Table des matières

1 Setup du package

Ce package se divise en deux fichiers scripts dont les templates sont fournis. Le premier permet de charger des résultats de simulation issus de fichiers **anydata.h5** et des constantes stockées dans un **AnyFileOut** (non obligatoire) et qui sauvegardera les variables chargées dans un fichier. Un second script qui chargera ces fichiers de résultats pour ensuite faire des graphiques.

Il faut placer ces deux scripts dans le même dossier que le dossier où **Anbody_Package** est situé.

1.1 Fichiers nécessaires

1.1.1 Fichiers .h5

Le script peut charger des données issues de fichiers **.anydata.h5**. Ces fichiers ne sont seulement capables de stocker des valeurs non-constantes d'une simulation.

1.1.2 Fichiers texte AnyFileOut

On peut aussi charger des constantes d'un fichier texte AnyFileOut comme des paramètres de simulation (nstep, muscle_recruitment type…) ou des paramètres du modèle Anybody créé.

1.1.3 Nom des fichiers

Ces deux fichiers doivent avoir le même nom.

1.2 Dossiers à créer

Au même endroit que les deux scripts, on doit créer un dossier de sauvegarde des résultats chargés (par exemple nommé **Saved Simulations**).

2 Charger les résultats

2.1 Structure des données

2.1.1 Structure des variables chargées

On peut charger des variables normales, les variables devant être chargées pour chaque muscle (mises dans le dossier **Muscles**) et des constantes (dans le dossier **Model informations**).

Abduction	dict	3	{'Description':'Angle d'abduction [°]', 'SequenceComposantes':['Total' ...
Elevation	dict	3	{'Description':'Angle d'élévation dans le plan de la scapula [°]', 'Se ...
Loaded Variables	dict	4	{'Variables':{'Abduction':{...}, 'Elevation':{...}}, 'Muscles':{'Delto ...
Model informations	dict	2	{'Paramètres de simulation':{'Case':'middle-normal', 'MuscleRecruitmen ...
Muscles	dict	27	{'Deltoideus lateral':{'Deltoideus lateral 1':{...}, 'Deltoideus later ...

Chaque variable a une description (qui sera affichée dans les graphiques) et des valeurs. Une variable peut être en 1 dimension donc n'a qu'une seule composante (`"Total"` par défaut)

Description	str	21	Angle d'abduction [°]
SequenceComposantes	list	1	['Total']
Total	Array of float64	(40,)	[15. 15.17024132 15.67986121 ... 119.32013879 119.82975868 ...

Le nom de la composante pourrait être renommé, par exemple `"Max"`

Description	str	13	Force max [N]
Max	Array of float64	(70,)	[0. 1.73913043 3.47826087 ... 116.52173913 118.26086957 ...
SequenceComposantes	list	1	['Max']

Ou bien pour un vecteur, on peut avoir plusieurs composantes (par défaut `["x", "y", "z"]`) et le total est calculé automatiquement. Le nom des composantes peuvent être changés par exemple pour indiquer des directions anatomiques.

Total	Array of float64	(70,)	[24.77227663 24.77273958 24.77416516 ... 26.86202325 26.87925055 26.8 ...
ML	Array of float64	(70,)	[24.74613031 24.7465477 24.74764493 ... 25.73527184 25.74330292 25.7 ...
IS	Array of float64	(70,)	[0.03946018 0.06356994 0.13626864 ... -5.21771382 -5.26471071 -5.2 ...
AP	Array of float64	(70,)	[1.13717484 1.13708492 1.13787549 ... 5.66034794 5.66209199 5.66266381 ...
SequenceComposantes	list	4	['Total', 'AP', 'IS', 'ML']
Description	str	30	Position du centre de pression

Ici, quand on a chargé la variable COP, on a indiqué que les axes x, y et z correspondaient dans l'ordre aux directions antéropostérieures, inférosupérieures et médiolatérales. La séquence de composantes devient donc [`"AP", "IS", "ML"`] avec un total qui est calculé.

2.1.2 Variables de muscles

Pour les muscles, on charge une liste de muscle qui peuvent être chacun composés d'une ou de plusieurs fibres musculaires (**muscle_part**).

Biceps brachii long head	dict	1	{'Biceps brachii long head':{'Fm':{...}, 'Ft':{...}, 'Activity':{...}, ...
Biceps brachii short head	dict	1	{'Biceps brachii short head':{'Fm':{...}, 'Ft':{...}, 'Activity':{...} ...
Coracobrachialis	dict	7	{'Coracobrachialis 1':{'Fm':{...}, 'Ft':{...}, 'Activity':{...}, 'F or ...
Deltoideus anterior	dict	5	{'Deltoideus anterior 1':{'Fm':{...}, 'Ft':{...}, 'Activity':{...}, 'F ...
Deltoideus lateral	dict	5	{'Deltoideus lateral 1':{'Fm':{...}, 'Ft':{...}, 'Activity':{...}, 'F ...
Deltoideus posterior	dict	5	{'Deltoideus posterior 1':{'Fm':{...}, 'Ft':{...}, 'Activity':{...}, ' ...
Downward Subscapularis	dict	5	{'Downward Subscapularis 1':{'Fm':{...}, 'Ft':{...}, 'Activity':{...}, ...
Infraspinatus	dict	7	{'Infraspinatus 1':{'Fm':{...}, 'Ft':{...}, 'Activity':{...}, 'F origi ...

Donc chaque variable de muscles est chargée de la même façon qu'une variable normale mais pour chaque partie de muscles (activité, force musculaire…).

Pour les muscles en plusieurs parties (ex : le deltoïde latéral a 4 parties), chaque partie est chargée séparément avec des numéros et on calcule un muscle qui combine les 4 parties.

Deltoideus lateral	dict	5	{'Fm':{'Sequence_Composantes':[...], 'Description':'Force musculaire [...
Deltoideus lateral 1	dict	5	{'Fm':{'Description':'Force musculaire [Newton]', 'SequenceComposantes ...
Deltoideus lateral 2	dict	5	{'Fm':{'Description':'Force musculaire [Newton]', 'SequenceComposantes ...
Deltoideus lateral 3	dict	5	{'Fm':{'Description':'Force musculaire [Newton]', 'SequenceComposantes ...
Deltoideus lateral 4	dict	5	{'Fm':{'Description':'Force musculaire [Newton]', 'SequenceComposantes ...

Cette combinaison de muscle prend chaque variable de toutes les parties et en combine chaque composantes (**combine_muscle_part_operations**).

Les opérations de combinaisons peuvent être : `"total"` ***(Par défaut)*****,** `"max"`**,** `"min"`**,** `"mean"` ou plusieurs opérations dans une liste

Par exemple, on peut combiner la force musculaire Fm en un total.

Description	str	25	Force musculaire [Newton]
SequenceComposantes	list	1	['Total']
Total	Array of float64	(70,)	[13.25195245 13.30393418 13.45999965 ... 20.47039009 20.32490026 20.2 ...

On peut calculer l'activité maximal d'un groupe de muscle et l'activité moyenne :

Description	str	23	Activité Musculaire [%]
Max	Array of float64	(70,)	[1.51083810e-08 1.50321319e-08 1.48082407e-08 ... 3.13844700e-10 3.10 ...
Mean	Array of float64	(70,)	[8.61133864e-09 8.57432507e-09 8.46558799e-09 ... 1.70056883e-10 1.68 ...
Sequence_Composantes	list	2	['Max', 'Mean']

Ou bien la force musculaire projetée selon 3 directions en un total et une moyenne pour chaque composante. Quand on combine le total, le nom combiné est le nom de l'opération (`"Max"`, `"Mean"`) et pour les autres composantes on aura comme nom `"Nom_Operation_Nom_Composante"` comme ici les composantes combinées sont:

```
["Total", "Mean", "Total_AP", "Mean_AP", "Total_IS", "Mean_IS", "Total_ML", "Mean_ML"]
```

Description	str	44	Force Musculaire à l'insertion du muscle [N]
Sequence_Composantes	list	8	['Total', 'Mean', 'Total_AP', 'Mean_AP', 'Total_IS', 'Mean_IS', 'Total ...
Mean	Array of float64	(70,)	[1.70690936e-10 1.70693019e-10 1.70684417e-10 ... 6.78829585e-02 6.88 ...
Mean_AP	Array of float64	(70,)	[2.62964890e-11 2.63121609e-11 2.63533092e-11 ... 1.26344306e-02 1.26 ...
Mean_IS	Array of float64	(70,)	[-2.76252333e-11 -2.76139861e-11 -2.75792540e-11 ... 8.79056311e-03 ...
Mean_ML	Array of float64	(70,)	[-1.06049819e-10 -1.06067497e-10 -1.06112189e-10 ... -4.04701494e-02 ...
Total	Array of float64	(70,)	[1.87760030e-09 1.87762321e-09 1.87752858e-09 ... 7.46712543e-01 7.56 ...
Total_AP	Array of float64	(70,)	[2.89261379e-10 2.89433770e-10 2.89886401e-10 ... 1.38978737e-01 1.39 ...
Total_IS	Array of float64	(70,)	[-3.03877566e-10 -3.03753847e-10 -3.03371794e-10 ... 9.66961942e-02 ...
Total_ML	Array of float64	(70,)	[-1.16654801e-09 -1.16674247e-09 -1.16723408e-09 ... -4.45171643e-01 ...

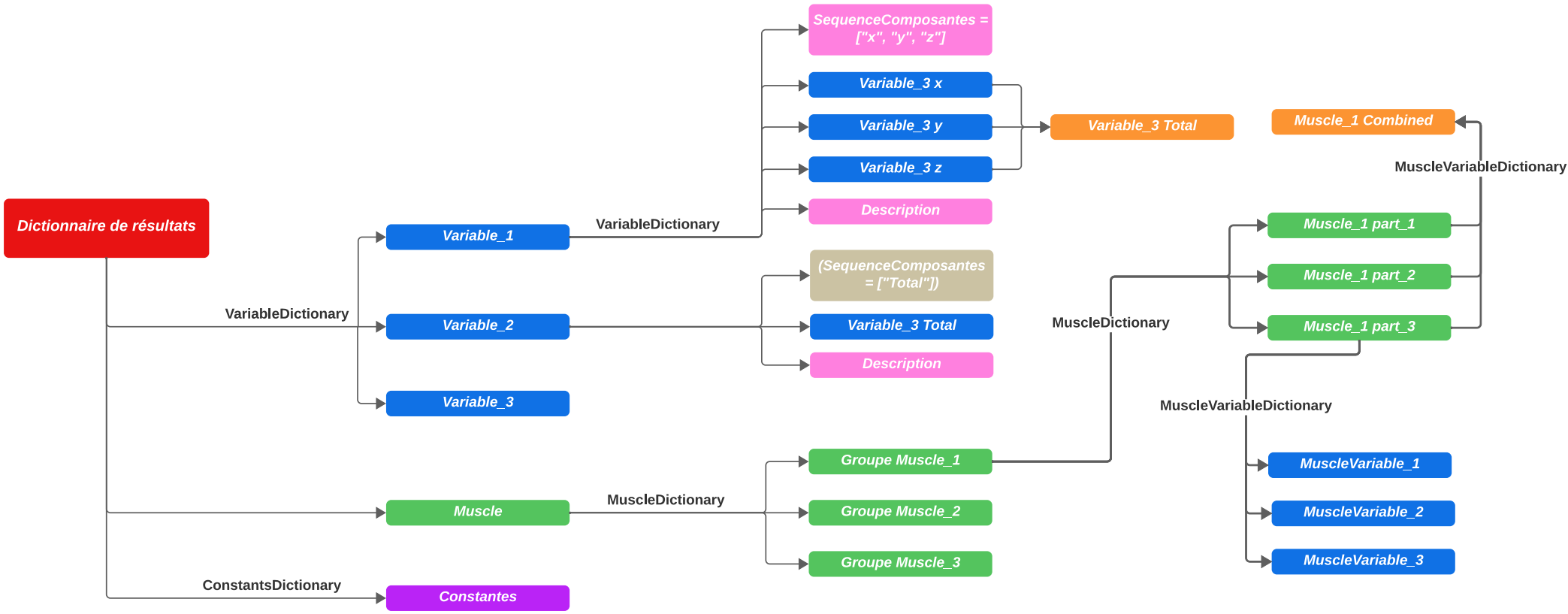
On peut donc avoir les variables de chaque fibre musculaire et aussi les variables de la combinaison de chaque partie d'un groupement de muscles. Attention quand on trace les graphiques, il faut donc prendre en compte que les fibres musculaires et le groupe musculaire combiné n'ont pas forcément les mêmes noms de composantes.

2.1.3 Constantes

Les constantes sont stockées dans **MuscleInformations** et peuvent être n'importe quelle valeur constante. Comme la position du mannequin, des paramètres de simulation (nStep, tEnd, le type de recrutement musculaire). Elles sont rassemblées par des catégories qui sont choisies lors du chargement.

Mannequin	dict	3	{'GlenohumeralFlexion':0.0, 'GlenohumeralAbduction':15.0, 'Glenohumera ...
ParametresFDK	dict	14	{'k0':Numpy array, 'k1':-1.68, 'k2':1.12, 'k3':0.9, 'k4':-0.05, 'kz':0 ...
Paramètres de simulation	dict	6	{'Case':'middle-normal', 'MuscleRecruitment':'MR_Polynomial', 'nStep': ...
Paramètres implants	dict	11	{'HumerusName':'TeteCeraver_51', 'GlenoidName':'GleneCeraver_T3', 'Cas ...
Positions initiales	dict	3	{'px':-0.001, 'py':-0.001, 'pz':-0.0013}

Structure des dictionnaires de résultats



Légende :

Dictionnaire de résultats | Muscle | Variable | Constantes | Calculé | Valeur par défaut | Information supplémentaires

2.1.4 Structure des données résultats

Il y a 3 niveaux de données résultats. Les simulations seules, avec seulement un seul fichier h5. Dans les graphiques c'est cette configuration qui est prise par défaut.

Une autre structure peut être de charger les résultats d'un même modèle qui aurait des cas de simulations différents. Par exemple le cas 1 a une force externe de 10N, le cas 2 a une force de 20N mais on reste sur le même modèle. Ou bien, chaque cas a des types de recrutement musculaire différents, comme ci-dessous où on a chargé 3 cas de simulation différents.

MR_MinMaxStrict	dict	16	{'Loaded Variables':{'Variables':{...}, 'Muscles':{...}, 'MuscleVariab ...
MR_Polynomial	dict	16	{'Loaded Variables':{'Variables':{...}, 'Muscles':{...}, 'MuscleVariab ...
MR_QuadraticAux	dict	16	{'Loaded Variables':{'Variables':{...}, 'Muscles':{...}, 'MuscleVariab ...

Ou bien un cas peut être une donnée de validation venant de la littérature

MR_MinMaxStrict	dict	16	{'Loaded Variables':{'Variables':{...}, 'Muscles':{...}, 'MuscleVariab ...
MR_Polynomial	dict	16	{'Loaded Variables':{'Variables':{...}, 'Muscles':{...}, 'MuscleVariab ...
MR_QuadraticAux	dict	16	{'Loaded Variables':{'Variables':{...}, 'Muscles':{...}, 'MuscleVariab ...
Wickham	dict	2	{'Muscles':{'Deltoideus lateral':{...}, 'Deltoideus anterior':{...}, ' ...

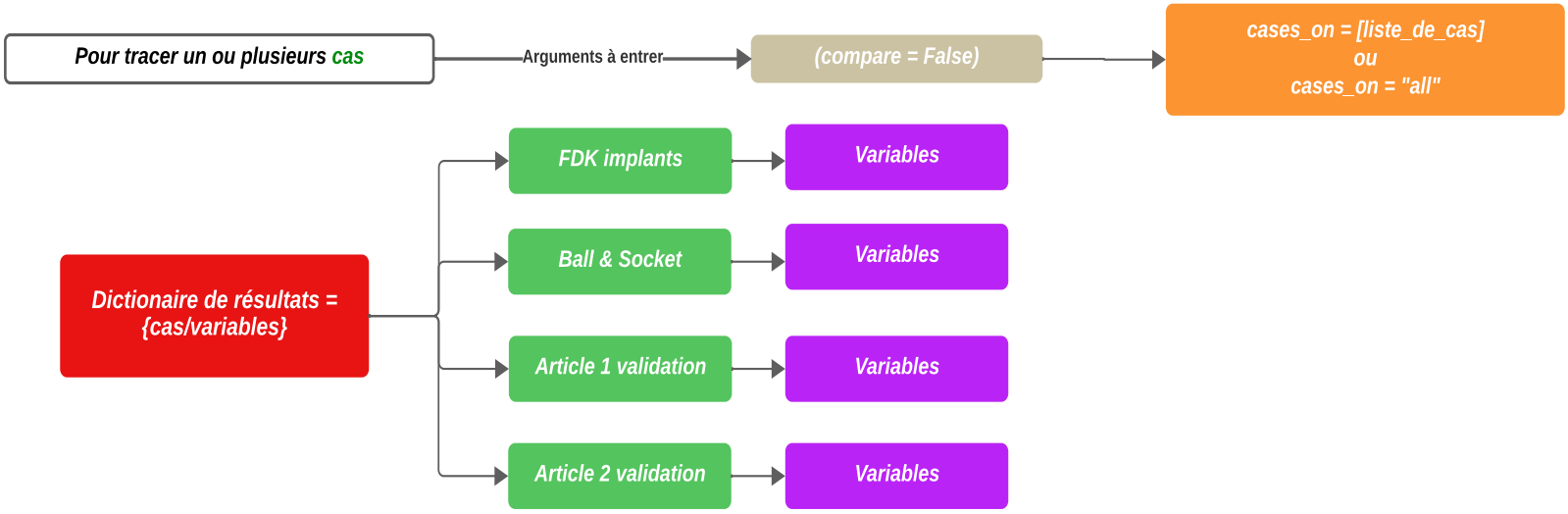


Figure 1: Résultats avec 4 cas de simulations

Pour tracer un graphique de plusieurs cas, il faudrait seulement entrer une liste d'un ou plusieurs cas dans la fonction graphiques utilisée en mettant par exemple :

`case_on = ["FDK implants", "Ball & Socket"]` pour sélectionner une liste de cas à tracer ou bien `cases_on = "all"` pour tracer tous les cas sur me même graphiques.

Enfin, on peut aussi comparer plusieurs modèles différents mais qui ont chacun les mêmes cas de simulation. Par exemple, un modèle avec une articulation glénohumérale normale et un modèle avec une articulation avec du FDK qui auraient chacune été simulées avec les mêmes trois types de recrutement musculaires.

On pourra donc par exemple comparer Normal et FDK pour chaque cas de recrutement musculaire. Ou bien, on peut avoir 2 modèles différents qui n'ont aucun cas de simulation, mais cela reviendrait à avoir 2 cas de simulations, la structure comparaison ne serait pas pertinente.

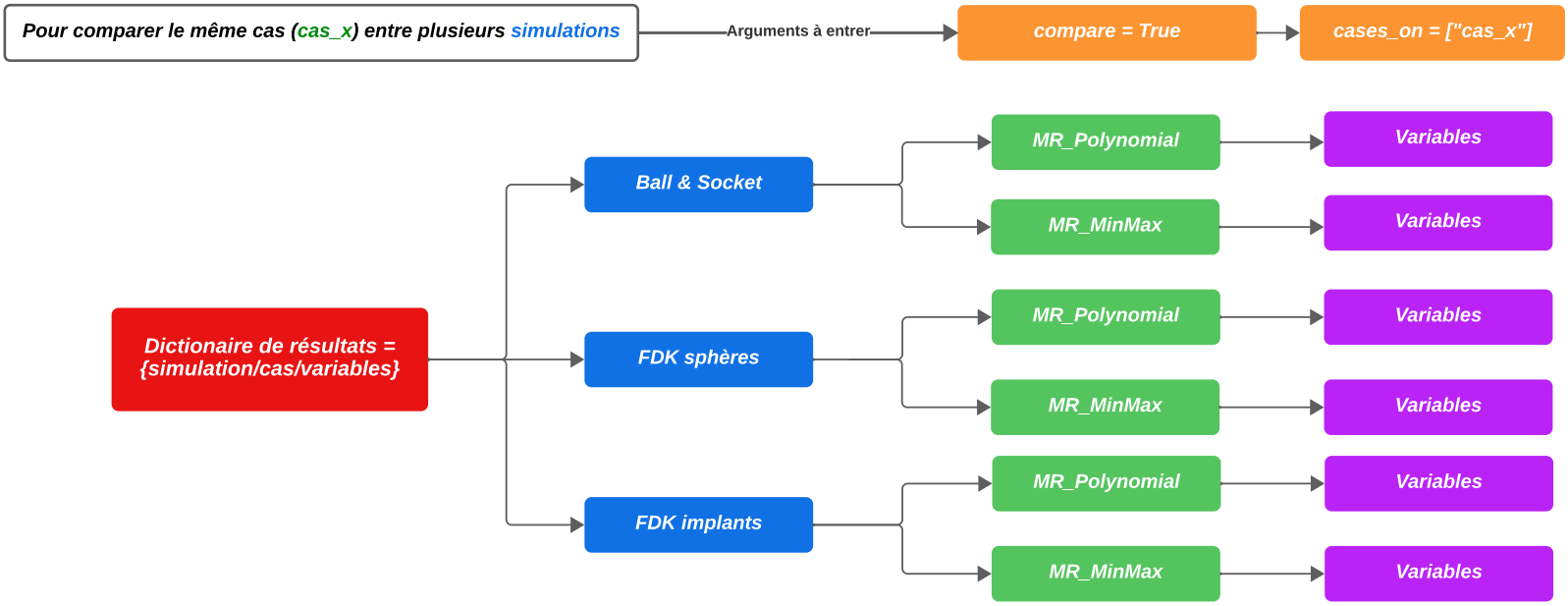


Figure 2: Structure pour comparer des cas de simulations entre plusieurs modèles

Dans ce cas, dans les fonctions de graphiques on doit indiquer `compare = True` et on ne peut afficher sur le graphique qu'un seul cas qui sera comparé entre toutes les simulations. Donc par exemple pour comparer le recrutement musculaire MinMax entre le FDK implants, FDK sphères et le Ball & Socket on entre en plus `cases_on = ["MR_MinMax"]`.

2.2 Chargement des variables

2.2.1 Liste des variables à charger: VariableDictionary

On liste toutes les variables voulues dans ce dictionnaire.

On indique le nom avec lequel on veut l'appeler, le nom dans Anybody, le chemin d'accès (à partir du dossier **Output**), la description de la variable qui sera indiquée dans les graphiques. Et les noms des composantes si on veut autre chose que la séquence par défaut x,y,z (3D) ou Total (1D).Chargement des variables musculaires

```
"Abduction": {"VariablePath": "Output.rotD", "VariableDescription": "Angle
d'abduction [°]"},

"COP": {"VariablePath": "Output.FileOut.COPlocalImplant",
"VariableDescription": "Position du centre de pression",
"SequenceComposantes": ["AP", "IS", "ML"]}
```

2.3 Chargement des variables musculaires

2.3.1 Liste des muscles à charger : MuscleDictionary

Dictionnaire qui choisit les muscles à charger et on choisit comment nommer et rassembler les fibres musculaires (appelées dans le coude **muscleparts**) en des groupes de fibre musculaires.

```
# Muscles
MuscleDictionary = {"Deltoideus lateral": ["deltoideus_lateral", "_part_", [1, 4]],
                    "Deltoideus posterior": ["deltoideus_posterior", "_part_", [1, 4]],
                    "Deltoideus anterior": ["deltoideus_anterior", "_part_", [1, 4]],
                    "Supraspinatus": ["supraspinatus", "_", [1, 6]],
                    "Infraspinatus": ["infraspinatus", "_", [1, 6]],
                    "Serratus anterior": ["serratus_anterior", "_", [1, 6]],
                    "Lower trapezius": ["trapezius_scapular", "_part_", [1, 3]],
                    "Middle trapezius": ["trapezius_scapular", "_part_", [4, 6]],
                    "Upper trapezius": ["trapezius_clavicular", "_part_", [1, 6]],
                    "Biceps brachii long head": ["biceps_brachii_caput_longum", "", []],
                    "Biceps brachii short head": ["biceps_brachii_caput_breve", "", []],
                    "Pectoralis major clavicular": ["pectoralis_major_clavicular", "_part_", [1, 5]],
                    "Pectoralis major sternal": ["pectoralis_major_thoracic", "_part_", [1, 10]],

                    "Pectoralis major": [["pectoralis_major_thoracic", "_part_", [1, 10]],
                                         ["pectoralis_major_clavicular", "_part_", [1, 5]]
                                         ],

                    "Pectoralis minor": ["pectoralis_minor", "_", [1, 4]],
                    "Latissimus dorsi": ["latissimus_dorsi", "_", [1, 11]],
                    "Triceps long head": ["Triceps_LH", "_", [1, 2]],
                    "Upper Subscapularis": ["subscapularis", "_", [1, 2]],
                    "Downward Subscapularis": ["subscapularis", "_", [3, 6]],
                    "Subscapularis": ["subscapularis", "_", [1, 6]],
                    "Teres minor": ["teres_minor", "_", [1, 6]],
                    "Teres major": ["teres_major", "_", [1, 6]],
                    "Rhomboideus": ["rhomboideus", "_", [1, 3]],
                    "Levator scapulae": ["levator_scapulae", "_", [1, 4]],
                    "Sternocleidomastoid clavicular": ["Sternocleidomastoid_caput_clavicular", "", []],
                    "Sternocleidomastoid sternum": ["Sternocleidomastoid_caput_Sternum", "", []],
                    "Coracobrachialis": ["coracobrachialis", "_", [1, 6]]
                    }
```

Par exemple on peut rassembler les muscles : **deltoideus_lateral_part_1**, **deltoideus_lateral_part_2**, **deltoideus_lateral_part_3**, **deltoideus_lateral_part_4** en un groupe nommé **Deltoideus lateral**. Donc dans ce cas, les muscles ont tous le même nom (**deltoideus_lateral**), suivi de "**_part_**" suivi d'un numéro de **1** à **4**.

Ce qui donne le code :

```
"Deltoideus lateral": ["deltoideus_lateral", "_part_", [1, 4]]
```

Un autre exemple, on voudrait rassembler les fibres claviculaires (1 à 5) et les fibres thoraciques (1 à 10) du pectoralis major en un seul groupe **Pectoralis mahor**. Donc le code pour charger ces 2 sous-groupes en un seul groupe est une liste avec 2 lignes similaires au code précédent :

```
"Pectoralis major": [["pectoralis_major_thoracic", "_part_", [1, 10]],
                     ["pectoralis_major_clavicular", "_part_", [1, 5]]
                    ]
```

Ce groupe aura donc 15 parties (les 10 thoraciques suivies des 5 claviculaires) et seront numérotées Pectoralis major 1 à 15.

On peut aussi charger une seule partie d'un muscle et le mettre seul dans un groupe, comme mettre le deltoïde latéral partie 1 dans une catégorie seule **Deltoideus lateral**.

```
"Deltoideus lateral": ["deltoideus_lateral", "_part_", [1]]
```

Ou bien le chef long du biceps qui n'a qu'une seule partie de muscle sans numéro à côté (biceps_brachii_caput_longum). On met `"", []` pour indiquer qu'il n'y a ni de nom ni de numéro indiquant une partie de muscle.

```
"Biceps brachii long head": ["biceps_brachii_caput_longum", "", []]
```

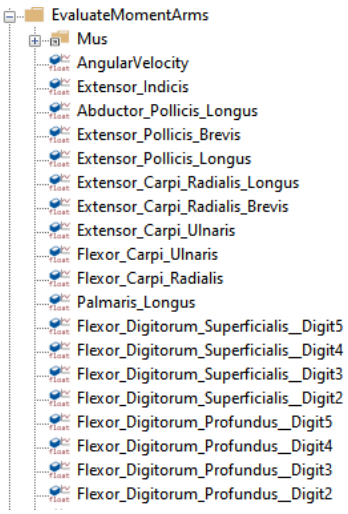
2.3.2 Liste des variables musculaires à charger : MuscleVariableDictionary

On indique la même chose que pour les variables (chemin de la variable, description, sequence de composantes si on veut les renommer), sauf que le dossier à indiquer est le dossier où tous les muscles se situent par exemple **Output.Model.BodyModel.Right.ShoulderArm.Mus** ou bien un raccourci à ce dossier nommé **Output.Mus** et on entre le nom de la variable à charger (Fm, Activity…) ou bien une variable qui est dans un sous-dossier du muscle (**Ins.r** pour avoir les coordonnées de l'insertion musculaire).

```
"Fm": {"MuscleFolderPath": "Output.Mus", "AnybodyVariableName": "Fm",
"VariableDescription": "Force musculaire [Newton]"},

"Position insertion": {"MuscleFolderPath": "Output.Mus",
"AnybodyVariableName": " Ins.r ", "VariableDescription": "Position de
l'insertion musculaire [m]", "SequenceComposantes": ["AP", "IS", "ML"]}
```

Ou bien par exemple si on a calculé les bras de leviers de chaque partie musculaire et les a placés dans une variable avec le nom du muscle correspondant :



On aura le chemin qui est **Output.EvaluateMomentArms** et le nom de la variable `""` pour indiquer que le nom de la variable est directement stocké dans une variable ayant le nom du muscle.

Et on ajoute aussi le/les type(s) d'opération(s) de combinaison des parties de muscles de chaque groupe musculaire (**combine_muscle_part_operations).** Si on n'indique rien, le total de chaque composante sera calculé, mais on peut entrer une liste contenant une ou plusieurs des opérations de combinaisons (`"total"`, `"min"`, `"max"`, `"mean"`).

2.4 Calculs avancés lors du chargement des variables

Ces opérations sont valables pour les variables normales et les variables de muscles.

2.4.1 Facteur multiplicateur

MultiplyFactor pour multiplier chaque composante d'une variable par un certain nombre. Peut être utile par exemple pour convertir de mètres en millimètres (**1000**) ou de radians vers degrés (**180/np.pi**). Donc on peut ajouter par exemple :`"MultiplyFactor"` **:** `1000`

2.4.2 Rotations

Cette opération peut allonger le temps de chargement des simulations, surtout si on le fait sur une variable musculaire sur beaucoup de simulations.

Si on veut appliquer une matrice de rotation qui change au cours du temps **à un vecteur seulement**, on peut entrer le chemin d'accès de cette matrice de rotation dans l'entrée **rotation_matrix_path** et on peut faire une transformation inverse en indiquant `"inverse_rotation" : True`**.**

Par exemple si une variable est un vecteur exprimé dans le repère global et qu'on veut le faire tourner pour le mettre dans le repère local d'un AnyRefNode, on prend la matrice de rotation **AnyRefNode_Name.Axes** de ce node. On doit appliquer une rotation inverse puisque la matrice **Axes** est la matrice de rotation permettant de transformer le repère local vers le repère global alors qu'on veut faire l'opération inverse. Par exemple pour mettre la variable dans la séquence de l'ISB dans le repère local de la scapula on ajoute :

```
"rotation_matrix_path": "Output.Seg.Scapula.AnatomicalFrame.ISB_Coord.Axes",
"inverse_rotation": True, "SequenceComposantes": ["AP", "IS", "ML"]
```

2.4.3 Inverser la direction d'un axe

Si on veut inverser la direction de l'axe y d'un vecteur 3D [x, y, z], c'est-à-dire multiplier les coordonnées en y par -1, on peut entrer :

```
"Composantes_Inverse_Direction": [False, True, False]
```

Donc la composante y sera multipliée par -1 et x et z seront inchangés.

2.4.4 Offset

On peut faire en sorte de décaler toutes les valeurs d'une ou plusieurs composantes d'une variable pour que sa première donnée soit à une certaine valeur. Par exemple, si la translation initiale varie beaucoup entre les simulations, on peut vouloir comparer les amplitudes au lieu de la valeur absolue. Donc on peut indiquer de décaler toutes les valeurs pour que la première valeur soit en 0,0,0:

```
"offset": [0, 0, 0]
```

Ou bien, si on veut décaler à 0 seulement la première composante mais pas les autres :

```
"offset": [0, False, False]
```

2.4.5 Sélectionner une ligne ou colonne particulière d'une matrice

Si un vecteur que l'on veut charger est contenu dans une ligne ou dans une colonne d'une matrice, on peut entrer le chemin de cette matrice et indiquer le numéro de la ligne ou de la colonne à charger. La première colonne ou ligne correspond au chiffre 0.

Sélectionner la première ligne : `"select_matrix_line": 0`

Sélectionner la seconde colonne : `"select_matrix_column": 1`

2.4.6 Calculer la direction d'un vecteur

Parfois on ne veut pas obtenir la valeur d'un vecteur mais sa direction. On peut donc calculer la direction du vecteur selon ses 3 composantes. Et le la composante "Total" sera donc 1 (la norme du vecteur directeur). Pour cela, il faut entrer :

```
"vect_dir": True
```

2.4.7 Chargement des forces aux extrémités des muscles

Cet argument est à mettre dans les variables musculaires.

Si on veut charger le vecteur de la force appliquée par le muscle à chacune de ses extrémités (origine et insertion) on doit charger la variable musculaire **`RefFrameOutput.F`**. Cette variable est une matrice (n x 3) dont chaque ligne représente le vecteur force dans les coordonnées globales à différents points du muscle. Ces points les ReferenceFrame qui ont été utiles pour définir le muscle. L'ordre est : l'origine, les via-points, l'insertion et les positions des wrapping surfaces.

On peut voir quelle ligne correspond à quelle ReferenceFrame dans la liste de pointeurs **RefFrameOutput. RefFrameArr**.

Pour des muscles simples, la matrice ne contient que 2 lignes (origine, insertion) et pour des muscles avec un ou plusieurs via-points, l'ordre est : origine, via_1, via_2, insertion. Donc la position correspondant à l'insertion varie donc on ne peut pas simplement utiliser **`select_matrix_row : 1`** car l'insertion n'est pas tout le temps à la deuxième ligne.

Donc on utilise l'argument **`select_muscle_RefFrame_output`** :

```
# Sélection de la variable musculaire
"AnybodyVariableName": "RefFrameOutput.F"

# Sélection de la force à l'origine
"select_muscle_RefFrame_output": "origine"

# Sélection de la force à l'insertion
"select_muscle_RefFrame_output": "insertion"
```

Cet argument compte le nombre de via-points de ce muscle et déduit la position de l'insertion. Et pour l'origine, la position est toujours la première ligne.

Ensuite, comme cette force est dans le repère global, on peut la projeter sur un repère, par exemple celui de la scapula, pour savoir si ce muscle (s'il y est attaché) effectue une force médiolatérale, inférosupérieure ou antéropostérieure. Ce qui peut donner ce code (disponible dans le template) et calculer la somme vectorielle de cette force pour chaque groupe de muscle et la moyenne :

```
"F insertion": {"MuscleFolderPath": "Output.Mus", "VariableDescription": "Force Musculaire à
l'insertion du muscle [N]",
"SequenceComposantes": ["AP", "IS", "ML"],
"combine_muscle_part_operations": ["total", "mean"]

# Sélection de la variable de RefFrameOutput
, "AnybodyVariableName": "RefFrameOutput.F"

# Sélection de la force à l'insertion
"select_muscle_RefFrame_output": "insertion",

# Rotation du vecteur dans le repère de la scapula
"rotation_matrix_path": "Output.Seg.Scapula.AnatomicalFrame.ISB_Coord.Axes",
"inverse_rotation": True,
}
```

2.5 Chargement des constantes : ConstantsDictionary

Dans ce dictionnaire, on doit premièrement entrer le chemin d'accès anybody de l'objet AnyFileOut. À la différence des variables venant des fichiers h5, ce chemin d'accès doit être le chemin absolu et pas le chemin depuis le dossier Output.

La première entrée de ce dictionnaire est donc :

```
"AnybodyFileOutPath": "Main.Study.FileOut"
```

Ensuite, les autres entrées du dictionnaire auront le nom de la catégorie de constante et contiendront la liste des constantes à mettre dans cette catégorie.

Mannequin	dict	3	{'GlenohumeralFlexion':0.0, 'GlenohumeralAbduction':15.0, 'Glenohumera ...
ParametresFDK	dict	14	{'k0':Numpy array, 'k1':-1.68, 'k2':1.12, 'k3':0.9, 'k4':-0.05, 'kz':0 ...
Paramètres de simulation	dict	6	{'Case':'middle-normal', 'MuscleRecruitment':'MR_Polynomial', 'nStep': ...
Paramètres implants	dict	11	{'HumerusName':'TeteCeraver_51', 'GlenoidName':'GleneCeraver_T3', 'Cas ...
Positions initiales	dict	3	{'px':-0.001, 'py':-0.001, 'pz':-0.0013}

Pour créer la catégorie Mannequin on entre :

```
"Mannequin": ["GlenohumeralFlexion", "GlenohumeralAbduction",
"GlenohumeralExternalRotation"]
```

2.6 Chargement des simulations

define_variables_to_load

Après avoir construit les 3 dictionnaires pour cette simulation on les combine avec la fonction **define_variables_to_load**. La variable créée servira à dire quelles variables charger dans le dossier h5.

Si on veut charger plusieurs fichiers h5 mais qui ont des noms de variables différents ou des structures différentes, il suffit de créer un autre dictionnaire de variable.

Par exemple, si on a un modèle FDK et un modèle BallAndSocket qui ont les mêmes muscles et constantes mais des variables différentes on pourra créer deux dictionnaires de variables :

FDK_VariableDictionary et **BallAndSocket_VariableDictionary** et créer deux dictionnaires de variables à charger :

```
# Première liste de variables pour les modèles avec FDK
FDK_Variables = define_variables_to_load(FDK_VariableDictionary,
MuscleDictionary, MuscleVariableDictionary, FDK_ConstantsDictionary)

# seconde liste de variables pour les modèles avec Ball And Socket
BallAndSocket_Variables =
define_variables_to_load(BallAndSocket_VariableDictionary, MuscleDictionary,
MuscleVariableDictionary, BallAndSocket_ConstantsDictionary)
```

2.6.1 Chargement d'un fichier h5

Pour charger un fichier h5 seul qui aura la structure avec une seule simulation, on utilise la fonction **load_simulation** en entrant le chemin d'accès au dossier contenant le fichier h5 et le fichier texte de constantes, le nom du fichier et les dictionnaires des variables à charger :

```
Results = load_simulation(FileDirectory, FileName, VariablesToLoad)
```

2.6.2 Chargement de plusieurs fichiers pour former des cas de simulation

Pour créer un dictionnaire de résultats avec cas de simulations, le processus est le même mais on entre une liste de noms de fichiers à la place d'un seul nom et aussi une liste contenant les noms à donner aux cas de simulations :

```
Results_Simulation_Cases = load_simulation_cases(SaveDataDir, Files,
CaseNames, FDK_Variables)
```

2.6.3 Fonction pour créer des dictionnaires de comparaison

La fonction **create_compared_simulations** permet d'assembler des dictionnaires de résultats **avec cas de simulation** en un dictionnaire de résultats qui compare des cas simulations. Il faut donc entrer le nom des simulations dans une liste :

```
simulation_names = ["nom_simulation_1", "nom_simulation_2",
"nom_simulation_3"]
```

Et ensuite, mettre dans les arguments qui suivent les différents dictionnaires de résultats avec cas de simulation.

```
ComparedSimulations = create_compared_simulations(simulation_names,
Results_Simulation_Cases_1, Results_Simulation_Cases_2,
Results_Simulation_Cases_3)
```

2.7 Combiner les variables de différents cas de simulations

La fonction **combine_simulation_cases** peut être utile si on veut par exemple faire la moyenne de plusieurs cas de simulations et combiner cette moyenne en un seul cas de simulation. On pourra ensuite sauvegarder le dictionnaire obtenu ou l'utiliser directement pour faire des graphiques.

Important : Les cas de simulations doivent avoir exactement les mêmes variables (donc avoir été chargées avec le même dictionnaire de variables)

```
combined_dictionary = combine_simulation_cases(result_dictionary, combine_cases, operation)
```

On entre premièrement le dictionnaire de résultats avec cas de simulation **result_dictionary**

Ensuite, on entre un dictionnaire dont chaque entrée a le nom du cas de simulation qui sera créé, et sa valeur est une liste des cas de simulation à combiner.

```
combine_cases = {"combined_case_name_1": [list_of_cases_to_combine_1]
                 "combined_case_name_2": [list_of_cases_to_combine_2]
                 "combined_case_name_3": [list_of_cases_to_combine_3]
                 }
```

Par exemple, si notre dictionnaire de résultats a des cas de simulations de **cas_1** à **cas_6**, où on veut faire la moyenne du cas 1 et 3 dans une catégorie **cas_13**, faire la moyenne de 2 et 4 dans **cas_24** et laisser le cas 5 tel qu'il est, on entre :

```
combine_cases = {"cas_13": ["cas_1", "cas_3"],
                 "cas_24": ["cas_2", "cas_4"],
                 "cas_5": ["cas_5"]
                 }
```

Il y a plusieurs choix d'opérations de combinaisons entre cas de simulation : la moyenne, la somme, trouver le minimum ou le maximum. On peut donc entrer dans **operation** une des valeurs suivantes : `"mean", "total", "min" "max"`.

2.8 Dans le cas où des simulations ont échoué

Dans certains cas, la simulation échoue à un certain pas de temps car par exemple les muscles sont overload ou que le FDK atteint une trop grosse erreur ou bien que deux os entrent en contact. On peut tout de même sauvegarder et charger ce fichier h5 et enlever les données aberrantes. Donc on peut entrer le numéro du premier pas de temps à enlever et les pas de temps suivants seront enlevés. **Attention, les numéros de pas de temps vont de 0 à nStep-1 dans Anybody. Il faut donc bien indiquer ce nombre.**

Par exemple, pour enlever tous les pas de temps après le pas de temps 51 pour une simulation sans cas de simulation. On entre :

```
Results = load_simulation(FileDirectory, FileName, VariablesToLoad, Failed = 51)
```

Dans le cas où on charge des cas de simulations, on entre une liste de nombre. Par exemple, le cas 1 a échoué à 51, le 2 à 57 et le 3 à 60, on entre donc :

```
Results_Simulation_Cases = load_simulation_cases(SaveDataDir, Files,
CaseNames, FDK_Variables, Failed = [51, 57, 60])
```

2.9 Sauvegarde des dictionnaires de résultats

Le chargement des simulations peut être long quand on en charge plusieurs. Pour ne pas avoir à les recharger à chaque fois qu'on crée des graphiques, les dictionnaires de résultat peuvent être sauvegardés sous forme de fichiers **.pkl** qui seront rechargés par le script qui permet de faire les graphiques.

On utilise donc la fonction **save_results_to_file**, en entrant le nom de la variable à sauvegarder, le chemin au dossier dans lequel on veut sauvegarder les dictionnaires de résultats (ici dans un dossier nommé **Saved Simulations**) et le nom du fichier à créer (qui peut être le même nom que la variable pour plus de clarté).

```
save_results_to_file (Results, "Saved Simulations", "Results")
```

3 Graphiques

3.1 Chargement des dictionnaires de résultats sauvegardés

La première étape pour faire des graphiques est d'importer les dictionnaires de résultats préalablement sauvegardés avec **save_results_to_file**.

On peut ensuite utiliser **load_results_from_file** pour réimporter la variable en donnant le chemin d'accès au dossier où est sauvegardé le fichier (ici **Saved Simulations**) et le nom du fichier.

```
Results = load_results_from_file("Saved Simulations", "Results")
```

3.2 Paramètres esthétiques de tous les graphiques

3.2.1 Contrôle de la taille de la police des graphiques

Les tailles de police sont déterminées par défaut, mais on peut augmenter la taille soit tous les textes du graphique en définissant une taille de référence qui permettra de définir automatiquement les différentes tailles de tous les éléments du graphique (référence : 10 par défaut)

```
# Contrôle de la taille de la police
matplotlib.rcParams.update({'font.size': 10})
```

Ou bien on peut contrôler individuellement la taille du texte de chaque partie du graphique :

```
# Titre des cases des subplots
matplotlib.rcParams.update({'axes.titlesize': 12})

# Titre du graphique
matplotlib.rcParams.update({'figure.titlesize': 12})

# Nom des axes
matplotlib.rcParams.update({'axes.labelsize': 10})

# Graduations des axes
matplotlib.rcParams.update({'xtick.labelsize': 10})
matplotlib.rcParams.update({'ytick.labelsize': 10})

# Légende
matplotlib.rcParams.update({'legend.fontsize': 10})
```

3.2.2 Contrôle du style de lignes selon le nom de la simulation ou du cas de simulation

Pour chaque donnée tracée sur un graphique, les couleurs sont choisies automatiquement et le style de ligne est une ligne pleine. Mais parfois on peut avoir besoin que la couleur d'un certain cas de simulation soit toujours le même pour rester consistent entre les graphiques ou on veut définir un style de ligne différent.

Ainsi, on peut définir un dictionnaire qui changera le style de la ligne en fonction du nom du cas de simulation tracé ou bien du nom de la simulation qu'on compare aux autres. Si le nom du cas de simulation n'est pas dans ce dictionnaire, son style sera déterminé automatiquement.

On peut y définir la couleur (`"color"`), le type de marqueur (`"marker"`, aucun marqueur par défaut), la tille du marqueur (`"markersize"`), le style de ligne (`"lignestyle"` ligne continue par défaut) et l'épaisseur de la ligne (`"linewidth"`). On peut trouver les noms de couleurs, les types de marqueurs et les types de lignes dans la documentation du package python matplotlib.

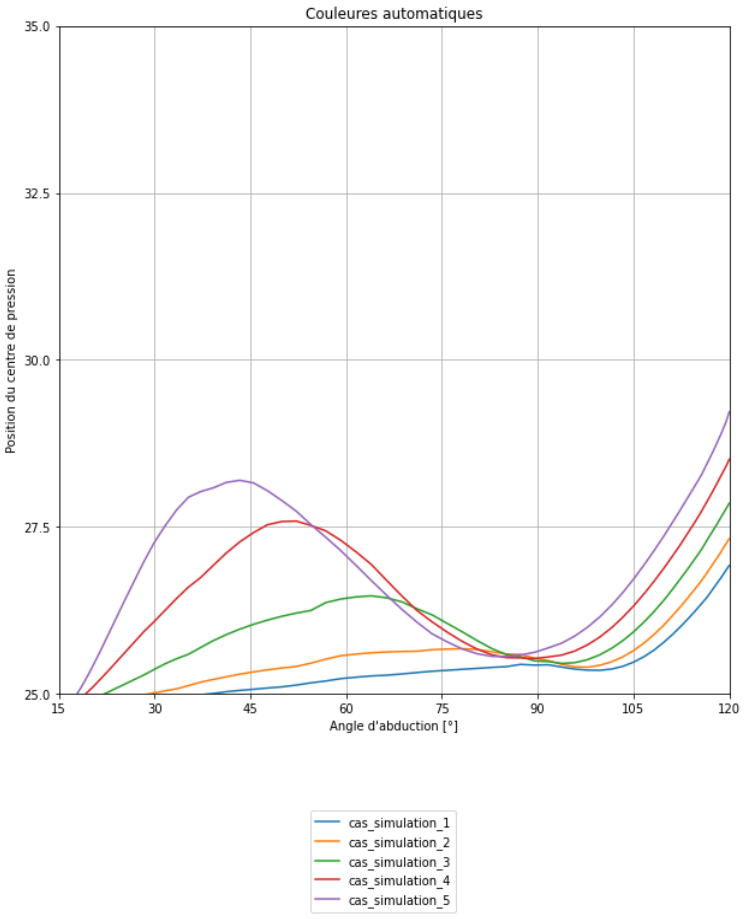


Figure 4: Couleurs automatiques

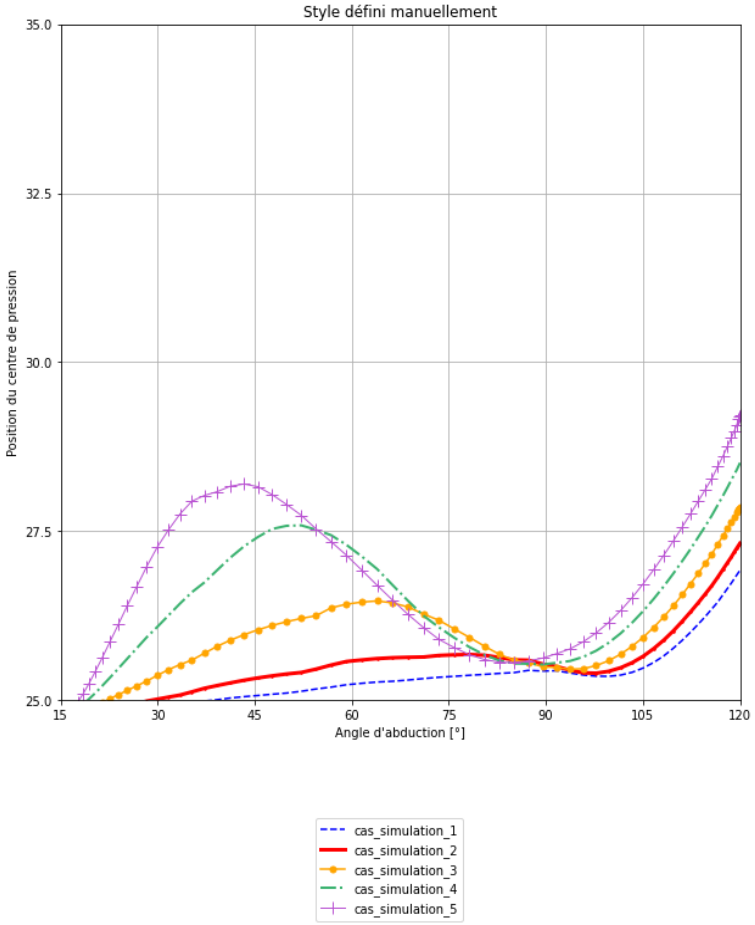


Figure 4: Couleur choisie en fonction du nom du cas de simulation

Les dictionnaire de style entré pour le graphique ci-dessus est :

```
SimulationsLineStyleDictionary = {
    "cas_simulation_1": {"color": "blue", "linestyle": "--"},

    "cas_simulation_2": {"color": "red", "marker": ".", "markersize": 3,
"linewidth": 3},

    "cas_simulation_3": {"color": "orange", "marker": "o", "markersize": 5},

    "cas_simulation_4": {"color": "mediumseagreen", "linestyle": "-.",
"linewidth": 2},

    "cas_simulation_5": {"color": "mediumorchid", "marker": "+",
"markersize": 10}
    }
```

Ensuite, ce dictionnaire est entré dans la fonction **define_simulations_line_style** et tous les graphiques qui seront fait après cette fonction auront ce style appliqué.

```
define_simulations_line_style(SimulationsLineStyleDictionary)
```

On pourrait donc avoir plusieurs dictionnaires de styles, il suffirait de réappeler cette fonction avant de tracer des graphiques avec un nouveau style.

3.3 Choix du texte de la légende

Dans les graphiques, une légende indique le nom du cas de simulation tracé ou le nom de la simulation qui est comparée selon le nom donné dans le chargement. Mais parfois on veut avoir une description plus complète. On crée donc un dictionnaire qui contiendra ces descriptions. Si le nom mis dans la légende n'est pas dans le dictionnaire, son nom restera inchangé.

Par exemple à la place des noms des cas de simulation on voudrait mettre la valeur de la force externe utilisée pour ce cas de simulation. Et à la place du nom de l'auteur d'un article de validation, on voudrait la citation complète de l'article avec la date. Mais on a oublié de déclarer la description pour le **cas de simulation 4.**

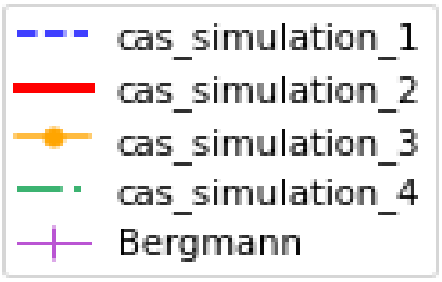


Figure 6: Légende automatique

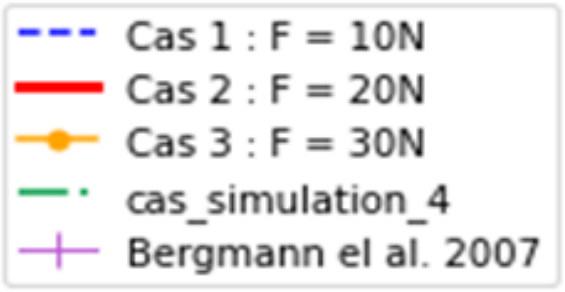


Figure 6: Légende avec description

La légende modifiée a été déterminée avec un dictionnaire contenant le nom des cas de simulation associés à leur description et en mettant ce dictionnaire dans la fonction **define_simulation_description.**

```
SimulationDescriptionDictionary = {
    "cas_simulation_1", "Cas 1 : F = 10N",
    "cas_simulation_2", "Cas 2 : F = 20N",
    "cas_simulation_3", "Cas 3 : F = 30N",

    "cas_simulation_5", "Bergmann et al. 2007"
}

# Définition des descriptions des simulations et cas de simulation
define_simulation_description(SimulationDescriptionDictionary)
```

Comme pour les styles de lignes, on peut avoir plusieurs versions de ce dictionnaire de description, si par exemple on veut faire plusieurs versions de graphiques avec des descriptions dans deux langues différentes.

3.4 Structure des fonctions de graphiques

Il existe trois types de fonction de graphiques qui prennent toutes en charge les 3 types de données (résultats avec 1 cas de simulations, résultats à plusieurs cas de simulation et résultats avec comparaison de cas de simulation). Les fonctions ont toutes des structures similaires.

Il existe 3 fonctions de graphique différente (**graph**, **muscle_graph**, **COP_graph**) mais leur structure sont très similaires. Elles ont des arguments qui sont obligatoires et d'autres qui sont facultatifs car ils ont des valeurs par défaut.

```
graph(data, "variable_x", "variable_y", "figure_title", compare=False,
cases_on=False, composante_x="Total", composante_y=["Total"])
```

3.4.1 Arguments obligatoires

On entre le dictionnaire des résultats à utiliser, le nom de la variable en x, le nom de la variable en y et le titre du graphique.

3.4.2 Arguments pour indiquer la structure de donnée utilisée

Ensuite, on spécifie si on compare des cas de simulations si c'est le cas `compare = True` (**`False`** par défaut sans avoir besoin de le mettre)

On entre le nom des cas de simulations à afficher sur le graphique si on en a :

- Si on compare, ne mettre qu'un seul cas de simulation : `cases_on=["nom_cas"]`
    - La légende du graphique indiquera le nom des simulations tracées et leur couleur
- Si on ne compare pas, mettre soit `cases_on="all"` pour tracer tous les cas, ou mettre une liste du/des cas à tracer `cases_on = ["cas_1", "cas_3"]`
    - La légende du graphique indiquera les noms des cas de simulations tracés et leur couleur

3.4.3 Sélection des composantes de la variable à entrer

Par défaut, c'est la composante "Total" de la variable qui sera tracée.

En x, on ne peut tracer qu'une seule composante, donc pour par exemple tracer la composante "x" entrer : `composante_x="x"`

En y, dans certains cas on peut tracer plusieurs composantes **(voir les cas particuliers dans les flowcharts des fonctions)**. Donc `composante_y` est une liste (même si on ne met qu'une seule composante). Par exemple pour tracer la composante nommée "y" on indique `composante_y = ["y"]` et pour tracer x et y on entre `composante_y = ["x", "y"]`

3.4.4 Fonction graph

On utilise cette fonction pour tracer des variables normales.

```
graph(data, "variable_x", "variable_y", "figure_title", compare=False,
cases_on=False, composante_x="Total", composante_y=["Total"])
```

3.4.5 Fonction muscle_graph

On utilise la fonction **muscle_graph** fonction pour tracer des variables de muscles.

```
muscle_graph(data, "muscle_name", "variable_x", "variable_y", "figure_title",
compare=False, cases_on=False, composante_x="Total", composante_y=["Total"],
MusclePartOn=False)
```

La seule différence avec **graph** est qu'on doit aussi indiquer en deuxième argument le nom du muscle à tracer.

On indique le nom du muscle sans numéro. Par défaut c'est le muscle combiné qui sera tracé mais on peut choisir les numéros des parties de muscles à tracer `MusclePartOn = [1,2]` ou `MusclePartOn=[1]`, `MusclePartOn="allparts"` pour tracer toutes les parties musculaires sauf la partie combinée et `MusclePartOn="all"` pour tracer toutes les parties et le muscle combiné (**pour ce cas, ne fonctionne que si les composantes ont les mêmes noms entre le total et les parties**).

Il y a des cas particuliers où on ne peut pas tracer plusieurs parties de muscles ou bien plusieurs composantes en y, se référer au f**lowchart** de cette fonction.

3.4.6 COP_graph

La fonction **COP_graph** est faite si on veut tracer la même variable en x et y mais avec une composante différente et permet aussi de tracer un contour (par exemple le contour d'un implant si on veut tracer le centre de pression).

```
COP_graph(data, COP_contour=None, variable="COP", "figure_title",
compare=False, cases_on=False, composantes=["x", "y"])
```

Ici on indique juste le nom de la variable à tracer (**COP** par défaut), les composantes en x et y `composantes=["x", "y"]` par défaut. Et on peut mettre dans COP_contour une liste de coordonnées x et y à tracer en plus de la variable tracée choisie. Avec la première colonne, les coordonnées en x et la seconde les coordonnées en y.

define_COP_contour

Fonction qui peut être utilisée pour définir le contour à partir d'un fichier par exemple un fichier picked points (.pp).

```
COP_contour = define_COP_contour("Nom_fichier", "type de fichier")
```

3.5 Premade graphs

On peut créer des scripts de graphiques automatiques qui combinent plusieurs fonctions de graphique pour faire des graphique complexes automatiquement. Tous ces scripts sont des fonctions mises dans le fichier **LoadOutput/PremadeGraphs**, et pour les utiliser dans le script d'analyse des résultats on fait **PremadeGraphs.nom_fonction.** Les arguments sont généralement très similaires aux fonctions normales mais avec des arguments supplémentaires.

3.5.1 muscle_graph_from_list

Cette fonction sert à tracer des variables d'une liste de muscles (`MuscleList`) dans un subplot dont on doit spécifer la taille (`subplot_dimension = [dimension_x, dimension_y]`).

```
PremadeGraphs.muscle_graph_from_list(data, MuscleList, subplot_dimension,
variable_x, variable_y, figure_title, cases_on=False, compare=False,
composante_y=["Total"], MusclePartOn=False)
```

3.5.2 graph_all_muscle_fibers

Cette function permet de tracer des variables du muscle combiné sur un graphique seul et de toutes les parties de muscles sur un second graphique. On donne donc une liste de muscles à tracer (`MuscleList`).

```
PremadeGraphs.graph_all_muscle_fibers(data, MuscleList, variable_x,
variable_y, composante_y_muscle_part=["Total"],
composante_y_muscle_combined=["Total"], cases_on=False, compare=False)
```

On doit aussi spécifier quelle composante(s) utiliser pour la variable en y pour le muscle combiné (`composante_y_muscle_combined`) et pour les parties de muscles (`composante_y_muscle_part`) si jamais leur valeur par défaut (`["Total"]`) pour les deux ne convient pas.

3.5.3 graph_by_variable à ajouter et renommer graph_by_category

3.6 Arguments esthétiques facultatifs pour les fonctions de graphique

Ces arguments peuvent être entrés dans n'importe quelle fonction qui font des graphiques (**graph**, **muscle_graph**, **COP_graph**) et dans les fonctions de **PremadeGraphs** si cet argument n'est pas déjà utilisé par cette fonction.

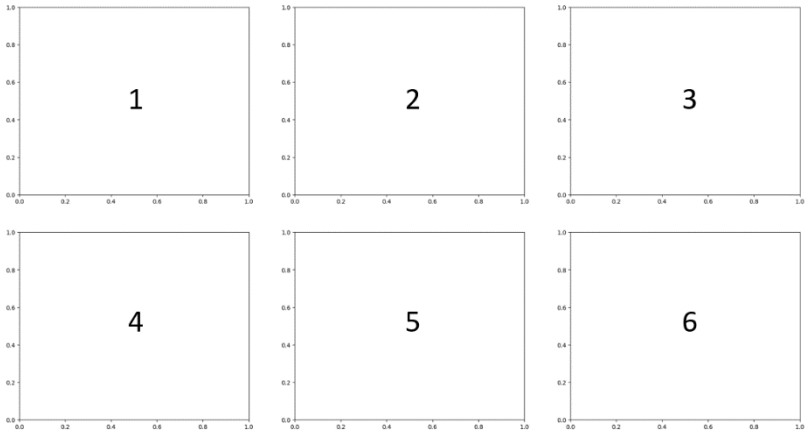
3.6.1 Subplot

L'argument **subplot** permet de placer un certain graphique dans une case. Un même graphique peut contenir un graphique de muscle sur une case et un graphique de COP sur une autre. Cet argument est souvent est déjà défini par les fonctions de **PremadeGraphs**, il n'est donc pas nécessaire de le redéfinir pour ces fonctions.

C'est un dictionnaire ayant la structure suivante :

```
subplot = {"dimension": [nbe_lignes, nbe_colonnes], "number":
numéro_de_la_case}
```

On doit en premier tracer la première case et ensuite aller jusqu'à la dernière. Les numéros vont de 1 à nbe_cases_verticalement * nbe_cases_horizontalement. Si on a dimension [2, 3] le graphique créé aura cette forme et ces numéros de case :



On peut ajouter dans **subplot** l'entrée `"last_subplot": True`

Si on n'a pas rempli toutes les cases d'un graphique mais qu'on veut obliger la légende à s'afficher. Par exemple, quand on arrive à la case 5 qui est la dernière case à remplir on peut mettre :

```
subplot = {"dimension": [2, 3], "number": 5, "last_subplot": True}
```

3.6.2 Titre d'une case de subplot

On peut aussi mettre un titre à cette case en ajoutant dans la fonction de graphique :

`subplot_title="titre_de_la_case"` qui mettra ce titre à la case en cours.

3.6.3 Limites du graphique

On peut contrôler les limites des axes x et y de chaque case individuellement ou on peut faire en sorte que toutes les cases aient la même taille si les limites mises automatiquement ne conviennent pas.

```
xlim = [xmin, xmax], ymin = [ymin, ymax]
```

On peut aussi contrôler la graduation des axes et la grille. On peut entrer les intervalles de graduations en x et en y :

```
grid_x_step = nombre, grid_y_step = nombre
```

Aussi, au lieu d'entrer les limites pour chaque case, on peut entrer des limites et grilles pour le dernier et ajouter `same_lim = True`. Si une des limites n'est pas indiquée, la limite manquante appliquée sur toutes les cases du graphique sera mise automatiquement en mettant la limite choisie automatiquement la plus large possible.

Par exemple, si on n'entre que les limites en x et pas la limite en y pour la dernière case en activant **same_lim**, si la première case a ses valeurs en y qui vont de **0** à **100** et la seconde case de **-100** à **10**. Alors tous les graphiques auront leur limite en y de **-100** à **100**.

Bug report : Ne mettre same_lim qu'au tout dernier graph (si on fait un add_graph, mettre le same_lim sur le add_graph et pas avant). Dû à un bug de matplotlib pas résolu.

3.6.4 Annotation de graphiques

On peut mettre des annotations sur les graphiques pour indiquer où la variable en y a atteint un maximum, minimum, pic maximal, pic minimal ou quelle est sa première valeur ou sa dernière valeur. On doit pour cela mettre **`graph_annotation_on=True`**.

Mode d'annotation **annotation_mode** (first, last, max, min, max_peak, min_peak)

Par défaut la variable qu'on inspecte est la variable tracée en x (sauf pour COP où c'est Abduction) et on regarde par défaut la composante "Total". Une légende indiquera aussi, quelle est cette variable et sa composante.

- Pour changer la variable à inspecter (Seulement une variable non musculaire) :
  `annotation_variable = "nom_variable"`
- Pour changer la composante à inspecter :
  `annotation_composante = "nom_composante"`

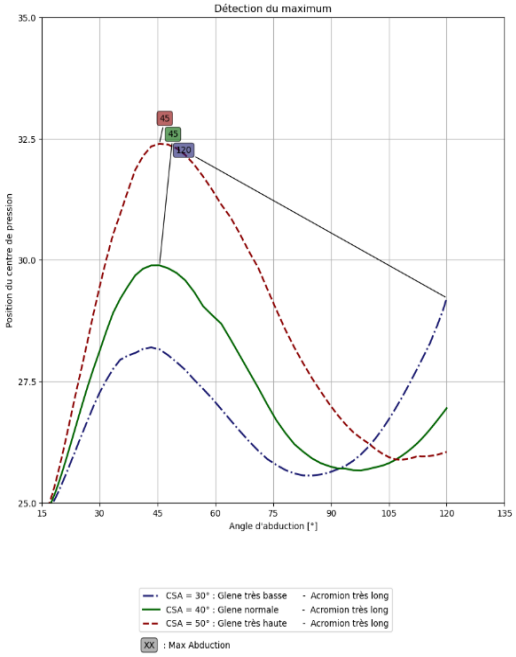


Figure 9 : Par défaut on détecte le maximum de la variable en x

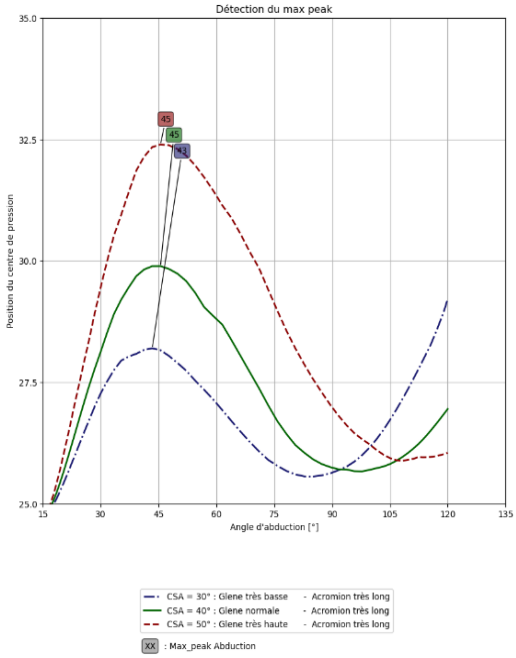


Figure 9 : annotation_mode = "max_peak'

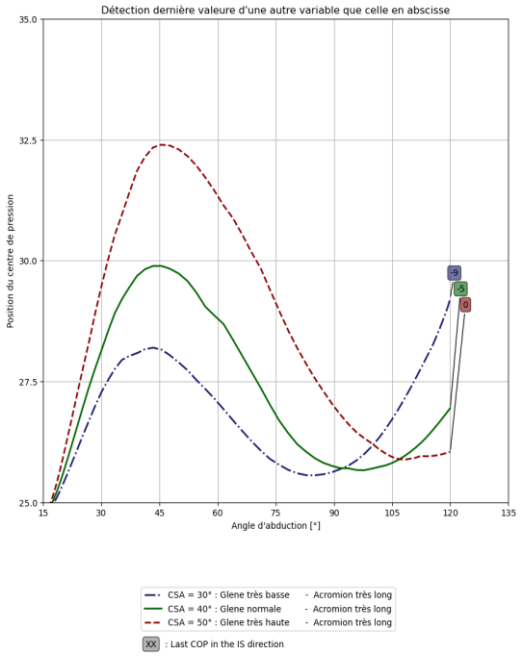
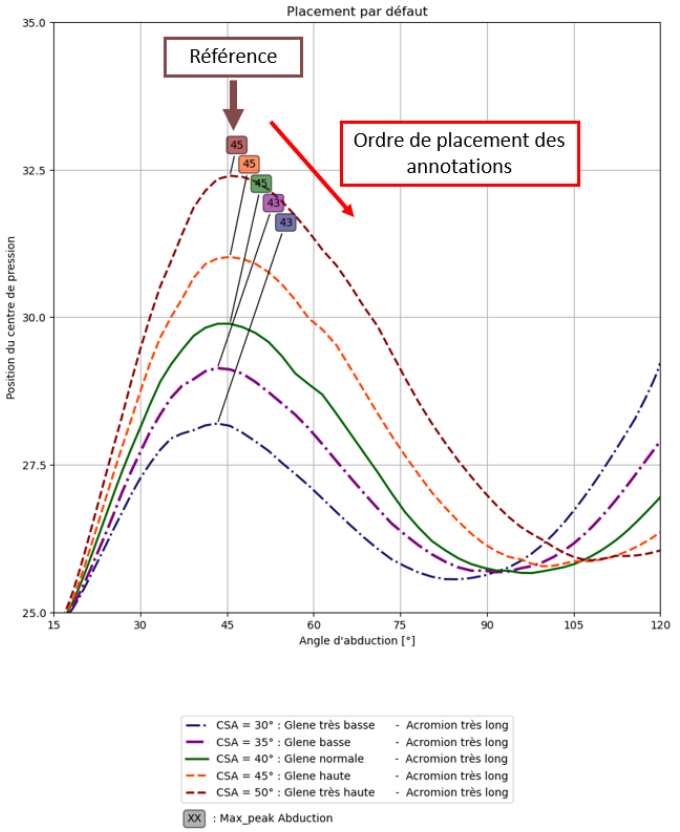


Figure 9 : annotation_mode='last', annotation_variable='COP', annotation_composante='IS'

3.6.5 Mode de placement des annotations

Parfois le placement par défaut des annotations n'est pas adéquat car elles sont trop proches des courbes et nuisent à la lisibilité.

Par défaut, une boîte est choisie comme référence (la boîte la plus haute en y), elle est décalée du point qu'elle indique (vers le haut et la droite). Ensuite la deuxième boîte la plus haute est placée par rapport à la première avec un certain espacement (en bas à droite).



On peut donc influer sur :

Le choix de la référence : **annotation_reference_mode** elle est choisie en fonction des coordonnées qui sont pointées par les annotations

- `annotation_reference_mode = "max_y"` (Par défaut)
- Valeures possibles : `"max_y", "max_x", "min_x", "min_y"`

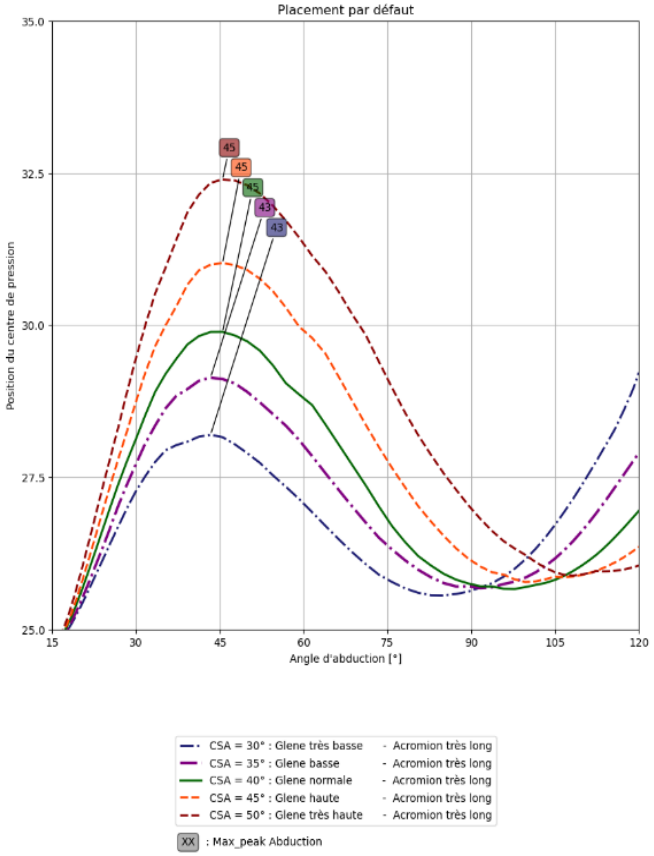


Figure 11 : Placement par défaut

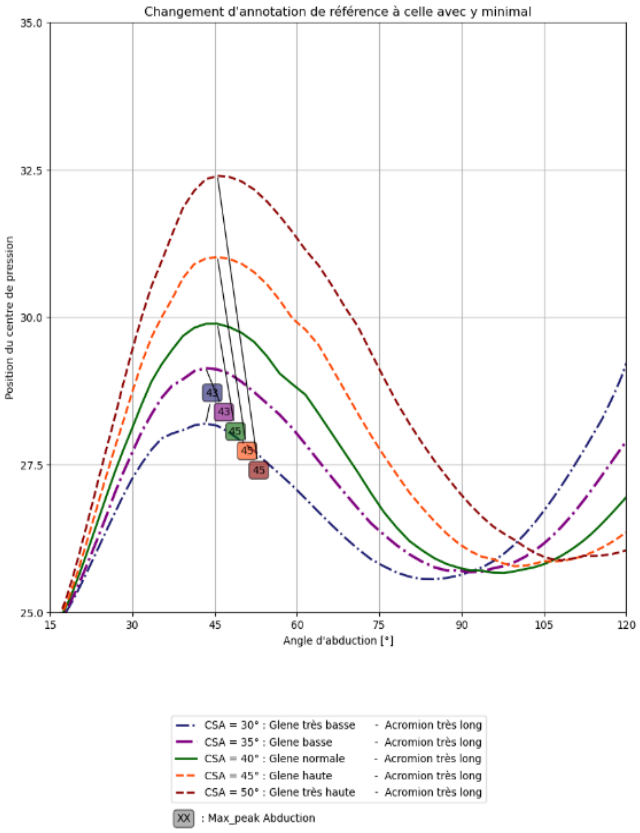


Figure 11: annotation_reference_mode="min_y"

- Le décalage de la référence par rapport au point qu'elle indique : **annotation_reference_offset**
    - Par défaut : `annotation_reference_offset = [0, 3]`
        - Ne se décale pas en x et se décale de 3 fois la longueur de la police vers le haut :

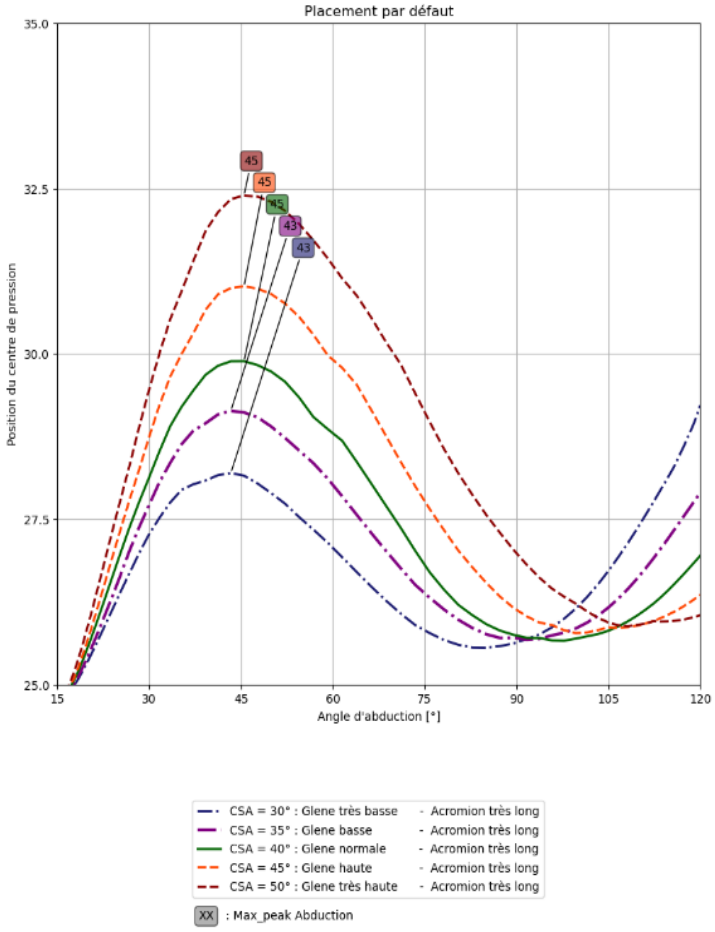


Figure 13 : Décalage de la référence par défaut

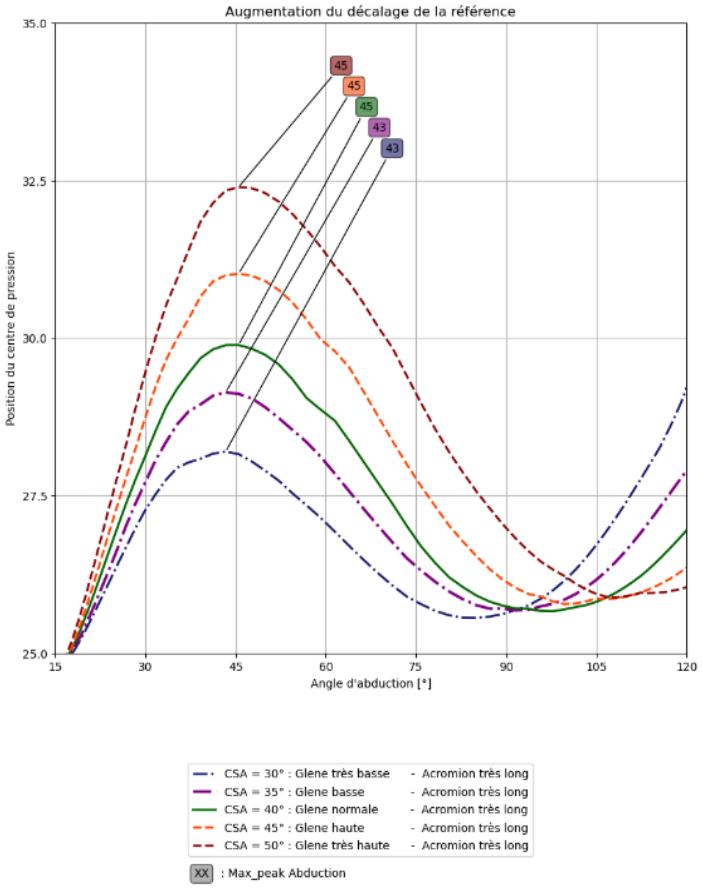


Figure 13 : Augmentation du décalage de la référence à [6, 12]

- Le décalage entre les annotations : **annotation_offset**
    - Par défaut : `annotation_offset = [0.8, -2.1]`
        - Se décale vers la droite de 0.8 fois la longueur de la boîte de référence

          Se décale vers le bas de 2.1 fois la hauteur de la boîte de référence

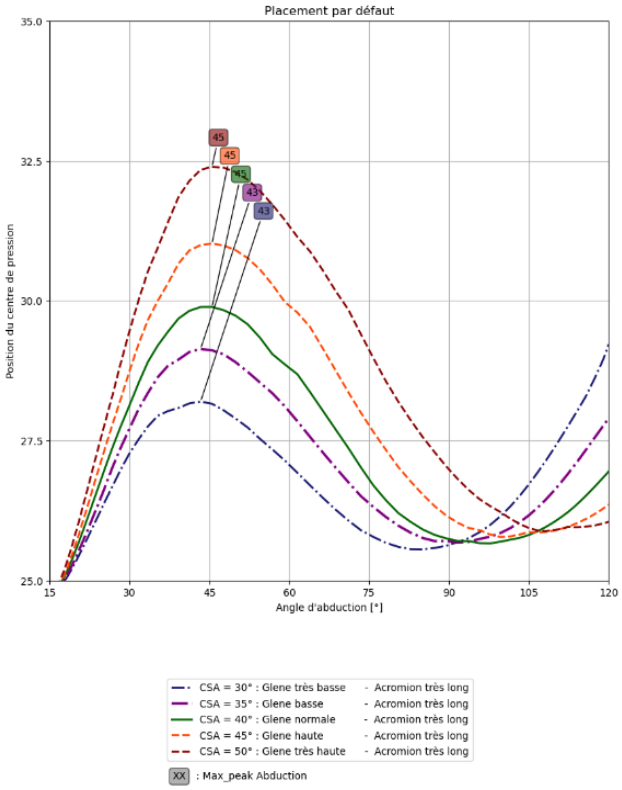


Figure 15 : Décalage entre annotation par défaut

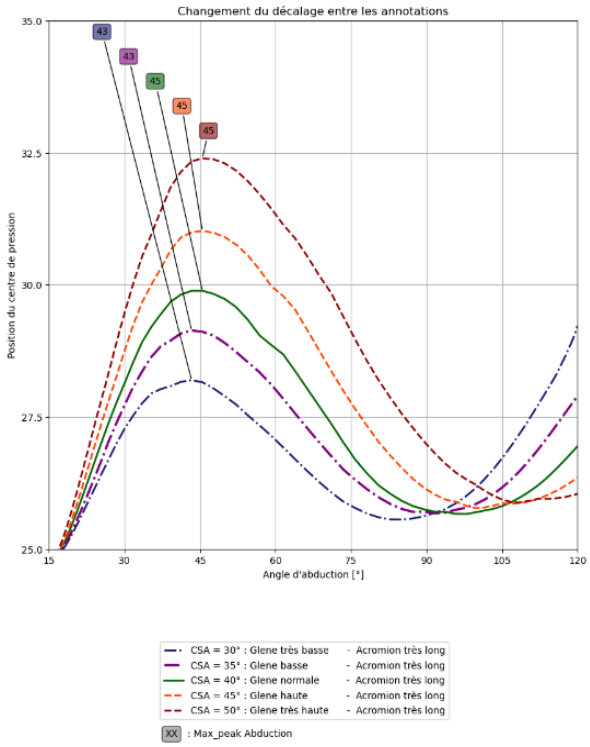


Figure 15 : Décalage entre annotation changé à [-2, 3]

3.6.6 Taille du graphique

On peut augmenter la taille du graphique avec l'argument **figsize.** Cet argument est utile pour les cas où il y a beaucoup de cases de subplot pour aider à ce qu'elles ne se superposent pas ou augmenter leur taille en contrôlant la taille de la figure entière.

```
figsize = [taille_horizontale, taille_verticale]
```

3.6.7 Position de la légende

La position de la légende est par défaut en bas au centre `legend_position = "lower center"`

Elle peut être mise au centre gauche `legend_position = "center left"`

3.6.8 add_graph et label

D'habitude quand on essaye de tracer deux fois d'affilée sur la même case de subplot, le graphique précédent est effacé. On peut mettre `add_graph = True` si on veut ajouter une courbe d'un autre dictionnaire de résultat. Par exemple si on veut ajouter une donnée sur la case 2 du subplot :

```
# Graphique du premier dictionnaire de données sur la case 2
graph(Results_1, "Abduction", "GHLin", "Translation en fonction de l'angle
d'abduction", cases_on="all", subplot = {"dimension": [1, 2], "number": 2})

# Graphique du second dictionnaire de données sur la même case
graph(Results_2, "Abduction", "Translation", "Translation en fonction de
l'angle d'abduction", cases_on="all", subplot = {"dimension": [1, 2],
"number": 2}, add_graph=True)
```

Cet argument fonctionnerait aussi pour les **PremadeGraphs**, du moment que les deux données peuvent être tracé avec cette fonction. Il y aurait quelques ajustements à faire sur les argumenst d'entrées avec les **graph_by_category** par exemple mais cela serait possible.

On peut aussi ajouter `label="description de la donnée"` pour obliger une certaine donnée unique à prendre une autre description dans la légende que celle automatiquement prévue par le code.

4 Comparer les résultats à la littérature

Pour valider les résultats, on peut vouloir ajouter aux graphiques des résultats des données issues de la littérature. Comme pour les résultats, on doit tout d'abord les charger et ensuite les tracer sur des graphiques.

4.1.1 Chargement depuis un fichier excel

Un fichier excel préremplis est situé dans le dossier **Template (Template_importation_littérature)**. Chaque onglet rassemble toutes les données de plusieurs auteurs sur une variable en particulier. On peut charger des variables normales et des variables musculaires. Pour chaque cas, la variable qui nous intéresse (variable en y) doit être accompagnée de données pour l'axe des x.

4.1.2 Prérequis sur les données

Sur un seul onglet, les données doivent être similaires pour tous les auteurs. C'est-à-dire que pour tous les auteurs, les variables et x et en y doivent avoir le même nom.

Les variables peuvent être avec une seule composante ou bien avec plusieurs composantes et cela peut être différent entre plusieurs auteurs. Pour plusieurs composantes, chaque composante en y devra être décrite en fonction d'une variable en x unidimensionnelle (qui doit être la même pour le même auteur).

Par exemple, si on mesure la translation dans 3 dimensions en fonction de l'abduction, chaque composante de translation est exprimée en fonction de l'abduction. Les valeurs en x peuvent différer mais doivent représenter la même mesure (l'abduction totale).

4.1.3 Setup de l'onglet de variable

Le nom de l'onglet doit donc être adapté selon le type de variable. Soit **Nom_variable** pour une variable normale, soit **Muscle.Nom_variable** pour une variable musculaire. Il existe donc deux onglets de template pouvant être copiés (ils peuvent rester dans le document, les template ne seront pas chargés).

4.1.4 Informations sur les variables

Ensuite on doit entrer les informations des variables en x et en y, leur description qui servira à être affichée dans les graphiques, un facteur multiplicateur. Toutes les valeurs en rouge sont les valeurs à modifier.

Variable Informations	
Variable x	**Name_Variable_x**
VariableDescription	Description_variable_x
MultiplyFactor	1.00
Variable y	**Name_Variable_y**
VariableDescription	Description_variable_y
MultiplyFactor	1.00

4.1.5 Interpolation des valeurs

Dans certain cas, on pourrait avoir besoin d'interpoler les valeurs entrées avant de les charger. On peut donc entrer plusieurs informations d'interpolation.

Interpolation	On/Off
Minimal value	min_value_variable_x
Maximum value	max_value_variable_x
Number of interpolation points	100

On doit indiquer **On** pour l'activer. On peut l'utiliser dans plusieurs cas.

Par exemple, si on prend les valeurs à la main depuis un graphique, il se peut que par exemple un mouvement d'abduction (variable en x) ait été effectué de 0 à 90° mais que sur les données entrées, à cause de la précision de la prise des données à la main, les données aillent de -1° à 89°. Dans ce cas on peut activer l'interpolation et indiquer 0 dans la case **min_value_variable_x** et 90 dans la case **max_value_variable_x.** Ainsi, **avant** de sauvegarder les valeurs, le code trouvera une fonction d'interpolation qui décrit la courbe **f(variable_x) = variable_y** et créera une nouvelle liste de points en x (depuis le minimum jusqu'au maximum) avec **100** points intermédiaires par défaut (cette valeur peut être changée dans la dernière case des informations d'interpolation). Ensuite de nouvelles valeurs en y seront calculées pour correspondre aux nouvelles valeurs en x.

L'autre cas est le cas où les données en y ont plusieurs composantes. Chaque composante en y doit être décrite en fonction d'une variable en x. Dans le cas où les données en x ne sont pas les mêmes

valeurs pour chaque composante, on peut activer l'interpolation pour que chaque composante puisse être exprimée en fonction du même vecteur x. Donc similairement, une série de point est créée en x (du minimum entré au maximum entré) et chaque composante ont leur valeur qui sont recalculées.

Dans les autres cas : une composante ou plusieurs composantes mais avec les mêmes valeurs en x, l'interpolation peut être désactivée sauf si on voudrait calculer plus de points intermédiaires.

4.1.6 Entrée des données

Pour un auteur avec une variable en y unidimensionnelle, les champs à remplir sont :

Auteur_2_Name	
Name_Variable_x Total	Name_Variable_y Component_1

Et pour une variable y en 3D :

Auteur_1_Name					
Name_Variable_x Total	Name_Variable_y Component_1	Name_Variable_x Total	Name_Variable_y Component_2	Name_Variable_x Total	Name_Variable_y Component_3

Pour des variables musculaires, la seule différence est l'ajout d'une ligne pour le nom du muscle qui auront chacun une variable en x et en y.

Auteur_1_Name					
Muscle_Name_1		Muscle_Name_2		Muscle_Name_3	
Name_Variable_x Total	Name_Variable_y Component_1	Name_Variable_x Total	Name_Variable_y Component_1	Name_Variable_x Total	Name_Variable_y Component_1

4.2 Charger le fichier excel

Depuis le code de chargement des résultats (ou du code des graphiques puisque ce chargement prend peu de temps), on peut utiliser la fonction **load_literature_data** en indiquant le nom du fichier (`file_name`) et chemin d'accès au fichier excel (`directory_path`)

```
Results_Literature = load_literature_data(file_name, directory_path)
```

On obtient donc un dictionnaire dont chaque entrée contient un dictionnaire avec des cas de simulations portant sont les noms des auteurs pour chaque variable (donc onglet) du document excel.

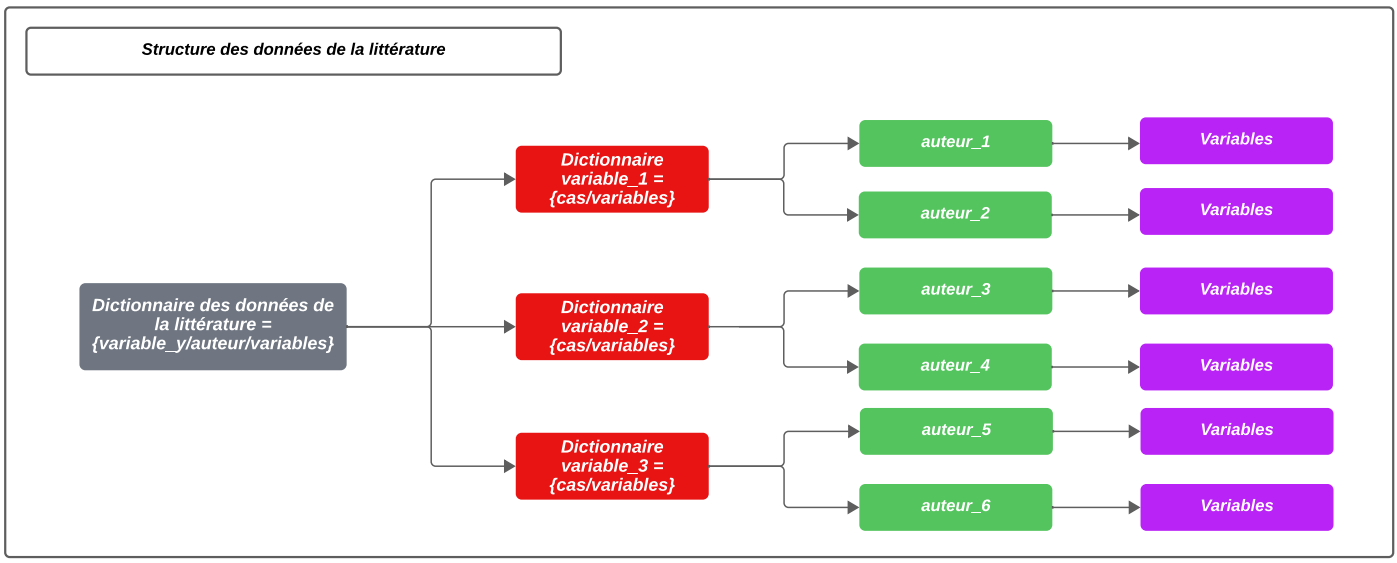


Figure 16 : Structure des données de la littérature

4.3 Ajouter à un graphique existant

Chaque dictionnaire de variable de littérature peut être utilisé comme un dictionnaire de résultats avec cas de simulation. On peut donc tracer ses données ou bien l'ajouter à des graphiques déjà existants qui tracent les mêmes variables en x et en y. Il y a deux façons de tracer des résultats qui se comparent à la littérature.

4.3.1 Tracer en fusionnant les dictionnaires

On peut soit fusionner un dictionnaire de résultats avec cas de simulation avec un dictionnaire de variable de littérature. Et utiliser ce nouveau dictionnaire comme un dictionnaire de résultats avec cas de simulation.

```
# Fusion des dictionnaires
Results_Compare_Litterature_Variable_1 = {**Results, **Results_Litterature["Variable_1"]}

# Trace les résultats comparés à la littérature
graph(Results_Compare_Litterature_Variable_1, "Abduction", "Translation", "Translation en fonction de l'angle d'abduction", cases_on="all")
```

Et il faudrait faire cela pour chaque variable de la littérature et il faudrait que les noms des variables et les séquences de composantes correspondent exactement pour que cela soit utilisable, ce qui n'est pas forcément le cas.

4.3.2 Avec l'argument add_graph

On a vu précédemment que l'argument `add_graph` des fonctions de graphiques pouvait être utile pour ajouter des données à un graphique déjà existant. On peut donc utiliser cela pour tracer en premier un graphique issu de résultats et faire un second graphique sur la même case mais avec les données issues de la littérature.

Par exemple, dans nos résultats on veut tracer la translation (appelée ***GHLin*** dans nos données) en fonction de l'angle d'abduction. Et comparer la translation trouvée dans la littérature (appelée ***Translation*** dans les données de littérature) en sélectionnant le dictionnaire contenant les données de translation (`Results_Literature["Translation"]`) et en ajoutant `add_graph=True`.

```
# Graphique de nos données de translation (GHLin)
graph(Results, "Abduction", "GHLin", "Translation en fonction de l'angle
d'abduction", cases_on="all")

# Graphique des données de la littérature de translation (Translation)
graph(Results_Literature["Translation"], "Abduction", "Translation",
"Translation en fonction de l'angle d'abduction", cases_on="all",
add_graph=True)
```

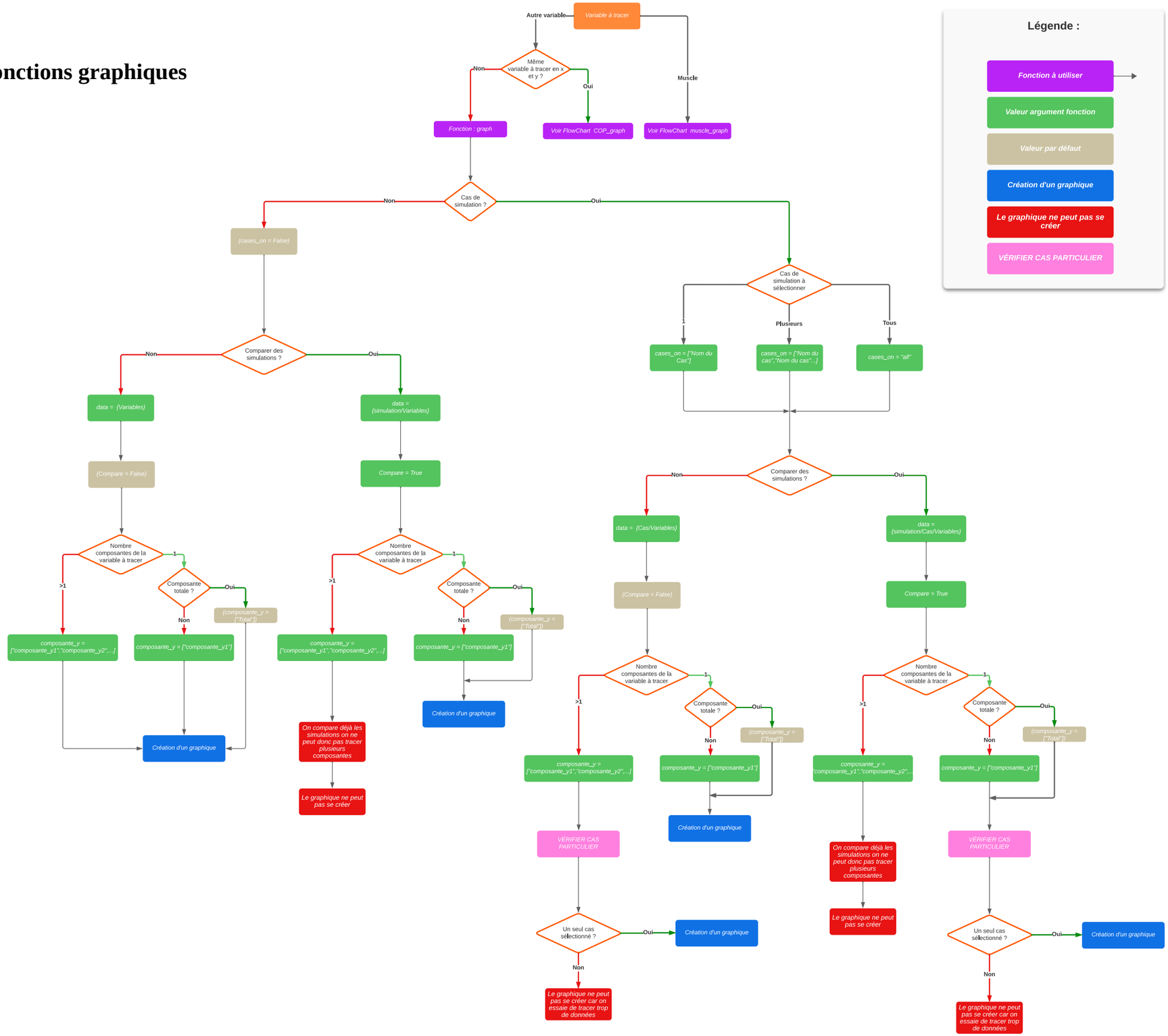
On voit que le nom des variables peut être différents entre nos données et les données de littérature, l'important est que les variables en x et y aient la même description et aient des données décrivant les mêmes quantités, mais les noms des variables et des composantes importe peu.

On peut aussi ajouter une entrée avec le nom des auteurs au dictionnaire `SimulationsLineStyleDictionary` pour par exemple donner plus d'informations sur l'article en donnant par exemple l'année et le nombre de participants dans cette étude :

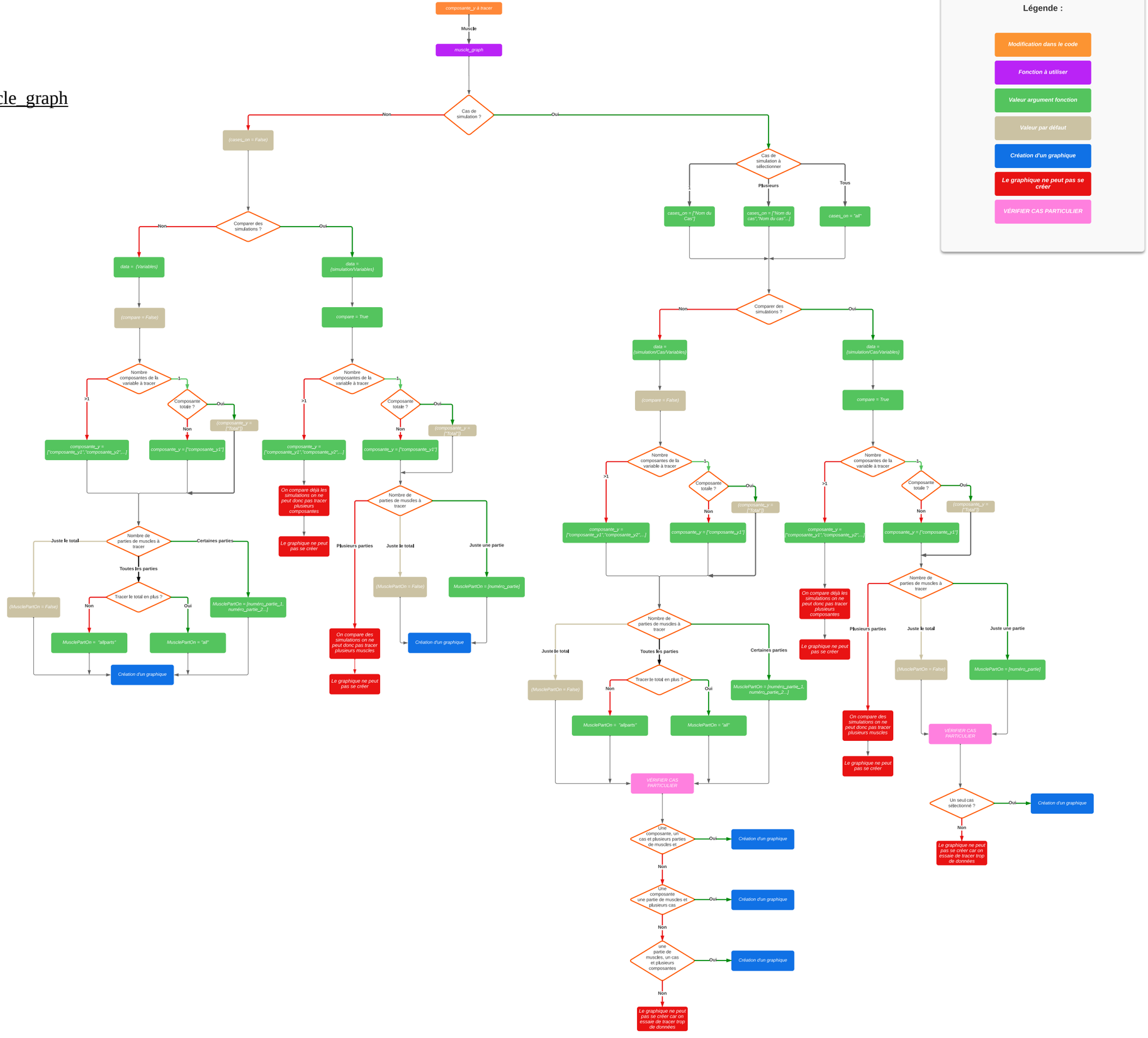
```
SimulationsLineStyleDictionary = {"Wickham": "Wickham et al. 2010, n=24"}
```

5 FlowChart des fonctions graphiques

A. FlowChartgraph

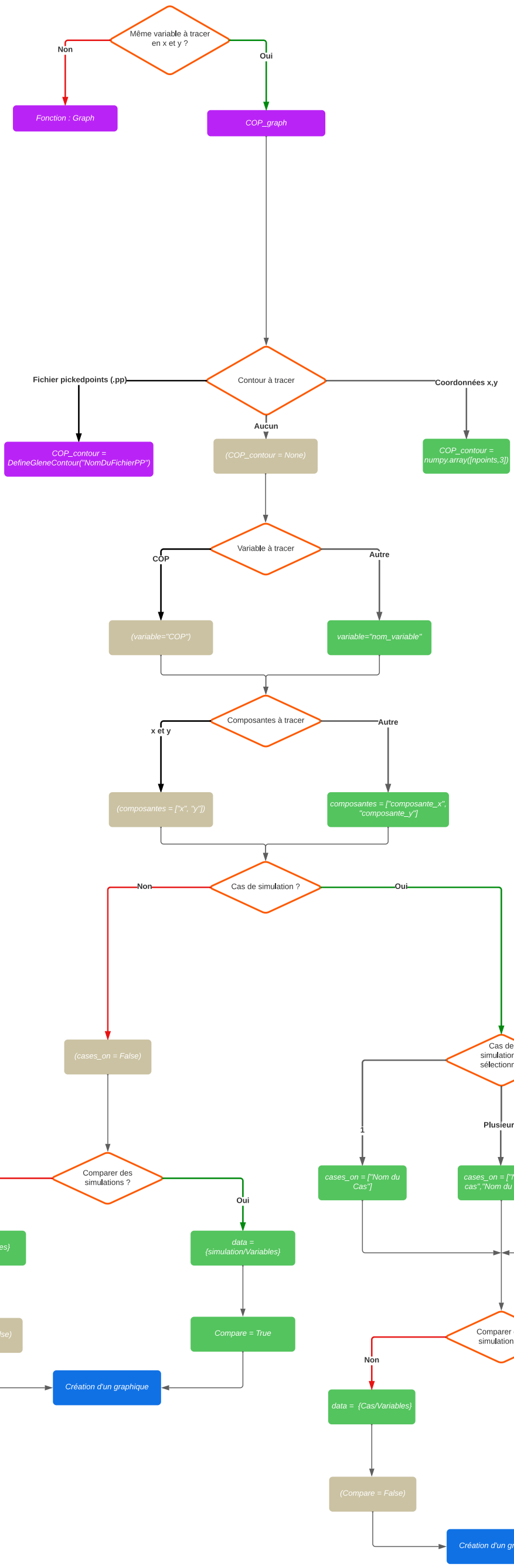


B. Flowchart muscle_graph



C. FlowChart COP_graph

Aucun cas particulier qui empêcherait de tracer le graphique.



Légende :

- *Modification dans le code*
- *Fonction à utiliser*
- *Valeur argument fonction*
- *Valeur par défaut*
- *Création d'un graphique*
- *Le graphique ne peut pas se créer*
- *VÉRIFIER CAS PARTICULIER*

D. FlowChart Structure des données

E. FlowChart structure des variables