CÔNG THỨC NỘI SUY STIRLING VÀ BESSEL

Nguyễn Việt Anh 20200039

Hà Nội 2022

MI-TN K65

Giảng viên hướng dẫn: Hà Thị Ngọc Yến

Mục lục

1 Giới thiệu

Cho hàm số $y = f(x)$ liên tục trên đoạn $[a, b]$, được biểu diễn qua bảng các giá trị

$$y_i = f(x_i)\ (x_i \in [a, b], i = \overline{0, n})$$

Cụ thể hơn

x	x_0	x_1	x_2	...	x_n
$f(x)$	y_0	y_1	y_2	...	y_n

Với $\{(x_i, y_i)\}_{i=0}^{n}$ là $n+1$ bộ số cho trước, việc xây dựng 1 hàm số xấp xỉ $f(x)$ có $n+1$ điểm giá trị trên trùng với $f(x)$, được gọi là nội suy.

Trên thực tế có rất nhiều phương pháp nội suy nhiều biến phức tạp, xử lý những hàm số biến thiên khó đoán. Ví dụ như phép nội suy *Krigin* là công cụ quan trọng trong địa chất học, dùng để xác định độ ẩm, nhiệt độ,... của địa hình; hay nội suy *IDW (Inverse Distance Weighting)* được sử dụng nhiều trong Digital Elevation Model và dự báo thời tiết.

Tuy nhiên mô hình đơn giản nhất cho nội suy là nội suy đa thức, và bài báo cáo này sẽ giới thiệu 2 công thức nội suy đa thức khá tối ưu: nội suy Stirling và Bessel.

2 Kiến thức chung

2.1 Sai phân

Với n điểm $\in \mathbb{R}^2$ (còn được gọi là các điểm dữ liệu) tạo thành bộ $\{(x_i, y_i)\}_{i=1}^{n}$, với $x_{k+1} - x_k = h$ là hằng số $k = \overline{1, n-1}$:

$$f(x_i) = y_i$$

Ta định nghĩa sai phân tại y_k (sai phân cấp 1) là:

$$\Delta y_k = y_{k+1} - y_k$$

Với $p \in \mathbb{N}$, ta định nghĩa sai phân cấp p là:

$$\Delta^p y_k = \Delta^{p-1} y_{k+1} - \Delta^{p-1} y_k$$

2.2 Bảng sai phân

Định nghĩa: Bảng sai phân là 1 bảng liệt kê toàn bộ các cấp sai phân sinh ra được từ 1 bộ điểm cho trước $\{(x_i, y_i)\}_{i=-n}^{n}$. Ví dụ với $n = 3$:

x	y	Δy	$\Delta^2 y$	$\Delta^3 y$	$\Delta^4 y$	$\Delta^5 y$	$\Delta^6 y$	...
$\vdots$	$\vdots$							
x_{-3}	y_{-3}							
		Δy_{-3}						
x_{-2}	y_{-2}		$\Delta^2 y_{-3}$					
		Δy_{-2}		$\Delta^3 y_{-3}$				
x_{-1}	y_{-1}		$\Delta^2 y_{-2}$		$\Delta^4 y_{-3}$			
		Δy_{-1}		$\Delta^3 y_{-2}$		$\Delta^5 y_{-3}$		
x_0	y_0		$\Delta^2 y_{-1}$		$\Delta^4 y_{-2}$		$\Delta^6 y_{-3}$	...
		Δy_0		$\Delta^3 y_{-1}$		$\Delta^5 y_{-2}$		
x_1	y_1		$\Delta^2 y_0$		$\Delta^4 y_{-1}$			
		Δy_1		$\Delta^3 y_0$				
x_2	y_2		$\Delta^2 y_1$					
		Δy_2						
x_3	y_3							
$\vdots$	$\vdots$							

2.3 Nội suy đa thức

Định nghĩa: Nội suy đa thức là phương pháp xấp xỉ 1 hàm chưa biết $f(x)$ bằng 1 đa thức $P(x)$ từ 1 bộ k điểm (còn được gọi là mốc nội suy) $\{(x_i, y_i)\}_{i=1}^{n}$ ($P(x)$ gọi là đa thức nội suy)

- **Tính chất cơ bản:**

1. $P(x_i) = f(x_i) \quad , i = \overline{1, k}$
2. $\deg P = k - 1$
3. Đa thức nội suy từ 1 bộ điểm là tồn tại duy nhất

Tính chất thứ nhất và thứ hai có thể thấy hiển nhiên, bạn đọc tự chứng minh, ta sẽ đi chứng minh tính chất thứ 3

Chứng minh: Giả sử tồn tại $P_1(x)$ và $P_2(x)$ là 2 đa thức bậc $n-1$ thỏa mãn:

$$P_1(x_i) = P_2(x_i) = f(x_i) \quad , i = \overline{1, n}$$

Thấy rằng đa thức $G(x) = P_1(x) - P_2(x)$ có ít nhất n nghiệm phân biệt (từ x_1 đến x_n) đồng thời $\deg G \leq n - 1$ (do $\deg P_1 = \deg P_2 = n - 1$). Vì thế, $G(x) \equiv 0$, vì thế $P_1(x) \equiv P_2(x)$ và đa thức nội suy tồn tại duy nhất.

2.4 Công thức nội suy Gauss I

Giả sử bộ dữ liệu được cho là $\{(x_i, y_i)\}_{i=-n}^{n}$ .Công thức nội suy trung tâm Gauss I là:

$$\begin{aligned}P(x_0 + ht) = \; & y_0 + t\Delta y_0 + \frac{t(t-1)}{2!}\Delta^2 y_{-1} + \frac{(t+1)t(t-1)}{3!}\Delta^3 y_{-1} \\ & + \frac{(t+1)t(t-1)(t-2)}{4!}\Delta^4 y_{-2} + \cdots + \frac{(t+n-1)(t+n-1)\ldots(t-n)}{(2n)!}\Delta^{2n} y_{-n}\end{aligned}$$

(NOTE: Do chủ đề về nội suy Gauss đã được trình bày ở các báo cáo trước, nên báo cáo này chỉ nhắc lại và không chứng minh chi tiết)

Sai số của công thức trên được xấp xỉ là:

$$|R(x)| \leq \left| \frac{\Delta^{2n} y_{-n}}{(2n)!} . t \prod_{i=1}^{n} (t^2 - i^2) \right|$$

Từ công thức trên có thể thấy thứ tự sử dụng sai phân là $y_0 \to \Delta y_0 \to \Delta^2 y_{-1} \to \Delta^3 y_{-1} \to \cdots \to \Delta^{2n} y_{-n}$. Từ đó ta xây dựng bảng sai phân cho công thức Gauss I, giả sử với 7 điểm $\{(x_i, y_i)\}_{i=-3}^{3}$:

x	y	Δy	$\Delta^2 y$	$\Delta^3 y$	$\Delta^4 y$	$\Delta^5 y$	$\Delta^6 y$	...
$\vdots$	$\vdots$							
x_{-3}	y_{-3}							
		Δy_{-3}						
x_{-2}	y_{-2}		$\Delta^2 y_{-3}$					
		Δy_{-2}		$\Delta^3 y_{-3}$				
x_{-1}	y_{-1}		$\Delta^2 y_{-2}$		$\Delta^4 y_{-3}$			
		Δy_{-1}		$\Delta^3 y_{-2}$		$\Delta^5 y_{-3}$		
x_0	y_0		$\Delta^2 y_{-1}$		$\Delta^4 y_{-2}$		$\Delta^6 y_{-3}$	...
		Δy_0		$\Delta^3 y_{-1}$		$\Delta^5 y_{-2}$		
x_1	y_1		$\Delta^2 y_0$		$\Delta^4 y_{-1}$			
		Δy_1		$\Delta^3 y_0$				
x_2	y_2		$\Delta^2 y_1$					
		Δy_2						
x_3	y_3							
$\vdots$	$\vdots$							

Bảng sai phân nội suy Gauss I

Mũi tên liền trong bảng sai phân trên đi qua tất cả các vị trí sai phân mà công thức Gauss I sẽ lần lượt sử dụng

2.5 Công thức nội suy Gauss II

Ta xét với cùng bộ dữ liệu như công thức Gauss I. Công thức nội suy Gauss II là:

$$\begin{aligned} P(x_0+ht) = \; & y_0 + t\Delta y_{-1} + \frac{t(t+1)}{2!}\Delta^2 y_{-1} + \frac{(t+1)t(t-1)}{3!}\Delta^3 y_{-2} \\ & + \frac{(t+2)(t+1)t(t-1)}{4!}\Delta^4 y_{-2} + \cdots + \frac{(t+n)(t+n-1)\dots(t-n+1)}{(2n)!}\Delta^{2n} y_{-n} \end{aligned}$$

Sai số của công thức Gauss II là:

$$|R(x)| \leq \left| \frac{\Delta^{2n} y_{-n}}{(2n)!}.t\prod_{i=1}^{n}(t^2-i^2) \right|$$

Hơi khác với công thức Gauss I, có thể thấy thứ tự sử dụng sai phân của công thức Gauss II là $y_0 \to \Delta y_{-1} \to \Delta^2 y_{-1} \to \Delta^3 y_{-2} \to \cdots \to \Delta^{2n} y_{-n}$. Vậy nên bảng sai phân của công thức Gauss II, ví dụ với bộ 7 điểm $\{(x_i, y_i)\}_{i=-3}^{3}$:

x	y	Δy	$\Delta^2 y$	$\Delta^3 y$	$\Delta^4 y$	$\Delta^5 y$	$\Delta^6 y$	...
$\vdots$	$\vdots$							
x_{-3}	y_{-3}							
		Δy_{-3}						
x_{-2}	y_{-2}		$\Delta^2 y_{-3}$					
		Δy_{-2}		$\Delta^3 y_{-3}$				
x_{-1}	y_{-1}		$\Delta^2 y_{-2}$		$\Delta^4 y_{-3}$			
		Δy_{-1}		$\Delta^3 y_{-2}$		$\Delta^5 y_{-3}$		
x_0	y_0		$\Delta^2 y_{-1}$		$\Delta^4 y_{-2}$		$\Delta^6 y_{-3}$	...
		Δy_0		$\Delta^3 y_{-1}$		$\Delta^5 y_{-2}$		
x_1	y_1		$\Delta^2 y_0$		$\Delta^4 y_{-1}$			
		Δy_1		$\Delta^3 y_0$				
x_2	y_2		$\Delta^2 y_1$					
		Δy_2						
x_3	y_3							
$\vdots$	$\vdots$							

Bảng sai phân nội suy Gauss II

Ta sử dụng đường nét đứt làm "đường đi"trong bảng sai phân cho công thức Gauss II, để phân biệt với đường nét liền của công thức Gauss I.

3 Lý thuyết chính

3.1 Công thức nội suy Stirling

3.1.1 Ý tưởng

Giả sử ta được cho 1 bộ điểm dữ liệu, công thức nội suy Gauss I và II đã đưa ra cách kết nạp mốc nội suy tối ưu hơn về mặt sai số khi "đường đi"trên bảng sai phân xoay quanh 1 điểm "chính giữa".

Cách tiếp cận trên nhìn chung là khá tốt về độ chính xác tuy nhiên chưa tối ưu về mặt khối lượng tính toán, nhưng nếu để ý kỹ vào các thành phần ví dụ như $t(t+1)(t-1)$ nếu có thể ghép cặp các nhân tử 'đối nhau': $t+i$ và $t-i$ để tạo ra 1 hàm hoặc chẵn hoặc lẻ, và xét đa thức ban đầu theo ẩn t^2 thay cho t, số lượng tính toán sẽ giảm đi rất nhiều.

Ví dụ:

$$(t-p)(t-(p-1))\dots(t+p) = t\left(t^2-1\right)\left(t^2-4\right)\dots\left(t^2-p^2\right)$$

Thấy rằng tuy bằng nhau, nhưng vế trái cần $2p+1$ phép cộng trừ và $2p$ phép nhân, còn vế phải nếu coi t^2 là ẩn và thực hiện nhân riêng t lẻ ra, máy tính chỉ cần thực hiện $p+1$ phép nhân và p phép cộng trừ

Tuy nhiên những phần tử ví dụ như $(t+2)(t+1)t(t-1)$ ở công thức Gauss I khi ghép cặp sẽ chừa ra $t+2$, nhưng thú vị rằng "nửa còn lại"của nó lại được tìm thấy ở công thức Gauss II, tại phần tử $(t+1)t(t-1)(t-2)$

$\Longrightarrow$ Công thức nội suy Stirling sẽ cụ thể hóa ý tưởng trên

3.1.2 Xây dựng công thức

Cho bộ điểm gồm $2n+1$ mốc nội suy cách đều $\{(x_i, y_i)\}_{i=-n}^{n}$ (tức là $x_{k+1} - x_k = h$ là hằng số), công thức nội suy Stirling bằng trung bình cộng của công thức Gauss I và II

$$\text{Stirling} = \frac{\text{Gauss I} + \text{Gauss II}}{2}$$

Ta viết lại hai công thức quan trọng:

1. **Công thức nội suy Gauss I**:

$$\begin{aligned} P(x_0+ht) = {} & y_0 + t\Delta y_0 + \frac{t(t-1)}{2!}\Delta^2 y_{-1} + \frac{(t+1)t(t-1)}{3!}\Delta^3 y_{-1} \\ & + \frac{(t+1)t(t-1)(t-2)}{4!}\Delta^4 y_{-2} + \dots + \frac{(t+n-1)(t+n-1)\dots(t-n)}{(2n)!}\Delta^{2n} y_{-n} \end{aligned}$$

2. **Công thức nội suy Gauss II**:

$$\begin{aligned} P(x_0+ht) = {} & y_0 + t\Delta y_{-1} + \frac{t(t+1)}{2!}\Delta^2 y_{-1} + \frac{(t+1)t(t-1)}{3!}\Delta^3 y_{-2} \\ & + \frac{(t+2)(t+1)t(t-1)}{4!}\Delta^4 y_{-2} + \dots + \frac{(t+n)(t+n-1)\dots(t-n+1)}{(2n)!}\Delta^{2n} y_{-n} \end{aligned}$$

Từ hai công thức trên ta rút ra dạng tổng quát của sai phân các cấp của Gauss I và Gauss II

a. **Sai phân cấp lẻ** $2k+1$:

$$\textbf{Gauss I: } \frac{\prod_{i=-k}^{k}(t-i)}{(2k+1)!}\Delta^{2k+1}y_{-k} \quad \textbf{, Gauss II: } \frac{\prod_{i=-k}^{k}(t-i)}{(2k+1)!}\Delta^{2k+1}y_{-k-1}$$

Cộng 2 tích trên chia đôi ta sẽ được sai phân cấp thứ $2k+1$ của công thức nội suy Stirling là:

$$\frac{\prod_{i=-k}^{k}(t-i)}{(2k+1)!}\frac{\Delta^{2k+1}y_{-k}+\Delta^{2k+1}y_{-k-1}}{2} = \frac{t\prod_{i=1}^{k}(t^2-i^2)}{(2k+1)!}\frac{\Delta^{2k+1}y_{-k}+\Delta^{2k+1}y_{-k-1}}{2}$$

$\Longrightarrow$ Chúng ta đã cụ thể hóa thành công ý tưởng

b. **Sai phân cấp chẵn** $2k$:

$$\textbf{Gauss I: } \frac{\prod_{i=-k}^{k-1}(t-i)}{(2k)!}\Delta^{2k}y_{-k} \quad \textbf{, Gauss II: } \frac{\prod_{i=-k+1}^{k}(t-i)}{(2k)!}\Delta^{2k}y_{-k}$$

Cộng 2 tích trên vào chia đôi ta được:

$$\frac{1}{2}(t+k+t-k)\frac{\prod_{i=-k+1}^{k-1}(t-i)}{(2k)!}\Delta^{2k}y_{-k} = \frac{\prod_{i=0}^{k-1}(t^2-i^2)}{(2k)!}\Delta^{2k}y_{-k}$$

Tổng hợp 2 kết quả trên ta được công thức nội suy Stirling với $2n+1$ mốc nội suy:

$$\begin{aligned} P(x_0+ht) = \ & y_0 + t\frac{\Delta y_0+\Delta y_{-1}}{2} + \frac{t^2}{2!}\Delta^2 y_{-1} + \frac{(t+1)t(t-1)}{3!}\frac{\Delta^3 y_{-1}+\Delta^3 y_{-2}}{2} + \dots \\ & + \frac{t(t^2-1^2)(t^2-2^2)\dots(t^2-(n-1)^2)}{(2n-1)!}\frac{\Delta^{2n-1}y_{-(n-1)}+\Delta^{2n-1}y_{-n}}{2} \\ & + \frac{t^2(t^2-1^2)(t^2-2^2)\dots(t^2-(n-1)^2)}{(2n)!}\Delta^{2n}y_{-n} \end{aligned}$$

Xem xét bảng sai phân cho công thức Stirling, ta sẽ không còn di chuyển trên 1 đường đi, ma sẽ đi đồng thời 2 đường đi cùng lúc:

x	y	Δy	$\Delta^2 y$	$\Delta^3 y$	$\Delta^4 y$	$\Delta^5 y$	$\Delta^6 y$	...
$\vdots$	$\vdots$							
x_{-3}	y_{-3}							
		Δy_{-3}						
x_{-2}	y_{-2}		$\Delta^2 y_{-3}$					
		Δy_{-2}		$\Delta^3 y_{-3}$				
x_{-1}	y_{-1}		$\Delta^2 y_{-2}$		$\Delta^4 y_{-3}$			
		Δy_{-1}		$\Delta^3 y_{-2}$		$\Delta^5 y_{-3}$		
x_0	y_0		$\Delta^2 y_{-1}$		$\Delta^4 y_{-2}$		$\Delta^6 y_{-3}$	...
		Δy_0		$\Delta^3 y_{-1}$		$\Delta^5 y_{-2}$		
x_1	y_1		$\Delta^2 y_0$		$\Delta^4 y_{-1}$			
		Δy_1		$\Delta^3 y_0$				
x_2	y_2		$\Delta^2 y_1$					
		Δy_2						
x_3	y_3							
$\vdots$	$\vdots$							

Bảng sai phân nội suy Stirling

(**Chú thích:** $\longrightarrow$ tượng trưng cho công thức Gauss I, $\dashrightarrow$ tượng trưng cho công thức Gauss II)

3.1.3 Sai số công thức

Sai số của nội suy Stirling cũng bằng trung bình cộng 2 sai số của nội suy Gauss I và Gauss II. Với $2n+1$ mốc nội suy, như đã đề cập ở phần lý thuyết chung, sai số của Gauss I và Gauss II bằng nhau và bằng:

$$\left|\frac{\Delta^{2n} y_{-n}}{(2n)!}.t\prod_{i=1}^{n}(t^2-i^2)\right|$$

$\Longrightarrow$ Sai số của nội suy Stirling sẽ là:

$$|R(x)| \leq \left|\frac{\Delta^{2n} y_{-n}}{(2n)!}.t\prod_{i=1}^{n}(t^2-i^2)\right|$$

3.2 Công thức nội suy Bessel

3.2.1 Ý tưởng

Ý tưởng của công thức Bessel tương tự với nội suy Stirling chỉ khác là ta sẽ thay đổi công thức 1 chút để có thể làm việc với chẵn điểm

3.2.2 Xây dựng công thức

Cho $2n+2$ điểm dữ liệu $\{(x_i, y_i)\}_{i=-n}^{n+1}$, công thức nội suy Bessel cũng bằng trung bình cộng của công thức Gauss I và công thức Gauss II, tuy nhiên mốc nội suy trung tâm của nội suy Gauss II được dịch chuyển sang vị trí tiếp theo:

$$\text{Bessel} = \frac{\text{Gauss I} + \text{Gauss II (chuyển)}}{2}$$

Ta viết lại hai công thức quan trọng khi có $2n+2$ điểm dữ liệu:

1. **Công thức Gauss I** (mốc nội suy trung tâm tại x_0):

$$\begin{aligned} P(x_0+ht) = \ & y_0 + t\Delta y_0 + \frac{t(t-1)}{2!}\Delta^2 y_{-1} + \frac{(t+1)t(t-1)}{3!}\Delta^3 y_{-1} \\ & + \frac{(t+1)t(t-1)(t-2)}{4!}\Delta^4 y_{-2} + \cdots + \frac{(t+n)(t+n-1)\ldots(t-n)}{(2n+1)!}\Delta^{2n+1} y_{-n} \end{aligned}$$

2. **Công thức Gauss II** (mốc nội suy trung tâm tại $x_1 = x_0 + h$):

$$\begin{aligned} P(x_1+ht) = \ & y_1 + t\Delta y_0 + \frac{t(t+1)}{2!}\Delta^2 y_0 + \frac{(t+1)t(t-1)}{3!}\Delta^3 y_{-1} \\ & + \frac{(t+2)(t+1)t(t-1)}{4!}\Delta^4 y_{-1} + \cdots + \frac{(t+n)(t+n-1)\ldots(t-n)}{(2n+1)!}\Delta^{2n+1} y_{-n} \end{aligned}$$

Tuy nhiên, để có thể cộng vào chia đôi, ta cần xét công thức Gauss I và Gauss II tại cùng 1 giá trị, trong khi $x_1 + ht = (x_0 + ht) + t$, nên ta thực phép đổi biến t như sau

- Thay t bởi $t + \frac{1}{2}$ vào **Công thức Gauss I** ta được:

$$\begin{aligned} P\left(x_0 + \frac{h}{2} + ht\right) = \ & y_0 + \left(t+\frac{1}{2}\right)\Delta y_0 + \frac{\left(t+\frac{1}{2}\right)\left(t-\frac{1}{2}\right)}{2!}\Delta^2 y_{-1} \\ & + \frac{\left(t+\frac{3}{2}\right)\left(t+\frac{1}{2}\right)\left(t-\frac{1}{2}\right)}{3!}\Delta^3 y_{-1} + \frac{\left(t+\frac{3}{2}\right)\left(t+\frac{1}{2}\right)\left(t-\frac{1}{2}\right)\left(t-\frac{3}{2}\right)}{4!}\Delta^4 y_{-2} \\ & + \cdots + \frac{\left(t+\frac{2n+1}{2}\right)\left(t+\frac{2n-1}{2}\right)\ldots\left(t-\frac{2n-1}{2}\right)}{(2n+1)!}\Delta^{2n+1} y_{-n} \end{aligned}$$

- Thay t bởi $t - \dfrac{1}{2}$ vào **Công thức Gauss II** ta được:

$$\begin{aligned} P\left(x_1 - \frac{h}{2} + ht\right) = \; & y_1 + \left(t - \frac{1}{2}\right)\Delta y_0 + \frac{\left(t - \frac{1}{2}\right)\left(t + \frac{1}{2}\right)}{2!}\Delta^2 y_0 \\ & + \frac{\left(t + \frac{1}{2}\right)\left(t - \frac{1}{2}\right)\left(t - \frac{3}{2}\right)}{3!}\Delta^3 y_{-1} + \frac{\left(t + \frac{3}{2}\right)\left(t + \frac{1}{2}\right)\left(t - \frac{1}{2}\right)\left(t - \frac{3}{2}\right)}{4!}\Delta^4 y_{-1} \\ & + \cdots + \frac{\left(t + \frac{2n-1}{2}\right)\left(t + \frac{2n-3}{2}\right)\ldots\left(t - \frac{2n+1}{2}\right)}{(2n+1)!}\Delta^{2n+1} y_{-n} \end{aligned}$$

Bây giờ 2 công thức Gauss I và II đã xét tại cùng 1 điểm (do $x_0 + \frac{h}{2} + ht = x_1 - \frac{h}{2} + ht$, ta thống nhất là $x_0 + \frac{h}{2} + ht$), ta xét sai phân bậc chẵn và lẻ 2 công thức lúc này:

a. **Sai phân cấp lẻ** $2k+1$:

$$\textbf{Gauss I: } \frac{\prod\limits_{i=-k-1}^{k-1}\left(t - \frac{1}{2} - i\right)}{(2k+1)!}\Delta^{2k+1} y_{-k} \quad \textbf{, Gauss II:} \frac{\prod\limits_{i=-k}^{k}\left(t - \frac{1}{2} - i\right)}{(2k+1)!}\Delta^{2k+1} y_{-k}$$

Cộng 2 kết quả trên ta được sai phân cấp lẻ $2k+1$ của công thức Bessel:

$$\frac{t\prod\limits_{i=0}^{k-1}\left(t^2 - \left(i + \frac{1}{2}\right)^2\right)}{(2k+1)!}\Delta^{2k+1} y_{-k}$$

b. **Sai phân cấp chẵn** $2k$:

$$\textbf{Gauss I: } \frac{\prod\limits_{i=-k}^{k-1}\left(t - \frac{1}{2} - i\right)}{(2k)!}\Delta^{2k} y_{-k} \quad \textbf{, Gauss II: } \frac{\prod\limits_{i=-k}^{k-1}\left(t - \frac{1}{2} - i\right)}{(2k)!}\Delta^{2k} y_{-(k-1)}$$

Cộng 2 kết quả trên ra được sai phân cấp chẵn $2k$ của công thức Bessel:

$$\frac{\prod\limits_{i=0}^{k-1}\left(t^2 - \left(i + \frac{1}{2}\right)^2\right)}{(2k)!}\frac{\Delta^{2k} y_{-k} + \Delta^{2k} y_{-(k-1)}}{2}$$

Vậy ta có công thức nội suy Bessel với $2n+2$ điểm dữ liệu:

$$\begin{aligned}
P\left(x_0+\frac{h}{2}+ht\right) = {} & \frac{y_0+y_1}{2}+t\Delta y_0+\frac{t^2-\frac{1}{4}}{2!}\frac{\Delta^2 y_{-1}+\Delta^2 y_0}{2}+\frac{t\left(t^2-\frac{1}{4}\right)}{3!}\Delta^3 y_{-1} \\
& +\frac{\left(t^2-\frac{1}{4}\right)\left(t^2-\frac{9}{4}\right)}{4!}\frac{\Delta^4 y_{-2}+\Delta^4 y_{-1}}{2}+\dots \\
& +\frac{\left(t^2-\frac{1}{4}\right)\left(t^2-\frac{9}{4}\right)\dots\left(t^2-\left(n-\frac{1}{2}\right)^2\right)}{(2n)!}\frac{\Delta^{2n} y_{-n}+\Delta^{2n} y_{-(n-1)}}{2} \\
& +\frac{t\left(t^2-\frac{1}{4}\right)\left(t^2-\frac{9}{4}\right)\dots\left(t^2-\left(n-\frac{1}{2}\right)^2\right)}{(2n+1)!}\Delta^{2n+1} y_{-n}
\end{aligned}$$

Bảng sai phân của công thức Bessel cũng sẽ đi cả 2 nhánh giống như Stirling, nhưng do mốc nội suy trung tâm của công thức thành phần Gauss II đã chuyển sang vị trí tiếp theo, vì thế điểm xuất phát của công thức Gauss II (đường nét đứt) sẽ xuất phát từ x_1. Cụ thể, ví dụ với 6 điểm $\{(x_i, y_i)\}_{i=-2}^{3}$:

x	y	Δy	$\Delta^2 y$	$\Delta^3 y$	$\Delta^4 y$	$\Delta^5 y$	$\dots$
$\vdots$	$\vdots$						
x_{-2}	y_{-2}						
		Δy_{-2}					
x_{-1}	y_{-1}		$\Delta^2 y_{-2}$				
		Δy_{-1}		$\Delta^3 y_{-2}$			
x_0	y_0		$\Delta^2 y_{-1}$		$\Delta^4 y_{-2}$		
		Δy_0		$\Delta^3 y_{-1}$		$\Delta^5 y_{-2}$	$\dots$
x_1	y_1		$\Delta^2 y_0$		$\Delta^4 y_{-1}$		
		Δy_1		$\Delta^3 y_0$			
x_2	y_2		$\Delta^2 y_1$				
		Δy_2					
x_3	y_3						
$\vdots$	$\vdots$						

Bảng sai phân nội suy Bessel

(**Chú thích:** $\longrightarrow$ tượng trưng cho công thức Gauss I, $\dashrightarrow$ tượng trưng cho công thức Gauss II)

3.2.3 Sai số công thức

Với công thức sai số của nội suy Bessel, ta xuất phát từ công thức sai số của Gauss I và Gauss II khi có $2n+2$ điểm dữ liệu $\{(x_i, y_i)\}_{i=-n+1}^{n}$, với mốc nội suy trung tâm lần lượt là x_0 và x_1.

1. Sai số của công thức Gauss I với x_0 là mốc nội suy trung tâm, với $2n+2$ điểm dữ liệu:

$$|R(x_0+ht)| \leq \left|\frac{\Delta^{2n+1}y_{-n}}{(2n+1)!}\prod_{i=-n}^{n+1}(t-i)\right|$$

Ta thực hiện thay t bởi $t+\frac{1}{2}$ giống khi xây dựng công thức

$$\begin{aligned}\left|R\left(x_0+\frac{h}{2}+ht\right)\right| &\leq \left|\frac{\Delta^{2n+1}y_{-n}}{(2n+1)!}\prod_{i=-n}^{n+1}\left(t+\frac{1}{2}-i\right)\right| \\ &= \left|\frac{\Delta^{2n+1}y_{-n}}{(2n+1)!}\prod_{i=0}^{n}\left(t^2-\left(i+\frac{1}{2}\right)^2\right)\right|\end{aligned}$$

2. Sai số của công thức Gauss II với $x_1 = x_0 + h$ là mốc nội suy trung tâm, với $2n+2$ điểm dữ liệu:

$$|R(x_1+ht)| \leq \left|\frac{\Delta^{2n+1}y_{-n}}{(2n+1)!}\prod_{i=-(n+1)}^{n}(t-i)\right|$$

Ta thực hiện thay t bởi $t-\frac{1}{2}$ giống khi xây dựng công thức

$$\begin{aligned}\left|R\left(x_1-\frac{h}{2}+ht\right)\right| &\leq \left|\frac{\Delta^{2n+1}y_{-n}}{(2n+1)!}\prod_{i=-(n+1)}^{n}\left(t-\frac{1}{2}-i\right)\right| \\ &= \left|\frac{\Delta^{2n+1}y_{-n}}{(2n+1)!}\prod_{i=0}^{n}\left(t^2-\left(i+\frac{1}{2}\right)^2\right)\right|\end{aligned}$$

Thấy rằng 2 công thức sai số trên bằng nhau, nên công thức sai số cho nội suy Bessel với $2n+2$ điểm dữ liệu là:

$$\left|R\left(x_0+\frac{h}{2}+ht\right)\right| \leq \left|\frac{\Delta^{2n+1}y_{-n}}{(2n+1)!}\prod_{i=0}^{n}\left(t^2-\left(i+\frac{1}{2}\right)^2\right)\right|$$

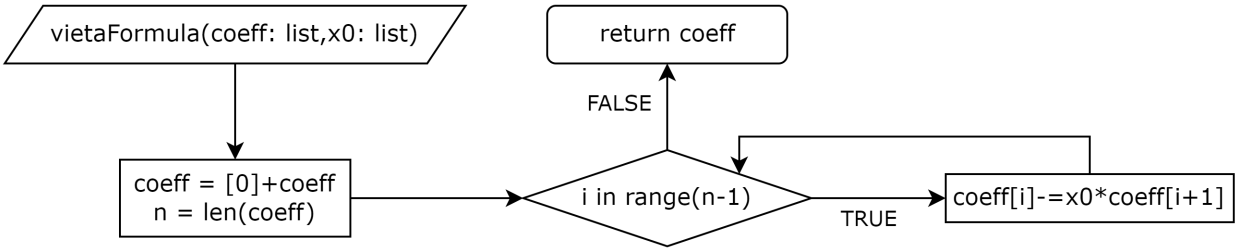
4 Thuật toán

4.1 Các gói chung

1. Gói vietaFormula(coeff: list, x0: float) $\longrightarrow$ list

- **Đầu vào:** List `coeff` và số thực `x0`
- **Đầu ra:** List chứa các hệ số khi nhân đa thức hệ số `coeff` với `(x-x0)` (hệ số đa thức xuất phát từ hệ số tự do)

Sơ đồ thuật toán:



Code:

```python
def vietaFormula(coeff:list,x0:float) -> list:
    coeff = [0] + coeff # thêm 0 vào đầu list hệ số
                        # tương đương nhân đa thức ban đầu với x
    n = len(coeff)
    for i in range(n - 1):
        coeff[i] -= x0*coeff[i + 1] # hệ số khi nhân ra bằng khai triển đa thức
    return coeff
```

Ví dụ:

```python
print(vietaFormula([2, -3, 1], 3))
>>> [-6, 11, -6, 1]
```

Do `[2, -3, 1]` là hệ số của $(2 - 3x + x^2)$ nên ta xét phép nhân đa thức:

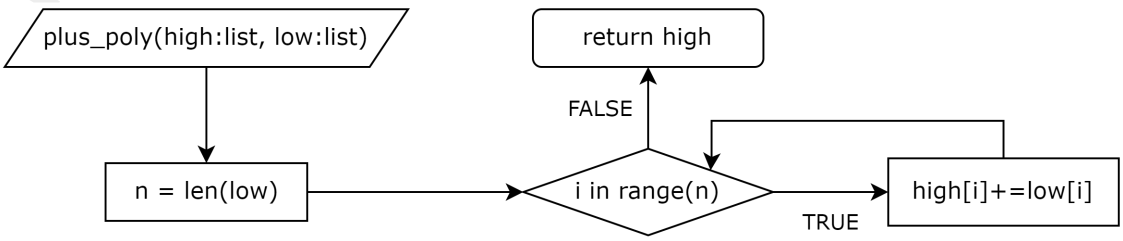
$$(2 - 3x + x^2)(x - 3) = -6 + 11x - 6x^2 + x^3$$

$\Longrightarrow$ Đầu ra sẽ là `[-6, 11, -6, 1]`

2. Gói plus_poly(high: list,low: list) $\longrightarrow$ list

- **Đầu vào:** List `high` và list `low`
- **Đầu ra:** Hệ số đa thức tổng nếu coi `high` là các hệ số đa thức cấp cao hơn

Sơ đồ thuật toán:



Code:

```python
def plus_poly(high: list,low: list) -> list:
    n = len(low)
    for i in range(n):
        high[i] += low[i] # cộng các hệ số từ bậc tự do
    return high
```

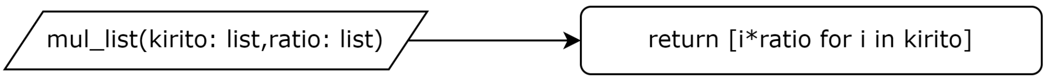
Ví dụ:

```python
print(plus_poly([1, 4, 6, 7],[2, 4, 3])
>>> [3, 8, 9, 7]
```

3. Gói mul_list(kirito: list,ratio: float) $\longrightarrow$ list:

- **Đầu vào:** List `kirito` và số thực `ratio`
- **Đầu ra:** List `kirito` sau khi nhân mỗi phần tử với `ratio`

Sơ đồ thuật toán:



Code:

```python
def mul_list(kirito: list,ratio: float) -> list:
return [i*ratio for i in kirito]
```

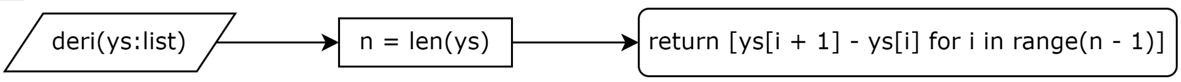
Ví dụ:

```python
print(mul_list([1,2,3,4],1111)
>>> [1111, 2222, 3333, 4444]
```

4. Gói deri(ys: list) $\longrightarrow$ list

- **Đầu vào:** List `ys`
- **Đầu ra:** Sai phân cấp tiếp theo của `ys`

Sơ đồ thuật toán:



Code:

```python
def deri(ys:list):
    n = len(ys)
    return [ys[i+1]-ys[i] for i in range(n-1)]
```

Ví dụ:

```python
print(deri([1,4,6,9])
>>> [3, 2, 3]
```

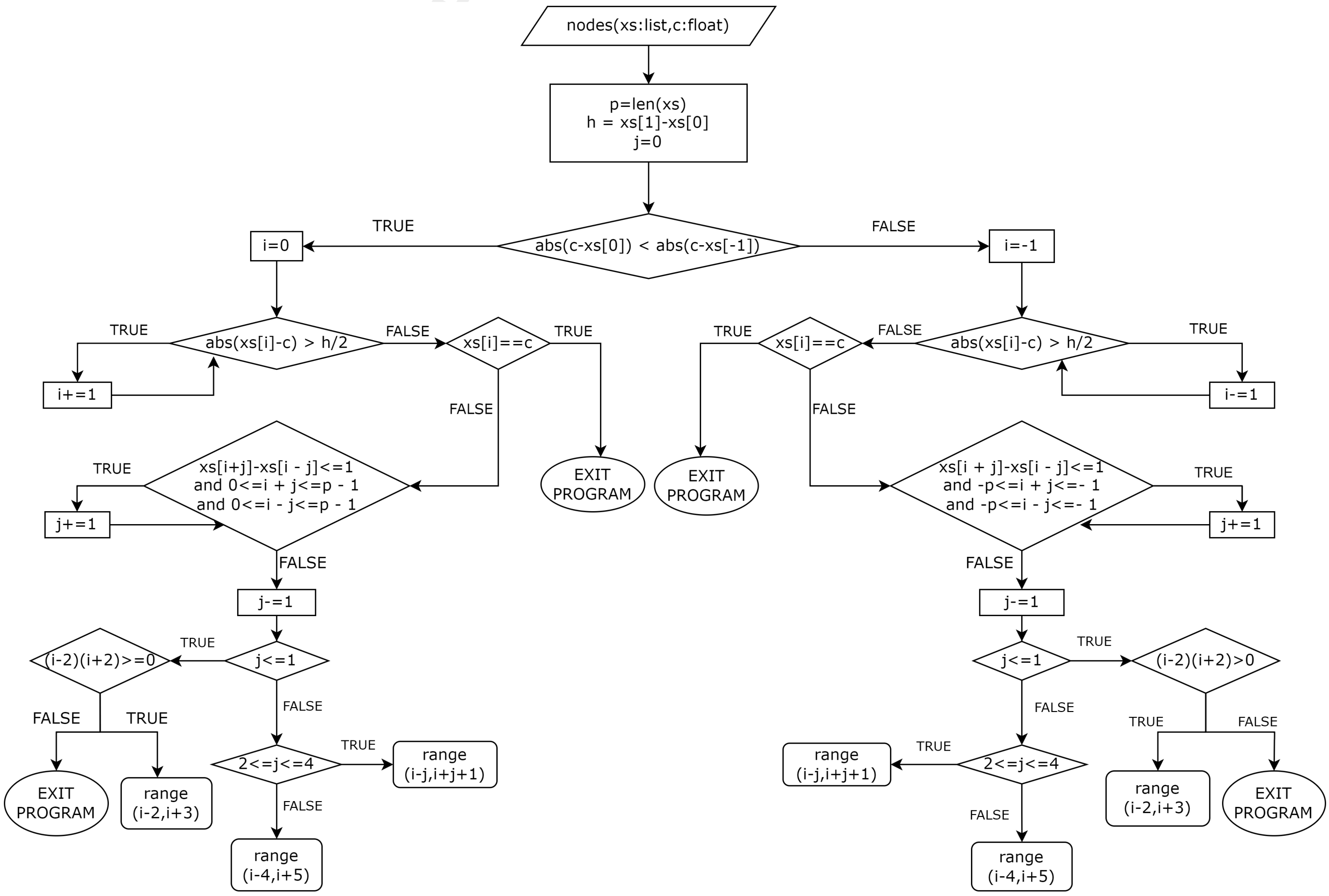
Gói `deri(ys)` đơn giản là lấy số sau trừ số trước, nên với input `[1,4,6,9]`
$\Longrightarrow$ Output sẽ là `[4-1, 6-4, 9-6]`=`[3, 2, 3]`

4.2 Thuật toán nội suy Stirling

1. Gói nodes(xs: list,c: float) $\longrightarrow$ list

- **Đầu vào:** List xs và số thực c
- **Đầu ra:** List index những vị trí cần thiết cho nội suy Stirling

Sơ đồ thuật toán:



Vì thuật toán của gói này khá dài, nên ta sẽ phân tích từng phần nhỏ:

Bước 1: Gán các giá trị cần thiết

```
def nodes(xs:list,c:float):
    p = len(xs) # độ dài của list xs
    h = xs[1]-xs[0] # khoảng cách giữa 2 x liên tiếp
    j = 0 # khi kết thúc gói, 2j+1 sẽ là số mốc nội suy sẽ lấy
```

Bước 2: Tìm vị trí của x_0

Ý tưởng ở đây là nếu `c` gần `x` đầu hơn ta sẽ đếm từ đầu để tìm vị trí x_0 (gán `i = 0`). Ở công thức Stirling, mốc nội suy trung tâm sẽ chọn ở điểm gần `c` nhất nên ta sẽ tăng `i` lên 1 đơn vị cho đến khi khoảng cách từ `c` đến `xs[i]` không vượt quá $\frac{h}{2}$.

$\Longrightarrow$ Kết thúc vòng lặp `while`, giá trị của `i` sẽ là vị trí của x_0.

Nếu `xs[i]` **đúng bằng** `c`, đồng nghĩa điểm người nhập cần tĩnh đã có trong bộ dữ liệu, ta sẽ thoát chương trình bằng hàm `exit`.

```
    if abs(c-xs[0]) < abs(c-xs[-1]):
        i = 0
    while abs(xs[i]-c) > h/2:
        i+=1
    if xs[i]==c:
        exit(colored(f'VALUE KNOWN AT {c} IS: {ys[i]}','yellow'))
        # dòng exit này sử dụng list ys ngoài chương trình
        # nếu muốn gói đóng kín, khai báo biến ys: líst vào phần input
```

Bước 3: Tìm số mốc nội suy sẽ trích xung quanh `i`

Để tìm số mốc nội suy xung quanh `xs[i]`, ta đi tìm `j`, chính là số điểm ở mỗi bên của x_0, ban đầu $j = 0$. Trường hợp lý tưởng ta muốn số mốc nội suy không quá ít (≥ 5), không quá nhiều (≤ 9) và khoảng nội suy ≤ 1 (tức $x_{i+j} - x_{i-j} \leq 1$), đây là cách ta tìm j. Tuy nhiên ta không muốn quá 9 hay dưới 5 ta có các trường hợp sau:

$$\begin{cases} 0 \leq j \leq 1: & \text{Lấy 5 mốc nội suy xung quanh i (nếu có đủ), kết thúc gói} \\ 2 \leq j \leq 4: & \text{Lấy j điểm nội suy mỗi bên của i và trả về kết quả, kết thúc gói} \\ j \geq 5: & \text{Lấy 9 điểm nội suy xung quanh i} \end{cases}$$

Code:

```
    while xs[i+j]-xs[i-j]<=1 and 0<=i+j<=p-1 and 0<=i-j<=p-1:
        j+=1
    j-=1
```

Ta thực hiện cộng `j` 1 đơn vị cho đến khi vượt biên hoặc khi khoảng nội suy > 1, sau đó ta lùi 1 bước, để có thể nhận giá trị 'lớn nhất thỏa mãn' thay vì 'đầu tiên không thỏa mãn'

- **Trường hợp 1:** $j \leq 1$

Ta kiểm tra xung quanh i có đủ 5 mốc nội suy hay không, nếu có thì lấy những vị trí đó, thoát gói, nếu không ta thoát toàn bộ chương trình do mô hình nội suy quá thô sơ.

```
        if j<=1:
            if (i-2)*(i+2)>=0: # kiểm tra xung quanh i có đủ 5 MNS không
                return range(i-2,i+3) # trả về list 5 vị trí xung quanh
            else:
                exit('Too few usable data points to perform Stirling Interpolation')
                #Thoát toàn bộ chương trình
```

- **Trường hợp 2:** $2 \leq j \leq 4$

Lúc này j đã nằm trong vùng lý tưởng, ta trả về vị trí $2j+1$ mốc nội suy xung quanh i

```
        elif 2<=j<=4:
            return range(i-j,i+j+1)
```

- **Trường hợp 3:** $j \geq 5$

Bây giờ j lại quá lớn đồng nghĩa nếu ta lấy như ở TH2 đồng nghĩa phép nội suy thực hiện ít nhất với 11 mốc nội suy, rất hao phí số lượng tính toán mà độ chính xác chẳng tăng được bao nhiêu.
Vì thế ta chỉ lấy 9 điểm xung quanh i (không cần kiểm tra như ở TH1 vì vùng có thể lấy (từ $i-j$ đến $i+j$) còn rộng hơn vùng ta muốn lấy (từ $i-4$ đến $i+4$)).

```
        else:
            return range(i-4,i+5) # 9 điểm xung quanh i
```

Trường hợp c gần `xs[-1]` (x cuối) hơn là `xs[0]` (x đầu) hoàn toàn tương tự. Code:

```
    else:
        i=-1
        while abs(xs[i]-c) > h/2:
            i-=1
        if xs[i]==c:
            exit(colored(f'VALUE KNOWN AT {c} IS: {ys[i]}','yellow'))
        while xs[i+j]-xs[i-j]<=1 and -p<=i+j<=-1 and -p<=i-j<=-1:
            j+=1
        j-=1
        if j<=1:
            if (i-2)*(i+2)>0:
                return range(i-2,i+3) # min 5 nodes
            else:
```

```
            exit('Too few usable data points to perform Stirling Interpolation')
    elif 2<=j<=4:
        return range(i-j,i+j+1)
    else:
        return range(i-4,i+5) # max 9 nodes
```

$\Longrightarrow$ **Kết luận gói** nodes(): $\begin{cases} \text{Ưu tiên 1: khoảng nội suy} \leq 1, \text{ tức } x_{cuối} - x_{đầu} \leq 1 \\ \text{Ưu tiên 2: Ít nhất 5 và nhiều nhất 9 mốc nội suy} \\ \text{Không đủ 5 mốc nội suy: Thoát chương trình tổng} \end{cases}$

2. Gói Stirling_Overhaul(xs: list,ys: list,x: float)

- **Đầu vào:** List xs, list ys và số thực x

- **Đầu ra:**

  1. Giá trị xấp xỉ $f(x)$ bằng nội suy Stirling
  2. Hệ số từ bậc tự do của đa thức nội suy $P(x_0 + ht)$ theo biến t
  3. Sai số của công thức Stirling

Tại sao chúng ta không tìm đa thức nội suy theo biến x? Câu trả lời là do khối lượng tính toán quá lớn, trong khi sự phụ thuộc tuyến tính giữa biến t và x giúp việc đổi biến từ x sang t vô cùng đơn giản.

Thuật toán của gói này khá phức tạp nên sẽ được giải thích bằng lời:

Bước 1: Gán các giá trị cần thiết

```
def Stirling_overhaul(xs:list,ys:list,x:float):
    ys = [ys[i] for i in nodes(xs,x)] ### lấy các mốc cần thiết
    xs = [xs[i] for i in nodes(xs,x)] # NOTE: buộc phải lấy nodes của ys trước, xs sau
    print(colored(f'Required nodes for x = {x} and corresponding values:','yellow'))
    for m,i in enumerate(xs):
        print([xs[m],ys[m]]) # in ra các điểm đã lấy
    h = xs[1]-xs[0] # khoảng cách giữa 2 mốc nội suy liên tiếp
    n = int((len(xs)-1)/2) # là giá trị n trong công thức
    x0 = xs[n] # why not? :v It's comprehensible
    y0 = ys[n]
    t = (x-x0)/h # là giá trị t trong lý thuyết
    coeff = [1] # sẽ được giải thích sau
    t_coeffs = [y0] # kết thúc sẽ chứa toàn bộ hệ số theo ẩn t
    even_coeff = [] # kết thúc sẽ chứa toàn bộ hs bậc chẵn trừ hs tự do
    odd_coeff = [] # kết thúc sẽ chữa toàn bộ hs bậc lẻ
```

Lưu ý: `xs[n]` trong chương trình chính là x_0 trong lý thuyết, đồng nghĩa `xs[n-k]` và `ys[n-k]` trong chương trình sẽ lần lượt chính là x_{-k} và y_{-k} trong lý thuyết.

Bước 2: Tìm `odd_coeff` và `even_coeff`

Ta viết lại công thức Stirling:

$$\begin{aligned}
P(x_0+ht) = {} & y_0 + t\frac{\Delta y_0 + \Delta y_{-1}}{2} + \frac{t^2}{2!}\Delta^2 y_{-1} + \frac{(t+1)t(t-1)}{3!}\frac{\Delta^3 y_{-1} + \Delta^3 y_{-2}}{2} + \dots \\
& + \frac{t(t^2-1^2)(t^2-2^2)\dots(t^2-(n-1)^2)}{(2n-1)!}\frac{\Delta^{2n-1}y_{-(n-1)} + \Delta^{2n-1}y_{-n}}{2} \\
& + \frac{t^2(t^2-1^2)(t^2-2^2)\dots(t^2-(n-1)^2)}{(2n)!}\Delta^{2n}y_{-n} \\
= {} & y_0 + \left(\frac{\Delta^2 y_{-1}}{2!}t^2 + \frac{\Delta^4 y_{-2}}{4!}t^2(t^2-1) + \dots + \frac{\Delta^{2n}y_{-n}}{(2n)!}\prod_{i=0}^{n-1}\left(t^2-i^2\right)\right) \\
& + \frac{1}{t}\left(\frac{\Delta y_0 + \Delta y_{-1}}{2}t^2 + \dots + \frac{\Delta^{2n-1}y_{-(n-1)} + \Delta^{2n-1}y_{-n}}{2.(2n-1)!}\prod_{i=0}^{n-1}\left(t^2-i^2\right)\right) \qquad (1)
\end{aligned}$$

Để thuận tiện, ta đặt

$$a_i = \frac{\Delta^{2i-1}y_{-(i-1)} + \Delta^{2i-1}y_{-i}}{2.(2i-1)!} \quad , b_i = \frac{\Delta^{2i}y_{-i}}{(2i)!} \quad , P_i(t) = t\prod_{j=1}^{i-1}\left(t^2-j^2\right) \quad , i = \overline{1,n}$$

Từ cách viết của công thức (1), có thể thấy ta cần tách riêng hệ số bậc chẵn và bậc lẻ (chính vì thế nên trong phần gán giá trị mới có `odd_coeff` và `even_coeff` riêng). Bảng sau sẽ chỉ ra các hệ số bậc lẻ ta cần xử lý:

	t	t^3	t^5	t^7	**Multiplier**
$P_1(t)$	1				a_1
$P_2(t)$	-1	1			a_2
$P_3(t)$	4	-5	1		a_3
$\vdots$	$\vdots$			$\ddots$	$\vdots$
$P_n(t)$					a_n

Bảng trên liệt kê các hệ số bậc lẻ của 1 số $P_i(t)$ đầu tiên khi chưa nhân với cột **Multiplier**.

Bắt đầu vòng lặp để tìm `odd_coeff` và `even_coeff`:

```
fact = 1 # giai thừa, khai báo riêng để giảm KL tính toán
for i in range(n):
    ys = deri(ys[:]) # lấy cấp sai phân tiếp theo

    fact *= 2*i+1
    a_i = (ys[n-i]+ys[n-i-1])/(2*fact) # chính là a_i trong phân tích gói
```

Ta sẽ sử dụng coeff để lưu các hệ số của $P_i(t)$ sau mỗi lần lặp, ghi đè lên coeff cũ (cách chuyển sẽ được đề cập sau).

Với lần lặp đầu tiên, coeff = [1] chính là hệ số của $P_1(t)$ (các lần lặp sau tương tự), khi đó ta muốn odd_coeff sẽ chỉ là $[a_1]$, ta sử dụng gói mul_list() để nhân coeff với a_1, rồi cộng list đó vào odd_coeff cũ (ban đầu đang trống) bằng gói plus_poly().

Toàn bộ giải thích trên trong chương trình chỉ ngắn gọn 1 dòng code:

```
odd_coeff = plus_poly(mul_list(coeff,a_i),odd_coeff)
```

Ta làm tương tự với even_coeff. Code:

```
    ys = deri(ys[:]) # lấy cấp sai phân tiếp theo

    fact *= 2*i+2
    b_i = ys[n-i-1]/fact # chính là b_i trong phân tích gói
    even_coeff = plus_poly(mul_list(coeff,b_i),even_coeff)
```

Để có hệ số của $P_2(t)$, ta dùng list 2 phần tử để lưu, và suy ra $P_2(t)$ từ $P_1(t)$ bằng gói vietaFormula() ở phần **Các gói chung**. Ví dụ:

coeff = vietaFormula(coeff,1)

$\Longrightarrow$ coeff ban đầu chỉ là [1], nhưng sau dòng trên sẽ trở thành [-1,1]. Tương tự ta suy ra hệ số của $P_3(t)$ từ $P_2(t)$:

coeff = vietaFormula(coeff,4)

Khi đó coeff sẽ thành [4,-5,1] .Tổng quát, ta có dòng lệnh cuối mỗi vòng lặp:

```
coeff = vietaFormula(coeff,(i+1)**2)
```

Kết thúc vòng lặp trên, ta sẽ có odd_coeff chứa các hệ số bậc lẻ của t, even_coeff chứa các hệ số bậc chẵn của t trong đa thức nội suy.

Bước 3: Tính giá trị xấp xỉ $f(x)$

Để đơn giản trong tính toán, ta sẽ chuyển odd_coeff và even_coeff thành dạng ma trận hàng bằng hàm numpy.array()

```
odd_coeff = np.array(odd_coeff)
even_coeff = np.array(even_coeff)
```

Sau đó tạo 1 vector cột `t_vec` là

$$\begin{pmatrix} t^2 \\ t^4 \\ \vdots \\ t^{2n} \end{pmatrix}$$

(**NOTE:** Ta chỉ lấy từ t^2 chứ không phải hệ số tự do vì đã biết hệ số tự do là y_0 khi thay $t = 0$ vào công thức Stirling)

Code tạo `t_vec`:

```
t_sq = t**2 # đặt để giảm KL tính toán sau này
q = 1
t_vec = []
for i in range(1,n+1):
    q *= t_sq # số tiếp theo của t_vec gấp t_sq lần số trước
    t_vec.append([q]) # thêm q vào t_vec
t_vec = np.array(t_vec) # chuyển thành dạng vector cột
```

Từ cách đặt này ta có thể viết giá trị cần xấp xỉ đơn giản bằng:

$$f(x_0 + ht) \approx \texttt{y0 + even\_coeff @ t\_vec + (odd\_coeff @ t\_vec)/t}$$

(Dấu @ là phép nhân ma trận trong python)

Code in giá trị xấp xỉ:

```
print(colored(f'Stirling interpolation deduces value at x = {x}: ','yellow'))
print(y0 + even_coeff @ t_vec + (odd_coeff @ t_vec) / t)
```

Bước 4: Tìm hệ số của $P(x_0 + ht)$ theo t:

Ta đã có `odd_coeff` và `even_coeff`, bây giờ chỉ cần xâu chúng lại, ta sẽ được `t_coeffs`. Code:

```
for j in range(n):
    t_coeffs.append(odd_coeff[j])
    t_coeffs.append(even_coeff[j])
print(f'The coefficients with respect to t = (x-{x0})/{h} (increasing power) are:')
print(t_coeffs) # in ra kết quả t_coeffs
```

Bước 5: Tính toán sai số

Ta viết lại công thức sai số của nội suy Stirling:

$$|R(x)| \leq \left| \frac{\Delta^{2n} y_{-n}}{(2n)!} .t \prod_{i=1}^{n} (t^2 - i^2) \right|$$

Ta sử dụng hàm prod() để tính toán trực tiếp công thức trên. Code:

```
error = abs(ys[0]/fact*t*prod((t_sq-i**2 for i in range(1,n+1))))
print('The error expectation shall not exceed: ')
print(error) # in ra kết quả
```

3. Chương trình chính

Ta chạy chương trình chính :hàm main().

Nếu điểm cần tính không nằm trong khoảng nội suy (còn được gọi là ngoại suy), ta thoát chương trình. Code:

```
def main():
    if not xs[0] <= start <= xs[-1]: # điểm cần tính nằm ngoài khoảng nội suy
        exit(colored('EXTRAPOLATION NOT INCLUDED','red')) # thoát chương trình tổng
    Stirling_overhaul(xs,ys,start)
main()
```

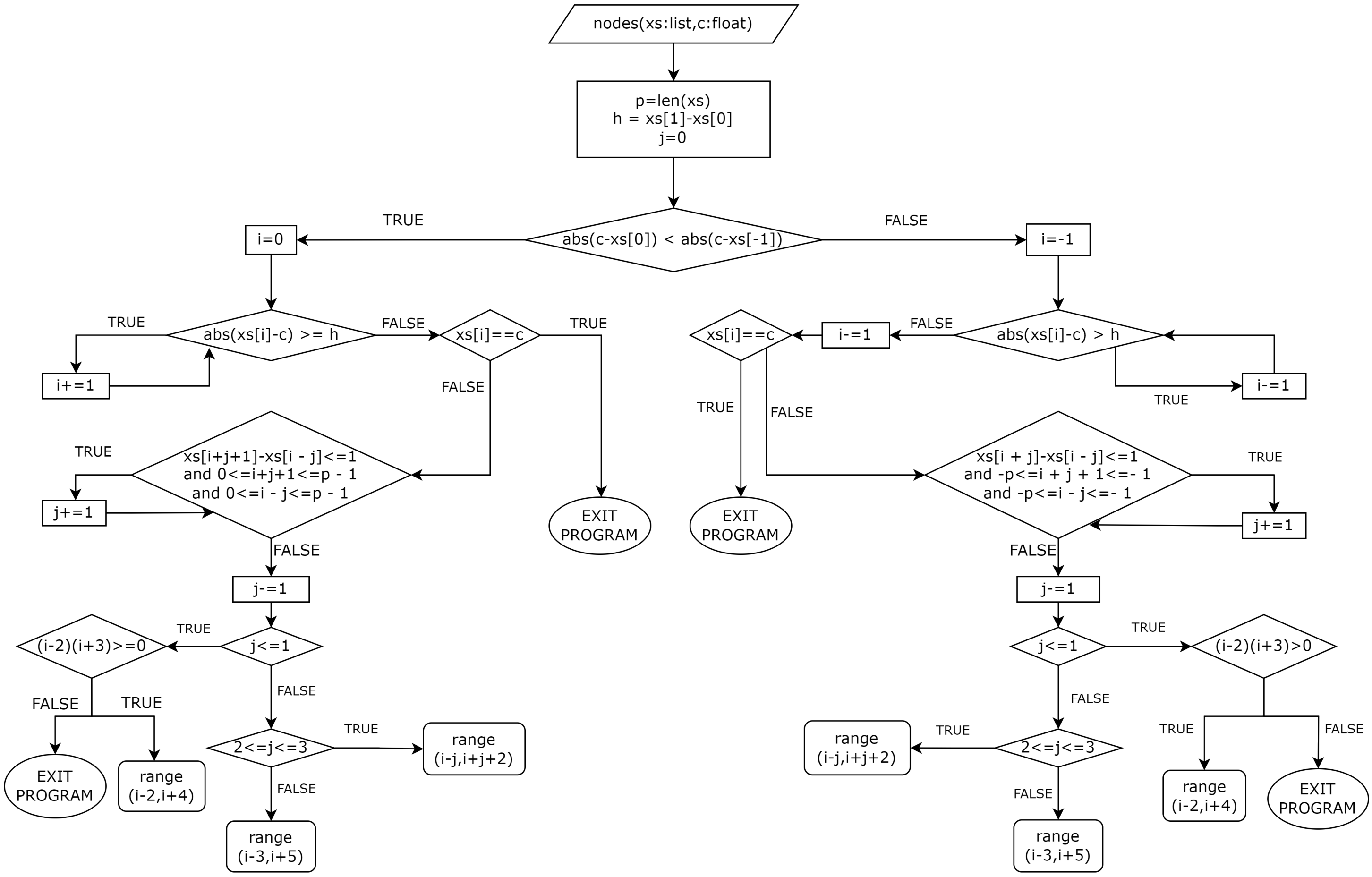
4.3 Thuật toán nội suy Bessel

1. Gói nodes(xs: list,c: float) $\longrightarrow$ list

- **Đầu vào:** List xs và số thực c

- **Đầu ra:** List index những vị trí cần thiết cho nội suy Stirling

Gói nodes() của Bessel khá giống với Stirling, nên ta sẽ chỉ chỉ ra những điểm khác.
Sơ đồ thuật toán:



Nhìn vào nhánh trái, ở khối lệnh kiểm tra logic đầu tiên, ta so sánh khoảng cách từ `xs[i]` đến c với h thay vì $\dfrac{h}{2}$, vì ta tìm vị trí của x_0 và x_1 với $x_0 \leq c < x_1$. $\Longrightarrow$ Kết thúc vòng lặp đó, `i` sẽ là vị trí của x_0

Phần còn lại của nhánh trái diễn ra tương tự như Stirling, nhưng ta chặn dưới số mốc nội suy là 6 và chặn trên là 8 (vì số mốc nội suy cần thiết cho nội suy Bessel là số chẵn).

Nhánh phải có 1 phần nhỏ khác với nhánh trái, là khi thoát vòng lặp `i-=1`, ta phải thực hiện `i-=1` một lần nữa vì nếu không vị trí của `i` sẽ là tại x_1 chứ không phải x_0.

Phần còn lại hoàn toàn tương tự nhánh trái

2. Gói Bessel_Overhaul(xs: list,ys: list,x: float)

- **Đầu vào:** List `xs`, list `ys` và số thực `x`

- **Đầu ra:**

1. Giá trị xấp xỉ $f(x)$ bằng nội suy Bessel
2. Hệ số từ bậc tự do của đa thức nội suy $P(x_0 + ht)$ theo biến t
3. Sai số của công thức Bessel

Bước 1: Gán các giá trị cần thiết:

```python
def Bessel_overhaul(xs:list,ys:list,x:float):
    ys = [ys[i] for i in nodes(xs,x)] ### lấy những mốc cần thiết
    xs = [xs[i] for i in nodes(xs,x)]
    print(f'Required nodes for x = {x} and corresponding values:')
    for m,i in enumerate(xs):
        print([xs[m],ys[m]])
    h = xs[1]-xs[0]
    n = int(len(xs)/2-1) # giá trị n trong lý thuyết
    x0 = xs[n]
    y0 = ys[n]
    t = (x-xs[n])/h-1/2
    coeff=[-0.25,1]
    t_coeffs = [] # sẽ chứa toàn bộ hệ số của đa thức nội suy theo t
    even_coeff = [(ys[n]+ys[n+1])/2] # hệ số bậc chẵn đầu tiên
    ys = deri(ys[:])
    odd_coeff = [ys[n]] # hệ số bậc lẻ đầu tiên
```

Bước 2: Tìm `odd_coeff` và `even_coeff`

Chúng ta phải tách riêng lần đầu lấy hệ số chẵn và lẻ cho `odd_coeff` và `even_coeff` vì trong vòng lặp sẽ chuyển `ys = deri(ys[:])` trước mỗi lần cho giá trị vào `odd_coeff` hay `even_coeff`.
Code:

```python
fact = 1
for i in range(1,n+1):
    ys = deri(ys[:]) # lấy sai phân cấp tiếp theo

    fact *= 2*i
    diff = (ys[n-i]+ys[n-i+1])/(2*fact)
    even_coeff = plus_poly(mul_list(coeff,diff),even_coeff)
    ys = deri(ys[:])

    fact *= 2*i+1
    diff = ys[n-i]/fact
    odd_coeff = plus_poly(mul_list(coeff,diff),odd_coeff)
    coeff = vietaFormula(coeff,(i+1/2)**2)
```

Phần còn lại tương tự nội suy Stirling, tuy nhiên `t_vec` không thể bỏ qua hệ số tự do nữa vì ở đây ta chưa biết!

Ngoài ra, giá trị đầu của `coeff` không phải là [1] nữa là mà là [-0.25, 1], đồng nghĩa ta đã "làm hộ"lần lặp đầu tiên cho `coeff`

$\Longrightarrow$ Chính vì thế trong vòng lặp giá trị `i` xuất phát từ 1 thay vì 0 như ở thuật toán của Stirling.

Bước 3: Tính giá trị xấp xỉ $f(x)$ và `t_coeffs`:

Bước này tương tự như nội suy Stirling, code:

```
    odd_coeff = np.array(odd_coeff)
    even_coeff = np.array(even_coeff)
    t_sq = t**2
    q = 1
    t_vec = [[1]] # chứa cả hệ số tự do
    for i in range(n):
        q *= t_sq
        t_vec.append([q])
    t_vec = np.array(t_vec)

    print(f'Bessel interpolation deduces value at x = {x}:')
    print(even_coeff @ t_vec + t*(odd_coeff @ t_vec))
    for j in range(n+1):
            t_coeffs.append(even_coeff[j])
            t_coeffs.append(odd_coeff[j])
    print(f'The coefficients with respect to t :')
    print(t_coeffs)
```

Bước 4: Tính toán sai số: Ta tính toán trực tiếp bằng hàm `prod()` cho công thức sai số của Bessel:

$$\left|R\left(x_0+\frac{h}{2}+ht\right)\right| \leq \left|\frac{\Delta^{2n+1}y_{-n}}{(2n+1)!}\prod_{i=0}^{n}\left(t^2-\left(i+\frac{1}{2}\right)^2\right)\right|$$

Code:

```
    error = abs(ys[0]/fact*prod((t_sq-(i+0.5)**2 for i in range(n+1))))
    print(colored('The error expectation shall not exceed: ','yellow'))
    print(error)
```

3. Chương trình chính

Hàm `main()` của thuật toán nội suy Bessel tương tự với thuật toán nội suy Stirling.

Code:

```
    def main():
        if not xs[0] <= start <= xs[-1]:
            exit(colored('EXTRAPOLATION NOT INCLUDED','red'))
        Bessel_overhaul(xs,ys,start)
    main()
```

5 Hệ thống ví dụ

Ví dụ 1

2.5	24.145
3.0	22.043
3.5	20.225
4.0	18.644
4.5	17.262
5.0	16.047

Cần tính giá trị tại:

3.9

Kết quả mong đợi:

1. Nội suy Stirling lấy 5 mốc nội suy từ $(3.0, 22.043) \to (5.0, 16.047)$
2. Nội suy Bessel lấy 6 mốc nội suy từ $(2.5, 24.145) \to (5.0, 16.047)$

Chạy với thuật toán nội suy Stirling

```
Stirling_Overhaul(xs,ys,start) # xs,ys là các giá trị input, start là điểm cần xấp xỉ
>>> Required nodes for x = 3.9 and corresponding values:
    [3.0, 22.043]
    [3.5, 20.225]
    [4.0, 18.644]
    [4.5, 17.262]
    [5.0, 16.047]
>>> Stirling interpolation deduces value at x = 3.9:
    [18.9431504]
>>> The coefficients with respect to t :
    [18.644, -1.4756666666666673, 0.09925000000000332,
    -0.005833333333333061, 0.00024999999999992693]
>>> Substituting t = -0.2 : # thay ngược vào hệ số kiểm tra kết quả
    18.943150399999997
>>> The error expectation shall not exceed:
    0.00019007999999994446
```

Chạy với thuật toán nội suy Bessel

```
Bessel_Overhaul(xs,ys,start)
>>> Required nodes for x = 3.9 and corresponding values:
    [2.5, 24.145]
    [3.0, 22.043]
    [3.5, 20.225]
    [4.0, 18.644]
    [4.5, 17.262]
    [5.0, 16.047]
>>> Bessel interpolation deduces value at x = 3.9:
    [18.937903808]
>>> The coefficients with respect to t are:
    [19.399140625, -1.57924322916667, 0.14137500000000305,
    -0.007020833333332366, 0.0002499999999992693, -2.5000000000297008e-05]
>>> Substituting t = 0.3 :
    18.937903807999998
>>> The error expectation shall not exceed:
    5.322240000063233e-05
```

$\Longrightarrow$ **Nhận xét:** Nếu có đủ mốc nội suy, mặc dù khoảng nội suy > 1, nhưng thuật toán vẫn vận hành

Ví dụ 2

10	0.23967
11	0.28060
12	0.31788
13	0.35209
14	0.38368

Cần tính giá trị tại:

13.6

Kết quả mong đợi: Cả 2 phép nội suy đều không hoạt động do quá ít mốc nội suy xung quanh. Chạy thuật toán nội suy Stirling

```
Stirling_Overhaul(xs,ys,start)
>>> TOO FEW USABLE DATA POINTS TO PERFORM STIRLING INTERPOLATION
```

Chạy thuật toán nội suy Bessel

```
Bessel_Overhaul(xs,ys,start)
>>> TOO FEW USABLE DATA POINTS TO PERFORM BESSEL INTERPOLATION
```

$\Longrightarrow$ **Nhận xét:** Nếu quá ít mốc nội suy khả dụng, thuật toán sẽ không vận hành

Ví dụ 3

Ta tạo input từ hàm $f(x) = x^{\frac{5}{3}} + \frac{1}{x}$

2.000	3.674802103936399
2.125	3.982941776349901
2.250	4.30785501306194
2.375	4.648770750927007
2.500	5.005039373300484
2.625	5.376105969434416
2.750	5.7614906458824
2.875	6.160773780052506
3.000	6.573584802489046
3.125	6.999593541705848
3.250	7.438503458897224
3.375	7.890046296296298
3.500	8.353977796816569
3.625	8.830074245352346

Cần tính giá trị tại:

2.88

Kết quả mong đợi:

1. Thuật toán nội suy Stirling lấy 9 điểm từ $(2.375,\ 4.6487) \rightarrow (3.375,\ 7.8900)$
2. Thuât toán nội suy Bessel lấy 8 điểm từ $(2.500,\ 5.0050) \rightarrow (3.375,\ 7.8900)$
3. Sai số cả 2 thuật toán trong tầm ước lượng (giá trị chính xác $f(2.88) = 6.177028804787914$)

Chạy thuật toán nội suy Stirling (trang tiếp theo)

```
Stirling_Overhaul(xs,ys,start)
>>> Required nodes for x = 2.88 and corresponding values:
    [2.375, 4.648770750927007]
    [2.5, 5.005039373300484]
    [2.625, 5.376105969434416]
    [2.75, 5.7614906458824]
    [2.875, 6.160773780052506]
    [3.0, 6.573584802489046]
    [3.125, 6.999593541705848]
    [3.25, 7.438503458897224]
    [3.375, 7.890046296296298]
>>> Stirling interpolation deduces value at x = 2.88:
    [6.177028804792725]
>>> The coefficients with respect to t :
    [6.160773780052506, 0.4061052204774444, 0.006762271208412067,
    -5.80794039440977e-05,1.6703633124568833e-06,-6.26561987861053e-08,
    2.556620602260177e-09, -1.1397858600499631e-10, 4.872109638130706e-12]
>>> Substituting t = 0.04 :
    6.177028804792725
>>> The error expectation shall not exceed:
    1.1199785392557324e-10
```

Kiểm tra sai số: $|\texttt{6.177028804787914 - 6.177028804792725}|$

$$= \texttt{4.811262499515558e-12} < \texttt{1.1199785392557324e-10}$$

$\Longrightarrow$ Sai số của Stirling trong tầm ước lượng

Chạy với thuật toán nội suy Bessel

```
Bessel_Overhaul(xs,ys,start)
>>> Required nodes for x = 2.88 and corresponding values:
    [2.5, 5.005039373300484]
    [2.625, 5.376105969434416]
    [2.75, 5.7614906458824]
    [2.875, 6.160773780052506]
    [3.0, 6.573584802489046]
    [3.125, 6.999593541705848]
    [3.25, 7.438503458897224]
    [3.375, 7.890046296296298]
>>> Bessel interpolation deduces value at x = 2.88:
    [6.177028804765003]
>>> The coefficients with respect to t :
    [6.365509800436791, 0.412824748201697, 0.006677582634633666,
    -5.488916706660856e-05, 1.5222317053893863e-06, -5.555061978838409e-08,
    2.2941146211103497e-09, -9.44901474524735e-11]
>>> Substituting t = -0.46 :
```

```
    6.177028804765003
>>> The error expectation shall not exceed:
    5.376477216319789e-10
```

$$\textit{Kiểm tra sai số:}\quad |\texttt{6.177028804787914 - 6.177028804765003}| = \texttt{2.291056233616 47e-11} < \texttt{5.376477216319789e-10}$$

$\Longrightarrow$ Sai số của Bessel trong tầm ước lượng

$\Longrightarrow$ **Nhận xét:**

1. Dù có rất nhiều mốc nội suy khả dụng, Stirling và Bessel sẽ chỉ lấy 1 lượng nhất định chứ không xét tất cả.
2. Sai số đưa ra trong tầm được ước lượng.

Ví dụ 4

2.5	24.145
3.0	22.043
3.5	20.225
4.0	18.644
4.5	17.262
5.0	16.047

Cần tính giá trị tại:

5.6

Kết quả mong đợi: Cả 2 thuật toán đều không hoạt động do ngoại suy

Chạy thuật toán nội suy Stirling

```
Stirling_Overhaul.main()
>>> EXTRAPOLATION NOT INCLUDED
```

Chạy thuật toán nội suy Bessel

```
Bessel_Overhaul.main()
>>> EXTRAPOLATION NOT INCLUDED
```

$\Longrightarrow$ **Nhận xét:** Nếu điểm cần xấp xỉ nằm ngoài khoảng nội suy, chương trinh sẽ không hoạt động!

6 Khối lượng tính toán của Stirling, Bessel và Gauss I, Gauss II

6.1 Thuật toán Stirling, Bessel

Bây giờ ta so sánh sự tối ưu của Stirling và Bessel so với công thức Gauss I và Gauss II bằng xét khối lượng tính toán

Ta sẽ quan tâm nhiều nhất đến bậc cao nhất và hệ số bậc cao nhất của khôi lượng tính toán đối với n (n là giá trị được định nghĩa trong lý thuyết và thuật toán của mỗi công thức), bởi lẽ khi n đủ lớn, thành phần mũ cao nhất sẽ tăng mạnh nhất!

Ta sẽ xét các dòng lệnh tiêu tốn 1 lượng đa thức bậc 2 của n trong thuật toán nội suy Stirling (tương tự với Bessel)

Giá sử có $2n+1$ mốc nội suy sử dụng cho thuật toán Stirling. Trích lại code:

```
1   for i in range(n):
2
3       ys = deri(ys[:])
4       fact *= 2*i+1
5       a_i = (ys[n-i]+ys[n-i-1])/(2*fact) # chính là a_i trong phân tích gói
6       odd_coeff = plus_poly(mul_list(coeff,a_i),odd_coeff)
7
8       ys = deri(ys[:])
9       fact *= 2*i+2
10      b_i = ys[n-i-1]/fact # chính là b_i trong phân tích gói
11      even_coeff = plus_poly(mul_list(coeff,b_i),even_coeff)
12      coeff = vietaFormula(coeff,(i+1)**2)
```

Code vòng lặp chính

Khi đó các dòng lệnh sau tiêu tốn 1 lượng đa thức bậc 2 của n khối lượng tính toán:

1. Lệnh `ys = deri(ys[:])` ở dòng 3 vs 8:
   List ys ban đầu có 2n+1 giá trị, môi khi lệnh trên thực hiện, giả sử đang có k phần tử, sẽ cần $2k$ phép toán. Bạn đọc có thể dễ thấy từ gói `deri()` ở phần **Các gói chung**. Trích code:

   ```
   def deri(ys:list):
       n = len(ys)
   return [ys[i+1]-ys[i] for i in range(n-1)] # 2n phép tính
   ```

   Mà lệnh `ys = deri(ys[:])` được thực hiện $2n$ lần, mỗi lần giảm 1 đơn vị, nên dòng lệnh này sẽ hao $2n^2$ phép tính

2. Lệnh `odd_coeff = plus_poly(mul_list(coeff,a_i),odd_coeff)` ở dòng 6 (tương tự với `even_coeff` ở dòng 11) của **Code vòng lặp chính**. Trích code gói:

```
def plus_poly(high: list,low: list) -> list:
    n = len(low)
    for i in range(n):
        high[i]+=low[i]
    return high #len(low) phép tính
def mul_list(kirito: list,ratio: float) -> list:
    return [i*ratio for i in kirito] # len(kirito) phép tính
```

Số phần tử của `coeff` chạy từ 1 đến n khi kết thúc vòng lặp. khi `coeff` có k phần tử, khi đó `odd_coeff` có $k-1$ phần tử, và dòng code

`odd_coeff = plus_poly(mul_list(coeff,a_i),odd_coeff)`

sẽ tiêu tốn $2k$ phép tính, tổng phép tính: $\sum_{k=1}^{n} 2k = n(n+1)$

Tuy nhiên ta chỉ quan tâm bậc cao nhất, nên ta coi dòng lệnh trên tiêu hao n^2 phép tính

3. Lệnh `coeff = vietaFormula(coeff,(i+1)**2)` ở cuối mỗi lần lặp. Trích code gói:

```
def vietaFormula(coeff:list,x0:float) -> list:
    coeff = [0] + coeff
    n = len(coeff)
    for i in range(n-1):
        coeff[i] -= x0*coeff[i + 1] # 3n phép tính
    return coeff
```

Số phần tử của `coeff` tăng dần từ 1 đến n trong vòng lặp, nếu `coeff` có k phần tử, dòng lệnh trên qua vòng lặp sẽ tiêu tốn $\sum_{k=1}^{n} 3k = \dfrac{3n(n-1)}{2}$

Tuy nhiên ta chỉ xét bậc cao nhất, nên dòng lệnh sẽ tiêu hao $\dfrac{3n^2}{2}$ phép tính

4. Tương tự `even_coeff = plus_poly(mul_list(coeff,diff),even_coeff)` tiêu hao n^2 phép tính

$\Longrightarrow$ **Code vòng lặp chính** cần tổng cộng $2n^2 + n^2 + \dfrac{3n^2}{2} + n^2 = \dfrac{11n^2}{2}$

Vậy thuật toán nội suy Stirling và Bessel có độ phức tạp tính toán $o\left(\dfrac{11n^2}{2}\right)$

6.2 Thuật toán Gauss I, Gauss II

Giả sử có $2n+1$ điểm dữ liệu, ta sẽ tìm khối lượng tính toán của **nội suy Gauss I**.

Để phân biệt, ta đặt $p = 2n+1$, và tìm với p khi xét khối lượng tính toán của nội suy Gauss I. Áp dụng ý tưởng trên xây dựng **Code vòng lặp chính** cho công thức Gauss I, sẽ nhìn như sau:

```
    fact = 1
    for i in range(1,p):
        ys = deri(ys[:])
        fact *= i
        if i%2!=0:
            multiplier = ys[cen - int((i - 1)/2)]/fact
            t_coeffs = plus_poly(mul_list(coeff,multiplier),t_coeffs)
            coeff = vietaFormula(coeff,int((i + 1)/2))
        else:
            multiplier = ys[cen - int(i/2)]/fact
            t_coeffs = plus_poly(mul_list(coeff,multiplier),t_coeffs)
            coeff = vietaFormula(coeff,-int(i/2))
```

Bạn đọc đếm số lượng tính toán tương tự phần **A.** sẽ suy ra độ phức tạp thuật toán của Gauss I (tương tự với Gauss II) là $o\left(3p^2\right)$.

Mà $p = 2n + 1$, nên độ phức tạp tính toán sẽ là $o(12n^2)$

$\Longrightarrow$ Số lượng tính toán của Stirling và Bessel, $o\left(\dfrac{11n^2}{2}\right)$,giảm đáng kể so với Gauss I và Gauss II, $o(12n^2)$

7 Đánh giá

7.1 Ưu điểm

▪ Thuật toán nội suy Stirling và Bessel giảm đáng kể khối lượng tính toán so với nội suy Gauss I và Gauss II.

▪ Việc chặn số lượng mốc nội suy sẽ lấy của nội suy Stirling và Bessel ($5 \leq$ *Stirling* $\leq 9,\ 6 \leq$ *Bessel* ≤ 8) đảm bảo khối lượng tính toán không quá lớn nhưng vẫn duy trì sai số ổn định

▪ Mặc dù không tính đa thức nội suy theo biến x, nhưng nhờ sự phụ thuộc tuyến tính của x và t, nên những hàm đạo hàm, tích phân có thể dễ dàng đổi biến theo t

7.2 Nhược điểm

▪ Nội suy Stirling và Bessel không xử lý trường hợp mốc không cách đều.

▪ Không hữu dụng khi điểm cần xáp xỉ gần biên.

▪ Cũng giống như những phương pháp nội suy đa thức khác, Stirling và Bessel sẽ cho ra kết quả không ổn định (sai số đáng kể) khi input không biến thiên đơn điệu.

8 Tài liệu tham khảo

[1] Lê Trọng Vinh. *Giáo trình Giải tích số*. Nhà xuất bản Khoa học và Kỹ thuật. 2007