```
% ADSIP Project Script
% This is the top level script to run compressive sensing with wavelet
% transforms, zero−tree structure exploitation, and Bayesian estimation
% using Gibbs sampling
clear all;

% Initialize values
smax = 6; % set for scaling calculation below; if not used elsewhere, move there
M0 = 4; % set for scaling calculation below; if not used elsewhere, move there
Msvec = [4 3*4.^(1:smax)];
gamcoeffs = 10^−6*[1 1 1 1];
explicit_coeffs = 4;
betacoeffs = [.9*12 .1*12 1/sum(Msvec) 1−1/sum(Msvec) .5 .5 1 explicit_coeffs];
%number of elements
% Read in picture
% Picture Size
% Create Psi Matrix (Wavelet Transform)
% Create Permutation Matrices
% Create Phi Matrix (Sampling Matrix)
phi = sampling_matrix(2000,sum(Msvec),explicit_coeffs);
% Add option here to directly sample "DC" coefficients?
% QUESTION: Do we want to be able to run both simultaneously?
% Combine Wavelet and Sampling matrices
% Create scaling (parent and s−level vectors
% NOTE: MAY NEED TO MOVE SOME OF THE VALUES HERE UP TOP
N = M0*4^smax;
scaling = zeros(N,2);
s = 0;
Mtot = M0;
Ms = M0*3/4;
stepa = 0;
for iter = 1:(N/4) % No children for last level
    scaling(iter,1) = s;
    if s == 0
        scaling(iter,2) = 0;
    elseif s == 1
        scaling(iter,2) = 0;
        num = stepa+2*(row−1)+skip*(col−1);
        scaling(num,2) = iter;
        scaling(num+1,2) = iter;
        scaling(num+skip/2,2) = iter;
        scaling(num+skip/2+1,2) = iter;
        if row ==2^s
            row = 1;
            col = col+1;
        else
            row = row+1;
        end
    else
        num = stepa+2*(row−1)+skip*(col−1);
        scaling(num,2) = iter;
        scaling(num+1,2) = iter;
        scaling(num+skip/2,2) = iter;
        scaling(num+skip/2+1,2) = iter;
        if row ==2^s
            row = 1;
            col = col+1;
        else
            row = row+1;
        end
    end
    if iter == Mtot
        s = s+1;
        Ms = Ms*4;
        stepa = iter+Ms+1;
```

```
        Mtot = Mtot*4;
        row = 1;
        col = 1;
        skip = 2^(s+2);
    elseif iter == Mtot−2*Ms/3
        stepa = stepa+Mtot;
        row = 1;
        col = 1;
    elseif iter == Mtot−Ms/3
        stepa = stepa+Mtot;
        row = 1;
        col = 1;
    end
end
scaling(N/4+1:end,1) = 6;
% Other Initialization?

% Main algorithm
% Sample Image
load test_images
[V, Psi, P] = multilevel_haar(test_image{4},1);
theta_true = P*V(:);
v = phi*theta_true;

% Initialize values for bayesian model (theta, pi, alpha, etc, se inputs to compinference
theta = (−2 + 3*randn(sum(Msvec),1)).*(rand(N,1) < 0.5);
theta(1:explicit_coeffs) = v(1:explicit_coeffs);
[pi, pi_s, mu, alpha, alphas, phi, alphan] = initialize(theta, v, phi, scaling, ...
                                                        gamcoeffs, Msvec, ...
                                                        betacoeffs);

% Loop for Bayesian model (use compinference call here)


L = 20;
mse = zeros(L,1);
h_waitbar = waitbar(0,'Bayesian Inference...');

for l = 1:L
    waitbar(l/L,h_waitbar,'Bayesian Inference...');

    old_theta = theta;
    [theta, pi, pi_s,  mu, alpha, alphas, alphan] = compinference(theta, pi, pi_s, ...
                                                          mu, alpha, alphas, phi, alphan, ...
                                                          v, scaling, gamcoeffs, Msvec, ...
                                                          betacoeffs);
    mse(l) = norm(theta_true−theta);
end

close(h_waitbar);

figure(1); clf;
subplot(2,2,1);
showme(reshape(P.'*significant(theta,0.05),128,[]));
subplot(2,2,2);
showme(reshape(transform(Psi,P.'*theta),128,[]));
subplot(2,2,3);
showme(reshape(P.'*theta,128,[]));
subplot(2,2,4);
plot(mse);

% Inverse transform converged estimate
% Reorder with permutations (may be combined with above step)
% Generate Picture comparison (original vs our reconstructed)
% Calculate actual metric
```

```
function [ theta, pi, pi_s,  mu, alpha, alphas, alphan ] = compinference( theta, pi, pi_s, m
u, alpha, alphas, phi, alphan, v, scaling, gamcoeffs, Ms, betacoeffs)
%compinference This function implements the MCMC inference model for
%zero−tree compressive sensing.  It only runs a single loop, so iterations
%must be run in the code outside the function
%   All these values are ainput and updated in the function:
%   theta − The estimate of the theta vector − Nx1
%   pi − The estimate of the pi values.  This is a Nx1 vector with
%   the values of pi for each theta value as the vectors for each variable.
%   (pi_tilda)
%   pi_s − The values of pi for different values of s and different
%   parents.  (Nx1)
%   mu − The Nx1 vector of mu values
%   alpha − The Nx1 vector alpha values (alpha_tilda)
%   alphas − a vector of the values of alpha for different s−levels
%   alphan − the noise variance
%   These values are input only for use in the update:
%   phi − The matrix of basis functions
%   v − The values from the compressive sampling of the image
%   scaling is a vector with the s and i values in the corresponding
%   locations.  It is Nx2 with s values in the first column and the parent
%   locations in the 2nd column
%   gamcoeffs − 4x1 vector with [a0, b0, c0, d0] in it for the gamma
%   distributions
%   Ms is the passed in sizes of the different scales in order.
%   betacoeffs − 8x1 vector with [e0r,f0r, e0s0,f0s0,e0s1,f0s1,e0sc,explicit_coeffs] in it f
or
%   the beta distributions


% Initialization
N = length(theta);
M = length(v);
smax = max(scaling(:,1));

% Block process for s and i

% Draw for theta(s,i)
pick = rand(N,1); % draws for which distribution to use
check =  pick < pi; % creates a vector of 1s and zeros
% use the 1s to use the non−zero distribution and the zeros to pick the zero distribution
theta = check.*(mu+randn(N,1)./sqrt(alpha));
theta(1:betacoeffs(8)) = v(1:betacoeffs(8));

% Update alpha(s,i)
alpha = alphas(scaling(:,1)+1) + alphan*sum(phi.*phi,1).';

% Update mu(s,i)
phi_times_theta = bsxfun(@times, phi, theta.');
phi_times_theta_j = bsxfun(@minus, sum(phi_times_theta,2), phi_times_theta);
vhat = bsxfun(@minus, v, phi_times_theta_j);
mu = alphan./alpha.*sum(phi.*vhat,1).';

% Draw and calculate intermediate pi value = A
% calculate pi = A/(1+A) (verify) to update pi
temp1 = pi_s./sqrt(alphas(scaling(:,1)+1)).*randn(N,1); % +1 to account for s = 0
temp2 = (1−pi_s).*(mu+1./sqrt(alpha).*randn(N,1));
temp3 = temp1./temp2;
pi = temp3./(1+temp3);
pi(1:4) = 1;

temp3 = 0;
cin = 2+gamcoeffs(3);
for iter = 1:4
    temp3 = temp3+theta(iter)^2;
```

```
    din = temp3*.5+gamcoeffs(4);
    alphas(iter) = gamrnd(cin,din);
end

% draw for alpha(s)
count = 4;
for iter = 2:smax+1
    temp1 = 0;
    temp3 = 0;
    for jiter = 1:Ms(iter)
        if theta(count+jiter) == 0
            temp2 = 0;
        else
            temp2 = 1;
        end
        temp1 = temp1+temp2;
        temp3 = temp3+theta(count+jiter)^2;
    end
    count = count + Ms(iter);
    cin = temp1*.5+gamcoeffs(3);
    din = temp3*.5+gamcoeffs(4);
    alphas(iter) = gamrnd(cin,din);
end
% draw for pi(sc)
% pi_s(1) = 1; %betarnd(betacoeffs(7),betacoeffs(8));
% draw for pi(r)
temp1 = 0;
temp2 = 0;
count = Ms(1);

temp1 = nnz(theta(count+(1:Ms(2)+1)));
temp2 = Ms(2) − temp1;

er = temp1 + betacoeffs(1);
fr = temp2 + betacoeffs(2);
pi_s(5) = betarnd(er,fr);
% draw for pi^0(s) and draw for pi^1(s)
count = 16;
temppi = zeros(smax−1,2);
for iter = 1:smax−1
    temp1 = 0;
    temp2 = 0;
    temp3 = 0;
    temp4 = 0;
    for jiter = 1:Ms(iter+2)
        if theta(count+jiter) == 0
            if theta(scaling(count+jiter,2)) == 0
                temp2 = temp2 +1;
            else
                temp4 = temp4+1;
            end
        else
            if theta(scaling(count+jiter,2)) == 0
                temp1 = temp1 +1;
            else
                temp3 = temp3+1;
            end
        end
    end
    e0 = betacoeffs(3) + temp1;
    f0 = betacoeffs(4) + temp2;
    e1 = betacoeffs(5) + temp3;
    f1 = betacoeffs(6) + temp4;
    temppi(iter,1) = betarnd(e0*Ms(iter+2),f0*Ms(iter+2));
    temppi(iter,2) = betarnd(e1*Ms(iter+2),f1*Ms(iter+2));
```

```
end
% assign value locations for the different pi into pi_s
% pi_s(2:4) = pi_s(1);
pi_s(6:16) = pi_s(5);

stemp = 2;
stemp2 = stemp;
count = Ms(stemp+1);
start = sum(Ms(1:stemp));

for iter = Ms(stemp2+1):N
    if theta(scaling(iter,2)) == 0
        pi_s(iter) = temppi(stemp−1,1);
    else
        pi_s(iter) = temppi(stemp−1,2);
    end
    if iter == start+count
        start = start+count;
        count = count*4;
        stemp = stemp+1;
    end
end

% draw for alphan
v_minus_phi_theta = v − phi*theta;
alphan = gamrnd(gamcoeffs(1)+M/2,gamcoeffs(2)+v_minus_phi_theta.'*v_minus_phi_theta/2);
end
```

```
clear all;

load test_images

SQRTN = length(test_image{1});
N = SQRTN*SQRTN;

display('Generating Transform Matrix...');
[~, Psi, ~] = multilevel_haar(speye(SQRTN),0);

for M = [ 2000 6000 ]

    display('Generating Sampling Matrix...');
    Phi = sampling_matrix(2000,N,0);
    H = Phi*Psi';

    for i = 1:length(test_image)
        U = double(test_image{i})/256−0.5;
        u = U(:);
        N = length(u);
        V = transform(Psi.',U);

        display('Random Sampling...');
        y = H*u;

        display('Basis Pursuit...')
        cvx_begin
          variable t(N)
          minimize( norm(t,1) )
          subject to
            y == Phi*t
        cvx_end

        display('Complete!')

        T = reshape(t,size(U));
        UU = reshape(transform(Psi,T),size(U));

        fig1 = figure(1); clf;
        title1 = sprintf('Image Estimate, %i Samples, Basis Pursuit',M);
        subplot(2,2,2); showme(UU); title(title1); colorbar off;
        title2 = sprintf('True Image: "%s"',image_title{i});
        subplot(2,2,1); showme(U); title(title2); colorbar off;
        title3 = sprintf('Coefficient Estimate');
        subplot(2,2,4); showme(significant(T,0.5)); title(title3); colorbar off;
        title4 = sprintf('True Wavelet Coefficients');
        subplot(2,2,3); showme(significant(V,0.5)); title(title4); colorbar off;
        filename = sprintf('out/%s_m%i_cvx.eps',image_title{i},M);
        print(fig1, '−depsc2', filename);
    end
end
```

```
function [pi, pi_s, mu, alpha, alphas, phi, alphan] = initialize( theta, v, phi, scaling, ga
mcoeffs, Ms, betacoeffs )
%UNTITLED5 Summary of this function goes here
%   Detailed explanation goes here
% Initialization
N = length(theta);
M = length(v);
smax = max(scaling(:,1));

pi = zeros(N,1);
pi_s = zeros(N,1);
mu = zeros(N,1);
alpha = zeros(N,1);
alphas = zeros(smax+1,1);

% Block process for s and i

temp3 = 0;
cin = 2+gamcoeffs(3);
for iter = 1:4
    temp3 = temp3+theta(iter)^2;
    din = temp3*.5+gamcoeffs(4);
    alphas(iter) = gamrnd(cin,din);
end

% draw for alpha(s)
count = 4;
for iter = 2:smax+1
    temp1 = 0;
    temp3 = 0;
    for jiter = 1:Ms(iter)
        if theta(count+jiter) == 0
            temp2 = 0;
        else
            temp2 = 1;
        end
        temp1 = temp1+temp2;
            temp3 = temp3+theta(count+jiter)^2;
    end
    count = count + Ms(iter);
    cin = temp1*.5+gamcoeffs(3);
    din = temp3*.5+gamcoeffs(4);
    alphas(iter) = gamrnd(cin,din);
end
% draw for pi(sc)
% pi_s(1) = 1; %betarnd(betacoeffs(7),betacoeffs(8));
% draw for pi(r)
temp1 = 0;
temp2 = 0;
count = Ms(1);

temp1 = nnz(theta(count+(1:Ms(2)+1)));
temp2 = Ms(2) − temp1;

er = temp1 + betacoeffs(1);
fr = temp2 + betacoeffs(2);
pi_s(5) = betarnd(er,fr);
% draw for pi^0(s) and draw for pi^1(s)
count = 16;
temppi = zeros(smax−1,2);
for iter = 1:smax−1
    temp1 = 0;
    temp2 = 0;
    temp3 = 0;
    temp4 = 0;
```

```
    for jiter = 1:Ms(iter+2)
        if theta(count+jiter) == 0
            if theta(scaling(count+jiter,2)) == 0
                temp2 = temp2 +1;
            else
                temp4 = temp4+1;
            end
        else
            if theta(scaling(count+jiter,2)) == 0
                temp1 = temp1 +1;
            else
                temp3 = temp3+1;
            end
        end
    end
    e0 = betacoeffs(3) + temp1;
    f0 = betacoeffs(4) + temp2;
    e1 = betacoeffs(5) + temp3;
    f1 = betacoeffs(6) + temp4;
    temppi(iter,1) = betarnd(e0*Ms(iter+2),f0*Ms(iter+2));
    temppi(iter,2) = betarnd(e1*Ms(iter+2),f1*Ms(iter+2));
end
% assign value locations for the different pi into pi_s
% pi_s(2:4) = pi_s(1);
pi_s(6:16) = pi_s(5);

stemp = 2;
stemp2 = stemp;
count = Ms(stemp+1);
start = sum(Ms(1:stemp));

for iter = Ms(stemp2+1):N
    if theta(scaling(iter,2)) == 0
        pi_s(iter) = temppi(stemp−1,1);
    else
        pi_s(iter) = temppi(stemp−1,2);
    end
    if iter == start+count
        start = start+count;
        count = count*4;
        stemp = stemp+1;
    end
end

% draw for alphan
v_minus_phi_theta = v − phi*theta;
alphan = gamrnd(gamcoeffs(1)+M/2,gamcoeffs(2)+v_minus_phi_theta.'*v_minus_phi_theta/2);

% Update alpha(s,i)
alpha = alphas(scaling(:,1)+1) + alphan*sum(phi.*phi,1).';

% Update mu(s,i)
phi_times_theta = bsxfun(@times, phi, theta.');
phi_times_theta_j = bsxfun(@minus, sum(phi_times_theta,2), phi_times_theta);
vhat = bsxfun(@minus, v, phi_times_theta_j);
mu = alphan./alpha.*sum(phi.*vhat,1).';

% Draw and calculate intermediate pi value = A
% calculate pi = A/(1+A) (verify) to update pi
temp1 = pi_s./sqrt(alphas(scaling(:,1)+1)).*randn(N,1); % +1 to account for s = 0
temp2 = (1−pi_s).*(mu+1./sqrt(alpha).*randn(N,1));
temp3 = temp1./temp2;
pi = temp3./(1+temp3);
pi(1:4) = 1;
end
```

```
function [V, Psi, P] = multilevel_haar(U,s);

    v = double(U(:))/256−0.5; % Vectorize and normalize the image
    N = length(U);            % Get the dimension of the image

    N2 = N^2;                 % Length of the image vector
    IN2 = speye(N2);          % A N^2 x N^2 identity matrix
    I2 = speye(2);            % A 2x2 identity matrix
    I4 = speye(2);            % A 4x4 identity matrix

    Psi = IN2;                % The complete wavelet basis matrix
    P = IN2;                  % The complete permutation matrix

    % Perform a multilevel wavelet decomposition by iterating over each level
    for n = 2.^(log2(N):−1:s+1)

        n2 = n^2;             % Length of the current sub−image vector
        I = speye(n/2);       % An n/2 x n/2 identity matrix

        % Generate a 1−D Haar wavelet transform matrix of size n x n
        A = 1/sqrt(2)*[ kron(I,[1,  1]); ...
                        kron(I,[1, −1])  ];

        % Convert the 1−D transform matrix to a 2−D transform of
        % size n^2 x n^2 which operates on the vectorized 2−D image
        Psi_n = kron(A',A');

        % Create a block matrix which allows us to apply the transform to only
        % the current sub−image, located at the top−left of the matrix
        Psi_n_N = IN2;
        Psi_n_N(1:n2,1:n2) = Psi_n;

        % Apply the transform to the image
        v = Psi_n_N'*v;

        % Rearrange the transformed sub image to group the four coefficient
        % groups (ie Approximation, Horizontal, Vertical, and Diagonal) together
        pn = kron(kron(I2,I),kron([1 0],I));
        P_n =  vertcat(pn,rot90(pn,2));
        % P_n = kron(I2,kron([kron(I,[1 0]);kron(I,[0 1])],I));

        % Create a block matrix which allows us to apply the permutation to only
        % the current sub−image, located at the top−left of the matrix
        P_n_N = IN2;
        P_n_N(1:n2,1:n2) = P_n;

        % Apply the permutation to the image
        v = P_n_N*v;

        % Update Psi and P
        Psi = Psi * Psi_n_N * P_n_N';
        P = P_n_N * P;

    end

    % Permute the coefficient vector back to make it easier to visualize
    V = reshape(P'*v, N, []);

    % Remove the permutations from the Psi matrix
    Psi = Psi*P;

end
```

```
function Phi = sampling_matrix(M,N,s)

    Phi = zeros(M,N);
    Phi(1:s,1:s) = eye(s);

    for n = s+1:N
        Phi(s+1:M,n) = round(1.2*rand(M−s,1)+0.4)−1;

    end
    % Phi = zeros(M,N);
    % Phi(1:s,1:s) = eye(s);

    % for n = s+1:N
    %     Phi(s+1:M,n) = 2*floor(2*rand(M−s,1))−1;
    % end

end
```