Dispense di

Laboratorio di Calcolo Numerico 1

Prof. Marco Caliari

a.a. 2018/19

Questi appunti non hanno nessuna pretesa di completezza. Sono solo alcune note ed esercizi che affiancano l'insegnamento di Laboratorio di Calcolo Numerico 1. Sono inoltre da considerarsi in perenne "under revision" e pertanto possono contenere discrepanze, inesattezze o errori. Si userà il termine MATLAB per indicare il linguaggio di programmazione comune a GNU Octave e Matlab®.

Questa è la versione del 3 giugno 2019. La versione più aggiornata si trova all'indirizzo

http://profs.scienze.univr.it/caliari/aa1819/calcolo_numerico1/dispense.pdf

Indice

Capitolo 1

Aritmetica floating point

Parte di questo capitolo è presa da [2].

1.1 Cambiamento di base

Per convertire un numero intero x da una base qualsiasi (per esempio, 10) alla base 2 (usata dalla macchina), cominciamo con la sua parte intera. L'osservazione chiave è

$$\begin{aligned}\lfloor x \rfloor &= d_n 2^{n-1} + d_{n-1}2^{n-2} + \ldots + d_2 2^1 + d_1 2^0 = \\ &= 2(d_n 2^{n-2} + d_{n-1}2^{n-3} + \ldots + d_2 2^0) + d_1, \quad d_n = 1\end{aligned}$$

da cui si evince che la cifra d_1 (che può essere 0 oppure 1) si ottiene prendendo il resto della divisione intera del numero $\lfloor x \rfloor$ per 2. A questo punto, basta ripetere il ragionamento con

$$\begin{aligned}\lfloor \lfloor x \rfloor /2 \rfloor &= d_n 2^{n-2} + d_{n-1}2^{n-3} + \ldots + d_2 2^0 = \\ &= 2(d_n 2^{n-3} + d_{n-1}2^{n-4} + \ldots + d_3 2^0) + d_2\end{aligned}$$

e così via. Si arriva a

$$\underbrace{\lfloor \ldots \lfloor \lfloor \lfloor x \rfloor /2 \rfloor /2 \rfloor / \ldots 2 \rfloor}_{n-1 \text{ volte}} = \lfloor \lfloor x \rfloor / 2^{n-2} \rfloor = d_n 2 + d_{n-1}2^0 = 2(d_n 2^0) + d_{n-1}$$

e si finisce con

$$\underbrace{\lfloor \ldots \lfloor \lfloor \lfloor x \rfloor /2 \rfloor /2 \rfloor / \ldots 2 \rfloor}_{n \text{ volte}} = \lfloor \lfloor x \rfloor / 2^{n-1} \rfloor = d_n = 1.$$

L'operazione va ripetuta dunque $n = \lfloor \log_2(x) \rfloor + 1$ volte oppure, equivalentemente, fino a che $\lfloor \lfloor x \rfloor / 2^n \rfloor = 0$.

Per convertire invece la parte decimale $y = x - \lfloor x \rfloor$, $0 \leq y < 1$, si parte dall'osservazione che

$$y = c_1 2^{-1} + c_2 2^{-2} + \ldots + c_{m-1} 2^{-m+1} + c_m 2^{-m} + \ldots$$

(un numero y, anche se con sviluppo decimale finito, può avere sviluppo diadico infinito) da cui

$$2y = c_1 2^0 + c_2 2^{-1} + \ldots + c_{m-1} 2^{-m+2} + c_m 2^{-m+1} + \ldots$$

e

$$\lfloor 2y \rfloor = c_1 2^0 = c_1.$$

Si prosegue con

$$2y - c_1 = c_2 2^{-1} + \ldots + c_{m-1} 2^{-m+2} + c_m 2^{-m+1} + \ldots$$

da cui

$$2(2y - c_1) = c_2 2^0 + c_3 2^{-1} + \ldots + c_{m-1} 2^{-m+3} + c_m 2^{-m+2} + \ldots$$

e quindi

$$\lfloor 2(2y - c_1) \rfloor = c_2 2^0 = c_2.$$

In generale,

$$\underbrace{\lfloor 2(\ldots((2y - c_1) - c_2)\ldots - c_{m-1}) \rfloor}_{m \text{ volte}} = c_m 2^0 = c_m.$$

Si finisce se

$$2(\ldots((2y - c_1) - c_2)\ldots - c_{m-1}) = 0$$

(da cui $c_m = 0$) per un certo m. Oppure, occorre decidere quante cifre calcolare. Al calcolatore, siccome ogni numero inserito (in base 10) viene convertito (ed eventualmente approssimato) in base 2 con 52 cifre diadiche (in rappresentazione normalizzata), si ha che lo sviluppo diadico di y risulta sempre finito. Per concludere, si ha

$$x = (d_n d_{n-1} \ldots d_1 . c_1 c_2 \ldots c_m \ldots)_2.$$

1.2 Precisione doppia

La *rappresentazione normalizzata in virgola mobile* di un numero $x \neq 0$ al calcolatore è

$$\mathrm{fl}(x) = \pm(1.d_1 d_2 \ldots d_m)_2 \cdot 2^p, \quad d_i \in \{0, 1\}, \quad L \leq p \leq U.$$

Ovviamente corrisponde a

$$\mathrm{fl}(x) = \pm \left(1 + \sum_{i=1}^{m} d_i 2^{-i} \right) \cdot 2^p.$$

Il numero m di cifre e il range $L \leq p \leq U$ determina il tipo di dati da rappresentare. La scelta $m = 52$, $L = -1022$, $U = 1023$ corrisponde alla *precisione doppia* secondo lo standard IEEE754. Dunque, quando un numero viene immesso in un sistema che usa la base decimale (tutti) e la precisione doppia, ne vengono calcolate le prime $m + 1$ cifre dello sviluppo diadico

$$\pm \left(1 + \sum_{i=1}^{m+1} d_i 2^{-i} \right) \cdot 2^p.$$

Poi si procede prendendo le prime m cifre (dopo la virgola) di

$$\pm \left(1 + \sum_{i=1}^{m+1} d_i 2^{-i} + \frac{1}{2} 2^{-m} \right) \cdot 2^p.$$

Sembra complicato, ma si tratta semplicemente di arrotondare per difetto se $d_{m+1} = 0$ e per eccesso se $d_{m+1} = 1$. Dunque, la differenza tra il numero x e la sua approssimazione $\mathrm{fl}(x)$ vale al massimo

$$|\mathrm{fl}(x) - x| \leq \frac{1}{2} 2^{p-m}.$$

La cifra d_{m+1} si chiama *guard digit*. Data la finitezza dell'esponente p, si può calcolare il più piccolo numero positivo rappresentabile ed il più grande. Si ha

$$x_{\min} = (1.0 \ldots 0)_2 \cdot 2^L \approx (2.2251)_{10} \cdot 10^{-308}$$
$$x_{\max} = (1.11 \ldots 1)_2 \cdot 2^U = (2 - 2^{-m}) \cdot 2^U \approx (1.7977)_{10} \cdot 10^{308}$$

Questi due numeri si possono ottenere in MATLAB con i comandi `realmin` e `realmax`. In realtà è possibile rappresentare numeri più piccoli di $x_{\min}$. Poiché l'1 prima della virgola non viene memorizzato (essendo comune a tutti numeri eccetto 0), si può effettivamente memorizzare il numero

$$(0.0 \ldots 01)_2 \cdot 2^L = 2^{L-m} \approx (4.9407)_{10} \cdot 10^{-324}.$$

La regione di numeri reali maggiore di $x_{\max}$ si chiama *overflow* e la regione di numeri reali positivi minori di 2^{L-m} si chiama *underflow*. I numeri positivi più piccoli di questo vengono convertiti in 0. I numeri più grandi di $x_{\max}$ vengono convertiti in 2^{U+1} che viene visualizzato come `Inf`.

Teorema 1. *Se* $0 \neq x < 2^{U+1}$*, allora*

$$\left|\frac{\mathrm{fl}(x) - x}{x}\right| \leq 2^{-m-1}.$$

Dimostrazione. Si ha

$$\left|\frac{\mathrm{fl}(x) - x}{x}\right| \leq \frac{2^{-m-1} \cdot 2^p}{(1.d_1 \dots d_m)_2 \cdot 2^p} \leq \frac{2^{-m-1}}{1} = 2^{-m-1}.$$

□

Il numero $u = 2^{-m-1}$ è chimamato *unit roundoff*. Per il solo fatto di convertire un numero in numero macchina si deve prevedere di commettere un *errore relativo* dell'ordine di u. Detto diversamente, non ha senso pretendere che le cifre dopo la sedicesima circa (in base 10, con notazione esponenziale) siano corrette e non ha senso calcolarle. Detto ancora in un altro modo, con la precisione doppia i numeri sono approssimati alla sedicesima (circa) cifra decimale. Ovviamente i numeri che possono essere rappresentati esattamente, lo sono. Un altro modo di scrivere è

$$\mathrm{fl}(x) = x(1 + \epsilon), \quad |\epsilon| \leq u.$$

Un numero strettamente legato a u è la *precisione di macchina* ε*:* è definita come la differenza tra il più piccolo numero macchina più grande di 1 e 1 stesso. Vale

$$(1.0 \dots 01)_2 - 1 = 2^{-m} = 2u = \approx 2.2204 \cdot 10^{-16}.$$

Si ottiene in MATLAB con il comando `eps`. In libri con diverse notazioni, `eps` viene indicato come il più piccolo numero macchina che sommato a 1 produce un numero maggiore di 1. Non è così nel nostro caso. Il numero

$$(1.1 \dots 1)_2 \cdot 2^{-53} = (1 - 2^{-53}) \cdot 2^{-52}$$

è ovviamente rappresentabile e minore di `eps`. Ma

```
>> 1 + (1 - 2 ^ -53) * 2 ^ -52 > 1
ans = 1
```

e anzi

```
>> 1 + (1 - 2 ^ -53) * 2 ^ -52 == 1 + eps
ans = 1
```

Ad ogni modo, i comandi

```
>> 1 + eps >= 1
ans = 1
>> 1 + eps /2 == 1
ans = 1
```

dicono che MATLAB sta usando la doppia precisione con $m = 52$ cifre. Infatti

$$\texttt{1 + eps / 2} = (1.\underbrace{0\ldots01}_{53\text{ cifre}})_2 \cdot 2^0$$

non può essere rappresentato[1]. Il comando `eps` può essere usato anche per calcolare il numero macchina immediatamente più grande di un numero macchina dato: per esempio

```
>> eps (2)
ans = 4.440892098500626e-16
```

perché il numero immediatamente più grande di 2 è

$$(1.0\ldots01)_2 \cdot 2.$$

1.3 Proprietà delle operazioni

Le quattro operazioni fondamentali devono interpretarsi in aritmetica di macchina in questo modo

$$x \star y = \mathrm{fl}(\mathrm{fl}(x) \star \mathrm{fl}(y)).$$

Infatti le operazioni vengono fatte tra i numeri arrotondati con alcune guard digits e il risultato finale arrotondato a sua volta. Purtroppo in aritmetica di macchina molte proprietà delle operazioni e delle funzioni elementari non sono preservate. Vediamo alcuni esempi.

- La somma è commutativa, ma non associativa. Per esempio

```
>> (1 + eps / 2) + eps / 2 > 1
ans = 0
>> (1 + eps / 2 + eps / 2) > 1
ans = 0
>> 1 + (eps / 2 + eps / 2) > 1
ans = 1
```

[1] Di fatto, l'operazione che viene eseguita è $((0.\underbrace{10\ldots0}_{53\text{ cifre}})_2 + (0.\underbrace{0\ldots0}_{53\text{ cifre}})_2) \cdot 2 = 1$.

Questo esempio dice che conviene sommare a partire dai numeri più piccoli.

- Il prodotto è commutativo, ma non associativo e non distributivo rispetto alla somma. Per esempio

```
>> 3.2 * (2.3 - 1.1) == 3.2 * 2.3 - 3.2 * 1.1
ans = 0
```

- Il prodotto tra matrici non è in generale commutativo. Se $AB = BA$ si dice che le due matrici A e B commutano. In alcuni casi però lo è, in aritmetica esatta: $A^2A = AA^2 = A^3$. Ma non in aritmetica di macchina. Lo si verifica facilmente prendendo una matrice `A = randn (4)`.

- $\sin(\pi)$ non fa zero

```
>> sin (pi)
ans = 1.22464679914735e-16
```

Vediamo ora nel dettaglio il fenomeno della *cancellazione* numerica. Vogliamo calcolare l'espressione $(1 + x) - 1$ per x piccolo. Supponiamo di lavorare in base 10 e con $t = 8$. Prendiamo $x = 1.10000000 \cdot 10^{-8}$. Il valore di $1 + x$ nel sistema in uso è 1.00000001. Dunque, $(1 + x) - 1 = 1.00000000 \cdot 10^{-8}$, mentre il risultato corretto rappresentato nel sistema in uso è ovviamente $x = 1.100000000 \cdot 10^{-8}$. L'errore relativo è circa del 10%, non piccolo. Si potrebbe obiettare che l'errore assoluto è circa 10^{-9}, cioè piccolo. Ma se l'operazione di interesse fosse invece $((1-x)-1)/x$, l'errore *assoluto* sarebbe circa 0.1. Se prendiamo invece $x = 1.1 \cdot 10^{-9}$, il risultato finale di $((1 + x) - 1)/x$ è addirittura 0. Il fatto che la precisione di macchina sia molto maggiore ci permette di avere risultati ragionevoli con valori di x anche più piccoli. Ma

```
>> x = 1e-16;
>> ((1 + x) - 1) / x
ans = 0
```

Tornando a $x = 1.10000000 \cdot 10^{-8}$, il problema nasce dal fatto che nell'arrotondamento di $(1 + x) = 1.00000001(10...)$ tutte le cifre tra parentesi vanno perse (si può pensare che siano sostituite da 0, ma questo ci sta) *e poi,* dopo aver sottratto 1, rimane $1.00000000 \cdot 10^{-8}$. Tale fenomeno non è legato ad un particolare esempio sfortunato. Consideriamo infatti $x \neq y$ e valutiamo

l'errore relativo nel calcolare $\mathrm{fl}(x) - \mathrm{fl}(y)$ (supponendo anche che non ci sia bisogno di arrotondare ulteriormente il risultato). Si ha

$$\begin{aligned}\left|\frac{(x-y)-(\mathrm{fl}(x)-\mathrm{fl}(y))}{x-y}\right| &= \left|\frac{(x-\mathrm{fl}(x))-(y-\mathrm{fl}(y))}{x-y}\right| \leq \\ &\leq \left|\frac{x-\mathrm{fl}(x)}{x}\right|\left|\frac{x}{x-y}\right| + \left|\frac{y-\mathrm{fl}(y)}{y}\right|\left|\frac{y}{x-y}\right| \leq \\ &\leq \epsilon_1\left|\frac{x}{x-y}\right| + \epsilon_2\left|\frac{y}{x-y}\right|.\end{aligned}$$

Se dunque ϵ_1 (ϵ_2) è diverso da 0, cioè x (y) è stato approssimato *e se* $x - y$ è un numero piccolo, allora l'errore relativo può essere grande. In aritmetica binaria le cose funzionano allo stesso modo. L'unica cosa da tener presente è che alcuni numeri *sembra* che possano essere rappresentati esattamente, producendo $\epsilon_1 = \epsilon_2 = 0$. Ma la conversione in base 2 può produrre uno sviluppo diadico infinito che deve essere troncato e arrotondato. Le cifre cancellate sono quelle che permettono al numero di avere infiniti zeri nello sviluppo decimale da un certo punto in poi. È il caso proprio di $x = 1.1 \cdot 10^{-8}$. Sembrerebbe che avendo a disposizione circa 16 cifre decimali non ci fossero problemi a calcolare $((1 + x) - 1)$, ma invece

```
>> format long e
>> x = 1.1e-8
x = 1.10000000000000e-08
>> (1 + x) - 1
ans = 1.10000000219657e-08
```

E, ancora più sorprendentemente (forse)

```
>> format long e
>> x = 1.49011611938477e-8
x = 1.49011611938477e-8
>> (1 + x) - 1
ans = 1.49011611938477e-8
```

Questo perché x corrisponde a 2^{-26} e $1 + x$ è perfettamente rappresentabile in base 2.

1.4 Esercizi

1. Scrivere una function `res = polinomio (a, x)` che valuta un polinomio
$$a_1 x^n + a_2 x^{n-1} + \ldots + a_n x + a_{n+1}$$

nel vettore `x`.

2. Scrivere una function `res = horner (a, x)` che valuta un polinomio nella forma di Horner

$$(((a_1x + a_2)x + a_3)x + \ldots + a_n)x + a_{n+1}$$

Quale delle due è più vantaggiosa? Si confrontino le function `polinomio` e `horner` con la function `polyval`.

3. Si implementi un algoritmo che calcola la potenza p-esima di una matrice usando le seguenti relazioni

$$p = d_1 + 2d_2 + \ldots + 2^{n-1}d_n = \sum_{i=1}^{n} 2^{i-1}d_i, \quad d_i \in \{0, 1\},\ d_n = 1$$

$$A^p = A^{\sum_{i=1}^{n} 2^{i-1}d_i} = \prod_{i=1}^{n} A^{2^{i-1}d_i} = \prod_{i=1}^{n} \left(A^{2^{i-1}}\right)^{d_i}$$

$$A^{2^i} = A^{2^{i-1}} A^{2^{i-1}}$$

Si dica quanti prodotti matriciali si risparmiano per il calcolo di A^{17} rispetto all'algoritmo naive

$$A^{17} = \underbrace{(((A \cdot A) \cdot A) \ldots \cdot A)}_{16 \text{ volte}}.$$

4. Si scopra se Matlab® e GNU Octave calcolano A^3 come AA^2 oppure A^2A.

5. Si scopra quanto vale la precisione di macchina della propria calcolatrice, di Excel e di Libreoffice.

6. Si consideri il seguente algoritmo per l'approssimazione della radice quadrata di 2. Come prima cosa, si costruisce un rettangolo di base $x_1 = 1$ e altezza $2/x_1$. Ovviamente ha area 2. Se fosse un quadrato, allora il suo lato sarebbe la radice cercata. Evidentemente x_1 è troppo piccolo perché sia un tale lato. Prendiamo allora x_2 come la media tra la base e l'altezza del rettangolo, cioè $x_2 = (x_1 + 2/x_1)/2 = 1.5$. L'altezza risulta essere $2/1.5 = 4/3$. Si prosegue in questo modo. Quante medie è necessario calcolare per avere l'approssimazione di radice di 2 con 15 cifre decimali corrette?

7. Le radici di $ax^2 + bx + c$, con $a = 10^{-10}$, $b = 1$ e $c = 10^{-4}$ sono ovviamente
$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$
Le si calcolino e poi si provi a valutare il *residuo* $ax_{1,2}^2 + bx_{1,2} + c$. Si moltiplichi adesso la formula risolutiva per
$$\frac{-b \mp \sqrt{b^2 - 4ac}}{-b \mp \sqrt{b^2 - 4ac}}$$
si facciano i calcoli e si ricalcolino le radici. Cosa si osserva?

8. Si calcoli $a^2 - b^2$ con $a = 1.4 \cdot 10^{154}$ e $b = 1.3 \cdot 10^{154}$.

9. Si calcoli la media aritmetica di $a = 1.7 \cdot 10^{308}$ e $b = 1.6 \cdot 10^{308}$.

10. La successione definita per ricorrenza
$$\begin{aligned} z_1 &= 0 \\ z_2 &= 2 \\ z_{n+1} &= 2^{n-1/2}\sqrt{1 - \sqrt{1 - 4^{1-n}{z_n}^2}}, \quad n > 1 \end{aligned}$$
converge a π. La si implementi in MATLAB e si produca un grafico semilogaritmico nelle ordinate dell'errore relativo $|z_n - \pi|/\pi$, per $n = 1, 2, \ldots, 30$. Si individui il problema e si proponga una strategia per risolverlo.

11. Si dimostri che la successione dell'esercizio precedente converge a π. *(Sugg.: l'area del poligono regolare di m lati inscritto nel cerchio unitario è $\frac{m}{2}\sin\frac{2\pi}{m}$. Indicata con z_n l'area del poligono di $m = 2^n$ lati...)*

12. Supponiamo che uno studente abbia media pesata 105.1 e prenda 4.1 punti di tesi e 0.3 punti di valutazione del curriculum. La somma fa 109.5 e la funzione da usare per arrotondare è `round`

```
>> round (0.49)
ans = 0
>> round (0.5)
ans = 1
```

Si calcoli il punteggio finale dello studente in MATLAB

```
>> round (105.1 + 4.1 + 0.3)
```

(Per fortuna i docenti conoscono l'aritmetica di macchina...)

13. Si ripeta l'esercizio sopra convertendo i numeri in *precisione singola*, cioè usando i valori `single (105.1)`, `single (4.1)` e `single (0.3)`. La precisione singola usa (memorizza) solo 23 cifre binarie.

14. Si consideri il seguente algoritmo che somma gli elementi del vettore `a`

```
ret = 0;
c = 0;
for i = 1:length (a)
  y = a(i) - c;
  t = ret + y;
  c = (t - ret) - y; % recupera la parte di y esclusa da t
  ret =  t;
end
```

    e lo si testi nel caso precedente. Funziona bene perché la riga con il commento serve a recuperare la parte di `y` (piccolo rispetto a `ret` se i numeri da sommare sono tutti positivi) che viene persa nella somma con `ret`. Questo algoritmo si chiama *sommatoria di Kahan (o compensata)*.

15. Un altro esempio di cosa succede nel sommare numeri piccoli a numeri grandi è l'approssimazione di $\exp(-10)$ tramite le troncate di Taylor

$$s_n = \sum_{i=0}^{n} \frac{x^i}{i!}, \quad x = -10.$$

    Si analizzi il comportamento del seguente codice

```
ret = 1;
px = x;
for i = 1:n
  ret = ret + px;
  px = px * x / (i + 1);
end
```

Capitolo 2

Equazioni non lineari

2.1 Ordine dei metodi

Data la successione $\{x_j\}_j$ convergente a z, l'ordine di convergenza è p se

$$\lim_{j\to\infty} \frac{|x_{j+1} - z|}{|x_j - z|^p} = C \neq 0.$$

Si ricava dunque

$$p = \lim_{j\to\infty} \frac{\log|x_{j+1} - z| - \log C}{\log|x_j - z|} \approx \lim_{j\to\infty} \frac{\log|x_{j+1} - z|}{\log|x_j - z|}. \tag{2.1}$$

Se non si conosce z, si può usare una sua approssimazione x_J, con J sufficientemente maggiore di $j + 1$.

2.2 Metodo di bisezione

Data la successione $\{x_j\}_j$ prodotta dal metodo di bisezione convergente alla radice z di $f(x)$, il criterio d'arresto basato sul *residuo* $f(x_j)$ (cioè dedurre $x_j \approx z$ se $f(x_j) \approx 0$) non è un buon criterio. Si ha infatti, usando il teorema di Lagrange,

$$f(z) - f(x_j) = f'(x_z)(z - x_j)$$

da cui

$$\left|\frac{f(x_j)}{f'(x_j)}\right| \approx \left|\frac{f(x_j)}{f'(x_z)}\right| = |z - x_j| \tag{2.2}$$

se $x_j \approx z$. È dunque il *residuo pesato* ad essere una buona approssimazione dell'errore assoluto (cosa che avviene con il metodo di Newton). È però certamente vero che se $f(x_j) = 0$ allora $x_j = z$. Tuttavia, a causa degli errori di arrotondamento, l'ipotetica istruzione

```
if (f (x(j)) == 0)
  % trovata la radice
end
```

dovrebbe essere sostituita da

```
if (abs (f (x(j))) <= eps)
  % trovata la radice
end
```

È anche ipotizzabile di moltiplicare `eps` per un opportuno parametro di *sicurezza*, per esempio 10.

2.3 Metodo di Newton

L'iterazione base del metodo di Newton è

$$\delta_j = -f(x_j)/f'(x_j), \quad x_{j+1} = x_j + \delta_j.$$

Per quanto visto in (2.2), si ha

$$|\delta_j| = \left|\frac{f(x_j)}{f'(x_j)}\right| = |x_{j+1} - x_j| \approx |z - x_j|.$$

Dunque, $|\delta_j|$ è una stima di errore assoluto per x_j. D'altra parte, avendolo a disposizione, lo si può sommare a x_j per ottenere x_{j+1}, approssimazione senz'altro migliore (in caso di convergenza), per la quale però $|\delta_j|$ è una sovrastima dell'errore.

2.3.1 Metodo di Newton per radici multiple

Se la molteplicità di una radice z è $r > 1$, allora la funzione di iterazione del metodo di Newton

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

è tale per cui $g'(z) = 1 - 1/r$ e pertanto il metodo di Newton converge solo linearmente. Si può dunque considerare il metodo

$$x_{j+1} = x_j - r\frac{f(x_j)}{f'(x_j)}$$

per il quale la funzione di iterazione $g_r(x)$ soddisfa $g_r'(z) = 0$ e quindi il metodo è del secondo ordine. Ovviamente, bisogna conoscere la molteplicità r.

2.3.2 Metodo di Newton–Horner per le radici dei polinomi

Consideriamo il polinomio

$$p_n(x) = a_1 x^n + a_2 x^{n-1} + \ldots + a_n x + a_{n+1}$$

e la sua valutazione mediante schema di Horner

$$p_n(x) = (((a_1 x + a_2)x + a_3)x + \ldots + a_n) + a_{n+1}.$$

Dati il vettore riga `a` dei coefficienti ed uno scalare `z`, lo schema di Horner si implementa mediante

```
n = length (a) - 1;
b(1) = a(1);
for j = 2:n + 1
  b(j) = b(j - 1) * z + a(j);
end
pnz = b(n + 1);
```

e restituisce $b_{n+1} = p_n(z)$ (si confronti con `polyval (a, z)`). Il polinomio (dipendente da z) definito da

$$g_{n-1}(x; z) = b_1 x^{n-1} + b_2 x^{n-2} + \ldots + b_{n-1} x + b_n$$

è chiamato *polinomio associato* a $p_n(x)$. Valgono le seguenti uguaglianze:

$$\begin{aligned} p_n(x) &= b_{n+1} + (x - z) g_{n-1}(x; z) \\ p_n'(z) &= g_{n-1}(z; z) \end{aligned}$$

Se inoltre z è una radice di $p_n(x)$, allora $b_{n+1} = 0$ e $p_n(x) = (x - z) g_{n-1}(x; z)$ e dunque $g_{n-1}(x; z)$ ha per radici le restanti $n - 1$ radici di $p_n(x)$.

Le considerazioni fatte permettono di implementare in maniera efficiente il metodo di Newton per il calcolo di tutte le radici (anche complesse) dei polinomi. Infatti, dato il punto x_j^1 (approssimazione all'iterazione j del metodo di Newton della prima radice del polinomio $p_n(x)$) e $b_{n+1}^j = p_n(x_j^1)$, l'algoritmo per calcolare x_{j+1}^1 (approssimazione all'iterazione $j+1$ della prima radice del polinomio) è

$$x_{j+1}^1 = x_j^1 - b_{n+1}^j / g_{n-1}(x_j^1; x_j^1)$$

ove $g_{n-1}(x_j^1; x_j^1)$ è calcolato nuovamente mediante lo schema di Horner. Una volta arrivati a convergenza, diciamo con il valore $x_{J_1}^1$, il polinomio in x $g_{n-1}(x; x_{J_1}^1)$ avrà le rimanenti $n-1$ radici di $p_n(x)$ e si applicherà nuovamente

il metodo di Newton–Horner a tale polinomio. Tale tecnica è detta *deflazione* e si arresterà a $g_1(x; x_{J_{n-1}}^{n-1}) = b_1 x + b_2$ per il quale $x^n = -b_2/b_1$. Si osservi che le fattorizzazioni del tipo $p_n(x) = (x - z)g_{n-1}(x; z)$ generate dall'algoritmo sono *approssimate* e pertanto, alla fine del processo, si potrebbero usare le radici approssimate ottenute come punti di partenza per il metodo di Newton applicato al polinomio di partenza.

2.4 Iterazioni di punto fisso

Per il metodo di punto fisso, il test di arresto basato sullo *scarto* $|x_{j+1} - x_j|$ non è sempre un buon criterio. Si ha infatti

$$(x_{j+1} - x_j) + (x_j - z) = x_{j+1} - z = g(x_j) - g(z) = g'(x_z)(x_j - z)$$

da cui

$$x_{j+1} - x_j = (g'(x_z) - 1)(x_j - z)$$

e dunque se $g'(x_z) \approx g'(z) \approx 1$, un errore grande $x_j - z$ potrebbe produrre uno scarto piccolo.

2.5 Esercizi

1. Si applichi un metodo per la ricerca della radice di $f(x) = \sin(x)$ contenuta in $[-\pi/2 + 1000\pi, \pi/2 + 1000\pi]$. Si calcoli il numero di iterazioni necessario quando si richiede una tolleranza relativa pari a 10^{-6} oppure una tolleranza assoluta pari a 10^{-6}.

2. Scrivere la function `[x, its, delta] = puntofisso (g, x0, tol, maxits)` che implementa il metodo delle iterazioni di punto fisso con criterio d'arresto basato sullo scarto.

3. Si individui un metodo di iterazione di punto fisso che converga linearmente alla radice positiva dell'equazione

   $$0 = x^2 - \sin(\pi x)\mathrm{e}^{-x}$$

   ed un metodo di iterazione di punto fisso che converga quadraticamente alla radice nulla.

Capitolo 3

Sistemi lineari

3.1 Sulla norma-2 di matrici

La norma-2 di una matrice è

$$\sup_{\|x\|_2=1} \frac{\|Ax\|_2}{\|x\|_2} = \|A\|_2 = \sqrt{\rho(A^T A)}$$

ove $\|x\|_2 = \sqrt{x^T x}$ e $\rho(A)$ indica il *raggio spettrale* di A (l'autovalore più grande in modulo). È indifferente usare $A^T A$ o AA^T perché gli autovalori sono gli stessi: se λ è autovalore di $A^T A$, allora

$$A^T Ax = \lambda x \Rightarrow (AA^T)(Ax) = \lambda(Ax)$$

e dunque λ è autovalore di AA^T. E viceversa. Poi, $A^T A$ è una matrice simmetrica, e dunque i suoi autovalori sono reali[1]. Sono anche non negativi: infatti se $A^T Ax = \lambda x$, con x di norma-2 unitaria, allora $x^T A^T Ax = \|Ax\|_2^2 = \lambda$.

3.2 Condizionamento di un sistema lineare

Si chiama numero di condizionamento di un sistema lineare $Ax = b$ (con A non singolare) la quantità

$$\text{cond}(A) = \|A\| \|A^{-1}\|.$$

[1] Da $Ax = \lambda x$ e $A\overline{x} = \overline{\lambda}\overline{x}$, si ha $x^* Ax = \lambda x^* x$ e $x^T A\overline{x} = x^* Ax = \overline{\lambda} x^T \overline{x} = \overline{\lambda} x^* x$, dunque $(\lambda - \overline{\lambda}) x^* x = 0$.

Ovviamente esiste un numero di condizionamento per ogni norma (submoltiplicativa) scelta, ma in ogni caso è un numero maggiore o uguale a 1. Infatti

$$\|I\| = \|I \cdot I\| \leq \|I\|\|I\|$$

e dunque

$$1 \leq \|I\| = \|AA^{-1}\| \leq \|A\|\|A^{-1}\| = \text{cond}(A).$$

Supponiamo che b sia diverso dal vettore nullo. Se invece di x un metodo numerico per la soluzione di sistemi lineari calcola $\tilde{x}$, allora possiamo considerare il residuo $r = b - A\tilde{x}$, cioè $A\tilde{x} = b - r$. Confrontata questa equazione con $Ax = b$, si ha

$$A(x - \tilde{x}) = r$$

da cui

$$\|\tilde{x} - x\| \leq \|A^{-1}\|\|r\|.$$

Usando anche disuguaglianza

$$\|b\| \leq \|A\|\|x\|$$

si ottiene

$$\frac{\|\tilde{x} - x\|}{\|x\|} \leq \|A\|\|A^{-1}\|\frac{\|r\|}{\|b\|} = \text{cond}(A)\frac{\|r\|}{\|b\|}.$$

Questa disuguaglianza dice che l'errore relativo rispetto alla vera soluzione del sistema lineare $Ax = b$ è maggiorato dal residuo (quantità calcolabile dopo aver risolto il sistema) *amplificato* dal numero di condizionamento (quantità calcolabile o comunque stimabile).

3.2.1 Le famigerate matrici di Hilbert

Consideriamo una matrice di Hilbert di ordine n

$$H_n = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{n} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \cdots & \vdots \\ \frac{1}{3} & \frac{1}{4} & \ddots & & \\ \vdots & \vdots & & & \\ \frac{1}{n} & \cdots & & & \frac{1}{2n-1} \end{bmatrix}$$

che può essere generata con il comando `hilb (n)`. Sono matrici simmetriche e definite positive, con un numero di condizionamento che cresce come $\mathcal{O}((1+$

$\sqrt{2})^{4n}/\sqrt{n})$. È possibile scrivere esattamente l'inversa di una matrice di Hilbert, che ha elementi

$$(H_n)^{-1}_{ij} = (-1)^{i+j}(i+j-1)\binom{n+i-1}{n-j}\binom{n+j-1}{n-i}\binom{i+j-2}{i-1}^2$$

e può essere generata dal comando `invhilb (n)`. Dunque se ne può calcolare il numero di condizionamento, per esempio

```
>> n = 10;
>> H = hilb (n);
>> norm (H) * norm (invhilb (n))
ans = 1.6026e+13
```

Si preferisce però usare il comando `cond (H)`, che non calcola l'inversa della matrice. È possibile verificare che

```
>> n = 10;
>> H = hilb (n);
>> b = H * (1:n)';
>> x = H \ b; % Risolve il sistema lineare di soluzione (1:n)'
>> norm (x - (1:n)') / norm ((1:n)')
ans = 1.7583e-04
>> r = norm (b - H * x);
>> norm (r)
ans = 1.4621e-14
>> cond (H) * norm (r) / norm (b)
ans = 0.012351
```

Il comando `A \ b` (chiamato *backslash*, equivalente a `mldivide (A, b)`) risolve i sistemi lineari usando il metodo di eliminazione gaussiana con pivoting per righe.

Si potrebbe pensare che in questo caso, poiché si conosce analiticamente l'inversa della matrice, potrebbe risultare conveniente moltiplicare l'inversa per il vettore `b`

```
>> Hinv = invhilb (n);
>> x = Hinv * b;
>> norm (x - (1:n)') / norm ((1:n)')
ans = 5.7040e-05
```

Il risultato è solo leggermente migliore. In generale, calcolare l'inversa di una matrice e poi moltiplicare per il termine noto è un modo più costoso e senza maggior accuratezza rispetto all'eliminazione gaussiana (vedi [1]).

3.3 Sistemi triangolari

Dato un sistema lineare non singolare nella forma

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n-1}x_{n-1} + a_{1n}x_n = b_1 \\ a_{22}x_2 + a_{23}x_3 + \ldots + a_{2n-1}x_{n-1} + a_{2n}x_n = b_2 \\ \ddots \quad \vdots \quad = \vdots \\ a_{n-1n-1}x_{n-1} + a_{n-1n}x_n = b_{n-1} \\ a_{nn}x_n = b_n \end{cases}$$

(si chiama *triangolare* superiore), è possibile risolverlo mediante la tecnica delle *sostituzioni all'indietro*

```
x = b;
x(n) = x(n) / A(n,n);
for i = n-1:-1:1
  x(i) = (x(i) - A(i,i+1:n) * x(i+1:n)) / A(i,i); % * prodotto scalare
end
```

Il numero di operazioni richiesto è $\mathcal{O}(n^2)$. Analogamente si possono risolvere sistemi triangolari inferiori.

3.4 Metodo di eliminazione gaussiana

Il metodo diretto più usato per risolvere sistemi *densi* di equazioni lineari è il metodo dell'eliminazione gaussiana con pivoting per righe. Per "densi", intendiamo che la matrice $A \in \mathbb{R}^{n\times n}$ che definisce il sistema lineare ha un numero di elementi diversi da zero proporzionale a n^2. L'implementazione con scambio effettivo delle righe è

```
n = length (A);
A = [A, b];
for k = 1:n - 1
  [~, idx] = max (abs (A(k:n, k)));
  temp = A(k, :);
  A(k, :) = A(idx + k - 1, :);
  A(idx + k - 1, :) = temp;
  for i = k + 1:n
    mik = A(i, k) / A(k, k);
    A(i, k) = 0;
    for j = k + 1:n + 1
```

```
      A(i, j) = A(i, j) - mik * A(k, j);
    end
  end
end
U = A(:, 1:n);
bhat = A(:, n + 1);
```

Tale implementazione è nota come *kij right-looking* ed è orientata per righe. Ci sono altre cinque versioni. Al termine dell'esecuzione, `U` e `bhat` contengono la matrice triangolare superiore e il termine noto $\hat{b}$ per cui

$$Ax = b \Leftrightarrow Ux = \hat{b}.$$

La strategia del pivoting per righe, con scelta dell'elemento di modulo massimo, non è l'unica possibile. Si può per esempio considerare il *pivoting scalato,* in cui, definito

$$d_i = \max_{1 \le j \le n} |a_{ij}|$$

la riga p, $k \le p \le n$ da selezionare al passo k è quella per cui

$$\frac{|a_{pk}^{(k)}|}{d_p} \ge \frac{|a_{ik}^{(k)}|}{d_i}, \quad k \le i \le n.$$

Con la scelta di questo pivoting, l'eliminazione gaussiana è molto simile a quanto realizza il comando \ di Matlab® per matrici dense.

3.4.1 Fattorizzazione LU

Dal metodo di eliminazione gaussiana, memorizzazando i moltiplicatori m_{ik} si ottiene la fattorizzazione LU ($PA = LU$) di una matrice. Con il comando `lu` di MATLAB, quando invocato con due argomenti di output `[out1, out2] = lu (A)`, si ottiene $U =$`out1` e $P^{-1}L =$`out2`. Quest'ultima matrice *non* è triangolare inferiore.

La fattorizzazione LU di una matrice si usa ogniqualvolta sia necessario risolvere più di un sistema lineare con la stessa matrice e diversi termini noti.

Matrici di permutazione

Dato un vettore di permutazione p di lunghezza n, è possibile creare la corrispondente matrice di permutazione P in MATLAB con i comandi

```
I = eye (n);
P = I(p, :);
```

In tal modo, dato un vettore colonna qualsiasi v, si ha

```
>> n = 4;
>> p = [1; 4; 2; 3];
>> I = eye (n);
>> P = I(p, :);
>> v = rand (n, 1)
v =

   0.28869
   0.18484
   0.71816
   0.99014
>> P * v
ans =

   0.28869
   0.99014
   0.18484
   0.71816
>> v(p)
ans =

   0.28869
   0.99014
   0.18484
   0.71816
```

Se si vuole invece applicare la permutazione inversa $P^{-1}v$, si può procedere in questo modo

```
>> w = zeros (n, 1);
>> w(p) = v
ans =

   0.28869
   0.71816
   0.99014
   0.18484
>> P\v
ans =
```

```
   0.28869
   0.71816
   0.99014
   0.18484
```

Infine, se si vuole trovare p a partire da P, si può usare il comando

```
>> find (P') - (0:n:(n-1)*n)'
ans =

   1
   4
   2
   3
```

3.5 Esercizi

1. Si implementi una function `C = matmul (A, B)` che calcola il prodotto matriciale AB, A e B due matrici rettangolari dalle dimensioni compatibili, senza usare l'operazione `A*B`.

2. Si verifichi numericamente la stima del numero di condizionamento per le matrici di Hilbert.

3. Si risolvano sistemi lineari con matrici di Hilbert fino a ordine 15.

4. Si implementi la function `x = bs (A, b)` che risolve un sistema lineare triangolare superiore (`bs` sta per *backward substitutions*). Si può testare su matrici generate da `A = triu (randn (10))`.

5. Si implementi la function `x = fs (A, b)` che risolve un sistema lineare triangolare inferiore (`fs` sta per *forward substitutions*). Si può testare su matrici generate da `A = tril (randn (10))`.

6. Si implementi una function `x = triang (A, b)` che risolve sistemi lineari che, previo scambio di righe, sono triangolari superiori. Può essere utile il comando `find (x)` (consultarne l'`help`). Si provi ad implementare la function senza scambiare effettivamente le righe della matrice *(difficile...)*

7. Gli algoritmi proposti lavorano per righe. Si consideri l'algoritmo qui schematizzato

$$\begin{bmatrix} \hat{A} & \hat{a} \\ 0 & a_{nn} \end{bmatrix} \begin{bmatrix} \hat{x} \\ x_n \end{bmatrix} = \begin{bmatrix} \hat{b} \\ b_n \end{bmatrix}$$

ove $\hat{a}$ è un vettore colonna di dimensione $n-1$ e $\hat{A}$ una matrice triangolare superiore di dimensione $(n-1) \times (n-1)$. Si può ricavare

$$x_n = \frac{b_n}{a_{nn}}, \quad \hat{b}' = \hat{b} - \hat{a}x_n$$

e poi risolvere il sistema triangolare superiore $\hat{A}\hat{x} = \hat{b}'$ e così via. Si implementi questo algoritmo, sia per sistemi triangolari superiori che inferiori, in forma iterativa. Si testi la velocità di esecuzione (con i comandi `tic` e `toc`) per sistemi lineari con matrici

```
n = 10000; A = triu (randn (n)) + n * eye (n); b = randn (n, 1);
```

8. Si implementi il metodo di eliminazione gaussiana con pivoting scalato per righe. Lo si confronti con il pivoting classico per la soluzione del sistema lineare
$$\begin{cases} x_1 + 200x_2 = 100 \\ x_1 + x_2 = 1 \end{cases}$$
calcolando l'errore rispetto alla soluzione analitica.

9. Si generino sistemi lineari random di dimensione 10 e si testi la function `lup0_kij`, sia nella versione con due argomenti di output che con tre. Si confrontino i risultati con il comando \.

10. Data la fattorizzazione $PA = LU$ si calcoli la soluzione del sistema lineare $A^2x = b$.

11. Si usi il metodo di eliminazione gaussiana per calcolare l'inversa di una matrice data.

12. Si usi il metodo di eliminazione gaussiana per calcolare il determinante di una matrice data.

13. Si implementi una function `x = cramer (A, b)` che realizza il metodo di Cramer per la soluzione di un sistema lineare $Ax = b$
$$x_i = \frac{\det(A_i)}{\det(A)}$$
ove A_i, in notazione MATLAB, è `[A(:, 1:i - 1), b, A(:,i + 1:n)]`.

Capitolo 4

Interpolazione e approssimazione polinomiale

4.1 Matrice di Vandermonde

Il problema dell'interpolazione polinomiale di grado n in una dimensione è *unisolvente:* esiste un unico polinomio di grado (al più) n che interpola $n+1$ coppie di punti $\{(x_i, y_i)\}_{i=1}^{n+1}$, con $x_i \neq x_j$ se $i \neq j$. Significa che il sistema lineare

$$\begin{bmatrix} x_1^n & x_1^{n-1} & \dots & x_1 & 1 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ x_{n+1}^n & x_{n+1}^{n-1} & \dots & x_{n+1} & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_{n+1} \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_{n+1} \end{bmatrix}$$

ha un'unica soluzione. La matrice di tale sistema lineare si chiama *di Vandermonde*.

In MATLAB si può usare il comando `vander` per calcolare la matrice di Vandermonde e risolvere il sistema lineare. Successivamente, con il comando `polyval` si può valutare il polinomio nel punto x.

4.2 Polinomi elementari di Lagrange

Si può costruire esplicitamente il polinomio interpolatore, attraverso i polinomi elementari di Lagrange

$$L_i(x) = \frac{(x-x_1)\cdot(x-x_2)\cdot\ldots\cdot(x-x_{i-1})\cdot(x-x_{i+1})\cdot\ldots\cdot(x-x_{n+1})}{(x_i-x_1)\cdot(x_i-x_2)\cdot\ldots\cdot(x_i-x_{i-1})\cdot(x_i-x_{i+1})\cdot\ldots\cdot(x_i-x_{n+1})}.$$

Sono polinomi di grado n e $L_i(x_j) = \delta_{ij}$. Dunque, il polinomio definito da

$$p_n(x) = \sum_{j=1}^{n+1} y_j L_j(x)$$

interpola le coppie, cioè $p_n(x_i) = y_i$.

La rappresentazione con polinomi elementari di Lagrange restituisce direttamente la valutazione del polinomio, senza calcolarne esplicitamente i coefficienti.

4.3 Interpolazione nella forma di Newton

Il polinomio di interpolazione può essere scritto nella forma di Newton

$$\begin{aligned} p_n(x) &= d_1 + d_2(x - x_1) + d_3(x - x_1)(x - x_2) + \ldots + d_{n+1}(x - x_1)(x - x_2)\ldots(x - x_n) = \\ &= p_{n-1}(x) + d_{n+1}(x - x_1)(x - x_2)\ldots(x - x_n) \end{aligned}$$

ove $\{d_i\}_{i=1}^{n+1}$ sono le differenze divise

$$d_i = f[x_1, x_2, \ldots, x_{i-1}, x_i].$$

Poiché sono invarianti per permutazione, si possono costruire ricorsivamente con il seguente algoritmo

$$\begin{cases} f[x_i] = f(x_i) \\ f[x_{i_1}, x_{i_2}] = \dfrac{f[x_{i_2}] - f[x_{i_1}]}{x_{i_2} - x_{i_1}} \\ \vdots \quad = \quad \vdots \\ f[x_{i_1}, x_{i_2}, \ldots, x_{i_{j-1}}, x_{i_j}] = \dfrac{f[x_{i_2}, \ldots, x_{i_{j-1}}, x_{i_j}] - f[x_{i_1}, x_{i_2}, \ldots, x_{i_{j-1}}]}{x_{i_j} - x_{i_1}} \end{cases}$$

La notazione $f[\ldots]$ lascia intendere che si sta interpolando una funzione $f(x)$. In realtà, basta conoscere i valori $y_i = f(x_i)$ per poter calcolare le differenze divise. Possiamo scrivere due algoritmi: uno che non usa vettori ausiliari

```
for i = 1:n + 1
  d(i) = y(i);
  for j = 1:i - 1
    d(i) = (d(i) - d(j)) / (x(i) - x(j));
  end
end
```

e uno che usa un vettore ausiliario `t` (che verrà comodo in seguito)

```
t(1) = y(1);
d(1) = t(1);
for i = 1:n + 1
  t(i) = y(i);
  for j = i - 1:-1:1
    t(j) = (t(j + 1) - t(j)) / (x(i) - x(j));
  end
  d(i) = t(1);
end
```

Entrambi gli algoritmi possono essere modificati per valutare direttamente il polinomio nel punto x. Se poniamo

$$\pi_i(x) = \prod_{j=1}^{i+1}(x - x_j)$$

la formula di rappresentazione dell'errore di interpolazione dice

$$f(x) - p_i(x) = \frac{f^{(i+1)}(\xi_i)}{(i+1)!}\pi_i(x)$$

ove ξ_i è un punto che sta nel più piccolo intervallo chiuso contenente $x \cup \{x_j\}_{j=1}^{i+1}$. Abbiamo anche

$$f(x) - p_i(x) = f[x_1, x_2, \ldots, x_{i+1}, x]\pi_i(x).$$

Infatti, $p_i(t) + f[x_1, x_2, \ldots, x_{i+1}, x]\pi_i(t)$ è per costruzione il polinomio in t di grado $i+1$ che interpola f nei nodi $x_1, x_2, \ldots, x_{i+1}, x$. Confrontando le due formule di rappresentazione dell'errore, si ha

$$\frac{f^{(i+1)}(\xi_i)}{(i+1)!} = f[x_1, x_2, \ldots, x_{i+1}, x]$$

e dunque

$$\begin{aligned} f(x) - p_i(x) = \frac{f^{(i+1)}(\xi_i)}{(i+1)!}\pi_i(x) = f[x_1, x_2, \ldots, x_{i+1}, x]\pi_i(x) \approx \\ \approx f[x_1, x_2, \ldots, x_{i+1}, x_{i+2}]\pi_i(x) = \\ = d_{i+2}(x - x_1)(x - x_2)\ldots(x - x_{i+1}). \end{aligned}$$

D'altra parte,

$$p_{i+1}(x) = p_i(x) + d_{i+2}(x - x_1)(x - x_2)\ldots(x - x_{i+1}).$$

Quindi, arrivati al calcolo di $p_i(x)$, se $|d_{i+2}\pi_i(x)|$ è più piccolo di una tolleranza prefissata possiamo decidere di interrompere il calcolo. In realtà, a causa delle approssimazioni introdotte, è meglio interrompere il calcolo quando

$$|d_{i+2}\pi_i(x)| + |d_{i+3}\pi_{i+1}(x)| \leq \text{tol} \cdot |p_i(x)|. \tag{4.1}$$

4.3.1 Interpolazione con nodi ripetuti

Supponiamo che tra i nodi di interpolazione ve ne siano alcuni ripetuti. A meno di un riordinamento, supponiamo che i nodi ripetuti siano consecutivi. Siano dunque

$$\underbrace{x_1, x_2, \ldots, x_{m_1}}_{=z_1}, \underbrace{x_{m_1+1}, x_{m_1+2}, \ldots, x_{m_1+m_2}}_{=z_2}, \ldots, \underbrace{x_{m_{j-1}+1}, x_{m_{j-1}+2}, \ldots, x_{m_{j-1}+m_j}}_{=z_j},$$

cioè il punto z_i ha molteplicità m_i. Definiamo

$$f[\underbrace{x, x, \ldots, x}_{k \text{ volte}}] = \frac{f^{(k-1)}(x)}{(k-1)!}$$

e calcoliamo le differenze divise $\{d_i\}_{i=1}^{n+1}$ *come al solito*. Allora l'interpolazione nella forma di Newton soddisfa

$$p_n^{(k)}(z_i) = f^{(k)}(z_i), \quad k = 1, 2, \ldots, m_i, \quad i = 1, 2, \ldots, j.$$

4.4 Approssimazione ai minimi quadrati

Supponiamo di avere molti dati $\{(x_i, y_i)\}_{i=1}^{n+1}$, eventualmente affetti da errore, e di volerli approssimare con un polinomio di grado m basso ($m \ll n$). Possiamo tentare di minimizzare

$$\sum_{i=1}^{n+1} (y_i - p_m(x_i))^2$$

ove i gradi di libertà (le incognite) sono i coefficienti del polinomio, cioè $a_1, a_2, \ldots, a_{m+1}$. Quindi, vogliamo minimizzare

$$\Phi(\boldsymbol{a}) = \|\boldsymbol{y} - V\boldsymbol{a}\|_2^2 = \boldsymbol{y}^T\boldsymbol{y} - 2\boldsymbol{a}^T V^T \boldsymbol{y} + \boldsymbol{a}^T V^T V \boldsymbol{a}$$

ove V è la matrice di Vandermonde rettangolare di grado m. Consideriamo il seguente vettore

$$\boldsymbol{a}^* = (V^T V)^{-1}(V^T \boldsymbol{y})$$

e calcoliamo $\Phi(\boldsymbol{a}) = \Phi((\boldsymbol{a} - \boldsymbol{a}^*) + \boldsymbol{a}^*)$. Si ha

$$\begin{aligned}\Phi(\boldsymbol{a}) = \boldsymbol{y}^T\boldsymbol{y} - 2(\boldsymbol{a} - \boldsymbol{a}^*)^T V^T\boldsymbol{y} + (\boldsymbol{a} - \boldsymbol{a}^*)^T V^T V(\boldsymbol{a} - \boldsymbol{a}^*) + \\ - 2\boldsymbol{a}^{*T} V^T\boldsymbol{y} + \boldsymbol{a}^{*T} V^T V\boldsymbol{a}^* + \\ + 2(\boldsymbol{a} - \boldsymbol{a}^*)^T V^T V\boldsymbol{a}^*.\end{aligned}$$

Osserviamo che

$$V^T V\boldsymbol{a}^* = V^T\boldsymbol{y}$$

e dunque

$$\begin{aligned}\Phi(\boldsymbol{a}) = \boldsymbol{y}^T\boldsymbol{y} - 2(\boldsymbol{a} - \boldsymbol{a}^*)^T V^T\boldsymbol{y} + (\boldsymbol{a} - \boldsymbol{a}^*)^T V^T V(\boldsymbol{a} - \boldsymbol{a}^*) + \\ - 2\boldsymbol{a}^{*T} V^T\boldsymbol{y} + \boldsymbol{a}^{*T} V^T V\boldsymbol{a}^* + \\ + 2(\boldsymbol{a} - \boldsymbol{a}^*)^T V^T\boldsymbol{y} = \\ = \boldsymbol{y}^T\boldsymbol{y} + (\boldsymbol{a} - \boldsymbol{a}^*)^T V^T V(\boldsymbol{a} - \boldsymbol{a}^*) + \\ - 2\boldsymbol{a}^{*T} V^T\boldsymbol{y} + \boldsymbol{a}^{*T} V^T V\boldsymbol{a}^* = \Phi(\boldsymbol{a}^*) + (\boldsymbol{a} - \boldsymbol{a}^*)^T V^T V(\boldsymbol{a} - \boldsymbol{a}^*) = \\ = \Phi(\boldsymbol{a}^*) + \|V(\boldsymbol{a} - \boldsymbol{a}^*)\|_2^2 \geq \Phi(\boldsymbol{a}^*).\end{aligned}$$

Quindi, il vettore $\boldsymbol{a}^*$ rappresenta i coefficienti del polinomio dei *minimi quadrati* e per trovarlo basta risolvere il sistema lineare (quadrato)

$$(V^T V)\boldsymbol{a}^* = V^T\boldsymbol{y}.$$

In MATLAB, si può fare anche `astar = V \ y`, oppure usare il comando `polyfit`.

4.5 Esercizi

1. *Esempio di Runge.* Si consideri l'interpolazione della seguente funzione

$$y = \frac{1}{1 + x^2}$$

nell'intervallo $[-5, 5]$. Si disegni il polinomio interpolatore su n nodi equispaziati (sia usando la matrice di Vandermonde che i polinomi elementari di Lagrange), con $n = 2, 4, \ldots, 22$.

2. Si ripeta l'esercizio precedente usando, invece di nodi equispaziati, i seguenti

$$x_i = 5\cos\left(\frac{(2i-1)\pi}{2(n+1)}\right), i = 1, 2, \ldots, n+1$$

Cosa si nota?

3. I nodi introdotti sopra (detti *di Chebyshev*, se riferiti all'intervallo $(-1,1)$) dovrebbero essere simmetrici nell'intervallo $(-1,1)$. Ma in aritmetica di macchina non lo sono esattamente (mostrarlo). Trovare una formula equivalente che li renda numericamente simmetrici. *(Sugg.:* $\cos(x) = \sin(\pi/2 - x) \ldots$*)*

4. Si scriva la matrice di Vandermonde del sistema lineare corrispondente alle seguenti condizioni di interpolazione

$$\begin{cases} p_3(0) = 1 \\ p_3'(0) = 1 \\ p_3(1) = \mathrm{e} \\ p_3(2) = \mathrm{e}^2 \end{cases}$$

   Si trovi il polinomio e lo si disegni nell'intervallo $[0,2]$, assieme alla funzione e^x.

5. Per l'esempio precedente, si verifichi che l'interpolazione nella forma di Newton restituisce lo stesso risultato.

6. Si implementi un algoritmo che riordina un insieme dato di nodi $X = \{x_i\}_{i=1}^{n+1}$ in un insieme $Z = \{z_i\}_{i=1}^{n+1}$ tale che

$$|z_1| = \max_{x_i \in X} |x_i|$$
$$\prod_{k=1}^{j} |z_{j+1} - z_k| = \max_{x_i \in X} \prod_{k=1}^{j} |x_i - z_k|, \quad j = 1, 2, \ldots, n$$

   Questo ordinamento si chiama *alla Leja*.

7. (Tratto da [3]) Si considerino i nodi di Chebyshev introdotti nell'esercizio 3 per valori di n tra 50 e 80 e vi si interpoli la funzione

$$f(x) = \sqrt{1+x}.$$

   Si calcoli il massimo errore tra $f(x)$ e il polinomio di interpolazione $p_n(x)$, valutandoli su $n+1$ nodi equispaziati in $[-1,1]$. Si ripeta l'esercizio preordinando i nodi di interpolazione come suggerito nell'esercizio precedente.

8. Si modifichi il file `intpolyvalnewt.m` in modo che accetti una tolleranza `tol` e interrompa il calcolo quando (4.1) è soddisfatta. *(Si può sostituire il ciclo* `for` *con* `while`*, oppure usare il comando* `break`*.)*

9. *Compressione.* Supponiamo di avere molti dati $\{(x_i, y_i)\}_{i=1}^{n}$, frutto di misurazioni. Vogliamo comprimerli in un polinomio di grado $m \ll n$ che risulti interpolante su $m+1$ dati. Quali coppie $\{(x_{i_j}, y_{i_j}\}_{j=1}^{m+1}$ scegliamo per interpolare? *(Sugg.: Leja...)*

10. Dimostrare che la matrice $V^T V$ è invertibile e quindi $\boldsymbol{a}^*$ ben definito.

11. Si generino 1000 coppie di dati in questo modo

```
x = rand (1000, 1);
y = 2 * x + 1 + randn (1000, 1) / 10;
```

e si calcoli la *retta di regressione lineare* (polinomio di grado uno che meglio approssima nel senso dei minimi quadrati).

Capitolo 5

Quadratura

La function di base per la quadratura in MATLAB è `quad`, la cui sintassi di base è

```
>> quad (fun, a, b)
```

Le versioni più recenti di Matlab® hanno introdotto anche `integral`. Queste function richiescono di solito a calcolare bene anche integrali impropri ed evitare punti di singolarità. Ma non sono infallibli. Per esempio, per approssimare il valore di

$$\int_{-\infty}^{\infty} x^2 \mathrm{e}^{-x^2} \mathrm{d}x = \frac{\sqrt{\pi}}{2} \approx 0.886227$$

in Matlab® R2019a si può fare

```
>> quad (@(x) x .^ 2 .* exp (-x .^ 2), -10, 10)
ans =
     8.862275289520790e-01
>> quad (@(x) x .^ 2 .* exp (-x .^ 2), -20, 20)
ans =
     2.341224229802663e-07
>> quad (@(x) x .^ 2 .* exp (-x .^ 2), -30, 30)
ans =
     3.697500737084623e-18
>> quad (@(x) x .^2 .* exp (-x .^2), -30, 31)
ans =
     8.862273628583137e-01
```

Poiché l'integranda è simmetrica, meglio usare questa proprietà

```
>> 2 * quad(@(x) x .^2 .* exp (-x .^2 ), 0, 30)
ans =
     8.862278125547811e-01
>> 2*quad(@(x) x.^2.*exp(-x.^2),0,50)
ans =
     8.862118289009481e-01
```

O no?

```
>> 2 * quad (@(x) x .^ 2 .* exp (-x .^ 2), 0, 100)
ans =
     8.505568796545164e-18
```

5.1 Formula di quadratura del rettangolo

La più semplice formula di quadratura è quella del *rettangolo,* che prevede di approssimare un integrale definito

$$\int_a^b f(x)\mathrm{d}x$$

con

$$(b-a)f(x_1), \quad x_1 \in [a,b].$$

Con un po' di fortuna (un po' tanta), potremmo prendere x_1 in modo da avere il valore esatto dell'integrale (ce lo assicura il teorema della media integrale). In realtà, i valori solitamente usati per x_1 sono a, b, oppure $(a+b)/2$. In quest'ultimo caso parliamo di *formula del punto medio.* Si ha, se $f \in \mathcal{C}^2(a,b)$,

$$\int_a^b f(x)\mathrm{d}x = (b-a)f\left(\frac{a+b}{2}\right) + \frac{(b-a)^3}{24}f''(\xi), \quad \xi \in (a,b).$$

Tale formula è dunque *esatta* se $f(x)$ è un polinomio di grado uno e, in generale, accurata solo quando $b-a$ è piccolo.

5.1.1 Formule di Eulero esplicito e implicito e del punto medio implicito

La più importante applicazione della formula del rettangolo si ha nella risoluzione di equazioni differenziali del primo ordine. Dato infatti

$$\begin{cases} y'(t) = f(t, y(t)), & t \in [t_0, T] \\ y(t_0) = y_0 \end{cases}$$

la soluzione analitica può essere scritta come

$$y(t) = y(t_0) + \int_{t_0}^{t} f(s, y(s))\mathrm{d}s.$$

Infatti,

$$y(t) - y(t_0) = \int_{t_0}^{t} y'(s)\mathrm{d}s = \int_{t_0}^{t} f(s, y(s))\mathrm{d}s.$$

La sua approssimazione al tempo $t_1 = t_0 + k$ può essere scritta come

$$y(t_1) = y(t_0) + \int_{t_0}^{t_1} f(s, y(s))\mathrm{d}s \approx y(t_0) + kf(t_0, y(t_0)).$$

Ne viene fuori lo schema di *Eulero esplicito*

$$\begin{cases} y_{n+1} = y_n + kf(t_n, y_n), & n = 0, 1, \ldots, m-1 \\ y_0 = y(t_0) \end{cases}$$

ove $k = (T - t_0)/m$.

Si può pensare anche di usare l'approssimazione

$$y(t_1) = y(t_0) + \int_{t_0}^{t_1} f(s, y(s))\mathrm{d}s \approx y(t_0) + kf(t_1, y(t_1)).$$

Lo schema che ne deriva si chiama *Eulero implicito*

$$\begin{cases} y_{n+1} = y_n + kf(t_{n+1}, y_{n+1}), & n = 0, 1, \ldots, m-1 \\ y_0 = y(t_0) \end{cases}$$

e richiede, ad ogni passo temporale n, di risolvere l'equazione

$$0 = F_n(x) = x - y_n - kf(t_{n+1}, x)$$

tramite il metodo di bisezione (poco usato) o il metodo di Newton, a partire dalla soluzione iniziale $x_1 = y_n$. Si ha ovviamente

$$F_n'(x) = 1 - k\frac{\partial f(t_{n+1}, x)}{\partial y}.$$

Infine, si può pensare di usare l'approssimazione

$$y(t_1) = y(t_0) + \int_{t_0}^{t_1} f(s, y(s))\mathrm{d}s \approx y(t_0) + kf((t_0 + t_1)/2, (y(t_0) + y(t_1))/2).$$

Lo schema che ne deriva si chiama *punto medio implicito*

$$\begin{cases} y_{n+1} = y_n + kf((t_n + t_{n+1})/2, (y_n + y_{n+1})/2), & n = 0, 1, \ldots, m-1 \\ y_0 = y(t_0) \end{cases}$$

ed è il più accurato dei tre.

5.2 Formula dei rettangoli

La formula dei rettangoli si scrive

$$\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{n}\sum_{i=1}^{n} f(x_i)$$

ove $x_i = x_0 + (i-1)(b-a)/n$, $i = 1, 2, \ldots, n$. Il nodo x_0 va scelto in $[a, a+(b-a)/n]$. La formula è esatta per polinomi di grado zero e di ordine $h = (b-a)/n$ in generale, a meno di prendere $x_0 = a + (b-a)/(2n)$, per cui diventa esatta per polinomi di grado uno e di ordine $\mathcal{O}(h^2)$.

5.3 Formula dei trapezi

La formula dei trapezi si scrive

$$\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{2n}\left(f(x_1) + 2\sum_{i=2}^{n} f(x_i) + f(x_{n+1})\right)$$

ove $x_i = a + (i-1)(b-a)/n$, $i = 1, 2, \ldots, n+1$. Se definiamo il vettore riga dei pesi `w = [1, 2 * ones(1, n - 1), 1]` e il vettore colonna dei nodi di quadratura `x = linspace (a, b, n + 1)'`, allora la formula di quadratura si può calcolare come

```
(b - a) / (2 * n) * (w * f (x))
```

ove l'ultimo `*` è un prodotto scalare. La formula è esatta per polinomi di grado uno e di ordine $\mathcal{O}(h^2)$.

5.3.1 Formula dei trapezi per funzioni periodiche

La formula dei trapezi per funzioni periodiche con $f(a) = f(b)$ si semplifica in

$$\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{n}\sum_{i=1}^{n} f(x_i)$$

5.4 Formula di Cavalieri–Simpson

La formula di Cavalieri–Simpson si scrive (per n pari)

$$\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{3n}\left(f(x_1) + 4f(x_2) + 2f(x_3) + \ldots + 2f(x_{n-1}) + 4f(x_n) + f(x_{n+1})\right)$$

e si può scrivere attraverso il vettore di pesi

```
w = [1, repmat([4, 2], 1, n / 2 - 1), 4 , 1]
```

La formula è esatta per polinomi di grado tre e di ordine $\mathcal{O}(h^4)$.

5.4.1 Formula di Cavalieri–Simpson adattativa

Indichiamo con $S_3(a,b)$ la formula di Cavalieri–Simpson di passo $h = (b-a)/2$ (quindi a tre punti)

$$S_3(a,b) = \frac{h}{3}(f(a) + 4f(a+h) + f(b)) = \int_a^b f(x)\mathrm{d}x + \frac{b-a}{180}h^4 f^{(4)}(\xi).$$

Si ha quindi

$$S_5(a,b) = S_3\left(a, \frac{a+b}{2}\right) + S_3\left(\frac{a+b}{2}, b\right) = \int_a^b f(x)\mathrm{d}x + \left(\frac{h}{2}\right)^4 \frac{b-a}{180} f^{(4)}(\bar{\xi}) = \\ = \int_a^b f(x)\mathrm{d}x + \frac{1}{90}\frac{h^5}{16} f^{(4)}(\bar{\xi}).$$

Supponiamo che $f^{(4)}(\xi) = f^{(4)}(\bar{\xi})$. Si ha allora

$$S_3(a,b) - S_3\left(a, \frac{a+b}{2}\right) - S_3\left(\frac{a+b}{2}, b\right) = \frac{b-a}{180} f^{(4)}(\bar{\xi}) \left(h^4 - \left(\frac{h}{2}\right)^4\right) = \\ = \frac{1}{90}\frac{15}{16} h^5 f^{(4)}(\bar{\xi})$$

da cui

$$\left| \int_a^b f(x)\mathrm{d}x - S_3\left(a, \frac{a+b}{2}\right) - S_3\left(\frac{a+b}{2}, b\right) \right| = \\ \frac{1}{15} \left| S_3(a,b) - S_3\left(a, \frac{a+b}{2}\right) - S_3\left(\frac{a+b}{2}, b\right) \right|.$$

Dunque, se si vuole un errore assoluto massimo pari a tol, basta verificare che

$$\left| S_3(a,b) - S_3\left(a, \frac{a+b}{2}\right) - S_3\left(\frac{a+b}{2}, b\right) \right| \leq 15 \cdot \mathrm{tol}.$$

Se è così, allora

$$S_3\left(a, \frac{a+b}{2}\right) + S_3\left(\frac{a+b}{2}, b\right)$$

è l'approssimazione desiderata, altrimenti si applica ricorsivamente questo ragionamento su ognuno degli intervalli $[a, (a+b)/2]$ e $[(a+b)/2, b]$, chiedendo tolleranza tol/2. Si preferisce usare $10 \cdot \mathrm{tol}$ invece di $15 \cdot \mathrm{tol}$.

5.5 Verifica dell'ordine

Supponiamo di avere una formula di ordine p rispetto ad h, cioè il suo errore è

$$e(h) = \mathcal{O}(h^p) = \mathcal{O}(n^{-p})$$

(per la precisione, sarebbe $\mathcal{O}((n+1)^{-p})$). Allora, dati due valori h_1 e h_2 qualunque si ha

$$\frac{e(h_1)}{e(h_2)} = \left(\frac{h_1}{h_2}\right)^p = \left(\frac{n_2}{n_1}\right)^p$$

da cui

$$\log \frac{e(h_1)}{e(h_2)} = \log e(h_1) - \log e(h_2) = p(\log h_1 - \log h_2) = -p(\log n_1 - \log n_2).$$

Questo significa che se in un grafico si rappresentano in ascisse i valori $\log h$ (oppure $\log n$) e in ordinate i corrispondenti valori $\log e(h)$, si ottengono dei punti allineati ad una retta di pendenza p (o $-p$). La stessa cosa se invece dei logaritmi si rappresentano i valori originari in un grafico *logaritmico*.

Metodo	Errore
Rettangoli ($x_0 = a$)	$f'(\xi)\frac{b-a}{2}h$
Rettangoli ($x_0 = a + h/2$)	$f''(\xi)\frac{b-a}{24}h^2$
Trapezi	$-f''(\xi)\frac{b-a}{12}h^2$
Cavalieri–Simpson	$-f^{(4)}(\xi)\frac{b-a}{180}h^4$

Tabella 5.1: Espressioni degli errori per $n+1$ punti (n pari per Cavalieri–Simpson), $h = (b-a)/n$.

5.6 Esercizi

1. Implementare la function `rettangoli (fun, n, a, b, x0)` e la function `trapezi (fun, n, a, b)`.

2. Implementare la function `euleroimplicito (odefun, tspan, y0)`.

3. Si calcoli analiticamente
$$\int_{-1}^{1} |x|^{p/2} \mathrm{d}x$$

per p intero non negativo. Si dica per quali valori di p le formule dei trapezi e di Cavalieri–Simpson convergono con l'ordine aspettato, rispettivamente $\mathcal{O}(h^2)$ e $\mathcal{O}(h^4)$.

4. Si implementi la function `trapezi_adat (fun, a, b, tol)`.

Bibliografia

[1] Alex Druinsky and Sivan Toledo. How Accurate is inv(A)*b? *arXiv e-prints*, page arXiv:1201.6035, Jan 2012.

[2] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, second edition, 2002.

[3] L. Reichel. Newton interpolation at Leja points. *BIT Numer. Math.*, 30:332–346, 1990.