APPENDIX A– Matlab Code

```
function [net,tr,Ac,El] = trainwithimprovedbr(net,Pd,Tl,Ai,Q,TS,VV,TV)
% This Matlab Code is an improved and modified version of standard
% procedure commonly used for Bayesian Regularization backpropagation
% in Matlab software.
%
% This improved and modified version is in particular intended and very
% useful for making NN fatigue life prediction utilizing limited data
% examples. Nevertheless, this code should work as well for similar or
% related applications.
%
% This improved version gives more stable and consistent approximation
% results compared to the standard one and has been verified to
% benchmark problem of sinus wave with noise.
%
% Example
%
%     Here is a problem consisting of inputs p and targets t that we
%     would like to solve with a network.  It involves fitting a noisy
%     sine wave.
%
%       p = [-1:.05:1];
%       t = sin(2*pi*p)+0.1*randn(size(p));
%
%     Here a two-layer feed-forward network is created.  The network's
%     input ranges from [-1 to 1].  The first layer has 20 TANSIG
%     neurons, and the second layer has one PURELIN neuron.  The
%     TRAINWITHIMPROVEDBR network training function is to be used. The
%     plot of the resulting network output should show a smooth and
%     stable response, without overfitting.
%
%       % Create a Network
%  net=newff([-1 1],[20,1],{'tansig','purelin'},'trainwithimprovedbr');
%
%       % Train and Test the Network
%       net.trainParam.epochs = 70;
%       net.trainParam.show = 10;
%       net = train(net,p,t);
%       a = sim(net,p)
%       figure
%       plot(p,a,p,t,'+')
%
%  Syntax
%
%    [net,tr,Ac,El] = trainwithimprovedbr (net,Pd,Tl,Ai,Q,TS,VV,TV)
%    info = trainwithimprovedbr (code)
%
%  Description
%
%    TRAINWITHIMPROVEDBR is a network training function that updates
%     the weight and bias values according to Levenberg-Marquardt
%     optimization. It minimizes a combination of squared errors and
%     weights and, then determines the correct combination so as to
%     produce a network which generalizes well.  The process is called
%     Bayesian regularization.
%
%    TRAINWITHIMPROVEDBR (NET,Pd,Tl,Ai,Q,TS,VV,TV) takes these inputs,
%      NET - Neural network.
%      Pd  - Delayed input vectors.
```

```
%      Tl  - Layer target vectors.
%      Ai  - Initial input delay conditions.
%      Q   - Batch size.
%      TS  - Time steps.
%      VV  - Either empty matrix [] or structure of validation vectors.
%      TV  - Either empty matrix [] or structure of test vectors.
%    and returns,
%      NET - Trained network.
%      TR  - Training record of various values over each epoch:
%            TR.epoch - Epoch number.
%            TR.perf - Training performance.
%            TR.vperf - Validation performance.
%            TR.tperf - Test performance.
%            TR.mu - Adaptive mu value.
%      Ac  - Collective layer outputs for last epoch.
%      El  - Layer errors for last epoch.
%
%    Training occurs according to the TRAINLM's training parameters,
%    shown here with their default values:
%      net.trainParam.epochs     100  Maximum number of epochs to train
%      net.trainParam.goal         0  Performance goal
%      net.trainParam.mu       0.005  Marquardt adjustment parameter
%      net.trainParam.mu_max   1e-10  Maximum value for mu
%      net.trainParam.max_fail     5  Maximum validation failures
%      net.trainParam.mem_reduc    1  Factor to use for memory/speed
%                                     trade off.
%      net.trainParam.min_grad 1e-10  Minimum performance gradient
%      net.trainParam.show        25  Epochs between displays (NaN for
%                                     no displays)
%      net.trainParam.time       inf  Maximum time to train in seconds
%
%    Dimensions for these variables are:
%      Pd - NoxNixTS cell array, each element P{i,j,ts} is a DijxQ
%           matrix.
%      Tl - NlxTS cell array, each element P{i,ts} is a VixQ matrix.
%      Ai - NlxLD cell array, each element Ai{i,k} is an SixQ matrix.
%    Where
%      Ni = net.numInputs
%      Nl = net.numLayers
%      LD = net.numLayerDelays
%      Ri = net.inputs{i}.size
%      Si = net.layers{i}.size
%      Vi = net.targets{i}.size
%      Dij = Ri * length(net.inputWeights{i,j}.delays)
%
%    If VV is not [], it must be a structure of validation vectors,
%      VV.PD - Validation delayed inputs.
%      VV.Tl - Validation layer targets.
%      VV.Ai - Validation initial input conditions.
%      VV.Q  - Validation batch size.
%      VV.TS - Validation time steps.
%    which is used to stop training early if the network performance
%    on the validation vectors fails to improve or remains the same
%    for MAX_FAIL epochs in a row.
%
%    If TV is not [], it must be a structure of validation vectors,
%      TV.PD - Validation delayed inputs.
%      TV.Tl - Validation layer targets.
%      TV.Ai - Validation initial input conditions.
%      TV.Q  - Validation batch size.
%      TV.TS - Validation time steps.
%    which is used to test the generalization capability of the
```

```
%    trained network.
%
%    TRAINWITHIMPROVEDBR (CODE) returns useful information for each
%    CODE string:
%      'pnames'    - Names of training parameters.
%      'pdefaults' - Default training parameters.
%
%  Network Use
%
%    You can create a standard network that uses TRAINWITHIMPROVEDBR
%    with NEWFF, NEWCF, or NEWELM.
%
%    To prepare a custom network to be trained with
%    TRAINWITHIMPROVEDBR:
%    1) Set NET.trainFcn to 'trainlm'.
%       This will set NET.trainParam to TRAINWITHIMPROVEDBR's default
%       parameters.
%    2) Set NET.trainParam properties to desired values.
%
%    In either case, calling TRAIN with the resulting network will
%    train the network with TRAINWITHIMPROVEDBR.
%
%    See NEWFF, NEWCF, and NEWELM for examples.
%
%  Algorithm
%
%    TRAINWITHIMPROVEDBR can train any network as long as its weight,
%    net input,and transfer functions have derivative functions.
%
%     Bayesian regularization minimizes a linear combination of squared
%     errors and weights.  It also modifies the linear combination
%     so that at the end of training the resulting network has good
%     generalization qualities.
%
%     See MacKay (Neural Computation, vol. 4, no. 3, 1992, pp. 415-447)
%     and Foresee and Hagan (Proceedings of the International Joint
%     Conference on Neural Networks, June, 1997) for more detailed
%     discussions of Bayesian regularization.
%
%     This Bayesian regularization takes place within the Levenberg-
%     Marquardt algorithm. Backpropagation is used to calculate the
%     Jacobian jX of performance PERF with respect to the weight and
%     bias variables X. Each variable is adjusted according to
%     Levenberg-Marquardt,
%
%      jj = jX * jX
%      je = jX * E
%      dX = -(jj+I*mu) \ je
%
%    where E is all errors and I is the identity matrix.
%
%    The adaptation of MU value is controlled by the predicted
%    decreased term wrt change in performance value shown below:
%     L = (-dX'*(beta*je+alph*X)) + dX'*dX*mu;
%    The update rule is:
%     if (perf-perf2)>(0.75*L),mu = mu/2;
%     elseif (perf-perf2)<=(0.25*L),mu = 2*mu;
%
%    The parameter MEM_REDUC indicates how to use memory and speed to
%    calculate the Jacobian jX.  If MEM_REDUC is 1, then TRAINLM runs
%    the fastest, but can require a lot of memory. Increasing MEM_REDUC
%    to 2 cuts some of the memory required by a factor of two, but
```

```
%    slows TRAINLM somewhat.  Higher values continue to decrease the
%    amount of memory needed and increase the training times.
%
%    Training stops when any of these conditions occur:
%
%    1) The maximum number of EPOCHS (repetitions) is reached.
%    2) The maximum amount of TIME has been exceeded.
%    3) Performance has been minimized to the GOAL.
%    4) The performance gradient falls below MINGRAD.
%    5) MU exceeds MU_MAX.
%    6) Validation performance has increase more than MAX_FAIL times
%       since the last time it decreased (when using validation).
%
%  See also NEWFF, NEWCF, TRAINGDM, TRAINGDA, TRAINGDX, TRAINLM,
%           TRAINRP, TRAINCGF, TRAINCGB, TRAINSCG, TRAINCGP,
%           TRAINBFG.
%
% Standard version: Copyright 1992-2005 The MathWorks, Inc.
% ($Revision: 1.1.6.2 $ $Date: 2005/12/22 18:20:52 $)
% Improved version: by Mas Irfan P. Hidayat (1st revision 2008)


% FUNCTION INFO
% =============

if isstr(net)
  switch (net)
    case 'pnames',
    net = {'epochs','show','goal','time','min_grad','max_fail', ...
             'mem_reduc','mu','mu_dec','mu_inc','mu_max'};
    case 'pdefaults',
    trainParam.epochs = 200;
    trainParam.show = 25;
    trainParam.goal = 0;
    trainParam.time = inf;
    trainParam.min_grad = 1e-10;
    trainParam.max_fail = 5;
    trainParam.mem_reduc = 1;
    trainParam.mu = 0.005;
    trainParam.mu_max = 1e10;
    net = trainParam;
    % Command to get default gradient function
    case 'gdefaults',
       % Pd contains information about a dynamic (~=0) or static (==0)
network
       if Pd ==0
          net='calcjx';
       else
          net='calcjxfp';
       end
    otherwise,
    error('Unrecognized code.')
  end
  return
end

% CALCULATION
% ===========
```

```
% Constants
this = 'TRAINWITHIMPROVEDBR';
epochs = net.trainParam.epochs;
goal = net.trainParam.goal;
max_fail = net.trainParam.max_fail;
mem_reduc = net.trainParam.mem_reduc;
min_grad = net.trainParam.min_grad;
mu = net.trainParam.mu;
mu_max = net.trainParam.mu_max;
show = net.trainParam.show;
time = net.trainParam.time;
gradientFcn = net.gradientFcn;
net.performFcn = 'sse';
doValidation = ~isempty(VV);
doTest = ~isempty(TV);

% Parameter Checking
if (~isa(epochs,'double')) | (~isreal(epochs)) | (any(size(epochs)) ~=
1) | ...
  (epochs < 1) | (round(epochs) ~= epochs)
  error('Epochs is not a positive integer.')
end
if (~isa(goal,'double')) | (~isreal(goal)) | (any(size(goal)) ~= 1) |
...
  (goal < 0)
  error('Goal is not zero or a positive real value.')
end
if (~isa(max_fail,'double')) | (~isreal(max_fail)) |
(any(size(max_fail)) ~= 1) | ...
  (max_fail < 1) | (round(max_fail) ~= max_fail)
  error('Max_fail is not a positive integer.')
end
if (~isa(mem_reduc,'double')) | (~isreal(mem_reduc)) |
(any(size(mem_reduc)) ~= 1) | ...
  (mem_reduc < 1) | (round(mem_reduc) ~= mem_reduc)
  error('Mem_reduc is not a positive integer.')
end
if (~isa(min_grad,'double')) | (~isreal(min_grad)) |
(any(size(min_grad)) ~= 1) | ...
  (min_grad < 0)
  error('Min_grad is not zero or a positive real value.')
end
if (~isa(mu,'double')) | (~isreal(mu)) | (any(size(mu)) ~= 1) | ...
  (mu <= 0)
  error('Mu is not a positive real value.')
end
if (~isa(mu_max,'double')) | (~isreal(mu_max)) | (any(size(mu_max)) ~=
1) | ...
  (mu_max <= 0)
  error('Mu_max is not a positive real value.')
end
if (mu > mu_max)
  error('Mu is greater than Mu_max.')
end
if (~isa(show,'double')) | (~isreal(show)) | (any(size(show)) ~= 1) |
...
  (isfinite(show) & ((show < 1) | (round(show) ~= show)))
  error('Show is not ''NaN'' or a positive integer.')
end
if (~isa(time,'double')) | (~isreal(time)) | (any(size(time)) ~= 1) |
...
  (time < 0)
```

```
  error('Time is not zero or a positive real value.')
end

% Initialize
flag_stop = 0;
stop = '';
startTime = clock;
X = getx(net);
numParameters = length(X);
ii = sparse(1:numParameters,1:numParameters,ones(1,numParameters));
[ssE,El,Ac,N,Zb,Zi,Zl] = calcperf(net,X,Pd,Tl,Ai,Q,TS);
if (doValidation)
  VV.net = net;
  vperf = calcperf(net,X,VV.Pd,VV.Tl,VV.Ai,VV.Q,VV.TS);
  VV.perf = vperf;
  VV.numFail = 0;
end
tr = newtr(epochs,'perf','vperf','tperf','mu','gamk','ssX','gradient');

% Initialize regularization parameters
numErrors = 0;
for i=1:size(El,1)
  for j=1:size(El,2)
    numErrors = numErrors + prod(size(El{i,j}));
  end
end
gamk = numParameters;
if ssE==0,
    beta = 1;
else
  beta = (numErrors - gamk)/(2*ssE);
end
if beta<=0,
  beta=1;
end
ssX = X'*X;
alph = gamk/(2*ssX)*numErrors;
perf = beta*ssE + alph*ssX;

% Train
for epoch=0:epochs

  % Jacobian
  [je,jj,normgX]=calcjejj(net,Pd,Zb,Zi,Zl,N,Ac,El,Q,TS,mem_reduc);

  % Training Record
  epochPlus1 = epoch+1;
  tr.perf(epochPlus1) = ssE;
  tr.mu(epochPlus1) = mu;
  tr.gamk(epochPlus1) = gamk;
  tr.ssX(epochPlus1) = ssX;
  tr.gradient(epochPlus1) = normgX;
  tr.alph(epochPlus1) = alph;
  tr.beta(epochPlus1) = beta;
  if (doValidation)
    tr.vperf(epochPlus1) = vperf;
  end
  if (doTest)
    tr.tperf(epochPlus1) =
    calcperf(net,X,TV.Pd,TV.Tl,TV.Ai,TV.Q,TV.TS);
  end
```

```
% Stopping Criteria
currentTime = etime(clock,startTime);
if (ssE <= goal)
  stop = 'Performance goal met.';
elseif (epoch == epochs)
  stop = 'Maximum epoch reached.';
elseif (currentTime > time)
  stop = 'Maximum time elapsed.';
elseif (normgX < min_grad)
  stop = 'Minimum gradient reached.';
elseif (mu > mu_max)
  stop = 'Maximum MU reached.';
elseif (doValidation) & (VV.numFail > max_fail)
  stop = 'Validation stop.';
elseif flag_stop
  stop = 'User stop.';
end

% Progress
if isfinite(show) & (~rem(epoch,show) | length(stop))
  % We present training function, search function and gradient
    function
  fprintf('%s%s%s',this,'-',gradientFcn);
  if isfinite(epochs) fprintf(', Epoch %g/%g',epoch, epochs); end
  if isfinite(time) fprintf(', Time %g%%',currentTime/time/100); end
  if isfinite(goal) fprintf(', %s
     %g/%g',upper(net.performFcn),ssE,goal); end
  if isfinite(goal) fprintf(', SSW %g',ssX); end
  if isfinite(min_grad) fprintf(', Grad
     %4.2e/%4.2e',normgX,min_grad); end
  if isfinite(numParameters) fprintf(', #Par
     %4.2e/%g',gamk,numParameters); end
  fprintf('\n')
  flag_stop = plotbr(tr,this,epoch);
  if length(stop) fprintf('%s, %s\n\n',this,stop); end
end
if length(stop), break; end

% APPLY LEVENBERG MARQUARDT: INCREASE MU TILL ERRORS DECREASE
while (epoch <= net.trainParam.epochs & mu <= mu_max & normgX >=
    net.trainParam.min_grad)
  % CHECK FOR SINGULAR MATRIX
  [msgstr,msgid] = lastwarn;
  lastwarn('MATLAB:nothing','MATLAB:nothing')
  warnstate = warning('off','all');
  dX = -(beta*jj + ii*(mu+alph)) \ (beta*je + alph*X);
  [msgstr1,msgid1] = lastwarn;
  flag_inv = isequal(msgid1,'MATLAB:nothing');
  if flag_inv, lastwarn(msgstr,msgid); end;
  warning(warnstate);
  X2 = X + dX;
  ssX2 = X2'*X2;
  net2 = setx(net,X2);

  [ssE2,E2,Ac2,N2,Zb2,Zi2,Zl2] = calcperf(net2,X2,Pd,Tl,Ai,Q,TS);

  EEEE = E2{2,1}(1,:);
  perf2 = beta*ssE2 + alph*ssX2;

  L = (-dX'*(beta*je+alph*X)) + dX'*dX*mu;
```

```
    if (perf-perf2)>(0.75*L),
        mu = mu/2;
    elseif (perf-perf2)<=(0.25*L),
        mu = 2*mu;
    break
    end

    if (perf2 < perf) & ( ( sum(isinf(dX)) + sum(isnan(dX)) ) == 0 ) &&
      flag_inv
      X = X2; net = net2; Zb = Zb2; Zi = Zi2; Zl = Zl2;
      N = N2; Ac = Ac2; El = E2;

    if (mu <= mu_max)
    % Update regularization parameters and performance function
    warnstate = warning('off','all');
    gamk = numParameters - alph*trace(inv(beta*jj+ii*alph));
    warning(warnstate);
        if ssX==0,
            alph = 1;
        else
            alph = gamk/(2*(ssX));
        end
        if ssE==0,
            beta = 1;
        else
            beta = (numErrors - gamk)/(2*ssE);
        end
        EEE = El{2,1}(1,:);
        perf = beta*EEE*EEE' + alph*X'*X;
        ssE = (EEE*EEE')*mu;

    % Validation
        if (doValidation)
            vperf = calcperf(net,X,VV.Pd,VV.Tl,VV.Ai,VV.Q,VV.TS);
            if (vperf < VV.perf)
                VV.perf = vperf; VV.net = net; VV.numFail = 0;
            elseif (vperf > VV.perf)
                VV.numFail = VV.numFail + 1;
            end
        end
    end

    end

  end

  sssE = (EEE*EEE')*((EEEE*EEEE')/(EEE*EEE'))*2;
  ssE = max(ssE,sssE);
  ssX = (X'*X);

end

if (doValidation)
  net = VV.net;
end

% Finish
tr = cliptr(tr,epoch);

%===============================END=========================
```