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/addition.cpp

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

std::vector<double> add_vec(const std::vector<double>& v1, const std::vector<double>& v2){
        /* add two vectors together */
    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){
        /* add a scalar into a vector */
    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;
}
```

code/addition.R

```
# initialize two vectors
# fill vec1 with 1 to 10
# fill vec2 with 10 to 19
vec1 <- 1:10
vec2 <- 10:19

# add vec1 and vec2
vec3 <- vec1 + vec2

# add a scalar to a vector
vec4 <- vec1 + 5.5
```

1.2 Subtraction

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

code/subtraction.cpp

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

std::vector<double> sub_vec(const std::vector<double>& v1, const std::vector<double>& v2){
    /* subtract two vectors */
    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){
    /* subtract a vector by a scalar */
    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;
}
```

code/subtraction.R

```
# initialize two vectors
# fill vec1 with 1 to 10
# fill vec2 with 10 to 19
vec1 <- 1:10
vec2 <- 10:19

# subtract vec1 and vec2
vec3 <- vec1 - vec2

# subtract a vector by a scalar
vec4 <- vec1 - 5.5
```

1.3 Multiplication

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

code/multiplication.cpp

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

std::vector<double> mult_vec(const std::vector<double>& v1, const std::vector<double>& v2){
        /* multiply two vectors */
        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){
    /* multiply a scalar into a vector */
    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;
}
```

code/multiplication.R

```
# initialize two vectors
# fill vec1 with 1 to 10
# fill vec2 with 10 to 19
vec1 <- 1:10
vec2 <- 10:19

# multiply vec1 and vec2
vec3 <- vec1 * vec2

# multiply a vector by a scalar
vec4 <- vec1 * 5.5
```

1.4 Division

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

code/division.cpp

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

std::vector<double> div_vec(const std::vector<double>& v1, const std::vector<double>& v2){
    /* divide two vectors */
    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){
    /* divide a vector by a scalar */
    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;
}
```

code/division.R

```
# initialize two vectors
# fill vec1 with 1 to 10
# fill vec2 with 10 to 19
vec1 <- 1:10
vec2 <- 10:19

# divide vec1 and vec2
vec3 <- vec1 / vec2

# divide a vector by a scalar
vec4 <- vec1 / 5.5
```

Chapter 2

Computing the Sum 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/sum.cpp

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

double sum1(const std::vector<double>& v1){
        /* compute the sum of a vector */
        double acc = 0;
        for(int i = 0; i < v1.size(); i++){
                acc += v1[i];
        }
        return acc;
}

double sum2(const std::vector<double>& v1){
        /* compute the sum of a vector */
        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/cumsum.cpp

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

std::vector<double> cumsum1(const std::vector<double>& v1){
        /* compute the cumulative sum of a vector */
        std::vector<double> v(v1.size());
        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){
        /* compute the cumulative sum of a vector */
    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 Sum of Squares

Compute the sum of squares of a vector. This is done by taking the inner product of a vector with itself.

code/ssq.cpp

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

double ssq(const std::vector<double>& v1){
    /* compute the sum of squares of a vector */
    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;
}
```

2.4 R Code

code/sum.R

```r
# initialize a vector
# fill vec1 with 1 to 10
vec1 <- 1:10

# compute the sum of vec1
num1 <- sum(vec1)

```

```
# compute the cumulative sum of vec1
vec2 <- cumsum(vec1)

# compute the sum of squares of vec1
num2 <- sum(vec1 * vec1)

# compute the sum of squares using the inner product
# this actually makes num3 a 1x1 matrix
num3 <- vec1 %*% vec1
```

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/sqrt.cpp

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

std::vector<double> vec_sqrt(const std::vector<double>& v1){
        /* function to apply sqrt to each element of a vector */
        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/squared.cpp

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

std::vector<double> vec_squared(const std::vector<double>& v1){
	/* square each element of the vector */
	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){
    /* square each element of the vector */
    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 Power

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

code/pow.cpp

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


std::vector<double> vec_pow(const std::vector<double>& v1, const double& n){
        /* nth power of each element of a vector */
    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;
}
```

3.4 User Defined Function

We can also define functions that take a single argument. Apply the user defined function to each element in the vector. Here I define a simple function to add 3 to a number.

code/myfun.cpp

```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;
}
```

3.5 R code

code/applying.R

```r
# initialize a vector
# fill vec1 with 1 to 10
vec1 <- 1:10

# square each element in vec1
# multiply vec1 by itself
vec2 <- vec1 * vec1

# use the ^ operator to square each element in vec1
vec3 <- vec1^2

# function to add 3 to the argument
add3 <- function(x){
  x + 3
}

# pass vec1 as an argument to add3
add3(vec1)

# because the type of argument is not specified in R, I
# can also use this on a scalar
add3(5)
```

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 and sort a vector in descending order.
lstsetlanguage=C++

code/sort.cpp

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


void sort_a(std::vector<double>& v1){
        /* sort vector in ascending order */
        sort(v1.begin(), v1.end());
}

void sort_d(std::vector<double>& v1){
        /* sort vector in descending order */
        sort(v1.begin(), v1.end(), std::greater<double>());
}

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;
        }

        // use random_shuffle to change the order of the elements
        std::random_shuffle(vec1.begin(), vec1.end());

        std::cout << "shuffled vector" << std::endl;
        for(int i = 0; i < vec1.size(); i++){
                std::cout << vec1[i] << " ";
        }
        std::cout << std::endl;

        /* vec1 sorted in ascending order */
        sort_a(vec1);

        std::cout << "sorted vector" << std::endl;
        for(int i = 0; i < vec1.size(); i++){
                std::cout << vec1[i] << " ";
        }
        std::cout << std::endl;

        /* vec1 sorted in descending order*/
        sort_d(vec1);

        std::cout << "reversed vector" << std::endl;
        for(int i = 0; i < vec1.size(); i++){
                std::cout << vec1[i] << " ";
        }
        std::cout << std::endl;

        return 0;
}
```

code/sort.R

```
# initialize a vector
# fill vec1 using rnorm
vec1 <- rnorm(10)

# sort vec1 in ascending order
sort(vec1, decreasing = FALSE)

# sort vec1 in ascending order
sort(vec1, decreasing = TRUE)
```

4.2 Counting

Count the elements of the vector. This is very simple with the v.size() member function.

Count the elements of a vector meeting a specified criteria. I don't define any functions here, just demonstrating the different uses for the count_if function.

lstsetlanguage=C++

code/countif.cpp

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

bool is_even(int x){
        return ((x % 2) == 0);
}
```

```

bool is_odd(int x){
        return ((x % 2) == 1);
}

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;
        }

        int count1, count2, count3, count4, count5, count6, count7, count8;
        int num = 5;

        count1 = count_if(vec1.begin(), vec1.end(), bind2nd(std::less<double>(), num));
        count2 = count_if(vec1.begin(), vec1.end(), bind2nd(std::less_equal<double>(), num));

        count3 = count_if(vec1.begin(), vec1.end(), bind2nd(std::greater<double>(), num));
        count4 = count_if(vec1.begin(), vec1.end(), bind2nd(std::greater_equal<double>(), num));

        count5 = count_if(vec1.begin(), vec1.end(), bind2nd(std::equal_to<double>(), num));
        count6 = count_if(vec1.begin(), vec1.end(), bind2nd(std::not_equal_to<double>(), num));

        count7 = count_if(vec1.begin(), vec1.end(), is_even);
        count8 = count_if(vec1.begin(), vec1.end(), is_odd);

        std::cout << "There are: " << std::endl;
        std::cout << count1 << " numbers less than " << num << std::endl;
        std::cout << count2 << " numbers less than or equal to " << num << std::endl;
        std::cout << count3 << " numbers greater than than " << num << std::endl;
        std::cout << count4 << " numbers greater than or equal to " << num << std::endl;
        std::cout << count5 << " numbers equal to " << num << std::endl;
        std::cout << count6 << " numbers not equal to " << num << std::endl;

        std::cout << "There are " << count7 << " even numbers" << std::endl;
        std::cout << "There are " << count8 << " odd numbers" << std::endl;

        return 0;
}
```

code/countif.R

```
# initialize a vector
# fill vec1 with 1 to 10
vec1 <- 1:10

# length of vec1
len1 <- length(vec1)

num <- 5

# count the elements of vec1 that are less than num
count1 <- length(vec1[vec1 < num])
```

```

# count the elements of vec1 that are less than or equal to num
count2 <- length(vec1[vec1 <= num])

# count the elements of vec1 that are greater than num
count3 <- length(vec1[vec1 > num])

# count the elements of vec1 that are greater than or equal to num
count4 <- length(vec1[vec1 >= num])

# count the elements of vec1 that are equal to num
count5 <- length(vec1[vec1 == num])

# count the elements of vec1 that are not equal to num
count6 <- length(vec1[vec1 != num])

# count the elements of vec1 that are even
count7 <- length(vec1[(vec1 %% 2) == 0])

# count the elements of vec1 that are odd
count8 <- length(vec1[(vec1 %% 2) != 0])

cat("There are: ", "\n")
cat(count1, "numbers less than", num, "\n")
cat(count2, "numbers less than or equal to", num, "\n")
cat(count3, "numbers greater than", num, "\n")
cat(count4, "numbers greater than or equal to", num, "\n")
cat(count5, "numbers equal to", num, "\n")
cat(count6, "numbers not equal to", num, "\n")
cat(count7, "even numbers", num, "\n")
cat(count8, "odd numbers", num, "\n")
```

4.3 Slicing

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

TODO

Sum the elements of a vector meeting a specified criteria

TODO

Chapter 5

Summary Statistics of Vectors in C++ and R

5.1 Mean

Compute the mean of the vector

code/mean.cpp

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

double mean(const std::vector<double>& v1){
        /* compute the mean of a vector */
        return std::accumulate(v1.begin(), v1.end(), 0.0) / (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 = mean(vec1);

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

        return 0;
}
```

5.2 Median

Compute the median of the vector

code/median.cpp

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

double vec_median(std::vector<double> v1){
	/* compute the median of a vector */
	int sz = v1.size();
	sort(v1.begin(), v1.end());
	if(sz % 2 != 0){
		int med_pos = (sz + 1) / 2;
		double med = v1[med_pos - 1];
		return med;
	} else {
		int med_pos_l = (sz / 2) - 1;
		int med_pos_u = (sz / 2);
		double med = (v1[med_pos_l] + v1[med_pos_u]) / (double)2;
		return med;
	}
}

int main(){
	std::vector<double> vec1(9);
	std::vector<double> vec2(9);

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

	double num1 = vec_median(vec1);

	std::cout << "the median is: " << num1 << std::endl;

	return 0;
}
```

5.3 Minimum and Maximum

The function min returns the minimum value of the vector. The function which_min returns the index of the minimum value.

code/min.cpp

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

double min(const std::vector<double>& v1){
	/* minimum element of a vector */
	return *std::min_element(v1.begin(), v1.end());
}

int which_min(const std::vector<double>& v1){
	/* position of the minimum element */
```

```
        std::vector<double>::const_iterator iter;
        iter = std::min_element(v1.begin(), v1.end());
        return distance(v1.begin(), iter);
}

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;
        }

        // use random_shuffle to change the order of the elements
        std::random_shuffle(vec1.begin(), vec1.end());

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

        std::cout << "Minimum: " << min(vec1) << "; ";
        std::cout << "at position: " << which_min(vec1) << std::endl;

        return 0;
}
```

The function max returns the minimum value of the vector. The function which_max returns the index of the maximum value.

code/max.cpp

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

double max(const std::vector<double>& v1){
        /* maximum element of a vector */
        return *std::max_element(v1.begin(), v1.end());
}

int which_max(const std::vector<double>& v1){
        /* position of the maximum element */
        std::vector<double>::const_iterator iter;
        iter = std::max_element(v1.begin(), v1.end());
        return distance(v1.begin(), iter);
}

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;
        }

        // use random_shuffle to change the order of the elements
        std::random_shuffle(vec1.begin(), vec1.end());

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

        std::cout << "Maximum: " << max(vec1) << "; ";
        std::cout << "at position: " << which_max(vec1) << std::endl;

        return 0;
}
```

5.4 Percent Rank

TODO Compute the percent rank of a vector.

code/sumstats.R

```
# initialize a vector
# fill vec1 with 1 to 9
vec1 <- 1:9

# shuffle the elements of vec1
vec1 <- sample(vec1)
cat(vec1, "\n")

# compute the mean of vec1
mean1 <- mean(vec1)
cat("The mean is:", mean1, "\n")

#compute the median of vec1
median1 <- median(vec1)
cat("The median is:", median1, "\n")

# minimum value of vec1
min1 <- min(vec1)

# location of the minimum value
min1.location <- which.min(vec1)

cat("Minimum:", min1, ";\n")
cat("at index:", min1.location, "\n")

# maximum value of vec1
max1 <- max(vec1)

# location of the maximum value
max1.location <- which.max(vec1)

cat("Maximum:", max1, ";\n")
cat("at index:", max1.location, "\n")
```

```

# I can also use the summary function
summary(vec1)
```

Chapter 6

"Run" Functions in C++ and R

One of the great things about R is all the built-in functions (sum, sd, var, etc.) and the extensive list of packages if you are looking for a function or feature that is not included in base R. The TTR and zoo packages include several functions that run over a certain window such as runSD, runMin, runMax, runMean. For example, runMean calculates a simple moving average over n periods.

6.1 Running Min and Max

The run_min and run_max are useful for computing donchian channels for technical analysis.

code/run_min.cpp

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

std::vector<double> run_min(const std::vector<double>& v1, int n){
        /* calculate the running min of a vector */
        std::vector<double> v(v1.size());
        int sz = v.size();
        for(int i = 0; i < sz; i++){
                v[i+n-1] = *std::min_element(v1.begin() + i, v1.end() - sz + n + i);
        }
        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(10);

        vec2 = run_min(vec1, 3);

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

        return 0;
}
```

code/run_max.cpp

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

std::vector<double> run_max(const std::vector<double> v1, int n){
        /* compute the running maximum of a vector */
        std::vector<double> v(v1.size());
        int sz = v.size();
        for(int i = 0; i < sz; i++){
                v[i+n-1] = *std::max_element(v1.begin() + i, v1.end() - sz + n + i);
        }
        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 = run_max(vec1, 3);

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

        return 0;
}
```

6.2 Running Mean

The run_mean function is useful for computing the simple moving average for technical analysis.

code/run_mean.cpp

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

```

```
std::vector<double> run_mean(const std::vector<double>& v, int x){
        /* computes the moving average */
        std::vector<double> vec(v.size());
        int sz = v.size();
        for(int i = 0; i < sz; i++){
                vec[i+x-1] = std::accumulate(v.begin() + i, v.end() - sz + x + i, 0.0);
        }
        std::transform(vec.begin(), vec.end(), vec.begin(),
                std::bind2nd(std::divides<double>(), (double)x));
        return vec;
}

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

        // fill vec1 with 1 to 10
        for(int i = 0; i < vec1.size(); i++){
                vec1[i] = i + 1.3;
        }

        vec2 = run_mean(vec1, 3);

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

        return 0;
}
```