

---

# COMPGV19: Tutorial 4

## Table of Contents

Exercise 1 .....	1
Initialisation .....	1
SOLVE - Without preconditioning .....	1
SOLVE - With preconditioning .....	3
conjugateGradient.m .....	11

Marta Betcke and Kiko Rul#lan

## Exercise 1

Implement the linear preconditioned Conjugate Gradient method

```
clear all, close all;
```

### Initialisation

```
n = 1e2;
tol = 1e-12;
maxIter = 1e3;
% Initial point
x0 = zeros(n, 1);

% Matrix definition - Use sparse matrices
b = ones(n, 1);
A1 = spdiags((1:n)', 0, n, n);
A2 = spdiags([ones(n-1, 1); 100], 0, n, n);
A3 = spdiags([-ones(n-1, 1); 0], -1, n, n) + spdiags(2*ones(n, 1), 0,
    n, n) + spdiags([0; -ones(n-1, 1)], 1, n, n);
% A4 = hilb(n);

% Define xtrue
xtrue = zeros(n,1);
xtrue(floor(n/4):floor(n/3)) = 1;
xtrue(floor(n/3)+1:floor(n/2)) = -2;
xtrue(floor(n/2)+1:floor(3/4*n)) = 1/2;

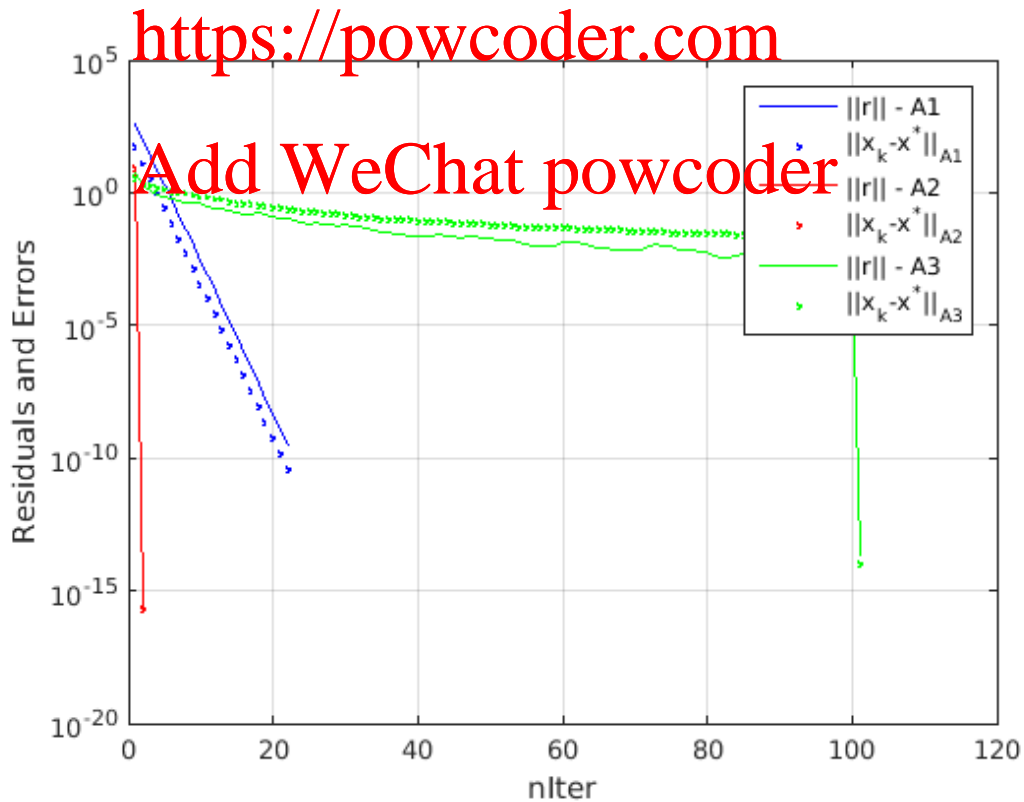
% Alternative xtrue
% [V, Lambda] = eig(full(A3));
% k = 5;
% xtrue = V(:,1:5)*randn(5,1);
```

### SOLVE - Without preconditioning

```
% Identity operator
M = @(y) y;
```

```
% Linear preconditioned Conjugate gradient backtracking line search
[xMin1, nIter1, resV1, infoCG1] = conjugateGradient(A1, A1*xtrue, tol,
    maxIter, M, x0, xtrue);
[xMin2, nIter2, resV2, infoCG2] = conjugateGradient(A2, A2*xtrue, tol,
    maxIter, M, x0, xtrue);
[xMin3, nIter3, resV3, infoCG3] = conjugateGradient(A3, A3*xtrue, tol,
    maxIter, M, x0, xtrue);

% Plot Residuals and Errors
figure;
semilogy(resV1, 'b');
hold on;
semilogy(infoCG1.errA, '*b', 'MarkerSize', 2);
semilogy(resV2 + eps, 'r'); % add eps for log plot
semilogy(infoCG2.errA + eps, '*r', 'MarkerSize', 2);
semilogy(resV3, 'g');
semilogy(infoCG3.errA, '*g', 'MarkerSize', 2);
xlabel('nIter');
ylabel('Residuals and Errors');
legend('||r|| - A1', '||x_k - x^*||_{A1}', '||r|| - A2', '||x_k - x^*||_{A2}',
    '||r|| - A3', '||x_k - x^*||_{A3}');
grid on;
saveas(gcf, '../figs/01_01_ResidualsErrors', 'png');
```

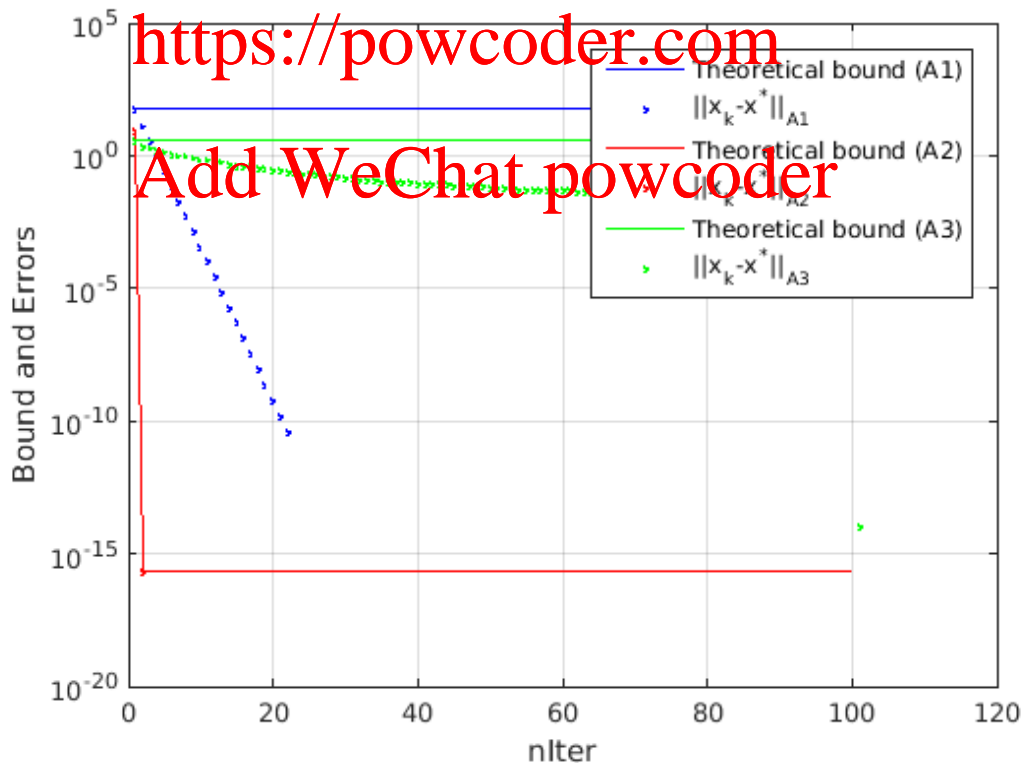


```
% Plot Residuals and Errors
lambda1 = sort(eig(A1), 'descend');
```

```

lambda2 = sort(eig(A2), 'descend');
lambda3 = sort(eig(A3), 'descend');
bound1 = abs((lambda1 - lambda1(end))./(lambda1 +
    lambda1(end))).*sqrt(xtrue'*A1*xtrue);
bound2 = abs((lambda2 - lambda2(end))./(lambda2 +
    lambda2(end))).*sqrt(xtrue'*A2*xtrue);
bound3 = abs((lambda3 - lambda3(end))./(lambda3 +
    lambda3(end))).*sqrt(xtrue'*A3*xtrue);
figure;
semilogy(bound1, 'b');
hold on;
semilogy(infoCG1.errA, '*b', 'MarkerSize', 2);
semilogy(bound2 + eps, 'r'); % add eps for log plot
semilogy(infoCG2.errA + eps, '*r', 'MarkerSize', 2);
semilogy(bound3, 'g');
semilogy(infoCG3.errA, '*g', 'MarkerSize', 2);
xlabel('nIter');
ylabel('Bound and Errors');
legend('Theoretical bound (A1)', '|x_k - x^*|_{A1}', 'Theoretical
    bound (A2)', '|x_k - x^*|_{A2}', 'Theoretical bound (A3)', '|x_k -
    x^*|_{A3}');
grid on;
saveas(gcf, 'figs/02_errs_theoret_cg1.png');

```



## SOLVE - With preconditioning

```
%=====
```

```
% A1
%=====

A = A1;

% Diagonal preconditioner is equivalent to Cholesky for diagonal
  matrices
M = @(y) diag(1./diag(full(A)))*y;

% Eigenvalue distribution of A and M^(-1)*A
figure;
semilogy(eig(full(A)), 'k. ');
hold on;
semilogy(real(eig(M(full(A)))), 'ro', 'MarkerSize', 5);
legend('A', 'M^{-1}A')
title('Eigenvalues for A and M^{-1}A');
grid on;
saveas(gcf, '../figs/01_03_EigenvaluesA1', 'png');

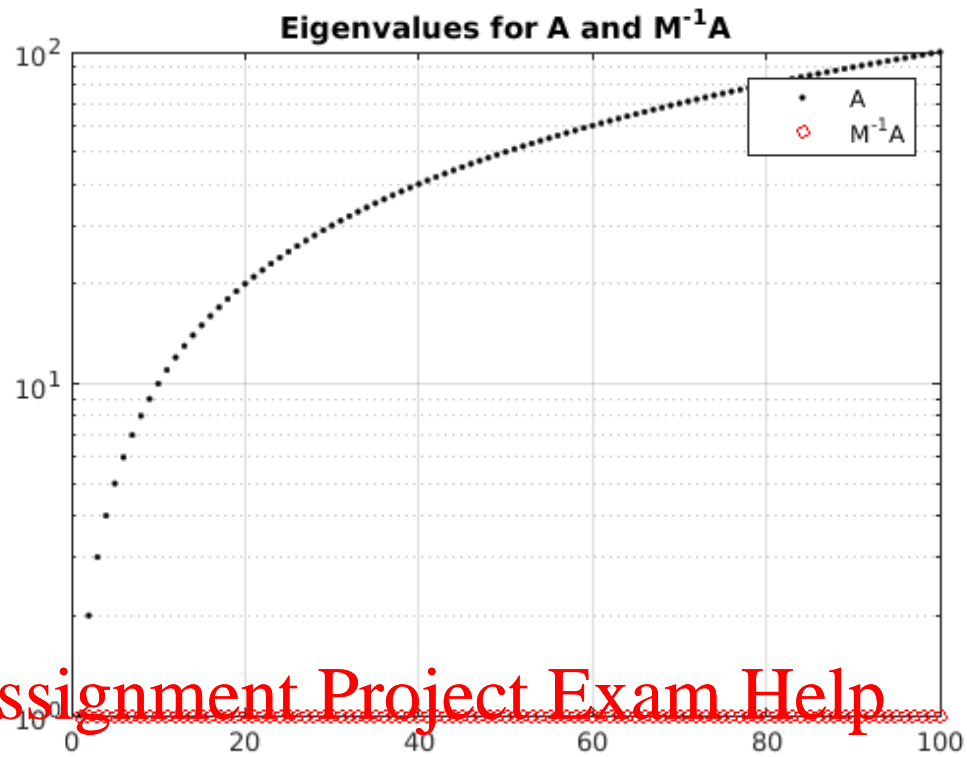
% Solution with CG
[xMin_d, nIter_d, resV_d, infoCG_d] = conjugateGradient(A, A*xtrue,
  tol, maxIter, M, x0, xtrue);

% Plot residuals and errors
figure;
semilogy(resV1, 'k');
hold on;
semilogy(infoCG1.errA, 'r--', 'LineWidth', 2);
semilogy(resV_d, 'r');
semilogy(infoCG_d.errA, 'r--', 'LineWidth', 2);
grid on;
xlabel('iterations');
legend('||r||', '||x_k - x^*||_A', '||r|| - diagonal', '||x_k - x^*||_A -
  diagonal');
title('Residuals and Errors');
saveas(gcf, '../figs/01_04_ResidualsPreconditioningA1', 'png');
```

Assignment Project Exam Help

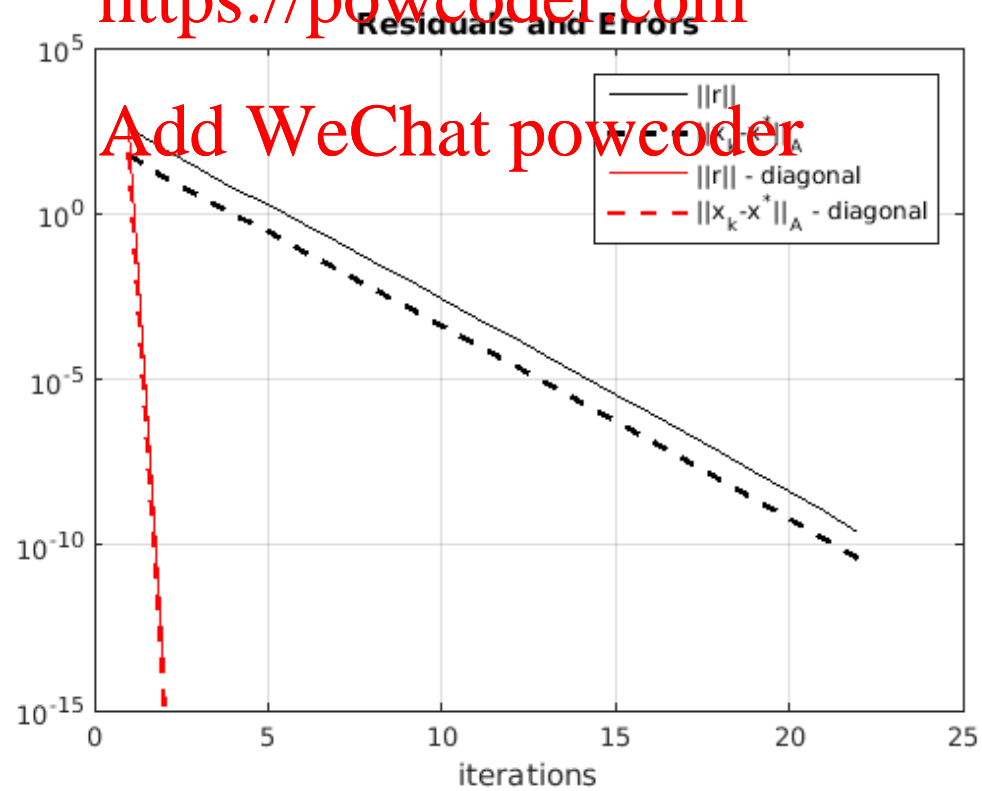
<https://powcoder.com>

Add WeChat powcoder



Assignment Project Exam Help

<https://powcoder.com>



=====

```
% A2
%=====

A = A2;

% Diagonal preconditioner is equivalent to Cholesky for diagonal
  matrices
M = @(y) diag(1./diag(full(A)))*y;

% Eigenvalue distribution of A and M-1A
figure;
semilogy(eig(full(A)), 'k. ');
hold on;
semilogy(real(eig(M(full(A)))), 'ro', 'MarkerSize', 5)
legend('A', 'M-1A')
title('Eigenvalues for A and M-1A');
grid on;
saveas(gcf, '../figs/01_05_EigenvaluesA2', 'png');

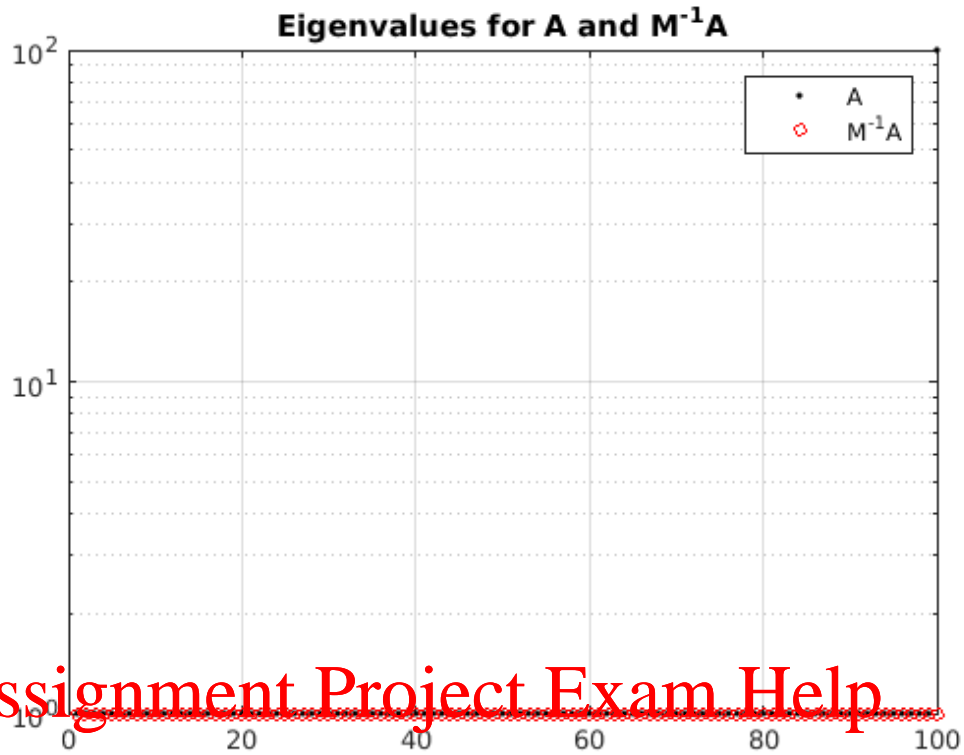
% Solution with CG
[xMin_d, nIter_d, resV_d, infoCG_d] = conjugateGradient(A, A*xtrue,
  tol, maxIter, M, x0, xtrue);

% Plot residuals and errors
figure;
semilogy(resV_d + eps, 'r', 'LineWidth', 4);
hold on;
semilogy(infoCG_d.errA + eps, 'r--', 'LineWidth', 2);
semilogy(resV2 + eps, 'k');
semilogy(infoCG2.errA + eps, 'k--', 'LineWidth', 2);
grid on;
xlabel('iterations');
legend('||r|| - diagonal', '||x_k - x*||_A - diagonal', '||r||', '||
x_k - x*||_A');
title('Residuals and Errors');
saveas(gcf, '../figs/01_06_ResidualsPreconditioningA2', 'png');
```

Assignment Project Exam Help

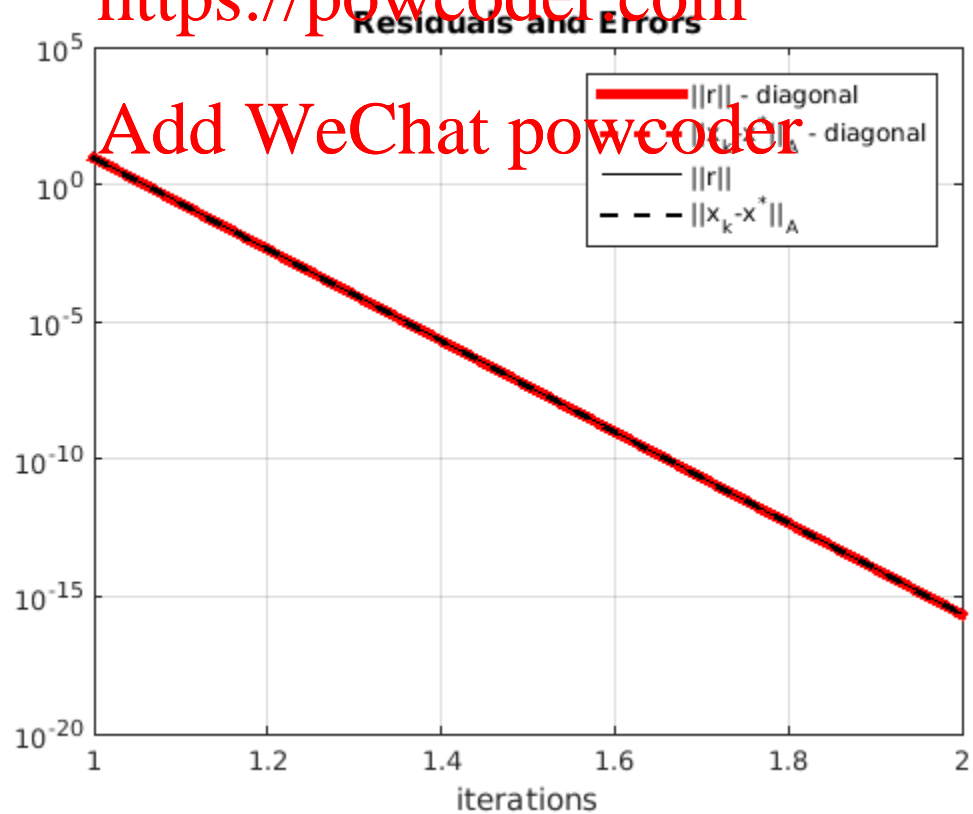
<https://powcoder.com>

Add WeChat powcoder



Assignment Project Exam Help

<https://powcoder.com>



Add WeChat powcoder

```

%=====
% A3
%=====
A = A3;

% Diagonal
M = @(Y) diag(1./diag(full(A)))*Y;
% Cholesky
L_c = chol(A, 'lower'); % M = L*L' -> M*x = y -> (L*L')*x = y -> x =
    L'\(L\Y);
M_c = @(Y) L_c'\(L_c\Y);
% Incomplete Cholesky
icholOpts.type = 'ict';
icholOpts.droptol = 4e-1;
icholOpts.michol = 'off'; % this tridiagonal matrix is to simple for
    modification
L_i = ichol(A, icholOpts);
M_i = @(Y) L_i'\(L_i\Y);

% Sparsity pattern
figure;
spy(L_c);
title('Cholesky sparsity');
saveas(gcf, '../figs/01_07_SparsityCholeski', 'png');

figure;
spy(L_i);
title('Incomplete Cholesky sparsity');
saveas(gcf, '../figs/01_08_SparsityIncompleteChol', 'png');

% Eigenvalue distribution
figure;
semilogy(eig(full(A)), 'mx');
hold on;
semilogy(real(eig(M(full(A)))), 'r*') % eigenvalues of A and M^(-1)*A
semilogy(real(eig(M_c(full(A)))), 'bo') % eigenvalues of A and
    M^(-1)*A
semilogy(real(eig(M_i(full(A)))), 'ks') % eigenvalues of A and
    M^(-1)*A
legend('A', 'M^{-1}A - diagonal', 'M^{-1}A - Cholesky', 'M^{-1}A - I.
    Cholesky');
title('Eigenvalues for A and M^{-1}A');
grid on;
saveas(gcf, '../figs/01_09_EigenvaluesA3', 'png');

% Solution with CG
[xMin_c, nIter_c, resV_c, infoCG_c] = conjugateGradient(A, A*xtrue,
    tol, maxIter, M_c, x0, xtrue);
[xMin_i, nIter_i, resV_i, infoCG_i] = conjugateGradient(A, A*xtrue,
    tol, maxIter, M_i, x0, xtrue);
[xMin_d, nIter_d, resV_d, infoCG_d] = conjugateGradient(A, A*xtrue,
    tol, maxIter, M, x0, xtrue);

figure;

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

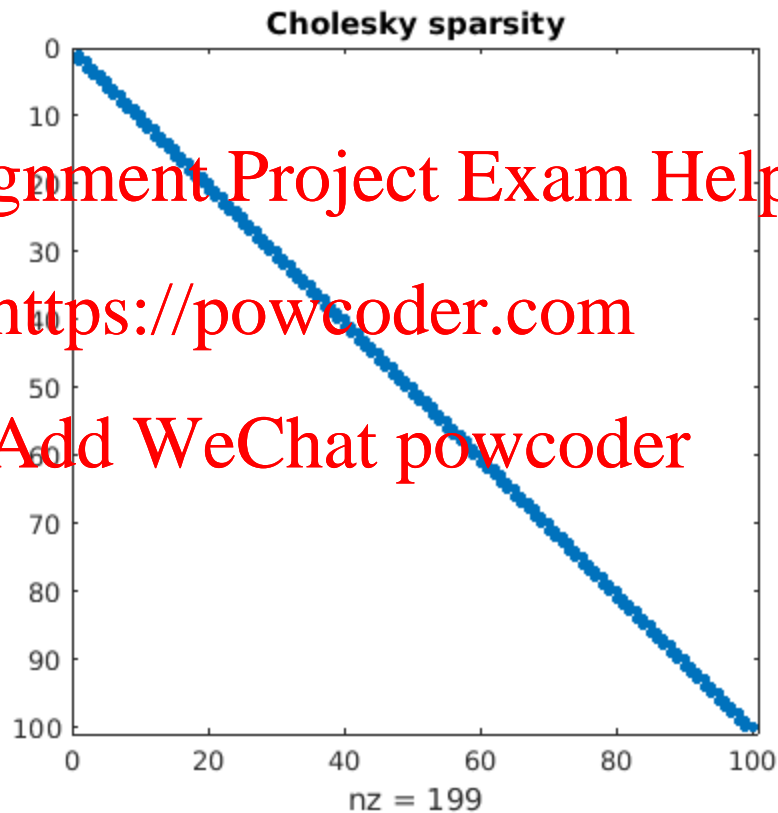


```

grid on;
semilogy(resV3,'xm','MarkerSize', 2); hold on;
semilogy(resV_d,'*r','MarkerSize', 2);
semilogy(resV_c,'ob', 'MarkerSize', 2);
semilogy(resV_i,'sk', 'MarkerSize', 4);

semilogy(infoCG3.errA, 'm-','MarkerSize', 2);
semilogy(infoCG_d.errA, 'r--','MarkerSize', 2);
semilogy(infoCG_c.errA, 'b-', 'MarkerSize', 2);
semilogy(infoCG_i.errA, 'k:', 'MarkerSize', 2);
legend('||r||', '||r|| - diagonal', '||r|| - Cholesky', '||r|| - inc. Cholesky', '||x_k-x^*||_A', '||x_k-x^*||_A - diagonal', '||x_k-x^*||_A - Cholesky', '||x_k-x^*||_A - inc. Cholesky');
title('Residuals and Errors');
saveas(gcf, '../figs/01_10_ResidualsPreconditioningA3', 'png');

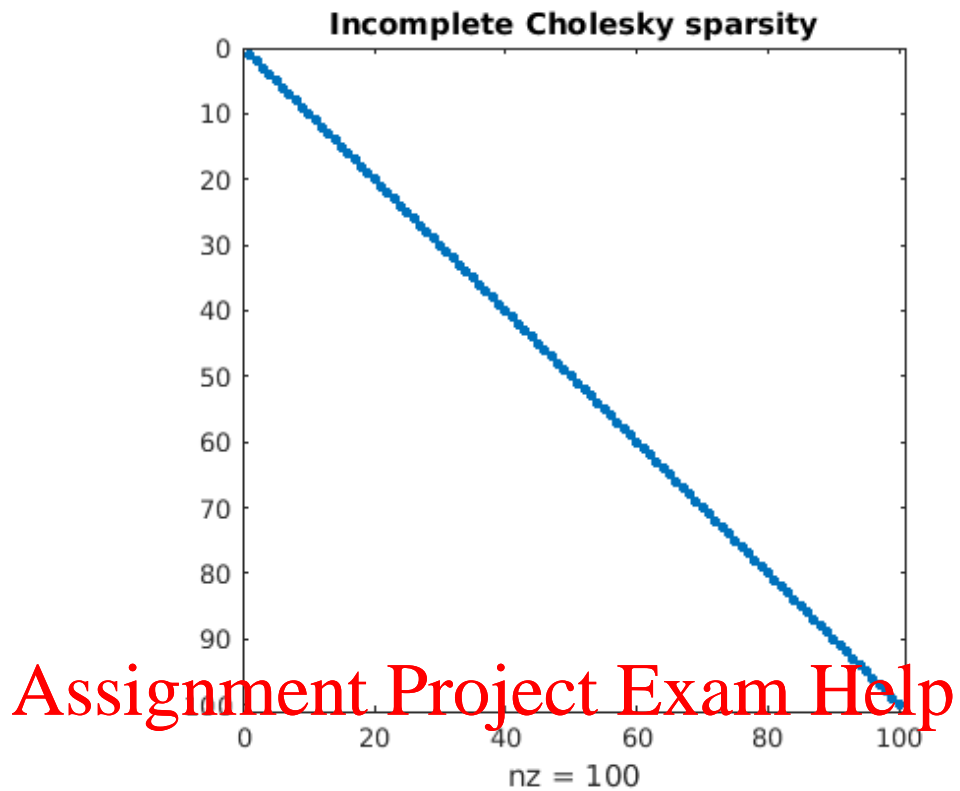
```



Assignment Project Exam Help

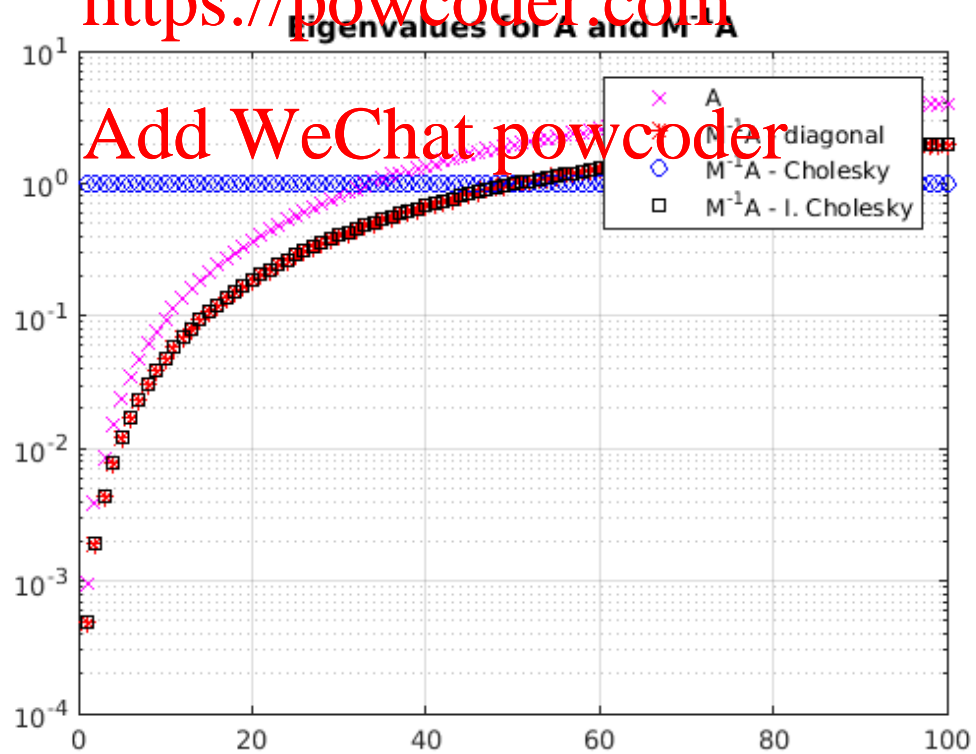
<https://powcoder.com>

Add WeChat powcoder

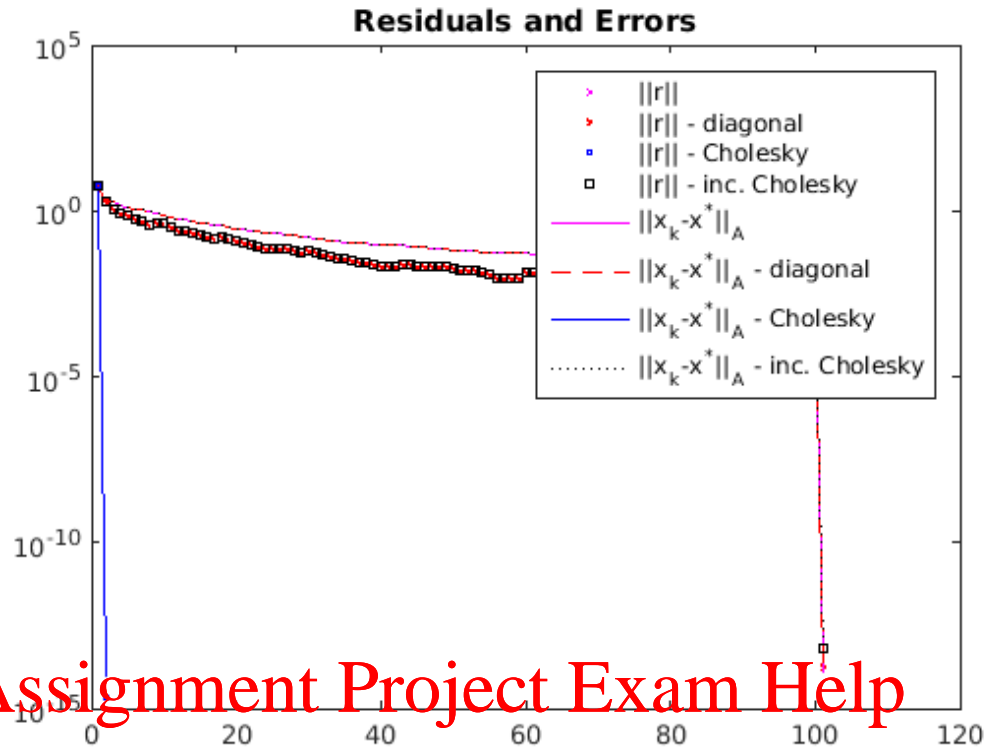


Assignment Project Exam Help

<https://powcoder.com>



Add WeChat powcoder



Assignment Project Exam Help

<https://powcoder.com>

## conjugateGradient.m

Add WeChat powcoder

Conjugate gradient method.

```
function [xMin, nIter, resV, info] = conjugateGradient(A, b, tol,
    maxIter, M, x0, xtrue)
% CONJUGATE GRADIENT Solves the implicitly symmetric preconditioned
CG
%function [x, flag, relres, iter, resvec, xs, V] = mcg(A, b, tol,
    maxit, M, x0)
% INPUTS
% A: symmetric matrix
% b: specifies the vector to solve Ax = b
% tol: tolerance for the residual
% maxIter: maximum number of iterations
% M: linear preconditioner matrix
% x0: initial iterate
%
% OUTPUTS
% xMin: solution of the system
% nIter: number of iterations taken
% resV: vector of residuals
% info: structure with information about the iteration
% - xs: iterate history
```

```

% Notice that the left preconditioned NE (with M inner product) and
% the right preconditioned NE (with M^-1 inner product) produce the
% same algorithm,
% hence only one version.
%
% Copyright (C) 2017 Marta M. Betcke, Kiko Rul-ian

if ~strcmp(class(A), 'function_handle')
    afun = @(x) A*x;
else
    afun = A;
end
if ~strcmp(class(M), 'function_handle')
    mfun = @(x) M\x;
else
    mfun = M;
end

n = length(b);

if nargin < 6
    x0 = zeros(n,1);
end

%Initialize:
x = x0; % Initial x
r0 = b - afun(x); % residual b - Ax
z0 = mfun(r0); % M^(-1)r0
p0 = z0; % conjugate gradient
rnorm = norm(r0); % residual norm |r|
bnorm = norm(z0); % M-norm of r0

alpha = zeros(maxIter,1);
beta = zeros(maxIter,1);
resV = zeros(maxIter+1,1);
resV(1) = rnorm;
xs = zeros(n, maxIter+1);
V = zeros(n, maxIter+1);
xs(:,1) = x;
V(:,1) = z0;
j = 1;
while j <= maxIter && rnorm/bnorm >= tol %|r|/|b| > tol

    %Preconditioned Conjugate Gradient step
    Ap0 = afun(p0);
    alpha(j) = (r0'*z0) / (Ap0'*p0);
    x = x + alpha(j)*p0;
    r1 = r0 - alpha(j)*Ap0;
    z1 = mfun(r1);
    beta(j) = (r1'*z1) / (r0'*z0);
    p1 = z1 + beta(j)*p0;

    %Compute norm
    rnorm = norm(r1);

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```
resV(j+1) = rnorm;
xs(:,j+1) = x;

%Compute the Lanczos vectors : v = scalar * r
V(:,j+1) = z1;

%Next step
p0 = p1;
r0 = r1;
z0 = z1;
j = j+1;
end

%Assign output
if rnorm/bnorm > tol,
    %If MCG iterated maxIter without convergence, return the solution
    with the minimal residual
    flag = 1;
    [~, jmin] = min(resV);
    x = xs(:,jmin);
    rnorm = resV(jmin);
    nIter = jmin-1;
else
    flag = 0;
    nIter = j-1;
end

relres = rnorm/norm(b);
resV = resV(1:(min(j,maxIter+1)));
info.xs = xs(:,1:(min(j,maxIter+1)));
V = V(:,1:(min(j,maxIter+1)));
xMin = x;

if nargin > 6
    for k = 1:min(j,maxIter+1)
        info.errA(k) = sqrt((xs(:,k)-xtrue)'*afun(xs(:,k)-xtrue));
    end
end
```

*Published with MATLAB® R2015a*