'MATH+ECON+CODE' MASTERCLASS ON MATCHING MODELS, OPTIMAL TRANSPORT AND APPLICATIONS

Alfred Galichon (NYU)

Vectors and matrices in R

- One of the reason the R language is so appealing is the powerful matrix algebra functionalities. However, there are a few points to understand to make efficient use of it. This tutorial is a brief introduction to these topics – vectorization, operations on vectors and matrices, higher-dimensional arrays, Kronecker products and sparse matrices.
- This is *not* a tutorial on R itself. They are plenty good ones available on the web.

- R is a language based on matrices; however matrices are represented in a *vectorized* way as a sequence of numbers in the computer's memory. This representation can involve either stacking the lines, or stacking the columns
- Different programming languages can use either of the two stacking conventions:
  - Stacking the columns (Column-major order) is used by Fortran, Matlab, R, and most underlying core linear algebra libraries (like BLAS). A 2×2 matrix A is then vectorized as $(A_{11}, A_{21}, A_{12}, A_{22})$. Thus, we shall remember that R represents matrices by **varying the first index first**.
  - Stacking the lines (Row-major order) is used by C, and is the default convention for for Python (Numpy).
- More generally, a $2 \times 2 \times 2$ 3-dimensional array A will be represented as $(A_{111}, A_{211}, A_{121}, A_{221}, A_{112}, A_{212}, A_{122}, A_{222})$, and so on.
- See 'B00_vectorization.R'.

- If f is a function defined to take a scalar argument, then applying f to a vector will return the value of f on the first entry of the vector and will generate a warning message. E.g. f(c(1,2)) will return the same as f(1).
- We may want to apply f on each entries of the vector. In this case, one needs to *vectorize* the function, by doing Vectorize(f)(c(1,2)), which will return [f(1),f(2)].
- Note that a number of built-in operators and functions are already vectorized.
  - E.g. c(-2,2)>0 will produce [FALSE, TRUE]; ifelse(c(-2,2)>0,1,-1) will return [-1,1].
  - But all functions are not vectorized, e.g. sum, max...
- See 'B00_vectorizingFunctions.R'.

- Contrary to Matlab, in R, * is not the matrix product, but the termwise product. Calling v1 * v2 thus requires that v1 and v2 have the same length and dimensions, unless v1 and v2 are two vectors and the dimension of one is a multiple of the dimension of the other, in which case the smaller vector is repeated the ad-hoc number of times to equate its size with the other one.
- When multiplying a matrix by a vector with the same size, we will get a matrix with the same dimensions as the first matrix, and whose vectorized entries will be the product of the vectorized matrix times the vector.
- See 'B00_vectorMultiplication.R'.

- Let $A[i,j,k]$ a 3-dimensional array of dimension $I \times J \times K$. Consider f a function that takes a vector argument, and assume f is not vectorized.
- apply(X = A,MARGIN = 2,FUN = f) will produce a 1-dimensional array B whose entry $B[j]$ will be $f\left(A[,j,]\right)$.
- apply(X = A,MARGIN = c(1,3),FUN = f) will produce a 2-dimensional array B whose entry $B[i,k]$ will be $f(A[i,,k])$.
- See 'B00_apply.R'.

- For A an $n \times m$ matrix and B an $p \times q$ matrix, the Kronecker product $A \otimes B$ is the $np \times mq$ matrix defined as

$$A \otimes B = \begin{pmatrix} a_{11}B & ... & a_{1m}B \\ ... & ... & ... \\ a_{n1}B & ... & a_{nm}B \end{pmatrix}.$$

- One has

$$(A \otimes B)^\top = A^\top \otimes B^\top$$

and, provided the dimensions are compatible,

$$(A \otimes B) . (C \otimes D) = (A.B) \otimes (C.D),$$

and

$$vec\left(BXA^\top\right) = (A \otimes B)\, vec(X).$$

- See 'B00_kronecker.R'.

- Sparse matrices are handled in R using a specific package called Matrix. library(Matrix).
- The foremost instance of this class is identity matrices, which are constructed using Diagonal(n). We can also specify the nonzero elements by using sparseMatrix(i = ..., j = ..., dims = ..., x = ...)
- Note that some operations preserve sparsity, others don't.
  - Inverse does not
  - Sum or product with a dense matrix does not
  - Sum or product with a sparse matrix does
  - Multiplication by a scalar does
  - Kronecker with a full or sparse matrix does
- See 'B00_sparseMatrices.R'.