SIT213 Etape 3
IMT Atlantique

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

21 SEPTEMBRE

Créé par :
LE DUC Elouan
MAQUIN Philippe
LE GRUIEC Clément
LE JEUNE Matthieu
FRAIGNAC Guillaume

Table des matières

Intentions du projet

Il s'agit de réaliser, par équipe de 4 ou 5 élèves, une maquette logicielle (en Java) simulant un système de transmission numérique élémentaire. On intégrera donc dans la chaîne un bloc de modulation numérique.

Le système sera assemblé suivant une bibliothèque de modules comportant des ports d'entrée, des ports de sortie et des paramètres physiques. Ces derniers pourront être déterminés à partir des activités du module SIT 212. Le système global sera mis au point progressivement sur 5 séances au cours desquelles les modules seront raffinés, complétés, validés et connectés selon un schéma de transmission de type « point-à-point ».

Outre la qualité technique de la réalisation, on insistera sur les points suivants :

1. La qualité de documentation de la maquette logicielle (notamment la JavaDoc).

2. Les efforts de validation des résultats de simulation produits par la maquette.

3. La maîtrise du processus de travail : gestion des versions successives de la maquette logicielle et du dossier technique afférent, synergie de l'équipe, démarche qualité. Concernant ce tout dernier critère, le respect des exigences de mise en forme du livrable sera primordial.

3ème étape du projet

➢ Objectifs

Dans cette phase du projet nous allons travailler sur une transmission non-idéale avec canal bruité de type « gaussien » d'un signal analogique. La propagation dans le canal est modélisée de manière théorique par un bruit blanc additif gaussien qu'il conviendra de régler en fonction des paramètres du transmetteur vus à l'étape précédente. On fera attention à surveiller le TEB en réception.

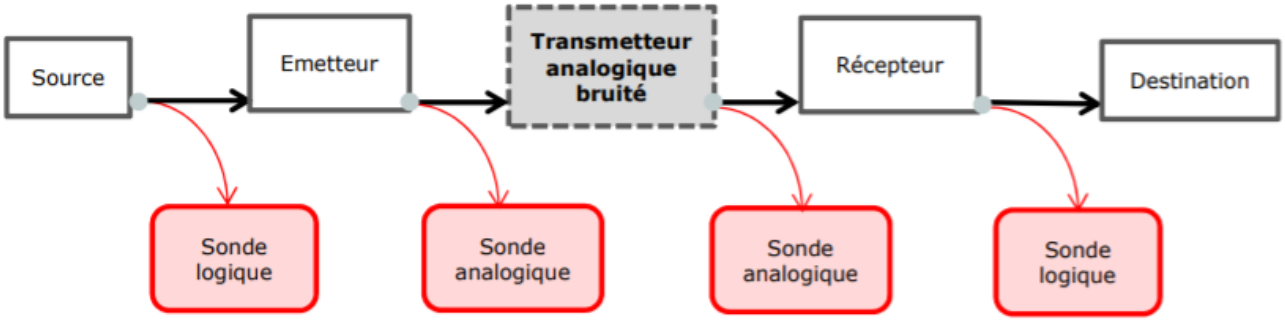


Figure 1 : Modélisation de la chaîne de transmission à l'étape 3.

Par défaut le simulateur doit utiliser une chaîne de transmission logique, avec un message aléatoire de longueur 100, sans utilisation de sondes et sans utilisation de transducteur.

En plus des paramètres des deux premières étapes on va ici ajouter un bruit gaussien.

L'option ***-snrpb s*** en transmission analogique bruitée permet de donner la valeur du rapport signal sur bruit par bit (Eb/N0 en dB). Nous ferons attention à utiliser un paramètre flottant. Par défaut le « SNRPB » est à 10000000 (bruit non visible).

➢ Analyse des actions à mener

Nous allons devoir ajouter un bruit blanc gaussien dans le simulateur de signaux analogiques que nous avons développé au cours de la seconde étape. Pour générer le bruit blanc gaussien on utilise la formule suivante :

$$b(n) = \sigma_b \sqrt{-2\ln(1 - a_1(n))} \cos(2\pi a_2(n)) \qquad \begin{matrix} a_1(n) \sim \mathcal{U}[0,1[\text{ (loi uniforme)} \\ a_2(n) \sim \mathcal{U}[0,1[\end{matrix}$$

Par la suite nous allons additionner le signal analogique créé à la précédente étape au bruit blanc gaussien. Comme l'illustre le schéma ci-dessous.

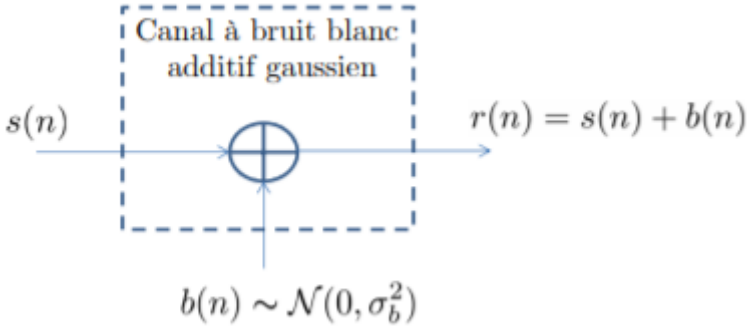


Figure 2 : Ajout du bruit blanc gaussien.

On obtient ainsi le signal r(n) qui correspond au signal d'entrée additionné au bruit blanc. Il faudra vérifier que le bruit généré suit une loi gaussienne (histogramme).

Comme pour l'étape précédente nous retrouvons les 3 formes à prendre en compte, à savoir :

- NRZ
- NRZT
- RZ

D'après le schéma de la *figure 1* nous devons modifier notre Transmetteur analogique « parfait » en un transmetteur analogique « bruité gaussien ». Les blocs émetteur, récepteur, source et destination sont les mêmes qu'à la précédente étape. Pour ce qui concerne les sondes permettant de visualiser les signaux, elles ont déjà été programmées par les enseignants.

Nous devons implémenter le paramètre suivant : *-snrpb s*. Nous devrons nous assurer de détecter les paramètres du programme et d'assurer de leur conformité en utilisant des Regex par exemple.

➢ Programmation

À la suite de l'étape 2 notre programme avait subi un test de performance. Il s'est avéré qu'il n'était pas aussi performant que souhaité (lenteurs).

Nous avons changé le maximum de boucles for classiques par des foreach.

```
for (int i = 0; i < informationRecue.nbElements(); i++) {
    Boolean b=informationRecue.getIemeElement(i);
    //code...
}

//changé en

for (Boolean recu : informationRecue) {
    Boolean b = recu;
    //code
}
```

Dans le code ci-dessus les boucles font la même chose, cependant dans la 2e version c'est plus optimisé.

Une des plus grosses améliorations est le changement du type de liste utilisée par l'information. En effet dans le code d'origine fourni ce sont des LinkedList qui ne sont pas adaptées à l'utilisation du simulateur. Elles ont été remplacées par des ArrayList beaucoup plus rapide dans notre cas d'usage.

Opération	ArrayList	LinkedList
get()	O(1)	O(n)
add()	O(1)	O(1)
remove()	O(n)	O(1)

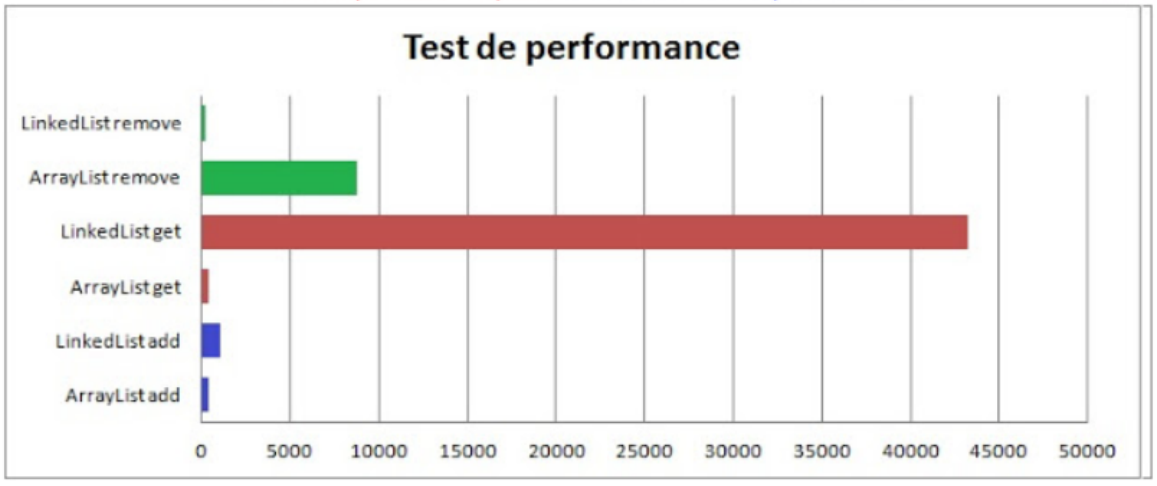


Figure 3 : source http://www.codeurjava.com/ - ArrayList vs LinkedList (abscisse = temp en ms)

Concernant l'utilisation du programme, ceux sont toujours les mêmes méthodes, elles ont juste été légèrement modifiées pour correspondre aux nouvelles listes.

```
private ArrayList<T> content;

public Information() {
        this.content = new ArrayList<T>();
}

public Information(T[] content) {
        this.content = new ArrayList<T>();
        for (int i = 0; i < content.length; i++) {
            this.content.add(content[i]);
        }
}

public void add(T valeur) {
        this.content.add(valeur);
}
```

Grâce à cette optimisation il est désormais possible de simuler une chaine de transmission avec 1 million de symboles en entrée (une dizaine de seconde avec affichage activé). Les sondes quant à elles mettent un peu de temps à afficher l'ensemble des symboles qui sont très nombreux. (4*1Million à afficher).

En plus de ces améliorations, d'autres méthodes ont été reprises afin de les simplifier et donc optimiser le code pour un résultat identique.

Ces améliorations participent à rendre notre programme meilleur, dans le cadre des simulations le but est d'avoir un résultat qualitatif le plus rapidement possible.

➢ Tests et validations du programme

A partir de la troisième étape nous avons commencé à automatiser nos tests. Ces différents tests ont pour but de vérifier si l'ensemble du programme est correctement installé sur une machine.

✓ Résultats attendus

Nous sommes dans le cas d'une chaine de transmission avec un transmetteur analogique bruité gaussien. Le signal reçu doit être le plus fidèle possible au signal émis qu'il soit converti sous une forme NRZ, NRZT ou encore RZ.

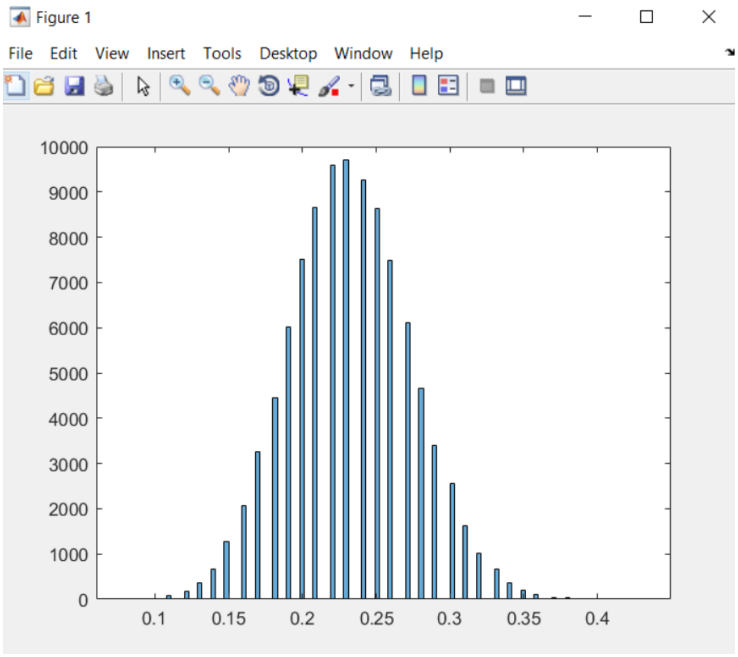
Dans tous les cas le TEB doit être le plus proche de 0 à la fin de la chaine de transmission si tout se passe bien. Les formes d'ondes choisies doivent être utilisées selon leurs spécifications.

Pour les tests nous utiliserons un signal avec les paramètres suivants :

- Amplitude max : 5V ;
- Amplitude min : -5V ;
- Nb échantillon : 60 ;
- Seed : 40 ;
- Longueur (mess) : 20 ;
- Snrpb : 1 ;

Le signal aléatoire généré sera égal à 1101 0011 0100 1001 1111 Ce message est intéressant, il permet de tester différents cas de figure, par exemple deux 1 qui se suivent, le passage d'un 0 à un 1 etc… (On utilise le même signale que pour la seconde étape)

✓ Tests avec signal de type NRZ



Exemple d'histogramme obtenue à partir du résultat de 100K simulations et un Snrpb à -6dB

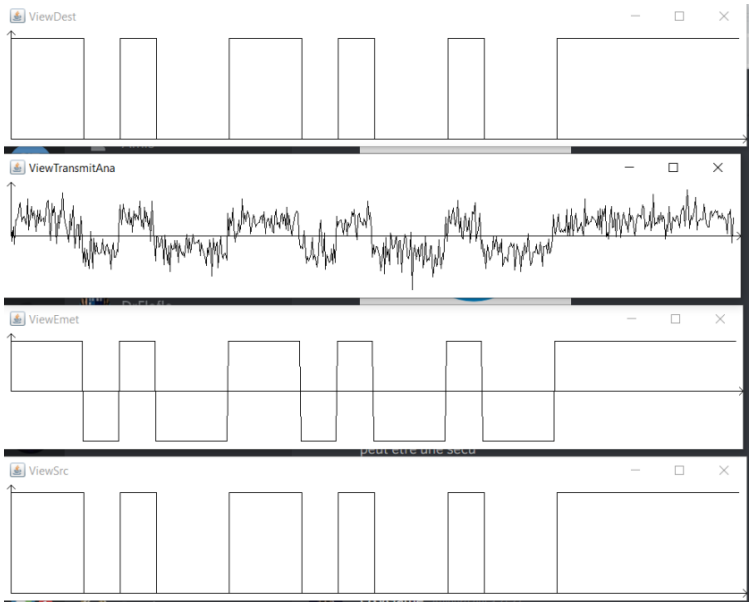
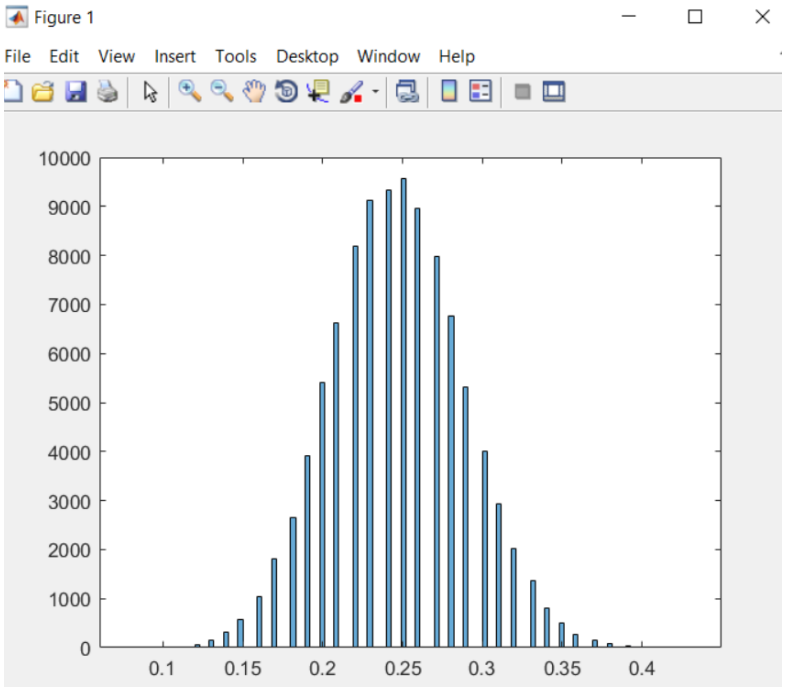


Figure 4 : Tests du signal de type NRZ

« java Simulateur -s -mess 20 -seed 40 -form NRZ -ampl -5 5 -snrpb 1 => TEB : 0.0 »

Le signal émis correspond bien au signal reçu. Le TEB le confirme et est bien égal à 0. On peut également observer que le bruit ajouté dans transmission est bien décomposé à la réception.

✓ Tests avec signal de type NRZT



Exemple d'histogramme obtenue à partir du résultat de 100K simulations et un Snrpb à -6dB

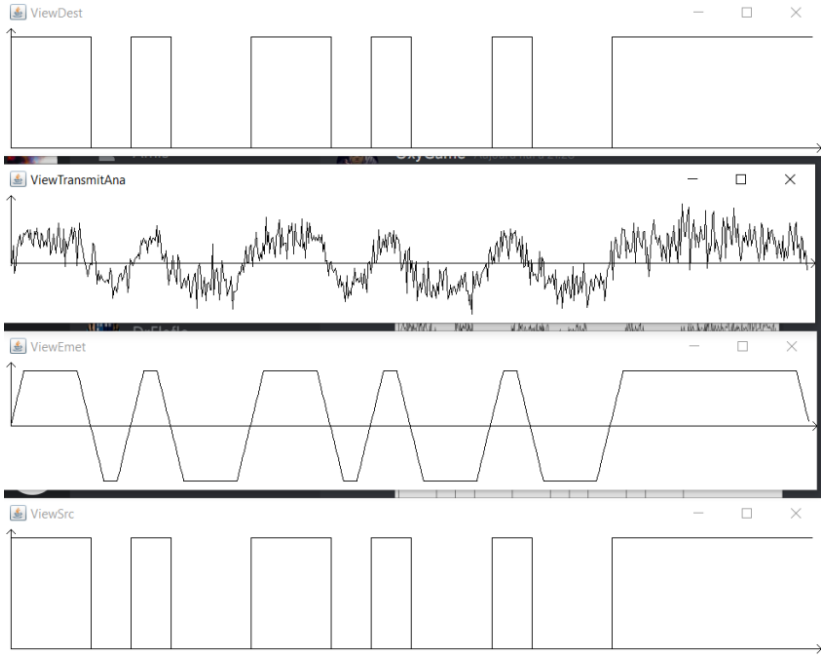
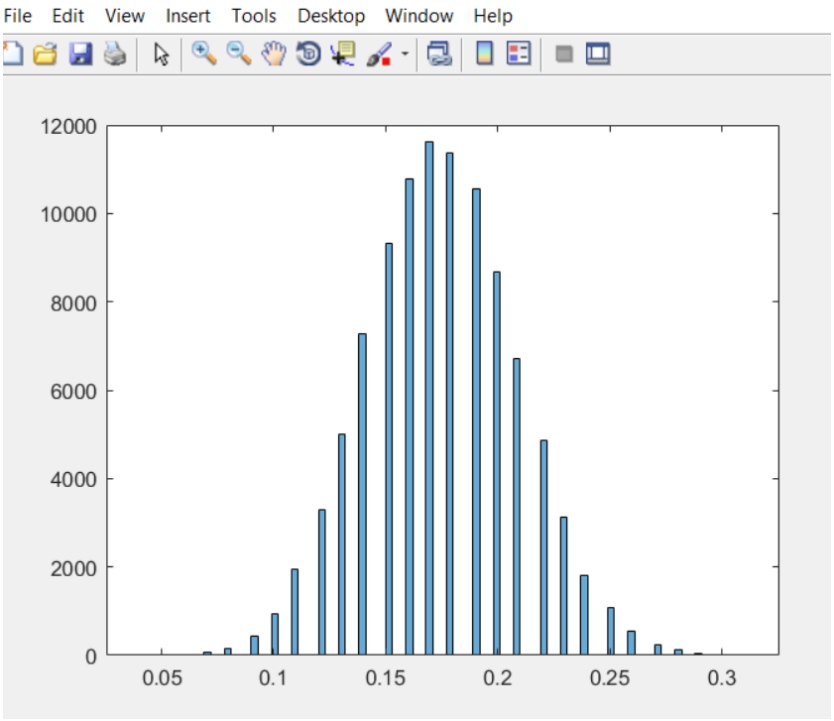


Figure 5 : Tests du signal de type NRZT

« java Simulateur -s -mess 20 -seed 40 -form NRZT -ampl -5 5 -snrpb 1 => TEB : 0.0 »

Le signal émis correspond bien au signal reçu. Le TEB le confirme et est bien égal à 0. Tout comme le précèdent signal le bruit a bien été traité.

✓ Tests avec signal de type RZ



Exemple d'histogramme obtenue à partir du résultat de 100K simulations et un Snrpb à -6dB

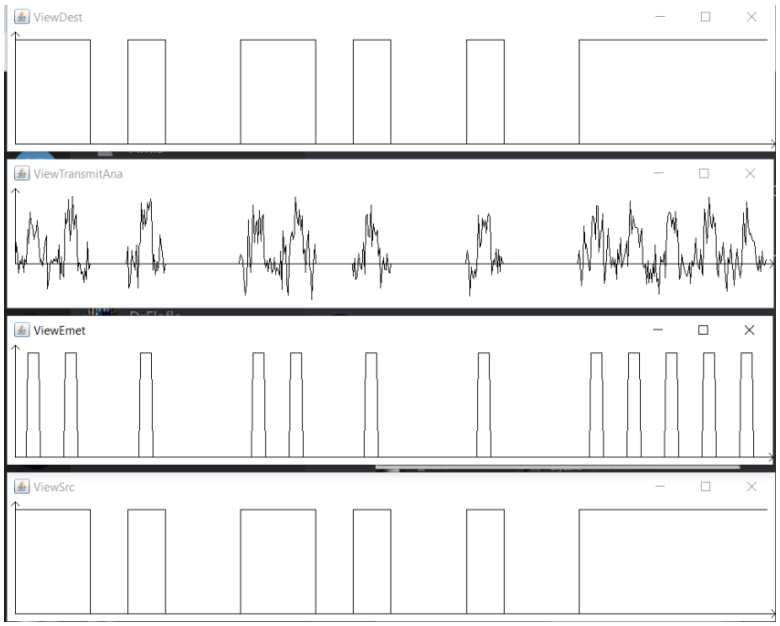


Figure 6 : Tests du signal de type NRZ

« java Simulateur -s -mess 20 -seed 40 -form RZ -ampl -5 5 -snrpb 1 => TEB : 0.0 »

Le signal émis correspond bien au signal reçu. Le TEB le confirme et est bien égal à 0. Tout comme le précèdent signal le bruit a bien été traité.

✓ Tests avec signal de type NRZT et beaucoup de bruit

Nous augmentons fortement le bruit en fixant le snrpb à -20dB. On s'attend à avoir un signal mal décodé et avec un TEB différent de 0.0. En effet -20dB correspond à avoir 100 fois moins de signal que de bruit.

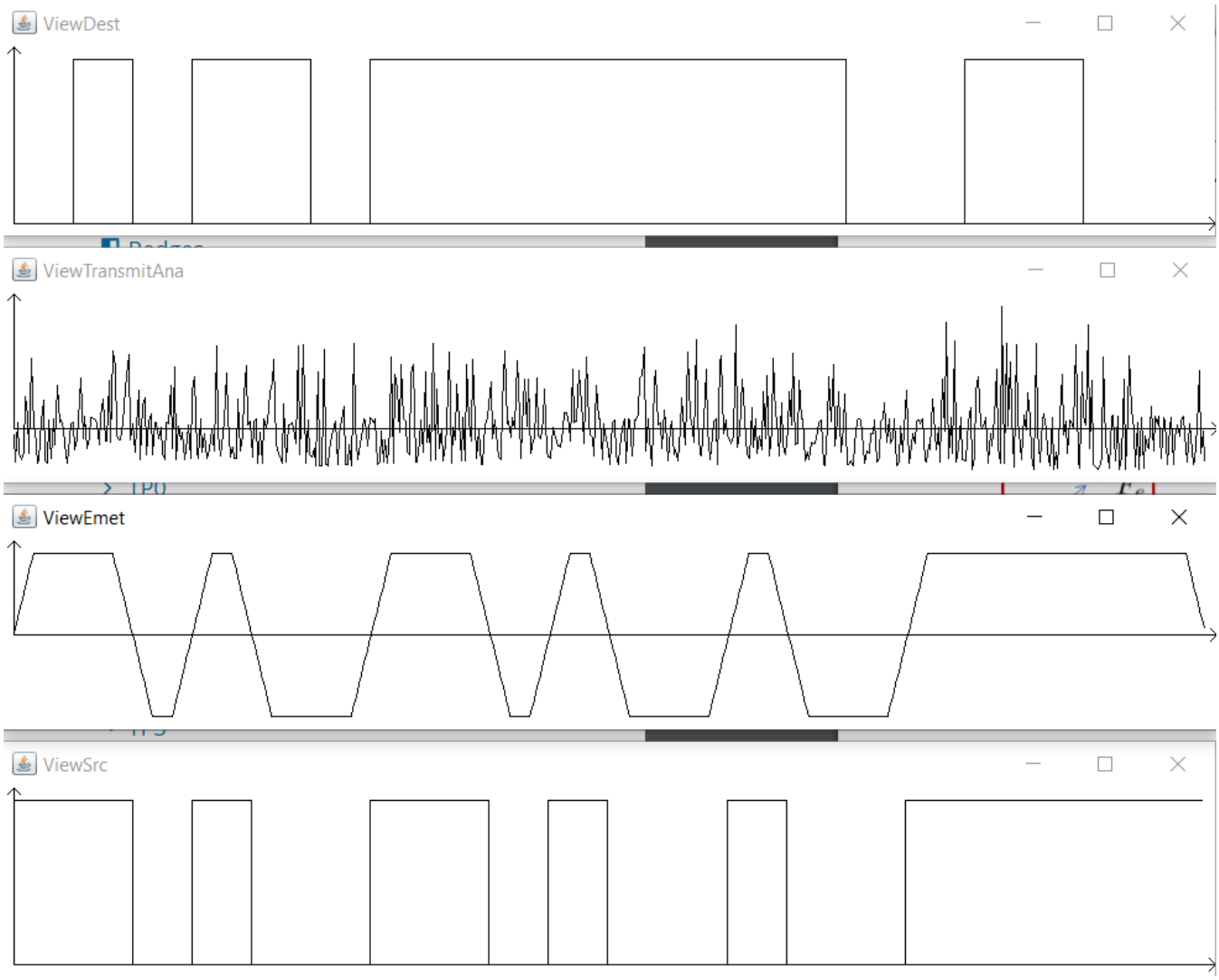


Figure 7 : NRZT avec un snrpb à -20dB

« java Simulateur -s -mess 20 -form NRZT -ampl -5 5 -snrpb -20 -seed 40 => TEB : 0.45 »

Comme on s'y attendait le signal est très mal décodé en réception. Le TEB est très mauvais sachant qu'il tend vers 0.5 pour un SNRpb tendant vers $-\infty$.

Conclusion

Nous avons pu simuler un transmetteur analogique bruité. Les tests se sont passés comme prévus, les résultats sont donc satisfaisants.
Dans la prochaine étape nous allons devoir développer une transmission non-idéale avec divers bruits « réels » qui viendront dans un premier temps remplacer le bruit gaussien. Ensuite nous feront l'addition de ces deux bruits.

Table des illustrations