OPTIMIZING R CODE WITH RCPP

Random number generation

Romain François
Consulting Datactive, ThinkR

Generating single random numbers

```
// one number from a N(0,1)
double x = R::rnorm( 0, 1 ) ;

// one number from a U(-2,2)
double y = R::runif( -2, 2 ) ;

// ...
```

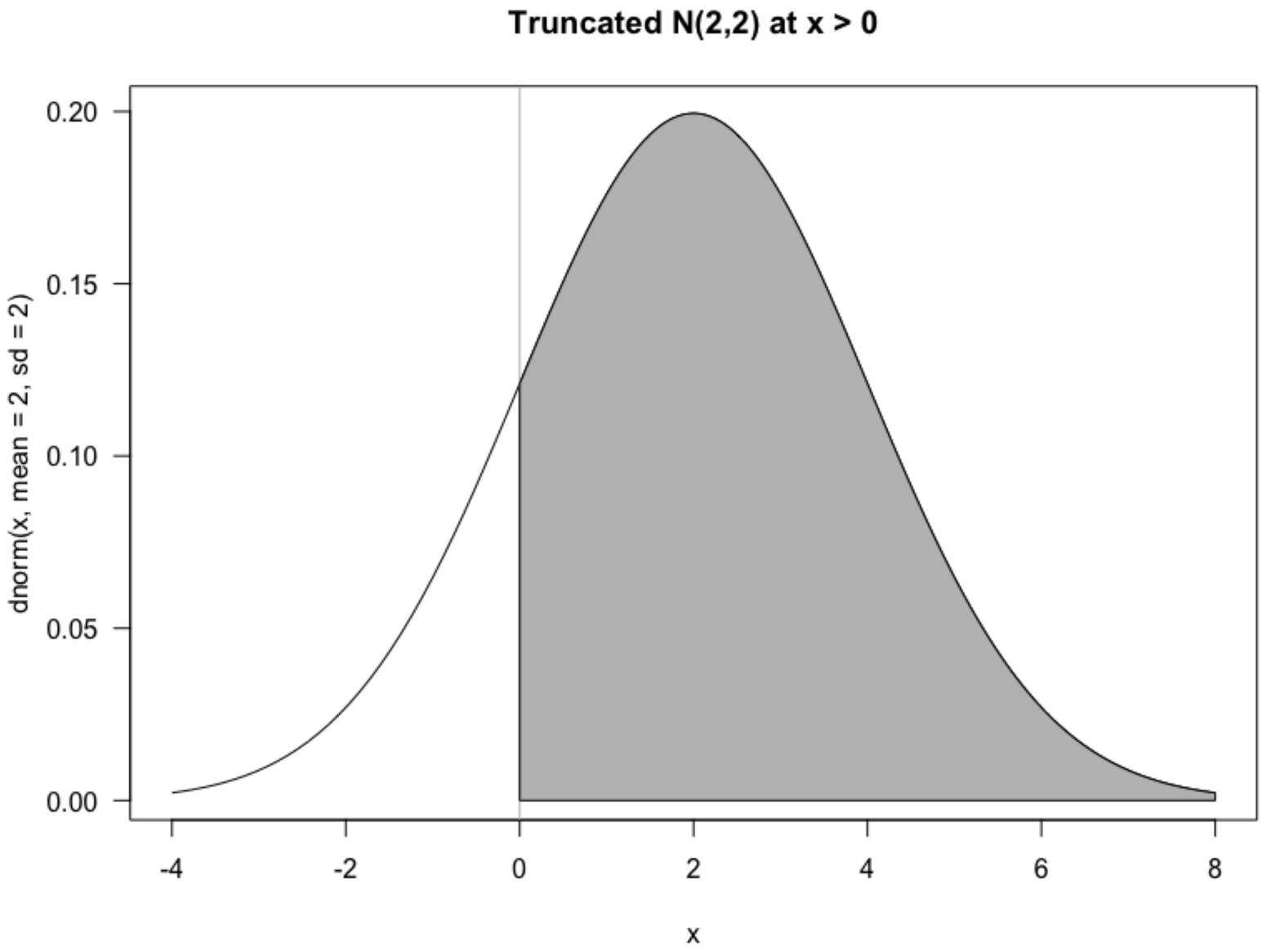
Generating vectors

Random number generators in the `Rcpp::` namespace.

```
NumericVector x = rnorm(10, 0, 2) ;
// same as this below
// because of using namespace Rcpp ;
//
// NumericVector x = Rcpp::rnorm(10, 0, 2) ;
```

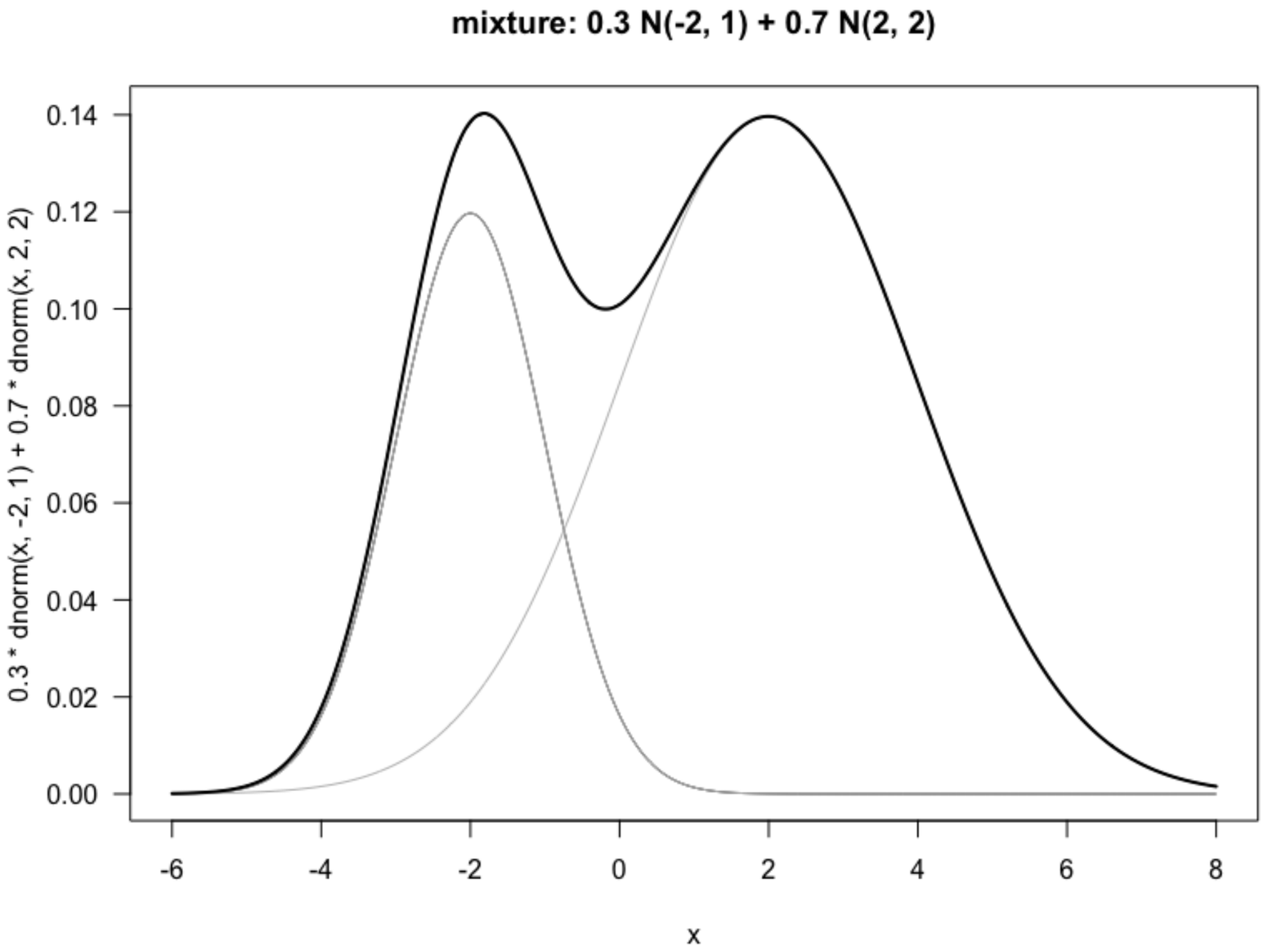
Alternative using scalar versions from `R::`

```
// same as
NumericVector x(10) ;
for(int i=0; i<10; i++){
    x[i] = R::rnorm(0, 2) ;
}
```


Truncated N(2,2) at x > 0
dnorm(x, mean = 2, sd = 2)
0.20
0.15
0.10
0.05
0.00
-4
-2
0
2
4
6
8
x

Rejection sampling

```
// we generate n numbers
NumericVector x(n) ;

// fill the vector in a loop
for( int i=0; i<n; i++){
    // keep generating d until it gets positive
    double d  ;
    do {
        d = ... ;
    } while( d < 0 ) ;

    x[i] = d ;

}
```


mixture: 0.3 N(-2, 1) + 0.7 N(2, 2)
0.3 * dnorm(x, -2, 1) + 0.7 * dnorm(x, 2, 2)
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
-6
-4
-2
0
2
4
6
8
x

Generate from a mixture of distributions

- Choose the component of the mixture using the weights

```
int component( NumericVector weights, double total_weight ){
    // return the index of the selected component
}
```

- Generate the number using the parameters of the selected components

```
NumericVector rmix( int n, NumericVector weights, NumericVector means,
                    NumericVector sds ){

    NumericVector res(n) ;

    for( int i=0; i<n; i++){
        // find which component to use
        ...

        // simulate using the mean and sd from the selected component
        ...
    }

    return res ;
}
```

OPTIMIZING R CODE WITH RCPP

Let's practice!

OPTIMIZING R CODE WITH RCPP

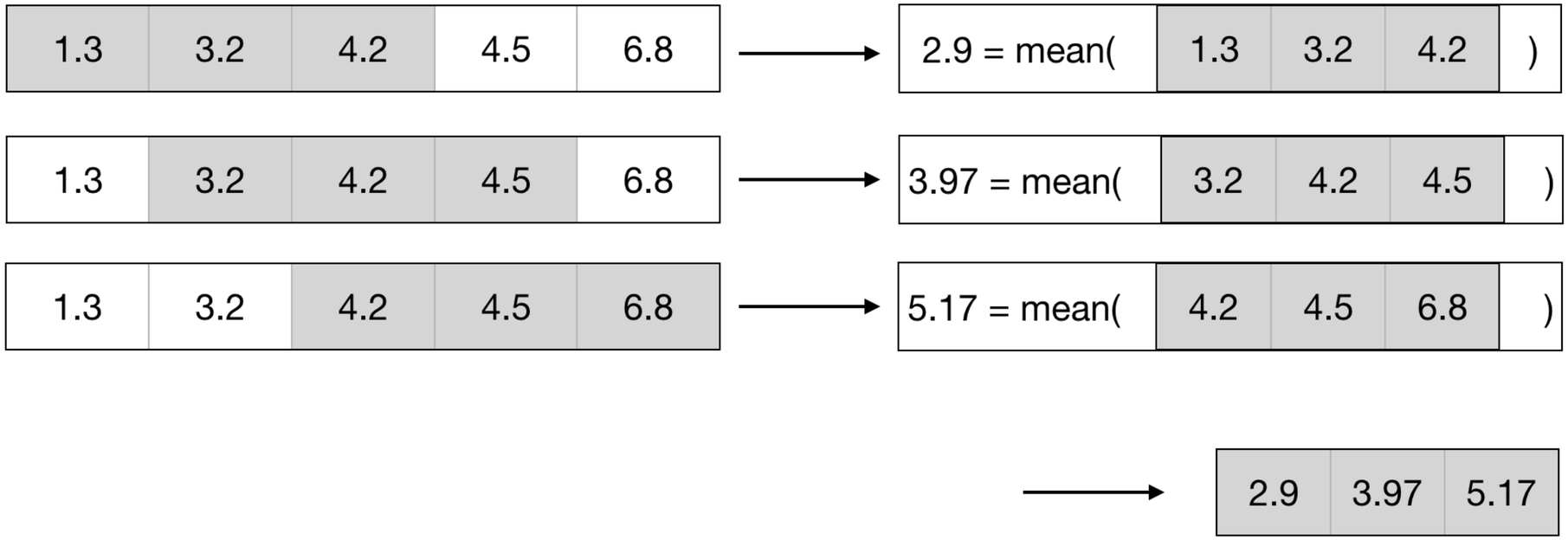
Rolling operations

Romain François
Consulting Datactive, ThinkR

Rolling means

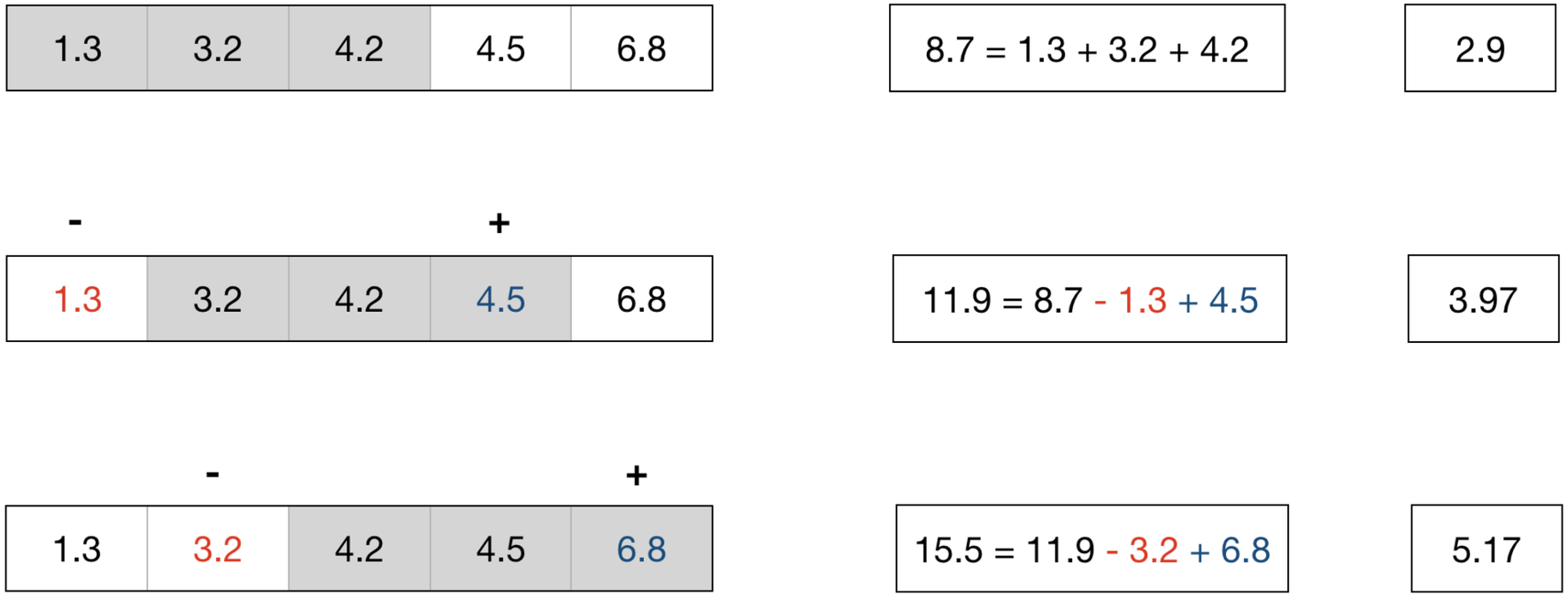
```
rollmean1 <- function(x, window = 3){
  n <- length(x)

  # create empty vector full of NA
  res <- rep(NA, n)

  # fill the values
  for( i in seq(window, n) ){
    idx <- seq(i-window+1,i)
    res[i] <- mean(x[idx])
  }
  res
}
```

Rolling means



- Make an integer vector to hold indice, extract the relevant part of `x`
- Call the `mean` function on that extract

Alternative algorithm



Alternative algorithm

```
rollmean2 <- function(x, window = 3){
  n <- length(x)
  res <- rep(NA, n)

  # first value
  total <- sum(head(x,window))
  res[window] <- total / window

  # remaining values
  for( i in seq(window+1, n) ){
    total <- total + x[i] - x[i-window]
    res[i] <- total / window
  }
  res
}
```

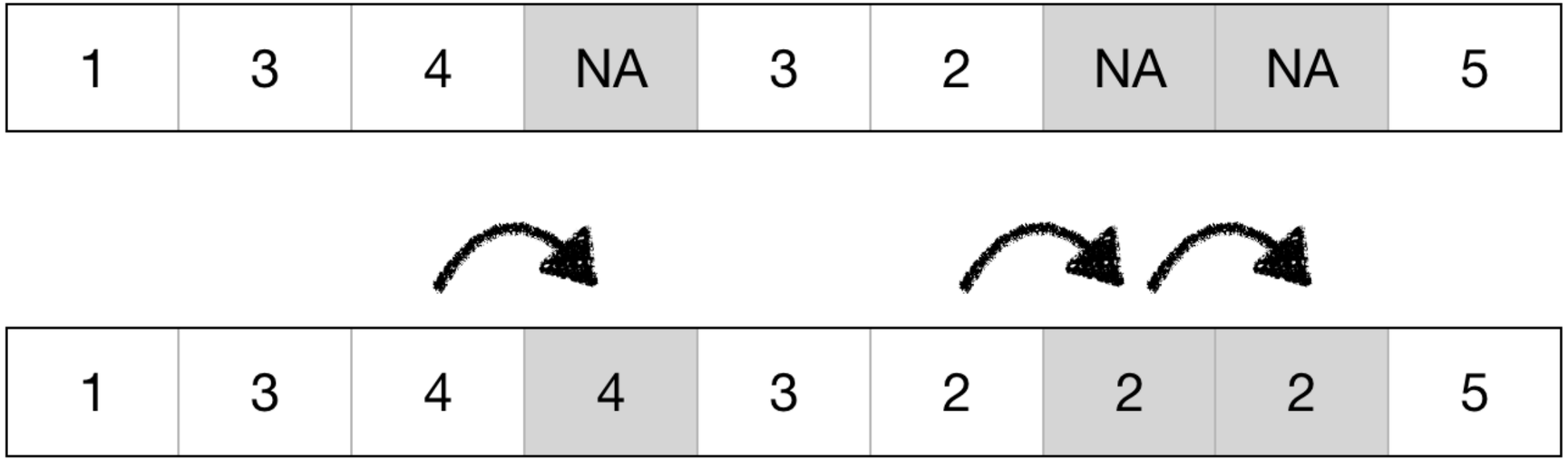
Hackstucious (hack + astucious) vectorization

```
x <- c(1.3, 3.2, 4.2, 4.5, 6.8)

start <- sum(x[1:3])

head( x, -3 )
1.3 3.2

tail( x, -3 )
4.5 6.8

c( start, start + cumsum( tail(x, -3) - head( x, -3 ) ) )
8.7 11.9 15.5

c( start, start + cumsum( tail(x, -3) - head( x, -3 ) ) ) / 3
2.900000 3.966667 5.166667
```

Comparison

```
library(microbenchmark)
x <- rnorm(1e5)

microbenchmark(
    rollmean1(x, 3),
    rollmean2(x, 3),
    rollmean3(x, 3)
)
Unit: milliseconds
            expr        min         lq       mean     median ...
 rollmean1(x, 3) 833.667884 857.507753 971.250098 893.206776 ...
 rollmean2(x, 3)  10.539993  11.034244  12.293105  11.396629 ...
 rollmean3(x, 3)   1.429817   1.625453   3.070925   3.067068 ...
```

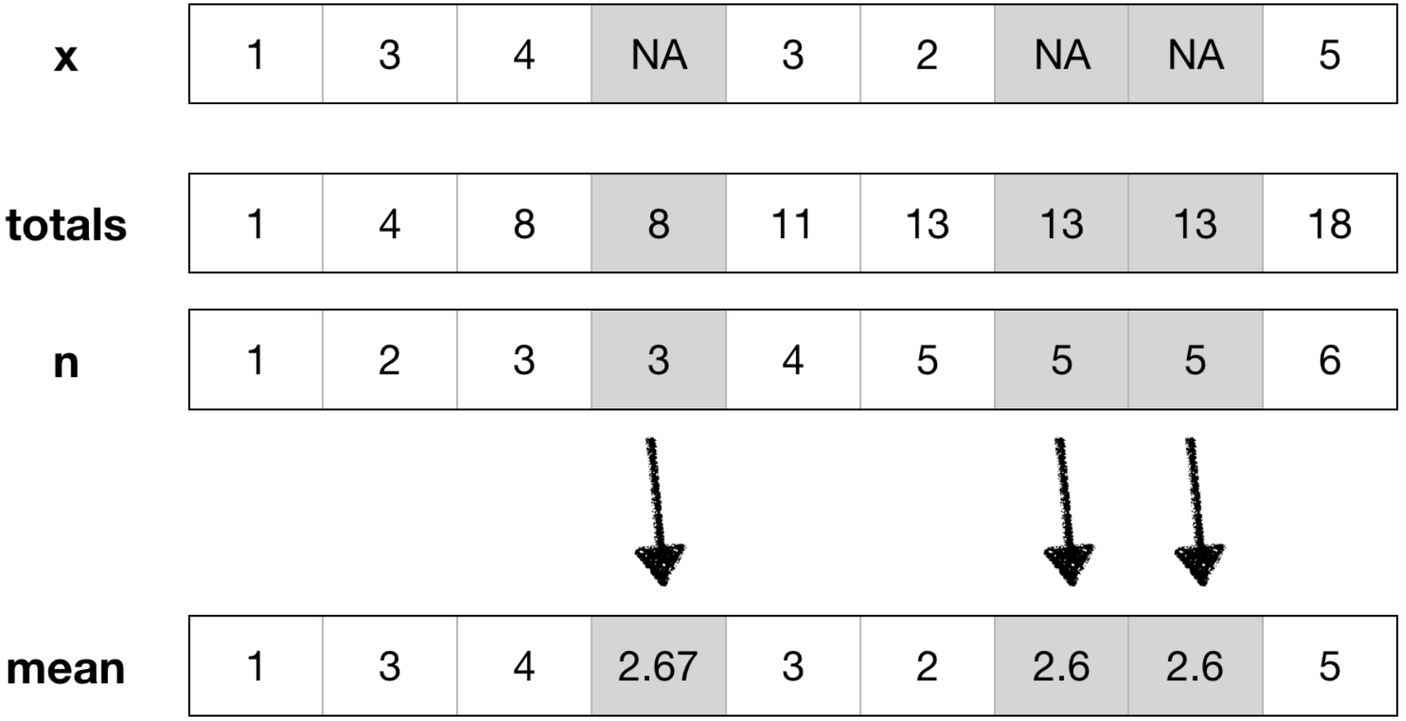
Last observation carried forward



Last observation carried forward

```
na_locf1 <- function(x){
  current <- NA

  res <- x
  for( i in seq_along(x)){
    if( is.na(x[i]) ){
      # replace with current
      res[i] <- current
    } else {
      # set current
      current <- x[i]
    }
  }

  res
}
```

Mean carried forward



Mean carried forward

```
na_meancf1 <- function(x){
  # ( cumulative sum of non NA values ) / ( cumulative count of non NA )
  means <- cumsum( replace(x, is.na(x), 0) ) / cumsum(!is.na(x))

  # replace the missing values by the means
  x[is.na(x)] <- means[is.na(x)]
  x
}

# iterative version
na_meancf2 <- function(x){
  total <- 0
  n     <- 0
  for( i in seq_along(x) ){
    if( is.na(x[i]) ){
        x[i] <- total / n
    } else {
        total <- x[i] + total
        n <- n + 1
    }
  }
}
```

Comparisons

```
x <- rnorm(1e5)
x[ sample(1e5, 100) ] <- NA

microbenchmark( na_meancf1(x), na_meancf2(x) )
Unit: milliseconds
          expr       min        lq      mean    median ...
 na_meancf1(x)  1.176276  2.785667  3.237009  3.474028 ...
 na_meancf2(x) 16.945271 17.430133 19.678276 18.625274 ...
```

OPTIMIZING R CODE WITH RCPP

Let's practice!

OPTIMIZING R CODE WITH RCPP

Auto regressive model

Romain François
Consulting Datactive, ThinkR

Auto regressive model, AR

$$X_i = \sum_{j=1}^{np} \phi_p X_{i-j} + \epsilon_i$$

```
ar <- function(n, phi, sd){
    x  <- epsilon <- rnorm(n, sd = sd)
    np <- length(phi)

    for( i in seq(np+1, n)){
        x[i] <- sum(x[seq(i-1, i-np)] * phi) + epsilon[i]
    }
    x
}
```

AR in C++

- First ☐, to fill the `np` first values

```
NumericVector x(n) ;

// initial loop
for( ___ ; __ < np ; ___ ){
    x[i] = R::rnorm(___) ;
}
```

- Main part with outer and inner ☐

```
// outer loop
for( ___ ; ___ ; ___ ){

    double value = rnorm(___) ;

    // inner loop
    for( ___ ; ___ ; ___ ){
        value += ___ ;
    }
    x[i] = value ;
}
```

Moving average simulation

$$X_i = \epsilon_i + \sum_{j=1}^{nq} \theta_j \epsilon_{i-j}$$

```
ma <- function(n, theta, sd){
    epsilon <- rnorm(n, sd = sd)
    x <- numeric(n)
    nq <- length(theta)

    for( i in seq(nq+1, n)){
        x[i] <- sum(epsilon[seq(i-1, i-nq)] * theta) + epsilon[i]
    }
    x
}
```

Moving average simulation

```
#include <Rcpp.h>
using namespace Rcpp ;

// [[Rcpp::export]]
NumericVector ma( int n, double mu, NumericVector theta, double sd ){
    int nq = theta.size() ;

    // generate the noise vector at once
    // using the Rcpp::rnorm function, similar to the R function
    NumericVector eps = Rcpp::rnorm(n, 0.0, sd) ;

    // init the output vector of size n with all 0.0
    NumericVector x(___) ;

    // start filling the values at index nq + 1
    for( int i=nq+1; i<n; i++){

        ____
    }
    return x ;
}
```

ARMA(p,q) = AR(p) + MA(q)

$$X_i = \epsilon_i + \sum_{j=1}^{np} \phi_j X_{i-j} + \sum_{j=1}^{nq} \theta_j \epsilon_{i-j}$$

OPTIMIZING R CODE WITH RCPP

Let's practice!

OPTIMIZING R CODE WITH RCPP

Congratulations!

Romain François
Consulting Datactive, ThinkR

evalCpp and cppFunction

- Evaluating simple C++ statements

```
evalCpp( "40+2" )
42
```

- Creating a C++ function from the R console

```
cppFunction( "double add( double x, double y){
    return x + y ;
})

add( 40, 2 )
42
```

For loops

```
for( init ; condition ; increment ){
    body
}
```

- init: what happens at the beginning
- condition: should the loop continue
- increment: after each iteration
- body: what the loop does

For loops

```
for( int i=0; i<n; i++){
    // do something with i
}
```

Vector indexing

```
NumericVector x = ... ;
int n = x.size() ;
```

```
// first value
x[0]
```

```
// second value
x[1]
```

```
// last value
x[n-1]
```

C++ files with Rcpp

```
#include <Rcpp.h>
```

```
using namespace Rcpp ;
```

```
// [[Rcpp::export]]
double add( double x, double y){
    return x + y ;
}
```

```
// [[Rcpp::export]]
double twice( double x){
    return 2.0 * x;
}
```

Typical Rcpp function

```
#include <Rcpp.h>
using namespace Rcpp ;

// [[Rcpp::export]]
double fun( NumericVector x ){

    // extract data from input and prepare outputs
    int n = x.size() ;
    double res = 0.0 ;

    // loop around input and/or output
    for(int i=0; i<n; i++){
        // do something with x[i]
    }

    // return output
    return res ;
}
```

OPTIMIZING R CODE WITH RCPP

Congratulations!