# 2011年冬学期
# 特別実験レポート

中間智弘
05091546

# Contents

# Chapter 1

# はじめに

重力波は、光速で伝搬する時空の歪みである。重力波の存在は、1916 年に A.Einstein によって予言されている。また、連星パルサー PSR1913+16 の観測からその存在は間接的に証明されている。これは、重力波の放出によって、連星パルサーの公転軌道が変化することが確認されたことによる結論である。この観測結果と理論による予測とは非常に高い精度で一致しており、重力波の存在を示唆する有力な証拠の一つとなっている。

しかし、重力波と物質との相互作用は非常に微小であるため、その直接検出は未だなされていない。重力波は透過力が強いため、高エネルギー天体現象の中心部や、ブラックホールに関連する現象を直接観測することが可能になる。また、宇宙マイクロ波背景放射より初期の宇宙を直接観測できる可能性もある。このように、重力波の観測は、新しい天文学や宇宙論の手段となることが期待されている。

このような背景のもと、我々は背景重力波の検出に関連した二つの実験を行った。背景重力波とは様々な周波数、振幅の重力波が重なり合ったものである。この背景重力の起源としては、天体起源であるものと初期宇宙起源であるもののが考えられている。この背景重力波が観測できれば、インフレーションや宇宙の構造形成に関連した情報が得られることになる。

## 背景重力波のデータ解析 (実験前半)

石徹白氏の行った、地上検出器であるねじれ型重力波検出器 TOBA を用いた実験における、背景重力波のデータ解析の手法を参考にして、Python の演習を行った。測定は 2009 年 8 月 15 日の夜に 7.5 時間にわたって行われ、そのうちの安定していてノイズの少ない部分の 3.5 時間を解析で用いた。

## ランダムノイズの生成 (実験後半)

後半の実験では衛星搭載検出器である SWIM のデータを用いた。使用したデータは 2010/7/15 17:30:00-21:30:00 (JST) の 240 分間で取られたものである。実験では、このデータのスペクトルと同じようなスペクトルを持つような時系列データを生成するプログラミングを、Python を用いて作成した。これは、実際の重力波検出実験のための解析プログラムに含まれるパラメータを、プログラミングで生成された時系列データを用いてあらかじめ行っておくためである。そうすることで、実際の測定データを使う前にパラメータの調節を行うことができる。

これらの検出器は、より大型で高感度な検出器を将来作るためのプロトタイプと位置づけられており、これらプロトタイプを用いたデータ解析の目的は、将来的な重力波の検出にむけたデータ解析手法を確立することである。

# Chapter 2

# 背景重力波のデータ解析

石徹白氏の行った、地上検出器であるねじれ型重力波検出器 TOBA を用いた実験における、背景重力波のデータ解析の手法を参考にして、Python の演習を行った [3]。測定は 2009 年 8 月 15 日の夜に 7.5 時間にわたって行われ、そのうちの安定していてノイズの少ない部分の 3.5 時間を解析で用いた。TOBA は将来より大型で高感度な検出器を作るためのプロトタイプと位置づけられており、これらプロトタイプを用いたデータ解析の目的は、将来的な重力波の検出にむけたデータ解析手法を確立することである。

## 2.1 ねじれ型重力波検出器

ねじれ型重力波検出器では、逆 T 字型をした棒状の試験質量 (torsion antenna mass,TAM) を用いている。TAM の上端にはネオジウム磁石が取り付けられていて、その上に超伝導体を設置する。磁石と超伝導体の間に働く磁力により、TAM が宙づりにされるようになっている (磁気浮上)。この状態で、TAM は重力波が引き起こす潮汐力を受けて回転する。この回転を計測することで、重力波の信号がえられることになる。

磁気浮上を用いることで、TAM が周囲の装置と接触して摩擦力などを受けることがなくなる。このため、測定を行う際のノイズをおさえることができる [3]。

## 2.2 ノイズスペクトル密度

重力波検出器の出力は重力波の信号とノイズを含んでいる。検出器の入力と出力はスカラー量である一方、重力波はテンソル $h_{ij}$ で記述される。そのため、検出器への入力は次のようにかける。

$$h(t) = D^{ij} h_{ij}(t) \tag{2.1}$$

$D^{ij}$ は検出器の性質で決まるテンソルである。ノイズが無い状態での検出器の出力は、周波数空間で検出器への入力と次のように関係づけられる。

$$\tilde{h}_{\rm out}(f) = T(f)\tilde{h}(f) \tag{2.2}$$

ここで $T(f)$ はシステムの伝達関数である。実際には出力にはノイズも含まれているので、

$$s_{\rm out}(t) = h_{\rm out}(t) + n_{\rm out}(t) \tag{2.3}$$

となる。ノイズ源は複数あるが、それらをまとめてノイズを入力として考え、n(t) を次のように定義する。

$$\tilde{n}(f) = T^{-1}(f)\tilde{n}_{\rm out}(f) \tag{2.4}$$

そうすると検出器の出力は

$$s(t) = h(t) + n(t) \tag{2.5}$$

と書ける。ノイズが定常であると仮定すると、ノイズのフーリエ成分のアンサンブル平均は

$$\langle \tilde{n}^*(f)\tilde{n}(f')\rangle = \delta(f-f')\frac{1}{2}S_{\rm n}(f) \tag{2.6}$$

ここで、$S_{\rm n}(f)$ の次元は $\mathrm{Hz}^{-1}$ である。

また、$\langle n^2(t)\rangle$ は $S_{\rm n}(f)$ の積分で表される。

$$\left\langle n^2(t)\right\rangle = \left\langle n^2(t=0)\right\rangle = \int_{-\infty}^{\infty} df df' \,\langle n^*(f)n(f')\rangle = \frac{1}{2}\int_{-\infty}^{\infty} df S_{\rm n}(f) = \int_0^{\infty} df S_{\rm n}(f) \tag{2.7}$$

$S_{\rm n}(f)$ はノイズスペクトル密度（あるいはノイズパワースペクトル）と言われる。検出器のノイズは $\sqrt{S_{\rm n}(f)}$ で特徴づけられる。これは spectral strain sensitivity と呼ばれ、$\mathrm{Hz}^{-1/2}$ の次元を持つ。

## 2.3 プログラミングの概要

### 2.3.1 エラー信号のスペクトル

測定は 2009 年 8 月 15 日の夜に 7.5 時間にわたって行われ、そのうちの安定していてノイズの少ない部分の 3.5 時間を解析で用いた。データは $T_{\rm eff} = 3.5$hours にわたる volt の次元をもつエラー信号である。プログラミングでは、データを $t_{\rm s} = 102.4$sec ごとに 120 個の部分に分割し、それぞれの部分をフーリエ変換し、スペクトルを計算する。離散フーリエ変換では、データの周期性が仮定されるが、このときの境界の影響をおさえるためにハニング窓を用いた。

### 2.3.2 重力波振幅等価雑音への換算

次に上で求められたスペクトルを、周波数空間で重力波振幅 $h(f)$ に換算する。実験ではエラー信号が PD(Photo Detector) から得られ、サーボフィルタを通してエラー信号をアクチュエータにフィードバックする [3]。フィードバックはまず OL(Optical Lever) を用いた後にレーザー干渉計を用いて行われる。フィードバック信号は DAQ(Data Acquisition System) でローパスフィルターを通して記録される。strain sensitivity、h(f)、は記録された信号 $V_{\rm DAQ}$ から推計される。

$$h(f) = \frac{A(1+G)}{GW}\tilde{V}_{\rm DAQ}(f) \tag{2.8}$$

G=MASF はサーボシステムのオープンループ伝達関数で、M、S、$S_{\rm OL}$、F、A、W はそれぞれ TAM, 干渉計、OL、サーボフィルタ、アクチュエータ、ローパスフィルタの周波数応答である。

### 2.3.3 データの選別

$f_0 = 0.2$Hz 成分を取り出し、$h(f_0)$ の分布が得られる。このうち $h(f_0)$ が大きい 5 %の部分は取り除く。そして 0.2Hz 成分 $h(f_0)$ の平均と分散を計算する。作成したプログラミングは巻末に添付した。

## 2.4 Frequentist の上限

### 2.4.1 Frequentist の解釈

集合 S、その部分集合 A,B を考える。条件確率 $P(A|B)$(B が与えられたもとでの A の確率) は次のように定義される。

$$P(A|B) = \frac{P(A\cap B)}{P(B)} \tag{2.9}$$

Freaquentist の解釈では、A,B は繰り返し行われる実験の結果であり、確率 P(A) は A が起こる頻度として定義される。これは実験を無限回行ったときの極限において考える。この解釈では、あるデータを得る確率が定義できて、そしてある仮定の下で、あるデータを得る条件確率を考えることができる。

### 2.4.2 信頼区間

ある物理量 $x$ の同一の測定を繰り返し行うとする。ある信頼度で、例えば 90 %で、実験結果を $x_1 < x < x_2$ などと表したいとする。未知である $x$ の真の値 $x_t$ は固定されたパラメータである。それぞれの繰り返しでは異なった区間 $[x_1, x_2]$ が得られるが、繰り返し行われる実験のうちの 90 %で $x_t$ がこの区間に含まれるように区間を決めたい。

実験値が真の値 $x_t$ のまわりで、標準偏差 $\sigma$ のガウス分布に従う場合にどのようにしてこの区間を決定するかを考える。

$$P(x|x_t) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left[-\frac{(x-x_t)^2}{2\sigma^2}\right] \tag{2.10}$$

ある一回の実験で $x_0$ という値が得られたとする。このとき区間の下限 (lower limit) は $P(x|x_1)$ の面積の 5 %が $x > x_0$ にあるような $x_1$ とする。もし真の値 $x_t$ がそのようにして定めた $x_1$ よりも小さかったとしたら、測定された $x_0$ という値は、繰り返し行われる実験のうちの 5 %より少ない実験で発生する統計的なゆらぎによるものであることになる。同様に、区間の上限 (upper limit) は $P(x|x_2)$ の面積の 5 %が $x < x_0$ にあるように $x_2 > x_0$ を見つけることで得られる。

## 2.5 0.2Hz 成分の重力波の上限

背景重力波の等方性と非偏光性 (unpolarization) から、等価スペクトル密度 (equivalent spectral density) は重力波振幅等価雑音 $h(f_0)$ と以下のように関係づけられる。

$$\Omega_{\rm eq}(f_0) = \frac{10\pi^2}{3H_0^2} f_0^3 h^2(f_0) \tag{2.11}$$

ここで、$H_0$ はハッブル定数である。$h(f_0)$ の平均から、$\Omega_{\rm eq}(f_0)$ の最適な推定値が $3.3 \times 10^{17}$ と計算された。また、$h(f_0)$ の分散から、$\Omega_{\rm eq}(f_0)$ の統計誤差は $3.2 \times 10^{16}$ と見積もられた。

上限を求めるときは、ノイズを考慮せずに計算する。プログラミングで計算した $h(f_0)$ の平均値を上の式に使って、$\Omega_{\rm eq}(f_0)$ の最適な推定値を得る。重力波に対する Frequentist の上限、$\Omega_{\rm gw}^{\rm UL}$、は次式で定義される信頼係数を使って記述される。

$$C = \int_{\Omega_{\rm eq}(f_0)}^{\infty} P(\Omega_{\rm es}(f_0)|\Omega_{\rm gw}^{\rm UL}(f_0)) d\Omega_{\rm es}(f_0) \tag{2.12}$$

ここで、$P(\Omega_{\rm es}(f_0)|\Omega_{\rm gw}^{\rm UL}(f_0))$ は条件確率分布であり、中心極限定理から、ガウス分布に従う。

$$P(\Omega_{\rm es}(f_0)|\Omega_{\rm gw}^{\rm UL}(f_0)) \propto \exp\left[-\frac{(\Omega_{\rm es}(f_0) - \Omega_{\rm gw}^{\rm UL}(f_0))^2}{2\Omega_{\rm es}(f_0)/N}\right] \tag{2.13}$$

(4) 式から計算した $\Omega_{\rm eq}(f_0)$ の最適な推定値を用い、モンテカルロシミュレーションを実行すると、(5) 式から信頼係数と上限 $\Omega_{\rm gw}^{\rm UL}$ の関係がわかる。

# Chapter 3

# ランダムノイズの生成

後半の実験では衛星搭載検出器である SWIM のデータを用いた。SWIM ではねじれ型重力波検出器が用いられていて、重力波が到来した時の空間のゆがみにより、棒状の試験質量に回転力がかかる効果を観測している。また、試験質量には永久磁石が取り付けられており、周囲に配置したコイルによる磁力で非接触で位置駆動できるようになっている。

使用したデータは 2010/7/15、17:30:00-21:30:00 (JST) の 240 分間で取られたものである。実験では、このデータのスペクトルと同じようなスペクトルを持つ時系列データを生成するプログラミングを、Python を用いて作成した。これは、実際の重力波検出実験のための解析プログラムに含まれるパラメータの調整を、プログラミングで生成された時系列データを用いてあらかじめ行っておくためである。そうすることで、実際の測定データを使う前にパラメータの調整を行うことができる。

作成したプログラミングでは、G. Heinzel が作成した LISO というプログラミングで用いられた手法を参考にした [1]。LISO は、伝達関数の極と零点を入力すると、その伝達関数をスペクトルに持つような時系列を生成する C 言語のプログラムを出力する、MATHMATICA のプログラムである。

## 3.1 ランダムノイズ生成の計算過程

作成したプログラミングの計算過程を述べる [1]。

### 3.1.1 極、零点と伝達関数

まずはじめに、伝達関数 $H(s)(s = 2\pi if)$ の極、零点の周波数とそれらの Q 値から、伝達関数を以下のような多項式の比の形に表現する。

$$H(s) = \frac{P(s)}{Q(s)} = \frac{a_0 + a_1 s + \cdots + a_m s^m}{b_0 + b_1 s + \cdots + b_{n-1} s^{n-1} + s^n}, n > m \tag{3.1}$$

上式の H(s) を以下に列挙する因子の積として計算し、多項式の係数 $a_0, a_1, \cdots, a_m, b_0, b_1, \cdots, b_{n-1}$ が得られる。

・周波数 f における極

$$1/(1 + \frac{s}{2\pi f})$$

・周波数 f における Q 値つきの極

$$1/(1 + \frac{s}{2\pi fQ} + \frac{s^2}{(2\pi f)^2})$$

・周波数 f における零点

$$(1+\frac{s}{2\pi f})$$

・周波数 f における Q 値つきの零点

$$(1+\frac{s}{2\pi fQ}+\frac{s^2}{(2\pi f)^2})$$

・伝達関数を定数倍する因子

### 3.1.2 時系列の計算

伝達関数を式 (1) のように表現した時に得られる多項式の係数 $a_0, a_1, \cdots, a_m, b_0, b_1, \cdots, b_{n-1}$ を用いて、時系列データを計算していく。$\vec{\mathrm{r}}$ を平均 0、分散 1 の標準正規分布に従う長さ n のベクトルとし、同じく長さ n のベクトル $\vec{\mathrm{a}} = (a_0, a_1, \cdots, a_m, 0, \cdots, 0)$ を定義する。このとき次のような手順で、目的の時系列データ $x_i$ を計算することができる。

$$\vec{y_0} = T^{\mathrm{init}} \cdot \vec{r} \tag{3.2}$$

$$\vec{y_i} = E \cdot \vec{y}_{i-1} + T^{\mathrm{prop}} \cdot \vec{r} \tag{3.3}$$

$$x_i = \vec{a} \cdot \vec{y_i} \tag{3.4}$$

以下に上の計算で用いる行列の定義を順に見ていく。

・$T^{\mathrm{init}}$

まず $B = (B_{ij})$ を以下のように構成する。

for $i = 0$ to $n-1$ do
  if $i$ is even then
    $j_0 \leftarrow i/2$
    $s \leftarrow (-1)^{j_0}$
    $j \leftarrow j_0$
    for $k = 0$ to $n$ do
      $B_{ij} \leftarrow s \cdot b_k$
      $k \leftarrow k+2$
      $s \leftarrow -s$
      $j \leftarrow j+1$
  else ($i$ is odd)
    $j_0 \leftarrow (i+1)/2$
    $s \leftarrow (-1)^{(j_0+1)}$
    $j \leftarrow j_0$
    for $k = 1$ to $n$ do
      $B_{ij} \leftarrow s \cdot b_k$
      $k \leftarrow k+2$
      $s \leftarrow -s$
      $j \leftarrow j+1$

次に以下の方程式を解く。

$$B \cdot \vec{m} = \vec{k} \tag{3.5}$$

ここで $\vec{m} = (m_0, m_1, \cdots, m_{n-1})^t, \vec{k} = (0, 0, \cdots, 0, 1/2)^t$ である。次に、

$$C_{ij}^{\mathrm{init}} = \begin{cases} (-1)^{(i-j)/2} m_{(j+i)/2}, & \text{if} \quad (i+j) = \text{even} \\ 0, & \text{else} \end{cases} \tag{3.6}$$

を構成し、Cholesky decomposition、$C^{\text{init}} = T^{\text{init}} \cdot (T^{\text{init}})^t$ によって $T^{\text{init}}$ が計算される。

・$T^{\text{prop}}$

$$A = (A_{ij}) = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \ddots & 0 \\ \vdots & \vdots & 0 & \ddots & 0 \\ 0 & 0 & 0 & \ddots & 1 \\ -b_0 & -b_1 & -b_2 & \cdots & -b_{n-1} \end{bmatrix} \tag{3.7}$$

$$E = \exp(A\Delta t) = (E_{ij}) \quad (i = 0, \ldots, n-1; \quad j = 0, \ldots, n-1). \tag{3.8}$$

とする。ここで $\Delta t = 1/f_{\text{samp}}$ である。まず次の方程式を解く。

$$D \cdot \vec{p} = \vec{q} \tag{3.9}$$

行列 D は以下のように構成される。

for $i = 0$ to $N-1$ do
  for $j = 0$ to $N-1$ do
    $D_{ij} \leftarrow 0$
for $i = 0$ to $n-1$ do
  for $j = i$ to $n-1$ do  for $k = 0$ to $n-1$ do
    $D_{g(i,j),g(j,k)} \leftarrow D_{g(i,j),g(j,k)} + A_{ij}$
    $D_{g(i,j),g(i,k)} \leftarrow D_{g(i,j),g(i,k)} + A_{ij}$

ここで

$$g(i,j) = \begin{cases} \frac{i(i+1)}{2} + j, & \text{if} \quad i \le j \\ \frac{j(j+1)}{2} + i, & \text{else} \end{cases} \tag{3.10}$$

である。また、式 (3.9) の右辺は次のように構成する。

for $i = 0$ to $n-1$ do
  for $j = i$ to $n-1$ do

$$q_{g(i,j)} = \begin{cases} (E_{n-1,n-1})^2 - 1, & \text{if} \quad i = n-1 \\ E_{i,n-1} \cdot E_{j,n-1}, & \text{else} \end{cases} \tag{3.11}$$

方程式 (3.9) を $p$ について解き、

$$C_{ij}^{\text{prop}} = p_{g(i,j)} \tag{3.12}$$

とすれば、Cholesky decomposition、$C^{\text{prop}} = T^{\text{prop}} \cdot (T^{\text{prop}})^t$ によって $T^{\text{prop}}$ が得られる。

## 3.2 実験結果

まず LISO を用いて、SWIM から得られたデータのスペクトルの形を再現するように、伝達関数の極、零点及びそれらの Q 値を調節した。その結果、以下のような極と零点の組み合わせが SWIM のスペクトルを比較的良く再現できることがわかった。
Q 値なしの零点が 5 個で、それらの周波数 [Hz] は、

98.5599781922,
$13.5806817955 \times 10^{-3}$
$14.1633977546 \times 10^{-3}$
$15.0706905484 \times 10^{-3}$
$15.1244329004 \times 10^{-3}$

Q 値付きの極が 3 個で、それらの周波数 [Hz]、Q 値をそれぞれこの順に書くと

$864.0639956069 \times 10^{-6}, 594.3882430466$
$48.0422517921 \times 10^{-3}, 316.5424045767$
$849.5491774735 \times 10^{-6}, 803.3840475413$

Q 値つきの零点と Q 値なしの極は用いなかった。
伝達関数を定数倍する因子は

32.4870824870

となった。
これをもとに作成したプログラミングは巻末に添付した。計算は二回行い、SWIM の測定データのスペクトル、プログラミングから生成された時系列データのフーリエ変換及び $|H(s)|$ を比較したグラフは、それぞれの計算で Figure3.1,3.2 のようになった。いずれも生成した時系列データのフーリエ変換と $|H(s)|$ のグラフが、低周波側と高周波数側で一致していないことがわかる。これは、プログラミングの計算精度が不十分だったために引き起こされている可能性がある。あるいは、時系列のフーリエ変換のふるまいが二つのグラフで異なることから、この不一致は計算過程で乱数を用いたことに付随して生じる、計算の不確定な要素に起因している可能性もある。

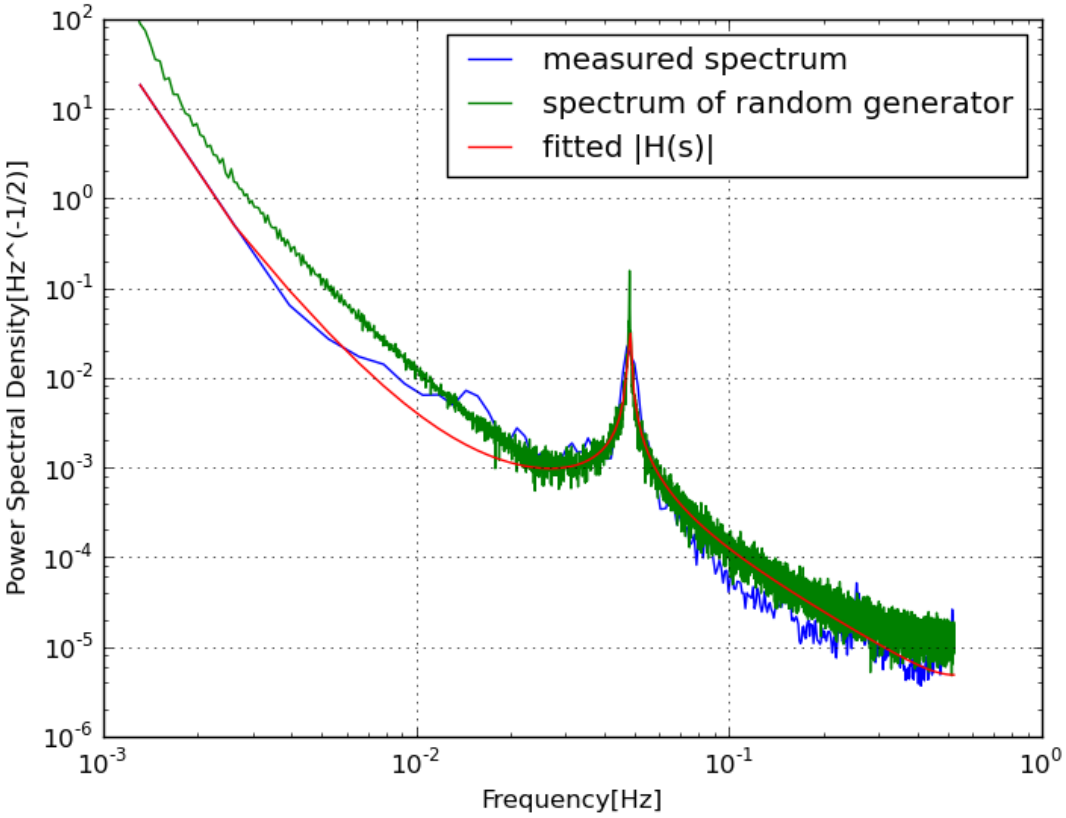
measured spectrum
spectrum of random generator
fitted |H(s)|
Power Spectral Density[Hz^(-1/2)]
Frequency[Hz]
$10^{2}$
$10^{1}$
$10^{0}$
$10^{-1}$
$10^{-2}$
$10^{-3}$
$10^{-4}$
$10^{-5}$
$10^{-6}$
$10^{-3}$
$10^{-2}$
$10^{-1}$
$10^{0}$

Figure 3.1:

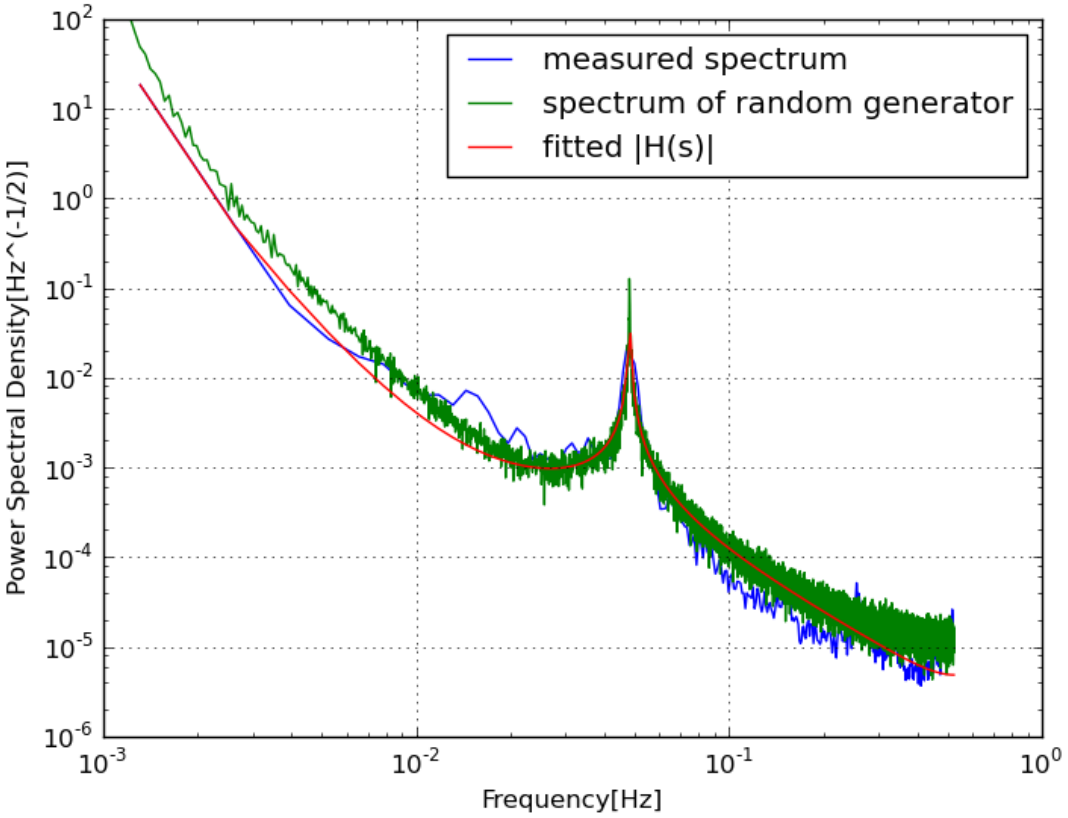
measured spectrum
spectrum of random generator
fitted |H(s)|
Power Spectral Density[Hz^(-1/2)]
Frequency[Hz]
$10^{2}$
$10^{1}$
$10^{0}$
$10^{-1}$
$10^{-2}$
$10^{-3}$
$10^{-4}$
$10^{-5}$
$10^{-6}$
$10^{-3}$
$10^{-2}$
$10^{-1}$
$10^{0}$

Figure 3.2:

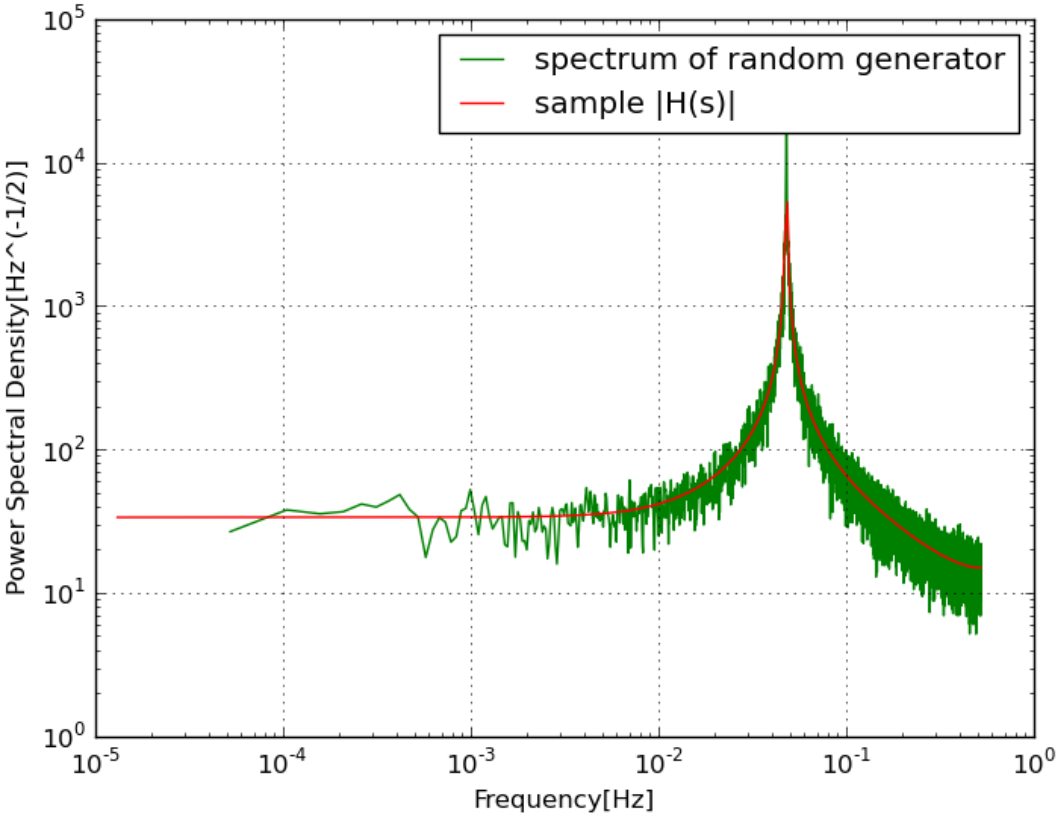


Figure 3.3:

比較のために、スペクトルがより単純な場合の計算結果を以下に記す。
Q 値なしの零点が 1 個で、その周波数 [Hz] は、

$15.1244329004 \times 10^{-3}$

Q 値付きの極が 1 個で、その周波数 [Hz] と Q 値をこの順に書くと

$48.0422517921 \times 10^{-3}, 316.5424045767$

Q 値つきの零点と Q 値なしの極は用いなかった。
伝達関数を定数倍する因子は

32.4870824870

とした。
グラフは Figure 3.3 のようになった。このグラフでは、プログラミングから生成された時系列データのフーリエ変換と $|H(s)|$ がよく一致していることが確認できる。このことは、プログラムに入力する極と零点が多く計算が複雑になる場合に、時系列データのフーリエ変換と $|H(s)|$ が一致しなくなる可能性を示唆する。

## 3.3 ランダムノイズ生成の計算の原理

ランダムノイズを生成するためのプログラミングで行った計算の原理について述べる [2]。

### 3.3.1 ガウシアンプロセスと線形フィルター

x(t) を定常過程とし、E を期待値をとる演算子とする。自己相関関数は次のように定義される。

$$R(t_1, t_2) = E(x(t_1)x(t_2)) \tag{3.13}$$

また、

$$R(\tau) = E(x(t)x(t-\tau)) \tag{3.14}$$

と書いたとき、パワースペクトル密度は、

$$S(\omega) = \int_{-\infty}^{\infty} R(\tau)e^{-i\omega\tau}d\tau \tag{3.15}$$

である。線形システムは次のように線形変換で表現される。

$$x(t) = \int_{-\infty}^{t} g(t-s)w(s)ds \tag{3.16}$$

この変換は、入力 w(s) に対する出力 x(t) を生成する。入力と出力それぞれのパワースペクトル密度 $S_w(\omega), S_x(\omega)$ は次のように関係づけられる。

$$S_x(\omega) = |G(\omega)|^2 S_w(\omega) \tag{3.17}$$

伝達関数 $G(\omega)$ は g(t) のフーリエ変換である。上式で $|G(\omega)|^2 = S(\omega)$ とし、ランダム信号 w を $S_w(\omega) = 1$ となるように選べば、パワースペクトル密度が $S(\omega) = S_x(\omega)$ であるような信号 x を生成できるということが示唆される。条件 $S_w(\omega) = 1$ は、白色ノイズとして知られる理想的なランダム過程 w(t) を定義する。$S(\omega)$ に対して、以下の条件を仮定する。

$$0 \leq S(\omega) < \infty, S(\omega) = S(-\omega), S(\omega) \to 0 \ as \ w \to \pm\infty \tag{3.18}$$

このとき $S(\omega)$ は次のように表現できる [4]。

$$S(\omega) = \left|\frac{P(i\omega)}{Q(i\omega)}\right|^2 \tag{3.19}$$

これより $G(\omega)$ を

$$G(\omega) = \frac{P(i\omega)}{Q(i\omega)} \tag{3.20}$$

と選ぶことができる。あるいは微分演算子 D=d/dt を用いれば

$$x(t) = \frac{P(D)}{Q(D)}w(t) \tag{3.21}$$

となる。つまり $x(t)$ を得るには、まず以下の微分方程式を解く。

$$Q(D)\phi(t) = w(t) \tag{3.22}$$

このとき必要な信号は次式から得られる。

$$x(t) = P(D)\phi(t) \tag{3.23}$$

### 3.3.2 Cholesky decomposition

$P = (p_{ij})$ を正定値の実対称行列とする。このとき下三角行列 T を用いて、以下のように行列 P を分解することができる [6]。

$$P = TT^* \tag{3.24}$$

### 3.3.3 ガウシアンプロセスのシミュレーション

モーメント行列 M が正定値行列であり、n 次元多変数ガウス分布の独立なサンプル $z^{(0)}, z^{(1)}, z^{(2)}, \cdots$ を生成したいとする。それは平均値が 0、分散が 1 であるようなガウス分布に従う一次元の数列 $w_n$ を用いて行うことができる。まず以下のベクトルを定義する。

$$w^{(0)} = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}, w^{(1)} = \begin{bmatrix} w_{n+1} \\ \vdots \\ w_{2n} \end{bmatrix}, w^{(3)} = \begin{bmatrix} w_{2n+1} \\ \vdots \\ w_{2n} \end{bmatrix}, \cdots \tag{3.25}$$

これらはモーメント行列が恒等行列であるような n 次元のガウス分布をしたベクトルである。次に、行列 M が Cholesky decomposition、$M = TT^*$ によって分解されたとし、

$$z^{(i)} = Tw^{(i)} \tag{3.26}$$

を定義する。これらが求めたかったベクトルである。なぜならこのモーメント行列が以下のように M に一致しているからである。

$$[E(z_\alpha z_\beta)] = E(zz^*) = E(Tww^*T^*) = TT^* = M \tag{3.27}$$

次に、ある時間間隔 $\Delta t$ にたいして、パワースペクトル $S(\omega)$ を持つような時系列データ x(t) を $t = 0, \Delta t, 2\Delta t$ に対して計算したいとする。式 (3.22) を以下の形に書く。

$$\phi^n(t) + a_1\phi^{n-1}(t) + \cdots + a_n\phi(t) = w(t) \tag{3.28}$$

w(t) は白色ノイズである。$x(t) = P(D)\phi(t)$ を計算するために、状態ベクトルを以下のように定義する。

$$z(t) = \begin{bmatrix} \phi(t) \\ \phi'(t) \\ \vdots \\ \phi^{n-1}(t) \end{bmatrix} \tag{3.29}$$

z は以下の微分方程式を満たす。

$$\frac{dz(t)}{dt} = Az(t) + f(t) \tag{3.30}$$

$$A = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ a_n & -a_{n-1} & -a_{n-2} & \cdots & -a_1 \end{bmatrix}, f(t) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ w(t) \end{bmatrix} \tag{3.31}$$

まずベクトル z(0) を計算する。モーメント行列は

$$M = E(z(0)z^*(0)) = E(z(t)z^*(t)) = [E(z_i z_j)] = [E\phi^{(i-1)}\phi^{(j-1)}] \tag{3.32}$$

行列 M の成分は [5]

$$m_{ij} = \begin{cases} 0, & i+j = \text{odd} \\ (-1)^{(j-i)/2} m_{(j+i)/2}, & i+j = \text{even} \end{cases} \tag{3.33}$$

であり、$m_0, m_1, \cdots, m_{n-1}$ は次の方程式を解くことで得られる。

$$(-1)^k \sum_{k/2 \le q \le (n+k)/2} (-1)^q a_{n-2q+k} m_q = \begin{cases} 0, & k = 0, \cdots, n-2 \\ \frac{1}{2}, & k = n-1 \end{cases} \tag{3.34}$$

ここで $a_0 = 1$ である。行列 M を計算できたら Cholesky decomposition、$M = TT^*$ を用いて、

$$z(0) = z^{(0)} = Tw^{(0)} \tag{3.35}$$

と z(0) が計算できる (式 (3.26))。次に、z(t) が計算できたときに $z(t + \Delta t)$ をどのように計算するかを考える。式 (3.30) から、

$$z(t + \Delta t) = e^{A\Delta t} z(t) + r \tag{3.36}$$

ここで、

$$r = \int_0^{\Delta t} e^{(\Delta t - s)A} f(t+s) ds \tag{3.37}$$

である。r を計算するために、モーメント行列 $M_r$ を考える。

$$M_r = E(rr^*) = E \int_0^{\Delta t} \int_0^{\Delta t} e^{(\Delta t - s_1)A} f(t+s_1) f^*(t+s_2) e^{(\Delta t - s_2)A^*} ds_1 ds_2 = \int_0^{\Delta t} e^{sA} C e^{sA^*} ds \tag{3.38}$$

上式の被積分関数を J(s) とおくと、

$$\frac{d}{ds} J(s) = AJ(s) + J(s)A^* \tag{3.39}$$

これを $0 < s < \Delta t$ で積分すると

$$e^{A\Delta t} C [e^{A\Delta t}] - C = AM_r + M_r A^* \tag{3.40}$$

ここで、

$$\exp A\Delta t = [e_{ij}], M_r = [\mu_{ij}] \tag{3.41}$$

と定義すると、式 (3.40) は

$$\sum_{k=1}^{n} (a_{ik}\mu_{kj} + a_{jk}\mu_i k) = \begin{cases} e_{in}e_{jn}, & \text{if} \quad i < n \quad \text{or} \quad j < n \\ e_{nn}^2 - 1, & \text{if} \quad i = n \quad \text{and} \quad j = n \end{cases} \tag{3.42}$$

この式から $M_r$ を計算し、Cholesky decomposition、$M_r = T_r T_r^*$ により $T_r$ を得る。$t + \Delta t = \nu \Delta t$ とし、$w^{(\nu)}$ を式 (3.25) で定義されたベクトルの一つだとする。すると

$$r = T_r w^{(\nu)}, \quad z(t + \Delta t) = e^{A\Delta t} z(t) + r \tag{3.43}$$

となる。z(0) はすでに計算されているので、この式を繰り返し用いて $t = 0, \Delta t, 2\Delta t, \cdots$ に対して z(t) が計算されることになる。式 (3.19) において、多項式 P を次のように表したとする。

$$P(\zeta) = b_0 \zeta^m + b_1 \zeta^{m-1} + \cdots + b_m \tag{3.44}$$

パワースペクトル密度 $S(\omega)$ を持つ、求めたい x(t) は次式から計算される。

$$x(t) = b_0 z_{m+1}(t) + b_1 z_m(t) + \cdots + b_m z_1(t), \quad t = 0, \Delta t, 2\Delta t, \cdots, \tag{3.45}$$

ここで $z_1(t), \cdots, z_n(t)$ はベクトル z(t) の成分であり、上式は式 (3.23)$x(t) = P(D)\phi(t)$ と等価である。

# 背景重力波の解析で作成したプログラミング

```
from numpy import *
import pylab
import scipy.fftpack
import matplotlib.pyplot as plt

def filter_design20090119(fr):
f0=5.4*(10**-3)
gamma=7.46*(10**-3)
w=2*pi*fr
c1=10*(10**-6)
c2=470*(10**-12)
r1=1000
r2=10*(10**3)
r3=10*(10**3)
z1=(r1+1.0/(1j*w*c1))*r2/((r1+1.0/(1j*w*c1))+r2)
z2=(r3*1.0/(1j*w*c2))/(r3+1.0/(1j*w*c2))
z=z2/z1
A=-1.0*(10**3)/(-w**2+1j*w*gamma+((2*pi*f0)**2))
amp_s=abs(z)
pha_s=angle(z,deg=True)
z=z*A
amp_l=abs(z)
pha_l=angle(z,deg=True)
return(fr,amp_s,pha_s,amp_l,pha_l)

def spe(data,nfft,samp):
    l=len(data)
    av=l//nfft
    window=hanning(nfft)
    window=matrix([window,window])
    zeron=zeros(av,dtype=int16)
    window=window[zeron]
    Be=8.0/3.0
    y=reshape(data[0:nfft*av],(av,nfft))
    y=multiply(window,y)
    ys=scipy.fftpack.fft(y,nfft)
    ys=ys.T
    ls=abs(ys)*sqrt(1.0/nfft/samp*2*Be)
    spe=ls[0:nfft/2]
    return spe

def f(samp,nfft):
    f=samp*arange(nfft/2.)/nfft
    return f[:,newaxis]

test=loadtxt('data_samp10cut1.dat')
mea=test.mean()
data=test-mea
samp=10
```

```
cal=4.0/(4.0*pi*0.050/(1064e-9))*1.0/(0.097*2)
data=data[600:2.6e+5]
(spe,fr)=(spe(data,2**10,samp),f(samp,2**10))
(fr,amp_s,pha_s,amp_l,pha_l)=filter_design20090119(fr)
A=abs(spe[20,:])*cal*(1+(amp_l[20])/28.9)
B=A[110:230]
out=B<0.6e-8
C=B[out]
D=C**2
(y,bin,pat)=plt.hist(D,bins=16)
x=(bin[:-1]+bin[1:])/2

m=D.mean()
s=D.std()
print 'm=',m
print 's=',s
H=3.24*(10**-18)*0.7
print 'H=',H

le=len(D)
print 'le=',le
Omega_gw=10*(pi**2)/(3*(H**2))*(0.2**3)*m
err_omega_gw=10*(pi**2)/(3*(H**2))*(0.2**3)*s/sqrt(le)
print 'Omega_gw=',Omega_gw
print 'err_omega_gw=',err_omega_gw

plt.plot(x,y)
plt.plot(0.5*(x[1:]+x[:-1]),exp(-1.8*(10**17)*0.5*(x[1:]+x[:-1])+3.8))
plt.xlabel('Output')
plt.ylabel('Number')
plt.title('prog.py')
plt.grid(True)
plt.show()
```

# ランダムノイズ生成のプログラミング

```
from numpy import *
import scipy.linalg
import matplotlib.pyplot as plt

def A(b):
n=len(b)-1
a=zeros((n,n))
i=0
j=0
while i<n-1:
a[i,i+1]=1.0
i=i+1
while j<=n-1:
a[n-1,j]=-b[j]
j=j+1
print 'A=',a
return a

def E(A,dt,n):
E=scipy.linalg.expm3(dt*A,n)
print 'E=',E
return E

def B(b):
n=len(b)-1
bi=zeros((n,n))
print 'b=',b
i=0
while i<=n-1:
if i%2==0:
k=0
jo=i/2
s=(-1)**jo
j=jo
while k<=n:
bi[i,j]=s*b[k]
k=k+2
s=-s
j=j+1
else:
k=1
j1=(i+1)/2
s=(-1)**(j1+1)
j=j1
while k<=n:
bi[i,j]=s*b[k]
k=k+2
s=-s
j=j+1
```

```
i=i+1
print 'B=',bi
return bi

def C_init(m):
n=len(m)
i=0
c=zeros((n,n))
while i<=n-1:
j=0
while j<=n-1:
if (i+j)%2==0:
c[i,j]=((-1)**abs((i-j)/2))*m[(i+j)/2]
else:
c[i,j]=0.0
j=j+1
i=i+1
print 'C_init=',c
return c

def g(i,j):
if i>=j:
g=i*(i+1)/2+j
else:
g=j*(j+1)/2+i
return g

def D(A):
n=size(A,1)
print 'n=',n
N=n*(n+1)/2
d=zeros((N,N))
i=0
j=0
while i<=n-1:
j=i
while j<=n-1:
k=0
while k<=n-1:
d[g(i,j),g(j,k)]=d[g(i,j),g(j,k)]+A[i,k]
d[g(i,j),g(i,k)]=d[g(i,j),g(i,k)]+A[j,k]
k=k+1
j=j+1
i=i+1
print 'D=',d
return d

def q(E):
n=size(E,1)
print 'n=',n
N=n*(n+1)/2
```

```
q=zeros(N)[:,newaxis]
i=0
j=0
while i<=n-1:
j=i
while j<=n-1:
if i==n-1:
q[g(i,j)]=(E[n-1,n-1])**2-1
else:
q[g(i,j)]=E[i,n-1]*E[j,n-1]
j=j+1
i=i+1
print 'q=',q
return q

def C_prop(p,n):
cp=zeros((n,n))
i=0
while i<=n-1:
j=0
while j<=n-1:
print(i)
print(j)
cp[i,j]=p[g(i,j)]
j=j+1
i=i+1
print 'C_prop=',cp
return cp

def cholesky(A):
    return linalg.cholesky(A)

def gauss(n):
    return random.normal(0, 1, n)

def solution(D,q):
    return linalg.solve(D,q)

def k_vec(b):
n=len(b)-1
k=zeros(n)[:,newaxis]
k[n-1]=1./2.
print k
return k

def a_vec(a,b):
m=len(a)
n=len(b)-1
vec_a=zeros(n)
i=0
while i<=m-1:
```

```
vec_a[i]=a[i]
i=i+1
v=vec_a
return v

def time_series(n,n_b,T_init,a_vec,E,T_prop):    # n is the number of required samples
i=0
x=[]
while i<=n-1:
r=gauss(n_b)
if i==0:
y=dot(T_init,r)
x1=inner(a_vec,y)
x2=append(x,x1)
else:
y1=dot(E,y)
y2=dot(T_prop,r)
y=y1+y2
x1=inner(a_vec,y)
x2=append(x2,x1)
i=i+1
return x2

import scipy
import math
from numpy.lib import scimath
import numpy
from numpy import *

def inverse(a):
    b=a[len(a)-1]
    for i in range(len(a)-1):
        b=hstack((b,a[len(a)-2-i]))
    return b

u=array([1,2,4,7])
print 'inverse(u)=',inverse(u)

def transfer3(zero,pole,zerowithQ,Qofzero,polewithQ,Qofpole):
c=poly(-2.0*pi*zero)
     print 'c=',c
e=pi*zerowithQ/Qofzero*(-1+scimath.sqrt(1-4*Qofzero**2))
print 'e=',e
f=pi*zerowithQ/Qofzero*(-1-scimath.sqrt(1-4*Qofzero**2))
d=poly(hstack((e,f)))
print 'f=',f
print 'd=',d
     a1=polymul(c,d)
print 'a1=',a1
lp=len(pole)
lz=len(zero)
```

```
if lp==0:
dp=1.0
else:
dp=linalg.det(diag(pole))
if lz==0:
dz=1.0
else:
dz=linalg.det(diag(zero))
     a2=a1*((2.0*pi)**(lp-lz))*dp/dz
print 'a2=',a2
lpQ=len(polewithQ)
lzQ=len(zerowithQ)
if lpQ==0:
dpQ=1.0
else:
dpQ=linalg.det(diag(polewithQ))
if lzQ==0:
dzQ=1.0
else:
dzQ=linalg.det(diag(zerowithQ))
     a3=a2*((2.0*pi)**(2*lpQ-2*lzQ))*((dpQ/dzQ)**2)
    print inverse(a3)
return inverse(a3)

def transfer4(zero,pole,zerowithQ,Qofzero,polewithQ,Qofpole):
g=poly(-2.0*pi*pole)
h=pi*polewithQ/Qofpole*(-1+scimath.sqrt(1-4*Qofpole**2))
i=pi*polewithQ/Qofpole*(-1-scimath.sqrt(1-4*Qofpole**2))
       p=poly(hstack((h,i)))
b1=polymul(g,p)
print 'g=',g
print 'h=',h
print 'i=',i
print 'p=',p
      print inverse(b1)
return inverse(b1)

from numpy import *
import scipy.linalg
import matplotlib.pyplot as plt

zero=array([98.5599781922,13.5806817955*(10**(-3)),14.1633977546*(10**(-3)),15.07069054
pole=array([])
zerowithQ=array([])
Qofzero=array([])
polewithQ=array([864.0639956069*(10**(-6)),48.0422517921*(10**(-3)),849.5491774735*(10*
Qofpole=array([594.3882430466,316.5424045767,803.3840475413])
factor=32.4870824870

a=(transfer3(zero,pole,zerowithQ,Qofzero,polewithQ,Qofpole))*factor
b=transfer4(zero,pole,zerowithQ,Qofzero,polewithQ,Qofpole)
```

```
X=loadtxt('swim_noise100617_per_rHz2.txt')
fre=X[:,0]
out=X[:,1]
f_samp=2*fre[-1]
A=A(b)
B=B(b)
k=k_vec(b)
dt=1./f_samp
E=E(A,dt,1000)
print size(E,1)
m=solution(B,k)
print 'm=',m
C_init=C_init(m)
D=D(A)
q=q(E)
p=solution(D,q)
n_b=len(b)-1
C_prop=C_prop(p,n_b)
T_init=cholesky(C_init)
print 'T_init=',T_init
T_prop=cholesky(C_prop)
print 'T_prop=',T_prop
a_vec=a_vec(a,b)
print 'a_vec=',a_vec
data=time_series(10**5,n_b,T_init,a_vec,E,T_prop)/sqrt(2)

import numpy as np
#{{{ periodgram(x,fs,window=None):

def periodgram(x,fs,window=None):
    """
    Compute a one sided periodgram
    """
    N=np.size(x)  #Number of points
    if not window is None:
        x=x*window
        windFact=(window**2).sum()/N
    else:
        windFact=1
    p=np.fft.fft(x)  #DFFT
    p=1.0/(fs*N)*np.abs(p)**2/windFact  #Periodgram
    p=2*p[:N/2] #Extract the first half and multiply with 2 to conserve the energy.
    #Frequency array
    f=np.fft.fftfreq(N,1.0/fs)
    f=f[:N/2]
    return (f,p)
#}}}
#{{{ psd(x,fs,navg=5,overlap=0,window=None):

def psd(x,fs,navg=5,overlap=0,window=None, power=False):
```

```
    """psd(x,fs,navg=5,overlap=0,window=None)
    - Calculates power spectrum density of a given time series.
    x: One dimensional array representing a time series.
    fs: The sampling frequency of the time series.
    navg: The number of averages to be performed. x is divided into
          navg parts and a periodgram is caluculated for each part.
          Then the periodgrams are averaged to obtain a smooth spectrum.
    overlap: A number between 0 and 1. When overlap is non-zero,
             neighboring parts are overlapped by this fraction.
    window: window function. numpy.hanning and numpy.hamming are popular ones.
    power: If true, the output is a power spectrum (in the unit of power).
           If false, the output is a power spectrum density (in the unit of amplitude).
    """
    # Determine the number of samples in each segment
    N=np.floor(np.size(x)/(1.0+(1.0-overlap)*(navg-1)));
    # Make N even
    if np.mod(N,2):
        N=N-1
    # Increment between the first samples of adjacent segments
    Ninc=np.floor(N*(1-overlap));
    #Create window
    if not window is None:
        windata=window(N)
    else:
        windata=None
    #Initialization
    p=np.zeros(N/2)
    f=np.zeros(N/2)
    for ii in range(navg):
        # Extract samples to be analyzed
        data=x[ii*Ninc:(ii)*Ninc+N];
        # Calculate periodogram
        (f,p1)=periodgram(data,fs,windata)
        p=p+p1
    if power:
        p=p/navg
    else:
        p=np.sqrt(p/navg)
    return (f,p)
#}}}

import numpy as np
from psd import *
import scipy.stats as st
import matplotlib.pyplot as plt

#Calculate PSD
(ff,pp)=psd(data,f_samp,5,window=np.hanning)
#Plot
plt.loglog(fre,out)
plt.loglog(ff,pp)
```

```
from numpy import *
import scipy.linalg
import matplotlib.pyplot as plt
plt.xlabel('Frequency[Hz]')
plt.ylabel('Amplitude')
plt.title('testdata')
plt.grid(True)

nume=poly1d(inverse(transfer3(zero,pole,zerowithQ,Qofzero,polewithQ,Qofpole))*factor)
deno=poly1d(inverse(transfer4(zero,pole,zerowithQ,Qofzero,polewithQ,Qofpole)))
f1=1.311*(10**(-3))
f2=523.2*(10**(-3))
freq=linspace(f1,f2,num=400)
fr=freq*2.0*pi*1j
nu1=nume(fr)
de1=deno(fr)
y1=nu1/de1
z1=(abs(y1))**2
i=1
zz2=0
while i<=25:
f2=(i*f_samp-freq)*2.0*pi*1j
f3=(i*f_samp+freq)*2.0*pi*1j
nu2=nume(f2)
nu3=nume(f3)
de2=deno(f2)
de3=deno(f3)
y2=nu2/de2
y3=nu3/de3
z2=(abs(y2))**2+(abs(y3))**2
zz2=zz2+z2
i=i+1
az=sqrt(z1+zz2)
plt.loglog(freq,az)
plt.xlim(10**(-3),1.0)
plt.ylim(10**(-6),10**2)
plt.show()
```

# Bibliography


[1] G.Heinzel: Generation or Random time series with prescribed spectra, S2-AEI-TN-3034,1.0,2007/06/18

[2] Franklin, joel N.: Numerical simulation of stationary and non-stationary gaussian random processes,
SIAM review, Volume 7, Issue 1, page 68-80, 1965.

[3] Koji Ishidoshiro: Search for low-frequency gravitaional waves using a sperconducting magnetically-levitated torsion antenna.

[4] W. B. Davenport, Jr., And W. L. Root. :An Introduction to the Theory of Random Signals and Noise, McGraw-Hill, New York,1958.

[5] J. N. Franklin, The covariance matrix of a continuous autoregressive vector time-series,Ann. Math. Statist., 34 (1963), pp. 1259-1264

[6] F. R. Gantmacher, The Theory of Matrices, vols. 1 and 2, transl., K. A. Hirsch, Chelsea, New York, 1959