VarTable

VarTable est un paquet pour rendre la réalisation des tableaux de signe plus simple
Ce paquet est construit sur Cetz
(version : 0.2.1)

Table des Matières

1 - Introduction

Ce paquet a été réalisé pour rendre la création de tableau de signe plus simple. Pour cela, ce paquet fournis la fonction « `tabvar` », dont les arguments sont décrit dans cette documentation.

Si vous rencontrez un bug, merci de me prévenir via mon GitHub.

P.S : Je sais que mon français n'est pas des plus exellent, donc si cette documentation vous brûle les yeux et que vous avez un peu de temps à perdre, alors vous serez la bien venus pour amélioré cette documentation

Remerciment : Je tiens à remercier supersurviveur et dododu74, pour leur aide au début du projet, (notament la correction des premières documentations)
Ainsi que Akilon27 qui sans lui, les tableaux ne seraits pas aussi customisable.

2 - description générale

tabvar

Retourne un tableau de variation

Parameters

```
tabvar(
  variable: content,
  label: array,
  domain: array,
  contents: array,
  table-style: style,
  nocadre: bool,
  arrow-mark: mark,
  arrow-style: style,
  line-0: bool,
  line-style: style,
  hatching-style: tiling,
  first-column-width: length,
  first-line-height: length,
  element-distance: length,
  values: array,
  fill-color,
  add: content
)
```

variable `content`

`variable` est la variable qui contient la variable du tableau (comme x ou t)
Exemple : si la variable de la fonction est t, alors :

```
variable : $ t $
```

Default: `$ x $`

label `array`

`label` est un array qui contient des array de longueur 2, une pour chaque ligne du tableau, dont le premier élément est le titre de la ligne et le second est le type de la ligne : signe (`s`) ou variation (`v`)
Exemple : pour le tableau de variation de la fonction f, vous devriez écrire :

```
label : (
  ([Signe de $f$], "s"), // la première ligne est un tableau de signe
  ([Variation de $f$], "v") // la seconde ligne est un tableau de variation
)
```

Default: `()`

domain `array`

Les valeurs prises par la variable
par exemple, si votre fonction change de signe ou atteint un extremum pour $x \in \{0, 1, 2, 3\}$
vous devriez écrire :

```
domain: ($0$, $1$, $2$, $3$)
```

Default: `()`

contents `array`

Le contenu de la table
Voir 2.2 pour plus de détails

Default: `((),)`

table-style `style`

Optionelle
Le style de la table
le type style est définis par Cetz, ainsi je vous recommande de vous référer au manuel de Cetz.
Attention : Si vous ne mettez pas le paramètre de style : `mark` a `none`, alors toute les lignes du tableau aurons une tête en flèche

Default: `(stroke: 1pt + black, mark: (symbol: none))`

nocadre `bool`

Pour cacher le cadre externe du tableau

Default: `false`

arrow-mark `mark`

Le style de la tête de flèche.
N.B. le type `mark` est définis par Cetz

Default: `(end: "straight")`

arrow-style `style`

Optionelle :
Le style des flèches.
Attention : le paramètre `mark` est supplanté par le paramètre `arrow-mark`

Default: `(stroke: black + 1pt)`

line-0 `bool`

Optionelle
Si vous voulez changer la bar par défaut dans les tableaux de signe, pour une bar avec un zéro en sont centre

Default: `false`

line-style `style`

Optionelle
Si vous voulez le style de toutes les bars de séparation entre les signes

Default: `(stroke: black + 1pt)`

hatching-style `tiling`

Optionelle
le style des hachures s'il y a des zones hachurées

Default: `tiling(size: (30pt, 30pt))[`

```
    #place(line(start: (0%, 100%), end: (100%, 0%), stroke: 2pt))
    #place(line(start: (-100%, 100%), end: (100%, -100%), stroke: 2pt))
    #place(line(start: (0%, 200%), end: (200%, 0%), stroke: 2pt))
  ]
```

first-column-width `length`

Optionelle
change la largeur de la première colonne

Default: `none`

first-line-height `length`

change la hauteur de la première ligne (celle du domaine et de la variable)

Default: `none`

element-distance `length`

Optionelle
change la distance entre deux éléments

Default: `none`

values `array`

pour ajouter des valeurs entre deux valeurs prédéfinis

Default: `((),)`

fill-color

Optionelle
La couleur de remplissage pour les valeurs sur les flèches (devrait être identique à l'arrière plan du tableau) Par défaut : white

Default: `white`

add `content`

Pour ajouter plus d'éléments via Cetz

Default: `()`

3 - Le paramètre de contenu

Le paramètre contenu est un array avec un élément par ligne (par label).

Chaque éléments sont eux même des arrays avec un élément pour chaque colonne, avec un format différents pour les signes et les variations qui seront détaillés ci-dessous.

3.1 - Le format pour les Signes

Il doit être positioné au même index dans l'array `contents` qu'un label possèdant le string `"s"`, ce qui indique que la ligne doit être considéré comme un tableau de signe

De plus, il doit contenir autant d'éléments que le domaine moins un (un par intervalle), plus un argument optionel si le dernier élément est non défini

Chaque éléments doit être d'une de ces formes :

- `()` - Vide : pour étendre le dernier signe en partant de la gauche sur les intervals marqués vides
- `body` - Le cas basique, constitué du type body de typst, comme `$ + $` ou `$ - $`
- `(style de la barre, body)` - Pour spécifier un style particulier à la barre de **devant** le signe, ce style peut être : `"|"` la barre simple, `"||"` une double barre ou `"0"` pour une barre avec un zéro en sont centre

NB : le paramètre `line-0` change la barre par défaut pour la barre avec un zéro `"0"`.

Vous pouvez mettre en plus à la fin le string `"||"`, pour rajouter un double barre à la toute fin

3.1.1 - Un array classique pour les signes

Un tableau de signe classique :

```
#tabvar(
  init: (
    variable: $t$,
    label: (([signe], "s"),),
  ),
  domain: ($2$, $4$, $6$, $8$),
  contents: (($+$, $-$, $ + $),),
)
```

t	2	4	6	8
signe	+	−	+	

Un exemple plus complexe :

```
#tabvar(
  variable: $t$,
  label: (
    ([signe 1], "s"),
    ([signe 2], "s"),
  ),
  domain: ($ 2 $, $4$, $6$, $8$),
  contents: (
    ("Hello world !", $-$, $ 3 / 2 $),
    ($1/3/9/27$, $+$, "Goodbye word !"),
  ),
)
```

t	2	4	6	8
signe 1	Hello world !	−	$\frac{3}{2}$	
signe 2	$\frac{\frac{1}{3}}{\frac{9}{27}}$	+	Goodbye word !	

Note : Sur le second example, le tableau est comprimé à l'aide de la fonction scale

3.1.2 - Une barre de séparation customisé

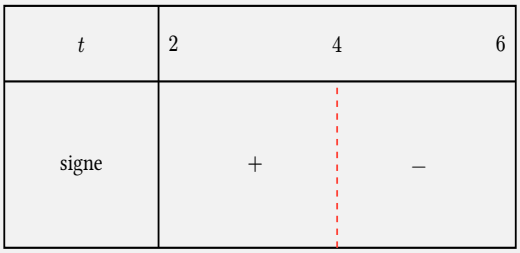
3.1.2.1 - Le style de la barre

Vous pouvez modifier le style de la barre

Le style de la barre est un dictionary, du type `"style"` définis par Cetz.
Pour faire simple, si vous voulez changer uniquement le stroke des barres, vous avez juste a mettre `(stroke: votre stroke)`.
Pour des usages plus complexe référez vous au manuel de Cetz.

Example :

```
#tabvar(
  line-style: (
    stroke: (paint: red, dash: "dashed")
  ),
  variable: $t$,
  label: (([signe], "s"),),
  domain: ($2$, $4$, $6$,),
  contents: (
    ($+$, $-$),
  ),
)
```



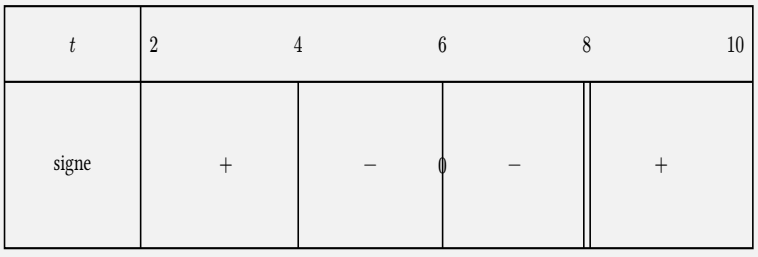
3.1.2.2 - Le type de la barre

Pour tout les signes sauf le premier, au lieux de placé directement un signe, vous pouvez mettre un couple, dont le premier éléments définis le type de la barre placée avant le signe.
Il y a trois type différents de barre :

- `"|"` : une barre simple
- `0` : une barre avec un zéro en sont centre
- `||` une double barre, pour les valeurs non-définis

Exemple

```
#tabvar(
  variable: $t$,
  label: (([signe], "s"),),
  domain: ($2$, $4$, $6$, $8$, $10$),
  contents: (
    (
      $+$,
      ("|", $-$),
      ("0", $-$),
      ("||", $+$)
    ),
  ),
)
```



Si vous voulez avoir une double barre avant le premier signe, vous pouvez utilisez le couple avec en premier éléments `"||"`, à la place du premier signe ; pour mettre une double barre à la fin, ajoutez à la fin de l'array le string `"||"`.

Example :

```
#tabvar(

  variable: $t$,
  label: (([signe], "s"),),

  domain: ($2$, $4$, $6$),
  contents: (
    (
      ("||", $+$),
      $-$,
      "||"
    ),
  ),
)
```

t	2	4	6
sign	+	−	

3.1.3 - Ignorer une valeur

Quand votre tableau de signe possède plus d'un sous tableau, alors vous seriez tanté de vouloir mettre un même signe pour plusieurs valeurs du domaine.
Pour celà c'est assez simple, au lieux de mettre un signe directement, mettez simplement un couple vide `()`

Example :

```
#tabvar(
  line-0: true,
  variable: $t$,
  label: (
    ([signe], "s"),
  ),
  domain: ($2$, $4$, $6$, $8$),
  contents: (
    ($+$, (), $-$),
  ),
)
```

t	2	4	6	8
signe		+	0	−

3.1.4 - Hachurage pour un intervalle non définis

Il se peut que vos fonctions ne soient pas définis sur un ou plusieurs intervalle malheuresement présent dans le domaine du tableau de signe, pour celà la convention veut que l'on hache la zone en question.
Étant donnée que les signes portent sur les intervalles du domaine, il en résulte une syntaxe ralativement simple d'usage, dont on poura distinguer 4 cas :

- le premier cas et le plus courant, celui où les deux bornes de l'intervalle indéfini le sont également, ainsi à la place où vous auriez mis votre signe (ou tout autres éléments), vous renseignerez l'élément suivant : `"|h|"`
- le second cas, relativement présent également, est celui où les deux bornes sont définits, ainsi vous omettrez les deux barres « | » de l'élément présenté ci-dessus, i.e. vous renseignerez `"h"`

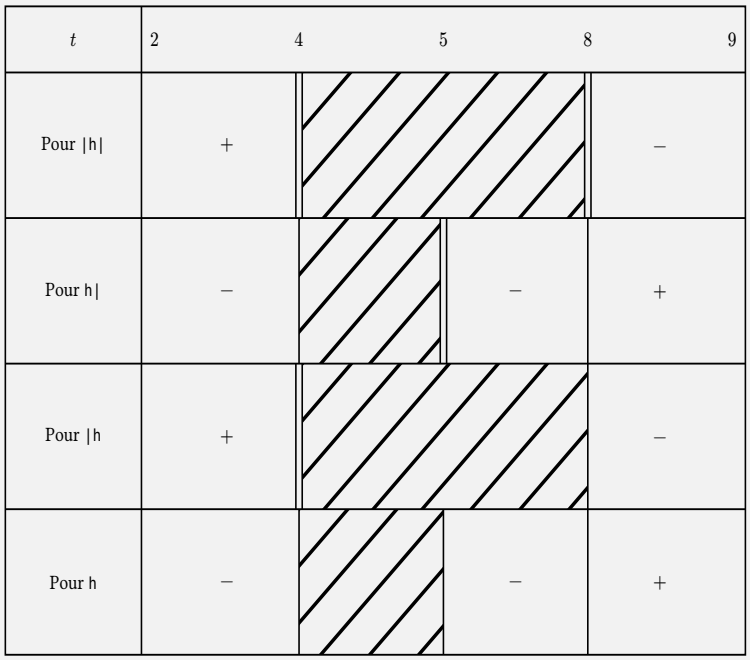
- les deux autres cas, sont celui où seul l'une des deux bornes est définis, ainsi, comme vous l'auriez sans doute compris, retirer (resp. rajouter) la barre pour le côter où l'élément est défini (resp. indéfinis), soit : pour une valeurs définis à gauches `"h|"`; pour une valeur définis à droite `"|h"`

Remarque : Vous avez sans doute compris que la bare « | » symbolise les doubles barres indéfini, de même que le « h » représente le « h » de hachurage, ainis il est naturel de mettre ou non les barres au besoins

Pour étendre le hachurage sur plus d'un des intervalles du domaine, il vous suffit de sauter l'élément suivant avec toujours la même notation, à savoir `()`

Example :

```
  #tabvar(
    variable: $t$,
    label: (
      ([Pour `|h|`], "s"),
      ([Pour `h|`], "s"),
      ([Pour `|h`], "s"),
      ([Pour `h`], "s"),
    ),
    domain: ($ 2 $, $4$, $5$, $8$,
$9$),
    contents: (
      ($+$, "|h|", (), $-$),
      ($-$, "h|", $-$, $+$),
      ($+$, "|h", (), $-$),
      ($-$, "h", $-$, $+$),
    ),
  )
```



3.2 - Le format pour les variations

Il doit être positioné au même index dans l'array `contents` que un label possèdant le string `"v"`, ce qui indique que la ligne doit être considéré comme un tableau de signe

De plus il doit avoir autant d'éléments que le domaine, sinon le programme renverras une erreur.

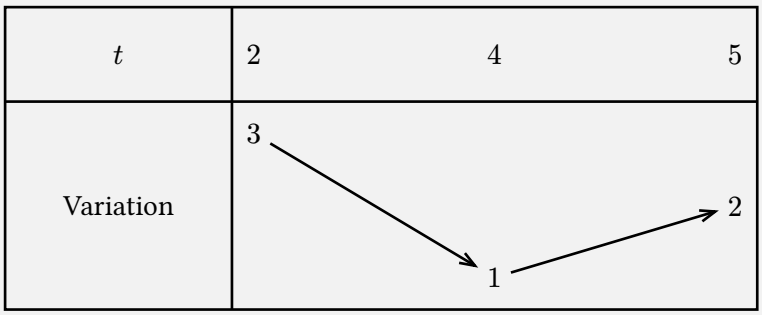
Chacun des éléments qui le compose doit être sous l'une de ces formes :

- `()` - Vide : pour étendre la flèche précédente au prochain élément
- `(position, body)` - Le cas classique, constitué de la position de l'élément (top, center, bottom), et du body
- `(pos1, pos2, "||", body1, body2)` - Le cas où l'élément est non défini
- `(pos, "||", body)` - Une écriture condensé de la forme précédente
- `"h"` ou `"|h"` - La balise de début d'une zone hachurées
- `"H"` ou `"H|"` - La balise de fin d'une zone hachurées

3.2.1 - Un array classique pour les sous tableaux de variation

Un array pour les sous tableaux de variation, doit contenir au moins deux éléments, à savoir la position et l'élément lui-même.
La position peut être 3 élément, à savoir : `top`, `center` et `bottom`, mais ne peut être aucun autre type « alignement »

```
#tabvar(
  variable: $t$,
  label: (
    ([Variation], "v"),
  ),
  domain: ($ 2 $, $4$, $5$),
  contents: (
    (
      (top, $3$),
      (bottom, $1$),
      (center, $2$),
    ),
  ),
)
```



3.2.2 - Les valeurs indéfinis

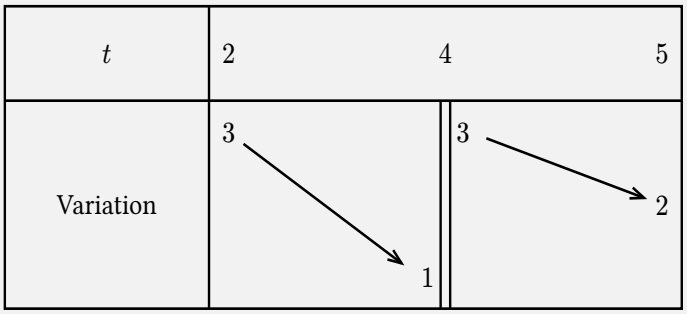
Si votre fonction n'est pas définis en certain points comme $f(x) = \frac{1}{x}$ pour $x = 0$, vous voudrez sans doute mettre une double barre pour signifier que ces valeurs sont indéfini.

⋆ Pour chaque valeurs du domaine excepté le début et la fin :

l'array doit avoir cette forme `(pos1, pos2, "||", élément1, élément2)` où :
- `pos1` et `pos2` sont au choix `top`, `center` ou `bottom`
- `"||"` est là pour spécifier que la valeur est non définis
- `élément1` et 2 est de type `contents` où `élément1` est l'élément avant la barre et `élément2` après

Example :

```
#tabvar(
  variable: $t$,
  label: (
    ([Variation], "v"),
  ),
  domain: ($ 2 $, $4$, $5$),
  contents: (
    (
      (top, $3$),
      (bottom, top, "||", $1$, $3$),
      (center, $2$),
    ),
  ),
)
```

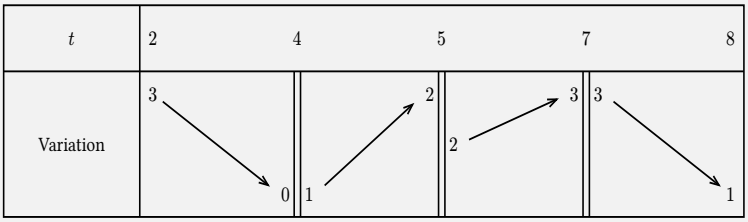


Dans le cas où `pos1` et `pos2` sont identique, alors vous pouvez n'en mettre qu'un seul des deux, de même pour `élément1` et `2`

Example :

À la place de `(top, top, "||", $0$, $0$)`, vous pouvez mettre `(top, "||", $0$)`

```
#tabvar(
  variable: $t$,
  label: (
    ([Variation], "v"),
  ),
  domain: ($ 2 $, $4$, $5$, $7$, $8$),
  contents: (
    (
      (top, $3$),
      (bottom, "||", $0$, $1$),
      (top, center, "||", $2$),
      (top, "||", $3$),
      (bottom, $1$),
    ),
  ),
)
```

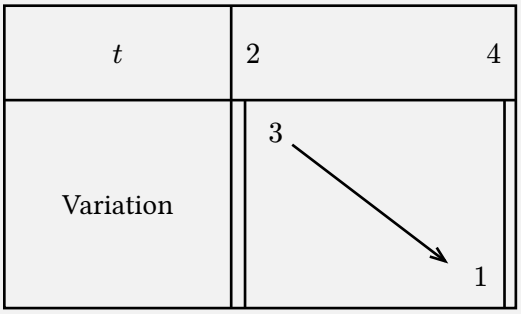


⋆ Pour le début et la fin

Ici comme il n'y a qu'un élément, alors l'array est comme la notation compressé vue précédement, i.e. : `(pos, "||", élément)`

Example :

```
#tabvar(
  variable: $t$,
  label: (
    ([Variation], "v"),
  ),
  domain: ($ 2 $, $4$),
  contents: (
    (
      (top, $3$),
      (bottom, "||", $1$),
    ),
  ),
)
```

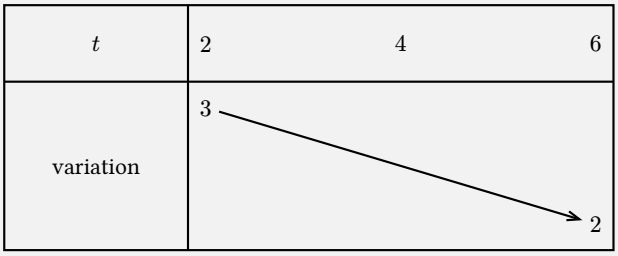


3.2.3 - Ignorer une valeur

Quand vous utilisez plusieurs fonctions dans un même tableau de signe, vous voudriez probablement ignorer certaine valeur du domaine, pour celà, comme pour les sous-tableaux de signe, il suffit de mettre un array vide « `()` »

Example :

```
#tabvar(
  variable: $t$,
  label: (([variation], "v"),),
  domain: ($2$, $4$, $6$),
  contents: (
    (
      (top, $3$),
      (),
      (bottom, $2$),
    ),
  ),
)
```



3.3 - Hachurage pour un intervalle non définis

À la différence des sous-tableaux de signe, ici, les éléments portent sur chacune des valeurs du domaine, et non les intervalles.
Ainsi pour indiquer qu'un certain intervalle est non définit, on utiliseras quatres balises, dont deux « d'ouverture » et deux de « fermeture ».

- Les balise « d'ouverture » sont : `"h"` et `"|h"`, la seconde balise précise que la fonction n'est pas défini pour cette valeur
- Les balise de « fermeture » sont : `"H"` et `"H|"`, la seconde balise précise que la fonction n'est pas défini pour cette valeur

Ainsi il vous suffiras de mettre cette balise entre l'alignement et la valeurs de la fonction,
e.g. `(top, [balise], $3)`

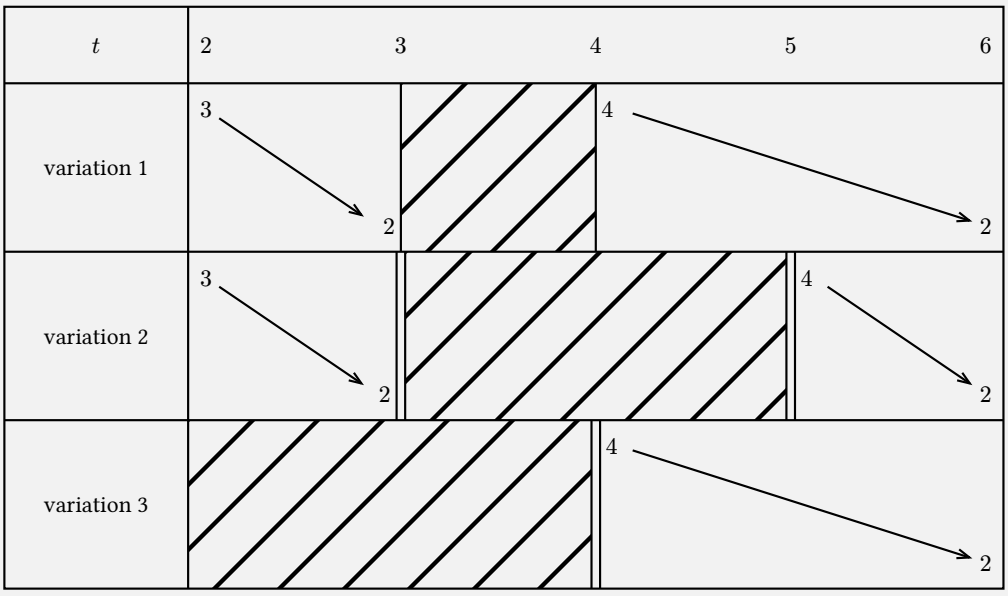
De plus si vous voulez étendre l'hachurage sur plus d'un intervalle, il vous suffie de mettre des array vide entre les deux éléments contenant une balise d'ouverture et de fermeture

Attention :

- les balise « `|h` » et « `H|` », ne sont respectivement pas compatible avec le premier élément et le dernier élément
- Et faites gaffe à ne pas mettre d'éléments non vide entre deux balises, ceci casse le tableau

Example :

```
#tabvar(
  variable: $t$,
  label: (
    ([variation 1], "v"),
    ([variation 2], "v"),
    ([variation 3], "v"),
  ),
  domain: ($2$, $3$, $4$, $5$, $6$),
  contents: (
    (
      (top, $3$),
      (bottom, "h", $2$),
      (top, "H", $4$),
      (),
      (bottom, $2$),
    ),
    (
      (top, $3$),
      (bottom, "|h", $2$),
      (),
      (top, "H|", $4$),
      (bottom, $2$),
    ),
    (
      (bottom, "|h", $2$),
      (),
      (top, "H|", $4$),
      (),
      (bottom, $2$),
    ),
  ),
)
```



4 - Redimensionner le tableau

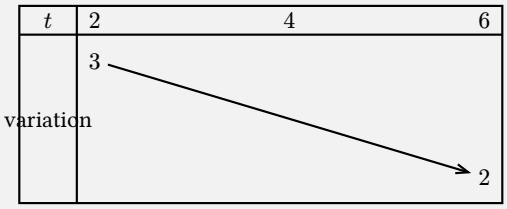
4.1 - Première ligne et colonne

Comme indiqué dans la section 2, il existe deux paramètre effectuant exactement ce qu'il est question ici, i.e. modifier la première ligne et la première colonne.

Ces deux paramètre prenne un type `lenght`, ils doivent être toute fois positif !

Example :

```
#tabvar(
  variable: $t$,
  label: (([variation], "v"),),
  domain: ($2$, $4$, $6$),
  first-column-width: 1cm,
  first-line-height: 5mm,
  contents: (
    (
      (top, $3$),
      (),
      (bottom, $2$),
    ),
  ),
)
```



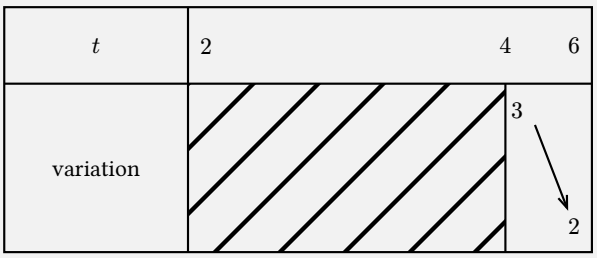
N.B.: Si c'est deux paramètre ne sont pas remplis, alors la hauteur et la largeur se calerons sur la taille du texte contenu,
cependant, si celui-ci est trop petit alors la première colonne feras 30mm le largeur et la première ligne feras 12mm de haut

4.2 - Redimensionner l'espacement entre les élélemts

Pour modifier l'écart entre les éléments du domaines, remplacez l'élément avant l'écart à modifier par un couple de la forme « `(content, lenght)` », où `content` est l'élément du domaine à cette endroit, et `lenght` la distance entre cette élément et le prochain.
Ainsi comme vous l'avez compris le dernier élément ne peut être remplacez par un telle couple.

Example :

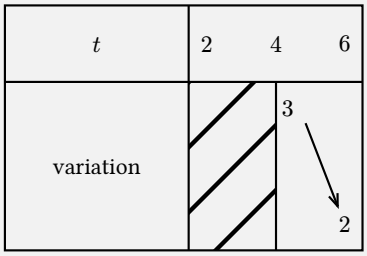
```
#tabvar(
  variable: $t$,
  label: (([variation], "v"),),
  domain: (($2$, 5cm), ($4$, 1cm), $6$),
  contents: (
    (
      "h",
      (top, "H", $3$),
      (bottom, $2$),
    ),
  ),
)
```



Si vous voulez modifier tous les écarts de la même manière, il vous suffie d'utiliser le paramètre `element-distance`

Example :

```
#tabvar(
  variable: $t$,
  label: (([variation], "v"),),
  domain: ($2$, $4$, $6$),
  element-distance: 1cm,
  contents: (
    (
      "h",
      (top, "H", $3$),
      (bottom, $2$),
    ),
  ),
)
```

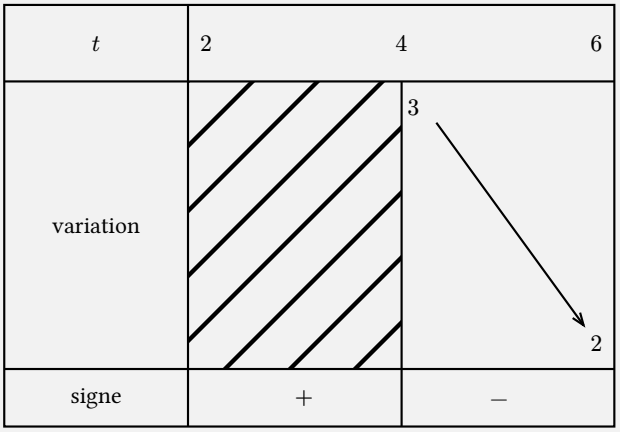


4.3 - Redimensionner la hauteur des sous-tableaux

Pour modifier cette hauteur, rajoutez dans le label, dans l'array correspondant au sous-tableau, entre le content et la balise signe `"s"` ou variation `"v"`, la hauteur que vous souhaitez.
Sachez que par défaut cette hauteur est au minimum : 13,5 mm

Example :

```
#tabvar(
  variable: $t$,
  label: (
    ([variation], 5cm, "v"),
    ([signe], 10mm, "s"),
  ),
  domain: ($2$, $4$, $6$),
  contents: (
    (
      "h",
      (top, "H", $3$),
      (bottom, $2$),
    ),
    ($+$, $-$),
  ),
)
```



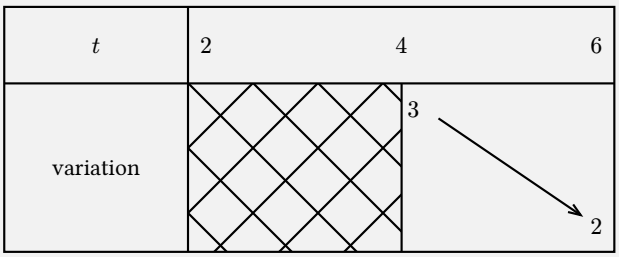
5 - customisation du hachurage

Pour celà il suffie de mettre un objet de type `tiling` au paramètre `hatching-style`

Example :

```
  #tabvar(
    #let exemple = tiling(size: (30pt, 30pt))[
    #place(line(start: (0%, 0%), end: (100%,
100%)))
    #place(line(start: (0%, 100%), end: (100%,
0%)))
  ]

  #tabvar(
    variable: $t$,
    label: (
      ([variation], "v"),
    ),
    domain: ($2$, $4$, $6$),
    hatching-style: exemple,
    contents: (
      (
        "h",
        (top, "H", $3$),
        (bottom, $2$),
      ),
    ),
  )
```



De plus le paquet viens avec sont lot de hachurages prè-définis, fait par Alkion (merci à lui), dont en voici la présentation :

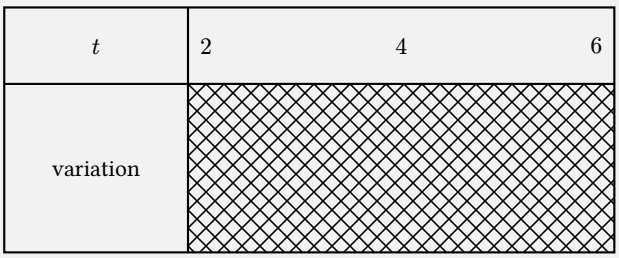
⋆ `grille`

Définition :

```
  #let grille = tiling(size: (8pt, 8pt))[
    #place(line(start: (0%, 0%), end: (100%, 100%), stroke: .7pt))
    #place(line(start: (0%, 100%), end: (100%, 0%), stroke: .7pt))
  ]
```

Example :

```
  #tabvar(
    variable: $t$,
    label: (
      ([variation], "v"),
    ),
    domain: ($2$, $4$, $6$),
    hatching-style: grille,
    contents: (
      (
        "h",
        (),
        (top, "H", $3$),
      ),
    ),
  )
```



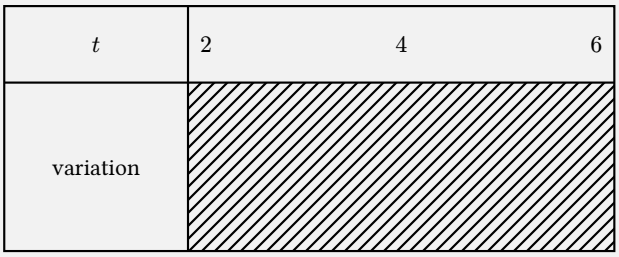
⋆ `nelines`

Définition :

```
#let nelines = tiling(size: (6pt, 6pt))[
  #place(line(start: (100%, 0%), end: (0%, 100%)))
  #place(line(start: (100%, -100%), end: (-100%, 100%)))
  #place(line(start: (200%, 0%), end: (0%, 200%)))
]
```

Example :

```
#tabvar(
  variable: $t$,
  label: (
    ([variation], "v"),
  ),
  domain: ($2$, $4$, $6$),
  hatching-style: nelines,
  contents: (
    (
      "h",
      (),
      (top, "H", $3$),
    ),
  ),
)
```



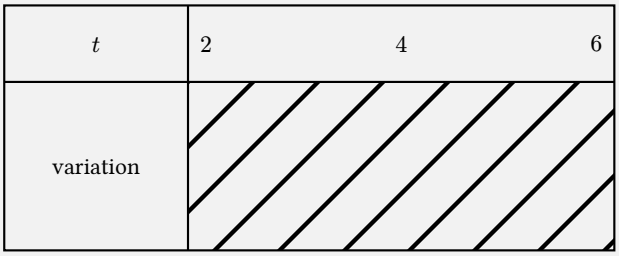
⋆ bignelines

Définition :

```
#let bignelines = tiling(size: (30pt, 30pt))[
  #place(line(start: (100%, 0%), end: (0%, 100%), stroke: 2pt))
  #place(line(start: (100%, -100%), end: (-100%, 100%), stroke: 2pt))
  #place(line(start: (200%, 0%), end: (0%, 200%), stroke: 2pt))
]
```

Example :

```
#tabvar(
  variable: $t$,
  label: (
    ([variation], "v"),
  ),
  domain: ($2$, $4$, $6$),
  hatching-style: bignelines,
  contents: (
    (
      "h",
      (),
      (top, "H", $3$),
    ),
  ),
)
```



⋆ nwlines

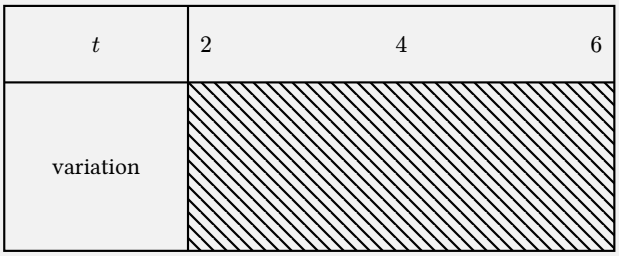
Définition :

```
#let nwlines = tiling(size: (6pt, 6pt))[
  #place(line(start: (0%, 0%), angle: 45deg))
  #place(line(start: (-100%, 0%), angle: 45deg))
```

```
    #place(line(start: (100%, 0%), end: (200%, 100%)))
    #place(line(start: (100%, -100%), angle: -135deg))
  ]
```

Example :

```
#tabvar(
  variable: $t$,
  label: (
    ([variation], "v"),
  ),
  domain: ($2$, $4$, $6$),
  hatching-style: nwlines,
  contents: (
    (
      "h",
      (),
      (top, "H", $3$),
    ),
  ),
)
```



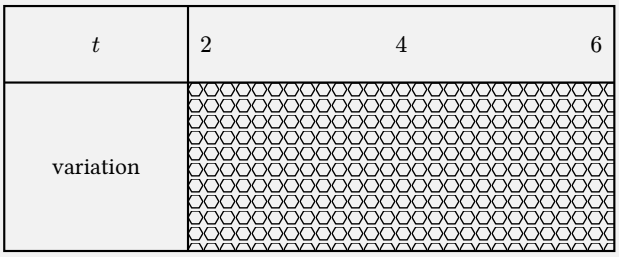
⋆ `hatch`

Définition :

```
#let hatch = tiling(size: (7pt, 7pt))[
  #place(polygon.regular(vertices: 6, size: 6.5pt, stroke: .6pt))
]
```

Example :

```
#tabvar(
  variable: $t$,
  label: (
    ([variation], "v"),
  ),
  domain: ($2$, $4$, $6$),
  hatching-style: hatch,
  contents: (
    (
      "h",
      (),
      (top, "H", $3$),
    ),
  ),
)
```



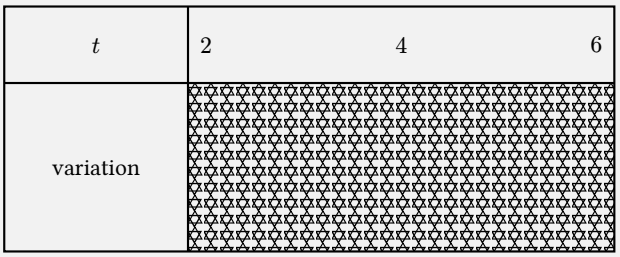
⋆ `etoile`

Définition :

```
#let etoile = tiling(size: (7pt, 7pt))[
  #place(rotate(180deg, origin: center + horizon)[#polygon.regular(vertices: 3,
size: 6.5pt, stroke: .6pt)])
  #place(polygon.regular(vertices: 3, size: 7pt, stroke: .5pt))
]
```

Example :

```
#tabvar(
  variable: $t$,
  label: (
    ([variation], "v"),
  ),
  domain: ($2$, $4$, $6$),
  hatching-style: etoile,
  contents: (
    (
      "h",
      (),
      (top, "H", $3$),
    ),
  ),
)
```

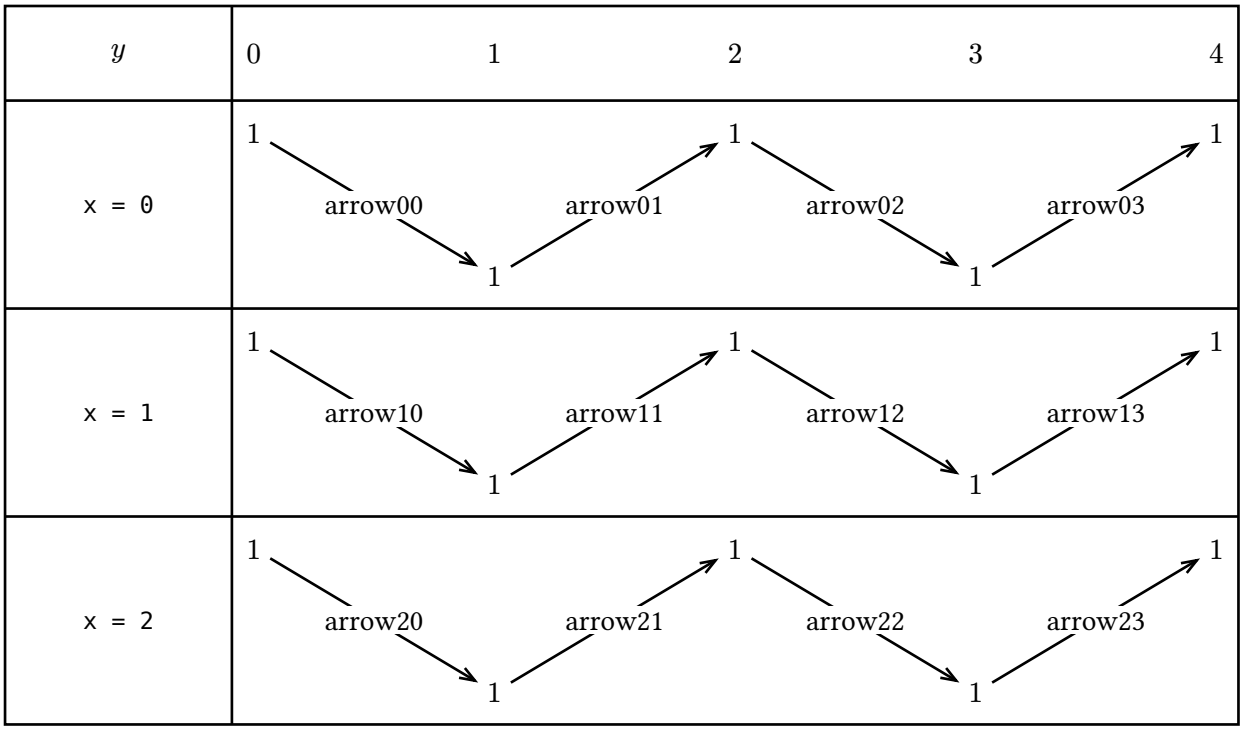


6 - Ajouter des valeurs dans les sous-tableaux de variation

Il est possible en effet d'ajouter des valeurs dans les sous-tableaux de variation, sans alonger le domaine.
Ce cas peut être utile à ceux qui voudrais explicité sur leur tableaux une valeurs précise prise par vorte fonction du au théorème des valeurs intermédiaires.

C'est ici que l'argument `values` sert, en effet vous allez mettre dans `values` autant de valeurs que vous voulez ajouter que vous voulez.
Les éléments que vous ajoutez doivent avoir cette forme : `("arrowxy", content1, content2)`, où :

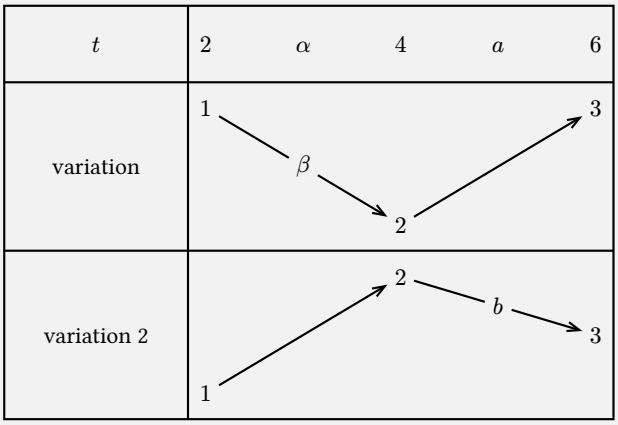
- `x` et `y` dans `"arrowxy` sont les coordonnée de la flèche sur la quel vous voulez ajouter une valeur, ces coordonées commence en haut à gauche par `x = 0, y = 0`



- `content1` le content qui seras placé au niveau du domain
- `content2` le content qui seras placé sur la flèche

Example :

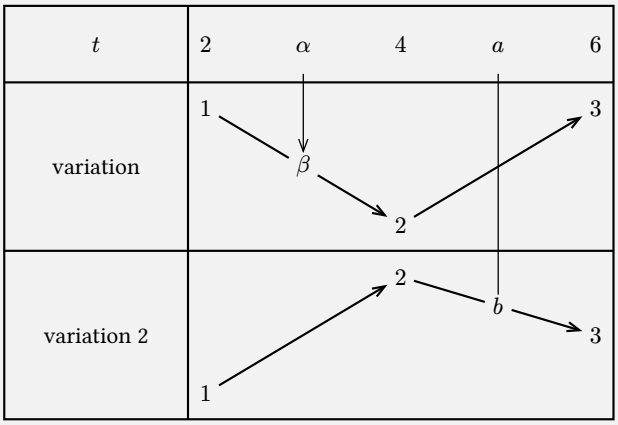
```
#tabvar(
  fill-color: luma(95%),
  variable: $t$,
  label: (
    ([variation], "v"),
    ([variation 2], "v"),
  ),
  domain: ($2$, $4$, $6$),
  contents: (
    (
      (top, $1$),
      (bottom, $2$),
      (top, $3$),
    ),
    (
      (bottom, $1$),
      (top, $2$),
      (center, $3$),
    ),
  ),
  values: (
    ("arrow00", $alpha$, $beta$),
    ("arrow11", $a$, $b$),
  ),
)
```



De plus, il est possible d'ajouter une flèche ou une ligne joignant la valeur dans le domaine et celle sur la flèche

Example :

```
#tabvar(
  fill-color: luma(95%),
  variable: $t$,
  label: (
    ([variation], "v"),
    ([variation 2], "v"),
  ),
  domain: ($2$, $4$, $6$),
  contents: (
    (
      (top, $1$),
      (bottom, $2$),
      (top, $3$),
    ),
    (
      (bottom, $1$),
      (top, $2$),
      (center, $3$),
    ),
  ),
  values: (
    ("arrow00", $alpha$, $beta$, "f"),
    ("arrow11", $a$, $b$, "l"),
  ),
)
```



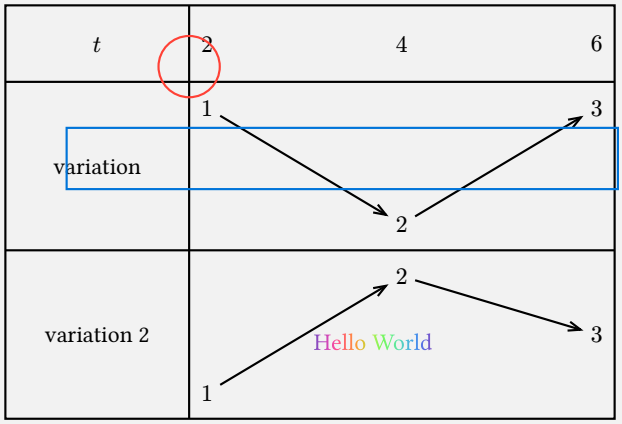
7 - ajouter ce que vous voulez avec `add`

Il est en effet possible d'ajouter autemps d'éléments que vous voulez (tant que cetz le peut) à vos tableaux, pour celà il suffie d'ajouter ces éléments dans le paramètre `add`

Attention : Pour ajouter des éléments propre à Cetz, comme `content, rect, etc` vous devez y importé dans votre fichier Cetz

Example :

```
#tabvar(
  variable: $t$,
  label: (
    ([variation], "v"),
    ([variation 2], "v"),
  ),
  domain: ($2$, $4$, $6$),
  contents: (
    (
      (top, $1$),
      (bottom, $2$),
      (top, $3$),
    ),
    (
      (bottom, $1$),
      (top, $2$),
      (center, $3$),
    ),
  ),
  add: {
    cetz.draw.circle((3, -1), stroke: red, radius: 5mm)
    cetz.draw.content((6, -5.5), text(fill: gradient.linear(..color.map.rainbow))
[Hello World])
    cetz.draw.rect((1, -2), (10, -3), stroke: blue)
  },
)
```



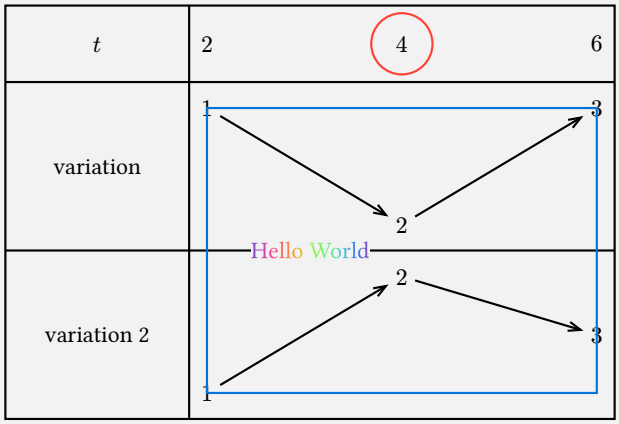
Pour simplifier le procésus chaque éléments du tableau possède un « nom » qui permet, par le système de coordonnée de Cetz, d'attacher les éléments ajoutés au éléments déjà présent.
Voici un tableau qui résume les noms :

éléments	nom	précision
la variable	`var`	
domaine	`domainx`	x représente le x-ième élément du domain
label	`labely`	y représente le y-ième label
ligne entre les sous-tableaux	`line-betwen-table-nby`	y représente la y-ième ligne note : la ligne 0 sépare le domaine du reste
cadre	`cadre`	toujours utilisable même avec `nocadre` mis à `true`
ligne entre les labels et les sous-tableaux	`line-separating-labels-tables`	

ligne passant au niveau du domaine, centré	`line-centred-domain`	cette ligne n'est pas visible
flèches dans les sous-tableaux de variation	`arrowxy`	sont exactement ceux rencontré dans la section 6 si référer pour plus de précision
les éléments dans un sous-tableau de variation	`variationxy`	`x` et `y` sont les coordonnées de l'élément
les éléments dans un sous-tableau de signe	`signxy`	`x` et `y` sont les coordonnées de la barre. fonctionne de la même manière que pour `arrowxy`.
le hachurage	`hatchingxy`	`x` et `y` sont les coordonnée du hachurage
l'élément dans le domaine, pour une valeur ajouter	`depart_valuesx`	`x` est le x-ième élément ajouter
l'élément dans le sous-tableau de variation, pour une valeur ajouter	`fin_valuesx`	`x` est le x-ième élément ajouter

1er Example :

```
#tabvar(
  variable: $t$,
  label: (
    ([variation], "v"),
    ([variation 2], "v"),
  ),
  domain: ($2$, $4$, $6$),
  contents: (
    (
      (top, $1$),
      (bottom, $2$),
      (top, $3$),
    ),
    (
      (bottom, $1$),
      (top, $2$),
      (center, $3$),
    ),
  ),
  add: {
    import cetz.draw: *
    circle("domain1", stroke: red, radius: 5mm)
    content(
      "line-betwen-table-nb1",
      text(fill: gradient.linear(..color.map.rainbow))[Hello World],
      frame: "rect",
      fill: luma(95%),
      stroke: none,
    )
    rect("variation02", "variation10", stroke: blue)
  },
)
```



2ième Example :

```
#tabvar(
  variable: $t$,
  label: (
    ([variation], "v"),
    ([variation 2], "v"),
  ),
  domain: ($2$, $4$, $6$),
  contents: (
    (
      (top, $1$),
      (bottom, $2$),
      (top, $3$),
    ),
    (
      (bottom, $1$),
      (top, $2$),
      (center, $3$),
    ),
  ),
  add: {
    import cetz.draw: *
    polygon("line-centred-domain.82%", 5, stroke: red, radius: 5mm)
    line("var", "label1", stroke: blue)
    image("")
  },
)
```