

# BROUILLON - ALGORITHME D'EUCLIDE ET COEFFICIENTS DE BACHET-BÉZOUT POUR LES HUMAINS

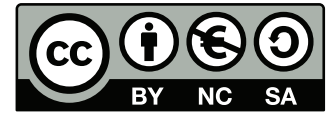
CHRISTOPHE BAL

*Document, avec son source L<sup>A</sup>T<sub>E</sub>X, disponible sur la page  
<https://github.com/bc-writing/drafts>.*

---

## Mentions « légales »

Ce document est mis à disposition selon les termes de la licence Creative Commons “Attribution - Pas d'utilisation commerciale - Partage dans les mêmes conditions 4.0 International”.



## TABLE DES MATIÈRES

1.	Où allons-nous ?	2
2.	L'algorithme « human friendly » appliqué de façon magique	3
2.1.	Un exemple complet façon « <i>diaporama</i> »	3
2.2.	Phase 1 – Au début était l'algorithme d'Euclide...	3
2.3.	Phase 2 – Remontée facile des étapes	3
2.4.	Et voilà comment conclure !	5
3.	Pourquoi cela marche-t-il ?	6
3.1.	Avec des arguments élémentaires	6
3.2.	Avec des matrices pour y voir plus clair	7
4.	Des coefficients via des algorithmes programmables	8
4.1.	La version « humaine » à la main	8
4.2.	La version « humaine » via les matrices	9
4.3.	Pas terribles...	9
4.4.	On peut faire mieux !	10
5.	AFFAIRE À SUIVRE...	11

## 1. OÙ ALLONS-NOUS ?

Un résultat classique d'arithmétique dit qu'étant donné  $(a; b) \in \mathbb{N}^* \times \mathbb{N}^*$ , il existe  $(u; v) \in \mathbb{Z} \times \mathbb{Z}$  tel que  $au + bv = \text{pgcd}(a; b)$ . Les entiers  $u$  et  $v$  seront appelés « *coefficients de Bachet-Bézout* » et  $au + bv = \text{pgcd}(a; b)$  sera nommée « *relation de Bachet-Bézout* ». Notons qu'il n'y a pas unicité car nous avons par exemple :

$$(-3) \times 12 + 1 \times 42 = 4 \times 12 + (-1) \times 42 = 6 = \text{pgcd}(12; 42)$$

Nous allons voir comment trouver de tels entiers  $u$  et  $v$  tout d'abord de façon humainement rapide puis ensuite via un algorithme programmable efficace.

## 2. L'ALGORITHME « HUMAN FRIENDLY » APPLIQUÉ DE FAÇON MAGIQUE

2.1. **Un exemple complet façon « *diaporama* ».** Sur le lieu de téléchargement de ce document se trouve un fichier PDF de chemin relatif `bezout-coef-for-human/slide-version.pdf` présentant la méthode sous la forme d'un diaporama. Nous vous conseillons de le regarder avant de lire les explications ci-après.

2.2. **Phase 1 – Au début était l'algorithme d'Euclide...** Pour chercher des coefficients de Bachet-Bézout pour  $(a; b) = (27; 141)$ , on commence par appliquer l'algorithme d'Euclide « *verticalement* » comme suit.

141

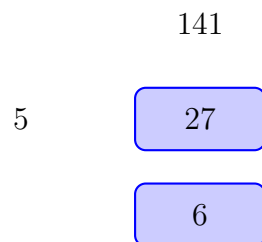
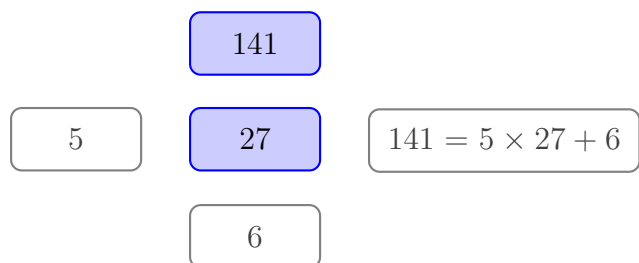
141

27

27

Étape 1 : le plus grand naturel est mis au-dessus.

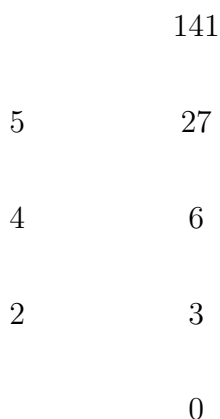
Étape 2 : deux naturels à diviser.



Étape 3 : première division euclidienne.

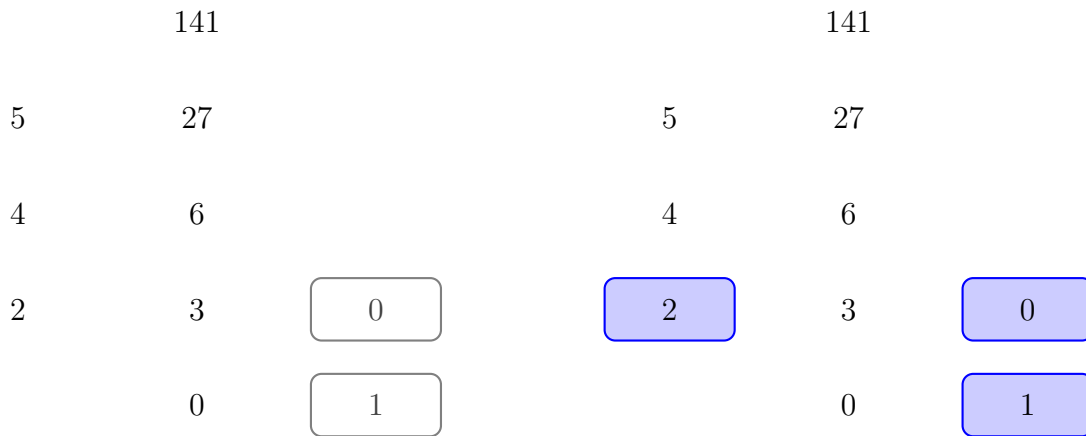
Étape 4 : on passe aux deux naturels suivants.

En répétant ce processus, nous arrivons à la représentation suivante où nous n'avons pas besoin de garder la trace des divisions euclidiennes.



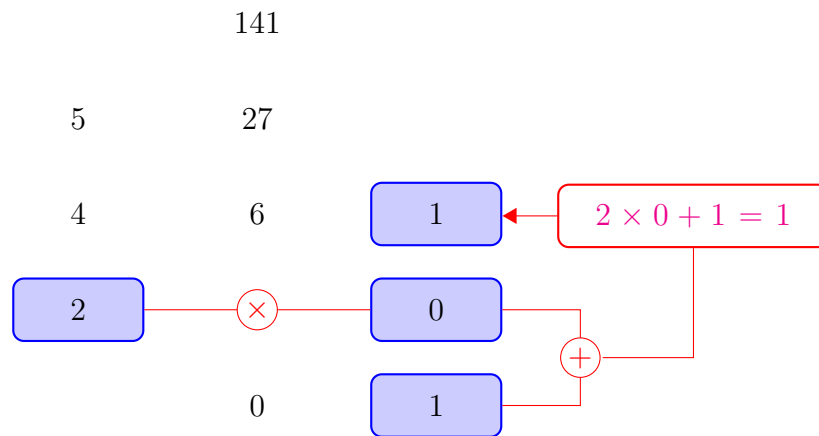
Étape finale (1<sup>re</sup> phase) : l'algorithme d'Euclide « vertical ».

2.3. **Phase 2 – Remontée facile des étapes.** La méthode classique consiste à remonter les calculs. Mais comment faire cette remontée tout en évitant un claquage neuronal ? L'astuce est la suivante.

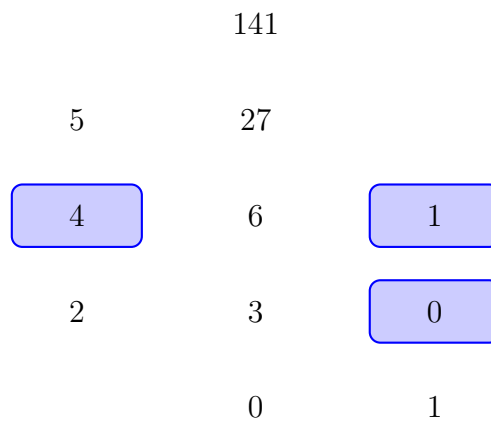


*Étape 1 : ajout d'une nouvelle colonne.*

*Étape 2 : on n'utilise pas la colonne centrale.*



*Étape 3 : on fait une sorte de division « inversée ».*



*Étape 4 : on passe aux trois naturels suivants.*

En répétant ce processus, nous arrivons à la représentation suivante.

	141	21
5	27	4
4	6	1
2	3	0
	0	1

*Étape finale (2<sup>e</sup> phase) : remontée en voie libre des calculs.*

#### 2.4. Et voilà comment conclure !

	141	21	
5	27	4	$141 \times 4 - 21 \times 27 = -3$
4	6	1	
2	3	0	
	0	1	

*Étape finale (la vraie) : on finit avec un produit en croix.*

Des coefficients de Bachet-Bézout s'obtiennent sans souci via l'équivalence suivante où nous avons  $3 = \text{pgcd}(27, 141)$ .

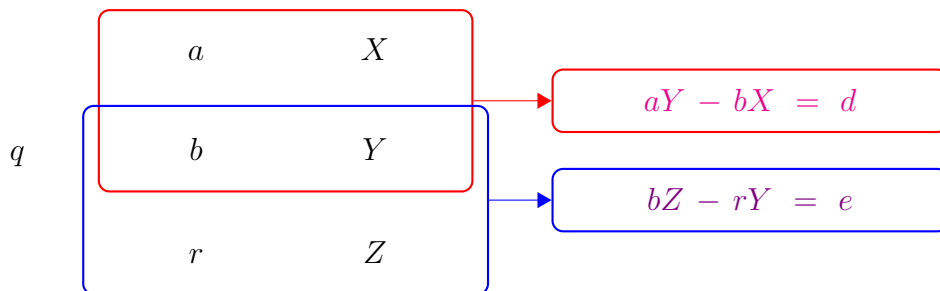
$$141 \times 4 - 27 \times 21 = -3 \iff 27 \times 21 - 141 \times 4 = 3$$

Nous allons voir, dans la section qui suit, que l'on obtient forcément à la fin  $\pm \text{pgcd}(27, 141)$ .

En pratique, nous n'avons pas besoin de détailler les calculs comme nous l'avons fait à certains moments afin d'expliquer comment procéder. Avec ceci en tête, on comprend toute l'efficacité de la méthode présentée, mais pas encore justifiée, car il suffit de garder une trace minimale, mais complète, des étapes tout en ayant à chaque étape des opérations assez simples à effectuer. Il reste à démontrer que notre méthode marche à tous les coups. Ceci est le propos de la section suivante.

## 3. POURQUOI CELA MARCHE-T-IL ?

3.1. **Avec des arguments élémentaires.** Commençons par une preuve vérificative qui malheureusement ne nous permet pas de voir d'où vient l'astuce (*nous explorerons ceci dans la sous-section suivante*).



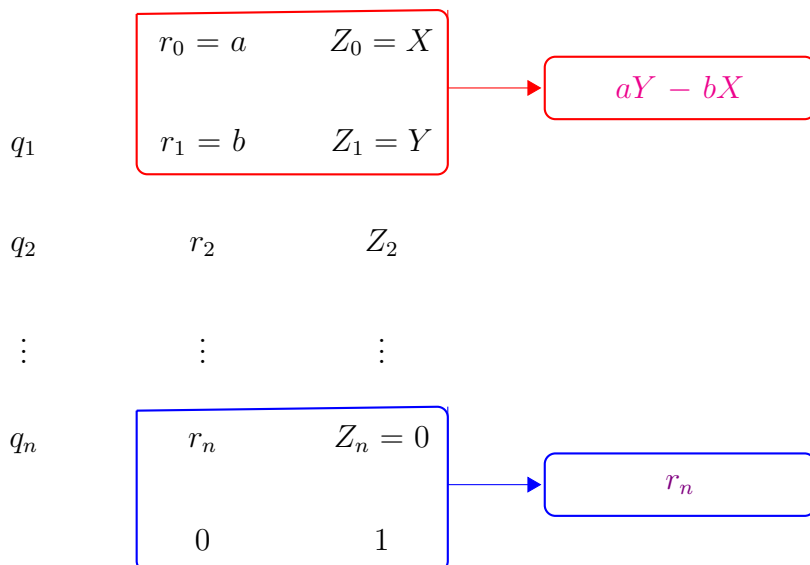
*Calculs faits dans les deux phases.*

Par construction, nous avons  $a = qb + r$  et  $X = qY + Z$ . Ceci nous donne :

$$\begin{aligned}
 d &= aY - bX \\
 &= (qb + r)Y - b(qY + Z) \\
 &= rY - bZ \\
 &= -e
 \end{aligned}$$

Donc si l'on fait « *glisser* » des carrés sur les deux colonnes de droite, les produits en croix dans ces carrés ne différeront que par leur signe.

La représentation symbolique « *complète* » ci-dessous donne  $aY - bX = \pm \text{pgcd}(a; b)$  car le dernier reste non nul de l'algorithme d'Euclide est  $\text{pgcd}(a; b)$ . Ceci prouve la validité de la méthode dans le cas général. On comprend au passage l'ajout initial du 0 et du 1 dans la 3<sup>e</sup> colonne. Bien entendu,  $(-1)$  aurait pu convenir à la place de 1, et l'on constate que 0 peut être remplacé par n'importe quelle valeur entière.



*Représentation symbolique au complet.*

**3.2. Avec des matrices pour y voir plus clair.** Oublions tout ce que nous avons vu précédemment. Soit  $(a; b) \in \mathbb{N}^* \times \mathbb{N}^*$  avec  $a > b$ . Nous cherchons  $(u; v) \in \mathbb{Z} \times \mathbb{Z}$  tel que  $au + bv = \text{pgcd}(a; b)$ . La petite astuce est de noter que  $au + bv = \det M$  où  $M = \begin{pmatrix} a & -v \\ b & u \end{pmatrix}$ .

Nous allons poser  $X = -v$  et  $Y = u$  de sorte que  $M = \begin{pmatrix} a & X \\ b & Y \end{pmatrix}$  et raisonner en supposant l'existence de  $u$  et  $v$ <sup>1</sup>.

Soit ensuite  $a = qb + r$  la division euclidienne de  $a$  par  $b$ . L'algorithme d'Euclide nous fait alors travailler avec  $(b; r)$  au lieu de  $(a; b)$ . Comme  $r = a - qb$ , on peut considérer la matrice  $N = \begin{pmatrix} a - qb & X - qY \\ b & Y \end{pmatrix}$  qui vérifie  $\det N = \det M$  puis, afin d'avoir  $b$  en haut, la matrice  $P = \begin{pmatrix} b & Y \\ a - qb & X - qY \end{pmatrix}$  qui vérifie  $\det P = -\det M$ .

Notant  $Z = X - qY$ , de sorte que  $P = \begin{pmatrix} b & Y \\ r & Z \end{pmatrix}$ , nous avons  $X = Z + qY$ . Ceci justifie la construction utilisée lors de la phase de remontée.

Pour passer à une nouvelle preuve, notons que  $\begin{pmatrix} b & Y \\ r & Z \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & -q \end{pmatrix} \cdot \begin{pmatrix} a & X \\ b & Y \end{pmatrix}$  puis introduisons les notations suivantes.

- $r_0 = a$ ,  $r_1 = b$ ,  $Z_0 = X$  et  $Z_1 = Y$  où  $Z_0$  et  $Z_1$  ne sont pas connus pour le moment.
- Pour  $k \in \mathbb{N}^*$ , on note  $r_{k-1} = r_k q_k + r_{k+1}$  la division euclidienne de  $r_{k-1}$  par  $r_k$ , puis ensuite on pose  $Z_{k+1} = Z_{k-1} - Z_k q_k$  de sorte que  $Z_{k-1} = Z_k q_k + Z_{k+1}$ .
- On note enfin  $M_k = \begin{pmatrix} r_k & Z_k \\ r_{k+1} & Z_{k+1} \end{pmatrix}$  pour  $k \in \mathbb{N}$  et  $Q_k = \begin{pmatrix} 0 & 1 \\ 1 & -q_k \end{pmatrix}$  pour  $k \in \mathbb{N}^*$  de sorte que nous avons  $M_{k+1} = Q_{k+1} \cdot M_k$  pour  $k \in \mathbb{N}$ . Il est immédiat que  $Q_k$  est inversible d'inverse  $R_k = \begin{pmatrix} q_k & 1 \\ 1 & 0 \end{pmatrix}$  avec  $\det R_k = -1$ .

L'algorithme d'Euclide nous donne l'existence de  $n \in \mathbb{N}$  un indice minimal tel que  $r_{n+1} = 0$ , et que pour cet indice nous avons  $r_n = \text{pgcd}(a; b)$ .

Comme  $M_n = \prod_{k=n}^1 Q_k \cdot M_0$ , nous avons  $M_0 = \prod_{k=1}^n R_k \cdot M_n$  avec  $M_0 = \begin{pmatrix} r_0 & Z_0 \\ r_1 & Z_1 \end{pmatrix} = \begin{pmatrix} a & X \\ b & Y \end{pmatrix}$  et  $M_n = \begin{pmatrix} r_n & Z_n \\ 0 & Z_{n+1} \end{pmatrix} = \begin{pmatrix} \text{pgcd}(a; b) & Z_n \\ 0 & Z_{n+1} \end{pmatrix}$ .

Comme  $\det M_0 = \pm 1 \det M_n$  car  $\det R_k = -1$ , il suffit de choisir  $Z_{n+1} = 1$ , avec  $Z_n$  quelconque, pour avoir  $aY - bX = \pm \text{pgcd}(a; b)$ . Voilà comment découvrir la méthode visuelle vue précédemment où le choix particulier  $Z_n = 0$  sert juste à simplifier les premiers calculs.

**Remarque 1.** Comme  $Z_n$  peut être quelconque, nous pouvons produire une infinité de coefficients de Bachet-Bézout. En effet, les étapes de « remontée » sont du type  $Z_{k-1} = Z_k q_k + Z_{k+1}$  avec toujours  $q_k > 0$ , donc des valeurs naturelles non nulles<sup>2</sup> différentes de  $Z_n$  produiront des valeurs différentes de  $Z_0$ .

1. Ce n'est qu'à la fin de la preuve que nous aurons effectivement prouver l'existence de  $u$  et  $v$ .

2. En fait, il n'est pas besoin d'imposer une condition particulière à  $Z_n$ . Voyez-vous pourquoi?

## 4. DES COEFFICIENTS VIA DES ALGORITHMES PROGRAMMABLES

4.1. **La version « humaine » à la main.** Il n'est pas dur de coder directement la méthode humaine par descente puis remontée<sup>3</sup>. Voici un algorithme, peu efficace mais instructif, où  $\star$  est un symbole à part,  $R[-1]$  le dernier élément de la liste  $R$  et  $R[-2]$  l'avant-dernier, et enfin  $[x, y] + [r, s, t] \stackrel{\text{déf}}{=} [x, y, r, s, t]$  (*additionner des listes c'est les concaténer et donc  $R + [r]$  est un raccourci pour « on ajoute l'élément  $r$  à droite de la liste  $R$  »*).

**Algorithme 1** : Descente et remontée avec du papier et un crayon

**Donnée** :  $(a; b) \in \mathbb{N}^* \times \mathbb{N}^*$  avec  $b \leq a$

**Résultat** :  $(u; v) \in \mathbb{Z} \times \mathbb{Z}$  tel que  $au + bv = \text{pgcd}(a; b)$

**Actions**

```
# Phase de descente
# Q est une liste qui va stocker les quotients entiers  $q_k$ .
# R est une liste qui va stocker les restes  $r_k$  (rappelons que
#  $r_0 = a$  et  $r_1 = b$ ).
 $Q \leftarrow [\star]$  ;  $R \leftarrow [a, b]$ 
Tant Que  $R[-1] \neq 0$  :
     $\alpha \leftarrow R[-2]$  ;  $\beta \leftarrow R[-1]$ 
     $\alpha = q\beta + r$  est la division euclidienne standard.
     $Q \leftarrow Q + [q]$  ;  $R \leftarrow R + [r]$ 

# Phase de remontée
# Z est une liste qui va stocker les entiers tout à droite.
 $\varepsilon \leftarrow 1$  ;  $Z \leftarrow [1, 0]$  ;  $c \leftarrow (-1)$ 
Tant Que  $Q[c] \neq \star$  :
     $z \leftarrow Q[c] \cdot Z[-2] + Z[-1]$ 
     $Z \leftarrow Z + [z]$ 
     $\varepsilon \leftarrow (-\varepsilon)$  ;  $c \leftarrow c - 1$ 

# On gère le signe devant pgcd grâce à  $\varepsilon$ .
 $u \leftarrow \varepsilon \cdot Z[-2]$  ;  $v \leftarrow (-\varepsilon \cdot Z[-1])$ 
Renvoyer  $(u; v)$ 
```

Nous avons traduit brutalement ce que l'on fait humainement mais à bien y regarder, la seule liste dont nous avons réellement besoin est  $Q$ . On peut donc proposer la variante suivante programmable qui est à la fois proche de la version de descente et remontée tout en limitant l'impact sur la mémoire.

3. Sur le lieu de téléchargement du document que vous lisez, se trouvent les fichiers `down.py` et `up.py` dans le dossier `bezout-coef-for-human/euclid2tikz`. Ces codes traduisent directement la méthode à la main par descente puis remontée.



**Algorithme 2** : Descente et remontée moins mémophage**Donnée** :  $(a; b) \in \mathbb{N}^* \times \mathbb{N}^*$  avec  $b \leq a$ **Résultat** :  $(u; v) \in \mathbb{Z} \times \mathbb{Z}$  tel que  $au + bv = \text{pgcd}(a; b)$ **Actions**

# Phase de descente

 $Q \leftarrow [\star]$ **Tant Que**  $b \neq 0$  :     $a = qb + r$  est la division euclidienne standard.     $Q \leftarrow Q + [q]$      $a \leftarrow b ; b \leftarrow r$ 

# Phase de remontée

 $u \leftarrow 1 ; v \leftarrow 0$  $\varepsilon \leftarrow 1 ; c \leftarrow (-1)$ **Tant Que**  $Q[c] \neq \star$  :     $temp \leftarrow Q[c]v + u ; u \leftarrow v ; v \leftarrow temp$      $\varepsilon \leftarrow (-\varepsilon) ; c \leftarrow c - 1$  $u \leftarrow \varepsilon \cdot u ; v \leftarrow (-\varepsilon \cdot v)$ **Renvoyer**  $(u; v)$ 

4.2. **La version « humaine » via les matrices.** Voici la version matricielle de l'algorithme de remontée et descente où de nouveau on limite l'impact sur la mémoire. La matrice  $R$  correspond au produit cumulé des matrices  $R_k$ .

**Algorithme 3** : Descente et remontée via les matrices**Donnée** :  $(a; b) \in \mathbb{N}^* \times \mathbb{N}^*$  avec  $b \leq a$ **Résultat** :  $(u; v) \in \mathbb{Z} \times \mathbb{Z}$  tel que  $au + bv = \text{pgcd}(a; b)$ **Actions** $a_0 \leftarrow a ; \varepsilon \leftarrow 1$  $R \leftarrow \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ **Tant Que**  $b \neq 0$  :     $a = qb + r$  est la division euclidienne standard.     $R \leftarrow R \cdot \begin{pmatrix} q & 1 \\ 1 & 0 \end{pmatrix}$      $a \leftarrow b ; b \leftarrow r$      $\varepsilon \leftarrow (-\varepsilon)$  $R \leftarrow R \cdot \begin{pmatrix} a_0 & 0 \\ 0 & \varepsilon \end{pmatrix}$  $u \leftarrow R_{22} ; v \leftarrow (-R_{12})$ **Renvoyer**  $(u; v)$ 

4.3. **Pas terribles...** Les algorithmes 2 et 3 vus ci-dessus sont informatiquement très maladroits. Voici pourquoi.

- (1) L'algorithme 2 utilise une liste de taille la somme de 1 et du nombre d'étapes de l'algorithme d'Euclide pour calculer  $\text{pgcd}(a; b)$ . Dans la section ??, nous verrons que ce nombre d'étapes est environ égal à 5 fois le nombre de chiffres de l'écriture décimale du plus petit des deux entiers  $a$  et  $b$ . Donc si l'on travaille avec des entiers de tailles assez grandes, la taille de la liste risque de devenir problématique.
- (2) Le problème avec l'algorithme 3 est le produit cumulé des matrices qui cache beaucoup d'opérations intermédiaires. Il serait bien de pouvoir s'en passer !

Dans la section qui suit nous allons voir que l'on peut chercher plus efficacement des coefficients de Bachet-Bézout.

**4.4. On peut faire mieux !** Dans la section 3.1, nous avons vu que la clé de la réussite de l'algorithme de descente et remontée est l'égalité  $aY - bX = bZ - rY$  dans la représentation ci-dessous où  $a = qb + r$  est la division euclidienne standard et  $X = qY + Z$ .

$$\begin{array}{cc}
 & \boxed{\begin{array}{cc} a & X \\ b & Y \end{array}} \\
 q & \boxed{\begin{array}{cc} b & Y \\ r & Z \end{array}}
 \end{array}$$

*Calculs faits dans les deux phases.*

Nous avons donc exhibé un invariant et dès que l'on arrive à  $r = 0$ , c'est à dire à la fin de la phase de descente, nous pouvons avoir  $bZ - rY = \text{pgcd}(a; b)$  grâce au choix  $Z = 1$ , et du coup en remontant les calculs nous arrivons à nos fins (*au signe près*).

La méthode précédente est peu efficace à cause de la nécessité de mémoriser certains calculs pour la phase de remontée. Ceci est une contrainte forte ! Nous allons essayer de nous passer de cette nécessité de mémoriser des choses. Pour cela repartons de la représentation symbolique « *complète* » ci-dessous où  $r_{n+1} = 0$  et  $r_n = \text{pgcd}(a; b)$ .

$$\begin{array}{ccc}
 & r_0 = a & Z_0 \\
 & & \\
 q_1 & r_1 = b & Z_1 \\
 & & \\
 q_2 & r_2 & Z_2 \\
 & & \\
 \vdots & \vdots & \vdots \\
 q_n & r_n & Z_n \\
 & & \\
 & r_{n+1} & Z_{n+1}
 \end{array}$$

*Représentation symbolique au complet où  $r_{n+1} = 0$ .*

Ce qui fait fonctionner l'algorithme de descente puis remontée c'est que les  $r_k$  et les  $Z_k$  vérifient la même relation de récurrence.

- (1)  $r_{k+2} = r_k - q_{k+1}r_{k+1}$  car  $r_k = q_{k+1}r_{k+1} + r_{k+2}$  est la division euclidienne standard.
- (2)  $Z_{k+2} = Z_k - q_{k+1}Z_{k+1}$  soit  $Z_k = q_{k+1}Z_{k+1} + Z_{k+2}$  par définition.

Nous allons essayer de construire deux suites  $(u_k)$  et  $(v_k)$  telles que  $au_k + bv_k = r_k$  car nous aurons alors la relation de Bachet-Bézout  $au_n + bv_n = r_n = \text{pgcd}(a; b)$ . Étant donné ce qui précède, il est maintenant naturel de supposer que  $u_{k+2} = u_k - q_{k+1}u_{k+1}$  et  $v_{k+2} = v_k - q_{k+1}v_{k+1}$ . En effet, ceci nous donne :

$$\begin{aligned} au_{k+2} + bv_{k+2} &= a(u_k - q_{k+1}u_{k+1}) + b(v_k - q_{k+1}v_{k+1}) \\ &= au_k + bv_k - q_{k+1}(au_{k+1} + bv_{k+1}) \\ &= r_k - q_{k+1}r_{k+1} \\ &= r_{k+2} \end{aligned}$$

Il nous reste à trouver les valeurs initiales. Ceci est immédiat puisque nous avons :

- (1)  $au_0 + bv_0 = r_0 = a$  donc  $(u_0; v_0) = (1; 0)$  s'impose.
- (2)  $au_1 + bv_1 = r_1 = b$  donc  $(u_1; v_1) = (0; 1)$  s'impose.

Nous voilà prêts à proposer un algorithme classique et efficace pour déterminer des coefficients de Bachet-Bézout.

#### Algorithme 4 : Classique et efficace

**Donnée :**  $(a; b) \in \mathbb{N}^* \times \mathbb{N}^*$  avec  $b \leq a$

**Résultat :**  $(u; v) \in \mathbb{Z} \times \mathbb{Z}$  tel que  $au + bv = \text{pgcd}(a; b)$

**Actions**

$u' \leftarrow 1; u'' \leftarrow 0$

$v' \leftarrow 0; v'' \leftarrow 1$

**Tant Que**  $b \neq 0$  :

$a = qb + r$  est la division euclidienne standard.

$temp_u \leftarrow u' - qu''; u' \leftarrow u''; u'' \leftarrow temp_u$

$temp_v \leftarrow v' - qv''; v' \leftarrow v''; v'' \leftarrow temp_v$

**Renvoyer**  $(u''; v'')$

---

## 5. AFFAIRE À SUIVRE...

---