Calcolo Numerico - Appunti - v1.2.0

260236

gennaio 2026

Prefazione

Ogni sezione di teoria presente in questi appunti, è stata ricavata dalle seguenti risorse:

- Calcolo Scientifico: Esercizi e problemi risolti con MATLAB e Octave. [2]
- Slide del corso. [1]

Altro:

GitHub repository

Questi appunti sono una risorsa non ufficiale e non dovrebbero rimpiazzare il materiale del corso o di qualsiasi altro libro sul "calcolo numerico". Non sono stati creati per scopi commerciali. Ho creato questi appunti per aumentare la mia cultura e, forse, potrebbero essere utili a qualcuno.

Infine, uno studente dovrebbe scegliere il materiale fornito dal professore o dalla professoressa o dal libro del corso. Questi appunti possono in ogni caso essere un materiale utile.

Indice

1 Equazioni non lineari

1.1 Introduzione

Il **calcolo degli zeri di una funzione** f reale di variabile reale o delle **radici dell'equazione** $f(x) = 0$, è un problema assai ricorrente nel Calcolo Scientifico.

In generale, *non è possibile* approntare metodi numerici che calcolino gli zeri di una generica funzione in un numero finito di passi. I metodi numerici per la risoluzione di questo problema sono pertanto necessariamente *iterativi*. A partire da uno o più dati iniziali, scelti convenientemente, essi generano una successione di valori $x^{(k)}$ che, sotto opportune ipotesi, convergerà ad uno zero α della funzione f studiata.

1.2 Il metodo di bisezione (o iterativo)

Sia f una funzione continua in $[a, b]$ tale che $f(a) f(b) < 0$. Per cui, vale il teorema degli zeri di una funzione continua, ossia f ammette almeno uno zero in (a, b).

Si supponga che ci sia un solo zero, indicato con α e nel caso in cui ce ne sia più di uno, individuare un intervallo tale che ne contenga solo uno.

Il **metodo di bisezione** (o **iterativo**) è una strategia che si suddivide nei seguenti passaggi:

1. **Dimezzare l'intervallo di partenza**;
2. **Selezionare tra i due sotto-intervalli ottenuti quello nel quale f cambia di segno agli estremi**;
3. **Applicare ricorsivamente questa procedura all'ultimo intervallo selezionato**.

Matematicamente parlando, dato $I^{(0)} = (a, b)$, e più in generale, $I^{(k)}$ il sotto-intervallo selezionato al passo k-esimo, si sceglie come $I^{(k+1)}$ il semi-intervallo di $I^{(k)}$ ai cui estremi f cambia di segno.

Questa procedura garantisce che ogni sotto-intervallo selezionato $I^{(k)}$ conterrà α. Questo poiché la successione $\left\{x^{(k)}\right\}$ dei punti medi dei sotto-intervalli $I^{(k)}$ dovrà ineluttabilmente convergere a α, in quanto la **lunghezza dei sotto-intervalli tende a** 0 per k che **tende all'infinito**.

Formalizziamo questa idea con un piccolo algoritmo. Ponendo:

$$a^{(0)} = a, \quad b^{(0)} = b, \quad I^{(0)} = \left(a^{(0)}, b^{(0)}\right), \quad x^{(0)} = \frac{a^{(0)} + b^{(0)}}{2}$$

Al passo $k \geq 1$ il metodo di bisezione calcolerà il semi-intervallo $I^{(k)} = \left(a^{(k)}, b^{(k)}\right)$ dell'intervallo $I^{(k-1)} = \left(a^{(k-1)}, b^{(k-1)}\right)$, nel seguente modo (si ricorda che α è lo zero che si sta cercando):

1. Calcolo $x^{(k-1)} = \dfrac{a^{(k-1)} + b^{(k-1)}}{2}$
2. Se $f\left(x^{(k-1)}\right) = 0$:
   (a) Allora $\alpha = x^{(k-1)}$ e l'algoritmo termina.
3. Altrimenti, se $f\left(a^{(k-1)}\right) \cdot f\left(x^{(k-1)}\right) < 0$:
   (a) Si pone $a^{(k)} = a^{(k-1)}$
   (b) Si pone $b^{(k)} = x^{(k-1)}$
   (c) Si incrementa $k + 1$ e si ripete ricorsivamente.
4. Altrimenti, se $f\left(x^{(k-1)}\right) \cdot f\left(b^{(k-1)}\right) < 0$:
   (a) Si pone $a^{(k)} = x^{(k-1)}$
   (b) Si pone $b^{(k)} = b^{(k-1)}$
   (c) Si incrementa $k + 1$ e si ripete ricorsivamente.

Esempio 1

Data la funzione $f(x) = x^2 - 1$, si parta da $a^{(0)} = -0.25$ e $b^{(0)} = 1.25$, e si applichi il metodo di bisezione:

1. Con $a^{(0)} = -0.25$ e $b^{(0)} = 1.25$:
   (a) Si calcola il punto medio:
   $$x^{(0)} = \frac{a^{(0)} + b^{(0)}}{2} = \frac{-0.25 + 1.25}{2} = 0.5$$
   (b) Si calcola la funzione con il punto medio come parametro:
   $$f(0.5) = 0.5^2 - 1 = -0.75$$
   (c) Dato che la funzione nel punto medio non è uguale a zero, l'algoritmo deve continuare. Per farlo, bisogna sostituire il punto medio con uno dei due estremi. Per decidere quale dei due sostituire, è necessario capire in quale cambia valore la funzione. Si verifica inizialmente con $a^{(0)}$:
   $$\begin{aligned} f\left(a^{(0)}\right) f\left(x^{(0)}\right) < 0 &= f(-0.25)\, f(0.5) < 0 \\ &= (-0.9375 \cdot -0.75) < 0 \\ &= 0.703125 \text{ ✗} \end{aligned}$$

(d) Si procede con l'algoritmo, provando adesso la $b^{(0)}$:

$$\begin{aligned} f\left(x^{(0)}\right) f\left(b^{(0)}\right) < 0 &= f(0.5)\, f(1.25) < 0 \\ &= (-0.75 \cdot 0.5625) < 0 \\ &= -0.421875 \text{ ✓} \end{aligned}$$

(e) Si pone $a^{(1)} = x^{(0)} = 0.5$

(f) Si pone $b^{(1)} = b^{(0)} = 1.25$

(g) Si incrementa k, $k = k + 1 = 0 + 1 = 1$

2. Con $a^{(1)} = 0.5$ e $b^{(1)} = 1.25$:

(a) Si calcola il punto medio:

$$x^{(1)} = \frac{a^{(1)} + b^{(1)}}{2} = \frac{0.5 + 1.25}{2} = 0.875$$

(b) Si calcola la funzione con il punto medio come parametro:

$$f(0.875) = 0.875^2 - 1 = -0.234375$$

(c) Dato che la funzione nel punto medio non è uguale a zero, l'algoritmo deve continuare:

$$\begin{aligned} f\left(a^{(1)}\right) f\left(x^{(1)}\right) < 0 &= f(0.5)\, f(-0.234375) < 0 \\ &= (-0.75 \cdot -0.945068359375) < 0 \\ &= 0.70880126953125 \text{ ✗} \end{aligned}$$

(d) Si procede con l'algoritmo:

$$\begin{aligned} f\left(x^{(1)}\right) f\left(b^{(1)}\right) < 0 &= f(-0.234375)\, f(1.25) < 0 \\ &= (-0.945068359375 \cdot 0.5625) < 0 \\ &= -0.5316009521484375 \text{ ✓} \end{aligned}$$

(e) Si pone $a^{(2)} = x^{(1)} = 0.875$

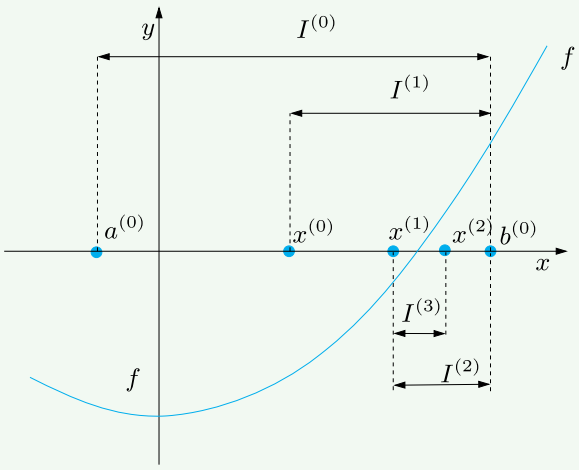
(f) Si pone $b^{(2)} = b^{(1)} = 1.25$

(g) Si incrementa k, $k = k + 1 = 1 + 1 = 2$

Si omettono i restanti calcoli per $k = 2, k = 3$, ma si lasciano qua di seguito i risultati:

- $I^{(2)} = (0.875, 1.25)$ e $x^{(2)} = 1.0625$
- $I^{(3)} = (0.875, 1.0625)$ e $x^{(2)} = 0.96875$

Nella seguente figura si possono vedere le iterazioni effettuate:



Iterazioni effettuate. [2]

Si noti che ogni intervallo $I^{(k)}$ contiene lo zero α. Inoltre, la successione $\left\{x^{(k)}\right\}$ converge necessariamente allo zero α in quanto ad ogni passo l'ampiezza $\left|I^{(k)}\right| = b^{(k)} - a^{(k)}$ dell'intervallo $I^{(k)}$ si dimezza.

Il valore $I^{(k)}$ può essere riassunto come:

$$\left|I^{(k)}\right| = \left(\frac{1}{2}\right)^{k} \cdot \left|I^{(0)}\right|$$

E di conseguenza l'**errore al passo *k*** può essere calcolato come:

$$\left|e^{(k)}\right| = \left|x^{(k)} - \alpha\right| < \frac{1}{2} \cdot \left|I^{(k)}\right| = \left(\frac{1}{2}\right)^{k+1} \cdot (b-a)$$

Inoltre, data una certa **tolleranza** ε, per **garantire che l'errore al passo** k **sia minore della tolleranza data** (ovvero, $\left|e^{(k)}\right| < \varepsilon$), basta applicare la seguente formula:

$$k_{\min} > \log_2\left(\frac{b-a}{\varepsilon}\right) - 1 \tag{1}$$

Dove $k_{\min}$ rappresenta il **numero <u>minimo</u> di iterazioni prima di trovare un intero che soddisfi la disuguaglianza**.

Possibile svantaggio

Il metodo di bisezione **non garantisce una riduzione monotona dell'errore**, ma solo il dimezzamento dell'ampiezza dell'intervallo all'interno del quale si cerca lo zero. Infatti, **non viene tenuto conto del reale andamento di** f e questo può provocare il mancato coinvolgimento di approssimazioni di α accurate.

1.3 Il metodo di Newton

Il **metodo di Newton** sfrutta la funzione f maggiormente rispetto al metodo di bisezione, usando i suoi valori e la sua derivata.

Si ricorda che la retta tangente alla curva $(x, f(x))$ nel punto $x^{(k)}$ è:

$$y(x) = f\left(x^{(k)}\right) + f'\left(x^{(k)}\right)\left(x - x^{(k)}\right)$$

Cercando un $x^{(k+1)}$ tale che la **retta tangente in quel punto sia uguale a zero** $y\left(x^{(k+1)}\right) = 0$, allora si trova:

$$x^{(k+1)} = x^{(k)} - \frac{f\left(x^{(k)}\right)}{f'\left(x^{(k)}\right)} \qquad k \geq 0 \tag{2}$$

Purché la derivata prima nel punto $x^{(k)}$ sia diversa da zero, cioè $f'\left(x^{k}\right) \neq 0$.

Questa equazione consente di calcolare una successione di valori $x^{(k)}$ a partire da un dato iniziale $x^{(0)}$. In altre parole, il **metodo di Newton calcola lo zero di f sostituendo localmente a f la sua retta tangente**.

A differenza del metodo di bisezione, tale **metodo converge allo zero in un solo passo quando la funzione f è lineare**, ovvero nella forma $f(x) = a_1 x + a_0$.

Limitazione

La **convergenza** del metodo di Newton non è garantita **per ogni scelta** di $x^{(0)}$, ma **soltanto** per valori di $x^{(0)}$ **sufficientemente vicini** ad α, ovvero **appartenenti ad un intorno** $I(\alpha)$ sufficientemente piccolo di α.

Alcune osservazioni a seguito anche di questa limitazione:

- A seguito di questa limitazione, risulta evidente che se $x^{(0)}$ è stato scelto opportunamente e se lo zero α è semplice ($f'(\alpha) \neq 0$), allora il metodo converge.
- Nel caso in cui f è derivabile con continuità pari a due, allora si ottiene la seguente convergenza:

  $$\lim_{k\to\infty} \frac{x^{(k+1)-\alpha}}{\left(x^{(k)} - \alpha\right)^2} = \frac{f''(\alpha)}{2f'(\alpha)} \tag{3}$$

  Il significato è: se $f'(\alpha) \neq 0$ il metodo di Newton converge almeno quadraticamente o con **ordine 2**.

  In parole povere, **per k sufficientemente grande, l'errore al passo $(k+1)$-esimo si comporta come il quadrato dell'errore al passo k-esimo, moltiplicato per una costante indipendente da k.**
- Se lo zero α ha molteplicità m maggiore di 1, ovverosia:

  $$f'(\alpha) = 0, \ldots, f^{(m-1)}(\alpha) = 0$$

Allora il metodo di Newton è ancora convergente, purché $x^{(0)}$ sia scelto opportunamente e $f'(x) \neq 0 \ \forall x \in I(\alpha) \setminus \{\alpha\}$. Tuttavia in questo caso l'ordine di convergenza è pari a 1. In tal caso, l'ordine 2 può essere ancora recuperato usando la seguente relazione al posto dell'equazione 2 ufficiale:

$$x^{(k+1)} = x^{(k)} - m \cdot \frac{f\left(x^{(k)}\right)}{f'\left(x^{(k)}\right)} \qquad k \geq 0 \tag{4}$$

Purché $f'\left(x^{(k)}\right) \neq 0$. Naturalmente, questo **metodo di Newton modificato** richiede una conoscenza a priori di m.

1.3.1 Come arrestare il metodo di Newton

Data una tolleranza fissa ε, esistono due tecniche applicabili per capire quando è necessario fermarsi ed evitare di continuare ad iterare:

- La **differenza fra due iterate consecutive**, il quale si arresta in corrispondenza del più piccolo intero $k_{\min}$ per il quale:

$$\left|x^{(k_{\min})} - x^{(k_{\min}-1)}\right| < \varepsilon \tag{5}$$

  (test sull'incremento).

- Un'altra tecnica applicata anche per altri metodi iterativi è il **residuo** al passo k, il quale è definito come:

$$r^{(k)} = f\left(x^{(k)}\right)$$

  Che è nullo quando $x^{(k)}$ è uno zero di f. In questo modo, il metodo viene arrestato alla prima iterata $k_{\min}$:

$$\left|r^{(k_{\min})}\right| = \left|f\left(x^{(k_{\min})}\right)\right| < \varepsilon \tag{6}$$

  Da notare che tale tecnica fornisce una **stima accurata dell'errore** solo quando $|f'(x)|$ è circa pari a 1 in un intorno di I_α dello zero α cercato.

  Attenzione! Se la derivata non è circa pari a 1 in un intorno dello zero cercato, la tecnica porterà:

  - Ad una **sovrastima** dell'errore se $|f'(x)| \gg 1$ per $x \in I_\alpha$
  - Ad una **sottostima** dell'errore se $|f'(x)| \ll 1$ per $x \in I_\alpha$

1.4 Il metodo delle secanti

Nel caso in cui la funzione f non sia nota, il metodo di Newton non può essere applicato. Per fortuna, arriva in soccorso il **metodo delle secanti**, il quale esegue una valutazione di $f'\left(x^{(k)}\right)$ andando a sostituire quest'ultima con un **rapporto incrementale calcolato su valori di f già noti**.

Più formalmente, assegnati due punti $x^{(0)}$ e $x^{(1)}$, per $k \geq 1$ si calcola:

$$x^{(k+1)} = x^{(k)} - \left(\frac{f\left(x^{(k)}\right) - f\left(x^{(k-1)}\right)}{x^{(k)} - x^{(k-1)}} \right)^{-1} \cdot f\left(x^{(k)}\right) \tag{7}$$

Quando converge?

Il metodo delle secanti converge a seguito di certe condizioni:

- **Converge ad α**, se:
  - α radice semplice[1];
  - $I(\alpha)$ è un opportuno intorno di α;
  - $x^{(0)}$ e $x^{(1)}$ sono sufficientemente vicini ad α
  - $f'(x) \neq 0 \ \forall x \in I(\alpha) \setminus \{\alpha\}$
- **Converge con ordine p super-lineare**, se:
  - $f \in \mathcal{C}^2(I(\alpha))$
  - $f'(\alpha) \neq 0$

  Ovvero, esiste una costante $c > 0$ tale che:

$$\left| x^{(k+1)} - \alpha \right| \leq c \left| x^{(k)} - \alpha \right|^p \qquad p = \frac{1+\sqrt{5}}{2} \approx 1.618\ldots \tag{8}$$

- **Convergenza lineare**, se:
  - Radice α è multipla.

  Come succederebbe usando il metodo di Newton.

[1] $f'(\alpha) \neq 0$

1.5 I sistemi di equazioni non lineari

Di solito i metodi presentati nelle pagine precedenti vengono inseriti in dei sistemi. Nella realtà ci sono varie condizioni che influiscono sul sistema in analisi. È per questo motivo che si introducono i sistemi.

Si consideri un generale sistema di equazioni non lineari:

$$\begin{cases} f_1(x_1, x_2, \dots, x_n) = 0 \\ f_2(x_1, x_2, \dots, x_n) = 0 \\ \vdots \\ f_n(x_1, x_2, \dots, x_n) = 0 \end{cases}$$

Dove $f_1, \dots, f_n$ sono funzioni non lineari. Si pongono i seguenti vettori:

- $\mathbf{f} \equiv (f_1, \dots, f_n)^T$
- $\mathbf{x} \equiv (x_1, \dots, x_n)^T$

Con l'obbiettivo di riscrivere il sistema in maniera più agevole:

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}$$

Esempio 2: esempio di sistema non lineare

Un esempio banale di sistema non lineare:

$$\begin{cases} f_1(x_1, x_2) = x_1^2 + x_2^2 - 1 = 0 \\ f_2(x_1, x_2) = \sin\left(\pi \dfrac{x_1}{2}\right) + x_2^3 = 0 \end{cases}$$

Prima di estendere i metodi di Newton e delle secanti si introduce la matrice Jacobiana.

Definizione 1: matrice Jacobiana

Senza entrare troppo nel gergo matematico (non è l'obbiettivo del corso), la **matrice Jacobiana** di una funzione è quella **matrice i cui elementi sono le derivate parziali prime della funzione**.

$$\mathbf{J_f} = \begin{pmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{pmatrix} \tag{9}$$

Che può essere riscritto in modo più leggibile come:

$$(\mathbf{J_f})_{ij} \equiv \frac{\partial f_i}{\partial x_j} \qquad i, j = 1, \dots, n \tag{10}$$

Dove rappresenta la derivata parziale della funzione f_i rispetto a x_j.

Il metodo di Newton e delle secanti può essere esteso sfruttando la matrice Jacobiana:

- Il **metodo di Newton** usando un sistema di equazioni non lineari diventa: dato $\mathbf{x}^{(0)} \in \mathbb{R}^n$, per $k = 0, 1, \ldots$, fino a convergenza

$$\begin{aligned} &\text{risolvere} && \mathbf{J_f}\left(\mathbf{x}^{(k)}\right) \boldsymbol{\delta}\mathbf{x}^{(k)} = -\mathbf{f}\left(\mathbf{x}^{(k)}\right) \\ &\text{porre} && \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \boldsymbol{\delta}\mathbf{x}^{(k)} \end{aligned} \tag{11}$$

  Se ne deduce che venga richiesto ad ogni passo la soluzione di un sistema lineare di matrice $\mathbf{J_f}\left(\mathbf{x}^{(k)}\right)$.

- Il **metodo delle secanti** usando un sistema di equazioni non lineari si basa sulla matrice Jacobiana e sul metodo di Broyden.

  L'**idea di base** è sostituire le matrici Jacobiane $\mathbf{J_f}\left(\mathbf{x}^{(k)}\right)$ (per $k \geq 0$) con delle matrici chiamate B_k, definite ricorsivamente a partire da una matrice B_0 che sia una approssimazione di $\mathbf{J_f}\left(\mathbf{x}^{(0)}\right)$.

  Dato $\mathbf{x}^{(0)} \in \mathbb{R}^n$, data $\mathrm{B}_0 \in \mathbb{R}^{n \times n}$ per $k = 0, 1, \ldots$, fino a convergenza

$$\begin{aligned} &\text{risolvere} && \mathrm{B}_k \boldsymbol{\delta}\mathbf{x}^{(k)} = -\mathbf{f}\left(\mathbf{x}^{(k)}\right) \\ &\text{porre} && \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \boldsymbol{\delta}\mathbf{x}^{(k)} \\ &\text{porre} && \boldsymbol{\delta}\mathbf{f}^{(k)} = \mathbf{f}\left(\mathbf{x}^{(k+1)}\right) - \mathbf{f}\left(\mathbf{x}^{(k)}\right) \\ &\text{calcolare} && \mathrm{B}_{k+1} = \mathrm{B}_k + \frac{\left(\boldsymbol{\delta}\mathbf{f}^{(k)} - \mathrm{B}_k \boldsymbol{\delta}\mathbf{x}^{(k)}\right) \boldsymbol{\delta}\mathbf{x}^{(k)^T}}{\boldsymbol{\delta}\mathbf{x}^{(k)^T} \boldsymbol{\delta}\mathbf{x}^{(k)}} \end{aligned} \tag{12}$$

  Da notare che non si chiede alla successione $\{\mathrm{B}_k\}$ così costruita di convergere alla vera matrice Jacobiana $\mathbf{J_f}(\boldsymbol{\alpha})$ ($\boldsymbol{\alpha}$ è la radice del sistema); questo risultato non è garantito tuttavia.

1.6 Iterazioni di punto fisso

Esempio 3: esempio di introduzione

Con una calcolatrice si può facilmente verificare che applicando ripetutamente la funzione cos partendo dal numero 1 si genera la seguente successione di numeri reali:

$$\begin{aligned} x^{(1)} &= \cos(1) &&= 0.54030230586814 \\ x^{(2)} &= \cos\left(x^{(1)}\right) &&= 0.85755321584639 \\ &\vdots \\ x^{(10)} &= \cos\left(x^{(9)}\right) &&= 0.74423735490056 \\ &\vdots \\ x^{(20)} &= \cos\left(x^{(19)}\right) &&= 0.73918439977149 \end{aligned}$$

Che tende al valore $\alpha = 0.73908513$.

Con l'esempio di introduzione è possibile capire il punto fisso. Essendo per costruzione $x^{(k+1)} = \cos\left(x^{(k)}\right)$ per $k = 0, 1, \ldots$ (con $x^{(0)} = 1$), α è tale che $\cos(\alpha) = \alpha$. Quindi, α viene detto punto fisso della funzione coseno.

Perché è interessante?

Se α è un punto fisso per il coseno, allora esso è uno zero della funzione $f(x) = x - \cos(x)$ ed il metodo appena proposto potrebbe essere usato per il calcolo degli zeri di f.

Non tutte le funzioni hanno un punto fisso

Non tutte le funzioni ammettono punti fissi. Ad esempio, ripetendo l'esperimento dell'esempio con una funzione esponenziale a partire da $x^{(0)} = 1$, dopo soli 4 passi si giunge ad una situazione di *overflow* (figura 1, pagina 15).

Definizione 2

Data una funzione $\phi : [a, b] \to \mathbb{R}$, trovare $\alpha \in [a, b]$ tale che:

$$\alpha = \phi(\alpha)$$

Se tale α esiste, viene detto un **punto fisso** di ϕ e lo si può determinare come limite della seguente successione:

$$x^{(k+1)} = \phi\left(x^{(k)}\right) \qquad k \geq 0 \tag{13}$$

Dove $x^{(0)}$ è un dato iniziale. L'algoritmo viene chiamato **iterazioni di punto fisso** e la funzione ϕ è detta **funzione di iterazione**.

Dalla definizione, si deduce che l'esempio introduttivo è un algoritmo di iterazioni di punto fisso per la funzione $\phi(x) = \cos(x)$.

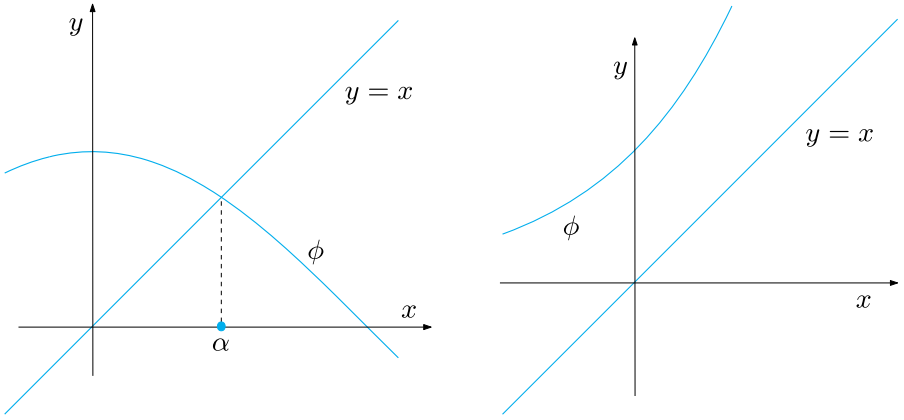


Figura 1: La funzione $\phi(x) = \cos(x)$ (sx) ammette un solo punto fisso, mentre la funzione $\phi(x) = e^x$ (dx) non ne ammette alcuno.

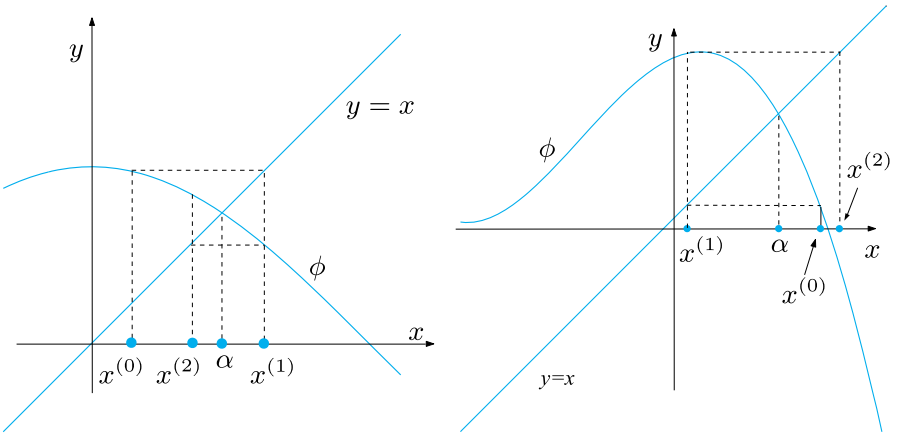


Figura 2: Rappresentazione delle prime iterazioni di punto fisso per due funzioni di iterazione. Le iterazioni convergono verso il punto fisso α (sx), mentre si allontanano da α (dx).

Definizione 3: quando una funzione ha un punto fisso?

Si consideri la successione (formula) 13 a pagina 14.

1. Si supponga che $\phi(x)$ sia continua nell'intervallo $[a,b]$ e che $\phi(x) \in [a,b]$ per ogni $x \in [a,b]$; allora **esiste almeno un punto fisso** $\alpha \in [a,b]$.

2. Si supponga inoltre che esista un valore L minore di 1 tale per cui:

$$|\phi(x_1) - \phi(x_2)| \leq L\,|x_1 - x_2|$$

Per ogni x_1, x_2 appartenente all'insieme $[a,b]$. Con tale supposizione, allora ϕ ha un **unico punto fisso** $\phi \in [a,b]$ e la successione definita nell'equazione 13 a pagina 14 converge a α, qualunque sia il dato iniziale $x^{(0)}$ in $[a,b]$.

La supposizione scritta in precedenza può essere riassunta in un'equazione:

$$\exists L < 1 \;\; t.c. \;\; |\phi(x_1) - \phi(x_2)| \leq L\,|x_1 - x_2| \quad \forall x_1, x_2 \in [a,b] \quad (14)$$

Nella pratica è però spesso **difficile delimitare a priori l'ampiezza dell'intervallo** $[a,b]$; in tal caso è utile il seguente risultato di convergenza locale:

Teorema 1 (di Ostrowski)**.** *Sia α un punto fisso di una funzione ϕ continua e derivabile con continuità in un opportuno intorno $\mathcal{J}$ di α. Se risulta $|\phi'(\alpha)| < 1$, allora esiste un $\delta > 0$ in corrispondenza del quale la successione $\left\{x^{(k)}\right\}$ converge ad α, per ogni $x^{(0)}$ tale che $\left|x^{(0)} - \alpha\right| < \delta$. Inoltre si ha:*

$$\lim_{k\to\infty} \frac{x^{(k+1)} - \alpha}{x^{(k)} - \alpha} = \phi'(\alpha) \quad (15)$$

Dal teorema si deduce che le iterazioni di punto fisso convergono almeno linearmente cioè che, per k sufficientemente grande, l'errore del passo $k+1$ si comporta come l'errore al passo k moltiplicato per una costante, $\phi'(\alpha)$ nel teorema, indipendente da k ed il cui valore assoluto è minore di 1. Per questo motivo la costante viene chiamata **fattore di convergenza** e la convergenza sarà tanto più rapida quanto più piccola è tale costante.

Definizione 4: quando il metodo di punto fisso è convergente

Si suppongano valide le ipotesi del teorema di Ostrowski 1. Se, inoltre, ϕ è derivabile con continuità due volte e se:

$$\phi'(\alpha) = 0 \qquad \phi''(\alpha) \neq 0$$

Allora il metodo di punto fisso (eq. 13) è convergente di ordine 2 e si ha:

$$\lim_{k\to\infty} \frac{x^{(k+1)} - \alpha}{\left(x^{(k)} - \alpha\right)^2} = \frac{1}{2}\phi''(\alpha) \quad (16)$$

Un'ultima osservazione interessante:

- Nel caso in cui $|\phi'(\alpha)| > 1$, se $x^{(k)}$ è sufficientemente vicino ad α, in modo tale che $\left|\phi'\left(x^{(k)}\right)\right| > 1$, allora $\left|\alpha - x^{(k+1)}\right| > \left|\alpha - x^{(k)}\right|$, e **non è possibile che la successione converga al punto fisso**.
- Nel caso in cui $|\phi'(\alpha)| = 1$ **non si può trarre alcuna conclusione** perché potrebbero verificarsi sia la convergenza sia la divergenza, a seconda delle caratteristiche della funzione di punto fisso.

2 Metodi risolutivi per sistemi lineari e non lineari

2.1 Metodi diretti per sistemi lineari

2.1.1 Metodo delle sostituzioni in avanti e all'indietro

Perché sono importanti i metodi numerici?

Si consideri il seguente **sistema lineare**:

$$Ax = b$$

Dove:

- $A \in \mathbb{R}^{n \times n}$ di componenti a_{ij} e $b \in \mathbb{R}^n$ sono valori noti.
- $x \in \mathbb{R}^n$ è il vettore delle incognite.
- La costante n rappresenta il numero di equazioni lineari delle incognite x_j.

Con queste caratteristiche, è possibile rappresentare la i-esima equazione nel seguente modo:

$$\sum_{j=1}^{n} a_{ij}x_j = b_i \quad \rightarrow \quad a_{1i}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n = b_i \qquad \forall i = 1, \ldots, n$$

La **soluzione esatta** del sistema, chiamata **formula di Cramer**, è:

$$x_j = \frac{\det(A_j)}{\det(A)} \tag{17}$$

Con $A_j = |a_1 \ \ldots \ a_{j-1} \ b \ a_{j+1} \ \ldots \ a_n|$ e a_i le colonne di A. Ovviamente la soluzione **esiste ed è unica se il determinante** della matrice A è **diverso da zero**:

$$\det(A) \neq 0$$

Purtroppo questo metodo è **inutilizzabile** poiché il calcolo di un determinante richiede all'incirca $n!$ (fattoriale di n) operazioni.

Risulta evidente che sia necessario uno studio approfondito di **metodi numerici che si traducano in algoritmi efficienti** da farli eseguire su calcolatori. Nelle seguenti pagine si introducono i primi due algoritmi "efficienti".

Definizione 1

Il seguente algoritmo rappresenta il **metodo delle sostituzioni in avanti**. Dati:

- $L \in \mathbb{R}^{n\times n}$ matrice triangolare inferiore non singolare (cioè con determinante diverso da zero $\det(L) \neq 0$)
- $\mathbf{b} \in \mathbb{R}^n$ vettore termine noto

La soluzione è data da $Lx = \mathbf{b}$ con $x \in \mathbb{R}^n$. Più in generale si ha:

$$x_i = \frac{b_i - \sum_{j=1}^{i=1} L_{ij}\, x_j}{L_{ii}} \tag{18}$$

Per comprendere meglio la definizione del metodo di sostituzioni in avanti, è possibile visualizzare in modo generale la matrice triangolare inferiore L (non singolare):

$$L_{n\times n} = \begin{bmatrix} L_{11} & 0 & 0 & 0 & 0 \\ & \ddots & 0 & 0 & 0 \\ & & \ddots & 0 & 0 \\ & L_{ij} & & \ddots & 0 \\ & & & & L_{nn} \end{bmatrix}$$

Dove ovviamente n è la grandezza della matrice quadrata. Dalla matrice, è possibile rappresentare le prime tre iterazioni, ovvero x_1, x_2 e x_3:

- La prima riga è:
$$x_1 = \frac{b_1}{L_{11}}$$
Si può scrivere anche in linea nel seguente modo: $L_{11}x_1 = b_1$.
- La seconda riga:
$$x_2 = \frac{b_2 - (L_{21}\cdot x_1 + L_{22}\cdot \cancelto{0}{x_2})}{L_{22}}$$
In cui x_1 è il risultato del punto precedente e x_2 è il risultato che attualmente si sta calcolando, quindi uguale a zero.
- La terza riga:
$$x_3 = \frac{b_3 - (L_{31}\cdot x_1 + L_{32}\cdot x_2 + L_{33}\cdot \cancelto{0}{x_3})}{L_{33}}$$

Il **numero di operazioni** richieste dal metodo delle sostituzioni in avanti è dato da 1 sottrazione, $i-1$ moltiplicazioni, $i-2$ addizioni e 1 divisione:

$$\#op. = \sum_{i=1}^{n} (i-1) + (i-2) + 1 + 1 = \sum_{i=1}^{n} (2i-1) = n^2 \tag{19}$$

Per completezza si presenta anche il metodo delle sostituzioni all'indietro.

Definizione 2

Il seguente algoritmo rappresenta il **metodo delle sostituzioni all'indietro**. Dati:

- $U \in \mathbb{R}^{n\times n}$ matrice triangolare superiore non singolare (cioè con determinante diverso da zero $\det(U) \neq 0$)
- $\mathbf{b} \in \mathbb{R}^n$ vettore termine noto

La soluzione è data da $Ux = \mathbf{b}$ con $x \in \mathbb{R}^n$. Più in generale si ha:

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} U_{ij}\, x_j}{U_{ii}} \tag{20}$$

Come per il metodo precedente, anche in questo caso è utile visualizzare la matrice generale:

$$U_{n\times n} = \begin{bmatrix} U_{11} & & & & \\ 0 & \ddots & & U_{ij} & \\ 0 & 0 & \ddots & & \\ 0 & 0 & 0 & \ddots & \\ 0 & 0 & 0 & 0 & U_{nn} \end{bmatrix}$$

Anche da questa matrice è possibile rappresentare le iterazioni:

- La prima riga è:
$$x_1 = \frac{b_1 - (U_{12}\cdot x_2 + U_{13}\cdot x_3 + U_{14}\cdot x_4)}{U_{11}}$$
- La seconda riga è:
$$x_2 = \frac{b_2 - (U_{23}\cdot x_3 + U_{24}\cdot x_4)}{U_{22}}$$
- La terza riga è:
$$x_3 = \frac{b_3 - (U_{34}\cdot x_4)}{U_{33}}$$
- L'ultima riga è:
$$x_4 = \frac{b_4}{U_{44}}$$

Si deduce ovviamente che l'ultima riga può essere generalizzata nel seguente modo:

$$x_n = \frac{b_n}{U_{nn}}$$

Infine, il **numero di operazioni** è il medesimo del metodo delle sostituzioni in avanti, quindi n^2.

2.1.2 Fattorizzazione LU: MEG e Cholesky

Sia $A \in \mathbb{R}^{n\times n}$. Si supponga che esistano due opportune matrici L ed U, triangolare inferiore e superiore, rispettivamente, tali che:

$$A = LU \tag{21}$$

L'equazione viene chiamata **fattorizzazione LU** (o **decomposizione LU**) di A.

La fattorizzazione LU è stata introdotta poiché se A **non è singolare** (quindi il determinante è diverso da zero) tali matrici devono essere **anch'esse non singolari**; questo assicura che i loro **elementi diagonali siano non nulli**. Da questa osservazione, si ottiene un risultato interessante perché la risoluzione di $Ax = b$ è *equivalente* alla risoluzione dei due seguenti sistemi triangolari:

$$Ly = b \qquad Ux = y \tag{22}$$

Dove y rappresenta la soluzione dell'equazione 18 a pagina 19, ovvero la risoluzione del metodo delle sostituzioni in avanti. Analogamente, la x rappresenta la soluzione dell'equazione 20 a pagina 20, ovvero la risoluzione del metodo delle sostituzioni all'indietro.

Chiaro, ma che algoritmi esistono per calcolare la fattorizzazione LU?

Esistono principalmente due algoritmi: il **Metodo di Eliminazione di Gauss (MEG)** e la **Fattorizzazione di Cholesky**.

- Senza entrare troppo nel dettaglio, la fattorizzazione LU viene chiamata anche fattorizzazione di Gauss poiché è dimostrato che è possibile applicare l'algoritmo di Gauss, ovvero il **Metodo di Eliminazione di Gauss (MEG)**.

  Il **MEG** è possibile **applicarlo** per alcuni tipi di matrici:

  1. Le **matrici a dominanza diagonale stretta**. Una matrice è detta **matrice a dominanza diagonale per righe** se:

  $$|a_{ii}| \geq \sum_{j=1\wedge j\neq i}^{n} |a_{ij}|\,, \qquad i = 1,\dots,n \tag{23}$$

  Una matrice è detta **matrice a dominanza diagonale per colonne** se:

  $$|a_{ii}| \geq \sum_{j=1\wedge j\neq i}^{n} |a_{ji}|\,, \qquad i = 1,\dots,n \tag{24}$$

  Quando nelle precedenti disuguaglianze è possibile sostituire il segno $\geq$ con quello $>$ si può dire che la matrice A è a dominanza diagonale **stretta**.

  2. Le **matrici reali simmetriche**[2]**e definite positive**[3].

[2] Una matrice è simmetrica se coincide con la sua matrice trasposta.

[3] Una matrice viene **definita positiva** se:

$$\forall \mathbf{x} \in \mathbb{R}^n \qquad \text{con } \mathbf{x} \neq \mathbf{0}, \qquad \mathbf{x}^T \mathrm{A}\mathbf{x} > 0$$

Il **calcolo dei coefficienti** dei fattori L ed U **richiede** circa $\frac{2n^3}{3}$ **operazioni**.

- Se la matrice A, cioè la matrice usata nella definizione, è **simmetrica** e **definita positiva**, è possibile trovare la **fattorizzazione di Cholesky**:

$$A = R^T R \tag{25}$$

Dove R è una matrice triangolare superiore con elementi positivi sulla diagonale. Inoltre, tale fattorizzazione è **unica**.

Il **calcolo della matrice** R **richiede** circa $\frac{n^3}{3}$ **operazioni** (cioè la **metà di quelle richieste per calcolare le due matrici della fattorizzazione LU**).

2.1.3 La tecnica del pivoting

Si introduce un metodo che consenta di portare a compimento il processo di fattorizzazione LU per una qualunque matrice A non simmetrica ($\det(A) \neq 0$).

La tecnica si basa sulla **permutazione** (cioè sullo scambio) opportuno di righe della matrice di partenza A. Purtroppo non è noto a priori quali siano le righe che dovranno essere tra loro scambiate; tuttavia questa decisione può essere presa ad ogni passo durante il quale si generino elementi nulli.

Dato che lo scambio tra righe comporta un cambiamento del pivot, questa tecnica viene chiamata **pivoting per righe**. La fattorizzazione che si trova restituisce la matrice A di partenza a meno di una permutazione fra le righe. Precisamente:

$$PA = LU \tag{26}$$

Dove P è un'opportuna **matrice di permutazione**. Ovvero, è una matrice uguale all'identità all'inizio del processo di fattorizzazione e se durante l'applicazione le righe di A vengono scambiate, allora deve essere fatto uno scambio analogo sulle righe di P. Per cui, alla fine sarà necessario risolvere i seguenti sistemi triangolari:

$$L\mathbf{y} = P\mathbf{b} \qquad U\mathbf{x} = \mathbf{y} \tag{27}$$

Dove y rappresenta la soluzione dell'equazione 18 a pagina 19, ovvero la risoluzione del metodo delle sostituzioni in avanti. Analogamente, la x rappresenta la soluzione dell'equazione 20 a pagina 20, ovvero la risoluzione del metodo delle sostituzioni all'indietro.

2.1.4 Errori di arrotondamento nel MEG

Prima di introdurre l'errore di arrotondamento nel Metodo di Eliminazione di Gauss, è necessario capire il problema alla fonte.

In generale, un elaboratore memorizza i numeri nel seguente modo:

$$x = (-1)^s \cdot (0.a_1a_2 \dots a_t) \cdot \beta^e = (-1)^s \cdot m \cdot \beta^{e-t} \qquad \text{con } a_1 \neq 0 \tag{28}$$

Dove:

- s è il **segno** e può essere uguale a 0 o 1.
- β è la **base** e può essere un numero intero positivo maggiore od uguale a 2.
- m è la **mantissa**, un intero la cui **lunghezza** t è il **numero massimo di cifre** a_i (con $0 \leq a_i \leq \beta - 1$) **memorizzabili**.
- e è l'**esponente** ed è un intero.

I numeri con un formato identico all'equazione 28 sono detti numeri **floating-point normalizzati** essendo variabile la posizione del punto decimale. Inoltre, le cifre $a_1a_2 \dots a_p$ (con $p \leq t$) vengono generalmente chiamate le **prime p cifre significative di x**.

Si noti che la condizione $a_1 \neq 0$ nell'equazione 28 impedisce che lo stesso numero possa avere più rappresentazioni (per esempio $0.1 \cdot 10^0$ uguale a $0.01 \cdot 10^1$).

L'insieme $\mathbb{F}$ è dunque l'**insieme dei numeri *floating point*** ed è completamente caratterizzato dalla base β, dal numero di cifre significative t e dall'intervallo (L, U) (con $L < 0$ ed $U > 0$) di variabilità dell'esponente e.

Sostituendo ad un numero reale $x \neq 0$ il suo rappresentante *floating point* $fl(x)$ in $\mathbb{F}$, è inevitabile un **errore di arrotondamento** uguale a:

$$\frac{|x - fl(x)|}{|x|} \leq \frac{1}{2}\epsilon_M \tag{29}$$

Dove:

- $\epsilon_M = \beta^{1-t}$ è la **epsilon macchina**, ovvero la distanza fra 1 ed il più piccolo numero *floating-point* maggiore di 1.
- $|x|$ è l'**errore relativo**.
- $|x - fl(x)|$ è l'**errore assoluto**.
- Il numero $u = \frac{1}{2}\epsilon_M$ è l'**unità di arrotondamento** poiché rappresenta il **massimo errore relativo che la macchina può commettere nella rappresentazione di un numero reale**.

Date queste osservazioni, si possono ricavare anche il **più grande** ed il **più piccolo numero positivo** di $\mathbb{F}$:

$$x_{\min} = \beta^{L-1} \qquad x_{\max} = \beta^U \cdot \left(1 - \beta^{-t}\right)$$

- Se un numero è minore del numero più piccolo positivo, allora si ha una situazione di **underflow**.
- Se un numero è maggiore del numero più grande positivo, allora si ha una situazione di **overflow**.

Quindi che cosa accade in una somma tra valori molto grandi?

Ottima domanda. Quando si sommano tra loro numeri che hanno all'incirca lo stesso module, ma segno opposto, il risultato della somma può essere assai impreciso e ci si riferisce a questa situazione con l'espressione **cancellazione di cifre significative**.

Risulta quindi necessario fare una distinzione. L'aritmetica (o la logica) utilizzata dal calcolatore viene chiamata **aritmetica floating-point** (quella spiegata fin ora); al contrario, l'**aritmetica esatta** si basa sulla effettuazione esatta delle operazioni elementari (quindi senza tener conto degli errori di arrotondamento) su operandi noti esattamente (e non attraverso la loro rappresentazione *floating-point*).

Va bene, ma perché è importante considerare questo aspetto?

Nonostante gli errori di arrotondamento sono generalmente piccoli, se ripetuti all'interno di algoritmi lunghi e complessi, possono portare ad effetti catastrofici (vedi per esempio l'incidente del razzo Ariane 5).

Adesso si prova a definire in modo formale questo comportamento.

- Con e_a si identificano i **tipi di errore che si manifestano a seguito di una serie di errori di arrotondamento**.
- Con e_t si identifica l'**errore di troncamento**. Tali errori sono assenti soltanto in quei modelli matematici che sono già di dimensione finita (per esempio nella risoluzione di un sistema lineare).
- Con e_c si identifica l'**errore computazionale**, ovvero l'insieme degli errori e_a e e_t.

Indicando con x la soluzione esatta del modello matematico e con $\widehat{x}$ la soluzione ottenuta al termine del processo numerico, allora l'**errore computazionale assoluto** sarà dunque:

$$e_c^{ass} = |x - \widehat{x}| \tag{30}$$

Nel caso in cui l'errore relativo fosse diverso da zero ($x \neq 0$):

$$e_c^{rel} = \frac{|x - \widehat{x}|}{|x|} \tag{31}$$

Relazione tra gli errori di arrotondamento e MEG

L'introduzione fatta nelle pagine precedenti è servita perché è possibile trovare errori di arrotondamento con il prodotto LU, la quale non ritorna esattamente la matrice A.

Come appena accennato, il **prodotto LU produce un errore di arrotondamento**. Esso può essere **ridotto usando la tecnica del pivoting** che consente di contenerlo.

Inoltre, quando si risolve numericamente il sistema lineare $\mathrm{A}\mathbf{x} = \mathbf{b}$ si può trovare la **soluzione esatta** $\widehat{\mathbf{x}}$ di un **sistema perturbato** della forma:

$$(\mathrm{A} + \delta\mathrm{A})\,\widehat{\mathbf{x}} = \mathbf{b} + \delta\mathbf{b} \tag{32}$$

Dove $\delta\mathrm{A}$ e $\delta\mathbf{b}$ sono rispettivamente una matrice ed un vettore di perturbazione che dipendono dallo specifico metodo numerico impiegato nella risoluzione del sistema.

Usando le norme si ha:

$$\frac{||\mathbf{x} - \widehat{\mathbf{x}}||}{||\mathbf{x}||} \leq \frac{\lambda_{\max}}{\lambda_{\min}} \cdot \frac{||\delta\mathbf{b}||}{||\mathbf{b}||} \tag{33}$$

Dove l'errore relativo sulla soluzione dipende dall'errore relativo sui dati attraverso la seguente constante (≥ 1):

$$K\,(A) = \frac{\lambda_{\max}}{\lambda_{\min}} \tag{34}$$

Essa viene chiamata **numero di condizionamento (spettrale) della matrice** A. Ovviamente, si ricorda che tale matrice A deve essere simmetrica e definita positiva.
Se la matrice A è una matrice simmetrica e definita positiva e δA una matrice non nulla simmetrica e definita positiva tale che $\lambda_{\max}\,(\delta A) < \delta_{\min}\,(A)$, allora vale:

$$\frac{||\mathbf{x} - \widehat{\mathbf{x}}||}{||\mathbf{x}||} \leq \frac{K\,(A)}{1 - \dfrac{\lambda_{\max}\,(\delta A)}{\lambda_{\min}\,(A)}} \cdot \left(\frac{\lambda_{\max}\,(\delta A)}{\lambda_{\max}\,(A)} + \frac{||\delta\mathbf{b}||}{||\mathbf{b}||} \right) \tag{35}$$

Infine, se le matrici A e δA non sono simmetriche e definite positive e δA è tale che $||\delta\mathrm{A}||_2 \cdot \left|\left|\mathrm{A}^{-1}\right|\right|_2 < 1$, vale la seguente stima:

$$\frac{||\mathbf{x} - \widehat{\mathbf{x}}||}{||\mathbf{x}||} \leq \frac{K_2\,(A)}{1 - \dfrac{K_2\,(A) \cdot ||\delta\mathrm{A}||_2}{||\mathrm{A}||_2}} \cdot \left(\frac{||\delta\mathrm{A}||_2}{||\mathrm{A}||_2} + \frac{||\delta\mathbf{b}||}{||\mathbf{b}||} \right) \tag{36}$$

Essendo $||\delta\mathrm{A}||_2 = \sqrt{\lambda_{\max}\,(A^T A)}$ e:

$$K_2\,(A) = ||\delta\mathrm{A}||_2 \cdot \left|\left|\delta\mathrm{A}^{-1}\right|\right|_2 \tag{37}$$

Il **numero di condizionamento in norma 2**.

- Se $K_2(\mathrm{A})$ (o $K(\mathrm{A})$) è "piccolo" la matrice A viene detta **ben condizionata** ed a **piccoli errori sui dati corrisponderanno errori dello stesso ordine di grandezza sulla soluzione**.
- Se $K_2(\mathrm{A})$ (o $K(\mathrm{A})$) è "grande" la matrice A viene detta **mal condizionata** ed *potrebbe accadere* che a **piccole perturbazioni sui dati corrispondano grandi errori sulla soluzione**.

Infine, volendo l'equazione 33 può essere riscritta introducendo il **residuo** $\mathbf{r}$:

$$\mathbf{r} = \mathbf{b} - \mathrm{A}\widehat{\mathbf{x}} \tag{38}$$

Chiaramente se $\widehat{\mathbf{x}}$ fosse la **soluzione esatta**, il **residuo sarebbe il vettore nullo**.

L'**efficacia** del residuo dipende dalla grande del numero di condizionamento di A. Infatti, sempre dall'equazione 33 si ricava:

$$\frac{||\mathbf{x} - \widehat{\mathbf{x}}||}{||\mathbf{x}||} \leq K_2(\mathrm{A}) \cdot \frac{||\mathbf{r}||}{||\mathbf{b}||} \tag{39}$$

- Se $K_2(\mathrm{A})$ è "piccolo" si avrebbe la **certezza che l'errore sarà piccolo quando lo è il residuo**.
- Se $K_2(\mathrm{A})$ è "grande" non si può avere la certezza che l'errore sarà piccolo quando lo è il residuo.

2.1.5 Il pivoting totale

Si opera un **pivoting totale** quando la ricerca del pivot è estesa alla sottomatrice $A^{(k)}$ costituita dagli elementi $a_{ij}^{(k)}$ con $i, j = k, \ldots, n$. A differenza della tecnica del pivoting introdotta nel capitolo 2.1.3 a pagina 22, il parziale prevede il **coinvolgimento delle righe e delle colonne**, e conduce alla costruzione di due matrici di permutazione, chiamate P e Q, una sulle righe, l'altra sulle colonne:

$$\mathrm{PAQ} = \mathrm{LU} \tag{40}$$

Inoltre, la **soluzione** del sistema $A\mathbf{x} = \mathbf{b}$ è ottenuta attraverso la **risoluzione di due sistemi triangolari e di una permutazione**:

$$\mathrm{L}\mathbf{y} = \mathrm{P}\mathbf{b} \qquad \mathrm{U}\mathbf{x}^* = \mathbf{y} \qquad \mathbf{x} = \mathrm{Q}\mathbf{x}^* \tag{41}$$

Dal punto di vista **computazionale**, la tecnica del pivoting totale ha un **costo superiore rispetto a quello parziale** (capitolo 2.1.3 a pagina 22) in quanto ad ogni passo della fattorizzazione devono essere svolti molti più confronti. Tuttavia, può **apportare dei vantaggi in termini di risparmio di memoria e di stabilità**.

Quindi riassumendo:

Come funziona?

Può essere visto come un pivoting parziale, ma in cui c'è un coinvolgimento anche delle colonne e non solo delle righe.

Svantaggi

Più costoso rispetto al pivoting parziale poiché opera su più fronti e quindi devono essere eseguiti più confronti.

Vantaggi

Non sempre, ma spesso si possono ottenere risparmi di memoria e un'alta stabilità.

2.1.6 Il *fill-in* di una matrice

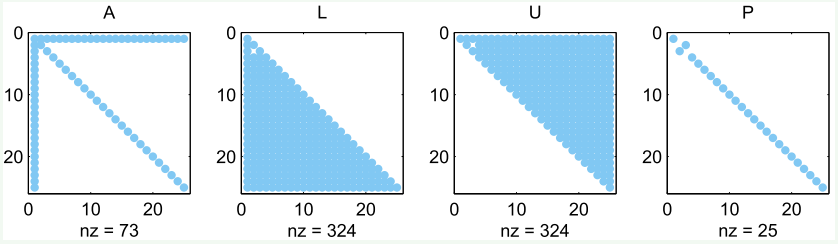
Un altro problema che è possibile riscontrare durante la fattorizzazione LU è il **fill-in**.

Come si manifesta il fenomeno del *fill-in*?

Durante la fattorizzazione LU, non è certo che le matrici L ed U ottenute mantengano la struttura del corrispondente triangolo della matrice A iniziale. Al contrario, il **processo di fattorizzazione tende** generalmente a **riempire le matrici** L ed U. Tale fenomeno **dipende fortemente dalla struttura e dal valore dei singoli elementi non nulli** della matrice A.

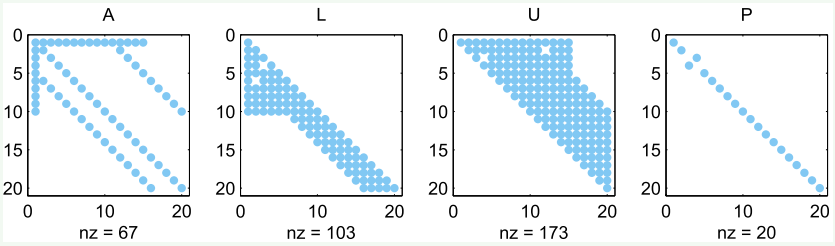
Esempio 1

Qua di seguito è possibile osservare un fenomeno di *fill-in* su una generica matrice.



Esempio 2

Un altro esempio di *fill-in* su una generica matrice. In questo caso, gli elementi non nulli della prima riga e della prima colonna di A inducono un riempimento totale delle corrispondenti colonne in U e righe in L, rispettivamente, mentre gli elementi non nulli sopra e sotto le diagonali di A comportano un riempimento delle diagonali superiori di U ed inferiori di L comprese tra quella principale e quelle non nulle di A.



Come risolvere il *fill-in*?

Per ovviare al problema del *fill-in*, si possono adottare **tecniche di riordinamento che permutano righe e colonne della matrice prima di realizzare la fattorizzazione**. Tuttavia, in alcuni casi la **sola applicazione della tecnica di pivoting totale** (paragrafo 2.1.5 pagina 27) consente di raggiungere lo stesso obiettivo.

Prima di introdurre un nuovo esempio, si forniscono due definizioni.

Definizione 3

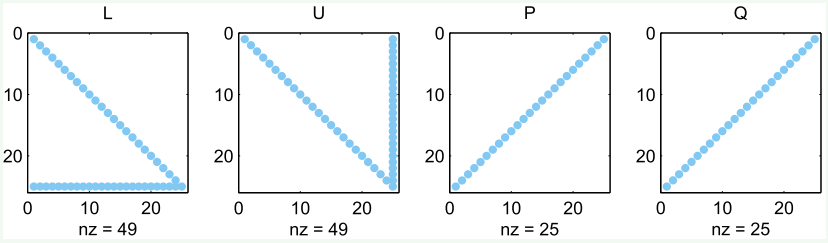
Una matrice quadrata di dimensione n è detta **matrice sparsa** se ha un numero di elementi non nulli dell'ordine di n (e non di n^2).

Definizione 4

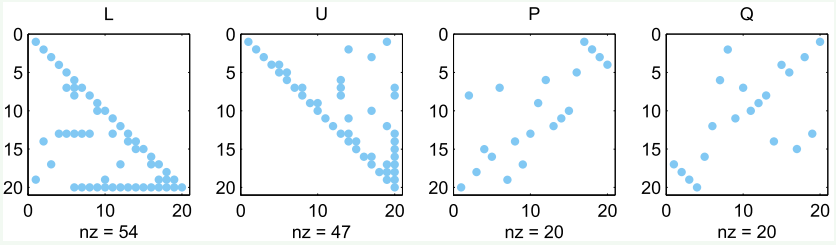
Si chiama **pattern** di una matrice sparsa l'insieme dei suoi elementi non nulli.

Esempio 3

Nella seguente figura sono mostrati i pattern delle matrici ottenute dalla fattorizzazione con pivotazione totale. Come si può vedere, il fenomeno di *fill-in* è contenuto.



Un altro esempio.



2.2 Metodi iterativi per sistemi lineari

Un **metodo iterativo** per la risoluzione del sistema lineare $Ax = b$ con:

- $A \in \mathbb{R}^{n \times n}$
- $b \in \mathbb{R}^n$
- $x \in \mathbb{R}^n$
- $\det(A) \neq 0$

Consiste nel costruire una successione di vettori del tipo:

$$\left\{\mathbf{x}^{(k)},\ k \geq 0\right\}$$

Di $\mathbb{R}^n$ che **converge** alla soluzione esatta $\mathbf{x}$, ossia tale che:

$$\lim_{k \to \infty} \mathbf{x}^{(k)} = \mathbf{x} \tag{42}$$

Per un qualunque vettore iniziale $\mathbf{x}^{(0)} \in \mathbb{R}$, ossia la **convergenza non deve dipendere dalla scelta di** $\mathbf{x}^{(0)}$.

Inoltre, si definisce ricorsivamente:

$$\mathbf{x}^{(k+1)} = \mathrm{B}\mathbf{x}^{(k)} + \mathbf{g} \qquad k \geq 0 \tag{43}$$

Essendo:

- B una matrice opportuna (dipendente da A)
- $\mathbf{g}$ un vettore opportuno (dipendente da A e da $\mathbf{b}$)

E come si possono scegliere questi valori?

La scelta della matrice B e del vettore $\mathbf{g}$ è piuttosto semplice. Devono essere **rispettate due proprietà**:

- **Condizione di consistenza**. I valori devono garantire che:

  $$\mathbf{x} = \mathrm{B}\mathbf{x} + \mathbf{g} \tag{44}$$

  Essendo $\mathbf{x} = \mathrm{A}^{-1}\mathbf{b}$, necessariamente dovrà aversi $\mathbf{g} = (\mathrm{I} - \mathrm{B})\,\mathrm{A}^{-1}\mathbf{b}$.

- **Condizione di convergenza**. Prima di dare la condizione, è necessario comprendere alcune cose. Prima di tutto, la seguente espressione:

  $$\mathbf{e}^{(k)} = \mathbf{x} - \mathbf{x}^{(k)}$$

  Viene identificata come l'**errore al passo** k. Adesso sottraendo l'equazione 43 dalla condizione di consistenza 44 si ottiene:

  $$\mathbf{e}^{(k+1)} = \mathrm{B}\mathbf{e}^{(k)}$$

  Per tale ragione B viene detta **matrice di iterazione** del metodo 43.

  E adesso, si presenta la condizione di convergenza:

– La matrice B è simmetrica[4] e definita positiva[5], allora:

$$\left|\left|\mathbf{e}^{(k+1)}\right|\right| = \left|\left|\mathrm{B}\mathbf{e}^{(k)}\right|\right| \leq \rho\left(\mathrm{B}\right)\left|\left|\mathbf{e}^{(k)}\right|\right| \qquad \forall k \geq 0 \tag{45}$$

Dove $\rho\left(\mathrm{B}\right)$ è il **raggio spettrale** di B ed è il massimo degli autovalori di B. Si tenga conto che nel caso di matrici simmetriche e definite positive esso coincide con il massimo autovalore.

– Iterando a ritroso il punto precedente, si ottiene:

$$\left|\left|\mathbf{e}^{(k)}\right|\right| \leq \left[\rho\left(\mathrm{B}\right)\right]^k \left|\left|\mathbf{e}^{(0)}\right|\right| \qquad k \geq 0 \tag{46}$$

Si noti che se $\rho\left(\mathrm{B}\right) < 1$, allora $\mathbf{e}^{(k)} \to \mathbf{0}$ per $k \to \infty$ per ogni possibile $\mathbf{e}^0$ (e, conseguentemente, per ogni $\mathbf{x}^{(0)}$).

In generale, la condizione sufficiente e necessaria di convergenza è la seguente.

Definizione 5: condizione sufficiente e necessaria di convergenza

Un metodo iterativo della forma dell'equazione 43, la cui matrice di iterazione B soddisfi la condizione di consistenza (eq. 44), è **convergente** per ogni $\mathbf{x}^{(0)}$ *se e soltanto se* $\rho\left(\mathrm{B}\right) < 1$ (raggio spettrale minore di 1).

Inoltre, minore è $\rho\left(\mathrm{B}\right)$, minore è il numero di iterazioni necessarie per ridurre l'errore iniziale di un dato fattore.

Perché introdurre i metodi iterativi?

L'esigenza di introdurre i metodi iterativi sorge nel momento in cui si ragiona sulla quantità di tempo spesa da un calcolatore per eseguire la fattorizzazione LU su matrici di grandi dimensioni. Difatti, con matrici con ordini di 10^7, sono necessari circa 11 giorni.

[4] Quindi corrisponde con la sua trasposta
[5] Definizione a pagina: 21

2.2.1 Il metodo di Jacobi

Se i coefficienti diagonali della matrice A non sono nulli, allora:

$$D = \text{diag}\,(a_{11}, a_{22}, \ldots, a_{nn})$$

Ovvero D è la matrice diagonale costruita a partire dagli elementi diagonali di A.

Il **metodo di Jacobi** corrisponde alla forma:

$$\mathrm{D}\mathbf{x}^{(k+1)} = \mathbf{b} - (\mathrm{A} - \mathrm{D})\,\mathbf{x}^{(k)} \qquad k \geq 0$$

Che per componenti assume la forma:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1, j\neq i}^{n} a_{ij} x_j^{(k)} \right) \qquad i = 1, \ldots, n \tag{47}$$

Dove $k \geq 0$ e $\mathbf{x}^{(0)} = \left(x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}\right)^T$ è il vettore iniziale.

Quando converge?

In due casi:

1. Se la matrice $\mathrm{A} \in \mathbb{R}^{n\times n}$ è a dominanza diagonale stressa per righe[6], allora il metodo di Jacobi converge.
2. Con la seguente definizione (si veda il paragrafo 2.2.2 per comprendere meglio le definizioni):

Definizione 6: convergenza di Jacobi e Gauss-Seidel

Se $A \in \mathbb{R}^{n\times n}$ è una **matrice tridiagonale**, quindi una matrice quadrata che al di fuori della diagonale principale e delle linee immediatamente al di sopra e al di sotto di essa (la prima sovradiagonale e la prima sottodiagonale), ha solo valori nulli. Un esempio

[6] In algebra lineare una **matrice a dominanza diagonale stretta per righe** o **matrice diagonale dominante stretta per righe** è una matrice quadrata $A \in \mathbb{C}^{n\times n}$ di ordine n i cui elementi diagonali sono maggiori (non stretta sarebbe maggiori o uguali) in valore assoluto della somma di tutti i restanti elementi della stessa riga in valore assoluto:

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}| \tag{48}$$

In senso non stretto:

$$|a_{ii}| \geq \sum_{j=1, j\neq i}^{n} |a_{ij}| \tag{49}$$

di amtrice tridiagonale:

$$\begin{bmatrix} 1 & 4 & 0 & 0 \\ 3 & 4 & 1 & 0 \\ 0 & 2 & 3 & 4 \\ 0 & 0 & 1 & 3 \end{bmatrix}$$

Se tale matrice è non singolare, quindi $\det(A) \neq 0$, con valori della diagonale diversi da zero, allora i metodi di Jacobi e di Gauss-Seidel sono **entrambi convergenti o entrambi divergenti**.

- Se convergono, la velocità dei metodi:

  $$\text{Gauss-Seidel} \gg \text{Jacobi}$$

  Precisamente il raggio spettrale della matrice di iterazione del metodo di Gauss-Seidel è il quadrato del raggio spettrale di quella del metodo di Jacobi.

HPC curiosity: è interessante notare che il metodo di Jacobi viene facilmente parallelizzato per aumentare le prestazioni di calcolo.

2.2.2 Il metodo di Gauss-Seidel

Con il metodo di Jacobi, ogni componente $x_i^{(k+1)}$ del nuovo vettore $\mathbf{x}^{(k+1)}$ viene calcolata indipendentemente dalle altre. Questa può essere la base di partenza per costruire un nuovo metodo più ottimizzato, poiché se per il calcolo di $x_i^{(k+1)}$ venissero usate le nuove componenti già disponibili $x_j^{(k+1)}$, $j = 1, \ldots, i-1$, assieme con le vecchie $x_j^{(k)}$, $j \geq i$.

Supponendo che gli elementi della diagonale non siano nulli, per $k \geq 0$, il **metodo di Gauss-Seidel**:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right) \qquad i = 1, \ldots, n \tag{50}$$

HPC curiosity: a differenza di Jacobi, questo metodo non può essere parallelizzato, ma è necessario operare in modo sequenziale.

Quando converge?

In tre casi:

1. Se la matrice $A \in \mathbb{R}^{n \times n}$ è a dominanza diagonale stressa per righe (vedi pag. 32), allora il metodo di Gauss-Seidel converge.
2. Se la matrice A è simmetrica (uguale alla sua trasposta) e definita positiva (vedi pag. 21), allora Gauss-Seidel converge.
3. Con la definizione di convergenza data per il metodo di Jacobi a pagina 32

Da notare: **non esistono risultati generali che consentono di affermare che il metodo di Gauss-Seidel converga sempre più rapidamente di quello di Jacobi**, a parte il caso della definizione a pagina 21.

2.2.3 Il metodo di Richardson

Una tecnica generale per costruire un metodo iterativo è basata sulla seguente **decomposizione additiva** (o **splitting**) della matrice A:

$$A = P - (P - A)$$

In cui P è un'opportuna matrice non singolare ($\det(P) \neq 0$) chiamata **precondizionatore** di A. Di conseguenza:

$$P\mathbf{x} = (P - A)\,\mathbf{x} + \mathbf{b}$$

È un sistema purché si ponga:

$$\begin{aligned} B &= P^{-1}(P - A) = I - P^{-1}A \\ \mathbf{g} &= P^{-1}\mathbf{b} \end{aligned}$$

Questa identità suggerisce la definizione del seguente metodo iterativo:

$$P\left(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right) = \mathbf{r}^{(k)} \qquad k \geq 0$$

In cui:

$$\mathbf{r}^{(k)} = \mathbf{b} - A\mathbf{x}^{(k)} \tag{51}$$

Denota il vettore residuo alla k-esima iterazione. Una generalizzazione di questo metodo iterativo è

$$P\left(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right) = \alpha_k \mathbf{r}^{(k)} \qquad k \geq 0 \tag{52}$$

Dove $\alpha_k \neq 0$ è un parametro che può cambiare ad ogni iterazione e che, a priori, servirà a migliorare le proprietà di convergenza della successione $\left\{\mathbf{x}^{(k)}\right\}$.

L'equazione 52 è nota come **metodo di Richardson** o **metodo di Richardson dinamico**; se $\alpha_k = \alpha$ per ogni $k \geq 0$ esso si dice **metodo di Richardson stazionario**.

Questo metodo richiede ad ogni passo di trovare il cosiddetto **residuo precondizionato $\mathbf{z}^{(k)}$** dato dalla soluzione del sistema lineare:

$$P\mathbf{z}^{(k)} = \mathbf{r}^{(k)} \tag{53}$$

Quindi, la nuova iterata è definita da $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{z}^{(k)}$.

Per questa ragione la matrice P deve essere scelta in modo tale che il costo computazionale richiesto dalla risoluzione di 53 sia modesto (ogni matrice P diagonale, tridiagonale o triangolare andrebbe bene a questo scopo).

Quando converge?

Per studiare la convergenza, si definisce la **norma dell'energia** associata alla matrice A:

$$||\mathbf{v}||_A = \sqrt{\mathbf{v}^T A \mathbf{v}} \qquad \forall \mathbf{v} \in \mathbb{R}^n \tag{54}$$

E si definisce la seguente definizione.

Definizione 7

Sia $A \in \mathbb{R}^{n \times n}$.
Per ogni matrice non singolare ($\det \neq 0$) $P \in \mathbb{R}^{n \times n}$ il **metodo di Richardson stazionario converge** se e solo se:

$$|\lambda_i|^2 < \frac{2}{\alpha} \mathrm{Re}\lambda_i \qquad \forall i = 1, \dots, n$$

In cui λ_i sono gli autovalori della matrice risultato di $P^{-1}A$.

In particolare, se gli autovalori della matrice risultato di $P^{-1}A$ sono reali, allora esso converge se e solo se:

$$0 < \alpha\lambda_i < 2 \qquad \forall i = 1, \dots, n$$

Se A e P sono entrambe simmetriche (quindi uguali alle loro rispettive trasposte) e definite positive (definizione pagina 21) il metodo di Richardson stazionario **converge per ogni possibile scelta di** $\mathbf{x}^{(0)}$ se e solo se:

$$0 < \alpha < \frac{2}{\lambda_{\max}}$$

Dove $\lambda_{\max}$ (che è maggiore di zero) è l'autovalore massimo della matrice risultato di $P^{-1}A$.

E ancora, il raggio spettrale $\rho(B_\alpha)$ della matrice di iterazione $B_\alpha = I - \alpha P^{-1}A$ è il minimo quando $\alpha = \alpha_{\text{opt}}$, dove:

$$\alpha_{\text{opt}} = \frac{2}{\lambda_{\min} + \lambda_{\max}} \tag{55}$$

Dove ovviamente $\lambda_{\min}$ è l'autovalore minimo della matrice risultato di $P^{-1}A$.

Infine, se $\alpha = \alpha_{\text{opt}}$, vale la seguente **stima di convergenza**:

$$\left|\left|\mathbf{e}^{(k)}\right|\right|_A \leq \left(\frac{K\left(P^{-1}A\right) - 1}{K\left(P^{-1}A\right) + 1}\right)^k \left|\left|\mathbf{e}^{(0)}\right|\right|_A \qquad k \geq 0 \tag{56}$$

Si noti che la massima velocità di convergenza è data anche dal raggio spettrale:

$$\rho_{\text{opt}} = \frac{K\left(P^{-1}A\right) - 1}{K\left(P^{-1}A\right) + 1} \tag{57}$$

2.2.4 Il metodo del Gradiente e del Gradiente Coniugato

Si definisce la funzione $\Phi : \mathbb{R}^n \to \mathbb{R}$:

$$\Phi(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{x}^T\mathbf{b} \tag{58}$$

Nel caso in cui la matrice A è simmetrica[7] e definita positiva[8], Φ è una funzione convessa, cioè tale che ogni per ogni $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ e per ogni $\alpha \in [0, 1]$ vale:

$$\Phi(\alpha\mathbf{x} + (1-\alpha)\mathbf{y}) \leq \alpha\Phi(\mathbf{x}) + (1-\alpha)\Phi(\mathbf{y}) \tag{59}$$

Ed ammette un unico punto stazionario $\mathbf{x}^*$ che è anche un **punto di minimo locale ed assoluto**. Quindi:

$$\mathbf{x}^* = \underset{\mathbf{x}\in\mathbb{R}^n}{\operatorname{argmin}}\, \Phi(\mathbf{x}) \tag{60}$$

È l'unica soluzione dell'equazione:

$$\nabla\Phi(\mathbf{x}) = A\mathbf{x} - \mathbf{b} = \mathbf{0} \tag{61}$$

Risolvere il **problema di minimo** dell'equazione 60 **equivale** pertanto a **risolvere il sistema lineare**[9].

Per un generico $\overline{\mathbf{x}} \in \mathbb{R}^n$ diverso da $\mathbf{x}^*$, $\nabla\Phi(\mathbf{x})$ è un vettore non nullo di $\mathbb{R}^n$ che individua la direzione lungo cui avviene:

- La **massima crescita** di Φ;
- Di conseguenza $-\nabla\Phi(\overline{\mathbf{x}})$ individua la direzione di **massima decrescita** di Φ a partire da $\overline{\mathbf{x}}$.

Inoltre, ricordando che il vettore residuo nel punto $\overline{\mathbf{x}}$ è $\overline{\mathbf{r}} = \mathbf{b} - A\overline{\mathbf{x}}$, grazie alla direzione lungo cui avviene la massima crescita 61, si ha che il **vettore residuo** è uguale a:

$$\overline{\mathbf{r}} = -\nabla\Phi(\overline{\mathbf{x}}) \tag{62}$$

Cioè il residuo **individua una possibile direzione lungo cui muoversi per avvicinarsi al punto di minimo $\mathbf{x}^*$**.

In generale, un vettore $\mathbf{d}$ rappresenta una **direzione di discesa per Φ** nel punto $\overline{\mathbf{x}}$ se si verificano le seguenti condizioni:

$$\begin{aligned} \nabla\Phi(\overline{\mathbf{x}}) \neq \mathbf{0} \quad &\Rightarrow \quad \mathbf{d}^T\nabla\Phi(\overline{\mathbf{x}}) < 0 \\ \nabla\Phi(\overline{\mathbf{x}}) = \mathbf{0} \quad &\Rightarrow \quad \mathbf{d} = \mathbf{0} \end{aligned} \tag{63}$$

Il **residuo** è pertanto una **direzione di discesa**. È banale dire che quindi per **avvicinarsi al punto di minimo** ci si **muoverà lungo opportune direzioni di discesa**. I **metodi di discesa** sono definiti nel seguente modo.

[7] Quindi corrisponde con la sua trasposta

[8] Definizione a pagina: 21

[9] Il gradiente è un argomento del calcolo differenziale vettoriale e il vettore gradiente di una funzione scalare punta secondo la direzione di massima crescita della funzione stessa. Se si è un po' arrugginiti a riguardo, si può dare un'occhiata su Wikipedia o su YouMath

Definizione 8: metodi di discesa

Assegnato un vettore $\mathbf{x}^{(0)} \in \mathbb{R}^n$, per $k = 0, 1, \dots$ fino a convergenza:

1. Si determina una direzione di discesa $\mathbf{d}^{(k)} \in \mathbb{R}^n$
2. Si determina un passo $\alpha_k \in \mathbb{R}$
3. Si pone $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}$

Ma quindi quale è la differenza tra Gradiente e Gradiente Coniugato?

I metodi del gradiente e del gradiente coniugato sono metodi di discesa che si **differenziano nella scelta delle direzioni di discesa**, mentre la scelta dei passi α_k è *comune* ad entrambi. Per cui, qua di seguito si esami la scelta dei passi, supponendo di aver già calcolato la nuova direzione $\mathbf{d}^{(k)}$.

La **scelta del passo** α_k è comune ad entrambi, quindi:

$$\alpha_k = \underset{\alpha \in \mathbb{R}^n}{\operatorname{argmin}} \, \Phi\left(\mathbf{x}^{(k)} + \alpha \mathbf{d}^{(k)}\right) \tag{64}$$

Ovvero un passo α_k tale che $\Phi\left(\mathbf{x}^{(k+1)}\right)$ sia il **minimo** di Φ lungo la direzione $\mathbf{d}^{(k)}$ passante per $\mathbf{x}^{(k)}$.

Definizione metodo del gradiente

Adesso si può dare la definizione del Gradiente. Il **metodo del gradiente** è caratterizzato dalla scelta:

$$\mathbf{d}^{(k)} = \mathbf{r}^{(k)} = -\nabla\Phi\left(\mathbf{x}^{(k)}\right) \qquad k = 0, 1, \dots \tag{65}$$

Quindi la **direzione di discesa ad ogni passo k è la direzione opposta a quella del gradiente** (come si vede dal meno davanti a ∇) della funzione Φ.

L'**algoritmo del metodo del gradiente** è: dato $\mathbf{x}^{(0)} \in \mathbb{R}^n$, si definisce $\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)}$ e si calcola:

$$\begin{aligned} &\text{per } k = 0, 1, \dots \\ &\qquad \alpha_k = \frac{\left(\mathbf{r}^{(k)}\right)^T \mathbf{r}^{(k)}}{\left(\mathbf{r}^{(k)}\right)^T A\mathbf{r}^{(k)}} \\ &\qquad \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{r}^{(k)} \\ &\qquad \mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha_k A\mathbf{r}^{(k)} \end{aligned} \tag{66}$$

Quando converge il metodo del gradiente?

Se $A \in \mathbb{R}^{n\times n}$ è simmetrica[10] e definita positiva[11] il metodo del gradiente converge alla soluzione del sistema lineare $A\mathbf{x} = \mathbf{b}$ per ogni dato iniziale $\mathbf{x}^{(0)} \in \mathbb{R}^n$ e inoltre vale la seguente **stima dell'errore**:

$$\left\|\mathbf{e}^{(k)}\right\|_A \leq \left(\frac{K(A)-1}{K(A)+1}\right)^k \left\|\mathbf{e}^{(0)}\right\|_A \qquad k \geq 0 \tag{67}$$

Dove $||\cdot||$ è la **norma dell'energia** definita a pagina 36 e $K(A)$ è il numero di condizionamento (spettrale) della matrice A definito a pagina 25.

Ma si può fare di meglio! Ed ecco che arriva il metodo del gradiente coniugato...

È possibile costruire delle direzioni di discesa "migliori" al fine di **giungere a convergenza in un numero di iterazioni inferiore rispetto a quelle del metodo del gradiente**. Partendo dunque dal metodo del gradiente e sfruttando la definizione ricorsiva dei vettori $\mathbf{x}^{(k)}$, si può ottenere:

$$\begin{aligned}
\mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)} \\
&= \mathbf{x}^{(k-1)} + \alpha_{k-1}\mathbf{d}^{(k-1)} + \alpha_k \mathbf{d}^{(k)} \\
&= \dots \\
&= \mathbf{x}^{(0)} + \sum_{j=0}^{k} \alpha_j \mathbf{d}^{(j)}
\end{aligned} \tag{68}$$

Si deduce che il vettore $\left(\mathbf{x}^{(k+1)} - \mathbf{x}^{(0)}\right) \in \mathbb{R}^n$ è una combinazione lineare delle direzioni di discesa $\left\{\mathbf{d}^{(j)}\right\}_{j=0}^{k}$.

Il **metodo del gradiente coniugato** costruisce un sistema di direzioni di discesa in $\mathbb{R}^n$ che siano tutte linearmente indipendenti fra di loro (quindi $\left\{\mathbf{d}^{(k)}\right\}_{k=0}^{n-1}$ sarà una base di $\mathbb{R}^n$) e tali che i valori $\{\alpha_k\}_{k=0}^{n-1}$ siano proprio i coefficienti dello sviluppo di $\left(\mathbf{x}^* - \mathbf{x}^{(0)}\right)$ rispetto alla base $\left\{\mathbf{d}^{(k)}\right\}_{k=0}^{n-1}$. Questo comporta che l'$n$-simo termine $\mathbf{x}^{(n)}$ della successione (eq. 68) coincide con la soluzione esatta $\mathbf{x}^*$.

Quando converge il metodo del gradiente coniugato?

Sia $A \in \mathbb{R}^{n\times n}$ una matrice simmetrica e definita positiva. Il metodo del gradiente coniugato per risolvere un sistema lineare **converge al più in n iterazioni** (in aritmetica esatta). Inoltre, il residuo $\mathbf{r}^{(k)}$ alla k-esima iterazione (con $k < n$) è ortogonale a $\mathbf{d}^{(j)}$ per $j = 0, \dots, k-1$ e:

$$\left\|\mathbf{e}^{(k)}\right\|_A \leq \frac{2c^k}{1+c^{2k}} \left\|\mathbf{e}^{(0)}\right\|_A \tag{69}$$

[10] Quindi corrisponde con la sua trasposta
[11] Definizione a pagina: 21

Con:

$$c = \frac{\sqrt{K(A)} - 1}{\sqrt{K(A)} + 1} \tag{70}$$

In aritmetica esatta il metodo del gradiente coniugato è pertanto un **metodo diretto in quanto termina dopo un numero finito di iterazioni**!

Il precondizionamento migliora il metodo del gradiente

Le iterazioni del metodo del gradiente, quando convergono alla soluzione esatta $\mathbf{x}^*$, sono caratterizzate da una **velocità di convergenza dipendente dal numero di condizionamento** della matrice A: **più grande è il numero di condizionamento di A e tanto più lenta sarà la convergenza**. Quindi, invece di risolvere il sistema lineare, si può risolvere il cosiddetto **sistema precondizionato** (equivalente):

$$P_L^{-1} A P_R^{-1} \widehat{\mathbf{x}} = P_L^{-1} \mathbf{b} \qquad \text{con } \widehat{\mathbf{x}} = P_R \mathbf{x} \tag{71}$$

Dove P_L e P_R sono opportune matrici non singolari tali che:

1. La matrice:
$$P = P_L P_R \tag{72}$$
è detta **precondizionatore**, è simmetrica e definita positiva;
2. $K\left(P^{-1}A\right) \ll K(A)$
3. La risoluzione di sistemi lineari della forma:
$$P_L \mathbf{s} = \mathbf{t} \qquad P_R \mathbf{v} = \mathbf{w} \tag{73}$$
è computazionalmente molto meno costosa della risoluzione del sistema originario $A\mathbf{x} = \mathbf{b}$.

Se si sceglie P_R uguale alla matrice identità, la matrice $P = P_L$ è detta **precondizionatore sinistro**, ed il sistema precondizionato assume la forma:

$$P^{-1} A \mathbf{x} = P^{-1} \mathbf{b} \tag{74}$$

Analogamente, se P_L è la matrice identità, la matrice $P = P_R$ è detta **precondizionatore destro**, ed il sistema precondizionato assume la forma:

$$A P^{-1} \widehat{\mathbf{x}} = \mathbf{b} \qquad \text{con } \widehat{\mathbf{x}} = P\mathbf{x} \tag{75}$$

Il **metodo del gradiente precondizionato** si ottiene applicando il metodo del gradiente al sistema precondizionato in cui si prenda $P_R = P_L^T$ e quindi $P_R^{-1} = P_L^{-T} \equiv \left(P_L^{-1}\right)^T$. Più precisamente, riscrivendo:

$$\underbrace{P_L^{-1} A P_L^{-T}}_{\widehat{A}} \underbrace{P_L^T \mathbf{x}}_{\widehat{\mathbf{x}}} = \underbrace{P_L^{-1} \mathbf{b}}_{\widehat{\mathbf{b}}} \tag{76}$$

2.3 Metodi numerici per sistemi non lineari

2.3.1 Introduzione

Molto spesso i problemi ingegneristici sono *non lineari*. Per risolvere tali problemi in modo ottimale, si adotta l'approssimazione numerica, la quale porta a dover risolvere un **sistema di equazioni non lineari**.

In generale, si ha il seguente problema:

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \longleftrightarrow \begin{cases} f_1(x_1, \ldots, x_j, \ldots, x_n) = 0 \\ \ldots \\ f_i(x_1, \ldots, x_j, \ldots, x_n) = 0 \\ \ldots f_n(x_1, \ldots, x_j, \ldots, x_n) = 0 \end{cases}$$

Dove:

- $\mathbf{x} \in \mathbb{R}^n$ è il vettore delle incognite x_j
- $\mathbf{f}$ è una funzione del tipo $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$
- f_i sono funzioni che esprimono relazioni non lineari fra le incognite

Questo caso è un **problema poiché presenta due principali difficoltà** dal punto di vista numerico:

1. Si è di fronte ad un **sistema**, per cui non si hanno equazioni singole.
2. Inoltre, il sistema è **non lineare**, e questo significa che deve essere linearizzato per poterlo risolvere.

2.3.2 Metodo di Newton

Il metodo di newton per sistemi lineari era stato introdotto nel paragrafo 1.3 (pagina 9). Può essere riscritto in modo equivalente per sistemi non lineari nel seguente modo.

Se la funzione $f : \mathbb{R}^n \to \mathbb{R}$ con $n \geq 1$, è di classe $C^2(\mathbb{R}^n)$, e se è possibile calcolare le sue derivate prime e seconde, allora è possibile applicare il metodo di Newton all'equazione vettoriale $\mathbf{f}(\mathbf{x}) = \nabla f(\mathbf{x}) = \mathbf{0}$ per calcolare il punto di minimo $\mathbf{x}^*$.

Dunque, il **metodo di Newton per sistemi non lineari** si formula nel seguente modo. Dato $\mathbf{x}^{(0)} \in \mathbb{R}^n$, per $k = 0, 1, \ldots$ e fino a convergenza, si deve:

$$\begin{aligned} &\text{risolvere} \quad \mathrm{H}\left(\mathbf{x}^{(k)}\right) \delta\mathbf{x}^{(k)} = -\nabla\mathbf{f}\left(\mathbf{x}^{(k)}\right) \\ &\text{ponendo} \quad \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \delta\mathbf{x}^{(k)} \end{aligned} \tag{77}$$

È interessante notare che la matrice Hessiana utilizzata ne metodo di Newton è uguale alla matrice Jacobiana $J_{\mathbf{f}}\left(\mathbf{x}^{(k)}\right)$ valutata nel punto $\mathbf{x}^{(k)}$.

Dato un punto $z \in \mathbb{R}^n$, si introduce la matrice Jacobiana relativa a $\mathbf{f}$:

$$J(\mathbf{z}) = \nabla\mathbf{f}(\mathbf{z}) \tag{78}$$

Di componenti:

$$j_{il} = \frac{\partial f_i(\mathbf{z})}{\partial x_l} \qquad i, l = 1, \ldots, n \tag{79}$$

Per ogni $\mathbf{x} \in \mathbb{R}^n$ si ha quindi $J(\mathbf{z}) \in \mathbb{R}^{n \times n}$. Avendo definito la matrice Jacobiana non singolare (determinante diverso da zero), si può riscrivere il metodo di Newton in modo alternativo nel seguente modo:

$$\begin{aligned} &\text{risolvere} \quad \mathrm{J}\left(\mathbf{x}^{(k)}\right) \delta\mathbf{x}^{(k)} = -\mathbf{f}\left(\mathbf{x}^{(k)}\right) \\ &\text{ponendo} \quad \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \delta\mathbf{x}^{(k)} \end{aligned} \tag{80}$$

Algoritmo

Ad ogni iterazione k, il primo passo del metodo di Newton consiste nella risoluzione di un sistema lineare di dimensione n:

1. Risolvere il sistema lineare $n \times n$:

$$\mathrm{J}\left(\mathbf{x}^{(k)}\right) \delta\mathbf{x}^{(k)} = -\mathbf{f}\left(\mathbf{x}^{(k)}\right)$$

   Infatti $\mathrm{J}\left(\mathbf{x}^{(k)}\right)$ è una matrice non singolare di coefficienti noti:

$$\frac{\partial f_i\left(\mathbf{x}^{(k)}\right)}{\partial x_l}$$

   E $-\mathbf{f}\left(\mathbf{x}^{(k)}\right)$ è il termine noto.

2. Una volta risolto il sistema lineare e ottenuto $\delta\mathbf{x}^{(k)}$, si aggiorna la variabile $\mathbf{x}^{(k+1)}$ mediante il secondo passo del metodo:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \delta\mathbf{x}^{(k)}$$

Come detto, questo procedimento avviene ad ogni iterazione k!

Il metodo di Newton, può essere considerato un:

- **Metodo locale** se:
  - Esiste un $\delta > 0$
  - È vera la condizione:
  $$\left|\left|\alpha - \mathbf{x}^{(0)}\right|\right| < \delta$$
  Per cui:
  $$\lim_{k\to\infty} \left|\left|\alpha - \mathbf{x}^{(k)}\right|\right| = 0$$
- **Metodo del secondo ordine** se:
  - Newton converge
  - La matrice Jacobiana J è derivabile

  Per cui:
  $$\frac{\left|\left|\alpha - \mathbf{x}^{(k+1)}\right|\right|}{\left|\left|\alpha - \mathbf{x}^{(k)}\right|\right|^2} \leq C$$

Si ricorda che α sono le radici di $\mathbf{f}$.

Criteri d'arresto

Dato che il metodo di Newton è un metodo iterativo, è necessario introdurre opportuni criteri di arresto. Quindi:

- Criterio sul **residuo**:
$$\left|\left|\mathbf{f}\left(\mathbf{x}^{(k)}\right)\right|\right| < \varepsilon \tag{81}$$
- Criterio sull'**incremento**:
$$\left|\left|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right|\right| < \varepsilon \tag{82}$$

Scegliendo un'opportuna tolleranza ε.

Costi computazionali e varianti ottimizzate

Il metodo di Newton per sistemi non lineari, richiede come **costo computazionale**:

$$\texttt{CPU TIME} = \#\text{ iterazioni} \times (C_{cos} + C_{sl}) \tag{83}$$

Dove:

- C_{cos} è il **tempo** necessario per **costruire la matrice Jacobiana** $J\left(\mathbf{x}^{(k)}\right)$.
- C_{sl} è il **tempo** di **risoluzione** del corrispondente **sistema lineare**.

Si possono fare alcune osservazioni riguardo questo calcolo:

- La matrice Jacobiana $J\left(\mathbf{x}^{(k)}\right)$ deve essere ricostruita ad ogni iterazione.
- Generalmente, il numero di iterazioni (# iterazioni) è molto basso poiché il metodo di Newton è di ordine 2.

In ogni caso, a causa di un tempo di processo alto nei casi di sistemi lineari con ordini maggiori, esistono due principali alternative:

1. **Aggiornamento Jacobiana ogni p iterazioni**. Dato un numero intero $p \geq 2$, l'idea è quella di aggiornare la matrice Jacobiana $J\left(\mathbf{x}^{(k)}\right)$ solo ogni p volte.

   Così facendo, il tempo di costruzione della matrice Jacobiana C_{cos} viene ridotto, dato che si riduce a p iterazioni.

   In questo metodo, si definisce $C_{\text{cos}-\text{Newton}}$ il tempo di costruzione della matrice Jacobiana che si avrebbe applicando il semplice metodo di Newton (e non tale variante!). E la variabile C_{cos} nella formula 83 diventa:

   $$C_{\text{cos}} = \frac{C_{\text{cos}-\text{Newton}}}{p} \tag{84}$$

   Ovviamente p sono il numero di iterazioni.

   Infine, utilizzando la fattorizzazione LU per la risoluzione del sistema lineare, il costo della fattorizzazione della matrice Jacobiana $J\left(\mathbf{x}^{(k)}\right)$ è suddiviso su p iterazioni. Di fatto, ad ogni iterazione mediamente si ha un costo di:

   $$\frac{2n^3}{3p}$$

   Tuttavia, così facendo, il numero di iterazioni è nettamente maggiore a quello del secondo ordine. Un numero di iterazioni troppo elevato rischia di perdere la convergenza. Per questo motivo, si dovrebbe rispettare la seguente relazione:

   $$\#\text{iterazioni} \times \frac{(C_{\text{cos}} + C_{\text{sl}})}{p} < \#\text{iterazioni} \times (C_{\text{cos}} + C_{\text{sl}}) \tag{85}$$

2. **Inexact Newton**. L'idea è quella di sostituire ad ogni iterazione k la matrice Jacobiana $J\left(\mathbf{x}^{k}\right)$ con una sua approssimazione $\tilde{J}^k$ che sia in generale facilmente costruibile e tale che il sistema lineare associato sia facilmente risolvibile.

   Tale tecnica è un **metodo esatto**, quindi raggiunta la convergenza la soluzione è α (la parola *inexact* si riferisce alla matrice Jacobiana!).

3 Approssimazione di funzioni e di dati

3.1 Interpolazione

In molte applicazioni concrete si conosce una funzione solo attraverso i suoi valori in determinati punti. Si supponga di conoscere $n+1$ coppie di valori $\{x_i, y_i\}$ con $i = 0, \ldots, n$, dove i punti x_i, tutti distinti, vengono chiamati **nodi**.

In tal caso, può apparire naturale richiedere che la funzione approssimante $\tilde{f}$ soddisfi le seguenti uguaglianze:

$$\tilde{f}(x_i) = y_i \qquad i = 0, 1, \ldots, n \tag{86}$$

Una tale funzione $\tilde{f}$ viene chiamata **interpolatore** dell'insieme di dati $\{y_i\}$ e le equazioni del tipo 86 sono le **condizioni di interpolazione**.

Esistono vari tipi di interpolatori:

- L'**interpolatore polinomiale**:

$$\tilde{f}(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

- L'**interpolatore trigonometrico**:

$$\tilde{f}(x) = a_{-M} e^{-iMx} + \cdots + a_0 + \cdots + a_M e^{iMx}$$

  Dove M è un intero pari a $\frac{n}{2}$ se n è pari, $\frac{(n+1)}{2}$ se n è dispari, e i è l'unità immaginaria.

- L'**interpolatore razionale**:

$$\tilde{f}(x) = \frac{a_0 + a_1 x + \cdots + a_k x^k}{a_{k+1} + a_{k+2} x + \cdots + a_{k+n+1} x^n}$$

3.2 Interpolazione Lagrangiana

L'interpolazione Lagrangiana è un'interpolazione di tipo polinomiale.

Definizione 1: interpolazione Lagrangiana

Per ogni insieme di coppie $\{x_i, y_i\}$, $i = 0, \dots, n$, con i nodi x_i distinti fra loro, esiste un unico polinomio di grado minore od uguale a n, che viene indicato con $\prod_n$ e viene chiamato **polinomio interpolatore** dei valori y_i nei nodi x_i, tale che:

$$\prod_n (x_i) = y_i \qquad i = 0, \dots, n \tag{87}$$

Quando i valori $\{y_i, i = 0, \dots, n\}$, rappresentano i valori assunti da una funzione continua f (ovvero $y_i = f(x_i)$), $\prod_n$ è detto **polinomio interpolatore** di f (in breve, interpolatore di f) e viene indicato con $\prod_n f$.

Quindi un *interpolatore* è una **funzione che assume il valore dei dati in corrispondenza dei nodi** x_i.

Come ottenere il polinomio interpolatore?

Per ogni k compreso tra 0 e n si costruisce un polinomio di grado n, denotato $\varphi_k(x)$, il quale interpola i valori y_i tutti nulli fuorché quello per $i = k$ per il quale $y_k = 1$, ovvero:

$$\varphi_k \in \mathbb{P}_n \qquad \varphi_k(x_j) = \delta_{jk} = \begin{cases} 1 & \text{se } j = k \\ 0 & \text{altrimenti} \end{cases}$$

Dove δ_{jk} è il simbolo di Kronecker. Si può dunque definire la formula dei **polinomi caratteristici di Lagrange** $\varphi_k(x)$:

$$\varphi_k(x) = \prod_{j=0, j\neq k}^{n} \frac{x - x_j}{x_k - x_j} \qquad k = 0, \dots, n \tag{88}$$

Che non è altro che:

$$\varphi_k(x) = \frac{(x - x_0)(x - x_1) \cdots (x - x_{k-1})(x - x_{k+1}) \cdots (x - x_n)}{(x_k - x_0)(x_k - x_1) \cdots (x_k - x_{k-1})(x_k - x_{k+1}) \cdots (x - x_n)}$$

Essi sono dati dal prodotto di n termini di primo grado, perciò sono dei **polinomi di grado** n.

Esempio 1: polinomi caratteristici di Lagrange

Si prenda:

- $n = 2$
- $x_0 = -1$
- $x_1 = 0$
- $x_2 = 1$

I 3 polinomi caratteristici di Lagrange sono dati da:

$$\begin{aligned}\varphi_0(x) &= \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} = \frac{1}{2}x(x-1)\\ \varphi_1(x) &= \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} = -(x+1)(x-1)\\ \varphi_2(x) &= \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} = \frac{1}{2}x(x+1)\end{aligned}$$

Come atteso, si nota che i 3 polinomi caratteristici di Lagrange sono dei polinomi di grado 2 e soddisfano la proprietà $\varphi_k(x_i) = \delta_{ik}$

Quanto detto finora, può essere generalizzato nel seguente modo:

$$\prod_n(x) = \sum_{j=0}^{n} y_j \varphi_j(x) \tag{89}$$

Proprietà interpolatore Lagrangiano

Le proprietà sono:

1. $\prod_n(x)$ è un **interpolatore**. Difatti, valutandolo per un generico nodo x_i, si ottiene:
$$\prod_n(x_i) = \sum_{j=0}^{n} y_j \varphi_j(x_i) = \sum_{j=0}^{n} y_j \delta_{ij} = y_i$$

2. $\prod_n(x)$ è un **polinomio di grado** n. Infatti, esso è dato dalla somma dei polinomi di grado n $y_j\varphi_j(x)$.

3. $\prod_n(x)$ è l'**unico polinomio di grado** m **interpolante gli** $n+1$ **dati** (x_i, y_i), $i = 0, \dots, n$.

   Si supponga esista un altro interpolatore polinomiale di grado n $\psi_n(x)$ che interpola i dati (x_i, y_i), $i = 0, \dots, n$. Si introduce il seguente polinomio di grado n:
   $$D(x) = \prod_n(x) - \psi_n(x)$$

   Allora vale il seguente risultato:
   $$D(x_i) = \prod_n(x_i) - \psi_n(x_i) = 0 \qquad \forall i = 0, \dots, n$$

   Di conseguenza, $D(x)$ ha $n+1$ zeri. Essendo un polinomio di grado n, si deve avere $D(x) \equiv 0$, da cui segue l'unicità.

3.2.1 Accuratezza (errore) nel caso di approssimazione di funzioni

Si consideri il caso di approssimazione dei valori di una funzione $f(x)$. Si ha dunque il seguente risultato.

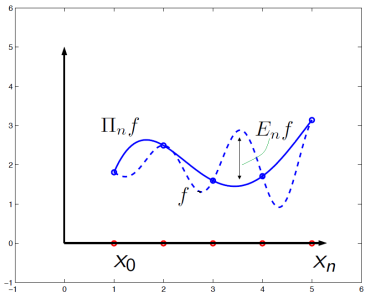
Definizione 2: quantificazione dell'errore

Sia I un intervallo limitato, e si considerino $n+1$ nodi di interpolazione distinti $\{x_i, i = 0, \ldots, n\}$ in I. Sia f derivabile con continuità fino all'ordine $n+1$ in I. Allora $\forall x \in I$, $\exists \xi_x \in I$ tale che:

$$E_n f(x) = f(x) - \prod_n f(x) = \frac{f^{(n+1)}(\xi_x)}{(n+1)!} \prod_{i=0}^{n} (x - x_i) \tag{90}$$

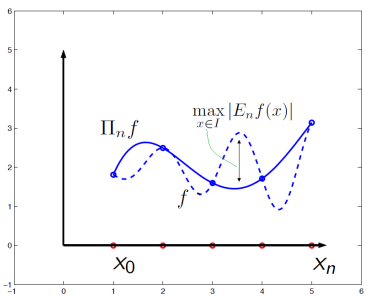
In altre parole, la definizione rappresenta come si può **quantificare l'errore** che si commette sostituendo ad una funzione f il suo polinomio interpolatore $\prod_n f$ (in questo caso Lagrangiano).

Nel seguente grafico si può notare che la funzione $E_n f(x)$ si annulla in corrispondenza dei nodi x_i. Inoltre, è in generale diverso da zero lontano dai nodi, ovvero con $x \neq x_i$.



Nel caso di nodi equispaziati, vale la seguente **stima dell'errore massimo dell'interpolatore Lagrangiano**:

$$\max_{x \in I} |E_n f(x)| \leq \frac{\left| \max_{x \in I} f^{(n+1)(x)} \right|}{4(n+1)} \cdot h^{n+1} \tag{91}$$



3.2.2 Convergenza dell'interpolatore Lagrangiano

All'aumento delle informazioni a disposizione $(n+1)$, idealmente si vorrebbe un **miglioramento dell'accuratezza di una funzione approssimante**. In particolare, si vorrebbe il seguente risultato:

$$\lim_{n\to\infty} \max_{x\in I} |E_n f(x)| = 0$$

Dall'equazione 91, a pagina 48, si può notare che il denominatore della frazione e il termine h^{n+1} tendono a zero quando n tende a infinito. Al contrario, il numeratore non è certo se sia limitato per n che tende a infinito.

Difatti, possono esistere casi in cui si ha:

$$\lim_{n\to\infty} \left| \max_{x\in I} f^{(n+1)}(x) \right| = \infty$$

Che portano a:

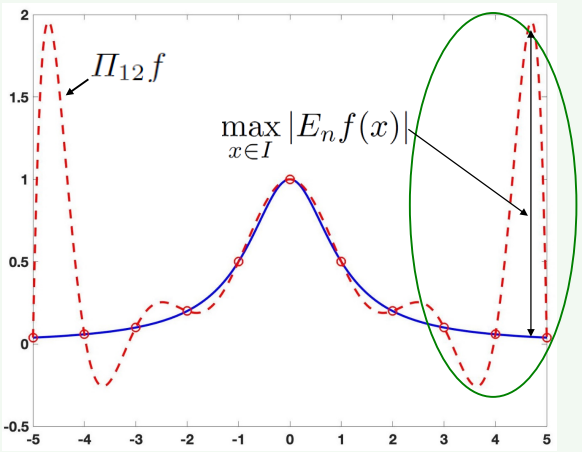
$$\lim_{n\to\infty} \max_{x\in I} |E_n f(x)| = \infty$$

Ovvero alla **non convergenza dell'interpolatore Lagrangiano**.

Esempio 2: fenomeno di Runge

Il **fenomeno di Runge** è un evento che si scatena quando **l'interpolatore Lagrangiano non raggiunge la convergenza**.
In particolare, in presenza di questo fenomeno, la funzione errore E_n presenta delle oscillazioni ai nodi estremi che crescono con il crescere di n.
Un esempio di interpolatore $\prod_{12} f$ (in linea continua) calcolato su 13 nodi equispaziati nel caso della funzione di Runge (in linea tratteggiata):

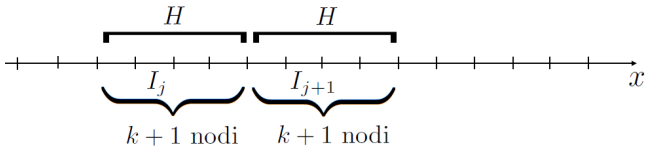
$$f(x) = \frac{1}{(1+x^2)}$$



3.3 Interpolazione Lagrangiana composita

Una miglioria che è possibile fare all'interpolazione Lagrangiana, è quella di introdurre un **interpolatore continuo dato dall'unione di tanti interpolatori Lagrangiani di basso ordine**, ovvero $k \ll n$.

Tali singoli interpolatori locali sono costruiti sugli intervalli disgiunti I_j ognuno composto da $k+1$ nodi e di lunghezza $H = kh$:



Tale interpolatore globale, chiamato **interpolazione Lagrangiana composita**, viene indicato con $\prod_k^H(x)$. Se si vuole enfatizzare che l'interpolatore è stato costruito per approssimare una certa funzione f, allora si utilizza la notazione $\prod_k^H f(x)$.

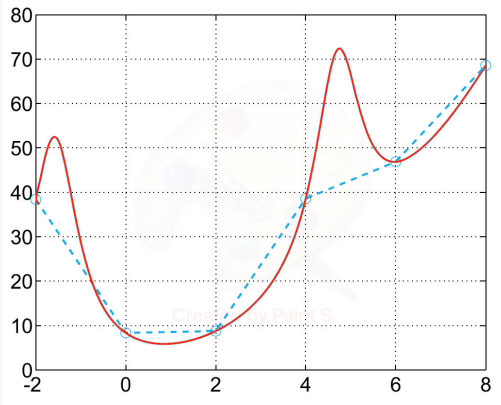
Esempio 3: interpolatore Lagrangiano composito lineare

Si consideri il caso $k = 1$. La funzione rappresentata in linea continua è:

$$f(x) = x^2 + \frac{10}{(\sin(x) + 1.2)}$$

Ed il suo interpolatore lineare composito rappresentato con la linea tratteggiata:

$$\prod_1^H f(x)$$



Vista la sua semplicità, questo interpolatore è molto utilizzato nelle applicazioni.

3.3.1 Accuratezza (errore) e convergenza dell'interpolatore Lagrangiano composito

A differenza dell'interpolazione Lagrangiana, all'aumentare del numero di informazioni $n+1$ che si hanno a disposizione, l'accuratezza dell'interpolatore Lagrangiano composito subisce un *miglioramento*.

All'aumentare del valore di n, verranno aumentati anche il numero di intervalli I_j su cui costruire gli interpolatori Lagrangiani locali, **senza variare il gradi locale k che rimarrà costante**. Questa tecnica riesce ad evitare il fenomeno di Runge, ovverosia che l'interpolatore Lagrangiano non raggiunge la convergenza.

Riguardo alla convergenza, si introduce l'**errore puntuale**:

$$E_k^H f(x) = f(x) - \prod_k^H f(x) \tag{92}$$

Il quale è dato dal massimo degli errori degli interpolatori Lagrangiani di grado k su ogni I_j. In caso di nodi equispaziati, si ottiene che la **stima dell'errore dell'interpolatore Lagrangiano composito** tende a zero nel caso in cui H tende a zero:

$$\begin{aligned}
\max_{x\in I}\left|E_k^H f(x)\right| &\leq \max_j \frac{\max_{x\in I_j}\left|f^{(k+1)}(x)\right|}{4(k+1)}\cdot h^{(k+1)} \\
&\leq \frac{\max_{x\in I}\left|f^{(k+1)}(x)\right|}{4(k+1)}\cdot h^{(k+1)} \\
&\quad \text{sostituendo } h \text{ con } \frac{H}{k} \\
&\leq \underbrace{\frac{\max_{x\in I}\left|f^{(k+1)}(x)\right|}{4(k+1)\,k^{(k+1)}}}_{\text{Indipendente da } H}\cdot H^{(k+1)}
\end{aligned} \tag{93}$$

Oltre ad un errore, questo dimostra la **convergenza dell'interpolatore Lagrangiano composito**.

Se H e k sono importanti, come devono essere scelti?

Dipende dall'origine dei dati:

- Se i dati provengono da una funzione $f(x)$ nota sull'intervallo $[a, b]$ che genera $y_i = f(x_i)$ con $i = 0, \ldots, n$, allora:
  - Si **decide il valore di** k da utilizzare cercando di non superare il valore 3 per non incorrere nel fenomeno di Runge (non convergenza!);
  - Si **sceglie il valore dell'ampiezza degli intervalli** H in modo da avere l'errore desiderato in base alla stima dell'errore riportato nell'equazione 93;
  - Si **partiziona l'intervallo** $[a, b]$ in intervallo di ampiezza H e su **ognuno di essi si considerano** $k + 1$ **nodi**.
- Se i dati provengono da misure, il numero di questi $n + 1$ è fissato. Per cui le operazioni da fare possono essere una delle seguenti:
  - Un'**interpolazione composita lineare**:

$$k = 1 \qquad H = \frac{(b - a)}{n}$$

  - Un'**interpolazione composita quadratica**:

$$k = 2 \qquad H = \frac{2\,(b - a)}{n}$$

3.4 Interpolazione sui nodi di Chebyshev

L'interpolazione Lagrangiana composita consente di evitare il fenomeno di Runge. Tuttavia, l'interpolazione Lagrangiana può essere modificata **posizionando i nodi in precise posizioni che garantiscono la stabilità al crescere di** n. Negli interpolatori presentati, tutti si basavano su nodi equispaziati. Nel seguente capitolo si mostrano i nodi di Chebyshev, ovvero un'ubicazione specifica dei nodi nello spazio.

Solo il fenomeno di Runge può essere evitato?

La risposta chiaramente è no. L'applicazione di questa tecnica consente:

- Di **utilizzare** *sempre* con successo il **polinomio interpolatore Lagrangiano** di grado n.
- Di evitare la non convergenza, ovvero il fenomeno di Runge.

Nodi di Chebyshev

Si consideri il caso in cui il dominio sia $[-1, 1]$ e $n+1$ dati ubicati in corrispondenza delle seguenti ascisse:

$$\widehat{x}_i = -\cos\left(\frac{\pi i}{n}\right) \qquad i = 0, \ldots, n \tag{94}$$

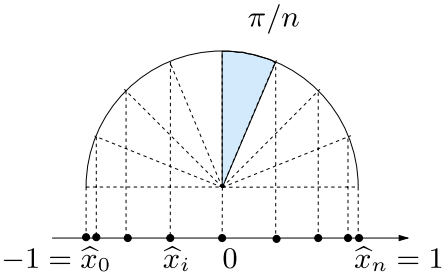


Figura 3: Distribuzione dei nodi di Chebyshev nell'intervallo $[-1, 1]$.

Questi **nodi di Chebyshev** sono **ottenuti come proiezioni sull'asse** x **di punti sulla circonferenza unitaria individuati da settori circolari ottenuti con lo stesso angolo** $\frac{\pi}{n}$.

E se viene applicato l'interpolatore Lagrangiano sui nodi di Chebyshev?

Come detto all'inizio, si può modificare direttamente l'interpolatore Lagrangiano. Quindi, si consideri l'interpolatore Lagrangiano di grado n costruito sui nodi di Chebyshev. Tale interpolatore si ottiene applicando le equazioni presentate

precedentemente, con l'unico vincolo di **prendere in considerazione $\widehat{x}_i$ come nodi su cui costruire gli n polinomi caratteristici di Lagrange**:

$$\begin{aligned} \widehat{\varphi}_k(x) &= \prod_{j=0, j\neq k}^{n} \frac{x-\widehat{x}_i}{\widehat{x}_k-\widehat{x}_i} \\ \prod_n^C(x) &= \sum_{j=0}^{n} y_j \widehat{\varphi}_j(x) \end{aligned} \tag{95}$$

La variabile $\prod_n^C(x)$ rappresenta l'**interpolatore Lagrangiano sui nodi di Chebyshev**. Si indica con la rappresentazione $\prod_n^C f(x)$ quando viene applicato a dati ottenuti dalla valutazione di una funzione f, ovvero $y_i = f(x_i)$.

3.4.1 Convergenza dell'interpolazione sui nodi di Chebyshev

Teorema 2. *Si supponga che la funzione $f(x)$ ammetta derivata continua fino all'ordine $s+1$ compreso, ovverosia $f \in C^{(s+1)}([-1,1])$.*

*Allora, si ha il seguente risultato di **convergenza per l'interpolazione sui nodi di Chebyshev**:*

$$\max_{x\in[-1,1]} \left| f(x) - \prod_n^C f(x) \right| \leq \tilde{C}\frac{1}{n^s} \tag{96}$$

Per un'opportuna costante C.

Dal precedente teorema, si possono fare 3 osservazioni:

1. Se $s \geq 1$ allora si è sicuri che ci sarà **sempre convergenza**:

$$\lim_{n\to\infty} \max_{x\in[-1,1]} \left| f(x) - \prod_n^C f(x) \right| = 0 \tag{97}$$

2. La **velocità di convergenza aumenta all'aumentare di** s.

3. Se il teorema è valido $\forall s > 0$, allora la **velocità di convergenza è *esponenziale***. Questa è la migliore stima che è possibile ottenere, dato che è più veloce di qualsiasi altro $\frac{1}{n^s}$:

$$\max_{x\in[-1,1]} \left| f - \prod_n^C f(x) \right| \leq \tilde{C}e^{-n} \tag{98}$$

3.4.2 Generalizzazione dell'intervallo

Nel caso in cui il dominio di interesse non sia più da $[-1, 1]$ (come viene utilizzato nelle precedenti equazioni), ma un intervallo generale $[a, b]$, la "conversione" può avvenire mappando i nodi $\widehat{x}_j$ da $[-1, 1]$ a $[a, b]$:

$$x = \psi(\widehat{x}) = \frac{a+b}{2} + \frac{b-a}{2}\,\widehat{x} \tag{99}$$

Applicando la **generalizzazione dell'intervallo ai nodi di Chebyshev**, si ottiene:

$$x_j^C = \psi(\widehat{x}_j) = \frac{a+b}{2} + \frac{b-a}{2}\,\widehat{x}_j \qquad j = 0, \ldots, n \tag{100}$$

Di conseguenza, il **polinomio di Lagrange applicato sui nodi di Chebyshev in un intervallo generale** diventa:

$$\varphi_k^C(x) = \prod_{j=0, j\neq k}^{n} \frac{x - x_j^C}{x_k^C - x_j^C} \tag{101}$$

3.5 Interpolazione trigonometrica

Nel caso di funzioni periodiche, ad esempio con periodo 2π e con $f(0) = f(2\pi)$, si ha il valore delle ascisse pari a:

$$x_j = \frac{2\pi j}{(n+1)} \qquad j = 0, \ldots, n$$

In questo caso, l'interpolatore Lagrangiano non approssima in modo ottimale la funzione f lontana dai nodi, dato che è un polinomio di grado n (per cui in generale non periodico!).

Supponendo n pari e $M = \frac{n}{2}$, si può costruire l'**interpolatore trigonometrico** $\tilde{f}(x)$ come la **combinazione lineare di seni e coseni**:

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{M} \left[a_k \cos(kx) + b_k \sin(kx)\right] \tag{102}$$

Per opportuni coefficienti complessi incogniti a_k, per $k = 0, \ldots, M$ e b_k per $k = 1, \ldots, M$. Come si può vedere, la funzione periodica $\tilde{f}(x)$ è di periodo 2π caratterizzata da $M+1$ **frequenze** e **coefficienti** a_k e b_k con $k = 0, \ldots, M$.

Sfruttando la nota formula di **Eulero**:

$$e^{ikx} = \cos(kx) + i \sin(kx)$$

L'**interpolatore trigonometrico** può essere riscritto in modo più compatto:

$$\tilde{f}(x) = \sum_{k=-M}^{M} c_k e^{ikx} \tag{103}$$

Inoltre, dall'equazione 102 i valori a_k, b_k con $k = 0, \ldots, M$ sono:

$$\begin{cases} a_k = c_k + c_{-k} \\ b_k = i(c_k - c_{-k}) \end{cases}$$

E nell'interpolazione trigonometrica con Eulero 103 si ha:

$$\begin{cases} c_k = \frac{1}{2}(a_k - ib_k) \\ c_{-k} = \frac{1}{2}(a_k + ib_k) \end{cases}$$

Quindi, le funzioni interpolatrici trigonometriche $\tilde{f}(x)$ sono $2M + 1 = n + 1$ equazioni nelle $n + 1$ incognite c_k.

La **distanza** (costante) fra due nodi $h = \frac{2\pi}{(n+1)}$ come:

$$x_j = jh \qquad j = 0, \ldots, n$$

Si impone inoltre il **vincolo di interpolazione**:

$$\tilde{f}(x_j) = \sum_{k=-M}^{M} c_k e^{ikjh} = f(x_j) \qquad j = 0, \ldots, n \tag{104}$$

A differenza dell'interpolazione Lagrangiana, i coefficienti della combinazione lineare c_k con $k = -M, \ldots, M$, sono incogniti. Si hanno di conseguenza $n + 1$ equazioni nelle $2M + 1 = n + 1$ incognite c_k. Tale equazione è una base di partenza per le formule nei paragrafi successivi.

3.5.1 Trasformata Discreta di Fourier

Dato un intero $m \in [-M, M]$, si moltiplica all'equazione di base 104, a pagina 56, a sinistra e a destra per $e^{-imx_j} \Rightarrow e^{imjh}$ e si sommano su j:

$$\sum_{j=0}^{n} \sum_{k=-M}^{M} c_k e^{ikjh} e^{-imjh} = \sum_{j=0}^{n} f(x_j) e^{-imjh} \tag{105}$$

In cui si può esporre una relazione esplicita fra i coefficienti e i valori noti della funzione $f(x_j)$:

$$c_m = \frac{1}{n+1} \sum_{j=0}^{n} f(x_j) e^{-imjh} \qquad m = -M, \ldots, M \tag{106}$$

La funzione $f(x_j)$ viene chiamata **Trasformata discreta di Fourier (DFT)**. **Queste sono $n+1$ equazioni nelle incognite** $f(x_j)$.

Parlando di DTF, è necessario introdurre anche lo **spazio fisico** e delle **frequenze**. Dati i vettori $\mathbf{c} \in \mathbb{R}^{n+1}$ di componenti c_k e $\mathbf{f} \in \mathbb{R}^{n+1}$ di componenti $f(x_j)$, è possibile riscrivere la trasformata di Fourier nello **spazio delle frequenze** k come:

$$c_k = \frac{1}{n+1} \sum_{j=0}^{n} f(x_j) e^{-ikjh} \qquad k = -M, \ldots, M$$

E in forma matriciale nel seguente modo:

$$\mathbf{c} = T\mathbf{f} \tag{107}$$

Con la matrice T composta da elementi:

$$T_{kj} = \frac{1}{n+1} e^{-ikjh}$$

Nel caso in cui si vuole passare dallo **spazio delle frequenze** $f(x_j)$ allo **spazio fisico** c_k, si può usare il vincolo di interpolazione (eq. 104, pag. 56):

$$\tilde{f}(x_j) = \sum_{k=-M}^{M} c_k e^{ikjh} \qquad j = 0, \ldots, n$$

E in forma matriciale nel seguente modo:

$$\mathbf{f} = T^{-1}\mathbf{c} \tag{108}$$

Con la matrice T^{-1} composta da elementi:

$$\left(T^{-1}\right)_{kj} = e^{ikjh}$$

La **trasformazione dallo spazio delle frequenze allo spazio fisico** è chiamata **Trasformata Discreta di Fourier inversa (Inverse Discrete Fourier Transform, IDFT)**.

3.5.2 Fast Fourier Transform (FFT)

Il calcolo dei coefficienti c_k può essere fatto ancora più rapidamente utilizzato la **Trasformata rapida di Fourier (Fast Fourier Transform, FFT)**.

Si consideri una generica matrice quadrata A di dimensione $(n+1) \times (n+1)$ e un vettore $\mathbf{x}$ di dimensione $n+1$. È possibile calcolare la componente w_j del vettore $\mathbf{w} = A\mathbf{x}$ come:

$$w_j = \sum_{k=1}^{n} A_{jk} x_k \tag{109}$$

Nella FFT, la matrice T ha una struttura particolare chiamata di Toeplitz. Questo comporta che l'elemento w_j del vettore $\mathbf{w} = T\mathbf{x}$ è determinato solo dagli elementi della matrice T, ovverosia T_{km} (con $j = km$). Una struttura tipica della matrice è:

$$T = \frac{1}{n+1} \begin{bmatrix} e^{-ih} & e^{-2ih} & e^{-3ih} & \dots & \dots & e^{-nih} \\ e^{-2ih} & e^{-4ih} & e^{-6ih} & \dots & \dots & e^{-2nih} \\ e^{-3ih} & e^{-6ih} & e^{-9ih} & \dots & \dots & e^{-3nih} \\ \vdots & & \ddots & & & \vdots \\ \vdots & & \ddots & & & \vdots \\ e^{-nih} & e^{-2nih} & e^{-3nih} & \dots & \dots & e^{-n^2ih} \end{bmatrix}$$

Il **costo computazione** del vettore $\mathbf{w}$, a causa della forma particolare della matrice di Toeplitz, è uguale a $n \log_2 n$ operazioni.

3.5.3 Espressione Lagrangiana dell'interpolatore trigonometrico

L'interpolatore trigonometrico può essere scritto in forma Lagrangiana, ovvero con coefficienti nella combinazione lineare dati dal risultato dei vari $f(x_j)$, nel seguente modo:

$$\tilde{f}(x) = \sum_{j=0}^{n} f(x_j)\, \varphi_j^T(x) \tag{110}$$

Dove il polinomio φ non è altro che:

$$\varphi_j^T(x_k) = \delta_{jk} \tag{111}$$

E in questo caso la convergenza?

La convergenza è possibile esprimerla tramite il seguente teorema.

Teorema 3. *Si supponga che la funzione $f(x)$ ammetta derivata continua e periodica di periodo 2π fino all'ordine $s+1$ compreso.*

*Allora, si ha il seguente risultato di **convergenza per l'interpolatore trigonometrico in forma Lagrangiana**:*

$$\max_{x \in [0,2\pi]} \left| f(x) - \tilde{f}(x) \right| \leq C \frac{1}{n^s} \tag{112}$$

Per un'opportuna costante C.

Dal precedente teorema, se ne deriva che la **convergenza è esponenziale qualora il precedente risultato valga per** $\forall s > 0$.

3.5.4 Fenomeno dell'aliasing e teorema di Shannon

Se il numero di dati $n+1$ non è sufficientemente elevato, l'**interpolatore trigonometrico non è in grado di descrivere le frequenze k più alte**. Questo problema viene chiamato **fenomeno di aliasing**.

Nel mondo reale, l'occhio umano campiona con una frequenza massima le informazioni luminose che provengono dal mondo esterno. Ma se tale campionamento non è sufficientemente frequente da catturare la frequenza massima del fenomeno esterno, allora l'immagine riprodotta dal cervello (che di fatto agisce in questo caso come un interpolatore trigonometrico) sarà caratterizzata da frequenze diverse (aliasing). E questo è anche il motivo per negli elicotteri si vedono le pale girare molto lentamente o addirittura in senso opposto.

In particolare, sia $k_{\max}$ la **frequenza massima** della funzione $f(x)$. Se $n \leq 2k_{\max}$, la **frequenza massima dell'interpolatore trigonometrico** $\tilde{f}(x)$ è minore di $k_{\max}$ (aliasing).

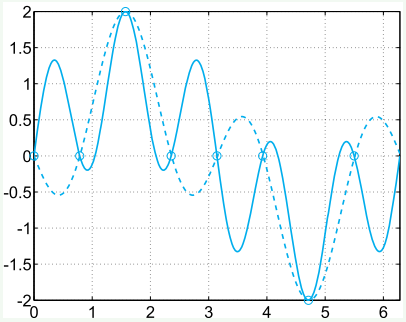
Esempio 4: aliasing

Gli effetti dell'aliasing si possono vedere confrontando, per esempio, le seguenti due funzioni:

- In linea continua:

$$f(x) = \sin(x) + \sin(5x)$$

- Linea tratteggiata, l'interpolatore trigonometrico $\tilde{f}(x)$ con $M = 3$



Quindi, il **teorema di Shannon** afferma che n deve essere:

$$n > 2k_{\max} \tag{113}$$

3.6 Il metodo dei minimi quadrati

Perché è necessario un altro metodo?

Al crescere del grado del polinomio dell'interpolazione di Lagrange, esso non garantisce una maggiore accuratezza nell'approssimazione di una funzione. Per aumentare l'accuratezza, è stato introdotto l'interpolazione Lagrangiana composita. Purtroppo, quest'ultima non è ottima per estrapolare informazioni da dati noti, ovvero **per generare nuove valutazioni in punti che giacciono al di fuori dell'intervallo di interpolazione**.

Quindi, i precedenti metodi esposti non sono ottimi:

- **Interpolazione Lagrangiana**: potrebbe soffrire del fenomeno di Runge, a causa del numero elevato di dati.
- **Interpolazione Lagrangiana composita**: terrebbe in conto solo gli ultimi $k+1$ dati, per cui non utilizzerebbe tutta la storia a disposizione e non darebbe un'espressione analiticamente semplice.
- **Interpolazione su nodi Chebyshev**: nella realtà è difficile da applicare poiché i dati a disposizione sono equispaziati.

Ragionamento per arrivare alla definizione del metodo

Si supponga di avere a disposizione un insieme di dati e ciascun elemento è formato da una coppia:

$$\{(x_i, y_i)\,,\ i = 0, \dots, n\}$$

In cui gli y_i potrebbero essere i valori che una funzione assume nei nodi x_i: $q(x_i) = y_i$.

L'obbiettivo è cercare un *polinomio globale* di grado m (con $m \geq 1$) molto più basso rispetto a n ($m \ll n$). Tale polinomio ha l'obbiettivo di **minimizzare lo *scarto quadratico medio***, ovverosia la somma dei quadrati degli errori nei nodi[12].

Dato lo **spazio dei polinomi di grado** m:

$$\mathbb{P}_m = \{p_m \ : \ \mathbb{R} \to \mathbb{R} \ : \ p_m(x) = b_0 + b_1 x + \cdots + b_m x^m\}$$

Allora, il (problema) **metodo dei minimi quadrati** consiste nel cercare il polinomio $q \in \mathbb{P}_m$ tale che:

$$\sum_{i=0}^{n} [y_i - q(x_i)]^2 \leq \sum_{i=0}^{n} [y_i - p_m(x_i)]^2 \qquad \forall p_m \in \mathbb{P}_m \tag{114}$$

Riscrivendo il polinomio $q(x)$ come:

$$q(x) = a_0 + a_1 x + \cdots + a_m x^m$$

[12] Si ricorda che l'errore di un nodo è la distanza tra la funzione lineare e il suo corrispettivo interpolatore. A pagina 48 è possibile vederlo visivamente.

Dove i coefficienti a sono incogniti, è possibile riformulare il metodo dei minimi quadrati (eq. 114). Quindi determinare $a_0, a_1, \ldots, a_m$ tali che:

$$\psi(a_0, a_1, \ldots, a_m) = \min_{\{b_i, i=0,\ldots,m\}} \psi(b_0, b_1, \ldots, b_m)$$

Dove:

$$\psi(b_0, b_1, \ldots, b_m) = \sum_{i=0}^{n} [y_i - (b_0 + b_1 x_i + \cdots + b_m x_i^m)]^2$$

Nel caso in cui si abbia una **retta di regressione** (cioè $m = 1$), il problema si riduce a:

$$\psi(b_0, b_1) = \sum_{i=0}^{n} \left[y_i^2 + b_0^2 + b_1^2 x_i^2 + 2b_0 b_1 x_i - 2b_0 y_i - 2b_1 x_i y_i\right]$$

Se ne ricava che il grafico della funzione ψ è un **paraboloide convesso** il cui punto di **minimo** (a_0, a_1) si trova imponendo le sue derivate parziali uguale a zero:

$$\frac{\partial \psi}{\partial b_0}(a_0, a_1) = 0 \qquad \frac{\partial \psi}{\partial b_1}(a_0, a_1) = 0$$

Che equivale a risolvere il seguente sistema di 2 equazioni e incognite:

$$\sum_{i=0}^{n} [a_0 + a_1 x_i - y_i] = 0 \qquad \sum_{i=0}^{n} \left[a_0 x_i + a_1 x_i^2 - x_i y_i\right] = 0$$

Ponendo $D = (n+1)\sum_{i=0}^{n} x_i^2 - \left(\sum_{i=0}^{n} x_i\right)^2$, la soluzione è:

$$\begin{aligned} a_0 &= \frac{1}{D}\left[\sum_{i=0}^{n} y_i \sum_{j=0}^{n} x_j^2 - \sum_{j=0}^{n} x_j \sum_{i=0}^{n} x_i y_i\right] \\ a_1 &= \frac{1}{D}\left[(n+1)\sum_{i=0}^{n} x_i y_i - \sum_{j=0}^{n} x_j \sum_{i=0}^{n} y_i\right] \end{aligned}$$

Il corrispondente polinomio:

$$q(x) = a_0 + a_1 x \tag{115}$$

È noto come **retta dei minimi quadrati**, o **retta di regressione**.

Attenzione

Da notare, anche se parzialmente scontato, che nel caso in cui m sia uguale a n ($m = n$), **il problema si riduce all'interpolatore Lagrangiano**.

4 Integrazione numerica

4.1 Introduzione

Alcuni problemi ingegneristici richiedono il calcolo di integrali, talvolta pure molto complessi. Non sempre si riesce a trovare in forma esplicita la primitiva di una funzione, anche nel caso in cui la si conosca, potrebbe essere difficile valutarla (come ad esempio $f(x) = \cos(4x)\cos(3\sin(x))$).

In tutti questi casi, è necessario utilizzare metodi numerici in gradi di restituire un valore approssimato della quantità di interesse, indipendentemente da quanto complessa sia la funzione da integrare o da differenziare. In questo capitolo, vengono presentati alcuni metodi per l'**approssimazione numerica di integrali di funzioni**, i quali vengono chiamate **formule di quadratura**.

Alcune notazioni e definizioni di introduzione

Si consideri l'integrale:

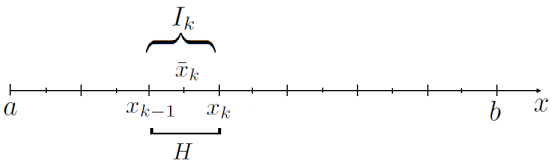
$$I(f) = \int_a^b f(x)\ \mathrm{d}x \tag{116}$$

Si suddivida l'intervallo $[a, b]$ in M intervalli I_k di ampiezza costante H:

$$[x_{k-1}, x_k] \quad k = 1, \ldots, M \qquad x_k = a + kH \quad k = 0, \ldots, M$$

Si introduce inoltre su ogni intervallo i punti medi:

$$\overline{x}_k = \frac{x_{k-1} + x_k}{2}$$



Si consideri una **formula di quadratura** per il calcolo approssimato dell'integrale nell'equazione 116 e sia $I_H(f)$ il valore approssimato ottenuto.

La **formula di quadratura** è di **ordine** p se l'*errore* soddisfa la seguente stima:

$$E_H = |I(f) - I_H(f)| \leq CH^p \tag{117}$$

Inoltre, la formula di quadratura ha **grado di esattezza pari ad** r se essa risulta esatta quando applicata **ai polinomi di grado minore o uguale ad** r, ovverosia se:

$$E_H = |I(f) - I_H(f)| = 0 \qquad \forall f \in \mathbb{P}^r(a, b) \tag{118}$$

4.2 Formula del punto medio composita

Data la proprietà di additività dell'integrale:

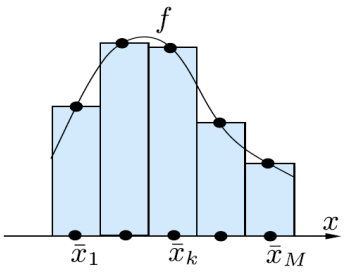
$$I(f) = \sum_{k=1}^{M} \int_{I_k} f(x)\ \mathrm{d}x$$

Si possono approssimare il valore dell'integrale su ogni intervallo e dopodiché sommare i contributi.

Per cui, l'idea del **punto medio composita** è quella di approssimare il valore dell'integrale della funzione:

$$\int_{I_k} f(x)\ \mathrm{d}x$$

Con l'area del rettangolo con altezza $f(\overline{x}_k)$:



Dato che l'altezza dei rettangoli è sempre pari ad H, si avrà la seguente approssimazione per $I(f)$:

$$I_{pm}(f) = H \sum_{k=1}^{M} f(\overline{x}_k) \tag{119}$$

Dove pm indica *punto medio*.

Inoltre, la **stima dell'errore del punto medio composita** è:

$$|I(f) - I_{pm}(f)| \leq \max_x |f''(x)|\ \frac{b-a}{24} H^2 \tag{120}$$

Si possono fare due **osservazioni** interessanti su questa stima dell'errore:

1. La formula del **punto medio composita** è di **ordine 2**.
2. La formula del **punto medio composita** ha **grado di esattezza pari a 1**. Dato che è stata assunta la continuità della derivata seconda di f, allora $f'' = 0$ in ogni x se f è una retta, ovverosia con $r = 1$.

Dall'ultima osservazione, risulta chiaro che la **formula del punto medio composita è esatta se applicata alle rette**.

4.3 Formula dei trapezi composita

Data la proprietà di additività dell'integrale:

$$I(f) = \sum_{k=1}^{M} \int_{I_k} f(x)\ \mathrm{d}x$$

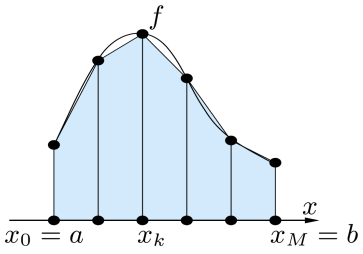
Si vuole approssimare l'area sottesa da f nell'intervallo I_k considerando il trapezio costruito sui punti x_{k-1} e x_k e sulle corrispondenti ordinate.

La formula dei **trapezi composita** è:

$$I_{tr}(f) = \frac{H}{2} \sum_{k=1}^{M} (f(x_{k-1}) + f(x_k)) \tag{121}$$

Per comodità di implementazione, è possibile riscriverla anche come:

$$I_{tr}(f) = \frac{H}{2} (f(a) + f(b)) + H \sum_{k=1}^{M-1} f(x_k)$$



Da notare che la formula dei trapezi composita equivale all'**integrale esatto dell'interpolatore Lagrangiano composito di grado 1**:

$$I_{tr}(f) = \int_a^b \left(\prod_1^H f(x) \right) \mathrm{d}x \tag{122}$$

Su ogni intervallo I_k si può considerare l'errore di interpolazione Lagrangiana.

La **stima dell'errore per la formula dei trapezi composita** è:

$$|I(f) - I_{tr}(f)| \leq \frac{1}{12} \max_x |f''(x)| (b-a) H^2 \tag{123}$$

Si possono fare due **osservazioni** riguardo alla stima dell'errore:

1. La **formula dei trapezi composita** è di **ordine 2**.
2. La **formula dei trapezi composita** ha **grado di esattezza pari a 1**.

Da notare che sia grado che ordine sono uguali al punto medio composita. Per cui, **è meglio scegliere questo metodo o il punto medio?** La scelta risiede principalmente sull'**avere a disposizione le coordinate dei punti medi o dei punti estremi degli intervalli**.

4.4 Formula di Simpson composita

Data la proprietà di additività dell'integrale:

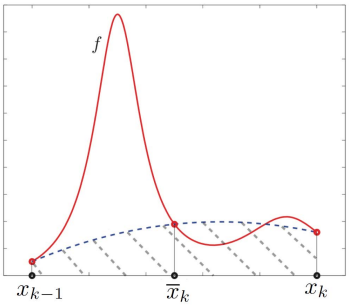
$$I(f) = \sum_{k=1}^{M} \int_{I_k} f(x)\ \mathrm{d}x$$

Si vuole approssimare l'area sottesa da f nell'intervallo I_k considerando l'area sottesa dalla parabola che interpola i punti x_{k-1}, $\overline{x}_k$ e x_k.

Integrando tali parabole su ogni intervallo I_k, si ottiene la **formula di Simpson composita**:

$$I_{sim}(f) = \frac{H}{6} \sum_{k=1}^{M} \left(f(x_{k-1}) + 4f(\overline{x}_k) + f(x_k) \right) \tag{124}$$

L'area sottesa alla parabole interpolante è quella tratteggiata in grigio:



Da notare che per la costruzione della forma di Simpson composita equivale all'**integrale esatto dell'interpolatore Lagrangiano composito di grado 2**:

$$I_{sim}(f) = \int_a^b \prod_{2}^{H} f(x)\ \mathrm{d}x \tag{125}$$

Assumendo che f abbia la derivata quarta continua su $[a, b]$, allora la **stima dell'errore per la formula dei trapezi composita** è:

$$|I(f) - I_{sim}(f)| \leq \frac{b-a}{2880} \max_x \left| f^{(iv)}(x) \right| H^4 \tag{126}$$

Si possono fare due **osservazioni** riguardo alla stima dell'errore:

1. La **formula di Simpson** è di **ordine 4**.
2. La **formula di Simpson** ha **grado di esattezza pari a 3**.

Per cui, tale metodo integra i polinomi che sono globalmente di:

- Grado $r = 1$ (rette)
- Grado $r = 2$ (parabole)
- Grado $r = 3$ (cubiche)

5 Approssimazione numerica di ODE

5.1 Problema di Cauchy

Un'equazione differenziale ordinaria ammette in generale infinite soluzioni. Per fissarne una è necessario imporre una condizione che prescriva il valore assunto dalla soluzione in un punto dell'intervallo di integrazione. In questa sezione ci si occuperà della risoluzione dei **problemi di Cauchy**, ossia di problemi nella seguente forma.

Problema di Cauchy

Sia $I = [t_0, T]$ l'intervallo temporale di interesse. Trovare la funzione vettoriale $\mathbf{y} : I \to \mathbb{R}^n$ che soddisfa:

$$\begin{cases} \mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}(t)) \\ \mathbf{y}(t_0) = \mathbf{y}_0 \end{cases} \tag{127}$$

Con $\mathbf{f} : I \times \mathbb{R}^n \to \mathbb{R}^n$ funzione vettoriale assegnata.

Problema di Cauchy *scalare*

Per semplicità, verrà considerato soltanto il **caso scalare**, ovvero in cui $n = 1$, ma i metodi di approssimazione che verranno introdotti saranno facilmente estendibili e applicabili anche con $n > 1$. Sia $I = [t_0, T]$ l'intervallo temporale di interesse. Trovare $y : I \subset \mathbb{R} \to \mathbb{R}$ tale che:

$$\begin{cases} y'(t) = f(t, y(t)) & \forall t \in I \\ y(t_0) = y_0 \end{cases} \tag{128}$$

Con $f : I \times \mathbb{R} \to \mathbb{R}$ funzione assegnata.

Definizione 1: esistenza e unicità della soluzione continua

Si supponga che la funzione $f(t, y)$ sia:

1. Limitata e continua rispetto ad entrambi gli argomenti.
2. Lipschitziana rispetto al secondo argomento, ossia esista una costante L positiva (detta costante di Lipschitz) tale per cui:

$$|f(t, y_1) - f(t, y_2)| \leq L |y_1 - y_2| \qquad \forall t \in I, \ \forall y_1, y_2 \in \mathbb{R} \tag{129}$$

Allora la **soluzione** del problema di Cauchy **esiste**, è **unica** ed è di **classe** C^1 su I.

5.2 Approssimazione di derivate

Al fine di approssimare il problema di Cauchy, è necessario prima introdurre un altro problema: **siano noti i valori $v(t_n)$ di una funzione v in corrispondenza di noti t_n, $n = 0, \dots, N$, e si vuole approssimare i valori della sua derivata prima $v'(t)$ negli stessi nodi**.

Approssimazione in avanti della derivata prima

Partendo dallo sviluppo in serie di Taylor:

$$v(t_{n+1}) = v(t_n) + hv'(t_n) + \frac{h^2}{2}v''(t_n) + \frac{h^3}{6}v'''(t_n) + O(h^4)$$

E ricordando che una quantità è un $O(h^p)$ se vale:

$$\lim_{h\to 0} \frac{O(h^p)}{h^p} < +\infty$$

Ovverosia se va a 0 con la stessa velocità o più velocemente di h^p. Scrivendo la derivata $v'(t_n)$ come:

$$v'(t_n) = \frac{v(t_{n+1}) - v(t_n)}{h} - \underbrace{\frac{h}{2}v''(t_n) - \frac{h^2}{6}v'''(t_n) + O(h^3)}_{O(h)}$$

Questo consente di introdurre un'**approssimazione in avanti della derivata prima**:

$$D^+ v(t_n) = \frac{v(t_{n+1}) - v(t_n)}{h} \tag{130}$$

Il cui **errore** è dato da:

$$\left| v'(t_n) - D^+ v(t_n) \right| = O(h) \tag{131}$$

Approssimazione all'indietro della derivata prima

In modo analogo, considerando il seguente sviluppo di Taylor:

$$v(t_{n-1}) = v(t_n) + hv'(t_n) + \frac{h^2}{2}v''(t_n) + \frac{h^3}{6}v'''(t_n) + O(h^4)$$

Usando i passaggi analoghi a quelli di prima, si arriva all'**approssimazione all'indietro della derivata prima**

$$D^- v(t_n) = \frac{v(t_n) - v(t_{n-1})}{h} \tag{132}$$

Il cui **errore** è dato da:

$$\left| v'(t_n) - D^- v(t_n) \right| = O(h) \tag{133}$$

Approssimazione centrata della derivata prima

Per completezza, si riporta anche l'**approssimazione centrata della derivata prima**:

$$D^c v(t_n) = \frac{v(t_{n+1}) - v(t_{n-1})}{2h} \tag{134}$$

E in questo caso l'**errore** commesso è:

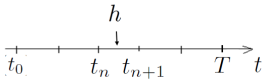
$$|v'(t_n) - D^c v(t_n)| \, 0O\left(h^2\right) \tag{135}$$

Infine, si evince che è un metodo di ordine 2.

E riguardo a Cauchy?

L'approssimazione numerica del problema di Cauchy si suddivide in 4 passi:

1. Si suddivide l'intervallo temporale in nodi:



2. Si scrive il problema di Cauchy per il generico nodo t_n valido per ogni $n = 1, \ldots, N = \frac{T}{h}$:
$$y'(t_n) = f(t_n, y(t_n))$$

3. Si sostituisce per ogni n a $y'(t_n)$ una delle approssimazioni presentate: in avanti (eq. 130), all'indietro (eq. 132) o centrata (eq. 134).

4. Si denota, per ogni n, con u_n la soluzione del nuovo problema approssimato ottenuto, la quale è una candidata per essere una buona approssimazione di $y(t_n)$.

La **soluzione approssimata** è composta in generale come:

$$\{u_0 = y_0, u_1, u_2, \ldots, u_n\}$$

5.3 I metodi di Eulero in avanti e all'indietro

Dato il problema di Cauchy, esso vale per ogni t nell'intervallo I, quindi in particolare anche in t_n:

$$y'(t_n) = f(t_n, y(t_n))$$

Approssimando la derivata sinistra con l'approssimazione in avanti introdotta a pagina 69, si ottiene:

$$D^+ y(t_n) = \frac{y(t_{n+1}) - y(t_n)}{h} \approxeq f(t_n, y(t_n))$$

Da notare che l'espressione assomiglia ad un'uguaglianza poiché l'espressione a sinistra è un'approssimazione della derivata e di conseguenza della funzione f.

L'idea è quella di introdurre come soluzione numerica u_n con $n = 1, \ldots, N$ la quale è una successione di valori che risolve in maniera esatta la precedente relazione.

$$\begin{aligned} \frac{y(t_{n+1}) - y(t_n)}{h} &\approxeq f(t_n, y(t_n)) \\ &\downarrow \\ \frac{u_{n+1} - u_n}{h} &= f(t_n, u_n) \end{aligned}$$

Da qui si ricava comodamente il **metodo di Eulero in avanti**:

$$u_{n+1} = u_n + hf(t_n, u_n) \qquad n = 0, \ldots, N-1 \tag{136}$$

In maniera analoga, partendo dal problema di Cauchy:

$$y'(t_{n+1}) = f(t_{n+1}, y(t_{n+1}))$$

Approssimando la derivata a sinistra con l'approssimazione all'indietro introdotto a pagina 69, si ottiene:

$$D^- y(t_{n+1}) = \frac{y(t_{n+1}) - y(t_n)}{h} \approxeq f(t_{n+1}, y(t_{n+1}))$$

Introducendo una soluzione numerica u_n con $n = 1, \ldots, N$ la quale è una successione di valori che risolve in maniera esatta la precedente relazione.

$$\begin{aligned} \frac{y(t_{n+1}) - y(t_n)}{h} &\approxeq f(t_{n+1}, y(t_{n+1})) \\ &\downarrow \\ \frac{u_{n+1} - u_n}{h} &= f(t_{n+1}, u_{n+1}) \end{aligned}$$

Da qui si ricava comodamente il **metodo di Eulero all'indietro**:

$$u_{n+1} = u_n + hf(t_{n+1}, u_{n+1}) \qquad n = 0, \ldots, N-1 \tag{137}$$

Perché il metodo di Eulero in avanti viene chiamato esplicito?

Per il metodo di Eulero in avanti, la condizione iniziale $u_0 = y_0$ è ben nota. Dopodiché, viene effettuato un **calcolo in sequenza**, ovverosia u_1, u_2, e così via. Tuttavia, non è un metodo iterativo poiché la soluzione è significativa per ogni n, essendo l'approssimazione della y a diversi istanti temporali.

Ovviamente i due metodi calcolano la nuova u_{n+1} in modo differente. Infatti, il metodo di Eulero in avanti calcola:

$$u_{n+1} = u_n + hf\left(t_n, u_n\right)$$

Il **nuovo valore è calcolato in maniera esplicita**. Difatti è sufficiente valutare la funzione f, anche se non lineare, per t_n, u_n ottenendo quindi un'espressione esplicita per il termine a destra. Questo è il motivo per cui il metodo di Eulero in avanti viene chiamato anche **metodo di Eulero esplicito**.

Perché il metodo di Eulero all'indietro viene chiamato implicito?

Per il metodo di Eulero all'indietro:

$$u_{n+1} = u_n + hf\left(t_{n+1}, u_{n+1}\right)$$

Non si ha un'espressione esplicita per il nuovo valore u_{n+1} **perché quest'ultimo compare anche a destra dell'uguale**!

Si pone come incognita $x = u_{n+1}$. Se la funzione f non è lineare rispetto a y, allora ad ogni passo temporale è necessario risolvere il **problema di ricerca degli zeri** della funzione:

$$g\left(x\right) = x - u_n - hf\left(t_{n+1}, x\right) \tag{138}$$

Se la funzione $g\left(x\right)$ è derivabile, allora è necessario introdurre ad ogni istante temporale un metodo iterativo, come quello di Newton per esempio. In ogni caso, a causa del fatto che u_{n+1} non può in generale essere determinato per via esplicita, il metodo di Eulero all'indietro viene chiamato anche **metodo di Eulero implicito**.

In generale, si possono **dividere i metodi numerici** per le equazioni ordinarie in **metodi espliciti**, la cui **soluzione al nuovo passo temporale è calcolata mediante un'espressione esplicita**, e **metodi impliciti** che invece **richiedono la soluzione di un'equazione non lineare**.

5.3.1 Assoluta stabilità

È stabile il metodo di Eulero?

Per discutere della stabilità, è necessario introdurre il concetto di **assoluta stabilità**. Un metodo numerico viene detto **assolutamente stabile** per un determinato valore di h maggiore di zero, se:

$$|u_{n+1}| \leq C_{AS} |u_n| \qquad C_{AS} < 1 \tag{139}$$

O, equivalentemente, se si è su un intervallo illimitato:

$$\lim_{n \to +\infty} |u_n| = 0$$

Adesso si analizzano i metodi singolarmente.

Assoluta stabilità: metodo di Eulero in avanti

Dato il **problema modello lineare**:

$$\begin{cases} y'(t) = \lambda y(t) & \lambda < 0 \\ y(0) = 1 \end{cases}$$

La cui soluzione esatta è: $y(t) = e^{\lambda t}$. Applicando il metodo di Eulero in avanti, si ottiene:

$$u_{n+1} = u_n + h\lambda u_n = (1 + h\lambda) u_n \longrightarrow C_{AS} = |1 + h\lambda|$$

Da cui l'assoluta stabilità è **garantita se**:

$$C_{AS} < 1 \longrightarrow -1 < 1 + h\lambda < 1$$

Ovvero h deve rispettare la seguente condizione:

$$h\lambda > -2 \Rightarrow h < -\frac{2}{\lambda} \tag{140}$$

Da notare che $h\lambda < 0$ è sempre vera.

Esempio 1: stabilità metodo di Eulero in avanti

Data la soluzione di un problema con $\lambda = -1$, ottenuto con il metodo di Eulero in avanti, nella seguente figura si può vedere come:

- Linea tratteggiata: rappresentata $h = \frac{30}{14}$. Quindi, essendo:

  $$h < -\frac{2}{\lambda} \Rightarrow \frac{30}{14} < -\frac{2}{-1} \Rightarrow \frac{30}{14} < 2 \text{ ✗}$$

  La soluzione numerica oscilla e infine esplode.

- Linea continua: rappresentata $h = \frac{30}{16}$. Quindi, essendo:

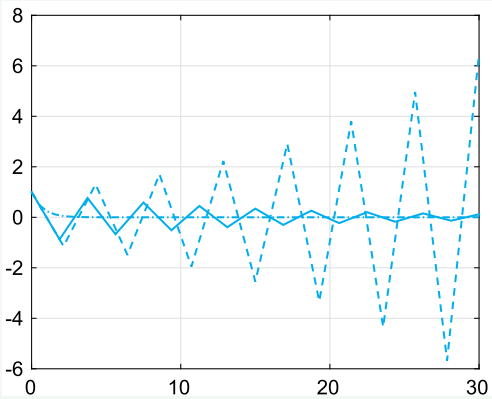
  $$\frac{30}{16} < 2 \text{ ✓}$$

La soluzione numerica oscilla ma non esplode. La sua oscillazione è dovuta al fatto che è molto vicina al limite 2.

- Linea tratto-punto: rappresentata $h = \frac{1}{2}$. Quindi, essendo:

$$\frac{1}{2} < 2 \checkmark$$

La soluzione numerica non oscilla.



Assoluta stabilità: metodo di Eulero all'indietro

Dato il solito problema modello lineare:

$$\begin{cases} y'(t) = \lambda y(t) & \lambda < 0 \\ y(0) = 1 \end{cases}$$

La cui soluzione esatta è: $y(t) = e^{\lambda t}$. Applicando il metodo di Eulero all'indietro, si ottiene:

$$u_{n+1} = u_n + h\lambda u_{n+1} \rightarrow u_{n+1} = \frac{1}{1-h\lambda} u_n \rightarrow C_{AS} = \left| \frac{1}{1-h\lambda} \right|$$

In questo caso, l'**assoluta stabilità è sempre garantita**, poiché:

$$C_{AS} < 1 \quad \forall h$$

Quindi, per il metodo di Eulero all'indietro si può essere certi che è **incondizionatamente assolutamente stabile**.

5.3.2 Problemi di Cauchy generali

Il concetto di stabilità introdotto nel paragrafo precedente riguarda il *problema modello lineare*. Per cui, per i problemi di Cauchy, in generale, che stabilità si ha?

Un metodo numerico produce una soluzione stabile se a *piccole* perturbazioni sul dato iniziale si producono perturbazioni *piccole* e decrescenti per n crescente.

Notando che per il *problema modello lineare* si ha:

$$f(y) = \lambda y \;\rightarrow\; f'(y) = \lambda$$

Ha senso introdurre la **stima dell'intervallo dei valori** di h che garantiscono una **soluzione stabile** per il ***metodo di Eulero in avanti***:

$$h < -\frac{2}{\overline{\lambda}} \qquad \overline{\lambda} = -\max_{t,y}\left|\frac{\partial f}{\partial y}\right| \qquad \frac{\partial f}{\partial y} < 0 \tag{141}$$

Essa rappresenta una situazione di cautela, la quale potrebbe essere molto restrittiva per il problema di Cauchy, ma sicuramente garantisce una soluzione numerica stabile.

In generale, vale la proprietà che un **metodo numerico assolutamente stabile per il *problema modello lineare*** sotto una condizione su h (piccolo a sufficienza) **produce una soluzione numerica stabile anche per il problema di Cauchy generico sotto la stessa condizione opportunatamente adattata**.

5.4 Il metodo di Crank-Nicolson

Dato il problema di Cauchy:

$$\begin{cases} y'(t) = f(t, y(t)) & \forall t \in I \, y(t_0) = y_0 \end{cases}$$

Dal teorema fondamentale del calcolo integrale limitato all'intervallo $[t_n, t_{n+1}]$ si ottiene:

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(t, y(t)) \, \mathrm{d}t$$

Adesso si applica una formula di quadratura a disposizione (punto medio composita, trapezi composita, Simpson composita) per approssimare l'integrale a destra dell'uguale e ottenere così un metodo di approssimazione per il problema di Cauchy.

Ed ecco che nasce il **metodo di Crank-Nicolson (CN)** applicando la formula dei trapezi composita (par. 4.3, pag. 66) considerando solo un intervallo $[t_n, t_{n+1}]$:

$$u_{n+1} = u_n + \frac{h}{2}\left(f(t_n, u_n) + f(t_{n+1}, u_{n+1})\right) \tag{142}$$

Esso è un **metodo implicito** e richiede ad ogni passo n di determinare la radice della funzione:

$$g(x) = x - \frac{h}{2} f(t_{n+1}, x) - u_n - \frac{h}{2} f(t_n, u_n)$$

Riguardo l'**assoluta stabilità**, si ha:

$$u_{n+1} = u_n + \frac{h\lambda}{2}(u_n + u_{n+1}) \;\rightarrow\; u_{n+1} = \frac{2 + h\lambda}{2 - h\lambda} u_n$$

Di conseguenza si ottiene:

$$C_{AS} = \left| \frac{2 + h\lambda}{2 - h\lambda} \right|$$

Che è sempre minore di 1. Per cui il metodo di Crank-Nicolson è **incondizionatamente assolutamente stabile**.

5.5 Il metodo di Heun

Partendo dal metodo di Crank-Nicolson introdotto nel paragrafo 5.4 a pagina 76:

$$u_{n+1} = u_n + \frac{h}{2}\left(f\left(t_n, u_n\right) + f\left(t_{n+1}, u_{n+1}\right)\right)$$

E al posto della variabile u_{n+1} a destra si introduce la sua approssimazione ottenuta con il metodo di Eulero in avanti, si ottiene il **metodo di Heun**:

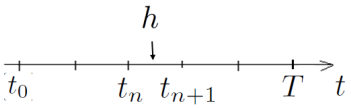
$$u_{n+1} = u_n + \frac{h}{2}\left(f\left(t_n, u_n\right) + f\left(t_{n+1}, u_n + hf\left(t_n, u_n\right)\right)\right) \tag{143}$$

Esso è un **metodo esplicito** con **condizione di assoluta stabilità uguale a quella del metodo di Eulero in avanti** (si veda a pagina 73 l'assoluta stabilità di tale metodo).

5.6 Convergenza

Per discutere l'**accuratezza della soluzione numerica** trovata da ciascun metodo introdotto per determinati valori di h, è necessario introdurre la definizione di **convergenza**:

$$|y(t_n) - u_n| = O(h) \qquad n = 0, \ldots, N \tag{144}$$



Modificando la convergenza appena introdotta (eq. 144), essa rappresenta la **stima dell'errore di convergenza**:

$$|y(t_n) - u_n| = O(h^p) \qquad n = 0, \ldots, N \tag{145}$$

Inoltre si dice che il **metodo numerico è di ordine** p. È interessante notare che all'**aumentare di** p, la **velocità di convergenza aumenta**.

Utilizzando λ del coefficiente del *problema modello lineare*, oppure:

$$\overline{\lambda} = -\max_{t,y} \left| \frac{\partial f}{\partial y} \right| \qquad \text{con } \frac{\partial f}{\partial y} < 0$$

Nella tabella 1 sono riassunti i metodi introdotti in questo capitolo.

Metodo	**Espl./Impl.**	**Convergenza**	**Assoluta stabilità**
Eulero in avanti	Esplicito	$O(h)$	$h < -\frac{2}{\lambda}$
Eulero all'indietro	Implicito	$O(h)$	Sempre garantita
Crank-Nicolson	Implicito	$O(h^2)$	Sempre garantita
Heun	Esplicito	$O(h^2)$	$h < -\frac{2}{\lambda}$

Tabella 1: Proprietà di convergenza e stabilità dei metodi per approssimare le ODE.

5.6.1 Consistenza dei metodi di Eulero

Innanzitutto si introduce l'**errore di troncamento locale** τ_n, il quale si **commette introducendo la soluzione esatta** y nello schema numerico. Per il **metodo di Eulero in avanti** si ha:

$$\tau_n = \left| f\left(t_n, y\left(t_n\right)\right) - \frac{y\left(t_{n+1}\right) - y\left(t_n\right)}{h} \right| \tag{146}$$

E per il **metodo di Eulero all'indietro** si ha:

$$\tau_n = \left| f\left(t_{n+1}, y\left(t_{n+1}\right)\right) - \frac{y\left(t_{n+1}\right) - y\left(t_n\right)}{h} \right| \tag{147}$$

Un metodo numerico è **consistente** se vale:

$$\lim_{h \to 0} \tau_n = 0 \qquad \forall n \tag{148}$$

Inoltre, se vale:

$$\tau_n = O\left(h^p\right) \qquad \forall n \tag{149}$$

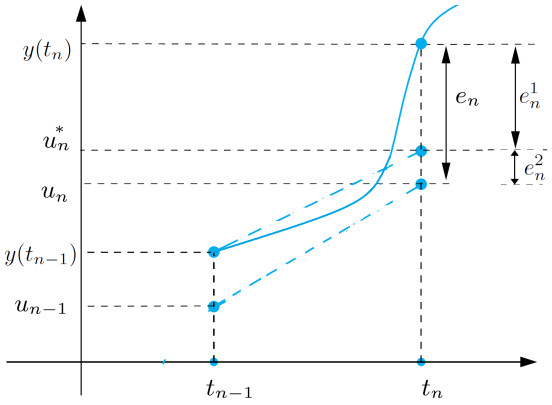
Si dice che l'**ordine di consistenza è** p. Riguardo i ***metodi di Eulero***, essi sono **consistenti del primo ordine**.

Si noti che l'**errore di troncamento locale** τ_n non è l'errore necessario per verificare la convergenza: $\left|y\left(t_n\right) - u_n\right|$. L'errore:

$$e_n = \left|y\left(t_n\right) - u_n\right|$$

È dato da due contributi:

- L'**errore** e_n^1 è **dovuto localmente al metodo numerico**, ottenuto partendo dalla vera soluzione $y\left(t_{n-1}\right)$.
- L'**errore** e_n^2 è **dovuto alla *propagazione degli errori*** commessi agli istanti precedenti



Riguardo al **metodo di Eulero in avanti** l'**errore** è dato da:

- Primo contributo dovuto localmente al metodo numerico:

$$e_n^1 = |y(t_n) - y(t_{n-1}) - hf(t_{n-1}, y(t_{n-1}))| = h\tau_{n-1} \tag{150}$$

  Da notare che il primo errore, a meno di un fattore h, è dato dall'errore di troncamento locale. Questo evidenzia come la **consistenza da sola non basti per avere la convergenza**.

- Secondo contributo dovuto alla propagazione degli errori:

$$e_n^* = |u_n^* - u_n| \tag{151}$$

Supponendo che valga la **condizione di assoluta stabilità** con $\overline{\lambda} = -\lambda_{\max}$:

$$h < \frac{2}{\lambda_{\max}}$$

Dunque si ha:

$$\begin{gathered} |e_n| \leq \left|e_n^1\right| + \left|e_n^2\right| \leq h\tau_{n-1} + |e_{n-1}| \\ \downarrow \\ |e_n| \leq nhO(h) = (t_n - t_0) O(h) \end{gathered} \tag{152}$$

Il **metodo di Eulero in avanti** è dunque **convergente** e del **primo ordine**. Lo stesso risultato vale anche per il **metodo di Eulero all'indietro**. In generale, è possibile affermare che, per un metodo convergente, l'**ordine di convergenza è uguale all'ordine di consistenza**.

La **convergenza** è una **proprietà della soluzione numerica** (o meglio all'errore), per n fissato, in ogni nodo e facendo tendere h a 0. Invece, l'assoluta stabilità analizza il comportamento della soluzione numerica per h fissato e al crescere di n.

In generale, i **metodi espliciti** (come Eulero in avanti e Heun) **garantiscono l'assoluta stabilità solo per un valore di h piccolo**. I **metodi impliciti** godono invece di ottime proprietà di **assoluta stabilità**; spesso questa è **incondizionata** (come per Eulero all'indietro e Crank-Nicolson).

6 Laboratorio

6.1 Introduzione al linguaggio MATLAB

L'introduzione al linguaggio di programmazione MATLAB sarà molto rapido. Si assume dunque che l'interfaccia grafica sia familiare e che concetti base di programmazione (per esempio "che cos'è una variabile?") siano ben noti.

In MATLAB, l'assegnazione di scalari a delle variabili è classica, quindi si utilizza il simbolo uguale: `a = 1` (assegnazione del valore `1` alla variabile `a`). Inoltre, il linguaggio è *case sensitive*, di conseguenza la variabile `a` è diversa dalla variabile `A`. Alcuni comandi utili e generali:

- `help` *`nome-comando`*, per avere informazioni in più riguardo al comando *`nome-comando`*;
- `clear` *`nome-variabile`*, per rimuovere la variabile *`nome-variabile`* dalla memoria. Se non viene inserito il *`nome-variabile`*, vengono rimosse tutte le variabili dalla memoria.
- `who`, per visualizzare le variabili attualmente in memoria.
- `clc`, per ripulire la *Command Window*.

Tabella 2: Argomenti trattati.

Well-known variables

Esistono alcune variabili che sono ben note e hanno valori prestabiliti. Tra le più importanti:

- `pi`, che rappresenta il π e MATLAB gli assegna il valore `3.1416`
- `i`, che rappresenta l'unità immaginaria e MATLAB gli assegna il valore `0.0000 + 1.0000i`
- `eps`, che rappresenta il più piccolo valore rappresentabile nel calcolatore (PC) attualmente in uso. Solitamente, `eps` ritorna il valore `2.2204e-16`.

Questo tipo di variabili possono essere ridefinite, ma non è una *good practice*.

Cambiare il formato delle variabili: `format`

Il comando `format` è utilizzato per cambiare il formato con cui sono rappresentate le variabili. MATLAB non cambia la precisione della variabile (quindi non si ottiene una precisione maggiore dopo la virgola), ma modifica soltanto la rappresentazione. Di default MATLAB utilizza una rappresentazione di tipo `short`. Tra i più utilizzati (di default `pi` è uguale a `3.1416`):

- `default` per reimpostare la rappresentazione di default.
- Decimale:
  - `short`, rappresentazione a 5 cifre:

```
>> format short
>> pi

ans =
    3.1416
```

  - `long`, rappresentazione a 15 cifre:

```
>> format long
>> pi

ans =
   3.141592653589793
```

- *Floating point*:
  - `short e`, rappresentazione a 5 cifre floating point:

```
>> format short e
>> pi

ans =
   3.1416e+00
```

  - `long e`, rappresentazione a 15 cifre floating point:

```
>> format long e
>> pi

ans =
     3.141592653589793e+00
```

Altri formati si possono trovare nella documentazione ufficiale.

Assegnamento di vettori e matrici

- **Vettore riga**, si può creare utilizzando uno spazio tra i valori o una virgola `,`:

```
>> a = [1 2 3 4]

a =
     1     2     3     4

>> b = [1, 2, 3, 4]

b =
     1     2     3     4
```

- **Vettore colonna**, si crea usando il punto e virgola `;`:

```
>> a = [1; 2; 3; 4]

a =
     1
     2
     3
     4
```

Talvolta può essere utile la generazione automatica di un vettore riga (sono ammessi anche i valori negativi e con la virgola ovviamente):

- **Vettore riga generato linearmente**, si crea usando i due punti e specificando il valore di inizio e il valore di fine:

```
>> a = [1 : 4]

a =
     1     2     3     4
```

- **Vettore riga generato usando un passo**, si crea usando i due punti e specificando (in ordine) il valore di inizio, il "salto", e il valore di fine. Nel caso in cui il salto sia troppo grande e si superi il valore di fine, MATLAB prenderà il primo valore ammissibile:

```
>> % Generazione con passo 1
>> a = [1 : 1 : 4]

a =
     1     2     3     4

>> % Generazione con passo 2
>> a = [1 : 2 : 5]

a =
     1     3     5

>> % Generazion con passo 2 (fine non raggiunta)
>> a = [1 : 2 : 6]

a =
     1     3     5

>> % ... ma cambiando l'upper bound
```

```
>> a = [1 : 2 : 7]

a =
     1     3     5     7
```

- **Vettore riga generato con valori uniformemente distanziati**, si crea usando la funzione `linspace`, la quale accetta tre parametri:
  - `x1`, valore di partenza.
  - `x2`, valore di fine.
  - `n`, numero di valori da generare; se non specificato, di default è `100`; se il valore inserito è zero o minore, viene creato un vettore vuoto.

```
>> % Vettore riga identico a: a = [1 : 4]
>> linspace(1, 4, 4)

ans =
     1     2     3     4

>> % Chiedendo piu' valori, MATLAB andra' ad utilizzare i
    decimali
>> linspace(1, 4, 6)
ans =

     1.000000000000000e+00     1.600000000000000e+00
     2.200000000000000e+00     2.800000000000000e+00
     3.400000000000000e+00     4.000000000000000e+00
```

Le matrici possono essere create a mano o usando la combinazione delle tecniche viste in precedenza:

- **Matrice**, le righe si creano usando gli spazi e le colonne si creano usando il punto e virgola:

```
>> a = [1 2 3 4; 5 6 7 8; 9 10 11 12]

a =
     1     2     3     4
     5     6     7     8
     9    10    11    12

>> a = [1, 2, 3, 4; 5, 6, 7, 8; 9, 10, 11, 12]

a =
     1     2     3     4
     5     6     7     8
     9    10    11    12
```

- **Matrice creata usando la generazione lineare dei vettori**, si possono utilizzare le tecniche precedenti e i punti e virgola:

```
>> % Usando a = [x : y]
>> a = [1 : 4; 5 : 8; 9 : 12]

a =
     1     2     3     4
     5     6     7     8
     9    10    11    12

```

```
>> % Usando a = [x : y : z]
>> a = [1 : 2 : 7; 9 : 2 : 15; 17 : 2 : 23]
a =

     1     3     5     7
     9    11    13    15
    17    19    21    23

>> % Usando linspace
>> a = [linspace(1, 4, 4); linspace(5, 8, 4); linspace(9, 12,
    4)]

a =
     1     2     3     4
     5     6     7     8
     9    10    11    12
```

Operazioni su vettori e matrici

- **Trasposizione**, la classica operazione eseguita con le matrici o vettori, si esegue con la keyword `'` oppure usando la funzione `transpose`:

```
>> a = [1 2 3 4]

a =
     1     2     3     4

>> a'

ans =
     1
     2
     3
     4

>> transpose(a)

ans =
     1
     2
     3
     4
```

- **Somma e sottrazione**
  - Tra **vettore e scalare**:

```
>> a = [1 2 3 4]

a =
     1     2     3     4

>> a + 1

ans =
     2     3     4     5

>> a - 1

ans =
     0     1     2     3
```

  - Tra **vettore e matrice**:

```
>> a = [1 2 3 4]

a =
     1     2     3     4

>> b = [1 2 3 4; 5 6 7 8; 9 10 11 12]

b =
     1     2     3     4
     5     6     7     8
     9    10    11    12

>> a + b

ans =
```

```
     2     4     6     8
     6     8    10    12
    10    12    14    16

>> a - b

ans =
     0     0     0     0
    -4    -4    -4    -4
    -8    -8    -8    -8
```

– Tra **matrice e scalare**:

```
>> b = [1 2 3 4; 5 6 7 8; 9 10 11 12]

b =
     1     2     3     4
     5     6     7     8
     9    10    11    12

>> b + 1

ans =
     2     3     4     5
     6     7     8     9
    10    11    12    13

>> b - 1

ans =
     0     1     2     3
     4     5     6     7
     8     9    10    11
```

– Tra **matrice e matrice**:

```
>> b = [1 2 3 4; 5 6 7 8; 9 10 11 12]

b =
     1     2     3     4
     5     6     7     8
     9    10    11    12

>> c = [13 14 15 16; 17 18 19 20; 21 22 23 24]

c =
    13    14    15    16
    17    18    19    20
    21    22    23    24

>> b + c

ans =
    14    16    18    20
    22    24    26    28
    30    32    34    36

>> b - c

ans =
   -12   -12   -12   -12
   -12   -12   -12   -12
   -12   -12   -12   -12
```

- **Prodotto**
  - **Prodotto matriciale**:

```
>> b = [1 2 3 4; 5 6 7 8; 9 10 11 12]

b =
     1     2     3     4
     5     6     7     8
     9    10    11    12

>> c = [13 14 15; 16 17 18; 19 20 21; 22 23 24]

c =
    13    14    15
    16    17    18
    19    20    21
    22    23    24

>> b * c

ans =
   190   200   210
   470   496   522
   750   792   834
```

  - **Prodotto punto per punto**, in MATLAB è possibile moltiplicare ogni cella di una matrice (o vettore) per la corrispettiva cella della matrice (o vettore) moltiplicata. La keyword utilizzata è `.*`:

```
>> b = [1 2 3 4; 5 6 7 8; 9 10 11 12]

b =
     1     2     3     4
     5     6     7     8
     9    10    11    12

>> c = [13 14 15 16; 17 18 19 20; 21 22 23 24]

c =
    13    14    15    16
    17    18    19    20
    21    22    23    24

>> b .* c

ans =
    13    28    45    64
    85   108   133   160
   189   220   253   288

>> d = [1 2 3 4]

d =
     1     2     3     4

>> b .* d

ans =
     1     4     9    16
     5    12    21    32
     9    20    33    48
```

- **Potenza**
  - **Potenza matriciale**:

```
>> b = [1 2 3; 4 5 6; 7 8 9]

b =
     1     2     3
     4     5     6
     7     8     9

>> b^2

ans =
    30    36    42
    66    81    96
   102   126   150
```

  - **Potenza punto per punto**, come per il prodotto, è possibile elevare al quadrato ogni valore della matrice (o vettore):

```
>> b = [1 2 3; 4 5 6; 7 8 9]

b =
     1     2     3
     4     5     6
     7     8     9

>> b.^2

ans =
     1     4     9
    16    25    36
    49    64    81
```

Funzioni intrinseche per vettori e matrici

Qua di seguito si elencano le funzioni più importanti da utilizzare per i vettori e le matrici.

- `size`, restituisce la dimensione del vettore o della matrice nel formato *`righe colonne`*. Specificando anche un valore (o vettore) come parametro, la funzione restituisce la dimensione (un vettore contenente le dimensioni richieste) nella "dimensione" richiesta:

```
>> a = [1 2 3 4]

a =
     1     2     3     4

>> size(a)

ans =
     1     4

>> size(a, 2)
ans =

     4

>> b = [1 2 3; 4 5 6; 7 8 9]

b =
     1     2     3
     4     5     6
     7     8     9

>> size(b)

ans =
     3     3

>> size(b, [2, 3])

ans =
     3     1
```

- `length`, restituisce la lunghezza del vettore e per le matrici restituisce il numero degli elementi per ogni riga:

```
>> a = [1 2 3 4]

a =
     1     2     3     4

>> length(a)

ans =
     4

>> b = [1 2 3 4 5 6; 7 8 9 10 11 12]

b =
     1     2     3     4     5     6
     7     8     9    10    11    12

```

```
>> length(b)

ans =
     6
```

- `max`, `min`, calcolano rispettivamente il massimo e il minimo valore delle componenti di un vettore; per le matrici viene presa in considerazione ogni colonna e calcolato il massimo o minimo:

```
>> a = [ 1 2 3 4]

a =
     1     2     3     4

>> max(a)

ans =
     4

>> min(a)

ans =
     1

>> b = [7 1 1 4; 2 3 9 10; 8 1 7 1]

ans =
     7     1     1     4
     2     3     9    10
     8     1     7     1

>> max(b)

ans =
     8     3     9    10

>> min(b)

ans =
     2     1     1     1
```

- `sum`, `prod`, calcola rispettivamente la somma e il prodotto degli elementi che compongono il vettore; nel caso di una matrice, viene presa in considerazione ogni colonna e calcolata la somma o il prodotto. Inoltre, i due comandi possono prendere un argomento in più per eseguire il calcolo in una dimensione specifica (cosa sensata con le matrici):

```
>> a = [ 1 2 3 4]

a =
     1     2     3     4

>> sum(a)

ans =
    10

>> prod(a)

ans =
    24
```

```
>> b = [7 1 1 4; 2 3 9 10; 8 1 7 1]

b =
     7     1     1     4
     2     3     9    10
     8     1     7     1

>> sum(b) % per colonne

ans =
    17     5    17    15

>> sum(b, 2) % per righe

ans =
    13
    24
    17

>> prod(b)

ans =
   112     3    63    40
```

- `norm`, la norma di un vettore o di una matrice. Passando un vettore o un matrice, viene calcolata di default la norma euclidea (norma 2):

$$||\mathrm{v}||_2 = \sqrt{\sum_{i=2}^{\texttt{length(v)}} \mathrm{v}_i^2}$$

  Passando un valore aggiuntivo, esso rappresenterà l'ordine della norma:

$$||\mathrm{v}||_n = \left(\sum_{i=2}^{\texttt{length(v)}} |\mathrm{v}_i|^n \right)^{\frac{1}{n}}$$

  Infine, con `inf` viene calcolata la norma infinito:

$$||\mathrm{v}||_\infty = \max_{1 \leq i \leq \texttt{length(v)}} |\mathrm{v}_i|$$

```
>> a = [1 2 3 4]

a =
     1     2     3     4

>> norm(a)

ans =
     5.477225575051661e+00

>> norm(a, 2)

ans =
     5.477225575051661e+00

>> norm(a, 3)
```

```
17
18 ans =
19      4.641588833612779e+00
20
21 >> norm(a, inf)
22
23 ans =
24      4
25
26 >> b = [7 1 1 4; 2 3 9 10; 8 1 7 1]
27
28 b =
29      7      1      1      4
30      2      3      9     10
31      8      1      7      1
32
33 >> norm(b, 2)
34
35 ans =
36      1.711222312384884e+01
37
38 >> norm(b, inf)
39
40 ans =
41     24
```

- `abs`, rappresenta il valore assoluto e restituisce il vettore o matrice dopo aver applicato il valore assoluto a ciascun elemento:

```
1  >> a = [-1 -2 -3 -4]
2
3  a =
4      -1     -2     -3     -4
5
6  >> abs(a)
7
8  ans =
9       1      2      3      4
10
11 >> b = [7 1 1 -4; 2 3 -9 10; 8 1 -7 1]
12 b =
13
14      7      1      1     -4
15      2      3     -9     10
16      8      1     -7      1
17
18 >> abs(b)
19
20 ans =
21      7      1      1      4
22      2      3      9     10
23      8      1      7      1
```

- `diag`, estrae la diagonale di una matrice esistente, oppure ne crea una con i valori dati come input. Inoltre, può creare una matrice con la diagonale spostata a seconda del valore dato (si veda l'esempio):

```
>> b = [7 1 1 4; 2 3 9 10; 8 1 7 1]

b =
     7     1     1     4
     2     3     9    10
     8     1     7     1

>> diag(b)

ans =
     7
     3
     7

>> diag(b, 1)

ans =
     1
     9
     1

>> diag(b, -1)

ans =

     2
     1

>> diag([1 2 3])

ans =
     1     0     0
     0     2     0
     0     0     3

>> diag([1 2 3], -1)

ans =
     0     0     0     0
     1     0     0     0
     0     2     0     0
     0     0     3     0
```

Funzioni matematiche elementari

Qua di seguito una lista di alcune funzioni matematiche elementari. Gli esempi e la sintassi non verranno mostrati poiché è sempre la medesima:

funzione(*parametro*)

Funzione	Comando
Radice quadrata	`sqrt`
Esponenziale	`exp`
Logaritmo Naturale	`log`
Logaritmo In Base 2	`log2`
Logaritmo In Base 10	`log10`
Seno	`sin`
Arcoseno	`asin`
Coseno	`cos`
Arcocoseno	`acos`
Tangente	`tan`

Tabella 3: Funzioni matematiche elementari.

Iterazione con il ciclo for

In MATLAB il ciclo for viene eseguito con la seguente sintassi.

```
for index = values
   statements
end
```

Di seguito si riporta un ciclo `for` che itera sulla diagonale secondaria di una matrice:

```
>> b = [7 1 1 4; 2 3 9 10; 8 1 7 1]

b =
     7     1     1     4
     2     3     9    10
     8     1     7     1

>> res = []

res =
     []

>> for i = 1 : size(b, 1)
    res(i) = b(i, size(b, 1) - i + 1);
end

>> res

res =
     1     3     8
```

Funzioni per definire vettori o matrici particolari

In queste pagine vengono presentate alcuni funzioni utili che consentono di creare matrici o vettori "particolari".

- **Vettore/Matrice nulla**, con la funzione `zeros` è possibile creare una matrice o un vettore di tutti zeri. I parametri ammessi corrispondono alla dimensione del vettore o matrice:

```
>> zeros(1, 4)

ans =
     0     0     0     0

>> zeros(4, 1)

ans =
     0
     0
     0
     0

>> zeros(4, 4)

ans =
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
```

- **Vettore unario/Matrice unaria**, con la funzione `ones` è possibile creare una matrice o un vettore di tutti uni. I parametri ammessi corrispondono alla dimensione del vettore o matrice:

```
>> ones(1, 4)

ans =
     1     1     1     1

>> ones(4, 1)

ans =
     1
     1
     1
     1

>> ones(4, 4)

ans =
     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1
```

- **Matrice identità**, con la funzione `eye` è possibile creare una matrice identità. I parametri ammessi corrispondono alla dimensione del vettore o matrice:

```
>> eye(3)

ans =
     1     0     0
     0     1     0
     0     0     1

>> eye(2, 3)

ans =
     1     0     0
     0     1     0

>> eye(1, 4)

ans =
     1     0     0     0
```

- **Matrice/Vettore riga di numeri casuali interi e non**, con il comando `rand` si genera una matrice di numeri casuali nell'intervallo $[0, 1]$ con la virgola, mentre con il comando `randi` si genera una matrice di numeri casuali interi (primo parametro deve essere specificato il range dei valori):

```
>> rand(3, 5)

ans =
    0.9157    0.6557    0.9340    0.7431    0.1712
    0.7922    0.0357    0.6787    0.3922    0.7060
    0.9595    0.8491    0.7577    0.6555    0.0318

>> % Matrice di valori interi random da 1 a 5
>> randi([1, 5], 3, 4)

ans =
     2     5     5     2
     1     4     1     4
     1     2     3     4

>> % Errore! L'intervallo e' sbagliato
>> randi([-1, -50], 3, 4)
Error using randi
First input must be a positive scalar integer value IMAX, or
    two integer values [IMIN IMAX] with IMIN less than or
    equal to IMAX.

>> randi([-50, -1], 3, 4)

ans =
   -41   -18   -37   -42
   -26   -15   -17   -45
   -28   -13   -18   -26
```

6.1.1 Esercizio

Creare una funzione (file) chiamato `mat_hilbert.m` che fornisca la matrice di Hilbert avente una generica dimensione `n`. Ogni cella della matrice di Hilbert deve rispettare la seguente condizione:

$$a_{ij} = \frac{1}{i + j - 1}$$

Dopo aver creato la funzione, utilizzare la funzione nativa di MATLAB `hilb`, per verificare il risultato ottenuto.

Soluzione

Il codice non ha bisogno di grandi spiegazioni. Vi è un controllo iniziale per verificare l'argomento inserito dall'utente e successivamente due cicli `for` per popolare la matrice:

```
function hilbert_matrix = mat_hilbert(n)

if n < 0
    error("n can't be 0 or less than zero")
end

hilbert_matrix = zeros(n);
for i = 1 : n
    for j = 1 : n
        hilbert_matrix(i, j) = 1 / (i + j - 1);
    end
end
```

Il risultato:

```
mat_hilbert(5)

ans =

    1.0000    0.5000    0.3333    0.2500    0.2000
    0.5000    0.3333    0.2500    0.2000    0.1667
    0.3333    0.2500    0.2000    0.1667    0.1429
    0.2500    0.2000    0.1667    0.1429    0.1250
    0.2000    0.1667    0.1429    0.1250    0.1111

hilb(5)

ans =

    1.0000    0.5000    0.3333    0.2500    0.2000
    0.5000    0.3333    0.2500    0.2000    0.1667
    0.3333    0.2500    0.2000    0.1667    0.1429
    0.2500    0.2000    0.1667    0.1429    0.1250
    0.2000    0.1667    0.1429    0.1250    0.1111
```

6.2 Zeri di funzione

6.2.1 Grafici di funzione

In MATLAB una funzione $f(x)$ viene memorizzata come un vettore. In particolare, il vettore y ottenuto valutando f nel vettore delle ascisse x. Per cui la rappresentazione della funzione $f(x)$ è di fatto la rappresentazione del vettore y contro il vettore x.

Per introdurre i concetti di funzione e grafici di funzione, si presentano qua di seguito alcuni esempi di caso d'uso.

Definire le seguenti variabili:

- x***: vettore di estremi*** 0 ***e*** 10 ***con passo*** 0.1
- $y = e^x + 1$

Il vettore delle ascisse x può essere costruito banalmente con il seguente costrutto:

```
x = [0 : 0.1 : 10];
```

Per quanto riguarda la **funzione**, si utilizza la keyword per indicare che f ha come input un valore (`x`) e rappresenta la funzione `exp(x)+1`. In questo caso, la funzione si dice anonima. Per dichiarare funzioni esplicite, si rimanda alla documentazione ufficiale.

```
f = @(x) exp(x) + 1
```

Una volta definita una **funzione**, per **valutarla in uno o più punti**, si utilizzerà banalmente la sintassi matematica:

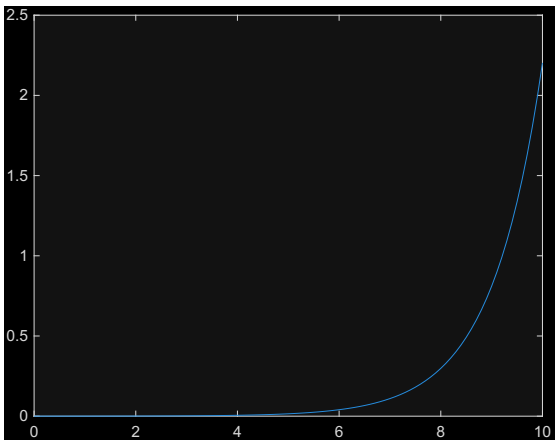
```
f(2)

ans =

    8.3891

f(0:3)

ans =

    2.0000    3.7183    8.3891   21.0855
```

Da notare che se l'argomento è un vettore, allora il risultato sarà un vettore della medesima lunghezza del vettore dato in input.

Utilizzando le variabili precedentemente definite, disegnare il grafico della funzione $y = e^x + 1$ ***nell'intervallo*** $[0, 10]$**.**

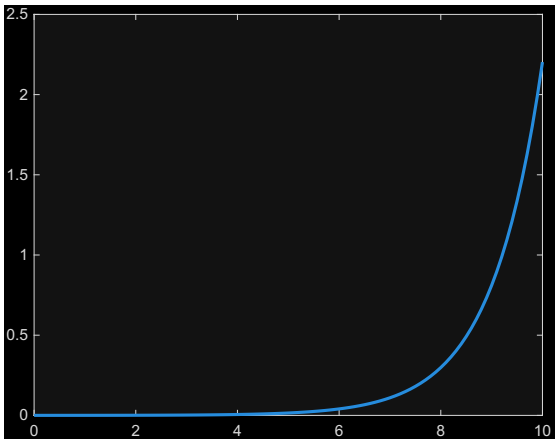
Per disegnare il grafico si utilizza il comando `plot`. Di default questa funzione disegna i valori in un piano cartesiano usando segmenti rettilinei (retta spezzata):

```
y = f(x);
plot(x, y)
```



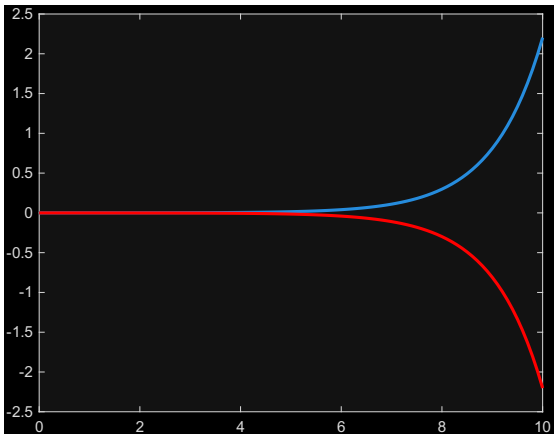
Per evitare che MATLAB sovrascriva la figura nella finestra aperta, è possibile numerarle usando la funzione `figure` (e.g. `figure(1); plot(x,y); figure(2); plot(0:3, 0:3)`).

La funzione `plot` accetta determinati valori per modificare il grafico finale. Nella documentazione ufficiale è possibile trovare l'intera lista e alcuni esempi. Scrivendo `plot(x, f(x), 'linewidth', 2)`, il parametro `'linewidth'` consente di definire lo spessore delle curve. Il valore che viene specificato in questo caso è 2 e il risultato:



Usando il comando `hold on` per fare un confronto tra i vari grafici e invocando di nuovo la funzione `plot` ma con parametri differenti, si ottiene:

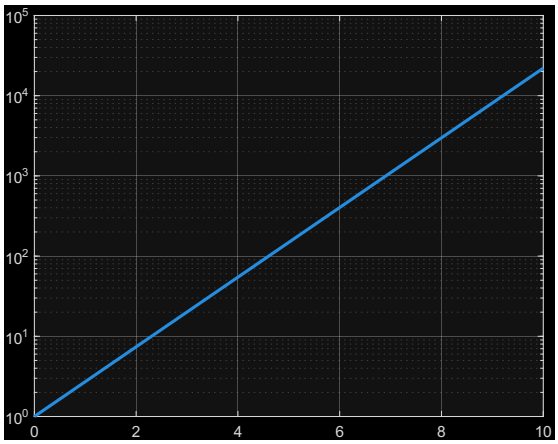
```
figure(1)
plot(x, f(x), 'linewidth', 2)
hold on
plot(x, -f(x), 'r', 'linewidth', 2)
```



Disegnare il grafico in scala semi-logaritmica (logaritmica solo per le ordinate) della funzione $y = e^x$ nell'intervallo $[0, 10]$. È possibile prevedere come sarà il grafico in scala semi-logaritmica della funzione $y = e^{2x}$? Verificare la risposta tracciando sulla medesima finestra le due funzioni utilizzando colori diversi per i due grafici.

Per disegnare il grafico in scala semi-logaritmica (logaritmica sulle ordinate) si utilizza il comando `semilogy` e si aggiunge anche la griglia:

```
semilogy(x, exp(x), 'linewidth', 2)
grid on
```

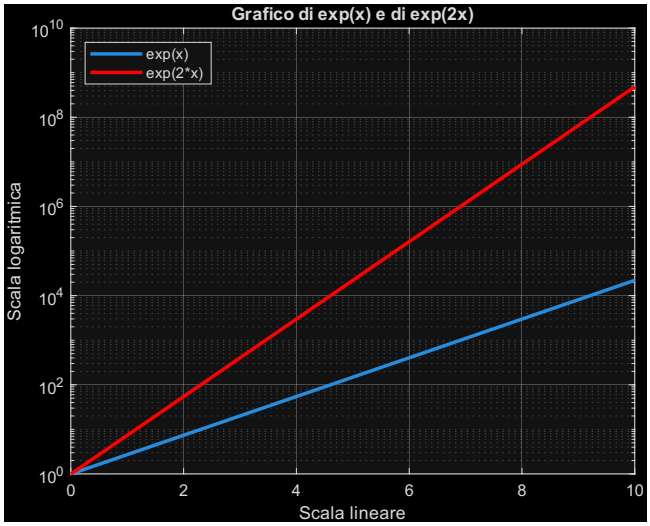


È una retta poiché $\log_{10}(y) = \log_{10}(e^x) = x \log_{10}(e)$.

- Il comando `semilogy` è l'equivalente di `plot` ma traccia un **grafico con l'asse delle ordinate in scala logaritmica**.
- Il comando `semilogx` traccia un **grafico con l'asse delle ascisse logaritmico**.
- Il comando `loglog` traccia un grafico in cui entrambi gli assi sono in scala logaritmica.

Passando alla risoluzione dell'esercizio, dato che $\log_{10}\left(e^{2x}\right) = 2x\log_{10}(e)$, disegnando in scala semi-logaritmica la funzione $y = e^{2x}$, si otterrà una retta con pendenza doppia rispetto alla retta precedentemente disegnata.

```
hold on
semilogy(x, exp(2 * x), 'r', 'linewidth', 2)
% oppure in un solo comando senza usare hold on
% semilogy(x, exp(x), 'b', x, exp(2*x), 'r', 'linewidth', 2)
title('Grafico di exp(x) e di exp(2x)')
xlabel('Scala lineare')
ylabel('Scala logaritmica')
grid on
legend('exp(x)', 'exp(2*x)', 'Location', 'NorthWest')
```



Il comando `legend` attribuisce alle curve disegnate da `plot` le stringhe di testo che gli vengono passate. Attenzione che alcune stringhe, come `'Location'` e `'NorthWest'`, vengono interpretate dalla funzione come comandi veri e propri. In questo caso si chiede di inserire una legenda in alto a sinistra.

6.3 Risoluzione di Sistemi di Equazioni Lineari

6.3.1 Metodi diretti

Si consideri la matrice di dimensione $n \times n$:

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & -1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & -1 & 0 & \cdots & 0 \\ \vdots & 0 & \ddots & \ddots & & \vdots \\ \vdots & \vdots & & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & -1 \end{bmatrix}$$

E $\mathbf{b}$ il vettore di dimensione n:

$$\mathbf{b} = [2, 0, 0, \ldots, 0]^T$$

1. Si ponga $n = 20$ e si assegnino in MATLAB la matrice A e il vettore dei termini noti $\mathbf{b}$.

```
n = 20;
% crea il vettore colonna composto da soli uno
R = ones(n, 1);
% crea una matrice di valori negativi (-1)
% sulla diagonale principale
% e sommala alla matrice creata come:
% - vettore R specificando il range per evitare
%   una matrice troppo grande;
% - -1 per indicare un livello sotto la diagonale principale:
A = -diag(R) + diag(R(1:n-1), -1);
% si riempi la prima riga della matrice A di uni
A(1, :) = 1;
% si crea un altro vettore di zeri
b = zeros(n, 1);
% e si sostituisce il primo valore con un 2
b(1) = 2;
```

   La funzione `diag` ha un parametro particolare, vedi la documentazione.

2. Si calcoli la fattorizzazione LU della matrice A, mediante la funzione MATLAB `lu`. Verificare che la tecnica del pivoting non è stata usata in questo caso.

```
% la funzione lu puo' essere utilizzata nel seguente modo
[L, U, P] = lu(A);
% in cui L ed U sono la fattorizzazione,
% mentre la matrice P indica la matrice permutazione.
% quest'ultima potrebbe avere delle permutazioni sulle righe
% dovute alla tecnica del pivoting.
% per controllare si puo' procedere controllando manualmente,
% quindi stampando la matrice P e controllare che sia
% una matrice identita',
% oppure invocare la funzione eye e verificare che siano
% uguali con un if statement
if P == eye(n)
    disp('pivoting effettuato!');
end
```

   Si veda a pagina 22 la spiegazione della matrice di permutazione.

3. Scrivere una funzione MATLAB `fwsub.m` che, dati in ingresso una matrice triangolare inferiore $L \in \mathbb{R}^{n \times n}$ e un vettore $\mathbf{f} \in \mathbb{R}^n$ restituisca in uscita il vettore $\mathbf{x}$, soluzione del sistema $L\mathbf{x} = \mathbf{f}$, calcolata mediante l'algoritmo della sostituzione in avanti (*forward substitution*). L'intestazione della funzione sarà ad esempio: `[x] = fwsub(L, f)`.

   Analogamente, scrivere la funzione `bksub.m` che implementi l'algoritmo della sostituzione all'indietro (*backward substitution*) per matrici triangolari superiori (U). Per controllare che le matrici L e U passate a `fwsub.m` e `bksub.m` siano effettivamente triangolari, è possibile utilizzare i comandi MATLAB `triu` e `tril` che, data una matrice, estraggono rispettivamente la matrice triangolare superiore e la matrice triangolare inferiore.

   Per creare una funzione, in MATLAB viene utilizzata la seguente sintassi:

```
function output_params = function_name(input_params)
    % Statements
end
```

   Introdotta la sintassi, si introduce il codice della funzione `fwsub.m`:

```
% si dichiara la funzione fwsub, che ha come input A e b
% e restituisce come output x
function x = fwsub(A,b)

    % ~ algoritmo di sostituzione in avanti ~
    % A: matrice quadrata triangolare inferiore
    % b: termine noto
    % x: soluzione del sistema Ax=b

    % si controlla che la matrice sia quadrata e
    % per farlo si calcola le dimensioni di b
    n = length(b);
    % se il numero di righe (size(A,1)) della matrice A
    % o se il numero di colonne (size(A,2)) della matrice A
    % sono diverse da n, allora le dimensioni non sono ammesse
    if (size(A, 1) ~= n || size(A, 2) ~= n)
        error("Dimensioni non ammesse");
    end

    % inoltre si controlla che la matrice sia una matrice
    % triangolare inferiore;
    % si utilizza la funzione tril per ottenere la
    % matrice triangolare inferiore
    if (A ~= tril(A))
        error("La matrice non e' triangolare inferiore");
    end

    % infine, si controllare che la matrice sia NON singolare,
    % ovvero che il determinante deve essere diverso da zero
    if (det(A) == 0)
        error("La matrice e' singolare");
    end

    % adesso l'algoritmo puo' iniziare;
    % si inizializza una matrice risultato, ovvero x,
    % nella quale verranno salvati i risultati
    x = zeros(n,1);
    % si applica la formula della sostituzione in avanti,
    % prima per la posizione (1,1)
    x(1) = b(1) / A(1,1);
    % e dopodiche' per tutte le posizioni della matrice
```

```
    for i = 2:n
        x(i) = (b(i) - A(i, 1:i-1) * x(1:i-1)) / A(i,i);
    end
```

Analogamente, si presenta il codice della funzione `bksub.m`:

```
% la funzione bksub avra' la stessa signature della funzione
% fwsub, ma la formula chiaramente sara' differente
function x = bksub(A, b)

    % ~ algoritmo di sostituzione all'indietro ~
    % A: matrice quadrata triangolare superiore
    % b: termine noto
    % x: soluzione del sistema Ax = b

    % 1. si esegue lo stesso controllo della funzione
    %    fwsub, si verifica che A sia quadrata
    n = length(b);
    if (size(A, 1) ~= n || size(A, 2) ~= n)
        error("Dimensioni non ammesse");
    end

    % 2. si controlla che sia effettivamente una
    %    matrice triangolare superiore,
    %    usando questa volta la funzione triu
    if (A ~= triu(A))
        error("La matrice non e' triangolare superiore");
    end

    % 3. l'ultimo controllo riguarda la "non singolarita'"
    %    ovvero, determinante diverso da zero
    if (det(A) == 0)
        error("La matrice e' singolare");
    end

    % 4. si parte con l'algoritmo e per farlo si inizia
    %    con l'ultima posizione;
    %    ovviamente si inizializza la matrice risultato
    x = zeros(n,1);
    x(n) = b(n) / A(n,n);
    % 5. si ricorda la sintassi del for statement
    %    initVal : step : endVal
    %    l'indice parte con un valore uguale a initVal,
    %    incrementa o decrementa a seconda dello step,
    %    termina quando raggiunge la condizione endVal
    for i = n-1 : -1 : 1
        x(i) = (b(i) - A(i, i+1:n) * x(i+1:n)) / A(i,i);
    end
```

4. Risolvere numericamente, utilizzando le funzioni `fwsub.m` e `bksub.m` implementate al punto precedente, i due sistemi triangolari necessari per ottenere la soluzione del sistema di partenza $A\mathbf{x} = \mathbf{b}$ mediante la fattorizzazione LU.

   Si utilizza la tecnica del pivoting e l'equazione 27 a pagina 22:

```
y = fwsub(L, P*b);
x = bksub(U, y);
```

5. Si calcoli la norma 2 dell'errore relativo

$$||\mathbf{err_{rel}}|| = \frac{||\mathbf{x} - \widehat{\mathbf{x}}||}{||\mathbf{x}||}$$

E la norma 2 del residuo normalizzata:

$$||\mathbf{r}|| = \frac{||\mathbf{b} - A\widehat{\mathbf{x}}||}{||\mathbf{b}||}$$

Sapendo che la soluzione esatta è il vettore di componenti:

$$\mathbf{x}(i) = \frac{2}{n} \qquad i = 1, \ldots, n$$

Si commenti il risultato ottenuto basandosi sul valore del numero di condizionamento della matrice A (si utilizzino i comandi `norm` e `cond`).

Il comando `norm` è stato spiegato a pagina 92.

```
x_ex = 2 / n * ones(n, 1);

err_rel = norm(x_ex - x) / norm(x_ex)
% il risultato: 5.1554e-16

r_nor = norm(b - A*x) / norm(b)
% il risultato: 2.7318e-16

K = cond(A)
% il risultato: 28.4998
```

6. Si ripeta il punto precedente per $n = 10, 20, 40, 80, 160$. Si rappresentino su un grafico in scala semi-logaritmica gli andamenti dell'errore relativo, del residuo normalizzato (si usa dire residuo normalizzato per la norma normalizzata del residuo) e del numero di condizionamento in funzione di n. Commentare il grafico ottenuto.

```
% si crea il vettore n
N = [10 20 40 80 160];
% e si inizializzano le variabili
K = [];
err_rel = [];
r_nor = [];

% per ogni n, si applicano i pezzi di codice precedenti
for n = N
    R = ones(n, 1);
    A = -diag(R) + diag(R(1:n-1), -1);
    A(1, :) = 1;

    b = zeros(n, 1);
    b(1) = 2;

    [L, U, P] = lu(A);

    y = fwsub(L, P*b);
    x_1 = bksub(U, y);

    x_ex = 2 / n * ones(n, 1);
    err_rel = [err_rel; norm(x_ex - x_1) / norm(x_ex)];
    r_nor = [r_nor; norm(b - A*x_1) / norm(b)];
```

```
25      K = [K; cond(A)];
26  end
27
28  % Semilog plot (y-axis has log scale)
29  semilogy(N, err_rel, '-s', N, r_nor, '-o', N, K, '-x')
30  legend('errore rel.', 'residuo norm.', 'n. di condizionamento'
        )
31  xlabel('dimensione n')
32  ylabel('err, r, K')
33  grid on
```

La seguente figura mostra l'andamento dell'errore relativo, del residuo normalizzato e del numero di condizionamento in funzione di n, in scala semi-logaritmica. Si noti che sia il residuo normalizzato sia l'errore relativo sono molto piccoli, dall'ordine di 10^{-16}, conseguenza del fatto che il numero di condizionamento $K(A)$ è in questo caso relativamente piccolo.

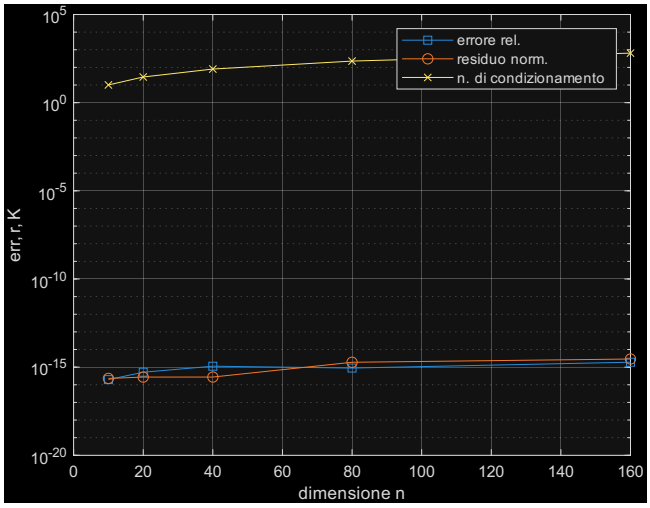


Figura 4: Andamento dell'errore relativo, del residuo normalizzato e del numero di condizionamento in funzione di n.

6.3.2 Metodi iterativi

I metodi iterativi stazionari sono considerati in genere nella seguente forma:

$$\mathbf{x}^{(k+1)} = B\mathbf{x}^{(k)} + \mathbf{f} \qquad k \geq 0$$

Dove B è detta matrice di iterazione. B e $\mathbf{f}$ identificano il metodo.

6.3.2.1 Metodo di Jacobi

Si consideri la matrice diagonale D degli elementi diagonali di A. Tale matrice è facilmente invertibile, se gli $a_{ii} \neq 0, i = 1, \ldots, n$, in quanto:

$$D = \begin{pmatrix} a_{11} & 0 & 0 & \cdots & 0 \\ 0 & a_{22} & 0 & \cdots & 0 \\ \vdots & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ 0 & 0 & \cdots & 0 & a_{nn} \end{pmatrix} \Longrightarrow D^{-1} = \begin{pmatrix} \frac{1}{a_{11}} & 0 & 0 & \cdots & 0 \\ 0 & \frac{1}{a_{22}} & 0 & \cdots & 0 \\ \vdots & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ 0 & 0 & \cdots & 0 & \frac{1}{a_{nn}} \end{pmatrix}$$

E il metodo può essere scritto direttamente in forma matriciale:

$$\mathbf{x}^{(0)} \text{ assegnato}$$
$$\mathbf{x}^{(k+1)} = B_J\mathbf{x}^{(k)} + \mathbf{f}_J$$

Dove $B_J = I - D^{-1}A = D^{-1}(D - A)$ è la matrice di iterazione di Jacobi e $\mathbf{f}_J = D^{-1}\mathbf{b}$

6.3.2.2 Metodo di Gauss-Seidel

Questo metodo si differenza dal metodo di Jacobi per il fatto che considera, oltre alla matrice D, anche le due matrici $-E$ e $-F$ triangolari superiore e inferiore della matrice A, ovvero:

$$-E = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ a_{21} & 0 & 0 & \cdots & 0 \\ \vdots & a_{32} & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ a_{n1} & a_{n2} & \cdots & a_{nn-1} & 0 \end{bmatrix} \qquad -F = \begin{bmatrix} 0 & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & 0 & a_{23} & \cdots & a_{2n} \\ \vdots & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & a_{n-1n} \\ 0 & 0 & \cdots & 0 & 0 \end{bmatrix}$$

Dunque, il seguente algoritmo o le seguenti istruzioni:

$$\mathbf{x}^{(0)} \text{ assegnato}$$
$$\mathbf{x}^{(k+1)} = B_{GS}\mathbf{x}^{(k)} + \mathbf{f}_{GS}$$

Dove $B_{GS} = (D - E)^{-1} F$ è la matrice d'iterazione di Gauss-Seidel e $\mathbf{f}_{GS} = (D - E)^{-1} \mathbf{b}$.

6.3.2.3 Esercizio

Si considerino la matrice:

$$A = \begin{bmatrix} 9 & -3 & 1 & & & & \\ -3 & 9 & -3 & 1 & & & \\ 1 & -3 & 9 & -3 & 1 & & \\ & 1 & -3 & 9 & -3 & 1 & \\ & & 1 & -3 & 9 & -3 & 1 \\ & & & 1 & -3 & 9 & -3 \\ & & & & 1 & -3 & 9 \end{bmatrix}$$

E il termine noto:

$$\mathbf{b} = \begin{bmatrix} 7 & 4 & 5 & 5 & 5 & 4 & 7 \end{bmatrix}^T$$

1. Costruire la matrice A (utilizzando i comandi Matlab `diag` e `ones`) e determinare il numero di elementi non nulli tramite il comando `nnz`. La matrice A è a dominanza diagonale per righe? È simmetrica e definita positiva?

   È stato richiesto di utilizzare i comandi `diag` e `ones` per costruire la matrice A. Quindi per farlo, si controlla rapidamente la documentazione dei due comandi:

```
>> help ones
 ones - Create array of all ones
    This MATLAB function returns the scalar 1.

    Syntax
      X = ones
      X = ones(n)
      X = ones(sz1,...,szN)
      X = ones(sz)

      X = ones(___,typename)
      X = ones(___,'like',p)

    Input Arguments
      n - Size of square matrix
        integer value
      sz1,...,szN - Size of each dimension
        two or more integer values
      sz - Output size
        row vector of integer values
      typename - Output class
        'double' (default) | 'single' | 'logical' | 'int8' | '
    uint8' | ...
      p - Prototype
        variable

>> help diag
 diag - Create diagonal matrix or get diagonal elements of
    matrix
    This MATLAB function returns a square diagonal matrix with
     the elements
    of vector v on the main diagonal.

    Syntax
      D = diag(v)
      D = diag(v,k)
```

```
34
35      x = diag(A)
36      x = diag(A,k)
37
38    Input Arguments
39      v - Diagonal elements
40        vector
41      A - Input matrix
42        matrix
43      k - Diagonal number
44        integer
```

Adesso si osservi la matrice A. È possibile notare che la diagonale principale ha tutti i valori uguale a 9, mentre sopra e sotto la diagonale principale, altre due diagonali con valore pari a -3 e allo stesso modo due diagonali con valore uguale a 1.

Usando entrambi i comandi, si può giungere al seguente risultato parziale:

```
>> diag(9*ones(1, n)) + diag(-3*ones(1,n-1), 1) + diag(1*ones
    (1,n-2), 2)

ans =

         9    -3     1     0     0     0     0
         0     9    -3     1     0     0     0
         0     0     9    -3     1     0     0
         0     0     0     9    -3     1     0
         0     0     0     0     9    -3     1
         0     0     0     0     0     9    -3
         0     0     0     0     0     0     9
```

Ed eseguendo con la stessa logica anche sotto la diagonale principale, si ottiene la matrice A richiesta:

```
n = 7;
A = diag(9*ones(1, n)) + ... % diagonale principale
    diag(-3*ones(1,n-1), 1) + diag(-3*ones(1,n-1), -1) + ...
    diag(1*ones(1,n-2), 2) + diag(1*ones(1,n-2), -2)

% ans =
%
%      9    -3     1     0     0     0     0
%     -3     9    -3     1     0     0     0
%      1    -3     9    -3     1     0     0
%      0     1    -3     9    -3     1     0
%      0     0     1    -3     9    -3     1
%      0     0     0     1    -3     9    -3
%      0     0     0     0     1    -3     9
```

Il numero di elementi non nulli si calcola con il comando `nnz`:

```
>> help nnz
 nnz - Number of nonzero matrix elements
    This MATLAB function returns the number of nonzero
    elements in matrix X.

    Syntax
      N = nnz(X)

    Input Arguments
      X - Input matrix
```

```
        matrix

>> nnz(A)

ans =

    29
```

E infine, per confermare che la matrice sia a dominanza diagonale per righe, simmetrica e definita positiva:

```
% applicando la definizione
n = 7;
A = diag(9*ones(1, n)) + ... % diagonale principale
    diag(-3*ones(1,n-1), 1) + diag(-3*ones(1,n-1), -1) + ...
    diag(1*ones(1,n-2), 2) + diag(1*ones(1,n-2), -2);
A_diag = diag(abs(A));
A_no_diag = diag(-3*ones(1,n-1), 1) + ...
            diag(-3*ones(1,n-1), -1) + ...
            diag(1*ones(1,n-2), 2) + diag(1*ones(1,n-2), -2);
% definizione:
% https://it.wikipedia.org/wiki/Matrice_a_diagonale_dominante
for index = n
    if not(A_diag(index) >= sum(abs(A_no_diag(index:index, 1:n
    ))))
        error("La matrice non e' a diagonale dominante")
    end
end

% oppure in modo piu' rapido:
if (2 * abs(diag(A)) - sum(abs(A), 2) > 0)
    disp("La matrice e' a diagonale dominante")
end

% una matrice e' simmetrica se e' uguale alla sua transposta
if not(A == transpose(A))
    disp("La matrice non e' simmetrica")
end

% il metodo piu' efficiente per controllare se una matrice
% e' simmetrica e definita positiva, e' con l'utilizzo
% della fattorizzazione di Cholesky
% (studiata nella parte di teoria)
% source: https://rb.gy/uko7gs
try chol(A);
    disp("La matrice e' simmetrica e definita positiva")
catch ME
    error("La matrice non e' simmetrica definita positiva")
end
```

2. Si calcolino le matrici di iterazione:

$$B_J = D^{-1}(D - A)$$
$$B_{GS} = (D - E)^{-1} F$$

Associate rispettivamente ai metodi di Jacobi e Gauss-Seidel e i relativi raggi spettrali. La condizione necessaria e sufficiente per la convergenza del metodo iterativo è soddisfatta in entrambi i casi?

Le matrici di iterazione dei due metodi si calcolano a partire dalla definizione:

```
D = diag(diag(A));
Bj = D\(D-A);   % matrice di iterazione di Jacobi

E=-tril(A,-1);
F=-triu(A,1);
Bgs=(D-E)\F; % matrice di iterazione di Gauss-Seidel

rho_j = max(abs(eig(Bj)))
rho_gs = max(abs(eig(Bgs)))
```

Si noti l'istruzione `D = diag(diag(A))`; il comando interno estrae la diagonale principale di A, restituendo un vettore, il quale viene elaborato dal comando più esterno che crea una seconda matrice quadrata identica alla dimensione di A ma con solo la diagonale principale.

Dal calcolo del raggio spettrale delle matrici si può concludere che in questo caso entrambi i metodi convergono, in quanto l'autovalore massimo risulta in modulo strettamente minore di 1. Si osservi che il raggio spettrale della matrice di iterazione del metodo di Gauss-Seidel è più basso di quello della matrice del metodo di Jacobi.

3. Scrivere la funzione Matlab che implementi il metodo di Jacobi inversione *matriciale* per il sistema lineare $A\mathbf{x} = \mathbf{b}$. L'intestazione della funzione sarà la seguente:

$$\texttt{[x,k] = jacobi(A,b,x0,toll,nmax).}$$

Il processo iterativo si arresta quando:

$$\frac{||\mathbf{r}^{(k)}||}{||\mathbf{b}||} \leq \texttt{toll}$$

(criterio d'arresto del residuo normalizzato).

```
function [x,k]=jacobi(A,b,x0,toll,nmax)

% Metodo di Jacobi
%
% A: matrice del sistema
% b: termine noto
% x0: vettore iniziale
% toll: tolleranza sul residuo normalizzato
% nmax: massimo numero di iterazioni
%
% x: soluzione ottenuta
% k: numero di iterazioni effettuate

n = size(b,1);

% Controlliamo che la matrice A sia quadrata e che,
% insieme al guess iniziale x0,
% abbia dimensioni compatibili con b.
if ((size(A,1) ~= n) || (size(A,2) ~= n) || (size(x0,1) ~= n))
  error('Dimensioni incompatibili')
end

% Controlliamo che la matrice A non abbia elementi
% diagonali nulli.
if (prod(diag(A)) == 0)
```

```
  error('res_normore: elementi diagonali nulli')
end

% Estraiamo la matrice D da A e calcoliamo la matrice
% d'iterazione e il termine noto g
D = diag(diag(A));
Bj = eye(n) - D\A;
g = D\b;

%Inizializziamo x come x0, calcoliamo il residuo
% e l'res_normore normalizzato
x = x0;
r = b - A*x;
res_norm = norm(r) / norm(b);

% Inizializziamo l'indice d'iterazione
k = 0;

while (res_norm > toll && k < nmax)
    k = k + 1;

    % Calcoliamo il nuovo x
    x=Bj*x+g;

    % Calcoliamo residuo e res_normore
    r = b - A*x;
    res_norm = norm(r)/norm(b);
end
```

4. Scrivere una funzione Matlab che implementi il metodo di Gauss-Seidel inversione *matriciale* per il sistema lineare $A\mathbf{x} = \mathbf{b}$. L'intestazione della funzione sarà la seguente:

`[x,k] = gs(A,b,x0,toll,nmax).`

```
function [x,k]=gs(A,b,x0,toll,nmax)

% Metodo di Gauss-Seidel
%
% A: matrice del sistema
% b: termine noto
% x0: vettore iniziale
% toll: tolleranza sul residuo normalizzato
% nmax: massimo numero di iterazioni
%
% x: soluzione ottenuta
% k: numero di iterazioni effettuate

n = size(b,1);

% Controlliamo che la matrice A sia quadrata e che,
% insieme al guess iniziale x0,
% abbia dimensioni compatibili con b.
if (( size(A,1)~=n) || (size(A,2)~=n) || (size(x0,1) ~= n) )
  error('dimensioni incompatibili')
end

% Controlliamo che la matrice A non abbia
% elementi diagonali nulli.
if (prod(diag(A)) == 0)
    error('errore: elementi diagonali nulli')
```

```
end

% Decomponiamo la matrice in D,E e F,
% e calcoliamo la matrice d'iterazione e il termine g
D=diag(diag(A));
E=-tril(A,-1);
F=-triu(A,1);
Bgs=(D-E)\F;
g=(D-E)\b;

% Inizializziamo x come x0, calcoliamo il residuo
% e l'errore normalizzato
x = x0;
r = b - A * x;
err = norm(r) / norm(b);

% Inizializziamo l'indice d'iterazione
k = 0;

while ( err > toll && k < nmax )
  k = k + 1;

  % Calcoliamo il nuovo x
  x=Bgs*x+g;

  % Calcoliamo residuo e errore
  r = b - A*x;
  err = norm(r)/norm(b);
end
```

5. Costruire il termine noto $\mathbf{b}$. Utilizzando le funzioni costruite nei punti 3 e 4, risolvere il sistema $A\mathbf{x} = \mathbf{b}$ ponendo $x^{(0)} = [0, 0, \ldots, 0]^T$, `toll` $= 10^{-6}$ e `nmax` $= 1000$. Confrontare il numero di iterazioni necessarie per arrivare a convergenza per i due metodi e commentare i risultati ottenuti.

   Il metodo di Gauss-Seidel converge più velocemente alla soluzione esatta in accordo con il corrispondente raggio spettrale che è più basso di quello della matrice del metodo di Jacobi.

```
b = transpose([7 4 5 5 5 4 7]);
toll = 1e-6;
x0 = zeros(n, 1);
nmax = 1000;

[xJ, kJ] = jacobi(A, b, x0, toll, nmax);
[xGS, kGS] = gs(A, b, x0, toll, nmax);


xJ
kJ
xGS
kGS

% xJ =
%
%      1.0000
%      1.0000
%      1.0000
%      1.0000
%      1.0000
%      1.0000
```

```
%      1.0000
%
%
% kJ =
%
%      49
%
%
% xGS =
%
%      1.0000
%      1.0000
%      1.0000
%      1.0000
%      1.0000
%      1.0000
%      1.0000
%
%
% kGS =
%
%      12
```

6.3.2.4 Metodo di Richardson

Il metodo di Richardson stazionario è basato sulla seguente legge. Dati $\mathbf{x}^{(0)}$ e $\alpha \in \mathbb{R}$ si calcola:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha \mathbf{r}^{(k)} \qquad k \geq 0 \tag{153}$$

Dove $\alpha \neq 0$ è un parametro costante per ogni iterazione. Questo metodo richiede, ad ogni passo k, di calcolare il *residuo* $\mathbf{r}^{(k)}$ definito come:

$$\mathbf{r}^{(k)} = \mathbf{b} - A\mathbf{x}^{(k)}$$

Il metodo di Richardson *stazionario* converge solo per $0 < \alpha < \frac{2}{\lambda_{\max}}$, in cui $\lambda_{\max}$ è il massimo degli autovalori della matrice A. Inoltre, è possibile calcolare un valore di α ottimale che massimizza la velocità di convergenza. Questo valore è dato da:

$$\alpha_{\text{opt}} = \frac{2}{\lambda_{\min} + \lambda_{\max}}$$

In cui $\lambda_{\min}$ è il minimo degli autovalori della matrice A. Per questo valore, la velocità di convergenza è data da:

$$\rho_{\text{opt}} = \frac{K(A) - 1}{K(A) + 1}$$

Dove $K(A)$ è il numero di condizionamento, definito anche come:

$$K(A) = \left|\left|A^{-1}\right|\right| \cdot ||A||$$

Dove $||\cdot||$ è una opportuna norma introdotta per la matrice. Vale sempre $K(A) \geq 1$. Se la matrice A è simmetrica e definita positiva, utilizzando la sua norma 2, vale:

$$K(A) = \left|\left|A^{-1}\right|\right|_2 \cdot ||A||_2 = \frac{\lambda_{\max}}{\lambda_{\min}}$$

Si noti che, dalla sua definizione, il metodo di Richardson (eq. 153) può essere riscritto nella seguente forma (utilizzando la definizione di $\mathbf{r}$):

$$\mathbf{x}^{(k+1)} = (I - \alpha A)\,\mathbf{x}^{(k)} + \alpha \mathbf{b} \qquad k \geq 0 \tag{154}$$

Segue che il metodo di Richardson è un metodo iterativo caratterizzato dalla matrice di iterazione $B_\alpha = I - \alpha A$ e da $\mathbf{f} = \alpha \mathbf{b}$.

6.3.2.5 Precondizionamento

Il numero di condizionamento di una matrice governa il rapporto tra l'errore relativo commesso dalla soluzione numerica nella risoluzione di un sistema lineare e il corrispondente residuo normalizzata, alla iterata k:

$$\frac{\left|\left|\mathbf{x}^{(k)} - \mathbf{x}\right|\right|}{\left|\left|\mathbf{x}^{(k)}\right|\right|} \leq K(A) \frac{\left|\left|\mathbf{r}^{(k)}\right|\right|}{||\mathbf{b}||}$$

Dove:

- $\mathbf{x}^{(k)}$ è la soluzione numerica
- $\mathbf{r}^{(k)}$ è il residuo, ovvero $\mathbf{b} - A\mathbf{x}^{(k)}$
- $\mathbf{x}$ è la soluzione esatta del sistema lineare

Inoltre, nel caso del metodo di Richardson, si ottiene:

$$\left|\left|\mathbf{e}^{(k+1)}\right|\right| \leq \frac{K(A) - 1}{K(A) + 1} \cdot \left|\left|\mathbf{e}^{(k)}\right|\right|$$

Come migliore stima ottenibile usando α_{opt}.

Per problemi ben condizionati ($K(A)$ non molto più grande di 1), la soluzione del problema con piccoli residui non differisce molto dalla soluzione del problema originale; al contrario, in problemi con la matrice mal condizionata ($K(A) \gg 1$) a piccoli residui possono corrispondere grandi errori e la convergenza è molto lenta.

L'idea del precondizionamento consiste nel cercare di ridurre il numero di condizionamento della matrice del sistema, pre-moltiplicandola per una matrice P^{-1} (P è chiamata *precondizionatore*). Si ottiene così il sistema equivalente:

$$P^{-1}A\mathbf{x}^{(k)} = P^{-1}\mathbf{b}$$

Ovviamente il precondizionatore è efficace se $K\left(P^{-1}A\right) \ll K(A)$ e se la soluzione del sistema lineare in P che sorge ad ogni iterazione non è troppo onerosa. La prima proprietà è solitamente verificata quando $P^{-1} \approx A^{-1}$, cioè quando P e A hanno uno spettro simile, ma dovendo tener conto della seconda condizione è opportuno scegliere P con una struttura speciale che mantenga basso il costo computazionale (ad esempio diagonale o triangolare).

Ad esempio, nel caso di Richardson, per alleviare la dipendenza della convergenza ottimale dal numero di condizionamento, si introduce la tecnica di precondizionamento, che consiste nel sostituire A con $P^{-1}A$. Questo metodo richiede, ad ogni iterata, di trovare il cosiddetto *residuo precondizionato* $\mathbf{z}^{(k)}$ dato dalla soluzione del sistema lineare:

$$P\mathbf{z}^{(k)} = \mathbf{r}^{(k)} \tag{155}$$

Di conseguenza, la nuova iterata è definita da $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha\mathbf{z}^{(k)}$. Si noti che la matrice P, oltre ad essere non singolare (determinante diverso da zero),

simmetrica (coincide con la sua trasposta) e definita positiva, deve essere scelta in modo tale che il costo computazionale richiesto dalla risoluzione del sistema (eq. 155) sia basso. Per il caso precondizionato, valgono i precedenti risultati su α_{opt} e ρ_{opt} a patto che si considerino gli autovalori di $P^{-1}A$ invece di quelli di A.

6.3.2.6 Metodo del gradiente

È possibile generalizzare il metodo di Richardson precondizionato tramite l'introduzione di un parametro di accelerazione dinamico α_k:

$$P\left(\mathbf{x}^{(k+1)-\mathbf{x}^{(k)}}\right) = \alpha_k \mathbf{r}^{(k)} \qquad k \geq 0,\ \alpha_k \neq 0$$

Con P il precondizionatore. Lo scopo dell'utilizzo di α_k è quello di poter calcolare facilmente il parametro di accelerazione evitando il calcolo (spesso oneroso) degli autovalori di A come per α_{opt}. Questo metodo è detto *metodo di Richardson dinamico*. Se α_k è scelto in modo ottimale, allora il metodo è detto del *gradiente* se $P = I$, oppure del *gradiente precondizionato* se $P \neq I$.

A partire dal metodo di Richardson, è possibile ottenere i metodi del *gradiente* ($P = I$) e del *gradiente precondizionato* ($P \neq I$) tramite l'aggiunta del parametro α_k: dato $\mathbf{x}^{(0)}$ assegnato, si ponga $\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)}$ e per $k = 0, 1, \ldots$

$$\begin{aligned} P\mathbf{z}^{(k)} &= \mathbf{r}^{(k)} \\ \alpha_k &= \frac{\left(\mathbf{z}^{(k)}\right)^T \mathbf{r}^{(k)}}{\left(\mathbf{z}^{(k)}\right)^T A\mathbf{z}^{(k)}} \\ \mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} + \alpha_k \mathbf{z}^{(k)} \\ \mathbf{r}^{(k+1)} &= \mathbf{r}^{(k)} + \alpha_k A\mathbf{z}^{(k)} \end{aligned} \tag{156}$$

6.3.2.7 Esercizi su Richardson e gradiente

Si considerino la matrice:

$$A = \begin{bmatrix} 9 & -3 & 1 & & & & \\ -3 & 9 & -3 & 1 & & & \\ 1 & -3 & 9 & -3 & 1 & & \\ & 1 & -3 & 9 & -3 & 1 & \\ & & 1 & -3 & 9 & -3 & 1 \\ & & & 1 & -3 & 9 & -3 \\ & & & & 1 & -3 & 9 \end{bmatrix}$$

E il termine noto:

$$\mathbf{b} = \begin{bmatrix} 7 & 4 & 5 & 5 & 5 & 4 & 7 \end{bmatrix}^T$$

1. Costruire la matrice A (utilizzando i comandi Matlab `diag` e `ones`).

```
n = 7;
A = diag(9*ones(1, n)) + ... % diagonale principale
    diag(-3*ones(1,n-1), 1) + diag(-3*ones(1,n-1), -1) + ...
    diag(1*ones(1,n-2), 2) + diag(1*ones(1,n-2), -2)

% ans =
%
%       9    -3     1     0     0     0     0
%      -3     9    -3     1     0     0     0
%       1    -3     9    -3     1     0     0
%       0     1    -3     9    -3     1     0
%       0     0     1    -3     9    -3     1
%       0     0     0     1    -3     9    -3
%       0     0     0     0     1    -3     9
```

2. Calcolare l'intervallo di valori di α per cui il metodo di Richardson stazionario non precondizionato converge. Determinare il valore ottimale di α per avere massima velocità di convergenza.

```
% calcolo di eigenvalues e eigenvectors
% V: right eigenvectors (matrice quadrata)
% D: eigenvalues (matrice diagonale)
[V,D] = eig(A);

% si ottiene l'autovalore massimo
lambda_max = max(diag(D))
% lambda_max =
%
%    16.0403

% il limite (suppenendo lambda_min = 0)
Lim = 2/lambda_max
% Lim =
%
%     0.1247

% si calcola infine alpha opt
lambda_min = min(diag(D))
alpha_opt = 2/(lambda_min+lambda_max)
% lambda_min =
%
%     4.9042
%
```

```
%
% alpha_opt =
%
%     0.0955
```

3. Scrivere una funzione Matlab che implementi il metodo di Richardson stazionario non precondizionato per il sistema lineare $A\mathbf{x} = \mathbf{b}$. I parametri in ingresso richiesti dalla funzione sono la matrice A, il termine noto $\mathbf{b}$, il guess iniziale $\mathbf{x}^{(0)}$, il coefficiente α, la tolleranza per il criterio d'arresto `toll` e il numero massimo di iterazioni ammesse `nmax`. La funzione restituisce la soluzione numerica $\mathbf{x}$ e il numero di iterazioni effettuate k.

   L'intestazione della funzione sarà la seguente:

   $$[\text{x, k}] = \text{richardson(A, b, x0, alpha, toll, nmax)}$$

   Il processo iterativo si arresta quando:

   $$\frac{||\mathbf{r}^{(k)}||}{||\mathbf{b}||} \leq \texttt{toll}$$

   (criterio d'arresto del residuo normalizzato).

```
function [x, k, err] = richardson(A, b, x0, alpha, toll, nmax)

% Metodo di Richardson stazionario
%
% A: matrice del sistema
% b: termine noto
% x0: vettore iniziale
% alpha: coefficiente di Richardson
% toll: tolleranza sul residuo normalizzato
% nmax: massimo numero di iterazioni
%
% x: soluzione ottenuta
% it: numero di iterazioni effettuate

n = size(b,1);
k = 0;

if ((size(A,1) ~= n) || (size(A,2) ~= n) || (size(x0,1) ~= n))
  error('Dimensioni incompatibili')
end

x = x0;
r = b - A*x;
errk = norm(r) / norm(b);
err = errk;

while (errk > toll && k < nmax)
    k = k + 1;
    x = x + alpha*r;
    r = b - A*x;
    errk = norm(r)/norm(b);
    err = [err; errk];
end
```

4. Costruire il termine noto $\mathbf{b}$. Utilizzando la funzione scritta al punto precedente, determinare la soluzione del sistema lineare $A\mathbf{x} = \mathbf{b}$, con A data dal punto 1.

Si ponga $\mathbf{x}^{(0)} = [0, 0, \ldots, 0]^T$, `toll` $= 10^{-6}$ e `nmax` $= 1000$. Si verifichi sperimentalmente che il metodo di Richardson stazionario non precondizionato converge solo se α appartiene all'intervallo trovato nel punto 2. In particolare, si scelga un α al di fuori dell'intervallo e si verifichi che il metodo diverge. Si provi, inoltre, sperimentalmente, che per α_{opt} il metodo converge più velocemente. In particolare, si scelga un α nell'intervallo di convergenza e si osservi il numero di iterazioni per aggiungere a convergenza è maggiore di quello ottenuto utilizzando α_{opt}.

```
% calcolo della transposta
b = transpose([7 4 5 5 5 4 7]);

% si pongono i dati richiesti
x0 = zeros(n,1);
toll = 1e-6;
nmax = 1000;

% si utilizza prima un alpha fuori dal limite del punto 2
alpha = 2;
[xR1,kR1] = richardson(A,b,x0,alpha,toll,nmax)
% xR1 =
%
%   -Inf
%    Inf
%   -Inf
%    Inf
%   -Inf
%    Inf
%   -Inf
%
%
% kR1 =
%
%    208

% adesso ci si avvicina al valore alpha
alpha = 0.11;
[xR2,kR2] = richardson(A,b,x0,alpha,toll,nmax)
% xR2 =
%
%     1.0000
%     1.0000
%     1.0000
%     1.0000
%     1.0000
%     1.0000
%     1.0000
%
%
% kR2 =
%
%     45

% e infine si utilizza alpha opt per verificare
% che sia il migliore
[xR3,kR3] = richardson(A,b,x0,alpha_opt,toll,nmax)
% xR3 =
%
%     1.0000
%     1.0000
%     1.0000
```

```
%      1.0000
%      1.0000
%      1.0000
%      1.0000
%
%
% kR3 =
%
%      22
```

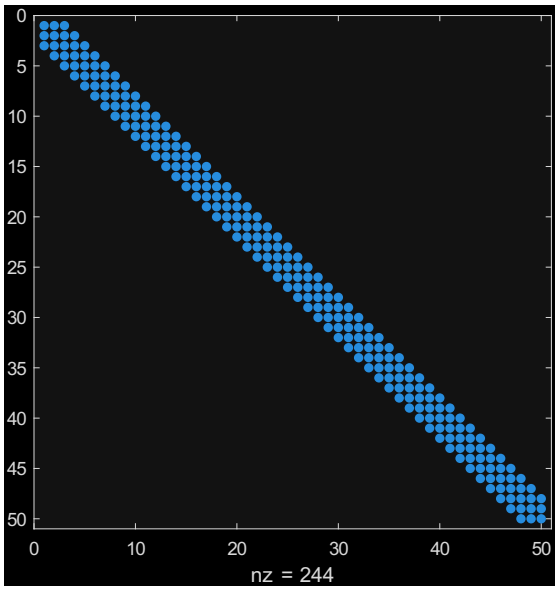
Adesso si passa al gradiente. Si consideri il problema lineare $A\mathbf{x} = \mathbf{b}$, dove la matrice $A \in \mathbb{R}^{n \times n}$ è pentadiagonale:

$$A = \begin{bmatrix} 4 & -1 & -1 & & & & \\ -1 & 4 & -1 & -1 & & & \\ -1 & 4 & -1 & -1 & -1 & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & -1 & -1 & 4 & -1 & -1 \\ & & & -1 & -1 & 4 & -1 \\ & & & & -1 & -1 & 4 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 0.2 \\ 0.2 \\ 0.2 \\ \vdots \\ 0.2 \\ 0.2 \\ 0.2 \end{bmatrix}$$

Si vuole risolvere tale problema con il metodo del gradiente, soddisfando una tolleranza di 10^{-5} per il criterio relativo al residuo normalizzato, a partire dal vettore iniziale $\mathbf{x}_0 = [0, \ldots, 0]^T$.

1. Costruire la matrice A con $n = 50$, il termine noto $\mathbf{b}$ ed il vettore soluzione iniziale $\mathbf{x}_0$. La matrice A è simmetrica e definita positiva? Se ne calcoli il numero di condizionamento, senza utilizzare il comando `cond` di Matlab.

```
n = 50;
A = diag(4*ones(n,1)) + ...
    diag(-ones(n-1,1),-1) + ....
    diag(-ones(n-1,1),1) + ...
    diag(-ones(n-2,1),-2) + ...
    diag(-ones(n-2,1),2);
% si visualizza la sparsity pattern della matrice
spy(A)
% si costruiscono i restanti vettori
b = 0.2*ones(n, 1);
x0 = zeros(n, 1);

% si controlla se e' simmetrica e definita positiva
% sfruttando la velocita' della fattorizzazione di Cholensky
try chol(A);
    disp("La matrice e' simmetrica definita positiva")
catch ME
    disp("La matrice non e' simmetrica definita positiva")
end
eigA = eig(A);
KA = max(eigA)/min(eigA)
% KA =
%
%   336.2412
```



2. Si scriva una funzione che implementi il metodo del gradiente e la si usi per risolvere il sistema lineare $A\mathbf{x} = \mathbf{b}$. L'intestazione della funzione sarà ad esempio:

```
[x, iter, err] = graddyn(A, b, x0, nmax, toll)
```

Dove `err` è il vettore contenente il residuo normalizzato ad ogni iterazione.

```
function [xn, iter, err] = graddyn (A, b, x0, nmax, tol)

% Metodo del gradiente per sistemi lineari
%
% Parametri di ingresso:
%   A       Matrice del sistema
%   b       Termine noto
%   x0      Vettore iniziale
%   nmax    Numero massimo di iterazioni
%   tol     Tolleranza sul test d'arresto
%
% Parametri in uscita
%   xn      Vettore soluzione
%   iter    Iterazioni effettuate
%   err     Vettore contenente gli errori relativi sul residuo

[n, m] = size (A);
if not(n == m)
  error ('matrice non quadrata')
end

iter = 0;
xn = zeros(n,1);

% Iterazioni
bnrm2 = norm (b);
r = b - A * x0;
errk = norm (r) / bnrm2;
err = errk;
```

```
xn = x0;

while (errk > tol) && (iter < nmax)
  z = r;
    transpose_z = transpose(z);
    alpha = transpose_z*r / (transpose_z*A*z);
    xn = xn + alpha * r;
    r = (eye(n)-alpha*A)*r;
    errk = norm (r) / bnrm2;
    err = [err errk];
%     xv = xn;
    iter = iter + 1;
end

if (iter == nmax)
    fprintf('Il metodo graddyn non converge in %d iterazioni \
    n', iter);
end
```

3. Utilizzare la funzione scritta al punto precedente per determinare la soluzione del sistema lineare $A\mathbf{x} = \mathbf{b}$.

```
% si impostano i valori
toll = 1e-5;
nmax = 10000;

% inizia il timer
tic
% si invoca il metodo del gradiente
[xGD, iterGD, errGD] = graddyn(A, b, x0, nmax, toll);
% si salva il tempo finale di esecuzione
timeGD=toc;
```

4. A partire dal `graddyn.m`, si scriva una funzione `gradprec.m` che implementi il metodo del gradiente precondizionato. L'intestazione della funzione sarà ad esempio:

   `[x, iter, err] = gradprec(A, b, x0, nmax, toll)`

   Si risolva il sistema lineare $A\mathbf{x} = \mathbf{b}$ applicando il metodo del gradiente precondizionato con il precondizionatore:

$$P = \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix}$$

```
function [xn, iter, err] = gradprec (A, b, P, x0, nmax, tol)

% Metodo del gradiente precondizionato
%
% Parametri di ingresso:
%   A       Matrice del sistema
%   b       Termine noto
%   P       Precondizionatore
%   x0      Vettore iniziale
```

```
%   nmax    Numero massimo di iterazioni
%   tol     Tolleranza sul test d'arresto
%
% Parametri in uscita
%   xn      Vettore soluzione
%   iter    Iterazioni effettuate
%   err     Vettore contenente gli errori relativi sul residuo

[n, m] = size (A);
if not(n == m)
  error ('matrice non quadrata')
end

iter = 0;
xn = zeros(n,1);

% Iterazioni
bnrm2 = norm (b);
r = b - A * x0;
errk = norm (r) / bnrm2;
err = errk;
xv = x0;

while (errk > tol) && (iter < nmax)
    z = P\r;
    transpose_z = transpose(z);
    alpha = transpose_z*r / (transpose_z*A*z);
    xn = xv + alpha * z;
    r = r-alpha*A*z;
    errk = norm (r) / bnrm2;
    err = [err errk];
    xv = xn;
    iter = iter + 1;
end

if (iter == nmax)
    fprintf('Il metodo gradprec non converge in %d iterazioni
    \n', iter);
end
```

```
P = diag(2*ones(n,1)) - ...
    diag(ones(n-1,1),-1) - ...
    diag(ones(n-1,1),1);

tic
[xPG, iterPG, errPG] = gradprec(A, b, P, x0, nmax, toll);
timePG=toc;
```

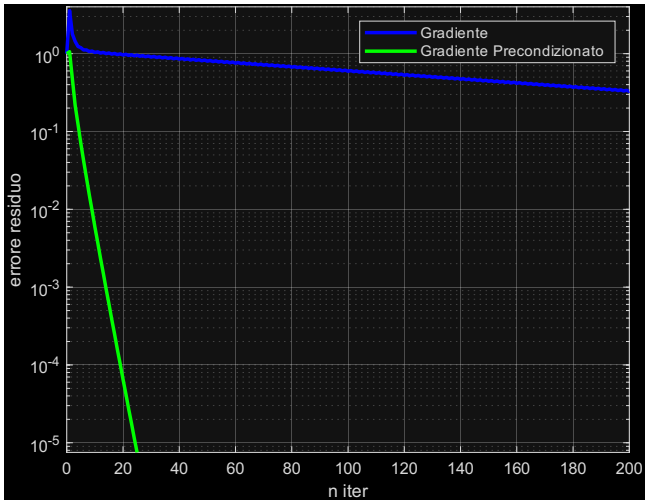
5. Disegnare su un grafico l'andamento del residuo normalizzato:

$$\frac{||\mathbf{r}^{(k)}||}{||\mathbf{b}||}$$

In funzione delle iterazioni k nei due casi (gradiente e gradiente precondizionato), e confrontare le curve ottenute.

```
figure
semilogy(...
    [0:iterGD], ...
    errGD, ...
    'b-', ...
```

```
    [0:iterPG], ...
    errPG, ...
    'g-', ...
    'Linewidth', ...
    2 ...
)
grid on
axis([0 200 0 4])
xlabel('n iter')
ylabel('errore residuo')
legend('Gradiente', 'Gradiente Precondizionato')
```



6. A partire da `richardson.m`, si scriva una funzione `richprec.m` che implementi il metodo di Richardson precondizionato. L'intestazione della funzione sarà ad esempio:

   ```
   [x, iter, err] = richprec(A, b, P, alpha, x0, nmax, toll)
   ```

   Si risolva il sistema lineare $A\mathbf{x} = \mathbf{b}$ applicando il metodo di Richardson precondizionato utilizzando la relativa α_{opt} e il precondizionatore P implementato al punto 4.

```
function [x, it, err] = richprec(A,b,P, alpha, x0,nmax, toll)

% Metodo di Richardson stazionario precondizionato
%
% A: matrice del sistema
% b: termine noto
% P: precondizionatore
% x0: vettore iniziale
% alpha: coefficiente di Richardson
% toll: tolleranza sul residuo normalizzato
% nmax: massimo numero di iterazioni
%
% x: soluzione ottenuta
% it: numero di iterazioni effettuate
% err: vettore contenente gli errori relativi sul residuo


```

```
n = size(b,1);
if ( ...
        not(size(A,1) == size (A,2)) || ...
        not(size(A,1) == n) || ...
        not(size(x0,1) == n) || ...
        not(size (P,2) == n) || ...
        not(size(P,1) == n) ...
)
    error('Dimesioni incompatibili')
end

it = 0;
x = x0;
r = b - A*x;
errk = norm(r)/norm(b);
err = errk;

% loop
while (it < nmax && errk > toll)
    it = it + 1;
    z = P\r;
    x = x + alpha*z;
    r = b - A*x;
    errk = norm(r)/norm(b);
    err = [err; errk];
end
```

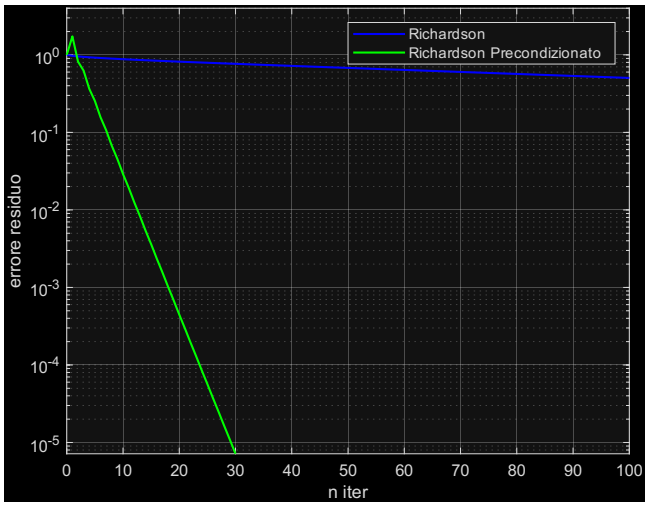
```
lambda_minRP = min(eig(P\A));
lambda_maxRP = max(eig(P\A));
alpha_optRP = 2/(lambda_maxRP + lambda_minRP);
tic
[xRP, iterRP, errRP] = richprec(A, b, P, alpha_optRP, x0, nmax
    , toll);
timeRP=toc;
```

7. Si risolva il sistema lineare $A\mathbf{x} = \mathbf{b}$ con il metodo di Richardson. Disegnare su un grafico l'andamento del residuo normalizzato:

$$\frac{||\mathbf{r}^{(k)}||}{||\mathbf{b}||}$$

In funzione delle iterazioni k nei due casi (Richardson e Richardson precondizionato), e confrontare le curve ottenute.

```
lambda_minR = min(eig(A));
lambda_maxR = max(eig(A));
alpha_optR = 2/(lambda_maxR + lambda_minR);

tic
[xR, iterR, errR] = richardson(A, b, x0, alpha_optR, toll,
    nmax);
timeR=toc;

figure
semilogy([0:iterR], errR, 'b-', [0:iterRP], errRP, 'g-', '
    Linewidth', 1.2)
grid on
axis([0 100 0 4])
xlabel('n iter')
ylabel('errore residuo')
legend('Richardson', 'Richardson Precondizionato')
```


Richardson
Richardson Precondizionato
errore residuo
n iter
10^0
10^-1
10^-2
10^-3
10^-4
10^-5
0
10
20
30
40
50
60
70
80
90
100

6.4 Sistema di equazioni non lineari

6.4.1 Metodo di Newton

Si consideri il problema della ricerca degli zeri del sistema non lineare $\mathbf{f}(\mathbf{x}) = \mathbf{0}$, dove $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$. Definendo:

- $\mathbf{x} = (x_1, \ldots, x_n)^T$
- $\mathbf{f} = (f_1, \ldots, f_n)^T$ con $f_1, \ldots, f_n$ funzioni $\mathbb{R}^n \to \mathbb{R}$

Il problema si può riscrivere nel seguente modo:

$$\begin{cases} f_1(x_1, \ldots, x_n) = 0 \\ f_2(x_1, \ldots, x_n) = 0 \\ \vdots \\ f_n(x_1, \ldots, x_n) = 0 \end{cases}$$

Il metodo di Newton per sistemi non lineari è il seguente. Dato $\mathbf{x}^{(0)} \in \mathbb{R}^n$, per $k \geq 0$:

1. Risolvere il sistema lineare:
$$J\left(\mathbf{x}^k\right) \delta\mathbf{x}^{(k)} = -\mathbf{f}\left(\mathbf{x}^{(k)}\right)$$
2. Ponendo:
$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \delta\mathbf{x}^{(k)}$$

Dove, per un generico punto $\mathbf{y}$, $J(\mathbf{y})$ è la matrice Jacobiana della funzione $\mathbf{f}$ e consiste in una matrice $\mathbb{R}^n \times \mathbb{R}^n$ le cui componenti sono:

$$J_{il}(\mathbf{y}) = \frac{\partial f_i(\mathbf{y})}{\partial y_l} \qquad i, l = 1, \ldots, n$$

Si osservi che:

1. L'applicazione del metodo di Newton richiede ad ogni iterazione la soluzione di un sistema lineare con matrice $A^{(k)} = J\left(\mathbf{x}^{(k)}\right)$.
2. È possibile dimostrare che se:
   (a) $\mathbf{x}^{(0)}$ è "sufficientemente" vicino alla soluzione α
   (b) $J\left(\mathbf{x}^{(k)}\right)$ è una matrice non singolare con $k = 0, 1, \ldots$

   Allora il metodo di Newton converge con ordine 2.

Si consideri il sistema non lineare seguente la cui incognite è il vettore $\mathbf{x} = [x_1, x_2]$:

$$\begin{cases} -x_1 + e^{3x_2} = 1 \\ -x_1 + x_1 x_2^2 = -2 \end{cases}$$

1. Scrivere Il sistema non lineare sotto la forma:

$$\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x})]^T = \mathbf{0}$$

Rappresentare $\mathbf{f}$ mediante una funzione Matlab di tipo *anonymous function* che, ricevuto in input un vettore colonna di 2 elementi, restituisce un vettore colonna di 2 elementi contenente la valutazione di $\mathbf{f}(\mathbf{x})$. Analogamente, calcolare la matrice Jacobiana $J(\mathbf{x})$ di $\mathbf{f}(\mathbf{x})$ e scrivere la *anonymous function* che restituisca tale matrice.

Dunque, il sistema da risolvere è:

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} -x_1 + e^{3x_2} - 1 \\ -x_1 + x_1 x_2^2 + 2 \end{bmatrix} = \mathbf{0}$$

Dove $\mathbf{x} = [x_1, x_2]^T$. La matrice Jacobiana è la seguente:

$$J(\mathbf{x}) = \frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \begin{bmatrix} -1 & 3e^{3x_2} \\ -1 + x_2^2 & 2x_1 x_2 \end{bmatrix}$$

Si introduce il vettore $\mathbf{f}$ e la matrice J utilizzando i seguenti comandi:

```
% il comando @ e' usato per creare funzioni
f = @(x)[-x(1)+exp(3*x(2))-1; -x(1)+x(1)*x(2)^2+2];
J = @(x)[-1 3*exp(3*x(2)); -1+x(2)^2 2*x(1)*x(2)];
```

2. Implementare il metodo di Newton per sistemi. L'intestazione della funzione sarà ad esempio:

```
[xvect, it] = newtonsys(fun, Jf, x0, toll, nmax)
```

Con ovvio significato dei parametri di ingresso e di uscita. Si utilizzi un criterio d'arresto basato sulla norma della differenza tra due iterate successive.

Suggerimento: si utilizzi il comando \ per la risoluzione dei sistemi lineari.

```
function [xvect, it] = newtonsys(fun, Jf, x0, toll, nmax)
%
% [xvect, nit] = newtonsys(fun, Jf, x0, toll, nmax)
%
% Metodo di Newton per sistemi non lineari di dimensione n
%
% Parametri di ingresso:
%
% fun funzione
% Jf matrice Jacobiana
% x0 vett. colonna di dimensione n contenente il dato iniziale
% toll Tolleranza sul test d'arresto
% nmax Numero massimo di iterazioni
%
% Parametri di uscita :
```

```
%
% xvect      Vett. contenente tutte le iterate calcolate
%            (l'ultima componente e' la soluzione)
% it         Iterazioni effettuate

it = 0;
err = toll+1;
xvect = x0;
%
while it < nmax && err >= toll
    f0 = fun(x0);
    Jf0 = Jf(x0);
    dx = -Jf0\f0; % -Jf(x0)\fun(x0)
    x1 = x0 + dx ;
    xvect = [xvect x1];
    it = it + 1;
    err = norm(dx);
    x0 = x1;
end


if it == nmax
    disp('errore - non converge')
end

fprintf('Numero di Iterazioni: %d \n', it);
n = length(x0);
for i = 1:n
    fprintf('x(%d) = %12.8f\n', i , xvect(i,end));
end
```

3. Utilizzare la funzione scritta al punto precedente per calcolare la soluzione del sitema non lineare proposto. Utilizzare una tolleranza di `toll = 1e-6`, un vettore iniziale `x0 = [1;0]`. Fornire il risultato e il numero di iterazioni effettuate.

   Si utilizza la funzione come richiesto.

```
toll=1e-6;
x0=[1;0];
Nmax=1000;

[x,iter] = newtonsys(f,J,x0,toll,Nmax);
% Numero di Iterazioni: 6
% x(1) =   2.39901359
% x(2) =   0.40782842
```

6.5 Approssimazione di funzioni e di dati

6.5.1 Interpolazione Lagrangiana e Composita Lineare

Esercizio 1

Si valuti il problema dell'approssimazione della funzione di Runge:

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

Mediante un'interpolazione polinomiale di Lagrange nell'intervallo $I = [-5, 5]$.
Si costruiscano i polinomi interpolanti $\prod_n f$ e dell'errore $E_n f(x) = |f(x) - \prod_n f(x)|$ e si calcoli $||E_n f||_\infty$:

$$||E_n f||_\infty = \max_{x \in [-5,5]} \left| f(x) - \prod_n f(x) \right|$$

Si costruisce il polinomio interpolante $\prod_n f(x)$ di grado $n = 5, 10$ della funzione f considerando nodi equispaziati sull'intervallo I. Per ciascun valore di n si vuole rappresentare graficamente $\prod_n f(x)$ e l'errore $E_n f(x) = |f(x) - \prod_n f(x)|$.

1. Si definisce la funzione di Runge.

```
fun = @(x) 1 ./ (1 + x .^ 2);
```

2. Si memorizzano gli estremi dell'intervallo I nelle variabili a, b e il vettore contenente le ascisse degli $n + 1$ nodi equispaziati.

```
% grado del polinomio interpolante
n = 5;
a = -5;
b = 5;
% nodi equispaziati
x_nod = linspace(a, b, n+1);
```

3. Si procede valutando la funzione in corrispondenza dei nodi.

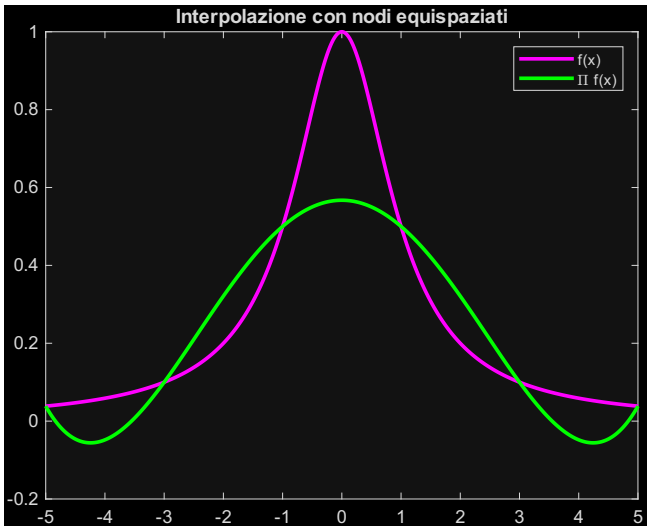
```
f_nod = fun(x_nod);
```

4. Si realizza il polinomio interpolante utilizzando le funzioni `polyfit` e `polyval`.

```
P = polyfit(x_nod, f_nod, n)
% P =
%
%     0.0000    0.0019   -0.0000   -0.0692   -0.0000    0.5673
x_dis = a : 0.01 : b;
poly_dis = polyval(P, x_dis);
```

5. Si visualizza il grafico della funzione e del polinomio interpolante $\prod_n f$ nell'intervallo I.

```
f_dis = fun(x_dis);
plot(x_dis, f_dis, 'm', x_dis, poly_dis, 'g', 'linewidth', 2)
title('Interpolazione con nodi equispaziati')
legend('f(x)', '\Pi f(x)')
```



6. Infine si studia l'andamento dell'errore $E_n f\,(x)$ e lo si rappresenta graficamente.

```
err_dis = abs(poly_dis - f_dis);
plot(x_dis, err_dis, 'm')
title('Errore di interpolazione nodi equispaziati')
```

Esercizio 2

Nella tabella sotto riportata vengono elencati i risultati di un esperimento eseguito per individuare il legame tra lo *sforzo* σ e la relativa *deformazione* ε di un campione di un tessuto biologico.

test	σ [MPa]	ε [cm/cm]
1	0.00	0.00
2	0.06	0.08
3	0.14	0.14
4	0.25	0.20
5	0.31	0.23
6	0.47	0.25
7	0.60	0.28
8	0.70	0.29

A partire da questi dati si vuole stimare, utilizzando opportune tecniche di interpolazione, la deformazione ε del tessuto in corrispondenza dei valori di sforzo per cui non si ha a disposizione un dato sperimentale. A tal fine, si considerino le seguenti funzioni interpolanti:

- Interpolazione polinomiale di Lagrange (`polyfit` e `polyval`);
- Interpolazione polinomiale composita lineare (`interp1`);

In particolare, a partire dal codice assegnato si vuole:

1. Rappresentare graficamente le singole funzioni interpolanti a confronto con i dati sperimentali;
2. Confrontare in un unico grafico i dati sperimentali con tutte le funzioni interpolanti;
3. Valutare, per ogni interpolante, la deformazione ε in corrispondenza di $\sigma = 0.40$ MPa e $\sigma = 0.75$ MPa;
4. Commentare i risultati ottenuti.

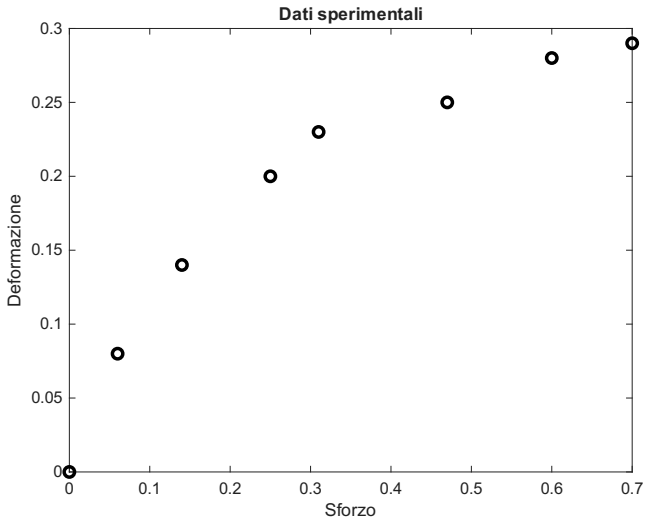
La soluzione:

1. Si definiscono in MATLAB i vettori componenti i dati sperimentali `sigma` (per lo sforzo σ) ed `epsilon` (per la deformazione ε).

```
sigma = [0 0.06 0.14 0.25 0.31 0.47 0.60 0.70];
epsilon = [0 0.08 0.14 0.20 0.23 0.25 0.28 0.29];
```

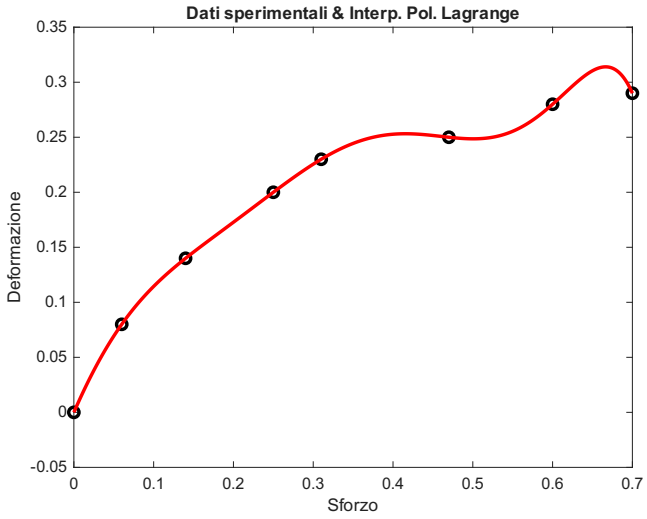
2. In figura viene riportato il grafico dei dati sperimentali, ottenuto con le seguenti istruzioni.

```
figure(1)
axes('FontSize', 12)
plot(sigma, epsilon, 'ko', 'LineWidth', 2)
title('Dati sperimentali')
xlabel('Sforzo')
ylabel('Deformazione')
```



3. L'*interpolazione polinomiale di Lagrange* viene realizzata mediante le funzioni MATLAB `polyfit` e `polyval`. Si ricorda che il numero di punti corrispondenti ai dati sperimentali determina (per definizione) il grado del polinomio di Lagrange, pari al numero di punti meno uno. Per disegnare il polinomio interpolatore di Lagrange si eseguono i seguenti comandi:

```
sigma_dis = linspace(min(sigma), max(sigma), 1000);
grado = length(sigma) - 1;
PL = polyfit(sigma, epsilon, grado);
epsilon_IL = polyval(PL, sigma_dis);

figure(2)
axes('FontSize', 12)
plot(sigma, epsilon, 'ko', sigma_dis, epsilon_IL, 'r', '
    LineWidth', 2)
xlabel('Sforzo')
ylabel('Deformazione')
title('Dati sperimentali & Interp. Pol. Lagrange')
```


Dati sperimentali & Interp. Pol. Lagrange
Deformazione
Sforzo
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
-0.05
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7

7 Esami

7.1 14/06/2025

Esercizio 1

Si consideri la matrice $A \in \mathbb{R}^{n \times n}$ ed il vettore $\mathbf{x}^* \in \mathbb{R}$ tali che:

$$A = \begin{bmatrix} 3 & -2 & 0 & 0 & \cdots & 0 \\ -1 & 3 & -2 & 0 & \cdots & 0 \\ 0 & -1 & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \vdots & \vdots & -1 & 3 & -2 \\ 0 & 0 & \cdots & 0 & -1 & 3 \end{bmatrix} \qquad \mathbf{x}^* = \begin{bmatrix} 1 \\ 2 \\ 3 \\ \vdots \\ n-1 \\ n \end{bmatrix}$$

1. Utilizzando gli opportuni comandi MATLAB, si costruiscano la matrice A ed il vettore $\mathbf{x}^*$ per $n = 10$. Si calcoli $\mathbf{b} \in \mathbb{R}^{10}$ tale che $\mathbf{b} = A\mathbf{x}^*$.

   Soluzione. Il primo passaggio è quello di costruire la matrice A e il vettore $\mathbf{x}^*$ in MATLAB. La matrice A è una matrice tridiagonale con elementi specifici, mentre il vettore $\mathbf{x}^*$ è un vettore colonna con valori da 1 a 10. I comandi MATLAB per costruire questi oggetti sono i seguenti:

```
n = 10;
A = diag(3*ones(n,1)) + diag(-1*ones(n-1,1),-1) + diag(-2*ones
    (n-1,1),1);
x_star = (1:n)';
b = A * x_star;
disp('Matrix A:');
disp(A);
disp('Vector x_star:');
disp(x_star);
disp('Vector b:');
disp(b);
```

```
Matrix A:
     3    -2     0     0     0     0     0     0     0     0
    -1     3    -2     0     0     0     0     0     0     0
     0    -1     3    -2     0     0     0     0     0     0
     0     0    -1     3    -2     0     0     0     0     0
     0     0     0    -1     3    -2     0     0     0     0
     0     0     0     0    -1     3    -2     0     0     0
     0     0     0     0     0    -1     3    -2     0     0
     0     0     0     0     0     0    -1     3    -2     0
     0     0     0     0     0     0     0    -1     3    -2
     0     0     0     0     0     0     0     0    -1     3

Vector x_star:
     1
     2
     3
     4
     5
     6
     7
     8
     9
    10
```

```
24
25 Vector b:
26     -1
27     -1
28     -1
29     -1
30     -1
31     -1
32     -1
33     -1
34     -1
35     21
```

2. Utilizzando il comando `lu` di MATLAB, si calcoli la fattorizzazione LU della matrice A precedentemente costruita. Utilizzando il comando `spy`, si determini se si è verificato il fenomeno del *fill-in*. Si commenti il risultato e si riportino tutti i comandi utilizzati.

   Soluzione. Per calcolare la fattorizzazione LU della matrice A in MATLAB, si utilizza il comando `lu`. Successivamente, si utilizza il comando `spy` per visualizzare la struttura delle matrici L e U e verificare la presenza del fenomeno del *fill-in*. I comandi MATLAB sono i seguenti:

```
[L,U,P] = lu(A);

figure;
subplot(1, 3, 1);
spy(A);
title('Original Matrix A');

subplot(1, 3, 2);
spy(L + U - eye(n));
title('L + U - I (after LU factorization)');

subplot(1, 3, 3);
spy(L + U - eye(n) - A);
title('Fill-in (new non-zeros in L+U)');
```

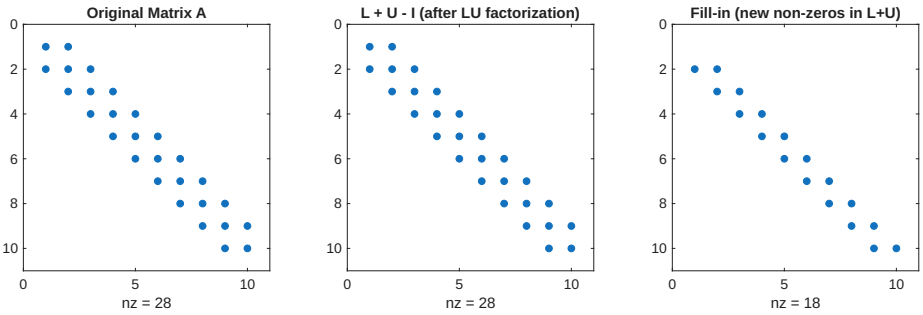
   Il fenomeno del **fill-in** si verifica quando la fattorizzazione LU introduce nuovi elementi non nulli nelle matrici L e U che non erano presenti nella matrice originale A. Più formalmente, il *fill-in* si verifica quando:

$$\text{pattern}\,(L + U - I) \nsubseteq \text{pattern}\,(A)$$

   Ovvero quando la fattorizzazione LU crea nuovi elementi non nulli in posizioni in cui A aveva zeri. Quindi, la matrice corretta da analizzare per il *fill-in* è $L + U - I$, dove I è la matrice identità. Questo perché L e U contengono gli elementi della matrice originale A più quelli introdotti dalla fattorizzazione. Sottraendo l'identità, si rimuovono gli elementi diagonali che sono sempre presenti in L e U. Se la matrice risultante ha più elementi non nulli rispetto ad A, allora si è verificato il *fill-in*.

   - Con `spy(A)` si visualizza la struttura della matrice originale A.
   - Con `spy(L + U - eye(n))` si visualizza la struttura della matrice risultante dalla somma di L e U meno l'identità. Ma attenzione, questa matrice include ancora gli elementi originali di A e non mostra direttamente il *fill-in*.

- Con `spy(L + U - eye(n) - A)` si visualizza la matrice che rappresenta il *fill-in*, ossia i nuovi elementi non nulli introdotti dalla fattorizzazione LU che non erano presenti in A.



Dalla figura, si può osservare che nonostante la matrice `L + U - I` abbia una struttura identica a quella di A, la matrice che rappresenta il *fill-in* mostra che sono stati introdotti nuovi elementi non nulli, indicando che il fenomeno del *fill-in* si è verificato durante la fattorizzazione LU.

3. Utilizzando le matrice L e U calcolate al punto precedente e le funzioni `fwsub.m` e `bksub.m` si risolva il sistema lineare $A\mathbf{x} = \mathbf{b}$. Si riporti il risultato ottenuto ed i comandi utilizzati.

   Soluzione. Per risolvere il sistema lineare $A\mathbf{x} = \mathbf{b}$ utilizzando la fattorizzazione LU, si procede in due passi: prima si risolve il sistema $L\mathbf{y} = \mathbf{b}$ tramite sostituzione in avanti, e poi si risolve il sistema $U\mathbf{x} = \mathbf{y}$ tramite sostituzione all'indietro. I comandi MATLAB per eseguire questi passaggi sono i seguenti:

```
% Risolvi Ly = b usando sostituzione in avanti
y = fwsub(L, b);
% Risolvi Ux = y usando sostituzione all'indietro
x = bksub(U, y);
disp('Soluzione x ottenuta risolvendo Ax = b:');
disp(x);
```

   E il risultato ottenuto sarà:

```
Soluzione x ottenuta risolvendo Ax = b:
     1
     2
     3
     4
     5
     6
     7
     8
     9
    10
```

4. Si calcoli la fattorizzazione LU della matrice $B = AA^T$, dove A^T è la matrice trasposta di A. Si determini se è stato eseguito *pivoting* e si riportino i comandi utilizzati. Utilizzare le matrici calcolate al punto precedente ed il comando \ di MATLAB per risolvere il sistema lineare $B\mathbf{x} = \mathbf{b}$.

Soluzione. Per calcolare la fattorizzazione LU della matrice $B = AA^T$ in MATLAB, si utilizza il comando `lu`. Per verificare se è stato eseguito il *pivoting*, si può controllare la matrice di permutazione P restituita dalla funzione `lu`. I comandi MATLAB sono i seguenti:

```
B = A * A';
[L_B, U_B, P_B] = lu(B);
disp('L factor of B:');
disp(L_B);
disp('U factor of B:');
disp(U_B);
disp('Permutation matrix P_B:');
disp(P_B);

if isequal(P_B, eye(n))
    disp('Il pivoting non e'' stato eseguito nella
    fattorizzazione LU di B.');
else
    disp('Il pivoting e'' stato eseguito nella fattorizzazione
     LU di B.');
end

x_B = B \ b;
disp('Soluzione x ottenuta risolvendo Bx = b usando \:');
disp(x_B);
```

Se il *pivoting* è stato eseguito, la matrice di permutazione P_B non sarà la matrice identità poiché il *pivoting* comporta la riorganizzazione delle righe della matrice per migliorare la stabilità numerica della fattorizzazione LU. Quindi, se P_B è diversa dalla matrice identità, significa che il *pivoting* è stato eseguito. Si ricorda che la matrice di permutazione P_B indica come le righe della matrice originale B sono state riorganizzate durante il processo di fattorizzazione LU; gli elementi non nulli in P_B indicano le posizioni delle righe originali nella matrice permutata. Per esempio, se la prima riga di P_B ha un elemento non nullo nella terza colonna, significa che la terza riga di B è stata spostata alla prima posizione nella matrice permutata.

```
L factor of B:
  1.0000         0         0         0         0         0         0         0         0    0
 -0.6923    1.0000         0         0         0         0         0         0         0    0
       0    0.2574    1.0000         0         0         0         0         0         0    0
       0         0   -0.2841    1.0000         0         0         0         0         0    0
  0.1538   -0.9802   -0.8847   -0.9461    1.0000         0         0         0         0    0
       0         0         0         0    0.4111    1.0000         0         0         0    0
       0         0         0         0         0   -0.3102    1.0000         0         0    0
       0         0         0         0         0         0   -0.4083    1.0000         0    0
       0         0         0         0         0         0         0   -0.4521    1.0000    0
       0         0         0   -0.3869   -0.9399   -0.7603   -0.7360   -0.7700   -0.8297  1.0

U factor of B:
 13.0  -9.0000    2.0000         0         0         0         0         0         0         0
    0    7.7692   -7.6154    2.0000         0         0         0         0         0         0
    0         0   -7.0396   13.4851   -9.0000    2.0000         0         0         0         0
    0         0         0   -5.1688   11.4430   -8.4318    2.0000         0         0         0
    0         0         0         0    4.8645   -6.2082    1.8922         0         0         0
    0         0         0         0         0   -6.4476   13.2220   -9.0000    2.0000         0
    0         0         0         0         0         0   -4.8986   11.2082   -8.3796    2.0000
    0         0         0         0         0         0         0   -4.4239   10.5788   -8.1834
    0         0         0         0         0         0         0         0   -4.2174    6.3003
    0         0         0         0         0         0         0         0         0    0.3978

Permutation matrix P_B:
    1     0     0     0     0     0     0     0     0     0
    0     1     0     0     0     0     0     0     0     0
```

```
    0     0     0     1     0     0     0     0     0     0
    0     0     0     0     1     0     0     0     0     0
    0     0     1     0     0     0     0     0     0     0
    0     0     0     0     0     0     1     0     0     0
    0     0     0     0     0     0     0     1     0     0
    0     0     0     0     0     0     0     0     1     0
    0     0     0     0     0     0     0     0     0     1
    0     0     0     0     0     1     0     0     0     0

Il pivoting e' stato eseguito nella fattorizzazione LU di B.
Soluzione x ottenuta risolvendo Bx = b usando l'operatore backslash:
     1.9624
     4.8872
     8.7367
    13.4358
    18.8339
    24.6302
    30.2228
    34.4079
    34.7782
    26.5188
```

5. (**T**) Definire il numero di condizionamento di una matrice A e discutere la stabilità della soluzione numerica di un generico sistema lineare $A\mathbf{x} = \mathbf{b}$. Dimostrare come il residuo può essere usato come stimatore dell'errore.

   Soluzione.

   Numero di condizionamento. Sia $A \in \mathbb{R}^{n\times n}$ una matrice **invertibile** e sia $\|\cdot\|$ una norma matriciale indotta (ad esempio, la norma 2 o la norma infinito). Il **numero di condizionamento** di A rispetto alla norma $\|\cdot\|$ è definito come:

   $$\kappa(A) = \|A\| \left\|A^{-1}\right\|$$

   Dove $\|A\|$ è la norma della matrice A e $\left\|A^{-1}\right\|$ è la norma della matrice inversa di A. Il numero di condizionamento $\kappa(A)$ misura **quanto** la soluzione di un sistema lineare (x in $A\mathbf{x} = \mathbf{b}$) può amplificare perturbazioni di dati (su $\mathbf{b}$ o su A). Più tale numero è grande, più il problema è **mal condizionato**, il che significa che piccole variazioni nei dati di input possono causare grandi variazioni nella soluzione.

   Stabilità della soluzione numerica di $A\mathbf{x} = \mathbf{b}$

   (a) **Condizionamento del problema** (*sensibilità ai dati*). Per studiare il condizionamento del problema, si introducono delle perturbazioni sui dati del problema.

   Se perturbiamo solo il termine noto:

   $$A\left(\mathbf{x} + \delta\mathbf{x}\right) = \mathbf{b} + \delta\mathbf{b}$$

   Allora, $A\delta\mathbf{x} = \delta\mathbf{b}$, e quindi $\delta\mathbf{x} = A^{-1}\delta\mathbf{b}$. Utilizzando le norme e $\mathbf{b} = A\mathbf{x}$, si ottiene:

   $$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \frac{\left\|A^{-1}\right\| \|\delta\mathbf{b}\|}{\|\mathbf{x}\|} \leq \frac{\left\|A^{-1}\right\| \|\delta\mathbf{b}\|}{\|A^{-1}\|^{-1} \|\mathbf{b}\|} = \kappa(A)\frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|}$$

   Questo significa che **un piccolo errore relativo su $\mathbf{b}$** può produrre un errore relativo su $\mathbf{x}$ amplificato da un fattore pari a $\kappa(A)$.

(b) **Stabilità dell'algoritmo** (*errori di arrotondamento*). Un algoritmo è **backward stable** ("stabile all'indietro") se la soluzione numerica $\hat{\mathbf{x}}$ calcolata dall'algoritmo è esattamente la soluzione di un problema "vicino":

$$(A + \delta A)\,\hat{\mathbf{x}} = \mathbf{b} + \delta\mathbf{b}$$

Con $\dfrac{\|\delta A\|}{\|A\|}$ e $\dfrac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|}$ piccoli (dell'ordine della precisione di macchina, $O(\varepsilon_{\text{mach}})$). Dunque, combinando il condizionamento del problema (punto precedente) con la stabilità dell'algoritmo, si ottiene che l'errore relativo sulla soluzione numerica è limitato da:

$$\frac{\|\hat{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|} \approx O(\kappa(A))\; O(\varepsilon_{\text{mach}})$$

Che significa che l'errore relativo sulla soluzione numerica è proporzionale al numero di condizionamento della matrice A e alla precisione di macchina. Quindi:

- Se $\kappa(A)\varepsilon_{\text{mach}} \ll 1$, la soluzione numerica è generalmente accurata.
- Se $\kappa(A)\varepsilon_{\text{mach}} \approx 1$, la soluzione numerica può essere imprecisa, con poche cifre significative. Si dice che il problema è **mal condizionato**.
- Se $\kappa(A)\varepsilon_{\text{mach}} > 1$, anche un algoritmo stabile può dare una soluzione con pochi (o zero) cifre significative. Si dice che il problema è **gravemente mal condizionato** (o **fortemente mal condizionato**). In altre parole, non è un problema dell'algoritmo, ma è il problema matematico ad essere intrinsecamente instabile da risolvere numericamente.

Il residuo come stimatore dell'errore. Sia $\hat{\mathbf{x}}$ la soluzione numerica approssimata del sistema lineare $A\mathbf{x} = \mathbf{b}$. Il **residuo** associato a questa soluzione è definito come:

$$\mathbf{r} = \mathbf{b} - A\hat{\mathbf{x}}$$

Sia $\mathbf{x}$ la soluzione esatta del sistema lineare e sia $e = \mathbf{x} - \hat{\mathbf{x}}$ l'errore commesso nella soluzione numerica. Allora, si ha:

$$\mathbf{r} = \mathbf{b} - A\hat{\mathbf{x}} = A\mathbf{x} - A\hat{\mathbf{x}} = A(\mathbf{x} - \hat{\mathbf{x}}) = Ae$$

Quindi, l'errore e può essere espresso in termini del residuo $\mathbf{r}$ come:

$$Ae = \mathbf{r} \implies e = A^{-1}\mathbf{r}$$

Inoltre, utilizzando le norme, si ottiene:

$$\|e\| = \left\|A^{-1}\mathbf{r}\right\| \leq \left\|A^{-1}\right\|\,\|\mathbf{r}\|$$

Dividendo entrambi i membri per $\|\mathbf{x}\|$ e utilizzando la relazione $\|\mathbf{b}\| = \|A\mathbf{x}\| \leq \|A\|\,\|\mathbf{x}\|$, si ottiene:

$$\frac{\|e\|}{\|\mathbf{x}\|} \leq \left\|A^{-1}\right\|\,\frac{\|\mathbf{r}\|}{\|\mathbf{x}\|} \leq \left\|A^{-1}\right\|\,\frac{\|\mathbf{r}\|}{\|A\|^{-1}\,\|\mathbf{b}\|} = \kappa(A)\,\frac{\|\mathbf{r}\|}{\|\mathbf{b}\|}$$

Quindi **il residuo relativo** $\frac{\|\mathbf{r}\|}{\|\mathbf{b}\|}$, moltiplicato per il numero di condizionamento $\kappa(A)$, fornisce una stima superiore (**upper bound**) dell'errore relativo nella soluzione numerica $\hat{\mathbf{x}}$. In altre parole, se il residuo è piccolo, allora l'errore nella soluzione numerica è anche piccolo, a meno che la matrice A non sia mal condizionata (cioè, abbia un grande numero di condizionamento).

Key Takeaways

Alcuni punti chiave da ricordare:

- **Residuo piccolo** non implica necessariamente un **errore piccolo** se la matrice è **mal condizionata** ($\kappa(A)$ grande).
- Se A è **ben condizionata** (cioè, $\kappa(A)$ piccolo), allora un **residuo piccolo** garantisce un **errore piccolo**, quindi è un buon indicatore di accuratezza.
- In pratica si usa spesso anche il **residuo normalizzato**:

$$\frac{\|\mathbf{r}\|}{\|A\|\ \|\hat{\mathbf{x}}\| + \|\mathbf{b}\|}$$

  Che è un indicatore di backward error (errore all'indietro) più robusto. Dopodiché, moltiplicando questo residuo normalizzato per $\kappa(A)$, si ottiene l'errore forward (errore in avanti) stimato.

6. Calcolare il condizionamento della matrice B ed il residuo per fornire una stima dell'errore relativo commesso nella risoluzione del sistema lineare $B\mathbf{x} = \mathbf{b}$. Riportare i comandi utilizzati.

   Soluzione. Per calcolare il condizionamento della matrice B e il residuo associato alla soluzione numerica ottenuta, si utilizzano i seguenti comandi MATLAB:

```
cond_B = cond(B);
r = b - B * x_B;
disp(['Condizionamento della matrice B: ', num2str(cond_B)]);
disp('Residuo r = b - Bx:');
disp(r);
norm_r = norm(r);
norm_B = norm(B);
norm_x_B = norm(x_B);
norm_b = norm(b);
residuo_normalizzato = norm_r / (norm_B * norm_x_B + norm_b);
disp([ ...
    'Residuo normalizzato: ', ...
    num2str(residuo_normalizzato) ...
]);
errore_stimato = cond_B * residuo_normalizzato;
disp([ ...
    'Stima dell''errore relativo: ', ...
    num2str(errore_stimato) ...
]);
```

E il risultato ottenuto sarà:

```
Condizionamento della matrice B: 911.3637
Residuo r = b - Bx:
   1.0e-13 *

         0
    0.0355
   -0.0711
   -0.0711
   -0.2842
   -0.4263
   -0.2842
    0.1421
         0
         0

Residuo normalizzato: 2.4185e-17
Stima dell'errore relativo: 2.2042e-14
```

Il sistema lineare $B\mathbf{x} = \mathbf{b}$ presenta un numero di condizionamento di circa 911.36, indicando che il problema è moderatamente mal condizionato ($\kappa(B) \approx 9.11 \times 10^2$). Tuttavia, calcolando il residuo normalizzato, si ottiene un valore molto piccolo di circa 2.42×10^{-17}. Moltiplicando questo residuo per il condizionamento della matrice B, si ottiene una stima dell'errore relativo commesso nella risoluzione del sistema lineare, che risulta essere circa 2.20×10^{-14}. Questo indica che, nonostante il problema sia moderatamente mal condizionato, l'errore relativo nella soluzione numerica è ancora molto piccolo, suggerendo che la soluzione ottenuta è affidabile. Infatti, calcolando il prodotto $\kappa(B) \cdot \varepsilon_{\text{mach}}$ (dove $\varepsilon_{\text{mach}} \approx 2.22 \times 10^{-16}$ per la precisione doppia), si ottiene un valore di circa 2.02×10^{-13}, che è ancora molto piccolo, confermando che la soluzione numerica è accurata nonostante il condizionamento della matrice.

Esercizio 2

Si consideri la funzione non lineare:

$$f(x) = (x-2)\, e^{(x-1)}$$

Dotata dello zero $\alpha = 2$.

Remark 1

Quando l'esercizio specifica:

$$f(x) = (x-2)e^{(x-1)} \quad \text{dotata dello zero } \alpha = 2$$

Intende semplicemente che:

$$f(\alpha) = 0 \quad \text{con } \alpha = 2 \quad \Rightarrow \quad f(2) = (2-2)e^{(2-1)} = 0 \cdot e^1 = 0$$

Con "zero" si intende la radice della funzione, ossia il valore di x per cui $f(x) = 0$.

1. (**T**) Si scriva il metodo di Newton per la determinazione numerica delle radici di una funzione $f : [a, b] \to \mathbb{R}$ come metodo di iterazioni di punto di fisso e si scrivano con precisione le condizioni per la convergenza, specificandone l'ordine. Si definisca un criterio d'arresto affidabile dandone motivazione.

   Soluzione. Il **Metodo di Newton come iterazione di punto fisso** viene formulato nel seguente modo. Dato il problema di trovare una radice α di una funzione $f : [a, b] \to \mathbb{R}$, con $\alpha \in [a, b]$ tale che $f(\alpha) = 0$, con f derivabile in $[a, b]$, il **metodo di Newton** è:

   $$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \quad k = 0, 1, 2, \ldots$$

   Ovvero, la soluzione al passo $k+1$ viene calcolata a partire dalla soluzione al passo k sottraendo il rapporto tra il valore della funzione e il valore della sua derivata prima in x_k. Questo metodo può essere interpretato come un **metodo di iterazioni di punto fisso** definendo la *funzione di iterazione* $\varphi(x)$ come:

   $$\varphi(x) = x - \frac{f(x)}{f'(x)} \quad \Rightarrow \quad x_{k+1} = \varphi(x_k)$$

   Infatti, α è una radice di f se e solo se è un punto fisso di φ, ossia:

   $$f(\alpha) = 0 \quad \Longleftrightarrow \quad \varphi(\alpha) = \alpha$$

   Condizioni di convergenza e ordine.

   (a) **Convergenza locale** (la più usata per Newton). Sia α una **radice semplice** di f, ossia:

   $$f(\alpha) = 0 \qquad f'(\alpha) \neq 0$$

E sia $f \in \mathcal{C}^2$ in un intorno di α (ossia, f è due volte continuamente derivabile vicino a α). Allora, **esiste** un intorno I di α tale che, per ogni scelta di $x_0 \in I$, la successione generata dal metodo di Newton è ben definita e converge a α.

In questo caso la convergenza è **quadratica (ordine 2)**:

$$\|e_{k+1}\| \leq C \, \|e_k\|^2 \quad \text{con } e_k = x_k - \alpha$$

Dove $C > 0$ è una costante dipendente da f e dall'intorno I. Più precisamente (asintoticamente):

$$e_{k+1} = \frac{f''(\alpha)}{2f'(\alpha)} \, e_k^2 + o\left(e_k^3\right)$$

Dove $o\left(e_k^3\right)$ indica termini di ordine superiore a e_k^2. Se invece α è una radice di molteplicità $m > 1$, Newton converge solo **linearmente** (ordine 1) con fattore di convergenza:

$$\|e_{k+1}\| \leq \frac{m-1}{m} \, \|e_k\|$$

(b) **Condizioni "globali" (sufficienti) via punto fisso su un intervallo**. Per usare il teorema delle contrazioni di punto fisso, basta garantire che φ sia una contrazione su un intervallo chiuso $[a, b]$ contenente la radice α. Quindi, si calcola la derivata di φ:

$$\varphi'(x) = 1 - \frac{(f'(x))^2 - f(x)f''(x)}{(f'(x))^2} = \frac{f(x)f''(x)}{(f'(x))^2}$$

Le condizioni sufficienti per la convergenza globale sono:

i. $f \in \mathcal{C}^2(I)$ e $f'(x) \neq 0$ per ogni $x \in I = [a, b]$ (Newton è ben definito, ossia non si divide per zero).

ii. $\varphi(I) \subseteq I$ (ossia, l'immagine di I tramite φ è contenuta in I, garantendo che le iterazioni rimangono in I).

iii. Esiste una costante $q < 1$ tale che:

$$\sup_{x \in I} |\varphi'(x)| = \sup_{x \in I} \left| \frac{f(x)f''(x)}{(f'(x))^2} \right| \leq q < 1$$

Allora φ è una contrazione, ed esiste un unico punto fisso $\alpha \in I$ e $k_t \to \alpha$ per ogni $x_0 \in I$. In altre parole, la convergenza è garantita per ogni scelta di $x_0 \in I$. In questo schema, la garanzia è di **convergenza lineare** con fattore di convergenza $\leq q$ (ossia, l'errore si riduce di almeno un fattore q ad ogni iterazione).

Criterio d'arresto affidabile. Un criterio di arresto robusto dovrebbe controllare:

(a) Quanto bene x_k (l'ultima iterata) soddisfa l'equazione (**residuo**, ossia $|f(x_k)|$). In questo modo si verifica se si è effettivamente vicini a una radice (ossia, se il valore della funzione è vicino a zero).

(b) Se l'iterazione sta ancora cambiando in modo significativo (**incremento**, ossia $|x_k - x_{k-1}|$). Questo aiuta a evitare di fermarsi prematuramente quando si è vicini alla radice ma non si è ancora abbastanza vicini.

In parole semplici, il metodo deve fermarsi quando sia il valore della funzione che la differenza tra iterazioni successive sono entrambi piccoli. Un possibile criterio di arresto potrebbe essere:

$$|f(x_k)| \leq \tau_f \quad \text{e} \quad |x_{k+1} - x_k| \leq \tau_x \, (1 + |x_{k+1}|)$$

Dove:

- **Criterio sul residuo**:
$$|f(x_k)| \leq \tau_f$$
Ovvero, il valore assoluto della funzione in x_k deve essere inferiore a una tolleranza predefinita τ_f.
- **Criterio sull'incremento** (relativo):
$$|x_{k+1} - x_k| \leq \tau_x \, (1 + |x_{k+1}|)$$
Ovvero, la differenza tra iterazioni successive deve essere inferiore a una tolleranza relativa τ_x.

Motivazione. Questo criterio combina due aspetti importanti della convergenza:

- Il criterio sul residuo assicura che si stia effettivamente avvicinando a una radice della funzione. Utilizzando solo questo criterio, si potrebbe fermarsi in un punto dove la funzione è piccola ma non necessariamente vicino alla radice (plateau o minimi locali).
- Il criterio sull'incremento assicura che le iterazioni stiano effettivamente convergendo e non stiano oscillando o stagnando. Utilizzando solo questo criterio, si potrebbe fermarsi in un punto dove le iterazioni non stanno più cambiando significativamente, ma la funzione potrebbe ancora essere lontana da zero.

Utilizzando entrambi i criteri, si può essere più sicuri che il metodo di Newton abbia effettivamente trovato una buona approssimazione della radice desiderata.

2. Si verifichi graficamente che $\alpha = 2$ è una radice per la funzione $f(x)$. Si riportino tutti i comandi MATLAB usati.

Soluzione. Per verificare graficamente che $\alpha = 2$ è una radice della funzione $f(x) = (x-2)e^{(x-1)}$, si può utilizzare MATLAB per tracciare il grafico della funzione e osservare dove essa interseca l'asse delle ascisse (ossia, dove $f(x) = 0$). Ecco i comandi MATLAB utilizzati:

```
x = linspace(0, 4, 400); % Intervallo da 0 a 4 con 400 punti
f = (x - 2) .* exp(x - 1); % Definizione della funzione
figure;
plot(x, f, 'b-', 'LineWidth', 2); % Grafico della funzione
hold on;
yline(0, 'r--'); % Asse x
```

```
xlabel('x');
ylabel('f(x)');
title('Grafico della funzione f(x) = (x - 2)e^{(x-1)}');
grid on;
legend('f(x)', 'y = 0');
hold off;
```

E il grafico risultante mostra chiaramente che la funzione $f(x)$ interseca l'asse delle ascisse in corrispondenza di $x = 2$, confermando che $\alpha = 2$ è effettivamente una radice della funzione.

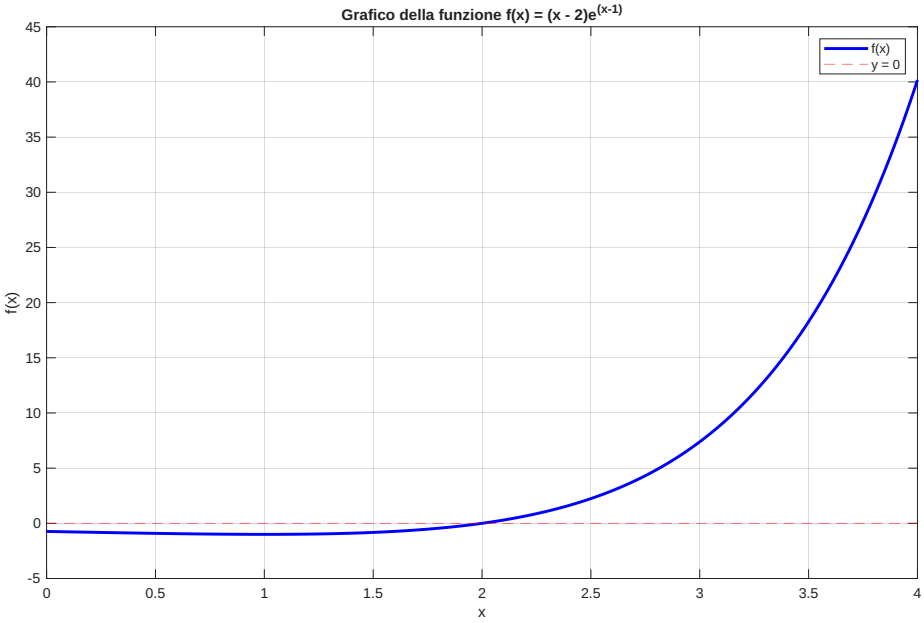


Figura 5: Grafico della funzione $f(x) = (x-2)e^{(x-1)}$ che mostra l'intersezione con l'asse delle ascisse in $x = 2$.

3. Si consideri il metodo delle iterazioni di punto fisso per l'approssimazione dello zero α di $f(x)$ usando la funzione di iterazione

$$\phi = x - \frac{(x-2)\, e^{(x-2)}}{2e-1}$$

Si verifichi graficamente che lo zero α di $f(x)$ coincide con un punto fisso di $\phi(x)$ e si riportino i comandi MATLAB usati.

Soluzione. Per verificare graficamente che lo zero $\alpha = 2$ di $f(x)$ coincide con un punto fisso della funzione di iterazione $\phi(x) = x - \dfrac{(x-2)e^{(x-2)}}{2e-1}$, si può utilizzare MATLAB per tracciare il grafico di $\phi(x)$ insieme alla retta $y = x$. I punti in cui i due grafici si intersecano corrispondono ai punti fissi di $\phi(x)$. Ecco i comandi MATLAB utilizzati:

```
phi = @(x) x - ((x - 2) .* exp(x - 2)) / (2 * exp(1) - 1); %
    Definizione della funzione di iterazione
figure;
```

```
fplot(phi, [0, 4], 'b-', 'LineWidth', 2); % Grafico di phi(x)
hold on;
fplot(@(x) x, [0, 4], 'r--', 'LineWidth', 2); % Grafico della
    bisettrice y = x
xlabel('x');
ylabel('\phi(x) e y = x');
title('Grafico della funzione di iterazione \phi(x) e della
    bisettrice y = x');
grid on;
legend('\phi(x)', 'y = x');
hold off;
```

E il grafico risultante mostra chiaramente che la funzione di iterazione $\phi(x)$ interseca la retta $y = x$ in corrispondenza di $x = 2$, confermando che lo zero $\alpha = 2$ di $f(x)$ coincide con un punto fisso di $\phi(x)$.

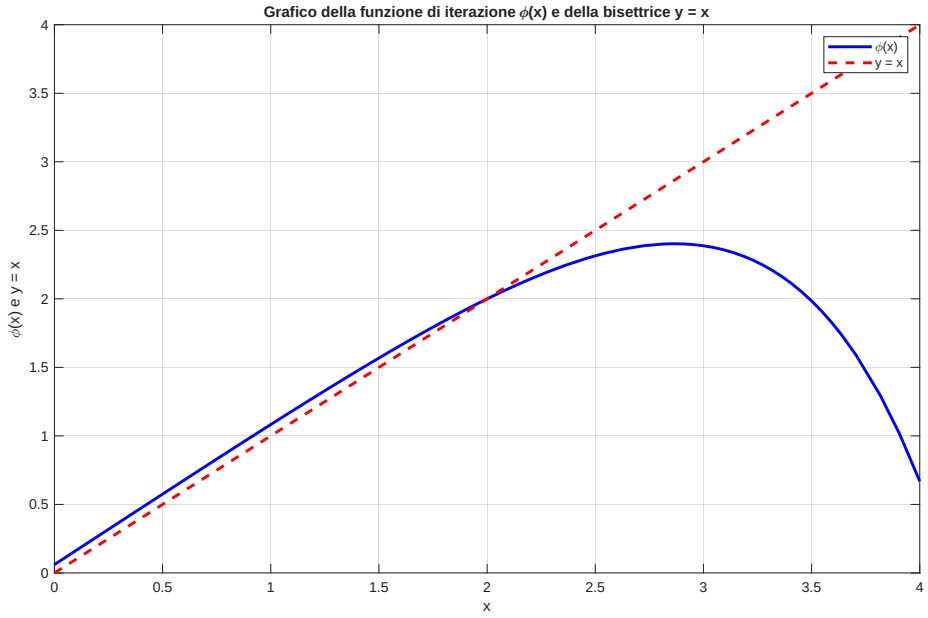


Figura 6: Grafico della funzione di iterazione $\phi(x)$ e della bisettrice $y = x$ che mostra l'intersezione in $x = 2$.

4. (**T**) Sempre considerando il metodo delle iterazioni di punto fisso e la funzione di iterazione (equazione al punto 3), si determini l'ordine di convergenza del metodo ad α per un'iterata iniziale $x^{(0)}$ "sufficientemente" vicino ad α. Si motivi la risposta data alla luce della teoria.

   Soluzione. Data la funzione di iterazione:

$$\phi(x) = x - \frac{(x-2)e^{(x-2)}}{2e-1}$$

   Si definisce g per comodità:

$$g(x) = (x-2)\, e^{(x-2)}$$

Quindi, la funzione di iterazione può essere riscritta come:

$$\phi(x) = x - \frac{g(x)}{2e-1}$$

Per determinare l'ordine di convergenza del metodo delle iterazioni di punto fisso, si calcola la derivata prima di $\phi(x)$:

$$\phi'(x) = 1 - \frac{g'(x)}{2e-1}$$

Dove:

$$g'(x) = e^{(x-2)} + (x-2)e^{(x-2)} = e^{(x-2)}\left(1 + (x-2)\right) = e^{(x-2)}(x-1)$$

Quindi:

$$\phi'(x) = 1 - \frac{e^{(x-2)}(x-1)}{2e-1}$$

Ora, si calcola $\phi'(\alpha)$ con $\alpha = 2$:

$$\phi'(2) = 1 - \frac{e^{(2-2)}(2-1)}{2e-1} = 1 - \frac{1 \cdot 1}{2e-1} = 1 - \frac{1}{2e-1} = \frac{2e-2}{2e-1}$$

Valore numerico:

$$\phi'(2) \approx \frac{2 \cdot 2.71828 - 2}{2 \cdot 2.71828 - 1} = \frac{3.43656}{4.43656} \approx 0.774$$

Poiché $0 < \phi'(\alpha) < 1$ (con $\alpha = 2$), secondo la teoria delle iterazioni di punto fisso, il metodo converge **linearmente** ad α per un'iterata iniziale $x^{(0)}$ sufficientemente vicina ad α. In particolare, l'ordine di convergenza è 1, e il fattore di convergenza è approssimativamente 0.774. Questo significa che l'errore si riduce di circa il 77.4% ad ogni iterazione, quando si è vicini alla radice.

5. Considerando la funzione di iterazione $\phi(x)$ al punto 3, si utilizzi la funzione MATLAB `ptofis.m` con $x^{(0)} = 1.5$, tolleranza $tol = 10^{-4}$ e numero massimo di iterazioni $N_{max} = 1000$. Si riportino il numero di iterazioni N, i valori delle iterate $x^{(1)}$ e $x^{(2)}$, la differenza tra iterate successive $\left|x^{(N)} - x^{(N-1)}\right|$, insieme a tutti i comandi MATLAB utilizzati.

   Soluzione. Per utilizzare la funzione MATLAB `ptofis.m` con la funzione di iterazione $\phi(x)$, si definisce prima la funzione di iterazione in MATLAB e poi si chiama la funzione `ptofis.m` con i parametri specificati. Ecco i comandi MATLAB utilizzati:

```
x0 = 1.5; % Iterata iniziale
tol = 1e-4; % Tolleranza
Nmax = 1000; % Numero massimo di iterazioni
[x, N] = ptofis(x0, Nmax, tol, phi);
disp(['Numero di iterazioni: ', num2str(N)]);
disp(['x^(1) = ', num2str(x(1))]);
disp(['x^(2) = ', num2str(x(2))]);
disp(['Differenza tra iterate successive: ', num2str(abs(x(N)
    - x(N-1)))]);
```

   Il risultato ottenuto è:

```
Numero di iterazioni: 32
x^(1) = 1.5
x^(2) = 1.5684
Differenza tra iterate successive: 0.0001088
```

Quindi, il metodo delle iterazioni di punto fisso ha richiesto 32 iterazioni per convergere alla radice con la tolleranza specificata. Le prime due iterate sono $x^{(1)} = 1.5$ e $x^{(2)} \approx 1.5684$. La differenza tra le ultime due iterate è circa 0.0001088.

6. Si utilizzi opportunamente la funzione MATLAB `stimap.m` per stimare l'ordine di convergenza. Si discutano criticamente i risultati ottenuti con quelli relativi al punto 4 e si riportino i comandi MATLAB utilizzati.

   Soluzione. Per stimare l'ordine di convergenza utilizzando la funzione MATLAB `stimap.m`, si può passare la sequenza delle iterate ottenute dal metodo delle iterazioni di punto fisso. Ecco i comandi MATLAB utilizzati:

```
[p, c] = stimap(x);
disp('Stima dell''ordine di convergenza p:');
disp(p);
disp('Stima del fattore di abbattimento c:');
disp(c);
```

Il risultato ottenuto è:

```
Stima dell'ordine di convergenza p:
  Columns 1 through 10

         0         0    1.2090    1.1786    1.1524    1.1293
    1.1090    1.0910    1.0752    1.0616

  Columns 11 through 20

    1.0500    1.0403    1.0322    1.0256    1.0202    1.0159
    1.0125    1.0098    1.0076    1.0060

  Columns 21 through 30

    1.0046    1.0036    1.0028    1.0022    1.0017    1.0013
    1.0010    1.0008    1.0006    1.0005

  Columns 31 through 32

    1.0004    1.0003

Stima del fattore di abbattimento c:
  Columns 1 through 10

         0         0    1.6194    1.4889    1.3816    1.2903
    1.2113    1.1426    1.0829    1.0314

  Columns 11 through 20

    0.9873    0.9498    0.9183    0.8920    0.8701    0.8521
    0.8373    0.8252    0.8154    0.8074

  Columns 21 through 30

    0.8010    0.7957    0.7915    0.7881    0.7854    0.7832
    0.7815    0.7801    0.7790    0.7781
```

```
30
31  Columns 31 through 32
32
33    0.7774    0.7768
```

Dalla stima dell'ordine di convergenza p, si osserva che l'ordine tende a 1 man mano che le iterazioni procedono, confermando la conclusione del punto 4 che il metodo converge linearmente. Il fattore di abbattimento c si avvicina a un valore inferiore a 1, indicando che l'errore si riduce ad ogni iterazione. Questi risultati sono coerenti con la teoria delle iterazioni di punto fisso, che prevede una convergenza lineare quando la derivata della funzione di iterazione in corrispondenza del punto fisso è compresa tra 0 e 1.

Esercizio 3

Data una funzione $f \in \mathcal{C}^0([0,1])$, si consideri il problema di approssimazione numerica dell'integrale:

$$I(f) = \int_0^1 f(x)\,dx$$

Mediante una formula di quadratura composita con n sottointervalli equispaziati di $[0,1]$.

1. (**T**) Si introduca, in generale, il problema dell'approssimazione numerica dell'integrale sopra definito, tramite formule di quadrature sia semplici che composite, definendo con precisione tutta la notazione utilizzata. Si introducano i concetti di ordine di convergenza e grado di esattezza.

   Soluzione. Sia $f \in \mathcal{C}^0([0,1])$ una funzione continua definita sull'intervallo $[0,1]$. Sia definito l'integrale:

   $$I(f) = \int_0^1 f(x)\,dx$$

   In generale, quando l'integrale non può essere calcolato esattamente o il calcolo esatto è troppo complesso, si ricorre a metodi di approssimazione numerica chiamati **formule di quadratura** usando un **operatore di quadratura** Q che fornisce un'approssimazione numerica di $I(f)$.

   Formule di quadratura semplici. Una **formula di quadratura semplice** su $([a,b])$ ha la forma generale:

   $$Q(f) = \sum_{i=0}^{m} w_i\, f(x_i)$$

   Dove:

   - $x_i \in [a,b]$ sono i **nodi di quadratura** (i.e., i punti in cui la funzione viene valutata).
   - $w_i \in \mathbb{R}$ sono i **pesi di quadratura** associati ai nodi (i.e., i coefficienti che pesano il contributo di ciascuna valutazione della funzione).
   - $m+1$ è il numero totale di nodi (e pesi) utilizzati nella formula (i.e., il numero di punti di valutazione meno uno).

   L'**errore di quadratura** è definito come:

   $$E(f) = I(f) - Q(f)$$

   Ovvero la differenza tra il valore esatto dell'integrale e l'approssimazione fornita dalla formula di quadratura. Esempi classici sono la formula del punto medio, la formula del trapezio e la formula di Simpson.

   Formule di quadratura composite. Poiché le formule di quadratura semplici possono essere imprecise per funzioni complesse o su intervalli ampi, si utilizzano spesso **formule di quadratura composite**. Queste

suddividono l'intervallo $[a, b]$ in n sottointervalli più piccoli e applicano una formula di quadratura semplice su ciascun sottointervallo.

Sia n il numero di sottointervalli, e sia $h = \dfrac{b-a}{n}$ la larghezza di ciascun sottointervallo. I nodi di suddivisione sono dati da:

$$x_i = a + ih \quad \text{per } i = 0, 1, \ldots, n \quad \text{con } h = \frac{b-a}{n}$$

La formula di quadratura composita è quindi:

$$Q_n(f) = \sum_{i=0}^{n} \omega_i f(x_i)$$

Dove ω_i sono i pesi associati ai nodi x_i nei sottointervalli e $f(x_i)$ sono le valutazioni della funzione nei nodi.

Adesso, si applica questa teoria al caso specifico dell'integrale su $[0, 1]$. Dunque, si suddivide l'intervallo $[0, 1]$ in n sottointervalli equispaziati:

$$x_i = 0 + ih = ih \quad \text{con } h = \frac{1-0}{n} = \frac{1}{n}, \quad i = 0, 1, \ldots, n$$

Applicando una formula semplice su ciascun sottointervallo e sommando i contributi, si ottiene una **formula di quadratura composita**:

$$Q_n(f) = \sum_{i=0}^{n} \omega_i f(x_i)$$

Dove ω_i sono i pesi associati ai nodi x_i. L'approssimazione dell'integrale è quindi:

$$I_n(f) \approx Q_n(f) \qquad \text{con errore } E_n(f) = I(f) - Q_n(f)$$

Ordine di convergenza. Si dice che una formula di quadratura composita ha **ordine di convergenza** p se esiste una costante $C > 0$, indipendente da h, tale che:

$$|E_n(f)| \leq C\, h^p \quad \text{per } h \to 0$$

Per ogni funzione f sufficientemente regolare. L'ordine p misura **la velocità con cui l'errore tende a zero** all'aumentare del numero di sottointervalli.

Grado di esattezza. Una formula di quadratura (semplice o composita) si dice **esatta di grado** r se:

$$Q(f) = I(f) \quad \text{per ogni polinomio } f \in \mathbb{P}_r$$

Dove $\mathbb{P}_r$ è lo spazio dei polinomi di grado minore o uguale a r. Il **grado di esattezza** è quindi il massimo grado dei polinomi per cui la formula fornisce il valore esatto dell'integrale.

In conclusione, il problema dell'approssimazione numerica dell'integrale consiste nel sostituire l'operatore integrale con un opportuno operatore di quadratura, semplice o composito. La qualità dell'approssimazione è descritta dall'ordine di convergenza, che misura il decadimento dell'errore al tendere del passo a zero, e dal grado di esattezza, che indica la classe di polinomi integrati esattamente dalla formula.

2. Sia ora $f(x) = (1-2x)\,e^{-2x}$. Utilizzando la funzione `quadcomp.m`, si calcoli l'integrale approssimato $I_1(f)$ corrispondente alla formula di quadratura di tipo semplice (ossia ottenuta con $n = 1$). Riportare il risultato ottenuto e i comandi MATLAB usati.

   Soluzione. Per calcolare l'integrale approssimato $I_1(f)$ utilizzando la funzione MATLAB `quadcomp.m` con $n = 1$, si definisce prima la funzione $f(x) = (1-2x)e^{-2x}$ in MATLAB e poi si chiama la funzione `quadcomp.m`. Ecco i comandi MATLAB utilizzati:

```
f = @(x) (1 - 2*x).*exp(-2*x);
a = 0;
b = 1;
M1 = 1; % numero di sottointervalli per la formula semplice
I1 = quadcomp(a, b, M1, f);
fprintf('Integrale approssimato I1 con n=1: %.8f\n', I1);
```

   Il risultato ottenuto è:

```
Integrale approssimato I1 con n=1: 0.12955351
```

   Quindi, l'integrale approssimato $I_1(f)$ calcolato con la formula di quadratura semplice (con $n = 1$) è circa 0.12955351.

3. Verificare il grado di esattezza della formula di quadratura implementata in `quadcomp.m` calcolando gli errori $E_i = |I(p_i) - I_1(p_i)|$ dove $p_i(x) = x^i$ per $i = 0, \ldots, 4$. Riportare i comandi utilizzati e commentare i risultati ottenuti.

   Soluzione. Per verificare il grado di esattezza della formula di quadratura implementata in `quadcomp.m`, si calcolano gli errori $E_i = |I(p_i) - I_1(p_i)|$ per i polinomi $p_i(x) = x^i$ con $i = 0, \ldots, 4$. Ecco i comandi MATLAB utilizzati:

```
degrees = 0:4;
errors = zeros(size(degrees));
for i = degrees
    % Definizione del polinomio p_i(x) = x^i
    p_i = @(x) x.^i;
    % Calcolo dell'integrale esatto
    I_exact = integral(p_i, a, b);
    % Calcolo dell'integrale approssimato con n=1
    I_approx = quadcomp(a, b, M1, p_i);
    % Calcolo dell'errore
    errors(i+1) = abs(I_exact - I_approx);
    fprintf('Grado %d: Errore E_%d = %.8e\n', i, i, errors(i
    +1));
end
```

   Il risultato ottenuto è:

```
Grado 0: Errore E_0 = 1.11022302e-16
Grado 1: Errore E_1 = 1.11022302e-16
Grado 2: Errore E_2 = 0.00000000e+00
Grado 3: Errore E_3 = 0.00000000e+00
Grado 4: Errore E_4 = 5.55555556e-03
```

Commento sui risultati ottenuti:

- Per i polinomi di grado 0 e 1, l'errore è praticamente zero (dell'ordine della precisione numerica di MATLAB), indicando che la formula di quadratura è esatta per questi gradi.
- Per i polinomi di grado 2 e 3, l'errore è esattamente zero, confermando che la formula di quadratura è esatta anche per questi gradi.
- Per il polinomio di grado 4, l'errore è circa $5.55555556e-03$, indicando che la formula di quadratura non è esatta per questo grado.

Quindi, si conclude che la formula di quadratura implementata in `quadcomp.m` ha un grado di esattezza pari a 3, poiché è esatta per tutti i polinomi fino al grado 3, ma non per il grado 4.

4. Utilizzando la funzione `quadcomp.m`, si calcoli l'integrale approssimato $I_n(f)$ e, sapendo che $I(f) = e^{-2}$, si calcoli l'errore commesso $E_n(f) = |I(f) - I_n(f)|$ per $n = 5, 10, 20, 40$. Riportare i risultati ottenuti nella forma esponenziale e i comandi MATLAB usati.

   Soluzione. Per calcolare l'integrale approssimato $I_n(f)$ utilizzando la funzione MATLAB `quadcomp.m` per diversi valori di n e calcolare l'errore commesso $E_n(f) = |I(f) - I_n(f)|$, si chiama la funzione `quadcomp.m` per i valori specificati di n. Ecco i comandi MATLAB utilizzati:

```
n_values = [5, 10, 20, 40];
I_exact = exp(-2);
errors_n = zeros(size(n_values));
for idx = 1:length(n_values)
    n = n_values(idx);
    I_approx_n = quadcomp(a, b, n, f);
    errors_n(idx) = abs(I_exact - I_approx_n);
    fprintf('n = %d: Errore E_n = %.8e\n', n, errors_n(idx));
end
```

   Il risultato ottenuto è:

```
n = 5: Errore E_n = 1.09681698e-05
n = 10: Errore E_n = 6.89334720e-07
n = 20: Errore E_n = 4.31434853e-08
n = 40: Errore E_n = 2.69740755e-09
```

   Quindi, gli errori commessi per i diversi valori di n sono:

   - Per $n = 5$: $E_5 \approx 1.09681698 \times 10^{-5}$
   - Per $n = 10$: $E_{10} \approx 6.89334720 \times 10^{-7}$
   - Per $n = 20$: $E_{20} \approx 4.31434853 \times 10^{-8}$
   - Per $n = 40$: $E_{40} \approx 2.69740755 \times 10^{-9}$

5. Riportare su un grafico in scala logaritmica l'andamento dell'errore in funzione di $h = 1/n$ confrontandolo con l'andamento h^s per un opportuno $s > 0$. Si riportino tutti i comandi utilizzati. Dedurre l'ordine di convergenza della formula di quadratura e commentare il risultato alla luce di quanto ottenuto al punto 3.

   Soluzione. Per riportare su un grafico in scala logaritmica l'andamento dell'errore in funzione di $h = 1/n$ e confrontarlo con l'andamento h^s, si utilizzano i dati ottenuti nel punto precedente. Ecco i comandi MATLAB utilizzati:

```
h_values = 1 ./ n_values;
figure;
loglog(h_values, errors_n, 'bo-', 'DisplayName', 'Errore E_n');
hold on;
s = 4; % ipotizziamo un ordine di convergenza s=4
loglog(h_values, h_values.^s, 'k--', 'DisplayName', sprintf('h^{%d}', s));
grid on;
xlabel('h = 1/n');
ylabel('Errore');
legend('show');
title('Andamento dell''errore in funzione di h');
hold off;
```

   Il grafico risultante mostra l'andamento dell'errore in funzione di h in scala logaritmica, insieme alla curva di riferimento h^s con $s = 4$.

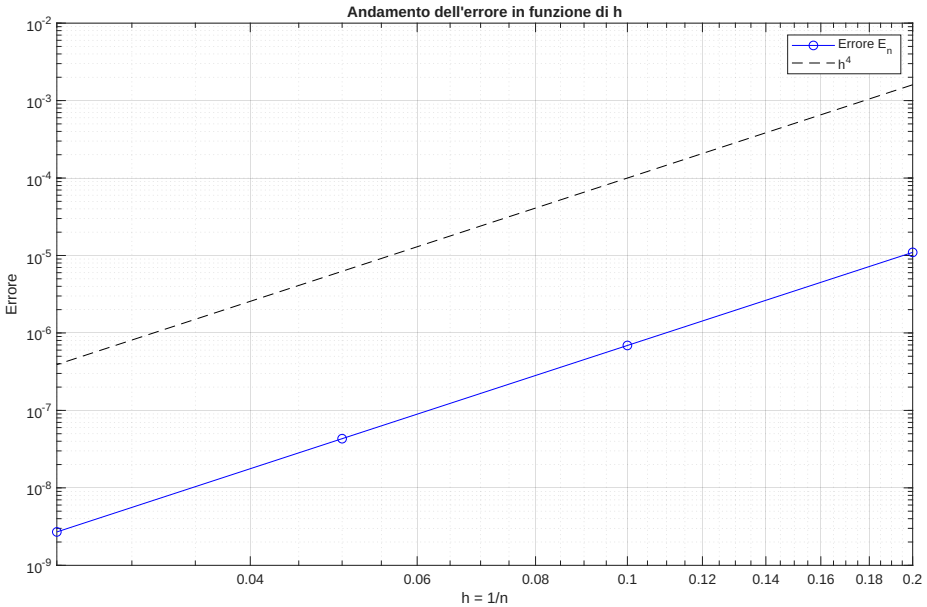


Figura 7: Andamento dell'errore in funzione di h in scala logaritmica, confrontato con h^4.

   Dall'osservazione del grafico, si nota che l'errore decresce rapidamente all'aumentare di n (diminuzione di h), e la pendenza della curva dell'errore è simile a quella della curva di riferimento h^4. Questo suggerisce che

l'ordine di convergenza della formula di quadratura è approssimativamente 4.

Commento sul risultato alla luce di quanto ottenuto al punto 3. Nel punto 3, si è determinato che la formula di quadratura ha un grado di esattezza pari a 3. Secondo la teoria delle formule di quadratura, una formula con grado di esattezza r tende ad avere un ordine di convergenza almeno pari a $r+1$. Pertanto, un grado di esattezza di 3 implica un ordine di convergenza atteso di almeno 4, che è coerente con l'osservazione grafica ottenuta in questo punto. Questo conferma la validità della formula di quadratura utilizzata e la sua efficacia nell'approssimare l'integrale.

7.2 24/07/2025

Esercizio 1

Si consideri la matrice $A \in \mathbb{R}^{n \times n}$ ed il vettore $\mathbf{b} \in \mathbb{R}^n$ tali che:

$$A = \begin{bmatrix} 2 & -1 & 0 & 0 & \cdots & 0 \\ -1 & 2 & -1 & 0 & \cdots & 0 \\ 0 & -1 & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \vdots & \vdots & -1 & 2 & -1 \\ 0 & 0 & \cdots & 0 & -1 & 2 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} -1 \\ -1 \\ -1 \\ \vdots \\ -1 \\ -1 \end{bmatrix}$$

1. Utilizzando gli opportuni comandi Matlab, si costruiscano la matrice A ed il vettore $\mathbf{b}$ per $n = 7$. Utilizzando il comando \ si risolva il sistema lineare $A\mathbf{x} = \mathbf{b}$. Riportare $\mathbf{x}$ e tutti i comandi Matlab usati.

   Soluzione

```
% dimensione di n
n = 7;
% creiamo un vettore di 1 e aggiungiamo 1
A = diag(ones(n,1)+1) + ...      % diag. principale
    diag(ones(n-1,1)-2, 1) + ... % diag. sopra principale
    diag(ones(n-1,1)-2, -1);     % diag. sotto principale
% creiamo vettore b
b = -ones(n,1);
% calcola soluzione di x
x = A\b;
```

   Il vettore `x` produce il seguente risultato:

$$\mathbf{x} = \begin{bmatrix} -3.5 \\ -6 \\ -7.5 \\ -8 \\ -7.5 \\ -6 \\ -3.5 \end{bmatrix}$$

2. Elencare e discutere:

   (a) Le condizioni sufficienti per la convergenza del metodo di Gauss-Seidel.

   (b) Le condizioni necessarie e sufficienti per la convergenza del metodo di Gauss-Seidel.

   Data B_{GS} la matrice di iterazione di Gauss-Seidel, dimostrare che $\|B_{GS}\| < 1$ implica la convergenza.

   Soluzione. Le condizioni sufficienti per la convergenza del metodo di Gauss-Seidel sono 3:

   - La matrice A deve essere a **dominanza diagonale stretta per righe**:

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}| \quad \forall i$$

- La matrice A deve essere **simmetrica definita positiva (SPD)**:

$$\mathbf{x}^T A\mathbf{x} > 0 \quad \forall \mathbf{x} \neq 0$$

- La matrice A deve essere una **Matrice di classe M (M-Matrice) non singolare**. Ovvero una matrice con diagonale positiva e elementi extradiagonali (non sulla diagonale) non positivi.

L'unica condizione necessaria e sufficiente per la convergenza è quando il raggio spettrale della matrice di iterazione B_{GS} è minore di uno:

$$\text{convergenza} \iff \rho(B_{GS}) < 1$$

Dimostrazione ($\|B_{GS}\| < 1$ implica la convergenza). Il metodo di Gauss-Seidel si scrive

$$x^{(k+1)} = B_{GS}x^{(k)} + c$$

Definendo l'errore $e^{(k)} = x^{(k)} - x^*$, si ha

$$e^{(k+1)} = B_{GS}e^{(k)} \quad \Rightarrow \quad e^{(k)} = B_{GS}^k e^{(0)}$$

Se $\|B_{GS}\| < 1$, allora:

$$\|e^{(k)}\| \leq \|B_{GS}\|^k \|e^{(0)}\| \to 0,$$

Per $k \to \infty$. Quindi $x^{(k)} \to x^*$: il metodo converge. QED

3. Verificare se la matrice A soddisfa le condizioni precedenti riportate. Motivare le risposte e riportare i comandi utilizzati.

Soluzione.

```
condition = true;
for row = 1:length(A)
    % valore sulla diagonale
    a_ii = abs(A(row, row));
    % somma della riga inclusa la diagonale
    row_sum = sum(abs(A(row, 1:n)));
    % se la condizione per la row-esima
    % non e' valida, exit
    if ~(a_ii > (row_sum - a_ii))
        condition = false;
        break
    end
end
if condition
    disp("La matrice A e' a dominanza diagonale stretta per
    righe");
else
    disp("La matrice A non e' a dominanza diagonale stretta
    per righe");
end

% dimostrare che e' simmetrica e definita positiva
if issymmetric(A) && all(eig(A) > 0)
    disp("La matrice e' simmetrica definita positiva")
else
    disp("La matrice NON e' simmetrica definita positiva")
end
```

```
26
27 % dimostrare che e' una M-Matrice non singolare
28 % 1. Extradiagonali non positivi
29 n = size(A, 1);
30 % creiamo la matrice con solo la diagonale principale
31 % diag(diag(A)) e sottraiamo alla matrice A la diagonale
32 % principale per ottenere la matrice con solo le
33 % extradiagonali
34 offdiag = A - diag(diag(A));
35 % verifichiamo che tutte le extradiagonali siano <= 0
36 first_m_cond = all(offdiag(:) <= 0);
37 if first_m_cond == false
38     disp("La matrice NON e' una M-matrice: extradiagonali
       positive")
39 end
40 % 2. Diagonale principale positiva
41 second_m_cond = all(diag(A) > 0);
42 if second_m_cond == false
43     disp("La matrice NON e' una M-matrice: diagonale
       principale non positiva")
44 end
45 % 3. A e' non singolare, cioe' rango(A) = n
46 % si ricorda che una matrice e' singolare se det(A) = 0
47 % e che rango(A) = n se e solo se det(A) != 0
48 % perche' il determinante e' il prodotto degli autovalori
49 % e una matrice e' singolare se ha almeno un autovalore nullo
50 third_m_cond = (rank(A) == n);
51 if third_m_cond == false
52     disp("La matrice NON e' una M-matrice: matrice singolare")
53 end
54
55 if first_m_cond && second_m_cond && third_m_cond
56     disp("La matrice e' una M-matrice non singolare")
57 end
```

Output:

```
La matrice A non e' a dominanza diagonale stretta per righe
La matrice e' simmetrica definita positiva
La matrice e' una M-matrice non singolare
```

Quindi, la matrice A soddisfa le condizioni di essere simmetrica definita positiva e di essere una M-matrice non singolare, ma non soddisfa la condizione di dominanza diagonale stretta per righe. Questo comporta che il metodo di Gauss-Seidel converge comunque, in quanto la condizione di dominanza diagonale stretta per righe è solo una condizione sufficiente, ma non necessaria. Infatti, la condizione necessaria e sufficiente per la convergenza del metodo di Gauss-Seidel è che il raggio spettrale della matrice di iterazione B_{GS} sia minore di uno, condizione che è soddisfatta in questo caso:

```
B_gs = eye(size(A,1)) - tril(A)\A;
rho_gs = max(abs(eig(B_gs)));
disp("Raggio spettrale di B_gs: " + rho_gs);
disp("Il metodo Gauss-Seidel converge? " + (rho_gs < 1));
```

Output:

```
Raggio spettrale di B_gs: 0.85355
Il metodo Gauss-Seidel converge? true
```

Quindi, il raggio spettrale di B_{GS} è minore di uno, confermando che il metodo di Gauss-Seidel converge per questa matrice A.

4. Usando la funzione `gs.m` si risolva il sistema lineare $Ax = b$ con il metodo iterativo di Gauss-Seidel per $x^{(0)} = (0, 0, 0, 0, 0, 0, 0)^T$, tolleranza $tol = 10^{-7}$ e massimo numero di iterazioni $N_{\max} = 700$. Indicata con x_{GS} la soluzione approssimata ottenuta, si riportino: il numero di iterazioni effettuate, l'errore $\|x_{GS} - x\|$ e tutti i comandi Matlab utilizzati.

   Soluzione.

```
x0 = zeros(n, 1);
tol = 1e-7;
n_max = 700;
[x_gs, k] = gs(A, b, x0, tol, n_max);
disp("Soluzione con Gauss-Seidel:")
disp(x_gs)
disp("Numero di iterazioni:")
disp(k)
% errore x_gs - x
err_gs = norm(x_gs - x);
disp("Errore tra soluzione esatta e soluzione con Gauss-Seidel
    :")
disp(err_gs)
```

   Output:

```
Soluzione con Gauss-Seidel:
   -3.5000
   -6.0000
   -7.5000
   -8.0000
   -7.5000
   -6.0000
   -3.5000

Numero di iterazioni:
   103

Errore tra soluzione esatta e soluzione con Gauss-Seidel:
   1.4486e-06
```

5. Si riportino il valore del parametro α che massimizza la velocità di convergenza del metodo di Richardson stazionario ed il corrispondente raggio spettrale della matrice di iterazione.

   Soluzione. Da definizione, il parametro α_{opt} che massimizza la velocità di convergenza del metodo di Richardson è:

$$\alpha_{\text{opt}} = \frac{2}{\lambda_{\min}(A) + \lambda_{\max}(A)}$$

   Dove λ rappresenta il vettore degli autovalori della matrice A. Invece, il raggio spettrale della matrice di iterazione corrispondente al parametro α_{opt} é:

$$\rho\left(I - \alpha_{\text{opt}} A\right) = \frac{\lambda_{\max}(A) - \lambda_{\min}(A)}{\lambda_{\max}(A) + \lambda_{\min}(A)}$$

   Implementato quanto detto su MATLAB, il codice è:

```
eigen_values = eig(A);
lambda_min = min(eigen_values);
lambda_max = max(eigen_values);
alpha_opt = 2 / (lambda_min + lambda_max);
disp("Parametro alpha che massimizza la velocita' di
    convergenza: " + alpha_opt)

rho = (lambda_max - lambda_min) / (lambda_max + lambda_min);
disp("Raggio spettrale del metodo di Richardson: " + rho);
```

Output:

```
Alpha che massimizza la velocita' di convergenza: 0.5
Raggio spettrale del metodo di Richardson: 0.92388
```

6. Si ripeta il punto 4 usando la funzione `richardson.m` per applicare il metodo iterativo di Richardson stazionario alla soluzione del sistema lineare con parametro α_{opt} calcolato al punto precedente. Si riportino la soluzione approssimata ottenuta ed il numero di iterazioni effettuate. Sulla base dei risultati ottenuti quale dei due metodi converge più velocemente?

   Soluzione. Al punto 4, il metodo di iterazione aveva le seguenti caratteristiche:

   - $x^{(0)} = (0, 0, 0, 0, 0, 0, 0)^T$
   - $tol = 10^{-7}$
   - $N_{\max} = 700$

   Quindi, il codice MATLAB:

```
x0 = zeros(n, 1);
tol = 1e-7;
n_max = 700;
[x_rich, k_rich, err_rich] = richardson(A, b, x0, alpha_opt,
    tol, n_max);
disp("Soluzione con Richardson:")
disp(x_rich)
disp("Numero di iterazioni: " + k_rich)
% errore x_rich - x
err_rich_final = norm(x_rich - x);
disp("Errore tra soluzione esatta e soluzione con Richardson:
    " + err_rich_final)
```

   Output:

```
Soluzione con Richardson:
   -3.5000
   -6.0000
   -7.5000
   -8.0000
   -7.5000
   -6.0000
   -3.5000

Numero di iterazioni: 203
Errore tra soluzione esatta e soluzione con Richardson: 1.7286
    e-06
```

   Il metodo di Gauss-Seidel ha impiegato la metà del numero di iterazioni rispetto al metodo di Richardson (103 contro 203), dimostrando che il metodo di iterazione che converge più rapidamente è Gauss-Seidel.

Esercizio 2

Sia $f(x) = \log(5x^2+1)$ definita nell'intervallo $I = [-2, 2]$. Si vuole approssimare f con un polinomio interpolante $\Pi_n f$ di grado n su $n+1$ nodi x_i definiti su I. Siano inoltre $\{\bar{x}_j,\, j = 1, \ldots, 701\}$ 701 punti equispaziati su I (estremi inclusi), per il calcolo dell'errore di interpolazione.

Remark 2

Prima di continuare con l'esercizio, è importante ricordare e capire il testo.

- Cosa si intende con "approssimare f con un polinomio interpolante $\prod_n f$ di grado n su $n+1$ nodi x_i definiti su I"? Vuol dire che:
  - Si scelgono $n+1$ **nodi** (cioè punti distinti nell'intervallo $I = [-2, 2]$), chiamiamoli $x_0, x_1, \ldots, x_n$.
  - Si calcolano i valori della funzione in quei nodi: $f(x_0), f(x_1), \ldots, f(x_n)$.
  - Esiste ed è unico un **polinomio di grado al più** n che "passa" per tutti questi punti, ovvero:

    $$\Pi_n f(x_i) = f(x_i), \quad i = 0, 1, \ldots, n$$

    Questo polinomio si chiama **Polinomio Interpolante** di f nei nodi dati.

  Quindi, in altre parole, $\Pi_n f$ non è altro che una funzione polinomiale che coincide con f in un insieme finito di punti, ed è usata per approssimare f anche altrove nell'intervallo.

  Nota: si dice "polinomio interpolante" perché il polinomio *interpola* la funzione, ovvero prende esattamente gli stessi valori della funzione f in un insieme finito di punti (i nodi di interpolazione) e "si inserisce" fra questi valori, passando per essi. Inoltre, interpolare non significa approssimare. Interpolare vuol dire costruire una funzione (nel nostro caso un polinomio) che coincide **esattamente** con f in punti scelti (i nodi), e la approssima negli altri.

- Perché vengono dati 701 punti equispaziati? Questi punti $\{\bar{x}_j\}$ non sono nodi di interpolazione. Servono, invece, come **griglia di controllo**:
  - In ciascun punto $\bar{x}_j$ si calcola il valore esatto $f(\bar{x}_j)$.
  - Poi si calcola il valore approssimato usando il polinomio interpolante $\Pi_n f(\bar{x}_j)$.
  - La differenza:

    $$E(\bar{x}_j) = f(\bar{x}_j) - \Pi_n f(\bar{x}_j)$$

    è l'**Errore di Interpolazione** in quel punto.

Avendo tanti punti equispaziati (701 è un numero grande, quindi la griglia è fitta), si può stimare bene **quanto e come il polinomio interpolante si discosta da** f su tutto l'intervallo $[-2, 2]$. Chiaramente, più punti vengono usati per testare, più accurata sarà la stima della **norma del massimo dell'errore**:

$$\|f - \Pi_n f\|_\infty \approx \max_{1 \le j \le 701} |f(\bar{x}_j) - \Pi_n f(\bar{x}_j)|$$

- Cos'è l'errore di interpolazione? Teoricamente, per una funzione $f \in C^{n+1}$, vale la formula:

$$f(x) - \Pi_n f(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!} \prod_{i=0}^{n} (x - x_i)$$

Dove $\xi(x)$ è un punto (non noto) nell'intervallo. Questo indica che l'errore dipende:

1. Dalla derivata $(n+1)$-esima di f (quanto f è "curva");
2. E dal termine $\prod(x - x_i)$ che cresce con n e con la scelta dei nodi.

Nella pratica, non potendo conoscere $\xi(x)$, si calcola l'errore numericamente sui 701 punti equispaziati.

Quindi, per riassumere: interpolazione significa costruire un polinomio che "passa" per i valori noti di f in $n+1$ nodi. L'errore di interpolazione è la differenza $f(x) - \Pi_n f(x)$, stimata sui 701 punti equispaziati. Quei punti servono solo come "griglia di test" per misurare quanto il polinomio approssima bene la funzione sull'intero intervallo.

1. Fissando $n = 10$, costruire il polinomio interpolante $\Pi_{10} f$ su nodi equispaziati (estremi inclusi) e calcolare il massimo dell'errore di approssimazione sui punti $\{\bar{x}_j\}_{j=1,\dots,701}$. Riportare il risultato ottenuto e i comandi MATLAB usati.

Soluzione

```
% funzione
f = @(x) log(5*x.^2 + 1); % funzione da approssimare

% griglia di controllo x_j
xj = linspace(-2, 2, 701);

% grado del polinomio
n = 10;

% nodi
xn = linspace(-2, 2, n + 1);

% polinomio interpolante di grado n sui nodi xn
Pf = polyfit(xn, f(xn), n);
% massimo errore di approssimazione
err = max(abs(f(xj) - polyval(Pf, xj)));
```

```
disp('Massimo errore di approssimazione');
disp(err);
```

Output:

```
Massimo errore di approssimazione
    0.9989
```

Escludendo la parte di setup, la quale non ha bisogno di spiegazioni approfondite, nella parte di calcolo effettivo, si lasciano alcune osservazioni:

- Comando `polyfit`:

  ```
  Pf = polyfit(xn, f(xn), n);
  ```

  In una sola invocazione, esegue due compiti:

  (a) Prende i **nodi** `xn` e i valori della funzione `f(xn)`;
  (b) Calcola i coefficienti del **polinomio interpolante** di grado n che passa esattamente per quei punti.

  Ovviamente, si potrebbe costruire a mano con la formula di Lagrange o Newton, ma MATLAB fornisce già un API. Il risultato di questa chiamata, è un vettore riga `Pf` con i coefficienti del polinomio in forma classica:

  $$p(x) = Pf(1)\,x^n + Pf(2)\,x^{n-1} + \cdots + Pf(n)\,x + Pf(n+1)$$

- Comando `polyval`:

  ```
  polyval(Pf, xj)
  ```

  Valuta il polinomio definito da `Pf` nei punti `xj`. Quindi sta calcolando $\Pi_{10} f(\bar{x}_j)$ per i 701 punti equispaziati nell'intervallo.

- A questo punto, si hanno due vettori:
  - `f(xj)` per i valori veri della funzione in 701 punti.
  - `polyval(Pf, xj)` per i valori del polinomio interpolante negli stessi 701 punti.

  Per calcolare l'errore, dobbiamo fare la differenza tra i valori veri e quelli interpolati:

  ```
  f(xj) - polyval(Pf, xj)
  ```

  Il quale è il **vettore degli error puntuali** $E(\bar{x}_j)$. Infine, prendiamo il **massimo in valore assoluto** di questi errori, ovvero la norma infinito dell'errore di interpolazione.

2. Ripetere il punto precedente usando i nodi di Chebyshev (estremi inclusi) nell'intervallo I. Indicando con $\Pi_{10}^C f$ l'interpolante corrispondente, riportare il massimo dell'errore di approssimazione sui punti $\{\bar{x}_j\}_{j=1,\dots,701}$ e i comandi Matlab usati.

Remark 3

Alcune osservazioni:

- Perché introdurre i nodi di Chebyshev? Se si scelgono i **nodi equispaziati**, come è stato fatto nell'esercizio precedente, l'errore di interpolazione può diventare molto grande per funzioni un po' oscillanti, soprattutto vicino agli estremi (**Fenomeno di Runge**). Per ridurre questo problema, si usano i **nodi di Chebyshev**, che non sono equispaziati: sono più **densi agli estremi** e più radi al centro.
- I nodi di Chebyshev di ordine $n+1$ (quindi per un polinomio di grado n) in un intervallo generico $[a, b]$ sono:

$$x_k = \frac{a+b}{2} + \frac{b-a}{2} \cdot \cos\left(\frac{2k+1}{2(n+1)} \cdot \pi\right), \quad k = 0, 1, \ldots, n$$

  Noti anche come **nodi di Chebyshev di prima specie**. Per esempio, con $[a, b] = [-1, 1]$ diventano semplicemente:

$$x_k = \cdot \cos\left(\frac{2k+1}{2(n+1)}\pi\right)$$

  Attenzione, questi nodi sono usati nel caso in cui gli estremi **non** siano inclusi.
- Differenza tra i nodi equispaziati e i nodi di Chebyshev:
  - **Equispaziati**: uniformi su $[a, b]$
  - **Chebyshev**: più concentrati agli estremi, quindi si riducono al minimo l'effetto di oscillazioni indesiderate, e quindi garantiscono che l'errore massimo teorico sia quasi ottimale.
- Con i **nodi di Chebyshev di seconda specie**, gli estremi vengono presi in considerazione:

$$x_k = \frac{a+b}{2} + \frac{b-a}{2} \cdot \cos\left(\frac{k}{n} \cdot \pi\right), \quad k = 0, \ldots, n$$

  Quindi $x_0 = a$ e $x_n = b$.

Soluzione. Il codice è identico al precedente ma differisce soltanto come si costruisce il vettore x:

```
% estremi intervallo
a = -2; b = 2;
k = 0:n;

% nodi di Chebyshev (seconda specie, estremi inclusi)
xnC = (a+b)/2 + (b-a)/2 * cos(k/n * pi);
% se fosse stato di prima specie:
% xnC = (a+b)/2 + (b-a)/2 * cos((2*k+1)/(2*(n+1)) * pi);

```

```
% polinomio interpolante di grado n sui nodi di Chebyshev
PfC = polyfit(xnC, f(xnC), n);
% massimo errore di approssimazione
erC = max(abs(f(xj) - polyval(PfC, xj)));
disp('Massimo errore di approssimazione con nodi di Chebyshev'
    );
disp(erC);
```

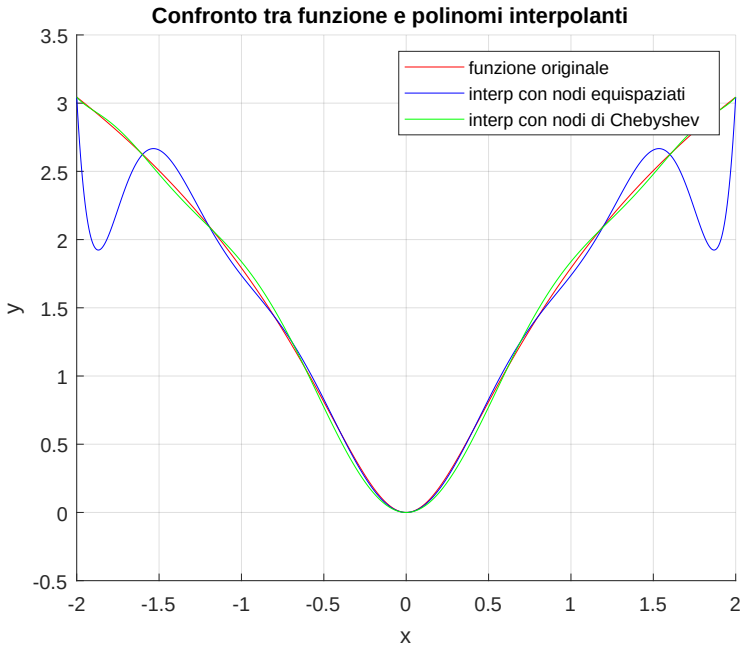
Risultato:

```
Massimo errore di approssimazione con nodi di Chebyshev
     0.060307
```

3. Rappresentare in un unico grafico: la funzione $f(x)$, i polinomi $\Pi_{10} f$ e $\Pi_{10}^{C} f$. Caricare l'immagine generata in formato `png` con opportuna legenda.

Soluzione

```
figure;
hold on;
plot(xj, f(xj), 'r', 'DisplayName', 'funzione originale');
plot(xj, polyval(Pf, xj), '-b', 'DisplayName', 'interp con
    nodi equispaziati');
plot(xj, polyval(PfC, xj), '-g', 'DisplayName', 'interp con
    nodi di Chebyshev');
legend show;
title('Confronto tra funzione e polinomi interpolanti');
xlabel('x');
ylabel('y');
grid on;
hold off;
```



4. Costruire il polinomio interpolante su nodi di Chebyshev al variare di $n = 10, 20, 30, 40$. Calcolare l'errore di interpolazione sui 701 punti $\overline{x}_j$ e riportare in notazione esponenziale il vettore degli errori al variare di n. Quale andamento si osserva?

   Soluzione

```
minVal = 10;
maxVal = 40;
step = 10;
errors = zeros(1, 5);

for n = minVal:step:maxVal
    k = 0:n;
    xnC = (a+b)/2 + (b-a)/2 * cos(k/n * pi);
    PfC = polyfit(xnC, f(xnC), n);
    erC = max(abs(f(xj) - polyval(PfC, xj)));
    errors(n/step) = erC;
end

disp('Errori al variare di n:');
disp(errors);
```

   Risultato:

```
Errori al variare di n:
     0.060307     0.004226     0.0003243    2.7112e-05
```

   Si può osservare una decrescita esponenziale dell'errore al variare di `n`.

5. Si introduca l'interpolante di Lagrange composito Π_k^H con $k \geq 1$ definendo con precisione la notazione utilizzata. A partire dalla stima di convergenza dell'interpolatore Lagrangiano, si deduca la stima dell'errore di interpolazione composita in funzione di H ed k.

Remark 4

L'interpolazione di Lagrange è un approccio diverso al problema dell'interpolazione.

- **Interpolante di Lagrange classico**. Si costruisce **un unico polinomio** $\Pi_n f$ di grado n che interpola f in $n+1$ nodi su tutto l'intervallo $[a, b]$. Ma in questo caso, se n diventa grande, si hanno oscillazioni (**Fenomeno di Runge**), instabilità numerica, costo alto.
- **Interpolante di Lagrange composito**. L'idea è di **non usare un unico polinomio di alto grado**, ma tanti polinomi di basso grado costruiti **pezzo per pezzo** su sottointervalli.
  - Si divide l'intervallo $[a, b]$ in M sottointervalli di ampiezza massima $H = \max |x_{i+1} - x_i|$.
  - Su ogni sottointervallo si costruisce un **polinomio interpolante di grado** k (con $k + 1$ nodi per sottointervallo).

  - L'interpolante composito $\Pi_k^H f$ è la funzione "a tratti" che, su ogni sottointervallo, coincide con il polinomio interpolante locale.

- **Convergenza e stima dell'errore**. Per l'interpolazione di Lagrange su un singolo intervallo, è noto che (se $f \in C^{k+1}$):

  $$|f(x) - \Pi_k f(x)| \leq C\, h^{k+1}, \quad h = \text{ampiezza dell'intervallo}$$

  Per la versione composita:

  - Ogni sottointervallo ha ampiezza $\leq H$;
  - L'errore su ciascun pezzo è $\mathcal{O}\left(H^{k+1}\right)$;
  - Quindi l'errore globale dell'interpolante composito Π_k^H è anch'esso:

  $$\left\|f - \Pi_k^H f\right\|_\infty \leq C\, H^{\,k+1}$$

Soluzione. Sia $I = [a, b]$ un intervallo e sia data una **partizione** (o griglia):

$$\mathcal{T}_H = a = x_0 < x_1 < \cdots < x_M = b, \qquad I_j = [x_j, x_{j+1}],\ j = 0, \ldots, M-1$$

Con **passo massimo**:

$$H := \max_{0 \leq j \leq M-1} h_j, \qquad h_j := x_{j+1} - x_j$$

Fissato un **grado** $k \geq 1$, su ciascun sottointervallo I_j, si scelgono $k+1$ **nodi locali** $\left\{x_j^{(i)}\right\}_{i=0}^{k} \subset I_j$ (ad es. equispaziati in I_j) e si definisce $\Pi_k^H f$ **a tratti** come il **polinomio di Lagrange** di grado k che interpola f nei nodi di I_j:

$$\left(\Pi_k^H f\right)|_{I_j} \;:=\; \Pi_k\left(f|_{I_j}\right) \quad \text{con} \quad \left(\Pi_k f\right)\left(x_j^{(i)}\right) = f\left(x_j^{(i)}\right),\ i = 0, \ldots, k$$

Stima dell'errore. Si assuma $f \in C^{k+1}([a, b])$. Sulla **singola** cella I_j di ampiezza h_j la classica stima dell'errore dell'interpolazione di Lagrange di grado k:

$$\|f - \Pi_k f\|_{L^\infty(I_j)} \;\leq\; C\, h_j^{k+1} \left\|f^{(k+1)}\right\|_{L^\infty(I_j)}$$

Con C indipendente da h_j (deriva dalla formula dell'errore con $\prod (x - x_i)$). Prendendo il massimo su tutte le celle e usando $h_j \leq H$:

$$\left\|f - \Pi_k^H f\right\|_{L^\infty([a,b])} \;\leq\; C\, H^{k+1} \left\|f^{(k+1)}\right\|_{L^\infty([a,b])}$$

Ossia **ordine** $(k+1)$ in H.

Caso $k = 1$, lineare composito. In particolare, se $f \in C^2$ e si usando $k = 1$ e nodi agli estremi di ogni I_j, vale:

$$\left\|f - \Pi_1^H f\right\|_\infty \;\leq\; \frac{H^2}{8}\, \|f''\|_\infty$$

Che mostra esplicitamente l'ordine 2 in H. Spiegazione dettagliata e suggerimenti per ricordare questa formula si trovano a pagina 181.

Esercizio 3

Si consideri il seguente problema di Cauchy:

$$\begin{cases} y'(t) = \left[\dfrac{\pi\cos(\pi t)}{2+\sin(\pi t)} - \dfrac{1}{2}\right] y(t) \quad t \in (0,10) \\ y(0) = 2 \end{cases}$$

La cui soluzione esatta è $y(t) = (2+\sin(\pi t))\, e^{-t/2}$.

1. Si riporti l'algoritmo del metodo di Eulero in avanti application al problema di Cauchy definendo con precisione tutta la notazione utilizzata. Dopo aver posto $f(t,y) = -\lambda y$, con $\lambda > 0$, si ricavi la condizione di assoluta stabilità per il metodo di Eulero in avanti.

   Soluzione

   (a) Si ha $t \in [0,10]$, la condizione iniziale $y(0) = 2$, e la funzione:

   $$f(t,y) = \left[\frac{\pi\cos(\pi t)}{2+\sin(\pi t)} - \frac{1}{2}\right] y$$

   Quindi:

   - Intervallo: $t \in [0,10]$
   - Numero di passi N
   - Passo, bisogna sceglierlo molto piccolo per evitare errori, quindi:

   $$h = \frac{10-0}{N} = \frac{10}{N}$$

   - Punti temporali su cui si approssima la soluzione:

   $$t_n = t_0 + nh \quad n = 0, 1, \ldots, N$$

   - Passo iniziale identico alla condizione iniziale:

   $$u_0 = y(0) = 2$$

   (b) In generale, il metodo di Eulero in avanti si scrive come:

   $$u_{n+1} = u_n + h \cdot f(t_n, u_n)$$

   Dove u_n rappresenta l'approssimazione di $y(t_n)$ e $f(t_n, u_n)$ è la pendenza della curva in quel punto, ovvero la derivata.
   Nel nostro caso, si ha:

   $$u_{n+1} = u_n + h \cdot \left(\frac{\pi\cos(\pi t_n)}{2+\sin(\pi t_n)} - \frac{1}{2}\right) \cdot u_n$$

(c) A questo punto si studia la stabilità del metodo ponendo:

$$f(t, y) = -\lambda y, \quad \lambda > 0$$

Quindi, il metodo di Eulero muta in:

$$u_{n+1} = u_n + h \cdot (-\lambda u_n) = u_n - h\lambda u_n = u_n \cdot (1 - h\lambda)$$

(d) La stabilità del metodo dipende dal fattore moltiplicativo:

$$g = 1 - h\lambda$$

Dato che viene moltiplicato ad ogni passo. Per garantire che la soluzione numerica sia stabile (ovvero non cresca indefinitamente), si richiede che il suo valore assoluto sia minore di 1:

$$|g| < 1 \quad \Rightarrow \quad |1 - h\lambda| < 1$$

Per definizione di valore assoluto, questo implica:

$$-1 < 1 - h\lambda < 1$$

L'obbiettivo è isolare $h\lambda$. Per cui si deve come primo passo rimuovere il $1-\dots$. Per fare questo, si sottrae 1 in tutte e due le disuguaglianze:

$$(-1) + (-1) < (1 - h\lambda) - 1 < 1 + (-1) \quad \Rightarrow \quad -2 < -h\lambda < 0$$

A questo punto, si moltiplica per -1 (cambiando il verso delle disuguaglianze):

$$2 > h\lambda > 0 \quad \Rightarrow \quad 0 < h\lambda < 2$$

Infine, dividendo per λ (positivo, quindi non cambia il verso):

$$\frac{1}{\lambda} \cdot 0 < \frac{1}{\lambda} \cdot h\lambda < \frac{1}{\lambda} \cdot 2 \quad \Rightarrow \quad 0 < h < \frac{2}{\lambda}$$

Quindi, la condizione di stabilità per il metodo di Eulero in avanti è:

$$0 < h < \frac{2}{\lambda}$$

Dove h è il passo scelto per l'iterazione e λ è il parametro della funzione $f(t, y) = -\lambda y$.

A pagina 186 si trova un riepilogo del metodo di Eulero in avanti e della sua stabilità (con una spiegazione più dettagliata).

2. Si riporti l'algoritmo del metodo di Eulero all'indietro con metodo di Newton applicato al problema di Cauchy, definendo con precisione tutta la notazione utilizzata.

Soluzione

(a) Si ha $t \in [0, 10]$, la condizione iniziale $y(0) = 2$, e la funzione:

$$f(t, y) = \left[\frac{\pi \cos(\pi t)}{2 + \sin(\pi t)} - \frac{1}{2} \right] y$$

Quindi:

- Intervallo: $t \in [0, 10]$
- Numero di passi N
- Passo, ottimo se scelto piccolo, ma Eulero all'indietro non è condizionato da questo (A-stabile):

$$h = \frac{10 - 0}{N} = \frac{10}{N}$$

- Punti temporali su cui si approssima la soluzione:

$$t_n = t_0 + nh \quad n = 0, 1, \ldots, N$$

- Passo iniziale identico alla condizione iniziale:

$$u_0 = y(0) = 2$$

(b) In generale, il metodo di Eulero all'indietro si scrive come:

$$u_{n+1} = u_n + h \cdot f\left(t_{n+1}, u_{n+1}\right)$$

Dove u_n rappresenta l'approssimazione di $y(t_n)$ e $f(t_{n+1}, u_{n+1})$ è la pendenza della curva nel punto successivo, che dipende da u_{n+1}. Eulero all'indietro è un metodo implicito, perché u_{n+1} compare sia a sinistra che a destra. Quindi, bisogna utilizzare un metodo numerico, come Newton, per risolvere l'equazione.

(c) Prima di tutto, si riscrivere l'equazione in modo da ottenere una funzione $\Phi(w)$:

$$\begin{aligned} u_{n+1} &= u_n + h \cdot f\left(t_{n+1}, u_{n+1}\right) \\ u_{n+1} - u_n - h \cdot f\left(t_{n+1}, u_{n+1}\right) &= 0 \\ \Phi(w) &= 0 \\ \Phi(w) &:= w - u_n - h \cdot f\left(t_{n+1}, w\right) \end{aligned}$$

Dove w è la variabile incognita che si vuole trovare (cioè u_{n+1}). A questo punto, si può applicare il metodo di Newton per trovare lo zero di $\Phi(w)$.

(d) Il metodo di Newton si scrive come:

$$w^{(k+1)} = w^{(k)} - \frac{\Phi\left(w^{(k)}\right)}{\Phi'\left(w^{(k)}\right)}$$

Dove:

- $w^{(k)}$ è l'iterazione corrente, ovvero l'approssimazione del valore che si sta cercando.
- $w^{(k+1)}$ è l'iterazione successiva, ovvero la nuova approssimazione calcolata.
- $\Phi'(w^{(k)})$ è la derivata di Φ valutata in $w^{(k)}$.

Quindi, si calcola la derivata di $\Phi(w)$:

$$\Phi'(w) = 1 - h \cdot \frac{\partial f}{\partial y}(t_{n+1}, w)$$

E si sostituisce nella formula di Newton:

$$w^{(k+1)} = w^{(k)} - \frac{w^{(k)} - u_n - h \cdot f\left(t_{n+1}, w^{(k)}\right)}{1 - h \cdot \dfrac{\partial f}{\partial y}(t_{n+1}, w^{(k)})}$$

Nel problema specifico, si ha:

$$f(t, y) = \left[\frac{\pi \cos(\pi t)}{2 + \sin(\pi t)} - \frac{1}{2}\right] y$$

Quindi, la derivata parziale di f rispetto a y è :

$$\frac{\partial f}{\partial y}(t, y) = \frac{\pi \cos(\pi t)}{2 + \sin(\pi t)} - \frac{1}{2}$$

Sostituendo tutto nella formula di Newton, si ottiene:

$$w^{(k+1)} = w^{(k)} - \frac{w^{(k)} - u_n - h \cdot \left[\dfrac{\pi \cos(\pi t_{n+1})}{2 + \sin(\pi t_{n+1})} - \dfrac{1}{2}\right] w^{(k)}}{1 - h \cdot \left[\dfrac{\pi \cos(\pi t_{n+1})}{2 + \sin(\pi t_{n+1})} - \dfrac{1}{2}\right]}$$

Dove $t_{n+1} = t_0 + (n+1)h$. L'algoritmo del metodo iterativo di Newton è:

i. Si parte da una **stima iniziale** (ad esempio, $w^{(0)} = u_n$).
ii. Si calcola $\Phi(w^{(k)})$ e la sua derivata $\Phi'(w^{(k)})$.
iii. Si aggiorna l'approssimazione con la formula di Newton.
iv. Si ripete fino a quando la differenza tra due iterazioni successive è minore di una tolleranza prefissata (ad esempio, $|w^{(k+1)} - w^{(k)}| < \text{tol}$).
v. Al termine, si prende come soluzione approssimata $u_{n+1} \approx w^{(k)}$.

Inoltre, nel nostro problema di Cauchy, l'equazione è lineare in y, quindi si potrebbe risolvere direttamente senza Newton.

A pagina 189 si trova un riepilogo del metodo di Eulero all'indietro con il metodo di Newton.

3. Utilizzando la funzione MATLAB `eulero_indietro.m` si risolva il problema di cauchy utilizzando il metodo di Eulero all'indietro per i valori del passo $h_1 = 0.05$ e poi $h_2 = 0.01$. Rappresentare sullo stesso grafico le soluzioni numeriche ottenute e confrontarle con la soluzione esatta. Commentare i risultati ottenuti e riportare tutti i comandi MATLAB usati.

   Soluzione Il problema di Cauchy era:

$$\begin{cases} y'(t) = \left[\dfrac{\pi \cos(\pi t)}{2 + \sin(\pi t)} - \dfrac{1}{2}\right] y(t) \quad t \in (0, 10) \\ y(0) = 2 \end{cases}$$

   La soluzione esatta è $y(t) = (2 + \sin(\pi t))\, e^{-t/2}$. Il codice MATLAB è:

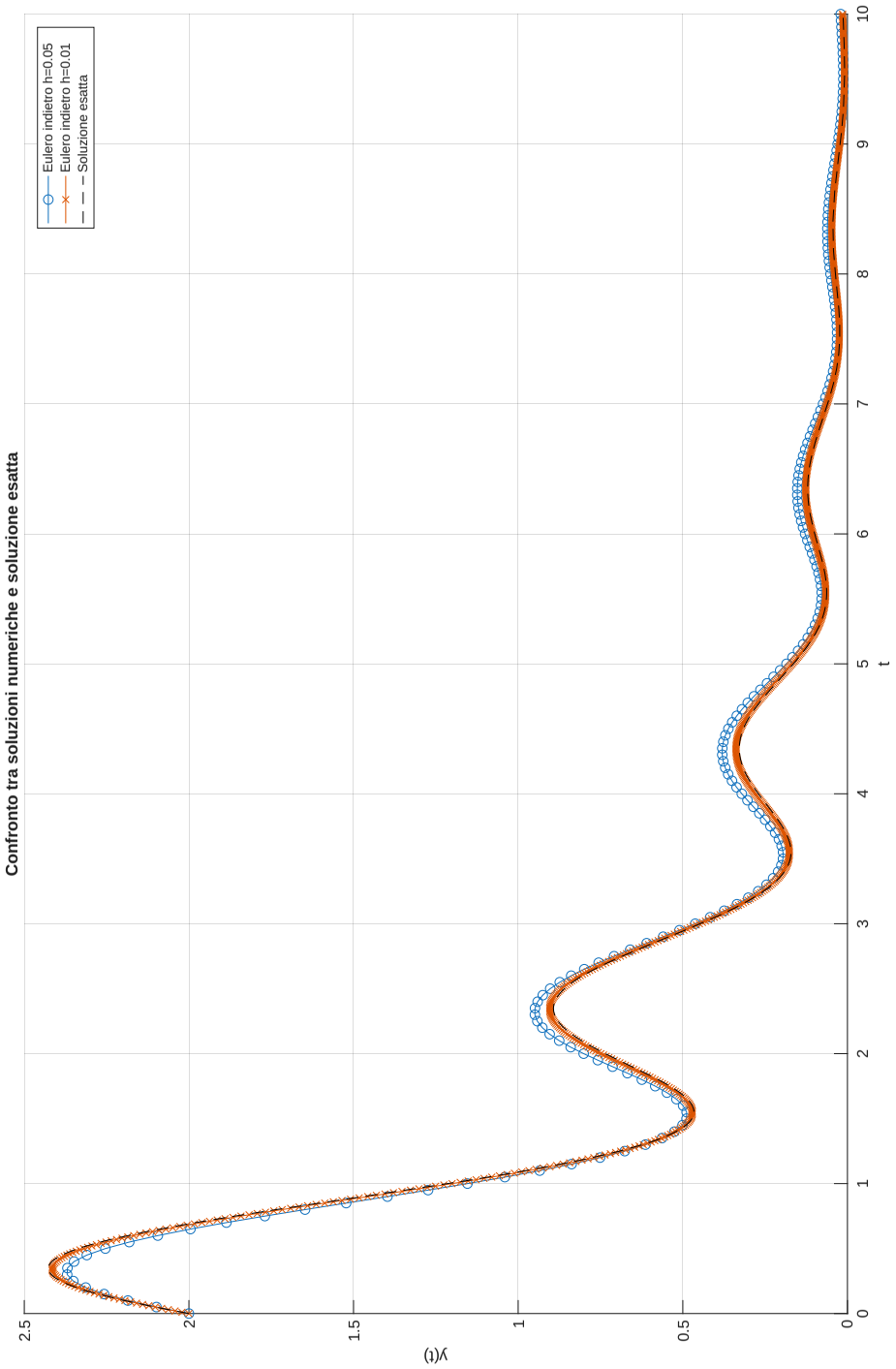
```
% problema Cauchy
f = @(t, y) (                                                ...
    (pi .* cos(pi .* t)) ./ (2 + sin(pi .* t)) - 1/2     ...
) .*y;

% derivata prima di f rispetto a y
f_dy = @(t, y) (                                             ...
    (pi .* cos(pi .* t)) ./ (2 + sin(pi .* t)) - 1/2     ...
);

min_interval = 0;
max_interval = 10;
cond_iniziale = 2;
h1 = 0.05;
h2 = 0.01;

% utilizzare eulero_indietro per calcolare le approssimazioni
[t1, y1] = eulero_indietro( ...
    f, f_dy, min_interval, max_interval, cond_iniziale, h1 ...
);
[t2, y2] = eulero_indietro( ...
    f, f_dy, min_interval, max_interval, cond_iniziale, h2 ...
);

% calcolare la soluzione esatta
y_esatta = @(t) (2 + sin(pi .* t)) .* exp(-t/2);
y1_esatta = y_esatta(t1);
y2_esatta = y_esatta(t2);

% calcolare l'errore
errore1 = abs(y1 - y1_esatta);
errore2 = abs(y2 - y2_esatta);

% stampare gli errori massimi
clc;
fprintf('Errore massimo con h=%f: %f\n', h1, max(errore1));
fprintf('Errore massimo con h=%f: %f\n', h2, max(errore2));

% rappresentare sullo stesso grafico le soluzioni numeriche
    ottenute
% e confrontarle con la soluzione esatta
figure;
hold on;
plot(t1, y1, 'o-', 'DisplayName', 'Eulero indietro h=0.05');
plot(t2, y2, 'x-', 'DisplayName', 'Eulero indietro h=0.01');
fplot(y_esatta, [min_interval, max_interval], 'k--', '
    DisplayName', 'Soluzione esatta');
xlabel('t');
ylabel('y(t)');
title('Confronto tra soluzioni numeriche e soluzione esatta');
legend;
grid on;
hold off;
```

Risultato:

```
Errore massimo con h=0.050000: 0.093496
Errore massimo con h=0.010000: 0.019042
```

Stampando gli errori massimi si noa che l'errore con $h = 0.01$ è circa 5 volte più piccolo di quello con $h = 0.05$, coerente con la convergenza di ordine 1 del metodo di Eulero all'indietro. Dal grafico, si può notare

che entrambe le soluzioni numeriche sono stabili e si avvicinano bene alla soluzione esatta, con quella a passo più piccolo che la segue più da vicino. Per il rapporto costo/accuratezza, $h = 0.01$ è preferibile se si desidera una soluzione molto precisa. Altrimenti, $h = 0.05$ offre un buon compromesso tra accuratezza e costo computazionale.


Confronto tra soluzioni numeriche e soluzione esatta
Eulero indietro h=0.05
Eulero indietro h=0.01
Soluzione esatta
y(t)
t
0
0.5
1
1.5
2
2.5
0
1
2
3
4
5
6
7
8
9
10

8 Domande Teoriche Frequenti

In questa sezione sono raccolte le domande di teoria più ricorrenti, emerse durante le lezioni, le esercitazioni e le prove d'esame. L'obiettivo è fornire un "compendio" sintetico e mirato, che permetta di:

- Avere un quadro immediato dei concetti fondamentali,
- Ripassare rapidamente i punti teorici più importanti,
- Orientarsi sulle domande che più spesso vengono utilizzate per verificare la comprensione.

Questa raccolta non sostituisce lo studio completo dei materiali, ma rappresenta una guida veloce per fissare e richiamare alla memoria i temi principali.

- ***Elencare e discutere:***

  1. ***Le condizioni sufficienti per la convergenza del metodo di Gauss-Seidel.***
  2. ***Le condizioni necessarie e sufficienti per la convergenza del metodo di Gauss-Seidel.***

  Data B_{GS} la matrice di iterazione di Gauss-Seidel, dimostrare che $\|B_{GS}\| < 1$ implica la convergenza.

  Soluzione. Le condizioni sufficienti per la convergenza del metodo di Gauss-Seidel sono 3:

  - La matrice A deve essere a **dominanza diagonale stretta per righe**:
  $$|a_{ii}| > \sum_{j \neq i} |a_{ij}| \quad \forall i$$
  - La matrice A deve essere **simmetrica definita positiva (SPD)**:
  $$\mathbf{x}^T A\mathbf{x} > 0 \quad \forall \mathbf{x} \neq 0$$
  - La matrice A deve essere una **Matrice di classe M (M-Matrice) non singolare**. Ovvero una matrice con diagonale positiva e elementi extradiagonali (non sulla diagonale) non positivi.

  L'unica condizione necessarie e sufficiente per la convergenza è quando il raggio spettrale della matrice di iterazione B_{GS} è minore di uno:
  $$\text{convergenza} \iff \rho(B_{GS}) < 1$$

  Dimostrazione ($\|B_{GS}\| < 1$ implica la convergenza). Il metodo di Gauss-Seidel si scrive
  $$x^{(k+1)} = B_{GS}x^{(k)} + c$$
  Definendo l'errore $e^{(k)} = x^{(k)} - x^*$, dove x^* rappresenta la soluzione esatta del sistema lineare che Gauss-Seidel cerca di raggiungere, e $x^{(k)}$ l'approssimazione della soluzione al passo k, si ha
  $$e^{(k+1)} = B_{GS}e^{(k)} \quad \Rightarrow \quad e^{(k)} = B_{GS}^k e^{(0)}$$

Se $\|B_{GS}\| < 1$, allora:

$$\|e^{(k)}\| \leq \|B_{GS}\|^k \|e^{(0)}\| \to 0,$$

Per $k \to \infty$. Quindi $x^{(k)} \to x^*$: il metodo converge. QED

Deepening: Spiegazione della Dimostrazione

L'idea principale della dimostrazione è: **l'errore ad ogni passo viene moltiplicato da una matrice**; se questa matrice "schiaccia" tutti i vettori di almeno un fattore $q < 1$, allora l'errore cala geometricamente $\to 0$.

- **Passo 0: Dal sistema $Ax = b$ all'iterazione**. Per Gauss-Seidel si scrive (con $A = D + L + U$):

$$\begin{aligned} x^{(k+1)} &= B_{GS}\, x^{(k)} + c \\ B_{GS} &= -(D+L)^{-1}U \\ c &= (D+L)^{-1}b \end{aligned}$$

  Questa è una **iterazione a punto fisso** $x^{(k+1)} = T(x^{(k)})$ con $T(x) = B_{GS}x + c$. Si ricorda che l'iterazione a punto fisso è un metodo iterativo per risolvere $A\mathbf{x} = \mathbf{b}$ che si può sempre scrivere nella forma $x^{(x+1)} = T(x^{(k)})$, dove T è una funzione; se esiste una x^* tale che $T(x^*) = x^*$, allora x^* si chiama punto fisso di T.

  Nel caso Gauss-Seidel, si ha:

$$x^{(k+1)} = B_{GS}x^{(k)} + c$$

  Quindi $T(x) = B_{GS}x^{(k)} + c$. Il punto fisso x^* soddisfa:

$$x^* = B_{GS}x^* + C$$

  Cioè proprio il sistema originale $Ax = b$.

- **Passo 1: Errore che si propaga**. Sia x^* la soluzione (il punto fisso): soddisfa $x^* = B_{GS}x^* + c$. Definiamo l'errore $e^{(k)} = x(k) - x^*$. Sottraendo le due relazioni:

$$e^{(k+1)} = B_{GS}e^{(k)}$$

  Quindi l'errore al passo successivo è semplicemente B_{GS} per l'errore attuale.

- **Passo 2: Usiamo una norma "coerente"**. Prendiamo una norma matriciale indotta (coerente con una norma vettoriale), cioè una per cui vale: $\|Av\| \leq \|A\|\,\|v\|$. Applicandola:

$$\left\|e^{(k+1)}\right\| = \left\|B_{GS}e^{(k)}\right\| \leq \|B_{GS}\| \left\|e^{(k)}\right\|$$

  Iterando:

$$\left\|e^{(k)}\right\| \leq \|B_{GS}\|^k \left\|e^{(0)}\right\|$$

– **Passo 3: Conclusione (contrazione geometrica).** Se $\|B_{GS}\| = q < 1$, allora $q^k \to 0$. Dunque $\left\|e^{(k)}\right\| \to 0$ e quindi $x^{(k)} \to x^*$. **Questo è tutto**. L'ipotesi $\|B_{GS}\| < 1$ garantisce convergenza **da qualunque** $x^{(0)}$.

- ***Si introduca l'interpolante di Lagrange composito Π_k^H con $k \geq 1$ definendo con precisione la notazione utilizzata. A partire dalla stima di convergenza dell'interpolatore Lagrangiano, si deduca la stima dell'errore di interpolazione composita in funzione di H ed k.***

  Soluzione. Sia $I = [a, b]$ un intervallo e sia data una **partizione** (o griglia):

$$\mathcal{T}_H = a = x_0 < x_1 < \cdots < x_M = b, \qquad I_j = [x_j, x_{j+1}],\ j = 0, \ldots, M-1$$

  Con **passo massimo**:

$$H := \max_{0 \leq j \leq M-1} h_j, \qquad h_j := x_{j+1} - x_j$$

  Fissato un **grado** $k \geq 1$, su ciascun sottointervallo I_j, si scelgono $k+1$ **nodi locali** $\left\{x_j^{(i)}\right\}_{i=0}^{k} \subset I_j$ (ad es. equispaziati in I_j) e si definisce $\Pi_k^H f$ **a tratti** come il **polinomio di Lagrange** di grado k che interpola f nei nodi di I_j:

$$\left(\Pi_k^H f\right)|_{I_j} \;:=\; \Pi_k\left(f|_{I_j}\right) \quad \text{con} \quad \left(\Pi_k f\right)\left(x_j^{(i)}\right) = f\left(x_j^{(i)}\right),\ i = 0, \ldots, k$$

  Stima dell'errore. Si assuma $f \in C^{k+1}([a, b])$. Sulla **singola** cella I_j di ampiezza h_j la classica stima dell'errore dell'interpolazione di Lagrange di grado k:

$$\|f - \Pi_k f\|_{L^\infty(I_j)} \;\leq\; C\, h_j^{k+1} \left\|f^{(k+1)}\right\|_{L^\infty(I_j)}$$

  Con C indipendente da h_j (deriva dalla formula dell'errore con $\prod(x - x_i)$). Prendendo il massimo su tutte le celle e usando $h_j \leq H$:

$$\left\|f - \Pi_k^H f\right\|_{L^\infty([a,b])} \;\leq\; C\, H^{k+1} \left\|f^{(k+1)}\right\|_{L^\infty([a,b])}$$

  Ossia **ordine** $(k+1)$ in H.

  Caso $k = 1$, lineare composito. In particolare, se $f \in C^2$ e si usando $k = 1$ e nodi agli estremi di ogni I_j, vale:

$$\left\|f - \Pi_1^H f\right\|_\infty \;\leq\; \frac{H^2}{8}\, \|f''\|_\infty$$

  Che mostra esplicitamente l'ordine 2 in H.

Key Takeaways: Interpolazione di Lagrange composita

Per ricordare:

- **Schema per l'interpolante di Lagrange composito**:

  1. **Partizione dell'intervallo**. Sia $[a, b]$ un intervallo suddiviso in sottointervalli $I_j = [x_j, x_{j+1}]$ con passo massimo $H_{\max_j h_j}$ con $h_j = x_{j+1} - x_j$.

     Cosa si intende con "partizione dell'intervallo"? Si considera un intervallo globale $[a, b]$. Per costruire un interpolante **composito**, non lo si tratta tutto in una volta, ma viene diviso in pezzi più piccoli:

     $$a = x_0 < x_1 < x_2 < \cdots < x_M = b$$

     I piccoli intervalli $[x_j, x_{j+1}]$ si chiamano **sottointervalli** o **celle**. Essi vengono indicati più formalmente come:

     $$I_j = [x_j, x_{j+1}], \quad j = 0, 1, \ldots, M-1$$

     Cosa rappresenta h_j? La **lunghezza** del sottointervallo I_j (ovvero la lunghezza della cella j-esima):

     $$h_j = x_{j+1} - x_j$$

     Ovviamente, se tutti gli x_j fossero equispaziati, allora h_j sarebbe costante. In generale può variare da un intervallo all'altro.

     Cos'è il "passo massimo"? Si definisce:

     $$H = \max_{0 \le j \le M-1} h_j$$

     Ovvero, tra tutte le celle nell'intervallo, si considera quella più lunga. Serve come **misura della finezza della partizione**: più H è piccolo, più la griglia è fitta. L'errore dell'interpolazione composita dipenderà proprio da H.

     Quindi, in altre parole, si divide l'intervallo in "pezzi", dove I_j rappresenta ogni piccolo intervallo; l'ampiezza di ogni intervallo è dato da h_j, e il sottointervallo più grande (il "peggiore"), entra nella stime dell'errore.

  2. **Locale**. Fissato un grado $k \ge 1$, su ogni I_j, si sceglie $k+1$ nodi (es. equispaziati) e si definisce $\left(\Pi_k^H f\right)|_{I_j}$ come il **polinomio di Lagrange di grado** k che interpola f in quei nodi.

L'idea "locale". L'intervallo "grande" scelto all'inizio $[a, b]$ è stato diviso in sottointervalli $I_j = [x_j, x_{j+1}]$. A questo punto, invece di costruire un **unico polinomio globale**, si inizia intanto con i **pezzi**: su ogni sottointervallo I_j si costruisce un piccolo polinomio interpolante.

Grado k e nodi locai. Si fissa un grado del polinomio, in questo caso $k \geq 1$ come suggerito dal testo. Su ciascun sottointervallo I_j si scelgono $k+1$ **nodi** (solitamente equispaziati, ma non è obbligatorio). Per esempio:

- $k = 1$, i due estremi: $\{x_j, x_{j+1}\}$
- $k = 2$, estremi e punto medio: $\left\{x_j, x_{j+\frac{1}{2}}, x_{j+1}\right\}$
- $k = 3$, estremi e due punti:

$$\left\{x_j, x_{j+\frac{1}{3}}, x_{j+\frac{2}{3}}, x_{j+1}\right\}$$

Polinomio di Lagrange locale. Su questi $k+1$ nodi si costruisce il **polinomio di Lagrange di grado** k che interpola f (la funzione). Si indica quindi il polinomio locale con:

$$\left(\Pi_k^H f\right)|_{I_j} \in \mathbb{P}_k$$

Ovvero appartiene allo spazio dei polinomi di grado $\leq k$ e soddisfa:

$$\left(\Pi_k^H f\right)\left(x_j^{(i)}\right) = f\left(x_j^{(i)}\right) \quad i = 0, \ldots, k$$

Quindi, con locale si intende che per ogni sottointervallo I_j si costruisce un polinomio di Lagrange di grado k che interpola la funzione in $k+1$ nodi scelti in quel sottointervallo. In altre parole, si dice locale, perchè si divide l'intervallo in tanti sottointervalli I_j, e su ciascuno si costruisce un polinomio di grado k che interpola **solo i nodi interni a quel sottointervallo**.

3. **Globale**. L'**interpolante di Lagrange composito**:

$$\Pi_k^H f \in X_h^k := \left\{v \in C^0\left([a, b]\right) : v|_{I_j} \in \mathbb{P}_k \forall j\right\}$$

L'idea "globale". Finora è stato visto la costruzione **locale**: su ciascun sottointervallo I_j un polinomio di Lagrange di grado k. Ora si mette insieme tutti questi polinomi locali per formare una funzione definita su tutto l'intervallo $[a, b]$. Questa funzione è l'**interpolante di Lagrange composito** $\Pi_k^H f$.

Spazio funzionale X_h^k. La notazione:

$$X_h^k := \left\{ v \in C^0\left([a,b]\right) : v|_{I_j} \in \mathbb{P}_k \quad \forall j \right\}$$

Vuol dire:

- $C^0\left([a,b]\right)$: le funzioni sono continue su tutto l'intervallo $[a,b]$ (non ci sono salti agli estremi delle celle).
- $v|_{I_j} \in \mathbb{P}_k$: la restrizione di v a ciascun sottointervallo I_j è un polinomio di grado $\leq k$. In parole semplici, su ogni cella, la funzione è un polinomio di grado k.
- $\forall j$: questo vale per tutti i sottointervalli I_j.

Quindi X_h^k è lo spazio di tutte le funzioni continue su $[a,b]$ che, su ogni sottointervallo I_j, sono polinomi di grado $\leq k$. L'interpolante composito $\Pi_k^H f$ appartiene a questo spazio, perchè costruito proprio in questo modo.

L'interpolante composito. Per come è stata costruita, l'interpolante $\Pi_k^H f$ apparitene a X_h^k, dato che:

- Su ogni sottointervallo I_j, è un polinomio di grado k (quindi la restrizione è in $\mathbb{P}_k$);
- Ed è continuo sugli estremi (tutti i polinomi locali coincidono col valore di f sui nodi condivisi).

– **Stima dell'errore**

1. **Locale**. Su I_j, ampiezza h_j, per $f \in C^{k+1}\left(I_j\right)$:

$$\|f - \Pi_k f\|_{L^\infty(I_j)} \leq C_k h_j^{k+1} \left\| f^{(k+1)} \right\|_{L^\infty(I_j)}$$

Dalla formula classica dell'errore di Lagrange.

Che cosa si vuole stimare? A questo punto, si ha la funzione originale f e il suo polinomio interpolante composito $\Pi_k f$ su un sottointervallo $I_j = [x_j, x_{j+1}]$. Si vuole stimare **quanto si discostano** su quell'intervallo, ovvero l'errore di interpolazione:

$$f(x) - \Pi_k f(x)$$

Questo passaggio è fondamentale per capire la qualità dell'approssimazione.

Formula dell'errore (caso generale). Se $f \in C^{k+1}(I_j)$, ovvero f è sufficientemente liscia (derivabile fino a ordine $k+1$), allora esiste un punto

$\xi(x) \in I_j$ tale che:

$$f(x) - \Pi_k f(x) = \frac{f^{(k+1)}(\xi_x)}{(k+1)!} \prod_{i=0}^{k} (x - x_i)$$

Dove:

- $x_0, \ldots, x_k$ sono i nodi usati nell'interpolazione su I_j;
- ξ_x è un punto (non noto) in I_j che dipende da x.
- $k+1$ è il numero di nodi, quindi il grado del polinomio più uno.
- $(k+1)!$ è il fattoriale di $k+1$.
- $f^{(k+1)}(\xi_x)$ è la derivata di ordine $k+1$ di f valutata in ξ_x.
- $\prod_{i=0}^{k}(x - x_i)$ è il prodotto di tutti i termini $(x - x_i)$, che misura la distanza di x dai nodi.

Quindi, l'errore dipende da due fattori:

(a) La derivata di ordine $k+1$ di f, che misura la "curvatura" della funzione.

(b) Il prodotto $\prod (x - x_i)$, che misura quanto x è lontano dai nodi, ovvero quanto l'interpolante può discostarsi dalla funzione.

Quanto vale quel prodotto? Se i nodi stanno in $I_j = [x_j, x_{j+1}]$, allora ogni fattore $(x - x_i)$ è al massimo grande quanto la lunghezza dell'intervallo:

$$|x - x_i| \leq h_j = x_{j+1} - x_j$$

Quindi, il prodotto di tutti i fattori è al massimo:

$$\left| \prod_{i=0}^{k} (x - x_i) \right| \leq h_j^{k+1}$$

Dove C_k è una costante che dipende solo da k, ovvero da come sono scelti i nodi (es. se sono equispaziati, se sono di Chebyshev, etc.).

Risultato finale. Mettendo tutto insieme:

$$|f(x) - \Pi_k f(x)| \leq C_k h_j^{k+1} \max_{y \in I_j} \left\| f^{(k+1)}(y) \right\|$$

Se si prende il massimo su $x \in I_j$, si ottiene la stima dell'errore in norma infinito su I_j:

$$\|f - \Pi_k f\|_{L^\infty(I_j)} \leq C_k h_j^{k+1} \left\| f^{(k+1)} \right\|_{L^\infty(I_j)}$$

In altre parole, l'errore **decresce come una potenza** h_j^{k+1}, quando il sottointervallo si restringe. L'ordine di convergenza è $k+1$: più alto è il grado del polinomio (k), più velocemente l'errore decresce al restringersi dell'intervallo ($h_j \to 0$). Infine, la costante C_k dipende solo da come sono scelti i nodi (es. equispaziati, Chebyshev, etc.), ovvero dal grado e dalla posizione dei nodi, non da h_j.

2. **Globale**. Poichè $h_j \leq H$ per tutti j, si ottiene:

$$\left\|f - \Pi_k^H f\right\|_{L^\infty([a,b])} \leq C_k H^{k+1} \left\|f^{(k+1)}\right\|_{L^\infty([a,b])}$$

Quindi ordine $k+1$ in H.

Da locale a globale. Finora è stata vista la stima dell'errore su un singolo sottointervallo I_j:

$$\|f - \Pi_k f\|_{L^\infty(I_j)} \leq C_k h_j^{k+1} \left\|f^{(k+1)}\right\|_{L^\infty(I_j)}$$

Cioè su ogni cella si sa quanto è grande l'errore. Tuttavia l'obiettivo è stimare l'errore su tutto l'intervallo $[a,b]$. Esso si ottiene prendendo il massimo su tutte le celle (sottointervalli):

$$\left\|f - \Pi_k^H f\right\|_{L^\infty([a,b])} \leq \max_{0 \leq j \leq M-1} \|f - \Pi_k f\|_{L^\infty(I_j)}$$

Uso del passo massimo H. Dato che ogni $h_j \leq H$, si può sostituire h_j con H nella stima dell'errore:

$$\|f - \Pi_k f\|_{L^\infty(I_j)} \leq C_k H^{k+1} \left\|f^{(k+1)}\right\|_{L^\infty(I_j)}$$

Prendendo il massimo su tutte le celle:

$$\left\|f - \Pi_k^H f\right\|_{L^\infty([a,b])} \leq C_k H^{k+1} \max_{0 \leq j \leq M-1} \left\|f^{(k+1)}\right\|_{L^\infty(I_j)}$$

La dipendenza dall'ampiezza massimo H mostra che **raffinando la partizione** (riducendo H), l'errore decresce. L'**ordine di convergenza** è $k+1$: se si usa un polinomio lineare ($k=1$), l'errore decresce come H^2 ($\mathcal{O}(H^2)$); se si usa un polinomio quadratico ($k=2$), l'errore decresce come H^3 ($\mathcal{O}(H^3)$), e così via. Infine, la costante C_k dipende solo da come sono scelti i nodi (es. equispaziati, Chebyshev, etc.), ovvero dal grado e dalla posizione dei nodi, non da H.

In altre parole, dall'ultima equazione, ne consegue che l'interpolante di Lagrange composito converge alla funzione originale al crescere della finezza della partizione (ovvero al diminuire di H), con un ordine di convergenza che dipende dal grado del polinomio usato in ogni sottointervallo.

- ***Si consideri il seguente problema di Cauchy:***

$$\begin{cases} y'(t) = \left[\dfrac{\pi\cos(\pi t)}{2+\sin(\pi t)} - \dfrac{1}{2}\right] y(t) & t \in (0,10) \\ y(0) = 2 \end{cases}$$

La cui soluzione esatta è $y(t) = (2+\sin(\pi t))\, e^{-t/2}$***. Si riporti l'algoritmo del metodo di Eulero in avanti applicato al problema di Cauchy definendo con precisione tutta la notazione utilizzata. Dopo aver posto*** $f(t,y) = -\lambda y$***, con*** $\lambda > 0$***, si ricavi la condizione di assoluta stabilità per il metodo di Eulero in avanti.***

Soluzione. Si ha un **problema di Cauchy** nella forma generale:

$$y'(t) = f(t, y(t)) \quad y(t_0) = y_0$$

L'obbiettivo dei **metodi numerici per ODE** (Ordinary Differential Equations) è costruire una sequenza di valori ($u_n \approx y(t_n)$) che approssimi la soluzione in punti discreti ($t_n = T_0 + nh$). In altre parole, si vuole costruire un metodo per approssimare la soluzione della ODE numericamente, calcolando dei valori $y(t_0), y(t_1), y(t_2), \ldots$, che si avvicinino alla curva vera.

Metodo di Eulero in avanti. Si immagini di avere una curva $y(t)$ ma di conoscerne solo un punto iniziale (t_0, y_0). Se si applica la **derivata** in quel punto, si ottiene la **pendenza** della curva in quel punto:

$$y'(t_0) = f(t_0, y_0)$$

A questo punto, si può **disegnare la tangente** alla curva in quel punto t_0. Se ci si sposta di un piccolo passo h lungo l'asse t, la variazione in y è di circa:

$$\Delta y = h \cdot y'(t_0) = h \cdot f(t_0, y_0)$$

Quindi, il nuovo punto sarà:

$$y(t_1) \approx y_0 + h \cdot f(t_0, y_0)$$

E questo è il cuore del **metodo di Eulero in avanti**.

Più formalmente:

1. Si sceglie un passo h (piccolo, per essere precisi), come per esempio $h = 0.1$.
2. Si creano i punti temporali:

$$t_n = t_0 + nh, \quad n = 0, 1, 2, \ldots, N$$

   Che rappresentano i punti in cui si vuole approssimare la soluzione.
3. Si definisce il punto iniziale:

$$u_0 = y_0 \quad \text{(valore iniziale noto)}$$

4. Dopodiché, per ogni passo che si vuole fare, si applica la formula di Eulero in avanti:
$$u_{n+1} = u_n + h \cdot f(t_n, u_n)$$
Dove u_n è l'approssimazione di $y(t_n)$.

Viene chiamato "**in avanti**" perché si usa l'informazione del punto attuale t_n per calcolare il punto successivo t_{n+1}.

Risposta. Per il problema specifico:

1. Si ha $t \in [0, 10]$, la condizione iniziale $y(0) = 2$, e la funzione:
$$f(t, y) = \left[\frac{\pi \cos(\pi t)}{2 + \sin(\pi t)} - \frac{1}{2}\right] y$$
Quindi:
   - Intervallo: $t \in [0, 10]$
   - Numero di passi N
   - Passo, bisogna sceglierlo molto piccolo per evitare errori, quindi:
$$h = \frac{10 - 0}{N} = \frac{10}{N}$$
   - Punti temporali su cui si approssima la soluzione:
$$t_n = t_0 + nh \quad n = 0, 1, \ldots, N$$
   - Passo iniziale identico alla condizione iniziale:
$$u_0 = y(0) = 2$$

2. In generale, il metodo di Eulero in avanti si scrive come:
$$u_{n+1} = u_n + h \cdot f(t_n, u_n)$$
Dove u_n rappresenta l'approssimazione di $y(t_n)$ e $f(t_n, u_n)$ è la pendenza della curva in quel punto, ovvero la derivata.
Nel nostro caso, si ha:
$$u_{n+1} = u_n + h \cdot \left(\frac{\pi \cos(\pi t_n)}{2 + \sin(\pi t_n)} - \frac{1}{2}\right) \cdot u_n$$

3. A questo punto si studia la stabilità del metodo ponendo:
$$f(t, y) = -\lambda y, \quad \lambda > 0$$
Quindi, il metodo di Eulero muta in:
$$u_{n+1} = u_n + h \cdot (-\lambda u_n) = u_n - h\lambda u_n = u_n \cdot (1 - h\lambda)$$

4. La stabilità del metodo dipende dal fattore moltiplicativo:

$$g = 1 - h\lambda$$

Dato che viene moltiplicato ad ogni passo. Per garantire che la soluzione numerica sia stabile (ovvero non cresca indefinitamente), si richiede che il suo valore assoluto sia minore di 1:

$$|g| < 1 \quad \Rightarrow \quad |1 - h\lambda| < 1$$

Per definizione di valore assoluto, questo implica:

$$-1 < 1 - h\lambda < 1$$

L'obbiettivo è isolare $h\lambda$. Per cui si deve come primo passo rimuovere il $1-\ldots$. Per fare questo, si sottrae 1 in tutte e due le disuguaglianze:

$$(-1) + (-1) < (1 - h\lambda) - 1 < 1 + (-1) \quad \Rightarrow \quad -2 < -h\lambda < 0$$

A questo punto, si moltiplica per -1 (cambiando il verso delle disuguaglianze):

$$2 > h\lambda > 0 \quad \Rightarrow \quad 0 < h\lambda < 2$$

Infine, dividendo per λ (positivo, quindi non cambia il verso):

$$\frac{1}{\lambda} \cdot 0 < \frac{1}{\lambda} \cdot h\lambda < \frac{1}{\lambda} \cdot 2 \quad \Rightarrow \quad 0 < h < \frac{2}{\lambda}$$

Quindi, la condizione di stabilità per il metodo di Eulero in avanti è:

$$0 < h < \frac{2}{\lambda}$$

Dove h è il passo scelto per l'iterazione e λ è il parametro della funzione $f(t, y) = -\lambda y$.

- ***Si consideri il seguente problema di Cauchy:***

$$\begin{cases} y'(t) = \left[\dfrac{\pi\cos(\pi t)}{2+\sin(\pi t)} - \dfrac{1}{2}\right] y(t) & t \in (0,10) \\ y(0) = 2 \end{cases}$$

La cui soluzione esatta è $y(t) = (2+\sin(\pi t))\, e^{-t/2}$***. Si riporti l'algoritmo del metodo di Eulero all'indietro con metodo di Newton applicato al problema di Cauchy, definendo con precisione tutta la notazione utilizzata.***

Soluzione. Dato il solito problema di Cauchy:

$$y'(t) = f(t, y(t)) \quad y(t_0) = y_0$$

L'obbiettivo è lo stesso di Eulero in avanti (pagina 186), ovvero calcolare dei valori approssimati $u_n \approx y(t_n)$ in certi istanti $t_n = t_0 + nh$.

Idea del metodo di Eulero all'indietro. Nel metodo di *Eulero in avanti*, era stata usata la derivata nel punto attuale t_n:

$$u_{n+1} = u_n + h \cdot f(t_n, u_n)$$

Ovvero, si utilizzava la derivata, quindi la pendenza, nel punto *attuale* per stimare dove sarebbe andata la curva nel punto *successivo*. Al contrario, nel **metodo di Eulero all'indietro**, si utilizzata una logica differente:

$$u_{n+1} = u_n + h \cdot f(t_{n+1}, u_{n+1})$$

In questo caso, la derivata f è valutata **al punto futuro** t_{n+1}, e non al punto attuale t_n. Viene detto "*all'indietro*" perché il calcolo della pendenza (derivata) è differente: si usa la pendenza *nel punto successivo*, il quale "guarda indietro" al punto precedente.

Differenze. Nonostante la differenza possa sembrare piccola, essa è fondamentale:

- In **Eulero in avanti**, si può calcolare subito u_{n+1} perché a destra compaiono solo quantità note.
- In **Eulero all'indietro**, invece, u_{n+1} compare anche a destra, precisamente dentro $f(\dots)$.

Quindi bisogna **risolvere un'equazione** per trovare u_{n+1}.

Conseguenze. In Eulero in avanti: "*ci si muove lungo la tangente calcolata al punto in cui ci si trova*"; in Eulero all'indietro: "*ci si muove lungo la tangente calcolata nel punto in cui si vuole arrivare*". Questo rende Eulero all'indietro un **metodo implicito**, mentre Eulero in avanti è un **metodo esplicito**. Perché:

- Esplicito: si può calcolare il nuovo punto direttamente dalle informazioni del punto attuale.

– Implicito: si deve risolvere un'equazione per trovare il nuovo punto, perchè esso compare anche a destra.

Inoltre, questa differenza ha un impatto sulla **stabilità** del metodo:

– Eulero in avanti rischia di "esplodere" (instabile) se il passo h è troppo grande.
– Eulero all'indietro è **molto più stabile**, anche per h grandi, specialmente quando f tende a far "decadere" la soluzione (come $y' = -\lambda y$).

Questo spiega anche perché Eulero all'indietro venga chiamato **A-stabile**, ovvero non diverge mai su sistemi che "dovrebbero" essere stabili.

Come risolvere l'equazione? Ricordando che la formula di Eulero all'indietro è:

$$u_{n+1} = u_n + h \cdot f(t_{n+1}, u_{n+1})$$

Il calcolo non può essere esplicito, dato che il termine u_{n+1} compare **sia a sinistra che a destra**. Quindi, per calcolare u_{n+1}, si deve **risolvere un'equazione**.

Portando tutti i termini noti a sinistra, si ottiene:

$$u_{n+1} - h \cdot f(t_{n+1}, u_{n+1}) - u_n = 0$$

Si definisce la funzione:

$$\Phi(w) = w - h \cdot f(t_{n+1}, w) - u_n$$

Quindi, l'equazione da risolvere diventa:

$$\Phi(w) = 0$$

Ma adesso si ha un problema simile: come si risolve $\Phi(w) = 0$? Se $f(t, y)$ è una funzione non lineare[13] in y, non si può risolvere analiticamente (ovvero con una formula) questa equazione. Per questo motivo, si deve utilizzare un **metodo numerico iterativo** per trovare lo zero di $\Phi(w)$ (cioè la soluzione dell'equazione). Uno dei metodi più comuni, usati e richiesti dall'esercizio è il **metodo di Newton**.

Metodo di Newton. Dato un'equazione $\Phi(w) = 0$, Newton costruisce una sequenza:

$$w^{(k+1)} = w^{(k)} - \frac{\Phi\left(w^{(k)}\right)}{\Phi'\left(w^{(k)}\right)}$$

Che converge (in genere molto velocemente) a una soluzione w^*. In questa equazione:

[13] Un'equazione (o funzione) è **lineare** se la variabile incognita y **compare solo moltiplicata per costanti o funzioni note di** t, **ma non elevata a potenze**, **non dentro funzioni non lineari** (e.g., seno, esponenziale, log, etc.).

Come regola pratica, se è possibile "tirare fuori" u_{n+1} con una semplice divisione o spostamento dei termini, l'equazione è lineare. Altrimenti, se u_{n+1} compare in una funzione o elevato a potenze, l'equazione è non lineare e richiede metodi numerici.

- $w^{(k)}$ è l'iterazione corrente, ovvero l'approssimazione del valore che si sta cercando.
- $\Phi'(w^{(k)})$ è la **derivata** di Φ valutata in $w^{(k)}$.

Quindi, andando ad esplicitare $\Phi(w)$ e la sua derivata:

$$\Phi(w) = w - h \cdot f(t_{n+1}, w) - u_n$$

$$\Phi'(w) = 1 - h \cdot \frac{\partial f}{\partial y}(t_{n+1}, w)$$

Dove $\dfrac{\partial f}{\partial y}$ è la derivata parziale di f rispetto alla seconda variabile (cioè y). Quindi, il metodo di Newton diventa:

$$w^{(k+1)} = w^{(k)} - \frac{w^{(k)} - h \cdot f\left(t_{n+1}, w^{(k)}\right) - u_n}{1 - h \cdot \dfrac{\partial f}{\partial y}(t_{n+1}, w^{(k)})}$$

L'algoritmo del metodo iterativo di Newton è:

1. Si parte da una **stima iniziale** (ad esempio, $w^{(0)} = u_n$).
2. Si calcola $\Phi(w^{(k)})$ e la sua derivata $\Phi'(w^{(k)})$.
3. Si aggiorna l'approssimazione con la formula di Newton.
4. Si ripete fino a quando la differenza tra due iterazioni successive è minore di una tolleranza prefissata (ad esempio, $|w^{(k+1)} - w^{(k)}| < \text{tol}$).

Al termine, si prende come soluzione approssimata $u_{n+1} \approx w^{(k)}$.

Risposta. Per il problema specifico:

1. Si ha $t \in [0, 10]$, la condizione iniziale $y(0) = 2$, e la funzione:

$$f(t, y) = \left[\frac{\pi \cos(\pi t)}{2 + \sin(\pi t)} - \frac{1}{2}\right] y$$

   Quindi:
   - Intervallo: $t \in [0, 10]$
   - Numero di passi N
   - Passo, ottimo se scelto piccolo, ma Eulero all'indietro non è condizionato da questo (A-stabile):

$$h = \frac{10 - 0}{N} = \frac{10}{N}$$

   - Punti temporali su cui si approssima la soluzione:

$$t_n = t_0 + nh \quad n = 0, 1, \ldots, N$$

   - Passo iniziale identico alla condizione iniziale:

$$u_0 = y(0) = 2$$

2. In generale, il metodo di Eulero all'indietro si scrive come:

$$u_{n+1} = u_n + h \cdot f\left(t_{n+1}, u_{n+1}\right)$$

Dove u_n rappresenta l'approssimazione di $y(t_n)$ e $f(t_{n+1}, u_{n+1})$ è la pendenza della curva nel punto successivo, che dipende da u_{n+1}. Eulero all'indietro è un metodo implicito, perché u_{n+1} compare sia a sinistra che a destra. Quindi, bisogna utilizzare un metodo numerico, come Newton, per risolvere l'equazione.

3. Prima di tutto, si riscrivere l'equazione in modo da ottenere una funzione $\Phi(w)$:

$$\begin{aligned} u_{n+1} &= u_n + h \cdot f\left(t_{n+1}, u_{n+1}\right) \\ u_{n+1} - u_n - h \cdot f\left(t_{n+1}, u_{n+1}\right) &= 0 \\ \Phi(w) &= 0 \\ \Phi(w) &:= w - u_n - h \cdot f\left(t_{n+1}, w\right) \end{aligned}$$

Dove w è la variabile incognita che si vuole trovare (cioè u_{n+1}). A questo punto, si può applicare il metodo di Newton per trovare lo zero di $\Phi(w)$.

4. Il metodo di Newton si scrive come:

$$w^{(k+1)} = w^{(k)} - \frac{\Phi\left(w^{(k)}\right)}{\Phi'\left(w^{(k)}\right)}$$

Dove:

- $w^{(k)}$ è l'iterazione corrente, ovvero l'approssimazione del valore che si sta cercando.
- $w^{(k+1)}$ è l'iterazione successiva, ovvero la nuova approssimazione calcolata.
- $\Phi'(w^{(k)})$ è la derivata di Φ valutata in $w^{(k)}$.

Quindi, si calcola la derivata di $\Phi(w)$:

$$\Phi'(w) = 1 - h \cdot \frac{\partial f}{\partial y}(t_{n+1}, w)$$

E si sostituisce nella formula di Newton:

$$w^{(k+1)} = w^{(k)} - \frac{w^{(k)} - u_n - h \cdot f\left(t_{n+1}, w^{(k)}\right)}{1 - h \cdot \dfrac{\partial f}{\partial y}(t_{n+1}, w^{(k)})}$$

Nel problema specifico, si ha:

$$f(t, y) = \left[\frac{\pi \cos(\pi t)}{2 + \sin(\pi t)} - \frac{1}{2}\right] y$$

Quindi, la derivata parziale di f rispetto a y è :

$$\frac{\partial f}{\partial y}(t, y) = \frac{\pi \cos(\pi t)}{2 + \sin(\pi t)} - \frac{1}{2}$$

Sostituendo tutto nella formula di Newton, si ottiene:

$$w^{(k+1)} = w^{(k)} - \frac{w^{(k)} - u_n - h \cdot \left[\frac{\pi \cos(\pi t_{n+1})}{2 + \sin(\pi t_{n+1})} - \frac{1}{2} \right] w^{(k)}}{1 - h \cdot \left[\frac{\pi \cos(\pi t_{n+1})}{2 + \sin(\pi t_{n+1})} - \frac{1}{2} \right]}$$

Dove $t_{n+1} = t_0 + (n+1)h$. L'algoritmo del metodo iterativo di Newton è:

(a) Si parte da una **stima iniziale** (ad esempio, $w^{(0)} = u_n$).

(b) Si calcola $\Phi(w^{(k)})$ e la sua derivata $\Phi'(w^{(k)})$.

(c) Si aggiorna l'approssimazione con la formula di Newton.

(d) Si ripete fino a quando la differenza tra due iterazioni successive è minore di una tolleranza prefissata (ad esempio, $|w^{(k+1)} - w^{(k)}| <$ tol).

(e) Al termine, si prende come soluzione approssimata $u_{n+1} \approx w^{(k)}$.

Inoltre, nel nostro problema di Cauchy, l'equazione è lineare in y, quindi si potrebbe risolvere direttamente senza Newton.

Riferimenti bibliografici

[1] Dede' Luca Fresca Stefania, Botti Michele. Calcolo numerico. Slides from the "Ingegneria Informatica" bachelor's degree course on Politecnico di Milano, 2024.

[2] A. Quarteroni, F. Saleri, and P. Gervasio. *Calcolo Scientifico: Esercizi e problemi risolti con MATLAB e Octave*. UNITEXT. Springer Milan, 2017.

Index

Symbols

A

B

C

D

E

F

N

O

P

R

S

T

U