiger voorstellen door de gehele getallen van 0 tot $M_2 - 1$ te gebruiken. Hierbij komt 0 overeen met de sequentie q_1q_1, 1 met de sequentie q_1q_2, ... , en $M_2 - 1$ met de sequentie q_Mq_M.
  - Voordat we de Huffmancodering kunnen toepassen moeten we eerst de codetabel opstellen, waarvoor de waarschijnlijkheden van de verschillende macrosymbolen gekend moeten zijn. Deze kunnen echter niet zomaar bepaald worden door de waarschijnlijkheden van de bronsymbolen met elkaar te vermenigvuldigen aangezien deze niet onafhankelijk zijn van elkaar. Als alternatief tellen we hoeveel keer elk macrosymbool voorkomt in de te coderen sequentie om zo hun relatieve frequenties te bepalen. Deze relatieve frequenties zullen worden gebruikt als macrosymboolwaarschijnlijkheden. Verder moet ook elke sequentie van 2 bronsymbolen omgezet worden naar het corresponderende macrosymbool. Implementeer dit alles in de functie *scalair_naar_vector*(). Wat is de entropie van de macrosymboolwaarschijnlijkheden?
  - Maak nu de codetabel met behulp van de functie *maak_codetabel_Huffman()*. Wat is het resulterend gemiddeld aantal codebits per bronsymbool? Is er voldaan aan de entropiegrenzen? Verklaar het verschil met scalaire broncodering.
  - De decoder van de Huffmancodering zal de geëncodeerde bits omzetten naar macrosymbolen die op hun beurt opnieuw moeten omgezet worden naar een sequentie van bronsymbolen. Implementeer dit laatste in de functie *vector_naar_scalair()*.
- Vul nu de functies *vaste_lengte_encodeer()* en *vaste_lengte_decodeer()* aan die respectievelijk de encodering en decodering met een vaste-lengte code implementeert. Zorg ervoor dat q_1 overeenkomt met allemaal 0-bits en q_M met allemaal 1-bits. Hoeveel bits zijn er minimaal nodig per bronsymbool? Is dit een groot verschil met de de Huffmancodering? Wat kan een voordeel zijn van de vaste-lengte code? Is het hier voordelig om vaste-lengte vectorcodering te beschouwen? Is dit altijd zo?

*C. Kanaalcodering (*Kanaalcodering.m/kanaalcodering.py)

Omdat we verwachten dat sommige bits foutief ontvangen zullen worden, voegen we nu bijkomende redundante bits toe om deze eventuele fouten te kunnen verbeteren aan de ontvanger. In dit deel van het project werk je best met een zelf gegeneerde random bitsequentie. In het bestand *Kanaalcodering.m* vind je de verschillende functies die je moet implementeren. Als praktische toepassing zullen we in deze sectie veronderstellen dat we een kanaal hebben dat gemodelleerd is als een BSC met p=0.05. Het criterium bij het ontwerp van de code en het retransmissieprotocol is dat de kans op een decodeerfout in een blok van 10 informatiebits kleiner moet zijn dan 0.001.

Eerst onderzoeken we de prestaties van de (14,10) lineaire blokcode met checkmatrix

$$\boldsymbol{H} = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}.$$

- Bepaal voor deze code
  - een generatormatrix $\boldsymbol{G}$,
  - de minimale Hammingafstand,
  - het foutdetecterend en foutcorrigerend vermogen en
  - een syndroomtabel.

Implementeer nu de encoder en de decoder van deze code in respectievelijk *encodeer_uitwendig()* en *decodeer_uitwendig()*. De encodering gebeurt aan de hand van de generatormatrix ($\boldsymbol{c} = \boldsymbol{bG}$) en de

decodering via het syndroom ($\boldsymbol{s} = \boldsymbol{r}\boldsymbol{H}^T$). In de decoder is het de bedoeling dat je zowel volledige decodering als zuivere foutdetectie implementeert. Hierbij stelt de outputvector *bitdec* de gedecodeerde bits na volledige decodering voor en is het $i$de element van vector *bool_fout* gelijk aan 1 als er in het $i$de codewoord een fout is gedetecteerd bij zuivere foutdetectie en gelijk aan 0 als er geen fout is gedetecteerd. In het laatste geval, kan het corresponderende codewoord dan gevonden worden in de sequentie *bitdec*.

- Bepaal in het geval van volledige decodering de kans op een decodeerfout, p_e. Geef zowel de exacte formule als een benadering voor $p \ll 1$. Voer tevens simulaties uit om de kans op een decodeerfout experimenteel te bepalen. Maak een figuur waarin de gesimuleerde resultaten en analytische uitdrukking met elkaar vergeleken worden als functie van p, waarbij p gaat van 0.001 tot 0.5. Gebruik hiervoor een dubbellogaritmische schaal.
- Is deze code krachtig genoeg om het ontwerpcriterium te halen bij $p = 0.05$? Voor welke waarden van p is dit criterium wel voldaan?
- Geef een analytische uitdrukking (exact + benadering) voor de kans op decodeerfalen, p_f, en voor de kans op een niet-detecteerbare fout, p_m, in het geval van zuivere foutdetectie. Hoe groot zijn beide kansen voor $p = 0.05$?

Om de kans op een decodeerfout te verminderen kan je in de praktijk gebruik maken van retransmissieprotocols. In een eerste retransmissieprotocol gebruiken we de code van hierboven en veronderstellen we dat wanneer het maximaal aantal transmissies bereikt wordt, de decoder volledige decodering toepast (Merk op dat dit een andere veronderstelling is dan in (21).).

- Leid de formule af voor de kans dat er een decodeerfout optreedt na maximum $T_{\max}$ retransmissies, p_e^{ARQ}, als functie van $p_{e,DEC}$ (17), $p_{f,DET}$ (15) en p_m (16).
- Maak aan de hand van simulaties een figuur die de kans p_e^{ARQ1} weergeeft als functie van $T_{\max}$, waarbij $T_{\max}$ gaat van 0 tot en met 15 bij $p = 0.05$. Vergelijk met het analytische resultaat. Maak hierbij ook een figuur van het gemiddeld informatiedebiet in functie van $T_{\max}$ (het totaal aantal informatiebits gedeeld door het totaal aantal verzonden bits). Bespreek de bekomen figuren. Waardoor zijn de prestaties van dit retransmissieprotocol beperkt? Is het mogelijk om met deze methode te voldoen aan het ontwerpcriterium? Indien ja, hoeveel retransmissies moeten er minimaal toegelaten worden?

Tip: Bij de simulaties van de retransmissieprotocols is het niet aan te raden om dit codewoord per codewoord te doen zoals in de praktijk, maar is het efficiënter om transmissie per transmissie te simuleren. Eerst simuleer je een grote aantal informatiewoorden in de eerste transmissie. Aan de hand van de variable *bool_fout* kan je dan nagaan welke codewoorden er een retransmissie nodig hebben en zo maak je een nieuwe bitstring die moet verzonden worden in de volgende transmissie.

In de praktijk wordt een retransmissieprotocol vaak geïmplementeerd aan de hand van 2 afzonderlijke codes: een inwendige code voor foutdetectie en een uitwendige code voor foutcorrectie[4]. Hierbij wordt het informatiewoord eerst gecodeerd met de inwendige code en vervolgens wordt het verkregen codewoord nogmaals gecodeerd met de uitwendige code. Na transmissie over het kanaal wordt de ontvangen bitsequentie eerst volledig gedecodeerd door de uitwendige code, waarna de decoder van de inwendige code zuivere foutdetectie uitvoert. Geeft deze laatste een NACK, dan wordt een retransmissie voor dit woord aangevraagd. Als uitwendige code zullen we bovenstaande (14,10) code gebruiken.

- Als inwendige code zullen we een *cyclic redundancy check* (CRC) code gebruiken, aangezien deze typisch een zeer lage kans op een niet-detecteerbare fout heeft. CRC-codes werken met binaire veeltermen. Een binaire veelterm van graad $q-1$ of lager $a_1x^{q-1} + a_2x^{q-2} + ... + a_{q-1}x + a_q$ correspondeert met de q-bit sequentie $(a_1a_2...a_{q-1}a_q)$. Met een informatiewoord van lengte k komt een informatieveelterm van graad $k-1$ overeen. Het verstuurde n-bit codewoord is dan gelijk aan de som van de verschoven informatieveelterm $b(x)x^n$ en de restveelterm bij deling van $b(x)x^n$ door een CRC-veelterm $g(x)$ van graad $n-k$. Bij zuivere foutdetectie aan de ontvanger wordt nagegaan of de veeltermvoorstelling van het ontvangen woord een veelvoud is van $g(x)$ (rest nul bij deling!). Toon

[4]We volgen hier de notatie van de syllabus, terwijl in de opnames de benamingen inwendig en uitwendig werden omgewisseld (omdat dit de meest gangbare conventie is).

aan dat elke CRC code een lineaire code is. Implementeer voor deze code de encoder en decoder voor zuivere foutdetectie in respectievelijk *encodeer_inwendig()* en *decodeer_inwendig()*.

Tip: Om in Matlab veeltermen met elkaar te vermenigvuldigen en te delen kan je gebruik maken van respectievelijk 'conv()' en 'deconv()'. Je mag er verder vanuit gaan dat in dit project de dimensie van de code (relatief) klein zal zijn. Hierdoor is het efficiënter om eerst de codetabel op te stellen en daarna gebruik te maken van deze codetabel voor zowel de codering als de decodering.

- De eerste inwendige code die we gebruiken is de CRC-5 code met $g(x) = (x+1)(x^4+x+1) = x^5+x^4+x^2+1$. Bepaal voor deze code
  - de dimensie k,
  - het gegarandeerd burstfoutdetecterend vermogen,
  - de minimale Hammingafstand, foutcorrigerend en foutdetecterend vermogen,
  - de kans op een niet-detecteerbare fout (in het bijzonder bij $p = 0.05$).
  - Simuleer voor dit retransmissieprotocol de kans op een decodeerfout, p_e^{ARQ2a}, bij $p = 0.05$. Merk op dat, in tegenstelling tot het vorige retransmissieprotocol, voor de laatste retransmissie de inwendige code zuivere foutdetectie doet (dus (21) is van toepassing). Alle ontvangen woorden die geen codewoorden zijn, worden als foutief beschouwd. Maak een figuur van p_e^{ARQ2a} als functie van $T_{\max}$ (van 0 tot en met 6) met daarop de simulatieresultaten samen met de benadering (zie cursus) $p_e^{ARQ} \approx (p_{out:e})^{T_{\max}+1}$, waarbij $p_{out:e}$ de kans op een decodeerfout van de uitwendige code voorstelt. Geef ook de figuur van het informatiedebiet als functie van $T_{\max}$. Bespreek en verklaar beide figuren. Is het mogelijk om met dit retransmissieprotocol te voldoen aan het ontwerpcriterium? Zo ja, hoeveel retransmissies zijn er minimaal nodig?
- Een tweede CRC code die we onderzoeken, is een CRC-8 code met $g(x) = (x+1)(x^7+x^3+1) = x^8+x^7+x^4+x^3+x+1$. Bepaal voor deze code opnieuw
  - de dimensie k,
  - het gegarandeerd burstfoutdetecterend vermogen,
  - de minimale Hammingafstand, foutcorrigerend en foutdetecterend vermogen,
  - de kans op een niet-detecteerbare fout (in het bijzonder bij $p = 0.05$).
  - Simuleer ook voor dit retransmissieprotocol de kans op een decodeerfout, p_e^{ARQ2b}, bij $p = 0.05$. Maak opnieuw een figuur van p_e^{ARQ2b} als functie van $T_{\max}$ (van 0 tot en met 6) met daarop de simulatieresultaten en de benadering. Toon ook de plot van het informatiedebiet als functie van $T_{\max}$. Wat is het verschil met de eerder besproken CRC code? Is het mogelijk om met dit retransmissieprotocol te voldoen aan het ontwerpcriterium? Zo ja, hoeveel retransmissies zijn er minimaal nodig?

*D. Modulatie en detectie (*ModDet.m/moddet.py*)*

Als laatste deel van dit project modelleren we hier de transmissie over een fysisch kanaal. Ook dit deel van het project wordt het best uitgevoerd met een random gegenereerde (ongecodeerde) bitsequentie.

- Eerst implementeren we de mapper die een bitsequentie omzet in een sequentie van data symbolen. De mapper dient de volgende constellaties te ondersteunen: BPSK, 4QAM en 4PAM. Implementeer hiervoor de functie *mapper()* en maak hierbij gebruik van Gray-Mapping. Implementeer tevens de demapper (*demapper()*) die de inverse operatie uitvoert.
- In een volgende stap wordt de bekomen sequentie van data symbolen op zenderpulsen geplaatst (zie vgl (5)) en wordt het basisbandsignaal op een draaggolf geplaatst met frequentie $f_0 = 2MHz$ (zie vgl (7)). In een praktisch systeem is het resultaat van deze operaties een continue signaal. In een Matlab/Python-simulatie moet alles echter voorgesteld worden als een sequentie van samples. Daarom bemonsteren we $s(t)$ aan een frequentie van $\frac{N_s}{T}$, waarbij N_s het aantal samples per symbool interval voorstelt. Als gevolg wordt het continue signaal $s(t)$ in Matlab/Python voorgesteld als sequentie van samples op tijdstippen $\frac{lN_s}{T}$ met $l \in \mathbb{Z}$.

  - In dit project zullen we uitsluitend gebruik maken van de square-root raised cosine pulse waarbij de symboolinterval $T = 1\mu s$ en de roll-off factor α tussen 0 en 1 ligt. Waaraan moet $N_s$voldoen opdat aan het bemonsteringstheorema van Nyquist/Shannon is voldaan? Kies voor de rest van het project de minimale N_s die hieraan voldoet.
  - In theorie heeft een square-root raised cosine pulse een oneindige duur, maar aangezien deze pulse uitsterft in de tijd, kunnen we $p(t)$ afknotten tot $2L_F$ symboolperioden, i.e., $p(t) = 0,\ |t| > L_F T$. Voor dit project stellen we L_F gelijk aan 10, wat resulteert in $20N_s+1$ monsterwaarden. Het effect van het afknotten van de pulse is het best te observeren in het frequentiedomein. Maak daarom voor $\alpha = 0.05$, $\alpha = 0.5$ en $\alpha = 0.95$ een figuur van $|P(f)|$ van de originele en afgeknotte pulse en vergelijk. Wat is het effect van het afknotten voor de verschillende waarden van α? Baseer je bij het construeren van $P(f)$ op sectie 2.5 in het extra document over 'ComputerSimulations'. Denk tevens goed na over de juiste schaal voor de figuren.
  - Implementeer de volledige modulatie in *moduleer()*.
- Het banddoorlaatsignaal $s(t)$ wordt verstuurd over een kanaal (schalingsfactor h_{ch} en reële witte Gaussiaanse ruis met spectrale dichtheid $\frac{N_0}{2}$). De samples van het ontvangen signaal **r** worden gegeven door

$$r\Big(l\frac{T}{N_s}\Big) = h_{\text{ch}} s\Big(l\frac{T}{N_s}\Big) + n_l \tag{27}$$

  De variantie van de ruis n_l wordt gegeven door (10). Implementeer nu de functie *kanaal()*. Stel h_{ch} in het volledige project gelijk aan 1.
- Vervolgens worden de ontvangen monsterwaarden gedemoduleerd met $f_0 = 2\,\text{MHz}$ ($\theta = \frac{\pi}{16}$) en aan een digitaal filter, gematched aan de zenderdpuls, aangelegd. Voor een square-root raised cosine pulse zijn zender-en ontvanger puls/filter identiek. De uitgang van dit matched filter wordt gegeven door uitdrukking (12) maar waarbij de sommatie-index l loop van $-L_F N_s$ tot $L_F N_s$. Deze operatie kan gemakkelijk geïmplementeerd worden in Matlab met behulp van de functie *conv()*:

$$\boldsymbol{y} = T_s conv(\boldsymbol{r}, \boldsymbol{p}) \tag{28}$$

  We beperken $p(t)$ opnieuw tot 20 symboolperioden, dus $L_F = 10$. Al deze stappen worden geïmplementeerd in de functie *demoduleer*(). Verwacht je dat $\boldsymbol{y}$ een reëel of complex signaal is voor de BPSK constellatie? Verklaar.
- Maak een oogdiagram van twee symboolperioden voor $\alpha = 0.05$, $\alpha = 0.5$ en $\alpha = 0.95$ op basis van het reële deel van de ontvangen sequentie $\boldsymbol{y}$ (zonder ruis, voor de BPSK constellatie). Dit is, plot de sequentie $\{y_n\}$ voor alle mogelijke symboolsequenties $\{a_k\}$ over twee symboolperioden. Maak daarvoor gebruik van de standaard Matlab-functie *eyediagram()* en laat het overgangsverschijnsel aan het begin en einde van de sequentie weg[5]. Bespreek het verband tussen de inter-symboolinterferentie en het tijdstip waarop de decimatie wordt uitgevoerd en bepaal hieruit het optimale tijdstip voor de decimatie. Is de verticale oogopening gelijk voor alle α? Is dit in overeenkomst met de theorie? Zo niet, waarom niet? Maak nu ook een oogdiagram voor de 4-PAM constellatie met $\alpha = 0.5$. Wat is het verschil met BPSK? Voer nu de decimatie uit op de correcte tijdstippen in de functie *decimatie()*. Stel voor de rest van dit project $\alpha = 0.5$.
- Implementeer nu de functie *maak_decisie_variable*() die het gedecimeerde signaal schaalt met de factor $\hat{h}_{\text{ch}}$ en de fase compenseert met een rotatie over hoek van -$\hat{\theta}$. Veronderstel voorlopig dat het kanaal en demodulator perfect gekend is ($\hat{h}_{\text{ch}} = h_{\text{ch}}$ en $\hat{\theta} = \theta$). Maak een scatterdiagram voor de 4QAM constellatie van het signaal $\{u_k\}$ en onderzoek de invloed van E_b/N_0. Maak hiervoor gebruik van de ingebouwde functie *scatter*().
- Op basis van het signaal u_k kan de ontvanger voor elke waarde het dichtstbijzijnde constellatiepunt zoeken. Dit noemt men harde decisie. Implementeer dit in *decisie()* en verifieer dat voor $N_0 = 0$ de juiste constellatiepunten worden teruggevonden.

[5] zie ook: sectie 11.1 in extra document 'ComputerSimulations'

scenario	1	2	3	4	5	6
gemiddeld aantal bits per originele sample						
gemiddeld kwadratische afwijking (GKA)						

Tabel III
RESULTATEN

– Bepaal nu de BER voor alle drie de constellaties (BPSK, 4QAM en 4PAM) aan de hand van simulaties voor verschillende waarden van E_b/N_0 en maak een figuur[6]. Beschouw enkel het interval van E_b/N_0 waarvoor $10^{-1} > BER > 10^{-4}$ (ongeveer). Vergelijk het bekomen resultaat met de curves uit de cursus en waar mogelijk met de analytische uitdrukking.

Tot nu toe hebben we verondersteld dat de ontvanger gebruik maakt van de correcte parameters (h_{ch}, θ) om de decisie uit te voeren. In de praktijk zijn deze parameters echter niet gekend en zullen die moeten geschat worden door de ontvanger wat kan leiden tot kanaalschattingsfouten. In dit project zullen we nagaan wat het gevolg is van de amplitudeschattingsfout en faseschattingsfout.

- Eerst bestuderen we de invloed van een amplitudeschattingsfout bij zowel de BPSK als 4PAM constellatie. Maak hiervoor een scatterdiagram in afwezigheid van ruis voor beide constellaties bij $\epsilon = 0.1$ met $\epsilon = \frac{\hat{h}_{\text{ch}} - h_{\text{ch}}}{h_{\text{ch}}}$. Wat bemerk je? Maak opnieuw voor beide constellaties een figuur met daarop de BER-curves voor volgende waarden van $\epsilon = 0, 0.1, 0.2$. Verklaar het resultaat.
- Vervolgens gaan we na wat het effect is van een faseschattingsfout als we de 4QAM constellatie gebruiken. Maak ook hier een scatterdiagram in afwezigheid van ruis bij $\phi = \frac{\pi}{16}$ met $\phi = \hat{\theta} - \theta$. Wat bemerk je? Maak vervolgens opnieuw een figuur met de BER-curves voor de volgende waarden van $\phi = 0, \frac{\pi}{16}, \frac{\pi}{8}, \frac{\pi}{4}$. Verklaar het resultaat.

E. Volledig systeem

In dit laatste deel zullen het volledige systeem toepassen op het audiofragment en nagaan hoe verschillende configuraties zowel de kwaliteit als de complexiteit (in termen van het aantal verzonden bits per sample) beïnvloeden. Bepaal hiervoor eerst de E_b/N_0-waarde voor de BPSK constellatie waarvoor geldt dat BER≈ 0.05. Verstuur het audiofragment na kwantisatie met de Lloyd-Max kwantisator over het kanaal op een draaggolf met frequentie $f_0 = 2\,\text{MHz}$ bij deze signaal-ruis verhouding. Doe dit voor volgende scenario's:

1) Vaste-lengte codering, geen kanaalcodering.
2) Vaste-lengte codering, uitwendige code.
3) Scalaire Huffmancodering, uitwendige code.
4) Scalaire Huffmancodering, retransmissieprotocol 1.
5) Scalaire Huffmancodering, retransmissieprotocol 2 met als inwendige code de CRC-8 code.
6) Vector Huffmancodering, retransmissieprotocol 2 met als inwendige code de CRC-8 code.

Bepaal voor de 6 gevallen telkens het gemiddeld aantal bits dat verstuurd moet worden per sample van het origineel audiofragment en de gemiddelde kwadratische afwijking (GKA) tussen het gereconstrueerde en het originele audiofragment. Vul daarvoor Tabel 3 aan. Bespreek tevens het auditieve verschil tussen de ontvangen audiofragmenten van de verschillende scenario's. Sla deze audiofragmenten ook op (naam: *ConfiguratieX.wav*) en dien deze mee in.

APPENDIX A: HUFFMANCODERING EN-DECODERING

Huffmancodering en -decodering maakt gebruik van een *binaire codeboom* (zie Figuur 1). Het opstellen van deze codeboom gaat als volgt:

[6]zie ook: sectie 7 in extra document over 'ComputerSimulations'. Hierin worden twee methodes besproken om tot betrouwbare resultaten te komen.

- Beschouw een lijst van de macrosymboolwaarschijnlijkheden.
- Schrijf de macrosymbolen onder elkaar. Schrijf de bijhorende macrosymboolwaarschijnlijkheden er naast.
- Associeer aan elk macrosymbool een *knoop*. Dit woorden de *bladeren* van de boom.
- Herhaal tot alle bladeren verbonden zijn met de boom:
  - Selecteer de twee knopen met de laagste waarschijnlijkheden in de lijst met waarschijnlijkheden.
  - Verbind deze twee knopen met een nieuwe knoop. [Teken twee *takken* en één nieuwe knoop, rechts van de bestaande knopen.]
  - Associeer aan de nieuwe knoop de som van de waarschijnlijkheden van de knopen waarmee je hem hebt verbonden.
  - Verwijder in de lijst met waarschijnlijkheden de twee laagste waarschijnlijkheden en voeg de som van deze twee waarschijnlijkheden toe aan de lijst. [De lijst bevat nu 1 waarschijnlijkheid minder.]
- Als alles goed gaat, eindig je met een knoop met waarschijnlijkheid 1. Dit is de *wortel* van de boom.
- Associeer aan elke tak een bitwaarde. Twee takken die verbonden zijn met eenzelfde knoop hebben een verschillende bitwaarde.
- Coderen: Vorm de informatiebitsequentie die hoort bij een bepaald macrosymbool door het pad te volgen van de wortel van de boom naar het bijhorende blad en de corresponderende bitsequentie op te schrijven.
- Decoderen: Vind het eerstvolgende macrosymbool terug door in de boom het pad te volgen vanaf de wortel dat overeenkomt met de ontvangen bitsequentie. Lees het macrosymbool af zodra je een blad hebt bereikt. Vind een volgend macrosymbool door opnieuw aan de wortel te beginnen, enz...

APPENDIX B: SYNDROOMDECODERING

Om een checkmatrix $\boldsymbol{H}$ van een lineaire blokcode te bepalen op basis van een generatormatrix $\boldsymbol{G}$ ga je als volgt te werk:

- Voer rijbewerkingen en/of kolompermutaties uit op $\boldsymbol{G}$ om een matrix te bekomen van de vorm $[\boldsymbol{IP}]$, met $\boldsymbol{I}$ de eenheidsmatrix.
- Vorm de matrix $\boldsymbol{X} = \left[\boldsymbol{P}^T \boldsymbol{I}\right]$.
- Pas op $\boldsymbol{X}$ de inverse kolompermutaties toe. De matrix die je bekomt is een checkmatrix $\boldsymbol{H}$ van de code.

Om $\boldsymbol{G}$ te bepalen op basis van $\boldsymbol{H}$, ga je op een gelijkaardige manier te werk.

Om een syndroomtabel op te stellen, ga je als volgt te werk:

- Overloop de codewoorden in volgorde van toenemend Hamminggewicht (aantal 1-en in de sequentie).
- Bepaal de bijhorende syndromen (20).
- Staat dit syndroom reeds in de tabel?
  - JA: ga verder met hetvolgende codewoord.
  - NEE: Schrijf het syndroom in de eerste kolom van de tabel. Schrijf het bijhorende codewoord op dezelfde lijn in de twee kolom van de tabel.

De codewoorden in de tweede kolom van de syndroomtabel fungeren als foutpatronen $\boldsymbol{e}$ die gebruikt worden in het geval van foutcorrectie (syndroomdecodering). De ontvanger berekent eerst het syndroom van het ontvangen woord. Vervolgens telt de ontvanger het foutpatroon dat hoort bij dit syndroom (zie syndroomtabel) binair op bij het ontvangen woord. Het resultaat is het gecorrigeerde codewoord.

In het geval van transmissie over een BSC met bitovergangswaarschijnlijkheid p, kan de syndroomtabel ook gebruikt worden om snel de kans op een decodeerfout in geval van volledige decodering $p_{e,COD}$, de

kans op een niet-detecteerbare fout p_m en de kans op decodeerfalen in het geval van zuiver foutdetectie $p_{f,DET}$ te bepalen.

$$\begin{aligned} p_{e,COR} &= 1 - \sum_{\boldsymbol{e}\in\mathcal{T}} p^{w(\boldsymbol{e})} (1-p)^{n-w(\boldsymbol{e})} \\ &\approx N_{e,t+1} p^{t+1} \end{aligned}$$

$$\begin{aligned} p_m &= \sum_{\boldsymbol{c}\in\mathcal{C}_\mathbf{0}} p^{w(\boldsymbol{c})} (1-p)^{n-w(\boldsymbol{c})} \\ &\approx N_{c,\mathcal{L}+1} p^{\mathcal{L}+1} \end{aligned}$$

$$\begin{aligned} p_{f,DET} &= \sum_{\boldsymbol{e}\in\mathcal{T}_\mathbf{0}} \sum_{\boldsymbol{c}\in\mathcal{C}} p^{w(\boldsymbol{e}+\boldsymbol{c})} (1-p)^{n-w(\boldsymbol{e}+\boldsymbol{c})} \\ &\approx Np \end{aligned}$$

met $\mathcal{C}_\mathbf{0}$ de verzameling van geldige codewoorden behalve het nulwoord, $\mathcal{T}$ de verzameling van de foutvectoren in de syndroomtabel, $\mathcal{T}_\mathbf{0}$ de verzameling $\mathcal{T}$ zonder het nulwoord, n de bloklengte van de codewoorden, $w(\cdot)$ een functie die het Hamminggewicht van een bitsequentie bepaalt. De benaderingen zijn geldig voor kleine waarden van p, waarbij $t > 0$ het gegarandeerd foutcorrigerend en $\mathcal{L} > 0$ het gegarandeerd foutdetecterend vermogen van de code voorstelt, en met

- $N_{e,t+1}$: het aantal foutpatronen in de syndroomtable van gewicht $t+1$.
- $N_{c,\mathcal{L}+1}$: het aantal codewoorden van gewicht $\mathcal{L}+1$.

APPENDIX C: LLOYD-MAX ALGORITME

Voor elke optimale kwantisator zijn de Lloyd-Max voorwaarden voldaan. De voorwaarden zijn als volgt:

$$\begin{aligned} r_i &= \frac{q_i + q_{i+1}}{2} \qquad i = 1, 2, ..., M-1 \\ q_i &= \frac{\int_{r_{i-1}}^{r_i} u f_U(u)\, du}{\int_{r_{i-1}}^{r_i} f_U(u)\, du} \qquad i = 1, 2, ..., M \end{aligned}$$

waarbij r_0 en r_M gegeven zijn door het domein van $f_U(u)$. Uit dit stelsel van niet-lineaire voorwaarden is het echter meestal niet eenvoudig om rechtstreeks de waardes voor r_i en q_i te bepalen. De optimale waarden kunnen echter wel benaderd worden door gebruik te maken van een iteratieve methode die om de beurt de grenzen en de niveaus opnieuw berekent. Dit proces gaat door tot wanneer de vermindering in distorsie voldoende klein is. Deze methode heet het Lloyd-Max algoritme en de stappen worden hieronder gegeven:

1) Initialiseer alle q_i (maak zelf een logische keuze), stel $l = 1$.
2) Update alle kwantisatiedrempels: $r_i = \frac{q_i+q_{i+1}}{2} \quad i = 1, 2, ..., M-1$
3) Update alle kwantisatieniveaus: $q_i = \frac{\int_{r_{i-1}}^{r_i} u f_U(u)\, du}{\int_{r_{i-1}}^{r_i} f_U(u)\, du} \quad i = 1, 2, ..., M$
4) Bepaal de distorsie: $\sigma^2_{\text{e,l}} = \int_{r_0}^{r_M} (F_Q(u) - u)^2 f_U(u)\, du$
5) Indien $\frac{\sigma^2_{\text{e,l-1}} - \sigma^2_{\text{e,l}}}{\sigma^2_{\text{e,l-1}}} < \epsilon$, stop. Zo niet, stel $l = l + 1$ en ga naar stap 2.

Hierbij is l de iteratie index en $\sigma^2_{\text{e,l}}$ de GKD na de $l$de iteratie. De stopvoorwaarde wordt bepaal door ϵ. Kies deze voldoende klein.