Calcolo numerico

Indice

Parte I — Teoria 5

1. Problemi matematici e metodi numerici 6
1.1. Definizioni 6
1.2. Aritmetica floating point 7

2. Vettori e matrici 9
2.1. Vettori 9
2.1.1. Operazioni 9
2.2. Matrici 10
2.2.1. Operazioni 10
2.2.2. Determinante 12
2.2.3. Applicazioni del determinante 12

3. Sistemi lineari 14
3.1. Definizione 14
3.2. Metodi diretti per sistemi lineari 15
3.2.1. Metodo delle sostituzioni in avanti 15
3.2.2. Metodo delle sostituzioni all'indietro 15
3.2.3. Metodo di eliminazione gaussiana (MEG) 15
3.2.4. Fattorizzazione LU 16
3.2.5. Fattorizzazione di Cholesky 16
3.3. Metodi iterativi per sistemi lineari 16
3.3.1. Autovalori e autovettori 16
3.3.2. Metodo di Jacobi 18
3.3.3. Metodo di Gauss-Seidel 19
3.3.4. Osservazioni 19
3.3.5. Verificare la convergenza 19
3.3.6. Test d'arresto 19

4. Interpolazione polinomiale 21
4.1. Polinomio interpolatore 21
4.1.1. Metodo di Vandermonde 21
4.1.2. Metodo di Lagrange 22
4.1.3. Errore di interpolazione 22
4.2. Retta di regressione 23
4.3. Spline lineare 24
4.3.1. Errore di interpolazione 24

5. Integrazione numerica 25
5.1. Formule di quadratura semplici 25
5.1.1. Formula del punto medio 25
5.1.2. Formula del trapezio 26
5.1.3. Formula di Cavalieri-Simpson 26
5.2. Formule di quadratura composite 27
5.2.1. Formula del punto medio composita 27
5.2.2. Formula del trapezio composita 27
5.2.3. Formula di Cavalieri-Simpson composita 28

6. Zeri di funzione **29**
6.1. Teorema degli zeri 29
6.2. Metodo di bisezione 29
6.3. Metodo di Newton 29

7. Metodi numerici per equazioni differenziali ordinarie **31**
7.1. Metodo di Eulero 31
7.2. Metodo di Crank-Nicolson 31
7.3. Metodo di Heun 31
7.4. Consistenza, convergenza e stabilità 32

8. Metodi numerici per sistemi di equazioni differenziali ordinarie **33**
8.1. Metodo di Eulero esplicito 33
8.2. Metodo di Eulero implicito 33

9. Come fare gli esercizi proposti nei TDE **34**
9.1. Sistemi lineari 34
9.2. Interpolazione polinomiale 34
9.3. Integrazione numerica 34
9.4. Zeri di funzione 34
9.5. Equazioni differenziali ordinarie 34

Parte II — Laboratorio **35**

1. Introduzione a Matlab **36**
1.1. Solite nozioni di ogni linguaggio di programmazione 36
1.2. Grafici 37

2. Vettori e matrici **38**
2.1. Vettori 38
2.2. Matrici 39

3. Sistemi lineari **41**
3.1. Metodi diretti 41
3.2. Metodi iterativi 41
3.2.1. Autovalori e autovettori 41
3.2.2. Metodo di Jacobi e Gauss-Seidel 41

4. Interpolazione polinomiale **42**
4.1. Lavorare con i polinomi 42
4.2. Polinomio interpolatore 42

5. Integrazione numerica **43**
5.1. Formule di quadratura 43

6. Zeri di funzione **44**
6.1. Metodo di bisezione 44
6.2. Metodo di Newton 44

7. Equazioni differenziali ordinarie **45**

8. Sistemi lineari di equazioni differenziali ordinarie **46**

9. Codici utilizzati **47**
9.1. Sistemi lineari 47
9.2. Interpolazione polinomiale 48

9.3. Integrazione numerica 48
9.4. Zeri di funzione 49
9.5. Equazioni differenziali ordinarie 50

Parte I — Teoria

1. Problemi matematici e metodi numerici

La **matematica numerica** (*o analisi numerica*) è la branca della matematica applicata che sviluppa algoritmi implementabili in un calcolatore per approssimare problemi matematici non risolubili per via analitica.

1.1. Definizioni

Un **problema matematico** in forma astratta è un problema che chiede di trovare u tale che

$$P(d, u) = 0,$$

con:

- d insieme dei dati;
- u soluzione;
- P operatore che esprime la relazione funzionale tra u e d.

Le due variabili possono essere numeri, vettori, funzioni, eccetera.

Un **metodo numerico** per la risoluzione approssimata di un problema matematico consiste nel costruire una successioni di problemi approssimati del tipo

$$P_n(d_n, u_n) = 0 \mid n \geq 1$$

oppure

$$P_h(d_h, u_h) = 0 \mid h > 0$$

che dipendono dai parametri n o h.

Un metodo numerico è **convergente** se

$$\lim_{n \to \infty} u_n = u$$

oppure

$$\lim_{h \to 0} u_h = u.$$

Il problema matematico $P(d, u) = 0$ è **ben posto** (*o stabile*) se, per un certo dato d, la soluzione u esiste ed è unica e dipende con continuità dai dati. Questa ultima proprietà indica che piccole perturbazioni (*variazioni*) dei dati d producono piccole perturbazioni nella soluzione u.

Per quantificare la dipendenza continua dai dati introduciamo il concetto di **numero di condizionamento** di un problema.

Consideriamo una funzione $f : [a, b] \longrightarrow \mathbb{R}$ in un punto x_0, ovvero

$$d := x_0 \quad u := f(x_0) \mid d, u \in \mathbb{R}.$$

Applichiamo lo sviluppo di Taylor di f in x_0, ovvero:

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \ldots$$

Ma allora:

$$f(x) - f(x_0) \approx f'(x_0)(x - x_0).$$

Dividiamo per $f(x_0)$ e aggiungiamo un fattore x_0 a sinistra:

$$\frac{f(x) - f(x_0)}{f(x_0)} \approx \frac{x_0 f'(x_0)}{f(x_0)} \frac{x - x_0}{x_0}.$$

Prendiamo i valori assoluti di entrambi i membri:

$$\left| \frac{f(x) - f(x_0)}{f(x_0)} \right| \approx \left| \frac{x_0 f'(x_0)}{f(x_0)} \right| \left| \frac{x - x_0}{x_0} \right|.$$

Osserviamo che le quantità

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

e

$$\Delta x_0 := \frac{x - x_0}{x_0}$$

sono rispettivamente la variazione relativa della soluzione $u := f(x_0)$ e del dato $d := x_0$.

Chiamiamo **numero di condizionamento del calcolo di una funzione f in x_0** la quantità

$$K_f(x_0) := \left| \frac{x_0 f'(x_0)}{f(x_0)} \right|.$$

Poiché vale

$$|\Delta f(x_0)| \approx K_f(x_0) |\Delta x_0|$$

diciamo che $K_f(x_0)$ esprime il rapporto tra la variazione relativa subita dalla soluzione e la variazione relativa introdotta nel dato.

Nell'approssimare numericamente un problema fisico si commettono errori di quattro tipi diversi:

1. **errori sui dati**, riducibili aumentando l'accuratezza nelle misurazioni dei dati;
2. **errori dovuti al modello**, controllabili nella fase modellistica matematica, quando si passa dal problema fisico a quello matematico;
3. **errori di troncamento**, dovuti al fatto che in un calcolatore il passaggio al limite viene approssimato, essendo che un calcolatore esegue operazioni nel discreto;
4. **errori di arrotondamento**, dovuti alla rappresentazione finita dei calcolatori.

L'analisi numerica studia e controlla gli errori 3 e 4.

1.2. Aritmetica floating point

L'**insieme dei numeri macchina** è l'insieme

$$\mathcal{F}(\beta, t, L, U) = \left\{ \sigma(.a_1 a_2 ... a_t)_\beta \beta^e \right\} \cup \{0\}$$

e con il simbolo

$$\text{float}(x) \in \mathcal{F}(\beta, t, L, U)$$

il generico elemento dell'insieme, cioè il generico **numero macchina**.

Questo insieme è definito da:

- σ *segno* di $\text{float}(x)$;
- β *base* della rappresentazione;
- e *esponente* tale che $L \leq e \leq U$, con $L < 0$ e $U > 0$;

- t numero di *cifre significative*;
- $a_1 \neq 0$ e $0 \leq a_i \leq \beta - 1$;
- $m = (.a_1 a_2 ... a_t)_\beta = \frac{a_1}{\beta} + \frac{a_2}{\beta^2} + ... + \frac{a^t}{\beta^t}$ *mantissa.*

Vediamo una serie di osservazioni:
- $|\text{float}(x)| \in [\beta^{L-1}, (1 - \beta^{-t})\beta^U]$;
- in **Matlab** si ha $\beta = 2$, $t = 53$, $L = -1021$ e $U = 1024$;
- il risultato di un'operazione fra numeri macchina non è necessariamente un numero macchina.

Preso il numero reale

$$x = \sigma(.a_1 a_2 ... a_t a_{t+1} a_{t+2})_\beta \beta^e \in \mathbb{R}$$

distinguiamo i seguenti casi:
- se $L \leq e \leq U \wedge a_i = 0 \quad \forall i > t$ allora si ha la rappresentazione esatta di x, ovvero $\text{float}(x) = x$;
- se $e < L$ allora si ha **underflow**, ovvero $\text{float}(x) = 0$;
- se $e > U$ allora si ha **overflow**, ovvero $\text{float}(x) = \infty$;
- se $\exists i > t \mid a_i \neq 0$ allora ho due casi:
  - **troncamento**:

    $$\text{float}(x) = \sigma(.a_1 a_2 ... a_t)_\beta \beta^e;$$

  - **arrotondamento**:

    $$\sigma\left(\begin{cases} (.a_1 a_2 ... a_t)_\beta \beta^e & \text{se } 0 \leq a_{t+1} < \frac{\beta}{2} \\ (.a_1 a_2 ... (a_t + 1))_\beta \beta^e & \text{se } \frac{\beta}{2} \leq a_{t+1} \leq \beta - 1 \end{cases} \right).$$

Si può dimostrare che l'errore commesso approssimando un numero reale x con la sua rappresentazione macchina $\text{float}(x)$ è maggiorato da

$$\left| \frac{\text{float}(x) - x}{x} \right| \leq k\beta^{1-t},$$

con $k = 1$ per troncamento e $k = \frac{1}{2}$ per arrotondamento.

La quantità

$$\text{eps} = k\beta^{1-t}$$

è detta **precisione macchina** nel fissato sistema floating point. La precisione si può caratterizzare come il più piccolo numero macchina per cui vale

$$\text{float}(1 + \text{eps}) > 1.$$

2. Vettori e matrici

2.1. Vettori

Una tabella di $m \times n$ numeri reali disposti in m righe e n colonne del tipo

$$A = \begin{bmatrix} a_{11} & ... & a_1 n \\ \vdots & \ddots & \vdots \\ a_{m1} & ... & a_{mn} \end{bmatrix} = \left(a_{ij}\right) \mid i = 1, ..., m \quad j = 1, ..., n$$

si chiama **matrice** di m righe e n colonne. Ogni elemento a_{ij} ha un indice di riga i e un indice di colonna j che indicano, appunto, riga e colonna di A in cui si trova quell'elemento. Indichiamo con $\mathbb{R}^{m \times n}$ l'insieme delle matrici $m \times n$.

Chiamiamo **vettore colonna** di dimensione n una matrice $n \times 1$ formata da n righe e una sola colonna. Analogamente, il **vettore riga** è una matrice di dimensione $1 \times n$ formata da una sola riga e n colonne.

Con il termine **vettore** indicheremo un vettore colonna, e l'insieme dei vettori di dimensione n lo indichiamo con $\mathbb{R}^n$.

Usiamo vettori e matrici per rappresentare molte grandezze fisiche che non possono essere rappresentate come scalari, ma come vettori, come spostamento, velocità, accelerazione, eccetera.

2.1.1. Operazioni

Siano $a = (a_i), b = (b_i) \in \mathbb{R}^n$ due vettori. Chiamiamo **vettore somma** il vettore $c = (c_i) \in \mathbb{R}^n$ tale che

$$c_i = a_i + b_i \forall i = 1...n.$$

Geometricamente parlando, il vettore somma è la diagonale del parallelogramma avente due lati coincidenti con a e b (*la famosa regola del parallelogramma*).

La somma di vettori gode di alcune proprietà:

- **commutativa**: $\forall a, b \in \mathbb{R}^n \quad a + b = b + a$;
- **associativa**: $\forall a, b, c \in \mathbb{R}^n \quad (a + b) + c = a + (b + c)$;
- **esistenza del neutro**: il vettore

$$0 = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

  è l'**elemento neutro** della somma, cioè $\forall a \in \mathbb{R}^n \quad a + 0 = 0 + a = a$;
- **esistenza dell'opposto**: per ogni vettore $a \in \mathbb{R}^n$ esiste un altro vettore $b \in \mathbb{R}^n$ tale che $a + b = 0$; tale vettore b viene detto **vettore opposto** di a e si indica con $-a$.

Siano $a = (a_i) \in \mathbb{R}^n$ un vettore e $\beta \in \mathbb{R}$ uno scalare. Chiamiamo **prodotto vettore-scalare** il vettore $c = (c_i) \in \mathbb{R}^n$ tale che

$$c_i = \beta a_i \forall i = 1, ..., n.$$

Valgono le due proprietà distributive:

- $\forall \alpha \in \mathbb{R} \quad \forall a, b \in \mathbb{R}^n \quad \alpha(a + b) = \alpha a + \alpha b$;
- $\forall \alpha, \beta \in \mathbb{R} \quad \forall a \in \mathbb{R}^n \quad (\alpha + \beta)a = \alpha a + \beta a$.

Siano $a = (a_i), b = (b_i) \in \mathbb{R}^n$ due vettori. Chiamiamo **prodotto scalare** lo scalare $c = a \cdot b \in \mathbb{R}$ tale che

$$c = a \cdot b = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + ... + a_n b_n.$$

Diciamo che l'applicazione

$$\|\cdot\| : \mathbb{R}^n \longrightarrow \mathbb{R}^+ \cup \{0\}$$

è una **norma vettoriale** se valgono le seguenti condizioni:

1. $\|x\| \geq 0 \quad \forall x \in \mathbb{R}^n$ e $\|x\| = 0$ se e solo se $x = 0$;
2. $\|\alpha x\| = |\alpha| \|x\| \quad \forall \alpha \in \mathbb{R} \quad \forall x \in \mathbb{R}^n$;
3. $\|x + y\| \leq \|x\| + \|y\| \quad \forall x, y \in \mathbb{R}^n$.

Le norme più famose sono:

- **norma 1**, essa è tale che

$$\|x\|_1 = \sum_{i=1}^{n} |x_i| \quad \forall x \in \mathbb{R}^n;$$

- **norma euclidea** (*norma 2*), essa è tale che

$$\|x\|_2 = \left(\sum_{i=1}^{n} x_i^2 \right)^{\frac{1}{2}} \quad \forall x \in \mathbb{R}^n;$$

- **norma** ∞ (*norma del massimo*), essa è tale che

$$\|x\|_\infty = \max_{1 \leq i \leq n} |x_i| \quad \forall x \in \mathbb{R}^n.$$

2.2. Matrici

Una matrice si dice **quadrata** di ordine n se $m = n$. Una matrice quadrata è **triangolare superiore** (*inferiore*) se

$$a_{ij} = 0 \mid i > j \quad (i < j),$$

cioè se sono nulli gli elementi al di sotto (*sopra*) della diagonale principale a_{ii}.

Se valgono entrambe le definizioni di triangolare la matrice è detta **diagonale**.

Data la matrice $A = \left(a_{ij}\right) \in \mathbb{R}^{m \times n}$, chiamiamo **matrice trasposta** la matrice $A^T = \left(a_{ij}^T\right) \in \mathbb{R}^{n \times m}$ ottenuta dallo scambio delle righe e delle colonne di A, ovvero

$$a_{ij} = a_{ji}^T.$$

Sia A una matrice quadrata di ordine n, essa si dice **simmetrica** se $A = A^T$, ovvero $a_{ij} = a_{ji} \quad \forall i, j = 1, ..., n$.

2.2.1. Operazioni

Siano $A = \left(a_{ij}\right), B = \left(b_{ij}\right) \in \mathbb{R}^{m \times n}$ due matrici. Chiamiamo **matrice somma** la matrice $C = \left(c_{ij}\right) \in \mathbb{R}^{m \times n}$ tale che

$$c_{ij} = a_{ij} + b_{ij} \quad \forall i = 1, ..., m \quad \forall j = 1, ..., n.$$

Anche la somma di matrici gode di alcune proprietà:

- **commutativa**: $\forall A, B \in \mathbb{R}^{m \times n} \quad A + B = B + A$;
- **associativa**: $\forall A, B, C \in \mathbb{R}^{m \times n} \quad (A + B) + C = A + (B + C)$;
- **esistenza del neutro**: la matrice

$$0 = \begin{bmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{bmatrix}$$

è l'**elemento neutro** della somma, cioè $\forall A \in \mathbb{R}^{m \times n} \quad A + 0 = 0 + A = A$;

- **esistenza dell'opposto**: per ogni matrice $A \in \mathbb{R}^n$ esiste un'altra matrice $B \in \mathbb{R}^{m \times n}$ tale che $A + B = 0$; tale matrice B viene detta **matrice opposta** di A e si indica con $-A$.

Siano $A = \left(a_{ij}\right) \in \mathbb{R}^{m \times n}$ una matrice e $\beta \in \mathbb{R}$ uno scalare. Chiamiamo **prodotto matrice-scalare** la matrice $C = \left(c_{ij}\right) \in \mathbb{R}^{m \times n}$ tale che

$$c_{ij} = \beta a_{ij} \quad \forall i = 1, ..., m \quad \forall j = 1, ..., n.$$

Valgono le due proprietà distributive:

- $\forall \alpha \in \mathbb{R} \quad \forall A, B \in \mathbb{R}^{m \times n} \quad \alpha(A + B) = \alpha A + \alpha B$;
- $\forall \alpha, \beta \in \mathbb{R} \quad \forall A \in \mathbb{R}^{m \times n} \quad (\alpha + \beta)A = \alpha A + \beta A$.

Sia $A = \left(a_{ij}\right) \in \mathbb{R}^{m \times n}$ una matrice e $b = (b_i) \in \mathbb{R}^n$ un vettore. Chiamiamo **prodotto matrice-vettore** di A per b il vettore $c = (c_i) \in \mathbb{R}^m$ tale che

$$c_i = \sum_{j=1}^{n} a_{ij} b_j = a_{i1} b_1 + ... + a_{in} b_n \quad \forall i = 1, ..., m.$$

Siano $A = \left(a_{ij}\right) \in \mathbb{R}^{m \times n}$ e $B = \left(b_{ij}\right) \in \mathbb{R}^{n \times k}$ due matrici. Chiamiamo **prodotto matrice-matrice** di A per B la matrice $C = \left(c_{ij}\right) \in \mathbb{R}^{m \times k}$ tale che

$$c_{ij} = \sum_{t=1}^{n} a_{ik} b_{kj} = a_{i1} b_{1j} + ... + a_{in} b_{nj} \quad \forall i = 1, ..., m \quad \forall j = 1, ..., k.$$

Il prodotto di matrici in generale *non è commutativo*, cioè $A \cdot B \neq B \cdot A$.

Si chiama **matrice identità** di ordine n la matrice quadrata $I = \left(i_{kj}\right)$ di ordine n tale che

$$i_{kj} = \begin{cases} 1 & \text{se } k = j \\ 0 & \text{se } k \neq j \end{cases}$$

Si può dimostrare che $A \cdot I = I \cdot A = A$.

L'applicazione

$$\|\cdot\| : \mathbb{R}^{n \times n} \longrightarrow \mathbb{R}^+ \cup \{0\}$$

è una **norma matriciale** se valgono le seguenti condizioni:

1. $\|A\| \geq 0 \quad \forall A \in \mathbb{R}^{n \times n}$ e $\|A\| = 0$ se e solo se $A = 0$;
2. $\|\alpha A\| = |\alpha| \|A\| \quad \forall \alpha \in \mathbb{R} \quad \forall A \in \mathbb{R}^{n \times n}$;
3. $\|A + B\| \leq \|A\| + \|B\| \quad \forall A, B \in \mathbb{R}^{n \times n}$;
4. $\|A \cdot B\| \leq \|A\| \cdot \|B\| \quad \forall A, B \in \mathbb{R}^{n \times n}$.

Definiamo la **norma matriciale indotta** dalla norma vettoriale come la quantità

$$\|A\| = \sup\left\{ \frac{\|Ax\|}{\|x\|} \quad \forall x \in \mathbb{R}^n / \{0\} \right\}.$$

Abbiamo alcune norme particolari:

- **norma 1** (*calcolata colonna per colonna*), essa è tale che

$$\|A\|_1 = \max_{1\le j\le n} \sum_{i=1}^{n} |a_{ij}|;$$

- **norma** ∞ (*calcolata per riga*), essa è tale che

$$\|A\|_\infty = \max_{1\le i\le n} \sum_{j=1}^{n} |a_{ij}|.$$

2.2.2. Determinante

Sia A una matrice quadrata di ordine 2, ovvero

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}.$$

Si chiama **determinante** di A il numero reale

$$\det(A) := a_{11}a_{22} - a_{12}a_{21} \in \mathbb{R}.$$

Ora vediamo determinanti per matrici di ordine maggiore.

Siano A matrice quadrata di ordine n e a_{ij} il generico elemento; si chiama **complemento algebrico** di a_{ij} il numero reale

$$\mathrm{compl}(a_{ij}) := (-1)^{i+j} \det(A_{ij}),$$

dove la matrice A_{ij} è la matrice quadrata di ordine $n-1$ ottenuta da A eliminando la riga i e la colonna j.

Sia A una matrice quadrata di ordine n. Fissata una qualunque riga o colonna di A, il determinante di A si ottiene sommando il prodotto di ogni elemento di tale riga o colonna per il suo complemento algebrico.

Il calcolo del determinante è indipendente dalla riga o colonna scelta, quindi conviene fissare la riga o colonna con il maggior numero di zeri.

Il determinante gode di alcune proprietà:

- se A è *triangolare* allora $\det(A) = a_{11}a_{22}...a_{nn}$;
- se A ha una riga o una colonna di soli zeri allora $\det(A) = 0$;
- se A ha due righe o colonne uguali allora $\det(A) = 0$;
- vale il **Teorema di Binet**, ovvero se A, B sono due matrici quadrate dello stesso ordine allora $\det(A \cdot B) = \det(A) \cdot \det(B)$.

2.2.3. Applicazioni del determinante

Sia A una matrice quadrata di ordine n. Si dice che A è **invertibile** se esiste una matrice A^{-1} detta **matrice inversa** di A, quadrata di ordine n, tale che $A \cdot A^{-1} = A^{-1} \cdot A = I_n$.

Teorema 2.2.3.1 (*Teorema di invertibilità*): Sia A una matrice quadrata di ordine n, allora A è invertibile se e solo se

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

Teorema 2.2.3.2: Sia A una matrice quadrata di ordine due, cioè

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

e supponiamo $\det(A) \neq 0$. Allora

$$A^{-1} = \frac{1}{\det(A)} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix}.$$

Sia A una matrice $m \times n$ e $k \in \mathbb{N}$ con $k \leq \min(m, n)$. Si chiama **minore** di ordine k estratto da A il determinante di una qualunque sottomatrice quadrata di ordine k di A, ottenuta prendendo gli elementi comuni a k righe di k colonne di A. Si chiama **caratteristica o rango** di A ($\mathrm{rk}(A)$) l'ordine massimo dei minori non nulli che si possono estrarre da A.

In altre parole, $\mathrm{rk}(A) = r$ se esiste un minore di ordine r diverso da zero e se tutti i minori di ordine $r + 1$ sono nulli.

Osserviamo due fatti. Sia A una matrice $m \times n$ non nulla, allora:

- $\mathrm{rk}(A) \geq 1$;
- $\mathrm{rk}(A) \leq \min(m, n)$.

In poche parole

$$1 \leq \mathrm{rk}(A) \leq \min(m, n).$$

3. Sistemi lineari

3.1. Definizione

Un **sistema lineare** di m equazioni in n incognite $x_1, x_2, ..., x_n$ è un sistema formato da m equazioni lineari in $x_1, x_2, ..., x_n$, ossia

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + ... + a_1nx_n = b_1 \\ ... \\ a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n = b_m \end{cases}.$$

Possiamo suddividere il sistema lineari in più componenti:

- il vettore $x \in \mathbb{R}^n$ tale che $x = (x_i)$ si chiama **vettore soluzione**;
- la matrice $A \in \mathbb{R}^{m \times n}$ tale che $A = \left(a_{ij}\right)$ si chiama **matrice dei coefficienti** del sistema;
- il vettore $b \in \mathbb{R}^m$ tale che $b = (b_i)$ si chiama **vettore termine noto**;
- la matrice $M \in \mathbb{R}^{m \times (n+1)}$ tale che $M = (A \mid b)$, ottenuta accostando alle colonne di A il vettore b, si chiama **matrice completa** del sistema.

In forma compatta, dati la matrice $A \in \mathbb{R}^{m \times n}$ e il vettore $b \in \mathbb{R}^m$, il problema da risolvere è quello di trovare il vettore $x \in \mathbb{R}^n$ tale che

$$Ax = b.$$

Abbiamo tre possibili condizioni:

- **sistema impossibile**: il sistema non ammette soluzioni;
- **sistema possibile determinato**: il sistema ammette una e una sola soluzione;
- **sistema possibile indeterminato**: il sistema ammette infinite soluzioni.

Teorema 3.1.1 (*Teorema di Cramer*): Siano A una matrice quadrata di ordine n e $b \in \mathbb{R}^n$, allora il sistema lineare $Ax = b$ ammette una e una sola soluzione se e solo se

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

Se il determinante è uguale a 0 potremmo avere sia sistema impossibile sia sistema possibile indeterminato.

Teorema 3.1.2 (*Rouché-Capelli*): Siano A una matrice $m \times n$ e $b \in \mathbb{R}^m$, allora il sistema lineare $Ax = b$ ammette soluzione se e solo se

$$\mathrm{rk}(A) = \mathrm{rk}(A \mid b).$$

Osserviamo che se $\mathrm{rk}(A) = \mathrm{rk}(A \mid b)$, chiamato $r = \mathrm{rk}(A)$ possiamo avere:

- $r = n$ e quindi il sistema ammette una e una sola soluzione;
- $r < n$ e quindi il sistema ammette infinite soluzioni.

I metodi numerici per la risoluzione di sistemi lineari si dividono in:

- **metodi diretti**: in assenza di errori di arrotondamento restituiscono la soluzione in un numero finito di passi;
- **metodi iterativi**: la soluzione è ottenuta come limite di una successione di vettori soluzione di sistemi lineari più semplici.

3.2. Metodi diretti per sistemi lineari

3.2.1. Metodo delle sostituzioni in avanti

Se vediamo che la matrice dei coefficienti è *triangolare inferiore* possiamo risolvere il sistema *«a cascata»* a partire dalla prima equazione. In poche parole, risolviamo la prima equazione per la prima variabile, poi sostituisco il risultato nelle altre equazioni e ripeto le stesse operazioni per la variabile successiva, visto che ci troviamo nella stessa situazione della prima.

Sia $L = \left(l_{ij}\right)$ una matrice $n \times n$ triangolare inferiore e $b \in \mathbb{R}^n$. Consideriamo il sistema lineare $Lx = b$. Il metodo delle sostituzioni in avanti consiste in

$$x_i = \frac{1}{l_{ii}} \left(b_i - \sum_{j=1}^{i} l_{ij} x_j \right) \quad i = 1, ..., n.$$

Questo algoritmo ha complessità $O(n^2)$.

3.2.2. Metodo delle sostituzioni all'indietro

Se vediamo che la matrice dei coefficienti è *triangolare superiore* possiamo risolvere *«ad arrampicata»* a partire dall'ultima equazione. In poche parole, risolviamo l'ultima equazione per l'ultima variabile, poi sostituisco il risultato nelle altre equazioni e ripeto le stesse operazioni per la variabile precedente, visto che ci troviamo nella stessa situazione della prima.

Sia $U = \left(u_{ij}\right)$ una matrice $n \times n$ triangolare inferiore e $b \in \mathbb{R}^n$, consideriamo il sistema lineare $Ux = b$. Il metodo delle sostituzioni all'indietro consiste in

$$x_i = \frac{1}{u_{ii}} \left(b_i - \sum_{j=i}^{n} u_{ij} x_j \right) \quad i = n, ..., 1.$$

Questo algoritmo ha complessità $O(n^2)$.

3.2.3. Metodo di eliminazione gaussiana (MEG)

Se non abbiamo una matrice triangolare superiore o inferiore usiamo il **metodo di eliminazione gaussiana**: trasformiamo il sistema $Ax = b$ in un sistema equivalente triangolare superiore $Ux = \overline{b}$ mediante combinazioni lineari delle righe. Si risolve poi il sistema appena trovato con il metodo delle sostituzioni all'indietro.

L'algoritmo segue i seguenti passi:

1. pongo $A^{(0)} = A$ e $b^{(0)} = b$;
2. per costruire $A^{(t)}$ e $b^{(t)}$, con $1 \leq t \leq n$, a partire da $A^{(t-1)}$ e $b^{(t-1)}$ devo porre a zero gli elementi sulla colonna t a partire dalla riga $t + 1$ con:
    1. ricopio le prime t righe di $A^{(t-1)}$ nella prime t righe di $A^{(t)}$ e i primi t elementi di $b^{(t-1)}$ nei primi t elementi di $b^{(t)}$;
    2. per ogni riga successiva $i \geq t + 1$ calcolo il coefficiente $K_i = \frac{a_{it}^{(t-1)}}{a_{tt}^{(t-1)}}$;
    3. si modifica l'equazione i-esima modificando ogni coefficiente con se stesso meno il coefficiente per il valore della riga t-esima sulla stessa colonna; modificare l'equazione vuol dire modificare ogni cella della riga i-esima della matrice ma anche il vettore dei termini noti;
3. mi fermo quando $A^{(t)}$ è triangolare superiore.

Questo algoritmo ha complessità $O(n^3)$.

Il MEG durante la sua applicazione costruisce una fattorizzazione della matrice A in due matrici L e U rispettivamente triangolare inferiore e triangolare superiore tali che $LU = A$.

3.2.4. Fattorizzazione LU

La fattorizzazione sopra citata è detta **fattorizzazione LU** e una volta calcolata il sistema lineare $Ax = b$ può essere scritto come $LUx = b$ e può essere risolto in due step:

- $Ly = b$ sistema triangolare inferiore (*sostituzioni in avanti*);
- $Ux = y$ sistema triangolare superiore (*sostituzioni all'indietro*).

Il vantaggio che offre questa fattorizzazione è quello di risolvere sistemi triangolari che costano meno del MEG.

L'algoritmo di fattorizzazione segue i seguenti passi:

1. definiamo le matrici $U = A$ e $L = I_n$;
2. applichiamo MEG alla matrice U ma modificando al tempo stesso la matrice L: durante il calcolo del coefficiente K_i usando il valore $a_{it}^{(k-1)}$, mettiamo in l_{it} il coefficiente appena calcolato.

3.2.5. Fattorizzazione di Cholesky

Una matrice simmetrica $A \in \mathbb{R}^{n \times n}$ si dice **definita positiva** se

$$Ax \cdot x \geq 0 \forall x \in \mathbb{R}^n$$

e

$$Ax \cdot x = 0 \iff x = 0.$$

Lemma 3.2.5.1 (*Criterio di Sylvester*): Una matrice A simmetrica di ordine n è definita positiva se e solo se

$$\det(A_k) > 0 \quad k = 1, ..., n$$

con A_k sottomatrice principale di ordine k formata dalle prime k righe e colonne.

Teorema 3.2.5.1: Sia $A \in \mathbb{R}^{n \times n}$ simmetrica definita positiva. Allora esiste una matrice $R \in \mathbb{R}^{n \times n}$ triangolare superiore tale che

$$A = R^T R.$$

Tale fattorizzazione della matrice A è detta **fattorizzazione di Cholesky**.

Con questa fattorizzazione trasformiamo il sistema $Ax = b$ nel sistema $R^T Rx = b$, che andiamo a risolvere in due step:

1. $R^T y = b$ sistema triangolare inferiore (*sostituzioni in avanti*);
2. $Rx = y$ sistema triangolare superiore (*sostituzioni all'indietro*).

Se si devono risolvere più sistemi lineari con la stessa matrice A, conviene applicare la fattorizzazione di Cholesky per risolvere dei sistemi triangolari che costano meno del MEG. Inoltre, il tempo di calcolo della fattorizzazione è $\approx \frac{1}{3}n^3$, che è la metà della fattorizzazione LU ($\approx \frac{2}{3}n^3$).

3.3. Metodi iterativi per sistemi lineari

3.3.1. Autovalori e autovettori

Sia A una matrice quadrata di ordine n. Il numero $\lambda \in \mathbb{C}$ è detto **autovalore** di A se esiste un vettore $v \in \mathbb{C}^n \mid v \neq 0$ tale che

$$Av = \lambda v.$$

Il vettore è detto **autovettore** associato all'autovalore λ. L'insieme $\sigma(A)$ degli autovalori di A è detto **spettro** di A.

Lemma 3.3.1.1: L'autovalore λ è soluzione dell'equazione caratteristica

$$p_A(\lambda) := \det(A - \lambda I) = 0,$$

dove $p_A(\lambda)$ è detto **polinomio caratteristico**.

Dal teorema fondamentale dell'algebra segue che una matrice di ordine n ha n autovalori.

Vediamo alcune proprietà:

- una matrice è **singolare** se e solo se ha almeno un autovalore nullo;
- se A è simmetrica definita positiva allora gli autovalori di A sono tutti positivi;
- siano $\lambda_i(A) \quad i = 1, ..., n$ gli autovalori della matrice $A \in \mathbb{R}^{n \times n}$. Allora

$$\det(A) = \prod_{i=1}^{n} \lambda_i(A).$$

- $\mathrm{tr}(A) := \sum_{i=1}^{n} a_{ii} = \sum_{i=1}^{n} \lambda_i(A)$, con $\mathrm{tr}(A)$ **traccia** di A.

Sia A una matrice quadrata di ordine n, si chiama **raggio spettrale** di A ($\rho(A)$) il massimo valore assoluto degli autovalori di A, ovvero

$$\rho(A) := \max_{i=1,...,n} |\lambda_i(A)|.$$

Lemma 3.3.1.2: Sia A una matrice quadrata di ordine n, allora

$$\|A\|_2 = \sqrt{\rho(A^T A)}.$$

Siano A una matrice quadrata di ordine n non singolare e $\|\cdot\|$ una generica norma di matrice; si chiama **numero di condizionamento** della matrice A, e si indica con $K(A)$, la quantità scalare

$$K(A) = \|A\| \cdot \left\|A^{-1}\right\|.$$

Una matrice A si dice **sparsa** se ha un numero elevato di elementi $a_{ij} = 0$. Comunemente, una matrice quadrata di ordine n è ritenuta sparsa quando il numero di elementi diversi da zero è di ordine $O(n)$.

Può capitare che la fattorizzazione LU o la fattorizzazione di Cholesky di una matrice sparsa A generino due matrici piene. Questo fenomeno è detto **fill-in** (*riempimento*), e questo rappresenta un problema se le matrici sono di grandi dimensioni, rendendo la risoluzione del sistema lineare inefficiente.

Per matrici le sparse di grandi dimensioni i metodi iterativi possono essere più efficienti dei metodi diretti. Ma cosa sono i metodi iterativi?

Un **metodo iterativo** per la risoluzione del sistema lineare $Ax = b$ consiste nel costruire una successione di vettori $x^{(k)} \in \mathbb{R}^n \mid k \geq 0$ con la speranza che

$$\lim_{k \to \infty} x^{(k)} = x,$$

a partire da un vettore iniziale $x^{(0)}$ dato.

In generale, un metodo iterativo per la risoluzione del sistema lineare $Ax = b$ ha la forma

$$x^{(k+1)} = Bx^{(k)} + g$$

con $B \in \mathbb{R}^{n \times n}$ **matrice di iterazione** e $g \in \mathbb{R}^n$.

Teorema 3.3.1.1 (*Teorema di convergenza*): Un metodo iterativo nella forma descritta è convergente, cioè

$$\lim_{k \to \infty} x^{(k)} = x,$$

se e solo se

$$\rho(B) < 1,$$

dove $\rho(B)$ è il raggio spettrale della matrice B.

3.3.2. Metodo di Jacobi

Il **metodo di Jacobi** isola nell'i-esima equazione l'i-esima incognita e, a partire da un vettore $x^{(0)} \in \mathbb{R}^n$, genera i passi successivi $k \geq 0$ con il seguente iterazione:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1 \wedge j \neq i}^{n} a_{ij} x_j^{(k)} \right) \quad i = 1, ..., n.$$

In poche parole, isoliamo l'i-esima incognita nell'i-esima equazione e risolviamo ogni equazione sostituendo i risultati ottenuti al punto precedente.

Dato il sistema $Ax = b$ creiamo le matrici D, E, F tali che:

- D è diagonale e contiene la diagonale di A;
- E è triangolare inferiore, contiene gli elementi triangolari inferiori di A cambiati di segno e ha 0 sulla diagonale;
- F è triangolare superiore, contiene gli elementi triangolari superiori di A cambiati di segno e ha 0 sulla diagonale;

Notiamo che $A = D - E - F$. Chiamiamo **matrice di iterazione di Jacobi** la matrice

$$B_j = D^{-1}(E + F).$$

Si può verificare che questo metodo si scrive in forma compatta come

$$x^{(k+1)} = B_j x^{(k)} + D^{-1} b.$$

Grazie al teorema di convergenza, questo metodo converge se e solo se

$$\rho(B_j) < 1.$$

Gli autovalori di B_j sono i λ tali che

$$\det(\lambda D - E - F) = 0.$$

3.3.3. Metodo di Gauss-Seidel

Come prima, isoliamo l'i-esima incognita nell'i-esima equazione e partiamo da un vettore iniziale $x^{(0)}$. Il metodo di Gauss-Seidel calcola tutte le incognite $x_i^{(k+1)}$ utilizzando $x_j^{(k+1)} \quad \forall j < i$, altrimenti utilizza $x_t^{(k)} \quad \forall t \geq i$.

L'iterazione generica del metodo di Gauss-Seidel, dato il sistema lineare $Ax = b$ con $A \in \mathbb{R}^{n \times n}$ è

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

Come prima, dividiamo A nelle matrici D, E, F. Chiamiamo **matrice di iterazione di Gauss-Seidel** la matrice

$$B_{gs} = (D - E)^{-1} F.$$

Si può verificare che questo metodo si scrive in forma compatta come

$$x^{(k+1)} = B_{gs} x^{(k)} + (D - E)^{-1} b.$$

Grazie al teorema di convergenza, questo metodo converge se e solo se

$$\rho(B_{gs}) < 1.$$

Gli autovalori di B_{gs} sono i λ tali che

$$\det(\lambda(D - E) - F) = 0.$$

3.3.4. Osservazioni

Se inseriamo la condizione $a_{ii} \neq 0$ assicuriamo che il metodo si possa costruire. Non è però garantita la convergenza, quindi non è sempre vero che

$$\lim_{k \to \infty} x^{(k)} = x.$$

3.3.5. Verificare la convergenza

Sia A una matrice quadrata di ordine n, allora essa è a **dominanza diagonale stretta per righe** se

$$|a_{ii}| > \sum_{j=1 \wedge j \neq i}^{n} |a_{ij}| \quad \forall i = 1, ..., n.$$

Teorema 3.3.5.1: Sia $A \in \mathbb{R}^{n \times n}$ matrice a dominanza diagonale stretta per righe, allora i metodi di Jacobi e Gauss-Seidel applicati al sistema lineare $Ax = b$ sono convergenti.

Teorema 3.3.5.2: Sia $A \in \mathbb{R}^{n \times n}$ una matrice simmetrica definita positiva, allora il metodo di Gauss-Seidel converge.

3.3.6. Test d'arresto

Per arrestare l'esecuzione dei due metodi abbiamo due possibili alternative:

- **test del residuo**: fissata una tolleranza toll $\ll 1$, arrestiamo il metodo iterativo se

$$\frac{\left\|b - Ax^{(k)}\right\|}{\|b\|} < \text{toll};$$

- **test dell'incremento**: fissata una tolleranza $\text{toll} \ll 1$, arrestiamo il metodo iterativo se

$$\frac{\left\|x^{(k+1)} - x^{(k)}\right\|}{\left\|x^{(k)}\right\|} < \text{toll} \ .$$

Notiamo che, se il numero di condizionamento della matrice A è grande, allora la convergenza è lenta.

4. Interpolazione polinomiale

4.1. Polinomio interpolatore

Dati $N+1$ punti nel piano $(x_i, y_i) \quad i = 0, ..., N$, con y_i valori che possono essere sia sperimentali che valutazioni di una funzione $f(\cdot)$ non nota in x_i, trovare il polinomio di grado N $P_N(x)$ tale che

$$P_N(x_i) = y_i \quad i = 0, ..., N$$

è il problema del **polinomiale interpolatore**.

Indichiamo con $\mathbb{P}_N$ l'insieme dei polinomi di grado N e con $x_i \quad i = 0, ..., N$ i punti detti **nodi di interpolazione**.

Per risolvere questo problema scriviamo il generico polinomio di grado N e imponiamo il passaggio per i punti dati, ottenendo un sistema lineare che sappiamo risolvere con i metodi visti nel capitolo precedente.

Teorema 4.1.1: Dati $N+1$ punti distinti $x_0, ..., x_N$ e $N+1$ corrispondenti valori $y_0, ..., y_N$, esiste uno e un solo polinomio interpolatore $P_N(x)$ di grado N tale che

$$P_N(x_i) = y_i \quad \forall i = 0, ..., N.$$

Dimostrazione 4.1.1: Assumiamo per assurdo che esistano due polinomi $P_N(x)$ e $Q_N(x)$ in $\mathbb{P}_N$ tali che

$$P_N(x_i) = Q_N(x_i) = y_i \quad \forall i = 0, ..., N.$$

Ma allora $P_N(x) - Q_N(x) \in \mathbb{P}_N$ e $P_N(x_i) - Q_N(x_i) = 0 \quad \forall i = 0, ..., N$, cioè quel polinomio si annulla in $N+1$ punti distinti.

Questo implica che $P_N(x) - Q_N(x) = 0 \quad \forall x \in \mathbb{R}$ perché per il teorema fondamentale dell'algebra, l'unico polinomio di grado N che si annulla in $N+1$ punti distinti è il polinomio banale identicamente nullo, quindi $P_N(x) = Q_N(x)$ e quindi $P_N(x)$ è unico. ■

Per dimostrare l'esistenza si procede in maniera costruttiva tramite **metodo di Vandermonde** o **metodo di Lagrange**.

4.1.1. Metodo di Vandermonde

Il generico polinomio è

$$P_N(x) = \sum_{j=0}^{N} c_j x^j = c_0 + c_1 x + ... + c_N x^N.$$

Se imponiamo il passaggio per i punti otteniamo un sistema lineare del tipo

$$\begin{cases} c_0 + c_1 x_0 + ... + c_N x_0^N = y_0 \\ ... \\ c_0 + c_1 x_N + ... + c_N x_N^N = y_N \end{cases}.$$

Questo metodo è implementato dalla funzione *polyfit* di Matlab.

La matrice del sistema

$$V = \begin{bmatrix} 1 & x_0 & ... & x_0^N \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_N & ... & x_N^N \end{bmatrix}$$

è detta **matrice di Vandermonde**. Se i punti x_i sono distinti allora $\det(V) \neq 0$ e quindi la soluzione esiste ed è unica.

4.1.2. Metodo di Lagrange

Definiamo $N+1$ polinomi di Lagrange $L_i(x) \quad i = 0, ..., N$ che soddisfano le seguenti proprietà:

- $L_i(x) \in \mathbb{P}_N$;
- $L_i(x_j) = 0 \quad \forall i \quad \forall j = 0, ..., N \wedge i \neq j$;
- $L_i(x_j) = 1 \quad \forall i = 0, ..., N$.

In pratica sono tutti polinomi L_i che si annullano in tutti i valori che non sono x_i.

Ogni polinomio è quindi nella forma

$$L_i(x) = \prod_{j=0 \wedge j \neq i}^{N} \frac{x - x_j}{x_i - x_j} = \frac{(x - x_0) \cdot ... \cdot (x - x_{i-1}) \cdot (x - x_{i+1}) \cdot ... \cdot (x - x_N)}{(x_i - x_0) \cdot ... \cdot (x_i - x_{i-1}) \cdot (x_i - x_{i+1}) \cdot ... \cdot (x_i - x_N)}.$$

Il polinomio interpolatore è dato da

$$P_N(x) = \sum_{i=0}^{N} y_i L_i(x).$$

Infatti, $\forall k = 0, ..., N$ vale

$$P_N(x_k) = \sum_{i=0}^{N} L_i(x_k) = y_0 L_0(x_k) + ... + y_k L_k(x_k) + ... + y_N L_N(x_k) =$$
$$= 0 + ... + y_k \cdot 1 + ... + 0 = y_k.$$

4.1.3. Errore di interpolazione

Consideriamo $f : \mathbb{R} \longrightarrow \mathbb{R}$ una funzione e $N+1$ punti $(x_i, y_i) \quad i = 0, ..., N$ tali che $y_i = f(x_i)$, e sia $P_N(x)$ il polinomio che interpola i punti (x_i, y_i).

Dato $x \in \mathbb{R}$, chiamiamo **errore di interpolazione** nel punto x la quantità

$$|f(x) - P_N(x)|.$$

Teorema 4.1.3.1: Siano:

- $x_0, ..., x_N$ un insieme di $N+1$ nodi distinti;
- $x \neq x_i \quad \forall i = 0, ..., N$;
- $f \in C^{N+1}(I_x)$, dove I_x più piccolo intervallo chiuso e limitato contenente i nodi $x_0, ..., x_N, x$.

Allora l'errore di interpolazione nel punto x è dato da

$$f(x) - P_N(x) = \frac{\omega(x)}{(N+1)!} f^{(N+1)}(\xi),$$

con $\xi \in I_x$ e

$$\omega(x) = (x - x_0) \cdot ... \cdot (x - x_N).$$

Corollario 4.1.3.1.1: Nelle ipotesi del teorema precedente si ha

$$|f(x) - P_N(x)| \leq \frac{\max_{t \in I_x} |\omega(t)|}{(N+1)!} \max_{t \in I_x} \left| f^{(N+1)}(t) \right|.$$

In generale non si può dedurre dal teorema e dal corollario che l'errore tende a 0 per $N \to \infty$. Infatti esistono funzioni per le quali l'errore può essere infinito, ossia

$$\lim_{n \to \infty} \max_{x \in I_x} |f(x) - P_N(x)| = +\infty.$$

Una funzione che ha questo comportamento è il **controesempio di Runge**, ovvero interpoliamo $f(x) = \frac{1}{1+x^2}$ nell'intervallo $[-5, 5]$ su nodi equispaziati. Se $N \to \infty$ allora l'errore cresce.

Una soluzione è utilizzare i **nodi di Chebishev**, definiti:

- sull'intervallo $[-1, 1]$ da

$$x_i = \cos\left(\pi \frac{2i+1}{2(N+1)}\right) \quad i = 0, ..., N;$$

- sul generico intervallo $[a, b]$ da

$$x_i = \frac{a+b}{2} + (b-a, 2) \cos\left(\pi \frac{2i+1}{2(N+1)}\right) \quad i = 0, ..., N.$$

4.2. Retta di regressione

Dati $N+1$ punti $(x_i, y_i) \quad i = 0, ..., N$ dove eventualmente $y_i = f(x_i)$, vogliamo trovare la retta $R(x) = a_0 + a_1 x$ che renda minima la funzione

$$E(a_0, a_1) = \sum_{i=0}^{N} (y_i - R(x_i))^2 = \sum_{i=0}^{N} (y_i - (a_0 + a_1 x_1))^2$$

al variare dei coefficienti a_0, a_1.

Diciamo che $R(x)$ approssima l'insieme dei dati **nel senso dei minimi quadrati** e questa retta è la **retta dei minimi quadrati** o **retta di regressione**.

Il minimo della funzione $E(a_0, a_1)$ si ottiene imponendo le condizioni

$$\begin{cases} \frac{\partial E(a_0, a_1)}{\partial a_0} = 0 \\ \frac{\partial E(a_0, a_1)}{\partial a_1} = 0 \end{cases}.$$

Svolgendo i conti abbiamo

$$\begin{cases} \sum_{i=0}^{N} 2(y_i - a_0 - a_1 x_i)(-1) = 0 \\ \sum_{i=0}^{N} 2(y_i - a_0 - a_1 x_1)(-x_i) = 0 \end{cases}.$$

Dobbiamo quindi risolvere il sistema lineare

$$\begin{cases} (N+1)a_0 + \left(\sum_{i=0}^{N} x_i\right)a_1 = \sum_{i=0}^{N} y_i \\ \left(\sum_{i=0}^{N} x_i\right)a_0 + \left(\sum_{i=0}^{N} x_i^2\right)a_1 = \sum_{i=0}^{N} x_i y_i \end{cases}.$$

Tale sistema è detto **sistema delle equazioni normali**.

4.3. Spline lineare

Dato un insieme di punti $(x_i, y_i) \quad i = 0, ..., N$ con $a = x_0 < x_1 < ... < x_n = b$, una **spline lineare interpolante** è una funzione $S^1(x) : [a, b] \longrightarrow \mathbb{R}$ tale che:

- S^1 è un polinomio di grado 1 su ogni sotto-intervallo $[x_{i-1}, x_i] \quad i = 1, ..., N$;
- S^1 è continua su $[a, b]$;
- $S^1(x_i) = y_i \quad i = 0, ..., N$.

La possiamo vedere come una *funzione a tratti* formata da N funzioni lineari, ognuna delle quali passa per due punti consecutivi.

4.3.1. Errore di interpolazione

Consideriamo una funzione $f : \mathbb{R} \longrightarrow \mathbb{R}$ e $N + 1$ punti $(x_i, y_i) i = 0, ..., N$ con $f(x_i) = y_i$. Sia $S^1(x)$ la spline lineare che interpola i punti (x_i, y_i). Dato $x \in \mathbb{R}$ chiamiamo **errore di interpolazione** nel punto x la quantità

$$\left|f(x) - S^1(x)\right|.$$

Teorema 4.3.1.1: Sia $f \in C^2([a, b])$, allora

$$\max_{x\in[a,b]} \left|f(x) - S^1(x)\right| \leq \frac{1}{8}h^2 \max_{x\in[a,b]} \left|f^{(2)}(x)\right|,$$

con

$$h = \max_{0\leq i\leq N-1} (x_{i+1} - x_i).$$

5. Integrazione numerica

Data la funzione $f : [a, b] \longrightarrow \mathbb{R}$, vogliamo calcolare l'integrale definito

$$I(f) = \int_a^b f(x)\,\mathrm{d}x.$$

In generale non possiamo calcolare $I(f)$ per via analitica (*esempio: funzione gaussiana*), ma possiamo solo approssimarlo numericamente tramite formule di quadratura.

5.1. Formule di quadratura semplici

Si chiama **formula di quadratura** una formula del tipo

$$I^{\sim}(f) = \sum_{i=1}^{n} \alpha_i f(x_i)$$

che approssima l'integrale $I(f) = \int_a^b f(x)dx$ mediante una combinazione lineare di valori della funzione in opportuni punti x_i, detti **nodi di quadratura**, moltiplicati per opportuni coefficienti α_i, detti **pesi di quadratura**.

Per costruire formule di quadratura possiamo approssimare l'integrale $I(f)$ con l'integrale di un polinomio P che interpola la funzione f in un determinato insieme di nodi nell'intervallo $[a, b]$, cioè

$$I(f) \approx I^{\sim}(f) := I(P) = \int_a^b P(x)dx.$$

Al variare del numero di nodi di interpolazione e della loro posizione avremo diverse formule di quadratura, dette **di tipo interpolatorio**.

Si chiama **grado di precisione** di una formula di quadratura il massimo intero $r \geq 0$ tale che

$$I^{\sim}(P) = I(P) \quad \forall P \in \mathbb{P}_r.$$

Lemma 5.1.1: Una formula di quadratura ha grado di precisione r se e solo se

$$I^{\sim}\left(x^k\right) = I\left(x^k\right) \quad \forall k = 0, ..., r.$$

5.1.1. Formula del punto medio

La **formula del punto medio** si ottiene scegliendo il polinomio di grado 0 che interpola $f(x)$ nel punto medio dell'intervallo $[a, b]$, cioè

$$I_{PM}^{\sim}(f) := (b-a) f\left(\frac{a+b}{2}\right).$$

Abbiamo quindi:

- un nodo di quadratura $x_1 = \frac{a+b}{2}$;
- un peso di quadratura $\alpha_1 = b - a$.

In poche parole, stiamo approssimando l'integrale con un rettangolo di base $b - a$ e altezza $f\left(\frac{a+b}{2}\right)$.

Si dimostra che l'errore di questa formula è

$$I(f) - I_{PM}^{\sim}(f) = \frac{(b-a)^3}{24} f^{(2)}(t) \quad t \in (a, b)$$

se $f \in C^2([a,b])$.

Questa formula ha grado di precisione 1:

- se $k = 0$ allora $f(x) = x^0 = 1$ e quindi

$$I(f) = I(1) = \int_a^b 1dx = b - a$$

$$I_{PM}^{\sim}(f) = I_{PM}^{\sim}(1) = (b-a)\underbrace{f\left(\frac{a+b}{2}\right)}_{\text{funzione banale}} = b - a;$$

- se $k = 1$ allora $f(x) = x^1 = x$ e quindi

$$I(f) = I(x) = \int_a^b x dx = \frac{b^2 - a^2}{2}$$

$$I_{PM}^{\sim}(f) = I_{PM}^{\sim}(x) = (b-a)\underbrace{f\left(\frac{a+b}{2}\right)}_{\text{identità}} = (b-a)\frac{a+b}{2} = \frac{b^2-a^2}{2};$$

- se $k = 2$ allora $f(x) = x^2$ e quindi

$$I(f) = I(x^2) = \int_a^b x^2 dx = \frac{b^3 - a^3}{3}$$

$$I_{PM}^{\sim}(f) = I_{PM}^{\sim}(x^2) = (b-a)\underbrace{f\left(\frac{a+b}{2}\right)}_{\text{quadrato}} = (b-a)\frac{(a+b)^2}{4} = \frac{a^2b + b^3 + 2ab^2 - a^3 - ab^2 - 2a^2b}{4}.$$

Vista questa dimostrazione, la formula del punto medio ha grado di precisione 1.

5.1.2. Formula del trapezio

La **formula del trapezio** si ottiene scegliendo il polinomio di grado 1 che interpola $f(x)$ negli estremi dell'intervallo $[a,b]$, cioè

$$I_T^{\sim}(f) \coloneqq \frac{b-a}{2}(f(a) + f(b)).$$

Abbiamo quindi:

- due nodi di quadratura $x_1 = a$ e $x_2 = b$;
- due pesi di quadratura $\alpha_1 = \alpha_2 = \frac{b-a}{2}$.

In poche parole, stiamo approssimando l'integrale con un trapezio di basi $f(a), f(b)$ e altezza $b - a$.

Si dimostra che l'errore di questa formula è

$$I(f) - I_T^{\sim}(f) = -\frac{(b-a^3)}{12} f^{(2)}(f) \quad t \in (a,b)$$

se $f \in C^2([a,b])$.

Non lo dimostriamo, ma la formula del trapezio ha grado di precisione 1.

5.1.3. Formula di Cavalieri-Simpson

La **formula di Cavalieri-Simpson** si ottiene scegliendo il polinomio di grado 2 che interpola $f(x)$ negli estremi e nel punto medio dell'intervallo $[a,b]$, ovvero

$$I_{CS}^{\sim}(f) \coloneqq \frac{b-a}{6}\left(f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right).$$

Abbiamo quindi:

- tre nodi di quadratura $x_1 = a$, $x_2 = \frac{a+b}{2}$ e $x_3 = b$;
- due pesi di quadratura $\alpha_1 = \alpha_3 = \frac{b-a}{6}$ e $\alpha_2 = \frac{2(b-a)}{3}$.

In poche parole, stiamo approssimando l'integrale con un parabola che passa negli estremi e nel punto medio dell'intervallo $[a, b]$.

Si dimostra che l'errore di questa formula è

$$I(f) - I_{CS}^{\sim}(f) = -\frac{(b-a)^5}{2880}f^{(4)}(t) \quad t \in (a,b)$$

se $f \in C^4([a,b])$.

Non lo dimostriamo, ma la formula del trapezio ha grado di precisione 3.

5.2. Formule di quadratura composite

Le **formule di quadratura composite** consistono in:

- introdurre una suddivisione dell'intervallo di integrazione $[a, b]$ in sotto-intervalli;
- utilizzando la proprietà additiva dell'integrale, scrivere quest'ultimo come una somma di integrali definiti su ciascun intervallo della suddivisione;
- approssimare tali integrali definiti mediante formule di quadratura semplici.

Da qui in poi siano:

- M il numero di sotto-intervalli;
- $H = \frac{b-a}{M}$ ampiezza dei sotto-intervalli;
- $a_i = a + iH \quad i = 0, ..., M \quad a_0 = a \wedge a_M = b$ estremi dei sotto-intervalli.

5.2.1. Formula del punto medio composita

La **formula del punto medio composita** approssima con la formula

$$I_{PM}^{\tilde{C}}(f) = \sum_{i=1}^{M} Hf\left(\frac{a_{i-1}+a_i}{2}\right).$$

L'errore nella *formula classica* è

$$I(f) - I_{PM}^{\tilde{C}}(f) = \frac{b-a}{24}H^2 f^{(2)}(\eta) \quad \eta \in (a,b).$$

L'errore nella *formula asintotica* è

$$I(f) - I_{PM}^{\tilde{C}}(f) = \frac{H^2}{24}(f'(b) - f'(a)) \quad \eta \in (a,b).$$

5.2.2. Formula del trapezio composita

La **formula del trapezio composita** approssima con la formula

$$I_{T}^{\tilde{C}}(f) = \sum_{i=1}^{M} \frac{H}{2}(f(a_{i-1}) + f(a_i)).$$

L'errore nella *formula classica* è

$$I(f) - I_T^{\tilde{C}} = -\frac{b-a}{12} H^2 f^{(2)}(\eta) \quad \eta \in (a,b).$$

L'errore nella *formula asintotica* è

$$I(f) - I_T^{\tilde{C}} = -\frac{H^2}{12}(f'(b) - f'(a)) \quad \eta \in (a,b).$$

5.2.3. Formula di Cavalieri-Simpson composita

La **formula di Cavalieri-Simpson composita** approssima con la formula

$$I_{CS}^{\tilde{C}} = \sum_{i=1}^{M} \frac{H}{6}\left(f(a_{i-1}) + 4f\left(\frac{a_{i-1}+a_i}{2}\right) + f(a_i)\right).$$

L'errore nella *formula classica* è

$$I(f) - I_{CS}^{\tilde{C}} = -\frac{b-a}{2880} H^4 f^{(4)}(\eta) \quad \eta \in (a,b).$$

L'errore nella *formula asintotica* è

$$I(f) - I_{CS}^{\tilde{C}} = -\frac{H^4}{2880}\big(f^{(3)}(b) - f^{(3)}(a)\big).$$

6. Zeri di funzione

Data una funzione $f : [a, b] \longrightarrow \mathbb{R}$ continua e tale che $f(a)f(b) < 0$, vogliamo trovare $\alpha \in (a, b)$ tale che $f(\alpha) = 0$. In generale il valore α non riusciamo a calcolarlo per via analitica (*esempio: equazioni non lineari*), ma possiamo solo approssimarlo numericamente.

6.1. Teorema degli zeri

Teorema 6.1.1 (*Teorema degli zeri*): Sia $f : [a, b] \longrightarrow \mathbb{R}$ continua in $[a, b]$ e tale che

$$f(a)f(b) < 0.$$

Allora esiste $\alpha \in (a, b)$ tale che

$$f(\alpha) = 0.$$

I metodi numerici per la ricerca degli zeri sono in generale iterativi, quindi costruiremo una serie di valori x_k con la speranza che

$$\lim_{k\to\infty} x_k = \alpha.$$

Nella pratica ci fermeremo ad un passo $\hat{k}$ tale che $x_{\hat{k}}$ sia vicino ad α.

6.2. Metodo di bisezione

Il **metodo di bisezione** segue i seguenti passi:

1. siano $a_0 = a$ e $b_0 = b$;
2. per ogni $k \geq 1$ eseguo:
    1. calcolo $x_k = \frac{a_{k-1}+b_{k-1}}{2}$
    2. controllo $f(a)f(x_k)$:
        - se il risultato è negativo restringo l'estremo destro a $b = x_k$;
        - se il risultato è positivo restringo l'estremo sinistro ad $a = x_k$;
    3. ricomincio dal punto 2.1.

Al k-esimo passo l'errore commesso è

$$|x_k - \alpha| < \frac{b-a}{2^k}.$$

Vantaggi	Svantaggi
Converge sempre	Convergenza lenta
Robusto	Una buona approssimazione la si raggiunge lentamente

6.3. Metodo di Newton

Il **metodo di Newton** segue i seguenti passi:

1. sia x_0 un punto;
2. per ogni $k \geq 1$ eseguo:
    1. calcolo la retta tangente in x_{k-1};
    2. cerco lo zero di questa retta tangente;
    3. sia x_k lo zero appena trovato;
    4. ricomincio dal punto 2.1.

Questo è l'algoritmo *«formale»* per il metodo di Newton. Vediamo quello più *«applicato»*:

1. sia x_0 un punto;
2. per ogni $k \geq 1$ eseguo:
    1. calcolo $x_k = x_{k-1} - \frac{f(x_0)}{f'(x_0)}$
    2. ricomincio dal punto 2.1.

Sia $r_{k-1}(x)$ la retta tangente in x_{k-1}. La sua definizione analitica è la seguente:

$$r_{k-1}(x) = f'(x_{k-1})(x - x_{k-1}) + f(x_{k-1}).$$

Vogliamo lo zero di questa retta, ovvero vogliamo trovare x tale che

$$f'(x_{k-1})(x - x_{k-1}) + f(x_{k-1}) = 0.$$

Ma allora

$$\begin{aligned} xf'(x_{k-1}) - x_{k-1}f'(x_{k-1}) &= -f(x_{k-1}) \\ xf'(x_{k-1}) &= x_{k-1}f'(x_{k-1}) - f(x_{k-1}) \\ x &= x_{k-1} - \frac{f(x_{k-1})}{f'(x_{k-1})}. \end{aligned}$$

Il valore di x appena trovato lo chiamiamo x_k.

Vantaggi	**Svantaggi**
La convergenza è quadratica (*veloce*): $\lvert x_{k+1} - \alpha \rvert \approx \lvert x_k - \alpha \rvert^2$	La convergenza dipende dalla scelta di x_0: se esso non è sufficientemente vicino ad α il metodo può non convergere

Teorema 6.3.1: Supponiamo

- $f \in C^2([a,b])$ (*regolarità*);
- $f'(x) \neq 0 \quad \forall x \in [a,b]$ (*monotonia stretta*);
- $f''(x) \neq 0 \quad \forall x \in [a,b]$ (*non cambia convessità*).

Chiamiamo **estremo di Fourier** x_0 l'unico punto tra a e b tale che

$$f(x_0)f''(x_0) > 0.$$

Allora il metodo di Newton, innescato con dato iniziale x_0 estremo di Fourier, è convergente con convergenza quadratica.

Come **test d'arresto** abbiamo due possibilità:

- **test del residuo**: fissata una tolleranza toll $\ll 1$, arrestiamo il metodo iterativo se

$$\frac{|f(x_k)|}{|f(x_0)|} < \text{toll};$$

- **test dell'incremento**: fissata una tolleranza toll $\ll 1$ arrestiamo il metodi iterativo se

$$\frac{|x_{k+1} - x_k|}{|x_k|} < \text{toll}\ .$$

7. Metodi numerici per equazioni differenziali ordinarie

Consideriamo il problema di Cauchy

$$\begin{cases} \frac{dy(t)}{dt} = f(t, y(t)) \quad t \in (t_0, T) \\ y(t_0) = y_0 \end{cases}.$$

Supponiamo che $f = f(t, y) : (t_0) \times \mathbb{R} \longrightarrow \mathbb{R}$ sia **Lipschitziana** rispetto a y e **uniformemente Lipschitziana** rispetto a t, cioè

$$\exists L > 0 \mid |f(t, y_1) - f(t, y_2)| \leq L|y_1 - y_2| \quad \forall t \in (t_0, T) \quad \forall y_1, y_2 \in \mathbb{R}.$$

Sotto queste ipotesi, il problema di Cauchy ammette una e una sola soluzione.

Introduciamo $N + 1$ **nodi di discretizzazione** in $[t_0, T]$, ovvero

$$h > 0 \quad t_j = t_0 + jh \quad j = 0, ..., N \quad t_N \leq T.$$

Denotiamo con y_j la soluzione esatta $y(\cdot)$ valutata in t_j, ovvero $y_j = y(t_j) \quad j = 0, ..., N$.

Un metodo numerico per l'approssimazione del problema di Cauchy è un algoritmo che costruisce $N + 1$ valori reali u_j che approssimano $y_j \quad \forall j = 0, ..., N$, ovvero $u_j \approx y_j$.

Un metodo numerico per l'approssimazione di Cauchy è detto **metodo ad un passo** se $\forall n \geq 0$ allora u_{n+1} dipende solo da u_n e non da u_j, per $j < n$. In caso contrario il metodo è detto **multistep**.

Un metodo numerico per l'approssimazione di Cauchy è detto **esplicito** se $\forall n \geq 0$ u_{n+1} si calcola come funzione dei passi precedenti u_j per $j \leq n$. In caso contrario il metodo è detto **implicito**, ovvero se $\forall n \geq 0$ u_{n+1} dipende implicitamente da se stesso attraverso la funzione f.

7.1. Metodo di Eulero

Abbiamo due versioni per il **metodo di Eulero**:

- **esplicito**: posto $u_0 = y_0$ allora

  $\forall n \geq 0 \quad u_{n+1} = u_n + hf(t_n, u_n)$;

- **implicito**: posto $u_0 = y_0$ allora

  $\forall n \geq 0 \quad u_{n+1} = u_n + hf(t_{n+1}, u_{n+1})$.

Nel caso di Eulero implicito ad ogni passo dobbiamo risolvere un'equazione non lineare, ad esempio tramite il metodo di Newton.

7.2. Metodo di Crank-Nicolson

Il **metodo di Crank-Nicolson** ha una sola versione: posto $u_0 = y_0$ allora

$$\forall n \geq 0 \quad u_{n+1} = u_n + \frac{h}{2}(f(t_n, u_n) + f(t_{n+1}, u_{n+1})).$$

7.3. Metodo di Heun

Il **metodo di Heun** ha una sola versione: posto $u_0 = y_0$ allora $\forall n \geq 0$ calcolo

$$u^*_{n+1} = u_n + hf(t_n, u_n)$$
$$u_{n+1} = u_n + \frac{h}{2}(f(t_n, u_n) + f(t_{n+1}, u^*_{n+1})).$$

7.4. Consistenza, convergenza e stabilità

La forma generale di un metodo esplicito ad un passo è

$$u_{n+1} = u_n + h\phi(t_n, u_n, f(t_n, u_n), h),$$

dove ϕ è detta **funzione incrementale**.

Sia $y(\cdot)$ la soluzione esatta di Cauchy. Poniamo

$$\varepsilon_{n+1} = y_{n+1} - y_n - h\phi(t_n, y_n, f(t_n, y_n), h) \quad 0 \leq n \leq N-1.$$

La quantità ε_{n+1} è l'errore che si commette pretendendo che la soluzione esatta soddisfi lo schema numerico.

Si chiama **errore di troncamento locale** la quantità

$$\tau_{n+1}(h) = \frac{\varepsilon_{n+1}}{h}.$$

Si chiama **errore di troncamento globale** la quantità

$$\tau(h) = \max_{0 \leq n \leq N-1} \tau_{n+1}(h).$$

Un metodo numerico è **consistente** se

$$\lim_{h \to 0} \tau(h) = 0.$$

Un metodo numerico è **consistente di ordine** p se

$$\tau(h) = O(h^p).$$

Un metodo numerico è detto **zero-stabile** se, in un dato intervallo limitato (t_0, T), piccole perturbazioni sui dati producono piccole perturbazioni sulla soluzione approssimata, per $h \to 0$.

Un metodo numerico è detto **convergente di ordine** p se

$$\exists C > 0 \mid |u_n - y_n| \leq Ch^p \quad 0 \leq n \leq N.$$

Teorema 7.4.1: Un metodo numerico è convergente se e solo se è consistente e zero-stabile.

Consideriamo ora il problema modello

$$\begin{cases} \frac{dy(t)}{dt} = -\lambda y(t) & t \in (0, \infty) \quad \lambda > 0 \\ y(0) = 1 \end{cases}$$

la cui soluzione esatta è $y(t) = e^{-\lambda t}$.

Un metodo numerico è detto **assolutamente stabile** se, applicato al problema modello, allora

$$u_n \to 0 \quad \text{se } t_n \to \infty.$$

Vediamo che proprietà hanno i metodi che abbiamo visto:

- Eulero esplicito è assolutamente stabile se e solo se $h < \frac{2}{\lambda}$;
- Eulero implicito è incondizionatamente assolutamente stabile;
- Heun è assolutamente stabile se e solo se $h < \frac{2}{\lambda}$;
- Crank-Nicolson è incondizionatamente assolutamente stabile.

8. Metodi numerici per sistemi di equazioni differenziali ordinarie

Dati $f(\cdot,\cdot) : (t_0, T) \times \mathbb{R}^n \longrightarrow \mathbb{R}^n$ e $y_0 \in \mathbb{R}^n$, vogliamo trovare $y(\cdot) : (t_0, T) \longrightarrow \mathbb{R}^n$ tale che

$$\begin{cases} \frac{dy(t)}{dt} = f(t, y(t)) \quad t \in (t_0, T) \\ y(t_0) = y^0 \end{cases},$$

con

$$f(t, y(t)) = \begin{pmatrix} f_1(t, y(t)) \\ ... \\ f_n(t, y(t)) \end{pmatrix} \quad | \quad y_0 = \begin{pmatrix} y_1^0 \\ ... \\ y_n^0 \end{pmatrix} \quad | \quad y(t) = \begin{pmatrix} y_1(t) \\ ... \\ y_n(t) \end{pmatrix}.$$

8.1. Metodo di Eulero esplicito

Per il **metodo di Eulero esplicito**, siano $u_1^0 = y_1^0, ..., u_n^0 = y_n^0$ e $h > 0$ un passo. Allora

$$\begin{cases} u_1^k = u_1^{k-1} + h f_1\big(t_{k-1}, u_1^{k-1}, ..., u_n^{k-1}\big) \\ ... \\ u_n^k = u_n^{k-1} + h f_n\big(t_{k-1}, u_1^{k-1}, ..., u_n^{k-1}\big) \end{cases} \quad \forall k \geq 1.$$

8.2. Metodo di Eulero implicito

Per il **metodo di Eulero implicito**, siano $u_1^0 = y_1^0, ..., u_n^0 = y_n^0$ e $h > 0$ un passo. Allora

$$\begin{cases} u_1^k = u_1^{k-1} + h f_1\big(t_{k-1}, u_1^k, ..., u_n^k\big) \\ ... \\ u_n^k = u_n^{k-1} + h f_n\big(t_{k-1}, u_1^k, ..., u_n^k\big) \end{cases} \quad \forall k \geq 1.$$

In questo metodo, ad ogni passo temporale per trovare i valori di $u_1^k, ..., u_n^k$ bisogna risolvere in generale un **sistema algebrico non lineare**.

9. Come fare gli esercizi proposti nei TDE

Le domande viste per ora nei TDE riguardano un piccolo insieme di argomenti rispetto a quelli totali visti nel corso, quindi scriviamo qua qualche consiglio su come fare questi esercizi.

9.1. Sistemi lineari

Qua abbiamo quattro domande principali:

- **risolvere tramite MEG un sistema lineare**: lo dice già la consegna, applicare il MEG;
- **discutere al variare di un parametro la risolubilità di un sistema**: abbiamo due modi per risolvere questo esercizio:
  - applicare il MEG e quando arriviamo ad avere il parametro in un papabile pivot studiamo quando il pivot è diverso da 0 e quando è uguale;
  - applicare il teorema di Cramer, calcolando il determinante; questo qua possiamo farlo solo se la matrice è quadrata;
- **calcolare la fattorizzazione LU**: applicare il MEG tenendo traccia dei coefficienti in una seconda matrice (*che è della fattorizzazione*);
- **calcolare alcune iterazioni dei metodi iterativi**: applicare i passi iterativi e trovare le componenti del vettore richiesto.

9.2. Interpolazione polinomiale

Qua abbiamo tre domande principali:

- **calcolare il polinomio interpolatore di un certo grado con il metodo di Lagrange**: calcolare CON MOLTA CALMA i singoli polinomi di Lagrange e poi calcolare il polinomio risultato;
- **calcolare la retta di regressione**: impostare il sistema delle equazioni normali;
- **stimare l'errore di interpolazione di una spline lineare**: usare il teorema che dà un upper bound all'errore di interpolazione.

9.3. Integrazione numerica

Qua abbiamo due domande principali:

- **formula di quadratura di un certo grado di precisione**: verificare per tutti i gradi t da 0 fino al grado richiesto, che la formula ha grado di precisione t;
- **minimo numero di sotto-intervalli affinché si scenda sotto un certo errore**: usare (*di solito*) la formula asintotica IN MODULO per ricavare M.

9.4. Zeri di funzione

Qua abbiamo una sola domanda:

- **calcolare gli estremi di Fourier e poi applicare il metodo di Newton**: lo dice la consegna.

9.5. Equazioni differenziali ordinarie

Infine, anche qua abbiamo una sola domanda:

- **calcolare la prima iterazione dei metodi iterativi**: in base al metodo richiesto calcolare FACENDO ATTENZIONE AI CONTI la prima iterazione.

Parte II — Laboratorio

1. Introduzione a Matlab

1.1. Solite nozioni di ogni linguaggio di programmazione

Ogni variabile in **Matlab** è una **matrice**, anche gli scalari, variabili dalle quali partiremo oggi.

L'**assegnamento** è quello classico di ogni linguaggio di programmazione, ormai mi sto annoiando. Ogni volta che si fa un assegnamento l'espressione viene valutata, il risultato viene salvato nella variabile e viene mostrato nel prompt il risultato. Per evitare la stampa della valutazione si usa il simbolo `;` alla fine dell'espressione. Se un'espressione viene valutata ma non viene assegnata a nessuna variabile, il risultato viene salvato nella variabile `ans`.

Con il comando `who` vengono mostrate le variabili presenti in memoria, mentre con il comando `whos` vengono stampate informazioni aggiuntive, ad esempio la dimensione (*vedremo poi la nozione di dimensione*), lo spazio che occupano in memoria, la classe di appartenenza (*tipo della variabile*) e gli attributi.

Per la classe di appartenenza, in Matlab noi useremo principalmente variabili `double`, `logical` e `function_handle`. Sicuramente ci sono molti altri tipi, ma per ora non ci interessa.

Con il comando `clear` vengono cancellate tutte le variabili in memoria, mentre con il comando `clear x` viene cancellata la variabile `x` dalla memoria.

In Matlab ci sono alcune variabili predefinite, ad esempio il **pigreco** (nella variabile `pi`) oppure l'**unità immaginaria** (nelle variabili `i` e `j`). Non è presente di default il numero di Nepero, ma lo possiamo calcolare facilmente con l'espressione `exp(1)`.

Queste variabili, nonostante siano predefinite, possono essere sovrascritte con dei nuovi valori. Per recuperare i valori originali di queste variabili basta utilizzare il comando `clear` visto in precedenza.

Abbiamo parlato di unità immaginaria, introduciamo i **numeri complessi**. Se la variabile `i` non è stata ridefinita, possiamo definire un numero complesso tramite la sua parte reale `a` e la sua parte immaginaria `b` come il numero `a + i*b`. Se `b` non è una variabile possiamo evitare l'operatore `*` tra la `i` e la `b`. I numeri complessi hanno come attributo la dicitura `complex`.

Sui numeri complessi abbiamo una serie di funzioni utili come `real`, `imag`, `conj` e `abs`. Ce ne sono molte altre, ma queste sono quelle più utili e utilizzate.

Per una lista di tutte le funzioni predefinite disponibili in Matlab si può usare il comando `help elfun`. Il comando `help` possiamo usarlo per avere informazioni anche su una specifica funzione.

Come ultimo tipo vediamo il `double`. I numeri in virgola mobile hanno una precisione enorme: dentro `realmax` e `realmin` abbiamo rispettivamente il valore massimo e il valore minimo **positivo**. Se andiamo oltre `realmax` viene restituito il valore `Inf`, che però è anche il risultato di una divisione per 0, quindi bisogna stare attenti a questo valore.

Di default, Matlab mostra un numero in virgola mobile con 5 cifre significative. Il numero di queste ultime può essere cambiato con:

- `format short`, per passare a 5 cifre significative;
- `format long`, per passare a 15 cifre significative;
- `format short e`, per passare a 5 cifre significative in notazione esponenziale;
- `format long e`, per passare a 15 cifre significative in notazione esponenziale.

La **cancellazione numerica** è la perdita di cifre significative quando si sottraggono numeri che sono quasi uguali. Questo è dato dal numero di cifre significative che vengono utilizzate.

Purtroppo, per questo fenomeno della cancellazione numerica, in Matlab non vale in generale la proprietà associativa della somma e la proprietà distributiva del prodotto rispetto alla somma.

In Matlab possiamo scrivere delle **funzioni** (*ma dai*). Essere vanno messe in un file a parte, che contiene solo quella funzione. Il nome della funzione e del file devono coincidere. La firma di una funzione Matlab è **`function [outputs] = name(inputs)`**.

Per aiutare la scrittura di funzioni più complesse, abbiamo anche i costrutti **`if`**, **`while`** e **`for`**.

Vediamo infine le **anonymous function**. Se una funzione deve essere valutata più volte in una serie di valori o magari è molto lunga e complessa è comodo definire la funzione una volta e poi valutarla tutte le volte che serve senza riscriverla. Possiamo fare questo con la sintassi **`nomefunzione = @(parametri) funzione`**. Quando voglio valutare una funzione la chiamo con i parametri indicati **in ordine**.

1.2. Grafici

In Matlab possiamo anche disegnare grafici (*matplotlib suca*). Il comando principale è **`plot(x,y)`**, dove **`x`** e **`y`** sono due vettori di uguale dimensione. Il comando crea una nuova finestra se non ce ne sono altre aperte, altrimenti usa l'ultima finestra grafica utilizzata e sovrascrive il grafico già presente. Per mantenere i grafici precedenti possiamo:

- usare il comando **`hold on`**;
- passare al comando **`plot`** una sequenza di coppie di vettori da graficare in sequenza.

Per aprire più finestre grafiche usiamo il comando **`figure(N)`**.

Al comando **`plot`** è possibile passare una serie di specifiche per definire:

- il **colore** della linea;
- il **simbolo** che viene disegnato come punto;
- come vengono **collegati** i vari punti del grafico.

Con il comando **`doc LineSpec`** vengono mostrate le opzioni per lo stile della linea.

Per rendere carino un grafico possiamo:

- modificare i bound del grafico con **`axis([minx maxx miny maxy])`**;
- inserire un titolo con **`title(titolo)`**;
- inserire una label per l'asse x con **`xlabel(label)`**;
- inserire una label per l'asse y con **`ylabel(label)`**;
- inserire una legenda con **`legend(legenda)`**. Se in una finestra grafica sono stati inseriti più grafici, la legenda si collegherà ai grafici nell'ordine di inserimento;
- inserire una griglia con **`grid on`**.

Infine, se vogliamo avere più grafici nella stessa finestra grafica ma uno accanto all'altro, usiamo il comando **`subplot(nrighe,ncolonne,area)`**. Questo comando crea una griglia $nr \times nc$ formata da aree numerate per riga a partire dal numero 1. Per selezionare un'area della griglia indichiamo il numero dell'area nel comando visto.

2. Vettori e matrici

2.1. Vettori

Un **vettore riga** può essere definito con la sintassi `[v1,...,vn]` oppure con la stessa ma senza virgole. Un **vettore colonna** segue la stessa sintassi ma le componenti sono separate da `;`.

Possiamo **generare** un vettore che contiene elementi con un certo *pattern* indicando il valore iniziale, il valore finale e lo step da eseguire tra un elemento e l'altro per generarli tutti. La sintassi per fare ciò è `inizio:step:fine`. Se lo step è uguale a 1 può non essere indicato. Può succedere che il valore finale indicato non sia presente come ultimo elemento del vettore: questo dipende dallo step scelto per generare il vettore.

Possiamo avere lo stesso comportamento con il comando `linspace(inizio,fine,n_elementi)`. Questo genera lo stesso vettore che otterremo con l'istruzione `inizio:step:fine`, con

$$\text{step} = \frac{\text{fine} - \text{inizio}}{\text{n\_elementi} - 1}.$$

Per generare vettori riga (*colonna*) di n elementi di soli **zeri** o soli **uni** si usano i comandi `zeros(1,n)` (`zeros(n,1)`) e `ones(1,n)` (`ones(n,1)`). Questi comandi prendono il numero di righe e il numero di colonne del vettore da generare.

Possiamo conoscere la **lunghezza** di un vettore con il comando `length(vettore)`. Con il comando `size(x)` invece ci viene restituita la **dimensione** della variabile `x`, ovvero il numero di righe e colonne della variabile. Questo comportamento vale per tutte le variabili, visto che ogni variabile in Matlab è una matrice, e quindi anche i vettori che stiamo considerando. Il comando `length` ritorna il massimo tra i due valori contenuti nel risultato del comando `size`.

Per **accedere** alle componenti del vettore usiamo la sintassi `vettore(indice)`.

ATTENZIONE: gli indici in Matlab partono da 1, non da 0 (*Matlab mi piaci un po' meno adesso*).

Con la parola chiave `end` possiamo accedere facilmente all'ultima componente del vettore. Possiamo accedere anche a più componenti contemporaneamente usando un vettore di indici.

Una volta che accediamo alle componenti di un vettore possiamo modificarle o cancellarle. Per quest'ultima operazione assegniamo il vettore vuoto `[]` alla componente selezionata.

Possiamo **trasporre** un vettore usiamo la sintassi `vettore'`. Questo modifica il vettore da riga a colonna e viceversa.

Per trasformare un vettore riga in una colonna usiamo la sintassi `vettore(:)`. Questo modo, se applicato ad un vettore colonna, restituisce una copia del vettore stesso.

Operazioni banali tra vettori:

- somma e sottrazione tra vettori;
- somma, sottrazione, moltiplicazione e divisione per uno scalare.

In generale la moltiplicazione, la divisione e la potenza tra vettori non è componente per componente, ma possiamo richiedere questa proprietà usando la **notazione puntata** prima dell'operazione da eseguire. Ad esempio, per eseguire l'elevamento a potenza scriviamo `v1.^v2`. Con la stessa notazione possiamo applicare tutte le funzioni matematiche presenti nella suite di Matlab.

È **assolutamente importante** che i vettori siano della stessa dimensione, tranne quando uno dei due operandi è una costante.

Per **concatenare** una serie di vettori creiamo un nuovo vettore composto dai vettori singoli contenuti tra `[]`. La sintassi è `[v1,...,vn]`, anche senza le virgole. Questa concatenazione *«allunga»* il vettore in orizzontale, e quindi i vettori devono avere tutti lo stesso numero di righe. Se invece vogliamo concatenare *«allungando»* in verticale, dividiamo i vettori da concatenare con il `;`. In questo caso, i vettori devono avere lo stesso numero di colonne.

Altre funzioni importanti sui vettori sono:

- `sum(vettore)`: somma delle componenti;
- `prod(vettore)`: prodotto delle componenti;
- `max(vettore)`: massima componente;
- `min(vettore)`: minima componente;
- `sort(vettore)`: componenti ordinate in modo crescente;
- `diff(vettore)`: vettore contenente le differenze tra due componenti successive, ovvero

$$[v(2)-v(1), v(3)-v(2), ..., v(\text{end})-v(\text{end}-1)].$$

Le ultime tre funzioni che abbiamo sui vettori sono:

- **prodotto scalare**, calcolabile con `dot(u,v)` e definito come somma dei prodotti delle componenti con lo stesso indice;
- **norma euclidea** (*norma 2*), calcolabile con `norm(v,2)` (*anche senza il 2*) e definita da

$$\|v\| = \sqrt{\sum_{i=1}^{n} v_i^2};$$

- **norma infinito** (*norma massima (????)*), calcolabile con `norm(v,inf)` e definita da

$$\|v\|_\infty = \max|v_i|.$$

2.2. Matrici

Le **matrici** possono essere definite come in Typst, quindi indicando una serie di vettori riga separati da `;`, oppure generate con le funzioni `ones` e `zeros` definite prima, indicando numero di righe e numero di colonne. Abbiamo le solite **operazioni** banali, quindi somma e differenza tra matrici, ma solo della stessa dimensione.

Possiamo **estrarre** un elemento dalla matrice indicando indice di riga e indice di colonna tra tonde, ma anche sotto-matrici, indicando un range sulle righe e un range sulle colonne. Se vogliamo selezionare tutta una riga o colonna indichiamo `:` nel range di quella parte. Come nei vettori, abbiamo il vettore vuoto `[]` per cancellare righe e colonne.

Possiamo **trasporre** una matrice come nei vettori, *assurdo*.

Possiamo calcolare il **prodotto** tra matrici con il classico `*`, ma anche il prodotto matrice-vettore o vettore-matrice. Abbiamo anche quello vettore-vettore ma uno deve essere riga e uno colonna o viceversa. **ATTENZIONE**: la moltiplicazione richiede numero di colonne del primo operando uguale al numero di righe del secondo operando.

Come nei vettori, esistono operazioni elemento per elemento con la **notazione puntata**.

In Matlab possiamo generare alcune matrici particolari:

- **identità** di dimensione $n \times n$ con `eye(n)`;
- **hilbert** di dimensione $n \times n$ con `hilb(n)` e contenente, nella cella $H(i,j)$, il valore $(i+j-1)^{-1}$;
- **randomica** di dimensione $n \times n$, definita con `rand(n)` e contenente valori casuali.

Possiamo **concatenare** due matrici come nei vettori, ma dobbiamo stare attenti alle dimensioni.

Sulle matrici abbiamo le stesse funzioni dei vettori, ma si comportando diversamente:

- **`sum(M)`**: vettore riga che contiene, nella posizione i, la somma degli elementi della colonna i;
- **`prod(M)`**: vettore riga che contiene, nella posizione i, il prodotto degli elementi della colonna i;
- **`max(M)`**: vettore riga che contiene, nella posizione i, il massimo degli elementi della colonna i;
- **`min(M)`**: vettore riga che contiene, nella posizione i, il minimo degli elementi della colonna i;
- **`sort(M)`**: matrice formata dalle colonne della matrice precedente ma ordinate in modo crescente.

Per calcolare il **determinante** di una matrice quadrata abbiamo la funzione **`det(M)`**. Collegato al concetto di determinante abbiamo il **rango**, calcolabile con la funzione **`rank(M)`**. Ricordiamo che il rango è la dimensione della più grande sotto-matrice quadrata di A avente determinante non nullo. Se $A \in \mathbb{R}^{m\times n}$ allora

$$\mathrm{rk}(A) \leq \min\{m, n\}.$$

Per calcolare l'**inversa** di una matrice quadrata abbiamo la funzione **`inv(M)`**. Una volta calcolata la matrice inversa, è buona cosa controllare se il prodotto tra la matrice e la sua inversa (*e viceversa*) risulta essere la matrice identità.

Abbiamo anche qua nelle matrici due **norme**:

- **norma 1**, calcolabile con **`norm(A,1)`** e definita da

$$\|A\|_1 = \max_{j=1,\dots,n} \sum_{i=1}^{n} |a_{ij}|.$$

  Si può calcolare anche con **`max(sum(abs(A)))`**;
- **norma infinito** (*norma massima (????)*), calcolabile con **`norm(v,inf)`** e definita da

$$\|A\|_\infty = \max_{i=1,\dots,n} \sum_{j=1}^{n} |a_{ij}|.$$

  Si può calcolare anche con **`max(sum(abs(A')))`**.

Vediamo infine tre funzioni che manipolano matrici.

Per generare **matrici triangolari superiori** e **inferiori** abbiamo le funzioni **`triu(M,n)`** e **`tril(M,n)`**. Possiamo indicare un parametro **`n`** che permette di spostare l'inizio della matrice da generare sulle sopra-diagonali (**`n`** *positivo*) o sulle sotto-diagonali (**`n`** negativo).

Per ultima, la funzione **`diag(?,t)`** ha due comportamenti diversi:

- se applicata ad un vettore v di lunghezza n, genera una matrice $n \times n$ con tutti gli elementi del vettore sulla diagonale e il resto 0. Indicando un numero come parametro aggiuntivo, spostiamo la diagonale sulle sopra-diagonali (**`t`** positivo) o sulle sotto-diagonali (**`t`** negativo). Se viene indicato questo numero come parametro, la dimensione della matrice quadrata è $K = |t| + \mathrm{length}(v)$;
- se applicata invece ad una matrice M darà l'effetto opposto, ovvero genera un vettore formato dai coefficienti sulla diagonale principale. Come prima, indicando un numero come parametro aggiuntivo selezioneremo le sopra-diagonali o le sotto-diagonali. Se viene indicato questo numero come parametro, la dimensione del vettore è $K = \mathrm{length}(M) - |t|$.

Queste ultime tre funzioni, se ben concatenate, permettono di creare le **matrici a banda**.

3. Sistemi lineari

3.1. Metodi diretti

In Matlab possiamo calcolare tramite **eliminazione gaussiana** le soluzioni di un sistema $Ax = b$ con la sintassi `A\b`. Questo a volte usa la sostituzione in avanti, a volte la sostituzione all'indietro.

Una seconda opzione è la **fattorizzazione LU**, calcolabile con `[L,U,P] = lu(A)`. La funzione torna anche una matrice P: questo perché non sempre è possibile fattorizzare A nelle matrici L e U, quindi utilizziamo una matrice P, detta **matrice di permutazione**, da applicare all'intero sistema lineare per permettere poi la fattorizzazione richiesta. Il sistema lineare diventa quindi $PAx = Pb$.

Se la matrice A è sparsa vorremmo sapere se `lu` abbia nelle stesse posizioni degli zeri. Questo non è possibile in generale, ma sarebbe molto comodo per le computazioni. Questo fenomeno è detto **fill-in**, ovvero si riempiono le celle delle matrici L e U con valori diversi da zero ma nella matrice A abbiamo degli zeri. Le eccezioni a questa regola sono le matrici a banda.

In Matlab, con il comando `spy(A)` possiamo graficare le celle della matrice con valori non nulli.

L'ultima fattorizzazione che abbiamo visto è quella di **Cholesky**, calcolabile con `R = chol(A)`.

3.2. Metodi iterativi

3.2.1. Autovalori e autovettori

Gli **autovalori** di una matrice A si calcolano con la funzione `[V,D] = eig(A)`. Questa ritorna anche gli **autovettori** relativi di ogni autovalore. Con gli autovalori possiamo calcolare il **raggio spettrale** e la **norma 2** della matrice A con il comando `norm(A)`.

3.2.2. Metodo di Jacobi e Gauss-Seidel

Per le matrici di iterazione, le tolleranze e altro vedere la parte di teoria.

Condizioni sufficienti per la convergenza:

- A diagonalmente dominante in senso forte $\implies$ J e GS convergono;
- A simmetrica definita positiva $\implies$ GS converge.

Condizione necessaria e sufficiente per la convergenza:

- I metodi iterativi convergono $\iff \rho(B) < 1$.
- A tridiagonale, J converge $\iff$ GS converge.

Per le matrici tridiagonali, GS è il doppio più veloce di J.

4. Interpolazione polinomiale

4.1. Lavorare con i polinomi

In Matlab non abbiamo degli oggetti built-in per contenere dei **polinomi**. Possiamo pensare di rappresentarli come vettori contenenti i coefficienti del polinomio, a partire da quello del grado maggiore (**coefficiente direttivo**) fino a quello del grado minore (**termine noto**).

Grazie a questa rappresentazione, in Matlab abbiamo le funzioni:

- **`polyval(p,points)`**: valuta il polinomio p nei punti dati;
- **`polysum(p,q)`**: somma il polinomio p al polinomio q. Definito da noi in un esercizio, dobbiamo stare attenti a sommare polinomi dello stesso grado o ad aggiungere un pad di zeri;
- **`conv(p,q)`**: moltiplica il polinomio p con il polinomio q;
- **`deconv(p,q)`**: divide il polinomio p per q, ritornando quoziente e resto;
- **`polyder(p)`**: deriva il polinomio p;
- **`polyint(p)`**: integra il polinomio p dando la primitiva con $C = 0$;
- **`roots(p)`**: calcola le radici di p;
- **`poly(r)`**: costruisce un polinomio p a partire dalle sue radici r.

Come **operazioni** banali abbiamo prodotto e divisione per uno scalare, moltiplicando o dividendo il vettore che rappresenta il polinomio per lo scalare dato.

4.2. Polinomio interpolatore

Il **polinomio interpolatore** si calcola con la funzione **`polyfit(x,y,n)`**. Questa richiede due vettori, che contengono i nodi di interpolazione, e il grado del polinomio interpolatore.

Il grado del polinomio risultante deve essere **NECESSARIAMENTE** uno in meno della grandezza del sample di nodi di interpolazione. Abbiamo due modi per garantire ciò:

- se ci viene dato un sample di punti, impostiamo il grado del polinomio alla lunghezza del sample meno uno;
- se ci viene dato il grado n del polinomio, generiamo un sample di $n + 1$ punti.

Quando le misurazioni sperimentali si pongono con una relazione lineare ma non in linea perfetta è meglio usare la **retta di regressione**. Essa è meno dispendiosa da calcolare rispetto rispetto al polinomio interpolatore. Essa si calcola con il comando **`polyfit(x,y,1)`**. Se i punti da approssimare sono due, la retta di regressione e il polinomio interpolatore coincidono.

Infine, possiamo partizionare un intervallo in m sotto-intervalli e interpolare ogni singolo sotto-intervallo. Il caso più semplice è la **spline lineare interpolante**, ovvero quella funzione che interpola ogni sotto-intervallo con delle rette. In Matlab, possiamo trovare una spline lineare con la funzione **`griddedInterpolant(x,y,'linear')`**.

Siano H la massima ampiezza dei sotto-intervalli e $E(H)$ l'errore di approssimazione tra la spline e la funzione in quel sotto-intervallo. Si può dimostrare che

$$E(H) \leq CH^2 \mid C \in \mathbb{R}.$$

5. Integrazione numerica

Per approssimare una **funzione integrale** in Matlab abbiamo la funzione **`integral(f,a,b)`**. L'area che viene restituita ha:

- un errore assoluto minore di **`1e-10`**;
- un errore relativo minore di **`1e-6`**;
- entrambe le precedenti

rispetto al valore esatto dell'integrale.

5.1. Formule di quadratura

Abbiamo tre **formule di quadratura** per approssimare un integrale:

- **punto medio** (*semplice e composito*);
- **trapezi** (*semplice e composito*);
- **Cavalieri-Simpson** (*semplice e composito*).

La prima e la terza formula sono state date come esercizio in classe. La formula dei trapezi composta è implementata in Matlab con la funzione **`trapz(x,y)`**.

(????) la formula dei trapezi semplice equivale a integrare la retta interpolante della funzione integranda negli estremi dell'intervallo. La formula dei trapezi composita equivale a integrare la spline lineare della funzione integranda nei punti di quadratura. (????)

Per le formule del punto medio e dei trapezi l'errore commesso è $O(H^2)$, mentre per la formula di Cavalieri-Simpson l'errore commesso è $O(H^4)$.

6. Zeri di funzione

Possiamo trovare gli zeri di una funzione data usando la funzione **`fzero(f,[a,b])`**.

ATTENZIONE: devono valere le ipotesi del **teorema degli zeri**, ovvero:

- la funzione nei due estremi deve essere di segno discorde;
- la funzione è continua.

Se cade la prima ipotesi la funzione solleva un errore. Inoltre, i punti in cui la funzione tocca l'asse delle x non sono considerati zeri perché non hanno segno discorde.

Se cade la seconda ipotesi, potrebbero essere riportati punti di discontinuità.

6.1. Metodo di bisezione

Il **metodo di bisezione** è un ottimo metodo per trovare gli zeri di una funzione, avendo attenzione a chiamare la funzione su intervalli che soddisfino le ipotesi del teorema degli zeri. Questo metodo converge sempre e l'errore ogni volta è minore della metà dell'intervallo corrente.

6.2. Metodo di Newton

Il **metodo di Newton** è un altro ottimo metodo, ma necessita della derivata della funzione data e soprattutto:

- rischia una divisione per zero;
- rischia una non convergenza.

7. Equazioni differenziali ordinarie

Con la funzione **`[T,U] = ode45(f,[a,b],y0)`** possiamo approssimare un problema di Cauchy nell'intervallo $[a, b]$ con condizione iniziale y_0 campionandolo in una serie di tempi T. Questi tempi sono ritornati assieme ai valori risultati dal campionamento U.

Conosciamo anche una serie di metodi utili per l'approssimazione di questi problemi di Cauchy.

Metodo	Ordine di convergenza	Caratteristiche
Eulero esplicito	1	Convergente, assolutamente instabile per step h non sufficientemente piccoli
Eulero implicito	1	Può essere non convergente, assolutamente stabile
Heun	2	Convergente, assolutamente instabile
Crank-Nicolson	2	Può essere non convergente

8. Sistemi lineari di equazioni differenziali ordinarie

Per risolvere **sistemi di equazioni differenziali ordinarie** abbiamo, come nelle equazioni differenziali classiche, la funzione **`ode45(F,[a,b],Y0)`**. In questo caso, la funzione F deve essere un **vettore colonna** di funzioni note, **vettoriali** in y, e Y_0 contiene le condizioni iniziali di ogni equazione.

9. Codici utilizzati

Come per la parte di teoria, di seguito vediamo un riassunto degli script file che sono stati utilizzati durante le lezioni di laboratorio, così da averli a disposizione subito senza doverseli ricordare.

9.1. Sistemi lineari

```
function [x,nit] = jacobi(A,b,x0,toll,nitmax)

D = diag(diag(A));
E = -tril(A,-1);
F = -triu(A,1);

it = 0;
err = toll + 1;
new_x = x0;
norm_b = norm(b,inf);

while it < nitmax && err > toll
    old_x = new_x;
    new_x = D\(b + (E + F)*old_x);
    it = it + 1;
    fe = norm(new_x-old_x, inf) / norm(new_x,inf);
    se = (norm(b-A*old_x, inf) / norm_b);
    err = max(fe, se);
end

x = new_x;
nit = it;

if it == nitmax && err > toll
    disp("Jacobi non converge.");
end

return

end
```

```
function [x,nit] = gauss_seidel(A,b,x0,toll,nitmax)

D = diag(diag(A));
E = -tril(A,-1);
F = -triu(A,1);

it = 0;
err = toll + 1;
new_x = x0;
norm_b = norm(b,inf);

while it < nitmax && err > toll
    old_x = new_x;
    new_x = (D-E)\(b + F*old_x);
    it = it + 1;
    fe = norm(new_x-old_x, inf) / norm(new_x,inf);
```

```
    se = (norm(b-A*old_x, inf) / norm_b);
    err = max(fe, se);
end

x = new_x;
nit = it;

if it == nitmax && err > toll
    disp("Gauss-Seidel non converge.");
end

return

end
```

9.2. Interpolazione polinomiale

```
function [pq] = polysum(p,q)

lenp = length(p);
lenq = length(q);

if lenp < lenq
    pad = lenq - lenp;
    p = [zeros(1,pad) p];
end

if lenq < lenp
    pad = lenp - lenq;
    q = [zeros(1,pad) q];
end

pq = p + q;
return

end
```

9.3. Integrazione numerica

```
function [area] = punto_medio(a,b,m,f)

x = linspace(a,b,m+1);
H = (b-a)/m;
area = H * sum(f(x(1:end-1) + H/2));
return

end
```

```
function [area] = cavalieri_simpson(a,b,m,f)
```

```
x = linspace(a,b,m+1);
H = (b-a)/m;

fs = 2*sum(f(x(2:end-1)));
ss = 4*sum(f(x(1:end-1) + H/2));
area = H * (f(a) + fs + ss + f(b)) / 6;

return

end
```

9.4. Zeri di funzione

```
function [x,nit] = bisezione(f, a, b, toll)

if f(a)*f(b) > 0
    disp("Intervallo non accettabile")
    x = [];
    nit = 0;
    return
end

k = 1;
ak = a;
bk = b;
xk = (ak+bk)/2;
err = (bk-ak)/2;

while err > toll
    if f(xk) == 0
        break
    end

    if f(ak)*f(xk) < 0
        bk = xk;
    else
        ak = xk;
    end

    xk = (ak+bk)/2;
    err = err/2;
    k = k + 1;
end

x = xk;
nit = k;

return

end
```

```
function [x,nit] = newton(f, df, x0, toll, nitmax)
```

```
k = 0;
err = toll + 1;

while k < nitmax && err > toll
    xk = x0 - (f(x0))/(df(x0));

    err = abs(xk - x0);
    k = k + 1;

    x0 = xk;
end

nit = k;
if k == nitmax && err > toll
    disp("Il metodo di Newton non converge");
    x = 0;
else
    x = xk;
end

return

end
```

9.5. Equazioni differenziali ordinarie

```
function [T,Y] = eulero(fun, T, y0)

u0 = y0;
h = T(2) - T(1);

Y = y0;
for n = 2:length(T)
    tn = T(n-1);
    un = u0 + h * fun(tn,u0);

    Y = [Y un];
    u0 = un;
end

T = T';
Y = Y';

return

end
```

```
function [T,U] = eulero_implicito(f,T,y0,dfy,toll,nitmax)

u0 = y0;
h = T(2) - T(1);
```

```
U = u0;
for i = 2:length(T)
    % Newton
    err = toll + 1;
    nit = 0;
    y = U(i-1);
    un = U(i-1);
    while nit < nitmax && err > toll
        yk = y - (y - un - h*f(T(i),y))/(1 - h*dfy(T(i),y));
        err = abs(yk - y);
        y = yk;
        nit = nit + 1;
    end

    if nit == nitmax && err > toll
        disp("Il metodo di Netwon adattato non converge")
        T = T';
        U = [];
        return
    end

    U = [U yk];
end

T = T';
U = U';
return

end
```

```
function [T,U] = crank_nicolson(f,T,y0,dfy,toll,nitmax)

if nargin == 4
    toll = 1e-6;
    nitmax = 20;
elseif nargin == 5
    nitmax = 20;
end

h = T(2) - T(1);
U = y0;

for n = 2:length(T)
    % Newton
    err = toll + 1;
    nit = 0;
    un = U(n-1);
    x0 = U(n-1);
    t0 = T(n-1);
    tn = T(n);

    while nit < nitmax && err > toll
        n1 = un + h/2 * f(t0,un);
        n2 = h/2 * f(tn,x0);
```

```
        num = x0 - n1 - n2;
        den = 1 - h/2 * dfy(tn,x0);

        xk = x0 - num/den;

        err = abs(xk - x0);
        x0 = xk;
        nit = nit + 1;
    end

    if nit == nitmax && err > toll
        disp("Il metodo di Netwon esotico non converge.")
        T = T';
        U = [];
        return
    end

    U = [U xk];
end

T = T';
U = U';
return

end
```

```
function [T,U] = heun(f,T,y0)

h = T(2) - T(1);
u0 = y0;
U = y0;

for n = 2:length(T)
    t0 = T(n-1);
    tn = T(n);

    un = u0 + (h/2) * (f(t0,u0) + f(tn,u0 + h*f(t0,u0)));

    U = [U un];
    u0 = un;
end

T = T';
U = U';
return

end
```