Matrices

Rachel

2022-12-12

First load the relevant matrices for this portifolio:

```
library(Matrix)
library(igraph)
```

```
##
## Attaching package: 'igraph'

## The following objects are masked from 'package:stats':
##
##     decompose, spectrum

## The following object is masked from 'package:base':
##
##     union
```

Dense Matrices

Solving Linear Systems

For a linear system $Ax = b$ we can use the `solve()` function in two ways, as demonstrated below

```
n <- 9

A <- as.matrix(Hilbert(n))
x <- matrix(rnorm(n), nrow=n, ncol=1)

b <- A %*% x

x_1 <- solve(A) %*% b
x_2 <- solve(A,b)
cbind(x_1, x_2)
```

```
##             [,1]       [,2]
##  [1,] -1.7067836 -1.7067779
##  [2,]  0.8398808  0.8398567
##  [3,] -0.9429549 -0.9428954
##  [4,] -1.0635513 -1.0636426
```

```
##  [5,] -0.3296127 -0.3295566
##  [6,]  0.5513086  0.5513024
##  [7,] -1.4577713 -1.4577911
##  [8,] -0.7494545 -0.7494445
##  [9,] -1.8508279 -1.8508286
```

We can see these differ slightly, but we can compare the error to determine which is better

```
c(norm(x-x_1, type='1'), norm(x-x_2, type='1'))
```

```
## [1] 2.722671e-04 8.280467e-06
```

and thus we can see that using `b` as an argument for the solve function has more accuracy than inverting `A` and then multiplying it by `b`. The decreased numerical error generally occurs due to the successive loss of precision as a result of the increased addition, multiplications being done when inverting `A`, and then multiplying by `b`.

Sparse Matrices

Using the `Matrix` Package

In a sparse matrix, most of the entries are `0`, and so the `Matrix` package allows us to store these as an object of class `dcGMatrix` (stores matrices in compressed sparse column format). A `dcGMatrix` has the following properties

```
nrows <- 100
ncols <- 100

vals <- sample(x=c(0, 1, 2), prob=c(0.98, 0.01, 0.01), size=nrows*ncols, replace=TRUE)
m <- Matrix(vals, nrow=nrows, ncol=ncols, sparse=TRUE)

str(m)
```

```
## Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##   ..@ i       : int [1:187] 33 56 58 83 59 16 49 95 0 90 ...
##   ..@ p       : int [1:101] 0 4 5 8 9 10 15 17 23 25 ...
##   ..@ Dim     : int [1:2] 100 100
##   ..@ Dimnames:List of 2
##   .. ..$ : NULL
##   .. ..$ : NULL
##   ..@ x       : num [1:187] 1 1 2 2 2 2 2 2 1 1 ...
##   ..@ factors : list()
```

We do also have the options of `dgRMatrix` (compressed sparse row format) and `dgTMatrix`(stores data in triplets). One can turn `dgCMatrix` objects into `dgTMatrix` objects (and vice versa), but we cannot coerce a `dgRMatrix` to or from other `Matrix` objects.

Note that `dgRMatrix` is rarely used due to it's inefficiency in practice.

Graph Dependencies

The adjacency matrix for a graph is often sparse, and so we can use a `dgCMatrix` to represent these graphs. To illustrate this, we construct a symmetric sparse matrix which will be our adjacencry matrix

```
edge <- data.frame(i = 1:20, j = sample(1:20, 20, replace = TRUE))

adj <- sparseMatrix(i = as.integer(edge$i),
                    j = as.integer(edge$j),
                    x = 1,
                    dims = rep(20, 2),
                    use.last.ij = TRUE
)
```

We can then use the `graph.adjacency()` function from the `igraph` package to build a graph

```
plot(graph.adjacency(adj,mode = "undirected"))
```