Homework 1

Problem1:

(a) Let A be the event that one pup is male, and let B be the event that the other pup is male. Obviously, $P(A) = P(B) = 1/2$. Then the probability of both pups are male is:

$$P(A \cap B) = P(A) * P(B) = \frac{1}{4} \tag{1}$$

(b) Let C be the event that the eldest pup is male, and let D be the event that both pups are male. $P(C) = 1/2$, and $P(D) = 1/4$ according to equation(1). Then the probability of both pups are male is:

$$P(D|C) = \frac{P(C \cap D)}{P(C)} = \frac{P(D)}{P(C)} = \frac{1}{2} \tag{2}$$

(c) Let E be the event that at least one pup is male, and let D be the event that both pups are male. $P(E) = 3/4$, and $P(D) = 1/4$. Then the probability of both pups are male is:

$$P(D|E) = \frac{P(E \cap D)}{P(E)} = \frac{P(D)}{P(E)} = \frac{1}{3} \tag{3}$$

(d) Let F be the event that at least one pup is a male born on a Wednesday, and let D be the event that both pups are male. $P(F) = 1 - (13/14)^2 = 27/196$, and $P(D) = 1/4$. Then the probability of both pups are male is:

$$P(D|F) = \frac{P(F|D) * P(D)}{P(F)} \tag{4}$$

Where $P(F|D)$ is the condition probability that at least one pup is a male pup born on Wednesday given that both pups are male.

Therefore, $P(F|D) = 1 - (6/7)^2 = 13/49$.

So,

$$P(D|F) = \frac{P(F|D) * P(D)}{P(F)} = \frac{13}{27}$$

Problem2:

(a) Matrix A is a matrix of $n \times n$, and $rank(A) < n$. Therefore, not all columns of matrix A are linearly independent. In this case, at lease one column is the linear combination of other columns. So, after finite column operations, at least one column can be all 0s. Then $det(A) = 0$.

On the other hand, $det(A) = \prod_{i=1}^{n} \lambda_i$, where λ_i is an eigenvalue of matrix A.

Therefore, A has an eiqenvalue equal to 0.

(b) Suppose there exists at least one eigenvalue λthat is smaller or equal to 0. Let x be the corresponding eigenvector. Therefore,

$$x^T Bx = x^T \lambda x = \lambda |x|^2 \leq 0$$

Which is contradictory to the given condition that $x^T Bx > 0$ for all x.

Therefore, all of the eigenvalues of B are strictly greater than 0.

(c) Let (λ_1, v_1) and $(\lambda_2, \_v_2)$ be two different eigenpairs of matrix C.

$$Cv_1 = \lambda_1 v_1$$

$$Cv_2 = \lambda_2 v_2$$

Multiply the second equation by v_1^T

$$\begin{aligned} \lambda_2 v_2 v_1^T &= (\lambda_2 v_2) \cdot (v_1) \\ &= (Cv_2) \cdot (v_1) \\ &= (Cv_2)^T v_1 \\ &= v_2^T C^T v_1 \\ &= v_2^T C v_1 \\ &= v_2^T (\lambda_1 v_1) \\ &= (v_2) \cdot (\lambda_1 v_1) \end{aligned} \tag{5}$$

Therefore, $(\lambda_2 v_2) \cdot (v_1) = \lambda_2 v_2 \cdot v_1 = \lambda_1 v_2 \cdot v_1$. Since $\lambda_1 \neq \lambda_2$, $v_2 \cdot v_1 = 0$.

As a conclusion, any two of the eigenvectors are orthogonal.

Problem3: The python code of the algorithm to compute Fibonacci sequence based exponentiation by squaring:

```
import numpy as np

def fibonacci(n):
    #check corner case
    if n < 0:
        raise Exception('negative input')
    #define the matrix
    matrix = np.matrix([[1, 1],[1, 0]])
    #call the helper function
    return helper(matrix,n)[0,1]

def helper(matrix,n):
    #check corner case
    if n < 0:
        raise Exception('negative input')
    #define the matrix
    result = np.matrix([[1, 0], [0, 1]])
    #loop until n = 0
    while(n!=0):
        #update result matrix
        if(n%2!=0):
            result = np.matmul(result, matrix)
        #divide n by 2 each iteration
        n = n//2
        #update the matrix each iteration
        matrix = np.matmul(matrix, matrix)

    return result
```

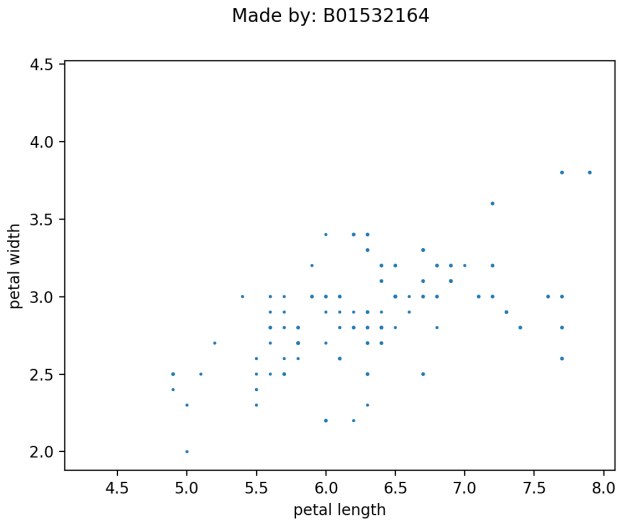
Explaination:
The algorithm is based on the exponentiation by squaring method, and it's an iteration version. Alternatively, a recursion version can be applied.
The fibonacci function just verify the input value and define the Edsger Dijkstra matrix. Then it calls the helper function.
The helper function takes in the Edsger Dijkstra matrix and the n. It implements the exponentiation by squaring method. In each iteration, n is divided by 2, and Edsger Dijkstra matrix is multiplied by itself. If n is odd, the result matrix will be updated.

Time complexity:
In each iteration, n is divided by 2. The matrix multiplication has linear additions and multiplications of size of 2. Therefore, it's an $O(logn)$ algorithm.

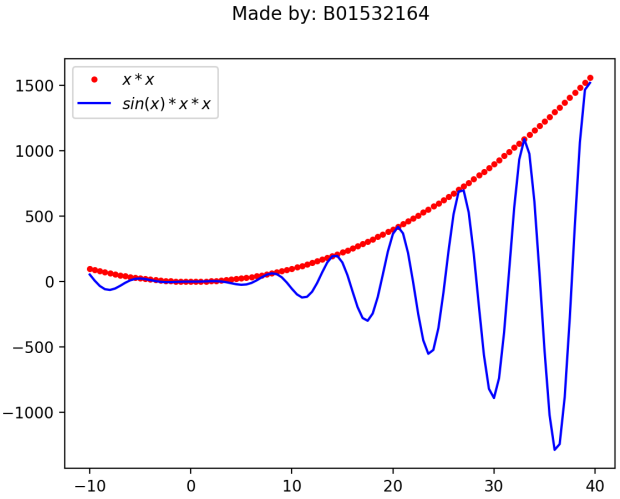
Problem4:

Part1: Matplotlib

1 The plot of iris data:



2 The plot of series data:



Part2: Numpy

1 1D array containing the values 2 through 6:

```
np.arrange(2,7)
```

2 A 4x4 matrix where all values are 1:

```
np.ones((4,4))
```

3 6x6 identity matrix:

```
np.identiy(6) or np.eyes(6)
```

4 Sum the values of each column of matrix A:

```
A.sum(axis = 0)
```

5 $C = A^T B$

```
np.matmul(A.transpose(),B)
```

6 4x2 matrix where all values in the first column are 0's and all the values in the second column are 1's.

```
A = np.zeros((4,1))
B = np.ones((4,1))
np.hstack((A,B))
```

7 All values > 3.0 are set to 1, and the rest to 0.

```
B = (A >3.0).astype(int)
```