CAAM 419/519, Homework #4

wnl1

December 2, 2022

1 main.cpp

Listing 1 shows the main.cpp file.

Listing 1: main.cpp file

```
#include <iostream>
#include "vector.h"
#include "matrix.h"

int main(void){
  Matrix<double> A(6,4);
  Matrix<double> B(4,5);
  Matrix<double> C(6,5);

  for (int i = 0; i < A.num_rows(); ++i){
    for (int j = 0; j < A.num_columns(); ++j){
      A[i][j] = (double) (i+j);
    }
  }
  for (int i = 0; i < B.num_rows(); ++i){
    for (int j = 0; j < B.num_columns(); ++j){
      B[i][j] = (double) 1 / (i + j + 1);
    }
  }
  for (int i = 0; i < C.num_rows(); ++i){
    for (int j = 0; j < C.num_columns(); ++j){
      C[i][j] = (double) i * j;
    }
  }

  Vector<double> x(5);
  for (int i = 0; i < x.length(); ++i){
    x[i] = i;
  }
  Vector<double> y(6);
  for (int i = 0; i < y.length(); ++i){
    y[i] = 1 - i;
  }

  double a = 1.5;

  A.print();
  B.print();
  C.print();
  x.print();
```

```
  y.print();
  std::cout << "a␣=␣" << a << std::endl;

  Vector<double> z = (A*B + C) * x + a * y;
  z.print();
  z = 3*z - (y - 1) / 2 + 0.5;
  z.print();
}
```

Using the following command shown in Listing 2, or running the Makefile, gives the output of the main.cpp file, as show in Listing 3.

Listing 2: Makefile command.

```
g++ main.cpp -I./include --std=c++11 -o main
```

Listing 3: main.cpp output

```
Matrix = [
0 1 2 3
1 2 3 4
2 3 4 5
3 4 5 6
4 5 6 7
5 6 7 8
]
Matrix = [
1 0.5 0.333333 0.25 0.2
0.5 0.333333 0.25 0.2 0.166667
0.333333 0.25 0.2 0.166667 0.142857
0.25 0.2 0.166667 0.142857 0.125
]
Matrix = [
0 0 0 0 0
0 1 2 3 4
0 2 4 6 8
0 3 6 9 12
0 4 8 12 16
0 5 10 15 20
]
Vector = [
0
1
2
3
4
]
Vector = [
1
0
-1
-2
-3
-4
]
a = 1.5
Vector = [
11.4286
47.9286
```

```
84.4286
120.929
157.429
193.929
]
Vector = [
34.7857
144.786
254.786
364.786
474.786
584.786
]
```

2 +=, *=, etc operators

Why "x += y" is more efficient than "x = x + y"?

"x = x + y" can be made more efficient by just incrementing the variable x directly. With "x = x + y", x and y are loaded into a function that allocates memory for their addition result, and then assigns the result to x. But, "x += y" makes better use of this because the += operator function directly increments x by the amount y, instead of allocating any memory. For the most part, this difference of memory allocation makes "x += y" more efficient than "x = x + y".

This response is extended to the reasoning why "x *= y" and similar operators are more efficient than their counterparts.