```
b = [1; 0; 0; 0];

for epsilon = [2*sqrt(eps) sqrt(eps) 2*eps, eps]
  disp("Epsilon = " + eps + ":")
  A = genA(epsilon);
  xreal = [
    1/(1+epsilon^2);
    -1/(epsilon^2+epsilon^4) + 1/(epsilon^2);
    1/(epsilon^2+epsilon^4) - 1/(epsilon^2)
  ];

  x = solveNormal(A, b);
  print_error("Normal equations", x, xreal, A, b)

  x = solveClassicalGS(A, b);
  print_error("ClassicalGS", x, xreal, A, b)

  x = solveModifiedGS(A, b);
  print_error("ModifiedGS", x, xreal, A, b)

  x = solveHouseholder(A, b);
  print_error("Householder", x, xreal, A, b)

  x = solveGivens(A, b);
  print_error("Givens", x, xreal, A, b)

  x = solveQR(A, b);
  print_error("QR", x, xreal, A, b)

  x = solveBackslash(A, b);
  print_error("Backslash", x, xreal, A, b)

  x = solveSVD(A, b);
  print_error("SVD", x, xreal, A, b)

  x = solveTSVD(A, b);
  print_error("TSVD", x, xreal, A, b)
end

function print_error(name, x, xreal, A, b)
  err = norm(x - xreal, 2);
  res = norm(b - A * x, 2);
  disp("  - " + name + ": err = " + err + ", residual = " + res);
end

% genA generates the matrix A with the given value of epsilon.
function A = genA(epsilon)
  A = [1 1 1; epsilon 0 0; 0 epsilon 0; 0 0 epsilon];
end

% solveNormal solves least squares via the normal equations.
function x = solveNormal(A, b)
  % Form B = A^TA and y = A^Tb
  B = A' * A;
  y = A' * b;
  % solve the system Bx = y
  x = B\y;
end

% solveClassicalGS solves least squares via classical Gram-Schmidt
% orthogonalization.
function x = solveClassicalGS(A, b)
  % compute size of matrix
  [rows, cols] = size(A);
  % initialize Q, R
  Q = zeros(rows, cols);
  R = zeros(cols, cols);
  % iterate though each column
  for j=1:cols
    % initialize Q column j from A
    Q(:, j) = A(:, j);
    for i=1:(j-1)
      % compute the dot product between A j and Q i
```

```
      R(i, j) = Q(:, j)' * A(:, i);
      Q(:, j) = Q(:, j) - R(i, j) * Q(:, i);
    end
    % compute R jj value
    R(j, j) = norm(Q(:, j));
    % normalize column j
    Q(:, j) = Q(:, j)./R(j, j);
  end
  % use the QR decomposition to solve
  x = R\(Q' * b);
end

% solveModifiedGS solves least squares via modified Gram-Schmidt
% orthogonalization.
function x = solveModifiedGS(A, b)
  % compute size of matrix
  [rows, cols] = size(A);
  % initialize Q, R
  Q = zeros(rows, cols);
  R = zeros(cols, cols);
  % iterate though each column
  for j=1:cols
    % initialize Q column j from A
    Q(:, j) = A(:, j);
    for i=1:(j-1)
      % compute the dot product between Q j and Q i
      R(i, j) = Q(:, j)' * Q(:, i);
      Q(:, j) = Q(:, j) - R(i, j) * Q(:, i);
    end
    % compute R jj value
    R(j, j) = norm(Q(:, j));
    % normalize column j
    Q(:, j) = Q(:, j)./R(j, j);
  end
  % use the QR decomposition to solve
  x = R\(Q' * b);
end

% solveHouseholder solves least squares via Householder Transformations QR.
% This is based off of the sample code in the textbook.
function x = solveHouseholder(A, b)
  [m,n] = size(A);
  p = zeros(1,n);
  for k = 1:n
    % define u of length = m-k+1
    z = A(k:m, k);
    e1 = [1; zeros(m-k,1)];
    u = z+sign(z(1)) * norm(z) * e1;
    u = u/norm(u);
    % update nonzero part of A by I-2uu^T
    A(k:m,k:n) = A(k:m,k:n)-2 * u*(u'*A(k:m,k:n));
    % store u
    p(k) = u(1);
    A(k+1:m, k) = u(2:m-k+1);
  end
  y = b(:);
  % transform b
  for k=1:n
    u = [p(k);
    A(k+1:m,k)];
    y(k:m) = y(k:m) - 2 * u * (u' * y(k:m));
  end
  % form upper triangular R and solve
  R = triu(A(1:n,:));
  x = R \ y(1:n);
end

% solveGivens solves least squares via Givens Rotation QR factorization.
function x = solveGivens(A, b)
  [rows, cols] = size(A);
  Q = eye(rows);
  % eliminate column by column
  for i=1:cols
```

```
    % then eliminate row by row
    for j=(i+1):rows
      % compute s and c values so we can make the rotation matrix
      alpha = sqrt(A(i,i)^2 + A(j, i));
      s = A(j,i)/alpha;
      c = A(i,i)/alpha;
      % create the rotation matrix
      G = eye(rows);
      G(i, i) = c;
      G(j, i) = -s;
      G(i, j) = s;
      G(j, j) = c;
      % rotate A and Q with the rotation matrix
      A = G * A;
      Q = G * Q;
    end
  end
  % transpose Q to get the actual Q matrix
  Q = Q';
  R = A;
  x = R\(Q' * b);
end

% solveQR solves least squares by using Matlab's qr function.
function x = solveQR(A, b)
  % compute the QR factorization
  [Q, R] = qr(A);
  % compute c = Q^Tb
  c = Q' * b;
  % solve the system Rx = c
  x = R\c;
end

% solveBackslash solves least squares by using Matlab's backslash operator.
function x = solveBackslash(A, b)
  % solve the system Ax = b
  x = A\b;
end

% solveSVD solves least squares via SVD.
function x = solveSVD(A, b)
  % compute SVD of A
  [U, S, V] = svd(A);
  % compute z = U^Tb
  z = U' * b;
  % compute pseudo inverse of S and apply it to z
  [rows, cols] = size(S);
  y = z(1:cols);
  diagS = diag(S);
  % divide y by non-zero elements of S
  y(diagS~=0) = y(diagS~=0)./diagS(diagS~=0);
  % Compute x = Vy
  x = V * y;
end

% solveTSVD solves least squares via TSVD.
function x = solveTSVD(A, b)
  % compute SVD of A
  [U, S, V] = svd(A);
  % truncate small values of S
  S(S<10^-6) = 0;
  % compute z = U^Tb
  z = U' * b;
  % compute pseudo inverse of S and apply it to z
  [rows, cols] = size(S);
  y = z(1:cols);
  diagS = diag(S);
  % divide y by non-zero elements of S
  y(diagS~=0) = y(diagS~=0)./diagS(diagS~=0);
  % Compute x = Vy
  x = V * y;
end
```