Group:

Amina Assal (aa4290)

Ivan Barral (iab2131)

Rafail Khalilov (rk2960)

Myric Lehner (mhl2157)

Red Pandas

Introduction

We are looking to create a language that recreates some of Numpy's core functionality natively so that compiled code can quickly churn through linear algebra problems, which would be useful in applications of robotics, data compression, and building neural networks in machine learning. Users can write programs to solve systems of equations and manipulate data in matrices. We would like this to be readable, intuitive, and fast, although realistically we will achieve at most two of those goals. In order to do this, we plan on implementing this as an object oriented language, centered around matrix objects. We plan on implementing type inference and allowing the user to define functions. We're thinking about implementing indexing as well as a Map() operation that would allow mapping between two matrix objects. The user defined functions will be passable into core operations and functions as well as allowing for a robust library to be developed.

Data Types

Int / Float

```
num1 = 3
num2 = 3.0
```

Tuple

```
tup = (0, 1)
```

Character

```
print("Hello World")
```

Vector

```
vector = vec([3,2,1])
vector_alt = [   [3],
                 [2],
                 [1]   ]
```

2-dimensional matrix

```
mat = [ [3,2,1],
        [1,3,4],
        [1,3,5] ]
mat_alt = populate(0, 3, 3) #creates a 3x3 zero matrix
```

Intrinsic matrix attributes:

- Row & Column by index methods that return the appropriate vector
- Rank
- Kernel
- Image

Operations:
Add, Subtract

```
mat = populate(0, 3, 3) #creates a 3x3 zero matrix
mat2 = populate (2,3,3) #creates a 3x3 matrix with value two
newMat = mat2 + mat
mat = newMat - mat
```

Multiply, Power

```
mat = populate(1, 3, 3) #creates a 3x3 one matrix
mat2 = populate (2,3,3) #creates a 3x3 matrix with value two
mat3 = mat2^3 * mat1
mat3*= mat2
```

Map

```
mat = populate(2, 3, 3) #creates a 3x3 matrix of 2s
mat = Map(mat, {_*2}) #returns 3x3 matrix of 6s
```

Transpose

```
mat = populate(0, 3, 3) #creates a 3x3 zero matrix
mat = mat^T
mat = transpose(mat)
```

Determinant

```
mat = populate(1, 3, 3) #creates a 3x3 one matrix
determinantVal = det(mat)
```

Inverse

```
mat = populate(1, 3, 3) #creates a 3x3 one matrix
Inv_mat = inv(mat)
Inv_mat_alt = mat^(-1)
```

Rank

```
mat = populate(1, 3, 3) #creates a 3x3 one matrix
rank = rank(mat)
```

Gauss Jordan (RREF)

```
mat = populate(1, 3, 3) #creates a 3x3 one matrix
Reduced_row= rref(mat)
```

Row Echelon

```
mat = populate(1, 3, 3) #creates a 3x3 one matrix
row_echelon = ref(mat)
```

Eigenvalues, Eigenvectors

```
mat = populate(1, 3, 3) #creates a 3x3 one matrix
eigen_vector = eigvec(mat)
```

```
eigen_value = eigval(mat)
```

Diagonalization

```
mat = populate(1, 3, 3) #creates a 3x3 one matrix
D = diag(mat)
```

Dot Product

```
vec1 = vec([1,2,3]) #creates a vector
vec2 = vec([3,2,1])
result = vec1*vec2
```

Cross Product

```
vec1 = vec([1,2,3]) #creates a vector
vec2 = vec([3,2,1])
result = vec1 *x vec2
```

Source Code

```
x_system = [ 1, 1 ]
y_system = [ 2, 3 ]

result = [ [5],
          [8] ]
alt_result = vec([5,8])

def solve_equation(x,y,r):
     system = [ x_system,
               y_system ]

     finalMatrix = system^(-1) * result
     return finalMatrix

final = solve_equation(x_system, y_system, alt_result)

print("final val:")
print(final)
```