# Computing for Math and Stats

Lecture 4

# Creating Matrices

- Matrices can be created like arrays.
- We can specify all the elements explicitly arranged in rows
  - A = [1 2 3; 4 5 6; 7 8 9]
- We can specify them implicitly
  - A = [1:3; 4:6; 7:9]
  - (semicolons can be replaced by newlines, according to taste)
- We can build them up from submatrices

# Building up Matrices

- Consider the following
  - A = [1:3; 4:6; 7:9]
- Expression 1:3 looks like a row vector…
- We can put in there whatever row vector we like
  - B = [A(3,:); A(2,:); A(1,:)]
- We can do the same with column vectors
  - C = [A(:,3), A(:,2), A(:,1)]

# Building up matrices

- We can build up matrices as a series of rows separated by semicolons
- We can build up matrices as a series of columns, separated by commas
- Why not build up matrices as a series of submatrices separated by either colons or commas

# Building up Matrices

- For example
  - C = [A,A,A]
  - D = [A;A;A]
- We can cary the idea further
  - G=[A,ones(3,1);ones(1,4)]
  - F=[eye(3),ones(3,1);ones(1,4)]
  - H=[A,A';A',A]

# Building up Matrices

- These techniques can be used to generate matrices
  - for testing
  - that are sparse (have many zeros)
  - that have some regularity
- These are rather powerful and very elegant techniques
  - Test them to make sure that they work the way you think they work
  - Use them with care
  - Only when they are the simplest route to solve the problem

# Accessing Matrices

- Matrices are accessed with parentheses “(“ and “)”
- This is different from most programming languages that use square brackets for array (matrix, vector) accessing
- Everything in Matlab is a matrix (kind of)
  - In old languages like lisp everything was a list

# Accessing Matrices

- Consider the following
  - A(1:2,2:3)
- This is a submatrix of A that comprises the first two rows and the last two columns
- Can also write
  - A(1:2,:)
- Saves keystrokes and confuses outsiders

# Accessing Matrices

- The expression
  - 1:2 is a row vector
- We can put in there any row vector we want
  - A([1, 3],[3, 1])
- It is quite flexible
- See shuffle2x2.m

# Adding More Elements

- Let
  - V=[1:2:10]
- V(6) is undefined
- But
  - V(6) = 4 expands the size of the matrix
- V(1,6) is what?
- Can we now do V(3,2)=12 ?

# Deleting Elements

- We do not need this too often, but useful to have
- Let V=[1:10]
- We delete an element with
- V(3)=[ ]

# Usefull Built-in Functions

- We know about eye(), ones(), zeros()
- We laso have
  - reshape(A,m,n) to put the elements of a into an m x n matrix
  - length(V) the number of elements in vector V
  - size(M) returns [m,n] the # of rows and # of columns of M
  - diag(V) a diagonal matrix with the elements of V as its diagonal
  - diag(M) the vector of diagonal elements of M
- See playdiag.m, playsize.m