Dispense di Calcolo Numerico
a.a. 2015-16
(versione del 2 dicembre 2016)

1 Algoritmi elementari e loro accelerazione

Premessa

Descriveremo gli algoritmi per il calcolo di alcune quantità dell'analisi numerica e ne analizzeremo il costo computazionale, inteso come il numero di operazioni elementari richiesto per il calcolo. Le operazioni elementari sono la somma algebrica, il prodotto e la divisione tra scalari che indicheremo semplicemente con "ops".

La nostra trattazione sarà necessariamente più grossolana di quella che si fa in informatica, poiché l'interesse del calcolo numerico è più focalizzato sulle questioni analitiche che su quelle informatiche. Tuttavia, ci sembra necessario fissare le notazioni, che useremo spesso.

Sia Ω l'insieme di tutti i dati di un problema, il *costo computazionale assoluto* è una funzione $f : \Omega \to \mathbb{Z}_{\geqslant}$, dove $\mathbb{Z}_{\geqslant}$ è l'insieme dei numeri interi non negativi, che a un insieme di dati assegna il numero di operazioni da effettuare.

Ad esempio, se si vuole considerare il costo computazionale della somma tra due vettori v, w della stessa dimensione, occorrerà definire una funzione f sull'insieme $\Omega = \cup_{n=1}^{\infty}\{(v, w) : v, w \in \mathbb{K}^n\}$ di tutte le coppie di vettori della stessa dimensione e a valori in $\mathbb{Z}_{\geqslant}$. Chiaramente, il valore di questa funzione f sarà n, dove n è la dimensione di v e w, perché sono richieste esattamente n somme di scalari per calcolare $v + w$. Ci si accorge che il costo, in questo caso, dipende solo dalla dimensione dei vettori e non dal loro valore.

In molti casi, il costo dipende da semplici quantità legate ai dati, come per esempio la "dimensione dei dati". È quindi conveniente, dato un algoritmo, stabilire le quantità essenziali che caratterizzano il costo computazionale e definire un costo computazionale relativo a partire da esse. Le quantità essenziali sono definite (abbastanza arbitrariamente) tramite una funzione $\ell : \Omega \to \mathcal{U}$, dove $\mathcal{U}$ è un opportuno insieme e vale che se $\ell(x) = \ell(y)$ allora $f(x) = f(y)$. (Un altro modo di definire ℓ è tramite una relazione di equivalenza su Ω.)

Nel caso della somma tra vettori, si può definire la dimensione dei dati come $\ell : \Omega \to \mathbb{Z}_{\geqslant}$ tale che $\ell(v, w) = n$ e così possiamo definire il costo computazionale relativo come una funzione $\mathcal{C} : \mathbb{Z}_{\geqslant} \to \mathbb{Z}_{\geqslant}$, che varrà $\mathcal{C}(n) = n$. La funzione f sarà data da $f = \mathcal{C} \circ \ell$.

Nel seguito, per semplicità, considereremo esclusivamente il costo computazionale relativo $\mathcal{C} : \mathcal{U} \to \mathbb{Z}_{\geqslant}$, che chiameremo semplicemente *costo computazionale*, definito su un insieme $\mathcal{U}$ delle possibili "dimensioni dei dati", mentre la funzione $\ell : \Omega \to \mathcal{U}$ sarà chiara dal contesto e verrà omessa.

Ad esempio, se lavoriamo con due vettori reali di uguale dimensione, la grandezza dei dati può essere intesa come il numero di componenti dei vettori, e quindi in questo caso $\mathcal{U}$ è l'insieme dei numeri naturali. Se lavoriamo con matrici quadrate di dimensione $n \times n$, la grandezza dei dati è di solito intesa come il lato della matrice, cioè n, e quindi $\mathcal{U} = \mathbb{Z}_{\geqslant}$, mentre se la matrice è rettangolare $m \times n$, la grandezza dei dati è data dalle due dimensioni e $\mathcal{U} = \mathbb{Z}_{\geqslant} \times \mathbb{Z}_{\geqslant}$. Si noti l'ambiguità della notazione, che non generalizza il caso $n \times n$, tuttavia è abbastanza comoda.

Il calcolo del costo computazionale fornisce indicazioni sull'efficienza di un dato algoritmo. Nella maggior parte dei casi si è interessati al comportamento dell'algoritmo quando il costo computazionale $\mathcal{C}(n_1, n_2, \ldots, n_t)$ tende a infinito (si parla di comportamento *asintotico*). Questo accade normalmente quando le dimensioni dei dati tendono a infinito.

Se $\mathcal{C}$ tende a infinito, allora può essere sufficiente la parte principale del suo sviluppo asintotico che spesso è più facile da calcolare. Addirittura, spesso è sufficiente una stima usando le notazioni asintotiche del tipo O.

Il motivo per cui interessa l'andamento a infinito è che spesso lo sviluppo asintotico del costo computazionale fornisce indicazioni sul comportamento dell'algoritmo per valori grandi della dimensione dei dati, che sono quelli che creano maggiori problemi per il calcolo.

Nota. Nella nostra trattazione consideriamo unitario il costo della singola operazione aritmetica a prescindere dalla precisione di macchina usata o dal fatto che $\mathbb{K}$ sia l'insieme dei numeri reali o complessi, anche se in realtà i costi effettivi sono diversi (ma sono multipli l'uno dell'altro).

Valutazione di un polinomio

Un polinomio a coefficienti in $\mathbb{K}$,

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0, \qquad a_i \in \mathbb{K},$$

quando $\mathbb{K} = \mathbb{R}$ o $\mathbb{K} = \mathbb{C}$, può essere identificato con una funzione $p : \mathbb{K} \to \mathbb{K}$. Un problema computazionale è quindi quello di valutare p in un dato $x \in \mathbb{K}$, cioè calcolare $p(x)$.

Un algoritmo semplice che esegue questa operazione, consiste nel calcolare i termini del tipo $a_i x^i$ e sommarli. Se si parte dai termini di grado minore, quando si sta calcolando $a_i x^i$, per $i > 0$, al passo precedente si è calcolato $a_{i-1} x^{i-1}$, e quindi non conviene ricalcolare x^i ogni volta, ma si può ottenere $x_i = x_{i-1} x$ con una sola operazione. L'algoritmo è il seguente

```
1 s=0;
2 p=1;
3 for i=0:n
4    s=s+a(i)*p;
5    p=p*x;
6 end
```

Si osserva che l'algoritmo esegue $3(n+1)$ operazioni aritmetiche.

Con un piccolo trucco è possibile fare di meglio. Consideriamo dapprima il caso $n = 3$ e scriviamo

$$p(x) = ((a_3x + a_2)x + a_1)x + a_0;$$

a questo punto si osserva che sono richieste due operazioni per ogni coppia di parentesi. In generale si può scrivere

$$p(x) = (\cdots((a_nx + a_{n-1})x + a_{n-2})x + \cdots a_1)x + a_0,$$

e tradurre la formula in un algoritmo nel modo seguente

```
1 p=a(n);
2 for i=n-1:-1:0
3    p=p*x+a(i);
4 end
```

Si osserva che questa volta le operazioni richieste sono $2n$, quindi il metodo è più conveniente. Questo algoritmo di valutazione è detto *metodo di Ruffini-Horner*.

Il metodo di Ruffini-Horner può essere usato anche per valutare una funzione razionale del tipo $r(x) = \frac{p(x)}{q(x)}$, dove p è un polinomio di grado m e q è un polinomio di grado n. È sufficiente applicare l'algoritmo a $p(x)$ e $q(x)$ separatamente e poi dividere i risultati, e questo richiede $2(m + n) + 1$ operazioni aritmetiche.

Algoritmi di base in algebra lineare

Dati due vettori $v, w \in \mathbb{K}^n$, che consideriamo vettori colonna, cioè $v, w \in \mathbb{K}^{n\times n}$, i loro trasposti $v^T, w^T \in \mathbb{K}^{1\times n}$ saranno considerati vettori riga. Possiamo definire il prodotto riga per colonna come

$$v^Tw = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} := v_1w_1 + \cdots + v_nw_n = \sum_{i=1}^{n} v_iw_i.$$

Si osservi che $w^Tv = v^Tw$.

L'operazione richiede il calcolo di n prodotti e $n-1$ somme e quindi il costo computazionale è $\mathcal{C}(n) = 2n-1$ o in termini asintotici $\mathcal{C}(n) \approx 2n$. Si può dire che il costo è lineare in n.

Per implementare il calcolo del prodotto scalare, è sufficiente scrivere un ciclo `for` che calcoli la somma $\sum_{i=1}^{n} v_iw_i$ e per questo è sufficiente un accumulatore s che conterrà la somma

```
1 s=0;
2 for i=1:n
3    s=s+v(i)*w(i);
4 end
```

Si noti che il codice esegue $2n$ ops e può essere leggermente migliorato inizializzando l'accumulatore con il primo termine della somma anziché con 0, ottenendo un algoritmo che richiede esattamente le $2n-1$ ops necessarie

```
1 s=v(1)*w(1);
2 for i=2:n
3    s=s+v(i)*w(i);
4 end
```

Il successivo algoritmo è il calcolo del prodotto di una matrice $A \in \mathbb{K}^{n\times n}$ per un vettore $b \in \mathbb{K}^n$, che può essere ottenuto a partire dal prodotti di una riga per una colonna. Si tratta di calcolare ciascuna delle

componenti del vettore $c = Ab \in \mathbb{K}^n$, dove la componente c_i, per $i = 1, \dots, n$, è data dal prodotto della riga i-esima di A, indicata con r_i^T per il vettore b, cioè

$$c_i := r_i^T b = a_{i1}b_1 + a_{i2}b_2 + \cdots + a_{in}b_n = \sum_{j=1}^{n} a_{ij}b_j, \quad i = 1, \dots, n.$$

Il costo è dato quindi da n prodotti scalari, cioè $\mathcal{C}(n) = n(2n-1) = 2n^2 - n \approx 2n^2$. Siccome il costo è dato da un polinomio di grado due si parla di costo quadratico.

Se la matrice $A \in \mathbb{K}^{m\times n}$, essa dovrà essere moltiplicata per un vettore $b \in \mathbb{K}^n$ e darà un vettore $c \in \mathbb{K}^m$. In questo caso la moltiplicazione di una riga di A per il vettore b richiederà $2n-1$ operazioni, e siccome le righe di A sono m, il costo totale dell'algoritmo è $\mathcal{C}(m,n) = 2mn - m \approx 2mn$. Si noti che anche questa volta si è ottenuto un polinomio di grado due, anche se in due variabili.

L'implementazione di questo algoritmo è anch'essa abbastanza semplice, sarà necessario un ciclo `for` per calcolare ciascuna delle componenti di c all'interno del quale è innestato un ciclo `for` che calcola la somma

```
1 for i=1:m
2    s=a(i,1)*b(1);
3    for k=2:n
4       s=s+a(i,k)*b(k);
5    end
6    c(i)=s;
7 end
```

Consideriamo infine il prodotto righe per colonne di due matrici, partendo dal caso quadrato: $A, B \in \mathbb{K}^{n\times n}$. Si vuole calcolare $C = AB$, in questo caso l'elemento c_{ij} di C, per ogni i e j, viene calcolato come il prodotto della riga i-esima di A, indicata con r_i^T, per la colonna j-esima di B, indicata con c_j:

$$c_{ij} := r_i^T c_j = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj} = \sum_{\ell=1}^{n} a_{i\ell}b_{\ell j}, \quad , i, j = 1, \dots, n,$$

dove r_i^T è la riga i-esima di A e c_j è la colonna j-esima di B. Siccome un prodotto tra una riga e una colonna costa $2n-1$ ops e questo va fatto per tutti gli i e j che sono n^2 allora si ha $\mathcal{C}(n) = 2n^3 - n^2 \approx 2n^3 = O(n^3)$ ops.

Se le matrici sono di dimensioni diverse, per esempio se $A \in \mathbb{K}^{m\times k}$ e $B \in \mathbb{K}^{k\times n}$ allora $C = AB \in \mathbb{K}^{m\times n}$ e quindi si devono fare mn prodotti di righe di A per colonne di B, essendo queste ultime lunghe k, il costo totale sarà $\mathcal{C}(n) = (2k-1)mn \approx 2mnk$.

Un altro modo di valutare il costo computazionale consiste nello scrivere dello pseudocodice per l'algoritmo e contare le operazioni elementari all'interno dei cicli a partire dal ciclo più interno. Il costo di un'istruzione del tipo `for k=k1:k2 f(k); end` è data dalla somma per k che va da k_1 a k_2 del costo della valutazione di $f(k)$.

Il prodotto righe per colonne si può scrivere in pseudocodice come

```
1 for i=1:m
2    for j=1:n
3       s=a(i,1)*b(1,j);
4       for t=2:k
5          s=s+a(i,t)*b(t,j);
6       end
7    c(i,j)=s;
8    end
9 end
```

I cicli innestati sono tre, quindi ci si aspetta un costo cubico. Il ciclo più interno, richiede 2 operazioni e va ripetuto $k-2+1$ volte, quindi costa $2k-2$ ops a cui va sommata l'operazione che si esegue durante l'assegnamento di s nella linea 4, per cui il costo della porzione di codice corrispondente alle righe 3-6 è di $2k-1$ operazioni e non dipende da j e da i. Per ottenere il costo complessivo è sufficiente sommare per ogni i e j il valore $2k-1$, ottenendo, come sopra, $\mathcal{C}(m,n,k) = mn(2k-1) = 2mnk - mn$.

Esercizio 1.1. Calcolare il costo computazionale della somma tra due matrici $A, B \in \mathbb{K}^{m\times n}$ e del prodotto αA, dove $\alpha \in \mathbb{K}$.

Soluzione. Semplicemente mn ops per ciascuna delle due operazioni. □

Esercizio 1.2. Dire che dimensione deve avere il vettore y affinché abbia senso il prodotto $y^T A$, con $A \in \mathbb{K}^{m\times n}$ e calcolare il costo computazionale di questa operazione.

Soluzione. Il vettore y deve avere 1 o m componenti. Nel primo caso si ha il prodotto di uno scalare per una matrice che richiede mn ops; nel secondo caso, detto $c = y^T A \in \mathbb{K}^n$, si ha $c_i = \sum_{k=1}^m y_k a_{ki}$, per $i = 1, \dots, n$, quindi il costo è dato da $2nm - n$ ops. □

Esercizio 1.3. Dati due vettori v, w dire quali dimensioni devono avere affinché abbia senso il prodotto vw^T e dire qual è il costo computazionale del calcolo del prodotto. Mostrare che vw^T può essere diverso da wv^T.

Soluzione. Se $v \in \mathbb{K}^{m\times 1}$ e $w \in \mathbb{K}^{n\times 1}$, chiaramente $w^T \in \mathbb{K}^{1\times n}$ e quindi, immaginando v e w^T come matrici, il loro prodotto ha senso e $vw^T \in \mathbb{K}^{m\times n}$ è la matrice

$$\begin{bmatrix} v_1w_1 & v_1w_2 & \cdots & v_1w_n \\ v_2w_2 & v_2w_2 & \cdots & v_2w_n \\ \vdots & \vdots & \vdots & \vdots \\ v_mw_1 & v_mw_2 & \cdots & v_mw_n \end{bmatrix}.$$

Si vede immediatamente che sono richieste esattamente mn moltiplicazioni per la costruzione di vw^T.

Per mostrare che vw^T può essere diverso da wv^T basta osservare che se v e w hanno dimensioni diverse, allora vw^T e wv^T hanno dimensioni diverse e quindi non possono essere uguali. □

Accelerazione degli algoritmi

Gli algoritmi visti sopra sono standard e funzionano su qualsiasi insieme di dati, tuttavia esistono vari modi per accelerarli nei problemi concreti, ottenendo possibilmente risultati computazionali migliori.

Alcuni tra i metodi comunemente usati per accelerare gli algoritmi consistono nel

1. cercare un algoritmo che funzioni meglio di quello standard sul problema o i problemi che stiamo studiando anche se non funziona nel caso generale; in questo caso si parla di algoritmi strutturati;
2. utilizzare le risorse di calcolo (quasi sempre un elaboratore con la sua architettura) per ottenere prestazioni migliori;
3. trovare algoritmi basati su idee nuove e che risolvono i problemi per tutti i dati ma che abbiano un costo computazionale minore rispetto alle implementazioni standard.

Ci soffermeremo sul punto 1 nel seguito del capitolo, dando una menzione del punto 2 nella prossima sezione. Per quanto riguarda il punto 3 ci limiteremo a un breve cenno su come si è tentato di accelerare il prodotto righe per colonne.

Il prodotto righe per colonne non è l'algoritmo con la minore complessità per il calcolo del prodotto tra due matrici quadrate: nel 1969, V. Strassen ha mostrato che è possibile calcolare il prodotto tra due matrici con un costo asintotico pari a $O(n^{\log_2 7})$ ops. Non è ancora chiaro quale sia la complessità di questo problema, l'unica cosa che si è riusciti a ottenere è di abbassare il costo. L'algoritmo più veloce oggi richiede $O(n^{2.3727})$ ops. L'utilità pratica di questo tipo di algoritmi è abbastanza limitata poiché la O nasconde costanti a volte molto grosse, inoltre non è frequente eseguire moltiplicazioni di matrici molto grandi, a meno che queste matrici non siano strutturate. Nel resto della trattazione considereremo solo il prodotto righe per colonne.

Algoritmi paralleli

Il computo del numero di operazioni richieste da un algoritmo è normalmente una misura del tempo di esecuzione, nell'ipotesi che ogni operazione elementare, se eseguita su calcolatore, richieda un tempo finito, grosso modo costante.

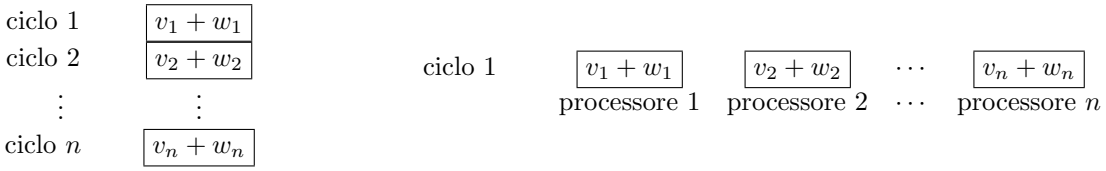
Le moderne architetture permettono di effettuare operazioni in parallelo su più processori e quindi effettuare più operazioni contemporaneamente. Se si riescono ad eseguire n operazioni su n processori, le operazioni saranno effettuate in una sola unità di tempo, che chiameremo *ciclo*.

Nella pratica occorre tenere conto di vari problemi, tra cui la comunicazione (tra i processori) e l'accesso alla memoria, tuttavia in molti casi sfruttare i processori in parallelo accorcia i tempi effettivi di esecuzione.

Nel seguito considereremo un modello semplificato in cui i calcoli vengono effettuati in sequenza dai processori coinvolti, un'operazione per ogni ciclo trascurando i problemi di comunicazione. Inoltre, per semplicità, tratteremo solo il caso in cui si disponga di un numero sufficientemente alto di processori.

Se si vogliono sommare due vettori $v, w \in \mathbb{R}^n$, occorre calcolare le n somme $v_i + w_i$ per $i = 1, \ldots, n$. Si osserva che per calcolare $v_1 + w_1$ non serve il risultato di $v_2 + w_2$, quindi queste due operazioni possono essere eseguite da due processori diverse nello stesso istante, anziché su un solo processore in due istanti consecutivi. Più in generale, poiché nessuna delle operazioni richiede il risultato delle altre, le n operazioni $v_i + w_i$ possono essere effettuate in un solo ciclo di calcolo su n processori in parallelo.

La seguente rappresentazione grafica mostra a sinistra un'esecuzione su un solo processore e a destra l'esecuzione su n processori in parallelo, evidenziando il fatto che le stesse operazioni vengono eseguite in un minor numero di cicli di calcolo.



Un'implementazione dell'algoritmo parallelo è data da:

```
1 parallel for i=1:n
2   u(i)=v(i)+w(i)
3 end
```

Si noti l'uso dell'istruzione `parallel for` che indica che ogni ciclo deve essere eseguito, se possibile, in parallelo. Useremo anche l'istruzione `parallel` per racchiudere un blocco di comandi che devono essere eseguiti, se possibile, in parallelo.

Se si devono sommare tre numeri $x_0, x_1, x_2 \in \mathbb{R}$, il numero di operazioni da effettuare è 2. Si possono utilizzare tre algoritmi, per esempio, si può porre $y = x_0 + x_1$ e $s = y + x_2$ oppure si può porre $z = x_1 + x_2$ e $s = x_0 + z$ e infine si può porre $t = x_0 + x_2$ e $s = t + x_1$. Si noti che, poiché in aritmetica finita la proprietà associativa non vale, sono tre algoritmi diversi.

In tutti e tre i casi la seconda operazione richiede il risultato della prima e quindi, anche disponendo di due processori, le due operazioni non possono essere eseguite in parallelo. Il calcolo richiede sempre 2 cicli su qualsiasi numero di processori.

Consideriamo ora la somma di 4 numeri reali $s = x_0 + x_1 + x_2 + x_3$. Il calcolo richiede 3 operazioni, tuttavia se si dispone di almeno 2 processori è possibile sommare durante il primo ciclo $y_0 = x_0 + x_1$ e $y_2 = x_2 + x_3$ in parallelo poiché nessuna delle due operazioni richiede il risultato dell'altra. Al secondo ciclo si può calcolare $s = y_0 + y_2$. L'implementazione di questo algoritmo è la seguente

```
1 parallel
2   s=x(0)+x(1);
3   t=x(2)+x(3);
4 end
5 s=s+t;
```

da cui, poiché il blocco parallelo viene eseguito in un ciclo, si deduce facilmente che il numero di cicli richiesti è 2. In questo modo abbiamo risparmiato 1 ciclo di calcolo rispetto alla somma sequenziale che consiste nel calcolare $z_1 = x_0 + x_1$, poi $z_2 = z_1 + x_2$ e infine $s = z_2 + x_3$, senza possibilità di parallelizzare.

Nel caso di 8 numeri si può procedere come in figura calcolando la somma in soli $3 = \log_2 8$ cicli anziché 7 che è il numero di operazioni necessarie.

$$\underbrace{\underbrace{x_0 + x_1}_{v_0^{(1)}} + \underbrace{x_2 + x_3}_{v_1^{(1)}}}_{v_0^{(2)}} + \underbrace{\underbrace{x_4 + x_5}_{v_2^{(1)}} + \underbrace{x_6 + x_7}_{v_3^{(1)}}}_{v_1^{(2)}}$$

Questo modo di procedere può essere generalizzato. Consideriamo il problema del calcolo della somma di n numeri reali, $x_i \in \mathbb{R}$ per $i = 0, \ldots, n-1$, cioè $s = \sum_{i=0}^{n-1} x_i$. Si può assumere che $n = 2^k$, altrimenti si possono aggiungere zeri quanto basta per arrivare a 2^k numeri.

L'algoritmo sequenziale per il calcolo della somma in pseudocodice ha la forma

```
1 s=0;
2 for i=0:2^k-1
3   s=s+x(i);
4 end
```

Posto $v_i^{(0)} = x_i$, per $i = 0, \ldots, n-1$, l'algoritmo parallelo ha invece questa forma

$$v_i^{(p)} = v_{2i}^{(p-1)} + v_{2i+1}^{(p-1)}, \qquad i = 0, \ldots, n/2^p - 1, \qquad p = 1, \ldots, k.$$

Un'implementazione in pseudocodice con sovrascrittura del vettore v è

```
1 for p=1:k
2    for i=0:n/2^p-1
3       v(i)=v(2*i)+v(2*i+1);
4    end
5 end
```

e la somma sarà contenuta in `v(0)`.

Si vede immediatamente che tutte le operazioni nel ciclo `for` interno sono indipendenti e si possono eseguire in un ciclo di calcolo e quindi in totale sono richiesti k cicli di calcolo, se si hanno a disposizione 2^{k-1} processori. Se n non è potenza di 2, il più piccolo k tale che $n \leqslant 2^k$ è $\lceil \log_2 n \rceil$. Si può concludere che con un numero di processori maggiore di $n/2$ è possibile effettuare la somma di n numeri in non più di $\lceil \log_2 n \rceil$ cicli di calcolo. L'implementazione è:

```
1 for p=1:k
2    n=n/2;
2    parallel for i=0:n-1
3       v(i)=v(2*i)+v(2*i+1);
4    end
5 end
```

Si noti che il parallel for interno va da 1 a $n/2^p - 1$, ma per evitare di calcolare ogni volta la potenza, viene utilizzata la variabile n che si dimezza a ogni passo.

Ora consideriamo gli algoritmi elementari di algebra lineare. Il prodotto scalare si può calcolare in $1+\lceil \log_2 n \rceil$ cicli su n processori, effettuando prima i prodotti e poi sommando gli n numeri così ottenuti con la somma parallela.

Il prodotto matrice-vettore richiede n prodotti scalari che si possono eseguire in parallelo, quindi su n^2 processori è possibile calcolare il prodotto matrice-vettore in $1 + \lceil \log_2 n \rceil$ cicli.

Il prodotto tra due matrici $n \times n$ righe per colonne, richiede n^2 prodotti scalari, tutti indipendenti, quindi su n^3 processori è possibile calcolare il prodotto matrice-matrice in $1 + \lceil \log_2 n \rceil$ cicli!

Anche se a prima vista non sembra, è possibile parallelizzare anche i due algoritmi per la valutazione di un polinomio visti sopra. Infatti il passo dell'algoritmo standard di valutazione può essere visto come il prodotto di una matrice 2×2 per un vettore

$$\begin{bmatrix} s \\ p \end{bmatrix} := \begin{bmatrix} 1 & a_i \\ 0 & x \end{bmatrix} \begin{bmatrix} s \\ p \end{bmatrix},$$

quindi l'intero procedimento può essere visto come il prodotto di $n+1$ matrici 2×2 per un vettore

$$\begin{bmatrix} 1 & a_n \\ 0 & x \end{bmatrix} \begin{bmatrix} 1 & a_{n-1} \\ 0 & x \end{bmatrix} \cdots \begin{bmatrix} 1 & a_2 \\ 0 & x \end{bmatrix} \begin{bmatrix} 1 & a_1 \\ 0 & x \end{bmatrix} \begin{bmatrix} 1 & a_0 \\ 0 & x \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Nell'algoritmo standard il prodotto sopra viene eseguito con l'ordine dato dalle parentesi nel seguente modo

$$\begin{bmatrix} 1 & a_n \\ 0 & x \end{bmatrix} \left(\begin{bmatrix} 1 & a_{n-1} \\ 0 & x \end{bmatrix} \cdots \left(\begin{bmatrix} 1 & a_2 \\ 0 & x \end{bmatrix} \left(\begin{bmatrix} 1 & a_1 \\ 0 & x \end{bmatrix} \left(\begin{bmatrix} 1 & a_0 \\ 0 & x \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right) \right) \right) \cdots \right).$$

Usando la proprietà associativa è possibile calcolare la stessa quantità, moltiplicando prima tra loro tutte le matrici e poi moltiplicando il risultato per il vettore $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$.

$$\left(\begin{bmatrix} 1 & a_n \\ 0 & x \end{bmatrix} \begin{bmatrix} 1 & a_{n-1} \\ 0 & x \end{bmatrix} \cdots \begin{bmatrix} 1 & a_2 \\ 0 & x \end{bmatrix} \begin{bmatrix} 1 & a_1 \\ 0 & x \end{bmatrix} \begin{bmatrix} 1 & a_0 \\ 0 & x \end{bmatrix} \right) \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Non sembra un grosso affare, visto che moltiplicare $n+1$ matrici 2×2 e poi moltiplicare il risultato per un vettore è più costoso che moltiplicarle consecutivamente per un vettore, ma facendo alcune osservazioni si scopre che può essere conveniente.

Per prima cosa, si osserva che tutte le matrici sono del tipo $\left[\begin{smallmatrix} 1 & \alpha \\ 0 & \beta \end{smallmatrix}\right]$, con $\alpha, \beta \in \mathbb{K}$, e che moltiplicando due matrici di questo tipo si ottiene un'altra matrice di questo tipo, infatti

$$\begin{bmatrix} 1 & \alpha \\ 0 & \beta \end{bmatrix} \begin{bmatrix} 1 & \gamma \\ 0 & \delta \end{bmatrix} = \begin{bmatrix} 1 & \gamma + \alpha\delta \\ 0 & \beta\delta \end{bmatrix}.$$

Avendo a disposizione due processori, un tale prodotto può essere eseguito in due cicli eseguendo prima i prodotti $\alpha\delta$ e $\beta\gamma$ contemporaneamente e, successivamente, la somma $\gamma + \alpha\delta$.

Come seconda osservazione, ci accorgiamo che lo stesso algoritmo usato per la somma parallela, può essere usato per moltiplicare 2^k matrici in $k\ell$ cicli, dove ℓ è il numero di cicli necessario per moltiplicare due matrici. Quindi, siccome per moltiplicare due matrici di quel tipo sono richiesti 2 cicli e la struttura si preserva, se $n = 2^k - 1$ allora è possibile calcolare il prodotto di tutte le matrici con $2k$ cicli. Se $n + 1$ non è potenza di due, si possono anteporre al prodotto matrici del tipo $\left[\begin{smallmatrix} 1 & 0 \\ 0 & x \end{smallmatrix}\right]$ in numero sufficiente a raggiungere 2^k matrici, e in questo caso il costo è di $2\lceil \log_2(n+1) \rceil$ cicli.

Infine, detta $M = \left[\begin{smallmatrix} 1 & s \\ 0 & t \end{smallmatrix}\right]$, per trovare il valore del polinomio, occorre moltiplicare M per il vettore $\left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right]$, e selezionare la prima componente del risultato. Ci si accorge facilmente che è proprio s e quindi non è necessario calcolare tale prodotto, ma la valutazione del polinomio si può leggere in m_{12}.

In definitiva, si riesce a valutare il polinomio in $2(1 + \lceil \log_2 n \rceil)$ cicli di calcolo utilizzando meno di $2n$ processori.

Procedendo in modo analogo per l'algoritmo di Ruffini-Horner si scopre che in questo caso il passo è dato dal prodotto

$$\begin{bmatrix} s \\ 1 \end{bmatrix} := \begin{bmatrix} x & a_i \\ 0 & 1 \end{bmatrix} \begin{bmatrix} s \\ 1 \end{bmatrix},$$

quindi l'intero procedimento può essere visto come il prodotto di $n + 1$ matrici 2×2 per un vettore

$$\begin{bmatrix} x & a_0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x & a_1 \\ 0 & 1 \end{bmatrix} \cdots \begin{bmatrix} x & a_{n-1} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x & a_n \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Ancora una volta, si osserva che tutte le matrici hanno una struttura molto particolare, infatti sono del tipo $\left[\begin{smallmatrix} \alpha & \beta \\ 0 & 1 \end{smallmatrix}\right]$, con $\alpha, \beta \in \mathbb{K}$, e che moltiplicando due matrici di questo tipo si ottiene un'altra matrice di questo tipo, infatti

$$\begin{bmatrix} \alpha & \beta \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \gamma & \delta \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \alpha\gamma & \alpha\delta + \beta \\ 0 & 1 \end{bmatrix}.$$

Avendo a disposizione più processori è possibile calcolare prima il prodotto tra le matrici in modo parallelo e poi moltiplicare il risultato per il vettore $\left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right]$ (quest'ultima operazione non è necessaria, in quanto, come prima, il risultato si trova nell'elemento m_{12} del prodotto delle matrici). Ancora una volta, si riesce a valutare il polinomio in $2\lceil \log_2(n+1) \rceil$ cicli di calcolo.

In ambito parallelo, quindi, non sembra che ci sia un vantaggio ad utilizzare un metodo ottenuto dall'algoritmo di Ruffini-Horner, invece che da quello tradizionale.

Strutture di matrici

Le matrici che provengono dalle applicazioni hanno, nella maggior parte dei casi, una qualche struttura più o meno facilmente identificabile. Possiamo definire *struttura di matrici* un qualsiasi sottoinsieme delle matrici, tuttavia le strutture interessanti sono quelle che ci permettono di progettare algoritmi *ad hoc* e che abbiano proprietà computazionali migliori rispetto a quelle date dall'algoritmo standard (per esempio un costo computazionale inferiore o maggiore stabilità numerica). In questo caso parleremo di *algoritmo strutturato*.

La struttura più facilmente identificabile è quella in cui un certo numero di elementi della matrice sono nulli. Ad esempio, una matrice A è detta diagonale se $a_{ij} = 0$ quando $i \neq j$, essa ha un numero di elementi non nulli che è al massimo n, inoltre le matrici diagonali formano una sottoalgebra delle matrici quadrate (cfr. Esercizio 1.4). Nella pratica, è spesso sufficiente rappresentare le matrici strutturate dando una loro rappresentazione grafica utilizzando i "puntini", oppure indicando solo gli elementi eventualmente non nulli. Si può anche indicare con un asterisco l'elemento che può essere non nullo e identificare in questo modo la struttura. Una matrice D diagonale si potrà rappresentare come

$$D = \begin{bmatrix} d_{11} & 0 & \cdots & 0 \\ 0 & d_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & d_{nn} \end{bmatrix} = \begin{bmatrix} d_{11} & & & \\ & d_{22} & & \\ & & \ddots & \\ & & & d_{nn} \end{bmatrix} = \begin{bmatrix} * & & & \\ & * & & \\ & & \ddots & \\ & & & * \end{bmatrix}.$$

Occorre prestare attenzione al fatto che nella definizione non è richiesto che gli elementi sulla diagonale siano non nulli.

Altri esempi di strutture sono le matrici triangolari superiori, cioè le matrici $T \in \mathbb{K}^{n\times n}$ tali che $t_{ij} = 0$ se $i > j$ e che si rappresentano come

$$T = \begin{bmatrix} t_{11} & t_{12} & \cdots & t_{1n} \\ 0 & t_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & t_{nn} \end{bmatrix} = \begin{bmatrix} * & * & \cdots & * \\ & * & \ddots & \vdots \\ & & \ddots & \vdots \\ & & & * \end{bmatrix}.$$

In modo analogo si possono definire le matrici triangolari inferiori (basta che $t_{ij} = 0$ per $i < j$).

Un altro esempio di struttura è dato dalle matrici tridiagonali, cioè le matrici $A \in \mathbb{K}^{n\times n}$ tali che $a_{ij} = 0$ se $|i - j| > 1$. Esse si rappresentano come

$$A = \begin{bmatrix} a_{11} & a_{12} & 0 & \cdots & 0 \\ a_{21} & a_{22} & a_{23} & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & a_{n-1,n} \\ 0 & \cdots & 0 & a_{n,n-1} & a_{nn} \end{bmatrix} = \begin{bmatrix} * & * & & & \\ * & * & * & & \\ & \ddots & \ddots & \ddots & \\ & & * & * & * \\ & & & * & * \end{bmatrix}.$$

L'insieme delle matrici tridiagonali formano uno spazio vettoriale di dimensione $3n-2$, tuttavia esse non formano un'algebra poiché il prodotto tra due matrici tridiagonali non è tridiagonale. Il termine tridiagonale è riferito al fatto che la matrice è nulla al di fuori di tre diagonali che sono la diagonale principale e la sua sopra- e sotto-diagonale. In modo analogo si può definire una matrice pentadiagonale.

In generale si può definire una matrice A *a banda di ampiezza* k come una matrice tale che $a_{ij} = 0$ per $|i - j| > k$ e quindi la matrice tridiagonale è a banda di ampiezza 1.

Se una struttura può essere definita per ogni m, n dove m e n sono le dimensioni della matrice, allora ha senso chiedersi qual è il massimo numero di elementi non nulli asintoticamente. Sia $S(m, n)$ il massimo numero di elementi non nulli di una matrice strutturata, se $S(m, n) \ll mn$ (nel senso che $\lim_{m,n\to\infty} S(m, n)/mn = 0$), allora si dice che la matrice è *sparsa* e la struttura si dice struttura di sparsità.

Nella pratica, si usa la definizione di matrice sparsa anche per matrici di dimensioni finite in cui il numero di elementi non nulli sia molto minore (in un qualche senso) delle dimensioni della matrice.

Per concludere, vorremmo sottolineare il fatto che una struttura utile dal punto di vista computazionale non deve essere per forza legata alla presenza di elementi nulli. Per esempio si può considerare la struttura

$$\mathcal{S}_n = \{A \in \mathbb{C}^{n\times n} \,:\, A = vw^T, v, w \in \mathbb{C}^n\}$$

in cui le matrici possono anche non avere alcun elemento nullo.

Esercizio 1.4. Provare che le matrici diagonali sono una sottoalgebra delle matrici $n \times n$ (con elementi in un campo $\mathbb{K}$).

Soluzione. Sia $\mathcal{D}_n$ l'insieme delle matrici diagonali di dimensione n. Occorre provare

1. $0 \in \mathcal{D}_n$ (dove 0 è intesa come la matrice nulla);
2. se $A, B \in \mathcal{D}_n$ allora $A + B \in \mathcal{D}_n$;
3. se $A \in \mathcal{D}_n$ e $\alpha \in \mathbb{K}$, allora $\alpha A \in \mathcal{D}_n$;
4. se $A, B \in \mathcal{D}_n$ allora $AB \in \mathcal{D}_n$.

Le prime tre proprietà ci dicono che $\mathcal{D}_n$ è un sottospazio vettoriale. Per mostrare la proprietà 1 è sufficiente osservare che la matrice nulla è una matrice diagonale. Per mostrare le proprietà 2 e 3 osserviamo che, se $\alpha \in \mathbb{K}$ e

$$A = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & a_{nn} \end{bmatrix}, \qquad B = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & a_{nn} \end{bmatrix}$$

allora

$$A + B = \begin{bmatrix} a_{11} + b_{11} & 0 & \cdots & 0 \\ 0 & a_{22} + b_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & a_{nn} + b_{nn} \end{bmatrix} \in \mathcal{D}_n, \qquad \alpha A = \begin{bmatrix} \alpha a_{11} & 0 & \cdots & 0 \\ 0 & \alpha a_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \alpha a_{nn} \end{bmatrix} \in \mathcal{D}_n.$$

In modo più formale, si può dire che se $a_{ij} = b_{ij} = 0$ per $i \neq j$ allora $a_{ij} + b_{ij} = 0$ e $\alpha a_{ij} = 0$ per $i \neq j$ e questo mostra che la somma e il prodotto per scalare rimangono diagonali. Infine, per la proprietà 4, date $A, B \in \mathcal{D}_n$ e posto $C = AB$, si osserva che $c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj}$ e la somma si riduce al solo addendo $a_{ii}b_{ij}$ (perché per $k \neq i$ il prodotto $a_{ik}b_{kj} = 0$). Ma $a_{ii}b_{ij} = 0$ se $i \neq 0$ (perché lo è b_{ij}) e quindi C è diagonale. In alternativa, si può osservare che

$$AB = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & a_{nn} \end{bmatrix} \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & a_{nn} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & 0 & \cdots & 0 \\ 0 & a_{22}b_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & a_{nn}b_{nn} \end{bmatrix} \in \mathcal{D}_n.$$

□

Esercizio 1.5. Dimostrare che la dimensione del sottospazio $\mathcal{D}_n$ delle matrici diagonali $n \times n$ è n.

Soluzione. La dimensione di un sottospazio vettoriale è il numero di elementi di una sua base. Occorre quindi trovare n matrici diagonali che sono linearmente indipendenti e generano $\mathcal{D}_n$. Tale insieme può essere, per esempio

$$D_1 = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 0 \end{bmatrix}, \qquad D_2 = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 0 \end{bmatrix}, \qquad \cdots, \qquad D_n = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{bmatrix}.$$

Si osserva che una combinazione lineare di esse è nulla, cioè

$$\alpha_1 D_1 + \alpha_2 D_2 + \cdots + \alpha_n D_n = \begin{bmatrix} \alpha_1 & 0 & \cdots & 0 \\ 0 & \alpha_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \alpha_n \end{bmatrix} = 0$$

se e solo se $\alpha_1 = \alpha_2 = \cdots = \alpha_n = 0$ e quindi sono linearmente indipendenti. Inoltre generano $\mathcal{D}_n$, infatti data $A \in \mathcal{D}_n$ si ha

$$A = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & a_{nn} \end{bmatrix} = a_{11} D_1 + a_{22} D_2 + \cdots + a_{nn} D_n.$$

□

Esercizio 1.6. Provare che le matrici triangolari superiori o inferiori sono una sottoalgebra delle matrici quadrate e trovarne la dimensione (che è il massimo numero di elementi non nulli).

Soluzione. È abbastanza evidente che la somma di due matrici triangolari è ancora triangolare, così come il prodotto di una matrice triangolare per uno scalare. Per quanto riguarda il prodotto: siano $S, T \in \mathbb{K}^{n\times n}$ due matrici triangolari superiori e sia $C = ST$, si osserva che per $i > j$,

$$c_{ij} = \sum_{k=1}^{n} s_{ik}c_{kj} = \sum_{k=1}^{i-1} s_{ik}c_{kj} + \sum_{k=i}^{n} s_{ik}c_{kj} = \sum_{k=1}^{i-1} 0 \cdot c_{kj} + \sum_{k=i}^{n} s_{ik} \cdot 0 = 0.$$

La prima delle due somme è nulla poiché $s_{ik} = 0$ per $i > k$, in particolare per $k = 1, \ldots, i-1$, mentre la seconda è nulla poiché $c_{kj} = 0$ per $k > j$, in particolare per $k = i, \ldots, n$. □

Esercizio 1.7. Mostrare che il prodotto di due matrici tridiagonali non è tridiagonale, dire se esso ha qualche struttura.

Soluzione. Il fatto che non sia tridiagonale si può osservare prendendo una matrice tridiagonale generica 3×3 con elementi positivi sulle tre diagonali e calcolare il quadrato.

Si osserva che il prodotto di due matrici tridiagonali è pentadiagonale. Questo fatto si può dimostrare considerando due matrici tridiagonali $A, B \in \mathbb{K}^{n\times n}$ (con $n > 3$) e il loro prodotto $C = AB$. Per fissare le idee consideriamo gli elementi di indice $i - j > 2$

$$c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj} = \sum_{k=1}^{j+1} a_{ik}b_{kj} + \sum_{k=j+2}^{n} a_{ik}b_{kj} = \sum_{k=1}^{j+1} 0 \cdot b_{kj} + \sum_{k=j+2}^{n} a_{ik} \cdot 0 = 0,$$

la prima somma è nulla poiché $a_{ik} = 0$ per $k < i - 1$ in particolare per $k = 1, \ldots, j+1$, mentre la seconda somma è nulla poiché $b_{kj} = 0$ per $k > j + 1$ dalla definizione di matrice tridiagonale. In modo del tutto analogo si dimostra che $c_{ij} = 0$ anche quando $i - j < -2$ e quindi alla fine $c_{ij} = 0$ per $|i - j| > 2$ e la matrice è pentadiagonale. □

Esercizio 1.8. Dire se le matrici a banda di ampiezza t sono sparse.

Soluzione. Sono sparse, in quanto per n che tende a infinito gli elementi non nulli sono al più

$$n + \sum_{k=1}^{t} 2(n-k) = n + 2\left(tn - \frac{t(t+1)}{2}\right) = (2t+1)n - t^2 - t \approx (2t+1)n \ll n^2.$$

□

Esercizio 1.9. Sia data la struttura $\mathcal{S}_n = \{A \in \mathbb{C}^{n\times n} \,:\, A = vw^T, v, w \in \mathbb{C}^n\}$, si mostri che $\mathcal{S}_n$ è l'insieme delle matrici di rango al più 1 e che anche se è chiuso rispetto al prodotto, non è un'algebra.

Soluzione. Le matrici in $\mathcal{S}_n$ hanno rango minore o uguale a 1, infatti, scrivendo A per colonne come $A = [w_1 v | w_2 v | \cdots | w_n v]$, si osserva che tutte le colonne sono multiple dello stesso vettore e quindi A non può avere rango maggiore di 1. Viceversa, se una matrice A ha rango minore o uguale a 1, tutte le colonne sono multiple di un unico vettore per cui $A = vw^T$.

Siano $A = v_1 w_1^T$ e $B = v_2 w_2^T$ allora $AB = (v_1 w_1^T)(v_2 w_2^T) = v_1 (w_1^T v_2) w_2^T$, ma $\alpha = w_1^T v_2$ è uno scalare e quindi si può scrivere $AB = (\alpha v_1) w_2^T$ da cui $AB \in \mathcal{S}_n$. La struttura $\mathcal{S}_n$ non è un'algebra per $n \geqslant 2$, in quanto non è uno spazio vettoriale, poiché la somma di due matrici di $\mathcal{S}_n$ non è necessariamente una matrice di $\mathcal{S}_n$. Questo fatto si può vedere con un controesempio: siano

$$A = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix},$$

si osserva che $A, B \in \mathcal{S}_2$, ma $A + B = I$ che ha rango 2 e quindi $A + B \notin \mathcal{S}_2$. Analoghi controesempi si possono trovare per ogni $n \geqslant 2$, mentre $\mathcal{S}_1$ è un'algebra poiché $\mathcal{S}_1 = \mathbb{R}$.

In alternativa, si può osservare che $I \notin \mathcal{S}_n$ usando un metodo di forza bruta, cioè cercando soluzioni dell'equazione

$$I = \begin{bmatrix} x \\ y \end{bmatrix} \begin{bmatrix} z & t \end{bmatrix} \iff \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} xz & xt \\ yz & yt \end{bmatrix} \iff \begin{cases} xz = 1, \\ xt = 0, \\ yz = 0, \\ yt = 1 \end{cases}$$

dove le incognite sono x, y, z, t. Affinché l'identità sia vera, occorre che $xt = 0$ e quindi si danno due casi: 1) $x = 0$ e in questo caso $xz = 1$ non può essere verificata; 2) $t = 0$ e in questo caso $yt = 1$ non può essere verificata. Quindi I non si può scrivere come prodotto di due vettori. □

Algoritmi strutturati

Diamo alcuni esempi di algoritmi strutturati, facendo vedere come varia il costo computazionale.

Come primo esempio cerchiamo un algoritmo strutturato per il calcolo del prodotto di una matrice diagonale $A \in \mathbb{K}^{n\times n}$ per un vettore $b \in \mathbb{K}^n$. Detto $c = Ab$, per ogni $i = 1, \ldots, n$, si ha

$$c_i = a_{i1}b_1 + \cdots + a_{i,i-1}b_{i-1} + a_{ii}b_i + a_{i,i+1}b_{i+1} + \cdots + a_{in}b_n = a_{ii}b_i.$$

Per ottenere b_i è sufficiente calcolare il prodotto $a_{ii}b_i$ e quindi eseguire una sola operazione, anziché $2n - 1$. Quindi il costo di questo algoritmo strutturato è $\mathcal{C}(n) = n \ll 2n^2 - n$.

Consideriamo ora il caso in cui la matrice A sia tridiagonale. Ci accorgiamo che per $n > 2$ è possibile calcolare c nel seguente modo

$$\begin{cases} c_1 = a_{11}b_1 + a_{12}b_2, \\ c_i = a_{i,i-1}b_{i-1} + a_{ii}b_i + a_{i,i+1}b_{i+1}, \quad i = 2, \ldots, n-1, \\ c_n = a_{n,n-1}b_{n-1} + a_{nn}b_n, \end{cases}$$

e quindi abbiamo un algoritmo strutturato che esegue 3 operazioni per calcolare c_1 e c_n e 5 operazioni per il calcolo dei restanti $n - 2$ valori di c_i per un costo totale pari a

$$\mathcal{C}(n) = 5(n-2) + 3 + 3 = 5n - 4 \approx 5n.$$

La formula è valida per $n > 2$.

Un altro modo per calcolare il costo computazionale consiste nello scrivere l'algoritmo in pseudocodice

```
1 c(1)=a(1,1)*b(1)+a(1,2)*b(2);
2 for i=2:n-1
3    c(i)=a(i,i-1)*b(i-1)+a(i,i)*b(i)+a(i,i+1)*b(i+1);
4 end
5 c(n)=a(n,n-1)*b(n-1)+a(n,n)*b(n);
```

da cui si vede chiaramente che il costo di ogni istruzione del ciclo è 5 e quindi il costo del ciclo è $5(n-1-2+1) = 5(n-2)$, le altre due istruzioni costano 6 ops e alla fine si giunge allo stesso risultato.

Come ulteriore esempio consideriamo il prodotto di una matrice triangolare superiore $T \in \mathbb{K}^{n\times n}$ per un vettore $x \in \mathbb{K}^n$, anche in questo caso si può semplificare la formula che fornisce c_i tale che $c = Tb$.

Ci si accorge che, siccome $a_{ij} = 0$ per per $i > j$, si ha

$$c_i = a_{i1}b_1 + \cdots + a_{in}b_n = a_{ii}b_i + \cdots + a_{in}x_n,$$

cioè

$$c_i = \sum_{k=1}^{n} a_{ik}b_k = \sum_{k=i}^{n} a_{ij}b_k.$$

In altre parole, la somma che definisce c_i può partire da i senza che cambi il risultato e quindi c_i è la somma di $n - i + 1$ prodotti. Il costo per il calcolo di c_i è quindi di $2(n - i) + 1$ operazioni. Il costo totale si ottiene sommando i costi necessari per il calcolo di ciascun c_i, e insomma alla fine

$$\mathcal{C}(n) = \sum_{i=1}^{n} 2(n-i) + 1 = 2\frac{n(n-1)}{2} + n = n^2.$$

Scrivendo l'algoritmo in pseudocodice

```
1 for i=1:n
2    c(i)=a(i,i)*b(i);
3    for k=i+1:n
4       c(i)=c(i)+a(i,k)*b(k);
5    end
6 end
```

si ottiene lo stesso risultato.

Per calcolare i costi computazionali possono essere utili le seguenti formule asintotiche

$$\sum_{k=1}^{n} k \approx \sum_{k=1}^{n}(n-k) \approx \frac{1}{2}n^2, \quad \sum_{k=1}^{n} k^2 \approx \frac{1}{3}n^3.$$

Naturalmente, il carattere asintotico delle precedenti formule, fa sì che esse continuino ad essere valide se, anziché sommare da 1 a n, si somma da un indice i_1 indipendente da n fino a $n - i_2$, dove i_2 è indipendente da n, ad esempio

$$\sum_{k=5}^{n+3} k^2 \approx \sum_{k=1000}^{n-5} (n-k)^2 \approx \frac{1}{3}n^3.$$

Nota. Lo sviluppo asintotico di $\sum_{k=1}^{n} k^\alpha$ con α intero positivo è dato dalla formula

$$\sum_{k=1}^{n} k^\alpha = \frac{k^{\alpha+1}}{\alpha+1},$$

la dimostrazione si può ottenere a partire dalla diseguaglianza $\int_0^n x^\alpha dx < \sum_{k=1}^{n} k^\alpha < \int_0^{n+1} x^\alpha dx$ che si verifica graficamente. Calcolando gli integrali si ottiene

$$\frac{n^{\alpha+1}}{\alpha+1} < \sum_{k=1}^{n} k^\alpha < \frac{(n+1)^{\alpha+1}}{\alpha+1} \approx \frac{n^{\alpha+1}}{\alpha+1},$$

da cui segue tutto.

Esercizio 1.10. Trovare un algoritmo strutturato per il calcolo del prodotto tra due matrici triangolari superiori e valutarne il costo computazionale.

Soluzione. Siano $A, B \in \mathbb{K}^{n\times n}$ e $C = AB$. Il prodotto di matrici triangolari è a sua volta triangolare, quindi basta calcolare c_{ij} per $i \leqslant j$,

$$c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj} = \sum_{k=1}^{i-1} a_{ik}b_{kj} + \sum_{k=i}^{j} a_{ik}b_{kj} + \sum_{k=j+1}^{n} a_{ik}b_{kj} = \sum_{k=i}^{j} a_{ik}b_{kj},$$

dove delle tre somme sopra la prima è nulla perché $a_{ik} = 0$ per $i > k$ e quindi per $k = 1, \ldots, j-1$, la terza è nulla perché $b_{kj} = 0$ per $k > j$ e quindi per $k = i+1, \ldots, n$.

Ora, per ogni $i \leqslant j$ si richiedono $2(j-i)+1$ ops e alla fine il costo è

$$\mathcal{C}(n) = \sum_{j=1}^{n} \sum_{i=1}^{j} 2(j-i)+1 \approx \sum_{j=1}^{n} 2(j^2 - \frac{j^2}{2}) = \sum_{j=1}^{n} j^2 \approx \frac{n^3}{3}.$$

L'esercizio può essere risolto anche in maniera meno formale facendo vedere con un disegno che per il calcolo di c_{ij} la sommatoria da calcolare ha indici estremi i e j. □

Prodotto di una matrice sparsa per un vettore

Finora abbiamo visto strutture più o meno facilmente identificabili dove l'implementazione di un algoritmo strutturato risultava semplice. Tuttavia, in altri casi è più difficile implementare un algoritmo strutturato.

Consideriamo il calcolo del prodotto di una matrice $A \in \mathbb{K}^{n \times n}$ che ha k elementi non nulli per un vettore $v \in \mathbb{K}^n$. Come abbiamo visto, se k è "piccolo" rispetto a n si dice comunemente che la matrice è sparsa.

Per prima cosa, ricordiamo l'implementazione per il calcolo del prodotto $c = Av$

```
1 for i=1:n
2   for j=1:n
3     c(i)=c(i)+a(i,j)*v(j);
4   end
5 end
```

dove si è assunto che `c` sia stato inizializzato come vettore nullo. Si osserva che nella formula, l'elemento a_{ij} compare solo una volta, moltiplicato per b_j e va a incrementare c_i. Quindi non è necessario calcolare il prodotto con l'ordine visto sopra, per esempio l'implementazione

```
1 for j=1:n
2   for i=1:n
3     c(i)=c(i)+a(i,j)*v(j);
4   end
5 end
```

fornisce lo stesso risultato. E la stessa cosa si ottiene se l'ordine con cui vengono selezionate le coppie (i, j) è arbitrario. A questo punto si può pensare di selezionare solo le coppie corrispondenti a elementi non nulli di A e fare un ciclo solo su esse.

Detto $\Omega \subset \{1, \ldots, n\} \times \{1, \ldots, n\}$ l'insieme delle coppie di indici tali che $a_{ij} \neq 0$, abbiamo che Ω contiene esattamente k elementi e il prodotto matrice-vettore $c = Av$ si può scrivere come

$$c_i = \sum_{(i,j) \in \Omega} a_{ik} v_k, \qquad i = 1, \ldots, n,$$

utilizzando la convenzione che la somma vuota è nulla (questo succede se non ci sono coppie del tipo $(i, k) \in \Omega$ e cioè se la riga i-esima di A è nulla). Notiamo che l'algoritmo scritto sopra richiede non più di $2k$ operazioni aritmetiche elementari anziché $2n^2$.

Un'implementazione in pseudocodice si può ottenere memorizzando l'insieme Ω in un array con k righe e due colonne che indicheremo con `ind`, in modo che `ind(t,1)` fornisca l'indice di riga e `ind(t,2)` fornisca l'indice di colonna dell'elemento t:

```
1 for t=1:k
2   i=ind(t,1);
3   j=ind(t,2);
4   c(i)=c(i)+a(i,j)*v(j);
5 end
```

dove si è assunto che il vettore `c` sia stato inizializzato a 0.

Questo algoritmo richiede solamente $2k$ ops, tuttavia è inefficiente dal punto di vista della gestione della memoria perché richiede che la matrice A venga mantenuta interamente in memoria. Ma, poiché gli elementi nulli non hanno nessuna funzione è possibile risparmiare memoria utilizzando un vettore di k strutture `mat`, in cui ogni struttura ha tre campi: uno per l'indice di riga i (ad es., `mat(t).i`), uno per l'indice di colonna j (ad es., `mat(t).j`) e uno per il valore a_{ij} (ad es., `mat(t).value`).

L'algoritmo relativo a questa implementazione, in pseudocodice è

```
1 for t=1:k
2   c(mat(t).i)=c(mat(t).i)+mat(t).value*b(mat(t).j);
3 end
```

Riferimenti bibliografici

Per quanto riguarda il metodo di Ruffini Horner, si faccia riferimento all'esempio 1.10 del testo [1]; per approfondire il metodo di Strassen, si faccia rifemento all'esempio 1.9 del testo [1] o all'articolo originale [2]; per gli algoritmi di somma sequenziale e parallela si faccia riferimento all'esempio 2.28 di [1], dove viene anche effettuata un'analisi dell'errore dei due algoritmi; per la parallelizzazione degli algoritmi di valutazione di polinomi si faccia riferimento a [3].

[1] R. Bevilacqua, D. Bini, M. Capovani, O. Menchi. *Metodi Numerici.* Zanichelli, Bologna. 1992.
[2] V. Strassen. *Gaussian Elimination is not Optimal.* Numer. Math., 13, 1969, pp. 354–356.
[3] D. Bertaccini, C. Di Fiore, P. Zellini. *Complessità e iterazione numerica. Percorsi, matrici e algoritmi veloci nel calcolo numerico.* Bollati Boringhieri, 2013.

2 Sistemi lineari e algoritmo di Gauss

Richiami sui sistemi lineari

Un sistema lineare di m equazioni in n incognite si può scrivere come

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2, \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m, \end{cases}$$

dove ogni equazione è lineare. Usando le sommatorie si può sintetizzare la scrittura come

$$\sum_{j=1}^{n} a_{ij}x_j = b_i, \quad i = 1, \ldots, m.$$

Tuttavia, c'è una notazione ancora più elegante. Definendo le matrici

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix},$$

il sistema può essere scritto come

$$Ax = b.$$

L'esistenza e la molteplicità delle soluzioni è caratterizzata dal seguente risultato dell'algebra lineare dove si usa la matrice $[A|b] \in \mathbb{K}^{m\times(n+1)}$ ottenuta incollando la colonna b in fondo alla matrice A.

Teorema 2.1 (Rouché-Capelli)**.** *Il sistema lineare $Ax = b$, con $A \in \mathbb{K}^{m\times n}$ e $b \in \mathbb{K}^m$, ammette soluzioni se e solo se il rango di A coincide con il rango di $[A|b]$. Nel caso in cui esistano soluzioni, esse formano un sottospazio affine di $\mathbb{K}^n$ di dimensione $n - \operatorname{rank}(A)$.*

Nel caso di matrici quadrate il teorema si semplifica molto e si ottiene il seguente corollario.

Teorema 2.2 (Cramer)**.** *Il sistema lineare quadrato $Ax = b$ ammette soluzione unica se e solo se la matrice A è invertibile.*

In quest'ultimo caso la soluzione simbolica si può scrivere semplicemente come $x = A^{-1}b$. Nell'ambito del calcolo numerico questa scrittura non è soddisfacente, al contrario, si vuole fornire un algoritmo che calcoli la soluzione.

Il problema della soluzione di un sistema lineare

Sia $A \in \mathbb{K}^{m\times n}$, essa definisce un'applicazione lineare $f : \mathbb{K}^n \to \mathbb{K}^m$, quella che al vettore $v \in \mathbb{K}^m$ associa il vettore $f(v) = Av \in \mathbb{K}^n$ (dove v e $f(v)$ sono espressi in opportune base). Si può definire l'insieme $\operatorname{Im}(A) = \{v \in \mathbb{K}^m : v = Aw \text{ per qualche } w \in \mathbb{K}^n\}$, detto l'*immagine* di A, che è un sottospazio vettoriale di $\mathbb{K}^m$ generato dalle colonne di A e la sua dimensione è $\operatorname{rank}(A)$ (cfr. Esercizio 2.4).

Tramite l'immagine è possibile avere un semplice criterio per l'esistenza di soluzioni.

Proposizione 2.3. *Il sistema lineare $Ax = b$ ammette soluzione se e solo se $b \in \operatorname{Im}(A)$.*

Ci si può chiedere sotto quali ipotesi la soluzione di un sistema lineare è un problema ben posto. Per l'esistenza è necessario che $b \in \operatorname{Im}(A)$.

Per l'unicità, occorre che l'insieme delle soluzioni sia formato da un solo elemento e quindi che sia di dimensione affine 1. In altre parole occorre che $n = \operatorname{rank}(A)$. Se $m < n$ allora il massimo rango di una matrice $m \times n$ è m e quindi $\operatorname{rank}(A) < n$ in ogni caso, ne segue che se $m < n$ allora, si hanno o nessuna o infinite soluzioni. Se $m \geqslant n$ si ha una soluzione se e solo se $n = \operatorname{rank}(A)$.

Vediamo ora la dipendenza continua dai dati. Questa proprietà è vera se, quando $Ax = b$ ammette soluzione unica, e se "perturbiamo di poco" A e b, ottenendo $\widetilde{A}$ e $\widetilde{b}$, allora anche il sistema $\widetilde{A}x = \widetilde{b}$ ammette soluzione unica. (Cioè se per ogni insieme $\{A_\varepsilon\}_\varepsilon$, $\{b_\varepsilon\}_\varepsilon$ tale che $\lim_{\varepsilon\to 0} A_\varepsilon = A$ e $\lim_{\varepsilon\to 0} b_\varepsilon = b$ si ha che esiste $t_0 > 0$ tale che per ogni $t < t_0$ il sistema lineare $A_t x = b_t$ ammette soluzione unica.)

Analizziamo il caso $m > n$. Si ha $\operatorname{rank}(A) = n < m$ quindi le colonne $c_1, \ldots, c_n$ non possono essere una base di $\mathbb{K}^m$, che è equivalente a dire che esiste $c_0 \in \mathbb{K}^m$ tale che c_0 non è combinazione lineare di $c_1, \ldots, c_n$. Se $Ax = b$ ammette soluzione allora $b \in \operatorname{Im}(A)$ e quindi $b = \alpha_1 c_1 + \cdots + \alpha_n c_n$ (poiché le colonne di A generano $\operatorname{Im}(A)$). Se si considera

il vettore $b_\varepsilon = b + \varepsilon c_0$, con $\varepsilon > 0$, allora $b_\varepsilon \notin \mathrm{Im}(A)$, e quindi il sistema lineare $Ax = b_\varepsilon$ non ammette soluzione, ma $\lim_{\varepsilon \to 0} b_\varepsilon = b$ e quindi non si ha dipendenza continua dai dati.

Infine nel caso in cui $m = n$ e A sia quadrata, il rango è n se e solo se la matrice è invertibile ed è possibile dimostrare che per ogni matrice invertibile esiste un intorno di matrici invertibili (poiché il determinante è una funzione continua). Quindi il caso $m = n$ e A matrice invertibile è l'unico in cui il problema è ben posto.

Esercizio 2.4. Sia $A \in \mathbb{K}^{m \times n}$. Mostrare che $\mathrm{Im}(A)$ è un sottospazio vettoriale di $\mathbb{K}^m$ generato dalle colonne di A.

Esercizio 2.5. Mostrare che la soluzione di un sistema lineare con matrice dei coefficienti invertibile dipende in modo continuo dai dati, cioè, dati A e b tali che A sia invertibile, allora la funzione $A^{-1}b$ è continua in A e b.

Idea degli algoritmi, sistemi triangolari

Nel seguito ci occuperemo di descrivere un'algoritmo per il calcolo della soluzione (unica) di un sistema quadrato con matrice dei coefficienti invertibile. Poiché sono note formule esplicite per la soluzione date da una funzione razionale dei dati (cfr. esercizio 2.5), allora si può pensare a un algoritmo che, in aritmetica esatta, fornisca la soluzione con un numero finito di operazioni. Un tale metodo è detto *metodo diretto* per la soluzione di un sistema lineare (in contrapposizione con i metodi iterativi che invece costruiscono una successione che approssima la soluzione). Il metodo diretto di base, che con un'opportuna variante, è anche quello più usato, è il metodo di Gauss.

L'idea del metodo di Gauss è di eseguire una serie di operazioni che trasformino il sistema $Ax = b$ in un altro sistema $Tx = \widetilde{b}$ più facile da risolvere, ma che abbia la stessa soluzione. Per prima cosa cerchiamo di capire quali sono i sistemi facili da risolvere.

Consideriamo un sistema lineare $Tx = \widetilde{b}$ la cui matrice dei coefficienti è triangolare superiore. Ad esempio, nel caso 3×3, un sistema con matrice dei coefficienti triangolare è

$$\begin{cases} t_{11}x_1 + t_{12}x_2 + t_{13}x_3 = \widetilde{b}_1, \\ t_{22}x_2 + t_{23}x_3 = \widetilde{b}_2, \\ t_{33}x_3 = \widetilde{b}_3, \end{cases}$$

la cui soluzione è ottenuta nel seguente modo

- osservando che nell'ultima equazione compare la sola variabile x_3 si può ricavare direttamente $x_3 = \widetilde{b}_3/t_{33}$;
- una volta calcolato x_3, si osserva che la seconda equazione può essere scritta come $t_{22}x_2 = \widetilde{b}_2 - t_{23}x_3$ e si può ricavare $x_2 = (\widetilde{b}_2 - t_{23}x_3)/t_{22}$;
- infine, utilizzando i valori già calcolati di x_2 e x_3, si può calcolare x_1 dalla prima equazione riscritta come $t_{11}x_1 = \widetilde{b}_1 - t_{12}x_2 - t_{13}x_3$.

Questo procedimento si può generalizzare a un sistema di n equazioni con matrice dei coefficienti triangolari, diciamo $Tx = \widetilde{b}$, ottenedo il *metodo di sostituzione all'indietro*. Si parte dall'ultima equazione, si calcola x_n e si sostituisce nella penultima, dove si calcola x_{n-1}, e così via, finché non si arriva alla prima.

Per scrivere in modo più formale l'algoritmo consideriamo l'equazione k-esima

$$t_{kk}x_k + t_{k,k+1}x_{k+1} + \cdots + t_{kn}x_n = \widetilde{b}_k,$$

che verrà risolta, quando sono già noti $x_{k+1}, \ldots, x_n$, e quindi può essere riscritta come

$$t_{kk}x_k = \widetilde{b}_k - t_{k,k+1}x_{k+1} + \cdots + t_{kn}x_n = \widetilde{b}_k - \sum_{j=k+1}^{n} t_{kj}x_j,$$

e alla fine

$$x_k = \frac{1}{t_{kk}}\left(\widetilde{b}_k - \sum_{j=k+1}^{n} t_{kj}x_j\right), \quad k = 1, \ldots, n, \tag{1}$$

con la convenzione che una somma da $n+1$ a n si intende uguale a zero.

L'equazione (1) può essere immediatamente trasformata in algoritmo, se la formula viene valutata a partire da x_n fino a x_1. Questo algoritmo prende il nome di metodo sostituzione all'indietro (o *backward (back) substitutiton*).

Diamo un'implementazione in pseudocodice del metodo di sostituzione all'indietro:

```
1 for k=n:-1:1
2    s=b(k);
3    for j=k+1:n
4       s=s-t(k,j)*x(j);
5    end
6    x(k)=s/t(k,k);
7 end
```

Analizziamo ora l'algoritmo di sostituzione all'indietro. Cercheremo di capire qual è il suo costo computazionale e per quali problemi l'algoritmo fornisce la soluzione senza *breakdown*, cioè senza interruzioni dovute a errori.

La valutazione della formula (1) (che coincide con le righe 2-6 del codice) richiede $n-k$ somme, $n-k$ prodotti e 1 divisione per un totale di $2(n-k)+1$ ops. Ma la formula va valutata per ogni k e quindi il costo totale è dato da

$$\mathcal{C}(n) = \sum_{k=1}^{n} 2(n-k)+1 = 2\frac{n(n-1)}{2} + n = n^2.$$

Si osservi che tale costo è identico al costo del prodotto di una matrice triangolare per un vettore, nonostante la risoluzione di un sistema lineare sia un problema appartentemente più difficile. Questo ci fa concludere che la risoluzione di un sistema lineare con matrice dei coefficienti triangolari è un "problema facile".

Ora cerchiamo di capire sotto quali ipotesi l'algoritmo termina. Si nota che l'unico caso in cui si ha un breakdown è quello in cui $t_{kk} = 0$ per qualche k, ma questo vale se e solo se la matrice T è singolare (si confronti l'esercizio 2.6) che è come dire che il sistema non ha soluzione unica. Ne deduciamo che nelle nostre ipotesi l'algoritmo di sostituzione all'indietro è sempre applicabile.

In modo totalmente analogo si può definire l'algoritmo di *sostituzione in avanti* (o *forward substitution*) per risolvere un sistema lineare con matrice dei coefficienti triangolare inferiore, diciamo $Lx = c$.

La differenza consiste nel fatto che adesso si può calcolare x_1 dalla prima equazione, sostituirlo nella seconda ottenendo x_2 e ottenere via via $x_3, \ldots, x_n$.

La formula per il termine x_k è

$$x_k = \frac{1}{\ell_{kk}}\left(c_k - \sum_{j=1}^{k-1} \ell_{kj} x_j\right), \quad k = 1, \ldots, n,$$

mentre un'implementazione in pseudocodice del metodo di sostituzione in avanti è data da

```
1 for k=1:n
2    s=c(k);
3    for j=1:k-1
4       s=s-l(k,j)*x(j);
5    end
6    x(k)=s/l(k,k);
7 end
```

Come per il metodo di sostituzione in avanti il costo dell'algoritmo è di n^2 ops ed è sempre applicabile se la matrice L è invertibile.

Nota. I sistemi triangolari non sono gli unici sistemi che si sanno risolvere in $O(n^2)$ ops. Un altro caso interessante è quello del sistema $Qx = b$ dove $Q \in \mathbb{R}^{n\times n}$ è una matrice ortogonale, cioè una matrice tale che $Q^TQ = I$. In questo caso $Q^{-1} = Q^T$ e quindi $x = Q^{-1}b = Q^Tb$ si ottiene moltiplicando la trasposta di Q per b, con un costo di circa $2n^2$ ops. Questa idea è usata nel metodo di soluzione di un sistema lineare basato sulla fattorizzazione QR.

Esercizio 2.6. Mostrare che una matrice triangolare è invertibile se e solo se tutti gli elementi della diagonale sono diversi da 0.

Soluzione. Una matrice è invertibile se e solo se il suo determinante è diverso da 0. Si osserva che il determinante di una matrice triangolare è il prodotto degli elementi sulla diagonale (cfr. esercizio 2.7) e questo prodotto è diverso da 0 se e solo se lo sono tutti i suoi fattori, cioè tutti gli elementi della diagonale. Si conclude che una matrice triangolare è invertibile se e solo se gli elementi sulla sua diagonale sono tutti nulli. □

Esercizio 2.7. Mostrare che il determinante di una matrice triangolare è il prodotto degli elementi sulla sua diagonale.

Soluzione. Sia $T \in \mathbb{K}^{n\times n}$. Si procede per induzione sulla dimensione della matrice. Per $n = 1$, la matrice T è sempre triangolare superiore. Supponiamo ora che $n > 1$ e che la proprietà sia vera per tutte le matrici triangolari di dimensione $n-1$. Applicando la regola di Laplace alla prima colonna della matrice

$$T = \begin{bmatrix} t_{11} & t_{12} & \cdots & t_{1n} \\ 0 & t_{22} & \cdots & t_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & t_{nn} \end{bmatrix},$$

si ha che

$$\det(T) = (-1)^{1+1} t_{11} \det\left(\begin{bmatrix} t_{22} & t_{23} & \cdots & t_{2n} \\ 0 & t_{33} & \cdots & t_{3n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & t_{nn} \end{bmatrix}\right) \overset{\text{ipotesi induttiva}}{=} t_{11}t_{22}\cdots t_{nn}.$$

□

Il metodo di Gauss

Abbiamo visto che la soluzione di un sistema lineare la cui matrice dei coefficienti è triangolare superiore richiede "solo" n^2 ops. L'algoritmo di Gauss permette di trasformare un sistema lineare $Ax = b$ in cui A e invertibile (e sotto altre ipotesi che preciseremo in seguito) in un sistema lineare $Ux = \widetilde{b}$ la cui matrice dei coefficienti è triangolare superiore.

Ci sono vari modi di vedere l'algoritmo di Gauss:

- come una sequenza di operazioni sulla matrice A con l'obiettivo di ottenere U triangolare superiore;
- come una fattorizzazione $A = LU$, dove L è triangolare inferiore con uno sulla diagonale e U è triangolare superiore.

Inizialmente, considereremo il primo approccio che è più vicino a quello che si vede nei corsi di algebra lineare (matematica discreta) ed è anche più comprensibile dal punto di vista algoritmico. In seguito parleremo dell'approccio basato sulla fattorizzazione che è più moderno e versatile.

L'algoritmo si basa sul fatto che in un sistema lineare di n equazioni in n incognite con soluzione unica, scelti $i, j \in \{1, \ldots, n\}$, se si sostituisce l'equazione i-esima con la somma dell'equazione i-esima e un multiplo dell'equazione j-esima la soluzione non cambia. Nel nostro formalismo matriciale questa operazione corrisponde a sostituire alla riga i della matrice, la somma della riga i con un multiplo della riga j.

Vediamo com'è possibile, con questo tipo di operazioni, ottenere una sistema triangolare a partire dalla matrice $[A|b]$. Struttureremo l'algoritmo in $n-1$ passi.

Al primo passo si pone $[A_1|b_1] = [A|b]$ e si cerca di ottenere una matrice che abbia la prima colonna nulla eccetto il primo elemento. Per far questo si eliminano gli elementi $a_{21}, a_{31}, \ldots, a_{n1}$ in sequenza, sostituendo la riga i, per $i = 2, \ldots, n$ con una opportuna combinazione di essa e della prima riga.

Per far questo si costruiscono i numeri $\ell_{21} = \frac{a_{21}}{a_{11}}$, $\ell_{31} = \frac{a_{31}}{a_{11}}$, e così via fino a $\ell_{n1} = \frac{a_{n1}}{a_{11}}$ e tramite essi si combinano le righe nel seguente modo $r_2 = r_2 - \dfrac{a_{21}}{a_{11}} r_1$, $r_3 = r_3 - \dfrac{a_{31}}{a_{11}} r_1$ e così via fino a $r_n = r_n - \dfrac{a_{n1}}{a_{11}} r_1$.

Per fissare le idee descriviamo in dettaglio il caso 4×4 (omettendo il vettore b_1 per semplicità):

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \quad \begin{array}{c} r_2 = r_2 - \dfrac{a_{21}}{a_{11}} r_1 \\ \longrightarrow \\ 7 \text{ ops} = 1\text{d}+3\text{m}+3\text{a} \end{array}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \quad \begin{array}{c} r_3 = r_3 - \dfrac{a_{31}}{a_{11}} r_1 \\ \longrightarrow \\ 7 \text{ ops} = 1\text{d}+3\text{m}+3\text{a} \end{array} \quad \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} & a_{34}^{(2)} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

$$\begin{array}{c} r_4 = r_4 - \dfrac{a_{41}}{a_{11}} r_1 \\ \longrightarrow \\ 7 \text{ ops} = 1\text{d}+3\text{m}+3\text{a} \end{array} \quad \left[\begin{array}{c|ccc} a_{11} & a_{12} & a_{13} & a_{14} \\ \hline 0 & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} & a_{34}^{(2)} \\ 0 & a_{42}^{(2)} & a_{43}^{(2)} & a_{44}^{(2)} \end{array}\right] = A_2,$$

$$L_2 = \begin{bmatrix} 1 & & & \\ \ell_{21} & 1 & & \\ \ell_{31} & 0 & 1 & \\ \ell_{41} & 0 & 0 & 1 \end{bmatrix}, \quad b_2 = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ b_3^{(2)} \\ b_4^{(2)} \end{bmatrix}.$$

Come si può osservare il costo del calcolo delle frazioni $\ell_{1i} := \frac{a_{1i}}{a_{11}}$ è di $n-1$ ops (3 ops nel caso 4×4), mentre la combinazione delle righe costa 2 operazioni per ogni elemento e quindi $2(n-1)$ per riga (6 nel caso particolare), per un totale $2(n-1)^2$ ops ($6 \cdot 3$ nel caso particolare). Il costo totale del calcolo di A_2 è quindi di $2(n-1)^2+(n-1)$ ops. La matrice L_2 accumula gli elementi ℓ_{ij} e non richiede ulteriori calcoli. Infine il calcolo di b_2 richiede 2 ops per ogni componente, quindi un totale di $2(n-1)$ ops, visto che la prima componente resta invariata (nel caso 4×4 le operazioni sono 6).

Al secondo passo si cerca di eliminare dalla matrice $[A_2|b_2]$ gli elementi della seconda colonna che stanno sotto l'elemento di posizione $(2,2)$ senza modificare la prima colonna. Per far questo è sufficiente ripetere lo stesso procedimento descritto sopra sulla matrice ottenuta eliminando la prima riga e la prima colonna da A_2 che è una matrice di dimensione $n-1$. Otterremo $[A_3|b_3]$ come desiderata, al passo tre occorrerà ripetere il procedimento per una matrice $n-2$; e così via fino ad arrivare al passo $n-1$ dove si lavora su una matrice 2×2.

Per fissare le idee riassumiamo gli altri passi nel caso $n = 4$.

$$A_2 = \left[\begin{array}{c|ccc} a_{11} & a_{12} & a_{13} & a_{14} \\ \hline 0 & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} & a_{34}^{(2)} \\ 0 & a_{42}^{(2)} & a_{43}^{(2)} & a_{44}^{(2)} \end{array}\right] \quad \begin{array}{c} r_i = r_i - \dfrac{a_{i1}^{(2)}}{a_{22}^{(2)}} r_2, i = 3,4 \\ \longrightarrow \\ 6 \text{ ops} = 2\text{d}+4\text{m}+4\text{a} \end{array}$$

$$A_3 = \left[\begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ \hline 0 & 0 & a_{33}^{(3)} & a_{34}^{(3)} \\ 0 & 0 & a_{43}^{(3)} & a_{44}^{(3)} \end{array}\right] \quad \begin{array}{c} r_4 = r_4 - \dfrac{a_{43}^{(3)}}{a_{33}^{(3)}} r_3, \\ \longrightarrow \\ 3 \text{ ops} = 1\text{d}+1\text{m}+1\text{a} \end{array}$$

$$U = A_4 = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & a_{24}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & a_{34}^{(3)} \\ 0 & 0 & 0 & a_{44}^{(4)} \end{bmatrix}$$

$$L = \begin{bmatrix} 1 & & & \\ \ell_{21} & 1 & & \\ \ell_{31} & \ell_{32} & 1 & \\ \ell_{41} & \ell_{42} & \ell_{43} & 1 \end{bmatrix}, \quad \widetilde{b} = b_4 = \begin{bmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(3)} \\ b_4^{(4)} \end{bmatrix}.$$

Al secondo passo si devono calcolare $n-2$ frazioni del tipo $\frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}$ e si devono sommare gli ultimi $n-2$ elementi delle ultime $n-2$ righe con quelli della seconda riga, per un totale di $2(n-2)^2$ operazioni.

In definitiva al passo k si effettuano $(n-k)$ operazioni per calcolare i valori $\ell_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}$ e $2(n-k)^2$ operazioni per fare le combinazioni delle righe. Il calcolo di $b^{(k)}$ richiede $2(n-k)$ operazioni.

Il costo totale della operazioni su A è dunque $\sum_{k=1}^{n-1} 2(n-k)^2 + (n-k)$ operazioni la cui parte principale è $\frac{2}{3}n^3$. Il calcolo di $\widetilde{b}$ richiede $\sum_{k=1}^{n-1} 2(n-k)$ operazioni la cui parte principale è n^2 (trascurabile rispetto a n^3). Una volta trovate U e $\widetilde{b}$, si può risolvere il sistema $Ux = \widetilde{b}$ con l'algoritmo di sostituzione all'indietro per cui sono necessarie altre $\sum_{k=1}^{n}(2k-1)$ operazioni, la cui parte principale è n^2 (trascurabile rispetto a n^3).

Si può concludere che la soluzione di un sistema lineare con il metodo di Gauss richiede $\frac{2}{3}n^3$ ops. Il metodo è detto anche metodo di eliminazione di Gauss o *Gaussian elimination*.

Per come è descritto però non è chiaro sotto quali condizioni l'algoritmo dovrebbe terminare, visto che vengono effettuate delle divisioni il cui denominatore può essere nullo. Prima di discutere di questi aspetti daremo alcuni differenti approcci dell'algoritmo di Gauss: prima riscriveremo l'algoritmo in termini delle singole componenti e poi faremo vedere che esso coincide con la fattorizzazione LU della matrice A.

Implementazione del metodo di Gauss

Riscriveremo la relazione di ricorrenza che fornisce gli elementi di A_{k+1} e b_{k+1}, per $k = 1, \ldots, n-1$, in termini degli elementi di A_k e b_k anziché delle righe delle matrici $[A_k|b_k]$, questo rende più facile l'implementazione.

Detti $a_{ij}^{(k)}$ gli elementi di A_k, posto $\ell_{ij}^{(k)} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}$ e $b_i^{(k)}$ gli elementi di $b^{(k)}$, al passo k si ha

$$\begin{cases} a_{ij}^{(k+1)} = a_{ij}^{(k)}, \quad i = 1, \ldots, k, \quad j = 1, \ldots, n, \\ a_{ij}^{(k+1)} = 0, \quad i = k+1, \ldots, n, \quad j = 1, \ldots, k, \\ a_{ij}^{(k+1)} = a_{ij}^{(k)} - \dfrac{a_{ik}^{(k)}}{a_{kk}^{(k)}} a_{kj}^{(k)}, \quad i = k+1, \ldots, n, \quad j = k+1, \ldots, n. \end{cases}$$

Da queste formule si deduce facilmente che il costo del passo k è di $2(n-k)^2$ ops e il costo totale per formare U è $\sum_{k=1}^{n-1} 2(n-k)^2$ ops la cui parte principale è $\frac{2}{3}n^3$ ops.

Il calcolo di $\widetilde{b} = b^{(n)}$ può essere ottenuto aggiungendo alla relazione sopra le righe

$$\begin{cases} b_i^{(k+1)} = b_i^{(k)}, \quad i = 1, \ldots, k \\ b_i^{(k+1)} = b_i^{(k+1)} - \dfrac{a_{ik}^{(k)}}{a_{kk}^{(k)}} b_k^{(k)}, \quad i = k+1, \ldots, n, \end{cases}$$

il cui costo è $2(n-k)$ (si ricordi $\frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}$ è già stato calcolato) che e quindi il costo totale per il calcolo di $\widetilde{b}$ è $\sum_{k=1}^{n-1} 2(n-k)$ ops la cui parte principale è n^2 ops. Questo costo è trascurabile rispetto al costo del calcolo degli elementi di A_k, tranne in rari casi in cui A è particolarmente strutturata.

Queste formule si traducono immediatamente in pseudocodice, dove, anziché definire le sequenze A_k e b_k per $k = 1, \ldots, n$, si sovrascrivono i dati, com'è uso in informatica.

```
for k=1:n-1
   % ciclo che scorre i passi del metodo di Gauss
   for i=k+1:n
      % ciclo che scorre le righe
      ell(i,k)=a(i,k)/a(k,k);
      a(i,k)=0;
      for j=k+1:n
        % ciclo che scorre gli elementi della riga i
        a(i,j)=a(i,j)-ell(i,k)*a(k,j);
      end
      b(i)=b(i)-ell(i,k)*b(k);
   end
end
```

A titolo di curiosità, osserviamo che se ci interessa solamente calcolare la matrice U, è possibile dare un'implementazione del metodo di Gauss in una sola istruzione

```
for k=1:n-1
  for i=k+1:n
    for j=n:-1:k
      a(i,j)=a(i,j)-a(i,k)*a(k,j)/a(k,k);
    end
  end
end
```

dove il terzo ciclo `for` prosegue con passo negativo perché il valore `a(i,k)`, che diventa nullo, va assegnato dopo che tutti gli altri elementi della riga (`a(i,j)` con $j > k$) sono stati calcolati.

Il metodo di Gauss come fattorizzazione LU

Mostreremo ora che il metodo di Gauss applicato ad una matrice A, se termina, permette di scrivere A come prodotto di una matrice triangolare inferiore con 1 sulla diagonale per una matrice triangolare superiore. In particolare $A = LU$ dove L è la matice degli elementi ℓ_{ij} e $U = A_n$. Tale fattorizzazione è detta *fattorizzazione* LU di A e, nell'ipotesi che la matrice A sia invertibile, se esiste è unica.

Teorema 2.8. *Sia $A \in \mathbb{K}^{n\times n}$ e siano L e U le matrici ottenute tramite il metodo di Gauss, allora $A = LU$.*

Dimostrazione. Ricordiamo che, detta $A_k = (a_{ij}^{(k)})_{i,j=1,\dots,n}$, la matrice ottenuta al passo k del metodo di Gauss, con $A_1 = A$ e $A_n = U$, si ha per definizione

$$u_{kj} = a_{kj}, \text{ per ogni } k \text{ e } j, \qquad a_{ij}^{(k+1)} = a_{ij}^{(k)} - \ell_{ik} a_{kj}^{(k)}, \text{ per } k = 1, \dots, n-1, \text{ e } i > k, j \geqslant k,$$

da cui segue che $\ell_{ik} a_{kj}^{(k)} = a_{ij}^{(k)} - a_{ij}^{(k+1)}$. Si osservi che $u_{kj} = a_{kj}^{(k)}$, per ogni k e j, segue dal fatto che la riga k-esima non viene mai modificata dal passo k in poi.

Dunque, per l'elemento di indici (i, j) di LU si ha

$$(LU)_{ij} = \sum_{k=1}^{n} \ell_{ik} u_{kj} \overset{\text{triangolarità}}{=} \sum_{k=1}^{\min\{i,j\}} \ell_{ik} u_{kj} \overset{[u_{kj} = a_{kj}^{(k)}]}{=} \sum_{k=1}^{\min\{i,j\}} \ell_{ik} a_{kj}^{(k)}$$

dove la penultima uguaglianza segue dal fatto che L è triangolare inferiore e quindi $\ell_{ik} = 0$ per $k > i$ e U è triangolare superiore e quindi $u_{kj} = 0$ per $k > j$.

Distinguiamo ora due casi: $i \leqslant j$ e $i > j$.

Se $i \leqslant j$, è $\min\{i, j\} = i$, per cui

$$\begin{aligned}(LU)_{ij} = \sum_{k=1}^{i} \ell_{ik} u_{kj} \overset{[\ell_{ii}=1]}{=} \sum_{k=1}^{i-1} \ell_{ik} a_{kj}^{(k)} + a_{ij}^{(i)} \overset{[a_{ij}^{(k)} - a_{ij}^{(k+1)} = \ell_{ik} a_{kj}^{(k)}]}{=} \sum_{k=1}^{i-1} \big(a_{ij}^{(k)} - a_{ij}^{(k+1)}\big) + a_{ij}^{(i)} \\ = a_{ij}^{(1)} - a_{ij}^{(2)} + a_{ij}^{(2)} - a_{ij}^{(3)} + \cdots + a_{ij}^{(i-1)} - a_{ij}^{(i)} + a_{ij}^{(i)} = a_{ij}^{(1)} = a_{ij}.\end{aligned}$$

Se $i > j$, è $\min\{i, j\} = j$, per cui

$$(LU)_{ij} = \sum_{k=1}^{j} \ell_{ik} u_{kj} = a_{ij}^{(1)} - a_{ij}^{(2)} + a_{ij}^{(2)} - a_{ij}^{(3)} + \cdots + a_{ij}^{(j)} - a_{ij}^{(j+1)} = a_{ij}^{(1)} - a_{ij}^{(j+1)} = a_{ij}$$

dove l'ultima uguaglianza segue dal fatto che $a_{ij}^{(j+1)} = 0$ poiché è un elemento sotto la diagonale e nella colonna j di A_{j+1} gli elementi sotto la diagonale sono nulli. □

Applicabilità dell'algoritmo di Gauss

L'algoritmo di Gauss applicato a una matrice $A \in \mathbb{K}^{n\times n}$ richiede che a ogni passo il pivot non sia nullo, cioè che $a_{kk}^{(k)} \neq 0$. Se esiste $k < n$ tale che $a_{kk}^{(k)} = 0$ mentre $a_{jj}^{(j)} \neq 0$ per $j = 1, \dots, k$, si dice che l'algoritmo ha *breakdown* al passo k, viceversa si dice che il metodo di Gauss è applicabile alla matrice A se non ha breakdown.

Diamo ora un teorema che fornisce una condizione necessaria e sufficiente affinché l'algoritmo sia applicabile ad una data matrice e cioè che i suoi minori principali di testa siano invertibili.

Teorema 2.9. *Sia $A \in \mathbb{K}^{n\times n}$ e siano M_k i suoi minori principali di testa di ordine k, per $k = 1, \dots, n-1$, allora l'algoritmo di Gauss è applicabile alla matrice A se e solo se* $\det(M_k) \neq 0$.

Dimostrazione. Supponiamo dapprima che il metodo sia applicabile, cioè che $a_{kk}^{(k)} \neq 0$ per ogni k, allora si può costruire la matrice $U = A_n$. Siano $N_1, \dots, N_{n-1}$, i suoi minori principali, essi sono triangolari superiori e quindi $\det(N_j) = a_{11}^{(1)} \cdots a_{jj}^{(j)} \neq 0$, per $j = 1, \dots, n-1$. Ma $\det(N_j) = \det(M_j)$ in quanto N_j è ottenuto da M_j mediante operazioni sulle righe che non cambiano il determinante e quindi $\det(M_k) \neq 0$ per ogni k. Infatti le operazioni sulle righe della matrice A_k, inducono operazioni sulle righe del minore dello stesso tipo (somma di una riga, con un multiplo di una riga sopra di essa).

Viceversa, ragioniamo per assurdo. Nell'ipotesi che $\det(M_k) \neq 0$ per ogni k, e che il metodo di Gauss abbia un breakdown al passo ℓ, cerchiamo una contraddizione. Siccome il metodo di Gauss ha breakdown al passo ℓ, è possibile costruire la matrice A_ℓ, ma $a_{\ell\ell}^{(\ell)} = 0$. Sia S_ℓ il minore principale di testa di ordine ℓ di A_ℓ, esso è triangolare superiore e il suo determinante coincide con quello di M_ℓ perché è ottenuto da esso mediante operazioni sulle righe che non cambiano il determinante, quindi $\det(M_\ell) = \det(S_\ell) = a_{11}^{(1)} \cdots a_{\ell\ell}^{(\ell)} = 0$. Assurdo, poiché avevamo supposto che $\det(M_\ell) \neq 0$. Ne concludiamo che il metodo di Gauss è applicabile.

□

Da questo teorema segue un'espressione esplicita per i pivot.

Corollario 2.10. *Sia $A \in \mathbb{K}^{n\times n}$ una matrice per cui il metodo di Gauss sia applicabile e siano M_k i suoi minori principali di testa e $a_{kk}^{(k)}$ i suoi pivot per $k = 1, \dots, n-1$. Allora $a_{11}^{(1)} = \det M_1$ e $a_{kk}^{(k)} = \det M_k / \det M_{k-1}$ per $k = 1, \dots, n-1$.*

Dimostrazione. Il primo caso si verifica direttamente, infatti $a_{11}^{(1)} = a_{11} = \det[a_{11}] = \det M_1$. Per $k > 1$ nel Teorema 2.9 si ha $\det M_k = a_{11}^{(1)} \cdots a_{kk}^{(k)}$ e $\det M_{k-1} = a_{11}^{(1)} \cdots a_{k-1,k-1}^{(k-1)}$, da cui segue che $a_{kk}^{(k)} = \det M_k / \det M_{k-1}$. □

Il legame tra la fattorizzazione LU di una matrice e l'algoritmo di Gauss ci permette di enunciare il seguente teorema.

Corollario 2.11. *Una matrice $A \in \mathbb{K}^{n\times n}$ ammette fattorizzazione LU unica se e solo se i suoi minori principali di testa di ordine k, per $k = 1, \dots, n-1$ sono invertibili.*

Stabilità del metodo di Gauss

L'applicabilità del metodo di Gauss è legata al fatto che i pivot siano tutti diversi da zero, tuttavia può succedere che il metodo sia applicabile ma, in aritmetica finita, non fornisca la soluzione cercata, cioè sia numericamente instabile come mostra il seguente esempio.

Si consideri il sistema lineare $Ax = b$ con

$$A = \begin{bmatrix} \varepsilon & 1 \\ 1 & 0 \end{bmatrix}, \quad b = \begin{bmatrix} 1+\varepsilon \\ 1 \end{bmatrix},$$

la cui soluzione unica, per ogni $\varepsilon > 0$, è $x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. Questo sistema lineare, per piccoli valori di ε, non ha grossi problemi di condizionamento, infatti il numero di condizionamento di A in norma 1 è $(1+\varepsilon)^2$.

Risolvendo il sistema lineare con il metodo di Gauss per $\varepsilon = 0.3 \cdot 10^{-1}$ e lavorando in aritmetica finita in base 10 e con 2 cifre significative si ha

$$\begin{cases} 0.3 \cdot 10^{-1}\widetilde{x}_1 + \widetilde{x}_2 = 1, \\ \quad -0.33 \cdot 10^{2}\widetilde{x}_2 = -0.33 \cdot 10^{2}, \end{cases}$$

la cui soluzione è $\widetilde{x} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, ben lontana dalla soluzione effettiva. Se ne deduce che il metodo di Gauss può essere instabile!

In generale, l'instabilità si ha quando i pivot sono piccoli e quindi gli elementi della matrice U possono diventare anche molto grandi. Per compredere come l'errore sia legato alla grandezza degli elementi di L e U, diamo il seguente risultato.

Teorema 2.12. *Sia A una matrice di numeri di macchina di ordine n e siano $\widetilde{L}$ e $\widetilde{U}$ le matrici della fattorizzazione LU di A effettivamente calcolate con il metodo di Gauss, allora $\widetilde{L}\widetilde{U} = A + H$, dove $|h_{ij}| \leqslant 2nu(|a_{ij}| + |\widetilde{\ell}_{ij}||\widetilde{u}_{ij}|) + O(u^2)$, dove u è la precisione di macchina.*

Ci si aspetta quindi un grosso errore se gli elementi di L e U diventano grandi relativamente ad A.

Per fortuna esiste una variante del metodo di Gauss che risolve molti dei problemi di instabilità; essa verrà descritta nella prossima sezione.

Esercizio 2.13. Si risolva in aritmetica finita in base 10 e con 2 cifre significative il sistema lineare ottenuto scambiando le righe della matrice A e del vettore b nell'esempio numerico precedente (con $\varepsilon = 0.3 \cdot 10^{-1}$).

Variante del massimo pivot parziale

La condizione di applicabilità vista sopra non è molto agevole da verificare, in quanto occorre dimostrare che i minori principali di testa di A sono invertibili. (Esistono tuttavia dei casi in cui la condizione di applicabilità si può dimostrare per via teorica.) Questo è un serio problema del metodo di Gauss.

Naturalmente si può pensare che il caso in cui la matrice abbia minori singolari è un caso raro. In aritmetica finita, tuttavia, ci sono seri problemi anche se il valore di $a_{kk}^{(k)}$ non è nullo, ma è molto piccolo, come mostra l'esempio nella precedente sezione.

Ne concludiamo che il metodo di Gauss è un algoritmo instabile. Fortunatamente, una piccola variante di esso si è rivelata estremamente stabile nei problemi pratici: si tratta della variante del massimo pivot parziale o *Gaussian elimination with partial pivoting (GEPP)*.

Siccome il problema è dato dal fatto che al passo k, l'elemento $a_{kk}^{(k)}$ può essere piccolo, la variante consiste nel cercare nella colonna k tra gli elementi $a_{kk}^{(k)}, a_{k+1,k}^{(k)}, \ldots, a_{nk}^{(k)}$, quello più grande in modulo, diciamo $a_{sk}^{(k)}$ e scambiare la riga k con la riga s. In questo modo si cerca di controllare l'aumento dei moduli degli elementi di L e U.

Vediamo l'implementazione in pseudocodice

```
for k=1:n-1
   % ciclo che scorre i passi del metodo di Gauss
   % prima si cerca l'indice dell'elemento di massimo modulo
   s=k; m=abs(a(k,k));
   for i=k+1:n
      t=abs(a(i,k));
      if (t>m)
         s=i; m=t;
      end
   end
   % se k non coincide con s si scambiano le righe k e s
   if (k!=s)
      for j=k:n
         swap=a(k,j); a(k,j)=a(s,j); a(s,j)=swap;
      end
      swap=b(k); b(k)=b(s); b(s)=swap;
   end
   for i=k+1:n
      % come nel metodo di Gauss standard
      ...
   end
end
```

La variante del massimo pivot parziale richiede $n-k$ confronti al passo k, che sommati danno un numero totale di confronti pari a $O(n^2)$. Se assumiamo che il costo di un confronto sia dell'ordine del costo di un'operazione aritmetica, l'aumento di costo risulta trascurabile rispetto al costo delle operazioni di eliminazione. Si può concludere che la variante è sostanzanzialmente gratuita.

Il metodo di Gauss con la variante del massimo pivot parziale, se applicato alla soluzione di un sistema lineare, costruisce la sequenza

$$[\widehat{A}_1|\widehat{b}_1] = [A|b] \to [A_1|b_1] \to [\widehat{A}_2|\widehat{b}_2] \to [A_2|b_2] \to \cdots \to [A_n|b_n] =: [U|\widetilde{b}],$$

dove $[\widehat{A}_k|\widehat{b}_k]$ è ottenuta eseguendo un eventuale scambio di righe su $[A_k|b_k]$.

Se il metodo di Gauss fallisce è perché ad un certo passo k, si ha che $a_{kk}^{(k)} = 0$; la variante del massimo pivot parziale cerca di porre un rimedio scambiando la riga k con una riga in cui l'elemento sulla colonna k ha massimo modulo. In linea di principio anche il metodo di Gauss con pivoting può fallire, se tutti gli elementi $a_{kk}^{(k)}, a_{k+1,k}^{(k)}, \ldots, a_{nk}^{(k)}$, sono nulli, tuttavia questo non può accadere se la matrice A è invertibile, come mostra il seguente risultato.

Teorema 2.14. *Sia $A \in \mathbb{K}^{n\times n}$ invertibile allora l'algoritmo di Gauss con la variante del massimo pivot parziale è applicabile alla matrice A.*

Dimostrazione. Supponiamo che il metodo di Gauss con pivoting non sia applicabile alla matrice A e mostriamo che essa non è essere invertibile, in questo modo avremo dimostrato il teorema.

Supponiamo che il metodo di Gauss con pivoting si arresti al passo $1 \leqslant k \leqslant n-1$, allora si avrà che $a_{kk}^{(k)} = 0$ (e anche $a_{jj}^{(j)} \neq 0$ per $1 \leqslant j < k$, si noti che questa condizione è vuota per $k = 1$).

Avendo supposto che il metodo fallisca al passo k, si ha che gli elementi $a_{ik}^{(k)}$ della colonna k della matrice A_k, per $i = k, \ldots, n$, sono nulli. Concentriamoci sulla sottomatrice ottenuta selezionando le prime k colonne di A_k, che chiamiamo $C_k \in \mathbb{K}^{n\times k}$. La matrice C_k ha righe nulle dalla k-esima in poi, quindi ha al più $k-1$ righe non nulle, da cui il suo rango è al più $k-1$ (non può essere k perché tutti i minori di ordine k hanno una riga nulla e quindi hanno determinante 0). Ma siccome C_k ha rango minore di k, una sua colonna è combinazione lineare delle altre.

Ma questa colonna di C_k è anche una colonna di A_k, che è combinazione lineare delle altre (tutte le colonne di C_k sono colonne di A_k) e quindi A_k non è invertibile e $\det(A_k) = 0$.

Le operazioni eseguite dal metodo di Gauss con pivoting possono cambiare solo il segno del determinante, allora si ha che $\det(A) = \det(A_k) = 0$ e quindi anche la matrice A non è invertibile. □

Come abbiamo suggerito nella precedente sezione l'instabilità è dovuta al fatto che nel metodo di Gauss gli elementi di L e U possono diventare grandi a piacere. Ora vediamo come crescono gli elementi di L e U nel caso in cui si usi la variante del massimo pivot parziale.

Innanzitutto si osserva che, nel caso in cui si usi la variante del massimo pivot parziale, gli elementi di L sono tutti minori o uguali a uno in modulo. Infatti, $|\ell_{ik}| = \frac{|\widehat{a}_{ik}^{(k)}|}{|\widehat{a}_{kk}^{(k)}|} \leqslant 1$ poiché $|\widehat{a}_{kk}^{(k)}| \geqslant |\widehat{a}_{ik}^{(k)}|$.

Per quanto riguarda gli elementi di U, detto $M = \max_{i,j=1,\ldots,n} |a_{ij}|$, si hanno i due casi $|a_{ij}^{(2)}| = |\widehat{a}_{ij}^{(1)}| \leqslant M$ o $|a_{ij}^{(2)}| = |\widehat{a}_{ij}^{(1)} - \ell_{i1}\widehat{a}_{ki}^{(1)}| \leqslant |\widehat{a}_{ij}^{(1)}| + |\ell_{i1}||\widehat{a}_{ki}^{(1)}| \leqslant 2M$ e quindi ogni ogni elemento di A_2 è in modulo minore o uguale a $2M$. In modo analogo si dimostra che ogni elemento di A_3 è in modulo minore o uguale a $4M$ e così via fino a mostrare che ogni elemento di $U = A_n$ è in modulo minore o uguale a $2^{n-1}M$.

In questo modo abbiamo mostrato che gli elementi di L e U non possono crescere a dismisura, ma sono controllati. La maggiorazione data sopra è ottimale (cfr. esercizio 2.15) quindi la crescita degli elementi di A_k è esponenziale nel caso peggiore. Tuttavia, per un motivo non del tutto chiaro, il caso peggiore non capita se non lo si va a cercare e nella pratica l'algoritmo di Gauss con la variante del massimo pivot parziale è usato comunemente con risultati ottimi.

Nel caso dell'algoritmo di Gauss standard applicato alla matrice A, le matrici L e U che vengono costruite sono tali che $A = LU$, cioè viene calcolata la fattorizzazione LU di A. Nel caso della variante del massimo pivot parziale questa equivalenza sparisce, tuttavia si può dimostrare che le matrici L e U ottenute dalla GEPP sono tali che LU sia uguale alla matrice ottenuta dalla matrice A permutando opportunamente le righe (la permutazione è quella ottenuta componendo tutti gli scambi di righe).

Esercizio 2.15. Si consideri la matrice

$$A = \begin{bmatrix} 1 & 0 & 0 & 1 \\ -1 & 1 & 0 & 1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & 1 \end{bmatrix}.$$

Si mostri che l'elemento di massimo modulo della matrice $U = A_4$ ottenuta applicando l'algoritmo GEPP alla matrice A è 8 cioè realizza la maggiorazione vista sopra. Trovare una matrice $n \times n$ il cui elemento di massimo modulo è 1 e tale che il massimo modulo di U è 2^{n-1}.

Soluzione. Applicando il metodo di Gauss con pivoting, si osserva che non sono necessari scambi di righe e quindi

$$A_1 = \begin{bmatrix} 1 & 0 & 0 & 1 \\ -1 & 1 & 0 & 1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & 1 \end{bmatrix} \to A_2 = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & -1 & 1 & 2 \\ 0 & -1 & -1 & 2 \end{bmatrix} \to A_3 = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & -1 & 4 \end{bmatrix} \to A_4 = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 8 \end{bmatrix},$$

da cui l'elemento di massimo modulo di A_4 è 8. In generale una matrice $n \times n$ tale che ha 1 sulla diagonale e sull'ultima colonna, mentre vale -1 sotto la diagonale e zero altrove produce una matrice U tale che l'elemento di massimo modulo è 2^{n-1}. □

Scambi di righe come prodotto di matrici

Supponiamo di voler permutare le righe di una matrice $A \in \mathbb{K}^{m\times m}$, secondo una permutazione $\sigma \in \mathfrak{S}_m$ data. Questa operazione può essere eseguita moltiplicando la matrice A per un'opportuna matrice, detta di permutazione, che è costruita a partire da σ.

Si può costruire una matrice di permutazione nel seguente modo: detti $e_1, \dots, e_m$ i vettori della base canonica di $\mathbb{K}^n$ e data $\sigma \in \mathfrak{S}_n$, si ha

$$\sigma \to \Pi_\sigma = [e_{\sigma(1)} \,|\, e_{\sigma(2)} \,|\, \cdots \,|\, e_{\sigma(m)}].$$

Esempio 2.16. Se $\sigma = \mathrm{id} \in \mathfrak{S}_m$, cioè la permutazione identica, per cui $\sigma(i) = i$ per ogni i, allora si ha $\rho(e) = I$. Infatti

$$\rho(e) = \left[e_{\sigma(1)} | e_{\sigma(2)} | \cdots | e_{\sigma(m)}\right] = [e_1 | e_2 | \cdots | e_m] = I.$$

Esempio 2.17. Se $\sigma \in \mathfrak{S}_2$ è la permutazione che scambia 1 e 2, cioè tale che $\sigma(1) = 2$ e $\sigma(2) = 1$, allora

$$\rho(\sigma) = \left[e_{\sigma(1)} | e_{\sigma(2)}\right] = [e_2 | e_1] = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

Esempio 2.18. Se $\sigma \in \mathfrak{S}_3$ è la permutazione tale che $\sigma(1) = 2$, $\sigma(2) = 3$ e $\sigma(3) = 1$, allora

$$\rho(\sigma) = \left[e_{\sigma(1)} | e_{\sigma(2)} | e_{\sigma(3)}\right] = [e_2 | e_1 | e_3] = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Applicando una matrice di permutazione Π_σ a un vettore $v \in \mathbb{K}^m$, gli elementi di v saranno permutati secondo σ, infatti

$$\Pi_\sigma v = [e_{\sigma(1)} \,|\, \cdots \,|\, e_{\sigma(m)}] \begin{bmatrix} v_1 \\ \vdots \\ v_m \end{bmatrix} = v_1 e_{\sigma(1)} + \cdots + v_m e_{\sigma(m)} =: \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix},$$

dove $v_i = w_{\sigma(i)}$, per $i = 1, \dots, m$. Infatti v_1, moltiplicato per $e_{\sigma(1)}$, va a finire nella posizione $\sigma(1)$ del vettore w, da cui $v_1 = w_{\sigma(1)}$ e questo si ripete per ognuna delle componenti.

In questo modo siamo riusciti a mandare l'elemento v_1 nella posizione $v_{\sigma(1)}$, l'elemento v_2 nella posizione $v_{\sigma(2)}$ e così via, riuscendo a permutare il vettore a piacere tramite una moltiplicazione.

Se moltiplichiamo la matrice Π_σ per un'altra matrice A, le sue righe saranno permutate secondo σ.

Esempio 2.19. Sia $A = \left[\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right]$, e supponiamo di volerne scambiare le righe, allora la permutazione da usare è quella tale che $\sigma(1) = 2$ e $\sigma(2) = 1$, che corrisponde alla matrice $\left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$. Verifichiamo che lo scambio viene effettuato

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} c & d \\ a & b \end{bmatrix}.$$

Esempio 2.20. Sia

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

e supponiamo di voler scambiare la riga di indice 2 con la riga di indice 4. Allora la permutazione che cerchiamo è tale che $\sigma(1) = 1, \sigma(2) = 4, \sigma(3) = 3$ e $\sigma(4) = 2$. La matrice da creare è

$$\Pi = [e_1 | e_4 | e_3 | e_2] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

Si può verificare che ΠA ha le 2 righe scambiate.

Esempio 2.21. Supponiamo di avere una matrice 4×4 e di voler mandare la riga 2 nella riga 4, la 4 nella 3 e la 3 nella 2. La permutazione che cerchiamo sarà

$$\sigma(1) = 1, \qquad \sigma(2) = 4, \qquad \sigma(3) = 2, \qquad \sigma(4) = 3,$$

e quindi la matrice che effettua lo scambio voluto è

$$\Pi_\sigma = [e_{\sigma(1)} \,|\, e_{\sigma(2)} \,|\, e_{\sigma(3)} \,|\, e_{\sigma(4)}] = [e_1 \,|\, e_4 \,|\, e_2 \,|\, e_3] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

Si verifica che $\Pi_\sigma A$ ha le righe scambiate nel modo voluto.

Apparentemente questo non porta un vantaggio, visto che la stessa permutazione si può operare molto più semplicemente. Tuttavia, vedere la permutazione come un prodotto per matrice è utile per ottenere una fattorizzazione di tipo LU anche nella variante del metodo di Gauss con pivoting. Infatti nella variante del massimo pivot parziale si ottinee una fattorizzazione, non più della matrice A, ma di una sua permutazione.

Teorema 2.22. *Sia U la matrice ottenuta all'ultimo passo del metodo di Gauss con la variante del massimo pivot parziale applicata ad A e L la matrice triangolare inferiore con 1 sulla diagonale ottenuta raccogliendo i rapporti ℓ_{ik} usati per eliminare le righe, allora esiste una matrice di permutazione Π tale che*

$$\Pi A = LU.$$

La matrice Π è associata alla permutazione ottenuta raccogliendo tutti gli scambi di riga effettuati durante l'algoritmo.

Si può dimostrare che le matrici di permutazione sono invertibili. Se disponiamo della fattorizzazione $\Pi A = LU$, e vogliamo risolvere il sistema lineare $Ax = b$, le sue soluzioni coincideranno con quelle del sistema lineare $\Pi Ax = \Pi b$ (cfr. Esercizio 2.23) che può essere riscritto come $LU = \Pi b$ e quindi il problema si risolve permutando il vettore b e risolvendo due sistemi triangolari (come nel caso della fattorizzazione LU standard).

Esercizio 2.23. Siano $A, M \in \mathbb{K}^{n\times n}$ con M invertibile e sia $b \in \mathbb{K}^n$, mostrare che le soluzioni dei sistemi lineari $Ax = b$ e $MAx = Mb$ coincidono. Mostrare inoltre che se M non è invertibile i due sistemi possono avere soluzioni diverse, per opportuni valori di A e b.

Soluzione. Sia x tale che $Ax = b$, allora i due vettori Ax e b coincidono. Se ad essi si applica una matrice M il risultato sarà lo stesso e quindi $MAx = Mb$. Viceversa, se $MAx = Mb$, utilizzando il ragionamento precedente e moltiplicando ambo i membri per M^{-1} si ottiene $M^{-1}MAx = M^{-1}Mb$ cioè $Ax = b$.

Per mostrare che la proprietà non vale se M non è invertibile basta pensare al caso $n = 1$ e $a \neq 0$, $m = 0$ e b qualsiasi, allora l'equazione $ax = b$ ha l'unica soluzione $x = b/a$, mentre l'equazione $max = mb$, che è $0x = 0$ ha infinite soluzioni. □

Calcolo del determinante

Oltre alla soluzione di sistemi lineari, l'algoritmo di Gauss può essere usato per altri problemi dell'algebra lineare numerica, come il calcolo del determinante e il calcolo dell'inversa di una matrice. Descriveremo prima come si può calcolare il determinante di una matrice $A \in \mathbb{K}^{n\times n}$.

L'osservazione chiave è che il passo del metodo di Gauss non cambia il determinante, cioè per ogni $k = 1, \dots, n-1$, si ha che $\det(A_k) = \det(A_{k+1})$. Questo segue dal fatto che la matrice A_{k+1} è ottenuta dalla matrice A_k con operazioni sulle righe che non cambiano il determinante (è noto che se in una matrice $n \times n$ si sostituisce la riga i-esima con la somma della riga i-esima e un multiplo di un'altra riga, il determinante non cambia).

Siccome $\det(A_k) = \det(A_{k+1})$ si ha che $\det(A) = \det(A_1) = \det(A_n) = \det(U)$ e quindi il determinante di A coincide con il determinante di U, ma U è una matrice triangolare e quindi $\det(A) = \det(U) = u_{11} \cdots u_{nn} = \prod_{i=1}^{n} u_{ii}$.

Nel caso del metodo di Gauss con la variante del massimo pivot parziale, oltre alle combinazioni di righe vengono effettuati degli scambi di righe. Siccome ogni scambio di righe cambia il segno del determinante, si avrà che $\det(A) = \det(U)$ se il numero di scambi di righe è pari, e $\det(A) = -\det(U)$ se il numero di scambi di righe è dispari. Più sinteticamente $\det(A) = (-1)^s \det(U)$ dove s è il numero di scambi di righe.

In conclusione l'algoritmo per il calcolo del determinante è il seguente:

- si calcola la matrice triangolare superiore U dalla matrice A con il metodo di Gauss con la variante del massimo pivot parziale contando il numero di scambi di righe s;
- si calcola $\det(A) = (-1)^s \prod_{i=1}^{n} u_{ii}$.

Il costo computazionale di questo algoritmo è dato dal costo del metodo di Gauss, in quanto il prodotto $\prod_i u_{ii}$ si calcola con $n - 1$ ops e quindi è trascurabile.

Sistema lineare con termine noto multiplo e inversa di una matrice

Un problema che capita molto di frequente è quello di risolvere più di un sistema lineare in cui la matrice dei coefficienti è la stessa, ma cambia il termine noto: cioè occorre risolvere i sistemi

$$Ax_1 = b_1, \dots, Ax_s = b_s,$$

dove $b_1, \ldots, b_s$ sono opportuni termini noti. Il modo più immediato ma inefficiente di risolvere tale problema è di risolvere separatamente i sistemi lineari. In questo modo si ha un costo computazionale pari a $\frac{2}{3}sn^3$, dato dal prodotto del costo della soluzione di ciascun sistema lineare tramite il metodo di Gauss, moltiplicato per il numero di sistemi lineari.

Si può considerare un algoritmo migliore. Si parte dalla matrice $[A|b_1|b_2|\cdots|b_s]$ ottenuta incollando alla matrice A tutti i termini noti e si applica l'eliminazione gaussiana (con pivoting) come nel caso in cui ci sia un solo termine noto. Dopo $n-1$ passi si ottiene la matrice $[U|\widetilde{b}_1|\widetilde{b}_2|\cdots|\widetilde{b}_s]$, tale che, per $i = 1, \ldots, s$, il sistema $Ux_i = \widetilde{b}_i$ ha la stessa soluzione del sistema $Ax_i = b_i$.

L'implementazione del metodo è simile a quella del metodo di Gauss, con la differenza che ora non bisogna aggiornare un solo vettore b ma s di essi che metteremo per comodità in una matrice $n \times s$. Nell'implementazione è sufficiente sostituire la riga

```
b(i)=b(i)-ell(i,k)*b(k);
```

con le righe

```
for t=1:s
   b(i,t)=b(i,t)-ell(i,k)*b(k,t);
end
```

Il costo di questo algoritmo è dato dal costo dall'eliminazione gaussiana sulla matrice A, che richiede $\frac{2}{3}n^3$ ops, a cui vanno sommate le operazioni necessarie per modificare i vettori b_i e per risolvere gli s sistemi triangolari. Le operazioni sui vettori b_i al passo k sono $2s$ per ogni $i = k+1, \ldots, n$, per un totale di $\sum_{k=1}^{n-1} 2s(n-k) \approx sn^2$ ops. Il costo della soluzione dei sistemi lineari triangolari è di sn^2 ops, se si usa la sostituzione all'indietro.

In conclusione si possono risolvere s sistemi lineari con la stessa matrice dei coefficienti e con membro destro diverso con un algoritmo che richiede $\frac{2}{3}n^3 + 2sn^2$ ops.

Questo procedimento richiede che i vettori b_i siano tutti noti quando iniziamo l'eliminazione gaussiana, ma invece può succedere di dover risolvere i sistemi in tempi diversi o che addirittura il valore del vettore b_{i+1} dipenda dalla soluzione del sistema $Ax_i = b_i$ (come nel caso del metodo delle potenze inverse per il calcolo degli autovalori di una matrice). Per questo motivo è preferibile usare la fattorizzazione LU di A (o di una sua permutazione, nel caso della variante del pivoting).

I sistemi $Ax_i = b_1, \ldots, Ax_i = b_s$, diventano $LUx_i = \Pi b_1, \ldots, LUx_i = \Pi b_s$, dove Π è la permutazione ottenuta componendo gli scambi del metodo di Gauss con pivoting. Ciascuno di questi ultimi sistemi può essere risolto in due passi, usando la sostituzione $Ux_i = y_i$. In totale si eseguono, come nel caso precedente $\frac{2}{3}n^3 + 2sn^2$ ops.

Per finire, consideriamo il problema del calcolo dell'inversa di una matrice. Ricordiamo che l'inversa di una matrice $A \in \mathbb{K}^{m\times m}$ è una matrice $X \in \mathbb{K}^{m\times m}$ tale che $AX = I$. Se denotiamo con $x_1, \ldots, x_m$, le colonne della matrice X e con $e_1, \ldots, e_m$ le colonne della matrice I, si ottiene

$$AX = A[x_1|x_2|\cdots|x_m] = [Ax_1|Ax_2|\cdots|Ax_n] = [e_1|e_2|\cdots|e_m] = I,$$

che equivale a risolvere gli m sistemi lineari $Ax_1 = e_1, \ldots, Ax_m = e_m$, ciascuno dei quali fornisce una colonna della matrice inversa.

Questo è un problema di termine noto multiplo pertanto il costo totale di questo algoritmo è $\frac{2}{3}m^2 + 2m^3 = \frac{8}{3}m^3$. In realtà si può fare di meglio, poiché i sistemi lineari da risolvere hanno una struttura speciale, infatti i vettori e_i, per $i = 1, \ldots, m$, sono tutti nulli escluso l'elemento i che è 1. Utilizzando questa struttura si riesce a trovare un algoritmo che calcola l'inversa di una matrice e che richiede $2m^3$ ops (cfr. esercizio 2.24). Si noti che è lo stesso costo del prodotto tra due matrici, anche se all'apparenza si tratta di un problema molto più difficile.

Esercizio 2.24. Trovare un algoritmo che calcoli l'inversa di una matrice $m \times m$ che ammette fattorizzazione LU e che abbia un costo computazionale asintotico di $2m^3$ ops.

Esercizio 2.25. Si risolva l'esercizio precedente nel caso di una qualsiasi matrice invertibile, utilizzando il metodo di Gauss con la variante del massimo pivot parziale.

3 Geometria euclidea

Il prodotto scalare e la norma euclidea

Sullo spazio vettoriale reale $\mathbb{R}^n$ è definito il prodotto scalare (euclideo)

$$\langle v, w\rangle := v^T w = v_1 w_1 + \cdots + v_n w_n, \quad v, w \in \mathbb{R}^n.$$

Lo spazio $\mathbb{R}^n$ con il prodotto scalare visto sopra è detto *spazio euclideo* di dimensione n.

Il prodotto scalare euclideo è una funzione da $\mathbb{R}^n \times \mathbb{R}^n$ in $\mathbb{R}$ che verifica alcune interessanti proprietà:

1. $\langle v, v\rangle \geqslant 0$ per $v \in \mathbb{R}^n$ e $\langle v, v\rangle = 0$ se e solo se $v = 0$ (definita positività);
2. $\langle v, w\rangle = \langle w, v\rangle$, per $v, w \in \mathbb{R}^n$ (simmetria);
3. $\langle \alpha v + \beta u, w\rangle = \alpha\langle v, w\rangle + \beta\langle u, w\rangle$ e $\langle v, \alpha u + \beta w\rangle = \alpha\langle v, u\rangle + \beta\langle v, w\rangle$, per ogni $v, u, w \in \mathbb{R}^n$ e $\alpha, \beta \in \mathbb{R}$ (bilinearità).

La dimostrazione delle tre proprietà è abbastanza immediata. Per dimostrare la definita positività si osserva che $\langle v, v\rangle = v_1^2 + v_2^2 + \cdots + v_n^2$ che è una somma di quadrati e quindi è non negativa per ogni $v \in \mathbb{R}^n$, mentre è nulla se e solo se $v_1 = v_2 = \cdots = v_n = 0$ cioè se $v = 0$. Per dimostrare la simmetria basta osservare che

$$v^T w = v_1 w_1 + \cdots + v_n w_n = w_1 v_1 + \cdots + w_n v_n = w^T v.$$

Infine, per dimostrare la bilinearità è sufficiente considerare tre vettori $v, u, w \in \mathbb{R}^n$, e due scalari $a, b \in \mathbb{R}$, da cui

$$\begin{aligned}\langle av + bu, w\rangle = (av_1 + bu_1)w_1 + \cdots + (av_n + bu_n)w_n &= av_1 w_1 + bu_1 w_1 + \cdots + av_n w_n + bu_n w_n \\ &= a(v_1 w_1 + \cdots + v_n w_n) + b(u_1 w_1 + \cdots u_n w_n) = a\langle v, w\rangle + b\langle u, w\rangle.\end{aligned}$$

La linearità rispetto al secondo argomento segue dalla simmetria infatti

$$\langle v, \alpha u + \beta w\rangle = \langle \alpha u + \beta w, v\rangle = \alpha\langle u, v\rangle + \beta\langle w, v\rangle = \alpha\langle v, u\rangle + \beta\langle v, w\rangle.$$

Il prodotto scalare ci permette di definire il concetto di ortogonalità. Diremo che $v \in \mathbb{R}^n$ è *ortogonale* a $w \in \mathbb{R}^n$ se e solo se $v^T w = 0$.

Esempio 3.1. I due vettori $v = \left[\begin{smallmatrix}1\\1\end{smallmatrix}\right]$ e $w = \left[\begin{smallmatrix}1\\-1\end{smallmatrix}\right]$ sono ortogonali, infatti $v^T w = 0$. Si osservi che le due rette generate dai vettori v e w sono perpendicolari.

Dal punto di vista geometrico il prodotto scalare tra due vettori è legato all'ampiezza del più piccolo angolo ϑ che formano le rette generate dai due vettori, in particolare si può definire $\cos\vartheta = \frac{\langle v,w\rangle}{\sqrt{\langle v,v\rangle\langle w,w\rangle}}$, questo coseno è 1 se le due rette coincidono e 0 se sono perpendicolari.

Il prodotto scalare ci permette di definire la *norma euclidea*

$$\|v\| = \sqrt{\langle v, v\rangle} = \sqrt{v^T v} = \sqrt{v_1^2 + \cdots + v_n^2},$$

che può essere intesa come una misura della grandezza o magnitudine o "lunghezza/modulo" di un vettore. Si noti che nel caso di $\mathbb{R}^1$ si ha che $\|v\| = \sqrt{v_1^2} = |v_1|$ e quindi la norma coincide con il valore assoluto, si può dire che la norma euclidea generalizza il valore assoluto.

La norma euclidea verifica le seguenti proprietà:

1. $\|v\| \geqslant 0$ per $v \in \mathbb{R}^n$ e $\|v\| = 0$ se e solo se $v = 0$ (positività);
2. $\|\alpha v\| = |\alpha|\|v\|$, per $v \in \mathbb{R}^n$ e $\alpha \in \mathbb{R}$ (omogeneità);
3. $\|v + w\| \leqslant \|v\| + \|w\|$ per ogni $v, w \in \mathbb{R}^n$ (subadditività).

La positività della norma euclidea segue dalla definizione, l'omogeneità si prova osservando che

$$\|\alpha v\| = \sqrt{\alpha^2 v_1^2 + \cdots + \alpha^2 v_n^2} = \sqrt{\alpha^2}\sqrt{v_1^2 + \cdots + v_n^2} = |\alpha|\|v\|.$$

La dimostrazione della subadditività è semplice ma noiosa e per questo non la riporteremo e rimandiamo il lettore a un qualsiasi testo di algebra lineare.

La norma è un numero che rappresenta la magnitudine di un vettore e ci permette quindi di confrontare diversi vettori. Per ogni vettore non nullo v esistono due suoi multipli di norma 1 che sono

$$w_1 \frac{v}{\|v\|}, \qquad w_2 = -\frac{v}{\|v\|},$$

ciascuno dei quali ha norma 1. Quando si divide un vettore per la sua norma, si dice che è stato *normalizzato*.

Per la norma euclidea si ha $\|v + w\| \leqslant \|v\| + \|w\|$ per ogni $v, w \in \mathbb{R}^n$ e l'uguaglianza vale solo in casi particolari (cfr. Eserzio 3.3), mentre vale sempre la seguente formula, nota in altri contesti come teorema del coseno o teorema di Carnot.

Teorema 3.2. *Siano $v, w \in \mathbb{R}^n$, allora*

$$\|v \pm w\|^2 = \|v\|^2 + \|w\|^2 \pm 2v^T w. \tag{2}$$

Dimostrazione. Dimostriamo solo il caso in cui compare il segno $+$, l'altro caso è del tutto analogo

$$\begin{aligned}\|v + w\|^2 = (v + w)^T(v + w) = (v^T + w^T)(v + w) = v^T v + v^T w + w^T v + w^T w \\ = v^T v + w^T w + 2v^T w = \|v\|^2 + \|w\|^2 + 2v^T w,\end{aligned}$$

dove abbiamo usato $\|u\|^2 = u^T u$ e $v^T w = w^T v$. □

Si noti che quando $v^T w = 0$ cioè se i due vettori v e w sono ortogonali vale l'eguaglianza semplificata $\|v \pm w\|^2 = \|v\|^2 + \|w\|^2$ che è nota come teorema di Pitagora.

Dalla formula (2), sommando le uguaglianze ottenute con il segno più e il segno meno, si ottiene una proprietà fondamentale della norma euclidea: la *legge del parallelogrammo*

$$\|v + w\|^2 + \|v - w\|^2 = 2\|v\|^2 + 2\|w\|^2.$$

La sua interpretazione geometrica è la seguente: "in un parallelogrammo la somma delle lunghezze al quadrato delle diagonali è uguale al doppio della somma delle lunghezze al quadrato dei lati". A questo punto ci dovremmo essere accorti che la geometria ottenuta tramite il prodotto scalare di $\mathbb{R}^n$ è quella a cui siamo abituati fin da piccoli, tuttavia non è l'unica geometria di interesse applicativo e per questo conviene generalizzare il concetto di prodotto scalare, prima però introduciamo il concetto di distanza e topologia euclidea.

Esercizio 3.3. Dire per quali vettori $v, w \in \mathbb{R}^n$ vale $\|v + w\| = \|v\| + \|w\|$.

Esercizio 3.4. Sia $u = \begin{bmatrix} v \\ w \end{bmatrix} \in \mathbb{C}^n$ un vettore partizionato a blocchi tali che $v \in \mathbb{C}^{k_1}$ e $w \in \mathbb{C}^{k_2}$ con $k_1 + k_2 = n$, allora

$$\|u\|^2 = \|v\|^2 + \|w\|^2.$$

Soluzione. Si osserva che

$$\begin{aligned}\|u\|^2 &= |u_1|^2 + |u_2|^2 + \cdots + |u_{k_1}|^2 + |u_{k_1+1}|^2 + |u_{k_1+2}|^2 \cdots + |u_n|^2 \\ &= |v_1|^2 + |v_2|^2 + \cdots + |v_{k_1}|^2 + |w_1|^2 + |w_2|^2 \cdots + |v_{k_2}|^2 \\ &= \|v\|^2 + \|w\|^2.\end{aligned}$$

□

Alla norma euclidea si associa in modo naturale una distanza o metrica euclidea

$$d(v, w) = \|v - w\|,$$

questa distanza verifica le proprietà che ogni buona distanza dovrebbe avere

1. $d(v, w) \geqslant 0$ per $v, w \in \mathbb{R}^n$ e $d(v, w) = 0$ se e solo se $v = w$ (positività);
2. $d(v, w) = d(w, v)$, per $v, w \in \mathbb{R}^n$ (simmetria);
3. $d(v, w) \leqslant d(v, u) + d(w, u)$ per ogni $v, v, u \in \mathbb{R}^n$ (diseguaglianza triangolare).

La distanza euclidea coincide con il concetto di distanza a cui siamo abituati, infatti vale:

$$d(v, w) = \sqrt{(v_1 - w_1)^2 + \cdots + (v_n - w_n)^2}.$$

La proprietà di diseguaglianza triangolare, di fatto, include il seguente principio della geometria euclidea: "in un triangolo la lunghezza di un lato è minore della somma delle lunghezze degli altri due".

Tramite la distanza si possono definire oggetti geometrici come la *palla aperta* di centro $v \in \mathbb{R}^n$ e raggio $R > 0$:

$$B(v, R) = \{z \in \mathbb{R}^n \,:\, d(z, v) < R\}.$$

La *palla chiusa* è ottenuta in modo analogo, ma la condizione è ora che $d(z, v) \leqslant R$.

Esempio 3.5. Nel caso in cui si consideri l'insieme dei numeri reali $\mathbb{R} = \mathbb{R}^1$ la distanza assume un'espressione molto facile, infatti $d(x, y) = \sqrt{(x-y)^2} = |x-y|$ e quindi la distanza tra due numeri reali è il valore assoluto. Una palla di centro x e raggio ρ è l'insieme $B(x, \rho) = \{z \in \mathbb{R} : |z - x| < \rho\}$. Risolvendo la disequazione $|z - x| < \rho$ si ottiene che la palla è l'intervallo aperto $(x - \rho, x + \rho)$.

Esempio 3.6. Consideriamo ora il caso di $\mathbb{R}^2$, dove i vettori vengono identificati con punti del piano euclideo. Per $v = \left[\begin{smallmatrix} x_1 \\ y_1 \end{smallmatrix}\right]$ e $w = \left[\begin{smallmatrix} x_2 \\ y_2 \end{smallmatrix}\right]$ si ottiene la nota formula $d(v, w) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. La palla in questo caso è il disco.

Per finire, notiamo che la metrica euclidea ci permette di definire gli insiemi aperti di $\mathbb{R}^n$. Diciamo che $\mathcal{U} \in \mathbb{R}^n$ è *aperto* se per ogni $x \in \mathbb{R}^n$ esiste una palla aperta di centro x interamente contenuta in $\mathcal{U}$. Gli aperti verificano le seguenti proprietà:

1. L'insieme vuoto e $\mathbb{R}^n$ sono aperti;
2. l'unione di aperti è aperta;
3. l'intersezione finita di aperti è aperta.

Un insieme è detto chiuso in $\mathbb{R}^n$ se il suo complementare è aperto. In questo modo abbiamo definito la *topologia euclidea*, che è quella che si usa nei corsi di analisi per definire la continuità e gli altri concetti di base.

Una funzione $f : \mathbb{R} \to \mathbb{R}$ è continua se $f^{-1}(\mathcal{U})$ è aperto ogni volta che $\mathcal{U}$ è aperto (per definizione $f^{-1}(\mathcal{U}) = \{x \in \mathbb{R} : f(x) \in \mathcal{U}\}$.)

Anche se a prima vista non sembra, questa definizione di continuità coincide con quella data di solito nei corsi di analisi.

Traslazioni e sottospazi affini

Poiché lo spazio euclideo è prima di tutto uno spazio vettoriale, su di esso sono definite le applicazioni lineari. Ricordiamo che a ogni applicazione lineare $\varphi : \mathbb{R}^n \to \mathbb{R}^n$ è associata una matrice una volta fissata la base di $\mathbb{R}^n$, che di solito è quella canonica data dai vettori $e_1, \dots, e_n$.

A parte le applicazioni lineari c'è un'altra categoria di applicazioni molto importanti nello spazio euclideo. Dato un vettore $v_0 \in \mathbb{R}^n$ definiamo la *traslazione di vettore* v_0 la funzione $t_{v_0} : \mathbb{R}^n \to \mathbb{R}^n$ tale che $t_{v_0}(v) = v + v_0$. La traslazione del vettore nullo coincide con l'identità ed è l'unico caso in cui una traslazione è lineare.

Dal punto di vista intuitivo la traslazione è molto semplice perché corrisponde a spostare tutti i punti nella stessa direzione e della stessa quantità.

Le traslazioni si possono comporre come funzioni e vale $t_{v_0} \circ t_{w_0} = t_{v_0+w_0}$, questo segue dal fatto che

$$(t_{v_0} \circ t_{w_0})(v) = t_{v_0}(t_{w_0}(v)) = t_{v_0}(v + w_0) = v + w_0 + v_0 = t_{v_0+w_0}(v).$$

Osserviamo infine che ogni traslazione ammette un'inversa, infatti

$$(t_{v_0} \circ t_{-v_0})(v) = t_{v_0}(v - v_0) = v - v_0 + v_0 = v, \quad (t_{-v_0} \circ t_{v_0})(v) = t_{-v_0}(v + v_0) = v - v_0 + v_0 = v.$$

Questo prova che $t_{v_0}^{-1} = t_{-v_0}$ e che le traslazioni formano un gruppo.

La traslazione ci permette di definire sottospazi più generali dei sottospazi vettoriali: i sottospazi affini. Sia $v_0 \in \mathbb{R}^n$ e V un sottospazio vettoriale di $\mathbb{R}^n$, allora l'insieme $W = t_{v_0}(V) := \{t_{v_0}(v) : v \in V\}$ è detto *sottospazio affine* di $\mathbb{R}^n$. In sostanza un sottospazio affine è "uno spazio vettoriale traslato".

Dato un sottospazio affine W, esiste un sottospazio vettoriale V unico e un vettore v_0 (non unico) tale che $W = t_{v_0}(V)$. Il sottospazio V è detto sottospazio vettoriale parallelo a W. Definiamo la *dimensione di un sottospazio affine* come la dimensione dello spazio vettoriale a esso parallelo.

Esempio 3.7. I sottospazi vettoriali di dimensione 1 di $\mathbb{R}^n$ sono gli insiemi del tipo $\{\alpha v : \alpha \in \mathbb{R}\}$, dove v è un vettore non nullo. Ogni sottospazio vettoriale di dimensione 1 è geometricamente una retta passante per l'origine e rappresenta una direzione, un sottospazio affine di dimensione 1 è una retta dello spazio $\mathbb{R}^n$. Ogni retta di $\mathbb{R}^n$ può essere scritta come l'insieme dei punti $v_0 + \alpha v$ per $\alpha \in \mathbb{R}$ e $v \neq 0$.

Esempio 3.8. I sottospazi vettoriali di dimensione 2 di $\mathbb{R}^n$ sono del tipo $\{\alpha v_1 + \beta v_2 : \alpha, \beta \in \mathbb{R}\}$, dove v_1, v_2 sono due vettori linearmente indipendenti. Un sottospazio affine di dimensione due è ottenuto traslando un sottospazio vettoriale di dimensione 2. Esso è geometricamente un piano e può essere scritto in forma parametrica come l'insieme dei punti $v_0 + \alpha v_1 + \beta v_2$ per $\alpha, \beta \in \mathbb{R}$ e v_1, v_2 linearmente indipendenti.

Esempio 3.9. L'unico sottospazio vettoriale di dimensione 0 di $\mathbb{R}^n$ è il vettore nullo $[0\ 0\ \cdots\ 0]^T$ indicato semplicemente con 0. I corrispondenti sottospazi affini sono i punti. Un punto è quindi un sottospazio affine di dimensione 0, questo fatto era stato catturato da Euclide nell'espressione "il punto non ha parti".

Esempio 3.10. I sottospazi vettoriali di dimensione $n-1$ di $\mathbb{R}^n$ sono detti iperpiani vettoriali e i corrispondenti sottospazi affini sono detti iperpiani affini. Per $n = 3$ gli iperpiani sono i piani, ma per $n = 2$ gli iperpiani sono le rette, mentre per $n = 1$ sono i punti.

Nel piano $\mathbb{R}^2$ una una retta che passa per l'origine è di solito indicata tramite un'equazione del tipo $ax+by = 0$, dove $a, b \in \mathbb{R}$ sono non entrambi nulli e $\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right] \in \mathbb{R}^2$. Essa è detta equazione implicita della retta ed ha il seguente significato geometrico: siano $u = \left[\begin{smallmatrix} a \\ b \end{smallmatrix}\right]$ e $z = \left[\begin{smallmatrix} y \\ x \end{smallmatrix}\right]$, allora l'equazione implicita corrisponde all'equazione $u^T z = 0$ cioè l'insieme dei vettori ortogonali al vettore $\left[\begin{smallmatrix} a \\ b \end{smallmatrix}\right]$ dei coefficienti. Abbiamo quindi due modi di vedere una retta per l'origine, o come l'insieme dei multipli di un vettore non nullo o come l'insieme dei vettori ortogonali a un vettore fissato non nullo. Ad esempio la retta $x + 2y = 0$ è la retta ortogonale al vettore $\left[\begin{smallmatrix} 1 \\ 2 \end{smallmatrix}\right]$.

Nel caso di $\mathbb{R}^3$ l'insieme dei vettori ortogonali a un vettore non nullo fissato $[a\ b\ c]^T$ è un piano e infatti è ben noto che l'equazione $ax+by+cz = 0$ rappresenta un piano che passa per l'origine, mentre per ottenere una retta bisogna mettere a sistema due di queste equazioni i cui coefficienti formano vettori linearmente indipendenti.

Chiamiamo infine *applicazione affine* una applicazione $\varphi : \mathbb{R}^n \to \mathbb{R}^n$ che può essere scritta come $\varphi(v) = Av + v_0$ per qualche $A \in \mathbb{R}^{n\times n}$ e $v_0 \in \mathbb{R}^n$. Una applicazione affine è la composizione di un'applicazione lineare e di una traslazione.

Proposizione 3.11. *Sia W un sottospazio affine di $\mathbb{R}^n$, e sia $\varphi(v) = Av + v_0$ un'applicazione affine, allora l'immagine di W tramite φ, cioè $\{\varphi(w) : w \in W\}$ è un sottospazio affine.*

Dimostrazione. Sia V il sottospazio vettoriale parallelo a W e sia w_0 un vettore tale che $W = t_{w_0}(V)$, allora ogni $w \in W$ può essere scritto come $w = v + w_0$ per qualche $v \in V$. Ora, $\varphi(w) = A(v + w_0) + v_0 = Av + (Aw_0 + v_0)$ e quindi l'immagine di W è ottenuta traslando (del vettore $Aw_0 + v_0$) l'immagine di V tramite A che è un sottospazio vettoriale poiché A è lineare. Quindi W è un sottospazio affine. □

Si osservi che la dimensione dell'immagine di W è minore o uguale a quella di W e può essere anche strettamente minore se il rango di A non è massimo.

C'è un modo per descrivere trasformazioni affini di $\mathbb{R}^n$ tramite applicazioni lineari da $\mathbb{R}^{n+1}$ in $\mathbb{R}^{n+1}$. Data la matrice $A \in \mathbb{R}^{n\times n}$ e il vettore $v_0 \in \mathbb{R}^n$, si costruisce la matrice $\widehat{A} \in \mathbb{R}^{(n+1)\times(n+1)}$ e il vettore $\widehat{v} \in \mathbb{R}^{n+1}$, entrambi a blocchi,

$$\widehat{A} := \begin{bmatrix} A & v_0 \\ 0 & 1 \end{bmatrix}, \qquad \widehat{v} := \begin{bmatrix} v \\ 1 \end{bmatrix}.$$

Ponendo $\widehat{w} = \widehat{A}\widehat{v}$ si nota che

$$\widehat{w} = \widehat{A}\widehat{v} = \begin{bmatrix} A & v_0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ 1 \end{bmatrix} = \begin{bmatrix} Av + v_0 \\ 1 \end{bmatrix},$$

da cui si scopre che le prime n componenti di $\widehat{w}$ contengono il vettore $Av + v_0$. Questo tipo di rappresentazione è comune nei programmi di grafica tridimensionale, dove si utilizza una matrice $\widehat{A}$ che è 4×4 per contenere le informazioni di una trasformazione affine.

Esercizio 3.12. Mostrare che il gruppo delle traslazioni su $\mathbb{R}^n$ è un gruppo abeliano.

Soluzione. È sufficiente provare che dati $w, u \in \mathbb{E}^n$ allora $t_w \circ t_u = t_u \circ t_w$ e infatti

$$(t_w \circ t_u)(v) = t_w(v + u) = v + u + w = v + w + u = t_u(w + u) = (t_u \circ t_w)(v).$$

□