Matrices, arrays, lists, and apply functions

Lecture 6, CPSC 499 Fall 2018

$\begin{pmatrix} 5 & 1 & 3 \\ 6 & 2 & 7 \\ 9 & 1 & 2 \end{pmatrix}$ Matrices vs. data frames

V1	V2	V3
a	1.5	TRUE
a	6	FALSE
b	20	TRUE

- Both are two-dimensional representations of data
- Both can be indexed like `x[a, b]`
- Both have row and column names

- Data frames: each column is one data type
- Matrix: whole thing is one data type

Constructing a matrix

- Use the `matrix` function
- First argument is a vector of values to go into it
- `nrow` and `ncol` arguments to tell it how many rows and columns
- `dimnames` is a list of two vectors, for row names and column names
- `byrow` determines how those values from the vector get put into the matrix (by rows or columns)

Matrices and data frames under the hood

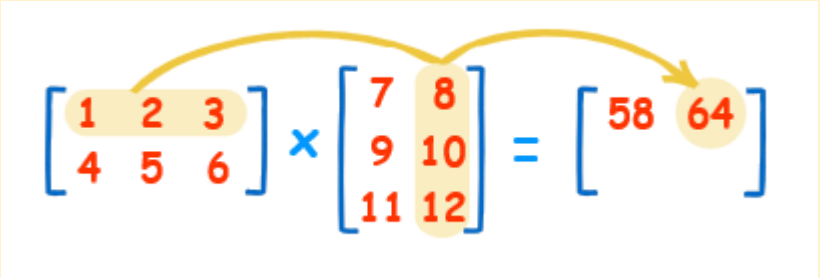
- A data frame is basically a list with rules about all vectors being the same length
  - Indexing with `$` or `[[]]` works like list
- A matrix is basically a vector with some extra attributes describing how to work with it in two dimensions
  - Indexing like `x[a]` works as though matrix were expanded into a vector

- Arithmetic tends to be much faster with matrices than with data frames (vectorization)

Importing data to a matrix

- Can do `as.matrix(read.table())`
  - Be sure to use `row.names` and `header` as appropriate
  - `read.table` makes a data frame, `as.matrix` converts it to matrix
- `matrix(scan())` is faster but less user-friendly
  - `scan` creates a vector with all values from the file
  - put it into a matrix with `matrix`, and set `nrow` and `ncol`

Matrix math



- Scalar and vectorized arithmetic still works
  - `mymatrix * 2` etc.
- Transpose (switch rows and columns) with the `t` function
- Do matrix multiplication with `%*%`
  - Multiplies every row of the first matrix by every column of the second matrix, and takes the sum
  - Independent variables `%*%` effect sizes = predicted values

Image: mathisfun.com

Mini exercise – matrix visualization

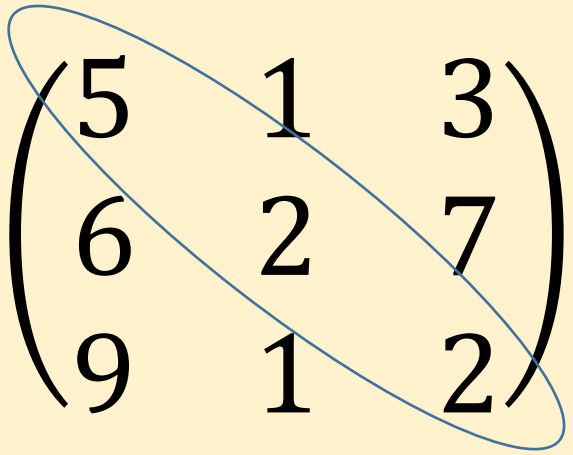
- Use the `image` function on the SNP matrix. What do you get?
- Now try the `heatmap` function on the SNP matrix. Are the results helpful? Maybe use `pdf` to generate a very large PDF for browsing through it.

Distances between rows of a matrix

- `dist` function
- Returns `dist` object, which we can make into a square matrix with `as.matrix`
- Visualize with `cmdscale` (principal coordinates analysis)
- Various packages use dist; for example see `ape` for phylogenetics
- Useful for seeing which of your observations are most similar/dissimilar to each other

Special subsets of square matrices

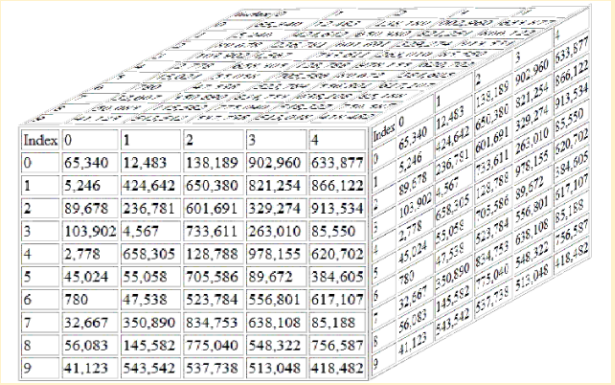
- `diag` returns the diagonal
- `upper.tri` and `lower.tri` create a Boolean matrix for indexing a matrix to get upper or lower triangles



Combining matrices

- If you have two matrices with the same number of rows, you can combine them into one matrix with `cbind`
- Likewise if they have the same number of columns, you can combine them with `rbind`
- `cbind` and `rbind` also work on data frames

Multidimensional arrays



Index	0	1	2	3	4
0	65,340	12,483	138,189	902,960	633,877
1	5,246	424,642	650,380	821,254	866,122
2	89,678	236,781	601,691	329,274	913,534
3	103,902	4,567	733,611	263,010	85,550
4	2,778	658,305	128,788	978,155	620,702
5	45,024	55,058	705,586	89,672	384,605
6	780	47,538	523,784	556,801	617,107
7	32,667	350,890	834,753	638,108	85,188
8	56,083	145,582	775,040	548,322	756,587
9	41,123	543,542	537,738	513,048	418,482

- Like a matrix but can have any number of dimensions, 3D, 4D, etc.
- Use the `array` function and give it a vector of data
- Use the `dim` argument to show how many dimensions and how big
  - `c(5, 10, 20)` will have 5 x 10 x 20 = 1000 values
- Use `dimnames` argument to add names

Image: https://blog.digilentinc.com/arrays-theyre-not-so-scary/

Arrays cont'd

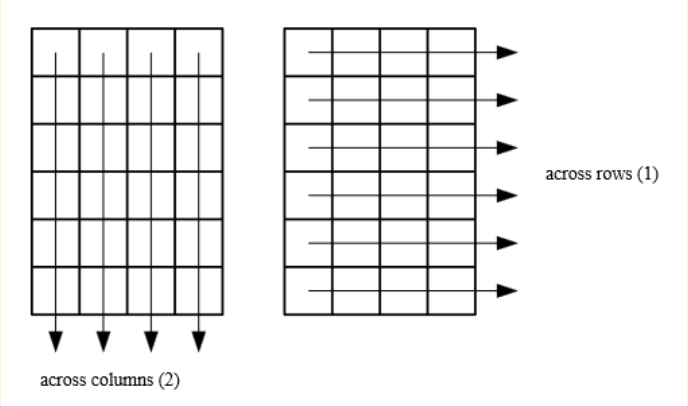
- Index with square brackets and commas
  - `myarr[2, 8, 12]` for a 3D array
- Use the `aperm` function to transpose
- Do you want data in multiple dimensions, or do you want a data frame with grouping factors?
  - (Depends on what you plan to do with it)

Math across dimensions

- `rowSums`, `rowMeans`, `colSums`, `colMeans`
- Work on data frames, matrices, and arrays
- Computationally efficient
- If using on multidimensional arrays, experiment to see how they work
- Output vectors from matrices and data frames

Functions across dimensions

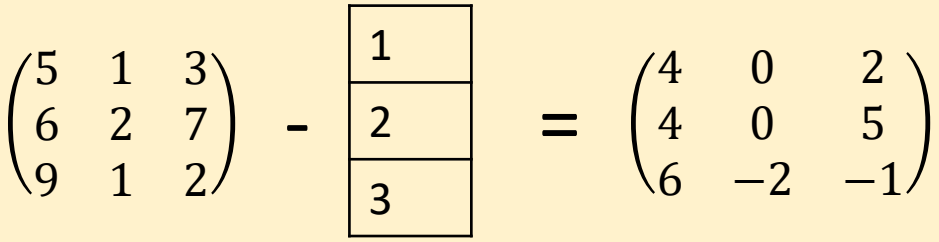
- What if you want the maximum of every row?
- Use the apply function
  - `apply(mymat, MARGIN = 1, FUN = max, na.rm = TRUE)`
- Use `MARGIN = 2` for columns, 3 for third dimension, etc.
- Works with your own custom functions
- Less code than writing a loop but similar computational time



Mini exercise

- What does the output of `apply` look like when you use `MARGIN = c(1, 2)`?
- Try it on a small matrix, then on a small 3D array.
- Use a simple function like `max` for testing.
- Adding rownames/colnames/dimnames may make the results easier to understand.

sweep



- Say I want to center a matrix by subtracting the mean from every column
- ```
  mymat <- sweep(mymat, MARGIN = 2,
                 STATS = colMeans(mymat),
                 FUN = "-")
  ```
- Also works on multidimensional arrays
  - e.g. could subtract a matrix from a 3D array
  ```

Working with lists

Lists


flamephoenix1991 on Flickr

- Example: output of gregexpr
- A vector that can store anything
  - Values of different types
  - Vectors
  - Larger objects
  - More lists
- Index to get a smaller list using `[]`
- Index to get individual elements using `[[]]` or `$`

Lists for function output

- What if you have a function and you want it to return multiple items?
- Put those items into a list and return the list
- Generally a good idea to use names so it is obvious what each item is
- ```
  return(list(VCF_header = headers,
              Genotype_table = geno_tab))
  ```
- ```
  myoutput$Genotype_table
  ```

`lapply` and `sapply`

- Applies a function to every item in a list
- `lapply` returns a list containing the output of the function for every item
- `sapply` returns a vector if possible
- Could use `sapply` with `length` function on the output of `gregexpr` from lab last week
- These functions also work on vectors, but there are not many cases where you need them on vectors

`mapply`

- Applies a function over multiple vectors or lists
- Each vector/list corresponds to one argument to the function
- Output is vector, array, or list, like output of `sapply`

`tapply`

- We covered briefly in first week
- Split a vector up based on a grouping factor from another vector
- Apply a function to each sub-vector
- The `by` function can similarly split rows of a data frame up by a grouping factor

Mini exercise

- Take our SNP matrix
- Get genetic groups from Msi_groups_and_phenotypes.csv from Week 3
- Use `by` to get allele frequencies for each genetic group
- (Split the rows of the matrix by genetic group, then use colMeans on submatrices to get allele freq)

Oct. 23 Midterm format

- Covers material through Oct. 11
- 30 points out of 100 points total for the class
- On Compass, but you must physically be in classroom
- 20 points (2/3 of exam): multiple choice and short answer, evenly spread over first 7 weeks of material
- 10 points (1/3 of exam): write a function to do X, where there are several choices for X
- Open notebook, can use Google etc. but must work alone