NumPy
Intro to NumPy: Arrays

- NumPy provides a multidimensional array
  - All elements must be the same data type
  - Many different datatypes supported
  - Size is fixed (memory is allocated for the size specified)
- Arithmetic operations work on arrays
- Provides MANY functions that operate on whole arrays
  - These operations are written in a compiled language, and run fast
  - Generally speaking, you want to avoid loops to get the best performance.
    - Can sometimes make code unreadable
- Lots of ways to create arrays
Intro to NumPy: Array Operations

- Arithmetic operator (+, -, /, *) work elementwise
  - A * B is not a matrix product, but instead multiples the corresponding elements in each array together
  - dot(A, B) does a dot product
- Universal functions (sin, cos, exp, ...) work elementwise
- New @ operator
  - Accepted for python 3.5, the “@” will be a new operator in python available for overloading. NumPy will implement it as matrix multiplication
    - http://legacy.python.org/dev/peps/pep-0465/
    - A @ B will be equivalent to np.dot(A, B)
- Array creation and operations examples...
Intro to NumPy: Array Indexing/Slicing

- Biggest source of confusion: selecting a range is best thought of as referring to the “edges” of the array locations
  - Differs from Fortran, IDL

For the array above:

- A[2:4] = [2 3]

- Note also: zero-based indexing

Note: this same behavior applies to Python lists and strings when slicing
Arrays

- Building block of many numerical methods
- Row vs. Column major: $A(m,n)$
  - First index is called the row
  - Second index is called the column
  - Multi-dimensional arrays are flattened into a one-dimensional sequence for storage
  - Row-major (C, python): rows are stored one after the other
  - Column-major (Fortran, matlab): columns are stored one after the other
- Ordering matters for:
  - Passing arrays between languages
  - Deciding which index to loop over first
Arrays

- This is why in Fortran, you want to loop as:

```fortran
double precision :: A(M,N)

do j = 1, N
    do i = 1, M
        A(i,j) = …
    enddo
enddo
```

- And in C:

```c
double A[M][N];

for (i = 0; i < M; i++) {
    for (j = 0; j < N; j++) {
        A[i][j] = …
    }
}
```
Intro to NumPy: Array Indexing/Slicing

- Remember, multi-dimensional arrays are stored in row-major fashion
  - Rows are stored one after the other, within a row, the column data is closest to one another

```
  1  2  3  
  4  5  6
```

- You see this when you print an array:
  - `a = numpy.arange(15).reshape(3,5)`
  - `print a`

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
[10 11 12 13 14]]
```

- Some slicing examples...

Note that the braces [ ] show that the columns are together

3 rows, 5 columns
• When “copying”, need to understand if two arrays, A and B, point to:
  – the same array (including shape and data/memory space)
  – the same data/memory space (but perhaps different shapes)
  – a separate copy of the data (i.e. stored separately in memory)

• B = A (assignment)
  – No copy is made. A and B point to the same data in memory and share the same shape, etc.

• B = A[:] (view or shallow copy)
  – The shape info for A and B are stored independently, but both point to the same memory location for the data

• B = A.copy() (deep copy)
  – The data in B is stored completely separately in memory from A

• Copying examples...
Many fancy ways to index arrays

A[A > 4] = 0

- Boolean indexing
- Similar to IDL's where command

Boolean indexing example...
Avoiding Loops

- Slicing (and using boolean indexing) can be used to avoid loops
More NumPy

- See the tutorial for some other features:
  - Shape manipulation
  - Merging/splitting arrays
  - Fancier indexing
  - Other numpy functions/methods
- NumPy functions page...