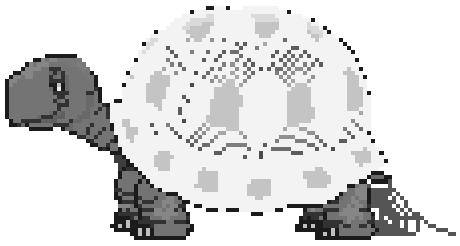
ANTONIO GRANDE

PROGRAMMAZIONE VBA E FINANZA

PROGRAMMAZIONE VBA E FINANZA

ANTONIO GRANDE



Soluzioni VBA di problemi di finanza classica e moderna

Antonio Grande
Programmazione VBA e finanza
Soluzioni VBA di problemi di finanza classica e moderna

ISBN: 978-88-907402-0-6

La versione digitale dell'opera è disponibile al sito:
http://antoniogrande.uniroma1.it

Email antonio.grande@uniroma1.it

a Pièveloce.
1955 – 2006

ABSTRACT

The VBA (Visual Basic for Application) language has among it's characteristics the capacity to interact with spreadsheet data, that has a very simple structure.
This study, after having examined the language basic notions, suggests a methodology to find out automatic solutions for classic and modern finance problems such as: the (pseudo)casual numbers generation, the price of the Asian and European Call/Put with Black-Scholes and Montecarlo methods, the efficient frontier with the Markowitz model, the download of stock prices from Internet.

ABSTRACT

Il linguaggio VBA (Visual Basic for Application) ha tra le sue caratteristiche quella di interagire con i dati del foglio elettronico che utilizza una struttura di dati concettualmente molto semplice.
Il libro, passati in rassegna i fondamentali dei linguaggio, propone le soluzioni automatiche di problemi di finanza classica e moderna tra cui citiamo: la generazione di numeri (pseudo)casuali, il calcolo di una put/call sia per le opzioni europee che per quelle asiatiche con Black-Scholes e Montecarlo, la frontiera efficiente con il modello di Markowitz, lo scaricamento delle quotazioni di titoli da Internet.

RINGRAZIAMENTI

L'anno scorso, durante un brevissimo viaggio in treno alla volta di una sede distaccata dell'università, la professoressa Giusy Bruno ed il sottoscritto, discutemmo sul progetto di una serie di appunti per la soluzione di problemi di finanza moderna con un linguaggio di programmazione.

L'idea di questo libro nasce da lì. Voglio ringraziare in modo particolare la professoressa Bruno per i consigli sulla scaletta degli argomenti e per la pazienza che ha avuto nelle occasioni in cui le ho chiesto chiarimenti e quant'altro.

La responsabilità di quanto è scritto nel libro è soltanto dell'autore.

Un altro sentito ringraziamento lo devo alle persone che si occupano di sviluppare e diffondere LaTeX, il software utilizzato per scrivere il libro, tra i quali ricordo: (mitico) Donald Erwin Knuth, Lorenzo Pantieri [5], André Miede [7] (cui si deve lo stile tipografico di questo libro), Robert Bringhurst [2], Tommaso Gordini [5] nonché la comunità LaTeX.

Antonio Grande

Roma, maggio 2012

INDICE

ELENCO DELLE FIGURE

Parte I

FONDAMENTALI VBA

INTRODUZIONE

CONTENUTO

Visual Basic for Application (VBA) è il linguaggio di programmazione Microsoft per la suite Office (Word, Excel, Power Point, Access).

Questo libro si occupa della soluzione di alcuni problemi di finanza classica e moderna con la programmazione in VBA del foglio elettronico Excel. Non è un manuale di Visual Basic.

Nella parte iniziale, capitoli 1-8, vengono illustrate la caratteristiche principali del linguaggio corredate da numerosi esempi. Al termine di ciascun capitolo è presente un paragrafo di esercizi relativi agli argomenti trattati.

I capitoli 9 e 10 sono dedicati a due argomenti di rilievo: le variabili con indice e le operazioni con le matrici.

Dal capitolo 11 in poi sono presentate le soluzioni di alcuni tra i principali problemi di finanza. Queste le soluzioni sono improntate alla generalizzazione del problema. Ricorrendo ad una facile semplificazione

Alcune importanti precisazioni che riguardano lo "stile" con cui sono stati scritti i programmi. Per motivi legati alle finalità del libro, di carattere essenzialmente didattico, si tenga presente che:

- le istruzioni non sempre cominciano a partire dalla prima colonna ma sono disposte secondo una tabulazione (allineamento orizzontale) particolare. Questa tecnica, riconducibile alla "programmazione strutturata" , si utilizza per facilitare la leggibilità del programma ma non è assolutamente vincolante;

- sempre al fine di miglioramento della leggibilità, abbiamo evitato di ricorrere ad istruzioni o artifici di programmazione. In altre parole si è preferito scrivere i programmi in uno stile orientato alla facilità di comprensione più che alla velocità di esecuzione. Si tenga presente che questo approccio porta, di solito, ad un aumento del numero di istruzioni.

VERSIONE DEL PROGRAMMA

Tutto il codice del libro è stato verificato con il programma Microsoft Office 2003. La versione più recente di questo programma è la 2010.

SCARICAMENTO DEI PROGRAMMI

La versione PDF del libro ed il codice dei programmi sono disponibili all'indirizzo: http://antoniogrande.uniroma1.it

CONVENZIONI TIPOGRAFICHE

Nel testo del libro sono state adottate le seguenti convenzioni tipografiche:

- quello che segue è lo stile con cui si fa riferimento ad una cartella, in questo caso alla cartella `Cartel1.xls`;
- quella di fianco è un esempio di sequenza di due comandi del menu comando1 → comando2;

1

L'AMBIENTE VBA

Il linguaggio *Visual Basic (for) Application*, VBA nel seguito, è una evoluzione del Basic, il linguaggio in dotazione ai personal computer di prima generazione (inizi degli anni '80). VBA è integrato nel programma Microsoft Office che, come sappiamo, comprende i programmi Word, Excel, Power Point e Access. Tra le sue caratteristiche ricordiamo anche quelle di ammettere la programmazione *ad oggetti* (l'argomento sarà discusso in un apposito capitolo).

1.1 L'IDE DI VBA

L'ambiente che utilizzeremo per la scrittura dei programmi si presenta sotto forma di interfaccia grafica a finestre di tipo IDE o *Integrated Development Environment* (ambiente di sviluppo integrato). In una interfaccia di questo tipo, si dispone di un insieme *intregrato* di strumenti atti a facilitare il programmatore nella scrittura dei programmi. Gli strumenti dell'IDE VBA sono:

- l'editore di testi ovvero un ambiente che facilita la scrittura delle istruzioni;
- il debugger ossia lo strumento che facilita il controllo delle istruzioni;
- un sistema per la navigazione tra i componenti del linguaggio.

La visualizzazione di questa finestra, chiamata anche finestra VBA, si ottiene con i tasti Alt+F11 durante una sessione Excel. La vediamo riprodotta nella figura 1.1. Da qui, come sarà chiarito meglio nel prosieguo, potremo accedere a comandi, icone e quant'altro sia utile alla memorizzazione, la scrittura, l'esecuzione ed il controllo (debugging) dei programmi. Può essere suddivisa idealmente in due parti: una superiore ed una inferiore e ciascuna di queste è divisibile in ulteriori parti.

Nella zona superiore si trova la barra del menu e, sotto di questa, la barra degli strumenti. Nella parte inferiore, suddivisa in tre sottofinestre ridimensionabili e fluttanti, possiamo individuare:[1]

- la finestra di gestione dei progetti (il suo titolo è Progetto - VBAProject);

1 se queste finestre non fossero visibili eseguire, per ciascuna di esse i comandi del menu: Visualizza → Gestione progetti per la prima, Visualizza → Finestra proprietà per la seconda, Visualizza → Codice per la terza.

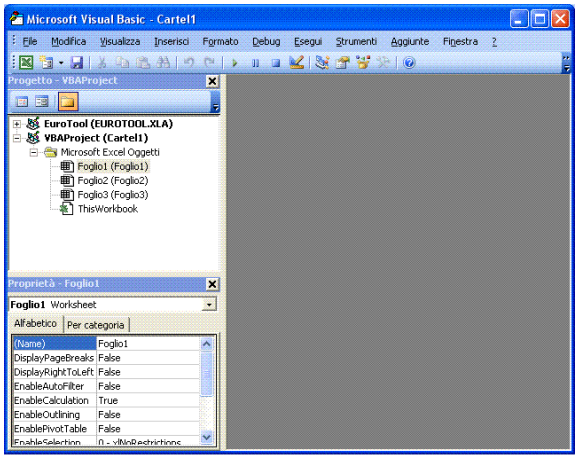


Figura 1.1: La finestra Visual Basic

- la finestra delle proprietà (il suo titolo è Proprietà - Foglio1);
- la finestra del codice (non visibile nella figura occupa normalmente la parte in grigio).

1.2 I COMANDI DEL MENU

La barra dei menu è composta da un insieme di comandi che, al momento attuale, non è il caso di conoscere in modo approfondito. La funzione dei comandi che la compongono sarà chiarita al momento in cui se ne presenterà l'occasione.

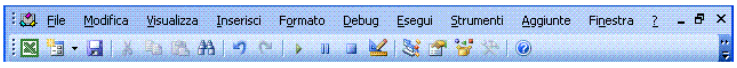


Figura 1.2: La barra dei menu (sopra) e la barra degli strumenti (sotto)

1.3 LA BARRA DEGLI STRUMENTI

La barra degli strumenti si trova sotto quella dei comandi. E' chiamata Standard in quanto esistono altre barre di strumenti che per semplicità, sono visualizzate a richiesta (si tratta delle barre Modifica, Debug e User Form). Al passaggio del mouse sopra le icone che la compongono, si visualizza il titolo corrispondente. Presentiamo una elencazione di quelli che necessitano di approfondimento in base all'ordine di visualizzazione.

Visualizza Microsoft Excel visualizza la finestra con la cartella Excel.

Inserisci User Form la sua funzione verrà commentata nel capitolo appositamente dedicato.

Salva salva la cartella di lavoro insieme a tutto cio' che si trova nella finestra dell'editor. Si tenga presente, se siamo partiti da una cartella vuota, che alla cartella di lavoro viene attibuito in automatico un nome costituito dal prefisso Cartel seguito da un numero intero progressivo. Se vogliamo attribuire un nome mnemonico alla cartella corrente si consiglia di non usare questa icona per il salvataggio. In questo caso è necessario ritornare nell'ambiente Excel e ricorrere ai comandi del menu File → Salva con nome.

Taglia

Copia

Incolla

Trova

Impossibile annullare

Impossibile ripetere

Esegui Sub/User Form si usa per eseguire un programma VBA; se ne parlerà nel prossimo paragrafo.

Interrompi sospende l'esecuzione del programma.

Ripristina si utilizza quando l'esecuzione del programma si arresta a causa di un errore. In questo caso viene evidenziata l'istruzione che lo ha generato e l'utente premendo questo pulsante riporta il programma allo stato iniziale (ovvero nello stato in cui si trovava prima dell'esecuzione).

Modalità progettazione

Gestione progetti visualizza la finestra corrispondente.

Finestra Proprietà visualizza la finestra corrispondente.

Visualizzatore Oggetti visualizza la finestra corrispondente.

1.4 LA FINESTRA PROGETTO

Si trova nella parte alta sinistra e visualizza un elenco dei progetti[2] sotto forma di diagramma ad albero collassabile (le cartelle attualmente aperte). Si tratta di una finestra che all'occorrenza può essere chiusa utilizzando l'icona a destra della sua barra del titolo. Per visualizzarla, oltreché nel modo descritto alla fine del paragrafo 1.1, si può ricorrere al pulsante Gestione progetti che si trova sulla barra degli strumenti di figura 1.2.

2 in prima approssimazione possiamo dire che un progetto coincide con una cartella ovvero un file di Excel.

Sotto la barra del titolo troviamo tre pulsanti. Del primo ci occuperemo ampiamente più avanti. Il secondo visualizza la cartella di lavoro (il foglio elettronico) mentre il terzo espande/comprime le cartelle che compaiono nella finestra del progetto.

1.5 LA FINESTRA PROPRIETÀ

La finestra Proprietà elenca tutte le proprietà dell'oggetto selezionato che di regola è il foglio corrente. Anche questa finestra, come la precedente, può essere chiusa. Per riaprirla basta utilizzare l'apposito pulsante che si trova a destra del pulsante Gestione progetti.

1.6 LA FINESTRA CODICE

La terza delle sottofinestre è quella c.d. del codice. In questa finestra verranno scritte le istruzioni del linguaggio VBA. Per visualizzarla si può ricorrere al tasto F7 nonchè alla sequenza di comandi del menu Visualizza → Codice.

Siamo ora in grado di scrivere il nostro primo programma che si compone delle istruzioni che riportiamo di seguito. Se facciamo attenzione a cosa accade dopo la scrittura della prima riga, ci accorgeremo che non appena avremo terminato la sua scrittura comparirà automaticamente l'istruzione di chiusura `End Sub`.

Codice 1.1: Una semplice procedura

```
Sub maschio()
    MsgBox "ciao bello!",  ,"Titolo  della   finestra "
End Sub
```

Si tratta di un semplicissimo programma (procedura in gergo VBA) chiamata `maschio`[3] che si compone di tre righe.

La prima e la terza riga contengono le istruzioni di inizio/fine programma. Le parentesi che seguono il nome della procedura vengono inserite automaticamente perché obbligatorie.

La seconda riga contiene il comando MsgBox che serve a visualizzare un messaggio all'interno di una finestra. Dopo il messaggio, racchiuso tra i primi doppi apici, vediamo due virgole separate da uno spazio (in seguito chiariremo il significato di questo) ed il titolo della finestra. Controllato di aver scritto tutto correttamente, possiamo eseguire questo programma.

L'esecuzione di una procedura si ottiene in uno dei seguenti modi[4]:

3 il nome di una procedura deve iniziare con un carattere alfabetico e può contenere soltanto lettere o numeri.

4 il cursore lampeggiante deve trovarsi sempre all'interno della procedura da eseguire.

- premendo il tasto Esegui Sub/User Form della Barra degli Strumenti;
- premendo il tasto F5;
- con i comandi del menu Esegui → Esegui Sub/UserForm.

Se non abbiamo commesso errori, ci apparirà la consueta finestra del foglio elettronico e, al suo interno, la finestra definita con l'istruzione 2 del codice. Il pulsante OK terminerà l'esecuzione ritornando alla finestra VBA.

1.7 GLI ERRORI DEL PROGRAMMA

Nel caso avessimo sbagliato qualcosa nella scrittura delle istruzioni è necessario cercare la causa dell'errore. Questa può dipendere sostanzialmente da due differenti motivi.

Per simulare il primo di questi cancelliamo una parte della seconda riga del programma (p.e. quella che segue la seconda virgola). In questo caso il comando manca di una sua parte essenziale, ovvero dell'intestazione della finestra. Non appena spostiamo il cursore dalla riga in cui ci troviamo viene visualizzata una finestra che riporta un messaggio simile a quello mostrato nella figura sottostante. Si nota

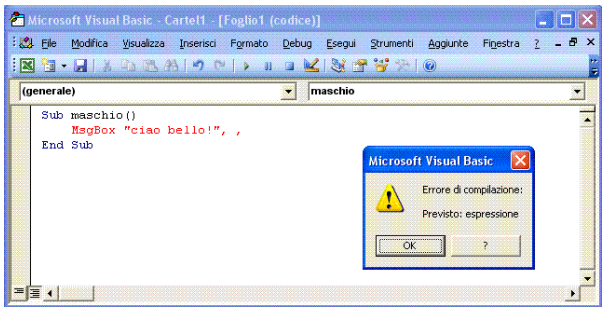


Figura 1.3: errore formale

come il comando che ha causato l'errore sia evidenziato con il colore rosso. Premuto il tasto OK possiamo ritornare alla finestra del codice per effettuare la correzione.

Diverso è il caso di istruzioni solo *formalmente* corrette. Per simulare questo, cambiamo la parola `MsgBox` in `MsgBux` e procediamo di nuovo all'esecuzione del programma.

La finestra dell'errore, simile alla precedente, riporta questa volta il messaggio Sub o Function non definita. Premuto il tasto OK ritorneremo alla finestra del codice come mostrato nella figura 1.4. Il programma è ancora in esecuzione ma è sospeso in attesa della correzione (si veda la dicitura [interruzione] che compare nella barra del titolo). Nella finestra sono evidenziate, in due diversi colori, la procedura in errore in giallo, ed il nome sconosciuto in blu. Una volta individuato l'erro-

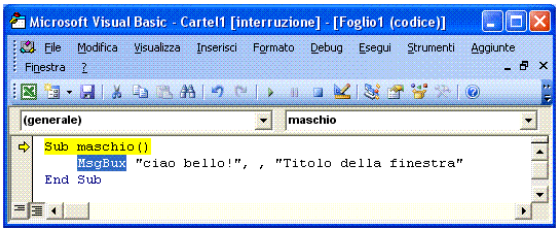


Figura 1.4: errore in esecuzione

re, procediamo alla correzione e selezioniamo nuovamente l'icona di esecuzione.

1.8 L'ESECUZIONE DI UNA MACRO

Una volta passati in rassegna i componenti fondamentali dell'interfaccia con l'ambiente di programmazione, possiamo approfondire meglio il funzionamento delle procedure.

Chiudiamo la cartella di lavoro senza preoccuparci di salvare alcunché. Apriamo quindi la cartella di lavoro `ciccio1.xls` ed utilizzando la combinazione di tasti Alt F11 visualizziamo la finestra del codice.

Il codice di ciccio1.xls.

```
Sub Main()
    femmina
End Sub
Sub maschio()
    MsgBox "ciao bello!", , "Titolo  della  Finestra"
End Sub
Sub femmina()
    MsgBox "ciao bella!", , "Titolo  della  Finestra"
End Sub
```

Il codice è organizzato in tre procedure: `Main, maschio, femmina` (anche qui tre nomi di fantasia).

La procedura `Main`, al suo interno contiene una richiamo alla procedura `femmina`. Le procedure `maschio` e `femmina` contengono l'oramai noto comando che visualizza una finestra con la stessa intestazione ("Titolo della finestra") ma ciascuna con un differente messaggio. In presenza di questa organizzazione, costituita dal programma chiamante `Main` e due programmi, ciascuno a sé stante, valgono le seguenti regole:

- in caso di richiesta di esecuzione di una procedura (che si ottiene con un clic sull'opportuno tasto della barra degli strumenti) verrà eseguita solo quella in cui si trova il cursore lampeggiante;

- nel caso in cui venga eseguita una procedura che contiene una chiamata ad un'altra verrà eseguita anche quest'ultima laddove richiesto dal chiamante.

Per verificare quanto sopra possiamo eseguire una delle tre procedure spostando opportunamente il cursore lampeggiante. In alternativa possiamo sostituire il nome della procedura chiamata da `Main` scrivendo `maschio` al posto di `femmina` etc.

ESERCIZI

1. A partire da un foglio di lavoro vuoto, scrivere nel modo ritenuto più idoneo una procedura con le seguenti istruzioni:

```
Sub primo()
        MsgBox "a", , "b"
End Sub()
```

   avendo l'accortezza di sostituire al posto di *a* un messaggio che riporta il giorno di nascita ed al posto di *b* un titolo per questa finestra.

2. Scrivere un programma che visualizza consecutivamente due finestre. La prima con il nome del vostro animale preferito, la seconda con la vostra data di nascita. Le finestra dovranno avere due titoli collegati all'informazione cui si riferiscono.

2 DATI ED ESPRESSIONI

Ci occuperemo ora delle tipologie di dati che può trattare un programma VBA nonché dello spazio che ciascuno di questi dati occupa in memoria. Allo scopo di chiarire meglio di cosa stiamo parlando, faremo ora alcune considerazioni preliminari che ci permetteranno esporre l'argomento oggetto di questo capitolo.

2.1 TIPOLOGIE DI DATI

Consideriamo la sequenza di caratteri "12042000". Apparentemente sembrerebbe trattarsi di un numero, ma a ben guardare potrebbe trattarsi anche di una data e precisamente del 12 aprile dell'anno 2000 o ancora del numero di telefono 12 04 20 00. Come facciamo, allora a determinare di cosa si tratta? La risposta è semplice (o difficile fate voi) e dipende dall'impiego che dovremo fare del dato in questione.

Se, per esempio, si tratta dell'importo di una fattura, probabilmente lo utilizzeremo per eseguire dei calcoli, conseguentemente si tratta di un numero (sarebbe meglio dire dato numerico).

Se invece è una data, la utilizzeremo in modo particolare ed infatti sommando a questa un valore, supponiamo 30, vorremmo ottenere 12052000 cioè il 12 maggio dello stesso anno, non 12042030 come se si trattasse di un numero.

La situazione si complica nel caso di un numero telefonico perchè esso, pur essendo costituito da caratteri numerici, non sarà mai utilizzato a fini di calcolo!. Ecco allora la necessità di dover sapere sempre la natura o l'impiego di un dato in un programma[1].

2.2 COSTANTI E VARIABILI

I dati all'interno di un programma si classificano come variabili o costanti. Un dato di tipo variabile è riconoscibile perché identificato attraverso un nome, un dato di tipo costante non è identificato attraverso un nome ma con il valore corrispondente.

Nella tabella 2.1, sono riportati alcuni esempi di costanti e di variabili. Si noti la differenza tra Uffa nome di variabile e Uffa costante. La seconda delle due è riconoscibile come tale, per essere racchiusa tra i doppi apici.

Riguardo al nome di una variabile possiamo dire che questo può essere costituito da una sequenza qualsiasi di caratteri (fino ad un

1 Un numero di telefono, pur essendo costituito da caratteri numerici, va considerato alla stregua di un nome.

massimo di 255). L'iniziale del nome deve essere necessariamente una lettera dell'alfabeto (maiuscola o minuscola). Si consiglia, ma non è obbligatorio, di scegliere un nome di variabile che ricordi, mnemonicamente, la funzione che svolge all'interno del programma.

Tabella 2.1: costanti e variabili

Costanti	**Variabili**
123, "123" , "Uffa!123"	Uffa, X123, Abc

2.3 LA DICHIARAZIONE DELLE VARIABILI

Abbimo visto in apertura quanto sia importante conoscere l'impiego cui è destinato una dato. Questo principio trova in VBA una immediata soluzione costituita dalle istruzioni di dichiarazione delle variabili[2]. L'istruzione di dichiarazione si mette di solito all'inizio della procedura e si utilizza per associare a ciascuna variabile il tipo di dato che questa dovrà contenere.

La cartella di lavoro `dichiarazione_delle_variabili.xls` riporta parecchi esempi di istruzioni di dichiarazione di dati. Nel codice 2.1 sono visibili le prime righe della procedura della cartella di lavoro.

Codice 2.1: Le dichiarazioni delle variabili

```
Dim CiccioRiccio, Caterina, _
      Tio    ' dichiarazioni senza tipo (viene assunto il tipo Variant)
             ' si esemplifica l'uso del carattere di continuazione
Dim Nome As String * 10 ' variabile stringa a lunghezza fissa
       . . .
```

Con la prima istruzione `Dim` si dichiarano tre variabili `CiccioRiccio, Caterina, Tio`. Il carattere finale della prima riga "_" è un carattere di continuazione dell'istruzione alla riga successiva.

Il carattere "'" (apice singolo) che si trova dopo `Tio`, rappresenta un commento. Tutto ciò che segue questo carattere non ha influenza sull'esecuzione del programma.

Dopo il nome o i nomi viene la descrizione del tipo di dato che deve essere preceduta dalla parola `As`. I tipi di variabile, lo spazio di memoria che occupano in bit[3], i numeri che si possono rappresentare con quel tipo di dato (se si tratta di variabile numerica), si trovano

2 La dichiarazione delle variabili non è obbligatoria ma la sua mancanza comporta una riduzione nella velocità di esecuzione. Noi la effettueremo in tutti i programmi del libro.

3 Un bit è un dispositivo fisico in grado di memorizzare una informazione binaria come per esempio 0 oppure 1. Si ricorda che 1 byte = 8 bit.

	A	B	C	D
1				
2	TIPO	LUNGHEZZA IN BYTE	ESTREMI/VALORI AMMESSI	
3				
4	Array	a seconda di quanto dichiarato		
5	Byte	8 bit	0	255
6	Boolean	16 bit	True	False
7	Currency	64 bit	(~)-922.000.000.000.000,5808	(~)922.000.000.000.000,5808
8				
9	Date	64 bit	1 gennaio 100(00:00:00)	31 12 9999(23:59:59)
10	Double	64 bit	-1.(14cifre)E308	4.(14cifre)E-324
11			-4(14cifre)E-324	1.(14cifre)E308

Figura 2.1: Tipo, bytes e valori corrispondenti

nelle due colonne successive del foglio Excel e sono parzialmente riprodotti nella figura 2.1. In base a quanto vediamo, possiamo dire per esempio che se una variabile è dichiarata di tipo *Byte*, non potrà essere utilizzata in calcoli che eccedano l'intervallo $0, 255$.

2.4 ISTRUZIONI DI ASSEGNAZIONE

Nella parte di codice che segue le dichiarazione delle variabili sono riportate, a titolo d'esempio, diverse istruzioni di assegnazione.

La forma generale di una istruzione di questo tipo è:

nome_di_variabile = espressione_qualsiasi

in cui *espressione_qualsiasi* è qualunque combinazione di nomi di variabili con simboli aritmetici, mentre *nome_di_variabile* è il nome di una variabile.

L'istruzione di assegnazione si utilizza per attribuire il risultato di un calcolo (a destra del segno di uguaglianza), ad una variabile (a sinistra del segno di uguaglianza). Non deve essere inteso come una espressione matematica! Prima di approfondire (si veda la sezione appositamente dedicata di seguito), commentiamo gli esempi proposti nel codice sottostante.

dichiarazione_delle_variabili.xls

```
        . . .
Dim Nome As String*10
                . . .
        DataNascita = #9/26/1953#
' calcoliamo i giorni passati dalla data di nascita ad oggi
' Now calcola la data odierna
        GiorniVissuti  = Now - DataNascita
' alcuni esempi di istruzioni di assegnazione e di uso di operatori:
'   assegnazione di una costante alfabetica ad una variabile
        Nome = "Antonio"
' - effettua il calcolo di una espressione ed assegna il risultato ad
    una variabile
        Gio = 1 + 1.51
' - assegna una variabile ad una cella (scrive nella cella A21)
        Cells(21, 1).Value = Gio
' - legge il contenuto della cella C21,
```

```
'     somma a questro la costante numerica 3,
'     scrive il risultato nella cella A23)
        Cells(23, 1).Value = Cells(21, 3).Value + 3
        . . .
```

La prima istruzione dopo `Dim` è un esempio di assegnazione di una data in formato inglese (mm/gg/aaaa) ad una variabile. A destra dell'uguale abbiamo una costante data riconoscibile perché racchiusa tra doppi cancelletti.

La seconda istruzione calcola la differenza, espressa in giorni, tra la data odierna (rappresentata da `Now`) ed il contenuto della variabile `DataNascita` (attualmente questa variabile contiene il dato 26 settembre 1953).

Il significato delle istruzioni successive è facilmente comprensibile dai commenti inseriti nel codice.

2.4.1 *L'incremento di una variabile*

Una conseguenza notevole della logica sottesa all'istruzione di assegnazione riguarda il concetto di incremento di una variabile. Con incremento di una variabile si intende la somma o la sottrazione di una certa quantità ad una variabile, riassegnando poi così ottenuto alla stessa variabile. Per poter fare questo dovranno essere definite almeno due funzionalità:

- una istruzione iniziale del valore 0 alla variabile (di 1 nel caso della produttoria);
- l'istruzione di assegnazione per l'incremento/decremento.

Un esempio di questo tipo è visibile nel codice seguente che si trova scritto nella parte finale della procedura:

```
. . .
VariabileModulo = 0
VariabileModulo = VariabileModulo + 2
. . .
```

Il significato della seconda riga di codice, riprendendo quanto detto all'inizio del paragrafo, è da intendersi:
il contenuto di `VariabileModulo`, pari a 0, deve essere sommato a 2 ed il risultato dell'operazione deve essere assegnato a `VariabileModulo`.

2.4.2 *Le espressioni logiche*

Oltre ai valori visti nei paragrafi precedenti, esiste un altro tipo di dato, e quindi una variabile, chiamato *logico* o *booleano*. Esso può va-

lere "Vero" oppure "Falso". Utilizzando opportuni operatori, chiamati operatori logici, è possibile costruire espressioni logiche il cui risultato può essere assegnato a variabili dello stesso tipo.

Codice 2.2: Una espressione logica

```
. . .
Dim Verro As Boolean
. . .
Verro = 4 > 5
. . .
```

Nel listato vediamo un esempio di espressione logica il cui risultato viene assegnato ad una variabile. Si noti l'uso dell'operatore logico ">" che definisce il tipo di espressione. Il dato contenuto in Verro è Falso in quanto 4 è minore di 5. Come già accennato nel paragrafo 2.4 abbiamo proceduto alla dichiarazione della variabile.

ESERCIZI

1. Nel codice VBA di questo capitolo sostituire nella istruzione che contiene la data di nascita la vostra data di nascita e visualizzare in una MsgBox il risultato ottenuto.

2. Scrivere una procedura che visualizza in una MsgBox il vostro cognome e nome separati dal carattere spazio. Si ipotizza che le celle A1 e B1 contengano rispettivamente il vostro nome e cognome. Nel codice, a parte le dichiarazioni di inizio/fine procedura, devono essere utilizzati solamente:
   - comando MsgBox;
   - l'operatore di concatenazione di stringhe &;
   - Cells(1,1).Value e Cells(1,2).Value;
   - la costante spazio .

3. Come nell'esercizio 2 ma assegnando il contenuto delle celle A1 e B1 ad altrettante variabili dichiarate in modo opportuno. Di conseguenza la MsgBox dovrà contenere i nomi delle variabili e non il riferimento alle due celle del foglio Cells(1,1).Value, Cells(1,2).Value.

4. Tre celle contengono tre numeri interi. Calcolare la somma e scrivere il risultato in un'altra cella.

5. Ricordando che:

   $5(dividendo) : 3(divisore) = 1(quoto \quad o \quad quoziente) \qquad resto2$

e che:

$$\text{quoto} \quad \text{x} \quad \text{quoziente} + \text{resto} = \text{dividendo}$$

calcolare il resto e il quoziente della divisione tra la somma calcolata nell'esercizio precedente con il numero 3 e scrivere questi due risultati in due celle scelte a piacere.

6. Calcolare il risulato della moltiplicazione tra la variabile ciccio e 5 assegnando nuovamente il risultato del calcolo alla variabile ciccio. Il valore di ciccio prima della moltiplicazione deve essere opportunamente inizializzato.

3 LE FUNZIONI

Qualunque linguaggio di programmazione è in grado di eseguire una delle quattro operazioni aritmetiche. Ognuno poi, in relazione all'ambiente cui è orientato, dispone di un insieme aggiuntivo di operatori, simili ai quattro operatori aritmetici e di uso frequente, che hanno lo scopo di facilitare l'utente nella soluzione di problemi. Questo insieme di operatori prende il nome di *funzioni*.

Il lettore pratico del programma Excel ne dovrebbe conoscere qualcuna, per averla utilizzata nella scrittura di formule in celle del foglio. Vedremo adesso come si possono calcolare le funzioni in VBA.

3.1 LE FUNZIONI EXCEL IN VBA

Nel programma VBA è ammesso il riferimento alle funzioni di Excel ma bisogna tenere presente alcune considerazioni.

La prima è relativa al fatto che le funzioni VBA e le funzioni di Excel, facendo parte di due applicazioni diverse, vanno riferite in modo diverso. Per questo motivo in VBA il riferimento ad una funzione Excel deve premettere il termine Application, ovvero il nome del programma Excel, al nome della funzione separando le due quantità con il carattere ". ".

Per esempio se vogliamo calcolare il massimo di quattro valori dovremo scrivere: Application.Max(3, 5, 123, 18).

Si nota che il nome della funzione Excel deve utilizzare il suo corrispondente in lingua inglese[1] [2].

Una seconda avvertenza riguarda la specificazione di zone di celle. VBA non segue la sintassi di Excel nel riferimento a zone di celle. Perciò, per esempio in relazione alla funzione di somma, non sarà possibile scrivere `Application.Sum(A1:G1)`! Il problema, che richiede conoscenze supplementari, verrà trattato e risolto nel paragrafo 7.7.

3.2 LE FUNZIONI VBA

Il programma VBA è dotato di un insieme di funzioni proprie. Nel foglio di lavoro `funzioni.xls` abbiamo elencate le principali, raggruppandole in base al tipo di risultato che producono. Nella colonna A si trova il nome della funzione e gli argomenti richiesti (N=numero,

1 Per un elenco con i nomi delle funzioni inglesi, relativi a quelli italiani, si consulti l'indirizzo http://support.microsoft.com/kb/638465/it, oppure si inserisca in un motore di ricerca il termine *Cross Reference Italiano-Inglese funzioni del foglio*.

2 Nella fattispecie Max ha lo stesso nome sia in inglese che in italiano di conseguenza non si trova nell'elenco suddetto.

S=stringa, D=data, E=espressione qualsiasi), nella colonna B si trova una istruzione di esempio, nella colonna C (quando ritenuto significativo), il risultato ottenuto. Le istruzioni VBA, utilizzate per il calcolo delle funzioni, sono contenute nella procedura Funzioni visibile nella finestra del codice (Alt+F11). I risultati sono riportati nella colonna C.

MATEMATICHE		
Abs(N)	Abs(-20)	20
Cos(N)	Cos(0)	1
Exp(N)	Exp(1)	2.718281828
Int(N)	Int(1234.56)	1234
Log(N)	Log(1)	0
Rnd(N)	Rnd	0.723522365
Round(N, P)	Round(1234.56),0	1235
Sgn(N)		-1
Sin(N)		0.893996664
Sqr(N)		1.414213562

CONVERSIONE DATI		
Asc(S)	Il codice ASCII di S	65
Chr(N)	il carattere ASCII di N	A
Str(N)	Str(200) & "ABC"	200ABC
Val(S)	Val("123")	123
CDate(E)	Cdate(365)	30-dic-00

Figura 3.1: Le principali funzioni VBA

Di seguito riepiloghiamo i principali punti da tenere presenti riguardo ad una funzione:

- calcola un solo valore;
- si identifica con un nome (solitamente) seguito da uno o più argomenti racchiusi tra parentesi tonde;
- può trovarsi in una espressione purché abbia un risultato compatibile con il dato di quell'espressione;
- i suoi argomenti possono essere sostituiti con altre espressioni purché compatibili.

Se per esempio, una funzione richiede come argomento un numero intero, al posto di questo potranno essere specificati: un numero intero, un'espressione aritmetica il cui risultato è un numero intero, l'indirizzo di una cella contenente un numero intero o una espressione numerica intera[3].

3.3 LA GUIDA DELLE FUNZIONI

Nel caso volessimo ottenere un aiuto maggiore, si può ricorrere ad una procedura che visualizza una descrizione di una determinata funzione (nonché di altri componenti del linguaggio) con alcuni esempi relativi al suo uso. I passaggi per l'attivazione di questa guida sono costituiti da:

1. selezione del comando Visualizza → Visualizzatore oggetti nel menu dei comandi;

3 La sostituibilità dei dati è molto utile nella programmazione perché rappresenta il modo di rendere "generalizzabile " un programma. Vedremo in seguito numerosi esempi di applicazione di tale principio.

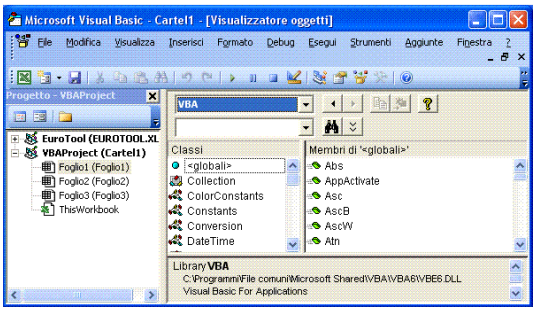


Figura 3.2: Il visualizzatore di oggetti

2. selezione del termine VBA nel primo menu a tendina in alto a sinistra visualizzato nella nella sottofinestra del codice;

3. (facoltativo) nella finestra di scorrimento intitolata Classi, subito sottostante quella del menu a tendina VBA, selezione della famiglia cui appartiene la funzione che cerchiamo (Conversion, DateTime, FileSystem, Financial etc.);

4. nel menu di destra compare l'elenco di tutte le funzioni di quella famiglia;

5. selezione della funzione;

6. premere il tasto "?" che si trova sopra l'elenco delle funzioni che stiamo consultando. A questo punto viene visualizzata la guida relativa alla funzione che ci interessa[4].

Una prima alternativa molto più rapida consiste nell'utilizzo del tasto F2 quando ci troviamo in ambiente VBE. In questo modo otterremo immediatamente la comparsa della finestra Visualizzatore oggetti.

Una seconda opzione consiste nel fare clic con il tasto sinistro del mouse in corrispondenza di uno qualunque dei caratteri del nome della funzione scritta nel codice VBA. A questo punto si preme il tasto F1 che aprirà la finestra della guida di quella funzione.

3.4 LA MODIFICA DELL'ORDINE DEGLI ARGOMENTI

Quando si vuole modificare l'ordine degli argomenti di una funzione si deve ricorrere ad una sintassi alternativa nella quale sia presente l'associazione tra nome e valore. La procedura finale presente nella cartella riferita in apertura del paragrafo 3.2 costituisce un esempio applicato alla funzione MsgBox. Le sue istruzioni sono visibili nel codice 3.1.

Ricordando che il primo ed il terzo argomento della funzione sono chiamati rispettivamente Prompt e Title, vediamo come come la

4 Deve essere installata la *Guida in linea di Microsoft Visual Basic*, diversamente la finestra apparirà vuota. Questo componente può essere installato in un momento successivo.

sintassi alternativa, presente nella seconda MsgBox, non abbia necessità di specificare il secondo argomento. Questo si spiega con il fatto che l'associazione argomento-valore viene fatta solamente per gli argomenti presenti. Si noti anche il carattere di separazione ":=" tra il nome di un argomento ed il suo valore[5].

Codice 3.1: La modifica degli argomenti della funzione

```
Sub messageBox()
    MsgBox "Suggerimento in posizione a  priori ", , "Titolo"
' sintassi alternativa che specifica i parametri della funzione
' secondo un ordine diverso da quello stabilito a priori
    MsgBox Title:="Titolo", Prompt:="Suggerimento specificato in  posizione
        diversa "
End Sub
```

ESERCIZI

1. Nelle celle da A1 ad H1 si trovano scritti una serie di numeri compresi tra 20 e 22. Calcolare la somma di questi numeri e scrivere il risultato nella cella A2 assieme al messaggio: "La somma dei numeri è". Se, per esempio la somma è 160, in A2 deve essere scritto: "La somma dei numeri è 160".

2. Come sopra ma la serie è composta da numeri compresi tra 1000 e 10000.

3. In A1 è scritto il tuo codice fiscale. Riporta in A2 il codice del mese in cui sei nato (il mese è codificato nel nono carattere).

4. La cella A1 contiene la costante stringa "12,2" (ovviamente nella cella non ci sono i doppi apici). Si deve scrivere in B1 il numero 12 e in C1 il numero 2.

5. La cella A1 contiene una costante stringa simile alla precedente in cui le parti che precedono e seguono la virgola hanno un numero di cifre non conosciuto a priori (mai però maggiore di 4). Si deve scrivere in B1 il numero che precede la virgola e in C1 il numero che segue la virgola.

6. Utilizzando lo strumento ritenuto più opportuno, chiarire la differenza tra le funzioni `Str` e `CStr`.

7. La cella A1 contiene il tuo indirizzo di posta elettronica. Scrivere la procedura che calcola in B1 ed in C1 rispettivamente il

5 In una procedura, terminata la scrittura di una funzione, dopo aver inserito la parentesi aperta, compare una finestra con il nome di quella funzione seguita dall'elenco degli argomenti.

prefisso ed il suffisso (stringa che precede e stringa che segue) il carattere "@ " in altrettante celle a tuo piacimento. Al termine visualizza con un comando `MsgBox` il risultato separando le due parti con un "a capo".

8. In A1 è scritto il tuo codice fiscale. Sapendo che il carattere 11 e 12 contengono il giorno della data di nascita, che nel caso di femmine alla data di nascita viene aggiunta la costante 40, si chiede il programma che scrive in una cella scelta a piacere il sesso corrispondente al codice fiscale.

9. La cella A1 contiene un codice fiscale. Spacchettarlo nei suoi componenti fondamentali: cognome, nome, anno di nascita, codice del mese di nascita, giorno di nascita, codice Istat del comune, carattere di controllo (ovvero inserire queste parti in altrettante variabili applicando una opportuna funzione. Fatto questo riscrivere il codice fiscale intero, quello ottenuto dallo spacchettamento, nella cella B1.

4 LE ISTRUZIONI CONDIZIONALI

Negli esempi visti finora, le istruzioni del programma venivano eseguite comunque. Questa modalità non è adatta a risolvere le situazioni in cui è richiesta una diversificazione nei comportamenti del programma in relazione a determinati eventi.

Quando questo accade è possibile utilizzare un insieme di istruzioni, chiamate "condizionali", che definiscono percorsi alternativi in relazione al codice da eseguire. Vediamo di cosa si tratta.

4.1 IF MONOISTRUZIONE

La prima e più semplice istruzione condizionale ha la seguente sintassi:

```
If espressione_condizionale Then istruzioni
```

in cui *espressione_condizionale* è una qualunque espressione logica, come quella del codice 2.2, e *istruzioni* rappresenta una o più istruzioni che devono trovarsi nella stessa riga dell'`If`. Qualora le istruzioni siano più di una, deve essere utilizzato il carattere ":" a separare l'una dall'altra[1].

A titolo di esempio, cui ci riferiremo in tutto il capitolo, supponiamo di voler calcolare le soluzioni di una equazione di secondo grado, dati i tre coefficienti A, B, C[2].

Nella cartella di lavoro `ifthenelse.xls` la procedura `IfThenElse1` contiene il primo esempio di `If` monoistruzione. Con questo tipo di istruzione condizionale la capacità elaborativa è molto bassa in relazione al problema che dobbiamo risolvere. L'unica possibilità che abbiamo è quella di scrivere una stringa relativa al tipo di soluzioni (reali/immaginarie). Nel codice 4.1 ne vediamo un esempio: il risultato del calcolo di Δ, viene riportato in A1.

Codice 4.1: If monoistruzione

```
Sub IfThenElse1()
    Dim A As Single, B As Single, C As Single, Delta As Single
```

1 Se ne sconsiglia l'uso per motivi di leggibilità. Si veda nel paragrafo successivo come risolvere questo problema.

2 L'assegnazione dei tre valori numerici alle corrispondenti variabili è stata fatta con istruzioni di assegnazione. Questa scelta comporta una perdita di generalità nella soluzione ma si giustifica per motivi di semplicità.

```
    A = 2
    B = 5
    C = 2
    Delta = (B ^ 2) - 4 * A * C
    Cells(1, 1).Value = "Due soluzioni reali ( distinte /non)"
    If Delta < 0 Then Cells(1, 1).Value = "Due soluzioni immaginarie"
End Sub
```

Cambiando i dati assegnati alle tre variabili A, B, C, sarà possibile effettuare altri calcoli però le potenzialità rimarranno comunque le stesse[3].

Fortunatamente l'istruzione `If` dispone di altre e più complesse forme di condizioni. Vediamo quali sono.

4.2 IF SEMPLICE

La soluzione migliore quando si debbano eseguire più istruzioni in subordine ad una condizione, è quella dell'`If` semplice. La sua sintassi è:

```
If espressione_condizionale Then
        . . .
    istruzioni
        . . .
End If
```

Come visto in precedenza questa soluzione non si presta ad aumentare le nostre capacità risolutive in relazione al problema da risolvere.

4.3 IF .. THEN .. ELSE

Quando si debbano eseguire insiemi di istruzioni mutuamente esclusivi, si impiega questo tipo di istruzione. La sintassi è:

```
If espressione_condizionale Then
        . . .
    istruzioni (blocco 1)
        . . .
Else
        . . .
    istruzioni (blocco 2)
        . . .
End If
```

3 Ricorrendo a qualche forzatura, si potrebbe aggirare l'ostacolo, ma si tratta di artifici che dati i nostri obbiettivi non è il caso di approfondire. Si veda il prossimo paragrafo.

In questo caso l'insieme di istruzioni che si trovano prima di `Else` viene eseguito se la condizione è vera mentre l'insieme compreso tra `Else` e `End If` viene eseguito se la condizione è falsa. Nel codice riprodotto di seguito è visibile un esempio di questo tipo.

```
Sub IfThenElse2()
    Dim A As Single, B As Single, C As Single, Delta As Single
    A = 2
    B = 5
    C = 2
    Delta = (B ^ 2) - 4 * A * C
    If Delta < 0 Then
            Cells(1, 1).Value = "Due soluzioni immaginarie"
        Else
            Cells(1, 1).Value = "Due soluzioni reali (distinte/non)"
        End If
```

La soluzione, dal punto di vista della leggibilità, è senz'altro migliore rispetto al caso precedente ma la capacità del programma, in relazione al problema da risolvere, rimane ancora scarsa.

4.4 IF NIDIFICATI

Nulla vieta che all'interno dei blocchi di istruzioni visti nel caso precedente, si trovino ulteriori istruzioni condizionali. Si parla in questo caso di If "nidificati". Questa soluzione permette di aumentare notevolmente le nostre capacità di calcolo ma la leggibilità risulta abbastanza penalizzata nonostante la differente tabulazione. Il codice che mostriamo si riferisce alla procedura `IfThenElse3`.

```
Sub IfThenElse3()
    . . .
    Delta = (B ^ 2) - 4 * A * C
    If Delta < 0 Then
            Cells(1, 1).Value = "Due soluzioni immaginarie"
        Else
            If Delta = 0 Then
                    X1 = -B / (2 * A)
                    Cells(1, 1).Value = "X1=X2= " & Str(X1)
                Else
                    X1 = (-B - Sqr(Delta)) / (2 * A)
                    X2 = (-B + Sqr(Delta)) / (2 * A)
                    Cells(1, 1).Value = "X1= " & Str(X1)
                    Cells(2, 1).Value = "X2= " & Str(X2)
                End If
        End If
End Sub
```

4.5 SELECT CASE

Questa soluzione permette di evitare i problemi accennati alla fine del paragrafo precedente. Essa è consigliabile quando le alternative su cui scegliere sono maggiori di due e i blocchi sono costituiti da svariate istruzioni. La sua sintassi è costituita da:

```
Select Case   espressione
    Case   elenco   condizioni1
                . . .
                . . .
    Case   elenco   condizioni2
                . . .
                . . .
    [Case Else
                . . .
                . . .]
    End Select
```

in cui:

- *espressione* è una qualunque espressione numerica o stringa;
- *elenco condizioni1*, *elenco condizioni2*, possono essere espressioni oppure espressioni condizionali;
- *istruzioni* sono una o più istruzioni.

Il comando calcola il risultato di espressione e lo confronta con gli elenchi di condizioni di ciascun `Case`. Quando una di queste condizioni è soddisfatta, vengono eseguite le istruzioni corrispondenti al `Case`. Il controllo del programma, fatto questo, passa alla prima istruzione seguente `End Select`. Viceversa, se nessuna delle condizioni dei vari elenchi di condizioni viene soddisfatta ed è presente `Case Else`, vengono eseguite le istruzioni che si trovano al suo interno[4].

Allo scopo di dimostrare il funzionamento del comando `Case` si consideri la procedura `IfThenElse4` ottenuto dalla sostituzione in `IfThenElse3` di `If` con `Case` (lasciamo al lettore un giudizio sulla leggibilità tra i due).

```
Sub IfThenElse4()
    Dim A As Single, B As Single, C As Single, Delta As Single, X1 As Single, X2
        As Single
    A = 2
    B = 5
    C = 2
    Delta = (B ^ 2) - 4 * A * C
    Select Case Delta
        Case Is < 0          ' Delta < 0
```

4 Le parentesi quadre evidenziano che si tratta di una parte che non è obbligatoria.

```
            Cells(1, 1).Value = "Due soluzioni immaginarie"
        Case Is = 0          ' Delta = 0
            X1 = -B / (2 * A)
            Cells(1, 1).Value = "X1=X2= " & Str(X1)
        Case Is > 0          ' si puo' scrivere anche Case Else
            X1 = (-B - Sqr(Delta)) / (2 * A)
            X2 = (-B + Sqr(Delta)) / (2 * A)
            Cells(1, 1).Value = "X1= " & Str(X1)
            Cells(2, 1).Value = "X2= " & Str(X2)
        End Select
End Sub
```

Qualora *elenco condizioni1, elenco condizioni2, etc.*, contengano più di una condizione, bisogna utilizzare la virgola come separatore. Inoltre se si devono imporre condizioni di maggiore, minore, uguale, si deve usare la parola chiave `Is` seguita dall'operatore di condizione e poi da un'altra espressione. Il codice seguente, adattato al nostro problema, ne mostra un esempio[5].

```
Sub case_regredito()
    Dim A As Single, B As Single, C As Single, Delta As Single
    A = 2
    B = 5
    C = 2
    Delta = (B ^ 2) - 4 * A * C
' qui scrivere espressione numerica o stringa
    Select Case Delta
' specificare Is nel caso si faccia uso di operatori di confronto (<, =,
    >, etc.)
        Case Is > 0, Is = 0
            Cells(1, 1).Value = "Due soluzioni reali ( distinte /non)"
        Case Else
            Cells(1, 1).Value = "Due soluzioni immaginarie"
        End Select
End Sub
```

La condizione, qualora debba riferirsi ad intervalli numerici, si può esprimere nella forma *espressione1 To espressione2*. Nel codice seguente ne vediamo un esempio assieme a forme condizionali viste in precedenza.

```
Sub case_senza_is()
        Dim I As Integer
        I = 8
        Select Case I
' nel caso si tratti di un intervallo finito
            Case 1 To 3
                MsgBox "I compreso tra 1 e 3"
' nel caso si tratti di una lista di variabili/espressioni
            Case 3, 5, 7, 9
```

5 Il nome attribuito a questa procedura non è casuale.

```
            MsgBox "I uguale a 3/5/7/9"
        Case 4, 6, 8
            MsgBox "I uguale a 4/6/8"
        Case Is < 1, Is > 9
            MsgBox "I > 9 oppure I < 1"
        End Select
End Sub
```

ESERCIZI

1. Riprodurre il codice 4.1 sostituendo alla scrittura nella cella A1 un opportuno messaggio.

2. La cella A1 contiene un codice fiscale (CF). Riportare in A2 il giorno di nascita relatvo a quel CF sotto forma di numero (il giorno di nascita è scritto nei caratteri 10-11 di CF). Si ricorda che in CF la codifica del giorno di nascita è maggiore di 31 quando si tratta di una femmina. In tal caso il giorno di nascita effettivo si ottiene sottraendo 40 al numero corrispondente in CF. Se la cella A1 contenesse `GRNNTN53P26H501S`, dopo aver eseguito il programma, la cella A2 dovrà contenere il numero 26 (il codice è di un maschio). Se A1 contenesse `GRNNTN53P46H501S`, dopo aver eseguito il programma, A2 dovrà contenere comunque il dato 26. In questo caso il sesso del soggetto intestatario del CF è femminile.

3. Riscrivere il codice di `ifthenelse1()` tenendo presente che i tre coefficienti sono scritti nelle celle A1, B1, C1.

4. Riscrivere il codice di `ifthenelse1()` modificando la condizione `Delta < 0` in `Delta >= 0`.

5. Riscrivere il codice di `ifthenelse3()` modificando la condizione `Delta < 0` in `Delta > 0`.

5 I CICLI

Il ciclo o loop costituisce uno dei costrutti fondamentali nella programmazione in quanto permette di gestire con facilità la ripetizione di una o più istruzioni. Gli strumenti disponibili alla gestione della ripetizione sono molteplici e dipendono dalla condizione che ne presiede il funzionamento. Nei paragrafi che seguono passeremo in rassegna i diversi modi di eseguire un loop.

5.1 IL CICLO FOR NEXT

La forma più semplice di ciclo VBA ha la seguente sintassi:

```
For nome_di_variabile = espressione1 To espressione2
        . . .
        istruzioni
        . . .
        Next nome_di_variabile
```

in cui *nome_di_variabile* è il nome di una variabile numerica qualsiasi chiamata variabile di controllo del loop. In genere è una variabile intera (ma questo non è obbligatorio), *espressione1*, *espressione2* sono espressioni numeriche qualsiasi come visto nel paragrafo 2.4.

Quando viene eseguita per la prima volta l'istruzione `For`, vengono svolti i seguenti automatismi:

1. il programma assegna alla variabile di controllo il valore di *espressione1* e lo confronta con quello di *espressione2*. Se il valore assegnato alla variabile di controllo è minore o uguale a quello di *espressione2* vengono eseguite le istruzioni comprese tra `For` e `Next`, altrimenti si passa ad eseguire la prima istruzione successiva a `Next`;

2. il valore della variabile di controllo viene incrementato di uno[1] e si procede a quanto descritto nel punto 1.

Nella cartella di lavoro `cicli.xls` si trova il codice VBA che discutiamo di seguito. Sotto il commento intitolato `CICLO1` è riportato il primo e più semplice esempio di loop. Questo, a parte le istruzioni `For/Next`, si compone di una sola istruzione costituita dalla visualizzazione di un messaggio a video. La variabile di controllo è `M`. I valori di inizio e fine loop sono rispettivamente 1 e 3. Come si dimostra

1 Vedremo in seguito una deroga a questo comportamento.

con la prima istruzione successiva al ciclo, terminata la ripetizione, la variabile di controllo vale 4.

Un semplice loop

```
. . .
For M = 1 To 3
        MsgBox "Il mattino ha l'oro in bocca", , "Titolo della Finestra"
        Next M
MsgBox "La variabile di controllo contiene" & Str(M)
. . .
```

Nell'esempio mostrato, eseguita per la prima volta l'istruzione `For`, la variabile di controllo `M` ha valore 1; si confronta questa con 3. Essendo 1<3, vengono eseguite le istruzioni interne al ciclo.

Fatto questo si incrementa di uno il valore della variabile di controllo (che diventa pari a 2) e si confronta nuovamente con quello di fine ciclo (3). Essendo il primo ancora inferiore al secondo, si eseguono le istruzioni... e così via per tre volte fino a quando la variabile di controllo non diventa pari a 4.

La variabile di controllo a questo punto, contiene un valore superiore a quello stabilito, il ciclo termina ed il programma passa ad eseguire la prima istruzione successiva.

5.1.1 *La sommatoria*

Sotto il commento `CICL02a` è riportato un esempio in cui i valori assunti dalla variabile di controllo nel corso del loop (1, 2, 3, 4, ..., 10) vengono utilizzati all'interno delle istruzioni da ripetere per eseguire la somma dei primi 10 numeri[2].

Codice 5.1: La sommatoria

```
. . .
Totale = 0
For M = 1 To 10
        Totale = Totale + M
        Next M
Cells(1, 3).Value = "La somma dei primi 10 numeri e' " & Str(Totale)
. . .
```

Si nota, trattandosi della sommatoria di un insieme numeri, l'istruzione di azzeramento ad inizio loop `Totale = 0` e, all'interno del ci-

2 ATTENZIONE all'interno di un ciclo è vietato assegnare un valore alla variabile di controllo (p.e. scrivere un'istruzione come M = 15). Questo perchè la gestione di qesta variabile è appannaggio del ciclo e la sua modifica dall'esterno può causare malfunzionamenti o errori nel programma.

clo, l'istruzione che calcola la sommatoria `Totale = Totale + M`. Si ricordi quanto discusso nel paragrafo 2.4.

Sotto i commenti `CICLO2b` e `CICLO2c` sono riportati cicli equivalenti all'esempio precedente[3].

Cicli equivalenti

```
        Totale = 0
        For M = 11 To 2 Step -1
                Totale = Totale + M - 1
                Next M
        Cells(1, 3).Value = "La somma dei primi 10 numeri e' " & Str(Totale)

    '========================================== CICLO2c

        Totale = 0
        For M = 2 To 20 Step 2
                Totale = Totale + M \ 2
                Next M
        Cells(1, 3).Value = "La somma dei primi 10 numeri e' " & Str(Totale)
```

Si nota l'uso della parola `Step` necessaria a definire incrementi della variabile di controllo diversi da 1. Oltre questo si possono notare le modifiche necessarie alle istruzioni interne al ciclo per correggere l'effetto dovuto ai valori assunti da M.

Nel primo ciclo i valori assunti da M nel corso del ciclo sono 11, 10, ..., 3, 2. Nel secondo 2, 4, ..., 18, 20.

Le istruzioni sotto `CICLO3` si caratterizzano per l'uso della variabile di controllo del ciclo in veste di indirizzo di colonna della cella in cui scrivere. A questo proposito si rammenti quanto detto nel capitolo sulle funzioni riguardo la sostituibilità degli argomenti.

```
        . . .
        For M = 1 To 10
                Cells(30, M).Value = M
                Next M
        . . .
```

3 Può capitare di voler "simulare" il comportamento di un programma. Una tecnica che consigliamo di seguire è quella di seguire passo passo le istruzioni che vogliamo controllare. Dotati di matita e gomma, disegnamo le variabili con un quadratino ed scriviamo vicono a questo il nome della variabile corrispondente. Poi incominciamo a leggere le istruzioni annotando nei quadratini i valori che il programma genera all'esecuzione delle istruzioni eventualmente cancellando o sovrascrivendo i contenuti precedenti. Questa tecnica risulta particolarmente utile per capire bene il funzionamento dei cicli, magari utilizzando un minor numero di ripetizioni.

	A	B
1	sotto	
2	il	
3	bambu'	
4	al	
5	tempo	
6	che	
7	fu	
8	c'era	
9	un	
10	pirata	
11	che	
12	ormai	
13	non	
14	c'e'	
15	piu'	
16		
17		

Figura 5.1: Il calcolo delle celle piene

5.2 IL CICLO DO WHILE

Questo ciclo si usa quando la ripetizione deve essere eseguita in base a condizioni non numeriche.

Consideriamo il seguente problema: abbiamo una colonna di celle piene consecutive. La sequenza si interrompe con una cella vuota. Ipotizzando di conoscere l'inizio della sequenza, vogliamo sapere quante sono le celle piene. La soluzione è legata alla possibilità di sapere quando una cella è piena o vuota. In questo caso è sufficiente spostarsi sulla sequenza di celle fintantoché non ci troviamo in presenza di una cella vuota. A questo punto in ciclo si interrompe[4].

Vedremo più avanti la soluzione di questo problema. Per adesso consideriamo la forma generale di questo ciclo costituita da:

```
Do While | Until  espressione_condizionale
. . .
Loop
```

L'impiego di `Do While` oppure di `Do Until` (il carattere "`|`" indica che si tratta di due parole mutuamente esclusive), è basato sul risultato dell'espressione condizionale. Se il ciclo deve essere ripetuto finché l'espressione condizionale è vera si usa `Do While`. Se il ciclo va ripetuto finquando la condizione è falsa, si utilizza `Do Until`. Nel codice che mostriamo di seguito sono riportati due esempi che ottengono lo stesso risultato con le due diverse modalità espressive di ripetizione. Il codice fa riferimento al foglio della figura 5.1.

4 in uno dei prossimi capitoli vedremo come questo problema sia facilmente risolvibile in modo diverso. Per ora ipotizzeremo di non avere nessun'altra risorsa che questa.

```
        . . .
'                                                    CICLO4
    M = 1
' premesso che Cells(riga, colonna).Value = "" significa cella vuota
' possiamo dire che
' il ciclo viene eseguito fintantoche' la condizione e' Vera
    Do While Cells(M, 2).Value <> ""
        M = M + 1
        Loop
    Cells (2, 3).Value = "Le celle occupate nella colonna A sono " & CStr(M - 1)
'
    M = 1
' il ciclo viene eseguito fintantoche' la condizione e' Falsa
    Do Until Cells (M, 1).Value = ""
        M = M + 1
        Cells (1, 8).Value = M
        Loop
    Cells (3, 3).Value = "Le celle occupate nella colonna A sono " & CStr(M - 1)
        . . .
```

5.3 IL CICLO DO . . LOOP WHILE/UNTIL

Nei precedenti paragrafi abbiamo visto come la condizione che determina l'esecuzione del ciclo viene testata a monte delle istruzioni da ripetere. Può capitare invece che le istruzioni interne al ciclo debbano essere eseguite almeno una volta. In questo caso la condizione che ne presiede il funzionamento deve trovarsi alla fine e non all'inizio del ciclo e si usa `Do .. Loop While`.

Si vuole contare in modo ordinale la posizione di una lettera dell'alfabeto (la lettera "g" nel nostro esempio). Per risolvere il problema mettiamo in due colonne del foglio in corrispondenza biunivoca ciascun carattere dell'alfabeto con il corrispondente valore ordinale. Se

I	J	K	L
	a	il primo	
	b	il secondo	
	c	il terzo	
	d	il quarto	
	e	il quinto	
	f	il sesto	
	g	il settimo	
	h	l'ottavo	
	i	il nono	
	j	il decimo	
	k	l'undicesimo	
	l	il dodicesimo	
	m	il tredicesimo	
	n	il quattordicesimo	
	o	il quindicesimo	
	p	il sedicesimo	
	q	il diciassettesimo	

Figura 5.2: La ricerca di un dato

eseguiamo un confronto tra la lettera che cerchiamo e tutti i caratteri dell'alfabeto, potremo sfruttare il contenuto di una variabile (M nel nostro esempio) che tiene conto dei confronti effettuati. Quando si verifica l'uguaglianza tra la lettera che cerchiamo ed uno dei caratteri, terminiamo il ciclo. A questo punto M, numero dei confronti effettuati fino a quel momento, sarà il dato che ci permette di indirizzare correttamente la cella con l'ordinale corrispondente.

```
Sub cicli4 ()
   Dim Carattere As String
   Dim M As Byte
   Carattere = "g"
   M = 0
' in questo esempio la condizione di terminazione del ciclo viene
    controllata alla fine
   Do
       M = M + 1
       Loop While Carattere <> Cells(M, 10).Value
   Cells(7, 3).Value = "La lettera " & Carattere & " e' " & _
                Cells(M, 11).Value & " carattere dell'alfabeto"
End Sub
```

5.4 COME INTERROMPERE UN CICLO

L'interruzione di un ciclo può essere intenzionale o dovuta ad errori di logica. Negli esempi precedenti le condizioni di terminazione del ciclo erano stabilite direttamente nei comandi che lo eseguivano sia nel caso del `For` che nelle svariate forme `Do`.

Alcuni problemi richiedono però una soluzione diversa. Si consideri, a titolo d'esempio, di voler cercare una parola nella sequenza di celle di figura 5.1. Se, per semplicità, ipotizziamo di conoscere il numero di celle piene della colonna, sarà conveniente ricorrere ad un ciclo `For`. All'interno del ciclo inseriamo una istruzione che, qualora la ricerca abbia esito positivo, provveda all'uscita. Il codice corrispondente sarà:

```
       Dim M As Byte, parolaDaCercare As String
' si provi con una parola presente oppure mancante
       parolaDaCercare = "ciccio"
       'parolaDaCercare = "pirata"
       For M = 1 To 15
              If parolaDaCercare = Cells(M, 2).Value Then Exit For
              Next M
       If M>15 Then
                     MsgBox "Eureka"
              Else
                     MsgBox "Buuu"
              End If
End Sub
```

Si nota il comando `Exit For` che provoca l'uscita dal ciclo quando la stringa da ricercare viene trovata. L'istruzione condizionale dopo il ciclo permette di sapere se la ricerca ha avuto esito positivo o negativo a causa della terminazione anticipata.

Diverso è il caso di interruzione forzata dovuta ad un errore di logica non prevsto nel programma. Il seguente codice "simula" una situazione di questo tipo nella quale il ciclo si ripete fino ad esaurire le righe della colonna. In queste condizioni, non avendo di meglio, si ricorre alla combinazione di tasti Ctrl+Interr.

```
Sub cicloInfinito ()
    Dim I As Long
    I = 0
    Do
        I = I + 1
        MsgBox "Sono a riga " & Str(I)
        Loop While Cells(I, 2) = ""
End Sub
```

ESERCIZI

1. Nella zona di celle A1.A13, sono presenti un insieme di 13 numeri uno per ogni cella. Scrivere il programma che calcola la somma e la media di questi numeri e scrive questi due risultati in altrettante celle (la soluzione non dovrà impiegare alcuna funzione Excel/VBA).

2. Riempire alcune celle del foglio con stringhe di caratteri. Calcolare la lunghezza media $\overline{x}$ delle stringhe che riempiono le celle in base alla relazione:

$$\overline{x} = \frac{numero\_totale\_di\_caratteri}{numero\_di\_celle\_piene}$$

   Il risultato dovrà comparire sotto forma di MsgBox.

3. Premesso che la funzione VBA `IsEmpty` restituisce "True/Vero" se l'espressione al suo interno è vuota, scrivere i programmi equivalenti a quelli del paragrafo 5.2.

4. Risolvere il problema del paragrafo 5.2 ipotizzando che la riga da cui incomincia la sequenza di celle piene possa essere una qualunque della colonna A.

3

6 LE FUNZIONI DEFINITE DALL'UTENTE

Nel capitolo 3 avevamo illustrato, in maniera dettagliata, le funzioni del programma VBA. Qui parleremo di funzioni definite dall'utente. L'esempio cui ci riferiremo in questo capitolo è costituito dalla richiesta di inserimento di un dato all'utente e dall'eventuale controllo di quanto immesso[1].

6.1 LA FUNZIONE INPUTBOX

Nella cartella di lavoro del paragrafo 3.2 abbiamo elencato, senza discuterne, la funzione `InputBox`. Essa visualizza una finestra che richiede l'inserimento di un dato in un apposito spazio chamato *campo*. Ne vediamo un esempio nella figura 6.1.

La funzione ha cinque argomenti costituiti da: la richiesta relativa al dato da inserire, il titolo della finestra, la risposta predefinita che viene visualizzata nel campo di input, le coordinate x,y, espresse in twips[2] dell'angolo alto sinistro della finestra.

Le istruzioni che producono la finestra della figura sono contenute nella procedura che fa parte della cartella di lavoro inpututente.xls.

Codice 6.1: La funzione InputBox

```
Sub interazione_con_utente()
    Dim inputUtente As String
    Dim InputUtente1 As Integer
    inputUtente = InputBox("Io voto per ", "SCHEDA ELETTORALE", _
                           , 0, 0)
    MsgBox inputUtente
End Sub
```

Si nota la dichiarazione della variabile `inputUtente` che, coerentemente con il risultato calcolato dalla funzione, è di tipo String.

Una prima utile applicazione di questa funzione è contenuta nella procedura `InserisceInputQualsiasi`, parzialmente visibile nel codice 6.2. Essa trasferisce il dato immesso nel campo di InputBox nelle celle della colonna A, partendo dalla prima riga. L'azione è controllata da un ciclo che si ripete fintantoché si inserisce nel campo almeno un carattere. Diversamente la ripetizione viene conclusa.

1 Se la richiesta è relativa ad un dato numerico sarà necessario accertarsi che non siano presenti caratteri estranei.

2 1 pixel = 15 twips

Figura 6.1: La funzione InputBox

6.2 LA FUNZIONE ISNUMERIC

Quando si inserisce un dato può essere utile verificare se esso è coerente con quanto richiesto. La funzione `IsNumeric` si rivela di grande utilità a questo fine perché controlla che nel suo argomento siano presenti soltanto caratteri numerici. Il risultato di questa funzione è una variabile booleana (ne avevamo parlato al termine del paragrafo 2.4) e sarà Vero qualora il test sulla presenza di sole cifre sia stato favorevole.

Codice 6.2: Ciclo di inserimento con InputBox

```
  . . .
Do              ' ciclo per la richiesta di un dato
          inputUtente = InputBox(" Inserisci  un dato", _
                                 "FINESTRA DI RICHIESTA DATO", _
                                 "", 500, 800)
          If inputUtente = "" Then Exit Do
          Cells(I, 1).Value = inputUtente
          I = I + 1
          Loop Until inputUtente = ""
  . . .
```

La procedura `prova_isnumeric` parzialmente visibile nel codice 6.3, calcola il risultato della funzione in relazione a diversi argomenti. Nei primi due esempi, in cui l'argomento è una stringa numerica o un numero, la funzione restituisce Vero[3].

Codice 6.3: Risultato di IsNumeric

```
' risultato Vero
MsgBox "Il risultato di IsNumeric(123) e' " & IsNumeric(123)
' risultato Vero
MsgBox "Il risultato di IsNumeric(""123"") e' " & IsNumeric("123")
' risultato Falso
MsgBox "Il risultato di IsNumeric("""") e' " & IsNumeric("")
```

3 In questo esempio e nel successivo si noti come, per utilizzare il riferimento ad una stringa all'interno di una costante stringa, debba essere scritta una sequenza di doppi apici.

```
' risultato Vero
MsgBox "Il risultato di IsNumeric su una cella vuota e' " & IsNumeric(Cells(1,
    1).Value)
a = Cells(1, 1).Value
MsgBox "Il valore di a dopo l'assegnazione di una cella vuota e' " _
                & Val(a)   ' questo spiega IsNumeric(cellavuota)=True
```

Nel terzo esempio la funzione `IsNumeric`, calcolata su una stringa vuota, ha come risultato Falso.

La sequenza di istruzioni dalla quarta riga in poi, vuole dimostrare cosa accade quando applichiamo la funzione ad una cella vuota. In questo caso la funzione restituisce True, risultato che si spiega con il fatto che per Excel una cella vuota contiene un valore pari a zero. A dimostrazione di questo, si considerino le righe successive in cui il contenuto della cella viene assegnato alla variabile a.

Sulla base di queste premesse, abbiamo realizzato la procedura `InserisceInputNumerico` che si occupa di controllare che il dato inserito sia numerico. In caso negativo si visualizza un messaggio di errore altrimenti si inserisce il dato in una cella a partire da ... Con l'inserimento di un dato "vuoto" si intende terminata l'acquisizione.

Codice 6.4: Acquisizione e controllo dell'input

```
Sub InserisceInputNumerico()            ' inserisce una sequenza di dati
    numerici
    Dim inputUtente As String           ' a partire da A1
    Dim ok As Boolean
    Dim I As Integer
    I = 1
    Do
        inputUtente = InputBox(" Inserisci un dato numerico." & Chr(13) & _
                                "N.B. per terminare non inserire niente e
                                    premere OK", _
                                  "ACQUISISCE INPUT NUMERICO
                                      DA TASTIERA", _
                                  "", 500, 800)
        If IsNumeric(inputUtente) = False Then
' se accedo qui e' x 2 motivi:
'             a. dato non numerico
'             b. dato vuoto (lunghezza = 0) (questo caso non lo
    considero ma terminera' il ciclo)
            If Len(inputUtente) > 0 Then MsgBox _
                        "Il dato inserito " & inputUtente & " non e'
                            numerico"
                Else
                    Cells(I, 1).Value = CStr(inputUtente)     ' input pieno,
                        inserisce il dato
                    I = I + 1
                    End If
        Loop While inputUtente <> ""
End Sub
```

La procedura `InserisceInputNumerico_varianteSelect` che si trova di seguito a quella appena vista, è la variante con l'uso del comando Select.

6.3 LA FUNZIONE AD UN RISULTATO

Nel capitolo dedicato alle funzioni abbiamo discusso delle modalità di richiamo delle funzioni sia di Excel che di VBA. In questo paragrafo e nei successivi parleremo di funzioni definite dall'utente. Consideriamo il seguente codice:

Una funzione definita dall'utente e ..

```
Function  cicciooriccio () As String
      cicciooriccio  = "123sftgrtg"
End Function
```

Nell'esempio `cicciooriccio`, è il nome di una funzione di tipo stringa. Nella seconda riga abbiamo un'istruzione di assegnazione di una costante stringa ("123sftgrtg") ad una variabile. Si nota, fatto non casuale, che questa variabile ha un nome identico a quello della funzione. Dopo aver scritto il codice che richiama questa funzione e proceduto alla sua esecuzione non dovrebbe essere difficile comprenderne il funzionamento.

.. la chiamata alla funzione.

```
Sub prova_function()
    MsgBox  cicciooriccio
End Sub
```

Alla luce di quanto abbiamo appena visto è possibile dire che:

- l'utente può definire funzioni personalizzate con o senza argomenti (nell'esempio la funzione era senza argomenti);
- all'interno della funzione `Function` deve trovarsi un'assegnazione ad una variabile che ha lo stesso nome e tipo della funzione;
- la funzione definita dall'utente si comporta come una qualsiasi funzione VBA;
- la funzione restituisce un solo risultato.

Nel caso di una `Function` con argomenti la variante consiste nella dichiarazione di questi tra le parentesi. Si consideri il seguente codice:

Una funzione ad un solo risultato e la sua chiamata

```
Function richiesta_dato_numerico_Cargomento(domanda As String) As String
    Dim inputUtente As String
    Do
        inputUtente = InputBox(domanda, "ACQUISISCE DATO NUMERICO", _
                      "", 500, 800)
        If IsNumeric(inputUtente) = False Then
                MsgBox "Il dato richiesto e' mancante o sbagliato", vbCritical
        ' potrebbe trattarsi anche di un input
        ' che contiene caratteri non numerici
        ' per garantire la prosecuzione del ciclo
        ' devo impostare a "vuoto" il contenuto della variabile
            inputUtente
                    inputUtente = ""
                    End If
        Loop While inputUtente = ""
    richiesta_dato_numerico_Cargomento = inputUtente
End Function
' ********************** la chiamata alla funzione
Sub prova_function_rdm1()
    Dim domanda As String
    domanda = "Quanti anni hai?"
    MsgBox richiesta_dato_numerico_Cargomento(domanda)
End Sub
```

Nell'esempio la funzione consta di un solo argomento. Al suo interno è definito un ciclo che si ripete fino a quando non è stato inserito un valore numerico[4].

6.4 LA FUNZIONE A PIÚ RISULTATI

Quando una funzione deve calcolare più di un risultato si usa il comando Sub al posto di Function.

Nell'esempio seguente risolveremo il problema del calcolo delle radici di una equazione di secondo grado. Per semplificare abbiamo scritto i tre coefficienti a, b, c, tra gli argomenti del programma chiamante e li abbiamo definiti come numeri interi.

La chiamata e . .

```
Sub chiama_equazione_IIgrado()
    Dim a As Integer, b As Integer, c As Integer
    Dim x1 As Single, x2 As Single
    Dim esito As Boolean
        equazione_2grado_sub x1, x2, 2, 4, 1, esito
        If esito = False Then
            MsgBox "Due soluzioni immaginarie"
            Exit Sub
```

4 Al momento, per semplicità, escluderemo che l'utente inserisca il separatore dei decimali. Il problema verrà affontato in uno dei prossimi capitoli.

```
        End If
    MsgBox "X1= " & CStr(x1) & Chr\$(13) & "X2= " & CStr(x2)
End Sub
```

Nel programma chiamante si nota, a differenza di quanto avviene con la Function, la mancanza di parentesi. Si noti anche la presenza, come ultimo argomento di una variabile booleana utile, dopo la chiamata del sottoprogramma, a conoscere nel programma chiamante l'esito dei calcoli: due soluzioni immaginarie/due soluzioni (distinte o coincidenti). In questo modo sapremo se procedere o meno al calcolo delle due radici.

Il sottoprogramma è abbastanza simile alla funzione. Per ovvi motivi il sottoprogramma non dichiara alcun tipo di dato. Si noti la conversione del tipo della variabile `delta` in quanto la funzione Sqr richiede un argomento Double[5].

.. la funzione a diversi risultati

```
Sub equazione_2grado_sub(x1 As Single, x2 As Single, a As Integer, _
        b As Integer, c As Integer, sw_ok As Boolean)
' eventuale dichiarazione di variabili non incluse tra gli argomenti
' decido di non trasferire al chiamante delta dunque qui scrivero'
    Dim delta As Integer
' calcolo delta
    delta = b ^ 2 - 4 * a * c
    sw_ok = True
    If delta < 0 Then
                sw_ok = False
        Else
' calcolo di x1 e x2 ATTENZIONE alle conversioni
                x1 = (-CSng(b) - CSng(Sqr(CDbl(delta)))) / CSng(2 * a)
                x2 = (-CSng(b) + CSng(Sqr(CDbl(delta)))) / CSng(2 * a)
        End If
End Sub
```

ESERCIZI

1. Un programma deve chiedere l'inserimento di tre dati numerici in tre celle del foglio scelte a piacere. Le istruzioni dovranno prevedere un ciclo che si ripete fintantoché tutti e tre i numeri non sono stati acquisiti.

2. Scrivere la Function che calcola le soluzioni di una equazione di II^ grado dati i tre coefficienti a, b, c, che si trovano in altrettante celle del foglio. Oviamente, considerato di non poter calcolare

5 Sarebbe stato più semplice definire a, b, c come variabili di tipo `Single`. La scelta è dovuta a motivi di coerenza con quanto fatto nel paragrafo 4.1.

i valori x1 e x2 come visto in precedenza, la funzione restituirà il risultato sotto forma di messaggio come "due soluzioni reali e distinte", oppure "due soluzioni immaginarie" etc.

3. Come nel problema n.2 tenendo presente che i tre coefficienti a, b, c, inseriti nelle celle sono dichiarate Single.

7 LA PROGRAMMAZIONE AD OGGETTI IN VBA

Le prime idee relative al concetto di "classi di oggetti" risalgono ad Aristotele (300 a.C) quando il filosofo parlava di "classe dei pesci e degli uccelli". Il fatto che degli oggetti siano unici ed allo stesso tempo parte di un insieme di elementi aventi caratteristiche e comportamenti comuni, è il concetto su cui si basano i linguaggi di programmazione ad oggetti.

L'idea di linguaggi di programmazione ad oggetti (Object Oriented Programming ovvero OOP) nasce intorno agli anni '80 e si basa sul fatto che un programma, così come il mondo circostante, si compone di oggetti che hanno proprietà e qualità particolari. In questo modo, i dati di un programma (anch'essi oggetti), hanno le loro proprietà e qualità che, per effetto della loro trasmissibilità, chiamata "ereditarietà" , facilitano notevolmente la programmazione[1].

7.1 PROPRIETÀ E METODI

Nella terminologia OOP, con il termine "proprietà", indichiamo le caratteristiche dell'oggetto equiparabili, nel linguaggio naturale, agli aggettivi ovvero alle qualità.

Tutti noi conosciamo l'oggetto *Radio*. Le sue qualità sono costituite dal modo di funzionare (analogica o digitale), oppure dal suo colore (bianco o verde), oppure dal suo peso. Proseguendo nell'analogia possiamo anche capire come alcune proprietà siano modificabili mentre altre non lo sono: posso cambiare il colore della radio, ma non posso modificare l'insieme delle frequenze che riceve.

Sugli oggetti possiamo poi eseguire determinate azioni che, nella terminologia OOP, si chiamano "metodi". Sempre in riferimento all'oggetto *Radio* possiamo eseguire un'azione costituita dall'accendersi, lo spegnersi, il sintonizzarsi, l'aumento o la diminuzione del volume.

Il vantaggio della programmazione ad oggetti è che il programma non si preoccupa di come fare ad eseguire l'azione, esso deve semplicemente chiedere di eseguirla. Dunque, se voglio accendere/spegnere la radio, eseguirò il metodo "spegni" oppure "accendi" senza preoccuparmi di come questo sia realizzato fisicamente nell'oggetto.

1 Se nel nostro programma creiamo un oggetto, copiandolo da uno preesistente, esso erediterà tutte le qualita/proprietà da quello originale senza bisogno di altro.

7.2 COME SI PROGRAMMA UN OGGETTO

Nell'OOP, quindi anche in VBA, la programmazione di un oggetto si espleta attraverso una delle seguenti attività:

- leggere/scrivere una proprietà;
- eseguire un metodo.

La sintassi per modificare la proprietà o applicare un metodo ad un oggetto è costituita da:

nome_dell'oggetto.nome_di_proprietà | *nome_di_metodo*

in cui dopo il punto, necessario a distinguere il nome dell'oggetto dal resto, bisogna specificare un nome valido di proprietà oppure di metodo.

7.3 LE PROPRIETÀ IN VBA

Le proprietà di un oggetto possono essere lette oppure assegnate. A questo proposito, l'esempio che ormai ci dovrebbe essere familiare, è quello che assegna un valore ad una proprietà come nel caso dell'istruzione:

```
. . .
Cells(21, 1).Value = Gio
. . .
```

Nell'esempio `Cells(21,1)` rappresenta l'oggetto; `Value` è la proprietà costituita dal contenuto (il valore); `Gio` è un'espressione nella forma più semplice (il nome di una variabile).

In generale, nel caso di un'assegnazione, la sintassi sarà del tipo:

Oggetto.Proprietà = espressione

mentre per la lettura, il riferimento all'oggetto ed alla proprietà, potrà essere ottenuto riferito in vari modi (si veda il codice poco oltre).

Si tenga presente che *espressione* è una qualsiasi espressione che abbia come risultato un valore compatibile con la proprietà (`Value` nel caso del nostro esempio) riferita all'oggetto `Cells` in base al criterio della compatibilità. Infine `Proprietà` è una qualsiasi proprietà valida per l'oggetto `Cells`.

Nella cartella di lavoro `proprieta_metodi.xls` sono presenti diverse procedure ricche di esempi significativi. I commenti a fianco delle istruzioni dovrebbero bastare a chiarirne il significato.

La prima di queste, denominata `proprieta`, visualizza con una funzione MsgBox, le proprietà di altrettanti oggetti del foglio. In tutti questi esempi il carattere "." separa l'oggetto (a sinistra) dalla proprietà (a destra).

Le proprietà di alcuni oggetti

```
        . . .
'                la proprieta' Name di ActiveSheet
    MsgBox ActiveSheet.Name, , "Il foglio attivo"
'                l'oggetto e' la cartella di lavoro
    MsgBox ThisWorkbook.Name, , "La cartella di lavoro"
    MsgBox ThisWorkbook.Path, , "Unita' e sottodir."
    MsgBox ThisWorkbook.FullName, , "Unita',sottodir., cartella di lavoro"
'                l'oggetto e' la cella corrente
    MsgBox ActiveCell.Address, , "L' indirizzo della cella corrente"
    MsgBox ActiveCell.Row, , "Il numero di riga della cella corrente"
    MsgBox ActiveCell.Column, , "Il numero di colonna della cella corrente"
    MsgBox Cells(1, 1).Address, , "L' indirizzo della cella A1"
    MsgBox Range("A3").Column, , "Il numero di colonna di una cella"
' esempio di comando di assegnazione di un valore alla proprieta'
    Formula dell'oggetto Cells
' si noti
    Cells(10, 3).Formula = "=B3*2"
        . . .
```

Solo una notazione sulla scrittura di una formula in una cella. Nell'ultima istruzione dell'esempio si nota che la formula deve essere racchiusa tra doppi apici (come se si trattasse di una stringa).

7.4 I METODI IN VBA

La sintassi per l'esecuzione di un metodo si esprime generalmente nella forma:

Oggetto.Metodo

Nel codice della procedura metodi_selezione_celleZone, abbiamo riportato diversi esempi del metodo di selezione di una zona di celle Tutti i metodi della procedura ipotizzano che sia attivo il foglio di lavoro cui si riferiscono le zone di celle. Diversamente occorrerà specificare esplicitamente quest'ultimo tenendo presente che la gerarchia dell'oggetto `Range` è:

```
Application(..).Workbook(..).Worksheet(..).Range(..)
```

in cui le parentesi si riferiscono agli eventuali argomenti richiesti da ciascun elemento della gerarchia.

La selezione di Celle/Zone

La programmazione degli oggetti definisce un grande numero di metodi per la selezione di celle o zone del foglio[2]. In questo primo

2 Per una trattazione completa su questo argomento si consulti http://support.microsoft.com/kb/291308/it in cui sono descritti minuziosa-

esempio vediamo come eseguire l'azione di selezione in relazione all'oggetto zona di celle compreso nell'intervallo C2 D10.

```
Sub metodi_selezione_celleZone()
        . . .
    Range("C2:D10").Select
' se levo il commento al comando successivo,
' notero' la zona evidenziata nel foglio Excel
'    Exit Sub
```

Per capire bene il suo effetto sul foglio Excel è consigliabile eliminare il carattere di commento che abbiamo inserito davanti all'istruzione di uscita dalla procedura per evitare di eseguire le istruzioni successive.

La sintassi definita all'inizio del paragrafo relativa all' esecuzione di un metodo non sempre è così "lineare". Nel prossimo esempio la zona di selezione, costituita da una sola cella, è individuata in base ad uno spostamento rispetto alla cella corrente.

```
        . . .

' si usa il riferimento denominato R1C1
' dunque 5 righe in avanti, 4 colonne indietro
' rispetto alla cella F7
    Range("F7").Offset(5, -4).Select
    Exit Sub
```

Il commento presente davanti al comando rende inutile ogni ulteriore considerazione. Si tenga presente che `Offset(5, 4)` è una proprietà che restituisce un oggetto. La riprova di questo si può avere consultando la finestra che visualizza gli oggetti del foglio con la procedura che descriviamo:

- aprire il Visualizzatore degli oggetti del foglio (tasto F2);
- selezionare Excel nel menu a discesa che si trova in alto a sinistra;
- nel pannello di sinistra, intitolato Classi, selezionare Range;
- selezionare Offset nella parte destra. Visualizzando il pannello inferiore (la finestra deve essere massimizzata) si vedrà che Offset restituisce un oggetto Range.

Il prossimo esempio è particolarmente importante perché seleziona la zona di celle piene contigue a partire da una cella predefinita.

---

mente parecchi metodi relativi alla selezione di zone di celle. Qualora il link non funzionasse correttamente si inserisca il termine *"How to select cells/ranges by using Visual Basic procedures in Excel"* in un motore di ricerca per ottenere lo stesso risultato.

```
. . .
ActiveSheet.Range("A1").CurrentRegion.Select
. . .
```

Particolarmente importante, per ovvi motivi, è la combinazione di comandi che effettua il Copia/Incolla di una zona di celle. A questo proposito consideriamo il codice 7.1.

Nella prima riga abbiamo eseguito la selezione di un zona di celle. In questo modo viene generato un oggetto Selection che è quello cui viene applicato il metodo di copia utilizzato nella seconda riga. Il terzo comando incolla a partire dalla cella E5. Il quarto e quinto comando servono rispettivamente ad eliminare la selezione della zona di output[3], ed a svuotare gli Appunti (tasto Esc in Excel).

Codice 7.1: Copia e incolla

```
. . .
Range("C2:D10").Select
Selection.Copy
Range("E5").PasteSpecial
Range("A6").Select
Application.CutCopyMode = False
. . .
```

Se i comandi necessari, ben 5 istruzioni, sono troppi, si può ricorrere ad una forma molto più sintetica che ottiene lo stesso risultato:

```
. . .
Range("C2:D10").Copy Destination:=Range("E5")
. . .
```

In questo caso `Destination` serve a definire il valore di un parametro del metodo Copy (si noti l'uso di ":=" , una forma sintattica analizzata nel paragrafo 3.4).

Una corretta valutazione dei due metodi, deve però tenere presente che il secondo di questi non è utilizzabile qualora si debbano effettuare copie multiple della stessa zona.

7.5 LA CANCELLAZIONE DI UNA ZONA

Un altro metodo utile è quello che cancella il contenuto di una zona. Il codice seguente ne dimostra un esempio: Più interessante, rispetto

3 La tecnica di selezione di una cella dovrà essere utilizzata in tutti i casi in cui si vuole "deselezionare" una zona come vedremo nel caso di cancellazione del contenuto di celle

```
    Range("F7:E5").Clear
```

alla generalizzazione del metodo, è il codice della prossima procedura. Se l'argomento di Range è una stringa di caratteri, è possibile esprimere la zona sotto forma di variabile. Di conseguenza sarà possibile il riutilizzo di questa procedura, opportunamente modificata, per essere generalizzata.

La cancellazione di una zona

```
Sub metodo_cancellaZona2()
    Dim numero_di_riga As Byte
    Dim ciccio As String
    numero_di_riga = 7
     ciccio = "E5:F" & CStr(numero_di_riga)
    Range(ciccio).Clear
End Sub
```

7.6 LE COLLEZIONI DI OGGETTI

All'inizio del capitolo abbiamo accennato alle classi di oggetti. Un concetto notevole su cui è necessario soffermarsi riguarda le c.d. "collezioni". Le collezioni sono insiemi di oggetti accomunati dalle stesse caratteristiche (metodi e proprietà). Esse stanno agli oggetti che le compongono come le variabili con indici stanno ai loro elementi. Nel caso del foglio Excel le collezioni predefinite[4] più importanti sono quelle che si riferiscono ad insiemi di:

- righe/colonne chiamati rispettivamente Rows e Columns;
- fogli chiamati Sheets;
- grafici chiamati Charts.

Nel primo esempio viene calcolata la proprietà Count che applicata alle righe di una zona (questa è la collezione) ne conteggia il numero. Si noti come nella stessa riga di codice sia possibile prima riferire l'oggetto zona di celle (a destra dell'istruzione di assegnazione) e contestalmente crearne uno nuovo con il comando `Set`

4 Come vedremo tra poco è possibile creare oggetti e quindi anche collezioni. Agli oggetti creati dall'utente, a differenza di quelli predefiniti, possono essere assegnate proprietà a piacimento.

```
' questa e' la dichiarazione dell'oggetto
    Dim esempio As Range
' creo l'oggetto esempio e gli assegno un range.
' A questo punto esempio
' e' l'oggetto "zona di celle A1:C4"
    Set esempio = Range("A1:C4")
' per il principio dell'EREDITARIETA',
' all'oggetto "esempio"
' si possono applicare le proprieta' (ed i metodi)
' validi per l'oggetto di cui e' copia
    MsgBox esempio.Rows.Count            ' ritorna 4
    MsgBox esempio.Columns.Count         ' ritorna 3
```

Le collezioni dispongono di strumenti in grado di gestire gli elementi che le compongono uno per uno. Nel seguente esempio si effettua un ciclo su tutti i fogli della cartella di lavoro corrente; quelli con un prefisso del nome diverso da "Foglio" vengono eliminati.

Nella fattispecie `ActiveWorkbook.Sheets` è un oggetto costituito dalla collezione di tutti i fogli della cartella attiva, `aSheet.Name` si riferisce al nome di ogni foglio (proprietà), `Sheets(aSheet.Name)` è un oggetto della collezione cui viene eventualmente applicato il metodo di cancellazione. Si noti anche la novità costituita dal modo di esprimere un ciclo applicato a tutti gli elementi di una collezione.

Il ciclo su una collezione

```
Dim aSheet As Object
    For Each aSheet In ActiveWorkbook.Sheets
'       eliminiamo tutti i worksheet con nome differente da Foglio(n)
        If Left(aSheet.Name, 6) <> "Foglio" Then
                Sheets(aSheet.Name).Delete
            End If
        Next
```

7.7 L'OGGETTO EXCEL

In alcuni casi, nel codice VBA, è necessario riferirsi al programma Excel come, p. e., quando dobbiamo utilizzare una funzione di questo programma che non ha corrispondenti in VBA. Il problema era stato posto, senza che venisse data una spiegazione, nel paragrafo **??**. Abbiamo ora la possibilità di risolvere il problema con il riferimento all'oggetto `Application` che in VBA è il nome del programma Excel. Così la somma di una zona di celle sarà calcolata con la funzione Sum che richiede come argomento l'oggetto `Range`. Nella stessa procedura viene impiegata un'altra interessante funzione del foglio che calcola il numero di celle piene di una zona. Si ricorda che il nome della

funzione Excel debba essere specificato in inglese.

```
Sub sintassi_funzione_conZona()
    Dim totCelle As Integer, conta_piene As Integer
' Sum e' l'equivalente inglese della funzione SOMMA
    totCelle = Application.Sum(Range("A1:B3"))
    MsgBox "la somma dei numeri in A1:B3 e' " & CStr(totCelle)
' CountA e' l'equivalente inglese della funzione CONTAVALORI
    conta_piene = Application.CountA(Range("A1:C4"))
    MsgBox "le celle piene in A1:C4 sono " & CStr(conta_piene)
End Sub
```

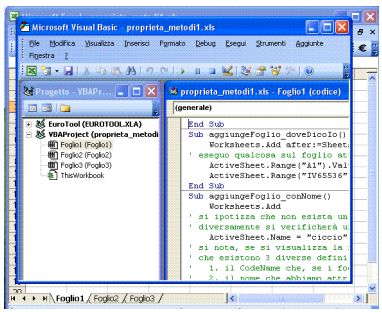
Il riferimento allo stesso oggetto, il programma Excel, viene utilizzato nel prossimo esempio in cui vediamo i comandi che chiudono una cartella di lavoro aperta. Si tratta dell'equivalente dei comandi del menu Excel File → Esci.

La prima riga della procedura ha la funzione di evitare la richiesta di salvare il file nel caso in cui fossero intervenute delle modifiche. Per capire meglio si provi a commentare tale riga senza salvare e ad eseguire la macro.

```
    . . .
ActiveWorkbook.Saved = True
Application.Quit
    . . .
```

7.8 LA GESTIONE DEI FOGLI

Nella programmazione VBA risulta di notevole importanza la gestione dei fogli di una cartella di lavoro. Negli esempi che andiamo ad illustrare, relativi a questo argomento, è consigliabile partire da una cartella di lavoro vuota. É consigliabile inoltre, per comprendere bene cosa accade all'esecuzione del codice, tenere aperta la finestra VBA in modo da vedere, in secondo piano, la finestra Excel nella parte che visualizza le linguette corrispondenti ai fogli come evidenziato nella figura seguente.



Come tutti sappiamo il programma Excel visualizza tre fogli di lavoro denominati, nella versione italiana Foglio1, Foglio2, Foglio3. Le istruzioni VBA per inserire un nuovo foglio nella cartella di lavoro sono:

```
    . . .
Dim WS As Worksheet
Set WS = Sheets.Add
    . . .
```

In questo esempio si procede dichiarando di un oggetto "foglio" e dalla sua creazione con un'istruzione di assegnazione a partire da un metodo (a destra dell'uguale). Eseguita questa procedura vediamo che è stato creato un nuovo foglio chiamato Foglio4 che si trova davanti a quello attivo prima della sua esecuzione. Volendo definire un comportamento diverso, dobbiamo ricorrere ad una variante del codice precedente costituita da:

```
Sheets.Add after:=Sheets(Sheets.Count)
```

che si differenzia dal precedente oltreché per la posizione di inserimento, anche perché l'oggetto non ha un riferimento costituito dal nome[5]

Con i comandi visti fono ad ora, ai nuovi fogli veniva dato un nome costituito dal prefisso Foglio seguito da un numero progressivo. Volendo attribuire un nome diverso possiamo agire sulla proprietà costituita dal nome dell'oggetto attribuendogli un nome diverso. Il codice per fare questo è dato da:

```
  Worksheets.Add
' si ipotizza che non esista un foglio con lo stesso nome
' diversamente si verifichera' una condizione di errore
  ActiveSheet.Name = "ciccio"
```

Tutti gli esempi appena visti, a condizione di essere partiti da una cartella di lavoro vuota, dovrebbero aver generato una situazione che può riassumersi nella figura seguente. Si notino le differenze tra la finestra del progetto e quella di Excel di cui sono visibili le linguette. Nella finestra del progetto i fogli sono ordinati in un qualche modo, in quella di Excel i fogli sono disposti in riferimento al foglio attivo prima del loro inserimento.

5 Per ognuno dei nuovi modi di inserire un foglio, si raccomanda di commentare le istruzioni eseguite in precedenza se fanno parte della stessa procedura o, in alternativa, di definire una procedura diversa. La ragione di questo sarà evidente nel seguito.

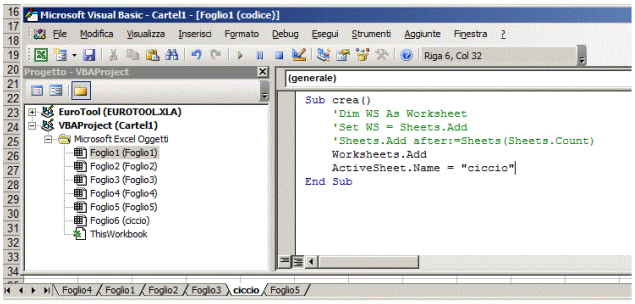


Figura 7.1: I fogli aggiunti

Per chiarire meglio questo comportamento, è bene precisare che ogni foglio di una cartella ha tre identificatori chiamati rispettivamente *Code Name, Tab Name, Index Number*.

Il *Code Name* è quello che nella finestra del progetto si presenta per primo accanto al simbolo del foglio a sinistra. Nella figura 7.1 l'ultimo *Code Name* è Foglio6.

Il *Tab Name* si riferisce alla linguetta visibile nel foglio Excel. La posizione di questa è correlata a quella del foglio attivo prima del suo inserimento. Il suo valore può essere gestito tramite il codice VBA.

L'*Index Number* è un indice numerico corrispondente all'ordinale di quel foglio nella finestra del progetto. Dunque 1 per il primo foglio, 2 per il secondo, ..., e così via. La sua gestione è a totale appannaggio del programma Excel. Resta il fatto che l'utente può conoscerne il contenuto. Si consideri il seguente esempio:

```
MsgBox ActiveWorkbook.Sheets("ciccio").Index
Sheets(3). Select
```

La prima istruzione visualizza l'Index Number, pari a 6, del foglio `ciccio`. La seconda costituisce un esempio di attivazione di un foglio grazie a codesto indice.

7.9 MISCELLANEA

La trattazione completa di tutti i metodi e le proprietà di VBA non è tra gli obiettivi di questo libro. In questo paragrafo ci occuperemo di illustrare altri metodi e proprietà che verranno impiegati nei prossimi capitoli.

Tra le funzionalità più utilizzate nella gestione delle celle del foglio possiamo annoverare quelle che si occupano di formato colore ed allineamento dei dati, occultamento o visualizzazione di righe o colonne, etc. Nel codice sottostante sono raccolti alcuni comandi che si occupano di queste funzionalità.

```
    . . .
    Range("B3:C2").NumberFormat = "0"
    Range("B2").Font.Bold = True
    Range("C2").Font. Italic  = True
    Range("D2").Font.Underline = xlUnderlineStyleSingle
    Range("B4").HorizontalAlignment = xlCenter
    . . .
```

Qui vediamo i vari comandi di formattazione del testo. Potremmo poi aver bisogno di definire colore di fondo o del carattere. Ricordando che nello scrivere il nome di un oggetto, premuto il carattere ".", compare la finestra che visualizza quelli disponibili in relazione all'oggetto, consideriamo il seguente un esempio:

```
    . . .
    Range("B2:C4").Interior.Color = vbYellow
    . . .
```

Si tratta del codice che definisce il colore di fondo di una zona. Volendo inserire dei bordi per le celle della zona dobbiamo utilizzare il codice sottostante:

```
    . . .
    Range("B2:C4").Select
    With Selection.Borders(xlEdgeLeft)
        .LineStyle = xlContinuous
        .Weight = xlThin
        .ColorIndex = xlAutomatic
    End With
    With Selection.Borders(xlEdgeTop)
        .LineStyle = xlContinuous
        .Weight = xlThin
        .ColorIndex = xlAutomatic
    End With
    With Selection.Borders(xlEdgeBottom)
        .LineStyle = xlContinuous
        .Weight = xlThin
        .ColorIndex = xlAutomatic
    End With
    With Selection.Borders(xlEdgeRight)
        .LineStyle = xlContinuous
        .Weight = xlThin
        .ColorIndex = xlAutomatic
    End With
' queste due istruzioni servono ad inserire
' le righe orizzontali/verticali interne alla zona
    Range("B2:C4").Borders.LineStyle = xlContinuous
    Range("B2:C4").Borders.Weight = xlThin
    . . .
```

ESERCIZI

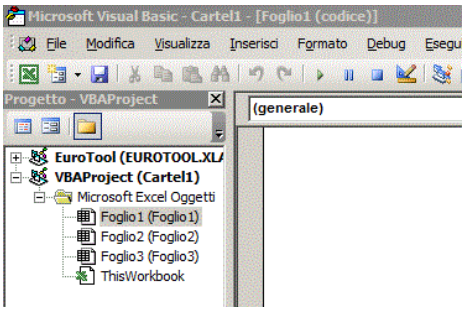
1. Assegnato il risultato di `Cells(5,2).Address` ad una opportuna variabile, si scriva la funzione che trasforma tale risultato da riferimento assoluto a riferimento relativo. Si tenga presente il risultato di `MsgBox Cells(3,3).Address` visto dianzi.

2. La colonna A, a partire da una riga qualsiasi, contiene una sequenza di celle piene. Contare le celle piene della zona utilizzando 2 differenti modalità.

8

LE LIBRERIE DI PROGRAMMI

Nei programmi presentati fino ad ora, il codice era stato scritto senza prestare attenzione alla sua collocazione. In tal caso esso si trova associato al foglio Foglio1. Una semplice verifica di questo si può avere sia aprendo una qualunque delle cartelle riferite nei capitoli precedenti, sia aprendo un foglio vuoto. Da un esame della figura, che si riferisce



allo stato della finestra VBA relativa ad un foglio vuoto, si può notare quale sia il foglio associato al codice eventualmente inserito. Si faccia caso infatti alla finestra del progetto nonché al titolo riportato nella finestra VBA.

Si supponga di avere una procedura che viene utilizzata da altre procedure che si trovano in diverse cartelle. Un semplice esempio potrebbe essere rappresentato dalla richiesta di input numerico visto in 6.3. Al momento attuale, la nostra unica risorsa sarebbe quella di copiarla nelle cartelle che la richiedono generando una forma di ridondanza che potrebbe diventare causa di non pochi problemi[1].

Un'altra fonte di problemi potebbe essere rappresentata da un cartella che si compone di un elevato numero di procedure. Tutti i problemi appena accennati, hanno una soluzione costituita dai concetti di modulo e di libreria. Vediamo di cosa si tratta.

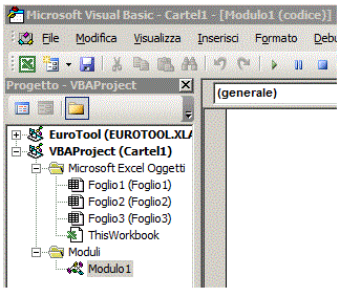
8.1 IL MODULO

Quando il numero di procedure di una cartella di lavoro incomincia a diventare cospicuo può ritornare utile organizzarle in una qualche forma. Il primo mattone di questa organizzazione è costituito dal concetto di *modulo* definibile come una raccolta di procedure.

Un modulo può essere creato con la sequenza di comandi Inserisci → Modulo. Dopo aver eseguito questa operazione si noterà, nella finestra del progetto la comparsa di una cartella denominata Moduli ed al

1 Si pensi al lavoro da fare qualora si dovesse procedere all'aggiornamento di una procedura copiata in svariate cartelle!

suo interno la comparsa del modulo Modulo1. Contestualmente verrà aggiornata l'intestazione nella barra del titolo della finestra VB. Un



modulo può essere eliminato con la sua selezione nella finestra del progetto quindi con l'apertura del menu contestuale che presenta, tra gli altri il comando Rimuovi seguito al nome del modulo che si vuole elminare.

8.2 LA LIBRERIA

La libreria è un insieme di moduli organizzati in una qualche forma decisa dall'utente in base alle sue preferenze. Con un'analogia possiamo pensare all'organizzazione di file in cartelle distinte in base al contenuto: una per i file musicali, un'altra per le foto, un'altra per i film. La libreria è qualcosa di analogo applicata alle procedure. Nel nostro caso, abbiamo deciso di realizzare una libreria suddividendola in due moduli: uno per le procedure fondamentali ed un altro per procedura varie.

Il modulo delle procedure fondamentali, sarà costituito dagli esempi di codice relativi ai fondamentali del Vba come per esempio le istruzioni di assegnazione, di condizione, di ripetizione e quant'altro. Il modulo delle procedure varie includerà procedure di servizio quali per esempio quello della richiesta di un input numerico e simili.

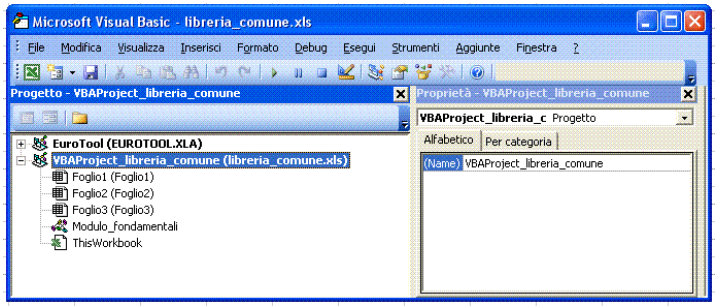
La creazione di una libreria si compone di diversi passaggi che descriviamo di seguito:

1. a partire da una nuova cartella di lavoro creare un nuovo modulo come visto nel paragrafo precedente;
2. modificare il nome del progetto (obbligatorio!) ed il nome del modulo (solo per motivi mnemonici);
   - selezionare il corrispondente elemento nella finestra Progetto;
   - modificare l'attributo (Name) nella finestra Proprietà. Noi abbiamo scelto `VBAProject_libreria_comune` per il progetto e `Modulo_fondamentali` per il modulo;

3. salvare il foglio di lavoro così com'è con un opportuno nome (abbiamo scelto `libreria_comune.xls`) e chiudere il foglio di lavoro;

4. aprire il foglio di lavoro da cui si vuole prelevare il codice da mettere in libreria e con un copia/incolla inserirlo nel foglio di lavoro (`libreria_comune` nel nostro esempio) avendo l'accortezza di inserirlo nel modulo che riteniamo più opportuno.

La figura sottostante riproduce la situazione al termine delle operazioni appena descritte. Si ribadisce l'importanza del punto 3 allorché si seleziona la voce `VBAProject` in quanto è questa operazione che influenza il cambiamento del nome della libreria.



Resta da vedere come effettuare il collegamento da parte di un progetto chiamante a uno dei progetti facenti parte della libreria. Nel semplice esempio che presentiamo, faremo ricorso alla funzione `RichiestaDatoNumerico` richiamandola da un programma che scrive tre numeri in altrettante celle del foglio.

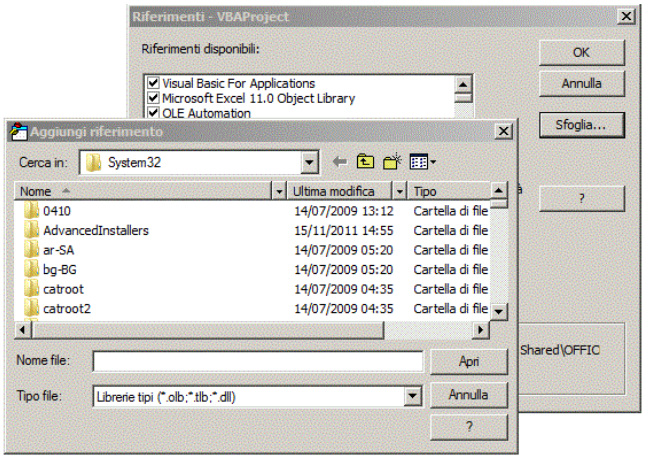
8.3 LA CHIAMATA ALLA LIBRERIA

Supponendo che la cartella di lavoro `libreria_comune.xls` contenga la funzione di inserimento di un dato numerico[2], procediamo all'aggancio o link della libreria da parte del programma che deve utilizzare una delle sue procedure. Dall'editor VBA della cartella da cui volgiamo collegarci alla libreria dovremo:

1. eseguire il comando del menu Strumenti –> Riferimenti. Si apre la finestra di dialogo intitolata `Riferimenti - VBAProject`;

2. premere il pulsante Sfoglia. Si apre la finestra di dialogo intitolata Aggiungi riferimento. A partire da questa finestra localizzare la posizione della cartella cui riferirsi selezionando per prima cosa il tipo di file attraverso la sua estensione, Microsoft Office Excel Files (*.xls,*.xla) nel nostro caso, scegliendo poi la cartella di lavoro corrispondente alla libreria nel primo campo della finestra;

2 A questo punto la condizione non è necessaria



3. confermare il tutto con la selezione del pulsante Apri; fatto questo la finestra di dialogo Riferimenti - VBAProject mostra un segno di spunta mentre nella finestra del progetto vedremo comparire il nome del progetto appena collegato;

4. terminare tutto con il pulsante OK.

Da questo momento in poi tutte le modifiche alla libreria comune si rifletteranno in automatico sul programma/i che contiene/contengono riferimenti a quella libreria (ricordarsi di salvare dopo queste modifiche la cartella di lavoro su cui abbiamo appena operato).

Se ancora non l'avessimo fatto, possiamo scrivere le istruzioni che fanno riferimento a procedure della libreria anche se, va detto per correttezza, questo fatto viene rilevato soltanto durante l'esecuzione.

Prima di chiudere il capitolo ricordiamo al lettore che uno degli errori più frequenti nell'uso delle librerie è quello di un collegamento non attivo con la libreria di programmi. Perciò, quando si dovesse verificare un errore su una procedura di una libreria chiamata da un qualche programma, verificare sempre l'esistenza di un collegamento alla cartella/libreria comune con il comando del menu del menu Strumenti –> Riferimenti. In particolare verificare la presenza del carattere di spunta corrispondente alla cartella che contiene la libreria.

ESERCIZI

1. La padronanza nell'uso delle librerie di programmi è di importanza fondamentale in VBA. Si consiglia perciò di ripetere più volte la creazioe e la chiamata di una libreria a partire da una cartella vuota.

Parte II

FINANZA

9 LE VARIABILI CON INDICE

Le variabili con indici costituiscono uno strumento molto utile nella programmazione. Questo è dovuto a due fattori:

- hanno una struttura concettuale estremamente semplice;
- il loro impiego riduce la quantità di codice da scrivere.

In questo capitolo, allo scopo di esemplificarne l'uso, ci occuperemo di calcolare una distribuzione di frequenza suddivisa in classi.

Una distribuzione di frequenza è il risultato di un procedimento di sintesi su un insieme di informazioni. Supponiamo di disporre di una serie di dati costituita dalle età di un gruppo di persone. Il procedimento di sintesi consiste, in questo caso, nel definire diversi gruppi di età e di contare nello stesso gruppo, le età appartenenti ad un determinato intervallo come quello delle persone aventi un'età compresa tra 20 e 30 anni[1]

9.1 DEFINIZIONE

Una variabile con indici in informatica è definita come una struttura astratta di dati[2] riconducibile ad un insieme omogeneo. I dati dell'insieme sono identificati dal un nome comune a tutti con l'aggiunta di un ulteriore identificatore chiamato *indice*. L'indice è sempre un numero intero, solitamente positivo, racchiuso tra parentesi tonde[3].

Se per esempio, vogliamo riferirci ad un insieme composto dai nomi dei frutti, possiamo dire che FRUTTA è il nome di questo insieme[4]. Così se associamo l'indice 1 al frutto mela, FRUTTA(1) identificherà il frutto mela, se associamo l'indice 2 al frutto arancia, FRUTTA(2) identificherà il frutto arancia 2 e così via. Ovviamente ciascun indice dovrà essere unico per quell'elemento dell'insieme.

9.2 LE VARIABILI CON DIVERSI INDICI

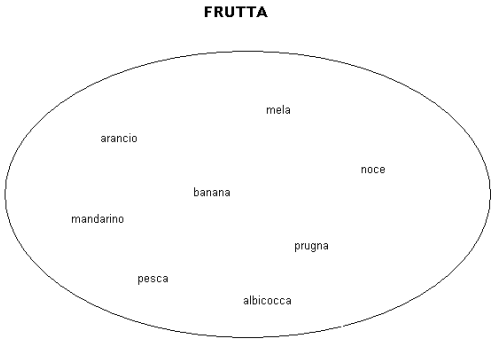
Nell'esempio visto in precedenza la variabile con indici aveva un solo indice. Quando però ciascun elemento della variabile è identificato da due, tre, .., indici si parla di variabile a due, tre, .., indici.

1 La funzione matrice Frequenza di Excel esegue questo calcolo. Richiede che le osservazioni si trovino nelle celle del foglio. Non sempre questa condizione può essere rispettata!

2 Il termine si riferisce al fatto che essa è concettualmente distinta rispetto al modo in cui è rappresentata fisicamente nel sistema di elaborazione.

3 In VBA l'indice potrà anche essere minore o uguale a zero.

4 Questo nome, definibile arbitrariamente, è stato scelto per motivi mnemonici.



L'esempio più semplice di variabile a 2 indici lo conosciamo dalle elementari ed è costituito della tabellina pitagorica: ciascuno dei suoi componenti è identificato da un indice di riga ed un indice di colonna. Quindi se Tabella_Pitagorica è il nome dell'insieme, potremo dire che 36 è un elemento di quell'insieme identificato dall'indice di riga 6 e dall'indice di colonna 6 e sarebbe:

Tabella_Pitagorica(6,6)=36

Un semplice esempio di variabile a tre indici è costituito dai quadretti di un quaderno o dai caratteri di un libro. Ciascun quadretto del quaderno, o carattere del libro, è identificabile come elemento della variabile con indici a tre dimensioni: riga, colonna, pagina. Attraverso questa convenzione la variabile quadretto(i, j, k) con opportuni valori associati ai tre indici, potra' individuare con precisione un determinato quadretto.

9.3 IL CALCOLO DI UNA DISTRIBUZIONE IN VBA

Nel problema che andremo a risolvere disponiamo di un insieme di osservazioni relative alle età di un insieme di soggetti. I dati possono pertanto essere, per esempio, i numeri 0, 55, 29, 43, ... Vogliamo riepilogare questi dati suddividendoli in una distribuzione di frequenza [5]. suddivisa in classi di età di ampiezza costante (a parte l'ultima). Nella figura sottostante è riprodotta la situazione della cartella di lavoro terminata l'esecuzione del programma. Dati e codice del problema sono contenuti nella cartella `distribuzione_di_frequenza.xls`. Si nota, nella colonna A a partire dalla prima cella, la sequenza dei dati che devono essere conteggiati e, nella zona D4:E14, la distribuzione delle frequenze. La variabile con indice si riferisce a queste frequenze e pertanto sarà costituita da un insieme di 11 elementi (quante sono le classi della distribuzione). Si tenga presente che nel problema che

5 Approfondimenti su un buon libro di statistica [3]

	A	B	C	D	E	F	G
1	12						
2	31			**classi**	**frequenze**		
3	0						
4	7			0-9	3		
5	21			10-19	2		
6	105			20-29	2		
7	78			30-39	1		
8	74			40-49	3		
9	11			50-59	2		
10	0			60-69	1		
11	43			70-79	2		
12	44			80-89	2		
13	54			90-99	1		
14	89			100 e oltre	1		
15	22						
16	90						

risolviamo non è conosciuto a priori il numero di osservazioni riportate nella colonna A (nel nostro esempio si tratta di 20 osservazioni). In queste condizioni la fine dei dati è segnalata da una cella vuota. La soluzione del problema è riassunta nei seguenti punti:

1. dichiarazione delle variabili;
2. azzeramento del vettore delle frequenze (si ricordi quanto detto nel paragrafo 2.4 e quanto riportato nel listato 5.1);
3. inizializzazione dell'indice di riga utile ad individuare, una alla volta nel ciclo, le celle che contengono le età;
4. ciclo da ripetere a condizione che la cella contenga un dato;
   - calcolo dell'indice del vettore delle frequenze in cui conteggiare quella età [6];
   - controllo dell'indice ed eventuale correzione[7];
   - incremento del vettore delle frequenze;
   - incremento dell'indice di riga (per la cella dell'età);
5. terminati i dati da suddividere in classi, procediamo alla copia del vettore delle frequenze nelle celle del foglio (zona E4:E14).

Nella procedura `calcola_distribuzione_frequenza` sono scritte le istruzioni del programma. Tra le dichiarazioni notiamo quella relativa alla variabile con indici destinata a memorizzare le frequenze (`freq`). Si tratta di una variabile con un indice composta di 11 elementi.

6 Con le classi della distribuzione ad ampiezza costante si usa la formula:

$$\frac{eta - limiteinferioreprimaclasse}{ampiezzadiclasse} + 1$$

con un operatore di divisione intera (simbolo "\").

7 Se abbiamo un'età pari a 120 anni, fatto poco probabile ma non impossibile, questa punto 4b produce un indice pari a 12. L' istruzione corregge questo errore.

```
Sub calcola_distribuzione_frequenza()
        Dim classe_eta As Integer
' dichiarazione e dimensionamento della variabile con indici
        Dim freq(11) As Integer
           . . .
        For Ir = 1 To 11
                freq(Ir) = 0
                Next Ir
           . . .
```

Nel codice è visibile la parte iniziale del programma quella che procede alla dichiarazione ed al dimensionamento della variabile con indici. Tra le prime istruzioni del programma abbiamo riprodotto anche il punto 2 della soluzione.

ESERCIZI

1. Risolvere il problema trattato in questo capitolo utilizzando i comandi visti nei paragrafi 7.4 e 7.5 per il conteggio delle celle piene della zona da utilizzare in un ciclo For Next che sostituisce il Do While.

10 LE OPERAZIONI SULLE MATRICI

Nella soluzione dei problemi di finanza, si ricorrere di frequente ad operazioni su variabili con indice. Queste operazioni, disponibili in Excel sotto forma di funzioni matrice, possono essere eseguite anche con la programmazione VBA. In questo capitolo ci occuperemo di questo problema calcolando: la matrice trasposta, il prodotto di matrici e la matrice inversa.

10.1 LA MATRICE TRASPOSTA

Data una matrice $\boldsymbol{A}$ di n righe e m colonne, la trasposta di $\boldsymbol{A}$, che indicheremo con $\boldsymbol{A}^T$, è una matrice in cui ciascun elemento si ottiene dal corrispondente elemento di $\boldsymbol{A}$ scambiando la posizione righe-colonne. In termini più rigorosi sarà:

$$\boldsymbol{A}^T_{ji} = \boldsymbol{A}_{ij} \quad ,\forall i,j$$

Dal punto di vista delle istruzioni VBA il calcolo è banale. Ciascun elemento della trasposta si ottiene scambiando gli indici del corrispondente elemento della matrice di partenza come visto nella formula precedente. Più rilevante è il problema del dimensionamento delle variabili con indici che ossia lo spazio di memoria necessario a memorizzarle.

Dal momento che abbiamo l'obbiettivo di utilizzare lo spazio *strettamente necessario* in relazione a qualunque problema, ricorreremo ad un dimensionamento *dinamico* delle variabili con indici[1]. Questo risultato si ottiene in VBA con la funzione `UBound` che ha due argomenti: la variabile con indici da dimensionare e l'indice su cui agire[2]. Essa restituisce il numero di elementi effettivamente allocati nella variabile per quell'indice. Se, per esempio, calcoliamo `Ubound(A,1)` intenderemo riferirci al calcolo del numero di elementi della prima dimensione (le righe nella nostra convenzione) della variabile con indici `A`. Con `Ubound(A,2)` intenderemo riferirci al calcolo del numero di elementi della seconda dimensione etc.

Nel codice sottostante sfruttiamo questa funzione per il calcolo del numero di righe e di colonne sia della matrice originale sia della trasposta. Coerentemente con questa impostazione, ad inizio sottoprogramma, la variabile con indici non è dimensionata (le parentesi tonde sono vuote).

1 Si tratta di un accorgimento che esegue il dimensionamento al momento in cui il programma è in esecuzione in base alle dimensioni effettive della variabile con indici.

2 L'indice di riga o l'indice di colonna nel caso di una variabile a 2 indici.

```
Sub mat_trasposta(matIn() As Single, matOut() As Single)
        . . .
    n = UBound(MatIn,1)
    m = UBound(MatIn,2)
    For I = 1 to n
        For J = 1 to M
            matOut(J,I) = matIn(I,J)
            Next J
        Next I
         . . .
```

Le parti del codice che qui non sono riprodotte sono quelle che assegnano il contenuto delle celle alla matrice in memoria e viceversa. L'assegnazione del contenuto delle celle alla matrice in memoria viene fatto in vista della riutilizzabilità del codice sotto forma di sottoprogramma da includere nella libreria comune. Nella figura sottostante si riproduce il foglio Excel in cui abbiamo calcolato la trasposta di una matrice data.

	A	B	C	D	E	F	G	H	I	J
1	**MATRICE TRASPOSTA**									
2										
3	MATRICE A				MATRICE A^T					
4										
5	1	3			1	3	6	5	7	
6	3	0			3	0	-9	11	12	
7	6	-9								
8	5	11								
9	7	12								
10										

10.2 IL PRODOTTO DI MATRICI

Date due matrici $\boldsymbol{A}_{n,m}$ e $\boldsymbol{B}_{m,k}$ riprodotte di seguito,

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{12} & \dots & a_{1n} \\ \vdots & \vdots & \vdots & \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \qquad \boldsymbol{B} = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1k} \\ b_{21} & b_{12} & \dots & b_{1k} \\ \vdots & \vdots & \vdots & \\ b_{n1} & b_{m2} & \dots & b_{nk} \end{bmatrix}$$

la matrice $\boldsymbol{C}_{m,k}$

$$\boldsymbol{C} = \begin{bmatrix} c_{11} & c_{12} & \dots & 1k \\ c_{21} & c_{12} & \dots & c_{1k} \\ \vdots & \vdots & \vdots & \\ c_{m1} & c_{m2} & \dots & c_{mk} \end{bmatrix}$$

prodotto della matrice $\boldsymbol{A}$ e della matrice $\boldsymbol{B}$, è una matrice di m righe (pari a quello delle righe $\boldsymbol{A}$), e di k colonne (pari a quello delle colonne di $\boldsymbol{B}$) in cui ciascun elemento $c_{i,j}$ è calcolato in base ad un metodo chiamato righe colonne perché ottenuto tramite la moltiplicazione di elementi di una riga con quelli di una colonna.

Più precisamente, con gli elementi delle matrici di cui sopra, la prima riga della matrice $\boldsymbol{C}$ si ottiene come mostrato di seguito:

$$c_{11} = \sum_{i=1}^{n} a_{1i} * b_{i1} \qquad c_{12} = \sum_{i=1}^{n} a_{1i} * b_{i2} \qquad \dots$$

$$\dots \qquad c_{1k} = \sum_{i=1}^{n} a_{1i} * b_{ik}$$

Prima di procedere è utile sviluppare due importanti considerazioni:

- la matrice prodotto di due matrici è ottenibile a patto che il numero di colonne della prima sia pari al numero di righe della seconda;
- nelle operazioni con le matrici non è applicabile la proprietà commutativa del prodotto a meno che una matrice sia l'inversa dell'altra.

Nello stesso foglio si trovano i dati e le istruzioni VBA che calcolano il prodotto di due matrici. Si nota nel programma chiamante, il dimensionamento dinamico (quello che si effettua al di fuori del comando `Dim`).

```
. . .
ReDim matA(2,3)
ReDim matB(3,4)
. . .
```

10.3 LA MATRICE INVERSA

Data una matrice $\boldsymbol{A}_{n,n}$ si definisce come inversa di $\boldsymbol{A}$ e si scrive $\boldsymbol{A}^{-1}$, quella matrice per cui valgono le seguenti uguaglianze:

$$\boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{I}$$

in cui la matrice $\boldsymbol{I}_{n,n}$, definita come la matrice identità, è data da:

$$\boldsymbol{I} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

ovvero una matrice che ha tutti gli elementi della diagonale principale pari a 1 e i restanti elementi uguali a 0.

Normalmente il calcolo dell'inversa presenta una certa difficoltà. Fortunatamente è facile trovare in Internet una versione di questo programma liberamente disponibile. La versione che abbiamo reperito si deve a Franco Languasco (☞ www.flanguasco.org) a cui va il nostro ringraziamento.

10.4 INSERIRE LE SUB NELLA LIBRERIA COMUNE

Le sub `mat_prodotto`, `mat_trasposta`, `mat_inversa`, come anticipato nel paragrafo 10.1, sono scritte in modo tale da poter essere utilizzate per qualunque dimensione del problema. Queste dunque, a pieno titolo, possono far parte della libreria comune. Il lettore le inserisca in un apposito modulo di questa libreria. Il modulo che abbiamo definito noi a questo scopo si chiama `Modulo_matrici`.

ESERCIZI

1. Scrivere il programma che calcola la matrice trasposta a partire da una matrice di 4 righe e 3 colonne. La matrice originale si trova già nel foglio in una posizione nota a priori. La trasposta deve essere riprodotta nel foglio in una posizone a scelta.

2. Scrivere il programma che calcola il prodotto di due matrici di dimensioni opportune (diverse da quelle trattate nel capitolo). Le matrici originali si trova già nel foglio in una posizione nota a priori. La matrice prodotto essere scritta in una posizone a scelta.

3. Cercare su Internet un esempio relativo al calcolo dell'inversa di una matrice. Riportare la matrice di partenza nel foglio e calcolare la sua inversa modificando opportunamente il programma trattato in questo capitolo.

11

IL PIANO DI AMMORTAMENTO A RATE COSTANTI

Un piano di ammortamento è una tabella nella quale viene rappresentata la situazione relativa al rimborso rateale del prestito di un capitale ad ogni scadenza. Nella tabella la somma da rimborsare è ripartita in rata, quote interessi, debito residuo ed estinto. Nel caso che stiamo per trattare, il rimborso è a rate costanti posticipate ed è chiamato ammortamento *francese*[1]. Un esempio piano di ammortamento francese è visibile nella figura sottostante. Nella cartella lavoro

RATA	QUOTA INT.	QUOTA CAPITALE	DEBITO ESTINTO	DEBITO RESIDUO
2310	500	1810	1810	8190
2310	410	1900	3710	6290
2310	315	1995	5705	4295
2310	215	2095	7800	2200
2310	110	2200	10000	0

`ammortamento_costante.xls` si trovano le istruzioni del programma. Si tratta di una soluzione generalizzata a questo problema. Viene richiesto all'utente di inserire, con altrettante `InputBox`, tre dati: il capitale da restituire, il numero di rate ed il tasso di interesse. Nella stessa cartella di lavoro è disponibile un'altra procedura che si occupa di ritornare ad un foglio di lavoro vuoto[2]. Il programma è articolato nei seguenti passaggi:

1. un ciclo gestisce l'acquisizione, sotto forma di input da parte dell'utente, di capitale, tasso di interesse e numero di rate; questi dati vengono inseriti nelle celle B1, B2, B3 insieme alle corrispondenti intestazioni
2. si scrivono in altrettante celle le intestazioni delle colonne del piano;
3. si imposta la proprietà larghezza delle colonne da B a G che dovrà essere pari a 12 (la larghezza standard è di 8,43 caratteri);
4. la zona che conterrà le testate di colonna (C6:G7) dovrà avere ridefinite le proprietà relative a tipo e allineamento del carattere (grassetto e centrato);
5. calcolo della numero dell'ultima riga del piano. Essa è in funzione del numero di rate. In base a questa riga, sarà possibile

1 Per approfondimenti su questi argomenti si veda [4].

2 Se infatti il calcolo di un nuovo piano comporta un numero di rate inferiore a quello calcolato in precedenza, si crea una sovrapposizione tra il piano nuovo e quello vecchio.

definire la zona che conterrà tutti i dati del del piano e la zona in cui copiare le formule della prima riga;

6. definizione, per tutte le celle del piano, del numero di cifre decimali (pari a zero);
7. scrittura delle formule necessarie al calcolo del piano: valore attuale di una rendita..., rata; impostazione di debito residuo e debito estinto iniziali (rispettivamente pari a 0 ed al capitale); scrittura delle formule che si trovano nella prima riga del piano;
8. copia delle formule;
9. abbellimenti finali costituiti dal nascondere la riga 9 del foglio (contiene i valori iniziali del debito estinto e residuo) e dall'eliminazione della griglia.

La traduzione in istruzioni dei punti da 1 a 2 non riveste particolare importanza. Meritano un approfondimento invece la definizione delle proprietà delle testate delle colonne interessate dal prospetto (punto 3) e le proprietà della zona in cui verrà scritto il piano (punto 4). La parte di codice che si occupa di questo è riportata di seguito.

```
Range("B:G").Columns.ColumnWidth = 12
With Range("C6:G7")
        .Font.Bold = True
        .HorizontalAlignment = xlCenter
        End With
```

L'istruzione della prima riga assegna un opportuno valore alla larghezza delle colonne dalla B alla G. In essa è visibile la scansione nella gerarchia degli oggetti a partire da `Range`, passando per `Columns` fino ad arrivare alla proprietà `ColumnWidth`. Successivamente sono visibili le istruzioni corrispondenti al punto 4.

Il comando `With` serve definire uno o più oggetti di una gerarchia che debbono essere riferiti nelle istruzioni successive. La sua validità si estende a tutte le righe successive fino al prossimo `End With` ed evita di riscrivere il riferimento all'oggetto `Range("C6:G7")`. Riportiamo di sotto il codice equivalente[3].

```
Range("C6:G7).Font.Bold = True
Range("C6:G7).HorizontalAlignment = xlCenter
```

La parte restante del programma si occupa della scrittura delle formule della prima riga del piano e della loro copia nelle righe successive in funzione delle rate. I commenti, presenti in abbondanza a

3 Anche se in questo caso la convenienza nell'utilizzarlo è quantomeno dubbia, lo abbiamo voluto proporre a titolo didattico.

fianco delle corrispondenti istruzioni, dovrebbero essere sufficienti a chiarirne il significato.

ESERCIZI

1. Si chiede di scrivere il programma VBA che calcola il piano per la restituzione di un prestito con il metodo americano. Nella figura sottostante ne riproduciamo un esempio. L'ammortamento americano prevede due tassi di interesse in relazione ad altrettante operazioni previste dal piano: un tasso relativo alla remunerazione del capitale ricevuto in prestito e dovuta al creditore, l'altro, generalmente inferiore al primo, relativo all'investimento di capitale accumulato di solito presso un istituto di credito.

2000000	capitale						
0.04	tasso remunerazione						
0.03	tasso accumulazione		**capitale versato**	**fondo inizio anno**	**interessi prodotti dal fondo nell'anno**	**fondo alla fine dell'anno**	**quota interessi al creditore**
6	n.o rate			(fondo anno prec.)			
6.46840988	s figurato i'	1	309195.00	0.00	0.00	309195.00	80000.00
	(montante rendita	2	309195.00	309195.00	9275.85	627665.85	80000.00
	cost. unit. Post.)	3	309195.00	627665.85	18829.98	955690.83	80000.00
80000	quota annua interessi	4	309195.00	955690.83	28670.72	1293556.55	80000.00
	x creditore	5	309195.00	1293556.55	38806.70	1641558.25	80000.00
309195.001	quota fondo amm.to	6	309195.00	1641558.25	49246.75	2000000.00	80000.00
389195.001	somma annua versata						
	dal debitore						

La quota interessi costante (cella A13) periodicamente dovuta al creditore è data dalla relazione:

$$C * i$$

mentre la quota capitale della cella A15, anch'essa costante, è data dalla relazione:

$$\frac{C}{s_{\overline{n}|i}}$$

in cui $s_{\overline{n}|i}$, montante di una rendita unitaria posticipata immediata, vale:

$$\frac{(1+i)^n - 1}{i}$$

12 LA USER FORM

Le modalità di colloquio con l'utente viste finora, hanno utilizzato le funzioni `MsgBox` e `InputBox`. Quest'ultima, nel programma che calcola il piano di ammortamento francese, richiedeva i dati tramite un ciclo che ripeteva per tre volte la richiesta di inserimento.

Il linguaggio VBA dispone di un'altra interfaccia di dialogo che si caratterizza per la ricca e diversificata dotazione di strumenti di colloquio. Essa evita, tra l'altro, di ricorrere a noiose ripetizioni. Vediamo di cosa si tratta.

12.1 LA FINESTRA USERFORM

La finestra di dialogo UserForm è un oggetto creato in ambiente VBA. Una volta che ci troviamo nell'ambiente editor VBA, selezioniamo la sequenza del menu Inserisci → UserForm.

Al termine di questi passaggi la finestra dell'editor si presenterà come nella figura sottostante. Si nota la comparsa di due nuove fine-

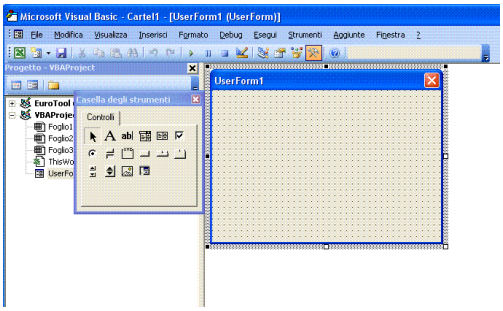


Figura 12.1: Finestra di dialogo in modalità editor

stre: una intitolata UserForm1, l'altra dal titolo Casella degli strumenti. Si tratta di due finestre fluttuanti[1]. Nella finestra del progetto notiamo la presenza di un nuovo elemento chiamato UserForm1.

La finestra Casella degli strumenti, elenca le interfacce di colloquio con l'utente che si possono inserire nella finestra che comparirà durante l'esecuzione del programma. Ne viene mostrato un esempio nella figura sottostante La finestra Casella degli strumenti elenca tutti gli strumenti di colloquio che possono essere inseriti in una finestra di dialogo. Nella tabella 12.1 alla fine del capitolo descriviamo gli strumenti standard più importanti in base all'ordine con cui compaiono nella stessa finestra. Si tenga presente che il primo di questi strumenti

1 Possono essere spostate facendo clic e mantenendo premuto il tasto sinisto del mouse.

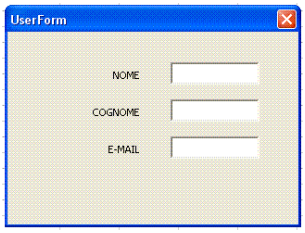


Figura 12.2: Finestra di dialogo durante l'esecuzione

Figura 12.3: La casella degli strumenti

(rappresentato da una freccia rivolta in alto a sinistra) non è compreso in questo elenco. La sua funzione sarà spiegata più avanti.

12.2 L'INSERIMENTO DEI CONTROLLI

Ora che conosciamo i controlli di cui disponiamo, vediamo come si procede al loro inserimento nella finestra di dialogo. Per facilitare il nostro lavoro abbiamo già disegnato la finestra di dialogo che vogliamo comporre e che vediamo qui sotto[2]. Dall'alto verso il basso

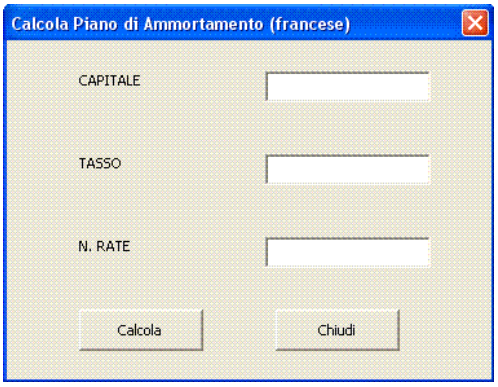


Figura 12.4: La finestra di dialogo per il Piano di ammortamento

notiamo: il titolo della finestra, tre caselle di testo e le corrispondenti etichette sulla sinistra, due pulsanti di comando con i relativi titoli.

Le operazioni da svolgere per ottenere una finestra simile possono riepilogarsi nei seguenti passaggi (se non abbiamo eseguito altre operazioni, sul computer dovremmo avere ancora la finestra intitolata UserForm1 ossia quella della figura 12.1):

2 Una volta padroneggiata la tecnica di inserimento e la conoscenza degli strumenti da inserire, si consiglia di fare un disegno su carta della finestra di dialogo che vogliamo realizzare. Quando il disegno su carta ci sembra soddisfacente, possiamo procedere alla sua effettiva realizzazione.

1. fare clic sullo strumento scelto nella casella degli strumenti (decidiamo di inserire la prima etichetta, quella intitolata CAPITALE);

2. spostarsi nella finestra intitolata UserForm1 (il puntatore cambia aspetto assumendo la forma di una piccola croce cui si affianca l'icona dello strumento prescelto);

3. portarsi nella posizione corrispondente ad uno dei quattro vertici del rettangolo che vogliamo disegnare fare clic e, mantenendo premuto il tasto sinistro, allargare il rettangolo fino a raggiungere approssimativamente (vedi punto 4) la dimensione voluta;

4. rilasciare il tasto del mouse senza preoccuparsi se le dimensioni o la posizione che abbiamo impostato siano giuste o meno. Di fianco allo strumento appena inserito si nota la comparsa del titolo Label1 e di otto quadrati sulla cornice del controllo. Questi segnalano che si tratta di un contorno ridimensionabile nonché di un elemento fluttuante.

Fatto questo, ripartendo dal punto 1 delle spiegazioni precedenti, possiamo passare all'immissione degli altri elementi che compongono la finestra. Per ora non ci preoccupiamo dei nomi che vediamo comparire, delle dimensioni dei vari oggetti nonchè delle loro posizioni.

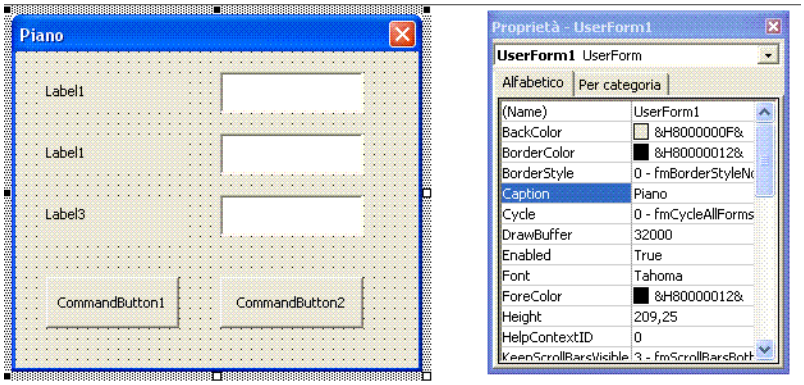
Selezioniamo l'oggetto e modifichiamo le sue dimensioni agendo su uno dei quadrati neri che appaiono sui lati. Consigliamo di fare pratica su questa parte in modo da velocizzare le proprie capacità di realizzare interfacce utente più o meno complesse.

Dopo aver inserito un oggetto è possibile, tramite le consuete procedure di copia/incolla, effettuarne la duplicazione. Questo risulta particolarmente utile soprattutto quando si devono inserire strumenti simili tra loro. Eseguita la copia è possibile spostare l'oggetto duplicato nel punto che riteniamo opportuno con le funzionalità di trascinamento disponibili con il mouse.

12.3 LA MODIFICA DELLE PROPRIETÁ

Dopo la fase di disegno del controllo si può passare alla operazioni che riguardano la modifica di alcune delle proprietà. La prima e più semplice tra queste è costituita dal nome che, per quanto ci riguarda, interessa gli oggetti UserForm e Label. Per effettuare questa operazione dobbiamo aver attivato la finestra delle proprietà con i comandi del menu Visualizza → Finestra Proprietà. La prima operazione riguarda la modifica di due proprietà dell'oggetto UserForm1 costituite da:

- titolo della finestra;
- nome dell'oggetto.



Selezionato l'oggetto UserForm facendo click, lo vediamo evidenziato con una cornice di otto quadratini. Adesso possiamo modificare le due proprietà suddette considerando nella finestra Proprietà (visibile in figura a destra) le righe Caption e (Name). Sostituiano in corrispondenza della seconda colonna di queste:

- UserForm1 con Calcola Piano .... (o quello che riteniamo più idoneo come titolo della finestra);
- ufPianoDiAmmortamento come nome dell'oggetto nella seconda colonna della riga (Name).[3]

Si raccomanda di prestare attenzione al nome dell'oggetto perché questo sarà importante in seguito. Ripetiamo questa procedura per i tre oggetti etichette cambiando la relativa proprietà da Label1, Label1, Label3 rispettivamente in CAPITALE, TASSO, N.RATE con i passaggi:

1. selezionare l'oggetto nella finestra di dialogo;
2. spostarsi nella finestra delle proprietà e modificare il titolo dell'etichetta che si trova nella seconda colonna in corrispondenza della riga intitolata Caption.

Lo stesso procedimento va seguito per i due pulsanti di comando. In questo caso oltre a modificare il loro titolo rispettivamente in Calcola e Chiudi cambiamo anche il nome in btnCalcola e btnChiudi.

Le motivazioni di quanto appena fatto diventano evidenti osservando cosa accade se si eseguono i comandi del menu Visualizza–> Codice (la finestra Progetto dovrebbe trovarsi ancora su ufPianoDiAmmortamento). In questa finestra si notano due elenchi a discesa. In quello di sinistra sono presenti gli oggetti inseriti nella finestra di dialogo secondo l'ordine alfabetico. Nell'elenco a discesa di destra sono mostrati gli eventi disponibili per quell'oggetto (si veda la prossima figura).

3 La scelta di questi nomi è dovuta a motivazioni mnemoniche. Noi abbiamo usato una convenzione che nomina gli oggetti dello stesso tipo con un prefisso identico seguito da un opportuno nome mnemonico e così sarà nel seguito. Per fare un esempio abbiamo scelto btnNome per i pulsanti di comando, ufNome per le finestre di dialogo, txtNome per le caselle di testo.

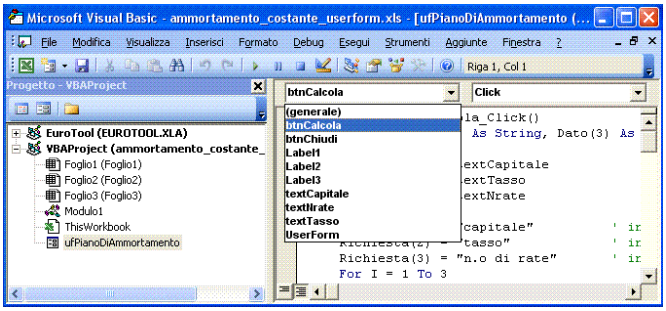


Figura 12.5: Gli oggetti della UserForm

Selezionato un elemento da questa lista, la finestra del codice mostrerà le istruzioni di inizio/fine della procedura associata a quell'evento. Se ancora non è stato scritta nessuna istruzione del programma, questa finestra, a differenza di quanto apare nella figura, conterrà soltanto la dichiarazione della procedura con il nome dell'oggetto e dell'evento collegati costituita, nel nostro caso, da btnCalcola_Click.

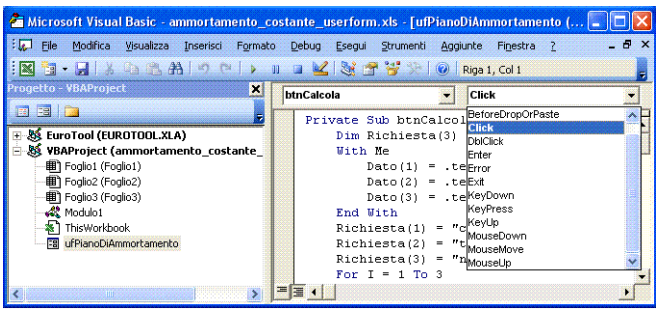


Figura 12.6: ... e gli eventi associati

12.4 L'ORDINE DI TABULAZONE

La modifica di controlli della finestra di dialogo, può portare a situazioni in cui durante l'esecuzione del programma, al momento in cui viene visualizzata la finestra di dialogo, il tasto di di tabulazione non segue l'ordine sequenziale con cui i controlli sono visibili nella finestra (in presenza di numerosi campi dove sarà finito il cursore lampeggiante?).

Per fare un esempio relativo ai controlli di cui sopra, potrebbe accadere di aver inserito la casella di testo che contiene il numero delle rate capitale in un momento successivo a quello in cui è stata inserita, p.e., la casella del tasso. Se così fosse, durante l'esecuzione del programma l'uso del tasto di tabulazione provocherebbe un salto al campo del numero di rate invece che a quello del tasso. La modifica di questo sgradevole comportamento, si ottiene ricorrendo alla sequenza di comandi del menù Visualizza → Visualizza Ordine di Tabulazione che visualizza un'apposita finestra. Si nota la presenza dei tasti Spo-

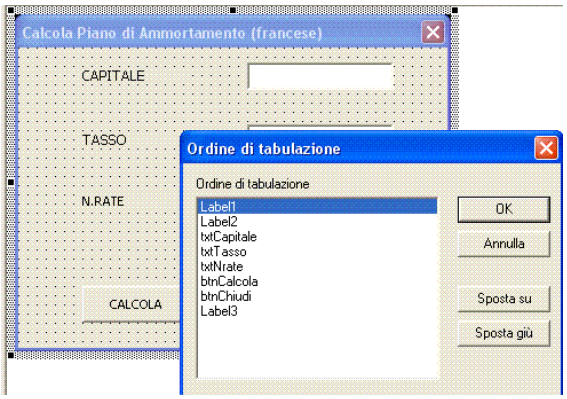


Figura 12.7: La modifica dell'ordine di tabulazione

sta su e Sposta giu che permettono di stabilire l'ordine ritenuto più idoneo.

12.5 LA PROGRAMMAZIONE DELLA FINESTRA DI DIALOGO

Nella modalità di funzionamento vista finora, l'esecuzione del programmma procede in base ad una logica che protremmo definire sequenziale. Le istruzioni sono eseguite una dopo l'altra dall'alto verso il basso con eventuali salti in presenza di comandi condizionali o di ciclo. La programmazione con le finestra di dialogo modifica questa logica a causa della presenza dei pulsanti di comando. Il loro impiego nel programma rivoluziona completamente la logica di esecuzione che da sequenziale diventa una esecuzione *ad eventi*. Questo si traduce in una modifica della soluzione del problema così come presentata nel capitolo precedente.

Tutto ciò premesso, ridefiniamo le funzionalità del programma che calcola il piano di ammortamento. L'adozione della finestra di dialogo personalizzata comporta una diversa organizzazione del codice che si baserà sul verificarsi dell'evento click su uno dei due pulsanti CALCOLA e CHIUDI. Il programma sarà composto da due sole procedure: una di visualizzazione della finestra di dialogo, l'altra di ripristino del foglio. Queste parte del Modulo1 che è stato inserito prima di procedere alla scrittura delle istruzioni che le compongono (si veda la finestra del progetto nelle figure precedenti).

```
Sub ammortamento_francese()
    ufPianoDiAmmortamento.Show
End Sub
Sub  RipristinaFoglio ()
    Dim Ciccio As String
          ....
End Sub
```

La procedura `ammortamento_francese` consiste del comando `Show` applicato all'oggetto `ufPianoDiAmmortamento`. Il comando esegue la visualizzazione della finestra di dialogo.

La procedura RipristinaFoglio non ha bisogno di alcun commento essendo identica a quella già vista nel capitolo precedente.

Entrambe le procedure fanno parte di un nuovo modulo, chiamato `Modulo1`, inserito con i comandi del menu Inserisci → Modulo.

Per quanto attiene al codice associato agli eventi, pur avendone accennato in precedenza, considerata l'importanza di questo passaggio, ripetiamo la sequenza delle operazioni:

1. selezione dell'oggetto `ufPianoDiAmmortamento` nella finestra del progetto;
2. apertura della finestra del codice con i comandi del menu Visualizza → Codice;
3. selezione del pulsante `btnCalcola` nella casella a discesa di sinistra e dell'evento `Click` in quella a destra.

Possiamo ora scrivere le istruzioni da eseguire al verificarsi dell'evento costituito dal clic sul tasto CALCOLA. Si tratta di una parziale modifica delle istruzioni scritte nel capitolo precedente che quindi possono essere copiate.

```
Private Sub btnCalcola_Click()
    Dim Richiesta(3) As String, Dato(3) As String
    Dato(1) = txtCapitale.Text
    Dato(2) = txtTasso.Text
    Dato(3) = txtNrate.Text
      . . .
```

Rammentando che ci troviamo al momento in cui si verifica l'evento Click sul pulsante CALCOLA possiamo dire che le istruzioni mostrate assegnano il valore della proprietà Text dei corrispondenti controlli ad altrettanti elementi di una variabile con indici (utile per generalizzare la istruzioni successive). Da questo punto in poi le istruzioni ricalcano grossomodo quelle già scritte nel programma del capitolo precedente.

```
    Richiesta(1) = "capitale"          ' in B1
    Richiesta(2) = "tasso"             ' in B2
    Richiesta(3) = "n.o di rate"       ' in B3
    For I = 1 To 3
        If IsNumeric(Dato(I)) = False Then
            MsgBox prompt:="Il " & Richiesta(I) & " manca o non e'
                numerico", _
                                Buttons:=vbCritical, _
                                Title:="ERRORE NEI DATI"

            Exit Sub
            End If
    Unload Me
    For I = 1 To 3                          ' si procede con inserimento/
        calcoli
        Cells(I, 2).Value = _
            CSng(Replace(Dato(I), ".", ","))' scrive i tre dati acquisiti
                 in B1, B2, B3
        Cells(I, 3).Value = Richiesta(I)    ' scrive le fincature in A1,
            A2, A3
        Next I
    Cells(4, 3).Value = " v.a. rendita unitaria annua posticipata temporanea"
     .....
End Sub
```

Dopo aver assegnato alla variabile con indici `Richiesta` tre costanti, controlliamo che i dati inseriti siano congrui (presenza del dato e controllo che si tratti di un numero).

Si nota, dopo il primo ciclo `For` il comando `Unload Me` necessario a scaricare il form dopo aver effettuato i controlli sui dati. A questo proposito si tenga presente che `Me` si riferisce al nome dell'UserForm in esecuzione in quel momento. Ultimo, ma necessario, il codice corrispondente all'evento Click associato al pulsante `CHIUDI`.

```
Private Sub btnChiudi_Click()
    ufPianoDiAmmortamento.Hide
End Sub
```

ESERCIZI

1. Tenendo presente il codice del paragrafo 4.5, scrivere il programma che visualizza una UserForm che richiede i dati necessari al calcolo delle radici di un'equazione di secondo grado. Eventualmente riutilizzare, con le opportune modifiche, l'interfaccia di dialogo impiegata nel calcolo del piano di ammortamento francese.

Tabella 12.1: Gli elementi della Casella degli strumenti

Etichetta/Label	Si utilizza per immettere un commento a strumenti di interfaccia. Si pensi, per esempio, ai commenti inseriti vicino ad una Casella di testo.
Casella di testo/TextBox	Crea un campo in cui è possibile immettere testo, numeri etc.
Cornice/Frame	La funzione di questo strumento è quella di fungere da contenitore di altri controlli da mettere insieme perché logicamente collegati tra loro. I controlli che si trovano al suo interno vanno creati dopo di esso.
Puls. di comando/CommandButton	Crea pulsanti che attivano determinate azioni quando l'utente fa clic su uno di essi.
Casella di controllo/CheckBox	Si tratta del quadrato che contiene un segno di spunta in caso di selezione. Quando in una finestra si trova più di una di queste caselle, si possono effettuare selezioni multiple ovvero le selezioni non sono mutuamente esclusive.
Pulsante di opzione/OptionButton	É una casella tonda che contiene un punto nero in caso di selezione. La presenza in contemporanea con altri pulsanti dello stesso tipo, inseriti in un controllo Cornice, comporta che essi siano mutuamente esclusivi.
Pulsante interruttore/ToggleButton	Disegna un pulsante che appare premuto/rilasciato.
Casella di riepilogo/ListBox	Disegna un elenco a discesa composto da diversi valori su cui effettuare una scelta. Selezionando la proprietà MultiSelect permette scelte multiple.
Casella combinata/ComboBox	Combinazione di casella di testo e casella di riepilogo. Nella casella di testo si possono inserire dati da una lista a discesa oppure liberamente scelti dall'utente. Per obbligare la scelta alla sola lista a discesa, bisogna modificare la proprietà Style.

13 IL GENERATORE DI NUMERI CASUALI NORMALI

Nella soluzione di alcuni problemi di finanza e in particolare nella simulazione con il metodo di Monte Carlo, è necessario generare una sequenza di numeri casuali distribuiti uniformemente per ottenere quindi una sequenza di numeri casuali distribuiti normalmente. In questo capitolo verrà proposta una soluzione accennando brevemente ad alcune questioni correlate.

13.1 DEFINIZIONI ED ESEMPI

Un insieme di numeri casuali deve rispettare le seguenti proprietà:

- ciascun numero della sequenza deve essere distribuito uniformemente in un determinato intervallo;
- tutti i numeri devono essere indipendenti tra loro.

Nel mondo reale è facile trovare esempi che generano sequenze di numeri casuali, si pensi all'estrazione di un numero del gioco del lotto con rimpiazzo ossia la reintroduzione nell'urna del numero estratto. Quando invece, per necessità, dobbiamo simulare lo stesso procedimento con un sistema di elaborazione, le cose si fanno un poco più complicate. I numeri così ottenuti, detti per questo *pseudocasuali*, presentano diversi problemi tra i quali i più rilevanti sono la mancanza di casualità e la tendenza, dopo un certo numero di estrazioni, a ripetere la sequenza (periodo).

13.2 IL GENERATORE DI MERSENNE-TWISTER

Alla fine degli anni '90 due studiosi giapponesi, Matsumoto e Mishimura, idearono un algoritmo utilizzabile con un sistema di elaborazione che minimizza le criticità appena accennate valido per gran parte delle applicazioni in cui si richiede un generatore di numeri casuali[1].

La versione VBA di questo algoritmo, che abbiamo reperito su Internet, si deve a Pablo Mariano Ronchi, cui va il nostro ringraziamento. Per quanto ci riguarda è stata inserita come sottoprogramma nella libreria comune in `Modulo_varie` e collegata con le modalità illustrate nel paragrafo 8.3.

1 per una più approfondita discussione di questi aspetti, assieme ad altri documenti reperibili in rete, si consulti l'articolo:
http://www.antoniogrande.uniroma1.it/VBA/the_twisted_road.pdf, ottima sintesi di brevità e chiarezza espositiva

Il sottoprogramma, per poter funzionare, richiede un "seme", nonchè la scelta di una procedura in relazione alla variabilità del numero casuale che si vuole generare.

Per quanto riguarda il seme si tratta di un numero compreso nell'intervallo -2147483648, 2147483647. Per ciascuno dei numeri interi compresi in questo intervallo viene generata una sequenza diversa di numeri casuali ovvero, a partire da uno stesso seme, la sequenza generata è sempre identica. La riproduzione della stessa sequenza di numeri è importante quando si debbano controllare i calcoli a partire da numeri così generati.

La variabilità dei valori del generatore si riferisce al valore massimo/minimo del numero prodotto dal generatore. Così, per esempio, la procedura `genrand_int32()` genera una sequenza di numeri compresi tra 0 e $232-1$, la procedura `genrand_real3b()` genera una sequenza di numeri compresi nell'intervallo 0.0 e 1.0 estremi esclusi.

13.3 LA TRASFORMAZIONE DI MARSAGLIA-BRAY

La trasformazione di Marsaglia-Bray è una variante di quella di Box-Muller. Queste due trasformazioni, risalenti agli anni '60, sono impiegate per generare una sequenza di numeri casuali normalmente distribuiti a partire da una sequenza di numeri casuali uniformemente distribuiti. La preferenza generalmente accordata alla prima si spiega con il fatto che la trasformazione Box-Muller richiede il calcolo di seno e coseno per ogni numero casuale che viene generato il che comporta un allungamento dei tempi di esecuzione. Per quanto con i sistemi di elaborazione attuali questo problema possa essere di poca rilevanza, nella nostra soluzione abbiamo scelto Marsaglia-Bray.

Questo algoritmo, a partire da coppie di numeri casuali u, v, appartenenti al "disco unitario ", calcola una coppia di numeri casuali distribuiti normalmente con le formule:

$$x = u \quad \sqrt{\frac{-2\ln R}{R}}$$

$$y = v \quad \sqrt{\frac{-2\ln R}{R}}$$

in cui

$$\left\{(u, v) \in R : u^2 + v^2 < 1\right\}$$

Vengono perciò ammesse le coppie per cui la cui somma dei quadrati sia inferiore a 1. Questa proprietà garantisce che si tratti di numeri, distribuiti uniformemente, che appartengono all'insieme disco unitario centrato nell'origine.

Nella cartella `generatore_ncasuali_normali.xls` si trova il codice

che descriviamo per sommi capi. A partire dalla richiesta di quanti numeri generare, si produce una sequenza di numeri casuali uniformemente distribuiti (algoritmo di Mersenne-Twister) che vengono trasformati in numeri distribuiti normalmente (con Marsaglia-Bray). Allo scopo di provare che la sequenza dei numeri si distribuisce in modo normale, abbiamo conteggiato questi numeri in una distribuzione di frequenza suddivisa in classi. Ad ulteriore dimostrazione di questo, oltreché a scopi didattici VBA, rappresentiamo graficamente la distribuzione con un istogramma.

Il generatore dei numeri e le istruzioni di calcolo della distribuzione, l'istogramma, la cancellazione dell'istogramma e la cancellazione dei dati eventualmente presenti nel foglio sono suddivisi rispettivamente in quattro procedure tutte contenute nello stesso modulo e facilmente riconoscibili. Non sono state impiegate finestre di dialogo ma si devono inserire due dati in altrettante celle del foglio: quanti numeri si dovranno generare (cella D1) e in quante classi suddividere la distribuzione (cella D2). Le classi hanno ampiezza pari a 0,1. La mancanza dei dati necessari ai calcoli viene controllata e, se del caso, fatta rilevare con un apposito messaggio.

Nella parte iniziale del programma vediamo le istruzioni necessarie al generatore. Si tratta di due istruzioni, corrispondenti ad altrettante procedure che fanno parte del pacchetto di Mersenne-Twister e Ronchi.

```
        . . .
    init_genrand 6                              ' inizializza il seme
    For I = 1 To N
        . . .
' con la chiamata a questa procedura generiamo numeri compresi nell'
    intervallo -1, +1
          rand1 = genrand_real4b()
          rand2 = genrand_real4b()
' poi calcoliamo il valore da sottoporre al test di accettazione
          s1 = rand1 ^ 2 + rand2 ^ 2
          If s1 <= 1 Then
              Ok = False
              s2 = Sqr(-2 * Log(s1) / s1)
' due numeri casuali distribuiti normalmente
              ncas(1) = rand1 * s2
              ncas(2) = rand2 * s2
                      . . .
'
' se l'indice "sfora" l'intervallo si genera un errore.
' Per evitarlo dobbiamo cautelarci quando si verifica tale evento.
'
                  If Indice > 50 Then Indice = 50
                  If Indice < -50 Then Indice = -50
                  Indice = Punta(Indice)
                  Freq(Indice) = Freq(Indice) + 1
                  Next K
              End If
```

Nelle figura vediamo lo stato del foglio dopo l'esecuzione del programma di estrazione. Oltre a produrre la distribuzione il programma scrive in due celle, rispettivamente in D4 e D5, il minimo ed il massimo tra tutti i numeri generati (solo a fini di controllo). La distribuzione, come si vede dalla figura, occupa le prime due colonne del foglio. Per velocizzare l'esecuzione si individua la classe cui appartiene ciascun numero con la tecnica del puntatore. Nella figura è visibile la relazione che sta alla base dei puntatori: il numero da classificare, che sappiamo essere compreso tra -50 e 50, viene posto in corrispondenza biunivoca con un altro numero che rappresenta l'indice cui della classe di appartenenza. Le istruzioni che eseguono queste funzionalità sono riprodotte nel codice seguente:

```
    ...
Indice = Int(ncas(K) / 0.1)
Indice = Punta(Indice)
Freq(Indice) = Freq(Indice) + 1
    ...
```

La prima istruzione trasforma il numero casuale decimale in un intero e lo assegna alla variabile `Indice`. Nella seconda istruzione questa variabile è l'indice del vettore `Punta` (si veda nella seconda colonna della figura i suoi elementi) che viene assegnato nuovamente ad `Indice`. Il gioco è fatto. La terza istruzione incrementa il vettore della distribuzione di frequenza.

Supponiamo, per esempio, che ncas(K) valga -4,7545. La prima istruzione assegna -47 ad Indice, la seconda gli assegna -35 (tutti i valori del vettore compresi tra -50 e -35 valgono -35). Nella terza istruzione l'elemento del vettore delle frequenze con indice -35 incrementa di uno il suo precedente valore.

L'assegnazione al vettore `Punta` dei valori che realizzano questa funzionalità viene ottenuta con due cicli. Nel primo si inizializzano gli elementi che si trovano nelle "code" (con l'esempio di sopra i valore compresi tra -50 e -35 e quelli compresi tra 35 e 50). Nel secondo ciclo si assegnano i valori rimanenti. Queste istruzioni si trovano all'inizio del programma e sono riportate di seguito.

```
For Indice = −50 To −nClassi
    Punta(Indice) = −nClassi
    Punta(Abs(Indice)) = nClassi
    Next Indice
For Indice = −nClassi To nClassi
    Punta(Indice) = Indice
    Next Indice
```

13.4 IL GRAFICO DELLE FREQUENZE

La creazione di un grafico in ambiente VBA è un argomento che richiederebbe una trattazione a parte data la notevole quantità di nozioni correlate nonché le molteplici tecniche disonibili. In questa parte ci limiteremo ad una trattazione elementare dei concetti relativi al grafico che vogliamo realizzare rimandando ai prossimi capitoli per ulteriori esempi.

La prima scelta da fare è quella relativa alla posizione del grafico ovvero se si tratta di un grafico interno ad un foglio o in un foglio a sé stante. La seconda attiene al tipo di grafico ovvero se vogliamo un grafico a barre, a punti, a torta etc. Le altre scelte relative a scala dei valori, legende, stile dei caratteri delle legende e del titolo sono abbastanza simili tra loro indipendentemente dalla posizionie e dal tipo. Nel nostro caso abbiamo deciso di disegnare un grafico in un foglio a parte ed inoltre abbiamo scelto un istogramma.

Una volta controllata la presenza dei dati necessari, calcoliamo l'estensione della zona con i dati del grafico. A questo punto passiamo alle istruzioni relative al grafico.

```
' aggiunge un oggetto Grafico (non appartenente ad alcun foglio)
Charts.Add
   With ActiveChart
' grafico tipo istogramma
      .ChartType = xlColumnClustered
' elimina le serie di dati eventualmente presenti
      Do Until . SeriesCollection .Count = 0
          . SeriesCollection (1).Delete
         Loop
```

Le prime istruzioni aggiungono un oggetto di tipo grafico, definiscono questo oggetto con il comando `With` per non ripeterlo nelle prossime istruzioni e gli assegnano il tipo con il comando `ChartType`[2].

Adesso bisogna definire le serie di dati che compongono il grafico, il nome del foglio ed il suo titolo. Ricordiamo che la zona con la serie dei dati è a dimensioni variabili il suo riferimento è stato calcolato in precedenza sotto forma di stringa.

```
                    . . .
                    ' definisce le serie del grafico
With . SeriesCollection .NewSeries
                     ' zona legenda ascisse
            .XValues = Worksheets("Foglio1").Range("A2:A" & ciccio)
                     ' zona con i dati
```

2 Una elencazione completa di tutti i tipi di grafico (Istogramma non in Pila, Istogramma in Pila, Barre, Linee, ..., Coni, Piramidi etc.) è piuttosto lunga e complessa anche perché nel comando il tipo di grafico deve essere specificato con il nome inglese. Per un approfondimento di questo problema si veda l'appendice A

```
        .Values = Worksheets("Foglio1").Range("B2:B" & ciccio)
                  ' nome della serie
        .Name = "frequenze"
        End With
                   ' propedeutico all'inserimento del nome dell'
                      oggetto Grafico
    .HasTitle = True
                  ' nome dell'oggetto grafico
    .Name = "GraficoMio"
                 ' titolo
    .ChartTitle.Characters.Text = "TitoloDelGrafico"
                 . . .
```

Come al solito abbiamo pensato di definire anche le funzionalità di cancellazione del grafico qualora si volesse procedere ad una pulizia della cartella. In questo caso abbiamo sfruttato la proprietà Count dell'oggetto Charts della cartella di lavoro.

```
Sub CancellaIstogramma
  If Application.ActiveWorkbook.Charts.Count > 0 Then Charts.Delete
End Sub
```

Nella figura sottostante abbiamo riprodotto il grafico che mostra la distribuzione dei numeri casuali normali. Ad un esame visivo si apprezza la bontà della distribuzione chiaramente normale. Si nota, al termine delle code, un rialzamento della distribuzione dovuto alla frequenze di numeri esterni alle classi.

	A	B	C	D	E
1	intervallo	frequenza	n. ripet.	250000	
2	-3,5	161	n. classi	35	
3	-3,4	66			
4	-3,3	101	ncasMin	-4,7545	
5	-3,2	134	ncasMax	4,492844	
6	-3,1	189			
7	-3	253			
8	-2,9	356			
9	-2,8	464			
10	-2,7	619			
11	-2,6	757			
12	-2,5	971			
13	-2,4	1251			
14	-2,3	1576			
15	-2,2	1999			
16	-2,1	2453			
17	-2	2859			
18	-1,9	3654			
19	-1,8	4378			
20	-1,7	5050			

indice	vettore
-50	-35
-49	-35
…	…
-35	-35
-34	-34
-33	-33
…	…
-1	-1
0	0
1	1
…	…
35	35
36	35
…	…
50	35

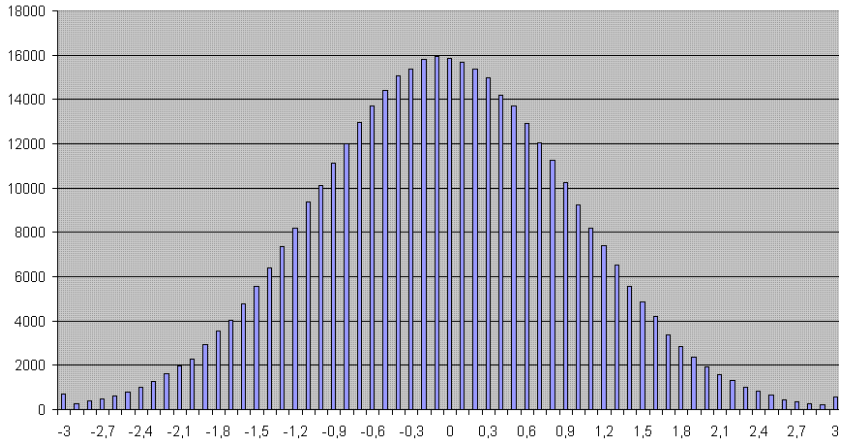


Figura 13.1: Il grafico delle frequenze

14 LA FORMULA DI BLACK SCHOLES

La formula di Black Scholes riveste una importanza fondamentale nei problemi di calcolo delle opzioni. Il suo calcolo in VBA non presenta grossi vantaggi rispetto ad Excel. In questo capitolo, a titolo puramente esemplificativo, la proporremo nella versione Excel ed in quella VBA.

La formula per quanto riguarda il prezzo di un'opzione call di tipo europeo in un mercato perfetto è data da:

$$C_t = S_t N(d_1) - X e^{-\delta(T-t)} N(d_2)$$

$$d_1 = \frac{\ln\left(\frac{S_t}{X}\right) + \left(\delta + \frac{\sigma^2}{2}\right)(T-t)}{\sigma\sqrt{T-t}}$$

$$d_2 = d_1 - \sigma\sqrt{T-t}$$

tenendo presente che:

- C_t è il prezzo della call,
- S_t è il prezzo del titolo o sottostante,
- σ è la volatilità del sottostante,
- X è il prezzo d'esercizio,
- δ è l'intensità istantanea di interesse,
- T è la scadenza (data di esercizio dell'opzione),
- t è l'epoca della valutazione,
- $N(.)$ è il valore della distribuzione normale standardizzata.

Nella formula le quantità d1 e d2 sono date da:

$$d_1 = \frac{\ln\left(\frac{S_t}{X}\right) + \left(\delta + \frac{\sigma^2}{2}\right)(T-t)}{\sigma\sqrt{T-t}}$$

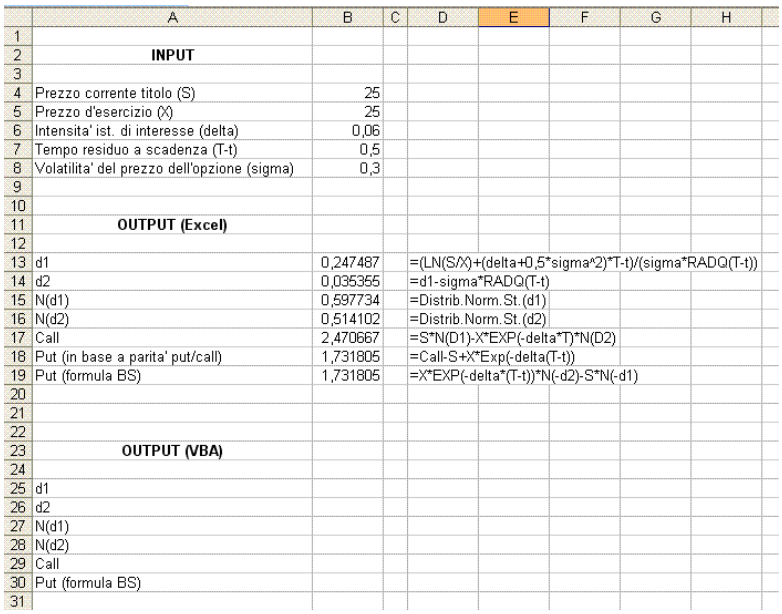
$$d_2 = d_1 - \sigma\sqrt{T-t}$$

La formula per una Put è invece data da:

$$P_t = X e^{-\delta(T-t)} N(-d_2) - S_t N(-d_1)$$

14.1 LA FORMULA EXCEL

Nella figura sottostante possiamo vedere come abbiamo organizzato i dati nel foglio della cartella di lavoro `black_scholes_formula.xls`. La colonna A contiene i commenti, la zona B4.B8 i dati necessari ai calcoli, la zona B13.B19 i risultati ottenuti con le formule Excel (per semplicità riscritte nelle stesse righe della colonna D). Le formule



	A	B	C	D	E	F	G	H
1								
2	**INPUT**							
3								
4	Prezzo corrente titolo (S)	25						
5	Prezzo d'esercizio (X)	25						
6	Intensita' ist. di interesse (delta)	0,06						
7	Tempo residuo a scadenza (T-t)	0,5						
8	Volatilita' del prezzo dell'opzione (sigma)	0,3						
9								
10								
11	**OUTPUT (Excel)**							
12								
13	d1	0,247487		=(LN(S/X)+(delta+0,5*sigma^2)*T-t)/(sigma*RADQ(T-t))				
14	d2	0,035355		=d1-sigma*RADQ(T-t)				
15	N(d1)	0,597734		=Distrib.Norm.St.(d1)				
16	N(d2)	0,514102		=Distrib.Norm.St.(d2)				
17	Call	2,470667		=S*N(D1)-X*EXP(-delta*T)*N(D2)				
18	Put (in base a parita' put/call)	1,731805		=Call-S+X*Exp(-delta(T-t))				
19	Put (formula BS)	1,731805		=X*EXP(-delta*(T-t))*N(-d2)-S*N(-d1)				
20								
21								
22								
23	**OUTPUT (VBA)**							
24								
25	d1							
26	d2							
27	N(d1)							
28	N(d2)							
29	Call							
30	Put (formula BS)							
31								

scritte nella zona D13.D19, per facilitarne la lettura, differiscono da quelle effettivamente presenti nella zona B13.B19 perché in queste celle al posto dei nomi dei dati si devono scrivere gli indirizzi. Per esempio la formula scritta in B14 è:

$$=B13-B8*RADQ(B7)$$

e non

$$=d1-sigma*RADQ(T)$$

Nella parte successiva del foglio si trova la zona in cui andremo ad inserire i risultati del programma VBA.

14.2 LA FORMULA IN VBA

La versione VBA è suddivisa in tre parti costituite nell'ordine dai controlli sulla presenza dei dati necessari ai calcoli (zona B4.B8), la chiamata al sottoprogramma `Black_Scholes`, la scrittura dei risultati dei calcoli nelle celle della zona B25.B30.

Il sottoprogramma `Black_Scholes` è stato incluso nella libreria comune in quanto ritenuto di uso frequente per altre procedure. Le sue istruzioni sono riprodotte di seguito.

```
Sub Black_Scholes(PrezzoTitolo As Single, _
                  PrezzoEsercizio As Single, _
                  Scadenza As Single, _
                  Interesse As Single, _
                  Sigma As Single, _
                  D1 As Single, _
                  D2 As Single, _
                  NormD1 As Single, _
                  NormD2 As Single, _
                  CCall As Single, _
                  PPut As Single)
    D1 = (Log(PrezzoTitolo / PrezzoEsercizio) + (Interesse + 0.5 * Sigma ^ 2) *
        Scadenza) / _
             (Sigma * Sqr(Scadenza))
    D2 = D1 - Sigma * Sqr(Scadenza)
    NormD1 = Application.NormSDist(D1)
    NormD2 = Application.NormSDist(D2)
    CCall = PrezzoTitolo * NormD1 - PrezzoEsercizio * Exp(-Interesse *
        Scadenza) * NormD2
      ...
```

Trattandosi di una funzione con più risultati, non poteva consistere in una Function. Tra i suoi argomenti elenchiamo i dati di input (i primi 5) e quelli risultanti dai calcoli al suo interno. Questi ultimi verranno utilizzati dal programma chiamante.
La variabile `CCall`, prezzo della Call, ha questo nome in quanto Call è una parola riservata al VBA e come tale non può essere utilizzata come nome di variabile (PPut lo abbiamo usato solo per analogia e non perché espressamente vietato).

ESERCIZI

1. Si chiede la procedura che calcola Call/Put con la formula di Black-Scholes in base ai dati acquisiti con una User Form. La User Form simile a quella visibile in figura deve contenere, tra gli altri, due campi denominati rispettivamente Call e Put al cui interno verranno scritti i risultati dei calcoli

Figura 14.1: La User Form dell'esercizio

15 LE SCELTE DI PORTAFOGLIO

Nella teoria delle scelte di portafoglio un posto di primo piano è costituito dal c.d. modello di Markowitz. In questo capitolo viene riportata la soluzione a questo problema. La cartella di riferimento si chiama `markowitz.xls`.

15.1 LA SOLUZIONE

La soluzione VBA che presentiamo, che individua i portafogli efficienti, si caratterizza per l'indipendenza dalla numerosità dei dati sia in relazione alla lunghezza dei periodo di osservazione sia per il numero degli asset che compongono il portafogli. Si nota come l'unico input necessiario, supponendo la presenza delle quotazioni nel foglio di lavoro (si veda il capitolo 16), sia costituito dalla definizione con il mouse della zona che contiene le osservazioni! (si veda, per esempio, Benninga [1]).

Tutti i risultati intermedi necessari al risultato finale sono presenti nel foglio di lavoro alo scopo di controllare i calcoli. Al termine viene visualizzato il grafico con la frontiera efficiente.

15.2 LA RICHIESTA DEI DATI

La soluzione prevede che le quotazioni dei titoli che fanno parte dell'asset si trovano già sul foglio. La specificazione della zona che contiene le quotazioni viene acquisita dal programma grazie ad un controllo RefEdit incluso nella UserForm disegnata per acquisire i dati. Pertanto visualizzata la finestra bisognerà semplicemente selezionare con il mouse la zona con le quotazioni che, dopo aver rilasciato il pulsante sinistro, verrà scritta nella casella RefEdit.

Nella stessa interfaccia di comunicazione con l'utente è presente un altro controllo utile a definire la posizione del grafico relativo alla frontiera efficiente (interno al foglio di lavoro oppure in un foglio ad hoc della stessa cartella). Nella figura sottostante riproduciamo questa interfaccia di dialogo.

15.3 IL CALCOLO DI REFEDIT

Una volta definita la della zona con le quotazioni dei titoli, dovremo procedere a calcolare, in base ad essa, una serie di dati utili ai calcoli successivi costituiti dalle matrici inverse, trasposte, etc. e, di

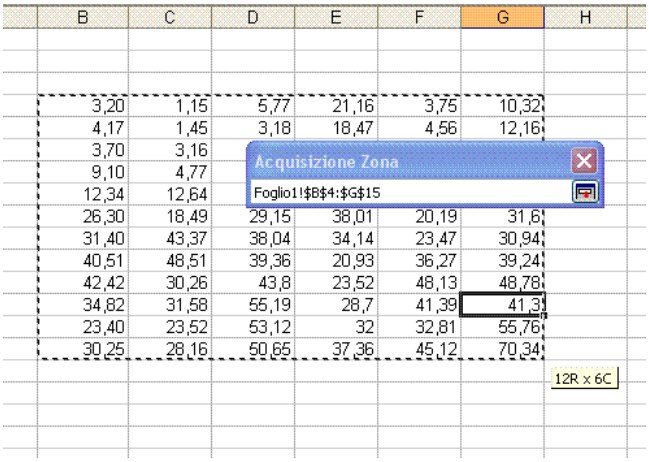
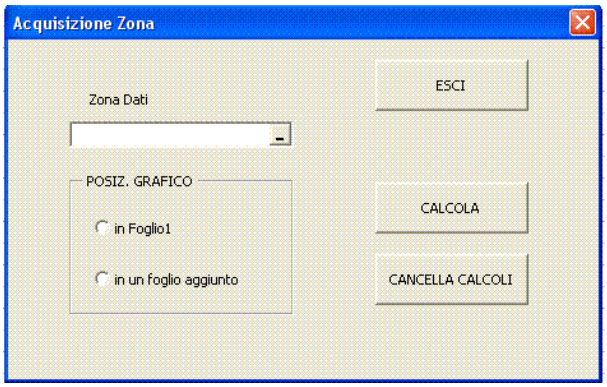


Figura 15.1: La selezione di una zona con Refedit



consguenza degli spazi necessari nel foglio di lavoro per riprodurle a fini di controllo.

La trasformazione della stringa costituita nel nostro esempio da:

```
Foglio1!$B$4:$G$15
```

in qualcosa corrispondente alle dimensioni in termini di numero di righe e colonne, costituisce l'oggetto di questo paragrafo.

Le parti in cui si articola il problema sono due:

- eliminazione dei caratteri estranei `Foglio1!` e `$`:
- trasformazione degli indirizzi di colonna da alfabetici a numerici per calcolare il numero di colonne e di righe della matrice con le quotazioni.

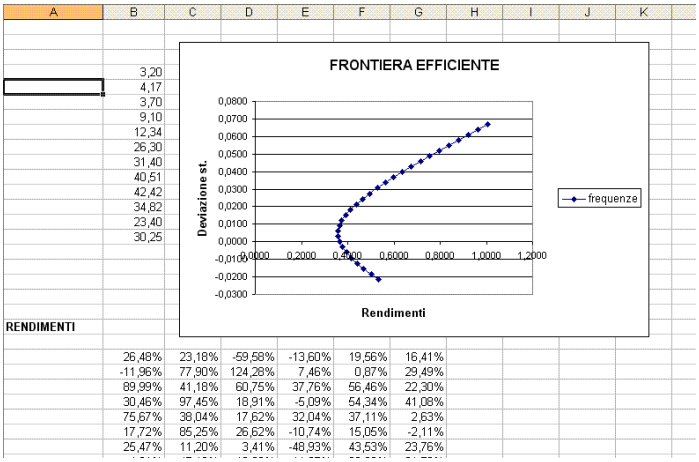
La soluzione, apparentemente complessa, ha utilizzato la posizione dei caratteri "!"e ":" come marcatori delle porzioni della stringa da scartare o da utilizzare per i nostri fini. Trattandosi di una procedura di uso frequente, è stata inserita nella libreria comune. Questa procedura è capace di funzionare anche al caso in cui la zona sia costituita da una sola cella. La riproduciamo di seguito in modo parziale.

```
Sub inizio_fineZona(Zona As String, foglio As String, iniZona As String, _
        finZona As String, nRighe As Integer, nColonne As Integer, _
        nrInizio As Integer, ncInizio As Integer, cInizio As String, _
        nrFine As Integer, cFine As String)
'
' Acquisisce una Zona come risultato di una RefEdit e restituisce:
    foglio, cella inizio,
'     cella fine

    Dim zonaNew As String, car As String
    Dim lunZona As Byte, posEscla As Byte, pos2Punti As Byte, I As Byte
'


    lunZona = Len(Zona)
' cerco il carattere ":", se non c'e e' stata selezionata solo una cella
     e non una zona
' per proseguire la zona Foglio1!\$B\$3 deve diventare Foglio1!\$B\$3:\
    $B\$3
    pos2Punti = InStr(1, Zona, ":")
    If pos2Punti = 0 Then
        pos\$ = InStr(1, Zona, "\$")
        Zona = Zona & ":" & Right(Zona, lunZona - pos\$ + 1)
        lunZona = Len(Zona)
        End If
    posEscla = InStr(1, Zona, "!")
    foglio = Left(Zona, posEscla - 1)
```

Il programma, risolto il problema del calcolo delle dimensioni della zona, consiste di una sequenza piuttosto lunga di calcoli quali media, deviazione standard, matrice varianze e covazianze, etc. che sono in funzione delle dimensioni dei dati di partenza. Il risultato finale, costituito dal grafico della frontiera efficiente, viene riprodotto nella figura sottostante. In questo caso era stata selezionata l'opzione di inserimento del grafico nel foglio con i dati. Le funzionalità di cancel-



lazione dei dati relativi ai calcoli e degli eventuali grafici sono state definite suddividendole in:

- la definizione del pulsante CANCELLA CALCOLI che provvede alla cancellazione parziale/totale di tutti i dati e del grafico eventualmente presente in un foglio a parte all'interno dell'intergaccia di dialogo;
- la definizione di una procedura sotto Modulo1 che provvede alla cancellazione del grafico eventualmente presente all'interno di Foglio1.

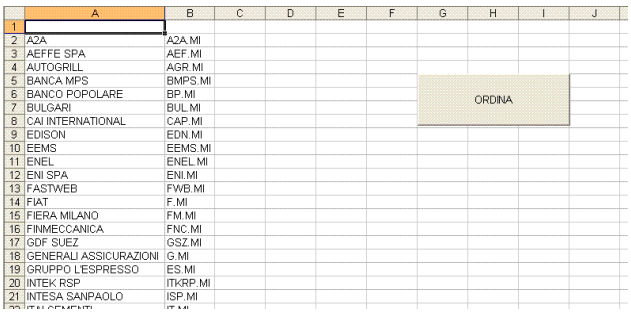
16 LE QUOTAZIONI DEI TITOLI

Nel capitolo precedente le quotazioni di un insieme di titoli erano presenti nel foglio di lavoro e sono state utilizzate per il calcolo della frontiera efficiente in base al modello di Markowitz. Il problema che risolveremo in questo capitolo riguarda le modalità con cui scaricare sul foglio quotazioni di titoli da siti web sempreché in un formato compatibile con il foglio elettronico. La soluzione VBA, per ragioni che illustreremo nel seguito, è stata suddivisa in due parti corrispondenti ad altrettante cartelle: la gestione dei nomi/codici dei titoli e lo scaricamento delle quotazioni.

16.1 I CODICI DI BORSA

La cartella di lavoro `elenco_titoli.xls` è deputata alla gestione "anagrafica" dei titoli. Bisogna sapere che le quotazioni dei titoli di una determinata società sono individuati da una sigla o "codice", in gergo borsistico. Il codice individua univocamente società e la borsa cui la quotazione da scaricare si riferisce e, attraverso questo codice, si procede alla richiesta della quotazione sul sito Web.

A titolo d'esempio, FIR.MI, FIP.MI, FSER.PA sono i codici dei titoli di "Fiat Industrial rsp" alla borsa di Milano, "Fiat Industrial prv" di Milano, di "Fiat Group S.p.A." alla borsa di Parigi[1]. La figura sottostante riproduce il contenuto della cartella in cui sono elencati la denominazione e il codice di una trentina di i titoli. Come si vede



nella colonna A abbiamo la denominazione e nella colonna B la sigla corrispondente. Si nota, in particolare, che i titoli sono ordinati alfabeticamente in base alla denominazione. Di conseguenza, ogni volta che ne aggiungiamo qualcuno dovremo procedere ad un loro ordi-

1 nomi e sigle si riferiscono all'estate 2011 e potrebbero essere diversi a seconda del riferimento temporale.

namento. Per svolgere questa funzionalità abbiamo a disposizione il pulsante ORDINA visibile in figura[2].

La creazione di questo pulsante si ottiene, a partire dalla finestra di Excel, con la sequenza di comandi del menu `Inserisci` → `Immagine` → `Forme`. Fatto questo si seleziona, nella finestra che compare, l'icona corrispondente al nome Forme e nella nuova finestra uno dei simboli che la compongono (noi abbiamo scelto la prima di queste). Dopo aver fatto clic, il puntatore del mouse si presenta come una crocetta che sta ad indicare che ci troviamo in una modalità di inserimento della forma prescelta. Dopo esserci posizionati sul punto di inserimento, mantenendo premuto il tasto sinistro del mouse, disegnamo la forma della grandezza desiderata. In un secondo momento possiamo procedere nelle modalità consuete ad un suo riposizionamento ovvero alla modifica delle sue dimensioni.

Terminato il disegno del pulsante bisogna associare il codice della macro all'oggetto inserito con l'apertura del menu contestuale. Questa associazione, che può essere fatta prima o dopo aver scritto il codice, definirà le istruzioni da eseguire al clic sull'oggetto.

16.2 LE QUOTAZIONI DEI TITOLI

Una seconda applicazione si occupa dello scaricamento vero e proprio delle quotazioni. Si tratta di una parte abbastanza articolata anche in relazione al fatto che, per ciascun titolo, vengono comunque restituite ben 6 colonne di dati. Nella figura sottostante viene mostrato un esempio dei dati scaricati per un titolo. I dati sono organizzati

	A	B	C	D	E	F	G	H
1	Date	Open	High	Low	Close	Volume	Adj Close	
2	05/05/2011	1.23	1.23	1.21	1.21	8443800	1.21	
3	04/05/2011	1.23	1.24	1.22	1.22	16207500	1.22	
4	03/05/2011	1.23	1.24	1.22	1.22	9599600	1.22	
5	02/05/2011	1.22	1.24	1.22	1.23	5929800	1.23	
6	29/04/2011	1.23	1.23	1.22	1.22	7330400	1.22	
7	28/04/2011	1.22	1.25	1.22	1.23	22234800	1.23	
8	27/04/2011	1.19	1.22	1.19	1.21	22184400	1.21	
9	26/04/2011	1.17	1.19	1.16	1.18	8902100	1.18	
10	21/04/2011	1.17	1.17	1.16	1.17	5384500	1.17	
11	20/04/2011	1.15	1.17	1.15	1.17	7795400	1.17	
12	19/04/2011	1.15	1.16	1.14	1.15	5932300	1.15	

sotto forma di matrice a due dimensioni. Nelle righe abbiamo le osservazioni per data di rilevazione in base a quanto selezionato nella finestra di dialogo (si veda in seguito). Nelle colonne, in corrispondenza di ciascuna data, sono elencati quotazione di apertura, massima, minima, etc. Poiché per il calcolo della fontiera efficiente ne potre-

2 Non sono ammesse righe vuote; per una loro eliminazione bisogna procedere manualmente.

mo utilizzare solo una, nella finestra di dialogo è stato inserito un opportuno strumento nel frame intitolato `COLONNA DA SCARICARE`.

Nella figura sottostante è visibile la finestra di dialogo che abbiamo definito per questa applicazione. Al suo interno sono presenti due nuovi strumenti costituiti da:

- il calendario;
- la casella di riepilogo.

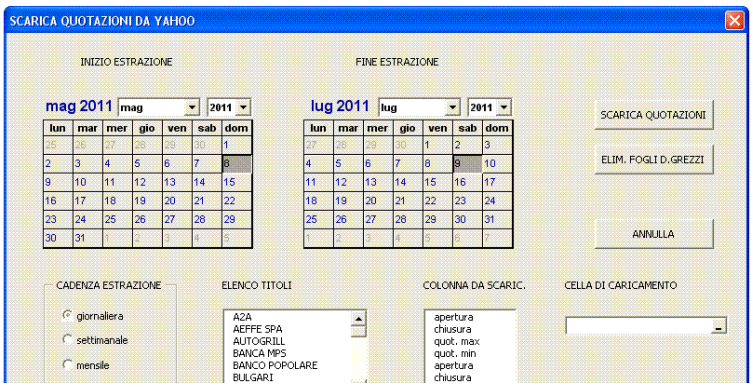
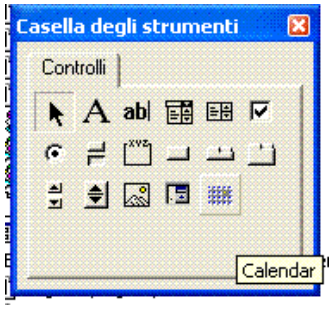


Figura 16.1: la finestra per i dati da Yahoo

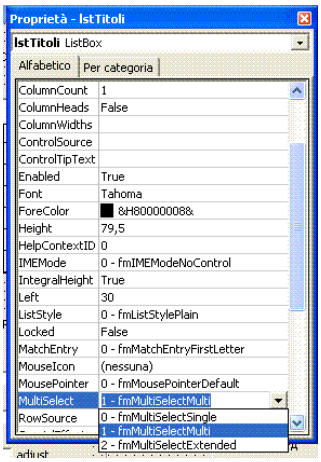
Il calendario è uno strumento di grande utilità nell'acquisizione di date perché impedisce errori formali quali p. e. "45" come numero del giorno oppure "marzio" come nome del mese. Non è disponibile nella dotazione standard di Excel 2003 e in questa versione, affinché sia visibile nell'interfaccia di dialogo, è necessario scaricare da Internet un'apposito programma[3].



3 si tratta dell'applicazione Microsoft AccessRuntime.exe.

Nella realizzazione della finestra di dialogo sono da tener presenti i seguenti punti:

- allo scopo di permettere la selezione di più titoli, nella riga che elenca le proprietà della casella di riepilogo con i nomi dei titoli (corrispondente alla riga MultiSelect, bisogna scegliere l'opzione frmMultiSelectMulti (vedere la figura sottostante);
- la colonna da scaricare deve essere solo una e, di conseguenza, lo strumento associato deve obbligare ad una scelta mutuamente esclusiva.



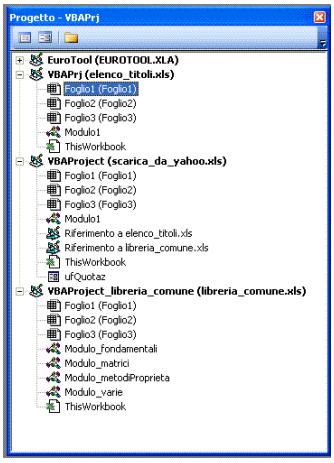
16.3 LO SCARICAMENTO DEI DATI

La seconda parte della nostra applicazione si occupa dello scaricamento vero e proprio dei dati da Internet. Trattandosi di una soluzione piuttosto articolata la riassumiamo nei seguenti punti:

1. visualizzazione della finestra di dialogo;
2. ciclo che si ripete per il numero di titoli selezionati;
   - per ogni titolo, aggiunta di un nuovo foglio inteso come worksheet;
   - nel nuovo foglio verranno riportate tutte e 6 le colonne di dati per quel titolo.

Allo scopo di chiarire meglio l'organizzazione del codice, riteniamo utile riprodurre la finestra del progetto nella figura sottostante. Come possiamo vedere, partendo dall'alto, abbiamo ben tre cartelle: quella che gestisce i nomi di tutti i titoli (`elenco_titoli.xls`), la cartella corrente (`scarica_da_yahoo.xls`) e la cartella della libreria comune.

Le istruzioni relative al punto 1 della soluzione si trovano in Modulo1. Nell'unica procedura che ne fa parte, prima della visualizzazione della finestra di dialogo devono essere inseriti i codici dei titoli in cui



selezionare quelli da scaricare ed il nome della colonna delle quotazioni (si tenga presente quanto riprodotto nella figura 16.2). Si noti anche, alla fine del secondo ciclo, il comando che spunta una delle tre opzioni relativa alla cadenza di scaricamento.

```
        . . .
For I = 1 To nTitoli
            ufQuotaz. lstTitoli .AddItem nomeTitoli(I)
             Cells(I, 256).Value = siglaTitoli (I)
            Next I
     For I = 1 To 5
            ufQuotaz.lstColonna.AddItem nomeColonna(I)
            Next I
'               spunto una delle tre opzioni
    ufQuotaz.optGiorna.Value = vbChecked
        . . .
```

La sintassi utilizzata nell'istruzione che inserisce le sigle nella casella di riepilogo sfrutta il metodo `AddItem` applicato all'oggetto`lstTitoli`. Lo stesso viene fatto per la casella relativa alla colonna da scaricare (nel ciclo successivo).

Selezionando uf_scarica_da_yahoo nella finestra del progetto, possiamo vedere il codice associato agli eventi di questa finestra di dialogo che ha come obiettivo la stringa per l'URL che scaricare i dati da Yahoo.

Prima di occuparcene un cenno alle istruzioni relative alle date selezionate con lo strumento `calendar`. Tutte le date vengono riportate alla forma gg/mm/aa grazie all'uso della funzione VBA Format. In particolare al numero del mese viene sottratto 1. Così se il mese è Gennaio questo varrà 0, se Febbraio 1, etc.

```
    . . .
dInizio = cldInizio.Value
dFine = cldFine.Value
ggI = Format(Day(dInizio), "00")
mmI = Format(Month(dInizio) - 1, "00")
aaI = Format(Year(dInizio), "00")
ggF = Format(Day(dFine), "00")
mmF = Format(Month(dFine) - 1, "00")
aaF = Format(Year(dFine), "00")
    . . .
```

16.4 LA STRINGA URL DI YAHOO

Una volta presisposti tutti i dati nel modo richiesto si può passare alla formazione della stringa che dovrà chiedere i dati al sito di Yahoo. In relazione al punto 2 della soluzione del paragrafo precedente, eseguiamo un ciclo che si ripete per il numero di titoli selezionati (da 0 a K - 1).

Nella stringa URL sono visibili 6 quantità identificate da `&a &b &c . ., &f .` Si tratta di delimitatori la cui funzione è quella di indicare le date che formano l'intervallo di scaricamento. Le altre due, `&g &z`, sono relative rispettivamente a cadenza e sigla del titolo (come facilmente desimibile dai nomi delle variabili che abbiamo messo di fianco).

Le due istruzioni successive servono ad aggiungere un foglio e a dare a questo un nome corrispondente alla sigla del titolo. Fatto questo si procederà all'inserimento nelle prime 6 colonne di questo foglio, a partire da A1, le quotazioni richieste per l'intervallo di tempo selezionato sul calendario.

```
    . . .
For I = 0 To K - 1
    URL = "URL;http://chart.yahoo.com/table.csv?s=" & siglaTitoli(I)
    URL = URL & "&a=" & mmI & "&b=" & ggI & "&c=" & aaI & _
          "&d=" & mmF & "&e=" & ggF & "&f=" & aaF & _
            "&g=" & cadenza & "&q=q&y=0&z=" & siglaTitoli(I) & "&x=.
                csv"
    Sheets.Add after:=Sheets(Sheets.Count)
    ActiveSheet.Name = siglaTitoli(I)
    . . .
```

Dopo aver copiato in altrettanti fogli i dati richiesti, si procede alla copia di una delle loro colonne (quella selezionata nella finestra di dialogo) in (Foglio1. Per poter eseguire questa operazione è necessario fare riferimento alla colonna scelta nella finestra di dialogo (Figura 16.1) e applicarla alla zona di input per la copia (la prima istruzione

del codice sottostante).

Occorre poi tenere presente la cella di caricamento selezionata sempre nella stessa finestra. Si noti la variabile corrispondente `IniZona` e l'uso della proprietà Offset che facilita il corretto indirizzamento per la copia della date associate a ciascuna riga delle quotazioni.

```
 zonaInput = nomeCol(colScar)
    For I = 4 To Sheets.Count
' seleziona il foglio di input
        Worksheets(Sheets(I).Name).Activate
        ActiveSheet.Range(zonaInput, ActiveSheet.Range(zonaInput).End(xlDown)).
            Select
        Selection.Copy
        Range("A1").Select
' seleziona il foglio di output
        Worksheets(Sheets(1).Name).Activate
' scrive la testata delle colonne (sigla dei titoli
        Range(Inizona).Offset(-1, I - 4).Select
```

L'ultima osservazione riguarda la cancellazione dei tutti i fogli scaricati (quelli che contengono i dati grezzi). La funzionalità è associata all'evento Click sul pulsante di comando ELIM. FOGLI D. GREZZI.

Questa operazione sfrutta un ciclo `For each` che permette di indirizzare tutti gli oggetti di una collezione. Nel nostro caso l'oggetto è aSheet mentre la collezione è `ActiveWorkbook.Sheets`. Terminato il ciclo si scarica la finestra di dialogo.

```
Private Sub btnEliminaDGrezzi_Click()
    Dim aSheet As Object
    For Each aSheet In ActiveWorkbook.Sheets
'       eliminiamo tutti i worksheet con nome differente da Foglio(n)
        If Left(aSheet.Name, 6) <> "Foglio" Then
            Sheets(aSheet.Name).Delete
        End If
    Next
    Unload Me
End Sub
```

17 IL METODO MONTE CARLO

Nell'ambito della valutazione di un'opzione, si può verificare il caso in cui il suo valore dipenda dall'evoluzione dei prezzi del sottostante prima della scadenza T (*opzioni Americane e Asiatiche*). Per queste ultime, chiamate appunto asiatiche aritmetiche, si perviene ad una stima del payoff atteso scontato come media aritmetica[1] di "cammini" o "paths" di prezzi scontati del sottostante riferiti a varie epoche t_i con $0 \geqslant i \geqslant T$. Nella parte che segue, ripresa da Glasserman [6], descriviamo i passaggi che calcolano il payoff nel modo descritto in precedenza.

17.1 METODOLOGIA RISOLUTIVA

Il punto di partenza è costituito dalla matrice dei cammini:

$$\boldsymbol{S} = \begin{bmatrix} S_{11} & S_{12} & \dots & S_{1m} \\ S_{21} & S_{22} & \dots & S_{2m} \\ \vdots & & \vdots & \vdots \\ S_{n1} & S_{n2} & \dots & S_{nm} \end{bmatrix}$$

in cui il generico elemento di una riga è dato dalla relazione:

$$S_{ij+1} = S_{ij} exp\left[(r - \frac{1}{2}\sigma^2)(t_{j+1} - t_j) + \sigma\sqrt{t_{j+1} - t_j}\, Z_{ij+1}\right]$$

con S_{i1} per $1 \geqslant i \geqslant n$ pari al prezzo corrente, mentre ciascun Z_{ij} si riferisce al generico elemento di una matrice di variabili casuali normali standardizzate riportata nella figura seguente.

$$\boldsymbol{Z} = \begin{bmatrix} Z_{11} & Z_{12} & \dots & Z_{1m} \\ Z_{21} & Z_{22} & \dots & Z_{2m} \\ \vdots & & \vdots & \vdots \\ Z_{n1} & Z_{n2} & \dots & Z_{nm} \end{bmatrix}$$

Effettuata la generazione della matrice **S** si perviene alla stima di un vettore colonna $\overline{\mathbf{S}}$ di n elementi in cui, ciascun elemento è dato dalla relazione:

$$\overline{S_i} = \frac{1}{m}\sum_{j=1}^{m} S(t_j)$$

1 questa metodologia prende il nome di metodo o stimatore di Monte Carlo

Ciascuno degli elementi del vettore $\overline{\mathbf{S}}$ verra' utilizzato per generare un vettore $\mathbf{C}$ anch'esso di n elementi in cui il generico elemento sarà dato dalla relazione:

$$C_i = exp(-rt) * max(0, \overline{S}_i - X)$$

che finalmete permetterà di calcolare il valore attuale stimato della call in base alla relazione:

$$\overline{C} = \frac{1}{m} \sum_{j=1}^{m} C_j$$

17.2 LA SOLUZIONE VBA

Dopo aver descritto i passaggi della soluzione, possiamo discutere della sua implementazione in VBA.

Si tratta dei dati del problema di Black-Scholes (prezzo iniziale, volatilità etc.) cui vanno aggiunti due ulteriori informazioni costituite dal numero di cammini e dal numero di sottoperiodi. Il loro inserimento è ottenuto con la finestra di dialogo che vediamo nella figura sottostante.

Nel codice seguente, relativo al verificarsi dell'evento CALCOLA, sono riportate le istruzioni che dopo aver scaricato dalla finestra di dialogo `uf_monteCarloNE` il contenuto dei campi dell'interfaccia di dialogo, controllano la presenza di dati coerenti. In caso di rilievi viene visualizzato un opportuno messaggio.

```
With Me
        Dato(1) = .txtCamm
        Dato(2) = .txtSottop
        Dato(3) = . txtPrezzoIniziale
        Dato(4) = .txtPrezzoFissato
        Dato(5) = .txtTassoInt
        Dato(6) = .txtScadenza
```

```
        Dato(7) = .txtSigma
    End With
For I = 1 To 7
            If IsNumeric(Dato(I)) = False Then
                    MsgBox prompt:=Richiesta(I) & _
                                " manca o non e' numerico", _
                                Buttons:=vbCritical, _
                                Title:="ERRORE NEI DATI"
                    Exit Sub
                    End If
            Next I
```

Coerentemente con l'implementazione dei dati in un foglio Excel, dovremo controllare di non superare i limiti imposti dal numero di colonne disponibili (al massimo 256). Qui abbiamo supposto un numero massimo di 100 colonne e 1000 righe. Questi dati sono stati assegnati alle due celle riferite nell'istruzione condizionale.

```
If Cells(1, 4).Value > 100 Or _
      Cells(2, 4).Value > 1000 Then
                MsgBox prompt:=" Superati i limiti per n, oppure m", _
                                    Buttons:=vbCritical, _
                                    Title:="ERRORE NEI DATI"
```

Come da nostra consuetudine generiamo i numeri casuali con l'algoritmo di Mersenne Twister che si trova nella nostra libreria dei programmi.

```
   . . .
                                                REM innesca il
                                                    generatore
init_genrand 4
    For I = 1 To N                              ' riempie la zona a
        partire da Cells(10,1)
                                                ' con i numeri casuali
        For J = 1 To M
            ok = True
            Do While ok
                                                ' genera le coppie
                rand1 = 2 * genrand_real3() - 1
                rand2 = 2 * genrand_real3() - 1
```

Una volta generata la matrice dei numeri casuali normali si passa a calcolare la matrice **S** attraverso un doppio ciclo che ha come unica difficoltà il "salvataggio" del prezzo iniziale. Nel codice seguente relativo al calcolo della matrice si nota nella prima istruzione l'assegnazione di questo valore alla variabile `salvaS` ed il suo riutilizzo nel corso del ciclo (prima del secondo `For`).

```
salvaS = s0
    STotCi = 0
    iRiga = 10 + N + 2
    Cells(iRiga, 1).Value = "Media"
    Cells(iRiga, 2).Value = "C"
    Cells(iRiga, 3).Value = "Media C(i)"
    For I = 1 To N
        sTot = 0
        s0 = salvaS
        For J = 1 To M
            Cells(I + iRiga, J + 2).Value = _
                    s0 * Exp((erre - 0.5 * sigma ^ 2) * t + sigma * _
                                Sqr(t) * Cells(11 + I, J).Value)
            s0 = Cells(I + iRiga, J + 2).Value
            sTot = sTot + s0
            Next J
        sMed(I) = sTot / CDbl(M)
        Ci(I) = Exp(-erre * t) * Application.Max(0, sMed(I) - kappa)
        Cells(I + iRiga, 1).Value = sMed(I)
        Cells(I + iRiga, 2).Value = Ci(I)
        STotCi = STotCi + Ci(I)
        Next I
Cells(2, 8) = STotCi / CDbl(N)
```

L'ultima riga del codice, calcolato il valore stimato della Call con il metodo Monte Carlo, riporta il risultato nella cella H2.

Parte III

APPENDICI

A

IL REGISTRATORE DELLE MACRO

Il programma Excel ha la possibilità di registrare tutte le azioni che l'utente effettua sul foglio elettronico. Questa funzionalitá si attiva con la sequenza di comandi del menu Strumenti → Macro → Registra nuova macro .... Dopo aver selezionato questi comandi, viene visualizzata una finestra di dialogo che permette di definire diversi dati tra cui il nome della macro (ovvero il nome della procedura) ed altri valori. La vediamo riprodotta nella figura sottostante. Dopo aver

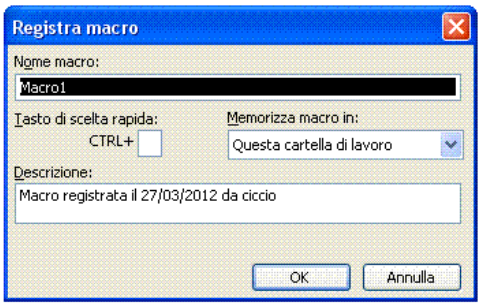


Figura A.1: La finestra che attiva il registratore

premuto il tasto OK è attivo il registratore che traduce tutte le attività svolte sul foglio Excel nelle corrispondenti istruzioni VBA.

A.1 ATTIVAZIONE E DISATTIVAZIONE DEL REGISTRATORE

La segnalazione che è attivato il registratore si desume dalla presenza di un apposita informazione sulla barra di stato (quella in basso a sinistra della finestra di Excel). Quando il registratore è attivo compare la scritta Registr.. Un altro segnale dell'attività di registrazione è dalla comparsa di una piccola finestra simile a quella che vediamo nella figura sottostante. Qualora questa non fosse visibile si può attivare

Figura A.2: La finestra del registratore

spuntando la voce Interrompi registrazione ottenuta dalla sequenza di comandi Strumenti → Personalizza e viceversa.

La fine della registrazione si può ottenere con la sequenza di comandi del menu Strumenti → Macro → Interrompi Registrazione.

Il codice della registrazione è visibile nell'editor VBA con i consueti comandi.

A.2 PRO E CONTRO

A questo punto il lettore più smaliziato potrebbe pensare che questa funzionalitá renda del tutto vano il significato di un libro sulla programmazione VBA ma non è così perché la tecnica della registrazione comporta diverse conseguenze che possono essere facilmente comprese considerando il seguente esempio.

Dopo aver attivato il registratore, inseriamo in tre celle a piacere altrettati dati ed analizziamo il codice ottenuto con la registrazione.

```
    . . .
    Range("F18").Select
        ActiveCell .FormulaR1C1 = "ahrd"
    Range("F19").Select
    ActiveCell .FormulaR1C1 = "djue"
    Range("F20").Select
    ActiveCell .FormulaR1C1 = "deks"
    Range("F21").Select
End Sub
```

Nell'esempio vediamo il codice prodotto dall'inserimento in tre celle di altrettante stringhe di caratteri. Si riconoscono i comandi di selezione ed inserimento di un dato nella cella attiva rispettivamente Range(F18).Select e ActiveCell.FormulaR1C1 = "ahrd".

Le considerazioni su questo frammento di codice sono estendibili a tutto quanto viene generato con la registrazione e riassumibili nei seguenti punti:

- il codice non è generalizzabile;
- dal momento che ci troviamo in ambiente Excel non è possibile utilizzare acuno strumento VBA come per esempio InputBox o cicli.

Nondimeno la tecnica è utile in situazioni particolari quali per esempio l'impossibilità di trovare un codice VBA o in riferimento a quanto detto nel paragrafo 13.4.

A.3 IL NOME DEL GRAFICO

La conoscenza del nome relativo al tripo di gafico che vogliamo disegnare, può essere ottenuta con questi passaggi:

1. attivare il registratore delle macro;
2. selezionare la sequenza di comandi del menu Inserisci → Grafico;
3. nella finestra intitolata Creazione .. - Passaggio 1 di 4 .. effettuare una selezione del tipo (a sinistra) e, a partire da questo, delle scelte (a destra);

4. per tutte le finestre successive relative ai passaggi da 2 a 4, premere sempre il tasto Avanti senza alcuna selezione ulteriore e chiudere con il tasto Fine;
5. interrompere la registrazione;
6. passare in ambiente VBA e prendere nota del tipo di grafico che si trova in corrispondenza della proprietà `ActiveChartType`.

La stessa procedura, cambiando quanto c'è da cambiare, si può impiegare per il nome delle funzioni in inglese.

BIBLIOGRAFIA

[1] Simon Benninga. *Modelli Finanziari*. McGraw–Hill, Milano, 2010.

[2] Robert Bringhurst. *The Elements of Typographic Style*. Version 3.2. Hartley & Marks Publishers, Point Roberts, WA, USA, 2008.

[3] Simone Borra e Agostino Di Ciaccio. *Statistica*. McGraw–Hill, Milano, 2008.

[4] Zwirner Giuseppe e Pavarin Vittorio. *Matematica Finanziaria*. Cedam, Padova, 1969.

[5] Lorenzo Pantieri e Tommaso Gordini. *L'arte di scrivere con LaTeX*, 2012. URL http://www.lorenzopantieri.net.

[6] Paul Glasserman. *Monte Carlo Methods in Financial Engineering*. Springer, New York, NY, USA, 2010.

[7] André Miede. *A Classic Thesis Style*, 2012. URL http://www.miede.de.