CPS343 Parallel and High Performance Computing Project 1 Spring 2022

Due: Monday February 21, 2022

For this project you will create two versions of a program written in C++ that uses the Power Method to estimate the dominant eigenvalue of a square diagonalizable matrix. In the first version each of the matrix and vector operations should be coded directly using for-loops, while in the second version you should use CBLAS routines. Please observe the following for both version:

1. The programs should accept the name of an HDF5 file containing the matrix data on the command line. The HDF5 file contains matrix data stored in row-major order with the path "`/A/value`". Data files containing different size matrices have been placed on the CS workstations in the directory `/gc/cps343/matrix`.

2. The programs should record and report: (1) the wall-clock time required to read the matrix from the file, and (2) the wall-clock time required to estimate the dominant eigenvalue. Wall-clock time (elapsed time) is returned by the `wtime()` function. To use this function, copy both `/gc/cps343/matrix/wtime.c` and `/gc/cps343/matrix/wtime.h` to your development directory and put the line

   ```
   #include "wtime.c"
   ```

   in your C++ code. You can determine the elapsed wall-clock time to "do some work" with the code sequence

   ```
   double t1 = wtime();
   // do some work
   double t2 = wtime();
   double elapsed_time = t2 - t1;
   ```

3. Use 1.0×10^{-6} for the tolerance between consecutive estimates of the eigenvalue to determine when the iteration has converged. Your program should stop iterating and provide a warning message that convergence was not achieved if it reaches 1000 iterations without converging.

4. A sample program written in Python 3 is provided in `/gc/cps343/matrix/pm_seq.py`. Use it as a baseline target for both the correct eigenvalues and expected computation times for the CBLAS version.

Bonus 1: (Additional 5%) Modify your programs so that they use the `getopt()` function to accept the tolerance and maximum number of iterations on the command line. You should use the same switches (`-e` and `-m`) and syntax as used in `/gc/cps343/matrix/pm_seq.py`; run `pm_seq.py` without any arguments to see the message that describes the arguments. Type `man 3 getopt` at a terminal prompt to read the getopt man page.

Bonus 2: (Additional 5%) The program with the fastest average estimation time over three consecutive runs on a test matrix no larger than 10000×10000 will receive this bonus.

What to turn in

Test your programs on all the matrix files found in `/gc/cps343/matrix/`. **Note:** several of the data files are quite large – *do not copy them to your directory.* It will considerably speed things up, however, if you do copy them to the `/tmp` directory and access them from there. Note that files in `/tmp` are erased when the computer reboots, so you may need to copy the files again at the start of each interactive session.

Record your results in a table, including data read times, computation times, number of power method iterations, and the eigenvalue estimates. You should also report the tolerance you used.

The Power Method algorithm

Given an $n \times n$ diagonalizable matrix A, a tolerance $\epsilon > 0$, and the maximum allowed number of iterations $M > 0$, the general power method algorithm can be formulated as follows.

$\mathbf{x} := (1, 1, 1, \cdots, 1)^T$	*initialize eigenvector estimate*
$\mathbf{x} := \mathbf{x}/\|\|\mathbf{x}\|\|$	*normalize eigenvector estimate*
$\lambda := 0$	*initialize eigenvalue estimate*
$\lambda_0 := \lambda + 2\epsilon$	*make sure* $\|\lambda - \lambda_0\| > \epsilon$
$k := 0$	*initialize loop counter*
while $\|\lambda - \lambda_0\| \geq \epsilon$ and $k \leq M$ do	
$\quad k := k + 1$	*update counter*
$\quad \mathbf{y} := A\mathbf{x}$	*compute next eigenvector estimate*
$\quad \lambda_0 := \lambda$	*save previous eigenvalue estimate*
$\quad \lambda := \mathbf{x}^T\mathbf{y}$	*compute new estimate:* $\lambda \approx \mathbf{x}^T A\mathbf{x}$
$\quad \mathbf{x} := \mathbf{y}$	*update eigenvector estimate*
$\quad \mathbf{x} := \mathbf{x}/\|\|\mathbf{x}\|\|$	*normalize eigenvector estimate*
end while	

If the while-loop terminates with $k \leq M$, then we conclude the algorithm has terminated successfully. In this case λ is the dominant eigenvalue and $\mathbf{x}$ is the corresponding normalized eigenvector.

The power method's rate of convergence depends on the difference between the magnitude of the dominant eigenvalue and the other eigenvalues. Also, the power method will fail if the matrix does not have any real eigenvalues.

Coding hints for both versions

Here are the basic structures you can use for the for-loop version of the program.

- The inner product (or dot product) $\mathbf{x}^T\mathbf{y}$ is computed with

```
double dot_product = 0.0;
for (int i=0; i<n; i++)
{
    dot_product += x[i] * y[i];
}
```

- The 2-norm $||\mathbf{x}||$ is computed with

```
double norm = 0.0;
for (int i=0; i<n; i++)
{
    norm += x[i] * x[i];
}
norm = sqrt(norm);
```

- The matrix-vector product $\mathbf{y} = A\mathbf{x}$, where A is $m \times n$, is computed with

```
for (int i=0; i<m; i++)
{
    y[i] = 0.0;
    for (int j=0; j<n; j++)
    {
      y[i] += a[i][j] * x[j];
    }
}
```

  Of course, the indexing for `a` will look different if the matrix A is stored in a 1-D array.

The CBLAS routines you will likely use for the second version include

- The inner product $\mathbf{x}^T\mathbf{y}$ can be computed with `cblas_ddot()`
- The 2-norm $||\mathbf{x}||$ can be computed with `cblas_dnrm2()`
- The matrix-vector product $\mathbf{y} = A\mathbf{x}$ can be computed with `cblas_dgemv()`
- A vector can be copied or scaled with `cblas_dcopy()` and `cblas_dscal()`, respectively.

Note: In order to use CBLAS in a C++ program you need to wrap the include statement for `cblas.h` inside an `extern` block:

```
extern "C" {
#include <cblas.h>
}
```