1.1

1.1 Matrices over Finite Fields

Division

Question 1

The program to find the inverse of each element of $GF(p)$ is shown on page 7, labeled

```
inverses(p)
```

This function takes the parameter p which is assumed to be a positive integer, the function returns an array of inverses, where the n^{th} entry represents the inverse of n in $GF(p)$.

Testing the programs functionality

```
>> print(inverses(2))
[1]

>> print(inverses(5))
[1, 3, 2, 4]

>> print(inverses(7))
[1, 4, 5, 2, 3, 6]
```

Note if that if p is a positive composite integer then the program will have $0s$ where inverses do not exist

```
>> print(inverses(8))
[1, 0, 3, 0, 5, 0, 7]
```

If we assume that p is prime then we know that $\forall a \in GF(p) \setminus \{0\}$ there exists an a^{-1} such that $aa^{-1} \equiv 1 \mod p$ so $(p-a)(p-a^{-1}) \equiv aa^{-1} \equiv 1 \mod p$. Therefore, if L is our array of inverses (and assume indexing starts at 1) then $L[a] = a^{-1}$ and $L[p-a] = p - a^{-1}$, so only need to calculate inverses for the first half of the list.

Question 2

For each element $a \in GF(p) \setminus \{0\}$ the following operations must occur

1. a^{-1} comparisons must be made to find a^{-1}
2. assignment of a^{-1} in the array

The assignment of an element into an array is of order $\Theta(1)$ so doing this for each $a \in GF(p) \setminus \{0\}$ means step 2 has order $\Theta(p)$ over the whole algorithm.
We know that each comparison has complexity $\Theta(1)$, and the total number of comparisons is $\sum_{a \in GF(p)\setminus\{0\}} a^{-1} = \sum_{n=1}^{p-1} n = \frac{(p-1)p}{2} = \Theta(p^2)$, so step 1 has complexity $\Theta(p^2)$ over the whole algorithm.
Adding both together, we get the complexity of this algorithm is $\Theta(p^2)$

Gaussian Elimination

Question 3

The function to perform Gaussian elimination in $GF(p)$ is shown on page 7, labelled

```
gauss_elim(M, mod)
```

Where `M` is the matrix is to be Gaussian eliminated and `mod` is the size of the finite field we are working in.

```
>> A1 = np.array([[0,1,7,2,10],
                  [8,0,2,5,1],
                  [2,1,2,5,5],
                  [7,4,5,3,0]])
>> print(gauss_elim(A1,11))
[[1 0 3 0 0]
 [0 1 7 0 0]
 [0 0 0 1 0]
 [0 0 0 0 1]]
```

So the matrix A_1 has rank 4 in $GF(11)$ and the row space has basis

$$(1,0,3,0,0)\,,(0,1,7,0,0)\,,(0,0,0,1,0)\,,(0,0,0,0,1)$$

```
>> print(gauss_elim(A1,19))
[[ 1  0  0  0 13]
 [ 0  1  0  0  6]
 [ 0  0  1  0  3]
 [ 0  0  0  1  1]]
```

So the matrix A_1 has rank 4 in $GF(19)$ and the row space has basis

$$(1,0,0,0,13)\,,(0,1,0,0,6)\,,(0,0,1,0,3)\,,(0,0,0,1,1)$$

```
>> A2 = np.array([[6,16,11,14,1,4],
                  [7,9,1,1,21,0],
                  [8,2,9,12,17,7],
                  [2,19,2,19,7,12]])
>> print(gauss_elim(A1,19))
[[ 1  0  0  9 11  9]
 [ 0  1  0 10  5  5]
 [ 0  0  1  9 14  7]
 [ 0  0  0  0  0  0]]
```

So the matrix A_2 has rank 3 in $GF(23)$ and the row space has basis

$$(1,0,0,9,11,9)\,,(0,1,0,10,5,5)\,,(0,0,1,9,14,7)$$

Kernels and Annihilators

Question 4

Assume A an $m \times n$ matrix is in row echelon form and let $L = \{l(1), \ldots, l(r)\}$ where r is the rank of the matrix, then for a given row i, $A_{ij} = 0$ for $j < l(i)$ and $A_{ij} = 0$ for $j = l(k)$ such that $i < k \leq r$. This implies that $A_{ij} = 0$ for all $j \in L \setminus \{l(i)\}$
So to solve the equation

$$A\mathbf{x} = \mathbf{0}$$

Then we require

$$[A\mathbf{x}]_i = 0 \ \forall i \iff \sum_{j=1}^{n} A_{ij}x_j = 0 \ \forall i \iff x_{l(i)} = -\sum_{GF(p)\setminus L} A_{ij}x_j \ \forall i$$

So $\mathbf{x}$ is uniquely determined by x_j such that $j \in GF(p) \setminus L$ So to find a basis of $\ker A$, iterate through basis $\{\mathbf{x'}_1, \ldots, \mathbf{x'}_{n-r}\}$ of the vector space $\{\mathbf{x} \in GF(p)^n : x_j = 0 \ \forall j \in L\}$ and use $[\mathbf{x}]_{l(i)} = -[A\mathbf{x}']_i$ to find a basis of $\ker A$
The function to perform this algorithm is shown on pages 8 and 9, labeled

```
kernel_basis(M, mod)
```

Where `M` is the matrix for which the kernel is to be found and `mod` is the size of the finite field we are working in.

```
>> B1 = np.array([[4,6,5,2,3],
                  [5,0,3,0,1],
                  [1,5,7,1,0],
                  [5,5,0,3,1],
                  [2,1,2,4,0]])
>> print(kernel_basis(B1, 13))
[array([[7],
       [2],
       [1],
       [2],
       [1]])]

>> print(kernel_basis(B1, 17))
[array([[0],
       [0],
       [0],
       [0],
       [0]])]

>> B2 = np.array([[3,7,19,3,9,6],
                  [10,2,20,15,3,0],
                  [14,1,3,14,11,3],
                  [26,1,21,6,3,5],
                  [0,1,3,19,0,3]])
>> print(kernel_basis(B2, 23))
[array([[6],
       [6],
       [9],
       [9],
```

```
        [9],
        [1]])]
```

Question 5

If U is a subspace of F^n then $\dim U + \dim U^{\circ} = \dim F^n = n$

Question 6

For this I defined a function to find a basis of U° as shown on page 9, labeled

`row_annihilator(U, mod)`
Where `U` is a matrix where the rows are the row basis of a subspace U and `mod` is the size of the finite field we are working in. we also define a function `col_annihilator(U, mod)` similarly for column spaces

```
>> U_annihilate = row_annihilator(A1, 19)
>> U_annihilate_annihilate = col_annihilator(U_annihilate, 19)
>> print(U_annihilate)
[[ 6]
 [13]
 [16]
 [18]
 [ 1]]
>> print(U_annihilate_annihilate)
[[ 1  1  0  0  0]
 [10  0  1  0  0]
 [16  0  0  1  0]
 [ 3  0  0  0  1]]
```

Now from Question 4, we know that in $GF(19)$ U has basis

$$(1,0,0,0,13), (0,1,0,0,6), (0,0,1,0,3), (0,0,0,1,1)$$

and from above we know that in $GF(19)$ $(U^{\circ})^{\circ}$ has basis

$$(1,1,0,0,0), (10,0,1,0,0), (16,0,0,1,0), (3,0,0,0,1)$$

Now note that in the field $GF(19)$

$$\begin{aligned}
(1,0,0,0,13) &= 13*(3,0,0,0,1) \\
(0,1,0,0,6) &= 6*(3,0,0,0,1) + (1,1,0,0,0) \\
(0,0,1,0,3) &= 3*(3,0,0,0,1) + (10,0,1,0,0) \\
(0,0,0,1,1) &= \quad (3,0,0,0,1) + (16,0,0,1,0)
\end{aligned}$$

So the 2 bases are equivalent, therefore $U = (U^{\circ})^{\circ}$

Question 7

Now if an $m_1 \times n$ matrix A has row space U and an $m_2 \times n$ matrix B has row space W, then the row space defined by concatenating matrices A and B, i.e by the matrix

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m_1 1} & a_{m_1 2} & \dots & a_{m_1 n} \\ b_{11} & b_{12} & \dots & b_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m_2 1} & b_{m_2 2} & \dots & b_{m_2 n} \end{pmatrix}$$

Is equivalent to $U + W$, so if we Gaussian eliminate the above matrix, we get a basis of the row space of $U + W$. We can use the fact that $U \cap W = (U^\circ)^\circ \cap (W^\circ)^\circ = (U^\circ + W^\circ)^\circ$ to find a basis of $U \cap W$ using only sum and annihilation of vector sub-spaces.

To find the basis of $U+W$ and $U \cap W$ I defined the functions `vector_row_space_sum(U, W, mod)` and `vector_col_space_sum(U, W, mod)` which given matrices `U` and `W` finds the basis of the respective row and column space. I also defined the function `bases(U, W, mod)` to quickly find and return the bases of U, W, $U + W$, $U \cap W$

```
>> U1, W1, U1pW1, U1aW1 = bases(A1, B1, 11)
>> print(U1)
[[1 0 3 0 0]
 [0 1 7 0 0]
 [0 0 0 1 0]
 [0 0 0 0 1]]
>> print(W1)
[[1 0 0 2 0]
 [0 1 0 3 0]
 [0 0 1 4 0]
 [0 0 0 0 1]]
>> print(U1pW1)
[[1 0 0 0 0]
 [0 1 0 0 0]
 [0 0 1 0 0]
 [0 0 0 1 0]
 [0 0 0 0 1]]
>> print(U1aW1)
[[7 5 1 0 0]
 [8 6 0 1 0]
 [0 0 0 0 1]]

>> A3 = np.array([[1,0,0,0,3,0,0],
                  [0,5,0,1,6,3,0],
                  [0,0,5,0,2,0,0],
                  [2,4,0,0,0,5,1],
                  [4,3,0,0,6,2,6]])
>> B3 = np.transpose(row_annihilator(A3, 19))
>> U2, W2, U2pW2, U2aW2 = bases(A3, B3, 19)
>> print(U2)
[[ 1  0  0  0  0  6  1]
 [ 0  1  0  0  0  3 14]
 [ 0  0  1  0  0 16  9]
 [ 0  0  0  1  0  0  8]
 [ 0  0  0  0  1 17  6]]
>> print(W2)
```

```
[[ 1  0  9 18  6  1 12]
 [ 0  1  0  2  0  4 14]]
>> print(U2pW2)
[[1 0 0 0 0 0 0]
 [0 1 0 0 0 0 0]
 [0 0 1 0 0 0 0]
 [0 0 0 1 0 0 0]
 [0 0 0 0 1 0 0]
 [0 0 0 0 0 1 0]
 [0 0 0 0 0 0 1]]
>> print(U2aW2)
[[0 0 0 0 0 0 0]]

>> B4 = np.transpose(row_annihilator(A3, 23))
>> U3, W3, U3pW3, U3aW3 = bases(A3, B4, 23)
>> print(U3)
[[ 1  0  0  0  0  8 22]
 [ 0  1  0  0  0  3 18]
 [ 0  0  1  0  0 21  6]
 [ 0  0  0  1  0  4  0]
 [ 0  0  0  0  1  5  8]]
>> print(W3)
[[ 1  0 17  8 15 21  8]
 [ 0  1  0  3  0  5 17]]
>> print(U3pW3)
[[ 1  0  0  0  0  0 12]
 [ 0  1  0  0  0  0 20]
 [ 0  0  1  0  0  0 20]
 [ 0  0  0  1  0  0 18]
 [ 0  0  0  0  1  0 19]
 [ 0  0  0  0  0  1  7]]
>> print(U3aW3)
[[11  3  3  5  4 16  1]]
```

$\dim U + \dim W = \dim(U + W) + \dim(U \cap W)$

Question 8

In the reals for the case where U is the row space of A_3 and $W = \{x^T \| x \in \ker A_3\}$ we would find that $\dim(U+W) = \dim(\mathbb{R}^7) = 7$ which implies $\dim(U \cap W) = 0$ but in the very last example, we saw that in $GF(23)$, $\dim(U + W) = 6$ and $\dim(U \cap W) = 1$.

Programs

Division

```
def inverses(p):
    inv = []
    for a in range(1,p):
        a_inv = 1
        for a_inv in range(1,p):
            if a*a_inv % p == 1:
                inv.append(a_inv)
                break
        else:
            inv.append(0)
    return inv
```

Gaussian Elimination

```
def transpose(M, i, j ,mod):
    # swap rows i and j
    M = M.copy()
    y = M[i].copy()
    M[i] = M[j].copy()
    M[j] = y
    return M

def multiply(M, i, a, mod):
    # multiply row i by a
    M = M.copy()
    M[i] = a*M[i] % mod
    return M

def subtract(M, i, j, a, mod):
    # subtract a * row j from row i
    M = M.copy()
    M[i] = (M[i] - a*M[j]) % mod
    return M

def gauss_elim(M, mod):
    M = M.copy()
    M = M % mod
    rows = M.shape[0]
    cols = M.shape[1]
    inv = inverses(mod)
    cur_col = 0
    cur_row = 0
    while cur_col < cols and cur_row < rows:
        if M[cur_row][cur_col] == 0:
            for row in range(cur_row, rows):
                if M[row][cur_col]!=0:
                    M=transpose(M, row, cur_col, mod)
                    break
            else:
                cur_col += 1
                continue
```

```
        lead_coef = M[cur_row][cur_col]
        M = multiply(M, cur_row, inv[lead_coef-1], mod)
        for row in range(0, rows):
            if row == cur_row:
                continue
            row_lead_coef = M[row][cur_col]
            M = subtract(M, row, cur_row, row_lead_coef, mod)
        cur_row+=1
        cur_col+=1
    return M
```

Kernels and Annihilators

```
def rank(M, mod):
    M = gauss_elim(M,mod)
    r=0
    rows = M.shape[0]
    cols = M.shape[1]
    cur_col = 0
    cur_row = 0
    while cur_col < cols and cur_row < rows:
        if M[cur_row][cur_col] > 0:
            r+=1
            cur_row+=1
        else:
            cur_col+=1
    return r

def undetermined_x(M, mod):
    M = gauss_elim(M,mod)
    rows = M.shape[0]
    cols = M.shape[1]
    cur_col = 0
    cur_row = 0
    undet_x = []
    while cur_col < cols and cur_row < rows:
        if M[cur_row][cur_col] > 0:
            cur_row+=1
            cur_col+=1
        else:
            undet_x.append(cur_col)
            cur_col+=1
    if cur_col != cols:
        undet_x = undet_x + [x for x in range(cur_col, cols)]
    return undet_x

def kernel_basis(M, mod):
    M = gauss_elim(M ,mod)
    r = rank(M, mod)
    cols = M.shape[1]
    undet_x = undetermined_x(M, mod)
    basis = []
    for i in undet_x:
        x = np.zeros((cols,1), dtype=int)
```

```
        x[i][0]=1
        j=cols-1
        cur_row = r-1
        determined_x = np.matmul(M, x)
        while j>=0:
            if j not in undet_x:
                x[j] = -determined_x[cur_row] % mod
                cur_row -= 1
            j-=1
        basis.append(x)
    if len(basis) == 0:
        basis.append(np.zeros((cols,1), dtype=int))
    return basis

def get_rid_of_empty(U):
    rows = U.shape[0]
    cols = U.shape[1]
    non_empty = []
    for i in range(rows):
        for j in range(cols):
            if U[i][j]!=0:
                non_empty.append(i)
                break
    U=U[non_empty]
    return U

def vector_row_space_sum(U, W, mod):
    # U and W given in the form of a matrix where U has row space U
    # and W has row space W
    U = gauss_elim(U,mod)
    W = gauss_elim(W,mod)
    full_space = np.concatenate((U, W))
    full_space = gauss_elim(full_space,mod)
    full_space=get_rid_of_empty(full_space)
    return full_space

def vector_col_space_sum(U, W, mod):
    row_U = np.transpose(U)
    row_W = np.transpose(W)
    row_full_space = vector_row_space_sum(row_U, row_W, mod)
    full_space = np.transpose(row_full_space)
    return full_space

def row_annihilator(U, mod):
    # U given in the form of a matrix with row space U
    basis = kernel_basis(U, mod)
    matrix_basis = np.concatenate(basis, axis=1)
    return matrix_basis

def col_annihilator(U, mod):
    return np.transpose(row_annihilator(np.transpose(U),mod))

def bases(U, W, mod):
```

```
# U
U = gauss_elim(U, mod)
U = get_rid_of_empty(U)

# W
W = gauss_elim(W, mod)
W = get_rid_of_empty(W)

# U + W
UpW = vector_row_space_sum(U, W, mod)

# U & W
UaW = col_annihilator(vector_col_space_sum(
                        row_annihilator(U, mod),
                        row_annihilator(W, mod),
                        mod),
                    mod)

return U, W, UpW, UaW
```