Practice Final

This practice final is meant to give you an idea what to expect on the actual final. But the practice final turns out to be harder (I think)! The actual final has the same general format, though, and covers a similar range of topics.

1: (2 points each) Let $u, v \in \mathbb{R}^n$. True or false:

1. $|\operatorname{fl}(u^T v) - u^T v| \leq n\epsilon_{\text{mach}}|u|^T|v| + O(\epsilon_{\text{mach}}^2)$
   True.

2. $\|u\|_\infty \leq \|u\|_2 \leq \sqrt{n}\|u\|_\infty$
   True.

3. For any real symmetric $A \in \mathbb{R}^{n\times n}$ there is an associated inner product $\langle x, y\rangle_A = x^T Ay$.
   False. A must also be positive definite.

4. For any square A, sensitivity of the eigenvalues of A is bounded by $\kappa_2(A) = \|A\|\|A^{-1}\|$.
   False. $\kappa_2(A)$ is the condition number for solving linear systems, and bears little relation to the sensitivity of the eigenvalues. For example, a symmetric matrix has perfectly well-behaved eigenvalues even if it is well conditioned.

5. The QR iteration is a backward stable method to compute eigenvalues.
   True. Most of the algorithms in this class are backward stable.

6. If A is a square matrix, every eigenvalue is within $\|A\|_1$ of some diagonal entry of A.
   True. At least, it's true in the interpretation I had intended (for any λ there exists an i such that $|A_{ii} - \lambda| \leq \|A\|_1$). This is a weaker statement than Gerschgorin's theorem.

2: (6 points each) One-liners in MATLAB:

1. Suppose A is a real symmetric matrix and `[L,U,P] = lu(A)` returns $P = I$. Compute the number of positive eigenvalues of A.

   Solution: Since A is symmetric, we can write $A = LU = LDL^T$, where $D = \text{diag}(U)$. Inertia is preserved under congruence, so A and D have the same number of positive, negative, and zero eigenvalues; therefore

   ```
   pos_eigs = sum(diag(U) > 0);
   ```

2. Suppose $A = QR$ is computed using Householder reflections, and you are given R. Compute $\det(A)$ (do not use `det` or refer explicitly to A or Q)

   Solution: The determinant of a product is a product of determinants; and the determinant of a Householder reflection is -1. The determinant of R, of course, is the product of the diagonal entries, and so

   ```
   n = length(R);
   detA = (-1)^(n-1) * prod(diag(R));
   ```

3. Suppose $A = QR$ is given. Solve $Ax = b$ in $O(n^2)$ time.

   Solution: `x = R\(Q'*b);` the parentheses are important!

3: (10 points) Fill in the following routine

```
function Sdot = differentiate_svd(A,E)
%
% Return Sdot = dS/dt(0) where S(t) is the singular value
% matrix for A+tE.  Assume the singular values of A are distinct.
```

Solution: The formula is exactly analogous to the formula for sensitivity of eigenvalues, and can be derived using the same trick. The change in the singular vectors must be orthogonal to the singular vectors themselves (to maintain orthogonality), so

$$\begin{aligned}
\delta\sigma_j &= \delta(u_j^T A v_j) \\
&= (\delta u_j)^T A v_j + u_j^T (\delta A) v_j + u_j^T A (\delta v_j) \\
&= (\delta u_j)^T u_j \sigma_j + u_j^T (\delta A) v_j + \sigma_j v_j^T (\delta v_j) \\
&= u_j^T (\delta A) v_j.
\end{aligned}$$

Thus:

```
  [U,S,V] = svd(A);
  Sdot = diag(diag(U'*E*V));
```

The one catch in all this is that A must be nonsingular, since zero singular values are non-differentiable (they locally look like an absolute value).

4: (20 points) Without using the MATLAB backslash, write a routine that solves $Ax = b$ where A is a symmetric positive definite tridiagonal.

```
function x = solve_tridiag(alpha, beta, b)
%
% Solve Ax = b where A = diag(alpha) + diag(beta,-1) + diag(beta,1)
```

Solution: Just do Gaussian elimination without pivoting to reduce the problem to $Ux = \hat{b}$, and solve by back substitution.

```
  n = length(alpha);
  x = b;
  for i = 1:n-1
    alpha(i+1) = alpha(i+1)-beta(i)^2/alpha(i);
    x(i+1)     = x(i+1)-beta(i)*x(i)/alpha(i);
  end
  for i = n:-1:2
    x(i) = x(i)/alpha(i);
    x(i-1) = x(i-1)-beta(i-1)*x(i);
  end
  x(1) = x(1)/alpha(1);
```

5: (20 points) Fill in the following routine

```
function A = generate_test(lambda, kappa)
%
% Produce a randomly-generated test matrix A = V Lambda V^{-1}
% where Lambda = diag(lambda) and cond(V) = kappa.
```

Solution: If one normalizes V so that the columns have unit length in the two-norm, this is a little tricky – more so than I would like for a two-hour exam. I hadn't thought about the normalization issue initially. Still, it's a good problem, and here is one solution.

```
  n = length(lambda);

  % Generate a random spd H with ones on the diagonal
  % and cond(H) = kappa^2.
  %
  H = randn(n);
  H = (H+H')/2;
  H = H-diag(diag(H));
  lH   = eig(H);
  lmin = min(lH);
  lmax = max(lH);
  tau  = (lmax-kappa^2*lmin)/(kappa-1)/(kappa+1);
  H    = H/tau+eye(n);

  % The columns of the Cholesky factor will have unit norm,
  % and cond(R) = kappa.  Scramble R with an orthogonal
  % matrix to get V = QR.
  %
  R = chol(H);
  [Q,RR] = qr(randn(n));

  A = Q*(R*diag(lambda)/R)*Q';
```

6: (20 points) Fill in the following routine. Your routine should not use `schur`, `eig`, or any other MATLAB functions beyond basic arithmetic and matrix operations.

```
function [Q,T] = schur2(A)
%
% Compute a Schur factorization of the 2-by-2 matrix A.
```

Solution: I guess you could do your own QR iteration, but I had in mind a closed form solution. All you really need to do is to compute an eigenvector of A; then append a vector orthonormal to that, and you have Q.

```
  % Coefficients and discriminant of char poly
  trAd2 = (A(1,1) + A(2,2))/2;
  detA  = A(1,1)*A(2,2)-A(1,2)*A(2,1);
  discr = trAd2^2-detA;

  % If eigenvalues are real, do something
  if discr >= 0
    lambda = trAd2 + sqrt(discr);

    % Compute an eigenvector (null vector of A-lambda*I)
    v1 = [A(1,2); lambda-A(1,1)];  nv1 = norm(v1);
    v2 = [lambda-A(2,2); A(2,1)];  nv2 = norm(v2);
    if nv1 > nv2
      v = v1/nv1;
    elseif nv2 > 0
      v = v2/nv2;
    else
      v = [1; 0];
    end

    % Construct an orthogonal matrix with v as first column.
    Q = [v(1), -v(2); v(2), v(1)];
  else
    Q = eye(2);
  end
  T = Q'*A*Q;
```