

# Metodi Computazionali della Fisica

Iman Rosignoli

February 2023

# Contents

<b>1 Lezione 1</b>	<b>6</b>
1.1 Introduzione al corso . . . . .	6
1.2 Rappresentazione numerica . . . . .	6
1.3 L'esame . . . . .	8
1.4 Il problema delle equazioni di II grado . . . . .	8
1.4.1 Algoritmo in Fortran . . . . .	8
<b>2 Lezione 2</b>	<b>10</b>
2.1 Introduzione alla lezione . . . . .	10
2.2 Successione di potenze . . . . .	10
2.2.1 Algoritmo in Fortran . . . . .	10
2.2.2 Riflessione analitica . . . . .	11
2.3 Algoritmo della derivazione . . . . .	12
<b>3 Lezione 3</b>	<b>15</b>
3.1 L'algoritmo della derivazione numerica a 5 punti . . . . .	15
3.1.1 Derivata seconda . . . . .	16
3.1.2 Algoritmo in Fortran della derivata prima . . . . .	16
3.2 Algoritmo di derivazione con funzione non simmetrica . . . . .	18
<b>4 Lezione 4</b>	<b>21</b>
4.1 Implementazione algoritmo della derivata seconda . . . . .	21
4.1.1 Derivata seconda a cinque punti . . . . .	21
4.1.2 Implementazione algoritmo derivata seconda . . . . .	22
4.1.3 Algoritmo attraverso il flesso . . . . .	23
4.1.4 Parità funzione $\sin x$ . . . . .	24
4.2 Discorso su $f''$ nei pressi dello zero con precisione maggiore . . . . .	24
4.2.1 Problema di $\sin x/x$ . . . . .	27
<b>5 Lezione 5</b>	<b>28</b>
5.1 Integrazione . . . . .	28
5.2 Integrazione per algoritmi deterministici . . . . .	28
5.2.1 Algoritmo del rettangolo . . . . .	29
5.2.2 Algoritmo del trapezio . . . . .	31
5.3 Algoritmo di Sympson . . . . .	33
5.3.1 Stima dell'errore C-S . . . . .	35
5.3.2 Struttura algoritmo . . . . .	35
5.4 Algoritmi adattivi . . . . .	35

<b>6 Lezione 6</b>	<b>36</b>
6.1 Algoritmo d'integrazione del trapezio . . . . .	36
6.1.1 Input di $N = 3$ . . . . .	37
6.1.2 Stabilità dell'algoritmo . . . . .	38
6.2 Richiesta di precisione nell'algoritmo . . . . .	38
6.3 Algoritmo di integrazione di Sympson . . . . .	39
6.3.1 Driver integration . . . . .	39
<b>7 Lezione 7</b>	<b>42</b>
7.1 Integrali impropri . . . . .	42
7.1.1 Ridefinizione funzione . . . . .	43
7.1.2 Bassa densità di punti . . . . .	44
7.1.3 Impostiamo un estremo superiore ragionevole . . . . .	44
7.2 Valore Principale dell'integrale . . . . .	46
<b>8 Lezione 8</b>	<b>48</b>
8.1 Integrali PV . . . . .	48
8.1.1 Prima run . . . . .	48
8.1.2 Si riduce $\delta$ . . . . .	49
8.2 Estremi di integrazione coerenti . . . . .	49
8.3 Routine dedicata al PV . . . . .	50
<b>9 Lezione 9</b>	<b>51</b>
9.1 Metodo delle variabili antitetiche . . . . .	51
9.1.1 Mapping lineare pseudo simmetrico . . . . .	52
9.1.2 Implementazione su intervallo simmetrico . . . . .	52
9.1.3 Implementazione su intervallo asimmetrico . . . . .	53
9.2 Generalizzazione del metodo agli intervalli asimmetrici . . . . .	54
9.3 Algoritmi di Gauss . . . . .	54
9.3.1 Zeri dei polinomi . . . . .	55
9.3.2 Polinomi di interpolazione di Lagrange . . . . .	56
<b>10 Lezione 10</b>	<b>57</b>
10.1 Algoritmi di Quadratura di Gauss . . . . .	57
10.1.1 Quadratura di Gauss-Laguerre . . . . .	57
10.1.2 Quadratura di Gauss-Legendre . . . . .	58
10.1.3 Quadratura di Gauss-Hermite . . . . .	58
10.1.4 Quadrature di Gauss-Chebyshev . . . . .	59
10.2 Ricerca degli zeri . . . . .	60
10.2.1 Algoritmo di bisezione . . . . .	60
10.2.2 Metodo di Newton-Raphson . . . . .	61
10.2.3 Metodo della secante . . . . .	61
10.2.4 Algoritmo della Falsa Posizione . . . . .	62
10.2.5 Algoritmo quadratico di Rosignoli . . . . .	62
<b>11 Lezione 11</b>	<b>64</b>
11.1 Implementazione della ricerca degli zeri . . . . .	64
11.1.1 Bisezione . . . . .	64
11.1.2 Algoritmo di Newton-Raphson . . . . .	64
11.1.3 Confronto tra i vari algoritmi . . . . .	64
11.2 Lunghezza di picco . . . . .	65

11.2.1	Estrapolazione numerica . . . . .	66
11.2.2	Curiosità . . . . .	67
<b>12 Lezione 12</b>		<b>68</b>
12.1	Applicazioni ricerca degli zeri . . . . .	68
12.1.1	Polinomi di Gauss-Legendre . . . . .	68
12.2	Metodo Monte Carlo . . . . .	69
12.2.1	Ago di Buffon . . . . .	69
12.3	Generazione di numeri pseudo random . . . . .	70
12.3.1	Algoritmo di Von Neumann . . . . .	71
12.3.2	Algoritmo della congruenza lineare . . . . .	71
12.3.3	Generatore built in . . . . .	72
12.3.4	Particolarità N-R. . . . .	72
<b>13 Lezione 13</b>		<b>74</b>
13.1	Metodo Monte Carlo . . . . .	74
13.2	Integrazione mediante MC . . . . .	74
13.2.1	Distribuzione di probabilità . . . . .	74
13.2.2	Momenti del valor medio ricavato con $N$ punti . . . . .	75
13.2.3	Utilizzo del MC in più dimensioni . . . . .	76
<b>14 Lezione 13</b>		<b>77</b>
14.1	Integrazione nidificata . . . . .	77
14.1.1	Confronto con la convergenza di MC . . . . .	77
14.2	Implementazione metodo MC . . . . .	78
14.2.1	Stima di $\pi$ . . . . .	78
14.3	Algoritmi di riduzione della varianza . . . . .	82
<b>15 Lezione 15</b>		<b>83</b>
15.1	Metodo HoM . . . . .	83
15.1.1	Derivata della $f$ sui bin . . . . .	83
15.1.2	Distribuzione punti HoM . . . . .	84
15.1.3	Ipercubo di 20 dimensioni . . . . .	84
15.2	Esercizio di QED . . . . .	85
15.2.1	Lavorio Analitico . . . . .	85
15.2.2	Script . . . . .	86
15.2.3	HoM . . . . .	87
<b>16 Lezione 16</b>		<b>88</b>
16.1	Simulazione di collisione . . . . .	88
16.1.1	Run dello script . . . . .	88
16.1.2	Output del programma . . . . .	88
16.1.3	Analisi dell'output . . . . .	89
16.1.4	File output dell'analisi . . . . .	89
16.2	Bremsstrahlung . . . . .	90
16.2.1	James Bond . . . . .	91
16.2.2	Emissione asimmetrica . . . . .	91
16.2.3	Simulazione numerica . . . . .	91
16.2.4	Metodo di riduzione della varianza . . . . .	93

<b>17 Lezione 17</b>	<b>94</b>
17.1 Campionamento di importanza . . . . .	94
17.1.1 Valor medio per distribuzione non uniforme . . . . .	95
17.1.2 Applicazione nel nostro caso . . . . .	95
17.1.3 Implementazione . . . . .	97
17.2 Option Pricing . . . . .	97
17.2.1 Opzioni . . . . .	98
17.2.2 Modello di Black&Scholes . . . . .	98
<b>18 Lezione 18</b>	<b>99</b>
18.1 Modello di Black&Scholes . . . . .	99
18.1.1 Eq. di Black & Scholes . . . . .	99
18.1.2 Modellizzazione numerica . . . . .	100
18.1.3 Generatore di numeri random normali . . . . .	101
18.2 Script Fortran . . . . .	102
18.3 Algoritmo Metropolis . . . . .	104
<b>19 Lezione 19</b>	<b>105</b>
19.1 Somma di distribuzioni uniformi . . . . .	105
19.2 Equazioni differenziali . . . . .	106
19.2.1 Algoritmo di Eulero . . . . .	106
19.2.2 Esempi di problemi integrabili . . . . .	107
19.2.3 Applicazione algoritmo di Eulero al primo esempio . . . . .	107
19.3 Eulero improved . . . . .	110
19.3.1 Esecuzione dello script . . . . .	110
<b>20 Lezione 20</b>	<b>112</b>
20.1 Metodo di Eulero . . . . .	112
20.2 Algoritmi Multistep . . . . .	113
20.2.1 Algoritmo integrale . . . . .	113
20.2.2 Derivata centrale . . . . .	114
20.3 Applicazione algoritmi . . . . .	114
20.3.1 Multistep integrale . . . . .	114
20.3.2 Multistep Der. Centrale . . . . .	114
20.4 Stabilità Soluzione . . . . .	116
20.4.1 Analisi di Stabilità di Von Neumann . . . . .	116
<b>21 Lezione 21</b>	<b>118</b>
21.1 Metodo di Runge Kutta . . . . .	118
21.1.1 Algoritmo del II ordine . . . . .	118
21.1.2 Algoritmo del IV ordine . . . . .	119
21.1.3 Script . . . . .	119
21.2 Simulazione del pendolo . . . . .	123
<b>22 Lezione 22</b>	<b>125</b>
22.1 Problema alle condizioni al contorno . . . . .	125
22.1.1 Algoritmo di shooting . . . . .	125
22.1.2 Esercizio di QM . . . . .	126
22.2 PDE . . . . .	127
22.2.1 Tre classi di PDE . . . . .	127
22.2.2 Condizioni al contorno . . . . .	128

22.2.3	Equazione di diffusione . . . . .	129
22.2.4	Schema esplicito . . . . .	130
22.2.5	Schema implicito . . . . .	131
22.2.6	Eq. d’Onda . . . . .	131
<b>23</b>	<b>Lezione 23</b>	<b>134</b>
23.1	Eq. del Calore . . . . .	134
23.1.1	Schema esplicito . . . . .	134
23.1.2	Schema Implicito . . . . .	138
23.1.3	Controllo bontà soluzione . . . . .	140
23.1.4	Note per l’esame . . . . .	140

# Chapter 1

## Lezione 1

### 1.1 Introduzione al corso

Si comincia a sviluppare l'abitudine a sviluppare soluzioni numeriche a problemi non risolvibili analiticamente. Se in analisi ho dei punti di riferimento, qui devo assicurarmi della bontà delle mie soluzioni.

### 1.2 Rappresentazione numerica

Il primo problema che si affronta è la rappresentazione dei numeri, che in una macchina sono rappresentati in un sistema di 0 e di 1. Si usa la rappresentazione in base 2, dove la sequenza di 0 e di 1 sono i coefficienti delle potenze di 2 nel mio sviluppo e, per esempio, se abbiamo un numero intero

$$187 = b_n 2^n + b_{n-1} 2^{n-1} \dots + b_0$$

Come otteniamo i coefficienti? Se dispari  $b_0 = 1$ , se pari vale zero. Sottraggo  $b_0$  al numero  $N$ , il numero risultante è pari

$$\frac{N - b_0}{2} = \frac{186}{2} = b_n 2^{n-1} + b_{n-1} 2^{n-2} \dots b_2 2 + b_1$$

Come prima, se dispari  $b_1 = 1$ , se pari vale zero. E così via. Infine abbiamo

$$\frac{2 - 0}{2} = 1$$

Perciò  $187 = 10111011$ . Dopotutto si possono fare le operazioni con i numeri in base 2. Adesso, tutto questo finalizzato al problema ai bit. I bit singoli sono raggruppati in gruppi e, per esempio, il **byte** sono 8 bit. Possiamo costruire il nostro array di 8 bit dedicando un bit al segno, poi ho 7 bit che possono assumere i valori di 0, 1. Al massimo arrivo al numero 127. È un range di numeri molto limitati, per aumentare la capacità di numeri rappresentati, si utilizzano delle architetture di 32, 64 bit, dove il massimo numero rappresentabile è  $2^{32} - 1 \sim 10^9$ . Se faccio un ciclo su un numero intero che va da  $1 - 10^{10}$ , mando in overflow una macchina di 32 bit. Ovviamente, con i numeri interi si fa poco, servono numeri frazionari. Nei numeri razionali la rappresentazione semplicemente ha un punto, la virgola e altre cifre. Ci sono potenze sia positive, nella parte prima della virgola, che potenze negative, nella parte dopo la virgola, con altri coefficienti. Abbiamo una forma del tipo

$$\alpha_n 2^n, \quad \alpha_{-1} 2^{-1} + \dots + \alpha_0 2^0 + \alpha_{-n} 2^{-n}$$

I coefficienti negativi si trovano in modo semplice. Prendo la parte frazionaria .625, posso scriverla come

$$.625 = \sum_{k=1}^n b_k 2^{-k}$$

Moltiplico il numero per 2 e guardo la parte intera 1.25. In questo caso  $b_1 = 1$ .

$$1.25 = b_1 + \sum_{k=2}^n b_k 2^{-k}$$

Perciò

$$.25 = \sum_{k=2}^n b_k 2^{-k}$$

Rimoltiplicando per 2 ottengo

$$.5 = b_2 + \sum_{k=3}^n b_k 2^{-k}$$

Da cui ricavo  $b_2 = 0$ . Adesso

$$1 = b_3 + \sum_{k=4}^n b_k 2^{-k}$$

Da cui ne deduco  $b_3 = 1$  e il resto è 0, dunque  $b_{k>3} = 0$ . In generale posso trattare il problema nel seguente modo, scrivendo il numero  $x$

$$x = s \cdot M \cdot b^e$$

$s$  è il segno,  $M$  è la mantissa,  $b$  è la base, elevata ad un certo  $e$  esponente. Il nostro numero, essendo in base 2, sarà rappresentato dalla coppia  $(M, e)$ . Prendo  $\frac{1}{2} = .5$ , in base 2 è

$$.5_{10} = \sum_{k=1}^n b_k 2^{-k} = .1$$

Posso includere il  $-1$  nell'esponente  $e$ . Quindi la mia rappresentazione include dedicare una parte dei 32 bit all'esponente e una parte alla mantissa. Tipicamente c'è il segno da considerare, gli si dedica un bit. 8 bit e i rimanenti 23 per la mantissa. Qui possiamo decidere come attribuire un numero sulla mantissa tra numero ed esponente, poiché la rappresentazione non è unica. Posso scrivere  $\frac{1}{2}$  come

$$[0, 00000000, 100000000000000000000000]$$

Posso anche dire che la mantissa la scriviamo come  $M = 2^1 \tilde{M}$ , dove  $\tilde{M} = 2^{-1}M$ , dunque ho un 0.1, che moltiplica un 0.1. La nuova mantissa è 0.01. Quindi, la mantissa con questa moltiplicazione per 2 e la divisione per 2, l'1 è stato shiftato verso 2. Ciò permette di far comparire un 2 al numeratore, prima era  $2^0$ . Un'altra possibile rappresentazione di questo numero è

$$[0, 00000001, 010000000000000000000000]$$

Questa cosa la possiamo fare con qualsiasi potenze vogliamo. Possiamo scrivere  $\frac{1}{2} = 2^2 \tilde{M}$ , perciò scriviamo 2 nella parte dedicata all'esponente, in questo caso shiftiamo l'1 a sinistra al numeratore e spostiamo a destra l'1 della mantissa.

$$[0, 00000010, 001000000000000000000000]$$

Posso aumentare il numeratore e la mantissa di conseguenza si sposta a destra. Dal punto di vista numerico le rappresentazioni sono tutte equivalenti. Lo sarebbero se disponessimo di un numero infinito di bit, con l'architettura a 64 bit vengono utilizzati 11 bit per l'esponente e 52 per la mantissa. Ci sono maggiori possibilità. Noi al calcolatore non disponiamo di un numero infinito di bit, sono tutte equivalenti finché a destra dell'ultimo bit non compare 1. Se noi scomponessimo il numero  $\frac{1}{3}$ , la storia sarebbe diversa. Se moltiplico il numero 10011011 per 2, la mantissa scorre a destra e l'ultimo 1 me lo perdo, la mantissa non corrisponde più al numero che avevo prima. Questa semplice osservazione è il cuore dei problemi di round-off numerici. Perché quando voi fate operazioni, i numeri vengono portati tutti alla stessa base e per portarli alla stessa base, se abbiamo numeri diversi dobbiamo svolgere l'operazione di shifting su uno dei due. Per esempio

$$10^{-7} = 2^{-x}, \quad x = ?$$

$2^{-x} = e^{-x \ln 2}$ , prendiamo il logaritmo di entrambe le parti

$$\ln 10^{-7} = \ln(e^{-x \ln 2}) = -x \ln 2$$

Viene fuori

$$-7 \ln 10 = -x \ln 2 \rightarrow x = \frac{7 \ln 10}{\ln 2} \simeq 23.25$$

Quindi  $10^{-7} = 2^{-23.25} = 2^{-23} \cdot 2^{-.25}$ . Immagino ora di svolgere l'operazione  $3 + 10^{-7}$ .  $3_{10} = 11_2$ . Devo portare la rappresentazione del 3 ad avere questo  $2^{-23}$ , perciò devo moltiplicare per un numero di volte, portandoli ad avere la stessa base, moltiplico per 2 per 23 volte, corrispondentemente devo far shiftare di 23 posti a destra la mantissa e in un'architettura a 32 bit porterà sempre al risultato 3, perdendo di precisione. Il risultato dell'operazione, data la rappresentazione finita dei numeri mi ha portato ad un round-off. Questa è una problematica di cui risente tutto lo sviluppo numerico computazionale. Si citano alcuni parametri legati alla rappresentazione numerica. Machine accuracy, il più piccolo numero che aggiunto a 1 dà un numero diverso da 1. Per le macchine a 32 bit è dell'ordine di  $10^{-8}$ . Questo dipende dal numero di cifre dedicate alla mantissa. Nel linguaggio che utilizziamo, **Fortran**, abbiamo

- *real\*4* con 8 cifre significative,
- *real\*8* con 16 cifre significative. Nella pratica le cifre diventano 15 ma l'ordine di grandezza è quello.
- *real\*16* con 32 cifre significative.

La possibilità di disporre di diverse precisioni permette di vedere la stabilità dei miei risultati. Tutto ha un prezzo, perché lavorando con meno cifre sto bloccando meno informazioni e le operazioni fluiscono in modo più veloce.

## 1.3 L'esame

Si hanno due possibilità, o si viene e si discute alla lavagna degli argomenti trattati, oppure se si ha qualche argomento che ci si prepara, si fa un piccolo lavoretto a livello di sviluppo algoritmo. Si porta e si discute il perché si affrontano i problemi in quel modo e si introduce il problema fisico in 5 minuti e si discutono i pro e i contro degli algoritmi utilizzati e i risultati ottenuti.

## 1.4 Il problema delle equazioni di II grado

Si affronta il problema principale dovuto ai round-off numerici.

$$ax^2 + bx + c = 0, \quad x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Potremmo avere dei problemi ad utilizzare la formula quando i coefficienti  $a, b, c$  diventano molto diversi fra di loro, come per esempio in  $(b \sim 10, a \sim 10^{-5}, c \sim 10^{-3})$  e in  $(b \sim 1, a \sim 1, c \sim 10^{-5})$ . Cosa si fa in questo caso? Riprendo la formula

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Vedo che la quantità  $\sqrt{b^2 - 4ac} \sim b$ , perciò quando faccio  $b^2$  non perdo cifre, perdo cifre nella parte relativa a  $4ac$ , quando però fate  $-b + \sqrt{b^2 - 4ac}$  le cifre si cancellano e bisogna avere traccia nella rappresentazione numerica di  $4ac$ , perché poi tutto viene diviso per  $a$ . Analiticamente in questo caso faccio

$$x_{1,2} = \frac{-b \pm b\sqrt{1 - \frac{4ac}{b^2}}}{2a} \approx \frac{-b \pm b(1 - \frac{2ac}{b^2})}{2a} \begin{cases} \frac{1}{2a}(-b - b + \frac{2ac}{b}) = (-\frac{b}{a} + \frac{c}{b}) \\ \frac{1}{2a}(-b + b - \frac{2ac}{b}) = (\frac{c}{b}) \end{cases}$$

Dove ho utilizzato lo sviluppo di McLaurin della radice. Quando abbiamo sotto controllo analiticamente, vediamo come conviene agire, però quando non ho il controllo, può avere problemi in base ai valori di  $a, b, c$ .

### 1.4.1 Algoritmo in Fortran

L'algoritmo si commenta. Nel calcolatore abbiamo un massimo numero rappresentabile e un minimo numero rappresentabile. Oltre i limiti massimi abbiamo **underflow** e **overflow**, non dipendono dalla mantissa ma dall'esponente, quindi lo spettro di numeri è veramente grande. Per esempio in singola precisione siamo a  $10^{60}$  e poi a  $10^{308}$ . Per le divisioni, diversamente dalla somma, se i numeri non differiscono di ordini di grandezza stellari, non pone alcun problema per l'**overflow**. Il  $q$  è la quantità da calcolare, dove vediamo una

somma. Dobbiamo controllare se questi due numeri diventano di ordine di grandezza diverso, o se vediamo delle cancellazioni. Nella forma numerica viene calcolata la differenza

$$q = -\frac{1}{2}(b + \text{sign}(b)\sqrt{b^2 - 4ac}), \quad x_1 = \frac{q}{a}, \quad x_2 = \frac{c}{q}$$

Se c'è cancellazione bisogna stare attenti a non perdere informazione. In questa scrittura se  $b < 0$  non c'è cancellazione, ho un numero negativo a cui aggiungo un numero negativo e se  $b > 0$  non c'è cancellazione perché ad un numero positivo aggiungo un numero positivo. Si prova a vedere il funzionamento, si compila con **gfortran -g eq-secondo-grado.f -o eq-secondo-grado.exe**. Se non faccio niente mi salva il file **eq-secondo-grado.exe**. Avvio il file **./eq-secondo-grado.exe**. Setto i parametri e ottengo le due versioni delle soluzioni.

Con l'algoritmo classico. Con precisione di tipo \*4

$$\begin{aligned}x_1 &= -6.49225855 \\x_2 &= -4.10747528E - 02\end{aligned}$$

Con precisione di tipo \*8

$$\begin{aligned}x_1 &= -6.4922587742407858 \\x_2 &= -4.1074559092548313E - 002\end{aligned}$$

Con l'algoritmo controllato. Con precisione di tipo \*4

$$\begin{aligned}x_1 &= -6.49225855 \\x_2 &= -4.10745591E - 02\end{aligned}$$

Con precisione di tipo \*8

$$\begin{aligned}x_1 &= -6.4922587742407858 \\x_2 &= -4.1074559092548042E - 002\end{aligned}$$

Le cifre a disposizione sono aumentate, sono diventate 15 e il primo numero è cambiato leggermente, le ultime due cifre sono diventate 55 → 77 su  $x_1$ . Sulle altre, prima avevamo 747528 e 745591, ora abbiamo 745590 e 745590, sono tutte cifre compatibili con l'algoritmo *computational friendly*. Con la doppia precisione abbiamo confermato la validità della soluzione dell'equazione di II grado data nella forma  $\frac{q}{a}$  e  $\frac{c}{q}$ . Se posso utilizzare formule che per poche cifre sono protette da cancellazioni, tanto meglio. Mi difendo dal *round-off*. Q/A:  $.e0$  descrive la precisione, con precisione doppia si ha la dicitura  $.d0$ . Se noi avessimo scritto semplicemente 4 al posto di  $4.e0$ , a seconda del compilatore e delle impostazioni della macchina, succede che quel 4 è caricato su una variabile dichiarata reale. 4 vuol dire 4.0000000000000 o meno. Se non associo la precisione, a seconda delle opzioni, le cifre dopo la virgola possono essere numeri casuali. Noi abbiamo in mente di mettere 4.000 e con la precisione riusciamo ad ottenere il risultato desiderato.

# Chapter 2

## Lezione 2

### 2.1 Introduzione alla lezione

Si studiano gli effetti dell'errore di roundoff. Nella seconda parte facciamo un'introduzione alla derivazione, dove ci sono aspetti non banali relativi a  $f(x + dx) - f(x)$ , una differenza tra numeri dello stesso ordine e si aggiunge al fatto che dobbiamo fare un'approssimazione rispetto alla definizione analitica. Prima di ciò studiamo gli effetti dei round-off numerici, dove dal punto di vista analitico non c'è alcuna approssimazione.

### 2.2 Successione di potenze

Il problema è molto semplice, consideriamo una successione e la sezione aurea

$$\phi = \frac{\sqrt{5} - 1}{2}, \quad a_n = \phi^n$$

Possiamo direttamente agire per esponenziazione progressiva e il numero sarà una potenza positiva di  $\phi$ , che tende progressivamente a 0, oppure calcoliamo  $\phi^n$  con una relazione di ricorsione. Se prendiamo la seguente espressione

$$\phi^n = -\phi^{n-1} + \phi^{n-2}, \quad \phi^0 = 1, \quad \phi^1 = \frac{\sqrt{5} - 1}{2}$$

Per il caso  $n = 2$

$$\phi^2 = -\phi^1 + \phi^0 = -\phi + 1 = -\frac{\sqrt{5} - 1}{2} + 1 = \frac{3 - \sqrt{5}}{2}$$

La supponiamo valida per  $n$  e dobbiamo dimostrare la sua validità per  $n + 1$ , procedendo quindi **per induzione**.

$$\phi^{n+1} = -\phi^n + \phi^{n-1}$$

Divido per  $\phi^{n-1}$ , questo ci da

$$\phi^2 = -\phi^1 + \phi^0, \rightarrow \phi^2 + \phi^1 - 1 = 0 \rightarrow \phi_{1,2} = \frac{\pm\sqrt{5} - 1}{2}$$

Si calcola numericamente questa quantità. Proviamo con un semplice eserciziello a calcolare  $\phi^n$  con la potenza ennesima oppure applicare l'algoritmo ricorsivo  $\phi^n = -\phi^{n-1} + \phi^{n-2}$ . Analiticamente sono la stessa cosa ma mentre da un lato faccio la potenza ennesima di numeri floating point, potrei aspettarmi piccole differenze dovute al round-off. Prendendo  $n = 10$ , potrei calcolarmi direttamente  $\phi^{10}$ , oppure applicare dieci volte l'algoritmo ricorsivo. Passiamo adesso alla fase di programma per vederlo in pratica.

#### 2.2.1 Algoritmo in Fortran

Al [link](#) è presente l'algoritmo con ricorsione. Le variabili del ciclo sono inizializzate al valore di partenza. con  $i = 2$  calcoliamo  $xn = \phi^2$ , stampiamo  $x1^i$  e successivamente stampiamo le varie risposte.

10  $xn = 8.12971592E - 03 \quad 8.13062023E - 03 \quad 0.999888778 \quad 0.617939174$

Si vede l'indice della potenza, i valori di  $xn$  calcolati con la relazione di ricorsione, oppure con  $x^i$ , sono molto simili, ci sono differenze sulle ultime cifre, poiché i round-off si propagano in modo diverso. Il rapporto

tra l' $n$ -esimo numero e il precedente è più o meno costante. Il numero inizia a degradare dalla decima potenza. Si vede che il rapporto del risultato della ricorsione rispetto al risultato della potenza diretta è molto vicino a 1, però oscilla dall'essere più o meno di 1. Abbiamo in genere 6 cifre buone, salvo la decima potenza in cui c'è un peggioramento nel rapporto. Si sta degradando la precisione. Se siamo interessati a calcolare  $\phi^7$ , questi risultati vanno benissimo, ma se siamo interessati a calcolare molti elementi della successione, questo risultato deve farci pensare. Se stampo  $n = 30$ , ricompilo il programma e lo faccio girare.

$$20 \ xn = -4.49419022E - 05 \ 6.61069862E - 05 \ -0.679835916 \ -0.225940259$$

I valori progrediscono a cambiare fino ad oltre il 64%, prima di andare nel negativo. La differenza peggiore tanto che

$$30 \ xn = -1.36575699E - 02 \ 5.37490848E - 07 \ -25409.8652 \ -1.61780369$$

La serie dei  $\phi^n$  ottenuti con la ricorsione diventa a segni alterni. Il rapporto tra le due predizioni è completamente perso rispetto ad 1, aumenta ad ogni iterazione di un ordine di grandezza. Il rapporto tra  $\phi^n$  e  $\phi^{n-1}$  passa da 0.618020177 a una zona di transizione a -1.61780369. Nella pratica i due metodi sono completamente imparagonabili e l'algoritmo ricorsivo è inaffidabile. I round-off sono presenti in entrambe le colonne. Dobbiamo chiederci se tutta questa differenza è dovuta davvero agli effetti di round-off. Questo è soltanto un primo esempio applicato agli elementi di una successione, che però si presenterà ancora quando faremo più avanti le equazioni differenziali. Le quantità con cui stiamo lavorando sono tutte in doppia precisione. Rispetto all'esercizio di adesso hanno un numero di cifre significative raddoppiato. Se il responsabile di tutto è il round-off, dovremmo vedere un pattern migliore.

12	$xn =$	3.1056200151340363E-003	3.1056200151418603E-003	0.99999999999748068	0.61803398874774351
----	--------	-------------------------	-------------------------	---------------------	---------------------

Il rapporto è 1, con una precisione migliore di prima, questo torna con il controllo migliore che ci aspettiamo dalle operazioni in doppia precisione. Questo è la soluzione del problema perché ci permette di dimostrare l'effetto del round-off sul problema, se non l'intero problema. La differenza da 1 inizia ad aumentare con l'aumentare degli step.

30	$xn =$	5.3744530248422961E-007	5.3749049985557113E-007	0.99991591038101391	0.61796216994775399
----	--------	-------------------------	-------------------------	---------------------	---------------------

Il numero 0.618 sembra cambiare un pochino. Rispetto a prima ho solamente cambiato il problema, se sono interessato a calcolare  $\phi^{25}$  con la doppia precisione, con un grado di approssimazione di  $10^{-8}$  funziona benissimo, ma se spingo in là le potenze un po' di più succede qualcosa di analogo a prima. Dopo il valore  $\phi^{40}$  la serie ricorsiva ritorna a segni alterni e il valore del rapporto torna a convergere a -1.618. A

50	$xn =$	-6.8366510141970593E-007	3.5531863700963522E-011	-19240.901833166914	-1.6177297929577077
----	--------	--------------------------	-------------------------	---------------------	---------------------

Il responsabile di tutto questo dunque è il round-off, perché abbiamo visto che con la doppia precisione abbiamo semplicemente spostato in avanti il problema, non risolvendolo di fatto.

## 2.2.2 Riflessione analitica

Bisogna fare una riflessione sulla parte analitica. Riguardo i segni alterni la ricorsione è instabile, mentre la potenza  $n$ -esima risulta stabile e sempre positiva. L'impatto del round-off sembra maggiore sulla relazione di ricorsione, questo perché si opera la sottrazione su due termini che non sono così drammaticamente diversi l'uno dall'altro che, in linea di principio non ci fanno pensare a grosse cancellazioni, ma in sottrazione si perde qualche cifra significativa. Siccome la potenza  $n$ -esima si ottiene come applicazione successiva della formula di ricorsione, qui la applichiamo e si perdono delle cifre  $n$  volte, con un'amplificazione del round-off. Se io calcolo la potenza  $n$ -esima con la potenza dovrei avere lo stesso tipo di problema, ma questo non è il caso, perché la potenza si calcola con qualche algoritmo con la serie esponenziale, dove se prendo  $\phi^n$  con la serie esponenziale troncata ad un certo punto, io sto sommando tutti i termini positivi. Ho round-off numerici, ma non ho differenze tra termini. La cosa diversa che c'è nell'algoritmo di ricorsione è la seguente, il nostro algoritmo

di ricorsione ha come soluzione due valori di  $\phi$ . Una, che è la soluzione di interesse, e anche la coniugata. Analiticamente c'è un'uguaglianza tra le due espressioni, purché si selezioni la radice positiva. L'algoritmo si basa su un'equazione di secondo grado che ha due radici distinte. Ma se noi siamo come condizione iniziale di sceglierne una, dovrebbe procedere senza sbagliare mai. Però, purtroppo, dal punto di vista numerico, quando diciamo che  $\phi^1 = \frac{\sqrt{5}-1}{2}$  è un floating point, non ne disponiamo un numero infinito di cifre. Affinché la ricorsione sia perfettamente equivalente alla potenza  $n$ -esima di  $\phi$ , è necessario disporre di un numero infinito di cifre. Se non riusciamo a calcolare  $\phi^1$  con precisione infinita vuol dire che la soluzione più generale dell'algoritmo di ricorsione sarà una combinazione delle potenze di  $\phi$  o del coniugato. Tra le due c'è una differenza molto grande. Se chiamo la radice positiva  $\phi_1$  e la radice negativa  $\phi_2$ , trovo che  $|\phi_1| < 1$  e il valore delle sue potenze tende a diminuire, invece, con  $\phi_2$ , abbiamo che  $|\phi_2| > 1$ , ed è negativo. Questo implica che se non siamo in grado di isolare esattamente con precisione infinita la radice positiva, il nostro algoritmo sarà sempre inquinato da una componente relativa alla radice negativa. Quando noi iteriamo l'algoritmo per calcolare le potenze superiori, succede che nella nostra combinazione lineare, la parte relativa a  $\phi_1$  diminuisce in modulo e la parte relativa a  $\phi_2$  tende ad aumentare per il suo modulo. Questa è la ragione fondamentale per cui si vede questo andamento bislacco.

$$(a\phi_1 + b\phi_2)^n \approx b^n \phi_2^n$$

Nelle equazioni di II grado c'era da tenere sotto controllo le operazioni nell'argomento della radice del discriminante. Lì le condizioni sono due, o la soluzione è precisa, o non lo è. Ma qui ci sono comportamenti diversi, di segno oscillante e il round-off la condizione iniziale contribuiscono a questo comportamento. Se lavoriamo con doppia precisione, la contaminazione dovuta all'altra radice sarà minore. Per questo in doppia precisione compare più avanti il contributo dell'altra radice, rispetto alla singola precisione. La quadrupla precisione sposta molto più avanti il problema. Vedremo questo problema con le equazioni differenziali, che implicano algoritmi che provano a ricostruire la funzione in esame a partire da determinate condizioni iniziali. Noi siamo in grado di fissare le condizioni iniziali in modo più preciso possibile. Al calcolatore il programma di Laplace con tutte le condizioni iniziali di tutte le particelle dell'universo e io vi darò la loro evoluzione futura, ha senso in modo analitico ma se uno pretende di implementare il tutto in un calcolatore, conduce a risultati che non hanno senso, perché non potrò mai implementare con precisione infinita la mia condizione iniziale. Gli errori di round-off possono essere piccoli o devastanti, a seconda delle applicazioni che stiamo facendo. Analiticamente, non siamo capaci di mettere  $a = 1, b = 0$ , con precisione infinita. è come se  $\phi_C^1 = \phi_1 + \varepsilon\phi_2$ , con  $\varepsilon$  tanto più piccolo, tanto più sono le cifre significative di cui dispongo. Questo problema affligge molti metodi risolutivi delle equazioni differenziali e bisogna tenerne conto. Rileva molta importanza la capacità di fissare le condizioni iniziali con grande precisione. Posso solo rendere  $\varepsilon$  sufficientemente piccolo da non interferire con la mia necessità. La ricorsione rimane utile se entrambe le due radici di un dato polinomio sono in modulo minori di uno. Se vogliamo avere un'idea del  $\phi^n$  calcolato con la potenza al variare della precisione, possiamo introdurre una variabile in doppia, dove  $dx_1$  è calcolata in doppia precisione. Adesso stampiamo  $i, xn, xi, dxi$ , perché così confrontiamo le successioni ottenute con diversa precisione. Con precisione infinita il rapporto sarebbe esattamente 1. Nel progredire delle potenze, la soluzione del calcolo di  $\phi^n$  si sta allontanando dal calcolo in doppia precisione. Tenendo conto che abbiamo portato le potenze fino a  $n = 50$ , sono partito con un numero rappresentato con una precisione minore e facendoci le potenze che ho nel numero che ho in singola precisione, ho una proiezione molto meno instabile rispetto alla relazione di ricorsione. C'è sempre un problema relativo alla condizione di round-off ma c'è un meccanismo perverso per cui c'è una piccola contaminazione della radice negativa nel numero che non riesco a rappresentare con infinita precisione che porta a un errore nella progressione dell'algoritmo.

## 2.3 Algoritmo della derivazione

Fatta la premessa delle successioni ci spostiamo verso la soluzione delle eq. differenziali. Prendiamo  $f(x), x \in \mathbb{R}$ , ne calcoliamo

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

Analiticamente ha senso, numericamente è meno chiaro. Dobbiamo fissare  $h$  molto piccolo e calcolare come converge la serie di numeri al variare di  $h$ . In questo rapporto incrementale, più  $h \rightarrow 0$ , più  $f(x+h) \simeq f(x)$ . Stiamo calcolando la derivata della funzione  $f$  dove  $f \sim 1$  e  $f' \sim 1$ , perciò  $f(x+h) - f(x) \sim h$ . Quando mettiamo valori piccoli numericamente per  $h$ , dobbiamo avere in mente il numero di cifre significative, per cui  $h$  è piccolo ma non indefinitivamente piccolo. In singola precisione  $h \sim 10^{-8}$  ma non  $h \sim 10^{-15}$ , perché siamo sotto alla **machine accuracy**. Risulterebbe  $h = 0$  e ne otterremmo solo risultati sbagliati. Siamo passati dal problema del limite alla forma discretizzata, ma questo porta un errore relativo alla precisione con cui questo rapporto approssima la derivata definita analiticamente. Sulla carta ci domandiamo che precisione abbia, appunto. Noi ci prendiamo lo sviluppo di Taylor, per cui prendiamo tutte funzioni analitiche.

$$f(x+h) = f(x) + hf'(x) + \frac{1}{2!}h^2f''(x) + \dots$$

Da questo deduciamo

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{1}{2!}hf''(x) + \dots$$

Perciò già notiamo dei termini correttivi di ordine  $h$ : sono errori che non hanno niente a che vedere con il round-off, che sono legati all'espressione che inserisco nel rapporto incrementale.  $h$  lo controlliamo analiticamente e dal punto di vista della derivata possiamo definire

$$\frac{f(x+h) - f(x-h)}{2h}$$

una definizione alternativa del rapporto incrementale, simmetrica. Dal punto di vista della definizione analitica del rapporto incrementale, non mi cambierebbe sostanzialmente niente. Vediamo se usare una forma o l'altra può avere qualche utilità dal punto di vista della precisione che andiamo ad utilizzare. Abbiamo visto che

$$f'(x) = \frac{f(x+h) - f(x)}{h} + O(h)$$

Adesso sviluppiamo

$$\begin{aligned} f(x+h) &= f(x) + hf'(x) + \frac{1}{2!}h^2f''(x) + \frac{1}{3!}h^3f'''(x) + \dots \\ f(x-h) &= f(x) - hf'(x) + \frac{1}{2!}h^2f''(x) - \frac{1}{3!}h^3f'''(x) + \dots \end{aligned}$$

Adesso costruiamo il rapporto incrementale

$$f(x+h) - f(x-h) = 2hf'(x) + \frac{2}{3!}h^3f'''(x)$$

Noi siamo interessati a

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{1}{3!}h^2f'''(x)$$

Questo è un altro algoritmo di implementazione della derivata. La **derivata forward**  $+h$  è quella che si calcola col punto più a destra. La **derivata backward**  $-h$  è quella che si calcola col punto più a sinistra. Quella che si calcola con entrambi i punti si chiama **derivata centrale**.

3
10.00000
1.00000 1.00000
10.00000 0.00000 0.00000 10.00000
10.00000 0.00000 0.00000 -10.00000
10.00000 -1.23945 -2.05932 -9.70627
10.00000 1.23945 2.05932 9.70627

Nel caso della derivata forward e della derivata backward ho una precisione di ordine  $h$ , con la derivata centrale ho una precisione di ordine  $h^2$ . Quindi algoritmi che dal punto di vista analitico sono perfettamente equivalenti possono avere grandi differenze nel comportamento numerico, nella precisione con la quale definisco la mia quantità. Se ho lavorato con 8 cifre significative posso pretendere al massimo  $h \sim 10^{-8}$ , mentre potrò definire la derivata con una precisione di  $10^{-16}$ . Perciò la derivata centrale è preferibile per questo motivo. Il guadagno

in precisione ha un prezzo, perché devo calcolare la funzione in più punti, come nella derivata backward la calcoliamo in  $x + h, x$ , nella derivata centrale la calcoliamo in  $x + h, x - h$ . Dobbiamo conoscere la funzione in punti vicini ma sia a destra che a sinistra. Nelle equazioni differenziali bisogna calcolare le derivate su una serie di punti che sono sparsi in un intervallo finito, che genera una serie di problemi.

# Chapter 3

## Lezione 3

### 3.1 L'algoritmo della derivazione numerica a 5 punti

In precedenza abbiamo introdotto l'algoritmo per la derivazione numerica. Abbiamo visto che ha una precisione di  $h$ , lo spacing tra i punti forward o backward. Poi abbiamo visto l'algoritmo della derivata centrale, che è più preciso dei due precedenti, con una precisione di ordine  $h^2$ , quindi con una convergenza più forte al valor vero. La derivata forward e backward indica solo due punti rispetto al punto  $x$ , rispetto ai quali calcolare la derivata. L'algoritmo centrale necessita la conoscenza di due punti simmetrici rispetto al punto nel quale sono interessato a calcolare la derivata. Se io conosco la funzione in  $x$ , in realtà mi serve la conoscenza della funzione in altri due punti. Se ci interessa raggiungere una precisione maggiore, dobbiamo aumentare l'informazione che inseriamo. Una cosa che viene in mente è di estendere il concetto di derivata centrale, includendo nella definizione i punti  $x \pm 2h$ , nei quali si ottengono gli sviluppi di Taylor.

$$f(x+2h) = f(x) + 2hf'(x) + \frac{1}{2!}4h^2f''(x) + \frac{1}{3!}8h^3f'''(x) + \frac{1}{4!}16h^4f''''(x) + \dots$$
$$f(x-2h) = f(x) - 2hf'(x) + \frac{1}{2!}4h^2f''(x) - \frac{1}{3!}8h^3f'''(x) + \frac{1}{4!}16h^4f''''(x) + \dots$$

Effettuo la differenza membro a membro delle due espressioni, osservando che sopravvivono i termini dispari

$$f(x+2h) - f(x-2h) = 4hf'(x) + \frac{16}{3!}h^3f'''(x) + O(h^5)$$

Richiamo definizione presa dai punti  $x \pm h$

$$f(x+h) - f(x-h) = 2hf'(x) + \frac{2}{3!}h^3f'''(x) + O(h^5)$$

Se prendo la combinazione 8 volte la definizione di  $x \pm h$  ricavo

$$8[f(x+h) - f(x-h)] = 16hf'(x) + \frac{16}{3!}h^3f'''(x) + O(h^5)$$

Sono giunto a questa espressione perché in questo modo posso sottrarre le due espressioni e far sparire il contributo di derivata terza

$$f(x+2h) - f(x-2h) - 8[f(x+h) - f(x-h)] = 4hf'(x) + \frac{16}{3!}h^3f'''(x) - 16hf'(x) - \frac{16}{3!}h^3f'''(x) + O(h^5)$$

Faccio emergere il termine positivo di derivata prima

$$f(x-2h) - 8[f(x-h) - f(x+h)] - f(x+2h) = 12hf'(x) + O(h^5)$$

Vediamo che sono coinvolti quattro punti in cui è necessario conoscere la funzione incriminata e poi la conoscenza della derivata prima della funzione nel punto

$$f'(x) = \frac{f(x-2h) - 8f(x-h) + 8f(x+h) - f(x+2h)}{12h} + O(h^4)$$

Facendo combinazioni lineari opportune di punti della funzione, possiamo calcolarci la derivata con la precisione analitica che desideriamo. Questo gioco può essere portato con una potenza di  $h$  desiderata. Possiamo raggiungere la potenza di  $h$  desiderata che ci fa soddisfare la precisione necessaria. Questo si chiama algoritmo a 5 punti. Stiamo vedendo questi algoritmi sulla carta, vedendo la precisione che ci permettono di raggiungere

dal punti di vista teorico. Eppure non è detto che funzionino nella pratica, perché per precisioni successive dobbiamo costruire combinazioni lineari opportune che coinvolgono un numero di termini sempre maggiore. Per esempio, rispetto alla banale derivata forward, che introduce una differenza tra due termini, con quattro termini sono presenti almeno due differenze. Per il principio per il quale si perde precisione ogni differenza, non è detto che la precisione effettivamente aumenti con lunghe combinazioni lineari e che un algoritmo sia migliore degli altri. Dal confronto tra gli algoritmi possiamo farci un'idea di quello che sta succedendo.

### 3.1.1 Derivata seconda

Lo stesso discorso si può fare per la derivata seconda, si può definire una derivata seconda con una derivata forward della derivata prima

$$f''(x) = \frac{f'(x+h) - f'(x)}{h} = \frac{f(x+2h) - f(x+h) - f(x+h) + f(x)}{h^2} = \frac{f(x+2h) - 2f(x+h) + f(x)}{h^2}$$

è l'algoritmo forward per la derivata seconda. Noi conosciamo la precisione relativa a ciascuna delle due derivate, vediamo che si mantenga la precisione con cui siamo partiti per la derivata prima. Per fare questo dobbiamo sviluppare i termini che compaiono nella definizione

$$f(x+2h) - 2f(x+h) + f(x) = h^2 f''(x) - f(x) + O(h^3)$$

Pertanto

$$f''(x) = \frac{f(x+2h) - 2f(x+h) + f(x)}{h^2} + O(h)$$

Non si è degradata la precisione che avevamo per l'algoritmo adottato per la derivata prima nella derivata seconda. Si possono costruire degli algoritmi che ne aumentano la precisione. In questa definizione abbiamo i punti  $x+2h, x+h, x$ . Se prendo

$$\begin{aligned} f(x-h) &= f(x) - hf'(x) + \frac{1}{2!}h^2 f''(x) - \frac{1}{3!}h^3 f'''(x) + \frac{1}{4!}h^4 f''''(x) + O(h^5) \\ f(x+h) + f(x-h) &= 2f(x) + h^2 f''(x) + O(h^4) \\ f''(x) &= \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + O(h^2) \end{aligned}$$

Con questa espressione dell'algoritmo della derivata otteniamo un ordine di precisione maggiore sull'incremento. Possiamo aumentare la precisione facendo ulteriori costruzioni di combinazioni lineari. è in generale possibile ottenere una precisione di grado arbitrario per una derivata di ordine  $n$ , sufficientemente elevata nelle potenze dell'incremento.

### 3.1.2 Algoritmo in Fortran della derivata prima

È disponibile al [link](#) la lista degli algoritmi in Fortran che adotteremo. Si dichiara la funzione, che deriva da un blocco esterno come **external f**. Per una funzione  $f(x) = x^2$ , di cui si calcola la derivata prima con i diversi algoritmi

```
enter value of x0
3.e0
enter step
0.01e0
forward derivative    =   6.00996017
backward derivative   =   5.99002838
central derivative    =   5.99999428
five point deriv     =   6.00001049
five point deriv     =   6.00001049
"analytical value" =   6.00000000
```

Con un incremento di  $10^{-2}$ , ci si aspetta una precisione di  $10^{-2}$  per gli algoritmi forward e backward, mentre per l'algoritmo centrale ci aspettiamo  $10^{-4}$ , mentre rende  $10^{-6}$ . La derivata a cinque punti è dello stesso ordine

della derivata centrale, dunque non guadagnamo molto, anche se sulla carta la derivata a cinque punti aveva una precisione di  $h^4$ . Quindi avrei dovuto avere almeno una precisione di  $10^{-8}$ , ma è già oltre il limite della precisione dovuta ai round-off numerici.

- Si verifica numericamente il comportamento atteso per la derivata forward

$$\frac{(3 + 0.01)^2 - 3^2}{0.01} = \frac{9 - 9 + 10^{-4} + 6 \cdot 0.01}{0.01} = 6.01$$

Esattamente come il codice restituisce.

- Proviamo adesso a capire come mai sia molto preciso l'algoritmo centrale

$$\frac{f(x+h) - f(x-h)}{2h} = \frac{(x+h)^2 - (x-h)^2}{2h} = \frac{4xh}{2h} = 2x$$

Restituisce di fatto la soluzione esatta. Numericamente infatti si riscontra una precisione di  $10^{-6}$ , al limite della singola precisione.

- Si studia l'algoritmo a 5 punti

$$\begin{aligned} f'(x) &= \frac{1}{12h} [f(x-2h) - 8f(x-h) + 8f(x+h) - f(x+2h)], \quad f = x^2 \\ &= \frac{1}{12h} [(x-2h)^2 - 8(x-h)^2 + 8(x+h)^2 - (x+2h)^2] \\ &= \frac{1}{12h} [-4xh + 16xh + 16xh - 4xh] \\ &= \frac{1}{12} [24x] = 2x \end{aligned}$$

Analiticamente non abbiamo termini spuri, si sono cancellati i termini che diminuiscono la precisione, siamo dunque sensibili ai round-off numerici, tant'è che l'algoritmo a cinque punti e l'algoritmo centrale ottengono la stessa precisione. Ragionando, riscopro che sono entrambi algoritmi esatti, perciò emerge sempre come precisione numerica  $10^{-4}$ . Le differenze presenti nell'algoritmo sono tra numeri dell'ordine di  $10^2$ , quindi, rispetto a 1, sono due ordini di grandezza in più e la cancellazione degli ordini 0.01 risente maggiormente del fatto che non stiamo vedendo una correzione di 0.01 rispetto a 1, ma rispetto a 100. Questo ci fa perdere qualche cifra significativa.

Se c'è una ragione della cancellazione dei termini spuri, è nella simmetria dell'algoritmo. Si va a prendere la funzione in punti esattamente simmetrici rispetto al punto in cui si calcola la derivata. Se prendo la funzione  $x^2$ , simmetrica a destra e sinistra rispetto al punto  $x = 3$ , ottengo valori simmetrici. Questa è la ragione del comportamento numerico, sviluppato analiticamente. Dobbiamo fare considerazioni sulla simmetria della funzione attorno al punto in cui vogliamo calcolarne la derivata. Se cambiamo la precisione, ci aspettiamo che il forward lasci fuori un termine  $h$ . Andiamo a riunire l'algoritmo sempre con la stessa finezza e con lo stesso valore d'inizio. Si possono fare diverse prove con funzioni particolari e punti caratteristici.  $e0 = 10^0$  in singola precisione, mentre  $d0 = 10^0$  in doppia precisione e  $q0 = 10^0$  in quadrupla precisione.  $2 \cdot 10^{-1}$  si scrive rispettivamente come  $e - 1$ ,  $d - 1$ ,  $q - 1$ . Posso costruire delle funzioni con più valori di input e di output con delle subroutine, si esegue l'istruzione **call "nome della subroutine"**. Non c'è più la dichiarazione di external ma vengono portate nell'espressione nella subroutine. Ci sono variabili che sono conosciute da tutte le parti del programma, purché richiami l'espressione **common block**, sono globali. Nell'utilizzare questo tipo di variabili bisogna stare attenti perché sono conosciute in tutte le parti del programma. Se in una parte le modifichiamo, il suo valore risulta alterato in tutte le parti del programma.

```

enter value of x0
3.d0
enter step
0.01d0
forward derivative = 6.0899999999998488
backward derivative = 5.9899999999998954
central derivative = 5.9999999999993721
five point deriv = 5.9999999999998428
five point deriv = 5.9999999999998419
''analytical value' = 6.0000000000000000

```

Notiamo che la precisione delle derivate forward e backward è rimasta invariata, mentre gli algoritmi centrale e a cinque punti sono aumentati moltissimo di precisione. Cominciano a sballare verso la 15<sup>a</sup> cifra. È esattamente quello che ci aspettiamo. Torna con le predizioni legate alla simmetria degli algoritmi negli argomenti. Se faccio questo calcolo in un punto differente l'andamento delle precisioni varia o rimane uguale? Provo  $x_0 = 1000$ .

```

enter value of x0
1000.d0
enter step
0.01d0
forward derivative = 2000.0099999946542
backward derivative = 1999.9899999937043
central derivative = 1999.9999999941792
five point deriv = 1999.9999999941792
five point deriv = 1999.9999999941792
''analytical value' = 2000.0000000000000

```

Abbiamo come  $f'(x) = 2000$ , la backward rispetta esattamente il numero che ci aspettiamo per via analitica, la centrale e la cinque punti sono uguali, ma abbiamo perso alcune cifre. Abbiamo perso circa una cifra. Abbiamo sempre lo stesso numero di cifre per la mantissa e lo stesso incremento. Abbiamo però aumentato il numero, con lo shift sulla mantissa ci mangiano le cifre che aumentano in  $x_0$ . Ogni operazione che l'algoritmo opera è sorgente di errore. Stiamo toccando con mano le varie sorgenti dei round-off numerici.

## 3.2 Algoritmo di derivazione con funzione non simmetrica

Poniamo ora come funzione  $f(x) = \ln x$ . Con funzioni particolari si possono assistere ad amplificazioni del round-off, come nel caso del  $\sin(x)$  per il punto  $x = \frac{\pi}{2}$ .

```

enter value of x0
42.e0
enter step
9.1e0
forward derivative = 2.37798691E-02
backward derivative = 2.38394737E-02
central derivative = 2.38096714E-02
five point deriv = 2.38100681E-02
five point deriv = 2.38100700E-02
''analytical value' = 2.38095243E-02

```

In questo caso la derivata bacward sembra più vicina al valore analitico. La derivata centrale risulta più precisa di quella a cinque punti. Ricordiamoci che stiamo facendo un limite per  $h \rightarrow 0$ . Dovremo provare ad avere più valori di  $h$ , per esempio

```

enter value of x0
42.e0
enter step
0.01e0
forward derivative = 2.37941742E-02
backward derivative = 2.38180161E-02
central derivative = 2.38060951E-02
five point deriv = 2.38120556E-02
five point deriv = 2.38120556E-02
``analytical value' = 2.38095243E-02

```

Il forward e il backward si sono avvicinati. La derivata centrale è molto precisa, come la five point. Sembra raggiungere una precisione di  $10^{-6}$ , ai limiti della singola precisione. Provo a diminuire ancora questo step

```

enter value of x0
42.e0
enter step
0.0001e0
forward derivative = 2.38418579E-02
backward derivative = 2.38418579E-02
central derivative = 2.38418579E-02
five point deriv = 2.42392235E-02
five point deriv = 2.42392235E-02
``analytical value' = 2.38095243E-02

```

I risultati sono andati tutti a ramengo, nel senso che la derivata foward e backward sono distanziati e sono uguali alla derivata centrale. La five point è ancora più imprecisa. Con quello step così piccolo, quello che è andato a dominanza totale è la perdita di precisione dovuta al round-off, alla faccia di tutta le precisioni considerate sulla carta. Se aumentiamo la precisione a **real\*** 8 otteniamo

```

enter value of x0
42.d0
enter step
0.01d0
forward derivative = 2.3806689792227331E-002
backward derivative = 2.3812358726660321E-002
central derivative = 2.3809524259443826E-002
five point deriv = 2.3809523809511145E-002
five point deriv = 2.3809523809511141E-002
``analytical value' = 2.3809523809523808E-002

```

La five point ha preso almeno dieci cifre buone. Con la doppia precisione abbiamo spostato il problema dei round-offe torna il comportamento che ci aspettiamo analiticamente. Se porto il passo  $h = 10^{-4}$  ottengo

```

enter value of x0
42.d0
enter step
0.0001d0
forward derivative = 2.3809495464099939E-002
backward derivative = 2.3809552156528468E-002
central derivative = 2.3809523810314204E-002
five point deriv = 2.3809523809574053E-002
five point deriv = 2.3809523809574053E-002
``analytical value' = 2.3809523809523808E-002

```

Qui c'è uno scaling buono. Con un passo di  $10^{-9}$ , perderei tutto il guadagno ottenuto. Se introduciamo la quadrupla precisione **real\*** 16.

```
enter value of x0
42.q0
enter step
0.00000001q0
forward derivative = 2.38095238066893424040780342242172981E-0002
backward derivative = 2.38095238123582766444408562036711203E-0002
central derivative = 2.38095238095238095242594452139442092E-0002
five point deriv = 2.38095238095238095238095222998694238E-0002
five point deriv = 2.38095238095238095238095222998694268E-0002
``analytical value' = 2.38095238095238095238095238084E-0002
```

La five point ha una precisione spaziale teorica di  $10^{-32}$ , al limite della quadrupla precisione. Cominciamo ad avere discrepanze con il valore analitico oltre le 20 cifre, mentre per la centrale già a 10 cifre e per la forward e la backward a 8 cifre si scostano. Da questo esercizio ci si rende conto del motivo del libro di bibliografia di Fisk, che è molto enciclopedico, serve per chi fa ricerca. Per vedere gli algoritmi di un problema inesplorato, ci sono tutte le soluzioni aggiornate, si chiama *The art of scientific computing*, ci sono diversi fattori in gioco e bisogna tenere in considerazione del loro interplay. Non c'è la possibilità di affrontare nella pratica quotidiana problemi per via analitica e se scegliamo per via numerica dobbiamo sempre considerare la precisione del risultato che abbiamo ottenuto.

# Chapter 4

## Lezione 4

### 4.1 Implementazione algoritmo della derivata seconda

Si descrive un'implementazione dell'algoritmo della derivata seconda, a seguito della quale si introduce l'integrazione. L'algoritmo si trova al [link](#), sotto la voce **derivativeII.f**. Ci sono diverse opzioni per la derivata seconda.

- La prima opzione è di utilizzare gli algoritmi visti in precedenza
- La seconda possibilità è di utilizzare gli algoritmi della derivata prima per calcolare la derivata seconda come derivata prima.

Abbiamo una versione forward della derivata prima.

$$f''(x_0) = \frac{f'(x_0 + h) - f'(x_0)}{h} + O(h) \Rightarrow f''(x_0) = \frac{f(x_0 + 2h) - 2f(x_0 + h) + f(x_0)}{h^2} + O(h)$$

Abbiamo la derivata centrale

$$f''(x_0) = \frac{f'(x_0 + h) - f'(x_0 - h)}{2h} + O(4h^2) \Rightarrow f''(x_0) = \frac{f(x_0 + h) - 2f(x_0) + f(x_0 - h)}{4h^2} + O(h^2)$$

In base a quale algoritmo della derivata prima si adotta si giunge a un risultato differente. Vediamo le implementazioni di queste espressioni insieme ad un algoritmo che non utilizza l'espressione analitica, ma dato un algoritmo numerico per la derivata prima, possiamo calcolare la derivata seconda. Nell'algoritmo numerico abbiamo messo la forward, la centrale e la five points. La cinque punti analitica non l'avevamo ricavata, la deriviamo come esercizio.

#### 4.1.1 Derivata seconda a cinque punti

$$f'(x) = \frac{f(x - 2h) - 8f(x - h) + 8f(x + h) - f(x + 2h)}{12h} + O(h^4)$$

$$f''(x) = \frac{1}{12h} [f'(x - 2h) - 8f'(x - h) + 8f'(x + h) - f'(x + 2h)] + O(h^4)$$

Sviluppo ogni termine con l'algoritmo della derivata a cinque punti.

$$f'(x + h) = \frac{f(x - h) - 8f(x) + 8f(x + 2h) - f(x + 3h)}{12h} + O(h^4)$$

$$f'(x + 2h) = \frac{f(x) - 8f(x + h) + 8f(x + 3h) - f(x + 4h)}{12h} + O(h^4)$$

$$f'(x - h) = \frac{f(x - 3h) - 8f(x - 2h) + 8f(x) - f(x + h)}{12h} + O(h^4)$$

$$f'(x - 2h) = \frac{f(x - 4h) - 8f(x - 3h) + 8f(x - h) - f(x)}{12h} + O(h^4)$$

Ricompongo lo sviluppo dei termini nella definizione di derivata seconda

$$\begin{aligned}
 f''(x) &= \frac{1}{(12h)^2} \left[ f(x - 4h) - 8f(x - 3h) + 8f(x - h) - f(x) \right. \\
 &\quad - 8(f(x - 3h) - 8f(x - 2h) + 8f(x) - f(x + h)) \\
 &\quad + 8(f(x - h) - 8f(x) + 8f(x + 2h) - f(x + 3h)) \\
 &\quad \left. - (f(x) - 8f(x + h) + 8f(x + 3h) - f(x + 4h)) \right] + O(h^4) \\
 &= \frac{1}{144h^2} \left[ f(x - 4h) - 16f(x - 3h) + 64f(x - 2h) + 16f(x - h) - 130f(x) \right. \\
 &\quad \left. + 16f(x + h) + 64f(x + 2h) - 16f(x + 3h) + f(x + 4h) \right] + O(h^4)
 \end{aligned}$$

#### 4.1.2 Implementazione algoritmo derivata seconda

Si calcola la derivata prima in punti diversi per fare i rapporti incrementali, dunque dobbiamo chiudere l'algoritmo della derivata prima in una funzione e chiamare questa funzione in diversi argomenti e mettere poi insieme gli output. Per questo si può usare la struttura di function, la più semplice. Per la derivata forward abbiamo una dichiarazione in **real\*4**. Si mostra l'esempio per la derivata a cinque punti

```

real*4 function d5p(x,h)
implicit none
real*4 x,h
real*4 f
external f
d5p= (f(x-2.e0*h) - 8.e0*f(x-h) + 8.e0*f(x+h)-f(x+2.e0*h))/12.e0/h
end

```

Si utilizzano la funzione `ln.x`. Chiamiamo le funzioni con gli argomenti. Possiamo scrivere anche le derivate seconde a partire dall'algoritmo della derivata prima reiterato, tutto numericamente. Dobbiamo implementare per la derivata forward e la derivata backward l'algoritmo che coinvolge direttamente il valore della derivata, senza coinvolgere il valore numerico della derivata prima.

$$\frac{f(x_0 + 2h) - 2f(x_0 + h) + f(x_0)}{h^2} + O(h)$$

```

print*, 'forw analytical' = ','
        (f(x+2.e0*h)-2.e0*f(x+h)+f(x))/h/h
print*, 'bckw analytical' = ',
        (f(x-2.e0*h)-2.e0*f(x-h)+f(x))/h/h
print*, 'central analytical' = ',
        (f(x+h)-2.e0*f(x)+f(x-h))/h/h

```

Si procede inserendo il valore del punto desiderato, con derivata seconda  $-\frac{1}{x^2}$ .

```

enter value of x0
5.e0
enter step
0.001e0
forward II derivative = 0.00000000
backward II derivative = -0.00000000
central II derivative = -5.96046411E-02
five point II deriv = -0.103476137
``analytical value' = -3.99999991E-02
forw analytical = 0.00000000
bckw analytical = 0.00000000
central analytical = -0.119209275

```

Variando il passo di un ordine di grandezza otteniamo due valori di circa  $-11.9209$  per i valori analytical della derivata forward e backward. Se ne deduce che c'è una notevole sensibilità rispetto al passo. La derivata

seconda si sposta da  $-0.10$  a  $-0.16$ . Come mai quando facciamo questo algoritmo e facciamo la differenza di derivate prime, calcolate come differenze di funzioni, vediamo se l'errore di round-off si propaga di meno, per la divisione per  $h^2$ .  $h^2 = 10^{-8}$  è ai limiti della singola precisione. Proviamo dunque ad aumentare la precisione in doppia.

```
enter value of x0
5.d0
enter step
0.001d0
forward II derivative = -3.9984005661963806E-002
backward II derivative = -4.0016005620202577E-002
central II derivative = -4.0000003143081386E-002
five point II deriv = -3.999999829373280E-002
``analytical value' = -4.000000000000001E-002
forw analytical      = -3.9984005884008411E-002
bckw analytical      = -4.0016005398157972E-002
central analytical   = -4.000000645079581E-002
```

La precisione influenza nettamente la bontà del risultato. Si vede che la forward e la backward hanno una precisione di  $10^{-3}$ , mentre la derivata centrale ha il doppio delle cifre, come atteso. La derivata a cinque punti dovrebbe scalare ad  $h^4 \simeq 10^{-12}$ , ma a tal punto prevale la precisione adottata e non quello che potrebbe guadagnare se aumentassimo la precisione dei numeri che stiamo utilizzando. Se io confronto il risultato tra l'algoritmo forward applicato partendo dal risultato numerico della derivata prima rispetto ai corrispondenti analitici. Quello che si vede è che data questa precisione la forward analitica differisce da quella numerica sulla settima cifra, mentre la centrale, fatta tutta numericamente, è un po' meno precisa di quella analitica perché ci sono molte più operazioni coinvolte nella versione totalmente numerica rispetto a quella analitica. C'è una piccola differenza e si capisce perché. Questi esercizi dipendono anche dal punto in cui si calcola la derivata.

#### 4.1.3 Algoritmo attraverso il flesso

Se calcolo la derivata seconda di  $\frac{\sin(x)}{x}$  a cavallo di un flesso, io devo ottenere 0, con quale bontà riesco ad approssimarla? Dal punto di vista numerico devo vedere 0, compatibilmente con la precisione che sto utilizzando. Se calcolo la derivata prima, prendo lo 0, nel ramo  $0 < x < \pi/2$ , il rapporto incrementale è circa 1 o 0.5, questo mi deve dare una differenza di ordine  $h$ , perché diviso per  $h$  mi deve dare circa 1. Mi si devono cancellare tre cifre se ho  $h \sim 10^{-3}$ . Se con la derivata prima mi metto vicino al massimo, se prendo un algoritmo forward, prendo il massimo e un punto appena dopo, quindi la funzione è di ordine 1 e il risultato deve essere 0, a denominatore ho  $h$  e il risultato della differenza deve essere più piccolo di  $h$ , nel limite deve essere 0. Ci possono essere problemi e i problemi di round-off possono acuire la propagazione dell'errore. La stessa cosa vale per la derivata seconda, utilizzando gli algoritmi numerici della derivata prima, ci sono più termini a fare differenze e le cancellazioni possono essere più critiche. A denominatore ho un termine di ordine  $h^2$ , perciò, siccome la derivata seconda è nulla, a numeratore devo avere un termine  $\mathcal{O}(h^2)$ .

```
enter value of x0
1.5e0
enter step
0.001e0
forward II derivative = -0.953674257
backward II derivative = -1.01327896
central II derivative = -0.998377740
five point II deriv = -1.00127041
``analytical value' = -0.997494996
forw analytical      = -0.953674197
bckw analytical      = -1.01327896
central analytical   = -1.01327896
```

L'errore è del 5% nella forward, dell'1% nella backward. Anche se hanno un ordine uguale di errore, hanno un errore diverso. La centrale è di ordine mille e la cinque punti non guadagna nulla, perché siamo oltre

la precisione. In queste differenze è dominante il round-off introdotto dalla precisione dei numeri che stiamo utilizzando come sorgente di errore. Non siamo sensibili alle differenze tra gli algoritmi. Le espressioni analitiche commettono lo stesso errore. Ci avviciniamo allo 0

```
enter value of x0
0.0e0
enter step
0.001e0
forward II derivative = -1.13248825E-03
backward II derivative = 1.13248825E-03
central II derivative = 0.00000000
five point II deriv = 1.49011603E-05
``analytical value' = -0.00000000
forw analytical = -1.16415322E-03
bckw analytical = 1.16415322E-03
central analytical = 0.00000000
```

Ci muoviamo di poco dallo 0 con  $x_0 = 0.01$ ,

```
enter value of x0
0.01e0
enter step
0.001e0
forward II derivative = -1.11460676E-02
backward II derivative = -1.02519980E-02
central II derivative = -1.00135794E-02
five point II deriv = -1.01476898E-02
``analytical value' = -9.99983307E-03
forw analytical = -1.11758690E-02
bckw analytical = -1.02445474E-02
central analytical = -9.31322575E-03
```

Il valore analitico è di  $f'' \sim -0.01$ , la derivata prima ci va al dieci per cento, la seconda al due per cento e grosso modo l'algoritmo della derivata centrale ci va al .1%. Se la calcoliamo esattamente in 0, vediamo questi comportamenti, dove abbiamo  $10^{-3}$  per la forward e la backward, mentre abbiamo  $10^{-5}$  per la cinque punti. Mostra un comportamento anche migliore rispetto al punto  $x_0 = 0.01$ . Il valore analitico in quel punto si approssima  $f'' \sim -0.01$ , la derivata centrale è migliore di quella a cinque punti, per un accidente. Se la valutiamo in  $x_0 = 0$ , compare questo risultato bizzarro, l'analitica è giustamente 0 e il valore della derivata centrale è 0, e gli altri sono uguali a tutte le cifre di segno opposto. Questo comportamento si spiega con la parità della funzione.

#### 4.1.4 Parità funzione $\sin x$

Ripercorrendo l'algoritmo analitico centrale, troviamo

$$f''(x_0) = \frac{f(x_0 + h) - 2f(x_0) + f(x_0 - h)}{4h^2} + O(h^2)$$

Contestualizziamo a  $x_0 = 0$

$$f''(0) = \frac{f(h) - 2f(0) + f(-h)}{4h^2} + O(h^2)$$

Imponiamo la condizione di disparità  $f(-x) = -f(x)$  e  $f(0) = 0$

$$f''(0) = \frac{f(h) - f(-h)}{4h^2} = 0$$

## 4.2 Discorso su $f''$ nei pressi dello zero con precisione maggiore

Questo ci fa capire l'importanza di avere un'idea della funzione sottostante che stiamo calcolando. Se ci spostiamo di poco gli algoritmi non vedono più la stessa simmetria, quindi manifestano le differenze che ci

aspettiamo tra i vari algoritmi. Per fare un test conviene aumentare la precisione e variare lo step. Impostiamo uno step di 0.0001 nel punto 0.

```
enter value of x0
0.e0
enter step
0.0001e0
forward II derivative = 0.00000000
backward II derivative = -0.00000000
central II derivative = 0.00000000
five point II deriv = 9.93410795E-05
``analytical value' = -0.00000000
forw analytical = 0.00000000
bckw analytical = 0.00000000
central analytical = 0.00000000
```

Viene tutto 0 tranne che per la cinque punti. Sembrerebbe dover concludere che gli algoritmi forward e backward sono più precisi dell'algoritmo a cinque punti, chiaramente questo è un accidente dovuto al round-off rispetto al punto nel quale calcolo la derivata. Riprovando nel punto .01

```
enter value of x0
0.01e0
enter step
0.0001e0
forward II derivative = -9.35792923E-02
backward II derivative = 9.35792923E-02
central II derivative = 0.00000000
five point II deriv = 6.50684070E-03
``analytical value' = -9.99983307E-03
forw analytical = -9.31322575E-02
bckw analytical = 9.31322575E-02
central analytical = 0.00000000
```

La derivata forward e backward sbaglia di un ordine di grandezza. Come valore assoluto, la derivata a cinque punti è migliore. La derivata centrale pesca punti che con questa precisione si sommano con valori delle funzioni a zero. Il risultato delle differenze dà 0 al numeratore e dovrebbe sparire con l'aumento della precisione.

```
enter value of x0
0.d0
enter step
0.001d0
forward II derivative = -9.9999975011755282E-004
backward II derivative = 9.9999975011755282E-004
central II derivative = 0.0000000000000000
five point II deriv = -2.7755575615628914E-014
``analytical value' = -0.0000000000000000
forw analytical = -9.9999975010714448E-004
bckw analytical = 9.9999975010714448E-004
central analytical = 0.0000000000000000
```

La derivata forward e la backward sono di ordine  $\mp 10^{-3}$ , mentre prima eravamo al 0.01. La media centrale incontra l'accidente dovuto alla simmetria della funzione, mentre la cinque punti va a  $10^{-14}$ , mostrando una precisione molto maggiore.

```

enter value of x0
0.01d0
enter step
0.001d0
forward II derivative = -1.0999777250497722E-002
backward II derivative = -8.9998777544897379E-003
central II derivative = -9.9998300004955354E-003
five point II deriv = -9.9998333328021882E-003
``analytical value' = -9.9998333341666645E-003
forw analytical = -1.0999777250511600E-002
bckw analytical = -8.9998777545036157E-003
central analytical = -9.9998324985112186E-003

```

Forward e backward mantengono la stessa precisione, la centrale migliora decisamente e le cinque cifre sono esatte. La cinque punti riprende fino alla ottava cifra. Si noti che la forma analitica implementata con l'algoritmo implementato dalla  $f$ , per la forward è uguale per una decina di cifre, mentre per la centrale ci sono, dopo cinque cifre alla sesta, differiscono. Questa è la differenza della propagazione del round-off nei vari algoritmi. La cinque punti completamente numerica è più vicina al valore analitico. Chiaramente, scegliere un algoritmo o l'altro implica una combinazione diversa dei numeri e noi sappiamo che sono affidabili alla sesta cifra. Al denominatore abbiamo  $h^2 \sim 10^{-6}$  e il risultato della derivata seconda è di  $10^{-2}$ , ciò implica che al numeratore ho un numero di  $10^{-8}$ . Vuol dire che devo avere sotto controllo le differenze tra numeri e lasciarmi un remnant di 12 cifre. Se avevo un tot di cifre, 6 le perdo con la divisione per  $h^2$  e quindi la mia precisione è molto più piccola rispetto a quella nominale che ho in mente. È difficile avere sotto controllo le cose al meglio del per cento in singola precisione, fatta eccezione per comportamenti accidentali. In generale, quindi, bisogna fare attenzione al valore del numeratore, dato il denominatore e il risultato che devo ottenere. Ciò ha a che vedere con l'ordine di grandezza della funzione della quale vado a calcolarne le differenze. Si studiano nella pratica successiva, magari per la tesi di laurea, si fanno applicazioni numeriche con algoritmi già esistenti e uno degli obiettivi del corso è quello di far vedere come i vari algoritmi hanno performance diverse in base alle condizioni nei quali vengono implementati. Prima di fidarci di un algoritmo a scatola chiusa, è meglio testarne la bontà in condizioni ottimali. Gli algoritmi che scriviamo nel corso sono pensati per essere applicati universalmente. Come faccio a fidarmi del risultato senza variare la dimensione numerica. Devo sempre chiedermi nella sequenza di operazioni che faccio quali differenze sono comprese. È un aspetto tanto banale quanto non semplice e dipende dall'analisi caso per caso. Quando facciamo le derivate si applica a situazioni molto varie e nel caso della funzione  $\sin x$  non sovengono pensieri relativi ai problemi più comuni che sono presenti. Gli step con cui vado a calcolare la deriivata, pensando al limite del rapporto incrementale e alla doppia precisione, prendendo  $h = 10^{-5/-6}$ , dovrebti ottenere un risultato migliore. Questo non è per nulla ovvio. Prendiamo appunto  $h = 10^{-6}$  e vediamo cosa salta fuori

```

enter value of x0
0.01d0
enter step
0.000001d0
forward II derivative = -9.9989461155303161E-003
backward II derivative = -1.0000680839006293E-002
central II derivative = -9.9993798241548859E-003
five point II deriv = -9.9993555935373735E-003
``analytical value' = -9.9998333341666645E-003
forw analytical = -9.9972113920543393E-003
bckw analytical = -1.0002415562482270E-002
central analytical = -9.9989461155303161E-003

```

Cosa vediamo? Vediamo che i risultati sono peggiori rispetto a quelli ottenuti con  $d - 3$ . Se prendiamo la five points, prima era in grande accordo con quella analitica, adesso già alla quarta cifra è in disaccordo. In un algoritmo lavorato analiticamente stiamo mettendo uno step al denominatore dell'ordine  $d - 12$ , quindi a numeratore dobbiamo avere, per ottenere un risultato di  $d - 2$ , un numero dell'ordine di  $d - 14$ . Se avessimo dei numeri di ordine 1, con una differenza e con il round-off perderemmo tutta la doppia precisione, che non è più sufficiente. Nel nostro caso, in realtà, siamo nell'ordine  $d - 2$ , quindi alcune cifre le abbiamo ancora, anche se sono risultati che differiscono dal valore vero del per cento o del permille. L'implementazione dell'algoritmo

con il passo di  $d - 6$  porta a maggiore imprecisione, questo è sempre da tener presente. Questi esercizi sono basati sulla derivata in un punto, si immagini quanto diventa raffinata la gestione della soluzione delle equazioni differenziali. Questi esercizi sono implementati con il valore analitico per confrontarne la bontà assoluta, altrimenti dovremmo solo paragonarne la stabilità passando a precisioni successive. Anche questo è il bello di affrontare problemi numerici.

#### 4.2.1 Problema di $\sin x/x$

```
enter value of x0
0.1d0
enter step
0.0001d0
forward II derivative = -0.33233192864301486
backward II derivative = -0.33233591434367327
central II derivative = -0.33233392982001675
five point II deriv = -0.33233393236427783
``analytical value' = -0.33233392841714249
forw analytical      = -0.33233192864301486
bckw analytical      = -0.33233591434367327
central analytical   = -0.33233393814668943
```

I risultati sono buoni. Se andiamo a vedere un plot di  $\frac{\sin x}{x}$ , il valore numerico della derivata seconda a  $x_0 = 0.1$  è  $f''(x_0) \approx -0.33233392$ . La derivata a cinque punti non avanza in precisione rispetto alla derivata centrale. Il round-off domina sulla precisione analitica. La differenza tra la derivata centrale puramente numerica e quella con la derivata prima implementata analiticamente è piuttosto piccola. Più ci avviciniamo a 0, siccome la funzione oscilla più frequentemente, per avere un risultato affidabile, occorre inserire uno step che non peschi un punto che appartiene alle oscillazioni successive. Se portassimo  $h = 0.001d0$  e  $x_0 = d - 3$ ,

```
enter value of x0
0.001d0
enter step
0.001d0
forward II derivative = -0.33333291649118024
backward II derivative = NaN
central II derivative = -0.33333316665218327
five point II deriv = NaN
``analytical value' = -0.33333323337137699
forw analytical      = -0.33333291649118024
bckw analytical      = NaN
central analytical   = NaN
```

Addirittura emergono dei valori **NaN**. I **Not a Number** perché escono? Nella forward c'è un valore buonissimo. Nella backward c'è **NaN**. Quindi perché? Abbiamo messo un  $x_0 = h = d - 3$ , è fattibile, ma noi andiamo a calcolare la funzione in  $x_0 - h$  e in 0 la funzione  $\frac{\sin x}{x}$  è uguale a una divergenza. L'assunzione è solamente  $\lim_{h \rightarrow 0}$ , non è riferito a  $x_0$ . Il problema dell'identificazione del punto adatto in cui calcolare la derivata e se vario i valori vicini non mi produce grosse instabilità.

# Chapter 5

## Lezione 5

### 5.1 Integrazione

In generale, gli algoritmi di integrazione si dividono in due grandi classi

- Algoritmi deterministici. Sono quelli che si vedono da qui a qualche lezione. I punti che vengono utilizzati per calcolare l'integrale sul dominio, nel caso di  $1 - D$ , sono distribuiti su un intervallo  $1 - D$  con un algoritmo deterministico, come equispaziati. Si raggiungono diversi tipi di precisioni a seconda degli algoritmi.
- Algoritmi Monte Carlo. Sono una classe separata nel senso che i numeri corrispondenti alle ascisse nel dominio di integrazione sono distribuiti secondo una distribuzione pseudo casuale, vedremo cosa comporta a livello di precisione e se ci sono vantaggi e quali. Ci saranno degli svantaggi e vedremo come ovviare a questi problemi.

L'integrazione numerica fa riferimento a questa classe di problemi. Gli algoritmi deterministici sono stati sviluppati per affrontare integrali monodimensionali, anche se possono essere adattati a coprire integrali multidimensionali. Si sviluppa un algoritmo ad hoc per il problema specifico o si utilizzano i medesimi utilizzati in  $1 - D$  in modo ricorsivo.

### 5.2 Integrazione per algoritmi deterministici

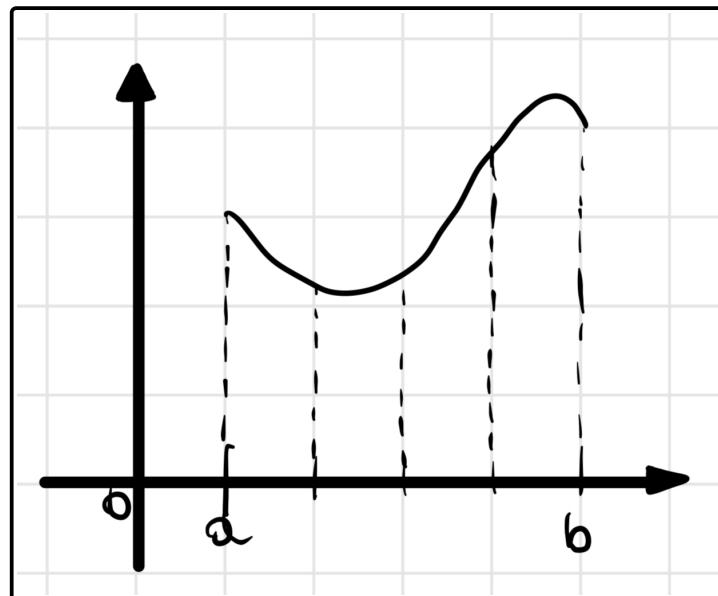
Abbiamo diversi algoritmi, ce ne sono di più banali e di più sofisticati.

- Algoritmo del rettangolo.
- Algoritmo del trapezio, ha delle proprietà che sono variazioni piccole.
- Algoritmo di Sympson.
- Algoritmi di Gauss, più sofisticati.

Vedo l'integrazione come

$$\int_a^b f(x)dx = \lim_{n \rightarrow \infty} \$n$$

Se abbiamo una funzione, definita tra  $[a, b]$ ,



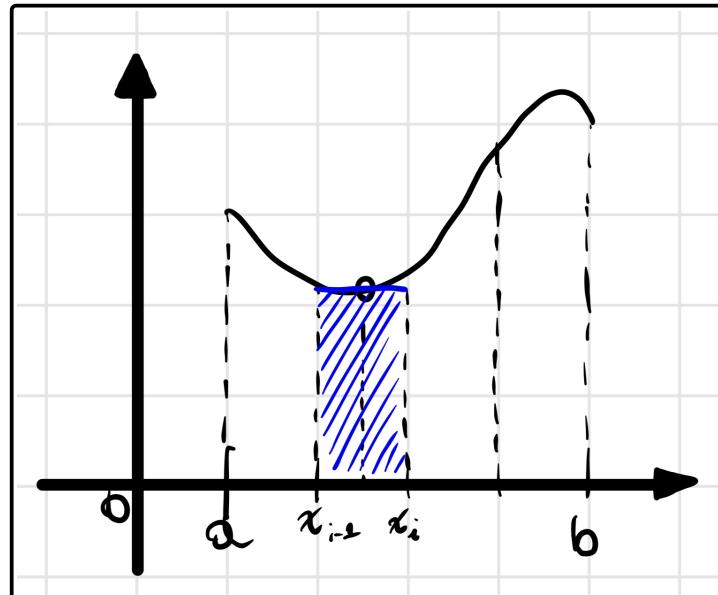
Noi facciamo una reticolazione del nostro intervallo e poi andiamo a calcolare un'approssimazione dell' integrale con un algoritmo.

### 5.2.1 Algoritmo del rettangolo

Ci focalizziamo su una separazione  $\Delta x = x_i - x_{i-1} = h$

$$\int_{x_{i-1}}^{x_i} f(x) dx = f\left(\frac{x_{i-1} + x_i}{2}\right) \Delta x = f_{i-\frac{1}{2}} h$$

Prendiamo il rettangolo che approssima la funzione in quell'intervalllo.



L'algoritmo è molto semplice. Si tratta di stimare qual'è l'errore corrispondente a questo algoritmo, se io approssimo l'integrale sul singolo intervallo e poi lo approssimo su tutti gli intervalli, l'errore sull'intervallino sarà amplificato dal numero di intervalli. Qual'è l'errore commesso? In questo caso dobbiamo fare degli sviluppi di Taylor, stiamo ipotizzando di sviluppare in serie funzioni analitiche, derivabili di classe  $f \in C^\infty$

sull'intervallo  $(a, b)$ .

$$\begin{aligned}
 f(x) &= f_{i-\frac{1}{2}} + f'_{i-\frac{1}{2}}(x - x_{i-\frac{1}{2}}) + \frac{1}{2}f''_{i-\frac{1}{2}}(x - x_{i-\frac{1}{2}})^2 + O\left((x - x_{i-\frac{1}{2}})^3\right) \\
 \int_{x_{i-1}}^{x_i} f(x) dx &= f_{i-\frac{1}{2}}h + \frac{1}{2} \int_{x_{i-1}}^{x_i} f''_{i-\frac{1}{2}}(x - x_{i-\frac{1}{2}})^2 dx \\
 &= f_{i-\frac{1}{2}}h + \frac{1}{2}f''_{i-\frac{1}{2}} \frac{1}{3}(x - x_{i-\frac{1}{2}})^3 \Big|_{x_{i-1}}^{x_i} \\
 &= f_{i-\frac{1}{2}}h + \frac{1}{2}f''_{i-\frac{1}{2}} \frac{2}{24}h^3 \\
 \int_{x_{i-1}}^{x_i} f(x) dx &= I_{i-1,i} + \frac{h^3}{24}f''_{i-\frac{1}{2}} + \dots
 \end{aligned}$$

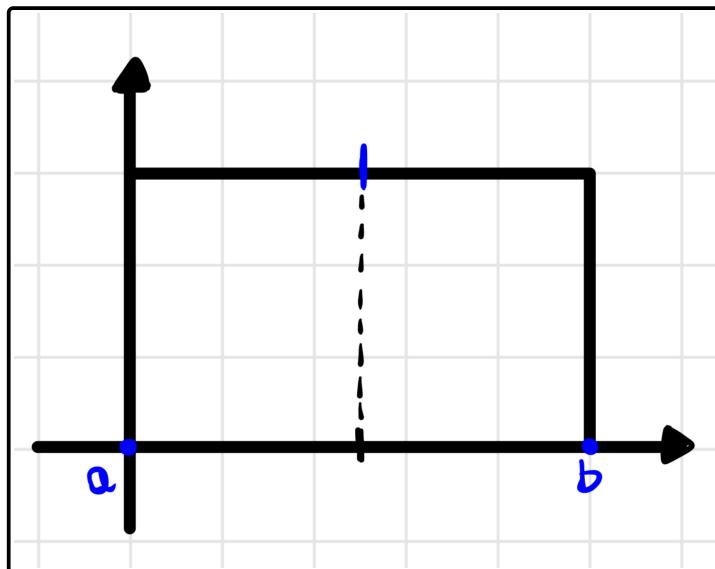
Stiamo approssimando la funzione con il termine  $f_{i-\frac{1}{2}}$  e vediamo l'errore portato dagli altri termini. Abbiamo il valore del nostro algoritmo, che coincide con il valor vero, tranne che per un termine che va come  $h^3$ . Nel limite di  $h \rightarrow 0$ , coincide con il valor vero. Ora si fa l'integrale su tutto l'intervallo  $(a, b)$ , che è uguale a  $\Delta x = \frac{b-a}{N}$ .

$$\begin{aligned}
 \int_a^b f(x) dx &= \sum_{i=0}^N \int_{x_{i-1}}^{x_i} f(x) dx = \sum_{i=0}^N I_{i-1,i} + \frac{h^3}{24} \sum_{i=0}^N f''_{i-\frac{1}{2}} \\
 &= \sum_{i=0}^N \int_{x_{i-1}}^{x_i} f(x) dx = \sum_{i=0}^N I_{i-1,i} + \frac{h^3}{24} N f''(\xi)
 \end{aligned}$$

Per il teorema della media  $\sum_{i=0}^N f''_{i-\frac{1}{2}} = N f''(\xi)$ , dove  $\xi \in (a, b)$ . Questa è la nostra stima dell'errore. C'è una relazione di proporzionalità inversa tra  $h$  ed  $N$ , dove  $h^3 = \frac{(b-a)^3}{N^3}$ , quindi si vede che il nostro algoritmo del rettangolo ci tira fuori un termine di errore di ordine

$$\frac{(b-a)^3}{N^3 24} N f''(\xi) \sim \frac{1}{N^2} \frac{(b-a)^3}{24} f''(\xi)$$

Quello che ci interessa, poiché cerchiamo un algoritmo universale, è prendere una reticolazione, calcolare la  $f$  nel punto medio dell'intervallo selezionato e ciò viene fatto per tutti gli intervalli e gli  $f(\bar{x})$  vengono moltiplicati per l'ampiezza dell'intervallo, costruendo l'approssimazione con il multirettangolo. Ciò produce un risultato che viene approssimato con una precisione di  $N^{-2}$ . Qui il parametro fondamentale è  $N$ , il numero di punti con cui si fa l'integrale, che corrisponde ad  $h$ , comunque convenzionalmente ci si riferisce ad  $N$  perché più l'integrale è preciso con un numero limitato di punti, minore è la complessità computazionale che l'algoritmo richiede e comporta tempi di calcolo più ridotti. Il problema degli algoritmi di calcolo deterministici è trovare un algoritmo che minimizzi l'errore. L'errore va come  $N^{-\alpha}$ , con  $\alpha$  più grande possibile. Se fosse possibile  $\alpha \rightarrow \infty$ , avremmo risolto il problema dell'integrale in modo esatto con un algoritmo.



L'algoritmo del rettangolo con un punto, rispetto ad un rettangolo è uguale a  $c(b - a)$ . Divido l'intervallo in due, calcolo la funzione nel punto medio

$$f\left(\frac{b-a}{2}\right)(b-a)$$

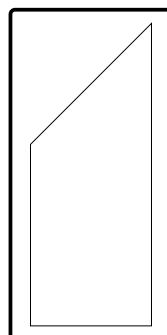
In questo caso il risultato dell'algoritmo coincide con l'integrale della funzione, inoltre tutte le derivate sono nulle, dunque l'algoritmo dà approssimazione esatta. Se prendo una funzione generica questo non succede e aumentando il numero di punti vedo un miglioramento del nostro algoritmo rispetto al valore analitico. Se devo fare un'integrazione numerica, a meno che la funzione sia analiticamente nota ma particolarmente complicata per cui la primitiva esiste, oppure è molto complicata da ottenere, spesso succede che la funzione è numericamente nota per punti e in questi casi bisogna ingegnarsi per ottenere un risultato numerico e cercare di ottenere una stima dell'errore più piccolo possibile che posso raggiungere. So per l'algoritmo che è circa

$$\sim \frac{1}{N^2}$$

Se noi abbiamo l'integrale di una funzione che conosciamo analiticamente e della quale conosciamo l'integrale, prendo  $N = 100$  punti e a questo punto supponiamo di avere un errore, che dipende dalla derivata seconda media, dunque non è esattamente  $N^{-2}$ , ma è  $N^{-2}\alpha$ . Se differisce dal valore dell'integrale del 10% e aumento il numero di punti a  $N = 200$ , dovrei aumentare la precisione di un fattore  $(200/100)^2 = 4$ , giungendo ad un errore del 2.5%. L'obiettivo della ricerca di algoritmi di integrazione numerica è vedere se si riesce a massimizzare l'esponente di  $N$ , così sono nati altri algoritmi.

### 5.2.2 Algoritmo del trapezio

è semplicemente basato su un'approssimazione diversa della funzione, prima approssimavo l'integrale sul sottointervallo di ampiezza  $\Delta x$  con il valore della  $f$  valutata nel punto medio, ora approssimo l'integrale con il trapezio che collega i valori della funzione estremi dell'intervallo.



Il valore dell'integrale secondo l'algoritmo è

$$I_{i-1,i} = \frac{1}{2}h(f(x_{i-1} + f(x_i)))$$

In totale è

$$\begin{aligned} \sum_i I_{i-1,i} &= \frac{1}{2}h[f_0 + 2f_1 + \dots + 2f(x_{N-1}) + f(x_N)] \\ &= h\left[\frac{1}{2}f_0 + f_1 + \dots + f(x_{N-1}) + \frac{1}{2}f(x_N)\right] \end{aligned}$$

A parte i punti estremi, il gioco della somma degli intervalli determina un fattore due della funzione. L'algoritmo è molto simile a prima ma leggermente diverso, per due motivi.

- Dati i punti della reticolazione andiamo a calcolare i valori della funzione sui punti della reticolazione.
- I punti non sono pesati tutti allo stesso modo.

L'algoritmo del rettangolo ci dice che prendo i punti di mezzo della reticolazione e calcolo il valore della funzione in essi. La cosa è diversa nel senso che peso gli estremi con un fattore  $\frac{1}{2}$  e negli altri ho un fattore 1. Tutti gli algoritmi differiranno per come sono pesati i punti interni all'intervallo. Questa idea sarà l'idea alla base dell'algoritmo più sofisticato di Gauss. Vediamo l'errore che si commette. Per fare la stima dell'errore è un po' più complicato ma sempre molto semplice.

### Stima dell'errore

Riferito all'intervallo  $[x_{i-1}, x_i]$

$$\begin{aligned} f(x) &= f_{i-1} + f'_{i-1}(x - x_{i-1}) + \frac{1}{2}f''_{i-1}(x - x_{i-1})^2 + \dots \\ f(x) &= f_i + f'_i(x - x_i) + \frac{1}{2}f''_i(x - x_i)^2 + \dots \\ f(x) &= \frac{1}{2}(f_{i-1} + f_i) + \frac{1}{2}f'_{i-1}(x - x_{i-1}) + \frac{1}{2}f'_i(x - x_i) \\ &\quad + \frac{1}{4}f''_{i-1}(x - x_{i-1})^2 + \frac{1}{4}f''_i(x - x_i)^2 \dots \\ \int_{x_{i-1}}^{x_i} f(x) dx &= I_{i-1,i} + \frac{1}{2}f'_{i-1} \int_{x_{i-1}}^{x_i} (x - x_{i-1}) dx + \frac{1}{2}f'_i \int_{x_{i-1}}^{x_i} (x - x_i) dx \\ &\quad + \frac{1}{4}f''_{i-1} \int_{x_{i-1}}^{x_i} (x - x_{i-1})^2 dx + \frac{1}{4}f''_i \int_{x_{i-1}}^{x_i} (x - x_i)^2 dx \dots \\ \int_{x_{i-1}}^{x_i} f(x) dx &= \frac{1}{2}h(f_{i-1} + f_i) + \frac{1}{4}f'_{i-1}(x - x_{i-1})^2 \Big|_{x_{i-1}}^{x_i} + \frac{1}{4}f'_i(x - x_i)^2 \Big|_{x_{i-1}}^{x_i} \\ &\quad + \frac{1}{12}f''_{i-1}(x - x_{i-1})^3 \Big|_{x_{i-1}}^{x_i} + \frac{1}{12}f''_i(x - x_i)^3 \Big|_{x_{i-1}}^{x_i} \dots \\ \int_{x_{i-1}}^{x_i} f(x) dx &= \frac{1}{2}h(f_{i-1} + f_i) + \frac{1}{4}f'_{i-1}(x_i - x_{i-1})^2 - \frac{1}{4}f'_i(x_{i-1} - x_i)^2 \\ &\quad + \frac{1}{12}f''_{i-1}(x_i - x_{i-1})^3 - \frac{1}{12}f''_i(x_{i-1} - x_i)^3 \dots \\ \int_{x_{i-1}}^{x_i} f(x) dx &= \frac{1}{2}h[f_{i-1} + f_i] + \frac{1}{4}[f'_{i-1} - f'_i](x_i - x_{i-1})^2 \\ &\quad + \frac{1}{12}[f''_{i-1} + f''_i](x_i - x_{i-1})^3 \dots \end{aligned}$$

Se prendiamo la semisomma otteniamo il risultato in riga 3. Adesso dobbiamo fare l'integrale nell'intervallo  $[x_{i-1}, x_i]$ . Si riconosce la porzione di intervallo  $x_i - x_{i-1} = h$

$$\int_{x_{i-1}}^{x_i} f(x) dx = \frac{1}{2} [f_{i-1} + f_i] h + \frac{1}{4} [f'_{i-1} - f'_i] h^2 + \frac{1}{12} [f''_{i-1} + f''_i] h^3 \dots$$

$$\int_a^b f(x) dx = h \left[ \frac{1}{2} f_0 + f_1 + \dots + f(x_{N-1}) + \frac{1}{2} f(x_N) \right]$$

$$+ \frac{1}{4} h^2 \left[ f'_0 - f'_1 + f'_1 - f'_2 + \dots + f'_{N-1} - f'_N \right]$$

$$+ \frac{1}{12} h^3 \left[ f''_0 + 2f''_1 + \dots + 2f''_{N-1} + f''_N \right] \dots$$

Si somma su tutti gli intervalli.

$$\int_a^b f(x) dx = I_{\text{trapezio}} + \frac{1}{4} h^2 \left[ f'_0 - f'_N \right]$$

$$+ \frac{1}{12} h^3 \left[ f''_0 + 2f''_1 + \dots + 2f''_{N-1} + f''_N \right] \dots$$

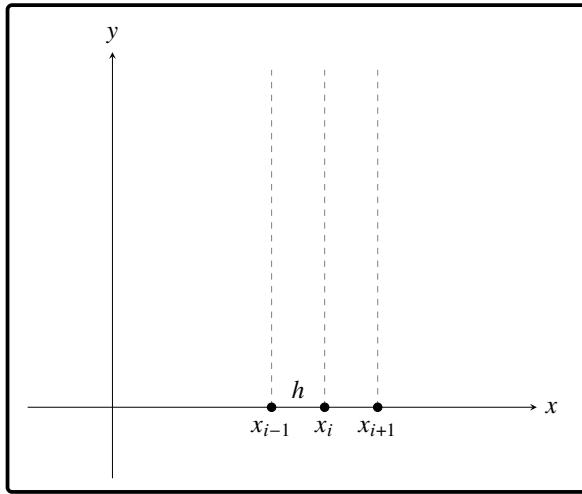
Uno può andare avanti a scrivere le derivate pari in funzione delle derivate dispari ma per i nostri scopi non è necessario, si tirano fuori i numeri di Bernoulli, portando il problema a potenze successive. L'algoritmo del trapezio è il *building block* dell'algoritmo di Sympson, che è quello più diffuso. Si vede che l'algoritmo del trapezio va come

$$\frac{1}{4} \frac{(b-a)^2}{N^2} [f'_0 - f'_N] \sim \frac{1}{N^2}$$

Si evince che l'algoritmo del trapezio, rispetto all'algoritmo del rettangolo non fa guadagnare nulla, perché, analogamente al rettangolo, la sommatoria di derivate seconde mi porta un ulteriore fattore  $N$  che fa diventare l'errore come  $\sim N^{-2}$ . Qualunque sia il numero di punti che sto utilizzando, ho sempre due punti estremali, e quindi lo scaling del termine di errore che coinvolge le derivate prime è sempre  $h^2 \sim N^{-2}$ , la stessa legge di scala del rettangolo. Si tenga conto che, mentre con il rettangolo non si può migliorare, nel trapezio si può migliorare su come ottenere le derivate pari in funzione delle derivate dispari e spostare lo scaling dell'errore ad una potenza superiore. L'algoritmo risulta particolarmente efficiente nei casi di derivata seconda molto piccola e di derivate prime note agli estremi dell'intervallo. Se noi prendiamo la funzione  $f(x) = ax + b$ , le derivate prime le conosco e la derivata seconda è nulla, mi porta a un risultato corretto.

## 5.3 Algoritmo di Sympson

Per introdurre l'algoritmo di Sympson dobbiamo ragionare su due intervalli adiacenti, dunque consideriamo i punti  $x_{i-1}, x_i, x_{i+1}$ . Dovremo introdurre reticolazioni con un numero pari di intervalli.



Questo codice rappresenta correttamente gli assi, i punti sull'asse delle  $x$  e le righe tratteggiate verticali come richiesto. Le etichette per le righe tratteggiate e l'etichetta per l'intervallo sono posizionate in modo appropriato. Puoi regolare i valori, le posizioni e lo stile degli elementi secondo le tue preferenze.

$$\begin{aligned}
 I_{i-1,i+1} &= \int_{x_{i-1}}^{x_{i+1}} f(x) dx \\
 &= \int_{x_{i-1}}^{x_{i+1}} \left[ f(x_i) + f'_i(x - x_i) + \frac{1}{2} f''_i(x - x_i)^2 \right. \\
 &\quad \left. + \frac{1}{3!} f'''_i(x - x_i)^3 + \frac{1}{4!} f^{IV}_i(x - x_i)^4 \dots \right] dx \\
 &= \int_{x_{i-1}}^{x_{i+1}} \left[ f(x_i) + \frac{1}{2} f''_i(x - x_i)^2 + \frac{1}{4!} f^{IV}_i(x - x_i)^4 \dots \right] dx \\
 &= 2h f_i + \frac{1}{3} h^3 f''_i + O(h^5)
 \end{aligned}$$

Se noi andiamo a integrare sull'intervallo simmetrico attorno ad  $x_i$ , i termini dispari contribuiscono con zero all'integrale. Noi abbiamo il nostro integrale approssimato fino ad  $h^5$ . Noi dobbiamo esprimere la derivata seconda con un algoritmo della derivata che abbiamo utilizzato. Per esempio abbiamo visto che la derivata seconda

$$f''_i = \frac{1}{h^2} (f_{i+1} - 2f_i + f_{i-1})$$

Questa aveva una precisione di ordine  $h^2$ , quindi il nostro integrale

$$\begin{aligned}
 I_{i-1,i+1} &= 2h f_i + \frac{1}{3} h^3 \frac{1}{h^2} (f_{i+1} - 2f_i + f_{i-1}) + O(h^5) \\
 &= \frac{1}{3} h [6f_i + f_{i+1} - 2f_i + f_{i-1}] + O(h^5) \\
 &= \frac{1}{3} h [f_{i+1} + 4f_i + f_{i-1}] + O(h^5)
 \end{aligned}$$

Mi interessa tirare fuori la derivata seconda utilizzando i punti del reticolato utilizzando termini con precisione minore o uguale ad  $h^5$ , in questo modo esprimo l'integrale in termini di punti del reticolato. Questo fa riferimento al singolo intervallino doppio. Con questo algoritmo, prendendo due intervallini e riferendomi al punto di mezzo, esprimendo la derivata seconda in termini dei tre punti del reticolo ho spostato l'errore ad  $h^5$ . Quando vado a sommare l'integrale tra  $[a, b]$

$$\int_a^b f(x) dx = \sum_i^{2N} \int_{x_{i-1}}^{x_{i+1}} f(x) dx = \frac{1}{3} h [f_0 + 4f_1 + 2f_2 + 4f_3 + 2f_4 + \dots + 2f_{2N-2} + 4f_{2N-1} + f_{2N}] + O(h^4)$$

### 5.3.1 Stima dell'errore C-S

Nell'algoritmo di Cavalieri-Sympson, l'unico accorgimento che devo avere è di avere un numero pari di punti nel reticolato. Prendo i punti estremi con peso uguale, poi peso i punti interni pari due volte e i dispari quattro volte. Da implementare è molto facile ma ci dà una precisione di ordine  $N^{-4}$ , notevole e accurata, a meno di valori divergenti delle derivate successive. Si vede che la convergenza è molto forte, se ho 100 punti e un errore del 4%, se raddoppio i punti, l'errore è di 2.5 %, vuol dire che è molto efficiente.

### 5.3.2 Struttura algoritmo

Adesso l'algoritmo è semplicemente della forma di fare la partizione, prendere i valori della funzione nei punti del reticolato e pesarli con dei pesi  $\omega_i$ , che sono  $\frac{1}{3}$  per gli estremi,  $\frac{4}{3}$  per i punti dispari e  $\frac{2}{3}$  per i punti pari.

$$I = \sum_i \omega_i f(x_i)$$

In base ai pesi che adotto ottengo precisioni diverse. A punti alterni li peso con  $\omega_i$ . Questo tipo di pesi ottiene una precisione migliore, come posso sviluppare la serie di pesi della funzione?

$$I_{a,b}(N) = \frac{1}{3}h\{S_d(N) + 2S_e(N) + 4S_o(N)\}$$

Dove i nomi delle serie stanno per  $S_d = f_0 + f_N$ ,  $S_e$  la somma degli *even* =  $f_2 + f_4 + \dots$ ,  $S_o$  è la somma degli *odd* =  $f_1 + f_3 + f_5 + \dots$  è un'altra forma dell'algoritmo scritto in precedenza. Il numero di intervalli è pari e il numero di intervalli è dispari. Se da una partizione a 6 intervalli voglio raddoppiare il numero di intervalli, devo rinumerare i punti  $0 \rightarrow 0, 1 \rightarrow 2, 2 \rightarrow 4, 3 \rightarrow 6, \dots 6 \rightarrow 12$ . Noi abbiamo già il valore della funzione nei nuovi punti pari, serve il valore della funzione nei nuovi punti dispari. Dobbiamo costruire la somma di Sympson con l'algoritmo del trapezio.

$$S_e(N_2) = S_e(N_1) + S_o(N_1)$$

Alla fine, quello che si ottiene è che la somma di Sympson, fatta con  $2N$  punti è uguale a

$$S_{\text{Sympson}}(2N) = \frac{4}{3}S_{2N}^{\text{Trap.}} - \frac{1}{3}S_N^{\text{Trap.}}$$

Questo risultato si ricava in seguito. Con Sympson, quando faccio un algoritmo numerico, faccio un integrale con 100 punti, non so se il valore a 200 punti sarà convergente. A seconda dei valori delle derivate della funzione dovrò fare attenzione a quale algoritmo utilizzare. C'è una nuova classe di algoritmi che si possono utilizzare, che sono gli algoritmi adattivi.

## 5.4 Algoritmi adattivi

Un'applicazione del Machine Learning è quello di aver sfornato gli algoritmi adattivi, algoritmi che capiscono quando è meglio utilizzare un algoritmo oppure un altro. In alcune porzioni di una funzione potrei utilizzare uno spacing grande, oppure uno spacing piccolo, e l'**algoritmo Vegas** lo fa molto bene. Lo vedremo con tecniche derivate dagli algoritmi Monte Carlo e applicate agli algoritmi Deterministici che si può variare la densità di punti della funzione nelle varie zone.

# Chapter 6

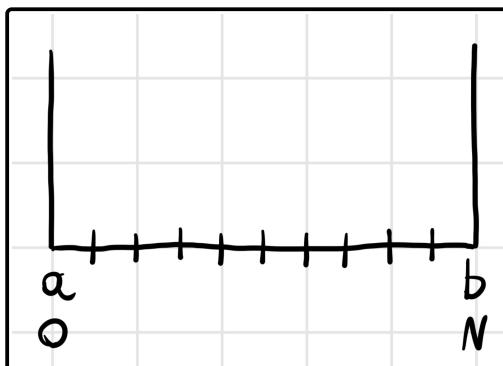
## Lezione 6

### 6.1 Algoritmo d'integrazione del trapezio

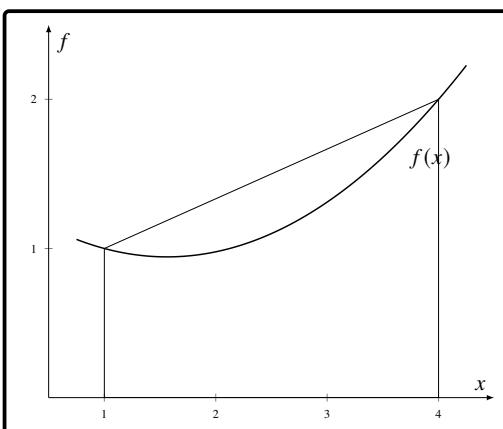
I codici dell'integrazione si trovano sotto *integration* al [link](#). Con l'algoritmo di Sympson scala come  $N^{-4}$ . Si può pensare ad una applicazione iterativa dell'algoritmo di Sympson come differenza di due algoritmi del trapezio.

$$\int_a^b f(x)dx = I_{\text{trapezio}} + \frac{1}{4}h^2 \left[ f'_0 - f'_N \right] + \frac{1}{12}h^3 \left[ f''_0 + 2f''_1 + \dots + 2f''_{N-1} + f''_N \right] \dots$$
$$I_{\text{trapezio}} = \sum_i I_{i-1,i} = h \left[ \frac{1}{2}f_0 + f_1 + \dots + f_{N-1} + \frac{1}{2}f_N \right]$$

Quando il risultato analitico non c'è dobbiamo inventarci un metodo per assicurarci la bontà del risultato numerico.



Ho  $N$  intervalli. Se pongo  $N = 1$



approssimo l'integrale con l'area del trapezio. L'algoritmo è racchiuso nella subroutine. In fortran qualunque quantità è sia input che output. Il valore che chiamo con la subroutine viene passato all'interno della subroutine. Facciamo la chiamata con un solo intervallo e ci restituisce un valore  $S$ , il controllo viene passato fuori dalla routine. Il valore  $S$  ottenuto viene richiamato dalla routine che poi porta  $N \rightarrow 2$  ed esegue il giro successivo. Si può eseguire con la precisione preferita. **real\*8** e **double** sono equivalenti. Compare l'istruzione **save**. Quando faccio il calcolo in una struttura di tipo function o subroutine e passo il controllo all'esterno della subroutine, le variabili locali della subroutine vengono cancellate, e non esistono più. Con l'istruzione **save** sto salvando il valore delle variabili anche per quando rientro. Tutto è condizionato dal valore di  $N$ , quindi la prima volta, la cosa più semplice che possiamo fare è mettere  $N = 1$ , quindi l'algoritmo calcola

$$s = \frac{1}{2}(b - a)(f(a) + f(b))$$

### Input di $N = 2$

**icount** è messo lì per ricordare il numero di chiamate della funzione. Si supponga di chiamare la subroutine con  $N = 2$ , devo mettere un punto interno, perciò utilizzo la variabile  $it = 2^{N-2}$ ,  $tnm = it$  e definisco l'ampiezza del singolo intervallo  $del = \frac{b-a}{tnm}$ . Con un punto in mezzo, il  $\delta$  sarà  $\frac{b-a}{1}$ , ma  $x = a + \frac{1}{2}del$ . Nel programma, per  $del$  si intende tutto l'intervallo precedente e intende  $x$  come l'ascissa del nuovo punto. Deve poter funzionare con  $N$  generico. La potenza di come è scritto è che funziona per  $N$  generico iterativamente. C'è questo ciclo *do* su  $j = 1, it$ , per  $N = 2$  è un ciclo da 1 a 1.  $sum =$  valore precedente  $+ func(x)$ , che in questo caso è uguale al punto di mezzo. Alla fine

$$s = \frac{1}{2} \left[ s + (b - a) \frac{sum}{tnm} \right]$$

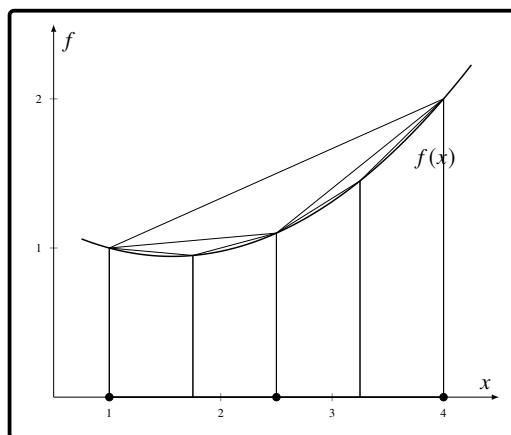
Abbiamo detto che con  $N = 2$  si ha  $sum = f(x)$  e  $tnm = 1$ . Il valore di  $s$  precedente era  $s = \frac{1}{2}(b-a)(f(a)+f(b))$ . Alla fine il controllo passa il valore di **icount** in **jcount**. Il controllo torna per capire quante chiamate della funzione ho utilizzato.

$$s = \frac{1}{2}(s + (b - a)f(x)) = \frac{1}{2}(b - a) \left( \frac{1}{2}f_a + f(x) + \frac{1}{2}f_b \right)$$

$icount = icount + it$ , dove **icount** era salvato alla prima chiamata  $it = 2$ , perciò  $jcount = 3$ . Di nuovo, deve funzionare per  $N$  generico.

#### 6.1.1 Input di $N = 3$

Noi abbiamo già chiamato per  $N = 2$ . Dobbiamo aggiungere l'informazione dei punti medi dei vecchi intervalli.  $\delta$ , per  $N = 3$ , è uguale a  $\frac{b-a}{2}$ . Mi sposto di metà di quell'intervallo, poi faccio l'iterazione e mi sposto nell'altro punto nuovo, che dista esattamente  $\delta$  dal precedente. Ciò succede alla seconda iterazione del ciclo *do*.



Parto con  $j = 1$  e dico che  $sum = sum + func(x)$ , poi  $x = x + \delta$ , e ricalcolo la somma. Se vado avanti con  $N = 4$ , aggiungo 4 punti addizionali e ricalcolo  $\delta$ . La prima volta mi piazzo nel primo punto, che dista  $\delta/2$  da  $a$ , e poi con salti di  $\delta$  aggiungo i valori della funzione su tutti i punti aggiuntivi.  $it = 4$  e il ciclo va a prendere tutti i punti dove, ogni volta, in questo caso, il  $\delta_4 = \frac{b-a}{4}$ . Scritto in questo modo l'algoritmo si presta

al calcolo con  $N$  qualunque. La scrittura compatta di una decina di righe, presuppone che io abbia calcolato tutti i punti da  $1 \rightarrow N - 1$ . Posso utilizzarla direttamente con  $N = 10$  e  $2^{10}$  intervalli. Uno si deve domandare perché strutturare la subroutine che può essere utilizzata in modo iterativo? Il motivo è che noi dobbiamo avere una stima dell'errore che commettiamo. Abbiamo visto che analiticamente l'errore scala come  $h^2$ .

### 6.1.2 Stabilità dell'algoritmo

Se noi utilizziamo l'algoritmo alla cieca con un certo  $N$ , non conosciamo l'andamento dell'errore. Ci sono due scenari.

- Conosciamo l'espressione analitica dell'integrale. Valuto la bontà dell'algoritmo sulla base di un benchmark e mi assicuro che si scosti il meno possibile da esso. Se facciamo la routine con  $n = 5$  otterremo un certo risultato  $I_5$ , che differirà da quello analitico quello che possiamo verificare è lo scaling se passiamo da  $n = 5 \rightarrow n = 6$ , i punti raddoppiano.  $I_6$  dovrebbe migliorare lo scostamento dal valore analitico di un valore  $\frac{1}{4}$ , invece  $I_7$  migliora di un valore  $\frac{1}{16}$ . Per questo motivo, in questa routine c'è l'output *jcount*, che è l'analogico di  $N$ , ed è l'indice che determina la precisione dell'algoritmo. Devo far attenzione al numero di chiamate della funzione. Da questo punto di vista mi fermo quando la stima dell'integrale numerico che si discosta dal valore analitico per meno dell'1%.
- Supponiamo di dover fare questo integrale, che per qualche motivo ci serve in un problema con una certa precisione, come all'1%. Il problema è, che facendo iterazioni successive, come faccio a sapere quando feramme l'algoritmo? Quando mi accontento della stima dell'integrale che ho? Chiaramente, se vado con  $N \rightarrow \infty$  l'integrale fornisce il risultato numerico con precisione infinita. Nella pratica, un algoritmo come questo non è particolarmente pesante dal punto di vista computazionale da calcolare, eppure ci sono casi più complicati, perché determinati algoritmi possono racchiudere in sé un insieme complicato quanto si vuole di calcoli da effettuare. Bisogna fare attenzione al numero di chiamate da utilizzare della funzione. Devo dare una stima dell'integrale con una certa affidabilità che il valore della nostra stima sia distante dal valore vero per  $\pm 1\%$ . L'unica cosa che ho è quella di fare determinate chiamate con determinati numeri di punti. Devo vedere quanto distino le stime per  $N$  ed  $\frac{N}{2}$  punti. Se distano meno dell'1% io posso essere soddisfatto del valore calcolato con  $N$ . Nel fare questo devo fare attenzione al fatto che questo è solo un criterio di convergenza e non un teorema. Questi due valori possono essere accidentalmente molto vicini ma magari lontani dal valore dell'integrale. Aggiungiamo una ragionevolezza aggiuntiva. Devo controllare che lo scaling dell'errore per  $\frac{N}{8}, \frac{N}{4}, \frac{N}{2}$  rispetti l'andamento di  $\frac{1}{N^2}$  che ho dall'algoritmo del trapezio. Questi sono alcuni parametri con cui giocare per rispondere alla domanda *come faccio a capire la bontà di un risultato numerico?* se non ho il valore analitico di riferimento.

$$|S_N - S_{N/2}| \leq 1\%$$

È probabile che  $|S_N - S| \leq 1\%$ . Non è una dimostrazione matematica, è un criterio di ragionevolezza. Può capitare che casualmente per l' $N$  scelto la differenza sia molto piccola. Se faccio un'altra reticolazione e faccio

$$|S_{2N} - S_N| \leq 1\%$$

È probabile che diverga. Per evitare questa possibilità un'altra informazione a cui si possiamo appigliare è andare a vedere se  $S_{N/8} \rightarrow S_{N/4} \rightarrow \dots$  rispetta lo scaling degli errori di  $\frac{1}{N^2}$ . Se l'algoritmo comincia a divergere vuol dire che non sto rispettando l'andamento dell'errore. Può succedere se la funzione è smooth in certi punti e in una piccola zona dell'intervallo di integrazione è più variante. Con i reticolati non ho visto questo punto e ho visto dei valori della funzione abbastanza vicini tra di loro. Quando faccio  $S_{2N}$  mi vedo una fluttuazione grande dell'integrale. È utile vedere che siamo in una regione dove c'è una stabilità dell'andamento della legge di  $\frac{1}{N^2}$  e provare un valore successivo. Tutto questo è implementato in **qtrap.f**.

## 6.2 Richiesta di precisione nell'algoritmo

**qtrap.f** contiene la richiesta di precisione dell'algoritmo e si stoppa quando raggiunge il limite di precisione. C'è la richiesta che  $\varepsilon$ , cioè la variabile *eps* è la richiesta di precisione relativa. Potrei strutturare la routine

in modo tale che ragioni in precisione assoluta. La precisione relativa dipende da quanto è grande l'integrale. Ragiono sulla stima della variabile al giro precedente. Faccio un loop sull'algoritmo trapezoidale, inserisco in input la richiesta di precisione e il massimo numero di chiamate successive che faccio con l'algoritmo trapezoidale. Mi fermo prima di  $jmax$ , oppure mi accorgo che esaurisce le reticolazioni massime e non ha ancora soddisfatto la richiesta di precisione. Ogni volta che itero la nuova chiamata del trapezio eseguo il controllo di precisione tra  $s - olds$ . Se questo è soddisfatto allora il controllo esce e abbiamo una stima accettabile. Non facciamo questo controllo subito da  $j = 1$ , perché meno chiamate ho nelle varie iterazioni, tanto più sono esposto ad errore. Avrei una stima totalmente sbagliata e da questo punto di vista l'algoritmo fa questo controllo dopo che ho calcolato almeno 5 reticolazioni, sto usando  $2^6$  punti almeno. Se il controllo, che è su numeri *floating point*, fosse  $\leq$  non avrebbe molto senso, perché le cifre dopo la virgola possono essere diverse dopo la quindicesima. Ogni stima è sempre 0. Questo controllo mi dice che se le due stime sono identicamente nulle, ritorno il risultato. Se non è rispettata, aggiorno la vecchia stima con  $s$  e passo ad un  $j$  successivo, l'algoritmo fa un'altra iterazione. Qui è inizializzata a  $-1.e30 = 10^{-30}$ . Questo numero viene aggiornato con la prima stima vera dell'integrale. Se faccio questi giri e non trovo mai soddisfatta la richiesta che le stime successive siano più vicine della precisione richiesta, alla fine del ciclo ho una stima di  $s$ , ma scrivo un **warning** che ho raggiunto il fondo scala sul numero di step nel trapezio. Questo è un sintomo del fatto che il valore della stima che ho dell'integrale non è tanto credibile in merito alla precisione target che avevo richiesto.

## 6.3 Algoritmo di integrazione di Sympson

Nell'algoritmo **qsymp.f** si ragiona sull'errore in modo analogo all'algoritmo del trapezio, eppure per Sympson si vede che può essere scritta come combinazione lineare di due iterazioni successive del trapezio. Qui la usiamo perché così la routine di Sympson si appoggia alla routine del trapezio, facciamo due chiamate successive e le utilizziamo per comporre la combinazione lineare

$$S_{\text{Sympson}}(2N) = \frac{4}{3}S_{2N}^{\text{Trap.}} - \frac{1}{3}S_N^{\text{Trap.}} = \frac{1}{3}(4 \cdot st - ost)$$

Con questa combinazione lineare otteniamo un fattore di convergenza  $\frac{1}{N^4}$ .

### 6.3.1 Driver integration

Ci sono una serie di cose che non servono, perché andiamo a confrontare gli algoritmi quando implementiamo il metodo Monte Carlo. La richiesta che abbiamo è di inserire in input un confronto tra trapezio in modo iterativo, oppure quando lo regoliamo sulla precisione richiesta con un massimo numero di chiamate. Se noi non diamo un massimo numero di chiamate e ragioniamo solo in termini di precisione, e la richiesta non viene mai incontrata, l'algoritmo non si ferma più. È bene avere un numero massimo di chiamate. Se arrivo a fondo scala, è bene fare un check, provando ad aumentare il numero di chiamate. Dobbiamo mettere in input gli estremi inferiori e superiori dell'integrazione e adesso facciamo un ciclo di chiamate sul trapezio, per vedere lo scaling degli errori da  $N \rightarrow N + 1$ . Vediamo come evolve in base alla reticolazione. Infine stampiamo il risultato di **Sympson**. Ci sono alcune funzioni da testare, proviamo con

$$\int_a^b x^2 dx \rightarrow \frac{b^3 - a^3}{3}$$

$$\int_a^b x^2 \cos(x) dx \rightarrow (x^2 - 2) \sin(x) + 2x \cos(x) \Big|_a^b$$

#### Funzione quadratica

Cominciamo con  $n = 10$ ,  $a = 0$ ,  $b = 5$  e una precisione dell'1%. La perdita per round-off non è così importante per gli integrali. È importante nelle derivate perché faccio differenze tra quantità vicine. Il massimo numero di chiamate lo setto a un integer, che ha il massimo intero rappresentabile impostato a  $2 \cdot 10^9$ , perciò lo possiamo impostare a 100000.

```

analytical value = 41.66666666666664
results from trapzd
n. of points= 2 integral= 62.50000000000000
n. of points= 3 integral= 46.87500000000000
n. of points= 5 integral= 42.96875000000000
n. of points= 9 integral= 41.99218750000000
n. of points= 17 integral= 41.74864687500000
n. of points= 33 integral= 41.68701171875000
n. of points= 65 integral= 41.671752929687500
n. of points= 129 integral= 41.667938232421875
n. of points= 257 integral= 41.666984558105469
n. of points= 513 integral= 41.666746139526367
n. of points= 1025 integral= 41.666686534881592

result from qtrap
n. of points= 33 integral= 41.687011718750000

result from qsymp
n. of points= 33 integral= 41.666666666666664

```

Vediamo come evolve la stima dell'integrale con algoritmo trapezoidale. Ho con 1025 punti 4 cifre significative buone. È vero che passando da una reticolazione all'altra l'errore migliora. Prendiamo da  $257 \rightarrow 1025$  un fattore  $\sim 4$  di punti. Se prima avevamo un errore di 3 sulla quarta cifra, mi aspetto che quadruplicando il numero di punti l'errore migliori di un fattore 16, quindi dovrebbe essere

$$\frac{3}{100000} \rightarrow \frac{2}{4000000}$$

L'errore è diminuito e sono quasi come nello scaling. Il trapezio si ferma a 33 chiamate, perché trova che la distanza dal valore analitico è  $\leq 1\%$ . Apparentemente Sympson ci ha messo lo stesso numero di punti, ma l'integrale è esatto. Qui, bisogna chiedersi **perché non si sia fermato prima**. C'è una corrispondenza allo stesso numero del valore analitico. Poiché l'algoritmo confronta due valori successivi dell'integrale e l'errore dovrebbe scalare come  $N^{-4}$ , la versione precedente aveva un numero di punti dimezzato, eppure l'errore  $2^4$  è comunque minore di 1%. Nel controllo sulla precisione si fa un numero minimo di 5 iterazioni, in modo tale che ci sia minore probabilità di cadere in cause accidentali di stop dell'algoritmo. Perciò il controllo comincia a farlo dal sesto step, ecco spiegato il numero di 33 chiamate. Quello che stupisce è comunque l'uguaglianza a tutte le cifre. Il motivo è che con un polinomio quadratico, l'algoritmo di Sympson lo integra esattamente con un numero finito di punti, perché si può vedere nello sviluppo teorico dell'algoritmo che abbiamo tralasciato i termini dalla derivata terza in poi. Di fatto è come se avessimo approssimato la funzione con degli archi di parabola tra un intervallo e l'altro. Quindi, se prendiamo una funzione quadratica l'algoritmo lo integra esattamente. Gli algoritmi di Gauss riescono ad integrare esattamente una classe più ampia di funzioni con un numero finito di punti.

### Funzione oscillante

```

analytical value = -19.218636462619923
results from trapzd
n. of points= 2 integral= 17.72886591451641
n. of points= 3 integral= -3.6534256971950185
n. of points= 5 integral= -15.634743812778717
n. of points= 9 integral= -18.340221764454444
n. of points= 17 integral= -19.000103703019107
n. of points= 33 integral= -19.164069793019383
n. of points= 65 integral= -19.204998946313534
n. of points= 129 integral= -19.215227342886340
n. of points= 257 integral= -19.217784198893899
n. of points= 513 integral= -19.218423397701358
n. of points= 1025 integral= -19.218583196453590

result from qtrap
n. of points= 33 integral= -19.164069793019383

result from qsymp
n. of points= 33 integral= -19.218725156352807

```

Vediamo che ci si ferma a 33 chiamate con l'algoritmo del trapezio e di Sympson. Vediamo cosa succede se setto la precisione a  $10^{-4}$ , con gli stessi parametri

```

analytical value = -19.218636462619923
results from trapzd
n. of points=      2    integral=   17.728886591451641
n. of points=      3    integral=   -3.6534256971950185
n. of points=      5    integral=   -15.634743812778717
n. of points=      9    integral=   -18.340221764454444
n. of points=     17    integral=   -19.000103703019107
n. of points=     33    integral=   -19.164069793019383
n. of points=     65    integral=   -19.204998946313534
n. of points=    129    integral=   -19.215227342886340
n. of points=    257    integral=   -19.217784198893899
n. of points=    513    integral=   -19.218423397701358
n. of points=   1025    integral=   -19.218583196453590

result from qtrap
n. of points=    513    integral=   -19.218423397701358

result from qsymp
n. of points=     33    integral=   -19.218725156352807

```

Sympson si ferma di nuovo a 33 e la differenza è  $\frac{5}{1000000}$ , mentre il trapezio fatica un pò di più e deve fare 4 reticolazioni in più. Questo è il difetto dello scaling di  $N^{-4}$ .

# Chapter 7

## Lezione 7

### 7.1 Integrali impropri

Abbiamo studiato algoritmi di integrazione deterministici e abbiamo visto come funzionano con integrali definiti senza alcuna particolare attenzione.

$$\int_{-\infty}^{\infty} f(x) dx$$
$$PV \int f(x) dx$$

Si affrontano integrali di questo tipo con funzioni che conosciamo bene. Quando si studia il corpo nero, si considera la radiazione totale messa, da cui deriva la legge di Stefan-Boltzmann  $\sigma \sim T^4$

$$\int_0^{\infty} \frac{x^3}{e^x - 1} dx = \frac{\pi^4}{15}$$

Si può risolvere in campo reale, senza andare al campo complesso e si vede come fare, nel senso che dal punto di vista numerico sono necessari alcuni accorgimenti. Se proviamo a inserire questo integrale nel nostro programma, cosa ci aspettiamo? Possiamo integrare fino a un certo cutoff e prendiamo i valori dell'integrale al variare del cutoff, cercando una saturazione, compatibilmente con la precisione che utilizziamo. Dobbiamo partire da ipotesi di convergenza della funzione. Se la funzione non decresce rapidamente, l'integrale non esiste. In questo caso, con un polinomio diviso un esponenziale crescente, dominerà la funzione esponenziale e infatti deriva dallo spettro di Planck.

$$\int_0^C \frac{x^3}{e^x - 1} dx$$

La convergenza è assicurata, proviamo a vedere mettendoci la nostra funzione nel file **funzione.f** con  $x = \frac{\hbar\omega}{kT}$  variabile adimensionale. Eseguiamo ora il file **integration.sh**. Impostiamo 10 iterazioni, accuratezza di  $10^{-4}$  e un massimo numero di chiamate di  $10^5$ . Gli estremi di integrazione li impostiamo come 0,  $C$ . Per determinare un contributo all'integrale, la funzione deve avere un valore superiore alla machine accuracy, perciò  $x^3 e^{-x} \leq \varepsilon$ . Prima di raggiungere l'underflow in doppia precisione, che è circa  $10^{-300}$ , corrisponderà a una certa  $x$ , prima di quello ci saranno  $X$  per cui l'integrandino è  $10^{-16}$  e qui stiamo sommando nella regione dell'ordine dell'unità. A quel punto, da lì in poi stiamo sommando cose che vengono registrate non diverse da zero.

```

analytical value = -19.218636462619923
results from trapzd
n. of points= 2 integral= NaN
n. of points= 3 integral= NaN
n. of points= 5 integral= NaN
n. of points= 9 integral= NaN
n. of points= 17 integral= NaN
n. of points= 33 integral= NaN
n. of points= 65 integral= NaN
n. of points= 129 integral= NaN
n. of points= 257 integral= NaN
n. of points= 513 integral= NaN
n. of points= 1025 integral= NaN

result from qtrap

```

Il valore analitico è quello della precedente funzione. Ad ogni modo sono venuti fuori i **NaN**, sulle iterazioni successive del trapezio. Lui si ferma quando il controllo tra un valore e il precedente è inferiore alla precisione richiesta, siccome sta confrontando dei **NaN** va in fondo al numero massimo di chiamate.

### 7.1.1 Ridefinizione funzione

L'estremo sinistro in 0 ha 0/0, che analiticamente è 0, però quando viene calcolata la funzione in questo modo, non viene fatta la semplificazione analitica, vado sull'estremo perché l'algoritmo del trapezio, selezionati gli estremi  $a, b$ , prende esattamente gli estremi, calcola  $f(0) + f(5)$ . Un'altra possibilità è quella di andare a sostituire  $0 \rightarrow c_1$ , in questo caso la funzione la conosciamo e possiamo impostare  $\lim_{x \rightarrow 0} \frac{x^3}{e^x - 1} = x^2$ . Possiamo sostituirci la funzione con il suo sviluppo vicino a 0. Dobbiamo farlo in una zona sufficientemente piccola, in modo tale che la sostituzione non ci fa perdere valore dell'integrale. Nel punto di cutoff non si devono generare forti discontinuità, devono essere al di sotto di una certa soglia. Dobbiamo correggere la funzione all'interno dell'algoritmo **funzione.f**. Possiamo scrivere  $c1 = 1.d - 5$

```

if(x - lt.c1) then
    func = x * *2
else
    func = x * *3/(exp(x) - 1.d0)
endif

```

Questa informazione si utilizza per la reticolazione. Il discorso che si fa è indipendente dal linguaggio. Un conto è dire, se trovo 0 metto l'altro valore, un conto è dire aumentare la precisione per ritardare un problema analitico, che permane sempre. Ci sono programmi in **Mathematica** che permettono di variare la precisione in base all'instabilità della funzione. Reimpostiamo l'algoritmo con gli stessi parametri

```

results from trapzd
n. of points= 2 integral= 2.1198921582200723
n. of points= 3 integral= 4.5531292757448805
n. of points= 5 integral= 4.8444923476339135
n. of points= 9 integral= 4.88806080343489620
n. of points= 17 integral= 4.8970440790167205
n. of points= 33 integral= 4.8991878242613005
n. of points= 65 integral= 4.8997165551928497
n. of points= 129 integral= 4.8998482875701788
n. of points= 257 integral= 4.8998811925169861
n. of points= 513 integral= 4.8998894169944638
n. of points= 1025 integral= 4.8998914730038816

result from qtrap
n. of points= 129 integral= 4.8998482875701788

result from qsymp
n. of points= 33 integral= 4.8999024060094944

```

Otteniamo dei numeri e il problema è determinare se sono corretti, e a che livello di precisione sono corretti. La differenza tra trapezio e Sympson è di  $10^{-4}$ , sembrerebbe soddisfatta la richiesta di precisione sull'intervallo (0, 5). Cosa ci mettiamo per modificare i risultati? Poniamo come estremi (0, 10000)

```

results from trapzd
n. of points=      2    integral=  0.0000000000000000
n. of points=      3    integral=  0.0000000000000000
n. of points=      5    integral=  0.0000000000000000
n. of points=      9    integral=  0.0000000000000000
n. of points=     17    integral=  5.6165403057888796E-261
n. of points=     33    integral=  1.8296772660587170E-126
n. of points=     65    integral=  8.2559548015749556E-060
n. of points=    129    integral=  4.3843349215450598E-027
n. of points=    257    integral=  2.5258754939840589E-011
n. of points=    513    integral=  4.7929849940454817E-004
n. of points=   1025    integral=  0.52223674592888136

result from qtrap
n. of points=  32769    integral=  6.4939032621944133

result from qsymp
n. of points=  65537    integral=  6.4939484372849323

```

Non sembra mostrare underflow.

### 7.1.2 Bassa densità di punti

Dobbiamo variare il numero di iterazioni e il massimo numero di chiamate 100000 e per estremo superiore di integrazione  $10^7$ .

```

results from trapzd
n. of points=      2    integral=  0.0000000000000000
n. of points=      3    integral=  0.0000000000000000
n. of points=      5    integral=  0.0000000000000000
n. of points=      9    integral=  0.0000000000000000
n. of points=     17    integral=  0.0000000000000000
n. of points=     33    integral=  0.0000000000000000
n. of points=     65    integral=  0.0000000000000000
n. of points=    129    integral=  0.0000000000000000
n. of points=    257    integral=  0.0000000000000000
n. of points=    513    integral=  0.0000000000000000
n. of points=   1025    integral=  0.0000000000000000

result from qtrap
n. of points=    33    integral=  0.0000000000000000

result from qsymp
n. of points=    33    integral=  0.0000000000000000

```

Vengono tutti 0, eppure non abbiamo raggiunto ancora  $10^{-300}$ . Perché tutti i risultati sono nulli? Perché per sommare i vari punti distribuiti da 0 a  $10^7$ , somma tutti 0. Sta sommando numericamente degli 0. Perché abbiamo perso i valori diversi da 0? Perché  $10^7/1025 \sim 10^4$ , e forse la funzione per  $x \sim 10^4$  è 0 per la precisione della macchina. Per  $b = 10^7$  servirebbero moltissimi intervalli. Dobbiamo vedere quanto più di 5 è il  $b$  corretto.

### 7.1.3 Impostiamo un estremo superiore ragionevole

Evitiamo di confrontare l'integrale con  $\frac{\pi^4}{15}$  per costruire una teoria che si basa solo sull'algoritmo numerico. Impostiamo, per esempio 10.

```

results from trapzd
n. of points=      2    integral=  0.22700995504843885
n. of points=      3    integral=  4.3532892939643641
n. of points=      5    integral=  6.2534818251520816
n. of points=      9    integral=  6.4176528710869292
n. of points=     17    integral=  6.4302541309221137
n. of points=     33    integral=  6.4316236707737442
n. of points=     65    integral=  6.4318547599553693
n. of points=    129    integral=  6.4319055763043114
n. of points=    257    integral=  6.4319178456455068
n. of points=    513    integral=  6.4319208858091894
n. of points=   1025    integral=  6.4319216441518829

result from qtrap
n. of points=    65    integral=  6.4318547599553693

result from qsymp
n. of points=    65    integral=  6.4319317896825767

```

I valori che escono sono convergenti, in questo caso  $10^{-4}$  è ottenuto con 65 chiamate per ciascun algoritmo. Ci si aspetta che Sympson sia molto più veloce, ma il valore a 33 punti di Sympson non raggiungeva il valore di  $10^{-4}$  di precisione. Il numero è variato notevolmente da  $\sim 4.89 \rightarrow 6.43$ . Non siamo ancora a saturazione. Si può provare ad aumentare. Si imposta  $b = 20$ .

```
results from trapzd
n. of points=      2    integral=   1.6489229013495296E-004
n. of points=      3    integral=   0.45410235624194517
n. of points=      5    integral=   4.4719975977986843
n. of points=      9    integral=   6.3313690282344659
n. of points=     17    integral=   6.4837458613851373
n. of points=     33    integral=   6.4932839428895184
n. of points=     65    integral=   6.4938803295866308
n. of points=    129    integral=   6.4939176679140616
n. of points=    257    integral=   6.4939200175993790
n. of points=    513    integral=   6.4939201684646761
n. of points=   1025    integral=   6.4939201788962473

result from qtrap
n. of points=     65    integral=   6.4938803295866308

result from qsymp
n. of points=    129    integral=   6.4939301140232049
```

Si giunge al valore di  $\sim 6.49$ . Con Sympson si effettuano più chiamate per raggiungere la precisione desiderata. Rispetto al valore precedente è variato dell'1%. Importiamo il valore analitico dell'integrale improprio e lo confrontiamo con i valori ottenuti dall'integrale.

```
analytical value = 6.4939393725851993
results from trapzd
n. of points=      2    integral=   1.6489229013495296E-004
n. of points=      3    integral=   0.45410235624194517
n. of points=      5    integral=   4.4719975977986843
n. of points=      9    integral=   6.3313690282344659
n. of points=     17    integral=   6.4837458613851373
n. of points=     33    integral=   6.4932839428895184
n. of points=     65    integral=   6.4938803295866308
n. of points=    129    integral=   6.4939176679140616
n. of points=    257    integral=   6.4939200175993790
n. of points=    513    integral=   6.4939201684646761
n. of points=   1025    integral=   6.4939201788962473

result from qtrap
n. of points=     65    integral=   6.4938803295866308

result from qsymp
n. of points=    129    integral=   6.4939301140232049
```

Sympson si avvicina di molto al valore analitico. Con la nostra esigenza di  $10^{-4}$  siamo soddisfatti dal valore ottenuto con Sympson. Supponiamo di non conoscere il valore analitico e basiamoci solo sul valore numerico. Il numero che mi dà  $10^{-4}$  con il trapezio è quello relativo a 65 chiamate, eppure è ancora lontano al valore più vicino al vero, che emerge dalle iterazioni successive. Il trapezio si è fermato accidentalmente a 65 chiamate. Anche il trapezio, seppur più lentamente, si avvicina al valore che determina l'algoritmo di Sympson. possiamo osservare che se proviamo ad utilizzare  $b = 30$

```
analytical value = 6.4939393725851993
results from trapzd
n. of points=      2    integral=   3.7898373023806254E-008
n. of points=      3    integral=   1.5486328661889521E-002
n. of points=      5    integral=   1.7587195627671928
n. of points=      9    integral=   5.7117242721383956
n. of points=     17    integral=   6.4424421041067701
n. of points=     33    integral=   6.4967207484877132
n. of points=     65    integral=   6.4937382337525786
n. of points=    129    integral=   6.4939268266050529
n. of points=    257    integral=   6.4939386136641861
n. of points=    513    integral=   6.4939393503568468
n. of points=   1025    integral=   6.4939393964005019

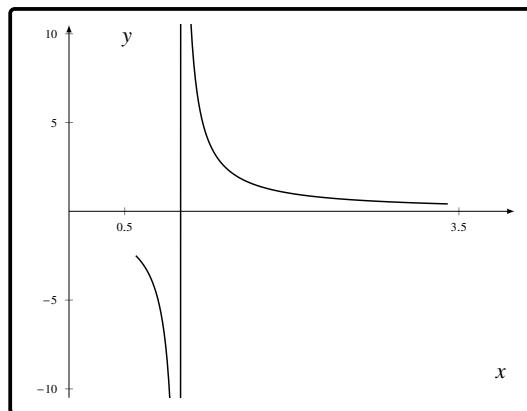
result from qtrap
n. of points=    129    integral=   6.4939268266050529

result from qsymp
n. of points=    257    integral=   6.4939425426838966
```

Ci stiamo muovendo intorno a dove eravamo prima, numericamente, anche se siamo a  $\frac{3}{2}$  del dominio in precedenza adottato. Sympson è andato in alto, le chiamate sono andate avanti con un'iterazione successiva e quindi, a seconda dell'estremo superiore che adottiamo, la reticolazione cambia e il valore dell'integrale che viene fuori cambia. Può essere utile adottare un numero maggiore di chiamate. Ad ogni modo riusciamo ad individuare un risultato numerico senza adottare il teorema di Cauchy. Con  $b = 10^4$  si ricorda che il trapezio converge con minore velocità. Tutto ciò per il corpo nero. Si può studiare il caso del valore principale dell'integrale.

## 7.2 Valore Principale dell'integrale

$$PV \int_a^b \frac{f(x)}{x - c} dx, \quad c \in [a, b]$$



Il valore principale è definito come

$$\lim_{\varepsilon \rightarrow 0} \int_a^{c-\varepsilon} \frac{f(x)}{x - c} dx + \int_{c+\varepsilon}^b \frac{f(x)}{x - c} dx$$

Ci sono tante quantità in fisica che sono definite con l'integrale al valore principale. Bisogna riconoscere dove si trova la singolarità.  $(x - c)^{2n+1}$  può essere elevato a una potenza dispari, affinché l'integrale esista. Possiamo calcolare l'integrale come somma dei due integrali, individuando dei valori di  $\varepsilon$  finiti ma tendenti a 0, con tutti i caveat che abbiamo appreso sul calcolo. Facciamo l'integrale con uno dei metodi che abbiamo visto e il risultato sarà la somma dei due integrali. C'è un possibile problema, che i due integrali, presi separatamente, divergono. Il risultato finito si ha dalla cancellazione delle divergenze che ho a destra e sinistra. Questo succede perché, per come è definito l'integrale del valore principale mi avvicino al valore di limite in modo simmetrico tra destra e sinistra rispetto a  $c$ , se non mi avvicinassi in modo simmetrico, l'integrale non esisterebbe. In questo caso il problema potenziale è che se io faccio la somma di questi due integrali, con  $\varepsilon$  piccolo, ma finito, esistono senza problemi. Eppure posso avere un numero  $-A$  e  $+B$  come risultati dell'integrazione. Questi numeri diventano sempre più grandi in valore assoluto e le routine numeriche mi danno un valore  $-A \pm \delta A$  e  $B \pm \delta B$ . Se ho una precisione relativa come  $\frac{\delta A}{A} = 10^{-4} = \frac{\delta B}{B}$ , e calcolo

$$\frac{-A + B}{B} \sim 10^{-3}$$

Questo vuol dire che la mia stima del valore principale dell'integrale

$$PV \int_a^b \frac{f(x)}{x - c} dx \sim 10\%$$

Questo perché ho richiesto una precisione di  $10^{-4}$  sui singoli integrali. Questo problema diventa acuto tanto più prendo  $\varepsilon$  piccolo, perché i valori di  $A$  e  $B$  diventano grandi e facendo la somma il risultato diventa molto più piccolo. Se prendiamo il

$$PV \int_{-a}^{+a} \frac{1}{x} dx = 0$$

per la simmetria. Fatta un’ipotesi, ipotizziamo che  $A \sim 10^5$ , bisogna fissare un valore di  $\varepsilon$ . Se aumento il valore di  $A$  e  $B$ , la precisione dell’integrale crolla, la somma dei due integrali ha raggiunto saturazione entro le cifre che stiamo mantenendo sotto controllo. Quello che succede è che la stima di  $A \sim 10^6$ , con una precisione di  $10^{-5}$ , perciò  $\delta A = 10$ . Analogamente  $B \sim 10^6$  e  $\delta B = 10$ . Il risultato vorretto dovrebbe essere  $-A + B \sim 0.7$ , ma siccome l’errore in valore assoluto è di 10, per avere un errore in valore assoluto di  $\delta = 0.01$ , qui dobbiamo andare ad avere sull’integrale una precisione di 0.01 o meglio. Più piccolo è  $\varepsilon$ , più grandi sono gli integrali e a parità di precisione relativa più grandi sono gli errori assoluti, perciò sono più grandi gli errori assoluti sul valore principale. Bisogna inventarsi qualcos’altro.

# Chapter 8

## Lezione 8

### 8.1 Integrali PV

Nel file **integration\_PV.f** è implementata la struttura solita dell'algoritmo di integrazione dove si ragiona in base alla precisione e adesso serve conoscere dove si trova il punto di singolarità. Se io prendo algoritmi codificati in linguaggi simbolici che lavorano numericamente, necessitano tutti dell'informazione sul punto di singolarità, altrimenti gli algoritmi non sanno come comportarsi. Il modo più semplice che abbiamo visto è quello di fare l'integrale a destra e sinistra. Stiamo parlando di integrali *1D*, per definizione è

$$\lim_{\delta \rightarrow 0} \int_a^{c-\delta} \frac{f(x)}{x-c} dx + \int_{c+\delta}^b \frac{f(x)}{x-c} dx$$

Un modo di fare questa integrazione è di computare i due integrali separatamente. C'è una variabile **ubound** che è l'*upper bound* **ubound = cpv - δ**. Poi c'è il *lower bound*, **lbound = cpv + δ**. La routine si ferma quando trova un errore relativo  $< \varepsilon$ . Si stampa sia la somma sia degli integrali che dei loro valori assoluti. A questo punto, si mette uno **stop** e si vede il funzionamento di questa routine. Integriamo la funzione  $\frac{\exp(x)}{x-c}$ , cioè il punto di singolarità è conosciuto dall'algoritmo. Si promuove la variabile **cpv**, che è un tipico **real\* 8**. L'ho promossa a variabile globale, inserendola nel blocco **singularity**. Dichiaro la variabile e riporto l'istruzione **common/singularity/cpv** in tutti e due i files, chiaramente non devono essere diversi i tipi. Il rischio di utilizzare questa variabile globale è che da fuori ho in mente di utilizzare un valore di input e che se cambio il valore di **cpv**, questo si riflette su tutte le parti del programma.

#### 8.1.1 Prima run

Quali estremi impostiamo?  $a = 0$ ,  $b = 42$ . Impostiamo il polo per  $c = 21$ . Precisione relativa  $\varepsilon = 1\%$ . Numero massimo di chiamate  $10^6$ . Valore dello spessore  $\delta = 1$ .

```
result from qsymp
n. of points=      129      integral=   -289351307.74560708
n. of points=      33       integral=   87268140254768720.

PV int =   87268139965417408.           872681405441200.38
```

Ha un valore spropositato. Questo numero potrebbe essere indice di problema? Vediamo, perché  $e^{42}$  è grande come numero. Con 33 chiamate ha già raggiunto la precisione relativa. Aumentiamo un po' la precisione tenendo sempre questi valori, la settiamo a  $10^{-3}$ .

```
result from qsymp
n. of points=      257      integral=   -289328567.91817123
n. of points=      129      integral=   87209809336652864.

PV int =   87209809047324304.           8720980962598.1426
```

Vediamo che il secondo numero è cambiato, mentre il primo era con una precisione dell 1%, cioè un 2 sulla terza cifra. Con 129 punti sembrano stabili. Abbiamo toccato il problema che per fare la sottrazione di questi due numeri, libero da numeri, se chiediamo l'1% sui due integrali non abbiamo attendibilità, perché un integrale è

dell'ordine dell'errore dell'altro. È meglio restringere gli intervalli, altrimenti dovremmo avere una precisione di  $\varepsilon = 10^{-10}$ . Inseriamo l'accuracy di  $1.d - 11$ , con  $10^7$  chiamate e gli stessi intervalli.

```
result from qsymp
n. of points=      16385    integral=   -289326984.52819985
n. of points=      8193     integral=   87209571343420288.

PV int =  87209571054093296.          872095.71632747271
```

L'integrale viene eseguito e con  $10^{-11}$  siamo ad un errore assoluto di  $10^7$ . L'integrale più piccolo è di ordine  $\sim 10^8$ , perciò almeno una cifra di esso sopravvive e il risultato finale è affidabile almeno per cifre dalla 17 alla 6. Questa integrazione è accettabile, bisogna però vedere come varia quando si restringe il  $\delta$ .

### 8.1.2 Si riduce $\delta$

Si mantiene la precisione, il numero di chiamate e gli estremi, ma si varia  $\delta = 0.01$ .

```
result from qsymp
too many steps in qsymp
n. of points=      1     integral=      NaN
n. of points=    16385    integral=  87209579141141968.

PV int =           NaN           NaN
Note: The following floating-point exceptions are signalling: IEEE_INVALID_FLAG_I
IEEE_DIVIDE_BY_ZERO
```

Abbiamo un **not a number** con un punto, perché? Perché ha arrotondato a 0 il denominatore.

#### Il debugger

Chiamiamo il debugger **gdb integration\_PV.exe**. Mettiamo un break point della funzione `funct`, cioè mettiamo un'etichetta e quando sta girando si ferma lì e possiamo vedere le istruzioni che esegue e i valori delle variabili. Dobbiamo modificare la funzione per poter stampare il valore della funzione perché stampare `func` vuol dire stamparne la definizione. Assegnamo a `check` il valore di `func`, e poiché non è input-output possiamo stamparla. Facciamo un'altra run con i valori di  $\delta = 10^{-4}$  e di precisione  $\varepsilon = 10^{-11}$  variati. Scrivendo  $n$  faccio `next`, cioè l'istruzione successiva. Il valore della funzione integranda con  $x = 0$  è pari a  $-\frac{1}{21} \sim -0.047619$ . Per  $x = 21 - 10^{-4}$  ottengo  $f \sim -10^{14}$ . Per  $x = 10.49995$  ottengo invece  $f \sim -3458$ . Non vedo dei not a number, perciò decido di rirunnarlo. Però se lo faccio vengono dei **NaN**. A cosa possono essere dovuti? Forse non al denominatore approssimato a nullo. È l'interplay tra valore di  $\delta$  che impostiamo e la precisione richiesta.

## 8.2 Estremi di integrazione coerenti

Ritento l'integrazione con estremi più piccoli, cioè  $a = 0$ ,  $b = 5$  e divergenza in  $c = 3$ , con precisione di  $\varepsilon = 10^{-4}$  e  $\delta = 1\%$ .

```
result from qsymp
n. of points=      2049    integral=   -80.842189867816487
n. of points=      1025    integral=   180.21160820466795

PV int =  99.369418336851467          2.6105379807248443E-002
```

Ottengo numeri più ragionevoli. La precisione relativa specificata indica che  $\delta I_1 \sim 10^{-3}$ , il secondo indica un errore di  $\delta I_2 \sim 10^{-2}$ . La somma dei due è  $I_1 + I_2 \sim 99.36941$ . Vediamo se riducendo il  $\delta$  il risultato è stabile. Esistono pacchetti a multiple precision che permettono di variare la precisione delle variabili a posteriori.

#### Riduciamo $\delta$

Aggiustiamo il  $\delta = 10^{-4}$ .

```
result from qsymp
n. of points=      262145    integral=   -173.14098065493764
n. of points=      131073    integral=   272.90773808588614

PV int =  99.766757430948502          4.4604871874082380E-002
```

Vediamo che con questo  $\delta$  gli integrali hanno impiegato più chiamate, ma ero già a conoscenza dei rischi. I valori, rispetto a prima, sono aumentati, circa di un fattore 2 e abbiamo degli errori assoluti del tipo  $\delta I_1 \sim 2 \cdot 10^{-2}$ , il secondo indica un errore di  $\delta I_2 \sim 3 \cdot 10^{-2}$ . Il numero che abbiamo ottenuto ora è  $I^{-4} \sim 99.76676 \pm 4.4605 \cdot 10^{-2}$ , mentre prima avevo un risultato del tipo  $I^{-2} \sim 99.36941 \pm 2.6105 \cdot 10^{-2}$ , quindi i due numeri sono incompatibili. Non è un problema numerico, semplicemente significa che non si è raggiunta la stabilità al variare di  $\delta \rightarrow 0$ . Noi dovremmo implementare numericamente il limite  $\lim_{\delta \rightarrow 0}$ . Temporaneamente, riduciamo ancora il valore di  $\delta = 10^{-6}$ .

```
result from qsymp
n. of points= 16777217      integral= -265.64228194434151
n. of points= 8388609       integral= 365.40862206999213

PV int = 99.766340125650629      6.3105090401433361E-002
```

Abbiamo molte più chiamate, però il numero comincia a stabilizzarsi,  $I^{-6} \sim 99.76634 \pm 6.3105 \cdot 10^{-2}$ . L'unica cosa è che cominciamo a vedere che man mano che riduciamo il  $\delta$ , i singoli valori degli integrali aumentano in valore assoluto ed aumentano gli errori. Possiamo diminuire il  $\delta = 10^{-8}$  e riscontriamo un tempo macchina maggiore, ed è possibile che raggiungiamo il fondo scala delle chiamate

```
result from qsymp
too many steps in qsymp
n. of points= 1      integral= NaN
n. of points= 1073741825      integral= 457.90477980351784

PV int = NaN          NaN          NaN
Note: The following floating-point exceptions are signalling: IEEE_INVALID_FLAG_I
IEEE_DIVIDE_BY_ZERO
```

Vediamo che l'implementazione numerica del limite  $\lim_{\delta \rightarrow 0}$  è poco pratica, dunque necessitiamo di un'altra idea che sia funzionale ai nostri scopi.

### Rilassiamo la richiesta di precisione

Per ora possiamo vedere come variano i valori se decidiamo di rilassare la precisione relativa, dunque  $\varepsilon = 10^{-4} \rightarrow 10^{-2}$ , mantenendo il  $\delta$ .

```
result from qsymp
too many steps in qsymp
n. of points= 1      integral= NaN
n. of points= 134217729      integral= 458.67835480529675

PV int = NaN          NaN          NaN
Note: The following floating-point exceptions are signalling: IEEE_INVALID_FLAG_I
IEEE_DIVIDE_BY_ZERO
```

Vediamo che le reticolazioni non riescono a convergere in un valore diverso da **NaN** e quindi rimane l'idea di dover trovare un metodo più efficace.

## 8.3 Routine dedicata al PV

Potremmo utilizzare delle routine che implementano l'integrale al valore principale, scoprendo che utilizzano dei metodi più grezzi. Negli algoritmi che abbiamo considerato bisogna trovare lo sweet spot tra precisione numerica di integrazione e il valore di  $\delta$  che sto utilizzando per implementare il limite, è un equilibrio molto instabile. La soluzione è molto semplice, dobbiamo fare un integrale ed abbiamo detto che tra i due integrali di

$$\lim_{\delta \rightarrow 0} \int_a^{c-\delta} \frac{f(x)}{x-c} dx + \int_{c+\delta}^b \frac{f(x)}{x-c} dx$$

stiamo facendo una cancellazione numerica di quelli che sono  $\pm\infty$ , così che il risultato sia finito. Il punto cruciale per cui il risultato è finito è che il limite sia definito in modo simmetrico rispetto all'asintoto, altrimenti l'integrale non esiste. Ciascuno dei due integrali, separatamente, per  $\delta = 0$ , non esiste. Vogliamo implementare nell'algoritmo numerico questo aspetto, cioè, invece di fare un integrale nell'intervallo  $[a, c)$  e nell'intervallo  $(c, b]$ , facciamo un unico integrale dove si prendono punti a destra e sinistra, in modo simmetrico, perché se prendo punti simmetrici a  $c$ , man mano che mi avvicino a  $c$ , la cancellazione dei contributi positivi e negativi la posso fare a livello di funzione integranda e non di integrale, nel suo complesso. Questo stratagemma permette di calcolare questi integrali. Stiamo sfruttando la correlazione tra i punti dei nostri intervalli, si chiama per questo **metodo delle variabili antitetiche**.

# Chapter 9

## Lezione 9

### 9.1 Metodo delle variabili antitetiche

$$\int_a^b \frac{f(x)}{x-c} dx = \lim_{\varepsilon \rightarrow 0} \int_a^{c-\varepsilon} \frac{f(x)}{x-c} dx + \int_{c+\varepsilon}^b \frac{f(x)}{x-c} dx$$

Studiando separatamente i due integrali incorriamo in problemi di stabilità numerica e non si riesce ad andare oltre una bassa precisione. Noi utilizziamo le routine di integrazione che abbiamo visto fin'ora per mettere una griglia negli intervalli  $[a, c - \varepsilon]$  e  $[c + \varepsilon, b]$ . L'operazione è ben definita analiticamente  $\forall \varepsilon$ , tuttavia ci sono problemi di natura operativa. Un altro modo di fare l'integrazione è di restringere l'operazione ad un intervallo, ma quando facciamo la media di  $\frac{f(x_i)}{x_i - c}$ , noi stiamo facendo

$$\begin{aligned} & \frac{1}{N} \sum_{i=1}^N \frac{f(x_i)}{x_i - c} + \frac{1}{M} \sum_{j=1}^M \frac{f(x_j)}{x_j - c} \\ &= \frac{1}{N} \sum_{i=1}^N \left( \frac{f(x_i)}{x_i - c} + \frac{f(y_i)}{y_i - c} \right) \end{aligned}$$

Possiamo aggiungere a questo un termine dell'altro intervallo. Nessuno ci dice che i punti  $x_i, x_j$  debbano essere scorrinati, quindi potremmo introdurre una correlazione tra i punti utilizzati nei due intervalli e se i punti sono scorrinati possiamo utilizzare un unico indice  $i$  per labellare i punti che entrano nel primo intervallo e con una mappa opportuna creiamo il reticolo anche sull'altro intervallo, sempre con l'indice  $i$ , con la variabile  $y$ . In questo modo l'integrale cambia volto. Quando prendiamo  $x_i \approx c$ , abbiamo dei grandi contributi positivi o negativi, a seconda del segno della  $f$  nel punto considerato che danno un grosso contributo all'integrale e poi abbiamo delle  $y_i$  che possono andare vicino o lontano al punto di singolarità e che possono dare una grande oscillazione nell'integrando, a seconda di quanto vicino sia a  $c$ . Si può implementare avendo presente la mappatura nei due intervalli  $[a, c - \varepsilon]$  e  $[c + \varepsilon, b]$ . Sulla carta funzionerebbe ma vedremo sempre dei problemi operativi. Possiamo sfruttare la correlazione tra i punti e introduco un mapping che mi correla  $x_i$  e  $y_i$ , in modo tale che quando ci avviciniamo al punto di singolarità noi andiamo in modo random ma in modo simmetrico.

$$\begin{aligned} f(x - \varepsilon)|_{x=c} &= f(c) - \varepsilon f'(c) + O(\varepsilon^2) \\ f(x + \varepsilon)|_{x=c} &= f(c) + \varepsilon f'(c) + O(\varepsilon^2) \\ f(c + \varepsilon) - f(c - \varepsilon) &= 2\varepsilon f'(c) + O(\varepsilon^3) \end{aligned}$$

Abbiamo la  $f(c \pm \varepsilon)$  che differisce per un fattore infinitesimo, ma abbiamo un contributo positivo e negativo, quindi la cancellazione propria della definizione di PV dell'integrale avviene nell'integrando. Questo vuol dire che nel fare questa somma avremo il contributo tipico dell'integrale al PV da cui estraiamo il contributo finito e non vediamo nessuna singolarità. È proprio quello che vuole l'integrale al PV che vuole gli estremi sinistro e destro simmetrici rispetto al valore di singolarità. Fa parte dei metodi generali di riduzione della varianza e si cerca di introdurre delle tecniche per ridurre l'oscillazione della funzione mentre sto cumulando i

valori. La varianza ha senso statistico quando effettuo il Monte Carlo, però ha senso anche in questo algoritmo deterministico nel senso di ridurre le oscillazioni. Se faccio la stima degli errori, ne ho sempre un beneficio.

### 9.1.1 Mapping lineare pseudo simmetrico

Per migliorare in modo significativo il funzionamento dell'algoritmo doppiamo prendere questo mapping. Dobbiamo prendere una mappa che fa sì che quando  $x_i \in [a, c - \varepsilon]$  e  $y_i \in [c + \varepsilon, b]$  vogliamo che la simmetria intorno a  $c$  ci sia, in particolare quando  $x_i \rightarrow c \Rightarrow y_i \rightarrow c$ . Possiamo prendere una mappatura lineare  $y = dx + e$  e le condizioni sono che quando

$$\begin{aligned} x = c - \delta &\Leftrightarrow y = c + \delta \\ c + \delta &= (c - \delta)d + e \\ d &= \frac{c + \delta}{c - \delta} - \frac{e}{c - \delta} \\ b = da + e &\Rightarrow d = \frac{b}{a} - \frac{e}{a} \end{aligned}$$

Questo ci permette di avere la simmetria intorno al punto  $c$ . Questo ci comporta che  $e$ , quindi ne deduciamo che  $d$ . Vogliamo inoltre che i punti siano agli estremi contemporaneamente per simmetria. Mettiamo a sistema le espressioni di  $d$  e troviamo

$$\begin{aligned} \frac{c + \delta}{c - \delta} - \frac{e}{c - \delta} &= \frac{b}{a} - \frac{e}{a} \\ e\left(\frac{1}{c - \delta} - \frac{1}{a}\right) &= \frac{c + \delta}{c - \delta} - \frac{b}{a} \\ e(a - (c - \delta)) &= (c + \delta)a + (c - \delta)b \\ e &= \frac{(c + \delta)a + (c - \delta)b}{a - (c - \delta)} \end{aligned}$$

Con il valore di  $e$  individuato possiamo dedurre il valore di  $d$

$$\begin{aligned} d &= \frac{b}{a} - \frac{1}{a} \frac{(c + \delta)a + (c - \delta)b}{a - (c - \delta)} \\ &= \frac{1}{a} \frac{b(a - (c - \delta)) - (c + \delta)a + (c - \delta)b}{a - (c - \delta)} \\ &= \frac{1}{a} \frac{ba - b(c - \delta) - (c + \delta)a + (c - \delta)b}{a - (c - \delta)} \\ &= \frac{b - (c + \delta)}{a - (c - \delta)} \end{aligned}$$

Con questa mappatura facciamo che  $\forall x \in [a, c - \delta] \exists y \in [c + \delta, b]$ . Qualche possibile problema è che i due intervalli non abbiano la stessa misura. Quando abbiamo l'intervallo simmetrico attorno a  $c$ , l'algoritmo funziona perfettamente. Se l'ampiezza  $b - c \neq c - a$  l'algoritmo non può funzionare in modo ottimale perché un punto preso con distanza  $\xi$  da  $c$ , non ha un corrispondente che dista  $\xi$  da  $c$  in verso positivo. Il punto più vicino lo prendiamo in modo simmetrico ma non tutti i punti simmetrici, eppure a noi interessa che il sampling dei punti simmetrici nell'avvicinarsi alla singolarità. Posso anche prendere punti simmetrici esattamente per far funzionare in modo ottimale l'algoritmo. Quando mi avvicino a  $c$  ho altri punti vicini nella griglia che mi possono dare cancellazioni. Si può cercare anche di estrarre il  $h = \min(b - c, c - a)$  e simmetrizzare, così, partendo da un dominio esteso  $h + g$ , dove  $g = \max(b - c, c - a)$  a un dominio esteso come  $2h$ . Al quale si aggiunge poi un integrale ordinario su  $g - h$ .

### 9.1.2 Implementazione su intervallo simmetrico

Facciamo le prime run con il software nel quale abbiamo incluso la correlazione tra i punti campionati nei due intervalli. Vediamo com'è fatta la funzione. In precedenza abbiamo utilizzato la funzione **func**, invece

utilizziamo **f1unc**.

$$f1unc = f1unc + \frac{1.d0 \cdot e^y}{y - cpv}$$

Dove  $y$  è la variabile che corre nell'altro intervallo. L'integrandi è la somma di due pezzi, dove si vede la cancellazione della singolarità a livello di integrando.

### Run con precisione e finezza progressivamente maggiori

Proviamo a integrare e a vedere come funziona. Facciamo una run con parametri  $\varepsilon = 1.d - 3$ , numero di chiamate massimo  $10^6$ , estremi di integrazione 0, 5 con  $c = 2.5$  e finezza  $\delta = 10^{-4}$ . La precisione relativa che chiediamo sull'integrale è la medesima che chiedevamo nel vecchio procedimento.

```
result from qsymp
n. of points=      65537      integral=   -104.87732644606432
n. of points=      65537      integral=    191.35452607895965

PV int =     86.477199632895335      0.29623185252502399
n. of points=      33      integral=    86.477204537289595

PV int =     86.477204537289595      8.6477204537289590E-002
```

Gli integrali nel vecchio caso vengono svolti con 65537 punti. Il metodo delle variabili antitetiche raggiunge la precisione richiesta in 33 punti. Cerchiamo di avvicinarci alla singolarità e di aumentare la precisione.  $\varepsilon = 10^{-4}, \delta = 10^{-5}$ .

```
result from qsymp
n. of points=      2097153      integral=   -132.92086232595642
n. of points=      1048577      integral=    219.40137411720636

PV int =     86.480511791249938      3.5232223644316274E-002
n. of points=      33      integral=    86.479397386997519

PV int =     86.479397386997519      8.6479397386997527E-003
```

I singoli integrali arrivano a  $2 \cdot 10^6$  e  $10^6$  di chiamate, mentre nel secondo caso il numero di chiamate è sempre 33. La cancellazione tra le divergenze positive e negative porta ad un risultato finito ed altamente preciso, senza alcun problema di aumentare il numero di chiamate. Spingo ancora di più precisione e finezza verso i valori di  $\varepsilon = 10^{-6}, \delta = 10^{-5}$ . Vediamo che con questi valori il metodo degli integrali destro e sinistro non riesce ad ottenere un valore in un tempo accettabile, di fatti il numero di chiamate supera i  $4 \cdot 10^6$ .

```
result from qsymp
n. of points=      4194305      integral=   -132.92084055725692
n. of points=      4194305      integral=    219.40021213320315

PV int =     86.479371575946232      3.5232105269046007E-004
n. of points=      33      integral=    86.479397386997519

PV int =     86.479397386997519      8.6479397386997518E-005
```

Noi stiamo facendo ciascun integrale con precisione  $10^{-6}$ , quindi con  $4 \cdot 10^6$  chiamate,  $10^{-6}$  vuol dire che abbiamo un errore assoluto di  $10^{-3}$  sul primo integrale e di  $2 \cdot 10^{-3}$  sul secondo. Faccio la somma dei due ed ottengo una cancellazione parziale, però ho guadagnato un ordine di grandezza di precisione rispetto a prima con l'integrale delle variabili antitetiche. Se aumento la precisione ho problemi con il vecchio metodo, mentre con il nuovo ho un risultato stabile.

### 9.1.3 Implementazione su intervallo asimmetrico

È sufficiente fare la run con estremi di integrazione asimmetrici rispetto al valore di divergenza.

#### Run

Proviamo una run con valori  $\varepsilon = 10^{-3}$ ,  $a = 0$ ,  $b = 5$ ,  $c = 3$ ,  $\delta = 10^{-4}$ .

```

result from qsympy
n. of points=      131073    integral=   -173.14251692598191
n. of points=      65537     integral=    272.91224728349431

PV int =   99.769730357512401      0.44605476420947621
n. of points=      65537     integral=   236.21343083575061

PV int =   236.21343083575061      0.23621343083575061

```

Il primo integrale effettua 131073 chiamate, il secondo 65537. L'integrale con il nuovo metodo ne effettua 65537, quindi un risultato molto diverso rispetto a prima. I risultati che ottengono sono nettamente differenti. Questo perché il mapping era pensato per essere applicato su di un intervallo simmetrico, dobbiamo perciò produrre una generalizzazione del metodo agli intervalli asimmetrici.

## 9.2 Generalizzazione del metodo agli intervalli asimmetrici

Si può cercare anche di estrarre il sottointervallo più piccolo  $h = \min(b - c, c - a)$  e simmetrizzare. Così, partendo da un dominio esteso  $h + g$ , dove  $g = \max(b - c, c - a)$  integro su un dominio esteso come  $2h$ . A questo si aggiunge poi un integrale ordinario su  $g - h$ . Per esempio, con  $b - c > c - a$  abbiamo che i valori  $h = c - a$  e  $g = b - c$ . L'intervallo simmetrico consiste in  $[a, 2c - a]$ . Il resto dell'intervallo asimmetrico consiste in  $[2c - a, b]$ .

$$f = f(x \in [a, 2c - a]) + f(y \in [2c - a, b])$$

Quando  $x$  è sul supporto di  $f$ , la  $f$  assume valori non nulli, mentre al di fuori di esso impongo che  $f$  sia 0. In questo modo, con una sola integrazione, semplicemente vado a definire la  $f$  come somma di questi contributi, uno quando sono nell'intervallo simmetrico e uno quando sono nell'intervallo asimmetrico. Si lascia come esercizio per la didattica integrativa asincrona.

## 9.3 Algoritmi di Gauss

È una classe di algoritmi introdotti da Gauss. Facciamo mente locale su quanto fatto fin ora.

$$I = \int_{-1}^{+1} f(x) dx = \sum_{i=1}^N w_i f(x_i)$$

Stiamo facendo una combinazione lineare dei valori della funzione. Abbiamo visto con questi algoritmi uno sviluppo di Taylor della funzione per stimarne la precisione. Possiamo affrontare il problema da un punto di vista diverso. Questi algoritmi sono esatti con un numero finito di punti se la  $f$  è un polinomio di un certo ordine. Se prendiamo Sympson e un polinomio, con  $N = 3$ , otteniamo una soluzione esatta con polinomi fino al grado 3. Dati  $N$  punti, questi algoritmi dove si fa una reticolazione equispaziata dei punti sull'ascissa, se sono nell'intervallo  $[-1, 1]$ ,  $x_n = -1 + 2 \frac{n-1}{N-1}$ . Mi sto muovendo di passi equispaziati all'interno dell'intervallo. In generale abbiamo polinomi con  $N$  punti che possono integrare funzioni polinomiali in modo esatto fino al grado  $N - 1$ . L'idea di Gauss è di estendere la classe di funzioni che sto integrando esattamente a dei polinomi di grado maggiore, anziché  $N - 1 \rightarrow 2N - 1$  con  $N$  punti. Questo come lo posso fare? Negli altri algoritmi avevo dato per scontato che la griglia dei punti sulle ascisse fosse fissata rigidamente in modo equispaziato, però, se noi rinunciamo a questa richiesta, vediamo cosa succede. Il nostro integrale è del tipo

$$\int_a^b f(x) dx = \sum_{n=1}^N w_n f(x_n)$$

Adesso vogliamo che questa espressione sia esatta, cioè calcolando la funzione in un numero finito di punti e con pesi  $w_n$  opportuni che io riesca ad integrare esattamente tutti i polinomi fino al grado  $2N - 1$ , questo con le regole di quadratura. Facciamo un esempio con  $N = 2$  per le funzioni

$$f(x) = 1, x, x^2, x^3$$

Il nostro programma di integrare queste funzioni con un numero finito di punti, a sinistra ci metto la funzione esatta e a destra ci metto l'approssimazione computazionale.

$$\begin{cases} f(x) = 1 \rightarrow (b-a) = w_1 + w_2 \\ f(x) = x \rightarrow \frac{1}{2}(b^2 - a^2) = w_1 x_1 + w_2 x_2 \\ f(x) = x^2 \rightarrow \frac{1}{3}(b^3 - a^3) = w_1 x_1^2 + w_2 x_2^2 \\ f(x) = x^3 \rightarrow \frac{1}{4}(b^4 - a^4) = w_1 x_1^3 + w_2 x_2^3 \end{cases}$$

Sia i punti che i pesi sono incognite di questo sistema. Ho 4 equazioni in 4 incognite. È un sistema lineare che posso in generale risolvere e trovo sia i punti  $x_{1,2}$  che  $w_{1,2}$ .

$$\int_a^b f(x) dx = \sum_{n=1}^N w_n f(x_n)$$

La mia formula di quadratura deve restituirmi esattamente il sistema. In generale, fissato  $N$ , posso ricondurre il mio problema integrale alla soluzione di un sistema lineare. Il problema è che la cosa diventa altamente non lineare appena aumento le potenze, posso avere termini fino al grado  $N+1$ . In generale non è facile. L'idea di Gauss è di sfruttare i polinomi ortogonali rispetto a una funzione peso, che chiamiamo  $w(x)$

$$\int_a^b w(x) \varphi_m(x) \varphi_n(x) = \delta_{nm} c_n$$

- Con  $w(x) = 1$  ottengo i polinomi di Legendre.
- Con  $w(x) = e^{-x}$  ottengo i polinomi di Laguerre.
- Con  $w(x) = e^{-x^2}$  ottengo i polinomi di Hermite, protagonisti dell'oscillatore armonico.
- Con  $w(x) = \frac{1}{\sqrt{1-x^2}}$  ottengo i polinomi di Chebyshev di prima specie.
- Con  $w(x) = \sqrt{1-x^2}$  ottengo i polinomi di Chebyshev di seconda specie.
- Con  $w(x) = (1-x^2)^{\alpha-\frac{1}{2}}$  ottengo i polinomi di Gegenbauer.
- Con  $w(x) = (1-x)^\alpha (1+x)^\beta$  ottengo i polinomi di Jacobi.
- Con  $w(x) = (1-x^2)^{\frac{\alpha}{2}}$  ottengo i polinomi di ultrasfera.

Rimodellando questi polinomi, il metodo diventa molto potente e vediamo come si procede nella pratica. Estendiamo il nostro problema

$$\int_a^b w(x) f(x) dx = \sum_{m=1}^N w_m f(x_m)$$

Questo è l'algoritmo di quadratura e chiediamo che sia esatto quando la  $f(x)$  è un polinomio di grado  $2N-1$  e adesso introduciamo il nostro polinomio  $\varphi_m(x)$ . Questi polinomi che abbiamo studiato, sappiamo che sono un sistema ortonormale e completo.

### 9.3.1 Zeri dei polinomi

Ogni polinomio può essere sviluppato su questi polinomi.

$$f(x) = q_{N-1}(x) \varphi_N(x) + r_N(x)$$

Dove  $q, r$  sono polinomi in  $x$ .

$$\int_a^b f(x)w(x)dx = \int_a^b w(x)q_{N-1}(x)\varphi_N(x)dx + \int_a^b w(x)r_{N-1}(x)dx$$

$$q_{N-1}(x) = \sum_{i=0}^{N-1} q_i \varphi_i(x)$$

$$\int_a^b w(x)q_{N-1}(x)\varphi_N(x)dx = \sum_{i=0}^{N-1} \int_a^b w(x)q_i \varphi_i(x)\varphi_N(x)dx = 0$$

Ci ricordiamo che i polinomi sono un SOC, quindi possiamo chiedere che  $q_{N-1}$  sia sviluppabile sulla base dei polinomi. Questo lo facciamo perché osserviamo che il primo integrale può essere scomposto sulla base e per la relazione di ortogonalità  $i \neq N$  fa 0. Questo è importante perché il nostro integrale iniziale lo abbiamo ricondotto all'integrale di un polinomio resto di grado  $N - 1$ . Osserviamo anche un altro fatto, che il nostro  $q_{N-1}\varphi_N$  è un polinomio di ordine  $2N - 1$ , quindi l'integrale deve avere una rappresentazione del tipo

$$\sum_{m=1}^N w_m q_{N-1}(x_m) \varphi_N(x_m)$$

Dove abbiamo già assunto che per i polinomi vale la formula di quadratura, con i pesi ed i punti da determinare. Il fatto che il polinomio iniziale era totalmente arbitrario e che la quadratura dell'integrale di  $q_{N-1}\varphi_N$  dev'essere uguale al suo sviluppo sulla base polinomiale, cioè 0. L'unica possibilità è che o  $q_{N-1} = 0$ , ma sappiamo che deriva dalla decomposizione del nostro polinomio arbitrario  $f$ , e quindi non è 0. L'unica possibilità che la somma sia 0 sta negli zeri del polinomio  $\varphi_N(x_m) = 0$ . Questo ci fissa l'algoritmo con cui determinare la reticolazione. Non sono più equispaziati ma sono gli zeri del polinomio di grado  $N$ . Per il teorema fondamentale dell'algebra, il polinomio ha al più  $N$  zeri distinti. I primi 20 zeri dei polinomi di Hermite, per esempio, sono disponibili al [link](#). Ho  $2N$  variabili, gli  $x_n$  sono fissati, perché gli unici  $x$  che risolvono l'integrale.

### 9.3.2 Polinomi di interpolazione di Lagrange

Dobbiamo fare un altro ragionamento sfruttando il fatto che deve esserci la validità per tutti i polinomi fino all'ordine  $2N - 1$ . Prendiamo una classe particolare di polinomi, i polinomi di interpolazione di Lagrange.

$$l_{i,N}(x) = \frac{(x - x_1) \dots (x - x_{i-1})(x - x_{i+1}) \dots (x - x_m)}{(x_i - x_1) \dots (x_i - x_{i-1})(x_i - x_{i+1}) \dots (x_i - x_m)}$$

$$l_{i,N}(x_j) = \begin{cases} 0 & \wedge j \neq i \\ 1 & \wedge j = i \end{cases} = \delta_{ij}$$

Vale la proprietà  $l_{i,N}(x_j) = 0 \forall j \in [1, N] \setminus \{i\}$ . al numeratore o denominatore non compare  $x - x_i$ . È un polinomio con  $N - 1$  zeri nei punti  $x_m$  tranne che in  $x_i$ . Il polinomio è di grado  $N - 1$  e la nostra formula di quadratura deve valere anche per i polinomi di interpolazione.

#### Pesi della quadratura

$$\int_a^b l_{i,N}(x)w(x)dx = \sum_{m=1}^N w_m l_{i,N}(x_m) = w_i$$

L'integrale deve essere scrivibile come somma di questi pesi per la funzione calcolata sugli zeri. Poiché vale la proprietà  $l_{i,N}(x_j) = \delta_{ij}$ , la sommatoria si riduce ad un termine. Se noi facciamo l'integrazione di questi polinomi particolari, troviamo i pesi  $w_i$ . Richiede un po' di lavoro preventivo, ma una volta fatto questo lavoro una volta sola, abbiamo l'algoritmo che funziona per un polinomio qualunque con la  $w(x)$  assegnata. Diventa un metodo molto potente. Come nel teorema del valor medio della funzione continua in cui l'integrale è uguale alla funzione nel punto medio per l'ampiezza dell'intervallo. In questo caso lo sto facendo per una classe di funzioni molto grande. Quando era in vita Gauss non c'era il calcolatore, quindi lo aveva introdotto per calcolare in modo approssimato degli integrali valutando le funzioni solo in alcuni punti con un ottimo grado di approssimazione.

# Chapter 10

## Lezione 10

### 10.1 Algoritmi di Quadratura di Gauss

#### 10.1.1 Quadratura di Gauss-Laguerre

Un integrale generico tra  $a, b$  di una funzione peso positiva per una  $f(x)$  lo possiamo quadrare in una sommatoria.

$$\int_a^b w(x)f(x)dx = \sum_{i=1}^N w_i f(x_i)$$

I punti  $x_i$  li abbiamo individuati essere gli zeri associati al polinomio di grado  $N$  e i  $w_i$  si ottengono i polinomi di interpolazione di Lagrange. La formula è costruita per essere esatta per polinomi di grado  $2N - 1$  per integrazione di polinomi. La funzione peso  $w$  può avere diverse forme, quelle tipiche sono

- Per  $w(x) = e^{-x}$  e l'integrale viene fatto tra 0 e  $\infty$ . In questo caso si utilizzano i polinomi di Laguerre, studiati nella soluzione dell'eq. di Schrödinger per l'atomo di idrogeno.  $L_0 = 1, L_1 = x + 1, L_2 = \frac{1}{2}(x^2 - 4x + 2), L_3 = \frac{1}{6}(-x^3 + 3x^2 - 18x + 6)$ . Sono generati dalla formula

$$L_n(x) = \frac{e^x}{n!} \frac{d^n}{dx^n} (e^{-x} x^n)$$
$$(n+1)L_{n+1}(x) = (2n+1-x)L_n(x) - nL_{n-1}(x)$$

Il problema dell'integrazione è stato spostato al problema di trovare le radici di questi polinomi. I pesi, per un numero più piccolo di punti, possiamo utilizzare i risultati individuati in precedenza, per esempio, i polinomi soddisfano la proprietà di ortogonalità.

$$\int_0^\infty e^{-x} L_m(x) L_n(x) dx = \delta_{n,m} C_m, \quad C_m = 1$$

Caso  $N = 2$

Se faccio un algoritmo di quadratura Gauss-Laguerre con  $N = 2$ , devo trovare le radici del polinomio di grado 2, cioè

$$x^2 - 4x + 2 = 0 \rightarrow \begin{cases} x_1 = 2 - \sqrt{2} \\ x_2 = 2 + \sqrt{2} \end{cases}$$

I pesi vanno individuati

$$w_1 = \int_0^\infty e^{-x} l_{1,2} dx, \quad l_{1,2} = \frac{x - x_2}{x_1 - x_2}$$
$$= \frac{1 + \sqrt{2}}{2\sqrt{2}}$$
$$w_2 = \int_0^\infty e^{-x} l_{2,2} dx = \frac{\sqrt{2} - 1}{2\sqrt{2}}$$

Applicando l'algoritmo per 2 punti questo significa che possiamo integrare esattamente una forma con polinomi fino al terzo grado.

$$\int_0^\infty e^{-x}(a + bx + cx^2 + dx^3)dx = w_1 f(x_1) + w_2 f(x_2), \quad f(x) = a + bx + cx^2 + dx^3$$

Poiché sposto il problema dell'integrazione alla ricerca degli zeri, ho dei risultati accurati per un grado contenuto. Se ho dei termini polinomiali per un grado elevato, devo soltanto andare come il grado  $N$  del polinomio. Questo vuol dire che devo trovare gli zeri di un polinomio di grado superiore al secondo. Il problema si sposta negli algoritmi di approssimazione degli zeri nel momento in cui supero il quinto grado. Le formule studiate in generale permettono di impostare la ricerca in modo completamente numerico. Dopodiché, se mi serve un  $N$  grande, è difficile anche integrare i polinomi di Lagrange. Nel caso di Laguerre esiste una formula generale per i pesi

$$w_i = \frac{x_i}{(n+1)^2 [L_{n+1}(x_i)]^2}$$

Si procede con l'illustrazione di altri casi particolari e si chiude l'argomento dell'integrazione. Lo script che stiamo usando è **driver\_integration.f** con **qgausslag.f**. Poiché stiamo facendo un integrale su  $\mathbb{R}^+$ , con gli altri metodi c'è il discorso di fare una mappatura della funzione tra 0 e  $\infty$ , oppure di mettere un cutoff superiore e vedere la dipendenza dall'estremo superiore. Utilizziamo per l'esempio della funzione  $func(x) = e^{-x}(3+x+25x^2+x^3)$ . Questo va in input come integrando negli algoritmi visti in precedenza, cioè non Gauss. Sotto abbiamo la funzione utilizzata per la routine di Gauss  $f(x) = 3+x+25x^2+x^3$ . L'algoritmo prende i pesi e calcola il polinomio in funzione degli zeri dei polinomi di Laguerre e calcola l'integrale. Con  $N = 2$  otteniamo 60 con la quadratura di Gauss, mentre con l'algoritmo di Sympson sono necessari 257 chiamate della funzione.

```
result from qsymp
n. of points=      16385      integral=   -289326984.52819985
n. of points=      8193       integral=   87209571343420288.

PV int =   87209571054093296.           872095.71632747271
```

Questi sono diversi ma in teoria l'integrale dovrebbe essere esatto e non risente dell'estremo superiore, mentre gli altri integrali sì. Se alziamo il cutoff otteniamo

```
result from qsymp
n. of points=      16385      integral=   -289326984.52819985
n. of points=      8193       integral=   87209571343420288.

PV int =   87209571054093296.           872095.71632747271
```

Con Sympson otteniamo il risultato con 513 chiamate, molte più di 2. Uno dei problemi dell'integrazione numerica con dei dati algoritmi esatti è di portare l'ampiezza dell'integrale che dobbiamo calcolare ad essere pari con l'ampiezza su cui è stato dedotto l'algoritmo.

### 10.1.2 Quadratura di Gauss-Legendre

Sfrutto i polinomi di Legendre, caratterizzati da  $w(x) = 1$ , ortogonali nell'intervallo  $[-1, 1]$ . Sono i polinomi  $P_0 = 1, P_1 = x, P_2 = \frac{1}{2}(3x^2 - 1)$ . Tutti i pesi si vede che sono  $w_i = 1$ . Si deduce che

$$w_i = \frac{2}{(1 - x_i)^2 (P'(x_i))}$$

### 10.1.3 Quadratura di Gauss-Hermite

Per integrali del tipo

$$\int_{\mathbb{R}} e^{-x^2} P^{(2n-1)}(x) dx = \sum_{i=1}^n w_i P^{(2n-1)}(x_i)$$

Dove  $x_i$  sono gli zeri del polinomio di Hermite di grado  $n$ .

### 10.1.4 Quadrature di Gauss-Chebyshev

L'integrale è della forma

$$\int_{-1}^1 \frac{P^{(2n-1)}(x)}{\sqrt{1-x^2}} dx, \quad \text{I specie}$$

$$\int_{-1}^1 P^{(2n-1)}(x) \sqrt{1-x^2} dx, \quad \text{II specie}$$

- Per i polinomi di prima specie vale la relazione di ricorsione  $T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$  e alcuni esempi sono  $T_0 = 1, T_1 = x, T_2 = 2x^2 - 1$ . Esistono delle forme chiuse per i zeri e i pesi

$$x_i = \cos\left(\frac{2i-1}{2n}\pi\right)$$

$$w_i = \frac{\pi}{n}$$

Si possono dedurre applicando iterativamente la regola della somma di angoli partendo dall'identità  $T_n(\cos \theta) = \cos(n\theta)$ .

- Per i polinomi di seconda specie vale una forma leggermente diversa.  $U_0 = 1, U_1 = 2x, U_2 = 4x^2 - 1$ , la relazione di ricorsione è  $U_{n+1}(x) = 2xU_n(x) - U_{n-1}(x)$ . I zeri e i pesi sono

$$x_i = \cos\left(\frac{i}{2n}\pi\right)$$

$$w_i = \frac{\pi}{n+1} \sin^2\left(\frac{i}{n+1}\pi\right)$$

Si possono scrivere come

$$U_n(\sin \theta) = \frac{\cos(n+1)\theta}{\sin \theta}$$

Queste sono classi generali di funzioni particolari. Sono formule esatte quando abbiamo esattamente un polinomio che moltiplica la funzione peso. Quando la funzione non è polinomio, ma è approssimata da esso, allora funzionano con grande accuratezza. La stima dell'errore di integrazione è più difficoltosa, benché precedentemente fosse una semplice espansione di Taylor.

#### Spettro di corpo nero

Se volessi approssimare l'integrale della funzione  $\frac{x^3}{e^x - 1}$ , posso scriverla in modo utile rispetto alla quadratura di Gauss-Laguerre.

$$\int_0^\infty e^{-x} \frac{x^3}{1 - e^{-x}} dx$$

So che il risultato è  $\frac{\pi^4}{15}$  e sfrutto gli zeri del polinomio di Laguerre  $x_i = 2 \pm \sqrt{2}$ . Se ho funzioni con singolarità quale algoritmo posso applicare? Prendiamo per esempio

$$\int_0^\infty \frac{x^3}{(e^x - 1)(x - 4)} dx$$

Come prima cosa devo testare le condizioni necessarie di convergenza. Poi effettuo una partizione del dominio di integrazione per avere degli intervalli simmetrici attorno alle divergenze, sui quali applicare l'algoritmo per il PV degli integrali.

#### Spettro di Corpo Nero e valore di Picco analitico

Definiamo la funzione  $W(xe^x) = x$  come la funzione inversa del prodotto tra esponenziale e  $x$ . La funzione è conosciuta e studiata sotto il nome di funzione W di Lambert. Essa per argomenti reali possiede due rami, uno positivo e uno negativo. Si può facilmente adoperare per ricavare il valore di picco dello spettro di corpo nero. Scriviamo la funzione dello spettro di emissione

$$f(x) = \frac{x^3}{e^x - 1}$$

Si deriva la funzione per ottenere

$$\begin{aligned} f'(x) &= -\frac{e^x x^3}{(e^x - 1)^2} + \frac{3x^2}{e^x - 1} = \frac{x^2}{e^x - 1} \left( -\frac{e^x x}{e^x - 1} + 3 \right) \\ &= \frac{x^2}{(e^x - 1)^2} (-e^x x + 3(e^x - 1)) = \frac{f(x)^2}{x^4} (-e^x(x - 3) - 3) \end{aligned}$$

Per trovare il picco, cioè il massimo, dobbiamo porre la derivata a 0.

$$f'(x) = 0 \Leftrightarrow -e^x(x - 3) - 3 = 0$$

Si può riscrivere l'equazione a destra come

$$-3e^{-3} = e^{x-3}(x - 3)$$

$$W(-3e^{-3}) = W(e^{x-3}(x - 3)) = x - 3$$

$$x = 3 + W((-3)e^{-3})$$

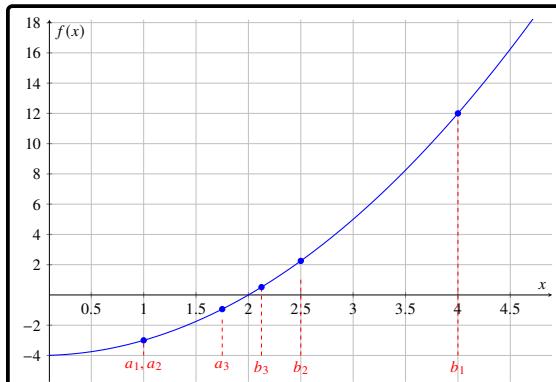
$$\simeq 3 - .178560\ 627877\ 921106\ 596808\ 669705\ 514804\ 654118\ 255926\ 885907\ 720142\ 306$$

$$\simeq 2.821439\ 372122\ 078893\ 403191\ 330294\ 485195\ 345881\ 744073\ 114092\ 279857\ 694$$

## 10.2 Ricerca degli zeri

Abbiamo esaurito la sezione relativa agli algoritmi di integrazione deterministici in una dimensione. Ora il problema è cercare gli zeri di funzioni in una dimensione. Ho un intervallo  $(a, b)$  sul quale la funzione  $f(x)$  è continua. Se  $f(a)f(b) < 0$  allora so per Weierstrass che ha almeno uno zero. Per casi nei quali ho funzioni tangenti all'asse reale, non posso applicare questo tipo di ragionamento. Posso applicare algoritmi come della **bisezione**.

### 10.2.1 Algoritmo di bisezione



So che  $f(a)f(b) < 0$  e che sull'intervallo  $(a, b)$  la  $f$  è continua. Calcolo il valore della funzione nel punto medio  $\frac{1}{2}(a+b) = \bar{x}$ . Poiché  $f(a)f(b) < 0$  identifico due casi

- Il caso in cui  $f(a)f(\bar{x}) > 0$ , nel quale lo zero è nell'intervallo  $(\bar{x}, b)$ .
- Il caso in cui  $f(a)f(\bar{x}) < 0$ , nel quale lo zero è nell'intervallo  $(a, \bar{x})$ .

Ogni volta identifico lo zero in un intervallo di lunghezza dimezzata. Ripeto l'algoritmo fino a che la dimensione dell'intervallo in cui localizzo lo zero non soddisfa i miei parametri di precisione. Se chiamo  $\varepsilon_n$  l'ampiezza dell'intervallo in cui identifico lo zero della funzione è legato al successivo da  $\varepsilon_{n+1} = \frac{\varepsilon_n}{2}$ . Se iterò questo ragionamento dal primo intervallo, che chiamo  $(b-a) = \varepsilon_0$ , ottengo  $\frac{\varepsilon_0}{2^n} = \varepsilon_{n+1}$ . Posso invertire la relazione per identificare  $n = \log_2 \frac{\varepsilon_0}{\varepsilon_{n+1}}$ . Questo mi aiuta a comprendere quando si ferma l'algoritmo. Se sto lavorando con numeri, in doppia precisione il mio zero è  $10^{-16}$ , la mia sensibilità. Questo algoritmo mi assicura **rapidità di convergenza** esponenziale. Per algoritmi generici posso stabilire la rapidità di convergenza tramite la formula

$$\varepsilon_{n+1} = k(\varepsilon_n)^m$$

Per l'algoritmo di bisezione abbiamo i parametri  $k = \frac{1}{2}$ ,  $m = 1$ . Esistono metodi più veloci della bisezione e il più veloce in assoluto è il metodo di Newton-Rapson.

### 10.2.2 Metodo di Newton-Raphson

Questo metodo prevede la conoscenza della derivata della funzione, oltre che del valore della funzione stessa.

$$f(x + \delta) = f(x) + f'(x)\delta + \frac{1}{2}\delta^2 f''(x) + \dots$$

Se  $x$  è la soluzione di tentativo,  $x + \delta$  è la successiva. Facciamo l'ipotesi che sia lo zero esatto. Da questo parte l'algoritmo. Se trascuriamo il termine di ordine  $\delta^2$  e ci fermiamo al primo ordine, otteniamo

$$\delta = -\frac{f(x)}{f'(x)}$$

Parto con un certo  $x_i$  e l'algoritmo mi dice che mi devo spostare in  $x + \delta = x - \frac{f(x)}{f'(x)}$ . Possiamo dunque scrivere che

$$\begin{aligned} x_{i+1} &= x_i - \frac{f(x_i)}{f'(x_i)} \\ \bar{x}_0 + \delta_{i+1} &= \bar{x}_0 + \delta_i - \frac{f(x_i)}{f'(x_i)} \\ \delta_{i+1} &= \delta_i - \frac{f(x_i)}{f'(x_i)} \end{aligned}$$

Abbiamo dedotto la stima della larghezza dell'intervallo in cui è identificata la soluzione successiva a partire dalla larghezza dell'intervallo precedente. Per capire quanto velocemente converge, facciamo due passaggi semplicissimi. Noi abbiamo per  $\bar{x}_0$  zero della funzione,  $x_i = \bar{x}_0 + \delta_i$ , e dallo sviluppo deduciamo che

$$\begin{aligned} f(x_i) &= f(\bar{x}_0) + \delta_i f'(\bar{x}_0) + \frac{1}{2}\delta_i^2 f''(\bar{x}_0) + \dots \\ f'(x_i) &= f'(\bar{x}_0) + \delta_i f''(\bar{x}_0) + \dots \\ \frac{f(x_i)}{f'(x_i)} &= \frac{\delta_i(f'(\bar{x}_0) + \frac{1}{2}\delta_i f''(\bar{x}_0))}{f'(\bar{x}_0) + \delta_i f''(\bar{x}_0)} \approx \frac{\delta_i}{f'(\bar{x}_0)} [f'(\bar{x}_0) + \frac{1}{2}\delta_i f''(\bar{x}_0)] [1 - \delta_i f''(\bar{x}_0)] \\ &= \frac{\delta_i}{f'(\bar{x}_0)} [f'(\bar{x}_0) - \delta_i f''(\bar{x}_0) + \frac{\delta_i}{2} f''(\bar{x}_0)] \\ &= \delta_i - \frac{\delta_i^2}{2} \frac{f''(\bar{x}_0)}{f'(\bar{x}_0)} \end{aligned}$$

Trascuro i termini di ordine superiore al secondo in  $\delta_i$ . Se riprendo la relazione tra intervalli successivi la posso riscrivere come

$$\delta_{i+1} = \delta_i - \frac{\delta_i^2}{2} \frac{f''(\bar{x}_0)}{f'(\bar{x}_0)}$$

In questo modo ho dedotto la legge di convergenza e trovo che la distanza dalla soluzione vera converge quadraticamente, perché dato un  $\delta_i$ , all'iterazione successiva, mi avvicino con distanza  $\delta_{i+1}^2$ . È l'algoritmo più veloce di cui possiamo disporre. Ha qualche rischio, con la bisezione partiamo da un intervallo, la dividiamo in due e ragionando sugli estremi **intrappolo** la soluzione nell'intervallo considerato. Con Newton-Raphson puro, se la funzione ha dei minimi locali, a seconda di come si parte, se valuto in prossimità del minimo la funzione, la derivata diventa quasi orizzontale e  $\delta$  diverge. Tipicamente per la ricerca degli zeri si applica, controllando il risultato con la bisezione. Si controlla di quanto ci si sposta e se lo spostamento è eccessivo, si segue la bisezione. Esistono dei metodi intermedi come il Newton discretizzato, o metodo delle secante.

### 10.2.3 Metodo della secante

Ansiché stimare la derivata per identificare lo zero della funzione, si va a cercare l'intercetta della secante. Parto con la soluzione di tentativo  $x_1$  e fisso un altro estremo di segno opposto nell'intervallo  $x_0$ . Traccio la secante tra  $(x_0, f(x_0))$  e  $(x_1, f(x_1))$ . Posso quindi scrivere

$$y - f(x_1) = \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_1)$$

L'idea è che in  $x$  la funzione deve annullarsi, quindi  $y = 0$ .

$$0 = f(x_1) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}x - \frac{f(x_1) - f(x_0)}{x_1 - x_0}x_1$$

$$x = x_1 - \frac{x_1 - x_0}{f(x_1) - f(x_0)}f(x_1)$$

È da intendersi in senso iterativo in questo modo, se ho  $x_1$  e  $x_0, x$ , posso assegnare indici come  $x_n = x_1, x_{n-1} = x_0, x_{n+1} = x$ , perciò

$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}f(x_n)$$

Questo non è nient'altro che l'algoritmo di Newton-Raphson con il rapporto incrementale esplicito, senza limite. Se per Newton-Raphson, l'indice  $m$  della relazione  $\varepsilon_{n+1} = k|\varepsilon_n|^m$  è uguale a 2, per il metodo della secante si dimostra coincidere a  $m = \frac{1+\sqrt{5}}{2} \approx 1.6$ . Una variante di questo è l'algoritmo della **Falsa Posizione**.

### 10.2.4 Algoritmo della Falsa Posizione

Quando identifico la secante, possono capitare casi in cui mi allontano dallo zero. Con la falsa posizione posso fissare un estremo e andare a vedere un ordinamento per il quale non posso allontanarmi dallo zero. Posso fare la secante tra una successione di punti e un estremo fisso. Determinare il picco di radiazione emessa non si può fare analiticamente, poiché bisogna trovare il massimo di  $f(x) = \frac{x^3}{e^x - 1}$ , ma si può fare con una semplice applicazione della ricerca degli zeri. Si possono generare da qui **Bellissimi Frattali**.

### 10.2.5 Algoritmo quadratico di Rosignoli

#### Idea di considerare il second'ordine

Se si riprende il polinomio di Taylor dal quale si è estratto l'algoritmo di Newton-Raphson, Per una funzione  $C^{(2)}(\mathbb{C})$  vale

$$f(z_0) = f(z + \delta) = f(z) + f'(z)\delta + \frac{1}{2}\delta^2 f''(z) + \frac{1}{6}\delta^2 f'''(z) + \dots$$

Se invece di troncare al primo odine tronco al secondo, posso isolare un'equazione di secondo grado in  $\delta = dz$ .

$$0 = f(z) + f'(z)\delta + \frac{1}{2}\delta^2 f''(z)$$

Chiamo  $f = f^{(0)}$ ,  $f' = f^{(1)}$ ,  $f'' = f^{(2)}$  e  $f^{(i)}(z_0) = f_0^{(i)}$ . Con questa notazione risolvo l'equazione di secondo grado

$$\begin{aligned} \delta &= \frac{-f^{(1)} \pm \sqrt{f^{(1)^2} - 2f^{(0)}f^{(2)}}}{f^{(2)}} = -\frac{f^{(1)}}{f^{(2)}} \left( 1 \mp \sqrt{1 - 2\frac{f^{(0)}f^{(2)}}{f^{(1)^2}}} \right) \\ &\approx -\frac{f^{(1)}}{f^{(2)}} \left( \frac{f^{(2)}f^{(0)}}{f^{(1)^2}} + \frac{f^{(2)^2}f^{(0)^2}}{2f^{(1)^4}} + \frac{f^{(2)^3}f^{(0)^3}}{2f^{(1)^6}} \right) \\ &= -\frac{f^{(0)}}{f^{(1)}} - \frac{f^{(2)}f^{(0)^2}}{2f^{(1)^3}} - \frac{f^{(2)^2}f^{(0)^3}}{2f^{(1)^5}} \end{aligned}$$

Sfrutto l'espansione di taylor della radice  $\sqrt{1+z} \approx 1 + \frac{1}{2}z - \frac{1}{8}z^2 + \frac{1}{16}z^3$  e seleziono la soluzione con segno positivo, cioè il + nella prima espressione. Si può riconoscere che il primo termine dello sviluppo di  $\delta$  coincide con l'algoritmo di Newton-Raphson, perciò posso scrivere

$$\delta_{i+1} = \delta_i - \frac{f^{(0)}}{f^{(1)}} - \frac{f^{(2)}f^{(0)^2}}{2f^{(1)^3}} - \frac{f^{(2)^2}f^{(0)^3}}{2f^{(1)^5}}$$

Riesco ad indentificare il termine  $\delta_i - \frac{f^{(0)}}{f^{(1)}}$  come il termine  $\frac{1}{2}\delta_i^2 \frac{f_0^{(2)}}{f_0^{(1)}}$ , dal quale si deduce la convergenza quadratica del metodo di N-R. In questo breve scritto si deduce che considerando ordini superiori si raggiungono convergenze più veloci dell'ordine quadratico.

### Velocità di convergenza

Se sviluppo le funzioni  $f^{(i)}$  in un intorno del punto  $z$  tale che  $z_0 = z + \delta$ , posso sviluppare

$$f^{(i)} \approx f_0^{(i)} + \delta f_0^{(i+1)} + \frac{\delta^2}{2} f_0^{(i+2)} + \frac{\delta^3}{6} f_0^{(i+3)}$$

Con l'indice  $i = 0, 1, 2$ . Si può riprendere l'espressione di  $\delta_{i+1}$ , ora

$$\delta_{i+1} = \frac{1}{2} \delta_i^2 \frac{f_0^{(2)}}{f_0^{(1)}} - \frac{f^{(2)} f^{(0)2}}{2 f^{(1)3}} - \frac{f^{(2)2} f^{(0)3}}{2 f^{(1)5}}$$

Considerando un modulo di calcolo simbolico per svolgere sinteticamente i conti, si dimostra che considerando i primi due dei tre termini a secondo membro, si riesce a cancellare il coefficiente di  $\delta_i^2$ , nel seguente modo

$$\begin{aligned} \delta_{i+1} &\approx \cancel{\frac{1}{2} \delta_i^2 \frac{f_0^{(2)}}{f_0^{(1)}}} - \frac{1}{2} \left( \cancel{\delta_i^2 \frac{f_0^{(2)}}{f_0^{(1)}}} + \delta_i^3 \left( \frac{f_0^{(3)}}{f_0^{(1)}} - 2 \frac{f_0^{(2)2}}{f_0^{(1)2}} \right) + O(\delta_i^4) \right) \\ &= -\frac{1}{2} \delta_i^3 \left( \frac{f_0^{(3)}}{f_0^{(1)}} - 2 \frac{f_0^{(2)2}}{f_0^{(1)2}} \right) \end{aligned}$$

Si nota che con la scomparsa del coefficiente di  $\delta_i^2$ , come succede per l'algoritmo di Newton-Raphson, porta la velocità di convergenza ad una potenza successiva. Perciò, per l'algoritmo quadratico, vale

$$\varepsilon_{n+1} = k \varepsilon_n^3$$

Appare logico supporre che considerando in modo esatto o anche approssimato, mediante lo sviluppo di Taylor, i primi  $n$  ordini della funzione sulla quale stiamo applicando l'algoritmo di ricerca degli zeri, si ottiene una velocità di convergenza superQuadratico e, per la precisione, del tipo  $n + 1$ .

```
Sviluppo complessivo quadratico
coefficiente di d1**0 = 0
coefficiente di d1**1 = 0
coefficiente di d1**2 = 0
coefficiente di d1**3 = -g3/(6*g1) + g2**2/(2*g1**2)
coefficiente di d1**4 = -g4/(4*g1) + 2*g2*g3/g1**2 - 5*g2**3/(8*g1**3)
coefficiente di d1**5 = -g5/(12*g1) + 3*g2*g4/(4*g1**2) + 2*g3**2/(3*g1**2) - 5*g2**2*g3/(8*g1**3) - 5*g2**4/(8*g1**4)
```

Sviluppando a due ordini superiori il termine di N-R emergono i coefficienti di  $\delta_i^n$  e si nota che i primi tre coefficienti sono nulli, perciò l'algoritmo ha velocità di convergenza superQuadratico e, nello specifico, cubica.

$$\delta_{i+1} = \left( -\frac{f_0^{(3)}}{6 f_0^{(1)}} + \frac{f_0^{(2)2}}{2 f_0^{(1)2}} \right) \delta^3$$

# Chapter 11

## Lezione 11

### 11.1 Implementazione della ricerca degli zeri

#### 11.1.1 Bisezione

Abbiamo introdotto l'algoritmo di Bisezione, di Newton e due varianti della secante. Gli algoritmi si trovano nella folder **rootFinding**. L'algoritmo individua progressivamente il sottointervallo di estensione metà della precedente. Se ogni volta l'intervallo si dimezza,  $2^{40}$ , produce un numero circa uguale a  $10^{-12}$ , vicino all'accuratezza di macchina della doppia precisione. Bisogna sempre controllare se  $\delta x < \text{accuratezza}$ .

#### 11.1.2 Algoritmo di Newton-Raphson

Si basa sulla formulazione

$$\delta_i = -\frac{f(x_i)}{f'(x_i)}$$

Prima l'algoritmo legge l'intervallo e i valori estremali della funzione. Calcola la derivata agli estremi. Fa i controlli per vedere che lo zero sia all'interno dell'intervallo. Si assegnano  $x_1$  e  $x_2$  in base al segno della derivata. Si fa un'assegnazione di tentativo con il punto di mezzo dell'intervallo e chiama  $\delta_x = \frac{b-a}{100}$ . Nel calcolare la derivata è importante che l'incremento della derivata sia contenuto all'interno dell'intervallo nel quale stiamo cercando la soluzione. L'idea è quella di cercare lo zero della derivata. Se lo scostamento tra le stime progressive è minore di  $10^{-14}$  si ritorna la soluzione individuata. Se il controllo non ha dato esito positivo si aggiorna la derivata in quell'intervallo e viene ripetuto il ciclo. Il problema di Newton-Raphson è che noi individuiamo un intervallo in cui è sicuramente intrappolata la soluzione. Al passo successivo, per valori piccoli della derivata, è possibile che si abbia un  $\delta$  molto grande. Si può provare a vedere nel dettaglio. Sono tutti molto simili tra loro e non ne varrebbe la pena di studiare nel dettaglio gli algoritmi.

#### 11.1.3 Confronto tra i vari algoritmi

Si fa un confronto tra la velocità di convergenza dei vari algoritmi tramite lo script **driver\_rootfinding.f**. Facciamo il confronto tra bisezione, secante, falsa posizione e Newton - Raphson. Cominciamo a vedere una funzione di tentativo come  $\cos(x)$ . Compiliamo tramite **gfortran -g driver\_rootfinding.f newton.f secant.f falseposition.f bisection.f**. Analiticamente si può dare una stima precisa di dove sono gli zeri. Si può partire da quelle e si va a cercare la soluzione precisa con gli algoritmi. Si implementa Gauss-Legendre di ordine generico con un algoritmo di questo genere. La ricerca degli zeri si utilizza a partire da degli zeri di tentativo. Se ho già una stima ragionevole di dove possa essere, ci è più utile. Con una funzione periodica, se partiamo con un intervallo che ha più di uno zero, si può arrivare con diversi algoritmi all'individuazione di zeri diversi.

```

looking for the root with bisection
x_0= 14.137573242187500 with 14 bisections
f(x_0)= -4.0630102225169536E-004
looking for the root with secant method
x_0= -23.561944901834611 with 9 iterations
f(x_0)= -8.8838300380345285E-011
looking for the root with false position
x_0= 7.8539816339744828 with 8 iterations
f(x_0)= 3.0616169978683831E-016
looking for the root with Newton-Raphson
x_0= 7.8539816339744908 with 3 iterations
f(x_0)= -7.6874440775142892E-015

```

La soluzione trovata con Newton-Raphson, curato del fatto che la soluzione non si cerca fuori dall'intervallo. Quando l'algoritmo produce uno scostamento che fa uscire dall'intervallo, si fa uso della bisezione. Inserendo come valori di innesco  $x_1 = 0$  e  $x_2 = 3$ , con accuratezza cercata di  $10^{-4}$ , otteniamo invece

```

looking for the root with bisection
x_0= 1.5708618164062500 with 15 bisections
f(x_0)= -6.5489611306567821E-005
looking for the root with secant method
x_0= 1.57079632679781092 with 4 iterations
f(x_0)= -1.1832125777453532E-009
looking for the root with false position
x_0= 1.5707963267948966 with 5 iterations
f(x_0)= 6.1232339957367660E-017
looking for the root with Newton-Raphson
x_0= 1.5707963267948966 with 3 iterations
f(x_0)= 6.1232339957367660E-017

```

con varie stime di  $\frac{\pi}{2}$ . La differenza tra la falsa posizione e Newton-Rapshon è di 5 iterazioni rispetto a 3, anche se con precisione confrontabile. Se rimaniamo sullo stesso intervallo ma proviamo a spingerci a una precisione di  $10^{-10}$ ,

```

looking for the root with bisection
x_0= 1.5707963268505409 with 35 bisections
f(x_0)= -5.5644316761872885E-011
looking for the root with secant method
x_0= 1.5707963267948966 with 6 iterations
f(x_0)= 6.1232339957367660E-017
looking for the root with false position
x_0= 1.5707963267948966 with 6 iterations
f(x_0)= 6.1232339957367660E-017
looking for the root with Newton-Raphson
x_0= 1.5707963267948966 with 3 iterations
f(x_0)= 6.1232339957367660E-017

```

Vedo che il metodo di bisezione comincia a saturare il numero massimo di iterazioni.

```

too many bisections in rtbis
looking for the root with bisection
x_0= 1.5707963267959713 with 40 bisections
f(x_0)= -1.0746346554971943E-012
looking for the root with secant method
x_0= 1.5707963267948966 with 6 iterations
f(x_0)= 6.1232339957367660E-017
looking for the root with false position
x_0= 1.5707963267948966 with 6 iterations
f(x_0)= 6.1232339957367660E-017
looking for the root with Newton-Raphson
x_0= 1.5707963267948966 with 4 iterations
f(x_0)= 6.1232339957367660E-017

```

Il gioco di localizzare maggiormente lo zero lo posso fare e se voglio esagerare con la precisione, con la doppia precisione non è possibile, devo aumentare la precisione dei float con cui calcolo.

## 11.2 Lunghezza di picco

Si può cercare di applicare il concetto della ricerca degli zeri all'individuazione della lunghezza di picco per lo spettro di corpo nero.

$$I(v)dv \simeq \frac{hv^3}{c^2} \frac{1}{e^{\frac{hv}{kT}} - 1} dv$$

Vogliamo trovare la lunghezza d'onda in corrispondenza del picco della distribuzione. Se conosco  $\nu = \frac{c}{\lambda}$ , ottengo quindi  $\frac{d\nu}{d\lambda} = -\frac{c}{\lambda^2}$ , trovo perciò

$$I(\lambda) \sim I(\nu) \left| \frac{d\nu}{d\lambda} \right|$$

Il valore della frequenza non posso convertirlo nel valore della lunghezza d'onda se passo per la curva di Planck. Non posso convertire  $\nu_{max}$  in  $\frac{c}{\lambda_{max}}$ . Perché ho una distribuzione in lunghezza pari a quella in frequenza per lo Jacobiano.

$$E \simeq \int_0^\infty I(\nu) d\nu = \int_0^\infty I_\lambda(\lambda) d\lambda$$

Perciò

$$I_\lambda(\lambda) = \frac{hc^3}{c^2\lambda^3} \frac{c}{\lambda^2} \frac{1}{e^{\frac{hc}{kT\lambda}} - 1} = \frac{hc^2}{\lambda^5} \frac{1}{e^{\frac{hc}{kT\lambda}} - 1}$$

Per trovare il massimo dobbiamo individuare lo zero della derivata.

$$\begin{aligned} \frac{dI_\lambda}{d\lambda} &= -5 \frac{hc^2}{\lambda^6} \frac{1}{e^{\frac{hc}{kT\lambda}} - 1} + \frac{hc^2 hc}{\lambda^7} \frac{e^{\frac{hc}{kT\lambda}}}{(e^{\frac{hc}{kT\lambda}} - 1)^2} \\ &= \frac{hc^2}{\lambda^6} \frac{1}{e^{\frac{hc}{kT\lambda}} - 1} \left[ -5 + \frac{hc}{kT\lambda} \frac{e^{\frac{hc}{kT\lambda}}}{e^{\frac{hc}{kT\lambda}} - 1} \right] \end{aligned}$$

Possiamo trascurare il primo termine a coefficiente e ci conviene utilizzare la variabile adimensionata  $x = \frac{hc}{kT\lambda}$ . La condizione si trasforma in

$$\begin{aligned} -5 + x \frac{e^x}{e^x - 1} &= 0 \\ xe^x - 5(e^x - 1) &= 0 \\ x - 5 + 5e^{-x} &= 0 \\ \frac{x}{5} - 1 + e^{-x} &= 0 \end{aligned}$$

Trovato lo zero di questa condizione in  $x$ , fissata una temperatura possiamo trovare la  $\lambda$  di picco. Rivedendo le trasformazioni effettuate si vede che

$$\bar{\lambda}_{max} = \frac{hc}{kT\bar{x}_{max}}$$

che sarebbe la legge di Vien. Noi, una volta identificato il valore numerico di  $x$ , potremo ricavare il valore fisico di  $\lambda$  mediante la trasformazione inversa  $\lambda = \frac{kT}{hc} \frac{1}{x}$ .

### 11.2.1 Estrapolazione numerica

Provo a ricavarmi il valore di  $x$  mediante gli algoritmi di individuazione degli zeri e pongo come estremi  $(0, 10)$ .

```

0.0000000000000000      0.0000000000000000
1.111111193895340     -0.44858479103938631
2.222222387790680     -0.44718753081652318
3.3333334326744080    -0.29765932366175862
4.444444775581360    -9.9367476420226364E-002
5.5555558204650879    0.11497708320837141
6.6666668653488159    0.33460600661825346
7.7777779102325439    0.55597452411446624
8.8888889551162720    0.77791570382345743
10.0000000000000000   1.0000453999297623
lambda_max= 2.4900958010171695E-007
lambda_min=          Infinity
looking for the root with Newton-Raphson
x_0= 0.0000000000000000      with      0 iterations
f(x_0)= 0.0000000000000000
T= 5778.0000000000000000      K
lambda at maximum=          Infinity m

```

Poiché la funzione nell'intervallo ha più di uno zero, come nel caso del coseno, a seconda dell'algoritmo impiegato si individua uno zero diverso. Siccome l'algoritmo prova a calcolare la funzione agli estremi, se all'estremo è zero, allora l'algoritmo si ferma e restituisce lo zero. Anche se pongo come estremo destro un

numero più grande ottengo lo stesso risultato. Se invece sposto l'estremo sinistro di una piccola quantità a destra, ottengo un risultato differente. Impostiamo come estremi ( $10^{-5}$ ,  $10$ ) e come accuratezza  $10^{-8}$ .

```

1.000000000000001E-005 -7.9999500002259083E-006
1.111200082784147 -0.44858593940846980
2.222300165568294 -0.44718681812008021
3.3333400993409756 -0.29765822815424758
4.4444500331136583 -9.9366430551318441E-002
5.5555602649092677 0.11497795491538376
6.6666701986819508 0.33460666904277581
7.7777801324546338 0.55597496762790288
8.8888900662273169 0.77791592589242997
10.000000000000000 1.0000453999297623
lambdamax= 2.4900958010171695E-007
lambdamin= 0.24900958010171692
looking for the root with Newton-Raphson
x_0= 4.9651142317442760 with 3 iterations
f(x_0)= -1.1102230246251565E-016
T= 5778.0000000000000 K
lambda at maximum= 5.0151833065528140E-007 m

```

Adesso è entrato nella routine che calcola le iterazioni di Newton-Raphson con una radice identificata in  $x_0 = 4.96511423$  con 3 iterazioni. [Mi sono ricavato tranquillamente  $(5 - 0.034885768255)^{-1} \approx .201405$  come valore adimensionale, il suo reciproco è  $5 + W((-5)e^{-5}) = 4.965114231744276303695$ .] La lunghezza di picco è di 500nm, che corrisponde alla lunghezza dello spettro del verde. La funzione in  $x_0 \approx 10^{-16}$ , quindi è una buona stima dello zero. Resituisce i valori che ci aspettiamo sensati della cosa. Se ripetiamo il calcolo con la temperatura ambiente  $T = 287.17K$  risulta

```

1.000000000000001E-005 -7.9999500002259083E-006
1.111200082784147 -0.44858593940846980
2.222300165568294 -0.44718681812008021
3.3333400993409756 -0.29765822815424758
4.4444500331136583 -9.9366430551318441E-002
5.5555602649092677 0.11497795491538376
6.6666701986819508 0.33460666904277581
7.7777801324546338 0.55597496762790288
8.8888900662273169 0.77791592589242997
10.000000000000000 1.0000453999297623
lambdamax= 5.0131615115948445E-006
lambdamin= 5.0131615115948440
looking for the root with Newton-Raphson
x_0= 4.9651142317442760 with 3 iterations
f(x_0)= -1.1102230246251565E-016
T= 287.0000000000000 K
lambda at maximum= 1.0096769737025143E-005 m

```

Risulta una lunghezza di picco di  $\lambda \approx .1\mu m$ .

### 11.2.2 Curiosità

Si possono ricondurre i problemi di ricerca degli zeri di polinomi ortogonali a diagonalizzazioni di matrici con individuazione degli autovalori. Si possono produrre frattali mediante l'algoritmo di individuazione degli zeri. Per fare questo bisogna produrre numeri random e studiare il metodo Monte Carlo (MC).

# Chapter 12

## Lezione 12

### 12.1 Applicazioni ricerca degli zeri

#### 12.1.1 Polinomi di Gauss-Legendre

Vediamo come vengono applicati i metodi di ricerca degli zeri in altri problemi. Abbiamo visto gli algoritmi di Gauss, in generale. Avevamo visto negli algoritmi di Gauss che l'algoritmo funziona se troviamo gli zeri del polinomio di Gauss associato di ordine  $n$ . Nel tempo sono stati elaborati dei metodi per fornire delle stime analitiche degli zeri. Se devo trovare  $n$  radici per  $n > 2$  può essere laborioso da mettere in pratica. Se abbiamo stime analitiche degli zeri, anche se non precise, possono essere fornite come innesco all'algoritmo di ricerca dello zero, con una convergenza semi immediata alla radice nelle immediate vicinanze. C'è un esempio nella folder **qgauss.f** e **qgauleg.f**. La struttura è la medesima dell'altra volta, c'è a routine qgauss che richiama la ricerca delle ascisse e dei pesi, restituite nella common. La routine è gauleg. Si sfrutta una stima analitica degli zeri

$$z_i = \cos\left(\pi \frac{i - \frac{1}{4}}{n + \frac{1}{2}}\right)$$

Quest'algoritmo semplicemente calcola la radice approssimata e fa partire un ciclo di iterazioni di NR con cui si va a cercare la soluzione con il grado di tolleranza che vogliamo. Si ferma se il valore di  $z$  è vicino sotto una determinata accuratezza, fissata a  $3 \cdot 10^{-14}$ . Si lascia l'analisi approfondita agli studenti. Si studia il qgauss. Si sistema il codice e con dei parametri di input come 15 iterazioni per il trapezio, accuratezza di  $10^{-4}$ ,  $10^4$  chiamate al massimo per **qtrap** e **qsymp** con estremi di integrazione (0, 5) e 3 punti per Gauss-Legendre.

```
n. of points=      129    integral=   1225.4579925537109
n. of points=      257    integral=   1225.4269981384277
n. of points=      513    integral=   1225.4192495346069
n. of points=     1025    integral=   1225.4173123836517
n. of points=     2049    integral=   1225.4168280959129
n. of points=     4097    integral=   1225.4167070239782
n. of points=     8193    integral=   1225.4166767559946
n. of points=    16385    integral=   1225.4166691889989
n. of points=    32769    integral=   1225.4166672972522

result from qtrap
n. of points=      257    integral=   1225.4269981384277

result from qsymp
n. of points=      33    integral=   1225.41666666666667

abscissa and weights for Gauss-Legendre routine
xgauss(j)=      4.4364916731037081      1.38888888888888817
xgauss(j)=      2.5000000000000000      2.2222222222222223
xgauss(j)=      0.56350832689629149      1.38888888888888817

result from Gauss Laugerre with      3 points
integral=   1225.4166666666624
```

Il risultato di Sympson e il risultato di Gauss sono equivalenti per un polinomio di grado 3, difatti i risultati coincidono. Se saliamo con il grado del polinomio da 3 a 7 con  $P_7(x) = 3 + x + 25x^2 + x^3 + x^7$ , devo calcolare  $n = 4$  punti. Questo perché riesco ad integrare esattamente fino a un grado  $2d - 1 = 7$ .

```

n. of points=      32769    integral=   50053.541879512588
result from qtrap
n. of points=      513     integral=   50054.413482279662
result from qsymp
n. of points=       65     integral=   50053.568820767861
abscissa and weights for Gauss-Legendre routine
xgauss(j)=        4.8727697808568964    0.32371241542215617
xgauss(j)=        4.3538279639984863    0.69926347872319183
xgauss(j)=        3.5146128784434927    0.95457512626279761
xgauss(j)=        2.5000000000000000    1.0448979591836736
xgauss(j)=        1.4853871215565071    0.95457512626279761
xgauss(j)=        0.64617203600151374    0.69926347872319183
xgauss(j)=        0.12723021914310362    0.32371241542215617
result from Gauss-Laguerre with      7 points
                                         integral=   50053.541666665486

```

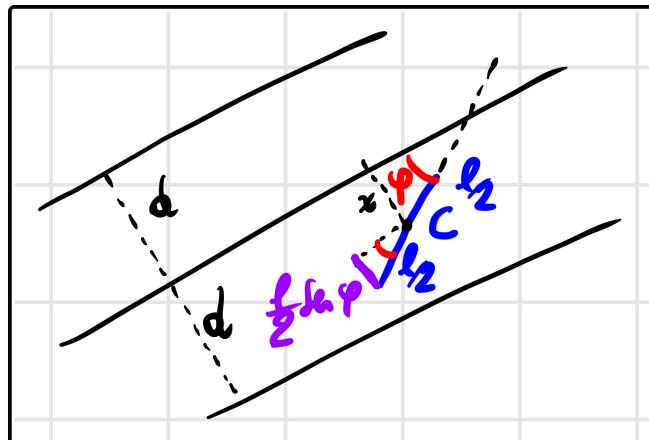
Trapezio e Sympson differiscono sotto 1/50.000 ma Sympson e Gauss-Legendre sono differenti, in quanto Gauss-Legendre integra esattamente mentre Sympson approssima con 65 chiamate.

## 12.2 Metodo Monte Carlo

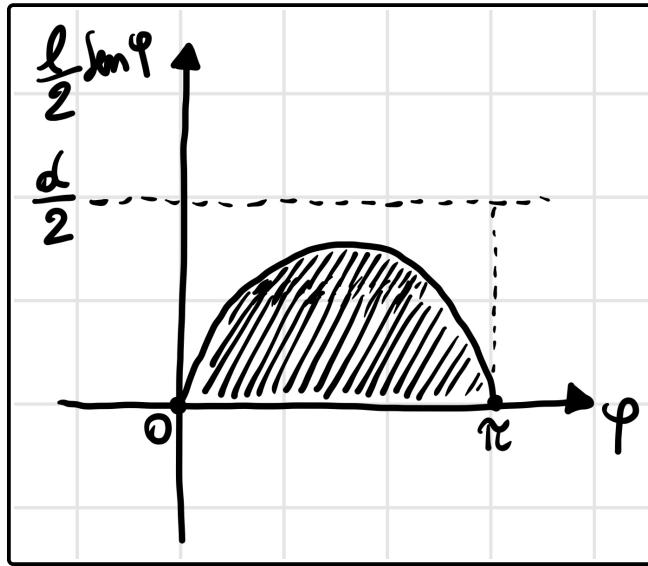
Si fa un'introduzione molto semplice, vedendo quello che ci interessa maggiormente, senza entrare nei dettagli. Si intende una classe di approcci ai metodi computazionali molto vasta che ha applicazioni nei settori più disparati. Ho un problema deterministico e lo voglio risolvere utilizzando variabili aleatorie. Il risultato deterministico deve diventare un valore medio di una variabile aleatoria appartenente ad una popolazione. C'è un esempio storico dell'applicazione del metodo Monte Carlo, cioè dell'**ago di Buffon**.

### 12.2.1 Ago di Buffon

Suppongo di avere delle strisce parallele equidistanti di  $d$  a terra. Suppongo di avere un'asta di lunghezza  $l < d$ . Lancio l'asta in aria e vedo cosa succede. Può al massimo incrociare una striscia. Ho due variabili aleatorie:  $x$  la distanza del centro  $C$  dalla striscia più vicina e  $\varphi$  l'angolo con cui la retta sulla quale soggiace l'asta interseca le strisce. Per  $x$  vale  $0 \leq x < d$ , mentre per  $\varphi$  vale  $0 < \varphi < \pi$ .



Il cateto  $i = \frac{l}{2} \sin \varphi$  ha la condizione di intersezione che  $i \geq x$ . Disegniamo il grafico della funzione  $f(\varphi) = \frac{l}{2} \sin \varphi$ .



La probabilità non è nient'altro che la frazione tra punti che caratterizzano un'intersezione e i punti globali. Se il punto  $(\varphi, x)$  è sotteso alla curva c'è stata intersezione. Abbiamo tutti i casi possibili che sono i punti del rettangolo  $[0, \pi] \times [0, d/2]$  e i casi favorevoli che sono i punti sottesi alla curva  $f(\varphi)$ .

$$\begin{aligned} p &= \frac{\int_0^\pi \frac{l}{2} \sin \varphi}{\frac{d}{2}\pi} = \frac{l}{\frac{d}{2}\pi} \\ &= \frac{2l}{d\pi} \end{aligned}$$

Questo è un risultato vero e posso fare una stima di questa probabilità andando a vedere con  $N$  lanci quante volte mi cade sotto la curva sul totale.

$$P_{\text{ext.}} = \frac{n}{N}$$

Io per grandi  $N$  posso eguagliare i due valori e ottenere

$$\begin{aligned} \frac{2l}{d\pi} &= \frac{n}{N} \\ \pi &= \frac{2lN}{dn} \end{aligned}$$

Da questa equazione posso estrarre una stima di  $\pi$ , dove il valor vero di  $\pi$  sarebbe  $\frac{2l}{dp}$ . Essendo una stia empirica ci serve conoscere l'errore associato a questa stima di  $\pi$ , che calcolo con la distribuzione Binomiale.

$$\langle \text{intersezioni} \rangle = Np$$

$$\text{Var}[\text{int.}] = Np(1 - p)$$

Questo ci permette di dare una stima dell'errore perché

$$\sigma^2\left(\frac{n}{N}\right) = \frac{1}{N^2}\sigma^2(n) = \frac{1}{N^2}Np(1 - p) = \frac{p(1 - p)}{N}$$

L'errore che associamo a questa serie di lanci è

$$\Delta p = \sqrt{\frac{p(1 - p)}{N}} \implies \Delta \pi = \frac{2l}{dp^2} \Delta p$$

Questo succedeva nel 1700. A meno che non si usi un'asta truccata il lancio è un processo casuale. Noi vogliamo calcolare  $\pi$ , un problema deterministico, ricorrendo ad un processo stocastico.

## 12.3 Generazione di numeri pseudo random

Come faccio ad utilizzare delle sequenze di numeri casuali? Utilizzo processi fisici come decadimenti radioattivi o lanci di dadi? Vogliamo strumenti utilizzabili in modo facile, come generare noi sequenze di numeri casuali, ma noi non ne siamo in grado. La sequenza è davvero random se l'algoritmo necessario a generare la sequenza contiene l'informazione della sequenza stessa. La sequenza può essere infinita, con

informazione infinita, mentre nell'algoritmo ho ridotto l'informazione. In pratica si usano numeri pseudo-random, sequenze di numeri generati con un algoritmo. Non sono veramente casuali ma lo sono da un punto di vista pratico. Le sequenze che generano somigliano a sequenze random.

### 12.3.1 Algoritmo di Von Neumann

è il metodo dei quadrati centrali. Parte dal fatto che prendiamo un intero e lo eleviamo al quadrato, prendendo le cifre centrali come parte della sequenza. Se partiamo da  $123 \rightarrow 15129$ , prendiamo le cifre 512, poi 262144, estraggo 214. Questo è un algoritmo codificabile in cui costruisco una sequenza di numeri che vorrei comparissero come casuali. Cosa vuol dire? Voglio usare 1000 numeri. La sequenza che genero non voglio che mi appaia correlata. Se realizzo al posto  $n$ -esimo un certo numero, apparentemente non mi determina il numero successivo. Questo lo posso fare operando dei test statistici. Ho il mio algoritmo con la mia sequenza, genero un istogramma da 10 – 100 bin. In questo modo categorizzo i miei numeri. Se alla fine la distribuzione la voglio uniforme, devo vedere delle fluttuazioni compatibili con la deviazione standard della distribuzione uniforme. Se ho  $n$  numeri e  $N$  bin, mediamente si distribuiscono  $\frac{n}{N}$  numeri per bin. Alla fine la distribuzione di probabilità in ogni bin sarà una distribuzione di Poisson, caratterizzata da un certo valor medio e da una varianza.

$$P(x, \mu) = \frac{\mu^x}{x!} e^{-\mu}$$

Mi vado a calcolare il  $\chi^2$  con i vari bin.

$$\chi^2 = \sum_{i=1}^N \left( \frac{M_j - \frac{n^2}{N}}{\sqrt{\sigma^2}} \right)^2$$

Vado a vedere il minimo del  $\chi^2$  tenendo conto dei vari gradi di libertà. Tengo conto dei gradi libertà, cioè che il totale delle entrate dei bin deve dare  $n$  totale. Se il  $\chi^2$  finale risulta dell'ordine di 1. Bisogna vedere anche le correlazioni. Questi algoritmi che generano sequenze di numeri pseudo-casuali possono generare correlazioni tra coppie di punti a distanza  $m$ . Se studio le correlazioni che ci sono tra questi punti e non vedo correlazioni particolari, è un altro sintomo che posso usare la sequenza fino a quel numero di punti che ho studiato statisticamente come sequenza pseudo-random. Non è detto che l'algoritmo generi per qualunque  $n$  una sequenza che appare sempre casuale. Se la sequenza passa tutti i test statistici allora posso utilizzarla. Con lo sviluppo dei calcolatori, adesso, fare calcoli con sequenze di numeri casuali fino alla  $10^{11}$  punti non presenta nessun problema. Si fa anche nei cluster, nelle farm di migliaia di PC. Posso lavorare con  $10^9$  punti per ogni computer e se ne ho 1000, posso lavorare con  $10^{12}$  punti.

### 12.3.2 Algoritmo della congruenza lineare

C'è un generatore di numeri casuali dell'IBM degli anni '60 dove si vede l'algoritmo Marsaglia di default tra tutti i calcolatori, che però a 11 punti generava punti correlati del 100%. Sono algoritmi del tipo

$$\begin{aligned} &x_0 \text{ seed} \\ &x_{n+1} = (ax_n + c) \bmod m \end{aligned}$$

Genera automaticamente una sequenza di numeri random tra  $[0, m - 1]$ . Si chiama algoritmo della congruenza lineare. Se voglio una sequenza con  $m$  grande, devo prendere  $m \approx 2^{30}$ . Un'evoluzione di questo sono gli algoritmi del metodo congruenziale come

$$x_{n+1} = (x_n + x_{n-k}) \bmod m$$

Questi sono solo esempi. La costruzione di algoritmi di sequenze di numeri pseudo-random è una branca della matematica applicata in continua evoluzione, perché ci interessa avere sequenze sempre maggiori. C'è un generatore che si chiama **Ranlux**, che non è di default, generato al CERN da un estremo di Computer Science, F. James che ha la proprietà che al momento non sono state individuate correlazioni di alcun tipo. Qualunque algoritmo ha un **periodo** se torna su un punto, perché da lì in poi sarà periodico. Con lo sviluppo della velocità di calcolo ci servono periodi sempre più grandi. Non si è trovato ancora il periodo di Ranlux. Questo metodo di generare numeri pseudo-casuali può sembrare affatto dalla periodicità della sequenza generata, d'altro lato se i test statistici che faccio mi dicono che posso fidarmi di quello che sto usando, hanno il vantaggio che fissati

i parametri dell'algoritmo, si fissa  $x_0$  come input della sequenza. Se governa la generazione della sequenza mediante  $x_0$ , questo mi dà una maniglia importante per ripetere la soluzione del problema. È importante poter riprodurre quello che ho visto, da questo punto di vista, se uso lo stesso  $x_0$ , ottengo il medesimo risultato. Potrei inizializzare  $x_0$  con un generatore random che prende il tempo del calcolatore. Per ogni ripetizione non avrò mai lo stesso risultato, che per i numeri casuali va bene, ma se mi accorgo di un problema nella sequenza, non posso poi risalire al problema e risolverlo.

### 12.3.3 Generatore built in

Tutti i calcolatori hanno un generatore di numeri random built-in. La convenzione è che i generatori producono una sequenza di punti distribuiti uniformemente nell'intervallo  $[0, 1]$ . È un elemento da ricordare. Ci possono servire distribuzioni di probabilità diverse dall'uniforme ma per ciò si vedrà.

### 12.3.4 Particularità N-R.

Si può andare a vedere la particolarità interessante data dall'algoritmo di N-R. per cercare gli zeri.

$$\delta_{i+1} = \delta_i - \frac{f(x_i)}{f'(x_i)}$$

È un algoritmo iterativo. Si provi ad immaginare di risolvere l'equazione  $z^3 - 1 = 0$ . Noi conosciamo le soluzioni  $z = 1, e^{\pm\frac{2}{3}\pi}$ . Si dispongono a triangolo equilatero. Se io innesco l'algoritmo con un numero complesso, come conosco il **bacino di convergenza** dell'algoritmo? Ciò è quanto descritto nell'algoritmo di **rootfinding** sotto il nome di convergeregion. In base al punto d'innenisco, a quale soluzione convergo? Mi serve generare una coppia di numeri per individuare un punto nel piano complesso. Uso il generatore **Ranlux**. Imposto il numero di punti M.C. che vado a testare e l'intero seed che inizializza la sequenza. È una versione dell'algoritmo che ha un po' d'anni ma si basa su studi di Computer Science e ancora oggi fornisce i risultati desiderati. Facciamo un ciclo M.C. e l'altra componente del generatore è l'algoritmo vero e proprio, cioè la subroutine runlux. Se fisso **len** a 2, otterrò un array di due numeri alla volta. Se il generatore produce numeri in **real\*4** e i faccio i conti in **real\*8**, non è un problema, in quanto posso semplicemente convertirli. Se prendo la  $x$ , il primo della coppia e lo moltiplico per l'estensione dell'intervallo, trasformo l'intervallo  $(0, 1)$  nell'intervallo  $(-1, 1)$ .

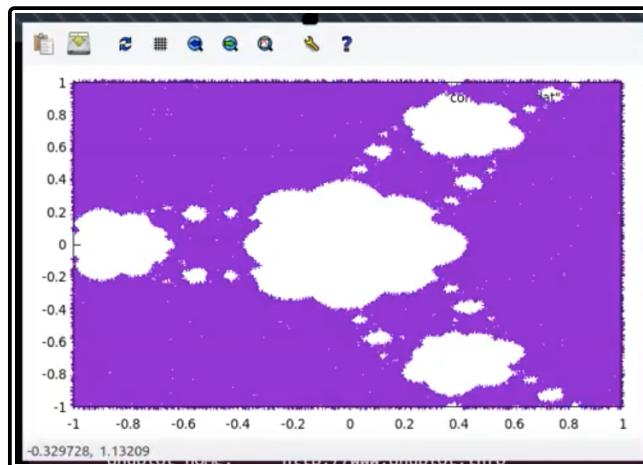
$$x_i \in [0, 1]$$

$$y = y_{\min} + (y_{\max} - y_{\min})x_1$$

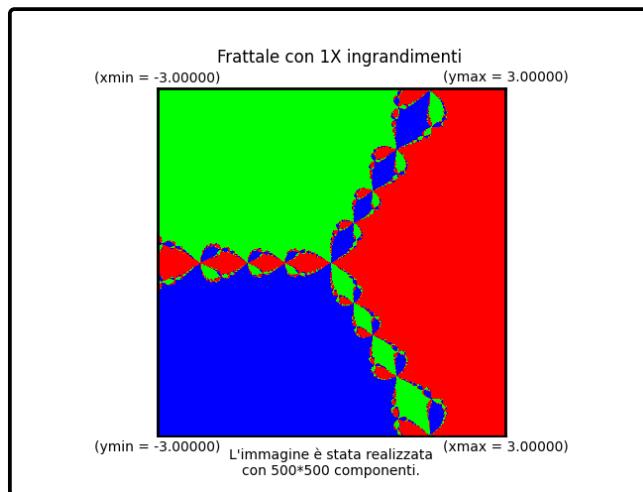
Con una trasformazione convessa. La trasformazione è lineare e se  $x_1$  appartiene alla trasformazione uniforme, anche  $y$  vi appartiene. Ricordiamoci che io estraggo random il numero di partenza. Faccio un ciclo di  $10^4$  iterazioni dove applico N-R.

$$z_{i+1} = z_i - \frac{f(z_i)}{f'(z_i)}$$

Il modulo del numero complesso è il raggio polare. L'argomento d'altronude è l'angolo polare. Se il modulo è maggiore di  $\sqrt{2}$ , sono uscito dal riquadro originale. L'algoritmo sostanzialmente si segna se dal punto di partenza ha trovato uno zero, altrimenti se non lo trova non lo segna. Per questo motivo si chiama bacino di convergenza. Se setto come massimo numero di punti  $10^5$  ho la  $x$  e la  $y$  dei punti che sono andati a convergere nei punti noti. Se provo a plottare il **convergence.dat**, mi va a porsi il puntino nel piano, si trova questo.



Se invece si segue l'algoritmo di N-R come l'ha sviluppato Rosignoli, si trova



La cosa interessante è la simmetria e le zone più piccole sono *self-similari*. Se noi dovessimo restringere, troveremmo un pattern grafico che si ripete. Con NR si trovano gli Julia sets. Questa caratteristica emerge dalla non linearità utilizzata iterativamente.

# Chapter 13

## Lezione 13

### 13.1 Metodo Monte Carlo

Ciò che io visualizzo con i frattali sono i bacini di convergenza, non gli zeri della funzione. Utilizzare il metodo MC significa risolvere un problema deterministico, impostato come problema aleatorio, sfruttando il valor medio di variabili aleatorie. Il Monte Carlo ha diverse accezioni in fisica, sostanzialmente tre e strettamente legati.

- Risoluzione di un problema ben specifico come l'integrazione numerica. Si vuole trovare un numero e si usa il metodo MC per calcolare l'integrale.
- Simulazione numerica. Vuol dire che andiamo a simulare gli outcome di un processo fisico di natura stocastica. Si studiano poi medie d'insieme. Si mostra come si applica il MC nell'ambito finanziario.
- Dai generatori standard escono distribuzioni di numeri standard nell'ipercubo di lato 1 e dimensione il vettore desiderato.

In tutte le applicazioni del MC si trovano le idee di integrazione stocastica. Si studiano questi tre aspetti e se ne vedono le correlazioni. Sono aspetti diversi per utilizzi diversi ma con lo stesso metodo sottostante.

### 13.2 Integrazione mediante MC

Cosa vuol dire fare un integrale con il metodo di Monte Carlo. L'idea è di sfruttare i numeri pseudo-random. Abbiamo una sequenza di numeri  $\{x_i\}$  che interpretiamo come variabile stocastica distribuita uniformemente. Noi integriamo in un intervallo  $(a, b)$ . Con i metodi deterministicci abbiamo partizionato l'intervallo e calcolato la funzione sul reticolo. A seconda di come si prende questa reticolazione se equispaziata o non equispaziata, si possono ottenere precisioni diverse e si possono integrare classi di funzioni in modo esatto. Proviamo a comprendere media e deviazione standard della funzione  $f(x_i)$ ?

#### 13.2.1 Distribuzione di probabilità

Si scrive un forma generale per la distribuzione di probabilità dei valori  $f$  che andiamo ad estrarre.

$$\pi(f) = \int_0^1 p(x)\delta(f(x) - f) = \int_0^1 \delta(f(x) - f)dx$$

Questo mi dà un modo per caratterizzare in modo univoco la **pdf** della  $f$ .

$$\int \pi(f)df = \int df \int_0^1 dx \delta(f(x) - f) = \int_0^1 dx \int df \delta(f(x) - f) = \int_0^1 dx = 1$$

Questo mi dice che se prendo  $\delta(f(x) - f)dx$  come **pdf** della mia  $f$ , è già normalizzata. Ne studiamo il valor medio.

$$\langle f \rangle = \int df f \pi(f) = \int df f \int_0^1 dx \delta(f(x) - f) = \int_0^1 dx \int df f \delta(f(x) - f) = \int_0^1 f(x)dx$$

Questo stabilisce il legame tra il valor medio di  $f$  e l'integrale definito.

$$\int_0^1 f(x)dx = \langle f \rangle \simeq \frac{1}{N} \sum_{i=1}^N f(x_i)$$

Possiamo ottenere una realizzazione particolare del valor medio con un determinato numero di punti. Questa è la formula basilare che ci lega il nostro integrale definito con il calcolo di una media di quantità stocastiche. Si intende processo stocastico di variabile aleatoria  $f(x)$ . Mi calcolo la sua varianza.

$$\sigma_f^2 = \langle (f - \langle f \rangle)^2 \rangle = \langle f^2 \rangle - 2\langle f \rangle^2 + \langle f \rangle^2 = \langle f^2 \rangle - \langle f \rangle^2$$

Posso equivagliarla all'integrale

$$\sigma_f^2 = \int_0^1 dx f^2(x) - \left[ \int_0^1 dx f(x) \right]^2$$

Le richieste sulla  $f$  si arricchiscono della richiesta che  $f$  sia quadrato sommabile.

### 13.2.2 Momenti del valor medio ricavato con $N$ punti

Il valor medio con  $N$  punti è un estimatore dell'integrale, ma voglio conoscere quanto è buona la stima. Perciò calcolo l'errore associato al valore medio.

$$\varphi = \frac{1}{N} \sum_{i=1}^N f(x_i)$$

è la somma di variabili casuali. Se possiamo capire qual'è il range in cui si muove statisticamente  $\varphi$  abbiamo una corrispondente stima dell'errore. Come si pone  $\varphi$  con un numero  $N$  di estrazioni rispetto al valore vero. Ci viene in aiuto il teorema del limite centrale. La somma di variabili casuali **i.i.d.** tende a una Gaussiana. Perciò posso calcolare l'errore gaussiano associato a  $\varphi$ .

$$\begin{aligned} \langle \varphi \rangle &= \frac{1}{N} N \int_0^1 f(x)dx \\ \sigma^2(\varphi) &= \frac{1}{N} \sigma_f^2 \end{aligned}$$

Posso anche scrivere

$$\left\langle \frac{1}{N} \sum_{i=1}^N f(x_i) \right\rangle = \frac{1}{N} \sum_{i=1}^N \langle f(x_i) \rangle = \frac{N}{N} \langle f(x) \rangle = \int_0^1 f(x)dx$$

Abbiamo sfruttato la linearità della media. La media converge alla stessa media di tutta la popolazione. È interessante dal punto di vista pratico dell'integrale. Se osservo la relazione per la varianza, trovo che

$$\sigma_\varphi = \frac{1}{\sqrt{N}} \sigma_f$$

Facciamo  $N$  realizzazioni con una dispersione attorno al valor vero. La sigma di questa media ci caratterizza quant'è questa dispersione. Ha un significato statistico, se abbiam un numero sufficientemente alto di punti abbiamo che stiamo rispettando la legge dei  $3\sigma$ . Si fa il valor medio, la deviazione standard ci dice l'errore da associare al Monte Carlo. La s.d. della  $f$  è fissa e quella sul valor medio invece, va come la s.d. della  $f$  per il coefficiente  $\frac{1}{\sqrt{N}}$ . I numeri pseudo-random possono essere distribuiti in  $d$  dimensioni. La convergenza dell'integrale va in una dimensione come  $\frac{1}{\sqrt{N}}$ . Gli algoritmi deterministici avevano convergenze del genere  $\frac{1}{N^4}$  o addirittura esatti. Se sono interessato a integrare in una dimensione in tempi rapidi, lo devo fare con gli algoritmi deterministici. L'errore calcolato per gli algoritmi deterministici è la differenza tra il valore con  $N$  punti e il valore con  $N/2$  punti. È un criterio di ragionevolezza senza significato statistico associato. Per l'errore associato all'integrazione col metodo MC l'errore ha significato di errore statistico. Per lavorare con numeri in un intervallo  $(a, b)$ , devo sempre operare la trasformazione convessa  $y = a + (b - a)x$ . Dove  $x \in (0, 1)$ .

$$\int_a^b f(y)dy = \int_0^1 dx f(y(x))(b - a)$$

### 13.2.3 Utilizzo del MC in più dimensioni

Facendo il confronto con gli algoritmi deterministicici in più dimensioni emerge che ha senso utilizzare il metodo MC per effettuare integrali multidimensionali.

$$\int dx dy dz f(x, y, z) = \int d\bar{x} f(\bar{x})$$

Se facciamo l'algoritmo deterministico con  $N$  punti, posso prendere, ad esempio in  $d = 2$

$$\int_a^b dx \int_c^d dy f(x, y)$$

Prendo quest'ordine di integrazione, fissata una  $x$  posso calcolare con l'algoritmo deterministico

$$G(x) = \int_c^d dy f(x, y)$$

A sua volta poi integro  $G(x)$  in  $dx$

$$\int dx G(x)$$

Questo procedimento si chiama integrazione nidificata, cioè **nested integration**. Quando si sale con il numero di dimensioni l'algoritmo MC supera in efficienza gli algoritmi deterministicici, per il trapezio di parla di 4 dimensioni, mentre per Sympson si parla di 8 dimensioni. Tutto ciò a parità di  $N$ . Il MC ha altre applicazioni che con la prossima lezione si vanno a trattare.

# Chapter 14

## Lezione 13

### 14.1 Integrazione nidificata

Abbiamo visto che gli algoritmi deterministici in una dimensione scalano come  $\frac{1}{N^\alpha}$ , mentre il MC scala come  $\frac{1}{\sqrt{N}}$ . Se facciamo un integrale multidimensionale, lo possiamo fare sia con un MC che con un algoritmo deterministico. Il dominio di integrazione è  $V$ .

$$\int_{x_1}^{x_2} dx \int_{y_1}^{y_2} dy \int_{z_1}^{z_2} dz f(x, y, z)$$

Posso integrare a "fette", cioè, fissati  $\bar{x}, \bar{y}$ , posso computare l'integrale

$$\int_{z_1}^{z_2} dz f(\bar{x}, \bar{y}, z) = \int dz g(z)$$

come un integrale unidimensionale in  $z$ , che per  $\bar{x}$  posso chiamare  $G(y)$ . Una volta risolto l'integrale in  $z$ , posso considerare l'integrale in  $y$  come

$$\int_{y_1(x)}^{y_2(x)} dy G(y)$$

Ripeto il processo fino ad arrivare all'ultima integrazione. Facciamo un'integrazione nidificata.

#### 14.1.1 Confronto con la convergenza di MC

Se svolgiamo l'integrale con un algoritmo MC, abbiamo l'integrale nella forma

$$\frac{1}{N} \sum f(x_i)$$

Dove ogni punto è una  $n$ -upla  $n$ -dimensionale. Come possiamo confrontare gli algoritmi? Con gli stessi punti. Se ho  $N$  punti e  $d$  dimensioni, devo pensare l'errore

$$\frac{1}{N^\alpha}$$

Come generato da  $N$  punti distribuiti in  $d$  dimensioni, in modo tale che il prodotto dei punti distribuiti su ogni asse mi restituisca il conteggio complessivo, perciò  $N^{1/d}$ . Il mio errore generato da un algoritmo deterministico sarà pari a

$$N^{-\frac{\alpha}{d}}$$

Se voglio fare un paragone con il metodo MC devo confrontare l'esponente con  $-\frac{1}{2}$ . Fissato un  $\alpha = 4$  del Sympson, possiamo vedere per quale  $d$  MC supera in precisione l'algoritmo deterministico, cioè

$$-\frac{\alpha}{d} > -\frac{1}{2} \rightarrow \frac{\alpha}{d} < \frac{1}{2}$$

Nel caso di  $\alpha = 4$  ho che per  $d < 2\alpha = 8$  il metodo MC converge in modo più veloce. Per esempio, per  $d = 7$  dimensioni, conviene utilizzare Sympson. In alcuni problemi lo spazio dei parametri ha una dimensionalità molto superiore a 8, perciò conviene utilizzare il metodo MC in tali contesti.

## 14.2 Implementazione metodo MC

Implementiamo il problema dle calcolo di  $\pi$ , però cominciando a vedere questi aspetti diversi della simulazione MC. Da un lato ci calcoliamo l'integrale, banale, e dall'altra possiamo utilizzare lo stesso problema per fare la simulazione di una distribuzione di punti, oppure con il MC possiamo utilizzare la stessa distribuzione utilizzata per il calcolo dell'integrale per ricavarci delle quantità differenziali. Si noti che la struttura generale che si vede oggi, eccezion fatta per la simulazione utilizzata per il calcolo di  $\pi$ , è uguale alla struttura di algoritmi utilizzati per simulare collisioni di particelle ad LHC come in fisica nucleare o in astrofisica. La differenza sta nello spazio in cui è ambientato il tutto, il quale comprende molti più parametri teorici, o in teorie dei campi si può fare il calcolo di medie della QFT con simulazioni di questo tipo dove il numero di gradi di libertà aumenta notevolmente. La tecnica è la stessa di quella che si vede oggi. da ciò l'importanza di avere chiaro quanto si vede oggi.

### 14.2.1 Stima di $\pi$

Se prendiamo la circonferenza di lato 1, prendiamo il quadrato di lato 1 nel primo quadrante, i cui lati sono paralleli agli assi e il cui vertice in basso a sinistra coincide con l'origine, allora l'area di piano racchiusa dal quarto di cerchio che giace sul primo quadrante è uguale a  $\frac{\pi}{4}$ . Simulando quello che fa Buffon, tiriamo una coppia di punti  $(x, y)$  nel quadrato unitario, e questo viene automatico perché i numeri random sono estratti nell'intervallo unitario, a questo punto l'area è proporzionale al rapporto tra i punti che cadono nel cerchio e i punti in generale.

$$A = \text{efficienza} \cdot A_1$$

L'area è perciò l'efficienza, intesa come rapporto di punti interni al cerchio ed interni al quadrato per l'area unitaria del quadrato.

- Un metodo per capire se il punto generato è interno oppure è esterno sta nel vedere se  $x^2 + y^2 < 1$ . In questo caso se sono sotto il raggio, sono nella zona che contribuisce all'efficienza.
- Un altro modo sta nel vedere la curva  $y = \sqrt{1 - x^2}$ . L'area di questa zona è direttamente l'integrale tra  $(0, 1)$  di  $\sqrt{1 - x^2}$  e questo è direttamente uguale a  $\frac{\pi}{4}$ .

$$4 \int_0^1 dx \sqrt{1 - x^2} = \pi$$

Posso calcolare l'espressione dell'errore con la formula della varianza. Calcolo la deviazione della varianza e poi la deviazione standard sotto radice e ottengo la stima dell'errore.

- Il terzo modo, uguale al primo, visto in modo leggermente diverso, che si estende al caso multidimensionale con facilità, è dire che l'integrale lo posso fare generando una  $x$  uniformemente tra  $(0, 1)$ , poi genero un secondo random  $y$  tra  $0, 1$ ) e accetto la  $x$  in base a come è messa la  $y$  rispetto alla funzione. Accetto la  $x$  proporzionalmente alla funzione  $\sqrt{1 - x^2}$ . è generalizzabile in modo pratico se abbiamo la funzione di  $N$  punti in un dominio  $d$ -dimensionale. Possiamo calcolare l'integrale con il metodo **Hit or Miss**. In un ipercubo di dimensione  $d$ , estraggo un array di  $d$  random ed è un possibile argomento della funzione, calcolo al  $f(\bar{x})$ , estraggo un random tra **(min, max)** della funzione. Poniamo come min 0 e max 3/4. Se il punto è nella zona dove la funzione è più alta di 3/4, è più probabile che sia accettato, se ci si trova nella zona in cui la funzione è più bassa di 3/4 è più probabile che il punto non sia accettato. Quindi genero una distribuzione in  $x$  proporzionale al valore della funzione. In questo è cruciale conoscere il massimo della funzione. è importante non andare in una situazione dove genero punti con una coordinata fino a un massimo che non coincide con il massimo della funzione.

Posso fare ciò con funzioni generiche anche con dei picchi. Se non conosco la funzione analiticamente può diventare un problema identificare il massimo della funzione che mi permette di evitare di distorcere la mia distribuzione di eventi in  $x$ . Operativamente si fa un integrale preventivo con il metodo standard in cui genero la  $x$  e faccio il valor medio di tutte le  $x$  che sto utilizzando e prendo il valor medio più alto. La probabilità di

identificare il massimo è tanto maggiore quanti più punti possono essere esplorati nel dominio della fusione. Con  $f_{\max}$  che trovo posso far partire la generazione dei punti per trovare una distribuzione in  $x$  che rispecchia la distribuzione di punti che ho.

### Implementazione geometrica

Le routine si trovano nelle directory **integration/pi\_greco**. Ci sono le solite dichiarazioni di variabili. Abbiamo i parametri dedicati a ranlux. Dobbiamo specificare un numero di punti e il random seed. Successivamente abbiamo un contatore  $i_{\text{count}}$  ed  $n_{\text{count}}$ . Nella prima implementazione sto calcolando il  $\pi$  con il controllo che la coppia  $(x, y)$  sia nel cerchio di raggio unitario. Il random seed viene inizializzato a priori. Ad ogni punto generato incremento  $i$ . Estraggo la coppia di random float. Controllo che  $x^2 + y^2 < 1$ . Se la coppia è accettata, incremento  $n$ . Al termine della routine calcolo l'efficienza della generazione random di punti con

$$\pi \simeq p = 4 \frac{n_{\text{count}}}{i_{\text{count}}}$$

L'errore di tale stima è stato calcolato come

$$4 \sqrt{\frac{p(1-p)}{i_{\text{count}}}}$$

Il 4 deriva da  $\frac{\frac{\pi}{4}R^2}{R^2} = \frac{\pi}{4}$ .

### Implementazione integrale

Adesso provvedo a fare l'integrale di  $\sqrt{1 - x^2}$  con il metodo MC. La struttura generale del programma è la stessa. C'è la chiamata del random seed iniziale che ci fa ripartire dal punto iniziale. Per ripartire dal primo numero pseudo-random manteniamo il seed. Si fa il ciclo nello stesso modo di prima. Estraggo l'array di entrambi e successivamente ne uso solo uno. Calcolo la funzione come  $\sqrt{1 - x^2}$ . Sto facendo la somma di  $f(x_i)$ . Prima di fare la media si fa la sommatoria dei valori della funzione. Nella formula dell'errore serve la varianza e dunque il valor medio di  $f^2$ . Per cui, si calcola anche la somma dei valori quadrati della funzione. La novità è la call **distribution**. Calcolando questo integrale posso anche stimare quantità differenziali. Sto facendo l'integrale di  $f(x)dx$ . Posso andare a istogrammare la  $f(x)$  in bin di ampiezza  $\Delta x$ . Al netto sto semplicemente costruendo un istogramma. Questo viene utilizzato anche per valutare l'incremento medio della funzione  $\frac{\Delta f}{\Delta x}$ . Quando sto facendo la generazione dei miei numeri random posso andare a distribuire i punti sui miei istogrammi. Tale operazione viene compiuta dalla call **distributions**. Preventivamente ho definito il numero di bin e vado a vedere su ciascun bin come si distribuisce la frequenza dei valori, per questo motivo vado a chiamare la routine **distribuzioni** e per ogni  $x$  mi calcola in quale bin finisce quella  $x$  e mi calcola quanto vale la distribuzione in quel bin. Questo modo di fare l'integrale mi permette di avere una stima del valore massimo della funzione nel mio dominio. Questo perché confronto sempre il valore  $f_{\text{val}}$  con il valore inizializzato di  $f_{\max}$ . Se maggiore di esso, ridefinisco il valore massimo. Il valore d'innesto del massimo è 0, conoscendo a priori il segno della funzione. L'idea è che se metto un numero sufficientemente alto di punti, ho una buona stima del massimo, che mi può servire in seguito per runnare l'**Hit or Miss**. C'è una parte che riguarda il valor medio. C'è il dettaglio della varianza sul campione finito, definita come

$$\sqrt{\frac{\sum d_i^2}{N-1}}$$

Faccio il valor medio sul totale delle somme che ho accumulato diviso  $N$  e faccio il valore quadratico medio. L'errore è la varianza sotto radice.

$$\sqrt{\frac{\langle f^2 \rangle - \langle f \rangle^2}{N-1}}$$

Questo per calcolare la mia stima di  $\pi$ . Per calcolare l'istogrammazione della funzione inizializzo i vari bin e chiamo **distributions**, utilizzando lo script omonimo. Per ogni bin posso calcolare l'errore esattamente come l'ho calcolato per l'insieme di punti complessivo. Lo script scrive su un file **.txt** dove a run conclusa trovo la sezione d'urto per ogni bin.

```

enter maximum number of calls for MC integration
10000
enter random number seed
12345

pi estimate
n. of points=      10000     pi=    3.1663999557495117     +-   1.62465723868991
42E-002

results from MC integration
n. of points= 10000.000000000000           integral=  0.78996941151904487     +- 
2.2165001104957711E-003
pi greco=   3.1598776460761795     +-   8.8660004419830844E-003

real    0m18,345s
user    0m0,011s
sys     0m0,007s

```

Noi abbiamo la stima di  $\pi$  fatta alla Buffon, estraendo la coppia di punti e poi calcolando l'errore con il metodo binomiale. Di sotto ho il risultato integrale. Per  $10^4$  punti ottengo una stima con errore ai centesimi con il metodo geometrico, mentre con il metodo integrale ottengo un errore al millesimo. Il motivo sta nel numero di punti che in effetti contribuiscono alla stima di  $\pi$ . Nel secondo valore, se utilizzo  $10^4$  punti li sto utilizzando tutti, mentre nel primo scarto i punti che cadono fuori dalla circonferenza. Per tale motivo l'errore è leggermente più alto nel primo caso. Se facciamo con l'**Hit or Miss** otteniamo un andamento analogo, con la differenza che l'algoritmo è impostato per essere meglio generalizzabile. L'errore con HoM è leggermente più grande, anche se scala con il numero di chiamate. Se riprovo con  $10^6$  punti e con gli stessi valori di input, ottengo un errore che scala di un fattore circa 10.

```

pi estimate
n. of points=      1000000     pi=    3.1419000625610352     +-   1.6419696242997628E-003
results from MC integration
n. of points= 1000000.0000000000           integral=  0.78540100559671644     +-   2.2321992297
154929E-004
pi greco=   3.1416040223868658     +-   8.9287969188619717E-004

```

Lo scaling, essendo influenzato da fattori statistici non è esattamente di 10, eppure è rispettato. questo è l'utilizzo del MC. Per fare questo integrale, se avessimo utilizzato  $10^6$  punti con Sympson avremmo ottenuto un errore più basso di svariati ordini di grandezza. Chiaramente i vari metodi di stima devono essere in accordo statistico. Con il metodo HoM l'implementazione, anche se non nella forma di generazione di  $y$ , si scartano dei punti e l'informazione utilizzata è minore dell'informazione generata, perciò l'errore è leggermente più alto, anche se dello stesso ordine e con i risultati compatibili statisticamente. Abbiamo una serie di variabili che inizializziamo a zero, i contatori di chiamate, il valore della funzione e i valori medi. Inoltre il massimo della funzione che noi calcoliamo con l'integrazione ed utilizziamo on HoM. All'interno del ciclo mettiamo in memoria il massimo della funzione. Finito il ciclo MC dobbiamo calcolare i valori medi, dividendo per il numero di chiamate che abbiamo fatto. Calcoliamo il valor medio della  $f$  e di  $f^2$ . Con questi due elementi possiamo calcolare l'errore.

### Hit or Miss

In questo caso conosciamo già il massimo della funzione, cioè 1. Facciamo l'inizializzazione delle variabili. inizializziamo il numero random e  $n_{byas}$ . La struttura di analisi dei risultati è banale in questo caso e viene condotta all'interno dello script, ma per casi più complicati l'analisi viene eseguita in seguito all'estrapolazione dei punti. Ci permette di fare un esempio su un problema di QED e di fare un'applicazione all'Econofisica. Chiamiamo le variabili  $x$  ad argomento della nostra funzione. Nel nostro caso di problema unidimensionale, dobbiamo chiamare un unico numero. Successivamente chiamiamo l'altro random. Se ci si pensa è la stessa cosa che abbiamo fatto nella prima implementazione dell'ago di Buffon, in cui si estrae la coppia e si vede se cade sotto o sopra la circonferenza. Se il punto è accettato aggiorno  $n_{hit}$ , altrimenti non lo aggiorno. Vogliamo che il nostro valore massimo dal quale estraiamo i valori random non sia più piccolo del massimo della funzione. Se  $f(x)$  è maggiore del massimo, incremento un contatore di warning, che è  $n_b$ , poi calcolo l'integrale tramite l'efficienza. Comincio a calcolare l'efficienza con il rapporto tra  $n_h$  e punti totali, poi conteggio anche il

conteggio dei byas.

$$\text{eff} = \frac{n_h}{N}, \quad \text{eff}_b = \frac{n_b}{n_h}$$

Successivamente moltiplico l'efficienza per la misura del domino. In questo caso si tratta del prodotto identico banale, giacché il dominio è il quadrato di lato 1. L'errore lo calcolo come

$$\Delta = f_{\max} \sqrt{\frac{\text{eff}(1 - \text{eff})}{N}}$$

Scrivo il valore dell'integrale con la stima che abbiamo adesso moltiplicata per 4. Questo, di per sé è identico alla simulazione di tipo geometrico, però registriamo i punti in  $x$  che andiamo a estrarre. Da un lato studiamo le distribuzioni bin per bin e scriviamo su un file .txt tutte le  $x$  che abbiamo estratto e successivamente ne studieremo la distribuzione. Si fa negli esperimenti di LHC con i momenti delle particelle uscenti. Io ho generato un evento possibile al quale è associata una probabilità che calcolo con la teoria delle interazioni fondamentali e con quella probabilità vado a vedere come quella distribuzione di eventi è distribuita nello spazio delle fasi. Tutto ciò è riassunto nella distribuzione delle  $x$  ma la struttura concettuale è la medesima. Vengono generati eventi tramite MC vengono registrati su file e successivamente analizzati oppure vengono direttamente analizzate le distribuzioni differenziali sui bin sulle singole osservabili. Chiaramente in quel caso ho un numero di dimensioni molto maggiore di 1. Facciamo un esempio semplice sulla QED ma vediamo di approfondire l'argomento.

```
pi estimate
n. of points= 100000 pi= 3.1416800022125244 +- 5.1928477471884444E-003

results from MC integration
n. of points= 100000.00000000000 integral= 0.78587176799160707 +- 7.0513456215
263396E-004
pi greco= 3.1434870719664283 +- 2.8205382486105358E-003

results from MC integration
n. of points= 100000.00000000000 integral= 0.78542000055313110 +- 1.2982119915
261508E-003
eff= 0.78542000055313110 bias eff= 0.0000000000000000
pi greco= 3.1416800022125244 +- 5.1928479661046030E-003
```

Adesso abbiamo aggiunto l'ultimo metodo di calcolo dell'integrale HoM. Genera un numero compatibile con gli altri due metodi e in particolare identico al  $\pi$  della prima versione, poiché le due versioni sono identiche, semplicemente implementate in modo differente. Viceversa, non avremmo lo stesso risultato se ponessimo  $f_{\max} = \frac{1}{2}$ . Non dovremmo avere un risultato sbagliato, ma con un'efficienza minore. Si tradurrà in un errore maggiore.

```
pi estimate
n. of points= 1000000 pi= 3.1389119625091553 +- 1.6440436556410369E-003

results from MC integration
n. of points= 1000000.0000000000 integral= 0.78518119133865627 +- 2.2341573937
213752E-004
pi greco= 3.1407247653546251 +- 8.9366295748855007E-004

results from MC integration
n. of points= 1000000.0000000000 integral= 0.47816500067710876 +- 1.0217990245
647604E-004
eff= 0.95633000135421753 bias eff= 0.90555876493453979
pi greco= 1.9126600027084351 +- 4.0871960982590417E-004
```

Con il massimo della funzione modificato ho un'efficienza altissima, ciononostante il  $\pi$  viene stimato con un valore nettamente sbagliato. Uno strumento per capire quanto un risultato di una simulazione sia buono, senza basarci su di un benchmark analitico, può essere l'efficienza di byas. Difatti, in modelli dove stimo io stesso il massimo, non mi dice nessuno che il valore che ho ottenuto coincida con quello effettivo. Se ci sono casi in cui il massimo della funzione è molto piccato, è possibile che io debba aumentare di molto il numero di punti totali, se generati con distribuzione uniforme, per poterne approssimare il valore con sufficiente accuratezza. Se metto come  $f_{\max} = 10$  mi aspetto che il 90% dei punti non venga accettato. Provo con  $10^6$  punti.

```

pi estimate
n. of points= 1000000 pi= 3.1410601139068604 +- 1.6425534440166209E-003

results from MC integration
n. of points= 1000000.0000000000 integral= 0.78512438828313913 +- 2.2344646423
231080E-004
pi greco= 3.1404975531325565 +- 8.9378585692924319E-004

results from MC integration
n. of points= 1000000.0000000000 integral= 0.78682996332645416 +- 2.6924335167
420998E-003
eff= 7.8682996332645416E-002 bias eff= 0.0000000000000000
pi greco= 3.1473198533058167 +- 1.0769734066968399E-002

```

Si vede che il byas è 0 ma l'errore è cresciuto di un ordine di grandezza. Questo vuol dire che il numero di punti che hanno contribuito a determinare il valore di  $\pi$  si è radicalmente ridotto in percentuale circa del 90%. L'efficienza di conseguenza crolla al 10%, causando un aumento del rumore.

## 14.3 Algoritmi di riduzione della varianza

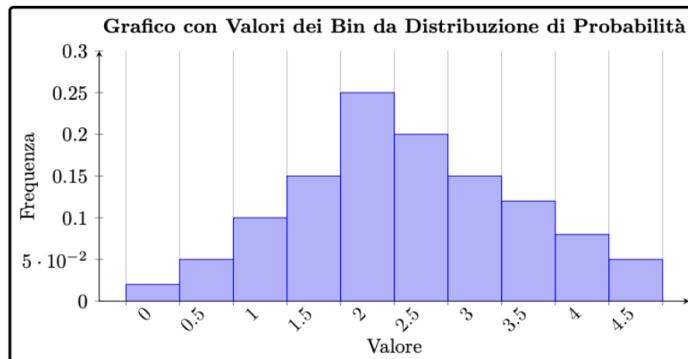
Se siamo interessati alla distribuzione delle  $x$  per le simulazioni MC della fisica delle collisioni di particelle, le  $x$  sono i momenti delle particelle. Vogliamo che siano distribuiti secondo quanto previsto dalla natura al meglio della descrizione che siamo in grado di dargli. Con le teorie a disposizione. Se facciamo la simulazione con un tetto massimo più basso stiamo dando una distribuzione finale dei 4-momenti completamente diversa da quanto previsto dalla nostra teoria. Il fatto che l'integrale sia sbagliato è strettamente connesso al fatto che non siamo in grado di generare distribuzioni di punti che rispecchia la teoria da cui siam partiti. Sono due facce della stessa medaglia. Questo introduce il problema di come ridurre le possibili difficoltà. Questo introduce gli **algoritmi di riduzione della varianza**. Si suddivide il dominio in più regioni. La mia funzione non ha queste variazioni molto grandi di valori e per ciascuna di esse possiamo utilizzare un  $f_{\max}^i$  molto diverso. La suddivisione del dominio in zone con varianze minimizzate si chiama metodo dello **stratified sample**. Si prende anche il **campionamento di importanza**, che implementiamo con gli esercizi di QED che ci permette di vedere il problema su un esempio pratico. In qualche modo sto dicendo che il metodo con cui sto producendo la mia distribuzione di numeri random non è sufficiente, perché nelle zone più importanti non genererà probabilmente sufficienti punti. Devo fare in modo di aumentare la densità di punti in quelle zone. Il byas è un aspetto che affligge tutte le simulazioni MC e bisogna stare attenti a questa cosa.

# Chapter 15

## Lezione 15

### 15.1 Metodo HoM

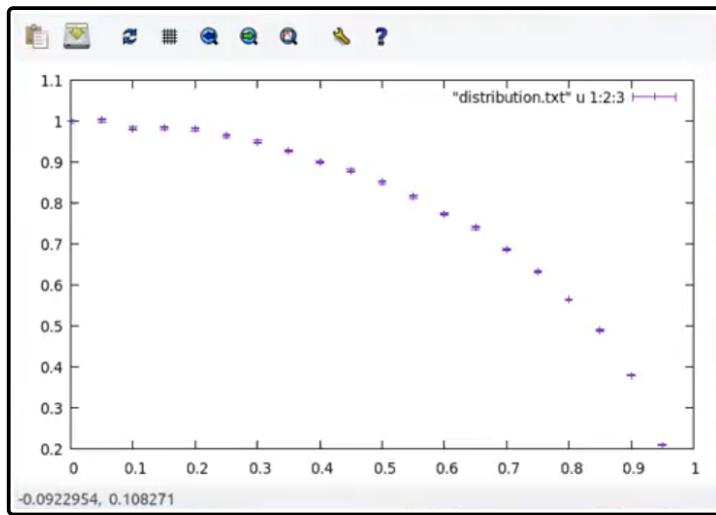
Quando genero una distribuzione di punti nel calcolare l'integrale, assegno i punti nei bin. Successivamente quando genero una distribuzione di punti con il metodo HoM e li metto nei bin posso integrare contando i punti nei bin. La routine di instogrammazione ci dà il valor medio del bin diviso la sua ampiezza. Questa è la derivata media nel bin. Sulla parte HoM si può fare la stessa cosa. Registriamo oltre alla distribuzione anche la parte umb, significando che la distribuzione registra i punti accettati. Nell'HoM ci si ricorda che abbiamo i punti di tentativo e la  $x$  viene accettata se la  $y$ , generata tra il minimo e massimo della funzione, è inferiore a  $f(x)$ . Facciamo la media degli  $f(x_i)$  che cadono nei bin.



L'informazione sulla  $y$  l'abbiamo utilizzata per accettare o meno il punto, perciò i punti nell'istogramma del metodo dell'integrale hanno peso  $f(x)$ , mentre nel metodo HoM registro punti senza peso. Si parla infatti di simulazioni **weighted** e **unweighted**. Pesata vuol dire che utilizzo tutti i punti che sto generando nello spazio delle fasi delle variabili indipendenti e la probabilità con cui calcolo l'ampiezza di scattering, moltiplicata per il fattore di spazio fasi va a definire la mia  $f$ . Questa la utilizzo per calcolarmi il mio integrale nel mio bin. Sotto la funzione voglio dare una distribuzione di punti nello spazio delle variabili indipendenti che rispecchi la distribuzione di probabilità proporzionale alla funzione di riferimento. Quindi, nel fare HoM l'informazione sulla  $f$  viene utilizzata per accettare o meno il punto, poi i punti hanno tutti peso 1. I punti dove la funzione è più alta sono più densi e dove la funzione è più bassa sono più diradati. La call **distributions\_unw** concettualmente mi dà la distribuzione  $\frac{df}{dx}$  discretizzata che avevo con l'altra metodologia.

#### 15.1.1 Derivata della $f$ sui bin

Un'altra modo che potrei utilizzare consiste nel salvare i dati su un file e poi andare ad analizzarli successivamente. Nel run fatto in precedenza, se guardiamo i punti in **distribution.txt**, ho 20 bin e all'interno c'è il calcolo della  $d\langle f \rangle / dx$ . Per ogni punto c'è il valore dell'istogramma con il suo errore. Possiamo visualizzarlo con GNUPLOT.



Abbiamo i 20 punti corrispondenti e ogni fluttuazione corrisponde al suo errore. Più si aumentano i punti, più si possono incrementare i bin. Per ogni bin il programma somma tutti i valori della funzione  $\sum \frac{d\langle f \rangle}{dx_i} \Delta x_i$ . Questo mi dovrebbe dare una stima dell'integrale con quei punti. Si stima in questo modo  $\frac{\pi}{4}$ .

### 15.1.2 Distribuzione punti HoM

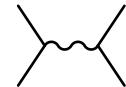
Invece se apro il file **points.dat** trovo i  $\sim 78439$  punti che sono stati scelti con la selezione di HoM dai  $10^5$  punti iniziali. Le  $x$  non sono ordinate, sono elencate nell'ordine in cui il random corrispondente è stato generato dall'algoritmo. Questi punti posso andare a leggerli con il programma **readpoints.f** e calcolarmi le distribuzioni che preferisco. Il ciclo viene fatto per un arbitrario numero massimo di punti nell'unità 11. Finisce di leggere il file e chiude la presa dati. Per ognuno di questi punti legge il valore in  $x$  e il numero della lista. Se noi avessimo questo procedimento su un problema di 20 dimensioni, avrei una lista di array di 20 valori. Facciamo una generazione di punti che mi prende molto tempo macchina e voglio, in base a tale generazione, ricavarmi tutte le proprietà differenziali sulle variabili. Se noi scegliessimo di impiegare gli eventi pesati, nel modo in cui si accettano tutte le  $x$ , significherebbe che dovrei mettere nelle distribuzioni la distribuzione in più e con 20 dimensioni ne avrei tantissime di possibili distribuzioni. Prendere i punti e rifarli girare potrebbe prendere più giorni. Il vantaggio che ci dà scrivere gli eventi su disco e che una volta segnati posso utilizzarli nel modo che voglio mi aiuta nel non dover poi rigenerare punti. A seconda della  $x$  che trovo, la metto in un bin e vado a cumulare le  $f(x)$  che finiscono in quel **bin** e poi calcolo la routine di instogrammazione. La varianza la calcolo bin per bin e in base a tale valore plotto l'errore sulle  $y$ . Posso anche svolgere questa routine in modo **unweighted**, dove non calcolo l'integrale della  $f$ , vado a mettere in ogni bin la frequenza con la quale cadono i punti nel bin. La frequenza mi dà l'efficienza e poi calcolo per l'area di riferimento per avere l'integrale in quel bin. Per ricavare l'errore utilizzo la formula dell'errore del conteggio di eventi, su ciascun bin. L'errore è proporzionale alla radice di  $\text{eff}(1 - \text{eff})$ .

### 15.1.3 Ipercubo di 20 dimensioni

Producendo i miei numeri random in 20 dimensioni, poiché devo studiare il problema in uno spazio di parametri più esteso. Nella routine di distribuzione ci mettiamo gli istogrammi che ci vengono in mente, che sono tanti. Posso innanzitutto graficare le variabili indipendenti e le funzioni di variabili. Lo spettro delle possibilità aumenta considerevolmente. Se non ho messo tutte le informazioni nel mio programma quando avvio la simulazione, devo aggiornare il file delle distribuzioni e rifarmi la simulazione da 0, per avere le distribuzioni che interessano di più. Al CERN per generare un punto nel simulatore di eventi ci si impiegheranno  $10^{-3}$  s. Questo vuol dire che se voglio simulare il mio evento con  $10^{11}$  punti, impiego  $10^8$  secondi, se non parallelizzo la simulazione. Questo vuol dire che impiegherei  $\sim 3$  anni per la simulazione di un evento. Se volessi simulare eventi di teorie moderne, potrei anche incrementare lo spazio dei parametri necessari. A questo punto, tipicamente non è chiaro quale osservabile differenziale sia la più indicata per avere informazioni cruciali sul mio modello, perciò, da quel punto di vista si comincia a studiare per gradi il problema e si vede dove mettere il focus. Perciò si introducono

delle altre osservabili, per capire se le idee che ci si è fatti sul problema sono corrette. A questo punto, se si fa tutto nel modo in cui si utilizzano tutti gli eventi per fare la distribuzione, metto altre  $n$  routine differenziali nell'istogrammazione, però poi per avere dei risultati devo ricominciare la simulazione dall'inizio. Può portare via  $10^3$  ore o di più. Il problema diventa pressante, ovviamente sono cose che vengono fatte su farm di server, dove si utilizza la parallelizzazione di computer, dove il problema di fondo rimane. L'altra opzione di procedere con la registrazione di eventi **unweighted** permette di fare un'unica simulazione di eventi completa, anche se necessita di molto tempo e poi posso andare a caratterizzare il mio problema in tutte le dimensioni che voglio, andando a studiare la distribuzione dei punti che ho già su disco, senza dover ripetere la simulazione. Prima faccio la simulazione di eventi al meglio delle mie conoscenze e poi registro tutto quello che riesco a registrare e poi lavoro sul file che contiene i punti, analizzandoli con lo script **readpoints.dat**. L'idea è quella di fare la generazione e poi l'analisi delle quantità differenziali in due step successivi.

## 15.2 Esercizio di QED



Supponiamo di voler simulare una collisione tra un elettrone e un positrone, . In Italia vive la tradizione storica degli acceleratori di particelle  $e^+$ ,  $e^-$ . Dunque studiamo la collisione che produce una coppia di muone e anti-muone  $\mu^-$ ,  $\mu^+$ . La sezione d'urto è proporzionale alla probabilità per unità di tempo e di flusso incidente di particelle entranti che si produca la coppia di particelle  $\mu^+\mu^-$ . È importante studiare anche la sezione d'urto differenziale  $\frac{d\sigma}{d\cos\theta d\varphi}$ , dove  $(\theta, \varphi)$  sono gli angoli con cui esce il muone. La sezione d'urto differenziale è la probabilità di trovare il muone in una regione compresa tra  $(\theta, \theta + d\theta)$  e  $(\varphi, \varphi + d\varphi)$ . La QED ci fornisce le regole per poter calcolare la sezione d'urto e l'ampiezza di probabilità di transizione tra la coppia iniziale e la coppia finale.

$$\sigma = \frac{1}{J} \int \frac{d^3 q^-}{(2\pi)^3 2E^-} \frac{d^3 q^+}{(2\pi)^3 2E^+} (2\pi)^4 \delta^{(4)}(p_1 + p_2 - q^- - q^+) \frac{1}{4} \sum_{\text{spin}} |M|^2$$

Dove  $M$  è l'elemento della matrice  $S$  che detta l'ampiezza di transizione con il modulo al quadrato, che si prende come fornito.  $M$  dipende da tutti i momenti delle particelle, che poi è sommata su tutti gli spin iniziali delle particelle. Il problema è vedere come calcolare questa quantità. È un processo principe della QED. Simulare numericamente la  $\delta^{(4)}$  non è bello, inoltre devo fare del lavoro analitico, perché se guardiamo lo spazio delle fasi,  $d^3 q^-$  vuol dire le 3 componenti del momento del  $\mu^-$ , come quantità indipendenti, alle quali devo aggiungere le 3 del  $\mu^+$ . Questo significa che in totale ho 6 quantità indipendenti ma la  $\delta^{(4)}$  mi dà 4 vincoli, perciò in totale ho 2 variabili indipendenti, che saranno  $\theta, \varphi$ , fissata l'energia iniziale delle particelle. Quello che si fa normalmente è che per l'integrale in  $\delta^{(4)}$ , prima si fa l'integrale nella  $\delta^{(3)}$  in  $d^3 q^+$ , si lascia poi l'integrale in  $d^3 q^-$  con la delta delle energie  $\delta^{(0)}$ , avendo già risolto la  $\delta^{(3)}$ . Se facevo l'integrale in  $(E^-, \theta, \varphi)$ , faccio l'integrale in  $dE^-$  facendo la  $\delta^{(0)}$  e quando faccio questo viene fuori una  $f(E^-)$ , poiché le relazioni che comprendono l'energia sono non lineari,  $E^2 = p^2 + m^2$ .

### 15.2.1 Lavorio Analitico

Se la relazione è non lineare, facendo l'integrale con la  $\delta^{(3)}$  in  $d^3 q^+$  mi fissa  $q^+ = p_1 + p_2 - q^-$ . Tolgo dunque l'integrale in  $d^3 q^+$  e la  $\delta^{(3)}$ , mentre l'altro è più complicato poiché ho una funzione non più lineare, che mi tira fuori uno Jacobiano non banale che devo calcolare.

$$\begin{aligned} & \delta^{(3)}(\bar{p}_1 + \bar{p}_2 - \bar{q}^- - \bar{q}^+) \\ & \delta^{(0)}(p_1^0 + p_2^0 - q^{-0} - q^{+0}) \end{aligned}$$

Per la parte spaziale considerare la  $\delta^{(3)}$  consiste in

$$\begin{aligned} & \int d^3 q^+ \delta^{(3)}(\bar{p}_1 + \bar{p}_2 - \bar{q}^- - \bar{q}^+) f(q) = f(q^+|_{\text{soluz}}) \\ & \bar{q}_+ = \bar{p}_1 + \bar{p}_2 - \bar{q}_- \\ & d^3 q^- = |q^-| d|q^-| d\cos\theta^- d\varphi^- \end{aligned}$$

Con la  $\delta$  posso fare l'integrale in  $d|q^-|$ , ricordandomi che

$$\int dx g(x)\delta(f(x)) = \int dx \frac{g(x)}{|f'(x)|_{x=x_0}}\delta(x-x_0) = \frac{g(x_0)}{|f'(x)|_{x=x_0}}$$

Per i motivi sopra elencati c'è da fare del lavoro analitico sulla formula, al termine avrò lo Jacobiano nello spazio delle fasi per l'elemento di matrice.

### 15.2.2 Script

La sezione d'urto è una quantità Lorentz invariante, si può calcolare in qualunque sistema di riferimento. Si può calcolare nel riferimento in cui  $e^-e^+$  hanno lo stesso momento e in particolare la componente  $z$  del momento uguale e opposta. Sono nel centro di massa. Una caratteristica dei collider  $e^+e^-$  è che si ha il centro di massa della collisione coincidente con il laboratorio.  $\alpha$  è la costante di accoppiamento  $\sim 137^{-1}$ . Il fattore di conversione è la conversione dalle unità naturali alle unità del sistema internazionale SI. I calcoli della QED si fanno tipicamente nel sistema di unità naturali  $\hbar = c = 1$ . Per esempio, spazio ed energia hanno dimensione inversa. La sezione d'urto ha dimensione  $E^{-2}$ . Poiché è una superficie, devo passare ai  $cm^2$ , o meglio, ai suoi sottomultipli, cioè **barn** =  $(10^{-24} cm^2)$ . Si utilizzano tipicamente i sottomultipli del barn, come il **picobarn**. Perciò il programma implementa automaticamente la conversione da  $GeV^{-2}$  a picobarn ( $3.8937 \cdot 10^8$ ). Si fa un ciclo con gli eventi weighted e un ciclo con gli eventi unweighted. Gli stessi step del calcolo del  $\pi$  sono la struttura generale di qualsiasi programma di simulazione. Inizializzo le quantità a 0, poi inizializzo il numero random con la chiamata **RLUXGO**. Simulo le variabili  $\theta, \varphi$  con 2 numeri random  $\xi_{1,2}$ , mentre la variabile dipendente la simulo con  $\xi_3$  nell'Hit or Miss. L'intervallo di riferimento però non è  $(0, 1)$ , in quanto noi simuliamo direttamente  $\cos \theta$ , che varia da  $(-1, 1)$ , come  $\varphi$  che varia in  $(0, 2\pi)$ .

$$\int_{-1}^{+1} d \cos \theta$$

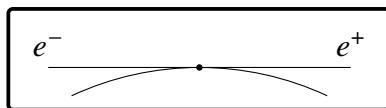
Dato  $\xi_1 \in (0, 1)$ , devo mappare il  $\cos \theta$  nell'intervallo  $(-1, 1)$ . Quindi mappo  $\cos \theta = -1 + 2\xi_1$ , e  $\varphi = 0 + 2\pi\xi_2$ . Da queste mappe estraggo gli Jacobiani.

$$\int_0^1 d\xi_1 \int_0^1 d\xi_2 4\pi$$

Inizializzo le variabili random e ne calcolo seno e coseno. Si inizializza l'energia del fascio con **ebeam**, con la quale calcolo  $s$  e sua radice. Si introducono le variabili mettendo la radiazione. Poiché quando interagiscono  $e^+e^-$  rallentano, dunque emettono radiazione vicino al punto di collisione. È un effetto importante che distorce di molto la descrizione dell'esperimento, per cui è importante tenerne conto. In questa descrizione non c'è radiazione ed è ciò che ci permetterà di eseguire il campionamento ad importanza, che ci permetterà di eseguire predizioni.  $\sqrt{s}$  è la radice a disposizione nel centro di massa, dunque costruisco le componenti del 4-momento di  $e^+e^-$ .

### Approssimazione Ultrarelativistica

Sto trascurando la massa dell'elettrone, sono in approssimazione ultra-relativistica, dunque  $E = |p|$ . Si pone l'asse  $z$  lungo la traiettoria del fascio del mio collider. Il collider è un anello e la collisione avviene in un punto che localmente si può approssimare ad una tratta rettilinea.



L'elettrone avrà 4-momento pari a  $(E, 0, 0, E)$  e il positrone avrà 4-momento pari a  $(E, 0, 0, -E)$ . La relazione tra energia e impulso di una particella è  $E = \sqrt{|p|^2 + m^2}$ . Se ci poniamo a  $E = 10 GeV$ , la rest-mass dell'elettrone è pari a  $m_e = .5 MeV$ . La massa rispetto all'energia è di  $5 \cdot 10^4$  volte più piccola. Dunque in tale approssimazione  $E = |p|$ . Abbiamo definito i momenti e c'è la chiamata **DPH12** che, data l'energia, la massa del muone, il coseno e il  $\varphi$  del muone che ho generato prima come variabili indipendenti, mi va a calcolare il  $Q_1 = q^+$  e il  $Q_2 = q^-$ . **F12** è l'integrale legato alla quantità integrale citata in precedenza. Lo Jacobiano totale è il prodotto degli Jacobiani delle variabili dipendenti e poi calcolo l'ampiezza di scattering con la chiamata

**matrixelement.** Ci restituisce numericamente il valore dell'elemento di matrice. Con tutti i conti in precedenza riesco a calcolare la sezione d'urto. La cosa importante è che registro, come fatto in  $\pi$ , l'  $f_{\max}$  e  $NHITWMAX$  è il numero di punti che chiedo di simulare nella simulazione weighted.

### 15.2.3 HoM

Con tutti i passi precedenti che coincidono, genero un ulteriore random number. Avendo registrato in precedenza il valore massimo della funzione, ho un'espressione dell'elemento di matrice leggermente più complicato. Si può ancora ricavare il valore massimo analitico ma si calcola facilmente in modo numerico. Il numero random ulteriore si chiama **rnumber**. Se il random è più grande della funzione nel punto non si accetta il punto, altrimenti lo si accetta, aggiornando il contatore di punti accettati. Qui c'è il salvataggio della cinematica su file. Finito il ciclo si calcola la sezione d'urto globale con l'informazione globale, ma abbiamo gli elementi salvati su disco che possiamo analizzare. La struttura è uguale a quella della simulazione per stimare  $\pi$ . Lo spazio dei parametri certo è più grand, da 2 a 3, e aggiungendo la radiazione diventano persino 4. Diventa interessante vedere che la distribuzione dell'angolo del muone non sarà più uniforme, sarà la distribuzione dell'angolo prevista dalla QED, quindi una determinata distribuzione non banale. Considerando la radiazione vediamo le caratteristiche della radiazione emessa nella collisione di particelle e possiamo andare a studiare l'energia emessa dalle particelle sotto forma di fotoni e andare a vedere come si distribuisce su tutti gli eventi.

# Chapter 16

## Lezione 16

### 16.1 Simulazione di collisione

Simulo la collisione di due fasci di elettroni e positroni che si incontrano nel centro di massa in una posizione in cui sono in direzione  $z$  e verso uguale e contrario. Nel c.m. l'energia dei muoni back to back è deltiforme  $E = \frac{\sqrt{s}}{2}$  meno effetti di massa. L'Hit or Miss proviene dall'estrazione di un terzo random che simula il valore dell'integrale, compreso tra 0 e il massimo dell'integrale. Una volta calcolato l'evento si scrive la cinematica su file. Ovviamente gli eventi sono semplici, registriamo le energie del fascio, le variabili di simulazione e il quadrimomento di ogni particella. Si utilizza la convenzione  $(t, x, y, z)$ . Tutto ciò si scrive nella prima unità del file. Il formato  $(1, 2197 \dots 2199)$  è una particolare formattazione del numero. Man mano che si procede nella simulazione si salvano gli elementi su disco e possiamo calcolarci la sezione d'urto integrata su tutto lo spazio delle fasi, anche se non possiamo da ciò ricavare direttamente quantità differenziali. Quando LHC sta prendendo dati i programmi di simulazione hanno un formato di questo tipo più complicato per simulare gli eventi. Si è anche scritta la componente  $z$  del quadrimomento per vedere che i coseni sono opposti tra loro.

#### 16.1.1 Run dello script

Forniamo come seed dello pseudo-number generator 312121, come energia  $10GeV$ . Il numero di eventi weighted si imposta come  $10^6$ . Per gli eventi unweighted scegliamo  $10^5$ .

```
ENTER SEED FOR RANDOM NUMBER GENERATION
312121
BEAM ENERGY (GeV)
10.d0
ENTER MAXIMUM NUMBER OF CALLS (for weighted evts)
1000000
ENTER MAXIMUM NUMBER OF CALLS (for unweighted evts)
100000
ICNT =      1000001

XSECT (weighted evts)=    108.62111565284431      +-   2.4299998918702020E-002  PB

results from MC unweighted integration
n. of points=      149987      integral=      108.61440435228469      +-   0.198281698
20473868
eff=    0.66673111915588379      bias eff=    0.0000000000000000

real     1m45,183s
user     0m4,308s
sys     0m0,222s
```

Abbiamo le sezioni d'urto integrate totali,  $108pb$  calcolata con il primo metodo e  $108pb$  calcolata con HoM. Non ci sono byas e i due numeri sono perfettamente compatibili entro l'errore.

#### 16.1.2 Output del programma

Se andiamo a vedere cosa è stato generato dal programma abbiamo il file **fort.1**.

3
10.00000
1.00000 1.00000
10.00000 0.00000 0.00000 10.00000
10.00000 0.00000 0.00000 -10.00000
10.00000 -1.23945 -2.05932 -9.70627
10.00000 1.23945 2.05932 9.70627

Abbiamo scritti gli eventi con energia, variabili di simulazione e componenti dei quadrimomenti. Studiamo la cinematica ultrarelativistica, trascurando effetti di massa. Per il positrone abbiamo componente  $z$  opposta all'elettrone. Per i muoni abbiamo componenti  $x$  e  $y$  diversi da 0. Evento per evento abbiamo diverse componenti  $(x, y, z)$ .

### 16.1.3 Analisi dell'output

Con questo file di eventi che contiene 100000 eventi possiamo fare un'analisi di variabili differenziali come vogliamo, utilizzando la routine distributions adattata per questo file. Non sapendo quanti eventi ci sono nel file, fissiamo la routine ad un massimo di  $10^9$  acquisizioni di eventi. La routine procede finché non trova il messaggio di fine file. Avendo letto le coordinate di ciascun evento, le ho caricate sui vettori  $p_1, p_2, q_1, q_2$ , vanno in input alla routine che calcola le distribuzioni. Fissiamo il peso di ogni evento ad 1 dall'inizio. Decidiamo variabili per la posizione dell'evento nello spazio delle fasi prevista dalla QED. L'angolo del muone, l'energia del muone, l'angolo del positrone e l'energia del positrone, compresa la massa invariante della coppia  $\mu_+\mu_-$ . Le energie sono interessanti se accendiamo la radiazione, se invece non la accendiamo sono fissate. Carichiamo il file riga per riga e ci sono degli indici, come definizioni, non di interesse al dettaglio, e si pongono gli estremi per ogni variabile. Il coseno dell'angolo del muone va  $(-1 - \varepsilon, 1 + \varepsilon)$ . Nell'istogramma si parte con l'estremo sinistro, perciò si vuole evitare che il punto giaccia sull'asse delle  $y$  per ragioni di visualizzazione. Per la massa invariante si prende la somma dei quadrimomenti e se ne calcola la componente temporale al quadrato meno le componenti spaziali al quadrato, come in una metrica  $\eta = (1, -1, -1, -1)$ , perciò difatti invariante di Lorentz. L'energia va da 0 a tutta l'energia del fascio. *fun* è la variabile che viene istogrammata. *pe* è il modulo del trimomento dell'elettrone. Si fissa un numero di distribuzioni che si preferisce e si fa girare il programma su un set di eventi.

### 16.1.4 File output dell'analisi

Runnare il file di analisi genera dei file

- **mu+mu-\_invariant-mass.txt**
- **mu-\_energy.txt**
- **mu+\_energy.txt**
- **mu-\_angle.txt**
- **mu+\_angle.txt**

i quali nomi sono fissati nelle variabili **distname** ( $i = 1, \dots, 5$ ).

#### Massa Invariante

Visualizziamo il file sulla massa invariante e poiché utilizziamo 20 bin, contiene 20 righe. Poiché operiamo tra  $\sim -10^{-6}$  per essere a sinistra del minimo, e  $10\text{GeV}$ , lo script ha suddiviso il range in 20 intervalli di ampiezza  $\sim .5\text{GeV}$ . Per ogni bin abbiamo la  $y$  e l'errore corrispondente, e sono tutti 0 tranne che nell'ultimo bin, poiché l'energia del muone è monocromatica, quindi  $E = \frac{\sqrt{s}}{2}$  e tutti gli eventi finiscono nell'ultimo bin. Con efficienza 100% abbiamo errore 0.

$$M(\mu^-\mu^-) = \sqrt{\eta(P_{\mu^-} + P_{\mu^+})} = \sqrt{(E_- + E_+)^2 - |\vec{p}_- + \vec{p}_+|^2}$$

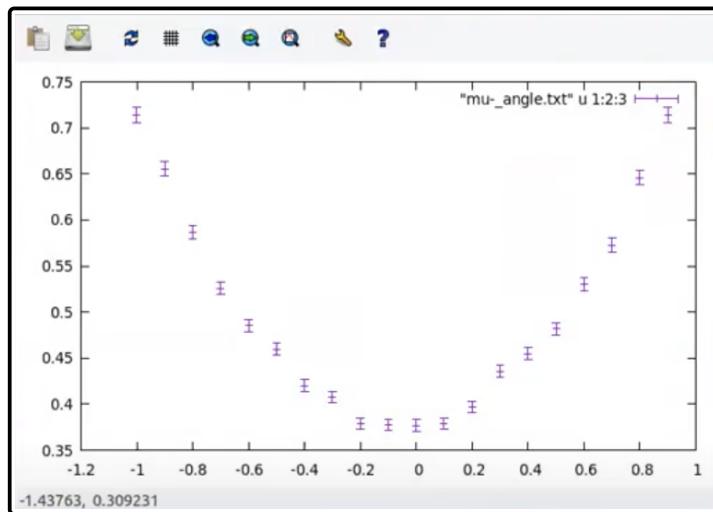
Siccome siamo nel centro di massa, la somma dei 3-momenti è 0, dunque

$$M(\mu^-\mu^-) = \sqrt{(E_- + E_+)^2} = E_+ + E_- = \sqrt{s}$$

che è la ragione per la quale si popola esclusivamente l'ultimo bin.

### Coseno dell'angolo

Il file sull'angolo non ha tutti i bin uguali a 0 tranne uno. È ripartito equamente tra tutti i bin. In particolare, dalla QED ci aspettiamo che, conservandosi la parità ci aspettiamo un andamento  $1 + \cos^2 \theta$ . In prossimità di  $\pm 1$  abbiamo una maggiore popolazione, infatti i bin corrispondenti sono più popolati, mentre al centro sono meno popolati. Se plotto i dati del file, utilizzando le 3 colonne con l'errore sulla  $y$ , emerge



una curva simmetrica rispetto all'origine, sebbene shiftata leggermente verso sinistra per effetto del bin-size. Il grafico rappresenta la  $\frac{d\sigma}{d \cos \theta}$  secondo la QED. Si fanno le analisi in questa modalità nella fisica nucleare, delle particelle, non solo per le sezioni d'urto totali, che hanno meno rilevanza ma le sezioni d'urto differenziali sono più importanti.

## 16.2 Bremsstrahlung

Quando un elettrone e un positrone annichilano sono sottoposti ad un'accelerazione localmente molto intensa, che provoca generazione di radiazione. La simulazione dello script si ferma al tree-level della QED, con le formule implementate negli elementi di matrice. Eppure un grosso effetto è l'emissione di radiazione. Se estendo la trattazione alla radiazione devo trattare, seppur brevemente, l'argomento. A livello dominante nella descrizione, si può scrivere l'emissione della radiazione in approssimazione semi-classica

$$\sigma(e^+e^- \rightarrow \mu^+\mu^-)$$

L'elettrone e il positrone che emettono fotoni, cedono una parte del quadrimomento. Qui entrano in gioco le variabili  $(x_1, x_2)$  che sono le frazioni corrispondenti di quadrimomento che hanno elettrone e il positrone dopo l'irraggiamento. Si può separare la bremsstrahlung dal processo di hard scattering e questo viene descritto come un doppio integrale sulle frazioni di quadrimomento restante delle funzioni di struttura.

$$\sigma(e^+e^- \rightarrow \mu^+\mu^-) = \int_0^1 dx_1 \int_0^1 dx_2 D(x_1, s) D(x_2, s) \int \frac{d\sigma}{d\Omega}(e^+e^- \rightarrow \mu^+\mu^-)$$

Le funzioni di struttura descrivono la densità di probabilità di trovare nell'elettrone del fascio dopo la variazione un elettrone con una frazione  $x_1$  del quadrimomento ad una certa scala di energia che corrisponde a  $\sqrt{s}$ . La stessa cosa per il positrone. Infine si moltiplica per il processo di kernel, dove  $d\Omega = d\cos \theta d\varphi$ . Eppure la  $\frac{d\sigma}{d\Omega}$  si riferisce non più al processo  $p_- + p_+ \rightarrow q_- + q_+$ , ma al processo  $x_1 p_- + x_2 p_+ \rightarrow q_- + q_+$ , poiché ciascuno ha perso energia nella radiazione.

### 16.2.1 James Bond

Le funzioni di struttura sono nella forma

$$D(x, s) = \frac{\beta}{2}(1-x)^{\frac{\beta}{2}-1} + \beta x, \quad \beta = \frac{2\alpha}{\pi} \left( \ln \frac{s}{m_e^2} - 1 \right)$$

Il nome della variabile  $\beta$  sta come **Bond, James Bond**, perché è stato introdotto da Bruno Cusheck a Frascati, dove sono nati gli acceleratori di particelle  $e^+e^-$ . Si è calcolato il fattore  $\beta$  alla macchina ADA (Anello di Accumulazione), ed è risultato  $\beta = 0.07$ . Dal nostro punto di vista dobbiamo fare la simulazione realistica. Abbiamo adesso un integrale 4-dimensionale. Nel kernel abbiamo un integrale in  $\cos \theta$  e in  $\phi$ . Poi dobbiamo integrare in  $dx_1$  e in  $dx_2$ .

### 16.2.2 Emissione asimmetrica

C'è un'altra complicazione per registrare gli eventi, poiché li registriamo nel laboratorio. Se noi abbiamo la radiazione, non è che la radiazione tra elettrone e positrone sia evento per evento simmetrica, ovvero  $x_1 \geq x_2$ . Dobbiamo capire cosa succede. Nel laboratorio si vede che un elettrone di momento  $x_i p_-$  collide con un positrone di momento  $x_2 p_+$ . Eppure, se  $x_1 \geq x_2$ , il CM non coincide più con il centro di massa del laboratorio. In generale

$$x_1 p_z^{e^-} + x_2 p_z^{e^+} \neq 0$$

Poiché ho sviluppato le formule del kernel con momenti simmetrici, e la sezione d'urto differenziale è in angolo e non Lorentz invariante, devo stare attento. Quello che è Lorentz invariante è al sezione d'urto, integrata sull'angolo stesso. Perciò posso sfruttare la simmetria di Lorentz invarianza della sezione d'urto, generando un processo di hard-scattering nel suo CM. Poi però, evento per evento, tenendo conto che conosco quali  $x_1$  e  $x_2$  sono presenti, spostarlo nel riferimento del laboratorio con una trasformazione di Lorentz. Il programma che descrive la radiazione, oltre che avere la doppia integrazione e le funzioni di struttura ha il blocco relativo alla trasformazione di Lorentz. C'è un piccolo dettaglio, bisogna estendere a 4 dimensioni l'integrazione, come eseguito nello script **ffbar\_rad**. Facendo girare lo script si vede che le frazioni di momento hanno uno spettro di valori, così come la massa invariante dei muoni, che non sarà più piccata su  $E$ , ma avrà una sua distribuzione, data dalla QED. A questo si aggiunge un'ulteriore complicazione. Guardiamo le densità di probabilità,

$$D(x, s) = \frac{\beta}{2}(1-x)^{\frac{\beta}{2}-1} + \beta x, \quad \beta = \frac{2\alpha}{\pi} \left( \ln \frac{s}{m_e^2} - 1 \right)$$

Noi dobbiamo integrare questa quantità tra  $(0, 1)$ . Dove  $-1 < \alpha < 0$ . Questo vuol dire che abbiamo un termine  $\frac{1}{1-x}$  elevato ad un esponente positivo. Vuol dire che questa funzione si comporta come  $\frac{1}{1-x}$ , e per  $x = 1$  diverge. Come integriamo questa cosa, estraendo uniformemente un random?

### 16.2.3 Simulazione numerica

Adesso generiamo 5 numeri random, 4 variabili di input e una quinta per l'Hit or Miss. I primi due sono direttamente quello che esce dal generatore, visto che sono compresi tra  $[0, 1]$ . Se noi calcoliamo  $S(e^+e^- \rightarrow \mu^+\mu^-)$  di Mandelstamm, ricordiamo la sua definizione  $S = \eta(p_1 + p_2, p_1 + p_2)$ . Per definire il fattore  $\hat{S}$  nell'hard-scattering con radiazione, devo includere le frazioni di quadrimomento,

$$\hat{S} = x_1 x_2 S$$

Se perdo tutta l'energia dell'elettrone o del positrone, non ho più energia per generare la coppia  $\mu^+\mu^-$ . Generando la coppia di muoni nel CM, devo disporre di sufficiente energia, conferita dalla coppia  $e^+e^-$ , da convertire nelle rest-mass dei muoni, perciò  $\hat{S} > 4m_\mu^2$ . Altrimenti, avendo prodotto troppa radiazione, la reazione non può avvenire. Sono poi definiti i momenti con il riscalamento delle quantità. Tutto procede automaticamente, tranne per il fatto che, dovendo generare nel CM devo generare i due momenti per i muoni che sono nel CM dell'interazione, che non coincide con il CM del laboratorio. Perciò trasformo i quadrimomenti dell'interazione nei quadrimomenti nel riferimento del laboratorio. Faccio questo perché ho sviluppato in QED le formule per il processo al tree-level nel riferimento del laboratorio, che in assenza di radiazione coincide con il CM. Un'altra variazione rispetto a prima è l'inclusione delle funzioni di struttura. Introduciamo le funzioni  $D$  di struttura e proviamo ad esplorare lo spazio delle fasi con la simulazione.

### Eventi weighted

Vediamo l'integrale che viene calcolato dal software. Poniamo  $10\text{GeV}$  come energia e  $10^5$  chiamate per i processi weighted, mentre  $10^4$  per i processi unweighted.

```
ENTER SEED FOR RANDOM NUMBER GENERATION
312121
BEAM ENERGY (GeV)
10.d0
ENTER MAXIMUM NUMBER OF CALLS (for weighted evts)
100000
ENTER MAXIMUM NUMBER OF CALLS (for unweighted evts)
10000
ICNT =      100112
XSECT (weighted evts)=   231.82417112693889      +-  26.084273165019717    PB
```

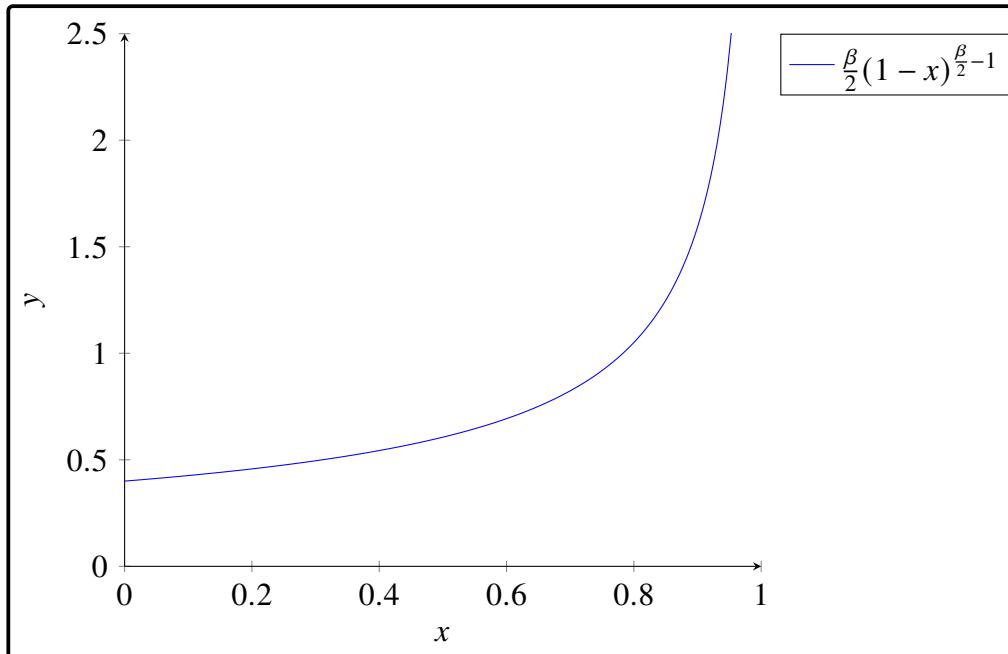
La sezione d'urto ha senso che sia raddoppiata? Perché parto con  $\frac{d\sigma}{d\cos\theta} \sim \frac{1}{s}$  e quindi sto facendo una media pesata con le probabilità date da  $D(x_1), D(x_2)$ . Poiché il processo di kernel va come  $\frac{1}{s} = \frac{1}{x_1 x_2 S}$ , è naturale che la sezione d'urto cresca. Sembra abbastanza sotto controllo, nel senso che qui l'errore aumenta di un fattore 10, anche se compatibile.

### Eventi unweighted

Transitiamo alla fase di Hit or Miss, dove utilizziamo il terzo numero random per confrontare il valore dell'integrale. Facciamo il ciclo MC da 1 a un miliardo, si aggiorna **ICNT** ogni volta che estraiamo un nuovo random. Il tutto si forma quando **nhitw** è sufficiente. Per raggiungere il milione di eventi vuol dire che l'1.1 % delle volte il controllo non passa. Perciò trova un altro punto e quindi **ICNT** è maggiore di  $1e6$ .

```
results from MC unweighted integration
n. of points= 91549653 integral= 249.84366348278277 +- 7.89675991
29825556
eff= 1.0933957128145266E-005 bias eff= 2.6973027735948563E-002
```

Il numero di punti trovati sono stati 91 milioni per estrarre  $10^4$  punti, questo perché l' $f_{\max}$  è risultato molto alto e la funzione  $\frac{\beta}{2}(1-x)^{\frac{\beta}{2}-1}$  è fatta come



una divergenza molto stretta, mentre in 0 vale  $\frac{\beta}{2}$ . È chiaro che anche se l' $f_{\max} = 22e6$  è finito, la funzione non è finita, in ogni caso avrà dei byas. Quando la  $x$  capita nelle zone alte viene scartata, quindi l'efficienza di generazione crolla. Questo è uno dei motivi per i quali l'efficienza  $10^{-5}$ , ogni  $10^5$  punti ne accetta uno. Il byas, d'altro canto, è del 3%.

### 16.2.4 Metodo di riduzione della varianza

Il rimedio va sotto il nome di metodo di riduzione della varianza. L'errore scala come  $\frac{c}{\sqrt{N}}$ . Se la distribuzione è la piatta, l'errore è 0, perché come esploro la funzione, ottengo la stessa forma. Perciò  $\langle f^2 \rangle - \langle f \rangle^2 = 0$ . L'ideale è minimizzare il più possibile  $c$ . Il fatto che per la distribuzione uniforme  $c = 0$  vuol dire che la distribuzione di punti uniformi in  $x$  è la distribuzione ideale. Dopo l'HoM, per funzioni non costanti, mi trovo densità di punti differenti in base al valore della funzione nell'intervallo. Se io fossi capace di generare secondo queste distribuzioni di punti, potrei avere errore 0, e questo mi introduce il concetto di campionamento ad importanza. Voglio andare ad utilizzare dei punti che non sono distribuiti uniformemente. Come faccio a ricondurni a distribuzioni non uniformi? Mi pongo questa domanda poiché i generatori di numeri random tipicamente estraggono da distribuzioni uniformi. Ci aiuta la statistica in questo, che ci dice che se ho  $p(x)$ , tale per cui

$$\int_0^1 dx p(x) = 1$$

Se faccio la cumulativa,

$$C(x) = \int_0^x dy p(y)$$

$C(x)$  è uniforme. Questo risultato va sotto il nome di **Teorema della cumulativa**. Questo suggerisce l'idea di ottimizzare la cumulativa come random, compreso tra (0, 1). Posso, sfruttando questo fatto, andare a generare secondo la  $p(x)$ . Parto con una distribuzione uniforme e con un cambio di variabile passo ad una distribuzione di punti che non è più uniforme. Questa è la soluzione, poiché abbiamo due funzioni molto piccate su 1 verso  $+\infty$  che creano dei problemi dal punto di vista pratico. Se noi aumentiamo i punti si vede che l'integrale che otteniamo non è affidabile, per l'incremento del bias. Anche le distribuzioni che si fanno a vedere non rispettano più l'arco di parabola previsto dalla QED e quindi l'unica via d'uscita è utilizzare il **campionamento di importanza** sulla funzione  $D(x, s)$ . Abbiamo visto che in assenza di radiazione la  $\frac{d\sigma}{d\cos\theta} \propto \cos^2\theta$  non ha problemi e si genera 1e6 punti senza problemi. Adesso invece la situazione è più seria.

#### Cumulativa uniforme

Come faccio ad ottenere  $p(x)$  dalla sua cumulativa? Devo sfruttare l'invertibilità della cumulativa, una delle condizioni necessarie a poter eseguire il campionamento di importanza.

$$C(x) = \int_0^x dx' \frac{\beta}{2} (1-x')^{\frac{\beta}{2}-1} + \beta(1+x')$$

Non si riuscirà mai ad invertire completamente tale espressione, ma si può riuscire ad individuare una sua approssimazione che ne mantiene il comportamento.

# Chapter 17

## Lezione 17

### 17.1 Campionamento di importanza

L'obiettivo del campionamento di importanza è di utilizzare punti distribuiti in un modo specifico, dettato dalle esigenze del problema. La funzione in questione è

$$p(x) = \frac{\beta}{2} (1 - x)^{\frac{\beta}{2} - 1} + \beta(1 + x)$$

Il concetto generale era quello che dobbiamo partire da una distribuzione uniforme, poiché il generatore random estrae numeri distribuiti uniformemente. Quindi vedere da questo come possiamo utilizzarlo. Ci viene fornito un modo dal teorema della cumulativa, definita come

$$C(x) = \int_{-\infty}^x dx' p(x')$$

Abbiamo definito una  $\Pi(f)$  e la definizione di **pdf** è

$$\Pi(f) = \int_{-\infty}^{+\infty} p(x) dx \delta(f(x) - f)$$

Nella definizione vedo che la  $p$  è una distribuzione di probabilità arbitraria, quindi l'integrale di una distribuzione di probabilità arbitraria è una variabile stocastica. Proviamo ad applicare tale definizione alla cumulativa

$$\Pi(C) = \int_{-\infty}^{+\infty} p(x) dx \delta\left(\int_{-\infty}^x dx' p(x') - c\right)$$

L'integrale di definizione di cumulativa è definito tra  $[0, 1]$ , perciò vediamo che per  $C \notin [0, 1] \implies \Pi(C) = 0$ .

$$\Pi(C) = \int_{-\infty}^{+\infty} p(x) dx \delta(g(x))$$

Ricordo la proprietà della

$$\delta(g(x)) = \frac{\delta(x - \bar{x})}{|g'(x)|_{x=\bar{x}}}, \quad g(\bar{x}) = 0$$

Svolgiamo l'integrale e ricordiamo che  $g'(x) = p(x)$ , perciò

$$\delta(g(x)) = \frac{\delta(x - \bar{x})}{p(\bar{x})}$$

Da questo si deduce che

$$\Pi(C) = \int_{-\infty}^{+\infty} p(x) dx \frac{\delta(x - \bar{x})}{p(\bar{x})} = \frac{p(\bar{x})}{p(\bar{x})} = 1$$

Questa è la prova che la cumulativa è una distribuzione uniforme  $\forall C \in [0, 1]$ . È la proprietà che fa al caso nostro perché noi, data una distribuzione di probabilità generica, che non ha una forma funzionale particolare, comunque, se costruiamo la cumulativa, abbiamo comunque una distribuzione uniforme tra  $[0, 1]$ . Possiamo perciò estrarre valori della cumulativa da generatori pseudo-random che "pescano" numeri in modo uniforme nell'intervallo  $[0, 1]$ . Possiamo dunque scrivere che

$$\int_{-\infty}^x dx' p(x') = C(x)$$

Abbiamo dimostrato che il valore medio di un integrale di una variabile aleatoria  $f(x)$  è

$$\langle f \rangle = \int df f \Pi(f) = \int dx f(x)$$

Questo ci lega l'integrale al valor medio, che calcoliamo come  $\frac{1}{N} \sum_{i=1}^N f(x_i)$ .

### 17.1.1 Valor medio per distribuzione non uniforme

Si è presupposto però che la distribuzione degli  $x_i$  fosse uniforme. Come cambia questa espressione per una distribuzione non uniforme?

$$\Pi(f) = \int dx p(x) \delta(f(x) - f)$$

Richiamo l'espressione di  $\langle f \rangle$

$$\begin{aligned} \langle f \rangle &= \int df f \Pi(f) = \int df f \int dx p(x) \delta(f(x) - f) \\ &= \int dx p(x) f(x) \end{aligned}$$

Se partiamo da una distribuzione di punti diversa dall'uniforme, il valor medio di  $f$  non è l'integrale della  $f$  semplicemente, ma l'integrale pesato con la distribuzione  $p(x)$ . Se al posto di  $\langle f \rangle$  poniamo  $\langle \frac{f}{p} \rangle$ , risulta

$$\langle \frac{f}{p} \rangle = \int dx f(x)$$

Nel discorso del campionamento di importanza è la cosa fondamentale. Dobbiamo integrare  $f(x)$  con punti non uniformi, ottendendo però

$$\langle \frac{f}{p} \rangle = \frac{1}{N} \sum_{i=1}^N \frac{f(x_i)}{p(x_i)}$$

Il denominatore riduce il contributo all'integrale dei punti più densi e viceversa per i punti meno densi. Quando  $p$  è uniforme, dividiamo per 1. Facciamo il caso particolare dove integriamo una funzione fattorizzata, dove sappiamo integrare perfettamente la  $g > 0$ .

$$\frac{N}{N} \int f(x) g(x) dx, \quad \int_{\mathbb{R}} g(x) dx = N$$

$$N \int f(x) \frac{g(x)}{N} dx$$

A meno di una normalizzazione ho la densità di probabilità  $\frac{g(x)}{N} = p(x)$ . Ho supposto di conoscerne la primitiva, perciò qual'è la cosa più semplice? Opero un cambio di variabile  $p(x)dx = dy$ . Ho assorbito la densità di probabilità nella mia misura  $dy$  e calcolo la  $f(x(y))$ .

$$\frac{N}{N} \int f(x) g(x) dx = N \int f(x) \frac{g(x)}{N} dx = N \int f(x(y)) dy$$

Viene interpretato tutto in termini di densità di probabilità di punti. I punti in  $y$  non sono più distribuiti uniformemente ma secondo la densità di probabilità  $p(x)$ . Tutto ciò nelle ipotesi sopra citate. È possibile avvicinarci all'integrale della  $g$  in base al suo andamento. In quel caco possiamo prendere la funzione nel limite in cui diventa piccata. Il limite dell'algoritmo è riuscire ad invertire  $x$  in funzione di  $y$ , dove  $y$  è il nostro numero random.

### 17.1.2 Applicazione nel nostro caso

Se facciamo il nostro caso specifico,

$$g(x) = \frac{\beta}{2} (1-x)^{\frac{\beta}{2}-1} + \beta(1+x)$$

L'inversione non è immediata, anzi, seppur integrabile, non si riesce ad invertire facilmente la primitiva. Eppure abbiamo una funzione polinomiale perciò possiamo approssimarla con il termine  $\frac{\beta}{2}(1-x)^{\frac{\beta}{2}-1}$ . Per il

campionamento ad importanza

$$\int_{[0,1]} dx_1 \int_{[0,1]} dx_2 D(x_1, s) D(x_2, s) \int d\cos\theta d\varphi \frac{d\sigma}{d\cos\theta d\varphi}(x_1, x_2, s)$$

Il fattore dominante della radiazione, il limite in cui l'energia dei fotoni emessi è molto piccola, è universale e coincide con l'espressione sopra riportata.

$$\int_0^1 dx_1 \int_0^1 dx_2 P(x_1, s) P(x_2, s) \frac{D(x_1, s) D(x_2, s)}{P(x_1, s) P(x_2, s)} \int d\cos\theta d\varphi \frac{d\sigma}{d\cos\theta d\varphi}(x_1, x_2, s)$$

Per le  $P$  dobbiamo prendere qualcosa che somigli alle funzioni di struttura nella forma funzionale o che le riproduca nella zone in cui diventa molto piccata, dunque sorgente dei problemi di varianza.

$$P(x, s) = \frac{\beta}{2} (1-x)^{\frac{\beta}{2}-1}$$

Valutiamone la normalizzazione

$$\begin{aligned} \int_0^1 \frac{\beta}{2} (1-x)^{\frac{\beta}{2}-1} dx &=_{y=1-x} \int_0^1 \frac{\beta}{2} y^{\frac{\beta}{2}-1} dy \\ y^{\frac{\beta}{2}} \Big|_0^1 &= 1 \end{aligned}$$

Risulta già normalizzata. Non dobbiamo preoccuparci di valutarne la normalizzazione. Calcoliamo la sua cumulativa

$$\begin{aligned} \int_0^x \frac{\beta}{2} (1-x')^{\frac{\beta}{2}-1} dx' &= \int_0^{C(x)} dC' = C(x) \\ [-(1-x')^{\frac{\beta}{2}}]_0^x &= C(x) \\ -(1-x)^{\frac{\beta}{2}} + 1 &= C(x) \end{aligned}$$

Dobbiamo invertire la relazione

$$x = 1 - (1-C)^{\frac{2}{\beta}}$$

Proviamo a valutare

$$C = 0 \implies x = 0$$

$$C = 1 \implies x = 1$$

Il nostro integrale della sezione d'urto diventa

$$\int dC_1 \int dC_2 \frac{D(x_1(C_1), s) D(x_2(C_2), s)}{P(x_1(C_1), s) P(x_2(C_2), s)} \int d\cos\theta d\varphi \frac{d\sigma}{d\cos\theta d\varphi}(x_1(C_1), x_2(C_2), s)$$

Se come  $D$  prendessimo esattamente la parte non polinomiale, il rapporto tra  $D$  e  $P$  varrebbe 1. è come se cambiando la distribuzione di  $x_1, x_2$  dall'uniforme all'altra che coincide con la  $D$ , abbiamo cambiato la funzione integranda e l'abbiamo posta = 1. Questa è l'idea del campionamento di importanza. Chiaramente, avendo la  $D$  un componente polinomiale, sarà  $\sim 1$  nella parte divergente e diventerà piccola nella parte convergente. Dunque costituirà un contributo piccolo. La parte relativa alle funzioni di struttura si sta muovendo tra  $[0, 1]$ . La varianza è ridotta drasticamente. Oltre a facilitare l'integrazione, riduce l'errore e a livello di generazione di punti della simulazione per vedere la distribuzione dei momenti nello spazio delle fasi, vediamo che i momenti vanno su e vediamo che c'è un byas importante, mentre ora è molto più facile ottenere risultati con piccolo bays. Più aumentiamo il valore massimo della nostra funzione, più l'esigenza di generazione crolla. Quindi se generiamo basandoci su un valore molto piccato, il rimanente dominio della funzione avrà un'efficienza bassa. Se utilizzo un  $f_{\max}$  piccolo, invece, ho un'efficienza di byas più alta. Quando la  $D$  tende ad infinito, il denominatore  $P$  controlla tale divergenza, trasformandola in una convergenza ad un rapporto finito. Il campionamento di importanza serve anche ad evitare di campionare troppo tratti costanti della funzione, concentrandosi sui tratti più piccati o variabili, focalizzandosi quindi sui valori più alti di derivata.

### 17.1.3 Implementazione

Per poter calcolare correttamente il rapporto tra  $D$  e  $P$ , devo chiamare una nuova funzione,

$$\frac{\frac{\beta}{2}(1-x)^{\frac{\beta}{2}-1} + \beta(1+x)}{\frac{\beta}{2}(1-x)^{\frac{\beta}{2}-1}} = 1 + \frac{\beta(1+x)}{\frac{\beta}{2}(1-x)^{\frac{\beta}{2}-1}} = 1 + \frac{2(1+x)}{(1-x)^{\frac{\beta}{2}-1}}$$

Proviamo una run con energia  $10GeV$ ,  $1e5$  punti weighted e  $1e3$  unweighted.

```
ICNT = 100001
XSECT (weighted evts)= 440.02639370174234 +- 23.740429881510245 PB
funvalmax= 1175344.2042435647

results from MC unweighted integration
n. of points= 2544276 integral= 462.41820617446990 +- 14.612765755263428
eff= 3.934321575990943E-004 bias eff= 2.1978022530674934E-002
Note: The following floating-point exceptions are signalling: IEEE_DIVIDE_BY_ZERO
real 0m28.399s
user 0m6.983s
sys 0m0.008s
```

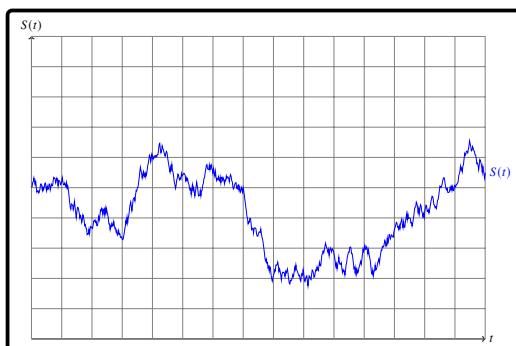
Si segnala una divisione per 0 ma con un numero di punti utilizzati molto migliore. Un fattore di efficienza dell'ordine di  $10^{-3}$ . Prima avevamo un fattore peggiore. Si dovrebbe capire perché c'è la divisione per 0. Il problema è il seguente. Abbiamo

$$x = 1 - (1 - C)^{\frac{2}{\beta}}$$

Quando  $C \sim 1$ , per effetto dei round-off è possibile che si ottenga una  $x \sim 0$ . Conviene generare una variabile  $1 - x = (1 - C)^{\frac{2}{\beta}}$ . Riscrivo  $x = 1 - (1 - x)$ . Andiamo a vedere la massa invariante, che non è più addensata sull'ultimo bin. Adesso la radiazione mi dà una distribuzione che dipende dalla radiazione. Se andiamo a vedere l'angolo, mantiene il suo andamento, perché generiamo variabili di radiazione e andiamo a calcolare il processo ad energie nel CM diverso, per effetto della perdita di energia per radiazione. Se istogrammiamo il  $\cos \theta$ , vedremo sempre un andamento simmetrico rispetto a 0, con fasci simmetrici. Gli effetti di Lorentz boost da una parte e dall'altra si compensano e si ottiene una figura sempre simmetrica. Se mi metto nel laboratorio e non nel CM, la figura cambia. Se ho fasci simmetrici che emettono in modo simmetrico, emettono radiazioni in modo asimmetrico, e allora si vedrebbero effetti sulla distribuzione angolare nel laboratorio. Si possono fare un po' di prove con questa versione del programma e controllare il perché di certi comportamenti.

## 17.2 Option Pricing

Bisogna introdurre l'argomento in generale. Si spendono due parole sull'Econofisica, cioè l'utilizzo dei metodi della Fisica Teorica nello studio dei mercati finanziari. È nato negli anni '60 con il primo studio di Mandelbrot dei frattali. C'è stato un grande sviluppo fino agli anni '90. Un aspetto da tenere presente dei mercati finanziari è che i prezzi dei vari prodotti finanziari è un processo stocastico, che non segue leggi deterministiche ma è guidato dalla legge della domanda e dell'offerta. Un esempio di processo stocastico è il Moto Browniano. Il prezzo di un prodotto  $S(t)$  può avere un andamento aleatorio.



### 17.2.1 Opzioni

Sono un contratto stipulato tra due parti. Il **writer** e l'**holder**. Il writer emette il contratto a un certo tempo  $t = 0$  e noi conosciamo il prezzo di qualunque merce  $S(0)$ . Le opzioni sono dei derivati e il loro prezzo dipende dal prezzo dell'azione.  $S(0)$  si trova pubblicamente ed è determinato dalla legge della domanda e dell'offerta. Ad un certo tempo fissato dal contratto  $T$  detto **maturity**, viene fissato  $X$ , il **prezzo strike**. Il contratto dà all'holder, che paga il contratto un prezzo iniziale, la possibilità di andare dal writer ed acquistare un certo numero di azioni. Il contratto viene stipulato a livello assicurativo e a livello speculativo.

- Nel caso assicurativo possiamo pensare al petrolio oppure al cambio di divise. Siccome il prezzo è un processo stocastico, si può andare ad un valore molto alto, oppure può scendere. L'holder è sicuro che per proteggersi da una variazione eccessiva del prezzo, pagherà il prezzo dell'azione come  $X$ . Una certa azienda si assume il rischio che il prezzo si alzi.
- A livello speculativo si stipula per differenza di visioni di writer ed holder. Qualora ci sia la situazione in cui  $S(T) < X$ , l'holder acquista a  $X$  e rivende sul mercato ad  $S(T)$ , perciò guadagna  $S(T) - X$ . Questo nel caso delle **call-option**, oppure Europee, Plain Vanilla, oppure Europee semplici. Se al tempo finale  $S(T) < X$ , l'holder non esercita il suo diritto di opzione.

Il prezzo iniziale dell'opzione deve essere razionale e giusto per entrambe le parti, altrimenti non si stipulerebbe. Da questo nasce il problema dell'**option-pricing**.

### 17.2.2 Modello di Black&Scholes

è stato risolto nel 1973 da Black&Scholes e Merton, matematici e fisici con PhD in finanza. Con certe ipotesi si riduce una SDE ad una PDE, cioè un'equazione differenziale deterministica. Per questo risultato Scholes e Merton hanno vinto il premio Nobel.

$$\begin{aligned} \text{SDE : } dS(t) &= \mu S(t)dt + \sigma S(t)dW(t) \\ \text{PDE : } rO &= \frac{\partial}{\partial t} O + rS \frac{\partial O}{\partial S} + \frac{1}{2} \sigma^2 S^2 \frac{\partial^2 O}{\partial S^2} \end{aligned}$$

I parametri sono  $\sigma$ ,  $\mu$ .  $\sigma$  ha il significato di volatilità delle opzioni, che nello sviluppo del modello viene assunto costante, anche se si dimostra che non lo è.  $r$  è il tasso di interesse privo di rischio, che viene dato quando si fa un deposito in banca. Qualunque investimento privo di rischio dà  $r$ .  $\mu$  è il tasso di interesse nel caso di rischio. Nel modello di B&S si è determinato il valore di  $O$  in un caso privo di rischio. Si riesce ad eliminare  $\mu$  con il Lemma di Itô e con la costruzione del portafoglio di B&S. Con questo modello vedremo come calcolare il valore delle opzioni con la simulazione MC. B&S danno un'espressione chiusa della soluzione, che noi forniremo in maniera numerica, confrontandola con il risultato analitico di B&S. Il metodo MC ci permetterebbe di esplorare finanza oltre il modello di B&S, eppure per brevità di didattica, non verrà trattato come argomento in questo corso. Il processo di Wiener consiste nel generare numeri che rispettano la distribuzione Gaussiana. Perciò non potremmo affrontare il problema con il campionamento di importanza, per l'impossibilità di invertire la cumulativa della Gaussiana, non esprimibile in termini di funzioni elementari.

# Chapter 18

## Lezione 18

### 18.1 Modello di Black&Scholes

Sotto le ipotesi del modello, si può trovare la soluzione analitica all'eq. differenziale, potendo confrontare la soluzione numerica con la soluzione analitica.

$$dS(t) = \mu S(t)dt + \sigma S(t)dW(t)$$

Dove  $W(t)$  è il processo di Wiener, la cui densità di probabilità è

$$\begin{aligned} p(x, \Delta t) &= \frac{1}{\sqrt{2\pi\Delta t}} e^{-\frac{x^2}{2\Delta t}} \\ \frac{\partial}{\partial t} p(x, t) &= \frac{1}{2} \frac{\partial^2}{\partial x^2} p(x, t) \\ p(x, 0) &= \delta(x) \end{aligned}$$

Si tratta di una PDE e nell'ultima lezione vediamo qualche esempio di soluzione numerica di queste. È esattamente la stessa forma che si trova in MQ nel path-integral. Lo stesso vale per l'equazione di Fokker-Planck, che contiene le stesse derivate dell'Eq. di Schrodinger, eccezion fatta per le quantità che non sono complesse, ma reali. Si pone attenzione sulle condizioni iniziali poste da Einstein per la descrizione dei processi di diffusione. L'equazione differenziale che descrive la variazione del prezzo spot  $S(t)$  è una SDE, cioè Stochastic Differential Equation. Questo perché contiene un termine  $dW(t)$ , il differenziale corrispondente al processo di Wiener, un processo stocastico. Tutto è proporzionale ad  $S(t)$ , è un GBM, Geometric Brownian Motion. Anziché il differenziale del prezzo spot, possiamo scrivere il differenziale del suo logaritmo. Per il Lemma di Itô il termine di drift si arricchisce di  $-\frac{1}{2}\sigma^2$ .

$$d \ln S(t) = \left( \mu - \frac{1}{2}\sigma^2 \right) dt + \sigma dW(t)$$

Ciò ha di bello che nel logaritmo è un Moto Browniano non Geometrico.

#### 18.1.1 Eq. di Black & Scholes

Si scrive l'eq. di B&S che include  $r$

$$rO = \frac{\partial}{\partial t} O + rS \frac{\partial}{\partial t} S + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 O}{\partial S^2}$$

$O$  è il prezzo dell'opzione.  $C$  è la call, quando esercito il diritto di acquistare.  $P$  è il diritto di vendere sottostante. Esistono due categorie di queste opzioni. Da una parte si acquista il diritto di acquistare sottostante ad un certo prezzo  $X$ , mentre dall'altra si acquista il diritto di vendere sottostante al prezzo  $X$ . Le funzioni di guadagno si scrivono corrispondentemente come  $C$  e  $P$ .

$$\begin{aligned} C(S, t = T) &= \max(S(T) - X, 0) \\ S = 0 \quad C(0, T) &= 0 \\ S \rightarrow \infty \quad C(S, t) &\rightarrow S \end{aligned}$$

Al Chicago Board of Trade i produttori di cotone ne producevano grandi quantità, anche se soggette a grandi fluttuazioni di prezzo in quel periodo. Hanno introdotto questo strumento come assicurazione, dove vendevano il prodotto a 6 mesi di distanza quando il prodotto era vendibile, minimo ad un prezzo  $X$ .

$$C(S, t) = SN(d_2) - XE^{-r(T-t)}N(d_1)$$

Il fattore  $d_1$  è un'espressione elaborata dei parametri del modello.

$$d_1 = \frac{\ln \frac{S}{X} + (r + \frac{\sigma^2}{2})(T - t)}{\sigma \sqrt{T - t}}$$

$$d_2 = d_1 - \sigma \sqrt{T - t}$$

La funzione  $N$  è la cumulativa della Gaussiana

$$N(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-\frac{x^2}{2}} dx$$

### 18.1.2 Modellizzazione numerica

Noi dovremmo fare una simulazione di tutte le possibili traiettorie percorse da  $S(t)$  e fare una media di ensamble su un insieme statistico di valori di  $S(T)$ . Con tale valore bisogna poi ricavare  $S(T) - X$ . Il prezzo  $C(T)$  numericamente, è

$$C(T) = \left( \frac{1}{N} \sum_{i=1}^N \max(S_i(T) - X, 0) \right) e^{-rT}$$

Dove  $e^{-rT}$  è il fattore di attualizzazione. Per scrivere il mio incremento  $dS(t)$  in termini discreti, devo prima scriverne lo sviluppo

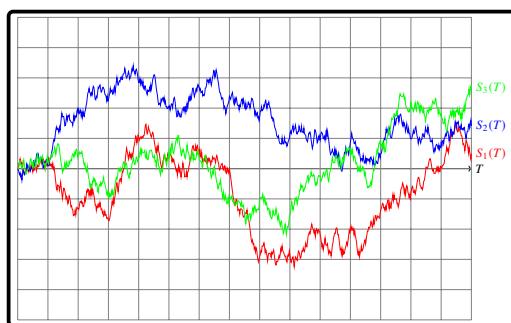
$$S(t_n + \Delta t) = S(t_n) + S(t_n)(\mu \Delta t + \sigma X_n \sqrt{\Delta t})$$

Il termine con la radice dà l'allargamento temporale della distribuzione di probabilità.  $X_n$  è un numero Gaussiano distribuito con media 0 e varianza 1. Passando al processo con il logaritmo, abbiamo

$$\ln S(t_n + \Delta t) - \ln S(t_n) = \left( \mu - \frac{\sigma^2}{2} \right) \Delta t + \sigma X_n \sqrt{\Delta t}$$

$$S(t_n + \Delta t) = S(t_n) \exp \left\{ \left( \mu - \frac{\sigma^2}{2} \right) \Delta t + \sigma X_n \sqrt{\Delta t} \right\}$$

Conosciamo il prezzo spot alla firma del contratto, ovvero  $S_0$ . Questo prezzo si evolve nel tempo. Poiché stiamo discretizzando gli incrementi, discretizziamo anche gli intervalli temporali. Naturalmente l'eq. di B&S ha una soluzione analitica nel limite del continuo, mentre nel mondo della finanza tutto ha una granularità, oltre la quale non si riesce ad andare. Avendo informatizzato tutto da qualche lustro, l'unità minima di tempo è dell'ordine del secondo. Il processo stocastico è compreso nel processo  $S(t)$  con il numero  $X_n$ .

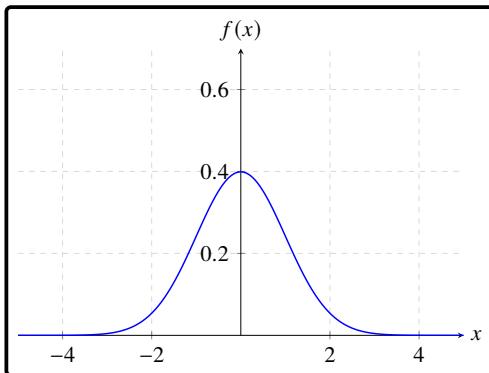


Io posso impostare  $N$  cammini e in base al percorso che compiono, terminano a valori differenti.  $S_i(t)$  è il valore dell'  $i$ -esima traiettoria compiuta. La distribuzione dei prezzi non l'abbiamo ancora vista.

### 18.1.3 Generatore di numeri random normali

Lo strumento che mi calcola le traiettorie occorre di un generatore di numeri random distribuiti in modo normale.

$$p(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$



Noi al momento abbiamo un generatore di numeri distribuiti in modo uniforme. Come possiamo trasformare la distribuzione dei numeri in una distribuzione gaussiana? Bisogna ragionare anche sul fatto che la distribuzione del logaritmo del prezzo non è normale, ma è una distribuzione con le code grasse. Si è provato a farla rientrare nella curva di Lévy, ma si scopre che neanche la distribuzione di Lévy riesce a sovrapporsi alle code delle serie storiche. L'idea di utilizzare la Gaussiana è la necessità di poter risolvere analiticamente il modello e di poter utilizzare il Teorema del Limite Centrale. La cosa interessante dell'Econofisica è che si è evoluta in parallelo allo sviluppo della comprensione del Moto Browniano. Un articolo del 1908 ha illustrato come si potesse approssimare la borsa di Parigi sovraccarica di informazioni come un sistema termodinamico. Inoltre Mantegna&Stanley hanno provveduto ad analizzare le serie storiche per comprendere se le distribuzioni dei prezzi fossero o meno delle gaussiane, ciò negli anni '90 – 2000, quando c'è stata l'esplosione di questi studi. Se necessito di 6 ore di calcolo per identificare la soluzione di un'eq. differenziale, la prospettiva diventa meno entusiasmante. Negli anni 2000 c'è stata l'esplosione dell'impiego di Matematici e Fisici nello studi della finanza. Abbiamo visto che per modificare la distribuzione di probabilità con il campionamento di importanza dobbiamo conoscerne la primitiva, perciò non possiamo seguire tale strada con la Gaussiana. Ci sono tre vie di uscita, due delle quali lasciano per esercizio.

1. Una somma di variabili **i.i.d.** sappiamo che sarà distribuita come una Gaussiana.

$$X_n \propto \sum_{i=1}^{N_1} x_i$$

Fissiamo un  $N$  e partendo dalle  $x$  distribuite uniformemente possiamo trovare la  $\varphi$  distribuita gaussianamente. Chiaramente se  $x_i \in [0, 1]$ , allora  $\varphi \in [0, N]$ . Come poi la varianza delle variabili è  $\frac{1}{12}$ . Bisogna poi fare un rescaling della variabile  $\varphi$  per farla tornare ad essere distribuita normalmente tra  $[0, 1]$ . Voglio che  $X_n$  abbia media  $\mu$  e varianza  $\sigma^2 = 1$ . Che la media sia nulla vuol dire che posso avere processi sia positivi che negativi.

2. L'altro modo è di fare semplicemente un Hit or Miss e adesso genero uniformemente, sull'asse delle  $X$  e faccio un HoM e accetto soltanto i punti compresi tra 0 e la campana. Dobbiamo stare attenti al fatto che il supporto della Gaussiana è  $\mathbb{R}$ , mentre noi possiamo generare numeri tra  $[-C, +C]$ .
3. La terza opzione, e anche la più efficiente, è l'algoritmo di Box-M'uller, che vediamo in breve. L'idea è che non so scrivere in forma chiusa l'integrale Gaussiano, ma so calcolare l'integrale definito in due dimensioni.

$$\int_{\mathbb{R}^+} dx dy e^{-(x^2+y^2)} = \pi$$

Si vuole sfruttare il fatto che in due dimensioni sappiamo calcolare l'integrale, e con questo trovare un modo di stimare l'integrale in una dimensione. Si genera una coppia di numeri random distribuiti uniformemente  $x_1, x_2$ , tali che

$$p(x_1, x_2)A = 1$$

sul quadrato  $[0, 1]^2$ . Proviamo a fare un cambio di coordinate.

$$y_1 = y_1(x_1, x_2)$$

$$y_2 = y_2(x_1, x_2)$$

$y_1, y_2$  sono funzioni di variabili stocastiche, e quindi esse stesse variabili stocastiche. Se abbiamo  $x_1, x_2$  distribuite uniformemente nel quadrato, la probabilità è  $p(x_1, x_2)dx_1dx_2$ . Se prendo i punti ed effettuo un cambio di variabile, la probabilità non deve cambiare.

$$p(x_1, x_2)dx_1dx_2 = p(y_1, y_2)dy_1dy_2$$

$$p(y_1, y_2) = p(x_1, x_2) \left| \frac{\partial(x_1, x_2)}{\partial(y_1, y_2)} \right|$$

Ottengo la seconda relazione dalla prima, esplicitando lo Jacobiano del cambio di coordinate e mi specializzo nel caso in cui  $p(x_1, x_2) = 1$ . Esplicito la legge di trasformazione

$$\begin{cases} y_1 = \sqrt{-2 \ln x_1} \sin 2\pi x_2 \\ y_2 = \sqrt{-2 \ln x_1} \cos 2\pi x_2 \end{cases}$$

Provo a sommare in quadratura le variabili

$$y_1^2 + y_2^2 = -2 \ln x_1$$

$$x_1 = e^{-\frac{1}{2}(y_1^2+y_2^2)}$$

ed ottengo un prodotto di gaussiane. Se invece provo a dividerle

$$\frac{y_1}{y_2} = \tan 2\pi x_2$$

$$x_2 = \arctan \frac{y_1}{y_2}$$

Con queste relazioni possiamo calcolare il determinante dello Jacobiano.

$$\begin{vmatrix} \frac{\partial x_1}{\partial y_1} & \frac{\partial x_1}{\partial y_2} \\ \frac{\partial x_2}{\partial y_1} & \frac{\partial x_2}{\partial y_2} \end{vmatrix} = \begin{vmatrix} -\frac{x_1 y_1}{2\pi} & -\frac{x_1 y_2}{2\pi} \\ -\frac{1}{2\pi} \frac{y_2}{y_1^2 + y_2^2} & \frac{1}{2\pi} \frac{y_1}{y_1^2 + y_2^2} \end{vmatrix} = \frac{1}{2\pi} e^{-\frac{1}{2}(y_1^2+y_2^2)}$$

$$p(y_1, y_2) = \frac{1}{2\pi} e^{-\frac{1}{2}(y_1^2+y_2^2)} = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}y_1^2} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}y_2^2}$$

Guardiamo l'uguaglianza tra primo e ultimo termine, vediamo che è fattorizzata tra due Gaussiane. È vero che non sappiamo calcolare a partire da un numero random distribuito uniformemente un numero random distribuito gaussianamente, però se partiamo da un array di numeri pseudo-random distribuiti uniformemente possiamo calcolare due numeri pseudo-random ciascuno distribuito gaussianamente, poiché la distribuzione è fattorizzata. Questa è la caratteristica speciale dell'algoritmo di Box-M'uller. Poiché si generano due numeri, si prova ad impiegarli entrambi.

La routine che si commenta ora è minimale nell'utilizzo dell'informazione. Prima si utilizza un bit di informazione e poi l'altro. La coppia è distribuita come una gaussiana bidimensionale, senza correlazione, poiché fattorizzabili.

## 18.2 Script Fortran

Importiamo lo script di generazione dei numeri gaussiani nello script di calcolo del prezzo delle opzioni. Provo a runnare lo script.

```

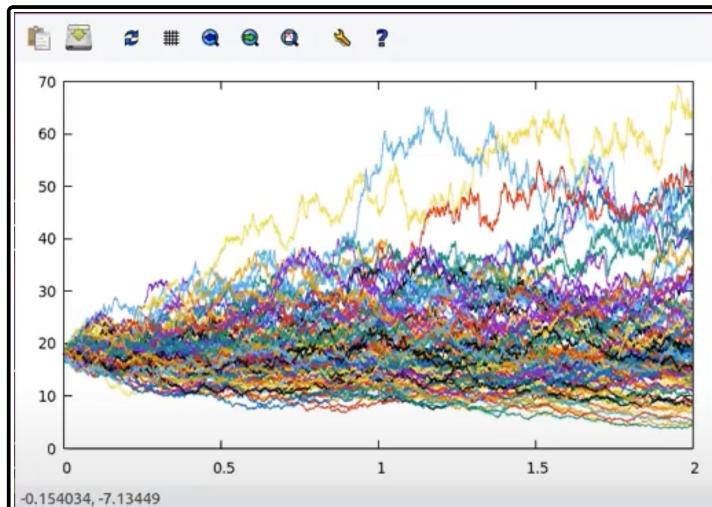
d1= 1.6754659446761757
d2= 1.1097805197269377

result from qsymp for d1
n. of points= 65 integral= 0.95307875022791089

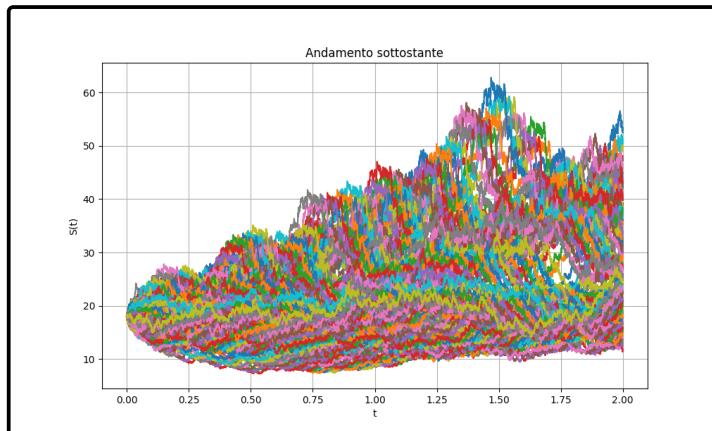
result from qsymp for d2
n. of points= 65 integral= 0.86645534452910666
analytical value c/k= 1.0061481136754817
analytical value c = 10.061481136754818
opt = 8.8064821776041100 +- 0.91093673733263703

```

Notiamo che mentre il valore analitico è di  $10.06$ , il valore ottenuto con il metodo MC è di  $8.81 \pm .91$ . C'è agreement tra il valore analitico e il valore MC, anche se chiaramente il valore analitico fa riferimento ad un intervallo di tempo continuo, quindi si potrebbe aumentare la granularità dell'intervallo nel metodo MC per aumentarne l'accordo. Posso aumentare il numero di traiettorie e la granularità. Se plotto i file prodotti dallo script, visualizzo tutti i cammini



Visto che sto studiando il logaritmo, i prezzi non sono simmetrici rispetto al prezzo iniziale. Utilizziamo i punti finali per fare una valutazione dell'opzione. Sembrano dei valori possibili di valore delle azioni. Questo è il caso più semplice, del quale disponiamo anche della soluzione analitica. Si possono però definire dei prodotti derivati che non consentono un'adeguata trattazione analitica e soltanto con un algoritmo che sfrutta il metodo MC si può stimare il prezzo, assumendo un modello per il sottostante. Al momento sono studiati i modelli che vanno oltre il limite gaussiano e che implementano la volatilità stocastica. In quel caso non è possibile ottenere una soluzione analitica e si procede per via numerica, affidandosi a tecniche MC di questo tipo. Ci sono quantità derivate che si possono calcolare sulla base delle traiettorie del sottostante. La stessa immagine con 10 volte la granularità temporale e il doppio delle traeittorie risulta come segue con **matplotlib**.



## 18.3 Algoritmo Metropolis

Se si ha un sistema con moltissimi gradi di libertà si implementa un prototipo di machine learnign, che impara da solo com'è fatta una funzione e va a campionare da solo sullo spazio delle fasi le regioni interessanti. Si utilizza molto in Meccanica Statistica. È stato utilizzato nello studio della diffusione di neutroni nella costruzione della bomba atomica.

# Chapter 19

## Lezione 19

### 19.1 Somma di distribuzioni uniformi

Se provo a generare numeri gaussiani sommando diversi numeri pseudo-random uniformi, sfruttando quindi il teorema del limite centrale, devo stare attento a sommare diverse distribuzioni di media 0 e varianza 1. Poiché la s.d. della distribuzione uniforme è  $\frac{b-a}{\sqrt{12}}$ , per averla pari ad 1, devo avere  $b-a = \sqrt{12}$ , perciò posso settare  $b = \sqrt{3}$  e  $a = -\sqrt{3}$ . Dopodiché, ogni numero  $\xi \in (0, 1)$  quando lo sommo

$$\varphi = \sum_{i=1}^N \xi_i$$

Se generassi invece numeri p-r u. sull'intervallo  $[0, 1]$ , avrei media  $\frac{1}{2}$  e s.d.  $\frac{1}{\sqrt{12}}$ . Perciò dovrei trasformare il numero generato nell'intervallo  $[0, 1]$  nel numero generato nell'intervallo  $[-\sqrt{3}, \sqrt{3}]$ . Per ottenere tale risultato devo operare una combinazione convessa

$$x_i = (1 - \xi_i)a + \xi_i b = \sqrt{3}(2\xi_i - 1)$$

Altrimenti posso trasformare direttamente la variabile  $\varphi$  nella variabile  $y$

$$\begin{aligned} y &= \varphi - \frac{N}{2} \\ \langle y \rangle &= \langle \varphi \rangle - \frac{N}{2} \\ \sigma^2(y) &= \sigma^2(\varphi) \end{aligned}$$

Per normalizzare la varianza effettuo un secondo cambio di variabile, stavolta sui numeri pseudo-random uniformi.

$$\begin{aligned} z &= \sqrt{12}\xi \\ \sigma^2(z) &= 12\sigma^2(\xi) = 1 \end{aligned}$$

In definitiva

$$\begin{aligned} \varphi &= \sqrt{3} \sum_{i=1}^N (2\xi_i - 1) \\ \varphi &= \sqrt{12} \sum_{i=1}^N \left( \xi_i - \frac{1}{2} \right) \end{aligned}$$

Dove sopra compare il risultato della manipolazione di ogni variabile e sotto compare la manipolazione della variabile somma e delle variabili. La soluzione è quindi unica, anche se gestendo le variabili si raggiunge la soluzione con un passaggio. Si tratta di una trasformazione simile al passaggio da Gaussiana a Gaussiana normalizzata.

## 19.2 Equazioni differenziali

La parte preponderante cura le ODE, con derivata rispetto ad una variabile. Cominciamo con l'equazione dell'oscillatore armonico.

$$\begin{cases} \frac{\partial^2}{\partial t^2} \varphi(t) + \omega^2 \varphi(t) = 0 \\ \varphi(0) = \varphi'(0) = \end{cases}$$

Il problema di Cauchy è risolvibile con un numero congruo di condizioni al contorno. Poiché l'equazione è del second'ordine, sono sufficienti due condizioni.

### 19.2.1 Algoritmo di Eulero

L'algoritmo più semplice è l'algoritmo di Eulero, che è molto semplice. Costruire la funzione vuol dire conoscere la funzione tra  $(0, T)$ . Dobbiamo passare ad una discretizzazione del nostro intervallo temporale e perciò dobbiamo introdurre la discretizzazione della derivata, cioè quanto curato nella prima lezione.

$$\frac{\partial^2}{\partial t^2} \varphi(t) \implies \frac{1}{h^2} [\varphi(t_{i+1}) - 2\varphi(t_i) + \varphi(t_{i-1})]$$

Si introduce la notazione compatta  $\varphi_i = \varphi(t_i)$ . Se faccio questa discretizzazione ottengo

$$\frac{1}{h^2} [\varphi_{i+1} - 2\varphi_i + \varphi_{i-1}] + \omega^2 \varphi_i = 0$$

In questo algoritmo compaiono tre valori della mia griglia. Possiamo, per esempio, risolvere questa equazione per  $\varphi_{i-1}$ .

$$\varphi_{i+1} = (2 - h^2 \omega^2) \varphi_i - \varphi_{i-1}$$

Questo è un algoritmo ricorsivo, nel senso che se conosco  $\varphi_i, \varphi_{i-1}$  posso calcolare il terzo punto. Si suppone  $\varphi_0 = 1$  e  $\varphi'_0 = 0$ . Per  $\varphi'_0$  posso scrivere

$$\frac{\varphi_1 - \varphi_0}{h} = 0 \rightarrow \varphi_1 = \varphi_0$$

Quindi per  $\varphi_0 = 1, \varphi_1 = 1$ , li pongo nella relazione ricorsiva e trovo  $\varphi_2$ . Di seguito individuo tutti i valori della funzione per istanti successivi. Fa parte della classe degli **algoritmi forward**, che date le condizioni iniziali si propaga in avanti l'informazione. Ci sono diversi fattori che possono contribuire alla precisione o meno della soluzione, come l'algoritmo per il calcolo della derivata seconda e la spaziatura. Il limite ideale è il limite in cui facciamo tendere  $h \rightarrow 0$ , anche se bisogna poi stare attenti a come si produce

$$m \frac{\partial^2}{\partial t^2} x(t) = F(x)$$

Se considero il sistema

$$\begin{cases} v(t) = \frac{dx}{dt} \\ m \frac{dv}{dt} = F(x) \end{cases}$$

L'espressione da risolvere numericamente la trasformiamo così in un sistema di equazioni. Entrambe sono equazioni del primo ordine. Il problema generale che andiamo a discutere è un problema del tipo

$$\frac{dy}{dx} = f(x, y)$$

$x$  è la variabile indipendente e  $y$  è la variabile indipendente.  $y(x)$  è la funzione che devo ricostruire. A questa espressione si affianca una condizione al contorno. L'algoritmo di Eulero su questa funzione si traduce in

$$\frac{y_{n+1} - y_n}{h} + O(h) = f(x_n, y_n)$$

Risolvo per  $y_{n+1}$

$$y_{n+1} = y_n + h f(x_n, y_n) + O(h^2)$$

è una forma semplicissima che si può vedere quanto bene funzioni.

### 19.2.2 Esempi di problemi integrabili

#### Primo esempio

Scriviamo

$$f(x, y) = -xy \quad y(0) = 1$$

$$\begin{cases} \frac{dy}{dx} = -xy \\ y(0) = 1 \end{cases}$$

Quindi

$$\begin{aligned} \frac{dy}{d} &= -xdx \\ d \ln y &= -\frac{1}{2} dx^2 \\ \ln(y) &= -\frac{1}{2} x^2 + C \\ y &= ke^{-\frac{1}{2} x^2} \end{aligned}$$

$C$  la determino con la condizione iniziale, poiché  $y(0) = 1 \Rightarrow e^C = 1 \Rightarrow C = 0$ .

#### Secondo esempio

$$\begin{cases} \frac{dy}{dx} = -y \\ y(0) = 1 \end{cases}$$

Essendo una ODE a var. sep., scrivo

$$\begin{aligned} \int \frac{dy}{y} &= - \int dx \\ \ln y &= -x + C \\ y &= e^{-x+C} \Rightarrow_{C=0} y = e^{-x} \end{aligned}$$

Per la condizione iniziale  $C = 0$ .

### 19.2.3 Applicazione algoritmo di Eulero al primo esempio

Possiamo confrontare la bontà della soluzione identificata con l'algoritmo di Eulero con la soluzione analitica dei problemi integrabili. Quando non ho la soluzione analitica come faccio? Quando ho un problema con condizioni iniziali e utilizzo un algoritmo di tipo forward, ho la mia funzione conosciuta nell'istante iniziale con la mia condizione iniziale e devo conoscere la mia funzione in un certo intervallo e non so se questa soluzione approssima la mia funzione. Poi mi serve conoscere la funzione in alcuni punti e un criterio che posso utilizzare è che se conosco la funzione, prendo la stessa equazione differenziale e metto il punto a cui sono arrivato. Stavolta però utilizzo un algoritmo backward per accertarmi che la corrispondente soluzione sia identica alla soluzione individuata con l'algoritmo forward. Vediamo così quando la nostra soluzione numerica sia affidabile. È un criterio di ragionevolezza, non analitico, ma appunto ragionevole.

#### Lo script

Per tenere traccia dei progressi dell'algoritmo, scrivo su un file lo step, la  $x$ , la  $y$  numerica, la  $y$  analitica e la differenza percentuale tra di esse, riferita alla media delle due.

$$\delta = 2 \frac{y_n - y_a}{y_n + y_a}$$

A questo punto ho calcolato la  $f(x_n, y_n)$ , conosco  $y_n$  e calcolo  $y_{n+1}$ . Perciò implemento l'algoritmo di Eulero

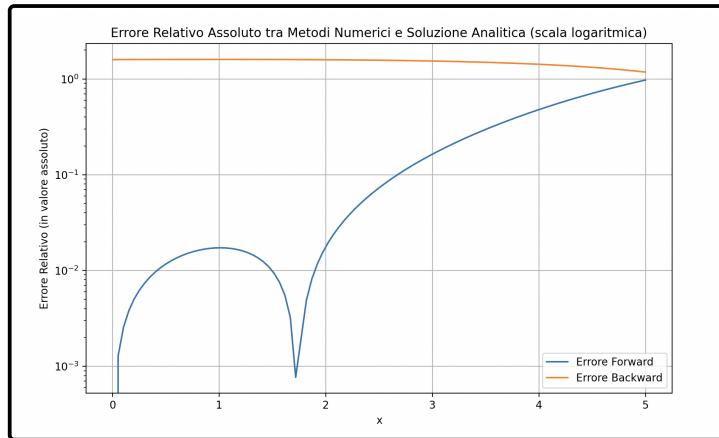
$$y_{i+1} = y_i + h(-x_i y_i)$$

#### Errore relativo

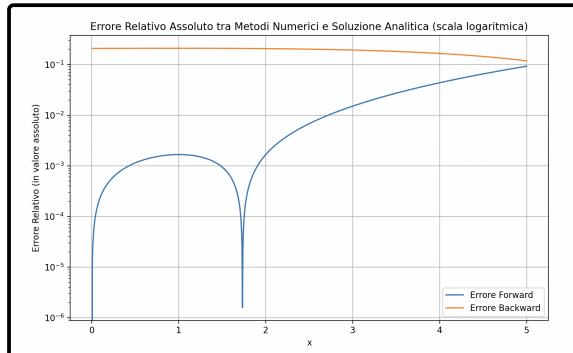
Se consideriamo la differenza percentuale tra i due valori, notiamo che con il numero di step aumenta.

1	0.00000000	1.00000000	1.00000000	0.0000000000000000
2	5.05050495E-02	1.00000000	0.998725414	1.2753985262538615E-003
3	0.101010099	0.997449219	0.994911492	2.5474577387734542E-003
4	0.151515156	0.992360711	0.988587201	3.8098026911075598E-003
5	0.202020198	0.984766901	0.979800701	5.0557682444914405E-003
6	0.252525240	0.974719286	0.968618453	6.2787167481970505E-003
7	0.303030312	0.962287903	0.955124378	7.4720754618444437E-003
8	0.353535354	0.947560489	0.939419031	8.6290897062454210E-003
9	0.404040396	0.930641472	0.921618104	9.74309923652390162E-003
10	0.454545438	0.911650717	0.901851177	1.0807311968981820E-002
11	0.505050480	0.890722096	0.880259991	1.1815032693025088E-002

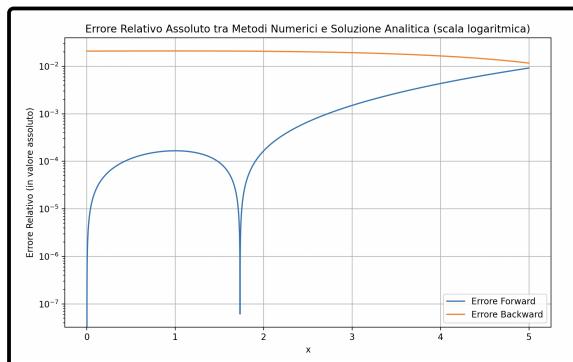
Aumenta al punto che nei pressi del 100 esimo step si raggiunge il 90% di differenza percentuale. Plotando gli errori in valore assoluto e in scala logaritmica troviamo



Per il backward si nota che l'errore relativo è sempre maggiore di 1 e si amplifica, con un ordine di grandezza di differenza per  $x = 0$ . Quindi con 100 steps il backward compie un errore di 10. Si nota che aumentando di ordini di grandezza il numero di steps, diminuisce l'errore del backward di altrettanti ordini. Per esempio, per  $10^3$  steps si può osservare



e per  $10^4$  si può apprezzare

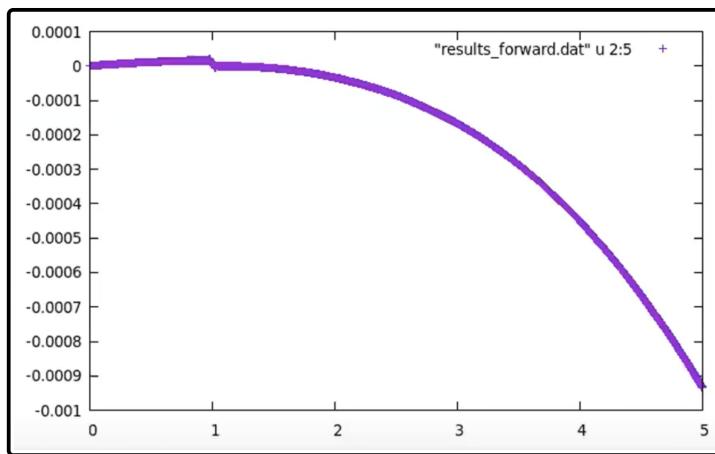


Difatti l'errore scende prima a  $10^{-1}$  e successivamente a  $10^{-2}$ . Ma quali sono le sorgenti di errore?

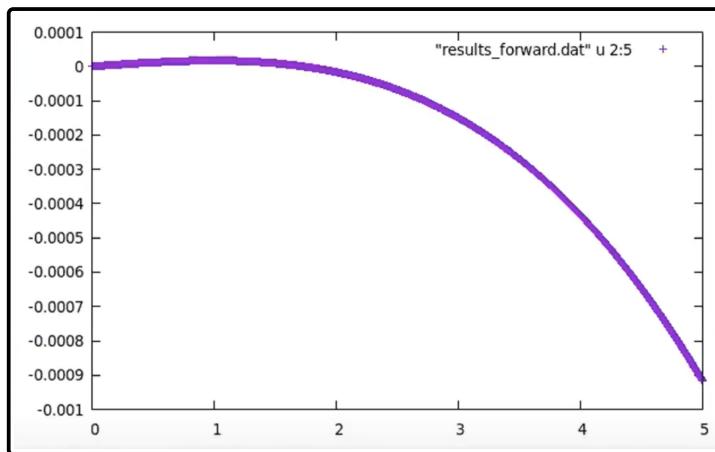
- Innanzitutto l'algoritmo utilizzato per il calcolo della derivata, cioè la derivata forward, che è dotato della precisione minore.

- La precisione **real\*4,8**

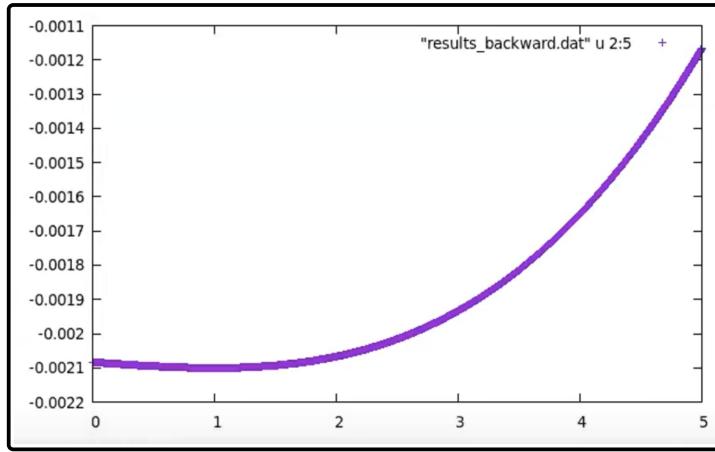
Se vogliamo vedere il problema in dettaglio dobbiamo comprendere come variano i risultati dell'algoritmo per spaziature differenti. Per esempio, se imposto  $10^5$  steps con precisione **real\*4**, mi accorgo di una discontinuità nella derivata della curva dell'errore.



è probabile che sia un'effetto della precisione adottata. Poiché stiamo adottando una spaziatura di  $10^{-5}$  con numeri che hanno una precisione di  $10^{-7}$ , compaiono effetti di roundoff. Se imposto una diversa precisione, posso ottenere risultati differenti. Perciò ripeto l'esecuzione dello script, il plot e ottengo un'immagine distinta.



Se aumento col numero di punti ad un certo punto non riesco a ricostruire la funzione. La spaziatura deve essere ragionevolmente piccola per l'accuratezza del risultato ma anche grande per la precisione punti che stiamo utilizzando. Nonostante questo la precisione finale a fine intervallo è una buona precisione di  $10^{-3}$ .



Nel backward l'errore aumenta fino al  $2 \cdot 10^{-3}$ , eppure a questo punto posso giocare a diminuire la spaziatura finché non arrivo ad una precisione compatibile con la doppia e poi potrei usare precisioni maggiori. Eppure conviene considerare il problema dell'algoritmo della derivata.

## 19.3 Eulero improved

L'Eulero standard è

$$y_{n+1} = y_n + h f(x_n, y_n) + O(h^2)$$

se voglio implementare un algoritmo della derivata più preciso devo utilizzare l'**Eulero improved**. Considero il fatto che

$$y(x_n + h) = y_n + h y'_n + \frac{1}{2} h^2 y''_n + O(h^3)$$

Eppure  $y'_n$  è la derivata calcolata in  $x_n$ , che per l'algoritmo è  $f(x_n, y_n)$ . Possiamo pensare di includere nell'algoritmo  $y''_n$ . Proviamo a scrivere

$$\begin{aligned} y''_n &= \frac{d}{dx} f(x_n, y_n) = \frac{\partial}{\partial x} f + \frac{\partial}{\partial y} f \frac{dy}{dx} \\ &= \frac{\partial}{\partial x} f + f(x_n, y_n) \frac{\partial}{\partial y} f \end{aligned}$$

Possiamo scrivere un algoritmo improved con

$$y_{n+1} = y_n + h f_n + \frac{1}{2} h^2 \left[ \frac{\partial}{\partial x} f_n + f_n \frac{\partial}{\partial y} f_n \right] + O(h^3)$$

Mi aspetto che la convergenza sia migliore perché tra due punti compio un errore più piccolo. Questo algoritmo va sotto il nome di **Eulero Improved**. Dobbiamo inserire informazione analitica in più, come le derivate parziali, che se la  $f$  fosse definita numericamente, sarebbero più difficili da dedurre.

### 19.3.1 Esecuzione dello script

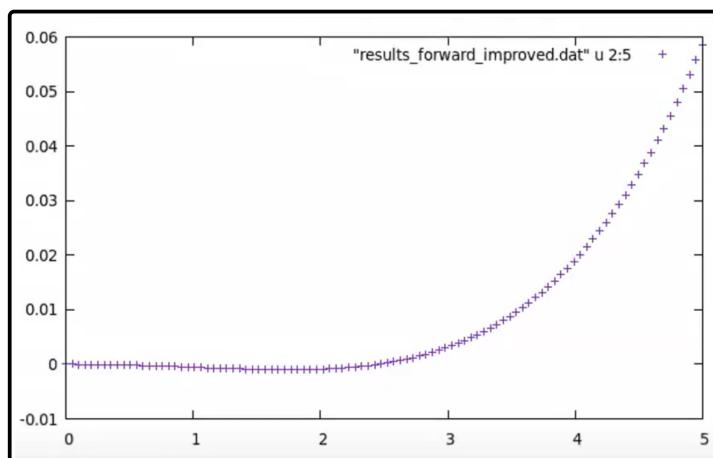
Cominciamo con i 100 steps, come prima. Vediamo che l'errore in questo caso sì cresce, ma è molto più piccolo.

1	0.00000000	1.00000000	1.00000000	0.0000000000000000
2	5.05050495E-02	0.998724639	0.998725414	-7.7584954630108950E-007
3	0.101010099	0.994906604	0.994911492	-4.9125906343017206E-006
4	0.151515156	0.988575161	0.988587201	-1.2179210449521394E-005
5	0.202020198	0.979778469	0.979800701	-2.2691128960925147E-005
6	0.252525240	0.968583167	0.968618453	-3.6429816450954363E-005
7	0.303030312	0.955073535	0.955124378	-5.3232976587100943E-005
8	0.353535354	0.939350367	0.939419031	-7.3095240814555096E-005
9	0.404040396	0.921529710	0.921618104	-9.5916008879895050E-005
10	0.454545438	0.901741505	0.901851177	-1.2161564802104902E-004

Una caratteristica tipica dei metodi di risoluzione delle equazioni differenziali è che l'errore continua ad accumularsi.

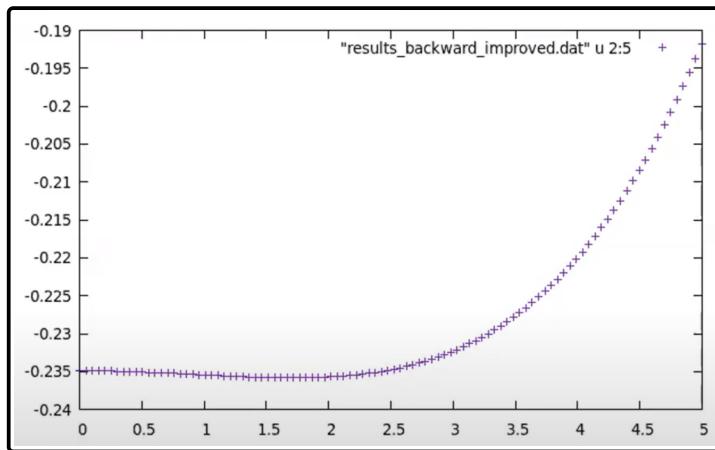
#### Errore Forward

Per il forward con Eulero si raggiungeva un'errore del 90%, ora si raggiunge il 6%.



#### Errore Backward

Per il backward si raggiunge un errore del 23%, eppure bisogna capirne il motivo.



Abbiamo imparato che, come immaginavamo dai metodi numerici per il calcolo delle derivate, quando li portiamo alle equazioni differenziali, ci sono dei problemi che sono critici, perché le ODE sono risolte in modo riscorsivo, calcolando ad ogni step le derivate. La performance degli algoritmi dipende anche dalla funzione che si va a calcolare. Se la funzione è piatta, questo coinvolge differenze di numeri simili, e questo introduce instabilità numeriche maggiori. Abbiamo poi imparato che i parametri importanti sono

- La spaziatura  $h$ , deve essere ragionevole se rapportata alla precisione della rappresentazione numerica
- La precisione della rappresentazione numerica
- La precisione analitica dell'algoritmo. Introducendo termini aggiuntivi si riesce a raggiungere valori molto più affidabili.
- La precisione dell'algoritmo di stima della derivata. Potrei utilizzare algoritmi più performanti ed ottenere metodi di risoluzione delle ODE più precisi. Un esempio è il metodo di Runge Kutta.

Alla fine si fa un'applicazione dei metodi di risoluzione delle ODE per il problema del pendolo con il termine forzante e smorzante.

# Chapter 20

## Lezione 20

### 20.1 Metodo di Eulero

Abbiamo studiato le ODE di I ordine, che anche se di ordine superiore si possono riportare ad un sistema di eq. del primo ordine. Successivamente abbiamo studiato dei problemi di Cauchy, cioè con condizioni iniziali e abbiamo studiato il metodo di Eulero.

$$\frac{dy}{dx} = f(x, y)$$

Si utilizza l'algoritmo della derivata forward per l'approssimazione della derivata prima. Suddividiamo l'intervallo di definizione del problema  $[a, b]$  in  $N$  intervalli di estensione  $h$ . Il numero di punti che utilizziamo è un altro parametro critico. La precisione numerica con cui si lavora e la precisione analitica con la quale passiamo dalla forma continua alla forma discretizzata sono altri parametri fondamentali. Abbiamo studiato i casi

- $$\begin{cases} \frac{dy}{dx} = -y \\ y(0) = 1 \end{cases} \Rightarrow y = e^{-x}$$

Sull'intervallo  $[0, a]$ . Abbiamo visto che quando facciamo l'evoluzione, un parametro di quanto sia reliable l'algoritmo è verificare che la backward propagation non restituiscia un punto iniziale troppo distante dal valore specificato dal problema.

- $$\begin{cases} \frac{dy}{dx} = -xy \\ y(0) = 1 \end{cases} \Rightarrow y = e^{-\frac{1}{2}x^2}$$

In questo caso c'è un errore del 90% con l'algoritmo standard.

Abbiamo introdotto dunque lo sviluppo di Taylor al second'ordine della soluzione. Questo per costruire l'algoritmo di Eulero Improved, che approssima la soluzione con un errore di  $h^3$ .

$$y_{n+1} = y_n + hf_n + \frac{1}{2}h^2 \left[ \frac{\partial}{\partial x} f_n + f_n \frac{\partial}{\partial y} f_n \right] + O(h^3)$$

## 20.2 Algoritmi Multistep

### 20.2.1 Algoritmo integrale

Un'altra classe di algoritmi che permette di risolvere ODE è quella degli Algoritmi Multistep. Il nome suggerisce l'algoritmo

$$\begin{aligned}\frac{dy}{dx} &= f(x, y) \\ \int_{x_n}^{x_{n+1}} \frac{dy}{dx} dx &= \int_{x_n}^{x_{n+1}} f(x, y) dx \\ y_{n+1} - y_n &= \int_{x_n}^{x_{n+1}} f(x, y) dx\end{aligned}$$

L'espressione è una soluzione integrale dell'ODE. Il problema è che noi pensiamo di partire dalla condizione iniziale e sapendo tutto della funzione al passo  $n$ -esimo, ricaviamo tutto del passo  $n + 1$ -esimo. Eppure in questo caso abbiamo un integrale in  $x$  sul dominio  $[x_n, x_{n+1}]$ . Facciamo uno sviluppo della funzione all'interno dell'integrale grazie all'interpolazione di Lagrange.

$$f(x, y) = \frac{x - x_n}{x_{n+1} - x_n} f_{n+1} + \frac{x - x_{n+1}}{x_n - x_{n+1}} f_n + O(h^2)$$

Se si pone  $x_n$  si trova  $f_n$  e se si pone  $x_{n+1}$ , produce  $f_{n+1}$ . In questa formulazione si trascurano ordini  $h^2$ . Utilizziamo come input l'interpolazione nell'integrale e ricaviamo la soluzione

$$y_{n+1} = y_n + \frac{f_{n+1}}{2h} h^2 + \frac{f_n}{2h} h^2$$

Utilizziamo le quantità calcolate allo step  $n$ -esimo per ricavare le quantità allo step  $n + 1$ -esimo. Passando dalla funzione alla sua approssimazione, non abbiamo risolto il problema perché poi abbiamo integrato su  $[x_n, x_{n+1}]$ . Si può reintrodurre questa approssimazione tra i passi  $(n - 1, n)$ . Questo, nel limite in cui  $h$  diventa sufficientemente piccolo, diventa una buona approssimazione.

$$f(x, y) = \frac{x - x_{n-1}}{h} f_n - \frac{x - x_n}{h} f_{n-1} + O(h^2)$$

Inserendo questa approssimazione della funzione nell'integrale, si ottiene la seguente espressione

$$y_{n+1} = y_n + h \left( \frac{3}{2} f_n - \frac{1}{2} f_{n-1} \right) + O(h^3)$$

Non dobbiamo conoscere le derivate della funzione, il che in determinati casi non analitici può significare un grande vantaggio. L'informazione in aggiunta in questo caso, difatti, è acquisita dallo step  $n - 1$ . È una differenza e poi vediamo quali conseguenze abbia. Per risolvere un problema con la condizione iniziale

$$y(0) = 1$$

dobbiamo far partire l'algoritmo con  $y_0, y_1$ . Come possiamo stimare nel mentre il punto  $y_1$ , per poter poi ricavare il valore di  $y_2$  con l'algoritmo multistep? Lo possiamo conoscere direttamente con uno sviluppo di Taylor sul primo punto, come si fa con l'algoritmo Eulero Improved.

$$y_{n+1} = y_n + h f + \frac{1}{2} h^2 \left[ \frac{\partial^2}{\partial x^2} f + f \frac{\partial}{\partial y} f \right]$$

Lo applichiamo per  $n = 0$  e ricaviamo  $y_1$ . In questo caso abbiamo perciò  $y_0, y_1$ , che poi non necessita la stima delle derivate. Questo è un algoritmo che verifichiamo sulle nostre funzioni.

## 20.2.2 Derivata centrale

Un altro algoritmo fa utilizzo della precisione con la quale sappiamo scrivere la derivata. Possiamo per esempio utilizzare l'algoritmo della derivata centrale.

$$\begin{aligned}\frac{dy}{dx} &= f(x, y) \\ \frac{y_{n+1} - y_{n-1}}{2h} &= f(x_n, y_n) \\ y_{n+1} &= y_{n-1} + 2hf(x_n, y_n) + O(h^3)\end{aligned}$$

In modo naturale emerge un algoritmo multistep nel quale si coinvolgono i punti  $n, n-1, n+1$ . La precisione è localmente di  $O(h^3)$ . Possiamo verificare sui problemi di Cauchy

$$\begin{cases} \frac{dy}{dx} = -y \\ y(0) = 1 \end{cases} \Rightarrow y = e^{-x}, \quad \begin{cases} \frac{dy}{dx} = -xy \\ y(0) = 1 \end{cases} \Rightarrow y = e^{-\frac{1}{2}x^2}$$

Proviamo a verificare per entrambi cosa succede.

## 20.3 Applicazione algoritmi

### 20.3.1 Multistep integrale

#### Gaussiana

Si specificano perciò gli algoritmi per questi due casi. Nel caso dell'algoritmo integrale e del problema Gaussiamo scriviamo

$$\begin{aligned}y_{n+1} &= y_n + h\left(\frac{3}{2}f_n - \frac{1}{2}f_{n-1}\right) + O(h^3) \\ f_n &= -x_n y_n \\ y_{n+1} &= y_n + h\left(-\frac{3}{2}x_n y_n + \frac{1}{2}x_{n-1} y_{n-1}\right)\end{aligned}$$

Come dati di input abbiamo lo step 0, con  $x_0, y_0$  e dobbiamo ricavarci  $y_1$ . Possiamo eseguire questo primo step con l'algoritmo di Eulero Improved. Perciò

$$\begin{aligned}y_1 &= y_0 + h(-x_0 y_0) + \frac{1}{2}h^2 \left[ -y_0 + (-x_0 y_0)(-x_0) \right] \\ &= y_0 - h(x_0 y_0) + \frac{1}{2}h^2 \left[ -y_0(1 - x_0^2) \right]\end{aligned}$$

Impostando un intervallo  $[a, b]$  posso scrivere  $x_0 = 0, y_0 = 1$  e ricavo lo step successivo con l'algoritmo di Eulero Improved.

$$y_1 = 1 - \frac{1}{2}h^2$$

#### Esponenziale

### 20.3.2 Multistep Der. Centrale

#### Gaussiana

Vediamo le prestazioni dell'algoritmo in precisione **real\*4** con  $10^3$  steps.

1	0.00000000	1.00000000	1.00000000	0.0000000000000000
2	6.00600615E-03	0.994012058	0.994011998	5.9963707565632974E-008
3	1.20120123E-02	0.99928296	0.988059819	1.1940188959846557E-002
4	1.80180185E-02	0.993867755	0.982143342	1.1866747924798663E-002
5	2.40240246E-02	0.999713182	0.976262271	2.3736034060494880E-002
6	3.00300308E-02	0.993579268	0.970416367	2.3587527670394088E-002
7	3.60360369E-02	0.999354780	0.964605510	3.5386936959165685E-002
8	4.20420421E-02	0.993146658	0.958829463	3.5161492093554549E-002
9	4.80480492E-02	0.998853207	0.953087986	4.6892006691067037E-002
10	5.40540554E-02	0.992570162	0.947380900	4.6588042674806859E-002

Quello che emerge dall'applicazione dell'algoritmo forward è che nel primo punto non c'è differenza e nel secondo c'è una differenza piccolissima, dovuta all'algoritmo Eulero Improved. Dal terzo punto in poi si vedono gli effetti dell'algoritmo Multistep Integrale, che genera un errore percentuale. Dopodiché si deteriora notevolmente. Riproviamo con la doppia precisione con gli stessi steps.

1	0.0000000000000000	1.0000000000000000	1.0000000000000000	0.0000000000000000
2	6.000000000000000E-003	0.9940120304486615	0.9999816410857654	-0.0000000000000000
3	1.201201201201201E-002	0.99992828778111698	0.9999799799960003	4.2942595523160575E-007
4	1.801801801801801E-002	0.99386775196273314	0.9998376868625786	-5.987851061673370E-003
5	2.402402402402402E-002	0.9997131823593209	0.99971146476914508	1.7180760370175519E-006
6	3.003003003003003E-002	0.99357925786651768	0.99954920028893501	-5.9905244922770625E-003
7	4.603603603603603E-002	0.99935477736653633	0.99935091280170119	3.8670674268385143E-006
8	4.204204204204204E-002	0.99314667186327488	0.9991162375719501	-5.9911354525716478E-003
9	4.804804804804804E-002	0.998532288482953	0.99884635849597880	6.8782638733588103E-006
10	5.405405405405405E-002	0.99257017999621588	0.99854014624485765	-5.9966202474698565E-003

Stavolta riproviamo tra  $[0, 1]$ .

1	0.0000000000000000	1.0000000000000000	1.0000000000000000	0.0000000000000000
2	1.0010010010010010E-003	0.9989995000000000	0.9999994989862350	-1.0004997487474670E-003
3	2.0020020020020020E-003	0.9980000000000000	0.9999799799960003	2.0030040001152989E-009
4	3.003003003003003E-003	0.99890549199650894	0.999954999664753	-1.005637627946285E-003
5	4.004004004004004E-003	0.9999199202009625	0.9999198400899619	8.0120642559412982E-009
6	5.005005005005005E-003	0.99898747603666904	0.99998747504088792	-1.000511799854973E-003
7	6.006006006006006E-003	0.9998198213557676	0.99998196410857654	1.8827325202297125E-008
8	7.0070070070070069E-003	0.9999754523273847	0.99997545123772701	-1.00051283321520990E-003
9	8.0080080080080079E-003	0.99996796846592007	0.99996793641791948	3.249027679724561E-008
10	9.00909090909089E-003	0.99895942068273547	0.99995941970174040	-1.0005398906666927E-003

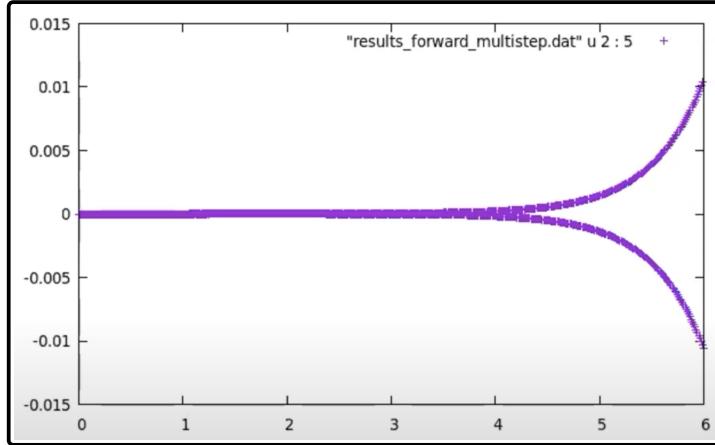
Si nota che ha un comportamento molto più accettabile di prima. Questo perché  $e^{-\frac{1}{2}x^2}$ , non appena  $x \sim 10$ , diventa molto piccolo. Siccome noi stiamo ricostruendo la funzione con l'algoritmo, non appena la gaussiana diventa piccola, noi stiamo facendo delle differenze tra numeri piccoli e la perdita di precisione diventa molto grande. Perciò si riscontra questa difficoltà nel ricostruire la Gaussiana a partire dalla sua ODE. Se vogliamo ampliare l'intervallo di integrazione della nostra ODE, dobbiamo disporre di una precisione numerica maggiore.

### Esponenziale

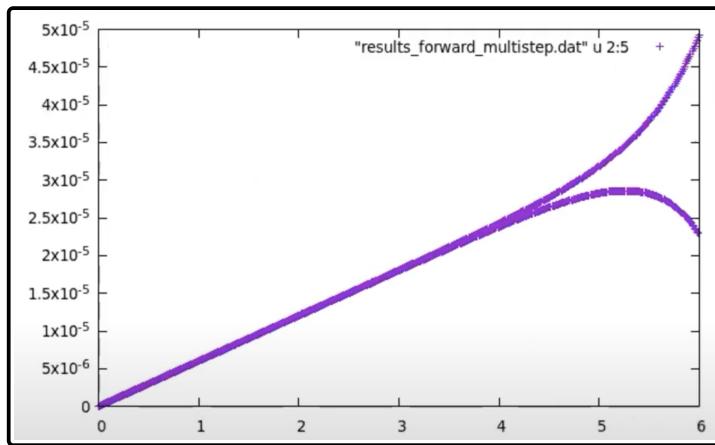
Proviamo l'algoritmo della derivata centrale per l'esponenziale negativo.

1	0.0000000000000000	1.0000000000000000	1.0000000000000000	0.0000000000000000
2	1.0010010010010010E-003	0.9989995000000000	0.99899949983337510	1.672932456583781E-010
3	2.0020020020020020E-003	0.9980000000000000	0.998000006733374	3.3433487547478442E-010
4	3.003003003003003E-003	0.99700156200049897	0.99700156150037448	5.0162862994005280E-010
5	4.004004004004004E-003	0.99600400199799599	0.99600400133199818	6.6866981592568307E-010
6	5.005005005005005E-003	0.99500749999449700	0.99500749916270648	8.3596407634977092E-010
7	6.006006006006006E-003	0.9940119949100001	0.9940119939400120	1.0030040815195339E-009
8	7.0070070070070069E-003	0.99301748599051098	0.99301748482838315	1.1702994715408750E-009
9	8.0080080080080079E-003	0.99282397199602399	0.99202397066935089	1.3373976501045956E-009
10	9.00909090909089E-003	0.991031450521254995	0.991031450521240940	1.5046349361240952E-009

Notiamo una precisione molto maggiore. Sulla quinta colonna abbiamo 3 ordini di grandezza migliori e notiamo la caratteristica che prima c'erano dei segni alterni che comparivano e ora non compaiono più. Potremmo dunque estenderci con l'intervallo, tornando a  $[0, 6]$ . Raggiungiamo una precisione di  $10^{-7}$  dopo 1000 punti. Per avere un'affidabilità del risultato dobbiamo fare questi studi. Dobbiamo vedere l'altro algoritmo con la stessa precisione con la derivata centrale.



Se plottiamo l'errore di questo algoritmo vediamo una divergenza, anche se meno severa che nel caso della gaussiana. Vediamo dei valori che cambiano di segno e come oscilla, continua a crescere molto rapidamente, un comportamento che non si vedeva con l'algoritmo di Eulero Improved. A seconda del tipo di problema di Cauchy che affrontiamo e dell'ODE, ci può essere questo problema. Lo possiamo spostare con la precisione numerica.



## 20.4 Stabilità Soluzione

Si nota l'amplificazione lineare dell'errore, caratteristica che fa parte degli algoritmi multistep, che coinvolgono 3 punti. La conoscenza della funzione in  $n, n - 1$  per trovare  $n + 1$ .

$$y_{n+1} = y_{n-1} + 2hf_n$$

Si tratta di un'equazione ricorsiva. Un'assunzione ricorsiva generica di questa equazione è che  $y_n = Ar^n$  e proviamo a sostituire al posto degli  $y$  la legge di potenza e a specificare l'algoritmo per il problema dell'esponenziale.

$$Ar^{n+1} = Ar^{n-1} - 2hAr^n$$

Possiamo semplificare sia la costante  $A$  che  $r^{n-1}$ , ottenendo

$$\begin{aligned} r^2 &= 1 - 2hr \\ r^2 + 2hr - 1 &= 0 \\ r &= -h \pm \sqrt{h^2 + 1} \end{aligned}$$

Perciò otteniamo due soluzioni distinte

$$\begin{aligned} r_1 &= -h + \sqrt{h^2 + 1} \\ r_2 &= -h - \sqrt{h^2 + 1} \end{aligned}$$

Questi sono i due valori di  $r$  per i quali l'algoritmo ricorsivo presenta le soluzioni naturali. Il modulo di  $r_1$  è minore di 1, mentre  $r_2 < 0$ , con  $|r_2| > 1$ .

$$y_n \sim (-1)^n (1 + h)^n$$

Riprendendo il discorso delle successioni numeriche affrontato alla Lezione 2 a pagina 10. Andando a selezionare la soluzione iniziale, scegliamo una delle due soluzioni possibili e sempre a causa dei round-off, non possiamo evitare che la soluzione generale sia una combinazione lineare delle due soluzioni. Perciò non possiamo far sì che la soluzione che adottiamo coincida con una delle due e non con una combinazione di queste. La soluzione che tende a divergere, prima o poi la ritroviamo nella soluzione numerica. I plot divergenti sono infatti la manifestazione di un comportamento di questo tipo. Possiamo cercare di mitigare questi effetti migliorandola, ma questi sono una caratteristica dell'algoritmo.

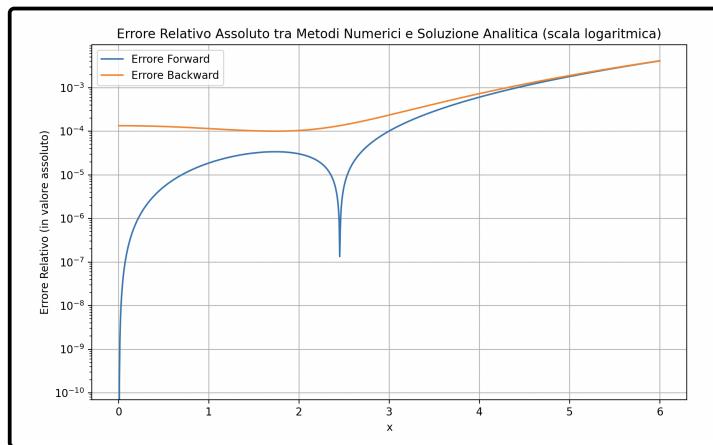
### 20.4.1 Analisi di Stabilità di Von Neumann

Si procede a condurre un'analisi d'instabilità. Questo problema si rivede anche nella risoluzione delle PDE e dovendo derivare su più variabili, il problema diventa più forte. Si deve introdurre un metodo quindi per determinare se la soluzione può essere stabile oppure no. Questo dipende dall'algoritmo e dall'equazione. Se scriviamo

$$\frac{dy}{dx} = -xy$$

Con la legge di potenza si trovano degli  $r_{1,2}$  tali che  $|r_{1,2}| < 1$ , perciò non mostrano delle instabilità. Dipende dunque dall'equazione che si sta esaminando. Si presenta il problema numerico per la differenza di numeri piccoli. E' un problema di cui non risente l'algoritmo di Eulero Improved e quindi se faccio l'analisi della

stabilità di Von Neumann emerge un'equazione lineare, che per il teorema fondamentale di Gauss ha una soluzione unica. Si può applicare questo tipo di analisi anche all'algoritmo multistep integrale e passando all'approssimazione che si è fatta. Si trova anche in quel caso  $|r_{1,2}| < 1$ . Difatti il suo errore è contenuto e non oscillante.



Abbiamo visto i problemi che si presentano per le ODE. Prossima lezione si studia l'algoritmo di Runge Kutta al 4 ordine e proviamo ad applicarlo ad un oscillatore con termine smorzante e forzante.

# Chapter 21

## Lezione 21

### 21.1 Metodo di Runge Kutta

#### 21.1.1 Algoritmo del II ordine

Esistono diversi tipi di questo metodo, specialmente al quarto ordine, che è il metodo che va per la maggiore. Si fa l'applicazione all'equazione di un pendolo con termine forzante e termine smorzante. Partiamo da un problema del tipo

$$\frac{dy}{dt} = f(t, y(t))$$

Si riconduce sempre alle differenze finite, però l'idea è di evitare di calcolare derivare come nel caso del metodo di Eulero Improved, perché questo implica conoscere la funzione in più punti localmente, oppure averne una rappresentazione analitica e quindi delle sue derivate. L'idea è di costruire una black-box universale che abbia delle prestazioni migliori.

$$\begin{aligned} y(t_0 + h) &= y(t_0) + hy' + \frac{1}{2}h^2y'' \\ &= y(t_0) + hf + \frac{1}{2}h^2\left(f \frac{\partial}{\partial t} f + f \frac{\partial}{\partial y} f\right) \end{aligned}$$

Possiamo dunque scrivere

$$y(t_0 + h) = y(t_0) + hf(y_0, t_0) + \frac{1}{2}h^2\left(\frac{\partial}{\partial t} f + f \frac{\partial}{\partial y} f\right)\Big|_{t=t_0} + O(h^3)$$

Stiamo valutando la  $f$  tra punti shiftati di poco nella nostra griglia. A questo punto, però, ci chiediamo come trovare questi  $\alpha, \beta, \delta, \gamma$  per far coincidere questa espressione con quella precedente fino all'ordine  $h^2$ . Per fare questo facciamo lo sviluppo di  $f$  con l'idea di andare a confrontare i coefficienti dei vari termini con l'espansione che abbiamo individuato. In seguito utilizzeremo l'espressione *mediata* e non quella che comprende il calcolo di derivate. L'idea è appunto di scrivere la funzione come combinazione di termini della funzione in punti vicini. Sempre prendendo la funzione senza doverne calcolare le derivate. Perciò

$$y(t_0 + h) = y(t_0) + h[\alpha f(t_0, y(t_0)) + \beta f(t_0 + \gamma h, y_0 + \delta h f_0)]$$

Facciamo sì che l'espressione di  $y(t_0 + h)$  sia uguale a quella che abbiamo ottenuto con lo sviluppo. Per fare questo scriviamo lo sviluppo di taylor della  $f$

$$f(t_0 + \gamma h, y_0 + \delta h f_0) = f(t_0, y_0) + \gamma h \frac{\partial}{\partial t} f\Big|_{t_0, y_0} + \delta h f(t_0, y_0)$$

Riscrivo l'espressione *mediata* sostituendo lo sviluppo della funzione.

$$y(t_0 + h) = y(t_0) + h\left\{\alpha f(t_0, y(t_0)) + \beta\left[f(t_0, y_0) + \gamma h \frac{\partial}{\partial t} f\Big|_{t_0, y_0} + h \delta f(t_0, y_0) \frac{\partial}{\partial y} f\Big|_{t_0, y_0}\right]\right\}$$

Questo ci dà

$$y(t_0 + h) = y(t_0) + h(\alpha + \beta)f(t_0, y_0) + h^2 \left[ \beta \left( \gamma \frac{\partial}{\partial t} f \right)_{t_0, y_0} + \delta f(t_0, y_0) \frac{\partial}{\partial y} f \Big|_{t_0, y_0} \right]$$

Confronto ora questa espressione con lo sviluppo fatto prima, uguagliando i coefficienti delle potenze di  $h$ .

$$\begin{cases} \alpha + \beta = 1 \\ \beta \gamma = \frac{1}{2} \\ \beta \delta = \frac{1}{2} \end{cases}$$

E' un sistema di 3 equazioni in 4 incognite che ha  $\dim = 1$ , perciò abbiamo un insieme di soluzioni grande quanto vogliamo. Possiamo cercare la combinazione di valori che minimizza il coefficiente dell'errore con ordine  $O(h^3)$ . Se prendiamo  $\gamma = \delta = 1$ , ci dà

$$\begin{aligned} \beta &= \frac{1}{2} \\ \alpha &= \frac{1}{2} \end{aligned}$$

come possibile soluzione. Un'altra che solitamente minimizza lo scarto è

$$\begin{aligned} \gamma &= \frac{3}{4} \\ \beta &= \frac{2}{3} \end{aligned}$$

### 21.1.2 Algoritmo del IV ordine

Questo è l'algoritmo al II ordine. Questa cosa si può spingere a ordini successivi, anche se bisogna fare un po' di lavoro in più con le derivate.

$$y(t_0 + h) = y(t_0) + \alpha_1 k_1 + \alpha_2 k_2 + \cdots + \alpha_n k_n$$

$$\begin{cases} k_1 = hf(t, y) \\ k_2 = hf(t + \nu_{21}h, y + \nu_{21}k_1) \\ \dots \\ k_n = hf(t + h \sum_{l=1}^{n-1} m\nu_{nl} k_l, y + \sum_{l=1}^{n-1} m\nu_{nl} k_l) \end{cases}$$

L'idea di costruire i coefficienti aggiungendo termini. Nel caso 2 il coefficiente  $\nu_{21} = 1$ . Abbiamo visto l'espressione esplicita per l'ordine  $n$ , eppure nel metodo di Runge Kutta Fellberg si intende che si utilizzino  $n$  coefficienti di questo tipo. Più si va avanti con i coefficienti più si fa un lavoro analitico, eppure una volta concluso il lavoro l'algoritmo è universale. Maggiore è l'ordine che si utilizza, maggiore è la lentezza dell'algoritmo, sebbene garantisca precisioni migliori. Quello maggiormente utilizzato è  $n = 4$ . C'è un legame tra questa forma e il metodo integrale, dove avevamo un'espressione di questo tipo, dove andavamo a sostituire la funzione nell'integrando e andavamo a valutarla in alcuni punti. Si può vedere in quella forma tanto che i coefficienti che si utilizzano per Runge Kutta sono parenti stretti dei coefficienti che abbiamo ricavato per i metodi di risoluzione degli integrali deterministici. Proviamo ad esaminare degli esempi per sperimentare l'efficienza dell'algoritmo.

### 21.1.3 Script

Viene chiamata la subroutine **rungekutta4** per stimare i valori della  $y2$  per l'algoritmo del second'ordine e di  $y4$  per l'algoritmo del IV ordine. In input abbiamo  $xi$  e  $yi2$ , con la spaziatura  $h$ . Scriviamo i valori di  $\alpha_1, \alpha_2, \nu_{21}$ . In seguito dobbiamo produrre lo step successivo. Lo step include l'espressione della  $yi = \sum_{i=1}^{2,4} \alpha_i k_i$ , perciò mi ricavo i vari valori di  $k_i$ , ovvero valori della funzione in punti pre-determinati. Si deduce che la formulazione dell'algoritmo del IV ordine è

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

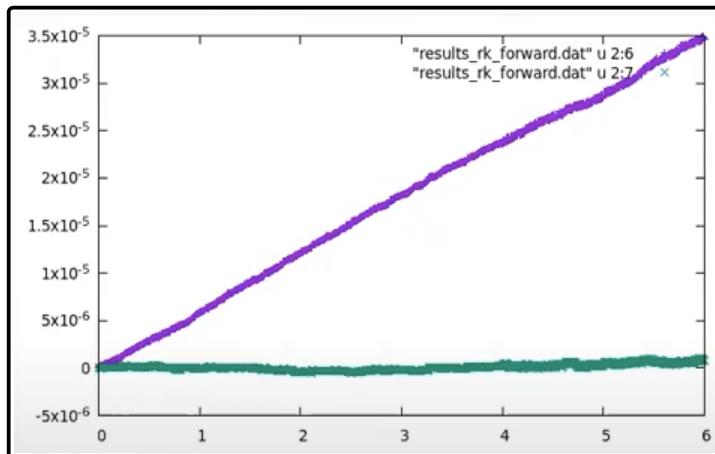
C'è una somiglianza molto forte con i pesi che avevamo nell'algoritmo di Sympson. A questo punto abbiamo la stima del II e IV ordine.

### Esponenziale con Runge Kutta

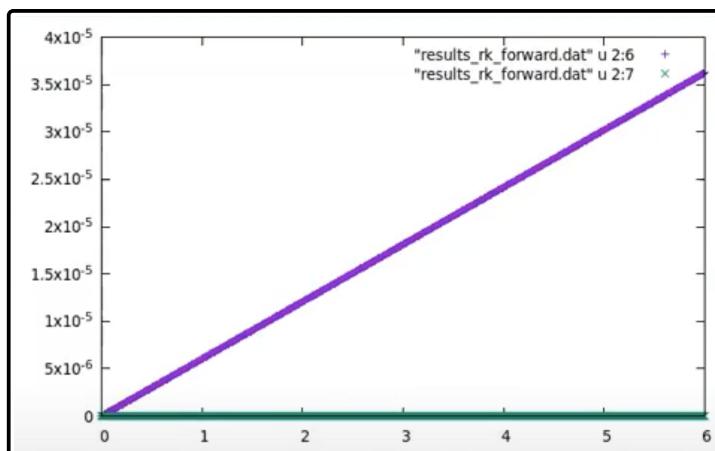
Possiamo provare a fare qualche esercizio, prima dell'esempio dell'equazione del pendolo. Impostiamo il numero di steps a  $10^3$ .

1	0.00000000	1.00000000	1.00000000	1.00000000	0.00000000	0.00000000
2	6.00600615E-03	0.994012858	0.994011998	0.994011998	5.99637104E-08	0.00000000
3	1.20128123E-02	0.988059938	0.988059810	0.988059810	1.20649858E-07	0.00000000
4	1.88186185E-02	0.982143521	0.982143283	0.982143342	1.8264994E-07	-6.06883361E-08
5	2.40246246E-02	0.976262510	0.976262212	0.976262271	2.44215670E-07	-6.10539317E-08
6	3.08366308E-02	0.970416665	0.970416367	0.970416367	3.07108564E-07	0.00000000
7	3.68366369E-02	0.964695510	0.964695510	0.964695510	3.08958590E-07	0.00000000
8	4.28426431E-02	0.958829761	0.958829463	0.958829463	3.10619786E-07	0.00000000
9	4.88486492E-02	0.953087986	0.953087986	0.953087986	3.12692180E-07	0.00000000
10	5.40540554E-02	0.947380988	0.947380988	0.947380988	3.14575885E-07	0.00000000

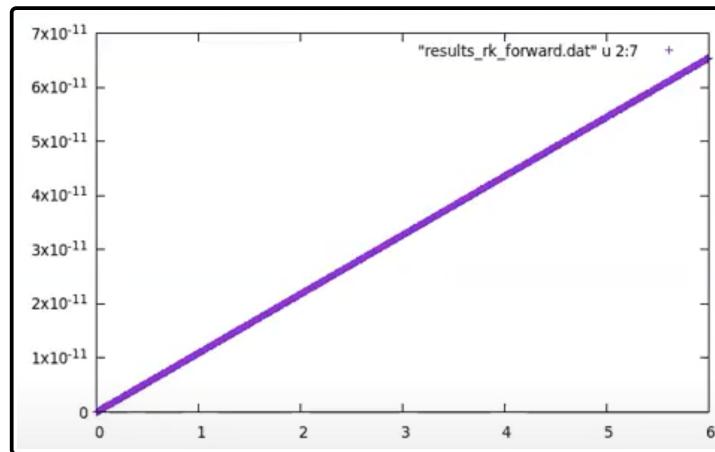
Si vede che RK II produce errori molto piccoli e addirittura RK IV produce l'errore 0 per la precisione **real\*4**. Con delle precisioni di  $10^{-8}$  è facile trovare lo 0. Aumentando le iterazioni l'errore si propaga, anche se piuttosto contenuto. RK IV è piuttosto oscillante, ma bisognerebbe testare l'algoritmo con una precisione maggiore. RK IV sulla carta, rispetto al RK II ha due ordini di grandezza in più di precisione, mentre nella pratica ha poco meglio di un ordine di grandezza in più. Plottiamo gli errori



Si può vedere che l'algoritmo del II ordine incrementa linearmente, mentre il IV ordine è pressocché stabile sullo 0. Mi aspetto un guadagno migliore passando con una doppia precisione, passando dal II al IV ordine.



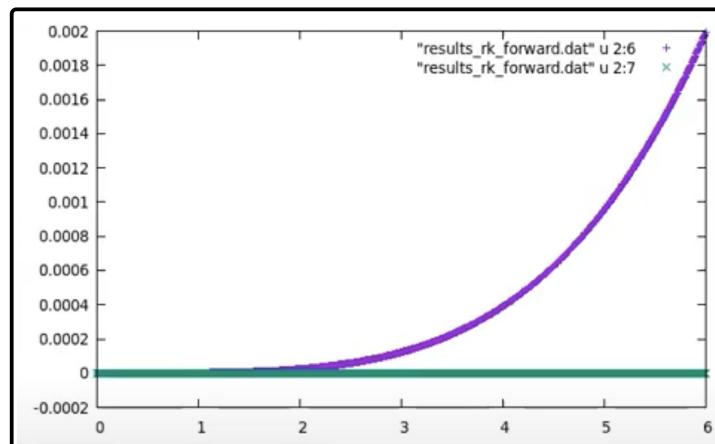
Notiamo che l'algoritmo produce un errore molto più piatto, ovvero con meno oscillazioni rispetto alla precisione **real\*4**.



Chiaramente anche il metodo del IV genera un errore crescente con le iterazioni, ma molto più piccolo.

### Gaussiana con Runge Kutta

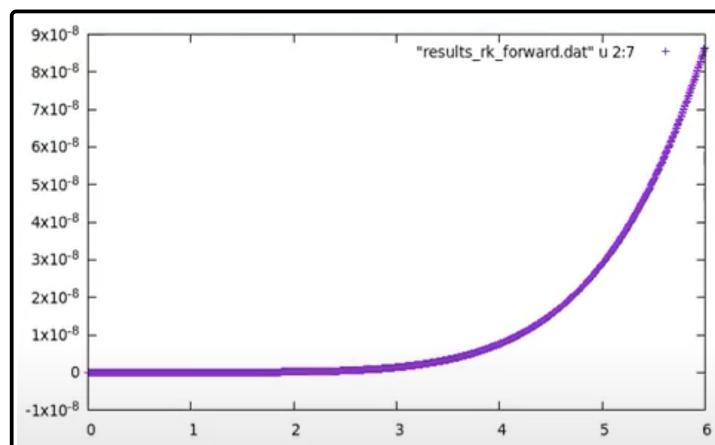
Per valori molto piccoli della funzione si perde facilmente la precisione numerica, per differenze tra numeri molto vicini. Se utilizzassimo algoritmi di ordine maggiore, non potremmo risolvere errori più piccoli in base alla precisione adottata. Visualizziamo dunque gli errori in questo caso.



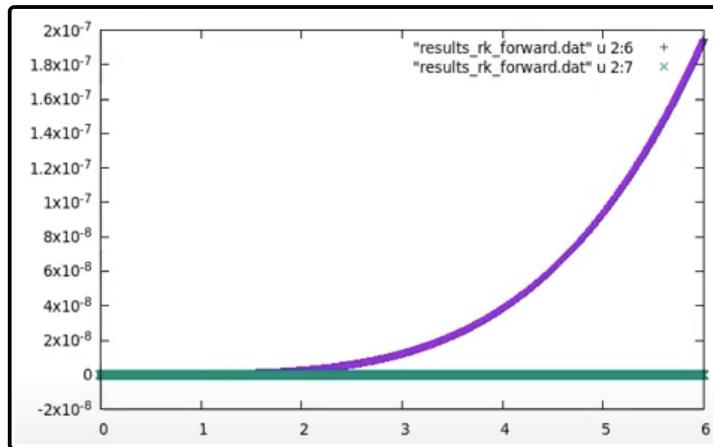
Notiamo che l'errore cresce in modo quadratico. Questo emerge dal fatto che stiamo costruendo la funzione

$$e^{-\frac{1}{2}x^2}$$

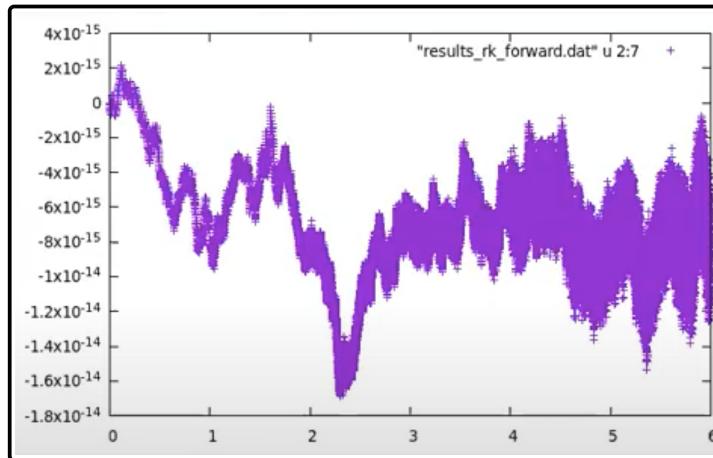
che progressivamente diventa esponenzialmente piccola. Noi abbiamo stabilito che l'algoritmo deve arrivare fino a 6, perciò dobbiamo lavorare con numeri piccoli e con derivate piccole. Le differenze per costruire il punto successivo mi fanno perdere maggiore precisione. I punti vicini rispetto a punti lontani mi fanno perdere maggiore precisione. Difatti, vedo dal grafico che allontanandomi da  $\emptyset$ , incrementa non linearmente. Se visualizzo l'errore della doppia precisione in scala



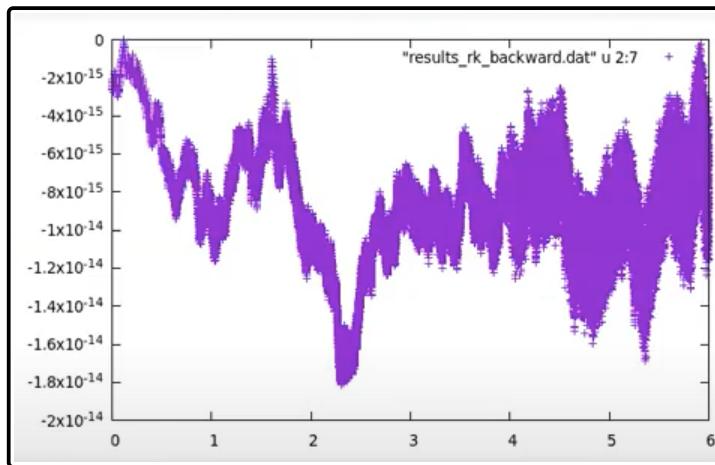
riesco a vedere che il problema si pone tale e quale su scale più piccole. Proviamo a visualizzare l'andamento dell'errore per numeri di punti più elevati, come  $10^5$ .



Si vede che aumentando i punti da  $10^3$  a  $10^5$  abbiamo un incremento di  $10^{-5}$  e mi fa guadagnare anche su precisione **real\*4**. Prima con l'algoritmo del II ordine arrivavo ad un errore di  $10^{-3}$ , ora invece arrivo a  $10^{-7}$ . Diminuire l'incremento in questo caso è stato molto utile. Il comportamento generalmente è dello stesso tipo, più mi allontano, più l'errore aumenta in modo non lineare. Il problema sta sia nella funzione che assume valori piccoli, che nelle sue derivate. Vado a costruire il punto successivo con una combinazione lineare di punti molto vicini tra loro. In punti molto vicini la funzione è vicina e facendo la differenza di punti molto vicini si può perdere precisione. Bisognerebbe provare con una ODE che mi costruisce il *seno* o il *coseno*.



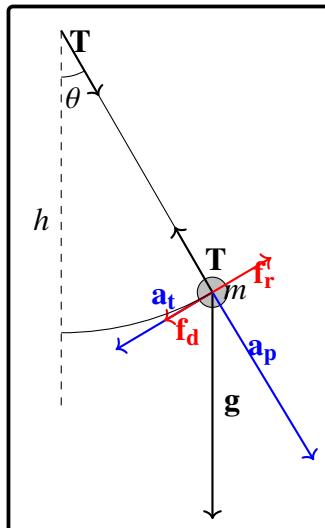
Con la doppia precisione e l'algoritmo del IV ordine cominciamo a vedere il rumore attorno a  $10^{-15}$ . Sono dominato dalla precisione con cui sto lavorando. Ho errori sulla scala del  $10^{-14/15}$ . Sta vincendo il rumore introdotto dalla precisione numerica. All'interno dello studio del path-integral per costruire strumenti finanziari derivati più elaborati, trovo l'esigenza di tenere sotto controllo la situazione completa con errori molto piccoli. Se visualizzo l'errore backward con  $10^5$  punti con l'algoritmo del IV ordine mi trovo sempre sulla stessa scala. E' un comportamento molto buono.



Nel secondo ordine sto lavorando sempre con doppia precisione e mi garantisce una precisione di  $h^3$ , quindi se ho  $10^5$  punti, ho un'ampiezza di ordine  $10^{-5}$ , perciò  $h^3 \sim 10^{-15}$ . Con la doppia precisione ero arrivato ad un errore di  $10^{-7}$ . Quando sto passando da 6 e torno indietro l'errore comincia a diminuire, perché lavoro con una precisione locale molto alta, compatibile con la doppia precisione. Per cui, in questo caso l'algoritmo iterativo guadagna quando la funzione diventa maggiore di  $\emptyset$ , perciò ha meno problemi a fare le operazioni successivamente.

## 21.2 Simulazione del pendolo

Studiamo il moto di un pendolo sottoposto a forza di gravità  $f_g$ , forzante  $f_d$  e smorzante  $f_r$ .



$$\begin{aligned}
 ma &= f_g + f_d + f_r \\
 ml \frac{d^2\theta}{dt^2} &= -mg \sin \theta + f_d(t) + f_r \\
 f_d(t) &= f_d^0 \cos(\omega_0 t) \\
 f_r &= -kv \\
 v &= l \frac{d\theta}{dt}
 \end{aligned}$$

L'equazione differenziale perciò diventa

$$ml \frac{d^2\theta}{dt^2} = -mg \sin \theta + f_d^0 \cos(\omega_0 t) - kl \frac{d\theta}{dt}$$

Nell'ambito della simulazione conviene lavorare con quantità adimensionate, perciò dividiamo ambo i membri per  $ml$ .

$$\frac{d^2\theta}{dt^2} = -\frac{g}{l} \sin \theta + \frac{f_d^0}{ml} \cos(\omega_0 t) - \frac{k}{m} \frac{d\theta}{dt}$$

Chiamiamo i coefficienti  $q = \frac{k}{m}$  e  $b = \frac{f_d^0}{ml}$  e riscriviamo l'equazione

$$\frac{d^2\theta}{dt^2} + q \frac{d\theta}{dt} + \frac{g}{l} \sin \theta = b \cos(\omega_0 t)$$

Ponendo  $g/l = 1$  posso trasformare questa equazione del second'ordine in due equazioni del I ordine che descrivono un pendolo di periodo  $T = 2\pi$ .

$$\begin{cases} \frac{dy_1}{dt} = y_2 \\ \frac{dy_2}{dt} = -qy_2 - \sin y_1 + b \cos \omega_0 t \end{cases}$$

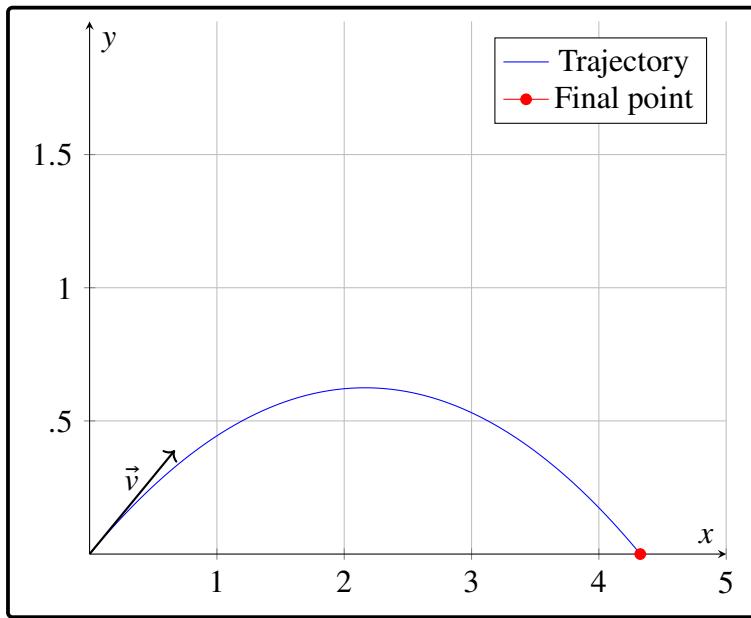
Questo è il sistema implementato nello script **pendulum**. Si possono fare degli esperimenti con lo script per verificare, per esempio, se il diagramma di fase spiraleggia fino a fermarsi nel caso di pendolo smorzato.

# Chapter 22

## Lezione 22

### 22.1 Problema alle condizioni al contorno

Abbiamo curato dei problemi alle condizioni iniziali, per i quali dato un certo numero di informazioni all'inizio, che equivalgono l'ordine del problema, fanno evolvere l'algoritmo sui punti della variabile indipendente e poi si arriva alla soluzione che si deve capire se parente della soluzione analitica. Un altro problema che si presenta anche nelle ODE è un problema con condizioni al contorno. In questo caso, per esempio, come nella meccanica classica, lanciamo un oggetto con angolo  $\neq 0$  rispetto all'orizzontale e l'equazione che governa questo moto è un'equazione oraria del II ordine, che ci permette di studiare il moto del proiettile fino al punto finale  $x_F$ .



Cosa cambia se ho la condizione iniziale sul modulo della velocità, non conosco l'alzo ma conosco il punto di atterraggio. Si trasforma il problema sul reticolato ma mettendo le condizioni iniziali e finali si può trasformare in un problema di algebra lineare, cosa che non abbiamo studiato per mancanza di tempo.

#### 22.1.1 Algoritmo di shooting

Ci si può ricondurre ad un problema alle condizioni iniziali, cioè il metodo di **shooting**. Questo vuol dire introdurre un parametro  $\delta$  che non conosciamo e simulare la traiettoria della particella che lanciamo. La particella cade in un punto  $\bar{x}$  e dovremo fare vari tentativi. Ad ogni tentativo c'è una quantità  $\varepsilon = |x_F - \bar{x}_i|$ , che conosco poiché  $x_F$  è una condizione al contorno. Si tratta di capire, per esempio, qual'è l'angolo migliore che ottimizza questa quantità ad essere vicina a  $\emptyset$ . Vuol dire che ho l'algoritmo che può essere Runge Kutta per fare la propagazione forward. Tutta la simulazione della propagazione della risoluzione dell'ODE, che porta a  $\bar{x}_i$  può essere racchiuso all'interno di una struttura che viene chiamata da uno degli algoritmi che abbiamo visto

per la ricerca degli zeri. Può essere una banale bisezione. La funzione è il risultato di questa simulazione e la ricerca dello zero si applica a questa quantità. Questo ci dimostra che possiamo assemblare algoritmi sviluppati in contesti distinti per costruire un algoritmo di risoluzione per un problema più complesso. Questo funziona bene e la velocità con cui si raggiunge la soluzione dipende da quanto riesco a dare un valore di prova a questo  $\delta_i$  che non sia troppo lontano al  $\theta$  vero che genera  $x_F$ . Se conosco una stima dell'angolo  $\theta$  che genera  $x_F$ , posso provare dei valori vicini a tale angolo. Devo fare un'esplorazione di tutte le possibilità di  $\delta_i$ . Anche un algoritmo di questo tipo si presta a uno studio da parte nostra. Abbiamo tutti gli strumenti e dobbiamo mettere assieme i pezzi.

### 22.1.2 Esercizio di QM

Si può fare sul testo di QM di Wong, dove si trova l'esercizio

$$\frac{\hbar^2}{2\mu} \nabla^2 \psi(\vec{r}) + V(\vec{r})\psi(\vec{r}) = E\psi(\vec{r})$$

$$E > 0$$

Si va in coordinate polari.

$$\psi(\vec{r}) = \sum_{l=0}^{\infty} \frac{1}{r} u_l(r) Y_{lm}(\vartheta, \varphi)$$

$$-\frac{\hbar^2}{2\mu} \left\{ \frac{d^2 u_l}{dr^2} - \frac{l(l+1)}{r^2} u_l(r) \right\} + V(r) u_l(r) = E u_l(r)$$

Si può inglobare la barriera di potenziale dovuta al momento angolare nel potenziale efficace.

$$V_{\text{eff}}(r) = V(r) + \frac{l(l+1)}{r^2}$$

Quindi la forma dell'eq. di Schrödinger diventa

$$-\frac{\hbar^2}{2\mu} \frac{d^2 u_l}{dr^2} + V_{\text{eff}}(r) u_l(r) = E u_l(r)$$

Questa è l'equazione. Si può studiare il caso particolare  $l = 0$ , dove si elimina la parte relativa ad  $l$  e si prende un potenziale della forma

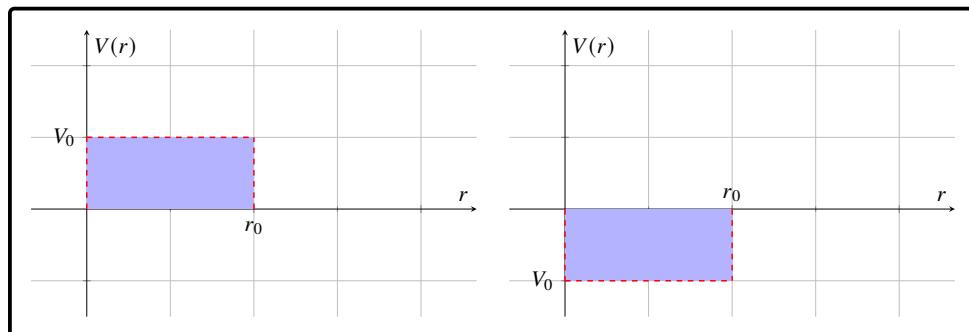
$$V(r) = \begin{cases} V_0 & r \leq r_0 \\ 0 & r > r_0 \end{cases}$$

La buca può essere attrattiva o repulsiva, a seconda del segno di  $V_0$ .

$V_0 > 0$  **repulsiva**

$V_0 < 0$  **attrattiva**

Nell'origine abbiamo la buca



Come condizione iniziale abbiamo che  $u(0) = 0$ . L'altra condizione emerge dai grandi  $r$ , dove  $V(r) = 0$ .

$$-\frac{\hbar^2}{2\mu} \frac{d^2 u_0}{dr^2} = E u_0(r)$$

$$\frac{d^2 u_0}{dr^2} + \frac{2\mu E}{\hbar^2} u_0 = 0$$

Al limite  $r \rightarrow \infty$  diventa un'equazione risolvibile con un ansatz del tipo

$$u_0(r) = A \sin(kr + \delta_0) \quad r \gg 0$$

Nel punto estremo della buca possiamo prendere la boundary condition  $u_0(r_0) = 1$ . Per un potenziale attrattivo abbiamo l'espressione

$$\delta_0 = \dots$$

Se vogliamo affrontare la cosa numericamente dobbiamo capire che è un'equazione del second'ordine con condizioni al contorno, una nel punto iniziale, una nel punto finale. Quindi possiamo mettere una condizione sulla derivata

$$\left. \frac{du_0}{dr} \right|_{r_0} \equiv \beta_1$$

che si va ad aggiungere alla condizione sulla funzione. Possiamo propagare forward la nostra soluzione numericamente facendo la reticolazione. Finiamo in  $r_0$  con  $u_0(r_0, \beta_1)$ , che in generale non coinciderà con  $u_0(r_d)$ , ma si minimizza tale distanza.

$$|u_0(r_0, \beta_1) - u_0(t_0)|$$

Ciò ci permette di trovare una buona soluzione approssimata del problema.

## 22.2 PDE

Sono più delicate delle ODE per quanto riguarda la convergenza. Facciamo una panoramica delle tipologie di equazioni che conosciamo

- Eq. delle Onde

$$\frac{\partial^2}{\partial t^2} \varphi - c^2 \frac{\partial^2}{\partial x^2} \varphi = 0$$

- Eq. di Diffusione

$$\frac{\partial}{\partial t} \varphi - k \frac{\partial^2}{\partial x^2} \varphi = 0$$

L'equazione di diffusione in un conduttore, dove  $k$  è la capacità termica.

- In condizioni di equilibrio dopo la termalizzazione, l'equazione di diffusione diventa l'equazione di Laplace

$$\nabla^2 \varphi = 0$$

In presenza di sorgenti abbiamo

$$\nabla^2 \varphi(\vec{r}) = f(\vec{r})$$

Questi sono i casi particolari e semplici che descrivono sistemi importanti nella fisica. In generale, la forma delle equazioni lineari del second'ordine alle derivate parziali è di questo tipo.

$$A \frac{\partial^2}{\partial x^2} \varphi + B \frac{\partial^2}{\partial x \partial t} \varphi + C \frac{\partial^2}{\partial t^2} \varphi + D \left( x, t, \varphi, \frac{\partial}{\partial x} \varphi, \frac{\partial}{\partial t} \varphi \right) = 0$$

Questa è la forma generale e queste equazioni precedenti rientrano in questa classe dove i coefficienti  $A, B, C$  possono essere funzioni di  $x, t$ .

### 22.2.1 Tre classi di PDE

Per risolvere l'equazione si procede con la diagonalizzazione, per annullare i coefficienti dei termini misti. Si generano chiaramente tre classi di equazioni. Questo perché nella procedura di diagonalizzazione compare il termine discriminante  $B^2 - 4AC$

- Per  $B^2 - 4AC > 0$  abbiamo l'equazione **Iperbolica**. In questa casistica rientra l'eq. delle onde che abbiamo visto in precedenza e l'eq. di Black & Scholes.
- Per  $B^2 - 4AC = 0$  abbiamo le equazioni **Paraboliche**. E' il caso dell'equazione di diffusione. Abbiamo una derivata prima uguale a delle derivate seconde.

- Per  $B^2 - 4AC < 0$  abbiamo le equazioni **Ellittiche**, in cui rientrano l'eq. di Poisson e l'equazione di Laplace.

Studiare questi tipi di equazioni ci dà un'idea di come simulare le eq. differenziali per una casistica varia e vasta.

### 22.2.2 Condizioni al contorno

Questi tipi di equazioni sono caratterizzate da due condizioni. Per di più abbiamo due variabili indipendenti. Per l'equazione di diffusione abbiamo

$$\frac{\partial}{\partial t} \varphi \propto \frac{\partial^2}{\partial x^2} \varphi$$

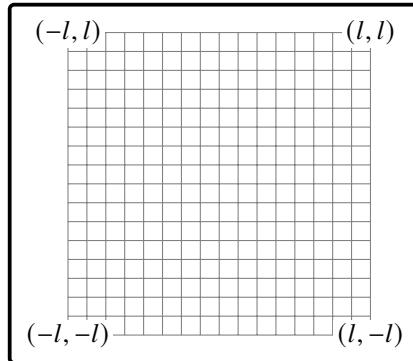
Per conoscere l'evoluzione della  $\varphi$  devo rericolare non in una, ma in due dimensioni. Questo tipo di equazioni sono caratterizzate dalla conoscenza in generale di condizioni al contorno. Per esempio, sul bordo della nostra regione che comprendono gli estremi delle nostre variabili. Possono essere di due tipi.

1. Le condizioni di Dirichlet dove specifichiamo il valore dei punti di bordo in funzione del tempo.
2. Le condizioni di Neumann, dove si specifica il valore del gradiente della normale al bordo, quindi implicano una conoscenza, facendo il mondo discretizzato, di due punti. Si dà la normale al bordo della mia regione.

Come per le ODE, si fa il reticolato e si vede l'eq. come diventa, a seconda delle espressioni che utilizziamo per le derivate. L'eq. di Poisson in due dimensioni è un'eq. ellittica.

$$\begin{aligned} \frac{\partial^2}{\partial x^2} V + \frac{\partial^2}{\partial y^2} V &= -\rho(x, y) \\ \frac{d^2\varphi}{dx^2}(x) + k^2\varphi(x) &= 0 \end{aligned}$$

Questa equazione può descrivere il potenziale elettrostatico generato da un filo di lughezza infinita. Se studiamo la struttura di cavo coassiale, il filo è circondato da una scatola di conduttore. Se vediamo questa cosa dall'alto, abbiamo una retta uscente dalla pagina e i quattro punti del quadrato, che sono posizionati in  $(-l, l)$   $(l, l)$   $(l, -l)$   $(-l, -l)$ .



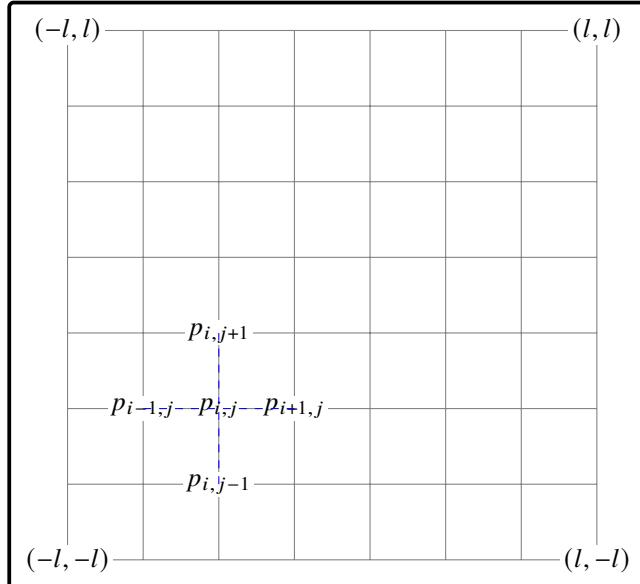
Se abbiamo un materiale conduttore esterno il potenziale su questi punti è sempre lo stesso valore. Questo vuol dire che  $V(x = \pm l; y) = V(x, y = \pm l) = V_0$ . Se tutto è collegato a terra,  $V_0 = 0$ . Questo è un esempio tipico di condizioni al contorno di Dirichlet. Se vogliamo conoscere il potenziale in questa regione, dobbiamo reticolare tutta la regione vista dall'alto. E' appunto, un quadrato, conosciamo il valore della funzione sul perimetro. Per esempio scrivo la derivata seconda

$$\begin{aligned} &\frac{1}{\Delta x^2} \left\{ V(x_{i+1}, y_j) - 2V(x_i, y_j) + V(x_{i-1}, y_j) \right\} \\ &+ \frac{1}{\Delta x^2} \left\{ V(x_i, y_{j+1}) - 2V(x_i, y_j) + V(x_i, y_{j-1}) \right\} \\ &= -\rho(x_i, y_j) \end{aligned}$$

Se identifico lo stesso intervallo sia su  $x$  che su  $y$ ,  $\Delta x = \Delta y = h$ , ottengo

$$V(x_{i+1}, y_j) + V(x_{i-1}, y_j) + V(x_i, y_{j+1}) + V(x_i, y_{j-1}) - 4V(x_i, y_j) = -h^2 \rho(x_i, y_j)$$

Questa è l'equazione discretizzata e rispetto al caso unidimensionale vediamo che devo valutare la funzione rispetto a molti più punti. Per un punto  $i, j$  ho coinvolti nell'equazione i punti  $(i+1, j)$ ,  $(i-1, j)$ ,  $(i, j+1)$ ,  $(i, j-1)$ . Cioè ho la richiesta di conoscere i punti nel quadrato:



Gauss Seidel o Relaxation Method, chiamato ufficiosamente *metodo di rilassamento*.

### 22.2.3 Equazione di diffusione

Vediamo il caso dell'equazione di diffusione.

$$\frac{\partial}{\partial t} \varphi(x, t) = D \frac{\partial^2}{\partial x^2} \varphi(x, t)$$

Indicizziamo i punti  $x$  del reticolo spaziale come  $j = 0, 1, 2, \dots, N_x$  e i punti del reticolo temporale li indicizziamo con  $k = 0, 1, 2, \dots, N_t$ . La dipendenza temporale della derivata richiede una condizione di input iniziale e la derivata seconda in  $x$  richiede delle condizioni al contorno ad istanti fissati. Conoscere

$$\varphi(t_0, x_j) \quad \forall j$$

vuol dire che conosciamo la funzione su tutti i punti del reticolo a  $t = 0$ . Per esempio

$$\varphi(t, x), \quad x = x_0 \text{ e } x = x_N \quad \forall t$$

vuol dire conoscere la funzione ai bordi del reticolo in  $x$  ad ogni tempo. A questo punto dobbiamo scrivere l'equazione discretizzata. Abbiamo

$$\begin{aligned} \left. \frac{\partial}{\partial t} \varphi \right|_{t=t_k} &= \frac{1}{\Delta t} \left\{ \varphi(x, t_{k+1}) - \varphi(x, t_k) \right\} \\ \left. \frac{\partial^2}{\partial x^2} \varphi \right|_{x=x_j} &= \frac{1}{(\Delta x)^2} \left\{ \varphi(x_{j+1}, t) - 2\varphi(x_j, t) + \varphi(x_{j-1}, t) \right\} \end{aligned}$$

Nell'equazione della derivata temporale si lascia non specificata la  $x$ , come nella derivata seconda spaziale non si specifica il tempo. Nella derivata seconda, abbiamo sostanzialmente usato la differenza finita, centrata su  $x_j$  e quindi possiamo prendere il tempo al corrispondente step che indichiamo con  $t_k$ . A questo punto abbiamo, per la derivata rispetto al tempo, la derivata rispetto a  $t_k$  e quindi viene naturale sostituire  $x = x_j$ . Con queste sostituzioni per le derivate l'equazione diventa.

$$\frac{\varphi(x_j, t_{k+1}) - \varphi(x_j, t_k)}{\Delta t} = D \frac{\varphi(x_{j+1}, t_k) - 2\varphi(x_j, t_k) + \varphi(x_{j-1}, t_k)}{(\Delta x)^2}$$

Se guardiamo l'equazione compare una dipendenza da  $t_{k+1}$ , però se andiamo a vedere le nostre condizioni iniziali, sul tempo  $t$  conosciamo l'array della funzione al tempo 0 in tutti i punti  $x_j$ . Quindi siamo capaci di

far partire l'algoritmo perché se prendiamo l'algoritmo al tempo 0, conosciamo tutti i valori con eccezione di  $\varphi(x_j, t_{k+1})$ . Stiamo muovendo, a seconda della  $j$ , l' $x_j$  sulla base del reticolato, che fa parte delle condizioni iniziali.

### 22.2.4 Schema esplicito

Per parlare di schema implicito-explicito che abbiamo nominato nella risoluzione delle ODE, abbiamo per lo schema **esplicito** la conoscenza della condizione iniziale al tempo 0, perciò possiamo conoscere la  $\varphi$  al tempo  $t_k$ . Preso un punto qualunque  $\varphi(x_j, t_k)$  possiamo stimare la  $\varphi(x_j, t_{k+1})$ . Poiché posso effettuare questa operazione per tutti i punti  $x_j$ , da uno strato precedente posso ottenere lo strato successivo

#### Analisi della stabilità

Può essere più delicata l'analisi di tutti gli algoritmi. Può essere utile fare un'analisi di stabilità. Proviamo con la soluzione esponenziale

$$\phi(x_j, t_n) = \xi^n e^{ikjx}$$

Con  $k$  numero d'onda sul reticolato. L'algoritmo che abbiamo scritto prima è uguale a

$$\xi^{n+1} = \xi^n e^{ikj\Delta x} + \xi^n \frac{D\Delta t}{(\Delta x)^2} [e^{ik(j+1)\Delta x} - 2e^{ikj\Delta x} + e^{ik(j-1)\Delta x}]$$

ho preso l'equazione e ho moltiplicato il secondo membro per  $\Delta t$ . Perciò scrivo  $\varphi_{j,k+1} = \varphi_{j,k} + \dots$ . Semplifichiamo perciò ambo i membri per  $\xi^n$  e per  $e^{ikj\Delta x}$ , difatti, divido per  $\phi_{j,n}$ .

$$\begin{aligned}\xi &= 1 + \frac{D\Delta t}{(\Delta x)^2} [e^{ik\Delta x} - 2 + e^{-ik\Delta x}] \\ \xi &= \frac{D\Delta t}{(\Delta x)^2} [e^{ik\Delta x} - 2 + e^{-ik\Delta x}]\end{aligned}$$

Riconoscendo la definizione esponenziale del coseno trasformiamo la quantità in

$$\begin{aligned}\xi &= 1 + 2 \frac{D\Delta t}{(\Delta x)^2} [\cos k\Delta x - 1] \\ &= 1 - 4 \frac{D\Delta t}{(\Delta x)^2} \sin^2 \frac{1}{2} k\Delta x\end{aligned}$$

La componente dovuta al seno quadro è sempre  $\leq 1$ . Per evitare che l'algoritmo diverga nel corso delle iterazioni della ricostruzione della soluzione della PDE è necessario che  $|\xi| < 1$ . Questo ci comporta dei vincoli tra  $\Delta t$  e  $\Delta x$ . **Si espone una versione corretta dell'analisi della stabilità**: prendiamo in considerazione il fatto che  $\xi$  è nella forma  $1 - x$ . Imporre il modulo  $\leq 1$  significa che la  $x$  è compresa nell'intervallo  $[0, 2]$ . Non è corretto a priori imporre  $|x| \leq 1$ , poiché nella fascia  $[-1, 0]$  porta a divergenza. La condizione di stabilità va studiata correttamente e a questa si dedicano le successive poche righe. Si ricorda che il modulo è  $\leq 1 \iff x \in [0, 2]$ . Poiché  $\sin^2$  è già non negativo, si studia il suo coefficiente. Il segno del coefficiente dipende da  $D$ , che, nell'ipotesi di incrementi positivi, deve quindi essere positivo.

$$\begin{aligned}0 &< 4 \frac{D\Delta t}{(\Delta x)^2} < 2 \\ 4 \frac{D\Delta t}{(\Delta x)^2} &< 2 \\ \frac{D\Delta t}{(\Delta x)^2} &< \frac{1}{2}\end{aligned}$$

Si esplicitano gli incrementi l'uno in funzione dell'altro

$$\begin{aligned}\Delta t &< \frac{(\Delta x)^2}{2D} \\ \sqrt{2D\Delta t} &< \Delta x < -\sqrt{2D\Delta t}\end{aligned}$$

Si cancella la soluzione non accettabile nell'ipotesi di incrementi positivi. Questo con lo schema **esplicito**.

### 22.2.5 Schema implicito

Un'altra possibilità è utilizzare uno schema **implicito**, dove, avendo posto per la stima della derivata seconda un tempo  $t_k$ , nulla mi vieta di porre  $t_{k+1}$ . In quel punto vado in uno schema implicito perché non ho più la possibilità di propagare come nello schema esplicito. Si cambia radicalmente il metodo di risoluzione.

$$\frac{\varphi(x_j, t_{k+1}) - \varphi(x_j, t_k)}{\Delta t} = D \frac{\varphi(x_{j+1}, t_{k+1}) - 2\varphi(x_j, t_{k+1}) + \varphi(x_{j-1}, t_{k+1})}{(\Delta x)^2}$$

Ovviamente adesso non posso utilizzare il metodo di prima perché precedentemente la mia incognita era la funzione del punto reticolare al tempo successivo, ora posso ottenere la funzione del punto reticolare al tempo precedente. Ho ottenuto quindi un algoritmo backward. Possiamo raggruppare quindi tutti i termini in  $\phi_{,k+1}$  e scrivere un'equazione del tipo

$$\begin{aligned} A\varphi^{k+1} &= \varphi^k, \quad A = \{a\}_{i,j} \\ \frac{D\Delta t}{(\Delta x)^2} &\equiv \eta \end{aligned}$$

Se si prova a riscrivere l'equazione precedente con un passaggio algebrico, troviamo

$$\varphi(x_j, t_{k+1}) = \varphi(x_j, t_k) + \eta [\varphi(x_{j+1}, t_{k+1}) - 2\varphi(x_j, t_{k+1}) + \varphi(x_{j-1}, t_{k+1})]$$

Possiamo riscrivere l'espressione isolando il termine  $\varphi_{j,k+1}$ .

$$-\eta\varphi(x_{j+1}, t_{k+1}) + (2\eta + 1)\varphi(x_j, t_{k+1}) - \eta\varphi(x_{j-1}, t_{k+1}) = \varphi(x_j, t_k)$$

Questa è la forma dalla quale si possono ricavare i coefficienti della matrice  $A$ . Eventualmente in seguito si può vedere l'applicazione di  $A$ . La soluzione forward consiste nel fare l'inverso della matrice, cioè  $A^{-1}$ .

$$\varphi^{k+1} = A^{-1}\varphi^k$$

Basta cambiare leggermente come stiamo scrivendo l'equazione discretizzata e cambia completamente il metodo di risoluzione del problema. Il metodo implicito ha la particolarità che, conducendo l'analisi della stabilità, troviamo che

$$\begin{aligned} \xi^{n+1} e^{ikj\Delta x} &= \xi^n e^{ikj\Delta x} + \eta \xi^{n+1} \frac{D\Delta t}{(\Delta x)^2} [e^{ik(j+1)\Delta x} - 2e^{ikj\Delta x} + e^{ik(j-1)\Delta x}] \\ &\quad - \xi\eta 2 \cos k\Delta x + \xi\eta 2 = 1 \end{aligned}$$

Adesso abbiamo che

$$\begin{aligned} \xi [1 + 2\eta(1 - \cos k\Delta x)] &= 1 \\ \xi &= \frac{1}{1 + 2\eta(1 - \cos k\Delta x)} \end{aligned}$$

Il coseno è compreso tra  $[-1, 1]$  e con  $\eta > 0$  l'espressione in modulo è sicuramente  $\leq 1$ . Perciò si dice che l'espressione è **incondizionatamente stabile**. Questo perché possiamo reticolare  $\Delta t$  e  $\Delta x$  come vogliamo, senza preoccuparci della stabilità della soluzione.

### 22.2.6 Eq. d'Onda

Si conduce una particolare analisi di stabilità che ne fa emergere un problemino. L'eq. delle onde è

$$\begin{aligned} \frac{\partial^2}{\partial t^2} \varphi &= v^2 \frac{\partial^2}{\partial x^2} \varphi \\ \begin{cases} r = v \frac{\partial}{\partial x} \varphi \\ s = \frac{\partial}{\partial t} \varphi \end{cases} \end{aligned}$$

Posso scrivere dall'eq. iniziale

$$\begin{aligned} \frac{\partial}{\partial t} s &= v \frac{\partial}{\partial x} r \\ \frac{\partial}{\partial x} s &= \frac{1}{v} \frac{\partial}{\partial t} r \end{aligned}$$

Posso ricondurmi ad un sistema di eq. del primo ordine. L'eq. differenziale alle derivate parziali di base da trattare è un'eq. del tipo

$$\frac{\partial}{\partial t} \varphi = -v \frac{\partial}{\partial x} \varphi$$

e descrive un'onda che si propaga in verso positivo  $\varphi = f(x - vt)$ . Abbiamo che  $x_j = x_0 + j\Delta x$  e  $t_n = t_0 + n\Delta t$ , quindi  $u_j^n \equiv u(x_j, t_n)$

$$\begin{aligned} \left. \frac{\partial}{\partial t} u \right|_{j,n} &= \frac{u_j^{n+1} - u_j^n}{\Delta t} + O(\Delta t) \\ \left. \frac{\partial}{\partial x} u \right|_{j,n} &= \frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x} + O(\Delta x^2) \end{aligned}$$

Abbiamo utilizzato la derivata forward perché ci permette di mostrare lo schema esplicito con meno passaggi. Con le due espressioni per le derivate posso dunque scrivere

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = -v \frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x}$$

uno schema esplicito. L'unica variabile che dipende dal tempo  $n + 1$  è la prima a primo membro. Le altre variabili dipendono dal tempo  $n$  e le  $x$  sono  $j - 1, j, j + 1$ . Localmente è tutto apposto. Possiamo quindi risolvere rispetto a  $u_j^{n+1}$ .

$$u_j^{n+1} = u_j^n - \frac{v\Delta t}{2\Delta x} (u_{j+1}^n - u_{j-1}^n)$$

Con questo algoritmo posso far evolvere in avanti la soluzione. Il problema è che in questo caso di algoritmo esplicito, proviamo a sostituire una soluzione di tentativo

$$u_j^n = \xi^n \exp(ikj\Delta x)$$

e troviamo

$$\xi^{n+1} \exp(ikj\Delta x) = \xi^n \exp(ikj\Delta x) - \frac{v\Delta t}{2\Delta x} \xi^n [\exp(ik(j+1)\Delta x) - \exp(ik(j-1)\Delta x)]$$

Dividendo per  $u_j^n$  si ottiene

$$\begin{aligned} \xi &= 1 - \frac{v\Delta t}{2\Delta x} [\exp(ik\Delta x) - \exp(-ik\Delta x)] \\ &= 1 - i \frac{v\Delta t}{\Delta x} \sin k\Delta x \end{aligned}$$

Il che è un problema, perché il modulo di  $\xi$  è

$$|\xi| = \sqrt{1 + \left(\frac{v\Delta t}{\Delta x}\right)^2 \sin^2 k\Delta x} \geq 1 \quad \forall \Delta x, \Delta t$$

Il che significa che questo metodo è **incondizionatamente instabile**. Questo è un esempio di un'equazione che diventa sempre instabile.

### Metodo di Lax

E' stata studiata in letteratura ed è stato individuato un metodo per renderla stabile, il **metodo di Lax**. Propone di sostituire al posto di  $u_j^n$  la media tra  $j + 1, j - 1$ , cioè

$$u_j^n \rightarrow \frac{1}{2}(u_{j+1}^n + u_{j-1}^n)$$

Quindi la soluzione diventa più complicata di prima.

$$u_j^{n+1} = \frac{1}{2}(u_{j+1}^n + u_{j-1}^n) - \frac{v\Delta t}{2\Delta x} (u_{j+1}^n - u_{j-1}^n)$$

Dall'associata analisi della stabilità emerge che

$$\begin{aligned}\xi &= \frac{1}{2} [e^{ik\Delta x} + e^{-ik\Delta x}] - \frac{v\Delta t}{2\Delta x} [e^{ik\Delta x} - e^{-ik\Delta x}] \\ &= \cos k\Delta x - i \frac{v\Delta t}{\Delta x} \sin k\Delta x\end{aligned}$$

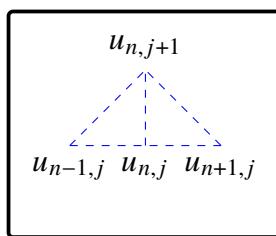
Calcolo il modulo quadro associato

$$|\xi|^2 = \cos^2 k\Delta x + \frac{v^2 \Delta t^2}{\Delta x^2} \sin^2 k\Delta x$$

Questo è meglio di prima, ma bisogna stare attenti. La quantità  $\frac{v^2 \Delta t^2}{\Delta x^2}$  deve essere  $\leq 1$ . Ovvero

$$\frac{|v|\Delta t}{\Delta x} \leq 1 \quad \text{stabile}$$

La condizione di stabilità si traduce in  $|v|\Delta t \leq \Delta x$ . Definisco il nostro reticolato che prende 3 punti e calcola quello successivo. Il vincolo si traduce graficamente nella massima altezza che può avere il triangolo rispetto alla sua base.



Lo spazio percorso in verticale deve essere minore di quello percorso in orizzontale saltando da un punto all'altro. C'è un vincolo sul reticolato in  $\Delta t$  e in  $\Delta x$ . L'idea è quella di mostrare alcune PDE, le principali, e di vedere come vengono scritte in forma discretizzata. Vediamo come vengono scritte sulla carta e vediamo delle soluzioni numeriche di queste equazioni. Il corpus degli algoritmi di risoluzione delle PDE è molto più complesso rispetto all'insieme degli algoritmi di risoluzione delle ODE, in quanto è più critica la stabilità.

# Chapter 23

## Lezione 23

### 23.1 Eq. del Calore

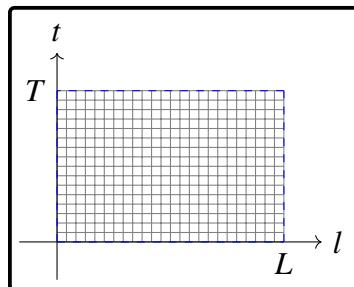
Vediamo l'implementazione algoritmica dell'eq. del calore. Abbiamo una sbarra che mettiamo a contatto con due sorgenti di temperatura. Le sorgenti sono a temperatura  $T_1, T_2$ . Studiamo la propagazione del calore in questa sbarra, caratterizzata da un coefficiente di conduzione termica  $D$ . L'equazione è di tipo parabolico

$$\frac{\partial}{\partial t} \varphi(x, t) = D \frac{\partial^2}{\partial x^2} \varphi(x, t)$$

Poniamo la temperatura all'istante iniziale  $t = 0$  uguale alla temperatura  $T_2$ .  $\varphi(x, 0) = T_2$ . Nel corso del tempo la temperatura del punto iniziale e del punto finale sono sempre le stesse, cioè  $\varphi(0, t) = T_1$  e  $\varphi(L, t) = T_2$ . Abbiamo visto in precedenza la discretizzazione che portava a questa eq. differenziale.

$$\frac{\varphi(x_i, t_{j+1}) - \varphi(x_i, t_j)}{\Delta t} = D \frac{\varphi(x_{i+1}, t_j) - 2\varphi(x_i, t_j) + \varphi(x_{i-1}, t_j)}{(\Delta x)^2}$$

Implementiamo numericamente l'algoritmo. Se abbiamo il filo di lunghezza  $L$  con il tempo sull'asse delle  $y$ , dobbiamo reticolare il rettangolo  $(0, 0) - (L, T)$ .



#### 23.1.1 Schema esplicito

##### Implementazione algoritmica

Conoscendo in tutti i punti la temperatura al tempo 0 possiamo propagare la temperatura in avanti nel tempo. Il programma è molto semplice. Ho  $n$  intervalli sulla lunghezza del filo e  $m$  sui tempi. Scrivo il valore di  $dx$  per dare un'idea dimensionale. Con tali valori inizializzo una matrice  $\varphi[n, m]$ . Per assegnare i valori della funzione alla matrice effettuo due cicli, uno sui tempi e uno nidificato sugli step del filo. L'algoritmo che utilizzo per lo step temporale successivo in ogni posizione è

$$\varphi(x_i, t_{j+1}) = \varphi(x_i, t_j) + \frac{D\Delta t}{(\Delta x)^2} \varphi(x_{i+1}, t_j) - 2\varphi(x_i, t_j) + \varphi(x_{i-1}, t_j)$$

Successivamente scrivo all'interno dei file di testo i risultati della ricostruzione. Ogni file di testo contiene tutti i valori della temperatura in un dato step temporale, con il nome che comprende un progressivo pari all'indice dello step temporale +100. Vediamo i risultati.

0.000000000000	373.000000000000
1.010101010101	273.000000000000
2.020202020202	273.000000000000
3.030303030303	273.000000000000
4.040404040404	273.000000000000
5.050505050505	273.000000000000
6.060606060606	273.000000000000
7.070707070707	273.000000000000
8.080808080808	273.000000000000
9.090909090909	273.000000000000
10.101010101010	273.000000000000

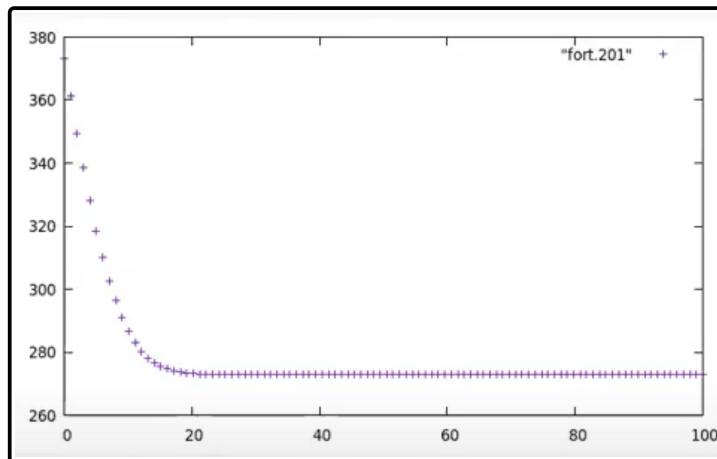
Il primo file contiene le condizioni iniziali rispetto alla lunghezza del filo.

0.000000000000	373.000000000000
1.010101010101	295.346280000000
2.020202020202	273.000000000000
3.030303030303	273.000000000000
4.040404040404	273.000000000000
5.050505050505	273.000000000000
6.060606060606	273.000000000000
7.070707070707	273.000000000000
8.080808080808	273.000000000000
9.090909090909	273.000000000000
10.101010101010	273.000000000000

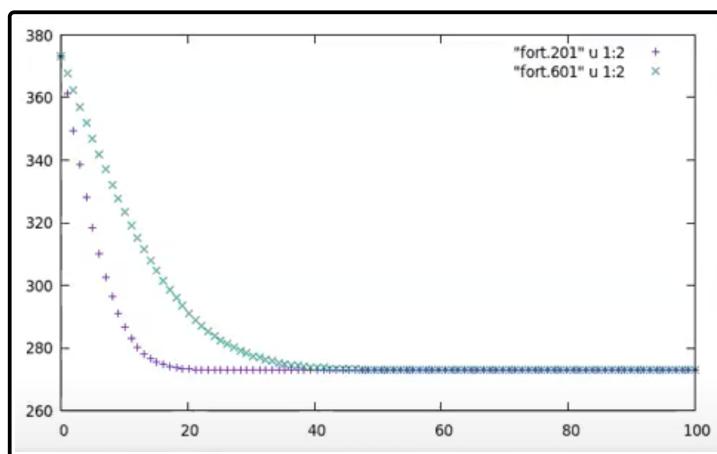
Alla seconda iterazione la temperatura si è mossa al secondo punto, lasciando invariati gli altri. E così via. Andiamo adesso a vedere l'ultimo file, che con 1000 step è il numero 1100.

0.000000000000	373.000000000000
1.010101010101	369.226405259532
2.020202020202	365.461245034641
3.030303030303	361.712897340360
4.040404040404	357.989627820805
5.050505050505	354.299535129323
6.060606060606	350.650498160039
7.070707070707	347.050125702972
8.080808080808	343.505709057518
9.090909090909	340.024178093780
10.101010101010	336.612061198863

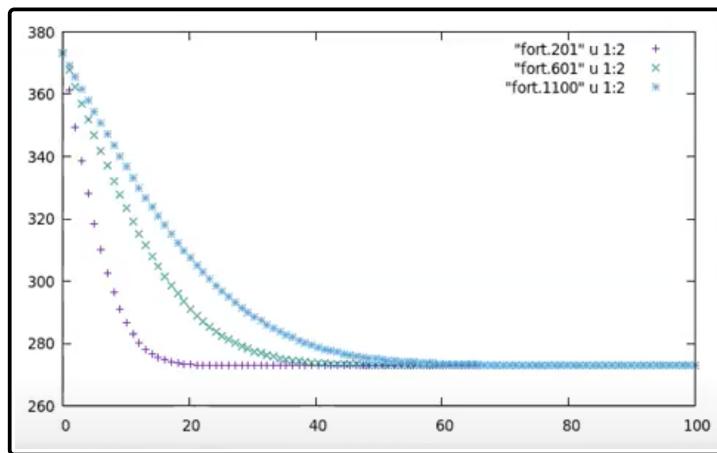
Notiamo che la temperatura dal primo punto, che rimane invariato, diminuisce ma si è propagata per tutto il resto del filo. Possiamo fare un plot di un file intermedio.



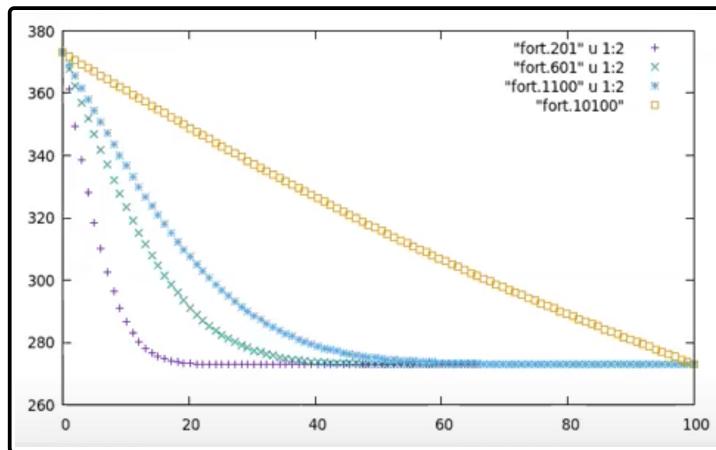
Si vede che la temperatura ha iniziato a propagarsi dall'estremo sinistro. Plotto le distribuzioni di due step distinti.



La distribuzione di temperatura si sta chiaramente evolvendo. Includo anche l'istante 1100.



Per la distribuzione limite si può pensare di trovare la soluzione analitica, che comprende una serie. Poi si può pensare di elaborare uno script sul come rappresentare numericamente una serie, eppure non ci si dedica in questo corso. La temperatura comunque decresce man mano che si raggiunge l'estremo destro.



Spostiamo i risultati ottenuti in questa simulazione nei risultati esplicativi. Si può fare un numero di step maggiore e vedere come cambia la distribuzione di temperatura. Proviamo a diminuire lo spacing da  $1\text{cm}$  a  $1\text{mm}$ . I punti da 100 diventano 1000. Settiamo  $dt = 1.974 \times 10^{-2} \text{s}$  per garantire la stabilità e notiamo che 1000 step con la risoluzione temporale specificata garantiscono una copertura di  $\sim 20\text{s}$ .

### Condizioni di stabilità

Proviamo a toccare con mano la stabilità del programma, commentando le righe dello script che correggono il valore di  $dt$ .

0.000000000000	373.000000000000
0.100100100100	2548.442280000000
0.200200200200	273.000000000000
0.300300300300	273.000000000000
0.400400400400	273.000000000000
0.500500500501	273.000000000000
0.600600600601	273.000000000000
0.700700700701	273.000000000000
0.800800800801	273.000000000000
0.900900900901	273.000000000000

Al primo step oltre le condizioni iniziali individuiamo già le prime deviazioni, con temperature oltre la  $T_1$ , perciò non logico. Andando a vedere l'ultimo step si vede chiaramente che i valori della temperatura sono tutti chiaramente dei **NaN**.

0.000000000000	373.000000000000
0.100100100100	NaN
0.200200200200	NaN
0.300300300300	NaN
0.400400400400	NaN
0.500500500501	NaN
0.600600600601	NaN
0.700700700701	NaN
0.800800800801	NaN
0.900900900901	NaN

Deduciamo che appena si viola la condizione di stabilità, l'algoritmo non converge e genera numeri fuori da ogni logica.

### 23.1.2 Schema Implicito

Possiamo utilizzare uno schema implicito e quando adottiamo questo schema è possibile che si verifichi una condizione di convergenza differente. Difatti per l'equazione del calore abbiamo visto che l'algoritmo implicito è incondizionatamente stabile.

$$\begin{aligned}\eta &= \frac{D\Delta t}{(\Delta x)^2} \\ -\eta\varphi(x_{i+1}, t_{j+1}) + (2\eta + 1)\varphi(x_i, t_{j+1}) - \eta\varphi(x_{i-1}, t_{j+1}) &= \varphi(x_i, t_j) \\ A\varphi^{j+1} &= \varphi^j, \quad A = \{a\}_{i,j} \\ \varphi^{j+1} &= A^{-1}\varphi^j \\ \xi &= \frac{1}{1 + 2\eta(1 - \cos k\Delta x)}\end{aligned}$$

La matrice  $A$  ha i coefficienti dati dai coefficienti della funzione  $\varphi_{i,j}$ . Perciò  $A_{i,j} = c_{i,j}$ . Per ogni riga la matrice ha solo 3 elementi non nulli e giacciono sulle tre diagonali centrali. Una matrice di questo tipo si chiama matrice **tridiagonale**. Possiamo scrivere la relazione di ricorsione nella forma

$$\begin{aligned}A_i^- \varphi_{i-1}^{j+1} + A_i^0 \varphi_i^{j+1} + A_i^+ \varphi_{i+1}^{j+1} &= \varphi_i^j \\ -\eta\varphi_{i-1}^{j+1} + (2\eta + 1)\varphi_i^{j+1} - \eta\varphi_{i+1}^{j+1} &= \varphi_i^j \\ A_i^- = -\eta, \quad A_i^0 = 2\eta + 1, \quad A_i^+ = -\eta\end{aligned}$$

#### Inversione matrice tridiagonale

Ci sono algoritmi per trovare l'inverso di una matrice generica. Eppure questa è una matrice tridiagonale, il che vuol dire che si può risolvere in modo molto più semplice che con un algoritmo generale, che è il seguente. Sappiamo che la  $\varphi_{i+1}$  si può ottenere come propagazione forward in  $x$  sempre allo stesso  $j + 1$ . Proponiamo un ansatz.

$$\varphi_{i+1}^{j+1} = \alpha_i \varphi_{i+1}^{j+1} + \beta_i$$

Vediamo se otteniamo qualcosa di sensato. Sostituiamo l'ansatz nell'equazione che conosciamo. Ci si riferisce a  $\varphi_i^j = b_i$ .

$$\begin{aligned}A\varphi_{j+1} &= \varphi_j \\ \varphi_{j+1} &= A^{-1}\varphi_j\end{aligned}$$

L'equazione funziona per tutte le  $x$ , quindi  $\varphi_j$  funge da termine noto, vettore dei termini noti e si chiama  $b_i$ .

$$A_i^- \varphi_{i-1}^{j+1} (A_i^0 + A_i^+ \alpha_i) \varphi_i^{j+1} + A_i^+ \beta_i = b_i$$

Questa è la nostra equazione di partenza che non ci dice niente. Però qui dentro risolviamo per  $\varphi_i$

$$\begin{aligned}\varphi_i &= \gamma_i A_i^- \varphi_{i-1} + (A_i^+ \beta_i - b_i) \gamma_i \\ \gamma_i &= -\frac{1}{A_i^0 + A_i^+ \alpha_i}\end{aligned}$$

Confrontiamo questa equazione con la nostra ipotesi di lavoro. Se scaliamo l'indice  $i$  di un'unità, anche questa diventa

$$\varphi_i = \alpha_{i-1} \varphi_{i-1} + \beta_{i-1}$$

Confrontando questa equazione con quella precedente possiamo scrivere che

$$\begin{aligned}\alpha_{i-1} &= \gamma_i A_i^- \\ \beta_{i-1} &= \gamma_i (A_i^+ \beta_i - b_i)\end{aligned}$$

E' soltanto confrontando queste due equazioni che possiamo ottenere l'espressione dei coefficienti.  $A_i^+, A_i^-$  sono delle costanti, perciò la prima equazione ci dà una relazione di ricorrenza che ci permette di calcolare  $\alpha_{i-1}$  in funzione di  $\alpha_i$ . Una relazione di ricorsione backwards. In queste espressioni non compare la  $\varphi$  in nessun modo, perché questi coefficienti sono tutti indipendenti da  $\varphi$ . Se riesco a trovare un  $\alpha$  a cui riesco ad

appigliarmi riesco a trovare gli  $\alpha$  per tutti i punti in  $x$ . Viceversa ho nella seconda equazione una relazione di ricorrenza che mi lega  $\beta_{i-1}$  con  $\beta_i$ , ma c'è anche il termine noto, che è  $\varphi_i^j$ , che è al tempo  $j$ , mentre tutte le altre quantità dove compariva la  $\varphi$  erano al tempo  $j + 1$ . Se noi mettiamo, per esempio,  $\alpha_n = 0$ , dove  $\alpha_n$  è il coefficiente  $\alpha$  all'ultimo punto del mio reticolo spaziale in  $x$ . Noi abbiamo che la nostra ipotesi di lavoro era  $\varphi_{i+1}^{j+1} = \alpha_i \varphi_{i+1}^{j+1} + \beta_i$ . Ricordiamoci che se prendiamo  $\alpha_{n-1} = 0$  abbiamo il termine  $\alpha_i \varphi_{i+1}^{j+1} = 0$  e abbiamo  $\varphi_{i+1}^{j+1} = \beta_i$ . Ma  $\varphi_n$  è l'ultima  $\varphi$  nella griglia spaziale. Abbiamo inoltre detto che la condizione iniziale ci fissa il profilo del materiale a temperatura costante. Possiamo dire, per esempio, che, in corrispondenza di  $\alpha_{n-1}=0$ , abbiamo che

$$\begin{cases} \alpha_{n-1} = 0 \\ \beta_{n-1} = \varphi_n^j \end{cases}$$

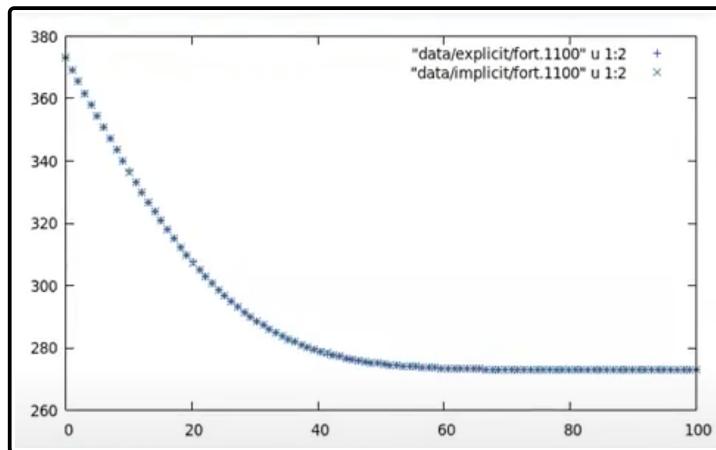
Ed è una impostazione molto furba perché da qui, le due relazioni sono di ricorrenza se noi abbiamo individuato un valore di  $(\alpha_i)$ . Se noi ci riferiamo ad  $\alpha_{n-1}$ , con questa impostazione che è corretta in base alla nostra condizione iniziale, da lì possiamo trovare tutti gli  $\alpha_i$  e i  $\beta_i$  con le relazioni di ricorsione. Questo è di una potenza computazionale notevole. Ricordiamoci che gli  $\alpha_i$  sono indipendenti dalle  $\varphi$ , perciò possiamo risolvere una volta sola che sia valida per tutti i punti. Poi possiamo trovare i  $\beta_i$  con la relazione di ricorsione. Questo purché facciamo attenzione al fatto che  $b_i = \varphi_i^j$ . L'array dei  $\beta_i$  dipenderà dal time slice temporale precedente. Individuati i due array, a questo punto possiamo usare la relazione lineare a ogni tempo, avendo conosciuto  $\varphi^j$  su tutto il reticolo, come all'istante 0 da condizioni iniziali. Ci mettiamo su ogni punto e abbiamo anche la condizione iniziale su  $j + 1$ , fissato  $i = 1$ . La temperatura all'estremo sinistro è fissata e a partire da quello facciamo la propagazione forward sulla linea a slicing temporale fissato e conosciamo tutte le  $\varphi$  su tutto l'array temporale. Questo ci permette di trovare la soluzione facendo un passaggio con  $n$  operazioni per trovare  $\alpha_i$  e  $\beta_i$  e facendo  $n$  iterazioni in avanti per trovare il campo di temperatura. Qui si vede che in questo modo ci ricordiamo che abbiamo detto che se partiamo dall'equazione scritta in forma implicita, facciamo un'analisi di stabilità di Von Neumann, non vengono condizioni particolari su  $\frac{\Delta t}{(\Delta x)^2}$ .

### Esecuzione script

Proviamo come parametri di input gli stessi del metodo esplicito, per confrontare i due metodi. Dunque  $L = 100 \text{ cm}$ ,  $dt = 1$  e  $m = 1000$ . Andiamo a vedere i file prodotti.

0.000000000000	373.000000000000
1.010101010101	369.223574345856
2.020202020202	365.455614892222
3.030303030303	361.704530842576
4.040404040404	357.978618046506
5.050505050505	354.286003913132
6.060606060606	350.634594204954
7.070707070707	347.032022292865
8.080808080808	343.485601414761
9.090909090909	340.002280433785
10.101010101010	336.588603538644

Spostiamo i file prodotti nella cartella **implicit**. Generiamo un plot di confronto tra i due metodi.



Notiamo accordo tra le due soluzioni. Questo è un buon indice della bontà della soluzione con il metodo implicito. Possiamo provare a fare un run aumentando lo slice temporale. Proviamo con 1 m di lunghezza del filo, 1000 slice spaziali e 1000 slice temporali.

0.000000000000	373.000000000000
0.100100100100	354.118867977074
0.200200200200	338.802707418819
0.300300300300	326.378411356412
0.400400400400	316.299963036467
0.500500500501	308.124439849674
0.600600600601	301.492547989343
0.700700700701	296.112832386780
0.800800800801	291.748867989594
0.900900900901	288.208869471675

Mentre con lo schema esplicito avevamo un comportamento anomalo, non sembra ci siano problemi in questo caso. Anche per quanto riguarda i tempi, l'algoritmo implicito impiega meno tempo, seppur confrontabile con il tempo dell'algoritmo esplicito.

### 23.1.3 Controllo bontà soluzione

Se uno si deve chiedere quanto siano attendibili questi risultati rispetto alla risoluzione del problema fisico, ci sono dei metodi che si possono seguire.

- Ci si riconduce ad un problema più semplice con soluzione analitica
- Si implementa la soluzione analitica in qualche limite
- Rispetto a diversi algoritmi già testati, provvedo a confrontare la soluzione generata dall'algoritmo che sto vagliando con quella degli algoritmi che so essere attendibili.
- Controllo se al variare del passo su  $x$  e su  $t$  la funzione è stabile. Con l'algoritmo esplicito non riesco ad andare oltre un certo limite, mentre riesco con l'algoritmo implicito.

### 23.1.4 Note per l'esame

L'attenzione è sull'algoritmo, non sul linguaggio. Si discute sul perché quell'algoritmo piuttosto che un altro. In base al problema da risolvere devo ragionare per capire quale algoritmo mi piace utilizzare. Si dà l'esame non necessariamente nelle date strette dell'appello.