# Aspetti computazionali dei metodi di Gauss e di Householder

Dario A. Bini, Università di Pisa

18 agosto 2019

**Sommario**

Questo modulo didattico contiene una discussione sugli aspetti computazionali dei metodi di Gauss e di Householder per risolvere sistemi lineari. Particolare attenzione viene rivolta a questioni di complessità e di stabilità numerica.

## 1 Introduzione

In questo articolo esaminiamo alcuni aspetti computazionali e implementativi relativi ai metodi di Gauss e di Householder per il calcolo delle fattorizzazioni LU e QR di una matrice $A$ e per la risoluzione di un sistema lineare $Ax = b$. Analizzeremo il costo computazionale e la stabilità numerica di questi metodi. In particolare, per il metodo di Gauss vedremo che il costo è dell'ordine di $\frac{2}{3}n^3$ operazioni aritmetiche, e, da un'analisi all'indietro dell'errore vedremo che non ci sono garanzie di stabilità numerica per tale metodo. Per questo introdurremo le strategie di massimo pivot che permettono di rendere il metodo di eliminazione Gaussiana numericamente affidabile.

Per quanto riguarda il metodo di Householder, un'analisi computazionale ci mostra che il costo computazionale è sensibilmente più alto, cioè dell'ordine di $\frac{4}{3}n^3$ operazioni aritmetiche, e che il metodo non incontra problemi di stabilità numerica anche se applicato senza strategie di massimo pivot. Discuteremo infine su come applicare tali metodi per il calcolo del determinante, dell'inversa e del rango di una matrice.

## 2 Metodo di eliminazione Gaussiana

Nel calcolo della fattorizzazione LU di una matrice $A$ si genera una successione di matrici $A_k$ tali che $A_{k+1} = M_k A_k$, dove $M_k = (m_{i,j}^{(k)})$ con

$$m_{i,j}^{(k)} = \begin{cases} 1 & \text{per } i = j, \\ -\frac{a_{i,k}^{(k)}}{a_{k,k}^{(k)}} & \text{per } j = k, i > k \\ 0 & \text{altrove} \end{cases} \tag{1}$$

e dove

$$A_k = \left[\begin{array}{ccc|ccc} a_{1,1}^{(k)} & \dots & a_{1,k-1}^{(k)} & a_{1,k}^{(k)} & \dots & a_{1,n}^{(k)} \\ & \ddots & \vdots & \vdots & & \vdots \\ 0 & & a_{k-1,k-1}^{(k)} & a_{k-1,k}^{(k)} & \dots & a_{k,n}^{(k)} \\ \hline 0 & \dots & 0 & a_{k,k}^{(k)} & \dots & a_{k+1,n}^{(k)} \\ \vdots & & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & a_{n,k}^{(k)} & \dots & a_{n,n}^{(k)} \end{array}\right] \tag{2}$$

La relazione $A_{k+1} = M_k A_k$ trascritta in componenti si riduce a

$$a_{i,j}^{(k+1)} = \begin{cases} a_{i,j}^{(k)} & \text{se } j < k, \text{ oppure } i \leq k \\ 0 & \text{se } j = k \text{ e } i > k \\ a_{i,j}^{(k)} + m_{i,k}^{(k)} a_{k,j}^{(k)} & \text{se } i > k,\ j > k \end{cases} \tag{3}$$

L'algoritmo per il calcolo della fattorizzazione LU è ricondotto ad applicare le (1) e (3). Se utilizziamo la stessa variabile $\mathtt{A} = (\mathtt{a_{i,j}})$ per memorizzare tutti i valori di $A^{(k)}$ e la stessa variabile $\mathtt{M} = (\mathtt{m_{i,j}})$ per memorizzare i valori degli $m_{i,k}^{(k)}$, l'algoritmo viene sintetizzato dalle relazioni seguenti

**Algoritmo 1**

1. *Per* $\mathtt{k} = 1, \dots, \mathtt{n} - 1$
2. *per* $\mathtt{i} = \mathtt{k} + 1, \dots, \mathtt{n}$
3. $\mathtt{m_{i,k}} = -\mathtt{a_{i,k}}/\mathtt{a_{k,k}}$
4. *per* $\mathtt{j} = \mathtt{k} + 1, \dots, \mathtt{n}$
5. $\mathtt{a_{i,j}} = \mathtt{a_{i,j}} + \mathtt{m_{i,k}}\mathtt{a_{k,j}}$
6. *Fine ciclo* $\mathtt{j}$
7. *Fine ciclo* $\mathtt{i}$
8. *Fine ciclo* $\mathtt{k}$

Nel caso della risoluzione di un sistema lineare $Ax = b$ l'algoritmo può essere modificato aggiungendo il calcolo di $b^{(k+1)} = M_k b^{(k)}$ mediante l'istruzione

$$\mathtt{b_i} = \mathtt{b_i} + \mathtt{m_{i,k}}\mathtt{b_k}$$

che va inserita subito dopo aver calcolato $\mathtt{m_{i,k}}$. In questo modo la soluzione del sistema si ottiene risolvendo il sistema triangolare $A_k x = b^{(k)}$.

Alla fine dell'applicazione dell'algoritmo 1 la matrice $A_{n-1}$ contenuta nella variabile $\mathtt{A}$ conterrà gli elementi della matrice $U$ della fattorizzazione $A = LU$. Non è difficile verificare che la matrice $L$ ha elementi dati da

$$\ell_{i,k} = -m_{i,k} = a_{i,k}^{(k)}/a_{k,k}^{(k)}.$$

Ciò segue dal fatto che

$$L = M_1^{-1} M_2^{-1} \cdots M_{n-1}^{-1}$$

e che $M_k$ differisce dalla matrice identica solo nella colonna $k$-esima in cui gli elementi di indice maggiore di $k$ sono $a_{i,k}^{(k)}/a_{k,k}^{(k)}$.

## 2.1 Costo computazionale

Al passo $k$-esimo dell'algoritmo descritto sopra viene richiesto il calcolo delle quantità $\mathtt{a_{i,j}}$ per $\mathtt{i,j} = \mathtt{k+1},\ldots,\mathtt{n}$ che comporta l'esecuzione di $2(n-k)^2$ operazioni aritmetiche. Altre $n-k$ divisioni sono richieste per il calcolo di $\mathtt{m_{i,k}}$ per $\mathtt{i} = \mathtt{k+1},\ldots,\mathtt{n}$. In totale si arriva ad un costo computazionale di

$$C_n = \sum_{k=1}^{n-1} \big(2(n-k)^2 + (n-k)\big) = \sum_{k=1}^{n-1} (2k^2 + k)$$

operazioni aritmetiche. Usando le formule

$$\sum_{i=1}^{m} i^2 = \frac{m^3}{3} + \frac{m^2}{2} + \frac{m}{6}, \quad \sum_{i=1}^{m} i = \frac{m(m+1)}{2},$$

si arriva al costo

$$C_n \doteq \frac{2}{3} n^3$$

dove $\doteq$ indica l'uguaglianza a meno di termini di ordine inferiore ad $n^3$.

## 2.2 Stabilità numerica e strategie del massimo pivot

Applicando gli strumenti standard dell'analisi degli errori è possibile condurre una analisi all'indietro del metodo di fattorizzazione LU descritto dall'algoritmo 1 si arriva a dimostrare il seguente risultato

**Teorema 1** *Sia $A$ una matrice $n \times n$ con sottomatrici principali di testa fino all'ordine $n-1$ non singolari per cui esiste unica la fattorizzazione $A = LU$. Se $\widetilde{L}$ e $\widetilde{U}$ sono le matrici effettivamente calcolate applicando l'algoritmo 1 in aritmetica floating point con precisione di macchina $u$, allora vale*

$$A + \Delta_A = \widetilde{L}\widetilde{U}$$

*con*

$$|\Delta_A| \leq 2nu(|A| + |\tilde{L}|\,|\tilde{U}|) + O(u^2)$$

*dove si è indicato con $|A|$ la matrice i cui elementi sono $|a_{i,j}|$ e dove la diseguaglianza tra matrici va intesa elemento per elemento. Inoltre, se $\tilde{y}$ è la soluzione*

*del sistema $\widetilde{L}y = b$ effettivamente calcolata in aritmetica floating point mediante sostituzione in avanti e $\tilde{x}$ è il vettore effettivamente calcolato risolvendo il sistema triangolare $\widetilde{U}x = \tilde{y}$ in aritmetica floating point mediante sostituzione all'indietro, vale*

$$(A + \widehat{\Delta}_A)\tilde{x} = b$$

*con*

$$|\widehat{\Delta}_A| \leq 4nu(|A| + |\widetilde{L}|\,|\widetilde{U}|) + O(u^2).$$

Questo risultato ci dice che i valori effettivamente calcolati $\widetilde{L}$ e $\widetilde{U}$ sono i fattori $L$ ed $U$ esatti di una matrice ottenuta perturbando $A$.

L'entità della perturbazione, nella componente proporzionale ad $u$, è lineare in $n$ ma dipende anche dalla grandezza in modulo degli elementi di $L$ e di $U$, di cui $\widetilde{L}$ e $\widetilde{U}$ sono i valori effettivamente calcolati, sulla quale non abbiamo alcun controllo a priori. Per cui se con certi dati gli elementi $a_{i,j}^{(k)}$ delle matrici $A_k$ o gli elementi $m_{i,k} = -a_{i,k}^{(k)}/a_{k,k}^{(k)}$ prendono valori molto elevati in modulo rispetto ai valori iniziali di $A$, allora non dobbiamo aspettarci un buon comportamento numerico del metodo di eliminazione gaussiana. Questo vale sia per il calcolo della fattorizzazione che per la risoluzione del sistema lineare.

Ciò accade ad esempio se per qualche $k$ un elemento pivot $a_{k,k}^{(k)}$ prende valori piccoli in modulo rispetto agli altri elementi $a_{i,k}^{(k)}$. In questo caso i rapporti $m_{i,k} = -a_{i,k}^{(k)}/a_{k,k}^{(k)}$ potrebbero prendere valori molto grandi in modulo.

Un modo per rimuovere questa eventualità consiste nell'effettuare una permutazione di righe alla matrice $A_k$ prima di proseguire col passo di triangolarizzazione. Più precisamente, si sceglie un indice $h$ per cui $|a_{h,k}| \geq |a_{i,k}|$ per ogni altro indice $i \geq k$. Successivamente si permuta la riga $h$-esima con la riga $k$-esima, ottenendo una nuova matrice che ha sempre la stessa struttura triangolare a blocchi (2) ma dove l'elemento pivot ha modulo maggiore o uguale a quello degli elementi sulla stessa colonna sotto la diagonale. Se indichiamo con $P_k$ la matrice di permutazione che effettua lo scambio delle componenti $k$ e $h$, allora il generico passo del metodo così modificato diventa

$$A_{k+1} = M_k P_k A_k.$$

Ora si osserva che se $M_i = I - v^{(i)}e^{(i)T}$, con $i < k$, allora

$$P_k M_i = \widetilde{M}_i P_k,$$

dove $\widetilde{M}_i = I - \tilde{v}^{(i)}e^{(i)T}$ e $\tilde{v}^{(i)} = P_k v^{(i)}$. Per cui, se $k$ è il primo intero in cui si effettua la permutazione delle righe, dalla relazione $A_k = M_{k-1}\cdots M_1 A$ si deduce che

$$A_{k+1} = M_k P_k A_k = M_k P_k M_{k-1}\cdots M_1 A = M_k \widetilde{M}_{k-1}\cdots \widetilde{M}_1 P_k A.$$

In modo analogo si vede induttivamente che, se la permutazione viene applicata ad ogni passo del metodo, vale

$$A_{k+1} = M_k \widetilde{M}_{k-1}\cdots \widetilde{M}_1 P_k P_{k-1}\cdots P_1 A.$$

Cioè, alla fine del procedimento si ottiene

$$A_n = M_{n-1}\widetilde{M}_{n-2}\cdots\widetilde{M}_1(P_{n-1}\cdots P_1)A$$

da cui la fattorizzazione LU della matrice $PA$:

$$PA = LU, \qquad P = P_{n-1}\cdots P_1,$$

con $P$ matrice di permutazione, o equivalentemente la fattorizzazione del tipo PLU: $A = P^T LU$.

Questa strategia, detta del *massimo pivot parziale* ha il vantaggio di contenere la potenziale crescita degli $|m_{i,k}|$ risultando $|m_{i,k}| \leq 1$. Dalla pratica del calcolo questa strategia fornisce un sostanziale miglioramento della stabilità numerica dell'eliminazione gaussiana anche se, come vedremo tra poco, non dà garanzie assolute di stabilità.

Un altro vantaggio di questa strategia è che permette di portare avanti il procedimento anche se nel calcolo si dovesse incontrare un pivot nullo. Infatti, il processo così come lo abbiamo descritto, può subire un arresto solo nel caso in cui al passo $k$-esimo risulta $a_{i,k}^{(k)} = 0$ per $i = k, \ldots, n$. Ma in questo caso non è più necessario svolgere il $k$-esimo passo di triangolarizzazione essendo la $k$-esima colonna della matrice in forma già triangolare. Basta quindi saltare il $k$-esimo passo e continuare il processo dal passo $k+1$. Chiaramente ciò accade solo se la matrice è singolare. A causa degli errori locali generati dall'aritmetica floating point le quantità che in teoria devono risultare nulle, nel calcolo vengono rappresentate da valori di modulo piccolo ma in generale non nullo. È quindi impossibile numericamente stabilire se una quantità calcolata in aritmetica floating point che risulta piccola in modulo rappresenti un valore nullo o un valore piccolo.

Come si è già detto, il fatto che con la strategia del massimo pivot parziale risulti $|m_{i,k}| \leq 1$, migliora in pratica le prestazioni numeriche dell'algoritmo di eliminazione gaussiana. Però questo non garantisce nulla sulla limitatezza uniforme del fattore $U$. Infatti, dalla relazione (1) si deduce che con la strategia del massimo pivot parziale vale

$$|a_{i,j}^{(k+1)}| \leq |a_{i,j}^{(k)}| + |m_{i,k}|\,|a_{k,j}^{(k)}| \leq |a_{i,j}^{(k)}| + |a_{k,j}^{(k)}|.$$

Quindi, se denotiamo con

$$g_k(A) = \max_{i,j} |a_{i,j}^{(k)}| / \max_{i,j} |a_{i,j}| \tag{4}$$

si ha che

$$g_{k+1}(A) \leq 2g_k(A).$$

Ciò conduce alla limitazione

$$g_n(A) = \max_{i,j} |u_{i,j}| / \max_{i,j} |a_{i,j}| \leq 2^{n-1}.$$

Purtroppo questa limitazione superiore ha una crescita esponenziale in $n$ che non promette nulla di buono. Inoltre la maggiorazione data non è lasca come

uno potrebbe sperare, infatti viene raggiunta dal seguente esempio che si riporta nel caso di $n = 5$:

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

Si vede immediatamente che dopo un passo del metodo di eliminazione gaussiana in cui non si effettuano permutazioni poiché l'elemento pivot ha sempre modulo massimo, la matrice $A_1$ risulta

$$A_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 2 \\ 0 & -1 & 1 & 0 & 2 \\ 0 & -1 & -1 & 1 & 2 \\ 0 & -1 & -1 & -1 & 2 \end{bmatrix}$$

Come si può vedere, gli elementi dell'ultima colonna, tranne il primo sono raddoppiati. Procedendo ulteriormente si ha

$$A_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 & 4 \\ 0 & 0 & -1 & 1 & 4 \\ 0 & 0 & -1 & -1 & 4 \end{bmatrix}$$

e alla fine l'elemento di posto $n, n$ vale $a_{n,n}^{(n)} = 2^{n-1}$.

Per fortuna casi come questo non si presentano nelle applicazioni importanti e questo esempio è più un artificio matematico che non un esempio tratto da un problema reale.

Esiste però un rimedio efficace che permette di raggiungere una maggior stabilità numerica. Si tratta della strategia del *massimo pivot totale*. Questa strategia consiste nell'operare nel seguente modo:

- al passo $k$-esimo si selezionano due indici $r, s \geq k$ tali che $|a_{r,s}^{(k)}| \geq |a_{i,j}^{(k)}|$ per ogni $i, j \geq k$;
- si scambiano le righe $k$ ed $r$ e le colonne $k$ ed $s$ di $A^{(k)}$;
- si prosegue con l'eliminazione gaussiana sulla matrice ottenuta.

Procedendo in questo modo si mantiene sempre un elemento pivot che ha modulo maggiore o uguale ai moduli degli elementi di indice $i, j \geq k$. Si può dimostrare che la crescita della quantità $g_k(A)$ definita in (4) ottenuta con la strategia del massimo pivot totale è limitata da

$$g_k(A) \leq \sqrt{n \prod_{j=2}^{n} j^{1/(j-1)}}. \tag{5}$$

Tale funzione ha una crescita molto più contenuta rispetto all'esponenziale. Ad esempio, per $n = 100$ la maggiorazione in (5) è all'incirca 3500.

Si conoscono rarissimi esempi di matrici $n \times n$, con $n = 18, 20, 25$ in cui il fattore $g_n(A)$ raggiunge valori di poco superiori a $n$. Nel resto dei casi noti il valore di $g_n(A)$ non supera $n$. Lo studio della crescita di $g_n(A)$ è un problema aperto non facile da risolvere.

Nel caso di una matrice $A$ singolare, la strategia del massimo pivot totale si arresta quando la matrice $A_k$ ha elementi nulli per $i, j \geq k$. In questo caso possiamo dire che il rango di $A$ è $k-1$. Non solo, ma la fattorizzazione $P_1 A P_2 = LU$ che si trova in questo modo fornisce anche una base per il nucleo (spazio nullo) di $A$ e per l'immagine di $A$. Questo fatto può essere utile in un calcolo in cui non vengono generati errori nelle operazioni. Infatti, in un calcolo in aritmetica floating point, a causa degli errori locali non siamo in grado di dire se un valore calcolato corrisponde a una quantità nulla o ad una di modulo piccolo.

Si osserva che il metodo di fattorizzazione LU può essere applicato anche a matrici di interi generando matrici $A_k$ razionali. Per matrici razionali è possibile mantenere una aritmetica senza errori locali rappresentando ciascun razionale come coppia di interi ed operando sui razionali con una aritmetica intera.

In questo modello di calcolo si incontra però un altro problema. Durante lo svolgimento di ciascuna operazione il numero di cifre con cui vengono rappresentati gli interi che danno il numeratore e il denominatore dei numeri razionali aumenta proporzionalmente al numero di operazioni aritmetiche. Ciò può rendere il calcolo estremamente lento. Si pensi ad esempio alla somma di due numeri del tipo $1/a + 1/b = (a + b)/ab$. Nel caso peggiore il denominatore non si semplifica col numeratore e l'intero $ab$ ha un numero di cifre pari a circa la somma delle cifre di $a$ e quelle di $b$.

Esiste una variante dell'eliminazione gaussiana nota come *metodo di Bareiss* che applicata ad una matrice di elementi interi mantiene il calcolo sugli interi. Questo metodo ha l'inconveniente di produrre interi il cui numero di cifre cresce proporzionalmente alla dimensione della matrice.

Con la variante di Bareiss, o con una aritmetica razionale esatta, è possibile calcolare il rango di una matrice seppur ad un costo computazionale che può essere molto elevato.

Esiste un approccio poco costoso per calcolare il rango di una matrice di interi che ha il prezzo di fornire un risultato non garantito al 100% ma comunque altamente affidabile. Si basa sugli algoritmi che usano la casualità quali gli algoritmi Montecarlo e gli algoritmi LasVegas. Un algoritmo Montecarlo usa al suo interno dei numeri generati in modo pseudocasuale e dà un risultato che, in relazione al numero casuale estratto può essere corretto o meno. La probabilità di errore si conosce ed è generalmente piccola. Un metodo LasVegas procede come un metodo Montecarlo, ma certifica il risultato. Cioè l'output di un metodo LasVegas è il risultato esatto oppure "fallimento".

Ad esempio, si consideri questo metodo per calcolare il rango di una matrice di interi. Si sceglie un numero primo $p$ a caso e si applica l'eliminazione gaussiana con la strategia del pivot totale (per eliminare eventuali pivot nulli) con

aritmetica modulo $p$. Se il processo arriva a termine in $k$ passi si otterrà una fattorizzazione del tipo

$$P_1 A P_2 = L A_k \qquad \mod p$$

dove $A_k$ è triangolare superiore e, o $k = n$ con $\det A_n \neq 0$, oppure $A_k$ ha elementi nulli per $i, j \geq k$. Ne segue che, nel primo caso $A$ è non singolare modulo $p$ e quindi è non singolare. Nel secondo caso il rango di $A$ modulo $p$ è $k-1$. Cioè tutte le sottomatrici di $A$ di dimensione maggiore o uguale a $k$ sono singolari modulo $p$ cioè i loro determinanti sono multipli interi di $p$, mentre c'è una sottomatrice di dimensione $k-1$ che ha determinante diverso da zero e non multiplo di $p$. Ne segue che il rango di $A$ è almeno $k-1$.

Se guardiamo a questa proprietà in termini probabilistici, potremmo scommettere che scegliendo un primo $p$ a caso la probabilità che il rango di $A$ modulo $p$ sia più piccolo del rango $r$ di $A$ sia molto bassa. Infatti affinché ciò accada occorre che gli $\binom{n}{r}$ determinanti di tutte le sottomatrici $r \times r$ di $A$ siano multipli del numero primo $p$ che abbiamo scelto a caso. Questa sembra una eventualità rara.

Se poi vogliamo essere più tranquilli, possiamo scegliere a caso $m \geq 2$ numeri primi $p_i$ per $i = 1, \ldots, m$ e ripetere il calcolo modulo $p_i$, $i = 1, \ldots, m$ ottenendo ranghi $r_i = k_i - 1$, $i = 1, \ldots, m$. In questo modo si deduce che il rango di $A$ è almeno $r = \max_i r_i$. La probabilità che il rango sia maggiore di $r$ è che gli $\binom{n}{r}$ determinanti delle sottomatrici $r \times r$ di $A$ siano multipli interi di tutti i $p_i$, $i = 1, \ldots, m$ che abbiamo scelto a caso. Si presume che tale probabilità sia bassa.

Negli algoritmi Montecarlo e LasVegas la probabilità di fallimento deve essere calcolata a priori per avere chiara l'affidabilità del metodo. Non è ora nostro compito fare questa analisi relativamente all'esempio mostrato.

Un'ultima considerazione sulle strategie di pivot è che il costo della strategia del massimo pivot parziale richiede $n-k+1$ confronti per passo che comportano globalmente una aggiunta di circa $n^2/2$ confronti al costo computazionale. Ciò non altera il costo globale che rimane $\frac{2}{3}n^3$, a meno di termini di ordine inferiore.

Diversa è la situazione della strategia del massimo pivot totale. Infatti, in questo caso, al generico passo $k$ occorre svolgere $(n-k+1)^2$ confronti per individuare l'elemento di massimo modulo in una sottomatrice $(n-k+1) \times (n-k+1)$. Globalmente questo comporta $\frac{1}{3}n^3$ confronti da aggiungere al costo computazionale. Equiparando il costo di un confronto a quello di una operazione aritmetica si arriva ad un costo globale di $n^3$ operazioni e confronti per l'eliminazione gaussiana con la strategia del massimo pivot totale.

## 2.3 Calcolo dell'inversa e del determinante

La conoscenza di una fattorizzazione LU o PLU di una matrice $A$ permette di calcolare agevolmente sia il determinante di $A$ che l'inversa se $\det A \neq 0$. Infatti, dalla relazione $PA = LU$ segue che $\det A = \pm \det U$ dove il segno dipende dalla parità o disparità della permutazione associata a $P$, cioè la parità o disparità del

numero di scambi di righe svolti durante l'esecuzione della strategia di pivot. Inoltre vale $\det U = \prod_{i=1}^{n} u_{i,i}$, per cui il calcolo di $\det A$ costa ancora $\frac{2}{3}n^3$ operazioni.

Per calcolare l'inversa di $A$, occorre risolvere gli $n$ sistemi lineari $LUx = e^{(k)}$, per $k = 1, \ldots, n$, dove al solito $e^{(k)}$ indica il $k$-esimo vettore della base canonica. Ciò è ricondotto alla risoluzione dei due sistemi

$$\begin{aligned} Ly &= e^{(k)} \\ Ux &= y \end{aligned}$$

Il primo sistema è di fatto ricondotto a risolvere un sistema triangolare inferiore $(n-k+1) \times (n-k+1)$ e comporta quindi $(n-k+1)^2$ operazioni aritmetiche a meno di termini di ordine inferiore. Il secondo richiede $n^2$ operazioni. Sommando per $k = 1, \ldots, n$ si arriva ad un costo aggiuntivo di $n^3/3 + n^3$ operazioni aritmetiche che aggiunte al costo della fattorizzazione LU comporta $2n^3$ operazioni aritmetiche.

È naturale chiedersi se il costo di $2n^3$ operazioni è necessario per invertire una matrice $n \times n$ o se è possibile costruire un algoritmo di inversione di costo più basso.

Nel 1969 Volker Strassen ha dimostrato che se esiste un algoritmo per moltiplicare matrici con al più $\gamma n^\theta$ operazioni, dove $\gamma > 0$ e $2 < \theta \leq 3$, allora esiste un algoritmo per invertire una matrice con al più $\gamma' n^\theta$ operazioni e viceversa. Inoltre Strassen ha dato un algoritmo per moltiplicare matrici $n \times n$ con al più $\gamma n^\theta$ con $\theta = \log_2 7 = 2.8073...$. All'inizio degli anni '80 sono stati individuati altri metodi per moltiplicare matrici con costo asintoticamente più basso. L'algoritmo di Coppersmith e Winograd impiega $\gamma n^\theta$ operazioni con $\theta = 2.376...$ La determinazione del valore minimo dell'esponente $\theta$ della complessità del prodotto di matrici e dell'inversione è un problema tuttora aperto.

## 2.4 Implementazione in Octave

Il calcolo della fattorizzazione LU mediante eliminazione gaussiana si realizza facilmente usando il linguaggio di programmazione *Octave*. Il seguente è una prima stesura di una *function* che calcola i fattori L ed U senza strategie di pivot. Poiché non ci sono controlli sulla nullità del pivot, la *function* si arresta se nello svolgimento dei calcoli si incontra un pivot nullo.

```
function [L,U]=flu(A)
% FLU calcola la fattorizzazione LU della matrice A
  n=size(A)(1);
  L=eye(n);
  for k=1:n-1
     for i=k+1:n
        m=A(i,k)/A(k,k);
        L(i,k)=m;
        for j=k+1:n
           A(i,j)=A(i,j)-m*A(k,j);
```

```
        end
      end
    end
    U=triu(A);
```

Una versione più efficiente in ambiente Octave si ottiene riscrivendo in forma vettoriale la parte relativa ai due cicli `for` pilotati dagli indici `i,j`; cioè in modo che le operazioni su vettori e matrici vengono svolte simultaneamente in termini di vettori e non singolarmente sulle componenti. Ad esempio, anziché scrivere

```
for i=k1:k2
   x(i)=y(i)*z(i+1);
end
```

conviene scrivere

```
x(k1:k2)=y(k1:k2).*z(k1+1:k2+1);
```

dove si ricorda che l'operatore .* usato nella precedente istruzione significa il prodotto componente a componente dei due vettori. Cioè il prefisso dato dal punto trasforma un operatore aritmetico tra scalari in un operatore aritmetico tra vettori o matrici dove la sua azione va intesa componente a componente.

Il tempo di esecuzione richiesto da un codice Octave aumenta notevolmente in presenza di cicli `for`. Un programma in cui i cicli `for` che svolgono operazioni sulle singole componenti di vettori o matrici vengono sostituiti da operazioni che coinvolgono in blocco vettori o matrici diventa molto più veloce.

Nel caso dell'algoritmo di fattorizzazione LU si ha il seguente codice più efficiente

```
function [L,U]=vflu(A)
% VFLU calcola la fattorizzazione LU di A in modo vettoriale
  n=size(A)(1);
  L=eye(n);
  for k=1:n-1
      L(k+1:n,k)=A(k+1:n,k)/A(k,k);
      A(k+1:n,k+1:n)=A(k+1:n,k+1:n)-L(k+1:n,k)*A(k,k+1:n);
  end
  U=triu(A);
```

Occorre ricordare che Octave ha la sua implementazione efficiente del calcolo della fattorizzazione LU di una matrice che è realizzata dalla function `lu`.

# 3 Il metodo di Householder

Nel calcolo della fattorizzazione QR mediante matrici di Householder si genera una successione di matrici $A_k$ tali che $A_{k+1} = M_k A_k$ dove $M_k = I - \beta_k u^{(k)} u^{(k)H}$

è matrice di Householder tale che

$$u_i^{(k)} = \begin{cases} 0 & \text{se } i < k \\ a_{k,k}^{(k)}(1 + \frac{(\sum_{i=k}^n |a_{i,k}^{(k)}|^2)^{1/2}}{|a_{k,k}^{(k)}|}) & \text{se } i = k \\ a_{i,k}^{(k)} & \text{se } i > k \end{cases} \tag{6}$$
$$\beta^{(k)} = 2/\sum_{i=k}^n |u_i^{(k)}|^2$$

e dove $A_k$ ha la forma (2).

Le relazioni che forniscono i valori di $a_{i,j}^{(k+1)}$ sono le seguenti

$$a_{i,j}^{(k+1)} = \begin{cases} a_{i,j}^{(k)} & \text{se } j < k, \text{ se } i \leq k \\ 0 & \text{se } j = k, \text{ e } i > k \\ a_{i,j}^{(k)} - \beta^{(k)} u_i^{(k)} \sum_{r=k}^n \bar{u}_r^{(k)} a_{r,j}^{(k)} & \text{se } i \geq k, j > k \end{cases} \tag{7}$$

Nel caso della risoluzione di un sistema lineare $Ax = b$ l'algoritmo può essere modificato aggiungendo il calcolo di $b^{(k+1)} = M_k b^{(k)}$ mediante

$$b_i^{(k+1)} = b_i^{(k)} - \beta_k u_i^{(k)} \sum_{r=k}^n \bar{u}_r^{(k)} b_r^{(k)}, \quad i = k, \ldots, n. \tag{8}$$

## 3.1 Costo computazionale

Per quanto riguarda il costo computazionale del $k$-esimo passo del metodo di Householder si hanno di $2(n-k+1)$ operazioni, a meno di costanti additive, per il calcolo di $u_1^{(k)}$ e per il calcolo di $\beta^{(k)}$. Inoltre l'aggiornamento di $a_{i,j}^{(k+1)}$ richiede $4(n-k+1)^2$ operazioni aritmetiche a meno di termini di ordine inferiore. Sommando su $k$ si arriva a un costo dominato da $\frac{4}{3}n^3$ operazioni. Questo costo non comprende il calcolo degli elementi del fattore $Q = M_1 M_2 \cdots M_{n-1}$, ma prevede di memorizzare la matrice $Q$ in modo implicito attraverso le matrici $M_k$, $k = 1, \ldots, n-1$.

Confrontando col metodo di eliminazione gaussiana per il calcolo della fattorizzazione LU abbiamo quindi un costo doppio.

## 3.2 Stabilità numerica

Il maggior costo computazionale viene bilanciato da una migliore stabilità numerica del metodo di Householder che differentemente dall'eliminazione gaussiana non richiede l'applicazione di strategie di massimo pivot. Vale infatti il seguente risultato.

**Teorema 2** *Si consideri il sistema $Ax = b$ e sia $\widetilde{R}$ la matrice triangolare superiore ottenuta applicando il metodo di Householder dato dalle formule* (6) *e* (7) *per il calcolo della fattorizzazione $A = QR$, in aritmetica floating point con precisione di macchina $u$. Sia inoltre $\tilde{x}$ la soluzione ottenuta risolvendo in aritmetica floating point il sistema $\widetilde{R}x = \tilde{b}^{(n)}$ dove $\tilde{b}^{(n)}$ è il vettore effettivamente*

*calcolato in aritmetica floating point mediante le* (8)*. Allora il vettore* $\tilde{x}$ *risolve il sistema perturbato* $(A + \Delta_A)\widetilde{x} = b + \delta_b$ *dove*

$$\|\Delta_A\|_F \leq u(\gamma n^2 \|A\|_F + n\|\widetilde{R}\|_F) + O(u^2), \qquad \|\delta_b\|_2 \leq \gamma n^2 u \|b\|_2 + O(u^2),$$

*dove* $\gamma$ *è una costante positiva.*

Dalla limitazione superiore data alla norma di Frobenius di $\Delta_A$ risulta che la stabilità del metodo di Householder è legata alla norma di Frobenius di $A$, che è una costante che non dipende dall'algoritmo, e dalla norma di Frobenius di $\widetilde{R}$. Quest'ultima matrice è la matrice effettivamente calcolata al posto di $R$ in aritmetica floating point e quindi la sua norma differisce da quella di $R$ per un termine proporzionale a $u$. Inoltre dalla relazione $A = QR$ e dalle proprietà delle norme di Frobenius si deduce che $\|R\|_F = \|A\|_F$ per cui la limitazione data nel teorema si può riscrivere come

$$\|\Delta_A\|_F \leq \gamma u(n^2 + n)\|A\|_F + O(u^2).$$

Cioè l'entità della perturbazione su $A$ dipende in modo quadratico da $n$. Ciò implica la possibilità di risolvere numericamente sistemi lineari con il metodo di Householder in modo efficace anche per dimensioni grandi di $n$. Occorre dire che nella pratica del calcolo la crescita dell'errore algoritmico in funzione di $n$ avviene in misura sostanzialmente inferiore a quella data nelle maggiorazioni dei teoremi 1 e 2.

# 4 Matrici speciali

Una matrice $A = (a_{i,j})$ si dice matrice a banda di ampiezza $2q + 1$ se $a_{i,j} = 0$ per $|i - j| > q$. Ad esempio, una matrice a banda di ampiezza 3, detta *matrice tridiagonale* ha la forma

$$A = \begin{bmatrix} a_1 & b_1 & & & \\ c_2 & a_2 & b_2 & & \\ & \ddots & \ddots & \ddots & \\ & & c_{n-1} & a_{n-1} & b_n \\ & & & c_n & a_n \end{bmatrix}.$$

È facile dimostrare che se esiste la fattorizzazione LU di $A$ allora i due fattori triangolari $L = (\ell_{i,j})$ ed $U = (u_{i,j})$ sono anch'essi a banda, cioè vale $\ell_{i,j} = u_{i,j} = 0$ per $|i - j| > q$. Inoltre, le matrici $A_k$ generate nel processo di eleiminazione gaussiana sono anch'esse matrici a banda di ampiezza $2k + 1$.

In questo caso la formule (1) si semplifica in $m_{i,k} = -a_{i,k}^{(k)}/a_{k,k}^{(k)}$, $i = k + 1, \ldots, \min(k + q, n)$, mentre la (3) si semplifica in

$$a_{i,j}^{(k+1)} = a_{i,j}^{(k)} + m_{i,k} a_{i,j}^{(k)}, \qquad i, j > k, \quad |i - j| \leq q.$$

In questo modo il costo computazionale del calcolo della fattorizzazione LU è minore o uguale a $(2q^2 + q)n$.

Una analoga proprietà vale per matrici per cui $a_{i,j} = 0$ se $j - i > q_1$ o se $i - j > q_2$ per $1 \leq q_1, q_2 < n$. Infatti anche questa struttura a banda si conserva nelle matrici $A_k$ e nei fattori $L$ ed $U$.

Nel caso del metodo di Householder la struttura si conserva nel fattore $U$ dove però l'ampiezza di banda diventa $q_1 + q_2$. Anche per il metodo di Householder applicato a una matrice a banda il costo computazionale si riduce.

# Riferimenti bibliografici

[1] D. Bini, M. Capovani, O. Menchi. Metodi Numerici per l'Algebra Lineare. Zanichelli, Bologna 1988.