**1. Mục đích chương trình**

Chương trình này được xây dựng để tính gần đúng giá trị tích phân xác định của một hàm số f(x) trên đoạn [a, b] bằng hai phương pháp:

* Phương pháp Hình thang (Trapezoidal Rule)
* Phương pháp Simpson (Simpson’s Rule)

Người dùng nhập vào:

* Biểu thức hàm f(x)
* Cận dưới *a*, cận trên *b*
* Số khoảng chia *n* hoặc sai số cho phép *epsilon*

Kết quả hiển thị gồm:

* Giá trị tích phân chính xác (nếu tính được bằng giải tích)
* Kết quả xấp xỉ theo từng phương pháp
* Sai số tương ứng giữa tích phân gần đúng và tích phân chính xác
* Bảng giá trị hàm f(x) chi tiết tại các điểm chia
* Đồ thị minh họa trực quan hàm và vùng tính tích phân gần đúng

**2.** **Các thư viện trong chương trình:**

**Streamlit**

* Là thư viện xây dựng giao diện web tương tác cho Python.
* Trong chương trình, Streamlit được dùng để:
  + Tạo ô nhập hàm, chọn phương pháp, nhập số khoảng hoặc sai số.
  + Hiển thị kết quả, bảng giá trị, đồ thị và cảnh báo lỗi theo thời gian thực.
* Giúp biến chương trình Python thành ứng dụng trực quan, dễ thao tác và trình bày.

**Sympy (sp)**

* Là thư viện tính toán đại số – giải tích.
* Trong mã, Sympy giúp:
  + Phân tích cú pháp biểu thức toán học do người dùng nhập (sympify, lambdify).
  + Tính tích phân chính xác (integrate) của hàm nhập vào.
  + Phát hiện lỗi như hàm không xác định, giá trị phức, hay tiệm cận trong đoạn tích phân.
* Nhờ Sympy, chương trình có thể so sánh sai số giữa giá trị gần đúng và giá trị chính xác.

**NumPy (np)**

* Là thư viện tính toán số học hiệu năng cao, làm việc với mảng và vector.
* Trong chương trình, NumPy được dùng để:
  + Tạo các điểm chia đều trên đoạn [a, b] (np.linspace).
  + Tính giá trị hàm tại các điểm, tính tổng có trọng số các giá trị này theo công thức hình thang và Simpson.
  + Kiểm tra giá trị phức, vô hạn, NaN để đảm bảo tính đúng đắn của phép tính.
* NumPy là lõi giúp hai phương pháp tích phân chạy nhanh và chính xác. Chương trình này hoạt động tốt với tất cả các hàm mà NumPy hỗ trợ. Cụ thể:
* Các hàm lượng giác: sin(x), cos(x), tan(x), arcsin(x), arccos(x), arctan(x)
* Hàm mũ và log: có thể nhập là exp(x) hoặc e\*\*x. Loganepe của x có thể viết là ln(x) hoặc log(x). Hàm logarit các cơ số khác của x nhập là log(x, cơ số), ví dụ log(x, 10).
* Hàm căn bậc 2: nhập là √x hoặc sqrt(x).
* Hàm đa thức và số mũ: x\*\*2, x\*\*n, 1/x, …
* Hầu hết các hàm toán học cơ bản mà NumPy có thể tính theo mảng: abs(x), round(x), floor(x), ceil(x),….

**Plotly**

* Là thư viện vẽ đồ thị tương tác, hỗ trợ zoom, pan và hiển thị đẹp trong web.
* Ở đây, Plotly được dùng để vẽ đồ thị hàm f(x), vùng tô dưới đồ thị (vùng tích phân), và các điểm chia.

**Pandas**

* Tạo các dataframe từ các mảng NumPy để tính bảng giá trị hàm.

**3. Hàm chuẩn hóa biểu thức nhập vào**

def normalize\_expr(expr\_str):

expr\_str = expr\_str.lower()

replacements = {'^': '\*\*', 'ln': 'log', '√': 'sqrt', 'e': 'E'}

for k, v in replacements.items():

expr\_str = expr\_str.replace(k, v)

return expr\_str

Người dùng có thể nhập biểu thức theo thói quen toán học như x^2, ln(x) hoặc √x.  
Hàm normalize\_expr() (normalize expression) có nhiệm vụ chuyển đổi các ký hiệu này sang cú pháp hợp lệ trong Python (x\*\*2, log(x), sqrt(x)…), giúp cho Sympy hiểu và xử lý được biểu thức.

**4. Kiểm tra lỗi và điều kiện xác định của hàm**

x = sp.Symbol('x')

# Bước 1: Kiểm tra cú pháp biểu thức

try:

f\_expr = sp.sympify(expr\_str)

f\_lambda = sp.lambdify(x, f\_expr, "numpy")

except Exception as e:

st.error("Cú pháp hàm không hợp lệ. Hãy kiểm tra lại (ví dụ: sin(x), e\*\*x, x\*\*2, log(x), ...)")

st.stop()

# Bước 2: Kiểm tra miền xác định trên đoạn [a, b]

X\_test = np.linspace(a, b, 400)

try:

Y\_test = f\_lambda(X\_test)

except Exception as e:

st.error(f"Lỗi khi tính giá trị hàm trên đoạn [{a}, {b}]. Có thể hàm không xác định tại một số điểm.\n\nChi tiết: {e}")

st.stop()

# Bước 3: Kiểm tra giá trị phức, vô hạn, hoặc NaN

if np.any(np.iscomplex(Y\_test)):

st.error("Hàm trả về giá trị phức trên đoạn tích phân. "

"Vui lòng chọn khoảng không chứa điểm khiến mẫu số âm hoặc căn của số âm.")

st.stop()

if np.any(~np.isfinite(Y\_test)):

st.error("Hàm có giá trị vô hạn hoặc không xác định (inf / nan) trong khoảng tích phân.\n"

"Vui lòng chọn đoạn không chứa tiệm cận hoặc điểm kỳ dị.")

st.stop()

Để tránh lỗi khi người dùng nhập các hàm không xác định trên đoạn [a,b], chương trình kiểm tra:

* Hàm có thể được tính hợp lệ trên toàn bộ đoạn
* Không chứa giá trị phức (complex), vô hạn (inf), hoặc không xác định (NaN)

Nếu vi phạm, ứng dụng sẽ báo lỗi hoặc cảnh báo cụ thể để người dùng chỉnh lại.

**5. Hai công thức tính gần đúng tích phân**

**a. Phương pháp Hình thang (Trapezoidal Rule)**

def trapezoidal\_rule(f, a, b, n):

x = np.linspace(a, b, n + 1)

y = f(x)

h = (b - a) / n

return h \* (y[0]/2 + np.sum(y[1:-1]) + y[-1]/2)

### Ý tưởng: Chia đoạn [a, b] thành n khoảng nhỏ bằng nhau, nối các điểm (xᵢ, f(xᵢ)) bằng các đoạn thẳng, và lấy tổng diện tích hình thang dưới từng đoạn. Nói cách khác:

### Đồ thị được **nội suy tuyến tính** trên từng đoạn con.

* Phần diện tích được tính chính là tổng **diện tích các hình thang** nằm **giữa đoạn thẳng nội suy đó và trục hoành**.

Tích phân xấp xỉ sẽ là kết quả chính xác tuyệt đối (sai số bằng 0) nếu hàm f(x) từ bậc 1 trở xuống.

**b. Phương pháp Simpson (Simpson’s Rule)**

def simpson\_rule(f, a, b, n):

if n % 2 == 1:

n += 1

x = np.linspace(a, b, n + 1)

y = f(x)

h = (b - a) / n

return (h/3)\*(y[0] + y[-1] + 4\*np.sum(y[1:-1:2]) + 2\*np.sum(y[2:-2:2]))

Ý tưởng: Phương pháp Simpson không lấy diện tích dưới các đoạn thẳng nối các điểm chia, mà lấy diện tích dưới một parabol nội suy (xấp xỉ) đi qua ba điểm chia liên tiếp của đồ thị.

* Nếu f(x) là bậc ≤ 2 suy ra cung parabol trùng với f(x) suy ra Simpson cho kết quả chính xác tuyệt đối.
* Nếu f(x) cong phức tạp hơn (bậc cao hơn) suy ra cung parabol chỉ là xấp xỉ gần đúng, và sai số phụ thuộc vào độ cong (đạo hàm bậc 4 của f).

**6. Hàm lựa chọn tính theo sai số cho phép hoặc khoảng chia**

def compute\_with\_tolerance(f, a, b, rule\_func, epsilon=None, n=None):

prev = None

if epsilon is not None:

n = 2

while True:

I1 = rule\_func(f, a, b, n)

if prev is not None and abs(I1 - prev) < epsilon:

break

prev = I1

n \*= 2

else:

I1 = rule\_func(f, a, b, n)

return I1, n

Hàm này cho phép lựa chọn hai cách nhập:

* Nếu người dùng nhập sai số cho phép *epsilon*: chương trình tự động tăng dần số khoảng chia (ban đầu n = 2) cho đến khi độ chênh giữa hai lần tính liên tiếp nhỏ hơn sai số cho phép.
* Nếu người dùng nhập số khoảng chia *n*: chương trình tính trực tiếp.

Hàm trả về:

* Giá trị tích phân gần đúng I1
* Số khoảng chia thực tế đã dùng n

**7. Tính tích phân chính xác bằng sympy**

I\_exact = float(sp.integrate(f\_expr, (x, a, b)))

Thư viện SymPy có khả năng tính tích phân chính xác.  
Nếu biểu thức có tích phân giải tích, kết quả chính xác được dùng để so sánh sai số với các phương pháp gần đúng.  
Trường hợp không tính được, chương trình sẽ hiển thị cảnh báo và chỉ hiển thị kết quả gần đúng.

**8. Tính sai số thực nghiệm**

err\_trap = abs(I\_trap - I\_exact)

err\_simp = abs(I\_simp - I\_exact)

Lấy hiệu của tích phân chính xác (tính bằng thư viện Sympy) với tích phân gần đúng mà ta tính được.

**9. Hàm tính sai số lý thuyết**

def theoretical\_error(f\_expr, a, b, n, method):

x = sp.Symbol('x')

try:

if method == "Hình thang":

f2 = sp.diff(f\_expr, x, 2)

f2\_lamb = sp.lambdify(x, f2, "numpy")

M2 = np.max(np.abs(f2\_lamb(np.linspace(a, b, 1000))))

return (b - a)\*\*3 / (12 \* n\*\*2) \* M2

elif method == "Simpson":

f4 = sp.diff(f\_expr, x, 4)

f4\_lamb = sp.lambdify(x, f4, "numpy")

M4 = np.max(np.abs(f4\_lamb(np.linspace(a, b, 1000))))

return (b - a)\*\*5 / (180 \* n\*\*4) \* M4

except:

return None

* err\_trap\_theory: error trapezoidal theoretical, tức là ước lượng sai số lý thuyết của phương pháp Hình thang.
* err\_simp\_theory: error Simpson theoretical, tức là ước lượng sai số lý thuyết của phương pháp Simpson.
* Ta có thể lấy giá trị cực đại tuyệt đối của đạo hàm bậc 2 (đối với ơhwuwong pháp Hình thang) hoặc bậc 4 (đối với Simpson) trong [a,b] để ước lượng sai số lý thuyết.
* Nếu không tính được đạo hàm (hàm phức tạp), sẽ hiện None và bỏ qua.

**10. Bảng giá trị tại các điểm chia**

def make\_table(X, Y, method):

n = len(X) - 1

h = (X[-1] - X[0]) / n

if method == "Hình thang":

weights = np.ones\_like(Y)

weights[0] = weights[-1] = 0.5

elif method == "Simpson":

weights = np.ones\_like(Y)

weights[1:-1:2] = 4

weights[2:-2:2] = 2

wf = weights \* Y

total = np.sum(wf)

Bảng này giúp minh họa rõ hơn cơ chế tính toán của từng phương pháp:

* Cột trọng số wᵢ thể hiện hệ số nhân tương ứng trong công thức tính tích phân.
* Cột wᵢ·f(xᵢ) là giá trị đóng góp của từng điểm vào tổng tích phân.

### 11. Biểu đồ minh họa

Để trực quan hóa các phương pháp tính tích phân, hệ thống sử dụng thư viện Plotly để vẽ đồ thị và các thành phần liên quan:

* Đồ thị hàm f(x)
  + Biểu diễn bằng đường liền nét màu xanh dương.
  + Thể hiện biến thiên của hàm trên khoảng tích phân đã cho.
* Các điểm chia và đường nối giữa chúng
  + Các điểm lấy mẫu được nối với nhau bằng đường nét đứt màu xanh lá cây.
  + Giúp quan sát số khoảng chia và vị trí các điểm được sử dụng trong tính tích phân.
* Vùng diện tích được tích phân
  + Phương pháp hình thang: diện tích gần đúng được tính bằng tổng các hình thang, giới hạn bởi đường nối các điểm chia và trục hoành.
  + Phương pháp Simpson: diện tích gần đúng được giới hạn bởi các cung parabol nội suy nối các điểm chia và trục hoành, được tô màu xanh ngọc.

**12. Tóm tắt**

| **Thành phần** | **Chức năng chính** |
| --- | --- |
| normalize\_expr() | Chuẩn hóa biểu thức nhập của người dùng |
| sympy.integrate() | Tính tích phân chính xác (nếu có thể) |
| trapezoidal\_rule() | Tính gần đúng tích phân theo quy tắc hình thang |
| simpson\_rule() | Tính gần đúng tích phân theo quy tắc Simpson |
| compute\_with\_tolerance() | Lặp tăng số khoảng hoặc dừng theo sai số cho phép |
| theoretical\_error() | TÍnh sai số lý thuyết |
| make\_table() | Tạo bảng minh họa trọng số và giá trị hàm tại các điểm chia |

**12. Hạn chế**

**Hạn chế về loại hàm và miền xác định**

* Code hiện tại chỉ xử lý các hàm thường xác định, giá trị thực trên toàn đoạn [a, b].
* Các hàm gây lỗi:
* Hàm có tiệm cận dọc, ví dụ 1/(x−1) khi đoạn tích phân chứa x = 1.
* Hàm chứa căn bậc chẵn của số âm, ví dụ khi đoạn chứa x < 2.
* Hàm có giá trị phức, ví dụ log(−x).
* Các hàm không có sẵn trong NumPy (ví dụ gamma(x), erf(x) nếu không import thêm) sẽ không chạy.
* Hiện tại, code sẽ thông báo lỗi và không cho phép tính gần đúng trong những trường hợp này.