Week 4: We're going to survive the midterm!

Amath 301

TA Session

Today's plan

1. Python detail: Eigenvalue/eigenvectors for small matrices
2. Expanding on Slice notation
3. Sorting data programatically
4. Debugging: The Golden Rule(s)
5. Practice debugging (Exercises)

Ordering of eigenvectors

Matlab

- `eig` and `eigs` guarantee the order of eigenvalues/eigenvector pairs.

Python

- `np.linalg.eig` - For large matrices, order is generally respected. For small matrices, this is not gaurenteed. Artifact of the algorithm, not an intentional feature.
- `np.linalg.eigh` - Gaurenteed order, least to greatest (ascending). Good for symmetric matrices! Flip the output to get largest to smallest.

When in doubt, check documentation.

Beyond slice notation - subsets

As on HW3, we wanted a good way to pull out a few columns.

Toy example: a 3x3 matrix, get first and third column.

Matlab

```
cols_to_extract = [1,3];  % Put these integers into a vector
data_matrix = [1,2,3;2,4,6;,3,6,9];  % some simple data
first_and_third = data_matrix(:, cols_to_extract)  % all rows, some cols
```

Python

```
cols_to_extract = np.asarray([1,3])   % Put these indices into an array
data_matrix = np.asarray([[1,2,3],[2,4,6],[3,6,9]])  % some simple data
first_and_third = data_matrix[:, cols_to_extract]  % all rows, some cols
```

Sorting data

Details of how sorting works left for your CS class(es).

Use built-in methods for sorting data.

Generally: Compute a sort, store the indexes to obtain sort. Then use the indexes to sort other data.

Matlab

```
x = [1,4,2,3];
[sorted, index] = sort(x);  % get sorted items and reordering vector
new_sorted = x(index);  % can use this to apply the reordering
```

Python

```
x = np.asarray([1,4,2,3])
index_array = np.argsort(x)
sorted = x[index_array]  # sorts along an axis
```

Sorting matrices - it's really about index notation

Continue examples from previous slide

Matlab

```
x = [1,4,2,3];
y = magic(4);
[sorted, index] = sort(x);  % get sorted items and reordering vector
new_sorted = x(index)  % can use this to apply the reordering
z = y(:, index)  % preserves row order, only reorder columns
```

Python

```
x = np.asarray([1,4,2,3])
y = np.arange(16).reshape(4,4)
index_array = np.argsort(x)
sorted = x[index_array]  # sorts along an axis
z = y[:, index_array]
```

Debugging: how to solve your own problems

Golden rules:

1. Don't panic when you see an error message. Seriously, don't.
   a. Don't panic, just in general. **Panic is the enemy of code**.
2. The actual bug is never where the computer says it is. Something went wrong long before it threw an error.
   a. If you only learn one thing today, learn this
3. The computer never lies, and is never wrong. It did exactly what you told it to do. **Exactly** what you told it to, or died trying.
4. Just because it finishes, doesn't mean it works. **Logic errors need to be caught by you**, the computer only catches syntax errors.

Debugging exercises

How to get paid the big bucks: learn to debug code.

The following code examples contains ≥ 1 error.

1. Find the error
2. Determine how to fix it
3. Fix it
4. Go to next bug (go to 1.)

Example 1: Matrix operations

Matrix multiplication

$$A = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} -1.001 & 1 \\ 1 & 1 \end{bmatrix}, \quad c = \begin{bmatrix} -1 & 0 \\ 0 & -3 \end{bmatrix}$$

Compute $AB - BC$.

```
A = [1,2;2,1]; B = [-1, 1;1,1]; C = [-1,0;0,-3];
result = A*B - B*C;
```

```
A = np.asarray([[1,2],[2,1]]); B=np.asarray([[-1,1],[1,1]])
C = np.asarray([[-1,0],[0,3]])
result = A@B - B@C
```

Example 2: More Matrix operations

Matrix multiplication

$$A = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} -1.001 & 1 \\ 1 & 1 \end{bmatrix}, \quad c = \begin{bmatrix} -1 & 0 \\ 0 & -3 \end{bmatrix}$$

Compute $A^T B - CB$.

```
A = [1,2;2,1]; B = [-1.001, 1;1,1]; C = [-1,0;0,-3];
result = A.'*B - B*C;
```

```
A = np.asarray([[1,2],[2,1]]); B=np.asarray([[-1.001,1],[1,1]])
C = np.asarray([[-1,0],[0,-3]])
result = A.T@B - B@C
```

Example 3: Even More Matrix operations

Matrix multiplication

$$A = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} -1.001 & 1 \\ 1 & 1 \end{bmatrix}, \quad c = \begin{bmatrix} -1 & 0 \\ 0 & -3 \end{bmatrix}$$

Compute $AB - CB$.

```
A = [1,2;2,1]; B = [-1.001, 1;1,1]; C = [-1,0;0,-3];
result = B*A - B*C;
```

```
A = np.asarray([[1,2],[2,1]]); B=np.asarray([[-1.001,1],[1,1]])
C = np.asarray([[-1,0],[0,-3]])
result = B@A - B@C
```

Example 4: We get it, Matrix multiplication is important

Matrix multiplication

$$A = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} -1.001 & 1 \\ 1 & 1 \end{bmatrix}, \quad c = \begin{bmatrix} -1 & 0 \\ 0 & -3 \end{bmatrix}$$

Solve $ABx = CB$ for x. There are **2! errors** here to find.

```
A = [1,2;2,1]; B = [-1.001, 1;1,1]; C = [-1,0;0,-3];
result = A.*b \ C.*b;
```

```
A = np.asarray([[1,2],[2,1]]); B=np.asarray([[-1.001,1],[1,1]])
C = np.asarray([[-1,0],[0,-3]])
result = np.linalg.solve(A@b, C@b)
```

Now onto Jacobi

The following codes attempt to do Jacobi iteration, but contains a logic error. All are syntactically valid.

Example 5: Jacobi iteration

We're trying to compute the solution by doing 5 steps of Jacobi iteration for:

$$Ax = b, \quad A = \begin{bmatrix} 5 & 2 \\ -1 & 3 \end{bmatrix}, b = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

```
solution_guess = np.zeros(2)
for step in range(5):
    new_guess = np.zeros(2)
    new_guess[0] = (3 - 2 * solution_guess[1]) / 5
    new_guess[1] = (2 - (-1) * solution_guess[0]) / 3
A2 = new_guess
```

```
solution_guess = zeros(1,2)
for step=1:5
    new_guess = zeros(1,2)
    new_guess(1) = (3 - 2 * solution_guess(2)) / 5
    new_guess(2) = (2 - (-1) * solution_guess(1)) / 3
end
A2 = new_guess
```

Example 6: Jacobi iteration: Typos 101

$$Ax = b, \quad A = \begin{bmatrix} 5 & 2 \\ -1 & 3 \end{bmatrix}, b = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

```
solution_guess = np.zeros(2)
for step in range(5):
    new_guess = np.zeros(2)
    new_guess[0] = (2 - 2 * solution_guess[1]) / 5
    new_guess[1] = (2 - (-1) * solution_guess[0]) / 3
    solution_guess = new_guess
A2 = new_guess
```

```
solution_guess = zeros(1,2)
for step=1:5
    new_guess = zeros(1,2)
    new_guess(1) = (2 - 2 * solution_guess(2)) / 5
    new_guess(2) = (2 - (-1) * solution_guess(1)) / 3
    solution_guess = new_guess
end
A2 = new_guess
```

Example 7: Jacobi iteration: Typos 102

$$Ax = b, \quad A = \begin{bmatrix} 5 & 2 \\ -1 & 3 \end{bmatrix}, b = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

```
solution_guess = np.zeros(2)
for step in range(5):
    new_guess = np.zeros(2)
    new_guess[0] = (3 - 2 * solution_guess[1]) / 5
    new_guess[1] = (2 - 1 * solution_guess[0]) / 3
    solution_guess = new_guess
A2 = new_guess
```

```
solution_guess = zeros(1,2)
for step=1:5
    new_guess = zeros(1,2)
    new_guess(1) = (3 - 2 * solution_guess(2)) / 5
    new_guess(2) = (2 - 1 * solution_guess(1)) / 3
    solution_guess = new_guess
end
A2 = new_guess
```

Example 8: Jacobi iteration: Are you sure where you're going?

$$Ax = b, \quad A = \begin{bmatrix} 5 & 2 \\ -1 & 3 \end{bmatrix}, b = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

```
solution_guess = np.zeros(2)
for step in range(2,8):
    new_guess = np.zeros(2)
    new_guess[0] = (3 - 2 * solution_guess[1]) / 5
    new_guess[1] = (2 - (-1) * solution_guess[0]) / 3
    solution_guess = new_guess
A2 = new_guess
```

```
solution_guess = np.zeros(2)
for step=2:7
    new_guess = zeros(1,2)
    new_guess(1) = (3 - 2 * solution_guess(2)) / 5
    new_guess(2) = (2 - (-1) * solution_guess(1)) / 3
    solution_guess = new_guess
end
A2 = new_guess
```

Example 9: The if statement

If $x > 100$, we want to set y to 0. If not, leave y as is.

```
y = 1
x = sqrt(10)
if x >= 20
    y = 0
end
```

```
y = 1
x = np.sqrt(10)
if x >= 20
    y = 0
```

Example 10: More fun with if

Loop over the numbers 1 through 10. Print/display numbers between 3.5 and 7.5.

```
for i = 1:10
    if i > 3 && i < 7
        i  % no semicolon here to do print it out
    end
end
```

```
for i in range(1, 11):
    if i > 3 and i < 7:
        print(i)
```