Code tests for module

In the module we can access a method that gives you the original eigenvalues for us to check the precision of our method using sample matrices who are symmetric.

For that, we can obtain the eigenvalues obtained through the algorithm, substract them from the eigenvalue vector, apply absolute value, and plot the resulting values for decreasing values of ϵ. What we look for is the presence of near 0 values for this difference.

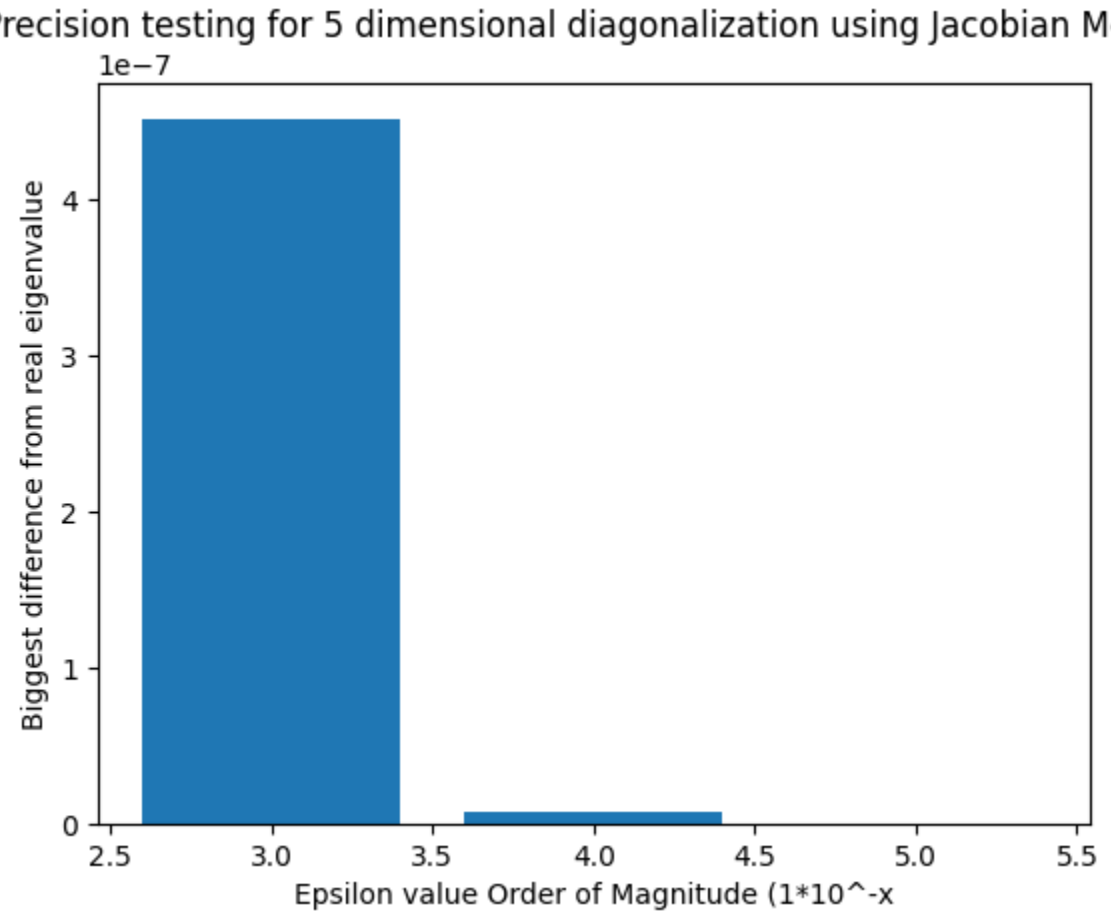
```
import proj_2_module as proj2
```

Case for $n = 5$

```
import numpy as np
import matplotlib.pyplot as plt
def pruebas(n,p,q):
  cs = proj2.cons(n)
  Q = np.dot(proj2.Jay(n,p,q,np.pi/5), np.dot(cs[0],proj2.Jay(n,p,q,np.pi/5).T))
  P = np.dot(proj2.Jay(n,p-1,q+1,np.pi/6.5), np.dot(Q,proj2.Jay(n,p-1,q+1,np.pi/6.5).T))
  A = np.dot(proj2.Jay(n,p-1,q+1,np.pi/6), np.dot(P,proj2.Jay(n,p-1,q+1,np.pi/6).T))
  return A, cs

A, cs = pruebas(5,1,3)
x = [3,4,5]
y = [np.sort(proj2.complex_eigen(A, 1/(10**i))[0]) for i in x]



aa = []


for i in range(len(y)):
```

```
  aa.append(np.abs(np.max(y[1]-np.sort(cs[1]))))
plt.bar(x,aa)
plt.ylabel("Biggest difference from real eigenvalue")
plt.xlabel("Epsilon value Order of Magnitude (1*10^-x")
n = 5
plt.title(f"Precision testing for {n} dimensional diagonalization using Jacobian Method")
```

```
Text(0.5, 1.0, 'Precision testing for 5 dimensional diagonalization using Jacobian Method')
```

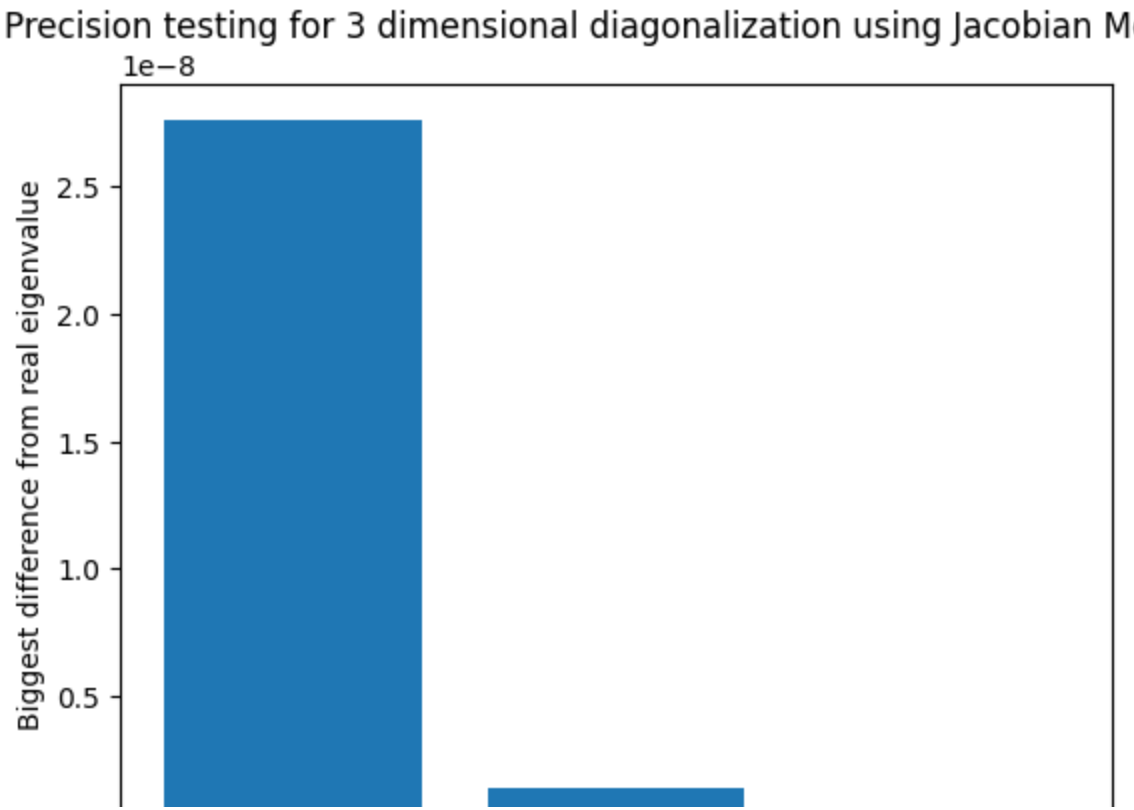


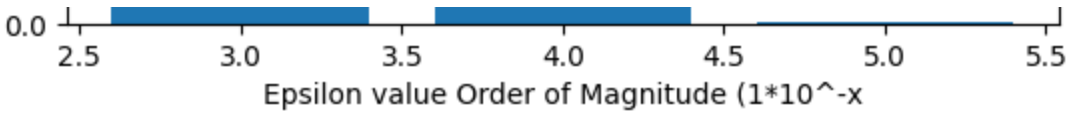
Tests show that it'd be best to check for $\epsilon \approx 10^{-3}$ up to $\epsilon \approx 10^{-5}$, so we'll present the tests for $n \in \{3, \ldots, 30\}$

n = 3

```
A, cs = pruebas(3,1,1)

y = [np.sort(proj2.complex_eigen(A, 1/(10**i))[0]) for i in x]


aa = []
for i in range(len(y)):
  aa.append(np.abs(np.max(y[i]-np.sort(cs[1]))))
plt.bar(x,aa)
plt.ylabel("Biggest difference from real eigenvalue")
plt.xlabel("Epsilon value Order of Magnitude (1*10^-x")
n = 3
plt.title(f"Precision testing for {n} dimensional diagonalization using Jacobian Method")
```

```
Text(0.5, 1.0, 'Precision testing for 3 dimensional diagonalization using Jacobian Method')
```

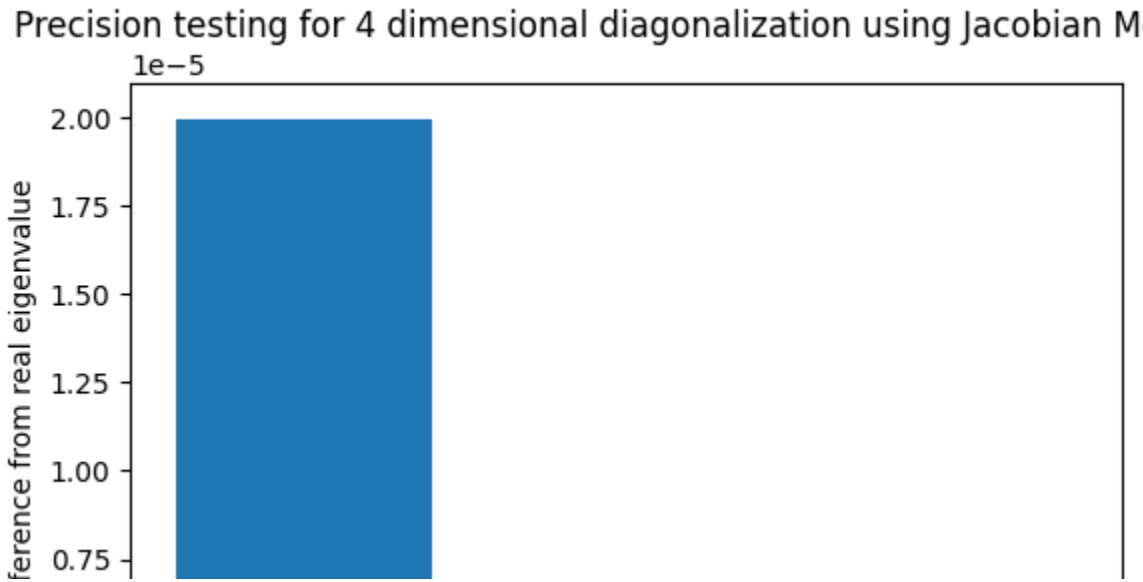


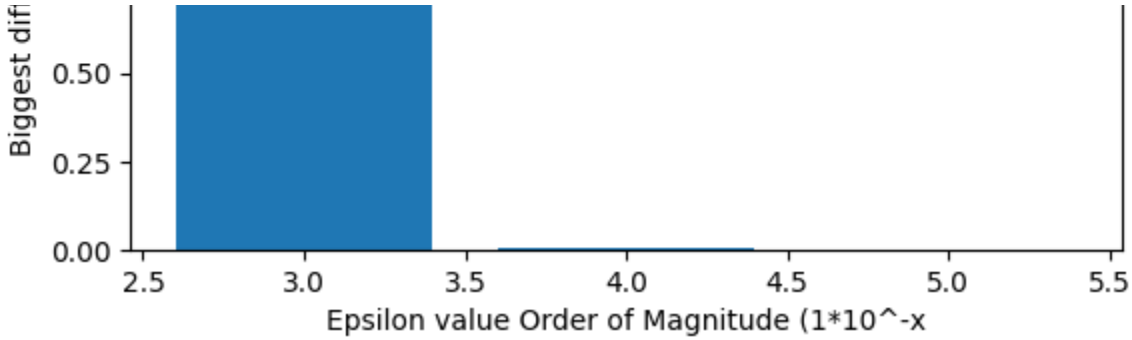


n = 4

```
A, cs = pruebas(4,1,2)

y = [np.sort(proj2.complex_eigen(A, 1/(10**i))[0]) for i in x]


aa = []
for i in range(len(y)):
  aa.append(np.abs(np.max(y[i]-np.sort(cs[1]))))
plt.bar(x,aa)
plt.ylabel("Biggest difference from real eigenvalue")
plt.xlabel("Epsilon value Order of Magnitude (1*10^-x")
n = 4
plt.title(f"Precision testing for {n} dimensional diagonalization using Jacobian Method")
```

```
Text(0.5, 1.0, 'Precision testing for 4 dimensional diagonalization using Jacobian Method')
```

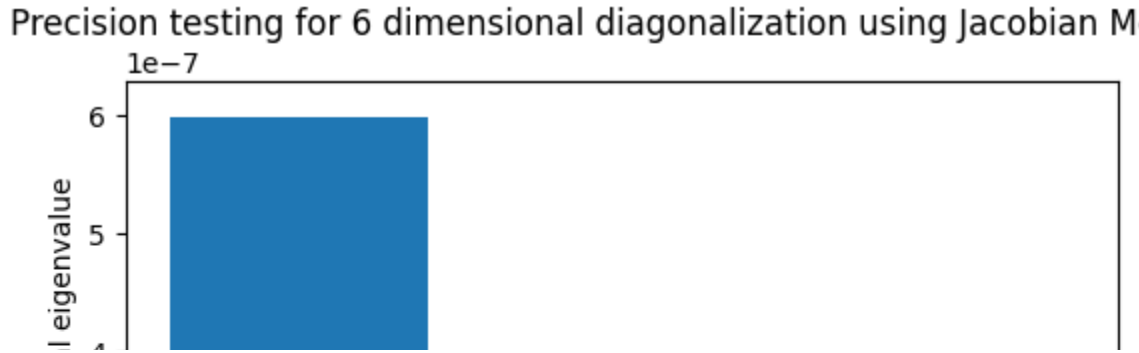


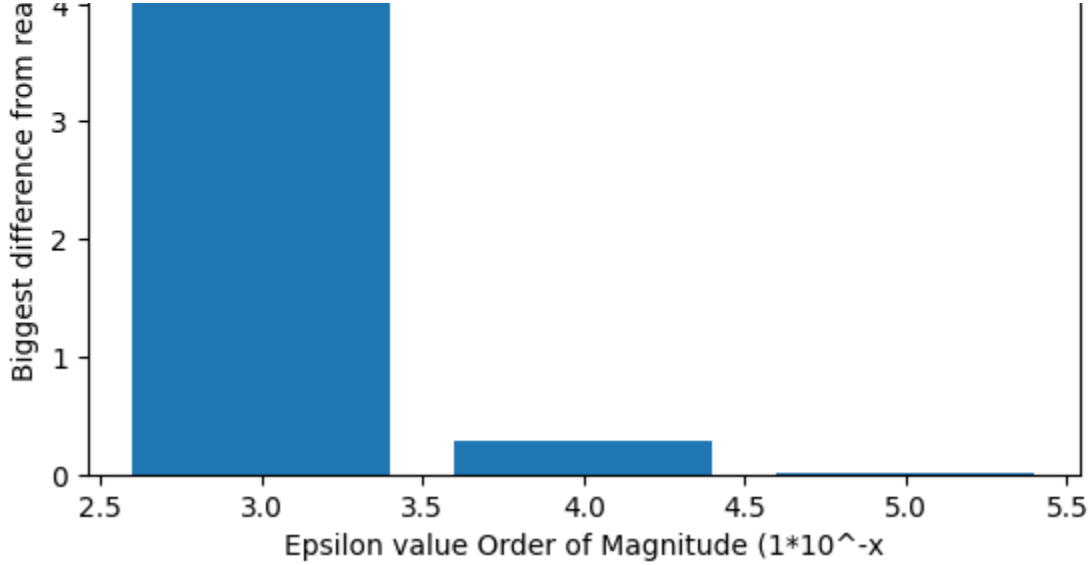


n = 6

```
A, cs = pruebas(6,2,4)

y = [np.sort(proj2.complex_eigen(A, 1/(10**i))[0]) for i in x]


aa = []
for i in range(len(y)):
  aa.append(np.abs(np.max(y[i]-np.sort(cs[1]))))
plt.bar(x,aa)
plt.ylabel("Biggest difference from real eigenvalue")
plt.xlabel("Epsilon value Order of Magnitude (1*10^-x")
n = 6
plt.title(f"Precision testing for {n} dimensional diagonalization using Jacobian Method")
```

```
Text(0.5, 1.0, 'Precision testing for 6 dimensional diagonalization using Jacobian Method')
```

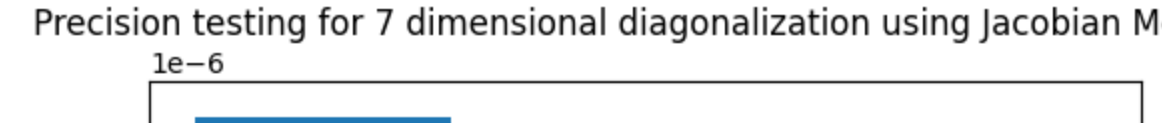


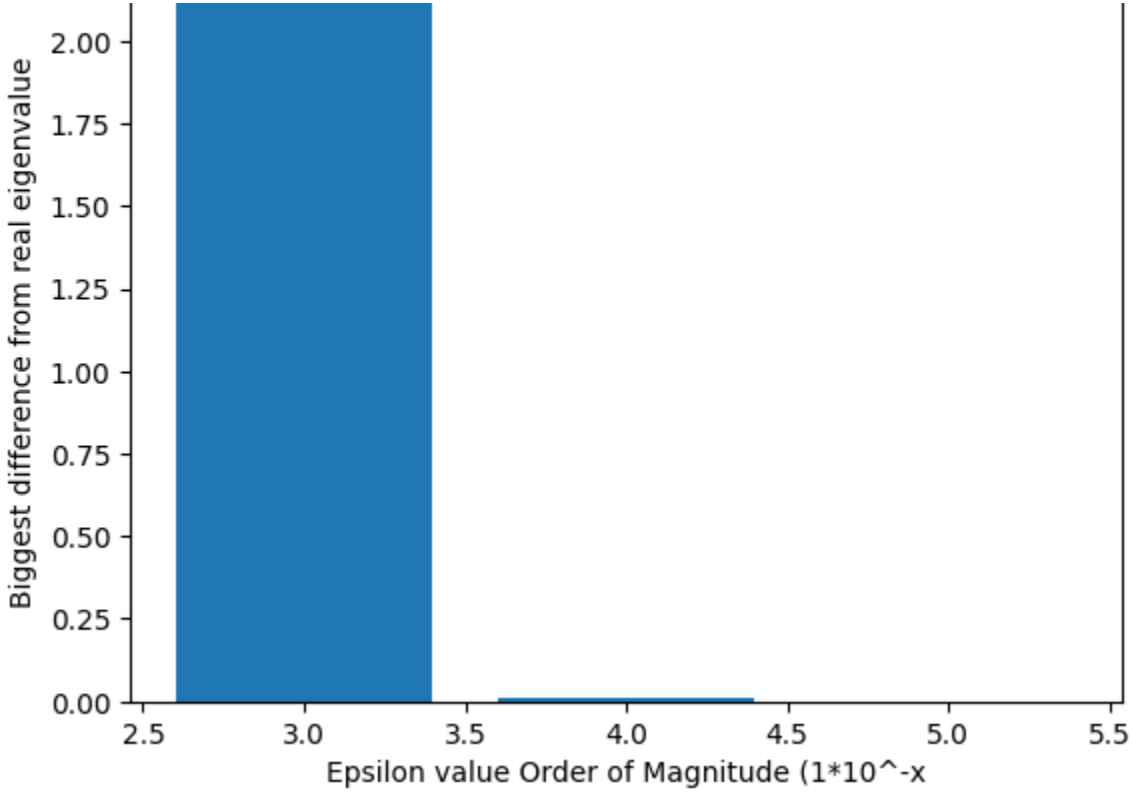


n = 7

```
A, cs = pruebas(7,3,5)

y = [np.sort(proj2.complex_eigen(A, 1/(10**i))[0]) for i in x]


aa = []
for i in range(len(y)):
  aa.append(np.abs(np.max(y[i]-np.sort(cs[1]))))
plt.bar(x,aa)
plt.ylabel("Biggest difference from real eigenvalue")
plt.xlabel("Epsilon value Order of Magnitude (1*10^-x")
n = 7
plt.title(f"Precision testing for {n} dimensional diagonalization using Jacobian Method")
```

```
Text(0.5, 1.0, 'Precision testing for 7 dimensional diagonalization using Jacobian Method')
```



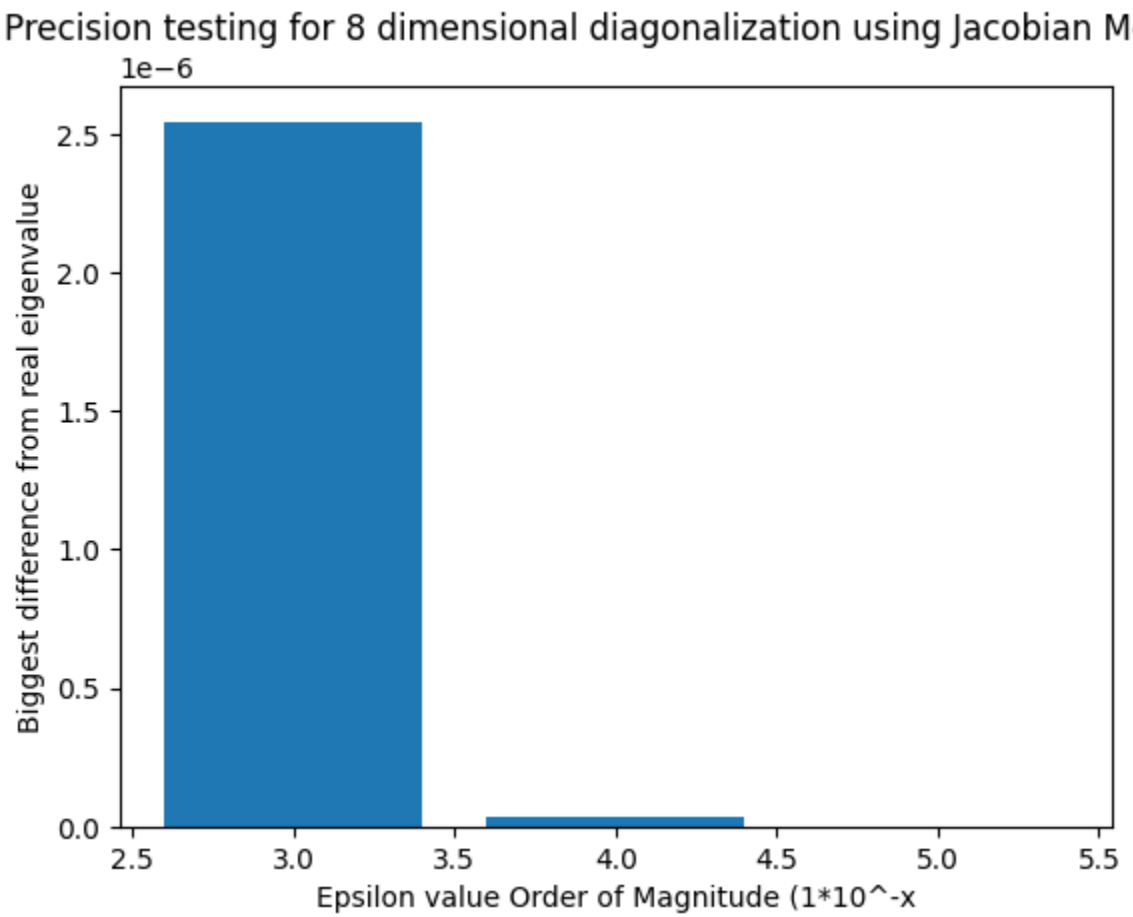


n = 8

```
A, cs = pruebas(8,2,6)

y = [np.sort(proj2.complex_eigen(A, 1/(10**i))[0]) for i in x]


aa = []
for i in range(len(y)):
  aa.append(np.abs(np.max(y[i]-np.sort(cs[1]))))
plt.bar(x,aa)
plt.ylabel("Biggest difference from real eigenvalue")
plt.xlabel("Epsilon value Order of Magnitude (1*10^-x")
n = 8
plt.title(f"Precision testing for {n} dimensional diagonalization using Jacobian Method")
```

```
plt.title(f'Precision testing for {n} dimensional diagonalization using Jacobian Method ')
```

```
Text(0.5, 1.0, 'Precision testing for 8 dimensional diagonalization using Jacobian Method')
```



n = 9

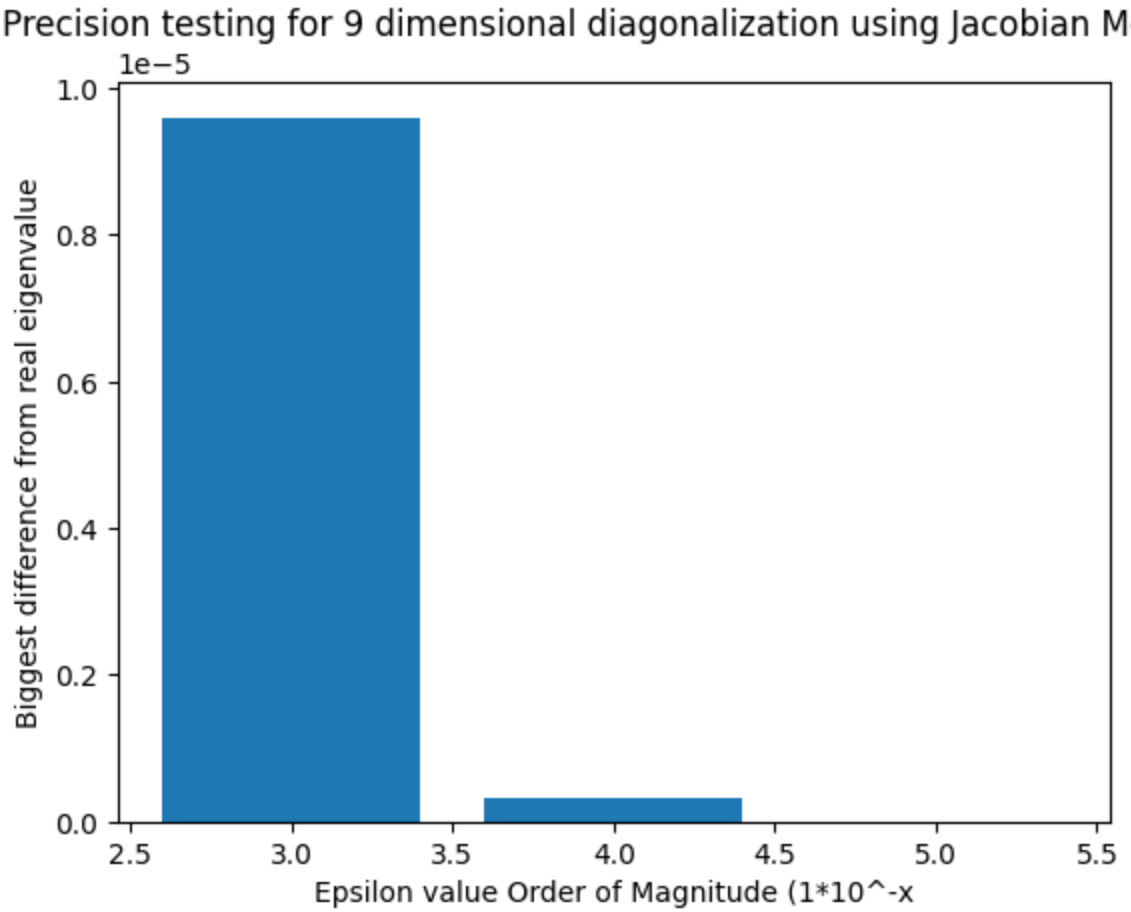
```
A, cs = pruebas(9,2,6)
```

```
y = [np.sort(proj2.complex_eigen(A, 1/(10**i))[0]) for i in x]
```

```
aa = []
for i in range(len(y)):
```

```
for i in range(len(y)):
  aa.append(np.abs(np.max(y[i]-np.sort(cs[1]))))
plt.bar(x,aa)
plt.ylabel("Biggest difference from real eigenvalue")
plt.xlabel("Epsilon value Order of Magnitude (1*10^-x")
n = 9
plt.title(f"Precision testing for {n} dimensional diagonalization using Jacobian Method")
```

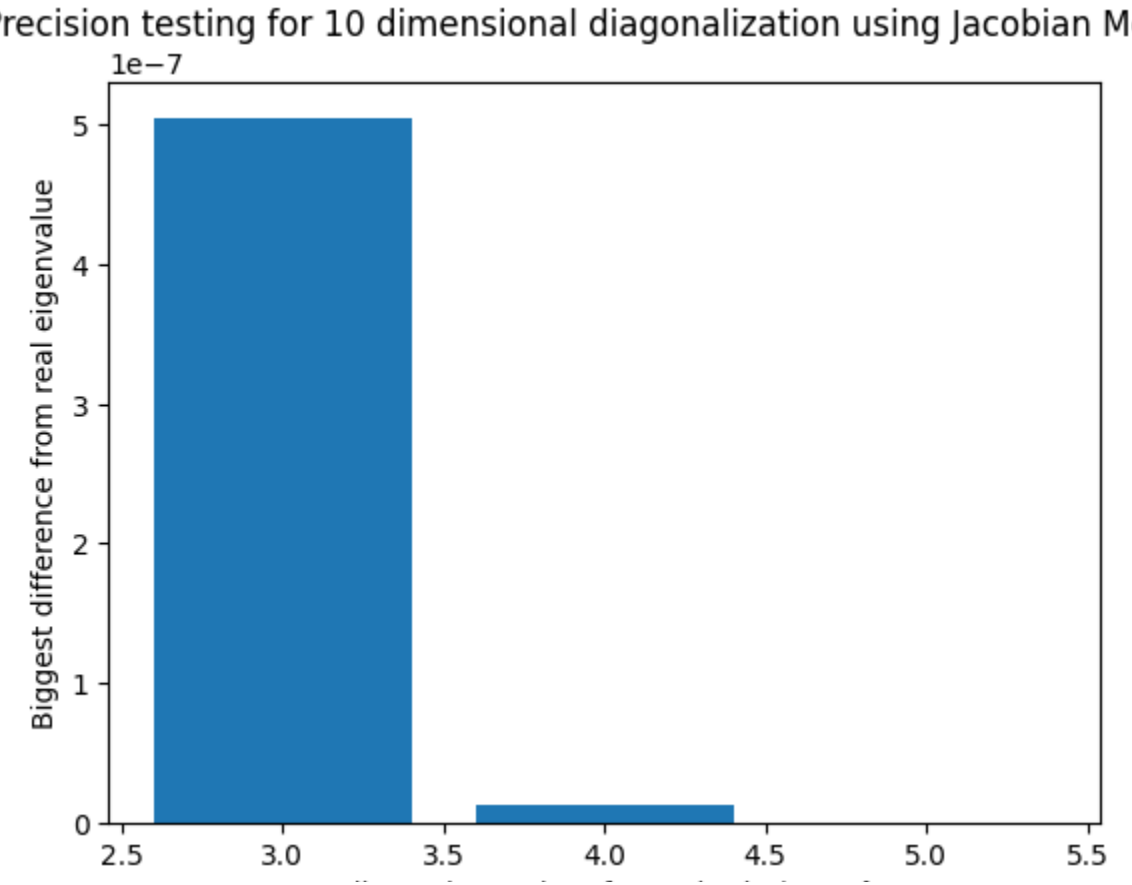
```
Text(0.5, 1.0, 'Precision testing for 9 dimensional diagonalization using Jacobian Method')
```



n = 10

```
A, cs = pruebas(10,2,6)

y = [np.sort(proj2.complex_eigen(A, 1/(10**i))[0]) for i in x]


aa = []
for i in range(len(y)):
  aa.append(np.abs(np.max(y[i]-np.sort(cs[1]))))
plt.bar(x,aa)
plt.ylabel("Biggest difference from real eigenvalue")
plt.xlabel("Epsilon value Order of Magnitude (1*10^-x")
n = 10
plt.title(f"Precision testing for {n} dimensional diagonalization using Jacobian Method")
```

```
Text(0.5, 1.0, 'Precision testing for 10 dimensional diagonalization using Jacobian Method')
```

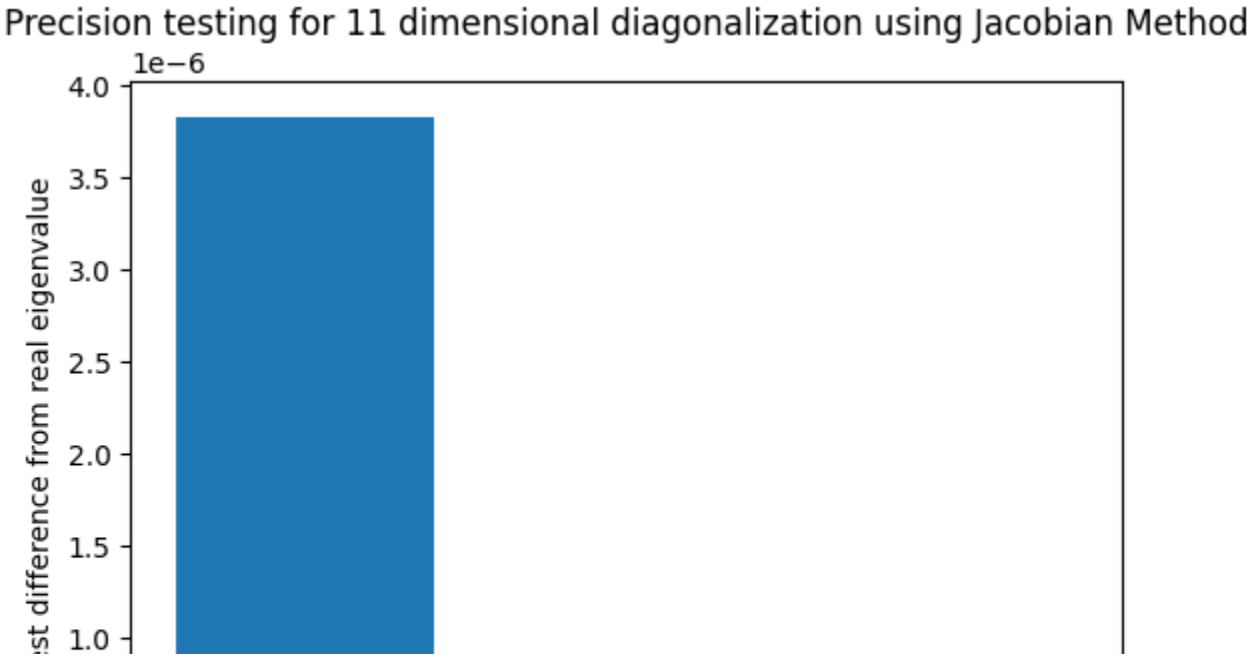


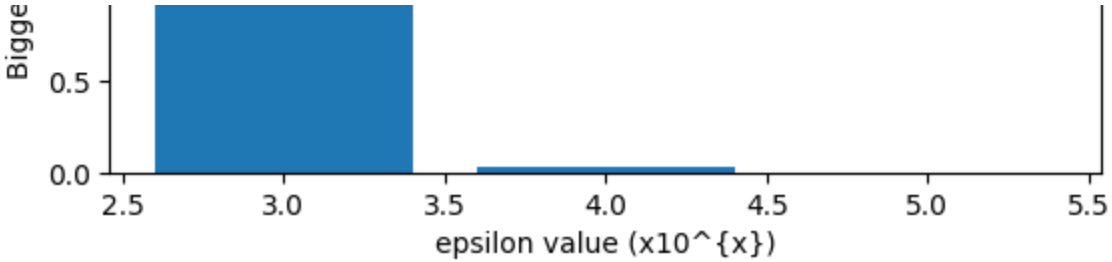
Epsilon value Order of Magnitude (1*10^-x

n = 11

```
A, cs = pruebas(11,2,9)

y = [np.sort(proj2.complex_eigen(A, 1/(10**i))[0]) for i in x]


aa = []
for i in range(len(y)):
  aa.append(np.abs(np.max(y[i]-np.sort(cs[1]))))
plt.bar(x,aa)
plt.ylabel("Biggest difference from real eigenvalue")
plt.xlabel("epsilon value (x10^{x})")
n = 11
plt.title(f"Precision testing for {n} dimensional diagonalization using Jacobian Method")
```

```
Text(0.5, 1.0, 'Precision testing for 11 dimensional diagonalization using Jacobian Method')
```

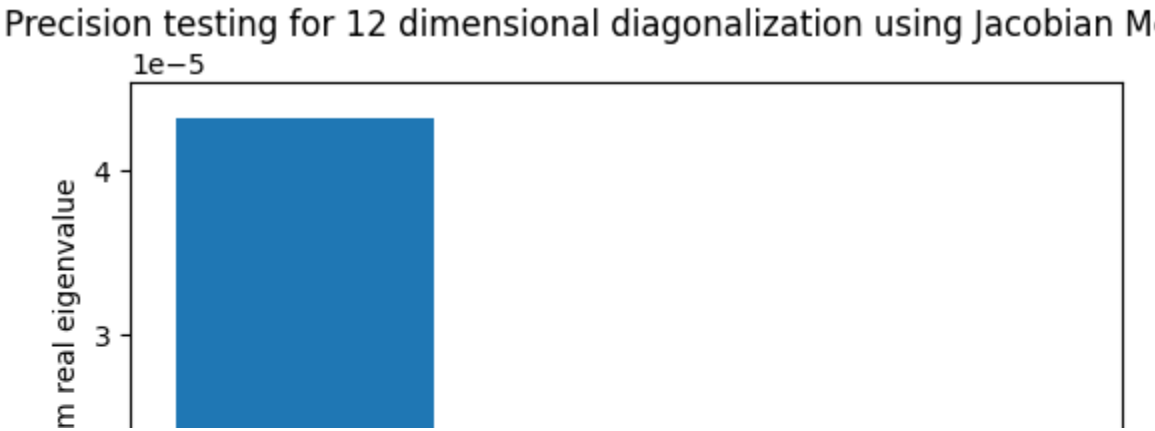


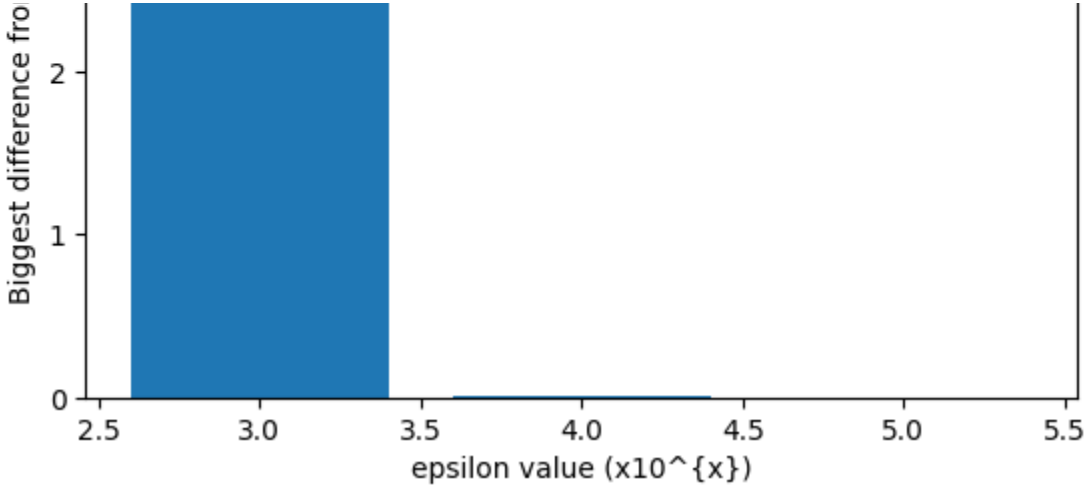


n = 12

```
A, cs = pruebas(12,2,9)

y = [np.sort(proj2.complex_eigen(A, 1/(10**i))[0]) for i in x]


aa = []
for i in range(len(y)):
  aa.append(np.abs(np.max(y[i]-np.sort(cs[1]))))
plt.bar(x,aa)
plt.ylabel("Biggest difference from real eigenvalue")
plt.xlabel("epsilon value (x10^{x})")
n = 12
plt.title(f"Precision testing for {n} dimensional diagonalization using Jacobian Method")
```

```
Text(0.5, 1.0, 'Precision testing for 12 dimensional diagonalization using Jacobian Method')
```

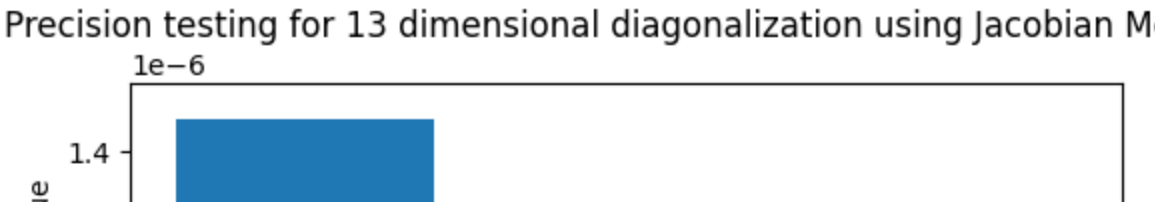


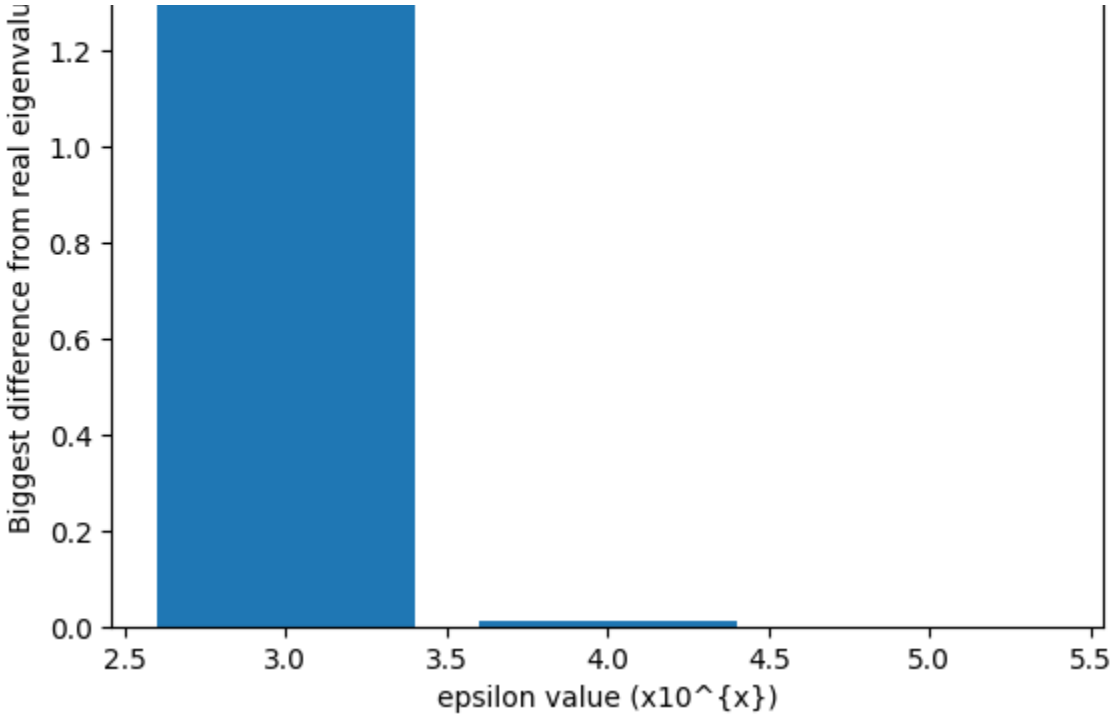


n = 13

```
A, cs = pruebas(13,4,10)

y = [np.sort(proj2.complex_eigen(A, 1/(10**i))[0]) for i in x]


aa = []
for i in range(len(y)):
  aa.append(np.abs(np.max(y[i]-np.sort(cs[1]))))
plt.bar(x,aa)
plt.ylabel("Biggest difference from real eigenvalue")
plt.xlabel("epsilon value (x10^{x})")
n = 13
plt.title(f"Precision testing for {n} dimensional diagonalization using Jacobian Method")
```

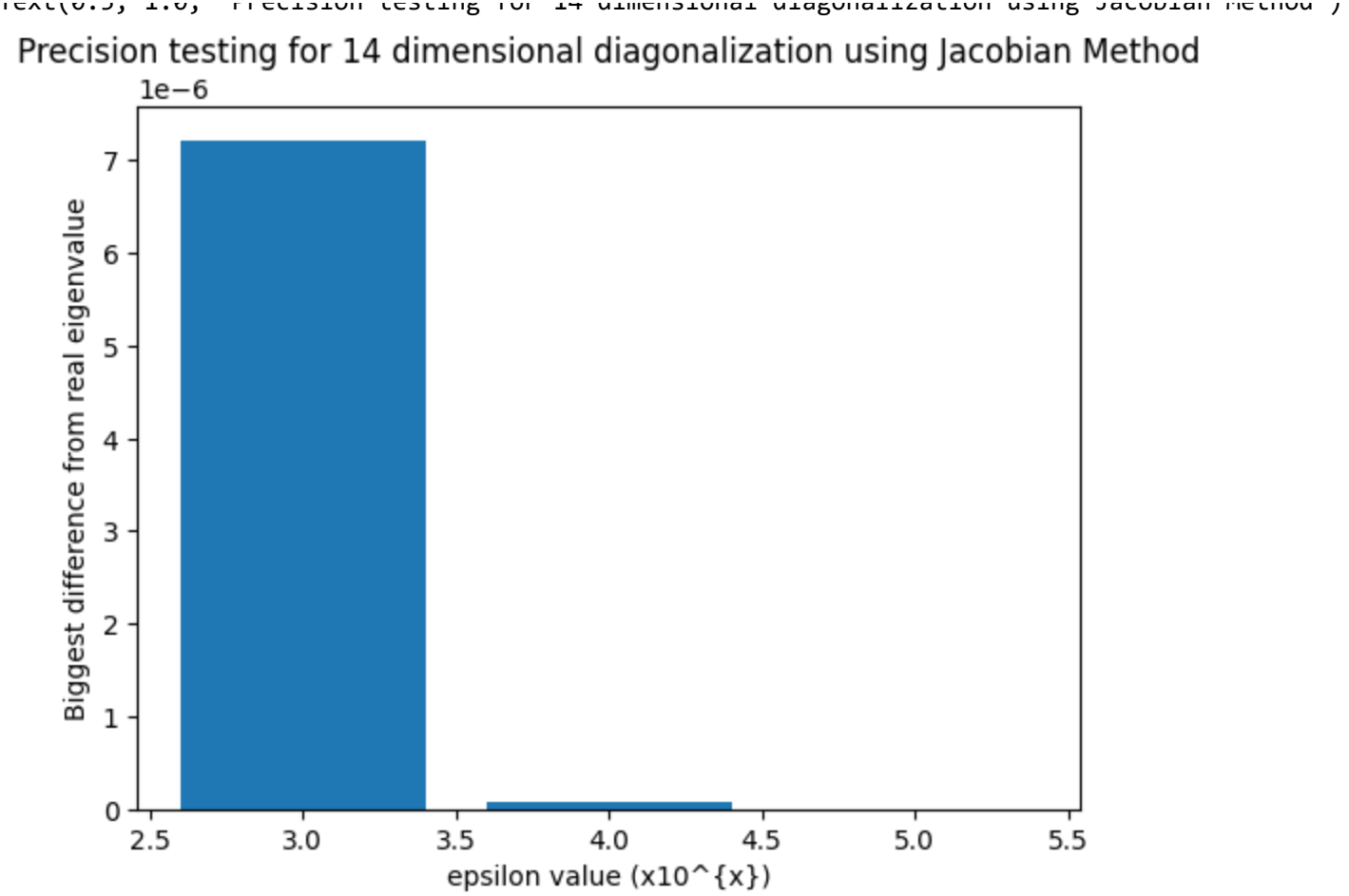
```
Text(0.5, 1.0, 'Precision testing for 13 dimensional diagonalization using Jacobian Method')
```





n = 14

```
A, cs = pruebas(14,5,12)

y = [np.sort(proj2.complex_eigen(A, 1/(10**i))[0]) for i in x]


aa = []
for i in range(len(y)):
  aa.append(np.abs(np.max(y[i]-np.sort(cs[1]))))
plt.bar(x,aa)
plt.ylabel("Biggest difference from real eigenvalue")
plt.xlabel("epsilon value (x10^{x})")
n = 14
plt.title(f"Precision testing for {n} dimensional diagonalization using Jacobian Method")

     Text(0.5, 1.0, 'Precision testing for 14 dimensional diagonalization using Jacobian Method')
```

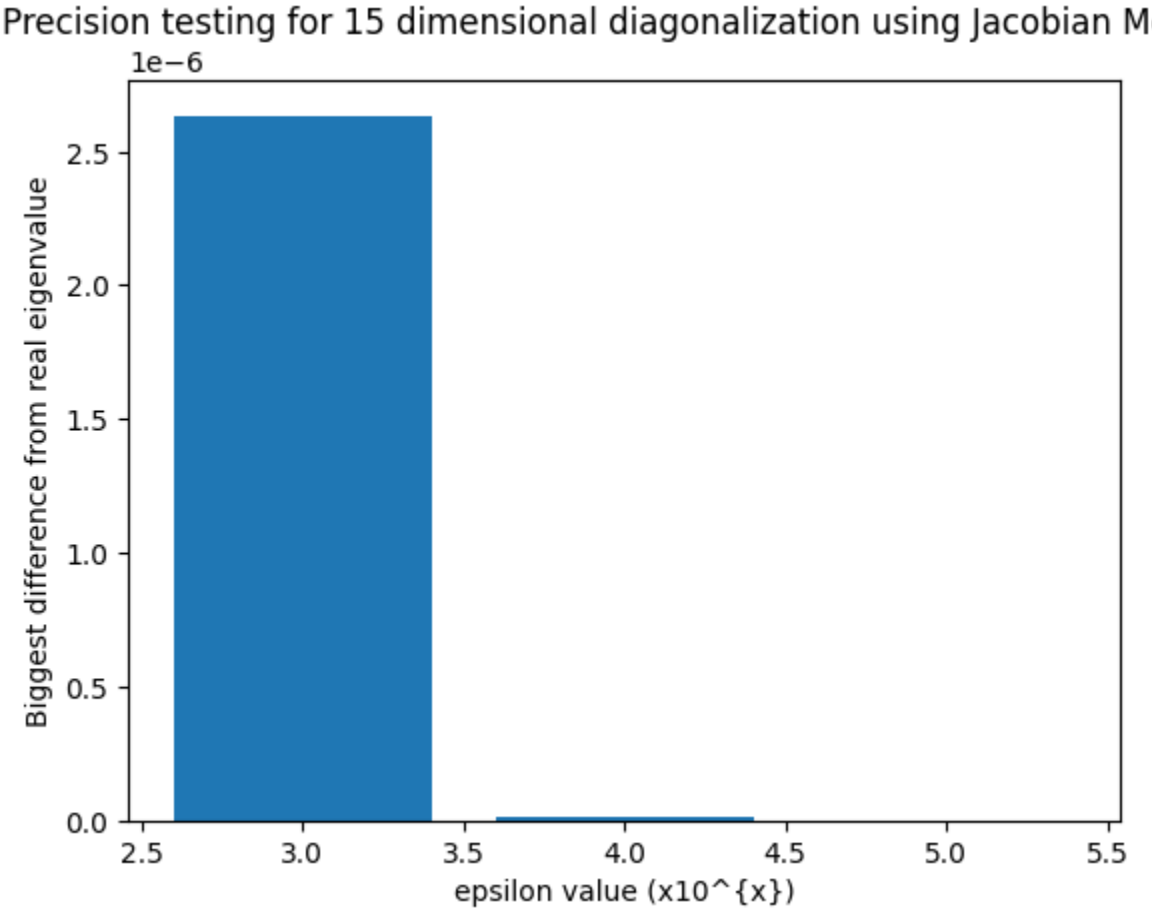


n = 15

```
A, cs = pruebas(15,7,13)

y = [np.sort(proj2.complex_eigen(A, 1/(10**i))[0]) for i in x]


aa = []
for i in range(len(y)):
  aa.append(np.abs(np.max(y[i]-np.sort(cs[1]))))
plt.bar(x,aa)
```

```
plt.bar(x,aa)
plt.ylabel("Biggest difference from real eigenvalue")
plt.xlabel("epsilon value (x10^{x})")
n = 15
plt.title(f"Precision testing for {n} dimensional diagonalization using Jacobian Method")
```

```
Text(0.5, 1.0, 'Precision testing for 15 dimensional diagonalization using Jacobian Method')
```

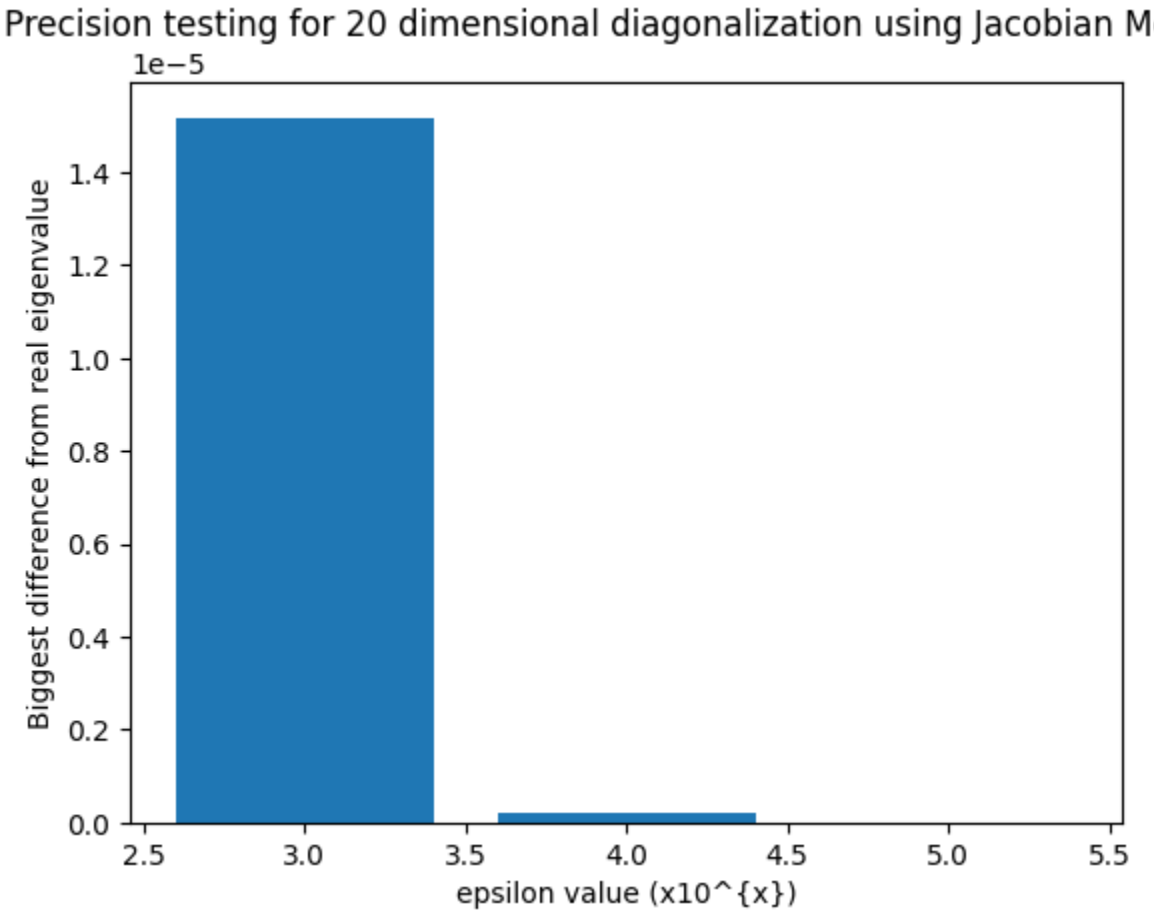


n = 20

```
A, cs = pruebas(20,9,16)
```

```
y = [np.sort(proj2.complex_eigen(A, 1/(10**i))[0]) for i in x]


aa = []
for i in range(len(y)):
  aa.append(np.abs(np.max(y[i]-np.sort(cs[1]))))
plt.bar(x,aa)
plt.ylabel("Biggest difference from real eigenvalue")
plt.xlabel("epsilon value (x10^{x})")
n = 20
plt.title(f"Precision testing for {n} dimensional diagonalization using Jacobian Method")
```

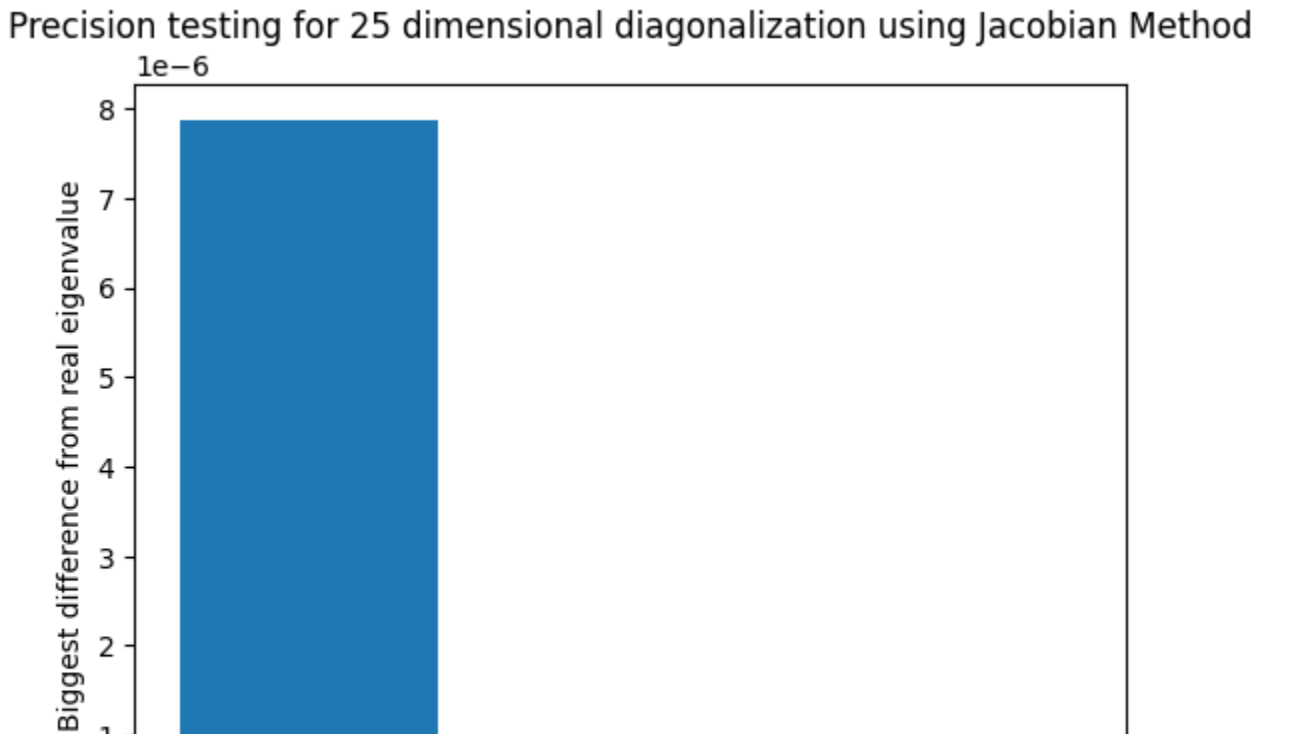
```
Text(0.5, 1.0, 'Precision testing for 20 dimensional diagonalization using Jacobian Method')
```

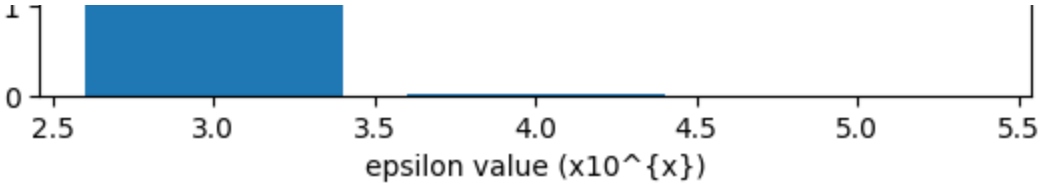


n = 25

```
A, cs = pruebas(25,15,18)

y = [np.sort(proj2.complex_eigen(A, 1/(10**i))[0]) for i in x]


aa = []
for i in range(len(y)):
  aa.append(np.abs(np.max(y[i]-np.sort(cs[1]))))
plt.bar(x,aa)
plt.ylabel("Biggest difference from real eigenvalue")
plt.xlabel("epsilon value (x10^{x})")
n = 25
plt.title(f"Precision testing for {n} dimensional diagonalization using Jacobian Method")
```

```
Text(0.5, 1.0, 'Precision testing for 25 dimensional diagonalization using Jacobian Method')
```

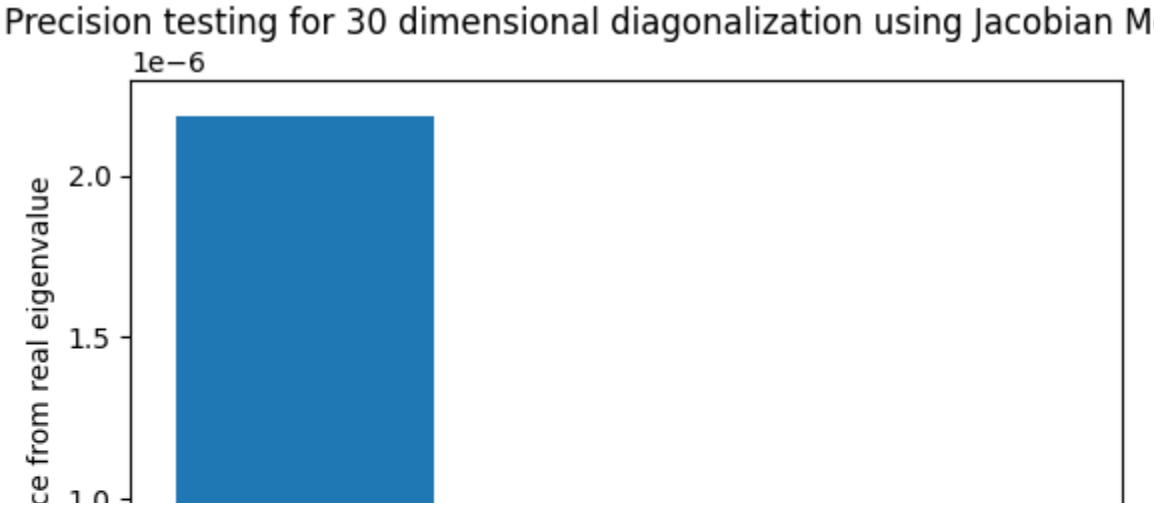


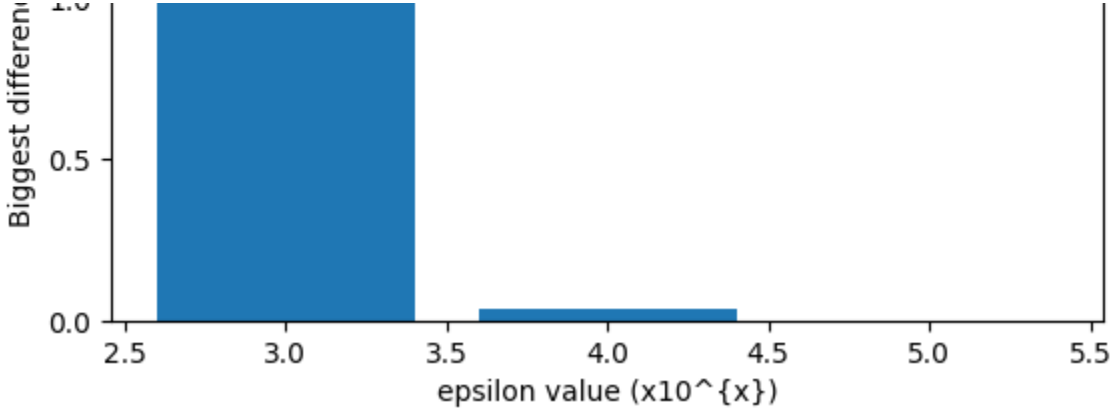


n = 30

```
A, cs = pruebas(30,12,25)

y = [np.sort(proj2.complex_eigen(A, 1/(10**i))[0]) for i in x]


aa = []
for i in range(len(y)):
  aa.append(np.abs(np.max(y[i]-np.sort(cs[1]))))
plt.bar(x,aa)
plt.ylabel("Biggest difference from real eigenvalue")
plt.xlabel("epsilon value (x10^{x})")
n = 30
plt.title(f"Precision testing for {n} dimensional diagonalization using Jacobian Method")
```

```
Text(0.5, 1.0, 'Precision testing for 30 dimensional diagonalization using Jacobian Method')
```





This shows that the precision attainable using $\epsilon \approx 10^{-5}$ seems to be consistently good throughout different dimensions, if compared with $\epsilon \approx 10^{-2}, 10^{-1}$ for the same dimension.