

PARTE SECONDA

PROGRAMMI

CAPITOLO PRIMO

INTRODUZIONE

1. Analisi strutturale mediante il personal computer

Come ampiamente mostrato nella prima parte del testo, la notazione matriciale ivi introdotta ed il conseguente approccio all'analisi strutturale consentono una notevole sintesi espressiva negli sviluppi teorici. Un'applicazione numerica svolta per via manuale risulterebbe però estremamente onerosa anche nel caso di strutture molto semplici. L'utilizzazione di questa notazione è quindi coincisa con la diffusione degli elaboratori elettronici, che hanno liberato l'ingegnere dall'onere di un gran numero di calcoli ripetitivi.

La seconda parte del testo è pertanto dedicata a mostrare in che modo la teoria illustrata in precedenza possa essere concretizzata in programmi di calcolo automatico. L'obiettivo finale è ovviamente anche quello di giungere a programmi sufficientemente completi e di immediata applicazione, che consentano di affrontare tanto le situazioni più banali quanto quelle più complesse che si possono incontrare nella progettazione di edifici. Ma, più ancora che questo, si spera di indirizzare il lettore verso una modalità di approccio ordinata e schematica, la *programmazione strutturata*, essenziale per realizzare con fatica accettabile nuovi programmi o adattare alle proprie esigenze i programmi qui presentati.

Programmare in maniera strutturata vuol dire suddividere il programma in moduli più semplici ed organizzare in ciascuno di essi la sequenza di istruzioni per il calcolatore utilizzando esclusivamente un numero limitato

di strutture logiche elementari. Una descrizione più dettagliata delle strutture logiche e delle caratteristiche della programmazione strutturata può essere ritrovata nel primo volume di questa serie, “Introduzione alla programmazione”, o in testi similari facilmente reperibili in libreria.

L’organizzazione della schema logico sequenziale è solo una delle fasi di stesura di un programma. Quella immediatamente successiva è la *codifica*, cioè la traduzione dello schema in istruzioni che il calcolatore è in grado di riconoscere, cioè che fanno parte di un *linguaggio* disponibile sul computer.

Fino a pochi anni fa, tutte le applicazioni scientifiche più serie venivano implementate su grossi calcolatori e codificate in FORTRAN, linguaggio sviluppato a partire dal 1954 dall’IBM ed organizzato in maniera specifica per le applicazioni matematiche (il nome stesso deriva da FORMula TRANslation, cioè traduzione di formule). La crescente diffusione dei personal computer ha però esteso sempre più l’uso del BASIC (Beginner’s All-purpose Symbolic Instruction Code, cioè linguaggio simbolico di uso generale per principianti), nato nel 1963 con lo scopo di fornire un approccio più semplice alla codifica, ma rimasto forse per tale motivo per lungo tempo un linguaggio “di serie B”, utilizzato solo per programmi di importanza minore.

La continua evoluzione dei linguaggi e la tendenza di ciascuno di essi ad inglobare le caratteristiche migliori degli altri ha portato a versioni del BASIC molto più potenti di quella iniziale e sotto molti aspetti pari o addirittura superiori al FORTRAN e ad un altro linguaggio oggi di moda, il Pascal. Si è quindi deciso di codificare i programmi illustrati nel testo mediante il Quick Basic della Microsoft, compilatore molto veloce (in inglese quick vuol dire svelto) che nelle sue ultime versioni (4.0 e 4.50) unisce alla potenza del linguaggio ed alla compatibilità col tanto usato ma ormai obsoleto GWBASIC notevoli facilitazioni in fase di scrittura e di controllo della correttezza di un programma, oltre alla possibilità di sfruttare a pieno la memoria del computer. Le sue caratteristiche principali sono richiamate nel paragrafo successivo, mentre si rinvia alla bibliografia citata alla fine del libro per una sua più dettagliata conoscenza.

2. Caratteristiche del Quick Basic

2.1. Sequenza delle istruzioni

Quando ci si riferisce all’utilizzazione del BASIC e delle sue prime versioni per personal computer, si sottointende l’uso di un programma in lin-

guaggio macchina, l'interprete BASIC, che crea un ambiente interattivo all'interno del quale si può sia eseguire calcoli in maniera diretta che scrivere ed eseguire programmi. In particolare, il testo del programma è memorizzato in un'apposita area di memoria e non compare sullo schermo a meno che non lo si richiami con il comando LIST. Le istruzioni del programma possono essere inserite e modificate in un ordine indipendente da quello che sarà poi l'ordine di esecuzione. Per tale motivo ogni istruzione deve essere preceduto da un numero che viene utilizzato dal calcolatore per definire la corretta sequenza di esecuzione.

Il Quick Basic è invece sostanzialmente un compilatore, cioè un programma in linguaggio macchina che rende eseguibile un programma già scritto per intero. Contemporaneamente esso è anche un word processor, che consente di scrivere il programma tenendolo sott'occhio sullo schermo come su un foglio di carta (ed è anzi un word processor "intelligente" perché controlla man mano la correttezza sintattica di quanto viene scritto). La sequenza standard di esecuzione è definita dall'ordine col quale le istruzioni compaiono sullo schermo e l'inserimento di nuove istruzioni avviene spostando il cursore nel punto voluto e scrivendo direttamente in tale posizione. Per tale motivo non è più necessario premettere un numero ad ogni istruzione; la numerazione delle righe diviene quindi un fatto facoltativo, ammesso soprattutto per mantenere la compatibilità col GWBASIC. In definitiva, è possibile avere righe senza numerazione, oppure che iniziano con un numero 0, in maniera ancor più generale, con una etichetta, intendendo con tale termine un insieme di caratteri seguito dal simbolo due punti.

Chiunque abbia provato a scrivere programmi in BASIC molto lunghi e a fondere pezzi di differenti programmi si sarà sicuramente scontrato con problemi legati alla numerazione: difficoltà nell'inserire in un listato parecchie righe consecutive, perché i numeri delle linee di programma tra cui inserirle sono troppo vicini tra loro; errori commessi nell'unire due blocchi preparati separatamente, se in questi alcuni numeri si ripetono e quindi alcune istruzioni vengono cancellate da altre. La possibilità di rinunciare alla numerazione, offerta dal Quick Basic, appare quindi come una semplificazione notevole, che conviene senz'altro sfruttare.

Un altro motivo che impone l'uso di numeri di riga nel GWBASIC è la necessità di alterare la sequenza standard mediante istruzioni di salto. La programmazione strutturata non prevede però salti, ma solo l'uso di precise strutture logiche che nel Quick Basic sono codificate con specifiche istruzioni. L'istruzione di salto non è pertanto mai usata nei programmi presentati.

Anche l'uso di routine (istruzione `GOSUB`) richiede nel `GWBasic` la presenza di un numero come indirizzo a cui saltare. Questa istruzione è utilizzata in più casi nei programmi del testo, ma si è preferito individuare l'inizio della routine con una etichetta (che può richiamare subito alla mente il significato del blocco di istruzioni che la costituiscono) anziché con un numero.

2.2. Codifica delle strutture logiche

Il `Quick Basic` è un linguaggio già orientato alla programmazione strutturata e quindi consente di codificare tutte le strutture logiche fondamentali con specifiche istruzioni.

Le strutture selettive “`IF...THEN...`” e “`IF...THEN...ELSE ...`” vengono codificate con le omonime istruzioni, che costituiscono una generalizzazione di quelle già presenti nel `GWBasic` perché consentono di sviluppare i blocchi alternativi su più righe separandoli con le parole chiave `IF`, `THEN`, `ELSE`, `END IF`

```
IF condizione THEN
.....
... blocco di istruzioni 1 ...
.....
ELSE
.....
... blocco di istruzioni 2 ...
.....
END IF
```

La struttura selettiva “`CASE...OF...`” può essere codificata in due maniere distinte. Se la scelta tra i diversi casi è condizionata dal valore di una variabile conviene usare l'istruzione `SELECT CASE`

```
SELECT CASE variabile
CASE valore-1
.....
... blocco di istruzioni 1 ...
.....
CASE valore-2
.....
... blocco di istruzioni 2 ...
.....
CASE valore-3
.....
```

```

... blocco di istruzioni 3 ...
.....
CASE ELSE
.....
... blocco di istruzioni 4 ...
.....
END SELECT

```

Quando le condizioni che controllano la scelta sono più complesse, conviene invece utilizzare una generalizzazione dell'istruzione IF THEN ELSE

```

IF condizione-1 THEN
.....
... blocco di istruzioni 1 ...
.....
ELSEIF condizione-2 THEN
.....
... blocco di istruzioni 2 ...
.....
ELSEIF condizione-3 THEN
.....
... blocco di istruzioni 3 ...
.....
ELSE
.....
... blocco di istruzioni 4 ...
.....
END IF

```

Le strutture di ciclo iterativo possono essere codificate individuando inizio e fine del ciclo con le parole chiave DO e LOOP e ponendo la condizione di controllo del ciclo all'inizio (nel caso della struttura "WHILE...DO...")

```

DO WHILE condizione
.....
.... blocco di istruzioni ....
.....
LOOP

```

o alla fine (nel caso della struttura "REPEAT...UNTIL...")

```

DO
.....
.... blocco di istruzioni ....
.....
LOOP UNTIL condizione

```

La struttura di ciclo enumerativa “FOR...DO...” deve invece essere codificata con le istruzioni FOR...NEXT... che non presentano alcuna variazione rispetto a quelle, già note, del GWBASIC.

Il Quick Basic consente di realizzare anche strutture di ciclo interrotto, inserendo l’istruzione EXIT DO o EXIT FOR all’interno del blocco da ripetere ciclicamente.

2.3. I sottoprogrammi: routine e procedure

Il termine *sottoprogramma* indica genericamente un blocco di istruzioni separato dal programma principale. In fase di esecuzione, quando il calcolatore incontra il comando di salto al sottoprogramma passa ad eseguire le istruzioni in esso contenute per tornare poi, alla fine, all’istruzione immediatamente successiva nella sequenza principale. Questo modo di procedere, nato per evitare di scrivere più volte blocchi di istruzioni che devono essere ripetuti identicamente, è utilizzato nella programmazione strutturata anche, e forse soprattutto, per scomporre il programma in blocchi elementari, indipendentemente dal fatto che questi blocchi vengano utilizzati una o più volte. In tale ottica, l’uso di sottoprogrammi ha una duplice funzione: da un lato, quella di evidenziare l’organizzazione complessiva del programma mostrando in maniera sintetica la sequenza di elaborazioni che esso svolge; dall’altro, quella di creare dei moduli elementari che possono essere riutilizzati nella realizzazione di altri programmi.

Se si passano in rassegna i principali linguaggi disponibili per le applicazioni scientifiche (BASIC, FORTRAN, Pascal), si possono distinguere in essi due tipi di sottoprogrammi. Il primo, tipico del BASIC, è semplicemente un blocco di istruzioni scritto separatamente dal programma principale, ma che ne fa parte a tutti gli effetti ed in particolare ne condivide le variabili. È così possibile utilizzare nel sottoprogramma tutte le informazioni definite nel programma principale ed assegnare in esso nuovi valori a variabili che verranno in seguito utilizzate nella sequenza principale. Il secondo, tipico di FORTRAN e Pascal, costituisce invece, come si suol dire, un *ambiente* distinto da quello del programma principale. Tutti i nomi di variabili in esso utilizzati hanno significato solo all’interno di esso; di conseguenza, se ad esempio nel sottoprogramma si assegna un valore alla variabile A e nel programma principale esiste una variabile con lo stesso nome, quest’ultima è distinta dalla prima ed il suo valore non viene modificato. Lo scambio di informazioni tra l’ambiente principale e quello del sottoprogramma avviene in tal caso mediante *parametri di scambio* che

sono elencati nel richiamare il sottoprogramma.

Nel seguito, in tutti i casi nei quali sarà necessario distinguere i due tipi innanzi descritti si denominerà convenzionalmente il primo *routine* ed il secondo *procedura*.

L'uso di routine, accettabile finché si opera con programmi di media complessità, mostra in maniera evidente i suoi limiti in due situazioni. Innanzitutto, quando si costruisce un programma molto grosso, con numerose variabili: per quanto ci si sforzi, è prima o poi quasi inevitabile decidere di utilizzare per una nuova variabile un nome che è già stato usato per un'altra (con effetti imprevedibili sui risultati e con enormi difficoltà nello scoprire la causa dell'errore). In secondo luogo, quando si inserisce un modulo preparato in precedenza in un programma diverso da quello originario: anche in questo caso è molto elevato il rischio di utilizzare lo stesso nome per variabili che dovrebbero essere distinte.

L'uso di procedure risolve brillantemente questi problemi, ma è meno comodo quando si vuole semplicemente evidenziare la struttura del programma dividendolo in blocchi logici. Poiché questi devono in genere condividere gran parte delle informazioni, il numero di parametri di scambio è in questo caso molto elevato ed è quindi facile commettere errori od omissioni nell'elencarli.

Una soluzione ottimale può essere quella di utilizzare, se consentito dal linguaggio adoperato, entrambi i tipi di sottoprogrammi. In tale ottica, le routine vanno utilizzate per dividere il programma principale nei blocchi logici fondamentali (ingresso dati, elaborazioni iniziali, ecc.) rendendo in tal modo immediatamente evidente la sua organizzazione complessiva. Le procedure servono invece per creare moduli elementari di significato ben preciso, riutilizzabili anche in altri programmi (ad esempio moduli che effettuino le operazioni dell'algebra matriciale) e caratterizzati dall'aver un numero non elevato di parametri di scambio.

Il GWBASIC prevede solo il primo tipo di sottoprogramma. L'inizio del blocco di istruzioni della routine non è contrassegnato in maniera particolare; la sua fine logica è costituita dall'istruzione RETURN (cioè dal comando di ritorno alla sequenza principale), che per chiarezza di lettura andrebbe sempre disposta in coda al blocco. La routine viene richiamata dal programma principale mediante l'istruzione

GOSUB *numero-linea*

dove con *numero-linea* si intende il numero che contrassegna la prima riga

della routine.

Il Quick Basic rende disponibile anche il secondo tipo di sottoprogramma. La serie di istruzioni è in tal caso racchiusa tra le parole SUB e END SUB ed individuata mediante un nome

```
SUB nome-procedura (elenco variabili di scambio)
.....
.... blocco di istruzioni ....
.....
END SUB
```

La procedura viene richiamata dal programma principale (o da altri sottoprogrammi) mediante l'istruzione

```
CALL nome-procedura (valori o variabili di scambio)
```

Per un corretto uso delle procedure, è fondamentale chiarire bene in che modo avviene lo scambio di informazioni tra i due distinti ambienti del programma principale e del sottoprogramma. Come appena indicato, la definizione di una procedura viene effettuata facendo seguire al suo nome l'elenco dei nomi delle variabili di scambio; se tra queste deve essere incluso un intero array, se ne indica il nome seguito da due parentesi, senza indice. Ad esempio

```
SUB CalcolaValori (A, B, N, F(), V$)
```

è l'inizio di una procedura, di nome `CalcolaValori`, dotata di cinque parametri di scambio; i primi tre sono variabili numeriche semplici, il quarto un array numerico, l'ultimo una variabile alfanumerica.

Nel chiamare la procedura con l'istruzione `CALL` si fa seguire al nome un elenco di valori o di variabili (con nomi definiti nel programma chiamante), con un numero di termini e tipo delle informazioni che si devono accordare perfettamente con quanto previsto nella definizione della procedura. La procedura innanzi definita può ad esempio essere richiamata con

```
CALL CalcolaValori(5, (F), J, A(), "testo")
```

Lo scambio di informazioni può avvenire, come si vede, mediante scambio di valori oppure di nomi di variabili. Quando nell'elenco è indicato un valore, questo viene assegnato alla corrispondente variabile del sottoprogramma; nell'esempio mostrato ad A si assegna il valore 5, a V\$ "testo", a B il valore corrente della variabile F del programma chiamante (con (F)

si indica proprio il valore di F). Quando invece si fornisce il nome di una variabile, questa coincide a tutti gli effetti con la corrispondente nell'elenco di definizione, anche se i nomi sono differenti. Nell'esempio, anziché creare una nuova area di memoria per la variabile N della procedura il nome di questa viene collegato all'area già definita per la variabile J del programma chiamante, consentendo così di leggerne o modificarne il valore. Analogamente, l'array F() viene collegato all'array A(), assumendone quindi anche il numero di indici e di elementi.

Nel descrivere le informazioni comunicate ad una procedura si è soliti distinguere tra informazioni fornite ad essa (contenute nelle variabili di ingresso) ed informazioni ottenute da essa (contenute nelle variabili di uscita). Usualmente, nel chiamare una procedura si inseriscono nell'elenco valori numerici per le variabili di ingresso e nomi di variabili per quelle di uscita. Nell'esempio, quindi, A, B e V\$ potrebbero essere le variabili di ingresso, N e F() quelle di uscita.

2.4. Altre istruzioni

Il Quick Basic consente l'uso di numerose istruzioni non presenti nel BASIC standard. Per comodità del lettore, si richiamano in ordine alfabetico le istruzioni utilizzate nei programmi presentati nel testo, accompagnate da una breve spiegazione. Maggiori dettagli andranno ricercati nei manuali forniti dalla Microsoft insieme al linguaggio.

DECLARE SUB *nome-procedura (elenco variabili di scambio)*

Questa istruzione deve essere posta all'inizio del programma principale, per indicare quali sono le procedure in esso utilizzate, quanti parametri ciascuna richiede e quale è il tipo dei parametri.

DIM SHARED *nome-variabile*

La variabile indicata viene considerata come appartenente contemporaneamente al programma principale ed alle procedure, anche se il suo nome non è compreso tra i parametri di scambio.

FREEFILE

Funzione che fornisce il più piccolo valore disponibile come numero di buffer per l'apertura di un file.

LBOUND (*nome-array, n*)

Funzione che fornisce il valore minimo dell'indice n dell'array indicato.

RESUME

Nel caso si sia verificato un errore, questa istruzione indica di riprendere l'esecuzione del programma ripetendo la stessa istruzione per la quale è accaduto l'errore.

RESUME NEXT

Nel caso si sia verificato un errore, questa istruzione indica di riprendere l'esecuzione del programma dalla istruzione immediatamente successiva a quella nella quale l'errore è accaduto.

SWAP *variabile-1, variabile-2*

Scambia tra loro i valori delle due variabili indicate.

TIMER

Funzione che fornisce il numero di secondi trascorsi dalla mezzanotte, in base all'orologio interno del computer.

UBOUND (*nome-array, n*)

Funzione che fornisce il valore massimo dell'indice n dell'array indicato.

3. Contenuto del dischetto

A ciascuna copia del libro sono allegati tre dischetti da 5.25 pollici, formattati a 360 Kbytes per elaboratori con sistema operativo MS-DOS. I dischetti contengono esclusivamente i file di programmi e dati descritti nel testo; in essi non vi è quindi il sistema operativo o il compilatore Quick Basic, che si ritiene siano già in possesso del lettore. Se l'elaboratore non è dotato di disco rigido, occorre pertanto accendere il calcolatore dopo aver disposto il proprio disco di sistema nel drive A. Diventa così immediatamente possibile utilizzare i dischetti forniti insieme al libro mandando in esecuzione le versioni compilate dei programmi. Il testo BASIC originario dei programmi può invece essere utilizzato, per modificarli o per sfruttare per altre applicazioni le procedure in essi contenute, solo dopo aver caricato in memoria il linguaggio Quick Basic.

Nei dischetti è riportato innanzitutto il file `HELP.EXE` che una volta mandato in esecuzione fornisce indicazioni sul contenuto dei dischi. Le informazioni da visualizzare sono memorizzate in file di testo con l'estensione `.HLP`.

I dischetti contengono poi 10 file aventi l'estensione `.BAS`, che costituiscono i programmi e le procedure in Quick Basic descritti nei capitoli successivi:

- `ASTA.BAS` insieme di procedure che forniscono la matrice di rigidità, le azioni di incastro perfetto e le caratteristiche di sollecitazione per diversi tipi di aste e di carichi (cap. 4).
- `DIAGRAM.BAS` programma esemplificativo per il tracciamento dei diagrammi delle caratteristiche di sollecitazione in un portale (cap.4).
- `MATRIX.BAS` insieme di procedure per la effettuazione di operazioni sulle matrici (cap. 2).
- `PROCOM.BAS` insieme di procedure comuni a tutti gli schemi intelaiati considerati (cap. 5).
- `PROGEN.BAS` insieme di procedure caratteristiche del telaio generico, utilizzate anche per il telaio spaziale (cap. 5).
- `PRORET.BAS` insieme di procedure caratteristiche del telaio a maglie rettangolari o trapezie, utilizzate anche per il telaio spaziale (cap. 6).
- `SOLSIST.BAS` insieme di procedure per la risoluzione dei sistemi di equazioni lineari (cap. 3).
- `TELGEN.BAS` programma principale per la risoluzione di uno schema intelaiato piano generico (cap. 5).
- `TELRET.BAS` programma principale per la risoluzione di uno schema intelaiato piano a maglie rettangolari o trapezie (cap.6).
- `TELSPA.BAS` programma principale per la risoluzione di uno schema spaziale di telai piani generici o a maglie rettangolari o trapezie (cap. 7).

I dischetti contengono inoltre, per ciascuno dei programmi innanzi indicati (`DIAGRAM`, `TELGEN`, `TELRET`, `TELSPA`), un file con l'estensione `.MAK` ed uno con l'estensione `.EXE`, che contengono rispettivamente l'elenco dei file che costituiscono il programma e la sua versione compilata, direttamente eseguibile.

Nei dischetti sono infine presenti una serie di file privi di estensione, nei quali sono contenuti i dati richiesti dai programmi per l'elaborazione dei

calcoli riportati come esempio nel testo.

4. Utilizzazione dei programmi

Come noto, nel BASIC esistono più istruzioni che consentono l'assegnazione di valori alle variabili di ingresso, cioè ai dati del problema.

L'istruzione INPUT si presta ad un ingresso dati conversazionale, nel quale i singoli valori siano richiesti espressamente all'utente. Non è però facilmente possibile la correzione di eventuali errori o la riutilizzazione degli stessi dati a distanza di tempo.

L'istruzione READ richiede invece la memorizzazione sequenziale dei dati in linee di programma contraddistinte dalla parola DATA. Essendo in tal modo possibile la correzione ed il riutilizzo dei dati, tale istruzione è preferibile alla precedente per quei problemi che richiedono una notevole mole di informazioni in ingresso. I dati così definiti fanno però parte integrante del programma, e non possono quindi venire modificati se esso è stato compilato.

L'istruzione INPUT# opera in maniera analoga alla READ, ma preleva i valori direttamente da un file sequenziale. Oltre a presentare i vantaggi già evidenziati per tale istruzione, essa è quindi utilizzabile anche in programmi compilati.

È infine possibile usare le istruzioni che consentono una gestione diretta della tastiera e dello schermo per realizzare maschere di ingresso dati che uniscono il vantaggio della interattività con la possibilità di correzione e riutilizzo dei valori.

Tra le quattro alternative elencate, l'ultima è indubbiamente da preferire, ma richiede un maggior onere di programmazione. Si è pertanto fatto ricorso all'istruzione READ nel programma DIAGRAM, che ha solo uno scopo esemplificativo. Nei restanti programmi si è invece utilizzata l'istruzione INPUT#. Per adoperarli occorre quindi aver preparato in precedenza il file dati con un qualsiasi word processor.

Per quanto riguarda l'output di dati e risultati, si è voluto consentire all'utente la scelta dell'unità cui inviare tali informazioni. Nei programmi si è pertanto adoperata l'istruzione di scrittura su file (PRINT#) e si è aggiunta all'inizio la richiesta del nome del file di output. Indicando per esso CON (cioè *console*) i valori di uscita vengono visualizzati sullo schermo. Indicando PRN (*printer*) i valori vengono stampati su carta. Se invece si indica il nome di un file, i risultati vengono memorizzati in esso. Quest'ultima alternativa è particolarmente comoda, perché il contenuto del file può essere successivamente esaminato sullo schermo, oppure inviato alla stampa

con l'istruzione del sistema operativo `TYPE nome-file > PRN` oppure `PRINT nome-file`.

5. Utilizzazione delle procedure e dei programmi in BASIC

Una delle caratteristiche principali del Quick Basic è il consentire che i programmi siano frazionati in più file (o, come si suol dire, in più moduli). Si è così potuto raggruppare in singoli file ciascun insieme di procedure omogenee, come ad esempio quelle relative alle operazioni matriciali, creando in tal modo una volta per tutte una biblioteca di procedure che potrà essere utilizzata in seguito anche per altri programmi. Lavorando nell'ambiente Quick Basic, per unire ad un programma che si sta realizzando un file già esistente è sufficiente caricare in memoria quest'ultimo con l'opzione "Load File..." e salvare poi il programma con l'opzione "Save All..." (nella versione italiana rispettivamente "Carica file..." e "Salva tutto..."). Il Quick Basic crea in tal caso automaticamente un file con l'estensione .MAK che indica quali file costituiscono il programma. Da questo momento in poi, per caricare in memoria tutti i moduli del programma è sufficiente utilizzare nel caricamento l'opzione "Open Program..." (in italiano "Apri programma..."). Nella fase di compilazione viene creato un file rilocabile (con l'estensione .OBJ) per ciascun modulo, e questi file vengono poi riuniti dal Linker in un unico elemento assoluto (con l'estensione .EXE).

CAPITOLO SECONDO

OPERAZIONI MATRICIALI

1. Generalità

Il linguaggio BASIC, sia nella versione standard che in quelle più sofisticate, è privo di istruzioni matriciali. Per sopperire a questa carenza, che rischierebbe di vanificare la sintesi espressiva consentita dall'uso di tale notazione nell'analisi strutturale, è stato realizzato un insieme di procedure che abilitano le istruzioni in questione.

Le procedure, contenute nel file MATRIX.BAS, possono essere divise in tre gruppi:

- procedure che facilitano la costruzione o l'esame di una matrice:
AzzeraMat, CostanteMat, DuplicaMat,
IdentitaMat, InserMat, LimitiMat;

- procedure che implementano le operazioni matriciali:
InversaMat, ProdottoCostMat, ProdottoMat,
SommaMat, SottraeMat, TrasponeMat;

- procedure per leggere o inviare in uscita il contenuto di una matrice:
InputMat, LprintMat, OutputMat,
PrintMat, WriteMat.

La funzione svolta da ciascuna di esse può essere facilmente dedotta dal nome attribuitole, ed è comunque esplicitamente indicata nel listato riportato nel paragrafo 3. In tale listato sono anche evidenziate le variabili di ingresso ed uscita che costituiscono i parametri di scambio tra programma chiamante e procedura.

Tutte le procedure, essendo dedicate all'analisi matriciale, operano su array ad una o due dimensioni. Gli array monodimensionali sono considerati vettori colonna, cioè costituiti da più righe ed un'unica colonna, e sono quindi del tutto equivalenti ad array bidimensionali dimensionati in modo da avere come secondo indice il valore 1.

Ciascun array deve essere dimensionato prima di essere utilizzato come parametro di scambio. Il valore minimo di ogni indice deve essere pari ad 1; valori inferiori (come lo 0, che spesso si ha come default) sono ammessi, ma i corrispondenti elementi dell'array non vengono presi in considerazione. Il valore massimo deve essere definito esattamente, in funzione delle necessità operative; le procedure operano infatti sull'intero array, valutandone automaticamente numero di dimensioni e limiti degli indici.

Si fa infine presente che le procedure sono state scritte in modo da poter effettuare, quando le regole dell'algebra matriciale lo consentano, l'esecuzione delle operazioni operando sulla stessa matrice di partenza. Ad esempio, la trasposta di una matrice rettangolare ha un numero di righe e colonne diverso (e più precisamente scambiato) rispetto alla matrice di partenza. In generale, quindi, si effettuerà la trasposizione di una matrice $A()$ mediante l'istruzione

```
CALL TrasponeMat(A(), B())
```

che mette il risultato nella matrice $B()$ distinta dalla precedente. Se la matrice è quadrata, è però possibile memorizzare la trasposta direttamente al posto della matrice di partenza mediante l'istruzione

```
CALL TrasponeMat(A(), A())
```

nella quale la variabile di uscita coincide con quella di ingresso.

In definitiva, nella realizzazione delle procedure si è preferito privilegiare la generalità (ammettendo l'uso di vettori colonna e la coincidenza tra matrici di ingresso ed uscita) rispetto alla sinteticità ed alla rapidità di esecuzione. Un attento esame del listato mostra le conseguenze di questa scelta: suddivisione di ciascuna procedura in più sottoprocedure, individuate dall'aggiunta di una cifra al nome base, per poter operare con array mono e bidimensionali; introduzione di matrici ausiliarie, per evitare la

perdita di informazioni in ingresso nel caso di coincidenza delle matrici.

2. Procedimenti per l'esecuzione delle operazioni

Gran parte dei procedimenti per l'esecuzione di operazioni su matrici sono molto semplici; essi possono essere facilmente compresi mediante l'esame della relativa codifica e, pertanto, non si ritiene opportuno dilungarsi sulla loro descrizione. Solo a titolo di esempio, si prende in esame la procedura `ProdottoMat`, che effettua il prodotto di due matrici, **A** e **B**, fornendo come risultato la matrice **C**.

Si ricorda dall'algebra delle matrici che, affinché il prodotto possa essere effettuato, è necessario che il numero di colonne della prima matrice sia uguale al numero di righe della seconda (condizione di compatibilità del prodotto); la matrice prodotto ottenuta avrà un numero di righe uguale a quello della prima matrice e un numero di colonne uguale a quello della seconda. Nella prima parte della procedura viene quindi richiamata tre volte la procedura `LimitiMat`, per determinare il numero di dimensioni degli array **A**, **B** e **C** ed i limiti di ciascun indice, e ne viene poi controllata la compatibilità. Si passa infine ad effettuare il prodotto, rinviando a tre distinte procedure (`ProdottoMat1`, `2` e `3`) a seconda del numero di dimensioni dei primi due array. Il valore numerico degli elementi componenti la matrice prodotto **C** è dato dalla relazione

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

ovvero, il generico elemento c_{ij} di **C** è pari alla somma dei prodotti degli elementi della riga i -esima di **A** per gli omologhi elementi della colonna j -esima di **B**. Si noti però nella procedura l'uso intermedio di un array ausiliario **D**, nel quale vengono temporaneamente conservati i valori ottenuti in modo da evitare problemi nel caso che la matrice prodotto sia coincidente con una delle matrici fattore.

L'unica operazione matriciale di una certa complessità è quella di inversione, che può essere effettuata solamente su matrici quadrate. Si definisce matrice inversa di una matrice quadrata **A** di ordine n una matrice **B**, dello stesso ordine, tale che il prodotto $\mathbf{B} \times \mathbf{A}$, ovvero il prodotto $\mathbf{A} \times \mathbf{B}$, risulti uguale alla matrice identità **I**; la matrice inversa viene denotata col simbolo \mathbf{A}^{-1} . Una matrice per la quale sia possibile costruire l'inversa è detta *non singolare*.

La determinazione della matrice inversa può essere effettuata operando contemporaneamente sulla matrice da invertire e sulla matrice identità, mediante combinazioni lineari di righe e colonne tali da trasformare nella matrice identità quella di partenza. Si consideri, infatti, l'uguaglianza

$$\mathbf{A} \times \mathbf{A}^{-1} = \mathbf{I}$$

che scaturisce dalla definizione innanzi richiamata. L'uguaglianza continua a sussistere se si opera contemporaneamente sulle righe di \mathbf{A} ed \mathbf{I} , effettuandone una qualsiasi combinazione lineare (il prodotto $\mathbf{A} \times \mathbf{A}^{-1}$ è un prodotto "righe per colonne", e quindi effettuando una combinazione lineare delle righe del primo fattore, \mathbf{A} , si ottiene la stessa combinazione nelle righe del risultato, \mathbf{I}). In particolare, se con tali operazioni si trasforma la \mathbf{A} nella matrice identità e la \mathbf{I} in una matrice \mathbf{B} si avrà

$$\mathbf{I} \times \mathbf{A}^{-1} = \mathbf{B}$$

ovvero, per la definizione stessa di matrice identità

$$\mathbf{B} = \mathbf{A}^{-1}$$

Dal punto di vista operativo si presentano però due problemi. Il primo è connesso alla necessità di effettuare un controllo sul valore assoluto dei termini diagonali: valori molto piccoli possono dar luogo a rilevanti errori di approssimazione mentre valori nulli vanificano addirittura l'operazione anche quando la matrice è invertibile. Il secondo è legato all'opportunità di effettuare l'operazione di inversione sulla stessa matrice di partenza in modo da dimezzare l'ingombro di memoria richiesto.

Il primo problema può essere risolto associando al procedimento di inversione il riordinamento delle equazioni, ovvero procedendo, in ciascuna fase della trasformazione della matrice \mathbf{A} in matrice identità, all'individuazione della riga e della colonna cui corrisponde l'elemento maggiore in valore assoluto.

Si considerino ad esempio le matrici di ordine n ($n = 3$)

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

dove \mathbf{A} è la matrice da invertire e \mathbf{B} la matrice risultante, inizialmente pari alla matrice identità.

Si suppone che l'elemento maggiore, genericamente indicato con a_{rc} , sia a_{32} , posto nella terza riga ($r = 3$), seconda colonna ($c = 2$). Si inizia il primo passo di trasformazione rendendo unitario tale elemento, ovvero dividendo la corrispondente riga di \mathbf{A} e di \mathbf{B} per il valore dell'elemento stesso. Si ottiene così

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a'_{31} & 1 & a'_{33} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & b'_{33} \end{bmatrix}$$

dove i termini contrassegnati dall'apice sono stati valutati mediante le relazioni

$$a'_{rj} = \frac{a_{rj}}{a_{rc}} \tag{2.1}$$

$$b'_{rj} = \frac{b_{rj}}{a_{rc}}$$

L'apice è usato solo per evidenziare gli elementi modificati, perché i nuovi valori vanno conservati al posto dei precedenti e quindi a'_{rj} diventa, a tutti gli effetti, il nuovo a_{rj} .

Si prosegue quindi sottraendo alle restanti righe un multiplo della riga r , in modo da azzerare tutti gli altri elementi della colonna c ottenendo

$$\mathbf{A} = \begin{bmatrix} a'_{11} & 0 & a'_{13} \\ a'_{21} & 0 & a'_{23} \\ a_{31} & 1 & a_{33} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1 & 0 & b'_{13} \\ 0 & 1 & b'_{23} \\ 0 & 0 & b_{33} \end{bmatrix}$$

Gli elementi della generica riga i , diversa da r , sono valutati mediante le relazioni:

$$\begin{aligned} a'_{ij} &= a_{ij} - a_{ic} a_{rj} \\ b'_{ij} &= b_{ij} - a_{ic} b_{rj} \end{aligned} \tag{2.2}$$

Per la colonna r della matrice \mathbf{B} , che ha l'elemento b_{rr} unitario e tutti gli altri nulli, le espressioni (2.1) e (2.2) possono particolarizzarsi nella forma

$$b'_{rr} = \frac{1}{a_{rc}} \tag{2.3}$$

$$b'_{ir} = -\frac{a_{ic}}{a_{rc}}$$

Si osservi infine che in tutte le relazioni compaiono gli elementi della colonna c di \mathbf{A} , che è quindi necessario memorizzare temporaneamente in un vettore ausiliario \mathbf{C} .

Nel secondo passo di trasformazione l'elemento maggiore va ancora ricercato fra tutti quelli della matrice ad esclusione però degli elementi delle righe e colonne già esaminate (nel nostro caso quelli della terza riga e della seconda colonna).

Nell'ipotesi che l'elemento maggiore sia $l'a_{21}$, ripetendo le due fasi di trasformazione precedentemente definite si ottiene

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & a_{13} \\ 1 & 0 & a_{23} \\ 0 & 1 & a_{33} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1 & b_{12} & b_{13} \\ 0 & b_{22} & b_{23} \\ 0 & b_{32} & b_{33} \end{bmatrix}$$

Nel successivo, ultimo passo di trasformazione l'elemento maggiore della matrice va ricercato escludendo anche quelli della seconda riga e della prima colonna, ed è quindi necessariamente $l'a_{13}$. Ripetendo ancora una volta le fasi di trasformazione si giunge a

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$$

La matrice \mathbf{A} così ottenuta rappresenta, a meno di uno scambio di righe, la matrice identità cercata. Il procedimento si completa quindi con tale scambio, spostando la prima riga in ultima posizione e facendo avanzare di una posizione le altre due righe, fornendo la ricercata matrice inversa

$$\mathbf{A}^{-1} = \mathbf{B} = \begin{bmatrix} b'_{11} & b'_{12} & b'_{13} \\ b'_{21} & b'_{22} & b'_{23} \\ b'_{31} & b'_{32} & b'_{33} \end{bmatrix} = \begin{bmatrix} b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \\ b_{11} & b_{12} & b_{13} \end{bmatrix}$$

Il secondo problema, relativo alla opportunità di eseguire l'operazione di inversione sulla stessa matrice di partenza, può essere risolto riesaminando le operazioni effettuate.

Con riferimento all'esempio mostrato, si può notare innanzitutto che il primo passo di trasformazione ha portato la seconda colonna di \mathbf{A} a valori predefiniti (0 ed 1) che è inutile memorizzare. Contemporaneamente, la particolare forma iniziale di \mathbf{B} ha fatto sì che solo la terza colonna di essa risentisse delle modifiche. Appare quindi possibile sfruttare lo spazio superfluo in \mathbf{A} per conservare le informazioni contenute in \mathbf{B} , spostando la

terza colonna di \mathbf{A} al posto della seconda e memorizzando nel posto così liberato la terza colonna di \mathbf{B}

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{13} & b_{13} \\ a_{21} & a_{23} & b_{23} \\ a_{31} & a_{33} & b_{33} \end{bmatrix}$$

Per mantenere il ricordo degli spostamenti effettuati si utilizza un vettore ausiliario \mathbf{M} , costituito da una riga ed n colonne. Inizialmente si pone in ciascun elemento di esso un numero pari all'indice di colonna, col segno negativo per indicare convenzionalmente che la corrispondente colonna di \mathbf{A} contiene elementi appartenenti ad \mathbf{A}

$$\mathbf{M} = [-1 \quad -2 \quad -3]$$

Lo scambio tra seconda e terza colonna di \mathbf{A} , effettuato nel primo passo di trasformazione, viene registrato in \mathbf{M} scambiando tra loro gli elementi m_2 ed m_3 . Il nuovo m_3 viene inoltre cambiato di segno, per indicare che nella terza colonna sono ora contenuti elementi di \mathbf{B} . \mathbf{M} diventa quindi

$$\mathbf{M} = [-1 \quad -3 \quad 2]$$

Si osservi che l'aver in posizione 3 il valore 2 indica che l'elemento reso unitario in questo passo è a_{32} , informazione necessaria alla fine per ordinare in maniera corretta le righe.

In maniera analoga, nel secondo passo è diventata superflua la prima colonna di \mathbf{A} mentre è diventata significativa una nuova colonna di \mathbf{B} , la seconda. Si può quindi invertire la attuale seconda colonna di \mathbf{A} (ex terza colonna) con la prima e memorizzare al suo posto la seconda colonna di \mathbf{B} , modificando contemporaneamente anche il vettore \mathbf{M}

$$\mathbf{A} = \begin{bmatrix} a_{13} & b_{12} & b_{13} \\ a_{23} & b_{22} & b_{23} \\ a_{33} & b_{32} & b_{33} \end{bmatrix} \quad \mathbf{M} = [-3 \quad 1 \quad 2]$$

Si ricorda che i valori degli elementi della nuova colonna di \mathbf{B} sono stati determinati mediante le (2.3), mentre quelli delle altre colonne di \mathbf{A} e \mathbf{B} sono stati modificati usando le (2.1) e (2.2). Queste ultime trattano allo stesso modo gli elementi di \mathbf{A} e di \mathbf{B} e non richiedono alcuna modifica per l'accorpamento delle due matrici in una.

Con l'ultimo passo di trasformazione si giunge infine a

$$\mathbf{A} = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} \quad \mathbf{M} = \begin{bmatrix} & & \\ 3 & & \\ & 1 & \\ & & 2 \end{bmatrix}$$

Le inversioni di riga necessarie per l'ottenimento della matrice inversa sono chiaramente indicate dal vettore \mathbf{M} : l'attuale prima riga va in terza posizione, la seconda in prima, la terza in seconda.

La complessità operativa del procedimento di inversione qui descritto può essere valutata calcolando il numero di operazioni che esso comporta; in particolare, può ritenersi sufficiente contare solo le moltiplicazioni e divisioni, che richiedono al calcolatore un tempo nettamente maggiore rispetto ad addizioni e sottrazioni. È facile constatare che in ciascun passo di trasformazione si devono applicare le (2.1) e (2.2) agli elementi di $n - 1$ colonne, di \mathbf{A} o di \mathbf{B} , e le (2.3) a quelli di una colonna di \mathbf{B} . Poiché ciascuna colonna contiene n elementi, ed in ogni relazione compare una sola moltiplicazione o divisione, si effettuano n^2 operazioni per passo. Essendo ancora n il numero di passi necessari per la determinazione della matrice inversa, il numero totale di operazioni richiesto è quindi n^3 .

È interessante effettuare un confronto con il numero di operazioni necessario per il prodotto matriciale. Se si moltiplica una matrice $m \times p$ per una $p \times n$, la determinazione di ogni elemento della matrice prodotto richiede p moltiplicazioni. Poiché essa contiene $m \times n$ elementi, il numero totale di operazioni è pari a $p \times m \times n$. Se si moltiplicano tra loro due matrici quadrate di ordine n , occorrono quindi n^3 operazioni. L'onere numerico dell'inversione è pertanto sostanzialmente pari a quello del prodotto.

3. Codifica

```

.                                     MATRIX.BAS
.
.                                     procedure per le operazioni matriciali
.
.
.                                     rev. 09.89
.

' ----- Dichiarazione delle procedure utilizzate -----
DECLARE SUB AzzeraMat1 (A(), NRA!)
DECLARE SUB AzzeraMat2 (A(), NRA!, NCA!)
DECLARE SUB CostanteMat1 (V!, A(), NRA!)
DECLARE SUB CostanteMat2 (V!, A(), NRA!, NCA!)

```

```

DECLARE SUB DuplicaMat (A!(), B!())
DECLARE SUB DuplicaMat1 (A!(), NRA!, B!())
DECLARE SUB DuplicaMat2 (A!(), NRA!, B!())
DECLARE SUB DuplicaMat3 (A!(), NRA!, B!())
DECLARE SUB DuplicaMat4 (A!(), NRA!, NCA!, B!())
DECLARE SUB InputMat1 (A!(), F!, NRA!)
DECLARE SUB InputMat2 (A!(), F!, NRA!, NCA!)
DECLARE SUB InserMat1 (A!(), NRA!, RI!, B!())
DECLARE SUB InserMat2 (A!(), NRA!, RI!, CI!, B!())
DECLARE SUB InserMat3 (A!(), NRA!, NCA!, RI!, CI!, B!())
DECLARE SUB LimitiMat (A!(), DA!, LR!, LC!, NR!, NC!)
DECLARE SUB OutputMat (A!(), F!, F$)
DECLARE SUB OutputMat1 (A!(), F!, F$, NRA!)
DECLARE SUB OutputMat2 (A!(), F!, F$, NRA!, NCA!)
DECLARE SUB ProdottoCostMat1 (K!, A!(), NRA!, B!())
DECLARE SUB ProdottoCostMat2 (K!, A!(), NRA!, NCA!, B!())
DECLARE SUB ProdottoMat1 (A!(), B!(), NRA!, NCB!, C!())
DECLARE SUB ProdottoMat2 (A!(), B!(), NRA!, NCA!, C!())
DECLARE SUB ProdottoMat3 (A!(), B!(), NRA!, NCB!, NCA!, C!())
DECLARE SUB SommaMat1 (A!(), B!(), NRA!, C!())
DECLARE SUB SommaMat2 (A!(), B!(), NRA!, NCA!, C!())
DECLARE SUB SottraeMat1 (A!(), B!(), NRA!, C!())
DECLARE SUB SottraeMat2 (A!(), B!(), NRA!, NCA!, C!())
DECLARE SUB TrasponeMat1 (A!(), NRA!, B!())
DECLARE SUB TrasponeMat2 (A!(), NRA!, NCA!, B!())
DECLARE SUB WriteMat1 (A!(), F!, NRA!)
DECLARE SUB WriteMat2 (A!(), F!, NRA!, NCA!)

```

ErroreLimitiMat:

RESUME NEXT

```

' ===== AzzeraMat =====
'
' Procedura per l'azzeramento di un array monodimensionale o bidimensionale
'
' Variabili di ingresso:
'   A()      array da azzerare
'
' Variabili di uscita:
'   A()      array azzerato
'
' Procedure utilizzate:
'   LimitiMat
' -----
'

```

SUB AzzeraMat (A())

```

' determina il numero di indici e di righe e colonne dell'array A
' controlla che il numero di righe e colonne di A sia corretto
CALL LimitiMat(A(), DA, LRA, LCA, NRA, NCA)
IF LRA > 1 OR LCA > 1 THEN
  CLS
  PRINT "ERRORE 1 in AzzeraMat - limiti inferiori di A maggiori di 1"
  END
END IF

' azzerare l'array
IF DA = 1 THEN

```

```

        CALL AzzeraMat1(A(), NRA)
    ELSE
        CALL AzzeraMat2(A(), NRA, NCA)
    END IF

END SUB

SUB AzzeraMat1 (A(), NRA)

    FOR R = 1 TO NRA
        A(R) = 0
    NEXT R

END SUB

SUB AzzeraMat2 (A(), NRA, NCA)

    FOR R = 1 TO NRA
        FOR C = 1 TO NCA
            A(R, C) = 0
        NEXT C
    NEXT R

END SUB

' ===== CostanteMat =====
'
'   Procedura per la creazione di un array (monodimensionale o bidimensionale)
'   con tutti gli elementi identicamente uguali ad un valore assegnato
'
'   Variabili di ingresso:
'       V           valore da assegnare agli elementi dell'array
'
'   Variabili di uscita:
'       A()        array ottenuto
'
'   Procedure utilizzate:
'       LimitiMat
'
' -----
SUB CostanteMat (V, A())

' determina il numero di indici e di righe e colonne dell'array A
' controlla che il numero di righe e colonne di A sia corretto
CALL LimitiMat(A(), DA, LRA, LCA, NRA, NCA)
IF LRA > 1 OR LCA > 1 THEN
    CLS
    PRINT "ERRORE 1 in CostanteMat - limiti inferiori di A maggiori di 1"
    END
END IF

' assegna il valore agli elementi dell'array
IF DA = 1 THEN
    CALL CostanteMat1(V, A(), NRA)
ELSE
    CALL CostanteMat2(V, A(), NRA, NCA)
END IF

```

END SUB

SUB CostanteMat1 (V, A(), NRA)

```
FOR R = 1 TO NRA
  A(R) = V
NEXT R
```

END SUB

SUB CostanteMat2 (V, A(), NRA, NCA)

```
FOR R = 1 TO NRA
  FOR C = 1 TO NCA
    A(R, C) = V
  NEXT C
NEXT R
```

END SUB

```
' ===== DuplicaMat =====
'
' Procedura per duplicare un array monodimensionale o bidimensionale;
' consente anche di duplicare un vettore colonna in una matrice con
' una sola colonna, e viceversa
'
' Variabili in ingresso:
'   A()   array da duplicare
'
' Variabili in uscita:
'   B()   array duplicato di A()
'
' Procedure utilizzate:
'   LimitiMat
'
' -----
```

SUB DuplicaMat (A(), B())

```
' determina il numero di righe e colonne degli array A e B
' controlla che il numero di righe e colonne sia corretto
CALL LimitiMat(A(), DA, LRA, LCA, NRA, NCA)
CALL LimitiMat(B(), DB, LRB, LCB, NRB, NCB)
IF LRA > 1 OR LCA > 1 OR LRB > 1 OR LCB > 1 THEN
  CLS
  PRINT "ERRORE 1 in DuplicaMat - limiti inferiori di A o B maggiori di 1"
  END
END IF
IF NRA <> NRB OR NCA <> NCB THEN
  CLS
  PRINT "ERRORE 2 in DuplicaMat - numero di righe e colonne di A e B"
  PRINT "                                     non compatibili tra loro"
  END
END IF

' duplica l'array
IF DA = 1 AND DB = 1 THEN
  CALL DuplicaMat1(A(), NRA, B())
ELSEIF DA = 1 AND DB = 2 THEN
```

```

        CALL DuplicaMat2(A(), NRA, B())
    ELSEIF DA = 2 AND DB = 1 THEN
        CALL DuplicaMat3(A(), NRA, B())
    ELSE
        CALL DuplicaMat4(A(), NRA, NCA, B())
    END IF

END SUB

SUB DuplicaMat1 (A(), NRA, B())

    FOR R = 1 TO NRA
        B(R) = A(R)
    NEXT R

END SUB

SUB DuplicaMat2 (A(), NRA, B())

    FOR R = 1 TO NRA
        B(R, 1) = A(R)
    NEXT R

END SUB

SUB DuplicaMat3 (A(), NRA, B())

    FOR R = 1 TO NRA
        B(R) = A(R, 1)
    NEXT R

END SUB

SUB DuplicaMat4 (A(), NRA, NCA, B())

    FOR R = 1 TO NRA
        FOR C = 1 TO NCA
            B(R, C) = A(R, C)
        NEXT C
    NEXT R

END SUB

```

```

* ===== IdentitaMat =====
*
*   Procedura per la creazione di una matrice identita' ovvero una matrice
*   quadrata con gli elementi della diagonale principale uguali ad uno e
*   tutti gli altri nulli
*
*   Variabili di ingresso:
*   =====
*
*   Variabili di uscita:
*   A()      matrice identita'
*
*   Procedure utilizzate:
*   LimitiMat
*
* -----

```

```
SUB IdentitaMat (A())
```

```

' determina il numero di indici e di righe e colonne dell'array A
' controlla che il numero di righe e colonne di A sia corretto
CALL LimitiMat(A(), DA, LRA, LCA, NRA, NCA)
IF LRA > 1 OR LCA > 1 THEN
  CLS
  PRINT "ERRORE 1 in IdentitaMat - limiti inferiori di A maggiori di 1"
  END
END IF
IF NRA <> NCA THEN
  CLS
  PRINT "ERRORE 2 in IdentitaMat - matrice A non quadrata"
  END
END IF

' crea matrice identita'
FOR R = 1 TO NRA
  FOR C = 1 TO NCA
    IF R = C THEN
      A(R, C) = 1
    ELSE
      A(R, C) = 0
    END IF
  NEXT C
NEXT R

```

```
END SUB
```

```

===== InputMat =====
'
' Procedura per l'input di un array monodimensionale o bidimensionale;
' l'array viene letto da un file gia' aperto nel programma principale
'
' Variabili di ingresso:
'   A()   array da leggere
'   F     buffer del file da cui leggere l'array
'
' Variabili di uscita:
'   ==
'
' Procedure utilizzate:
'   LimitiMat
'
-----

```

```
SUB InputMat (A(), F)
```

```

' determina il numero di indici e di righe e colonne dell'array A
' controlla che il numero di righe e colonne di A sia corretto
CALL LimitiMat(A(), DA, LRA, LCA, NRA, NCA)
IF LRA > 1 OR LCA > 1 THEN
  CLS
  PRINT "ERRORE 1 in InputMat - limiti inferiori di A maggiori di 1"
  END
END IF

' effettua l'input dell'array

```

```

    IF DA = 1 THEN
        CALL InputMat1(A(), F, NRA)
    ELSE
        CALL InputMat2(A(), F, NRA, NCA)
    END IF

```

```

END SUB

```

```

SUB InputMat1 (A(), F, NRA)

```

```

    FOR R = 1 TO NRA
        INPUT #F, A(R)
    NEXT R

```

```

END SUB

```

```

SUB InputMat2 (A(), F, NRA, NCA)

```

```

    FOR R = 1 TO NRA
        FOR C = 1 TO NCA
            INPUT #F, A(R, C)
        NEXT C
    NEXT R

```

```

END SUB

```

```

' ===== InserMat =====
'
' Procedura per l'inserimento di un array monodimensionale o bidimensionale
' in un array di dimensioni pari o maggiori
'
' Variabili di ingresso:
'   A()   sottoarray da inserire
'   B()   array in cui inserire
'   RI    riga iniziale
'   CI    colonna iniziale
'
' Variabili di uscita:
'   B()   array risultante
'
' Procedure utilizzate:
'   LimitiMat
' -----

```

```

SUB InserMat (A(), B(), RI, CI)

```

```

' determina il numero di indici e di righe e colonne degli array A e B
' controlla che il numero di righe e colonne di A e B sia corretto
CALL LimitiMat(A(), DA, LRA, LCA, NRA, NCA)
CALL LimitiMat(B(), DB, LRB, LCB, NRB, NCB)
IF LRA > 1 OR LCA > 1 OR LRB > 1 OR LCB > 1 THEN
    CLS
    PRINT "ERRORE 1 in InserMat - limiti inferiori di A o B maggiori di 1"
END
END IF
IF DA > DB THEN
    CLS
    PRINT "ERRORE 2 in InserMat - array di dimensioni incompatibili"

```

```

    END
  END IF
  IF (NRB - RI + 1) < NRA THEN
    CLS
    PRINT "ERRORE 3 in InserMat - sottoarray non inseribile"
  END
  END IF
  IF DA = 2 AND (NCB - CI + 1) < NCA THEN
    CLS
    PRINT "ERRORE 3 in InserMat - sottoarray non inseribile"
  END
  END IF
  IF RI < 1 OR (DB = 2 AND CI < 1) THEN
    CLS
    PRINT "ERRORE 4 in InserMat - riga o colonna iniziale minore di 1"
  END
  END IF

  ' inserisce array
  IF DA = 1 AND DB = 1 THEN
    CALL InserMat1(A(), NRA, RI, B())
  ELSEIF DA = 1 AND DB = 2 THEN
    CALL InserMat2(A(), NRA, RI, CI, B())
  ELSE
    CALL InserMat3(A(), NRA, NCA, RI, CI, B())
  END IF
END SUB

SUB InserMat1 (A(), NRA, RI, B())

  FOR R = 1 TO NRA
    B(RI - 1 + R) = A(R)
  NEXT R

END SUB

SUB InserMat2 (A(), NRA, RI, CI, B())

  FOR R = 1 TO NRA
    B(RI - 1 + R, CI) = A(R)
  NEXT R

END SUB

SUB InserMat3 (A(), NRA, NCA, RI, CI, B())

  FOR R = 1 TO NRA
    FOR C = 1 TO NCA
      B(RI - 1 + R, CI - 1 + C) = A(R, C)
    NEXT C
  NEXT R

END SUB

' ===== InversaMat =====
'
' Procedura per l'inversione di una matrice A()
'
```

```

' Variabili di ingresso:
'   A()           matrice da invertire
'
' Variabili di uscita:
'   A()           matrice invertita
'
' Procedure utilizzate:
'   LimitiMat
'
-----
SUB InversaMat (A())

' determina il numero di indici e di righe e colonne dell'array A
' controlla che il numero di righe e colonne di A sia corretto
CALL LimitiMat(A(), DA, LRA, LCA, NRA, NCA)
IF LRA > 1 OR LCA > 1 THEN
  CLS
  PRINT "ERRORE 1 in InversaMat - limiti inferiori di A maggiori di 1"
  END
END IF
IF NRA <> NCA THEN
  CLS
  PRINT "ERRORE 2 in InversaMat - matrice A non quadrata"
  END
END IF
N = NRA
DIM M(N), C(N)
' prepara il vettore di controllo
FOR I = 1 TO N
  M(I) = -I
NEXT I
'
' ciclo da ripetere N volte
'
FOR K = 1 TO N
  ' individua l'elemento maggiore, in valore assoluto, della matrice
  D = 0
  FOR I = 1 TO N
    IF M(I) < 0 THEN
      FOR J = 1 TO N
        IF M(J) < 0 THEN
          IF ABS(D) < ABS(A(I, J)) THEN
            R = I
            C = J
            D = A(R, C)
          END IF
        END IF
      END IF
    END IF
  NEXT J
  END IF
NEXT I
IF ABS(D) < 1E-30 THEN
  CLS
  PRINT "ERRORE 3 in InversaMat - matrice singolare"
  END
END IF
' scambia le colonne
M(C) = -M(C)
SWAP M(C), M(R)

```

```

FOR I = 1 TO N
  C(I) = A(I, C)
  SWAP A(I, C), A(I, R)
NEXT I
' divide la riga per l'elemento maggiore
FOR J = 1 TO N
  IF J <> R THEN A(R, J) = A(R, J) / D
NEXT J
' azzera i restanti termini della colonna
FOR I = 1 TO N
  IF I <> R THEN
    FOR J = 1 TO N
      IF J <> R THEN A(I, J) = A(I, J) - C(I) * A(R, J)
    NEXT J
  END IF
NEXT I
' memorizza nella colonna i fattori di riduzione
C(R) = -1
FOR I = 1 TO N
  A(I, R) = -C(I) / D
NEXT I
NEXT K
'
' scambia le righe
FOR K = 1 TO N
  FOR I = 1 TO N
    IF M(I) = K THEN
      SWAP M(I), M(K)
      FOR J = 1 TO N
        SWAP A(I, J), A(K, J)
      NEXT J
    END IF
  NEXT I
NEXT K

```

END SUB

```

* ===== LimitiMat =====
*
* Procedura per la determinazione del numero di dimensioni
* e di righe e colonne di un array monodimensionale o bidimensionale
*
* Variabili di ingresso:
*   A()   array in esame
*
* Variabili di uscita:
*   DA   numero di dimensioni dell'array (1 o 2)
*   LR   primo indice di riga
*   LC   primo indice di colonna
*   NR   secondo indice di riga
*   NC   secondo indice di colonna (se l'array e' monodimensionale vale 1)
*
* Procedure utilizzate:
*   ==
*
* Annotazioni:
*   la procedura non controlla se il numero di dimensioni e' superiore a 2
*
* -----

```

```
SUB LimitiMat (A(), DA, LR, LC, NR, NC)
```

```

  LR = LBOUND(A, 1)
  NR = UBOUND(A, 1)
  ON ERROR GOTO ErroreLimitiMat
  LC = LBOUND(A, 2)
  NC = UBOUND(A, 2)
  ON ERROR GOTO 0
  IF LC = 0 AND NC = 0 THEN
    DA = 1
    NC = 1
  ELSE
    DA = 2
  END IF

```

```
END SUB
```

```

===== LprintMat =====
,
,
,   Procedura per la stampa su carta
,   di un array monodimensionale o bidimensionale
,
,   Variabili di ingresso:
,   A()      array da stampare
,   F$      stringa di formattazione per l'output
,           (se F$="" si usa il formato libero)
,
,   Variabili di uscita:
,   =====
,
,   Procedure utilizzate:
,   OutputMat
,
,-----
,

```

```
SUB LprintMat (A(), F$)
```

```

  F = FREEFILE
  OPEN "O", #F, "PRN"
  CALL OutputMat(A(), F, F$)
  CLOSE #F

```

```
END SUB
```

```

===== OutputMat =====
,
,
,   Procedura per l'output di un array monodimensionale o bidimensionale;
,   l'output viene inviato a un file gia' aperto nel programma principale
,
,   Variabili di ingresso:
,   A()      array da stampare
,   F        buffer del file a cui inviare l'output
,   F$      stringa di formattazione per l'output
,           (se F$="" si usa il formato libero)
,
,   Variabili di uscita:
,   =====
,
,

```

```

' Procedure utilizzate:
'   LimitiMat
'
' -----
'

```

```

SUB OutputMat (A(), F, F$)

```

```

' determina il numero di indici e di righe e colonne dell'array A
' controlla che il numero di righe e colonne di A sia corretto
CALL LimitiMat(A(), DA, LRA, LCA, NRA, NCA)
IF LRA > 1 OR LCA > 1 THEN
  CLS
  PRINT "ERRORE 1 in OutputMat - limiti inferiori di A maggiori di 1"
  END
END IF

' effettua l'output dell'array
IF DA = 1 THEN
  CALL OutputMat1(A(), F, F$, NRA)
ELSE
  CALL OutputMat2(A(), F, F$, NRA, NCA)
END IF

```

```

END SUB

```

```

SUB OutputMat1 (A(), F, F$, NRA)

```

```

FOR R = 1 TO NRA
  IF F$ = "" THEN
    PRINT #F, A(R)
  ELSE
    PRINT #F, USING F$; A(R)
  END IF
NEXT R

```

```

END SUB

```

```

SUB OutputMat2 (A(), F, F$, NRA, NCA)

```

```

FOR R = 1 TO NRA
  FOR C = 1 TO NCA
    IF F$ = "" THEN
      PRINT #F, A(R, C),
    ELSE
      PRINT #F, USING F$; A(R, C),
    END IF
  NEXT C
  PRINT #F,
NEXT R

```

```

END SUB

```

```

' ===== PrintMat =====
'
' Procedura per la stampa su schermo
' di un array monodimensionale o bidimensionale
'
' Variabili di ingresso:
'   A()      array da stampare

```

```

'      F$      stringa di formattazione per l'output
'              (se F$="" si usa il formato libero)
'
' Variabili di uscita:
'      ==
'
' Procedure utilizzate:
'      OutputMat
'
'-----
SUB PrintMat (A(), F$)

    LOCATE 25, 1
    F = FREEFILE
    OPEN "O", #F, "CON"
    CALL OutputMat(A(), F, F$)
    CLOSE #F
    PRINT

END SUB

'===== ProdottoCostMat =====
'
' Procedura per il prodotto di una costante per un array
'
' Variabili di ingresso:
'      K      costante da moltiplicare
'      A()   array da moltiplicare
'
' Variabili di uscita:
'      B()   array prodotto
'
' Procedure utilizzate:
'      DuplicaMat
'      LimitiMat
'
'-----
SUB ProdottoCostMat (K, A(), B())

' determina il numero di indici e di righe e colonne degli array A e B
' controlla che il numero di righe e colonne di A e B sia corretto
CALL LimitiMat(A(), DA, LRA, LCA, NRA, NCA)
CALL LimitiMat(B(), DB, LRB, LCB, NRB, NCB)
IF LRA > 1 OR LCA > 1 OR LRB > 1 OR LCB > 1 THEN
    CLS
    PRINT "ERRORE 1 in ProdottoCostMat - limiti inferiori di A o B maggiori di 1"
    END
END IF
IF NRA <> NRB OR NCA <> NCB THEN
    CLS
    PRINT "ERRORE 2 in ProdottoCostMat - numero di righe e colonne di A e B"
    PRINT "                                non compatibili tra loro"
    END
END IF

' esegue il prodotto
IF DA = 1 AND DB = 1 THEN

```

```

    CALL ProdottoCostMat1(K, A(), NRA, B())
ELSEIF DA = 2 AND DB = 1 THEN
    DIM E(NRA)
    CALL DuplicaMat(A(), E())
    CALL ProdottoCostMat1(K, E(), NRA, B())
ELSEIF DA = 1 AND DB = 2 THEN
    DIM E(NRA)
    CALL ProdottoCostMat1(K, A(), NRA, E())
    CALL DuplicaMat(E(), B())
ELSE
    CALL ProdottoCostMat2(K, A(), NRA, NCA, B())
END IF

END SUB

SUB ProdottoCostMat1 (K, A(), NRA, B())

    FOR R = 1 TO NRA
        B(R) = K * A(R)
    NEXT R

END SUB

SUB ProdottoCostMat2 (K, A(), NRA, NCA, B())

    FOR R = 1 TO NRA
        FOR C = 1 TO NCA
            B(R, C) = K * A(R, C)
        NEXT C
    NEXT R

END SUB

' ===== ProdottoMat =====
'
' Procedura per il prodotto tra due array (righe per colonne)
'
' Variabili di ingresso:
'   A()      primo array da moltiplicare
'   B()      secondo array da moltiplicare
'
' Variabili di uscita:
'   C()      array risultante
'
' Procedure utilizzate:
'   DuplicaMat
'   LimitiMat
' -----
SUB ProdottoMat (A(), B(), C())

    ' determina il numero di indici e di righe e colonne degli array A, B e C
    ' controlla che il numero di righe e colonne di A, B e C siano corretti
    CALL LimitiMat(A(), DA, LRA, LCA, NRA, NCA)
    CALL LimitiMat(B(), DB, LRB, LCB, NRB, NCB)
    CALL LimitiMat(C(), DC, LRC, LCC, NRC, NCC)
    IF LRA > 1 OR LCA > 1 OR LRB > 1 OR LCB > 1 OR LRC > 1 OR LCC > 1 THEN
        CLS
    
```

```

PRINT "ERRORE 1 in ProdottoMat - limiti inferiori di A, B o C maggiori di 1"
END
END IF
IF NCA <> NRB OR NRA <> NRC OR NCB <> NCC THEN
CLS
PRINT "ERRORE 2 in ProdottoMat - numero di righe e colonne di A, B e C"
PRINT "
non compatibili tra loro"
END
END IF

' calcola il prodotto
IF DA = 1 AND DB = 2 THEN
CALL ProdottoMat1(A(), B(), NRA, NCB, C())
ELSEIF DA = 2 AND DB = 1 THEN
CALL ProdottoMat2(A(), B(), NRA, NCA, C())
ELSE
CALL ProdottoMat3(A(), B(), NRA, NCB, NCA, C())
END IF

END SUB

SUB ProdottoMat1 (A(), B(), NRA, NCB, C())

DIM D(NRA, NCB)
FOR R = 1 TO NRA
FOR C = 1 TO NCB
D(R, C) = A(R) * B(1, C)
NEXT C
NEXT R
CALL DuplicaMat(D(), C())

END SUB

SUB ProdottoMat2 (A(), B(), NRA, NCA, C())

DIM D(NRA)
FOR R = 1 TO NRA
FOR K = 1 TO NCA
D(R) = D(R) + A(R, K) * B(K)
NEXT K
NEXT R
CALL DuplicaMat(D(), C())

END SUB

SUB ProdottoMat3 (A(), B(), NRA, NCB, NCA, C())

DIM D(NRA, NCB)
FOR R = 1 TO NRA
FOR C = 1 TO NCB
FOR K = 1 TO NCA
D(R, C) = D(R, C) + A(R, K) * B(K, C)
NEXT K
NEXT C
NEXT R
CALL DuplicaMat(D(), C())

END SUB

```

```

===== SommaMat =====
'
' Procedura per la somma di due array monodimensionali o bidimensionali
'
' Variabili di ingresso:
'   A()  primo array da sommare
'   B()  secondo array da sommare
'
' Variabili di uscita:
'   C()  array risultante
'
' Procedure utilizzate:
'   DuplicaMat
'   LimitiMat
'
-----

```

```
SUB SommaMat (A(), B(), C())
```

```

' determina il numero di indici e di righe e colonne degli array A, B e C
' controlla che il numero di righe e colonne di A, B e C sia corretto
CALL LimitiMat(A(), DA, LRA, LCA, NRA, NCA)
CALL LimitiMat(B(), DB, LRB, LCB, NRB, NCB)
CALL LimitiMat(C(), DC, LRC, LCC, NRC, NCC)
IF LRA > 1 OR LCA > 1 OR LRB > 1 OR LCB > 1 OR LRC > 1 OR LCC > 1 THEN
  CLS
  PRINT "ERRORE 1 in SommaMat - limiti inferiori di A, B o C maggiori di 1"
  END
END IF
IF NCA <> NCB OR NCB <> NCC OR NRA <> NRB OR NRB <> NRC THEN
  CLS
  PRINT "ERRORE 2 in SommaMat - numero di righe e colonne di A, B e C"
  PRINT "          non compatibili tra loro"
  END
END IF

```

```

' effettua la somma dei due array
IF DA = 1 AND DB = 1 THEN
  CALL SommaMat1(A(), B(), NRA, C())
ELSEIF DA = 2 AND DB = 1 THEN
  DIM E(NRA)
  CALL DuplicaMat(A(), E())
  CALL SommaMat1(E(), B(), NRA, C())
ELSEIF DA = 1 AND DB = 2 THEN
  DIM E(NRA)
  CALL DuplicaMat(B(), E())
  CALL SommaMat1(A(), E(), NRA, C())
ELSE
  CALL SommaMat2(A(), B(), NRA, NCA, C())
END IF

```

```
END SUB
```

```
SUB SommaMat1 (A(), B(), NRA, C())
```

```

DIM D(NRA)
FOR R = 1 TO NRA
  D(R) = A(R) + B(R)
NEXT R

```

```

CALL DuplicaMat(D(), C())

END SUB

SUB SommaMat2 (A(), B(), NRA, NCA, C())

  FOR R = 1 TO NRA
    FOR C = 1 TO NCA
      C(R, C) = A(R, C) + B(R, C)
    NEXT C
  NEXT R

END SUB

```

```

' ===== SottraeMat =====
'
' Procedura per la sottrazione di due array
'
' Variabili di ingresso:
'   A()      primo array
'   B()      secondo array da sottrarre al primo
'
' Variabili di uscita:
'   C()      array risultante
'
' Procedure utilizzate:
'   DuplicaMat
'   LimitiMat
'
' -----
SUB SottraeMat (A(), B(), C())

```

```

' determina il numero di indici e di righe e colonne degli array A, B e C
' controlla che il numero di righe e colonne di A, B e C siano corrette
CALL LimitiMat(A(), DA, LRA, LCA, NRA, NCA)
CALL LimitiMat(B(), DB, LRB, LCB, NRB, NCB)
CALL LimitiMat(C(), DC, LRC, LCC, NRC, NCC)
IF LRA > 1 OR LCA > 1 OR LRB > 1 OR LCB > 1 OR LRC > 1 OR LCC > 1 THEN
  CLS
  PRINT "ERRORE 1 in SottraeMat - limiti inferiori di A, B o C maggiori di 1"
  END
END IF
IF NCA <> NCB OR NCB <> NCC OR NRA <> NRB OR NRB <> NRC THEN
  CLS
  PRINT "ERRORE 2 in SottraeMat - numero di righe e colonne di A, B e C"
  PRINT "
          non compatibili tra loro"
  END
END IF

' effettua la sottrazione
IF DA = 1 AND DB = 1 THEN
  CALL SottraeMat1(A(), B(), NRA, C())
ELSEIF DA = 2 AND DB = 1 THEN
  DIM E(NRA)
  CALL DuplicaMat(A(), E())
  CALL SottraeMat1(E(), B(), NRA, C())
ELSEIF DA = 1 AND DB = 2 THEN
  DIM E(NRA)

```



```

        END
    END IF

    ' effettua la trasposizione
    IF DA = 1 THEN
        CALL TrasponeMat1(A(), NRA, B())
    ELSE
        CALL TrasponeMat2(A(), NRA, NRA, B())
    END IF

END SUB

SUB TrasponeMat1 (A(), NRA, B())

    DIM D(1, NRA)
    FOR R = 1 TO NRA
        D(1, R) = A(R)
    NEXT R
    CALL DuplicaMat(D(), B())

END SUB

SUB TrasponeMat2 (A(), NRA, NCA, B())

    DIM D(NCA, NRA)
    FOR R = 1 TO NRA
        FOR C = 1 TO NCA
            D(C, R) = A(R, C)
        NEXT C
    NEXT R
    CALL DuplicaMat(D(), B())

END SUB

' ===== WriteMat =====
'
' Procedura per memorizzare su un file i valori degli elementi
' di un array monodimensionale o bidimensionale;
' l'array viene scritto su un file gia' aperto nel programma principale
'
' Variabili di ingresso:
'   A()      array da inviare al file
'   F        buffer del file a cui inviare l'array
'
' Variabili di uscita:
'   =====
'
' Procedure utilizzate:
'   LimitiMat
'
' -----
SUB WriteMat (A(), F)

    ' determina il numero di indici e di righe e colonne dell'array A
    ' controlla che il numero di righe e colonne di A sia corretto
    CALL LimitiMat(A(), DA, LRA, LCA, NRA, NCA)
    IF LRA > 1 OR LCA > 1 THEN
        CLS
    
```

```

        PRINT "ERRORE 1 in WriteMat - limiti inferiori di A maggiori di 1"
        END
    END IF

    ' effettua l'output su file dell'array
    IF DA = 1 THEN
        CALL WriteMat1(A(), F, NRA)
    ELSE
        CALL WriteMat2(A(), F, NRA, NCA)
    END IF

END SUB

SUB WriteMat1 (A(), F, NRA)

    FOR R = 1 TO NRA
        WRITE #F, A(R)
    NEXT R

END SUB

SUB WriteMat2 (A(), F, NRA, NCA)

    FOR R = 1 TO NRA
        FOR C = 1 TO NCA
            WRITE #F, A(R, C)
        NEXT C
    NEXT R

END SUB

```

4. Esempio

Un esempio di programma principale che utilizza le procedure qui descritte è fornito dal programma per la risoluzione di un sistema di equazioni, sviluppato solo per finalità didattiche (perché il problema viene ripreso nel capitolo successivo) e listato nel seguito. Esso prevede innanzitutto la lettura dei dati (numero di equazioni N , matrice dei coefficienti delle incognite A , termini noti B) conservati in apposite linee contraddistinte dall'istruzione `DATA`. Dopo aver stampato i dati, la matrice A viene duplicata in $A1$ e quest'ultima viene invertita. Per controllo, si effettua il prodotto $A \times A1$ che deve fornire, a meno degli errori di arrotondamento, la matrice identità (si è effettuata la duplicazione prima dell'inversione di A proprio per non perdere la matrice di partenza e per poter effettuare il controllo). Il vettore risultato X viene infine valutato moltiplicando la matrice inversa per il vettore termini noti.

RISOLUZIONE DI UN SISTEMA DI EQUAZIONI
mediante l'inversione della matrice dei coefficienti

```

DECLARE SUB DuplicaMat (A!(), B!())
DECLARE SUB InversaMat (A!())
DECLARE SUB LprintMat (A!(), F$)
DECLARE SUB ProdottoMat (A!(), B!(), C!())

OPTION BASE 1

READ N
DIM A(N, N), A1(N, N), B(N), X(N)

' lettura della matrice dei coefficienti A e del vettore termine noto B
FOR I = 1 TO N
  FOR J = 1 TO N
    READ A(I, J)
  NEXT J
NEXT I
FOR I = 1 TO N
  READ B(I)
NEXT I

' stampa dei dati
LPRINT "Matrice dei coefficienti :"
CALL LprintMat(A, "##.### ")
LPRINT
LPRINT "Vettore termini noti :"
CALL LprintMat(B, "##.### ")
LPRINT

' inversione della matrice e controllo dell'inversione
CALL DuplicaMat(A(), A1())
CALL InversaMat(A1())
LPRINT "Inversa della matrice dei coefficienti :"
CALL LprintMat(A1(), "##.#### ")
LPRINT
CALL ProdottoMat(A(), A1(), A())
LPRINT "Controllo dell'inversione - prodotto della matrice per l'inversa :"
CALL LprintMat(A(), "##.##### ")
LPRINT

' calcolo e stampa del risultato
CALL ProdottoMat(A1(), B(), X())
LPRINT "Vettore risultato :"
CALL LprintMat(X(), "##.### ")

END

' dati per il programma
DATA 7
DATA 3, 2, 1, 0, 0, 0, 0
DATA 2, -2, -1, -2, 0, 0, 0
DATA 1, -1, 2, -1, 1, 0, 0
DATA 0, -2, -1, 4, 2, -3, 0

```

DATA 0, 0, 1, 2,-1, 1,-1
 DATA 0, 0, 0,-3, 1, 2,-1
 DATA 0, 0, 0, 0,-1,-1, 5
 DATA -1,-7, 5,-7,-2, 2, 9

L'output fornito dal programma è riportato nel seguito. Si noti come il prodotto della matrice A per la sua inversa A1 fornisce una matrice che si discosta leggermente dall'identità. I termini che dovrebbero essere rigorosamente nulli sono invece molto piccoli (dell'ordine di 10^{-7}) ma diversi da zero, in conseguenza delle approssimazioni connesse alla modalità di memorizzazione dei numeri reali nel calcolatore.

Matrice dei coefficienti :

| | | | | | | |
|-------|--------|--------|--------|--------|--------|--------|
| 3.000 | 2.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 2.000 | -2.000 | -1.000 | -2.000 | 0.000 | 0.000 | 0.000 |
| 1.000 | -1.000 | 2.000 | -1.000 | 1.000 | 0.000 | 0.000 |
| 0.000 | -2.000 | -1.000 | 4.000 | 2.000 | -3.000 | 0.000 |
| 0.000 | 0.000 | 1.000 | 2.000 | -1.000 | 1.000 | -1.000 |
| 0.000 | 0.000 | 0.000 | -3.000 | 1.000 | 2.000 | -1.000 |
| 0.000 | 0.000 | 0.000 | 0.000 | -1.000 | -1.000 | 5.000 |

Vettore termini noti :

-1.000
 -7.000
 5.000
 -7.000
 -2.000
 2.000
 9.000

Inversa della matrice dei coefficienti :

| | | | | | | |
|---------|---------|---------|---------|---------|---------|---------|
| 0.2272 | 0.1824 | -0.0464 | 0.0680 | 0.1159 | 0.0618 | 0.0355 |
| 0.1824 | -0.2090 | -0.1291 | -0.0440 | -0.1774 | 0.0054 | -0.0344 |
| -0.0464 | -0.1291 | 0.3972 | -0.1159 | 0.0070 | -0.1963 | -0.0379 |
| 0.0680 | -0.0440 | -0.1159 | 0.1700 | 0.2898 | 0.1546 | 0.0889 |
| 0.1159 | -0.1774 | 0.0070 | 0.2898 | -0.0174 | 0.4907 | 0.0947 |
| 0.0618 | 0.0054 | -0.1963 | 0.1546 | 0.4907 | 0.5951 | 0.2172 |
| 0.0355 | -0.0344 | -0.0379 | 0.0889 | 0.0947 | 0.2172 | 0.2624 |

Controllo dell'inversione - prodotto della matrice per l'inversa :

| | | | | | | |
|-----------|-----------|-----------|-----------|-----------|----------|-----------|
| 1.00E+00 | 2.98E-08 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 | 1.49E-08 |
| -1.49E-08 | 1.00E+00 | 0.00E+00 | -2.98E-08 | 0.00E+00 | 0.00E+00 | -2.98E-08 |
| 0.00E+00 | -4.47E-08 | 1.00E+00 | 0.00E+00 | -2.98E-08 | 0.00E+00 | 7.45E-09 |
| 0.00E+00 | -7.45E-09 | -5.96E-08 | 1.00E+00 | 0.00E+00 | 0.00E+00 | 0.00E+00 |
| 2.98E-08 | -4.10E-08 | 1.49E-08 | 6.71E-08 | 1.00E+00 | 1.19E-07 | 5.96E-08 |
| 3.73E-08 | -3.35E-08 | -7.45E-08 | -7.45E-09 | -3.73E-08 | 1.00E+00 | 0.00E+00 |
| -2.98E-08 | 1.49E-08 | 1.49E-08 | -2.98E-08 | 2.98E-08 | 1.19E-07 | 1.00E+00 |

Vettore risultato :

-2.000
1.000
3.000
-1.000
1.000
0.000
2.000

CAPITOLO TERZO

SOLUZIONE DEI SISTEMI DI EQUAZIONI LINEARI

1. Generalità

Un sistema di n equazioni lineari in n incognite, esplicitamente indicato da

$$\begin{aligned} a_{11} x_1 + a_{12} x_2 + a_{13} x_3 + \dots + a_{1n} x_n &= b_1 \\ a_{21} x_1 + a_{22} x_2 + a_{23} x_3 + \dots + a_{2n} x_n &= b_2 \\ a_{31} x_1 + a_{32} x_2 + a_{33} x_3 + \dots + a_{3n} x_n &= b_3 \\ \dots \dots + \dots \dots + \dots \dots + \dots + \dots \dots &= \dots \\ a_{n1} x_1 + a_{n2} x_2 + a_{n3} x_3 + \dots + a_{nn} x_n &= b_n \end{aligned}$$

può essere sinteticamente scritto, utilizzando la notazione matriciale, nella forma

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \dots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \dots \\ b_n \end{bmatrix}$$

ovvero

$$\mathbf{A} \mathbf{x} = \mathbf{B}$$

dove \mathbf{A} è la matrice quadrata di ordine n che raccoglie i coefficienti delle incognite del sistema, \mathbf{x} il vettore colonna delle incognite e \mathbf{B} il vettore colonna dei termini noti.

Per le regole dell'algebra matriciale la soluzione del sistema è data dalla relazione

$$\mathbf{x} = \mathbf{A}^{-1} \mathbf{B}$$

cioè dal prodotto dell'inversa della matrice dei coefficienti per il vettore termini noti. L'utilizzazione delle istruzioni matriciali, e quindi delle procedure illustrate nel capitolo precedente, non è però la maniera ottimale per risolvere il sistema. Il numero di moltiplicazioni e divisioni necessarie per l'inversione di una matrice di ordine n è, come mostrato in precedenza, approssimativamente pari ad n^3 . Ad esse vanno aggiunte altre n^2 moltiplicazioni per ciascun vettore di termini noti, occorrenti per effettuare il prodotto matriciale $\mathbf{A}^{-1} \mathbf{B}$. Esistono invece procedimenti numerici, come il metodo di riduzione di Gauss illustrato nei paragrafi successivi, che consentono di risolvere direttamente il sistema con un numero di operazioni circa tre volte inferiore.

Nelle applicazioni ai problemi strutturali vi è una ulteriore motivazione contro l'uso delle istruzioni matriciali. La matrice dei coefficienti contiene infatti una grande quantità di zeri e, nella quasi totalità dei casi, i termini non nulli sono racchiusi in una ristretta banda a cavallo della diagonale principale (ed eventualmente in fasce laterali, in dipendenza alla numerazione delle incognite). L'inversa di una matrice a banda è però una matrice piena, cioè senza termini nulli; questa caratteristica non produce quindi alcuna semplificazione, né sul numero di operazioni né sull'ingombro di memoria, nella fase di inversione. I procedimenti diretti di soluzione consentono invece di tener conto della particolare conformazione della matrice, riducendo le operazioni necessarie e l'occupazione di memoria in maniera tanto più sensibile quanto più ristretta è la banda. In tal modo si aumenta la velocità di soluzione e si ampliano considerevolmente le dimensioni massime delle strutture esaminabili con un determinato calcolatore.

Nei paragrafi successivi il metodo di riduzione viene descritto innanzitutto nelle sue generalità, cioè nel caso di matrice dei coefficienti piena (non a banda). Successivamente vengono mostrati gli adattamenti necessari per le applicazioni all'analisi strutturale, nella quale la matrice è simmetrica e con una particolare conformazione di banda.

Le procedure sviluppate per la risoluzione di un sistema di equazioni mediante il metodo di riduzione sono contenute nel file SOLSIST.BAS. Esse possono essere divise in due gruppi:

- procedure generali, per la risoluzione di un sistema con matrice dei coefficienti piena:
Triang, Risolve;
- procedure orientate all'analisi strutturale, per la risoluzione di un sistema con matrice dei coefficienti simmetrica e con la conformazione a banda descritta nella prima parte del testo:
Triang.K1, Risolve.K1 per banda memorizzata in un array monodimensionale
Triang.K2, Risolve.K2 per banda memorizzata in un array bidimensionale.

2. Il metodo di riduzione di Gauss

2.1. Le fasi del procedimento

Il metodo di riduzione (detto anche di eliminazione o di fattorizzazione) fu proposto dal matematico tedesco Karl Friedrich Gauss all'inizio del secolo scorso. Nel corso di questi quasi duecento anni ne sono state sviluppate numerose varianti, che si possono presentare più o meno utili a seconda dei casi. Tutte si riconducono sostanzialmente all'idea di sostituire ad ogni equazione del sistema una equivalente combinazione lineare, di se stessa e di altre, riducendo la matrice dei coefficienti ad una matrice triangolare superiore, cioè nella quale siano nulli tutti i termini al di sotto della diagonale principale (*prima fase — triangolarizzazione*)

$$\begin{bmatrix} a'_{11} & a'_{12} & a'_{13} & \dots & a'_{1n} \\ 0 & a'_{22} & a'_{23} & \dots & a'_{2n} \\ 0 & 0 & a'_{33} & \dots & a'_{3n} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & a'_{nn} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \dots \\ x_n \end{bmatrix} = \begin{bmatrix} b'_1 \\ b'_2 \\ b'_3 \\ \dots \\ b'_n \end{bmatrix}$$

o sinteticamente

$$\mathbf{A}' \mathbf{x} = \mathbf{B}'$$

Il sistema così ottenuto è di immediata soluzione, partendo dall'ultima equazione che contiene una sola incognita e proseguendo poi fino alla prima,

sostituendo man mano il valore trovato per ciascuna incognita nelle equazioni precedenti (*seconda fase — sostituzione all'indietro*).

Per comprendere meglio il procedimento, si esamini in dettaglio la fase di triangolarizzazione. Il procedimento inizia sottraendo multipli della prima equazione a tutte le altre in maniera tale da azzerare per esse i coefficienti dell'incognita x_1 : alla seconda equazione sarà sottratta la prima moltiplicata per a_{21}/a_{11} , alla terza la prima moltiplicata per a_{31}/a_{11} , e così via. Il termine a_{11} viene denominato *pivot* del primo passo di riduzione; i rapporti innanzi indicati possono essere chiamati *fattori di riduzione*. Si passa ora alla seconda equazione. Il valore corrente del termine a_{22} è il pivot del secondo passo di riduzione. Per annullare in tutte le equazioni successive alla seconda i coefficienti di x_2 , alla terza equazione sarà sottratta la seconda moltiplicata per a_{32}/a_{22} , alla quarta la seconda moltiplicata per a_{42}/a_{22} , e così via (si noti che i termini a_{22} , a_{32} , a_{42} qui utilizzati rappresentano i valori correnti dei coefficienti delle incognite, modificati rispetto a quelli iniziali per le operazioni effettuate nel primo passo). Si utilizza quindi la terza equazione per annullare i coefficienti di x_3 , e si prosegue con la quarta, la quinta ... fino alla penultima, completando con essa la triangolarizzazione.

Le operazioni da effettuare possono essere espresse analiticamente in maniera generale. Quando si utilizza l'equazione k ($k = 1 \div n-1$) per azzerare il termine a_{ik} , coefficiente di x_k nell'equazione i ($i = k+1 \div n$), devono essere calcolati i nuovi valori a'_{ij} dei termini dell'equazione ($j = k+1 \div n$) ed il nuovo termine noto b'_i con le espressioni

$$a'_{ij} = a_{ij} - a_{kj} \frac{a_{ik}}{a_{kk}} \quad (3.1)$$

$$b'_i = b_i - b_k \frac{a_{ik}}{a_{kk}}$$

La fase di sostituzione all'indietro parte invece dall'ultima riga, dalla quale si ricava $x_n = b'_n/a'_{nn}$. Sostituendo il valore così trovato in tutte le altre equazioni, il termine noto di ciascuna di esse verrà modificato sottraendo al suo valore corrente il prodotto dell'incognita x_n per il relativo coefficiente. Si passa quindi alla penultima equazione, nella quale rimane ora solo l'incognita x_{n-1} . Il suo valore viene determinato e sostituito nelle altre equazioni, allo stesso modo di quanto fatto per x_n . Si prosegue con l'equazione $n-2$, poi con la $n-3$... fino alla prima (quando si è risolta questa non è più necessaria la sostituzione del valore trovato perché non rimangono altre equazioni).

Analiticamente, si può dire che quando si utilizza l'equazione k ($k = n \div 1$) l'incognita x_k è fornita da

$$x_k = \frac{b'_k}{a'_{kk}} \quad (3.2)$$

Sostituendo questo valore nella equazione i ($i = 1 \div k - 1$, per $k > 1$), il nuovo valore b''_i del termine noto è espresso da

$$b''_i = b'_i - a'_{ik} x_k \quad (3.3)$$

Per confermare la già preannunciata convenienza del procedimento, si deve calcolare il numero di operazioni necessarie (contando solo le moltiplicazioni e divisioni, che richiedono al calcolatore un tempo nettamente maggiore rispetto ad addizioni e sottrazioni).

Per azzerare il termine a_{i1} , coefficiente della x_1 nell'equazione i , occorrono una divisione per calcolare il fattore di riduzione ed $n - 1$ moltiplicazioni per determinare i nuovi valori dei coefficienti delle altre incognite nell'equazione (cioè un totale di n operazioni), nonché una moltiplicazione per aggiornare il termine noto. Per azzerare i coefficienti della x_1 in tutte le $n - 1$ equazioni successive alla prima, sono quindi necessarie $n(n - 1) = n^2 - n$ operazioni su **A** ed $n - 1$ su **B**. Analogamente, per azzerare i coefficienti di x_2 occorrono $(n - 1)^2 - (n - 1)$ operazioni su **A** ed $n - 2$ su **B**, e così via. Nella fase di triangolarizzazione si effettuano dunque $n^2 - n + (n - 1)^2 - (n - 1) + \dots = (n^3 - n)/3$ operazioni su **A** ed $n - 1 + n - 2 + \dots = (n^2 - n)/2$ su **B**.

Nella seconda fase, occorre una divisione per determinare il valore di x_n ed $n - 1$ moltiplicazioni per sostituirlo nelle restanti equazioni, cioè in tutto n operazioni. Analogamente ne servono $n - 1$ per x_{n-1} , e così via, per un totale di $n + (n - 1) + \dots = (n^2 + n)/2$ operazioni su **B**.

La soluzione del sistema richiede pertanto l'effettuazione di circa $n^3/3$ operazioni su **A** ed n^2 su **B**.

2.2. Riduzione e fattorizzazione

Prima di procedere ulteriormente, è importante evidenziare alcune ulteriori caratteristiche del metodo di Gauss. Si prenda nuovamente in esame il primo passo della fase di triangolarizzazione. Si è visto che per annullare il termine a_{21} è necessario sottrarre alla seconda riga la prima moltiplicata per il fattore di riduzione a_{21}/a_{11} . Si può facilmente controllare che questa operazione equivale a premoltiplicare la matrice **A** di partenza per una

matrice elementare \mathbf{E}_{21} , che si differenzia dalla matrice identità solo per la presenza del termine $e_{21} = -a_{21}/a_{11}$

$$\mathbf{E}_{21} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ -\frac{a_{21}}{a_{11}} & 1 & \dots & 0 & \\ 0 & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}$$

In maniera analoga, l'azzeramento del generico termine a_{ik} equivale a pre-moltiplicare la matrice \mathbf{A} corrente (cioè già modificata nei passi precedenti) per la matrice \mathbf{E}_{ik} , caratterizzata dal termine $e_{ik} = -a_{ik}/a_{kk}$. Si ha pertanto

$$\mathbf{A}' = \mathbf{E}_n \mathbf{E}_{n-1} \dots \mathbf{E}_{ik} \dots \mathbf{E}_{31} \mathbf{E}_{21} \mathbf{A}$$

Questa relazione può essere invertita scrivendo

$$\mathbf{A} = \mathbf{E}_{21}^{-1} \mathbf{E}_{31}^{-1} \dots \mathbf{E}_{ik}^{-1} \dots \mathbf{E}_{n-1}^{-1} \mathbf{A}'$$

ovvero

$$\mathbf{A} = \mathbf{L} \mathbf{A}'$$

ed è facile constatare che l'inversione di \mathbf{E}_{ik} comporta soltanto il cambiare segno all'elemento e_{ik} e che il prodotto delle inverse delle matrici elementari fornisce la matrice \mathbf{L} , triangolare inferiore, che contiene tutti i fattori di riduzione

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ \frac{a_{21}}{a_{11}} & 1 & 0 & \dots & 0 \\ \frac{a_{31}}{a_{11}} & \frac{a_{32}}{a_{22}} & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ \frac{a_{n1}}{a_{11}} & \frac{a_{n2}}{a_{22}} & \frac{a_{n3}}{a_{33}} & \dots & 1 \end{bmatrix}$$

La matrice \mathbf{A}' può essere a sua volta scomposta nel prodotto di una matrice diagonale \mathbf{D} ed una triangolare superiore con valori unitari nella diagonale principale \mathbf{U}

$$\mathbf{A}' = \mathbf{D} \mathbf{U}$$

con

$$\mathbf{D} = \begin{bmatrix} a'_{11} & 0 & 0 & \dots & 0 \\ 0 & a'_{22} & 0 & \dots & 0 \\ 0 & 0 & a'_{33} & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & a'_{nn} \end{bmatrix}$$

e

$$\mathbf{U} = \begin{bmatrix} 1 & \frac{a'_{12}}{a'_{11}} & \frac{a'_{13}}{a'_{11}} & \dots & \frac{a'_{1n}}{a'_{11}} \\ 0 & 1 & \frac{a'_{23}}{a'_{22}} & \dots & \frac{a'_{2n}}{a'_{22}} \\ 0 & 0 & 1 & \dots & \frac{a'_{3n}}{a'_{33}} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}$$

In definitiva si può scrivere

$$\mathbf{A} = \mathbf{L} \mathbf{D} \mathbf{U} \quad (3.4)$$

evidenziando così che il procedimento di riduzione equivale alla scomposizione della matrice \mathbf{A} in tre fattori, una matrice triangolare inferiore \mathbf{L} , una diagonale \mathbf{D} ed una triangolare superiore \mathbf{U} . Per tale motivo esso è anche denominato metodo di fattorizzazione.

È possibile dimostrare (ma il farlo esula dai compiti del presente testo) che questa scomposizione è unica, cioè che non esiste una differente terna di matrici dello stesso tipo che, moltiplicate nello stesso ordine, diano ancora la matrice \mathbf{A} . Si noti però che è possibile effettuare il procedimento di riduzione all'incontrario, azzerando i termini al di sopra della diagonale principale. Si giunge in tal caso alla scomposizione

$$\mathbf{A} = \mathbf{U}' \mathbf{D}' \mathbf{L}' \quad (3.5)$$

essendo \mathbf{U}' una matrice triangolare superiore, \mathbf{D}' una matrice diagonale ed \mathbf{L}' una triangolare inferiore. Queste tre matrici sono differenti dalle $\mathbf{L} \mathbf{D} \mathbf{U}$, ma la scomposizione è ancora una volta unica.

2.3. Pivotaggio parziale

Il procedimento descritto cade in difetto se in un qualsiasi passo k della fase di triangolarizzazione il pivot a_{kk} è nullo. In tal caso, infatti, non ha

senso calcolare il rapporto a_{ik}/a_{kk} e non è possibile azzerare i coefficienti dell'incognita x_k nelle equazioni successive sommando ad esse multipli dell'equazione k . Se però in una di queste equazioni, ad esempio nella m , il termine a_{mk} è diverso da zero, per ottenere un nuovo pivot non nullo basta modificare l'ordine delle equazioni, scambiando tra loro la k con la m . Se invece il coefficiente di x_k è nullo in tutte le equazioni successive a k il sistema non è risolubile e la matrice dei coefficienti è una matrice singolare. Questa situazione si riscontra nelle applicazioni strutturali quando si esamina una struttura labile.

Un secondo problema che può riscontrarsi nella soluzione del sistema è legato alle inevitabili approssimazioni del calcolo. Ciascun elaboratore opera su valori memorizzati con un numero limitato di cifre significative, e quindi troncati o arrotondati. L'effettuazione delle numerose operazioni richieste per la soluzione del sistema può comportare un propagarsi ed incrementarsi degli errori di troncamento, in particolare quando i pivot hanno valori piccoli rispetto agli altri termini della matrice dei coefficienti. Nell'impostazione di una procedura generale per la risoluzione di un sistema di equazioni, oltre alla necessità di evitare pivot nulli si pone anche l'opportunità di rendere più grande possibile il loro valore. A tal fine, nel generico passo k della fase di triangolarizzazione si possono passare in rassegna tutte le equazioni, da k ad n , per individuare quella nella quale si ha il coefficiente di x_k massimo in valore assoluto. Tale equazione verrà quindi scambiata con la k , così come fatto per evitare pivot nulli.

Le manipolazioni effettuate sul sistema per ottenere pivot idonei sono denominate in generale pivotaggio. Il procedimento innanzi descritto, che richiede solo una modifica dell'ordine delle equazioni, è chiamato *pivotaggio parziale*, per distinguerlo da un procedimento più complesso, il *pivotaggio totale*. Quest'ultimo prevede che si assuma come pivot del passo k il massimo tra i coefficienti di tutte le incognite da k ad n , effettuando quindi non solo uno scambio delle equazioni ma anche un'inversione dell'ordine delle incognite. Tale tecnica può ridurre ulteriormente l'effetto degli errori di troncamento, ma è meno usata della precedente a causa dell'incremento dei tempi di soluzione conseguente al gran numero di confronti necessario per individuare ciascun pivot.

2.4. Risoluzioni multiple

La necessità di risolvere lo stesso sistema di equazioni più volte, con differenti colonne di termini noti, si presenta frequentemente nelle appli-

cazioni, in particolare per i problemi strutturali nei quali si devono usualmente valutare gli effetti di più schemi di carichi.

Un primo modo di affrontare il problema consiste nel memorizzare i termini noti in un array bidimensionale, cioè in una matrice con più colonne; è in tal caso possibile effettuare la triangolarizzazione e la sostituzione all'indietro contemporaneamente su tutte le colonne, ottenendo come risultato tutti i vettori soluzione.

Questo modo di procedere appare indubbiamente semplice e compatto, ma può risultare più o meno conveniente a seconda dell'organizzazione complessiva del programma, ed in particolare delle fasi di input e output.

L'unificazione della fase di calcolo risulta spontanea se si è scelto di presentare in maniera unitaria dati e risultati di tutte le condizioni di carico. In tal caso si procede a caricare in memoria centrale i dati dei diversi schemi di carico, preparare la matrice dei termini noti e calcolare i risultati. L'aspetto negativo è costituito dall'elevato ingombro di memoria, necessario non tanto per conservare le colonne aggiuntive, quanto per memorizzare l'intero insieme di dati di carico e di risultati di tutte le condizioni. L'ingombro può essere ridotto conservando le informazioni nella memoria di massa e richiamandole man mano, ma ciò allunga, spesso anche in misura notevole, i tempi di elaborazione.

Un'impostazione alternativa molto diffusa consiste nel presentare dati e risultati separatamente, schema per schema. In tal caso conviene operare sul singolo vettore termini noti ed è quindi necessario, per minimizzare il lavoro, scindere nettamente le due fasi del procedimento. Si può così effettuare quella di triangolarizzazione un'unica volta, solo sulla matrice dei coefficienti, memorizzando i fattori di riduzione. In un secondo momento, in maniera distinta per ciascuna colonna di termini noti, si ripetono su quest'ultima le stesse operazioni utilizzando i fattori memorizzati e si effettua quindi la sostituzione all'indietro.

Si può facilmente dimostrare che questa via non comporta in pratica alcun aggravio, di ingombro di memoria o di tempo, rispetto all'alternativa unitaria precedentemente descritta. Il generico passo k della fase di triangolarizzazione comporta l'azzeramento dei termini a_{ik} (con $i > k$) della matrice dei coefficienti, ottenuto sottraendo alla riga i la riga k moltiplicata per il fattore di riduzione a_{ik}/a_{kk} . Il valore del fattore deve essere salvato per poter poi ripetere la stessa operazione anche sul vettore termine noto. Poiché è ovviamente superfluo memorizzare il valore del termine a_{ik} , che si sa essere diventato nullo, è possibile conservare al suo posto il corrispondente fattore di riduzione. Se il procedimento prevede un pivotaggio parziale è necessario mantenere anche l'indicazione dello scambio di righe

effettuato, e quindi introdurre un ulteriore vettore IPVT, il cui elemento k conterrà il numero d'ordine della riga che è stata spostata in posizione k (sono sufficienti $n - 1$ elementi, perché la riga posta in ultima posizione viene univocamente determinata una volta individuate tutte le altre).

Nel testo si è seguita questa seconda impostazione. Pertanto per la risoluzione di un sistema di equazioni generico, con matrice dei coefficienti piena, sono fornite le due procedure, **Triang** e **Risolve**, che effettuano rispettivamente la prima e la seconda fase del procedimento.

3. Applicazione all'analisi strutturale

3.1. Ordine di esecuzione del procedimento

Alla fine del paragrafo 2.2 si è accennato alla possibilità di procedere alla riduzione in una maniera leggermente diversa, azzerando nella fase di triangolarizzazione i termini posti al di sopra della diagonale principale anziché quelli al di sotto di essa. Ciò richiede l'inversione dell'ordine in cui si opera sulle righe; si deve partire dall'ultima, assumendo come pivot il termine a_{nn} ed azzerando nelle altre equazioni i coefficienti dell'incognita x_n ; si passa poi alla penultima, e così via, fino alla seconda. Viceversa, la sostituzione all'indietro deve partire dalla prima riga, nella quale compare ora solo l'incognita x_1 , e procedere con la seconda, la terza ... fino all'ultima. Le espressioni utilizzate per l'aggiornamento dei coefficienti sono però ancora le (3.1), (3.2) e (3.3).

Seguire un'ordine o l'altro per un generico sistema di equazioni è sostanzialmente indifferente ai fini operativi, perché il risultato ottenuto e l'onere computazionale sono identici. È solo per motivi "storici" che il procedimento di riduzione è presentato nell'ordine utilizzato nel paragrafo 2.1.

Passando più specificamente alle applicazioni strutturali, si può notare che la triangolarizzazione è richiesta oltre che per la risoluzione delle equazioni di equilibrio del sistema anche per la determinazione della matrice di rigidità laterale di un telaio piano a partire dalla matrice completa. In questo secondo caso, avendo numerato le incognite ponendo in prima posizione quelle relative alla traslazione dei traversi bisogna necessariamente procedere dalla fine verso l'inizio ed arrestare anzitempo il procedimento, escludendo da esso il blocco relativo ai traversi. Nel caso in esame, con la specifica impostazione data al problema strutturale (che sarebbe potuta essere anche diversa), è quindi preferibile adottare l'ordine inverso rispetto a

quello classico ed organizzare la procedura di triangolarizzazione inserendo in essa una variabile di controllo che consenta l'arresto della riduzione ad una riga prefissata.

Si osserva inoltre che nella risoluzione di un singolo schema piano la ripetizione delle operazioni di triangolarizzazione sui termini noti e la sostituzione all'indietro vengono effettuate in maniera unitaria e completa. Il passaggio alla matrice di rigidità laterale, necessario per l'analisi dell'insieme spaziale di telai piani, richiede invece la separazione di questi due blocchi operativi e la distinzione tra incognite globali (spostamenti e rotazioni degli impalcati) e locali (rotazioni e spostamenti verticali dei nodi). La procedura di risoluzione è quindi anch'essa gestita da una variabile di controllo che consente l'effettuazione delle sole operazioni realmente necessarie.

3.2. Riduzione di una matrice simmetrica

La simmetria è una delle principali caratteristiche delle matrici strutturali. Grazie ad essa, è possibile individuare in maniera completa una matrice conservandone soltanto la diagonale principale ed il triangolo superiore (o quello inferiore), ed è quindi dimezzato l'ingombro di memoria necessario. A prima vista, non sembrerebbe possibile sfruttare questa caratteristica nella organizzazione del procedimento di riduzione, a causa della necessità di memorizzare i fattori di riduzione nel triangolo azzerato della matrice. È però facile constatare, come conseguenza del principio di unicità della scomposizione citato nel paragrafo 2.2, che la (3.5) diventa, grazie alla simmetria ¹

$$\mathbf{A} = \mathbf{U}' \mathbf{D}' \mathbf{U}'^T \quad (3.6)$$

cioè che la matrice triangolare inferiore ottenuta come risultato della riduzione è la trasposta della matrice triangolare superiore che contiene i fattori di riduzione, ed è pertanto sufficiente conservare solo una delle due.

Per meglio comprendere in che modo operare su una matrice memorizzata nella diagonale principale e nel triangolo superiore, si riesamini innanzitutto la fase di triangolarizzazione. Avendo invertito l'ordine, il pivot del primo passo è il termine a_{nn} . Per annullare il coefficiente di x_n

¹Infatti, essendo $A = U' D' L'$ deve essere, per le proprietà della trasposizione, $A^T = L'^T D'^T U'^T$; poiché per la simmetria si ha $A = A^T$, il principio di unicità comporta che $U' = L'^T$, $D' = D'^T$, $L' = L'^T$.

nell'equazione i , ai coefficienti di essa devono essere sottratti i corrispondenti termini dell'equazione n , moltiplicati per il fattore di riduzione a_i/a_n . Questo complesso di operazioni non altera la simmetria della sottomatrice ottenuta escludendo la riga e la colonna n . Infatti il nuovo valore a'_{ij} del coefficiente j -esimo dell'equazione i vale

$$a'_{ij} = a_{ij} - a_{nj} \frac{a_{in}}{a_{nn}} \quad (3.7)$$

ed è uguale al nuovo valore a'_{ji} del coefficiente i -esimo dell'equazione j

$$a'_{ji} = a_{ji} - a_{ni} \frac{a_{jn}}{a_{nn}}$$

poiché $a_{ij} = a_{ji}$, $a_{in} = a_{ni}$, $a_{jn} = a_{nj}$. Ciò evita il ricorso al triangolo inferiore e contemporaneamente consente una riduzione del numero di operazioni da effettuare. Infatti, procedendo all'azzeramento dei coefficienti di x_n facendo variare i da $n - 1$ ad 1 , nell'ordine, i nuovi valori dei termini a_{ij} del triangolo inferiore (per $j < i$) possono essere memorizzati in a_{ji} del triangolo superiore. Inoltre, l'espressione (3.7) deve essere applicata solo per j che va da 1 ad i , perché i termini corrispondenti a valori maggiori di j sono già stati determinati in precedenza. In generale, quindi, quando si utilizza l'equazione k ($k = n \div 2$) per azzerare il termine a_{ik} , coefficiente di x_k nell'equazione i ($i = k - 1 \div 1$), si calcolano i nuovi valori a'_{ji} ($j = 1 \div i$) con l'espressione

$$a'_{ji} = a_{ji} - a_{jk} \frac{a_{ik}}{a_{kk}} \quad (3.8)$$

nella quale tutti i termini si riferiscono al triangolo superiore perché $k > i \geq j$, e si memorizza in a_{ik} il fattore di riduzione a_{ik}/a_{kk} .

L'aggiornamento della colonna dei termini noti e la successiva fase di sostituzione all'indietro richiedono l'effettuazione delle stesse operazioni già descritte in precedenza, indipendentemente dalla simmetria della matrice. Occorre solo osservare che i coefficienti a_{ji} ora determinati appartengono alla matrice U^T che ha valori unitari nella diagonale principale, e quindi sono pari al rapporto tra quelli ottenuti col procedimento descritto per matrice non simmetrica ed i valori della diagonale principale. Di conseguenza, anche i coefficienti della colonna dei termini noti devono essere divisi per tali valori prima di iniziare la sostituzione all'indietro e non nel corso di questa fase.

Il numero di operazioni necessarie per la triangolarizzazione di una matrice dei coefficienti simmetrica può essere valutato riesaminando rapidamente il procedimento seguito. Nel generico passo che prende per base

l'equazione k occorre effettuare k per azzerare $a_{k-1 k}$ (1 divisione per calcolare il fattore di riduzione e $k-1$ moltiplicazioni per aggiornare i coefficienti delle altre incognite), $k-1$ per $a_{k-2 k}$, e così via, fino a 2 per azzerare a_{1k} , per un totale di $(k^2 + k - 2)/2$. Sommando i valori che tale espressione fornisce al variare di k da n a 2 si ottiene un totale di $(n^3 + 3n^2 - 4n)/6$, che è leggermente superiore alla metà di quelle necessarie in assenza di simmetria. Rimane invece invariato il numero di operazioni da effettuare sul vettore termini noti per la triangolarizzazione e la sostituzione all'indietro (n^2).

Nel procedimento finora descritto non si è ancora accennato alla possibilità di avere pivot nulli o molto piccoli ed alla conseguente opportunità di un pivotaggio. Bisogna subito osservare che lo scambio di righe altera la simmetria della matrice se non si effettua in contemporanea anche uno scambio di colonne (e quindi un'inversione dell'ordine delle incognite), e questo renderebbe più oneroso il pivotaggio. Tutte le applicazioni strutturali hanno però il vantaggio di condurre a matrici che oltre ad essere simmetriche sono anche definite positive. Si può dimostrare che in tal caso i pivot sono sempre maggiori di zero, e pertanto non è necessario effettuare uno scambio delle righe.

3.3. Riduzione di una matrice a banda

L'altra fondamentale caratteristica delle matrici strutturali, evidenziata nel capitolo 3 della prima parte del testo, è quella di presentare un gran numero di termini nulli. Nel caso di telai piani con traversi inestensibili, in conseguenza al criterio prescelto per la numerazione delle incognite i termini diversi da zero si trovano racchiusi in una banda diagonale e due fasce laterali, lungo il bordo sinistro e quello superiore (fig.3.1). Scomponendo la matrice di rigidezza della struttura, \mathbf{K} , in quattro sottomatrici, \mathbf{K}^{tt} , \mathbf{K}^{ta} , \mathbf{K}^{at} , \mathbf{K}^{aa} , le prime tre risultano sostanzialmente piene mentre la quarta (che è in genere nettamente maggiore delle altre) è una tipica matrice a banda, per la quale si indica nel seguito con s_b l'ampiezza della semibanda. Si usano inoltre i simboli n_t e n_a per indicare rispettivamente l'ordine delle matrici \mathbf{K}^{tt} e \mathbf{K}^{aa} .

La simmetria consente di memorizzare solo la metà superiore di \mathbf{K} , e quindi l'intera \mathbf{K}^{ta} e le metà superiori di \mathbf{K}^{tt} e \mathbf{K}^{aa} . \mathbf{K}^{tt} è sempre molto piccola rispetto alle altre e si preferisce usare per essa un array bidimensionale anche se metà di questo resta inutilizzato e quindi sprecato.

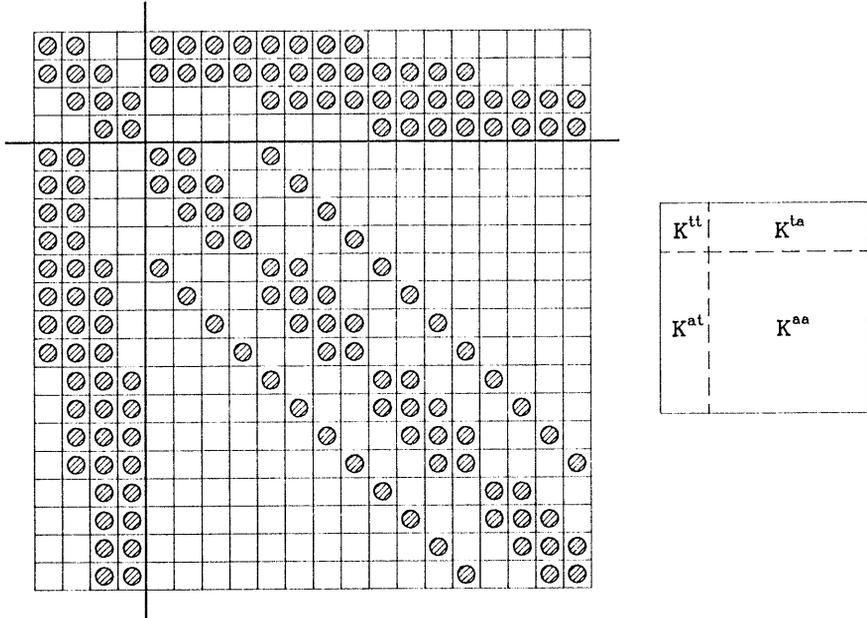


Fig. 3.1 — Scomposizione della matrice di rigidità \mathbf{K} di una struttura con traversi inestensibili

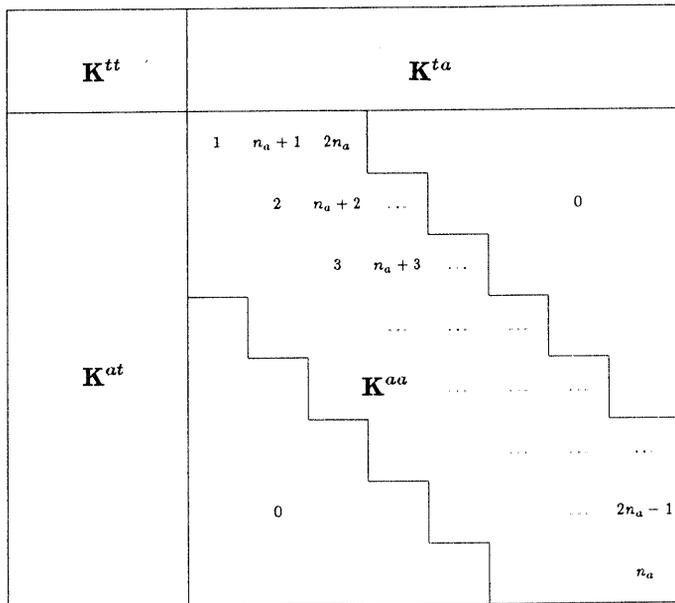


Fig. 3.2 — Numerazione progressiva degli elementi di \mathbf{K}^{aa} in un array monodimensionale

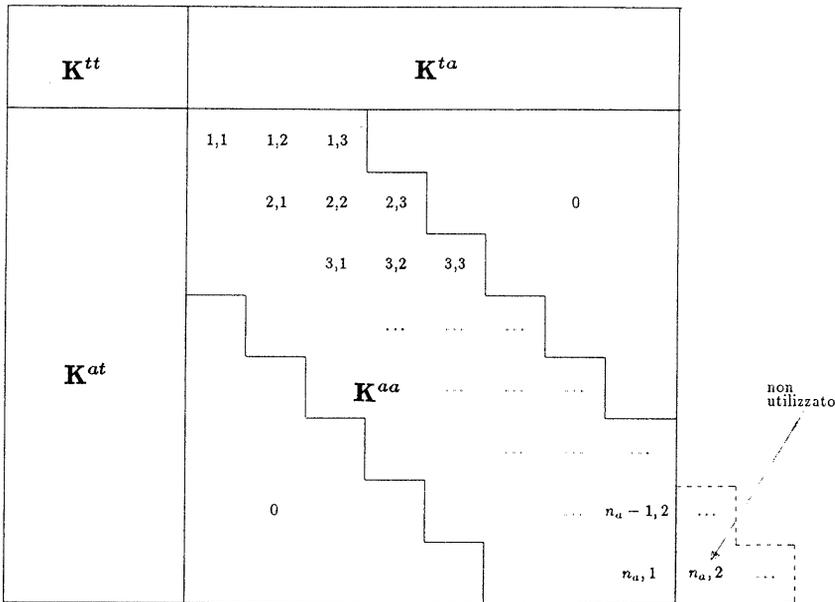


Fig. 3.3 — Numerazione progressiva degli elementi di K^{aa} in un array bidimensionale

K^{aa} è invece grande ed è quindi fondamentale sfruttare le caratteristiche di simmetria e di banda per minimizzare l'ingombro di memoria. A tal fine sono possibili due alternative: conservare gli elementi della banda in un array monodimensionale (fig.3.2) o in uno bidimensionale (fig.3.3). La prima scelta appare teoricamente la migliore, perché è necessario un numero di elementi esattamente pari a quello dei coefficienti racchiusi nella semibanda, $n_a s_b - s_b (s_b - 1)/2$; la seconda via richiede invece un array di n_a righe e s_b colonne, cioè $n_a s_b$ elementi, con un maggior ingombro che può diventare rilevante quando la semibanda è ampia (addirittura quasi il doppio se $s_b = n_a$). Il Quick Basic pone però un limite al massimo valore di ciascun indice e rende quindi impossibile utilizzare array monodimensionali con più di 32768 elementi, mentre con array bidimensionali se ne possono avere 32768×32768 . Nel caso specifico la seconda scelta consente di sfruttare meglio la memoria disponibile del calcolatore. Essendo questa una situazione contingente, legata al linguaggio scelto e quindi suscettibile di evoluzioni future, si esaminano comunque entrambe le alternative e si riportano per esse due distinte serie di procedure (di triangolarizzazione e di risoluzione).

Nel caso dell'array monodimensionale, gli elementi sono numerati considerando prima quelli della diagonale principale e poi via via quelli delle

diagonali successive. Il generico elemento posto nella riga i , colonna k (con $d = k - i \geq 0$) è quindi memorizzato in posizione $i + d[n_a - (d - 1)/2]$. Nell'array bidimensionale gli elementi rispettano invece l'organizzazione per righe della matrice \mathbf{K}^{aa} . L'elemento anzidetto viene quindi a trovarsi ancora nella riga i ma col secondo indice pari a $k - i + 1$.

La triangularizzazione procede sempre dal basso verso l'alto, ed interessa quindi prima \mathbf{K}^{aa} , poi \mathbf{K}^{tt} . Per l'azzeramento di ciascun termine vale ancora sostanzialmente la relazione (3.8), che deve essere però applicata tenendo conto sia della modalità di memorizzazione prescelta che della presenza dei termini nulli.

Nel passo che prende come base la riga k di \mathbf{K}^{aa} si devono annullare innanzitutto $s_b - 1$ (oppure $k - 1$, se $k < s_b$) coefficienti di \mathbf{K}^{aa} . In particolare, per azzerare K_{ik}^{aa} la relazione (3.8), deve essere applicata a tutti gli elementi della parte di riga appartenente a \mathbf{K}^{at} ed a quelli di \mathbf{K}^{aa} compresi tra la colonna $k - s_b + 1$ (perché nella riga k il primo elemento della banda occupa tale posizione) e la colonna i . Si ha pertanto

$$K_{ji}^{ta'} = K_{ji}^{ta} - K_{jk}^{ta} \frac{K_{ik}^{aa}}{K_{kk}^{aa}} \quad (3.9)$$

$$K_{ji}^{aa'} = K_{ji}^{aa} - K_{jk}^{aa} \frac{K_{ik}^{aa}}{K_{kk}^{aa}}$$

Si devono inoltre annullare n_t coefficienti di \mathbf{K}^{ta} . Per azzerare K_{ik}^{ta} occorre aggiornare solo i primi i coefficienti della riga i di \mathbf{K}^{tt} , perché i successivi (e tutti quelli di \mathbf{K}^{ta}) sono stati già modificati nell'azzerare termini precedenti. Si ha così

$$K_{ji}^{tt'} = K_{ji}^{tt} - K_{jk}^{ta} \frac{K_{ik}^{ta}}{K_{kk}^{aa}} \quad (3.10)$$

Queste espressioni devono essere ulteriormente modificate per quanto riguarda \mathbf{K}^{aa} per sostituire ai due indici che per esso compaiono un unico indice nel caso di memorizzazione in array monodimensionale o due nell'altro caso, con le regole innanzi definite.

La relazione da usare nella triangularizzazione della \mathbf{K}^{tt} è invece ottenuta direttamente sostituendo nelle (3.8) il simbolo K^{tt} al posto del simbolo a

$$K_{ji}^{tt'} = K_{ji}^{tt} - K_{jk}^{tt} \frac{K_{ik}^{tt}}{K_{kk}^{tt}} \quad (3.11)$$


```

SUB Risolve.K1 (KTT(), KTA(), KAA(), SB, RID, B())

' determina il numero di elementi degli array KTT, KTA, KAA, B
LRIT = LBOUND(KTT, 1)
LCTT = LBOUND(KTT, 2)
NRTT = UBOUND(KTT, 1)
NCTT = UBOUND(KTT, 2)
LRTA = LBOUND(KTA, 1)
LCTA = LBOUND(KTA, 2)
NRTA = UBOUND(KTA, 1)
NCTA = UBOUND(KTA, 2)
LRAA = LBOUND(KAA, 1)
NRAA = UBOUND(KAA, 1)
LRB = LBOUND(B, 1)
NRB = UBOUND(B, 1)

' controlla che le dimensioni degli array KTT, KTA, KAA siano corrette
' definisce NT, NA, N, ordine delle matrici dei coefficienti
IF LRIT > 1 OR LCTT > 1 OR LRTA > 1 OR LCTA > 1 OR LRAA > 1 THEN
    PRINT "ERRORE 1 in Risolve.K1 - limiti inferiori di KTT, KTA o KAA maggiori
        di 1"
    END
END IF
IF NRTT <> NCTT THEN
    PRINT "ERRORE 2 in Risolve.K1 - matrice KTT non quadrata"
    END
END IF
NT = NRTT
NA = NCTA
N = NT + NA
IF NRAA <> NA * SB - SB * (SB - 1) / 2 THEN
    PRINT "ERRORE 3 in Risolve.K1 - dimensioni di KAA non corrette"
    END
END IF
IF NRTA <> NT THEN
    PRINT "ERRORE 4 in Risolve.K1 - dimensioni di KTA non corrette"
    END
END IF
IF LRB > 1 THEN
    PRINT "ERRORE 5 in Risolve.K1 - limiti inferiori di B maggiori di 1"
    END
END IF
IF NRB < N THEN
    PRINT "ERRORE 6 in Risolve.K1 - array B insufficiente"
    END
END IF

SELECT CASE RID
CASE 0
    GOSUB TriangBa.K1
    GOSUB TriangBt.K1
    GOSUB DeterminaBt.K1
    GOSUB DeterminaBa.K1
CASE 1
    GOSUB TriangBa.K1
CASE 2
    GOSUB DeterminaBa.K1
END SELECT
EXIT SUB

```

TriangBa.K1:

```
' ripete sulla seconda parte di B le operazioni fatte per triang. KAA e KTT
FOR K = NA TO 1 STEP -1
  NK = NT + K
  C = B(NK)
  B(NK) = B(NK) / KAA(K)
  ' per l'azzeramento dei termini in KAA
  IF K < SB THEN LB = K ELSE LB = SB
  II = K - LB + 1
  DP = NA
  P = K
  FOR I = K - 1 TO II STEP -1
    NI = NT + I
    DP = DP - 1
    P = P + DP
    B(NI) = B(NI) - KAA(P) * C
  NEXT I
  ' per l'azzeramento dei termini in KTA
  FOR I = 1 TO NT
    B(I) = B(I) - KTA(I, K) * C
  NEXT I
NEXT K
RETURN
```

TriangBt.K1:

```
' ripete sulla prima parte di B le operazioni fatte per triang. KAA e KTT
FOR K = NT TO 1 STEP -1
  C = B(K)
  B(K) = B(K) / KTT(K, K)
  ' per l'azzeramento dei termini in KTT
  FOR I = 1 TO K - 1
    B(I) = B(I) - KTT(I, K) * C
  NEXT I
NEXT K
RETURN
```

DeterminaBt.K1:

```
' effettua la sostituzione all'indietro per le prime NT incognite
FOR K = 1 TO NT
  FOR I = K + 1 TO NT
    B(I) = B(I) - KTT(K, I) * B(K)
  NEXT I
NEXT K
RETURN
```

DeterminaBa.K1:

```
' effettua la sostituzione all'indietro per le altre NA incognite
FOR K = 1 TO NT
  FOR I = 1 TO NA
    NI = NT + I
    B(NI) = B(NI) - KTA(K, I) * B(K)
  NEXT I
NEXT K
FOR K = 1 TO NA
  NK = NT + K
  IF NA - K + 1 < SB THEN LB = NA - K + 1 ELSE LB = SB
  DP = NA
  P = K
```

```

FOR I = K + 1 TO K + LB - 1
  NI = NT + I
  P = P + DP
  DP = DP - 1
  B(NI) = B(NI) - KAA(P) * B(NK)
NEXT I
NEXT K
RETURN
END SUB

```

```

' ===== Risolve.K2 =====
'
' Risoluzione di un sistema di equazioni col metodo di riduzione di Gauss
' Seconda fase - risoluzione mediante sostituzione all'indietro
'
' NOTE: la matrice dei coefficienti K e' divisa in 4: KTT, KTA, KAT, KAA
' la matrice dei coefficienti e' simmetrica
' la matrice dei coefficienti e' a banda (in array bidimensionale)
' la matrice K e' stata resa triangolare inferiore, senza pivotaggio
'
' Variabili di ingresso:
' KTT(,) matrice dei coefficienti, triangolarizzata superiormente
' KTA(,) " " " , a banda (dimensioni NA,SB)
' KAA(,) " " "
' SB ampiezza della semibanda (compreso elemento diagonale)
' RID variabile di controllo del procedimento di risoluzione:
' RID = 0 triangolarizz. termini noti e sostituz. indietro
' RID = 1 triangolarizz. termini noti, esclusi i primi NT
' RID = 2 sostituzione indietro esclusi i primi NT termini
' B() vettore termini noti
'
' Variabili di uscita:
' B() vettore soluzione
'
' Variabili interne principali:
' NT ordine della matrice KTT
' NA ordine della matrice KAA
' N ordine della matrice dei coefficienti: N=NT+NA
'
'-----
SUB Risolve.K2 (KTT(), KTA(), KAA(), SB, RID, B())
' determina il numero di elementi degli array KTT, KTA, KAA, B
LRTT = LBOUND(KTT, 1)
LCTT = LBOUND(KTT, 2)
NRTT = UBOUND(KTT, 1)
NCTT = UBOUND(KTT, 2)
LRTA = LBOUND(KTA, 1)
LCTA = LBOUND(KTA, 2)
NRTA = UBOUND(KTA, 1)
NCTA = UBOUND(KTA, 2)
LRAA = LBOUND(KAA, 1)
LCAA = LBOUND(KAA, 2)
NRRA = UBOUND(KAA, 1)
NCAA = UBOUND(KAA, 2)
LRB = LBOUND(B, 1)
NRB = UBOUND(B, 1)

```

```

' controlla che le dimensioni degli array KTT, KTA, KAA siano corrette
' definisce NT, NA, N, ordine delle matrici dei coefficienti
IF LRIT > 1 OR LCTT > 1 OR LRTA > 1 OR LCTA > 1 OR LRAA > 1 OR LCAA > 1 THEN
  PRINT "ERRORE 1 in Risolve.K2 - limiti inferiori di KTT, KTA o KAA maggiori
    di 1"
  END
END IF
IF NRIT <> NCIT THEN
  PRINT "ERRORE 2 in Risolve.K2 - matrice KTT non quadrata"
  END
END IF
NT = NRIT
NA = NCTA
N = NT + NA
IF NRAA <> NA OR NCAA <> SB THEN
  PRINT "ERRORE 3 in Risolve.K2 - dimensioni di KAA non corrette"
  END
END IF
IF NRTA <> NT THEN
  PRINT "ERRORE 4 in Risolve.K2 - dimensioni di KTA non corrette"
  END
END IF
IF LRB > 1 THEN
  PRINT "ERRORE 5 in Risolve.K2 - limiti inferiori di B maggiori di 1"
  END
END IF
IF NRB < N THEN
  PRINT "ERRORE 6 in Risolve.K2 - array B insufficiente"
  END
END IF

SELECT CASE RID
CASE 0
  GOSUB TriangBa.K2
  GOSUB TriangBt.K2
  GOSUB DeterminaBt.K2
  GOSUB DeterminaBa.K2
CASE 1
  GOSUB TriangBa.K2
CASE 2
  GOSUB DeterminaBa.K2
END SELECT
EXIT SUB

TriangBa.K2:
' ripete sulla seconda parte di B le operazioni fatte per triang. KAA e KIT
FOR K = NA TO 1 STEP -1
  NK = NT + K
  C = B(NK)
  B(NK) = B(NK) / KAA(K, 1)
' per l'azzeramento dei termini in KAA
IF K < SB THEN LB = K ELSE LB = SB
II = K - LB + 1
FOR I = K - 1 TO II STEP -1
  D = K - I
  NI = NT + I
  B(NI) = B(NI) - KAA(I, D + 1) * C
NEXT I

```

```

      ' per l'azzeramento dei termini in KTA
      FOR I = 1 TO NT
        B(I) = B(I) - KTA(I, K) * C
      NEXT I
    NEXT K
  RETURN

```

TriangBt.K2:

```

      ' ripete sulla prima parte di B le operazioni fatte per triang. KAA e KTT
      FOR K = NT TO 1 STEP -1
        C = B(K)
        B(K) = B(K) / KTT(K, K)
        ' per l'azzeramento dei termini in KTT
        FOR I = 1 TO K - 1
          B(I) = B(I) - KTT(I, K) * C
        NEXT I
      NEXT K
  RETURN

```

DeterminaBt.K2:

```

      ' effettua la sostituzione all'indietro per le prime NT incognite
      FOR K = 1 TO NT
        FOR I = K + 1 TO NT
          B(I) = B(I) - KTT(K, I) * B(K)
        NEXT I
      NEXT K
  RETURN

```

DeterminaBa.K2:

```

      ' effettua la sostituzione all'indietro per le altre NA incognite
      FOR K = 1 TO NT
        FOR I = 1 TO NA
          NI = NT + I
          B(NI) = B(NI) - KTA(K, I) * B(K)
        NEXT I
      NEXT K
      FOR K = 1 TO NA
        NK = NT + K
        IF NA - K + 1 < SB THEN LB = NA - K + 1 ELSE LB = SB
        FOR I = K + 1 TO K + LB - 1
          D = K - I
          NI = NT + I
          B(NI) = B(NI) - KAA(K, 1 - D) * B(NK)
        NEXT I
      NEXT K
  RETURN

```

END SUB

```

' ===== Triang =====
'
' Risoluzione di un sistema di equazioni col metodo di riduzione di Gauss
' Prima fase - triangolarizzazione della matrice dei coefficienti
'
' NOTA: la matrice dei coefficienti viene resa triangolare superiore
' effettuando un pivotaggio parziale, cioe' uno scambio di righe
' per rendere massimo il termine sulla diagonale principale
'
' Variabili di ingresso:

```

```

'      A(.)      matrice dei coefficienti delle incognite, da triangolarizzare
'
' Variabili di uscita:
'      A(.)      matrice dei coefficienti, triangolarizzata superiormente
'                NOTA: nel triangolo inferiore sono conservati i fattori
'                usati per l'azzeramento
'      IPVT()    indice della riga del pivot K-esimo
'
' Variabili interne principali:
'      N         ordine della matrice dei coefficienti
'
' -----

```

```

SUB Triang (A(), IPVT())

```

```

' determina il numero di elementi degli array A ed IPVT
LRA = LBOUND(A, 1)
LCA = LBOUND(A, 2)
NRA = UBOUND(A, 1)
NCA = UBOUND(A, 2)
LRI = LBOUND(IPVT, 1)
NRI = UBOUND(IPVT, 1)

' controlla che le dimensioni degli array A ed IPVT siano corrette
IF LRA > 1 OR LCA > 1 OR LRI > 1 THEN
  PRINT "ERRORE 1 in Triang - limiti inferiori di A o IPVT maggiori di 1"
  END
END IF
IF NRA <> NCA THEN
  PRINT "ERRORE 2 in Triang - matrice dei coefficienti A non quadrata"
  END
END IF
IF NRI < NRA - 1 THEN
  PRINT "ERRORE 3 in Triang - array IPVT insufficiente"
  END
END IF

' definisce N, ordine della matrice dei coefficienti
N = NRA

' triangolarizzazione della matrice A
' NOTA: il ciclo non viene effettuato nel caso che N = 1
FOR K = 1 TO N - 1
  ' cerca il pivot per la riga K
  M = K
  FOR I = K + 1 TO N
    IF ABS(A(I, K)) > ABS(A(M, K)) THEN M = I
  NEXT I
  IPVT(K) = M
  SWAP A(M, K), A(K, K)
  IF A(K, K) = 0 THEN
    PRINT "ERRORE 4 in Triang - matrice dei coefficienti A singolare"
    END
  END IF
  ' determina i coefficienti moltiplicativi delle righe
  FOR I = K + 1 TO N
    A(I, K) = A(I, K) / A(K, K)
  NEXT I
  ' scambia le righe ed azzerata tutti gli elementi della colonna

```

```

FOR J = K + 1 TO N
  SWAP A(M, J), A(K, J)
  IF A(K, J) <> 0 THEN
    FOR I = K + 1 TO N
      A(I, J) = A(I, J) - A(I, K) * A(K, J)
    NEXT I
  END IF
NEXT J
NEXT K

```

```
END SUB
```

```

* ===== Triang.K1 =====
*
* Risoluzione di un sistema di equazioni col metodo di riduzione di Gauss
* Prima fase - triangolarizzazione della matrice dei coefficienti
*
* NOTE: la matrice dei coefficienti K e' divisa in 4: KTT, KTA, KAT, KAA
* la matrice dei coefficienti e' simmetrica
* la matrice dei coefficienti e' a banda (in array monodimensionale)
* la matrice K viene resa triangolare inferiore, senza pivotaggio
*
* Il procedimento di riduzione puo' essere limitato alla KAA e KTA,
* per ottenere in KTT la matrice di rigidezza laterale
*
* Variabili di ingresso:
* KTT(.) matrice dei coefficienti delle incognite, da triangolarizzare
* KTA(.) " " "
* KAA() " " " , a banda (dimensione N*SB-SB*(SB-1)/2)
* SB ampiezza della semibanda (compreso elemento diagonale)
* RID variabile di controllo del procedimento di riduzione:
* RID = 0 vengono triangolarizzate KTT e KAA
* RID = 1 viene triangolarizzata solo KAA
*
* Variabili di uscita:
* KTT(.) matrice dei coefficienti, triangolarizzata inferiormente
* KTA(.) " " "
* KAA() " " "
*
* Variabili interne principali:
* NT ordine della matrice KTT
* NA ordine della matrice KAA
*
* -----
SUB.Triang.K1 (KTT(), KTA(), KAA(), SB, RID)

```

```
' determina il numero di elementi degli array KTT, KTA, KAA
```

```

LRIT = LBOUND(KTT, 1)
LCIT = LBOUND(KTT, 2)
NRIT = UBOUND(KTT, 1)
NCIT = UBOUND(KTT, 2)
LRAT = LBOUND(KTA, 1)
LCAT = LBOUND(KTA, 2)
NRAT = UBOUND(KTA, 1)
NCAT = UBOUND(KTA, 2)
LRAA = LBOUND(KAA, 1)
NRAA = UBOUND(KAA, 1)

```

```

' controlla che le dimensioni degli array KTT, KTA, KAA siano corrette
' definisce NT, NA, ordine delle matrici dei coefficienti
IF LRIT > 1 OR LCIT > 1 OR LRIT > 1 OR LCIT > 1 OR LRIT > 1 THEN
  PRINT "ERRORE 1 in Triang.K1 - limiti inferiori di KTT, KTA o KAA maggiori
    di 1"
  END
END IF
IF NRIT <> NCTI THEN
  PRINT "ERRORE 2 in Triang.K1 - matrice KTT non quadrata"
  END
END IF
NT = NRIT
NA = NCTI
IF NRAA <> NA * SB - SB * (SB - 1) / 2 THEN
  PRINT "ERRORE 3 in Triang.K1 - dimensioni di KAA non corrette"
  END
END IF
IF NRIT <> NT THEN
  PRINT "ERRORE 4 in Triang.K1 - dimensioni di KTA non corrette"
  END
END IF

' triangolarizzazione della matrice dei coefficienti
' prima fase: triangolarizzazione di KAA
FOR K = NA TO 1 STEP -1
  IF KAA(K) = 0 THEN
    PRINT "ERRORE 5 in Triang.K1 - valore 0 nella diagonale principale di KAA"
    END
  END IF
  ' ciclo di azzeramento per le righe di KAT e KAA
  IF K < SB THEN LB = K ELSE LB = SB
  II = K - LB + 1
  DP = NA
  P = K
  FOR I = K - 1 TO II STEP -1
    DP = DP - 1
    P = P + DP
    IF KAA(P) <> 0 THEN
      ' determina il coefficiente moltiplicativo relativo alla riga I
      C = KAA(P) / KAA(K)
      ' modifica tutti gli elementi della riga I
      FOR J = 1 TO NT
        KTA(J, I) = KTA(J, I) - KTA(J, K) * C
      NEXT J
      DM = NA
      M = I
      DQ = DP
      Q = P
      FOR J = I TO II STEP -1
        KAA(M) = KAA(M) - KAA(Q) * C
        DM = DM - 1
        M = M + DM
        DQ = DQ - 1
        Q = Q + DQ
      NEXT J
      KAA(P) = C
    END IF
  NEXT I
  ' ciclo di azzeramento per le righe di KTT e KTA

```

```

FOR I = NT TO 1 STEP -1
  IF KTA(I, K) <> 0 THEN
    ' determina il coefficiente moltiplicativo relativo alla riga I
    C = KTA(I, K) / KAA(K)
    ' modifica tutti gli elementi della riga I
    FOR J = 1 TO I
      KTT(J, I) = KTT(J, I) - KTA(J, K) * C
    NEXT J
    KTA(I, K) = C
  END IF
NEXT I
NEXT K

' triangolarizzazione della matrice dei coefficienti
' seconda fase: triangolarizzazione di KTT
IF RID <> 1 THEN
  FOR K = NT TO 1 STEP -1
    IF KTT(K, K) = 0 THEN
      PRINT "ERRORE 6 in Triang.K1 - valore 0 nella diagonale principale
        di KTT"
    END IF
    ' ciclo di azzeramento per le righe di KTT
    FOR I = K - 1 TO 1 STEP -1
      IF KTT(I, K) <> 0 THEN
        ' determina il coefficiente moltiplicativo relativo alla riga I
        C = KTT(I, K) / KTT(K, K)
        ' modifica tutti gli elementi della riga I
        FOR J = 1 TO I
          KTT(J, I) = KTT(J, I) - KTT(J, K) * C
        NEXT J
        KTT(I, K) = C
      END IF
    NEXT I
  NEXT K
END IF
END SUB

```

```

===== Triang.K2 =====
'
' Risoluzione di un sistema di equazioni col metodo di riduzione di Gauss
' Prima fase - triangolarizzazione della matrice dei coefficienti
'
' NOTE: la matrice dei coefficienti K e' divisa in 4: KTT, KTA, KAT, KAA
' la matrice dei coefficienti e' simmetrica
' la matrice dei coefficienti e' a banda (in array bidimensionale)
' la matrice K viene resa triangolare inferiore, senza pivotaggio
'
' Il procedimento di riduzione puo' essere limitato alla KAA e KTA,
' per ottenere in KTT la matrice di rigidezza laterale
'
' Variabili di ingresso:
' KTT(.) matrice dei coefficienti delle incognite, da triangolarizzare
' KTA(.) " " "
' KAA(.) " " " , a banda (dimensioni NA,SB)
' SB ampiezza della semibanda (compreso elemento diagonale)
' RID variabile di controllo del procedimento di riduzione:
' RID = 0 vengono triangolarizzate KTT e KAA

```

```

      RID = 1      viene triangolarizzata solo KAA
*
*
* Variabili di uscita:
*   KTT(,)      matrice dei coefficienti, triangolarizzata inferiormente
*   KTA(,)      "      "      "
*   KAA(,)      "      "      "
*
* Variabili interne principali:
*   NT          ordine della matrice KTT
*   NA          ordine della matrice KAA
*
*-----
SUB Triang.K2 (KTT(), KTA(), KAA(), SB, RID)

  ' determina il numero di elementi degli array KTT, KTA, KAA
  LRIT = LBOUND(KTT, 1)
  LCIT = LBOUND(KTT, 2)
  NRIT = UBOUND(KTT, 1)
  NCIT = UBOUND(KTT, 2)
  LRTA = LBOUND(KTA, 1)
  LRCA = LBOUND(KTA, 2)
  NRTA = UBOUND(KTA, 1)
  NRCA = UBOUND(KTA, 2)
  LRAA = LBOUND(KAA, 1)
  LRCA = LBOUND(KAA, 2)
  NRAA = UBOUND(KAA, 1)
  NRCA = UBOUND(KAA, 2)

  ' controlla che le dimensioni degli array KTT, KTA, KAA siano corrette
  ' definisce NT, NA, ordine delle matrici dei coefficienti
  IF LRIT > 1 OR LCIT > 1 OR LRTA > 1 OR LRCA > 1 OR LRAA > 1 OR LRCA > 1 THEN
    PRINT "ERRORE 1 in Triang.K2 - limiti inferiori di KTT, KTA o KAA maggiori
      di 1"
  END
  END IF
  IF NRIT <> NCIT THEN
    PRINT "ERRORE 2 in Triang.K2 - matrice KTT non quadrata"
  END
  END IF
  NT = NRIT
  NA = NRCA
  IF NRAA <> NA OR NRCA <> SB THEN
    PRINT "ERRORE 3 in Triang.K2 - dimensioni di KAA non corrette"
  END
  END IF
  IF NRCA <> NT THEN
    PRINT "ERRORE 4 in Triang.K2 - dimensioni di KTA non corrette"
  END
  END IF

  ' triangolarizzazione della matrice dei coefficienti
  ' prima fase: triangolarizzazione di KAA
  FOR K = NA TO 1 STEP -1
    IF KAA(K, 1) = 0 THEN
      PRINT "ERRORE 5 in Triang.K2 - valore 0 nella diagonale principale di KAA"
    END
  END IF
  ' ciclo di azzeramento per le righe di KAT e KAA

```

```

IF K < SB THEN LB = K ELSE LB = SB
II = K - LB + 1
FOR I = K - 1 TO II STEP -1
  D = K - I
  IF KAA(I, D + 1) <> 0 THEN
    ' determina il coefficiente moltiplicativo relativo alla riga I
    C = KAA(I, D + 1) / KAA(K, 1)
    ' modifica tutti gli elementi della riga I
    FOR J = 1 TO NT
      KTA(J, I) = KTA(J, I) - KTA(J, K) * C
    NEXT J
    FOR J = II TO I
      M = I - J + 1
      KAA(J, M) = KAA(J, M) - KAA(J, M + D) * C
    NEXT J
    KAA(I, D + 1) = C
  END IF
NEXT I
' ciclo di azzeramento per le righe di KTT e KTA
FOR I = NT TO 1 STEP -1
  IF KTA(I, K) <> 0 THEN
    ' determina il coefficiente moltiplicativo relativo alla riga I
    C = KTA(I, K) / KAA(K, 1)
    ' modifica tutti gli elementi della riga I
    FOR J = 1 TO I
      KTT(J, I) = KTT(J, I) - KTA(J, K) * C
    NEXT J
    KTA(I, K) = C
  END IF
NEXT I
NEXT K

' triangolarizzazione della matrice dei coefficienti
' seconda fase: triangolarizzazione di KTT
IF RID <> 1 THEN
  FOR K = NT TO 1 STEP -1
    IF KTT(K, K) = 0 THEN
      PRINT "ERRORE 6 in Triang.K2 - valore 0 nella diagonale principale
        di KTT"
    END IF
  END IF
  ' ciclo di azzeramento per le righe di KTT
  FOR I = K - 1 TO 1 STEP -1
    IF KTT(I, K) <> 0 THEN
      ' determina il coefficiente moltiplicativo relativo alla riga I
      C = KTT(I, K) / KTT(K, K)
      ' modifica tutti gli elementi della riga I
      FOR J = 1 TO I
        KTT(J, I) = KTT(J, I) - KTT(J, K) * C
      NEXT J
      KTT(I, K) = C
    END IF
  NEXT I
NEXT K
END IF
END SUB

```

5. Esempio

Come esempio di programma principale che utilizza le procedure qui descritte è riportato un programma per la risoluzione di un sistema di equazioni generico mediante le procedure `Triang` e `Risolve`. Esso prevede la lettura dei dati (numero di equazioni N , matrice dei coefficienti A , termini noti B) conservati in apposite linee contraddistinte dall'istruzione `DATA`. Dopo aver stampato i dati, si effettuano le due fasi, di triangolarizzazione e di sostituzione all'indietro, ottenendo le incognite direttamente nel vettore B . Viene quindi stampata la matrice A modificata, contenente ora nel triangolo inferiore i fattori di riduzione, ed il vettore `IPVT` che indica l'eventuale scambio di righe nel pivotaggio parziale. Viene infine stampato il vettore B che racchiude i valori delle incognite.

RISOLUZIONE DI UN SISTEMA DI EQUAZIONI
mediante il metodo di riduzione di Gauss

```

DECLARE SUB LprintMat (A!(), F$)
DECLARE SUB Risolve (A!(), B!(), IPVT!())
DECLARE SUB Triang (A!(), IPVT!())

OPTION BASE 1

READ N
DIM A(N, N), B(N), IPVT(N - 1)

' lettura della matrice dei coefficienti A e del vettore termine noto B
FOR I = 1 TO N
  FOR J = 1 TO N
    READ A(I, J)
  NEXT J
NEXT I
FOR I = 1 TO N
  READ B(I)
NEXT I

' stampa dei dati
LPRINT "Matrice dei coefficienti : "
CALL LprintMat(A(), "##.### ")
LPRINT
LPRINT "Vettore termini noti : "
CALL LprintMat(B(), "##.### ")
LPRINT

' risoluzione col metodo di riduzione di Gauss
CALL Triang(A(), IPVT())
CALL Risolve(A(), B(), IPVT())

LPRINT "Matrice dei coefficienti triangolarizzata e fattori di riduzione : "
CALL LprintMat(A(), "##.### ")
LPRINT
LPRINT "Vettore scambi righe : "
```

```
CALL LprintMat(IPVT(), "##.###  ")
LPRINT
' stampa dei risultati
LPRINT "Vettore risultato : "
CALL LprintMat(B(), "##.###  ")
END
```

```
' dati per il programma
DATA 7
DATA 3, 2, 1, 0, 0, 0, 0
DATA 2,-2,-1,-2, 0, 0, 0
DATA 1,-1, 2,-1, 1, 0, 0
DATA 0,-2,-1, 4, 2,-3, 0
DATA 0, 0, 1, 2,-1, 1,-1
DATA 0, 0, 0,-3, 1, 2,-1
DATA 0, 0, 0, 0,-1,-1, 5
DATA -1,-7, 5,-7,-2, 2, 9
```

Il programma è stato mandato in esecuzione con i dati già usati per l'esempio del capitolo precedente. L'output ottenuto è riportato qui di seguito. Pur avendo utilizzato l'impostazione generale, la matrice dei coefficienti è in realtà simmetrica e dotata di banda. Si può notare di conseguenza che la stessa banda è presente nella matrice modificata **A**. Inoltre, dividendo i termini del triangolo superiore per i valori posti nella diagonale principale si ottengono i fattori di riduzione memorizzati nel triangolo inferiore, a conferma delle già citate proprietà delle matrici simmetriche.

Matrice dei coefficienti :

| | | | | | | |
|-------|--------|--------|--------|--------|--------|--------|
| 3.000 | 2.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 2.000 | -2.000 | -1.000 | -2.000 | 0.000 | 0.000 | 0.000 |
| 1.000 | -1.000 | 2.000 | -1.000 | 1.000 | 0.000 | 0.000 |
| 0.000 | -2.000 | -1.000 | 4.000 | 2.000 | -3.000 | 0.000 |
| 0.000 | 0.000 | 1.000 | 2.000 | -1.000 | 1.000 | -1.000 |
| 0.000 | 0.000 | 0.000 | -3.000 | 1.000 | 2.000 | -1.000 |
| 0.000 | 0.000 | 0.000 | 0.000 | -1.000 | -1.000 | 5.000 |

Vettore termini noti :

```
-1.000
-7.000
5.000
-7.000
-2.000
2.000
9.000
```

Matrice dei coefficienti triangolarizzata e fattori di riduzione :

| | | | | | | |
|-------|--------|--------|--------|--------|--------|--------|
| 3.000 | 2.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 0.667 | -3.333 | -1.667 | -2.000 | 0.000 | 0.000 | 0.000 |
| 0.333 | 0.500 | 2.500 | 0.000 | 1.000 | 0.000 | 0.000 |
| 0.000 | 0.600 | 0.000 | 5.200 | 2.000 | -3.000 | 0.000 |
| 0.000 | 0.000 | 0.400 | 0.385 | -2.169 | 2.154 | -1.000 |
| 0.000 | 0.000 | 0.000 | -0.577 | -0.993 | 2.408 | -1.993 |
| 0.000 | 0.000 | 0.000 | 0.000 | 0.461 | -0.828 | 3.811 |

Vettore scambi righe :

1.000
2.000
3.000
4.000
5.000
6.000

Vettore risultato :

-2.000
1.000
3.000
-1.000
1.000
-0.000
2.000

CAPITOLO QUARTO

LA SINGOLA ASTA

1. Generalità

Nel secondo capitolo della prima parte del testo è stato analizzato il comportamento di una generica asta in relazione alle sue caratteristiche geometriche, alle diverse condizioni di vincolo degli estremi ed alla presenza di una vasta gamma di tipologie di carico.

In particolare, sono state determinate: le relazioni che intercorrono, in assenza di carichi, tra azioni e componenti di movimento di estremità, sintetizzate dalla matrice di rigidità k dell'asta; le azioni di incastro perfetto \bar{S} , che insorgono per effetto dei carichi agenti sull'asta quando il movimento dei suoi estremi è impedito; le formulazioni che consentono di determinare le caratteristiche di sollecitazione in un qualsiasi punto dell'asta, in funzione dei carichi e delle azioni di estremità.

In analogia a quanto fatto per le operazioni matriciali e per la soluzione di sistemi di equazioni, anche in questo caso è stato realizzato un insieme di procedure che rendono immediatamente operativo quanto esposto nella parte teorica, in modo da facilitare il compito di chi desidera realizzare programmi di calcolo di validità generale.

Alle procedure, contenute nel file `ASTA.BAS`, sono stati assegnati nomi che richiamano alla mente quale prodotto forniscono ed il tipo di asta e di schema di carico cui si riferiscono.

Per individuare il tipo di asta si sono utilizzate le seguenti sigle:

| | | |
|--------|---|---|
| Cer | = | asta con cerniera ad uno o ad entrambi gli estremi; |
| Inc | = | asta perfettamente incastrata ai nodi di estremità; |
| NodRig | = | asta collegata a nodi rigidi non puntiformi; |
| Rig | = | asta con tratti rigidi coassiali agli estremi; |
| Wnk | = | asta su suolo elastico alla Winkler. |

Per i primi due tipi si è usata la sigla aggiuntiva Tag quando si è voluto indicare che per tale asta si tiene conto anche della deformazione conseguente al taglio (ottenendo quindi le sigle CerTag e IncTag).

Per individuare il tipo di carico si sono utilizzate le seguenti sigle:

| | | |
|--------|---|---|
| AzEstr | = | azioni di estremità; |
| DT | = | distorsioni termiche; |
| FM | = | forze e momenti applicati in un punto interno all'asta; |
| Qunif | = | carichi uniformemente distribuiti su tutta l'asta; |
| Qvar | = | carichi distribuiti con legge lineare su parte dell'asta. |

Le procedure possono essere divise in più gruppi:

- procedure che operano sui dati geometrici dell'asta, per fornire la matrice di rotazione Ld e quella di trasformazione T:
Geometria, T.Cer, T.Inc, T.Wnk;
- procedure che forniscono la matrice di rigidezza fondamentale dell'asta:
Kf.Cer, Kf.CerTag, Kf.Inc, Kf.IncTag, Kf.NodRig,
Kf.Rig, Kf.Wnk;
- procedure che effettuano la trasformazione della matrice di rigidezza da un sistema di riferimento a un'altro:
Kl. trasformazione da fondamentale a locale;
Kg.f trasformazione da fondamentale a globale;
Kg.l trasformazione da locale a globale;
- procedure che forniscono direttamente la matrice di rigidezza dell'asta nel sistema di riferimento globale:
Kg.Cer, Kg.CerTag, Kg.Inc, Kg.IncTag, Kg.NodRig,
Kg.Rig, Kg.Wnk;
- procedure riepilogative che conglobano gli effetti di Geometria e Kg.f, realizzate per minimizzare i tempi di esecuzione per specifici tipi di asta:
Asta.Inc;
- procedure per la determinazione delle azioni di incastro perfetto:
Sl.Cer.Qunif, Sl.Inc.DT, Sl.Inc.FM, Sl.Inc.Qunif,
Sl.Inc.Qvar, Sl.Rig.Qunif, Sl.Wnk.Qunif;


```

      per le matrici fondamentale e locale
      ,
      Variabili di ingresso:
      ,   X1   ascissa primo estremo
      ,   X2   ascissa secondo estremo
      ,   Y1   ordinata primo estremo
      ,   Y2   ordinata secondo estremo
      ,   E    modulo di elasticita' normale
      ,   A    area della sezione trasversale dell'asta
      ,   I    momento d'inerzia della sezione trasversale dell'asta
      ,
      Variabili di uscita:
      ,   L    lunghezza dell'asta
      ,   SnA1 seno dell'angolo di inclinazione dell'asta
      ,   CsA1 coseno dell'angolo di inclinazione dell'asta
      ,   Ld() matrice di rotazione dell'asta
      ,   Kg() matrice di rigidezza globale
      ,
      Procedure utilizzate:
      ,   Geometria
      ,
      -----
      ,
SUB Asta.Inc (X1, X2, Y1, Y2, E, A, I, L, SnA1, CsA1, Ld(), Kg())

      CALL Geometria(X1, X2, Y1, Y2, L, SnA1, CsA1, Ld())

      EIL = E * I / L
      EIL2 = EIL / L
      EIL3 = EIL2 / L
      EAL = E * A / L
      SN2 = SnA1 ^ 2
      CS2 = CsA1 ^ 2

      K1 = EAL * CS2 + 12 * EIL3 * SN2
      K2 = EAL * SN2 + 12 * EIL3 * CS2
      K3 = 6 * EIL2 * SnA1
      K4 = 6 * EIL2 * CsA1
      K5 = 4 * EIL
      K6 = 2 * EIL
      K7 = (EAL - 12 * EIL3) * SnA1 * CsA1

      Kg(1, 1) = K1
      Kg(1, 2) = K7
      Kg(1, 3) = -K3
      Kg(1, 4) = -K1
      Kg(1, 5) = -K7
      Kg(1, 6) = -K3

      Kg(2, 1) = K7
      Kg(2, 2) = K2
      Kg(2, 3) = K4
      Kg(2, 4) = -K7
      Kg(2, 5) = -K2
      Kg(2, 6) = K4

      Kg(3, 1) = -K3
      Kg(3, 2) = K4
      Kg(3, 3) = K5

```

$Kg(3, 4) = K3$
 $Kg(3, 5) = -K4$
 $Kg(3, 6) = K6$

$Kg(4, 1) = -K1$
 $Kg(4, 2) = -K7$
 $Kg(4, 3) = K3$
 $Kg(4, 4) = K1$
 $Kg(4, 5) = K7$
 $Kg(4, 6) = K3$

$Kg(5, 1) = -K7$
 $Kg(5, 2) = -K2$
 $Kg(5, 3) = -K4$
 $Kg(5, 4) = K7$
 $Kg(5, 5) = K2$
 $Kg(5, 6) = -K4$

$Kg(6, 1) = -K3$
 $Kg(6, 2) = K4$
 $Kg(6, 3) = K6$
 $Kg(6, 4) = K3$
 $Kg(6, 5) = -K4$
 $Kg(6, 6) = K5$

END SUB

```

' ===== CarSol.AzEstr =====
'
' Procedura per il calcolo di momento, taglio e sforzo normale:
' effetto delle azioni di estremita'
'
' Variabili di ingresso:
'   Fx      componente parallela all'asta dell'azione al primo estremo
'   Fy      componente ortogonale all'asta dell'azione al primo estremo
'   Mz      coppia concentrata Mz al primo estremo
'   X       ascissa punto calcolo delle sollecitazioni
'   L       lunghezza dell'asta
'
' Variabili di uscita:
'   Sn      sforzo normale all'ascissa X
'   Mf      momento flettente all'ascissa X
'   Ta      taglio all'ascissa X
'
' Procedure utilizzate:
'   =====
' -----
'

```

SUB Carsol.AzEstr (Fx, Fy, Mz, X, L, Sn, Mf, Ta)

```

IF X < 0 OR X > L THEN
  PRINT "ERRORE 1 in CarSol.AzEstr - valori dell'ascissa non accettabili"
  END
END IF

Sn = -Fx
Mf = Fy * X - Mz
Ta = Fy

```

END SUB

```

===== CarSol.FM =====
.
.
.  Procedura per il calcolo di momento, taglio e sforzo normale:
.  forza F (di componenti Fx e Fy) e coppia Mz concentrate
.
.  Variabili di ingresso:
.  Fx      componente parallela all'asta della forza concentrata F
.  Fy      componente ortogonale all'asta della forza concentrata F
.  Mz      coppia concentrata M
.  XO      ascissa punto applicazione azioni concentrate
.  X       ascissa punto calcolo delle sollecitazioni
.  L       lunghezza dell'asta
.
.  Variabili di uscita:
.  Sn      sforzo normale all'ascissa X
.  Mf      momento flettente all'ascissa X
.  Ta      taglio all'ascissa X
.
.  Procedure utilizzate:
.  =====
.
.-----

```

SUB Carsol.FM (Fx, Fy, Mz, XO, X, L, Sn, Mf, Ta)

```

IF XO < 0 OR XO > L OR X < 0 OR X > L THEN
  PRINT "ERRORE 1 in CarSol.FM - valori delle ascisse non accettabili"
END
END IF

IF X <= XO THEN
  Sn = 0
  Mf = 0
  Ta = 0
ELSE
  Sn = -Fx
  Mf = Fy * (X - XO) - Mz
  Ta = Fy
END IF

```

END SUB

```

===== CarSol.Quinif =====
.
.
.  Procedura per il calcolo di momento, taglio e sforzo normale:
.  carico uniformemente distribuito
.
.  Variabili di ingresso:
.  Q      carico uniformemente distribuito ortogonale all'asta
.  N      carico uniformemente distribuito parallelo all'asta
.  X      ascissa per il calcolo delle sollecitazioni
.  L      lunghezza dell'asta
.
.  Variabili di uscita:
.  Sn      sforzo normale all'ascissa X
.  Mf      momento flettente all'ascissa X

```

```

:      Ta      taglio all'ascissa X
:

```

```

: Procedure utilizzate:
:  ==
:
: -----
:

```

```

SUB CarSol.Qunif (Q, N, X, L, Sn, Mf, Ta)

```

```

  IF X < 0 OR X > L THEN
    PRINT "ERRORE 1 in CarSol.Qunif - valori dell'ascissa non accettabili"
  END
  END IF

```

```

  Sn = -N * X
  Mf = Q * X ^ 2 / 2
  Ta = Q * X

```

```

END SUB

```

```

===== CarSol.Qvar =====
=====

```

```

: Procedura per il calcolo di momento e taglio:
: carico distribuito con andamento lineare su parte dell'asta
:

```

```

: NOTA: il carico e' ortogonale all'asse e non provoca sforzo normale
:

```

```

: Variabili di ingresso:

```

```

:   Q1      valore del carico a sinistra
:   Q2      valore del carico a destra
:   X1      ascissa inizio carico
:   X2      ascissa fine carico
:   X       ascissa per calcolo sollecitazioni
:   L       lunghezza dell'asta
:

```

```

: Variabili di uscita:

```

```

:   Mf      momento flettente all'ascissa X
:   Ta      taglio all'ascissa X
:

```

```

: Procedure utilizzate:
:  ==
:
: -----
:

```

```

SUB Carsol.Qvar (Q1, Q2, X1, X2, X, L, Mf, Ta)

```

```

  IF X1 < 0 OR X1 > L OR X2 < 0 OR X2 > L OR X < 0 OR X > L THEN
    PRINT "ERRORE 1 in CarSol.Qvar - valori delle ascisse non accettabili"
  END
  END IF

```

```

  IF X1 > X2 THEN
    PRINT "ERRORE 2 in CarSol.Qvar - ordine delle ascisse non accettabile"
  END
  END IF

```

```

  IF X <= X1 THEN
    Mf = 0
    Ta = 0
  ELSEIF X < X2 THEN

```

```

Dx1 = X - X1
Dx2 = X - X2
D = X2 - X1
U = Dx1 / D
Mf = ((1 - U / 3) * Q1 + U / 3 * Q2) * Dx1 ^ 2 / 2
Ta = ((1 - U / 2) * Q1 + U / 2 * Q2) * Dx1
ELSE
Dx2 = X - X2
D = X2 - X1
TO = (Q1 + Q2) / 2 * D
MO = (2 / 3 * Q1 + 1 / 3 * Q2) * D ^ 2 / 2
Mf = TO * Dx2 + MO
Ta = TO
END IF
END SUB

===== Geometria =====
,
,
,   Procedura per la valutazione della matrice di rotazione
,   e di parametri che dipendono solo dalla geometria dell'asta
,
,   Variabili di ingresso:
,   X1      ascissa primo estremo
,   X2      ascissa secondo estremo
,   Y1      ordinata primo estremo
,   Y2      ordinata secondo estremo
,
,   Variabili di uscita:
,   L       lunghezza dell'asta
,   SnAl    seno dell'angolo di inclinazione dell'asta
,   CsAl    coseno dell'angolo di inclinazione dell'asta
,   Ld()    matrice di rotazione dell'asta
,
,   Procedure utilizzate:
,   AzzeraMat
,
,-----
SUB Geometria (X1, X2, Y1, Y2, L, SnAl, CsAl, Ld())

L = SQR((X2 - X1) ^ 2 + (Y2 - Y1) ^ 2)
IF L = 0 THEN
  PRINT "ERRORE 1 in Geometria - asta di lunghezza nulla"
  END
END IF
SnAl = (Y2 - Y1) / L
CsAl = (X2 - X1) / L

CALL AzzeraMat(Ld())
Ld(1, 1) = CsAl
Ld(1, 2) = -SnAl
Ld(2, 1) = SnAl
Ld(2, 2) = CsAl
Ld(3, 3) = 1
Ld(4, 4) = CsAl
Ld(4, 5) = -SnAl
Ld(5, 4) = SnAl
Ld(5, 5) = CsAl

```

Ld(6, 6) = 1

END SUB

```

===== Kf.Cer =====
.
.
. Procedura per la definizione della matrice di rigidezza fondamentale
. per un'asta incernierata agli estremi
.
. Variabili di ingresso:
.   ES      estremo incernierato
.           1 = primo estremo
.           2 = secondo estremo
.           3 = entrambi gli estremi
.   E       modulo di elasticita' normale
.   A       area della sezione trasversale dell'asta
.   I       momento d'inerzia della sezione trasversale dell'asta
.   L       lunghezza dell'asta
.
. Variabili di uscita:
.   Kf()    matrice di rigidezza fondamentale
.
. Procedure utilizzate:
.   Azzeramat
.
. -----

```

SUB Kf.Cer (ES, E, A, I, L, Kf())

```

CALL Azzeramat(Kf())
SELECT CASE ES
CASE 1, 2
  Kf(1, 1) = 3 * E * I / L
  Kf(2, 2) = E * A / L
CASE 3
  Kf(1, 1) = E * A / L
END SELECT

```

END SUB

```

===== Kf.CerTag =====
.
.
. Procedura per la definizione della matrice di rigidezza fondamentale
. per asta incernierata agli estremi tenendo conto della deformazione a taglio
.
. Variabili di ingresso:
.   ES      estremo incernierato
.           1 = primo estremo
.           2 = secondo estremo
.           3 = entrambi gli estremi
.   E       modulo di elasticita' normale
.   G       modulo di elasticita' tangenziale
.   CHI     fattore di taglio
.   A       area della sezione trasversale dell'asta
.   I       momento d'inerzia della sezione trasversale dell'asta
.   L       lunghezza dell'asta
.
. Variabili di uscita:
.   Kf()    matrice fondamentale dell'asta

```

```

.
.  Procedure utilizzate:
.  Azzeramat
.
. -----
SUB Kf.CerTag (ES, E, G, CHI, A, I, L, Kf())

  eps = 3 * E * I * CHI / (G * A * L ^ 2)

  CALL Azzeramat(Kf())
  SELECT CASE ES
    CASE 1, 2
      Kf(1, 1) = 3 * E * I / L / (1 + eps)
      Kf(2, 2) = E * A / L
    CASE 3
      Kf(1, 1) = E * A / L
  END SELECT

END SUB

. ===== Kf.Inc =====
.
.  Procedura per la definizione della matrice di rigidezza fondamentale
.  per un'asta perfettamente incastrata ai nodi di estremita'
.
.  Variabili di ingresso:
.  E      modulo di elasticita' normale
.  A      area della sezione trasversale dell'asta
.  I      momento d'inerzia della sezione trasversale dell'asta
.  L      lunghezza dell'asta
.
.  Variabili di uscita:
.  Kf()   matrice di rigidezza fondamentale
.
.  Procedure utilizzate:
.  Azzeramat
.
. -----
SUB Kf.Inc (E, A, I, L, Kf())

  CALL Azzeramat(Kf())
  EIL = 2 * E * I / L
  Kf(1, 1) = 2 * EIL
  Kf(1, 2) = EIL
  Kf(2, 1) = EIL
  Kf(2, 2) = Kf(1, 1)
  Kf(3, 3) = E * A / L

END SUB

. ===== Kf.IncTag =====
.
.  Procedura per la definizione della matrice di rigidezza fondamentale
.  per un'asta perfettamente incastrata ai nodi di estremita'
.  tenendo conto della deformazione a taglio
.
.  Variabili in ingresso:

```

```

'      E      modulo di elasticita' normale
'      G      modulo di elasticita' tangenziale
'      CHI    fattore di taglio
'      A      area della sezione trasversale dell'asta
'      I      momento d'inerzia della sezione trasversale dell'asta
'      L      lunghezza dell'asta

```

```

' Variabili in uscita:
'      Kf()   matrice di rigidezza fondamentale

```

```

' Procedure utilizzate:
'      Azzeramat

```

```

SUB Kf.IncTag (E, G, CHI, A, I, L, Kf())

```

```

      eps = 3 * E * I * CHI / (G * A * L ^ 2)

```

```

      CALL Azzeramat(Kf())
      EIL = 2 * E * I / L
      Kf(1, 1) = 2 * EIL * (1 + eps) / (1 + 4 * eps)
      Kf(1, 2) = EIL * (1 - 2 * eps) / (1 + 4 * eps)
      Kf(2, 1) = Kf(1, 2)
      Kf(2, 2) = Kf(1, 1)
      Kf(3, 3) = E * A / L

```

```

END SUB

```

```

===== Kf.NodRig =====
'
' Procedura per la determinazione della matrice di rigidezza fondamentale
' nel caso di nodi rigidi non puntiformi
' (asta con tratti rigidi non coassiali)
'
' Variabili di ingresso:
'      E      modulo di elasticita' normale
'      A      area della sezione trasversale
'      I      momento d'inerzia della sezione trasversale
'      L      lunghezza dell'asta
'      Sx     coefficiente lunghezza x tratto sinistro
'      Sy     coefficiente lunghezza y tratto sinistro
'      Tx     coefficiente lunghezza x tratto destro
'      Ty     coefficiente lunghezza y tratto destro
'
' Variabili di uscita:
'      Kf()   matrice di rigidezza fondamentale
'
' Procedure utilizzate:
'      ProdottoMat
'      TrasponeMat
'      Kf.Inc
'
-----

```

```

SUB Kf.NodRig (E, A, I, L, Sx, Sy, Tx, Ty, Kf())

```

```

      C = 1 / SQR((1 + Tx - Sx) ^ 2 + (Ty - Sy) ^ 2)
      Ls = L / C

```

```

SnAl = (Ty - Sy) * C
CaAl = (1 - Sx + Tx) * C .

DIM C(3, 3)
C(1, 1) = 1 + C * (Sy * SnAl + Sx * CaAl)
C(2, 1) = -C * (Ty * SnAl + Tx * CaAl)
C(3, 1) = C * SnAl / L
C(1, 2) = C * (Sy * SnAl + Sx * CaAl)
C(2, 2) = 1 - C * (Ty * SnAl + Tx * CaAl)
C(3, 2) = C * SnAl / L
C(1, 3) = (Sy * CaAl - Sx * SnAl) * L
C(2, 3) = -(Ty * CaAl - Tx * SnAl) * L
C(3, 3) = CaAl

CALL Kf.Inc(E, A, I, Ls, Kf())
CALL ProdottoMat(C(), Kf(), Kf())
CALL TrasponeMat(C(), C())
CALL ProdottoMat(Kf(), C(), Kf())

```

END SUB

```

* ===== Kf.Rig =====
*
*
*   Procedura per la definizione della matrice di rigidezza fondamentale
*   per un'asta perfettamente incastrata ai nodi di estremita'
*   avente tratti rigidi di estremita'
*
*   Variabili di ingresso:
*   E      modulo di elasticita' normale
*   A      area della sezione trasversale dell'asta
*   I      momento d'inerzia della sezione trasversale dell'asta
*   L      lunghezza dell'asta
*   S      coefficiente lunghezza tratto rigido sinistro
*   T      coefficiente lunghezza tratto rigido destro
*
*   Variabili di uscita:
*   Kf()   matrice di rigidezza fondamentale
*
*   Procedure utilizzate:
*   AzzeraMat
*
* -----

```

SUB Kf.Rig (E, A, I, L, S, T, Kf())

```

C = 1 / (1 - S - T)
A1 = C * ((1 - T) ^ 2 + S * (1 - T) + S ^ 2)
A2 = C * ((1 - S) ^ 2 + T * (1 - S) + T ^ 2)
B = C * (1 + S + T - 2 * (S ^ 2 - S * T + T ^ 2))

CALL AzzeraMat(Kf())
EIL = 2 * E * I / L
Kf(1, 1) = 2 * EIL * A1
Kf(1, 2) = EIL * B
Kf(2, 1) = Kf(1, 2)
Kf(2, 2) = 2 * EIL * A2
Kf(3, 3) = E * A / L * C

```

END SUB

```

' ===== Kf.Wnk =====
'
' Procedura per la definizione della matrice di rigidezza fondamentale
' per un'asta su suolo elastico alla Winkler
'
' Variabili di ingresso:
'   K      costante di sottofondo
'   B      larghezza trave
'   E      modulo di elasticita' normale
'   A      area della sezione trasversale dell'asta
'   I      momento d'inerzia della sezione trasversale dell'asta
'   L      lunghezza dell'asta
'
' Variabili di uscita:
'   Kf()   matrice di rigidezza fondamentale
'
' Procedure utilizzate:
'   AzzerMat
'   InserMat
'   InversaMat
'   ProdottoCostMat
'   ProdottoMat
'
' -----

```

```
SUB Kf.Wnk (K, B, E, A, I, L, Kf())
```

```

CONST en = 2.718
lambda = (K * B / (4 * E * I)) ^ .25
lambdaL = lambda * L
e1 = en ^ lambdaL
e2 = en ^ (-lambdaL)
SnL = SIN(lambdaL)
CsL = COS(lambdaL)
sp = e1 * SnL
cp = e1 * CsL
ss = e2 * SnL
cs = e2 * CsL

```

```

DIM K1(4, 4)
K1(1, 1) = 0
K1(1, 2) = 1
K1(1, 3) = 0
K1(1, 4) = 1
K1(2, 1) = lambda
K1(2, 2) = lambda
K1(2, 3) = lambda
K1(2, 4) = -lambda
K1(3, 1) = sp
K1(3, 2) = cp
K1(3, 3) = ss
K1(3, 4) = cs
K1(4, 1) = lambda * (sp + cp)
K1(4, 2) = lambda * (-sp + cp)
K1(4, 3) = lambda * (-ss + cs)
K1(4, 4) = lambda * (-ss - cs)

```

```
DIM K2(4, 4)
```

```

K2(1, 1) = lambda
K2(1, 2) = -lambda
K2(1, 3) = lambda
K2(1, 4) = lambda
K2(2, 1) = -1
K2(2, 2) = 0
K2(2, 3) = 1
K2(2, 4) = 0
K2(3, 1) = lambda * (sp - cp)
K2(3, 2) = lambda * (sp + cp)
K2(3, 3) = lambda * (-ss - cs)
K2(3, 4) = lambda * (ss - cs)
K2(4, 1) = cp
K2(4, 2) = -sp
K2(4, 3) = -cs
K2(4, 4) = ss
CALL ProdottoCostMat(2 * E * I * lambda ^ 2, K2(), K2())

CALL InversaMat(K1())
CALL ProdottoMat(K2(), K1(), K2())
CALL Azzeramat(Kf())
CALL InserMat(K2(), Kf(), 1, 1)
Kf(5, 5) = E * A / L

```

END SUB

```

' ===== Kg.Cer =====
'
'   Procedura per la definizione della matrice di rigidezza globale
'   per un'asta incernierata agli estremi
'
'   Variabili di ingresso:
'   ES      estremo incernierato
'           1 = primo estremo
'           2 = secondo estremo
'           3 = entrambi gli estremi
'   E      modulo di elasticita' normale
'   A      area della sezione trasversale dell'asta
'   I      momento d'inerzia della sezione trasversale dell'asta
'   L      lunghezza dell'asta
'   Ld()   matrice di rotazione
'
'   Variabili di uscita:
'   Kg()   matrice di rigidezza globale
'
'   Procedure utilizzate:
'   Kf.Cer
'   Kg.f
'   T.Cer
'
' -----
SUB Kg.Cer (ES, E, A, I, L, Ld(), Kg())

```

```

SELECT CASE ES
CASE 1, 2
  N = 2
CASE 3
  N = 1

```

```

END SELECT
DIM Kf(N, N), T(6, N)

CALL Kf.Cer(ES, E, A, I, L, Kf())
CALL T.Cer(ES, L, T())
CALL Kg.f(Ld(), T(), Kf(), Kg())

```

```

END SUB

```

```

' ===== Kg.CerTag =====
'
' Procedura per la definizione della matrice di rigidezza globale
' per un'asta incernierata agli estremi
' tenendo conto della deformazione a taglio
'
' Variabili di ingresso:
'   ES      estremo incernierato
'           1 = primo estremo
'           2 = secondo estremo
'           3 = entrambi gli estremi
'   E       modulo di elasticita' normale
'   G       modulo di elasticita' tangenziale
'   CHI     fattore di taglio
'   A       area della sezione trasversale dell'asta
'   I       momento d'inerzia della sezione trasversale dell'asta
'   L       lunghezza dell'asta
'   Ld()    matrice di rotazione
'
' Variabili di uscita:
'   Kg()    matrice di rigidezza globale
'
' Procedure utilizzate:
'   Kf.CerTag
'   Kg.f
'   T.Cer
'
' -----
'

```

```

SUB Kg.CerTag (ES, E, G, CHI, A, I, L, Ld(), Kg())

```

```

SELECT CASE ES
CASE 1, 2
  N = 2
CASE 3
  N = 1
END SELECT
DIM Kf(N, N), T(6, N), Kloc(6, 6)

CALL Kf.CerTag(ES, E, G, CHI, A, I, L, Kf())
CALL T.Cer(ES, L, T())
CALL Kg.f(Ld(), T(), Kf(), Kg())

```

```

END SUB

```

```

' ===== Kg.f =====
'
' Procedura generale per la definizione della matrice di rigidezza globale
' a partire dalla matrice di rigidezza fondamentale
'

```

```

* Variabili di ingresso:
*   Ld()   matrice di rotazione
*   T()    matrice di trasformazione
*   Kf()   matrice di rigidezza fondamentale
*
* Variabili di uscita:
*   Kg()   matrice di rigidezza globale
*
* Variabili interne:
*   LT()   prodotto di Ld per T
*   LTT()  trasposta del prodotto di Ld per T
*
* Procedure utilizzate:
*   ProdottoMat
*   TrasponeMat

```

```

-----
SUB Kg.f (Ld(), T(), Kf(), Kg())

```

```

CALL LimitiMat(Kf(), DK, LRK, LCK, NRK, NCK)
DIM LT(6, NRK), LTT(NRK, 6), K1(6, NRK)

CALL ProdottoMat(Ld(), T(), LT())
CALL TrasponeMat(LT(), LTT())
CALL ProdottoMat(LT(), Kf(), K1())
CALL ProdottoMat(K1(), LTT(), Kg())

```

```

END SUB

```

```

===== Kg.Inc =====
*
* Procedura per la definizione della matrice di rigidezza globale
* per un'asta perfettamente incastrata ai nodi di estremita'
*
* Variabili di ingresso:
*   E      modulo di elasticita' normale
*   A      area della sezione trasversale dell'asta
*   I      momento d'inerzia della sezione trasversale dell'asta
*   L      lunghezza dell'asta
*   Ld()   matrice di rotazione
*
* Variabili di uscita:
*   Kg()   matrice di rigidezza globale
*
* Procedure utilizzate:
*   Kf.Inc
*   Kg.f
*   T.Inc

```

```

-----
SUB Kg.Inc (E, A, I, L, Ld(), Kg())

```

```

DIM Kf(3, 3), T(6, 3), Kloc(6, 6)
CALL Kf.Inc(E, A, I, L, Kf())
CALL T.Inc(L, T())
CALL Kg.f(Ld(), T(), Kf(), Kg())

```

END SUB

```

===== Kg.IncTag =====
.
.
. Procedura per la definizione della matrice di rigidezza globale
. per un'asta perfettamente incastrata ai nodi di estremita'
. tenendo conto della deformazione a taglio
.
. Variabili in ingresso:
. E      modulo di elasticita' normale
. G      modulo di elasticita' tangenziale
. CHI    fattore di taglio
. A      area della sezione trasversale dell'asta
. I      momento d'inerzia della sezione trasversale dell'asta
. L      lunghezza dell'asta
. Ld()   matrice di rotazione
.
. Variabili in uscita:
. Kg()   matrice di rigidezza globale
.
. Procedure utilizzate:
. Kf.IncTag
. Kg.f
. T.Inc
.
-----

```

SUB Kg.IncTag (E, G, CHI, A, I, L, Ld(), Kg())

```

DIM Kf(3, 3), T(6, 3), Kloc(6, 6)
CALL Kf.IncTag(E, G, CHI, A, I, L, Kf())
CALL T.Inc(L, T())
CALL Kg.f(Ld(), T(), Kf(), Kg())

```

END SUB

```

===== Kg.1 =====
.
.
. Procedura generale per la definizione della matrice di rigidezza globale
. a partire dalla matrice di rigidezza nel sistema di riferimento locale
.
. Variabili di ingresso:
. Ld()   matrice di rotazione
. Kloc() matrice di rigidezza locale
.
. Variabili di uscita:
. Kg()   matrice di rigidezza globale
.
. Procedure utilizzate:
. ProdottoMat
. TrasponeMat
.
-----

```

SUB Kg.1 (Ld(), Kloc(), Kg())

```

DIM LdT(6, 6)
CALL ProdottoMat(Ld(), Kloc(), Kg())
CALL TrasponeMat(Ld(), LdT())

```

```

CALL ProdottoMat(Kg(), LdT(), Kg())
END SUB
' ===== Kg.NodRig =====
'
'   Procedura per la determinazione della matrice di rigidezza globale
'   nel caso di nodi rigidi non puntiformi
'   (asta con tratti rigidi non coassiali)
'
'   Variabili di ingresso:
'   E      modulo di elasticita' normale
'   A      area della sezione trasversale
'   I      momento d'inerzia della sezione trasversale
'   L      lunghezza dell'asta
'   Sx     coefficiente lunghezza x tratto sinistro
'   Sy     coefficiente lunghezza y tratto sinistro
'   Tx     coefficiente lunghezza x tratto destro
'   Ty     coefficiente lunghezza y tratto destro
'   Ld()   matrice di rotazione
'
'   Variabili di uscita:
'   Kg()   matrice di rigidezza globale
'
'   Procedure utilizzate:
'   Kf.NodRig
'   Kg.f
'   T.Inc
' -----
SUB Kg.NodRig (E, A, I, L, Sx, Sy, Tx, Ty, Ld(), Kg())
  DIM Kf(3, 3), T(6, 3), Kloc(6, 6)
  CALL Kf.NodRig(E, A, I, L, Sx, Sy, Tx, Ty, Kf())
  CALL T.Inc(L, T())
  CALL Kg.f(Ld(), T(), Kf(), Kg())
END SUB

```

```

' ===== Kg.Rig =====
'
'   Procedura per la definizione della matrice di rigidezza globale
'   per un'asta perfettamente incastrata ai nodi di estremita'
'   con tratti rigidi di estremita'
'
'   Variabili di ingresso:
'   E      modulo di elasticita' normale
'   A      area della sezione trasversale dell'asta
'   I      momento d'inerzia della sezione trasversale dell'asta
'   L      lunghezza dell'asta
'   S      coefficiente lunghezza tratto rigido sinistro
'   T      coefficiente lunghezza tratto rigido destro
'   Ld()   matrice di rotazione
'
'   Variabili di uscita:
'   Kg()   matrice di rigidezza globale
'
'   Procedure utilizzate:

```

```

.      Kf.Rig
.      Kg.f
.      T.Inc
.
.-----
SUB Kg.Rig (E, A, I, L, S, T, Ld(), Kg())

  DIM Kf(3, 3), T(6, 3), Kloc(6, 6)
  CALL Kf.Rig(E, A, I, L, S, T, Kf())
  CALL T.Inc(L, T())
  CALL Kg.f(Ld(), T(), Kf(), Kg())

```

END SUB

```

.===== Kg.Wnk =====
.
.  Procedura per la definizione della matrice di rigidezza globale
.  per un'asta su suolo elastico alla Winkler
.
.  Variabili di ingresso:
.  K      costante di sottofondo
.  B      larghezza trave
.  E      modulo di elasticita' normale
.  A      area della sezione trasversale dell'asta
.  I      momento d'inerzia della sezione trasversale dell'asta
.  L      lunghezza dell'asta
.  Ld()   matrice di rotazione
.
.  Variabili di uscita:
.  Kg()   matrice di rigidezza globale
.
.  Procedure utilizzate:
.  Kf.Wnk
.  Kg.f
.  T.Wnk
.-----

```

```

SUB Kg.Wnk (K, B, E, A, I, L, Ld(), Kg())

  DIM Kf(5, 5), T(6, 5), Kloc(6, 6)
  CALL Kf.Wnk(K, B, E, A, I, L, Kf())
  CALL T.Wnk(T())
  CALL Kg.f(Ld(), T(), Kf(), Kg())

```

END SUB

```

.===== Kl. =====
.
.  Procedura generale per la definizione della matrice di rigidezza locale
.  a partire dalla matrice di rigidezza fondamentale
.
.  Variabili di ingresso:
.  T()    matrice di trasformazione
.  Kf()   matrice di rigidezza fondamentale
.
.  Variabili di uscita:
.  Kloc() matrice di rigidezza locale

```

```

,
,   Procedure utilizzate:
,     LimitiMat
,     ProdottoMat
,     TrasponeMat
,
, -----
,
SUB Kl. (T(), Kf(), Kloc())

    CALL LimitiMat(Kf(), DK, LRK, LCK, NRK, NCK)
    DIM K1(6, NRK), TT(NRK, 6)

    CALL ProdottoMat(T(), Kf(), K1())
    CALL TrasponeMat(T(), TT())
    CALL ProdottoMat(K1(), TT(), Kloc())

END SUB

, ===== Sl.Cer.Qunif =====
,
,   Procedura per il calcolo delle azioni di incastro perfetto
,   per un'asta incernierata agli estremi: carico uniforme
,
,   Variabili di ingresso:
,     ES      estremo incernierato
,             1 = primo estremo
,             2 = secondo estremo
,             3 = entrambi gli estremi
,     Q      carico uniformemente distribuito ortogonale all'asse
,     N      carico uniformemente distribuito parallelo all'asse
,     L      lunghezza dell'asta
,
,   Variabili di uscita:
,     Sl()   vettore azioni di incastro perfetto
,
,   Procedure utilizzate:
,     Azzeramat
,
, -----
,
SUB Sl.Cer.Qunif (ES, Q, N, L, Sl())

    CALL Azzeramat(Sl())
    Sl(1, 1) = -N * L / 2
    Sl(4, 1) = Sl(1, 1)
    SELECT CASE ES
    CASE 1
        Sl(2, 1) = -3 / 8 * L * Q
        Sl(5, 1) = -5 / 8 * L * Q
        Sl(6, 1) = 1 / 8 * L ^ 2 * Q
    CASE 2
        Sl(2, 1) = -5 / 8 * L * Q
        Sl(3, 1) = -1 / 8 * L ^ 2 * Q
        Sl(5, 1) = -3 / 8 * L * Q
    CASE 3
        Sl(2, 1) = -1 / 2 * L * Q
        Sl(5, 1) = Sl(2, 1)
    END SELECT

```

END SUB

```

===== Sl.Inc.DT =====
*
*
* Procedura per il calcolo delle azioni di incastro perfetto
* per un'asta perfettamente incastrata ai nodi di estremita':
* distorsione termica trapezia
*
* Variabili di ingresso:
*   DTu   variazione termica uniforme
*   DTf   variazione termica a farfalla
*   E     modulo di elasticita' normale
*   A     area della sezione trasversale dell'asta
*   I     momento d'inerzia della sezione trasversale dell'asta
*   H     altezza della sezione trasversale dell'asta
*   At    coefficiente di dilatazione termica
*
* Variabili di uscita:
*   Sl()  vettore azioni di incastro perfetto
*
* Procedure utilizzate:
*   ==
*
-----

```

SUB Sl.Inc.DT (DTu, DTf, E, A, I, H, At, Sl())

```

Sl(1, 1) = E * A * At * DTu
Sl(2, 1) = 0
Sl(3, 1) = -E * I * At / H * DTf
Sl(4, 1) = -Sl(1, 1)
Sl(5, 1) = 0
Sl(6, 1) = Sl(3, 1)

```

END SUB

```

===== Sl.Inc.FM =====
*
*
* Procedura per il calcolo delle azioni di incastro perfetto
* per un'asta perfettamente incastrata ai nodi di estremita':
* forza F e coppia M concentrate
*
* Variabili di ingresso:
*   Fx    componente della forza concentrata parallela all'asta
*   Fy    componente della forza concentrata normale all'asta
*   Mz    coppia concentrata
*   XO    distanza dal primo estremo delle azioni concentrate
*   L     lunghezza dell'asta
*
* Variabili di uscita:
*   Sl()  vettore azioni di incastro perfetto
*
* Procedure utilizzate:
*   ==
*
-----

```

SUB Sl.Inc.FM (Fx, Fy, Mz, XO, L, Sl())

```

IF XO < 0 OR XO > L THEN
  PRINT "ERRORE 1 in Sl.Inc.FM - valori dell'ascissa non accettabili"
END
END IF

```

```

A = XO / L
B = 1 - A
Sl(1, 1) = -B * Fy
Sl(2, 1) = -B ^ 2 * (1 + 2 * A) * Fx + 6 * A * B / L * Mz
Sl(3, 1) = -A * B ^ 2 * L * Fx + B * (2 * A - B) * Mz
Sl(4, 1) = -A * Fy
Sl(5, 1) = -A ^ 2 * (1 + 2 * B) * Fx - 6 * A * B / L * Mz
Sl(6, 1) = A ^ 2 * B * L * Fx + A * (2 * B - A) * Mz

```

```
END SUB
```

```

===== Sl.Inc.Qunif =====
,
,
,   Procedura per il calcolo delle azioni di incastro perfetto
,   per un'asta perfettamente incastrata ai nodi di estremita':
,   carico uniformemente distribuito
,
,   Variabili di ingresso:
,   Q   carico uniformemente distribuito ortogonale all'asta
,   N   carico uniformemente distribuito parallelo all'asta
,   L   lunghezza dell'asta
,
,   Variabili di uscita:
,   Sl() vettore azioni di incastro perfetto
,
,   Procedure utilizzate:
,   ==
,
,-----

```

```
SUB Sl.Inc.Qunif (Q, N, L, Sl())
```

```

Sl(1, 1) = -N * L / 2
Sl(2, 1) = -Q * L / 2
Sl(3, 1) = -Q * L ^ 2 / 12
Sl(4, 1) = Sl(1, 1)
Sl(5, 1) = Sl(2, 1)
Sl(6, 1) = -Sl(3, 1)

```

```
END SUB
```

```

===== Sl.Inc.Qvar =====
,
,
,   Procedura per il calcolo delle azioni di incastro perfetto
,   per un'asta perfettamente incastrata ai nodi di estremita':
,   carico distribuito con andamento lineare su parte dell'asta
,
,   Variabili di ingresso:
,   Q1  valore del carico a sinistra
,   Q2  valore del carico a destra
,   L   lunghezza dell'asta
,   X1  distanza dal primo estremo dell'inizio carico
,   X2  distanza dal primo estremo della fine carico
,

```

```

*   Variabili di uscita:
*   S1()    vettore azioni di incastro perfetto
*
*   Procedure utilizzate:
*   =====
*
* -----
SUB Sl.Inc.Qvar (Q1, Q2, L, X1, X2, S1())
  IF X1 < 0 OR X1 > L OR X2 < 0 OR X2 > L THEN
    PRINT "ERRORE 1 in Sl.Inc.Qvar - valori delle ascisse non accettabili"
    END
  END IF
  IF X1 > X2 THEN
    PRINT "ERRORE 2 in Sl.Inc.Qvar - ordine delle ascisse non accettabile"
    END
  END IF

  r1 = 3 / 2 - (X1 + X2 / 2) / L
  s1 = 3 - 2 * r1
  r2 = (X2 + X1 / 2) / L
  s2 = 3 - 2 * r2
  T = (X2 - X1) / L

  r = r1: S = s1
  GOSUB FGH
  f1 = F: g1 = G: h1 = H

  r = r2: S = s2
  GOSUB FGH
  f2 = F: g2 = G: h2 = H

  S1(1, 1) = 0
  S1(2, 1) = -(20 * r1 * T + h1) / 60 * L * Q1 - (10 * s2 * T - h2) / 60 * L * Q2
  S1(3, 1) = -f1 * L ^ 2 / 20 * Q1 - g2 * L ^ 2 / 30 * Q2
  S1(4, 1) = 0
  S1(5, 1) = -(10 * s1 * T - h1) / 60 * L * Q1 - (20 * r2 * T - h2) / 60 * L * Q2
  S1(6, 1) = g1 * L ^ 2 / 30 * Q1 + f2 * L ^ 2 / 20 * Q2

  EXIT SUB

FGH:
  F = T / 27 * (40 * r ^ 2 * S + T ^ 2 * (2 * T - 30 * r + 15))
  G = T / 18 * (20 * r * S ^ 2 - T ^ 2 * (2 * T - 30 * r + 30))
  H = 3 * F - 2 * G
  RETURN

END SUB

* ===== Sl.Rig.Qunif =====
*
*   Procedura per il calcolo delle azioni di incastro perfetto
*   per un'asta perfettamente incastrata ai nodi di estremita'
*   avente tratti rigidi di estremita': carico uniforme
*
*   Variabili di ingresso:
*   Q    carico uniformemente distribuito ortogonale all'asta
*   N    carico uniformemente distribuito parallelo all'asta

```

```

.      L      lunghezza asta
.      S      coefficiente lunghezza tratto rigido sinistro
.      T      coefficiente lunghezza tratto rigido destro
.
.  Variabili di uscita:
.      Sl()   vettore azioni di incastro perfetto
.
.  Procedure utilizzate:
.  =====
.
.-----
SUB Sl.Rig.Qunif (Q, N, L, S, T, Sl())

      Z1 = (1 - S - T) ^ 2 + 6 * S * (1 - S - T) + 6 * S ^ 2
      Z2 = (1 - S - T) ^ 2 + 6 * T * (1 - S - T) + 6 * T ^ 2

      Sl(1, 1) = -N * L / 2
      Sl(2, 1) = -(1 + S - T) / 2 * L * Q
      Sl(3, 1) = -Z1 * L ^ 2 / 12 * Q
      Sl(4, 1) = Sl(1, 1)
      Sl(5, 1) = -(1 - S + T) / 2 * L * Q
      Sl(6, 1) = Z2 * L ^ 2 / 12 * Q

END SUB

.  ===== Sl.Wnk.Qunif =====
.
.  Procedura per il calcolo delle azioni di incastro perfetto
.  per un'asta su suolo elastico alla Winkler: carico uniforme
.
.  Variabili di ingresso:
.      Q      carico uniformemente distribuito ortogonale all'asta
.      N      carico uniformemente distribuito parallelo all'asta
.      L      lunghezza dell'asta
.      K      costante di sottofondo
.      B      larghezza della trave
.      Kloc() matrice di rigidezza locale
.
.  Variabili di uscita:
.      Sl()   vettore azioni di incastro perfetto
.
.  Procedure utilizzate:
.      Azzeramat
.      ProdottoMat
.
.-----
SUB Sl.Wnk.Qunif (Q, N, L, K, B, Kloc(), Sl())

      CALL Azzeramat(Sl())
      Sl(2, 1) = -1 / (K * B) * Q
      Sl(5, 1) = Sl(2, 1)
      CALL ProdottoMat(Kloc(), Sl(), Sl())
      Sl(1, 1) = Sl(1, 1) - N * L / 2
      Sl(4, 1) = Sl(4, 1) - N * L / 2

END SUB

```

```

* ===== T.Cer =====
*
* Procedura per la definizione della matrice di trasformazione
* per un'asta incernierata agli estremi
*
* Variabili di ingresso:
*   ES      estremo incernierato
*           1 = primo estremo
*           2 = secondo estremo
*           3 = entrambi gli estremi
*   L       lunghezza dell'asta
*
* Variabili di uscita:
*   T()     matrice di trasformazione
*
* Procedure utilizzate:
*   AzzerMat
*
* -----

```

```

SUB T.Cer (ES, L, T())

```

```

CALL AzzerMat(T())

```

```

SELECT CASE ES

```

```

CASE 1

```

```

T(1, 2) = -1

```

```

T(2, 1) = 1 / L

```

```

T(4, 2) = 1

```

```

T(5, 1) = -1 / L

```

```

T(6, 1) = 1

```

```

CASE 2

```

```

T(1, 2) = -1

```

```

T(2, 1) = 1 / L

```

```

T(3, 1) = 1

```

```

T(4, 2) = 1

```

```

T(5, 1) = -1 / L

```

```

CASE 3

```

```

T(1, 1) = -1

```

```

T(4, 1) = 1

```

```

END SELECT

```

```

END SUB

```

```

* ===== T.Inc =====
*
* Procedura per la definizione della matrice di trasformazione
* per un'asta perfettamente incastrata ai nodi di estremita'
*
* Variabili di ingresso:
*   L       lunghezza dell'asta
*
* Variabili di uscita:
*   T()     matrice di trasformazione
*
* Procedure utilizzate:
*   AzzerMat
*
* -----

```

```
SUB T.Inc (L, T())
```

```
CALL Azzeramat(T())
T(1, 3) = -1
T(2, 1) = 1 / L
T(2, 2) = 1 / L
T(3, 1) = 1
T(4, 3) = 1
T(5, 1) = -1 / L
T(5, 2) = -1 / L
T(6, 2) = 1
```

```
END SUB
```

```

' ===== T.Wnk =====
'
' Procedura per la definizione della matrice di trasformazione
' per un'asta su suolo elastico alla Winkler
'
' Variabili di ingresso:
'   ==
'
' Variabili di uscita:
'   T()      matrice di trasformazione
'
' Procedure utilizzate:
'   Azzeramat
'
' -----
'
'

```

```
SUB T.Wnk (T())
```

```
CALL Azzeramat(T())
T(1, 5) = -1
T(2, 1) = 1
T(3, 2) = 1
T(4, 5) = 1
T(5, 3) = 1
T(6, 4) = 1
```

```
END SUB
```

3. Esempio: diagramma delle caratteristiche di sollecitazione

3.1. Generalità

La conoscenza dell'andamento delle caratteristiche di sollecitazione lungo le aste è di fondamentale importanza per il progettista strutturale durante le fasi di verifica delle sezioni e di disposizione delle armature in strutture in cemento armato. La graficizzazione di tale andamento mediante diagrammi ne fornisce una visione immediata e globale che agevola

l'individuazione delle sezioni più sollecitate ed inoltre consente una valutazione qualitativa del comportamento della struttura, essenziale per esprimere un giudizio di accettazione dei risultati e per valutare la correttezza dei dati forniti al calcolatore.

Per i motivi citati si è ritenuto utile preparare una procedura, denominata *Diagramma*, che consente di disegnare sullo schermo il diagramma dello sforzo normale, del momento flettente o del taglio in un'asta. I valori assunti da questi enti alla generica ascissa x sono univocamente definiti dalla geometria dell'asta, dai carichi su essa agenti e dai valori delle caratteristiche di sollecitazione in un estremo (ottenuti come risultato della soluzione dello schema iperstatico).

Nel definire i possibili carichi agenti sull'asta, si è ipotizzata la presenza di:

- un carico uniformemente distribuito, con una componente parallela ed una ortogonale all'asse, applicato a tutta l'asta;
- due carichi distribuiti ortogonali all'asse, variabili con legge lineare ed applicati a parte dell'asta;
- tre insiemi di forze e momenti concentrati applicati in punti distinti all'interno dell'asta; per semplicità operativa si è supposto che questi punti siano forniti nell'ordine in cui si succedono nell'asta, inserendo comunque un controllo per verificare che tale condizione sia rispettata.

Alla presenza di azioni concentrate corrisponde una discontinuità del diagramma delle caratteristiche di sollecitazione. Per tale motivo si è divisa l'asta in quattro tratti, individuati dagli estremi e dai punti di applicazione delle azioni. Ciascun tratto è diviso in un numero intero di intervalli, di ampiezza non superiore ad $1/n$ della lunghezza totale dell'asta. Il numero n è stato conservato nella variabile *Punti*, che nel caso esaminato è stata posta uguale a 100 per ottenere un diagramma molto accurato; valori più bassi forniscono diagrammi meno precisi, ma richiedono un minor onere computazionale e quindi minori tempi di esecuzione. I valori assunti dalle caratteristiche di sollecitazione nei punti di divisione tra gli intervalli sono valutati mediante la routine *CalcolaCarSol*, che utilizza le procedure *CarSol.AzEstr*, *CarSol.FM*, *CarSol.Qunif*, *CarSol.Qvar*. La routine *TracciaDiagr*, infine, calcola le coordinate del punto rappresentativo del valore da diagrammare, ottenute spostandosi ortogonalmente all'asta di una quantità pari al valore moltiplicato per un fattore di scala Sc (fornito come dato d'ingresso alla procedura), ed unisce il punto in esame al precedente.

Si riporta di seguito il listato della procedura *Diagramma*, che fa parte del file *DIAGRAM.BAS*. Nel listato è contenuta anche la descrizione dei parametri

di scambio, delle principali variabili interne e delle procedure richiamate dal sottoprogramma.

```

===== Diagramma =====
.
.
. Procedura per il disegno del diagramma
. del momento flettente, del taglio o dello sforzo normale
.
. NOTE: occorre essersi gia' posizionati nella modalita' grafica (SCREEN)
. ed aver definito il campo di coordinate (WINDOW)
.
. Variabili di ingresso:
. X1 ascissa del primo estremo dell'asta
. Y1 ordinata del primo estremo dell'asta
. X2 ascissa del secondo estremo dell'asta
. Y2 ordinata del secondo estremo dell'asta
. S() array che contiene le caratt. di sollecitaz. al primo estremo
. indice: 1 -> N, 2 -> M, 3 -> T
. P() array che contiene le componenti del carico uniforme sull'asta
. indice: 1 -> Px, 2 -> Py
. F() array bidimensionale che contiene le azioni concentrate
. primo indice: 1 -> Fx, 2 -> Fy, 3 -> Mz
. secondo indice: numero d'ordine della forza (1-3)
. Xf() vettore ascisse punti applicazione azioni concentrate (1-3)
. Q() array bidimensionale che contiene i carichi variabili
. primo indice: 1 -> Q1, 2 -> Q2
. secondo indice: numero d'ordine dei carichi (1-2)
. Xq() array bidimensionale che contiene le ascisse dei car. variabili
. primo indice: 1 -> X1, 2 -> X2
. secondo indice: numero d'ordine dei carichi (1-2)
. Sc fattore di scala della caratteristica da diagrammare
. CS caratteristica di sollecitazione da diagrammare:
. 1 = sforzo normale
. 2 = momento flettente
. 3 = taglio
.
. Variabili di uscita:
.
.
. Principali variabili interne:
. L lunghezza dell'asta
. X ascissa variabile
. Nx,Mx,Tx caratteristiche di sollecitazione all'ascissa X
. XP,YP coordinate del punto del diagramma
. Punti numero di punti dell'asta nei quali sono calcolati N, M, T
.
. Procedure utilizzate:
. CarSol.AzEstr
. CarSol.FM
. CarSol.Qunif
. CarSol.Qvar
.
. -----

```

SUB Diagramma (X1, Y1, X2, Y2, S(), P(), F(), Xf(), Q(), Xq(), Sc, CS)

Punti = 100

```

' calcola lunghezza asta e sue proiezioni
Dx = X2 - X1
Dy = Y2 - Y1
L = SQR(Dx ^ 2 + Dy ^ 2)
SnAl = Dy / L
CsAl = Dx / L

' disegna l'asta
LINE (X1, Y1)-(X2, Y2)

' disegna il diagramma
PSET (X1, Y1)
Xprec = 0
FOR J = 1 TO 4
  IF J = 4 THEN
    Xsucc = L
  ELSE
    Xsucc = Xf(J)
  END IF
  DeltaX = Xsucc - Xprec
  IF Xsucc <> 0 AND DeltaX <> 0 THEN
    IF DeltaX < 0 THEN
      PRINT "ERRORE 1 in Diagramma - le azioni concentrate non sono assegnate"
      PRINT "                                nell'ordine in cui si succedono lungo"
      PRINT "                                l'asta"
    END
  END IF
  SZ = INT(DeltaX / L * Punti + 1)
  FOR S = 0 TO SZ
    X = Xprec + S * DeltaX / SZ
    GOSUB CalcolaCarSol
    GOSUB TracciaDiagr
  NEXT S
  Xprec = Xsucc
END IF
NEXT J
LINE -(X2, Y2)
EXIT SUB

```

CalcolaCarSol:

```

' per azioni di estremita'
CALL Carsol.AzEstr(-S(1), S(3), -S(2), X, L, Nx, Mx, Tx)

' per carichi uniformemente distribuiti
IF P(1) <> 0 OR P(2) <> 0 THEN
  CALL CarSol.Qunif(P(2), P(1), X, L, Sn, Mf, Ta)
  Nx = Nx + Sn
  Mx = Mx + Mf
  Tx = Tx + Ta
END IF

' per forze e coppie concentrate
FOR J = 1 TO 3
  IF F(1, J) <> 0 OR F(2, J) <> 0 OR F(3, J) <> 0 THEN
    CALL Carsol.FM(F(1, J), F(2, J), F(3, J), Xf(J), X, L, Sn, Mf, Ta)
  END IF
NEXT J

```

```

        Nx = Nx + Sn
        Mx = Mx + Mf
        Tx = Tx + Ta
    END IF
NEXT J

' per carichi distribuiti con legge lineare
FOR J = 1 TO 2
    IF Q(1, J) <> 0 OR Q(2, J) <> 0 THEN
        CALL Carsol.Qvar(Q(1, J), Q(2, J), Xq(1, J), Xq(2, J), X, L, Mf, Ta)
        Mx = Mx + Mf
        Tx = Tx + Ta
    END IF
NEXT J

RETURN

TracciaDiagr:

SELECT CASE CS
CASE 1
    V = Nx * Sc
CASE 2
    V = -Mx * Sc
CASE 3
    V = Tx * Sc
END SELECT
XP = X1 + X * CsAl - V * SnAl
YP = Y1 + X * SnAl + V * CsAl
LINE -(XP, YP)

RETURN

END SUB

```

3.2. Applicazione

La procedura Diagramma può essere utilizzata ciclicamente per tracciare il diagramma delle caratteristiche della sollecitazione per un intero schema intelaiato. Un esempio di ciò è costituito dal programma principale del file `DIAGRAM.BAS`, listato nelle pagine seguenti. Esso legge i dati relativi ad uno schema geometrico e a due schemi di carico e consente di scegliere quale caratteristica diagrammare, e per quale schema di carico la si vuole. Poiché il programma ha solo finalità esemplificative, non ci si è curati particolarmente della fase di input (i dati sono contenuti nello stesso programma in istruzioni `DATA`) né della individuazione di criteri generali per la scelta del fattore di scala (fornito anch'esso come dato). Il programma può comunque essere usato dal lettore sostituendo il blocco dati con nuovi valori, oppure adattato per una lettura dei dati dal disco o, meglio ancora, trasformato

in un sottoprogramma che operi in coda al programma di calcolo ricevendo automaticamente da esso le informazioni necessarie per la diagrammazione.

I dati utilizzati nel programma si riferiscono al portale mostrato in figura 4.1, soggetto a due schemi di carico. Il primo prevede un carico verticale uniformemente distribuito sui traversi, il secondo due forze concentrate verticali su un'asta ed una coppia concentrata al centro di un'altra asta. Non si sono quindi sfruttate tutte le potenzialità della procedura Diagramma (forze in tre punti distinti, carichi variabili), e questo è evidenziato dalla presenza di numerosi zeri tra i dati.

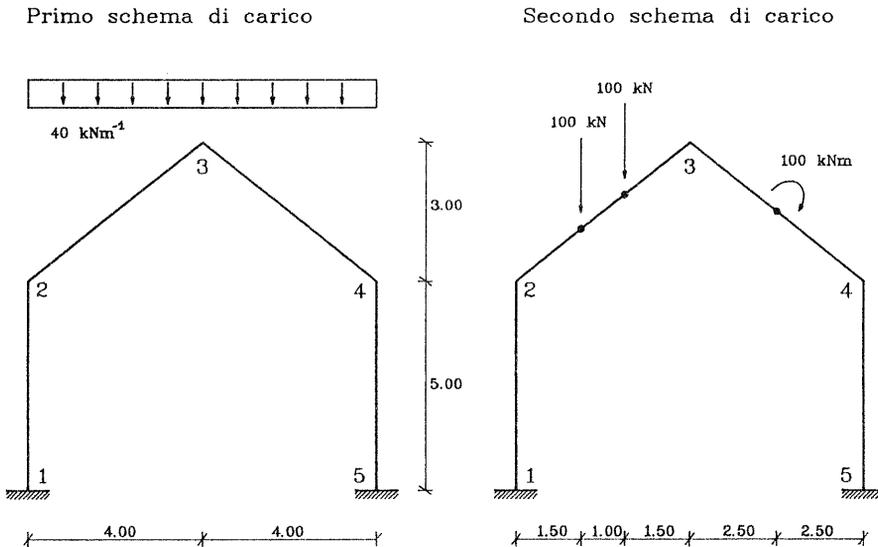


Fig. 4.1 — Portale soggetto a due schemi di carico

Nella figura 4.2 sono riuniti i sei diagrammi ottenuti come risultato dell'elaborazione grafica. Si noti che il taglio e lo sforzo normale mantengono lo stesso segno indipendentemente dal verso di percorrenza dell'asta (cioè da quale nodo è scelto come primo estremo); i loro valori positivi sono riportati nel verso positivo dell'asse locale y e quindi da un lato o dall'altro dell'asta a seconda di qual'è il primo e quale il secondo estremo.

Invece il momento flettente è riportato sempre dal lato delle fibre tese, indipendentemente dal verso dell'asta, perché invertendo l'ordine dei nodi si modifica anche il segno di esso.

Primo schema di carico

Secondo schema di carico

DIAGRAMMA DELLO SFORZO NORMALE

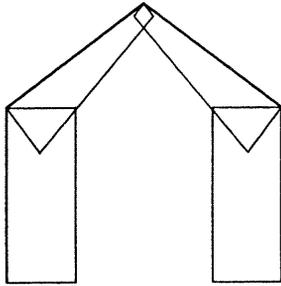


DIAGRAMMA DELLO SFORZO NORMALE

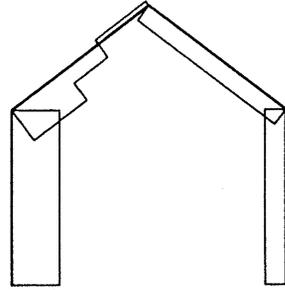


DIAGRAMMA DEL MOMENTO FLETTENTE

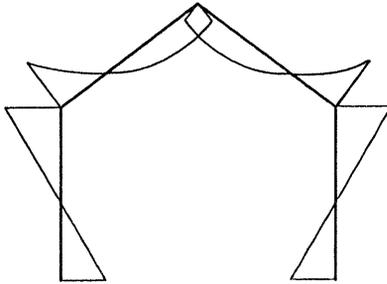


DIAGRAMMA DEL MOMENTO FLETTENTE

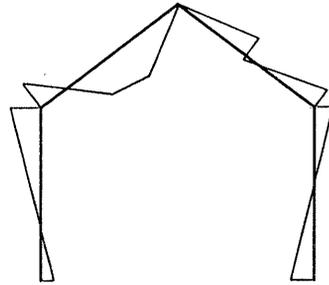


DIAGRAMMA DEL TAGLIO

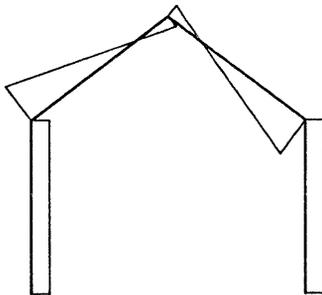


DIAGRAMMA DEL TAGLIO

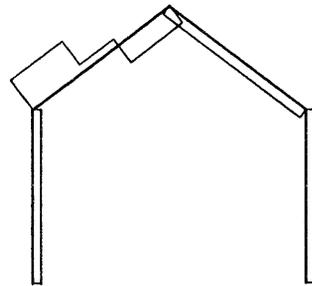


Fig. 4.2 — Diagramma delle caratteristiche di sollecitazione del portale


```

PRINT "                                0 - fine esecuzione"
DO
  K$ = INKEY$
  LOOP UNTIL K$ <> ""
  K = ASC(K$) - 48
  SELECT CASE K
    CASE 1, 2, 3
      RESTORE Schema1
      CS = K
    CASE 4, 5, 6
      RESTORE Schema2
      CS = K - 3
    CASE ELSE
  END SELECT
  IF K >= 1 AND K <= 6 THEN
    SCREEN 2
    WINDOW (Xmin, Ymin)-(Xmax, Ymax)
    CLS
    LOCATE 1, 26
    SELECT CASE CS
      CASE 1
        PRINT "DIAGRAMMA DELLO SFORZO NORMALE"
      CASE 2
        PRINT "DIAGRAMMA DEL MOMENTO FLETTENTE"
      CASE 3
        PRINT "          DIAGRAMMA DEL TAGLIO"
    END SELECT
    ' lettura dei carichi sulle aste e disegno diagramma
    FOR J = 1 TO JZ
      READ S(1), S(2), S(3)
      READ P(1), P(2)
      FOR N = 1 TO 3
        READ F(1, N), F(2, N), F(3, N), Xf(N)
      NEXT N
      FOR N = 1 TO 2
        READ Q(1, N), Q(2, N), Xq(1, N), Xq(2, N)
      NEXT N
      X1 = X(E1(J))
      Y1 = Y(E1(J))
      X2 = X(E2(J))
      Y2 = Y(E2(J))
      CALL Diagramma(X1, Y1, X2, Y2, S(), P(), F(), Xf(), Q(), Xq(), Sc, CS)
    NEXT J
    LOCATE 24, 1
    PRINT "Per continuare premi un tasto qualsiasi";
    DO
      K$ = INKEY$
    LOOP UNTIL K$ <> ""
  END IF

  LOOP UNTIL K = 0
  SCREEN 0
  END

DATA 5,4
DATA 0,0,0,5,4,8,8,5,8,0
DATA 1,2,2,3,3,4,4,5
DATA -6,14,-3,10
DATA .01

```

Schema1:

DATA -200,128.30,-58.14,0,0
DATA 0,0,0,0,0,0,0,0,0,0,0
DATA 0,0,0,0,0,0,0,0,0
DATA -166.51,-162.40,125.12,-24,-32
DATA 0,0,0,0,0,0,0,0,0,0,0
DATA 0,0,0,0,0,0,0,0,0
DATA -46.51,63.18,34.88,24,-32
DATA 0,0,0,0,0,0,0,0,0,0,0
DATA 0,0,0,0,0,0,0,0,0
DATA -200,-162.40,58.14,0,0
DATA 0,0,0,0,0,0,0,0,0,0,0
DATA 0,0,0,0,0,0,0,0,0

Schema2:

DATA -141.43,36.89,-25.23,0,0
DATA 0,0,0,0,0,0,0,0,0,0,0
DATA 0,0,0,0,0,0,0,0,0
DATA -105.04,-89.28,98.00,0,0
DATA -60,-80,0,1.875,-60,-80,0,3.125,0,0,0,0
DATA 0,0,0,0,0,0,0,0,0
DATA -55.33,0.75,-31.72,0,0
DATA 0,0,-100,2.5,0,0,0,0,0,0,0,0
DATA 0,0,0,0,0,0,0,0,0
DATA -58.57,-57.83,25.23,0,0
DATA 0,0,0,0,0,0,0,0,0,0,0,0
DATA 0,0,0,0,0,0,0,0,0

