CS 3430: SciComp with Py
Assignment 6
Determinants, Inverses, and Cramer's Rule

Vladimir Kulyukin
Department of Computer Science
Utah State University

February 17, 2018

1 Learning Objectives

1. Determinants
2. Matrix Inverse
3. Cramer's Rule
4. Solving Square Linear Systems
5. Numpy

2 Introduction

We will stay with linear algebra in this assignment to learn how to compute determinants, adjoint matrices, and to use Cramer's rule to solve square linear systems. In 2D and 3D, determinants are areas and volumes. In m-dimensional spaces, determinants are critical in computing volumes of m-dimensional boxes, which lies at the very heart of most of integral calculus.

Determinats are useful in that they enable us to compute inverses of matrices. This gives us another way to inverse a matrix without augmenting a matrix with the corresponding identity matrix and then row reducing the augmented matrix.

Cramer's rule, named after the Swiss mathematician Gabriel Cramer (1704 - 1752), is a beautiful method of solving square linear systems. Learning Cramer's rule will add another method to your repertoire of solving linear systems in addition to the Gauss-Jordan method. Knowing Cramer's rule is useful, because it routinely shows up in advanced calculus and many other areas of scientific computing. While Cramer's rule still enjoys much theoretical fame, it is not widely used in linear algebra systems any more, because more efficient matrix algorthms have been designed and discovered in the past 20 years.

This assignment will also give you more exposure to `numpy`, which will come in handy when we get to implementing artificial neural networks and, a bit later, computer vision and machine learning algorithms.

3 The Determinant of a Square Matrix

The determinant of a 1×1 matrix is its single value. This determinant is called the **first-order determinant**. To compute the determinant of a 2×2, let us define a generic 2×2 matrix

$$A = \begin{bmatrix} a_1 & a_2 \\ b_1 & b_2 \end{bmatrix},$$

where a_1, a_2, b_1, b_2 are real numbers. The following equation defines the so-called **second-order determinant**.

$$det(A) = \begin{vmatrix} a_1 & a_2 \\ b_1 & b_2 \end{vmatrix} = a_1 b_2 - a_2 b_1.$$

Let us work out an example and compute the determinant of this matrix

$$A = \begin{bmatrix} 2 & 3 \\ 1 & 4 \end{bmatrix}.$$

Here is the actual computation.

$$det(A) = \begin{vmatrix} 2 & 3 \\ 1 & 4 \end{vmatrix} = 2 \cdot 4 - 3 \cdot 1 = 5.$$

The determinant of a 2×2 matrix is the area of the parallelograms defined by the 2D row vectors in the matrix. Let us define the determinant of a 3×3 matrix, i.e., the **third-order determinant**. Here is a generic 3×3 matrix

$$A = \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix}.$$

The third-order determinant is computed as follows:

$$det(A) = \begin{vmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{vmatrix} = a_1 \cdot \begin{vmatrix} b_2 & b_3 \\ c_2 & c_3 \end{vmatrix} - a_2 \cdot \begin{vmatrix} b_1 & b_3 \\ c_1 & c_3 \end{vmatrix} + a_3 \cdot \begin{vmatrix} b_1 & b_2 \\ c_1 & c_2 \end{vmatrix}.$$

Let us compute the determinant of this matrix

$$A = \begin{bmatrix} 2 & 1 & 3 \\ 4 & 1 & 2 \\ 1 & 2 & -3 \end{bmatrix}.$$

The determinant of A is

$$det(A) = \begin{bmatrix} 2 & 1 & 3 \\ 4 & 1 & 2 \\ 1 & 2 & -3 \end{bmatrix} = 2 \cdot \begin{vmatrix} 1 & 2 \\ 2 & -3 \end{vmatrix} - 1 \cdot \begin{vmatrix} 4 & 2 \\ 1 & -3 \end{vmatrix} + 3 \cdot \begin{vmatrix} 4 & 1 \\ 1 & 2 \end{vmatrix} = 2 \cdot (-7) - (-14) + 3 \cdot (7) = 21.$$

The determinant of a 3×3 matrix is the volume of the box, or parallelepiped, defined by the 3D row vectors in the matrix.

You may have noticed that we have defined the second-order determinant in terms of the first-order determinants and the third-order determinant in terms of the second-order determinants. If your intuition has now kicked in and senses the presence of recursion, it is spot on! The fourth-order determinant is defined in terms of the third-order determinants, the fifth-order in terms of the fourth-order determinants and so on.

To capture this intuition formally so that we can implement it, we define the **minor matrix** A_{ij} of an $n \times n$ matrix A. The minor matrix A_{ij} is an $(n-1) \times (n-1)$ matrix obtained from A by removing row i and column j. For example, let

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}.$$

Let's compute the minor matrices A_{11}, A_{12}, and A_{13} of A.

$$A_{11} = \begin{bmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{bmatrix};$$

$$A_{12} = \begin{bmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{bmatrix};$$

$$A_{13} = \begin{bmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}.$$

Implement the function `minorMat(A, i, j)` that takes an $n \times n$ matrix A and computes its minor matrix A_{ij}. Below are a few trial runs.

```
>>> import numpy as np
>>> A = np.array([
    [2, 3],
    [1, 4],
    ],
    dtype=float)
>>> minorMat(A, 0, 0)
array([[ 4.]])
>>> minorMat(A, 0, 1)
array([[ 1.]])
>>> minorMat(A, 1, 0)
array([[ 3.]])
>>> minorMat(A, 1, 1)
array([[ 2.]])
>>> A = np.array([
    [4, 2, 1, 3],
    [-1, 0, 2, 8],
    [5, -6, 0, -1],
    [0, 2, 2, 3]
    ],
    dtype=float)
>>> minorMat(A, 0, 0)
array([[ 0.,  2.,  8.],
       [-6.,  0., -1.],
       [ 2.,  2.,  3.]])
>>> minorMat(A, 2, 2)
array([[ 4.,  2.,  3.],
       [-1.,  0.,  8.],
       [ 0.,  2.,  3.]])
>>> minorMat(A, 2, 3)
array([[ 4.,  2.,  1.],
       [-1.,  0.,  2.],
       [ 0.,  2.,  2.]])
```

Determinants can be computed with minor matrices. If A is an $n \times n$ matrix and A_{ij} is its minor, then $det(A_{ij}) = |A_{ij}|$. If A is a 3×3 matrix

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix},$$

then

$$det(A) = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11}|A_{11}| - a_{12}|A_{12}| + a_{13}|A_{13}|.$$

The numbers $|A_{11}|$, $-|A_{12}|$, and $|A_{13}|$ are called **cofactors**. The cofactor c_{ij} of the entry a_{ij} in an $n \times n$ matrix A is defined as

$$c_{ij} = (-1)^{i+j} det(A_{ij}),$$

where A_{ij} is the minor of A. The determinant of an $n \times n$ matrix A is

$$det(A) = \begin{vmatrix} a_{11} & a_{12} & ... & a_{1n} \\ a_{21} & a_{22} & ... & a_{2n} \\ . & . & ... & . \\ . & . & ... & . \\ . & . & ... & . \\ a_{n1} & a_{n2} & ... & a_{nn} \end{vmatrix} = a_{11}c_{11} + a_{12}c_{12} + ... + a_{1n}c_{1n}.$$

Let's work out an example and compute the determinant of the matrix

$$A = \begin{bmatrix} 5 & -2 & 4 & -1 \\ 0 & 1 & 5 & 2 \\ 1 & 2 & 0 & 1 \\ -3 & 1 & -1 & 1 \end{bmatrix}.$$

$$det(A) = \begin{vmatrix} 5 & -2 & 4 & -1 \\ 0 & 1 & 5 & 2 \\ 1 & 2 & 0 & 1 \\ -3 & 1 & -1 & 1 \end{vmatrix} = 5(-1)^2 \begin{vmatrix} 1 & 5 & 2 \\ 2 & 0 & 1 \\ 1 & -1 & 1 \end{vmatrix} + (-2)(-1)^3 \begin{vmatrix} 0 & 5 & 2 \\ 1 & 0 & 1 \\ -3 & -1 & 1 \end{vmatrix}$$

$$+4(-1)^4 \begin{vmatrix} 0 & 1 & 2 \\ 1 & 2 & 1 \\ -3 & 1 & 1 \end{vmatrix} + (-1)(-1)^5 \begin{vmatrix} 0 & 1 & 5 \\ 1 & 2 & 0 \\ -3 & 1 & -1 \end{vmatrix}$$

$$= 5(-8) + 2(-22) + 4(10) + 1(36) = -8.$$

Implement the function `det(A)` that computes the determinant of an $n \times n$ matrix `A`. Below are a few test runs.

```
>>> import numpy as np
>>> A = np.array([
    [5, -2, 4, -1],
    [0, 1, 5, 2],
    [1, 2, 0, 1],
    [-3, 1, -1, 1]
    ],
    dtype=float)
>>> det(A)
-8.0
```

```
>>> A = np.array([
    [2, 3],
    [1, 4],
    ],
    dtype=float)
>>> det(A)
5.0
>>> A = np.array([
    [-5, 0, 2],
    [6, 1, 2],
    [2, 3, 1]
    ],
    dtype=float)
>>> det(A)
57.0
>>> A  = np.array([
    [4, 2, 1, 3],
    [-1, 0, 2, 8],
    [5, -6, 0, -1],
    [0, 2, 2, 3]
    ],
    dtype=float)
>>> det(A)
-352.0
>>> A = np.array([
    [2, 7, -3, 8, 3],
    [0, -3, 7, 5, 1],
    [0, 0, 6, 7, 6],
    [0, 0, 0, 9, 8],
    [0, 0, 0, 0, 4]
    ],
    dtype=float)
>>> det(A)
-1296.0
>>> A = np.array([
    [3, 2, 0, 1, 3],
    [-2, 4, 1, 2, 1],
    [0, -1, 0, 1, -5],
    [-1, 2, 0, -1, 2],
    [0, 0, 0, 0, 2]
    ],
    dtype=float)
>>> det(A)
12.0
```

Implement the function `cofactor(A, i, j)` that computes the cofactor of the entry a_{ij} in an $n \times n$ matrix A. Here are a few examples.

```
>>> import numpy as np
>>> A = np.array([
    [2, 1, 0, 1],
    [3, 2, 1, 2],
    [4, 0, 1, 4],
```

```
    [1, 0, 2, 1]
    ],
    dtype=float)
>>> cofactor(A, 1, 0)
7.0
>>> cofactor(A, 1, 1)
-7.0
>>> cofactor(A, 2, 3)
-3.0
>>> cofactor(A, 3, 1)
3.0
```

There is an amazing property between determinants and cofactors. It turns out that, given an $n \times n$ matrix A, we can pick any row i in A and compute the determinant of A as follows:

$$det(A) = a_{i1}c_{i1} + a_{i2}c_{i2} + ... + a_{in}c_{in},$$

where c_{ij} is the cofactor of a_{ij} in A. This formula is known as the *expansion by minors on the i-th row* of A. Similarly, if we take any column j in A, we can compute the determinant of A as follows:

$$det(A) = a_{1j}c_{1j} + a_{2j}c_{2j} + ... + a_{nj}c_{nj},$$

where c_{ij} is the cofactor of a_{ij} in A. This formula is known as the *expansion by minors on the i-th column* of A.

Implement the functions `expandByRowMinors(A, r)` and `expandByColMinors(A, c)` that compute the determinant of A by expansion by minors on the r-th row or c-th column, respectively. Below are a few test runs.

```
>>> import numpy as np
>>> A = np.array([
    [3, 2, 0, 1, 3],
    [-2, 4, 1, 2, 1],
    [0, -1, 0, 1, -5],
    [-1, 2, 0, -1, 2],
    [0, 0, 0, 0, 2]
    ],
    dtype=float)
>>> for c in xrange(A.shape[1]):
        print (c, expandByColMinors(A, c))
(0, 12.0)
(1, 12.0)
(2, 12.0)
(3, 12.0)
(4, 12.0)
>>> for r in xrange(A.shape[0]):
        print (r, expandByRowMinors(A, r))
(0, 12.0)
(1, 12.0)
(2, 12.0)
(3, 12.0)
(4, 12.0)
```

4 Matrix Inverse

Let A be an $n \times n$ matrix and let A' be the matrix whose entries are the cofactors of the corresponding entries of A. In other words, $A'[i,j] = \texttt{cofactor}(A,i,j)$. The **adjoint matrix** of A is $(A')^T$, i.e., it is the transpose of the cofactor matrix. Let $adj(A)$ denote the adjoint matrix of A. Then there is another way to compute the inverse of an $n \times n$ matrix A:

$$A^{-1} = \frac{1}{det(A)} adj(A).$$

Implement the functions `cofactorMat(A)` and `adjointMat(A)` that take $n \times n$ matrix A and compute its cofactor matrix and adjoint matrix, respectively. Use these functions to implement the function `inverseMat(A)` that computes the inverse of A. Let's define `checkInverse(A)` and use it in a few test runs below.

```
def checkInverse(A):
    D = det(A)
    print(D)
    if D != 0.0:
        print(inverseMat(A))
        print(np.dot(A, inverseMat(A)))
```

```
>>> A = np.array([
    [5, -2, 4, -1],
    [0, 1, 5, 2],
    [1, 2, 0, 1],
    [-3, 1, -1, 1]
    ],
    dtype=float)
>>> checkInverse(A)
-8.0
[[ 1.   -0.5   0.5   1.5 ]
 [-2.75  1.25 -0.5  -4.75]
 [-1.25  0.75 -0.5  -2.25]
 [ 4.5  -2.    1.5   8.  ]]
[[ 1.  0.  0.  0.]
 [ 0.  1.  0.  0.]
 [ 0.  0.  1.  0.]
 [ 0.  0.  0.  1.]]
>>> A = np.array([
    [2, 7, -3, 8, 3],
    [0, -3, 7, 5, 1],
    [0, 0, 6, 7, 6],
    [0, 0, 0, 9, 8],
    [0, 0, 0, 0, 4]
    ],
    dtype=float)
>>> checkInverse(A)
-1296.0
[[ 0.5         1.16666667 -1.11111111 -0.22839506  1.45679012]
 [ 0.         -0.33333333  0.38888889 -0.11728395 -0.2654321 ]
 [-0.          0.          0.16666667 -0.12962963  0.00925926]
 [ 0.         -0.          0.          0.11111111 -0.22222222]
 [-0.          0.         -0.          0.          0.25      ]]
```

```
[[  1.00000000e+00   0.00000000e+00  -4.44089210e-16   0.00000000e+00
    0.00000000e+00]
 [  0.00000000e+00   1.00000000e+00   0.00000000e+00   1.11022302e-16
   -1.66533454e-16]
 [  0.00000000e+00   0.00000000e+00   1.00000000e+00   0.00000000e+00
    2.22044605e-16]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   1.00000000e+00
    0.00000000e+00]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
    1.00000000e+00]]
```

5 Cramer's Rule

Gabriel Cramer discovered a method to compute the solution to $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A}$ is an invertible $n \times n$ matrix. Cramer's rule states that the square linear system $\mathbf{Ax} = \mathbf{b}$ where

$$\mathbf{b} = \begin{bmatrix} b_1 \\ . \\ . \\ . \\ b_n \end{bmatrix}$$

has the following solution:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ . \\ . \\ . \\ x_n \end{bmatrix},$$

where $x_k = det(\mathbf{B}_k)/det(\mathbf{A})$, for $1 \leq k \leq n$ and $\mathbf{B}_k$ is the matrix obtained from $\mathbf{A}$ by replacing the k-th column of $\mathbf{A}$ by the column vector $\mathbf{b}$.

Implement the function `cramer(A, b)` that takes the `A` and `b` components of the square linear system $\mathbf{Ax} = \mathbf{b}$ and uses Cramer's rule to return $\mathbf{x}$. Use your implementation of `cramer(A, b)` to solve the following square linear system.

$$\begin{aligned} 5x_1 - 2x_2 + x_3 &= 1 \\ 3x_1 + 2x_2 + 0x_3 &= 7 \\ x_1 + x_2 - x_3 &= 0 \end{aligned}$$

Let's set up the `A` and `b` components of the linear system.

```
>>> A = np.array([
    [5, -2, 1],
    [0, 1, 2],
    [1, 6, -1]
    ],
    dtype=float)
>>> b = np.array([1, 0, 4], dtype=float)
```

Let's solve it with Cramer's rule and check the returned solution.

```
>>> cramer(A, b)
array([ 0.47142857,  0.54285714, -0.27142857])
>>> np.dot(A, cramer(A, b))
array([ 1.,  0.,  4.])
>>> b
array([ 1.,  0.,  4.])
```

6 What to Submit

Write your solutions in `cs3430_s18_hw06.py` and submit it via Canvas.

Happy Hacking!