```
function Gibbsmodel(initialization,K_iter)

% Gibbs model for different neighborhoods, potentials, etc.
%  - initialization: 'norm' (Guassian) or 'unif' (uniform)
%  - number of iterations: 3-5
N = 128;

% Precompute possible potential values for 2-element cliques (pixel pairs)
% and 8-bit intensities for Ising, abs and square potential functions.
% Ising
for k = 0:255
    for l = 0:255
        if k == l
            pot_ising(k+1,l+1) = 0;
        else
            pot_ising(k+1,l+1) = 5;
        end
    end
end

% Absolute value
for k = 0:255
    for l = 0:255
        pot_abs(k+1,l+1) = abs(k-l);
    end
end

% Squared
for k = 0:255
    for l = 0:255
      pot_square(k+1,l+1) = (k-l)*(k-l);
    end
end

% Initialize to zero-mean, unit-standard-deviation Gaussian, then scale to
% required standard deviaiotn and mean of 128, and clip to [0,255] range
rng('default');
if initialization == 'norm'
    disp('Normal initialization');
    uinit = max(min(round(50 * randn(N+2) + 128),255),0);
elseif initialization == 'unif'
    disp('Uniform initialization');
    uinit = round(255*rand(N+2));
else
    error('INIT','Invalid initialization');
end

% Run for temperature 1.0
temp = 1.0;
u_1_ising_t1 = gibbs1(N,pot_ising,temp,uinit,K_iter);
u_2_ising_t1 = gibbs2(N,pot_ising,temp,uinit,K_iter);
```

```
u_1_abs_t1 = gibbs1(N,pot_abs,temp,uinit,K_iter);
u_2_abs_t1 = gibbs2(N,pot_abs,temp,uinit,K_iter);

figure(1);
subplot(2,3,1); imshow(uinit(2:N+1,2:N+1),[]);
title('Initial image');
subplot(2,3,2); imshow(u_1_ising_t1(2:N+1,2:N+1),[]);
title(['Poten: Ising, Neighb: 1-st order, Temp: ',num2str(temp)]);
subplot(2,3,3); imshow(u_2_ising_t1(2:N+1,2:N+1),[]);
title(['Poten: Ising, Neighb: 2-nd order, Temp: ',num2str(temp)]);
subplot(2,3,5); imshow(u_1_abs_t1(2:N+1,2:N+1),[]);
title(['Poten: Abs, Neighb: 1-st order, Temp: ',num2str(temp)]);
subplot(2,3,6); imshow(u_2_abs_t1(2:N+1,2:N+1),[]);
title(['Poten: Abs, Neighb: 2-nd order, Temp: ',num2str(temp)]);

imwrite(uinit(2:N+1,2:N+1)/255,'img_init.tif','TIFF');
imwrite(u_1_ising_t1(2:N+1,2:N+1)/255,'img_ising_n1_t1.tif','TIFF');
imwrite(u_2_ising_t1(2:N+1,2:N+1)/255,'img_ising_n2_t1.tif','TIFF');
imwrite(u_1_abs_t1(2:N+1,2:N+1)/255,'img_abs_n1_t1.tif','TIFF');
imwrite(u_2_abs_t1(2:N+1,2:N+1)/255,'img_abs_n2_t1.tif','TIFF');

% Run for temperature 5.0
temp = 2.0;
u_1_ising_t2 = gibbs1(N,pot_ising,temp,uinit,K_iter);
u_2_ising_t2 = gibbs2(N,pot_ising,temp,uinit,K_iter);
u_1_abs_t2 = gibbs1(N,pot_abs,temp,uinit,K_iter);
u_2_abs_t2 = gibbs2(N,pot_abs,temp,uinit,K_iter);

figure(2);
subplot(2,3,1); imshow(uinit(2:N+1,2:N+1),[]);
title('Initial image');
subplot(2,3,2); imshow(u_1_ising_t2(2:N+1,2:N+1),[]);
title(['Poten: Ising, Neighb: 1-st order, Temp: ',num2str(temp)]);
subplot(2,3,3); imshow(u_2_ising_t2(2:N+1,2:N+1),[]);
title(['Poten: Ising, Neighb: 2-nd order, Temp: ',num2str(temp)]);
subplot(2,3,5); imshow(u_1_abs_t2(2:N+1,2:N+1),[]);
title(['Poten: Abs, Neighb: 1-st order, Temp: ',num2str(temp)]);
subplot(2,3,6); imshow(u_2_abs_t2(2:N+1,2:N+1),[]);
title(['Poten: Abs, Neighb: 2-nd order, Temp: ',num2str(temp)]);

imwrite(u_1_ising_t2(2:N+1,2:N+1)/255,'img_ising_n1_t2.tif','TIFF');
imwrite(u_2_ising_t2(2:N+1,2:N+1)/255,'img_ising_n2_t2.tif','TIFF');
imwrite(u_1_abs_t2(2:N+1,2:N+1)/255,'img_abs_n1_t2.tif','TIFF');
imwrite(u_2_abs_t2(2:N+1,2:N+1)/255,'img_abs_n2_t2.tif','TIFF');
```

```
function u = gibbs1(N,pot_mtrx,temperature,uinit,K_iter)

% Sample NxN image from Gibbs distribution using 1-st order neighborhood:
%  - using potential values for 2-element cliques stored in pot_mtrx
%  - using natural temperature: temperature (e.g., 1.0)
%  - starting from initial state: uinit

% Iteratively select new intensity value at each pixel by sampling from
% Gibbs distribution.

prob = double(zeros(256,1));
cum_prob = double(zeros(256,1));
disp('1-st order neighborhood');

u = uinit; % Has dimension (N+2)x(N+2)
for m = 1:K_iter
    r = rand(N);
    for j = 2:N+1
        for i = 2:N+1
            for val = 0:255
                enrg = pot_mtrx(val+1,u(j-1,i  )+1) + ...
                       pot_mtrx(val+1,u(j+1,i  )+1) + ...
                       pot_mtrx(val+1,u(j  ,i-1)+1) + ...
                       pot_mtrx(val+1,u(j  ,i+1)+1);
                prob(val+1) = exp(-enrg/temperature);
            end
            cum_prob = cumsum(prob);
            if cum_prob(256) > 0
                cum_prob = cum_prob/cum_prob(256);
            end
            index = min(find(cum_prob >= r(j-1,i-1)));
            if index > 0
                u(j,i) = index - 1;
            else
                u(j,i) = 0;
            end
        end
    end
end
```

```
function u = gibbs2(N,pot_mtrx,temperature,uinit,K_iter)

% Sample NxN image from Gibbs distribution using 2-nd order neighborhood:
%  - using potential values for 2-element cliques stored in pot_mtrx
%  - using natural temperature: temperature (e.g., 1.0)
%  - starting from initial state: uinit

% Iteratively select new intensity value at each pixel by sampling from
% Gibbs distribution.

prob = double(zeros(256,1));
cum_prob = double(zeros(256,1));
disp('2-nd order neighborhood');

u = uinit; % Has dimension (N+2)x(N+2)
for m = 1:K_iter
    r = rand(N);
    for j = 2:N+1
        for i = 2:N+1
            for val = 0:255
                enrg = pot_mtrx(val+1,u(j-1,i-1)+1) + ...
                       pot_mtrx(val+1,u(j-1,i  )+1) + ...
                       pot_mtrx(val+1,u(j-1,i+1)+1) + ...
                       pot_mtrx(val+1,u(j  ,i-1)+1) + ...
                       pot_mtrx(val+1,u(j  ,i+1)+1) + ...
                       pot_mtrx(val+1,u(j+1,i-1)+1) + ...
                       pot_mtrx(val+1,u(j+1,i  )+1) + ...
                       pot_mtrx(val+1,u(j+1,i+1)+1);
                prob(val+1) = exp(-enrg/temperature);
            end
            cum_prob = cumsum(prob);
            if cum_prob(256) > 0
                cum_prob = cum_prob/cum_prob(256);
            end
            index = min(find(cum_prob >= r(j-1,i-1)));
            if index > 0
                u(j,i) = index - 1;
            else
                u(j,i) = 0;
            end
        end
    end
end
```