```
function X = coordinate_descent(A, Y)

% Determine required dimensions:
[~, num_bases] = size(A);         [~, num_cases] = size(Y);

% Initalize the coefficients:
X = zeros(num_cases,num_bases);

% Specify default parameters:
max_iter = 128;
gamma = 0.95000;
tolerance = 0.000001;

% Precompute static components:
AtA = A'* A;
YtA = Y' * A;
Pj = diag(AtA)';

% Specify gradient step sizes:
alphas = [1,3e-1,1e-1,3e-2,1e-2]; num_alphas = length(alphas);

% Append no-progress step-size:
alphas_plus_zero = [alphas 0.0];  no_progress = num_alphas + 1;

% Apply coordinate descent:
for iter = 1:max_iter

  % Compute the gradient vector:
  Y_minus_Ax_t_A = YtA - X * AtA;
  Qj = Y_minus_Ax_t_A + repmat(Pj, [num_cases,1]) .* X;
  Xstar = (Qj + sign(-Qj) * gamma) ./ repmat(Pj, [num_cases,1]);

  % Zero out small coefficients:
  Xstar( abs(-Qj) < gamma ) = 0;

  % Prepare for the line search:
  D = Xstar - X;     DtAtA = D * AtA';
  av = sum(0.5 * DtAtA .* D,2);
  bv = sum(- Y_minus_Ax_t_A .* D,2);

  % Find the minimizing step size:
  minHx = gamma * norms(X,2);

  % Solve the line-search equations:
  Hx = ones(num_alphas,num_cases);
  for k = 1:num_alphas
    Hx(k,:) = av * alphas(k) * alphas(k) + ...
      bv * alphas(k) + gamma * norms(X + alphas(k) * D, 2);
  end
  [Hx, I] = min(Hx, [], 1);
  I( ~( Hx' < minHx * ( 1 - tolerance ) ) ) = no_progress;

  % Terminate loop if no progress:
  if all( I == no_progress ); break; end

  % Apply the gradient step update:
  X = X + repmat(alphas_plus_zero(I)',[1,num_bases]) .* D;
end
```