Hes·so // VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

Filière Systèmes industriels

Orientation Power and Control

Diplôme 2009

Anthony Veuthey

Modélisation d'un réseau basse tension : Simuler dynamiquement la « qualité de l'électricité »

Professeur	Gilbert-André Morand
Expert	Marcel Maurer

Sion, le 6 juillet 2009

Données du travail de diplôme
Daten der Diplomarbeit

☒ FSI ☐ FTV	Année académique **/** *Studienjahr* **2008/09**	No TD / *Nr. DA* **pc/2009/43**
Mandant **/** *Auftraggeber* ☒ HES—SO Valais ☐ Industrie ☐ Ecole hôte	Etudiant **/** *Student* **Anthony Veuthey**	Lieu d'exécution / *Ausführungsort* ☒ HES—SO Valais ☐ Industrie ☐ Ecole hôte
Professeur / *Dozent* **Gilbert-André Morand**	Expert / *Experte (données complètes)* **M. Marcel Maurer**	
Travail confidentiel / *vertrauliche Arbeit* ☐ oui / ja [1] ☒ non / nein		

Titre / *Titel*

Modèlisation d'un réseau basse tension :

Simuler dynamiquement la « qualité de l'électricité »

Description et Objectifs / *Beschreibung und Ziele*

L'objectif est de simuler dynamiquement à l'aide de programme comme Matlab les comportements liés au raccordement de plusieurs onduleurs sur un réseau de distribution local basse tension.

Une ligne de transmission, un onduleur, ou un consommateur seront modélisés par une fonction de transfert. Des effets physiques comme les phénomènes de battement ou de réflexion d'ondes mobiles sur les lignes de transmission pourront être étudiés.

Signature ou visa / *Unterschrift oder Visum*	Délais / *Termine*
Resp. de la filière *Leiter des Studieng.:*	Attribution du thème / *Ausgabe des Auftrags:* 16.02.2009
[1]Etudiant/Student:	Remise du rapport / *Abgabe des Schlussberichts:* 06.07.2009, 12:00
	Exposition publique / *Ausstellung Diplomarbeiten:* 04.09.2009
	Défense orale / *Mündliche Verfechtung:* Semaine / Woche 35

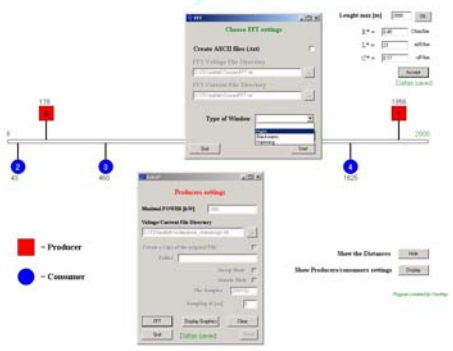
[1] *Par sa signature, l'étudiant-e s'engage à respecter strictement le caractère confidentiel du travail de diplôme qui lui est confié et des informations mises à sa disposition.*
Durch seine Unterschrift verpflichtet sich der Student, die Vertraulichkeit der Diplomarbeit und der dafür zur Verfügung gestellten Informationen zu wahren.

Rapport reçu le / *Schlussbericht erhalten am ……* Visa du secrétariat / *Visum des Sekretariats ……*

Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

Modélisation d'un réseau basse tension

Diplômant Anthony Veuthey



Objectif du projet

L'objectif est de simuler dynamiquement à l'aide de programme comme Matlab les comportements liés au raccordement de plusieurs onduleurs sur un réseau de distribution local basse tension.

Méthodes | Expériences | Résultats

Une ligne de transmission, un onduleur, un consommateur seront modélisés par une fonction de transfert. Des effets physiques comme les phénomènes de battement ou de réflexion d'ondes mobiles sur les lignes de transmission pourront être étudiés.
L'idée est de concevoir un logiciel informatique capable d'effectuer ; des acquisitions de mesures, des calculs d'impédance, des affichages graphiques et des bilans de puissance pour des minis-réseaux comprenant des producteurs et des consommateurs.
L'avantage de ce programme est qu'il ne travaille pas uniquement à 50Hz mais sur une plage de fréquence comprise entre 2kHz et 100kHz.
Les objectifs fixés ont été atteint. Il reste à ajouter la partie « Gestion du réseau » selon un modèle précis de type NEPLAN.

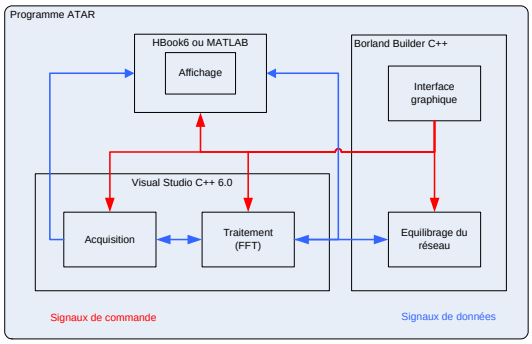
Travail de diplôme
| édition 2009 |

Filière
Systèmes industriels

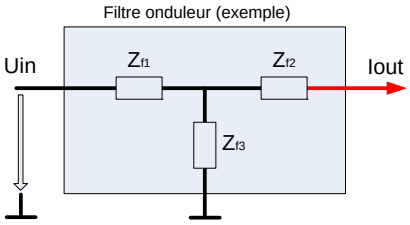
Domaine d'application
Power & Control

Professeur responsable
M. Gilbert-André Morand
Gilbert.Morand@hevs.ch

Expert
M. Marcel Maurer



Voici l'architecture globale du programme ATAR. L'affichage se fait avec le logiciel Hbook6, le traitement des signaux sous Visual Studio et l'interface graphique avec Borland Builder C++.



Ce schéma bloc représente le modèle généralisé du filtre de sortie d'un onduleur.

HES-SO Valais
Route du Rawyl 47
1950 Sion

Tél. 027 606 85 11
URL www.hevs.ch

Table des matières

1 Introduction

1.1 Projet IGOR

Le projet IGOR (Interférences Générées par les Onduleurs sur le Réseau) est une étude sur les risques liés au raccordement de plusieurs onduleurs sur un réseau de distribution local basse tension. Nous savons que les onduleurs injectent dans le réseau du courant non-sinusoïdal. Celui-ci possède une fondamentale à 50Hz et une série d'harmoniques de faible amplitude autour de la fréquence de commutation des MOS (sortes d'interrupteurs qui hachent le signal DC d'entrée afin de créer un signal AC en sortie). Ce courant est filtré à la sortie de l'onduleur afin de diminuer l'amplitude de ces harmoniques voire de les supprimer. Les filtres sont dimensionnés afin que le taux d'harmoniques généré par l'onduleur ne dépasse pas les normes en vigueur. Le tableau ci-dessous définit l'amplitude maximale des harmoniques en fonction du type de charge. La Classe-A défini les appareils électroménagers. Les onduleurs sont contrôlés afin qu'ils respectent cette norme.

Harmonic n	Class-A *[A]*	Class-B *[A]*	Class-C *% of I_1*	Class-D *[ma/W]*
2	1.08	1.62	2	
3	2.30	3.45	30*PF	3.4
4	0.43	0.65		
5	1.44	2.16	10	1.9
6	0.30	0.45		
7	0.77	1.12	7	1
8	0.23	0.35		
9	0.40	0.60	5	0.5
10	0.18	0.28		
11	0.33	0.50	3	0.35
12	0.15	0.23		
13	0.21	0.32	3	0.296
14 – 40 (even)	1.84/n	2.76/n		
15 – 39 (odd)	2.25/n	3.338/n	3	3.85/n

Figure 1 : Tableau tiré de la norme IEC 61000-3-2

Nous remarquons que la norme donne des valeurs jusqu'à la 40ème harmonique (2kHz) puis elle reprend à partir de la 3000ème harmonique (150kHz). Nous n'avons aucune information sur les harmoniques présentes en dehors des normes et cela peut être une source de perturbations pour un réseau car la fréquence de commutation des MOS se situe entre 15 et 25 kHz. Les battements générés par des onduleurs sur une même ligne sont une autre source de problème. Etant donné que l'horloge interne de chaque onduleur est différente, la fréquence de la tension de l'harmonique dû à la commutation des MOS n'est pas exactement la même que celle d'un second onduleur. Cette faible différence de fréquence entre les deux signaux génère un battement à une fréquence élevé sur la fondamentale du réseau. L'amplitude de ce battement atteint le double de la valeur de la tension initiale de l'harmonique. Le projet IGOR tente de déterminer si ces perturbations peuvent avoir des effets importants sur un réseau basse tension.

1.2 Objectifs

1.2.1 Approche mathématique avant tout

Le principal objectif est d'utiliser des outils ou algorithmes mathématique que nous comprenons et maîtrisons totalement. Nous ne voulons à aucun moment durant ce projet devoir faire confiance à une fonction mathématique que nous ne connaissons pas. Pour cela il faut prendre beaucoup de temps à l'étude et à la compréhension de celles-ci.

1.2.2 But final

Fournir un outil qui peut aider à réaliser le projet IGOR est le but que nous cherchons à atteindre. L'idée est de concevoir un logiciel informatique capable d'analyser des minis-réseaux comprenant des producteurs et des consommateurs, avec comme objectif de connaître les perturbateurs et le type de perturbations qu'ils provoquent. Ce logiciel travaillera sur toutes les fréquences et pas seulement sur le 50Hz. Obtenir une base de données fournie en information sur différents types de consommateurs et producteurs est également un but à atteindre.

1.3 Cahier des charges

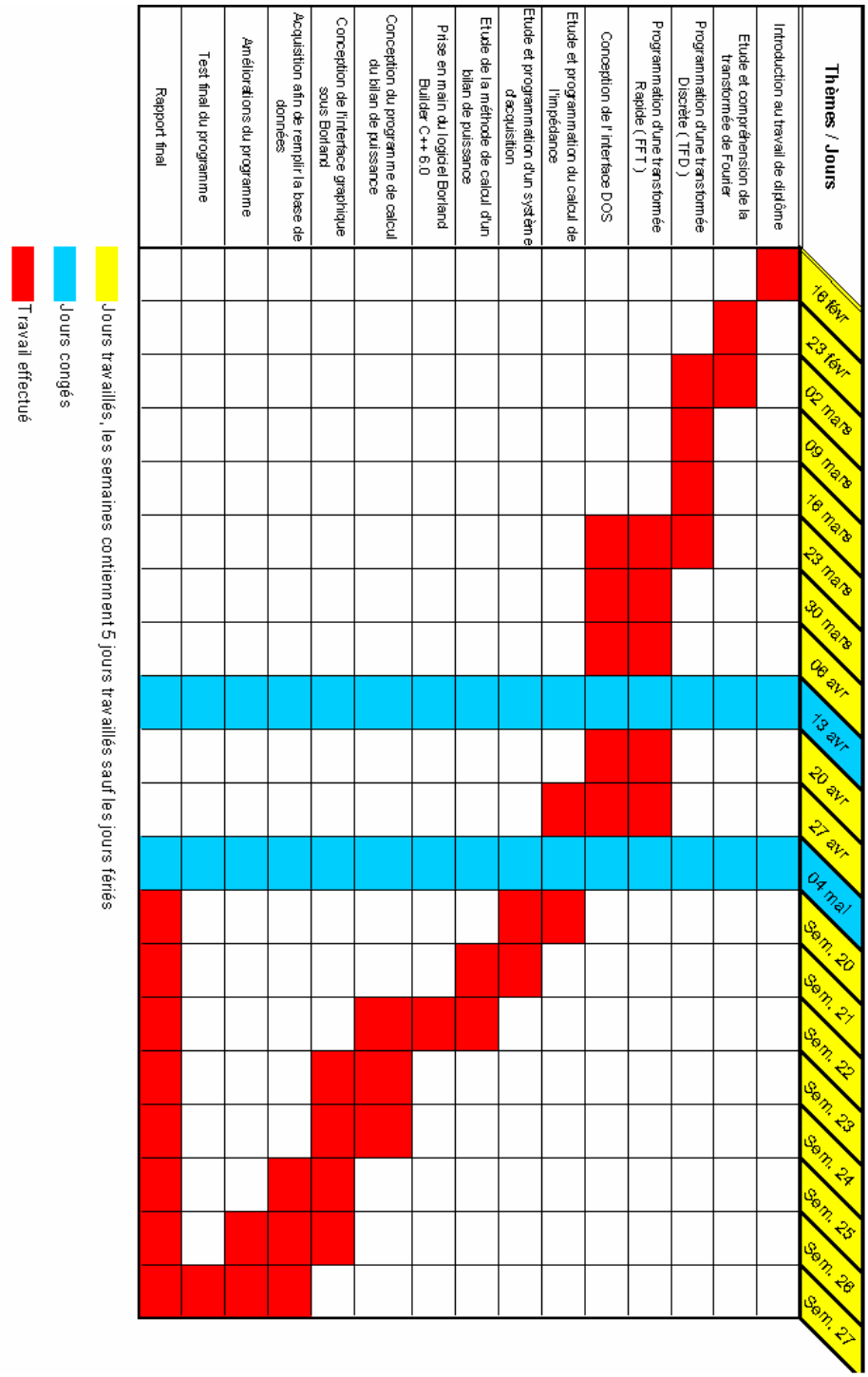
Notre projet est un « projet pilote », le cahier des charges ne comporte que peu voire aucune contrainte.

Les étapes à suivre sont les suivantes :

- Etudier les Transformées de Fourier et Laplace
- Programmer les Transformées
- Développer un système d'acquisition de mesures
- Effectuer un traitement de signaux
- Programmer un calcul d'impédance à partir des mesures précédentes
- Programmer un bilan de puissance
- Afficher les résultats

1.3.1 Planning

Nous avons débuté notre travail le 16 février 2009 à raison d'une journée par semaine. A partir du 11 mai de la même année et jusqu'au 06 juillet, date de la remise du rapport final, nous sommes à plein temps sur le travail (5 jours/semaine).



2 Développement théorique

2.1 Les Transformées de Fourier et de Laplace

2.1.1 Transformée de Laplace

Il va falloir, pour ce projet, travailler avec des fonctions de transfert. Pour passer d'un système temporelle à un système fréquentielle nous pouvons utiliser soit la transformée de Laplace soit la transformée de Fourier. L'équation de la transformée de Laplace est la suivante :

$$F(p) = \mathcal{L}\{f(t)\} = \int_0^{+\infty} e^{-pt} f(t)\, dt.$$

Figure 3 : Laplace avec t en [s]

La notation « p », pour la variable de Laplace, est préférée au « s » dans certains pays. La transformée de Laplace est utilisé pour résoudre des équations différentielles, tandis que celle de Fourier pour l'analyse de signaux. Dans notre cas nous aurons besoin de la transformée de Fourier mais la différence entre les deux est-elle si grande ? Les deux différences sont les suivantes :

- Le « s ou p » de Laplace est égal à « iω » pour Fourier
- La borne d'intégration inférieur de Laplace = 0 tandis que celle de Fourier = $-\infty$

On peut en conclure que les deux transformées sont quasiment identiques.

2.1.2 TFD (Transformée de Fourier Discrète)

Afin de comprendre comment fonctionne une Transformée de Fourier, il est nécessaire de créer sa propre Transformée de Fourier Discrète, que nous nommerons « TFD » par la suite.
La formule mathématique de la TFD est la suivante :

$$\mathcal{F}(f) : \nu \mapsto \hat{f}(\nu) = \int_{-\infty}^{+\infty} f(t)\, e^{-i2\pi\nu t}\, dt$$

Figure 4 : Fourier avec t en [s] et ν en [Hz]

La variable f(t) représente le signal à analyser. En résumé ce signal est une addition de plusieurs fonctions sinusoïdales et cosinusoïdales. F(f) contient le module et la phase du signal en fonction de la fréquence, sous forme complexe.

À l'intérieur de l'intégrale nous avons une ou plusieurs multiplications de fonctions sinusoïdales et cosinusoïdales. Ces calculs peuvent facilement se faire sur Excel, voir chapitre 3.1.1.
L'intégrale est une somme de ces produits. Théoriquement les bornes d'intégrations qui vont de -∞ à +∞ ne sont pas un problème, mais pratiquement il est difficile de pouvoir intégrer de -∞ à +∞. Les signaux mesurés possèdent forcément un départ et une fin. Il y a également la période d'échantillonnage, que nous nommerons « δt » par la suite, qui, théoriquement, est infiniment petit. Pour notre projet il n'est pas nécessaire d'atteindre l'infiniment petit, une valeur proche de la microseconde suffit à avoir une résolution convenable. Le calcul exact du δt pour notre projet est fait au chapitre 3.1.1.

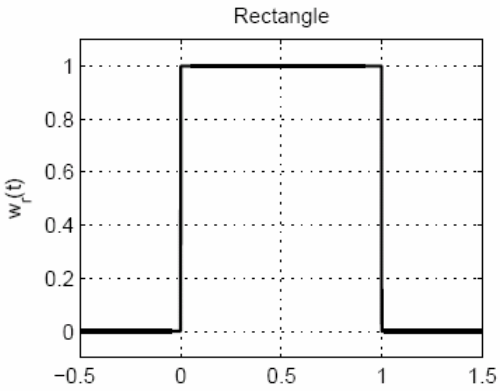
2.1.3 Fenêtrage

Comment s'y prendre afin de corriger ce problème de bornes d'intégration ? Un professeur en ingénierie électrique, du nom de Frédéric J. Harris, à extensivement étudiées des fenêtres d'observation utilisées en analyse spectrale afin de résoudre le problème qui nous préoccupe. Il a même donné son nom à un type de fenêtre. Les résultats de ces recherches sont parus en 1978 dans l'ouvrage « *On the use of windows for harmonic analysis with DFT* ».

Nous nous contentons ici de mentionner 4 fenêtres fréquemment appliquées à l'enregistrement d'un signal. Elles sont définies comme suit :

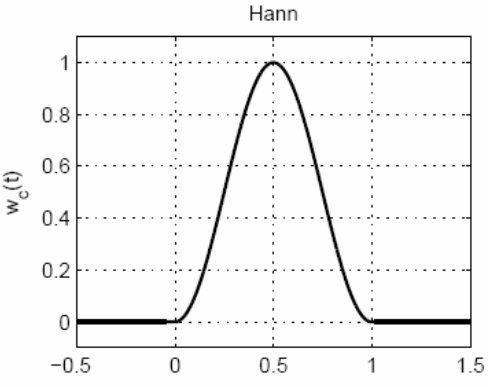
Fenêtre rectangulaire :

$$w_r[n] = 1 \quad \text{pour} \quad 0 \leq n < N$$



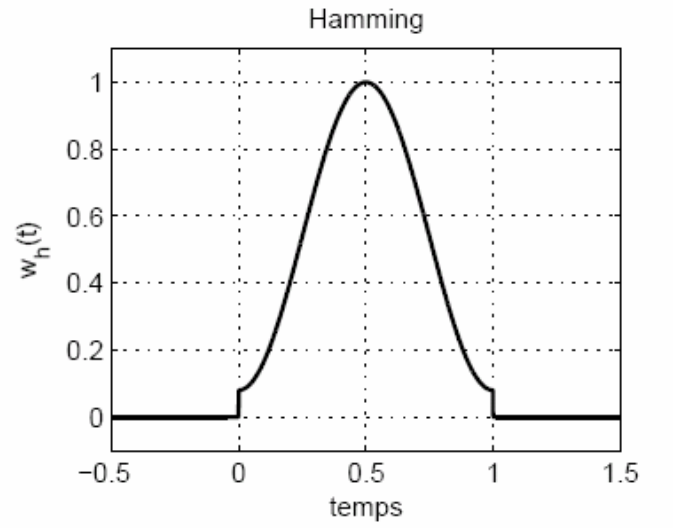
Fenêtre de Hann :

$$h(t) = \begin{cases} 0.5 - 0.5\cos 2\pi\frac{t}{T} & \text{si } t \in [0, T] \\ 0 & \text{sinon.} \end{cases}$$



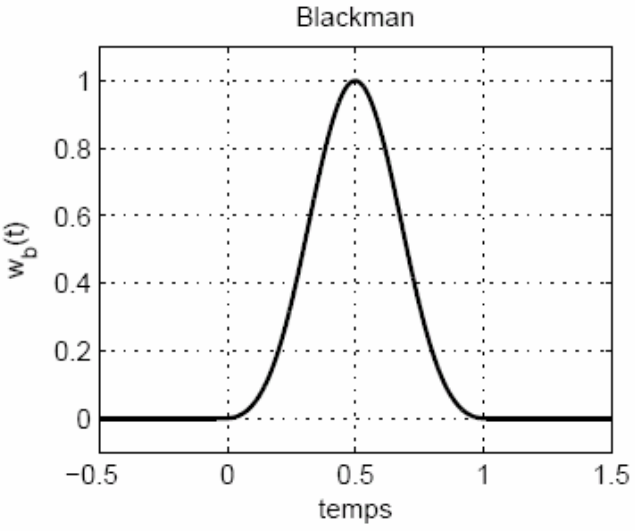
Fenêtre de Hamming :

$$w_h[n] = 0.54 - 0.46\ \cos\left(2\pi\frac{n}{N}\right) \quad \text{pour} \quad 0 \leq n < N$$



Fenêtre de Blackman-Harris :

$$w_b[n] = 0.42 - 0.5\ \cos\left(2\pi\frac{n}{N}\right) + 0.08\ \cos\left(4\pi\frac{n}{N}\right) \quad \text{pour} \quad 0 \leq n < N$$



3 Réalisation

Les bases des Transformées de Fourier et Laplace sont acquises. Néanmoins nous avons décidé de programmer une TFD sous Excel, afin de comprendre le mieux possible comment cet algorithme fonctionne.

3.1 Programmation de l'algorithme de la TFD

3.1.1 Sous EXCEL

Excel n'est pas très adapté pour faire des boucles répétitives (boucle FOR par exemple). Nous devons choisir une plage pour l'analyse fréquentielle assez restreinte : 100Hz, 200Hz maximum. Il faut également limiter la durée du signal mesuré à 10secondes maximum et le dt à 0.1ms minimum. Tout cela afin de ne pas dépasser les capacités du tableur.
Néanmoins cela permet de tester le programme et de bien comprendre tout ce qui a été fait entre la mesure du signal et l'affichage de la transformée.

3.1.1.1 Résultats

Voici comment fonctionne le programme :

En premier lieu, entrer l'amplitude et la fréquence de 2 sinus (la fréquence doit être comprise en 1 et 100Hz) ensuite à l'aide de la macro « ctrl+a », le programme calcul et construit lui-même le graphique de la FFT de la somme de ces deux sinusoïdes. En voici un exemple :

Pour deux sinusoïdes (10V, 50 Hz et 5V , 75Hz) simulés sur 4 secondes avec une fenêtre de Hann qui est exactement de même durée, nous obtenons la courbe suivante :

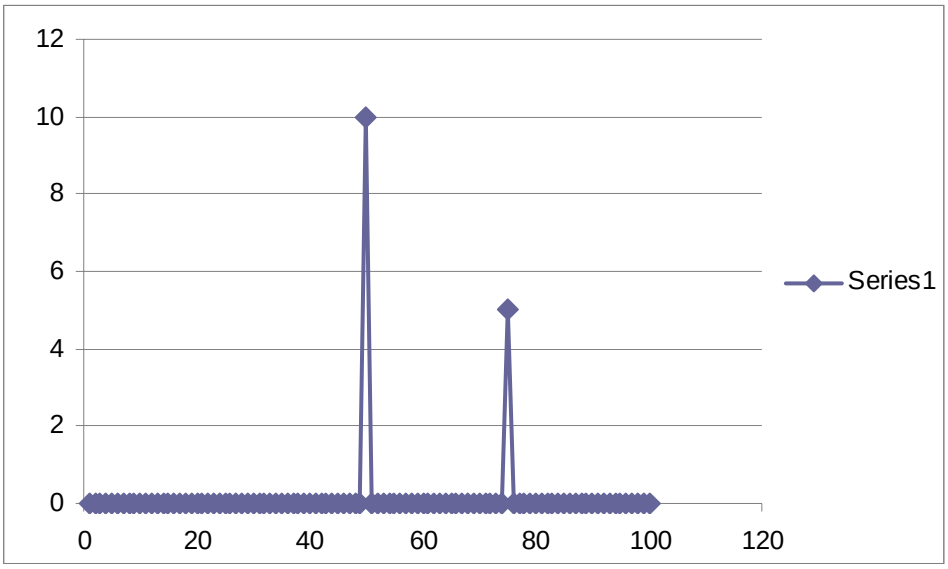


Figure 5 : TFD de 2 sinus sous Excel

Nous remarquons effectivement : une sinusoïde à 50 Hz avec une amplitude 10V et une seconde à 75 Hz avec une amplitude de 5V. Cependant nous nous apercevons rapidement que ce mode de calcul est long et totalement inadapté au projet en question.

Le fichier Excel se trouve sur le CD contenant l'ensemble du travail de diplôme dans le répertoire *TD* sous le nom : *fourier.xls*.

Pour fixer le δt définitif, il faut connaître la fréquence du signal le plus rapide que nous voulons mesurer. Pour l'ensemble du projet IGOR nous allons mesurer des signaux allant jusqu'à 100kHz. Ce qui correspond à une période de 10µs. Hors Harry Nyquist et Claude Shannon ont énoncé que la fréquence d'échantillonnage d'un signal doit être égale ou supérieur au double de la fréquence maximale contenue dans ce signal, afin de convertir ce signal d'une forme analogique à une forme numérique. Souvent, la fréquence d'échantillonnage corrigée par le théorème de Nyquist-Shannon, ne porte le nom que d'un des deux protagonistes, comme le montre l'équation ci-dessous.

$$f_{Nyquist} = \frac{f_e}{2}$$

En conclusion, pour mesurer un signal à 100kHz, il faut échantillonner deux fois plus rapidement, soit à 200KHz. Le δt correspondant vaut **5µs**.

Un point important à respecter est l'incrément fréquentiel Δf. La relation qui nous aide à calculer cette valeur est la suivante : $\Delta f = \frac{1}{N \cdot \Delta t}$ (Référence : Théorie sur les éléments d'analyse spectrale numérique par Freddy Mudry, chapitre 5.2 page 158).

N est le nombre total de points et Δt la période d'échantillonnage. Cette relation est utile lorsque nous travaillons avec la FFT.

Excel est vite dépassé par le nombre impressionnant de point à calculer (un nombre qui se chiffre en millions). Il faut se tourner sur des programmes plus évolués du type de Matlab ou Visual Studio C++.

3.1.2 Sous MATLAB

Nous avons décidé de retranscrire toutes les opérations de l'algorithme de la TFD sur Excel, en langage Matlab. Arrivé à la fin du programme, celui-ci est beaucoup trop lent. Nous comptons 1 minute pour une TFD qui travaille sur une plage de fréquence de 100 Hz et un δt de 5 µs. Alors que le projet IGOR demande une plage de fréquence de 98kHz pour le même δt. Il est possible d'augmenter la vitesse de traitement du programme en limitant le nombre de calculs de fonctions sinusoïdales et cosinusoïdales à 4000 au lieu de 1 million. Comme ces fonctions sont « deux*pi » périodiques, nous avons inscrit 4000 résultats de ces fonctions dans un vecteur. Le premier résultat correspond à cos(0) et le 4000ème à cos(2*pi), nous avons une résolution de 1/4000. Il suffit, par la suite, de parcourir ce vecteur pour obtenir un résultat à la place d'effectuer un calcul de fonction sinusoïdale qui prend plus de temps au processeur. La valeur 4000 n'est pas obligatoire, il faut la choisir en fonction de nos besoins. Si nous voulons une plus grande précision de calcul, il faut choisir une valeur plus grande au détriment du temps de traitement qui sera plus long.

3.1.2.1 Résultats

La courbe ci-dessous a été réalisée avec exactement les mêmes conditions que celles sous Excel (voire chapitre 3.1).

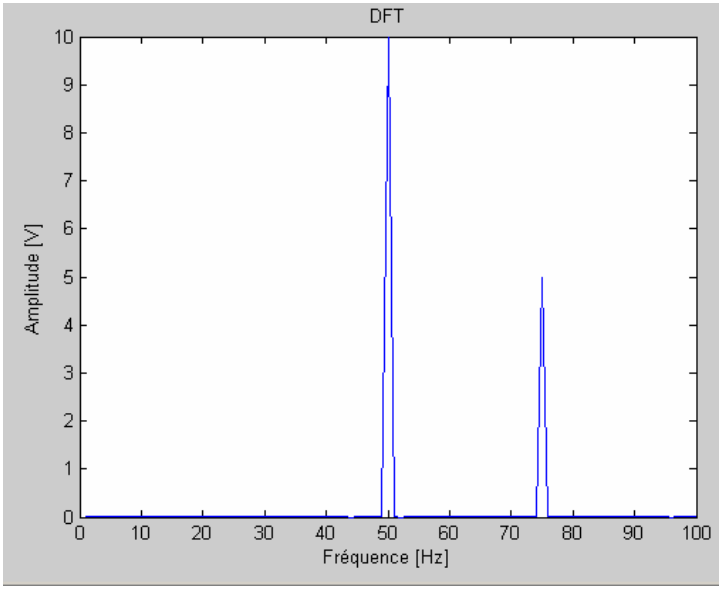


Figure 6 : TFD de 2 sinus sous Matlab

Le résultat est identique. L'avantage de travailler sur MATLAB est que l'on peut plus aisément modifier la plage de fréquence ou la période d'échantillonnage. Par contre, dès que le nombre de points à calculer dépasse les 100'000, le traitement se fait de plus en plus long. Malgré l'amélioration apportée avec la diminution du nombre de calcul de fonctions sinusoïdales et cosinusoïdales, le temps de traitement est trop long (étonnamment, il n'y a eu aucun changement par rapport à avant) et il faut chercher une autre solution. Le code du programme MATLAB se trouve sur le CD dans le répertoire *matlab* sous le nom *my_fft.m* avec les fonctions *monsin.m* et *moncos.m* qui y sont rattachées.

Nous voulons passer tout le code Matlab en langage C++ afin de créer un fichier exécutable. Ceci permettrait de gagner en vitesse du traitement grâce au compilateur de Visual Studio. Effectivement l'exécutable est environ 1000 fois plus rapide que le fichier Matlab. Toutefois, cela ne suffit pas et nous devons trouver une solution afin de rendre le programme encore 100 à 1000 fois plus rapide.

Le programme en C++ étant identique à celui en langage MATLAB, inutile de donner plus d'information le concernant. Le code du programme de la TFD en C++ est en *annexe 8*.

3.2 FFT (Fast Fourier Transform)

3.2.1 Etude

Après une étude assez poussée du programme, nous n'avons pas trouvé de solutions afin de rendre le calcul de la TFD plus rapide. Nous nous sommes tournés vers un autre algorithme qui n'est nul autre que la Transformée de Fourier Rapide, que nous nommerons FFT par la suite. Il est important de signaler que la FFT n'est pas une nouvelle transformation. Ce n'est rien d'autre qu'un moyen plus rapide d'obtenir **le même résultat** que la TFD. Evidemment il existe plusieurs algorithmes capables de calculer la FFT d'un signal, nous avons choisi de nous intéresser à celui de Cooley-Tukey qui publièrent leur méthode en 1965. Il a été démontré par la suite que c'est Carl Friedrich Gauss, en 1805, qui a inventé cet algorithme.

Cet algorithme utilise le fait que la TFD globale peut être décomposée en TFD de séquence de plus en plus courte. C'est une méthode **récursive**. Néanmoins cet algorithme impose que le nombre de points analysés soit une puissance de 2.

Si nous regardons le nombre d'opérations arithmétiques (sommes et produits) de la TFD nous obtenons : N proportionnel à N^2. Par contre en ce qui concerne la FFT nous obtenons $N \approx N \cdot \log_2(N)$.
Si nous prenons notre exemple :

- Echantillonnage 5µs
- Temps d'acquisition 10secondes

Ce qui fait $N = \frac{T}{\Delta t} = \frac{10}{5 \cdot 10^{-6}} = 2'000'000$, mais comme N doit être une puissance de 2 il faut prendre N = 2^{21}= 2'097'152 points. Ce qui fait un temps d'acquisition de 10.48576 secondes. Maintenant si nous faisons le rapport entre le TFD et la FFT cela nous donne : $\frac{N^2}{N \cdot \log_2(N)} = \frac{N}{\log_2(N)} = \frac{2'097'152}{21} \approx 100'000$

En résumé, pour N environ égal à 2 millions de points, la FFT est **100'000 fois plus rapide** que la TFD !!!

Faut-il créer notre FFT sous MATLAB ou sous Visual studio C++ ? Il existe une fonction sous MATLAB qui se nomme « FFT » mais impossible de trouver le code source qui la compose. La programmation de la FFT se fera sous Visual C++, car le code source de l'algorithme se trouve sur le net.

3.2.2 Programmation sous VISUAL STUDIO C++

Nous avons récupéré le code source de la FFT sur le net et l'avons adapté à notre programme en créant une interface homme-machine. Le résultat est sans appel. En 30 secondes, le programme est capable de calculer et de remplir deux fichier .txt de 2millions de points chacun. Le premier contient le signal de 10secondes échantillonné à 5us, ce signal correspond au signal mesuré. Le deuxième contient les résultats de la FFT. Ces résultats sont sous la forme de 2 colonnes de 2 millions de valeurs. Une pour la fréquence, l'autre pour l'amplitude. Matlab contient un affichage graphique évolué. Il est intéressant de pouvoir visualiser les résultats de la FFT sur ce logiciel.
Le code de la FFT se trouve en *annexe 8-9*.

3.2.3 Résultats

Nous avons testé notre programmation en simulant un signal qui contient une somme de 6 fonctions sinusoïdales. Les paramètres des ces 6 fonctions sont ci-dessous :

```
// je charge ici ma fonction ( tension )

i=0;   A[i]=   230.0;     f[i] =   50.0;          w[i]=2 * PI * f[i];
i=1;   A[i]=   120.0;     f[i] =   60.0;          w[i]=2 * PI * f[i];
i=2;   A[i]=   70.0;      f[i] =   100.0;         w[i]=2 * PI * f[i];
i=3;   A[i]=   50.0;      f[i] =   200.0;         w[i]=2 * PI * f[i];
i=4;   A[i]=   20.0;      f[i] =   5000.0;        w[i]=2 * PI * f[i];
i=5;   A[i]=   10.0;      f[i] =   50000.0;       w[i]=2 * PI * f[i];
iAMax = 6;     // Détermine le nombre de sinus que contient le signal
```

A[i] représente l'amplitude de la fonction sinusoïdale en [V] ou [A].
f[i] représente la fréquence en [Hz] et w[i] la pulsation en [rad/s].

La partie du code qui génère ce signal est la suivante :

```
// ------------------------------------------------------------------------
//  Remplit le vecteur xr avec une somme de signaux déjà définis
//

int FaireFonctionAAnalyserU(long N,float dt, float A_random,int iMax,float A[],float w[],float xr[])
{
  FILE*file3=NULL;

  file3=fopen("D:\\TD\\matlab\\signalU.txt","wt");

  int i,j;
  float t;
  float xsum;
  float x=0.0;

 // fprintf(file3,"dt\txr\n\n");

  for(i=0;i<N;i++){
      xsum = 0.0;
      t = (float)i*dt;
      for(j=0;j<iMax;j++) {
         xsum += A[j] * sin (w[j] * t);
      }
//    x= random(0,10)/10.0;
      xr[i] = xsum;//+x ;

      fprintf(file3,"%lf\t%lf\n",t,xr[i]);
  }
  fclose(file3);
  return(0);
}
```

Les paramètres de la FFT sont :

- δt : 5us
- Fenêtrage : Hann

Le premier graphique ci-dessous représente le signal en fonction du temps et le second représente sa FFT, c'est-à-dire le même signal mais en fonction de la fréquence :

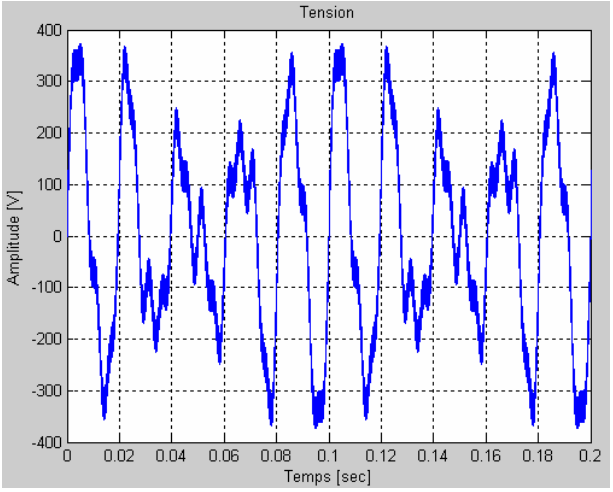


Figure 7 : Signal de tension

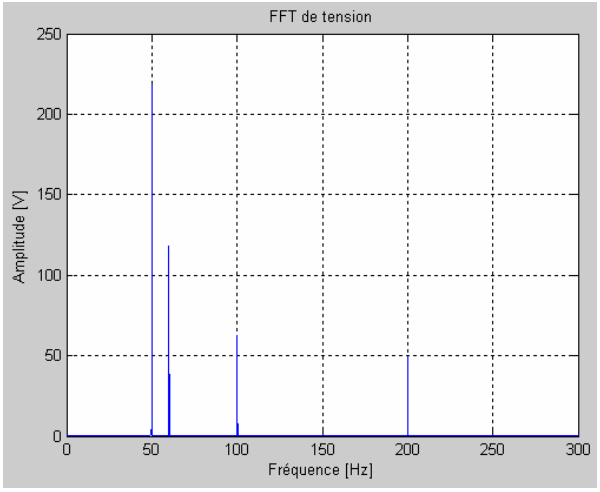
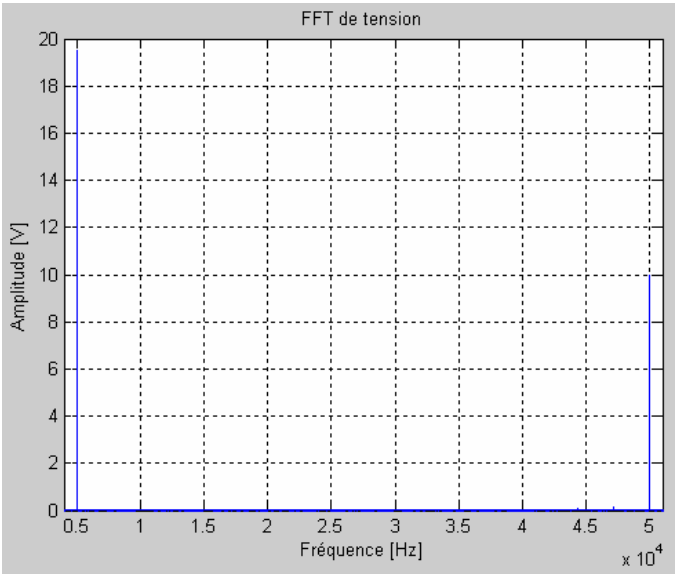


Figure 8 : FFT de tension

La courbe de la FFT de tension contient uniquement les 4 premières fonctions sinusoïdales qui composent notre signal. Les deux autres fonctions sont à haute fréquence, et sont représentées sur le graphique ci-dessous:



Sur cette courbe on aperçoit les deux dernières fonctions sinusoïdales qui composent notre signal. La première est à 5kHz/20V et la seconde à 50kHz/10 V.

La courbe de la FFT a été séparée en deux afin de pouvoir analyser les pics de fréquence (nous nommons comme cela les fonctions sinusoïdales représentées en fonction de la fréquence) avec plus de précisions car la plage de fréquence entre le 1er et le 6ème pic est très grande : 50kHz.

Nous nous apercevons que la FFT calcule **juste** et **rapidement**.
Pour réaliser toute ces opérations c'est-à-dire :

- Calculer le signal
- Calculer la FFT du signal
- Copier les valeurs dans un fichier

Le programme a mis exactement 30 secondes.

3.3 Traitement de FFT en continu

Nous devons étudier et réfléchir à un système pratique qui utiliserait notre FFT afin de fournier au projet IGOR un outil performant. Nous pensons qu'en mesurant presque en continue un réseau quelconque, il est possible d'observer une grande partie des harmoniques qui circulent sur ce réseau en fonction des divers fournisseurs et consommateurs qui y sont connectés. A l'aide d'une approche statistique, nous pouvons par la suite déterminer quel producteur ou quel consommateur est la source de ces harmoniques.

Le diagramme de flux suivant représente le principe de fonctionnement du programme qui effectue le traitement des signaux après acquisition de ceux-ci.

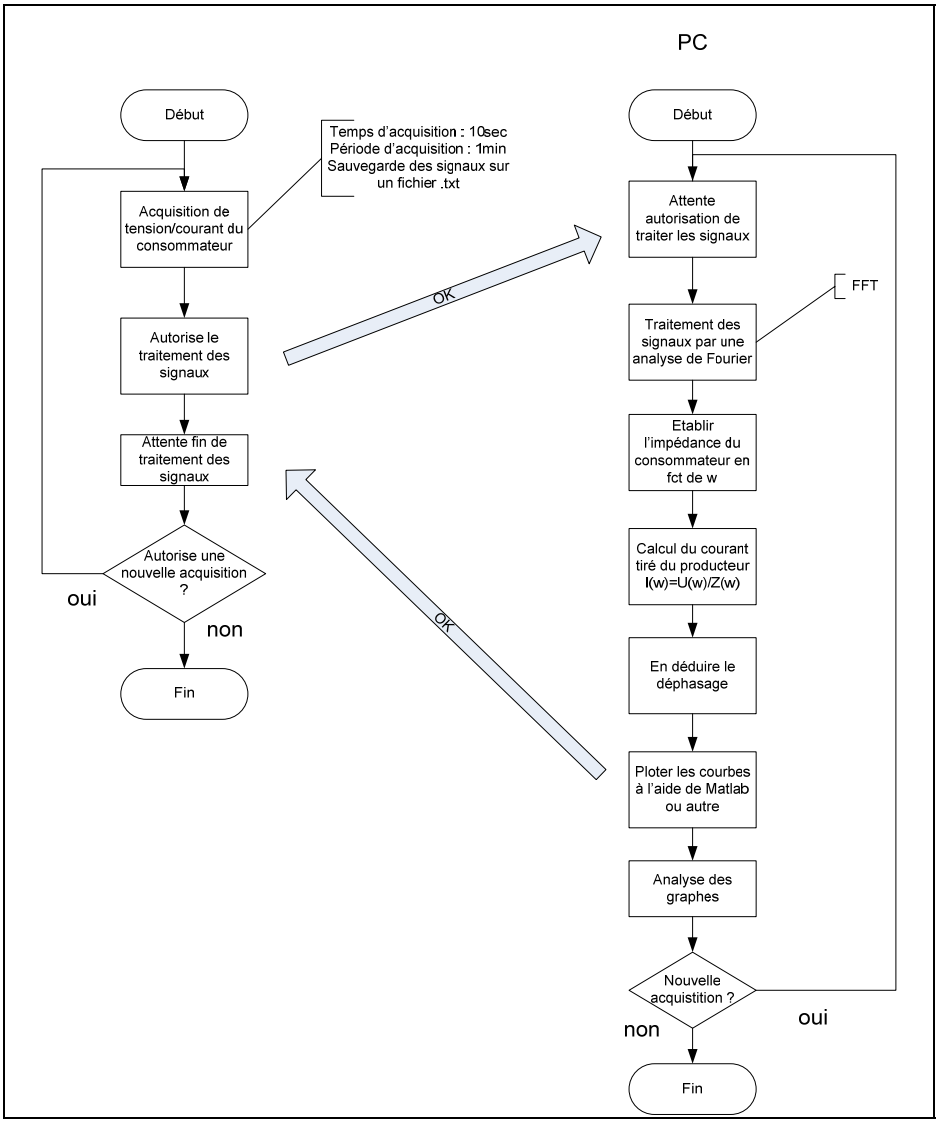


Figure 9 : Diagramme de flux

Ce système est abandonné pour un autre que nous jugeons moins aléatoire. La précision d'un système qui se base sur des statistiques n'est pas assez grande pour le projet IGOR.

3.4 Création d'un modèle de réseau

Ce modèle contient des producteurs, des consommateurs et une ligne horizontale qui représente le réseau. Nous utilisons un programme informatique pour interagir avec tous les éléments de ce modèle. La figure suivante représente l'aspect général du programme que nous allons créer.

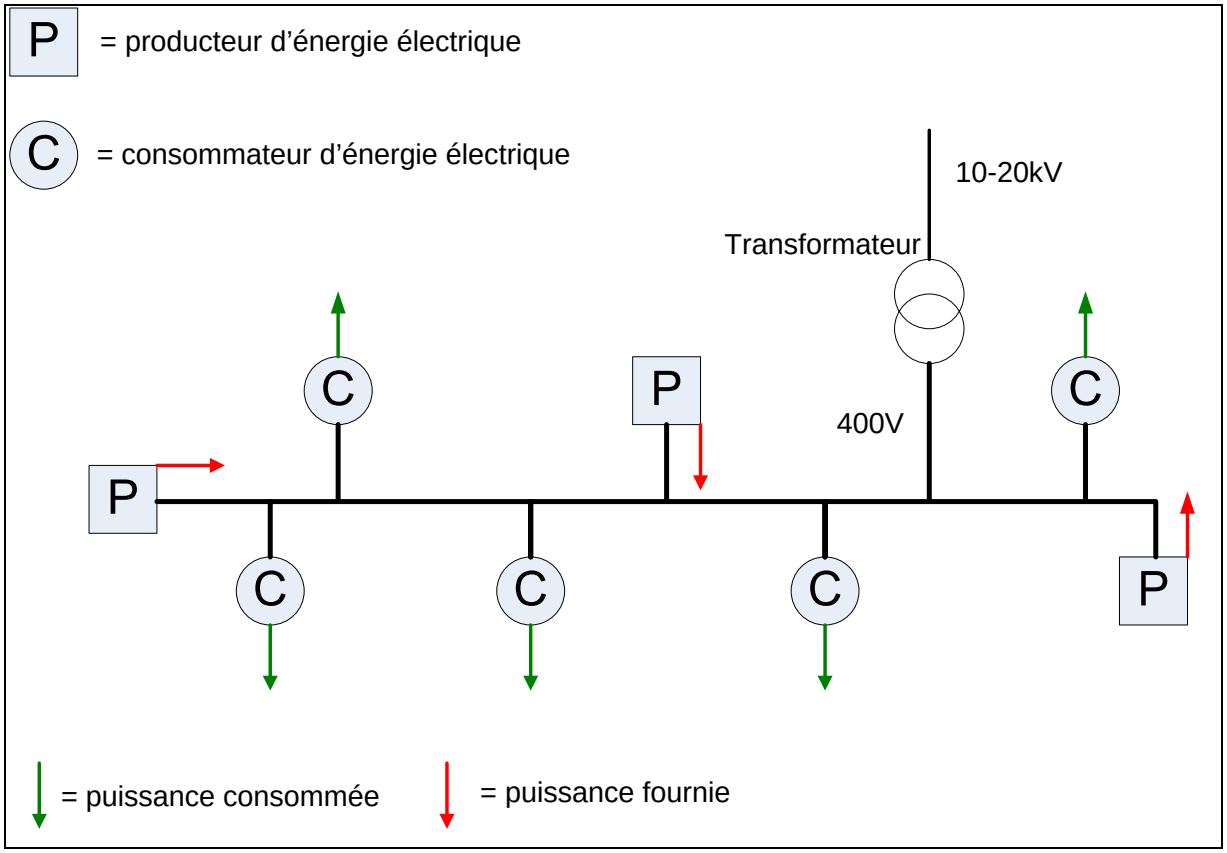


Figure 10 : Schéma de principe du programme

3.4.1 Cahier des charges du modèle

Nous imaginons un système contenant les fonctionnalités suivantes : sauvegarde des fichiers acquis, traitement des données, affichage graphique, équilibrage du réseau.
Chaque producteur est modélisé par une tension de sortie et une puissance en fonction de la fréquence. Chaque consommateur est modélisé par une impédance en fonction de la fréquence. Nous devons également prendre en compte l'impédance de ligne qui est définie par une résistance, une inductance et une capacité. Les tensions de sortie des producteurs et les diverses impédances sont obtenues par des mesures. Nous devons remplir une base de données avec ces mesures afin d'en effectuer le traitement par la suite. Nous pouvons également déplacer, ajouter, modifier ou supprimer les producteurs/consommateurs comme nous le souhaitons.

Nous choisissons, pour débuter, de nous intéresser à la base de données et surtout au moyen de la remplir.

3.4.2 Acquisition des données

Le premier problème est de trouver par quel moyen acquérir toutes ces données. Un fichier texte contenant 2 millions de points « pèse » environ 50 Mo de mémoire et rares sont les oscilloscopes capables d'enregistrer ce type de fichier sur leurs disques dures. Voici les solutions envisagées :

3.4.2.1 L'analyseur de spectre

Les analyseurs de spectre LeCroy, une grande marque américaine qui se situe dans les plus grands fournisseurs d'oscilloscopes sur le plan mondial, sont extrêmement performants vis-à-vis de leurs concurrents. La HES-SO Valais de Sion possède quelques uns de ces analyseurs et nous autorise à les utiliser. L'avantage de posséder Windows comme système d'exploitation n'en n'est pas vraiment un. Malgré le fait que cet analyseur peut recevoir le programme d'acquisition sur son OS, il est très lent et son système plante quelques fois. Néanmoins il possède un disque dur interne de 40Go ce qui permet une réserve de mémoire suffisante à notre base de donnée. Sa capacité d'échantillonnage est un autre point positif, car il est capable d'échantillonner à 4us sur un temps de 10sec ce qui est idéal pour notre projet. Le point négatif se situe au niveau de la résolution de celui-ci : 12bits sur l'échelle verticale, cela veut dire $2^{12} = 4096$ points. Si nous mesurons une tension de 400V, l'analyseur prend un point chaque 1/10 de volt. Ce n'est pas suffisant pour notre projet.

Figure 11 : Analyseur de spectre LeCroy 6050A

3.4.2.2 La carte d'acquisition

L'avantage de la carte d'acquisition est au niveau de la résolution de celle-ci : 16bits sur l'échelle verticale contre 12 pour l'oscilloscope. Le reste des données techniques (guide d'installation carte + programme) de la carte se trouvent dans un carton écrit *Carte d'acquisition*. La carte est facilement programmable avec LabVIEW. Il est également fournit avec la carte du code en C++ permettant le lancement de certains types d'acquisition.

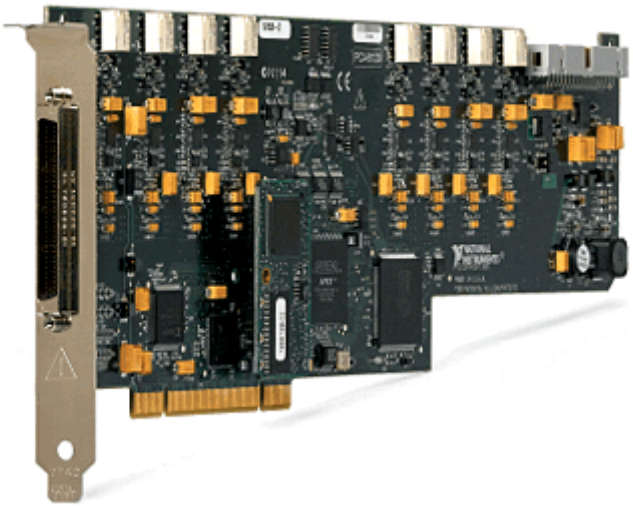


Figure 12 : Carte NI PCI-6122

Cette carte nécessite un bloc de connexion et un câble pour les relier. Le guide d'installation du bloc se trouve avec le guide d'installation de la carte. Le type du bloc choisit pour le projet est un BNC-2110 et le câble est un SH-68-68-EP (2m) de 68 pins.

Figure 13 : Bloc BNC-2110

Figure 14 : Câble SH-68-68-EP

Nous avons décidé de commander une carte d'acquisition, car ces caractéristiques sont meilleures par rapport à l'analyseur LeCroy. Toutefois la carte doit mettre un certain temps avant d'être en notre possession, c'est pourquoi les premières mesures sont faites à l'aide de l'analyseur de spectre. Un analyseur de spectre est un oscilloscope spécialisé dans l'analyse fréquentielle.

3.4.2.3 Marche à suivre

3.4.2.3.1 Analyseur de spectre

Il faut programmer l'analyseur afin qu'il échantillonne son signal mesuré à 5 µs ou moins (voir explication chapitre 3.1.1.1). La durée du signal et le nombre d'échantillons déterminent la période d'échantillonnage. Un signal de durée de 10 secondes contenant 2.5 millions d'échantillons donne un δt de 4 µs.

3.4.2.3.2 Carte d'acquisition

Un programme nommé LabView SignalExpress est fourni avec la carte d'acquisition. Nous avons la possibilité de choisir nous même le nombre d'échantillons ainsi que le δt avec ce programme. Il se trouve sur le CD d'installation de la carte d'acquisition dans le carton écrit *Carte d'acquisition.*

3.4.3 Traitement des données

La méthode d'acquisition est trouvée. Nous pouvons dès à présent remplir notre base de données avec des mesures sur des producteurs et des consommateurs. L'étape suivante consiste à traiter ces mesures afin de pouvoir les utiliser.

3.4.3.1 Traitement par FFT

Nous avons besoin d'effectuer une FFT sur toutes les mesures que contient notre base de données afin de passer d'un domaine temporelle à un domaine fréquentielle. Notre fonction FFT, qui a été testée au chapitre 3.2.3, sert de base solide à la réalisation de notre programme. C'est sur cette base que viendra ce greffer le reste des autres fonctions. Le code de cette FFT se trouve en *annexe 8-9*.

3.4.3.2 Lecture des données

Notre programme doit pouvoir lire et interpréter les différents fichiers crées soit par l'analyseur soit par la carte d'acquisition. Dans tous les cas notre programme doit faire en sorte que le nombre d'échantillons du signal mesuré soit identique au nombre d'échantillons de la FFT de ce signal. Le nombre d'échantillons de la FFT doit être une puissance de 2 (voir chapitre 3.2.1). Si le fichier acquis possède un nombre d'échantillons entre deux puissances de 2, le programme doit stopper la lecture de ce fichier lorsque le nombre d'échantillons lus atteint la plus grande puissance de 2 possible. Prenons un exemple :

Le fichier acquis possède 2.5 millions d'échantillons. A quel nombre d'échantillons, le programme doit-il stopper sa lecture ? 2^{21}=2'097'152 et 2^{22}=4'194'304 , le programme doit s'arrêter à la ligne 2'097'152.

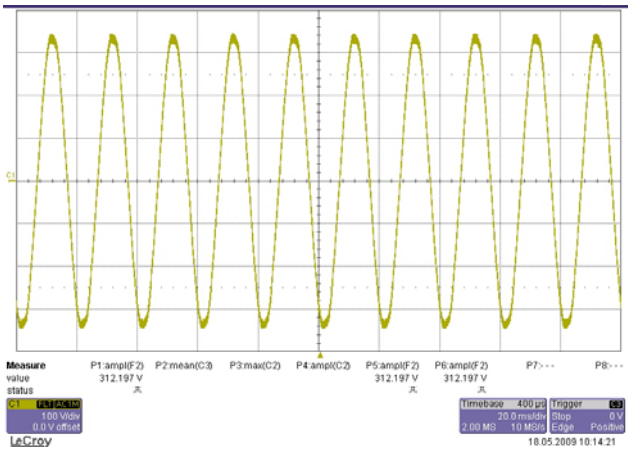
Voici la partie du code qui réalise cette lecture de fichier :

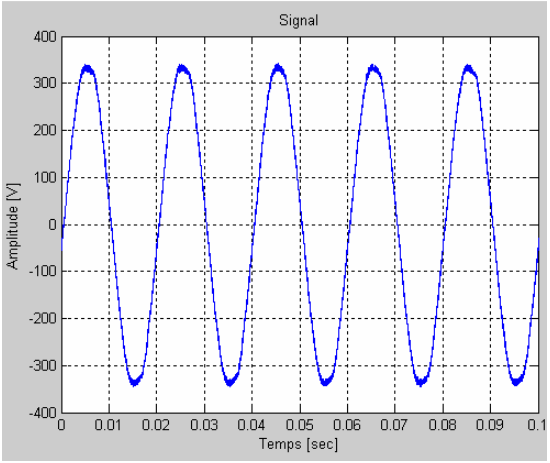
```
int LireSignalMesure(char *filename,char filew[100],float *t,float *xr,int N)
{
   int error = 0;
   char ligne[200];
//   float x;
   int i=0;
   float temps;
   float x;
   FILE *fileWrite=NULL;
   FILE *streamFile = fopen (filename, "rt" );
   fileWrite = fopen(filew,"wt");
   if(streamFile == NULL) return(-1);              // si le fichier à lire n'existe pas, on quitte
   for(;;)
   {
        fgets(ligne,200,streamFile);
        if(feof(streamFile)) break;
        if(i>=N) break;                            // dès que le programme a lu N lignes il stop la lecture
        /*
        if(i==0) {                                 // sauter en souplesse la premiere ligne de commentaire
            i++;
            continue;
        }
        */
        sscanf(ligne,"%f\t%f",&temps,&x);          // scan le fichier ligne après ligne

        if (temps < 0 )                            // supprime les temps négatifs et les entêtes du fichier
        {
            continue;
        }
        else
        {
            xr[i] = x;
            t[i]=temps;
            fprintf(fileWrite,"%lf\t%lf\n",t[i],xr[i]); // fait une copie du fichier
            i++;
        }
   }
   return(0);
   fclose (streamFile);
   fclose (fileWrite);
}
```

3.4.3.2.1 Résultats

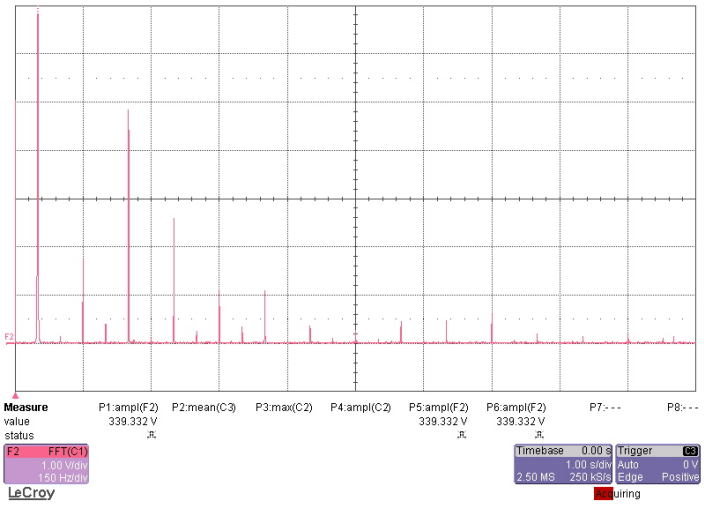
Voici le signal mesuré. Il s'agit de la tension du réseau « perturbé » de l'école. Nous nous apercevons que le sinus n'est pas parfait. Nous allons obtenir quelques harmoniques de tension lors de la FFT.

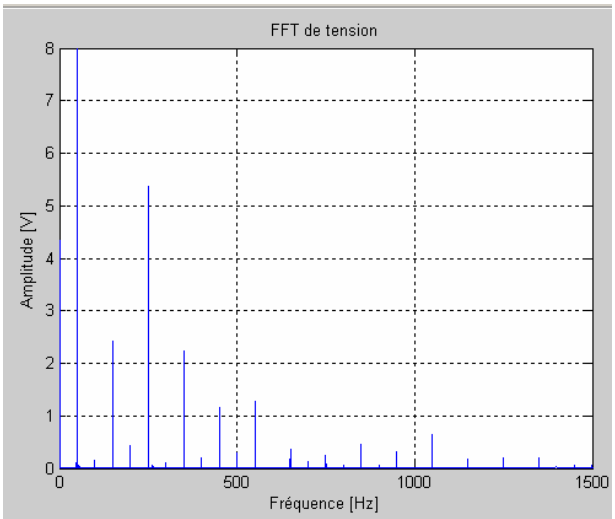




Cette courbe représente la même tension de réseau qu'auparavant, mais cette fois-ci le signal a été lu et réinterprété par le programme. L'affichage a été fait sous MATLAB.

La tension possède quelques harmoniques de très faible amplitude : 5V maximum pour la 5ème harmonique (250Hz). L'échelle verticale est de 1V/div, c'est pourquoi la fondamentale n'est pas à 325V crête, nous en voyons qu'une partie. L'échelle horizontale est à 150Hz/div.





Voici la courbe représentant la FFT de la tension du réseau calculée par le programme. Le résultat est très concluant. La forme générale entre le FFT du LeCroy et celle du programme est identique. Les légères différences qu'on constate au niveau de l'amplitude des harmoniques sont expliquées ci-dessous.

Remarque :

Au chapitre 3.1.1.1, nous avons parlé de la méthode utilisée afin de calculer Δf . Cette valeur dépend totalement du nombre de points que définit l'échantillonnage du signal. L'analyseur échantillonne à 4us et nous voulons 2'097'152 points pour effectuer la FFT. La durée d'analyse est de 8.4 secondes environ. La définition spectrale $\Delta f = 0.1192Hz$ n'est pas du tout un sous-multiple des composantes spectrales. Nous sommes dans l'impossibilité de trouver la valeur exacte des fréquences originales. Néanmoins le résultat s'approche fortement de la valeur souhaitée.

3.4.3.3 Calcul de l'impédance du consommateur

La suite du projet concerne le calcul de l'impédance d'un consommateur Z(ω) en fonction de sa tension U(ω) et de son courant I(ω). Ces deux derniers ont été obtenus par FFT sur une mesure de tension et de courant. U(ω) et I(ω) sont des valeurs complexes. Si nous définissons Ur(ω) et Ui(ω) étant respectivement la partie réelle et imaginaire de la tension, de même pour le courant Ir(ω) et Ii(ω) alors :

$$Z(\omega) = \frac{Ur(\omega) + jUi(\omega)}{Ir(\omega) + jIi(\omega)} = \frac{Ur(\omega)\cdot Ir(\omega) - jUr(\omega)\cdot Ii(\omega) + jUi(\omega)\cdot Ir(\omega) + Ui(\omega)\cdot Ii(\omega)}{Ir^2(\omega) + Ii^2(\omega)} =$$

$$\frac{1}{Ir^2(\omega) + Ii^2(w)} \cdot \left[(Ur(\omega)\cdot Ir(\omega) + Ui(\omega)\cdot Ii(\omega)) + j(Ui(\omega)\cdot Ir(\omega) - Ur(\omega)\cdot Ii(\omega))\right]$$

Le module de $Z(\omega) = \sqrt{\mathrm{Re}^2(Z(\omega)) + \mathrm{Im}^2(Z(\omega))}$, impédance en [Ohm].

La phase de $Z(\omega) = \arctan\left(\frac{\mathrm{Im}(Z(\omega))}{\mathrm{Re}(Z(\omega))}\right)$, déphasage entre la tension et le courant en [rad].

3.4.3.3.1 Programmation

Voici la partie du code qui effectue le calcul de l'impédance en fonction des parties réelles et imaginaires de la tension et du courant :

```
FILE*fileZ=NULL;
float I_limit = 3 ;
float x,phi,f,df,denominateur,Zreel,Zimag ;
int i ;
// ------------------------------------------------------------------------------
// ouverture fichier impédance

if(fileZ==NULL ){
    fileZ = fopen("D:\\TD\\matlab\\imped.txt","wt");
}

if(fileZ == NULL) {
    printf("plus de place sur disque pour le fichier %s","D:\\TD\\matlab\\imped.txt");
    return(1);
}
// ------------------------------------------------------------------------------------------
// Calcul impedance

df =  1.0 / ((float)N * dt);
for(i=0;i<N/2;i++)
{
    f    = ((float)i) * df;
    if ( sqrt(XrI[i]*XrI[i]+XiI[i]*XiI[i]) <= I_limit )
    {
        z[i] = -10.0 ;  // si le courant est trop bas on assigne à Z une valeur non significative
    }
    else
    {
        denominateur=(1/(XrI[i]*XrI[i]+XiI[i]*XiI[i]));
        Zreel=(XrU[i]*XrI[i]+XiU[i]*XiI[i]);
        Zimag=(XiU[i]*XrI[i]-XrU[i]*XiI[i]);
        z[i] = sqrt(((Zreel*denominateur)*(Zreel*denominateur))+((Zimag*denominateur)*(Zimag*denominateur)));
        phi = atan(Zimag/Zreel);
    }

    fprintf(fileZ,"%i\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\n",i,f,Zreel,Zimag,denominateur,phi,z[i]);
}
fclose(fileZ);
```

3.4.3.3.2 Résultats

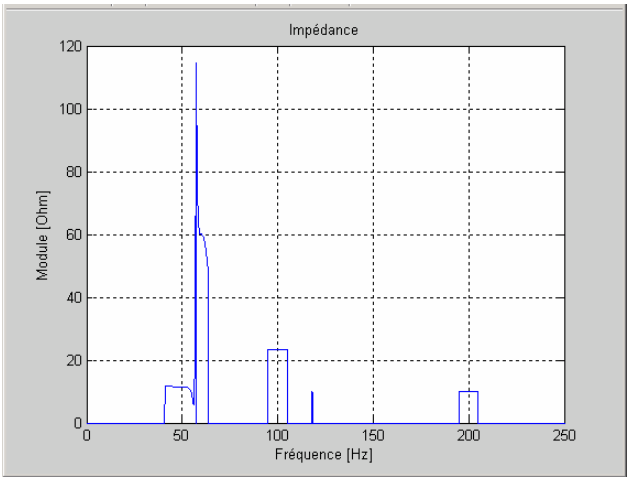
Pour une tension composée des 4 fonctions sinusoïdales suivantes :

U[1]= 230.0 V f[1] = 50.0 Hz
U[2]= 120.0 V f[2] = 60.0 Hz
U[3]= 70.0 V f[3] = 100.0 Hz
U[4]= 50.0 V f[4] = 200.0 Hz

Et un courant suivant :

I[1]= 20.0 A f[1] = 50.0 Hz
I[2]= 2.0 A f[2] = 60.0 Hz
I[3]= 3.0 A f[3] = 100.0 Hz
I[4]= 5.0 A f[4] = 200.0 Hz

L'impédance calculée est la suivante :

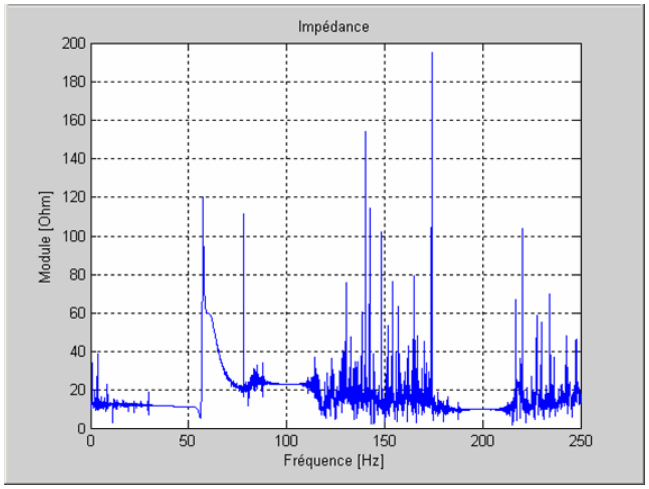


Sur cette courbe nous nous apercevons qu'aux alentours des fréquences des pics de nos signaux, il y a une impédance. La valeur est correcte, il suffit de diviser l'amplitude de la tension par le courant pour l'obtenir. En ce qui concerne tous les points qui possèdent une impédance égale à 0, l'explication est faite ci-dessous.

Remarque :

Les pics sur la courbe d'impédance apparaissent lorsque les parties imaginaires et réelles du courant possèdent une faible amplitude (proche de 0).
Afin de limiter ces pics, il faut empêcher le calcul de l'impédance pour des courants plus petits qu'une valeur déterminée à l'avance. Pour chaque pic filtré, nous lui donnons une valeur de 0. Dans le cas ci-dessus, le courant maximal (norme) atteint 20 A, donc la limitation du courant se fait dans une gamme de 1 à 5 ampères afin d'avoir un filtrage correct. Nous avons choisi 3 ampère Il reste néanmoins quelques pics qui n'ont pas été filtré. Comparé au graphique non filtré le résultat est sans appel.

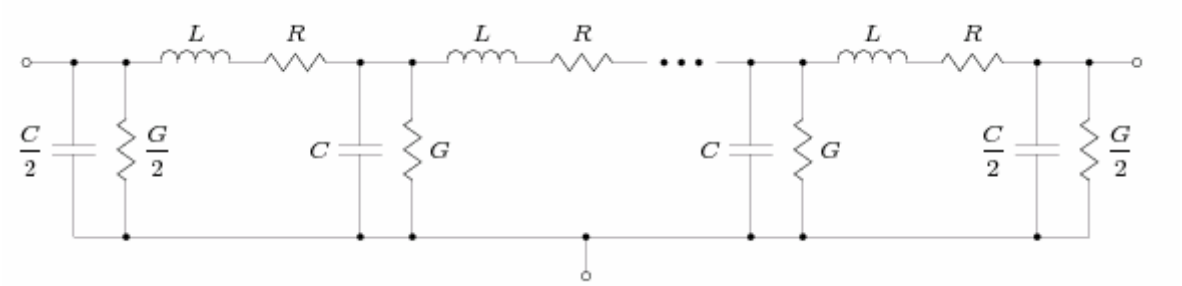
Courbe d'impédance non filtrée :



Ici les pics sont nombreux et atteignent des valeurs importantes. De plus, il est plus aisé d'observer les valeurs d'impédance sur le graphique filtré.

3.4.3.4 Calcul de l'impédance de ligne

Une ligne est toujours représentée par le schéma suivant qu'on nomme plus communément « schéma en π ». Ce circuit en composé ; d'une résistance, d'une inductance et d'une capacité linéique.



Les capacités « C/2 » définissent le début et la fin d'une ligne, nous négligerons ces valeurs pour notre calcul. La conductance G est négligée également à cause de sa très faible valeur. En *annexe 1* se trouve une fiche technique d'un type de câble couramment utilisé en pratique. Le diamètre d'un câble utilisé pour un réseau basse tension se situe entre 25 et 50 mm^2 (référence : M. Heinz-Herbert Krönig, professeur à la HES-SO Sion). R, L et C utilisés pour le calcul ne sont pas des valeurs linéiques, il faut connaître la longueur du câble afin de pouvoir les calculer. Pour ce calcul nous définissons Zi et Zr comme étant respectivement les parties réelles et imaginaires de l'impédance.

$$Zi(\omega) = \omega L + \frac{1}{\omega C} = 2 \cdot \pi \cdot f \cdot L + \frac{1}{2 \cdot \pi \cdot f \cdot C}$$

$$Zr = R$$

$$|Z(\omega)| = \sqrt{Zr^2 + Zi(\omega)^2}$$

$$\varphi(w) = \arctan\left(\frac{zi(w)}{zr}\right)$$

3.4.3.4.1 Programmation

Nous avons décidé de créer une matrice à deux dimensions comme illustré ci-dessous pour 2 producteurs et 3 consommateurs :

L'impédanceZ_{02} représente l'impédance de ligne entre le producteur 0 et le consommateur 2. Idem pour les autres impédances.

Zligne

Producers \ Consumers	C2	C3	C4
P0	Z_{02}	Z_{03}	Z_{04}
P1	Z_{12}	Z_{13}	Z_{14}

$Z_{pc} = Zr + jZi$

La fonction suivante calcul l'impédance de ligne. Il suffit de se déplacer à travers la matrice et de lancer la fonction pour la remplir. Pour ce faire : 2 boucles « for » imbriquées suffisent.

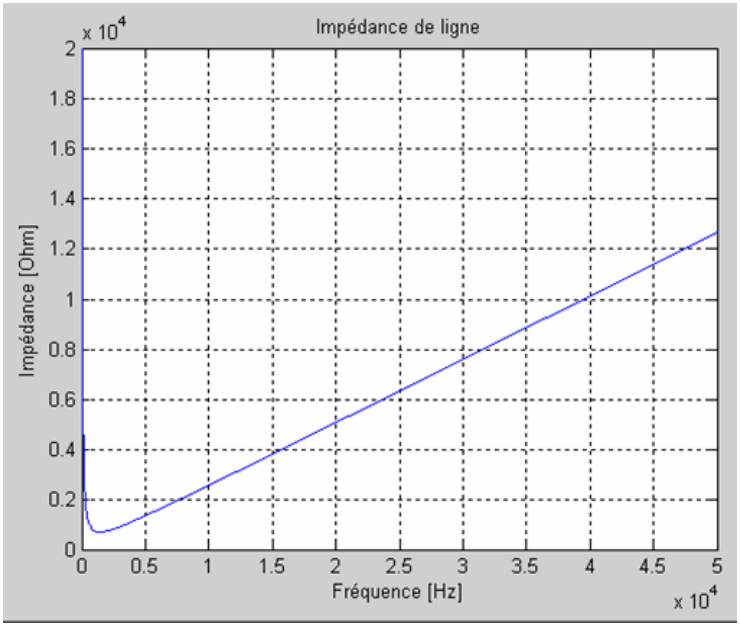
```
void calculLigne(       long N,
                        float dt,
                        float *Zi,
                        float Zr,
                        float *Norme,
                        float prodPos ,
                        float consPos,
                        float Retoile,
                        float Letoile,
                        float Cetoile)
{
        float f,df ;
        float R,L,C;
        int i,lenght ;
        float pi ;
        lenght  =abs(prodPos-consPos);

        R = lenght* 1e-3 * Retoile ;
        L = lenght* 1e-6 * Letoile ;
        C = lenght* 1e-9 * Cetoile ;

        df =  1.0 / ((float)N * dt);
        Zr = R ;
        for(i=0;i<N/2;i++)
        {
                f   = (((float)i)+1) * df;
                Zi[i] = (2*PI*f*L)+(1/(2*PI*f*C)) ;
                Norme[i]= sqrt( Zr*Zr+ Zi[i]*Zi[i]);
        }

}
```

3.4.3.4.2 Résultats



Pour des basses fréquences, l'impédance est élevée à cause de la capacité du câble. Ensuite pour des hautes fréquences le câble devient totalement inductif comme le montre cette courbe.

3.4.4 Interface utilisateur

Avant de s'intéresser à l'affichage graphique, il est nécessaire que nous travaillons sur l'interface utilisateur. L'interface « DOS » sous Visual Studio C++ n'est pas adaptée pour offrir un rendu propre et simple d'utilisation. Nous devons utiliser des logiciels spécialement conçus pour la création d'interfaces graphiques. Le programme que nous avons choisi est le suivant : Borland Builder C++. La différence avec du C++ standard « séquentielle » est que Borland C++ est « événementielle ». Lorsque l'utilisateur clic sur un bouton ou une zone de saisie le programme va appeler une fonction qui correspond à l'action qui vient d'être effectuée. A chaque « événement » correspond une fonction. Les fonctionnalités de notre interface se trouvent au chapitre 3.4.1.
La programmation de cette interface est en *annexe 2 à 8*.

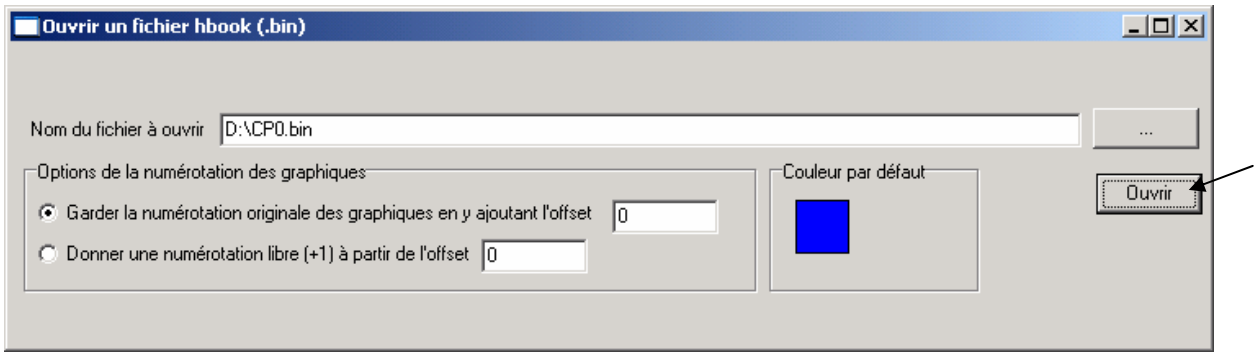
3.4.5 Affichage graphique

La partie traitement des signaux et interface graphique est maintenant terminée. Nous pouvons réfléchir à un système d'affichage graphique. Nous avons le choix entre : MATLAB ou Hbook6. Ce dernier est un programme moins évolué que MATLAB mais est capable d'afficher des graphiques. Comment afficher un graphique sur MATLAB depuis notre programme ? Cela n'est pas possible. La fonction qui permet d'afficher un graphique sur MATLAB se trouve dans un fichier « .m ». Les « .m » sont des fichiers exécutables uniquement depuis MATLAB. De ce fait nous ne pouvons pas lancer ce fichier depuis l'extérieur. Le seul moyen est de générer un fichier « .txt » depuis notre programme et de le lire depuis MATLAB. Evidemment cela n'est pas la solution la plus pratique. Nous nous sommes tournés vers Hbook6 que nous connaissons bien. Son guide d'utilisation se trouve ci-dessous.

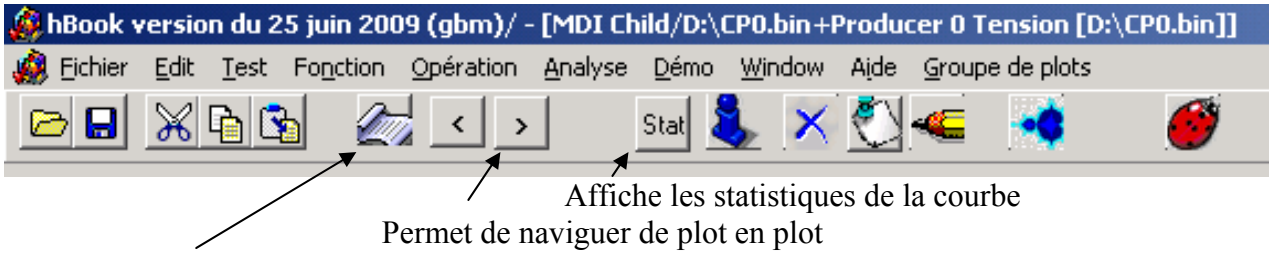
3.4.5.1 Guide d'utilisation de Hbook6

Hbook6 lit des fichiers « .bin ». La première chose à faire est de configurer Windows afin qu'il oblige Hbook6 à être le programme par défaut pour la lecture de fichiers « .bin ».
Le principe est le suivant : notre programme génère un fichier « .bin » qui contient toutes les valeurs de nos signaux. Ensuite il suffit de lancer le fichier depuis notre programme et Hbook6 se charge du reste.

La fenêtre suivante apparaît et vous avez juste à cliquez sur « Ouvrir » pour afficher les graphiques que contient le fichier.



Vous arrivez ensuite sur le premier [1]plot que contient votre fichier. Une description des fonctions qui nous sont utiles de la barre d'outils est faite ci-dessous :



Affiche les statistiques de la courbe

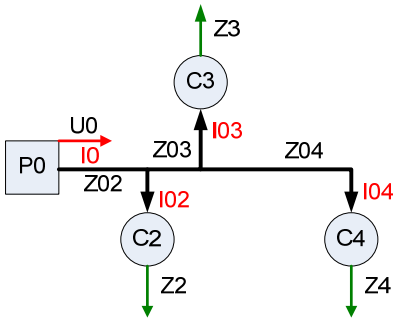
Permet de naviguer de plot en plot

Affiche la liste des plots présents dans votre fichier

[1]Plot = fenêtre contenant tous les points qui composent un signal.
Vous avez la possibilité de déplacer les axes horizontaux et verticaux de votre graphique afin de réduire les échelles. Pour ce faire, cliquez sur un axe et rapprochez le du centre du graphe. Pour faire un « zoom out », faites un clic gauche et déplacé la souris vers la gauche. Pour faire un « zoom in », faites un clic gauche et déplacé la souris vers la droite. Double-cliquez sur les axes pour voir apparaître une fenêtre qui vous permette d'entrer les valeurs minimales et maximales de vos échelles. En faisant un clic droit sur le graphique une fenêtre apparait et vous pouvez modifier certaines caractéristiques graphiques de votre signal par exemple : l'épaisseur des traits, le style de trait, etc…
Vous avez la possibilité de sauver vos graphiques sous format BMP ou JPEG. En ce qui concerne les autres fonctions, elles vous sont totalement inutiles, mais vous pouvez quand même les découvrir par vous-même.

3.4.6 Equilibrage du réseau

Le terme équilibrage est synonyme de répartition. Cette fonction répartie le courant que fourni chaque producteur aux consommateurs. Nous avons réussi à déterminer : la tension de sortie des producteurs, l'impédance de ligne et l'impédance des consommateurs. Le schéma ci-dessous va nous aider à expliquer la méthode de calcul que nous avons utilisé.



I0=I02+I03+I04

Zpc = impédance de ligne entre p et c

Zi = impédance du consommateur i

p = producteur

c = consommateur

Nous affirmons que le courant $I_{02} = \frac{U_0}{(Z_{02} + Z_2)}$ (courant qui est produit pas P0 et consommé par C2). Ensuite si nous voulons obtenir le courant total fourni par un producteur il suffit d'additionner tous les courants qui sont produits par ce producteur et consommés par les consommateurs. Le calcul est le même si on a 'x' producteurs et 'y' consommateurs. Nous devons prendre en compte le courant maximal qui peut être tiré de chaque producteur. Mais afin de faciliter les calculs nous estimons que chaque producteur possède une puissance infinie.

3.4.6.1 Calcul

$$\mathrm{Re}(I_{02}) = \mathrm{Re}\left(\frac{U_0 r + U_0 i}{(Z_{02}r + Z_2 r + Z_{02}i + Z_2 i)}\right) = \frac{1}{(Z_{02}r + Z_2 r)^2 + (Z_{02}i + Z_2 i)^2} \cdot \left(U_0 r \cdot (Z_{02}r + Z_2 r) + U_0 i \cdot (Z_{02}i + Z_2 i)\right)$$

$$\mathrm{Im}(I_{02}) = \mathrm{Im}\left(\frac{U_0 r + U_0 i}{(Z_{02}r + Z_2 r + Z_{02}i + Z_2 i)}\right) = \frac{1}{(Z_{02}r + Z_2 r)^2 + (Z_{02}i + Z_2 i)^2} \cdot \left(U_0 i \cdot (Z_{02}r + Z_2 r) - U_0 r \cdot (Z_{02}i + Z_2 i)\right)$$

3.4.6.2 Programmation

La fonction qui réalise le calcul des courants consommés est la suivante :

```
void calculCourant(     long N,
                        float *ir,
                        float *ii,
                        float *norme,
                        float zr,
                        float *zi,
                        float *XrU,
                        float *XiU,
                        float *Zr,
                        float *Zi)
{
        int i ;
        float deno ;

        for(i=0;i<N/2;i++)
        {
                deno = 1/(((zr+Zr[i])*(zr+Zr[i]))+((zi[i]+Zi[i])*(zi[i]+Zi[i])));
                ir[i]= deno*(XrU[i]*(zr+Zr[i])+XiU[i]*(zi[i]+Zi[i]));
                ii[i]=deno* (XiU[i]*(zr+Zr[i])-XrU[i]*(zi[i]+Zi[i]));
                norme[i]= sqrt(ir[i]*ir[i]+ii[i]*ii[i]);
        }
}
```

ir[i] correspond à la partie réelle du courant calculé et ii[i] à la partie imaginaire. L'impédance de ligne est représentée par zr pour la partie réelle et zi pour la partie imaginaire. Zr et Zi représente le consommateur et XrU, XiU la tension de sortie du producteur.

3.4.6.3 Résultats

Nous avons remarqué que cette méthode de calcul est totalement incorrecte. Le fait que nous ne prenons pas en compte les autres impédances du circuit rend notre calcul erroné. Au chapitre 6.1.3 une méthode de calcul matriciel a été développée. Nous pensons que cette méthode permet un calcul rapide et simple de n'importe quel circuit comportant des impédances en série et en parallèle.

3.4.7 Assemblage du logiciel ATAR

Les fonctions sont toutes terminées. Avant de pouvoir effectuer les tests finaux nous devons réunir toutes les fonctions et réussir à les faire communiquer ensemble afin de créer un programme que nous nommerons : ATAR (Acquisition, Traitement, Affichage, Réseau). Le schéma ci-dessous représente cette communication.

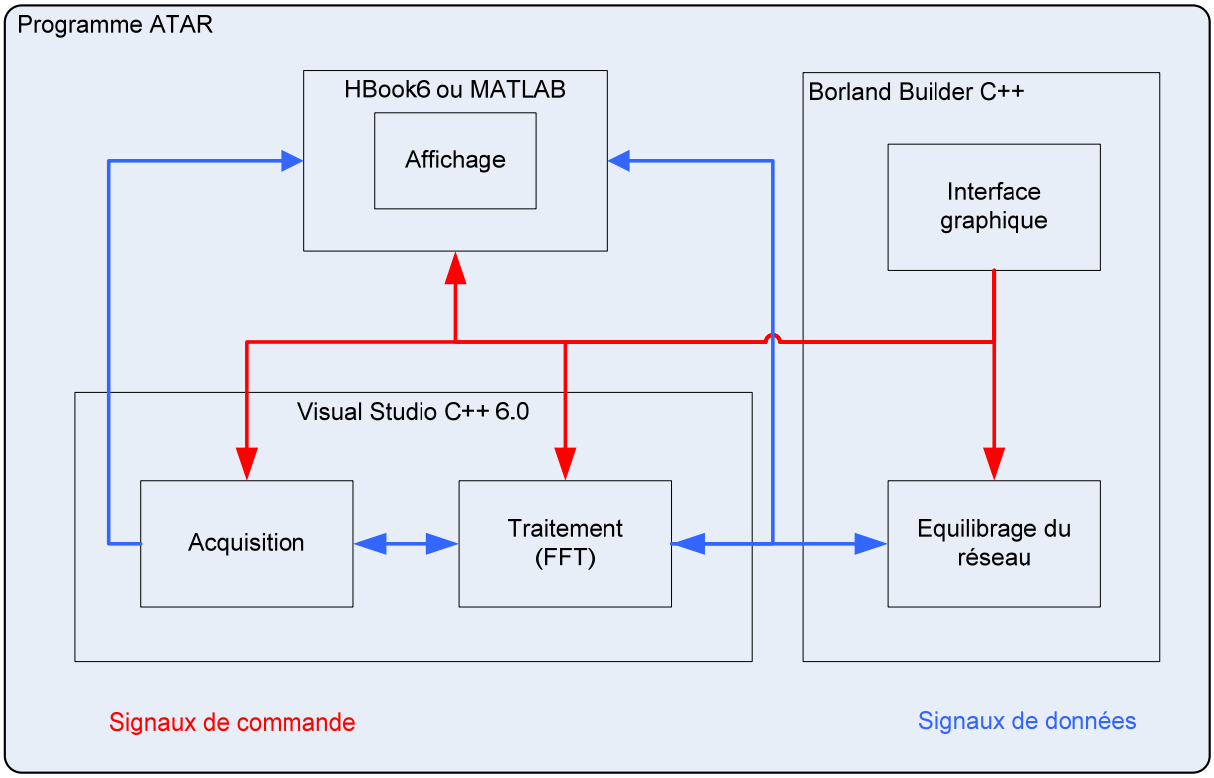
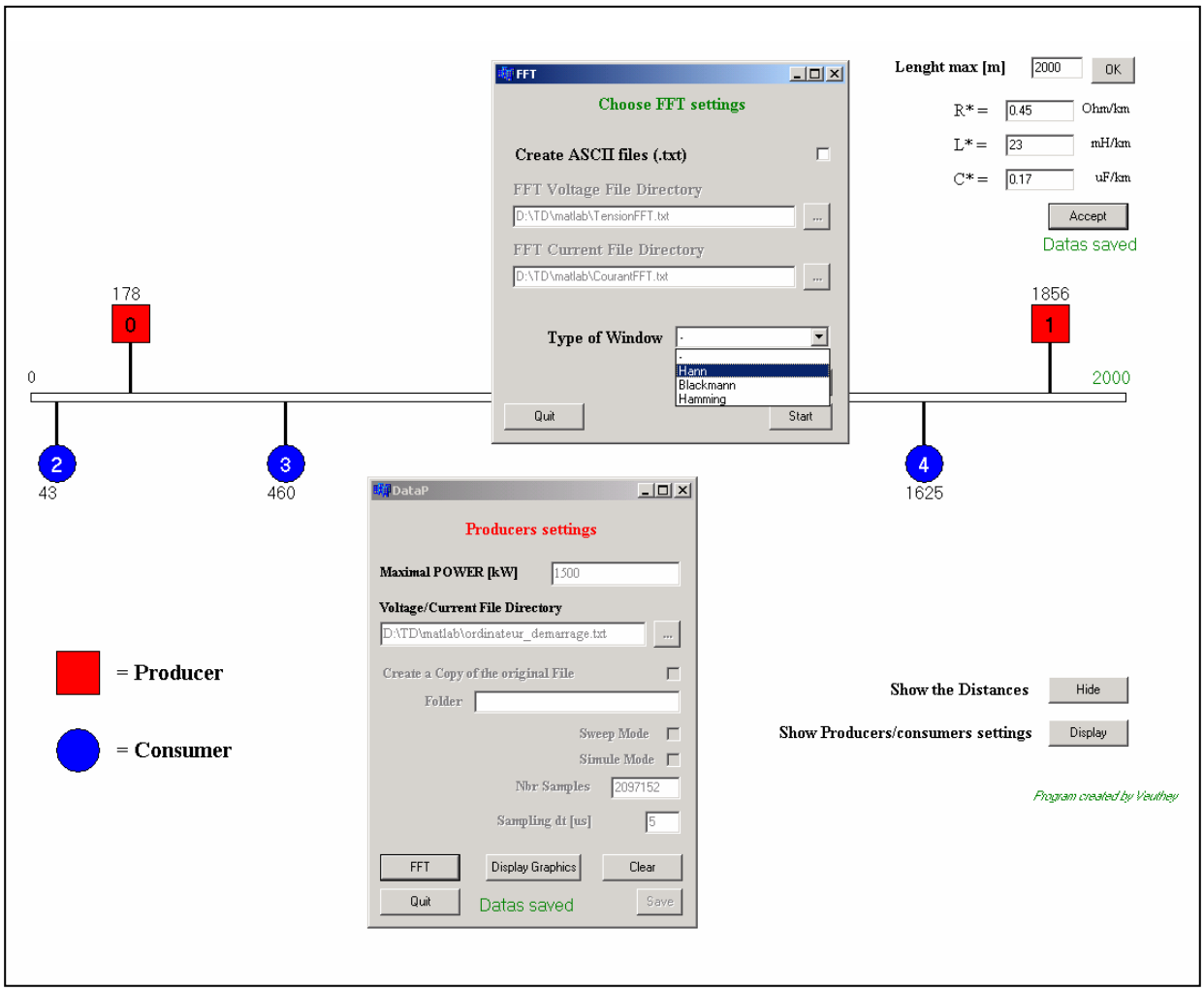


Figure 15 : Schéma bloc du programme

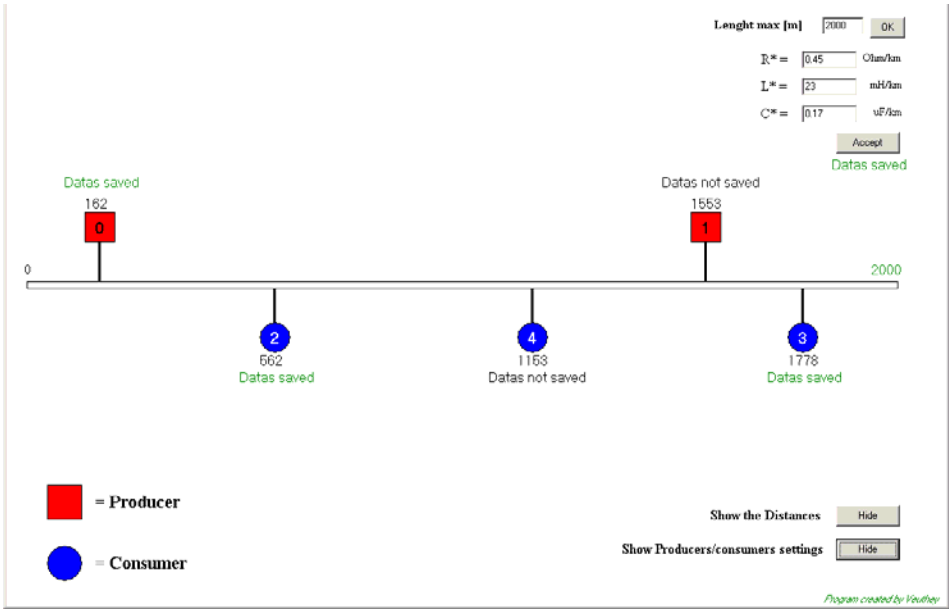
Le bloc « Acquisition » signifie lecture et sauvegarde des fichiers acquis pour notre base de données (voire chapitre 3.4.3.2). Le bloc « Traitement » effectue des calculs de FFT et d'impédances. Le bloc « Equilibrage du réseau » calcul les courants fournis et consommés. L'interface graphique, à l'aide d'un utilisateur, gère l'ensemble des fonctions du programme.

3.4.8 Superposition des résultats

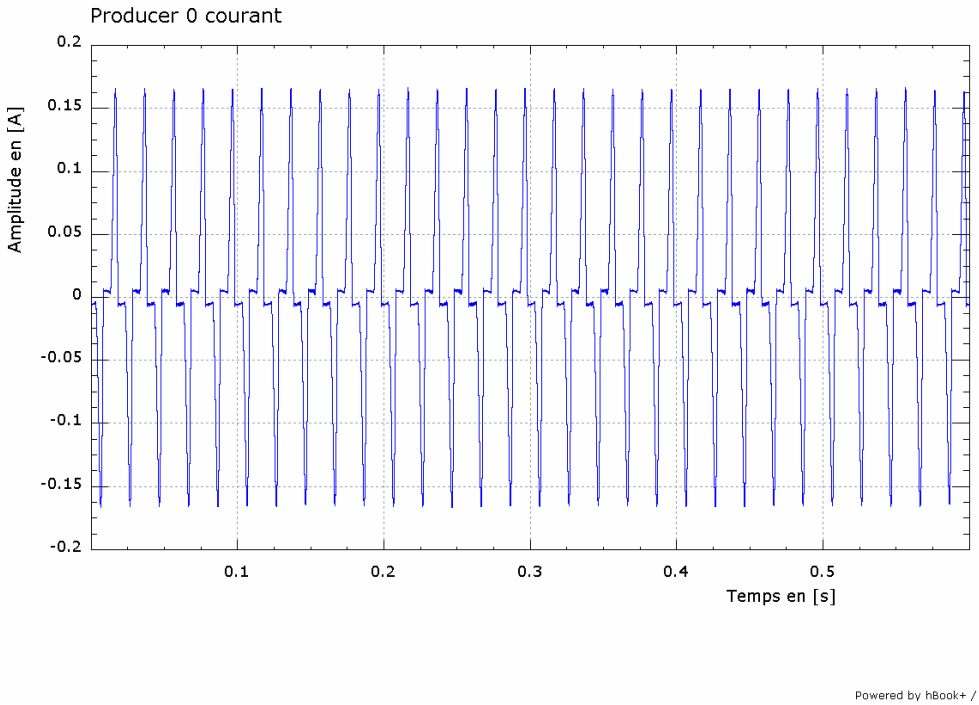
Mise à part la fonction « Equilibrage du réseau », le programme est terminé. Ci-dessous une série d'images et graphiques nous révèlent l'aspect final du programme ATAR.



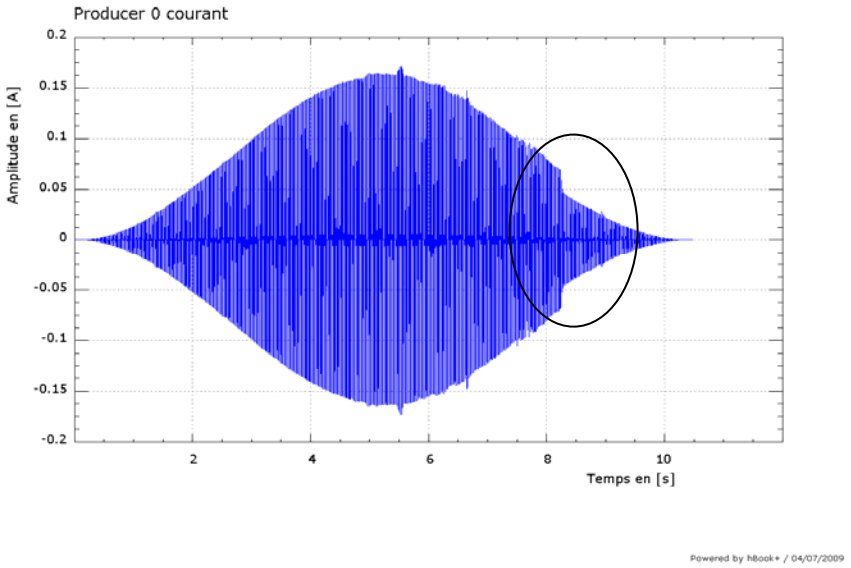
La fenêtre principale du programme décrivant un réseau composé de deux producteurs et trois consommateurs. Sur celle-ci se trouvent deux fenêtres secondaires. La première vers le bas s'affiche lorsque nous souhaitons entrer des caractéristiques à un producteur. La seconde traite ces données par une FFT.



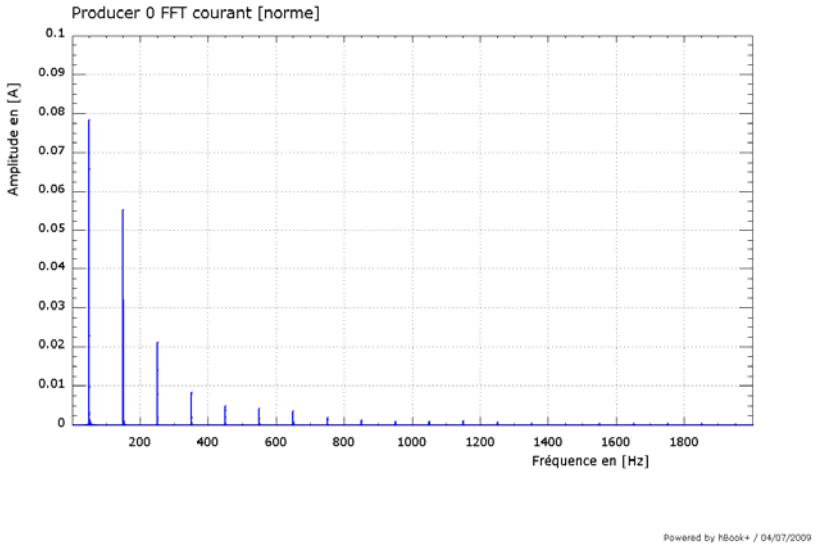
Cette image nous montre l'aspect d'une autre fonction graphique que contient notre programme. Celle-ci consiste à afficher l'état de sauvegarde des données de chaque producteur et consommateur. Si les producteurs/consommateurs ont un fichier attaché, le programme affiche « datas saved ». Dans le cas contraire il affiche « datas not saved ».



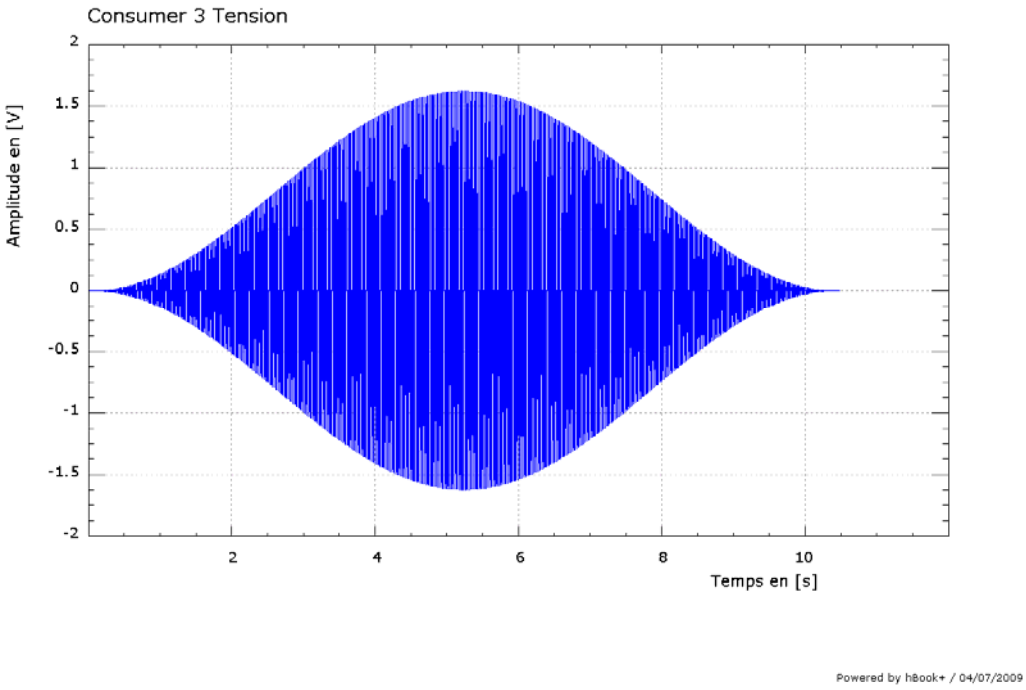
Cette courbe nous montre l'allure du courant tiré par un ordinateur en marche. L'acquisition de cette mesure a été faite avec la carte d'acquisition. Le programme sauvegarde cette acquisition dans sa mémoire et nous pouvons ensuite l'afficher sur hbook6.



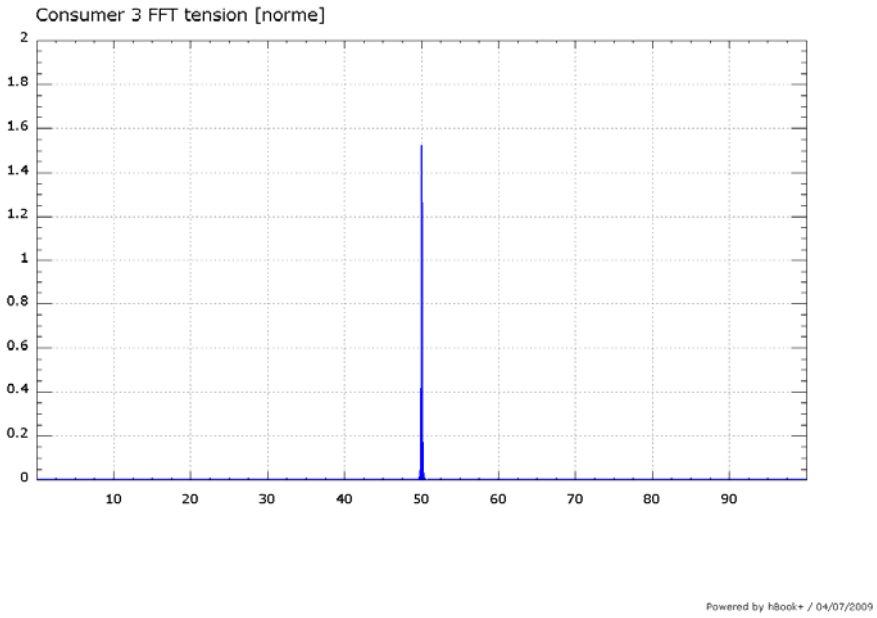
Le graphique ci-dessus représente le courant précédant (courant de l'ordinateur) mais avec une fenêtre de Hann. Nous remarquons que la valeur du courant est assez faible pour un ordinateur. Cela est du à la sonde de Pearson que nous avons utilisé et qui a un rapport de mesure de 1 :10. Comme la compensation de la sonde n'a pas été faite, nous avons des valeurs d'amplitude dix fois trop faible.



Après traitement du signal précédant par une FFT nous obtenons le graphique ci-dessus. Le signal possède un certain nombre d'harmoniques. Nous nous apercevons que la fondamentale à 50Hz ne possède pas une amplitude identique au signal en fonction du temps. Nous avons conclu que comme le signal a un changement d'amplitude vers les 8secondes (cercle noir sur graphique) la FFT de celui-ci est faussée.



Pour prouver que ce n'est pas une erreur du logiciel ou de l'algorithme de la FFT, voici le graphique d'un signal de tension du réseau avec sa FFT en dessous. L'amplitude est de 1.6 V car la sonde utilisée pour la mesure avait un facteur de réduction de 1 :200.

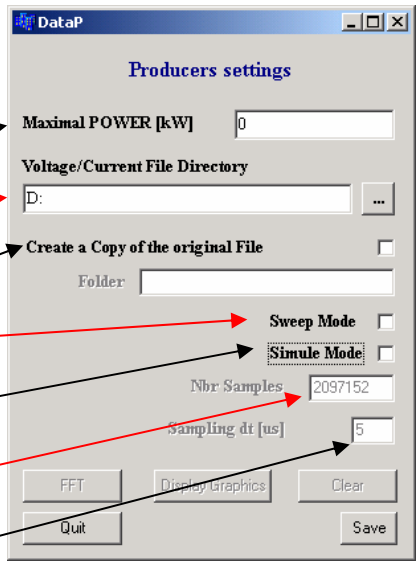


L'amplitude de la fondamentale est bien identique à l'amplitude du signal en fonction du temps.

3.4.8.1 Guide d'utilisation du programme

Lancez le programme nommé ATAR qui se trouve sur le CD du même nom.
Après son exécution vous arrivez sur la page principale où se trouve un long trait horizontale qui représente le câble d'un réseau quelconque. Voici les étapes à suivre pour une utilisation correcte du programme :

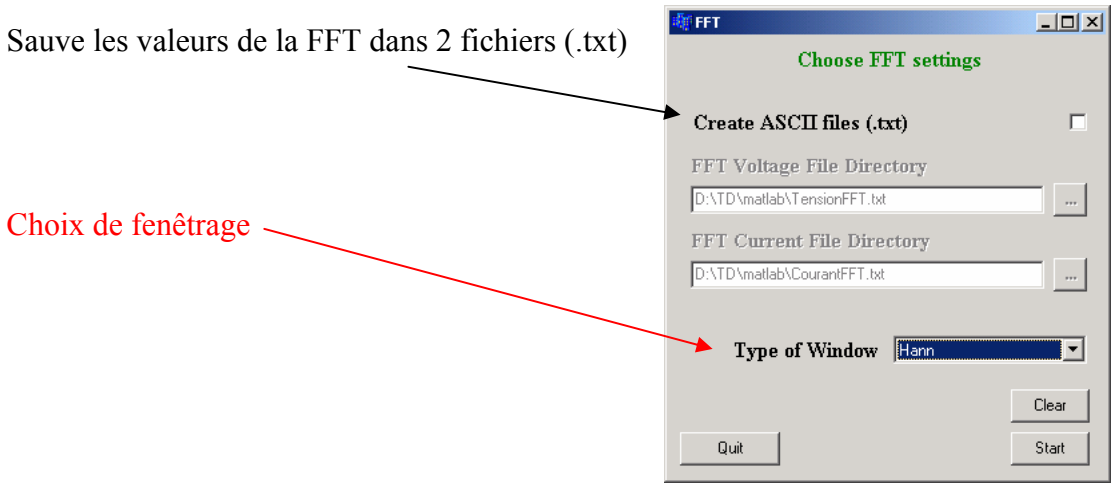
1. Définissez la longueur du câble à l'aide de la zone de saisie en haut à droite. Après avoir cliqué sur « OK » vous voyez apparaître, sur la droite du câble, la longueur que vous avez entrée, en vert. Cela signifie qu'elle à bien été enregistrée.
2. Une fenêtre apparaît au même instant vous invitant à saisir le nombre de producteurs et de consommateurs voulu. Sur ce point veuillez prêter une attention toute particulière aux contraintes qui ont été fixées afin de ne pas saturer les capacités de l'ordinateur, ces contraintes se trouvent après le guide de fonctionnement. Cliquez sur « insert » pour terminer. Vous pouvez soit fermer la fenêtre avec « Quit » soit la laisser au second plan. A noter que si vous fermer la fenêtre vous pouvez toujours la faire réapparaître soit en modifiant la longueur du câble soit en cliquant sur « View » puis « Tools ».
3. Positionnez vos consommateurs et producteurs où bon vous semble sur le câble en faisant un « cliquer-glisser », une sécurité a été prévue afin de ne pas sortir sur les extérieurs.
4. Si vous faites clic droit de la souris sur un producteur, la fenêtre suivante s'ouvre et vous pouvez la compléter.



Puissance de sortie du producteur
Fichier contenant U/I mesuré à la sortie du producteur
Possibilité de faire une copie du fichier original
Sweep Mode génère un balayage prédéfini ($A=20, f_{min}=50Hz, f_{max}=500Hz$)
Simule Mode génère U/I de façon prédéfini (voir chapitre 6.1.1 pour les valeurs)
Si « Sweep ou Simule » est actif, il faut définir le nombre d'échantillons qui composera le signal
Le « δt » correspond à la période d'échantillonnage

Pour sauver votre configuration, cliquez sur le bouton « save », en bas à droite. La fenêtre du type « consommateur » est identique. Si vous cliquez sur « clear » après avoir sauvé, vous effacez toutes les donnés liées à ce producteur/consommateur.

5. Vous avez la possibilité d'afficher le signal sur un graphique avec le bouton « Display Graphics ».
6. En cliquant sur le bouton « FFT » vous accédez à la fenêtre suivante :



7. Cliquez sur « Start » pour démarrer le calcul de la FFT. Ensuite « Quit » pour revenir ou « Clear » pour effacer.
8. Vous êtes revenu à la fenêtre précédente. Si après avoir effectué un calcul de FFT vous cliquez sur « Display Graphics » vous affichez la totalité des graphiques.
9. A tout moment vous pouvez revenir à la fenêtre principale. Si vous cliquez sur « Display Distances » vous affichez la distance qui sépare chaque prod/cons du point de référence « 0 ». Si vous cliquez sur « Display Prod/Cons settings » vous savez quelles données ont été sauvées ou pas.
10. Dès que toutes les données ont été sauvées vous pouvez lancer la partie équilibrage du réseau à l'aide de l'onglet « Run->run ». Cette fonction n'est pas terminée. Le seul calcul effectué est : impédance de ligne.

3.4.8.1.1 Contraintes

- Le nombre total de consommateurs et de producteurs doit être adapté aux capacités de l'ordinateur sur lequel le programme est installé pour les raisons suivantes : chaque consommateur ou producteur se voit allouer de la mémoire afin de pouvoir enregistrer toutes les données que nous lui assignons. C'est plus de 100Mo de mémoire vive qui est allouée à chaque fois. Tout dépend de la mémoire RAM de l'ordinateur qui supporte le programme.

4 Tests

Une série de tests intermédiaires ont été effectué. Ces tests assurent que le programme fonctionne correctement, ils se trouvent dans le protocole de tests ci-dessous.

4.1 Protocole de tests

Un protocole de tests a été effectué sur les fonctions du programme suivantes : interface graphique, traitements des données. La fonction « équilibrage du réseau » doit subir des modifications.

Fonction	Description du test	Résultats	Décision
Graphique	Définir la longueur câble à 0m	Message d'alarme	OK
	Définir la longueur câble à 2000m	Valeur sauvée + affichage fenêtre P/C	OK
	Nombre de P/C égal à 0	Message d'alarme	OK
	Insertion de 2 P et 3 C	P/C insérés correctement avec numérotation	OK
	Modification le nombre de P/C à 1/4	Modification correcte + mise à jour de la numérotation	OK
	Déplacer les P/C sur le câble (clic gauche), le C4 en bout de ligne	Chaque P ou C peut être déplacé / Bloquage aux extrémités	OK
	Afficher/cacher les distances des P/C	Affiche/cache + met à jour avec déplacement des P/C	OK
	Modifier longueur câble à 500m avec affichage distances des P/C	Distances mises à jour	OK
	Entrer R*=0.5 ,L*=38 et C*=0.2 puis "Accept"	Affichage du message "Datas saved"	OK
	Entrez R*=0 et laisser les autres valeurs	Message "Datas saved" supprimé + message d'alarme	OK
	Clic droit sur C1	Affichage fenêtre "consumer settings"	OK
	Clic droit sur P0	Affichage fenêtre "producer settings"	OK
Traitement	Sauver un fichier de 3 colonnes de 2097152 lignes	Fichier chargé correctement	OK
	Faire de même mais en cochant "Create copy..."	Fichier chargé correctement + copie du fichier créée	OK
	Afficher le graphique avec "Display graphics"	Lancement de Hbook6 + affichage correcte	OK
	Effacer les datas avec "Clear"	Datas effacées + mémoire libéré	OK
	Sauver fichier de 5 colonnes de 1048576 lignes + affichage	Pas d'erreur mais affichage incorrecte	KO
	Sauver fichier de 3colonnes de 2097152 lignes + FFT + affichage	Fichier sauvé + affichage des plots corrects	OK
	Mode "Sweep", N = x , dt = x + affichage + copy	Copy correcte, affichage correcte	OK
	Mode "Simule", N = x , dt = x + affichage + copy	Copy correcte, affichage correcte	OK
	Sauver fichier dans consommateur 'x' + FFT + affichage	Affichage de l'impédance + autres	OK

Nous avons un problème avec l'enregistrement des fichiers dans certaines conditions. Une amélioration est prévue (voire chapitre 6.2.2).

5 Analyse des résultats

Nous allons commenter le fonctionnement du programme en le divisant en 4 parties.

5.1 Interface graphique

Le résultat obtenu est très satisfaisant. Aucun bug n'a été détecté jusqu'à présent. Il y a possibilité d'améliorer l'apparence de la page principale et d'ajouter quelques fonctionnalités (voire chapitre 6.2). Nous avons passé beaucoup de temps à étudier et programmer le système de déplacement des producteurs/consommateurs. Nous sommes vraiment satisfait du résultat obtenu.

5.2 Traitement des données

C'est la première fonction que nous ayons réalisé et c'est la plus abouti. La FFT et la TFD fonctionnent parfaitement. Le calcul d'impédance nous donne des résultats satisfaisants. Il reste néanmoins à améliorer légèrement la lecture et sauvegarde des fichiers d'acquisition. La fonction qui pose encore problème dans le traitement des données est le calcul du déphasage. Nous avons une variation du déphasage très importante entre chaque fréquence et nous ne savons pas pourquoi. *En annexe 3* se trouve un graphique qui nous montre le déphasage obtenu pour un consommateur quelconque.

5.3 Affichage

L'affichage fonctionne très bien. Nous avons modifié le programme d'affichage afin qu'il soit plus rapide à afficher les graphiques. Six ou sept graphiques contenant chacun plus d'un million de points ralentissent beaucoup le programme.

5.4 Equilibrage du réseau

Nous n'avons aucun résultat concernant l'équilibrage du réseau car celui-ci reste à faire.

6 Conclusion

6.1 Reste à faire

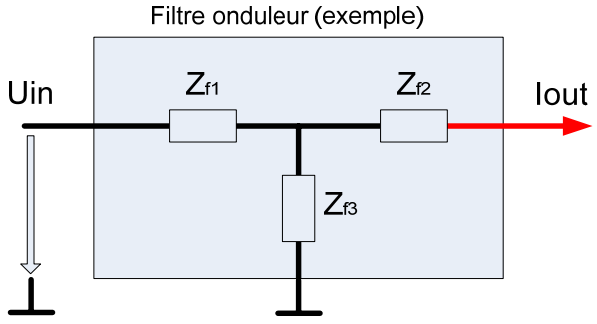
Une description de l'ensemble des travaux qu'il reste à effectuer sur le projet est faite ci-dessous.

6.1.1 Mesures sur consommateurs

Nous avons effectué des mesures sur des consommateurs divers comme par exemple : chargeur de natel, ordinateur, micro-onde, aspirateur. Mais si vous connectez ces « charges » au réseau pour les mesures, celui-ci influence plus ou moins fortement l'impédance totale et la mesure se voit automatiquement faussée. Le seul moyen que nous ayons trouvé afin de mesurer l'impédance d'un consommateur est de le débrancher du réseau et de lui injecter un signal qui balaie des fréquences variant de 2kHz à 50 ou 100kHz et de mesurer pour chaque fréquence le courant et la tension à ces bornes. La réalisation de cet outil est un projet en cours et n'est pas encore terminé. Pour tester le programme j'ai utilisé les mesures effectuées sur les charges lorsqu'elles étaient connectées au réseau. Il reste à faire des mesures correctes sur des consommateurs afin de pouvoir déterminer leurs impédances.

6.1.2 Ajout d'un filtre de sortie pour onduleur

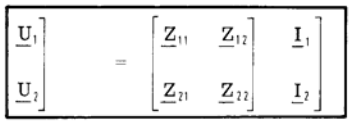
Le problème avec la mesure sur consommateurs est le même avec la mesure sur producteurs. Nous ne pouvons pas faire une mesure de tension de sortie d'un onduleur si celui-ci est connecté au réseau. Un onduleur possède un filtre de sortie qu'il est possible de modéliser. Nous pensons qu'il est possible de mesurer la tension de l'onduleur avant le filtre (tension PWM) sans que celui-ci soit connecté au réseau. Ensuite suivant le type de schéma du filtre, figure ci-dessous, nous pouvons calculer le courant de sortie du filtre et l'appliquer à notre modèle de réseau. Il reste à : déterminer les valeurs du filtre de sortie de l'onduleur, déterminer le type de montage du filtre, faire une fonction qui calcule le courant de sortie en fonction de la tension de sortie (voire chapitre suivant), implémenter le tout sur le programme.

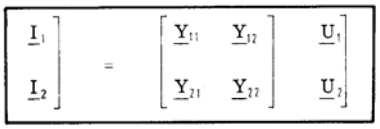


6.1.3 Equilibrage du réseau

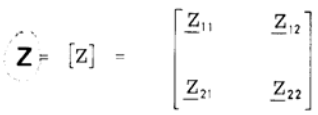
La théorie des « bi-portes » permet de créer des modèle pour des composants électroniques complexes comme : un transistor, un MOS, ou un transformateur. Cette théorie permet également de résoudre des circuits comportant un nombre élevé d'impédance totalement inconnue rien qu'avec des mesures. Comme toutes les impédances qui composent notre modèle sont connues, cette théorie devient plus simple.

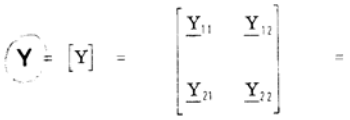
Cette théorie définie des matrices d'impédances ou d'admittances en fonction de la tension/ courant d'entré et tension/courant de sortie. Ci –dessous les matrices d'impédances et d'admittances.





Die Matrix





En fonction de la complexité du circuit d'impédances que nous avons, il faut choisir entre une matrice d'admittances ou d'impédances. A l'aide d'un tableau de conversion nous pouvons transformer la matrice d'impédances/admittances en une matrice appelée « Kettenmatrix » (matrice des coefficients). Ce tableau se trouve dans la livre Technische Elektrizitätslehre 3 qui est mentionné au chapitre 7.

Cette matrice des coefficients contient 4 coefficients qui lient la tension/courant d'entrée à la tension/courant de sortie.

Impedanz- und Admittanzmatrix eines Zweitores verknüpfen das Strompaar mit dem Spannungspaar. Oft ist es zweckmässiger, die Eingangsgrössen $\underline{U}_1$, $\underline{I}_1$ in Funktion der Ausgangsgrössen $\underline{U}_2$, $\underline{I}_2$ darzustellen :

$$\begin{bmatrix} \underline{U}_1 \\ \underline{I}_1 \end{bmatrix} = [A] \begin{bmatrix} \underline{U}_2 \\ -\underline{I}_2 \end{bmatrix}$$

Zweitorgleichungen in Kettenform **) (19.15)

[A] ist dabei die Kettenmatrix: ***)

$$\mathbf{A} = [A] = \begin{bmatrix} \underline{A}_{11} & \underline{A}_{12} \\ \underline{A}_{21} & \underline{A}_{22} \end{bmatrix} \quad (19.16)$$

*) Die Admittanzen $\underline{Y}_{11}$, $\underline{Y}_{12}$ sind hier Eingangs- und Uebertragungsleitwerte, also nicht dieselben Grössen wie in (19.2), wo sie Trennbündelleitwerte bedeuten.

**) Besonders zu beachten ist das Minuszeichen bei $-\underline{I}_2$. Diese Konvention erweist sich später (Abschnitt 19.14) als vorteilhaft beim Zusammenschalten von Zweitoren.

***) Diese Grössen $\underline{A}_{11} \ldots \underline{A}_{22}$ haben nichts zu tun mit den Adjunkten $\underline{A}_{11} \ldots \underline{A}_{nn}$ in Gl.(19.3)(19.4).

L'avantage de ces « Kettenmatrix » est que nous pouvons les mettre en série autant de fois que nous le voulons. Le calcul d'une nouvelle matrice à partir de deux « Kettenmatrix » en série est fait ci-dessous.

Besonders häufig gebraucht wird die Kettenschaltung von Zweitoren. (Fig. 19.52).

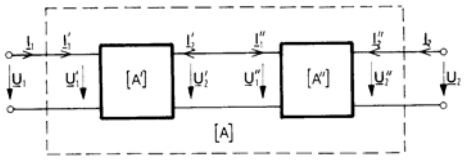


Fig. 19.52 Kettenschaltung von Zweitoren

Gleichungen der einzelnen Glieder:

$$\begin{bmatrix} \underline{U}_1 \\ \underline{I}_1 \end{bmatrix} = \begin{bmatrix} \underline{U}_1' \\ \underline{I}_1' \end{bmatrix} = [A'] \begin{bmatrix} \underline{U}_2' \\ -\underline{I}_2' \end{bmatrix}$$

$$\begin{bmatrix} \underline{U}_1'' \\ \underline{I}_1'' \end{bmatrix} = [A''] \begin{bmatrix} \underline{U}_2'' \\ -\underline{I}_2'' \end{bmatrix} = [A''] \begin{bmatrix} \underline{U}_2 \\ -\underline{I}_2 \end{bmatrix}$$

Zusammenschaltbedingungen:

$$\begin{bmatrix} \underline{U}_2' \\ -\underline{I}_2' \end{bmatrix} = \begin{bmatrix} \underline{U}_1'' \\ \underline{I}_1'' \end{bmatrix}$$

Gleichung des gesamten Zweitores durch Einsetzen:

$$\begin{bmatrix} \underline{U}_1 \\ \underline{I}_1 \end{bmatrix} = [A'] \cdot [A''] \begin{bmatrix} \underline{U}_2 \\ -\underline{I}_2 \end{bmatrix}$$

$$\boxed{[A] = [A'] \cdot [A'']}$$ Kettenschaltung von Zweitoren (19.94)

Cette explication concerne la méthode générale de calcul. Notre réseau ne comporte que des impédances en série et en parallèle ce qui rend la méthode de calcul encore plus simple. Ci-dessous des exemples de « Kettenmatrix » pour des montages d'impédances/admittances simples.

Impédance en parallèle :

19.15 Spezielle Zweitore

Im folgenden sind katalogartig einige elementare Zweitore mit ihren zugehörigen, wichtigsten Matrizen zusammengestellt. Allerdings existieren bei vielen speziellen Zweitoren nicht alle Matrizen. Aus den einfachen Zweitoren können die komplizierteren nach den Zusammenschaltregeln gewonnen werden.

a) Querimpedanz

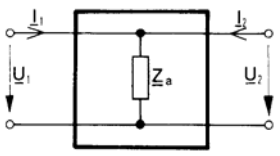


Fig. 19.54

$$\mathbf{Z}_Q = \underline{Z}_a \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad \mathbf{A}_Q = \begin{bmatrix} 1 & 0 \\ \underline{Z}_a^{-1} & 1 \end{bmatrix} \quad \mathbf{H}_Q = \begin{bmatrix} 0 & 1 \\ -1 & \underline{Z}_a^{-1} \end{bmatrix} \qquad (19.96)$$

Montage en T :

T - Glied

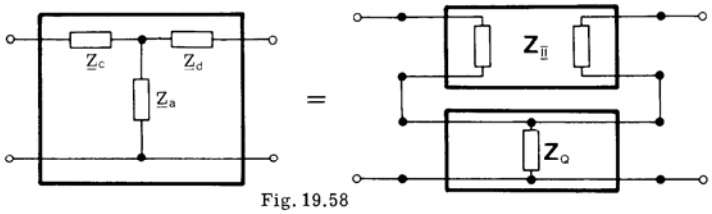


Fig. 19.58

$$Z_T = Z_Q + Z_{II}$$

$$Z_T = \begin{bmatrix} (Z_a + Z_c) & Z_a \\ Z_a & (Z_a + Z_d) \end{bmatrix} \quad A_T = \frac{1}{Z_a} \begin{bmatrix} (Z_a + Z_c)(Z_a Z_c + Z_a Z_d + Z_c Z_d) \\ 1 \quad (Z_a + Z_d) \end{bmatrix}$$

$$Y_T = \frac{1}{Z_a Z_c + Z_a Z_d + Z_c Z_d} \begin{bmatrix} (Z_a + Z_d) & -Z_a \\ -Z_a & (Z_a + Z_c) \end{bmatrix} \qquad (19.100)$$

$$H_T = \frac{1}{Z_a + Z_d} \begin{bmatrix} (Z_a Z_c + Z_a Z_d + Z_c Z_d) & Z_a \\ -Z_a & 1 \end{bmatrix}$$

Montage en π :

g) π -Glied

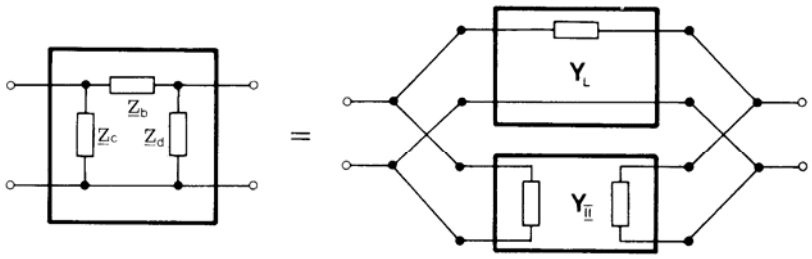


Fig. 19.59

$$Y_\pi = Y_L + Y_{II}$$

$$Y_\pi = \begin{bmatrix} (Y_b + Y_c) & -Y_b \\ -Y_b & (Y_b + Y_d) \end{bmatrix} \quad A_\pi = \frac{1}{Y_b} \begin{bmatrix} (Y_b + Y_d) & 1 \\ (Y_b Y_c + Y_b Y_d + Y_c Y_d) & (Y_b + Y_c) \end{bmatrix} \qquad (19.102)$$

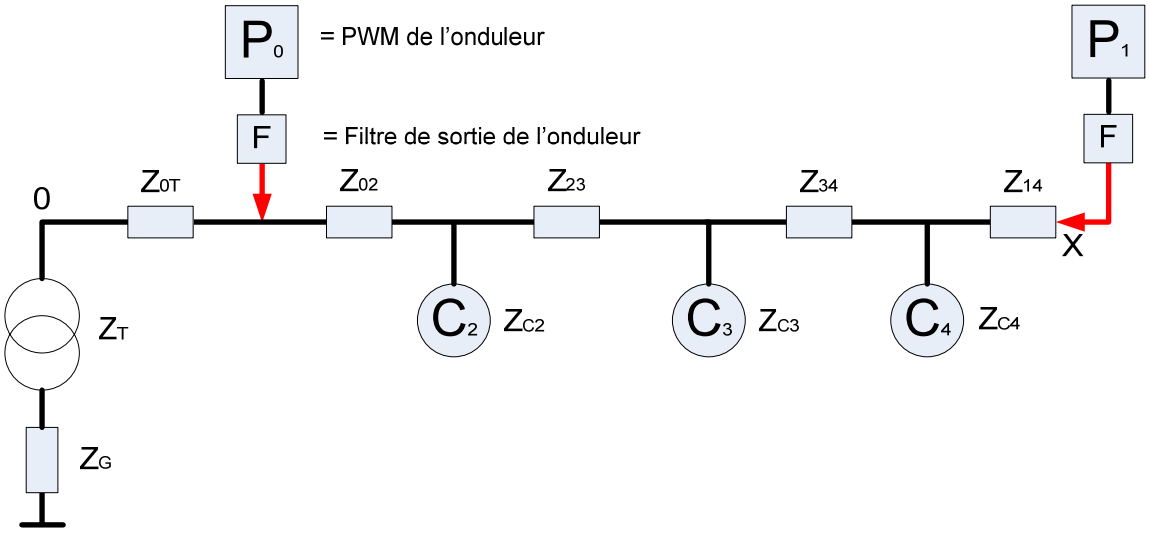
Z_π und H_π bilde man dual zum Fall e).

Nous pensons qu'en combinant ces « Kettenmatrix » il est possible de modéliser tout notre réseau. L'impédance de ligne est un montage en π et l'impédance d'un consommateur est une impédance en parallèle.

6.2 Améliorations possibles

6.2.1 Modifications de l'aspect graphique du programme

Afin de compléter notre modèle il faut ajouter l'impédance d'un transformateur et l'impédance du réseau qui se trouve derrière se transformateur. Nous devons également ajouter un filtre à la sortie de chaque producteur. En cliquant sur le filtre, nous pouvons modifier ses caractéristiques. L'affichage des impédances de ligne sur la fenêtre principale peut être rajouté.



6.2.2 Lecture/sauvegarde de fichiers

Cette action fonctionne bien uniquement si le nombre de colonnes et de lignes contenu dans le fichier est connu. Il faut modifier le programme afin qu'il soit capable de lire et interpréter n'importe quel signal.

6.2.3 Compensation de la sonde de mesure

Il est envisageable d'ajouter une compensation du rapport de la sonde de mesure qui a fait l'acquisition. Une simple multiplication du signal mesuré par l'inverse du rapport suffit.

6.3 Conclusion générale

Le but du projet qui consistait à développer un outil qui permet d'avancer dans l'étude du projet IGOR a été atteint. Il faut encore travailler sur le programme afin d'implémenter la partie équilibrage du réseau et de le finaliser. Le résultat global du projet est satisfaisant.

6.4 Remerciements

Pour conclure, je souhaite remercier mon responsable de projet Monsieur Gilbert-André Morand avec qui j'ai collaboré pendant toute la durée de ce travail de diplôme et le service informatique qui était toujours présent les nombreuses fois où mon pc criait au secours.

7 Bibliographie

- Eléments d'analyse spectrale numérique, document internet par Freddy Mudry
- Technische Elektrizitätslehre 3 1.Teil von Prof.Dr.G. Epprecht, 1973
- www.siteduzero.com, pour les tutoriels C et C++

8 Annexes

Annexe 1	:	Fiche technique de câble
Annexe 2	:	Graphique déphasage
Annexe 3	:	Code « bilanMain »
Annexe 4	:	Code « bilanFFT »
Annexe 5	:	Code « bilanDataP »
Annexe 6	:	Code « bilanDataC »
Annexe 7	:	Code « bilanTools »
Annexe 8	:	Code « fourier.c »
Annexe 9	:	Code « fft.c »

Date et signature

Sion, le 06 juillet 2009

Anthony Veuthey

Annexe 1

Fiche technique câble

GKN 4-Leiter NS-Polymerkabel 1/0.6kV

Ceanderkabel

Aufbau

- Kupferleiter bis 10 mm² eindrähtig, ab 16 mm² verseilt
- Leiterisolation aus EPR, vernetzt
- drei isolierte Leiter miteinander verseilt
- Polster aus Gummiregenerat
- konzentrischer Aussenleiter aus Kupferdrähten mit Kupferwendel
- Aussenmantel aus PE, halogenfrei, schwarz mit zwei gelben Längsstreifen
- Aderfarben: 3x6/6 mm², 3x10/10 mm², 2LN oder 3L ab 3x16/16 mm², 3L

Anwendung

In Verteilnetzen und Industrieanlagen.
Verlegung in Rohranlagen, in Innenräumen, Kabelkanälen und im Erdreich.
Der PE-Mantel garantiert sehr gute Isolationswerte im Betrieb, ist verschleissfest und damit optimal für die Verlegung.
Das entsprechende Zubehör finden Sie im Kapitel Zubehör.

Normen

HD 603, Part 7, Section E (2004)

GKN

Technische Daten

Querschnitt mm²	Durchmesser mm	Gewicht kg/100 m	Wechselstromwiderstand bei 60 °C und 50 Hz Ω/km	Reaktanz bei 50 Hz Ω/km	Impedanz bei 60 °C und 50 Hz Ω/km	Kapazität bei 50 Hz µF/km	min. Biegeradius bei Verlegung mm	min. Biegeradius bei Installation mm	max. zulässige Zugkraft kN
3x6/6	17	48	3.564	0.085	3.564	0.136	200	150	0.7
3x10/10	19	69	2.118	0.080	2.120	0.146	200	150	1.2
3x16/16	22	97	1.331	0.075	1.333	0.155	250	150	1.9
3x25/25	26	145	0.842	0.075	0.845	0.158	300	200	3.0
3x50/50	32	257	0.449	0.072	0.455	0.166	350	200	6.0
3x95/95	41	455	0.225	0.070	0.236	0.174	450	250	11.4
3x150/150	50	703	0.146	0.070	0.162	0.174	550	300	18.0
3x240/240	62	1090	0.091	0.069	0.114	0.175	650	400	28.8

Belastbarkeit

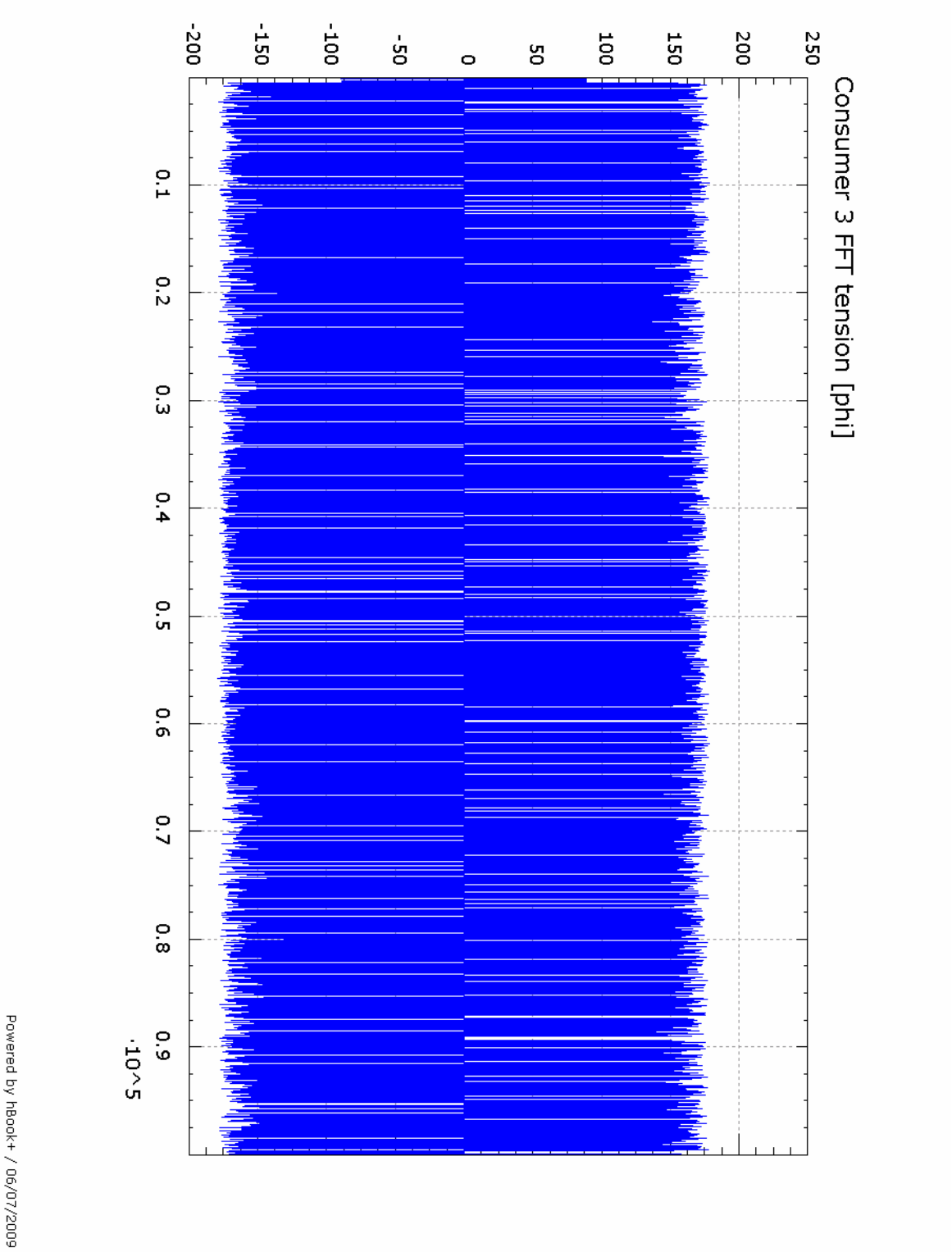
Verlegung	im Rohr in Erde					in Luft		
Betriebsart	Dauerlast		Industrielast		Notbetrieb[4]	Dauer- oder Industrielast		Notbetrieb[4]
Leitertemperatur	60 °C	90 °C	60 °C	90 °C	110 °C	60 °C	90 °C	110 °C
Erdung	⏚	⏚	⏚	⏚	⏚	⏚	⏚	⏚
Querschnitt mm²	A	A	A	A	A	A	A	A
3x6/6	38	49	39	51	57	38	55	63
3x10/10	52	67	53	69	77	53	75	86
3x16/16	68	88	71	91	102	71	101	116
3x25/25	89	115	93	119	133	94	134	153
3x50/50	130	168	135	175	195	140	199	229
3x95/95	196	254	206	266	297	218	312	358
3x150/150	252	326	266	344	384	287	409	470
3x240/240	333	432	353	459	514	385	552	634

[4] Notbetrieb während höchstens 8h/Tag und 100h/Jahr (Rohrtemperatur darf 50 °C übersteigen)
Angaben über Spannungsabfall, Transport, Verlegung, Montage und Prüfungen siehe Kapitel "Technische Informationen"

BRUGG CABLES Well connected.
ENERGIESYSTEME
Telefon +41 (0)56 460 33 33 • Fax +41 (0)56 460 34 83 • info.energiekabel@brugg.com

Annexe 2

Graphique déphasage


Consumer 3 FFT tension [phi]
250
200
150
100
50
0
-50
-100
-150
-200
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
.10^5
Powered by hBook+ / 06/07/2009

Annexe 3

Code « bilanMain »

```
/*****************************************************************************
 * This file defines all functions and actions linked with the main window of
 * the program
 *****************************************************************************/

#include <vcl.h>
#include "stdio.h"
#include "memory.h"
#include "Math.hpp"
#pragma hdrstop

#include "bilanMain.h"
#include "bilanTools.h"
#include "bilanDataP.h"
#include "bilanDataC.h"
#include "bilanFFT.h"
#include <fourier.h>
#include <hbook.h>

//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TFormMain *FormMain;
TLabel *Labellong;
TLabel *Labeloketoile;
TLabel *LabelProdPower;
TLabel *LabelCalcImpedOk ;
TLabel *LabelCalcCourantOk;

int flagMove;
int Movex;
int Movey;
int lequel = -1;
int longueurMax ;
int realXValueRound;
int nbrProd,nbrCons,nbrTot;
int nbrProdOld,nbrConsOld ;
bool first = true ;  // these flags are used to make a thing once
bool first2=true ;
bool first3=true ;
bool first4=true ;
bool dispOk=false ;
bool dispSettings=false ;
float realXValue = 1.0 ;
float retoile,cetoile,letoile;
float ITot[50];
char filename[200] = "";

extern int nbrTot,nbrProd,nbrCons;
extern int nbrProdOld,nbrConsOld;
extern bool producerOk = false ;
extern bool consumerOk = false ;

//***************************************************************************//
//    this structure describ the impedance of line
//***************************************************************************//

struct ZLIGNE
{
long   NAllocated;
float zlr;
float *zli;
float *norme ;
};
struct ZLIGNE zligne[50][50];

//***************************************************************************//
//    this structure describ the current of line
//***************************************************************************//

struct ILIGNE
{
long   NAllocated;
float *ir;
float *ii;
float *norme ;
```

```
};
struct ILIGNE iligne[50][50];

//*************************************************************************//
//     this structure describ the shape (line) between every prod/cons and
//     the grid cable
//*************************************************************************//

struct LINE
{
TShape *Shape;
float x;
};
struct LINE line[50];

//*************************************************************************//
//     this structure describ a lot of label
//*************************************************************************//

struct LABEL
{
TLabel *Label;
};

struct LABEL labeld[50];
struct LABEL labelp[50];
struct LABEL labelf[50];
struct LABEL label[50];


//*************************************************************************//
//   We constrain a maximum of 50 consumers and producers
//*************************************************************************//

// au maximum on a 50 consomateur ou producteur.
// la mémoire des enregistrements doit ête allouée.
struct CONSUMER_PRODUCER consumer_producer[50];


//*************************************************************************//
//     This function save all points of every vector into every plots
//*************************************************************************//

int RemplirGraphique(int lequel,long N)
{
        int id,ir,i;
        char titre[200];

        id = lequel * 1000;
        if ( consumer_producer[lequel].type == 'c')
        {
        sprintf(titre,"Consumer %d Tension ",lequel);
        }
        else
        {
        sprintf(titre,"Producer %d Tension ",lequel);
        }
        ir = hbook1_(id+1,titre,N,0,N*consumer_producer[lequel].dt);  // Define the axis of plot
        ir = hset_histoLineStyle(id+1,"stick");
        ir = hreset_(id+1);
        for(i=0;i<N;i++) {
                ir = hfill_(id+1,(( float)i + 0.5 )* consumer_producer[lequel].dt,0.,
                consumer_producer[lequel].xrU[i]);
        }

        if ( consumer_producer[lequel].type == 'c')
        {
        sprintf(titre,"Consumer %d courant",lequel);
        }
        else
        {
        sprintf(titre,"Producer %d courant",lequel);
        }
        ir = hbook1_(id+2,titre,N,0,N*consumer_producer[lequel].dt);
        ir = hset_histoLineStyle(id+2,"stick");
        ir = hreset_(id+2);
```

```
	for(i=0;i<N;i++) {
		ir = hfill_(id+2,(( float)i + 0.5 )* consumer_producer[lequel].dt,0.,
		consumer_producer[lequel].xrI[i]);
	}

	if ( consumer_producer[lequel].FFTsaved == true )
	{

	if ( consumer_producer[lequel].type == 'c')
	{
	sprintf(titre,"Consumer %d FFT tension [norme]",lequel);
	}
	else
	{
	sprintf(titre,"Producer %d FFT tension [norme]",lequel);
	}
	ir = hbook1_(id+10,titre,N/2,0,1./(2.*consumer_producer[lequel].dt));
	ir = hset_histoLineStyle(id+10,"stick");
	ir = hreset_(id+10);
	for(i=0;i<N/2;i++) {
		ir = hfill_(id+10,consumer_producer[lequel].f[i],0.,
		consumer_producer[lequel].normeU[i]);
	}

	if ( consumer_producer[lequel].type == 'c')
	{
	sprintf(titre,"Consumer %d FFT tension [phi]",lequel);
	}
	else
	{
	sprintf(titre,"Producer %d FFT tension [phi]",lequel);
	}
	ir = hbook1_(id+11,titre,N/2,0,1./(2.*consumer_producer[lequel].dt));
	ir = hset_histoLineStyle(id+11,"stick");
	ir = hreset_(id+11);
	for(i=0;i<N/2;i++) {
		ir = hfill_(id+11,consumer_producer[lequel].f[i],0.,
		consumer_producer[lequel].phiU[i]);
	}

	if ( consumer_producer[lequel].type == 'c')
	{
	sprintf(titre,"Consumer %d FFT courant [norme]",lequel);
	}
	else
	{
	sprintf(titre,"Producer %d FFT courant [norme]",lequel);
	}
	ir = hbook1_(id+20,titre,N/2,0,1./(2.*consumer_producer[lequel].dt));
	ir = hset_histoLineStyle(id+20,"stick");
	ir = hreset_(id+20);
	for(i=0;i<N/2;i++) {
		ir = hfill_(id+20,consumer_producer[lequel].f[i],0.,
		consumer_producer[lequel].normeI[i]);
	}

	if ( consumer_producer[lequel].type == 'c')
	{
	sprintf(titre,"Consumer %d FFT courant [phi]",lequel);
	}
	else
	{
	sprintf(titre,"Producer %d FFT courant [phi]",lequel);
	}
	ir = hbook1_(id+21,titre,N/2,0,1./(2.*consumer_producer[lequel].dt));
	ir = hset_histoLineStyle(id+21,"stick");
	ir = hreset_(id+21);
	for(i=0;i<N/2;i++) {
		ir = hfill_(id+21,consumer_producer[lequel].f[i],0.,
		consumer_producer[lequel].phiI[i]);
	}

	if ( consumer_producer[lequel].type == 'c' )
	{
	      sprintf(titre,"Consumer %d Impedance [norme]",lequel);
	      ir = hbook1_(id+30,titre,N/2,0,1./(2.*consumer_producer[lequel].dt));
```

```
              ir = hset_histoLineStyle(id+30,"stick");
              ir = hreset_(id+30);
              for(i=0;i<N/2;i++) {
                ir = hfill_(id+30,consumer_producer[lequel].f[i],0.,
                consumer_producer[lequel].normeZ[i]);
              }
        }

        }

        return(0);
}

//***************************************************************************//
//     This function save all plots into the "hbook6" program
//***************************************************************************//

int SavePlot(int lequel)
{
        char extension[20]= "" ;
        char directory[100] = "";
        char name[50] = "";
        AnsiString lequelString ;
        int id ;
        id = lequel * 1000;


        // sauve les plots sur disque
        sprintf(extension,".bin") ;
        sprintf (directory,"D:\\CP") ;
        lequelString = IntToStr(lequel);
        strcat(directory,lequelString.c_str());
        strcat(directory,extension);

        FILE *lunBin = NULL;
        int ir;
        sprintf(filename,directory);
        lunBin = fopen(filename,"wb");
        if(lunBin == NULL)
        {
                 ShowMessage("Error : can't save the file");
        }
        else
        {

                ir = hsave_(id+1,lunBin);
                ir = hsave_(id+2,lunBin);
                if ( consumer_producer[lequel].FFTsaved == true )
                {
                ir = hsave_(id+10,lunBin);
                ir = hsave_(id+11,lunBin);
                ir = hsave_(id+20,lunBin);
                ir = hsave_(id+21,lunBin);
                if ( consumer_producer[lequel].type == 'c' )
                {
                ir = hsave_(id+30,lunBin);
                }
                }
        }
        fclose(lunBin);
        return(0);
}

//***************************************************************************//
//       Free memory for the producer/consumer settings
//***************************************************************************//

int LibererMemoireConsumer_producer(int i,bool FreeFFT)
{

if(consumer_producer[lequel].NAllocated == 0) return(0);
if(i>=50) return(-1);
if(i<  0) return(-1);

// ................
```

```
if (FreeFFT == false )
{
if(consumer_producer[i].t != NULL) {free(consumer_producer[i].t);
                                   consumer_producer[i].t = NULL;}
if(consumer_producer[i].xrU != NULL) {free(consumer_producer[i].xrU);
                                   consumer_producer[i].xrU = NULL;}
if(consumer_producer[i].xrI != NULL) {free(consumer_producer[i].xrI);
                                   consumer_producer[i].xrI = NULL;}
}
else
{
if(consumer_producer[i].f != NULL) {free(consumer_producer[i].f);
                                   consumer_producer[i].f = NULL;}
if(consumer_producer[i].XrU != NULL) {free(consumer_producer[i].XrU);
                                   consumer_producer[i].XrU = NULL;}
if(consumer_producer[i].XiU != NULL) {free(consumer_producer[i].XiU);
                                   consumer_producer[i].XiU = NULL;}
if(consumer_producer[i].XrI != NULL) {free(consumer_producer[i].XrI);
                                   consumer_producer[i].XrI = NULL;}
if(consumer_producer[i].XiI != NULL) {free(consumer_producer[i].XiI);
                                   consumer_producer[i].XiI = NULL;}
if(consumer_producer[i].phiU!= NULL) {free(consumer_producer[i].phiU);
                                   consumer_producer[i].phiU = NULL;}
if(consumer_producer[i].phiI!= NULL) {free(consumer_producer[i].phiI);
                                   consumer_producer[i].phiI = NULL;}
if(consumer_producer[i].normeU!= NULL) {free(consumer_producer[i].normeU);
                                   consumer_producer[i].normeU = NULL;}
if(consumer_producer[i].normeI!= NULL) {free(consumer_producer[i].normeI);
                                   consumer_producer[i].normeI = NULL;}
if(consumer_producer[i].Zr != NULL) {free(consumer_producer[i].Zr);
                                   consumer_producer[i].Zr = NULL;}
if(consumer_producer[i].Zi != NULL) {free(consumer_producer[i].Zi);
                                   consumer_producer[i].Zi = NULL;}
if(consumer_producer[i].normeZ != NULL) {free(consumer_producer[i].normeZ);
                                   consumer_producer[i].normeZ = NULL;}
if(consumer_producer[i].phiZ != NULL) {free(consumer_producer[i].phiZ);
                                   consumer_producer[i].phiZ = NULL;}
if(consumer_producer[i].ITotI != NULL) {free(consumer_producer[i].ITotI);
                                   consumer_producer[i].ITotI = NULL;}
if(consumer_producer[i].ITotR != NULL) {free(consumer_producer[i].ITotR);
                                   consumer_producer[i].ITotR = NULL;}
}
consumer_producer[i].NAllocated = 0;
return(0);

}

//*************************************************************************//
//    Allocate memory for the producer/consumer settings
//*************************************************************************//

int AllouerMemoireConsumer_producer(int N,int i,bool AllocateFFT)
{
if(i>=50) return(-1);
if(i<  0) return(-1);

// ..............
if ( AllocateFFT == false )
{
        consumer_producer[i].t    =(float   *) malloc(N * sizeof(float));
        consumer_producer[i].xrU    =(float   *) malloc(N * sizeof(float));
        consumer_producer[i].xrI    =(float   *) malloc(N * sizeof(float));
        memset(consumer_producer[i].t,0, N * sizeof(float));
        memset(consumer_producer[i].xrU,0, N * sizeof(float));
        memset(consumer_producer[i].xrI,0, N * sizeof(float));
}
else
{
consumer_producer[i].f    =(float   *) malloc(N * sizeof(float));
consumer_producer[i].XrU    =(float   *) malloc(N * sizeof(float));
consumer_producer[i].XiU    =(float   *) malloc(N * sizeof(float));
consumer_producer[i].XrI    =(float   *) malloc(N * sizeof(float));
consumer_producer[i].XiI    =(float   *) malloc(N * sizeof(float));
consumer_producer[i].phiU   =(float   *) malloc(N * sizeof(float));
consumer_producer[i].phiI   =(float   *) malloc(N * sizeof(float));
consumer_producer[i].normeU =(float   *) malloc(N * sizeof(float));
```

```
consumer_producer[i].normeI =(float   *) malloc(N * sizeof(float));
consumer_producer[i].Zr    =(float   *) malloc(N * sizeof(float));
consumer_producer[i].Zi    =(float   *) malloc(N * sizeof(float));
consumer_producer[i].normeZ    =(float   *) malloc(N * sizeof(float));
consumer_producer[i].phiZ    =(float   *) malloc(N * sizeof(float));
consumer_producer[i].ITotI    =(float   *) malloc(N/2 * sizeof(float));
consumer_producer[i].ITotR    =(float   *) malloc(N/2 * sizeof(float));

memset(consumer_producer[i].f,0, N * sizeof(float));
memset(consumer_producer[i].XrU,0, N * sizeof(float));
memset(consumer_producer[i].XiU,0, N * sizeof(float));
memset(consumer_producer[i].XrI,0, N * sizeof(float));
memset(consumer_producer[i].XiI,0, N * sizeof(float));
memset(consumer_producer[i].phiU,0, N * sizeof(float));
memset(consumer_producer[i].phiI,0, N * sizeof(float));
memset(consumer_producer[i].normeU,0, N * sizeof(float));
memset(consumer_producer[i].normeI,0, N * sizeof(float));
memset(consumer_producer[i].Zr,0, N * sizeof(float));
memset(consumer_producer[i].Zi,0, N * sizeof(float));
memset(consumer_producer[i].normeZ,0, N * sizeof(float));
memset(consumer_producer[i].phiZ,0, N * sizeof(float));
memset(consumer_producer[i].ITotI,0, N/2 * sizeof(float));
memset(consumer_producer[i].ITotR,0, N/2 * sizeof(float));
}

consumer_producer[i].NAllocated = N;

return(0);
}

//***************************************************************************//
//    Allocate memory for the impedance of line matrix
//***************************************************************************//

int AllouerMemoireImpedance(int N,int i,int j)
{
if(i>=50) return(-1);
if(i<  0) return(-1);
if(j>=50) return(-1);
if(j<  0) return(-1);

//..............
zligne[i][j].zli =(float   *) malloc(N/2 * sizeof(float));
zligne[i][j].norme =(float   *) malloc(N/2 * sizeof(float));
memset(zligne[i][j].zli,0, N/2 * sizeof(float));
memset(zligne[i][j].norme,0, N/2 * sizeof(float));

zligne[i][j].NAllocated = N/2;
return (0);
}

//***************************************************************************//
//     Free the memory for the impedance of line matrix when it's necessary
//***************************************************************************//

int LibererMemoireImpedance(int i,int j)
{

if(zligne[i][j].NAllocated == 0) return(0);
if(i>=50) return(-1);
if(i<  0) return(-1);
if(j>=50) return(-1);
if(j<  0) return(-1);

// ..............
if(zligne[i][j].zli != NULL) {free(zligne[i][j].zli );zligne[i][j].zli = NULL;}
if(zligne[i][j].norme != NULL) {free(zligne[i][j].norme );
                        zligne[i][j].norme = NULL;}

zligne[i][j].NAllocated = 0;
return (0);
}

//***************************************************************************//
//       Allocate memory for the current matrix
//***************************************************************************//
```

```
int AllouerMemoireCourant(int N,int i,int j)
{
if(i>=50) return(-1);
if(i<  0) return(-1);
if(j>=50) return(-1);
if(j<  0) return(-1);

//..............
iligne[i][j].ir =(float   *) malloc(N/2 * sizeof(float));
iligne[i][j].ii =(float   *) malloc(N/2 * sizeof(float));
iligne[i][j].norme =(float   *) malloc(N/2 * sizeof(float));
memset(iligne[i][j].ir,0, N/2 * sizeof(float));
memset(iligne[i][j].ii,0, N/2 * sizeof(float));
memset(iligne[i][j].norme,0, N/2 * sizeof(float));

iligne[i][j].NAllocated = N/2;
return (0);
}

//***************************************************************************//
//      Free the memory for the current matrix when it's necessary
//***************************************************************************//

int LibererMemoireCourant(int i,int j)
{

if(iligne[i][j].NAllocated == 0) return(0);
if(i>=50) return(-1);
if(i<  0) return(-1);
if(j>=50) return(-1);
if(j<  0) return(-1);

// ................
if(iligne[i][j].ir != NULL) {free(iligne[i][j].ir );iligne[i][j].ir = NULL;}
if(iligne[i][j].ii != NULL) {free(iligne[i][j].ii );iligne[i][j].ii = NULL;}
if(iligne[i][j].norme != NULL) {free(iligne[i][j].norme );
                        iligne[i][j].norme = NULL;}

iligne[i][j].NAllocated = 0;
return (0);
}

float max(float a,float b)
{
if(a>=b)
return a;
else
return b;
}

//***************************************************************************//
//       Make an action when the program is launch
//***************************************************************************//

__fastcall TFormMain::TFormMain(TComponent* Owner)
        : TForm(Owner)
{
        Run2->Enabled = false  ;
}

//***************************************************************************//
//      Free the memory when the user quit the program
//***************************************************************************//

void __fastcall TFormMain::Quitter1Click(TObject *Sender)
{
        int i,j;
        for(i=0;i<50;i++) {
                LibererMemoireConsumer_producer(i,true);
                LibererMemoireConsumer_producer(i,false);
                for (j=0;j<50;j++){
                LibererMemoireImpedance(i,j);
                }
        }
        Close();
```

```
}

//---------------------------------------------------------------------------//
//       Display the "Tools" window
//---------------------------------------------------------------------------//

void __fastcall TFormMain::Tools1Click(TObject *Sender)
{
        Tools->Show();
}

//---------------------------------------------------------------------------//
//    Create the shapes corresponding as the number of prod and cons
//---------------------------------------------------------------------------//

void __fastcall TTools::ButtonInsertClick(TObject *Sender)
{
        int i ;
        int nbrTotOld;

        nbrProdOld=nbrProd;
        nbrConsOld=nbrCons;
        nbrTotOld = nbrProdOld+nbrConsOld ;


        nbrProd = StrToInt(EditProd->Text);
        nbrCons = StrToInt(EditCons->Text);
        nbrTot = nbrProd+nbrCons;
        if (nbrProd == 0 && nbrCons==0 )
        {
        ShowMessage("Entrez un nombre de producteurs/consommateurs svp !");
        }

        // Clear the old Shapes

        if ( first == false )
        {

        for (i=0;i<nbrTotOld;i++)
        {
                delete consumer_producer[i].Shape;
                delete line[i].Shape;
                delete label[i].Label;
                if ( dispOk == true )
                {
                        delete labeld[i].Label;
                }
                producerOk = false ;
                consumerOk = false ;
        }

        }

        // Create the producer's Shapes

        for(i=0;i<nbrProd;i++)
        {

        consumer_producer[i].Shape = new TShape(this);
        line[i].Shape = new TShape(this);
        consumer_producer[i].Shape->Parent = FormMain;
        line[i].Shape->Parent = FormMain;

        //set all the appearance properties
        consumer_producer[i].Shape->Shape = stSquare;
        consumer_producer[i].Shape->Brush->Color = clRed;
        consumer_producer[i].Shape->Left = 100 + 100 * i ;
        consumer_producer[i].Shape->Top = 246;
        consumer_producer[i].Shape->Width = 36;
        consumer_producer[i].Shape->Height = 36;
        consumer_producer[i].x = consumer_producer[i].Shape->Left;
        consumer_producer[i].y = consumer_producer[i].Shape->Top;
        consumer_producer[i].type = 'p' ;

        line[i].Shape->Shape = stRectangle;
        line[i].Shape->Brush->Color = clBlack;
```

```
        line[i].Shape->Left = consumer_producer[i].Shape->Left+16;
        line[i].Shape->Top = consumer_producer[i].Shape->Top+consumer_producer[i].Shape->Height;
        line[i].Shape->Width = 3;
        line[i].Shape->Height = (328-(consumer_producer[i].Shape->Top+consumer_producer[i].Shape->W

        label[i].Label = new TLabel(this);
        label[i].Label->Parent = FormMain ;
        label[i].Label->Left = consumer_producer[i].Shape->Left+12.5 ;
        label[i].Label->Top = consumer_producer[i].Shape->Top+7 ;
        label[i].Label->Transparent = true ;
        label[i].Label->Font->Size = 14 ;
        label[i].Label->Caption = i ;

        producerOk = true ;
        }

        // Create the consumer's Shapes

        for(i=nbrProd;i<nbrTot;i++)
        {

        consumer_producer[i].Shape = new TShape(this);
        line[i].Shape = new TShape(this);
        consumer_producer[i].Shape->Parent = FormMain;
        line[i].Shape->Parent = FormMain;

        //set all the appearance properties
        consumer_producer[i].Shape->Shape = stCircle;
        consumer_producer[i].Shape->Brush->Color = clBlue;
        consumer_producer[i].Shape->Left = 150 + 100 * (i-nbrProd) ;
        consumer_producer[i].Shape->Top = 376;
        consumer_producer[i].Shape->Width = 36;
        consumer_producer[i].Shape->Height = 36;
        consumer_producer[i].x = consumer_producer[i].Shape->Left ;
        consumer_producer[i].y = consumer_producer[i].Shape->Top ;
        consumer_producer[i].type = 'c';

        line[i].Shape->Shape = stRectangle;
        line[i].Shape->Brush->Color = clBlack;
        line[i].Shape->Left = consumer_producer[i].Shape->Left+16;
        line[i].Shape->Top = 337;
        line[i].Shape->Width = 3;
        line[i].Shape->Height = (consumer_producer[i].Shape->Top-337);

        label[i].Label = new TLabel(this);
        label[i].Label->Parent = FormMain ;
        label[i].Label->Left = consumer_producer[i].Shape->Left+12.5 ;
        label[i].Label->Top = consumer_producer[i].Shape->Top+7 ;
        label[i].Label->Transparent = true ;
        label[i].Label->Font->Size = 14 ;
        label[i].Label->Font->Color = clWhite ;
        label[i].Label->Caption = i ;

        consumerOk = true ;

        }


        dispOk=false ;
        first=false ;
}

//***************************************************************************//
//    This function saves and displays the lenght of the grid cable.
//    It updates the distances immediately too.
//***************************************************************************//

void __fastcall TFormMain::ButtonOKClick(TObject *Sender)
{
        int i ;
        longueurMax = StrToInt(longMax->Text);
        if ( longueurMax <= 0 )
        {
                ShowMessage("Improbably lenght!");
        }
        else
```

```
        {

                Tools->Left = FormMain->Left + 150 ;
                Tools->Top = FormMain->Top + 150 ;
                Tools->Show();

                if ( first2 == true )
                {
                        Labellong = new TLabel(this);
                        Labellong->Parent = FormMain ;
                }
                Labellong->Font->Size = 12 ;
                Labellong->Left= 1017;
                Labellong->Top = 304 ;
                Labellong->Caption=longueurMax;
                Labellong->Font->Color = clGreen ;

                // Updates the lengths immediately

                if ( dispOk == true )
                {
                        for (i=0;i<nbrTot;i++)
                        {
                        realXValue = ((consumer_producer[i].x-9)*longueurMax)/1025;
                        realXValueRound = RoundTo(realXValue,0);

                        labeld[i].Label->Font->Size = 12;
                        labeld[i].Label->Left = consumer_producer[i].x;
                        labeld[i].Label->Transparent = true ;

                        //set all the appearance properties

                        labeld[i].Label->Caption = realXValueRound ;
                        realXValue = 0 ;
                        }
                        for (i=0;i<nbrProd;i++)
                        {
                        labeld[i].Label->Top = consumer_producer[i].y-20;
                        }
                        for (i=nbrProd;i<nbrTot;i++)
                        {
                        labeld[i].Label->Top = consumer_producer[i].y+35;
                        }

                }
                first2 = false ;
        }
}

//***************************************************************************//
//   This function displays or hides the distance between each producers or
//   consumers and the reference point.
//***************************************************************************//


void __fastcall TFormMain::ButtonDisplayClick(TObject *Sender)
{

        int i ;
        realXValue = 0 ;
        Run2->Enabled = true ;

        if (nbrProd == 0 && nbrCons == 0 )
        {
                ShowMessage("Entry a number of prod/cons.");
        }
        else if ( dispOk == false )
        {
                // Displays the distance between Producers/Consumers
                // and the reference

                ButtonDisplay->Caption = "Hide";

                for (i=0;i<nbrTot;i++)
                {
```

```
                realXValue = ((consumer_producer[i].x-9)*longueurMax)/1025;
                realXValueRound = RoundTo(realXValue,0);
                consumer_producer[i].distance = realXValueRound ;

                        if (nbrProdOld != nbrProd || nbrConsOld != nbrCons)
                        {
                        labeld[i].Label = new TLabel(this);
                        labeld[i].Label->Parent = FormMain;
                        labeld[i].Label->Font->Size = 12;
                        labeld[i].Label->Left = consumer_producer[i].x;
                        labeld[i].Label->Transparent = true ;
                        }

                labeld[i].Label->Caption = realXValueRound ;
                realXValue = 0 ;
                }
                if (nbrProdOld != nbrProd || nbrConsOld != nbrCons)
                {
                for (i=0;i<nbrProd;i++)
                {
                        labeld[i].Label->Top = consumer_producer[i].y-20;
                }
                for (i=nbrProd;i<nbrTot;i++)
                {
                        labeld[i].Label->Top = consumer_producer[i].y+35;
                }
                }

                dispOk = true ;
        }
        else if ( dispOk == true )
        {
                for (i=0;i<nbrTot;i++)
                {
                        delete labeld[i].Label;
                }
                dispOk = false ;
                ButtonDisplay->Caption = "Display";
        }

}

//---------------------------------------------------------------------------//
//    This function is called once when the user move the mouse on the
//    main window and after he have inserted the producers and consumers
//---------------------------------------------------------------------------//


void __fastcall TFormMain::FormMouseMove(TObject *Sender, TShiftState Shift,
          int X, int Y)
{
        int i ;
        if ( producerOk == true || consumerOk == true )
        {
                for (i=0;i<nbrTot;i++)
                {
                consumer_producer[i].Shape->OnMouseDown = MyShapeMouseDown;
                consumer_producer[i].Shape->OnMouseUp = MyShapeMouseUp;
                consumer_producer[i].Shape->OnMouseMove = MyShapeMouseMove;
                label[i].Label->OnMouseDown = MyShapeMouseDown;
                label[i].Label->OnMouseUp = MyShapeMouseUp;
                label[i].Label->OnMouseMove = MyShapeMouseMove;
                producerOk=false ;
                consumerOk=false ;
                }
        }
}

//---------------------------------------------------------------------------//
//     Enter in this function when the user clic on a Shape ( p or c )
//---------------------------------------------------------------------------//

void __fastcall TFormMain::MyShapeMouseDown(TObject *Sender,
      TMouseButton Button, TShiftState Shift, int X, int Y)
{
        int i ;
```

```
         for(i=0;i<nbrTot;i++)
        {

        // Define on which shape you have make the clic

        if(consumer_producer[i].Shape == Sender || label[i].Label == Sender)
        {
                lequel = i;
                break;
        }
        }

        // Check if it was a left clic or right clic

        if((GetAsyncKeyState(VK_LBUTTON) & 0x8000) != 0)
        {
                // Left button clic
                flagMove=1;
                Movex=X;
        }
        else if (consumer_producer[lequel].type == 'c')   // Right button clic
        {
                DataC->ShowModal();
        }
        else
        {
                DataP->ShowModal();
        }

}

//***************************************************************************//
//     Enter in the function when you move the mouse with a left clic
//     on the shape
//***************************************************************************//

void __fastcall TFormMain::MyShapeMouseMove(TObject *Sender, TShiftState Shift,
          int X, int Y)
{

        int i ;
        Run2->Enabled = true ;
        if ( flagMove == 1 )
        {

                realXValue = ((consumer_producer[lequel].x-9)*longueurMax)/1025;
                realXValueRound = RoundTo(realXValue,0);

                // Forbid the User to place Prod or Cons out of the Grid limit

                if ( realXValueRound > 0 && realXValueRound < longueurMax )
                {
                consumer_producer[lequel].Shape->Left += X-Movex;
                consumer_producer[lequel].x=consumer_producer[lequel].Shape->Left;
                consumer_producer[lequel].distance = realXValueRound ;
                line[lequel].Shape->Left += X-Movex;
                label[lequel].Label->Left += X-Movex;

                first3 = false ;
                }
                else if (realXValueRound <=1 )
                {
                        if ( X-Movex <=0 )
                        {
                        // Don't move
                        }
                        else
                        {
                        consumer_producer[lequel].Shape->Left += X-Movex;
                        consumer_producer[lequel].x=consumer_producer[lequel].Shape->Left;
                        consumer_producer[lequel].distance = realXValueRound ;
                        line[lequel].Shape->Left += X-Movex;
                        label[lequel].Label->Left += X-Movex;

                        }
                }
```

```
                else if ( X-Movex >=0 )
                {
                // Don't move
                }
                else
                {
                consumer_producer[lequel].Shape->Left += X-Movex;
                consumer_producer[lequel].x=consumer_producer[lequel].Shape->Left;
                consumer_producer[lequel].distance = realXValueRound ;
                line[lequel].Shape->Left += X-Movex;
                label[lequel].Label->Left += X-Movex;

                }
                if (dispOk == true )
                {
                // Move the distance and settings Labels with the shape

                if ( consumer_producer[lequel].type == 'c')
                {
                labeld[lequel].Label->Top = consumer_producer[lequel].y+35;
                }
                else
                {
                labeld[lequel].Label->Top = consumer_producer[lequel].y-20;
                }
                labeld[lequel].Label->Left = consumer_producer[lequel].x;
                labeld[lequel].Label->Caption = realXValueRound ;
                }
                if (dispSettings == true )
                {
                if ( consumer_producer[lequel].saved == false )
                {
                labelp[lequel].Label->Left = consumer_producer[lequel].x-35;
                }
                else
                {
                labelf[lequel].Label->Left = consumer_producer[lequel].x-25;
                }
                }
        }
}

//***************************************************************************//
//     When you release the left clic then this function is call
//***************************************************************************//

void __fastcall TFormMain::MyShapeMouseUp(TObject *Sender, TMouseButton Button,
          TShiftState Shift, int X, int Y)
{
        flagMove=0;
        first3 = true ;
}

//***************************************************************************//
//    Read and save the cable settings
//***************************************************************************//

void __fastcall TFormMain::ButtonAcceptClick(TObject *Sender)
{
        retoile = StrToFloat(Retoile->Text);
        letoile = StrToFloat(Letoile->Text);
        cetoile = StrToFloat(Cetoile->Text);

        if ( retoile <= 0 || letoile <= 0 || cetoile <= 0 )
        {
                ShowMessage("Improbably lenght!");
        }
        else
        {
                if ( first4 == true )
                {
                        Labeloketoile = new TLabel(this);
                        Labeloketoile->Parent = FormMain ;
                }
                Labeloketoile->Font->Size = 12 ;
                Labeloketoile->Left= 970;
```

```
                Labeloketoile->Top = 180 ;
                Labeloketoile->Caption="Datas saved";
                Labeloketoile->Font->Color = clGreen ;
        }
        first4 = false ;

}

//***************************************************************************//
//    Forbidden the user to enter a another key that numerical key
//***************************************************************************//

void __fastcall TFormMain::longMaxKeyPress(TObject *Sender, char &Key)
{
        if ((Key < 48 || Key > 57)
        && Key != VK_BACK
        && Key != VK_DELETE
        && Key != DecimalSeparator)
        Key = 0;
}

//***************************************************************************//
//    Forbidden the user to enter a another key that numerical key
//***************************************************************************//

void __fastcall TFormMain::RetoileKeyPress(TObject *Sender, char &Key)
{
        if ((Key < 48 || Key > 57)
        && Key != VK_BACK
        && Key != VK_DELETE
        && Key != DecimalSeparator)
        Key = 0;
        Labeloketoile->Caption="";

}

//***************************************************************************//
//    Forbidden the user to enter a another key that numerical key
//***************************************************************************//

void __fastcall TFormMain::LetoileKeyPress(TObject *Sender, char &Key)
{
        if ((Key < 48 || Key > 57)
        && Key != VK_BACK
        && Key != VK_DELETE
        && Key != DecimalSeparator)
        Key = 0;
        Labeloketoile->Caption="";
}

//***************************************************************************//
//   Forbidden the user to enter a another key that numerical key
//***************************************************************************//

void __fastcall TFormMain::CetoileKeyPress(TObject *Sender, char &Key)
{
        if ((Key < 48 || Key > 57)
        && Key != VK_BACK
        && Key != VK_DELETE
        && Key != DecimalSeparator)
        Key = 0;
        Labeloketoile->Caption="";
}

//***************************************************************************//
//  This function check if the producers and consumers datas are saved or not
//  and displays it.
//***************************************************************************//

void __fastcall TFormMain::ButtonDisplayCarClick(TObject *Sender)
{
        // Display the Producers and Consumers Settings

        int i ;

        if ( dispSettings == false )
```

```
        {
        ButtonDisplayCar->Caption = "Hide" ;
        dispSettings = true ;

        for ( i=0; i<nbrTot; i++ )
        {


                if ( consumer_producer[i].saved == false )
                {
                labelp[i].Label = new TLabel(this);
                labelp[i].Label->Parent = FormMain;

                labelp[i].Label->Font->Size = 12;
                labelp[i].Label->Left = consumer_producer[i].x-35;
                if ( consumer_producer[i].type == 'c' )
                {
                labelp[i].Label->Top = consumer_producer[i].y+55;
                }
                else
                {
                labelp[i].Label->Top = consumer_producer[i].y-45;
                }
                labelp[i].Label->Caption = "Datas not saved";
                }
                else
                {
                labelf[i].Label = new TLabel(this);
                labelf[i].Label->Parent = FormMain;

                labelf[i].Label->Font->Size = 12;
                labelf[i].Label->Font->Color = clGreen;
                labelf[i].Label->Left = consumer_producer[i].x-25;
                if ( consumer_producer[i].type == 'c' )
                {

                labelf[i].Label->Top = consumer_producer[i].y+55;
                }
                else
                {
                labelf[i].Label->Top = consumer_producer[i].y-45;
                }
                labelf[i].Label->Caption = "Datas saved";
                }

        }
        }
        else
        {
        ButtonDisplayCar->Caption = "Display" ;
        for ( i=0; i<nbrTot; i++ )
        {

                if (consumer_producer[i].saved == true )
                {
                        delete labelf[i].Label;
                }
                else
                {
                delete labelp[i].Label;
                }
        }
        dispSettings = false ;
        }
}

//***************************************************************************//
//  This function make a lot of things :
//  1. It calculate the impedance of line between every producers and consumers.
//  2. Then it calculate the currents in every point on the grid
//***************************************************************************//


void __fastcall TFormMain::Run2Click(TObject *Sender)
{
  int i,j,ir,k,l ;
```

```
  bool RunOk ;
  float dt ;
  float Distance;
  float DistanceMax,DistanceMaxOld = 0.0 ;

  //Check if the all datas are saved

  for ( i=0;i<nbrTot;i++)
  {
        if ( consumer_producer[i].saved == true )
        {
                RunOk = true ;
        }
        else
        {
                RunOk = false ;
        }
  }

  // if it's ok we can go further

  if ( RunOk == true  )
  {

  dt = consumer_producer[i].t[1]-consumer_producer[i].t[0];

  if ( Labellong->Caption == "" || Labeloketoile->Caption == "" )
  {
     ShowMessage("Error : The lenght of cable or cable skills are not saved");
  }
  else
  {

  LabelCalcImpedOk = new TLabel(this) ;
  LabelCalcImpedOk->Parent = FormMain ;

  LabelCalcCourantOk = new TLabel(this);
  LabelCalcCourantOk->Parent = FormMain;

// Create matrix with the producers for the lines and the consumers for columns

  for (i=0 ;i<nbrProd ; i++ )
  {
        for (j=nbrProd ; j<nbrTot ; j++ )
        {
           // Allocate memory for impedance line

           ir = AllouerMemoireImpedance(consumer_producer[i].N,i,j);

           //  Fill the impedance matrix

           calculLigne( consumer_producer[i].N,
                        dt,
                        zligne[i][j].zli,
                        zligne[i][j].zlr,
                        zligne[i][j].norme,
                        consumer_producer[i].distance,
                        consumer_producer[j].distance,
                        retoile,
                        letoile,
                        cetoile);

           LabelCalcImpedOk->Font->Size = 14 ;
           LabelCalcImpedOk->Left = 75 ;
           LabelCalcImpedOk->Top = 75 ;
           LabelCalcImpedOk->Font->Color = clGreen ;
           LabelCalcImpedOk->Caption = "Impedance of lines calulated" ;


           // Calcul the currents of line  and fill the matrix

           calculCourant (      consumer_producer[i].N,
                                iligne[i][j].ir,
                                iligne[i][j].ii,
                                iligne[i][j].norme,
                                zligne[i][j].zlr,
```

```
                                zligne[i][j].zli,
                                consumer_producer[i].XrU,
                                consumer_producer[i].XiU,
                                consumer_producer[j].Zr,
                                consumer_producer[j].Zi);

          LabelCalcCourantOk->Font->Size = 14 ;
          LabelCalcCourantOk->Left = 75 ;
          LabelCalcCourantOk->Top = 115 ;
          LabelCalcCourantOk->Font->Color = clGreen ;
          LabelCalcCourantOk->Caption = "Current of line calulated" ;

          // Calcul the sum of all current for each producer

          for ( k= 0 ; k< consumer_producer[i].N/2 ; k++ )
          {
                consumer_producer[i].ITotR[k] += iligne[i][j].ir[k] ;
                consumer_producer[i].ITotI[k] += iligne[i][j].ii[k] ;
                consumer_producer[j].ITotR[k] += iligne[i][j].ir[k] ;
                consumer_producer[j].ITotI[k] += iligne[i][j].ii[k] ;
          }
       }    // End of for(j)
  }     // End of for(i)
  }     // End of else
  }     // End of if
  else
  {
       ShowMessage("Error : save all producers/consumers files befor that");
  }
}
//---------------------------------------------------------------------------
```

Annexe 4

Code « bilanFFT »

```
/*****************************************************************************
 * This file defines all functions and actions linked with FFT window
 *****************************************************************************/

#include <vcl.h>
#pragma hdrstop
#include "stdio.h"
#include "Math.h"

#include "bilanMain.h"
#include "bilanFFT.h"
#include "bilanDataP.h"
#include <fourier.h>

//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
#define TWO_PI (28318530717958647692528676655900576839433L)
#define PI TWO_PI/2.0

TFormFFT *FormFFT;
TProgressBar *ProgressFFT;
TLabel *LabelFFTEnd ;

AnsiString Directory;
int  fenetre,sweep_on,simule_on ;
bool first = true ;

extern struct CONSUMER_PRODUCER consumer_producer[];
extern int lequel ;

TButton *ButtonDisplayP;
TCheckBox *CheckBoxSweepP;
TCheckBox *CheckBoxSimuleP;
TCheckBox *CheckBoxSweepC;
TCheckBox *CheckBoxSimuleC;

//***************************************************************************//
//    subroutine to make an action when the FFTwindow was called
//***************************************************************************//

__fastcall TFormFFT::TFormFFT(TComponent* Owner)
        : TForm(Owner)
{
}

//***************************************************************************//
//    close the window when the user clic on "quit" button
//***************************************************************************//

void __fastcall TFormFFT::ButtonQuitClick(TObject *Sender)
{
        Close();
}

//***************************************************************************//
//    The main function of this subroutine is to launch the FFT ! After that
//    it calculate the impedance of this consumer
//    The others lines is for graphics application
//***************************************************************************//

void __fastcall TFormFFT::ButtonStartClick(TObject *Sender)
{
        // Variables Declaration

        AnsiString  nomfftU ;
        AnsiString  nomfftI ;
        AnsiString SaveCopyUFFT ;
        AnsiString SaveCopyIFFT ;

        bool error,copy ;
        int er,ir;

        // Create a new Progress Bar

        ProgressFFT = new TProgressBar(this);
```

```
	ProgressFFT->Parent = FormFFT;

	// Define the Properties of the Progress Bar

	ProgressFFT->Left = 105;
	ProgressFFT->Top = 300;
	ProgressFFT->Width = 115;
	ProgressFFT->Height = 20;
	ProgressFFT->Smooth=true ;
	ProgressFFT->Min = 0;
	ProgressFFT->Max = 100;
	ProgressFFT->Step = 1 ;
	ProgressFFT->Position = 0;
	SendMessage(ProgressFFT->Handle,PBM_SETBARCOLOR,0,clGreen);

	// Receives the User's datas

	nomfftU = EditDirectoryU->Text ;
	nomfftI = EditDirectoryI->Text ;

	// Test if the User's datas are corrects

	if ( nomfftU == "" || nomfftI == "" )
	{
		 ShowMessage("Error : FFT Voltage/Current File Directory empty!");
		 error = true ;
	}
	else
	{
		error = false ;
	}

	if ( CheckBoxCopyFFT->Checked == true )
	{

	SaveCopyUFFT = EditDirectoryU->Text;
	SaveCopyIFFT = EditDirectoryI->Text;
	if ( SaveCopyUFFT == "" || SaveCopyIFFT == "")
	{
		ShowMessage("Error : empty case -> copy name !");
		error = -1;
	}
	copy = true ;
	}
	else
	{
	 copy = false ;
	 SaveCopyUFFT = "" ;
	 SaveCopyIFFT = "" ;

	}

	// Allocate memory

	ir =  LibererMemoireConsumer_producer(lequel,true);
	ir =  AllouerMemoireConsumer_producer(consumer_producer[lequel].N,lequel,true);

	// Run the FFT function

	if ( error == false )
	{
		er =FFT(sweep_on,  // = 1 -> run FFT with sweep signal
				   // = 0 ->  "   " without "     "
			simule_on, // = 1 -> run FFT with a programmed signal
				   // = 0 ->  "   "    "  "   mesured     "
			fenetre,   // = 1 -> Hann
				   // = 2 -> Blackmann
				   // = 3 -> Hamming
			copy,
			consumer_producer[lequel].dt,
			nomfftU.c_str(),
			nomfftI.c_str(),
			consumer_producer[lequel].N,
			consumer_producer[lequel].f,
			consumer_producer[lequel].xrU,
			consumer_producer[lequel].xrI,
```

```
                        consumer_producer[lequel].XrU,
                        consumer_producer[lequel].XiU,
                        consumer_producer[lequel].XrI,
                        consumer_producer[lequel].XiI,
                        consumer_producer[lequel].normeU,
                        consumer_producer[lequel].normeI,
                        consumer_producer[lequel].phiU,
                        consumer_producer[lequel].phiI);

        }
        if ( er == -1 )
        {
                ShowMessage("Error : Cannot open files!");
        }
        else
        {
        delete ProgressFFT ;

        consumer_producer[lequel].FFTsaved = true ;

        // if it's a consumer, then it make the calcul of impedance

        if ( consumer_producer[lequel].type == 'c' )
        {
        ir = calculConso(       consumer_producer[lequel].f,
                                consumer_producer[lequel].Zr,
                                consumer_producer[lequel].Zi,
                                consumer_producer[lequel].normeZ,
                                consumer_producer[lequel].phiZ,
                                consumer_producer[lequel].N,
                                consumer_producer[lequel].XrI,
                                consumer_producer[lequel].XiI,
                                consumer_producer[lequel].XrU,
                                consumer_producer[lequel].XiU);
        }

         // Create a new Label

        if ( first == true )
        {
        LabelFFTEnd = new TLabel(this);
        LabelFFTEnd->Parent = FormFFT;
        }

        // Define the Properties of the Label

        LabelFFTEnd->Font->Size = 12 ;
        LabelFFTEnd->Left= 120;
        LabelFFTEnd->Top = 300 ;
        LabelFFTEnd->Caption="FFT saved";
        LabelFFTEnd->Font->Color = clGreen ;
        ButtonClearFFT->Enabled = true ;

        first = false ;
        }
}

//***************************************************************************//
//     Open Dialog box to search a file directory
//***************************************************************************//

void __fastcall TFormFFT::ButtonSearchDirectoryUClick(TObject *Sender)
{
if(SaveDialogFFT->Execute()){
        EditDirectoryU->Text = SaveDialogFFT->FileName;
}
}

//***************************************************************************//
//    Open Dialog box to search a file directory
//***************************************************************************//

void __fastcall TFormFFT::ButtonSearchDirectoryIClick(TObject *Sender)
{
if(SaveDialogFFT->Execute()){
        EditDirectoryI->Text = SaveDialogFFT->FileName;
```

```
}
}

//***************************************************************************//
//    Enter in this function when the user choose a window for the FFT,
//    It save the result into a variable
//***************************************************************************//

void __fastcall TFormFFT::ComboBoxFFTChange(TObject *Sender)
{
        fenetre = ComboBoxFFT->ItemIndex ;
}

//***************************************************************************//
//    It's a graphic function only
//***************************************************************************//

void __fastcall TFormFFT::FormShow(TObject *Sender)
{
        if ( consumer_producer[lequel].FFTsaved == true )
        {
                LabelFFTEnd->Caption="FFT saved";
                ComboBoxFFT->ItemIndex = fenetre ;
                ComboBoxFFT->Enabled = false ;

                CheckBoxCopyFFT->Enabled = false ;
                LabelUFFT->Enabled = false ;
                LabelIFFT->Enabled = false ;
                EditDirectoryU->Enabled = false ;
                EditDirectoryI->Enabled = false ;
                ButtonSearchDirectoryU->Enabled = false ;
                ButtonSearchDirectoryI->Enabled = false ;

        }
        else
        {
                CheckBoxCopyFFT->Checked = false ;
                CheckBoxCopyFFT->Enabled = true ;

                LabelUFFT->Enabled = false ;
                LabelIFFT->Enabled = false ;
                EditDirectoryU->Enabled = false ;
                EditDirectoryI->Enabled = false ;
                ButtonSearchDirectoryU->Enabled = false ;
                ButtonSearchDirectoryI->Enabled = false ;

                LabelFFTEnd->Caption="";
                ComboBoxFFT->ItemIndex = 0 ;
                ComboBoxFFT->Enabled = true ;
        }
}

//***************************************************************************//
//    Enter in this function when the user clic on the checkbox to create a
//    copy of the original file. It's a graphic function only
//***************************************************************************//


void __fastcall TFormFFT::CheckBoxCopyFFTClick(TObject *Sender)
{
        if (  CheckBoxCopyFFT->Checked == true )
        {
                LabelUFFT->Enabled = true ;
                LabelIFFT->Enabled = true ;
                EditDirectoryU->Enabled = true ;
                EditDirectoryI->Enabled = true ;
                ButtonSearchDirectoryU->Enabled = true ;
                ButtonSearchDirectoryI->Enabled = true ;

        }
        else
        {
                LabelUFFT->Enabled = false ;
                LabelIFFT->Enabled = false ;
                EditDirectoryU->Enabled = false ;
                EditDirectoryI->Enabled = false ;
```

```
                ButtonSearchDirectoryU->Enabled = false ;
                ButtonSearchDirectoryI->Enabled = false ;
        }
}

//***************************************************************************//
//    When the user clic on the "clear" button, then it frees all the memory
//    and it forbid somes actions.
//***************************************************************************//


void __fastcall TFormFFT::ButtonClearFFTClick(TObject *Sender)
{
                int ir ;

                ButtonClearFFT->Enabled = false ;
                CheckBoxCopyFFT->Checked = false ;
                CheckBoxCopyFFT->Enabled = true ;
                EditDirectoryU->Enabled = false ;
                EditDirectoryU->Text = "" ;
                EditDirectoryI->Enabled = false ;
                EditDirectoryI->Text = "" ;
                LabelUFFT->Enabled = false ;
                LabelIFFT->Enabled = false ;
                ButtonSearchDirectoryU->Enabled = false ;
                ButtonSearchDirectoryI->Enabled = false ;


                ComboBoxFFT->Enabled = true ;
                ComboBoxFFT->ItemIndex = 0 ;

                LabelFFTEnd->Caption = "";

                ir = LibererMemoireConsumer_producer(lequel,true);
                consumer_producer[lequel].FFTsaved = false ;

}

//***************************************************************************//
//
//***************************************************************************//
```

Annexe 5

Code « bilanDataP »

```
/*****************************************************************************
 * This file defines all functions and actions linked with Producer window
 *****************************************************************************/

#include <vcl.h>
#include <FileCtrl.hpp>
#include "stdio.h"
#pragma hdrstop

#include "bilanDataP.h"
#include "bilanFFT.h"
#include "bilanMain.h"
#include <fourier.h>
#include <ASQProcess.h>
#include <hbook.h>
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"

// Global var declaration

TDataP *DataP;
TLabel *LabelSavedP ;
TProgressBar *ProgressSavedP ;


AnsiString Directory;

int Power;
int first = true ;
char filecopy[200] ;
bool copyS ;

extern int lequel,dispOk,sweep_on,simule_on;
extern struct CONSUMER_PRODUCER consumer_producer[];
extern char filename[];

//***************************************************************************//
//     subroutine to make an action when the Producer window was called
//***************************************************************************//

__fastcall TDataP::TDataP(TComponent* Owner)
        : TForm(Owner)
{
}

//***************************************************************************//
//    Close the window when the user clic on the "quit" button
//***************************************************************************//

void __fastcall TDataP::ButtonQuitPClick(TObject *Sender)
{
        Close();
}

//***************************************************************************//
//    Forbidden the user to enter a another key that numerical key
//***************************************************************************//

void __fastcall TDataP::EditPowerKeyPress(TObject *Sender, char &Key)
{
        if ((Key < 48 || Key > 57)
        && Key != VK_BACK
        && Key != VK_DELETE
        && Key != DecimalSeparator)
        Key = 0;
}

//***************************************************************************//
//    Forbidden the user to enter a another key that numerical key
//***************************************************************************//

void __fastcall TDataP::EditSamplesPKeyPress(TObject *Sender, char &Key)
{
        if ((Key < 48 || Key > 57)
        && Key != VK_BACK
```

```
        && Key != VK_DELETE
        && Key != DecimalSeparator)
        Key = 0;
}

//***************************************************************************//
//    Open Dialog box to search a file directory
//***************************************************************************//

void __fastcall TDataP::ButtonDirectoryPClick(TObject *Sender)
{
if(odTension->Execute()){
        EditDirectoryP->Text = odTension->FileName;
}
}

//***************************************************************************//
//    The main function of this subroutine is : loading file ! The others lines
//    are for graphics apparence.
//***************************************************************************//

void __fastcall TDataP::ButtonSaveClick(TObject *Sender)
{
      consumer_producer[lequel].N = 2097152 ;

      // Local var declaration

      AnsiString SaveCopy ;

      int error,i,ir ;
      int iAMax;
      int iBMax;
      float w[20];
      float freq[20];
      float A[20];
      float B[20];

      if ( simule_on == 1 ||sweep_on == 1 )
      {
        if ( EditSamplingP->Text == "" || EditSamplesP->Text == "")
        {
        ShowMessage("Error : it remains empty case !");
        error = -1 ;
        }
        else
        {

        // Receives the User's datas and make some tests

        consumer_producer[lequel].N = StrToInt(EditSamplesP->Text);
        consumer_producer[lequel].dt =  StrToInt(EditSamplingP->Text)*0.000001;
        error = 0 ;

        if ( CheckBoxCopyP->Checked == true )
        {
                SaveCopy = EditFolderP->Text ;
                if ( SaveCopy == "")
                {
                        ShowMessage("Error : empty case -> copy name !");
                        sprintf(filecopy,"");
                        error = -1;
                }
                else
                {
                        copyS = true ;
                        sprintf(filecopy,SaveCopy.c_str());
                        error = 0 ;
                }
        }
        else
        {
                SaveCopy = "";
                sprintf(filecopy,"");
                copyS = false ;
        }
```

```
        // Allocate memory

        ir =  LibererMemoireConsumer_producer(lequel,false);
        ir =  AllouerMemoireConsumer_producer(consumer_producer[lequel].N,lequel,false);



        if ( sweep_on == 1 )
        {
                // create a sweep signal

                ir = Sweep(     consumer_producer[lequel].N,
                                consumer_producer[lequel].dt,
                                (2*3.14159*50),  // wmin
                                (2*3.14159*500), // wmax
                                20.0,            // magnitude
                                consumer_producer[lequel].xrU); // vector contains the sweep signal
        }
        else
        {

          //


                // load the voltage settings

          i=0;  A[i]=  230.0;  freq[i] =  50.0;    w[i]=2 * 3.14159 * freq[i];
          i=1;  A[i]=  120.0;  freq[i] =  60.0;    w[i]=2 * 3.14159 * freq[i];
          i=2;  A[i]=  70.0;   freq[i] =  100.0;   w[i]=2 * 3.14159 * freq[i];
          i=3;  A[i]=  50.0;   freq[i] =  200.0;   w[i]=2 * 3.14159 * freq[i];
/*        i=4;  A[i]=  20.0;   freq[i] =  5000.0;  w[i]=2 * 3.14159 * freq[i];
          i=5;  A[i]=  10.0;   freq[i] =  50000.0; w[i]=2 * 3.14159 * freq[i];*/

          iAMax = 4;  // define the number of sin functions

                // load the current settings

          i=0;  B[i]=  20.0;  freq[i] =  50.0;     w[i]=2 * 3.14159 * freq[i];
          i=1;  B[i]=  2.0;   freq[i] =  60.0;     w[i]=2 * 3.14159 * freq[i];
          i=2;  B[i]=  3.0;   freq[i] =  100.0;    w[i]=2 * 3.14159 * freq[i];
          i=3;  B[i]=  5.0;   freq[i] =  200.0;    w[i]=2 * 3.14159 * freq[i];
/*        i=4;  B[i]=  10.0;  freq[i] =  5000.0;   w[i]=2 * 3.14159 * freq[i];
          i=5;  B[i]=  15.0;  freq[i] =  50000.0;  w[i]=2 * 3.14159 * freq[i];*/
          iBMax = 4;

        // create this 2 functions

        ir = FaireFonctionAAnalyser(    consumer_producer[lequel].N,
                                        consumer_producer[lequel].dt,
                                        0.0,
                                        iAMax,
                                        iBMax,
                                        A,
                                        B,
                                        w,
                                        consumer_producer[lequel].xrU,
                                        consumer_producer[lequel].xrI);

        }
        }
      }
      else        // No sweep and no simule
      {

      if (EditPower->Text == "" || EditDirectoryP->Text == ""  )
      {
          ShowMessage("Error : it remains empty case !");
          error = -1 ;
      }
      else
      {
        consumer_producer[lequel].PowerMax = StrToFloat(EditPower->Text);
        Directory = EditDirectoryP->Text ;
        sprintf(consumer_producer[lequel].FileAcqUI,Directory.c_str());
        error = 0 ;
      }
```

```
  if ( CheckBoxCopyP->Checked == true )
  {
    SaveCopy = EditFolderP->Text;
    if ( SaveCopy == "" )
    {
            ShowMessage("Error : empty case -> copy name !");
            error = -1;
    }
    else
    {
    copyS = true ;
    error = 0 ;
    }
  }
  else
  {
    copyS = false ;
    SaveCopy = "" ;
  }
  if ( error != -1 )
  {

  // Allocate memory

  ir =  LibererMemoireConsumer_producer(lequel,false);
  ir =  AllouerMemoireConsumer_producer(consumer_producer[lequel].N,lequel,false);

  // Create a Progress Bar

  ProgressSavedP = new TProgressBar(this);
  ProgressSavedP->Parent=DataP;

  // Define the Properties of the Progress Bar

  ProgressSavedP->Left = 98;
  ProgressSavedP->Top = 368;
  ProgressSavedP->Width = 105;
  ProgressSavedP->Height = 16;
  ProgressSavedP->Smooth=true ;
  ProgressSavedP->Max = 100;
  ProgressSavedP->Step = 1 ;
  ProgressSavedP->Position = 0;
  SendMessage(ProgressSavedP->Handle,PBM_SETBARCOLOR,0,clRed);

  error = LireSignalMesure( consumer_producer[lequel].FileAcqUI,  // Name of file to load
                            SaveCopy.c_str(),                     // Name of file to make a cop
                            consumer_producer[lequel].type,
                            copyS,                                 // = 0 -> no copy of file
                                                                   // = 1 -> make a copy of file
                            consumer_producer[lequel].xrU,        // vector contains voltage
                            consumer_producer[lequel].xrI,        // vector contains current
                            consumer_producer[lequel].t,          // vector contains time
                            consumer_producer[lequel].N);         // nbr samples
  if ( error == -1 )
  {
    delete ProgressSavedP;
    ShowMessage("Error : Saving File miss!");
  }
  else
  {
    delete ProgressSavedP;

    consumer_producer[lequel].dt = consumer_producer[lequel].t[1]-consumer_producer[lequel].t[0
    if ( first == true )
    {
    LabelSavedP = new TLabel(this);
    LabelSavedP->Parent=DataP;
    }
    LabelSavedP->Font->Size = 12 ;
    LabelSavedP->Left= 100;
    LabelSavedP->Top = 365 ;
    LabelSavedP->Font->Color = clGreen ;
    LabelSavedP->Caption="Datas saved";

    // Permission to run FFT
```

```
        ButtonFFTP->Enabled = true ;

        // Permission to clear and show graphics

        ButtonClearP->Enabled = true ;
        ButtonSaveP->Enabled = false ;
        ButtonDisplayP->Enabled = true ;

        consumer_producer[lequel].saved = true ;

        first = false ;
      }         // End of else
      }         // End of if
      }         // End of else

      if ( simule_on == 1 || sweep_on == 1 )
      {
        if ( first == true )
        {
        LabelSavedP = new TLabel(this);
        LabelSavedP->Parent=DataP;
        }
        LabelSavedP->Font->Size = 12 ;
        LabelSavedP->Left= 100;
        LabelSavedP->Top = 365 ;
        LabelSavedP->Font->Color = clGreen ;
        LabelSavedP->Caption="Datas saved";
        // Permission to run FFT

        ButtonFFTP->Enabled = true ;

        // Permission to clear and show graphics

        ButtonClearP->Enabled = true ;
        ButtonSaveP->Enabled = false ;
        ButtonDisplayP->Enabled = true ;
        consumer_producer[lequel].saved = true ;
        first = false ;
      }
}

//***************************************************************************//
//    When the user clic on the "clear" button, then it frees all the memory
//    and it forbid somes actions.
//***************************************************************************//

void __fastcall TDataP::ButtonClearPClick(TObject *Sender)
{

        int ir;

        EditDirectoryP->Text = "";
        EditPower->Text = "" ;
        EditSamplesP->Text = "" ;
        EditSamplingP->Text = "" ;
        Directory = "" ;
        Power = 0 ;
        EditPower->Enabled = true ;
        EditSamplesP->Enabled = false;
        EditDirectoryP->Enabled = true ;
        CheckBoxCopyP->Checked = false ;
        CheckBoxCopyP->Enabled = true ;
        ButtonSaveP->Enabled = true ;
        ButtonDisplayP->Enabled = false ;
        ButtonClearP->Enabled = false ;
        ButtonFFTP->Enabled = false ;
        ButtonDirectoryP->Enabled = true ;
        CheckBoxSweepP->Checked = false ;
        CheckBoxSimuleP->Checked = false ;
        CheckBoxSweepP->Enabled = true ;
        CheckBoxSimuleP->Enabled = true ;

        if ( consumer_producer[lequel].FFTsaved == true )
        {
                ir = LibererMemoireConsumer_producer(lequel,false);
```

```
                ir = LibererMemoireConsumer_producer(lequel,true);
        }
        else
        {
                ir = LibererMemoireConsumer_producer(lequel,false);
        }
        consumer_producer[lequel].saved = false ;
        consumer_producer[lequel].FFTsaved = false ;
        LabelSavedP->Caption = "" ;
        simule_on = 0 ;
        sweep_on = 0 ;
        ir = hdelet_(0);
}

//***************************************************************************//
//    It's a graphic function only
//***************************************************************************//

void __fastcall TDataP::CheckBoxCopyPClick(TObject *Sender)
{
        if (  CheckBoxCopyP->Checked == true )
        {
                LabelFolderP->Enabled = true ;
                EditFolderP->Enabled = true ;
        }
        else
        {
                LabelFolderP->Enabled = false ;
                EditFolderP->Enabled = false ;
        }
}

//***************************************************************************//
//    Show the FFT window when the user clic on the FFT button
//***************************************************************************//

void __fastcall TDataP::ButtonFFTPClick(TObject *Sender)
{
        FormFFT->ShowModal();
}

//***************************************************************************//
//    This function display the graphics with "hbook6" program
//***************************************************************************//

void __fastcall TDataP::ButtonDisplayPClick(TObject *Sender)
{
        // Insert the points into graphics

        RemplirGraphique(lequel,consumer_producer[lequel].N);

        // Save all plots

        SavePlot(lequel);

        // Launch the file

        ShellExecute(NULL,"open",filename,"","",SW_NORMAL);

}

//***************************************************************************//
//    It's a graphic function only. It's to save the graphics settings when
//    you quit a window after haven saved it.
//***************************************************************************//


void __fastcall TDataP::FormShow(TObject *Sender)
{
        if (consumer_producer[lequel].saved==false)
        {
        LabelSamplesP->Enabled = false ;
        LabelSamplingP->Enabled = false ;
        LabelFolderP->Enabled = false ;
        LabelSavedP->Caption = "" ;
        EditFolderP->Enabled = false ;
```

```
        EditPower->Enabled = true ;
        EditSamplesP->Enabled = false ;
        EditSamplesP->Text = "2097152" ;
        EditSamplingP->Enabled = false ;
        EditSamplingP->Text= "5" ;
        EditDirectoryP->Enabled = true ;
        EditDirectoryP->Text = "D:" ;
        EditPower->Text = "0" ;
        EditFolderP->Enabled = false ;
        ButtonClearP->Enabled = false ;
        ButtonDisplayP->Enabled = false ;
        ButtonDirectoryP->Enabled = true ;
        ButtonSaveP->Enabled = true ;
        ButtonFFTP->Enabled = false ;
        CheckBoxCopyP->Enabled = true ;
        CheckBoxSweepP->Checked = false ;
        CheckBoxSimuleP->Checked = false ;
        CheckBoxSweepP->Enabled = true ;
        CheckBoxSimuleP->Enabled = true ;
        }
        else
        {
        LabelSavedP->Caption="Datas saved";
        EditDirectoryP->Text = consumer_producer[lequel].FileAcqUI ;
        EditDirectoryP->Enabled = false ;
        EditSamplesP->Text = consumer_producer[lequel].N ;
        EditSamplesP->Enabled = false ;
        EditPower->Text = consumer_producer[lequel].PowerMax ;
        EditPower->Enabled = false ;
        EditFolderP->Enabled = false ;
        ButtonClearP->Enabled = true ;
        ButtonDisplayP->Enabled = true ;
        ButtonFFTP->Enabled = true ;
        ButtonSaveP->Enabled = false ;
        ButtonDirectoryP->Enabled = false ;
        CheckBoxCopyP->Enabled = false ;
        CheckBoxSweepP->Enabled = false ;
        CheckBoxSimuleP->Enabled = false ;
        }
}

//***************************************************************************//
//    When you clic on this check box you enter into this function and a
//    variable is set. It's a graphic function too
//***************************************************************************//

void __fastcall TDataP::CheckBoxSweepPClick(TObject *Sender)
{
        if (  CheckBoxSweepP->Checked == true )
        {
                LabelSamplesP->Enabled = true ;
                LabelSamplingP->Enabled = true ;
                LabelTensionP->Enabled = false ;
                EditSamplesP->Enabled = true ;
                EditSamplesP->Text = "2097152" ;
                EditSamplingP->Enabled = true ;
                EditSamplingP->Text = "5" ;
                EditDirectoryP->Enabled = false ;
                ButtonDirectoryP->Enabled = false ;
                CheckBoxSimuleP->Enabled = false ;
                sweep_on=1 ;
        }
        else
        {
                LabelSamplesP->Enabled = false ;
                LabelSamplingP->Enabled = false ;
                LabelTensionP->Enabled = true ;
                EditSamplesP->Enabled = false ;
                EditSamplingP->Enabled = false ;
                EditDirectoryP->Enabled = true ;
                ButtonDirectoryP->Enabled = true ;
                CheckBoxSimuleP->Enabled = true ;
                sweep_on=0 ;
        }
}
```

```
//*************************************************************************//
//    When you clic on this check box you enter into this function and a
//    variable is set. It's a graphic function too
//*************************************************************************//

void __fastcall TDataP::CheckBoxSimulePClick(TObject *Sender)
{
        if (  CheckBoxSimuleP->Checked == true )
        {
                LabelSamplesP->Enabled = true ;
                LabelSamplingP->Enabled = true ;
                LabelTensionP->Enabled = false ;
                EditSamplesP->Enabled = true ;
                EditSamplesP->Text = "2097152" ;
                EditSamplingP->Enabled = true ;
                EditSamplingP->Text = "5" ;
                EditDirectoryP->Enabled = false ;
                ButtonDirectoryP->Enabled = false ;
                CheckBoxSweepP->Enabled = false ;
                simule_on=1;
        }
        else
        {
                LabelSamplesP->Enabled = false ;
                LabelSamplingP->Enabled = false ;
                LabelTensionP->Enabled = true ;
                EditSamplesP->Enabled = false ;
                EditSamplingP->Enabled = false ;
                EditDirectoryP->Enabled = true ;
                ButtonDirectoryP->Enabled = true ;
                CheckBoxSweepP->Enabled = true ;
                simule_on=0;
        }
}

//*************************************************************************//
//
//*************************************************************************//
```

Annexe 6

Code « bilanDataC »

```
/*****************************************************************************
 * This file defines all functions and actions linked with Consumer window
 *****************************************************************************/

#include <vcl.h>
#include <FileCtrl.hpp>
#include "stdio.h"
#pragma hdrstop
#include "bilanDataC.h"
#include "bilanDataP.h"
#include "bilanMain.h"
#include "bilanFFT.h"
#include <fourier.h>
#include <hbook.h>
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"

TDataC *DataC;
TLabel *LabelSavedC;
TProgressBar *ProgressSavedC;

AnsiString Directory;

extern struct CONSUMER_PRODUCER consumer_producer[] ;
extern int lequel,nbrTot,sweep_on,simule_on ;
extern char filename[];
extern bool copyS ;

//***************************************************************************//
//     subroutine to make an action when the Consumer window was called
//***************************************************************************//

__fastcall TDataC::TDataC(TComponent* Owner)
        : TForm(Owner)
{

}

//***************************************************************************//
//    Close the window when the user clic on the "quit" button
//***************************************************************************//

void __fastcall TDataC::ButtonQuitCClick(TObject *Sender)
{
        Close();
}
//***************************************************************************//
//    Forbidden the user to enter a another key that numerical key
//***************************************************************************//

void __fastcall TDataC::EditSamplesCKeyPress(TObject *Sender, char &Key)
{
        if ((Key < 48 || Key > 57)
        && Key != VK_BACK
        && Key != VK_DELETE
        && Key != DecimalSeparator)
        Key = 0 ;
}

//***************************************************************************//
//    Forbidden the user to enter a another key that numerical key
//***************************************************************************//

void __fastcall TDataC::EditPowerKeyPress(TObject *Sender, char &Key)
{
        if ((Key < 48 || Key > 57)
        && Key != VK_BACK
        && Key != VK_DELETE
        && Key != DecimalSeparator)
        Key = 0 ;
}

//***************************************************************************//
//    Open Dialog box to search a file directory
//***************************************************************************//
```

```
void __fastcall TDataC::ButtonDirectoryCClick(TObject *Sender)
{
if(odImped->Execute()){
        EditDirectoryC->Text = odImped->FileName;
}
}

//***************************************************************************//
//    The main function of this subroutine is : loading file ! The others lines
//    are for graphics apparence.
//***************************************************************************//

void __fastcall TDataC::ButtonSaveCClick(TObject *Sender)
{

      consumer_producer[lequel].N = 2097152 ;

      // Local var declaration

      AnsiString SaveCopy ;

      int error,i,ir ;
      float dt ;


      // Receives the User's datas

      if ( EditDirectoryC->Text == "" )
      {
        ShowMessage("Error : it remains empty case !");
        error = -1 ;
      }
      else
      {
        Directory = EditDirectoryC->Text ;
        sprintf(consumer_producer[lequel].FileAcqUI,Directory.c_str());
        error = 0 ;
      }
      if ( CheckBoxCopyC->Checked == true )
      {
        SaveCopy = EditFolderC->Text;
        if ( SaveCopy == "" )
        {
                ShowMessage("Error : empty case -> copy name !");
                error = -1;
        }
        else
        {
                copyS = true ;
                error = 0 ;
        }
      }
      else
      {
        copyS = false ;
        SaveCopy = "" ;
      }

      if ( error != -1 )
      {

      // Allocate memory

      ir =  LibererMemoireConsumer_producer(lequel,false);
      ir =  AllouerMemoireConsumer_producer(consumer_producer[lequel].N,lequel,false);

      // Create a Progress Bar

      ProgressSavedC = new TProgressBar(this);
      ProgressSavedC->Parent=DataC;

      // Define the Properties of the Progress Bar

      ProgressSavedC->Left = 96;
      ProgressSavedC->Top = 225;
```

```
      ProgressSavedC->Width = 113;
      ProgressSavedC->Height = 16;
      ProgressSavedC->Smooth=true ;
      ProgressSavedC->Max = 100;
      ProgressSavedC->Step = 1 ;
      ProgressSavedC->Position = 0;
      SendMessage(ProgressSavedC->Handle,PBM_SETBARCOLOR,0,clBlue);

      error = LireSignalMesure( Directory.c_str(),
                                SaveCopy.c_str(),
                                consumer_producer[lequel].type,
                                copyS,
                                consumer_producer[lequel].xrU,
                                consumer_producer[lequel].xrI,
                                consumer_producer[lequel].t,
                                consumer_producer[lequel].N );
      if ( error == -1 )
      {
        ShowMessage("Error : Saving File miss!");
        delete ProgressSavedC;
      }
      else
      {
        delete ProgressSavedC;

consumer_producer[lequel].dt = consumer_producer[lequel].t[1]-consumer_producer[lequel].t[0];

        LabelSavedC = new TLabel(this);
        LabelSavedC->Parent=DataC;
        LabelSavedC->Font->Size = 12 ;
        LabelSavedC->Left= 120;
        LabelSavedC->Top = 225;
        LabelSavedC->Font->Color = clGreen ;
        LabelSavedC->Caption="Datas saved";

        // Permission to run FFT

        ButtonFFTC->Enabled = true ;

        ButtonClearC->Enabled = true ;
        ButtonSaveC->Enabled = false ;
        ButtonDisplayC->Enabled = true ;

        consumer_producer[lequel].saved = true ;
      }
      }

}

//***************************************************************************//
//    When the user clic on the "clear" button, then it frees all the memory
//    and it forbid somes actions.
//***************************************************************************//


void __fastcall TDataC::ButtonClearCClick(TObject *Sender)
{
        int ir ;

        EditDirectoryC->Text = "";
        Directory = "" ;
        EditDirectoryC->Enabled = true ;
        CheckBoxCopyC->Checked = false ;
        CheckBoxCopyC->Enabled = true;
        ButtonDirectoryC->Enabled = true ;
        ButtonSaveC->Enabled = true ;
        ButtonDisplayC->Enabled = false ;
        ButtonClearC->Enabled = false ;
        ButtonFFTC->Enabled = false ;


        if ( consumer_producer[lequel].FFTsaved == true )
        {
                ir = LibererMemoireConsumer_producer(lequel,false);
                ir = LibererMemoireConsumer_producer(lequel,true);
        }
```

```
        else
        {
                ir = LibererMemoireConsumer_producer(lequel,false);
        }

        consumer_producer[lequel].saved = false ;
        consumer_producer[lequel].FFTsaved = false ;
        LabelSavedC->Caption = "" ;

        // Delete all graphics

        ir = hdelet_(0);
}

//---------------------------------------------------------------------------//
//    It's a graphic function only.
//---------------------------------------------------------------------------//

void __fastcall TDataC::CheckBoxCopyCClick(TObject *Sender)
{

        if (  CheckBoxCopyC->Checked  == true )
        {
                LabelFolderC->Enabled = true ;
                EditFolderC->Enabled = true ;
        }
        else
        {
                LabelFolderC->Enabled = false ;
                EditFolderC->Enabled = false ;
        }
}

//---------------------------------------------------------------------------//
//    It's a graphic function only. It's to save the graphics settings when
//    you quit a window after haven saved it.
//---------------------------------------------------------------------------//

void __fastcall TDataC::FormShow(TObject *Sender)
{

        if (consumer_producer[lequel].saved==false)
        {
        LabelFolderC->Enabled = false ;
        LabelSavedC->Caption = "" ;
        EditDirectoryC->Text = "D:" ;
        EditFolderC->Enabled = false ;
        EditDirectoryC->Enabled = true ;
        ButtonDirectoryC->Enabled = true ;
        ButtonFFTC->Enabled = false ;
        ButtonClearC->Enabled = false ;
        ButtonDisplayC->Enabled = false ;
        ButtonSaveC->Enabled = true ;
        CheckBoxCopyC->Enabled = true ;

        }
        else
        {
        LabelSavedC->Caption="Datas saved";
        EditDirectoryC->Text = consumer_producer[lequel].FileAcqUI ;
        EditDirectoryC->Enabled = false ;
        ButtonFFTC->Enabled = true ;
        ButtonClearC->Enabled = true ;
        ButtonDisplayC->Enabled = true ;
        ButtonSaveC->Enabled = false ;
        ButtonDirectoryC->Enabled = false ;
        ButtonDisplayC->Enabled = true ;
        CheckBoxCopyC->Enabled = false ;

        }
}

//---------------------------------------------------------------------------//
//     Display the FFT window when the user clic on the FFT button
//---------------------------------------------------------------------------//
```

```
void __fastcall TDataC::ButtonFFTCClick(TObject *Sender)
{
        FormFFT->ShowModal();
}

//***************************************************************************//
//     This function display the graphics with "hbook6" program
//***************************************************************************//

void __fastcall TDataC::ButtonDisplayCClick(TObject *Sender)
{
        // Insert the points into graphics

        RemplirGraphique(lequel,consumer_producer[lequel].N);

        // Save all plots

        SavePlot(lequel);

        // Launch the file

        ShellExecute(NULL,"open",filename,"","",SW_NORMAL);
}

//***************************************************************************//
//
//***************************************************************************//
```

Annexe 7

Code « bilanTools »

```
/*****************************************************************************
 * This file defines all functions and actions linked with Tools window
 *****************************************************************************/

#include <vcl.h>
#include <stdio.h>
#pragma hdrstop

#include "bilanTools.h"
#include "bilanMain.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TTools *Tools;

//****************************************************************************//
//     subroutine to make an action when the tools window was called
//****************************************************************************/

__fastcall TTools::TTools(TComponent* Owner)
        : TForm(Owner)
{

}

//****************************************************************************//
//    Close the window when the user clic on the "quit" button
//****************************************************************************//

void __fastcall TTools::ButtonQuitClick(TObject *Sender)
{
        Close();
}

//****************************************************************************//
//    Forbidden the user to enter a another key that numerical key
//****************************************************************************//

void __fastcall TTools::EditProdKeyPress(TObject *Sender, char &Key)
{
        if ((Key < 48 || Key > 57)
        && Key != VK_BACK
        && Key != VK_DELETE
        && Key != DecimalSeparator)
        Key = 0;
}

//****************************************************************************//
//    Forbidden the user to enter a another key that numerical key
//****************************************************************************//

void __fastcall TTools::EditConsKeyPress(TObject *Sender, char &Key)
{
        if ((Key < 48 || Key > 57)
        && Key != VK_BACK
        && Key != VK_DELETE
        && Key != DecimalSeparator)
        Key = 0;
}

//****************************************************************************//
//
//****************************************************************************//
```

Annexe 8

Code « fourier.c »

```
/*****************************************************************************
 * This file defines all functions that they have made with Visual Studio C++ *
 *****************************************************************************/

#include "stdafx.h"
#include "string.h"
#include "stdlib.h"
#include "stdio.h"
#include "memory.h"
#include "conio.h"
#include "math.h"
#include "time.h"
#include "fenetres.h"
#include "openfile.h"
#include "display.h"
#include "acquisition.h"
#include "arrondi.h"
#include "impedConso.h"
#include <ctype.h>
#include <fft.h>
#include "bilanMain.h"
#include "bilanDataC.h"
#include "bilanDataC.h"


extern TProgressBar *ProgressSavedP ;
extern TProgressBar *ProgressSavedC ;
extern TProgressBar *ProgressFFT;
extern bool copyS ;
extern char filecopy[] ;

int my_random(int min, int max)
{
    float rando=0.0;
        rando=rand()%(max - min + 1) + min;
        return rando;
}

//*****************************************************************************//
// Remplit le vecteur xr avec une amplitude qui balaye sur N point avec
// des fréquences de wMin à wMax pour des pas de temps de dt.
//*****************************************************************************//

int Sweep(long  N, float dt, float wMin,float wMax, float B, float xr[])
{
  long i;
  float t, y, w_t;
  FILE *filefoncSweep=NULL;
  if ( copyS == true )
  {
        filefoncSweep = fopen(filecopy,"wt");
  }
  for(i=0;i<N;i++){
      t = (float)i*dt;
      w_t = wMin + (float)i * (wMax -wMin) / (float) N;
      y = B * sin(w_t * t);
      xr[i] = y;
      if ( copyS == true )
      {
        fprintf(filefoncSweep,"%lf\t%lf\n",t,xr[i]);
      }
  }
  if ( copyS == true )
  {
        fclose(filefoncSweep);
  }
  return(0);
}


//*****************************************************************************//
// Fills the xrU and xrI vector with a sum of sinus preloaded
//*****************************************************************************//

int FaireFonctionAAnalyser(     long N,
                                float dt,
```

```
                                float A_random,
                                int iAMax,
                                int iBMax,
                                float A[],
                                float B[],
                                float w[],
                                float *xrU,
                                float *xrI)
{

 FILE *filesimule=NULL;

  int i,j;
  float t;
  float xsumA,xsumB;

  if ( copyS == true )
  {
        filesimule = fopen(filecopy,"wt");
  }

  for(i=0;i<N;i++){
      xsumA = 0.0;
      xsumB = 0.0;
      t = (float)i*dt;
      for(j=0;j<iAMax;j++) {
         xsumA += A[j] * sin (w[j] * t);
      }
      for(j=0;j<iBMax;j++) {
         xsumB += B[j] * sin (w[j] * t);
      }
      xrU[i] = xsumA;
      xrI[i] = xsumB;
      if ( copyS == true )
      {
        fprintf(filesimule,"%lf\t%lf\t%lf\n",t,xrU[i],xrI[i]);
      }
  }
  if ( copyS == true )
  {
        fclose(filesimule);
  }
  return(0);
}

//*************************************************************************//
// This function multiplie the mesured signal with a window
//*************************************************************************//

int MultFenetre(long N,float fnt[],float xr[])
{
        int i;
        int inc = 0 ;
        for(i=0;i<N;i++)
        {
                if ( inc == (N/10) )
                {
                        ProgressFFT->StepIt();
                        inc = 0 ;
                }
                xr[i]=xr[i]*fnt[i];
                inc ++ ;
        }
        return(0);
}

//*************************************************************************//
// This function read an ASCII file ( .txt ). It can make a copy of the
// original file too.
//*************************************************************************//

int LireSignalMesure(   char *filename,   //
                        char *filew,
                        char type,
                        bool copy,
                        float *xrU,
```

```
                        float *xrI,
                        float *t,
                        long N)
{
   int i=0;
   int inc = 0 ;
   float temps;
   float xU,xI;
   FILE *fileWrite = NULL;
   FILE *streamFile = NULL;
   char ligne[200];
   if(strlen(filew)) fileWrite = fopen(filew,"wt");
   streamFile = fopen (filename, "rt" );

   if(streamFile == NULL) return(-1);      // si le fichier à lire n'existe pas, on quitte
   for(;;)
   {
               if ( type == 'c')
               {
               if ( inc == (N/100) )
               {
                       ProgressSavedC->StepIt();
                       inc = 0 ;
               }
               }
               else
               {
               if ( inc == (N/100) )
               {
                       ProgressSavedP->StepIt();
                       inc = 0 ;
               }
               }

               fgets(ligne,200,streamFile);
               if(feof(streamFile)) break;
               if(i>=N) break;         // dès que le programme a lu N lignes il stop la lecture

               sscanf(ligne,"%f\t%f\t%f",&temps,&xU,&xI);      // scan le fichier ligne après ligne

               xrU[i] = xU;
               xrI[i] = xI;
               t[i]=temps;
               if ( copy == true )
               {
                       fprintf(fileWrite,"%lf\t%lf\t%lf\n",t[i],xrU[i],xrI[i]); // fait une copie
               }
               i++;
               inc++;
   }
   fclose (streamFile);
   if(strlen(filew)) fclose (fileWrite);
   return(0);
}

//*************************************************************************//
// Formules mathématiques des différentes fenêtres
//*************************************************************************//

double Hann(double x)
{
  return(0.5 *(1 - cos(TWO_PI *x )));
}

double Blackmann(double x)
{
  return(0.42  -0.5 * cos(TWO_PI *x ) +0.08 * cos(2.0*TWO_PI * x) );
}

double Hamming(double x)
{
  return(0.55  - 0.46 * cos(TWO_PI *x ) );
}

//*************************************************************************//
// Génère une fenêtre de type Hann,Blackmann ou Hamming
```

```
//***************************************************************************//

void GenereFenetre(int fen,long N,float *fnt)
  {
          int i;

// Génère une fenêtre de type Hann

          if ( fen == 1 )
          {
                  for(i=0;i<N;i++) {
                        fnt[i] =  Hann((float)i/(float)N);
                  }
          }

// Génère une fenêtre de type Blackmann

          else if (fen == 2)
          {
                  for(i=0;i<N;i++) {
                        fnt[i] =  Blackmann((float)i/(float)N);
                  }
          }


// Génère une fenêtre de type Hamming

          else if (fen==3)
          {
                  for(i=0;i<N;i++) {
                        fnt[i] =  Hamming((float)i/(float)N);
                  }
          }
  }

//***************************************************************************//
// Fonction réalisant un arrondi à 9 chiffres après la virgule
//***************************************************************************//

float fArrondi(float fValeur)
{

  float fDecimal,fResultat;

  if (fValeur<0)

     {
          fValeur *= 1.0e9;
                  fDecimal=fValeur-ceil(fValeur);
          if (fDecimal > 0.5)
             fResultat=floor(fValeur)/1.0e9;
          else
            fResultat=ceil(fValeur)/1.0e9;

     }
  else
     {
          fValeur *= 1.0e9;
                  fDecimal=fValeur-floor(fValeur);
          if (fDecimal< 0.5)
             fResultat=floor(fValeur)/1.0e9;
          else
             fResultat=ceil(fValeur)/1.0e9;
     }

  return fResultat;
}

//***************************************************************************//
// Fonction effectuant le calcul d'impédance de consommateur en fonction de
// U(w) et I(w).
//***************************************************************************//

int calculConso(float *f,
                float *Zr ,
                float *Zi,
```

```
                float *normeZ,
                float *phiZ,
                float N ,
                float *XrI,
                float *XiI,
                float *XrU,
                float *XiU)
{
        FILE*fileZ=NULL;

        int i ;
        float I_limit = 0.1 ;
        float denominateur;


        // ouverture fichier impédance

        if(fileZ==NULL ){
                fileZ = fopen("D:\\TD\\matlab\\imped.txt","wt");
        }

        if(fileZ == NULL) {
        ShowMessage("No more free space on the disc for the file");
                return(-1);
        }

        // Calcul impedance

        for(i=0;i<N/2;i++)
        {
                if ( sqrt(XrI[i]*XrI[i]+XiI[i]*XiI[i]) <= I_limit )
                {
      // si le courant est trop bas on assigne à Z une valeur non significative

                        normeZ[i] = 0.0 ;
                }
                else
                {
                        denominateur=(1/(XrI[i]*XrI[i]+XiI[i]*XiI[i]));
                        Zr[i]=(XrU[i]*XrI[i]+XiU[i]*XiI[i]);
                        Zi[i]=(XiU[i]*XrI[i]-XrU[i]*XiI[i]);
                        normeZ[i] = sqrt(((Zr[i]*denominateur)*(Zr[i]*denominateur))
                        +((Zi[i]*denominateur)*(Zi[i]*denominateur)));
             //         phiZ[i] = atan(Zi[i]/Zr[i]);
                }

                fprintf(fileZ,"%lf\t%lf\n",f[i],normeZ[i]);
        }
        fclose(fileZ);
        return (0);
}

//*************************************************************************//
// Fonction effectuant le calcul d'impédance de ligne en fonction des ses
// caractéristiques.
//*************************************************************************//

void calculLigne(       long N,
                        float dt,
                        float *Zi,
                        float Zr,
                        float *Norme,
                        float prodPos ,   // distance between reference point and this producer
                        float consPos,    // distance between reference point and this consumer
                        float Retoile,
                        float Letoile,
                        float Cetoile)
{
        FILE *fileLigne=NULL;

        int i,lenght ;
        float f,df ;
        float R,L,C;
        float pi ;
        lenght  =abs(prodPos-consPos);
```

```
	R = lenght* 1e-3 * Retoile ;
	L = lenght* 1e-6 * Letoile ;
	C = lenght* 1e-9 * Cetoile ;
	fileLigne = fopen("D:\\TD\\matlab\\ligne.txt","wt");

	df =  1.0 / ((float)N * dt);
	Zr = R ;
	for(i=0;i<N/2;i++)
	{
		f   = (((float)i)+1) * df;
		Zi[i] = (2*PI*f*L)+(1/(2*PI*f*C)) ;
		Norme[i]= sqrt( Zr*Zr+ Zi[i]*Zi[i]);
		fprintf(fileLigne,"%lf\t%lf\n",f,Norme[i]);

	}
	fclose(fileLigne);
}

//*************************************************************************//
//  Routine de calcul du courant fourni par P et consommé par C
//*************************************************************************//

void calculCourant(	long N,
			float *ir,
			float *ii,
			float *norme,
			float zr,
			float *zi,
			float *XrU,
			float *XiU,
			float *Zr,
			float *Zi)
{
	int i ;
	float deno ;

	for(i=0;i<N/2;i++)
	{
		deno = 1/(((zr+Zr[i])*(zr+Zr[i]))+((zi[i]+Zi[i])*(zi[i]+Zi[i])));
		ir[i]= deno*(XrU[i]*(zr+Zr[i])+XiU[i]*(zi[i]+Zi[i]));
		ii[i]=deno* (XiU[i]*(zr+Zr[i])-XrU[i]*(zi[i]+Zi[i]));
		norme[i]= sqrt(ir[i]*ir[i]+ii[i]*ii[i]);
	}
}

//*************************************************************************//
// Routine principale du mode FFT
//*************************************************************************//

int FFT(int sweep_on,
	int simule_on,
	int fenetre,
	bool copy,
	float dt,
	char nomfftU[],
	char nomfftI[],
	long N,
	float *f,
	float *xrU,
	float *xrI,
	float *XrU,
	float *XiU,
	float *XrI,
	float *XiI,
	float *yU,
	float *yI,
	float *phiU,
	float *phiI)

{

	srand(time( NULL ));
	float df;
	float xsum,t;
	long i,j;
	int inc =0 ;
```

```
        float A_random;
        float *xi1=NULL;
        float *xi2=NULL;
        float *fnt=NULL;


        FILE *filefftU=NULL;
        FILE *filefftI=NULL;

if ( copy == true )
{

        // Voltage File

    if(filefftU==NULL ){
        filefftU = fopen(nomfftU,"wt");
        }
        if(filefftU == NULL) {
                return(-1);
        }

        // Current File

    if(filefftI==NULL ){
        filefftI = fopen(nomfftI,"wt");
        }
        if(filefftI == NULL) {
        return(-1);
        }

}

        // Allocate time- and frequency-domain memory.

        xi1  = (float*)   malloc( N * sizeof(float));
        xi2  = (float*)   malloc( N * sizeof(float));
        fnt  = (float*)   malloc( N * sizeof(float));
        memset(xi1, 0, N * sizeof(float));
        memset(xi2, 0, N * sizeof(float));
        memset(fnt, 0, N * sizeof(float));


  // Sélection de la fenêtre

  GenereFenetre(fenetre,N,fnt);

  // Application de la fenêtre sur le signal mesuré

   MultFenetre(N,fnt,xrU);

  if ( sweep_on == 0)
  {
          MultFenetre(N,fnt,xrI);

         // Calculate FFT (courant)

          my_fft(N, xrI,xi2,XrI,XiI);
          df =  1.0 / ((float)N * dt);

          for(i=0;i<N/2;i++)
          {
                if ( inc == (N/80) )
                {
                        ProgressFFT->StepIt();
                        inc = 0 ;
                }

                f[i]   = ((float)i) * df;
                yI[i]   = sqrt(XrI[i]*XrI[i] + XiI[i]*XiI[i]);
                yI[i]  /= (float) N;
                yI[i]  *= 4.0;

                if ( XrI[i] != 0 )
                {
                        phiI[i] = atan(XiI[i]/XrI[i])*360/TWO_PI;
                }
```

```
                if ( copy == true )
                {
                fprintf(filefftI,"%i\t%lf\t%lf\t%lf\t%lf\n",i,f[i],XrI[i],XiI[i],yI[i]);
                }

                inc ++ ;
          }
  }

 // Calculate FFT (tension)

  my_fft(N, xrU,xi1, XrU,XiU);

  df =  1.0 / ((float)N * dt);
  for(i=0;i<N/2;i++)
  {
        if ( inc == (N/80) )
        {
                ProgressFFT->StepIt();
                inc = 0 ;
        }
    f[i]   = ((float)i) * df;
    yU[i]   = sqrt(XrU[i]*XrU[i] + XiU[i]*XiU[i]);
    yU[i]  /= (float) N;
    yU[i]  *= 4.0;

    if ( XrU[i] != 0 )
    {
    phiU[i] = atan(XiU[i]/XrU[i])*360/TWO_PI;
    }
    if ( copy == true )
    {
    fprintf(filefftU,"%i\t%lf\t%lf\t%lf\t%lf\n",i,f[i],XrU[i],XiU[i],yU[i]);
    }
    inc++ ;
  }

  free(xi1);
  free(xi2);
  free(fnt);

  if ( copy == true )
  {
  fclose(filefftU);
  fclose(filefftI);
  }
  return(0);

}


//*************************************************************************//
// Tableau cos
//*************************************************************************//

/*
double moncos(double in)
{
        static int tcos_ok = 0;
        int      itcos;
        double static tcos[4000];

        int i;

        if (tcos_ok == 0) {
                for (i=0;i<4000;i++) {
                        tcos[i] = cos(((double)i)*2.0*PI/4000.0);
                }
                tcos_ok = 1;
        }

        itcos=((int)(in*600/(2*PI)) % 4000);

        if(itcos>4000)  itcos=4000;
        if(itcos<0)     itcos=0;
```

```
        return(tcos[itcos]);

}                   */

//*************************************************************************//
// Tableau sin
//*************************************************************************//

/*
double monsin(double in)
{
        static int tsin_ok = 0;
        int      itsin;
        double static tsin[4000];

        int i;

        if (tsin_ok == 0) {
                for (i=0;i<4000;i++) {
                        tsin[i] = sin(((double)i)*2.0*PI/4000.0);
                }
                tsin_ok = 1;
        }

        itsin=((int)(in*4000/(2*PI)) % 4000);

        if(itsin>4000)  itsin=4000;
        if(itsin<0)     itsin=0;

        return(tsin[itsin]);

}      */

//*************************************************************************//
// Routine principale du mode DFT
//*************************************************************************//
/*
int DFT(int frequenceStart,int frequenceEnd, int frequencePas, char rep_nom_dft)
{
        double  x=0;
        double  T=10;                                   // Temps d'acquisition
        double  f0=1780;                                // Fréquence du signal
        double  f1=2000;
        double  A=10;                                   // Amplitude du signal
        double  B=A/2;
        double w0=2*PI*f0;
        double w1=2*PI*f1;
        double dt=1e-5;
        double hannSum=0;
        double sum_sc=0;
        double sum_ss=0;
        double wt,hannSumdt;
        int imax,i,f;
        int ifmax;
        double t,w,temps;
        char filenameOutput[300];

        double *h    =NULL;
        double *ampl =NULL;
        double *yt      =NULL;
        double sincos,sinsin;
        int   frequenceMax;

        FILE *fichier=NULL;

        //.....................................................................

                if (rep_nom_dft != " ") {
                        sprintf(filenameOutput,rep_nom_dft);
                }
                else fichier = stdout;


        imax = int(T/dt);
        h  = (double *) malloc(sizeof(double)*imax);
```

```
	yt = (double *) malloc(sizeof(double)*imax);
	ampl = (double *) malloc(sizeof(double)*ifmax);

	ifmax = int(1./(2.0*dt));
	ifmax = frequenceEnd;

	// Création de la fenêtre de Hann

	hannSum = 0;
	for (i=0;i<imax;i++) {
		temps= (double) i * dt;
		wt=2.0 * PI * temps / T;
		h[i]= 0.5-0.5 * moncos(wt);
		hannSum = hannSum + h[i];
	}

	hannSumdt = hannSum * dt;

	//	printf("Hann Termine\n");

	for (i=0;i<imax;i++) {
		t =(double)i * dt;
		yt[i]   =       A*sin(w0*t)+B*sin((w1)*t);
	}

	// ouverture fichier

	if(fichier==NULL ){
		fichier = fopen(filenameOutput,"wt");
	}

	if(fichier == NULL) {
		printf("plus de place sur disque pour le fichier %s",filenameOutput);
		return(1);
	}

	fprintf(fichier,"fréquence\tamplitude\n");

	for (f=frequenceStart;f<=frequenceEnd;f=f+frequencePas) {
		w=2*PI*f;
		sum_sc = 0;
		sum_ss = 0;
		for (i=0;i<imax;i++) {
			temps =(double)i * dt;
			wt = w * temps;
			sincos =  h[i] * yt[i]*cos(wt)*dt;
			sinsin =  h[i] * yt[i]*sin(wt)*dt;
			sum_sc = sum_sc + sincos;
			sum_ss = sum_ss + sinsin;
		}
		sum_sc = 2.0 * sum_sc / hannSumdt;
		sum_ss = 2.0 * sum_ss / hannSumdt;
		ampl[f] = sqrt(sum_sc * sum_sc + sum_ss * sum_ss);
		fprintf(fichier,"%d\t%f\n",f,ampl[f]);
		printf("Frequence : %d\n",f);
	}

	free(h);
	free(yt);
	free(ampl);

	fclose(fichier);

	return 0;
}
*/
```

Annexe 9

Code « fft.c »

```
/*--------------------------------------------------------------------------
   fft.c - fast Fourier transform and its inverse (both recursively)
   Copyright (C) 2004, Jerome R. Breitenbach.  All rights reserved.

   The author gives permission to anyone to freely copy, distribute, and use
   this file, under the following conditions:
      - No changes are made.
      - No direct commercial advantage is obtained.
      - No liability is attributed to the author for any damages incurred.
  --------------------------------------------------------------------------*/

/****************************************************************************
 * This file defines a C function fft that, by calling another function     *
 * fft_rec (also defined), calculates an FFT recursively.  Usage:           *
 *   fft(N, x, X);                                                          *
 * Parameters:                                                              *
 *   N: number of points in FFT (must equal 2^n for some integer n >= 1)    *
 *   x: pointer to N time-domain samples given in rectangular form (Re x,   *
 *      Im x)                                                               *
 *   X: pointer to N frequency-domain samples calculated in rectangular form*
 *      (Re X, Im X)                                                        *
 * Similarly, a function ifft with the same parameters is defined that      *
 * calculates an inverse FFT (IFFT) recursively.  Usage:                    *
 *   ifft(N, x, X);                                                         *
 * Here, N and X are given, and x is calculated.                            *
 ****************************************************************************/

#include "fft.h"
#include "math.h"
#include "stdlib.h"


// FFT
void my_fft(int N, float (*xr) , float (*xi), float (*Xr) , float (*Xi) )
{
  // Declare a pointer to scratch space.
  float (*XXr) = malloc(N * sizeof(float));
  float (*XXi) = malloc(N * sizeof(float));

  // Calculate FFT by a recursion.
  my_fft_rec(N, 0, 1, xr, xi, Xr, Xi, XXr,XXi);

  // Free memory.
  free(XXr);
  free(XXi);
}

// FFT recursion
void my_fft_rec(int N, int offset, int delta,
             float (*xr), float (*xi),
             float (*Xr), float (*Xi),
             float (*XXr), float (*XXi))
{
  int N2 = N/2;            // half the number of points in FFT
  int k;                   // generic index
  float cs, sn;           // cosine and sine
  int k00, k01, k10, k11;  // indices for butterflies
  float tmp0, tmp1;       // temporary storage

  if(N != 2)  // Perform recursive step.
    {
      // Calculate two (N/2)-point DFT's.
      my_fft_rec(N2, offset, 2*delta, xr,xi, XXr,XXi, Xr,Xi);
      my_fft_rec(N2, offset+delta, 2*delta, xr,xi, XXr,XXi, Xr,Xi);

      // Combine the two (N/2)-point DFT's into one N-point DFT.
      for(k=0; k<N2; k++)
        {
          k00 = offset + k*delta;    k01 = k00 + N2*delta;
          k10 = offset + 2*k*delta;  k11 = k10 + delta;
          cs = cos(TWO_PI*k/(float)N); sn = sin(TWO_PI*k/(float)N);
          tmp0 = cs * XXr[k11] + sn * XXi[k11];
          tmp1 = cs * XXi[k11] - sn * XXr[k11];
          Xr[k01] = XXr[k10] - tmp0;
          Xi[k01] = XXi[k10] - tmp1;
          Xr[k00] = XXr[k10] + tmp0;
```

```
          Xi[k00] = XXi[k10] + tmp1;
        }
    }
  else  // Perform 2-point DFT.
    {
      k00 = offset; k01 = k00 + delta;
      Xr[k01] = xr[k00] - xr[k01];
      Xi[k01] = xi[k00] - xi[k01];
      Xr[k00] = xr[k00] + xr[k01];
      Xi[k00] = xi[k00] + xi[k01];
    }
}

// IFFT
void my_ifft(int N, float (*xr), float (*xi), float (*Xr), float (*Xi))
{
  int N2 = N/2;       // half the number of points in IFFT
  int i;              // generic index
  float tmp0, tmp1;  // temporary storage

  // Calculate IFFT via reciprocity property of DFT.
  my_fft(N, Xr, Xi, xr, xi);
  xr[0] = xr[0]/N;    xi[0]  = xi[0]/N;
  xr[N2] = xr[N2]/N;  xi[N2] = xi[N2]/N;
  for(i=1; i<N2; i++)
    {
      tmp0 = xr[i]/N;       tmp1 = xi[i]/N;
      xr[i]= xr[N-i]/N;  xi[i] = xi[N-i]/N;
      xr[N-i] = tmp0;       xi[N-i] = tmp1;
    }
}
```