INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY HYDERABAD

Spring Semester, 2018
Subject: Introduction to Parallel Scientific Computing (CSE504)

Tutorial II
March 10, 2018

Due: 23.03.18 **Instructor: Dr. Pawan Kumar** Maximum Marks: 45

INSTRUCTIONS:

The codes can be written in either C, C++, MATLAB, Octave (Open source clone of MATLAB), FORTRAN, or Python. However, it is also recommended to write these codes in C/C++. Usually, if time permits, it is a good practice to write two versions of the same algorithm, first in MATLAB/Octave or Python, and then convert the same to C/C++. Write useful comments, and do proper indentation.

For testing, write a code that generates random matrix of desired size. In MATLAB, this can be done with `A = rand(m,n)`.

The questions prefixed with * are "somewhat" challenging, they may not be considered for grading, however, they may be seen as a sign of interest or motivation for working in this field in future. You may use MATLAB or other resources in ABACUS HPC system to write and run your codes.**

Please consult TAs if you have any doubts. In case you are busy, you may submit this assignment after deadline with a penalty of 10 percent.

Have fun with matrix algorithms!

1. (**Orthogonalization**)

   1. Given a set of linearly independent vectors $\{v_1, v_2, \cdots, v_m\}$, the Gram-Schmidt algorithm produces orthonormal vectors $\{q_1, q_2, \cdots, q_m\}$. In the Algorithm 1 below, $\{v_1, v_2, \cdots, v_m\}$ are overwritten by $\{q_1, q_2, \cdots, q_m\}$. Write a C/C++/Matlab/Python code for that implements this algorithm.
   2. (a) In this part of the exercise, we would like to verify whether Gram-Schmidt equals QR. Let $v_1 = [3, -3, 3, -3]^T$, $v_2 = [1, 2, 3, 4]^T$, and $S = \text{span}\{v_1, v_2\} \in \mathbb{R}^4$. Apply the Gram-Schmidt process (i.e., Algorithm 1 below) to S to obtain an orthonormal basis of S. Save the coefficients r_{jk}.

      (b) Let

$$A = \begin{bmatrix} 3 & 1 \\ -3 & 2 \\ 3 & 3 \\ -3 & 4 \end{bmatrix}.$$

Algorithm 1 Gram-Schmidt, $[q_1, q_2, \cdots, q_m] = \text{gram\_schmidt}(v_1, v_2, \cdots, v_m)$

```
for k = 1 to m do
    for i = 1 to k − 1 do
        r_ik ← ⟨v_k, v_i⟩   (= v_i^T v_k)
    end for
    for i = 1 to k − 1 do (skipped when k = 1)
        v_k ← v_k − v_i r_ik
    end for
    r_kk ← ‖v_k‖_2
    if r_kk = 0 then
        Set flag that v_1, v_2, ··· , v_k are dependent, exit
    end if
    v_k ← (1/r_kk) v_k
end for
```

Use the result of part (a) to build an isometric (A square orthonormal matrix is called isometry if some of the columns are removed) $Q \in \mathbb{R}^{4\times 2}$, and an upper triangular $R \in \mathbb{R}^{2\times 2}$ such that $A = QR$.

3. In the computation of $\hat{q}_k$, (see class notes), the classical Gram-Schmidt process calculates all of the coefficients r_{ik} at once, then makes the following update

$$\hat{q}_k = v_k - \sum_{i=1}^{k-1} r_{ik} q_i$$

all at once. The modified Gram-Schmidt process computes the coefficients all at one time. As soon as $r_{1k} = \langle v_k, q_1 \rangle$ has been computed, it is used to update v_k to obtain $v_k^{(1)}$.

$$v_k^{(1)} = v_k - r_{1k} q_1.$$

With this choice $v_k^{(1)}$ is orthogonal to q_1 (Why? Hint: Class notes.). Next the coefficient r_{2k} is computed using $v_k^{(1)}$ instead of v_k. That is, we compute $r_{2k} = \langle v_k^{(1)}, q_2 \rangle$. Then we do another update to obtain $v_k^{(2)}$ by removing appropriate component from $v_k^{(1)}$:

$$v_k^{(2)} = v_k^{(1)} - r_{2k} q_2$$

Here $v_k^{(2)}$ is such that it is now orthogonal to both q_1 and q_2 (How?). Now $v_k^{(2)}$ is used instead of v_k to compute r_{3k}, and so on. Continuing this way, after $k-1$ such updates, we have $v_k^{(k-1)}$ which is orthogonal to $q_1, \cdots, q_{k-1}$. We then obtain q_k by orthonormalizing $v_k^{(k-1)}$. That is, $q_k = v_k^{(k-1)} / \|v_k^{(k-1)}\|$. The full steps to obtained modified Gram-Schmidt algorithm is shown in Algorithm 2. Comparing with classical Gram-Schmidt, we notice that the two inner i loops have been condensed to a single loop. It is thus obvious that the two algorithms have the same flop count. In practice, modified version incurs less round-off error.

2. **(Algorithms for Computing Eigenvalues and Eigenvectors)** The eigenvalues and eigenvectors are like DNA of a matrix. They reveal most about the action of the matrix on a vector. Let

Algorithm 2 Modified Gram-Schmidt, $[q_1, q_2, \cdots, q_m] = \text{modified\_gram\_schmidt}(v_1, v_2, \cdots, v_m)$

```
for k = 1 to m do
    for i = 1 to k − 1 do
        r_ik ← ⟨v_k, v_i⟩
        v_k ← v_k − v_i r_ik
    end for
    r_kk ← ‖v_k‖_2
    if r_kk = 0 then
        Set flag that v_1, v_2, ··· , v_k are dependent, exit
    end if
    v_k ← (1/r_kk) v_k
end for
```

$\lambda \in \mathbb{C}, v \in \mathbb{C}^n, v \neq 0$, then λ and v are called eigenvalue and eigenvector pair if the following holds:

$$Av = \lambda v.$$

All the eigenvalues of a matrix can be computed by finding the roots of the characteristic equation:

$$\det(A - \lambda I) = 0$$

This is a polynomial of degree n, and the roots are the eigenvalues. Once the eigenvalues are found, eigenvectors can be found by solving the equation $(A - \lambda I)x = 0$ for x for all λ.

1. **(Basic Facts)**
   (a) Show that the characteristic polynomial of

   $$B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

   is $\lambda^2 - 5\lambda - 2$, a polynomial of degree 2.
   (b) Show that the characteristic polynomial of

   $$B = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 3 & 4 \\ 0 & 1 & 1 \end{bmatrix}$$

   is $\lambda^3 - 5\lambda + \lambda + 1$, a polynomial of degree 3.
   (c) We see that eigenvalues are given by the roots of a polynomial equation. We also recall that there is no general formula to find the roots of a general polynomial of degree 5 or higher. Given a general fifth degree polynomial $ax^5 + bx^4 + cx^3 + dx^2 + ex + f = 0$, write a matrix whose characteristic equation is this polynomial.
2. **(Algorithm to find eigenvalues)** When thinking about algorithms for computing eigenvalue, the concept of similar matrices is useful. Two matrices A and B are called similar when there exists a non-singular matrix S such that $S^{-1}AS = B$. We have shown

in the class that two similar matrices have same eigenvalues, and eigenvectors are also obtained easily (How?). Now, if B happens to be a matrix for which it is easy to compute eigenvalue, for example, if B happens to be a triangular (lower or upper or diagonal), then eigenvalues are on the diagonal, which will be required eigenvalues of A. Hence, for computing eigenvalues, we look for suitable similarity transforms such that the given matrix is transformed into a matrix whose eigenvaues are easier to find. To this end, we recall the Schur theorem (class notes) that states that for any $A \in \mathbb{C}^{n\times n}$, there exists a unitary matrix $U \in \mathbb{C}^{n\times n}$, and an upper triangular matrix $T \in \mathbb{C}^{n\times n}$ such that $T = U^*AU$. The proof done in class was not a constructive proof (What is a constructive proof, and why the proof done in class was not constructive?). Later, we discussed a way to at least reduce the matrix A to upper Hessenberg form by applying similarity unitary transform. Then the eigenvalue of the Hessenberg matrix will be the same as eigenvalues of the given matrix A. We sketch the basic ideas for transforming the matrix A to Hessenberg matrix as follows. Partition A as follows:

$$A = \begin{bmatrix} a_{11} & c^T \\ b & \hat{A} \end{bmatrix}$$

Let $\hat{Q}_1$ be a reflector such that $\hat{Q}_1 b = [-\tau_1, 0, \cdots, 0]^T$, where $\tau = \|b\|_2$, and let

$$Q_1 = \begin{bmatrix} 1 & 0^T \\ 0 & \hat{Q}_1 \end{bmatrix}.$$

Let

$$A_{1/2} = Q_1 A = \begin{bmatrix} a_{11} & c^T \\ \tilde{b} & \hat{Q}_1\hat{A} \end{bmatrix}$$

where $\tilde{b} = [-\tau_1, 0, \cdots, 0]^T$. Thus, $A_{1/2}$ has the desired zeros in the first column. This is same as the QR decomposition, except that we ignored the 1st entry of the first column of A, i.e., we we rotated $A(2:n, 1)$ instead of $A(1:n, 1)$ as we do in QR. Since, we are interested in similarity transform of the matrix so that eigenvalues are preserved, we multiplty Q_1^* from right to $A_{1/2}$ to obtain $A_1 = A_{1/2}Q_1^* = Q_1AQ_1^*$. Since Q is a householder matrix $Q^* = Q$ (How?). Note that because of the form of Q_1, this does not destroy the zeros in the first column of $A_{1/2}$ when it is multiplied from right by Q_1^* to obtain A_1 as follows:

$$A_1 = \begin{bmatrix} a_{11} & c^T \\ \tilde{b} & \hat{Q}_1\hat{A}\hat{Q}_1^* \end{bmatrix}.$$

Let $\hat{A}_1 = \hat{Q}_1\hat{A}\hat{Q}_1^*$. Then we consider $Q_2 \in \mathbb{C}^{n\times n}$ as below

$$Q_2 = \begin{bmatrix} I_{2\times 2} & 0_{2\times(n-2)} \\ 0_{(n-2)\times 2} & \hat{Q}_2 \end{bmatrix},$$

where $0_{m\times n} \in \mathbb{R}^{m\times n}$ denotes a zero matrix, and $I_{m\times n} \in \mathbb{R}^{m\times n}$ denotes identity matrix. Here $\hat{Q}_2$ is a householder matrix that rotates the vector $\hat{A}_1(2:n, 1)$, or in other words, it rotates the vector $A_1(3:n, 2)$ We now apply Q_2 on both sides of A_1 to get $A_2 = Q_2A_1Q_2^*$. This process continues, and finally $A_{(n-1)}$ is a Hessenberg matrix. The full steps are shown in Algorithm 3 below. In Algorithm 3, Q matrix is overwritten, write an algorithm to form Q and verify that QAQ^* is a Hessenberg matrix. Write code that implements Algorithm 3.

Algorithm 3 Hessenberg Reduction, H = hessenberg_reduction(A)

for $k = 1$ to $n-2$ **do**
  $\beta \leftarrow \max\{\, |A_{ik}| \quad | \quad i = k+1, \ldots, n\}$
  $\gamma_k \leftarrow 0$
  if $\beta \neq 0$ **then**
    // Set up the reflector
    $A(k+1:n,k) \leftarrow \beta^{-1} A(k+1:n,k)$
    $\tau_k \leftarrow \sqrt{A(k+1,k)^2 + \cdots + A(n,k)^2}$
    if $A(k+1,k) < 0$ **then**
      $\tau_k = -\tau_k$
    end if
    $\eta \leftarrow A(k+1,k) + \tau_k$
    $A(k+1,k) \leftarrow 1$
    $A(k+2:n,k) \leftarrow A(k+2:n,k)/\eta$
    $\gamma_k \leftarrow \eta/\tau_k$
    $\tau_k \leftarrow \tau_k \beta$
    // Multiply on the left by $\hat{Q}_k$
    $b(k+1:n,1) \leftarrow A(k+1:n,k)^T A(k+1:n,k+1:n)$
    $b(k+1:n,1) \leftarrow -\gamma_k b^T_{k+1:n,1}$
    $A(k+1:n,k+1:n) \leftarrow A(k+1:n,k+1:n) + A(k+1:n,k) b^T(k+1:n,1)$
    // Multiply on the right by $\hat{Q}_k$
    $b(1:n,1) \leftarrow A(1:n,k+1:n) A(k+1:n,k)$
    $b(1:n,1) \leftarrow -\gamma_k b(1:n,1)$
    $A(1:n,k+1:n) \leftarrow A(1:n,k+1:n) + b(1:n,1) A^T(k+1:n,k)$
    $A(k+1,k) = -\tau_k$
  end if
  $\tau_{n-1} \leftarrow -A(n,n-1)$
end for

3. We have done similarity transforms on the given matrix A to obtain a Hessenberg matrix H, hence, the eigenvalues of A are the eigenvalues of H. There is no direct way to transform H to an upper or lower or diagonal matrix using a direct approach. We need help from another algorithm called QR algorithm. The QR algorithm is different from QR factorization. The QR algorithm is the two-step algorithm to compute eigenvalues of a general matrix. The algorithm is as follows:

   - Do QR factorization of A : $Q^i R^i \leftarrow A^i$
   - Multiply R^i with Q^i obtained in 1st step $A^{i+1} \leftarrow R^i Q^i$

   After sufficient number of iterations, A^i converges to an upper triangular matrix (in practice, it reduces to block upper triangular, in that case, the eigenvalue computation proceeds to computing eigenvalues of the diagonal blocks (divide and conquer!)). Note that in general it is not possible that a QR algorithm will stop in finite number of steps, for if it always leads to an upper triangular matrix in a finite number of steps for any matrix of size larger than or equal to 5, then it would violate Abel's theorem! A final step in estimating eigenvalues of a matrix A is to run the QR algorithm on the Hessenberg matrix obtained in item above. To eliminate the entries below the diagonal, we use

rotation matrices

$$\hat{Q}_1^R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

to rotate a vector $x = [x_1, x_2]^T$ to $[\|x\|_2, 0]^T$. We notice that the corresponding $\cos(\theta)$ and $\sin(\theta)$ are given as follows:

$$\cos(\theta) = \frac{x_1}{\|x\|_2},$$
$$\sin(\theta) = \frac{x_2}{\|x\|_2}.$$

We then have $(\hat{Q}_1^R)^T x = [\|x\|_2, 0]^T$. The global rotation matrix (of dimension same as A) is

$$Q_1^R = \begin{bmatrix} \hat{Q}_1^R & \\ & I_{(n-1)\times(n-1)} \end{bmatrix},$$

and

$$(Q_1^R)^T H Q_1^R = \left[\begin{array}{c|c} \|H(1:2,1)\|_2 & * \\ \hline \tilde{\tau}_1 & \\ 0 & \tilde{H}_1 \\ \vdots & \\ 0 & \end{array}\right],$$

where we notice that zero at $(2,1)$ entry created when multiplying H from the left by $(Q_1^R)^T$ becomes non-zero (denoted as $\tilde{\tau}_1$) again when $(Q_1^R)^T H$ is multiplied from right by Q_1^R. This way even when one may apply series of rotators to H to make it upper triangular, but when H is multiplied from right by these rotators, the final matrix becomes upper Hessenberg matrix. If you combine the two steps of QR algorithm then we have: $A^{i+1} = Q^i A^i (Q^i)^T$. When applying QR algorithm on H, we have $H^{i+1} = Q^i H^i (Q^i)^T$. Thus, one may ask if one keeps getting Hessenberg matrix by using QR algorithm, then how does it help in finding the eigenvalue? Fortunately, after certain number of steps the matrix H^{i+1} approaches to an (block) upper triangular matrix, and the eigenvalues of the diagonal blocks are the (approximate) eigenvalues of A, because H^{i+1} is similar to A. So, to summarize, after obtaining Hesseinberg matrix using reflectors in item 2 above, you now run QR algorithm on the Hessenberg matrix. To do QR factorization required in first step of QR algorithm, you will use the rotators discussed above. Write a code that uses item 2 to transform a matrix into upper Heissenberg matrix, then applies QR algorithm on the Hessenberg matrix to find all eigenvalues of the given matrix A. **NOTE:** Running QR algorithm on Hessenberg matrix does not directly lead to upper triangular matrix, in practice, we get one eigenvalue at a time. Also, to speedup the convergence of eigenvalues, we will choose shifts. So, here are the final steps:

(a) Transform a given matrix A into upper hessenberg matrix H

(b) Run QR algorithm on H. Set $H^1 = H$. Run the following i-loop:

   i. Choose shift $\sigma = H^i(n,n)$. Set $H^i = H^i - \sigma I$. (Each eigenvalue is shifted by σ!)

ii. $Q_iR_i \leftarrow H^i$ (Do QR Factorization of H^i)

iii. $H^i \leftarrow R_iQ_i$ (Overwrite H^i by R_iQ_i)

iv. $H^i = H^i + \sigma I$ (Eigenvalue shifted back to original eigenvalues!)

v. If $H^i(n, n-1)$ is close to zero, declare $H^i(n, n)$ as eigenvalue. It may happen that some other entry (instead of $H^i(n, n-1)$) of subdiagonal (subdiagonal is the diagonal below main diagonal) of H^i becomes zero (despite choosing $H^i(n, n)$ as shift (why?)), in that case, if the entry of subdiagonal say $H^i(k, k-1)$ goes to zero somewhere in the middle for some k, $1 < k < n-1$, then the required eigenvalues of the matrix will be the eigenvalues two (splitted) diagonal blocks, i.e., of the subblocks $H^i(1 : k-1, 1 : k-1)$ and $H^i(k : n, k : n)$ of matrix H^i. To compute the eigenvalues, the QR iteration is now done on each of these blocks separately. Note that each of these diagonal blocks are also Hessenberg. Note that any number below 10^{-15} can be declared to be zero in double precision arithmetic.

vi. Go to step (b) to compute remaining eigenvalues, by applying QR algorithm on the remaining blocks.

NOTE: For a worked out example, and matlab script, please refer to the demo done in class.

4. Given a large sparse matrix A, write a code that implements power method to compute the largest eigenvector of a matrix A. Recall that in a power method algorithm in the following.

Algorithm 4 (Power Method), $q = \text{power\_method}(A)$, $A \in \mathbb{R}^{n\times n}, q \in \mathbb{R}^n$

Choose $q^0 \in \mathbb{R}^n$, a random vector
Choose m, the maximum number of iterations of power method
for $k = 1$ to m **do**
    Set $q^k \leftarrow Aq^{k-1}$
    Normalize q^k : overwrite q^k with $q^k/\|q^k\|$
    if $\|q^k - q^{k-1}\| < 14^{-10}$ **then**
        return q^k as output
    end if
end for

Since the input matrix is large and sparse (for example, Google page rank matrix), use CSR format to store the matrix, and to compute CSR-matrix times vector product (Refer Assignment-1 for CSR matrix).