**混沌时间序列分析源程序**

因为毕业设计的原因，开始接触混沌时间序列分析，刚开始的确有些不好入门，对初学者来说一些程序自己编写挺困难，特别希望有高手指点或者有参考程序。

我在论坛上得到了大家的热心帮助，作为一个“过来人”看到论坛上经常有人求助时间序列分析的相关程序，或者想参考，或者是急用，我相信大家也遇到过相似的问题——有些贴出来的程序往往是错误的或者无法运行。

为了感谢论坛和大家的帮助，也为了帮助那些初学者更快的入门，我会在这个帖子里贴出我自己毕业设计时的Matlab源程序以供大家参。

时间序列分析的主要程序也就是这些了，希望大家先看一些相关的书籍，有一些基础的知识，然后参考这些程序，请大家在熟悉理解这些程序的基础上使用，因为本人为了程序的编写方便，部分程序得到的结果并不是最终的结果，需要大家寻找到线性区或这无标度区进行拟合，相信这是比较简单的事情。

可能由于本人的疏忽，程序里有些小的错误，请大家见谅，也希望大家指正！

时间序列的时间延迟计算

function Tau=autocorrelation(data,tau\_max)

%data:输入的时间序列

%tau\_max:最大时间延迟

%Tau:返回所求时间序列的时间延迟

N=length(data);

%时间序列长度

x\_mean=mean(data);

%时间序列的平均值

data=data-x\_mean;

SSd=dot(data,data);

R\_xx=zeros(1,tau\_max);

%自相关函数初始化

for tau=1:tau\_max

%计算自相关函数

    for ii=1:N-tau

        R\_xx(tau)=R\_xx(tau)+data(ii)\*data(ii+tau);

    end

    R\_xx(tau)=R\_xx(tau)/SSd;

end

plot(1:tau\_max,R\_xx);

hold on

%作自相关函数图

line([1 tau\_max],[0 0])

title('自相关法求时间延迟');

ylabel('自相关函数');

xlabel('时间延迟');

Tau=0;

for jj=2:tau\_max

%求时间序列的时间延迟

    if R\_xx(jj-1)\*R\_xx(jj)<=0

       if abs(R\_xx(jj-1))>abs(R\_xx(jj))

           Tau=jj;break

       else

           Tau=jj-1;break

       end

    end

end

关联积分计算

function C\_I=correlation\_integral(X,M,r)

%该函数用来计算关联积分

%C\_I:关联积分的返回值

%X:重构的相空间矢量，是一个m\*M的矩阵

%M::M是重构的m维相空间中的总点数

%r:Heaviside 函数中的搜索半径

sum\_H=0;

for i=1:M-1

    for j=i+1:M

        d=norm((X(:,i)-X(:,j)),inf);

        %计算相空间中每两点之间的距离，其中NORM(V,inf) = max(abs(V)).

        if r>d

        %sita=heaviside(r,d);%计算Heaviside 函数之值n

           sum\_H=sum\_H+1;

        end

    end

end

C\_I=2\*sum\_H/(M\*(M-1));

%关联积分的值

相空间重构

function Data=reconstitution(data,m,tau)

%该函数用来重构相空间

% m:嵌入空间维数

% tau:时间延迟

% data:输入时间序列

% Data:输出,是m\*n维矩阵

N=length(data);

% N为时间序列长度

M=N-(m-1)\*tau;

%相空间中点的个数

Data=zeros(m,M);

for j=1:M

  for i=1:m

  %相空间重构

    Data(i,j)=data((i-1)\*tau+j);

  end

end

时间序列分解

function Data=disjoint(data,t)  
% 此函数用于将时间序列分解成t个不相交的时间序列  
% data:输入时间序列  
% t:延迟，也是不相交时间序列的个数  
% Data:返回分解后的t个不相交的时间序列  
N=length(data);

%data的长度  
for i=1:t  
    for j=1:(N/t)  
        Data(j,i)=data(i+(j-1)\*t);  
    end  
end

Heaviside函数的计算

function sita=heaviside(r,d)

% 该函数用来计算Heaviside函数的值

%sita:Heaviside函数的值

%r:Heaviside函数的搜索半径

%d:两点之间的距离

if (r-d)<0

    sita=0;

else

    sita=1;

end

延迟时间和时间窗口计算

function [Smean,Sdeltmean,Scor,tau,tw]=C\_CMethod(data,max\_d)

% 本函数用于求延迟时间tau和时间窗口tw

% data：输入时间序列

% max\_d：最大时间延迟

% Smean，Sdeltmean,Scor为返回值

% tau：计算得到的延迟时间

% tw：时间窗口

N=length(data);

%时间序列的长度

Smean=zeros(1,max\_d);

%初始化矩阵

Scmean=zeros(1,max\_d);

Scor=zeros(1,max\_d);

sigma=std(data);

%计算序列的标准差

% 计算Smean,Sdeltmean,Scor

for t=1:max\_d

    S=zeros(4,4);

    Sdelt=zeros(1,4);

    for m=2:5

        for j=1:4

            r=sigma\*j/2;

            Xdt=disjoint(data,t);

            % 将时间序列data分解成t个不相交的时间序列

            s=0;

           for tau=1:t

                N\_t=floor(N/t);

                % 分成的子序列长度

                Y=Xdt(:,tau);

                % 每个子序列

                %计算C(1,N/t,r,t),相当于调用Cs1(tau)=correlation\_integral1(Y,r)

                Cs1(tau)=0;

                for ii=1:N\_t-1

                    for jj=ii+1:N\_t

                        d1=abs(Y(ii)-Y(jj));

                        % 计算状态空间中每两点之间的距离,取无穷范数

                        if r>d1

                            Cs1(tau)=Cs1(tau)+1;

                        end

                    end

                end

                Cs1(tau)=2\*Cs1(tau)/(N\_t\*(N\_t-1));

                Z=reconstitution(Y,m,1);

                % 相空间重构

                M=N\_t-(m-1);

                Cs(tau)=correlation\_integral(Z,M,r);

                % 计算C(m,N/t,r,t)

                s=s+(Cs(tau)-Cs1(tau)^m);

                % 对t个不相关的时间序列求和

           end

           S(m-1,j)=s/tau;

        end

        Sdelt(m-1)=max(S(m-1,:))-min(S(m-1,:));

        % 差量计算

    end

    Smean(t)=mean(mean(S));

    % 计算平均值

    Sdeltmean(t)=mean(Sdelt);

    % 计算平均值

    Scor(t)=abs(Smean(t))+Sdeltmean(t);

end

% 寻找时间延迟tau：即Sdeltmean第一个极小值点对应的t

for i=2:length(Sdeltmean)-1

    if Sdeltmean(i)<Sdeltmean(i-1)&Sdeltmean(i)<Sdeltmean(i+1)

        tau=i;

        break;

    end

end

% 寻找时间窗口tw：即Scor最小值对应的t

for i=1:length(Scor)

    if Scor(i)==min(Scor)

        tw=i;

        break;

    end

end

混沌吸引子关联维计算

function [ln\_r,ln\_C]=G\_P\_good(data,tau,min\_m,max\_m,ss)

% 本函数是利用G-P 方法计算混沌吸引子关联维

% data::待计算的时间序列

% tau:  时间延迟

% min\_m:最小嵌入维

% max\_m:最大嵌入维

% ss:半径搜索次数

N=length(data);

%待计算的时间序列长度

ln\_C=zeros((max\_m-min\_m)/2+1,ss);

ln\_r=zeros((max\_m-min\_m)/2+1,ss);

for m=min\_m:2:max\_m

    Y=reconstitution(data,m,tau);

    %重构相空间

    M=N-(m-1)\*tau;

    %相空间点的数目

    d=zeros(M-1,M);

    for i=1:M-1

        for j=i+1:M

            d(i,j)=max(abs(Y(:,i)-Y(:,j)));

            %计算相空间中每两点之间的距离

        end

    end

    max\_d=max(max(d));

    %相空间中两点之间的最大距离

     for i=1:M-1

    %计算相空间中两点之间的最小距离

        for j=1:i

            d(i,j)=max\_d;

        end

     end

    min\_d=min(min(d));

    %相空间中两点之间的最小距离

    delt=(max\_d-min\_d)/ss;

    %搜索半径增加的步长

    for k=1:ss

        r=min\_d+k\*delt;

        %C(k)=correlation\_integral(Y,M,r);计算关联积分

        sum\_H=0;

        for i=1:M-1

            for j=i+1:M

                 if r>d(i,j)

                 %计算heaviside(r,d) 函数之值

                    sum\_H=sum\_H+1;

                 end

            end

         end

        C(k)=2\*sum\_H/(M\*(M-1));

        %关联积分的值

        ln\_C((m-min\_m)/2+1,k)=log(C(k));

        %求lnC(r)

        ln\_r((m-min\_m)/2+1,k)=log(r);

        %求lnr

    end

    clear d;

    clear Y;

    plot(ln\_r((m-min\_m)/2+1,:),ln\_C((m-min\_m)/2+1,:));

    %画出双对数图

    hold on;

end

重构相空间进行K\_L变换

function KL\_Data=K\_L\_GP(data,m,tau)

%该函数用来对时间序列的重构相空间进行K\_L变换

%data:输入的时间序列

%m:重构相空间的维数

%tau:重构相空间的时间延迟

%KL\_lamda:重构相空间协方差矩阵的m个特征值

%KL\_Data:进行K\_L变换后的m\*m维矩阵

Data=reconstitution(data,m,tau);

%对时间序列进行相空间重构

KLX=mean(Data');

%计算重构相空间矩阵每一行的平均值

KLdata=Data-KLX'\*ones(1,length(data)-(m-1)\*tau);

%相空间矩阵每一行元素减去此行的平均值

KLData=(KLdata\*KLdata')/(length(data)-(m-1)\*tau);

%求协方差矩阵

[V,D]=eig(KLData);

%计算协方差矩阵的m个特征值和特征向量

%KL\_lamda=zeros(1,m);

%for ii=1:m

    %KL\_lamda(ii)=D(ii,ii); %将协方差矩阵的特征值赋给KL\_D

%end

KL\_Data=V'\*Data;

%计算K\_L变换后的矩阵

混沌吸引子关联维计算

function [ln\_r,ln\_C]=G\_P\_KL(data,tau,min\_m,max\_m,ss)

% 本函数是利用基于KL变换的G-P 方法计算混沌吸引子关联维

% data::待计算的时间序列

% tau:  时间延迟

% min\_m:最小嵌入维

% max\_m:最大嵌入维

% ss:半径搜索次数

N=length(data);

%待计算的时间序列长度

ln\_C=zeros((max\_m-min\_m)/2+1,ss);

ln\_r=zeros((max\_m-min\_m)/2+1,ss);

for m=min\_m:2:max\_m

    YY=K\_L\_GP(data,m,tau);

    %重构相空间并对相空间进行KL变换

    Y=YY(fix(m/2.5):end,:);

    clear YY;

    M=N-(m-1)\*tau;

    %相空间点的数目

    d=zeros(M-1,M);

    for i=1:M-1

        for j=i+1:M

            d(i,j)=max(abs(Y(:,i)-Y(:,j)));

            %计算相空间中每两点之间的距离

        end

    end

    max\_d=max(max(d));

    %相空间中两点之间的最大距离

     for i=1:M-1

    %计算相空间中两点之间的最小距离

        for j=1:i

            d(i,j)=max\_d;

        end

     end

    min\_d=min(min(d));

    %相空间中两点之间的最小距离

    delt=(max\_d-min\_d)/ss;

    %搜索半径增加的步长

    for k=1:ss

        r=min\_d+k\*delt;

         %C(k)=correlation\_integral(Y,M,r);计算关联积分

        sum\_H=0;

        for i=1:M-1

            for j=i+1:M

                 if r>d(i,j)

                 %计算heaviside(r,d) 函数之值

                    sum\_H=sum\_H+1;

                 end

            end

         end

        C(k)=2\*sum\_H/(M\*(M-1));

        %关联积分的值

        ln\_C((m-min\_m)/2+1,k)=log(C(k));

        %求lnC(r)

        ln\_r((m-min\_m)/2+1,k)=log(r);

        %求lnr

    end

    clear d;

    clear Y;

    plot(ln\_r((m-min\_m)/2+1,:),ln\_C((m-min\_m)/2+1,:));

    %画出双对数图

    hold on;

end

Hurst指数分析

function [ln\_RS,ln\_n]=Hurst(data,n\_max)

%本函数是用Hurst指数分析时间序列

%data：待分析的时间序列

%n\_max：子序列的自大长度

%ln\_RS：返回的ln(R/S)的值

%ln\_n：返回的ln(n)的值

N=length(data);

%待分析时间序列的长度

ln\_n=log(10:10:n\_max)';

%返回的ln(n)的值

for n=10:10:n\_max

    a=floor(N/n);

    %时间序列分成子序列的个数

    X=reshape(data(1:n\*a),n,a);

    %把时间序列分成a个长度为n的子序列

    aver=mean(X);

    %计算每一个子序列的平均值

    cumdev=X-ones(n,1)\*aver;

    %每个子序列的元素减去本列的平均值

    cumdev=cumsum(cumdev);

    %计算每一个子序列的积累离差

    stdev=std(X);

    %计算每一个子序列的均方差

    RS=(max(cumdev)-min(cumdev))./stdev;

    %计算每一个子序列的R/S值

    ln\_RS(n/10,1)=log(mean(RS));

    %计算所有子序列R/S值的平均值

end

plot(ln\_n,ln\_RS)

关联维和Kolmogorov熵计算

function [D2,K2]=Kolmogorov\_D2(X,Y,m\_delt,tau)

%本函数用来计算关联维和Kolmogorov熵

%X:lnr满足线性区域的点

%Y:lnC满足线性区域的点

%m\_delt:嵌入维的增量

%tau:时间延迟

%D2为关联维，K2为Kolmogorov熵序列

[mm,nn]=size(X);

%计算在每个嵌入维下满足线性区域的点数mm和总嵌入维数nn

X\_mean=mean(X);

%每个嵌入维下满足线性区域的lnr平均值

Y\_mean=mean(Y);

%每个嵌入维下满足线性区域的lmC平均值

X\_new=X-ones(mm,1)\*X\_mean;

Y\_new=Y-ones(mm,1)\*Y\_mean;

D2=sum(sum(X\_new.\*Y\_new))/sum(sum(X\_new.\*X\_new));

%计算关联为D2

KK=Y\_mean-D2\*X\_mean;

for ii=1:nn-1

    KK\_delt(ii)=KK(ii)-KK(ii+1);

end

K2=KK\_delt/(m\_delt\*tau);

%计算Kolmogorov熵序列

FFT计算序列平均周期

function T\_mean=period\_mean\_fft(data)

%该函数使用快速傅里叶变换FFT计算序列平均周期

%data：时间序列

%T\_mean：返回快速傅里叶变换FFT计算出的序列平均周期

Y = fft(data);

%快速FFT变换

N = length(data);

%FFT变换后数据长度

Y(1) = [];

%去掉Y的第一个数据，它是data所有数据的和

power = abs(Y(1:N/2)).^2;

%求功率谱

nyquist = 1/2;

freq = (1:N/2)/(N/2)\*nyquist;

%求频率

subplot(121)

plot(freq,power);

grid on

%绘制功率谱图

xlabel('频率')

ylabel('功率')

title('功率谱图')

period = 1./freq;

%计算周期

subplot(122)

plot(period,power); grid on

%绘制周期－功率谱曲线

ylabel('功率')

xlabel('周期')

title('周期—功率谱图')

[mp,index] = max(power);

%求最高谱线所对应的下标

T\_mean=period(index);

%由下标求出平均周期

最大lyapunov指数计算

function lambda\_1=largest\_lyapunov\_exponent(data,m,tau,P)

%注意："这个程序得到的lambda\_1不能当做最大lyapunov指数，因根据所作出的曲线选择线性区进行拟合，此处的处理是为了程序的方便"

%本函数使用小数据量方法计算最大lyapunov指数

%data:时间序列

%m:嵌入维

%tau:时间延迟

%P:使用 FFT计算出的时间序列平均周期

%lambda\_1:函数返回的最大最大lyapunov指数值

N=length(data);

%序列长度

delt\_t=1;

Y=reconstitution(data,m,tau );

%重构相空间

M=N-(m-1)\*tau;

%M是m维重构相空间中总点数

d=zeros(M-1,M);

for j=1:M

    d\_min=1e+100;

    for jj=1:M

    %寻找相空间中每个点的最近距离点，并记下该点下标

        if abs(j-jj)>P

        %限制短暂分离

            d\_s=norm(Y(:,j)-Y(:,jj));

            %计算分离后两点的距离

            if d\_s<d\_min

                d\_min=d\_s;

                idx\_j=jj;

            end

        end

    end

    max\_i=min((M-j),(M-idx\_j));

    %计算点j的最大演化时间步长i

    for k=1:max\_i

    %计算点j与其最近邻点在i个离散步后的距离

       d(k,j)=norm(Y(:,j+k)-Y(:,idx\_j+k));

    end

end

%对每个演化时间步长i，求所有的j的lnd(i,j)平均

[l\_i,l\_j]=size(d);

for i=1:l\_i

    q=0;

    y\_s=0;

    for j=1:l\_j

        if d(i,j)~=0

            q=q+1;

            y\_s=y\_s+log(d(i,j));

        end

    end

    if q>0

      y(i)=y\_s/(q\*delt\_t);

    end

end

x=1:length(y);

pp=polyfit(x,y,1);

lambda\_1=pp(1);

yp=polyval(pp,x);

plot(x,y,'-')

利用互信息法求时间延迟

function [Tau,I\_sq]=mutual\_information(data,tau\_max,n)

%本函数是利用互信息法求时间序列的时间延迟Tau

%data：待计算的时间序列

%tau\_max：最大时间延迟

%n：等间隔小格子划分数

I\_sq=zeros(tau\_max,1);

N=length(data);

%时间序列的长度

for tau=1:tau\_max

    s=data(1:N-tau);q=data(tau+1:N);

    %把单变量时间序列扩充到二维相空间(s,q)上

    as=min(s);bq=min(q);

    %在重构的相图上取框，均匀划分成n\*n个小格子

    delts=(max(s)-as)/n;deltq=(max(q)-bq)/n;

    N\_sq=zeros(n);

    for ii=1:n

    %计算位于格子(ii,jj)内的相点个数

        for jj=1:n

            for k=1:N-tau

                as\_k=(s(k)-as)/delts; bq\_k=(q(k)-bq)/deltq;

                if as\_k>=ii-1&as\_k<ii&bq\_k>=jj-1&bq\_k<jj

                       N\_sq(ii,jj)=N\_sq(ii,jj)+1;

                end

            end

        end

    end

    Ntotal=sum(sum(N\_sq));

    Ps=sum(N\_sq)/Ntotal;

    %计算位于一维s格子内的概率

    Pq=sum(N\_sq')/Ntotal;

    %计算位于一维q格子内的概率

    Psq=N\_sq/Ntotal;

    %计算位于二维格子(ii,jj)内概率

    H\_s=0;

    %计算s的熵

    H\_q=0;

    %计算q的熵

    for i=1:n

        if Ps(i)~=0

            H\_s=H\_s-Ps(i)\*log(Ps(i));

        elseif Pq(i)~=0

            H\_q=H\_q-Pq(i)\*log(Pq(i));

        end

    end

    H\_sq=0;

    %计算(s,q)的联合熵

    for i=1:n

        for j=1:n

            if Psq(i,j)~=0

                H\_sq=H\_sq-Psq(i,j)\*log(Psq(i,j));

            end

        end

    end

    I\_sq(tau)=H\_s+H\_q-H\_sq;

    %计算tau下的互信息函数

    clear s q;

    %清空变量s和q

end

混沌和噪声识别

function sigma= PCA(data,m,tau)

%该函数用主分量分析(PCA)方法识别混沌和噪声。混沌信号的主分量谱图应是一条过定

%点且斜率为负值的直线，而噪声信号的主分量谱图应是一条与X轴接近平行的直线，故

%可以用主分量分布标准方差作为识别混沌和噪声的一种特征。

%data：输入的待分析时间序列

%m：重构相空间的维数

%tau:重构相空间的时间延迟

%sigma：主分量分布的标准方差

Data=reconstitution(data,m,tau);

%对时间序列进行相空间重构

KLX=mean(Data');

%计算重构相空间矩阵每一行的平均值

KLdata=Data-KLX'\*ones(1,length(data)-(m-1)\*tau);

%相空间矩阵每一行元素减去此行的平均值

A=(KLdata\*KLdata')/(length(data)-(m-1)\*tau);

%求协方差矩阵

%A=(Data\*Data');

lamda=eig(A);

%计算协方差矩阵的特征值

lamda=-sort(-lamda);

%将特征值从大到小排序

lamda\_PCA=log(lamda/sum(lamda));

plot(lamda\_PCA);

%做主分量谱图

ylabel('PCA')

title('主分量谱图')

sigma=std(lamda\_PCA);

%计算主分量分布的标准方差