échantillonage de signal (niveau 2)

Je vous propose ici de coder l'échantillonnage d'un signal et sa reconstruction à partir des échantillons.

5.1 Signal = somme de sinusoides

Pour commencer, définissez un type `Sinuoide` qui regroupe les trois caractéristique d'une sinusoide : amplitude, fréquence et déphasage.

Faites ensuite une fonction `sin_pure` qui calcule la valeur en un point `x` d'une sinusoide telle que définie ci-dessus ; par exemple :

```
double sin_pure(double x, Sinusoide const& s = { 1.0, 1.0, 0.0 });
```

Définissez ensuite un type `Signal` qui regroupe plusieurs sinusoïde pures et une fonction `signal` qui calcule la valeur en un point `x` d'un tel signal, somme de sinusoïde ; par exemple :

```
double signal(double x, Signal const& s);
```

5.2 Signal échantilloné

Définissez maintenant un type `Signal_Echant` qui regroupe :

- une fréquence (d'échantillonnage);
- un temps initial (point de dépar);
- un ensemble d'échantillons (valeurs).

Définissez ensuite une fonction `echantillonne` qui à partir d'un signal, d'une fréquence d'échantillonnage, d'un temps initial et d'un temps final, construit (et retourne) le signal échantillonné. Son prototype pourrait par exemple être :

```
Signal_Echant echantillonne(Signal const& s, double freq,
                            double t_min = 0.0,
                            double t_max = 1.0);
```

Pour construire un tel signal échantillonné, il faut `1+size_t(freq * (t_max - t_min))` échantillons correspondant chacun au signal évalué au temps `t_min + i / freq`.

5.3 Reconstruction

Comme nous ne pouvons pas faire une somme infinie, la formule de reconstruction implémentée ici ne sera qu'une approximation de la formule théorique : nous allons simplement sommer sur tous les échantillons (elle sera donc plus fausse aux bords) :
la valeur reconstruite en un points *x* est la somme sur tous les échantilllons, chacun multiplié par le sinus cardinal de la fréquence d'échantiollanage fois *x* moins le temps initial, le tout moins *i*, le numéro d'échantillon:

```
sinc(freq_ech * (x - t_initial) - i)
```

Nous vons conseillons pour cela, bien sûr, d'écrire une fonction auxiliaire « sinus cardinal ». Afin d'éviter le problème du calcul en 0 : retourner la valeur 1 si l'argument est plus petit que, disons, `1e-8`.

5.4 `main()`

Pour voir/expérimenter les effets du théorème d'échantillonnage, je vous propose dans le `main()` de :

- créer un signal, somme de plusieurs composantes sinusoïdales;
- de définir un `t_min` et un `t_max`, bornes du temps d'observation;
- créer deux versions échantillonnée de ce signal, une avec une fréquence plus grande que le double de sa fréquence maximale et une autre avec une fréquence inférieure à cette limite;
- d'afficher le temps, la valeur du signal et des deux reconstructions (pour chacune des versions échantillonées), pour chaque temps entre `t_min` `t_max` ; par exemple :

```
0 0.189358 0.189358 0.189358
0.002 0.287194 0.23914 0.198669
0.004 0.383332 0.291892 0.208352
... etc.
```

Si vous ne savez pas comment faire tout ça, voici un exemple de `main()` possible :

```
int main()
{
  Signal s({
    { 1.0 , 2.0, 0.0 }, // première composante
    { 0.5 , 4.0, 0.1 }, // ...
    { 0.33, 6.0, 0.2 }, // ...
    { 0.25, 8.0, 0.3 }, // ...
   });

  constexpr double t_min(0.0);
  constexpr double t_max(2.0);

  Signal_Echant se1(echantillonne(s, 20.0, t_min, t_max));
  Signal_Echant se2(echantillonne(s, 11.0, t_min, t_max));

  // « dessin » des trois courbes
  constexpr size_t nb_points(1000);
  constexpr double dt((t_max-t_min) / nb_points);

  for (double t(t_min); t <= t_max; t += dt) {
    cout << t
         << " " << signal(t, s)
         << " " << reconstruction(t, se1)
         << " " << reconstruction(t, se2)
      << endl;
  }

  return 0;
}
```

5.5 Déboguer

Pour vous aider, si nécessaire, à corriger votre programme, expliquons d'où vient la seconde ligne donnée ci-dessus :

```
0.002 0.287194 0.23914 0.198669
```

Ces quatre nombres représentent respectivement, le temps, le signal d'origine, le signal échantillonné à la première fréquence (20 Hz) puis reconstruit et le signal échantillonné à la seconde fréquence (11 Hz) puis reconstruit, provenant de la ligne du `main()` :

```
    cout << t
         << " " << signal(t, s)
         << " " << reconstruction(t, se1)
         << " " << reconstruction(t, se2)
      << endl;
```

Le signal calculé ici est
$\sin(4\pi t)+0.5\sin(8\pi t+0.1)+0.33\sin(12\pi t+0.2)+0.25\sin(16\pi t+0.3)$,
ce qui, pour t=0.02 donne :
0.0251301 + 0.0748503 + 0.089737 + 0.0974768 = 0.287194.

Le signal échantilloné à 20 Hz puis reconstruit est calculé comme la somme des 41 termes suivants :

```
 0.189358  * sinc(20 * (0.002 - 0) -  0 ) =  0.189358  *  0.99737     =  0.18886
 1.44431   * sinc(20 * (0.002 - 0) -  1 ) =  1.44431   *  0.0415571   =  0.0600213
 0.75564   * sinc(20 * (0.002 - 0) -  2 ) =  0.75564   * -0.0203545   = -0.0153807
 0.731161  * sinc(20 * (0.002 - 0) -  3 ) =  0.731161  *  0.013478    =  0.00985457
 0.257756  * sinc(20 * (0.002 - 0) -  4 ) =  0.257756  * -0.0100744   = -0.00259675
 0.0582359 * sinc(20 * (0.002 - 0) -  5 ) =  0.0582359 *  0.00804331  =  0.000468409
-0.305928  * sinc(20 * (0.002 - 0) -  6 ) = -0.305928  * -0.00669376  =  0.00204781
-0.660188  * sinc(20 * (0.002 - 0) -  7 ) = -0.660188  *  0.00573201  = -0.0037842
-0.896827  * sinc(20 * (0.002 - 0) -  8 ) = -0.896827  * -0.00501191  =  0.00449481
-1.57352   * sinc(20 * (0.002 - 0) -  9 ) = -1.57352   *  0.00445255  = -0.00700616
 0.189358  * sinc(20 * (0.002 - 0) - 10 ) =  0.189358  * -0.0040055   = -0.000758473
 1.44431   * sinc(20 * (0.002 - 0) - 11 ) =  1.44431   *  0.00364004  =  0.00525734
 0.75564   * sinc(20 * (0.002 - 0) - 12 ) =  0.75564   * -0.00333569  = -0.00252058
```

```
 0.731161  * sinc(20 * (0.002 - 0) - 13 ) =  0.731161  *  0.0030783   =  0.00225073
 0.257756  * sinc(20 * (0.002 - 0) - 14 ) =  0.257756  * -0.00285779  = -0.000736615
 0.0582359 * sinc(20 * (0.002 - 0) - 15 ) =  0.0582359 *  0.00266677  =  0.000155301
-0.305928  * sinc(20 * (0.002 - 0) - 16 ) = -0.305928  * -0.00249967  =  0.00076472
-0.660188  * sinc(20 * (0.002 - 0) - 17 ) = -0.660188  *  0.00235229  = -0.00155295
-0.896827  * sinc(20 * (0.002 - 0) - 18 ) = -0.896827  * -0.00222131  =  0.00199213
-1.57352   * sinc(20 * (0.002 - 0) - 19 ) = -1.57352   *  0.00210416  = -0.00331093
 0.189358  * sinc(20 * (0.002 - 0) - 20 ) =  0.189358  * -0.00199874  = -0.000378476
 1.44431   * sinc(20 * (0.002 - 0) - 21 ) =  1.44431   *  0.00190338  =  0.00274907
 0.75564   * sinc(20 * (0.002 - 0) - 22 ) =  0.75564   * -0.0018167   = -0.00137277
 0.731161  * sinc(20 * (0.002 - 0) - 23 ) =  0.731161  *  0.00173758  =  0.00127045
 0.257756  * sinc(20 * (0.002 - 0) - 24 ) =  0.257756  * -0.00166506  = -0.000429179
 0.0582359 * sinc(20 * (0.002 - 0) - 25 ) =  0.0582359 *  0.00159835  =  9.30813e-05
-0.305928  * sinc(20 * (0.002 - 0) - 26 ) = -0.305928  * -0.00153678  =  0.000470144
-0.660188  * sinc(20 * (0.002 - 0) - 27 ) = -0.660188  *  0.00147978  = -0.000976931
-0.896827  * sinc(20 * (0.002 - 0) - 28 ) = -0.896827  * -0.00142685  =  0.00127964
-1.57352   * sinc(20 * (0.002 - 0) - 29 ) = -1.57352   *  0.00137758  = -0.00216765
 0.189358  * sinc(20 * (0.002 - 0) - 30 ) =  0.189358  * -0.0013316   = -0.000252149
 1.44431   * sinc(20 * (0.002 - 0) - 31 ) =  1.44431   *  0.00128859  =  0.00186113
 0.75564   * sinc(20 * (0.002 - 0) - 32 ) =  0.75564   * -0.00124827  = -0.000943246
 0.731161  * sinc(20 * (0.002 - 0) - 33 ) =  0.731161  *  0.0012104   =  0.000884998
 0.257756  * sinc(20 * (0.002 - 0) - 34 ) =  0.257756  * -0.00117476  = -0.000302802
 0.0582359 * sinc(20 * (0.002 - 0) - 35 ) =  0.0582359 *  0.00114116  =  6.64562e-05
-0.305928  * sinc(20 * (0.002 - 0) - 36 ) = -0.305928  * -0.00110942  =  0.000339403
-0.660188  * sinc(20 * (0.002 - 0) - 37 ) = -0.660188  *  0.0010794   = -0.00071261
-0.896827  * sinc(20 * (0.002 - 0) - 38 ) = -0.896827  * -0.00105097  =  0.000942538
-1.57352   * sinc(20 * (0.002 - 0) - 39 ) = -1.57352   *  0.00102399  = -0.00161127
 0.189358  * sinc(20 * (0.002 - 0) - 40 ) =  0.189358  * -0.000998369 = -0.000189049
```

ce qui donne la valeur 0.23914.

Le signal échantilloné à 11 Hz puis reconstruit est calculé comme la somme des 23 termes suivants :

```
 0.189358  * sinc(11 * (0.002 - 0) -  0 ) =  0.189358  *  0.999204   =  0.189207
 0.851989  * sinc(11 * (0.002 - 0) -  1 ) =  0.851989  *  0.022477   =  0.0191502
 0.482616  * sinc(11 * (0.002 - 0) -  2 ) =  0.482616  * -0.0111135  = -0.00536355
-0.0101872 * sinc(11 * (0.002 - 0) -  3 ) = -0.0101872 *  0.00738163 = -7.51984e-05
-0.643095  * sinc(11 * (0.002 - 0) -  4 ) = -0.643095  * -0.00552602 =  0.00355375
-1.53079   * sinc(11 * (0.002 - 0) -  5 ) = -1.53079   *  0.00441593 = -0.00675985
 1.45684   * sinc(11 * (0.002 - 0) -  6 ) =  1.45684   * -0.00367723 = -0.00535713
 0.72608   * sinc(11 * (0.002 - 0) -  7 ) =  0.72608   *  0.00315026 =  0.00228734
 0.0696877 * sinc(11 * (0.002 - 0) -  8 ) =  0.0696877 * -0.00275539 = -0.000192017
-0.528292  * sinc(11 * (0.002 - 0) -  9 ) = -0.528292  *  0.00244848 = -0.00129351
-1.06421   * sinc(11 * (0.002 - 0) - 10 ) = -1.06421   * -0.0022031  =  0.00234455
 0.189358  * sinc(11 * (0.002 - 0) - 11 ) =  0.189358  *  0.00200241 =  0.000379172
 0.851989  * sinc(11 * (0.002 - 0) - 12 ) =  0.851989  * -0.00183524 = -0.0015636
 0.482616  * sinc(11 * (0.002 - 0) - 13 ) =  0.482616  *  0.00169383 =  0.000817468
-0.0101872 * sinc(11 * (0.002 - 0) - 14 ) = -0.0101872 * -0.00157265 =  1.60209e-05
-0.643095  * sinc(11 * (0.002 - 0) - 15 ) = -0.643095  *  0.00146765 = -0.00094384
-1.53079   * sinc(11 * (0.002 - 0) - 16 ) = -1.53079   * -0.0013758  =  0.00210605
 1.45684   * sinc(11 * (0.002 - 0) - 17 ) =  1.45684   *  0.00129476 =  0.00188626
 0.72608   * sinc(11 * (0.002 - 0) - 18 ) =  0.72608   * -0.00122274 = -0.000887809
 0.0696877 * sinc(11 * (0.002 - 0) - 19 ) =  0.0696877 *  0.00115831 =  8.07203e-05
-0.528292  * sinc(11 * (0.002 - 0) - 20 ) = -0.528292  * -0.00110033 =  0.000581298
-1.06421   * sinc(11 * (0.002 - 0) - 21 ) = -1.06421   *  0.00104788 = -0.00111516
 0.189358  * sinc(11 * (0.002 - 0) - 22 ) =  0.189358  * -0.0010002  = -0.000189396
```

ce qui donne la valeur 0.198669.

5.6 Dessiner les résultats

Le plus simple pour visualiser ce que votre programme a sorti est d'utiliser un outil de dessin externe, comme par exemple `gnuplot`, qui est un programme (à lancer depuis la ligne de commande) de dessin de données et de fonctions.
Vous pouvez aussi, naturellement, utiliser n'importe quel autre logiciel capable de dessiner des courbes à partir de points, comme des tableurs (LibreOffice, ou autres) ou Matplotlib, Octave, etc.

`gnuplot` lit des fichiers dont chaque ligne contient un point à dessiner, les coordonnées étant par colonnes, exactement comme le sort votre programme. Pour que la sortie de votre programme soit stockée dans un fichier, il suffit de le lancer depuis un terminal et « rediriger sa sortie » dans un fichier avec le signe `>`. Cela se fait simplement comme

ceci:

```
./signal > data.txt
```

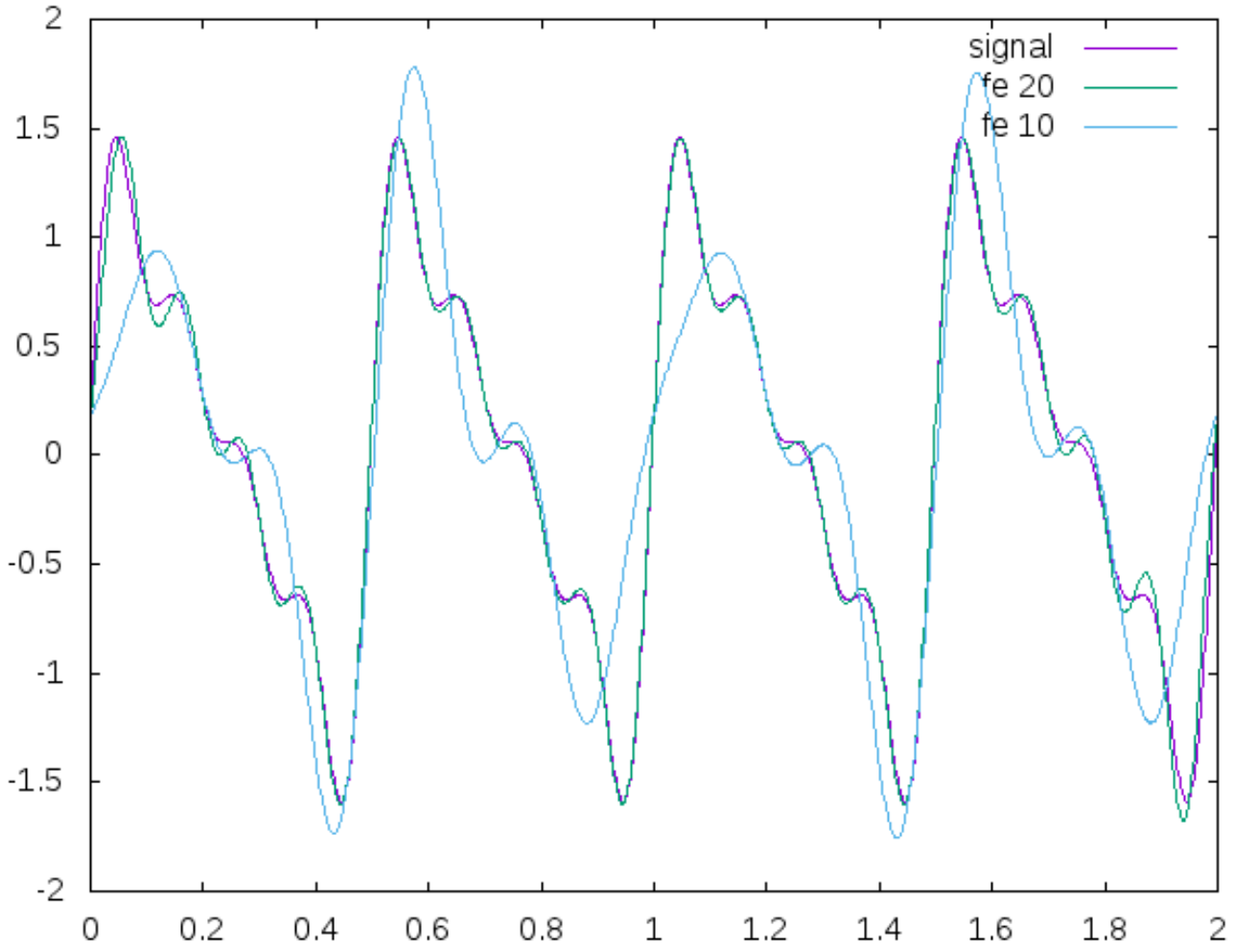
C'est aussi simple que cela ! Avec ça, vous devriez avoir, dans le répertoire courant, un fichier `data.txt` qui contient :

```
0 0.189358 0.189358 0.189358
0.002 0.287194 0.23914 0.198669
0.004 0.383332 0.291892 0.208352
0.006 0.477329 0.347325 0.218412
0.008 0.568761 0.405119 0.228854
0.01 0.657223 0.464923 0.23968
... etc.
```

Une fois que vous avez un tel fichier (bon pour `gnuplot`), il suffit de lancer la commande `gnuplot` dans le terminal, puis, dans `gnuplot`, de taper

```
set style data line
plot "data.txt" u 1:2 t "signal", "data.txt" u 1:3 t "fe 20",
     "data.txt" u 1:4 t "fe 10"
```

et vous devriez alors voir ceci (pour le `main()` donné plus haut) :



NOTE : les écarts aux bords (en début et fin) entre le signal d'origine et le premier signal reconstruit (correctement échantilloné) ne sont simplement du qu'au fait que nous ne faisons pas une somme infinie mais simplement une somme finie dans la formule de reconstruction. Il faut donc regarder « au milieu » pour voir que la reconstruction est assez bonne (parfaite en théorie avec sommes infinies).

Pour quitter `gnuplot` : simplement `quit` ou '`Control-D`'.

Annexe : explication des commandes `gnuplot` utilisées

Le « `u 1:3` » signifie que l'on utilise la 1àre et la 3e colonne pour dessiner. Sinon `gnuplot` utilise les deux premières (i.e. il fait un « `u 1:2` »).

Le « `t "..."` » est pour donner un titre à une courbe (légende sur le coté en haut à droite).

La commande « `set style data line` » demande à `gnuplot` d'utiliser simplement des lignes pour dessiner les données; sinon par défaut `gnuplot` utilise des « points » (petits symoles, en fait) pour dessiner les fichiers de données.