CAAM 419/519, Homework #3

wnl1

November 1, 2022

1 Diagonal Matrix

Listing 1 shows the general structure for a diagonal matrix.

Listing 1: Diagonal Matrix Output.

```
Running Diagonal Matrix...
DiagonalMatrix = [
   0.00
            2.00
                     4.00
                              6.00
                                       8.00
]
```

Consider a regular square matrix of size 100x100 with entry values of (i+j) in the diagonal and 0.0 elsewhere, where i is the row and j is the column index (i and j ranging from 0 - 99). Also consider a vector with entries [1 2 3 ... 99 100]. The norm of the difference between the product of the regular square matrix (storing values of zero) and vector and the product of a matrix storing only the diagonal of the regular square matrix (storing only the non-zero values) and the same vector is 0.0. This verifies the correctness of my diagonal matrix-vector multiplication implementation.

Listing 2 shows hows the diagonal matrix-vector multiplication is implemented. For diagonal matrices, the only non-zero entries are when the i index equal the j index. When multiplying a vector, the positional argument (i) is equivalent for the vector input and output. In fact, in this implementation of a diagonal matrix, the diagonal is represented by a (pointer) array. Thus, it is only necessary to iterate through the length of the diagonal. So, at each iteration, the i-th position of the output vector is the product of the i-th positions of the input vector and diagonal since the positional arguments of the multipliers are the same positional argument for their product.

Listing 2: Diagonal Matrix-Vector Multiplication Implementation.

```
void multiply_DiagonalMatrix_Vector(Vector* out, DiagonalMatrix* A, Vector* x){
  for (int i = 0; i < A->n; ++i){
    out->ptr[i] = A->ptr[i] * x->ptr[i];
  }
}
```

Considering a regular 100x100 square matrix and a 100x1 vector filled with zeros, matrix-vector multiplication of these requires 10,000 multiplication calls. However, with a diagonal matrix implementation, multiplication with the same vector only requires 100 (number of entries in a diagonal of a 100x100 matrix) multiplication calls. Thus, we would expect diagonal matrix-vector multiplication with the implementation seen in Listing 2 to be about 100 times faster than regular matrix-vector multiplication since the speed is mainly limited by multiplication. In fact, averaging from 100 samples of matrix-vector multiplications, diagonal matrix multiplication varies from 92.18 to 176.06 times faster than regular matrix multiplication.

2 Upper Triangular Matrix

Listing 3 shows the general structure for an upper triangular matrix.

Listing 3: Upper Triangular Matrix Output.

```
Running Upper Triangular Matrix...
UpperTriangularMatrix = [
   0.00     1.00     2.00     3.00     4.00
            2.00     3.00     4.00     5.00
```

```
                    4.00      5.00      6.00
                              6.00      7.00
                                        8.00
]
```

Consider a regular square matrix of size 100x100 with entry values of (i+j) in the upper triangle and 0.0 elsewhere, where i is the row and j is the column index (i and j ranging from 0 - 99). Also consider a vector with entries [1 2 3 ... 99 100]. The norm of the difference between the product of the regular square matrix (storing values of zero) and vector and the product of a matrix storing only the upper triangle of the regular square matrix (storing only the non-zero values) and the same vector is 0.0. This verifies the correctness of my upper triangular matrix-vector multiplication implementation.

Listing 4 shows hows the upper triangular matrix-vector multiplication is implemented. For upper triangular matrices, the only non-zero entries are when the i index is less than or equal to the j index. Thus, when we iterate through the entries row by row, we set the j index equal to the i index at the start of a new row to make sure that the j index is always greater than or equal to the i index. Since in the implementation of an upper triangular matrix, the non-zero entries are represented by a continuous (pointer) array, in this implementation of matrix-vector multiplication, the positional argument of the array is represented by i*(N - 1) - i*(i - 1)/2 + j where N is the size of the square matrix (100). The positional arguments of the input and output vectors are that of the regular matrix-vector multiplication.

Listing 4: Upper Triangular Matrix-Vector Multiplication Implementation.

```
void multiply_UpperTriangularMatrix_Vector(Vector* out, UpperTriangularMatrix* A,
                                            Vector* x){
  for (int i = 0; i < A->n; ++i){
    for (int j = i; j < A->n; ++j){
      out->ptr[i] += A->ptr[i*(A->n - 1) - i*(i - 1)/2 + j] * x->ptr[j];
    }
  }
}
```

Considering a regular 100x100 square matrix and a 100x1 vector filled with zeros, matrix-vector multiplication of these requires 10,000 multiplication calls. However, with an upper triangular matrix, multiplication with the same vector only requires 5050 (number of entries in the upper triangle of a 100x100 matrix) multiplication calls. Thus, we would expect upper triangular matrix-vector multiplication with the implementation seen in Listing 4 to be about 1.98 times faster than regular matrix-vector multiplication since the speed is mainly limited by multiplication. In fact, averaging from 100 samples of matrix-vector multiplications, upper triangular matrix multiplication varies from 1.77 to 1.87 times faster than regular matrix multiplication.

3 Tridiagonal Matrix

Listing 5 shows the general structure for a diagonal matrix.

Listing 5: Tridiagonal Matrix Output.

```
Running Tridiagonal Matrix...
TridiagonalMatrix = [
   0.00      1.00
   1.00      2.00      3.00
             3.00      4.00      5.00
                       5.00      6.00      7.00
                                 7.00      8.00
]
```

Consider a regular square matrix of size 100x100 with entry values of (i+j) in the tridiagonal and 0.0 elsewhere, where i is the row and j is the column index (i and j ranging from 0 - 99). Also consider a vector with entries [1 2 3 ... 99 100]. The norm of the difference between the product of the regular square matrix (storing values of zero) and vector and the product of a matrix storing only the tridiagonal of the regular square matrix (storing only the non-zero values) and the same vector is 0.0. This verifies the correctness of my tridiagonal matrix-vector multiplication implementation.

Listing 6 shows hows the tridiagonal matrix-vector multiplication is implemented. The implementation of a tridiagonal matrix considers each diagonal as its own (pointer) array, each of which is stored in another (pointer) array. The diagonals are split between the upper, middle, and lower diagonals. Inherently, the middle diagonal has one more entry than the upper and lower diagonals. Because of this, we deal with the last entry of the middle diagonal last, and iterate through the length of the upper and lower diagonals. First, let's focus on line 4 of Listing 6, which focuses on the middle diagonal. The multiplication of the middle diagonal is directly analogous to the diagonal matrix-vector multiplication. And outside the for loop in line 7, the last entry of the middle diagonal is taken into account for the matrix-vector multiplication. Now, in lines 3 and 5, the diagonals are offset. The upper diagonal is offset by one row and so the input vector positional argument is incremented to correct the multiplication. Similarly, the lower diagonal is offset by one column and so the output vector positional argument is incremented to correct the multiplication.

Listing 6: Tridiagonal Matrix-Vector Multiplication Implementation.

```
1 void multiply_TridiagonalMatrix_Vector(Vector* out, TridiagonalMatrix* A, Vector* x){
2   for (int i = 0; i < A->n - 1; ++i){
3     out->ptr[i] += A->ptr[0][i] * x->ptr[i+1];
4     out->ptr[i] += A->ptr[1][i] * x->ptr[i];
5     out->ptr[i+1] += A->ptr[2][i] * x->ptr[i];
6   }
7   out->ptr[A->n - 1] += A->ptr[1][A->n - 1] * x->ptr[A->n - 1];
8 }
```

Considering a regular 100x100 square matrix and a 100x1 vector filled with zeros, matrix-vector multiplication of these requires 10,000 multiplication calls. However, with a tridiagonal matrix, multiplication with the same vector only requires 298 (number of entries in the tridiagonal of a 100x100 matrix) multiplication calls. Thus, we would expect tridiagonal matrix-vector multiplication with the implementation seen in Listing 6 to be about 33.6 times faster than regular matrix-vector multiplication since the speed is mainly limited by multiplication. In fact, averaging from 100 samples of matrix-vector multiplications, tridiagonal matrix multiplication varies from 32.70 to 48.27 times faster than regular matrix multiplication.