6.4 フィルター　スペクトラル分析の基礎　　工事中

各種スペクトラムの形

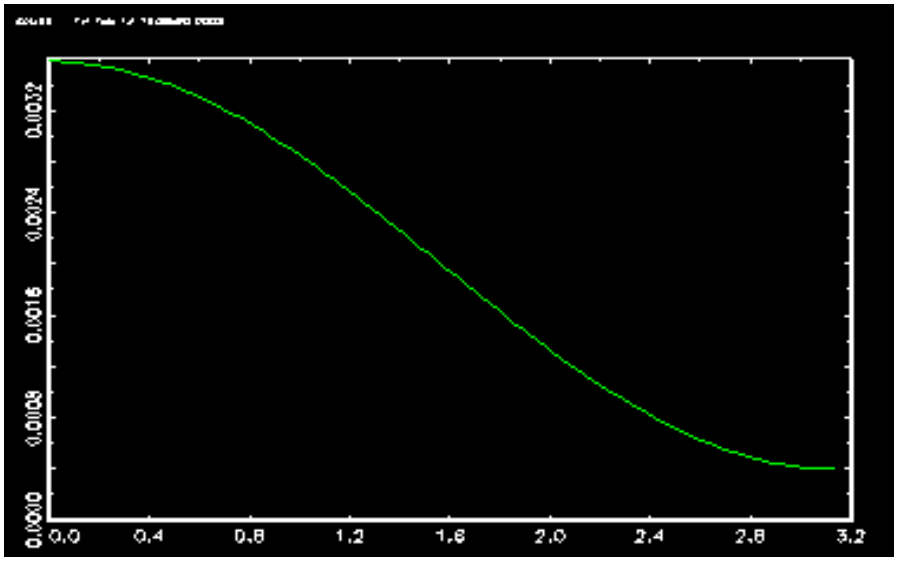
MA(1)

プログラム

```
new; cls;
theta=0.5; s2=0.01;
call specma1(theta,s2);

proc specma1(theta,s2);
    local n,lambda,spec;
    n=101;
    lambda=seqa(0,pi/(n-1),n);
    spec=(s2/(2*pi))*(1+theta^2+2*theta*cos(lambda));
    library pgraph;
    graphset;
    xy(lambda,spec);
    retp(spec);
endp;
```

グラフ表示



上のグラフは、[0,π]について描かれている。

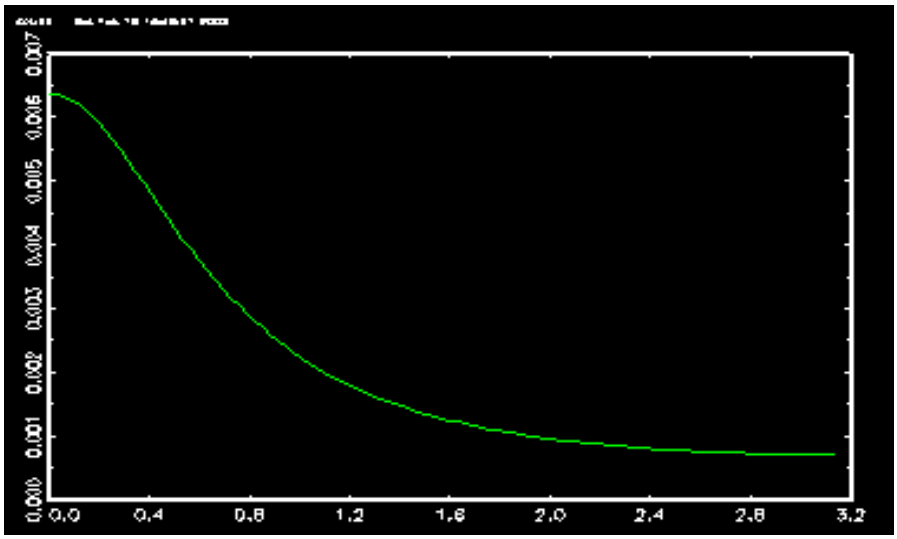
AR(1) φ＝0.5（正値）のケース

プログラム

```
new; cls;
phi=0.5; s2=0.01;
call specar1(phi,s2);

proc specar1(phi,s2);
    local n,lambda,spec;
    n=101;
    lambda=seqa(0,pi/(n-1),n);
    spec=(s2/(2*pi))*(1/(1+phi^2-2*phi*cos(lambda)));
    library pgraph;
    graphset;
    xy(lambda,spec);
    retp(spec);
endp;
```

グラフ表示

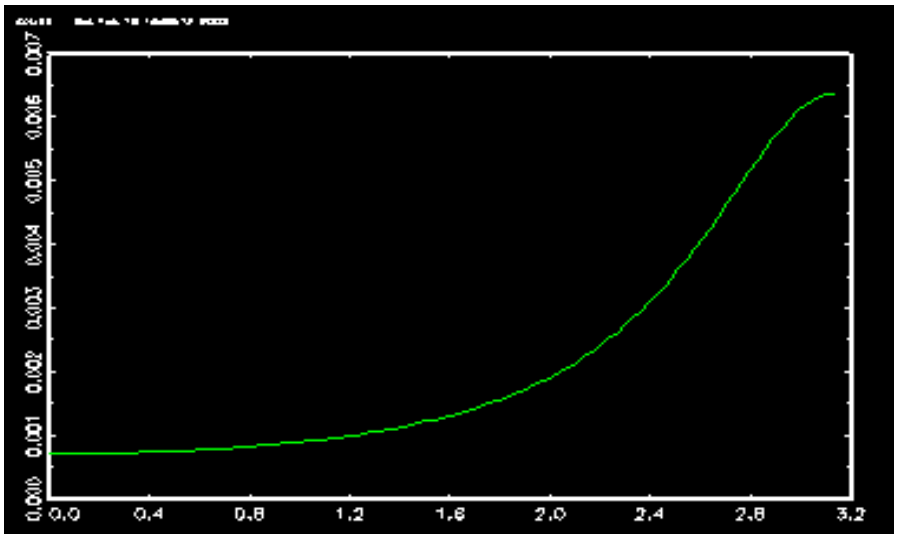


AR(1) φ＝－0.5（負値）のケース

プログラム

```
new; cls;
phi=-0.5; s2=0.01;
call specar1(phi,s2);
```

グラフ表示



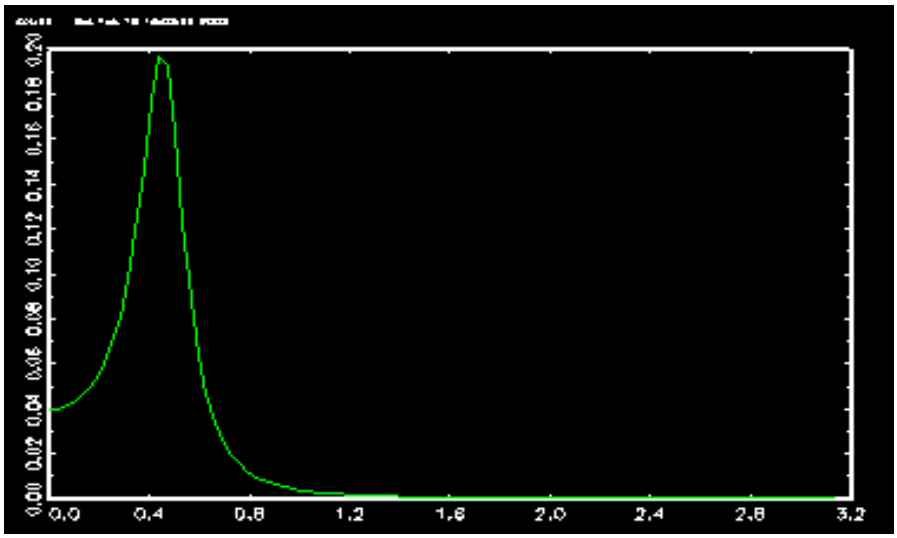
AR(2)

プログラム

```
new; cls;
phi={1.6,-0.8}; s2=0.01;
call specar2(phi,s2);

proc specar2(phi,s2);
    local n,lambda,spec;
    n=101;
    lambda=seqa(0,pi/(n-1),n);
    spec=(s2/(2*pi))*(1/(1+phi[1]^2+phi[2]^2-2*phi[1]*(1-phi[2])*cos(lambda)-
2*phi[2]*cos(2*lambda)));
    library pgraph;
    graphset;
    xy(lambda,spec);
    retp(spec);
endp;
```

グラフ表示



AR(p)一般形

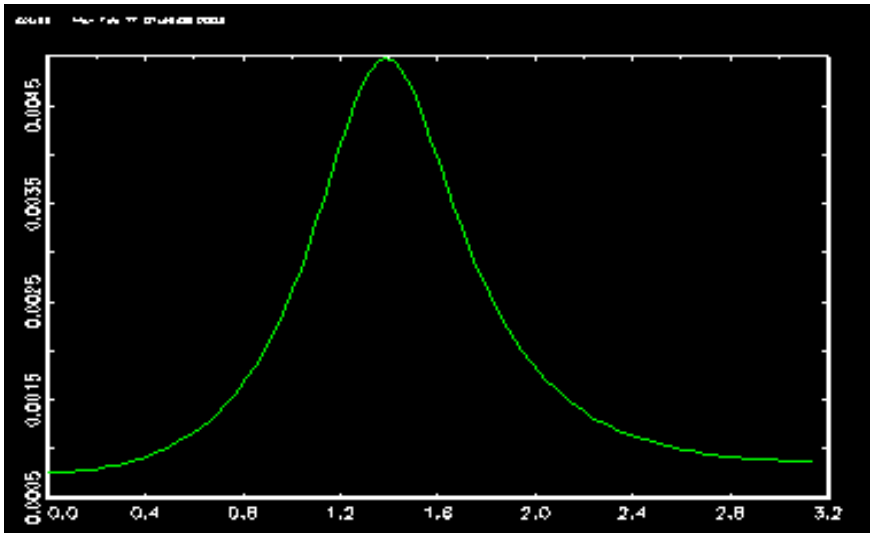
プログラム

```
new; cls;
phi={0.05,-0.4,-0.1}; s2=0.01;
call specarp(phi,s2);

proc specarp(phi,s2);
    local n,lambda,i,W,Wbar,k,spec;
    n=101;
    lambda=seqa(0,pi/(n-1),n);
    let i=1i;
    W=1; Wbar=1;
    k=1;
    do while k<=rows(phi);
        W=W-phi[k]*exp(-lambda*(-i)*k);
        Wbar=Wbar-phi[k]*exp(-lambda*i*k);
        k=k+1;
    endo;
    spec=(s2/(2*pi))*(1./(W.*Wbar));
    library pgraph;
```

```
    graphset;
    xy(lambda,spec);
    retp(spec);
endp;
```

グラフ表示



上の波形はある特殊なパラメータの場合である。上では、AR(3)のケースであるが、各パラメータの強弱で波形は異なる。なお、AR(1)やAR(2)のパラメータをφに入れても、自動的にその次数を行数から判断してこれまでの結果と同じグラフになるようにしている。各自試してもらいたい。上では虚数表現±1ｉを含むの指数関数の元の計算式を使っている。すなわち、$e^{-i\lambda}+e^{i\lambda}$を計算したものが$2\cos\lambda$を含む式に対応している。

ARMA(p,q)の一般形

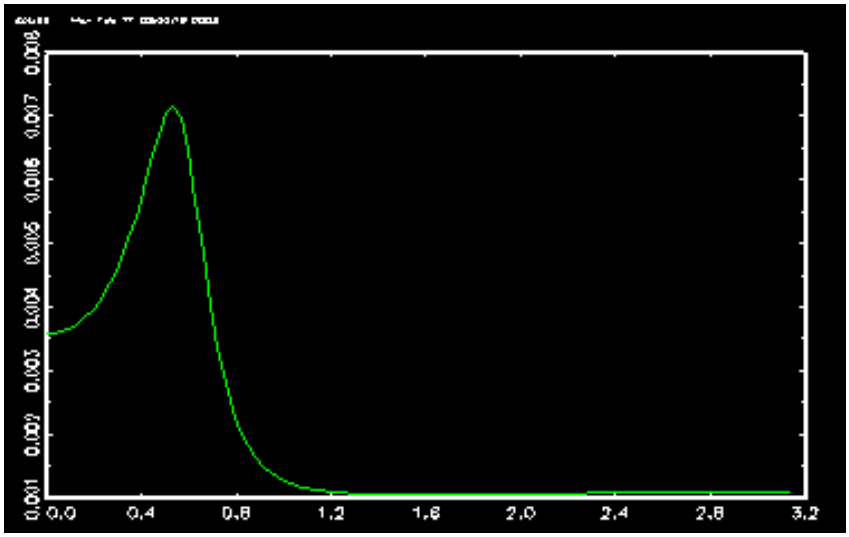
プログラム

```
new; cls;
phi={1.4,-0.7}; theta={-1.05,0.5}; s2=0.01;
call specarma(phi,theta,s2);

proc specarma(phi,theta,s2);
    local n,lambda,i,W,Wbar,Q,Qbar,k,spec;
    n=101;
    lambda=seqa(0,pi/(n-1),n);
    let i=1i;
    W=1; Wbar=1; Q=1; Qbar=1;
```

```
    k=1;
    if phi/=0;
        do while k<=rows(phi);
            W=W-phi[k]*exp(-lambda*(-i)*k);
            Wbar=Wbar-phi[k]*exp(-lambda*i*k);
            k=k+1;
        endo;
    endif;
    k=1;
    if theta/=0;
        do while k<=rows(theta);
            Q=Q+theta[k]*exp(-lambda*(-i)*k);
            Qbar=Qbar+theta[k]*exp(-lambda*i*k);
            k=k+1;
        endo;
    endif;
    spec=(s2/(2*pi))*((Q.*Qbar)./(W.*Wbar));
    library pgraph;
    graphset;
    xy(lambda,spec);
    retp(spec);
endp;
```

グラフ表示



この場合は、AR(p)の際に分母だけだったものが、MA(q)の分子の部分がその次数だけつけ

加わる。ただし、引き算ではなく、足し算で加わる。MA(q)やAR(p)単独の一般形も計算できるように、θまたはφを０と設定すれば分子または分母が１となるように一般化できるように０でないケースのみ計算するようにIF分岐させている。当然のことながら、どちらかのパラメータを０と設定すれば、これまでのMA(1)やAR(q)のケースもこれ１つで計算でき結果は全く同じになる。高次のMA(q)の計算もこれで対応できる。

ウインドウによるスペクトラムの推定

以下では与えられる系列は、すでに平均が取り除かれ、必要ならばトレンドが除去されているものとする。通常の工学の計算ではこの作業までスペクトラム計算内部に含まれているが、我々は経済データを扱うのでこの作業はその固有で慣習的な方法で処理されるため、この作業をすでに行なった前提で、その処理された系列をインプットとして与えるものとする。

Tukey Window

対称ウエイトw_jの全体の項数を2k+1として、$w_j = 0.5 + 0.5\cos(\pi j/k)$　$j = 0,1,..k$　なる$w_j = w_{-j}$という左右対称なウエイトのWindowでもってピリオドグラムを加重平均した値を計算して平滑化する方法である。

プログラム

```
proc tukeyspec(bw,y);
    local n,w,j,i,a,k,f,p;
    if (bw<0 or 0.5<=bw);
        errorlog "ERROR:bw must be 0<=bw<0.5";
        retp(".");
    endif;
    n=rows(y);
    /* weight of Tukey window */
    k=0;
    w=1;
    j=1;
    do while ( 1/(n*sumc(w^2))<bw ) and ( j<n/2 );
        a=0.5+0.5*cos(pi*seqa(0,1,j)/j);
        k=rows(a)-1;
        w=zeros(2*k+1,1);
        w[k+1]=a[1];
        i=1;
        do while i<=k;
```

```
            w[k+1-i]=a[i+1]; w[k+1+i]=a[i+1];
            i=i+1;
        endo;
        w=w/sumc(w);
        j=j+1;
    endo;
    if k==0;
        errorlog "ERROR:too small bw.";
        retp(".");
    endif;
    /* periodogram of y */
    f=seqa(0,1,n)/(2*n-1);
    let i=1i;
    p=exp(i*2*pi*f*seqa(1,1,n)')*y;
    p=abs(p)^2/n;
    p=p[k:1]|p[1]|p|p[n:n-k];
    p=conv(p,w,rows(w),rows(w)+rows(f));
    f=0|f;
    /* graph */
    library pgraph;
    graphset;
    xy(f,p~zeros(rows(p),1));
    retp(f~p);
endp;
```

Hamming Window

Tukeyのものと全く同様にして係数だけが違う$w_j = 0.54 + 0.46\cos(\pi j/k)$　$j = 0,1,..k$なる$w_j = w_{-j}$という左右対称なウエイトのWindowでもってピリオドグラムを加重平均した値を計算して平滑化する方法である。

プログラム

```
proc hammingspec(bw,y);
    local n,w,j,i,a,k,f,p;
    if (bw<0 or 0.5<=bw);
        errorlog "ERROR:bw must be 0<=bw<0.5";
        retp(".");
    endif;
```

```
    n=rows(y);
    /* weight of Hamming window */
    k=0;
    w=1;
    j=1;
    do while ( 1/(n*sumc(w^2))<bw ) and ( j<n/2 );
        a=0.54+0.46*cos(pi*seqa(0,1,j)/j);
        k=rows(a)-1;
        w=zeros(2*k+1,1);
        w[k+1]=a[1];
        i=1;
        do while i<=k;
            w[k+1-i]=a[i+1]; w[k+1+i]=a[i+1];
            i=i+1;
        endo;
        w=w/sumc(w);
        j=j+1;
    endo;
    if k==0;
        errorlog "ERROR:too small bw.";
        retp(".");
    endif;
    /* periodogram of y */
    f=seqa(0,1,n)/(2*n-1);
    let i=1i;
    p=exp(i*2*pi*f*seqa(1,1,n)')*y;
    p=abs(p)^2/n;
    p=p[k:1]|p[1]|p|p[n:n-k];
    p=conv(p,w,rows(w),rows(w)+rows(f));
    f=0|f;
    /* graph */
    library pgraph;
    graphset;
    xy(f,p~zeros(rows(p),1));
    retp(f~p);
endp;
```

Bartlett Window

対称ウエイトw_jの全体の項数を2k+1として、w_j＝1-(j/k)　j =0,1,..k　なるw_j＝w_{-j}という左右対称なウエイトのWindowでもってピリオドグラムを加重平均した値を計算して平滑化する方法である。

プログラム

```
proc bartlettspec(bw,y);
    local n,w,j,a,k,i,f,p;
    if (bw<0 or 0.5<=bw);
        errorlog "ERROR:bw must be 0<=bw<0.5";
        retp(".");
    endif;
    n=rows(y);
    /* weight of Bartlett window */
    k=0;
    w=1;
    j=1;
    do while ( 1/(n*sumc(w^2))<bw ) and ( j<n/2 );
        a=1-seqa(1,1,j)/j;
        k=rows(a);
        w=zeros(2*k+1,1);
        w[k+1]=1;
        i=1;
        do while i<=k;
            w[k+1-i]=a[i]; w[k+1+i]=a[i];
            i=i+1;
        endo;
        w=w/sumc(w);
        j=j+1;
    endo;
    if k==0;
        errorlog "ERROR:too small bw.";
        retp(".");
    endif;
    /* periodogram of y */
    f=seqa(0,1,n)/(2*n-1);
```

```
    let i=1i;
    p=exp(i*2*pi*f*seqa(1,1,n)')*y;
    p=abs(p)^2/n;
    p=p[k:1]|p[1]|p|p[n:n-k];
    p=conv(p,w,rows(w),rows(w)+rows(f));
    f=0|f;
    /* graph */
    library pgraph;
    graphset;
    xy(f,p~zeros(rows(p),1));
    retp(f~p);
endp;
```

Daniell Window

すべてが等しいウエイト $w_j=1/(2k+1)$ $j=-k,\ldots,-1,0,1,\ldots,k$ という Window でもってピリオドグラムを加重平均した値を計算して平滑化する方法である。例えば、k=2 のときには、w={1/5,1/5,1/5,1/5,1/5}となる。

プログラム

```
proc daniellspec(bw,y);
    local n,w,j,k,i,f,p;
    if (bw<0 or 0.5<=bw);
        errorlog "ERROR:bw must be 0<=bw<0.5";
        retp(".");
    endif;
    n=rows(y);
    /* weight of Daniell window */
    w=1;
    j=1;
    do while ( 1/(n*sumc(w^2))<bw ) and ( j<n/2 );
        w=(1/(2*j+1))*ones(2*j+1,1);
        j=j+1;
    endo;
    k=(rows(w)-1)/2;
    if k==0;
        errorlog "ERROR:too small bw.";
        retp(".");
```

```
    endif;
    /* periodogram of y */
    f=seqa(0,1,n)/(2*n-1);
    let i=1i;
    p=exp(i*2*pi*f*seqa(1,1,n)')*y;
    p=abs(p)^2/n;
    p=p[k:1]|p[1]|p|p[n:n-k];
    p=conv(p,w,rows(w),rows(w)+rows(f));
    f=0|f;
    /* graph */
    library pgraph;
    graphset;
    xy(f,p~zeros(rows(p),1));
    retp(f~p);
endp;
```

Parzen Window

対称ウエイトw_jの全体の項数を2k+1として、$w_j=1-6(j/k)^2+6(j/k)^3$ $j=0,1,..k/2$および$w_j=2(1-j/k)^3$ $j=k/2+1,...k$なる$w_j=w_{-j}$という左右対称なウエイトのWindowでもってピリオドグラムを加重平均した値を計算して平滑化する方法である。

プログラム

```
proc parzenspec(bw,y);
    local n,w,j,k,i,f,p,s,a1,a2,a;
    if (bw<0 or 0.5<=bw);
        errorlog "ERROR:bw must be 0<=bw<0.5";
        retp(".");
    endif;
    n=rows(y);
    /* weight of Parzen window */
    k=0;
    w=1;
    j=1;
    do while ( 1/(n*sumc(w^2))<bw ) and ( j<n/2 );
        s=seqa(1,1,2*j);
        k=rows(s);
```

```
        a1=1-6*(s[1:k/2]/k)^2+6*(s[1:k/2]/k)^3;
        a2=2*(1-s[k/2+1:k]/k)^3;
        a=a1|a2;
        w=zeros(2*k+1,1);
        w[k+1]=1;
        i=1;
        do while i<=k;
            w[k+1-i]=a[i]; w[k+1+i]=a[i];
            i=i+1;
        endo;
        w=w/sumc(w);
        j=j+1;
    endo;
    if k==0;
        errorlog "ERROR:too small bw.";
        retp(".");
    endif;
    /* periodogram of y */
    f=seqa(0,1,n)/(2*n-1);
    let i=1i;
    p=exp(i*2*pi*f*seqa(1,1,n)')*y;
    p=abs(p)^2/n;
    p=p[k:1]|p[1]|p|p[n:n-k];
    p=conv(p,w,rows(w),rows(w)+rows(f));
    f=0|f;
    /* graph */
    library pgraph;
    graphset;
    xy(f,p~zeros(rows(p),1));
    retp(f~p);
endp;
```

このWindowはBartlettと同様、その定義によれば、ｊ＝ｋの最後の一番外側のウエイトは必ず０になる。そうはならない場合には、そのプログラムは誤りである。上のすべてのWindowから得られるウエイトの和は１、また等ウエイトのDaniellを除いて、真ん中のウエイトw_0が一番大きくなる。

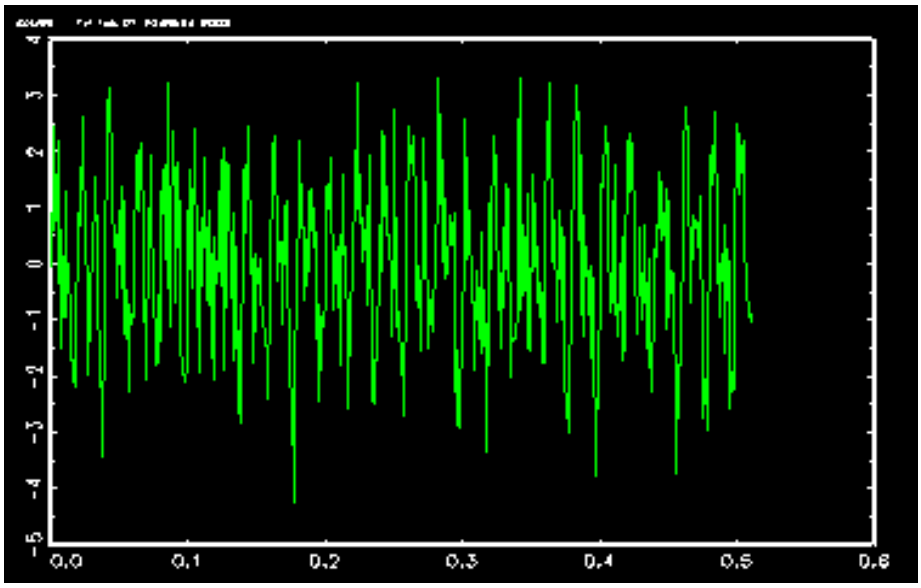
ＦＦＴ（高速フーリエ変換）の利用

プログラム

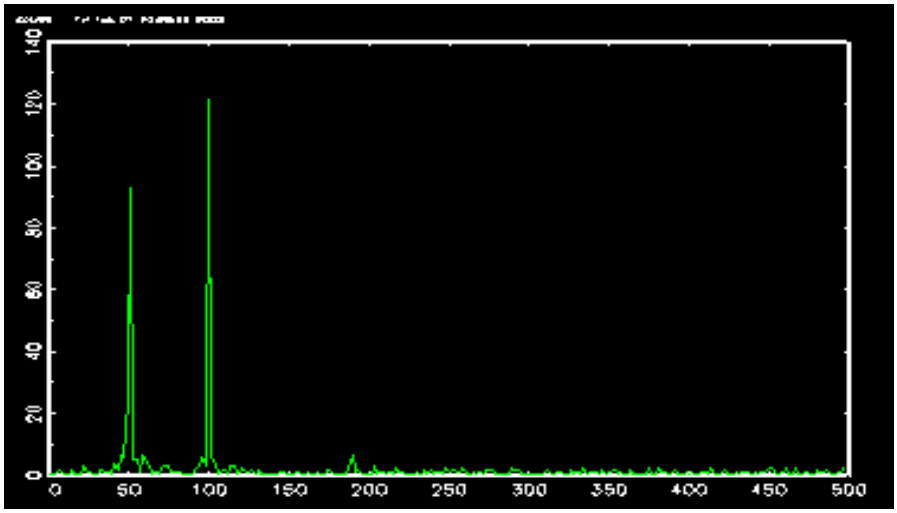
```
new; cls;
t=seqa(0,0.001,512);
s1=50;
s2=100;
y=sin(2*pi*s1*t)+sin(2*pi*s2*t)+rndn(rows(t),1);
library pgraph;
graphset;
pqgwin many;
    xy(t,y);

y=fft(y);
ps=y.*conj(y)*512;
ps=ps[1:256];
f=1000*seqa(0,1,256)/512;
    xy(f,ps);
```

グラフ表示



上のグラフでは単なるランダムな振るまいにしか見えませんが、これをフーリエ変換したものを表示すると次のようになります（上のグラフの最初の256ポイントだけを表示してあります）。



明らかに、50と100の周波数のところに信号が隠れていることがわかります。なお、GAUSSのFFTの計算ではデータ数がｎとすれば/nでスケーリングされた値を計算するので、複素共役をかけ合わせたあとで、そのデータ数で割るのではなくて１つ分をかけています。

GAUSSでは、

１Dと２DのケースのFFTには

fft(data)

多元のケースのFFTには、その第二要素に次元を列ベクトルで与えて

fftm(data,dim)

という形で関数計算をします。なお、dataの行数（個数）は２のべき乗の数（2,4,8,16,32,64,128,256,512,..）でなくてはなりません。そうでない場合には０で補われます。また計算された値はデータ数で割ったスケーリングされた値になります。

通常の工学でのFTTは上のように利用されて、隠されている周波数を見つけることに利用されることがメインになります。一方、金融や経済の時系列データを読み解くのに、上の後半部分のFFTの計算結果をログログのグラフで表示して周波数領域での波形を見分けることもできますが、これは平滑されていないので経済学的計量経済学的には大きな成果は期待できません。

FFTの利用とBand Passフィルター

この章のこれまでは周波数領域での話やその平滑法を示しましたが、もう一度、計量経済学がその主たる領域である実際のデータ領域に戻ってフィルターを作成してみましょう。以下では、中央化移動平均で純粋にトレンド＋サイクルであると仮定されるデータをさらに２のべき乗である最後の64個のデータについて分析するものとします。

プログラム

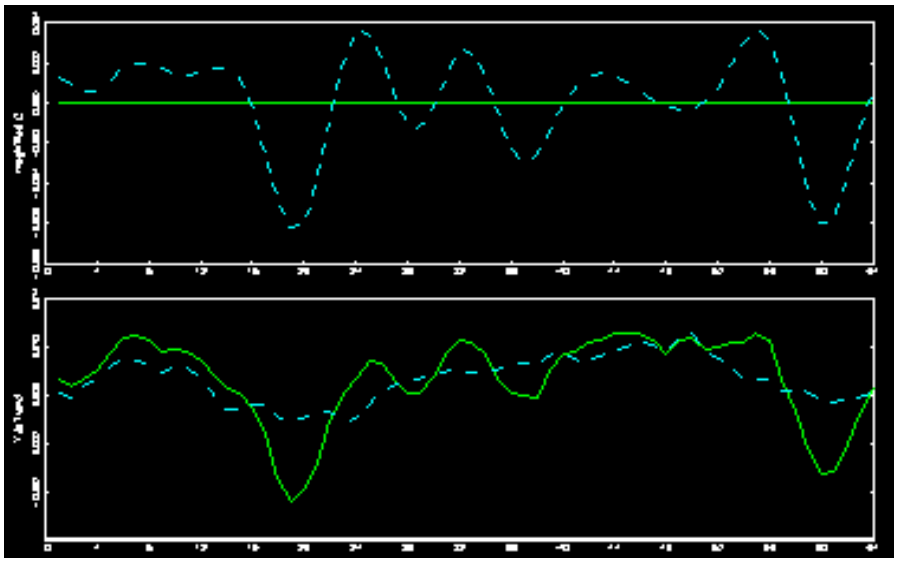
```
new; cls;
load data[91,2]=d:datafile17.txt;
y=(data[.,2]);
y=cma(y);
n=rows(y);
y=ln(y);
y=y[2:n,.]-y[1:n-1,.];
n=rows(y);
y=y[n-64+1:n];
call bandpass(y,6,32,8);

proc(2)=bandpass(y,high,low,k);
    local t,a,i,phi,theta,b,c,g;
    t=rows(y);
    /* the same weight calculation as in Baxter-King filter */
    a=zeros(k+1,1);
    a[1]=(2*pi/high-2*pi/low)/pi;
    i=1;
    do while i<=k;
        a[i+1]=(sin(i*2*pi/high)-sin(i*2*pi/low))/(i*pi);
        i=i+1;
    endo;
    phi=0;
    theta=a[1]+2*sumc(a[2:k+1]);
    theta=phi-theta/(2*k+1);
    a=a+theta;
    /* Fourier Transformation */
    b=zeros(t,1);
    b[1:k+1]=a;
    b[t-k+1:t]=a[k+1:2];
```

```
    c=real(ffti(fft(b).*fft(y)))*t;
    c=c[1:t];
    g=y-c;
    /* Graph */
    library pgraph;
    begwind;
    window(2,1,1);
    setwind(1);
        graphset;
        ylabel("maginified C");
        xtics(0,t,4,0);
        xy(seqa(1,1,t),zeros(t,1)~c);
    setwind(2);
        graphset;
        ylabel("Y & Trend");
        xtics(0,t,4,0);
        xy(seqa(1,1,t),y~g);
    endwind;
    retp(g,c);
endp;

proc cma(y);
    local t,k,m,i,m1,m2;
    t=rows(y);
    k=cols(y);
    m=zeros(t-4,k);
    i=1;
    do while i<=t-4;
        m1=(y[i,.]+y[i+1,.]+y[i+2,.]+y[i+3,.])/4;
        m2=(y[i+1,.]+y[i+2,.]+y[i+3,.]+y[i+4,.])/4;
        m[i,.]=(m1+m2)/2;
        i=i+1;
    endo;
    retp(m);
endp;
```

グラフ表示（最後の64データ）



GAUSSでは、逆フーリエ変換を行なうには、inverseという意味のｉをつけて

ffti(data)

とします。多元の場合にはfftmiとなります（なお、fftmとfftmiといった多元の関数はバージョン３の後半からのものです）。

離散フーリエ変換の利用

上ではFFTを用いた計算でしたが、手軽に利用できる反面、与えるデータは常に２のべき乗の数であるという制限があります。ここでは、離散フーリエ変換を用いて同じBand Passフィルターを作成してみます。離散の場合、discreteという意味のｄをつけて

離散フーリエ変換DFTには

dfft(data)

その逆フーリエ変換には

dffti(data)

となります。

プログラム

```
new; cls;
load data[91,2]=d:datafile17.txt;
y=(data[.,2]);
```

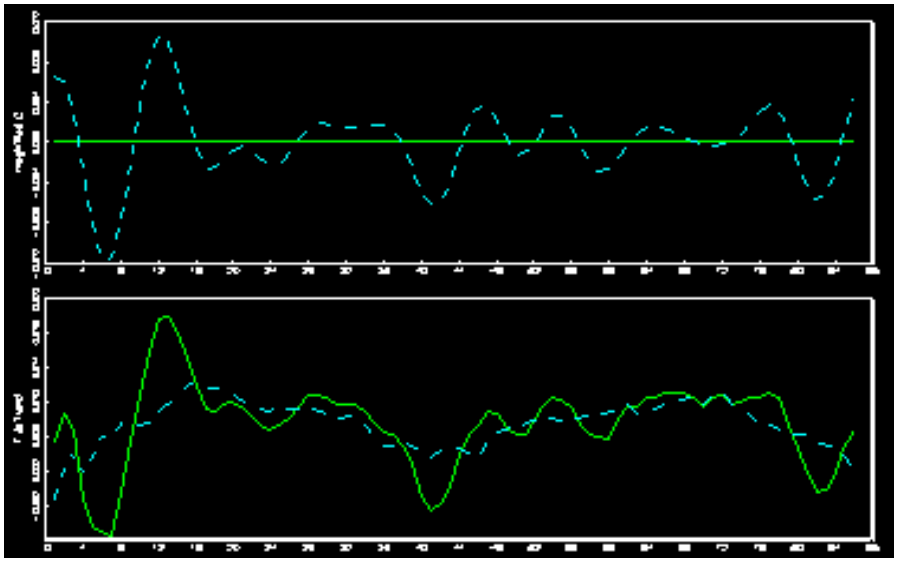
```
y=cma(y);
n=rows(y);
y=ln(y);
y=y[2:n,.]-y[1:n-1,.];
call dbandpass(y,6,32,8);

proc(2)=dbandpass(y,high,low,k);
    local t,a,i,phi,theta,b,c,g;
    t=rows(y);
    /* the same weight calculation as in Baxter-King filter */
    a=zeros(k+1,1);
    a[1]=(2*pi/high-2*pi/low)/pi;
    i=1;
    do while i<=k;
        a[i+1]=(sin(i*2*pi/high)-sin(i*2*pi/low))/(i*pi);
        i=i+1;
    endo;
    phi=0;
    theta=a[1]+2*sumc(a[2:k+1]);
    theta=phi-theta/(2*k+1);
    a=a+theta;
    /* Fourier Transformation */
    b=zeros(t,1);
    b[1:k+1]=a;
    b[t-k+1:t]=a[k+1:2];
    c=real(dffti(dfft(b).*dfft(y)))*t;
    g=y-c;
    /* Graph */
    library pgraph;
    begwind;
    window(2,1,1);
    setwind(1);
        graphset;
        ylabel("maginified C");
        xtics(0,t,4,0);
        xy(seqa(1,1,t),zeros(t,1)~c);
```

```
    setwind(2);
        graphset;
        ylabel("Y & Trend");
        xtics(0,t,4,0);
        xy(seqa(1,1,t),y~g);
    endwind;
    retp(g,c);
endp;
```
（proc cmaをここに置く）

上のプログラムでは、その前のFFTを用いたBand Passフィルターのfftおよびその逆変換ffti の代わりに、離散フーリエ変換及びその逆変換のdfftおよびdfftiに変更して余分なデータのカットの部分を除いただけのプログラムである。離散変換の場合、スピード的にはFFTに劣るとされているが、経済時系列データ程度の計算ではほとんど大差なく瞬時に計算される。高速フーリエ変換とその逆変換に比べて、離散変換とその逆変換はデータの個数を２のべき乗の数に限定する必要は全くない。

グラフ表示（全データ）



これまでの時系列フィルターのグラフ表示と同じく、上図はゼロ基準線とサイクル。そして、下図は実線がサイクル＋トレンドに対して、破線がトレンドに相当する。GAUSSでは上下の縮尺は自動的に定まるので、この場合、上のグラフの方が若干拡大されたようになっているが、下図の実線に対する破線の差異が上図の破線のゼロのまわりの動きになっている。なお、上では中央化平均で前後２データずつをカットした後のデータを用いている。