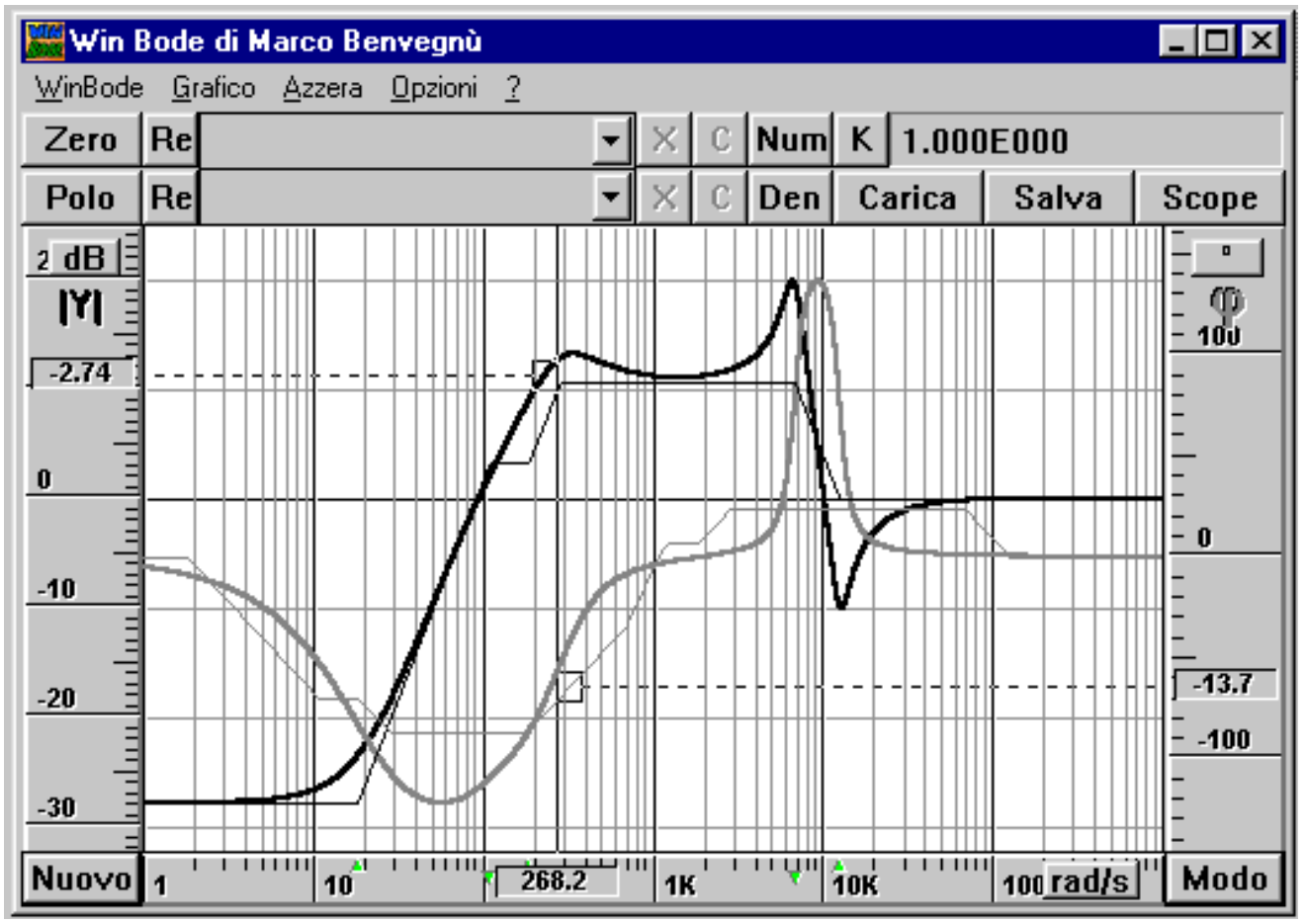
WinBode

DESCRIZIONE DEL CODICE SORGENTE



ESAMI DI MATURITA'

Anno Scolastico 1997/1998

Allievo : Marco Benvegnù

Classe : 5^{A} Elettronici, Sez. B

INTRODUZIONE

Questa relazione descrive il codice sorgente di WinBode, un'applicazione Windows che permette di tracciare i diagrammi di Bode asintotici e reali di una funzione qualsiasi.

Le caratteristiche principali di WinBode sono:

- facilità di inserimento della funzione, qualunque sia la forma in cui è espressa: forma di Bode, costanti di tempo, forma fattorializzata semplice, poli e zeri, polinomi al numeratore e al denominatore, e combinazioni delle precedenti;
- possibilità di confronto diretto fra diagramma reale ed asintotico e valutazione dei relativi errori di approssimazione;
- presenza di vari strumenti che facilitano l'analisi del diagramma ottenuto:
    - ◊ linee guida che visualizzano i valori del modulo e della fase e la differenza fra diagramma reale ed asintotico alla pulsazione alla quale si trova il puntatore;
    - ◊ una finestra che, a partire dalla pulsazione inserita da tastiera, ricava tali valori in modo molto più preciso (v. finestra «Scope»);
    - ◊ linee che evidenziano le ordinate a 0dB e a 180° ± k 360°, per lo studio della stabilità nei sistemi reazionati;
    - ◊ demarcatori, sull'asse della pulsazione, che evidenziano la posizione delle radici inserite, con distinzione fra poli e zeri reali e coppie complesse coniugate;
- asse delle ascisse in rad/s o in Hz;
- asse del modulo in dB o in valore assoluto lineare;
- asse della fase in gradi o radianti;
- possibilità di salvataggio ed esportazione dei dati
- possibilità di stampa

Al programma sono state apportate alcune modifiche durante la stesura della relazione, atte a rendere più leggibile il codice. In effetti esso è stato ottimizzato per la chiarezza: molte parti si sarebbero potute raccogliere in procedure, ad esempio le sezioni che affrontano i vari processi sui poli e poi sugli zeri sono molto simili e sono state ottenute con operazioni di copia ed incolla e relative modifiche; tuttavia, se non si fosse fatto in questo modo, la comprensione del codice sarebbe diventata cosa ardua.

Non sono stati aggiunti tanti commenti al codice, ma l'uso di identificatori chiari e di un'adeguata indentazione dovrebbero evitare confusioni.

Inoltre si noti che, mentre nei segmenti di codice trascritti nella relazione sono stati eliminati i punti e virgola prima degli «end» (considerati da molti come errore di forma), nel codice reale essi sono ancora presenti: ciò deriva dal fatto che io utilizzo una programmazione di linea, cioè faccio corrispondere ad ogni riga un comando, ed utilizzo delle macro che automaticamente aggiungono il punto e virgola quando si va a capo, per cui, eliminarlo prima dell'«end» e riaggiungerlo ogni volta che inserisco una nuova istruzione risulterebbe in una perdita di tempo.

Le procedure che gestiscono la grafica sono state in parte ricavate da un altro programma di mia produzione (presente nel dischetto alla cartella 4Poli). In particolare l'oggetto TGraficaHT, che in WinBode è poco utilizzato, è stato creato proprio per 4Poli, nel quale mette in evidenza le proprie potenzialità.

In questa relazione ho cercato di spiegare gli aspetti fondamentali della programmazione orientata agli oggetti (OOP) e della programmazione Windows, sorvolando su parecchi dettagli; in questo modo, anche chi non conosce il Pascal per Windows, ma ha una certa conoscenza del Pascal per DOS, dovrebbe comprendere, in linea generale, il funzionamento di base del programma. Tuttavia, alcune semplificazioni che si sono rese necessarie per la trattazione di argomenti delicati, potrebbero apparire, agli occhi dell'esperto, come delle eresie.

NOTA: il dischetto allegato contiene i seguenti file:

WinBode.Pas	codice sorgente in Pascal (si consiglia di visualizzarlo con il Pascal per Windows impostando le tabulazioni a due caratteri);
WinBode.Res	file delle risorse;
WinBode.Exe	versione eseguibile del programma.

Nella cartella 4Poli sono inoltre presenti i file relativi ad un programma che permette il calcolo delle impedenze immagine di ingresso ed uscita di un quadripolo; tuttavia esso non è stato completamente testato e può quindi presentare qualche imperfezione.

INIZIALIZZAZIONE

L'oggetto *TApplication* che sovraintende a tutta l'applicazione si chiama *TMyApplication*; esso è completamente definito dalla seguente sezione dichiarativa:

```
Type   TMyApplication = object(TApplication)
          procedure InitMainWindow; virtual;
          procedure InitInstance; virtual
       end;
```

la cui implementazione diventa:

```
procedure TMyApplication.InitMainWindow;
begin
   MainWindow := New(PBode, Init(nil, 'Win Bode di Marco Benvegnù'))
end;

procedure TMyApplication.InitInstance;
begin
   TApplication.InitInstance;
   HAccTable := LoadAccelerators(HInstance, 'WinBode')
end;
```

InitInstance carica la tabella dei tasti di scelta rapida, mentre *InitMainWindow* assegna all'oggetto puntato da *PBode (TBode)* il compito di gestire la finestra principale.

In effetti *TBode* rappresenta il corpo principale del programma e la procedura di inizializzazione ad essa inerente è la prima a dare effetti concreti.

```
Constructor TBode.Init(AParent: PWindowsObject; ATitle: PChar);
var Mr,Mg,Mb:Byte;
begin
   inherited Init(AParent, ATitle);
   Attr.Menu := LoadMenu(HInstance, 'Menu_1');
   Attr.X:=GetPrivateProfileInt('WinBode','X',20,'WinBode.Ini');
   Attr.Y:=GetPrivateProfileInt('WinBode','Y',20,'WinBode.Ini');
   Attr.W:=GetPrivateProfileInt('WinBode','W',600,'WinBode.Ini');
   Attr.H:=GetPrivateProfileInt('WinBode','H',440,'WinBode.Ini');
   MR:=GetPrivateProfileInt('WinBode','MR',$FF,'WinBode.Ini');
   MG:=GetPrivateProfileInt('WinBode','MG',$E5,'WinBode.Ini');
   MB:=GetPrivateProfileInt('WinBode','MB',$00,'WinBode.Ini');
   ModCol:=Rgb(Mr,Mg,Mb);
   MR:=GetPrivateProfileInt('WinBode','FR',$FF,'WinBode.Ini');
   MG:=GetPrivateProfileInt('WinBode','FG',$00,'WinBode.Ini');
   MB:=GetPrivateProfileInt('WinBode','FB',$00,'WinBode.Ini');
   FasCol:=Rgb(Mr,Mg,Mb);
   if GetPrivateProfileInt('WinBode','Reale',1,'WinBode.Ini')=0 then Reale:=False;
   if GetPrivateProfileInt('WinBode','Asintotico',0,'WinBode.Ini')=1 then Asintotico:=True;

   Grafico:=New(PGraf,Init(@Self,''));
   AsseX:=New(PAsseX,Init(@Self,''));
   AsseY:=New(PAsseY,Init(@Self,''));
   AsseF:=New(PAsseF,Init(@Self,''));
   CBZeri := New(PComboBox, Init(@Self, 800, 73, 0, 180, 100, cbs_DropDownList, 0));
   CBPoli := New(PComboBox, Init(@Self, 800, 73, 23, 180, 100, cbs_DropDownList, 0));
   CBZeri^.Attr.Style:=CBZeri^.Attr.Style and not cbs_Sort;
   CBPoli^.Attr.Style:=CBPoli^.Attr.Style and not cbs_Sort;
   Scuro:=CreatePen(0,1,$00707070);
   Bordo:=CreatePen(0,3,$00);
   Grigio:=CreatePen(0,1,$00909090);
   Bianco:=CreatePen(0,1,$00FFFFFF);
   Nero:=CreatePen(0,1,$00000000);
   ModPen:=CreatePen(0,2,ModCol);
   FasPen:=CreatePen(0,2,FasCol);
   ModPenAs:=CreatePen(0,1,ModCol);
   FasPenAs:=CreatePen(0,1,FasCol);
   PModPen:=CreatePen(0,3,ModCol);
   PFasPen:=CreatePen(0,3,FasCol);
   NodiPenIm:=CreatePen(0,1,$0000FF00);
   NodiPen:=CreatePen(0,1,$00FF0000);
   Tratt:=CreatePen(2,1,$00);

   FontGriglia:=CreateFont(14,0,0,0,700,0,0,0,0, out_default_precis,
                clip_default_precis, default_quality,0,'Arial');

   uKStat := New(PStatic, Init(@Self, 222, '', 358, 1, 160, 21, 20));
   uKStat := New(PStatic, Init(@Self, 221, '1', 362, 4, 155, 18, 20));
```

Lettura del file di configurazione WinBode.Ini per l'inizializzazione di diverse variabili, come le dimensioni e la posizione della finestra e i colori delle linee, salvate alla fine dell'ultima sessione.

Combo Boxes contenenti le liste degli zeri e dei poli con disattivazione dell'ordinamento automatico.

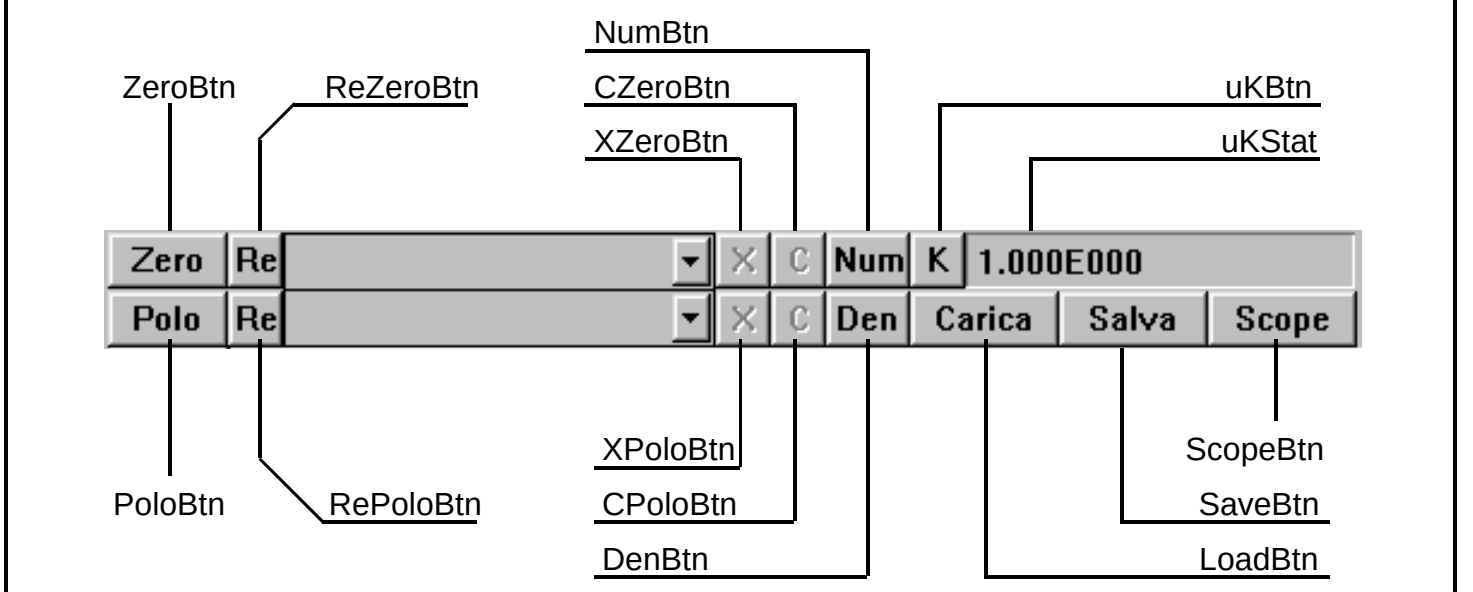
Creazione di tutti gli oggetti TPen che vengono utilizzati per tracciare le linee ed i grafici.

Font (tipo di carattere) utilizzato in comune dai tre righelli.

Le seguenti inizializzazioni comportano l'apparizione dei vari pulsanti. *ModeBtn* e *NuovoBtn* sono i due pulsanti in basso. Per quanto riguarda gli altri, essi appartengono tutti alla barra degli strumenti. L'immagine in basso chiarisce le corrispondenze fra le variabili ed i pulsanti.

```
   ZeroBtn := New(PButton, Init(@Self, 101, 'Zero', 0, 0, 50, 23, False));
   PoloBtn := New(PButton, Init(@Self, 102, 'Polo', 0, 24, 50, 23, False));
   ReZeroBtn := New(PButton, Init(@Self, 111, ' Re ', 50, 0, 23, 23, False));
   RePoloBtn := New(PButton, Init(@Self, 112, ' Re ', 50, 24, 23, 23, False));
   XZeroBtn := New(PButton, Init(@Self, 121, ' X ', 253, 0, 23, 23, False));
   XPoloBtn := New(PButton, Init(@Self, 122, ' X ', 253, 24, 23, 23, False));
   CZeroBtn := New(PButton, Init(@Self, 131, ' C ', 276, 0, 23, 23, False));
   CPoloBtn := New(PButton, Init(@Self, 132, ' C ', 276, 24, 23, 23, False));
   NumBtn := New(PButton, Init(@Self, 151, ' Num ', 299, 0, 35, 23, False));
   DenBtn := New(PButton, Init(@Self, 152, ' Den ', 299, 24, 35, 23, False));
   uKBtn := New(PButton, Init(@Self, 211, ' K ', 334, 0, 23, 23, False));
   LoadBtn := New(PButton, Init(@Self, 301, ' Carica ', 334, 24, 62, 23, False));
   SaveBtn := New(PButton, Init(@Self, 302, ' Salva ', 396, 24, 62, 23, False));
   ScopeBtn := New(PButton, Init(@Self, 303, ' Scope ', 458, 24, 62, 23, False));
   ModeBtn := New(PButton, Init(@Self, 110, ' Modo ',0,0, 10, 10, True));
   NuovoBtn := New(PButton, Init(@Self, 109, ' Nuovo ',0,0, 10, 10, True));
   StrCopy(InText,'');
   Printer := New(PPrinter, Init)
end;
```

Associazione alla stampante.



Il Destructor *TBode.Done*, chiamato alla chiusura di WinBode, fa l'esatto opposto di *TBode.Init*, liberando la memoria da tutti gli oggetti creati nel corso dell'applicazione. Infine, prima di chiamare il *Done* ereditato da *TWindow*, il quale gestisce la chiusura dell'oggetto, vengono salvate nel file di configurazione tutte leinformazioni necessarie al prossimo avvio di WinBode. Si noti che il salvataggio avviene solamente quando la variabile booleana *Salva* è impostata a TRUE, e ciò si ottiene con un'apposita voce del menu (Opzioni/Salva all'uscita).

```
Destructor TBode.Done;
var    D:TRect;
       T:Array [0..50] of Char;

begin
   FreeModule(ModuloBWCC);
   DeleteObject(FasPenAs);
   DeleteObject(ModPenAs);
   DeleteObject(FasPen);
   DeleteObject(ModPen);
   DeleteObject(Grigio);
   DeleteObject(Bianco);
   DeleteObject(Nero);
   DeleteObject(Scuro);
   DeleteObject(FontGriglia);
   DeleteObject(NodiPen);
   DeleteObject(NodiPenIm);
   DeleteObject(Tratt);
```

```
    if Salva then
    begin
       GetWindowRect(HWindow,D);
       Str(D.Left,T);
       WritePrivateProfileString('WinBode','X',T,'WinBode.Ini');
       Str(D.Top,T);
       WritePrivateProfileString('WinBode','Y',T,'WinBode.Ini');
       Str(D.Right-D.Left,T);
       WritePrivateProfileString('WinBode','W',T,'WinBode.Ini');
       Str(D.Bottom-D.Top,T);
       WritePrivateProfileString('WinBode','H',T,'WinBode.Ini');

       Str(GetRValue(ModCol),T);
       WritePrivateProfileString('WinBode','MR',T,'WinBode.Ini');
       Str(GetGValue(ModCol),T);
       WritePrivateProfileString('WinBode','MG',T,'WinBode.Ini');
       Str(GetBValue(ModCol),T);
       WritePrivateProfileString('WinBode','MB',T,'WinBode.Ini');
       Str(GetRValue(FasCol),T);
       WritePrivateProfileString('WinBode','FR',T,'WinBode.Ini');
       Str(GetGValue(FasCol),T);
       WritePrivateProfileString('WinBode','FG',T,'WinBode.Ini');
       Str(GetBValue(FasCol),T);
       WritePrivateProfileString('WinBode','FB',T,'WinBode.Ini');

       if Reale then WritePrivateProfileString('WinBode','Reale','1','WinBode.Ini')
          else WritePrivateProfileString('WinBode','Reale','0','WinBode.Ini');
       if Asintotico then WritePrivateProfileString('WinBode','Asintotico','1','WinBode.Ini')
          else WritePrivateProfileString('WinBode','Asintotico','0','WinBode.Ini');

       if Salva then WritePrivateProfileString('WinBode','Salva','1','WinBode.Ini')
          else WritePrivateProfileString('WinBode','Salva','0','WinBode.Ini')
    end;
    inherited Done
end;
```

Salvataggio delle dimensioni della finestra:

- X;
- Y;
- Larghezza;
- Altezza.

Salvataggio dei colori del modulo e della fase nel formato RGB (componenti del rosso, del verde e del blu).

Anche l'opzione di salvataggio automatico viene salvata.

La procedura virtuale *WmSize* viene eseguita ogni volta che le dimensioni della finestra cambiano. Essa è strutturata in modo che i pulsanti in basso, i righelli e il grafico appaiano sempre in modo corretto, e cioè vengano riposizionati in concomitanza ai cambiamenti di dimensione. In questo modo la finestra grafica può esserepiccola o grande a piacere, senza limiti di risoluzione.

```
procedure TBode.WmSize;
var R:TRect;

begin
   GetClientRect(HWindow, R);
   MoveWindow(Grafico^.HWindow, 50, 47, R.Right-100, R.bottom - 69, True);
   MoveWindow(AsseX^.HWindow,50,R.bottom -22, R.Right-100, 22, True);
   MoveWindow(AsseY^.HWindow,1,47, 49, R.bottom-69, True);
   MoveWindow(AsseF^.HWindow,R.Right-49,47,49, R.bottom-69, True);
   MoveWindow(ModeBtn^.HWindow,R.Right-50,R.bottom -23, 50, 23, False);
   MoveWindow(NuovoBtn^.HWindow,0,R.bottom -23, 50, 23, False);
   GetWindowRect(HWindow, R);
   with R do if Bottom-Top<200 then MoveWindow(Hwindow,Left,Top,Right-Left,200,True);
   with R do if Right-Left<200 then MoveWindow(Hwindow,Left,Top,200,Bottom-Top,True)
end;
```

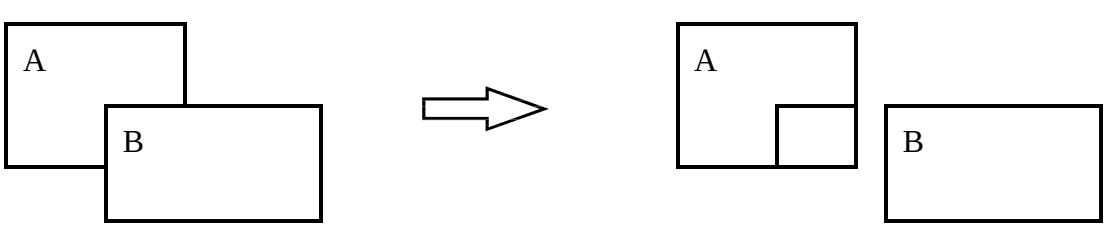
Le coordinate fornite da *GetClientRect* non sono assolute bensì relative: il punto di riferimento (0,0) si trova nell'angolo superiore sinistro. Perciò il limite superiore (Top) e quello sinistro (Left) sono sempre nulli, mentre quello destro (Right) e quello inferiore (Bottom) corrispondono rispettivamente alla larghezza ed all'altezzadell'area cliente, ossia della parte interna alla finestra con l'esclusione della barra del titolo, dell'eventuale menu a tendine e del frame (ossia il bordo).Le ultime tre righe servono ad impedire che la finestra venga restrinta al punto tale che l'area grafica sparisca.

GetClassName, *GetWindowClass* e *SetUpWindow* comportano alcune inizializzazioni tipiche di ogniapplicazione Windows e di scarso interesse, come il caricamento in memoria di librerie o la definizione del colore di fondo finestra.

Per creare l'effetto rilievo attorno all'area di inserzione della costante di guadagno statico (v. figura precedente) si è presentato necessario ricorrere alla semplice procedura *Paint* riportata di seguito:

```
procedure TBode.Paint(PDC: HDC; var PaintInfo: TPaintStruct);
begin
   SelectObject(PDC,Bianco);
   Moveto(PDC,357,22);
   Lineto(PDC,518,22);
   Lineto(PDC,518,0);
   SelectObject(PDC,Scuro);
   Lineto(PDC,357,0);
   Lineto(PDC,357,22)
end;
```

NOTA: Le procedure virtuali nominate *Paint* vengono chiamate da altre procedure (ad es. *UpdateWindow* e *RedrawWindow*) che a loro volta vengono attivate dal messaggio *WM_Paint*, inviato dal sistema operativo o da un'altra applicazione quando è necessario ridisegnare una parte della finestra. Ad esempio, facendo riferimento alla figura seguente, quando l'applicazione B, sovrapposta alla A, viene spostata, ad A giunge il messaggio *WM_Paint* con le coordinate dell'area tratteggiata che deve essere ridisegnata. Tali coordinate giungono alla procedura *Paint* attraverso il record *PaintInfo* assieme a molte altre informazioni.



FUNZIONI E PROCEDURE

Si visionano ora alcune delle procedure e delle funzioni che vengono utilizzate nel resto del programma e delle quali bisogna quindi essere a conoscenza prima di procedere con l'analisi. Esse non sono associate ad alcun messaggio, sono delle semplici routine «a chiamata», situate appena dopo la parte dichiarativa, e che sono perciò visibili a tutte le procedure dichiarate successivamente.

```
function Segno(X:Real):ShortInt;
begin
   if X<0 then Segno:=-1 else Segno:=1
end;
```

Segno è una funzione che fornisce 1 quando il parametro *X* è positivo o nullo, -1 quando è negativo. Esso viene utilizzato più che altro nel calcolo della fase, come nel seguente esempio relativo ad uno zero nullo:

```
Fase:=Fase-pi/2*Segno(Zero[N].Re)
```

ArcTg ricava l'argomento di un fasore di coordinate x,y; il suo nome deriva dal fatto che si comporta come l'arcotangente goniometrica (calcolabile con la funzione predefinita del Pascal ArcTan), con la differenza però che la periodicità è 2π anziché di π.

```
function ArcTg(y,x:Real):Real;
begin
   if x>0 then ArcTg:=ArcTan(y/x)
   else if x<0 then ArcTg:=pi+ArcTan(y/x)
   else if y>0 then ArcTg:=pi/2 else if y<0 then ArcTg:=-pi/2 else ArcTg:=0
end;
```

La procedura *u2K* (letto in inglese diventa «μ to k », ossia « daμ a k ») fornisce in uscita su*kRe* e *kIm* la parte reale e la parte immaginaria della costante di guadagno statico ricavate a partire dal coefficiente di guadagno, il quale è memorizzato in modulo (dB) e fase (rad) nelle variabili globali *uM* e *uF*. Il legame fra μ e K è fornito dalla seguente espressione:

$$K = u \cdot \frac{(-P_1)\cdot(-P_2)\cdot\ldots\cdot(-P_N)}{(-Z_1)\cdot(-Z_2)\cdot\ldots\cdot(-Z_N)} = u \cdot \frac{\prod_N (-P_N)}{\prod_N (-Z_N)}$$

Prima di tutto si ricava il modulo del K statico implementando la seguente formula:

$$kM = uM - \sum_N 20 \cdot Log_{10}\sqrt{\mathrm{Re}[Z_N]^2 + \mathrm{Im}[Z_N]^2} + \sum_N 20 \cdot Log_{10}\sqrt{\mathrm{Re}[P_N]^2 + \mathrm{Im}[P_N]^2}$$

La fase è pari a quella di μ, a meno di π nel caso in cui il numero delle radici reali positive sia in numero dispari (le radici complesse sono sempre presenti a coppie coniugate, ed il loro prodotto è sempre positivo).

Infine si ricavano la parte reale e la parte immaginaria trigonometricamente.

Si tenga presente che, delle coppie di radici immaginarie, ne viene memorizzata nella matrice una sola (indifferentemente quella a parte immaginaria positiva o negativa), per cui l'operazione di radice quadrata non viene effettivamente mai utilizzata.

```
procedure u2K(var kRe,kIm:Real);
var    kM,kF:Real;
       N:Integer;

begin
   kM:=uM;
   for N:=1 to NZeri do if Zero[N].Im=0 then
   begin
      if Zero[N].Re<>0 then kM:=kM-20*ln(Abs(Zero[N].Re))/ln(10)
   end
   else kM:=kM-20*ln(Sqr(Zero[N].Re)+Sqr(Zero[N].Im))/ln(10);
   for N:=1 to NPoli do if Polo[N].Im=0 then
   begin
      if Polo[N].Re<>0 then kM:=kM+20*ln(Abs(Polo[N].Re))/ln(10)
   end
   else kM:=kM+20*ln(Sqr(Polo[N].Re)+Sqr(Polo[N].Im))/ln(10);

   kF:=uF;
   for N:=1 to NZeri do if (Zero[N].Im=0) and (Zero[N].Re>0) then kF:=kF-pi;
   for N:=1 to NPoli do if (Polo[N].Im=0) and (Polo[N].Re>0) then kF:=kF+pi;
   kRe:=exp(ln(10)*(kM/20))*cos(kF);
   kIm:=exp(ln(10)*(kM/20))*sin(kF);
   if Abs(kIm)<Abs(kRe)/1e4 then kIm:=0;
   if Abs(kRe)<Abs(kIm)/1e4 then kRe:=0
end;
```

Le ultime due righe annullano completamente *kIm* o *kRe* quando essi assumono valori relativamente troppo piccoli. Questo accorgimento si è reso necessario in quanto altrimenti le approssimazioni non permetterebbero mai di raggiungere lo zero, anche quando si dovrebbe.

K2u si comporta in modo duale rispetto a *u2K*, ricavando dai valori forniti in *kRe* e in *kIm* il modulo *kM* e la fase *kF* corrispondenti, e da questi, tramite l'applicazione della seguente formula, i parametri del coefficiente di guadagno statico, i quali vengono memorizzati nelle due variabili globali *uM* e *uF*.

$$u = K \cdot \frac{(-Z_1)\cdot(-Z_2)\cdot\ldots\cdot(-Z_N)}{(-P_1)\cdot(-P_2)\cdot\ldots\cdot(-P_N)} = K \cdot \frac{\prod_N (-Z_N)}{\prod_N (-P_N)}$$

L'implementazione in Pascal di queste operazioni è del tutto simile a quella ottenuta per u2K, e perciò ci si limita a presentare l'intestazione (si noti che il passaggio è in questo caso per valore, dato che l'uscita avviene direttamente sulle variabili globali):

```
procedure K2u(kRe,kIm:Real);
```

ModuloW è una funzione che calcola il modulo reale del diagramma di Bode ad una certa pulsazione ω, assegnata attraverso il parametro *w*. Si è adottata la massima precisione nei calcoli ricorrendo a variabili di tipo reale Extended (precisione che sarebbe eccessiva per la sola grafica), in quanto si ricorre a questa funzione anche per il calcolo di valori visualizzati numericamente.

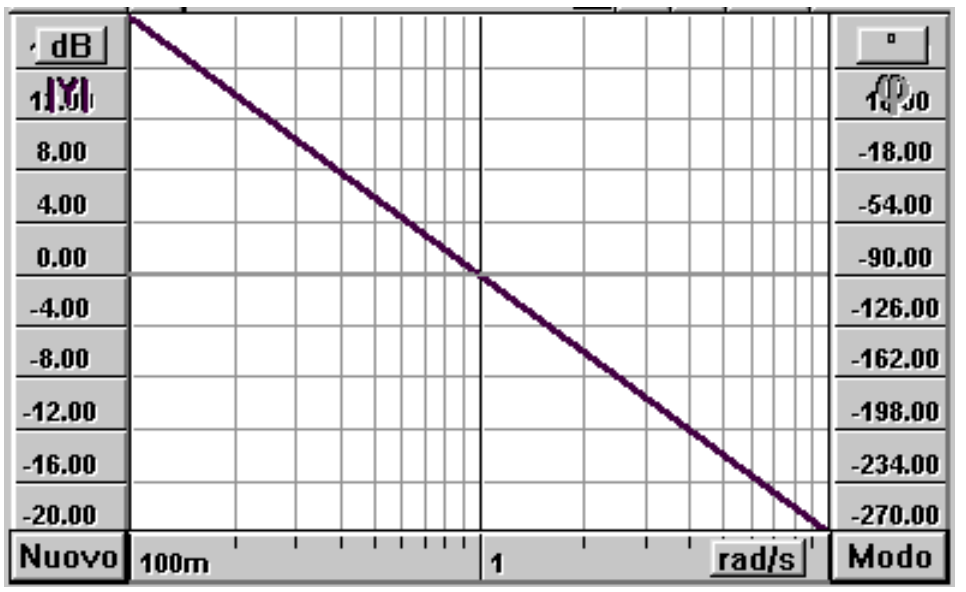
In sostanza, le operazioni effettuate consistono nel sommare ciclicamente i contributi dei singoli poli e zeri per giungere al valore finale del modulo.

CONTRIBUTI DELLE RADICI

Si prendono con il segno positivo se la radice è uno zero, negativo se è un polo (R è il valore della radice).

RADICE REALE NELL'ORIGINE:

$$M(w) = \pm 20 \cdot Log_{10}(w)$$



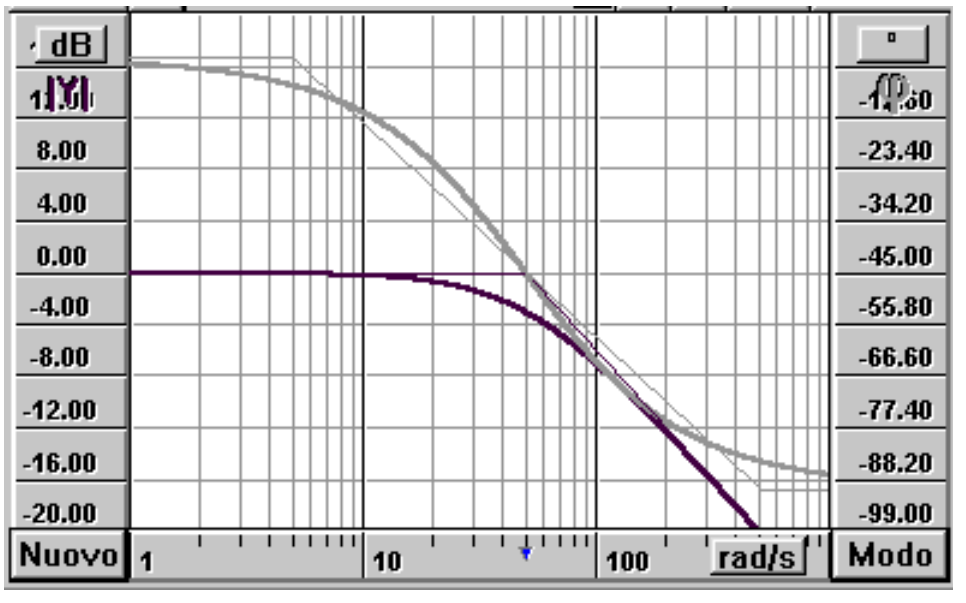
Esempio di polo nell'origine realizzato con WinBode.

Il diagramma del modulo è rappresentato dalla linea più scura e dal righello sinistro, mentre la fase è associata alla linea orizzontale più chiara allineata al valore -90° del righello destro.

NOTA: Nei grafici seguenti la linea spessa rappresenta l'andamento reale del modulo o della fase e la linea sottile è il corrispondente diagramma asintotico.

RADICE REALE:

$$M(w) = \pm 20 \cdot Log_{10}\sqrt{1 + \frac{w^2}{\mathrm{Re}[R]^2}} = \pm 10 \cdot Log_{10}(1 + \frac{w^2}{\mathrm{Re}[R]^2})$$



Esempio di diagramma di Bode con:

m = 1

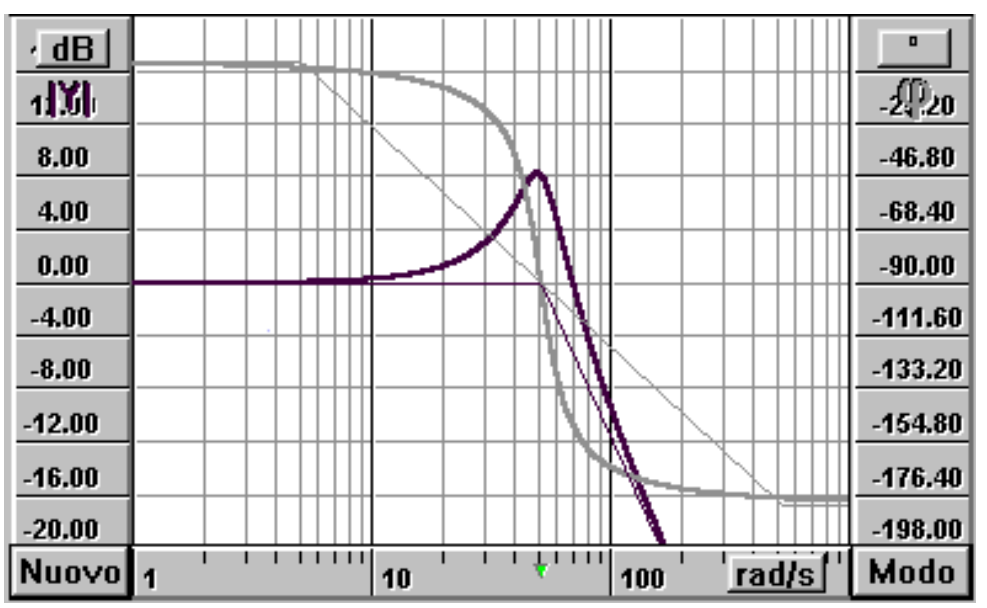
polo = -50

RADICE IMMAGINARIA:
Si definisce prima il parametro pulsazione naturale Wn come:

$$Wn = \sqrt{\mathrm{Re}[R]^2 + \mathrm{Im}[R]^2}$$

il quale viene poi utilizzato nella seguente:

$$M(w) = \pm 20 \cdot Log_{10} \sqrt{(1 - \frac{w^2}{Wn^2})^2 + \frac{4 \cdot w^2 \cdot \mathrm{Re}[R]^2}{Wn^4}}$$



Esempio di diagramma di Bode con:

m = 1

poli = -10 ± j50

Wn ≅ 51

A questo punto è possibile prendere in considerazione l'algoritmo in Pascal:

```
function ModuloW(w:Extended):Extended;
var    Wn:Extended;
       N:Integer;
       Modulo:Extended;

begin
   Modulo:=uM;
   for N:=1 to NPoli do
   begin
      if Polo[N].Im=0 then
         if Polo[N].Re<>0 then Modulo:=Modulo-10*ln(1+Sqr(w)/Sqr(Polo[N].Re))/ln(10)
         else Modulo:=Modulo-10*ln(Sqr(w))/ln(10)
      else
      begin
         WN:=Sqr(Polo[N].Re)+Sqr(Polo[N].Im);
         if (Sqr(w)<>Wn) or (Polo[N].Re<>0) then
         Modulo:=Modulo-20*ln(Sqrt(Sqr(1-Sqr(w)/WN)+4*Sqr(w)*Sqr(Polo[N].Re)/Sqr(WN)))/ln(10)
      end
   end;

   for N:=1 to NZeri do
   begin
      if Zero[N].Im=0 then
         if Zero[N].Re<>0 then Modulo:=Modulo+10*ln(1+Sqr(w)/Sqr(Zero[N].Re))/ln(10)
         else Modulo:=Modulo+10*ln(Sqr(w))/ln(10)
      else
      begin
         WN:=Sqr(Zero[N].Re)+Sqr(Zero[N].Im);
         if (Sqr(w)<>Wn) or (Zero[N].Re<>0) then
         Modulo:=Modulo+20*ln(Sqrt(Sqr(1-Sqr(w)/WN)+4*Sqr(w)*Sqr(Zero[N].Re)/Sqr(WN)))/ln(10)
      end
   end;
   ModuloW:=Modulo
end;
```

FaseW è una funzione che calcola la fase reale del diagramma di Bode ad una certa pulsazione ω assegnata attraverso il parametro *w*. Per *FaseW* valgono le stesse considerazioni effettuate precedentemente per *ModuloW*, in particolare per quanto riguarda il metodo di calcolo, basato sulla somma degli sfasamenti in radianti introdotti dai vari poli e dalle radici. Le formule diventano le seguenti:

CONTRIBUTI DELLE RADICI

Si prendono con il segno positivo se la radice è uno zero, negativo se è un polo.
Per quanto riguarda i diagrammi corrispondenti alle tre tipologie definite, vedere le figure precedenti.

RADICE REALE NELL'ORIGINE:

$$F(w) = \pm\frac{p}{2}$$

RADICE REALE:

$$F(w) = \pm ArcTan\left(\frac{w}{\mathrm{Re}[-R]}\right)$$

RADICE IMMAGINARIA:

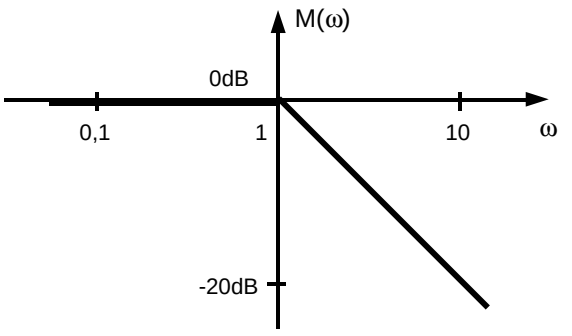
$$F(w) = \pm\left[ArcTan\left(\frac{1-\frac{w^2}{Wn^2}}{\frac{2\cdot w\cdot \mathrm{Re}[R]}{Wn^2}}\right) - \frac{p}{2} Segno(\mathrm{Re}[R])\right]$$

Il tutto viene ottenuto con la seguente funzione:

```
function FaseW(w:Extended):Extended;
var    Wn:Extended;
       N:Integer;
       Fase:Extended;

begin
   Fase:=uF;
   for N:=1 to NPoli do
   begin
      if Polo[N].Im=0 then
        if Polo[N].Re<>0 then Fase:=Fase-arctan(w/-Polo[N].Re) else Fase:=Fase-pi/2
      else
      begin
         WN:=Sqr(Polo[N].Re)+Sqr(Polo[N].Im);
         if Polo[N].Re<>0 then
           Fase:=Fase+pi/2*Segno(Polo[N].Re)-arctan((1-Sqr(w)/Wn)/(2*w*Polo[N].Re/Wn))
         else if Sqr(w)>Wn then Fase:=Fase-pi
      end
   end;

   for N:=1 to NZeri do
   begin
      if Zero[N].Im=0 then
        if Zero[N].Re<>0 then Fase:=Fase+arctan(w/-Zero[N].Re) else Fase:=Fase+pi/2
      else
      begin
         WN:=Sqr(Zero[N].Re)+Sqr(Zero[N].Im);
         if Zero[N].Re<>0 then
           Fase:=Fase-pi/2*Segno(Zero[N].Re)+arctan((1-Sqr(w)/Wn)/(2*w*Zero[N].Re/Wn))
         else if Sqr(w)>Wn then Fase:=Fase+pi
      end
   end;
   FaseW:=Fase
end;
```

ModuloAsW calcola il valore del modulo che si otterrebbe dal diagramma di Bode **asintotico** alla pulsazione fornita in *w*. Invece di calcolare anche i diagrammi asintotici punto per punto, si sarebbe potuto effettuare i calcoli solo per i punti di cambiamento della pendenza e congiungerli con linee rette, con conseguente notevoleincremento di velocità. Si è scelta l'altra strada per due motivi:

- i tempi rimangono comunque contenuti;
- la gestione grafica per il diagramma reale e quello asintotico rimane la stessa.

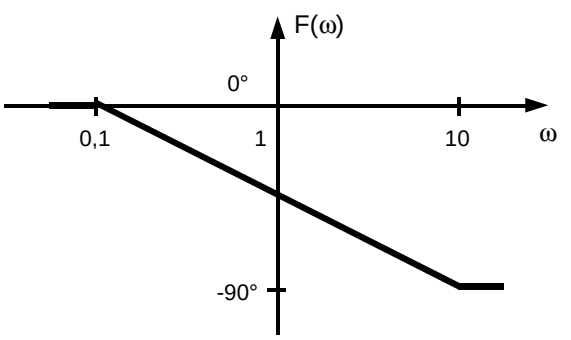


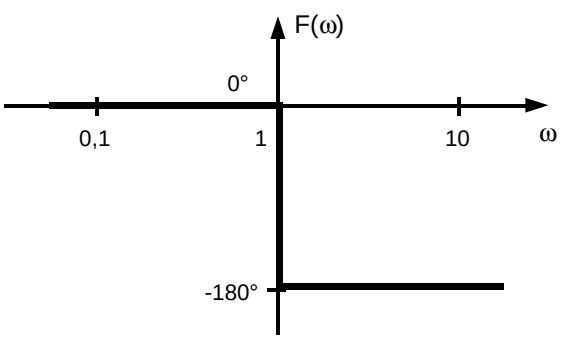
Il procedimento adottato è molto simile a quello visto per le due funzioni precedenti: esso si basa ancora sulla somma dei singoli contributi. Come si vede chiaramente dalla figura a fianco, riferita ad un polo reale pari a ±1, tali contributi valgono 0db per pulsazioni minori di quella corrispondente alla radice, mentre aumentano (zero) o diminuiscono (polo) con pendenza di 20dB/dec a pulsazioni maggiori.

```
function ModuloAsW(w:Extended):Extended;
var    Wn:Extended;
       N:Integer;
       Modulo:Extended;

begin
   Modulo:=uM;
   for N:=1 to NPoli do
   begin
      Wn:=Sqrt(Sqr(Polo[N].Re)+Sqr(Polo[N].Im));
      if Wn<>0 then
      begin
         if w>Wn then if Polo[N].Im=0 then Modulo:=Modulo-20*ln(w/Wn)/ln(10)
                      else Modulo:=Modulo-40*ln(w/Wn)/ln(10)
      end
      else Modulo:=Modulo-20*ln(w)/ln(10)
   end;

   for N:=1 to NZeri do
   begin
      Wn:=Sqrt(Sqr(Zero[N].Re)+Sqr(Zero[N].Im));
      if Wn<>0 then
      begin
         if w>Wn then if Zero[N].Im=0 then Modulo:=Modulo+20*ln(w/Wn)/ln(10)
                      else Modulo:=Modulo+40*ln(w/Wn)/ln(10)
      end
      else Modulo:=Modulo+20*ln(w)/ln(10)
   end;
   ModuloAsW:=Modulo
end;
```

Se la radice è reale, pendenza 20dB/dec, se è immaginaria, essa rappresenta una coppia di radici e perciò la pendenza va raddoppiata (40dB/dec).

FaseAsW calcola il valore della fase che si otterrebbe dal diagramma di Bode asintotico alla pulsazionefornita in w. La convenzione adottata è quella di far rimanere costante la fase per pulsazioni minori ad 1/10 e maggiori a 10 volte la pulsazione della radice, e di farla variare linearmente (su scala logaritmica) all'interno di tale intervallo; quando la radice è immaginaria ed è **pura** (parte reale nulla) si compie invece un salto di 180°, come avviene nel diagramma reale, in modo che la fase asintotica non si discosti troppo da quella reale.





Le figure riportate alla pagina precedente chiariscono come la funzione (riportata di seguito), risponda in presenza rispettivamente di un polo reale di valore -1 e di una coppia di poli immaginari puri di valore ±j1; se al posto dei poli si hanno degli zeri, gli andamenti sono simmetrici rispetto all'asse delle ascisse.

```
function FaseAsW(w:Extended):Extended;
var    Wn:Extended;
       N:Integer;
       Fase:Extended;

begin
   Fase:=uF;
   for N:=1 to NPoli do
   begin
      Wn:=Sqrt(Sqr(Polo[N].Re)+Sqr(Polo[N].Im));
      if Wn<>0 then
      begin
         if Polo[N].Im=0 then
         begin
            if w>Wn*10 then Fase:=Fase+pi/2*Segno(Polo[N].Re)
              else if w>Wn/10 then Fase:=Fase+pi/4*ln(w/Wn*10)/ln(10)*Segno(Polo[N].Re)
         end
         else if Polo[N].Re<>0 then
         begin
            if w>Wn*10 then Fase:=Fase+pi*Segno(Polo[N].Re)
              else if w>Wn/10 then Fase:=Fase+pi/2*ln(w/Wn*10)/ln(10)*Segno(Polo[N].Re)
         end
         else if w>Wn then Fase:=Fase-pi
      end
      else Fase:=Fase-pi/2
   end;

   for N:=1 to NZeri do
   begin
      Wn:=Sqrt(Sqr(Zero[N].Re)+Sqr(Zero[N].Im));
      if Wn<>0 then
      begin
         if Zero[N].Im=0 then
         begin
            if w>Wn*10 then Fase:=Fase-pi/2*Segno(Zero[N].Re)
              else if w>Wn/10 then Fase:=Fase-pi/4*ln(w/Wn*10)/ln(10)*Segno(Zero[N].Re)
         end
         else if Zero[N].Re<>0 then
         begin
            if w>Wn*10 then Fase:=Fase-pi*Segno(Zero[N].Re)
              else if w>Wn/10 then Fase:=Fase-pi/2*ln(w/Wn*10)/ln(10)*Segno(Zero[N].Re)
         end
         else if w>Wn then Fase:=Fase+pi
      end
      else Fase:=Fase+pi/2
   end;
   FaseAsW:=Fase
end;
```

Polo nell'origine → Fase costante pari a -π/2

Zero nell'origine → Fase costante pari a π/2

La procedura ***Equazione*** permette il calcolo delle radici di un'equazione di grado qualunque (teoricamente) della quale siano noti i coefficienti. E' stata ricavata a partire da un programma per il calcolo delle soluzioni di un'equazione riportato su di un manuale. Esso era originariamente in linguaggio BASIC, è stato perciò convertito in Pascal e trasformato in procedura.

Equazione riceve in ingresso il grado ***n*** dell'equazione da calcolare e i coefficienti (in ordine, da quello di x^n al termine noto) attraverso l'array "*a*", e fornisce le soluzioni nell'array "*pz*".

Si riporta la sola intestazione:

```
type   vettore1=array[1..25] of Real;
       vettore2=array[1..25] of complex;

procedure Equazione(var a:vettore1; var pz:vettore2; n:integer);
```

NOTA: Per equazioni di grado superiore al 2° le soluzioni sono approssimate (l'approssimazione è comunque tale da non alterare i risultati, e può essere ulteriormente ridotta modificando una costante locale).

Ora che è stato portato a termine l'esame delle routine di calcolo, si ritorna alle procedure dell'oggetto *TBode*:

AddZero e *AddPolo* sono procedure che gestiscono l'aggiunta di una nuova radice. Esse hanno la stessa struttura, l'unica differenza è che una agisce sugli zeri e l'altra sui poli; per questo motivo l'analisi verrà condotta solo per *AddZero*.

La parte dichiarativa è:

```
procedure AddZero(var Msg: TMessage); virtual id_First + 101;
```

e quindi essa viene attivata ogni volta che dalla coda dei messaggi esce *id_First+101* (i messaggi sono sostanzialmente dei codici Word ai quali però vengono attribuite delle costanti mnemoniche); *id_First* (=32768) corrisponde all'offset che viene aggiunto agli identificatori di pulsante per distinguerli, ad esempio, da quelli di menu, per i quali si utilizza *cm_First* (=40960), o dai messaggi del Windows veri e propri (*wm_First*=0). In questo modo è possibile adottare lo stesso identificatore per una voce del menu o per un pulsante senza incorrere in chiamate reciproche.

Dato ora che l'Id 101 è stato assegnato, in fase di inizializzazione, al pulsante *ZeroBtn* (v. *TBode.Init*) con l'assegnazione:

```
ZeroBtn := New(PButton, Init(@Self, 101, 'Zero', 0, 0, 50, 23, False));
```

si conclude che, ogni volta che si preme il pulsante ZeroBtn, il quale aggiunge alla coda dei messaggi la Word *id_First+101*, viene eseguita la procedura *AddZero*, di seguito riportata e commentata:

```
procedure TBode.AddZero;
var    NuovoVal:Real;
       ErrorPos:Integer;
       KRe,KIm:Real;

procedure Check;
begin
   if Errorpos<>0 then MessageBox(HWindow, 'Inserire un numero reale', 'Dato errato', mb_Ok)
end;

begin
   if KStatico then u2K(KRe,KIm);
   NZeri:=NZeri+1;

   repeat {parte reale}
      if Application^.ExecDialog(New(PInputDialog,
              Init(@Self, 'Zero','Parte reale dello zero',InText, SizeOf(InText)))) = id_OK
        then Val(InText, NuovoVal, ErrorPos)
      else
      begin
         NZeri:=NZeri-1;
         Exit
      end;
   Check
   until ErrorPos = 0;
   Zero[NZeri].Re:=NuovoVal;

   repeat {parte immaginaria}
      if Application^.ExecDialog(New(PInputDialog,
             Init(@Self, 'Zero','Parte immaginaria dello zero',InText, SizeOf(InText)))) = id_OK
        then Val(InText, NuovoVal, ErrorPos)
      else
      begin
         NZeri:=NZeri-1;
         Exit
      end;
   Check
   until ErrorPos = 0;
   Zero[NZeri].Im:=NuovoVal;
   if KStatico then K2u(KRe,KIm);
   InvalidateRect(HWindow,nil,True)
end;
```

La procedura *Check* visualizza un messaggio d'errore se *ErrorPos* non è nullo.

Se *KStatico*=True, e cioe se il pulsante *uKBtn* si trova su 'K', si deve memorizzare il valore del guadagno e ricalcolarlo dopo l'aggiunta dello zero (v. penultima riga) in modo da non alterare K.

Val restituisce in *ErrorPos* la posizione in cui si è riscontrato un errore (0 se il dato è corretto).

Aggiorna i diagrammi con il nuovo valore.

NOTA: *InvalidateRect* assegna una zona dell'area utente che deve essere ridisegnata, e forza quindi l'invio del messaggio *WM_Paint*. Il secondo parametro è un puntatore ed una variabile *TRect* contenente le coordinate dell'area da ridisegnare. Se tele puntatore viene sostituito con **nil** viene ridisegnata l'intera finestra.

AddZeroRe e *AddPoloRe* sono versioni ridotte di *AddZero* e *AddPolo*, le quali chiedono l'introduzione della sola parte reale.

AddZeroAcc e *AddPoloAcc* sono associate ai messaggi *cm_first + 888* e *cm_first + 889*, corrispondenti ai tasti di scelta rapida 'z' e 'p' (si noti che essi vengono considerati dal Pascal come dei comandi del menu, da cui l'utilizzo di *cm_first*). Il loro unico scopo è quello di richiamare rispettivamente *AddZeroRe* e *AddPoloRe*.

DelZero e *DelPolo* vengono eseguite in corrispondenza alla pressione dei pulsanti con la X, i quali sono attivi solo quando una radice è stata selezionata. Il loro scopo è di eliminare tale radice (o coppia di radici se è immaginaria).

```
procedure TBode.DelZero;
var    N,Count:Integer;
       KRe,KIm:Real;

begin
   if KStatico then u2K(KRe,KIm);
   Count:=CBZeri^.GetSelIndex;
   N:=0;
   repeat
      Inc(N);
      if Zero[N].Im=0 then Dec(Count) else Dec(Count,2)
   until count<0;
   Dec(NZeri);
   repeat
      Zero[N].Re:=Zero[N+1].Re;
      Zero[N].Im:=Zero[N+1].Im;
      Inc(N)
   until N>NZeri;
   if KStatico then K2u(KRe,KIm);
   InvalidateRect(HWindow,nil,True);
   CBZeri^.SetSelIndex(-1);
   SendMessage(HWindow,WM_Command,0,makelong(0,CBN_DROPDOWN))
end;
```

Riceve da *GetSelIndex* il numero ordinale della voce selezionata e calcola il corrispondente N per la matrice degli zeri.

Trasla tutti gli zeri successivi a quello eliminato in dietro di una posizione.

Annulla la selezione e chiede l'aggiornamento delle due liste dei poli e degli zeri.

Anche *EditZero* ed *EditPolo* agiscono sulla selezione, modificando il valore della radice. Queste due procedure sono associate ai pulsanti con la 'C' (Cambia) sulla barra degli strumenti.

GrafoMode è collegato al pulsante 'Modo' in basso a destra, alla pressione del quale le due variabili booleane *Asintotico* e *Reale* vengono portate ciclicamente agli stati (False,True)-(True,False)-(True,True) tramite la struttura condizionale:


```
if Asintotico and Reale then Asintotico:=False else
if Asintotico and not Reale then Reale:=True else
if not Asintotico and Reale then
begin
   Asintotico:=True;
   Reale:=False
end;
```

Viene inoltre riposizionato il simbolo di selezione per il menu.

Nuovo è invece collegato al pulsante a sinistra ed il suo compito è quello di azzerare tutti i poli e gli zeri (semplicemente annullando i valori dei contatori *NPoli* e *NZeri*) e portare ad 1 il coefficiente di guadagno.

ToggleuK è una procedura che viene attivata dal pulsante *uKBtn* e che risponde modificando il valore visualizzato nel pulsante stesso (u/K) e aggiornando il testo nella finestrella *uKStat* con i corrispondenti valori (parte reale e parte immaginaria) di μ o di K.

Num e *Den* permettono l'inserimento dei coefficienti di un polinomio, le cui radici (calcolate con la procedura *Equazione*) saranno aggiunte rispettivamente agli zeri o ai poli esistenti. Le operazioni effettuate da *Num* sono (il funzionamento di *Den* è del tutto simile):

- chiedere all'utente il grado (memorizzato in *Count*) del termine da aggiungere e i coefficienti, memorizzandoli nell'array *Eq* a partire da quello di grado massimo (quindi il termine noto si troverà in Eq[Count+1]);

- provvedere ad eliminare, con il seguente ciclo, i coefficienti di maggior peso se sono nulli, in modo da inviare ad *Equazione* un polinomio equivalente di grado minore:

```
while Eq[1]=0 do
begin
   if Count=0 then Exit;
   Dec(Count);
   for n:=1 to Count do Eq[n]:=Eq[n+1]
end;
```

- correggere il coefficiente di guadagno statico se il coefficiente di peso maggiore è diverso da 1:

```
uM:=uM+20*ln(Abs(Eq[1]))/ln(10);
if Eq[1]<0 then uF:=uF+pi;
```

- se quello che rimane del polinomio ha ancora soluzioni, eseguire *Equazione* ed aggiornare correttamente, con le soluzioni ottenute, la matrice contenente gli Zeri:

```
if Count<>0 then
begin
   Equazione(Eq,So,Count);
   Repeat
      Inc(NZeri);
      Zero[NZeri]:=So[Count];
      if So[Count].Im<>0 then Dec(Count);
      Dec(Count)
   until Count=0
end;
```

La procedura *WMCommand* viene in genere ereditata senza modifiche dall'oggetto *TWindow*, il quale lo eredita a sua volta da *TWindowObject*. Essa processa tutti i messaggi provenienti dal menu, dagli oggetti di controllo (pulsanti, liste, barre di scorrimento, ecc.) e dai tasti di accelerazione. E' quindi possibile, sostituendola con una proprietaria, intercettare uno di tali messaggi.

Avendo bisogno, nel nostro caso, di controllare quando viene espansa una delle liste dei poli o degli zeri (comando *CBN_DropDown*), in modo da aggiornare le liste stesse con i valori contenuti nelle matrici *Poli* e *Zeri*, si è adottata tele sostituzione. E' ovvio che la nuova procedura deve chiamare quella ereditata in modo da non ostacolare la gestione dei comandi. Si verifica poi che il comando che ha attivato *WMCommand* sia effettivamente quello desiderato con la condizione (il record Msg contiene informazioni riguardo a quale comando è stato inviato e da quale oggetto):

```
if HiWord(Msg.LParam)=CBN_DROPDOWN then
```

Se la condizione è verificata si aggiornano le liste.

A questo punto può essere interessante osservare come, in questa ed altre procedure, un valore numerico complesso (nell'esempio *Zero[N]*) venga convertito in una stringa in notazione esponenziale (in *Testo*):

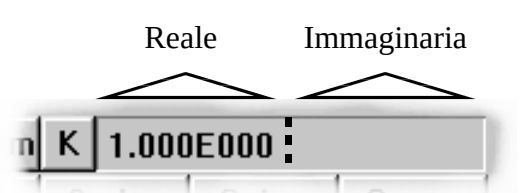
```
if Zero[N].Re<>0 then
begin
   Mult:=Trunc(ln(Abs(Zero[N].Re))/ln(10));       { N,Mult:Integer }
   Str(Zero[N].Re/exp(Mult*ln(10)):6:3,Testo);    { Temp,Testo:Array [0..50] of Char }
   wvsprintf(Temp,'E%03i',Mult);
   StrCat(Testo,Temp)
end;
```

```
If Zero[N].Im<>0 then
begin
   Mult:=Trunc(ln(Abs(Zero[N].Im))/ln(10));
   Str(Abs(Zero[N].Im)/exp(Mult*ln(10)):0:3,Temp);
   wvsprintf(Temp2,'E%03i',Mult);                { Temp2,Testo2:Array [0..50] of Char }
   StrCat(Temp,Temp2);
   StrCopy(Testo2,Testo);
   StrCat(Testo2,'+j');
   StrCat(Testo2,Temp);
   CBZeri^.AddString(Testo2);
   StrCat(Testo,' -j');
   StrCat(Testo,Temp)
end;
```

WMLButtonDown, essendo dichiarata nel seguente modo:

```
procedure WMLButtonDown(var Msg: TMessage); virtual wm_First + wm_LButtonDown;
```

viene attivata dalla pressione del tasto sinistro del mouse, alla quale corrisponde l'invio da parte del sistema operativo del messaggio *wm_LButtonDown*. Se le coordinate alle quali si trova il puntatore, ricevute sempre attraverso *Msg*, sono contenute nell'area di inserimento del coefficiente o della costante di guadagno (*uKStat*), allora, a seconda se ci si trova nella metà sinistra o in quella destra, viene richiesta all'utente la parte reale o quella immaginaria del guadagno:



GESTIONE DEL MENU

Tutte le procedure il cui nome comincia con le lettere 'CM' sono dichiarate in modo tale che venganorichiamate dalle corrispondenti voci del menu.

Le procedure relative al menu qui a fianco sono:

CMReale
CMAsintotico
CMEntrambi

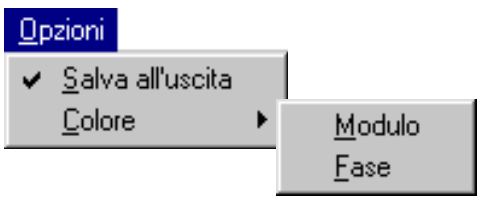
CMOrdinateOnOff
CMAscisseOnOff
CMAssiCriticiOnOff

CMStandard
CMAvanzate

Questo primo gruppo è formato da procedure che hanno un funzionamento molto simile: esse agiscono sulle variabili booleane di configurazione (Reale, Asintotico, Ordinate, Ascisse, AssiCritici, Standard, Avanzate) e aggiornano lo stato di selezione (√) delle voci del menu attraverso comandi del tipo:

```
CheckMenuItem(GetMenu(HWindow),1001,mf_ByCommand or MF_UnChecked);
```

CMAzzeraZeri annulla tutti gli zeri lasciando inalterati i poli ed il coefficiente diguadagno, *CMAzzeraPoli* fa lo stesso per i poli, *CMAzzeraTutto* annulla tutte le variabili, allo stesso modo della procedura *Nuovo* vista precedentemente.



CMSalva attiva/disattiva il salvataggio automatico all'uscita della configurazione attuale (dimensioni della finestra, colori delle linee, ecc.). Con l'ultima condizione si è fatto in modo che all'avviosuccessivo l'autosalvataggio rimanga disattivato.

```
procedure TBode.CMSalva;
begin
   if not Salva then
   begin
      Salva:=True;
      CheckMenuItem(GetMenu(HWindow),3101,mf_ByCommand or MF_Checked)
   end
   else
   begin
      Salva:=False;
      CheckMenuItem(GetMenu(HWindow),3101,mf_ByCommand or MF_UnChecked);
      if GetPrivateProfileInt('WinBode','Salva',0,'WinBode.Ini')=1 then
        WritePrivateProfileString('WinBode','Salva','0','WinBode.Ini')
   end;
end;
```

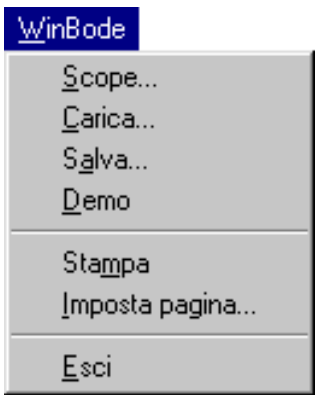
CMColoreM e *CMColoreF* aprono le finestre che permettono la selezione dei colori per i diagrammi del modulo e della fase. Al riguardo, si tenga presente che la gestione di tali finestre è quasi del tutto automatica: è sufficiente configurare un record di tipo*TChooseColor* con informazioni riguardanti il tipo di finestra desiderata, quali pulsanti od opzioni abilitare, ecc. ed assegnarlo come parametro alla funzione *ChooseColor*. Provvede poi il sistema operativo a creare e gestire la finestra, seguendo sia le specifiche dell'utente che i limiti imposti dalla configurazione video (risoluzione, numero dei colori...).

Altri esempi di finestre che godono di questo supporto sono quelle per la selezione del tipo di carattere e per l'apertura od il salvataggio di file. Esse vengono chiamate, nel Pascal ed in altri linguaggi per il Windows, «Common Dialogs».

CMAiuto e *CMAutore* visualizzano due finestre di dialogo contenenti informazioni sul programma e sull'autore. La struttura di tali finestre è definita nel file delle risorseWinBode.Res, un file che può contenere dati multiuso (icone, cursori, menu, ecc.) e che viene compilato assieme al programma ed aggiunto in coda all'eseguibile.

L'help in linea è stato organizzato sotto forma di finestra anziché di file ipertestuale come di consueto, oltre che per la semplicità intrinseca, per i seguenti motivi:

- non è necessario alcun file aggiuntivo (.hlp), con notevoli vantaggi dal punto di vista dell'installazione;
- si è adottata una struttura schematica che permettesse una rapida consultazione;
- essendo ridotte al minimo le istruzioni, è più probabile che l'utente si soffermi su tutti i punti, mentre in un file di help esteso è più facile perdere qualche argomento.



Per il menu a fianco valgono le seguenti associazioni:

CMScope
CMFileCarica
CMFileSalva
CMDemo

CMFilePrint
CMFileSetup

CMFileEsci

CMDemo crea una nuova funzione casuale con tre poli e tre zeri.

CMFilePrint e *CMFileSetup* gestiscono la stampa (si vedrà più avanti come) e la configurazione della stampante.

CMFileEsci causa con l'istruzione:

```
DestroyWindow(HWindow);
```

l'uscita dall'applicazione (preceduta dall'esecuzione di *TBode.Done*).

CMFileSalva salva la funzione corrente in un file tramite l'apertura di una «Common Dialog» (per la selezione del file il cui nome viene restituito in *FileName*) e termina nel seguente modo:

```
Assign(Out,FileName);                    { Out:File }
reset(Out,1);                            { FileName:array[0..200] of Char }
if FileSize(out)>0 then
  if MessageBox(HWindow, 'File esistente. Sovrascrivere il file?','Salva dati', mb_YesNo)=id_no then
  begin
     Close(Out);
     Exit;
  end;
Close(Out);
rewrite(Out,1);
```

Verifica che il file assegnato non contenga già dei dati, altrimenti chide se si desidera sovrascriverli.

TestLine è una stringa che viene salvata all'inizio del file in modo tale che, quando si carica, tramite un'operazione di confronto, WinBode può accertare che il file sia effettivamente compatibile.

```
BlockWrite(Out,TestLine,156,Ris);        { Ris:Word }
BlockWrite(Out,uM,7,Ris);
BlockWrite(Out,uF,7,Ris);
BlockWrite(Out,NZeri,2,Ris);
BlockWrite(Out,Zero,20*NZeri,Ris);
BlockWrite(Out,NPoli,2,Ris);
BlockWrite(Out,Polo,20*NPoli,Ris);
Close(Out);
```

Salva:	il coefficiente di guadagno statico
	il numero degli zeri e gli zeri stessi
	il numero dei poli e la matrice dei poli

CMFileCarica agisce in modo duale eseguendo le seguenti operazioni:

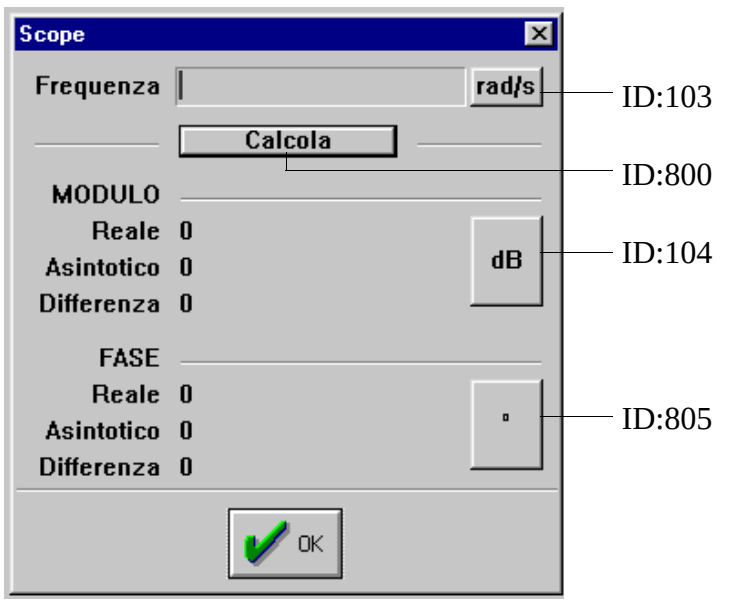
- chiamata di una «Common Dialog» per la selezione del file
- verifica della coincidenza dell'header con quello salvato
- lettura del guadagno
- lettura del numero degli zeri e degli zeri stessi
- lettura del numero dei poli e dei poli stessi

CaricaB, *SalvaB*, e *ScopeB* sono collegate ai tre pulsanti omonimi della barra degli strumenti e richiamano rispettivamente le procedure *CMFileCarica* e *CMFileSalva* appena viste e *CMscope.*

FINESTRA «SCOPE»

In definitiva, premendo il pulsante Scope o selezionando la voce corrispondente del menu, si causal'esecuzione della seguente procedura, la quale provvede ad inizializzare e gestire l'oggetto *TScopeDialog.*

```
procedure TBode.CMScope;
var
   D: TScopeDialog;
begin
   D.Init(@Self, 'Scope');
   D.Execute;
   D.Done
end;
```



In particolare, la riga *Init(@Self, 'Scope')* associa a *TScopeDialog* la risorsa di nome *Scope*, situata nel solito file WinBode.Res, la quale determina il «layout» della finestra di dialogo da creare, ossia la disposizione e le proprietà dei vari pulsanti ed oggetti che la costituiscono.

NOTA: Le risorse possono essere create con un apposito programma (Resource Workshop) fornito assieme al Borland Pascal. Esso mette a disposizione del programmatore una serie di strumenti per creare immagini, icone, gruppi di dati, menu, ecc. che sono appunto le risorse, e che possono essere compilate in un file associabile ai programmi Pascal tramite direttive del tipo:

```
{$R WinBode.Res}
```

Per quanto riguarda le finestre di dialogo, il generatore di risorse permette la definizione del layout per via grafica, e cioè posizionando i vari elementi che la compongono tramite il mouse, a differenza ad esempio della barra degli strumenti, per la quale si sono inseriti direttamente nel codice sorgente le coordinate e le dimensioni dei pulsanti (v. *TBode.Init*). Risulta quindi possibile, con il compilatore di risorse, creare in modo immediato finestre come quelle dell'Help e dell'autore di WinBode. Tuttavia, finché è sufficiente la visualizzazione e la gestione di qualche pulsante di uso generale, avviene tutto in modo automatico, quando invece si vuole l'esecuzione di operazioni specifiche, è necessario creare un oggetto che prenda il controllo della finestra. A grandi linee è proprio come se si avesse l'esecuzione di un'applicazione nell'applicazione, ed al riguardo si noti la somiglianza fra *CMScope* e le righe costituenti il programma principale:

```
var
   MyApp: TMyApplication;

begin
   MyApp.Init('Win Bode (Ver. Didattica)');
   MyApp.Run;
   MyApp.Done
end.
```

nonché fra la parte dichiarativa di *TBode* e quella di *TScopeDialog* (l'oggetto in questione):

```
type   TScopeDialog = object(TDialog)
          procedure Calcola(var Message: TMessage); virtual id_First + 800;
          procedure RadHz(var Msg: TMessage); virtual id_First + 103;
          procedure dBMod(var Msg: TMessage); virtual id_First + 104;
          procedure RadDeg(var Msg: TMessage); virtual id_First + 805
       end;
```

i codici di identificazione sono ora relativi a quattro dei pulsanti della finestra (v. figura), come in TBode avveniva per il collegamento fra le procedure ed i pulsanti dell'applicazione.

Senza scendere troppo nei particolari, si sappia solo che le procedure di *TScopeDialog* sfruttano comandi dedicati per la lettura della pulsazione inserita e per modificare i 6 valori d'uscita, mentre richiamano le funzioni *ModuloW*, *ModuloAsW*, *FaseW*, *FaseAsW* per effettuare i calcoli.

GESTIONE DELLA GRAFICA

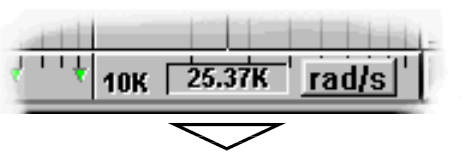
I rimanenti oggetti hanno tutti qualcosa a che fare con la grafica, e costituiscono, insieme, più della metà del codice sorgente. Essi sono:

TGraficaHT	ottimizzatore grafico
TScivi	visualizzatore comune dei valori
TGraf	finestra del grafico
TAsseX	righello della pulsazione
TAsseY	righello del modulo
TAsseF	righello della fase

TGraficaHT è un oggetto derivato da *TWindow* che aggiunge a quest'ultimo una maggior potenzialità grafica. In particolare fa in modo che l'utente non si accorga del refresh delle finestre: per le finestre di tipo *TWindow*, ogni volta che l'area grafica deve essere ridisegnata, si provvede prima a cancellarla completamente, dopodiché vi si disegna sopra; ciò comporta tuttavia due problemi:

- si nota un effetto di sfarfallio;
- si vede l'area grafica mentre viene ridisegnata;

questi difetti sono tanto più evidenti quanto meno potente è l'hardware utilizzato. Invece TGraficaHT effettua le operazioni di cancellazione e tracciatura in memoria, e solo quando sono state completate sovrascrive l'area grafica interessata con l'immagine in memoria.



Finestra TScrivi

TScrivi è un oggetto utilizzato in comune dai tre righelli per visualizzare la finestrella con il valore corrispondente alla posizione attuale del cursore nel grafico. Per ottenere tale risultato è quindi sufficiente modificare la variabile *Valore* (v. parte dichiarativa) e riposizionare la finestra nella posizione desiderata.

La procedura *TScrivi.Paint* provvede poi a tracciare le quattro linee di ombreggiatura e a stampare, in posizione centrata rispetto alla finestra, il contenuto di *Valore* (nel formato 'numero-moltiplicatore').

```
PScrivi = ^TScrivi;
TScrivi = object(TGraficaHT)
   Valore  : Real;
   constructor Init(AParent:PWindowsObject);
   procedure Paint(PDC: HDC; var PaintInfo: TPaintStruct); virtual
end;
```

Si noti che *TScrivi* è di tipo *TGraficaHT*. Se si fosse ricorsi a *TWindows*, si sarebbe avuto un effetto trasparenza durante gli spostamenti, per il fatto che viene disegnato prima lo sfondo e poi sopra la finestra di *TScrivi*.

La conversione di un valore numerico reale in una stringa di testo nel formato 'numero-moltiplicatore', e cioè, ad esempio 12.56K al posto di 12560, si ottiene con i seguenti passi:

- calcolo dell'ordine di grandezza di *Valore*;
- conversione della sola parte significativa in stringa;
- modifica dell'ultimo carattere di Stringa sfruttando il seguente array, che è definito globalmente:

```
const  Moltiplicatore:array[0..8] of Char = 'pnµm KMGT';
```

questo interessante metodo, che permette di ottenere in modo semplicissimo un formato sicuramente più compatto e più leggibile di qualunque altro, è implementato in Pascal da poche righe:

```
StrCopy(Stringa,'      ');                                  { Stringa : Array [0..10] of Char }
if Valore<>0 then Molt:=Trunc(ln(Abs(Valore))/ln(1000)) else Molt:=0;          { Molt : Integer }
Str(Valore/exp(Molt*ln(1000)):5:5,Stringa);
if (Molt>=-4) and (Molt<5) then Stringa[5]:=Moltiplicatore[Molt+4];
```

TAsseX gestisce il righello della pulsazione, corrispondente all'asse delle ascisse. Il suo aspetto grafico è presentato in figura:

Si noti come nella fase di inizializzazione vengano creati due oggetti, il pulsante per la selezione della pulsazione o della frequenza (visibile nella parte destra della figura), e l'oggetto *TScrivi*, che viene visualizzato solo quando si è in fase di rilevamento dei valori dal diagramma.

```
Constructor TAsseX.Init(AParent: PWindowsObject; ATitle: PChar);
begin
   inherited Init(AParent, ATitle);
   Attr.Style := ws_Child or ws_Visible;
   RadHzBtn := New(PButton, Init(@Self, 400, 'rad/s', 10, 3, 40, 16, False));
   Scrivi:= New(PScrivi,init(@self))
end;
```

Per posizionare in modo corretto il pulsante *RadHzBtn*, indifferentemente dalle dimensioni della finestra, si ricorre ad una procedura *WmSize* del tipo già visto per *TBode*, e cioè:

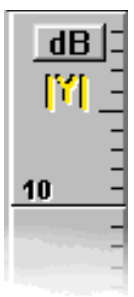
```
procedure TAsseX.WmSize;
var
   R: TRect;
begin
   GetClientRect(HWindow, R);
   MoveWindow(RadHzBtn^.HWindow,R.Right-50,3, 40, 16, False)
end;
```

al riguardo si noti come sia ancora valida la filosofia dell'applicazione nell'applicazione, di cui si è trattato precedentemente, tipica della programmazione ad oggetti (OOP, Object Orientered Programming): quando le dimensioni della finestra principale vengono modificate, *TBode.WmSize* riposiziona i pulsanti *ModeBtn* e *NuovoBtn* e ridimensiona gli oggetti che compongono *TBode*, fra cui *TAsseX*, mentre *TAsseX.WmSize* provvede automaticamente allo spostamento di *RadHzBtn.*

Il funzionamento di *TAsseX.Paint* può essere riassunto nelle seguenti operazioni:

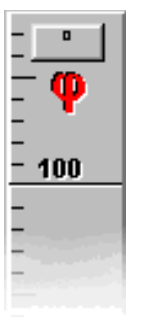
- tracciatura delle linee di contorno che realizzano l'effetto rilievo
- avvio di un ciclo che traccia nella posizione corretta le linee ed i valori delle decadi, previa conversione nel formato 'numero-moltiplicatore', nel modo già visto, se ci si trova nel range coperto dalla costante *Moltiplicatore*, e cioè fra 10^{-12} e 10^{12}, altrimenti nel formato esponenziale.
- avvio di due cicli, uno per i poli ed uno per gli zeri, per la tracciatura dei triangolini che evidenziano la posizione delle varie radici.

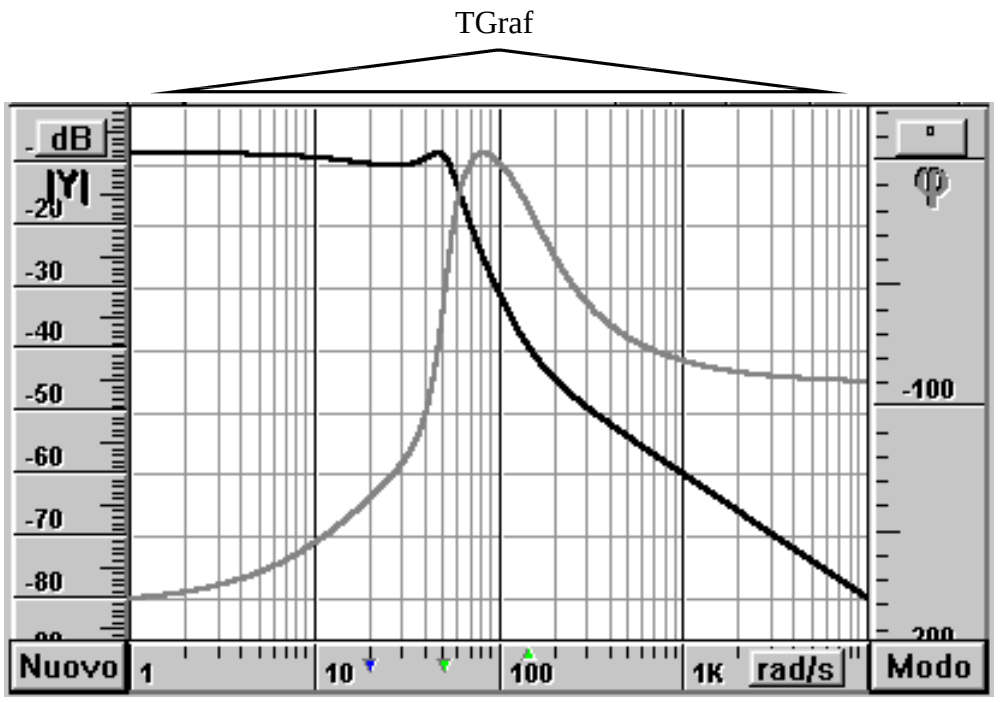
si tenga presente che *TAsseX* si basa su due soli valori, il numero delle decadi da tracciare e il valore corrispondente alla decade di inizio (il diagramma comincia e termina sempre su pulsazioni o frequenze pari a potenze di 10); tali valori sono contenuti nelle variabili globali *Decadi* e *XMin*, modificate, come vedremo, da *TGraf.Paint* durante l'autoscala.



TasseY e *TAsseF* sono praticamente identici, e presentano molte analogie con *TAsseX*. Le principali differenze sono:

- linee di gradazione posizionate in modo condizionato dalla variabile booleana *Fisso*: se True, utilizza indistintamente dieci divisioni, ciò può portare tuttavia a valori numerici corrispondenti difficilmente leggibili; se False, adatta le divisioni nel modo migliore;
- visualizzazione dei simboli del modulo e della fase rispettando i colori corrispondenti.





L'oggetto che gestisce il diagramma vero e proprio è *TGraf*. In particolare, *TGraf.Paint* effettua tutte le procedure necessarie alla tracciatura del grafico, sia per l'uscita a video che su stampante.

Si inizia esaminando il funzionamento normale di TGraf.Paint, rinviando alla fine la presentazione delle modifiche effettuate per ottenere la stampa.

1) con due cicli si determinano il valore minimo e quello massimo raggiunto dalla pulsazione associata ad ogni radice (se è reale, coincide con il valore assoluto della radice stessa, se è una coppia immaginaria, è pari alla ω_n). Si escludono ovviamente dal calcolo i poli e gli zeri nell'origine.

2) per fare in modo che il diagramma cominci e termini sempre su pulsazioni pari a potenze di 10, si approssimano i due valori trovati; inoltre, si diminuisce di una decade il valore iniziale (*Xmin*), si aumenta di due quello finale (*XMax*) e si calcola *Decadi* (si veda in figura precedente, ad esempio, la posizione delle radici ed il risultato ottenuto) . Ecco come si è dovuto procedere:

```
if XMax>=1 then Nmax:=Trunc(Ln(XMax)/Ln(10))+2          {NMax,NMin,Decadi : Integer}
else Nmax:=Trunc(Ln(XMax)/Ln(10))+1;

if XMin>=1 then  Nmin:=Trunc(Ln(Xmin)/Ln(10))-1
else Nmin:=Trunc(Ln(Xmin)/Ln(10))-2;

Decadi:=Nmax-Nmin;
XMax:=exp(NMax*ln(10));
XMin:=exp(NMin*ln(10));
```

si ricordi che *XMin* e *Decadi* sono variabili globali utilizzate anche da *TAsseX.Paint.*

3) Se la voce del menu 'Grafico/Ordinate/Autoscala reale' non è selezionata, e cioè si desidera che l'autoscala delle ordinate venga effettuata in base al diagramma reale piuttosto che a quello asintotico, si effettua il seguente ciclo:

```
GetClientRect(Hwindow,Dimens);                                  {Dimens : TRect}
for x:=0 to (Dimens.Right-Dimens.Left) do                       {x : Integer}
begin
   w:=exp(x*ln(Xmax/Xmin)/(Dimens.Right-Dimens.Left))*Xmin;     {w:Extended}
   if Hz then w:=w*(2*pi);
   Modulo:=ModuloW(w);
   if Modulo>YMax Then YMax:=Modulo;
   if Modulo<YMin Then YMin:=Modulo;
   Fase:=FaseW(w);
   if Fase>FMax Then FMax:=Fase;
   if Fase<FMin Then FMin:=Fase
end;
```

esso calcola i valori minimo e massimo raggiunti dal modulo e dalla fase fra i limiti *Xmin* e *XMax.* Questo metodo, tuttavia, comporta parecchi calcoli (dipende dalla larghezza della finestra, infatti il numero di cicli è pari al numero di pixel che si trovano fra il margine sinistro e quello destro), e su sistemi lenti ciò può portare a delle inutili pause fra un aggiornamento e l'altro. Questo è uno dei motivi per cui si è aggiunta la voce del menu suddetta, la quale costringe il programma ad effettuare i calcoli per l'autoscala (asintotica) solo in alcuni punti:

- per il modulo, alle pulsazioni corrispondenti alle varie radici;
- per la fase, una decade prima e una decade dopo ogni radice.

4) si ritoccano *Xmin, XMax, Fmin, FMax* in modo tale da ottenere il 10% di spazio libero fra i limiti superiore ed inferiore della finestra e le linee del grafico (v. figura a pag. precedente), tenendo anche conto di certesituazioni particolari, ad esempio quando esistono solo radici nell'origine e quindi la fase è costante, nel qual caso *Fmin* e *FMax* coincidono e perciò devono essere opportunamente modificate, oppure, quando si ricorre all'autoscala asintotica in presenza di una sola radice, e si ha allora il calcolo del modulo in un unico punto e quindi *Xmin=XMax.*

NOTA: se non vi sono ne poli ne zeri, l'autoscala (formata dai 4 punti precedenti), viene completamente saltata e si assegnano ai limiti i seguenti valori:

```
XMax:=0.1;
Ymin:=10;
YMax:=20+uM;
Ymin:=-20+uM;
FMax:=pi+uF;
FMin:=-pi+uF;
```

5) si applicano le opportune conversioni a seconda delle unità di misura selezionate. Le variabili booleane *dB* e *Rad* vengono impostate rispettivamente dalle procedure *TAsseY.dBMod* e *TAsseF.RadDeg*, le quali sono collegate ai due pulsantini situati sui righelli (v. figura a pag. precedente). Se *dB*=True, il modulo è in dB, altrimenti è il corrispondente valore lineare assoluto; se *Rad*=True, la fase è misurata in radianti, altrimenti in gradi.

```
if not dB then
begin
   YMax:=exp(ln(10)*(YMax/20));
   YMax:=YMax+0.2*YMax;
   YMin:=0
end;
if not Rad then
begin
   FMax:=FMax/pi*180;
   FMin:=FMin/pi*180
end;
```

6) se *Ordinate* è true, e cioè se la voce del menu 'Grafico/Ordinate' è attiva, devono essere disegnate le linee orizzontali corrispondenti alle ordinate. Il metodo di tracciatura è condizionato dallo stato assunto dalla variabile booleana *Fisso*, in modo simile a quanto si è visto per *TAsseY* e *TAsseF*. Quando *Fisso*=True, il procedimento che permette di tracciare le 10 divisioni fisse è semplicemente il seguente:

```
if Ordinate then
begin
   if fisso then for y:=1 to 10 do                                  {y : Integer}
   begin
      Moveto(PDC,Dimens.Left,Round(Dimens.Bottom/10*y));
      lineto(PDC,Dimens.Right,Round(Dimens.Bottom/10*y))
   end
   else
   ...
   ...
```

si noti che quando *Fisso*=False, le linee orizzontali vengono allineate ai segni sul righello del modulo (quello a destra) se *OrdinateMod*=True, su quelli del righello della fase in caso contrario; tale variabile booleana viene impostata dalle procedure sensibili alla pressione del tasto sinistro del mouse *TAsseY.WMLButtonDown* e *TAsseF.WMLButtonDown.*

7) allo stesso modo, se *Ascisse* (Grafico/Ascisse) è pari a True, si tracciano le linee delle ascisse.

8) se *AssiCritici*=True (voce 'Grafico/Assi critici' attiva), si disegna una linea orizzontale (più scura delle altre) corrispondente a 0 dB e diverse linee orizzontali a ±180°, ±540°, ±900°, ecc.

9) a questo punto si avviano quattro cicli che effettuano la tracciatura vera e propria degli andamenti del modulo e della fase; essi sono condizionati dallo stato delle variabili booleane*Reale* ed *Asintotico* che, come si è visto, possono essere modificate via menu o tramite il pulsante 'Modo'. Sono tutti simili a quello riportato di seguito (valido anche per il diagramma asintotico dato che si è adottato pure per esso il calcolo punto per punto)

```
{selezione colore}
for x:=0 to Dimens.Right-Dimens.Left do
begin
   w:=exp(x/(Dimens.Right-Dimens.Left)*ln(Xmax/Xmin))*Xmin;
   if Hz then w:=w*(2*pi);
   Modulo:=ModuloW(w);
   if not dB then Modulo:=exp(ln(10)*(Modulo/20));
   y:=Trunc(Dimens.Bottom/(Ymax-Ymin)*-Ymin + Modulo/(Ymax-Ymin)*Dimens.Bottom);
   if x<>0 then lineto(PDC,Dimens.Left+x,Dimens.Bottom-y)
   else Moveto(PDC,Dimens.Left,Dimens.Bottom-y)
end;
```

GESTIONE DELLA STAMPA

Il Pascal per Windows mette a disposizione un oggetto, chiamato *TWindowPrintout*, tramite il quale è possibile stampare un oggetto di tipo visuale statico (ovviamente non si possono stampare animazioni, ma i singoli fotogrammi sì). Ad esempio, volendo stampare una finestra associata alla variabile*Grafico*, è sufficiente ricorrere alle seguenti istruzioni:

```
P := New(PWindowPrintout, Init('Diagramma WinBode', Grafico));
Printer^.Print(@Self, P);
```

la prima riga crea l'oggetto di tipo *TWindowPrintout*, la seconda effettua la stampa.

Tuttavia, la finestra in questione viene stampata alla sua risoluzione attuale, la quale è in genere troppo bassa relativamente a quella della stampante, per cui si ottiene, come risultato su carta, un diagramma con linee molto spesse. Per ottenere una rappresentazione adeguata è quindi necessario ricorrere a qualche aggiustamento; questa è la procedura che, come si è visto, viene attivata dalla voce del menu 'WinBode/Stampa':

```
procedure TBode.CMFilePrint(var Msg: TMessage);
var
   P: PPrintout;
begin
   Stampa:=True;
   Grafico^.Show(sw_Hide);
   MoveWindow(Grafico^.HWindow,0,0,1500,1000,True);
   InvalidateRect(Grafico^.HWindow,nil,True);
   P := New(PWindowPrintout, Init('Diagramma WinBode', Grafico));
   Printer^.Print(@Self, P);
   Stampa:=False;
   WmSize(Msg);
   InvalidateRect(HWindow,nil,True);
   Grafico^.Show(sw_Show);
   Dispose(P, Done)
end;
```

in pratica, dopo aver reso invisibile la finestra grafica, la si sovradimensiona alla risoluzione di 1500×1000, si forza l'aggiornamento alla nuova risoluzione, e solo allora si effettua la stampa; infine si ripristina il tutto. La procedura *TGraf.Paint* (che è quella che effettua l'aggiornamento) contiene parecchie istruzioni condizionate dallo stato della variabile booleana *Stampa*, la quale, come si vede dal codice sorgente, viene modificata in fase di stampa. Le modifiche apportate al diagramma quando *Stampa*=True sono:

- linee del modulo e della fase rese più spesse di quelle orizzontali e verticali della griglia;
- esclusione di alcune istruzioni grafiche, come quelle che disegnano le linee di ombreggiatura;
- aggiunta degli assi, con relative scale graduate (anche il font viene cambiato), indici e unità di misura;
- aggiunta di due pallini demarcatori che distinguono la linea del modulo da quella della fase.

Alcune di queste alterazioni si sono rese necessarie per ottenere stampe ottimali anche in bianco e nero.