Université Abdelmalek Esaadi

Faculté des sciences et techniques - Tanger

1ère année cycle d'ingénieur GéoInformation

Module : algorithmique et programmation

Rapport de projet de fin de module

Encadré par : Pr. Sanae KHALI ISSA

Réalisé par : AKAYAB Nadir / EL BAKALI Ibtissam

Année universitaire : 2022/2023

Sommaire :

INTRODUCTION :

Python est un langage de programmation populaire et polyvalent. Il a été conçu pour être lisible et facile à apprendre, ce qui en fait un excellent choix pour les débutants et les professionnels.

Python est utilisé dans de nombreux domaines, y compris la science de données, l'analyse de données, l'apprentissage automatique, les jeux, la finance et la création de sites web.

Il existe de nombreuses bibliothèques et Framework Python qui permettent de réaliser facilement de nombreuses tâches courantes en programmation.

Voici les principaux avantages de Python :

Facile à apprendre et à utiliser : Python a une syntaxe simple et directe, ce qui le rend facile pour les débutants à apprendre et à comprendre.

C'est un langage de haut niveau : Python est un langage de haut niveau, ce qui signifie qu'il est abstrait du matériel informatique et du système d'exploitation. Cela facilite l'écriture et le débogage de code.

C'est un langage interprété : Python est un langage interprété, ce qui signifie qu'il n'a pas besoin d'être compilé avant d'être exécuté. Cela facilite le test et le débogage du code rapidement.

C'est un langage à typage dynamique : En Python, vous n'avez pas besoin de spécifier le type de données.

Python et les interfaces graphiques (GUI) :

Pour créer une interface graphique pour votre programme, Il existe de nombreux outils et bibliothèques Python que vous pouvez utiliser Voici quelques exemples populaires :

Tkinter : Tkinter est une bibliothèque de base fournie avec Python qui permet de créer des interfaces graphiques en utilisant le kit de développement Tk.

PyQt : PyQt est une bibliothèque qui vous permet de créer des interfaces graphiques en utilisant le Framework Qt.

PyGTK : PyGTK est une bibliothèque qui vous permet de créer des interfaces graphiques en utilisant le GTK+ toolkit.

WxPython : wxPython est une bibliothèque qui vous permet de créer des interfaces graphiques en utilisant le wxWidgets toolkit.

PySide : PySide est une bibliothèque qui vous permet de créer des interfaces graphiques en utilisant le Framework Qt.

Il y a beaucoup d'autres bibliothèques et outils disponibles pour la création d'interfaces graphiques en Python. Le choix dépend de vos préférences et de vos besoins spécifiques pour votre projet

Python Tkinter GUI

Voici une liste de quelques méthodes fondamentales de Tkinter :

Méthode	Description
.Tk()	Crée une fenêtre principale
.Label(parent, text=...)	crée un widget d'étiquette de texte
.Button(parent, text=..., command=...)	Crée un widget bouton
. Entry(parent)	Crée un widget champ de saisie de texte
.Canvas(parent, width=..., height=...) :	Crée un widget toile pour dessiner des formes et des images
.pack()	Organise le widget dans la fenêtre en utilisant un empilage de boîtes
.grid()	Organise le widget dans la fenêtre en utilisant une grille de cellules
.place()	Organise le widget dans la fenêtre en spécifiant explicitement ses coordonnées
.destroy()	Détruit le widget et libère les ressources système associées
.set(value)	Définit la valeur du widget (par exemple, le texte dans un champ de saisie de texte)
.get()	Retourne la valeur du widget (par exemple, le texte dans un champ de saisie de texte)
.winfo_children()	Retourne une liste de tous les widgets enfants d'un widget parent

.mainloop()	Démarre la boucle principale de l'interface graphique, qui attend et gère les événements utilisateur
.config(option)	Modifie les options d’un widget
. title()	Définir le titre de la fenêtre principale
.text()	Créer une zone de texte dans une interface graphique
.insert(index, string)	Insère du texte à l’index spécifié
.delete(start, end)	Supprime du texte entre les indices de début et de fin spécifié
.geometry()	Définir la taille et la position de la fenêtre principale
.Frame()	Créer un conteneur de widgets
.place_forget()	Retirer un widget de l’interface graphique, sans le détruire
.StringVar()	Lier la valeur d’un widget à une variable de type chaine de caractère
.IntVar()	Lier la valeur d’un widget à une variable de type entier

L'objectif :

Réalisation d'une calculatrice scientifique avec interface graphique en utilisant le langage python.

Les fonctionnalités essentielles à programmer sont :

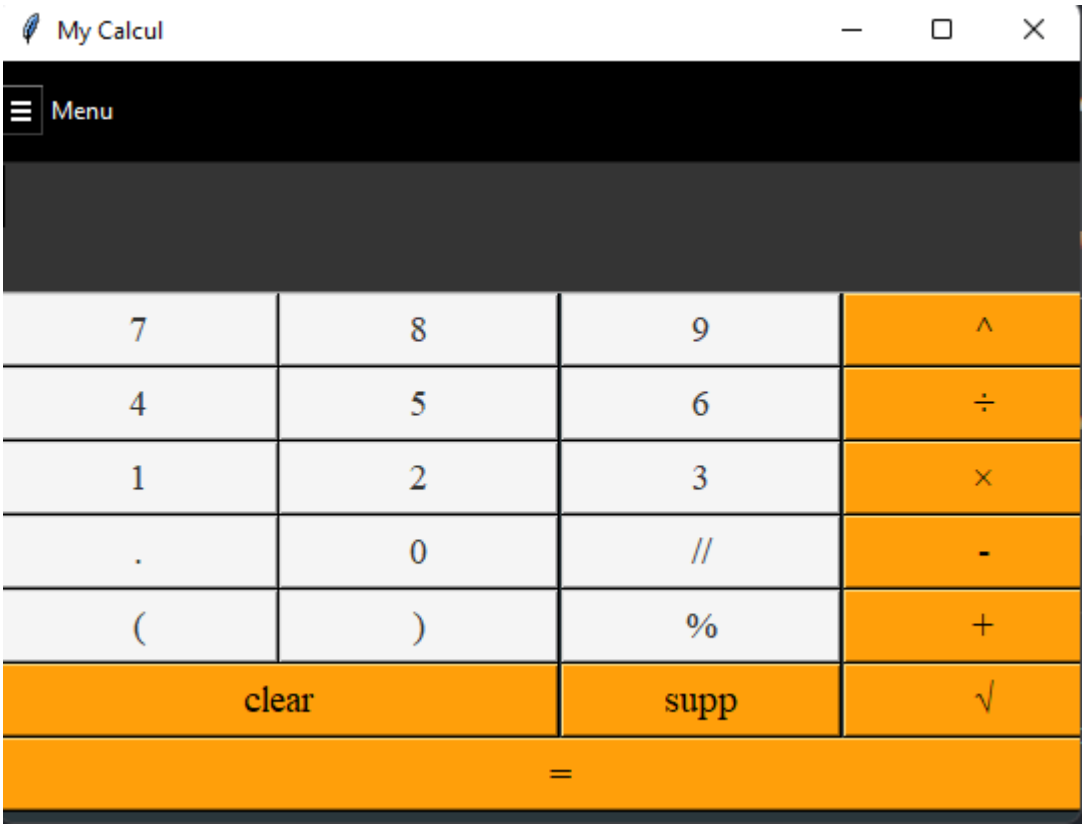
- **Les opérations mathématiques fondamentales** : addition, division, multiplication, reste de la division, soustraction, etc.
- **Les opérations logiques** : and, or, not, etc.
- **Les fonctions mathématiques** : PGCD, factorielle, cube, puissance, carré, racine carré, exponentielle, valeur absolue, logarithme, etc.
- **Les fonctions trigonométriques** : sin, cos, tan, inh, cosh, tanh, etc.
- **La conversion des nombres entre les différentes bases** : décimale, binaire, octale et hexadécimale.
- **La résolution des équations mathématiques** avec leurs représentations graphiques
- **Le calcul vectoriel et matriciel**

Pour atteindre cet objectif, nous prévoyons de développer une interface utilisateur pratique et facile à utiliser avec la bibliothèque Tkinter, comprennent un écran, des boutons numériques et des boutons de commande pour les opérations arithmétiques et aussi un menu qui va contenir 6 choix :

1. Cal simple : il s'agit d'une calculatrice permettant de faire les opérations arithmétiques de base (addition, division, multiplication, reste de la division, pourcentage, etc.)
2. Cal scientifique : il s'agit d'une calculatrice permettant de faire les opérations mathématiques avancées, telles que les calculs trigonométriques et les fonctions mathématiques (PGCD, factorielle, racine carré …)
3. Opé . logique : il s'agit d'une calculatrice permettant de faire les opérations logiques.
4. Base : il s'agit d'une calculatrice permettant La conversion des nombres entre les différentes bases informatique
5. Résolution : une calculatrice pour la résolution et la représentation graphique des équations mathématiques
6. Matrice : une calculatrice pour le calcul matriciel et aussi la résolution des systèmes linéaires
7. Interpolation : une calculatrice pour l'interpolation polynômiale
8. Intégration : une calculatrice pour calculer une valeur approchée d'un intégrale

Réalisation du projet :

Lorsqu'on ouvre la calculatrice, on se trouve automatiquement dans la fenêtre Calcul Simple



En cliquant sur le bouton Menu :

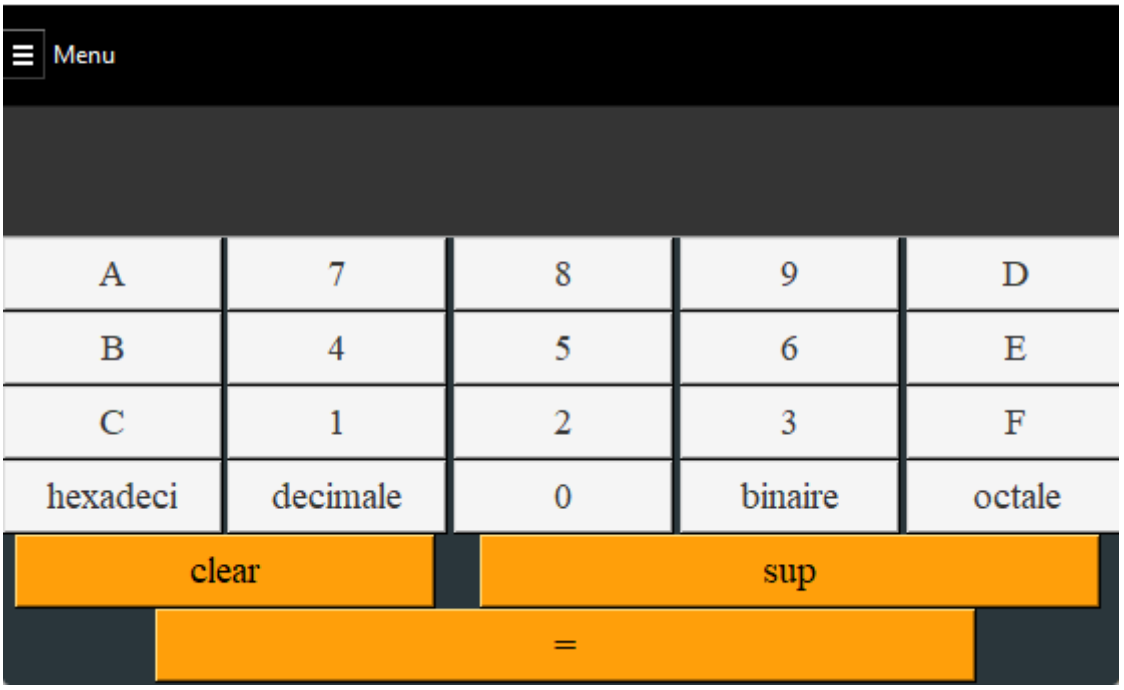
En choisissant Fonctions, des boutons de fonctions sont ajoutés à l'interface.

change	factorial	e	π
log	log10	log2	exp
abs	degrees	radians	max
min	pgcd	,	

En appuyant sur le bouton changer, d'autres fonctions viennent remplacer celles qui étaient présentes auparavant

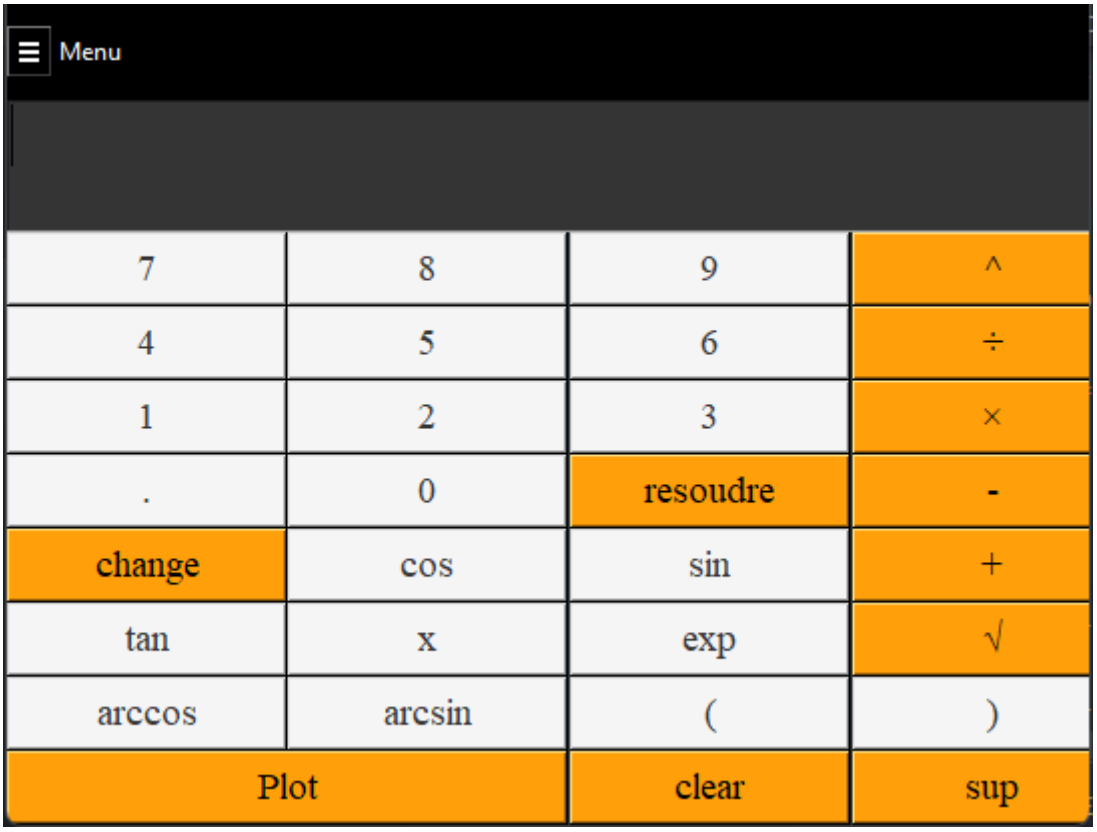
change	cos	sin	tan
arccos	arcsin	arctan	cosh
sinh	tanh	acosh	asinh
atanh	round	,	

Nous avons choisi l'interface suivante pour les conversions entre les bases numériques.



Voici l’interface pour les opérations logiques :

Voici l'interface pour la résolutions des équations :

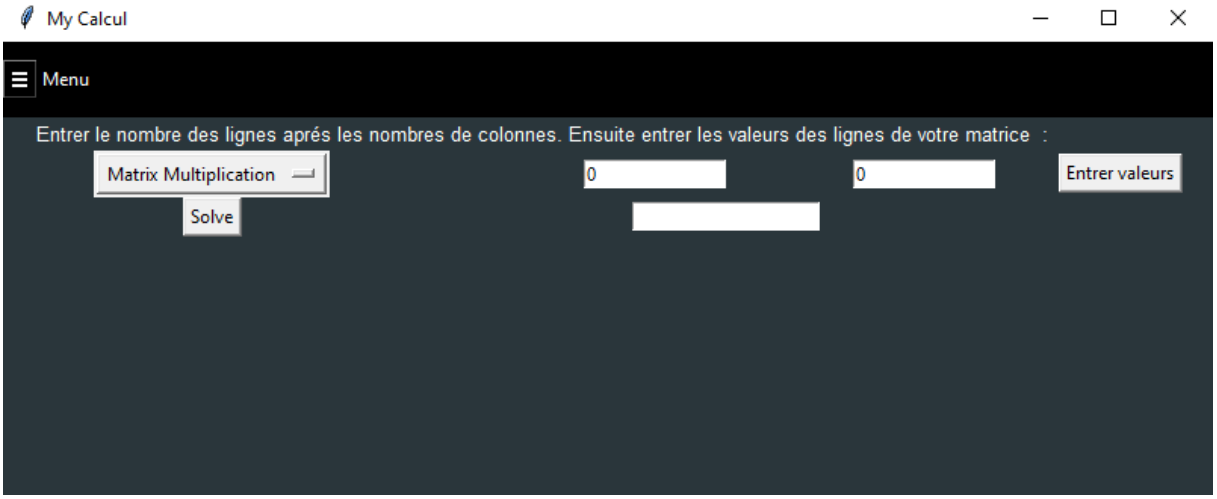


En appuyant sur le bouton plot, un graphe est affiché

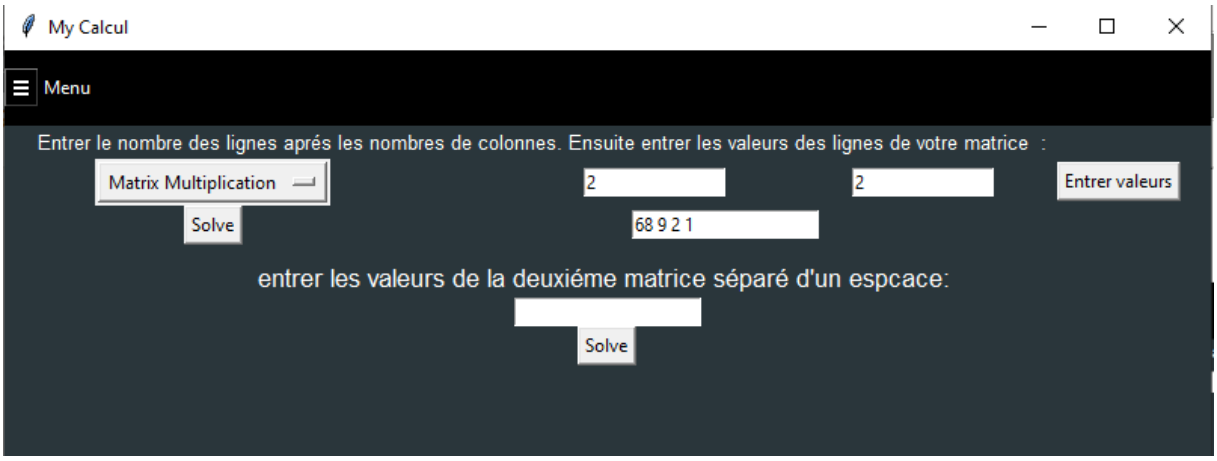
Voici l'interface pour le calcul matriciel :

Les deux premiers Entry sont pour entrer le nombre des lignes et le nombre des colonnes.

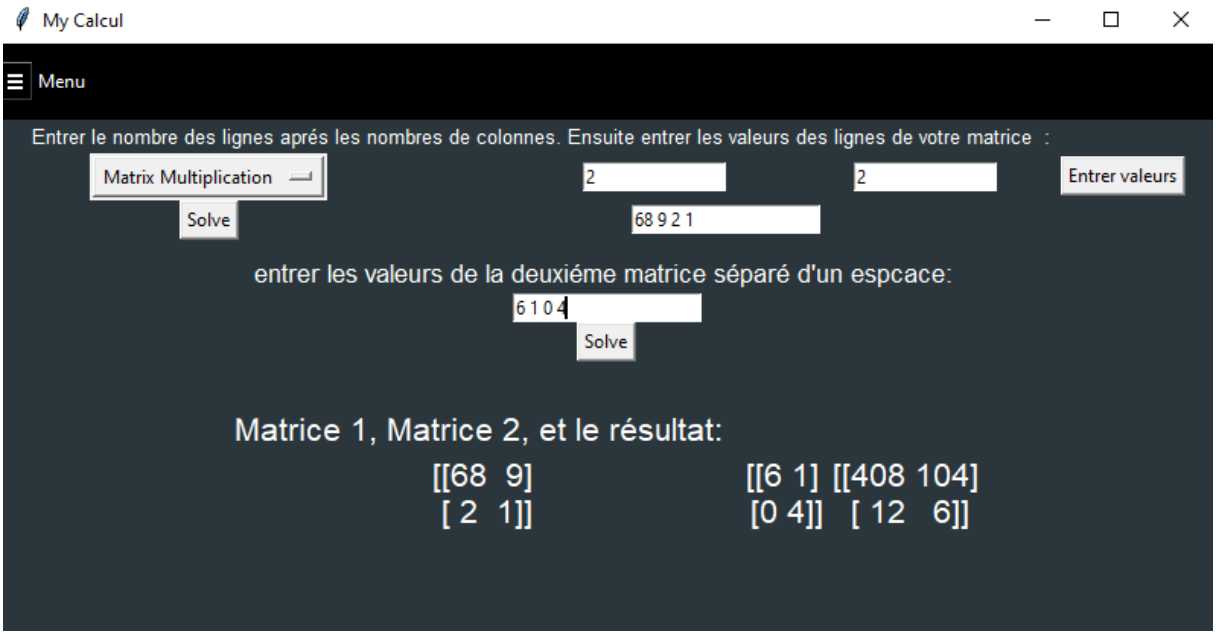
Et pour entrer les valeurs de la matrice il suffit de cliquer sur « entrer valeurs ».



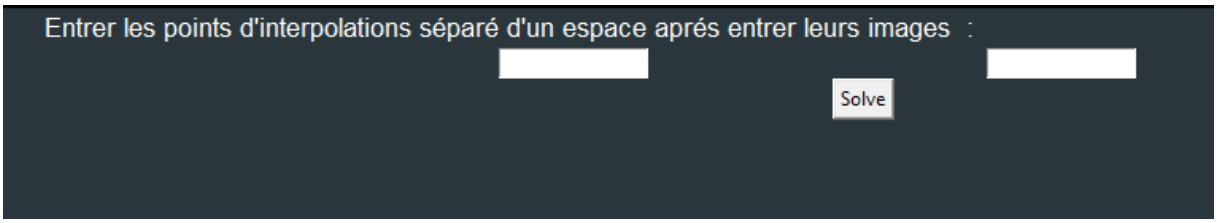
Si vous voulez entrer la deuxième matrice, cliquez sur Solve :



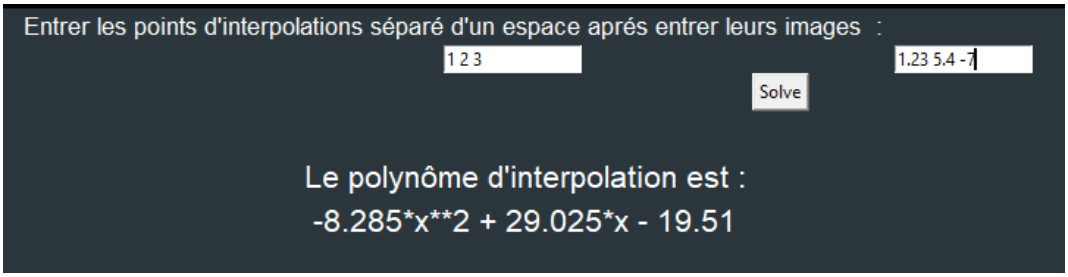
Et pour afficher le résultat, cliquez sur Solve :



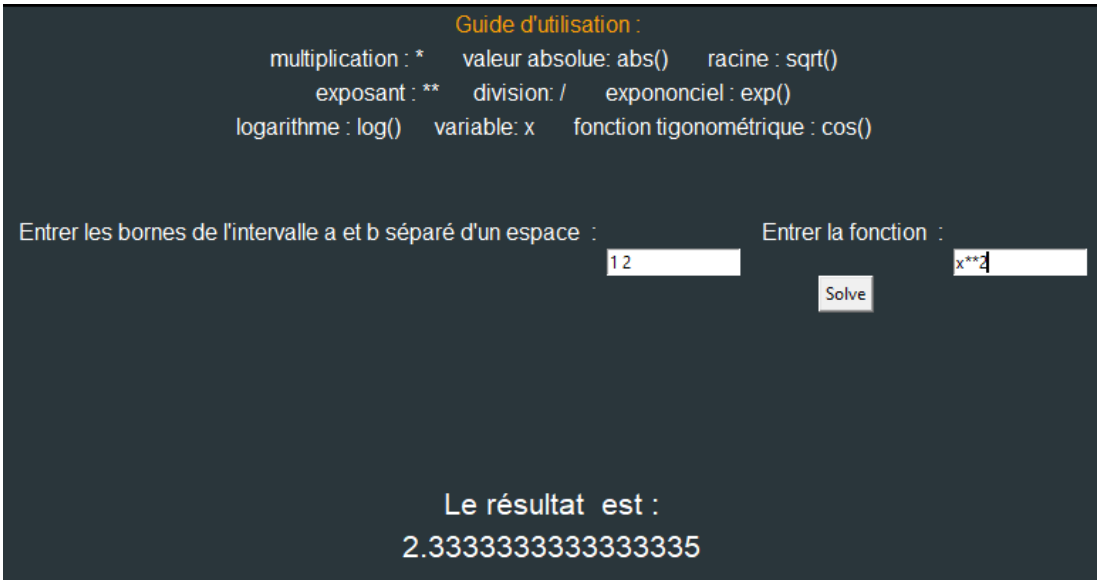
Voici l'interface pour l'interpolation :



Exemple :



Voici l’interface pour l’intégration numérique avec exemple :



Le code

Les bibliothèques importées :

```
from sympy import *
import tkinter as tk
from math import *
from numpy import *
from matplotlib.backends.backend_tkagg import (FigureCanvasTkAgg)
import matplotlib.pyplot as plt
import scipy
from scipy.integrate import quad
```

Le module sympy : sympy est une bibliothèque Python pour les mathématiques symboliques. Elle permet de manipuler et de résoudre des équations et des problèmes mathématiques.

Le module math : une bibliothèque standard de Python qui fournit des fonctions mathématiques de base, telles que sqrt (racine carrée), sin (sinus), cos (cosinus) et tan (tangente).

Le module numpy : une bibliothèque Python pour le calcul scientifique. Elle fournit des outils efficaces pour travailler avec des tableaux et des matrices de données numériques.

FigureCanvasTkAgg du module matplotlib : une classe de la bibliothèque matplotlib qui permet de créer un widget graphique Tkinter à partir d'un objet Figure de matplotlib. Ce widget peut être utilisé pour afficher des graphiques dans une fenêtre Tkinter ou dans un autre widget Tkinter.

Le module matplotlib.pyplot : est un module de la bibliothèque matplotlib qui fournit une interface de haut niveau pour créer des graphiques en Python. Il permet de tracer des lignes, des barres, des histogrammes, des diagrammes en secteurs, et bien d'autres types de graphiques en utilisant des commandes simples.

Le module scipy : est un module Python pour le calcul scientifique. Il fournit un large éventail d'outils avancés pour les tâches courantes en science des données, telles que l'interpolation, l'optimisation, le traitement du signal, et l'analyse de données.

Quad du module scipy.integrate: Cette fonction effectue une intégrale définie d'une fonction à partir d'une limite inférieure donnée jusqu'à une limite supérieure.

```
9   root = tk.Tk()
10  root.title('MyCalcul')
11  frame = tk.Frame(root, bg='#2A363B', height=440, width=700)
12  frame.pack(expand=True, fill=tk.BOTH, side=tk.BOTTOM)
13  interface_menu = tk.Frame(root, bg='black')
14  interface_menu.place(x=0, y=40, height=500, width=200)
```

Pour créer une fenêtre principale avec un titre 'MyCalcul', ainsi qu'un cadre (frame) et un menu (interface_menu) est un frame réservé pour les choix du menu.

La ligne 9 : root = tk.Tk() crée une nouvelle fenêtre principale.

La ligne 10 : **root.title('MyCalcul')** définit le titre de la fenêtre comme 'MyCalcul'.

La ligne 11 : frame = tk.Frame(root, bg='#2A363B', height=440, width=700) crée un nouveau cadre (frame) dans la fenêtre principale root. Le cadre a une hauteur de 440 pixels et une largeur de 700 pixels, et un fond de couleur '#2A363B'.

La ligne 12 : frame.pack(expand=True, fill=tk.BOTH, side=tk.BOTTOM) place le cadre en bas de la fenêtre et remplit tout **l'espace** disponible.

La ligne 13 : interface_menu = tk.Frame(root, bg='black') crée un nouveau cadre (frame) dans la fenêtre principale root. Ce cadre a un fond de couleur 'black'.

La ligne 14 : interface_menu.place(x=0, y=40, height=500, width=200) place le cadre en haut à gauche de la fenêtre, à 40 pixels de distance du haut de la fenêtre. Le cadre a une hauteur de 500 pixels et une largeur de 200 pixels.

```
16  field_text = ""
17  result = ''
18  field_coptext = ""
19  bol = True
```

Field_text, **result** et **field_coptext** sont des variables de type **str** utilisés pour stocker et afficher les différentes commandes.

bol est une variable de type **bool** utilisé par le bouton **change**

Fonction supprimer (field) :

```
22  def supprimer(field):
23      global field_text
24      global field_coptext
25      field_text = field_text.rstrip(field_text[-1])
26      field_coptext = field_coptext.rstrip(field_coptext[-1])
27      field.delete("1.0", "end")
28      field.insert("1.0", field_coptext)
```

Elle prend un paramètre **field** (un champ de texte). La fonction utilise les deux variables globales **field_text** et **field_coptext**.

Ligne 25 et 26 : la fonction commence par utiliser la méthode **rstrip ()** pour supprimer le dernier caractère de la chaîne de caractères **field_text** et **field_coptext.**

Ligne 27 : la fonction utilise la méthode **delete()** pour supprimer tout le texte contenu dans le champ.

Ligne 28 : la fonction utilise la méthode **insert()** de l'objet **field** pour insérer la chaîne de caractères **field_coptext** dans le champ.

Fonction clear () :

```
31  def clear(field):
32      global field_text
33      global field_coptext
34      field_coptext = ""
35      field_text = ""
36      field.delete("1.0", "end")
```

La fonction **supprimer ()** qui prend un paramètre **field** (un champ de texte). La fonction utilise les deux variables globales **field_text** et **field_coptext**.

Cette fonction permet de vider le champ **field** et les deux variables **field_text** et **field_coptext.**

Fonction calculate () :

Elle essaie d'exécuter l'expression arithmétique contenue dans la variable "field_text" en utilisant la fonction "eval" de Python et stocke le résultat dans la variable "result". Si une exception est levée lors de l'exécution de cette expression, la chaîne "Error" est stockée dans la variable "result" à la place.

```
77  def calculate(field):
78      global result
79      global field_text
80      global field_coptext
81      try:
82          result = str(eval(field_text))
83          field_text = ""
84
85      except Exception as e:
86          result = "Error"
87      finally:
88          field.delete("1.0", "end")
89          field.insert(tk.INSERT, field_coptext+'\n')
90          field.insert(tk.INSERT, result)
91          field_text = result
```

Enfin, pour afficher le résultat à l'écran, Elle supprime tout le contenu du champ (grâce à la méthode "delete"), insère d'abord "field_coptext" qui contient toute l'expression mathématique que l'utilisateur a saisi, suivi d'un saut de ligne (grâce à la méthode "insert"), puis insère le contenu de la variable "result" qui est le résultat final (encore une fois en utilisant la méthode "insert"). Enfin, la variable "field_text" est mise à jour en prenant la valeur de "result".

Fonction add_to_field (ajouter, field):

Cette fonction ajoute du texte à une zone de saisie spécifiée (field). La fonction prend en entrée une chaîne de caractères à ajouter (ajouter) et la zone de saisie à laquelle ajouter cette chaîne (field). La fonction utilise des variables globales pour stocker le contenu de la zone de saisie (field_text) et une version modifiée de ce contenu pour l'affichage (field_coptext). Elle utilise des remplacements de chaîne pour remplacer certains termes spécifiques (comme "sqrt") avec des symboles appropriés (comme "√") avant d'ajouter le texte à la

zone de saisie. Elle utilise également des boucles pour parcourir des listes de termes spécifiques et remplacer les opérateurs mathématiques couramment utilisés avec des symboles appropriés. Enfin, elle efface le contenu de la zone de saisie et y insère la version modifiée du contenu (field_coptext).

Fonction convert (field) :

```
130 > def convert(field):···
```

La fonction convert, qui prend en paramètre la variable **field**. La fonction utilise également les deux variables globales nommées **field_text** et **result**.

On va prendre le cas de la conversion de la base binaire vers les autres bases,

Et le même raisonnement est appliqué sur les autres conversions.

La fonction vérifie si la chaîne de caractères contenue dans **field_text** commence par **binaire**. Si c'est le cas, elle enlève "binaire" du début de la chaîne de caractères et la stocke de nouveau dans **field_text**.

Ensuite, la fonction vérifie si **field_text** se termine par "decimale", "octale", "hexadeci" ou "binaire". Selon le cas, elle enlève le suffixe correspondant de **field_text** et effectue une opération spécifique. Si **field_text** se termine par "decimale", la fonction évalue la chaîne de caractères comme une expression binaire et stocke le résultat en décimal dans **result**. Si **field_text** se termine par "octale", la fonction évalue la chaîne de caractères comme une expression binaire, la convertit en octal et stocke le résultat dans **result**. Si **field_text** se termine par "hexadeci", la fonction évalue la chaîne de caractères comme une expression binaire, la convertit en hexadécimal et stocke le résultat dans **result**. Si **field_text** se termine par "binaire", la fonction stocke simplement la chaîne de caractères **field_text** dans **result**.

Si aucun de ces cas n'est vrai, la fonction stocke **ERROR** dans **result.**

Fonction create_dictio_buttons () :

```
344 def create_dictio_buttons(field, dictio, wid):
345     global frame
346     for c, v in dictio.items():
347         btn = tk.Button(frame, text=c, fg="#343434", bg='#f5f5f5',  width=wid, font=(
348             "Times New Roman", 14), command=lambda x=c:  add_to_field(x, field))
349         if (c == "gcd"):
350             btn['text'] = "pgcd"
351         elif (c == "pi"):
352             btn['text'] = "\u03C0"
353         btn.grid(row=v[0], column=v[1])
```

Cette fonction prend en argument trois valeurs : **field, dictio**, et **wid**. La fonction crée une série de boutons avec le texte de chaque clé dans **dictio** et les place dans une grille de disposition définie par la valeur de chaque clé dans **dictio**. Les boutons exécuteront la fonction **add_to_field** lorsqu'on clique dessus, en passant en argument la clé du bouton ainsi que l'argument **field**. Les attributs **fg** et **bg** définissent respectivement les couleurs de texte et de fond du bouton. L'attribut font définit la famille de police et la taille du texte du bouton. L'attribut **width** définit la largeur du bouton.

Si la clé est égale à "gcd", le texte du bouton sera défini sur "pgcd". Si la clé est égale à "pi", le texte du bouton sera défini sur la lettre grecque pi (\u03C0).

Exemple des dictionnaires qu'on a utilisé :

```
92  hexadeci = {'A': (2, 1), 'B': (3, 1), 'C': (
93      4, 1), 'D': (2, 5), 'E': (3, 5), 'F': (4, 5)}
94
95  numbers_base_op = {'7': (2, 2), '8': (2, 3), '9': (2, 4), '4': (3, 2), '5': (3, 3), '6': (
96      3, 4), '1': (4, 2), '2': (4, 3), '3': (4, 4),  '0': (5, 3)}
97
98  matrice = {'7': (6, 1), '8': (6, 2), '9': (6, 3), '4': (7, 1), '5': (7, 2), '6': (
99      7, 3), '1': (8, 1), '2': (8, 2), '3': (8, 3),  '0': (9, 2), '.': (9, 1)}
```

Fonction create_buttons_operations (field):

```
356  def create_buttons_operations(field):
357      i = 2
358      global frame
359      for operator, symbol in operations.items():
360          button = tk.Button(frame, text=symbol, bg="#FF9F0A", width=13,  font=(
361              "Times New Roman", 14),
362              command=lambda x=operator: add_opperators(x, field))
363          button.grid(row=i, column=4)
364          i += 1
```

Le dictionnaire operations :

```
421  operations = {"**": "^",  "/": "\u00F7", "*": "\u00D7", "-": "-",
422                "+": "+", "sqrt": "√"}
```

La fonction prend en argument une seule valeur **field**. La fonction crée une série de boutons avec le texte de chaque symbole dans le dictionnaire **operations** et les place dans une grille de disposition. Les boutons exécuteront la fonction **add_operators** lorsqu'on clique dessus, en passant en argument la clé du symbole ainsi que l'argument **field**. L'attribut **bg** définit la couleur de

fond du bouton. L'attribut font définit la famille de police et la taille du texte du bouton. L'attribut **width** définit la largeur du bouton.

La variable **i** est initialisée à **2** et est utilisée pour définir la position de la ligne de chaque bouton dans la grille de disposition. La boucle parcourt chaque clé et valeur dans le dictionnaire **operations**, créant un bouton pour chaque itération avec le texte défini sur la valeur (symbole) et l'attribut command défini sur la fonction **add_operators**. L'attribut **row** du bouton est défini sur la valeur de i et l'attribut **column** est défini sur 4 pour chaque itération. La valeur de i est incrémentée de 1 à la fin de chaque itération.

Fonction create_button () :

```
367   def create_button(field, l, c, cs, wid, fun, signe):
368       global frame
369       button_equal = tk.Button(frame, text=signe, bg="#FF9F0A", width=wid, font=(
370           "Times New Roman", 14),
371           command=lambda: fun(field))
372       button_equal.grid(row=l, column=c, columnspan=cs)
```

La fonction prend en argument six valeurs **: field, l, c, cs, wid, fun**, et **signe**. La fonction crée un bouton avec le texte défini sur la valeur signe et le place dans une grille de disposition. Le bouton exécutera la fonction **fun** lorsqu'on clique dessus, en passant en argument la valeur **field**. L'attribut **bg** définit la couleur de fond du bouton. L'attribut **font** définit la famille de police et la taille du texte du bouton. L'attribut **width** définit la largeur du bouton.

L'attribut **row** du bouton est défini sur la valeur de **l** et l'attribut **column** est défini sur **c**. L'attribut **columnspan** est défini sur **cs**.

Fonction change () :

```
383   def changer(field, fonction, fonction_1, wid):
384       global bol
385       global frame
386       if bol == True:
387           create_dictio_buttons(field, fonction, wid)
388       else:
389           create_dictio_buttons(field, fonction_1, wid)
390       bol = not bol
```

Cette fonction prend en argument quatre valeurs : **field, fonction, fonction_1,** et **wid**. La fonction vérifie la valeur de la variable globale **bol**. Si bol est **True**, la fonction appelle la fonction create_dictio_buttons avec les arguments field et

fonction et leur passe la valeur de wid. Si bol est **False**, la fonction appelle la fonction create_dictio_buttons avec les arguments field et **fonction_1** et leur passe la valeur de wid.

Ensuite, la valeur de bol est modifiée en son contraire (True devient False et False devient True).

Fonction resoudreequa () :

```
393    def resoudreequa(field):
394        global field_text
395        global field_coptext
396        for operator, symbol in resoudre_trigo.items():
397            field_coptext = field_coptext.replace(operator, f'{symbol}')
398        x = Symbol('x')
399        equation = sympify(field_coptext)
400        solutions = str(solve(equation))
401        field.delete("1.0", "end")
402        field.insert("1.0", solutions)
```

```
549    resoudre_trigo = {'arccos': 'acos', 'arcsin': 'asin', 'arctan': 'atan',
550                      'arccosh': 'acosh', 'arcsinh': 'asinh', 'arctanh': 'atanh', '√': 'sqrt'}
```

La fonction prend en argument une seule valeur **field**. La fonction utilise la variable globale **field_coptext** et remplace chaque clé dans le dictionnaire **resoudre_trigo** (car le module sympy utilise les fonctions arccos, arcsin comme acos, asin …) par sa valeur correspondante. Ensuite, la fonction crée un objet Symbol nommé x et utilise la fonction **sympify** pour transformer la chaîne de caractères **field_coptext** en un objet sympy. La fonction solve est utilisée pour résoudre l'équation représentée par cet objet sympy.

La fonction **solve** renvoie un objet contenant les solutions de l'équation, que la fonction transforme en chaîne de caractères grâce à la fonction **str.** Le contenu actuel du champ de texte (**field**) est effacé grâce à la méthode **delete**, puis la chaîne de caractères des solutions est insérée au début du champ de texte grâce à la méthode **insert**.

Fonction plot () :

Cette fonction permet de dessiner les graphes. La fonction crée ensuite un ensemble de valeurs x en utilisant la fonction linspace de NumPy. Elle crée 1000 points espacés également entre -10 et 10. La fonction crée alors un ensemble de valeurs y en utilisant la fonction eval. Cette fonction évalue la

chaîne de caractères stockée dans field_text comme s'il s'agissait d'une expression Python et renvoie le résultat.

La fonction utilise alors Matplotlib pour créer un objet figure et un objet axe, qu'elle stocke dans les variables fig et ax. Elle trace alors les valeurs y contre les valeurs x en utilisant la méthode plot de l'objet axe. Elle ajoute des étiquettes aux axes x et y en utilisant les méthodes set_xlabel et set_ylabel.

La fonction crée ensuite un objet FigureCanvasTkAgg, qui est un widget Tkinter qui affiche une figure Matplotlib. La figure est passée à l'objet FigureCanvasTkAgg en tant qu'argument et l'objet est créé à l'intérieur du widget frame. La méthode get_tk_widget est alors utilisée pour récupérer le widget Tkinter associé à l'objet FigureCanvasTkAgg et il est placé à l'emplacement spécifié en utilisant la méthode place.

Enfin, la méthode root.geometry est utilisée pour définir la taille de la fenêtre principale Tkinter.

```
def plot():
    global field_text
    global frame
    x = linspace(-10, 10, 1000)
    y = eval(field_text)
    fig, ax = plt.subplots()
    ax.plot(x, y)
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    canvas = FigureCanvasTkAgg(fig,master=frame)

    canvas.get_tk_widget().grid()
    canvas.get_tk_widget().place(x=600,y=0)
    root.geometry("1200x700")
```

Fonction delete_pages () :

```
16  def delete_pages():
17      global frame
18      for i in frame.winfo_children():
19          i.destroy()
```

Puisque notre calculatrice est formée d'un menu de plusieurs options et on doit switcher entre les différentes options, c'est pour cela qu'on a utilisé cette fonction afin de réinitialiser le contenu du frame.

Fonction cal_simple () :

```
def calsimple_page():

    delete_pages()
    global interface_menu
    interface_menu.place_forget()
    global frame
    field = tk.Text(frame,height=2, width=40, font=(
        "Times New Roman", 20), fg='#f5f5f5', bg="#343434")
    field.grid(row=1, column=1, columnspan=5)

    field.focus()
    root.geometry("540x380")
    create_dictio_buttons(field, numbers, 13)
    create_buttons_operations(field)
    create_dictio_buttons(field, simpleop, 13)
    create_button(field, 8, 1, 4, 55, calculate, "=")
    create_button(field, 7, 1, 2, 27, clear, "clear")
    create_button(field, 7, 3, 1, 13, supprimer, "supp")
```

La fonction commence par utiliser la fonction "delete_pages" pour détruire tous les widgets enfants de la variable "frame". Ensuite, elle utilise la méthode "**place_forget**" de tkinter sur "interface_menu" pour masquer le menu.

La fonction crée ensuite un widget de champ de texte en utilisant tkinter et le place dans "frame" en utilisant la méthode "**grid**". Elle change également la taille de la fenêtre racine en appelant la méthode "**geometry**" de tkinter.

Ensuite, la fonction appelle plusieurs autres fonctions pour créer des widgets de bouton et les placer dans l'interface utilisateur. Ces fonctions sont "**create_dictio_buttons**", "**create_buttons_operations**", "**create_dictio_buttons**", "**create_button**". Ces fonctions créent des boutons en utilisant tkinter et leur assignent des commandes à exécuter lorsqu'ils sont cliqués

Fonction fonction_page() :

La fonction commence par appeler la fonction appelée delete_pages afin de réinitialise le frame, puis elle utilise la méthode place_forget sur interface_menu pour masquer l'interface de menu.

La fonction déclare également une variable globale appelée bol, puis crée un widget de type Text et le place dans frame à une position de grille spécifique.

Cette fonction définit également la taille de la fenêtre et appelle plusieurs autres fonctions pour créer les boutons et les placer dans frame.

La fonction crée également un bouton appelé **changer_btn** et le place dans frame à une position de grille spécifique, et assigne une fonction lambda en tant qu'attribut commande au bouton. La fonction termine en inversant la valeur de bol

```
def fonction_page():
    delete_pages()
    global frame
    global interface_menu
    interface_menu.place_forget()
    global bol
    field = tk.Text(frame, height=2, width=40, font=(
        "Times New Roman", 20), fg='#f5f5f5', bg="#343434")
    field.grid(row=1, column=1, columnspan=5)
    field.focus()

    root.geometry("540x500")
    create_dictio_buttons(field, numbers, 13)
    create_buttons_operations( field)
    create_dictio_buttons(field, simpleop, 13)
    create_button(field, 8, 1, 4, 55, calculate, "=")
    create_button(field, 7, 1, 2, 27, clear, "clear")
    create_dictio_buttons(field, fonction, 13)
    bol = not bol
    changer_btn = tk.Button(frame, text='change',
                            font=('Bold', 15), fg='black', bg='#FF9F0A', width=12, command=lambda: changer( field, fonction, fonction_1, 13))
    changer_btn.grid(row=9, column=1)
    create_button(field, 7, 3, 1, 13, supprimer, "sup")
```

Fonction base_page() :

```
def base_page():
    delete_pages()

    global frame
    global interface_menu
    interface_menu.place_forget()

    field = tk.Text(frame, height=2, width=40, font=(
        "Times New Roman", 20), fg='#f5f5f5', bg="#343434")
    field.grid(row=1, column=1, columnspan=5)
    field.focus()
    root.geometry("540x340")
    create_dictio_buttons(field, numbers_base_op, 10)
    create_dictio_buttons(field, hexadeci, 10)
    create_dictio_buttons( field, base, 10)
    create_button(field, 6, 3, 3, 30, supprimer, "sup")
    create_button(field, 6, 1, 2, 20, clear, "clear")
    create_button(field, 7, 1, 5, 40, convert, "=")
```

La variable field est un widget Text qui est placé dans le widget frame. La fonction appelle alors plusieurs autres fonctions pour créer des boutons par appelée la fonction create_dictio_buttons qui prend comme paramètre un dictionnaire dont les clés sont les noms des boutons et les valeurs sont des

tuples composés des coordonnées des lignes et des colonnes. La fonction create_button est appelée pour créer un bouton "**sup**" pour appeler la fonction supprimer qui efface le dernier caractère saisi, un bouton "**clear**" pour appeler la fonction clear qui efface totalement l'écran et un bouton "**=**" pour appeler la fonction convert.

Fonction logique_page () :

```
def logique_page():
    delete_pages()
    global frame
    global interface_menu
    interface_menu.place_forget()
    root.geometry("540x350")

    field = tk.Text(frame, height=2, width=40, font=(
        "Times New Roman", 20), fg='#f5f5f5', bg="#343434")
    field.grid(row=1, column=1, columnspan=5)
    field.focus()

    create_dictio_buttons(field, numbers_base_op, 10)
    create_button( field, 7, 1, 5, 32, calculate, "=")
    create_button(field, 6, 4, 2, 16, clear, "clear")
    create_button( field, 6, 1, 2, 16, supprimer, "sup")
    create_dictio_buttons(field, logique, 10)
```

Cette fonction permet d'effectuer les opérations logiques, elle suit le même principe que la fonction précédente.

Fonction resolution_page() :

Cette fonction permet de résoudre des équations. Lorsque la fonction est appelée, les widgets précédemment créés dans la fenêtre sont supprimés, la fenêtre est redimensionnée et un champ de texte est créé. Des boutons sont ensuite créés pour les opérations arithmétiques de base, les chiffres et certaines fonctions mathématiques. Il y a aussi un bouton qui permet de changer de fonctions mathématiques affichées. Enfin, il y a un bouton pour tracer la courbe représentant la solution de l'équation, un bouton pour effacer le contenu du champ de texte et un bouton pour résoudre l'équation.

Fonction createMatrix() :

```
def createMatrix(entries, rows, columns):
    matrix = array(entries).reshape(rows, columns)
    return matrix
```

Cette fonction a trois arguments : "**entries**" (les valeurs de la matrice), "**rows**" (nombre des lignes de la matrice) et "**columns**" (nombre des colonnes de la matrice). Elle crée une matrice en utilisant la bibliothèque "**array**" de Python en utilisant les "entries" et en spécifiant le nombre de "**rows**" et de "**columns**" souhaité. La fonction retourne ensuite cette matrice.

Fonction PDP() :

```
def PDP(matrix):
    eVals, eVecs = linalg.eig(matrix)
    invVecs = linalg.inv(eVecs)
    D = diag(eVals)
    D = D.round(5).real
    P = eVecs.round(5).real
    Pinv = invVecs.round(5).real
    return P, D, Pinv
```

Cette fonction prend en entrée une matrice "**matrix**". La fonction calcule les valeurs propres et vecteurs propres de cette matrice en utilisant la bibliothèque "**linalg**" de Python. Elle calcule également l'inverse des vecteurs propres. Ensuite, la fonction crée une matrice diagonale "D" en utilisant les valeurs propres. La fonction arrondit ensuite les valeurs des matrices "**D**", "**P**" (vecteurs propres) et "**Pinv**" (inverse des vecteurs propres) à 5 décimales et ne garde que la partie réelle. Enfin, la fonction retourne les matrices **"P**", "**D**" et "**Pinv**".

Fonction LU() :

```
def LU(matrix):

    P, L, U = scipy.linalg.lu(matrix)
    L = L.round(4).real
    U = U.round(4).real
    return P, L, U
```

Cette Fonction prend en entrée une matrice "matrix". La fonction utilise la bibliothèque "**scipy.linalg**" de Python pour décomposer la matrice en forme LU

(décomposition en forme LU signifie que la matrice peut être écrite comme le produit de deux matrices triangulaires inférieures et supérieures). La fonction arrondit ensuite les valeurs des matrices "**L**" (matrice triangulaire inférieure) et "**U**" (matrice triangulaire supérieure) à 4 décimales et ne garde que la partie réelle. Enfin, la fonction retourne les matrices "**P**", "**L**" et "**U**".

Fonction addMat() :

```
def addMat(m1, m2):
    m3 = m1 + m2
    return m3
```

Cette fonction prend en entrée deux matrices "m1" et "m2". La fonction calcule l'addition des deux matrices et retourne la matrice "m3".

Fonction soustrMat() :

```
def soustrMat(m1, m2):
    m3 = m1 - m2
    return m3
```

Cette fonction prend en entrée deux matrices "m1" et "m2". La fonction calcule la soustraction des deux matrices et retourne la matrice "m3".

Fonction determinat() :

```
def determinat(matrix):
    return linalg.det(matrix)
```

Elle retourne le déterminant de la matrice entré en paramètre.

Fonction inverse() :

```
def inverse(matrix):
    return linalg.inv(matrix)
```

Elle retourne l'inverse de la matrice entré en paramètre.

Fonction mul() :

```
def mul(m1, m2):
    m3 = matmul(m1, m2)
    m3 = m3.round(5).real
    return m3
```

Cette prend en entrée deux matrices "**m1**" et "**m2**". La fonction calcule le produit des deux matrices en utilisant la fonction "**matmul**" de Python. La fonction arrondit ensuite les valeurs de la matrice "**m3**" résultante à 5 décimales et ne garde que la partie réelle. Enfin, la fonction retourne la matrice "**m3**".

Fonction QR() :

```
def QR(matrix):
    q, r = scipy.linalg.qr(matrix)

    q = q.round(4).real
    r = r.round(4).real
    return q, r
```

Cette fonction prend en entrée une matrice "**matrix**". La fonction utilise la bibliothèque "**scipy.linalg**" de Python pour décomposer la matrice en forme QR. La fonction arrondit ensuite les valeurs des matrices "**q**" (matrice orthogonale) et "r" (matrice triangulaire supérieure) à 4 décimales et ne garde que la partie réelle. Enfin, la fonction retourne les matrices "**q**" et "**r**".

Fonction chol () :

```
def chol(matrix):
    L = scipy.linalg.cholesky(matrix, lower=True)
    LT = transpose(L)
    return L, LT
```

Cette fonction prend en entrée une matrice "**matrix**". La fonction utilise la bibliothèque "**scipy.linalg**" de Python pour calculer la décomposition de Cholesky de la matrice. La décomposition de Cholesky d'une matrice est une décomposition de cette matrice en forme A = LL*, où L est une matrice triangulaire inférieure et L* sa transposée. La fonction utilise également la fonction "transpose" de Python pour calculer la transposé de L.

Fonction update() :

```
def update(Matrix_fram):
    global answ
    Rows = testRows.get()
    Cols = testCols.get()

    answ = tk.Entry(Matrix_fram, text="")
    answ.grid(column=1, row=2, columnspan=3)
```

Cette fonction prend en entrée une "**Matrix_fram**". La fonction déclare une variable globale "**answ**" qui est une entrée de texte dans un widget Tkinter. La fonction définit également deux variables locales "**Rows**" et "**Cols**" qui sont respectivement égales aux valeurs des widgets "**testRows**" et "**testCols**". La fonction place finalement la variable "**answ**" dans la grille du widget "**Matrix_fram**" en lui attribuant une colonne et une ligne de départ, ainsi qu'un nombre de colonnes à étendre.

Fonction calculatematrice() :

Cette fonction prend en entrée un "**Matrix_fram**". La fonction calcule la décomposition de Cholesky de la matrice "**Mat1**" en appelant la fonction "**chol**" définie précédemment. Ensuite, la fonction utilise une boucle pour détruire tous les widgets enfants de "**Matrix2_frame**" et de "**Matrix_result_frame**".

Ensuite, la fonction vérifie si la variable "**selection**" est égale à "**Matrix Addition**", "**Matrix Multiplication**" ou "**Matrix Soustraction**". Si c'est le cas, la fonction effectue les actions suivantes :

- Utilise une boucle pour détruire tous les widgets enfants de "**Matrix2_frame**" et de "**Matrix_result_frame**"
- Crée un widget "I**nfo**" qui est un label avec un texte prédéfini et qui est ajouté à "**Matrix2_frame**"
- Crée un widget "**answ1**" qui est une entrée de texte et qui est ajouté à "**Matrix2_frame**" pour entrer les valeurs de la deuxième matrice.
- Crée un widget qui est un bouton avec un texte prédéfini et qui appelle la fonction "**Solve**" lorsqu'il est cliqué. Le widget est ajouté à "**Matrix2_frame**"

Ensuite, la fonction vérifie si la variable "**selection**" est égale à "**Cholesky Decomposition**". Si c'est le cas, la fonction effectue les actions suivantes :

- Utilise une boucle pour détruire tous les widgets enfants de "**Matrix2_frame**" et de "**Matrix_result_frame**"
- Crée un widget "result" qui est un label avec un texte prédéfini et qui est ajouté à "**Matrix2_frame**"
- Crée un widget **"resultLT**" qui est un label avec le contenu de la matrice "**LT**" et qui est ajouté à "**Matrix2_frame**"
- Crée un widget "**resultL**" qui est un label avec le contenu de la matrice "L" et qui est ajouté à "Matrix2_frame"

Et c’est le même principe pour les autres conditions.

Fonction Solve () :

Cette fonction prend en entrée les matrices "**Mat1**", "**Matrix_result_frame**" et "**Matrix2_frame**", ainsi que la variable "**selection**". La fonction déclare deux variables locales "**rows**" et "**columns**" qui sont respectivement égales aux valeurs des widgets "**testRows**" et "**testCols**". La fonction lit également les valeurs entrées dans le widget "**answ1**" pour récupérer la matrice.

Ensuite, la fonction crée une matrice "**Mat3**" en utilisant les valeurs lues et la fonction "**createMatrix**" définie précédemment. La fonction vérifie ensuite la valeur de la variable "**selection**" et effectue une opération sur les matrices "**Mat1**" et "**Mat3**" en conséquence.

Enfin, la fonction utilise une boucle pour détruire tous les widgets enfants de "**Matrix_result_frame**", puis crée plusieurs widgets "**result**" qui sont des labels avec les contenus des matrices "**Mat1**", "**Mat3**" et "**Mat4**" et qui sont ajoutés à "**Matrix_result_frame**".

Fonction matrice_page () :

Cette fonction "**matrice_page**" qui affiche une page d'interface graphique utilisateur contenant différents widgets et frames.

La fonction commence par cacher l'interface de menu et définit plusieurs variables globales "**frame**", "**Matrix1_frame**" (entrer la matrice 1), "**Matrix2_frame**" (entrer la matrice 2), et "**Matrix_result_frame**" (afficher le résultat). Ensuite, la fonction définit la taille de la fenêtre principale et ajoute les frames "**Matrix1_frame**", "**Matrix2_frame**" et "**Matrix_result_frame**" à "**frame**".

La fonction ajoute ensuite un label "**Instructions**" avec des instructions pour l'utilisateur. Elle crée également deux widgets de type "**Entry**" qui permettent à l'utilisateur de saisir les dimensions de la matrice qu'il souhaite créer. Un bouton "**updatE**" est également créé et appellera la fonction "**update**" lorsqu'il est cliqué.

La fonction crée une liste de chaînes de caractères "**Options_List**" contenant les différentes options d'opérations de matrices disponibles. Elle définit également une variable "**selection**" de type "**StringVar**" et lui assigne la valeur de la première chaîne de caractères de "**Options_List**". Un widget "**drop_down**" de type "**OptionMenu**" est ensuite créé et affichera les options

de "**Options_List**". Un bouton "**calc**" est également créé et appellera la fonction "**calculatematrice**" lorsqu'il est cliqué.

Lagrange_polynomial() :

```
def lagrange_polynomial(x_points, y_points):
    x = Symbol('x')
    lagrange_poly = 0
    for i in range(len(x_points)):
        xi = x_points[i]
        yi = y_points[i]
        term = yi
        for j in range(len(x_points)):
            if i == j:
                continue
            xj = x_points[j]
            yj = y_points[j]
            term *= (x - xj) / (xi - xj)
        lagrange_poly += term

    return simplify(lagrange_poly)
```

Cette fonction sert à générer le polynôme de Lagrange, elle prend comme paramètres : un ensemble de points (x_points) avec leurs images (y_points) et renvoie un polynôme qui passe par tous ces points.

Calcul_polynome() :

La fonction vérifie d'abord que les entrées 'x' et 'y' sont valides en les convertissant en listes de nombres réels, puis vérifie que ces listes ont la même longueur. Si ces conditions ne sont pas remplies, des messages d'erreur sont affichés et la fonction ne calcule pas de polynôme. Si les entrées sont valides, la fonction utilise la fonction lagrange_polynomial pour calculer le polynôme d'interpolation, puis affiche le résultat final dans une zone de texte.

Calcul_integral() :

Cette fonction calcule l'intégrale d'une fonction donnée sur un intervalle spécifié. La fonction prend en entrée une zone de saisie pour l'intervalle de calcul (interval), une zone de saisie pour la fonction (f) et un cadre d'affichage pour les résultats (frame3). Elle utilise la bibliothèque SymPy pour définir une variable symbolique x, convertir la fonction en entrée en une expression

mathématique utilisable, et utiliser la fonction quad pour calculer l'intégrale de cette fonction sur l'intervalle spécifié. Si les entrées ne sont pas valides, des messages d'erreur sont affichés. Si les entrées sont valides, la fonction affiche le résultat de l'intégrale dans une zone de texte.

Les deux fonctions **interpolation()** et **integration()** pour créer les pages.

Créer le menu :

```
calsimple_page()
```

Lorsqu'on exécute le code, Cette instruction permettent d'ouvrir la calculatrice sur calcul simple par défaut.

Fonction menu_selection() :

```
def menu_selection():
    global interface_menu
    interface_menu.place(x =0,y=20, height =500,width=200)
    calsimple_btn = tk.Button(interface_menu, text='Cal. Simple',
                                        font=('Bold', 15), fg='#FF9F0A', bg='black', bd=0, command=lambda:calsimple_page())
    calsimple_btn.place(x=10, y=50)
    fonction_btn = tk.Button(interface_menu, text='Fonctions',
                                       font=('Bold', 15), fg='#FF9F0A', bg='black', bd=0, command=lambda:fonction_page())
    fonction_btn.place(x=10, y=90)
    base_btn = tk.Button(interface_menu, text='Bases',
                     font=('Bold', 15), fg='#FF9F0A', bg='black', bd=0, command=lambda:  base_page())
    base_btn.place(x=10, y=130)
    logique_btn = tk.Button(interface_menu, text='Opé. Logique',
                        font=('Bold', 15), fg='#FF9F0A', bg='black', bd=0, command=lambda:  logique_page())
    logique_btn.place(x=10, y=170)
    resolution_btn = tk.Button(interface_menu, text='Resolution',
                        font=('Bold', 15), fg='#FF9F0A', bg='black', bd=0, command=lambda: resolution_page())
    resolution_btn.place(x=10, y=210)
    matrice_btn = tk.Button(interface_menu, text='Matrices',
                        font=('Bold', 15), fg='#FF9F0A', bg='black', bd=0, command=lambda:     matrice_page())
    matrice_btn.place(x=10, y=250)
```

Cette fonction permet d'ajouter les boutons (les options du menu) et les placer sur **interface_menu** . Le menu est créé sous la forme d'un widget "Frame" et est placé dans la fenêtre. Lorsqu'un bouton est cliqué, la fonction correspondante à l'interface sélectionnée est appelée. Les opérations disponibles sont "Calcul simple", "Fonctions", "Bases", "Opérations logiques", "Résolution d'équations" et "Manipulation de matrices".

```
head_frame= tk.Frame(root, bg ='BLACK',highlightbackground='white')

menu_btn= tk.Button(head_frame,text='☰', bg='BLACK',fg ='white',command=menu_selection)
title_btn_menu=tk.Label(head_frame, text='Menu', bg ='BLACK',fg='white')
menu_btn.pack(side=tk.LEFT)
title_btn_menu.pack(side=tk.LEFT)
head_frame.pack(side=tk.TOP,fill=tk.X)
head_frame.pack_propagate(False)
head_frame.configure(height=50)



root.mainloop()
```

Un widget "**head_frame**" est créé pour afficher un bandeau en haut de la fenêtre. Un bouton et un titre sont ajoutés au bandeau. Le bouton a pour commande d'appeler la fonction de menu de sélection d'interfaces lorsqu'il est cliqué. Enfin, la fonction "**mainloop**" de la fenêtre est appelée pour rendre l'interface interactive.

Conclusion :

En conclusion, la création d'une calculatrice avec une interface graphique utilisateur (GUI) en utilisant Python s'est avérée être un projet fructueux. Nous avons réussi à développer une calculatrice fonctionnelle avec une interface utilisateur conviviale qui permet à l'utilisateur d'effectuer des calculs de manière efficace.

L'utilisation de la bibliothèque Tkinter de Python a permis de faciliter la création de l'interface graphique. Nous avons également réussi à inclure des fonctionnalités avancées telles que la gestion des erreurs et la sauvegarde des opérations récentes.

Nous sommes convaincus que cette calculatrice peut être utilisée de manière efficace pour des applications professionnelles ou personnelles. Nous espérons que cette calculatrice sera utile pour les utilisateurs et nous sommes impatients de continuer à développer de nouvelles fonctionnalités et à améliorer encore son rendement.