

## Plotter per Diagrammi di Bode

Implementazione (v. 1.2.0)

<http://escher07.altervista.org>

### Generalità

Abbiamo visto nel corrispondente [documento](#) gli elementi principali della teoria dei diagrammi di Bode. Lo scopo di queste pagine è quello di utilizzare tali elementi per scrivere un programma in C# in grado di rappresentare a schermo i diagrammi di Bode di una generica FDT. In particolare le funzionalità saranno le seguenti:

- Rappresentazione interna e formato testo della FDT
- Diagrammi di Modulo e Fase
- Diagrammi Asintotici di Modulo e Fase

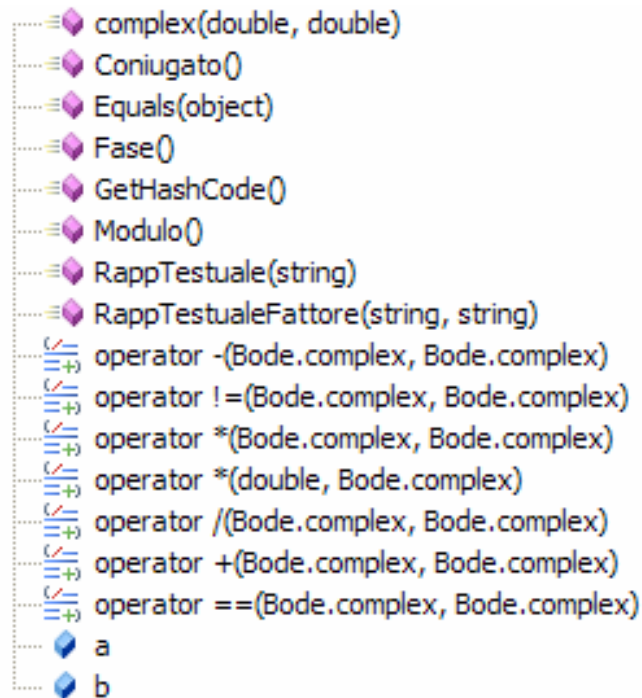
### Rappresentazione della FDT

Si è scelto di rappresentare internamente la FDT attraverso le sue singolarità, ovvero (in prima approssimazione) come vettore di poli e zeri. Si è ipotizzato il data entry direttamente agganciato a questa rappresentazione e la rappresentazione testuale come conseguenza della stessa. Ovvero non: si è implementato alcun parser che dalla  $G(s)$  inserita come stringa produca la rappresentazione interna, mentre si è prevista una funzione che per ogni aggiornamento della FDT (es. Inserimento di un polo) produca subito a video la rappresentazione come stringa della FDT aggiornata.

In pratica la FDT è modellizzati attraverso una opportuna classe FunzTrasf le cui proprietà accessibili sono queste:

- $K$  per il guadagno nella forma di Bode
- VettPoli, arraylist contenente variabili di tipo complesso
- VettZeri, arraylist contenente variabili di tipo complesso

Il tipo complesso (complex) è definito tramite un'altra opportuna classe, dotata di queste funzionalità:



A parte le solite funzioni (modulo, fase, coniugato...) e gli overload degli operatori notiamo due funzioni che restituiscono tipi string ovvero:

- RappTestuale
- RappTestualeFattore

Questi sono utilizzati per la rappresentazione testuale della FDT e ad esempio su un complesso  $(a,b)=(1,2)$  restituiscono rispettivamente:

- "1+j2"
- "[s-(1+j2)]"

Gestendo tutti i casi in cui a e/o b sono nulli (es. per  $(a,b)=0$  restituiscono "0" ed "s" e non cose inutilmente prolisse tipo "0+j0" o "[s-(0+j0)]".

Una nota sull'overload dell'operatore "=". Se lo inseriamo senza l'overload delle funzioni Equals() e GetHashCode otteniamo degli warning e dei funzionamenti anomali (es. posto  $z1=1+j2$  e  $z2=1+1+j2$  il codice  $(z1==z2)$  restituisce false!).

Le due funzioni sono state implementate così:

```
public override bool Equals(object obj)
{
    bool retval = false;
    if (obj == null)
    {
        retval = false;
    }
    if (Object.ReferenceEquals(this, obj))
    {
```

```

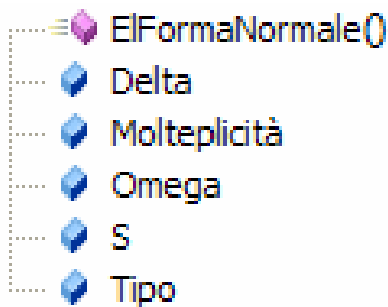
        retval = true;
    }
    if (this.GetType() != obj.GetType())
    {
        retval = false;
    }
    return retval;
}

public override int GetHashCode()
{
    return a.GetHashCode() * b.GetHashCode();
}

```

Tornando alla nostra FDT esiste anche qui una funzione di rappresentazione testuale simile a quelle di cui sopra. E' detta "RAW" perché isola con dei simboli opportuni le potenze che saranno poi associate a carattere apice nell'editor di tipo RTF.

Da notare che questa rappresentazione, chiamata ogni volta che viene aggiunto un nuovo elemento (polo, zero, costante) costruisce una rappresentazione "canonica" che è un array (realizzato dalla funzione FormaNormale()) di Elementi di questo tipo:



Dove S è di tipo complex, Tipo = Polo o Zero, Omega e Delta sono :

```

ElFn.Omega = S.Modulo(); ;
ElFn.Delta = S.a/Z.Modulo();

```

ovvero pulsazione di taglio e fattore di smorzamento (significativo, cioè diverso da 1 solo per coppie di coniugati) e dove la molteplicità ha significato ovvio.

## Diagrammi di Modulo e Fase

Una volta creata la rappresentazione interna della FDT ed implementata la classe complex la determinazione di Modulo e Fase ad una certa frequenza è abbastanza banale.

Nella prima stesura del programma erano state previste per lo scopo – come forse è più intuitivo – due funzioni di tipo double, Modulo(f) e Fase(f). Ciascuna di esse calcolava la quantità  $s(f)=j2\pi f$  e poi ricava il valore del modulo della FDT attraverso lo scorrimento di tutti gli zeri ed i poli.

Per avere una maggiore uniformità con la gestione dei diagrammi asintotici e per evitare di effettuare due volte la spazzolata in frequenza è stato deciso di accorpare queste funzioni nella seguente routine:

```
public int ModuloFase(double f, ref double M, ref double P)
```

Questa, ciclata nell'intervallo di frequenze scelto permette di popolare un array di punti fisici, ovvero di elementi come questo:

```
public class ElGrafFDT
{
    public double A;
    public double P;
    public double f;
}
```

Opportune routine presenti nelle form grafiche serviranno poi per convertire i punti fisici in punti dello schermo.

## Diagrammi Asintotici di Modulo e Fase

Mentre è stato relativamente facile implementare i grafici “reali” con delle funzioni come direttamente ricavabili dalla rappresentazione interna qualche difficoltà in più ce l’abbiamo nel caso dei diagrammi asintotici.

Il procedimento che qui si è adottato prevede di partire dalla rappresentazione della FDT in forma normale ovvero come array dei seguenti record:

- Radice
- Tipo
- Molteplicità

L’array (ricavato dal metodo FormaNormale di cui sopra) deve essere realizzato in modo da risultare in forma minima ossia con eseguite tutte le possibili elisioni poli/zeri.

Un esempio concreto chiarirà il tutto. Se questa è la  $G(s)$ :

$$G(s) = 100 \frac{s(s-1)(s+1)}{(s-4)(s^2+2s+10)}$$

Questa sarà la forma normale:

Radice	Tipo	Molteplicità
0	Zero	1
-1	Zero	1
1	Zero	1
4	Polo	1
-1+j3	Polo	1
-1-j3	Polo	1

Tabella 1

Dalla forma normale ricaviamo il diagramma asintotico del **modulo** coi seguenti passi:

1. Determinare il guadagno statico  $\mu$  nella forma di Bode (in pratica il valore in 0 della FDT privata dei poli e zeri nell'origine).
2. Determinare il contributo del polo/zero nell'origine se presente.
3. Determinare il contributo di ogni singolarità a pulsazione non nulla.
4. Sommare i vari valori in dB.

Nel nostro caso avremo per il primo punto che:

$$\mu = 100 \frac{(0-1)(0+1)}{(0-4)(0^2 + 2*0 + 10)} = 2.5 = 7.96dB$$

Per il secondo dobbiamo tener conto che nell'origine abbiamo uno zero con molteplicità uno quindi il relativo contributo è:

$$H(s)=s \Rightarrow A(f)=20\log 2\pi f$$

Questa espressione per il caso generale deve tener conto della molteplicità e del fatto che oltre che uno zero si può avere (in alternativa) anche un polo nell'origine. Il "20" precedente deriva dalla nota tabellina:

Tipo Radice	Variazione di Pendenza / dec
Zero Reale	+20dB * Molteplicità
Zero Complesso	+40dB * Molteplicità
Polo Reale	-20dB * Molteplicità
Polo Complesso	-40dB * Molteplicità

Tabella 2

Per il terzo punto il contributo dei singoli fattori si può valutare con questa formula: detta  $f_n$  la pulsazione di taglio si ha che, espressa l'ampiezza in dB:

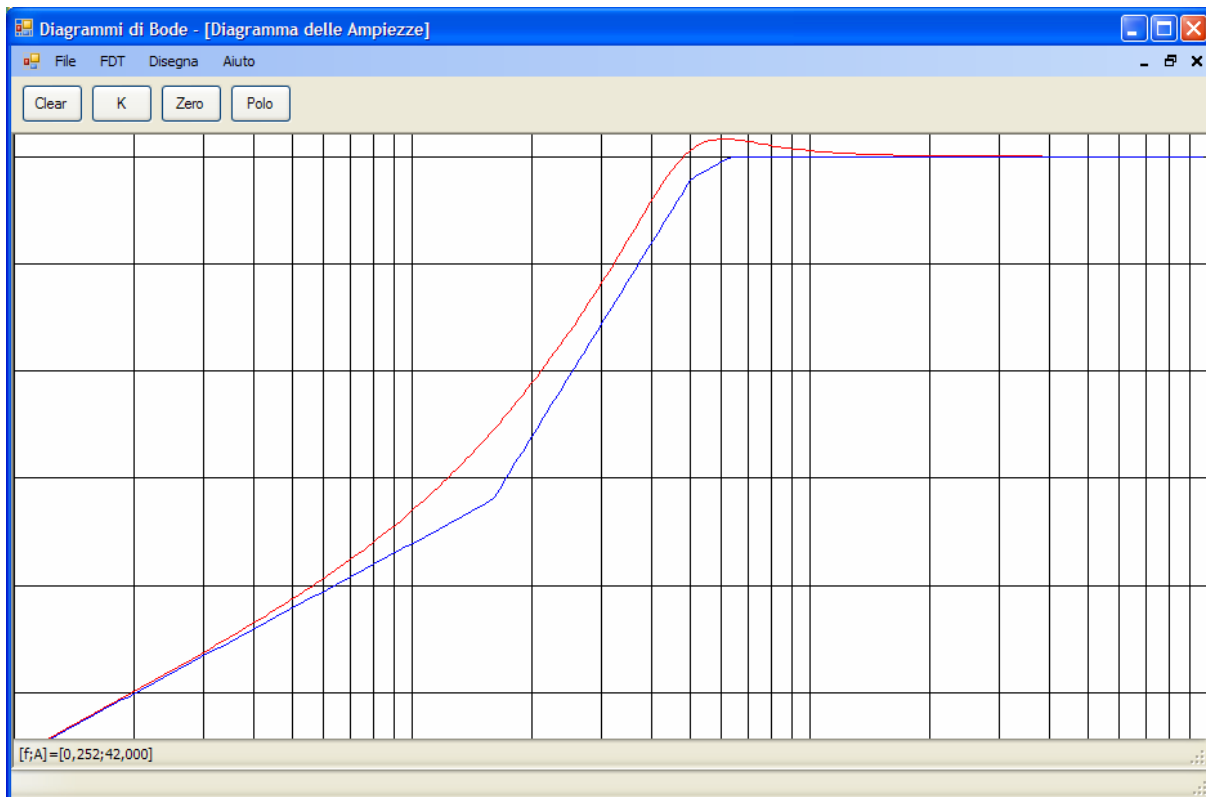
```
if (f < fn) A = 0;
if (f >= fn) A = dA * Math.Log10(f / fn);
```

dove nel nostro caso:

Singularità	dA	fn	A(f) per f>fn
1	+20	0.159	$20\log(f/0.159)$
-1	+20	0.159	$20\log(f/0.159)$
4	-20	0.637	$-20\log(f/0.637)$
$-1\pm j3$	-40	0.503	$-40\log(f/0.637)$

Tabella 3

Note tutte queste funzioni di f, ovvero informaticamente, array di coppie [f:A] con f appartenente ad [fstart,fstop] possiamo tracciare il nostro grafico:



Per quanto riguarda la fase è utile qualche precisazione in più. Innanzitutto per **fase** di un numero complesso intendiamo quella fra -180 e +180 gradi. Questo vuol dire che la corrispondente funzione sarà questa:

```
public double Fase()
{
    return Math.Atan2(b, a);
}
```

Ovvero quella funzione per cui risulta:

Quadrante	a	b	Fase
1	+	+	$\text{Atan}(b/a)$
2	-	+	$180-\text{Atan}( b/a )$
3	-	-	$-180+\text{Atan}( b/a )$

4	+	-	-Atan( b/a )
---	---	---	--------------

Tabella 4

Avere la fase in un intervallo limitato costringe a ricorrere ad una serie di funzioni di normalizzazione per cui ad esempio 375 gradi viene riportato in 15 gradi.

La Tabella 4 è poi fondamentale per vari aspetti.

Innanzitutto perché ci mostra che la nostra funzione ha una discontinuità fra il secondo ed il terzo quadrante e quindi che devono essere inseriti nel programma opportuni accorgimenti per evitare che errori dovuti ad arrotondamenti (per i quali ad esempio ottengo 180.1 invece di 179.9) falsino completamente l'aspetto del grafico.

Poi perché con questa deve essere coerente la tabella delle fasi limite, siffatta:

Tipo Radice	Plim : Fase per $f \rightarrow \infty$
Zero Reale	+90° * Molteplicità
Zero Complesso	+180° * Molteplicità * Sgn(-a)
Polo Reale	-90° * Molteplicità
Polo Complesso	-180° * Molteplicità * Sgn(-a)

Tabella 5

Qualche spiegazione è d'obbligo. Se abbiamo un fattore zero reale la situazione è questa:

$$H(s) = s - a \Rightarrow H(j\omega) = j\omega - a \Rightarrow P(\omega) = \text{Atan}2(-a, \omega)$$

Se  $a > 0$  il punto  $(-a, \omega)$  è nel secondo quadrante quindi

$$P(\omega) = 180 - \text{Atan}(\omega/|a|) \rightarrow 180 - 90 = +90$$

Se  $a < 0$  invece il punto è nel primo. Abbiamo perciò:

$$P(\omega) = \text{Atan}(\omega/|a|) \rightarrow 90$$

Per cui all'infinito il contributo di uno zero a parte reale positiva e di uno a parte reale negativa è sempre di 90 gradi, anche se la fase fa "giri" diversi.

Nel caso in cui lo zero sia complesso abbiamo quanto segue:

$$H(s) = (s-z)(s-z^*) = s^2 - z^*s - zs + zz^* = s^2 - 2\text{Re}z \cdot s + |z|^2$$

$$\Rightarrow P(\omega) = \text{Atan}2(-\omega^2 + |z|^2, -2\text{Re}z \cdot \omega) \cong \text{Atan}2(-\omega^2, -2a\omega)$$

Se  $a = \text{Re}z > 0$  il nostro punto sta per  $\omega \rightarrow \infty$  nel terzo quadrante quindi:

$$P(\omega) = -180 + \text{Atan}(2|a|/\omega) \rightarrow -180$$

Se invece  $a = \text{Re}z < 0$  il nostro punto sta per  $\omega \rightarrow \infty$  nel secondo quadrante quindi:

$$P(\omega) = 180 - \text{Atan}(2|a|/\omega) \rightarrow 180$$

Il discorso si può ripetere a segni invertiti (a causa dell'inversione numeratore/denominatore) per i poli. Mancano però ancora alcuni elementi per tracciare il grafico della fase. Prendiamo ad esempio il fattore introdotto da uno zero reale di molteplicità uno per il quale come si è visto:

$$H(s) = s - a$$

La sua fase all'infinito è +90 sia che a sia positivo che nel caso contrario. Nel primo caso deve tenerci dal sopra, nel secondo dal sotto. E' pertanto indispensabile per ogni fattore conoscere anche la sua fase iniziale che sarà:

Tipo Radice	Pini : Fase per $f \rightarrow 0$
Zero Reale	0 se $a \leq 0$ 180* Molteplicità se $a > 0$
Zero Complesso	0
Polo Reale	0 se $a \leq 0$ 180* Molteplicità se $a > 0$
Polo Complesso	0

Tabella 6

Dato che ad esempio se  $H(s) = s - 1$  allora  $H(0) = -1 \Rightarrow P(\omega) = 180$  e se  $H(s) = (s-z)(s-z^*) \Rightarrow H(0) = |z|^2 \Rightarrow P(0) = 0$ . Tutti questi elementi portano a dire che per individuare il grafico asintotico di fase occorrono i seguenti passaggi:

1. Determinare la fase del K
2. Determinare il contributo del polo/zero nell'origine se presente.
3. Determinare il contributo di ogni singolarità a pulsazione non nulla.
4. Sommare i vari valori nella stessa unità (es. Gradi)
5. Normalizzare all'intervallo scelto (-180,180)

Per il K banalmente la fase è 0 se positivo, 180 se negativo. Per il polo/zero nell'origine basta tener presente che la fase che introduce, essendo un multiplo di  $jN$  è la corrispondente  $P_{lim}$  moltiplicata per la molteplicità. Per il contributo delle singolarità a pulsazioni non nulle vale la seguente formula:

```

if (f < fspread1) P = Pini;
if ((f >= fspread1) && (f < fspread2))
{
    dP = FncUtili.NormalizzaFaseInDEG(Plim - Pini);
    P = Pini + (dP/ (2 * d * Math.Log10(costanti.fspread))) * Math.Log10(f /
fspread1);
}
if (f >= fspread2) P = Plim;

```

dove fspread1 e 2 sono le pulsazioni limite derivanti dalla "regola del 4.81" di cui nella teoria ovvero:

```

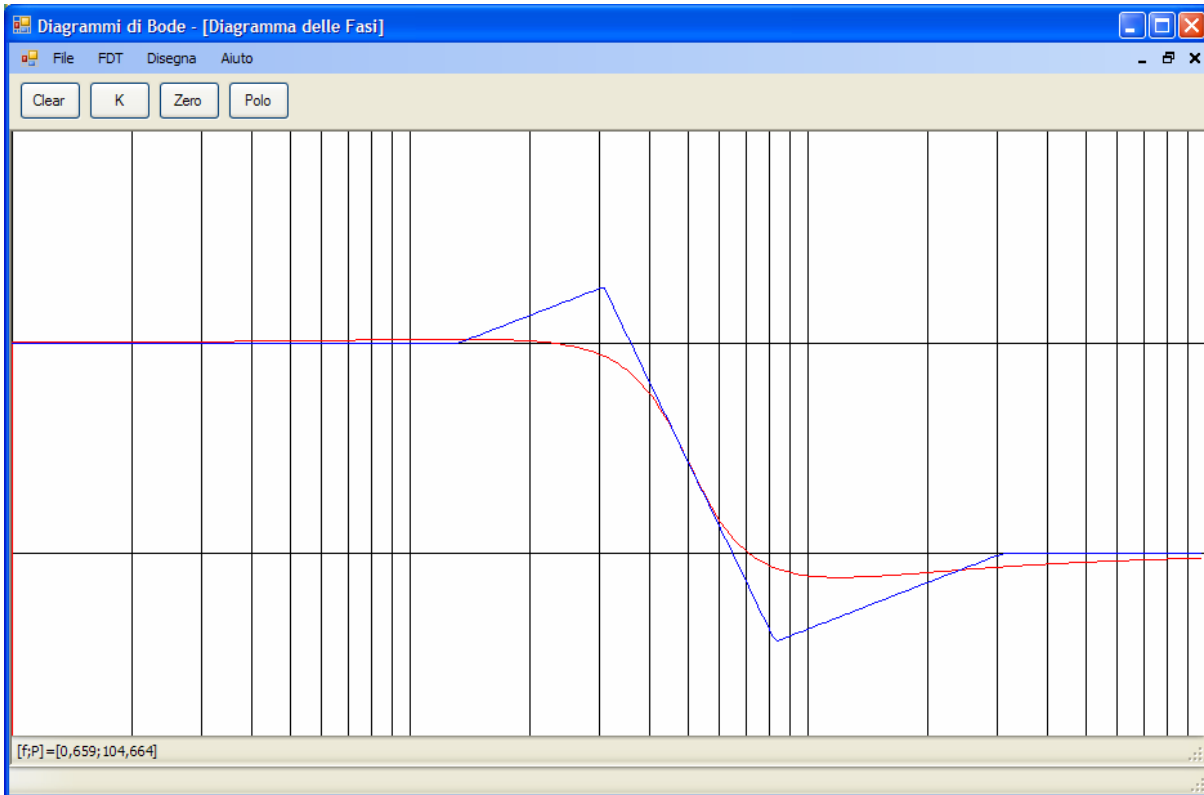
fspread1 = fn / Math.Pow(costanti.fspread,d);
fspread2 = fn * Math.Pow(costanti.fspread,d);

```

con:

```
fn = EIVFN.Omega / (2 * Math.PI);  
d = Math.Abs(EIVFN.Delta);
```

dove EIVFN è il generico elemento della forma normale della nostra FDT. In questo modo giungiamo al seguente grafico:



Nel caso particolare è utile tenere presente quanto segue:

$$G(s)=100^*\{s[s-1][s+1]\}/\{[s-(-1-j3)][s-(-1+j3)][s-4]\}$$

Zero Reale :  $w = 0$  cioè  $f = 0\text{Hz}$  : molt = 1

Zero Reale :  $w = 1$  cioè  $f = 0,159\text{Hz}$  : molt = 1

Zero Reale :  $w = 1$  cioè  $f = 0,159\text{Hz}$  : molt = 1

Poli Compl. Coniugati :  $w_n = 3,162$  cioè  $f_n = 0,503\text{Hz}$  :  $d = -0,316$  : molt = 1

Polo Reale :  $w = 4$  cioè  $f = 0,637\text{Hz}$  : molt = 1

Da cui:

$$f_{\text{spread1}}(4) = 0,637/4,81=0,132$$

$$f_{\text{spread1}}(-1\pm j3) = 0,503/4,81^{0,316}=0,306$$

ovvero il fronte di salita del polo 4 ( $P_{ini}=-180$  e  $P_{lim}=-90$ ,  $dP>0$ ) precede la discesa legata ai poli complessi ( $P_{ini}=0$  e  $P_{lim}=-180$ ).

C'è un'ultima cosa da evidenziare nell'algoritmo di calcolo del diagramma asintotico della fase ed e la illustriamo con un esempio.

Diciamo che la nostra  $G(s)$  è  $s - a$  con  $a>0$ . La **tabella 6** vuole in questo caso che il nostro grafico asintotico parta da 180 gradi. Ora il grafico "reale" è fatto con un algoritmo diverso da quello dell'asintotico e niente impedisce, per arrotondamenti vari di fatto imprevedibili, che questo parta da -180 che matematicamente è ben distinto da 180 ma in un contesto a precisione è molto vicino al 180 a causa della  $\text{Atan2}()$ .

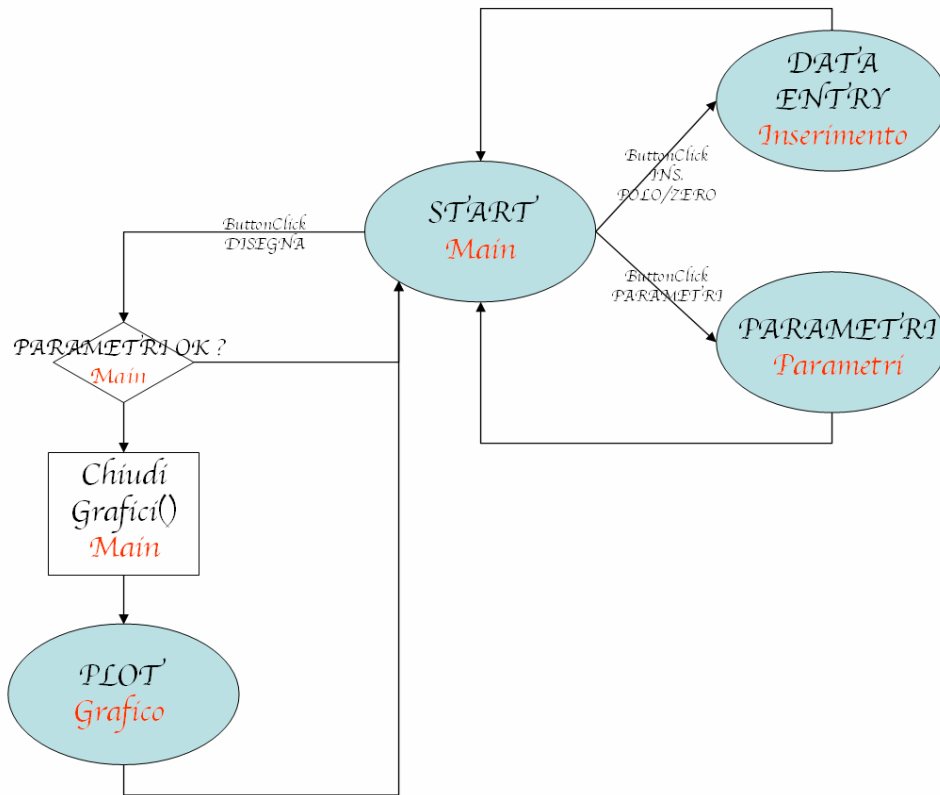
La soluzione che ho trovato non è il massimo dell'eleganza però al momento non mi viene in mente niente di meglio! In sintesi per ogni fase calcolata del diagramma asintotico si va a vedere se è "molto" diversa da quella del reale : se questo succede il nostro grafico asintotico viene corretto con uno o più giri di fase. Questa è la routine:

```
Err = PAsintDeg - PRealeDeg;
PAsintDegNew = PAsintDeg;
while (Math.Abs(Err) > 180)
{
    if (Err > 0)
    {
        PAsintDegNew -= 360;
        Err -= 360;
    }
    else
    {
        PAsintDegNew += 360;
        Err += 360;
    }
}
```

Così ad esempio se  $P_{AsintDeg}=179$  e  $P_{RealeDeg}=-181$  porremo  $P_{AsintDegNew}:=179-360=-181$ , rimettendo a posto le cose. Una osservazione: avrebbe matematicamente più senso che il valore con cui confrontare  $|\text{Err}|$  fosse 360 o giù di lì. Il punto è : quanto è il "giù di lì" ? Ho notato che la cifra da mettere si discosta da 360 quante più sono le singolarità. A questo punto piuttosto che mettere 359 o 350 e rimanere fuori in qualche caso particolare per un grado o ho messo 180 : in pratica anche se diagramma reale e asintotico sono in opposizione di fase è pressoché certo non è una condizione fisiologica (come ad esempio il forte scostamento nei diagrammi dell'ampiezza quando  $\delta \rightarrow 0$ ) e che c'è una rotazione di troppo, per cui si prova a risolvere il problema aggiungendola (o togliendola).

## L'applicazione e gli elementi grafici

Il flusso principale è questo:



L'interazione con le form ausiliarie Data Entry e Parametri avviene con la tecnica dei delegati (callback) : in pratica dalla pressione del bottone viene creata una nuova istanza della form ausiliaria in questione con il puntatore della routine di gestione nella form chiamante passato alla form chiamata. Esempio:

```

Finserimento = new frm_inserimento();
    Finserimento.AggiornaFunzTrasf += new
AggiornaFunzTrasfEventHandler(this.AggiornaFunzTrasf);
  
```

Attraverso queste vengono "popolati" l'insieme dei parametri grafici (fstart,fstop etc...) e l'istanza della classe FunzTrasf che sono presenti nella main form e corrispondono a queste sue proprietà:

```

public FunzTrasf G = new FunzTrasf();
public ObjParamGrafici SetParametri = new ObjParamGrafici();
  
```

L'aggiornamento dei grafici a seguito di pressione del "bottone" (in realtà è una voce di meni) omonimo comporta il lancio della routine `Ridisegna()` della main form. Questa lancia la `Disegna(..)` delle form di modulo e fase, passandogli tutti gli argomenti necessari; ad esempio per il modulo succede quanto segue:

```

Fgrafico_modulo.Disegna(PuntiGrafico, PuntiGraficoAsint, f1, f2, Amin, Amax,
SetParametri.TipoAsseX, SetParametri.UdmAsseYmodulo, SetParametri.AutoscaleX,
SetParametri.AutoscaleYmodulo);
  
```

La Disegna(..) riceve come input i vettori PuntiGrafico e PuntiGraficoAsint contenenti i punti fisici X,Y e li trasforma in vettori di punti in coordinate schermo con la chiamata a questa routine:

```
AggiornaDati(mem_VPtFisiciGraf, mem_VPtFisiciGrafAsint, mem_f1, mem_f2,  
mem_Amin, mem_Amax, mem_TipoAsseX, mem_TipoAsseY, mem_AutoscaleX,  
mem_AutoscaleY);
```

Poi genera un evento paint:

```
this.panell.Paint += new  
System.Windows.Forms.PaintEventHandler(this.panell.Paint);
```

con cui i vettori di coordinate schermo sono tradotti in pixel con un'opportuna istanza della classe Graphics (possibile solo all'interno di un evento Paint).

Da notare che il popolamento dei vettori di valori fisici viene effettuato a livello di main chiamando questi metodi pubblici della G:

```
G.PopolaPuntiGrafico(ref PuntiGrafico, f1, f2, Nstep, TipoAssef);  
G.PopolaPuntiGraficoAsint(ref PuntiGraficoAsint, f1, f2, Nstep, TipoAssef);
```

In questo modo le form grafiche si preoccuperanno solo di riportare a schermo delle matrici di punti fisici senza chiamare direttamente la G. Questo consente uno sviluppo più pulito dell'elaborazione che non sarebbe possibile se le form grafiche (due, una per il modulo e una per la fase) chiamassero direttamente corrispondenti routines della G (che è una).

A queste routines viene passato anche il tipo di asse delle frequenze che può essere lineare o logaritmico. Questo perché gli Nstep campioni devono essere equispaziati e la condizione di equidistanza è diversa nelle due condizioni. Infatti se l'asse f è lineare abbiamo che le formule della "spazzolata" in frequenza sono queste:

$$\begin{aligned}f_0 &= f_{\text{start}} \\ f_n &= f_{n-1} + df \\ df &= (f_{\text{stop}} - f_{\text{start}}) / N_{\text{step}}\end{aligned}$$

Se invece l'asse f è logaritmico dobbiamo porre:

$$\begin{aligned}f_0 &= f_{\text{start}} \\ f_n &= \alpha * f_{n-1} \\ \alpha &= (f_{\text{stop}} / f_{\text{start}})^{1/N_{\text{step}}}\end{aligned}$$

Dato che imporre l'equispaziatura in un asse logaritmico vuol dire:

$$\log(f_n) - \log(f_{n-1}) = \frac{\log(f_{\text{stop}}) - \log(f_{\text{start}})}{N_{\text{step}}}$$

Cosa che applicando le proprietà dei logaritmi conduce facilmente alle formule e alla definizione di  $\alpha$  di cui sopra.