

L'extension *intexgral*

v4.1.0

Valentin Dao*

28 juillet 2026

Résumé

Tandis que la composition d'intégrales est très fréquente en $\text{\LaTeX}$, cette dernière s'avère souvent peu pratique. Lorsque l'expression se complexifie, il devient alors difficile d'en modifier les différents éléments dans un code source peu lisible. Pour répondre à ce problème, le package *intexgral* met à disposition une macro centrale dont le seul argument est l'intégrande. Tout le reste (les symboles, les bornes, les variables d'intégration...) pourra aisément être modifié grâce à une interface $\langle \text{clé}=\text{valeur} \rangle$. Ce package étant écrit en expl3 , il dépend donc de *l3kernel*.

*E-mail : vdao.texdev@gmail.com

Table des matières

Résumé	1
I Documentation	4
1 Options de package	5
2 Présentation de la macro principale	5
3 Liste des clés	6
3.1 Bornes d'intégration	6
3.1.1 Définir les bornes	6
3.1.2 Modifier le style	7
3.2 Modes d'affichage	7
3.3 Symbole (et encore des bornes)	8
3.3.1 Sélectionner le symbole	8
3.3.2 Générer plusieurs intégrales	9
3.3.3 Définir des bornes ponctuellement	9
3.3.4 Motifs récurrents de bornes	9
3.4 Différentielles	10
3.4.1 Spécifier les variables d'intégration	10
3.4.2 Inclure le jacobien	10
3.4.3 Modifier le symbole des différentielles	11
3.4.4 Réaliser une intégrale curviligne	11
3.4.5 Ajouter un exposant	11
4 Macros annexes	11
4.1 Emplacements des différentielles	12
4.1.1 Permuter la position	12
4.1.2 Choisir une position personnalisée	12
4.2 Retour sur la clé <i>limits</i>	13
4.2.1 Définir un mot-clé	13
4.2.2 Saisir des bornes symétriques	14
4.3 Retour sur la clé <i>variables</i>	15
4.3.1 Définir un mot-clé	15
4.4 Retour sur la clé <i>symbol</i>	16
5 Syntaxe spéciale	17
5.1 Présentation de la syntaxe	17
5.2 Fonctionnement de la syntaxe	18
6 Paramètres annexes	19
Historique des modifications	21
II Implémentation	22
1 Initialisation	23
1.1 Déclaration du package	23

1.2	Vérifications préliminaires	23
2	Options de package	24
2.1	Définition des variables	24
2.2	Déclaration des options	24
3	Messages	25
4	Variantes utiles	27
5	Variables internes	27
5.1	Tokens	27
5.2	Booléens	28
5.3	Séquences et clists	29
5.4	Liste de propriétés	29
5.5	Muskip	29
5.6	String	30
6	Macros et séquences auxiliaires	30
6.1	Séquences principales	31
6.2	Séquences subsidiaires	31
7	Bornes d'intégration	32
8	Variables	35
9	Syntaxe spéciale	36
10	Composition de l'intégrale	38
10.1	Mode <i>default</i>	39
10.2	Mode <i>nested</i>	41
10.3	Mode <i>product</i>	42
10.4	Composition finale	43
11	Déclaration des clés	43
12	Macros d'interface utilisateur	47
12.1	Mots-clés pour les bornes	47
12.2	Mots-clés pour les variables	48
12.3	Mots-clés pour les symboles	49
12.4	Macros de configuration	50
13	Configuration du package	51
13.1	Symbole	51
13.2	Bornes	51
13.3	Variables	52
13.4	Paramètres par défaut	52
	Index	53

Licence

© 2025 Valentin Dao, publié sous la ~~La~~TeX Project Public License (LPPL) 1.3c

Polices

Chronicle Text G3 Roman

© 2002, 2007 Hoefler & Frere-Jones.

Whitney-Medium

© 1996, 2009 Hoefler & Frere-Jones.

Monospace Argon Medium

© 2023, GitHub

Dépôt

 [Voir dépôt GitHub.](#)

Remerciements

Un grand merci à Plante et Slurpy de m’avoir accompagné dans cette aventure que fut l’apprentissage de $\text{\TeX}$, vos conseils ont été précieux. Je tiens également à remercier Anthony pour avoir toujours gentiment révisé les nouvelles versions et m’avoir donné des idées pour celles à venir.

PREMIÈRE PARTIE

DOCUMENTATION

1 Options de package

`\intexgralsetup` `\intexgralsetup` {<options de package>}

nouveau: v2.0.0

Ces options peuvent être déclarées de façon classique avec `\usepackage` ou bien avec `\intexgralsetup` dans le préambule.

`invert-limits` `invert-limits=<true | false>`

Cette clé permet d'inverser la convention d'ordre des limites expliquée plus tard. Le document ne peut suivre qu'une seule convention.

`invert-diff` `invert-diff=<true | false>`

mise à jour: v3.0.0

En sciences physiques, il est courant de voir les variables d'intégration placées avant l'intégrande. Cette clé intervertit donc leurs positionnements.

`limits-mode` `limits-mode=<limits | nolimits>`

nouveau: v3.0.0

Applique `\limits` ou `\nolimits` à toutes les intégrales.

`italic` `italic=<true | false>`
`upright` `upright=<true | false>`

mise à jour: v4.0.0

Applique un «d» droit ou italique pour les différentielles.

Remarque : depuis la version v4.0.0, le package *derivative* n'est plus utilisé pour gérer les différentielles. Le nombre d'options y est donc plus restreint mais devrait convenir à la plupart des cas d'usage dans le cadre des intégrales.

2 Présentation de la macro principale

`\integral` `\integral` [<liste de clés>] {<intégrande>}

mise à jour: v3.0.0

Cette macro est celle qui permet de composer les intégrales. Elle doit naturellement être utilisée en mode mathématique uniquement. En voici un premier exemple sous sa forme la plus simple :

```
\begin{equation}
\integral{x}
\end{equation}
```

$$\int x \, dx \tag{1}$$

Puisque cette macro est axée autour de l'utilisation de clés, la première partie de cette documentation les présentera en les regroupant par domaines. La seconde partie quant à elle introduira les quelques macros annexes qui viennent compléter l'usage de certaines clés. Le reste de ce document exposera les autres fonctionnalités du package.

3 Liste des clés

3.1 Bornes d'intégration

3.1.1 Définir les bornes

limits `limits= {⟨liste-mixte⟩}, ⟨mot-clé⟩`
limits* `limits*= {⟨liste-mixte⟩}, ⟨mot-clé⟩`

Cette clé détermine les bornes d'intégration à utiliser. Sous sa forme la plus simple, la valeur de la clé prend comme argument une liste d'éléments séparés par une virgule (*comma-separated values* ou `⟨csv-list⟩`). Cette simplification de la saisie impose notamment de fixer une convention : le premier et deuxième élément de la liste correspondront à la borne inférieure et supérieure respectivement. Cette convention peut être inversée à l'aide d'`invert-limits` vu en section 1.

```
\begin{equation}
  \integral[limits={1, 10}]{f(x)}
\end{equation}
```

$$\int_1^{10} f(x) \, dx \quad (2)$$

L'avantage notable de la clé `limits` est qu'on peut lui spécifier les bornes de plusieurs intégrales, et le package compose automatiquement les symboles correspondants. Pour séparer les paires de bornes, il faut utiliser le point-virgule (*semicolon-separated values* ou `⟨ssv-list⟩`¹).

```
\begin{equation}
  \integral[limits={1, 2; 3, 4}, variables={x, y}]{f(x, y)}
\end{equation}
```

$$\int_1^2 \int_3^4 f(x, y) \, dx \, dy \quad (3)$$

Il est également possible de renseigner des bornes prédéfinies à l'aide de mots-clés (voir sous-sous-section 4.2.1). Enfin, la variante étoilée permet d'exprimer les bornes de l'intégrale sous forme d'intervalle. Le cas échéant, le sens des crochets s'adapte automatiquement à la présence de `±\infty`.

```
\begin{equation}
  \integral[limits*={1, 10}]{f(x)}
\end{equation}
```

1. On entend donc par `⟨liste-mixte⟩` le fait que `limits` puisse accepter un ensemble de `⟨csv-list⟩` dans une `⟨ssv-list⟩` plus globale.

$$\int_{[1,10]} f(x) \, dx \quad (4)$$

3.1.2 Modifier le style

limits-mode `limits-mode=\langle \text{limits} | \text{nolimits} \rangle`

nouveau: v4.1.0 Clé homonyme de celle vue en section 1, qui permet de modifier le style ponctuellement plutôt que globalement.

3.2 Modes d’affichage

mode `mode=\langle \text{default} | \text{nested} | \text{product} \rangle`

nouveau: v3.0.0 *Uniquement* lorsque la clé `limits` est employée, il est possible d’alterner entre trois styles d’affichage distincts. Chacun d’entre eux est entièrement compatible avec l’option `invert-diff`.

default Compose l’ensemble des symboles, puis l’intégrande, puis les variables d’intégration. La macro opérant par défaut dans ce mode, il n’est donc pas nécessaire d’utiliser la clé avec cette valeur.

nested Compose une intégrale *imbriquée* où symbole et intégrande alternent avant que toutes les variables d’intégration soient écrites.

product Compose un produit d’intégrales où symbole, intégrande et variables d’intégration alternent.

Bien sûr, rien ne vous empêche pour le mode produit d’utiliser plusieurs fois la macro `\integral` d’affilée pour produire le même résultat que dans l’exemple ci-dessous. Pour celles et ceux qui préféreraient cependant obtenir le résultat avec une seule macro, cette méthode est permise.

Important : pour les deux derniers modes, l’intégrande devra être *découpée*. Afin de correctement effectuer cette action, il faut avoir recours à la même méthode qu’avec `limits`, c’est-à-dire en employant le point-virgule.

MODE **default**

```
\begin{equation}
  \integral[limits={1, 2; 3, 4; 5, 6}, variables={x, y, z}, mode=default]{xyz}
\end{equation}
```

$$\int_1^2 \int_3^4 \int_5^6 xyz \, dx \, dy \, dz \quad (5)$$

MODE **nested**

```
\begin{equation}
  \integral[limits={1, 2; 3, 4; 5, 6}, variables={z, y, x}, mode=nested]{x;y;z}
\end{equation}
```

$$\int_1^2 x \int_3^4 y \int_5^6 z \, dz \, dy \, dx \quad (6)$$

MODE **product**

```
\begin{equation}
  \integral[limits={1, 2; 3, 4; 5, 6}, variables={x, y, z}, mode=product]{x;y;z}
\end{equation}
```

$$\int_1^2 x \, dx \int_3^4 y \, dy \int_5^6 z \, dz \quad (7)$$

Pour bien fonctionner, il est évident que les clés `limits` et `variables` (voir 3.4.1), ainsi que l'intégrande, doivent avoir le même nombre d'éléments.

3.3 Symbole (et encore des bornes)

Les clés `limits(*)` se révèlent très utiles lorsque l'on souhaite composer une intégrale définie. Néanmoins, cela couvre seulement les expressions finales où les bornes de chaque variable ont été spécifiées. Pour des cas plus généraux (les intégrales indéfinies donc) elles sont relativement malcommodes. Par exemple, pour composer une intégrale double sur une surface S , on devrait écrire `limits={, ; S, }`. Il va sans dire que c'est assez mal adapté pour l'utilisateur et peu optimal pour le package. Par ailleurs, le glyphe ne serait pas correct. L'ensemble des clés à suivre propose donc une façon de facilement modifier le symbole ainsi que les bornes.

3.3.1 Sélectionner le symbole

`symbol`

`symbol`=⟨séquence de contrôle⟩

mise à jour: v3.0.0

Cette clé accepte une macro désignant un symbole d'intégration. Tout symbole préalablement défini par une séquence de contrôle est recevable. Si vous tentez d'en utiliser une qui n'est pas définie, elle sera substituée à `\int` et un message d'avertissement sera émis.

Remarque : à part la vérification de l'existence de la macro, aucun contrôle particulier n'est effectué et la valeur de la clé est utilisée telle quelle pour composer le symbole. Ceci implique notamment que le symbole va dépendre de la façon dont il est défini. Par exemple, `amsmath` et `unicode-math` ne définissent pas `\iint` de la même manière. Le résultat sera donc naturellement différent : l'un colle deux `\int` ensemble avec un certain crénage pour définir `\iint`, tandis que l'autre définit le vrai glyphe U+222C.

```
\begin{equation}
  \integral[symbol=\iint, llimit=S, variables={x, y}]{f(x, y)}
\end{equation}
```

$$\iint_S f(x, y) \, dx \, dy \quad (8)$$

3.3.2 Générer plusieurs intégrales

nint `nint=<entier>`

Cette clé accepte un entier n qui permet de composer n intégrales. Il est conseillé d’avoir recours à cette clé seulement si le nombre de symboles dépasse 4 afin de privilégier les glyphes définis par la police mathématique utilisée.

```
\begin{equation}
  \integral[nint=5, llimit=\omega, variables={x_1, x_2, x_3, x_4, x_5}]{f(x_1, x_2,
    x_3, x_4, x_5)}
\end{equation}
```

$$\iiint\limits_{\Omega} f(x_1, x_2, x_3, x_4, x_5) dx_1 dx_2 dx_3 dx_4 dx_5 \quad (9)$$

3.3.3 Définir des bornes ponctuellement

llimit `llimit=<borne inférieure>`
ulimit `ulimit=<borne supérieure>`

mise à jour: v3.0.0

Ces deux clés permettent de spécifier les bornes inférieures et supérieures respectivement. Elles ne sont adaptées que si un seul symbole est affiché. Si vous avez besoin de spécifier les deux bornes, la clé `limits` devra être privilégiée.

```
\begin{equation}
  \integral[llimit={x^2 + y^2 \leq 1}, variables={x, y}]{f(x, y)}
\end{equation}
```

$$\int_{x^2+y^2 \leq 1} f(x, y) dx dy \quad (10)$$

3.3.4 Motifs récurrents de bornes

Les bornes suivent parfois des schémas courants qui peuvent se généraliser avec des clés, évitant ainsi de surcharger l’argument de `llimit`.

domain `domain={(*-liste)}`
domain* `domain*={(*-liste)}`

mise à jour: v3.0.0

Ces clés acceptent une liste dont le délimiteur est un astérisque. Ensuite, chaque élément de la liste est analysé de la façon suivante :

- Le premier token de l’item se voit passer `\mathbb` (précédé d’`\uppercase` au cas où la touche SHIFT serait trop loin pour vos doigts).
- Les tokens restants, qui peuvent être vides, sont placés comme exposant (ou en indice pour la variante étoilée de la clé).

```
\begin{equation}
  \integral[symbol=\iint, domain={r*r}, variables={x, y}]{xy}
\end{equation}
```

$$\iint_{\mathbb{R} \times \mathbb{R}} xy \, dx \, dy \quad (11)$$

boundary `boundary= {⟨borne inférieure⟩}`

Cette clé place simplement le symbole ∂ avant la borne inférieure.

```
\begin{equation}
\int_{\partial S} G(\vec{r}) \cdot d\vec{r}
\end{equation}
```

$$\oint_{\partial S} G(\vec{r}) \cdot d\vec{r} \quad (12)$$

3.4 Différentielles

3.4.1 Spécifier les variables d'intégration

variables `variables= {⟨liste-csv⟩}, ⟨mot-clé⟩, none`

mise à jour: v3.0.0

Cette clé permet de définir les variables d'intégration sous forme d'une `⟨csv-list⟩`. La clé peut, tout comme `limits`, accepter un mot-clé comme argument. Ce comportement est aussi expliqué plus tard (voir sous-sous-section 4.3.1).

Remarque : si aucune variable n'est renseignée, c'est-à-dire que `variables` n'est pas appelée, le package place automatiquement « dx » (ou « $d\vec{r}$ » si `diff-vec` est actif). De plus, si `none` est passé comme valeur, aucune variable ne sera affichée.

```
\begin{equation}
\int_{variables=t} t^2
\end{equation}
```

$$\int t^2 \, dt \quad (13)$$

3.4.2 Inclure le jacobien

jacobian `jacobian`

Cette clé active l'affichage du jacobien lorsque défini par `\NewVariableKeyword`.

```
\begin{equation}
\int_{limits={0, R; 0, 2\pi; 0, \pi}} r^2 \, dr \int_0^{2\pi} \sin \theta \, d\theta \int_0^\pi d\phi
\end{equation}
```

$$\int_0^R r^2 \, dr \int_0^{2\pi} \sin \theta \, d\theta \int_0^\pi d\phi \quad (14)$$

3.4.3 Modifier le symbole des différentielles

diff-symb `diff-symb=<séquence de contrôle>`

Cette clé permet de modifier le symbole des différentielles.

```
\begin{equation}
  \int e^{+\frac{i}{\hbar} S[q(t)]} \mathcal{D}q(t)
\end{equation}
```

$$\int e^{+\frac{i}{\hbar} S[q(t)]} \mathcal{D}q(t) \quad (15)$$

3.4.4 Réaliser une intégrale curviligne

diff-vec `diff-vec`

Cette clé applique un vecteur à la variable d'intégration en plus d'un point médian. Elle n'est compatible qu'avec le mode `default`. L'utiliser avec les options `nested` ou `product` n'aura aucun effet.

```
\begin{equation}
  W = \int_C \vec{F} \cdot d\vec{s}
\end{equation}
```

$$W = \int_C \vec{F} \cdot d\vec{s} \quad (16)$$

3.4.5 Ajouter un exposant

diff-order `diff-order=<liste-csv>`

Cette clé place un exposant à chaque différentielle.

```
\begin{equation}
  S[\phi] = \int \mathcal{L}(\phi, \partial_\mu \phi, x^\mu) d^4x
\end{equation}
```

$$S[\phi] = \int \mathcal{L}(\phi, \partial_\mu \phi, x^\mu) d^4x \quad (17)$$

4 Macros annexes

En plus de la large gamme de clés définies par le package, quelques macros viennent également améliorer leurs usages.

4.1 Emplacements des différentielles

4.1.1 Permuter la position

`\invertdiff` `\invertdiff`

nouveau: v4.1.0 Cette macro inverse le statut de l'option `invert-diff`, qu'elle soit initialisée à `true` ou `false`.

```
\invertdiff
\begin{equation}
  \integral{f(x)}
\end{equation}
\invertdiff
\begin{equation}
  \integral[variables=\omega]{f(\omega)}
\end{equation}
```

$$\int dx f(x) \tag{18}$$

$$\int f(\omega) d\omega \tag{19}$$

4.1.2 Choisir une position personnalisée

`\differentials` `\differentials`

Bien que l'option `invert-diff` existe, il est souvent souhaitable de pouvoir placer les différentielles où l'on veut. Par exemple, il est fréquent de les voir au numérateur d'une fraction lorsque celui-ci vaut 1. Pour répondre à ce besoin, le package met à disposition la macro `\differentials`, dont le fonctionnement varie légèrement d'un mode d'affichage à un autre :

- `default` : compose toutes les différentielles d'un coup. Fonctionne avec `invert-diff/\invertdiff`.
- `nested` : compose toutes les différentielles d'un coup. La macro *n'est pas* définie si `invert-diff/\invertdiff` est actif.
- `product` : compose une seule différentielle à la fois. Il faudra donc la répéter entre chaque point-virgule. Fonctionne aussi avec `invert-diff/\invertdiff`.

```
\begin{gather}
  \integral{\frac{\differentials}{x}}{\mid 1cm}
  \langle 0 \mid T \{ \phi(x) \phi(y) \} \mid 0 \rangle = \integral[diff-order=4,
    variables=p]{\frac{\differentials}{(2\pi)^4} \frac{e^{-i p \cdot (x-y)}}{p^2 - m
    ^2 + i \epsilon}}
\end{gather}
```

$$\int \frac{dx}{x} \quad (20)$$

$$\langle 0 | T\{\phi(x)\phi(y)\} | 0 \rangle = \int \frac{d^4p}{(2\pi)^4} \frac{e^{-ip \cdot (x-y)}}{p^2 - m^2 + i\epsilon} \quad (21)$$

4.2 Retour sur la clé *limits*

4.2.1 Définir un mot-clé

```
\NewLimitsKeyword \NewLimitsKeyword {<mot-clé>} {<bornes>}
\RenewLimitsKeyword
\ProvideLimitsKeyword
\DeclareLimitsKeyword
```

mise à jour: v4.0.0

Il est courant de devoir renseigner les mêmes bornes dans une intégrale. Pour faciliter leur écriture, les clés `limits(*)` acceptent, en plus des bornes explicites, un mot-clé en désignant une paire prédéfinie par l'une de ces quatre macros :

- `\NewLimitsKeyword` Crée un nouveau mot-clé et émet une erreur s'il existe déjà.
- `\RenewLimitsKeyword` Redéfinit un mot-clé et émet une erreur s'il n'existe pas déjà.
- `\ProvideLimitsKeyword` Ne crée un nouveau mot-clé que s'il n'existe pas déjà. Aucun message ne sera émis le cas échéant.
- `\DeclareLimitsKeyword` Crée un nouveau mot-clé quoi qu'il en soit, écrasant toute définition préalable.

La saisie d'un nouveau mot-clé devra se faire dans le respect de la convention d'ordre des bornes. Vous pouvez ainsi écrire :

```
\usepackage{intexgral}
\NewLimitsKeyword{key1}{A, B}
\intexgralsetup{invert-limits=true}
\NewLimitsKeyword{key2}{B, A}
```

`key1` et `key2` produiront alors la même intégrale. Voici la liste des mots-clés déjà définis par le package et les bornes qu'ils contiennent :

<code>ab</code>	a, b
<code>unit</code>	$0, 1$
<code>real</code>	$-\infty, +\infty$
<code>positive</code>	$0, +\infty$
<code>negative</code>	$-\infty, 0$
<code>circle</code>	$0, 2\pi$
<code>scircle</code>	$0, \pi$
<code>qcircle</code>	$0, \pi/2$
<code>height</code>	$0, H$

radius 0, R
length 0, L
time 0, T

Remarque : les mots-clés contiennent les *deux* bornes d’une intégrale en même temps. Ils doivent donc être précédés et/ou suivis de point-virgule.

On pourrait donc modifier l’expression d’une intégrale de la façon suivante :

```
\begin{equation}
  \integral[limits={height;circle;radius}, variables={\rho, \theta, z}]{\rho}
\end{equation}
```

$$\int_0^H \int_0^{2\pi} \int_0^R \rho \, d\rho \, d\theta \, dz \quad (22)$$

Il est tout à fait possible de mélanger ces mots-clés avec la syntaxe plus explicite de `limits`.

```
\begin{equation}
  \integral[limits={height;0,W;length}, variables={x, y, z}]{\kappa(T)\nabla T \cdot \mathbf{n}}
\end{equation}
```

$$\int_0^H \int_0^W \int_0^L \kappa(T) \nabla T \cdot \mathbf{n} \, dx \, dy \, dz \quad (23)$$

4.2.2 Saisir des bornes symétriques

`limits` `limits= {<+-limite>}`

nouveau: v4.1.0

Lorsque des bornes symétriques doivent être renseignées, le package peut les identifier à l’aide d’une syntaxe propre à la clé `limits`. Pour ce faire, il suffira d’insérer les tokens +- avant la borne concernée.

Remarque : la même remarque que ci-dessus s’applique. De plus, cette syntaxe *n’est pas* compatible avec les mots-clés vus juste avant. Le but est de permettre la saisie de bornes symétriques occasionnelles qui ne justifient pas entièrement la création d’un mot-clé pour une seule utilisation.

```
\begin{equation}
  a_n = \frac{1}{\pi} \integral[limits={+-\pi}]{f(x)\cos(x)}
\end{equation}
```

$$a_n = \frac{1}{\pi} \int_{-\pi}^{+\pi} f(x) \cos(x) \, dx \quad (24)$$

4.3 Retour sur la clé *variables*

4.3.1 Définir un mot-clé

```
\NewVariableKeyword \NewVariableKeyword {⟨mot-clé⟩} {⟨variables⟩} [⟨jacobien⟩]  
\RenewVariableKeyword  
\ProvideVariableKeyword  
\DeclareVariableKeyword
```

mise à jour: v4.0.0

Tout comme pour les limites, les mêmes groupes de variables réapparaissent souvent. On peut donc également définir des mots-clés contenant un ensemble de variables d'intégration préenregistrées. Les variantes de cette macro ont le même comportement que pour `\NewLimitsKeyword`. La seule différence est qu'il est ici possible de définir un jacobien, sous forme d'une `⟨csv-list⟩` si besoin. Son affichage est ensuite contrôlé par la clé `jacobian` expliquée précédemment. Voici la liste des mots-clés de variables déjà définis par le package, avec le jacobien pour certains :

```
cartesian     $x, y, z$   
planar        $x, y$   
polar         $r, \theta (r)$   
cylindrical   $r, \theta, z (r)$   
cylindrical*  $\rho, \theta, z (\rho)$   
spherical     $r, \theta, \phi (r^2, \sin \theta)$ 
```

```
\begin{equation}  
  \integral[triple=V, variables=spherical]{f(r, \theta, \phi)}  
\end{equation}
```

$$\iiint_V f(r, \theta, \phi) \, dr \, d\theta \, d\phi \quad (25)$$

4.4 Retour sur la clé *symbol*

```
\NewSymbolKeyword \NewSymbolKeyword {<clé>} {<symbole>}
\RenewSymbolKeyword
\ProvideSymbolKeyword
\DeclareSymbolKeyword
```

nouveau: v3.0.0

Afin de composer des intégrales indéfinies, le package offre essentiellement deux clés : `symbol` et `llimit`. Bien qu'elles soient simples d'utilisation, il demeure toujours laborieux de les écrire toutes les deux avec leurs valeurs. C'est pourquoi *intexgral* met à disposition des clés dites *raccourcies*, dont le but est de combiner l'action de sélection du symbole et des bornes. Contrairement aux deux macros précédentes, il s'agit ici de créer de toutes nouvelles clés, et non simplement des valeurs spécifiques qui leur sont attribuées à `symbol`. Toutes seules, ces clés modifient le symbole intégral. La valeur de ces clés correspondra elle à la borne inférieure. Voici la liste de l'ensemble des *clés-symbole* définie par le package :

<code>single</code>	symbole utilisé = <code>\int</code>
<code>double</code>	symbole utilisé = <code>\iint</code>
<code>triple</code>	symbole utilisé = <code>\iiint</code>
<code>quadruple</code>	symbole utilisé = <code>\iiiiint</code>
<code>contour</code>	symbole utilisé = <code>\oint</code>
<code>surface</code>	symbole utilisé = <code>\oiint</code>
<code>volume</code>	symbole utilisé = <code>\oiiint</code>

Important : offrir la possibilité de créer soi-même une clé à la macro `\integral` représente un risque qui sera mieux expliqué dans la prochaine section. Retenez pour l'instant que toutes ces macros émettront un message d'avertissement si vous tentez de créer une nouvelle *clé-symbole* portant le même nom qu'un groupe de bornes définies.

```
\begin{gather}
\integral[double=S, variables=planar]{f(x, y)}\[1cm]
\integral[contour=\mathcal{C}, diff-vec]{f(\vec r)}
\end{gather}
```

$$\iint_S f(x, y) \, dx \, dy \quad (26)$$

$$\oint_{\mathcal{C}} f(\vec{r}) \cdot d\vec{r} \quad (27)$$

5 Syntaxe spéciale

5.1 Présentation de la syntaxe

`\integral` `\integral` [`(\limits:variables(+j):mode)`] {`(intégrande)`}

nouveau: v3.0.0

En ce qui concerne les intégrales définies, l'utilisateur sera amené à employer au maximum quatre clés pour les composer : `limits`, `variables`, `mode` et `jacobian`. On peut néanmoins leur reprocher la même chose qu'avant ; écrire ces quatre clés, pouvant recevoir des arguments assez longs, va à l'encontre de l'objectif principal de ce package. Ainsi, l'utilisateur peut désigner comme argument optionnel d'`\integral`, une syntaxe dite *spéciale* qui n'est propre qu'à `intexgral`. La logique est la suivante :

- Vous spécifiez un argument *valide* de la clé `limits`
- Vous spécifiez un argument *valide* de la clé `variables`
- Vous spécifiez un argument *valide* de la clé `mode`
- Et vous séparez le tout d'un deux-points.

Voyons un exemple pour mieux comprendre.

```
\begin{equation}
  \integral[1, 2; 4, 5:y, x:nested]{x; y}
\end{equation}
```

$$\int_1^2 x \int_4^5 y \, dy \, dx \quad (28)$$

On entend par *argument valide* le fait que toutes les formes de valeurs acceptées par les quatre clés peuvent être pareillement adoptées par la syntaxe spéciale. Cela comprend donc les mots-clés.

```
\begin{equation}
  \integral[radius;circle;scircle:spherical:product]{r;\theta;\phi}
\end{equation}
```

$$\int_0^R r \, dr \int_0^{2\pi} \theta \, d\theta \int_0^\pi \phi \, d\phi \quad (29)$$

Afin d'inclure le jacobien, il suffit d'écrire `+j` après l'argument de `variables`. Le mode peut aussi être indiqué à l'aide d'une initiale seulement.

```
\begin{equation}
  \integral[radius;circle;scircle:spherical+j:p]{;};
\end{equation}
```

$$\int_0^R r^2 \, dr \int_0^{2\pi} \sin \theta \, d\theta \int_0^\pi d\phi \quad (30)$$

Il est important de répéter que dans la configuration classique de l'argument optionnel, il n'est bien sûr pas obligatoire de renseigner les quatre clés citées. Il en va de

même pour cette syntaxe. Puisque la clé `mode` n'est pas nécessaire pour `default`, cette partie peut être omise.

```
\begin{equation}
  \integral[1, 2; 3, 4:x, y]{xy}
\end{equation}
```

$$\int_1^2 \int_3^4 xy \, dx \, dy \quad (31)$$

Vous pouvez aussi toujours profiter du placement automatique du « dx » avec cette syntaxe en faisant abstraction de `variables`.

```
\begin{equation}
  \integral[1, 10]{x}
\end{equation}
```

$$\int_1^{10} x \, dx \quad (32)$$

Un cas particulier notable de cette syntaxe spéciale est que si vous souhaitez modifier la variable sans spécifier les bornes, vous pouvez effectuer cette action plus rapidement de la façon suivante :

```
\begin{equation}
  \integral[:z]{f(z)}
\end{equation}
```

$$\int f(z) \, dz \quad (33)$$

Même si elle n'est pas recommandée.

5.2 Fonctionnement de la syntaxe

Du point de vue de l'utilisateur, il est relativement simple de choisir quelle syntaxe adopter. Toutefois, le package doit pouvoir distinguer les deux. Sans trop rentrer dans les détails² de l'implémentation, il est porté à votre attention que le package tente d'extraire la première clé de l'argument optionnel³ et vérifie si elle existe. C'est pourquoi, si vous créez une clé avec `\NewSymbolKeyword` dont le nom pourra probablement servir de borne d'intégration, `intexgral` peut faussement évaluer la nature de l'argument optionnel. Dans ce genre de situation, l'interprétation des clés sera erronée et le résultat incorrect.

2. Les plus courageux d'entre vous sont tout de même invités à lire le code source.

3. Ou du moins, le groupe de tokens qu'il pense correspondre à une clé.

6 Paramètres annexes

\IntegralSetup \IntegralSetup {⟨liste de paramètres⟩}

nouveau: v3.0.0
mise à jour: v4.0.0

Cette macro permet, sous la syntaxe $\langle \text{clé}=\text{valeur} \rangle$, de modifier des paramètres liés à certaines clés. Toutes les assignations effectuées sont locales et la macro peut donc être placée dans un groupe si besoin. Voici un exemple de la macro avec les assignations par défaut du package :

```
\IntegralSetup{
  diff-symb    = \l_@@_default_differential_symbol_tl, % voir au début de l'implé
               mentation
  defaultvar   = {x},
  defaultvar*  = {r},
  varsep       = \thinmuskip,
  diffsep      = \thinmuskip,
  innersymbsep = {-5mu},
  postsymbsep  = {-1mu},
  vectorstyle  = \vec,
  domainstyle  = \mathbb,
  novar        = false
}
```

diff-symb diff-symb=⟨séquence de contrôle⟩

nouveau: v4.0.0

Cette clé contrôle le symbole utilisé pour les différentielles. Si l'option **italic** est utilisée, **\mathnormal** est appliqué (voir au début de l'implémentation). Contrairement à la clé homonyme de **\integral**, celle-ci agit à un niveau plus global.

defaultvar defaultvar={⟨variables⟩}

defaultvar*

nouveau: v3.0.0

Cette clé modifie la variable d'intégration insérée automatiquement lorsque **variables** n'est pas utilisée. L'argument peut tout à fait correspondre à un mot clé défini par **\NewVariableKeyword**. La variante étoilée de la clé contrôle la variable d'intégration à composer avec **diff-vec**.

varsep varsep=⟨muexpr⟩

nouveau: v4.0.0

Cette clé modifie l'espacement entre l'intégrande (ou le jacobien s'il est affiché) et les variables.

diffsep diffsep=⟨muexpr⟩

nouveau: v4.0.0

Cette clé modifie l'espacement entre les différentielles elles-mêmes.

innersymbsep innersymbsep=⟨muexpr⟩

nouveau: v4.0.0

Lorsque le mode **default** est en vigueur, c'est-à-dire que la clé **limits** est utilisée, le package insère un créneau entre les symboles pour légèrement les rapprocher. Cette clé permet donc de régler cet espacement.

postsymbsep postsymbsep=⟨muexpr⟩

nouveau: v4.0.0

Cette clé contrôle l'espacement entre le symbole et l'élément qui suit (intégrande ou différentielle selon **invert-limits**).

vectorstyle	<code>vectorstyle=</code> $\langle$ séquence de contrôle $\rangle$
nouveau: v3.0.0	Cette clé modifie la macro à appliquer aux variables quand <code>diff-vec</code> est en action. Il est donc possible d'altérer le style du vecteur : gras, souligné, ou même un autre style d'un package particulier, <code>esvect</code> par exemple.
domainstyle	<code>domainstyle=</code> $\langle$ séquence de contrôle $\rangle$
nouveau: v3.0.0	Semblablement, on peut changer la macro à appliquer pour les clés <code>domain(*)</code> .
novar	<code>novar=</code> $\langle$ true false $\rangle$
nouveau: v3.0.0 mise à jour: v4.0.0	Lorsqu'une large partie du document nécessite de ne pas inclure les différentielles, il est possible de prolonger dans la durée l'action plus ponctuelle de <code>variables=none</code> .

Historique des modifications

4.1.0 (27-07-2026)

Ajouté

- ▶ Clé `limits-mode` pour `\integral`.
- ▶ Macro `\invertdiff`.
- ▶ Syntaxe `+-` pour les limites.

Modifié

- ▶ Optimisation du code source.

4.0.0 (06-06-2026)

Ajouté

- ▶ Implémentation détaillée.
- ▶ Clés `varsep`, `diffsep`, `postsymbsep` et `diff-order`.
- ▶ Mot-clé de variable `cylindrical*`.

Modifié

- ▶ Le package `derivative` n'est plus une dépendance. Les différentielles sont maintenant gérées de façon interne.
- ▶ Clé `symbolskip`, renommée en `innersymbsep`.
- ▶ Clé `hide-diff`, renommée en `novar`.

Supprimé

- ▶ Clés `diff-star` et `diff-options`.

3.0.1 (02-01-2026)

Corrigé

- ▶ Bug avec le jacobien dans la syntaxe spéciale ([problème #3](#)).
- ▶ Documentation anglaise et française ([problème #4](#), [problème #6](#) et [problème #7](#)).

Modifié

- ▶ Les mots-clés `positive` et `real` contiennent maintenant un signe `+` ([problème #5](#)).

3.0.0 (24-12-2025)

Ajouté

- ▶ Syntaxe spéciale.
- ▶ Clés `domain*` et `mode`.
- ▶ Macros `\IntegralSetup` et `\NewSymbolKeyword`.

Supprimé

- ▶ Macros `\defaultdiff`, `\defaultvdiff` et `\vdiffstyle` en faveur de `\IntegralSetup`.

- ▶ Clés contrôlant symbole et borne, maintenant gérées à un plus haut niveau avec `\NewSymbolKeyword`.
- ▶ Clé `int-split`, en faveur de `mode`.
- ▶ Macro `\NewIntegralSymbol`.

Modifié

- ▶ Les noms de quelques clés (`variable` en `variables`, `lower-lim` et `upper-lim` en `llimit` et `ulimit`, `int-symb` en `symbol`, `invert-differentials` et `hide-differentials` en `invert-diff` et `hide-diff`).
- ▶ `jacobian` et `diff-star` ne nécessitent plus de valeur booléenne. Il suffit maintenant de les écrire pour activer leurs fonctionnalités.
- ▶ `hide-diff` attribué à une option locale plutôt qu'à une option de package.
- ▶ `limits-mode` attribué à une option de package plutôt qu'à une clé de macro.

2.0.1 (13-09-2025)

Corrigé

- ▶ Problème de compatibilité entre `unicode-math` et `amssymb` selon l'ordre de chargement ([problème #2](#)).

2.0.0 (09-09-2025)

Ajouté

- ▶ Macros `\intexgralsetup`, `\defaultdiff`, `\defaultvdiff` et `\vdiffstyle`.

Modifié

- ▶ Messages d'avertissement liés aux symboles non-existants. Ils ne sont maintenant provoqués que lorsque l'intégrale est composée.

Supprimé

- ▶ Clé `diff-vec-style` en faveur de `\vdiffstyle`.
- ▶ Message d'avertissement lié à la mauvaise utilisation des points-virgules en conjonction avec la clé `int-split`.

Corrigé

- ▶ Bug où l'intégrande n'était pas réinitialisée lorsque `\integral` était employée successivement dans le même groupe $\TeX$ ([détails ici](#)).
- ▶ Des restes de `log expl3` qui n'auraient pas dû être là.

1.1.0 (29-07-2025)

Ajouté

- ▶ Variantes étoilées pour les clés contrôlant symbole et borne (clés `single`, `contour` etc).

1.0.0 (26-07-2025) — Version initiale.

PARTIE II

IMPLÉMENTATION

1 Initialisation

```
1 <*package>
2 <@@=intexgral>
```

1.1 Déclaration du package

```
3 \NeedsTeXFormat{LaTeX2e}
4 \RequirePackage{expl3}[2025-05-14]
5
6 \def\intexgral@module{intexgral}
7 \def\intexgral@version{v4.1.0}
8 \def\intexgral@date{2026-07-27}
9 \def\intexgral@description{A LaTeX package for typesetting integrals}
10
11 \ProvidesExplPackage
12   \intexgral@module
13   \intexgral@date
14   \intexgral@version
15   \intexgral@description
```

1.2 Vérifications préliminaires

On vérifie que la version d'expl3 soit suffisamment récente avant de poursuivre. Le package requiert la macro `\tl_if_regex_match:nnTF` disponible à partir de décembre 2024.

```
16 \msg_new:nnn { intexgral } { expl3-too-old }
17 {
18   Your~expl3~installation~is~too~old,~
19   "intexgral"~requires~expl3~dated~2024-12-08~or~later.\iow_newline:
20   Package~loading~has~been~aborted.\iow_newline:
21   \msg_see_documentation_text:n { intexgral }
22 }
23
24 \@ifpackagelater{expl3}{2024-12-08}{}
25 { \msg_critical:nn { intexgral } { expl3-too-old } }
```

Puisqu'*intexgral* utilise `\mathbb` pour les clés `domain(*)`, on doit s'assurer que la macro existe. La vérification est placée dans un *hook* exécuté au début du document au cas où *amsfonts* ou tout package définissant `\mathbb` serait chargé après *intexgral*. Cela sert notamment à éviter les conflits entre packages (*unicode-math* et *amssymb* par exemple).

```
26 \hook_gput_code:nnn { begindocument/before } { . }
27 { \cs_if_exist:NF \mathbb { \RequirePackage{amsfonts} } }
```

2 Options de package

2.1 Définition des variables

`\l_@@_invert_limits_bool`

Booléen servant à inverser la convention d'ordre des bornes.

28 `\bool_new:N \l__intexgral_invert_limits_bool`

`\l_@@_deactivate_variables_bool`

Détermine si les différentielles doivent être affichées ou non.

29 `\bool_new:N \l__intexgral_deactivate_variables_bool`

`\l_@@_invert_differential_bool`

Détermine l'ordre d'affichage des différentielles.

30 `\bool_new:N \l__intexgral_invert_differential_bool`

`\l_@@_default_differential_symbol_tl`

Contient le symbole par défaut pour les différentielles.

31 `\tl_new:N \l__intexgral_default_differential_symbol_tl`

`\l_@@_limits_style_tl`

Contient la primitive $\TeX$ à utiliser pour le style des bornes.

32 `\tl_new:N \l__intexgral_limits_style_tl`

2.2 Déclaration des options

On peut dès à présent définir les options de package avant de les charger.

```
33 \keys_define:nn { intexgral }
34   {
35     invert-limits .bool_set:N = \l__intexgral_invert_limits_bool,
36     invert-limits .usage:n    = { preamble },
37     invert-limits .initial:n  = { false },
38
39     invert-diff .bool_set:N = \l__intexgral_invert_differential_bool,
40     invert-diff .usage:n    = { preamble },
41     invert-diff .initial:n  = { false },
42
43     limits-mode .choice:,
44     limits-mode / limits .code:n =
45       { \tl_set_eq:NN \l__intexgral_limits_style_tl \tex_limits:D },
46     limits-mode / nolimits .code:n =
47       { \tl_set_eq:NN \l__intexgral_limits_style_tl \tex_nolimits:D },
48
49     limits-mode .usage:n    = { preamble },
50     limits-mode .initial:n  = { nolimits },
51
52     italic .choice:,
53     italic / true .code:n =
54       {
55         \tl_set:Nn \l__intexgral_default_differential_symbol_tl
```

```

56         { \mathnormal{d} }
57     },
58     italic / false .code:n =
59     {
60         \tl_set:Nn \l__intexgral_default_differential_symbol_tl
61         { \mathrm{d} }
62     },
63     italic .usage:n = { preamble },
64     italic .initial:n = { false },
65
66     upright .choice:,
67     upright / true .code:n =
68     {
69         \tl_set:Nn \l__intexgral_default_differential_symbol_tl
70         { \mathrm{d} }
71     },
72     upright / false .code:n =
73     {
74         \tl_set:Nn \l__intexgral_default_differential_symbol_tl
75         { \mathnormal{d} }
76     },
77     upright .usage:n = { preamble },
78     upright .initial:n = { true }
79 }
80
81 \ProcessKeyOptions[intexgral]

```

3 Messages

La partie suivante de l'implémentation définit l'ensemble des messages d'avertissement et d'erreur.

\g_@@_integral_number_int

On commence par créer un compteur global servant à numéroté les intégrales composées dans le document.

```
82 \int_gzero_new:N \g__intexgral_integral_number_int
```

\@@_warning_msg_header: ★

On peut dès lors définir une macro servant d'en-tête aux messages d'avertissement. Celle-ci inclut le compteur vu juste avant.

```

83 \cs_new:Nn \__intexgral_warning_msg_header: {
84     (
85         integral~no.~
86         \int_use:N\g__intexgral_integral_number_int\c_space_tl
87         \msg_line_context:
88     )
89     \iow_newline:
90 }

```

Tous les messages sont maintenant définis.

```

91 \msg_new:nnnn { intexgral } { symb-key-alr-def }
92 { Symbol~key~"\tl_trim_spaces:n{#1}"~is~already~defined. }
93 { Use~\token_to_str:N \RenewSymbolKeyword\ instead. }
94

```

```

95 \msg_new:nnnn { integragr } { symb-key-not-def }
96   { Symbol~key~"\tl_trim_spaces:n{#1}"~is~not~defined. }
97   { Use~\token_to_str:N \NewSymbolKeyword\ instead. }
98
99 \msg_new:nnnn { integragr } { diff-group-alr-def }
100  { Differential~group~"\tl_trim_spaces:n{#1}"~is~already~defined. }
101  { Use~\token_to_str:N \RenewVarKeyword\ instead. }
102
103 \msg_new:nnnn { integragr } { diff-group-not-def }
104  { Differential~group~"\tl_trim_spaces:n{#1}"~is~not~defined. }
105  { Use~\token_to_str:N \NewVarKeyword\ instead. }
106
107 \msg_new:nnnn { integragr } { lim-group-alr-def }
108  { Limits~group~"\tl_trim_spaces:n{#1}"~is~already~defined. }
109  { Use~\token_to_str:N \RenewLimitsKeyword\ instead. }
110
111 \msg_new:nnnn { integragr } { lim-group-not-def }
112  { Limits~group~"\tl_trim_spaces:n{#1}"~is~not~defined. }
113  { Use~\NewLimitsKeyword\ instead. }
114
115 \msg_new:nnn { integragr } { key-exists-for-lim }
116   {
117     Symbol~key~already~defined~with~
118     \token_to_str:N \NewLimitsKeyword\ \__integragr_warning_msg_header:
119     Key~"#1"~already~exists~for~predefined~limits.~
120     This~can~disrupt~the~functioning~of~the~special~syntax.
121   }
122
123 \msg_new:nnn { integragr } { unknown-symb }
124   {
125     Unknown~integral~symbol~\__integragr_warning_msg_header:
126     The~symbol~"\tl_trim_spaces:n{#1}"~has~been~replaced~with~
127     \token_to_str:N\int.
128   }
129
130 \msg_new:nnn { integragr } { no-jacobian }
131   {
132     Jacobian~unavailable~\__integragr_warning_msg_header:
133     A~Jacobian~was~requested~for~the~"#1"~keyword,
134     ~but~none~was~declared~for~the~latter.
135   }
136
137 \msg_new:nnn { integragr } { use-glyph-instead }
138   {
139     Consider~using~the~dedicated~glyph~\__integragr_warning_msg_header:
140     The~key~"nint"~was~used~with~fewer~than~
141     5~regular~integral~signs.
142   }

```

4 Variantes utiles

```
\str_if_eq:neTF
\str_if_eq:neF
\keys_if_exist:nVTF
\seq_use:Ne
\str_if_eq_p:en
\prop_gput:Nne
\seq_gset_split:cnn
\seq_set_split:Nnf
```

```
143 \cs_generate_variant:Nn \str_if_eq:nnTF { neTF }
144 \cs_generate_variant:Nn \str_if_eq:nnF { neF }
145 \cs_generate_variant:Nn \keys_if_exist:nnTF { nVTF }
146 \cs_generate_variant:Nn \seq_use:Nn { Ne }
147 \cs_generate_variant:Nn \str_if_eq_p:nn { en }
148 \cs_generate_variant:Nn \prop_gput:Nnn { Nne }
149 \cs_generate_variant:Nn \seq_gset_split:Nnn { cnn }
150 \cs_generate_variant:Nn \seq_set_split:Nnn { Nnf }
```

5 Variables internes

5.1 Tokens

```
\l_@@_integrand_tl
```

Token contenant l'intégrande. Il est utilisé tel quel pour le mode `default` tandis que pour les autres modes, il est séparé ultérieurement en une séquence.

```
151 \tl_new:N \l__intexgral_integrand_tl
```

```
\l_@@_integral_symbol_tl
```

Token contenant le symbole de l'intégrale. Lorsque non modifié par la clé `symbol`, il faut que le token soit tout de même initialisé à `\int`.

```
152 \tl_new:N \l__intexgral_integral_symbol_tl
153 \tl_set_eq:NN \l__intexgral_integral_symbol_tl \int
```

```
\l_@@_lower_limit_tl
```

```
\l_@@_upper_limit_tl
```

Tokens contenant les bornes d'intégration.

```
154 \tl_new:N \l__intexgral_lower_limit_tl
155 \tl_new:N \l__intexgral_upper_limit_tl
```

```
\l_@@_differential_symbol_tl
```

Token contenant le symbole des différentielles. Il contient `\l_@@_default_differential_symbol_tl`.

```
156 \tl_new:N \l__intexgral_differential_symbol_tl
```

```
\l_@@_vectorial_differential_style_tl
```

```
\l_@@_domain_style_tl
```

Tokens contenant les macros à appliquer aux clés `diff-vec` et `domain(*)`.

```
157 \tl_new:N \l__intexgral_vectorial_differential_style_tl
158 \tl_new:N \l__intexgral_domain_style_tl
```

`\l_@@_domain_char_tl`
`\l_@@_domain_dimension_tl`

Tokens utilisés pour stocker respectivement le caractère et la dimension d'un domaine d'intégration.

```
159 \tl_new:N \l__intexgral_domain_char_tl
160 \tl_new:N \l__intexgral_domain_dimension_tl
```

`\c_@@_left_bracket_tl`
`\c_@@_right_bracket_tl`

Tokens constants pour la clé `limits*`.

```
161 \tl_const:Nn \c__intexgral_left_bracket_tl { [ }
162 \tl_const:Nn \c__intexgral_right_bracket_tl { ] }
```

`\l_@@_key_name_tl`

Token contenant le potentiel nom d'une clé, utilisé pour différencier la syntaxe `<clé=valeur>` de la syntaxe spéciale.

```
163 \tl_new:N \l__intexgral_key_name_tl
```

5.2 Booléens

`\l_@@_manual_differential_bool`

Booléen servant à déterminer si les différentielles sont insérées par le biais de `\differentials`.

```
164 \bool_new:N \l__intexgral_manual_differential_bool
```

`\l_@@_custom_variables_bool`

Booléen servant à déterminer si les variables sont définies avec `variables`.

```
165 \bool_new:N \l__intexgral_custom_variables_bool
```

`\l_@@_separate_integral_bool`

Booléen servant à déterminer si l'intégrale est *séparée* en plusieurs parties, c'est-à-dire si les modes `nested` ou `product` sont en vigueur.

```
166 \bool_new:N \l__intexgral_separate_integral_bool
```

`\l_@@_vectorial_differential_bool`

Booléen décidant s'il faut appliquer des vecteurs aux différentielles.

```
167 \bool_new:N \l__intexgral_vectorial_differential_bool
```

`\l_@@_jacobian_bool`

Booléen décidant s'il faut afficher le jacobien ou non.

```
168 \bool_new:N \l__intexgral_jacobian_bool
```

5.3 Séquences et clists

`\l_@@_domain_seq`

Séquence contenant les domaines d'intégration et leurs dimensions (les séquences principales seront expliquées plus tard).

169 `\seq_new:N \l__intexgral_domain_seq`

`\l__intexgral_default_variables_seq`

`\l__intexgral_default_vectorial_variables_seq`

Séquences contenant les variables par défaut, utilisées lorsque la clé `variables` n'est pas renseignée.

170 `\seq_new:N \l__intexgral_default_variables_seq`

171 `\seq_new:N \l__intexgral_default_vectorial_variables_seq`

`\l_@@_differential_order_clist`

Liste contenant les puissances des différentielles (clist et non séquence car `.seq_set:N` n'existe pas pour les clés).

172 `\clist_new:N \l__intexgral_differential_order_clist`

5.4 Liste de propriétés

`\g_@@_limits_keyword_prop`

`\g_@@_differential_group_keyword_prop`

Une liste de propriété est créée pour stocker d'un côté les mots clés propres aux bornes, et de l'autre, ceux pour les différentielles.

173 `\prop_new:N \g__intexgral_limits_keyword_prop`

174 `\prop_new:N \g__intexgral_differential_group_keyword_prop`

5.5 Muskip

`\l_@@_symbol_inner_sep_muskip`

`\l_@@_symbol_post_sep_muskip`

`\l_@@_differential_sep_muskip`

`\l_@@_variable_sep_muskip`

Muskips internes pour stocker les espacements avec respectivement:

1. L'espacement entre les symboles d'intégration lorsque plusieurs sont générés par `limits(*)`.
2. L'espacement qui suit le symbole.
3. L'espacement entre les différentielles lorsque la clé `variables` est utilisée.
4. L'espacement entre les différentielles et l'intégrande.

175 `\muskip_new:N \l__intexgral_symbol_inner_sep_muskip`

176 `\muskip_new:N \l__intexgral_symbol_post_sep_muskip`

177 `\muskip_new:N \l__intexgral_variable_sep_muskip`

178 `\muskip_new:N \l__intexgral_differential_sep_muskip`

5.6 String

`\l_@@_display_mode_str`

Variable interne pour stocker le mode d’affichage de l’intégrale. La raison d’utiliser un *string* et non pas un token s’explique par le manque d’un `\tl_case:nnTF`.

```
179 \str_new:N \l__intexgral_display_mode_str
```

6 Macros et séquences auxiliaires

On définit ici les deux primitives TeX `\mkern` et `\mathchoice` avec une syntaxe *expl3*.

`\@@_mkern:N` `\@@_mkern:N` `\muskip`

```
180 \cs_new_protected:Npn \__intexgral_mkern:N #1
181 { \tex_mkern:D #1 \scan_stop: }
```

`\@@_mathchoice:nnnn` `\@@_mathchoice:nnnn` `{\display style}` `{\inline style}` `{\script style}`
`{\scriptscript style}`

```
182 \cs_new_eq:NN \__intexgral_mathchoice:nnnn \mathchoice
```

`\@@_invert_limits:w` `\@@_invert_limits:w` `\langle`borne inférieure`\rangle`,`\langle`borne supérieure`\rangle``\q_stop`

Macro inversant les limites pour l’option `invert-limits`. L’action est purement développable et plus efficace qu’un `\seq_inverse:N` dans le contexte du package, étant donné qu’au maximum deux éléments sont présents.

```
183 \cs_new:Npn \__intexgral_invert_limits:w #1,#2\q_stop{#2,#1}
```

`\@@_general_integral_symbol:`

On crée une macro qui contient l’expression générale d’une intégrale: symbole, style des bornes, borne inférieure et borne supérieure. Il ne reste plus qu’à assigner les tokens pour composer un symbole complet, et éventuellement répéter la macro pour des intégrales définies sur plusieurs variables.

```
184 \cs_new:Npn \__intexgral_general_integral_symbol:
185 {
186   \l__intexgral_integral_symbol_tl
187   \l__intexgral_limits_style_tl
188   \c_math_subscript_token
189   { \l__intexgral_lower_limit_tl }
190   \c_math_superscript_token
191   { \l__intexgral_upper_limit_tl }
192 }
```

6.1 Séquences principales

Au vu du grand nombre de clés existantes et de combinaisons possibles, il faut judicieusement organiser et stocker les tokens afin de composer l'intégrale finale. L'approche adoptée a été de créer des séquences dédiées à chaque élément d'une intégrale:

`\l_@@_integral_symbol_seq`

Une pour le symbole (ou *les* symboles, si la clé `limits*` est en usage).

193 `\seq_new:N \l__intexgral_integral_symbol_seq`

`\l_@@_integrand_seq`

Une pour l'intégrande (ou *les* intégrandes, si découpée en plusieurs parties par `nested` ou `product`).

194 `\seq_new:N \l__intexgral_integrand_seq`

`\l_@@_jacobian_seq`

Une pour le jacobien.

195 `\seq_new:N \l__intexgral_jacobian_seq`

`\l_@@_differential_seq`

Et enfin, une pour les différentielles.

196 `\seq_new:N \l__intexgral_differential_seq`

De cette façon, on peut construire l'intégrale finale en jouant sur la façon d'extraire les différents éléments de ces séquences. En voici une illustration plus concrète: Pour une intégrale indéfinie.

$$\boxed{\overset{S}{\iint_S} \boxed{\overset{I}{f(x,y)} \boxed{\overset{D}{dx dy}}}}$$

Pour une intégrale définie.

$$\boxed{\overset{S_1}{\int_a^b} \boxed{\overset{S_2}{\int_c^d} \boxed{\overset{I_1}{f(x)} \boxed{\overset{I_2}{g(y)} \boxed{\overset{D_1}{dx} \boxed{\overset{D_2}{dy}}}}}}}}$$

6.2 Séquences subsidiaires

Quelques séquences supplémentaires seront nécessaires durant les étapes intermédiaires qui consisteront à préparer les quatre séquences principales listées ci-dessus.

`\l_@@_variables_seq`

Tout d'abord, une séquence contenant les variables d'intégration.

197 `\seq_new:N \l__intexgral_variables_seq`

`\l_@@_limits_seq`

Ensuite, une séquence contenant les paires de limites d'intégration, c'est-à-dire les éléments de la clé `limits` séparés en deux par le point-virgule.

198 `\seq_new:N \l__intexgral_limits_seq`

```
\l_@@_upper_limits_seq
\l_@@_lower_limits_seq
```

Deux séquences contenant respectivement les bornes supérieures et inférieures. Ainsi, si la clé `limits` contient les éléments `a`; `b`, `c`; `d` et `e`; `f`, alors la séquence des bornes inférieures contiendra les éléments `a`, `c` et `e`, tandis que celle des bornes supérieures contiendra les éléments `b`, `d` et `f`.

```
199 \seq_new:N \l__integragl_upper_limits_seq
200 \seq_new:N \l__integragl_lower_limits_seq
```

7 Bornes d'intégration

```
\@@_has_pm_p:n
```

On définit une macro conditionnelle qui va gérer la syntaxe `+-<limite>`. Elle se contente de vérifier que le premier et deuxième token de l'argument de `limits` correspondent respectivement à `+` et `-`.

```
201 \prg_new_conditional:Npnn \__integragl_has_pm:n #1 { p }
202 {
203     \bool_lazy_and:nnTF
204     { \exp_args:Ne \str_if_eq_p:nn { \str_item:nn { #1 } { 1 } } { + } }
205     { \exp_args:Ne \str_if_eq_p:nn { \str_item:nn { #1 } { 2 } } { - } }
206     { \prg_return_true: }
207     { \prg_return_false: }
208 }
```

```
\@@_trim_pm:w
```

Ensuite, une macro délimitée est créée afin de supprimer le `+-`

```
209 \cs_new:Npn \__integragl_trim_pm:w +-#1\q_stop{#1}
```

```
\@@_parse_limits:n ★ \@@_parse_limits:n {<borne inférieure>} {< borne supérieure>}, <mot-clé>
```

Cette macro exécute l'une des actions suivantes selon la valeur booléenne qui est vraie:

- Si l'argument est un mot-clé défini pour les bornes, la macro se développe en son contenu en l'inversant si besoin (selon le statut de `\l_@@_invert_limits_bool`).
- Si l'argument débute par les tokens `+-`, supprime ces derniers et ajoute manuellement les signes.
- Si aucune de ces deux conditions n'est remplie, laisse simplement l'argument dans le flux d'entrée.

```
210 \cs_new:Npn \__integragl_parse_limits:n #1
211 {
212     \bool_case:nF
213     {
214         { \prop_if_in_p:Nn \g__integragl_limits_keyword_prop { #1 } }
215         {
216             \bool_if:NTF \l__integragl_invert_limits_bool
217             {
218                 \exp_last_unbraced:Ne \__integragl_invert_limits:w
```

```

219             { \prop_item:Nn \g__intexgral_limits_keyword_prop { #1 } }
220             \q_stop
221         }
222         { \prop_item:Nn \g__intexgral_limits_keyword_prop { #1 } }
223     }
224     { \__intexgral_has_pm_p:n { #1 } }
225     {
226         -\__intexgral_trim_pm:w #1\q_stop,
227         +\__intexgral_trim_pm:w #1\q_stop
228     }
229 }
230 { #1 }
231 }

```

\@@_set_limits_regular:

Une fois que l'argument de la clé `limits` a été converti en une séquence (action effectuée au sein de `.code:n` dans la déclaration des clés), on assigne chaque paire de borne à une séquence temporaire. Celle-ci est créée de sorte à ce que si un mot-clé est présent, il est substitué à ses bornes correspondantes en vertu de la macro précédente. L'action est exécutée dans un développement de type `\romannumeral` afin d'éviter le développement des bornes contenant des macros fragiles (type `\mathbb`). On peut alors extraire la borne inférieure et la borne supérieure de cette séquence pour les assigner à leur séquence respective.

```

232 \cs_new_protected:Nn \__intexgral_set_limits_regular:
233 {
234     \seq_map_inline:Nn \l__intexgral_limits_seq
235     {
236         \seq_set_split:Nnf \l_tmpa_seq { , }
237         { \__intexgral_parse_limits:n { ##1 } }
238         \seq_put_right:Ne \l__intexgral_lower_limits_seq
239         { \seq_item:Nn \l_tmpa_seq { 1 } }
240         \seq_put_right:Ne \l__intexgral_upper_limits_seq
241         { \seq_item:Nn \l_tmpa_seq { 2 } }
242     }
243 }

```

\@@_set_limits_starred:

La macro suivante est similaire à la précédente, mais elle s'occupe des bornes sous forme d'intervalle lorsque la clé `limits*` est utilisée.

```

244 \cs_new_protected:Npn \__intexgral_set_limits_starred:
245 {
246     \seq_map_indexed_inline:Nn \l__intexgral_limits_seq
247     {
248         \seq_set_split:Nnf \l_tmpa_seq { , }
249         { \__intexgral_parse_limits:n { ##2 } }
250         \bool_if:NTF \l__intexgral_invert_limits_bool
251         {
252             \tl_set:Ne \l__intexgral_lower_limit_tl
253             {
254                 \seq_item:Nn \l_tmpa_seq { 2 }
255                 \tex_mathpunct:D,
256                 \seq_item:Nn \l_tmpa_seq { 1 }
257             }
258         }

```

```

259     {
260         \tl_set:Ne \l__intexgral_lower_limit_tl
261         {
262             \seq_item:Nn \l_tmpa_seq { 1 }
263             \tex_mathpunct:D,
264             \seq_item:Nn \l_tmpa_seq { 2 }
265         }
266     }
267     \bool_if:NTF \l__intexgral_invert_limits_bool
268     { \seq_put_right:Ne \l__intexgral_upper_limits_seq }
269     { \seq_put_right:Ne \l__intexgral_lower_limits_seq }
270     {
271         \str_case:e:nnF { \seq_item:Nn \l_tmpa_seq { 1 } }
272         { { -\infty } { \tex_left:D \c__intexgral_right_bracket_tl } }
273         { \tex_left:D \c__intexgral_left_bracket_tl }
274         \tl_use:N \l__intexgral_lower_limit_tl
275         \str_case:e:nnF { \seq_item:Nn \l_tmpa_seq { 2 } }
276         { { +\infty } { \tex_right:D \c__intexgral_left_bracket_tl } }
277         { \tex_right:D \c__intexgral_right_bracket_tl }
278     }
279 }
280 }

```

`\@@_set_limits:nn` `\@@_set_limits:nn {<borne inférieure>} {<borne supérieure>}`

Selon l'option `invert-limits`, cette macro assigne les bornes inférieures et supérieures aux tokens dédiés. Si elle est active, l'assignation devient quelque peu contre-intuitive (la borne inférieure est assignée à `\l__@@_upper_limit_tl` et inversement), mais produira tout de même le résultat attendu. Le développement des bornes est empêché afin d'éviter les problèmes de macros fragiles comme vu précédemment.

```

281 \cs_new_protected:Npn \__intexgral_set_limits:nn #1#2
282 {
283     \bool_if:NTF \l__intexgral_invert_limits_bool
284     {
285         \tl_set:Ne \l__intexgral_lower_limit_tl { \exp_not:n { #2 } }
286         \tl_set:Ne \l__intexgral_upper_limit_tl { \exp_not:n { #1 } }
287     }
288     {
289         \tl_set:Ne \l__intexgral_lower_limit_tl { \exp_not:n { #1 } }
290         \tl_set:Ne \l__intexgral_upper_limit_tl { \exp_not:n { #2 } }
291     }
292 }

```

`\@@_generate_integral_sequence:`

Une fois les séquences des bornes inférieures et supérieures remplies, on peut utiliser la structure des séquences à notre avantage pour générer automatiquement autant de symboles que nécessaire. On procède à une boucle incrémentée dans laquelle on effectue deux actions:

1. À l'aide de `\@@_set_limits:nn`, on assigne les bornes d'intégrations aux tokens `\l_@@_lower_limit_tl` et `\l_@@_upper_limit_tl` à partir du i^e élément des séquences `\l_@@_lower_limits_seq` et `\l_@@_upper_limits_seq` respectivement.
2. On place à droite (c.-à-d. dans l'ordre) de la séquence `\l_@@_integral_symbol_seq` la macro `\@@_general_integral_symbol:`, qui contient la forme générale d'une intégrale. À noter qu'on développe au maximum son contenu afin de récupérer l'assignation immédiate des tokens.

```
293 \cs_new_protected:Nn \__intexgral_generate_integral_sequence:
294 {
295     \int_set:Nn \l_tmpa_int
296     {
297         \seq_if_empty:NTF \l__intexgral_lower_limits_seq
298         { \seq_count:N \l__intexgral_upper_limits_seq }
299         { \seq_count:N \l__intexgral_lower_limits_seq }
300     }
301     \int_step_inline:nn { \l_tmpa_int }
302     {
303         \__intexgral_set_limits:nn
304         { \seq_item:Nn \l__intexgral_lower_limits_seq { ##1 } }
305         { \seq_item:Nn \l__intexgral_upper_limits_seq { ##1 } }
306         \seq_put_right:Ne \l__intexgral_integral_symbol_seq
307         { \__intexgral_general_integral_symbol: }
308     }
309 }
```

8 Variables

`\@@_parse_variables:n` `\@@_parse_variables:n` {<variable d'intégration>}, <mot-clé>

Cette macro s'applique à chaque élément de l'argument de la clé `variables` pour vérifier s'il s'agit d'un mot-clé ou d'une liste explicite, de la même manière que pour les limites. Il s'agit principalement de transférer les éléments de `\l_@@_variables_seq` (contenant par exemple « x, y, z ») vers `\l_@@_differential_seq` (qui contiendra « dx, dy, dz »); tout en prenant en compte les différentes options qui ont été appliquées. Dans le cas où aucune variable n'est saisie, la macro s'occupe d'y placer x ou $\vec{r}$ en fonction de la valeur du booléen `\l_@@_vectorial_differential_bool`.

```
310 \cs_new_protected:Nn \__intexgral_parse_variables:
311 {
312     \bool_if:NF \l__intexgral_custom_variables_bool
313     {
314         \bool_if:NTF \l__intexgral_vectorial_differential_bool
315         {
316             \seq_set_eq:NN
317             \l__intexgral_variables_seq
318             \l__intexgral_default_vectorial_variables_seq
```

```

319         }
320     {
321         \seq_set_eq:NN
322         \l__intexgral_variables_seq
323         \l__intexgral_default_variables_seq
324     }
325 }
326 \seq_map_indexed_inline:Nn \l__intexgral_variables_seq
327 {
328     \seq_put_right:Ne \l__intexgral_differential_seq
329     {
330         \exp_not:V \l__intexgral_differential_symbol_tl
331         \clist_if_empty:NF \l__intexgral_differential_order_clist
332         {
333             \c_math_superscript_token
334             {
335                 \clist_item:Nn
336                 \l__intexgral_differential_order_clist
337                 { ##1 }
338             }
339         }
340         \bool_if:NT \l__intexgral_vectorial_differential_bool
341         { \exp_not:V \l__intexgral_vectorial_differential_style_tl }
342         { ##2 }
343     }
344 }
345 }

```

9 Syntaxe spéciale

\@@_retrieve_key_name:w ★ \@@_retrieve_key_name:w <clé>=<valeur> \q_stop

Afin de gérer la syntaxe spéciale, on commence par créer une macro délimitée qui extrait la valeur de la clé et son nom, et qui ne renvoie que ce dernier.

```

346 \cs_new:Npn \__intexgral_retrieve_key_name:w #1=#2\q_stop { #1 }

```

\@@_extract_first_key_name:n \@@_extract_first_key_name:n {<argument optionnel d'\integral>}

On associe ensuite l'argument optionnel reçu par `\integral` à une *comma list* temporaire, dont on extrait le premier élément; on vérifie alors s'il contient un signe «=». Le cas échéant, on emploie la macro précédente pour ne récupérer que le nom de la clé; sinon, on copie simplement le groupe de tokens récupéré dans `\l__@@_key_name_tl`.

```

347 \cs_new_protected:Npn \__intexgral_extract_first_key_name:n #1
348 {
349     \seq_set_split:Nnn \l_tmpa_seq { : } { #1 }
350     \tl_set:Ne \l_tmpa_tl { \seq_item:Nn \l_tmpa_seq { 1 } }
351     \tl_if_in:NnTF \l_tmpa_tl { = }
352     {
353         \tl_set:Ne \l__intexgral_key_name_tl
354         {
355             \exp_last_unbraced:NV
356             \__intexgral_retrieve_key_name:w \l_tmpa_tl \q_stop
357         }
358     }

```

```

359         { \tl_set_eq:NN \l__intexgral_key_name_tl \l_tmpa_tl }
360     }

```

```

\@@_parse_integral_keys:n \@@_parse_integral_keys:n {⟨première clé de l'argument optionnel⟩}

```

En se servant du module *l3keys*, on peut évaluer si le groupe de tokens capturé correspond à une clé définie, auquel cas on applique directement `\keys_set:nn`. Si le test se révèle négatif, on assigne manuellement les clés en considérant l'argument optionnel comme une séquence à délimiter avec le deux-point. Des séquences temporaires sont employées pour manipuler les différentes parties de l'argument. On vérifie notamment la présence du token `+j` pour activer la clé `jacobian`, et on ajoute la variable par défaut si elle n'est pas spécifiée. Enfin, on assigne les valeurs aux clés `limits`, `variables` et `mode` en fonction des éléments extraits.

```

361 \cs_new_protected:Npn \__intexgral_parse_integral_keys:n #1
362 {
363     \keys_if_exist:nVTF { integral } \l__intexgral_key_name_tl
364     { \keys_set:nn { integral } { #1 } }
365     {
366         \regex_split:nnN { : } { #1 } \l_tmpa_seq
367         \str_if_eq:eeTF
368         { \seq_item:Nn \l_tmpa_seq { 2 } }
369         { +j }
370         {
371             \exp_args:NNne \seq_set_item:Nnn \l_tmpa_seq { 2 }
372             {
373                 \seq_use:Nn
374                 \l__intexgral_default_variables_seq
375                 { , }
376                 +j
377             }
378         }
379         {
380             \int_compare:nNnT { \seq_count:N \l_tmpa_seq } < { 2 }
381             {
382                 \seq_put_right:Ne
383                 \l_tmpa_seq
384                 { \seq_use:Nn \l__intexgral_default_variables_seq { , } }
385             }
386         }
387         \int_compare:nNnT { \seq_count:N \l_tmpa_seq } < { 3 }
388         { \seq_put_right:Nn \l_tmpa_seq { d } }
389         \seq_set_split:Nne \l_tmpa_seq
390         { + }
391         { \seq_item:Nn \l_tmpa_seq { 2 } }
392         \keys_set:ne { integral }
393         {
394             limits = { \seq_item:Nn \l_tmpa_seq { 1 } },
395             variables = { \seq_item:Nn \l_tmpa_seq { 1 } },
396             mode =
397             {
398                 \str_case:enF { \seq_item:Nn \l_tmpa_seq { 3 } }
399                 {
400                     { d } { default }
401                     { n } { nested }
402                     { p } { product }
403                     { default } { default }
404                     { nested } { nested }

```

```

405         { product } { product }
406     }
407     { default }
408 },
409 \bool_if:nT
410 {
411     \int_compare_p:nNn { \seq_count:N \l_tmpb_seq } > { 1 }
412     &&
413     \str_if_eq_p:en { \seq_item:Nn \l_tmpb_seq { 2 } } { j }
414 }
415 { jacobian }
416 }
417 }
418 }

```

10 Composition de l'intégrale

\@@_integral_preconfiguration:

Avant de pouvoir composer l'intégrale dans le document, un certain nombre de vérifications doivent être effectuées. En effet, selon la combinaison de clés renseignée dans l'argument optionnel, les séquences ne sont pas nécessairement encore remplies. On effectue donc l'ensemble des vérifications suivantes:

- On regarde si l'intégrande doit être séparée en plusieurs parties par les modes `nested` et `product`, auquel cas on remplit la séquence de l'intégrande en séparant les éléments à partir du point-virgule. Sinon, on ajoute simplement l'intégrande (`\l_@@_integrand_tl`) à la séquence.
- On vérifie si la séquence des symboles est vide, signe que la clé `limits*` n'a pas été appelée. En effet, seule l'utilisation de cette dernière engendre l'exécution de toutes les macros présentées en section 7. Dans ce cas-là, on ajoute simplement à la séquence la forme générale de l'intégrale, dont le symbole et les bornes seront éventuellement modifiés par les clés `symbol`, `ulimit` et `llimit`.
- On contrôle ensuite la désactivation des différentielles en plus de détecter la présence de `\differentials` à l'aide d'une expression régulière.

```

419 \cs_new_protected:Nn \__intexgral_integral_preconfiguration:
420 {
421     \bool_if:NTF \l__intexgral_separate_integral_bool
422     {
423         \seq_set_split:NnV
424             \l__intexgral_integrand_seq
425             { ; }
426             \l__intexgral_integrand_tl
427     }
428     {
429         \seq_put_right:NV
430             \l__intexgral_integrand_seq
431             \l__intexgral_integrand_tl
432     }
433     \seq_if_empty:NT \l__intexgral_integral_symbol_seq
434     {
435         \seq_put_right:Nn \l__intexgral_integral_symbol_seq

```

```

436         { \__intexgral_general_integral_symbol: }
437     }
438     \bool_if:NF \l__intexgral_deactivate_variables_bool
439     { \__intexgral_parse_variables: }
440     \tl_if_regex_match:NnTF \l__intexgral_integrand_tl { \c{differentials} }
441     { \bool_set_true:N \l__intexgral_manual_differential_bool }
442     { \bool_set_false:N \l__intexgral_manual_differential_bool }
443 }

```

On peut désormais définir l'ensemble des six macros qui serviront à composer une intégrale précise; trois modes autorisés par `\mode`, chacun accompagné d'une variante pour les différentielles inversées. On notera par ailleurs que `\differentials` n'est créé que si le booléen `\l__manual_differential_bool` est vrai, afin d'éviter des définitions inutiles.

10.1 Mode *default*

`\@@_print_default_integral:`

Pour le mode par défaut, on compose dans l'ordre: symbole intégrale, intégrande, jacobien (si applicable), et différentielles. Puisque les éléments sont successifs, on applique `\seq_use:Nn` pour chacune des séquences en insérant un crénage lorsque c'est nécessaire. À noter que si `\l__vectorial_differential_bool` est vrai, on ajoute un point médian et supprime le crénage précédent (pour des raisons d'espacement propres à `\cdot`).

```

444 \cs_new_protected:Npn \__intexgral_print_default_integral:
445 {
446     \bool_if:NT \l__intexgral_manual_differential_bool
447     {
448         \cs_set_protected:Npn \differentials
449         {
450             \seq_use:Nn \l__intexgral_differential_seq
451             { \__intexgral_mkern:N \__intexgral_differential_sep_muskip }
452         }
453     }
454     \seq_use:Nn \l__intexgral_integral_symbol_seq
455     { \__intexgral_mkern:N \l__intexgral_symbol_inner_sep_muskip }
456     \__intexgral_mkern:N \l__intexgral_symbol_post_sep_muskip
457     \seq_use:Nn \l__intexgral_integrand_seq { }
458     \bool_if:NT \l__intexgral_jacobian_bool
459     { \seq_use:Nn \l__intexgral_jacobian_seq { } }
460     \__intexgral_mkern:N \l__intexgral_variable_sep_muskip
461     \bool_if:NT \l__intexgral_vectorial_differential_bool
462     { \tex_unkern:D \cdot }
463     \bool_if:NF \l__intexgral_manual_differential_bool
464     {
465         \seq_use:Nn \l__intexgral_differential_seq
466         { \__intexgral_mkern:N \l__intexgral_differential_sep_muskip }
467     }
468 }

```

`\@@_print_default_integral_inv:`

Pour les différentielles inversées, rien de bien différent, si ce n'est que `\l__integrand_seq` et `\l__differential_seq` sont interverties.

```

469 \cs_new_protected:Npn \__intexgral_print_default_integral_inv:
470 {
471   \bool_if:NT \l__intexgral_manual_differential_bool
472   {
473     \cs_set_protected:Npn \differentials
474     {
475       \seq_use:Nn \l__intexgral_differential_seq
476       { \__intexgral_mkern:N \l__intexgral_differential_sep_muskip }
477     }
478   }
479   \seq_use:Nn \l__intexgral_integral_symbol_seq
480   { \__intexgral_mkern:N \l__intexgral_symbol_inner_sep_muskip }
481   \__intexgral_mkern:N \l__intexgral_symbol_post_sep_muskip
482   \bool_if:NF \l__intexgral_manual_differential_bool
483   {
484     \seq_use:Nn \l__intexgral_differential_seq
485     { \__intexgral_mkern:N \l__intexgral_differential_sep_muskip }
486   }
487   \__intexgral_mkern:N \l__intexgral_variable_sep_muskip
488   \bool_if:NT \l__intexgral_vectorial_differential_bool
489   { \tex_unkern:D \cdot}
490   \seq_use:Nn \l__intexgral_integrand_seq { }
491   \bool_if:NT \l__intexgral_jacobian_bool
492   { \seq_use:Nn \l__intexgral_jacobian_seq { } }
493 }

```

10.2 Mode *nested*

\@@_print_nested_integral:

Dans le mode imbriqué, on initie une boucle incrémentée où l'on extrait le i^e élément de chacune des séquences du symbole, de l'intégrande et du jacobien. Après l'itération, on compose toutes les différentielles à nouveau à l'aide de `\seq_use:Nn`.

```
494 \cs_new_protected:Npn \__intexgral_print_nested_integral:
495 {
496   \bool_if:NT \l__intexgral_manual_differential_bool
497   {
498     \cs_set_protected:Npn \differentials
499     {
500       \seq_use:Nn \l__intexgral_differential_seq
501       { \__intexgral_mkern:N \l__intexgral_differential_sep_muskip }
502     }
503   }
504   \int_step_inline:nn { \seq_count:N \l__intexgral_integral_symbol_seq }
505   {
506     \seq_item:Nn \l__intexgral_integral_symbol_seq { ##1 }
507     \__intexgral_mkern:N \l__intexgral_symbol_post_sep_muskip
508     \seq_item:Nn \l__intexgral_integrand_seq { ##1 }
509     \bool_if:NT \l__intexgral_jacobian_bool
510     { \seq_item:Nn \l__intexgral_jacobian_seq { ##1 } }
511   }
512   \__intexgral_mkern:N \l__intexgral_variable_sep_muskip
513   \bool_if:NF \l__intexgral_manual_differential_bool
514   {
515     \seq_use:Nn \l__intexgral_differential_seq
516     { \__intexgral_mkern:N \l__intexgral_differential_sep_muskip }
517   }
518 }
```

\@@_print_nested_integral_inv:

Ici, c'est la séquence des différentielles qui apparaît dans la boucle et celle de l'intégrande qui est utilisée à la fin, après l'itération.

```
519 \cs_new_protected:Npn \__intexgral_print_nested_integral_inv:
520 {
521   \int_step_inline:nn { \seq_count:N \l__intexgral_integral_symbol_seq }
522   {
523     \seq_item:Nn \l__intexgral_integral_symbol_seq { ##1 }
524     \__intexgral_mkern:N \l__intexgral_symbol_post_sep_muskip
525     \seq_item:Nn \l__intexgral_differential_seq { ##1 }
526   }
527   \__intexgral_mkern:N \l__intexgral_variable_sep_muskip
528   \seq_use:Nn \l__intexgral_integrand_seq { }
529   \bool_if:NT \l__intexgral_jacobian_bool
530   { \seq_use:Nn \l__intexgral_jacobian_seq { } }
531 }
```

10.3 Mode *product*

\@@_print_product_integral:

Le mode produit est très similaire au mode imbriqué, dans la mesure où cette fois-ci, toutes les séquences apparaissent dans la boucle.

```
532 \cs_new_protected:Npn \__intexgral_print_product_integral:
533 {
534   \bool_if:NT \l__intexgral_manual_differential_bool
535   {
536     \cs_set_protected:Npn \differentials
537     {
538       \seq_pop_left:NN \l__intexgral_differential_seq \l_tmpa_tl
539       \tl_use:N \l_tmpa_tl
540     }
541   }
542   \int_step_inline:nn { \seq_count:N \l__intexgral_integral_symbol_seq }
543   {
544     \seq_item:Nn \l__intexgral_integral_symbol_seq { ##1 }
545     \__intexgral_mkern:N \l__intexgral_symbol_post_sep_muskip
546     \seq_item:Nn \l__intexgral_integrand_seq { ##1 }
547     \bool_if:NT \l__intexgral_jacobian_bool
548     { \seq_item:Nn \l__intexgral_jacobian_seq { ##1 } }
549     \__intexgral_mkern:N \l__intexgral_variable_sep_muskip
550     \bool_if:NF \l__intexgral_manual_differential_bool
551     { \seq_item:Nn \l__intexgral_differential_seq { ##1 } }
552   }
553 }
```

\@@_print_product_integral_inv:

De même, peu de changements.

```
554 \cs_new_protected:Npn \__intexgral_print_product_integral_inv:
555 {
556   \bool_if:NT \l__intexgral_manual_differential_bool
557   {
558     \cs_set_protected:Npn \differentials
559     {
560       \seq_pop_left:NN \l__intexgral_differential_seq \l_tmpa_tl
561       \tl_use:N \l_tmpa_tl
562     }
563   }
564   \int_step_inline:nn { \seq_count:N \l__intexgral_integral_symbol_seq }
565   {
566     \seq_item:Nn \l__intexgral_integral_symbol_seq { ##1 }
567     \bool_if:NF \l__intexgral_manual_differential_bool
568     {
569       \__intexgral_mkern:N \l__intexgral_symbol_post_sep_muskip
570       \seq_item:Nn \l__intexgral_differential_seq { ##1 }
571     }
572     \__intexgral_mkern:N \l__intexgral_variable_sep_muskip
573     \seq_item:Nn \l__intexgral_integrand_seq { ##1 }
574     \bool_if:NT \l__intexgral_jacobian_bool
575     { \seq_item:Nn \l__intexgral_jacobian_seq { ##1 } }
576   }
577 }
```

10.4 Composition finale

\@@_print_integral:

La macro principale composant l'intégrale incrémente tout d'abord le compteur global permettant de numérotéer chaque intégrale. Ensuite, elle mène les actions préparatoires expliquées précédemment. Enfin, selon le placement des différentielles et du mode d'affichage demandé, l'une des six macros est appelée.

```
578 \cs_new_protected:Nn \__intexgral_print_integral:
579 {
580     \int_gincr:N \g__intexgral_integral_number_int
581     \__intexgral_integral_preconfiguration:
582     \bool_if:NTF \l__intexgral_invert_differential_bool
583     {
584         \str_case:VnF \l__intexgral_display_mode_str
585         {
586             { default } { \__intexgral_print_default_integral_inv: }
587             { nested } { \__intexgral_print_nested_integral_inv: }
588             { product } { \__intexgral_print_product_integral_inv: }
589         }
590         { \__intexgral_print_default_integral_inv: }
591     }
592 {
593     \str_case:VnF \l__intexgral_display_mode_str
594     {
595         { default } { \__intexgral_print_default_integral: }
596         { nested } { \__intexgral_print_nested_integral: }
597         { product } { \__intexgral_print_product_integral: }
598     }
599     { \__intexgral_print_default_integral: }
600 }
601 }
```

11 Déclaration des clés

La création des clés pour `\integral` et `\IntegralSetup` est relativement triviale. Les quelques actions élémentaires pour les clés `limits` et `variables` qui n'ont pas été couvertes par les macros sont effectuées ici.

```
602 \keys_define:nn { integral }
603 {
604     mode .choice:,
605     mode / default .code:n =
606     {
607         \bool_set_false:N \l__intexgral_separate_integral_bool
608         \str_set:Nn \l__intexgral_display_mode_str { default }
609     },
610     mode / nested .code:n =
611     {
612         \bool_set_true:N \l__intexgral_separate_integral_bool
613         \str_set:Nn \l__intexgral_display_mode_str { nested }
614     },
615     mode / product .code:n =
616     {
617         \bool_set_true:N \l__intexgral_separate_integral_bool
618         \str_set:Nn \l__intexgral_display_mode_str { product }
```

```

619     },
620     mode .default:n = { default },
621
622     limits-mode .choice:,
623     limits-mode / limits .code:n =
624         { \tl_set_eq:NN \l__intexgral_limits_style_tl \tex_limits:D },
625     limits-mode / nolimits .code:n =
626         { \tl_set_eq:NN \l__intexgral_limits_style_tl \tex_nolimits:D },
627
628     limits .code:n =
629     {
630         \seq_set_split:Nnn \l__intexgral_limits_seq { ; } { #1 }
631         \__intexgral_set_limits_regular:
632         \__intexgral_generate_integral_sequence:
633     },
634
635     limits* .code:n =
636     {
637         \seq_set_split:Nnn \l__intexgral_limits_seq { ; } { #1 }
638         \__intexgral_set_limits_starred:
639         \__intexgral_generate_integral_sequence:
640     },
641
642     llimit .tl_set:N = \l__intexgral_lower_limit_tl,
643
644     ulimit .tl_set:N = \l__intexgral_upper_limit_tl,
645
646     symbol .code:n =
647     {
648         \cs_if_exist:NTF #1
649         { \tl_set_eq:NN \l__intexgral_integral_symbol_tl #1 }
650         {
651             \msg_warning:nnn { intexgral } { unknown-symb } { #1 }
652             \tl_set_eq:NN \l__intexgral_integral_symbol_tl \int
653         }
654     },
655
656     nint .code:n =
657     {
658         \int_compare:nNnT { #1 } < { 5 }
659         { \msg_warning:nn { intexgral } { use-glyph-instead } }
660         \tl_clear:N \l__intexgral_integral_symbol_tl
661         \int_step_inline:nn { #1 }
662         {
663             \tl_put_right:Nn \l__intexgral_integral_symbol_tl
664             { \int }
665             \int_compare:nNnT { ##1 } < { #1 }
666             {
667                 \tl_put_right:Nn \l__intexgral_integral_symbol_tl
668                 {
669                     \__intexgral_mathchoice:nnnn
670                     { \tex_mkern:D -12mu \scan_stop: }
671                     { \tex_mkern:D -8mu \scan_stop: }
672                     { \tex_mkern:D -4mu \scan_stop: }
673                     { \tex_mkern:D -2mu \scan_stop: }
674                 }
675             }
676         }

```

```

677     },
678
679     domain .code:n =
680     {
681         \tl_set:Nn \l_tmpa_tl { #1 }
682         \seq_set_split:NnV \l__intexgral_domain_seq { * } \l_tmpa_tl
683         \seq_map_inline:Nn \l__intexgral_domain_seq
684         {
685             \tl_if_empty:NF \l__intexgral_lower_limit_tl
686             { \tl_put_right:Nn \l__intexgral_lower_limit_tl { \times } }
687             \tl_set:Ne \l__intexgral_domain_char_tl
688             { \exp_args:Ne \str_uppercase:n { \tl_head:n { ##1 } } }
689             \tl_set:Ne \l__intexgral_domain_dimension_tl
690             { \tl_tail:n { ##1 } }
691             \tl_put_right:Ne \l__intexgral_lower_limit_tl
692             {
693                 \exp_not:V \l__intexgral_domain_style_tl
694                 { \l__intexgral_domain_char_tl }
695                 \c_math_superscript_token { \l__intexgral_domain_dimension_tl }
696             }
697         }
698     },
699
700     domain* .code:n =
701     {
702         \tl_set:Nn \l_tmpa_tl { #1 }
703         \seq_set_split:NnV \l__intexgral_domain_seq { * } \l_tmpa_tl
704         \seq_map_inline:Nn \l__intexgral_domain_seq
705         {
706             \tl_if_empty:NF \l__intexgral_lower_limit_tl
707             { \tl_put_right:Nn \l__intexgral_lower_limit_tl { \times } }
708             \tl_set:Ne \l__intexgral_domain_char_tl
709             { \exp_args:Ne \str_uppercase:n { \tl_head:n { ##1 } } }
710             \tl_set:Ne \l__intexgral_domain_dimension_tl
711             { \tl_tail:n { ##1 } }
712             \tl_put_right:Ne \l__intexgral_lower_limit_tl
713             {
714                 \exp_not:V \l__intexgral_domain_style_tl
715                 { \l__intexgral_domain_char_tl }
716                 \c_math_subscript_token { \l__intexgral_domain_dimension_tl }
717             }
718         }
719     },
720
721     boundary .code:n =
722     { \tl_set:Nn \l__intexgral_lower_limit_tl { \partial #1 } },
723
724     variables .code:n =
725     {
726         \bool_set_true:N \l__intexgral_custom_variables_bool
727         \str_if_eq:nnTF { #1 } { none }
728         { \bool_set_true:N \l__intexgral_deactivate_variables_bool }
729         {
730             \prop_get:NnNTF
731             \g__intexgral_differential_group_keyword_prop { #1 } \l_tmpa_tl
732             {
733                 \seq_set_split:NnV
734                 \l__intexgral_variables_seq

```

```

735         { , }
736         \l_tmpa_tl
737         \seq_if_exist:cTF { g__intexgral_#1_jacobian_seq }
738         {
739             \seq_set_eq:Nc
740             \l__intexgral_jacobian_seq
741             { g__intexgral_#1_jacobian_seq }
742         }
743         { \msg_warning:nnn { intexgral } { no-jacobian } { #1 } }
744     }
745     { \seq_set_split:Nnn \l__intexgral_variables_seq { , } { #1 } }
746 }
747 },
748
749 jacobian .code:n =
750 { \bool_set_true:N \l__intexgral_jacobian_bool },
751 diff-vec .code:n =
752 { \bool_set_true:N \l__intexgral_vectorial_differential_bool },
753 diff-symb .tl_set:N = \l__intexgral_differential_symbol_tl,
754 diff-order .clist_set:N = \l__intexgral_differential_order_clist,
755 }
756
757 \keys_define:nn { IntegralSetup }
758 {
759     diff-symb .tl_set:N = \l__intexgral_differential_symbol_tl,
760     defaultvar .code:n =
761     {
762         \seq_set_split:Nnn
763         \l__intexgral_default_variables_seq
764         { , } { #1 }
765     },
766     defaultvar* .code:n =
767     {
768         \seq_set_split:Nnn
769         \l__intexgral_default_vectorial_variables_seq
770         { , } { #1 }
771     },
772     postsymbsep .muskip_set:N = \l__intexgral_symbol_post_sep_muskip,
773     varsep .muskip_set:N = \l__intexgral_variable_sep_muskip,
774     diffsep .muskip_set:N = \l__intexgral_differential_sep_muskip,
775     vectorstyle .tl_set:N = \l__intexgral_vectorial_differential_style_tl,
776     domainstyle .tl_set:N = \l__intexgral_domain_style_tl,
777     innersymbsep .tl_set:N = \l__intexgral_symbol_inner_sep_muskip,
778     novar .bool_set:N = \l__intexgral_deactivate_variables_bool,
779 }

```

12 Macros d'interface utilisateur

12.1 Mots-clés pour les bornes

`\@@_new_limits_group:nn` `\@@_new_limits_group:nn` `{<mot-clé>}` `{<bornes>}`

Pour le fonctionnement de `\NewLimitsKeyword`, on enregistre le mot clé et sa valeur dans une liste de propriétés. Une particularité qui se doit néanmoins d'être expliquée est qu'il y a un double inversement des bornes: au moment de la création du mot-clé, et au moment d'en extraire son contenu (cf. `\@@_parse_limits:nn`). La raison est qu'il est difficile de rattacher à un mot clé le statut de *borne inversée*. Ainsi, quelle que soit la convention choisie, les bornes sont stockées *dans le même ordre*. Ceci permet également aux mots-clés définis par le package d'être dans le bon ordre, même si la convention est changée après son chargement.

```
780 \cs_new_protected:Npn \__intexgral_new_limits_group:nn #1#2
781 {
782     \prop_gput:Nne \g__intexgral_limits_keyword_prop { #1 }
783     {
784         \bool_if:NTF \l__intexgral_invert_limits_bool
785             { \__intexgral_invert_limits:w #2\q_stop }
786             { #2 }
787     }
788 }
```

`\NewLimitsKeyword` Cette macro est documentée en sous-sous-section 4.2.1.

```
789 \NewDocumentCommand\NewLimitsKeyword{ m m }
790 {
791     \prop_if_in:NnTF \g__intexgral_limits_keyword_prop { #1 }
792     { \msg_error:nnn { intexgral } { lim-group-alr-def } { #1 } }
793     { \__intexgral_new_limits_group:nn { #1 } { #2 } }
794 }
```

`\RenewLimitsKeyword` Cette macro est documentée en sous-sous-section 4.2.1.

```
795 \NewDocumentCommand\RenewLimitsKeyword{ m m }
796 {
797     \prop_pop:NnNTF \g__intexgral_limits_keyword_prop { #1 } \l_tmpa_tl
798     { \__intexgral_new_limits_group:nn { #1 } { #2 } }
799     { \msg_error:nnn { intexgral } { lim-group-not-def } { #1 } }
800 }
```

`\ProvideLimitsKeyword` Cette macro est documentée en sous-sous-section 4.2.1.

```
801 \NewDocumentCommand\ProvideLimitsKeyword{ m m }
802 {
803     \prop_if_in:NnF \g__intexgral_limits_keyword_prop { #1 }
804     { \__intexgral_new_limits_group:nn { #1 } { #2 } }
805 }
```

`\DeclareLimitsKeyword`

Cette macro est documentée en sous-sous-section 4.2.1.

```
806 \NewDocumentCommand\DeclareLimitsKeyword{ m m }
807 {
808     \prop_remove:Nn \g__intexgral_limits_keyword_prop { #1 }
809     \__intexgral_new_limits_group:nn { #1 } { #2 }
810 }
```

12.2 Mots-clés pour les variables

`\@@_new_variables_group:nnn` `\@@_new_variables_group:nnn` `{<mot-clé>}` `{<variable>}` [`<jacobien>`]

Tout comme pour les bornes, on enregistre les groupes de différentielles dans une liste de propriétés, avec leur nom, leur valeur et éventuellement leur jacobien associé.

```
811 \cs_new_protected:Npn \__intexgral_new_variables_group:nnn #1#2#3
812 {
813     \prop_gput:Nnn \g__intexgral_differential_group_keyword_prop
814         { #1 } { #2 }
815     \tl_if_blank:nF { #3 }
816     {
817         \seq_new:c { g__intexgral_#1_jacobian_seq }
818         \seq_gset_split:cnn { g__intexgral_#1_jacobian_seq } { , } { #3 }
819     }
820 }
```

`\NewVariableKeyword`

Cette macro est documentée en sous-sous-section 4.3.1.

```
821 \NewDocumentCommand\NewVariableKeyword{ m m o }
822 {
823     \prop_if_in:NnTF \g__intexgral_differential_group_keyword_prop { #1 }
824         { \msg_error:nnn { intexgral } { diff-group-alr-def } { #1 } }
825         { \__intexgral_new_variables_group:nnn { #1 } { #2 } { #3 } }
826 }
```

`\RenewVariableKeyword`

Cette macro est documentée en sous-sous-section 4.3.1.

```
827 \NewDocumentCommand\RenewVariableKeyword{ m m o }
828 {
829     \prop_pop:NnNTF \g__intexgral_differential_group_keyword_prop { #1 }
830         \l_tmpa_tl
831         {
832             \seq_gclear:c { g__intexgral_#1_jacobian_seq }
833             \__intexgral_new_variables_group:nnn { #1 } { #2 } { #3 }
834         }
835     { \msg_error:nnn { intexgral } { diff-group-not-def } { #1 } }
836 }
```

`\ProvideVariableKeyword`

Cette macro est documentée en sous-sous-section 4.3.1.

```

837 \NewDocumentCommand\ProvideVariableKeyword{ m m o }
838 {
839     \prop_if_in:NnF \g__intexgral_differential_group_keyword_prop { #1 }
840     { \__intexgral_new_variables_group:nnn { #1 } { #2 } { #3 } }
841 }

```

`\DeclareVariableKeyword`

Cette macro est documentée en sous-sous-section 4.3.1.

```

842 \NewDocumentCommand\DeclareVariableKeyword{ m m o }
843 {
844     \prop_remove:Nn \g__intexgral_differential_group_keyword_prop { #1 }
845     \seq_gclear:c { g__intexgral_#1_jacobian_seq }
846     \__intexgral_new_variables_group:nnn { #1 } { #2 } { #3 }
847 }

```

12.3 Mots-clés pour les symboles

Cette section ne nécessite pas de macro intermédiaire, puisqu'on modifie directement les clés du module *integral*.

`\NewSymbolKeyword`

Cette macro est documentée en sous-section 4.4.

```

848 \NewDocumentCommand\NewSymbolKeyword{ m m }
849 {
850     \prop_if_in:NnT \g__intexgral_limits_keyword_prop { #1 }
851     { \msg_warning:nnn { intexgral } { key-exists-for-lim } { #1 } }
852     \keys_if_exist:nnTF { integral } { #1 }
853     { \msg_error:nnn { intexgral } { symb-key-alr-def } { #1 } }
854     {
855         \keys_define:nn { integral }
856         {
857             #1 .meta:n =
858             {
859                 symbol=#2,
860                 llimit=##1
861             },
862         }
863     }
864 }

```

`\RenewSymbolKeyword`

Cette macro est documentée en sous-section 4.4.

```

865 \NewDocumentCommand\RenewSymbolKeyword{ m m }
866 {
867     \prop_if_in:NnT \g__intexgral_limits_keyword_prop { #1 }
868     { \msg_warning:nnn { intexgral } { key-exists-for-lim } { #1 } }
869     \keys_if_exist:nnTF { integral } { #1 }
870     {
871         \keys_define:nn { integral }
872         {
873             #1 .undefine:
874             #1 .meta:n =
875             {

```

```

876             symbol=#2,
877             llimit=##1
878         }
879     }
880 }
881 { \msg_error:nnn { intexgral } { symb-key-not-def } }
882 }

```

\ProvideSymbolKeyword

Cette macro est documentée en sous-section 4.4.

```

883 \NewDocumentCommand\ProvideSymbolKeyword{ m m }
884 {
885     \prop_if_in:NnT \g__intexgral_limits_keyword_prop { #1 }
886     { \msg_warning:nnn { intexgral } { key-exists-for-lim } { #1 } }
887     \keys_if_exist:nnF { integral } { #1 }
888     {
889         \keys_define:nn { integral }
890         {
891             #1 .meta:n =
892             {
893                 symbol=#2,
894                 llimit=##1
895             },
896         }
897     }
898 }

```

\DeclareSymbolKeyword

Cette macro est documentée en sous-section 4.4.

```

899 \NewDocumentCommand\DeclareSymbolKeyword{ m m }
900 {
901     \keys_define:nn { integral }
902     {
903         #1 .undefine:
904         #1 .meta:n =
905         {
906             symbol=#2,
907             llimit=##1
908         },
909     }
910 }

```

12.4 Macros de configuration

\IntegralSetup

Cette macro est documentée en section 6.

```

911 \NewDocumentCommand\IntegralSetup{ m }
912 { \keys_set:nn { IntegralSetup } { #1 } }

```

\intexgralsetup

De même, mais sans oublier de restreindre l'utilisation au préambule.
Cette macro est documentée en section 1.

```

913 \NewDocumentCommand\intexgralsetup{ m }
914   { \keys_set:nn { intexgral } { #1 } }
915 \@onlypreamble\intexgralsetup

```

\invertdiff

Définition assez triviale qui se contente d'inverser le booléen vu au début de l'implémentation.

```

916 \NewDocumentCommand\invertdiff{ }
917   { \bool_set_inverse:N \l__intexgral_invert_differential_bool }

```

\integral

La macro finale ne tient qu'en quelques lignes. Si un argument optionnel est fourni, elle effectue les actions nécessaires pour juger de la nature de celui-ci (syntaxe spéciale ou non), et assigne les clés en conséquence. Elle assigne ensuite l'argument principal au token correspondant, avant d'appeler la macro centrale qui compose l'intégrale. Le tout est exécuté dans un groupe afin que toutes les modifications demeurent locales.

Cette macro est documentée en section 2.

```

918 \NewDocumentCommand\integral{ 0{} m }
919   {
920     \group_begin:
921     \tl_if_empty:nF { #1 }
922       {
923         \__intexgral_extract_first_key_name:n { #1 }
924         \__intexgral_parse_integral_keys:n { #1 }
925       }
926     \tl_set:Nn \l__intexgral_integrand_tl { #2 }
927     \__intexgral_print_integral:
928     \group_end:
929   }

```

13 Configuration du package

La dernière partie du package s'occupe de définir les mots-clés par défaut pour les symboles, les bornes et les variables, ainsi que les paramètres par défaut pour la composition des intégrales.

13.1 Symbole

```

930 \NewSymbolKeyword{single}    {\int}
931 \NewSymbolKeyword{double}    {\iint}
932 \NewSymbolKeyword{triple}    {\iiint}
933 \NewSymbolKeyword{quadruple} {\iiiiint}
934 \NewSymbolKeyword{contour}   {\oint}
935 \NewSymbolKeyword{surface}    {\oiint}
936 \NewSymbolKeyword{volume}    {\oiint}

```

13.2 Bornes

```

937 \NewLimitsKeyword{ab}        {a, b}
938 \NewLimitsKeyword{real}      {-\infty, +\infty}
939 \NewLimitsKeyword{positive}  {0, +\infty}
940 \NewLimitsKeyword{negative}  {-\infty, 0}
941 \NewLimitsKeyword{unit}      {0, 1}
942 \NewLimitsKeyword{circle}    {0, 2\pi}

```

```

943 \NewLimitsKeyword{scircle}    {0, \pi}
944 \NewLimitsKeyword{qcircle}    {0, \pi/2}
945 \NewLimitsKeyword{height}     {0, H}
946 \NewLimitsKeyword{radius}     {0, R}
947 \NewLimitsKeyword{length}     {0, L}
948 \NewLimitsKeyword{time}       {0, T}

```

13.3 Variables

```

949 \NewVariableKeyword{cartesian} {x, y, z}
950 \NewVariableKeyword{planar}    {x, y}
951 \NewVariableKeyword{polar}     {r, \theta} [r]
952 \NewVariableKeyword{cylindrical} {r, \theta, z} [r]
953 \NewVariableKeyword{cylindrical*} {\rho, \theta, z} [\rho]
954 \NewVariableKeyword{spherical} {r, \theta, \phi} [r^2, \sin\theta]

```

13.4 Paramètres par défaut

```

955 \IntegralSetup{
956   diff-symb   = \l__intexgral_default_differential_symbol_tl,
957   defaultvar  = {x},
958   defaultvar* = {r},
959   varsep      = \thinmuskip,
960   diffsep     = \thinmuskip,
961   innersymbsep = {-5mu},
962   postsymbsep = {-1mu},
963   vectorstyle = \vec,
964   domainstyle = \mathbb,
965   novar       = false
966 }
967 \</package>

```

Index

Symbols

_ 93, 97, 101, 105, 109, 113, 118

B

bool commands:

\bool_case:nTF 212
 \bool_if:NTF 216, 250, 267, 283,
 312, 314, 340, 421, 438, 446,
 458, 461, 463, 471, 482, 488, 491,
 496, 509, 513, 529, 534, 547,
 550, 556, 567, 574, 582, 784
 \bool_if:nTF 409
 \bool_lazy_and:nnTF 203
 \bool_new:N 28, 29, 30, 164, 165,
 166, 167, 168
 \bool_set_false:N . . . 442, 607
 \bool_set_inverse:N 917
 \bool_set_true:N . 441, 612, 617,
 726, 728, 750, 752

C

\c 440

\cdot 462, 489

clist commands:

\clist_if_empty:NTF 331
 \clist_item:Nn 335
 \clist_new:N 172

cs commands:

\cs_generate_variant:Nn . 143,
 144, 145, 146, 147, 148, 149, 150
 \cs_if_exist:NTF 27, 648
 \cs_new:Nn 83
 \cs_new:Npn . 183, 184, 209, 210,
 346
 \cs_new_eq:NN 182
 \cs_new_protected:Nn 232, 293,
 310, 419, 578
 \cs_new_protected:Npn
 . 180, 244, 281, 347, 361, 444,
 469, 494, 519, 532, 554, 780, 811
 \cs_set_protected:Npn 448, 473,
 498, 536, 558

D

\DeclareLimitsKeyword 806
 \DeclareSymbolKeyword 899
 \DeclareVariableKeyword 842
 \def 6, 7, 8, 9
 \differentials . . 448, 473, 498, 536,
 558

E

exp commands:

\exp_args:Ne 204, 205, 688, 709
 \exp_args:NNne 371
 \exp_last_unbraced:Ne . . . 218
 \exp_last_unbraced:NV . . . 355
 \exp_not:n . . 285, 286, 289, 290,
 330, 341, 693, 714

G

group commands:

\group_begin: 920
 \group_end: 928

H

hook commands:

\hook_gput_code:nnn 26

I

\iiiint 933

\iiint 932

\iint 931

\infty 272, 276, 938, 939, 940

\int 127, 153, 652, 664, 930

int commands:

\int_compare:nNnTF . . 380, 387,
 658, 665
 \int_compare_p:nNn 411
 \int_gincr:N 580
 \int_gzero_new:N 82
 \int_set:Nn 295
 \int_step_inline:nn . 301, 504,
 521, 542, 564, 661
 \int_use:N 86
 \l_tmpa_int 295, 301

\integral 918

\IntegralSetup 911, 955

intexgral internal commands:

\l__intexgral_custom_variables_-
 bool 165, 312,
 726
 \l__intexgral_deactivate_-
 variables_bool 29, 438, 728,
 778
 \l__intexgral_default_differential_-
 symbol_tl . 31, 55, 60, 69, 74,
 956
 \l__intexgral_default_variables_-
 seq 170, 323, 374, 384,
 763

<code>\l__intexgral_default_vectorial_-</code> <code>variables_seq</code> ... 171, 318, 769	<code>\l__intexgral_integral_-</code> <code>symbol_tl</code> 152, 153, 186, 649, 652, 660, 663, 667
<code>\g__intexgral_differential_-</code> <code>group_keyword_prop</code> 174, 731, 813, 823, 829, 839, 844	<code>\l__intexgral_integrand_seq</code> . . 194, 424, 430, 457, 490, 508, 528, 546, 573
<code>\l__intexgral_differential_-</code> <code>order_clist</code> .. 172, 331, 336, 754	<code>\l__intexgral_integrand_tl</code> 151, 426, 431, 440, 926
<code>\l__intexgral_differential_-</code> <code>sep_muskip</code> 178, 451, 466, 476, 485, 501, 516, 774	<code>\l__intexgral_invert_differential_-</code> <code>bool</code> 30, 39, 582, 917
<code>\l__intexgral_differential_-</code> <code>seq</code> 196, 328, 450, 465, 475, 484, 500, 515, 525, 538, 551, 560, 570	<code>__intexgral_invert_limits:w</code> 183, 218, 785
<code>\l__intexgral_differential_-</code> <code>symbol_tl</code> ... 156, 330, 753, 759	<code>\l__intexgral_invert_limits_-</code> <code>bool</code> 28, 35, 216, 250, 267, 283, 784
<code>\l__intexgral_display_mode_-</code> <code>str</code> .. 179, 584, 593, 608, 613, 618	<code>\l__intexgral_jacobian_bool</code> . . 168, 458, 491, 509, 529, 547, 574, 750
<code>\l__intexgral_domain_char_tl</code> .. 159, 687, 694, 708, 715	<code>\l__intexgral_jacobian_seq</code> .. . 195, 459, 492, 510, 530, 548, 575, 740
<code>\l__intexgral_domain_dimension_-</code> <code>tl</code> 160, 689, 695, 710, 716	<code>\l__intexgral_key_name_tl</code> 163, 353, 359, 363
<code>\l__intexgral_domain_seq</code> 169, 682, 683, 703, 704	<code>\c__intexgral_left_bracket_tl</code> 161, 273, 276
<code>\l__intexgral_domain_style_tl</code> 158, 693, 714, 776	<code>\g__intexgral_limits_keyword_-</code> <code>prop</code> 173, 214, 219, 222, 782, 791, 797, 803, 808, 850, 867, 885
<code>__intexgral_extract_first_-</code> <code>key_name:n</code> 347, 923	<code>\l__intexgral_limits_seq</code> 198, 234, 246, 630, 637
<code>__intexgral_general_integral_-</code> <code>symbol:</code> 184, 307, 436	<code>\l__intexgral_limits_style_tl</code> .. 32, 45, 47, 187, 624, 626
<code>__intexgral_generate_integral_-</code> <code>sequence:</code> 293, 632, 639	<code>\l__intexgral_lower_limit_tl</code> 154, 189, 252, 260, 274, 285, 289, 642, 685, 686, 691, 706, 707, 712, 722
<code>__intexgral_has_pm:n</code> ... 201	<code>\l__intexgral_lower_limits_-</code> <code>seq</code> . 200, 238, 269, 297, 299, 304
<code>__intexgral_has_pm_p:n</code> . 224	<code>\l__intexgral_manual_differential_-</code> <code>bool</code> 164, 441, 442, 446, 463, 471, 482, 496, 513, 534, 550, 556, 567
<code>\g__intexgral_integral_-</code> <code>number_int</code> 82, 86, 580	<code>__intexgral_mathchoice:nnnn</code> 182, 669
<code>__intexgral_integral_preconfiguration\</code> 419, 581	<code>__intexgral_mkern:N</code> . 180, 451, 455, 456, 460, 466, 476, 480, 481, 485, 487, 501, 507, 512, 516, 524, 527, 545, 549, 569, 572
<code>\l__intexgral_integral_-</code> <code>symbol_seq</code> 193, 306, 433, 435, 454, 479, 504, 506, 521, 523, 542, 544, 564, 566	

`\__intexgral_new_limits_-`
`group:nn . 780,793,798,804,`
`809`
`\__intexgral_new_variables_-`
`group:nnn 811,825,833,840,`
`846`
`\__intexgral_parse_integral_-`
`keys:n . . . . . 361,`
`924`
`\__intexgral_parse_limits:n .`
`. . . . . 210,237,249`
`\__intexgral_parse_variables:`
`. . . . . 310,439`
`\__intexgral_print_default_-`
`integral: . . . . . 444,595,`
`599`
`\__intexgral_print_default_-`
`integral_inv: . . 469,586,`
`590`
`\__intexgral_print_integral:`
`. . . . . 578,927`
`\__intexgral_print_nested_-`
`integral: . . . . . 494,`
`596`
`\__intexgral_print_nested_-`
`integral_inv: . . . . . 519,`
`587`
`\__intexgral_print_product_-`
`integral: . . . . . 532,`
`597`
`\__intexgral_print_product_-`
`integral_inv: . . . . . 554,`
`588`
`\__intexgral_retrieve_key_-`
`name:w . . . . . 346,`
`356`
`\c__intexgral_right_bracket_-`
`tl . . . . . 162,272,`
`277`
`\l__intexgral_separate_-`
`integral_bool 166,421,607,`
`612,617`
`\__intexgral_set_limits:nn . .`
`. . . . . 281,303`
`\__intexgral_set_limits_-`
`regular: . . . . . 232,`
`631`
`\__intexgral_set_limits_-`
`starred: . . . . . 244,`
`638`
`\l__intexgral_symbol_inner_-`
`sep_muskip . . 175,455,480,`
`777`
`\l__intexgral_symbol_post_-`
`sep_muskip 176,456,481,507,`
`524,545,569,772`
`\__intexgral_trim_pm:w . . 209,`
`226,227`
`\l__intexgral_upper_limit_tl`
`. . 155,191,286,290,644`
`\l__intexgral_upper_limits_-`
`seq . . . . . 199,240,268,298,`
`305`
`\l__intexgral_variable_sep_-`
`muskip 177,460,487,512,527,`
`549,572,773`
`\l__intexgral_variables_seq .`
`. 197,317,322,326,734,745`
`\l__intexgral_vectorial_-`
`differential_bool 167,314,`
`340,461,488,752`
`\l__intexgral_vectorial_-`
`differential_style_tl 157,`
`341,775`
`\__intexgral_warning_msg_-`
`header: . . 83,118,125,132,`
`139`
`\intexgralsetup . . . . . 913,915`
`\invertdiff . . . . . 916`
`iow commands:`
`\iow_newline: . . . . . 19,20,89`

K

`keys commands:`
`\keys_define:nn . . 33,602,757,`
`855,871,889,901`
`\keys_if_exist:nnTF . 145,363,`
`852,869,887`
`\keys_set:nn 364,392,912,914`

M

`\mathbb . . . . . 27,964`
`\mathchoice . . . . . 182`
`\mathnormal . . . . . 56,75`
`\mathrm . . . . . 61,70`
`msg commands:`
`\msg_critical:nn . . . . . 25`
`\msg_error:nnn . . 792,799,824,`
`835,853,881`
`\msg_line_context: . . . . . 87`
`\msg_new:nnn 16,115,123,130,137`
`\msg_new:nnnn 91,95,99,103,107,`
`111`

\msg_see_documentation_text:n
 21
 \msg_warning:nn 659
 \msg_warning:nnn . 651, 743, 851,
 868, 886

muskip commands:

\muskip_new:N 175, 176, 177, 178

N

\NeedsTeXFormat 3
 \NewDocumentCommand
 . 789, 795, 801, 806, 821, 827,
 837, 842, 848, 865, 883, 899,
 911, 913, 916, 918
 \NewLimitsKeyword 113,
 118, 789, 937, 938, 939, 940, 941,
 942, 943, 944, 945, 946, 947,
 948
 \NewSymbolKeyword 97, 848, 930, 931,
 932, 933, 934, 935, 936
 \NewVariableKeyword . 821, 949, 950,
 951, 952, 953, 954
 \NewVarKeyword 105

O

\oiint 936
 \oint 935
 \oint 934

P

\partial 722
 \phi 954
 \pi 942, 943, 944
 prg commands:

\prg_new_conditional:Npnn 201
 \prg_return_false: 207
 \prg_return_true: 206

\ProcessKeyOptions 81
 prop commands:

\prop_get:NnNTF 730
 \prop_gput:Nnn ... 148, 782, 813
 \prop_if_in:NnTF 791, 803, 823,
 839, 850, 867, 885
 \prop_if_in_p:Nn 214
 \prop_item:Nn 219, 222
 \prop_new:N 173, 174
 \prop_pop:NnNTF 797, 829
 \prop_remove:Nn 808, 844

\ProvideLimitsKeyword 801
 \ProvidesExplPackage 11
 \ProvideSymbolKeyword 883
 \ProvideVariableKeyword 837

Q

quark commands:

\q_stop 183, 209, 220, 226, 227,
 346, 356, 785

R

regex commands:

\regex_split:nnN 366
 \RenewLimitsKeyword 109, 795
 \RenewSymbolKeyword 93, 865
 \RenewVariableKeyword 827
 \RenewVarKeyword 101
 \RequirePackage 4, 27
 \rho 953

S

scan commands:

\scan_stop: . 181, 670, 671, 672,
 673

seq commands:

\seq_count:N 298, 299, 380, 387,
 411, 504, 521, 542, 564
 \seq_gclear:N 832, 845
 \seq_gset_split:Nnn . 149, 818
 \seq_if_empty:NTF ... 297, 433
 \seq_if_exist:NTF 737
 \seq_item:Nn
 . 239, 241, 254, 256, 262, 264,
 271, 275, 304, 305, 350, 368,
 391, 394, 395, 398, 413, 506,
 508, 510, 523, 525, 544, 546,
 548, 551, 566, 570, 573, 575

\seq_map_indexed_inline:Nn ..
 246, 326

\seq_map_inline:Nn . 234, 683,
 704

\seq_new:N 169, 170, 171, 193, 194,
 195, 196, 197, 198, 199, 200, 817

\seq_pop_left:NN 538, 560

\seq_put_right:Nn 238,
 240, 268, 269, 306, 328, 382,
 388, 429, 435

\seq_set_eq:NN ... 316, 321, 739

\seq_set_item:Nnn 371

\seq_set_split:Nnn 150,
 236, 248, 349, 389, 423, 630,
 637, 682, 703, 733, 745, 762, 768

\seq_use:Nn 146, 373,
 384, 450, 454, 457, 459, 465,
 475, 479, 484, 490, 492, 500,
 515, 528, 530

`\l_tmpa_seq` . 236, 239, 241, 248,
 254, 256, 262, 264, 271, 275,
 349, 350, 366, 368, 371, 380,
 383, 387, 388, 391, 394, 398
`\l_tmpb_seq` . . 389, 395, 411, 413
`\sin` 954
 str commands:
`\str_case:nnTF` . . 398, 584, 593
`\str_case_e:nnTF` 271, 275
`\str_if_eq:nnTF` . . 143, 144, 367,
 727
`\str_if_eq_p:nn` . 147, 204, 205,
 413
`\str_item:nn` 204, 205
`\str_new:N` 179
`\str_set:Nn` 608, 613, 618
`\str_uppercase:n` 688, 709

T
 $\TeX$ and $\LaTeX 2_{\epsilon}$ commands:
`\@ifpackagelater` 24
`\@onlypreamble` 915
`\intexgral@date` 8, 13
`\intexgral@description` . . 9, 15
`\intexgral@module` 6, 12
`\intexgral@version` 7, 14
 tex commands:
`\tex_left:D` 272, 273
`\tex_limits:D` 45, 624
`\tex_mathpunct:D` 255, 263
`\tex_mkern:D` 181, 670, 671, 672,
 673
`\tex_nolimits:D` 47, 626
`\tex_right:D` 276, 277
`\tex_unkern:D` 462, 489
`\theta` 951, 952, 953, 954
`\thinmuskip` 959, 960
`\times` 686, 707

tl commands:
`\c_space_tl` 86
`\tl_clear:N` 660
`\tl_const:Nn` 161, 162
`\tl_head:n` 688, 709
`\tl_if_blank:nTF` 815
`\tl_if_empty:NnTF` 685, 706
`\tl_if_empty:nTF` 921
`\tl_if_in:NnTF` 351
`\tl_if_regex_match:nnTF` . 440
`\tl_new:N` 31, 32, 151, 152, 154, 155,
 156, 157, 158, 159, 160, 163
`\tl_put_right:Nn` 663, 667, 686,
 691, 707, 712
`\tl_set:Nn` 55,
 60, 69, 74, 252, 260, 285, 286,
 289, 290, 350, 353, 681, 687,
 689, 702, 708, 710, 722, 926
`\tl_set_eq:NN` . 45, 47, 153, 359,
 624, 626, 649, 652
`\tl_tail:n` 690, 711
`\tl_trim_spaces:n` . 92, 96, 100,
 104, 108, 112, 126
`\tl_use:N` 274, 539, 561
`\l_tmpa_tl` 350,
 351, 356, 359, 538, 539, 560,
 561, 681, 682, 702, 703, 731, 736,
 797, 830
 token commands:
`\c_math_subscript_token` . 188,
 716
`\c_math_superscript_token` 190,
 333, 695
`\token_to_str:N` 93, 97, 101, 105,
 109, 118, 127

V

`\vec` 963