特別実験　レポート

理学部物理学科４年　05-091561
三上　諒

平成 23 年 2 月 3 日

目 次

第1章　はじめに

我々は、特別実験において、背景重力波の検出を最終的な目標として、研究を行った。
背景重力波とは、インフレーションや宇宙の相転移といった宇宙初期の過程で放出される重力波や、宇宙の階層構造が形成される際の超大質量ブラックホールの合体により放出される重力波など、様々な過程で生じる重力波のランダムな重ね合わせで生じるものである ([2])。背景重力波を検出し、その特徴（周波数特性など）を知ることは、単なる重力波実験にとどまらず、宇宙論や、天文学に新たな知見を与えることに貢献する、と言えるだろう。
この背景重力波を検出するための検出器として、地上検出器の TOBA（本郷と、京都大の 2 か所にある。）と、衛星搭載検出器である SWIM がある。これらの検出器で背景重力波を本当に検出するのは難しいが、将来的に、より大型で、高感度な検出器を作るための、「プロトタイプ」の役割を果たす検出器である。そして、これらの検出器を用いたデータ解析は、将来的な背景重力波の検出に向け、データ解析の手法を確立する、という役割を果たす。
以上のような動機、目標のもと、我々は、以下のようなことを行った。

実験前半

実験前半では、論文 [1]、[2] に基づき、プログラミング言語 Python を用いた重力波データ解析の基礎について学んだ。具体的には、TOBA を用いて、2009 年 8 月 15 日に 7.5 時間にわたって取得された重力波データから、重力波エネルギー密度を見積もるまでの過程について、論文の著者である石徹白氏より頂いた MATLAB のプログラムを用いながら理解するとともに、そのプログラムを、Python で実行できるように書き換えた。

実験後半

実験後半では、論文 [3]、[4] に基づき、与えられたパワースペクトルに対し、そのスペクトルを近似的に再現するような時系列 (フェイクノイズ) を生成するプログラムを作る、ということを行った。この過程は、例えば、未完成のある検出器に対し、その検出器から得られる（と想定される）データを

解析するアルゴリズムが先に完成した場合に、そのアルゴリズムをチェックするために必要となる過程である。

　以下で、行ったことについて詳しく述べていく。なお、論理の展開を分かりやすくするために、作成したプログラムや、その出力結果等は、「付録」にまとめて示した。

第2章　実験前半

—Python による重力波データ解析の基礎—

2.1　ここでの課題

論文 [1]、[2] において、TOBA を用いて、2009 年 8 月 15 日に 7.5 時間にわたって取得された重力波データから、0.2Hz における背景重力波 (GWB) のエネルギー密度 $\Omega_{\rm gw}$ の上限を求める際に、石徹白氏より頂いたプログラムの計算、あるいは、プログラムにより出力された数値がどのように使われているか、述べる。また、頂いた MATLAB のプログラムを、Python で実行できるように書き換える。

2.2　TOBA について

TOBA は、”TOrsion-Bar Antenna” の略で、回転センサーと、試験質量からなる。試験質量は逆 T 字型をしており、試験質量の最上部にある Nd の磁石と、試験質量の上に置かれた超伝導体の間に働く磁力により支えられている (磁気浮上)。このため、試験質量は、回転の自由度を持つことができ、重力波がやってくると、それにより生じる潮汐力により回転する。したがって、試験質量の角度変動をとらえることで、重力波をとらえることができる。また、超伝導体による磁気浮上の技術を用いることにより、回転自由度内で復元力や摩擦力を感じず、また、熱雑音を抑えることができるという利点がある。

2.3　課題に対する答えと、解析結果

2.3.1　頂いたプログラム

石徹白氏より頂いたプログラムは、以下の 3 つである。

- main20100715.m
- filter_design20090119.m
- fft_data2.m

プログラムの内容は、付録 A に掲載した。

2.3.2　プログラムの内容と、その計算結果

3 つのプログラムは、ねじれアンテナ（TOBA）を用いた、0.2Hz における、GWB（背景重力波）の上限を求めるためのプログラムである。
プログラムの中に現れている、「data_samp10cut1.dat」は、Volt の次元をもつエラー信号である。我々は、これを、重力波振幅に換算する必要がある。今、その換算を周波数空間で行うため、fft_data2.m のプログラムにより、信号のフーリエ変換を行った。
filter_design20090119 は、サーボフィルターの伝達関数を計算している。実際のフィルターは、抵抗、コンデンサー、オペアンプからなるアナログ回路であり、プログラム中の c1、c2、r1、r2、r3 はそれぞれ、使用したコンデンサーと抵抗の値を意味している。
また、amp_s、pha_s、は、純粋なサーボフィルターのゲインと位相であり、amp_l、pha_l は、アクチュエータ効率などが含まれたゲインと位相を表している。
重力波振幅等価雑音 (GW strain-equivalent noise level)$h(f)$ と、記録された信号 V_{DAQ} の間には、

$$h(f) = \frac{A(1+G)}{GW}\tilde{V}_{\mathrm{DAQ}}(f) \tag{2.1}$$

という関係がある。
ただし、
G=MASF:openloop transfer function of the servo system
M:TAM の周波数応答 (frequency response)
A:アクチュエータ (actuator) の周波数応答
S:干渉計 (interferometer) の周波数応答
F:サーボフィルター (servo filter) の周波数応答
W:ローパスフィルター (low pass filter) の周波数応答
である。
(1) 式に基づき、プログラム main20100715.m において、A=abs(spe(21,:))*cal.*(1+amp_l(21)/28.9) の cal.*(1+amp_l(21)/28.9) の部分で、エラー信号の 0.2Hz 成分に関して、オープンループ補正をして、重力波振幅等価雑音 $h(f)$ に変換している。
背景重力波（GWB）の等方性 (isotropy) と、偏極していないこと (unpolarization) を仮定すると、GW energy equivalent spectral density$\Omega_{\mathrm{eq}}(f_0)$ と、重力波振幅等価雑音 $h(f_0)$ の間には、

$$\Omega_{\mathrm{eq}}(f_0) = \frac{10\pi^2}{3H_0^2} f_0^3 h^2(f_0) \tag{2.2}$$

という関係がある。
プログラム main20100715.m で出力される数値 m により、$h^2(f)$ の 0.2Hz 成分の平均が求められるが、それを (2) 式に代入することによって求めた、$\Omega_{\mathrm{eq}}(f_0)$

の最適な推定値 (optimal estimator) は、3.3×10^{17} となる。
また、プログラム main20100715.m の数値 s により、$h^2(f)$ の 0.2Hz 成分の標準偏差が求められ、それを (2) 式に代入して、$\Omega_{\rm eq}(f_0)$ の標準 (統計) 誤差 (standard(statistical) error) は、3.2×10^{16} となる。
GWB の、Frequentist の上限 $\Omega_{\rm gw}^{\rm UL}(f_0)$ は、信頼度 C を用いて、

$$C = \int_{\Omega_{\rm eq}(f_0)}^{\infty} P(\Omega_{\rm es}(f_0)|\Omega_{\rm gw}^{\rm UL}(f_0))d\Omega_{\rm es}(f_0) \tag{2.3}$$

で表される。
$P(\Omega_{\rm es}(f_0)|\Omega_{\rm gw}^{\rm UL}(f_0))$ は、条件付き確率分布 (conditional probability distribution) で、中心極限定理より、ガウス分布に従う。

$$P(\Omega_{\rm es}(f_0)|\Omega_{\rm gw}^{\rm UL}(f_0)) \propto \exp\left\{-\frac{(\Omega_{\rm es}(f_0)-\Omega_{\rm gw}^{\rm UL}(f_0))^2}{2\Omega_{\rm es}(f_0)/N}\right\} \tag{2.4}$$

$\Omega_{\rm es}(f_0)$ は、ノイズのない N 個のサンプルを用いて見積もられる、$\Omega_{\rm gw}(f_0)$ の推定値 (the estimator of $\Omega_{\rm gw}(f_0)$ using N sample without noise) である。
今、$\Omega_{\rm eq}(f_0)$ が分かっているから、C を決めれば、$\Omega_{\rm gw}^{\rm UL}(f_0)$ を求めることができる。

2.3.3 MATLAB のプログラムを Python のプログラムに書き換える

頂いた三つのプログラムを、Python で実行できるようにプログラムを書き換えた。その結果は、付録 A に掲載している。(なお、三つのプログラムを一つにまとめた。)

第3章　実験後半

―与えられたパワースペクトルから、フェイクノイズを作る手法の実践―

3.1　ここで行ったこと

論文 [3]、[4] に基づき、与えられたスペクトルに対し、まず、LISO というプログラムを用いて、そのスペクトルを多項式の比の関数として近似する。次に、その近似した関数をもとに、もとのスペクトルを近似的に再現するような時系列 (フェイクノイズ) を生成するプログラムを作成する。最後に、もともとの与えられたスペクトル、LISO により近似した関数、生成した時系列をフーリエ変換したスペクトル、の 3 者を比較する。なお、ここでの「与えられたスペクトル」とは、SWIM が 2010 年 7 月 15 日の 17:30:00〜21:30:00 (JST) の 240 分間に取得したデータのスペクトルを用いた。

3.2　SWIM について

2009 年 1 月 23 日、宇宙航空研究開発機構 (JAXA) による温室効果ガス観測技術衛星いぶき (GOSAT) が、H-IIA ロケットによって種子島宇宙センターから打ち上げられた。この際、ロケットには、余剰打ち上げ能力を利用した相乗り副衛星が 7 機搭載されていたが、その中の 1 機、SDS-1 の中に搭載された装置が、SWIM である。SWIM は、宇宙用計算機 SpaceCube2 と、SWIM$_{\mu\nu}$ により構成されており、このうち、SWIM$_{\mu\nu}$ は、重力波センサモジュールと、そのセンサを制御、管理する電子回路を集積した基板からなっている。センサモジュールは、ねじれ型と呼ばれるタイプの重力波検出器となっており、重力波が到来したときの空間のゆがみにより、センサモジュール内部の棒状の試験質量に回転力がかかる効果を観測するものである。この装置では、重力波の回転力効果を、制御信号から取り出す、という仕組みになっている。

3.3　原理

線形系では、入力のパワースペクトル密度 $S_x(\omega)$ と、出力のパワースペクトル密度 $S_y(\omega)$ の間に、

$$S_y(\omega) = |H(\omega)|^2 S_x(\omega) \tag{3.1}$$

の関係がある。ここで、$H(\omega)$ は、伝達関数である。我々は、これから、上式において、出力のパワースペクトル密度が $S(\omega)$ となるような、時系列を作る方法について考えていく。
我々は今、上式で、$S(\omega) = |H(\omega)|^2$ を満たすように伝達関数 $H(\omega)$ を選び、(入力のパワースペクトル密度) $= 1$ となるように、入力を選ぶ（白色雑音)。

パワースペクトル密度 $S(\omega)$ は、

$$0 \leq S(\omega) < \infty, \quad S(\omega) = S(-\omega), \quad S(\omega) \to 0 \quad (\omega \to \pm\infty) \tag{3.2}$$

の条件を満たすとき、

$$S(\omega) = \left| \frac{P(\mathrm{i}\omega)}{Q(\mathrm{i}\omega)} \right|^2 \tag{3.3}$$

の形で表せる。（ここで、ω は実数。$P(z)$ と $Q(z)$ は実係数を持つ z の多項式であり、また、多項式 P の次数は、多項式 Q の次数より小さい。さらに、$Q(z) = 0$ の解は、$\mathrm{Re}[z] < 0$ を満たす。)
今、この条件を満たし、パワースペクトル $S(\omega)$ が、ω についての多項式の比の関数で表せると仮定する。すると、

$$H(\omega) = \frac{P(\mathrm{i}\omega)}{Q(\mathrm{i}\omega)} \tag{3.4}$$

と表せる。したがって、今、$s \equiv \mathrm{i}\omega = 2\pi \mathrm{i} f$ とし、

$$H(s) = \frac{P(s)}{Q(s)} = \frac{a_0 + a_1 s + \cdots + a_m s^m}{b_0 + b_1 s + \cdots + b_{n-1} s^{n-1} + s^n} \qquad (n > m) \tag{3.5}$$

と表せる。あるいは、$D \equiv \frac{\mathrm{d}}{\mathrm{d}t}$(時間微分演算子) として、

$$x(t) = \frac{P(D)}{Q(D)} = w(t) \tag{3.6}$$

つまり、定常状態の $\phi(t)$ に対して、$Q(D)\phi(t) = w(t)$ を解き、そののち、$P(D)\phi(t) = x(t)$ を解くことにより、欲しい時系列 $x(t)$ を得ることができる。
そこで、まず、$w(t)$ を白色雑音として、

$$b_0\phi(t) + b_1\dot{\phi}(t) + \cdots + \phi^{(n)}(t) = w(t) \tag{3.7}$$

を計算する。ここで、

$$z(t) \equiv \begin{bmatrix} \phi(t) \\ \dot{\phi}(t) \\ \ddot{\phi}(t) \\ \vdots \\ \phi^{(n)}(t) \end{bmatrix} \qquad (t = 0, \Delta t, 2\Delta t, \cdots) \tag{3.8}$$

と定義すると、上式は、

$$\frac{\mathrm{d}z}{\mathrm{d}t} = Az(t) + f(t) \tag{3.9}$$

ただし、

$$A = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \ddots & 0 \\ \vdots & \vdots & 0 & \ddots & 0 \\ 0 & 0 & 0 & \ddots & 1 \\ -b_0 & -b_1 & -b_2 & \cdots & -b_{n-1} \end{bmatrix}, \quad f(t) = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ w(t) \end{bmatrix} \tag{3.10}$$

である。上記のベクトル z に対して、$z(0)$ を求める。このベクトルは、正定値のモーメント行列 M を持つ。M は、

$$M = [m_{ij}] = \begin{bmatrix} 0 & (i+j = \text{奇数}) \\ (-1)^{\frac{j-i}{2}} m_{\frac{i+j}{2}} & (i+j = \text{偶数}) \end{bmatrix} \tag{3.11}$$

の成分を持ち、$m_0, m_1, \cdots, m_{n-1}$ は、n 個の線形方程式

$$(-1)^k \sum_{\frac{k}{2} \le q \le \frac{n+k}{2}} (-1)^q b_{2q-k} m_q = \begin{bmatrix} 0 & (k = 0, 1, \cdots, n-2) \\ \frac{1}{2} & (k = n-1) \end{bmatrix} \tag{3.12}$$

を解くことで得られる。解いて得られた行列 M に対し、Cholesky decomposition

$$M = TT^* \tag{3.13}$$

により、行列 T を求める。
ここで、$w_1, w_2, \cdots$ を、平均が 0、分散が 1 のランダム数とし、

$$w^{(0)} \equiv \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}, w^{(1)} \equiv \begin{bmatrix} w_{n+1} \\ w_{n+2} \\ \vdots \\ w_{2n} \end{bmatrix}, \quad \cdots \tag{3.14}$$

というベクトルを定義すると、

$$z(0) = Tw^{(0)} \tag{3.15}$$

により、$z(0)$ を求められる。(モーメント行列を計算すると、$E(z(0)z^*(0)) = E(Tw^{(0)}(w^{(0)})^*T^*) = TT^* = M$ より、確かにモーメント行列 M を持つ。)

次に、z は、微分方程式 (3.9) を満たすから、

$$z(t+\Delta t) = \mathrm{e}^{A\Delta t}z(t) + r \tag{3.16}$$

ただし、

$$r = \int_0^{\Delta t} \mathrm{e}^{(\Delta t - s)A} f(t+s)\mathrm{d}s \tag{3.17}$$

である。r を計算するために、以下のモーメント行列を考える。

$$\begin{aligned} M_r &= E(rr^*) \\ &= E\int_0^{\Delta t}\int_0^{\Delta t} \mathrm{e}^{(\Delta t - s_1)A} f(t+s_1)f^*(t+s_2)\mathrm{e}^{(\Delta t - s_2)A^*}\mathrm{d}s_1\mathrm{d}s_2 \\ &= \int_0^{\Delta t} \mathrm{e}^{sA}C\mathrm{e}^{sA^*}\mathrm{d}s \end{aligned} \tag{3.18}$$

上式の被積分関数を $J(s)$ とすると、

$$\frac{\mathrm{d}}{\mathrm{d}s}J(s) = AJ(s) + J(s)A^* \tag{3.19}$$

が成り立つ。$0 < s < \Delta t$ で積分することにより、

$$\mathrm{e}^{A\Delta t}C[\mathrm{e}^{A\Delta t}]^* = AM_r + M_rA \tag{3.20}$$

を得る。ここで、

$$\exp(A\Delta t)] \equiv [e_{ij}], \quad M_r \equiv [\mu_{ij}] \qquad (i,j = 1,2,\cdots n) \tag{3.21}$$

と定義すると、上式は、

$$\sum_{k=1}^{n}(a_{ik}\mu_{kj} + a_{jk}\mu_{ik}) = \left[\begin{array}{l} e_{in}e_{jn} \quad (i < n \text{ または } j < n) \\ e_{nn}^2 - 1 \quad (i = n \text{ かつ } j = n) \end{array} \right. \tag{3.22}$$

と書ける。上式の右辺を b_{ij} と書くことにし、i と j を入れ替えても、上式は変わらないことを用いれば、

$$\sum_{k\le j} a_{ik}\mu_{jk} + \sum_{k>j} a_{ik}\mu_{kj} + \sum_{k\le i} a_{jk}\mu_{ik} + \sum_{k>i} a_{jk}\mu_{ki} = b_{ij} \quad (1 \le j \le i \le n) \tag{3.23}$$

について解けばよいと言える。(3.23) を解き、M_r を得たら、Cholesky decomposition$M_r = T_rT_r^*$ により T_r を求める。次に、$t = 0, \Delta t, 2\Delta t, \cdots$ に対して、z を求めていく。$t + \Delta t = \nu\Delta t$ とおく。式 (3.16)、つまり

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

と、

$$r = T_r w^{(\nu)} \tag{3.24}$$

により、z が求められる。($w^{(\nu)}$ は、(3.14) で定義したもの。) これより、必要となる時系列 $x(t)$ は、

$$x(t) = a_0 z_1(t) + a_0 z_1(t) + \cdots + a_m z_{m+1}(t) \tag{3.25}$$

によって得ることができる。(z_i は、ベクトル z の i 成分。) なぜならば、上式は、ベクトル z の定義 (3.8) より、まさに $P(D)\phi(t) = x(t)$ となっているからである。

3.4 LISO について

LISO は、あるスペクトルに対し、多項式の比の関数を作ってもとのスペクトルに似せること (フィッティング) を行うプログラムである。多項式の比の関数 $H(s)$ は、

- 「`pole` f」
  因子 $\frac{1}{(1+\frac{s}{2\pi f})}$ に対応。
- 「`zero` f」
  因子 $(1 + \frac{s}{2\pi f})$ に対応。
- 「`pole` f Q」
  因子 $\frac{1}{(1+\frac{s}{2\pi fQ}+\frac{s^2}{(2\pi f)^2})}$ に対応。
- 「`zero` f Q」
  因子 $(1 + \frac{s}{2\pi fQ} + \frac{s^2}{(2\pi f)^2})$ に対応。
- 「`factor` x」
  関数全体を x 倍する。

これらのステートメントの組み合わせ（因子のかけ合わせ）によって構成される。(今回の実験で用いたものについてのみ述べている。)

3.5 実践の結果

3.5.1 作成したプログラム

前節までで述べた原理、また、論文 [3]、[4] をもとに作成した、フェイクノイズを生成させ、もともとの与えられたスペクトル、LISO により近似した関数、生成した時系列をフーリエ変換したスペクトル、の 3 者を比較するためにプロットさせるプログラムは、付録 A に示した。

3.5.2　得られたスペクトル

前節で述べたプログラムを、同じ日に二度実行して得られたスペクトルを以下に掲載する。なお、LISO でフィッティングした結果の Pole,Zero 等は、
$(f, Q) = (864.0639956069 \times 10^{-6}\text{Hz}, 594.3882430466)$,
$(48.0422517921 \times 10^{-3}\text{Hz}, 316.5424045767)$,
$(849.5491774735 \times 10^{-6}\text{Hz}, 803.3840475413)$ の「`pole` f Q」と、
$f = 98.5599781922\text{Hz}, 13.5806817955 \times 10^{-3}\text{Hz}$,
$14.1633977546 \times 10^{-3}\text{Hz}, 15.0706905484 \times 10^{-3}\text{Hz}$,
$15.1244329004 \times 10^{-3}\text{Hz}$ の「`zero` f」である。

図 3.1: プログラム実行の結果、得られたスペクトル (1)

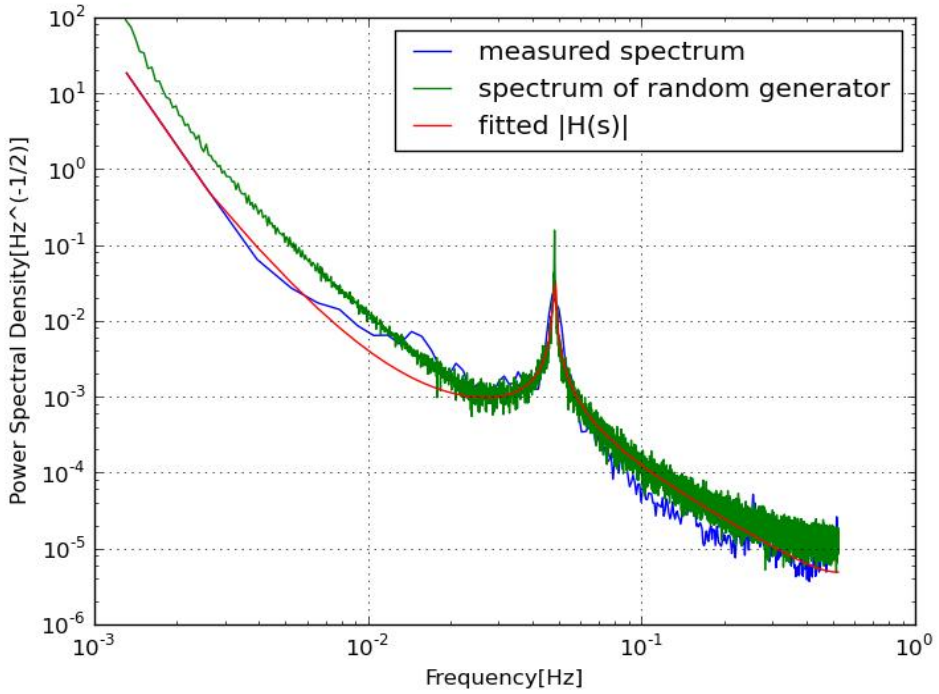
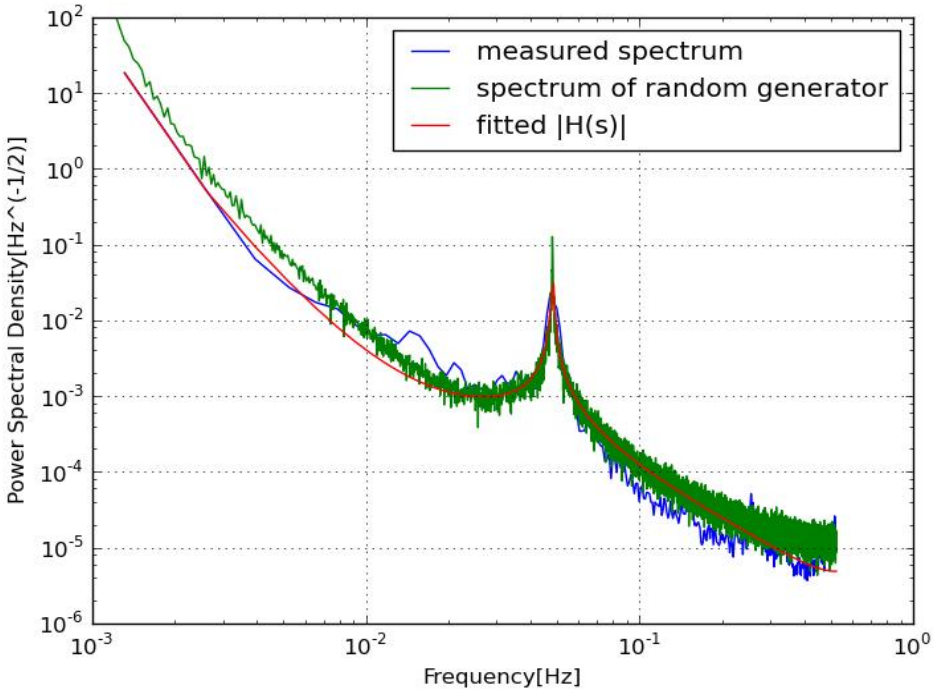


図 3.2: プログラム実行の結果、得られたスペクトル (2)



3.5.3　考察

前節で掲載したスペクトルからは、同じプログラムを実行しても、生成した時系列をフーリエ変換したスペクトル (緑線) は、目で分かるほど変動することが分かる。同じプログラムなので、プログラム内で用いた、平均 0、分散 1 のランダム数のランダム性が原因と考えられる。
しかしながら、どちらのスペクトルにしても、LISO により与えられたスペクトルを近似した関数 (赤線) と緑線は若干のずれがある。これでは、LISO によって、あるスペクトルをうまく近似できたとしても、プログラムで生成した時系列では、もとのスペクトルを再現できないことになってしまう。そこで、我々は、別の例で新たなスペクトルを作ってみた。

図 3.3: スペクトルの例　—単純なモデル—

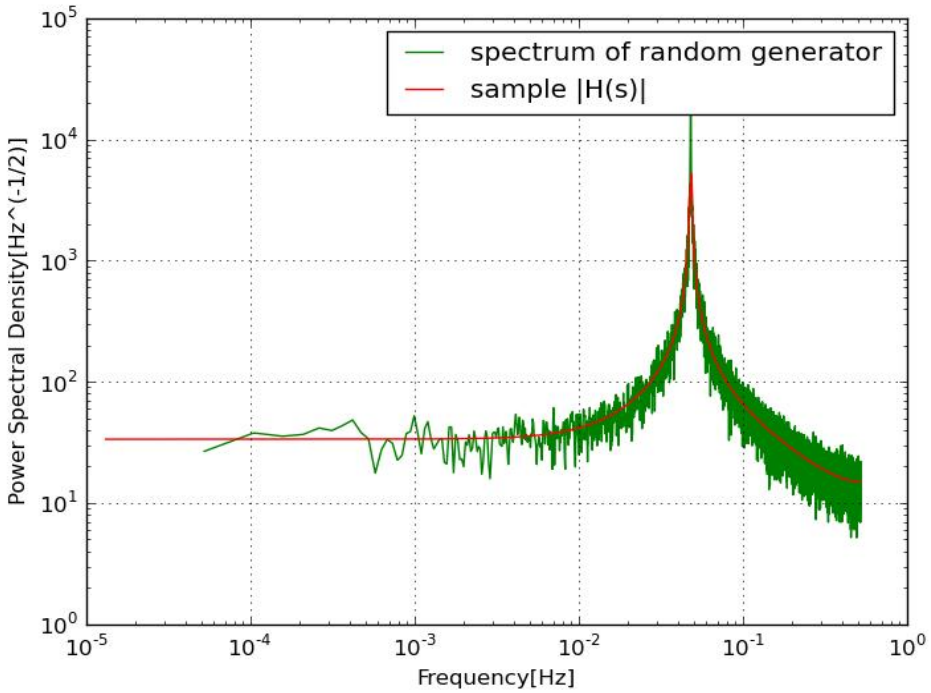


図 3.3 は、$f = 48.0422517921\text{mHz}, Q = 316.5424045767$ の「`pole` f Q」と、$f = 15.1244329004\text{mHz}$ の「`zero` f」を用いて、$H(s)$ を作り、図 3.1, 図 3.2 と同じ操作をしたものである。(SWIM のデータは除いてある。) 図 3.3 より、単純なモデルであれば、もとの関数をよく再現していると言える。たくさんの例を試したわけではないので、断定はできないが、入力する Pole や Zero の数が多くなり、関数が複雑になると、もとの関数を再現できにくくな

る、ということが起こるのではないかと思われる。

付 録A　本文中に出てきたプログラムについて

A.1　石徹白氏より頂いたプログラム

A.1.1　main20100715.m(改訂版)

```
function main2011031.m

%% Data load
load ('/Users/koji/Documents/MATLAB/gw_analysis/raw_data/data_\
samp10cut1.dat');

data = (data_samp10cut1 - mean (data_samp10cut1));
samp = 10;
cal   = 4/(4*pi*0.050/1064e-9)*1/(0.097*2);
l1   = length (data);
t    = (1:l1)/samp;

%% Rejection broken-data
data = data (601:2.6e+5);

[spe,fr]=fft_data2 (data,2^10,samp,1);
[fr,amp_s,pha_s,amp_l,pha_l]=filter_design20090119 (fr);

%% loglog (fr,abs (spe').*cal*(1+amp_l/28.9));

% Select 0.2 Hz
A=abs (spe (21,:))*cal.*(1+amp_l (21)/28.9);

% Data quality cut by eye
B=A (111:230);

% Data qality cut large 3% data
```

```
out=B<0.60e-8;
C=B (out).^2;

% plot histgrum
 [y,x]=hist (C,2^4);

 m = mean (C);
 s = std (C);

 H=3.24e-18*0.7;

 l = length (C);

 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(l);

plot (x,y,x,exp (-1.8e+17*x+3.8))
```

A.1.2 filter_design20090119.m

```
function [fr,amp_s,pha_s,amp_l,pha_l]=filter_design20090119(fr);

%% Pre Set
%f0 = 4.9e-3;
%amma = 7.46e-8;

f0 = 5.4e-3;
gamma = 7.46e-3;

if nargin<=0;
    fr_i = [-3:0.01:1]';
    fr   = 10.^fr_i;
end

w  = 2*pi*fr;

%% Servo amp
c1 = 10e-6;
```

```
c2 = 470e-12;
r1 = 1000;
r2 = 10e+3;
r3 = 10e+3;

z1 = (r1+1./(i*w*c1))*r2./((r1+1./(i*w*c1))+r2);
z2 = (r3*1./(i*w*c2))./(r3+1./(i*w*c2));

z  = z2./z1;

%% Mechanical transfer function

A  = -1e+3./(-w.^2 + i*w*gamma + (2*pi*f0).^2);

%% Results

amp_s = abs(z);
pha_s = angle(z)*180/pi;

z  = z.*A;

amp_l = abs(z);
pha_l = angle(z)*180/pi;
```

A.1.3 fft_data2.m

```
function [spe,f]=fft_data2(data,nfft,samp,opt);

% function [spe,f]=fft_data(data,nfft,samp,opt);
%
% Calculate spectrum with FFT
%   for real or complex number data series
%
%   data  : data (may be complex)
%   nfft   : point number
%   samp  : sampling rate [Hz]
%   opt   : Option for figure plotting (Default: opt=0)
%             0 : Do not plot figure
%             1 : Plot figure
```

```
%
%       May 01, 2002, Masaki Ando
%       June 04, 2008, Modified by Koji ISHIDOSHIRO
%       July 21, 2009, Removed averagin by Koji ISHIDOSHIRO

if nargin<=3,  opt=0; end

l=length(data);

av=fix(l/nfft);

    % Apply the window to the array of offset signal segments.
    window=hanning(nfft);  Be=8/3;
    %window=ones(nfft,1);  Be=1;
    y = reshape(data(1:nfft*av),nfft,av);
    y = window(:,ones(1,av)).*y;
    ys = fft(y,nfft);
    if av==1
    %  ls = sqrt( ys.*conj(ys) ./nfft./samp*2 *Be);
       ls = ys.* sqrt (1./nfft./samp*2 *Be);
   sp=ls(1:nfft/2);
    else
    %  ls = sqrt( mean( (ys.*conj(ys))' )'./nfft./samp*2 *Be);
    %  ls = sqrt( ( (ys.*conj(ys))' )'./nfft./samp*2 *Be);
       ls = ys.* sqrt (1./nfft./samp*2 *Be);
   sp=ls(1:nfft/2,:);
    end;

    fp  = samp*(0:nfft/2-1)/nfft;
     if any(any(imag(data)))    % data are complex
        fm = samp*(-nfft/2+1:0)/nfft;
        sm=ls(nfft/2+1:nfft,:);
        spe=[sm;sp];
        f=[fm,fp]';
    else
        spe=sp;
        f=fp';
    end;
```

```
if opt==1
    if any(any(imag(data)))    % data are complex
      semilogy(f,abs(spe))
    else
      loglog(f,abs(spe))
    end;
   xlabel('Frequency [Hz]')
   ylabel('Amplitude')
   title([ 'Average : ',num2str(av),'   Band width : ',num2str(f(2)-f(1)),'
Hz'])
   grid on
end
```

A.2 三つのプログラムを Python のプログラムに書き換えたもの

```
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()
```

A.3　フェイクノイズ生成プログラム

```
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


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)),15.0706
pole=array([])
zerowithQ=array([])
Qofzero=array([])
polewithQ=array([864.0639956069*(10**(-6)),48.0422517921*(10**(-3)),849.5491774735*
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,power=False)
#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('Power Spectral Density[Hz^(-1/2)]')
plt.grid(True)


nume=poly1d(inverse(transfer3(zero,pole,zerowithQ,Qofzero,polewithQ,Qofpole))*facto
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.legend(('measured spectrum','spectrum of random generator','fitted
|H(s)|'))
plt.show()
```

参考文献

[1] K.Ishidoshiro,*Search for low-frequency gravitational waves using a superconducting magnetically-levitated torsion antenna*,2009 年度博士論文

[2] K.Ishidoshiro *et al.*,*First Observational Upper Limit on a Gravitaional Wave Background at 0.2Hz with a Torsion-bar Antenna*

[3] G.Heinzel,*Generation of Random time series with prescribed spectra*,S2-AEI-TN-3034

[4] Joel N.Franklin,*Numerical Simulation of Stationary and Non-Stationary Gaussian Random Processes*,SIAM Review,Vol.7,No.1,page 68-80,1965