Expression d'une assertion impliquant la conjecture de Goldbach dans $\mathbb{C}$

L'assertion mathématique ci-dessous exprime le fait que n, un nombre pair supérieur ou égal à 6, possède un décomposant de Goldbach x (i.e. un nombre premier tel que $n - x$ est premier également).

C'est une assertion plus forte que la conjecture de Goldbach (elle implique la conjecture) dans la mesure où le nombre premier x que définit l'assertion est compris entre $\sqrt{n}$ et $n/2$. Les nombres premiers inférieurs à $\sqrt{n}$ dont le complémentaire est premier ne vérifient pas l'assertion (a).

L'assertion (a) exprime le fait que x est un nombre premier compris entre $\sqrt{n}$ et $n/2$ et l'assertion b exprime le fait que $n - x$ est un nombre premier également.

ε doit être "suffisamment petit". Une valeur de 10^{-10} convient pour détecter les décomposants de Goldbach des nombres 6 à 100 par exemple.

$$\boxed{\begin{array}{l} \forall n \geq 6, \exists x \geq 3 \,/ \\ \\ \left| \displaystyle\prod_{\substack{2 \leq x \leq \frac{n}{2} \\ 2 \leq p \leq \sqrt{n}}} \left(1 - \exp\left(\frac{2i\pi x}{p}\right)\right) \right| \qquad > \varepsilon \qquad \text{(a)} \\ \\ \qquad \text{et} \\ \\ \left| \displaystyle\prod_{\substack{2 \leq x \leq \frac{n}{2} \\ 2 \leq p \leq \sqrt{n}}} \left(1 - \exp\left(\frac{2i\pi (n-x)}{p}\right)\right) \right| \qquad > \varepsilon \qquad \text{(b)} \end{array}}$$

On peut réécrire les exponentielles en utilisant la racine p-ième de -1 (notée $\sqrt[p]{-1}$) ainsi :

$$\boxed{\begin{array}{l} \forall n \geq 6, \exists x \geq 3 \,/ \\ \\ \left| \displaystyle\prod_{\substack{2 \leq x \leq \frac{n}{2} \\ 2 \leq p \leq \sqrt{n}}} \left(1 - \left(\sqrt[p]{-1}\right)^{2x}\right) \right| \qquad > \varepsilon \qquad \text{(a)} \\ \\ \qquad \text{et} \\ \\ \left| \displaystyle\prod_{\substack{2 \leq x \leq \frac{n}{2} \\ 2 \leq p \leq \sqrt{n}}} \left(1 - \left(\sqrt[p]{-1}\right)^{2(n-x)}\right) \right| \qquad > \varepsilon \qquad \text{(b)} \end{array}}$$

Le programme qui trouve les décompositions de Goldbach des nombres pairs successifs de 6 à 100 est très court. On peut le lire ci-dessous.

On fournit après le programme les images qui montrent où se placent les nombres premiers, dont certains fournissent des décompositions de Goldbach, en dehors du cercle-unité. La position d'un nombre relativement à la position du cercle-unité est modifiée à chaque traversée d'un carré parfait.

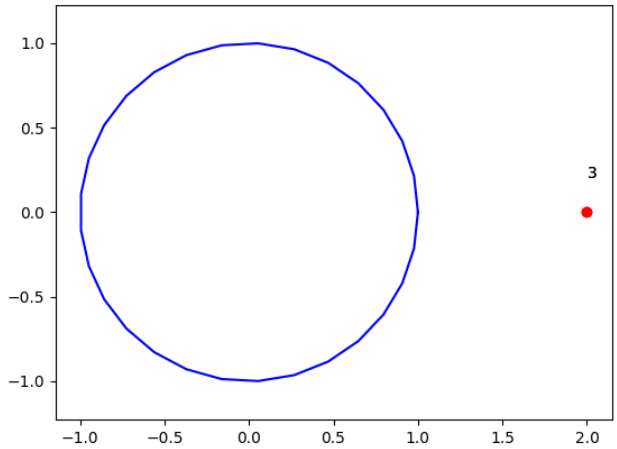
```
from cmath import *
import numpy as np
from numpy import exp, pi, sqrt, linspace, cos, sin
import matplotlib.pyplot as plt

for n in range(6,103,2):
     theta = linspace(0,2*pi,30)
     r = 1
     a = r*cos(theta)
     b = r*sin(theta)
     z = a+1j*b
     plt.plot(a,b, color='blue')
     racine = int(sqrt(n))
     for x in range(2,n//2+1):
          prodbas = 1
          prodhaut = 1
          for p in range(2,racine+1):
               xx = 2*pi*x/p
               letout = exp(2*pi*1j*n/p)
               lun = exp(xx*1j)
               prodbas = prodbas*(1-lun)
               produit = letout/lun
               prodhaut = prodhaut*(1-produit)
          if abs(prodbas) > pow(10,-10) and abs(prodhaut) > pow(10,-10):
               print(x, ' dg de ',n)
               plt.scatter(prodbas.real, prodbas.imag, color='red')
               plt.annotate(x,xy=(prodbas.real, prodbas.imag+0.2))
               plt.scatter(prodhaut.real, prodhaut.imag, color='red')
               plt.annotate(n-x,xy=(prodhaut.real, prodhaut.imag+0.2))
     plt.axis('equal')
     plt.show()
```

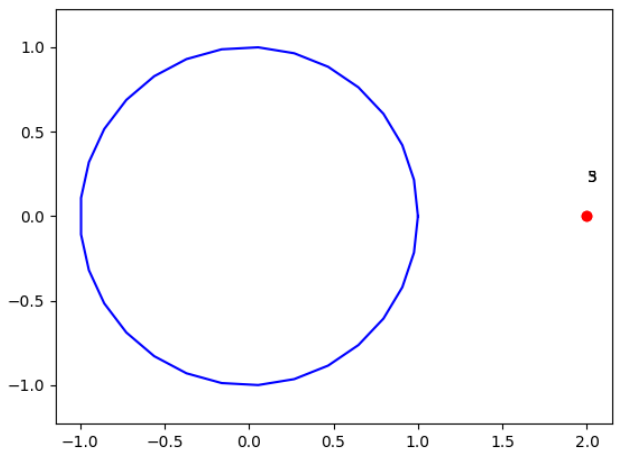
Note pour les programmeuses : Python a des réactions bizarres par rapport aux types des variables : il calcule correctement en remplaçant

$\exp(2i\pi x/p)$	par
$\text{power}(\text{power}(-1, 1/p), 2 * x)$	ou bien encore par
$\text{power}(\text{power}(-1, 1/p), 2 * x)$	ou enfin par
$\text{power}(\exp(\log(-1)/p), 2 * x)$	

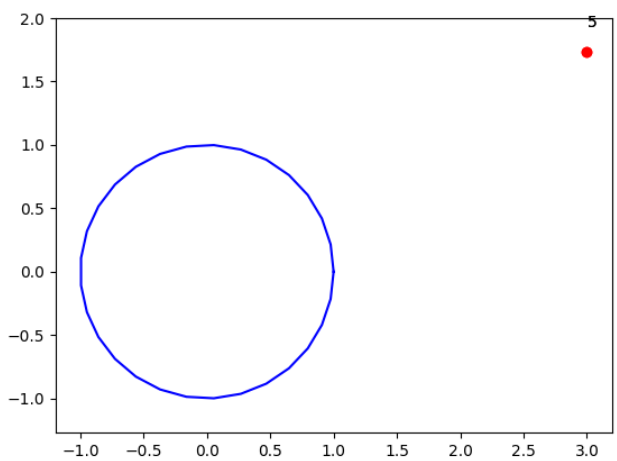
en prenant la fonction *power* dans le module `mpmath` (i.e. cette fonction travaille sur des complexes) mais on ne peut pas intervertir les exposants, la racine p-ième de 1 valant toujours 1 (*rappel* : selon la belle formule d'Euler, $log(-1) = i\pi$), en python, *log* désigne le logarithme népérien.


1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
3

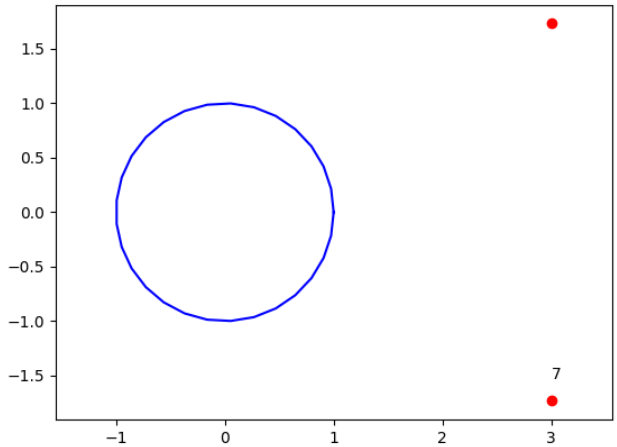
n=6


1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
5

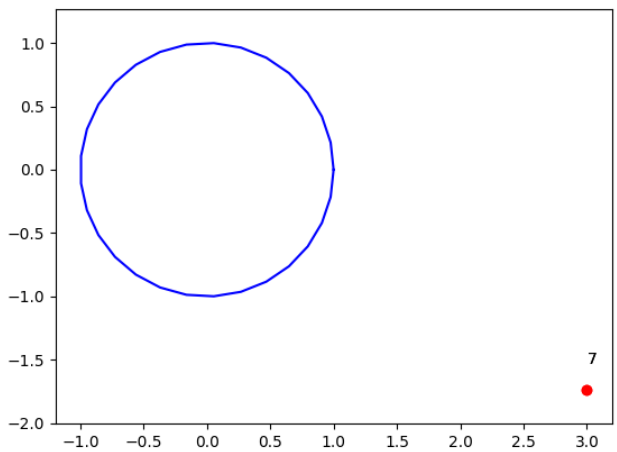
n=8


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
5

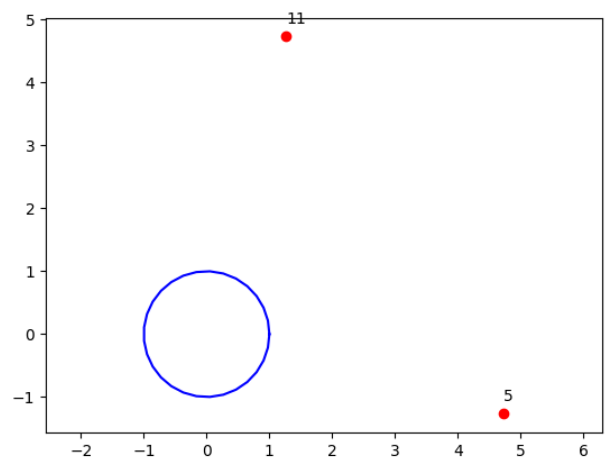
n=10



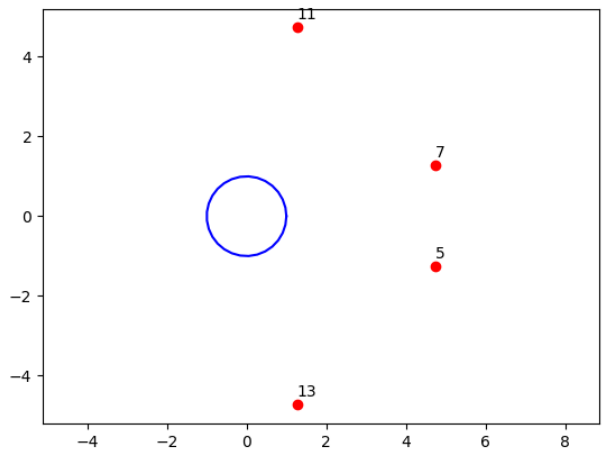
n=12



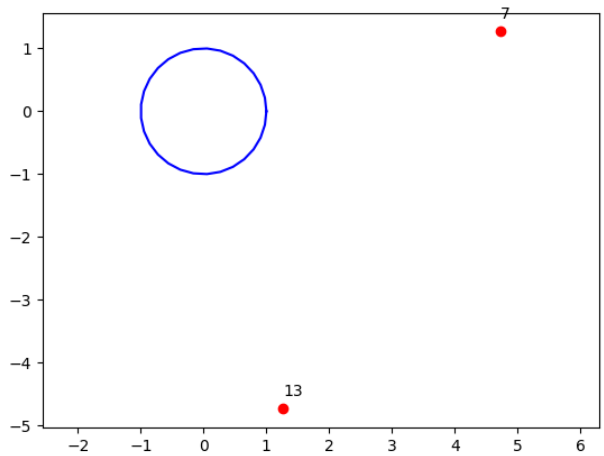
n=14



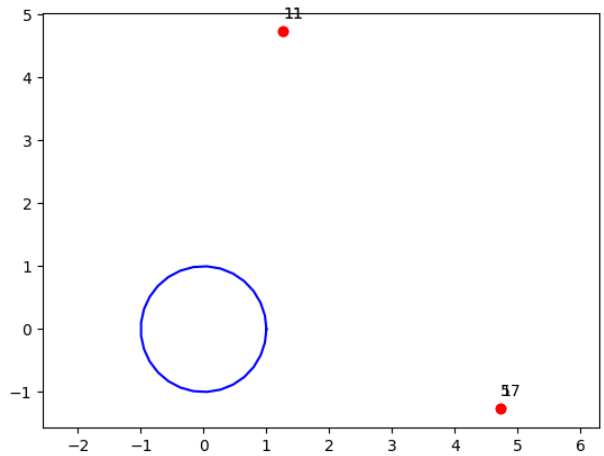
n=16



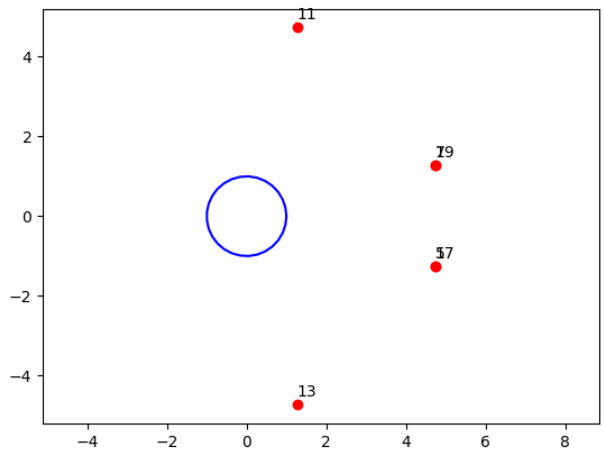
n=18



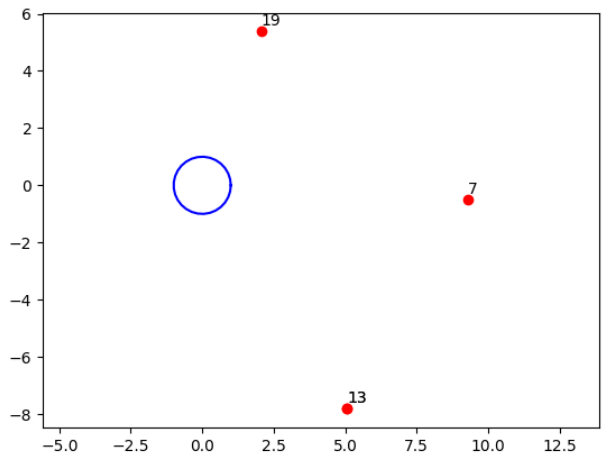
n=20



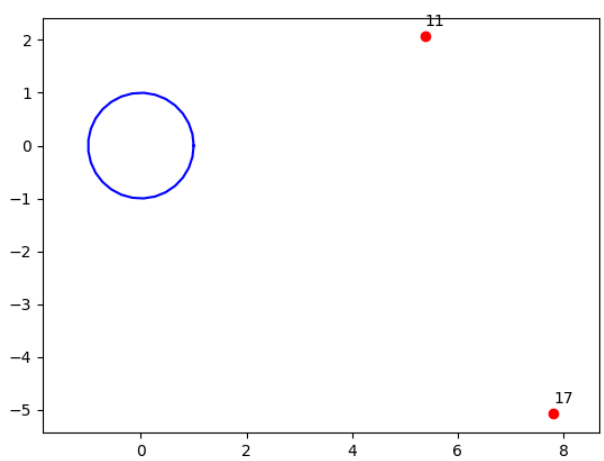
n=22



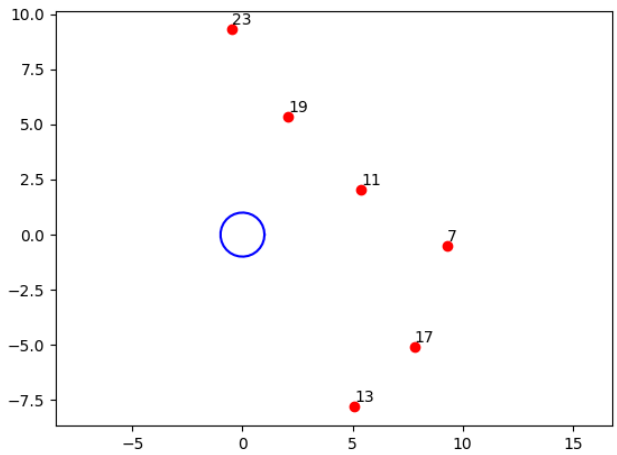
n=24



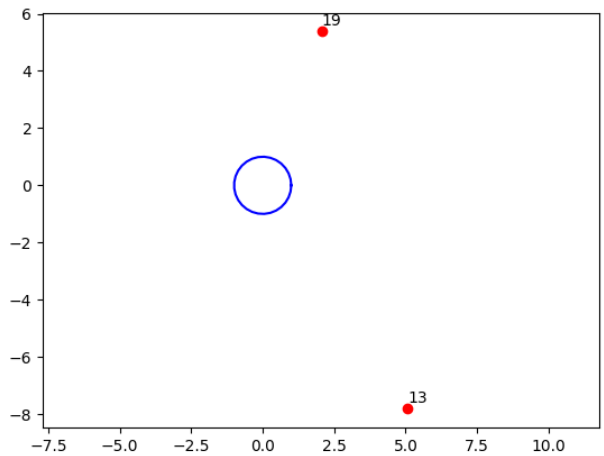
n=26



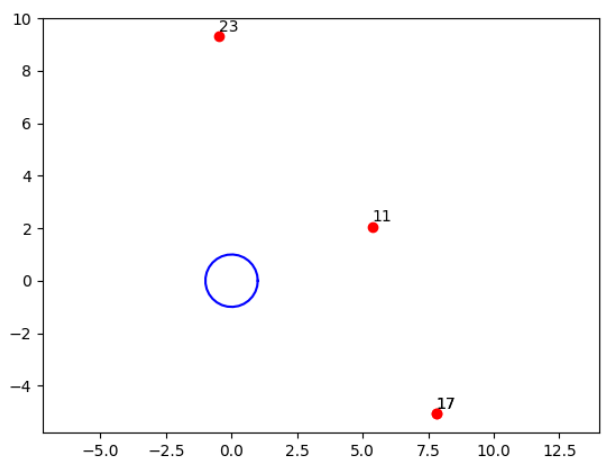
n=28



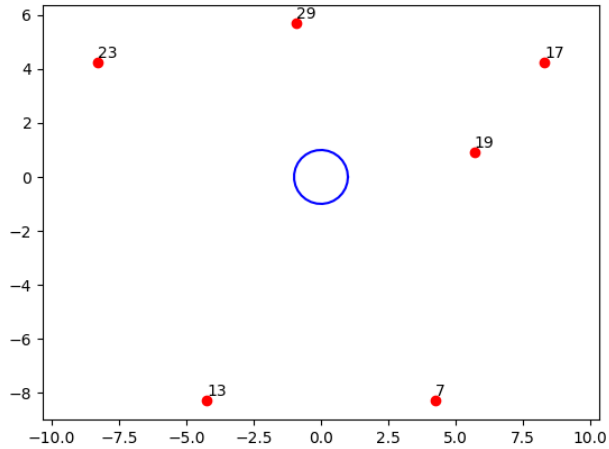
n=30



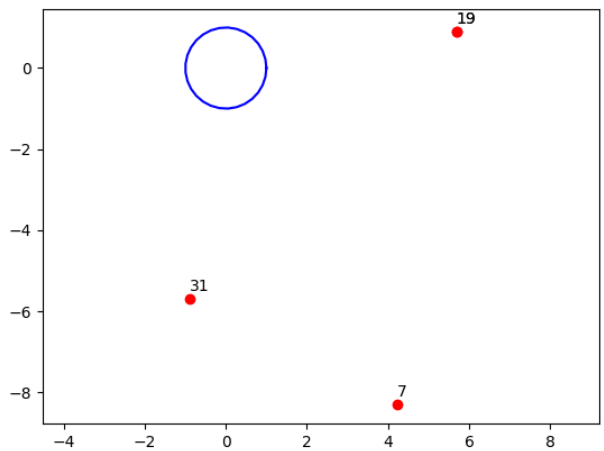
n=32



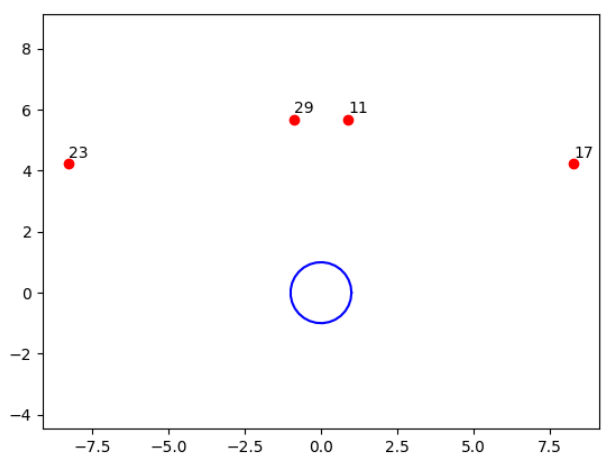
n=34



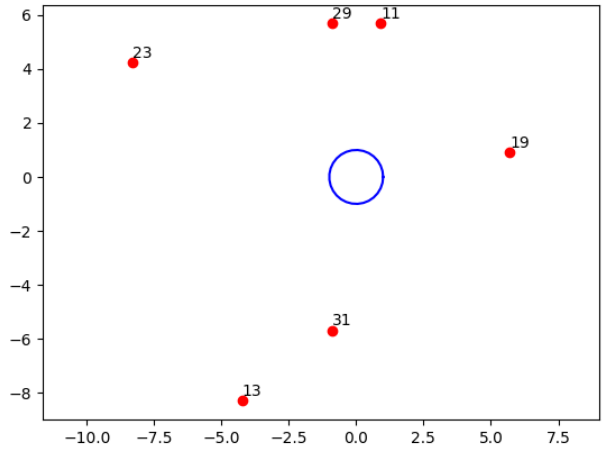
n=36



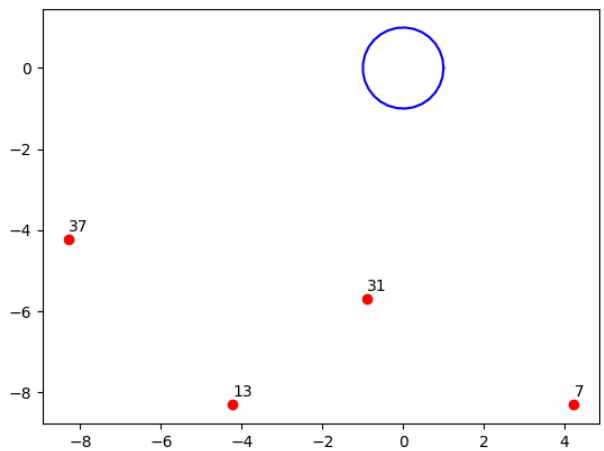
n=38



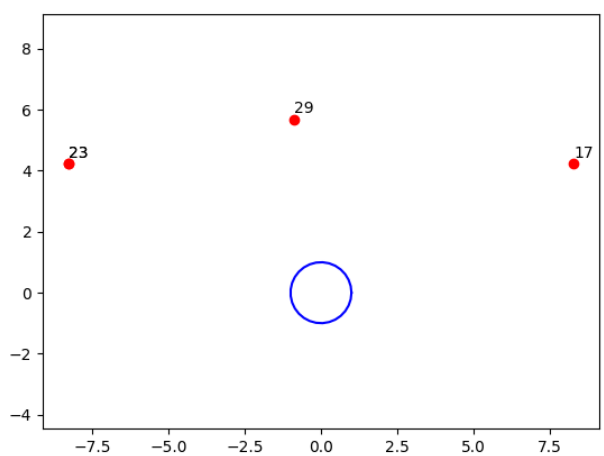
n=40



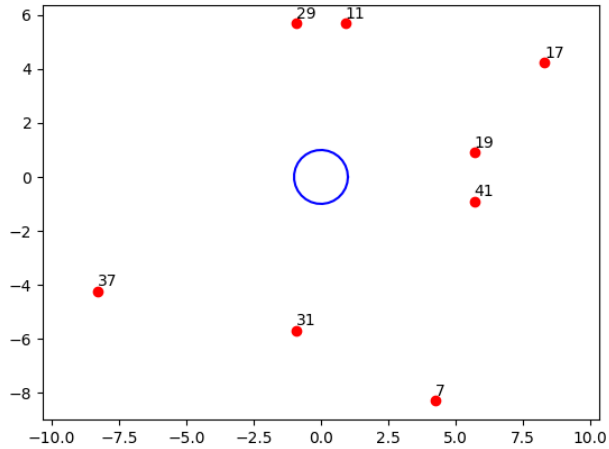
n=42



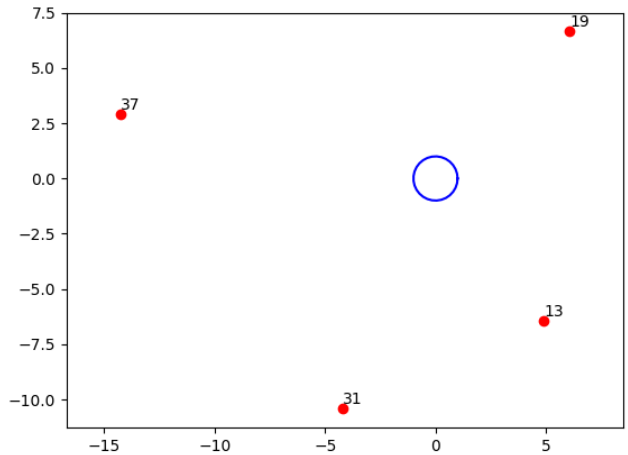
n=44



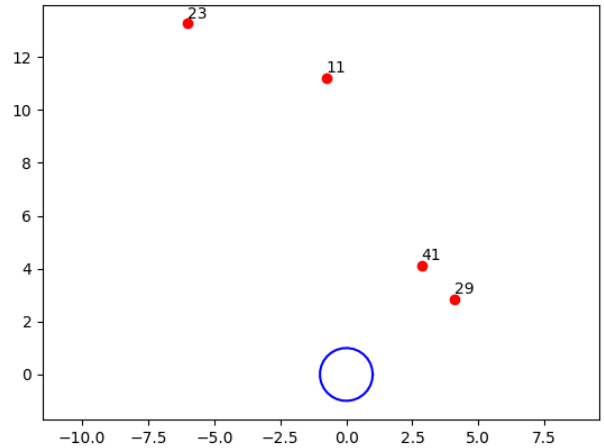
n=46



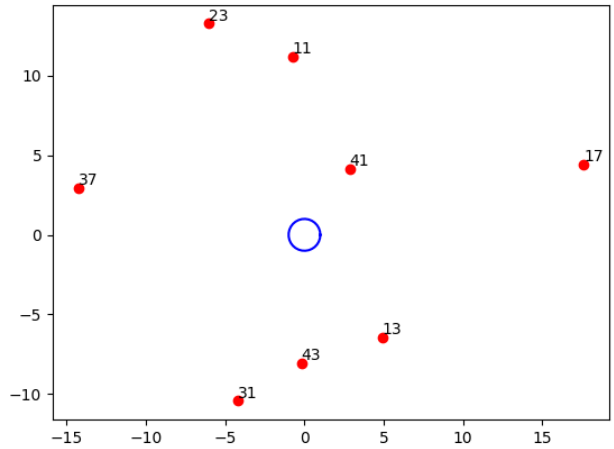
n=48



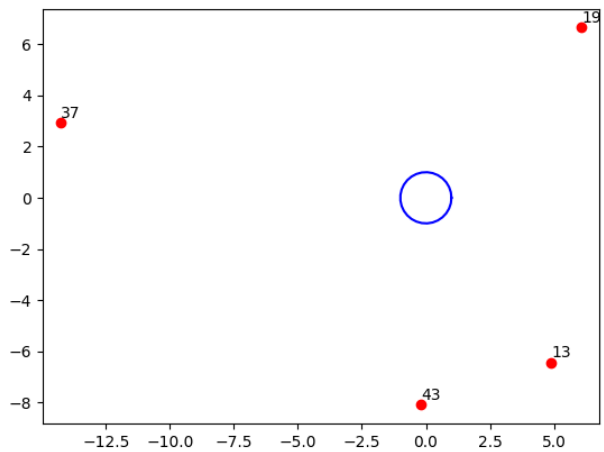
n=50



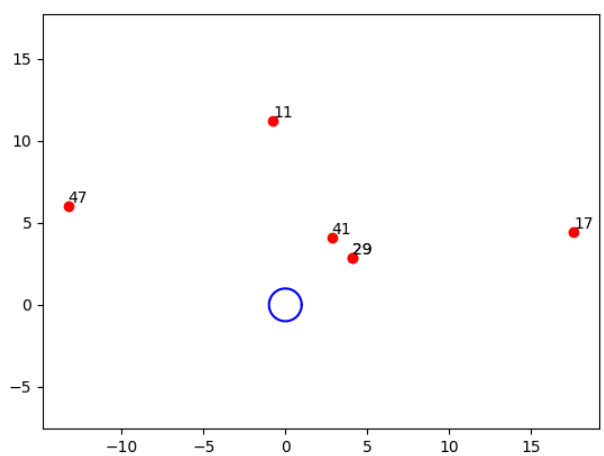
n=52


23
11
41
17
37
13
43
31
10
5
0
−5
−10
−15
−10
−5
0
5
10
15

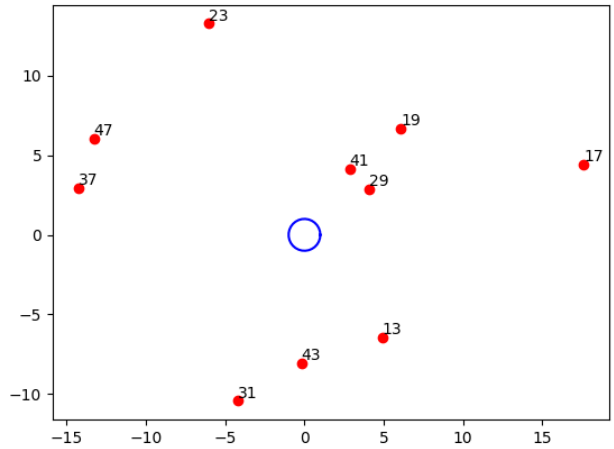
n=54


19
37
13
43
6
4
2
0
−2
−4
−6
−8
−12.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0

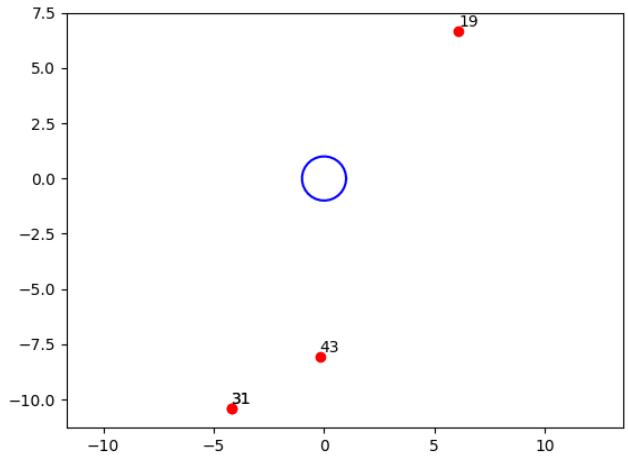
n=56


11
47
41
29
17
15
10
5
0
−5
−10
−5
0
5
10
15

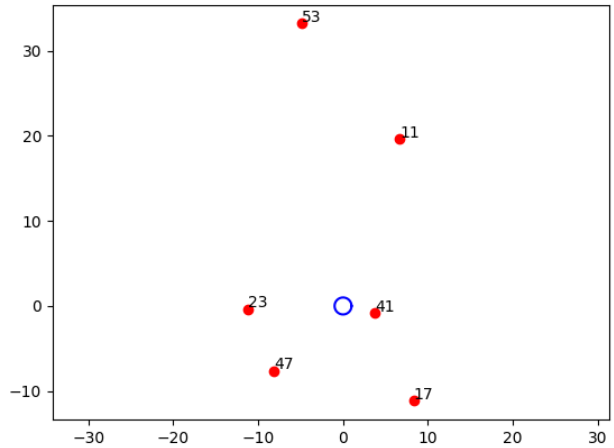
n=58


23
47
19
17
41
29
37
13
43
31
10
5
0
−5
−10
−15
−10
−5
0
5
10
15

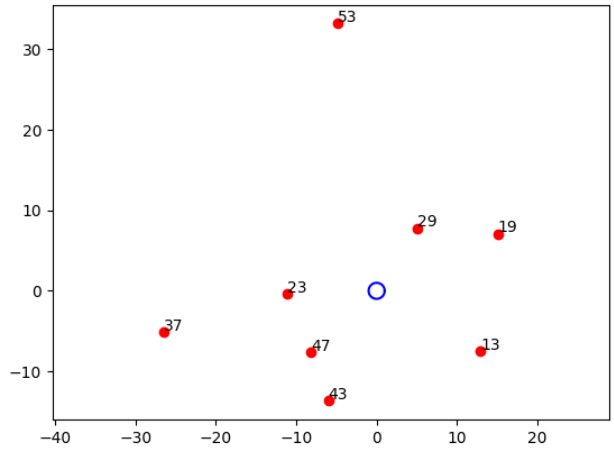
n=60


19
43
31
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10
−5
0
5
10

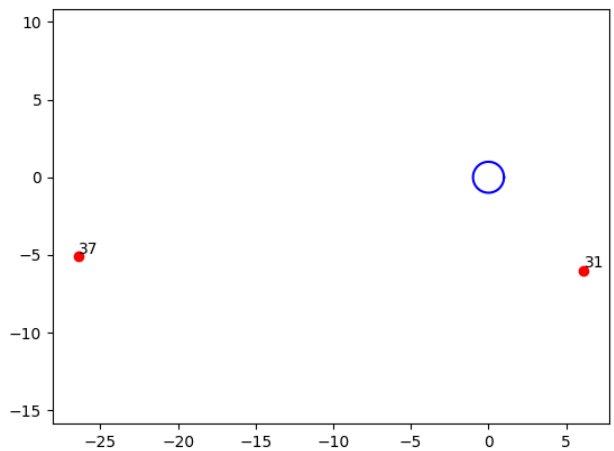
n=62


53
11
23
41
47
17
30
20
10
0
−10
−30
−20
−10
0
10
20
30

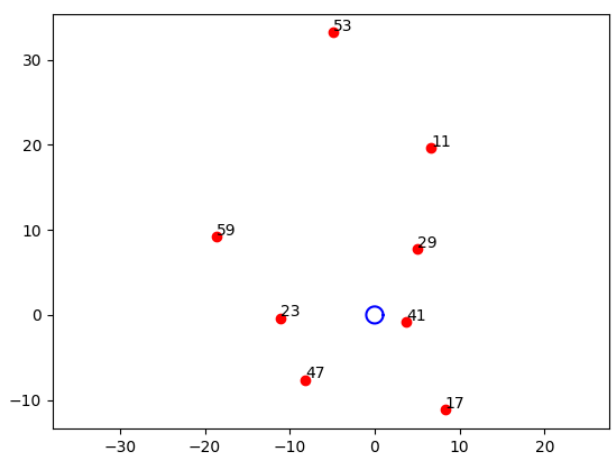
n=64



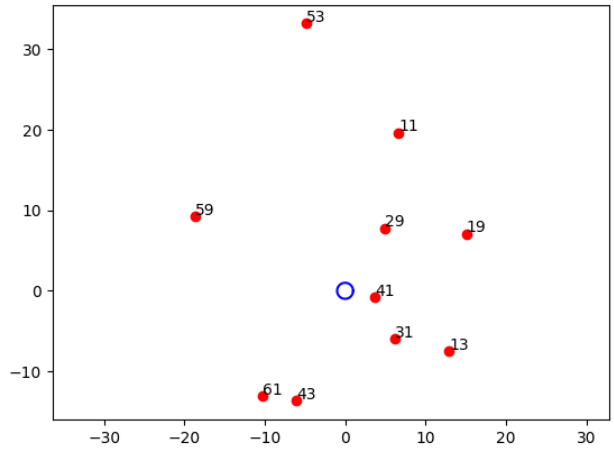
n=66



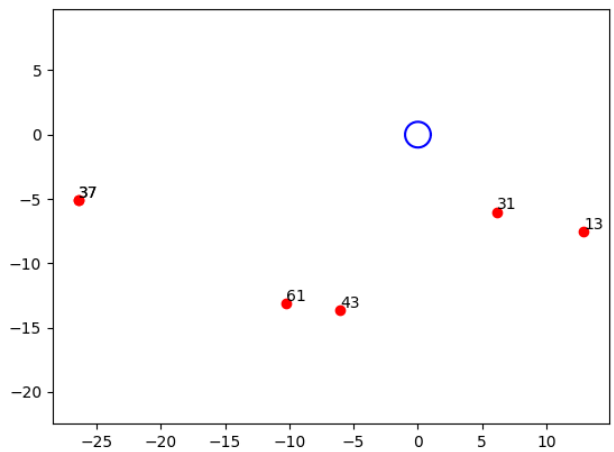
n=68



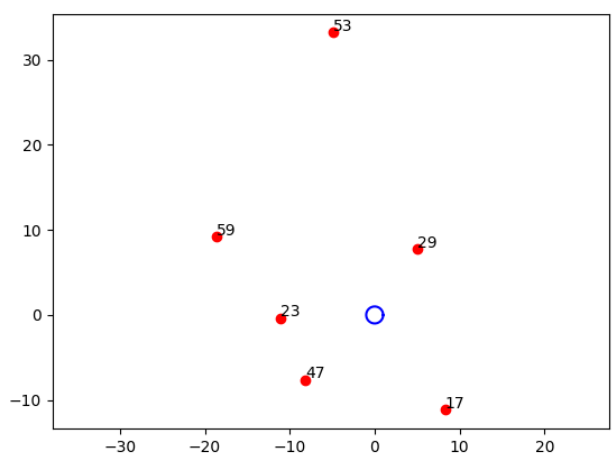
n=70



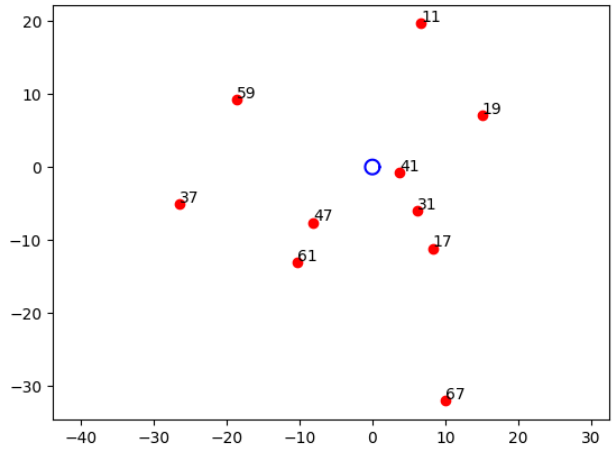
n=72



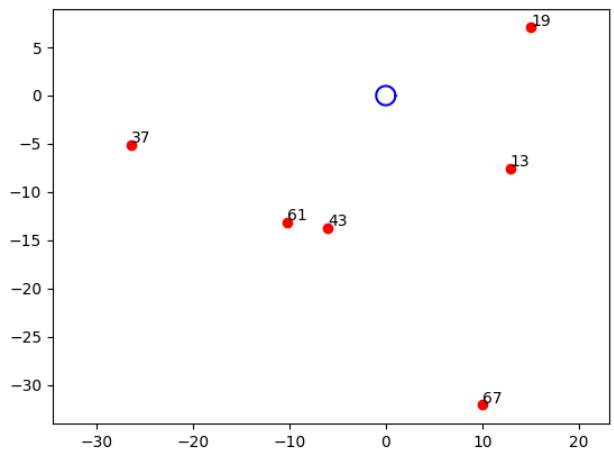
n=74



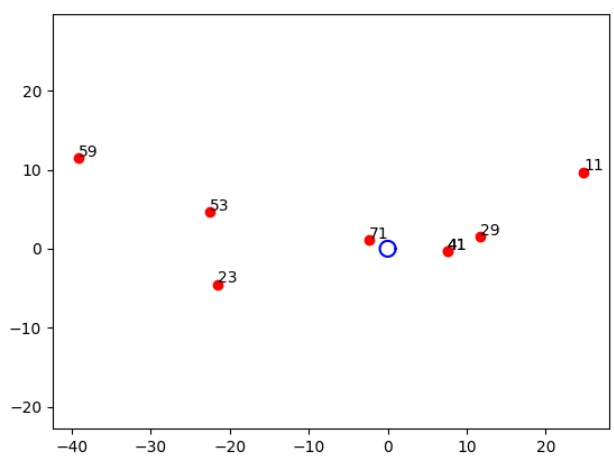
n=76



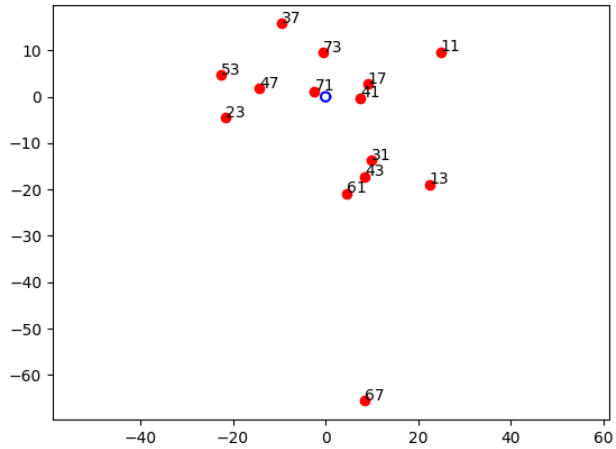
n=78



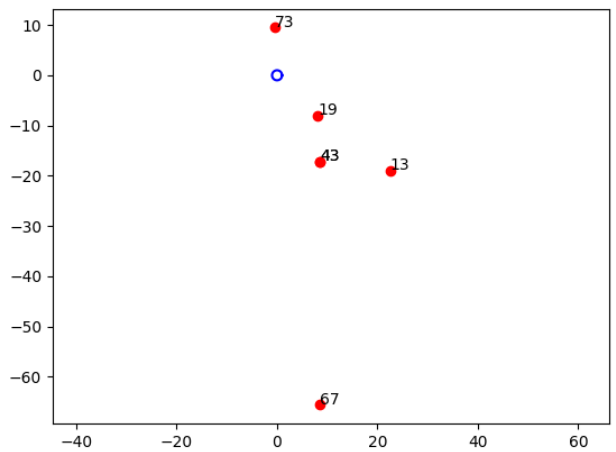
n=80



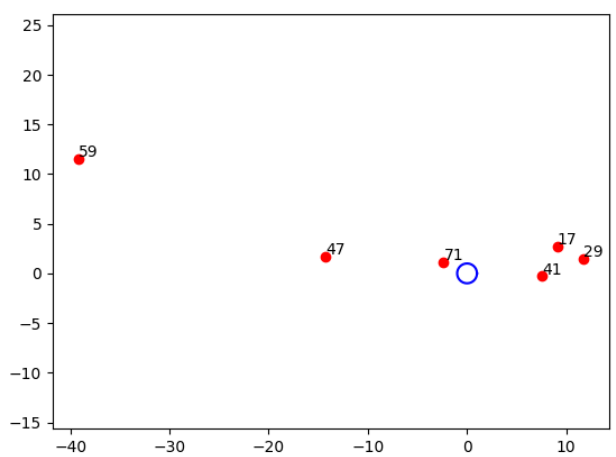
n=82


37
73
11
53
47
71
17
41
23
31
43
13
61
67

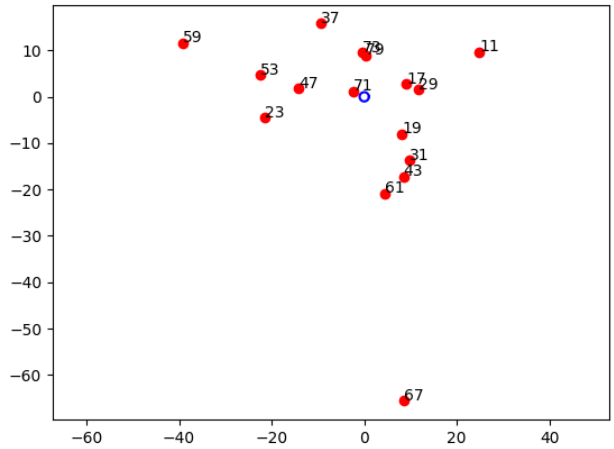
n=84


73
19
43
13
67

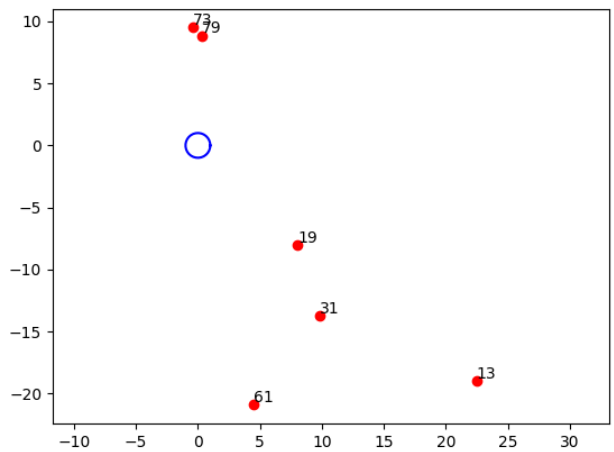
n=86


59
47
71
17
29
41

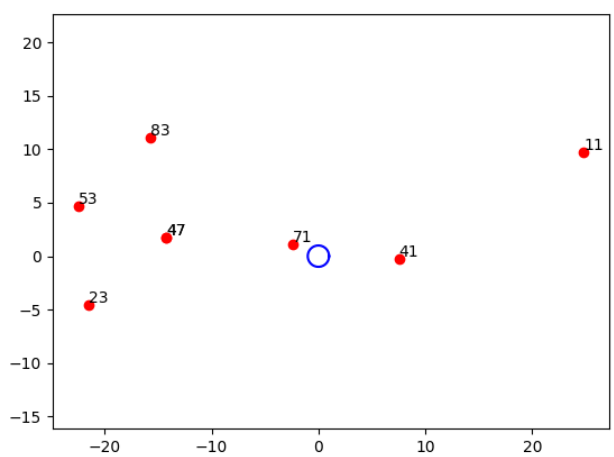
n=88


37
59
11
53
47
71
17
29
23
19
31
43
61
67

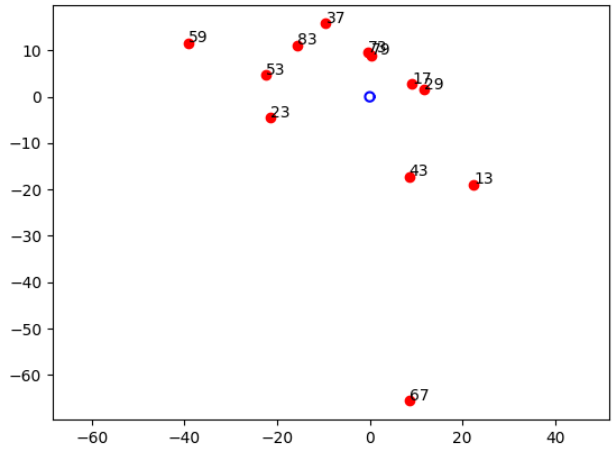
n=90


73
79
19
31
13
61

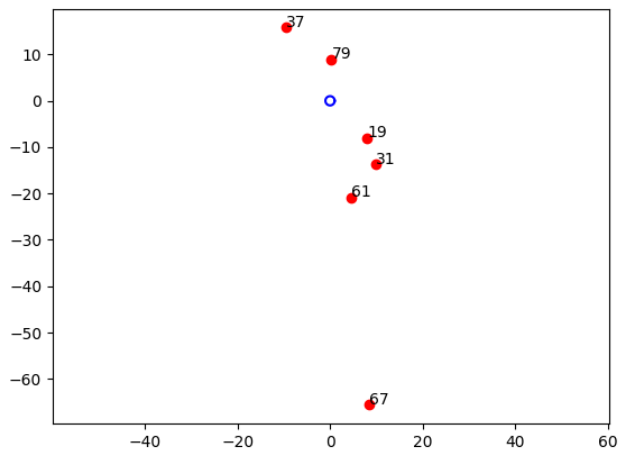
n=92


83
11
53
47
71
41
23

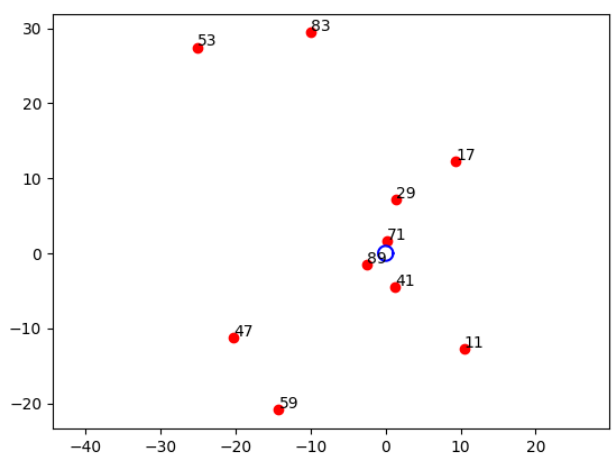
n=94


37
59
83
79
53
17
29
23
43
13
67

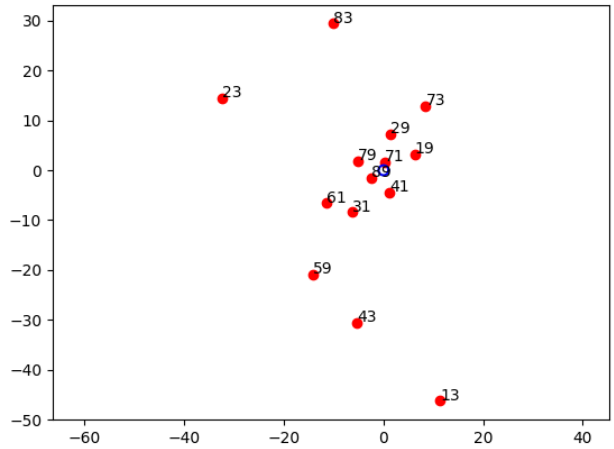
n=96


37
79
19
31
61
67

n=98


83
53
17
29
71
41
47
11
59

n=100



n=102

On est alors paré pour **tout ramener sur le nombre premier 2, qui ramène lui-même tout à des calculs de puissances de i (alias $1j$ en python)**. Le programme devient :

```
import mpmath
from mpmath import exp, sqrt, power, log
import numpy
from numpy import cos, sin, linspace, pi
import matplotlib.pyplot as plt

for n in range(6,103,2):
      theta = linspace(0,2*pi,30)
      r = 1
      a = r*cos(theta)
      b = r*sin(theta)
      z = a+1j*b
      plt.plot(a,b, color='blue')
      racine = int(sqrt(n))
      for x in range(2,n//2+1):
            prodbas = 1
            prodhaut = 1
            for p in range(2,racine+1):
                  lun = power(1j,4*x/p)
                  letout = power(1j,4*n/p)
                  prodbas = prodbas*(1-lun)
                  produit = letout/lun
                  prodhaut = prodhaut*(1-produit)
            if abs(prodbas) > pow(10,-10) and abs(prodhaut) > pow(10,-10):
                  print(x, ' dg de ',n)
                  plt.scatter(prodbas.real, prodbas.imag, color='red')
                  plt.annotate(x,xy=(prodbas.real, prodbas.imag+0.2))
                  plt.scatter(prodhaut.real, prodhaut.imag, color='red')
                  plt.annotate(n-x,xy=(prodhaut.real, prodhaut.imag+0.2))
      plt.axis('equal')
      plt.show()
```

Ce programme a le même comportement que le programme précédent, il trouve bien les décomposants 19, 31 et 37 de 98 (!). Les définitions précédemment proposées deviennent :

$$\forall n \geq 6, \exists x \geq 3 \,/$$

$$\prod_{\substack{2 \leq x \leq \frac{n}{2} \\ 2 \leq p \leq \sqrt{n}}} \left(1 - i^{\frac{4x}{p}}\right) \neq 0 \qquad \text{(a)}$$

et

$$\prod_{\substack{2 \leq x \leq \frac{n}{2} \\ 2 \leq p \leq \sqrt{n}}} \left(1 - i^{\frac{4(n-x)}{p}}\right) \neq 0 \qquad \text{(b)}$$

Et du fait de ces trouvailles, voici le plus petit programme qu'on ait écrit pour trouver les nombres premiers (`mpmaths` est la bibliothèque `Python` qui gère les nombres complexes, et on peut maintenant tester correctement le fait que le produit soit non nul, alors que les erreurs dans les multiplications complexes ne le permettaient pas) :

```
import mpmath
from mpmath import sqrt, power

for n in range(2,100):
      racine = int(sqrt(n))
      prodbas = 1
      for p in range(2,racine+1):
            lun = power(1j,4*n/p)
            prodbas = prodbas*(1-lun)
      if abs(prodbas) != 0:
            print(n, ' ', end='')
```

Et dont le résultat est bien sûr :

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

Mais cela reste une idée tordue : il s'agit simplement de savoir si les divisions euclidiennes du nombre par tous les nombres de 2 à sa racine ne tombent pas juste (n'ont pas pour reste 0), on peut écrire le programme ci-dessous, et cela n'avance à rien, finalement, même si ça aura été amusant : en python, le symbole "/" trouve un quotient réel pour des opérandes réels, tandis que le symbole "//" trouve le quotient entier, l'égalité "$a/b == a//b$" signifie que la division de a par b a un reste nul.

```
import math
from math import sqrt

for n in range(2,100):
      racine = int(sqrt(n))
      produit = 1
      for p in range(2,racine+1):
            lun = n/p != n//p
            produit = produit*lun
      if produit == 1:
            print(n, ' ', end='')
```