TP5 : QPSK - Réception : partie bande de base

Introduction

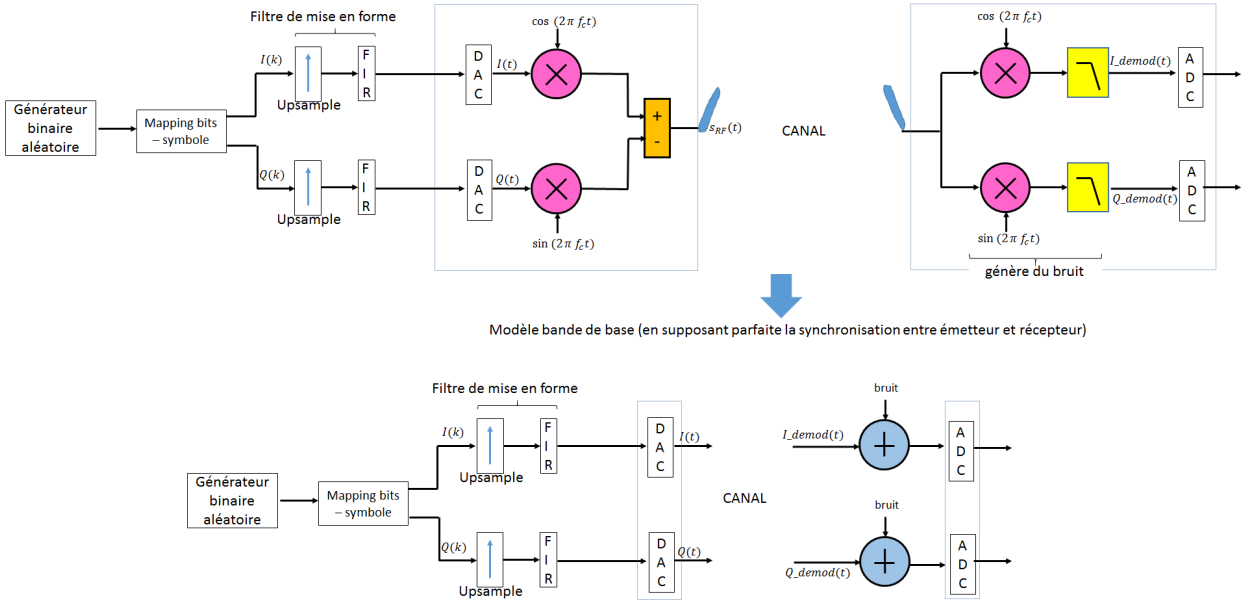
Nous consacrons deux séances à la QPSK. La première traitait du modulateur, de la partie RF du démodulateur, de l'implémentation d'un filtre de mise en forme pour réduire la bande côté émetteur.

Ce que nous avons vu, en particulier, c'est que côté récepteur, si les filtres de réception éliminent la composante de fréquence double (autour de $2f_c$), on retrouve les mêmes $I(t)$ et $Q(t)$ que côté émetteur – autrement dit les parties RF sont transparentes.

En fait, il y a deux limites :

— ça ne fonctionne que si la fréquence porteuse côté récepteur est *parfaitement synchrone* de celle de l'émetteur. Or, en réalité, les oscillateurs ont une incertitude relative de l'ordre du ppm (incertitude de 1 kHz si $f_c = 1$ GHz). Dans ce TP, nous travaillerons en simulation uniquement (sans USRP), mais avec des références dans le texte à une implantation pratique. *Nous supposerons que les oscillateurs sont parfaits*, de sorte qu'on pourra ne pas représenter les partie RF côté émetteur et récepteur. On dira qu'on utilise un *modèle bande de base* ;

— l'étage RF du récepteur ajoute du bruit ; on en tient compte en l'ajoutant à l'entrée du récepteur (partie 2).



Dans ce TP, nous traitons la partie bande de base du récepteur. En particulier, nous allons parler d'interférences entre symboles, de diagramme de l'oeil, de bruit et de filtre adapté ; enfin, de la récupération du timing symbole.

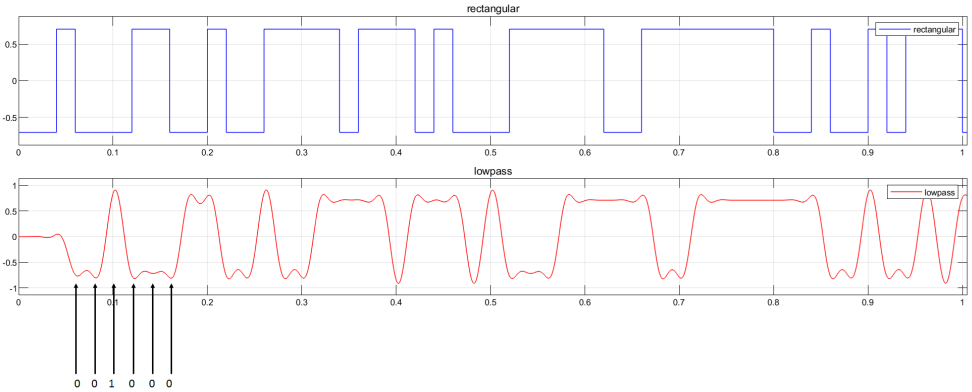
Pendant la majeure partie du TP, on utilise un débit de symboles, noté D_s, de 50 symboles/s, comme dans la partie simulation du TP4.

1 ISI et diagramme de l'oeil

Côté émetteur, le filtre de mise en forme permet de réduire la bande occupée à une bande environ égale au débit de symboles (50 Hz), ou un peu plus. Mais pourquoi utiliser pour cela, la plupart du temps, un filtre en cosinus surélevé, et pas un autre filtre ?

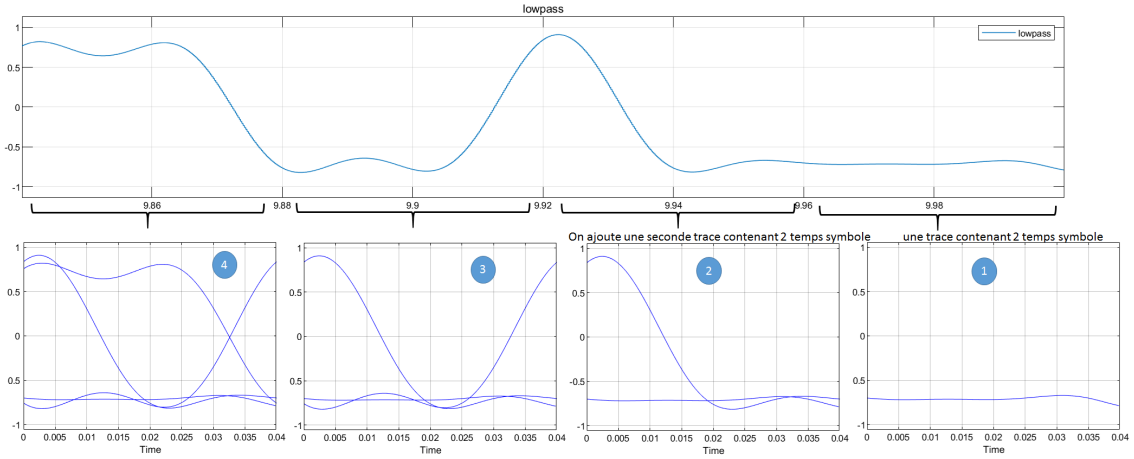
1. Pour le comprendre, lancer Matlab, ouvrir le fichier `recepteur_qpsk_1`. Comme expliqué précédemment, il s'agit d'un modèle bande de base, donc sans les parties RF (pas de transposition vers f_c puis de retour en bande de base). Nous avons ici 4 filtres : le filtre en cosinus surélevé, un autre filtre qui a été utilisé dans certaines applications, le filtre gaussien (GSM) ; un filtre passe-bas appliqué à la sortie du filtre rectangle pour éliminer les lobes – sur le modèle de ce qu'on fait pour des signaux analogiques ; enfin, le filtre rectangle (I et Q sont maintenus constants sur le temps symbole grâce à un DAC).
   Simuler, et observer d'abord les spectres à gauche. Les trois filtres permettent de réduire la bande par rapport au sinus cardinal du filtre rectangle. Ce n'est donc pas cet aspect qui permet réellement de trancher entre eux.

2. Ouvrir le scope (au milieu). Vérifier que les trois signaux $I(t)$ reçus, certes déformés, "arrondis", par rapport à la sortie du filtre rectangle, permettront sans doute de récupérer les niveaux de I et Q, du moins s'ils sont échantillonnés *à l'instant idéal*, ou optimal (cette question de l'instant idéal est traitée dans la partie 4), comme représenté ci-dessous pour la sortie du filtre passe-bas :

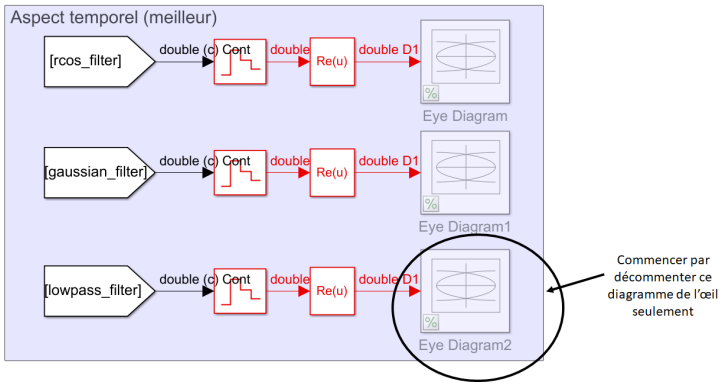


   En fait, pour trancher, il faut savoir quel filtre est le moins moins susceptible de causer une erreur – autrement dit, avec quelle sortie on aura la probabilité d'erreur bit la plus faible, et ce *pour la même puissance du signal* (imaginer que tous les signaux sont plus ou moins amplifiés pour être mis à la même échelle) *et le même bruit*.

3. On utilise pour cela un outil très visuel : *le diagramme de l'oeil*. La construction du diagramme est simple : on superpose un grand nombre de traces durant 2 ou 4 temps symboles. Observer le schéma ci-dessous (à lire de droite à gauche) et repérer sur chaque diagramme du bas la trace du signal du haut qui vient d'être superposée :



Lorsque vous avez bien compris ce schéma, décommenter (clic droit > `Uncomment`), à droite du modèle, le diagramme de l'oeil du bas seulement.



Relancer la simulation, puis cliquer sur le petit engrenage pour accéder aux propriétés ; augmenter progressivement le nombre de traces à afficher (`Traces to display`) à 2, 5, 10, etc, jusqu'à 100, pour bien vérifier le principe du diagramme de l'oeil. Où se situeront les instants idéaux pour échantillonner le signal (ie pour décider si on a plutôt un I positif ou négatif ?)

A présent, décommenter les deux autres diagrammes de l'oeil et relancer la simulation. Observer les résultats, en particulier le diagramme de la sortie du filtre en cosinus surélevé. Pourquoi, à la vue de ce dernier, peut-on dire qu'avec le filtre en cosinus surélevé, il n'y a pas d'ISI (Inter Symbols Interférences ou interférences entre symboles) ? A quelle propriété de la réponse impulsionnelle du filtre cela est-il dû ?

Pourquoi le filtre passe-bas est-il moins bon ?

2 Canal AWGN et filtrage du bruit

La partie RF du récepteur créé du bruit thermique, qui est :

— blanc ;

— gaussien.

Se rappeler que pour un processus aléatoire stationnaire à temps continu $X(t)$, on a :

$$P_X = \mathbb{E}[X^2(t)]$$

Et par ailleurs on a par définition :

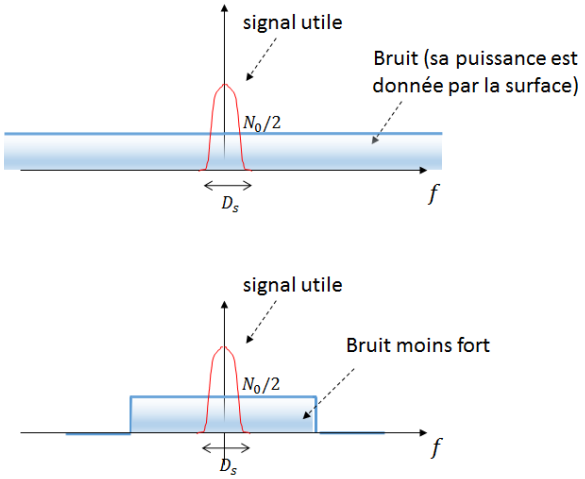
$$R_{XX}(0) = \mathbb{E}[X^2(t)]$$

Enfin, si la moyenne est nulle :

$$R_{XX}(0) = C_{XX}(0) = Var[X(t)]$$

La variance est donc égale à la puissance.

Pour réduire la puissance du bruit (donc sa variance), il suffit de réduire sa bande par filtrage (rappel : la puissance est l'intégrale de la DSP le long de l'axe des fréquences).
Ci-dessous, la DSP du signal utile et celle du bruit, avant et après filtrage :



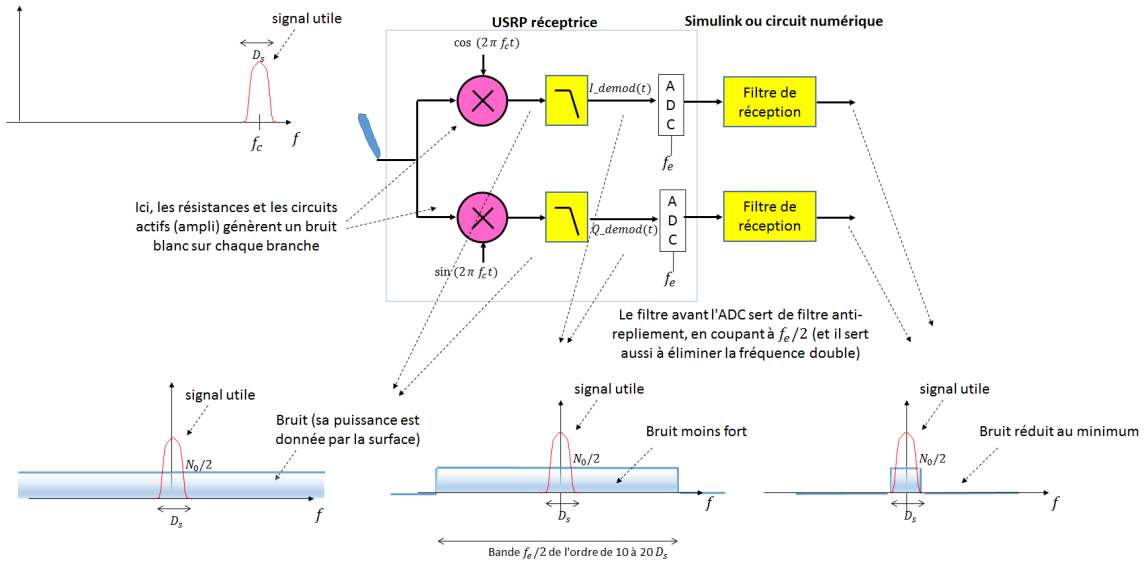
En pratique

Dans une USRP utilisée comme récepteur, après retour en bande de base, le signal entaché de bruit est numérisé par un ADC à la fréquence d'échantillonnage f_e.Le filtre situé *avant* l'ADC sert à éliminer la composante double à $2f_c$; en fait il sert aussi à éviter le repliement de spectre (contrainte plus forte) et pour cela, il coupe à $f_e/2$.
Ce faisant, ce filtre opère une première réduction de la puissance du bruit.

Le bruit est encore trop fort ; il faut réduire sa bande davantage. Ainsi, le premier élément de la partie bande de base du récepteur est *un autre filtre*, numérique celui-ci, appelé *filtre de réception.*

Jusqu'où réduire la bande ? Jusqu'à *la limite du spectre utile*, pour réduire au maximum la puissance du bruit, sans trop déformer le signal. On garde une bande d'environ D_s de $-D_s/2$ à $+D_s/2$, où $D_s = 1/T_s$ est le débit symbole [1].
Les différentes opérations décrites sont résumées sur le schéma suivant. (Note : en sortie des multiplieurs, on n'a pas représenté la composante à $2f_c$)

1. Rappel : quelle était la largeur du premier lobe du signal émis avec un filtre rectangle ?



Plus précisément, on peut montrer que le filtre de réception est *optimal*, c'est-à-dire qu'il maximise le rapport signal sur bruit, et donc minimise la probabilité d'erreur, s'il est *adapté* au filtre d'émission.

Filtre adapté : définition

Si l'on note $h_e(t)$ la réponse impulsionnelle côté émetteur, il faut avoir côté récepteur :

$$h_r(t) = h_e^*(-t)$$

Pour un filtre d'émission réel et symétrique, comme le filtre en cosinus surélevé, le filtre de réception doit lui être identique.

Simulation

Ouvrir le fichier `recepteur_qpsk_2` (à chaque fois que vous ouvrez un nouveau fichier, fermer le précédent).
Dans un modèle bande base, comme les parties RF ne sont pas représentées, on ajoute le bruit au niveau du canal. On parle de canal AWGN (Additive White Gaussian Noise).

1. Vérifier qu'à gauche, on trouve le même émetteur que précédemment (avec filtre en cosinus surélevé)
2. La sortie du filtre de mise en forme est-elle réelle ou complexe ? (Pour le vérifier, onglet `Debug > Information Overlays > Ports > Alias Data Types`, le `(c)` sur les fils signifie complexe.) Comment accéder à I et Q ?
3. Observer le canal ; pour ajouter du bruit à la fois sur la voie I et sur la voie Q, on a généré un *bruit complexe*. Au niveau du canal, on est à temps continu. On a dans ce cas, pour un bruit blanc $X(t)$, dont la bande a déjà été réduite à B :

   $$P_X = Var[X(t)] = S_{XX}(f) \times B$$

   où $S_{XX}(f)$ est la DSP de bruit, $N_0/2$.
   Double-cliquer sur l'un des blocs générant le bruit. Le paramètre `Noise Power` désigne en fait la DSP, tandis que le `sample time` est l'inverse de la bande de bruit. Que vaut la variance du bruit ajouté à I et à Q ?
4. A droite, l'ADC du récepteur est simulé par un bloc `zero-order hold`. Double-cliquer sur ce bloc et vérifier que la fréquence d'échantillonnage f_e en sortie vaut $20 \times D_s = 1$ kHz. Observer le filtre de réception : est-ce bien le filtre adapté à celui d'émission ? (Rappel : la syntaxe est `rcosdesign(`α`, Span, OSR, Shape)` ; le seul paramètre qui doit changer entre les deux est l'OSR à cause du choix de $f_e = 20D_s$).

5. Lancer la simulation et observer le diagramme de l'oeil en sortie. Y a-t-il de l'ISI ?

En fait, vu de la sortie, tout se passe comme si l'on avait deux filtres : le filtre d'émission (de mise en forme) suivi du filtre de réception. L'ensemble des deux est appelé *filtre produit*. Sa réponse impulsionnelle vaut :

$$h_p(t) = h_e(t) * h_r(t)$$

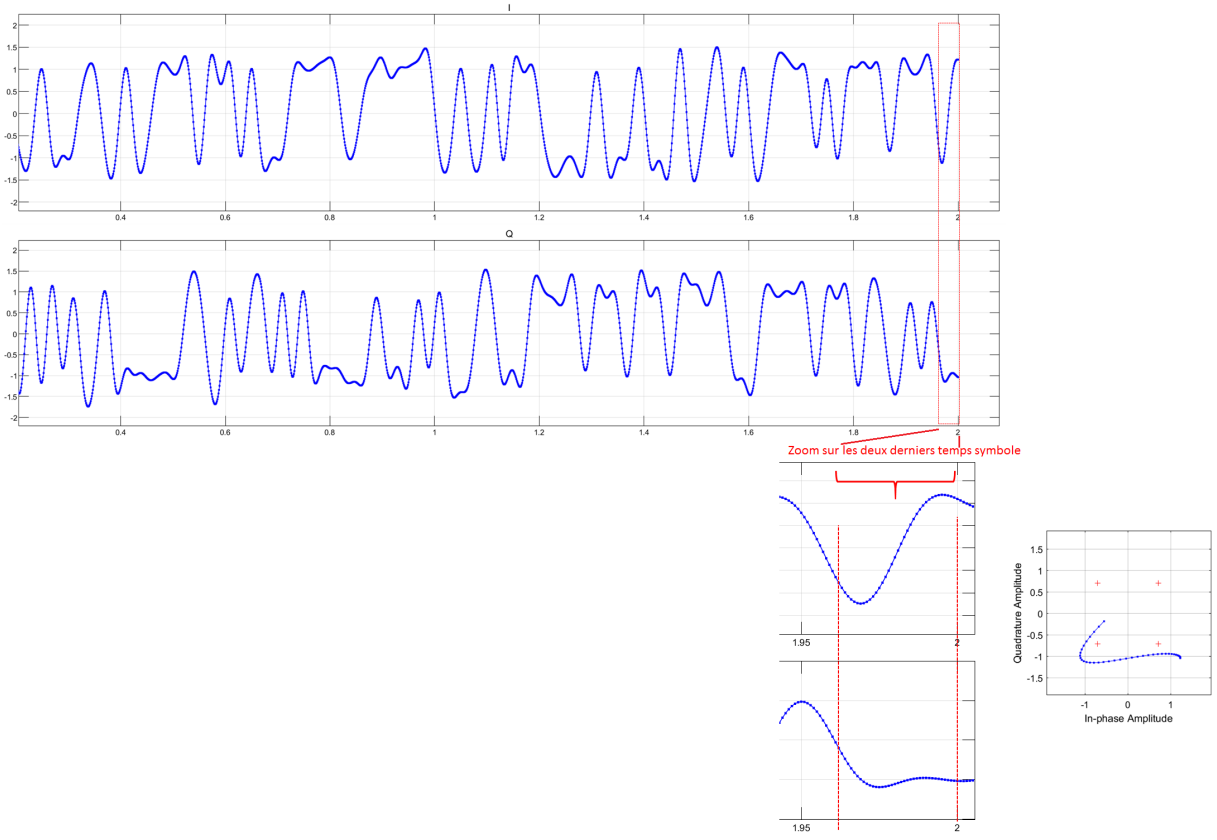
Pour éviter d'avoir de l'ISI en sortie du filtre de réception, c'est le filtre produit qui doit être un filtre en cosinus surélevé. Pour cela, il suffit d'utiliser de chaque côté un filtre en *racine de cosinus surélevé*.

6. Modifier les deux filtres en remplaçant le paramètre `'normal'` par `'sqrt'`. Puis relancer la simulation et vérifier le diagramme de l'oeil en sortie.
7. A présent, nous allons changer la variance du bruit pour que celui-ci soit significatif. Régler la variance pour qu'elle soit égale à 0.1, sur chaque voie I et Q.

8. Pour ne pas trop ralentir la simulation, le mieux est de décommenter les affichages un par un, puis de lancer à chaque fois la simulation. Observer successivement, en essayant de voir l'impact du filtre de réception :
   — le scope après ajout du bruit ;
   — le diagramme de l'oeil de I correspondant (avant filtrage) ;
   — le diagramme de l'oeil après filtrage ;
   — le spectre avant et après filtrage.
9. Idem en augmentant un peu la variance du bruit, par exemple à 0,5.

3 Le timing symbole

Après filtrage, on obtient dans le récepteur des signaux I et Q du type de ceux-ci :



En bas, on a zoomé sur les deux derniers symboles. Le but du jeu est de récupérer un échantillon bien placé – l'échantillon idéal du diagramme de l'oeil – à chaque temps symbole.

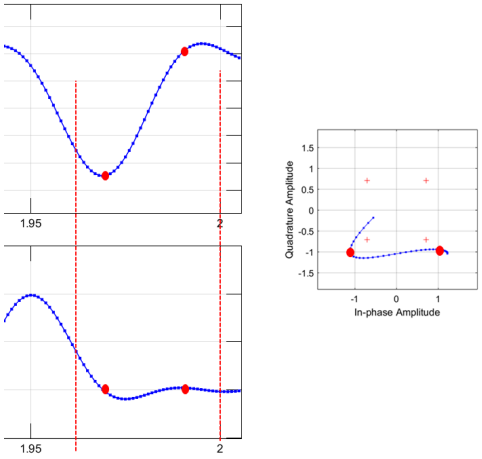
A droite, on a représenté la constellation (Q en fonction de I) pour les deux mêmes temps symbole seulement.

1. Noter approximativement les valeurs de I et Q au début de la partie zoomée, et en déduire le sens de parcours de cette constellation.

Pour avoir une chance d'avoir un échantillon placé à l'instant idéal sur chaque temps symbole, il en faut plus qu'un par temps symbole. C'est pourquoi nous avons dans la partie précédente choisi :

$$f_e = 20 \times D_s$$

soit 20 échantillons par temps symbole.
Sur les deux temps symboles encadrés ci-dessus, les échantillons idéaux seraient probablement ceux qu'on a repérés ci-dessous :



La constellation observée après qu'on a conservé un seul échantillon par temps symbole est un bon indicateur de la qualité de la récupération du timing symbole : si tout va bien, en superposant de nombreux temps symbole, on doit retrouver la constellation classique de la QPSK, avec 4 points.

2. Supposons qu'on a pu déterminer expérimentalement que, si les échantillons d'un temps symbole sont numérotés de 0 à 19, l'échantillon idéal, à conserver, était systématiquement le numéro 9. Ouvrir le fichier `recepteur_qpsk_3`. Dans ce fichier, on a ajouté un bloc pour garder l'échantillon idéal parmi les 20 : le bloc `Downsample`. Double-cliquer sur ce bloc : quel paramètre indique qu'on garde l'échantillon numéro 9 ?

Taux d'erreur binaire, probabilité d'erreur

Ainsi, on obtient sur chaque temps symbole l'échantillon idéal pour $I + jQ$. On peut en déduire par Mapping le couple de bits reçu.
Dans le bloc du bas à droite, on compare les bits reçus aux bits émis pour mesurer le taux d'erreur binaire. On souhaiterait comparer ce résultat à la probabilité d'erreur théorique [2].

2. On s'en doute : le taux d'erreur binaire converge vers la probabilité d'erreur si le nombre de bits testés est suffisant (loi des grand nombres). Mais il faut peut-être laisser tourner assez longtemps... Pour n bits envoyés, l'intervalle de confiance à 95% a une demi-largeur $L = 1.96 \times \sigma/\sqrt{n}$. On peut déduire σ de la probabilité d'erreur théorique p : $\sigma = \sqrt{p \times (1-p)}$. Par exemple, pour un bruit de variance 0.1 et 300000 bits envoyés, on trouve $L \simeq 10^{-4}$ (voir PBS).

A la sortie du filtre de réception, sur chaque voie, I et Q, si l'on conserve l'échantillon idéal, on obtient un échantillon ayant deux valeurs possibles, que nous noterons $\pm E$.
Par ailleurs, nous admettons qu'à la sortie du bloc `Downsample` :

— la variance du bruit n'est pas modifiée (les coefficients du filtre de réception ont été normalisés pour cela) ; on la note σ^2 (sur la voie I et sur la voie Q, elle est identique).

— le bruit est toujours blanc, donc deux échantillons de bruit sont indépendants (ce ne serait pas vrai avec n'importe quel filtre !)

3. Exprimer dans ce cas la probabilité d'erreur en fonction de E et σ, en utilisant la fonction Q (queue de la gaussienne centrée réduite).

4. A la sortie du bloc de Mapping, on sait que I et Q valent $\pm A = \pm\sqrt{2}/2$. On montre que E, dans notre cas, est fonction de A et des facteurs de sur-échantillonnage que l'on a utilisés, côté réception (noté OSR_r) et côté émission (noté OSR_e) :

$$E = A \times \sqrt{\frac{OSR_r}{OSR_e}}$$

(On rappelle qu'ici on a $OSR_e = 10$ et $OSR_r = 20$).

Le bloc le plus en bas calcule la probabilité d'erreur théorique (vous pouvez accéder au code en double-cliquant dessus). Revenir au schéma, lancer la simulation, observer les constellations avant et après la sélection de l'échantillon idéal. Puis, vérifier que le taux d'erreur binaire mesuré permet, si le temps est suffisamment long, d'estimer la probabilité d'erreur, avec peut-être une légère différence due à l'imprécision sur l'instant de sélection de l'instant optimal.
Vous pouvez relancer plusieurs fois : la `seed` des blocs change à chaque simulation, ce qui génère à chaque fois un bruit et des bits différents.
Idem en modifiant la variance de bruit : essayer une valeur plus forte, une valeur très faible (faire `CTRLE+H`).

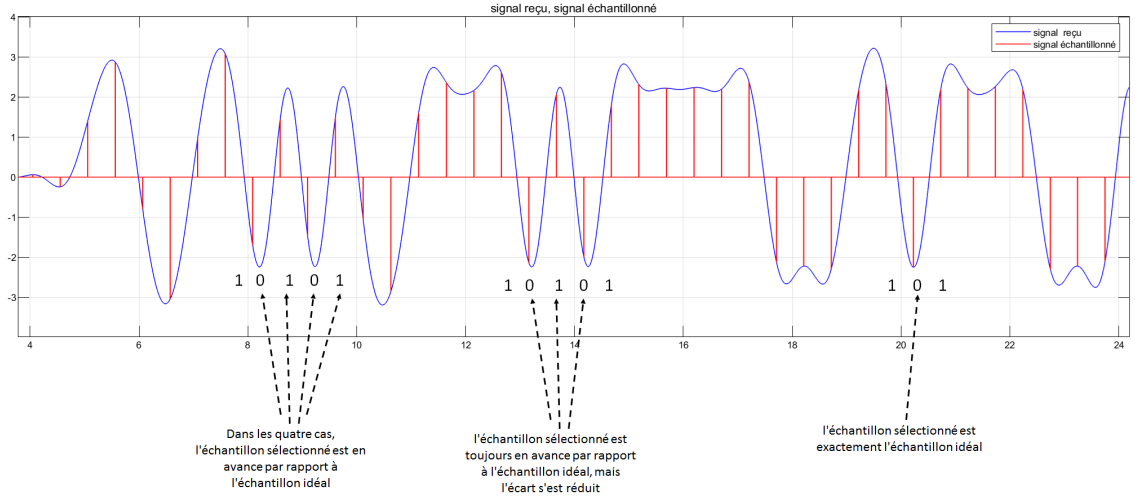
Où est le problème ?

En pratique, cette méthode ne peut pas marcher, pour deux raisons.

5. On suppose à présent que le canal introduit un délai non prévisible : décommenter le bloc situé juste après l'ajout de bruit. Simuler. Expliquer le résultat observé (constellation, taux d'erreur binaire). Puis recommenter le bloc introduisant le délai (clic droit, `Comment through`)
6. Pour finir, on considère que la fréquence d'échantillonnage de l'ADC, qui devrait être de 1000 Hz, est en fait de 1001 Hz (erreur de 0.1%). Changer la valeur en faisant `CTRLE+H`. Simuler. Expliquer le résultat observé (constellation, taux d'erreur binaire).

Le NCC

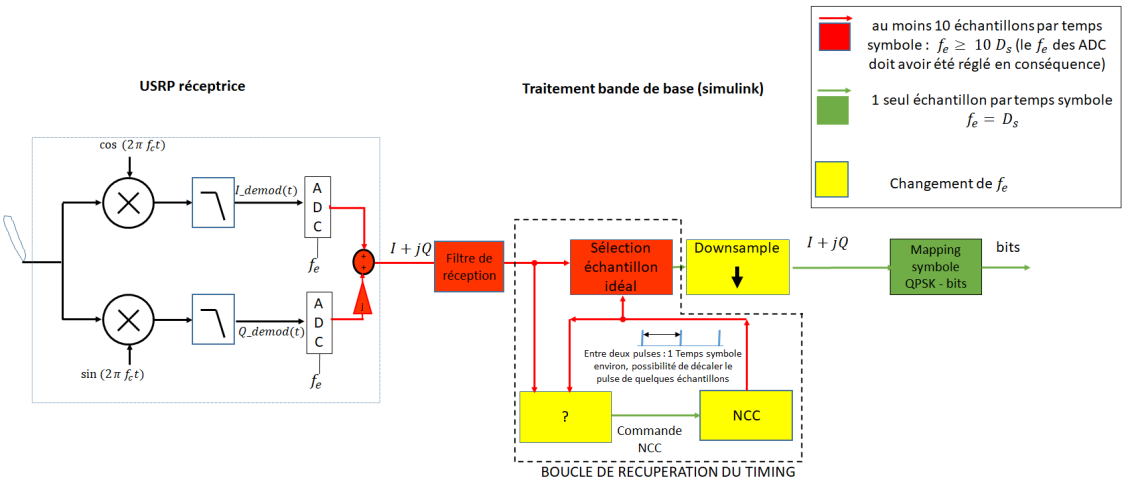
Pour résoudre le problème, l'idée est de sélectionner l'échantillon à l'aide d'une horloge délivrant *un pulse* à l'instant idéal à chaque temps symbole. Au départ, le pulse n'a en fait aucune raison de se trouver à l'instant idéal ; mais l'horloge doit *adapter automatiquement sa période* pour que le pulse puisse, au symbole suivant, intervenir un peu plus tôt s'il était en retard ; ou inversement un peu plus tard s'il était en avance. Ceci est illustré sur le schéma suivant. Dans ce schéma, le "signal échantillonné" est obtenu à partir du signal reçu et de la sortie de l'horloge (un pulse tous les temps symbole).



Il n'est pas forcément évident de voir si le pulse est bien placé en observant les signaux. Mais c'est assez facile dans le cas où l'on a (au moins) 2 changements de bits consécutifs : 0-1-0 ou 1-0-1. Dans ce cas, pour le bit du milieu, le pulse devrait intervenir sur le maximum pour 0-1-0, sur le minimum pour 1-0-1. C'est en cherchant ce type de motifs (mis en évidence sur le schéma) qu'on peut affirmer que l'horloge a, effectivement, légèrement augmenté sa période pour décaler le pulse jusqu'à atteindre l'instant idéal.

Pour que la période puisse légèrement varier, l'horloge doit comporter une *entrée de commande*. On appelle une telle horloge NCC (Numerically Controlled Clock).

Le schéma de principe pour notre récepteur est :



On rappelle que dans ce TP, nous traitons seulement la partie bande de base ; l'entrée du récepteur est $I_{\text{demod}}(t)$ et $Q_{\text{demod}}(t)$.
Remarquer que la numérisation par l'ADC doit être à $f_e \geq 10 \times D_s$ (couleur rouge) ; c'est seulement lorsque l'échantillon idéal est sélectionné qu'on peut réduire f_e afin de ne garder qu'un échantillon par temps symbole.

A partir de maintenant, on travaille à 2 symboles/s.
Le NCC doit délivrer un pulse toutes les 500 ms quand son entrée de commande, notée u, est nulle. Pour cela, il va compter de 1 à 100, ajoutant $1 + u$ à chaque incrément :

$$0 \quad \ldots \quad 1+u \quad \ldots \quad 2\times(1+u) \quad \ldots \quad \text{etc}$$

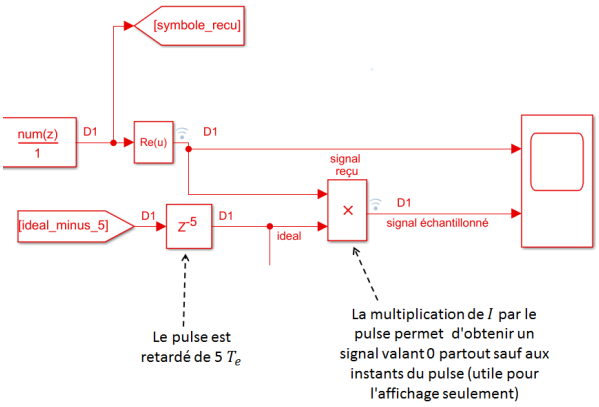
Quand on atteint ou dépasse 100, on remet le compteur à 0, et on a le pulse en sortie.

1. Pour que l'on ait un pulse toutes les 500 ms quand $u = 0$, quel doit être l'intervalle de temps entre deux incréments du compteur ? Dans la suite, on suppose qu'on a cet intervalle quelle que soit la valeur de u.
2. L'entrée de commande u est comprise entre -0,5 et 0,5. Si u est positif, la période est-elle inférieure ou supérieure à 500 ms ?
3. Ouvrir le fichier NCC. L'entrée qui va commander le NCC est une constante qui vaut 0. Nous la modifierons plus tard. L'incrémentation se fait dans une fonction Matlab intégrée à simulink, qui sera appelée *à chaque sample time de l'entrée*. Double cliquer sur la constante et modifier le sample time avec la valeur trouvée au (1).
4. Dans la fonction Matlab, on aimerait incrémenter le compteur avec une ligne du type `i=i+1+u` ; L'ennui, c'est que la valeur de i ne sera pas conservée entre les appels à la fonction, sauf si la variable a été déclarée, en haut du code, de la façon suivante :

```
1          persistent i;
2          if isempty(i)
3          i=0;
4          end
```

   Les lignes 2 à 4 permettent l'initialisation lors du premier appel (ie au temps $t = 0$).

   Compléter le code. La sortie y doit être égale à i (utile pour le debug uniquement). Simuler et valider par l'observation des signaux sur le scope.
5. Idem en modifiant la constante à 0.1, puis -0.1.
6. Ouvrir le fichier `recepteur_qpsk_4_1`. Ce schéma reprend le modèle 3, mais avec un débit de symboles D_s de 2 symboles par seconde et, au niveau de l'ADC, un f_e 100 fois plus élevé, de 200 Hz. Ces valeurs sont initialisées dans les `Callback functions` accessibles en faisant `CTRLE+H`.
   On trouve en bas du modèle le NCC. Deux boutons permettent de valeur de l'entrée u à -0,1 ou 0,1. Par ailleurs, à la sortie du filtre de réception en racine de cos surélevé, on multiplie I par le pulse, ce qui permet d'obtenir l'affichage du signal échantillonné. Plus précisément, le pulse correspondant à l'instant idéal supposé intervient avec $5T_e$ de retard par rapport à la sortie du NCC. Cela ne change rien, puisqu'au départ la position du pulse est aléatoire ; il y aura une utilité dans la suite, pour pouvoir accéder à un échantillon en avance de $5T_e$ par rapport au pulse.



   L'encadré `sélection de l'échantillon idéal` permet lui de ne garder réellement qu'un échantillon par temps symbole (le détail de l'implémentation simulink n'est pas important).

Dernier point : ce modèle s'exécute en temps réel [3].

Lancer le modèle, tout de suite faire un clic droit sur le quadrillage blanc, `Block Parameters`, et mettre un `Time Span` à 4 ou 5 (à refaire dès que vous lancez la simulation car il y a un bug sur cette version de Matlab : on doit l'obliger à refaire la mise à l'échelle à chaque fois au démarrage), et observer le tracé. (A la place de ce quadrillage blanc, vous pouvez utiliser le scope de sortie sur cette version de Matlab).
Sans arrêter la simulation, essayer d'appuyer sur les boutons `its too early !` ou `its too late !` pour déplacer le pulse afin qu'il se trouve à l'instant idéal. N'oubliez pas : il faut chercher les suites 0-1-0 ou 1-0-1. Regarder en même temps la constellation.
Indication : Au départ, le pulse est en avance. Il faut appuyer une dizaine de fois sur `its too early !` pour déplacer le pulse au bon endroit.

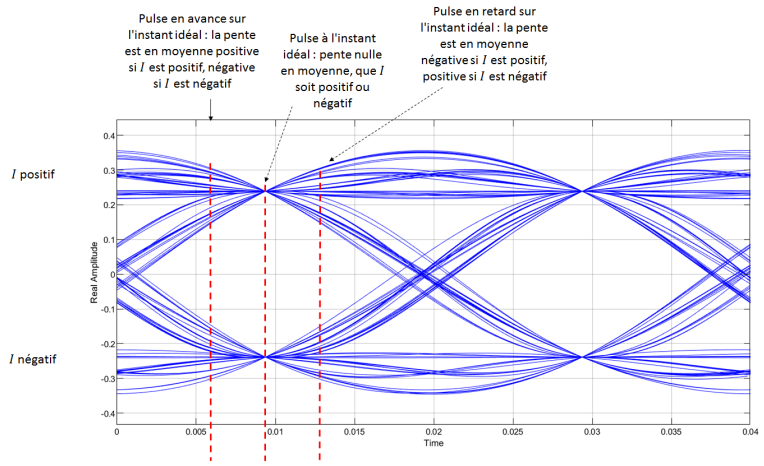
7. Idem en décommentant le délai (dont la valeur change à chaque simulation) entre DAC et ADC ; idem en changeant la `seed` du générateur de bits.

Le TED et le filtre de boucle

A présent, on voudrait que le déplacement du pulse soit automatique. Le composant permettant d'évaluer l'erreur de timing (le décalage entre le pulse et l'instant idéal), pour ensuite pouvoir commander le pulse en conséquence, porte le nom de *Timing Error Detector* (TED).

Ce composant est celui qui a demandé le plus d'ingéniosité. Dans les années 70, plusieurs solutions ont été proposées ; nous allons présenter l'une des plus classiques, le détecteur `early-late` (Lindsey et Simon, 1973).

L'idée de base est que, à l'instant idéal, la pente de I (ou Q, mais on utilise seulement I dans le raisonnement) vaut *en moyenne* 0 :



Si le pulse est en avance sur l'instant idéal, le NCC doit augmenter la période ; pour cela, l'entrée de commande doit être négative. Or, la pente de I est négative si I est négatif, mais positive si I est positif. Si on appelle t_0 l'instant du pulse, la grandeur

$$-\frac{\mathrm{d}I(t_0)}{\mathrm{d}t} \times sgn\left[I(t_0)\right]$$

est donc toujours négative quand le pulse est en avance.

3. Si vous voulez l'exécuter sur votre PC personnel, ceci demande la toolbox `Simulink Real-time Desktop`, puis d'installer le noyau temps réel en tapant sous matlab : `sldrtkernel -install` (et redémarrage).

De même, si le pulse est en retard sur l'instant idéal, le NCC doit diminuer la période ; pour cela, l'entrée de commande doit être positive. La pente de I est positive si I est négatif, mais négative si I est positif. Si on appelle t_0 l'instant du pulse, la grandeur

$$-\frac{\mathrm{d}I(t_0)}{\mathrm{d}t} \times sgn\left[I(t_0)\right]$$

est donc toujours positive quand le pulse est en retard.

Le signal I ayant été numérisé (dans notre cas on a 100 échantillons par temps symbole), on calcule la dérivée en faisant la différence entre deux échantillons. Si k est l'instant courant du pulse, le TED calculera :

$$\left[I(k+5) - I(k-5)\right] \times sgn\left[I(t_0)\right]$$

Autrement dit, on on utilise 3 échantillons : celui du pulse, un échantillon situé juste avant (early sample), ici 5 échantillons avant, et un situé juste après (late sample), d'où le nom du détecteur.

8. D'après ce qui vient d'être dit, à quelle fréquence la sortie de ce détecteur va-t-elle être mise à jour ?
9. Ouvrir `recepteur_qpsk_4_2`. Sous simulink, l'échantillon situé 5 échantillons avant le pulse est celui qu'indique la sortie du NCC. Vérifier sur le modèle que le TED (en bas) est bien le early-late. Ce bloc travaille au rythme des symboles (il met à jour sa sortie 2 fois par seconde). On a ajouté en sortie un bloc `Window Integrator` qui moyenne sur les 10 derniers symboles. Lancer la simulation et observer la sortie sur la jauge, vérifier qu'elle est bien négative quand le pulse est en avance, nulle quand il est bien placé, positive quand il est en retard.

10. Ouvrir `recepteur_qpsk_5`. Cette fois, on ferme la boucle, c'est-à-dire qu'on connecte la sortie du TED et l'entrée du NCC. Entre les deux, un filtre va moyenner les résultats des derniers échantillons, et un gain ajuste la valeur.
    Sur le schéma bloc de la page 9, où mettre le TED et le filtre ?
11. Lancer la simulation et vérifier le fonctionnement [4].

12. Idem en réintroduisant les défauts des questions 5 et 6 : le délai sur le canal, la fréquence d'échantillonnage de l'ADC (`CTRLE+H`).

4 Connaissances, Capacités

Une connaissance est un concept théorique (qu'on pourrait parfois chercher sur Wikipedia). Une capacité est associée à un verbe d'action : il s'agit de savoir faire quelque chose.

S'il reste du temps, vérifier que vous avez acquis les connaissances visées en rédigeant une synthèse du TP (une demi-page à une page).

4. Noter que pour une application réelle, le dimensionnement du filtre est crucial : il détermine la stabilité de la boucle, sa robustesse au bruit, son temps de réponse. Le calcul doit donc se faire de manière rigoureuse ; on fait appel à la théorie des asservissements (hors programme TC).

Connaissance (Co)/ Capacité (Ca)	**Acquise ?**
Co : Condition d'ISI nulle sur la réponse impulsionnelle du filtre de mise en forme. Diagramme de l'oeil : utilité vis à vis de l'ISI, principe de construction	
Co : Rôle du filtre de réception, notion de filtre adapté	
Ca : Exprimer la probabilité d'erreur pour un signal à deux niveaux entaché de bruit	
Co : Exprimer le problème de la récupération du timing symbole : pourquoi une structure de boucle est-elle indispensable ?	
Ca : Savoir coder une fonction Matlab simple	
Co : Rôle du NCC et du TED dans le contexte de la récupération du timing symbole.	