Chapter 1

Basic Arithmetic Operations on Vectors in C++ and R

The first thing to notice is that with functions in C++, the return type and argument type have to be declared in the function. Also note functions can have the same name with different arguments. In R, if x and y are vectors we can add the vectors with x + y. We can't do this in C++, at least not without overloading the + operator. Rcpp sugar makes this possible with a little bit of what seems like "magic". Rcpp sugar will be covered later.

1.1 Addition

Simple program to add two vectors and add a scalar to a vector

code/add_vec.cpp

```cpp
#include <vector>
#include <functional>
#include <iostream>

std::vector<double> add_vec(const std::vector<double>& v1,
        const std::vector<double>& v2){
    std::vector<double> v(v1.size());
    std::transform(v1.begin(), v1.end(), v2.begin(), v.begin(),
                std::plus<double>());
    return v;
}

std::vector<double> add_vec(const std::vector<double>& v1,
        const double& n){
        std::vector<double> v(v1.size());
        std::transform(v1.begin(), v1.end(), v.begin(),
                std::bind2nd(std::plus<double>(), n));
        return v;
}

int main(){
        std::vector<double> vec1(10);
        std::vector<double> vec2(10);

        // fill vec1 with 1 to 10
        // fill vec2 with 10 to 19
        for(int i = 0; i < vec1.size(); i++){
                vec1[i] = i + 1;
                vec2[i] = i + 10;
        }

        std::vector<double> vec3(10);
        vec3 = add_vec(vec1, vec2);

        std::vector<double> vec4(10);
        vec4 = add_vec(vec1, 5.5);


        std::cout << "Result of add_vec(vec1, vec2)" << std::endl;
        for(int i = 0; i < vec3.size(); i++){
                std::cout << vec3[i] << " ";
        }
        std::cout << std::endl;

        std::cout << "Result of add_vec(vec1, 5.5)" << std::endl;
        for(int i = 0; i < vec4.size(); i++){
                std::cout << vec4[i] << " ";
        }
        std::cout << std::endl;

        return 0;
}
```

1.2 Subtraction

Simple program to subtract two vectors and subtract a vector by a scalar

code/sub_vec.cpp

```
#include <vector>
#include <functional>
#include <iostream>

std::vector<double> sub_vec(const std::vector<double>& v1,
        const std::vector<double>& v2){
    std::vector<double> v(v1.size());
    std::transform(v1.begin(), v1.end(), v2.begin(), v.begin(),
                    std::minus<double>());
    return v;
}

std::vector<double> sub_vec(const std::vector<double>& v1,
        const double& n){
        std::vector<double> v(v1.size());
        std::transform(v1.begin(), v1.end(), v.begin(),
                        std::bind2nd(std::minus<double>(), n));
        return v;
}

```

```
int main(){
        std::vector<double> vec1(10);
        std::vector<double> vec2(10);

        // fill vec1 with 1 to 10
        // fill vec2 with 10 to 19
        for(int i = 0; i < vec1.size(); i++){
                vec1[i] = i + 1;
                vec2[i] = i + 10;
        }

        std::vector<double> vec3(10);
        vec3 = sub_vec(vec1, vec2);

        std::vector<double> vec4(10);
        vec4 = sub_vec(vec1, 5.5);

        std::cout << "Result of sub_vec(vec1, vec2)" << std::endl;
        for(int i = 0; i < vec3.size(); i++){
                std::cout << vec3[i] << " ";
        }
        std::cout << std::endl;

        std::cout << "Result of sub_vec(vec1, 5.5)" << std::endl;
        for(int i = 0; i < vec4.size(); i++){
                std::cout << vec4[i] << " ";
        }
        std::cout << std::endl;

        return 0;
}
```

1.3 Multiplication

Simple program to multiply two vectors and multiply a vector by a scalar

code/mult_vec.cpp

```
#include <vector>
#include <functional>
#include <iostream>

std::vector<double> mult_vec(const std::vector<double>& v1,
        const std::vector<double>& v2){
        std::vector<double> v(v1.size());
        std::transform(v1.begin(), v1.end(), v2.begin(), v.begin(),
                        std::multiplies<double>());
        return v;
}

std::vector<double> mult_vec(const std::vector<double>& v1,
        const double& n){
        std::vector<double> v(v1.size());
        std::transform(v1.begin(), v1.end(), v.begin(),
                        std::bind1st(std::multiplies<double>(), n));
        return v;
}

int main(){
        std::vector<double> vec1(10);
        std::vector<double> vec2(10);

        // fill vec1 with 1 to 10
        // fill vec2 with 10 to 19
        for(int i = 0; i < vec1.size(); i++){
            vec1[i] = i + 1;
            vec2[i] = i + 10;
        }

        std::vector<double> vec3(10);
        vec3 = mult_vec(vec1, vec2);

        std::vector<double> vec4(10);
        vec4 = mult_vec(vec1, 5.5);

        std::cout << "Result of mult_vec(vec1, vec2)" << std::endl;
        for(int i = 0; i < vec3.size(); i++){
            std::cout << vec3[i] << " ";
        }
        std::cout << std::endl;

        std::cout << "Result of mult_vec(vec1, 5.5)" << std::endl;
        for(int i = 0; i < vec4.size(); i++){
            std::cout << vec4[i] << " ";
        }
        std::cout << std::endl;

        return 0;
}
```

1.4 Division

Simple program to divide two vectors and divide a vector by a scalar

code/div_vec.cpp

```
#include <vector>
#include <functional>
#include <iostream>

std::vector<double> div_vec(const std::vector<double>& v1,
    const std::vector<double>& v2){
    std::vector<double> v(v1.size());
    std::transform(v1.begin(), v1.end(), v2.begin(), v.begin(),
                std::divides<double>());
    return v;
}

```

```
std::vector<double> div_vec(const std::vector<double>& v1,
        const double& n){
        std::vector<double> v(v1.size());
        std::transform(v1.begin(), v1.end(), v.begin(),
                        std::bind2nd(std::divides<double>(), n));
        return v;
}

int main(){
        std::vector<double> vec1(10);
        std::vector<double> vec2(10);

        // fill vec1 with 1 to 10
        // fill vec2 with 10 to 19
        for(int i = 0; i < vec1.size(); i++){
                vec1[i] = i + 1;
                vec2[i] = i + 10;
        }

        std::vector<double> vec3(10);
        vec3 = div_vec(vec1, vec2);

        std::vector<double> vec4(10);
        vec4 = div_vec(vec1, 5.5);

        std::cout << "Result of div_vec(vec1, vec2)" << std::endl;
        for(int i = 0; i < vec3.size(); i++){
                std::cout << vec3[i] << " ";
        }
        std::cout << std::endl;

        std::cout << "Result of div_vec(vec1, 5.5)" << std::endl;
        for(int i = 0; i < vec4.size(); i++){
                std::cout << vec4[i] << " ";
        }
        std::cout << std::endl;

        return 0;
}
```

Chapter 2

Computing the Sum and Mean of Vectors in C++ and R

2.1 Sum

Compute the sum of a vector. The sum1 function uses a for loop and the sum2 function uses std::accumulate from the STL.

code/vec_sum.cpp

```
#include <vector>
#include <numeric>
#include <iostream>

double sum1(const std::vector<double>& v1){
	double acc = 0;
	for(int i = 0; i < v1.size(); i++){
		acc += v1[i];
	}
	return acc;
}

double sum2(const std::vector<double>& v1){
	return std::accumulate(v1.begin(), v1.end(), 0.0);
}

int main(){
	std::vector<double> vec1(10);

	// fill vec1 with 1 to 10
	for(int i = 0; i < vec1.size(); i++){
		vec1[i] = i + 1;
	}

	double num1, num2;

	num1 = sum1(vec1);
	num2 = sum2(vec1);

	std::cout << "sum1(vec1) = " << num1 << std::endl;
	std::cout << "sum2(vec2) = " << num2 << std::endl;
```

```

        return 0;
}
```

2.2 Cumulative Sum

Compute the cumulative sum of a vector. The cumsum1 function uses a for loop whereas the cumsum2 function uses std::partial_sum from the STL.

code/vec_cumsum.cpp

```
#include <vector>
#include <numeric>
#include <iostream>

std::vector<double> cumsum1(const std::vector<double>& v1){
        std::vector<double> v(v1.size());
        v[0] = v1[0];
        double acc = 0;
        for(int i = 0; i < v.size(); i ++){
                acc += v1[i];
                v[i] = acc;
        }
        return v;
}

std::vector<double> cumsum2(const std::vector<double>& v1){
        std::vector<double> v(v1.size());
        std::partial_sum(v1.begin(), v1.end(), v.begin());
        return v;
}

int main(){
        std::vector<double> vec1(10);

        // fill vec1 with 1 to 10
        for(int i = 0; i < vec1.size(); i++){
                vec1[i] = i + 1;
        }

        std::vector<double> vec3(10);
        vec3 = cumsum1(vec1);

        std::vector<double> vec4(10);
        vec4 = cumsum2(vec1);

        std::cout << "Result of cumsum1(vec1)" << std::endl;
        for(int i = 0; i < vec3.size(); i++){
                std::cout << vec3[i] << " ";
        }
        std::cout << std::endl;

        std::cout << "Result of cumsum2(vec1)" << std::endl;
        for(int i = 0; i < vec4.size(); i++){
```

```
        std::cout << vec4[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

2.3 Mean

Compute the mean of a vector. Note how I use the same sum2 function that was defined earlier by copy/-pasting it into the file. In general, this is a bad thing to do, but I am doing this for simplicity and so each file can be run as a stand-alone program. The correct way would be to define the sum2 function in it's own .cpp file, then declare it in a header file, then include the header file.

code/vec_mean.cpp

```
#include <vector>
#include <numeric>
#include <iostream>


double sum2(const std::vector<double>& v1){
    return std::accumulate(v1.begin(), v1.end(), 0.0);
}

double mean(const std::vector<double>& v1){
    return sum2(v1) / (double)v1.size();
}

int main(){
    std::vector<double> vec1(10);

    // fill vec1 with 1 to 10
    for(int i = 0; i < vec1.size(); i++){
        vec1[i] = i + 1;
    }

    double num1;

    num1 = mean(vec1);

    std::cout << "mean(vec1) = " << num1 << std::endl;

    return 0;
}
```

Chapter 3

Applying Functions to Vectors in C++ and R

3.1 Square Root

Compute the square root of each element in the vector

code/vec_sqrt.cpp

```
#include <vector>
#include <algorithm>
#include <math.h>
#include <iostream>

std::vector<double> vec_sqrt(const std::vector<double>& v1){
        std::vector<double> v(v1.size());
        std::transform(v1.begin(), v1.end(), v.begin(), sqrt);
        return v;
}


int main(){
        std::vector<double> vec1(10);

        // fill vec1 with 1 to 10
        for(int i = 0; i < vec1.size(); i++){
                vec1[i] = i + 1;
        }

        std::vector<double> vec2(vec1.size());
        vec2 = vec_sqrt(vec1);

        std::cout << "Result of vec_sqrt(vec1)" << std::endl;
        for(int i = 0; i < vec2.size(); i++){
                std::cout << vec2[i] << " ";
        }
        std::cout << std::endl;

        return 0;
}
```

3.2 Square

The vec_squared function computes the square each element in the vector using a for loop. The vec_squared1 function computes the square each element in the vector using std::multiplies by multiplying the vector by itself.

code/vec_squared.cpp

```
#include <vector>
#include <algorithm>
#include <iostream>

std::vector<double> vec_squared(const std::vector<double>& v1){
        std::vector<double> v(v1.size());
        for(int i = 0; i < v1.size(); i++){
                v[i] = v1[i] * v1[i];
        }
        return v;
}

std::vector<double> vec_squared1(const std::vector<double>& v1){
        std::vector<double> v(v1.size());
        std::transform(v1.begin(), v1.end(), v1.begin(), v.begin(),
                        std::multiplies<double>());
        return v;
}

int main(){
        std::vector<double> vec1(10);

        // fill vec1 with 1 to 10
        for(int i = 0; i < vec1.size(); i++){
                vec1[i] = i + 1;
        }

        std::vector<double> vec3(10);
        vec3 = vec_squared(vec1);

        std::vector<double> vec4(10);
        vec4 = vec_squared1(vec1);

        std::cout << "Result of vec_squared(vec1)" << std::endl;
        for(int i = 0; i < vec3.size(); i++){
                std::cout << vec3[i] << " ";
        }
        std::cout << std::endl;

        std::cout << "Result of vec_squared1(vec1)" << std::endl;
        for(int i = 0; i < vec4.size(); i++){
                std::cout << vec4[i] << " ";
        }
        std::cout << std::endl;

        return 0;
}
```

3.3 Sum of Squares

Compute the sum of squares using std::inner_product

code/ssq.cpp

```
#include <vector>
#include <numeric>
#include <iostream>

double ssq(const std::vector<double>& v1){
        return std::inner_product(v1.begin(), v1.end(), v1.begin(), 0.0);
}

int main(){
        std::vector<double> vec1(10);

        // fill vec1 with 1 to 10
        for(int i = 0; i < vec1.size(); i++){
                vec1[i] = i + 1;
        }

        double num1;
        num1 = ssq(vec1);

        std::cout << "Result of ssq(vec1): " << num1 << std::endl;

        return 0;
}
```

3.4 Power

Compute the "n" power of each element in the vector.

code/vec_pow.cpp

```
#include <vector>
#include <functional>
#include <cmath>
#include <iostream>


std::vector<double> vec_pow(const std::vector<double>& v1, const double& n){
        std::vector<double> v(v1.size());
        for(int i = 0; i < v.size(); i++){
                v[i] = pow(v1[i], n);
        };
        return v;
}

int main(){
        std::vector<double> vec1(10);

        // fill vec1 with 1 to 10
        for(int i = 0; i < vec1.size(); i++){
                vec1[i] = i + 1;
        }

        std::vector<double> vec3(10);
        vec3 = vec_pow(vec1, 3.5);


        std::cout << "Result of vec_pow(vec1, 3.5)" << std::endl;
        for(int i = 0; i < vec3.size(); i++){
                std::cout << vec3[i] << " ";
        }
        std::cout << std::endl;

        return 0;
}
```

3.5 User Defined Function

Apply the user defined function to each element in the vector. Here I define a simple function to add 3 to a number.

code/vec_myfun.cpp

```
#include <vector>
#include <algorithm>
#include <math.h>
#include <iostream>

double add3(const double& x){
        return x + 3;
}
std::vector<double> vec_myfun(const std::vector<double>& v1){
        std::vector<double> v(v1.size());
        std::transform(v1.begin(), v1.end(), v.begin(), add3);
        return v;
}


int main(){
        std::vector<double> vec1(10);

        // fill vec1 with 1 to 10
        for(int i = 0; i < vec1.size(); i++){
                vec1[i] = i + 1;
        }

        std::vector<double> vec2(vec1.size());
        vec2 = vec_myfun(vec1);

        std::cout << "Result of vec_myfun(vec1)" << std::endl;
        for(int i = 0; i < vec2.size(); i++){
                std::cout << vec2[i] << " ";
        }
        std::cout << std::endl;

        return 0;
}
```

Chapter 4

Vector Slicing and Dicing in C++ and R

Maybe come up with a better, more technical, term for the title

4.1 Vector Sorting

Sort a vector in ascending order
sort a vector in descending order

4.2 Minimums and Maximums

Find the minimum value of the vector
Find the maximum value of the vector
TODO: runMin and runMax

4.3 Counting

Count the elements of the vector
Count the elements of a vector meeting a specified criteria

4.4 Slicing

Given a vector, return a new vector of elements meeting a specified criteria
Sum the elements of a vector meeting a specified criteria