Christian Gößl 762627
BaPh 6 18. September 2014

Programm zur Erstellung von Periodogrammen

1 Einleitung

Dieses Programm passiert auf den Ausführungen aus den Paper von Horne & Baliunas (1986) [1], worin eine Methode zur Berechnung von Periodogrammen vorgestellt wird. Diese Periodogramme sind das Ergebnis einer Fouriertransformation mit den Erweiterungen aus dem Paper. Das Programm ist in der Programmiersprache C++ geschrieben und kann zur Auswertung von Zeitreihen benutzt werden. Der vollständige Quellcode kann im Appendix eingesehen werden.

2 Funktionen des Programms

Es kann eine Zeitreihe in dieser Form (t,x,σ) eingelesen werden, daraus wird ein Periodogramm und im aktuellen Verzeichnis die Ergebnisdaten im Format (ω , Amplitude) in einer Datei mit dem Namen *Fourier-* am Anfang erstellt. Die Zeitreihe wird zur weiteren Verarbeitung in 2 Felder gespeichert. Beim Start des Programms wird der Nutzer aufgefordert den Namen der zu untersuchende Datei einzugeben, gefolgt mit der Eingabe der größtmöglichen gewollten Frequenz ω. Dies wird durch die Funktion initial bereitgestellt.

```
40 void initial( double &w, double &deltaw, double &ende, double x
       [], double t[], int &N_0, char Dateiname[])
41 {
42   w = 0.;
43   cout << "Geben sie den Dateinamen ein der sich im aktuellen
       Verzeichnis befindet:" << endl;
44   cin >> Dateiname;
45   cout << "Geben sie ein oberes Intervallende der Frequenzen
       ein:" << endl;
46   cin >> ende;
47   N_0 = ReadData( x, t, Dateiname); // zur Bestimmung der Größ
       e/Länge der Arrays
48   deltaw = 1. / (t[N_0] - t[0]);
49 }
```

Diese und weitere Funktion werden in der Prozedur *main* gestartet.

```
22 int main()
23 {
24   int
25     N_0// Anzahl der Daten
26   ;
27   double
28     w, //Frequenz
29     deltaw, // Schrittweite der Omegas Frequenzen
30     ende // Intervallende der Frequenzen
```

[1] The Astrophysical Journal, 302:757-763, 1986 March 15 Horne & Baliunas http://adsabs.harvard.edu/abs/1986ApJ...302..757H

```
31      , x[MaximalLaenge] // Messwerte
32      , t[MaximalLaenge] // Zeitwerte
33      ;
34    char Dateiname[200];
35    BuildSignal(); // Test Signal erzeugen
36    initial( w, deltaw, ende, x, t, N_0, Dateiname); //
        Anfangswerteerzeugung
37    FourierTrafo( w, deltaw, ende, x, t, N_0, Dateiname);
38  }
```

Zu Testzwecken kann der Name *TestDaten.dat* mit $\omega = 10$ eingeben werden, der Name der Ausgabedatei lautet dann *Fourier-TestDaten.dat*. Die TestDaten werden in der Funktion *BuildSignal* erzeugt, dabei wird zu einer Cosinus Funktion ein Gaußsches Rauschen aufaddiert.

```
51  void BuildSignal()
52  {
53    ofstream dateiout ; // Objekt der Klasse ofstream anlegen
54    dateiout.open("TestDaten.dat") ; // öffnen der Datei
55    int i = 0 , vorzeichen;
56    double deltat = 0.01, ampl = 0.75, freq = 5., ende = 10., x;
        // Anfangswerte für das Test Signal
57    double random, sigma = 0.1, T;
58    time_t ttime ;
59    time(&ttime) ;
60    srand((unsigned)time((long *) 0)) ;// initialisieren des
        Zufallsgenerators
61    for (double t = 0. ; t <= ende ; t = t + deltat)
62    {
63      srand(i);
64      random = rand()/ (RAND_MAX + 1.0);// Zufallszahlen zwischen
          0 und 1
65      x = random*(0.3);
66      if (rand() % 2 == 0) //damit die Streuung nicht nur
          aufaddiert sondern auch mal substrahiert wird
67      vorzeichen = +1.;
68      else vorzeichen = -1.;
69      dateiout << t << "  " << ( ampl * cos(freq * t) + 1 *
          vorzeichen * 1. /(sigma * sqrt(2.*M_PI) ) * exp(-(x *x)
          /(2. * sigma *sigma) ) )<< " " << 3. << endl ;
70      i = i + 1;
71    }
72  }
```

Den Median der Zeitreihe wird in der Funktion *median* berechnet, wobei ein Sortieralgorithmus (Minimumsuche) verwendet wird.

```
160 void median( double x[], int &N_0, double &x_med)
161 {
162   int Min_x, i ,j;
163   double x_temp[N_0];
```

```
164   for (int k = 0; k < N_0; k++) // temporäries Feld zur
          Bestimmung des Medians
165   {
166     x_temp[k] = x[k];
167     //cout << x_temp[k] << endl;
168   }
169   for (j = 0; j < (N_0 - 1); j++) // Sortierung des Arrays
170   {
171     Min_x = j; // Anfangsminimum setzen
172     for (i = j + 1; i < N_0; i++) // Suche nach dem kleinsten
          Wert in dem verbleibenen Array
173     {
174       if ((x_temp[i]) < (x_temp[Min_x]) )
175         {
176         Min_x = i; // gefundenes Minimum speichern
177       }
178     }
179     if(Min_x != j) // gefundenes Minimum mit der Einganszahl in
          der Schleife tauschen
180     {
181         x_temp[Min_x] = x_temp[Min_x] + x_temp[j];
182         x_temp[j] = x_temp[Min_x] - x_temp[j];
183         x_temp[Min_x] = x_temp[Min_x] - x_temp[j];
184     }
185   }
186   if (fmod(N_0, 2) == 0) // Median Bestimmung wenn Länge des
        Arrays gerade ist
187   {
188     x_med = (double(x_temp[N_0 / 2]) + double(x_temp[(N_0) / 2
          + 1])) / 2.;
189   }
190   else // für ungerade
191   {
192     x_med = x_temp[(N_0) / 2 + 1];
193   }
194   /*for (int j = 0; j < (N_0); j++)
195   {
196     cout << x_temp[j] << endl;
197   }*/
198   //cout << x_temp[N_0 / 2] << " " << x_temp[(N_0) / 2 + 1] <<
        endl;
199   //cout << x_med  << endl;
200 }
```

Der Median wird dann in der Funktion *FourierTrafo* verwendet. Dort werden die Formeln zur Berechnung der Fouriertransformation aus dem Paper von Horne & Baliunas (1986) verwendet. Aus einer Zeitreihe $X(t_j)$ mit $i = 1, ..., N_0$ wird ein Periodogramm erzeugt mit der folgenden Funktion $P_X(\omega)$. Zu beachten ist das hier zusätzlich der Median X_{med} bei der Zeitreihe vorher abgezogen wird.

$$P_X(\omega) = \frac{1}{2}\left(\frac{\left[\sum_{j=1}^{N_0}(X(t_j)-X_{med})cos(\omega(t_j-\tau)\right]^2}{\sum_{j=1}^{N_0}(X(t_j)-X_{med})cos^2(\omega(t_j-\tau)} + \frac{\left[\sum_{j=1}^{N_0}(X(t_j)-X_{med})sin(\omega(t_j-\tau)\right]^2}{\sum_{j=1}^{N_0}(X(t_j)-X_{med})sin^2(\omega(t_j-\tau)}\right)$$

$$tan(2\omega\tau) = \left(\sum_{j=1}^{N_0} sin(2\omega t_j)\right)/\left(\sum_{j=1}^{N_0} cos(2\omega t_j)\right)$$

Diese Funktion muss noch normiert werden mit der absoluten Varianz.

$$\sigma^2 = \sum_{j=1}^{N_0}(X(t_j)-X_{med})^2$$

$$P_N(\omega) = {}^{P_X(\omega)}/_{\sigma^2}$$

```
101 void FourierTrafo( double &w, double &deltaw, double &ende,
        double x[], double t[], int &N_0, char Dateiname[])
102 {
103   double
104     P = 0., alt_P = 0., w_0 = 0.
105     , co = 0., co2 = 0., si = 0., si2 = 0., var = 0.//
            Hilfsvariablen für die Summenberechnung
106     , taw = 0. , x_med = 0.;
107   char Dateiout[208] = "Fourier-";
108   ofstream dateiout ; // Objekt der Klasse ofstream anlegen
109   for (int i = 0; i < 200 ; i = i + 1) // Namenserstellung der
          Outputdatei
110   {
111     Dateiout[i + 8] = Dateiname[i];
112   }
113   dateiout.open(Dateiout);
114   median(x, N_0, x_med); // Median Bestimmung
115   for (w; w <= ende ; w = w + deltaw)
116   {
117     co = 0.;
118     co2 = 0.;
119     si = 0.;
120     si2 = 0.;
121     var = 0.;
122     taw = tau(w, t, N_0); // Bestimmung von Tau
123     if (w != 0.) // wenn der Sinus bei Null berechnet wird,
            wird bei der Berechnung von P durch Null geteilt
124     {
125       for (int i = 0 ; i < N_0 ; i++) // Summenberechnung der
              Fourieranalyse plus Varianz Bestimmung
126       {
127         co = co + (x[i] - x_med) * cos(w * (t[i] - taw));//
128         co2 = co2 + cos(w * (t[i] - taw)) * cos(w * (t[i] -
              taw));
129         si = si + (x[i] - x_med) * sin(w * (t[i] - taw));//
130         si2 = si2 + pow(sin(w * (t[i] - taw)),2.);
```

```
131             var = var + (x[i] - x_med) * (x[i] - x_med);//
132           }
133           P = (1. / 2. * (co * co/ co2 + si * si / si2)) / (var /
                  (double) N_0);
134           if (P > alt_P) // Bestimmung der größten Frequenz
135           {
136             w_0 = w;
137             alt_P = P;
138           }
139       }
140       else P = 0.;
141       dateiout << w << " " << P << endl ; // Ergebnisse in die
              offene Datei schreiben
142     }
143     dateiout << "# w_0 = " << w_0 << " P_0 = " << P << endl ; //
          Ergebnisse in die offene Datei schreiben
144 }
```

Zusätzlich wird die Frequenz ω_0 des stärksten Signals gesucht und in die Ausgabedatei geschrieben.

```
134           if (P > alt_P) // Bestimmung der größten Frequenz
135           {
136             w_0 = w;
137             alt_P = P;
138           }
139       }
140       else P = 0.;
141       dateiout << w << " " << P << endl ; // Ergebnisse in die
              offene Datei schreiben
142     }
143     dateiout << "# w_0 = " << w_0 << " P_0 = " << P << endl ; //
          Ergebnisse in die offene Datei schreiben
```

3 Anwendung des Programms

Zur Anwendung werden die folgenden Zeitreihen verwendet *lmc-cepheid.dat*, *x-ray.dat* und *doubleP.dat*. Die sich ergebenden Periodogramme wurden mittels Gnuplot ausgewertet (Siehe Abb. 1). Der Gnuplot Code zur Erzeugung der Diagramme befindet sich im Appendix.

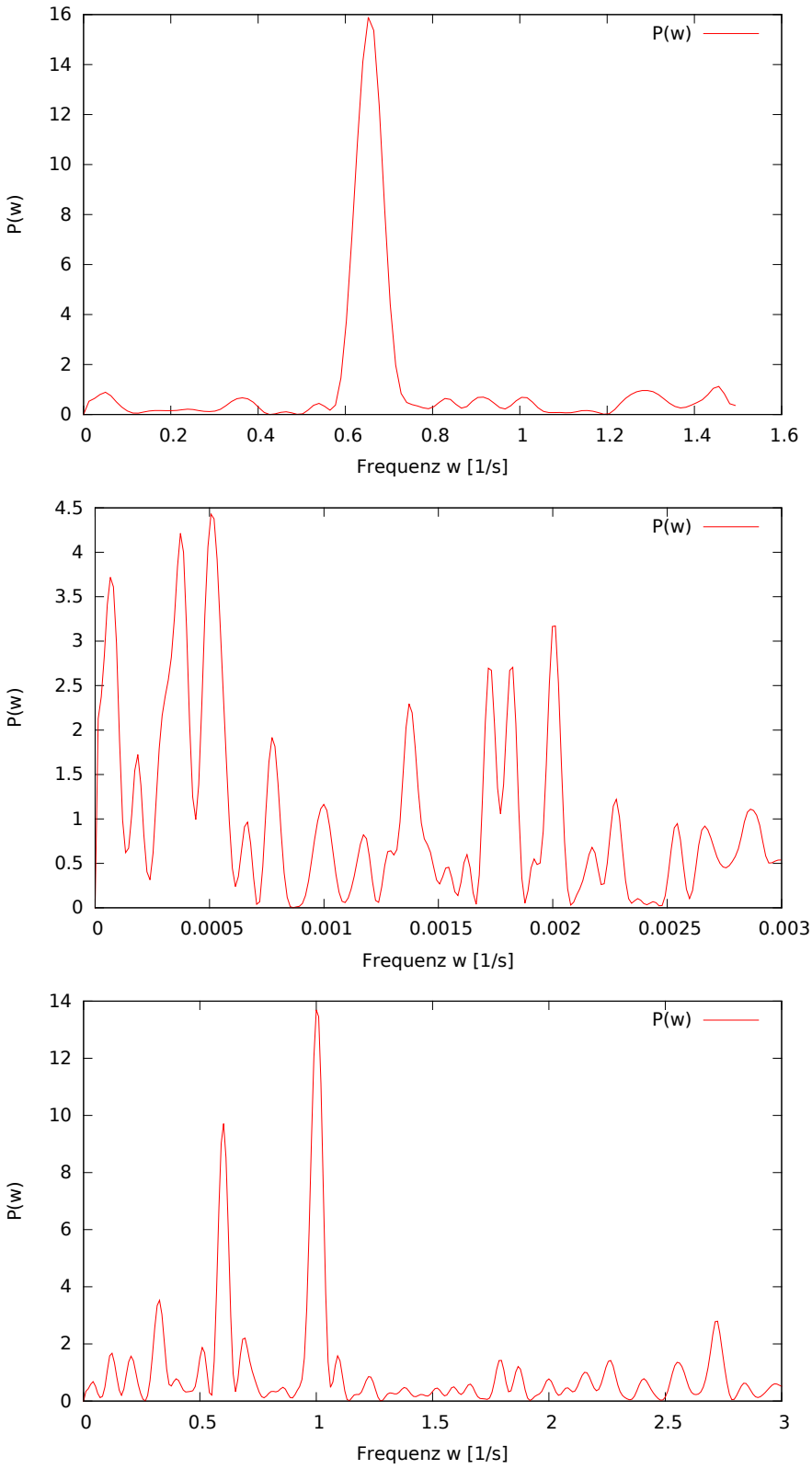

Abbildung 1: Periodogramme der Datensätze v.o.n.u *lmc-cepheid.dat*, *x-ray.dat* und *doubleP.dat*

4 Appendix

4.1 gesamter Quellcode

```
1  #include <stdlib.h>
2  #include <iostream>
3  #include <iomanip>
4  #include <cmath>
5  #include <cstdlib>
6  #include <time.h>
7  #include <fstream>
8  #include <stdlib.h>      /* strtod */
9
10 using namespace ::std ;
11
12 double tau(double &w, double t[], int &N_0);
13 double ReadData( double x[], double t[], char Dateiname[]);
14 void FourierTrafo( double &w, double &deltaw, double &ende,
       double x[], double t[], int &N_0, char Dateiname[]);
15 void initial( double &w, double &deltaw, double &ende, double x
       [], double t[], int &N_0, char Dateiname[]);
16 void BuildSignal();
17 void median( double x[], int &N_0, double &x_med);
18
19
20 const long MaximalLaenge = 499999;
21
22 int main()
23 {
24   int
25     N_0// Anzahl der Daten
26   ;
27   double
28     w, //Frequenz
29     deltaw, // Schrittweite der Omegas Frequenzen
30     ende // Intervallende der Frequenzen
31     , x[MaximalLaenge] // Messwerte
32     , t[MaximalLaenge] // Zeitwerte
33     ;
34   char Dateiname[200];
35   BuildSignal(); // Test Signal erzeugen
36   initial( w, deltaw, ende, x, t, N_0, Dateiname); //
       Anfangswerteerzeugung
37   FourierTrafo( w, deltaw, ende, x, t, N_0, Dateiname);
38 }
39
40 void initial( double &w, double &deltaw, double &ende, double x
       [], double t[], int &N_0, char Dateiname[])
41 {
42   w = 0.;
```

```
43    cout << "Geben sie den Dateinamen ein der sich im aktuellen
          Verzeichnis befindet:" << endl;
44    cin >> Dateiname;
45    cout << "Geben sie ein oberes Intervallende der Frequenzen
          ein:" << endl;
46    cin >> ende;
47    N_0 = ReadData( x, t, Dateiname); // zur Bestimmung der Größ
          e/Länge der Arrays
48    deltaw = 1. / (t[N_0] - t[0]);
49  }
50
51  void BuildSignal()
52  {
53    ofstream dateiout ; // Objekt der Klasse ofstream anlegen
54    dateiout.open("TestDaten.dat") ; // öffnen der Datei
55    int i = 0 , vorzeichen;
56    double deltat = 0.01, ampl = 0.75, freq = 5., ende = 10., x;
          // Anfangswerte für das Test Signal
57    double random, sigma = 0.1, T;
58    time_t ttime ;
59    time(&ttime) ;
60    srand((unsigned)time((long *) 0)) ;// initialisieren des
          Zufallsgenerators
61    for (double t = 0. ; t <= ende ; t = t + deltat)
62    {
63      srand(i);
64      random = rand()/ (RAND_MAX + 1.0);// Zufallszahlen zwischen
            0 und 1
65      x = random*(0.3);
66      if (rand() % 2 == 0) //damit die Streuung nicht nur
          aufaddiert sondern auch mal substrahiert wird
67      vorzeichen = +1.;
68      else vorzeichen = -1.;
69      dateiout << t << "  " << ( ampl * cos(freq * t) + 1 *
          vorzeichen * 1. /(sigma * sqrt(2.*M_PI) ) * exp(-(x *x)
          /(2. * sigma *sigma) ) )<< " " << 3. << endl ;
70      i = i + 1;
71    }
72  }
73
74  double ReadData( double x[], double t[], char Dateiname[])
75  {
76    int k = 0;
77    double g; // zum Einlesen der Sigmas die hier nicht weiter
          verwendet werden
78    char text[40], buffer[100];
79    ifstream dateiin ; // Objekt der Klasse ifsteam anlegen
80    dateiin.open(Dateiname) ;
81    while ( !dateiin.eof() ) // solange einlesen bis Dateiende
          erreicht
```

```
82    {
83      dateiin >> text;
84      if (atof(text) == 0) // Überprüfung ob das erste Zeichen
            ein Zahl ist
85      {
86          dateiin.getline(buffer,100) ; // einlesen der ganzen
              Zeile damit bei der nächste Zeile angefangen wird
87      }
88      if  (atof(text) != 0)
89      {
90          t[k] = strtod(text, NULL); // Anfangs eingelesene Zahl
              in das Array packen
91          dateiin >> x[k] >> g;
92          //cout << t[k] <<" " << x[k] << " " << g << endl ;
93          k = k + 1;
94      }
95    }
96    dateiin.close() ;
97    //cout << (k - 1) << endl;
98    return (k - 1); // Leider geht die Schleife weiter als nötig
        und der zu viel gezählte Counter abgezogen werden
99  }
100
101 void FourierTrafo( double &w, double &deltaw, double &ende,
      double x[], double t[], int &N_0, char Dateiname[])
102 {
103   double
104     P = 0., alt_P = 0., w_0 = 0.
105     , co = 0., co2 = 0., si = 0., si2 = 0., var = 0.//
          Hilfsvariablen für die Summenberechnung
106     , taw = 0. , x_med = 0.;
107   char Dateiout[208] = "Fourier-";
108   ofstream dateiout ; // Objekt der Klasse ofstream anlegen
109   for (int i = 0; i < 200 ; i = i + 1) // Namenserstellung der
        Outputdatei
110   {
111     Dateiout[i + 8] = Dateiname[i];
112   }
113   dateiout.open(Dateiout);
114   median(x, N_0, x_med); // Median Bestimmung
115   for (w; w <= ende ; w = w + deltaw)
116   {
117     co = 0.;
118     co2 = 0.;
119     si = 0.;
120     si2 = 0.;
121     var = 0.;
122     taw = tau(w, t, N_0); // Bestimmung von Tau
123     if (w != 0.) // wenn der Sinus bei Null berechnet wird,
          wird bei der Berechnung von P durch Null geteilt
```

```
124     {
125         for (int i = 0 ; i < N_0 ; i++) // Summenberechnung der
                Fourieranalyse plus Varianz Bestimmung
126         {
127           co = co + (x[i] - x_med) * cos(w * (t[i] - taw));//
128           co2 = co2 + cos(w * (t[i] - taw)) * cos(w * (t[i] -
                taw));
129           si = si + (x[i] - x_med) * sin(w * (t[i] - taw));//
130           si2 = si2 + pow(sin(w * (t[i] - taw)),2.);
131           var = var + (x[i] - x_med) * (x[i] - x_med);//
132         }
133         P = (1. / 2. * (co * co/ co2 + si * si / si2)) / (var /
                (double) N_0);
134         if (P > alt_P) // Bestimmung der größten Frequenz
135         {
136           w_0 = w;
137           alt_P = P;
138         }
139     }
140     else P = 0.;
141     dateiout << w << " " << P << endl ; // Ergebnisse in die
            offene Datei schreiben
142   }
143   dateiout << "# w_0 = " << w_0 << " P_0 = " << P << endl ; //
        Ergebnisse in die offene Datei schreiben
144 }
145
146 double tau(double &w, double t[], int &N_0)
147 {
148   double
149     si = 0.,
150     co = 0.
151     ;
152   for (int i = 0 ; i < N_0 ; i++)
153   {
154     si = si + sin( 2. * w * t[i]);
155     co = co + cos( 2. * w * t[i]);
156   }
157   return (atan(si / co) / ( 2. * w));
158 }
159
160 void median( double x[],int &N_0, double &x_med)
161 {
162   int Min_x, i ,j;
163   double x_temp[N_0];
164   for (int k = 0; k < N_0; k++) // temporäries Feld zur
        Bestimmung des Medians
165   {
166     x_temp[k] = x[k];
167     //cout << x_temp[k] << endl;
```

```
168   }
169   for (j = 0; j < (N_0 - 1); j++) // Sortierung des Arrays
170   {
171     Min_x = j; // Anfangsminimum setzen
172     for (i = j + 1; i < N_0; i++) // Suche nach dem kleinsten
          Wert in dem verbleibenen Array
173     {
174       if ((x_temp[i]) < (x_temp[Min_x]) )
175         {
176         Min_x = i; // gefundenes Minimum speichern
177       }
178     }
179     if(Min_x != j) // gefundenes Minimum mit der Einganszahl in
          der Schleife tauschen
180     {
181         x_temp[Min_x] = x_temp[Min_x] + x_temp[j];
182         x_temp[j] = x_temp[Min_x] - x_temp[j];
183         x_temp[Min_x] = x_temp[Min_x] - x_temp[j];
184     }
185   }
186   if (fmod(N_0, 2) == 0) // Median Bestimmung wenn Länge des
        Arrays gerade ist
187   {
188     x_med = (double(x_temp[N_0 / 2]) + double(x_temp[(N_0) / 2
          + 1])) / 2.;
189   }
190   else // für ungerade
191   {
192     x_med = x_temp[(N_0) / 2 + 1];
193   }
194   /*for (int j = 0; j < (N_0); j++)
195   {
196     cout << x_temp[j] << endl;
197   }*/
198   //cout << x_temp[N_0 / 2] << " " << x_temp[(N_0) / 2 + 1] <<
        endl;
199   //cout << x_med  << endl;
200 }
```

4.2 Gnuplot Code

```
1 set terminal pdfcairo enhanced color
2 set xlabel "Frequenz w [1/s]"
3 set ylabel "P(w)"
4 set output "plot-doubleP.pdf"
5 plot "Fourier-doubleP.dat" using ($1):($2) with lines title 'P(
    w)'
6 #replot
7 set terminal pdfcairo enhanced color
```