L'extension intexgral

v4.0.0

Valentin Dao*

6 juin 2026

Résumé

Tandis que la composition d'intégrales est très fréquente en 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 ⟨clé=valeur⟩. Contrairement à la méthode classique, où l'utilisateur choisit l'ordre des bornes avec les caractères actifs _ et ^, le package a dû fixer une convention. Ainsi, pour les deux clés gérant les bornes qui seront présentées, l'entrée supposée sera _⟨borne inférieure⟩^⟨borne supérieure⟩. Ce package étant écrit en expl3, il dépend donc de *l3kernel*.

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

Table des matières

Licence

Polices

Chronicle Text G3 Roman

Whitney-Medium

Monaspace Argon Medium

Dépôt

Voir dépôt GitHub.

Remerciements

Un grand merci à Plante et Slurpy de m'avoir accompagné dans cette aventure que fut l'apprentissage de 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. 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'option 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}
```

$$\int x \,\mathrm{d}x \tag{1}$$

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 en suivant la convention édictée dans le résumé. 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⟩).

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

$$\int_{1}^{10} f(x)\,\mathrm{d}x \tag{2}$$

L'avantage notable de la clé `limits` est qu'on peut lui spécifier les bornes d'intégration 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}
```

$$\int_{1}^{2}\int_{3}^{4} f(x,y)\,\mathrm{d}x\,\mathrm{d}y \tag{3}$$

Il est également possible de renseigner des bornes prédéfinies à l'aide de mots-clés (voir sous-section 4.2). 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 $\pm$`\infty`

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

$$\int_{[1,10]} f(x)\,\mathrm{d}x \tag{4}$$

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

3.2 Modes d'affichage

mode

mode=⟨default | nested | product⟩

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 écrits.

`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 sera, d'une certaine manière, *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}
```

$$\int_1^2\int_3^4\int_5^6 xyz\,\mathrm{d}x\,\mathrm{d}y\,\mathrm{d}z \tag{5}$$

MODE nested

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

$$\int_1^2 x\int_3^4 y\int_5^6 z\,\mathrm{d}z\,\mathrm{d}y\,\mathrm{d}x \tag{6}$$

MODE product

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

$$\int_1^2 x\,\mathrm{d}x \int_3^4 y\,\mathrm{d}y \int_5^6 z\,\mathrm{d}z \tag{7}$$

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. 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 : *amsmath* colle deux `\int` ensemble avec un certain crénage pour définir `\iint`, tandis qu'*unicode-math* définit le vrai glyphe `U+222C`.

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

$$\iint_S f(x,y)\,\mathrm{d}x\,\mathrm{d}y \tag{8}$$

3.3.2 Générer beaucoup d'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}
```

$$\idotsint_{\Omega} f(x_1, x_2, x_3, x_4, x_5)\,\mathrm{d}x_1\,\mathrm{d}x_2\,\mathrm{d}x_3\,\mathrm{d}x_4\,\mathrm{d}x_5 \tag{9}$$

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}
```

$$\int_{x^2+y^2\leq 1} f(x, y)\,\mathrm{d}x\,\mathrm{d}y \tag{10}$$

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}
```

$$\iint_{\mathbb{R}\times\mathbb{R}} xy\,\mathrm{d}x\,\mathrm{d}y \tag{11}$$

`boundary` `boundary=⟨borne inférieure⟩`

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

```
\begin{equation}
  \integral[symbol=\oint, boundary=S, diff-vec]{G(\vec r)}
\end{equation}
```

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

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'une intégrale 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-section 4.3)

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

```
\begin{equation}
  \integral[variables=t]{t^2}
\end{equation}
```

$$\int t^2 \, \mathrm{d}t \tag{13}$$

3.4.2 Inclure le jacobien

`jacobian` `jacobian`

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

```
\begin{equation}
  \integral[limits={0, R; 0, 2\pi; 0, \pi}, variables=spherical, mode=
   product, jacobian]{}
\end{equation}
```

$$\int_0^R r^2 \, \mathrm{d}r \int_0^{2\pi} \sin\theta \, \mathrm{d}\theta \int_0^{\pi} \mathrm{d}\phi \tag{14}$$

3.4.3 Modifier le symbole des différentielles

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

Cette clé permet de modifier le symbole ou bien le style des différentielles.

```
\begin{equation}
  \integral[variables={q(t)}, diff-symb=\mathcal{D}]{e^{+\dfrac{i S[q(t)]}{
   \hbar}}}
\end{equation}
```

$$\int e^{+\frac{iS[q(t)]}{\hbar}} \mathcal{D}q(t) \tag{15}$$

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.

```
\IntegralSetup{vectorstyle=\mathbf}
  \begin{equation}
    W = \integral[single=C, variables=s, diff-vec]{\mathbf F}
  \end{equation}
```

$$W = \int_C \mathbf{F} \cdot \mathrm{d}\mathbf{s} \tag{16}$$

3.4.5 Ajouter un exposant

diff-order `diff-order=`⟨liste-csv⟩

Cette clé place un exposant à chaque différentielle.

```
  \begin{equation}
    S[\phi] = \integral[diff-order=4]{\mathcal L (\phi, \partial_\mu \phi, x^
     \mu)}
  \end{equation}
```

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

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 personnalisés des différentielles

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

```
\begin{gather}
  \integral{\frac{\differentials}{x}}\\[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}
```

$$\int \frac{\mathrm{d}x}{x} \tag{18}$$

$$\langle 0 \mid T\{\phi(x)\phi(y)\} \mid 0\rangle = \int \frac{\mathrm{d}^4p}{(2\pi)^4} \frac{e^{-ip\cdot(x-y)}}{p^2 - m^2 + i\epsilon} \tag{19}$$

4.2 Retour sur la clé *limits*

`\NewLimitsKeyword` `\RenewLimitsKeyword` `\ProvideLimitsKeyword` `\DeclareLimitsKeyword`

`\NewLimitsKeyword {⟨mot-clé⟩} {⟨bornes⟩}`

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 dans 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 :

`ab`	a, b
`unit`	$0, 1$
`real`	$-\infty, +\infty$
`positive`	$0, +\infty$
`negative`	$-\infty, 0$
`circle`	$0, 2\pi$
`sircle`	$0, \pi$
`qcircle`	$0, \pi/2$
`height`	$0, H$
`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 l'intégrale de la façon suivante :

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

$$\int_0^H \int_0^{2\pi} \int_0^R \rho \,\mathrm{d}\rho \,\mathrm{d}\theta \,\mathrm{d}z \tag{20}$$

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}
```

$$\int_0^H \int_0^W \int_0^L \kappa(T)\nabla T \cdot \mathbf{n} \,\mathrm{d}x \,\mathrm{d}y \,\mathrm{d}z \tag{21}$$

4.3 Retour sur la clé *variables*

\NewVariableKeyword
\RenewVariableKeyword
\ProvideVariableKeyword
\DeclareVariableKeyword

\NewVariableKeyword {⟨mot-clé⟩} {⟨variables⟩} [⟨jacobien⟩]

mise à jour: v4.0.0

De façon similaire, les mêmes groupes de variables réapparaissent souvent. On peut donc également définir des mots-clés contenant un ensemble de variables d'intégration préenregistrés. 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, θ (r)
cylindrical	r, θ, z (r)
cylindrical*	ρ, θ, z (ρ)
spherical	r, θ, ϕ $(r^2, \sin\theta)$

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

$$\iiint_V f(r,\theta,\phi)\,\mathrm{d}r\,\mathrm{d}\theta\,\mathrm{d}\phi \tag{22}$$

4.4 Retour sur la clé *symbol*

\NewSymbolKeyword
\RenewSymbolKeyword
\ProvideSymbolKeyword
\DeclareSymbolKeyword

\NewSymbolKeyword {⟨clé⟩} {⟨symbole⟩}

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 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 *clé-symbole* définie par le package :

`single`	symbole utilisé = `\int`
`double`	symbole utilisé = `\iint`
`triple`	symbole utilisé = `\iiint`
`quadruple`	symbole utilisé = `\iiiint`
`contour`	symbole utilisé = `\oint`
`surface`	symbole utilisé = `\oiint`
`volume`	symbole utilisé = `\oiiint`

> ***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{equation}
  \integral[contour=\mathcal{C}, diff-vec]{f(\vec r)}
\end{equation}
```

$$\oint_{\mathcal{C}} f(\vec{r}) \cdot \mathrm{d}\vec{r} \tag{23}$$

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}
```

$$\int_{1}^{2} x \int_{4}^{5} y \,\mathrm{d}y \,\mathrm{d}x \tag{24}$$

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}
```

$$\int_{0}^{R} r \,\mathrm{d}r \int_{0}^{2\pi} \theta \,\mathrm{d}\theta \int_{0}^{\pi} \phi \,\mathrm{d}\phi \tag{25}$$

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}
```

$$\int_{0}^{R} r^2 \,\mathrm{d}r \int_{0}^{2\pi} \sin\theta \,\mathrm{d}\theta \int_{0}^{\pi} \mathrm{d}\phi \tag{26}$$

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}
```

$$\int_{1}^{2}\int_{3}^{4} xy \,\mathrm{d}x \,\mathrm{d}y \tag{27}$$

Vous pouvez aussi toujours profiter du placement automatique du « $\mathrm{d}x$ » avec cette syntaxe en faisant abstraction de `variables`.

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

$$\int_{1}^{10} x \,\mathrm{d}x \tag{28}$$

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

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

$$\int f(z) \,\mathrm{d}z \tag{29}$$

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 détails[2] de l'implémentation, il est porté à votre intention que le package tente d'extraire la première clé de l'argument optionnel[3] et vérifie si elle existe. C'est pourquoi, si vous créez une clé avec `\NewSymbolKeyword` dont le nom pourra probablement servir de bornes d'intégrations, *`intexrgal`* 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

nouveau: v3.0.0
mise à jour: v4.0.0

`\IntegralSetup {⟨liste de paramètres⟩}`

Cette macro permet, sous la syntaxe ⟨clé=valeur⟩, 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
   'implémentation
  defaultvar   = {x},
  defaultvar*  = {r},
  varsep       = \thinmuskip,
  diffsep      = \thinmuskip,
  innersymbsep = {-5mu},
  postsymbsep  = {-1mu},
  vectorstyle  = \vec,
  domainstyle  = \mathbb,
  novar    = false
}
```

diff-symb

nouveau: v4.0.0

`diff-symb=⟨séquence de contrôle⟩`

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

defaultvar
defaultvar*

nouveau: v3.0.0

`defaultvar= {⟨variables⟩}`

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

nouveau: v4.0.0

`varsep=⟨muexpr⟩`

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

diffsep

nouveau: v4.0.0

`diffsep=⟨muexpr⟩`

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

innersymbsep

nouveau: v4.0.0

`innersymbsep=⟨muexpr⟩`

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

postsymbsep

nouveau: v4.0.0

`postsymbsep=⟨muexpr⟩`

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

vectorstyle

nouveau: v3.0.0

`vectorstyle=⟨séquence de contrôle⟩`

Cette clé modifie la macro à appliquer aux variables quand `diff-vec` est en action. Il est donc possible d'altérer le style du vecteur : gras, souligné, ou même un autre style d'un package particulier, *esvect* par exemple.

`domainstyle` `domainstyle=⟨séquence de contrôle⟩`

nouveau: v3.0.0 Semblablement, on peut changer la macro à appliquer pour les clés `domain(*)`.

`novar` `novar=⟨true | false⟩`

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 `variables=none`.

Historique des modifications

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`, `\defaultdiff`, `\defaultvdiff` et `\vdiffstyle` en faveur de `\IntegralSetup`.
- Clés controlant symbole et borne, maintenant gérées à un plus haut niveau avec `\NewSymbolKeyword`.
- Clé `int-split`, en faveur de `mode`.
- Macro `\NewInegralSymbol`.

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.

v2.0.1 (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 de 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 controlant symbole et borne (clés `single`, `contour` etc).

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

PARTIE II

IMPLÉMENTATION

1 Initialisation

```
⟨*package⟩
⟨@@=intexgral⟩
```

1.1 Déclaration du package

```
\NeedsTeXFormat{LaTeX2e}
\RequirePackage{expl3}[2025-05-14]

\def\intexgral@module{intexgral}
\def\intexgral@version{v4.0.0}
\def\intexgral@date{2026-05-26}
\def\intexgral@description{A LaTeX package for typesetting integrals}

\ProvidesExplPackage
  \intexgral@module
  \intexgral@date
  \intexgral@version
  \intexgral@description
```

1.2 Vérifications préliminaires

On vérifie que la version d'expl3 soit suffisamment récente avant de poursuivre. Le package requiert la macro `\regex_if_match:nnTF`, précédemment nommée `\regex_match:nnTF` dans les versions antérieures à mai 2025.

```
\msg_new:nnn { intexgral } { expl3-too-old }
  {
    Your~expl3~installation~is~too~old,~
    "intexgral"~requires~expl3~dated~2025-05-14~or~later.\iow_newline:
    Package~loading~has~been~aborted.\iow_newline:
    \msg_see_documentation_text:n { intexgral }
  }

\@ifpackagelater{expl3}{2025-05-14}{}
  { \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).

```
\hook_gput_code:nnn { begindocument/before } { . }
  { \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.

```
\bool_new:N \l__intexgral_invert_limits_bool
```

`\l_@@_deactivate_variables_bool`

Détermine si les différentielles doivent être affichées ou non.

```
\bool_new:N \l__intexgral_deactivate_variables_bool
```

`\l_@@_invert_differential_bool`

Détermine l'ordre d'affichage des différentielles.

```
\bool_new:N \l__intexgral_invert_differential_bool
```

`\l_@@_default_differential_symbol_tl`

Contient le symbole par défaut pour les différentielles.

```
\tl_new:N \l__intexgral_default_differential_symbol_tl
```

`\l_@@_limits_style_tl`

Contient la primitive TEX à utiliser pour le style des bornes.

```
\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.

```
\keys_define:nn { intexgral }
  {
    invert-limits .bool_set:N = \l__intexgral_invert_limits_bool,
    invert-limits .usage:n    = { preamble },
    invert-limits .initial:n  = { false },

    invert-diff .bool_set:N = \l__intexgral_invert_differential_bool,
    invert-diff .usage:n    = { preamble },
    invert-diff .initial:n  = { false },

    limits-mode .choice:,
    limits-mode / limits .code:n =
      { \tl_set_eq:NN \l__intexgral_limits_style_tl \tex_limits:D },
    limits-mode / nolimits .code:n =
      { \tl_set_eq:NN \l__intexgral_limits_style_tl \tex_nolimits:D },

    limits-mode .usage:n   = { preamble },
    limits-mode .initial:n = { nolimits },

    italic .choice:,
      italic / true .code:n  =
        {
          \tl_set:Nn \l__intexgral_default_differential_symbol_tl
```

```
            { \mathnormal{d} }
        },
      italic / false .code:n =
        {
          \tl_set:Nn \l__intexgral_default_differential_symbol_tl
            { \mathrm{d} }
        },
    italic .usage:n   = { preamble },
    italic .initial:n = { false },

    upright .choice:,
      upright / true .code:n  =
        {
          \tl_set:Nn \l__intexgral_default_differential_symbol_tl
            { \mathrm{d} }
        },
      upright / false .code:n =
        {
          \tl_set:Nn \l__intexgral_default_differential_symbol_tl
            { \mathnormal{d} }
        },
    upright .usage:n   = { preamble },
    upright .initial:n = { true }
  }

\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 numéroter les intégrales composées dans le document.

```
\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.

```
\cs_new:Nn \__intexgral_warning_msg_header: {
  (
    integral~no.~
    \int_use:N\g__intexgral_integral_number_int\c_space_tl
    \msg_line_context:
  )
  \iow_newline:
}
```

Tous les messages sont maintenant définis.

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

```

```
\msg_new:nnnn { intexgral } { symb-key-not-def }
  { Symbol~key~"\tl_trim_spaces:n{#1}"~is~not~defined. }
  { Use~\token_to_str:N \NewSymbolKeyword\ instead. }

\msg_new:nnnn { intexgral } { diff-group-alr-def }
  { Differential~group~"\tl_trim_spaces:n{#1}"~is~already~defined. }
  { Use~\token_to_str:N \RenewVarKeyword\ instead. }

\msg_new:nnnn { intexgral } { diff-group-not-def }
  { Differential~group~"\tl_trim_spaces:n{#1}"~is~not~defined. }
  { Use~\token_to_str:N \NewVarKeyword\ instead. }

\msg_new:nnnn { intexgral } { lim-group-alr-def }
  { Limits~group~"\tl_trim_spaces:n{#1}"~is~already~defined. }
  { Use~\token_to_str:N \RenewLimitsKeyword\ instead. }

\msg_new:nnnn { intexgral } { lim-group-not-def }
  { Limits~group~"\tl_trim_spaces:n{#1}"~is~not~defined. }
  { Use~\NewLimitsKeyword\ instead. }

\msg_new:nnn { intexgral } { key-exists-for-lim }
  {
    Symbol~key~already~defined~with~
    \token_to_str:N \NewLimitsKeyword\ \__intexgral_warning_msg_header:
    Key~"#1"~already~exists~for~predefined~limits.~
    This~can~disrupt~the~functioning~of~the~special~syntax.
  }

\msg_new:nnn { intexgral } { unknown-symb }
  {
    Unknown~integral~symbol~\__intexgral_warning_msg_header:
    The~symbol~\tl_trim_spaces:n{#1}~has~been~replaced~with~
    \token_to_str:N\int.
  }

\msg_new:nnn { intexgral } { no-jacobian }
  {
    Jacobian~unavailable~\__intexgral_warning_msg_header:
    A~Jacobian~was~requested~for~the~"#1"~keyword,
    ~but~none~was~declared~for~the~latter.
  }

\msg_new:nnn { intexgral } { use-glyph-instead }
  {
    Consider~using~the~dedicated~glyph~\__intexgral_warning_msg_header:
    The~key~"nint"~was~used~with~fewer~than~
    5~regular~integral~signs.
  }
```

4 Variantes utiles

\regex_if_match:nV*TF*
\str_if_eq:ne*TF*
\str_if_eq:neF
\keys_if_exist:nV*TF*
\seq_use:Ne
\str_if_eq_p:en
\prop_gput:Nne
\seq_gset_split:cnn
\seq_set_split:Nnf

```
\cs_generate_variant:Nn \regex_if_match:nnTF { nVTF }
\cs_generate_variant:Nn \str_if_eq:nnTF      { neTF }
\cs_generate_variant:Nn \str_if_eq:nnF       { neF  }
\cs_generate_variant:Nn \keys_if_exist:nnTF  { nVTF }
\cs_generate_variant:Nn \seq_use:Nn          { Ne   }
\cs_generate_variant:Nn \str_if_eq_p:nn      { en   }
\cs_generate_variant:Nn \prop_gput:Nnn       { Nne  }
\cs_generate_variant:Nn \seq_gset_split:Nnn  { cnn  }
\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 ulterieurement en une séquence.

```
\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`.

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

\l_@@_lower_limit_tl
\l_@@_upper_limit_tl

Tokens contenant les bornes d'intégration.

```
\tl_new:N \l__intexgral_lower_limit_tl
\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`.

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

\l_@@_default_differential_tl
\l_@@_default_vector_differential_tl

Tokens contenant les différentielles par défaut, utilisés lorsque la clé `variables` n'est pas renseignée.

```
\tl_new:N \l__intexgral_default_differential_tl
\tl_new:N \l__intexgral_default_vector_differential_tl
```

\l_@@_vectorial_differential_style_tl
\l_@@_domain_style_tl

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

```
\tl_new:N \l__intexgral_vectorial_differential_style_tl
\tl_new:N \l__intexgral_domain_style_tl
```

\l_@@_domain_char_tl
\l_@@_domain_dimension_tl

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

```
\tl_new:N \l__intexgral_domain_char_tl
\tl_new:N \l__intexgral_domain_dimension_tl
```

\c_@@_left_bracket_tl
\c_@@_right_bracket_tl

Tokens constants pour la clé `limits*`.

```
\tl_const:Nn \c__intexgral_left_bracket_tl { [ }
\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.

```
\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`.

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

\l_@@_custom_variables_bool

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

```
\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.

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

`\l_@@_vectorial_differential_bool`

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

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

`\l_@@_jacobian_bool`

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

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

5.3 Séquences et clist

`\l_@@_domain_seq`

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

```
\seq_new:N \l__intexgral_domain_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).

```
\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 crée pour stocker d'un côté les mots clés propres aux bornes, et de l'autre, ceux pour les différentielles.

```
\prop_new:N \g__intexgral_limits_keyword_prop
\prop_new:N \g__intexgral_differential_group_keyword_prop
```

5.5 Musksip

\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.

```
\muskip_new:N \l__intexgral_symbol_inner_sep_muskip
\muskip_new:N \l__intexgral_symbol_post_sep_muskip
\muskip_new:N \l__intexgral_variable_sep_muskip
\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:nn`*TF*

```
\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⟩

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

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

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

\@@_invert_limits:w `\@@_invert_limits:w` ⟨borne inférieure⟩,⟨borne supérieure⟩`\q_stop`

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

```
\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.

```
\cs_new:Npn \__intexgral_general_integral_symbol:
  {
    \l__intexgral_integral_symbol_tl
    \l__intexgral_limits_style_tl
    \c_math_subscript_token
      { \l__intexgral_lower_limit_tl }
    \c_math_superscript_token
      { \l__intexgral_upper_limit_tl }
  }
```

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).

```
\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`).

```
\seq_new:N \l__intexgral_integrand_seq
```

`\l_@@_jacobian_seq`

Une pour le jacobien.

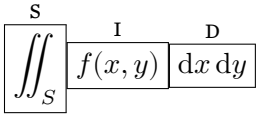
```
\seq_new:N \l__intexgral_jacobian_seq
```

`\l_@@_differential_seq`

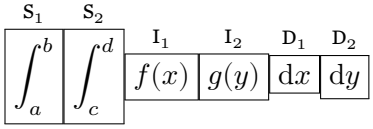
Et enfin, une pour les différentielles.

```
\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.



Pour une intégrale définie.



6.2 Séquences subsidiaires

Quelques séquences suplémentaires 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.

```
\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.

```
\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`.

```
\seq_new:N \l__intexgral_upper_limits_seq
\seq_new:N \l__intexgral_lower_limits_seq
```

7 Bornes d'intégration

`\@@_parse_limits:n` `\@@_parse_limits:n {⟨borne inférieure⟩}{⟨ borne supérieure⟩}, ⟨mot-clé⟩`

Cette macro vérifie si son argument correspond à un mot-clé défini pour les bornes, auquel cas elle se développe en son contenu en l'inversant si besoin (selon le statut de `\l_@@_invert_limits_bool`). Sinon, la macro se développe simplement en son argument.

```
\cs_new:Npn \__intexgral_parse_limits:n #1
  {
    \prop_if_in:NnTF \g__intexgral_limits_keyword_prop { #1 }
      {
        \bool_if:NTF \l__intexgral_invert_limits_bool
          {
            \exp_last_unbraced:Ne \__intexgral_invert_limits:w
              { \prop_item:Nn \g__intexgral_limits_keyword_prop { #1 } }
              \q_stop
          }
          { \prop_item:Nn \g__intexgral_limits_keyword_prop { #1 } }
      }
      { #1 }
  }
```

\@@_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 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 executée 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.

```
\cs_new_protected:Nn \__intexgral_set_limits_regular:
  {
    \seq_map_inline:Nn \l__intexgral_limits_seq
      {
        \seq_set_split:Nnf \l_tmpa_seq { , }
          { \__intexgral_parse_limits:n { ##1 } }
        \seq_put_right:Ne \l__intexgral_lower_limits_seq
          { \seq_item:Nn \l_tmpa_seq { 1 } }
        \seq_put_right:Ne \l__intexgral_upper_limits_seq
          { \seq_item:Nn \l_tmpa_seq { 2 } }
      }
  }
```

\@@_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.

```
\cs_new_protected:Npn \__intexgral_set_limits_starred:
  {
    \seq_map_indexed_inline:Nn \l__intexgral_limits_seq
      {
        \seq_set_split:Nnf \l_tmpa_seq { , }
          { \__intexgral_parse_limits:n { ##2 } }
        \bool_if:NTF \l__intexgral_invert_limits_bool
          {
            \tl_set:Ne \l__intexgral_lower_limit_tl
              {
                \seq_item:Nn \l_tmpa_seq { 2 }
                \tex_mathpunct:D,
                \seq_item:Nn \l_tmpa_seq { 1 }
              }
          }
          {
            \tl_set:Ne \l__intexgral_lower_limit_tl
              {
                \seq_item:Nn \l_tmpa_seq { 1 }
                \tex_mathpunct:D,
                \seq_item:Nn \l_tmpa_seq { 2 }
              }
          }
            \bool_if:NTF \l__intexgral_invert_limits_bool
              { \seq_put_right:Ne \l__intexgral_upper_limits_seq }
              { \seq_put_right:Ne \l__intexgral_lower_limits_seq }
          {
            \str_case_e:nnF { \seq_item:Nn \l_tmpa_seq { 1 } }
              {
```

```
                { -\infty } { \tex_left:D \c__intexgral_right_bracket_tl }
            }
            { \tex_left:D \c__intexgral_left_bracket_tl }
          \tl_use:N \l__intexgral_lower_limit_tl
          \str_case_e:nnF { \seq_item:Nn \l_tmpa_seq { 2 } }
            {
              { +\infty } { \tex_right:D \c__intexgral_left_bracket_tl }
            }
            { \tex_right:D \c__intexgral_right_bracket_tl }
        }
    }
}
```

`\@@_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 précédement.

```
\cs_new_protected:Npn \__intexgral_set_limits:nn #1#2
  {
    \bool_if:NTF \l__intexgral_invert_limits_bool
      {
        \tl_set:Ne \l__intexgral_lower_limit_tl { \exp_not:n { #2 } }
        \tl_set:Ne \l__intexgral_upper_limit_tl { \exp_not:n { #1 } }
      }
      {
        \tl_set:Ne \l__intexgral_lower_limit_tl { \exp_not:n { #1 } }
        \tl_set:Ne \l__intexgral_upper_limit_tl { \exp_not:n { #2 } }
      }
  }
```

`\@@_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 symbole 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.

```
\cs_new_protected:Nn \__intexgral_generate_integral_sequence:
  {
    \int_set:Nn \l_tmpa_int
      {
```

```
        \seq_if_empty:NTF \l__intexgral_lower_limits_seq
          { \seq_count:N \l__intexgral_upper_limits_seq }
          { \seq_count:N \l__intexgral_lower_limits_seq }
      }
    \int_step_inline:nn { \l_tmpa_int }
      {
        \__intexgral_set_limits:nn
          { \seq_item:Nn \l__intexgral_lower_limits_seq { ##1 } }
          { \seq_item:Nn \l__intexgral_upper_limits_seq { ##1 } }
        \seq_put_right:Ne \l__intexgral_integral_symbol_seq
          { \__intexgral_general_integral_symbol: }
      }
  }
```

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 «$\mathrm{d}x, \mathrm{d}y, \mathrm{d}z$»); tout en prennant 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`.

```
\cs_new_protected:Nn \__intexgral_parse_variables:
  {
    \bool_if:NTF \l__intexgral_custom_variables_bool
      {
        \seq_map_indexed_inline:Nn \l__intexgral_variables_seq
          {
            \seq_put_right:Ne \l__intexgral_differential_seq
              {
                \exp_not:V \l__intexgral_differential_symbol_tl
                \clist_if_empty:NF \l__intexgral_differential_order_clist
                  {
                    \c_math_superscript_token
                      {
                        \clist_item:Nn
                        \l__intexgral_differential_order_clist
                        { ##1 }
                      }
                  }
                \bool_if:NT \l__intexgral_vectorial_differential_bool
                  { \exp_not:V \l__intexgral_vectorial_differential_style_tl }
                  { ##2 }
              }
          }
      }
      {
        \seq_put_right:Ne \l__intexgral_differential_seq
          {
            \exp_not:V \l__intexgral_differential_symbol_tl
            \clist_if_empty:NF \l__intexgral_differential_order_clist
              {
```

```
                % TODO: permettre de soumettre une séquence en variables par défaut
                \c_math_superscript_token
                  {
                    \clist_item:Nn
                    \l__intexgral_differential_order_clist
                    { 1 }
                  }
              }
            \bool_if:NTF \l__intexgral_vectorial_differential_bool
              {
                \exp_not:V \l__intexgral_vectorial_differential_style_tl
                  {\l__intexgral_default_vector_differential_tl}
              }
              { \l__intexgral_default_differential_tl }
          }
      }
  }
```

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.

```
\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 reçut 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`.

```
\cs_new_protected:Npn \__intexgral_extract_first_key_name:n #1
  {
    \seq_set_split:Nnn \l_tmpa_seq { : } { #1 }
    \tl_set:Ne \l_tmpa_tl { \seq_item:Nn \l_tmpa_seq { 1 } }
    \tl_if_in:NnTF \l_tmpa_tl { = }
      {
        \tl_set:Ne \l__intexgral_key_name_tl
          {
            \exp_last_unbraced:NV
            \__intexgral_retrieve_key_name:w \l_tmpa_tl \q_stop
          }
      }
      { \tl_set_eq:NN \l__intexgral_key_name_tl \l_tmpa_tl }
  }
```

`\@@_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 optionel comme une séquence à délimiter avec le deux-point. Des séquences temporaires sont employées pour 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.

```
\cs_new_protected:Npn \__intexgral_parse_integral_keys:n #1
  {
    \keys_if_exist:nVTF { integral } \l__intexgral_key_name_tl
      { \keys_set:nn { integral } { #1 } }
      {
        \regex_split:nnN { : } { #1 } \l_tmpa_seq
        \str_if_eq:eeTF
          { \seq_item:Nn \l_tmpa_seq { 2 } }
          { +j }
          {
            \exp_args:NNne \seq_set_item:Nnn \l_tmpa_seq { 2 }
              { \l__intexgral_default_differential_tl+j }
          }
          {
            \int_compare:nNnT { \seq_count:N \l_tmpa_seq } < { 2 }
              {
                \seq_put_right:NV
                  \l_tmpa_seq
                  \l__intexgral_default_differential_tl
              }
          }
        \int_compare:nNnT { \seq_count:N \l_tmpa_seq } < { 3 }
          { \seq_put_right:Nn \l_tmpa_seq { d } }
        \seq_set_split:Nne \l_tmpb_seq
          { + }
          { \seq_item:Nn \l_tmpa_seq { 2 } }
        \keys_set:ne { integral }
          {
            limits    = { \seq_item:Nn \l_tmpa_seq { 1 } },
            variables = { \seq_item:Nn \l_tmpb_seq { 1 } },
            mode      =
              {
                \str_case:enF { \seq_item:Nn \l_tmpa_seq { 3 } }
                  {
                    { d       } { default }
                    { n       } { nested  }
                    { p       } { product }
                    { default } { default }
                    { nested  } { nested  }
                    { product } { product }
                  }
                { default }
              },
            \bool_if:nT
              {
                \int_compare_p:nNn { \seq_count:N \l_tmpb_seq } > { 1 }
```

```
                &&
                \str_if_eq_p:en { \seq_item:Nn \l_tmpb_seq { 2 } } { j }
              }
              { jacobian }
          }
      }
  }
```

10 Composition de l'intégrale

\@@_integral_preconfiguration:

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

— On regarde si l'intégrande doit être séparé 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, seul l'utilisation de cette clé engendre l'execution de toutes les macros présentées en section 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.

```
\cs_new_protected:Nn \__intexgral_integral_preconfiguration:
  {
    \bool_if:NTF \l__intexgral_separate_integral_bool
      {
        \seq_set_split:NnV
          \l__intexgral_integrand_seq
          { ; }
          \l__intexgral_integrand_tl
      }
      {
        \seq_put_right:NV
          \l__intexgral_integrand_seq
          \l__intexgral_integrand_tl
      }

    \seq_if_empty:NT \l__intexgral_integral_symbol_seq
      {
        \seq_put_right:Nn \l__intexgral_integral_symbol_seq
          { \__intexgral_general_integral_symbol: }
      }

    \bool_if:NF \l__intexgral_deactivate_variables_bool
      { \__intexgral_parse_variables: }

```

```
    \regex_if_match:nVTF { \c{differentials} } \l__intexgral_integrand_tl
      { \bool_set_true:N \l__intexgral_manual_differential_bool }
      { \bool_set_false:N \l__intexgral_manual_differential_bool }
  }
```

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 précédant (pour des raisons d'espacement propres à `\cdot`).

```
\cs_new_protected:Npn \__intexgral_print_default_integral:
  {
    \bool_if:NT \l__intexgral_manual_differential_bool
      {
        \cs_set_protected:Npn \differentials
          {
            \seq_use:Nn \l__intexgral_differential_seq
              { \__intexgral_mkern:N \l__intexgral_differential_sep_muskip }
          }
      }
    \seq_use:Nn \l__intexgral_integral_symbol_seq
      { \__intexgral_mkern:N \l__intexgral_symbol_inner_sep_muskip }
    \__intexgral_mkern:N \l__intexgral_symbol_post_sep_muskip
    \seq_use:Nn \l__intexgral_integrand_seq {  }
    \bool_if:NT \l__intexgral_jacobian_bool
      { \seq_use:Nn \l__intexgral_jacobian_seq {  } }
    \__intexgral_mkern:N \l__intexgral_variable_sep_muskip
    \bool_if:NT \l__intexgral_vectorial_differential_bool
      { \tex_unkern:D \cdot}
    \bool_if:NF \l__intexgral_manual_differential_bool
      {
        \seq_use:Nn \l__intexgral_differential_seq
          { \__intexgral_mkern:N \l__intexgral_differential_sep_muskip }
      }
  }
```

`\@@_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.

```
\cs_new_protected:Npn \__intexgral_print_default_integral_inv:
  {
    \bool_if:NT \l__intexgral_manual_differential_bool
      {
```

```
        \cs_set_protected:Npn \differentials
          {
            \seq_use:Nn \l__intexgral_differential_seq
              { \__intexgral_mkern:N \l__intexgral_differential_sep_muskip }
          }
      }
    \seq_use:Nn \l__intexgral_integral_symbol_seq
      { \__intexgral_mkern:N \l__intexgral_symbol_inner_sep_muskip }
    \__intexgral_mkern:N \l__intexgral_symbol_post_sep_muskip
    \bool_if:NF \l__intexgral_manual_differential_bool
      {
        \seq_use:Nn \l__intexgral_differential_seq
          { \__intexgral_mkern:N \l__intexgral_differential_sep_muskip }
      }
    \__intexgral_mkern:N \l__intexgral_variable_sep_muskip
    \bool_if:NT \l__intexgral_vectorial_differential_bool
      { \tex_unkern:D \cdot}
    \seq_use:Nn \l__intexgral_integrand_seq {  }
    \bool_if:NT \l__intexgral_jacobian_bool
      { \seq_use:Nn \l__intexgral_jacobian_seq {  } }
  }
```

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`.

```
\cs_new_protected:Npn \__intexgral_print_nested_integral:
  {
    \bool_if:NT \l__intexgral_manual_differential_bool
      {
        \cs_set_protected:Npn \differentials
          {
            \seq_use:Nn \l__intexgral_differential_seq
              { \__intexgral_mkern:N \l__intexgral_differential_sep_muskip }
          }
      }
    \int_step_inline:nn { \seq_count:N \l__intexgral_integral_symbol_seq }
      {
        \seq_item:Nn \l__intexgral_integral_symbol_seq { ##1 }
        \__intexgral_mkern:N \l__intexgral_symbol_post_sep_muskip
        \seq_item:Nn \l__intexgral_integrand_seq { ##1 }
        \bool_if:NT \l__intexgral_jacobian_bool
          { \seq_item:Nn \l__intexgral_jacobian_seq { ##1 } }
      }
    \__intexgral_mkern:N \l__intexgral_variable_sep_muskip
    \bool_if:NF \l__intexgral_manual_differential_bool
      {
        \seq_use:Nn \l__intexgral_differential_seq
          { \__intexgral_mkern:N \l__intexgral_differential_sep_muskip }
      }
  }
```

\@@_print_nested_integral_inv:

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

```
\cs_new_protected:Npn \__intexgral_print_nested_integral_inv:
  {
    \int_step_inline:nn { \seq_count:N \l__intexgral_integral_symbol_seq }
      {
        \seq_item:Nn \l__intexgral_integral_symbol_seq { ##1 }
        \__intexgral_mkern:N \l__intexgral_symbol_post_sep_muskip
        \seq_item:Nn \l__intexgral_differential_seq { ##1 }
      }
    \__intexgral_mkern:N \l__intexgral_variable_sep_muskip
    \seq_use:Nn \l__intexgral_integrand_seq {  }
    \bool_if:NT \l__intexgral_jacobian_bool
      { \seq_use:Nn \l__intexgral_jacobian_seq {  } }
  }
```

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 aparaissent dans la boucle.

```
\cs_new_protected:Npn \__intexgral_print_product_integral:
  {
    \bool_if:NT \l__intexgral_manual_differential_bool
      {
        \cs_set_protected:Npn \differentials
          {
            \seq_pop_left:NN \l__intexgral_differential_seq \l_tmpa_tl
            \tl_use:N \l_tmpa_tl
          }
      }
    \int_step_inline:nn { \seq_count:N \l__intexgral_integral_symbol_seq }
      {
        \seq_item:Nn \l__intexgral_integral_symbol_seq { ##1 }
        \__intexgral_mkern:N \l__intexgral_symbol_post_sep_muskip
        \seq_item:Nn \l__intexgral_integrand_seq { ##1 }
        \bool_if:NT \l__intexgral_jacobian_bool
          { \seq_item:Nn \l__intexgral_jacobian_seq { ##1 } }
        \__intexgral_mkern:N \l__intexgral_variable_sep_muskip
        \bool_if:NF \l__intexgral_manual_differential_bool
          { \seq_item:Nn \l__intexgral_differential_seq { ##1 } }
      }
  }
```

\@@_print_product_integral_inv:

De même, peu de changements.

```
\cs_new_protected:Npn \__intexgral_print_product_integral_inv:
  {
    \bool_if:NT \l__intexgral_manual_differential_bool
      {
        \cs_set_protected:Npn \differentials
```

```
          {
            \seq_pop_left:NN \l__intexgral_differential_seq \l_tmpa_tl
            \tl_use:N \l_tmpa_tl
          }
      }
    \int_step_inline:nn { \seq_count:N \l__intexgral_integral_symbol_seq }
      {
        \seq_item:Nn \l__intexgral_integral_symbol_seq { ##1 }

        \bool_if:NF \l__intexgral_manual_differential_bool
          {
            \__intexgral_mkern:N \l__intexgral_symbol_post_sep_muskip
            \seq_item:Nn \l__intexgral_differential_seq { ##1 }
          }
        \__intexgral_mkern:N \l__intexgral_variable_sep_muskip
        \seq_item:Nn \l__intexgral_integrand_seq { ##1 }
        \bool_if:NT \l__intexgral_jacobian_bool
          { \seq_item:Nn \l__intexgral_jacobian_seq { ##1 } }
      }
  }
```

10.4 Composition finale

\@@_print_integral:

La macro principale composant l'intégrale incrémente tout d'abord le compteur global permettant de numéroter 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.

```
\cs_new_protected:Nn \__intexgral_print_integral:
  {
    \int_gincr:N \g__intexgral_integral_number_int
    \__intexgral_integral_preconfiguration:
    \bool_if:NTF \l__intexgral_invert_differential_bool
      {
        \str_case:VnF \l__intexgral_display_mode_str
          {
            { default } { \__intexgral_print_default_integral_inv: }
            { nested  } { \__intexgral_print_nested_integral_inv:  }
            { product } { \__intexgral_print_product_integral_inv: }
          }
          { \__intexgral_print_default_integral_inv: }
      }
      {
        \str_case:VnF \l__intexgral_display_mode_str
          {
            { default } { \__intexgral_print_default_integral: }
            { nested  } { \__intexgral_print_nested_integral:  }
            { product } { \__intexgral_print_product_integral: }
          }
          { \__intexgral_print_default_integral: }
      }
  }
```

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.

```
\keys_define:nn { integral }
  {
    mode .choice:,
      mode / default .code:n =
        {
          \bool_set_false:N \l__intexgral_separate_integral_bool
          \str_set:Nn \l__intexgral_display_mode_str { default }
        },
      mode / nested .code:n =
        {
          \bool_set_true:N \l__intexgral_separate_integral_bool
          \str_set:Nn \l__intexgral_display_mode_str { nested }
        },
      mode / product .code:n =
        {
          \bool_set_true:N \l__intexgral_separate_integral_bool
          \str_set:Nn \l__intexgral_display_mode_str { product }
        },
      mode .default:n = { default },

    limits .code:n =
      {
        \seq_set_split:Nnn \l__intexgral_limits_seq { ; } { #1 }
        \__intexgral_set_limits_regular:
        \__intexgral_generate_integral_sequence:
      },

    limits* .code:n =
      {
        \seq_set_split:Nnn \l__intexgral_limits_seq { ; } { #1 }
        \__intexgral_set_limits_starred:
        \__intexgral_generate_integral_sequence:
      },

    llimit .tl_set:N = \l__intexgral_lower_limit_tl,

    ulimit .tl_set:N = \l__intexgral_upper_limit_tl,

    symbol .code:n =
      {
        \cs_if_exist:NTF #1
          { \tl_set_eq:NN \l__intexgral_integral_symbol_tl #1 }
          {
            \msg_warning:nnn { intexgral } { unknown-symb } { #1 }
            \tl_set_eq:NN \l__intexgral_integral_symbol_tl \int
          }
      },

    nint .code:n =
      {
        \int_compare:nNnT { #1 } < { 5 }
          { \msg_warning:nn { intexgral } { use-glyph-instead } }
```

```
652
653         \tl_clear:N \l__intexgral_integral_symbol_tl
654
655         \int_step_inline:nn { #1 }
656           {
657             \tl_put_right:Nn \l__intexgral_integral_symbol_tl
658               { \int }
659             \int_compare:nNnT { ##1 } < { #1 }
660               {
661                 \tl_put_right:Nn \l__intexgral_integral_symbol_tl
662                   {
663                     \__intexgral_mathchoice:nnnn
664                       { \tex_mkern:D -12mu \scan_stop: }
665                       { \tex_mkern:D -8mu \scan_stop: }
666                       { \tex_mkern:D -4mu \scan_stop: }
667                       { \tex_mkern:D -2mu \scan_stop: }
668                   }
669               }
670           }
671       },
672
673     domain .code:n =
674       {
675         \tl_set:Nn \l_tmpa_tl { #1 }
676         \seq_set_split:NnV \l__intexgral_domain_seq { * } \l_tmpa_tl
677         \seq_map_inline:Nn \l__intexgral_domain_seq
678           {
679             \tl_if_empty:NF \l__intexgral_lower_limit_tl
680               { \tl_put_right:Nn \l__intexgral_lower_limit_tl { \times } }
681             \tl_set:Ne \l__intexgral_domain_char_tl
682               { \exp_args:Ne \str_uppercase:n { \tl_head:n { ##1 } } }
683             \tl_set:Ne \l__intexgral_domain_dimension_tl
684               { \tl_tail:n { ##1 } }
685             \tl_put_right:Ne \l__intexgral_lower_limit_tl
686               {
687                 \exp_not:V \l__intexgral_domain_style_tl
688                   { \l__intexgral_domain_char_tl }
689                 \c_math_superscript_token { \l__intexgral_domain_dimension_tl }
690               }
691           }
692       },
693
694     domain* .code:n =
695       {
696         \tl_set:Nn \l_tmpa_tl { #1 }
697         \seq_set_split:NnV \l__intexgral_domain_seq { * } \l_tmpa_tl
698         \seq_map_inline:Nn \l__intexgral_domain_seq
699           {
700             \tl_if_empty:NF \l__intexgral_lower_limit_tl
701               { \tl_put_right:Nn \l__intexgral_lower_limit_tl { \times } }
702             \tl_set:Ne \l__intexgral_domain_char_tl
703               { \exp_args:Ne \str_uppercase:n { \tl_head:n { ##1 } } }
704             \tl_set:Ne \l__intexgral_domain_dimension_tl
705               { \tl_tail:n { ##1 } }
706             \tl_put_right:Ne \l__intexgral_lower_limit_tl
707               {
708                 \exp_not:V \l__intexgral_domain_style_tl
709                   { \l__intexgral_domain_char_tl }
```

```
                \c_math_subscript_token { \l__intexgral_domain_dimension_tl }
              }
          }
      },

    boundary .code:n =
      { \tl_set:Nn \l__intexgral_lower_limit_tl { \partial #1 } },

    variables .code:n =
      {
        \bool_set_true:N \l__intexgral_custom_variables_bool
        \str_if_eq:nnTF { #1 } { none }
          { \bool_set_true:N \l__intexgral_deactivate_variables_bool }
          {
            \prop_get:NnNTF
              \g__intexgral_differential_group_keyword_prop { #1 } \l_tmpa_tl
                {
                  \seq_set_split:NnV
                  \l__intexgral_variables_seq
                  { , }
                  \l_tmpa_tl
                  \seq_if_exist:cTF { g__intexgral_#1_jacobian_seq }
                    {
                      \seq_set_eq:Nc
                        \l__intexgral_jacobian_seq
                        { g__intexgral_#1_jacobian_seq }
                    }
                    { \msg_warning:nnn { intexgral } { no-jacobian } { #1 } }
                }
                { \seq_set_split:Nnn \l__intexgral_variables_seq { , } { #1 } }
          }
      },

    jacobian .code:n =
      { \bool_set_true:N \l__intexgral_jacobian_bool },
    diff-vec .code:n =
      { \bool_set_true:N \l__intexgral_vectorial_differential_bool },
    diff-symb .tl_set:N = \l__intexgral_differential_symbol_tl,
    diff-order .clist_set:N = \l__intexgral_differential_order_clist,
  }

\keys_define:nn { IntegralSetup }
  {
    diff-symb    .tl_set:N     = \l__intexgral_differential_symbol_tl,
    defaultvar   .tl_set:N     = \l__intexgral_default_differential_tl,
    defaultvar*  .tl_set:N     = \l__intexgral_default_vector_differential_tl,
    postsymbsep  .muskip_set:N = \l__intexgral_symbol_post_sep_muskip,
    varsep       .muskip_set:N = \l__intexgral_variable_sep_muskip,
    diffsep      .muskip_set:N = \l__intexgral_differential_sep_muskip,
    vectorstyle  .tl_set:N     = \l__intexgral_vectorial_differential_style_tl,
    domainstyle  .tl_set:N     = \l__intexgral_domain_style_tl,
    innersymbsep .tl_set:N     = \l__intexgral_symbol_inner_sep_muskip,
    novar        .bool_set:N   = \l__intexgral_deactivate_variables_bool,
  }
```

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, quelque 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.

```
\cs_new_protected:Npn \__intexgral_new_limits_group:nn #1#2
  {
    \prop_gput:Nne \g__intexgral_limits_keyword_prop { #1 }
      {
        \bool_if:NTF \l__intexgral_invert_limits_bool
          { \__intexgral_invert_limits:w #2\q_stop }
          { #2 }
      }
  }
```

`\NewLimitsKeyword`

Cette macro est documentée en sous-section 4.2.

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

`\RenewLimitsKeyword`

Cette macro est documentée en sous-section 4.2.

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

`\ProvideLimitsKeyword`

Cette macro est documentée en sous-section 4.2.

```
\NewDocumentCommand\ProvideLimitsKeyword{ m m }
  {
    \prop_if_in:NnF \g__intexgral_limits_keyword_prop { #1 }
      { \__intexgral_new_limits_group:nn { #1 } { #2 } }
  }
```

\DeclareLimitsKeyword

Cette macro est documentée en sous-section 4.2.

```
\NewDocumentCommand\DeclareLimitsKeyword{ m m }
  {
    \prop_remove:Nn \g__intexgral_limits_keyword_prop { #1 }
    \__intexgral_new_limits_group:nn { #1 } { #2 }
  }
```

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.

```
\cs_new_protected:Npn \__intexgral_new_variables_group:nnn #1#2#3
  {
    \prop_gput:Nnn \g__intexgral_differential_group_keyword_prop
      { #1 } { #2 }
    \tl_if_blank:nF { #3 }
      {
        \seq_new:c { g__intexgral_#1_jacobian_seq }
        \seq_gset_split:cnn { g__intexgral_#1_jacobian_seq } { , } { #3 }
      }
  }
```

\NewVariableKeyword

Cette macro est documentée en sous-section 4.3.

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

\RenewVariableKeyword

Cette macro est documentée en sous-section 4.3.

```
\NewDocumentCommand\RenewVariableKeyword{ m m o }
  {
    \prop_pop:NnNTF \g__intexgral_differential_group_keyword_prop { #1 }
      \l_tmpa_tl
      {
        \seq_gclear:c { g__intexgral_#1_jacobian_seq }
        \__intexgral_new_variables_group:nnn { #1 } { #2 } { #3 }
      }
      { \msg_error:nnn { intexgral } { diff-group-not-def } { #1 } }
  }
```

\ProvideVariableKeyword

Cette macro est documentée en sous-section 4.3.

```
\NewDocumentCommand\ProvideVariableKeyword{ m m o }
  {
    \prop_if_in:NnF \g__intexgral_differential_group_keyword_prop { #1 }
      { \__intexgral_new_variables_group:nnn { #1 } { #2 } { #3 } }
  }
```

\DeclareVariableKeyword

Cette macro est documentée en sous-section 4.3.

```
\NewDocumentCommand\DeclareVariableKeyword{ m m o }
  {
    \prop_remove:Nn \g__intexgral_differential_group_keyword_prop { #1 }
    \seq_gclear:c { g__intexgral_#1_jacobian_seq }
    \__intexgral_new_variables_group:nnn { #1 } { #2 } { #3 }
  }
```

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.

```
\NewDocumentCommand\NewSymbolKeyword{ m m }
  {
    \prop_if_in:NnT \g__intexgral_limits_keyword_prop { #1 }
      { \msg_warning:nnn { intexgral } { key-exists-for-lim } { #1 } }
    \keys_if_exist:nnTF { integral } { #1 }
      { \msg_error:nnn { intexgral } { symb-key-alr-def } { #1 } }
      {
        \keys_define:nn { integral }
          {
            #1 .meta:n =
              {
                symbol=#2,
                llimit=##1
              },
          }
      }
  }
```

\RenewSymbolKeyword

Cette macro est documentée en sous-section 4.4.

```
\NewDocumentCommand\RenewSymbolKeyword{ m m }
  {
    \prop_if_in:NnT \g__intexgral_limits_keyword_prop { #1 }
      { \msg_warning:nnn { intexgral } { key-exists-for-lim } { #1 } }
    \keys_if_exist:nnTF { integral } { #1 }
      {
        \keys_define:nn { integral }
          {
            #1 .undefine:
            #1 .meta:n =
```

```
            {
              symbol=#2,
              llimit=##1
            }
        }
    }
    { \msg_error:nnn { intexgral } { symb-key-not-def } }
}
```

\ProvideSymbolKeyword

Cette macro est documentée en sous-section 4.4.

```
\NewDocumentCommand\ProvideSymbolKeyword{ m m }
  {
    \prop_if_in:NnT \g__intexgral_limits_keyword_prop { #1 }
      { \msg_warning:nnn { intexgral } { key-exists-for-lim } { #1 } }
    \keys_if_exist:nnF { integral } { #1 }
      {
        \keys_define:nn { integral }
          {
            #1 .meta:n =
              {
                symbol=#2,
                llimit=##1
              },
          }
      }
  }
```

\DeclareSymbolKeyword

Cette macro est documentée en sous-section 4.4.

```
\NewDocumentCommand\DeclareSymbolKeyword{ m m }
  {
    \keys_define:nn { integral }
      {
        #1 .undefine:
        #1 .meta:n =
          {
            symbol=#2,
            llimit=##1
          },
      }
  }
```

12.4 Macros de configuration

\IntegralSetup *Cette macro est documentée en section 6.*

```
\NewDocumentCommand\IntegralSetup{ m }
  { \keys_set:nn { IntegralSetup } { #1 } }
```

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

```
\NewDocumentCommand\intexgralsetup{ m }
  { \keys_set:nn { intexgral } { #1 } }
\@onlypreamble\intexgralsetup
```

\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 conséquences. Elle assigne ensuite l'argument principal au token correspondant, avant d'appeler la macro centrale qui compose l'intégrale. Le tout est executé dans un groupe afin que toutes les modifications demeurent locales.
Cette macro est documentée en section 2.

```
\NewDocumentCommand\integral{ O{} m }
  {
    \group_begin:
    \tl_if_empty:nF { #1 }
      {
        \__intexgral_extract_first_key_name:n { #1 }
        \__intexgral_parse_integral_keys:n { #1 }
      }
    \tl_set:Nn \l__intexgral_integrand_tl { #2 }
    \__intexgral_print_integral:
    \group_end:
  }
```

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

```
\NewSymbolKeyword{single}     {\int}
\NewSymbolKeyword{double}     {\iint}
\NewSymbolKeyword{triple}     {\iiint}
\NewSymbolKeyword{quadruple}  {\iiiint}
\NewSymbolKeyword{contour}    {\oint}
\NewSymbolKeyword{surface}    {\oiint}
\NewSymbolKeyword{volume}     {\oiiint}
```

13.2 Bornes

```
\NewLimitsKeyword{ab}         {a, b}
\NewLimitsKeyword{real}       {-\infty, +\infty}
\NewLimitsKeyword{positive}   {0, +\infty}
\NewLimitsKeyword{negative}   {-\infty, 0}
\NewLimitsKeyword{unit}       {0, 1}
\NewLimitsKeyword{circle}     {0, 2\pi}
\NewLimitsKeyword{scircle}    {0, \pi}
\NewLimitsKeyword{qcircle}    {0, \pi/2}
```

```
\NewLimitsKeyword{height}    {0, H}
\NewLimitsKeyword{radius}    {0, R}
\NewLimitsKeyword{length}    {0, L}
\NewLimitsKeyword{time}      {0, T}
```

13.3 Variables

```
\NewVariableKeyword{cartesian}    {x, y, z}
\NewVariableKeyword{planar}       {x, y}
\NewVariableKeyword{polar}        {r, \theta}        [r]
\NewVariableKeyword{cylindrical}  {r, \theta, z}     [r]
\NewVariableKeyword{cylindrical*} {\rho, \theta, z}   [\rho]
\NewVariableKeyword{spherical}    {r, \theta, \phi}  [r^2, \sin\theta]
```

13.4 Paramètres par défaut

```
\IntegralSetup{
  diff-symb    = \l__intexgral_default_differential_symbol_tl,
  defaultvar   = {x},
  defaultvar*  = {r},
  varsep       = \thinmuskip,
  diffsep      = \thinmuskip,
  innersymbsep = {-5mu},
  postsymbsep  = {-1mu},
  vectorstyle  = \vec,
  domainstyle  = \mathbb,
  novar        = false
}
```

⟨/package⟩

Index

K

M

T

V