Laboratoire 4

Analyse de Fourier

Ce labo vous introduit à une technique mathématico-physique d'une grande utilité dans l'analyse de signal et, plus généralement, dans l'interprétation physique de données expérimentales: l'analyse de Fourier. C'est de l'argent en banque pour PHY-1620 l'hiver prochain, et ce sera également une excellente occasion de consolider vos habiletés en intégration numérique, acquises (laborieusement...?) au laboratoire de la semaine passée.

4.1 Objectifs

1. Apprendre des notions de base en analyse de signal
2. Commencer à apprivoiser les séries de Fourier
3. Apprendre la la manipulation de fichiers en entrée/sortie en C
4. Approfondir votre compréhension du codage des fonctions en C
5. Approfondir votre compréhension du contrôle des boucles et énoncés conditionnels en C.

4.2 Rapport de Lab

Votre rapport de Lab est à remettre la semaine prochaine **au début** de votre prochain laboratoire. Il doit inclure des réponses (plus graphiques et listing des codes source) aux questions posées aux sections 4.7 et 4.9, Avant de commencez la rédaction de votre rapport, relisez bien le petit document distribué en annexe au plan de cours, traitant du contenu des rapports de labo (également disponible sur la page web du cours).

4.3 Les signaux multipériodiques

Plusieurs étoiles sont sujettes à des variations périodiques de leur structure, qui se traduisent souvent en variations observables de leur luminosité. La Figure 4.1 illustre un exemple particulièrement frappant, soit celui de l'étoile naine blanche G117-B15A. Tout comme le temps de transit des ondes séismiques nous renseigne sur la structure interne de la Terre, les oscillations stellaires contiennent de l'information relative à la structure interne des étoiles, inaccessible à l'observation directe. **L'astérosismologie** est une sous-discipline de l'astrophysique centrée sur l'extraction de cette information, et son point de départ est habituellement d'identifier les fréquences d'oscillations présentes dans les données observationnelles. Souvent, comme sur la Figure 4.1, l'oscillation stellaire est caractérisée par plusieurs fréquences présentes simultanément. Le premier défi est donc d'extraire ces fréquences du signal multipériodique. Ensuite, il est souvent utile d'estimer les amplitudes des différents modes d'oscillation présents,

puisque ces amplitudes contiennent de l'information relative aux mécanismes de production et d'amortissement des oscillations.

Ce genre d'analyse, dite **analyse spectrale** ou **analyse de Fourier**, ne se limite évidemment pas aux oscillations stellaires, mais est très communes dans plusieurs branches de la physique, car elle permet souvent d'identifier les modes naturels d'oscillation d'un système, ou de distinguer entre diverses types de variabilité dans une mesure expérimentale (variations stochastique, multipériodique, chaotique, turbulente, etc.).

4.4 Lire des données numériques sur fichier

La première chose que vous devrez faire est de copier les données de la Figure 4.1 dans votre répertoire local pour le labo 4. Ces données sont disponibles sur la partition Web de Mononcle Paul, et la manière la plus rapide de les ramasser est de taper:

```
wget http://www.astro.umontreal.ca/~ paulchar/phy1234/labos/G117-B15A.dat
```

Ce qui devrait copier le fichier dans le répertoire où vous tapez la commande, sous le même nom. Vous pouvez examiner le contenu de ce fichier via la commande Linux `more`. Il consiste en deux colonnes de nombre réels. La première correspond au temps d'observation, mesuré ici en secondes, et la deuxième au résidu d'intensité photométrique ΔI, soit la variation de l'intensité photométrique par rapport à la valeur moyennée sur la durée des observations.

Il s'agit maintenant de voir comment, en langage C, on peut lire ce fichier et emmagasiner les valeurs de t et ΔI dans des tableaux. Le petit bout de code C qui suit effectue ceci, et présuppose que le fichier de données est déjà copié dans le répertoire courant sous le nom `G117-B15A.dat`:

```
#include <stdio.h>
#define NDATAMAX 10000
int main(void)
{
/* Ce programme est un exemple de lecture de fichier */
/* Declarations ----------------------------------  */
   int  ndata=0 ;
   float t[NDATAMAX], di[NDATAMAX] ;
   FILE *fd ;
/* Executable ------------------------------------  */
   fd=fopen("G117-B15A.dat","r") ;
   while ( !feof(fd) && ndata < NDATAMAX )
     {
       fscanf (fd, "%f %f\n", &t[ndata], &di[ndata] ) ;
       ndata+=1 ;
     }
   if ( ndata == NDATAMAX ) { printf ("NDATAMAX atteint\n") ; }
   fclose(fd) ;
}
```

Il y a plusieurs choses à noter ici:

1. L'ouverture d'un fichier de données en mode lecture commence par la spécification d'un descripteur de fichier avec l'instruction `FILE`; attention, le "`*`" devant le nom du descripteur de fichier (ici `fd`) est important, il indique que `fd` est un *pointeur*, un type particulier de variable en C; ouverture avec `fopen`, avec le paramètre `"r"` pour "read" (ce serait `"w"`, pour "write", dans le cas d'un fichier où le code devrait écrire des données); et fermeture du fichier avec `fclose` une fois la lecture terminée.

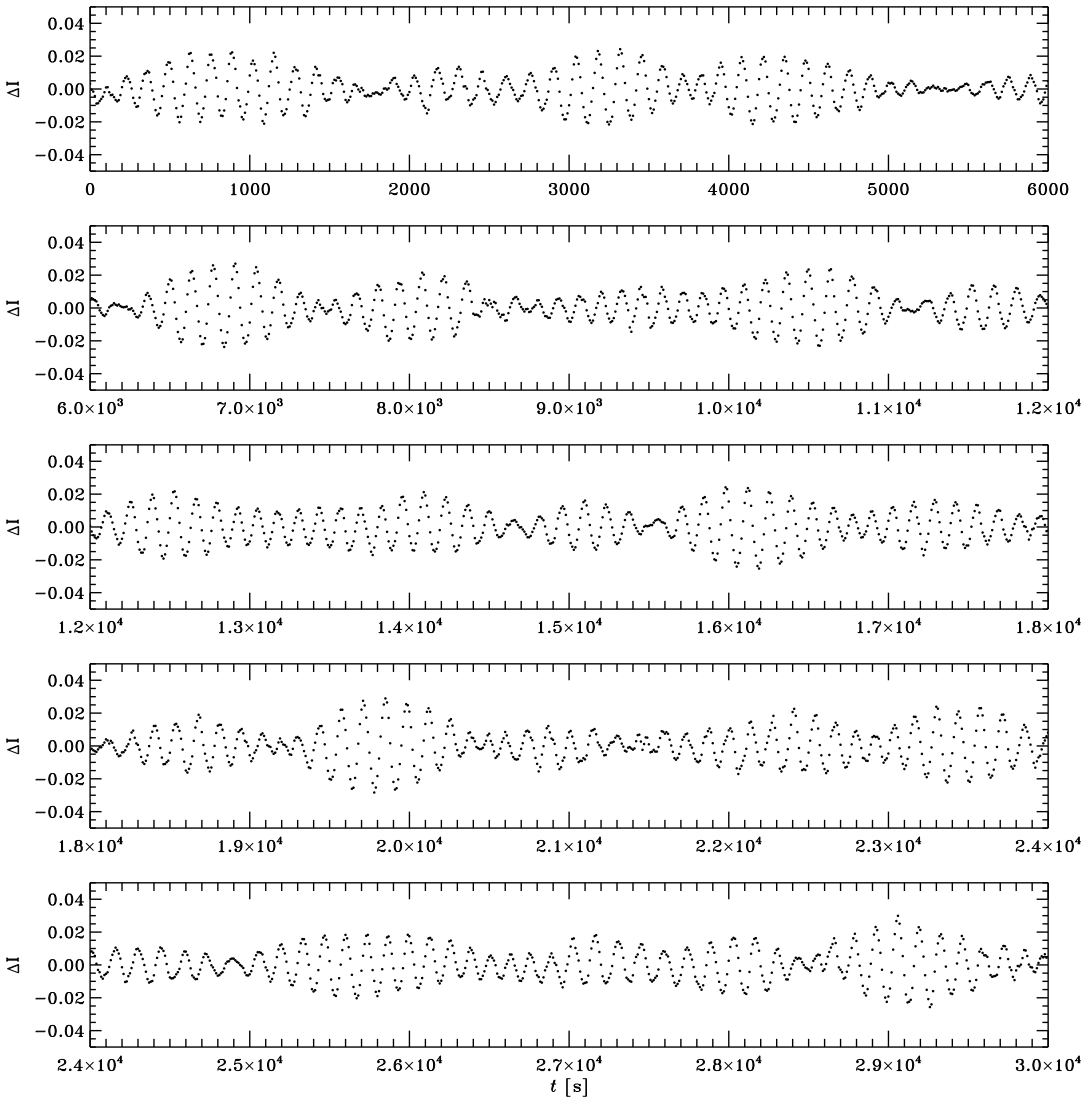

Figure 4.1: Fluctuations de la luminosité de l'étoile naine blanche G117-B15A. Dans cette étoile les pulsations déforment la surface et causent des hausses ou baisses de la température, qui se traduisent en faibles variations de la luminosité de l'étoile. Le résidu d'intensité $\Delta I(t)$ correspond ici à l'intensité au temps t moins la valeur moyenne de l'intensité sur l'ensemble des données; autrement dit, $\Delta I = 0$ correspond à la luminosité moyenne de l'étoile (ou, plus précisément, à l'intensité photométrique dans la bande passante du détecteur utilisé pour les observations; la luminosité y est directement proportionnelle). Il s'agit ici de réelles observations astronomiques gentiment fournies par mon collègue Pierre Bergeron.

2. La lecture même se fait avec l'instruction `fscanf`, le pendant de `scanf` pour la lecture de fichier plutôt que du clavier. Le premier argument de `fscanf` **doit** être le descripteur de fichier `fd` déclaré par `FILE` et initialisé par l'instruction `fopen`. Si vous lisez des données provenant de fichiers différents, chacun doit être ouvert avec `fopen` mais en assignant un descripteur distinct pour chaque fichier.

3. Comme on ne sait pas d'avance, en général, combien de lignes de données on va lire, on définit les longueurs des tableaux à une taille raisonnablement élevée (ici 10000), et on ajoute un test à la condition de lecture pour empêcher les débordement de tableaux. On ajoute également un test, après la boucle de lecture, visant à prévenir l'usager si cette taille maximale est atteinte (dans lequel cas il manquera des données, et on devrait ici augmenter la valeur de `NDATAMAX` dans l'instruction `#define` en en-tête au programme, recompiler et re-exécuter.).

4. Examinez bien la condition contrôlant la boucle `while` lisant les données. La fonction-macro C `feof(fd)` évaluera à `FAUX`, donc `!feof(fd)` à `VRAI`, jusqu'à ce qu'un caractère "fin-de-fichier" ait été rencontré à la lecture du fichier identifié par le descripteur `fd`. Ce caractère est ajouté automatiquement chaque fois que vous sauvegardez un fichier. C'est un caractère "fantôme", invisible dans la plupart des programmes d'édition de fichiers comme KATE.

5. À la sortie de la boucle de lecture, la variable d'incrément `ndata` aura une valeur égale au nombre de points de données dans la séquence temporelle lue (mais le dernier élément est le `[ndata-1]`, car C commence sa numérotation des éléments de tableaux à 0, pas 1; encore une fois, ATTENTION!). La valeur de `ndata` sera utile par la suite...

Bien des choses bizarres peuvent se passer quand on lit des données d'un fichier, donc il est primordial, avant de commencer de jouer avec les données, de bien vérifier ce qu'on a lu; porter les variables lues en graphiques est souvent une façon aisée de détecter des aberrations introduites au moment de la lecture. Votre première tâche est de produire, immédiatement après la lecture des donnés, un graphique de ΔI versus t, et de vérifier que ça ressemble à la Figure 4.1 (sans nécessairement segmenter le graphique en 5 sections, sauf pour les très enthousiastes du graphisme PLPLOT...).. Il s'agira simplement d'insérer les commandes PLPLOT appropriées dans le code de lecture ci-dessus.

Une autre approche courante est d'insérer une instruction `printf` qui donne à l'écran un "écho" des variables lues à mesure que ces variables sont lues, ligne par ligne. On n'est jamais trop prudent...

4.5 Analyse de Fourier

Comme vous aurez l'occasion de l'explorer à fond en PHY-1620, toute fonction continue d'une variable (t, disons) peut être représentée comme une somme de fonctions harmoniques, appelée **série de Fourier**:

$$f(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} \left(a_k \cos(\omega_k t) + b_k \sin(\omega_k t) \right) \ , \tag{4.1}$$

où

$$\omega_k = k\omega_0 \ , \qquad \omega_0 = \frac{2\pi}{T} \ , \tag{4.2}$$

la fréquence fondamentale ω_0 étant déterminée par l'intervalle $[0, T]$ sur lequel on veut ainsi représenter $f(t)$. Les coefficients numériques a_k et b_k sont donnés par les intégrales suivantes:

$$a_k = \frac{2}{T} \int_0^T f(t) \cos(\omega_k t) \mathrm{d}t \ , \tag{4.3}$$

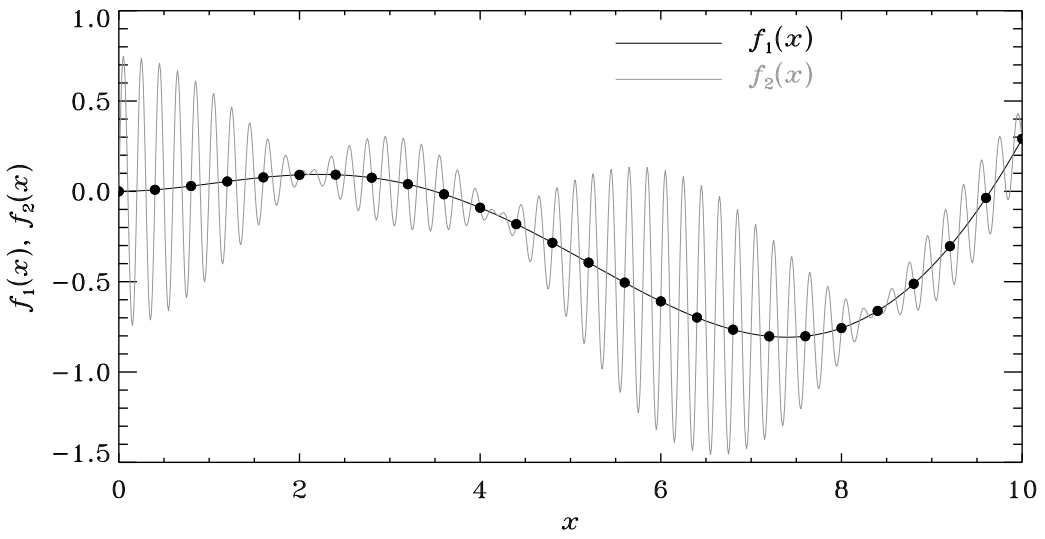

Figure 4.2: Illustration de l'idée de la fréquence de Nyquist. On peut toujours ajouter à une fonction quelconque n'importe quelle fonction harmonique de fréquence égale à un multiple de la fréquence de Nyquist associée au pas d'échantillonnage, sans rien changer aux valeurs échantillonnées de la fonction (voir texte).

$$b_k = \frac{2}{T}\int_0^T f(t)\sin(\omega_k t)\mathrm{d}t \ , \tag{4.4}$$

avec le coefficient $a_0/2$ dans l'éq. (4.1) correspondant alors à la valeur moyenne de $f(t)$ sur l'intervalle temporel considéré, soit $t \in [0, T]$. Pour les résidus photométriques considérés ici, selon leur définition même on a $a_0 = 0$.

Considérons maintenant une situation où $f(t)$ est échantillonnée à un nombre fini de points t_i:

$$f(t) \to f(t_i) \equiv f_i \ , \qquad t_i \in [0, T] \ ; \tag{4.5}$$

Dans le cas qui nous préoccupe, les paires (t_i, f_i) correspondent à chaque point de données sur la Figure 4.1, donc $i = 1, ..., \texttt{ndata}$. La fréquence fondamentale est toujours donnée par $\omega_0 = 2\pi/T$, mais il est maintenant superflu de calculer la somme dans l'éq. (4.1) jusqu'à $k \to \infty$; en effet, la discrétisation de t impose une fréquence maximale qui puisse être mesurée dans les données. Ceci est illustré sur la Figure 4.2. Les points noirs correspondent à une série de "données" dont on veut calculer la transformée de Fourier. Imaginons maintenant que ces données représentent un échantillonnage discret à intervalle de temps constant h d'une fonction continue. Cette fonction pourrait bien avoir l'air du trait noir sur la Figure. Cependant, les données pourraient tout aussi bien avoir été produites par la fonction correspondant au trait gris, qui offre une représentation tout aussi exacte des données que le trait noir. Il est essentiel de bien apprécier que *les données ne contiennent pas d'information nous permettant de distinguer entre ces deux possibilités.* Plus précisément, toute fonction harmonique ayant une période plus petite que *deux* points de maille ne laissera aucune trace cohérente dans les données. Cette fréquence de coupure est donc donnée par

$$\omega_c = \frac{\pi}{h} \ , \tag{4.6}$$

et est appellée **fréquence de Nyquist**. C'est la fréquence à laquelle la somme doit être tronquée dans l'éq. (4.1); la valeur correspondante de k est facile à calculer:

$$k_c = \frac{\omega_c}{\omega_0} = \frac{T}{2h} \ . \tag{4.7}$$

Dans votre cas, le h correspond à l'intervalle de temps entre deux points de données successifs, soit 10 secondes pour les observations de la Fig. 4.1.

Mais quel peut bien être l'avantage de représenter nos données en termes d'une série de Fourier? L'attrait vient du fait que pour un signal multipériodique qui ne contient que quelques modes ayant des périodes bien définies, seul quelques coefficients a_k et b_k auront une amplitude substantielle, et les fréquences correspondantes seront celles présentes dans le signal. Une telle **analyse de Fourier** offre donc une approche très efficace pour extraire les périodes présentes dans un signal multipériodique, et c'est ce que vous aurez à faire dans ce labo. Plus spécifiquement, il s'agira de calculer le **spectre de puissance**, soit la variation de la quantité

$$P(\omega_k) = \sqrt{a_k^2 + b_k^2} \ , \qquad k = 1, ..., k_c \tag{4.8}$$

en fonction de la fréquence $\omega_k \equiv k \times \omega_0$.

4.6 Analyse de Fourier, en version numérique

Le calcul des coefficients de Fourier a_k et b_k, requis dans le calcul du pectre de puissance, demande donc une calcul répété des intégrales (4.3) et (4.4), soit une évaluation par valeur de k, cette quantité variant de $k = 1$ à $k = k_c$ tel que donné par l'éq. (4.7). Il sera donc judicieux d'imbriquer le calcul de ces coefficients à l'intérieur d'une boucle appropriée, qui fera appel *à deux reprises* à une fonction style `integretrapeze1D`, une fois pour calculer a_k et la seconde pour calculer b_k. le calcul de l'intégrant est plus complexe cette fois, car `integretrapeze1D` doit accepter en argument deux tableaux 1D contenant les données, et la fréquence $\omega_k = k\omega_0$. Un appel pourrait avoit l'air de:

```
omega0= ... ;              /* La frequence fondamentale */
kc    = ... ;              /* La frequence de coupure */
for ( k=1 ; k < kc ; k++ ) {
   omega[k]=omega0*k ;
   a[k]=integreTrapeze1D(t,di,ndata,omega[k],code) ;
   b[k]= ... ;
}
```

où `code` est un entier valant 0 ou 1. L'idée est d'appeller la même fonction `integretrapeze1D` pour le calcul des a_k et b_k; cependant les a_k ont un $\cos(\omega t)$ dans l'intégrant, tandis que les b_k ont un $\sin(\omega t)$. La variable `code` peut donc être passée en argument à la fonction calculant l'intégrant (e.g., la fonction `ff`, dans le canevas présenté à la dernière section 3.5 du labo 3), et utilisée à l'intérieur de cette fonction pour contrôler une structure conditionnelle de type **if** ... **else** de manière à calculer l'expression mathématique correspondant à l'intégrant désiré.

Avant de continuer, remarquez bien, et comprenez bien, que vous avez ici deux séries de tableaux 1D contenant des nombres différents d'élements: les tableaux `t` et `di` contiennent les `ndata` points de données dont vous voulez calculer le spectre; mais les tableaux `a` et `b` contiennent les `kc` coefficients de Fourier correspondant aux `kc` fréquences $\omega_k = k \times \omega_0$ emmagasinées dans le tableau `omega` et qui devront être utilisées pour calculer le spectre de puissance, qui se retrouvera également contenu dans un tableau 1D contenant `kc` elements.

4.7 Validation

Nous allons (évidemment!) commencer par une validation utilisant un signal bi-périodique bidon. Utilisant les t_k (en version originale mesurés en secondes) associés aux données de la Figure 4.1, construisez un tableau `bidon` contenant les valeurs:

$$B(t_i) \equiv B_i \equiv \texttt{bidon[i]} = 3\sin\left(\frac{2\pi}{500}\texttt{t[i]}\right) + 5\cos\left(\frac{2\pi}{200}\texttt{t[i]}\right) \ , \qquad \texttt{i} = 0, ...\texttt{ndata} - 1 \tag{4.9}$$

Il s'agira de calculer la transformée de Fourier de ce signal artificiel, et de vérifier que les deux fréquences qui en ressortent sont bel et bien $2\pi/500$ et $2\pi/200$. Les étapes sont les suivantes:

1. Calculez la fréquence fondamentale ω_0, la fréquence de Nyquist ω_c, et le nombre critique k_c pour l'échantillonnage temporel des données de la Figure 4.1, qui sera identique à celui de la séquence `bidon` puisque celle-ci est construite à l'aide de la même discrétisation temporelle `t[ndata]`.

2. Calculez les coefficients a_k et b_k définis par les éqs. (4.3) et (4.4) par la méthode du trapèze (§2.4.1 des Notes de cours, et plus spécifiquement l'éq. (2.40); et labo 3). Notez bien, encore une fois, que vous devrez calculer les a_k et b_k pour chaque multiple de la fréquence fondamentale, jusqu'au k_c correspondant à la fréquence de Nyquist. Il serait judicieux ici d'utiliser la structure de code développée en dernière partie du labo 3, et le petit truc décrit ci-dessus avec le paramètre `code`. Les valeurs de ω_k et les a_k et b_k correspondant devraient être emmagasinés dans trois tableaux 1D de taille appropriée, vous en aurez besoin plus loin pour un graphique.

3. Calculez maintenant le spectre de puissance (éq. (4.8)), en emmagasinez le résultat dans un tableau 1D de taille appropriée (n'oubliez pas que vous avez une valeur de a_k et une de b_k pour chaque fréquence ω_k.

4. Portez en graphique le spectre de puissance , soit la courbe $P(\omega_k)$ versus ω_k; Ça devrait ressembler à la Fig. 4.3 ci-dessous. Essayer en version log-lin, comme sur la Fig. 4.3 (`style`= 20 dans l'appel à `plenv`), et aussi en version habituelle avec des axes linéaires (`style`= 1). Encore une fois, pour le graphique log-lin, n'oubliez pas de prendre le logarithme du spectre de puissance AVANT l'appel à `plline`, et de ne PAS poser `ymin`= 0.

5. A partir de vos résultats, déterminez maintenant les fréquences et amplitudes correspondant aux deux pics dans le spectre de puissance. En quoi ces valeurs correspondent-elles (ou pas) à l'équation (4.9)?

Reprenez l'exercice ci-dessus, i.e., avec les données bipériodiques bidons, mais cette fois en doublant l'échantillonnage en fréquence. Mécaniquement, le plus simple est de doubler artificiellement la fréquence fondamentale ω_0. Comparez le spectre résultant de cette manoeuvre avec le précédent, et discutez les similarités et différences. Croyez vous être en mesure de déterminer plus précisément les fréquences des pics avec une plus grande résolution en fréquence? Pourquoi?

4.8 Sorties graphiques

Comme vous aurez un rapport de lab à remettre cette semaine, il vous faudra sauvegarder vos graphiques sous une forme imprimable, ou insérable dans un programme d'édition de texte, si vous en utilisez un pour vos rapports. Le choix du format graphique de sortie se fait au moment de l'exécution, en fournissant un code numérique approprié parmi la sélection proposée interactivement par `plinit`. À date vous n'avez utilisé que la sortie graphique à l'écran, mais plusieurs autres options sont disponibles. Celles qui vous seront probablement les plus utiles sont postscript noir-et-blanc (option 4) ou couleur (option 13), svg (option 14), pdf (option 16) et png (option 19).

Vous aurez déjà remarqué que les couleurs de défaut, pour une sortie à l'écran (option 1), définissent un trait rouge sur fond noir; ceci demeurera le cas pour les sorties en png ou pdf; si cependant la sortie choisie est de type postscript, le tracé sera noir sur fond blanc. Pour forcer un fond blanc, vous devez appeler la fonction suivante *avant* votre appel à `plinit`:

```
plscolbg(255,255,255) ;
```

Remarquez que le nom de cette fonction commence par `pls...` plutôt que `pl...`; le "`s`" indique une fonction affectant les variables internes au système d'exploitation de l'ordi, plutôt qu'une fonction effectuant une action graphique.

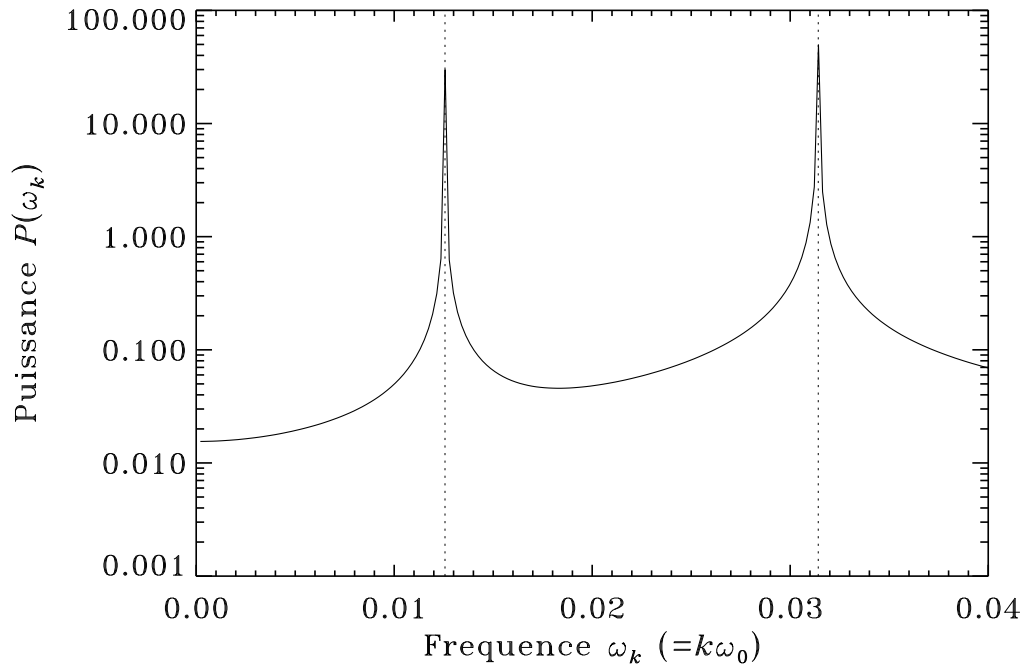

Figure 4.3: Spectre de puissance de la séquence temporelle bipériodique bidon définie par l'éq. (4.9). Les deux traits pointillés verticaux indiquent les fréquences correspondantes aux deux pics dans le spectre. Pour tracer un tel joli graphique log-lin en PLPLOT, il faut d'abord avoir pris le logarithme du spectre de puissance avant l'appel à `plline`, et avoir utilisé la valeur `style`= 20 pour le sixième paramètre de `plenv`; pour un graphique lin-log, ça aurait été plutôt `style`= 10.

4.9 Les fréquences d'oscillations de G117-B15A

Il est temps de revenir à G117-B15A. Utilisez exactement la procédure précédente (mais sans doubler la fréquence fondamentale ω_0) pour calculer le spectre de puissance des données photométriques de G117-B15A. Le spectre devrait être passablement plus complexe que celui des données bidons de la section précédente.

Il s'agit maintenant de développer une procédure algorithmique pour identifier les pics en fréquence. Considérons la stratégie suivante; une boucle (indice k) balayant les `kc` fréquences du spectre, de $k = 1$ à $k =$ `kc` $- 2$ (RAPPEL: en C les éléments d'un tableau de longueur N sont numérotés de 0 à $N-1$!!). Si un k correspond à un pic dans le spectre, alors on doit avoir

$$P_k > P_{k-1} \qquad \text{et} \qquad P_k > P_{k+1} \ .$$

Il s'agit de mettre en place un second indice j (disons) qui accumulera (et numérotera) les pics ainsi découverts; ça pourrait avoir l'air de ceci:

```
   ...
/* recherche de pics --------------------------------  */
   j=0 ;                          /* compteur de pics  */
   for ( k=1 ; k < kc-1 ; k++ ) {
      if ( p[k] > p[k-1] && p[k] > p[k+1] ) {
         frqpic[j] = omega0 * k ; /* frequence du pic j */
         amppic[j] = p[k] ;       /* puissance du pic j */
         j++ ;
```

```
   }
if ( j == 0 ) { printf ("Aucun pic dans le spectre\n") ; }
}
...
```

où `kc` est le nombre de points en fréquence les tableaux `frqpic` et `amppic` contiendront au final une liste des fréquences et amplitude des pics ainsi extraits, dont le nombre sera égal à la valeur de j à la sortie de la boucle. Notez le petit test de sécurité vérifiant si au moins un pic a été identifié; si ce n'est pas le cas, il y a certainement une erreur de codage quelquepart dans votre truc!

Produisez un tableau contenant la liste des fréquences et puissances correspondantes pour les dix plus hauts pics du spectre de GD117-B15A; réfléchissez à la manière dont on pourrait décider quels pics sont physiquement significatifs et lesquels ne le sont pas, et concluez votre rapport avec une brève discussion de ce point, absolument critique dans l'analyse de Fourier de tout signal multipériodique.

Et voilà, le labo 4 se termine ici!

Lectures supplémentaires:

Notes de cours: Section 2.4

Delannoy — premier langage : Sections 6.4, 8.1, 8.2

Delannoy — programmer en C : Sections 6.4, 10.4, 10.5