ภาคผนวก

โปรแกรม MATLAB ที่ใช้ในการวิเคราะห์

(ก) ฟังก์ชันของสัญญาณทดสอบ

ก.1 โปรแกรมหลัก

```
gam=8; N=2^gam; n=0:1:N-1; s=fix(N/6)+5; p=fix(N/4)+3; k=fix(N/20)+3; phi=pi/4.3;
f=3*realdisgatom(s,p,k,phi,N,n);                    %function sample #1
f=f+4*realdisgatom(s+1, p+fix(N/2),k+2,phi,N,n);
f(100:110)=f(100:110)+1;
f=2*sin(2*pi*k/N*n + phi);                          %function sample #2
f=2*(sin(2*pi*k/N*n + phi)).*(n<(N/2));             %function sample #3
f=zeros(1,N); f(100:150)=2.3;
```

ก.2 โปรแกรมย่อย

```
function turn=realdisgatom(s,p,k,phi,N,n)     %real discrete time-frequency atom in eq.(113)
     g=(exp(i*phi))*cpxdisgatom(s,p,k,N,n);
     g=g+(exp(-i*phi))*cpxdisgatom(s,p,-k,N,n);
     turn=kgp(s,p,k,phi,N,n)*g/2;

function turn=kgp(s,p,k,phi,N,n);             %calculate the normalize constant of eq(114)
     gp=cpxdisgatom(s,p,k,N,n);
     gn=cpxdisgatom(s,p,-k,N,n);
     inner=gp*gn';                            %when transpose it also make conjugation
     r=real((exp(i*2*phi))*inner);
     turn=sqrt(2)/sqrt(1+r);

function turn=cpxdisgatom(s,p,k,N,n)          %discrete complex Gabor atom in eq.(64)
     turn=pegwin(s,N,(n-p)).*exp(i*2*pi*k*n/N);

function turn=pegwin(s,N,n);                  %discrete periodic Gaussian window
     gs=0;     count=round(s/(N-1))+1;
```

```
    for a=-count:1:count
            gs=gs+gwin((n-a*N)/s);
    end
    %test for no periodize  gs=gwin(n/s);
     gs=gs/sqrt(s);                    %normalize to unit norm
    Ks=sum(gs.^2);      turn=gs/sqrt(Ks);

function turn = gwin(t)            %Gaussian window   g(t) = 2^1/2 e^(- pi t^2)
    turn = (2^(1/4))*exp(-pi*t.^2);
```

(ข) โปรแกรมการวิเคราะห์สัญญาณเสียงเต้นหัวใจด้วยวิธีการแม็ชชิงเพิชยูท

ข.1 โปรแกรมหลัก

```
clear, close all
% ploted curve color
    color=['b' 'g' 'm' 'y' 'r' 'c' 'k'];  % k = black colour
%---------------------------------------------------------------------------
% Read and select PCG
    [y,fs]=wavread('C:\MATLAB6p1\work\2546\HSound\3Com_9July\Late Systolic Murmur');
    fprintf('\n\t the sapling frequency record = %d\n',fs);
    plot(y),grid,...
      title('PCG recording (save figure & please ENTER)'),xlabel('samples')
    pause,close all
    Y=y;
    % when,...
      % Amplitude value of y in range [-1,+1],signal .wav
      % fs is the sample rate in Hertz
      % nbits is the number of bits per sample
%---------------------------------------------------------------------------
% Zoom PCG signal if it's very much data
    if fs>11025
      fprintf('\n\t Very much data,please select the data interval for REZOOM\n')
      pause(2)
```

```
        Y=selectpcg(Y,fs);
        figure,...
                plot(Y),grid,...
                title('REZOOM pcg signal (save figure & please ENTER)')
                pause,close all
    end         %if fs>11025
%---------------------------------------------------------------------------
% calculate data decimation factor
    df=fix(fs/2640);
    fprintf('\n\t the factor of data decimation = %d ,please select ONE cardiac cycle\n',df);
    [f,N]=redsamp(Y,fs,df);
    newfs=fs/df;
    T=1/newfs;
    n=0:N-1;
%---------------------------------------------------------------------------
% plot the selected PCG,to analysis
    figure,...
        plot(n*T,real(f)),grid,...
        xlabel('time(sec)'),title('Signal for analysis,decimation PCG')
        figure,...
                plot(n,real(f)),grid,...
                xlabel('samples'),title('Signal for analysis,decimation PCG')
    pause
    tic                             % starts a stopwatch timer.
%---------------------------------------------------------------------------
 Rf=f;
 f=Rf;
 close all
%---------------------------------------------------------------------------
% variables in used
    decomp=[];                      % decompose signal
    projsig=[];
    nrmsedisp=[];                   % display 'nrmse' (Error) in percent
    resig=0;                        % for reconstruction signal,it's not include the residue
```

```
    error=5;                    % minimun 'nrmse' requirement in Percent
    nrmse=100;                  % check for iteration process
    iter=0;                     % counting for the iteration
    loop=1;                     % counting for the color setting
%-----------------------------------------------------------------------------
% matching pursuit method
  while nrmse>error   %for ii=1:endproj  %while normRf>Rm
       iter=iter+1;
       figure(1),...
               hold on,grid on,...
               plot(n,Rf,color(loop)),pause(2)

    % search atoms by newton.m
       [g,newtdisp,x,phii]=newton(Rf,n,N);
       s=x(1);  p=x(2);   k=x(3);
       coef=Rf*g';
       proj=(coef)*g;                       % <f,g>=(f*g')=coefficeint
       Rf=Rf-proj;
       projsig=[projsig proj];
       decomp=[decomp;coef s p k phii];

    plot(n,proj,[color(loop) '+-']),pause(2)
    loop=loop+1;
    if loop==8                              % couting for colour setting
       loop=1;
    end          %if loop==

% find NRMSE (Normalized root-mean-square error)
    resig=resig+proj;                       % use for checking
    e=f-resig;
    nrmse=100*sqrt(sum(e.^2)/sum(f.^2)) ;   % reconstruct signal = e
    nrmsedisp=[nrmsedisp;nrmse];

% Show Newton result
```

```
        fprintf('\n\t-->newton found # %d\n',iter)
        fprintf('\n\t\t  Loop  \t |<f,g>|   \t\t s \t\t\t  p \t\t\tk \t\t  phase\t\t |grad|\n\n')
        disp(newtdisp)
        fprintf('\t-->NRMSE := %g\n',nrmse)
        normRf=sqrt(sum(Rf.^2));
        fprintf('\t-->|Rf|  := %g\n',normRf)
   end     % while
%------------------------------------------------------------------------------
        fprintf('\n\n\t-->the iteration of searching # %d loops',iter')
        fprintf('\n\t-->The Normalized root-mean-square:NRMSE(in Percent)****\n')
        disp(nrmsedisp)
%------------------------------------------------------------------------------
% subplot to decompose signal in figure(2)
        fig=2;  cc=0;    poss=0;  str=1; No=8;
        a=(length(projsig))/N;
        while a>=1
            cc=cc+1;
            poss=poss+1;
            n1=str;
            n2=n1+N-1;
            figure(fig),...
                        subplot(No,1,cc),...
                                    plot(n*T,projsig(n1:n2)),grid,...
                        str=poss*N;
            if cc==No
                        cc=0;
                        fig=fig+1;
            end % if cc==
            a=a-1;
        end %while a~=
%------------------------------------------------------------------------------
% subplot to residue and reconstruct signal
        figure,...
            subplot(2,1,1),plot(n*T,Rf,'r'),grid,...              % Residual signal
```

```
        subplot(2,1,2),plot(n*T,resig,'k'),grid     % Composite signal
    hold off

% compare the original signal ,the reconstruct signal and the residue
    figure,hold on,grid on,...
        plot(f),plot(resig,'+-k'),plot(Rf,'r'),...
        legend('f','reconstruct','Rf'),...
    hold off
%------------------------------------------------------------------------
% change discrete to continue parameters [ y=(s,p,(2*pi*k/N)) to y=(s,u,sigma) ]
    disp(['-->found : coff s p k phi']); decomp
    disp(['-->convert to continuous time'])
    disp(['-->coff s u freq phi'])
    decomp(:,4)=decomp(:,4)/(N*T); % CT  freq=k/NT
    decomp(:,3)=decomp(:,3)*T ; % CT u=pT
    decomp(:,2)=decomp(:,2)*T  % CT s=s*T
%------------------------------------------------------------------------
% test wigner distribution
    Resi=decomp;
    MaxFre=1/(2*T)
    tv=n*T;
    fv=0:MaxFre/100:MaxFre;
    [t,freq]=meshgrid(tv,fv);
    [ResiRow,ResiCol]=size(Resi);
    Wg=0;
    for I=1:1:ResiRow;
        Wg=Wg+Resi(I,1)*2*exp(-2*pi*(((t-Resi(I,3))/Resi(I,2)).^2 +(Resi(I,2)*2*pi*(freq-Resi(I,4))/(2*pi)).^2));
    end;  %for

% plot 3D
    figure,...
        mesh(t,freq,Wg);
%------------------------------------------------------------------------
  time=toc;
  fprintf('-->Total time: %g sec ( %g min )= %g h \n',time,time/60,time/3600);    save
```

ข.2 โปรแกรมย่อย

ข.2.1 ฟังก์ชันการกรองและเลือกสัญญาณเสียงเต้นหัวใจเพื่อนำมาวิเคราะห์

```
function Y=butterlpf(N,fc,fs,y)         % 8th-order Butterworth Low Pass Filter
    % Y=butterlpf(N,fc,fs,y)
    % when N:order, fc:cutoff frequency,fs:Sampling rate
    % -->return signal to Y
    wn=fc/(fs/2);                        % fs = sample rate of signal y
    [b,a]=butter(N,wn);                  % Nth order lowpass digital Butterworth filter
    Y=filter(b,a,y);

function Y=selectpcg(y,fs)              % for selection of PCG interval n1:n2
    n=0:length(y)-1;       T=1/fs;
    figure,...
      plot(n*T,y),grid,...               % Plot Normal Heart sounds
      xlabel('time (second)'),ylabel('Normalized amplitude')
      title('Please seclect the PCG')
    [xin,yin]=ginput(2);                 % เลือกช่วงสัญญาณที่จะนำมาวิเคราะห์
    n1=fix(fs*xin(1));
    n2=fix(fs*xin(2));
    Y=y(n1:n2);                          % ได้ช่วงสัญญาณที่จะนำมาวิเคราะห์ ช่วง n1:n2
```

ข.2.2 ฟังก์ชัน Garbor dictionary

```
function g=gausswin59(t)                % g(t) is the Gaussian window function and the continue signal
    g=(2^(1/4))*exp(-pi*t.^2);           % t=time

function gs=dscgauss63(s,n,N)           % gs(n) is 'the Discrete Gaussian window function and the discrete signal
    % s = scale factor,can only vary between 1 and N
    % p = transtation , vary between -a to +a  ( in eq.(63)vary between -inf ot +inf)
    % N = Sampling Rate or the sample of signal (suppose that signal is real)
    % Ks = K = Normalize constant the discrete norm of gs ,||gs(n)||=1
    % n = integer
    gs=0;  a=3;
```

```
    for p=-a:a
            ar=(n-p*N)/s;
            g=gausswin59(ar);
            gs=gs+g;
    end
  gs=gs/sqrt(s);
  Ks=sqrt(sum(gs.^2));                  % Ks is the Normalized constant |gs|=1
  gs=gs/Ks;

function gy=CpxGAtom64(s,p,phi,n,N)         % gy(n) is The Discrete Complex Gabor atom
  % gy(n)=gs(n-p)exp((2*pi*k*n/N))
  % y =(s,p,2*pi*k*n/N)
  % s is set of ]1,N[ , is not necessary integer
  % p(Time-position)amd k(Frequency index) are all integer
  % N=samples of signal
  gs=dscgauss63(s,n-p,N);
  gy=gs.*exp(i*phi*n);

function gyr=ReGAtom113(s,p,k,zeta,n,N)     % gyr is 'The Real discrete Time-Frequency atoms'
  % gyr = (K/2)[exp(i*zeta)(gy)+exp(-i*zeta)(gy-)]

  % y=(s,p,phi), y- = (s,p,-phi), phi=2*pi*k/N
  % K=Normalize constant
  % gy and gy- = Complex Garbor atoms , eq.(64)
  % zeta=phase, [0,2*pi[ , the complex phase of <Rf,gy>
  phi=2*pi*k/N;
  gy=CpxGAtom64(s,p,phi,n,N);
  gyneg=CpxGAtom64(s,p,-phi,n,N);

  % Compute the Normalize constant K, eq.(114)
    inner=gy*gyneg';
    inner=(exp(i*2*zeta))*inner;
    K=(sqrt(2))/(sqrt(1+real(inner)));
    gyr=K*((exp(i*zeta))*gy+(exp(-i*zeta))*gyneg)/2;
```

ข.2.3 ฟังก์ชันการค้นหาอะตอมจากดิกชันนารี

```
function [xo,gyo,inno,jpk]=sampD(f,n,N)    %Search for max|<f,g>|, and parameter s,p,k for newton start point
      % Sampling dictionary,search in sub-dictionary,in eq.(64)
      % sub-dictionary: y = (a^j,p[a^j]du,ka^[-j]dc) , a=2,du=0.5,de=pi
      %                        0<j<log2(N),0<=p<N2^(-j+1),0<=k<2^(j+1)
      % From theorem 2: y = (a^j,p[a^j]du,ka^[-j]de),for (j,p,k)E Z^3
      %                          du*de<2*pi
%-----------------------------------------------------------------------------
     a=2;   du=0.5;    de=pi;          inno=0;
     for jj=1:1:fix(log2(N))-1
        s=a^jj;
        for pp=0:1:(N*2^(-jj+1))-1
                 p=pp*s*du;
                 for kk=0:1:((2^(jj+1))-1)
                          phi=kk*de/s;
                          gy=CpxGAtom64(s,p,phi,n,N);
                          innN=abs(f*gy');
                          if innN>inno
                                   so=s;     po=p;          ko=kk*N/(s*2);
                                   jpk=[jj;pp;kk];     inno=innN;          gyo=gy;
                          end       % if innN
                 end        % for kk
         end        % for pp
      end           % for jj
      xo=[so;po;ko];

function [g,newtdisp,x,phii]=newton(f,n,N)       % 1. Search in sub-dictionary by 'sampD.m'
      % 2. Refind with a Newton search strategy to recover the time-frequency parameter
      %    that best match the signal components.
  %-----------------------------------------------------------------------------
  % Variable for used within function
      newtdisp=[];                         % display for searching any parameter
      newtloop=0;                          % for counting the newton loop
```

```
    del=0.00001;                    % used for finding the derivative
    kred=[];
%--------------------------------------------------------------------------
% search atom in sub-dictionary
    [xo,gyo,inno,jpk]=sampD(f,n,N);
    kred=[kred xo(3)];
% check for |grad| at this point
    Dx1=(inner(xo(1)+del,xo(2),xo(3),n,N,f)-inner(xo(1),xo(2),xo(3),n,N,f))/del;
    Dx2=(inner(xo(1),xo(2)+del,xo(3),n,N,f)-inner(xo(1),xo(2),xo(3),n,N,f))/del;
    Dx3=(inner(xo(1),xo(2),xo(3)+del,n,N,f)-inner(xo(1),xo(2),xo(3),n,N,f))/del;
    grado=[Dx1;Dx2;Dx3];
    slopeo=abs(sqrt(sum(grado.^2)));
    newtdisp=[newtdisp;newtloop inno xo(1) xo(2) xo(3)  angle(f*gyo') slopeo];
%--------------------------------------------------------------------------
% Set boundary for Newton search
    J=jpk(1,1);  P=jpk(2,1);   K=jpk(3,1);
    sl=2^(J-1/2);  pl=round((P-1)*(2^J)/2);  kl=round(N*(K-1)/(2*(2^J)));
    sh=2^(J+1/2);  ph=round((P+1)*(2^J)/2);  kh=round(N*(K+1)/(2*(2^J)));
        if sl<1;  sl=1;  end
        if sh>N;  sh=N;  end
        if pl<0;  pl=0;  end
        if ph>N;  ph=N;  end
        if kl<0;  kl=0;  end
        if kh>N;  kh=N;  end
% display the boudary of s,p,k
    fprintf('\n\nthe boundary of s,p,k:\n\n\t J     P     K\n')
    disp([J P K])
    fprintf('\n\t s     p     k\n')
    disp([[sl pl kl];xo';[sh ph kh]])
%--------------------------------------------------------------------------
% Newton search
    x=[xo(1);round(xo(2));round(xo(3))];
    error=0.001;                    % for check the slope (|grad|) in while loop
    slope=1;  inn1=inno;
```

```
while slope>error                          %for iterate=1:5 %
   newtloop=newtloop+1;
   % find the gradient
     Dx1=(inner(x(1)+del,x(2),x(3),n,N,f)-inner(x(1),x(2),x(3),n,N,f))/del;
     Dx2=(inner(x(1),x(2)+del,x(3),n,N,f)-inner(x(1),x(2),x(3),n,N,f))/del;
     Dx3=(inner(x(1),x(2),x(3)+del,n,N,f)-inner(x(1),x(2),x(3),n,N,f))/del;
     gradn=[Dx1;Dx2;Dx3];
   % find the hessian matrix
     x1=x(1)+del;  x2=x(2)+del;   x3=x(3)+del;
    H11=(((inner(x1+del,x(2),x(3),n,N,f)-inner(x1,x(2),x(3),n,N,f))/del)-Dx1)/del;
    H12=(((inner(x1,x2,x(3),n,N,f)-inner(x1,x(2),x(3),n,N,f))/del)-Dx2)/del;
    H13=(((inner(x1,x(2),x3,n,N,f)-inner(x1,x(2),x(3),n,N,f))/del)-Dx3)/del;
    H21=(((inner(x1,x2,x(3),n,N,f)-inner(x(1),x2,x(3),n,N,f))/del)-Dx1)/del;
    H22=(((inner(x(1),x2+del,x(3),n,N,f)-inner(x(1),x2,x(3),n,N,f))/del)-Dx2)/del;
    H23=(((inner(x(1),x2,x3,n,N,f)-inner(x(1),x2,x(3),n,N,f))/del)-Dx3)/del;
    H31=(((inner(x1,x(2),x3,n,N,f)-inner(x(1),x(2),x3,n,N,f))/del)-Dx1)/del;
    H32=(((inner(x(1),x2,x3,n,N,f)-inner(x(1),x(2),x3,n,N,f))/del)-Dx2)/del;
    H33=(((inner(x(1),x(2),x3+del,n,N,f)-inner(x(1),x(2),x3,n,N,f))/del)-Dx3)/del;
    H=[H11 H12 H13;H21 H22 H23;H31 H32 H33];
  % find the step newton
     d=-inv(H)*gradn;
  % find next point
     xn=x+d;
     xn(2)=round(xn(2));
     xn(3)=round(xn(3));

  % check for boundary of serching
     scale=10;       red=1;       intowhile=0;
     if ~isempty(find([(sl<xn(1))&(xn(1)<sh);...
              (pl<=xn(2))&(xn(2)<ph);...
              (kl<=xn(3))&(xn(3)<kh)]==0))
              for q=1:20
                     scale=scale-red;          dred=d*(scale/10);          xn=x+dred;
                     xn(2)=round(xn(2));       xn(3)=round(xn(3));         intowhile=1;
                     newtdisp=[newtdisp;newtloop 0 xn(1) xn(2) xn(3)  0 q];
```

```
                    if isempty(find([(sl<xn(1))&(xn(1)<sh);...
                                (pl<=xn(2))&(xn(2)<ph);...
                                (kl<=xn(3))&(xn(3)<kh)]==0))
                                break
                    end % if isempty
            end %for q=
    end %if ~isempty

    if intowhile==1
            d=dred;
            disp([' new s p k'])
            disp(xn')
    end %intowhile==

  % find complex atom,phii and |<f,gy>|
    gy=CpxGAtom64(xn(1),xn(2),2*pi*xn(3)/N,n,N);
    phii=angle(f*gy');      inn=abs(f*gy');
    if inn<inn1
            redstep=20;
            for trystep=(redstep-1):-1:0
                    tryd=d*(trystep/redstep);           xn=x+tryd;
                    xn(2)=round(xn(2));                 xn(3)=round(xn(3));
                    gy=CpxGAtom64(xn(1),xn(2),2*pi*xn(3)/N,n,N);
                     phii=angle(f*gy');           inn=abs(f*gy');
                    newtdisp=[newtdisp;newtloop inn xn(1) xn(2) xn(3)  phii trystep];
                    if inn>=inn1
                                x=xn;
                                break
                    end %inn>=
            end %for trystep
    else        x=x+d;        x(2)=round(x(2));         x(3)=round(x(3));
    end %if inn<

% set k,less than N/2
  kred=[kred x(3)];
  if x(3)>N/2
```

```
                x(3)=fix(N-x(3));       kred=[kred 4444 x(3)];
                gy=CpxGAtom64(x(1),x(2),2*pi*x(3)/N,n,N);
                phii=angle(f*gy');
        end %if x(3)

    Dx1=(inner(x(1)+del,x(2),x(3),n,N,f)-inner(x(1),x(2),x(3),n,N,f))/del;
    Dx2=(inner(x(1),x(2)+del,x(3),n,N,f)-inner(x(1),x(2),x(3),n,N,f))/del;
    Dx3=(inner(x(1),x(2),x(3)+del,n,N,f)-inner(x(1),x(2),x(3),n,N,f))/del;
    grad=[Dx1;Dx2;Dx3];
    slope=abs(sqrt(sum(grad.^2)));
    newtdisp=[newtdisp;newtloop inn x(1) x(2) x(3)  phii slope];

   % break if infgNew better infgf less than 0.5%
      if abs((inn-inn1)/inn1)<0.0005
                disp('xxxxxxxxxxxxxxxxxxxxxxxx')
                break
      else      inn1=inn;
      end  %if abs
   end % while error
%------------------------------------------------------------------------------
% find real atom
    g=ReGAtom113(x(1),x(2),x(3),phii,n,N);
% display k
    fprintf('\ndisplay k,for k>N/2 or not\n\n')
    disp(kred)


function h=inner(x1,x2,x3,n,N,f)     % Use for 3 variables (x1,x2,x3) optimization,Newton's method
        % This function is |<f,gy>|,use for Newton search
        % gy is Discrete atom in eq.(64)

        % create the Complex Gabor atom (Dictionary)
    phi=2*pi*x3/N;
    gy=CpxGAtom64(x1,x2,phi,n,N);                 % Discrete complex atoms in eq.(64)
    h=abs(f*gy');                                 % <f,gy>=sum[(f)(gy*)]
    % require h is maximum value,with Newton search
```

ข.2.4 ฟังก์ชันการลดอัตราการสุ่มตัวอย่าง

```
function [f,N]=redsamp(y,fs,df)        % Down the samples by data decimation
    % Butterword LPF:cut off freq.=1kHz,order 8th
        fc=1000;                       % cutoff frequency
        Y=butterlpf(8,fc,fs,y);        % Y is output Butterworth LPF
 %---------------------------------------------------------------------------
    % Select PCG for analysis
        Y=selectpcg(Y,fs);
        fprintf('\n\t Samples of data : %d \n',length(Y))
        n=0:length(Y)-1;    T=1/fs;
        figure,...
                plot(n,real(Y)),grid,...
                xlabel('samples(n)'),title('Before decimating')
        figure,...
                plot(n*T,real(Y)),grid,...
                xlabel('time(second)'),title('Before decimating')
 %---------------------------------------------------------------------------
    % Data decimation of the discrete-time signal
        f=decimate(Y,df);
        fprintf('\n\t Samples of data decimation: %d \n',length(f))
        N=length(f);        n=0:N-1;
```