HW 2

1: Iterative refinement  
Recall from class that in an iterative refinement scheme for solving $\hat{A}x = b$,

$$x_{k+1} = x_k + A^{-1}(b - \hat{A}x_k).$$

we have $\|x_{k+1} - x\| \leq \|A^{-1}E\|\|x_k - x\|$ if all the calculations are done in exact arithmetic. Now suppose that the residual is computed with some small error, so that the iteration in floating point looks like

$$x_{k+1} = x_k + A^{-1}(b - \hat{A}x_k + \delta_k).$$

Assuming $\|\delta_k\| < \epsilon$, what can we say about the asymptotic accuracy that the scheme can attain? Assume that $\|A^{-1}E\| < 1$.

2: Factoring a structured matrix.  
Let $A = I + uv^T$, where $\|u\|_2\|v\|_2 < 1$. Every principal minor of $A$ is nonsingular (why?), so we may write $A = LU$. Using the fact that each Schur complement in $A$ differs from an identity block by a rank 1 matrix, write an $O(n^2)$ time algorithm to compute $L$ and $U$. Your code should look like

```matlab
function [L,U] = p2lu(u, v)
% Equivalent to [L,U] = lu(I+u*v') without pivoting.
```

3: Bidiagonal conditioning.  
Let $B$ be an upper bidiagonal matrix, i.e.

$$B = \begin{bmatrix}
\alpha_1 & \beta_1 \\
\alpha_2 & \beta_2 & \ddots \\
& \ddots & \ddots \\
& & \alpha_{n-1} & \beta_{n-1} \\
& & & \alpha_n
\end{bmatrix}.$$

Write a routine to compute $\kappa_\infty(B) = \|B\|_\infty\|B^{-1}\|_\infty$ in $O(n)$ time. Your code should look like

```matlab
function [kappa] = p3cond(alpha,beta)
% Equivalent to
% B = diag(alpha) + diag(beta,1);
% kappa = cond(B,inf);
```
4: Modified Cholesky Suppose \( A \) is symmetric (maybe not positive definite), and consider the following modified Cholesky routine:

```matlab
function [Lmod, d] = modchol(A);

n = length(A);
d = zeros(n,1);
for j = 1:n
    if A(j,j) < 0
        d(j) = -2*A(j,j);
        A(j,j) = -A(j,j);
    end
    A(j,j) = sqrt(A(j,j));
    A(j+1:end,j) = A(j+1:end,j)/A(j,j);
end

Lmod = tril(A);
```

Argue that \( LL^T = A + D \), where \( D \) is a diagonal matrix whose entries are given by the elements of \( d \). Given \( Lmod \) and \( d \), write a routine \texttt{p4solve} to solve \( Ax = b \) in \( O(n^2 + nm^2) \) time, where \( m \) is the number of negative elements in \( d \).

```matlab
function x = p4solve(Lmod,d,b)
    % Solve (Lmod*Lmod'-diag(d))*x = b.
    hint: Use the Sherman-Morrison-Woodbury formula.
```