Introduction to Sage

Open the page of Sage cells at `gvsu.edu/s/0Ng`. There is also a Sage cell server at `https://sagecell.sagemath.org/`.

- In the first cell, enter

  ```
  2 + 5
  ```

  and evaluate the cell using either the button below the cell or by pressing Shift-Enter.

- Still using the first cell, enter

  ```
  2 + 5
  2 * 5
  ```

  and evaluate.

  General principle: *Sage only reports the result of the last operation in a cell.*

- Try this:

  ```
  print(2+5)
  2 * 5
  ```

- We can store results using variables:

  ```
  a = 2*5
  ```

  Notice that Sage finishes silently. While it does compute `2*5`, it then assigns the result to `a` so the last operation performed in the cell is the assigment.

- We can check the value of the variable by evaluating

  ```
  a = 2*5
  a
  ```

- Results obtained in one cell are available in other cells. In the second cell, evaluate

  ```
  2^a
  ```

- To define the matrix $A = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 1 & 3 \end{bmatrix}$, use

  ```
  A = matrix(2, 3, [1, 2, 1, 2, 1, 3])
  ```

  There are three arguments to the `matrix` command: the number of rows, the number of columns, and a Python `list` consisting of the entries read across the rows.

  To find the reduced row echelon form, use

  ```
  A.rref()
  ```

  General principle: *Once you define a* `thing`*, you can perform a natural* `action` *on the* `thing` *using*

  ```
  thing.action()
  ```

  Take note of the open and close parentheses.

- Now define the matrix $A = \begin{bmatrix} 1.0 & 2 & 1 \\ 2 & 1 & 3 \end{bmatrix}$, taking note that one entry has been changed to floating point. Find the reduced row echelon form of A. What do you notice?

  Mathematically, we know these are the same matrices, but Sage views the entries of the matrix as elements in a field in which any computations are performed. We don't need to worry about this now; just be aware of it.

  Incidentally, you can use

  `n(A.rref(), digits=3)` to modify the appearance of the result. I didn't show this to students in MTH 227.

- Define the matrix $A = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$. Vectors may be defined by

  ```
  v = vector([3, -2])
  ```

  This is a simple syntax: just provide a list of the elements of the vector.

  To multiply:

  ```
  A * v
  ```

  To augment:

  ```
  A.augment(v)
  ```

  Operations can be concatenated:

  ```
  A.augment(v).rref()
  ```

- We can assemble matrices from vectors:

  ```
  v1 = vector([2, -3])
  v2 = vector([1,4])
  matrix([v1, v2])
  ```

  Here we define two vectors and create a matrix using a `list` of vectors.

  Does this produce what you want? How can you make it produce what we want? Hint: `thing.action()`.

- We can also pull vectors out of matrices. What does this code do?

  ```
  A = matrix(2, 2, [1, 2, 2, 1])
  b = vector([3,0])
  x = A.augment(b).rref().column(2)
  x
  ```

  Helpful Python fact: Like most computer languages, Python starts counting at `0` so the third column of a matrix is indexed by `2`.

  Helpful Python trick: `A.column(-1)` pulls out the last column, `A.column(-2)` the next to last column, and so forth.

- We can also pull columns out of A into a matrix:

  ```
  A = matrix(2, 4, [3,-2,0,4,-1,-1,2,2])
  B = A.matrix_from_columns([2, 1])
  ```

- The $n \times n$ identity matrix is

```
I = identity_matrix(n)
```

Suppose that $A = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$. Evaluate

```
A - 3*I
```

- **Inverses:** Find A^{-1} using three different methods:

a) `thing.action()`

b) Augment A to $[\, A \mid I \,]$, row reduce, and extract.

c) `A^-1`

- Find the determinant of A.

- Find the eigenvalues of A.

- To find a basis for the null space Nul(B), use

```
B.right_kernel()
```

Using the matrix A above, find a basis for the eigenspaces E_3 and E_{-1}.

- Define the stochastic matrix $S = \begin{bmatrix} 0.4 & 0.3 \\ 0.6 & 0.7 \end{bmatrix}$. We know that $\lambda = 1$ is an eigenvalue.

What happens when we try to find Nul$(S - I)$?

How would you handle this with your students?

- Go back to $A = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$. The command

```
A.eigenmatrix_right()
```

attempts to diagonalize A and returns a pair of matrices D and P.

To save them, use

```
D, P = A.eigenmatrix_right()
```

Now verify that $A = PDP^{-1}$.

- What happens when we try to diagonalize $A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$?

- **Danger:** The `eigenmatrix_right` command tries to compute exactly in the ring defined by the matrix. If you try

```
S.eigenmatrix_right()
```

it will fail. You instead need to tell Sage to compute in a ring called `RDF`, which enables floating point approximate arithmetic. Use either

```
A = matrix(RDF, 2,2,[0.4,0.3,0.6,0.7])
A = matrix(RDF, A) # if A has already been defined
```

- You can create pages with custom commands. Go to `http://gvsu.edu/s/0TD` where you will find a page that contains three commands to implement the power method:

  `power(A, x, N)`

  iterates the power method N times, with an initial vector x, giving an approximation to the dominant eigenvalue and a corresponding eigenvector.

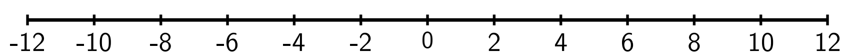
  `inverse_power(A, x, N)`

  iterates the inverse power method N times giving an approximation to the smallest (in absolute value) eigenvalue and a corresponding eigenvector.

  `find_closest_eigenvalue(A, s, x, N)`

  finds the eigenvalue closest to s and a corresponding eigenvalue.

  Use these commands to find the eigenvalues of the matrix B defined on that page.



- You should learn enough python to write a loop:

```
for i in range(10):
    print(i)
```

  and to define a function:

```
def fibonacci(n):
    if n == 0:  return 0
    if n == 1:  return 1
    return fibonacci(n-1) + fibonacci(n-2)
```

Quick reference: There is a useful "quick reference," a 2-page document outlining linear algebra commands in Sage. To find it, google `sagemath linear algebra quick reference`