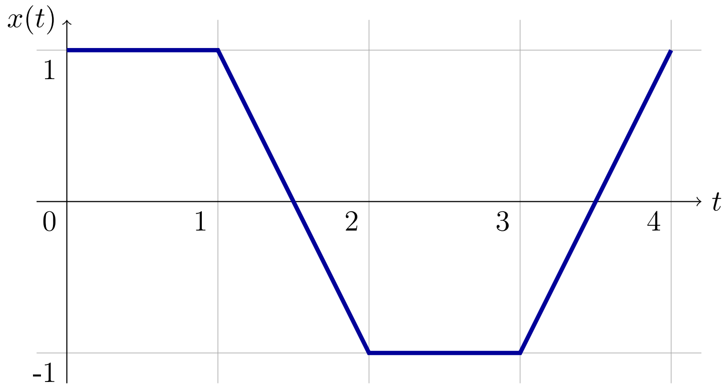
II – Fourier et convolution

1 – Manipulation de signaux

Considérons le signal réel $x(t)$ défini sur l'intervalle $[0, 4]$ dont la représentation est donnée ci-dessous :



On peut décrire ce signal comme une superposition de rampes $R(t) = t\,H(t)$ s'activant successivement :

$$x(t) = 1 - 2\,R(t-1) + 2\,R(t-2) + 2\,R(t-3).$$

MATLAB propose déjà, via sa librairie *Simulink*, des implémentations de l'échelon et de la rampe, mais nous allons travailler dans cette séance avec des implémentations « maison » afin de voir comment on peut manipuler des fonctions définies par l'utilisateur.

Par exemple, pour définir une fonction échelon `H` :

- rendez-vous dans l'éditeur de texte de MATLAB (ou cliquez sur *New function* dans l'interface graphique) et recopiez-y les lignes suivantes :

```
function res = H(t)
  %H Heaviside step function
  res = (sign(t) + 1)/2;              % notez l'utilisation du ;
end
```

- puis enregistrez le tout dans un fichier **au même nom que la fonction** `H.m` (vous le verrez apparaître dans l'explorateur de fichiers) ;

- lorsque le fichier de définition de la fonction est présent dans le répertoire de travail courant, vous pouvez maintenant l'utiliser comme n'importe quelle autre fonction pré-définie :

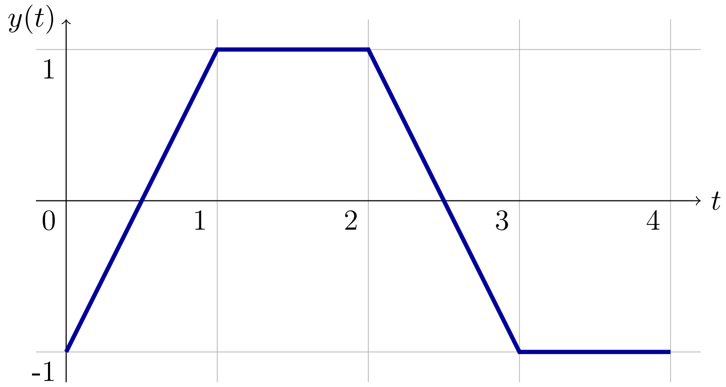
```
H(pi)
H(-sqrt(2))
H(0)        % convention de la valeur moyenne aux discontinuités
help H
```

a) À vous de jouer ! Définissez ainsi dans votre répertoire courant une fonction rampe `R`(t) et vérifiez que celle-ci se comporte bien comme attendu.

b) Vérifiez graphiquement que l'expression donnée ci-dessus décrit bien le signal x :

```
t = 0:.01:4;
t(end) = [];                             % pour éviter les problèmes de recollement
x = 1 - 2*R(t-1) + 2*R(t-2) + 2*R(t-3);
plot(t,x)
```

c) Construire de même le signal y dont la représentation sur $[0, 4]$ est celle-ci :



2 – Séries de Fourier

On calcule numériquement facilement les coefficients de Fourier de x ainsi que les sommes partielles de sa série de Fourier :

$$S_N(t) = \sum_{n=1}^{N} \left(a_n \cos\left(\frac{2\pi n t}{T}\right) + b_n \cos\left(\frac{2\pi n t}{T}\right) \right)$$

(comment sait-on ici qu'il n'y aura pas de terme constant, i.e. que $a_0 = 0$?)

```
N = 1;                                      % nombre d'harmoniques dans la somme partielle
T = 4;                                      % longueur de l'intervalle d'étude
sx = zeros(size(t));                        % somme partielle de série de Fourier

for n = 1:N
    cos_n = cos(2*pi*n*t/T);                % fonctions de base
    sin_n = sin(2*pi*n*t/T);

    a_n = dot(cos_n,x)/dot(cos_n,cos_n);    % coefficients de Fourier
    b_n = dot(sin_n,x)/dot(sin_n,sin_n);

    sx = sx + a_n*cos_n + b_n*sin_n;        % on ajoute la n-ième harmonique
end

plot(t,x,"b")                               % le signal en bleu
hold on
plot(t,sx,"r")                              % la somme partielle de Fourier en rouge
```

a) Observer comment les sommes partielles convergent vers le signal initial lors que le nombre de termes N augmente. Vos observations sont-elles cohérentes avec les résultats vus en cours ? Combien de termes N sont nécessaires pour que l'erreur maximale commise `max(abs(sx - x))` soit inférieure à 0,01 ?

Les fonctions x et y peuvent être vues comme les fonctions coordonnées paramétrant un carré :

```
axis equal
plot(x,y)  % un carré
```

Cela signifie que les sommes partielles `sx` et `sy` des séries de Fourier de x et y paramétrisent des bonnes approximations d'un carré, obtenues géométriquement avec N épicycles (voir par exemple ici ce que l'on obtiendrait en partant de fonctions x et y paramétrant une autre courbe).

b) Superposer sur un même graphe les approximations (`sx`, `sy`) du carré (`x`, `y`) afin d'observer comment celles-ci s'améliorent lorsque N augmente. À partir de quelle valeur de N l'approximation vous semble-t-elle satisfaisante ?

3 – Convolution

On vous fournit la fonction `ct_conv` qui permet de calculer la convolution continue de deux signaux temporels (plus précisément : une *approximation discrète* de la convolution continue de ces signaux).

La syntaxe à utiliser est :

```
[z, tt] = ct_conv(x, y, t);
```

pour obtenir une approximation numérique de la convolution $z = x * y$ de deux signaux `x` et `y` définis sur le même vecteur de temps `t` ; on récupère également un nouveau vecteur d'instants `tt` pour `z` (rappelez-vous l'étalement des supports). On observe donc le résultat avec :

```
plot(tt,z)
```

a) Par exemple, reprenez le calcul de $x(t) * y(t)$ croisé en classe de la fonction $x(t) = 2H(t)\,e^{-t}$ avec

$$y(t) = 3e^{-4(t-1)^2} + 2e^{-2(t-2{,}5)^2}$$

et vérifier graphiquement que x, y et z ont les allures attendues.

b) Définir une fonction porte $P(t) = \Pi_1(t)$ de largeur 1 ainsi qu'une autre, $Q(t) = \Pi_2(t)$, de largeur 2.

Évaluer $P * Q$ numériquement et discuter du résultat obtenu (support, régularité, aire totale, ...).