Problème posé

Il s'agit d'afficher toutes les listes d'entiers naturels $\{k;n\}$, telles que le point de coordonnées $(k;n)$ appartient au cercle de centre O et de rayon donné.
Par symétrie, on est ramené à afficher toutes les listes d'entiers naturels $\{k;n\}$, telles que le point de coordonnées $(k;n)$ appartient au cercle de centre O et de rayon donné, avec $x \geqslant 0$ et $y \geqslant 0$.
Trois algorithmes différents sont proposés (traduits ici dans le langage Python) :

```
def is_in_cercle_V1(rayon):
    """affiche les listes d'entiers {k, n} telles
    que le point (n; k) appartient au cercle
    Version la plus simple"""
    c=0
    for k in range(rayon+1):
        for n in range(rayon +1):
            c=c+1
            if k**2+n**2<=rayon**2:
                print('{',k,';',n,'}')
    return c
```

```
def is_in_cercle_V2(rayon):
    """affiche les listes d'entiers {k, n} telles
    que le point (n; k) appartient au cercle
    Version sans les doublons"""
    c=0
    for k in range(rayon+1):
        for n in range(k, rayon +1):
            c=c+1
            if k**2+n**2<=rayon**2:
                print('{',k,';',n,'}')
    return c
```

```
def is_in_cercle_V3(rayon):
    """affiche les listes d'entiers {k, n} telles
    que le point (n; k) appartient au cercle
    Version sans if"""
    c = 0
    for k in range(rayon+1):
        for n in range(k,(int(sqrt(rayon**2 - k ** 2)))
            +1):
            c=c+1
            print('{',k,';',n,'}')
    return c
```

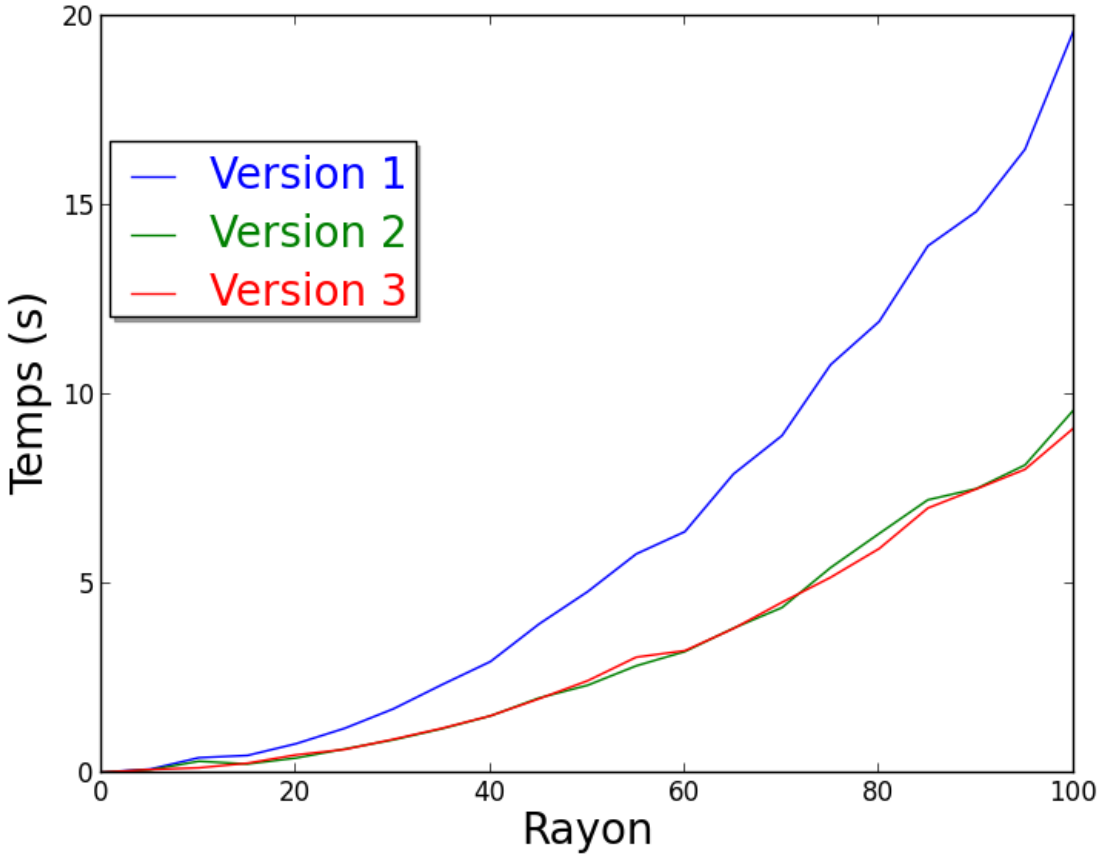
On se propose de comparer l'efficacité de ces trois algorithmes.

Comparaison des trois algorithmes

L'écriture de ces trois algorithmes montre que la version 2 implique moins de boucles que la version 1, et que la version 3 implique moins de boucles que le version 2. La version 3 est-elle néanmoins la plus rapide ?
Pour répondre à cette question, on peut mesurer le temps mis pour exécuter chacun de ces algorithmes avec un rayon de plus en plus grand.
En python, la fonction clock() permet de réaliser ces mesures :

```
1 def chrono(fonction, rayon=5):
2     debut=clock()
3     nb=fonction(rayon)
4     fin=clock()
5     return fin-debut
```

Voici les résultats obtenus :



On constate que les versions 2 et 3 sont sensiblement aussi rapides l'une que l'autre. Pourtant la version 3 comporte moins de boucles à l'exécution.
Une première idée pour expliquer ce paradoxe est la suivante : l'appel à la fonction racine carrée dans la version 3 provoque un temps de calcul plus important : il y a moins de boucles, mais plus de temps de calcul.
Cependant, une deuxième idée consiste à prendre en compte que l'opération qui prend le plus de temps dans un programme est l'affichage des résultats. On peut donc tenter de modifier nos algorithmes afin que les listes soient stockées dans un variable et affichées en une seule fois à la fin.

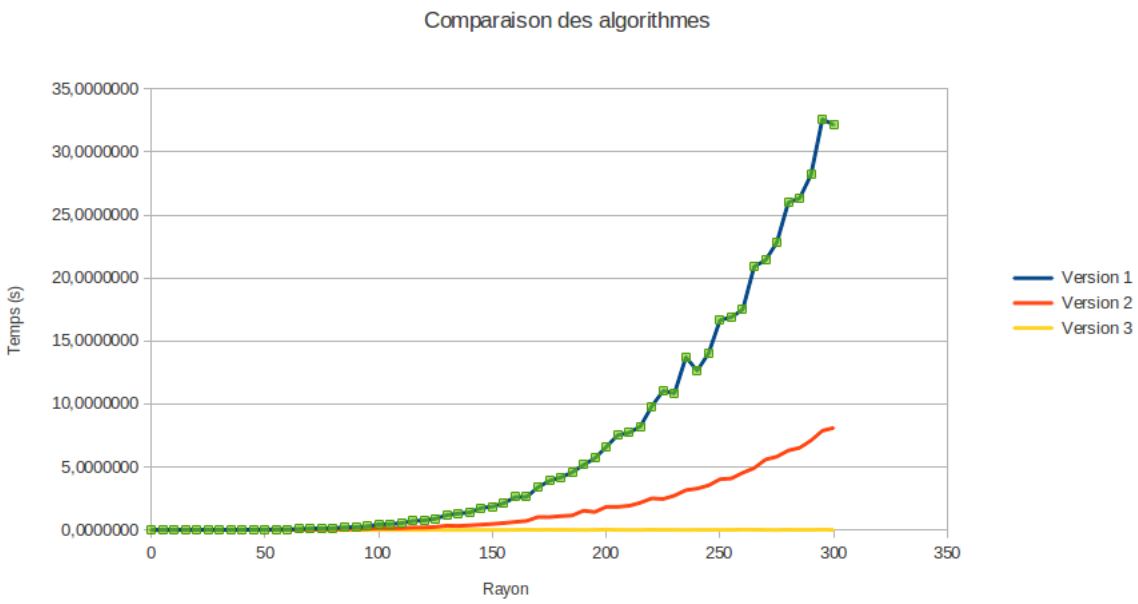
Voici les algorithmes modifiés en ce sens :

```
def is_in_cercle_V1(rayon):
    """affiche les listes d'entiers {n, k} telles
    que le point (n; k) appartient au cercle
    Version la plus simple"""
    c=0
    liste=[]
    for k in range(rayon+1):
        for n in range(rayon +1):
            c=c+1
            if k**2+n**2<=rayon**2:
                liste.append([k, n])
    print(liste)
    return c
```

```
def is_in_cercle_V2(rayon):
    """affiche les listes d'entiers {n, k} telles
    que le point (n; k) appartient au cercle
    Version qui sans doublons"""
    c=0
    liste=[]
    for k in range(rayon+1):
        for n in range(k, rayon +1):
            c=c+1
            if k**2+n**2<=rayon**2:
                liste.append([k, n])
    print(liste)
    return c
```

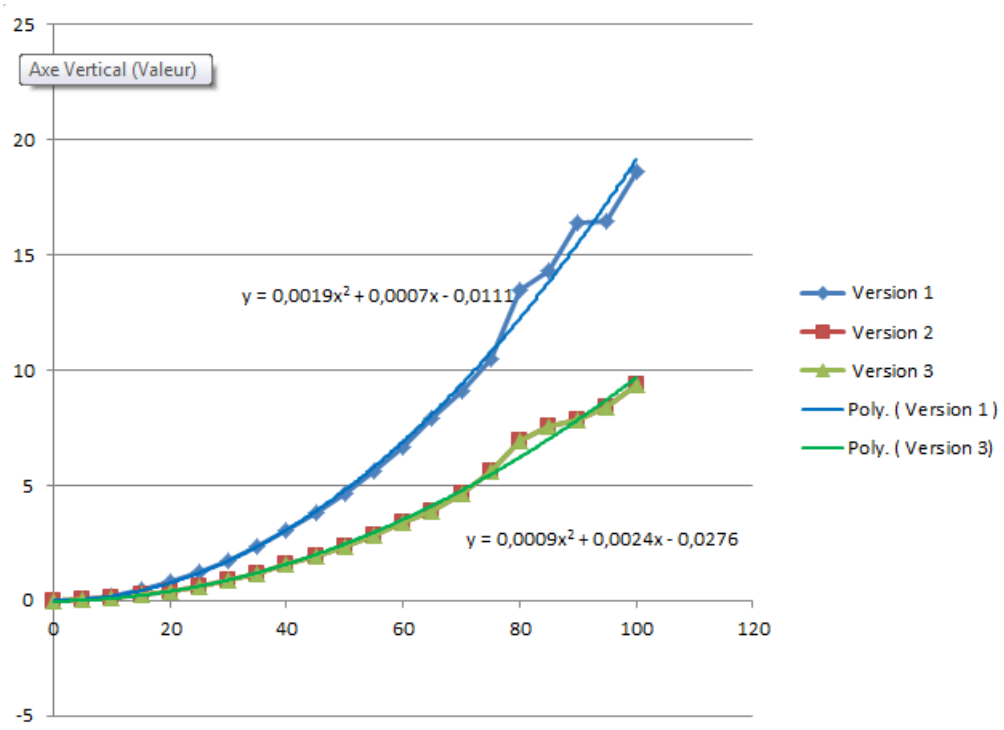
```
def is_in_cercle_V3(rayon):
    """affiche les listes d'entiers {n, k} telles
    que le point (n; k) appartient au cercle
    Version sans if"""
    c = 0
    liste=[]
    for k in range(rayon+1):
        for n in range(k,(int(sqrt(rayon**2 - k ** 2)))
            +1):
            c=c+1
            liste.append([k, n])
    return c
```

On se rend compte cette fois-ci que le temps d'exécution en en effet beaucoup plus faible pour l'algorithme n°3 que pour les deux autres, conformément au nombre de boucles effectuées.



Modélisation de la vitesse d'exécution

Les courbes obtenues peuvent être modélisée par un ajustement adéquat.
On recherche le meilleur ajustement en sauvegardant les coordonnées des points calculés dans un fichier de format CSV, lisible dans un tableur.



La vitesse d'exécution est donc, dans tous les cas, polynômiale.