Conclusion

Code for Stochastic GW Background

QYQ

2017.3.29

最近用simulator4的算法模拟了随机引力波背景辐射，在此对simulator4的算法和其中所用到的函数进行一个总结。

---

1. 随机引力波源的产生

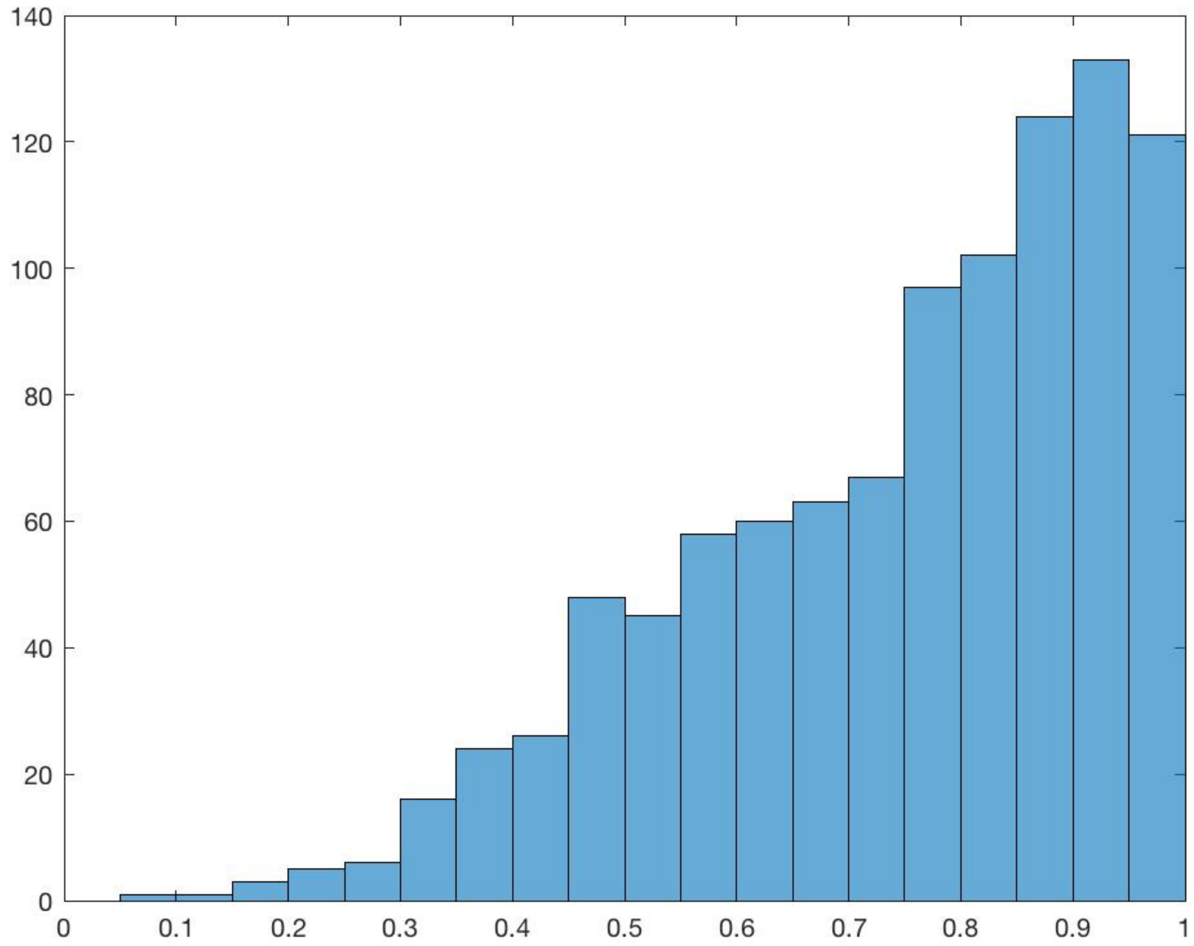
1.1 GenerateRandomGWSource

在这个算法中我们用了*GenerateRandomGWSource*这个函数来产生随机分布的引力波源的各项参数。

```
[Amp,alpha,delta,fgw,iota,Psi,Phi0,r]=GenerateRandomGWSource(Ns)
```

其中Amp，alpha，delta，fgw，iota，Psi，Phi0，r 分别是引力波源的 振幅，赤经，赤纬，频率，iota，Psi，Phi0，距离，Ns是源的数量。
在这个算法中，我们让引力波源在空间中均匀的分布，这就要求引力波源随半径的概率密度函数要和 r^2 成比例，我首先产生了r^3在（0，1）内的均匀分布，然后让其开三次方即可得到按照 r^2 分布的概率密度。

```
pr0 = random('uniform',0,1,1,Ns);
pr1 = pr0.^(1/3);
histgram(Pr1)
```



至于其他的参数：

```
log10Mc = random('uniform',6,10,1,Ns);
Mc = 10.^(log10Mc);
[alpha,delta]=SpherePointPicking(Ns);
log10fgw = random('uniform',-9,-6,1,Ns);% generate uniform distribution of l
og10(fgw)
fgw = 10.^(log10fgw);% get the real fgw
iota = random('uniform',0,pi,1,Ns);
Psi = random('uniform',0,pi,1,Ns);
Phi0 = random('uniform',0,pi,1,Ns);
```

其中Mc是Chirp Mass，在$(10^6, 10^9)$内均匀分布，fgw在$(10^{-9}, 10^{-6})$内均匀分布，iota，Psi，Phi0都在$(0, \pi)$内均匀分布。
这里引力波的振幅

$$Amp = \frac{G\mu a^2\omega^2}{c^4 D} = \frac{G^{5/3}}{c^4 D} M_c^{5/3}\omega^{-1/3}$$

1.2 SpherePointPicking

在这个函数内还嵌套了一个函数*SpherePointPicking*。这个函数是在球面上随机的取点，用来产生随机的引力波源的位置。

```
function [theta,phi]=SpherePointPicking(n)
%Uniform random Sphere Point Picking
%r = 1;
NN = n;
u = random('uniform',0,1,1,NN);
v = random('uniform',0,1,1,NN);
theta = 2*pi*u;
phi = acos(2*v-1)-pi/2;
```

2. 脉冲星阵列

为了测量引力波，我们需要选取能用的脉冲星阵列，并以此来测量引力波所带来的计时残差。我们需要知道脉冲星的位置和距离。在本算法中，我们用了17个已知并且能够用来测量的脉冲星：

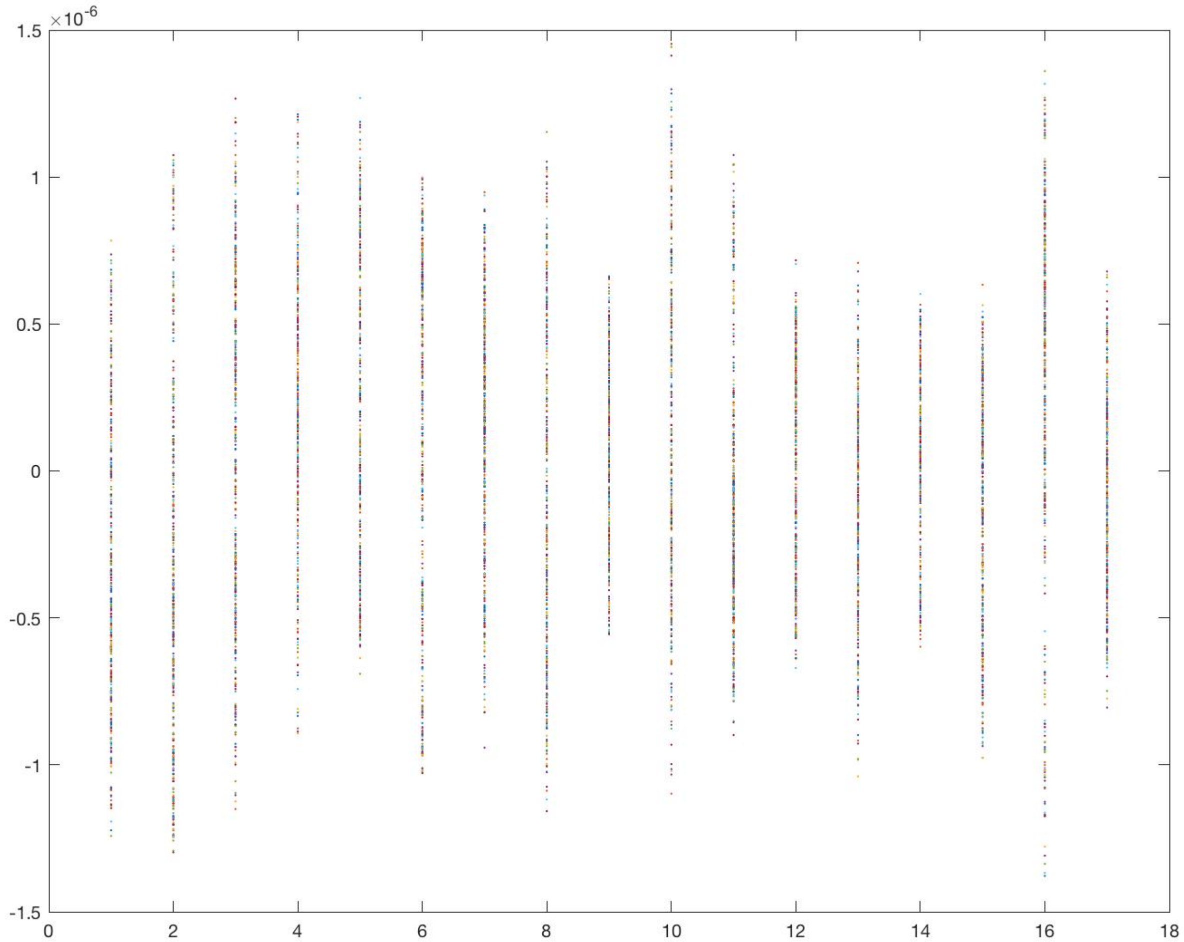
```
tmp1='J0030+0451';
alphaP(1)=(0*15+30*15/60)*pi/180;
deltaP(1)=(4+51/60)*pi/180;
distP(1)=1.376*kilo*pc2ly;  % in ly
sd(1)=1.0*10^(-8); %0.148;

tmp2='J0613-0200';
alphaP(2)=(6*15+13*15/60)*pi/180;
deltaP(2)=-(2+0/60)*pi/180;
distP(2)=6.318*kilo*pc2ly;
sd(2)=1.0*10^(-8); %0.178;

tmp3='J1713+0747';
alphaP(3)=(17*15+13*15/60)*pi/180;
deltaP(3)=(7+47/60)*pi/180;
distP(3)=7.524*kilo*pc2ly;
sd(3)=1.0*10^(-8); %0.03;

tmp4='J1909-3744';
alphaP(4)=(19*15+9*15/60)*pi/180;
deltaP(4)=-(37+44/60)*pi/180;
distP(4)=3.532*kilo*pc2ly;
sd(4)=1.0*10^(-8); %0.038;
```

```
%------------
tmp5='J1012+5307';
alphaP(5)=(10*15+12*15/60)*pi/180;
deltaP(5)=(53+7/60)*pi/180;
distP(5)=1.045*kilo*pc2ly;
sd(5)=1.0*10^(-8); %0.276;

tmp6='J1455-3330';
alphaP(6)=(14*15+55*15/60)*pi/180;
deltaP(6)=-(33+30/60)*pi/180;
distP(6)=6.593*kilo*pc2ly;
sd(6)=1.0*10^(-8); %0.787;

tmp7='J1600-3053';
alphaP(7)=(16*15+0*15/60)*pi/180;
deltaP(7)=-(30+53/60)*pi/180;
distP(7)=13.532*kilo*pc2ly;
sd(7)=1.0*10^(-8); %0.163;

tmp8='J1640+2224';
alphaP(8)=(16*15+40*15/60)*pi/180;
deltaP(8)=(22+24/60)*pi/180;
distP(8)=3.675*kilo*pc2ly;
sd(8)=1.0*10^(-8); %0.409;

% ---- add other nanograv pulsars  09/25/2014  YW
tmp9='J1643-1224';
alphaP(9)=(16*15+43*15/60)*pi/180;
deltaP(9)=-(12+24/60)*pi/180;
distP(9)=2.735*kilo*pc2ly;
sd(9)=1.0*10^(-8);

tmp10='J1744-1134';
alphaP(10)=(17*15+44*15/60)*pi/180;
deltaP(10)=-(11+34/60)*pi/180;
distP(10)=0.453*kilo*pc2ly;
sd(10)=1.0*10^(-8);

tmp11='J1853+1308';
alphaP(11)=(18*15+53*15/60)*pi/180;
deltaP(11)=(13+8/60)*pi/180;
distP(11)=7.243*kilo*pc2ly;
sd(11)=1.0*10^(-8);

tmp12='B1855+09';  % J1857+0943
alphaP(12)=(18*15+57*15/60)*pi/180;
deltaP(12)=(9+43/60)*pi/180;
```

```
distP(12)=3.239*kilo*pc2ly;
sd(12)=1.0*10^(-8);

tmp13='J1910+1256';
alphaP(13)=(19*15+10*15/60)*pi/180;
deltaP(13)=(12+56/60)*pi/180;
distP(13)=13.542*kilo*pc2ly;
sd(13)=1.0*10^(-8);

tmp14='J1918-0642';
alphaP(14)=(19*15+18*15/60)*pi/180;
deltaP(14)=-(6+42/60)*pi/180;
distP(14)=6.437*kilo*pc2ly;
sd(14)=1.0*10^(-8);

tmp15='B1953+29';  % J1955+2908
alphaP(15)=(19*15+55*15/60)*pi/180;
deltaP(15)=(29+8/60)*pi/180;
distP(15)=0.875*kilo*pc2ly;
sd(15)=1.0*10^(-8);

tmp16='J2145-0750';
alphaP(16)=(21*15+45*15/60)*pi/180;
deltaP(16)=-(7+50/60)*pi/180;
distP(16)=3.982*kilo*pc2ly;
sd(16)=1.0*10^(-8);

tmp17='J2317+1439';
alphaP(17)=(23*15+17*15/60)*pi/180;
deltaP(17)=(14+39/60)*pi/180;
distP(17)=5.281*kilo*pc2ly;
sd(17)=1.0*10^(-8);
```

3. 计时残差的计算

在本算法中，我们对每一个源对每一个脉冲星所产生的计时残差进行累加，得到一个最终的计时残差（每一个脉冲星）：



详细的计时残差的计算在函数*FUllResiduals*中。

```
for i=1:1:Np
    for j=1:1:Ns  % number of GW sources

        % GW sky location in Cartesian coordinate
        k=zeros(1,3);  % unit vector pointing from SSB to source
        k(1)=cos(delta_tmp(j))*cos(alpha_tmp(j));
        k(2)=cos(delta_tmp(j))*sin(alpha_tmp(j));
        k(3)=sin(delta_tmp(j));
        theta=acos(k*kp(i,:)');
        %sprintf('%d pulsar theta=%g',i,theta)
        %phiI(i)=mod(phi0-omega*distP(i)*(1-cos(theta)), 2*pi);  % modulus a
fter division
        %phiI(i)=mod(2*phi0-omega_tmp(l)*distP(i)*(1-cos(theta)), pi);  % mo
dulus after division, YW 09/10/13
        phiI(i)=mod(phi0(j)-0.5*omega_tmp(j)*distP(i)*(1-cos(theta)), pi);
% modulus after division, YW 04/30/14 check original def. of phiI

        %disp(['pulsar = ', num2str(i), ' ', num2str(phiI(i))])
```

```
        tmp = FullResiduals(alpha_tmp(j),delta_tmp(j),omega_tmp(j),phi0(j),p
hiI(i),alphaP(i),deltaP(i),...
            Amp(j),iota(j),thetaN(j),theta,yr);
        timingResiduals_tmp(i,:) = timingResiduals_tmp(i,:)+tmp';
        %fftsignal(i,:)=fft(timingResiduals_tmp(i,:));

        % calculate the perfect fitness value

        %snr_chr = snr_chr + dot(timingResiduals_tmp(i,:),timingResiduals_tm
p(i,:)) / sd(i)^2;

        % standardization of the true coordinates
        %                 stdTrueCoord(1)=(alpha_tmp(j)-xmaxmin(1,2))/(xmaxm
in(1,1)-xmaxmin(1,2));  % [0, 2*pi]
        %                 stdTrueCoord(2)=(delta_tmp(j)-xmaxmin(2,2))/(xmaxm
in(2,1)-xmaxmin(2,2));  % [-pi/2, pi/2]
        %                 stdTrueCoord(3)=(omega_tmp(j)-xmaxmin(3,2))/(xmaxm
in(3,1)-xmaxmin(3,2));  % [2, 20]
        %                 stdTrueCoord(4)= mod(phi0(j),pi)/pi;  % [0, pi]
        %                 stdTrueCoord(5)=(log10(Amp(j))-xmaxmin(5,2))/(xmax
min(5,1)-xmaxmin(5,2));
        %                 stdTrueCoord(6)=(iota(j)-xmaxmin(6,2))/(xmaxmin(6,
1)-xmaxmin(6,2));
        %                 stdTrueCoord(7)=(thetaN-xmaxmin(7,2))/(xmaxmin(7,1
)-xmaxmin(7,2));

    end


    %snr_chr=sqrt(snr_chr/Np);  % averaged snr--root of mean square of indiv
idual snr
    %snr_chr=sqrt(snr_chr);

    % signal + noise realizations
    %for jj=1:1:Nrlz

    %nf=nf+1;

    %rlz_id=jj; %num2str(jj);

    % structure to store the id tag for each metadata file
    %id=struct('snr_id',snr_id,'loc_id',loc_id,'omg_id',omg_id,'rlz_id',rlz_
id);

    %snr_tmp=0;

    %                 for i=1:1:Np
    %
```

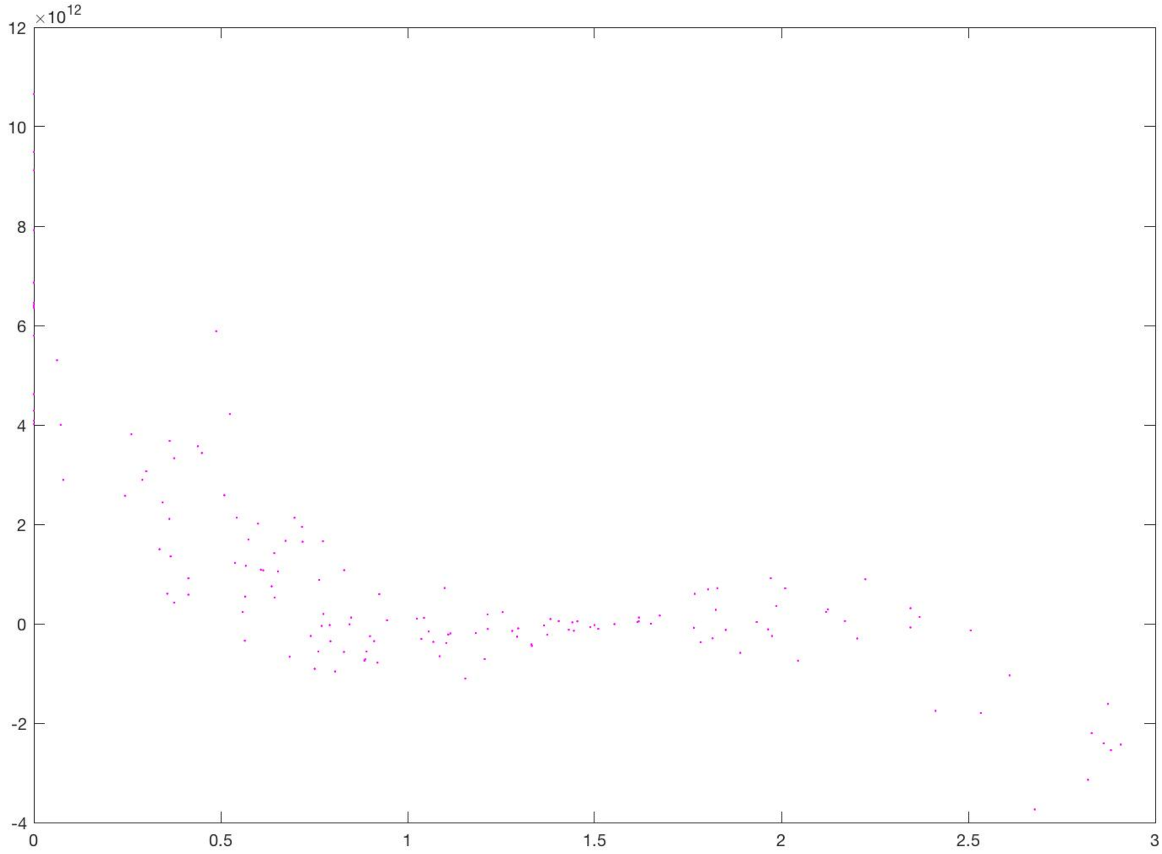
```
    %                      % generating a realization of noise
    %                      noise(i,:)=sd(i)*randn(1,N);  % Gaussian noise
    %                      % calculate the actual snr
    %                      %fftnoise(i,:)=fft(noise(i,:));
    %
    %                      %snr_tmp=snr_tmp+ fftsignal(i,:)*fftsignal(i,:)/ff
tnoise(i,:);
    %                      timingResiduals(i,:)=timingResiduals_tmp(i,:)+nois
e(i,:);  % add noise on signal
    %
    %                  end

    inParams = struct('Np',Np,'N',N,'Ns',Ns,'s',timingResiduals_tmp,'sd',sd,
...
        'alphaP',alphaP,'deltaP',deltaP,'kp',kp,'yr',yr);

    %perfect_fitness = LLR_PSOmpp(stdTrueCoord,inParams);  % - LogLikelihood
Ratio, minimization
    %true_fitness = fitnessTrue_ie(alpha_tmp(j),delta_tmp(j),omega_tmp(l),Am
p,iota,thetaN,phi0,phiI,inParams);

    %disp(['In simulator2: perfect_fitness: ', num2str(perfect_fitness)]);

    % save metadata into a file for each realization (file name rule)
    %filename=strcat('snr',num2str(ii),'loc',num2str(j),'omg',num2str(j),'rl
z',num2str(jj),'.mat');

    %end

    % end

end
```

4. 相关系数的计算

有了计时残差后，我们要计算每一对脉冲星之间的相关系数，可以根据下式进行计算

$$r(\theta) = \frac{1}{N}\sum_{i=0}^{N-1} R(t_i, \hat{k_1})R(t_i, \hat{k_2})$$

其中$R(t_i, \hat{k_1})$是i时刻第一个脉冲星的计时残差，$\hat{k_1}$是第一个脉冲星的位失。
把这17个脉冲星两两一对，分别计算相关系数后可以得到：




```
function [CE,thetaC] = CorrelationCoefficient()
run simulator4.m;
load('/Users/qianyiqian/desktop/matlabprograms/PTAcode/GWB/GWB.mat');
Np=getfield(simParams,'Np');
kp=getfield(simParams,'kp');
%N=getfield(simParams,'N');
cthetaC=zeros(1,(Np-1)*Np/2);%%%cos(theta) between every two pulsar
CE=zeros(1,(Np-1)*Np/2);%%%correlation coefficients
ct = 1;%% counter

for i=1:1:Np-1
    for j=1+i:1:Np

        cthetaC(:,ct)=kp(i,:)*kp(j,:)';%%% cos(theta) between every two puls
ar
        R=(timingResiduals_tmp(i,:)*timingResiduals_tmp(i,:)')...
        *(timingResiduals_tmp(j,:)*timingResiduals_tmp(j,:)');
        CE(:,ct) = timingResiduals_tmp(i,:)*timingResiduals_tmp(j,:)'/sqrt(R
);
        ct = ct+1;

    end
end
thetaC=acos(cthetaC)*180/pi;%%%% theta between every two pulsar
```

```
%plot(thetaC,CE,'.k');
```

5.Helling-Downs Curve

重复运行simulator4和CorrelationEfficient函数可以不断的得出相关系数，因为引力波源的位置是随机的，所以每一次的出的相关系数都不同，最后把这些相关系数做一个简单的代数平均后就能得出Helling-Downs Curve，并且和理论符合的很好。理论解是：

$$\xi(\theta) = \frac{3}{2}x\log x - \frac{x}{4} + \frac{1}{2}$$

$$x = [1 - \cos(\theta)]/2$$