Calcolo numerico

Indice

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 $A = \big(a_{ij}\big) \in \mathbb{R}^{m \times n}$ una matrice e $\beta \in \mathbb{R}$ uno scalare. Chiamiamo **prodotto matrice-scalare** la matrice $C = \big(c_{ij}\big) \in \mathbb{R}^{m \times n}$ 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 $A = \big(a_{ij}\big) \in \mathbb{R}^{m \times n}$ 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 $A = \big(a_{ij}\big) \in \mathbb{R}^{m \times n}$ e $B = \big(b_{ij}\big) \in \mathbb{R}^{n \times k}$ due matrici. Chiamiamo **prodotto matrice-matrice** di A per B la matrice $C = \big(c_{ij}\big) \in \mathbb{R}^{m \times k}$ 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 $I = \big(i_{kj}\big)$ 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

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

- **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 di invertibilità: sia A una matrice quadrata di ordine n, allora A è invertibile se e solo se $\det(A) \neq 0$.

Teorema: 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 $k \le \min(m, n)$. 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 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 di 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

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

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: 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.

Proposizione: 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

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

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

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

Proposizione: 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 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:

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

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: 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: 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 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: 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: 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 $P_N(x) - Q_N(x) = 0 \forall x \in \mathbb{R}$ 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**.

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.

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

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

4.1.1. 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: siano $x_0, ..., x_N$ un insieme di $N+1$ nodi distinti, $x \neq x_i \quad \forall i = 0, ..., N$ e $f \in C^{N+1}(I_x)$, 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: 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

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

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

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

con

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

5. Lezione 13

5.1. Integrazione numerica

Vogliamo calcolare l'integrale definito

$$I(f) = \int_a^b f(x)dx$$

data la funzione $f : [a, b] \longrightarrow \mathbb{R}$. In generale non possiamo calcolare $I(f)$ per via analitica, ma possiamo solo approssimarla numericamente tramite formule di quadratura.

Si chiama formula di quadratura una formula del tipo

$$I^{\tilde{f}} = \sum_{i=1}^{n} a_i f(x_i)$$

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 nodi di quadratura) moltiplicati per opportuni coefficienti (α_i pesi di quadratura).

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

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

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

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

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

Il nodo di quadratura è $\frac{a+b}{2}$ mentre il peso di quadratura è $a_1 = b - a$. 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 $f \in C^2([a, b])$.

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

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

Abbiamo quindi due nodi di quadratura $x_1 = a, x_2 = b$ e due pesi di quadratura $\alpha_1 = \alpha_2 = \frac{b-a}{2}$. L'errore con questa formula di quadratura è

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

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

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

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

Abbiamo quindi tre nodi di quadratura $x_1 = a, x_2 = \frac{a+b}{2}, x_3 = b$ e due pesi di quadratura $\alpha_1 = \alpha_3 = \frac{b-a}{6}, \alpha_2 = \frac{2(b-a)}{3}$. L'errore con questa formula di quadratura è

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

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

Si chiama grado di precisione di una formula di quadratura il massimo intero $r \geq 0$ tale che $I^{\sim}(P) = I(P) \quad \forall P \in \mathbb{P}_r$.

Proposizione: una formula di quadratura ha grado di precisione r se e solo se

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

Per il punto medio, proviamo $k = 0$ quindi $f(x) = x^0 = 1$ e quindi

$$I(f) = I(1) = \int_a^b 1dx = [x]_a^b = b - a$$
$$I_{PM}^{\sim}(f) = I_{PM}^{\sim}(1) = (b-a)\cdot 1 = b - a$$

quindi

$$I(1) = I_{PM}^{\sim}(1).$$

Proviamo $k = 1$ quindi $f(x) = x$ e quindi

$$I(f) = I(x) = \int_a^b x dx = \left[\frac{x^2}{2}\right]_a^b = \frac{b^2-a^2}{2}$$
$$I_{PM}^{\sim}(f) = I_{PM}^{\sim}(x) = (b-a)\frac{a+b}{2} = \frac{b^2-a^2}{2}$$

quindi

$$I(x) = I_{PM}^{\sim}(x).$$

Se provassimo con $k = 2$ avremmo due risultati diversi, quindi PM ha grado di precisione 1.

Il trapezio ha grado di precisione 1, Cavalieri-Simpson ha grado di precisione 3.

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, scriverlo come una somma di integrali definiti su ciascun intervallo della suddivisione;
- approssimare tali integrali definiti mediante formule di quadratura semplici.

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

La formula el punto medio composita approssima con

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

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

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

La formula del trapezio composita approssima con

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

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

$$I(f) - I_{T}^{\tilde{C}} = -\frac{H^2}{12}(f'(b) - f'(a)).$$

La formula di Cavalieri-Simpson composita approssima con

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

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

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

6. Lezione 14

7. Lezione 15

7.1. Zeri di funzione

Data una funzione $f : [a, b] \longrightarrow \mathbb{R}$ continua e tale che $f(a)f(b) < 0$ trovare $\alpha \in (a, b)$ tale che $f(\alpha) = 0$.

In generale α non riusciamo a calcolarlo per via analitica (a.e. eq non lineari), ma possiamo solo approssimarlo numericamente.

Teorema: sia $f : [a, b] \longrightarrow \mathbb{R}$ continua in $[a, b]$ e $f(a)f(b) < 0$ 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.

Il **metodo di bisezione** parte con $a_0 = a$ e $b_0 = b$. Calcolo $x_1 = \frac{a_0+b_0}{2}$ con $|x_1 - \alpha| < \frac{b-a}{2}$. Se consideriamo la parte di intervallo e vale ancora il prodotto negativo allora restringo a questa parte la ricerca e ricomincio, altrimenti restringo sulla seconda parte di segmento. Al passo n-esimo ho

$$|x_n - \alpha| < \frac{b-a}{2^n}.$$

Vantaggi:

- robusto;
- converge sempre.

Svantaggi:

- convergenza lenta;
- buona approssimazione la si raggiunge lentamente.

Il **metodo di newton** parte da un x_0, calcola la retta tangente in x_0 e cerca di annullare questa funzione. Il nuovo punto sarà il punto di partenza per la nuova iterazione.

In generale, sia $f : [a, b] \longrightarrow \mathbb{R}$ derivabile tale che $f(a)f(b) < 0$ e $f'(x) \neq \quad \forall x \in [a, b]$. Sia $x_0 \in [a, b]$, allora per $k = 0, 1, 2, \ldots$ poniamo $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$.

Vantaggi:

- la convergenza è quadratica (veloce), infatti

$$\left|x_{k+1} - \alpha\right| \approx |x_k - \alpha|^2.$$

Svantaggi:

- la convergenza dipende dalla scelta di x_0, se non è sufficientemente vicino ad α il metodo può non convergere.

Teorema: supponiamo

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

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

8. Lezione 16

9. Lezione 17

9.1. 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 rispetto a t, cioè

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

Con queste ipotesi Cauchy ammette una e una sola soluzione.

Abbiamo $N + 1$ 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 $y_j \coloneqq y(t_j) \quad j = 0, ..., N$.

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

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 $j < n$. Altrimenti è detto multistep.

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

Eulero esplicito: posto $u_0 = y_0 \quad \forall n \geq 0$ faccio

$$u_{n+1} = u_n + hf(t_n, u_n).$$

Eulero implicito: posto $u_0 = y_0 \quad \forall n \geq 0$ faccio

$$u_{n+1} = u_n + ht(t_{n+1}, u_{n+1})$$

In questo caso, ad ogni passo dobbiamo risolvere un'equazione non lineare, ad esempio tramite Newton.

Metodo di Crank-Nicolson: posto $u_0 = y_0 \quad \forall n \geq 0$ faccio

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

Metodo di Heun: posto $u_0 = y_0 \quad \forall n \geq 0$ faccio

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

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

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

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

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

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

Consideriamo ora il problema modello

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

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

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

$$u_n \to 0 \quad t_n \to \infty.$$

Come sono i nostri metodi:

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

10. Lezione 18

11. Lezione 19

11.1. Metodi numerici per sistemi di equazioni differenziali ordinarie

Consideriamo il problema di Cauchy

$$\begin{cases} \frac{\mathrm{d}y_1(t)}{\mathrm{d}t} = f_1(t, y_1(t), y_2(t)) \quad t \in (t_0, T) \\ \frac{\mathrm{d}y_2(t)}{\mathrm{d}t} = f_2(t, y_1(t), y_2(t)) \quad t \in (t_0, T) \\ y_1(t_0) = y_1^0 \\ y_2(t_0) = y_2^0 \end{cases}.$$

Sistemi di equazioni differenziali ordinarie.

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

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

dove

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

Abbiamo **due metodi di Eulero**:

- esplicito: posto $u_1^0 = y_1^0$ e $u_2^0 = y_2^0$ allora $\forall n \geq 0$ fare

$$\begin{cases} u_1^{n+1} = u_1^n + hf_1(t_n u_1^n, u_2^n) \\ u_2^{n+1} = u_2^n + hf_2(t_n, u_1^n, u_2^n) \end{cases};$$

- implicito: posto $u_1^0 = y_1^0$ e $u_2^0 = y_2^0$ allora $\forall n \geq 0$ fare

$$\begin{cases} u_1^{n+1} = u_1^n + hf_1\big(t_n u_1^{n+1}, u_2^{n+1}\big) \\ u_2^{n+1} = u_2^n + hf_2\big(t_n, u_1^{n+1}, u_2^{n+1}\big) \end{cases}.$$

Nel caso implicito, ad ogni passo temporale per trovare i valori di u_1^{n+1} e u_2^{n+1} bisogna risolvere in generale un sistema algebrico non lineare.

12. Lezione 20

13. Laboratorio 01

13.1. Introduzione

Ogni variabile in Matlab è una matrice, anche gli scalari. 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.

```
a = 6       % Viene mostrato nel prompt
b = 2.5;    % Non viene mostrato nel prompt
```

Se un'espressione viene valutata ma non viene assegnata a nessuna variabile, il risultato viene salvato nella variabile «ans», che conterrà ogni volta l'ultima espressione valutata non salvata.

```
6 + 2;
ans       % Viene mostrato 8 nel prompt
```

Con il comando «who» vengono mostrate le variabili in memoria, mentre con il comando «whos» vengono stampate informazioni aggiuntive, ad esempio tipo e dimensione delle variabili.

```
a = 6;
6 + 2;

who         % Vengono mostrate le variabili "a" e "ans"
whos        % Idem ma con informazioni aggiuntive
```

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

```
a = 6;
b = 10;
6 + 2;

clear a;
who             % Vengono mostrate le variabili "b" e "ans"

clear all;
who             % Non viene mostrato niente
```

Ci sono alcune varibili predefinite:

- pigreco: variabile «pi»;
- unità immaginaria: variabili «i» e «j».

Non è presente il numero di Nepero, ma che possiamo calcolare facilmente con la funzione `exp(1)`.

Le variabili predefinite possono essere sovrascritte. Per tornare ai valori originali si usa il comando «clear» visto in precedenza.

Come in ogni linguaggio di programmazione, ci sono delle operazioni fondamentali come somma, differenza, prodotto, divisione e potenza.

```
a = 3;
b = 4;

a + b, a - b, a * b, a / b, a ^ b
```

Sono presenti anche i numeri complessi. Se la variabile «i» non è stata ridefinita possiamo definire un numero complesso tramite la sua parte reale e la sua parte immaginaria, anche senza l'operatore «*» se la parte immaginaria è un numero.

Sui numeri complessi abbiamo una serie di funzioni utili, come «real» (*restituisce la parte reale*), «imag» (*restituisce la parte immaginaria*), «conj» (*restituisce il coniugato del numero complesso*) e «abs» (*restituisce il modulo del numero complesso*).

```
a = 5;

z1 = 2 + 2i;
z2 = a + a * i;

real(z1)            % viene mostrato 2
imag(z1)            % viene mostrato 2
conj(z1)            % viene mostrato 2 - 2i
abs(z1)             % viene mostrato sqrt(8)
```

Prima abbiamo visto «atan», ma non è l'unica funzione presente in Matlab: per vedere la lista completa usare il comando «help elfun».

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 «Inf», che però è anche il risultato di una divisione per 0.

Il numero di cifre signficiative di un numero può essere cambiato:

- «format short»: 5 cifre significative;
- «format long»: 15 cifre significative;
- «format short e»: 5 cifre significative in notazione esponenziale;
- «format long e»: 15 cifre significative in notazione esponenziale;

Purtroppo, in matlab non vale in generale la proprietà associativa della somma e la proprietà distributiva del prodotto rispetto alla somma.

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.

14. Laboratorio 02

Tutto in Matlab è un vettore, quindi ora vediamo i vettori.

Un **vettore riga** può essere definito come lista tra quadre, con gli elementi separati da spazio o virgole. Un **vettore colonna** è uguale ma le componenti sono separate dal punto e virgola.

```
u = [1 2 3]
v = [1,2,3]
w = [1;2;3]
```

Possiamo generare un vettore indicando il valore di inizio, il punto di fine e lo step da eseguire tra un elemento e l'altro per generarli tutti. Di default lo step è uguale a 1 quando non viene indicato. La sintassi è «[inizio,step,fine]». A volte l'ultimo valore del vettore non coincide con il valore finale perché lo step non lo raggiunge pienamente.

```
u = [1:10]
v = [1:1:10]
w = [1:2:10]
```

Con il comando «linspace» possiamo generare un vettore di una certa grandezza formato dai punti tra un punto iniziale e uno finale equispaziati. La sintassi è «linspace(inizio,fine,numero_elementi».

```
u = linspace(1,10,100)
```

In poche parole, generiamo un intervallo uguale a quello di linspace con l'istruzione «[inizio,step,fine]» con $\text{step} = \frac{\text{fine} - \text{inizio}}{\text{numero\_elementi} - 1}$

Per generare vettori riga (*colonna*) di soli zeri o soli uni si usano i comandi «zeros(1,n)» (*«zeros(n,1)»*) e «ones(1,n)» (*«ones(n,1)»*).

```
z1 = zeros(1,n)
z2 = zeros(n,1)

o1 = ones(1,n)
o2 = ones(n,1)
```

Possiamo conoscere la lunghezza di un vettore con il comando «length(vettore)». Con il comando «size(x)» ci viene restituita la dimensione della variabile «x», ovvero il numero di righe e colonne della variabile «x», visto che ogni variabile è un vettore.

```
u = [1,2,3];
length(u)
size(u)
```

Per accedere alle componenti del vettore usiamo il nome della variabile e indichiamo la posizione tra parentesi tonde. ATTENZIONE: gli indici partono da 1 (*Matlab mi piaci un po' meno adesso*). Con la parola chiave «end» accediamo all'ultima componente del vettore.

```
u = [1,2,3];
u(1)
u(3)
u(end)
```

Possiamo accedere anche a più componenti contemporaneamente usando un vettore di indici. Una volta che accediamo a queste componenti possiamo modificarle o cancellarle. Per cancellare un elemento da un vettore assegnamo il vettore vuoto «[]» alla componente selezionata.

```
u = [1,2,3,4,5];
u([1,4,5])
```

```
u(1:2:5)

u([1,2]) = [5,4]
```

Possiamo **trasporre** un vettore usando il singolo apice dopo il nome del vettore. Per ottenere un vettore colonna da un vettore riga possiamo usare un trick esotico. Se questo trick è usato su un vettore colonna viene restituita una copia del vettore.

```
u = [1,2,3];
u'
u(:)
```

Possiamo anche sommare e sottrarre vettori, sommare, sottrarre, moltiplicare e dividere per uno scalare.

```
u = [1,2,3];
v = [4,5,6];
u + v, u - v, u + 5, u - 5, 5 * u, u  / 5
```

Abbiamo anche operazioni componente per componente, come il prodotto, la divisione e l'elevamento a potenza. L'importante è avere vettori della stessa dimensione, tranne quando uno dei due operandi è una costante.

```
u = [1,2,3];
v = [4,5,6];
u.*v, u./v, u.^v, 2.^v
```

Con la stessa notazione puntata possiamo applicare tutte le funzioni matematiche presenti nella suite.

Per concatenare una serie di vettori creiamo un nuovo vettore composto dai vettori singoli.

```
u = [1,2,3];
v = [4,5,6];
[u v]
```

Altre funzioni importanti sui vettori sono:

- «sum»: calcola la somma delle componenti;
- «prod»: calcola il prodotto delle componenti;
- «max»: calcola la massima componente;
- «min»: calcola la minima componente;
- «sort»: ordina le componenti in modo crescente;
- «diff»: calcola il vettore contenente le differenze tra due componenti successive, ovvero $[v(2) - v(1), v(3) - v(2), ..., v(\text{end}) - v(\text{end} - 1)]$

Ultime tre funzioni sono il **prodotto scalare**, indicato con «dot(u,v)» e definito come somma dei prodotti delle componenti con lo stesso indice, la **norma euclidea** $\|v\| = \sqrt{\sum_{i=1}^{n} v_i^2}$, indicata con «norm(u,2)» o «norm(u)» e la **norma infinito** $\|v\|_\infty = \max|v_i|$, indicata con «norm(u,inf)».

```
u = [1,2,3];
v = [4,5,6];
dot(u,v), norm(u), norm(u,inf)
```

15. Laboratorio 03

Le matrici possono essere definite come in Typst, oppre generate con «ones» e «zeros» definendo numero di righe e numero di colonne. Possiamo fare somma e differenza tra matrici, ma solo della stessa dimensione.

```
A = [1 2 3; 4 5 6]
B = ones(2,3)
C = zeros(2,3)

B + C
```

Possiamo estrarre un elemento dalla matrice indicano 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.

```
A = [1 2 3; 4 5 6; 7 8 9]
A(1,2)
A(1:2,:) = []
```

Possiamo trasporre una matrice con il singolo apice, come nei vettori. Possiamo calcolare anche 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.

```
A = [1 2; 3 4]
B = [1 2 3; 4 5 6]
u = [1 2]

A*B
u*A
B*(u')
u*(u')
```

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

Esistono alcune matrici particolari:

- **identità**: di dimensione $n \times n$, definita con «eye(n)»;
- **hilbert**: di dimensione $n \times n$, definita 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 (*occhio a dove inseriamo la seconda*), ma dobbiamo stare attenti alle dimensioni.

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

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

Abbiamo anche la funzione «det» per calcolare il determinante di *matrici quadrate* e la funzione «rank» per calcolare il rango di una matrice. Il rango è la dimensione della più grande sotto-matrice quadrata avente determinante non nullo. Se $A \in \mathbb{R}^{m \times n}$ allora rank $\leq \min\{m, b\}$.

Per calcolare l'inversa di una matrice quadrata abbiamo la funzione «inv». È buona cosa controllare se il prodotto tra la matrice e la sua inversa (*e viceversa*) è la matrice identità.

Abbiamo anche qua le norme:

- norma 1: calcolabile con «norm(A,1)» e definita da $\|A\|_1 = \max_{j=1,\dots,n} \sum_{i=1}^{n} |a_{i,j}|$; si può calcolare anche con «max(sum(abs(A)))»;
- norma infinito: calcolabile con «norm(A,inf)» e definita da $\|A\|_\infty = \max_{i=1,\dots,n} \sum_{j=1}^{n} |a_{i,j}|$; si può calcolare anche con «max(sum(abs(A')))».

La funzione «diag», se applicata ad un vettore di lunghezza n, genera una matrice $n \times n$ con gli elementi del vettore sulla diagonale. Indicando un numero dopo il vettore si sposta quel vettore sulle sopra/sotto-diagonali, aumentando però la dimensione della matrice di «abs(numero)».

La funzione «diag», se applicata invece ad una matrice, darà l'effetto opposto, quindi ritorna un vettore formato dai coefficienti sulla diagonale principale. Come prima, indicando un numero selezioneremo le sopra/sotto-diagonali.

La funzione «diag» è comoda per creare le **matrici a banda**.

Infine, per generare matrici triangolari superiori e inferiori si usano i comando «triu(M)» e «tril(M)», con anche l'indicazioni di un numero per alzare/abbassare la banda di zeri. ATTENZIONE: con un numero positivo andiamo a selezionare le sopra-diagonali, con un numero negativo le sotto-diagonali.

16. Laboratorio 04

16.1. Script file

Possiamo scrivere script (*ma dai*). Abbiamo i costrutti if e for, ma anche tutti gli operatori relazionali.

16.2. Fattorizzazioni

In Matlab possiamo calcolare tramite eliminazione gaussiana le soluzioni del sistema $Ax = b$ con l'operatore «\» (*a volte si usa sostituzione in avanti, a volte all'indietro*).

Quando possiamo fare la fattorizzazione LU è un bene. Non sempre riusciamo, spesso usiamo una matrice di permutazione P tale che $PA = Pb$, e poi da qua fattorizziamo con LU come abbiamo visto. La fattorizzazione LU avviene con il comando «lu(M)» e ritorna gli elementi «[L,U,P]».

Se la matrice A è sparsa vorremmo sapere se in LU abbiamo nelle stesse posizioni degli zeri, ma questo non è possibile in generale, ma sarebbe stra comodo per le computazioni. Questo fenomeno è detto **fill-in**, ovvero si riempiono le celle in LU con valori diversi da zero ma nella A abbiamo degli zeri. Le eccezioni a questa regola sono le matrici a banda. Con il comando «figure» e «spy» possiamo vedere le celle della matrice con valori non nulli.

Ultima fattorizzazione è quella di Cholesky, che si ottiene con «chol(M)».

17. Laboratorio 05

17.1. Autovalori e autovettori

Data una matrice A quadrata di ordine n, il numero $\lambda \in \mathbb{C}$ si dice **autovalore** di A se esiste un vettore non nullo, detto autovettore, tale che $Av = \lambda v$. Gli autovalori si calcolano con il comando «eig(M)». Questo comando ritorna anche gli autovettori di ogni autovalore, metti in una matrice.

Con gli autovalori possiamo calcolare il raggio spettrale e anche la norma 2 con il comando «norm(M)».

17.2. Funzioni

Possiamo definire delle funzioni che hanno sintassi

```
function [out1, ...] = nomefunzione (in1, ...)
...
return
```

Un file contiene una sola funzione e il nome «nomefunzione» deve essere il nome del file che contiene tale funzione.

17.3. Jacobi e Gauss-Seidel

Per le matrici, le tolleranze e altro vedere la teoria.

Una matrice quadrata A di ordine n si dice diagonale dominante per righe (colonne) se sulla diagonale in abs ho somma maggiore uguale della somma di tutti gli altri sulla riga (colonna) in abs, ovvero

$$|a_{ij}| \geq \sum_{j=1 \wedge j \neq i}^{n} |a_{ij}| \quad \text{(altro uguale)}.$$

Se ho maggiore stretto si dice a diagonale dominante in senso forte (*o stretto*).

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 $\iff \rho(B) < 1$.

Abbiamo anche:

- A tridiagonale, J converge $\iff$ GS converge.

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

18. Laboratorio 06

Per disegnare i grafici si usa il comando «plot(x,y)» con due vettori di uguale dimensione. Il comando crea una nuova finestra se non ce ne sono aperte, altrimenti usa l'ultima finestra grafica utilizzata e sovrascrive il grafico già presente. Per mantenere i grafici usiamo il comando «hold on» oppure a plot passiamo una sequenza di vettori x e y da graficare in sequenza. Per aprire più finestre grafiche usiamo il comando «figure(N)», che genera delle finestre numerate in modo incrementale.

Al comando è possibile passare una serie di specifiche per:

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

Con «doc LineSpec» vengono mostrate le opzioni per lo stile della linea.

Con «axis([minx maxx miny maxy])» vengono modificati i bound del grafico.

I comandi:

- «title(titolo)»;
- «xlabel(label)»;
- «ylabel(label)»;
- «legend(legenda)» (*se più grafici la legenda è da passare in ordine*)

permettono di customizzare il grafico generato.

Il comando «grid on» mostra la griglia.

Con «subplot(righe,colonne,area)» crea una griglia righe $\times$ colonne numerata per riga da 1 che possiamo selezionare per mostrare dei grafici nella stessa finestra grafica.

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 ciò con le anonymous function, definite con la seguente sintassi:

```
nomefunzione = @(parametri) funzione
```

Questa funzione è poi valutabile passando *IN ORDINE* i parametri richiesti.

19. Laboratorio 07

In Matlab non abbiamo degli oggetti per contenere polinomi, ma possiamo rappresentarli come vettori contenenti i coefficienti a partire da quello del grado maggiore (*coefficiente direttivo*) fino a quello del grado minore (*termine noto*).

Possiamo:

- valutare un polinomio in una serie di punti con «polyval(p,points)»;
- prodotto e divisione per uno scalare moltiplicando e dividendo il vettore per lo scalare;
- sommare/sottrarre due polinomi con la funzione «polysum(p,q)» scritta da noi; **ATTENZIONE**: possiamo sommarli se hanno la stessa dimensione oppure mettiamo un pad di zeri;
- prodotto di due polinomi con «conv(p,q)»;
- divisione di due polinomi (*quoziente e resto*) con «deconv(p,q)»;
- derivare un polinomio con «polyder(p)»;
- integrare un polinomio con «polyint(p)» e poi calcolare anche gli integrali definiti;
- calcolare le radici di un polinomio con «roots(p)»;
- costruire un polinomio a partire dalle sue radici con «poly(r)».

20. Laboratorio 08

Dati $n+1$ punti, esiste un unico polinomio di grado n detto **polinomio interpolatore** che, calcolato nei punti, genera identità. In Matlab, il polinomio interpolatore si calcola con il comando «polyfit(x,y,n)».

Il grado del polinomio deve essere **NECESSARIAMENTE** uno in meno della grandezza del sample usato, e questo si fa:

- impostare il grado calcolando la lunghezza e poi meno uno;
- impostare un sample di $n+1$ fissato il grado n.

Non sempre i punti con «linspace» vanno bene: la funzione $f(x) = \frac{1}{1+x^2}$, detto **controesempio di Runge**, non fa diminuire l'errore ma bensì lo aumenta. I **nodi di Chebyshev** permettono di trovare un buon sample che diminuisca l'errore, essi sono nella forma

$$c_k = \cos\left(\frac{\pi(2k-1)}{2n}\right) \quad k = 1, ..., n$$

con n numero di nodi. Questo è per l'intervallo $[-1, 1]$, per l'intervallo generico $[a, b]$ facciamo

$$x_k = \frac{a+b}{2} + \frac{b-a}{2} c_k.$$

I punti si accumulano verso gli estremi del dominio.

Spesso le misurazioni sperimentali si pongono con una relazione lineare ma non in linea perfetta: in questo caso è meglio usare la **retta di regressione** (*meno dispendiosa*) rispetto al polinomio interpolatore (*più dispendioso*). Essa approssima nel senso dei minimi quadrati e 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 e passano per entrambi i punti in esame.

21. Laboratorio 09

Possiamo partizionare un intervallo in m sotto-intervallo e interpolare con Lagrange ogni singolo sotto-intervallo. Il caso più semplice è la **spline lineare interpolante**, ovvero interpolare con delle rette ogni sotto-intervallo. In Matlab possiamo trovare una spline con il comando «griddedInterpolant(x,y,"linear")».

Se H è la massima ampiezza dei sotto-intervalli e $E(H)$ l'errore di approssimazione tra la spline e la funzione in quel sotto-intervallo, si dimostra che $E(H) \leq CH^2$.

Per approssimare una funzione integrale in matlab abbiamo la funzione «integral(f,a,b)». Questo produce un errore assoluto minore di $1e-10$ e/o un errore relativo minore di $1e-6$.

Abbiamo tre formule di quadratura per approssimare un integrale:

- punto medio (semplice e composito);
- trapezi (semplice e composito);
- cavalieri-simpson (semplice e composito).

Per trapezi composita abbiamo in Matlab la funzione «trapz(x,y)».

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

Per punto medio e trapezi l'errore è $O(H^2)$, mentre per cavalieri-simpson l'errore è $O(H^4)$.

22. Laboratorio 10

Possiamo trovare gli zeri di una funzione usando la funzione «fzero(f, [a,b])». **ATTENZIONE**: deve valere l'ipotesi del teorema degli zeri, quindi la funzione nei due estremi deve essere di segno discorde. Inoltre, i punti in cui la funzione tocca l'asse delle x non sono considerati zeri perché non hanno segno discorde ($f(x) = x^2$). Inoltre, se la funzione non è continua potrebbero essere riportati punti di discontinuità. In questi ultimi due casi cade una delle ipotesi del teorema degli zeri.

Il **metodo di bisezione** è un altro modo utile per trovare gli zeri, restringendo sempre di più l'intervallo applicando il teorema degli zeri. Questo metodo converge sempre e l'errore ogni volta è meno della metà dell'intervallo corrente.

Il **metodo di Newton** invece necessita delle derivata della funzione, e soprattutto si rischia una divisione per zero e una non convergenza.

23. Laboratorio 11

La funzione «[T,U] = ode45(f,[a,b],y0)» permette di approssimare un problema di Cauchy in un intervallo $[a, b]$ campionandolo nei tempi T.

Abbiamo anche dei metodi:

- **metodo di Eulero esplicito**, convergente di ordine 1 e assolutamente instabile per valori di h non sufficientemente piccoli;
- **metodo di Eulero implicito**, può essere non convergente di ordine 1 e assolutamente stabile.
- **metodo di Heun**: esplicito convergente di ordine 2 e assolutamente instabile.

24. Laboratorio 12

Per i problemi di Cauchy abbiamo anche il **metodo di Crank-Nicolson** (*o dei trapezi*), di ordine 2 che può però non convergere.

Per risolvere invece sistemi di equazioni differenziali ordinarie abbiamo sempre la funzione «ode45(F, [a,b],Y0)» solo che F deve essere un vettore colonna di funzioni note (*devono essere a loro volta vettoriali in y*) e Y_0 contiene le condizioni iniziali di ogni equazione.