This assignment will require a formal technical report.

1 Matrix-Vector Product

The matrix-vector product is written symbolically as

$$y = Ax$$

where A is an $n \times m$ matrix, x is a column vector of length m and y is a column vector of length n. Each component of y is obtained via the formula

$$y_i = \sum_{j=1}^{m} a_{i,j}\, x_j, \quad i = 1, 2, \ldots, n.$$

One way to interpret this formula is to say that the ith component of y is the dot product of row i of A with the vector x. For example, let

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 4 & 2 \\ 6 & 3 & 5 \end{pmatrix}; \quad x = \begin{pmatrix} 1 \\ 3 \\ 2 \end{pmatrix}.$$

Then

$$\begin{aligned} y &= \begin{pmatrix} 1 & 2 & 3 \\ 1 & 4 & 2 \\ 6 & 3 & 5 \end{pmatrix} \begin{pmatrix} 1 \\ 3 \\ 2 \end{pmatrix} \\ &= \begin{pmatrix} 1(1) + 2(3) + 3(2) \\ 1(1) + 4(3) + 2(2) \\ 6(1) + 3(3) + 5(2) \end{pmatrix} \\ &= \begin{pmatrix} 13 \\ 17 \\ 25 \end{pmatrix}. \end{aligned}$$

The above algorithm is easily coded in FORTRAN90. Assuming the quantities A, x and y have the dimensions indicated above, then this can be accomplished as follows

```
!!!      Algorithm 1, Row Oriented   !!!

y = 0
DO i = 1,n
   DO j = 1,m
      y(i) = y(i) + A(i,j)*x(j)
   ENDDO
ENDDO
```

This is often called the row-based version of the algorithm because the matrix A can be considered to be partitioned by rows. It is also called the dot-product version because the inner j loop is computing the dot product of row i of the matrix A with the vector x.

2 Alternative Formulation

There is another interpretation to the matrix-vector product. In the expression

$$y = \begin{pmatrix} 1(1) + 2(3) + 3(2) \\ 1(1) + 4(3) + 2(2) \\ 6(1) + 3(3) + 5(2) \end{pmatrix}$$

from the example in Section 1, notice that the numbers in parentheses are the components of the vector x. This means that we can rewrite this expression in the following way:

$$\begin{aligned}
\begin{pmatrix} 1(1)+2(3)+3(2) \\ 1(1)+4(3)+2(2) \\ 6(1)+3(3)+5(2) \end{pmatrix} &= 1\cdot\begin{pmatrix} 1 \\ 1 \\ 6 \end{pmatrix} + 3\cdot\begin{pmatrix} 2 \\ 4 \\ 3 \end{pmatrix} + 2\cdot\begin{pmatrix} 3 \\ 2 \\ 5 \end{pmatrix} \\
&= \begin{pmatrix} 1 \\ 1 \\ 6 \end{pmatrix} + \begin{pmatrix} 6 \\ 12 \\ 9 \end{pmatrix} + \begin{pmatrix} 6 \\ 4 \\ 10 \end{pmatrix} \\
&= \begin{pmatrix} 13 \\ 17 \\ 25 \end{pmatrix}
\end{aligned}$$

Thus, the matrix-vector produce can be interpreted as a sum of scaled columns of the matrix A with the elements of x as the scaling factors. More precisely, this is a sum of `AXPY` operations.

This interpretation of the matrix-vector product operation is called the column-based or `AXPY` version. The algorithm for this version simply reverses the order of the loops:

```
!!!      Algorithm 2, Column Oriented   !!!

y = 0
DO j = 1,m
   DO i = 1,n
      y(i) = y(i) + A(i,j)*x(j)
   ENDDO
ENDDO
```

In this lab, you will experiment with these two algorithms. In particular, you will examine the differences in the computational performance of these algorithms and explain the differences you observe.

3 Assignment

1) Write a subroutine for the row-based matrix-vector product algorithm and a short driver program to test that it works. Your driver program should use dynamic allocation for your arrays and matrices. Use the matrix A and the vector x above as your test case.

2) Write a subroutine for the column-based algorithm and incorporate it into your driver routine from Part 1). Test it to confirm that both algorithms give you the same result for y.

3) Modify your driver to compute the matrix-vector product using the Level 2 BLAS `DGEMV` routine. Test it to make sure you get the same results as before (remember to include the `-lblas` flag when you compile your program).

4) How many floating point operations (FLOPS) are required to compute a matrix-vector product when A is $n \times n$? Theoretically, are all 3 routines doing the same amount of work?

5) Incorporate timing into your driver routine and time how long it takes each routine to execute (for the problem sizes used here, you should not have to do any averaging). Run the program for a value of $n = 5000$. Use the `RANDOM_NUMBER` subroutine to fill A and x with data values and be sure to run your program with the scheduler. Note that for this step, you should compile your code with the `-O0` flag (capital O followed by a zero), i.e.,

   ```
   pgf95 -O0 (.f90 codes) -lblas -o prog.exe
   ```

   What timing results do you obtain? Which routine is the fastest? How does the `DGEMV` time compare with the other routines? Based on what has been said in class regarding timing of routines and work being proportional to time, is this result surprising? Why?

6) Run your program for $n = 7500, 10000, 12500, 15000$. Use the resulting times to fit a correlation for each routine. Tabulate your correlation results. Are all the correlation powers the same? If not, which routine has the largest correlation power? Why is this difference in powers significant?

7) Use each correlation to predict the runtime for each routine for $n = 20000$. How accurately do your correlations predict the actual runtimes? What might be the reason for any differences you observe?

8) Compile the program using the `-O2` flag (using a capital O).

   ```
   $ pgf95 -O2 (.f90 files) -lblas -o prog.exe
   ```

   Run the program for a value $n = 5000$. How do your runtimes compare now? Based on the observed runtimes, what do you think the compiler did to your program (particularly the row version of the algorithm)?

   Repeat the correlation fit experiment from Step 6. Comment on any changes you observe in the correlation powers.

9) This last question is the key to the whole assignment. For the activities below, use the values of A and x from Section 1.

   a) Create a list of how the matrix A would be physically stored in memory.
   b) Manually trace through Algorithm 1 and create a list of how this algorithm accesses the elements of A.
   c) Manually trace through Algorithm 2 and create a list of how this algorithm accesses the elements of A.
   d) What is the difference in how the algorithms access the elements of A?
   e) Use the above results to explain why the Algorithm 2 is more computationally efficient than Algorithm 1.
   f) Discuss the value of a well written software library and the advantages that it can provide.

You should create a self-contained technical report (*i.e.,* all tables and figures should be formally inserted into your report). You should have tabulated computational timing results and figures of your correlation fit results. All of your figures and tables should be properly labeled and captioned with appropriate annotation. Include the final version of your program as an appendix.

Most of the credit from this assignment will come from the quality of your technical report and how well you answer the questions posed.