CT : Équations diophantiennes

Durée de l'épreuve : 2h

- Le sujet est volontairement long et il n'est pas nécessaire de le traiter entièrement pour pouvoir obtenir la note 20/20.
- Lorsqu'un programme en python est demandé, l'important n'est pas de savoir s'il s'exécute correctement sans erreur de syntaxe, mais si les opérations effectuées répondent à la question.
- Votre attention est attirée sur la rigueur attendue lorsqu'une démonstration mathématique est demandée. De plus, les pistes de recherches pourront être récompensées dans le cas de réponses incomplètes.

La notation.

(i) On rappelle que pour $a \in \mathbb{N}$ and $b \in \mathbb{N}^*$, l'instruction `a % b` calcule le reste de la division (euclidienne) de `a` par `b`. L'instruction `a // b` calcule le quotient de la division (euclidienne) de `a` par `b`.

(ii) On rappelle que d divise a, noté $d|a$, signifie qu'il existe $k \in \mathbb{Z}$ tel que $a = d \cdot k$. Le PGCD de deux entiers non nuls a et b noté $\mathsf{PGCD}(a,b)$, est le plus grand diviseur commun de a et b, c'est à dire :

$$\mathsf{PGCD}(a,b) = \max(\{d \in \mathbb{Z} \text{ tel que } d|a \text{ et } d|b\})$$

(iii) On rappelle que a et b sont dit premiers entre eux si $\mathsf{PGCD}(a,b) = 1$.

(iv) On rappelle les fonctions `floor` et `ceil` du module `math`, qui donnent respectivement les parties entières inférieures et supérieures d'un nombre ($\lfloor x \rfloor$ et $\lceil x \rceil$).

I. PGCD

1. Petites questions préliminaires

1. Soit $n \in \mathbb{N}^*$. Que vaut $\mathsf{PGCD}(n,0)$?
2. Soient $a \in \mathbb{N}$ and $b \in \mathbb{N}^*$. Montrer que $\mathsf{PGCD}(a,b)$ est bien défini, au sens où l'ensemble des diviseurs commun a bien un maximum.

2. Premier algorithme

Supposons que a et b sont deux entiers naturels, non-simultanément égaux à 0. Le premier algorithme est donné par la fonction Python suivante :

```
def gcd_simple(a, b):
    """Calculates greatest common divisor of a and b"""
    if a == 0:
        return b
    if a > b:
        return gcd_simple(b, a)
    return gcd_simple(a, b - a)
```

3. Combien y a-t-il d'appels récursifs de la fonction `gcd_simple` pour $(a,b) = (67,31)$?
4. Montrer que l'algorithme effectue au plus $2(a + b)$ appels.

3. Deuxième algorithme : algorithme d'Euclide

```
def gcd(a, b):
    if a == 0:
        return b
    if a > b:
        return gcd(b, a)
    return gcd(a, b - a * (b // a))
```

On remarque, d'après les rappels que `b - a * (b // a) == b % a`.

5. Combien y a-t-il d'appels récursifs pour $(a,b) = (67,31)$ avec cet algorithme ?
6. On définit les nombres de Fibonacci par $F_0 = F_1 = 1$ et $F_{n+2} = F_{n+1} + F_n, n \geqslant 0$. Combien d'appels récursifs seront effectués pour $(a,b) = (F_n,F_{n+1})$?
7. Montrer par récurrence sur n, que si l'algorithme nécéssite au moins $(2n + 1)$ appels récursifs, alors $a \geqslant F_n$ et $b \geqslant F_{n+1}$.

II. Algorithme d'Euclide étendu

Dans cet partie, on cherche à résoudre l'équation $ax + by = 1$. On cherche à écrire un programme Python qui résout ce problème en renvoyant le couple (x,y) solution, s'il existe.

Si a et b sont premiers entre eux (c'est à dire que $\mathsf{PGCD}(a,b) = 1$), alors il y a au moins une solution (c'est le théorème de Bézout). Si il y a plusieurs solutions, une seule suffit. S'il y en a pas le programme devra renvoyer `(None,None)`.

```
def extended_gcd(a, b):
    if a == 0:
        if b != 1:
            return None, None
        return 0, 1
    if b < a:
        u, v = extended_gcd(b, a)
        if u is None:
            return None, None
        return v, u
    u, v = . . . . .
    if u is None:
        return None, None
    return . . . . .
```

Par exemple, `extended_gcd(67, 31)` renvoie `(-6,13)`.

8. Compléter les deux lignes manquantes de la fonction Python `extended_gcd` ci-dessus.
9. Écrire une fonction `extended_gcd_d` qui permet de trouver une solution à l'équation $ax + by = m$.
   Indication :
   - m doit être divisible par $d = \mathsf{PGCD}(a,b)$, sinon il y a pas de solution
   - On peut donc réduire le problème, afin d'en résoudre un à l'aide de ce qui précède :
   $$\frac{a}{d}x_0 + \frac{b}{d}y_0 = 1$$
   - On se ramene au problème initial : $(x,y) = (x_0,y_0)$

Observation. On admettra la propriété suivante. Si a et b sont premiers entre eux, alors les solutions de $ax + by = m$ sont sous la forme $(x_0,y_0) + (-b,a) \cdot k$, avec $k \in \mathbb{Z}$.

10. Quelle est alors la forme générale de la solution de $ax + by = m$, si a et b ne sont pas premiers entre eux ?
    Indication :
    - Utiliser la même méthode que pour la question 9.
    - Il peut être utile de faire un exemple explicite pour trouver la forme générale des solutions.

III. Ensemble de solutions

On définit une fonction Python `findmin` qui permet de trouver une solution (x,y), si elle existe, tel que x soit minimal dans un intervalle donné $[x_1,x_2]$.

```
def findmin(a, b, m, x1, x2):
    x0, y0 = extended_gcd_d(a, b, m)
    if x0 is None:
        return None, None
    d = gcd(a, b)
    a0 = a // d
    b0 = b // d
    t = . . . . . .
    xmin = x0 - b0 * t
    ymin = y0 + a0 * t
    if xmin <= x2:
        return . . . . . .
    else:
        return None, None
```

11. Compléter les deux lignes de la fonction `findmin`. Donner la valeur du couple (x,y) en sortie de `findmin(2, 4, 2020, 0, 3000)`

    Indication : pour la valeur de `t`, on pourra utiliser la différence entre deux solutions adjacentes de la question précédente.

12. Écrire une fonction `findmax`, qui permet de trouver une solution (x,y) tel que x soit maximal dans un intervalle donné $[x_1,x_2]$. Donner la valeur du couple (x,y) en sortie de `findmax(67, 31, 1, 2000, 2040)`

    Indication : Inspirez vous fortement de la fonction précédente

Important. Les fonctions `findmin` et `findmax` seront réutilisées dans la suite du sujet

On se donne un rectangle $x_1 \leqslant x \leqslant x_2$, $y_1 \leqslant y \leqslant y_2$. La fonction suivante compte le nombre des solutions d'une équation $ax + by = m$ dans ce rectangle.

```
def count_sol(a, b, m, x1, x2, y1, y2):
    x0, y0 = extended_gcd_d(a, b, m)
    if x0 is None:
        return 0
    d = gcd(a, b)
    a0 = a // d
    b0 = b // d

    ## Étape 1. Trouver le min et le max pour x dans [x1, x2]
    x1_min, y1_min = . . . . . .
    x1_max, y1_max = . . . . . .
```

```
    ## Étape 2. Trouver le min et le max pour y dans [y1, y2]
    ## Indication: changement (a, b) -> (b, a)
    ## Qui devient minimal et qui devient maximal?
    y2_max, x2_max = . . . . . .
    y2_min, x2_min = . . . . . .

    if x1_min is None:
        return 0
    if x2_min is None:
        return 0

    ## Étape 3. Construire l'intersection des deux intervalles
    ## [x1_min, x1_max] et [x2_min, x2_max]
    xmin = . . . . . .
    xmax = . . . . . .

    ## Étape 4. On peut maintenant compter le nombre de solutions
    n_solutions = . . . . . .

    return n_solutions
```

13. Compléter les 4 étapes décrites dans la fonction `count_sol`. Cette fonction permet de compter le nombre de solutions de $ax + by = m$ appartenant au rectangle décrit par `x1, x2, y1, y2`.

On cherche à trouver la solution de $ax + by = m$ tel que $(x + y)$ soit positive et minimale. Pour cela, on commence par supposer que $d = \mathsf{PGCD}(a,b)$ divise m. Nous savons par ce qui précède que les solutions de l'équation sont sous la forme $(x,y) = (x_0,y_0) + (-\frac{b}{d},\frac{a}{d})k$.

14. Trouver en fonction de a et b, les solutions tel que $(x + y)$ soit positive et minimale.
15. Compléter la fonction `minsolsum` ci-dessous qui permet de trouver ces solutions de manière algorithmique.

```
def minsolsum(a, b, m):
    d = gcd(a, b)
    if m % d != 0:
        return None
    x0, y0 = extended_gcd_d(a, b, m)
    a0 = a // d
    b0 = b // d
```

```
    if a == b:
        return . . . . . .
    t = . . . . . .
    if a > b:
        t = . . . . .
    x = . . . . . .
    y = . . . . . .
    return x, y
```

16. Trouver en fonction de a et b, une solution $ax + by = m$ minimisant $x^2 + y^2$.

```
def minsolsquare(a, b, m):
    d = gcd(a, b)
    if m % d != 0:
        return None
    x0, y0 = extended_gcd_d(a, b, m)
    a0 = a // d
    b0 = b // d
    tmin = . . . . . .
    t1 = floor(tmin)
    t2 = ceil(tmin)
    x1, y1 = . . . . . .
    x2, y2 = . . . . . .
    sum_squares_1 = x1 * x1 + y1 * y1
    sum_squares_2 = x2 * x2 + y2 * y2
    if sum_squares_1 < sum_squares_2:
        return x1, y1
    else:
        return x2, y2
```

17. Compléter la fonction `minsolsquare` qui permet de trouver ces solutions de manière algorithmique.

18. Donner la solution de carré minimal de $2x + 4y = 2020$.