ETH zürich

P. Koumoutsakos
ETH Zentrum, CLT E 13
CH-8092 Zürich

High Performance Computing for Science and Engineering I

Fall semester 2018

Set 5 - Power Method, BLAS/LAPACK, OpenMP

Issued: October 26, 2018
Hand in (optional): November 02, 2018 23:59

Question 1: Power Method

Total: 36 points (10+12+8+6)

The power method is an iterative technique for locating the dominant (largest) eigenvalue of a matrix. In addition, the power method also computes the associated eigenvector.

Consider the symmetric $N \times N$ matrix A, where the diagonal elements are given by $A[i,i] = \alpha i$ for $i = 1 \dots N$ and the off-diagonal elements are random numbers drawn from a uniform distribution $\mathcal{U}[0,1]$ where $A[i,j] = A[j,i]$ for all $i \neq j$. Unless noted otherwise, use $\alpha \in \{1/8, 1/4, 1/2, 1, 3/2, 2, 4, 8, 16\}$ and $N = 1024$. Additionally, all results should be computed in double precision.

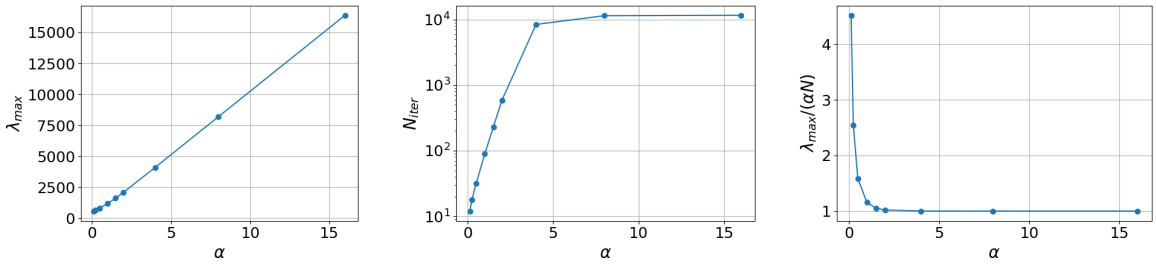
The power method produces a sequence of column vectors $\boldsymbol{q}^{(k)} \in \mathbb{R}^{N \times 1}$ given by

$$\boldsymbol{q}^{(k+1)} = \frac{A\boldsymbol{q}^{(k)}}{\|A\boldsymbol{q}^{(k)}\|_2} \tag{1}$$

If $\boldsymbol{q}^{(0)}$ is not deficient and the largest eigenvalue of A is unique, then $\boldsymbol{q}^{(k)}$ will converge to an eigenvector with eigenvalue $\lambda^{(k)}$.

a) Implement your own matrix multiplication program to calculate the dominant eigenvalue of the matrix A using the power method. Stop at the k-th iteration if the condition $|\lambda^{(k)} - \lambda^{(k-1)}| < 10^{-12}$ is satisfied. Use $\boldsymbol{q}^{(0)} = [1, 0, 0, \dots]^T$ as the initial guess. Report the following:
   i) The dominant eigenvalue for all values of α.
   ii) The smallest and largest iteration numbers for a converged solution using the set of matrices computed with the corresponding α values. Report the α values that correspond to the smallest and largest iteration numbers as well.

Fig. 1a shows the dominant eigenvalues as a function of α. Fig. 1b shows the number of iterations required for the algorithm to converge, also as a function of α. We observe that as α increases, the Power method requires more iterations to converge. The reason behind this is that as α increases, the matrix A becomes diagonally dominant, since the off-diagonal elements always have values smaller than 1. In diagonal matrices, the diagonal elements correspond to the eigenvalues. We define a diagonal matrix with $D[i,i] = \alpha i$, similar to the diagonal elements in our symmetric matrix A. Since the convergence of the Power method depends on the ratio between the two largest eigenvalues (as this ratio approaches 1, the slower is the convergence of the algorithm;



(a) (b) (c)

Figure 1: Results for the power method using a manual implementation. (a) Maximum eigenvalue with respect to α. (b) Number of iterations with respect to α. (c) Maximum eigenvalue normalized with αN, with respect to α.

see subquestion (d) for details), the convergence for matrix D will be determined by the ratio $\lambda_{N-1}/\lambda_N = \alpha(N-1)/(\alpha N) = (N-1)/N$, where λ_N, λ_{N-1} correspond to the first and second largest eigenvalues. Therefore, the larger the dimension N is, the slower the convergence. To study the approximation of matrix D by the matrix A, we can study the ratio between their largest eigenvalues. In Fig. 1c we divide the maximum eigenvalues of A by the corresponding eigenvalues of D, $\lambda_{max,A}/\lambda_{max,D} = \lambda_{max,A}/(\alpha N)$. Indeed, we observe that as α increases the eigenvalue ratio approaches unity, denoting that the diagonal elements of A are large enough compared to the off-diagonal elements, such that A resembles a diagonal matrix.

Point distribution:

1. Computation and presentation of the correct dominant eigenvalues in a table or in a plot. **(2 points)**
2. Computation and presentation of (at least) the smallest and largest number of iterations and their corresponding α values in a table or in a plot. Note that the number of iterations might differ from the solution since this depends on the error definition and the chosen convergence tolerance. **(2 points)**
3. Implementation of the Power method and eigenvalue computation using two or more matrix-vector multiplication calls *inside the loop body*. **(2 points)**
4. Implementation of the Power method and eigenvalue computation using only a single matrix-vector multiplication call *inside the loop body*. **(4 points)**
5. Use a pointer swap when updating $\mathbf{q}^{(k)}$ at the end of each iteration, k. **(2 points)**

b) Instead of a manual matrix-vector multiplication, use the CBLAS routines to perform the matrix operations of the power method. Write a program that allocates and initializes the matrix A, and computes the largest eigenvalue for the different values of α. Report the following:

i) The dominant eigenvalue for all values of α. Report the smallest and largest iteration numbers for convergence and the corresponding α values as well.

ii) Compute the time-to-solution (time to converged solution) of the CBLAS implementation, t_{power}, and of the manual matrix-vector multiplication implementation (previous

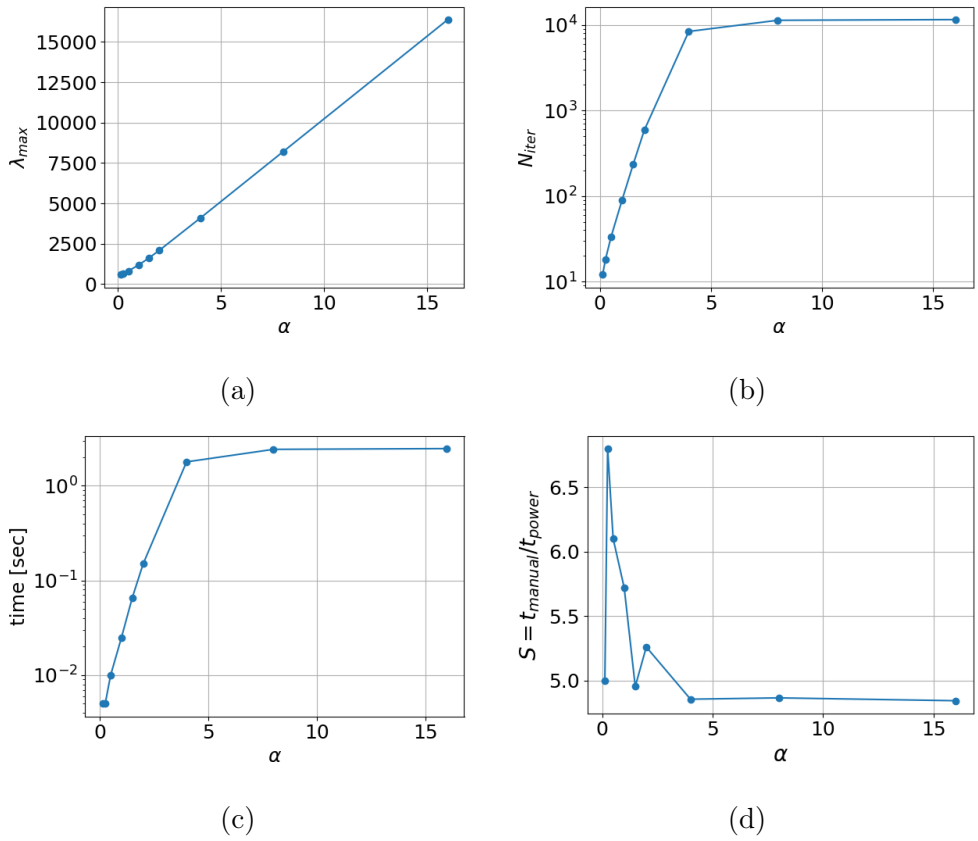


Figure 2: Results for the power method using a CBLAS implementation. (a) Maximum eigenvalue with respect to α. (b) Number of iterations with respect to α. (c) Execution time for the Power method using CBLAS. (d) Speedup of the CBLAS implementation over the manual implementation of the Power method.

subquestion), t_{manual}. Plot the speedup $S = t_{manual}/t_{power}$ as a function of α. If you observe a large speedup then examine your manual implementation and reason why.

iii) Report the time-to-solution for the CBLAS and the manual implementation for fixed $\alpha = 4$. Run the tests for the two matrix sizes $N = 4096$ and 8192.

Fig. 2a shows the dominant eigenvalues as a function of α. Fig. 2b shows the number of iterations required for the algorithm to converge, also as a function of α. Fig. 2d shows the speedup of the CBLAS implementation, compared to the manual implementation of the Power method. We observe that the optimized implementation of CBLAS is faster than the naive manual implementation of the Power method. This demonstrates the value of using highly optimized libraries when available. The oscillations observed for smaller α are due to the presence of other system processes. As the convergence of the power method becomes slower, the execution time becomes larger and the overhead from other processes is negligible. The time to solution for larger matrix sizes is shown in Table 1.

Point distribution:

1. Computation and presentation of the correct dominant eigenvalues in a table or in a plot. **(2 points)**

Table 1: Execution times for $\alpha = 4$ for larger matrix sizes.

	$N = 4096$	$N = 8192$
manual	318.47 sec	1516.23 sec
CBLAS	94.99 sec	449.94 sec
Speedup	3.35	3.37

2. Computation and presentation of (at least) the smallest and largest number of iterations and their corresponding α values in a table or in a plot. Note that the number of iterations might differ from the solution since this depends on the error definition and the chosen convergence tolerance. **(2 points)**
3. Computation and presentation of the speedup, $S = t_{manual}/t_{power}$, in a table or a plot. **(4 points)**
4. Implementation of the Power method and eigenvalue computation using five or more CBLAS function calls *inside the iteration loop*. **(3 points)**
5. Implementation of the Power method and eigenvalue computation using four or less CBLAS function calls *inside the iteration loop*. **(4 points)**

c) By using the Power method, we can compute only the eigenvector corresponding to the largest eigenvalue of a diagonalizable matrix A. In this subquestion you will solve the full eigenvalue problem by computing the eigenvalues of matrix A using an appropriate routine provided by the LAPACK library. The Intel Math Kernel Library (MKL) includes a high-performance implementation of both BLAS and LAPACK libraries. In order to use the MKL on Euler, you have to load the module with `module load mkl`. After loading the module, you can include the header `#include <mkl_lapack.h>` to access the LAPACK routines.

Write a program that allocates and initializes the same symmetric $N \times N$ matrix A as in the previous subquestions, and then calls the `LAPACKE_dsyev` routine of LAPACK to compute the full eigen solution of A. Report the following:

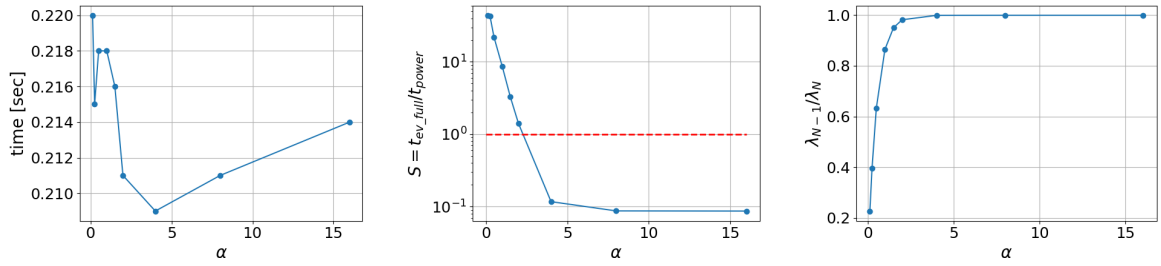
i) The **two** dominant eigenvalues for each α.

ii) The time required for your algorithm to converge, as a function of α.

iii) Compute the time-to-solution of the full eigen solution, t_{ev_full}. Plot the speedup $S = t_{ev\_full}/t_{power}$ as a function of α.

The values for the two dominant eigenvalues with respect to α can be found in the output file `lapack.txt`, produced by the solution code `eigenv_lapack.cpp`. Fig. 3a shows the execution time of the eigenvalue problem solution as a function of α. We observe that the time to solution of the full eigenvalue problem is not affected by the value of α (variations in the execution time are only about 5% of the mean execution time), as happened in the case of the Power method. In order to compare when the Power method is more efficient to use instead of solving the full eigenvalue problem, we plot the speedup between Power method in CBLAS and full eigen-problem solution in LAPACK, in Fig. 3b. We observe that when the Power method has a fast convergence (therefore at low values of α), it is more efficient than the full eigen-problem solution. However, when the Power method has a poor convergence, then the solution of the full eigenproblem is faster.

Point distribution:



(a) (b) (c)

Figure 3: Results of the full eigen-problem solution. (a) Execution time of the full eigenvalue solution using LAPACK with respect to α. (b) Speedup of the Power method (with CBLAS) compared to the solution of the full eigenvalue problem (with LAPACK). (c) Ratio between the two largest eigenvalues as a function of α.

1. Computation and presentation of the two dominant eigenvalues in a table or in a plot. **(2 points)**
2. Computation and presentation of the execution time as a function of α. **(2 points)**
3. Computation and presentation of the speedup, $S = t_{ev\_full}/t_{power}$, in a table or a plot. **(4 points)**

d) Prove on paper that the initial guess converges to the largest eigenvector. Comment on the convergence behavior of the Power method with respect to α and relate your explanation to the result of your proof.

Let the eigenvalues of A be $|\lambda_1| > |\lambda_2| \geq \ldots \geq |\lambda_n|$. We define a vector $\mathbf{b} = \sum_{i=1}^{n} c_i \mathbf{v}_i$ where $\mathbf{v}_i$ are the eigenvectors of A, and $c_1 \neq 0$. Then the Power method algorithm can be reformulated as

$$\mathbf{b}^{(k)} = \frac{A^k \mathbf{b}^{(0)}}{|A^k \mathbf{b}^{(0)}|} = \frac{A^k \mathbf{b}^{(0)}/\lambda_1^k}{|A^k \mathbf{b}^{(0)}|/\lambda_1^k}$$

By substituting $\mathbf{b}^{(0)}$ in $A^k \mathbf{b}^{(0)}/\lambda_1^k$ we get

$$A^k \mathbf{b}^{(0)}/\lambda_1^k = \sum_{i=1}^{n} c_i \frac{1}{\lambda_1^k} A^k \mathbf{v}_i = \sum_{i=1}^{n} c_i \left(\frac{\lambda_i}{\lambda_1}\right)^k \mathbf{v}_i$$

where we have used the fact that $A\mathbf{v} = \lambda\mathbf{v}$. The term $\left(\frac{\lambda_i}{\lambda_1}\right)$ is always less than one since the eigenvalues are sorted. Therefore as $k \to \infty$, $\left(\frac{\lambda_i}{\lambda_1}\right)^k \to 0$ for $i = 2, \ldots, n$ and the sum approaches $c_1 \mathbf{v}_1$.

The ratio between the two maximum eigenvalues is plotted in Fig. 3c. We observe as α becomes larger, the ratio of the two dominant eigenvalues approaches 1. This explains the poor convergence of the Power method for high α.

Point distribution:

1. Show that the Power method convergence depends on $\frac{\lambda_i}{\lambda_1}$. **(2 points)**
2. Explain the poor convergence of the Power method at large α by showing that as α increases, the ratio $\frac{\lambda_i}{\lambda_1}$ approaches 1. **(4 points)**

Question 2: OpenMP bug hunting

Total: 20 points (14+6)

a) Identify and explain any *bugs* in the following OpenMP code. Propose a solution. Assume all headers are included correctly.

```
1  // assume there are no OpenMP directives inside these two functions
2  void do_work(const float a, const float sum);
3  double new_value(int i);
4
5  void time_loop()
6  {
7      float t = 0;
8      float sum = 0;
9
10     #pragma omp parallel
11     {
12
13         for (int step=0; step<100; step++)
14         {
15             #pragma omp parallel for nowait
16             for (int i=1; i<n; i++) {
17                 b[i-1] = (a[i]+a[i-1])/2.;
18                 c[i-1] += a[i];
19             }
20
21             #pragma omp for
22             for (int i=0; i<m; i++)
23                 z[i] = sqrt(b[i]+c[i]);
24
25             #pragma omp for reduction(+:sum)
26             for (int i=0; i<m; i++)
27                 sum = sum + z[i];
28
29             #pragma omp critical
30             {
31                 do_work(t, sum);
32             }
33
34             #pragma omp single
35             {
36                 t = new_value(step);
37             }
38         }
39     }
40 }
```

1. The keywords *nowait* and *parallel* in the first for loop should be removed. **(2 points)**
2. The second and third for loops can be combined into a single for loop since the iteration space is the same. This would save the overhead for one `#pragma omp for`. **(2 points)**
3. The critical directive eliminates any parallelism and is not necessary since the arguments to *do_work* are passed by value and the function guarantees no OpenMP directives inside. **(4 points)**
4. A barrier must be placed after *do_work* as a thread could move on to update the value of *t* while other threads have not reached the function *do_work* yet. **(3 points)**
5. A *nowait* should be added to the *single* directive in line 34. **(2 point)**
6. The condition $m \leq n$ must hold. **(1 point)**

b) Identify and explain any *improvements* that can be made in the following OpenMP code. Propose a solution. Assume all headers are included correctly.

```
1  void work(int i, int j);
2
3  void nesting(int n)
4  {
5      int i, j;
6      #pragma omp parallel
7      {
8          #pragma omp for
9          for (i=0; i<n; i++)
10         {
11             #pragma omp parallel
12             {
13                 #pragma omp for
14                 for (j=0; j<n; j++) {
15                     work(i, j);
16                 }
17             }
18         }
19     }
20 }
```

1. Lines 6 and 8 can be combined into a single `#pragma omp parallel for`. This reduces the implicit barriers of these two lines from two to one. **(2 points)**
2. Lines 11 and 13 can be combined into a single `#pragma omp parallel for`. This reduces the implicit barriers of these two lines from two to one. **(2 points)**
3. Use only a single OpenMP directive (no nested parallelism) by using a collapse clause: `#pragma omp parallel for collapse(2)` **(6 points)**. In this case the code would look as follows:

```
1  void work(int i, int j);
2
3  void nesting(int n)
4  {
5      int i, j;
6      #pragma omp parallel for collapse(2)
7      for (i=0; i<n; i++) {
8          for (j=0; j<n; j++) {
9              work(i, j);
10         }
11     }
12 }
```