Contents

1 Aritmetica computazionale

1.1 Rappresentazione dei numeri reali

I **numeri finiti** sono utilizzati dai calcolatori per rappresentare i numeri reali poiché questi ultimi possono avere un numero infinito di cifre, che i calcolatori, avendo una memoria limitata, non sono in grado di rappresentare.

Teorema (Rappresentazione in base)**.** *Sia α un numero reale non nullo. Possiamo rappresentare tale numero con una base $\beta \geq 2$, un numero intero scelto da noi, nel seguente modo:*

$$\begin{aligned} \alpha &= \pm(\alpha_1\beta^{-1} + \alpha_2\beta^{-2} + \ldots)\beta^p \\ \alpha &= \pm(\sum_{i=1}^{\infty} \alpha_i\beta^{-i})\beta^p \end{aligned} \tag{1.1}$$

I vari termini della 1.1 vengono detti:

β	*base*
p	*esponente*
α_i	*cifre del numero*
$\sum_{i=1}^{\infty} \alpha_i\beta^{-i}$	*mantissa*

Ogni cifra α_i è un numero intero che varia tra 0 e $\beta - 1$. Ad esempio, se lavoriamo in base 10, le cifre saranno numeri interi compresi tra 0 e 9.
Per garantire l'unicità della rappresentazione, è necessario che $\alpha_1 \neq 0$. Se così non fosse, il numero 13 potrebbe essere rappresentato come 13, 013, 0013, eccetera, il che va contro l'unicità della rappresentazione.

Possiamo scrivere un numero $\alpha \in \mathbb{R}$ con $\alpha \neq 0$ in due modi:

1. **forma mista**.
$$\alpha = \begin{cases} \pm(0.000\alpha_1\alpha_2\ldots)_\beta & p \leq 0 \\ \pm(\alpha_1\alpha_2\ldots)_\beta & p > 0 \end{cases}$$
2. **forma scientifica**. L'idea è quella di spostare il punto decimale al primo numero $\neq 0$ e poi moltiplicare il tutto per β^p per riportare il numero al suo valore originale.
$$\alpha = \pm 0.\alpha_1\alpha_2\ldots \cdot \beta^p$$

Esempio:

$$\begin{aligned} \alpha = (12.37)_{10} \quad & \alpha = 0.12237 \cdot 10^2 \\ \alpha = (0.0045)_{10} \quad & \alpha = 0.45 \cdot 10^{-2} \\ & = (4 \cdot 10^{-1} + 5 \cdot 10^{-2}) \cdot 10^{-2} \end{aligned}$$

Definizione (Numeri finiti)**.** *L'insieme $\mathbb{F}$ dei numeri finiti è definito come l'insieme dei numeri espressi in base β (dove $\beta \geq 2$), utilizzando t cifre (con $t \geq 1$). Poiché anche l'esponente p potrebbe essere così grande da non poter essere rappresentato, è necessario limitare l'intervallo degli esponenti rappresentabili. Qui, λ indica il più piccolo esponente che può essere rappresentato e ω il più grande esponente rappresentabile.*

$$\begin{aligned} \mathbb{F}(\beta, t, \lambda, \omega) &= \{0\} \cup \{\alpha \in \mathbb{R} : \alpha = \pm 0.\alpha_1\alpha_2\ldots\alpha_t \cdot \beta^p, \\ &= \{0\} \cup \{\alpha \in \mathbb{R} : \alpha = \pm(\sum_{i=1}^{t} \alpha_i\beta^{-i})\beta^p, \\ &\text{con } 0 \geq \alpha_i < \beta, \text{ per } i = 1, 2, \ldots, t, \ \alpha_1 \neq 0, \lambda \leq p \leq \omega\} \end{aligned}$$

$\mathbb{F}$ è un sottoinsieme che rappresenta una discretizzazione di $\mathbb{R}$. In altre parole, $\mathbb{F}$ è un insieme discreto di numeri presi da $\mathbb{R}$, dove ciascun numero può essere espresso al più in t cifre. Questo significa che gli elementi di $\mathbb{F}$ sono una selezione discreta di numeri reali con una precisione limitata a t cifre decimali.

Per convenzione, utilizzeremo α per scrivere i numeri reali e $\tilde{\alpha}$ per scrivere i numeri finiti.

Esempio:

Determinare e posizionare sull'asse reale gli elementi di $\mathbb{F}(2, 3, -1, 2)$.
I numeri rappresentabili possono essere espressi come:

$$\tilde{\alpha} = \pm 0.\alpha_1\alpha_2\alpha_3 \cdot 2^p$$

$$\tilde{\alpha} = \pm(\sum_{i=1}^{3} \alpha_i \cdot 2^{-i}) \cdot 2^p$$

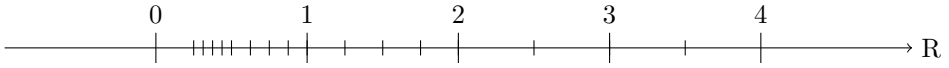
con $\tilde{\alpha} \in \mathbb{F},\ -1 \leq p < 3$ e $\alpha_1 \neq 0$
L'insieme delle possibili mantisse m_3 è dato da:

$$\begin{aligned} m_3 = \{&0.100, \\ &0.101, \\ &0.110, \\ &0.111\} \times \{2^{-1}, 2^0, 2^1, 2^2\} \end{aligned}$$

Pertanto, l'insieme degli elementi di $\mathbb{F}(2, 3, -1, 2)$ è composto da 33 elementi. Di questi, 16 sono positivi, 16 sono negativi e uno è lo zero.
Per capire come questi elementi sono posizionati sull'asse reale, li portiamo in base 10.

$$\begin{aligned} 0.100 &= 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 0 \cdot 2^{-3} = \tfrac{1}{2} = \tfrac{4}{8} \\ 0.101 &= 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = \tfrac{5}{8} \\ 0.110 &= 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} = \tfrac{3}{4} = \tfrac{6}{8} \\ 0.111 &= 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} = \tfrac{7}{8} \end{aligned}$$

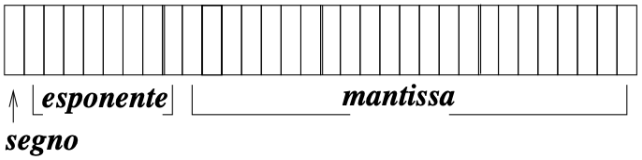
$\frac{4}{8} \cdot 2^{-1} = \frac{4}{16}$	$\frac{4}{8} \cdot 2^{0} = \frac{4}{8}$	$\frac{4}{8} \cdot 2^{1} = \frac{4}{4}$	$\frac{4}{8} \cdot 2^{2} = \frac{4}{2}$
$\frac{5}{8} \cdot 2^{-1} = \frac{5}{16}$	$\frac{5}{8} \cdot 2^{0} = \frac{5}{8}$	$\frac{5}{8} \cdot 2^{1} = \frac{5}{4}$	$\frac{5}{8} \cdot 2^{2} = \frac{5}{2}$
$\frac{6}{8} \cdot 2^{-1} = \frac{6}{16}$	$\frac{6}{8} \cdot 2^{0} = \frac{6}{8}$	$\frac{6}{8} \cdot 2^{1} = \frac{6}{4}$	$\frac{6}{8} \cdot 2^{2} = \frac{6}{2}$
$\frac{7}{8} \cdot 2^{-1} = \frac{7}{16}$	$\frac{7}{8} \cdot 2^{0} = \frac{7}{8}$	$\frac{7}{8} \cdot 2^{1} = \frac{7}{4}$	$\frac{7}{8} \cdot 2^{2} = \frac{7}{2}$



Notiamo come questi numeri sono equispaziati tra due potenze consecutive della base. Questo ci dà un'idea di come saranno fatti tutti gli insiemi di numeri finiti: tendono ad avere una densità maggiore vicino all'origine e si diradano man mano che ci si allontana da essa. La densità di questi numeri è direttamente influenzata dall'esponente negativo. Pertanto, è cruciale trovare un equilibrio tra numeri con esponenti sia positivi che negativi.

I numeri finiti sui calcolatori vengono rappresentati seguendo uno standard, come l'**ANSI/IEEE 754-1985**, che definisce formati specifici per la rappresentazione dei numeri in base 2.
Questo standard definisce 4 formati di numeri finiti, ma solo due di essi sono rigorosamente specificati. Gli altri due formati sono lasciati alla discrezione dei produttori di processori.
Lo scopo di uno standard è garantire la portabilità del codice, così che sia possibile eseguire lo stesso programma su differenti architetture ottenendo gli stessi risultati.
Gli n bit consecutivi dedicati per la memorizzazione di un numero finito vengono suddivisi tra le t cifre della mantissa ed un certo numero di bit $(\omega - \lambda + 1)$ per l'esponente p, più un bit per il segno del numero. Alcune tipiche rappresentazioni sono:

Basic precisione single	$\mathbb{F}(2, 24, -127, 128)$	32 bit
Basic precisione double	$\mathbb{F}(2, 53, -1023, 1024)$	64 bit



In precisione singola vengono destinati 24 bit alla mantissa (in realtà solo 23[1]) e 8 all'esponente ($2^8 = 256 = \omega - \lambda + 1$, con $\lambda = -127$ e $\omega = 128$), mentre in precisione doppia le cifre della mantissa sono 53 (memorizzati 52 bit) e dell'esponente 11 ($2^{11} = 2048 = \omega - \lambda + 1$, con $\lambda = -1023$ e $\omega = 1024$).
Si osservi che l'esponente è memorizzato per traslazione (*exponent biased*) e che la costante di traslazione (*bias*) è $-\lambda$. Quindi, se p è l'esponente del numero e $\tilde{p}$ è l'esponente memorizzato, possiamo trovare l'esponente memorizzato a partire dall'esponente originale utilizzando la seguente relazione:

$$\tilde{p} = p - \lambda$$

Dato un numero reale non nullo, α, per associare un numero finito ad esso, procediamo come segue:

1. **Rappresentazione esatta**. Se α è scritto nella forma $\alpha = \pm(.\alpha_1\alpha_2\ldots) \times \beta^p$ tale che $\lambda \leq p \leq \omega$, $\alpha_i = 0$ per $i > t$, allora è rappresentabile esattamente come un numero finito t di cifre e $\alpha \in \mathbb{F}(\beta, t, \lambda, \omega)$.

2. **Rappresentazione approssimata**. Altrimenti $\alpha \notin \mathbb{F}(\beta, t, \lambda, \omega)$ e quindi bisogna associargli un numero approssimato $\tilde{\alpha}$ che indicheremo con $fl(\alpha)$. Si hanno i seguenti casi:

   - $p \notin [\lambda, \omega]$, viene segnalata una condizione d'errore:

     $$\begin{array}{ll} p < \lambda & \textit{underflow} \\ p > \lambda & \textit{overflow} \end{array}$$

   - $p \in [\lambda, \omega]$, ma le cifre a_i con $i > t$ non sono tutte nulle, allora viene assegnato un numero finito $fl(\alpha)$ seguendo due possibili criteri:
     - **Troncamento** di α alla t-esima cifra

       $$fl_T(\alpha) = \pm(\sum_{i=1}^{t} \alpha_i \beta^{-i})\beta^p$$

     - **Arrotondamento** di α alla t-esima cifra

       $$fl_A(\alpha) = \pm fl_T((\sum_{i=1}^{t+1} \alpha_i \beta^{-i} + \frac{\beta}{2}\beta^{-(t+1)})\beta^p)$$

Esempio:

Il numero $\alpha = (0.11011)_2$ ha una mantissa di lunghezza 5, che è più lunga delle 3 cifre consentite in $\mathbb{F}(2, 3, -1, 2)$. Quindi, procediamo con l'operazione di arrotondamento:

$$\begin{array}{rll} fl_A(\alpha) = & 0.11011 & + \\ & 0.00010 & = \\ \hline & 0.11100 & \end{array}$$

[1]Essendo sempre α_1=1 per la rappresentazione binaria, la prima cifra può essere sottintesa senza mai essere fisicamente memorizzata.

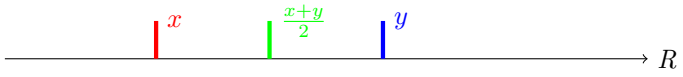
Esempio:

Consideriamo l'insieme dei numeri finiti $\mathbb{F}(10, 5, -50, 49)$. Per rappresentare un numero finito in questo insieme in memoria, dobbiamo definire il numero di posizioni necessarie. Nello specifico:

- **Segno**: una posizione è riservata per il segno. Se il numero è positivo si usa 0; se è negativo, si usa $\beta - 1$.
- **Esponente**: due posizioni sono destinate all'esponente. Usando la tecnica di memorizzazione per traslazione ($p - \lambda = \tilde{p}$), possiamo rappresentare gli esponenti da -50 a 49 attraverso valori memorizzati da 00 a 99.
- **Mantissa**: cinque posizioni sono dedicate alla mantissa.

$$\alpha = 0.0532 = 0.532 \cdot 10^{-1} \qquad fl(\alpha) = 04953200$$
$$\alpha = -237141 = -0.237141 \cdot 10^{6} \qquad fl(\alpha) = 95623714$$

Osservazione. *Siano x ed y due numeri $\in \mathbb{F}$ consecutivi positivi. Sia $\alpha \in \mathbb{R}$ tale che $x \leq \alpha < y$.*



Allora possiamo affermare che α non appartiene all'insieme $\mathbb{F}$ perché, per ipotesi, x e y sono consecutivi e non ci può essere un altro numero tra loro. Tuttavia, la rappresentazione approssimata $fl(\alpha)$ risulta essere:

$$fl_T(\alpha) = x \quad fl_A(\alpha) = \begin{cases} x & se\ \alpha < \frac{x+y}{2} \\ y & se\ \alpha \geq \frac{x+y}{2} \end{cases}$$

L'errore commesso nel troncamento sarà sempre maggiore o uguale dell'errore commesso nell'arrotondamento. Questo è il motivo per cui, con una base numerica pari, si preferisce utilizzare l'arrotondamento, poiché fornirà una migliore approssimazione del numero reale rispetto al troncamento.

La modalità di arrotondamento dello standard ANSI/IEEE-754 coincide con quella precedentemente descritta, con la particolarità dell'**arrotondamento ai pari**. Questa particolarità si applica quando un numero reale α è esattamente equidistante dai numeri finiti consecutivi x ed y, in altre parole, quando $\alpha = \frac{x+y}{2}$. In questa situazione l'arrotondamento funziona nel seguente modo:

$$fl_{AP}(\alpha) = \begin{cases} x & \text{se x è pari} \\ \text{y} & \text{se y è pari} \end{cases}$$

Sempre parlando dello standard ANSI/IEEE-754, per gestire risultati non rappresentabili, vengono utilizzati due valori speciali:

- **NaN**
- **Inf**

Invece, di avere un buco vicino allo zero dove i numeri molto piccoli verrebbero immediatamente arrotondati a zero, vengono inseriti dei numeri ulteriori per riempire questo buco e permettere ai valori di avvicinarsi progressivamente a zero. Questo meccanismo è chiamato **gradual underflow**.
Per rappresentare questi numeri estremamente piccoli, si fa uso della rappresentazione **denormalizzata**. In questa rappresentazione, la mantissa non inizia con il solito bit implicito di 1, ma con una serie di 0.

Esempio:

Si eseguano i passi necessari per rappresentare il numero reale $(-13.9)_{10}$ in un'area di memoria di 8 bit (1 per il segno, 3 per l'exponent biased e 4 per la mantissa), che permettono di memorizzare $\mathbb{F}(2, 5, -3, 4)$ per troncamento e arrotondamento.

1. **Conversione in binario**, prima la parte intera, quindi la parte decimale:

   - *Parte intera*:
     (a) Dividi il numero per 2.
     (b) Registra il resto della divisione (sarà 0 o 1).
     (c) Usa il quoziente ottenuto come nuovo numero e ripeti la divisione per 2.
     (d) Continua il processo fino a quando il quoziente diventa 0.
     (e) Leggi i resti della divisione in ordine inverso: questo sarà il numero in base 2 della parte intera.

   $$(13)_{10} = (1101)_2$$

   - *Parte decimale*:
     (a) Moltiplica la parte decimale per 2.
     (b) Registra la parte intera del risultato (sarà 0 o 1).
     (c) Usa la parte decimale del risultato come nuovo numero e ripeti la moltiplicazione per 2.
     (d) Continua questo processo finché non ottieni una parte decimale di 0 o si arriva al limite di precisione della mantissa.
     (e) Leggi i numeri interi in ordine di apparizione: questo sarà il numero in base 2 della parte decimale.

   $$0.9 \times 2 = \underline{1}.8$$
   $$0.8 \times 2 = \underline{1}.6$$
   $$0.6 \times 2 = \underline{1}.2$$
   $$0.2 \times 2 = \underline{0}.4$$
   $$0.4 \times 2 = \underline{0}.8$$
   $$(0.9)_{10} = (11100\ldots)_2$$

   da cui
   $$(-13.9)_{10} = (-1101.11100\ldots)_2$$

2. **Normalizzazione**: nello standard IEEE-754, la rappresentazione normalizzata dei numeri in virgola mobile prevede che la parte intera sia sempre 1.

   $$(-1101.11100\ldots)_2 = (-1.10111100\ldots)_2 \times 2^3$$

3. **Calcolo dell'esponente biased**:

   $$p - \lambda = \tilde{p} \to 3 - (-3) = 6$$

   $$(-1.10111100\ldots)_2 \times 2^3 = (-1.10111100\ldots)_2 \times 2^{(110)_2}$$

4. **Rappresentazione della mantissa**:

arrotondamento	troncamento
$1.10111 + 0.00001 = 1.1100$	1.1011

5. **Rappresentazione in memoria**: nello standard IEEE-754, con una mantissa di 5 bit, solo 4 bit vengono effettivamente memorizzati in memoria.

1	1	1	0	1	1	0	0

1	1	1	0	1	0	1	1

1.2 Errori di rappresentazione

Definizione. *Consideriamo un valore $\alpha \in \mathbb{R}$. Se $\alpha \notin \mathbb{F}(\beta, t, \lambda, \omega)$, allora la sua migliore approssimazione all'interno di questo insieme è data da $\tilde{\alpha} \in \mathbb{F}(\beta, t, \lambda, \omega)$. L'approssimazione di α con $\tilde{\alpha}$ introduce un* ***errore di rappresentazione****. Per quantificare tale errore, definiamo le seguenti metriche:*

$$E_{abs} = |\alpha - fl(\alpha)| \qquad \textit{errore assoluto}$$
$$E_{rel} = |\tfrac{\alpha - fl(\alpha)}{\alpha}| \ \textit{se } \alpha \neq 0 \qquad \textit{errore relativo}$$

Nel calcolo scientifico, l'errore relativo è preferito poiché fornisce una misura dell'errore "normalizzata", che non dipende dalla grandezza dei numeri confrontati.

Esempio:

Si converta quanto rappresentato nell'esempio precedente nuovamente in base 10. Successivamente, si valuti l'errore assoluto e l'errore relativo della rappresentazione.

6. **Decodifica**: per riconvertire il numero floating point appena determinato, faremo:

arrotondamento	troncamento
$(-1.1100)_2 \times 2^{(110)_2}$	$(-1.1011)_2 \times 2^{(110)_2}$
$(-1.1100)_2 \times 2^{(3)_{10}}$	$(-1.1011)_2 \times 2^{(3)_{10}}$
$(-1110.0)_2$	$(-1101.1)_2$
$-(8+4+2+0+0)_{10}$	$-(8+4+0+1+0.5)_{10}$
$-(14)_{10}$	$-(13.5)_{10}$

	arrotondamento	troncamento
errore assoluto	$-13.9 - (-14) = 0.1$	$-13.9 - (-13.5) = -0.4$
errore relativo	$\frac{0.1}{-13.9} = -0.0072$	$\frac{-0.4}{-13.9} = 0.0288$

Definizione. *Dato l'insieme dei numeri finiti $\mathbb{F}(\beta, t, \lambda, \omega)$, si dice* ***unità di arrotondamento*** *e la si indica con u, la quantità:*

$$u = \begin{cases} \beta^{1-t} & \textit{per troncamento} \\ \frac{1}{2}\beta^{1-t} & \textit{per arrotondamento} \end{cases}$$

Teorema. *Per ogni $\alpha \in \mathbb{R}$ e $\alpha \notin 0$ vale*

$$\left|\frac{\alpha - fl(\alpha)}{\alpha}\right| = u$$

Il teorema afferma che u, l'unità di arrotondamento, rappresenta il limite superiore dell'errore relativo quando si rappresenta un numero reale in un formato numerico finito.

Esempio:

Consideriamo l'insieme dei numeri finiti $\mathbb{F}(2, 5, -3, 4)$. Calcolare l'unità di arrotondamento u sia nel caso di troncamento che di arrotondamento.

$$u = \begin{cases} 2^{1-5} = \frac{1}{16} = 0.0625 & \text{per troncamento} \\ \frac{1}{2} \cdot 2^{1-5} = \frac{1}{32} = 0.0325 & \text{per arrotondamento} \end{cases}$$

Indicheremo con ϵ l'errore relativo.

Corollario. *Per ogni $\alpha \in \mathbb{R}$ e $\alpha \notin 0$ vale*

$$fl(\alpha) = \alpha(1 \pm \epsilon), \quad \textit{con } |\epsilon| < u$$

Dimostrazione.
Banalmente dato $\epsilon = \frac{\alpha - fl(\alpha)}{\alpha}$, per il Teorema si ha che $|\epsilon| < u$ e $fl(\alpha) = \alpha\epsilon + \alpha = \alpha(1 + \epsilon)$. ■

Precisione desiderata in base 10. La questione chiave è: quante cifre in base 10 sono necessarie per rappresentare con precisione ciò che è memorizzato in base 2?
Supponiamo di avere un numero rappresentato con t cifre in base 2. Vogliamo sapere a quante cifre, s, in base 10 questo corrisponde.
Partendo dall'equazione:

$$2^{-t} = 10^{-s}$$

e applicando il logaritmo in base 10 ad entrambi i lati:

$$-t \times \log_{10}(2) = -s$$

da qui possiamo isolare s:

$$s = t \times \log_{10}(2)$$

usando un'approssimazione per il logaritmo:

$$s \approx t \times 0.30103$$

Per esempio:

- Nella precisione 'basic single', con $t = 24$ cifre in base 2 per la mantissa, abbiamo:

$$s \approx 24 \times 0.30103 \approx 7.224$$

- Nella precisione 'basic double', con $t = 53$ cifre in base 2 per la mantissa, abbiamo:

$$s \approx 53 \times 0.30103 \approx 15.95459$$

Questo indica che ci servono circa 7-8 cifre in 'basic single' o circa 16 cifre in 'basic double' in base 10 per rappresentare con precisione ciò che è memorizzato in base 2. Utilizzando meno cifre, stiamo arrotondando e potremmo perdere informazioni.
Possiamo calcolare l'unità di arrotondamento u per 'basic single' e 'basic double':

- $u_{single} = \frac{1}{2} \times 2^{1-24} = 2^{-}24 \approx 5.96 \times 10^{-8}$
- $u_{double} = \frac{1}{2} \times 2^{1-53} = 2^{-}53 \approx 1.116 \times 10^{-16}$

Ora, se confrontiamo questi valori con le cifre necessarie in base 10 per una rappresentazione accurata, notiamo una relazione. Le cifre necessarie sono legate all'ordine di grandezza dell'unità di arrotondamento.
Questi valori forniscono un indicatore sull'ordine di grandezza minimo dei numeri che possono essere rappresentati accuratamente e sul numero massimo di cifre che possiamo stampare senza perdere informazioni.

1.3 Aritmetica finita

Dati due numeri a e b appartenenti a $\mathbb{F}(\beta, t, \lambda, \omega)$, l'operazione $a\ op\ b$ potrebbe produrre un risultato che non è contenuto in $\mathbb{F}(\beta, t, \lambda, \omega)$.

> *Esempio*:
>
> Siano $a = (0.34)_{10} \times 10^0$ e $b = (0.12)_{10} \times 10^{-2} \in \mathbb{F}(10, 2, \lambda, \omega)$. Eseguendo la somma si ha:
>
> $$0.34 + 0.0012 = 0.3412$$
>
> ma, $0.3412 \notin \mathbb{F}(10, 2, \lambda, \omega)$.

Per eseguire le operazioni in questo dominio, vengono definiti degli operatori specifici, che indichiamo con $\tilde{op}$ (ad esempio, $\tilde{+}, \tilde{-}, \tilde{*}, \tilde{/}$).

Definizione. *L'operatore $\tilde{op}$ tra due numeri $a, b \in \mathbb{F}$ è definito nel modo seguente:*

$$a\ \tilde{op}\ b = fl(a\ op\ b)$$

Questo significa che viene prima eseguita l'operazione in aritmetica esatta, e il risultato viene arrotondato per rientrare nell'insieme di numeri finiti $\mathbb{F}$.

Per soddisfare tali requisiti, si utilizzano dei registri posizionati vicino al processore. Questi registri, dotati di bit aggiuntivi, rispetto a quelli della mantissa, permettono di eseguire operazioni con una precisione superiore rispetto a quella raggiungibile con solo t bit. Tale maggiore precisione assicura che, arrotondando a t cifre, il risultato ottenuto aderisce alla definizione delineata sopra. Idealmente, un registro di lunghezza $t + 1$ bit, superiore a quella della mantissa stessa, garantirebbe la conformità a questa definizione.

Errore in aritmetica finita. Qual'è l'errore massimo che possiamo commettere durante un operazione con numeri finiti? Consideriamo l'errore relativo tra il risultato ottenuto in aritmetica finita e quello in aritmetica esatta, si può notare un interessante comportamento. Notiamo che, il risultato esatto di $(a\ op\ b)$ è un numero $\alpha \in \mathbb{R}$. Per definizione, in aritmetica finita, $(a\ \tilde{op}\ b)$ è l'approssimazione floating-point di α. Allora, per il teorema sopra menzionato, possiamo dedurre che l'errore relativo massimo tra α e $fl(\alpha)$ è minore dell'unità di arrotondamento u. Questo significa che u rappresenta l'errore relativo massimo che possiamo aspettarci in una singola operazione in aritmetica finita. Estendendo questa logica, se effettuiamo una serie di n operazioni, l'errore totale potrebbe essere al più nu.

$$\left| \frac{\overbrace{(a\ \tilde{op}\ b)}^{fl(\alpha)} - \overbrace{(a\ op\ b)}^{\alpha}}{\underbrace{a\ op\ b}_{\alpha}} \right| < u$$

Proprietà associativa. La proprietà associativa afferma che l'ordine in cui si raggruppano i termini durante un'operazione non modifica il risultato. Tuttavia, essa, così come le altre proprietà, non vale nell'aritmetica finita.

Esempio:

Considerati $a = 0.11 \times 10^0, b = 0.13 \times 10^{-1}, c = 0.14 \times 10^{-1} \in \mathbb{F}(10, 2, \lambda, \omega)$. Verificare se la proprietà associativa $(a\tilde{+}b)\tilde{+}c = a\tilde{+}(b\tilde{+}c)$ è valida.

$$\begin{aligned}
(0.11\tilde{+}0.013)\tilde{+}0.014 &= 0.11\tilde{+}(0.013\tilde{+}0.014) \\
fl(0.123)\tilde{+}0.014 &= 0.11\tilde{+}fl(0.027) \\
0.12\tilde{+}0.014 &= 0.11\tilde{+}0.03 \\
fl(0.134) &= 0.14 \\
0.13 \times 10^0 &\neq 0.14 \times 10^0
\end{aligned}$$

Concludendo, la proprietà associativa non è valida nell'ambito dell'aritmetica finita.

Questo significa che la stessa istruzione o espressione, se scritta in modi diversi, può produrre risultati differenti. Di conseguenza, è importante comprendere come evitare di scrivere operazioni che potrebbero causare errori più grandi.

1.3.1 Caratterizzazione di u

L'unità di arrotondamento u ha un'importanza numerica sia in relazione alla precisione di rappresentazione (Teorema) che in termini di precisione di calcolo (risultato precedente). La sua importanza numerica è ulteriormente sottolineata dalla seguente caratterizzazione:

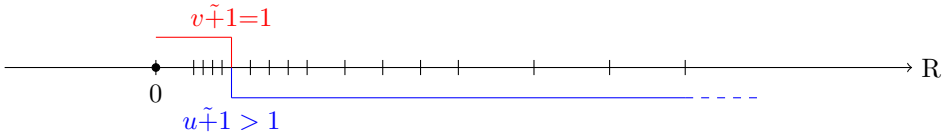
u è il più piccolo numero finito positivo tale che, se sommato a 1, viene "sentito" e risulta essere $>$ di 1.

$$u\tilde{+}1 > 1$$

Questo implica che per ogni numero finito $v < u$ sarà $v\tilde{+}1 = 1$.
Infatti, se sommiamo un numero v a 1:

$$\underbrace{1.0\ldots0}_{t \text{ cifre}} + \underbrace{0.0\ldots0}_{t \text{ cifre}}1$$

Tuttavia, il valore 1 rimane fuori dalle t cifre e nella somma viene arrotondato, e non viene "sentito" poiché il risultato sarà comunque 1.



Invece, se sommiamo un numero u a 1:

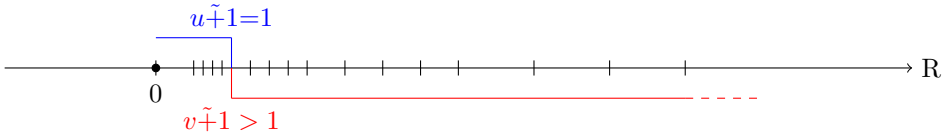
$$\begin{aligned} u\tilde{+}1 &= \frac{1}{2}\beta^{1-t} + 1 \\ &= \frac{\beta}{2}\beta^{-t} + 1 \\ &= 0.0\ldots0\frac{\beta}{2} + \underbrace{1.0\ldots0}_{t \text{ cifre}} \\ &= 1.0\ldots0\frac{\beta}{2} + 0.0 + 0.0\ldots0\frac{\beta}{2} \text{ per arrotondamento} \\ &= \underbrace{1.0\ldots1}_{t \text{ cifre}}0 \text{ per troncamento} \end{aligned}$$

$$1.0\ldots1 > 1$$

Nello standard IEEE-754, se consideriamo l'arrotondamento ai pari e dato che $1.0\ldots0\frac{\beta}{2} = \frac{x+y}{2}$, dove $x = 1$ e $y = 1.0\ldots1$, il valore pari più vicino è 1. Pertanto, 1 verrà scelto come risultato dell'arrotondamento. Questo cambia la caratterizzazione di u:

u è il più grande numero finito positivo tale che, se sommato a 1, risulta essere $= 1$.

$$u\tilde{+}1 = 1$$



Ma cosa ci serve tutto a questo? Per determinare proceduralmente u:

1. Esaminare le potenze negative di 2.
2. Continuare finché non si individua una potenza 2^{-t} che, quando sommata a 1, produce come risultato esattamente 1.
3. Tale potenza 2^{-t} rappresenta l'unità di arrotondamento u.

```
u=1
t=0
while (u+1>1)
    u=u/2
    t=t+1
end
stampa(u,t)
```

1.4 Analisi degli errori

Nella risoluzione di problemi in aritmetica finita su un calcolatore, dobbiamo prima chiederci cosa sia un "problema ben posto". Per definire un problema come **ben posto**, è necessario che soddisfi due requisiti:

- Il problema deve ammettere una e una sola soluzione.
- La soluzione deve dipendere con continuità dai dati in ingresso.

Un buon modo per modelizzare il nostro problema è tramite una funzione $f : \mathbb{R} \to \mathbb{R}$ che a partire da un dato x produce un risultato $f(x)$. Affinché il problema sia ben posto, questa funziona deve essere continua. E per la sua stessa definizione, una funzione restituisce sempre un unico output per ogni input.
Ora, consideriamo un tipico flusso di lavoro legato alla risoluzione di un problema su un calcolatore:

- **Input**. Il dato viene letto e tradotto in una rappresentazione in aritmetica finita.
- **Elaborazione**. Si applica un algoritmo, anch'esso in aritmetica finita, per elaborare il dato.
- **Valutazione del risultato**. Si esamina il risultato, l'errore associato e si decide se il risultato è accettabile oppure no.

Per valutare l'entità dell'errore, possiamo utilizzare:

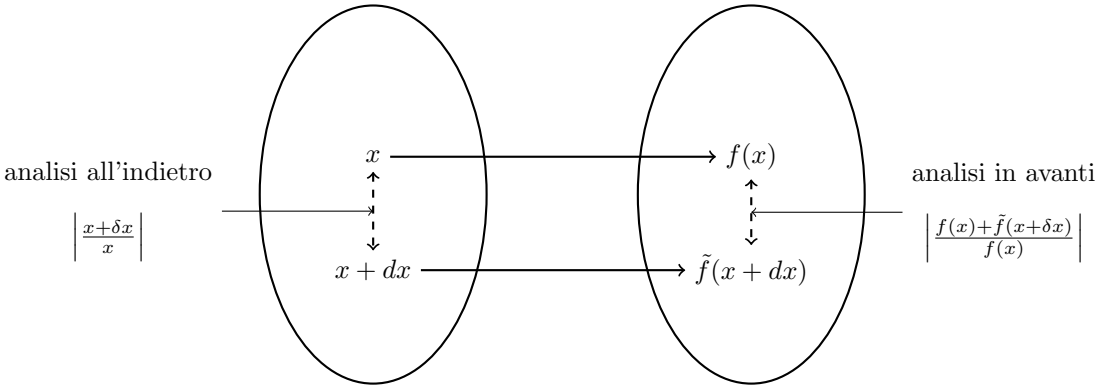


Figure 1: Analisi dell'errore

- **Analisi in avanti**: si calcola l'errore relativo sul sul risultato finale in termini degli errori introdotti dalle singole operazioni, trascurando i termini in cui compaiono prodotti di errori (analisi del 1° ordine).
- **Analisi all'indietro**: approccio opposto al precedente consiste nel considerare il risultato $\tilde{f}(x+\delta x)$ come risultato esatto derivato da dati iniziali perturbati rispetto a quelli reali. La valutazione è data quindi da un fattore δx sul dato iniziale x.

L'immagine 1 mostra che se la distanza tra x e $x+\delta x$ è piccola e, analogamente, la distanza tra $f(x)$ e $\tilde{f}(x+\delta x)$ è anch'essa piccola, allora possiamo considerare il risultato come "buono".

1.4.1 Analisi in avanti degli errori nelle operazioni di moltiplicazione e addizione

Applicando l'analisi in avanti alle operazioni di moltiplicazione e addizione si ottengono alcuni importanti risultati:

- **Moltiplicazione**: Siano $x, y \in \mathbb{R}$. Consideriamo la funzione $f : \mathbb{R}^2 \to \mathbb{R}$ definita come $f(x, y) = x \cdot y$, che moltiplica due numeri reali e restituisce il risultato.

  Applichiamo l'analisi in avanti, tenendo conto sia dell'errore sui dati che dell'errore di calcolo:

$$\begin{aligned} x \to fl(x) &= x(1+\epsilon_1) && |\epsilon_1| < u \\ y \to fl(y) &= y(1+\epsilon_2) && |\epsilon_2| < u \\ fl(x) \cdot fl(y) &= fl(fl(x)fl(y)) \\ &= fl(x)fl(y)(1+\epsilon_3) && |\epsilon_3| < u \\ &= x(1+\epsilon_1)y(1+\epsilon_2)(1+\epsilon_3) \end{aligned}$$

  Calcoliamo ora l'errore relativo:

$$\begin{aligned} \left|\frac{fl(fl(x)fl(y)) - f(x,y)}{f(x,y)}\right| &= \left|\frac{x(1+\epsilon_1)y(1+\epsilon_2)(1+\epsilon_3) - xy}{xy}\right| \\ &= \left|\frac{\cancel{(xy)}((1+\epsilon_1)(1+\epsilon_2)(1+\epsilon_3) - 1)}{\cancel{xy}}\right| \\ &= \left|(1+\epsilon_1)(1+\epsilon_2)(1+\epsilon_3) - 1\right| \\ &= \left|\cancel{1} + \epsilon_1 + \epsilon_2 + \epsilon_3 + \epsilon_1\epsilon_2 + \epsilon_1\epsilon_3 + \epsilon_2\epsilon_3 + \epsilon_1\epsilon_2\epsilon_3 - \cancel{1}\right| \end{aligned}$$

  Trascuriamo il prodotto di errori, poiché numericamente irrilevanti (analisi di 1° ordine):

$$\approx |\epsilon_1 + \epsilon_2 + \epsilon_3| \leq |\epsilon_1| + |\epsilon_2| + |\epsilon_3| < 3u$$

  Abbiamo quantificato un limite superiore per l'errore sul risultato finale. Dato che ci sono 3 operazioni e l'errore finale massimo è di $3u$, il risultato è sia accettabile che aspettato.

- **Addizione**: Siano dati $x, y \in \mathbb{R}$. Consideriamo la funzione $f : \mathbb{R}^2 \in \mathbb{R}$ definita come $f(x, y) = x + y$, che somma due numeri reali e restituisce il risultato.

Applichiamo l'analisi in avanti, tenendo conto sia dell'errore sui dati che dell'errore di calcolo:

$$\begin{aligned} x \to fl(x) &= x(1+\epsilon_1) && |\epsilon_1| < u \\ y \to fl(y) &= y(1+\epsilon_2) && |\epsilon_2| < u \\ fl(x) + fl(y) &= fl(fl(x) + fl(y)) \\ &= (fl(x)fl(y))(1+\epsilon_3) && |\epsilon_3| < u \\ &= (x(1+\epsilon_1) + y(1+\epsilon_2))(1+\epsilon_3) \end{aligned}$$

Calcoliamo ora l'errore relativo:

$$\begin{aligned} \Big|\frac{fl(fl(x)+fl(y)) - f(x,y)}{f(x,y)}\Big| &= \Big|\frac{(x(1+\epsilon_1) + y(1+\epsilon_2))(1+\epsilon_3) - (x+y)}{x+y}\Big| \\ &= \Big|\frac{(x+y+x\epsilon_1+y\epsilon_2)(1+\epsilon_3) - (x+y)}{x+y}\Big| \\ &= \Big|\frac{\cancel{(x+y)} + x\epsilon_1 + y\epsilon_2 + x\epsilon_3 + y\epsilon_3 + x\epsilon_1\epsilon_3 + y\epsilon_2\epsilon_3 - \cancel{(x+y)}}{x+y}\Big| \end{aligned}$$

Trascuriamo il prodotto di errori, poiché numericamente irrilevanti (analisi del 1° ordine):

$$\approx \Big|\frac{x}{x+y}\epsilon_1 + \frac{y}{x+y}\epsilon_2 + \frac{x+y}{x+y}\epsilon_3\Big| \leq \Big|\frac{x}{x+y}\Big||\epsilon_1| + \Big|\frac{y}{x+y}\Big||\epsilon_2| + |\epsilon_3| \not< 3u$$

Che cosa è successo? Non possiamo dire che l'errore sia sempre $< 3u$. Infatti, i fattori $\frac{x}{x+y}$ e $\frac{y}{x+y}$ agiscono come amplificatori degli errori ϵ_1 e ϵ_2. Ma quando questi fattori diventano grandi? Quando $x, y \in \mathbb{R}$ sono di segno opposto e con valori quasi uguali.

Questo fenomeno è conosciuto come **errore di cancellazione numerica**. Si tratta di una perdita di precisione durante operazioni di addizione o sottrazione. Esso si verifica quando:

- x e y sono di segno opposto e con valori quasi uguali;
- vi è un errore ϵ_1 nella rappresentazione di x o un errore ϵ_2 in quella di y.

Anche se questi fattori fossero grandi, in assenza degli errori ϵ_1 o ϵ_2, non avremmo un amplificazione dell'errore. Tuttavia, è proprio la presenza di errori nella rappresentazione, unita ai fattori di amplificazione, che può rendere il risultato finale meno preciso di quanto ci si potrebbe aspettare.

Esempio:

Sia $\mathbb{F}(10, 6, \lambda, \omega)$ con rappresentazione per arrotondamento e siano dati i numeri reali $\alpha = 0.147554326$ e $b = -0.147251742$.
La loro approssimazione nell'insieme $\mathbb{F}$ sarà: $fl(a) = 0.1475543 + 0.0000005 = 0.147554$ e $fl(b) = 0.1472517 + 0.0000005 = 0.147252$.
L'addizione esatta darà: $a+b = 0.302584 \times 10^{-3}$, mentre in aritmetica finita darà: $fl(fl(a)+fl(b)) = fl(0.147554 - 0.147252) = 0.000302 = 0.302000 \times 10^{-3}$.
L'errore relativo commesso sarà: $\frac{0.302000\times10^{-3} - 0.302584\times10^{-3}}{0.302584\times10^{-3}} \approx 0.2 \times 10^{-2}$, mentre $u = \frac{1}{2}10^{1-6} = 0.5 \times 10^{-5}$ comportando un grave errore.
Nel cancellare delle cifre a causa dell'arrotondamento, ho perso delle informazioni successive. Questa perdita si traduce in un errore che è maggiorato di ben tre ordini di grandezza.

1.4.2 Condizionamento di un problema e stabilità di un algoritmo

Ci interessa comprendere dell'errore totale, quanto di quest'ultimo sia risultato dall'approssimazione dei dati e quanto invece sia dovuto all'algoritmo che stiamo utilizzando.
Iniziamo dividendo l'errore totale in due componenti:

- **Condizionamento del problema**: Questo rappresenta l'errore che è intrinsecamente associato ai dati di input del problema. In altre parole, è l'errore che non possiamo eliminare in quanto è legato alla qualità dei dati stessi. Possiamo chiamarlo **errore inerente**. Per quantificarlo, prendiamo il dato reale, lo approssimiamo e poi valutiamo l'errore relativo tra il risultato ottenuto in aritmetica esatta utilizzando il dato approssimato e quello che avremmo ottenuto utilizzando il dato vero.

$$E_{in} = \Big|\frac{f(\tilde{x}) - f(x)}{f(x)}\Big|$$

- **Stabilità dell'algoritmo**: Questo rappresenta l'errore introdotto dall'algoritmo che stiamo utilizzando per risolvere il problema. È il contributo dell'algoritmo nell'amplificare gli errori presenti nei dati. Possiamo chiamarlo **errore algoritmico**. Per valutarlo, confrontiamo il risultato finale ottenuto utilizzando l'algoritmo in aritmetica finita con il risultato teorico che l'algoritmo avrebbero fornito operando in aritmetica esatta, considerando che dati iniziali siano già stati approssimati.

$$E_{alg} = \left| \frac{\tilde{f}(\tilde{x}) - f(\tilde{x})}{f(\tilde{x})} \right|$$

Teorema. *Siano x e $\tilde{x}$ tale che $f(x) \neq 0$ ed $f(\tilde{x}) \neq 0$. Indicati con $E_{tot} = \left| \frac{\tilde{f}(\tilde{x}) - f(x)}{f(x)} \right|$ l'errore relativo nell'analisi in avanti e con E_{in} ed E_{alg} gli errori inerente e algoritmo, si ha*

$$E_{tot} = E_{alg}(1 + E_{in}) + E_{in} \tag{1.2}$$

Trascurando il prodotto di errori, la 1.2 risulta così semplificata:

$$E_{tot} \approx E_{alg} + E_{in}$$

Un problema è definito come mal condizionato quando presenta un elevato errore inerente. Al contrario, quando l'errore inerente è ridotto, il problema è definito come ben condizionato. D'altro canto, se un algoritmo produce un ampio errore algoritmico, viene definito instabile. Se, invece, l'errore algoritmico è minimo, l'algoritmo è definito come numericamente stabile.

Esempio:

Calcolare gli errori inerente e algoritmico associati all'addizione di $x + y$, dove $x, y \in \mathbb{R}$.

$$\begin{aligned} \tilde{x} &= fl(x) = x(1+\epsilon_1) \quad |\epsilon_1| < u \\ \tilde{y} &= fl(y) = y(1+\epsilon_2) \quad |\epsilon_2| < u \end{aligned}$$

$$\begin{aligned} E_{in} &= \left| \frac{f(\tilde{x}, \tilde{y}) - f(x, y))}{f(x, y)} \right| \\ &= \left| \frac{x(1+\epsilon_1) + y(1+\epsilon_2) - (x+y)}{x+y} \right| \\ &= \left| \frac{\cancel{(x+y)} + x\epsilon_1 + y\epsilon_2 - \cancel{(x+y)}}{x+y} \right| \\ &= \left| \frac{x}{x+y}\epsilon_1 + \frac{y}{x+y}\epsilon_2 \right| \\ &\leq \left| \frac{x}{x+y} \right| |\epsilon_1| + \left| \frac{y}{x+y} \right| |\epsilon_2| \end{aligned}$$

$$\begin{aligned} E_{alg} &= \left| \frac{\tilde{f}(\tilde{x}, \tilde{y}) - f(\tilde{x}, \tilde{y})}{f(\tilde{x}, \tilde{y})} \right| \\ &= \left| \frac{(\tilde{x} + \tilde{y})(1+\epsilon_3) - (\tilde{x} + \tilde{y})}{\tilde{x} + \tilde{y}} \right| \\ &= |\epsilon_3| \end{aligned}$$

Esempio:

Si analizzi il condizionamento e la stabilità dell'algoritmo utilizzato calcolare la funzione $f : \mathbb{R} \to \mathbb{R}$ definita come $f(x) = \frac{(1+x)-1}{x}$ e dove $x \in \mathbb{F} \subset \mathbb{R}$ e $x \neq 0$.

$$\begin{aligned} \tilde{x} = fl(x) &= x(1+\epsilon_1) \qquad\qquad |\epsilon_1| < u \\ E_{in} &= \Big|\frac{f(\tilde{x}) - f(x)}{f(x)}\Big| \\ &= \Big|\frac{(1+x(1+\epsilon_1)) - 1}{x(1+\epsilon_1)} - 1\Big| \\ &= 0 \end{aligned}$$

Le operazioni principali e i loro errori associati sono:

- un'addizione con errore ϵ_1,
- una sottrazione con errore ϵ_2, e
- una divisione con errore ϵ_3.

$$\begin{aligned} E_{alg} &= \Big|\frac{\tilde{f}(\tilde{x}) - f(\tilde{x})}{f(\tilde{x})}\Big| \\ &= \Big|\frac{\frac{((1+x)(1+\epsilon_1)-1)(1+\epsilon_2)}{x}(1+\epsilon_3) - 1}{1}\Big| \\ &= \Big|\frac{\cancel{1} + x + (1+x)\epsilon_1 - \cancel{1}}{x}(1+\epsilon_2)(1+\epsilon_3) - 1\Big| \\ &= \Big|\frac{x(1+\epsilon_2)(1+\epsilon_3) + (1+x)\epsilon_1(1+\epsilon_2)(1+\epsilon_3)}{x} - 1\Big| \\ &= \Big|\cancel{1} + \epsilon_2 + \epsilon_3 + \epsilon_2\epsilon_3 + \frac{1+x}{x}\epsilon_1(1+\epsilon_2+\epsilon_3+\epsilon_2\epsilon_3) - \cancel{1}\Big| \end{aligned}$$

Trascurando il prodotto di errori, poiché numericamente irrilevanti (analisi di 1° ordine):

$$\approx \frac{1+x}{x}\epsilon_1 + \epsilon_2 + \epsilon_3$$

$$\frac{1}{x}\epsilon_1 + \epsilon_1 + \epsilon_2 + \epsilon_3 \leq \Big|\frac{1}{x}\Big||\epsilon_1| + \underbrace{|\epsilon_1| + |\epsilon_2| + |\epsilon_3|}_{3u}$$

Come interpretare questo risultato finale? Su quest'ultimo può esserci un errore $> 3u$. Se l'errore ϵ_1 nella prima addizione è $\neq 0$ e x è piccolo, il termine $\frac{1}{x}$ diventa grande, amplificando così l'errore.

1.4.3 Numero di condizione

Funzioni $f : \mathbb{R} \to \mathbb{R}$. Consideriamo una funzione $f : \mathbb{R} \to \mathbb{R}$ che sia differenziabile. Supponendo che vogliamo valutare questa funzione in un punto vicino a x_0, possiamo usare lo sviluppo di Taylor centrato in x_0:

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + o(h)$$

dove $h = x - x_0$.
Se vogliamo approssimare f in un punto vicino x_0, diciamo $\tilde{x}_0$ abbiamo:

$$f(\tilde{x}_0) = f(x_0) + f'(x_0)(\tilde{x}_0 - x_0) + o(h)$$

Calcoliamo l'errore inerente:

$$E_{in} = \left| \frac{f(\tilde{x}_0) - f(x_0)}{f(x_0)} \right|$$

Sostituendo lo sviluppo di Taylor per $f(\tilde{x}_0)$, possiamo riscrivere:

$$\approx \left| \frac{\cancel{f(x_0)} + f'(x_0)(\tilde{x}_0 - x_0) - \cancel{f(x_0)}}{f(x_0)} \right|$$

$$= \left| \frac{f'(x_0)(\tilde{x}_0 - x_0)}{f(x_0)} \cdot \frac{x_0}{x_0} \right| \qquad \text{per ipotesi } f(x_0) \neq 0 \text{ e } x_0 \neq 0$$

$$= \left| \frac{f'(x_0)x_0}{f(x_0)} \frac{\tilde{x}_0 - x_0}{x_0} \right|$$

Se si considera $\frac{\tilde{x}_0 - x_0}{x_0}$ come l'errore sui dati, possiamo riscrivere:

$$= \left| \frac{f'(x_0)x_0}{f(x_0)} \right| |\epsilon_{x_0}|$$

Ciò che emerge è che l'errore inerente non dipende soltanto dall'errore sui dati, ma anche da una quantità $\left| \frac{f'(x_0)x_0}{f(x_0)} \right|$ che amplifica tale errore. Questa quantità è nota come **numero di condizione** e lo denotiamo con $C(f, x_0)$.
Un valore elevato di $C(f, x_0)$ indica che il problema è mal condizionato in x_0, cioè piccoli errori nei dati possono portare a grandi errori nella soluzione.
Pertanto, per determinare se un problema differenziabile è mal condizionato in un dato punto, si può guardare il suo numero di condizione.

Generalizzazione per funzioni $f : \mathbb{R}^n \to \mathbb{R}$. Dati una funzione $f : \mathbb{R}^n \to \mathbb{R}$ e due vettori $x = (x_1, x_2, \ldots, x_n)$ e $\tilde{x} = (\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_n)$.
L'espansione di Taylor di funzione in più variabili può essere espressa come:

$$f(\tilde{x}) = f(x) + \sum_{i=1}^{n} (\tilde{x}_i - x_i) \frac{\delta f}{\delta x_i} + o(h)$$

dove $\frac{\delta f}{\delta x_i}$ rappresenta la derivata parziale di f rispetto alla i-esima componente e $h = \sum_{i=1}^{n} (\tilde{x}_i - x_i)$.

Calcoliamo l'errore inerente:

$$E_{in} = \left| \frac{f(\tilde{x}) - f(x)}{f(x)} \right|$$

Sostituendo lo sviluppo di Taylor per $f(\tilde{x})$, possiamo riscrivere:

$$\approx \left| \frac{\cancel{f(x)} + \sum_{i=1}^{n} (\tilde{x}_i - x_i) \frac{\delta f}{\delta x_i} - \cancel{f(x)}}{f(x)} \right|$$

$$\leq \sum_{i=1}^{n} \left| \frac{(\tilde{x}_i - x_i) \frac{\delta f}{\delta x_i}}{f(x)} \right| \qquad \text{per ipotesi } x_i \neq 0$$

$$= \sum_{i=1}^{n} \left| \frac{(\tilde{x}_i - x_i) \frac{\delta f}{\delta x_i}}{f(x)} \frac{x_i}{x_i} \right|$$

Se si considera $\epsilon_i = \frac{\tilde{x}_i - x_i}{x_i}$ come gli errori sui dati e le quantità $c_i = \frac{\frac{\delta f}{\delta x_i} x_i}{f(x)}$ i numeri di condizione, possiamo riscrivere:

$$= \sum_{i=1}^{n} |c_i \epsilon_i|$$

Ricapitolando:

$$\left| \frac{f(\tilde{x}) - f(x)}{f(x)} \right| \leq \sum_{i=1}^{n} |c_i \epsilon_i| \tag{1.3}$$

dove

$$c_i = \frac{\frac{\delta f}{\delta x_i} x_i}{f(x)} \tag{1.4}$$

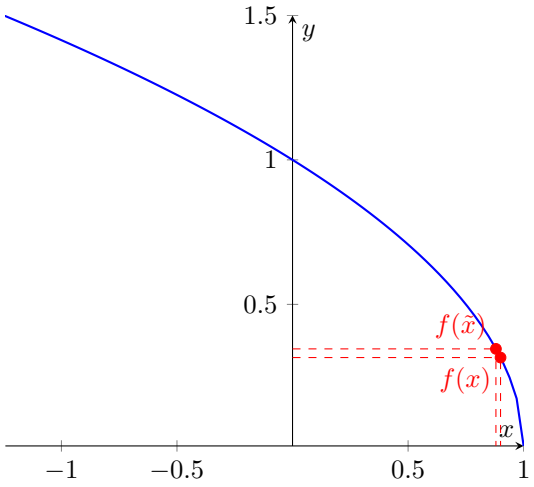
Se anche uno solo di questi numeri di condizione è grande, l'errore inerente sarà significativo. Per avere un errore inerente piccolo, è necessario che tutti i numeri di condizione siano piccoli.
Inoltre, grazie a quanto abbiamo visto, se la funzione è differenziabile, saremo in grado di stimare l'errore inerente più velocemente.

Esempio:

Consideriamo la funzione $f : \mathbb{R} \to \mathbb{R}$ definita come $f(x) = \sqrt{1-x}$, con $x \in \mathbb{R}$ e $x < 1$. La sua derivata è data da $f'(x) = -\frac{1}{2\sqrt{1-x}}$.
Calcoliamo il numero di condizione:

$$\left|\frac{xf'(x)}{f(x)}\right| = \left|\frac{x}{\sqrt{1-x}}(-\frac{1}{2\sqrt{1-x}})\right| = \left|\frac{x}{2(1-x)}\right|$$

Guardando questa espressione, notiamo che quando x si avvicina a 1, il numero di condizione aumenta rapidamente, suggerendo che il problema è mal condizionato in prossimità di $x = 1$.



Dal grafico, possiamo vedere che quando x si avvicina molto a 1, anche piccole variazioni in x possono causare variazioni significative in $f(x)$. Ciò significa che errori piccoli in x possono portare a errori grandi in $f(x)$.
Per avere un'idea numerica di questo comportamento, consideriamo l'insieme $\mathbb{F}(10, 4, \lambda, \omega)$.
Siano:

$$x_0 = 0.99984$$
$$\tilde{x}_0 = 0.9998$$

Calcoliamo l'errore sui dati:

$$\left|\frac{\tilde{x}_0 - x_0}{x_0}\right| = \left|\frac{0.9998 - 0.99984}{0.99984}\right| \approx 4 \times 10^{-5}$$

L'errore inerente è dato da:

$$E_{in} = \left|\frac{f(\tilde{x}_0) - f(x_0)}{f(x_0)}\right| = \left|\frac{0.014142 - 0.0126491}{0.0126491}\right| \approx 0.1180$$

Il numero di condizione in x_0 è:

$$C(f, x_0) = C(\sqrt{1-x}, 0.99984) = \left|\frac{0.99984}{2(1-0.99984)}\right| \approx 0.312 \times 10^4$$

Osserviamo che l'errore sui dati si amplifica di 4 ordini di grandezza, a causa del numero di condizione, causando un grave errore sul risultato.

Esempio:

Si stimi l'errore inerente applicando 1.3 nei casi già visti di moltiplicazione e addizione fra numeri reali:

- **moltiplicazione** $f(x_1, x_2) = x_1 \cdot x_2$; applicando la 1.4 si ha

  $$c_1 = \frac{x_1}{x_1 x_2} = 1 \quad c_2 = \frac{x_2}{x_1 x_2} x_1 = 1$$

  da cui si deduce che il problema in oggetto è ben condizionato e risulta

  $$E_{in} = \left| \frac{f(\tilde{x}) - f(x)}{f(x)} \right| \leq \sum_{i=1}^{2} |c_i \epsilon_i| = |c_1 \epsilon_1| + |c_2 \epsilon_2| = 1\,|\epsilon_1| + 1\,|\epsilon_2|$$

- **addizione** $f(x_1, x_2) = x_1 + x_2$; applicando la 1.4 si ha

  $$c_1 = \frac{x_1}{x_1 + x_2} 1 \quad c_2 = \frac{x_2}{x_1 + x_2} 1$$

  da cui si deduce che il problema è mal condizionato per $x_1 + x_2 \to 0$ e risulta

  $$E_{in} = \ldots = \left| \frac{x_1}{x_1 + x_2} \right| |\epsilon_1| + \left| \frac{x_2}{x_1 + x_2} \right| |\epsilon_2|$$

Errore analitico. Supponiamo di avere una funzione $g : \mathbb{R} \to \mathbb{R}$ o $g : \mathbb{R}^n \to \mathbb{R}$ che non può essere risolta in aritmetica reale. Un esempio tipico potrebbe essere una funziona che calcola il seno di un numero, per la quale un calcolatore potrebbe non avere una definizione diretta.
Per trattare questo problema computazionalmente, possiamo sostituire $g(x)$ con una sua approssimazione $f(x)$ che, al contrario, è direttamente calcolabile. Un modo comune per approssimare questa funzione è utilizzare lo sviluppo di Taylor.
L'errore introdotto nell'usare $f(x)$ al posto di $g(x)$ è ciò che chiamiamo **errore analitico** ed è definito come:

$$E_{an} = \left| \frac{f(x) - g(x)}{g(x)} \right|$$

L'errore analitico si verifica quando si approssima un problema *irrazionale* (cioè, un problema non risolvibile in aritmetica reale) con un problema *razionale* o direttamente calcolabile.

Esempio:

Si esamina il problema della valutazione della seguente funzione:

$$f(x_1, x_2) = \sqrt{x_1 + x_2} - \sqrt{x_1} \quad \text{con } x_1, x_1 + x_2 \geq 0$$

Esistono delle condizioni dei dati in ingresso che possono causare errori gravi e che possono invalidare il risultato?

1. Se x_2 è estremamente piccolo in confronto a x_1, si incorre in un errore di cancellazione numerica.
2. Se x_1 e x_2 sono di segno opposto e con valori quasi uguali, si incorre in un errore di cancellazione numerica.

Alla luce di ciò, si cerca un algoritmo differente che eviti il problema. Razionalizzando, otteniamo:

$$\begin{aligned} f(x_1, x_2) &= (\sqrt{x_1 + x_2} - \sqrt{x_1}) \frac{\sqrt{x_1 + x_2} + \sqrt{x_1}}{\sqrt{x_1 + x_2} + \sqrt{x_1}} \\ &= \frac{x_2}{\sqrt{x_1 + x_2} + \sqrt{x_1}} \end{aligned}$$

Ora, a denominatore si effettua un addizione fra quantità non negative. Questo elimina il rischio di cancellazione per la 1° condizione, ma non per la 2° condizione.
Per convincerci di ciò, analizziamo l'errore inerente. Utilizziamo la stima 1.3: nel caso specifico sarà $E_{in} = c_1\epsilon_1 + c_2\epsilon_2$ con c_1 e c_2 calcolati tramite la 1.4; facendo i conti si ottiene:

$$c_1 = -\frac{1}{2}\sqrt{\frac{x_1}{x_1 + x_2}} \quad c_2 = \frac{1}{2}(1 + \sqrt{\frac{x_1}{x_1 + x_2}}) = \frac{1}{2} - c_1$$

e il problema risulta mal condizionato proprio quando $x_1 + x_2 \to 0$.
Illustriamo la situazione con un esempio numerico, dove con f denotiamo il valore esatto, mentre con $\tilde{f}$ e $\hat{f}$, rispettivamente i risultati dei due algoritmi lavorando in $\mathbb{F}(10, 7, \lambda, \omega)$.

- Siano dati $x_1 = 0.1 \times 10^1$ e $x_2 = 0.1 \times 10^{-3}$, allora $c_2 \approx 1$ che indica un buon condizionamento in questo caso. I risultati esatto e calcolati con i due algoritmi sono:

  $$\begin{aligned} f(x_1, x_2) &= 0.4999875 \times 10^{-6} \\ \tilde{f}(x_1, x_2) &= 0.4994869 \times 10^{-6} \\ \hat{f}(x_1, x_2) &= 0.4999875 \times 10^{-6} \end{aligned}$$

  che indica l'instabilità del primo algoritmo nel caso di $|x_2| << |x_1|$.

- Siano dati $x_1 = 1$ e $x_2 = -1 + 10^{-6} = 0.999999$, allora $c_2 \approx 0.5 \times 10^-3$ che indica un cattivo condizionamento, infatti $x_1 + x_2 \to 0$. I risultati e satto e calcolati con i due algoritmi sono:

  $$\begin{aligned} f(x_1, x_2) &= -0.999 \\ \tilde{f}(x_1, x_2) = \hat{f}(x_1, x_2) &= -0.9985858 \end{aligned}$$

  che mostra come entrambi gli algoritmi ci restituiscono risultati scadenti.

- Siano dati $x_1 = 0.1$ e $x_2 = 0.2$, allora $c_2 \approx 0.85$ che indica un buon condizionamento anche in questo caso. I risultati esatto e calcolati con i due algoritmi sono:

  $$f(x_1, x_2) = \tilde{f}(x_1, x_2) = \hat{f}(x_1, x_2) = 0.1309858$$

2 Funzioni polinomiali

Definizione. *Una funzione $p : \mathbb{R} \to \mathbb{R}$ definita da*

$$p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n = \sum_{i=0}^{n} a_i x^i \tag{2.1}$$

è detta funzione polinomiale, dove n è un intero non negativo detto grado e $a_0, a_1, \ldots, a_n$ sono numeri reali fissati detti coefficienti. Inoltre,

- *se $a_n \neq 0$, si dice che $p(x)$ ha grado n;*
- *se tutti i coefficienti a_i, sono nulli, allora $p(x)$ è detto polinomio nullo.*

Con $\mathbf{P}_n$ si denota l'insieme di tutte le funzioni polinomiali con grado $\leq n$, insieme al polinomio nullo. $\mathbf{P}_n$ è uno spazio vettoriale di dimensione $n+1$. In uno spazio vettoriale di dimensione $n+1$, è necessario identificare $n+1$ funzioni polinomiali linearmente indipendenti per rappresentare univocamente tutte le altri funzioni all'interno di tale spazio come combinazione lineare di queste $n+1$ funzioni base.
È importante sottolineare che esistono infinite possibili basi per uno spazio vettoriale come $\mathbf{P}_n$.
Nel contesto del calcolo numerico, la scelta della base è fondamentale. Basi diverse corrispondono a coefficienti diversi, che possono influenzare l'errore inerente durante i calcoli.

Osservazione. *Un polinomio è una funzione razionale, direttamente calcolabile su un calcolatore.*

Teorema (fondamentale dell'algebra)**.** *Sia $p(x)$ un polinomio di grado $n \geq 1$. Allora, $p(x)$ ha esattamente n radici reali o complesse, contate con la sua molteplicità. Ciò significa che $p(x) = 0$ può essere riscritto come:*

$$p(x) = a_n (x - \alpha_1)^{m_1} (x - \alpha_2)^{m_2} \ldots (x - \alpha_k)^{m_k} \tag{2.2}$$

dove:

- *α_i (per $i = 1, \ldots, k$) sono le radici distinte del polinomio,*
- *m_i (per $i = 1, \ldots, k$) rappresenta la molteplicità della radice α_i,*
- *e la somma totale delle molteplicità è $m_1 + m_2 + \ldots + m_k = n$.*

Il teorema fondamentale dell'algebra ci fornisce un altro modo per esprimere il polinomio.

Teorema. *Siano $a(x)$ e $b(x)$ polinomi (dove $b(x)$ non è il polinomio nullo); allora è sempre possibile dividere $a(x)$ per $b(x)$ in modo da ottenere un quoziente $q(x)$ ed un resto $r(x)$. In altre parole, ogni polinomio $a(x)$ può essere espresso nella forma:*

$$a(x) = q(x)b(x) + r(x)$$

con $r(x) = 0$ o $r(x)$ con grado minore di quello di $b(x)$.

2.1 Valutazione di un polinomio

Il primo problema che affronteremo riguarda la valutazione di un polinomio. Ciò significa determinare il valore del polinomio per un assegnato valore $\bar{x}$.
Formalmente, data la funzione:

$$f(a_0, a_1, \ldots, a_n, \bar{x}) = a_0 + a_1 \hat{x} + \ldots + a_n \bar{x}^n$$

dove $f : \mathbb{R}^{n+2} \to \mathbb{R}$, vogliamo trovare il risultato di:

$$p(\bar{x}) = f(a_0, a_1, \ldots, a_n \bar{x})$$

In altre parole, inserendo i dati $a_0, a_1, \ldots, a_n$ e un valore specifico $\bar{x}$, vogliamo calcolare il valore del polinomio per quel valore.
Considerando una funzione polinomiale di grado n nella sua rappresentazione canonica 2.1, il metodo più immediato per la sua valutazione in corrispondenza di un assegnato valore $\bar{x}$ può essere come segue:

```
p=a[0]
s=1
for k=1..n
    s=s*x_bar
    p=p+a[k]*s
```

Il calcolo di $p(\bar{x})$ richiede $2n$ moltiplicazioni ed n addizioni, con un costo asintotico $\mathcal{O}(n)$. Possiamo fare meglio? Se si scrive il polinomio 2.1 nella seguente forma:

$$\begin{aligned} p(x) &= a_0 + x(a_1 + a_2x + \ldots + a_{n-1}x^{n-2} + a_nx_{n-1}) \\ &= a_0 + x(a_1 + x(a_2 + a_3x + \ldots + a_{n-1}x^{n-3} + a_nx^{n-2})) \\ &\vdots \\ &= a_0 + x(a_1 + x(a_2 + \ldots + x(a_{n-1} + xa_n)\cdots)) \end{aligned} \tag{2.3}$$

si ricava un differente metodo dovuto ad Horner:

```
p=a[n]
for k=n-1..0
    p=a[k]+x_bar*p
```

L'algoritmo di Horner richiede n moltiplicazioni ed n addizioni, con un costo asintotico $\mathcal{O}(n)$. Ancora, possiamo fare meglio?
Richiamando il teorema 2.2 sulla divisione di due polinomi, nel caso particolare in cui il polinomio divisore sia il binomio $(x - \bar{x})$ per un assegnato valore reale $\bar{x}$, si ha che esistono unici i polinomi $q(x)$ ed $r(x)$ per cui

$$p(x) = q(x)(x - \bar{x}) + r(x)$$

e poiché $r(x)$ deve essere di grado inferiore a quello di $(x - \bar{x})$ sarà una costante che indicheremo con r.

Teorema. *Il resto della divisione del polinomio $p(x)$ per $(x - \bar{x})$ è $p(\bar{x})$.*

Dal teorema appena enunciato si deduce che un metodo per valutare un polinomio $p(x)$ in un punto $\bar{x}$ consiste nel determinare il resto della divisione fra $p(x)$ e il binomio $(x - \bar{x})$. A tal fine è ben nota la regola di Ruffini, la quale viene applicata seguendo il seguente schema di calcolo:

	a_n	a_{nb-1}	$\ldots$	$\ldots$	a_2	a_1	a_0
$\bar{x}$		$\bar{x}b_n$	$\bar{x}b_{n-1}$	$\ldots$	$\bar{x}b_3$	$\bar{x}b_2$	$\bar{x}b_1$
	b_n	b_{n-1}	$\ldots$	$\ldots$	b_2	b_1	$r = p(x)$

- **Preparazione**:
  - Scrivi i coefficienti del polinomio $p(x)$ in ordine decrescente di grado sulla prima riga. Se manca un termine di un certo grado, includi un coefficiente 0 per quel grado.
  - Senza effettuare alcuna operazione, porta giù il primo coefficiente.
- **Procedimento**:
  - Procedi con la compilazione della seconda riga e della terza riga. Moltiplica l'elemento appena riportato nella terza riga per il valore assegnato $\bar{x}$. Scrivi il risultato nella seconda riga, nella colonna successiva.
  - Somma il coefficiente della prima riga con l' elemento presente nella seconda riga della colonna corrispondente e riporta il risultato nella terza riga della stessa colonna.

Reiterando il procedimento arriviamo all'ultimo elemento a destra sulla terza riga, che rappresenta il resto.
Le operazioni fatte sono:

$$\begin{aligned} b_n &= a_n \\ b_{n-1} &= a_{n-1} + \bar{x}b_n \\ b_{n-2} &= a_{n-2} + \bar{x}b_{n-1} \\ &\vdots \\ r &= a_0 + \bar{x}b_1 \end{aligned}$$

Come si può osservare, questo schema si traduce esattamente nello pseudocodice di Horner. Tuttavia, mentre Horner serve solamente a valutare il polinomio, Ruffini può essere anche utilizzato anche per valutare la derivata del polinomio.

2.1.1 Valutazione numerica della derivata

Se siamo interessati solo al valore della derivata in $\bar{x}$, allora, in modo più economico si può procedere nel seguente modo: per quanto detto nella sezione precedente, possiamo riscrivere $p(x)$ come

$$p(x) = q(x)(x - \bar{x}) + p(\bar{x})$$

derivando tale espressione si ha

$$p'(x) = q'(x)(x - \bar{x}) + q(x)$$

e valutandola in $\bar{x}$

$$\begin{aligned} p'(\bar{x}) &= q'(\bar{x}) \underbrace{(\bar{x} - \bar{x})}_{0} + q(\bar{x}) \\ &= q(\bar{x}) \end{aligned}$$

dove $q(x)$ ed i suoi coefficienti sono quelli che si ottengono applicando l'algoritmo di Ruffini per valutare $p(x)$ in $\bar{x}$.

```
p=a[n]
p1=0
for k=n-1..0
    p1=p+x_bar*p1
    p=a[k]+x_bar*p
```

analogamente si possono calcolare anche le derivate di ordine superiore. Derivando l'espressione ottenuta per $p'(x)$ si ottiene:

$$p''(x) = q''(x)(x - \bar{x}) + 2q'(x)$$

e valutando questa espressione in $\bar{x}$

$$\begin{aligned} p''(\bar{\bar{x}}) &= q''(\bar{x}) \underbrace{(\bar{x} - \bar{x})}_{0} + 2q'(\bar{x}) \\ &= 2q'(\bar{x}) \end{aligned}$$

Allora per ottenere $p''(\bar{x})$ è sufficiente calcolare $q'(\bar{x})$ che possiamo ottenere da Ruffini utilizzato per valutare $p'(x)$ in $\bar{x}$.

Esempio:

Dato il polinomio $p(x) = 1 + x - 2x^2 + 3x^4$, vogliamo calcolare $p(2)$, $p'(2)$ e $p''(2)$. Per fare ciò, possiamo utilizzare la regola di Ruffini:

	3	0	−2	1	1
2		6	12	20	42
	3	6	10	21	$43 = p(2)$

	3	6	10	21
2		6	24	68
	3	12	34	$89 = p'(2)$

	3	12	34
2		6	36
	3	18	$70 \to 2 * 70 = 140 = p''(2)$

Stima dell'errore algoritmico del metodo di Horner/Ruffini. Procediamo con un'analisi in avanti per determinare l'errore algoritmico associato al metodo di Horner/Ruffini. Questo metodo è utilizzato per calcolare il valore di un polinomio di grado n nella base canonica in un punto $\bar{x}$, rappresentato come $f(a_1, a_2, \ldots, a_n, x) = p(x)$, dove $f : \mathbb{R}^2 \to \mathbb{R}$. Si ottiene:

$$E_{alg} = \left| \frac{\tilde{f}(\tilde{a}_0, \tilde{a}_1, \ldots, \tilde{a}_n, \tilde{x}) - f(\tilde{a}_0, \tilde{a_1}, \ldots, \tilde{a}_n, \tilde{x})}{f(\tilde{a}_0, \tilde{a}_1, \ldots, \tilde{a}_n, \tilde{x})} \right| \leq \frac{\gamma 2n}{|p(\tilde{x})|} \sum_{i=0}^{n} \left| a_i \tilde{x}^i \right|$$

con

$$\gamma 2n \leq 2.01nu$$

Se ci limitassimo a considerare soltanto quest'ultimo, potremmo considerare l'algoritmo come stabile, dato che cresce linearmente con il grado del polinomio. Però, se i coefficienti o i valori di x fossero particolarmente grandi, ciò potrebbe causare un incremento significativo nell'errore algoritmico.

Stima dell'errore inerente del metodo di Horner/Ruffini. Assegnato un polinomio ed un punto in cui valutarlo, l'errore inerente misura a piccole variazione sui coefficienti (dati), come varia, in senso relativo, il valore del polinomio (risultato).

Esempio:

Sia assegnato il polinomio

$$p(x) = a_0 + a_1 x = 100 - x$$

e lo si voglia valutare in punti $\bar{x} \in [100, 101]$. Si perturba il coefficiente a_1 dell'1%, cioè $\left|\frac{\tilde{a}_1 - a_1}{a_1}\right| = \frac{1}{100}$; segue che $\tilde{a}_1 = a_1 \pm \frac{1}{100} a_1$, per cui il polinomio perturbato risulta:

$$\tilde{p}(x) = 100 - (1 - \frac{1}{100})x = 100 - \frac{99}{100}x$$

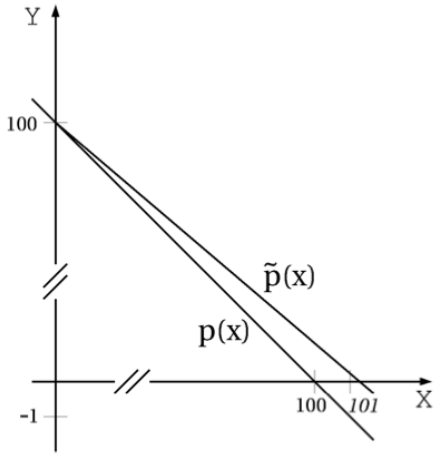
Valutandoli in $x = 101$ si ha:

$$p(101) = -1 \quad \tilde{p} = 0.01$$

commettendo un errore relativo dato da:

$$\left|\frac{\tilde{p}(101) - p(101)}{p(101)}\right| = 101\frac{1}{100}$$

Risulta, quindi che una perturbazione iniziale dell'1% porta una variazione sul risultato del 101%.



La causa di questa amplificazione dell'errore è evidente nel grafico sopra riportato. In pratica il coefficiente a_1 rappresenta l'inclinazione della retta la quale, anche se alterata minimamente, comporta grossi errori per punti lontani dall'origine.
Lo stesso comportamento si ottiene perturbando anche solo a_0, entrambi i coefficienti o valutando in altri punti dell'intervallo.

Procediamo, per questo esempio, all'analisi dell'errore inerente mediante la stima 1.3:

$$\begin{aligned} E_{in} &= \left|\frac{f(\tilde{a}_0, \tilde{a}_1, \tilde{x}) - f(a_0, a_1, x)}{f(a_0, a_1, x)}\right| \leq |c_0\epsilon_0| + |c_1\epsilon_1| + |c_x\epsilon_x| \\ &= \left|\frac{a_0}{p(x)}\epsilon_0\right| + \left|\frac{a_1 x}{p(x)}\epsilon_1\right| + \left|\frac{x\alpha_1}{p(x)}\epsilon_x\right| \end{aligned}$$

e nel caso $a_0 = 100$, $a_1 = -1$ e $x \in [100, 101]$ con, per esempio, $x = 101$ si ha:

$$= \left|\frac{100}{-1}\epsilon_0\right| + \left|\frac{-101}{-1}\epsilon_1\right| + \left|\frac{-101}{-1}\epsilon_x\right|$$

da qui si vede come il problema è mal condizionato. Infatti una minima perturbazione su uno dei coefficienti, per esempio, a_1 dell'1% ($\epsilon_0 = 0, \epsilon_1 = 1/100, \epsilon_x = 0$), fa sì che l'errore relativo assuma il valore $101\frac{1}{100}$ e cioè 101 volte maggiore di quello sul dato iniziale.
Dall'analisi fatta si evince che, per qualunque punto $x \in [100, 101]$, questo comportamento sarà inevitabile in quanto una lieve modifica dei coefficienti (ϵ_0 o $\epsilon_1 \neq 0$) viene grandemente amplificata (coefficienti c_0 e c_1).

Generalizzando l'analisi fatta in questo esempio alla valutazione di un generico polinomio $p(x)$ nella base canonica l'errore inerente si può rappresentare come la somma di due componenti:

$$\begin{aligned} E_{in} &\leq \left|\frac{a_0}{p(x)}\epsilon_0\right| + \left|\frac{a_1 x}{p(x)}\epsilon_1\right| + \left|\frac{a_2 x^2}{p(x)}\epsilon_2\right| + \ldots + \left|\frac{a_n x^n}{p(x)}\epsilon_n\right| + \left|\frac{xp'(x)}{p(x)}\epsilon_x\right| \\ &= E_{in1} + E_{in2} \end{aligned}$$

Si può desumere che:

- E_{in1} dipende da $p(x)$ e dai valori $a_i x^i$. Questi termini dipendono dalla base di rappresentazione;
- E_{in2} dipende dal valore di x, dal polinomio in esame $p(x)$ e dalla sua derivata, per cui è un errore che non dipende dalla base di rappresentazione.

Pertanto, l'espressione di un polinomio può incidere sull'errore inerente associato alla sua valutazione. Nel seguito vogliamo esaminare se è possibile cambiare la base di rappresentazione al fine di ridurre l'errore inerente. Consideriamo una base di $\mathbb{P}_n$ costituita da $n+1$ polinomi linearmente indipendenti come $\{\phi_0(x), \phi_1(x), \ldots, \phi_n(x)\}$; allora un generico polinomio $p(x)$ di grado al massimo n può essere espresso come una combinazione lineare di questi polinomi:

$$p(x) = \sum_{i=0}^{n} b_i \phi_i$$

Se ripetiamo l'analisi fatta per determinare E_{in} nel caso che $p(x)$ sia rappresentato nella base $\{\phi_i\}$, avremo:

$$E_{in} \leq \sum_{i=0}^{n} |c_i \epsilon_i| + |c_x \epsilon_x|$$

con

$$C_i = \frac{b_i \phi_i(x)}{p(x)} \quad C_x = \frac{xp'(x)}{p(x)}$$

L'idea è che, identificando una base di rappresentazione con valori "piccoli", potremmo ottenere un miglioramento sull'errore inerente E_{in1}.

> *Esempio*:
>
> Un primo esempio è rappresentato dalla base con centro $\{1, (x-c), (x-c)^2, \ldots, (x-c)^n\}$ con $c \in [a, b]$.

2.2 Polinomi nella base di Bernstein

Si potrebbe chiedersi se esista una base che, indipendentemente dal polinomio considerato, il problema sia sempre ben condizionato. La risposta, in generale, è no: spesso la scelta della base ottimale dipende dallo specifico problema. Tuttavia, esiste una base, la base di Bernstein, che si distingue come particolarmente efficace e versatile rispetto ad altre.

Definizione. *Con funzione polinomiale in forma di Bernstein si intende un espressione del tipo*

$$p(x) = \sum_{i=0}^{n} b_i B_{i,n}(x) \quad \textit{con } x \in [a, b]$$

dove i $B_{i,n}(x)$ sono i polinomi base di Bernstein definiti sull'intervallo $[a, b]$ da

$$B_{i,n}(x) = \binom{n}{i} \frac{(b-x)^{n-i}(x-a)^i}{(b-a)^n} \tag{2.4}$$

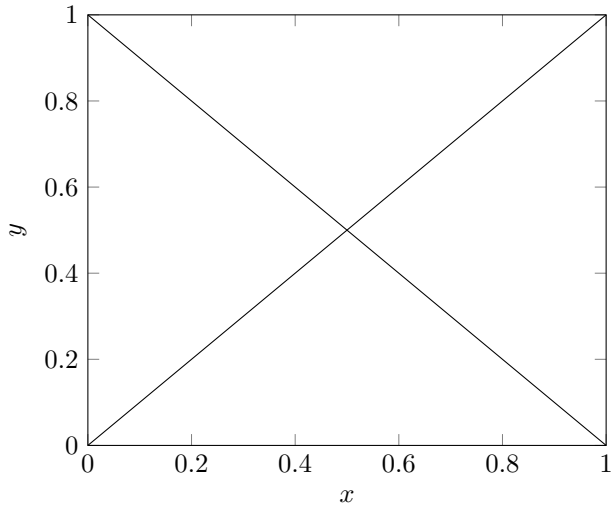
e $b_0, b_1, \ldots, b_n \in \mathbb{R}$ sono i coefficienti nella base di Bernstein.

con $\binom{n}{i} = \frac{n!}{i!(n-i)!}$.

- **n=1**

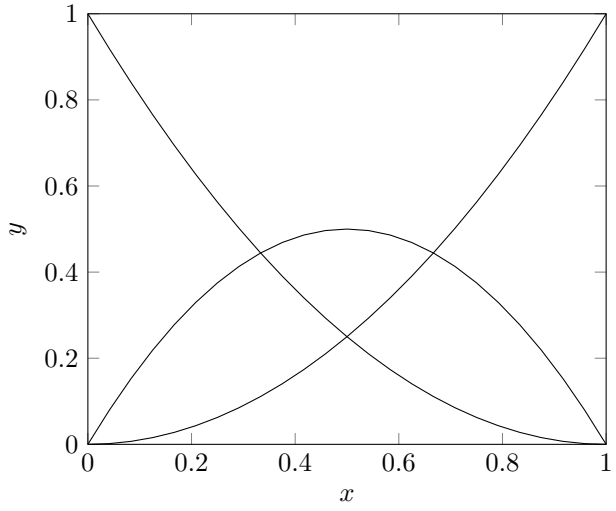
$$B_{0,1}(x) = \binom{1}{0} \frac{(b-x)^1(x-a)^0}{b-a} = \frac{b-x}{b-a}$$

$$B_{1,1}(x) = \binom{1}{0} \frac{(b-x)^0(x-a)^1}{b-a} = \frac{x-a}{b-a}$$



- **n=2**

$$B_{0,2}(x) = \binom{2}{0}\frac{(b-x)^2(x-a)^0}{(b-a)^2} = \frac{(b-x)^2}{(b-a)^2}$$
$$B_{1,2}(x) = \binom{2}{1}\frac{(b-x)^1(x-a)^1}{(b-a)^2} = \frac{(b-x)(x-a)}{(b-a)^2}$$
$$B_{2,2}(x) = \binom{2}{2}\frac{(b-x)^0(x-a)^2}{(b-a)^2} = \frac{(x-a)^2}{(b-a)^2}$$



Esempio:

Riprendiamo il polinomio $p(x) = 100 - x$ e lo riscriviamo nella base di Bernstein:

$$B_{0,1}(x) = \binom{1}{0}\frac{(101-x)1}{1} = 101 - x \quad B_{1,0}(x) = \binom{1}{1}\frac{1(x-100)}{1} = x - 100$$

da cui

$$\begin{aligned} p(x) = 100 - x &= b_0 B_{0,1}(x) + b_1 B_{1,1}(x) \\ &= b_0(101 - x) + b_1(x - 100) \end{aligned}$$

Ossia $b_0 = 0$ e $b_1 = -1$.
Rieseguendo l'analisi sull'errore inerente si ha:

$$\begin{aligned} E_{in} &\leq |c_0\epsilon_0| + |c_1\epsilon_1| + |c_x\epsilon_x| \\ &= \left|\frac{b_0(101-x)}{p(x)}\epsilon_0\right| + \left|\frac{b_1(x-100)}{p(x)}\epsilon_1\right| + \left|\frac{xp'(x)}{p(x)}\epsilon_x\right| \end{aligned}$$

valutandoli in $x = 101$ si ha:

$$\begin{aligned} &= \left|\frac{0(101-101)}{-1}\epsilon_0\right| + \left|\frac{-1(1)}{-1}\epsilon_1\right| + \left|\frac{101(-1)}{-1}\epsilon_x\right| \\ &= |\epsilon_1| + |101\epsilon_x| \end{aligned}$$

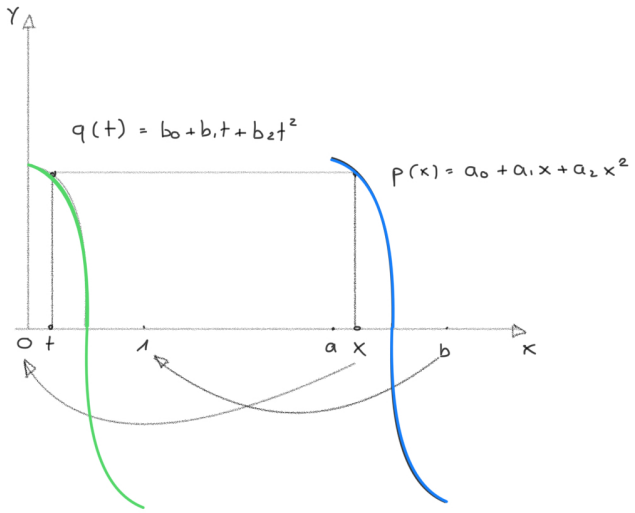
Siamo riusciti a rendere "piccoli" i numeri di condizioni che dipendono dalla base di rappresentazione. Tuttavia, il termine c_x, essendo indipendente dalla base di rappresentazione scelta, potrebbe causare un incremento significativo dell'errore inerente.

Considerando il problema evidenziato nell'esempio, si può fare qualcosa?

2.2.1 Cambio di variabile

I polinomi godono della proprietà di essere **invarianti per traslazione e scala dell'intervallo di definizione o cambio di variabile**. Questo significa che, per qualsiasi dato polinomio $p(x)$, definito in un intervallo $[a, b]$, è sempre possibile individuare un altro polinomio che assume gli stessi valori all'interno di un intervallo che è stato traslato o scalato. Questo permette di definire un'applicazione (mapping) tra i due intervalli $[a, b]$ e, per esempio, $[0, 1]$

$$\begin{aligned} &x \in [a, b] \to t \in [0, 1] \\ &t = \frac{x-a}{b-a} \text{ o viceversa } x = a + t(b-a) \end{aligned} \tag{2.5}$$



Per effettuare un cambio di variabile, possiamo sostituire, all'interno del polinomio, x con $a+t(b-a)$. Otteniamo un nuovo polinomio espresso in termini di t piuttosto che di x.
Consideriamo un polinomio $p(x)$ espresso nella base di Bernstein:

$$p(x) = \sum_{i=0}^{n} b_i B_{i,n}(x) \quad \text{con } x \in [a, b]$$

Vogliamo effettuare un cambio di variabile da x a t, con $t \in [0,1]$. Riscriviamo la 2.4 nel seguente modo:

$$= \binom{n}{i} \left(\frac{b-x}{b-a}\right)^{n-i} \left(\frac{x-a}{b-a}\right)^{i}$$

Sostituendo x con $a + t(b-a)$, otteniamo:

$$= \binom{n}{i} \left(\frac{\cancel{(b-a)}(1-t)}{\cancel{b-a}}\right)^{n-i} \left(\frac{\cancel{a} + t\cancel{(b-a)} - \cancel{a}}{\cancel{b-a}}\right)^{i}$$
$$= \binom{n}{i}(1-t)^{n-i}t^{i} = B_{i,n}(t)$$

Il cambio di variabile ha trasformato la base di Bernstein in funzione di t, mantenendo inalterati i coefficienti b_i. Pertanto, il polinomio dopo il cambio di variabile è:

$$p(t) = \sum_{i=0}^{n} b_i B_{i,n}(t) \quad \text{con } t \in [0,1]$$

Quindi un cambio di variabile, per un polinomio nella base di Bernstein, non comporta alcun errore o costo computazionale sui coefficienti in quanto restano uguali.
Ma come ci aiuta esattamente il cambio di variabile a ridurre il numero di condizione c_x?
Consideriamo la trasformazione:

$$\frac{xp'(x)}{p(x)} \to \frac{tp'(t)}{p(t)}$$

Se effettuiamo un cambio di variabile da x a t, dove t è definito nell'intervallo $[0,1]$, stiamo effettivamente ridimensionando la variabile x. Questa operazione ha un impatto anche sulla sua derivata. Infatti, il ridimensionamento di questo intervallo, per effetto della scala, potrebbe attenuare o "rilassare" la derivata del polinomio.

Esempio:

Riprendiamo il polinomio $p(x) = 100 - x$. Questo può essere espresso nella base di Bernstein come $p(x) = -1\underbrace{(x-100)}_{B_{1,1}(x)}$ con $x \in [100, 101]$. Ora, effettuiamo il cambio di variabile in $t \in [0,1]$.
I polinomi di Bernstein in termini di t sono:

$$B_{0,1}(t) = 1 - t \quad B_{1,1}(t) = t$$

In termini di t, il polinomio $p(x)$ diventa:

$$q(t) = -1 \underbrace{(t)}_{B_{1,1}(t)}$$

Se valutiamo in $x = 101$, questo corrisponde a $t = 1$. Calcoliamo ora, c_t:

$$c_t = \frac{tq'(t)}{q(t)} = \frac{1}{-1}(-1) = 1$$

Grazie al cambio di variabile, c_t è stato ridotto a un valore che non amplifica l'errore.

2.2.2 Proprietà dei polinomi di Bernstein

Da ora in poi useremo sempre i polinomi di Bernstein nell'intervallo $[0,1]$.

Definizione. *Il polinomio di Bernstein definito sull'intervallo* $[0,1]$ *è dato da:*

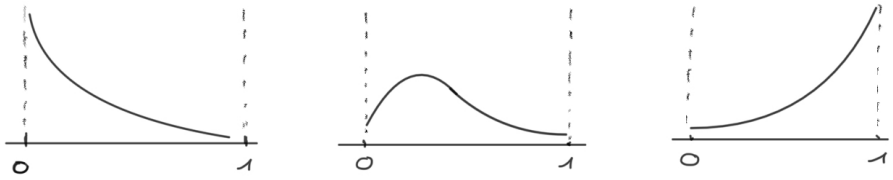
$$B_{in} = \binom{n}{i}(1-x)^{n-i} \cdot x^{i} \qquad \textit{con } x \in [0,1]$$

Inoltre, i polinomi della base di Bernstein $B_{i,n}(x)$ godono delle seguenti proprietà:

1. $B_{i,n} \geq 0 \quad i = 0, \ldots, n \quad \forall x \in [0,1]$;
   - Il valore del polinomio è sempre non negativo;

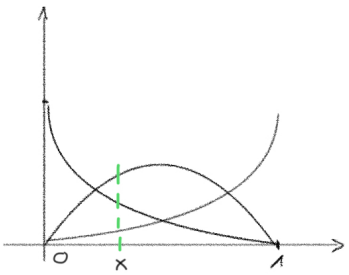
- $B_{i,n} = 0$ agli estremi dell'intervallo.

$$B_{0,n}(x) = (1-x)^n \qquad \text{tutti gli zero in 1 e nessuno zero in 0}$$
$$B_{1,n}(x) = \binom{n}{1}(1-x)^{n-1}x^1 \qquad \text{uno zero in 0 ed } n \text{ zeri in 1}$$
$$\vdots$$
$$B_{n,n}(x) = x^n \qquad \text{tutti gli zero in 0 e nessuno zero in 1}$$



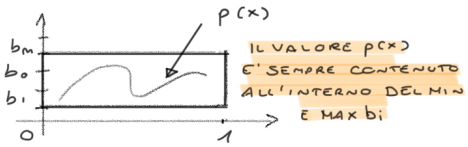
2. **Partizione dell'unità**: per ogni x nell'intervallo $[0,1]$, la somma dei valori dei polinomi di Bernstein è sempre uguale a 1.

$$\sum_{i=0}^{n} B_{in}(x) = 1 \quad \forall x \in [0,1]$$



3. Per le proprietà precedenti, $p(x)$ espresso nella base di Bernstein è una combinazione convessa dei b_i, da cui segue

$$\min_i\{b_i\} \leq p(x) \leq \max_i\{b_i\}$$



Ciò suggerisce che affinché il polinomio $p(x)$ possa avere zeri reali, i coefficienti b_i devono essere sia positivi che negativi.

4. Valutando il polinomio in $x = 0$, solo $B_{0,n}(0)$ ha valore 1, mentre tutti gli altri polinomi di Bernstein sono nulli. Di conseguenza, $p(0)$ corrisponde al coefficiente b_0. Analogamente, in $x = 1$, $p(1)$ coincide con b_n dato che solo $B_{n,n}(1)$ è non nullo e vale 1.

$$p(0) = b_0 \quad p(1) = b_n$$

Ma se volessimo valutare un polinomio nella base di Bernstein in un punto interno, anziché agli estremi? Poiché gli algoritmi di valutazione polinomiale che abbiamo visto finora sono pensati per la base canonica, è necessario introdurre due nuovi algoritmi specificamente pensati per la base di Bernstein.

2.2.3 Valutazione di un polinomio nella base di Bernstein

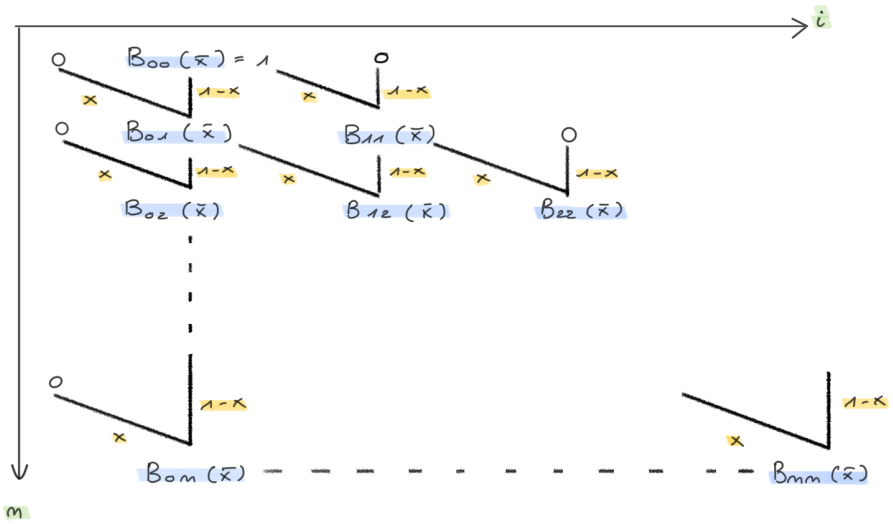
Formula ricorrente base di Bernstein. I polinomi base di Bernstein di grado n, sono determinabili dai polinomi di Bernstein di grado $n-1$, dalla seguente formula ricorrente

$$B_{i,n}(x) = xB_{i-1,n-1}(x) + (1-x)B_{i,n-1}(x)$$

con $B_{0,0}(x) = 1$ e $B_{i,n}(x) = 0$ per ogni $i \notin \{0, n\}$.

Passi:

1. **Inizializzazione**: si crea un vettore B di lunghezza $n+1$, inizializzato con tutti zeri tranne il primo elemento ($B_{0,0}$) che sarà 1. Questo vettore conterrà i valori dei polinomi base di Bernstein.
2. **Valutazione delle funzioni base di Bernstein**: si calcolano iterativamente i valori dei polinomi base di Bernstein di grado n usando la formula ricorrente.



3. **Calcolo di** $p(x) = \sum_{i=0}^{n} b_i B_{in}(\bar{x})$: si esegue il prodotto scalare del vettore riga b e il vettore colonna B^T per ottenere il valore del polinomio in $\bar{x}$.

```python
def bernstein_evaluation(b, x_bar):
    n = len(b)
    B = np.zeros((n,n))
    B[0][0] = 1.

    for i in range(1,n):
        for j in range(0,i+1):
            if j == 0:
                B[i][j] = (1-x_bar) * B[i-1][j]
            elif j == i:
                B[i][j] = x_bar * B[i-1][j-1]
            else:
                B[i][j] = x_bar * B[i-1][j-1] + (1-x_bar)*B[i-1][j]
    return sum([b[i] * B[n-1][i] for i in range(n)])
```

Quando si utilizza l'algoritmo per valutare un polinomio nella base di Bernstein:

- Il calcolo del valore dei polinomi base $B_{i,n}(\bar{x})$ per $i = 0, \ldots, n$ necessita di $\frac{n(n+1)}{2}$ addizioni e $n(n+1)$ moltiplicazioni.
- Per ottenere il valore finale del polinomio in $\bar{x}$ attraverso la combinazione lineare dei polinomi base e dei coefficienti del polinomio, sono necessarie ulteriori n addizioni e n moltiplicazioni

In totale, l'algoritmo ha un costo computazionale di $\mathcal{O}(n^2)$. Anche se questo costo potrebbe sembrare alto, va sottolineato che l'algoritmo è numericamente stabile, specialmente, nel calcolo dei polinomi base.

Algoritmo di de Casteljau. Un'alternativa all'algoritmo proposto è l'algoritmo di de Casteljau, che deriva dall'applicare ripetutamente la formula ricorrente all'espressione del polinomio. Vediamolo:

$$p(x) = \sum_{i=0}^{n} b_i B_{i,n}(x) \stackrel{\text{def}}{=} \sum_{i=0}^{n} b_i (x B_{i-1,n-1}(x) + (1-x) B_{i,n-1}(x))$$
$$= \sum_{i=0}^{n} b_i x B_{i-1,n-1}(x) + \sum_{i=0}^{n} b_i (1-x) B_{i,n-1}(x)$$

A questo punto, dobbiamo notare che il termine $B_{-1,n-1}$ non esiste. Inoltre, dato che la sommatoria effettivamente va solo fino a $n-1$, il termine $B_{n,n-1} = 0$. Dunque, regolando gli indici, abbiamo:

$$= \sum_{i=0}^{n-1} b_{i+1} x B_{i,n+1}(x) + \sum_{i=0}^{n-1} b_i (1-x) B_{i,n-1}(x)$$

Raccogliendo:

$$= \sum_{i=0}^{n-1} [b_{i+1} x + b_i (1-x)] B_{i,n-1}(x)$$
$$= \sum_{i=0}^{n-1} b_i^{[1]} B_{i,n-1}(x)$$

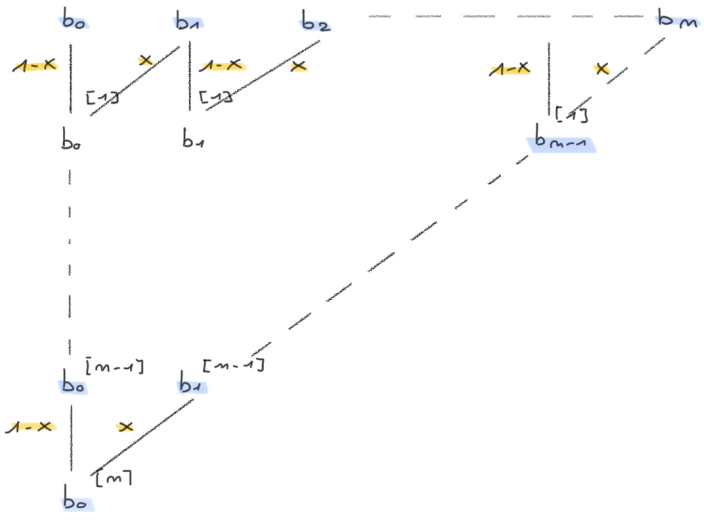
Riapplicando la formula ricorrente più volte, alla fine si ottiene:

$$= \sum_{i=0}^{0} b_0^{[n]} \overbrace{B_{0,0}(x)}^{1} = b_0^{[n]}$$

Si ha quindi il seguente algoritmo:

$$b_i^{[j]} = x b_{i+1}^{[j-1]} + (1-x) b_i^{[j-1]} \tag{2.6}$$

con $j = 0, \ldots, n$ e $i = 0, \ldots, n-j$.
Si inizia con i coefficienti iniziali del polinomio e si applica iterativamente l'equazione 2.6 fino a quando non si ottiene $b_0^{[n]}$. Quest'ultimo valore corrisponde al valore del polinomio quando viene valutato in x.




```python
def de_casteljau(b, x_bar):
    if len(b) == 1:
        return b[0]

    b_new = []
    for i in range(len(b)-1):
        b_new.append(x_bar * b[i+1] + (1-x_bar) * b[i])

    return de_casteljau(b_new, x_bar)
```

Quando si utilizza l'algoritmo di Casteljau per valutare un polinomio nella base di Bernstein:

- Il calcolo del coefficiente $b_0^{[n]}$ necessita di $\frac{n(n+1)}{2}$ addizioni e $n(n+1)$

Il costo totale dell'algoritmo è di $\mathcal{O}(n^2)$. Tuttavia, è meno costoso rispetto al precedente algoritmo. Ciò è dovuto al fatto che l'algoritmo di Casteljau elimina la necessità di calcolare esplicitamente la combinazione lineare per determinare il valore di $p(x)$.

Esempio:

Si consideri il polinomio nella base di Bernstein:

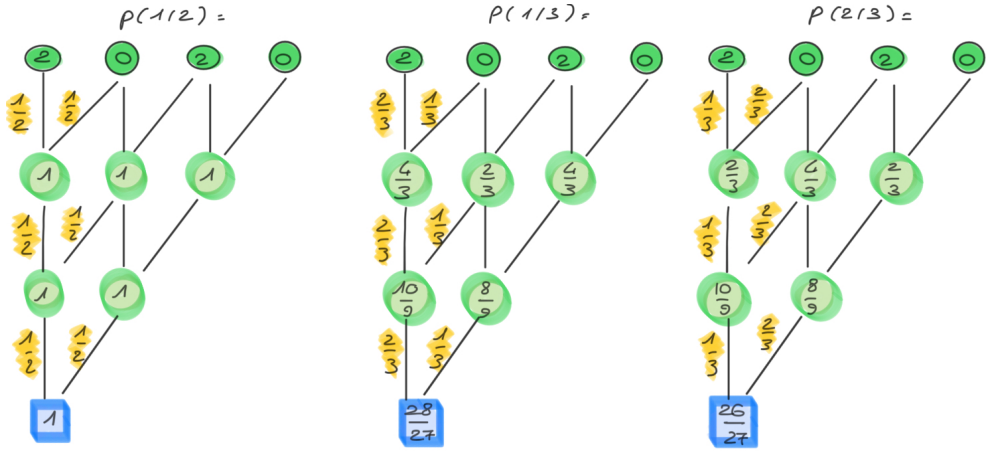
$$p(x) = 2\overbrace{B_{0,3}}^{b_0}(x) + 2\overbrace{B_{2,3}}^{b_2}(x) + 0\overbrace{B_a 1, 3(x)}^{b_1} + 0\overbrace{B_{3,3}(x)}^{b_3}$$

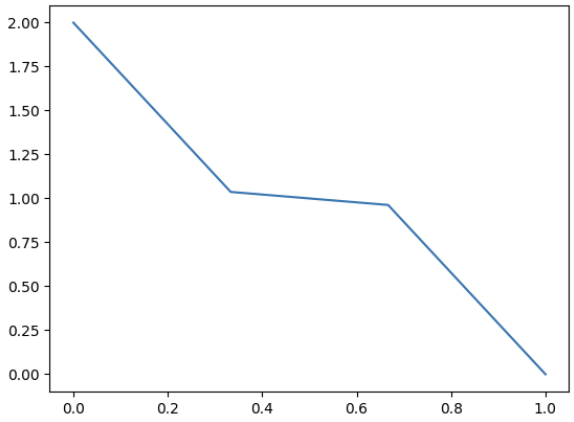
Applicare l'algoritmo di de Casteljau ai seguenti punti:

$$x = \left[0, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, 1\right]$$

Infine, disegnare un grafico per i punti $p(x)$.
Eseguendo l'algoritmo, otteniamo:

- $p(0) = 2, p(1) = 0$ dati dalle proprietà dei polinomi di Bernstein;
- $p(\frac{1}{2}) = 1, p(\frac{1}{3}) = \frac{28}{27}, p(\frac{2}{3}) = \frac{26}{27}$.





I polinomi base di Bernstein rappresentano una delle basi preferite per la rappresentazione di curve, in particolare per le curve di Bézier. Una curva 2D è definita come segue:

$$C(t) = (f(t), g(t)), \quad \text{dove} \quad t \in [0, 1].$$

dove $t \in [0, 1]$. Ogni t individua un punto sulla curva.
Utilizzando i polinomi di Bernstein come base, possiamo esprimere le funzioni $f(t)$ e $g(t)$ come segue:

$$f(t) = \sum_{i=0}^{n} x_i B_{in}(t) \quad g(t) = \sum_{i=0}^{n} y_i B_{in}(t)$$

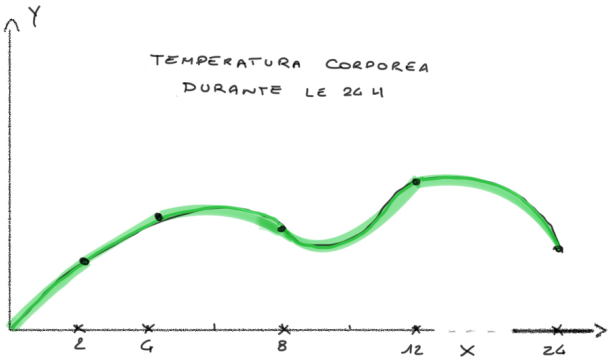
In termini vettoriali, la curva può essere descritta come una funzione vettoriale nella base di Bernstein:

$$C(t) = \sum_{i=0}^{n} P_i B_{in}(t)$$

dove $P_i = (x_i, y_i)$ rappresentano i punti di controllo.
Le curve di Bézier sono utilizzate nel disegno su calcolatore e nei font vettoriali. In questi font, un carattere è definito da curve. A differenza delle immagini raster, i disegni vettoriali mantengono nitidezza a qualsiasi livello di ingrandimento. Così, zoomando su un carattere o un disegno vettoriale, non si perde qualità.

3 Interpolazione polinomiale

Interpolazione di dati. Il problema dell'interpolazione di dati consiste nel trovare un polinomio che passi per un insieme di punti (x_i, y_i) dati. Più formalmente, si cerca un polinomio $p(x)$ appartenente all'insieme dei polinomi $\mathbb{P}_n$ tale che $p(x_i) = y_i$ per ogni $i = 0, \dots, n$.
In questo problema, abbiamo a disposizione gli y_i - i valori che desideriamo che il nostro polinomio assuma - e gli x_i - i punti corrispondenti a questi valori. Ciò che manca sono i coefficienti del polinomio, indicati con a.
Potremmo considerare l'interpolazione come il problema inverso ella valutazione di un polinomio. Nella valutazione, abbiamo i coefficienti del polinomio e gli x_i dove desideriamo valutare il polinomio. Invece, nell'interpolazione, abbiamo i valori y_i del polinomio e gli x_i, ma non conosciamo i coefficienti.
In pratica, i dati potrebbero provenire, ad esempio, da misurazioni sperimentali di un fenomeno e vogliamo rappresentarli con una funzione. Una volta identificato un polinomio interpolante adatto, questo può essere utilizzato per prevedere il comportamento della funzione all'interno dell'intervallo di interesse, incluso in punti che non sono stati originariamente misurati.



Interpolazione di funzioni. Il problema dell'interpolazione di funzioni mira a trovare un polinomio che approssima una data funzione $f(x)$ in un intervallo $x \in [a, b]$. Questo comporta:

1. **Selezione dei punti dell'interpolazione**:
   - A differenza dell'interpolazione di dati dove gli x_i sono dati, qui scegliamo autonomamente i punti x_i nell'intervallo $[a, b]$, con $i = 0, \dots, n$.
   - La scelta di questi punti è fondamentale: una selezione appropriata può portare a una rappresentazione accurata della funzione originale, mentre una scelta inadeguata può portare un'approssimazione imprecisa.
2. **Costruzione del polinomio**:
   - Si determina un polinomio $p(x) \in \mathbb{P}_n$ tale che $p(x_i)$ coincida con $f(x_i)$ per ogni $i = 0, \dots, n$.
3. **Valutazione dell'errore**:
   - Una volta costruito il polinomio, si valuta l'errore $|f(x) - p(x)|$ per ogni $x \in [a, b]$ per misurare l'accuratezza dell'approssimazione.

3.1 Esistenza e unicità dell'interpolazione polinomiale

Teorema. *Dati $n+1$ punti (x_i, y_i), $i = 0, \dots, n$ con x_i distinti, allora esiste ed è unico il polinomio $p \in \mathbb{P}_n$ che verifica le condizioni*

$$p(x_i) = y_i \quad i = 0, \dots, n$$

Dimostrazione.
Si consideri $p(x)$ in forma canonica

$$p(x) = a_0 + a_1 x + \dots + a_n x^n$$

Imponendo le $n + 1$ condizioni di interpolazione, cioè che $p(x_i) = y_i$ con $i = 0, \dots, n$, otteniamo:

$$\begin{cases} a_0 + a_1 x_0 + a_2 x_0^2 \dots + a_n x_0^n & = y_0 \\ a_0 + a_1 x_1 + a_2 x_1^2 \dots + a_n x_1^n & = y_1 \\ \dots & \vdots \\ a_0 + a_1 x_n + a_2 x_n^2 \dots + a_n x_n^n & = y_n \end{cases}$$

Cioè un sistema lineare di $n+1$ equazioni in $n+1$ incognite; se tale sistema ammette una ed una sola soluzione, allora tale sarà il polinomio. In forma matriciale il sistema si presenterà come:

$$Va = y \tag{3.1}$$

con

$$V = \begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ 1 & x_1 & x_1^2 & \dots & x_1^n \\ \dots & & & & \\ 1 & x_n & x_n^2 & \dots & x_n^n \end{bmatrix} \quad a = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} \quad y = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}$$

il sistema $Va = y$ ammette una e una sola soluzione se e solo se la matrice V, conosciuta come matrice di Vandermonde, è non singolare, il che si verifica se e solo se il $\det(V) \neq 0$. Il determinante della matrice di Vandermonde può essere espresso come:

$$\det(V) = \prod_{i,j=0,j>i}^{n} (x_j - x_i)$$

che nelle ipotesi che i punti x_i siano distinti risulta sempre non nullo. Ciò dimostra che il sistema lineare ha un'unica soluzione e quindi $p(x)$ esiste ed è unico. ■

La dimostrazione del Teorema fornisce anche un metodo per procedere alla determinazione del polinomio interpolante risolvendo il sistema lineare.

Esempio:

Siano dati il set di punti $x = [0, 1, 2], y = [0, 1, 0]$. Si determini il polinomio che interpola i dati.
In base al teorema visto, il polinomio che interpola questi dati esiste ed è unico, e può essere trovato in $\mathbb{P}_2$, ovvero $p(x) = a_0 + a_1 x + a_2 x^2$.

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$= \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 0 \end{array}\right]$$

$$= \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ \frac{3}{4} & \frac{1}{2} & 0 & 1 \\ 1 & 2 & 4 & 0 \end{array}\right]$$

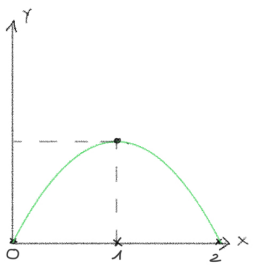
$$= \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 1 \\ 0 & 2 & 4 & 0 \end{array}\right]$$

$$= \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 2 \\ 0 & 0 & 4 & -4 \end{array}\right]$$

$$= \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & -1 \end{array}\right]$$

$$p(x) = 0 + 2x - x^2$$

Si osserva inoltre che il polimonio trovato è il polinomio di grado MINIMO che interpola i dati in $\mathbb{P}_n$.



3.2 Metodi di costruzione

Ora esamineremo vari metodi di interpolazione polinomiale, ciascuno basato su una diversa base polinomiale, allo scopo di identificare quale tra queste sia la migliore.

3.2.1 Base di Newton

Siano dati $n+1$ punti distinti (x_1, y_1) con $i = 0, \dots, n$. Il polinomio interpolante nella forma di Newton è costruito su una particolare base chiamata base di Newton, che viene costruita sui punti $x_i,\ i = 0, \dots, n$.
La base di Newton è costituita da $n+1$ funzioni base linearmente indipendenti

$$1, (x - x_0), (x - x_0)(x - x_1), \dots, (x - x_0)(x - x_1) \cdots (x - x_{n-1})$$

e forma una base polinomiale per lo spazio $\mathbb{P}_n$.
Il polinomio interpolante nella base di Newton può essere scritto come combinazione lineare di queste funzioni base:

$$p(x) = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1) + \dots + b_n(x - x_0)(x - x_1) \cdots (x - x_n)$$

Imponendo le condizioni di interpolazione a questo polinomio si ottiene un sistema di $n+1$ equazioni in $n+1$ incognite:

$$\begin{aligned}
p(x_0) = y_0 \Rightarrow \quad & b_0 + b_1 \overbrace{(x_0 - x_0)}^{0} + b_2 \overbrace{(x_0 - x_0)}^{0}(x_0 - x_1) + \dots + b_n \overbrace{(x_0 - x_0)}^{0}(x_0 - x_1) \cdots (x_0 - x_{n-1}) = y_0 \\
& b_0 = y_0 \\
p(x_1) = y_0 \Rightarrow \quad & b_0 + b_1(x_1 - x_0) + b_2(x_1 - x_0)\overbrace{(x_1 - x_1)}^{0} + \dots + b_n(x_1 - x_0)\overbrace{(x_1 - x_1)}^{0} \cdots (x_1 - x_{n-1}) = y_0 \\
& b_0 + b_1(x_1 - x_0) = y_0 \\
\vdots \quad & \vdots \\
p(x_n) = y_n \Rightarrow \quad & b_0 + b_1(x_n - x_0) + b_2(x_n - x_0)(x_n - x_1) + \dots + b_n(x_n - x_0)(x_n - x_1) \cdots (x_n - x_{n-1}) = y_n
\end{aligned}$$

Questo sistema può essere espresso in forma matriciale come:

$$Nb = y$$

con

$$N = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & (x_1 - x_0) & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & & \vdots \\ 1 & (x_n - x_0) & (x_n - x_0)(x_n - x_1) & \dots & (x_n - x_0) \cdots (x_n - x_{n-1}) \end{bmatrix} \quad c = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix} \quad y = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}$$

Poiché la matrice N è triangolare inferiore, il sistema è facilmente risolvibile per sostituzione in avanti. Ciò fornisce i coefficienti b_i, che rappresentano il polinomio che interpola i dati.

```python
def calculate_newton_base(x_bar, x):
    return np.prod([x_bar - x_i for x_i in x])

def newton_interpolation(x, y):
    n = len(x)
    L = np.zeros((n,n))

    L[:,0] = 1
    for i in range(1,n):
        for j in range(1, i+1):
            L[i][j] = calculate_newton_base(x[i], x[:j])

    return forward_substitution(L, y)
```

Esempio:

Siano dati nuovamente il set di punti $x = [0, 1, 2], y = [0, 1, 0]$. Vogliamo determinare il polinomio che interpola questi dati utilizzando la base di Newton.
Il polinomio interpolante si trova in $\mathbb{P}_2$ ed ha la seguente forma:

$$\begin{aligned} p(x) &= b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1) \\ &= b_0 + b_1(x - 0) + b_2(x - 0)(x - 1) \\ &= b_0 + b_1 x + b_2 x(x - 1) \end{aligned}$$

Possiamo calcolare i coefficienti risolvendo il seguente sistema lineare:

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 2 & 2 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 2 & 2 & 0 \end{array}\right]$$

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & -1 \end{array}\right]$$

$$p(x) = 0 + x - x(x - 1)$$

Se cambiato l'ordine dei punti in $x = [0, 2, 1], y = [0, 0, 1]$, il polinomio risultante rimane lo stesso. Il teorema, infatti, ci garantisce l'esistenza e unicità di un polinomio che passa attraverso quei punti specifici, indipendentemente dall'ordine. Quello che cambia è la base di rappresentazione utilizzata per esprimere il polinomio. Anche se il polinomio rimane invariato, i coefficienti possono cambiare a seconda dell'ordine dei punti. Quindi analiticamente, il polinomio non cambia, ma numericamente potrebbe cambiare.

$$p(x) = b_0 + b_1(x - 0) + b_2(x - 0)(x - 2)$$

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 1 & 1 & -1 & 1 \end{array}\right]$$

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 1 & -1 & 1 \end{array}\right]$$

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{array}\right]$$

$$p(x) = -x(x - 2)$$

Soluzione di un sistema lineare con matrice triangolare inferiore. Dato un sistema lineare con una matrice triangolare inferiore L, possiamo risolvere il sistema seguendo questi passaggi:

1. Dalla prima equazione, calcoliamo l'incognita $b_1 = \frac{y_1}{l_{11}}$.
2. Sostituiamo b_1 nelle equazioni successive e aggiorniamo i termini noti come segue:

   $$y_i = y_i - l_{i1} \cdot b_1, \text{ per } i = 2, \ldots, n$$

   Questo aggiornamento "sposta" il valore che abbiamo sostituito a destra nel vettore dei termini noti, riducendo la matrice del sistema e ottenendo un nuovo termine noto per tutte le equazioni successive.
3. Ripetiamo questo processo per tutte le incognite sconosciute, calcolando ogni volta l'incognita e sostituendo.

$$\begin{bmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \dots & 0 \\ \vdots & \ddots & & \vdots \\ l_{n1} & l_{n2} & \dots & l_{nm} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

$$\begin{bmatrix} l_{22} & 0 & \dots & 0 \\ l_{32} & l_{33} & \dots & 0 \\ \vdots & \ddots & & \vdots \\ l_{n2} & l_{n3} & \dots & l_{nm} \end{bmatrix} \begin{bmatrix} b_2 \\ b_3 \\ \vdots \\ b_n \end{bmatrix} = \begin{bmatrix} y_2 - l_{21} * \frac{y_1}{l_{11}} \\ y_3 - l_{31} * \frac{y_1}{l_{11}} \\ \vdots \\ y_n - l_{n1} * \frac{y_1}{l_{11}} \end{bmatrix}$$

Il costo computazionale di questo algoritmo è dell'ordine di $\mathcal{O}(m^2)$. In particolare, si effettuano $\frac{m(m+1)}{2}$ calcoli, ciascuno dei quali comprende un'addizione e una moltiplicazione, più m divisioni.

```
def forward_substitution(L, y):
    n = len(y)
    b = np.zeros(n)
    b[0] = y[0] / L[0][0]
    for k in range(0, n-1):
        for i in range(k+1, n):
            y[i] = y[i] - L[i][k] * b[k]
        b[k+1] = y[k+1] / L[k+1][k+1]
    return b.tolist()
```

Valutazione di un polinomio nella base di Newton. Il polinomio nella base di Newton è dato da:

$$p(x) = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1) + \ldots + b_n(x - x_0) \cdots (x - x_{n-1})$$

Possiamo esprimere il polinomio in una forma che facilita la sua valutazione,m similmente all'algoritmo di Horner, raccogliendo i termini simili:

$$\begin{aligned} p(x) &= b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1) + \ldots + b_n(x - x_0) \ldots (x - x_{n-1}) \\ p(x) &= b_0 + (x - x_0)(b_1 + b_2(x - x_1) + \ldots + b_n(x - x_1) \ldots (x - x_{n-1})) \\ p(x) &= b_0 + (x - x_0)(b_1 + (x - x_1)(b_2 + \ldots + b_n(x - x_2) \ldots (x - x_{n-1}))) \\ &\vdots \\ p(x) &= b_0 + (x - x_0)(b_1 + (x - x_1)(b_2 + \ldots + (x - x_{n-2})(b_{n-1} + b_n(x - x_{n-1})) \ldots) \end{aligned}$$

Questa riscrittura del polinomio rende possibile la sua valutazione seguendo una procedura analoga a quella dell'algoritmo di Horner. Tuttavia, a differenza dell'algoritmo di Horner, dove $\bar{x}$ è un valore fisso, qui esso è un vettore contenente tutte le differenze tra $\bar{x}$ e i punti della base di Newton. La valutazione viene effettuata partendo dalle parentesi interne e procedendo verso l'esterno.

```
def newton_evalutation(b, x, x_bar):
    p = b[-1]
    for k in range(len(b)-2, -1, -1):
        p = b[k] + (x_bar - x[k]) * p
    return p
```

3.2.2 Base di Bernstein

Siano dati $n+1$ punti $(x_i, y_i), i = 0, \ldots, n$ con x_i distinti e si vuole determinare $p(x) \in \mathbb{P}_n$ nella base di Bernstein:

$$p(x) = \sum_{i=0}^{n} c_i B_{i,n} \quad \text{con } x \in [a, b]$$

tale che

$$p(x_i) = y_i \quad i = 0, \ldots, n$$

La base di Bernstein è costruita su un intervallo $[a, b]$, dove $a = \min\{x_i\}$ e $b = \max\{x_i\}$, che sarà il più piccolo intervallo che contenga i punti. In questo intervallo viene definito il polinomio nella base di Bernstein per l'interpolazione.

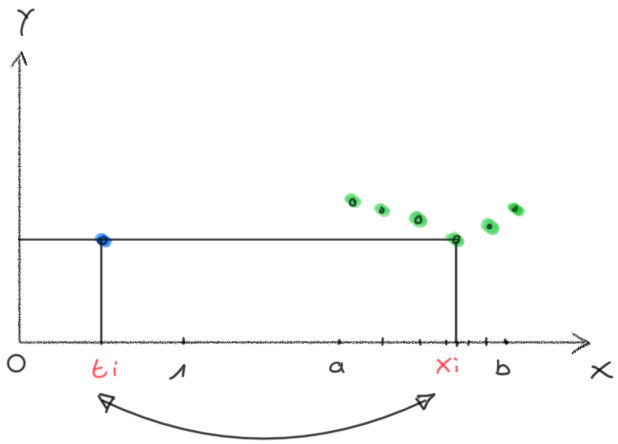
Si costruisce un sistema lineare imponendo le condizioni di esistenza:

$$\begin{aligned} p(0) = y_0 \Rightarrow &\quad \sum_{i=0}^{n} c_i B_{i,n}(x_0) = y_0 \\ p(1) = y_1 \Rightarrow &\quad \sum_{i=0}^{n} c_i B_{i,n}(x_1) = y_1 \\ \vdots &\qquad\qquad\qquad\quad \vdots \\ p(n) = y_n \Rightarrow &\quad \sum_{i=0}^{n} c_i B_{i,n}(x_n) = y_n \end{aligned}$$

Questo sistema può essere espresso in forma matriciale:

$$\begin{bmatrix} B_{0,n}(x_0) & B_{1,n}(x_0) & \dots & B_{n,n}(x_0) \\ B_{0,n}(x_1) & B_{1,n}(x_1) & \dots & B_{n,n}(x_1) \\ \vdots & \ddots & & \vdots \\ B_{0,n}(x_n) & B_{1,n}(x_n) & \dots & B_{n,n}(x_n) \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}$$

La matrice del sistema lineare è composta dai valori delle funzioni base di Bernstein valutati nel punti x_i. Tuttavia, per migliorare la precisione numerica, è conveniente traslare e scalare i punti di interpolazione nell'intervallo $[0, 1]$ mediante la 2.5.



Ora il problema consiste nell'interpolare i punti $(t_i, y_i), i = 0, \dots, n$ con un polinomio definito nella base di Bernstein:

$$p(t) = \sum_{i=0}^{n} c_i B_{in}(t) \qquad \text{con } t \in [0, 1]$$

Imponendo le condizioni di esistenza, otteniamo un sistema lineare con la stessa matrice, ma valutata nei punti $t_i = 0, \dots, n$ invece che x_i. Questa rappresentazione consente di ottenere gli stessi coefficienti c_i che si otterrebbero lavorando in $[a, b]$, ma con una maggiore precisione, in quanto meno suscettibile a problemi numerici.
Per la soluzione del sistema lineare si può applicare un algoritmo ad hoc perché la matrice è totalmente positiva, e perciò qualunque sottomatrice presa quadrata che noi ritagliamo da questa avrà il determinante > 0. Per matrici fatte così esistono algoritmi efficienti per trovare una soluzione con complessità $\mathcal{O}(n^2)$.

3.2.3 Base di Lagrange

Sia il metodo di Newton che il metodo di Bernstein, l'interpolazione richiede la soluzione di un sistema lineare. Esiste però un metodo di interpolazione che elimini completamente la necessità di risolvere un sistema lineare? Supponiamo di avere un insieme di punti (x_i, y_i) e vogliamo trovare un polinomio $p(x) \in \mathbb{P}_n$ che li interpoli. Anziché risolvere il problema di interpolazione su tutti i dati, possiamo semplificarlo costruendo un polinomio che ha zeri in tutti i punti x_i, tranne in uno specifico punto x_j, dove vogliamo che il polinomio valga 1. Per il Teorema 2.1, i polinomi $L_{i,n}(x)$ avendo come radici i punti x_j con $j = 0, \dots, n$ e $j \neq i$ saranno nella forma

$$L_{i,n}(x) = d_i(x - x_0)(x - x_1) \cdots (x - x_{i-1})(x - x_{i+1}) \cdots (x - x_n)$$

con d_i una costante da determinare in modo che $L_{i,n}(x_i) = 1$. Troviamo d_i come:

$$d_i(x_i - x_0)(x_i - x_1)\cdots(x_i - x_{i-1})(x_i - x_{i+1})\cdots(x_i - x_n) = 1$$
$$d_i = \frac{1}{(x_i - x_0)(x_i - x_1)\cdots(x_i - x_{i-1})(x_i - x_{i+1})\cdots(x_i - x_n)}$$

da cui in definitiva

$$\begin{aligned} L_{i,n}(x) =& \frac{(x - x_0)(x - x_1)\cdots(x - x_{i-1})(x - x_{i+1})\cdots(x - x_n)}{(x_i - x_0)(x_i - x_1)\cdots(x_i - x_{i-1})(x_i - x_{i+1})\cdots(x_i - x_n)} \\ =& \frac{\prod_{j=0, j\neq i}^{n}(x - x_j)}{\prod_{j=0, j\neq i}(x_i - x_j)} \quad i = 0, \ldots, n \end{aligned}$$

Sono $n + 1$ funzioni che formano una base polinomiale per lo spazio $\mathbb{P}_n$ detta base di Lagrange.

$$\{L_{0,n}(x), L_{1,n}(x), \ldots, L_{n,n}(x)\} \in \mathbb{P}_n$$

Ora che abbiamo questi polinomi base di Lagrange, il problema è banalmente risolto da

$$p(x) = \sum_{i=0}^{n} y_i L_{i,n}(x) \tag{3.2}$$

infatti per le condizioni $L_{i,j}(x_j) = \begin{cases} 1 & \text{se } i = j \\ 0 & \text{altrimenti} \end{cases}$ che definiscono i polinomi $L_{i,n}(x)$, banalmente in ogni punto x_j l'espressione polinomiale 3.2 varrà y_j per ogni $j = 0, \ldots, n$.
Dal punto di vista computazionale determinare tale polinomio non costa nulla, essendo i coefficienti del polinomio in tale base proprio i valori y_i assegnati. Al contrario la valutazione del polinomio interpolante comporta la determinazione e la valutazione dei polinomi $L_{i,n}(x)$.

Valutazione di un polinomio nella base di Lagrange. Calcolare i polinomi base di Lagrange può essere costoso, ma possiamo semplificarlo utilizzando una formula alternativa.

I forma baricentrica:
Iniziamo con il polinomio di Lagrange:

$$L_{i,n}(x) = d_i \prod_{j=0, j\neq i}^{n} (x - x_j)$$

dove $d_i = \frac{1}{\prod_{j=0, j\neq i}(x_i - x_j)}$
Possiamo calcolare una volta sola prima di valutare il polinomio:

$$p(\bar{x}) = \sum_{i=0}^{n} y_i L_{i,n}(\bar{x})$$

Successivamente, determiniamo:

$$l(x) = (x - x_0)(x - x_1)\cdots(x - x_n)$$

cioè, il prodotto di tutti i termini senza saltarne nessuno e riscriviamo il polinomio di Lagrange come:

$$L_{i,n}(x) = d_i \frac{l(x)}{x - x_i}$$

Da cui segue:

$$p(x) = \sum_{i=0}^{n} y_i \frac{w_i l(x)}{x - x_i}$$

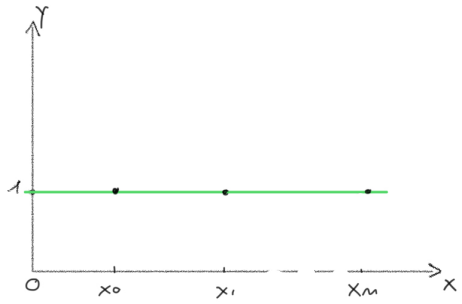
Poiché $l(x)$ non dipende più dall'indice di sommatoria, possiamo portarlo fuori:

$$l(x) \sum_{i=0}^{n} y_i \frac{w_i}{x - x_i}$$

dove w_i e $l(x)$ vengono calcolati una sola volta con n moltiplicazioni, e all'interno della sommatoria vengono effettuate 2 moltiplicazioni per n volte. Il costo computazionale totale per la valutazione è quindi $\mathcal{O}(n)$.

II forma baricentrica:
Consideriamo il polinomio nella forma di Lagrange che interpola $(x_i, y_i = 1)$ per $i = 0, \dots, n$. Essendo la funzione costante 1 l'unica che interpola i nostri dati in $\mathbb{P}_n$, allora il polinomio nella I forma di Lagrange corrispondente a 1, è dato da:

$$p(x) = l(x) \sum_{i=0}^{n} \frac{d_i}{x - x_i} 1 = 1 \tag{3.3}$$



Ora, dividendo la I forma di Lagrange per 1, ma scrivendo 1 come in 3.3, otteniamo:

$$p(x) = \frac{\cancel{l(x)} \sum_{i=0}^{n} y_i \frac{d_i}{x - x_i}}{\cancel{l(x)} \sum_{i=0}^{n} \frac{d_i}{x - x_i}}$$

Il vantaggio di questa rappresentazione deriva dal fatto che, avendo semplificato analiticamente $l(x)$, non è più necessario calcolarlo. Il costo computazionale rimane $\mathcal{O}(n)$, ma con meno operazioni da fare, e si svolge come segue:

- Nel ciclo, si eseguono una divisione e una moltiplicazione al numeratore.
- Dopo il ciclo, si effettua un'ultima divisione.

Questo rende l'algoritmo più efficiente e altamente stabile numericamente.

```python
def calculate_di(x, i):
    return 1/np.prod([x[i] - x[j] for j in range(len(x)) if j != i])

def lagrange_evaluation(x, y, x_bar):
    n = len(x)
    d = [calculate_di(x, i) for i in range(n)]

    num = 0
    den = 0
    for i in range(n):
        div = d[i]/(x_bar-x[i])
        num += y[i] * div
        den += div
    return num/den
```

3.3 Errore di interpolazione (interpolazione di funzioni)

Recap:

I termini C^k dove k è un intero non negativo, sono utilizzati per classificare le funzioni in base alla loro regolarità. Nello specifico:

- C^0: La funzione è continua.
- C^1: La funzione ha una derivata prima continua.
- C^2: La funzione ha derivate prima e seconda continue.
- $\vdots$
- C^∞: La funzione è infinitamente differenziabile e tutte le sue derivate sono continue.

Queste notazioni sono utilizzate per indicare quanto sia "liscia" una funzione, cioè quante derivate continue possiede.

Sia $f(x)$ è la funzione assegnata nell'intervallo $[a, b]$ e sia $p(x)$ il polinomio interpolante nei punti $(x_i, f(x_i)), i = 0, \dots, n$; ha senso chiedersi quanto sia grande l'errore di interpolazione

$$R(x) = |f(x) - p(x)|$$

che si commette in un punto $\bar{x} \in [a, b]$ diverso dai punti di interpolazione x_i.

Teorema. *Sia $f(x) \in C^{(n+1)}_{[a,b]}$; siano $x_0, x_1, \dots, x_n \in [a, b]$ punti di interpolazione distinti allora esiste un punto ξ che dipende da $\bar{x}$ (punto di valutazione) per cui*

$$f(\bar{x}) = p(\bar{x}) + \frac{w(\bar{x})}{(n+1)!} f^{(n+1)}(\xi)$$

con $w(\bar{x}) = (\bar{x} - x_0)(\bar{x} - x_1) \cdots (\bar{x} - x_n)$.

La parte evidenziata in rosso rappresenta l'errore di interpolazione. Questa quantità ci mostra come l'errore dipende dalla regolarità della funzione ($(n+1)$-esima derivata), dai punti interpolanti scelti e dal punto di valutazione $\bar{x}$ specifico.

Esempio:

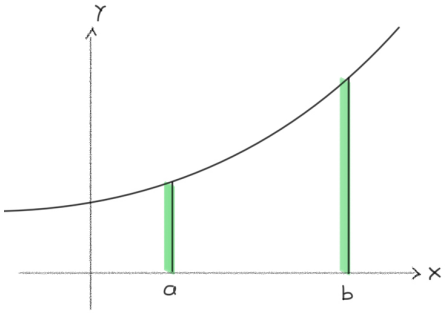
Sia $f(x) = e^x$ con $x \in [a, b]$. Si osserva che:

- $\left|f^{(n+1)}(\xi)\right| \leq e^b$
- $w(\bar{x}) = (\bar{x} - x_0)(\bar{x} - x_1) \cdots (\bar{x} - x_n) \leq (b-a)(b-a) \dots (b-a) = (b-a)^{n+1}$

da cui

$$R(x) \leq \frac{(b-a)^{n+1}}{(n+1)!} e^b \underset{n \to \infty}{\to} 0$$

Notiamo che il fattore $(n+1)!$ cresce più velocemente di $(b-a)^{n+1}$, e quindi l'intero fattore tende a zero quando n tende all'infinito.
Questo dimostra che il polinomio interpolante converge alla funzione all'aumentare del numero dei punti di interpolazione.



3.3.1 Punti equispaziati e di Chebyshev

Consideriamo la funzione $f(x) = \frac{1}{1+x^2}$, definita sull'intervallo $x \in [-5, 5]$, e il polinomio di interpolazione costruito utilizzando $n+1$ punti equispaziati $x_i = \frac{10}{n} i - 5 \quad i = 0, \dots, n$.
Il matematico Runge dimostrò un caso in cui l'errore di interpolazione non migliora, ma peggiora all'aumentare dei punti. Invece di convergere alla funzione, il polinomio inizia ad oscillare vicino agli estremi dell'intervallo, e queste oscillazioni si intensificano all'aumentare del numero di punti in cui vado ad interpolare e del grado del polinomio che vado ad utilizzare.
La causa di questo comportamento risiede proprio nella scelta dei punti equispaziati.

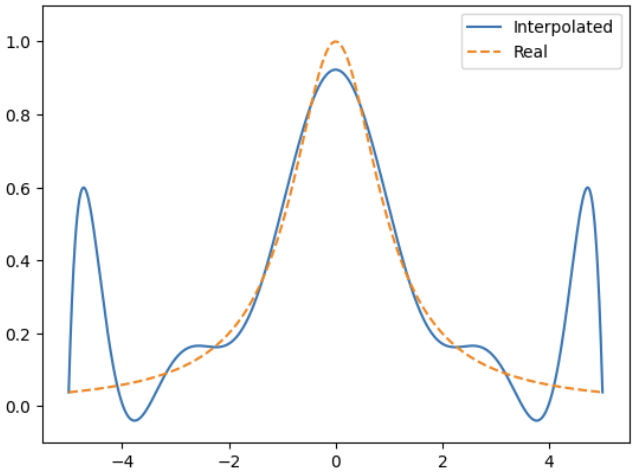


Figure 2: Interpolazione polinomiale della funzione di Runge su 11 punti equispaziati

Teorema. *Sia $I \in [-1, 1]$ e i punti di interpolazione siano gli zeri del polinomio di Chebyshev di grado $n+1$. Se $f(x) \in C_I^0$ e soddisfa la condizione di Lipschitz*[2]

$$|f(\bar{x} - f(\tilde{x}))| < K\,|\bar{x} - \tilde{x}|$$

allora il polinomio interpolante converge uniformemente a $f(x)$ su I per $n \to \infty$ e $|f(x) - p_n(x)| \underset{n\to\infty}{\to} 0$

Quindi se i punti di interpolazione vengono sono scelti opportunamente (come per esempio gli zeri del polinomio di Chebyshev di grado $n+1$[3]) e la funzione $f(x)$ è Lipschitziana si ha la convergenza dell'interpolante polinomiale. Più la f è regolare e più veloce la convergenza.

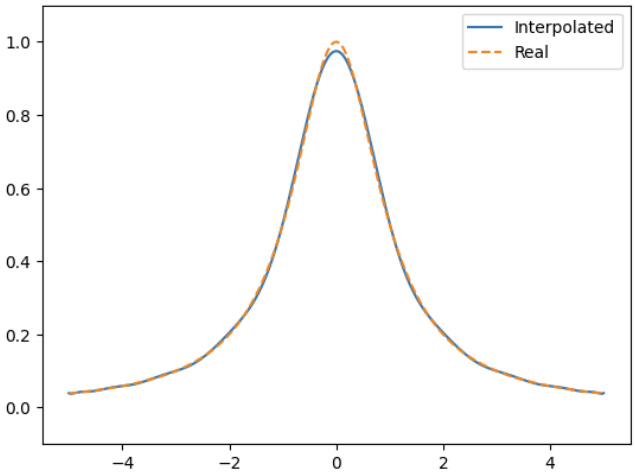


Figure 3: Interpolazione polinomiale della funzione di Runge su 21 punti zeri di Chebyshev

[2]La condizione di Lipschitz richiede che la funzione non abbia variazioni troppo brusche tra due punti vicini, aiutando a garantire la regolarità della funzione.

[3]$x_i = \cos(\frac{(2i+1)\pi}{2(n+1)}), \quad i = 0, \dots, n$ sono gli $n+1$ zeri reali del polinomio di Chebyshev di prima specie di grado $n+1$ definito in $[-1, 1]$; se i punti di interpolazione sono in $[a, b]$ si applichi un mapping degli zeri da $[-1, 1]$ in $[a, b]$

4 Integrazione numerica