Basic C++ through Rcpp

Advanced Statistical Programming Camp
Jonathan Olmsted (Q-APS)

Day 3: May 29th, 2014
AM Session

Outline

Why C++?

- C++ is a compiled language $\rightarrow$ fast.
  - Python and R are interpreted languages.
- R is written in C (closely related to C++), so C++ integrates well.
- C++ is popular $\rightarrow$ a lot of robust code to work with complex algorithms and data structures.
- C++ can be used were parallel computing can't.

What is Rcpp?

- Rcpp is an R package intended to ease the integration of C++ with R–based work.
- Rcpp:
  1. Provides R level functions to compile C++ code.
  2. Automates the creation of the full C++ code so users can focus on "snippets".
  3. Provides C++ level classes (i.e., kinds of objects) that behave like their R counterparts.
  4. Automates the creation of R level functions calling a user's code.
- Has dramatically increased the use of compiled code in the R community.

Using Rcpp

- Rcpp hides the details of *compiling*, *linking*, and *calling* the code.
- But a full development environment is still required.
- Rcpp can be used on all operating systems, but can be tricky to set up the first time.
- After the workshop, if you would like to set up Rcpp to work locally on your Mac or Windows OS–based machine, I can help with that.

Using Rcpp Today

- So that everyone uses a homogenous environment, we will use Rcpp on **Adroit** where setup was easy.
- This means the C++ source code **must** be on **Adroit**.
- We will run R *interactively* on **Adroit** in a terminal window. So, R code can just be copied and pasted in that terminal.
- There are multiple ways to turn C++ snippets into R functions with Rcpp.
  - We will just use `sourceCpp`.
  - Our C++ code will live in its own file `cpp`.

```
library("Rcpp")
sourceCpp("functions.cpp")
```

Objects in R, Rcpp, and C++

The native **types** of objects in C++ that we will use are:

- `double`: real values like 1.4 or -5/89
- `bool`: logical values of `true` or `false`
- `int`: integer values like 19 or -1

Objects in R, Rcpp, and C++

Rcpp provides **classes** collecting these types:

- `NumericVector` and `NumericMatrix`
  - elements are `double`
- `LogicalVector` and `LogicalMatrix`
  - elements are `bool`
- `IntegerVector` and `IntegerMatrix`
  - elements are `int`
- `List`
  - elements are any other Rcpp class or C++ type.

Rcpp maps R objects to the right C++ level object and from C++ level objects to the right R level object mostly automatically.

Outline

Basics of the C++ Language

- C++ is very similar to R in many ways. But some differences are important.
- However, variables **statically typed**.
  - You must *declare* the kind of object a variable will be.
  - Once you've declared this, it can not change.
  - This applies to the return values of functions, too.
- Expressions must end with a semicolon — ";"
- Indexing starts with 0.

```
a <- 1 ; a <- "char"
```

R is not statically typed.

C++ Type: double

```
// [[Rcpp::export()]]
double fD () {
    double x = 1 ;
    return(x) ;
}
```

```
fD()

## [1] 1
```

C++ Type: bool

```
// [[Rcpp::export()]]
bool fB () {
    bool x = true ;
    return(x) ;
}
```

```
fB()

## [1] TRUE
```

C++ Type: int

```
// [[Rcpp::export()]]
int fI () {
    int x = 1 ;
    return(x) ;
}
```

```
fI()

## [1] 1
```

C++ Comments

```
// [[Rcpp::export()]]
double fComment () {
    double x = 1.0 ;
    // a single line comment
    double y = x + 3.2 ;
    /* a multiple line
       comment
    */
    return(y) ;
}
```

```
fComment()

## [1] 4.2
```

Outline

Rcpp Class: NumericVector

```
// [[Rcpp::export()]]
Rcpp::NumericVector gNV1 () {
    // create length 4 vector
    // 0.0-valued entries
    Rcpp::NumericVector x(4) ;
    return(x) ;
}
```

```
gNV1()

## [1] 0 0 0 0
```

Rcpp Class: NumericVector

```
// [[Rcpp::export()]]
Rcpp::NumericVector gNV2 () {
    // create length 4 vector
    // 13.1-valued entries
    Rcpp::NumericVector x(4, 13.1) ;
    return(x) ;
}
```

```
gNV2()

## [1] 13.1 13.1 13.1 13.1
```

Rcpp Class: LogicalVector

```
// [[Rcpp::export()]]
Rcpp::LogicalVector gLV1 () {
    // create length 4 vector
    // false-valued entries
    Rcpp::LogicalVector x(4) ;
    return(x) ;
}
```

```
gLV1()

## [1] FALSE FALSE FALSE FALSE
```

Rcpp Class: LogicalVector

```
// [[Rcpp::export()]]
Rcpp::LogicalVector gLV2 () {
    // create length 4 vector
    // true-valued entries
    Rcpp::LogicalVector x(4, true) ;
    return(x) ;
}
```

```
gLV2()

## [1] TRUE TRUE TRUE TRUE
```

Rcpp Class: IntegerVector

```
// [[Rcpp::export()]]
Rcpp::IntegerVector gIV1 () {
    // create length 4 vector
    // 0-valued entries
    Rcpp::IntegerVector x(4) ;
    return(x) ;
}
```

```
gIV1()

## [1] 0 0 0 0
```

Rcpp Class: IntegerVector

```
// [[Rcpp::export()]]
Rcpp::IntegerVector gIV2 () {
    // create length 4 vector
    // -3-valued entries
    Rcpp::IntegerVector x(4, -3) ;
    return(x) ;
}
```

```
gIV2()

## [1] -3 -3 -3 -3
```

Rcpp Class: NumericMatrix

```
// [[Rcpp::export()]]
Rcpp::NumericMatrix gNM1() {
    // create 4 by 6 matrix
    // 0-valued entries
    Rcpp::NumericMatrix x(4, 6) ;
    return(x) ;
}
```

```
gNM1()
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    0    0    0    0    0    0
## [2,]    0    0    0    0    0    0
## [3,]    0    0    0    0    0    0
## [4,]    0    0    0    0    0    0
```

Rcpp Class: LogicalMatrix

```
// [[Rcpp::export()]]
Rcpp::LogicalMatrix gLM1() {
    // create 4 by 6 matrix
    // false-valued entries
    Rcpp::LogicalMatrix x(4, 6) ;
    return(x) ;
}
```

```
gLM1()
```

```
##      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]
## [1,] FALSE FALSE FALSE FALSE FALSE FALSE
## [2,] FALSE FALSE FALSE FALSE FALSE FALSE
## [3,] FALSE FALSE FALSE FALSE FALSE FALSE
## [4,] FALSE FALSE FALSE FALSE FALSE FALSE
```

Rcpp Class: IntegerMatrix

```
// [[Rcpp::export()]]
Rcpp::IntegerMatrix gIM1() {
    // create 4 by 6 matrix
    // 0-valued entries
    Rcpp::IntegerMatrix x(4, 6) ;
    return(x) ;
}
```

```
gIM1()
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    0    0    0    0    0    0
## [2,]    0    0    0    0    0    0
## [3,]    0    0    0    0    0    0
## [4,]    0    0    0    0    0    0
```

Rcpp Class: List

```
// [[Rcpp::export()]]
Rcpp::List gL1() {
    Rcpp::List x ;
    x["a"] = 1.0 ;
    x["b"] = -3 ;
    x["c"] = Rcpp::NumericVector(5, 2.3) ;
    x["d"] = Rcpp::IntegerMatrix(3, 3) ;
    return(x) ;
}
```

Rcpp Class: List

```
gL1()

## $a
## [1] 1
##
## $b
## [1] -3
##
## $c
## [1] 2.3 2.3 2.3 2.3 2.3
##
## $d
##      [,1] [,2] [,3]
## [1,]    0    0    0
## [2,]    0    0    0
## [3,]    0    0    0
```

Outline

Input Data and Dimensions

```
// [[Rcpp::export()]]
Rcpp::List h1 (Rcpp::NumericVector x,
               Rcpp::NumericMatrix y
               ) {
    int n1 = x.size() ;
    int n2 = x.length() ;
    int r = y.nrow() ;
    int c = y.ncol() ;
    Rcpp::List ret ;
    ret["a"] = n1 ;
    ret["b"] = n2 ;
    ret["c"] = r ;
    ret["d"] = c ;
    return(ret) ;
}
```

Input Data and Dimensions

```
h1(rnorm(4),
   diag(3) * runif(1)
   )

## $a
## [1] 4
##
## $b
## [1] 4
##
## $c
## [1] 3
##
## $d
## [1] 3
```

Input Data and Dimensions

```
h1(y = diag(3) * runif(1),
   x = rnorm(4)
   )

## $a
## [1] 4
##
## $b
## [1] 4
##
## $c
## [1] 3
##
## $d
## [1] 3
```

Indexing

```
// [[Rcpp::export()]]
Rcpp::List h2 (Rcpp::NumericVector x,
               Rcpp::NumericMatrix y
               ) {
    int r = y.nrow() ;
    int c = y.ncol() ;
    double e1 = x(0) ;
    double e2 = y(0, 0) ;
    Rcpp::List ret ;
    ret["a"] = e1 ;
    ret["b"] = e2 ;
    ret["c"] = y(r - 1, c - 1) ;
    return(ret) ;
}
```

The first element in a vector or matrix has an index of 0, not 1.

Indexing

```
h2(rnorm(4),
   diag(3) * runif(1)
   )

## $a
## [1] 1.224
##
## $b
## [1] 0.2892
##
## $c
## [1] 0.2892
```

Outline

Conditionals, Arithmatic, and Logical Operators

```
// [[Rcpp::export()]]
Rcpp::List i1 (Rcpp::NumericVector x
               ) {
    int n = x.length() ;
    Rcpp::List ret ;
    if (n > 3) {
        ret["a"] = n ;
    } else {
      ret["a"] = false ;
    }
    ret["b"] = n + 3.2 ; // type promotion
    return(ret) ;
}
```

Arithmetic: `+`, `-`, `/`, `*`, `pow()`, `sqrt()`
Logical: `==`, `!=`, `<=`, `<`, `>=`, `>`, `|` (or), `&` (and)

Conditionals, Arithmatic, and Logical Operators

```
i1(1:4)

## $a
## [1] 4
##
## $b
## [1] 7.2

i1(3)

## $a
## [1] FALSE
##
## $b
## [1] 4.2
```

For Loops, Arithmatic, and Mathematical Functions

```
// [[Rcpp::export()]]
Rcpp::List i2 (Rcpp::NumericVector x,
               double t
               ) {
    int n = x.length() ;
    Rcpp::NumericVector y(n) ;
    Rcpp::List ret ;
    for (int it = 0 ; it < n ; it++) {
        if (x(it) <= t) {
            y(it) = sqrt(pow(x(it) - 1.3, 4)) ;
        } else {
            y(it) = x(it) * 2 ;
        }
    }
    ret["x"] = x ; ret["y"] = y ;
    return(ret) ;
}
```

For Loops, Arithmatic, and Mathematical Functions

```
i2(rnorm(10), 0.2)
```

```
## $x
##  [1] -1.04889  1.29476  0.82554 -0.05569 -0.78438 -0.73350
##  [7] -0.21587 -0.33491 -1.08570 -0.08542
##
## $y
##  [1] 5.517 2.590 1.651 1.838 4.345 4.135 2.298 2.673 5.692
## [10] 1.919
```

For Loops, Rcout

```
// [[Rcpp::export()]]
Rcpp::NumericVector j1 (Rcpp::NumericVector x) {
    int n = x.length() ;
    Rcpp::NumericVector y(2) ;
    double total1 = 0.0 ;
    double total2 = 0.0 ;
    for (int it = 0 ; it < n ; it++) {
        Rcpp::Rcout << x(it) << std::endl ;
        total1 += x(it) ;
        total2 = total2 + x(it) ;
    }
    y(0) = total1 ;
    y(1) = total2 ;
    return(y) ;
}
```

`Rcout` prints **scalars** to the screen.

Rcout

```
j1(1:3)

## 1
## 2
## 3
## [1] 6 6
```

While Loops, Rcout

```
// [[Rcpp::export()]]
int j2 (double t) {
    int it = 0 ;
    double total = 0.0 ;
    while (total < t) {
        it++ ;
        total += it ;
        Rcpp::Rcout << it << " " << total << std::endl ;
    }
    return(it) ;
}
```

While Loops, Rcout

```
j2(34.2)

## 1 1
## 2 3
## 3 6
## 4 10
## 5 15
## 6 21
## 7 28
## 8 36
## [1] 8
```

While Loops, Rcout

```
its <- j2(34.2)

## 1 1
## 2 3
## 3 6
## 4 10
## 5 15
## 6 21
## 7 28
## 8 36

its

## [1] 8
```

There is not effect of `Rcout` on the return value of the function.

Outline

Special Values

```
// [[Rcpp::export()]]
Rcpp::List k1 (Rcpp::NumericVector x) {
    Rcpp::List ret ;
    ret["a"] = NA_REAL == x(0) ;
    ret["b"] = R_NegInf == x(0) ;
    ret["c"] = R_PosInf == x(0) ;
    ret["d"] = R_IsNA(x(0)) ;
    ret["e"] = R_IsNaN(x(0)) ;
    return(ret) ;
}
```

We can work with R values of `Inf`, `-Inf`, `NaN`, and `NA` at the C++ level.

Special Values

```
k1(4)

## $a
## [1] FALSE
##
## $b
## [1] FALSE
##
## $c
## [1] FALSE
##
## $d
## [1] 0
##
## $e
## [1] 0
```

Special Values

```
k1(NA)

## $a
## [1] FALSE
##
## $b
## [1] FALSE
##
## $c
## [1] FALSE
##
## $d
## [1] 1
##
## $e
## [1] 0
```

Notice how you must test for an `NA` value!

```
k1(Inf)

## $a
## [1] FALSE
##
## $b
## [1] FALSE
##
## $c
## [1] TRUE
##
## $d
## [1] 0
##
## $e
## [1] 0
```

```
k1(NaN)

## $a
## [1] FALSE
##
## $b
## [1] FALSE
##
## $c
## [1] FALSE
##
## $d
## [1] 0
##
## $e
## [1] 1
```

Outline

We can use this to work through the code for pairwise distances given a matrix of coordinates.

```
sourceCpp("distance.cpp")
```

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

- `# include` statements express dependencies beyond the normal C++ language.
- All of our Rcpp work will require `# include <Rcpp.h>`.
- We need `# include <math.h>` to provide trigonometric functions.
- Only functions with `// [[Rcpp::export()]]` above them are "exported" into R functions.

One Euclidean Pairwise Distance Calculation

This function **is not** exported to R, but calculates the Euclidean distance between two points given the four coordinates.

```
double dist1(double lat1,
             double long1,
             double lat2,
             double long2
             ) {
    double dist = sqrt(pow(lat1 - lat2, 2) +
                       pow(long1 - long2, 2)
                       ) ;
    return (dist) ;
}
```

Other functions defined later in the `cpp` file can call it. But, they must occur **after** `dist1` is defined.

Full Euclidean Pairwise Distance Matrix

```
// [[Rcpp::export()]]
Rcpp::NumericMatrix calcPWD (Rcpp::NumericMatrix x
                             ) {
    int outrows =  x.nrow() ;
    int outcols = x.nrow() ;
    Rcpp::NumericMatrix out(outrows, outcols) ;

    for (int arow = 0 ; arow < outrows ; arow++) {
        for (int acol = 0 ; acol < outcols ; acol ++) {
            out(arow, acol) = dist1(x(arow, 0),
                                    x(arow, 1),
                                    x(acol, 0),
                                    x(acol, 1)
                                    ) ;
        }
    }
    return (out) ;
}
```

This is the full definition of the pair-wise Euclidean distance function. Notice that it uses `dist1`.

Full Euclidean Pairwise Distance Matrix

Because `calcPWD` is exported, we can call it in R.

```
dfCounties <- read.csv("counties.csv")
mCoords <- as.matrix(dfCounties[, 1:2])
system.time({
    mDist <- calcPWD(mCoords)
    }
             )

##    user  system elapsed
##   0.151   0.025   0.176

dim(mDist)

## [1] 3109 3109
```

Full Accurate Pairwise Distance Matrix

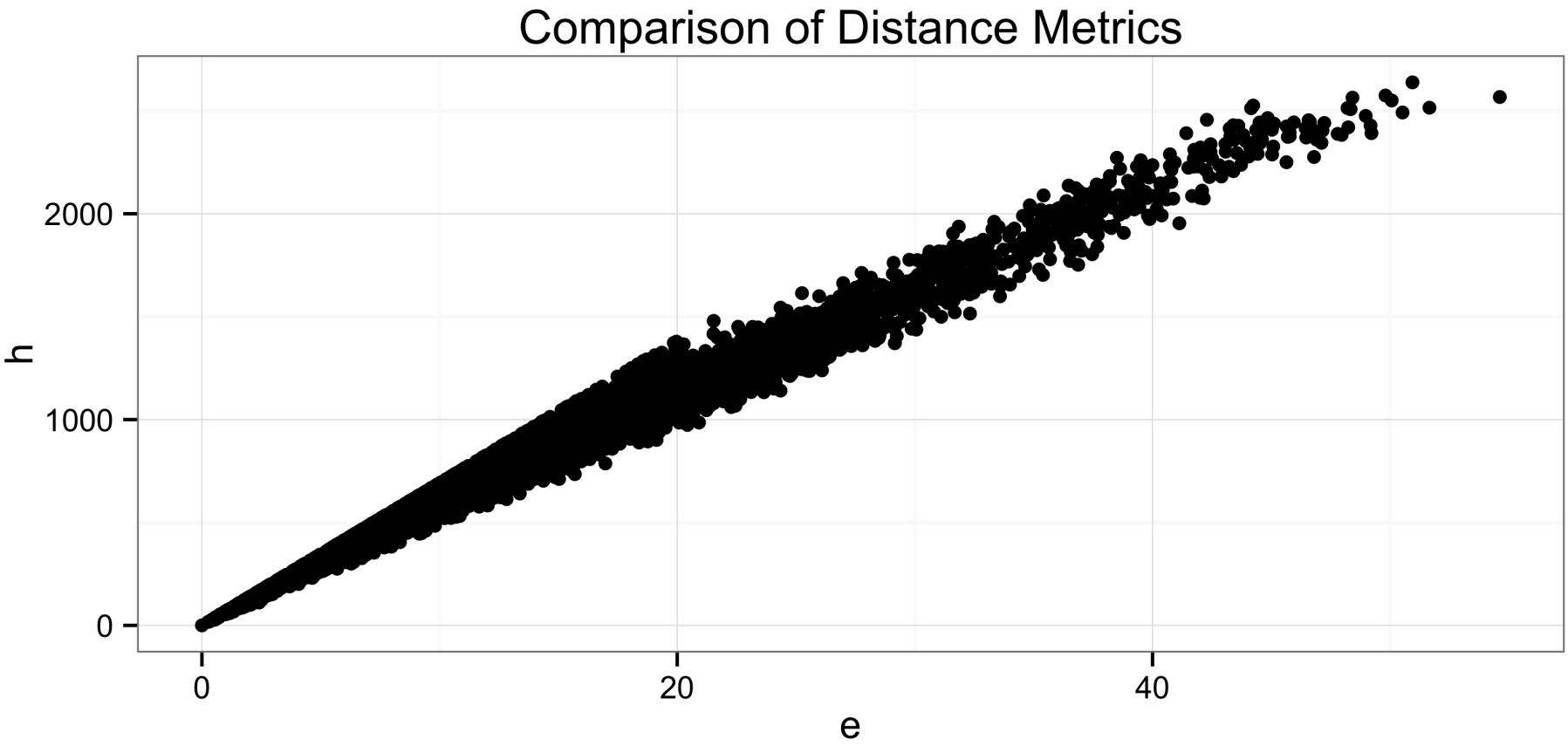
```
// [[Rcpp::export()]]
Rcpp::NumericMatrix calcPWDh (Rcpp::NumericMatrix x
                              ) {
  int nrows =  x.nrow() ;
  int ncols = x.nrow() ;
  Rcpp::NumericMatrix out(nrows, ncols) ;
  double rad = 3963.1676 ;
  double pi = 3.14159265358979238463 ;
  for(int arow = 0; arow < nrows; arow++) {
    for(int acol = 0; acol < ncols; acol++) {
      double phi1 = x(arow, 0) * pi / 180 ;
      double phi2 = x(acol, 0) * pi / 180 ;
      double lambda1 = x(arow, 1) * pi / 180 ;
      double lambda2 = x(acol, 1) * pi / 180;
      double q1 = 2 * rad ;
      double q2 = pow(sin((phi1 - phi2) / 2), 2) ;
      double q3 = pow(sin((lambda1 - lambda2) / 2), 2) ;
      double q4 = cos(phi1) * cos(phi2) ;
      out(arow, acol) = q1 * asin(sqrt(q2 + q4 * q3)) ;
    }
  }
  return(out) ;
}
```

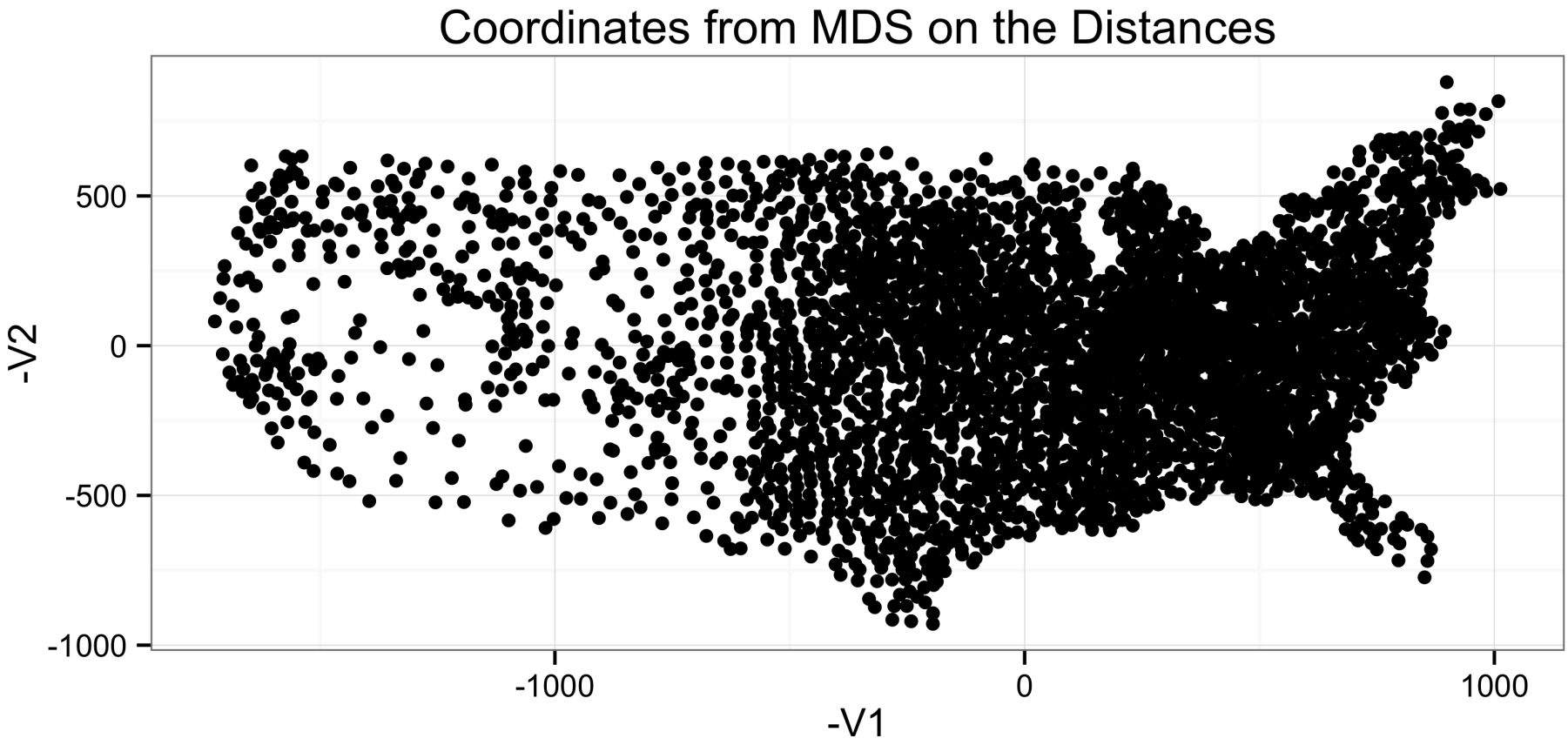
Full Accurate Pairwise Distance Matrix

The accurate calculation actually takes noticeably longer because of all the additional calculations involved in each iteration.

```
system.time({
    mDisth <- calcPWDh(mCoords)
    })

##    user  system elapsed
##   0.875   0.024   0.900

dim(mDisth)

## [1] 3109 3109
```

Can you create a new function based on this code which does not make duplicate calculations?


Comparison of Distance Metrics
2000
1000
0
h
0
20
40
e


Coordinates from MDS on the Distances
500
0
-500
-1000
-V2
-1000
0
1000
-V1