%%
%%

%%% Supplementary Material to:

%%% Calderon, C. P., Martinez, J. G., Carroll, R. J. and Sorensen, D.
%%% C. (2010). P-splines using derivative information. *Multiscale*
%%% Modeling and Simulation, 8, 1562-1580.

%%
%%

NOTE: for users without access to MATLAB, this program can generate executables that can be run on most platforms without requiring this program. A free "mcrinstaller" program made available through Mathworks is needed to launch the executable and it is available online: http://www.mathworks.com/matlabcentral/fileexchange/5268 (check for newest version)

If you require this, email the Chris Calderon and provide information about the operating system you are usign and an attempt to generate the executable you need will be made.

The driver for the MATLAB script demonstrating methods in the paper are below:

PuDI_demo.m (shows how to construct design matrix)

PSQR_DpEqI_demo.m (solver for Pspline with any design matrix and ridge penalty similar to one used in text)

PSQR_demo.m (solver for general Pspline with any design matrix and penalty matrix)

To run, copy the code below and make *.m text files. Place these in a folder, launch matlab and add the aforementioned folder to path (or cd to this location) and execute scripts.

e.g. type "[F,Fhat]=PuDI_demo(1);" to generate scatterplot data F, fit P-spline using GCV and output Fhat and plot (use argument "0" to turn plot off).

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Start PuDI_demo.m

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [F,Fhat]=PuDI_demo(plotflag);

%function [F,Fhat]=PuDI_demo(plotflag);

%function illustrating how to setup PuDI design matrix and
efficiently find smoothing parameter.

%input: plotflag; set = 1 to plot and any other value to not
(function of interest is internally coded; modifications are easy to
implement)

%output: F vector contaiing the true value of (f,df) and Fhat are the
PuDI estimates.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%

%Obtain scatterplot data of function and its derivative

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%

n=40;

x=[0:15/(n-1):15]'; %set design grid

[f,df]=pudiegfunc(x);F=[f;df];


%contaminate obserations with N(0,WW') noise.

sigma2=1;c=4;

W=diag([ones(1,n)*sqrt(sigma2) ones(1,n)*sqrt(sigma2*c) ]);

noise=W*randn(2*n,1);

zOBS=[f+noise(1:n);df+noise(n+1:end)];
```

```
%use noise covariance to construct GLS observation vector "y"

y=W\zOBS;  %this implementation assumes W is not ill-conditioned.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Setup Design Matrix

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

K=20;p=2;  %pick number of knots

kknots=[15/K:15/K:15];  %use uniform grid or let quantiles determine knot location.


%create a vector of the design grid and knot sequence for use in design matrices

knotVEC=repmat(kknots,size(x,1),1);

xVEC=repmat(x,1,size(knotVEC,2));


X=[ones(size(x)) x x.^2 ;zeros(size(x)) ones(size(x)) 2*x]; %hard code in X consistent with "p" selected

Z=[((xVEC-knotVEC).*(xVEC-knotVEC>0)).^p; (((xVEC-knotVEC).^(p-1)).*(xVEC-knotVEC>0))*p];


%construct design matrix using GLS weight

C=W\[X Z];%this implementation assumes W is not ill-conditioned.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Factor the Design Matrix to Form where GCV Search is Fast

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
[Q,R,V,s] = PSQR_matfac_DpEqI(C,K);



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Select Smoothing Parameter via GCV and Vectorized Solves Facilitating Brute Force Grid Search
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
alphaV=logspace(-14,14,1000); %select candidate smoothing parameters
[RSS] = PSQR_RSS(y,Q,R,V,s,K,sqrt(alphaV));
[GCV]=PSQR_GCV(RSS,Q,R,V,s,C,sqrt(alphaV),K,y);

%find the optimal smoothing parameter
[mv mi]=min(GCV);
alphaHAT=alphaV(mi);

%find the solution corresponding to alphahat
[RSS,beta] = PSQR_SolSteps_DpEqI(y,Q,R,V,s,K,sqrt(alphaHAT));
Fhat=[X Z]*beta; %multiply the estimated vector by the mixed model in the original coordinates

if(plotflag==1)
%plot results
figure;hold on;
plot(x,zOBS(1:n),'ko');plot(x,Fhat(1:n),'r--');plot(x,F(1:n));
xlabel('x','fontsize',18),ylabel('f(x)','fontsize',18);legend('ScatterPlot','P-Spline Estimate','True Function')
set(gca,'fontsize',14)
```

```
figure;hold on;

plot(x,zOBS(1+n:end),'ko');plot(x,Fhat(1+n:end),'r--
');plot(x,F(1+n:end));

xlabel('x','fontsize',18),ylabel('\partial
f(x)','fontsize',18);legend('ScatterPlot','P-Spline Estimate','True
Function')

set(gca,'fontsize',14)

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%

function [y,dy]=pudiegfunc(x)

y= -(2.5*sin(2*pi*x*1/15*6)+exp(-(x-3).^2/1/2)*9.0+exp(-(x-
10).^2/1/2)*5.0-2)/3;

dy= -(2.5*cos(2*pi*x*1/15*6)*2*pi*1/15*6 ...

                    +exp(-(x-3).^2/1/2)*9.0.*-(x-3) +...

                                            -exp(-(x-
10).^2/1/2)*5.0.*(x-10) )/3;

y=y(end:-1:1);

dy=-dy(end:-1:1); %the function is being reflected about 0, so
account for trivial transform y=-x

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% End PuDI_demo.m

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Start PSQR_DpEqI_demo.m

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%PSQR_DpEqI_demo.m
%example illustrating how to use and test scripts for special case
where
%the penalty matrix D^P=I

%obtain design matrix and observation vector
m=10;n=5;
C=randn(m,n);
y=randn(m,1);

nP=3;
Dp=eye(nP);
alphaV=[2 4];  %specify a vector containing  the smoothing parameter
values to be computed.

% do the matrix factorization once
[Q,R,V,s] = PSQR_matfac_DpEqI(C,nP);
% now obtain the vectorized solve (most applications only require
RSS,
% comment out "x solve" to speed up code below).
 [RSS,x] = PSQR_SolSteps_DpEqI(y,Q,R,V,s,nP,sqrt(alphaV));
% do a vectorized solve for various statistical  quantites.
[GCV,AICc,dfres,sigma2HAT,SmootherTrace]=PSQR_STATS_DpEqI(RSS,Q,R,V,s
,C,sqrt(alphaV),nP,y);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%
%Compare Solution to standard Algorithm
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%
%now find the same parameter beta using unmodified PSQR Algorithm
%first do matrix factorizations that only need to be carried out once
for given C.
[Q,R,V] = PSQR_matfac(C,nP);

%now solve using different candidate Dp or alpha
for i=1:max(size(alphaV))
[betaPSQR] = PSQR_SolSteps(y,Q,R,V,nP,sqrt(alphaV(i)),Dp);
%compare differences
 diff(i)=norm(x(:,i)-betaPSQR);
end
mess='check workspace in matlab [type "whos"] to see various
quantites compted'
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% End PSQR_DpEqI_demo.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Start PSQR_demo.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%PSQR_demo.m
%example illustrating how to use and test scripts
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%
%Define Design and Penalty Matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%
%obtain design matrix and observation vector
m=10;n=5;
C=randn(m,n);
y=randn(m,1);

nP=3;
Dp=randn(nP,nP);Dp=sqrtm(Dp'*Dp);  %construct a random positive
semidefinite matrix
alpha=4;  %specify the smoothing parameter.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%
%Execute PSQR Algorithm for two different \alpha values
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%
%now find the same parameter beta using PSQR Algorithm

%first do matrix factorizations that only need to be carried out once
for given C.
[Q,R,V] = PSQR_matfac(C,nP);
%now solve using different candidate Dp or alpha
[betaPSQR] = PSQR_SolSteps(y,Q,R,V,nP,sqrt(alpha),Dp);

%solve using existing matrix factorizations  using new candidate Dp
or alpha
[betaPSQR_smooth2] = PSQR_SolSteps(y,Q,R,V,nP,sqrt(alpha*2),Dp);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%
%Diagnostic Checks

%find the least squares solution using standard QR and compare result
in order  compare to
%PSQR
D=[zeros(n,n-nP) [zeros(n-nP,nP) ; Dp] ];  %add the zero columns to
penalty matrix
[Q R]=qr([C; sqrt(alpha)*D],0);
b=Q(1:m,:)'*y;betaQR_REF=R\b;
%find with normal equations
betaNORMALEQN=(C'*C+alpha*D'*D)\(C'*y);
%compare differences
 diff1=norm(betaQR_REF-betaPSQR)
 diff2=norm(betaNORMALEQN-betaPSQR)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% End PSQR_demo.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Start PSQR_GCV.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [GCV]=PSQR_GCV(RSS,Q,R,V,s,wC,muVEC,k,wy)
%
%function [costfuncVEC]=QR_pudi_COSTFUNC(Q,R,V,wC,muVEC)
%the wC and wy denote the weighted versions of the design mat and
observation vector (respectively).  it
%is assumed that [Q,R,V] come from "PSQR_matfac_DpEqId.m" where the
weighted design mat was used.
%
%

m=size(Q,1);

[trVEC, tr2VEC] = traceVEC_DpEqI(s,size(wC,2),muVEC);


% NOTE:GCV bekiw yses RSS instead of RSS/m
GCV=((RSS)./((1-trVEC/m).^2));


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% End PSQR_GCV.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Start PSQR_matfac.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [Q,R,V] = PSQR_matfac(C,k);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%function [Q,R,V] = PSQR_matfac(C,k);
%
%   Input:   C   an m by n matrix
%
%            k   a pos integer between 0 and n.  the first penalized
term should be in column n-k+1 of C.
%
%   Output:  Q   an m by n orthogonal matrix
%
%            R   an n by n upper triangular matrix
%                with lower right k by k block
%                non-negative  diagonal
%
%            V   k by k orthogonal vector
%
%   Results in  C = Q R V_1'  where V_1 = [[ I  0];
%                                         [ 0  V]]
%

     [Q,R] = qr(C,0);
     [m,n] = size(C);

     p = n-k+1;


      [U,S,V] = svd(R(p:n,p:n));

     Q(:,p:n) = Q(:,p:n)*U;
     R(:,p:n) = R(:,p:n)*V;
     R(p:n,p:n) = S;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% End PSQR_matfac.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Start PSQR_matfac_DpEqI.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [Q,R,V,s] = PSQR_matfac_DpEqI(C,k);
%function [Q,R,V,s] = PSQR_matfac_DpEqI(C,k);
%%Carries out matrix factorization for special case D^P=I.
%   Input:   C   an m by n matrix
%
%            k   a pos integer between 0 and n
%
%   Output:  Q   an m by n orthogonal matrix
%
%            R   the first n-k rows of an n by n upper triangular
matrix
%                with lower right k by k block
%                non-negative  diagonal
%
%            s   the diagonal of the kxk block mentioned above
%
%            V   k by k orthogonal
%
%
%   Results in  C = Q R V_1'  where V1 = [ I  0]
%                                        [ 0  V]
%
     [Q,R] = qr(C,0);
     [m,n] = size(C);

     p = n-k+1;

     Repstol=1/eps/10;
     condCHECK=cond(R(1:p-1,1:p-1));
     if(condCHECK>Repstol)
         message='WARNING:  Poor design matrix used.  Problem is not
with Z^P, but with the "free"/unpenalized portion of the matrix.'
         message=['Results obtained likely meaningless due to
roundoff errors / log(condition number) = ' num2str(log(condCHECK))]
         message='Hit any key to proceed  anyway (or CNTRL-C to
quit)'
         pause
     end

     [U,S,V] = svd(R(p:n,p:n),0);

     Q(:,p:n) = Q(:,p:n)*U;
     R(1:n-k,p:n) = R(1:n-k,p:n)*V;
     R = R(1:n-k,:);
     s = diag(S);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% End PSQR_matfac_DpEqI.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Start PSQR_RSS.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [RSS] = PSQR_RSS(b,Q,R,V,s,k,mu1);
%function [RSS] = PSQR_RSS(b,Q,R,V,s,k,mu1);
%Finds vector of RSS corresponding vector of smoothing parameters mu1
for special case D^P=I.
%
%
%    Input:  Q   an m by n orthogonal matrix
%
%            R   the first n-k rows of an
%                n by n upper triangular matrix
%                with lower right k by k block
%                a  non-negative  diagonal matrix
%                stored in the vector s
%
%
%            V   k by k orthogonal SVD factor (or Identity)
%
%            s   a k-vector with diag(s) the lower
%                k by k block of R
%
%            b   an m vector
%
%            k   a pos integer between 0 and n
%
%            mu1 a positive scalar (or column vector in vectorized
version)
%
%            Assumes these quantities computed by PSQR_matfac_DpEqI.m

     [m,n] = size(Q);
     z = zeros(k,1);

     nmk = n-k;
     p = nmk+1;

     nrep=size(mu1,2);
     mu1=repmat(mu1,size(s,1),1);
     s=repmat(s,1,nrep);
```

```
     rho =  s.*s + (mu1.*mu1);
%
%    Note: this could potentially overflow
%          unnecessarily.  see literature on Givens rotations  for
methods dealing with this case
%
     x=0;
     c = [Q'*b; z];
     c=repmat(c,1,nrep);

     c(p:n,:) = (s.*c(p:n,:) + c(n+1:n+k,:).*mu1)./rho;

     %vectorized solve is not necessary in most applications, e.g.
often
     %just many RSS values are  needed
     % (keep lines below to show how batch solve is possible...others
apps may utilize this)
     %x = [R(1:nmk, 1:nmk)\(c(1:nmk,:) - R(1:nmk,p:n)*c(p:n,:));
V*c(p:n,:)];

%   Compute RSS in a "vectorized" fashion
%   using orthogonality  of [C ; mu1*D]  to   [b;0]-[C; mu1*D]*x
%
%
%    simple relation xx:=[C;mu1*D];  yy:=[b-Cx; 0 - mu1*D*x]....then
inner product <xx,yy> is zero by definition at minimizer \hat{x}.
%    i.e. setting the cost function gradient equal to zero
%
%       norm(b)^2 - (norm([C ; mu1*D])^2 + norm(mu1*D*x)^2) = norm(b
- Cx)^2
%
%        beta^2   -                   beta1^2
%

%bottom of b1 contains constraint informaton.
     b1 = [c(1:nmk,:); s.*c(p:n,:); c(p:n,:)*sqrt(2).*mu1];

     beta = norm(b);

     beta1 = sqrt(sum(b1.^2));
     RSS=( (beta + beta1).*(beta - beta1));     %residual sum of
squares


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% End PSQR_RSS.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Start PSQR_SolSteps.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [x] = PSQR_SolSteps(y,Q,R,V,k,mu1,sqrtD);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%function [x] = PSQR_SolSteps(b,Q,R,V,k,mu1);
%    Input:  Q   an m by n orthogonal matrix
%
%            R   an n by n upper triangular matrix
%                with lower right k by k block
%                non-negative  diagonal
%
%            V   k by k orthogonal
%
%            y   obervation vector
%
%            k   a pos integer between 0 and n
%
%            mu1 a vector containing positive scalar smoothing
parameters
%
%           sqrtD is the Cholesky factor of the penalty term =
mu1^2*(sqrtD)'sqrtD
%
%           x   is a matrix of P-spline coefficients corresponding to
the mu1 vector and the proposed sqrtD
%
%            Assumes these quantities computed by PSQR_matfac.m


     [m,n] = size(Q);
     z = zeros(k,1);

     nmk = n-k;
     p = nmk+1;
     ck = [Q'*y; z];

     nvec=size(mu1,2); %assumes each column of mu1 is a new problem
(in general vector mu case, store as kxnvec)
     x=[];
    for i=1:nvec
     c=ck;

     [W,Rw] = qr([R(p:n,p:n) ; sqrtD*V*mu1(i)],0); %mu1 must be a k
vector corresponding to sqrt(D(\lambda))

     %debug=diag(mu1(:,i))*V
```

```
     c(p:n) = Rw\W'*c(p:n+k);

     xi = [R(1:nmk, 1:nmk)\(c(1:nmk) - R(1:nmk,p:n)*c(p:n));
V*c(p:n)];
     x=[x xi];
    end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% End PSQR_SolSteps.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Start PSQR_SolSteps_DpEqI.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [RSS,x] = PSQR_SolSteps_DpEqI(b,Q,R,V,s,k,mu1);
%function [RSS] = PSQR_matfac_DpEqI(b,Q,R,V,s,k,mu1);
%Carries out matrix factorization for special case D^P=I.
%
%
%    Input:  Q   an m by n orthogonal matrix
%
%            R   the first n-k rows of an
%                n by n upper triangular matrix
%                with lower right k by k block
%                a  non-negative  diagonal matrix
%                stored in the vector s
%
%
%            V   k by k orthogonal SVD factor (or Identity)
%
%            s   a k-vector with diag(s) the lower
%                k by k block of R
%
%            b   an m vector
%
%            k   a pos integer between 0 and n
%
%            mu1 a positive scalar (or column vector in vectorized
version)
%
%            Assumes these quantities computed by PSQR_matfac_DpEqI.m


     [m,n] = size(Q);
```

```
     z = zeros(k,1);

     nmk = n-k;
     p = nmk+1;


     nrep=size(mu1,2);
     mu1=repmat(mu1,size(s,1),1);
     s=repmat(s,1,nrep);



     rho =  s.*s + (mu1.*mu1);
%
%    Note: this could potentially overflow
%          unnecessarily.  see literature on Givens rotations  for
methods dealing with this case
%



     x=0;
     c = [Q'*b; z];
     c=repmat(c,1,nrep);

     c(p:n,:) = (s.*c(p:n,:) + c(n+1:n+k,:).*mu1)./rho;

     %vectorized solve is not necessary in most applications, e.g.
often
     %just many RSS values are  needed
     % (keep lines below to show how batch solve is possible...others
apps may utilize this)
     x = [R(1:nmk, 1:nmk)\(c(1:nmk,:) - R(1:nmk,p:n)*c(p:n,:));
V*c(p:n,:)];

%   Compute RSS in a "vectorized" fashion
%   using orthogonality  of [C ; mu1*D]  to   [b;0]-[C; mu1*D]*x
%
%
%    simple relation xx:=[C;mu1*D];  yy:=[b-Cx; 0 - mu1*D*x]....then
inner product <xx,yy> is zero by definition at minimizer \hat{x}.
%    i.e. setting the cost function gradient equal to zero
%
%       norm(b)^2 - (norm([C ; mu1*D])^2 + norm(mu1*D*x)^2) = norm(b
- Cx)^2
%
%        beta^2   -                 beta1^2
%

%bottom of b1 contains constraint informaton.
     b1 = [c(1:nmk,:); s.*c(p:n,:); c(p:n,:)*sqrt(2).*mu1];

     beta = norm(b);
```

```
     beta1 = sqrt(sum(b1.^2));
     RSS=( (beta + beta1).*(beta - beta1));     %residual sum of
squares



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% End PSQR_SolSteps_DpEqI.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%









%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Start PSQR_STATS_DpEqI.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function
[GCV,AICc,dfres,sigma2HAT,SmootherTrace]=PSQR_STATS_DpEqI(RSS,Q,R,V,s
,wC,muVEC,k,wy)
%
%function [costfuncVEC]=QR_pudi_COSTFUNC(Q,R,V,wC,muVEC)
%the wC and wy denote the weighted versions of the design mat and
observation vector (respectively).  it
%is assumed that [Q,R,V] come from "PSQR_matfac_DpEqId.m" where the
weighted design mat was used.
%
%



m=size(Q,1);



[trVEC, tr2VEC] = traceVEC_DpEqI(s,size(wC,2),muVEC);
SmootherTrace=trVEC;



% NOTE:GCV bekiw yses RSS instead of RSS/m
```

```
GCV=((RSS)./((1-trVEC/m).^2));

AICc =log(RSS)+2*(trVEC+1)./(m-trVEC-2);

dfres=m-2*trVEC+tr2VEC;

sigma2HAT=RSS./dfres;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% End PSQR_STATS_DpEqI.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Start traceVec_DpEqI.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [tr, tr2] = traceVEC_DpEqI(s,n,mu1);
%function [tr, tr2] = traceVEC_DpEqI(s,n,mu1);
%
%Returns the trace of the smoother matrix and the trace of the
"square" of smoother
%    Input:  s   the diagonal (non-negative) of
%                the lower right k by k block
%                an n by n upper triangular matrix
%                R obtained from qrfacR(C,k)
%
%            n   the number of columns of C
%
%            mu1 a positive scalar (or column vector in vectorized
version)
%
%            Assumes these quantities computed by qrfacR.m
%
%   Output:  tr=trace(SM_mu1) ;tr2=trace( SM_mu1'*SM_mu1)
%
%            where SM_mu1=C*((C'*C + mu1*mu1*D)\C');C = QRV'
```

```
    [k] = length(s);

     s = s.*s;


      if(size(mu1,1)==1)
       nvec=size(mu1,2);
       s=repmat(s,1,nvec);
       muV=repmat(mu1,size(s,1),1);
      else
        muV=mu1; %in case where mu1 is a scalar, repmat step above
isn't needed
       end

      z = s./(muV.^2 + s);


     tr = sum(z); %contribution from penalizd terms to trace of
"smoothing matrix"
     tr = n-k + tr; %trace of "smoothing matrix" with all terms
included
     tr2= n-k + sum(z.^2); %trace of "square of smoother matrix"

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% End traceVec_DpEqI.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```