Tra ConTeXt e MetaPost

Un percorso

di
Riccardo Nisi
r underscore nisi at tin dot it

Febbraio 2010

Indice

1 Premessa

Volevo conoscere ConTeXt per rinnovarmi, l'ho utilizzato per le mie lettere, ma quello che mi ha particolarmente interessato è stata la sua combinazione con METAPOST. Che avrei potuto utilizzare con LaTeX, ma allora il rinnovamento (nel programmino di apertura un punto cambia se stesso)? Quando ho raggiunto il livello di sopravvivenza ho sentito l'esigenza di dare un po' di organizzazione ai contenuti su cui più avevo lavorato.
Sono quelli che propongo. È possibile che possano interessare altri neofiti.
Si tratta di un documento ConTeXt in cui sono inseriti esempi METAPOST con codice e commento.
Per il documento e gli esempi i miei principali riferimenti sono stati *ConTeXt, the manual*, Hans Hagen, 2001; *MetaFun*, Hans Hagen, 2005; METAPOST*, A User's Manual*, John D. Hobby, 2009; *Learning* METAPOST *by Doing*, André Heck, 2005; *A very Brief Tytorial*, Urs Osvald, 2002; METAPOST*: A Reference Manual*, Peter Grogono, 2009; nonché la disponibilità e la competenza di Luigi Scarso di cui ho spesso approffittato e per cui gli sono molto riconoscente.

2 Cambia te stesso

```
%--
 a:=10;b:=5; u:=1cm; pickup pencircle scaled 15pt;
 drawdot(0,0) withcolor (.3,.6,.3);
 for i=1 upto 8:
   a:=.8a; b:=.5b;
   drawdot(a*u,b*u) withcolor (.7*i*.2,.4,.5);
 endfor
%--
```

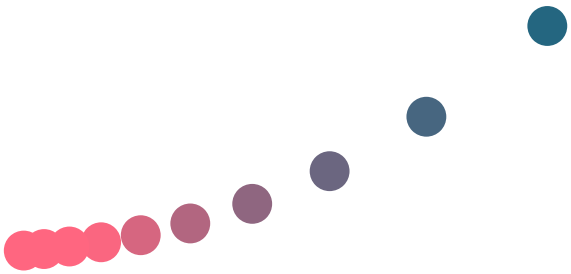

Fig. 1 Cerchi che mutano

3 Angoli e triangoli

Si prendono in esame comandi come `angle`, `dir`, `whatever` e la capacità di METAPOST di risolvere equazioni.

3.1 Angoli e bisettrici

Il comando `angle` opera con gli angoli della circonferenza goniometrica, positivi se presi in senso antiorario. Un comando come `angle(3,3)` individua l'angolo della semiretta `0-(3,3)` col semiasse positivo delle ascisse, mentre `angle(B-A)` pone l'origine della circonferenza in `A` e considera l'angolo della semiretta `AB` con l'asse `x`. La semiretta individuata da `angle(A-B)` è ruotata di 180° rispetto alla prima. Il comando `dir(angle(B-A))` restituisce il punto della semiretta `AB` che si trova sulla circonferenza unitaria di centro `A`. L'espressione `r*dir(x)` individua il punto che si trova a distanza `r` sulla semiretta per l'origine e che forma l'angolo `x` con l'asse delle ascisse.
Il primo esempio rappresentato dal triangolo rosso di `Fig.2` propone con `dir` due vettori-punto con distanza unitaria da (0,0), il percorso, chiuso da `cycle` e tracciato da `draw`, disegna un triangolo isoscele: `draw (origin -- dir(45) -- dir(0) -- cycle) scaled 3cm withcolor .625red;`.
Nel caso del triangolo giallo con `dir(angle(3,3))` si definisce la direzione del segmento (0,0) – (3,3), ma la distanza del vettore-punto, creata da `dir` rispetto al primo punto del percorso, è unitaria. Se il vettore, come in questo caso, viene moltiplicato per 1.41 questa sarà la distanza tra l'origine e il punto, che avrà coordinate (1,1). Per quanto detto il terzo punto, definito da `2*dir(angle(4,0))`, ha coordinate (2,0). Il comando `draw` unisce i punti costruendo un triangolo. Il codice per il triangolo giallo: `draw (origin -- 1.41*dir(angle(3,3))-- 2*dir(angle(4,0))--cycle ) scaled 3cm shifted (6cm,0) withcolor .625yellow;`.

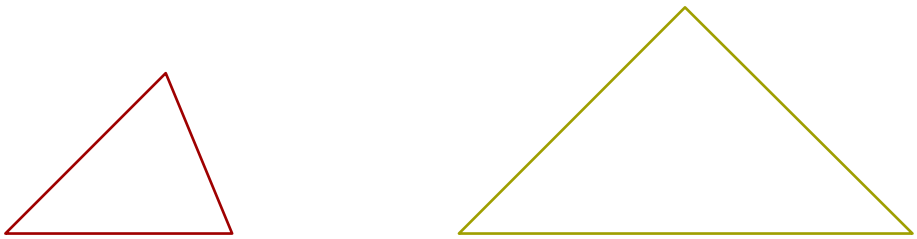

Fig. 2 Tracciamenti con i comandi `angle` e `dir`

Esempio del triangolo ABC, `Fig.3`.
Come già visto, il comando `dir`(α) individua le coordinate del punto della circonferenza unitaria centro nell'origine e raggio di direzione α. Nel comando `u*dir`(α) cambia la distanza ma il riferimento resta l'origine. Il modo più semplice per applicare `dir` ad un punto che non sia l'origine è tracciare quel segmento per origine e poi effettuare la traslazione nel punto previsto, come fa l'istruzione seguente: `draw (A-- dir(1/2*angle(B-C)))  scaled u  withcolor .625yellow dashed evenly;` che traccia il segmento unitario giallo applicato in A(0,0) e non in B o in C, come previsto dal codice. Per questo basta inserire il comando `shifted B` dopo `scaled u`.

Ma si può anche agire direttamente facendo riferimento alla proprietà che hanno i punti di essere sommati. Come si può vedere nei casi dei punti `F,S,T` le cui coordinate sono quelle del punto in cui si vuole applicare `dir` sommate a quelle del punto definito da `dir` rispetto all'origine. Si può bypassare la definizione del punto e introdurre le stesse istruzioni direttamente in `draw` come viene mostrato sotto per i punti `B` e `C`.
Le linee tratteggiate gialla e rossa in `A` sono esempi di `dir` applicati nell'origine invece che nel punto previsto. Le due punteggiate esterne al vertice `C`, visto che i segmenti magenta e blu sono bisettrici, sono i prolungamenti dei lati in `C` del triangolo. Quanto a `B`, che per la misura degli angoli va considerato al centro di una circonferenza unitaria, c'è da notare che `angle(C-B)+angle(A-B)` è $540°$ e che la semisomma ha la direzione della bisettrice di $\widehat{ABC}$ e che `angle(B-C)+angle(B-A)` è $135° + 45°$ e che i `2/3` dell'esempio, segmento blu con puntino, nella circonferenza goniometrica sono $120°$.

```
%
F=B+ u*dir(2/3*(angle(B-C)+angle(B-A)));
S=C+ u*dir(1/2*angle(C-B));
T=C+ u*dir(1/2*angle(B-C));
%
draw S withpen pencircle scaled 2pt withcolor blue;
draw T withpen pencircle scaled 2pt withcolor magenta;
draw F withpen pencircle scaled 3pt withcolor blue;
%
draw (A-- dir(angle(A-B)))  scaled 1u  withcolor .625green;
draw (A-- dir(angle(B-A)))  scaled 1u  withcolor red;
draw (A-- dir(1/2*angle(B-A)))  scaled u   withcolor blue;
draw (A-- dir(1/2*angle(A-B)))  scaled u withcolor magenta;
draw (A-- dir(1/2*angle(B-C)))  scaled u   withcolor .625yellow dashed evenly;
draw (A -- dir(angle(B-C)))  scaled u withcolor .625red dashed evenly;
draw (A -- dir(angle(A-C)))  scaled u  dashed withdots;
%
draw (B-- B+ u*dir(2/3*(angle(B-C)+angle(B-A))))   withcolor blue;
%
draw (C-- C+ u*dir(1/2*angle(C-B)))  withcolor blue;
draw (C-- C+ u*dir(1/2*angle(B-C)))  withcolor magenta;
draw (C-- C+ u*dir(angle(C-A)))  dashed withdots;
draw (C-- C+ u*dir(angle(C-B)))  dashed withdots;
%---
```

Ora, sempre nel triangolo ABC, si determinano le bisettrici degli angoli A e B e l'incentro I. Nel codice sono presenti i punti H, L, N e I di cui non si conoscono le coordinate. Ogni punto avrà bisogno di una coppia di equazioni che saranno risolte da METAPOST.
`H:` la prima equazione prende il segmento per B perpendicolare ad AC avente una qualsiasi lunghezza, definisce il secondo estremo di tale segmento con H e la mette a sistema con una equazione che definisce l'appartenenza di H anche al lato AC.

```
B - H = whatever*(A-C) rotated 90;
H     = whatever[A,C];
L=whatever[B,C]=A+whatever*dir(1/2*angle(B-A)+1/2*angle(C-A));
N=whatever[B,A]=C+whatever*dir(1/2*angle(B-C)+1/2*angle(A-C));
I=whatever[B,H]=whatever[A,L];
```

Indicando con L l'intersezione della bisettrice di $\widehat{BAC}$ con BC si scrivono le equazioni: `L = whatever[B,C];` cioè che L appartiene a BC; la seconda, `L=A+whatever*dir(1/2*angle(B-A)+1/2*angle(C-A));`, impone l'appartenenza di L alla bisettrice. Il comando `whatever` rappresenta il numero che moltiplicato per l'operatore `dir` determina la lunghezza di BH.
Volendo considerare BH come bisettrice: la prima equazione è `H=B+whatever*dir ((1/2* angle(A-B)+angle(C-B)))` la seconda sempre `H = whatever[A,C];`.
Per la bisettrice CN si ha: `N=whatever[B,A]=C+whatever*dir(1/2*angle(B-C)+1/2*angle(A-C));`.
Il `whatever*dir` dichiara l'appartenenza di N alla bisettrice uscente da C.

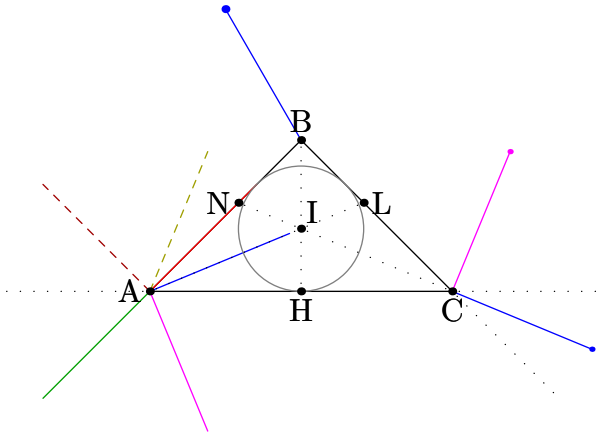

Fig. 3 Il comando `dir`, bisettrici e incentro.

Per l'incentro basta dichiararne l'appartenza a due delle bisettrici. `I: I=whatever[B,H]=whatever[A,L];`. Il raggio della circonferenza inscritta è la distanza IH. Il codice che la traccia `draw fullcircle scaled 2 length(I-H) shifted I withcolor .5white;`.

3.2 Ortocentro

Dopo le assegnazioni preliminari:

```
pair A,B,C,H,K,S,O;
A=(0,0); B=(3cm,0); C=(1cm,2cm);
```

per le altezze:

```
H - A = whatever * (B-C) rotated 90; H = whatever [B,C];
K - B = whatever * (A-C) rotated 90; K = whatever [A,C];
S - C = whatever * (A-B) rotated 90; S = whatever [A,B];
```

per l'ortocentro: `O = whatever [A,H]; O = whatever [B,K];`
per il tracciamento, `Fig.4`: `draw A--B--C--cycle;`

```
draw A--H dashed evenly;draw B--K dashed evenly; draw C--S dashed evenly;
draw O withpen pencircle scaled 2bp  withcolor red;
```

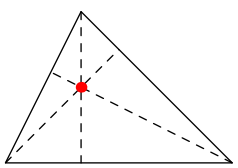

Fig. 4 Altezze e ortocentro.

3.3 Incentro

La bisettrice di un angolo può essere ottenuta oltre che, come visto, con `dir` anche con l'operatore `rotated`. Riferendosi al triangolo di `Fig.5`, indicando con I l'incentro, la bisettrice dell'angolo B si può ottenere ruotando il lato BC di metà dell'ampiezza di $\widehat{ABC}$. Occorre però una certa attenzione. Infatti a seconda del lato la rotazione può essere oraria o antioraria, con implicazioni nel codice. Inoltre il calcolo dell'ampiezza dell'angolo dovrà tener conto dei quadranti di collocazione dei lati dell'angolo.

Vediamo i tre vertici.

Prendendo come riferimento il vertice B, l'angolo del lato BC è del secondo quadrante come lo è quello del lato BA, che però ha un'ampiezza maggiore del primo. Trattandosi di una rotazione la bisettrice può essere ottenuta da una rotazione oraria, angolo negativo, di `(A-B)`, o antioraria di `(C-B)`. Dalla scelta dipende l'ordine con cui disporre i termini della differenza. Con la rotazione oraria di `(A-B)`, gli angoli, del secondo quadrante, debbono essere disposti in maniera che la differenza sia negativa cioè dal minore, `angle(C-B)`, si sottrae il maggiore, `angle(A-B)`. Al contrario se ruota `(C-B)`. I due codici:

```
(I-B) = whatever * ((B-A) rotated 1/2( angle(C-B) - angle(A-B)) );
(I-B) = whatever * ((B-C) rotated 1/2( angle(A-B) - angle(C-B)) );
```

Vertice A. I lati sono del I e IV quadrante con angolo positivo e negativo. Se l'ampiezza di $\widehat{A}$ si ottiene sottraendo dall'angolo positivo, `angle(C-A)`, quello negativo, `angle(B-A)`, si ruota `A-B` in senso antiorario: `(I-A) = whatever * ((A-B) rotated 1/2( angle(C-A) - angle(B-A)));`
Il contario se dovrà ruotare `A-C`:

```
(I-A) = whatever * ((A-C) rotated 1/2( angle(B-A) - angle(C-A)));
```

Con C abbiamo angoli del IV e III quadrante. L'ampiezza positiva si ottiene sottraendo dal minore il maggiore dei due angoli negativi, con `1/2( angle(B-C) - angle(A-C))`, e `C-A` dovrà ruotare in senso antiorario.

```
(I-C) = whatever * ((C-A) rotated 1/2( angle(B-C) - angle(A-C)));
```

Il punto I, incognito, è comune alle due bisettrici e METAPOST ne calcola le coordinate mettendole autonomamente a sistema. Per calcolare il raggio della circonfernza inscritta si indica con H la proiezione ortogonale di I su BC: `I-H = whatever * (B-C) rotated 90;  H = whatever[B,C];`.
Il codice.

```
%----------
 pair A,B,C,I,H; u:=1.5cm; labeloffset:=4pt;
 A=(0,0); B=(2u,-.5u); C=(u,u);
 draw A--B--C--cycle;
 (I-A) = whatever * ((A-C) rotated 1/2( angle(B-A) - angle(C-A)));
 (I-B) = whatever * ((B-A) rotated 1/2( angle(C-B) - angle(A-B)));
 draw I--A; draw I--B; draw I--C;
 I-H = whatever * (B-C) rotated 90;
 H = whatever[B,C];
 draw fullcircle scaled 2 abs(I-H) shifted I;
 dotlabel.lft("A",A);dotlabel.top("C",C);
 dotlabel.rt("B",B);dotlabel.bot("I",I);
 dotlabel.urt("H",H);
%----------
```

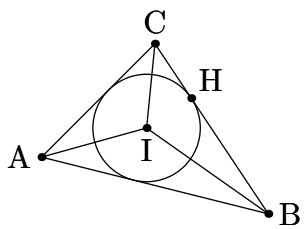

Fig. 5 Bisettrici e incentro.

3.4 Macro e bisettrici

Nei due seguenti esempi si vuole mettere a punto il codice di una macro che possa eseguire o facilitare compiti come il tracciamento delle bisettrici di un triangolo e la determinazione dell'incentro.

3.4.1 Esempio1

La vardef `vect` assegnati tre punti traccia il triangolo e la bisettrice `A-L`; ha tre argomenti A, B, C di tipo `pair`, dichiara il pair L inizialmente non noto mentre la linea `L=whatever[B,C]= A+whatever*dir(1/2*`

`angle(B-A)+1/2*angle(C-A))` è formata da due equazioni che assegnano il punto `L` al segmento AB e alla bisettrice dell'angolo A. Se riceve le coordinate dei tre argomenti A,B,C la macro sarà in grado di tracciare il relativo triangolo ,`Fig.6`, di determinare le coordinate di L e con `draw A--L` di tracciare la bisettrice uscente dal vertice con le coordinate del primo carattere del chiamante. Se il programma lancia l'invocazione `vect(B,C,A)` sarà tracciato lo stesso triangolo insieme alla bisettrice uscente dal vertice con le coordinate di B. Infine se l'invocazione è `vect(C,A,B)` sarà tracciata la terza bisettrice.
Il triangolo e le bisettrici sono correttamente tracciati e il programma principale è in grado di assegnare le etichette con i relativi caratteri. Sembrerebbe tutto risolto, ma non lo è: infatti non sono note né sono determinabili le coordinate dell'incentro del triangolo.

```
%-------------
vardef vect(expr  A,B,C)=
 pair L; draw (A--B--C -- cycle);
 L=whatever[B,C]= A+whatever*dir(1/2*angle(B-A)+1/2*angle(C-A));
 draw A--L;
enddef;
%
pair A,B,C; u:=2cm; A=(u,0); B=(u,u); C=(2u,0);
vect(A,B,C); vect(B,C,A); vect(C,A,B);
%-------------
```

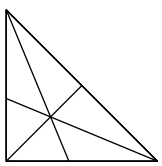

Fig. 6 Macro per il tracciamento delle bisettrici.

3.4.2 Esempio2

La vardef offre però un varco all'ambiente esterno. Il tracciamento delle bisettrici con il codice della macro è un risultato, occorre determinare l'incentro. Ad esempio perché non inserire nella macro la linea `I=whatever[B,K]= whatever[A,L];` come si farebbe per determinare l'incentro intersecando due bisettrici in un sorgente senza macro? Il fatto è che la macro è in grado di tracciare le tre bisettrici ma lo fa in operazioni separate in cui le tratta e vede una alla volta, ovviamente i tracciati permangono.
E se la macro *vede* una bisettrice alla volta nel suo ambito non può essere calcolato l'incentro. Le variabili definite nella macro non sono visibili all'esterno, ma se si chiama la macro con un parametro incognito che sia utilizzato in una sua equazione si crea quella comunicazione che mancava. Una equazione per la bisettrice che utilizza il `pair I`:
`(I-A)=whatever*( (A-C) rotated 1/2(angle(B-A)-angle(C-A)));`.
Passando il parametro I (incentro) alla macro questa lo accetta come incognita che non può risolvere direttamente ma la restituisce all'ambiente esterno che, effettuando due chiamate, riceve in risposta due equazioni aventi per incognite le coordinate di I, e MetaPost è specializzato nel risolvere queste situazioni. Ecco che per il programma principale I diventa noto e utilizzabile. Con le equazioni `I - H = whatever*(B-C) rotated 90; e H = whatever[B,C];` si trova su BC il piede H della perpendicolare da I che è anche il raggio della circonferenza inscritta nel triangolo, `Fig.7`,: `draw fullcircle scaled 2 abs(I-H) shifted I;`.

```
vardef vect(expr I, A,B,C)=
  (I-A)=whatever*( (A-C) rotated 1/2(angle(B-A)-angle(C-A)));
  % la linea che segue ÃŃ equivalente e alternativa a quella precedente.
    I=A+whatever*dir(1/2(angle(B-A)+angle(C-A)));
enddef;
%---------
pair I, A,B,C, H; u:=4cm;
```

```
A=(u,0); B=(u,u); C=(2u,0);
draw (A--B--C -- cycle);
vect(I, A,B,C); vect(I, B,C,A);
draw I--A;draw I--B;draw I--C;
I - H = whatever*(B-C) rotated 90; H    = whatever[B,C];
draw fullcircle scaled 2 abs(I-H) shifted I;
```

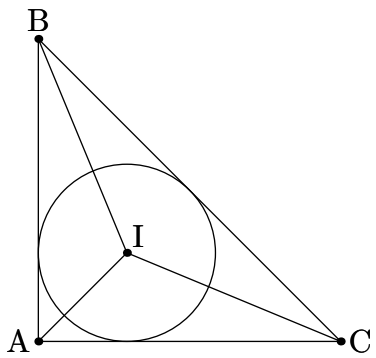

Fig. 7 Macro per bisettrici e incentro.

4 Percorsi e tangenti

METAPOST costruisce una curva, che non abbia una esplicita equazione matematica, a partire da due punti con uno o più archi di Bézier. Per un singolo arco due punti servono a individuarne gli estremi, altri due punti, di controllo, servono in particolare a definire la direzione della curva nei punti iniziale e terminale. Gli algoritmi di METAPOST sono in grado di provvedere in proprio, con propri criteri, ai punti di controllo. Le direzioni agli estremi, la tensione, il grado di curvatura della curva sono controllabili con specifici parametri. Dalle coordinate dei quattro punti si ricava, METAPOST lo fa autonomamente, l'equazione parametrica del percorso. La curva ottenuta, `Fig.8` , è un percorso controllato dal parametro `t`. Nel caso di un singolo arco $0 \leq t \leq 1$, in quello di n archi $0 \leq t \leq n$. Dato un percorso `p` il numero dei suoi archi è dato da `length(p)`.

Il percorso `p1` a sinistra è composto da archi contigui di Bézier di cui sono state assegnate le coordinate degli estremi. Per determinare le equazioni parametriche di ciascun arco, con $0 \leq t \leq 3$, a METAPOST sono necessari anche due punti di controllo che assegna in proprio con suoi algoritmi e con criteri casuali che sono visualizzati dai comandi `precontrol` e `postcontrol`. Il grafico evidenzia anche che nei punti di giunzione degli archi gli algoritmi tendono a non cambiare la direzione del percorso. Il percorso `p2` interseca `p1` in un punto `P` di cui il comando `intersectionpoint` calcola le coordinate cartesiane. Ma con `intersectiontimes` si calcolano anche le coordinate parametriche `tp2` e `tp1` che vengono visualizzate nel grafico, `p1` è intersecato in un punto del secondo arco.

```
%----------
u:=1cm;
pickup pencircle scaled 1pt;
path p[]; pair q[],P;
p1 = (0,0) .. (2u,-1u)..(4u,1u)..(5u,-1u);
p2 = (2u,.5u){dir -20} ..{dir -70} (3.5u,-1u);
draw p1;draw p2  withcolor .625red;
for t=0 upto length(p1):
 q1 := precontrol t of p1;
 q2 := postcontrol t of p1;
 draw q1--q2 withcolor .5white;
 draw q1 withpen pencircle scaled 3bp withcolor magenta;
 draw q2 withpen pencircle scaled 3bp withcolor cyan;
endfor;
(tp1,tp2)= p1 intersectiontimes p2;
P       := p1 intersectionpoint p2;
```

```
label.rt("tp1="&decimal tp1, (0,1.5u));
label.rt("tp2="&decimal tp2, (0,1.1u));
draw fullcircle scaled 4 shifted P withcolor green;
%----------
```

Il percorso `p` a destra, formato da tre archi di curva, viene rappresentato e diviso in dodici parti. Le divisioni del primo arco sono segnalate da trattini normali alla curva, quelle degli altri due archi da punti rossi.
L'operatore `length(p)` restituisce il numero dei segmenti di curva, 3, assegnati dalla definizione di `p`. Poiché si è scelto di suddividere il percorso in 12 parti ciascun arco di curva sarà diviso in 4 parti.
Il ciclo 4 divide il primo arco in quattro parti segnalate da trattini normali alla curva. Il ciclo 2 calcola le posizioni del `p` in cui tracciare i punti rossi: $\frac{i}{n} * length(p) = \frac{i}{12} * 3$ restituisce le posizioni `(1.25, 1.5,1.75, 2, 2.25, 2.5,2.75, 3)` (o meglio i valori del parametro t) che in questo elenco appaiono creare intervalli della stessa lunghezza ma in effetti ogni intervallo risulta 0.25 del proprio arco di appartenenza. Nel ciclo 3 ad ogni quarto degli archi 2 e 3, si provvede con `shifted point i of p` a posizionare il vettore unitario nel punto previsto del percorso. Il ciclo 3 provvede ad evidenziare con un punto verde gli estremi dei tre archi di `p`.

```
%----------
  path p;
  p = (9cm,-2cm) .. (8cm,0) .. (11cm,-1cm) .. (10cm,-1.5cm);
  draw p;
  pickup pencircle scaled 1;
  n:=12;

%---ciclo1, archi 2 e 3, punti rossi
 for t=1.25 step .25 until 3:
    fill fullcircle  withpen pencircle scaled 3bp
                shifted point t of p withcolor red; endfor;
%---ciclo2, frecce tg, archi 2 e 3
 for t=1 step length(p)/n until length(p):
    drawarrow (point t of p) -- .8cm*unitvector(direction t  of p)
                 shifted point t of p ; endfor;
%---ciclo3, anelli verdi agli estremi dei tre archi di p
 for t=0 upto 3:
    draw fullcircle  withpen pencircle scaled 5bp
              shifted point t of p withcolor green; endfor;
%---ciclo4, divide il primo arco con barrette normali alla curva
 for t=0 step .25 until 1:
   draw( (-3,0)--(3,0)) rotated (90+angle direction t of p)
          shifted point t of p withcolor red; endfor;

%-----
```

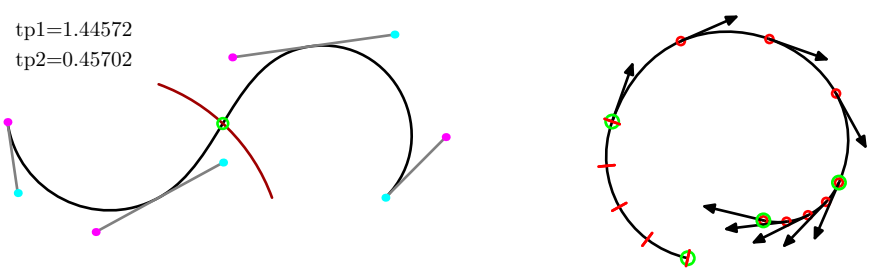

Fig. 8 Percorsi, controlli, intersezioni, suddivisioni e tangenti.

5 Trasformazioni del piano

Una trasformazione lineare è una funzione che porta gli elementi di un insieme V in un insieme W, $T : V \longrightarrow W$. Se A sono i punti di un insieme V si dice che T trasforma ogni punto $x \in A$ in $T(x)$, dove $T(x)$ è l'immagine di x in T.
Le trasformazioni di METAPOST possono essere applicate ai tipi `path`, `picture`, `pen`. Le equazione di una trasformazione del piano hanno sei coefficienti per cui dati tre punti e i loro corrispondenti in T, METAPOST è in grado di individuare le equazioni o la matrice della trasformazione. Oppure si possono assegnare i sei coefficienti della matrice della trasformazione. Questi metodi saranno trattati nei prossimi esempi.
Nel caso di trasformazioni notevoli come traslazione, dilatazione, rotazione esistono specifici comandi che rendono molto semplice applicare le trasformazioni.

5.1 Esempio1

È assegnato il triangolo di vertici `A=(0,0), B=(3u,0), C=(1.5u,3u),` sul quale si agisce con tre distinte trasformazioni che restituiscono i triangoli corrispondenti. Nella `Fig.9` il triangolo `ABC` è tracciato in nero, quelli che gli corrispondono in T sono rispettivamente verde, rosso blu. Vengono anche considerate una circonferenza e alcune rette.

- Trasformazione `T`

  È individuata da tre coppie di punti che si corrispondono in `T`
  ```
  (-1u,0) transformed T = (-1u,0);
  (0,1u)  transformed T =  (0,1u);
  (0,0)   transformed T =(-1u,1u);
  ```
  che METAPOST determina applicandola al triangolo assegnato.
  Volendo conoscere la matrice di `T` si può farlo applicando le tre coppie di punti all'equazione matriciale delle trasformazioni del piano

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}.$$

  Si trova che la `T` è composta da una simmetria rispetto alla bisettrice del primo e terzo quadrante, di matrice $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, e da una traslazione di vettore (−1,1); trasformazioni che si compongono nella simmetria rispetto alla retta (di punti uniti) di equazione y=x+1 e presente nel grafico come tratteggiata e che sono espresse dall'equazione

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

  che, moltiplicando e addizionando, può essere espressa in forma matriciale compatta o come semplici equazioni algebriche che applicate restituiscono il triangolo verde:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} y-1 \\ x+1 \end{pmatrix} \Longleftrightarrow \begin{cases} x' = & y-1 \\ y' = & x+1 \end{cases}$$

```
%----------
u:=2cm; pair A,B,C; path q;
q=fullcircle scaled 2u;
draw q  withpen pencircle scaled 1.5 withcolor red;
%---
transform T;
  ...
transform T';
  ...
transform T'';
  ...
%---
```

```
A=(0,0); B=(3u,0); C=(1.5u,3u);
path p;
p=A--B--C--cycle;
draw p withpen pencircle scaled 1.5;
draw p transformed T withcolor .625green;
draw p transformed T' withcolor .625red;
draw p transformed T'' withcolor blue;

drawarrow (-1.2u,0) -- (4u,0) withpen pencircle scaled 1.5;
drawarrow (0,-1.2u) -- (0,4u)withpen pencircle scaled 1.5;
draw (-1.5u,-.5u) -- (3u,4u) dashed evenly withcolor .5white;
draw (-1.5u,u) -- (4u,u) dashed evenly withcolor .5white;
%----------
```

- Trasformazione `T'`

  Dalla corrispondenza della seconda terna di coppie di punti
  `(1u,1u) transformed T' =(1u,1u);`
  `(0,1u)  transformed T' = (0,1u);`
  `(2u,2u) transformed T' = (2u,0);`
  METAPOST ottiene l'equazione $\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ 2 \end{pmatrix}$ che formalizza una simmetria rispetto all'asse x e una traslazione (0,2) che si compongono nella retta y=2 presente nel grafico come tratteggiata. Eseguendo moltiplicazione e addizione l'equazione diventa $\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ -y+2 \end{pmatrix}$ da cui si ottiene il triangolo rosso.

- Trasformazione `T''`

  METAPOST riconosce i coefficienti matriciali di una trasformazione solo se hanno la forma di variabili riconosciute dal proprio sistema di calcolo e pertanto l'equazione matriciale appropriata dovrà avere la forma

  $$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} xxpart & xypart \\ yxpart & yypart \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} xpart \\ ypart \end{pmatrix}.$$

  Nel caso di `T''` i sei coefficienti sono:
  `xpart T'' = ypart T'' = 1.5u;`
  `xxpart T'' = .6;`
  `yxpart T'' = 0;`
  `xypart T'' = 0;`
  `yypart T'' = -.3;`
  per cui l'equazione è $\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 0.6 & 0 \\ 0 & -0.3 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 1.5 \\ 1.5 \end{pmatrix}$.
  `T''` non è una trasformazione notevole, però si può notare che i due versori del riferimento creato da `T''` sono i'$(\frac{3}{5}, 0)$ e j'$(0, -\frac{3}{10})$ con una conseguente riduzione delle distanze e l'inversione dell'asse y ben evidenti nel triangolo blu.
  L'affinità individuata da `T''` è messa in evidenza anche applicandola alla circonferenza di raggio u e centro O che si trasforma in un'ellisse e che mostra il cambiamento del verso dell'asse y con il posizionamento di D(0,u) in `T''`. Dalla matrice si può ricavare che le rette `x=15/4u`, `y=15/13u` sono rette di punti uniti. METAPOST conferma tracciando le posizioni di due punti delle rette unite, `U` e `R`, nel centro del cerchio verde e quella dei corrispondenti in `T''` con il punto nero che si sovrappone a quello verde.

```
%----------
pair D,U,R,R';
D=(0,u);U=(15/4u,15/13u);R=(15/4u,0);
dotlabel.ulft("D",D);
dotlabel.ulft("",R)  withpen pencircle scaled 5  withcolor green;
```

```
dotlabel.top("",U) withpen pencircle scaled 5 withcolor green;
draw q transformed T''  withcolor red;
D:= D transformed T'';
U:= U transformed T'';
R':= R transformed T'';
dotlabel.top("D",D);
dotlabel.top("U",U);
dotlabel.top("R",R);
label.rt(btex$\scriptstyle T:\ \ t.angolo\ nero\ \rightarrow\ t.angolo\ verde$ etex, (3u,3u));
label.rt(btex $\scriptstyle T':\  t.angolo\ nero\ \rightarrow\ t.angolo\ rosso$ etex, (3u,2.75u));
label.rt(btex $\scriptstyle T'': t.angolo\ nero\ \rightarrow\ t.angolo\ blu$ etex, (3u,2.5u));
label.rt(btex $\scriptstyle y=x+1$ etex, (3u,4u));
label.rt(btex $\scriptstyle y=2$ etex, (4u,u));
%----------
```

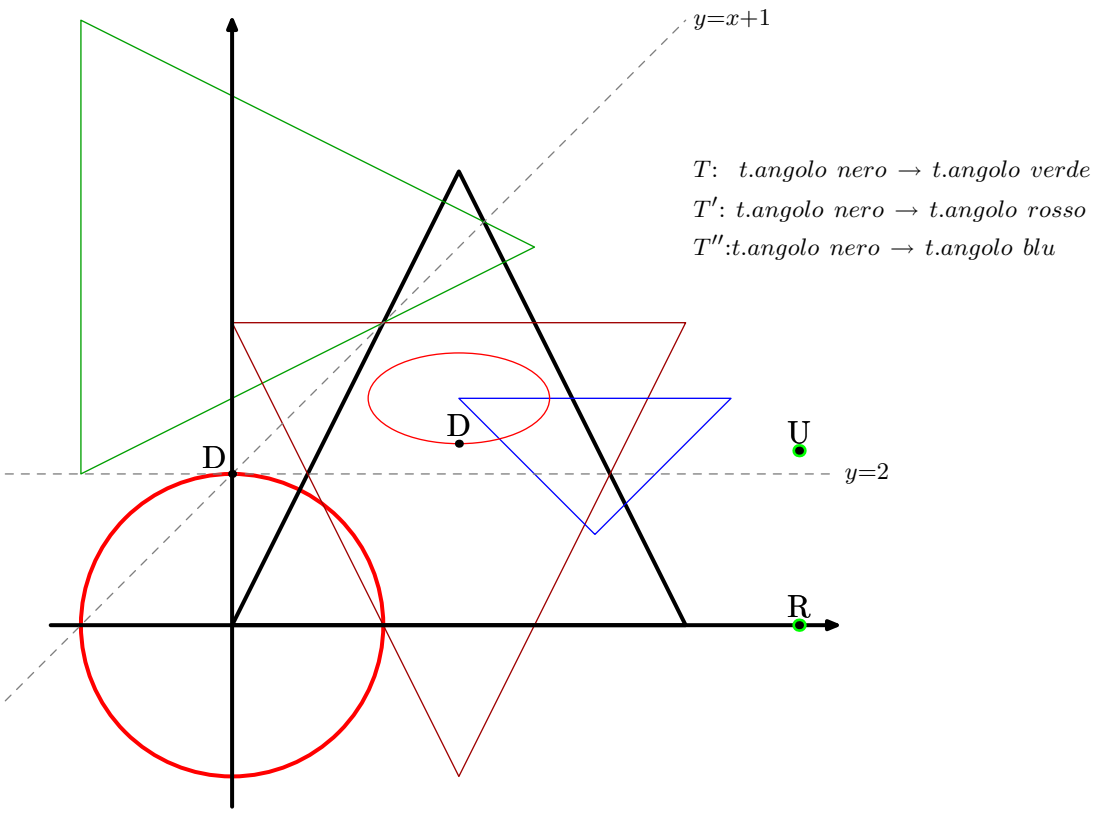

Fig. 9 Isometrie e affinità.

5.2 Esempio2

Il codice definisce un poligono regolare, un triangolo, e con `dir` usa le coordinate polari di METAPOST. Si definisce la trasformazione che sposta i vertici del poligono lungo un lato di appartenenza secondo un rapporto scelto, `Fig.10`, si traccia il poligono e si applica la trasformazione, e questo con un ciclo per un certo numero di volte, dando origine a figure di una certa eleganza del tutto affidate alle capacità e alla ottima strutturazione del motore di calcolo di METAPOST. A farlo personalmente sarebbe da perdersi già al secondo passaggio.

```
%--------
numeric r; r=1.7305; u:=3cm;
pair A,B,C,O; path p; O=(0,0);
```

```
A=u*dir(-30); B=u*dir(90);C=u*dir(210);
%B=(0,u); A=(u*r/2,-u/2);C=(-u*r/2,-u/2);
p=A--B--C--cycle; draw p;
%---
transform T;
  A transformed T =1/6[A,B];
  B transformed T =1/6[B,C];
  C transformed T =1/6[C,A];
%---
for i=0 upto 12:
 draw p;
 p:=p transformed T;
endfor
dotlabel.ulft("",0)  withpen pencircle scaled 5 withcolor green;
%--------
```

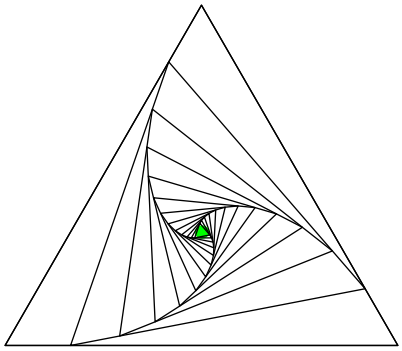

Fig. 10 Trasformazione iterata.

5.3 Esempio3

Il codice costruisce la classica sagoma del muso di un topolino che viene immessa in una variabile `picture`. Questa, che con il comando `addto topo contour` riempie il contorno di `fullcircle` con il colore assegnato, rende unitaria la figura formata da più elementi costruiti con i normali comandi per disegnare che vengono aggregati nella variabile di tipo `picture` dalla primitiva `addto`, `Fig.11`.

```
picture topo; topo := nullpicture;
addto topo contour fullcircle scaled 2u withcolor .6white;
addto topo contour fullcircle scaled u shifted (u*dir30) withcolor .6white;
 ...
```

Una volta costruita la `picture`, da una corrispondenza di tre coppie di punti si definisce la `T` che la trasformerà. Dalla sua equazione matriciale, che è $\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 0 & 3 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$, si possono ottenere varie informazioni: il punto unito $U\,(0{,}0)$, gli autovalori, che evidenziano l'azione dei fattori di dilatazione 2 e 3 applicati alle rette unite (ovviamente al segmento) che sono l'asse delle ascisse e la bisettrice del primo quadrante. Risultati che MetaPost conferma mostrando che le trasformate (verdi) di queste due rette si sovrappongono alle rette unite (nere). Facendo la stessa cosa per l'asse delle ordinate (asse di simmetria verticale) la retta trasformata (magenta) non si sovrappone all'originale.
Il `topo` originale è quella a destra.

```
%--------
u:=1cm; numeric r; r=0.707107;
path p,q,s;
p= (0,0) -- (2u,0);
q= (0,0) -- (2*r*u,2*r*u);
```

```
s= (0,0) -- (0,2u);
%---
picture topo;
  topo := nullpicture;
  addto topo contour fullcircle scaled 2u
        withcolor  .6white;%\MPcolor{tc};
  addto topo contour fullcircle scaled u
        shifted (u*dir30) withcolor  .6white;
  addto topo contour fullcircle scaled u
        shifted (u*dir150)
        withcolor .6white;
  addto topo contour fullcircle scaled .2u
         shifted (.5*u*dir30)  withcolor white;
  addto topo contour fullcircle scaled .2u
         shifted (.5*u*dir150)  withcolor white;
  addto topo contour fullcircle scaled .2u
         shifted (.4*u*dir-90)  withcolor .1white;
%---
  draw topo shifted (8u*dir0); %traccia la picture topo traslata a destra
  transform T;
  (0,0) transformed T = (0,0);
  (1,0) transformed T = (2,0);
  (0,1) transformed T = (1,3);

  draw topo transformed T ;
  draw p transformed T withcolor green;
  draw q transformed T withcolor green;
  draw s transformed T withcolor magenta;
  draw p;draw q;draw s;

def SampleText (expr s, f, c) =
draw s infont f scaled 5 shifted (3u,-3u) withcolor c ;
enddef ;
SampleText("Topolino", "\truefontname{RegularSlanted}", \MPcolor{tc}) ;\blank[big]
%--------
```

6 Proiezioni e il prodotto scalare

Un'altra azione geometrica importante è quella di proiettare un segmento su un altro, azione fondata sul concetto di prodotto scalare di due vettori, `Fig.12`.
Il codice (preso da André Heck, Learning METAPOST by Doing) prende in considerazione i vettori-punto `u1` e `u2` e costruisce la proiezione `u3` del primo vettore sul secondo e, definito il vettore `2*u2`, ne trova la proiezione `u4` su `u1`. Il prodotto scalare, per METAPOST `dotprod`, di `u1` e `u2` (che formano un angolo α), definito da `u1`$\cdot$ `u2`$= u_{1x} \cdot u_{2x} + u_{1y} \cdot u_{2y}$, è espresso anche dal prodotto della lunghezza della proiezione del primo vettore sul secondo per la lunghezza del secondo vettore, cioè: $u_1 \cdot u_2 =\mid u_1 \mid \cdot \mid u_2 \mid \cdot \cos\alpha$ per cui il rapporto $\frac{u_1 \cdot u_2}{|u_2|} =\mid u_1 \mid \cdot \cos\alpha$ esprime la (lunghezza della) proiezione di u_1 su u_2 mentre con $\frac{u_1 \cdot u_2}{|u_2| \cdot |u_2|} = \frac{|u_1|}{|u_2|} \cdot \cos\alpha$ quella lunghezza viene espressa in termini di u_2. Moltiplicando questo risultato per u_2, come avviene nel sorgente, la proiezione u_3 assume direzione e verso di u_2 e ha la lunghezza espressa in pt.
Su queste definizioni il sorgente effettua i suoi passi. L'oggetto principale del sorgente è la macro che viene introdotta da una `secondarydef`, dove secondary sta ad indicare che non si tratta di una macro primitiva MetaPost a cui è riservata la denominazione `primarydef`; le definizioni che distingono le macro MetaPost dalle altre permettono di immettere gli argomenti senza usare parentesi. Nel caso in questione la macro `projectedalong` prende due vettori, ne calcola il prodotto scalare che divide per il quadrato della lunghezza del secondo vettore ottenendo la proiezione di `v` su `w` espressa in termini di $\mid w \mid$. In effetti al primo passaggio, per trovare la proiezione di `u1` su `u2` si ottiene il numero 0.627 che la macro

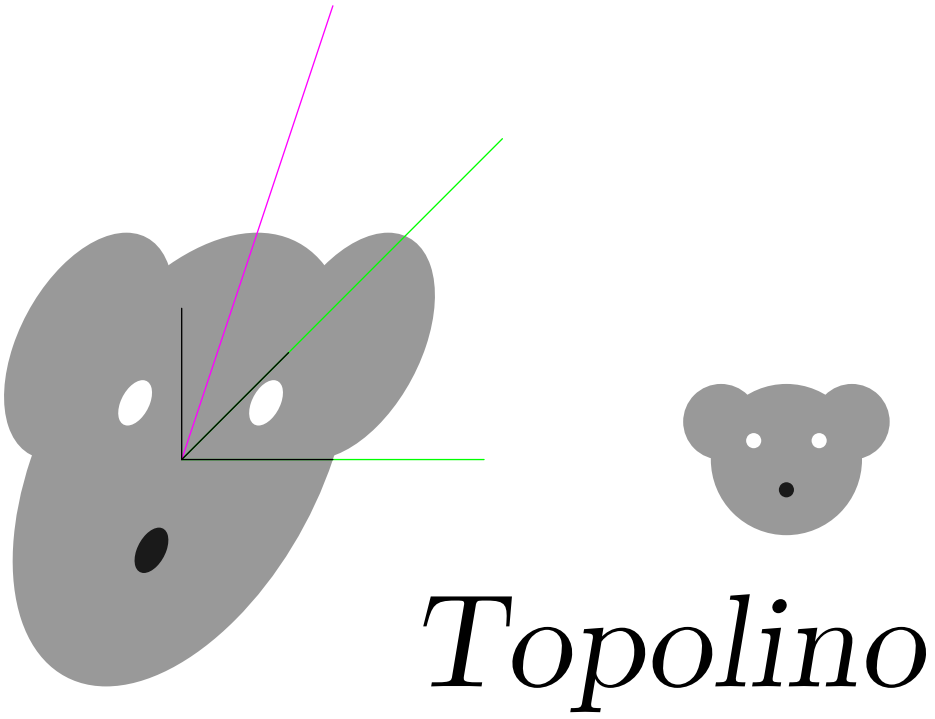

Fig. 11 Trasformazione affine.

moltiplica per | u_2 | ottenendo sia la lunghezza di `u3`, che è di 38.80pt, sia la direzione e il verso che sono quelli di `u2`. Lo stesso per la proiezione `u4`.

```
secondarydef v projectedalong w =
if pair(v) and pair(w):
(v dotprod w) / (w dotprod w) * w
else:
errmessage "arguments must be vectors"
fi
enddef;
%-----------
pair u[]; u1 = (20,80); u2 = (60,15);
drawarrow origin--u1;drawarrow origin--u2;drawarrow origin--2*u2;
u3 = u1 projectedalong u2;
u4 = 2*u2 projectedalong u1;
drawarrow origin--u3 withcolor blue;
draw u1--u3 dashed withdots;
```

Segue la riga che visualizza il segno di angolo retto che in questo caso è traslato in `u4`, le lunghezze ridotte a 6pt e la direzione-verso opposta a quella del vettore `u1`. Similmente per l'angolo retto su `u3`.

```
draw ((1,0)--(1,1)--(0,1)) zscaled (6pt*unitvector(-u1)) shifted u4;
```

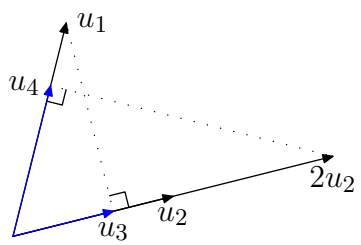

Fig. 12 Proiezioni ortogonali.

7 Curve e funzioni

Le curve parametriche e le equazioni sono un punto di forza di METAPOST. Non si possono però trascurare i tracciamenti di linee che rappresentano i punti di una funzione o di una curva espresse da relazioni matematiche o da una raccolta di dati. Di seguito si vedranno alcuni esempi.
Per il tracciamento di linee basate su raccolte di dati è bene rifarsi direttamente *Drawing Graphs with MetaPost* di John D. Hobby.

7.1 Una curva in coordinate polari

Una tipica curva espressa da coordinate polari è la *cardioide*. Nel caso specifico le coordinate sono tali da accentuare la curvatura e l'asse di simmetria è verticale. In METAPOST le coordinate (r, α) sono espresse nello stesso comando che è $(1 + 2 \cdot \cos\alpha)\cdot$`dir`$(\alpha)$. Per riempire la curva con un colore occorre che sia un ciclo e la curva polare pur avendo il punto iniziale e finale che coincidono non è vista da METAPOST come continua. Per renderla continua è necessario dividere la curva in due distinti percorsi e creare un ciclo con l'apposito comando `buildcycle`. Dopo di che si interviene prima con il riempimento e poi con il contorno, ad evitare che questo risulti coperto dal colore di riempimento che è opaco. La rotazione crea una simmetria verticale e sopra questa curva si sistemano le scritte, `Fig.13`.
Un aspetto delle `label` è quello di poterle trattare come un tipo `picture` e inserirle in un box. I passi sono:

- dichiarare una picture: `picture pic`
- inserire nella picture il contenuto della label: `pic:= thelabel.rt( ... )`
- crea nella posizione prevista dalla label il box vuoto cancellando il sottofondo: `unfill bbox pic`
- disegna il contenuto della picture: `draw pic`
- se è il caso si disegna il bounding box della picture: `draw bbox pic`

```
%-------
u:=50; w:=5; h:=4;
alt:=h*u;
larg:=w*u;
picture pic[];
path p[], pp;

% -- il riquadro
pickup pencircle scaled 1;
draw (-.5larg,-.9alt)--(.5larg,-.9alt)--(.5larg,.2alt)--(-.5larg,.2alt)--cycle withcolor
.5white;
% -- la cardioide
p1:=(1+2*cosd (0))*u*dir(0) for i=1 upto 180:..(1+2*cosd(i))*u*dir(i) endfor;
p2:=(1+2*cosd (180))*u*dir(180) for i=181 upto 360:..(1+2*cosd(i))*u*dir(i) endfor;
pp:=buildcycle (p1,p2) ;
fill pp   rotated -90 withcolor .7yellow;
draw pp rotated -90  withcolor .3(green+blue);
pickup pencircle scaled .2;
for i=0 step 45 until 360:
drawarrow (0,0) -- (1+2*cosd(i))*u*dir(i)   rotated -90 withcolor .3(green+blue);
endfor
% -- le etichette
pic1:=thelabel(btex $\scriptstyle r=1+2\cos\alpha$ etex, (.15larg,-.45alt));
unfill bbox pic1; draw pic1;
pic2:=thelabel(btex $\scriptstyle Curva\ in$ etex, (-.151larg,-.352alt));
unfill bbox pic2; draw pic2;
pic3:=thelabel(btex $\scriptstyle coordinate$ etex, (-.142larg,-.403alt));
unfill bbox pic3; draw pic3;
pic4:=thelabel(btex $\scriptstyle polari$ etex, (-.179larg,-.456alt));
unfill bbox pic4; draw pic4;
%-------
```

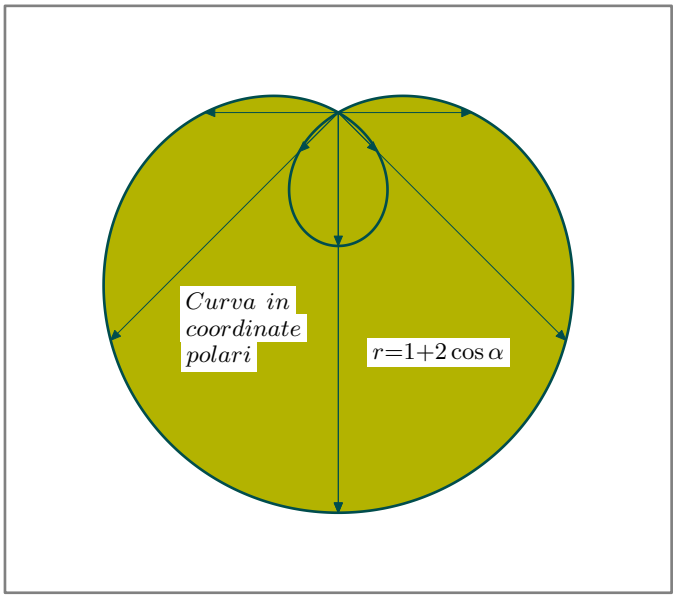

Fig. 13 Cardioide riferita all'origine.

Con il comando `dir` occorre fare attenzione al punto a cui fa riferimento, che di default è l'origine. Nella prima versione della cardioide l'origine è posta sull'asse di simmetria del riquadro e coincide con il punto angoloso della cardioide e i vettori `dir` vi fanno riferimento. Nella seconda versione, `Fig.14`, l'origine è il vertice basso a sinistra della riquadratura e `dir` dovrà a riferirsi a `z0=(.5*larg,.85*alt)`. Il codice della curva non cambia, ma a quello del riempimento e del tracciamento viene aggiunta la traslazione nel punto `z0`. Nel codice dei vettori che vogliono visualizzare il comando `dir` il primo `z0` che definisce

```
%---
drawarrow z0--(1+2*cosd(i))*u*dir(i) rotated -90 shifted z0 withcolor .3(green+blue);
%---
```

l'origine dei vettori potrebbe sembrare sufficiente (anche perché la cardioide è già correttamente posizionata) mentre i vettori sembrerebbero riferirsi a una curva ancorata all'origine del tipo `p1` e `p2`. Solo lo `shifted z0` mette le cose a posto (in proposito sarebbero graditi interventi). La traslazione in `z0` ovviamente vale anche per le etichette.

```
%-------
z0=(.5*larg,.85*alt);
%------ traccia il riquadro -----
pickup pencircle scaled 1;
draw (0,0)--(larg,0)--(larg,alt)--(0,alt)--cycle withcolor .5white;

%------ traccia la cardioide -----
p1:=(1+2*cosd (0))*u*dir(0) for i=1 upto 180:..(1+2*cosd(i))*u*dir(i) endfor;
p2:=(1+2*cosd (180))*u*dir(180) for i=181 upto 360:..(1+2*cosd(i))*u*dir(i) endfor;
pp:=buildcycle (p1,p2) ;
fill pp   rotated -90 shifted z0 withcolor .7yellow;
draw pp rotated -90 shifted z0 withcolor .3(green+blue);
pickup pencircle scaled .2;
for i=0 step 30 until 360:
drawarrow z0 -- (1+2*cosd(i))*u*dir(i)   rotated -90 shifted z0 withcolor .3(green+blue);
endfor
%-------
```

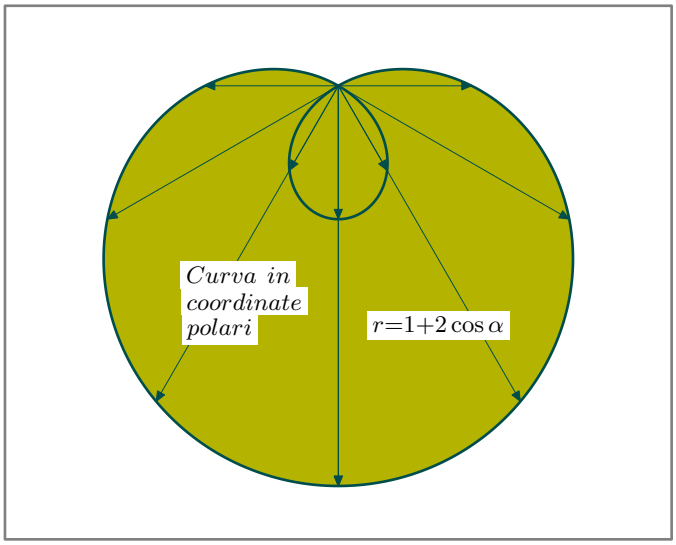

Fig. 14 Cardioide riferita al punto zo.

7.2 La funzione $y = x - 2\sin x$

La funzione viene tracciata e su una sua parte creato in ciclo. L'argomento di sin, una funzione MetaFun, è espresso in radianti e la sua variabile indipendente, al fine di esprimere le ascisse in π sarà moltiplicato per la costante `pi`.
In questo esempio l'equazione della funzione da tracciare è assegnato dalla `vardef f(expr x)=x-2*sin(x) enddef;` che verrà invocata dal ciclo `for`. L'argomento della funzione seno è espresso in radianti e la variabile che invoca la macro sarà moltiplicato per la costante `pi`, `Fig.15`.

```
%----
u:=28.125; alt:=13u; larg:=13u;
path r[],p; pair a[];
z0=(1.5u,1.5u); pi:=3.1415926; transform t;
t:=identity scaled u shifted z0;
%-- riquadro
pickup pencircle scaled 3;
draw (0,0)--(larg,0)--(larg,alt)--(0,alt)--cycle withcolor .7white;
%-- la maglia
pickup pencircle scaled 1;
for i=1,2,3:
 draw((0,3*i)--(3*pi,3*i)) scaled u shifted z0 withcolor .7white;
 draw((i*pi,0)--(i*pi,9)) scaled u shifted z0 withcolor .7white;
endfor
%-- assi cartesiani
pickup pencircle scaled 1;
drawarrow ((-.2,0)--(10,0)) transformed t;
drawarrow ((0,-.2)--(0,10)) transformed t;
%-- determina il p della funzione
p:= (0,0) for i=0.1 step .1 until 3:
..(i*pi,i*pi-2*sin(i*pi)) endfor;
%-- traccia p
p:=p transformed t ;
pickup pencircle scaled 2;
draw p withcolor blue;
%-- determina i percorsi da assemblare in un ciclo
r0=(1.5+.5*pi,1.5)*u--(1.5+2.5*pi,1.5)*u;
```

```
r1=(1.5+pi,1.5)*u-- (1.5+pi,10.5)*u;
r2=(1.5+2*pi,1.5)*u--(1.5+2*pi,10.5)*u;
r3=buildcycle(r0,r1,p,r2);
%-- intersezioni  curva/rette
a0=r1 intersectionpoint p;
a1=r2 intersectionpoint p;
%-- traccia il contorno e lo riempie
filldraw r3 withcolor .7yellow;
draw r3 withcolor .5(green+blue);
%-- le etichette
label.bot(btex $\pi$ etex,(pi,0) transformed t);
label.bot(btex $2\pi$ etex,(2*pi,0) transformed t);
label.bot(btex $3\pi$ etex,(3*pi,0) transformed t);
label.lft(btex $3$ etex,(0,3) transformed t);
label.lft(btex $6$ etex,(0,6) transformed t);
label.lft(btex $9$ etex,(0,9) transformed t);
label.rt(btex $y=x-2\sin x$ etex, (1,7) transformed t );
label.rt(btex $A=\frac32\pi^2+4$ etex, (pi+.4,3) transformed t );
%----
```

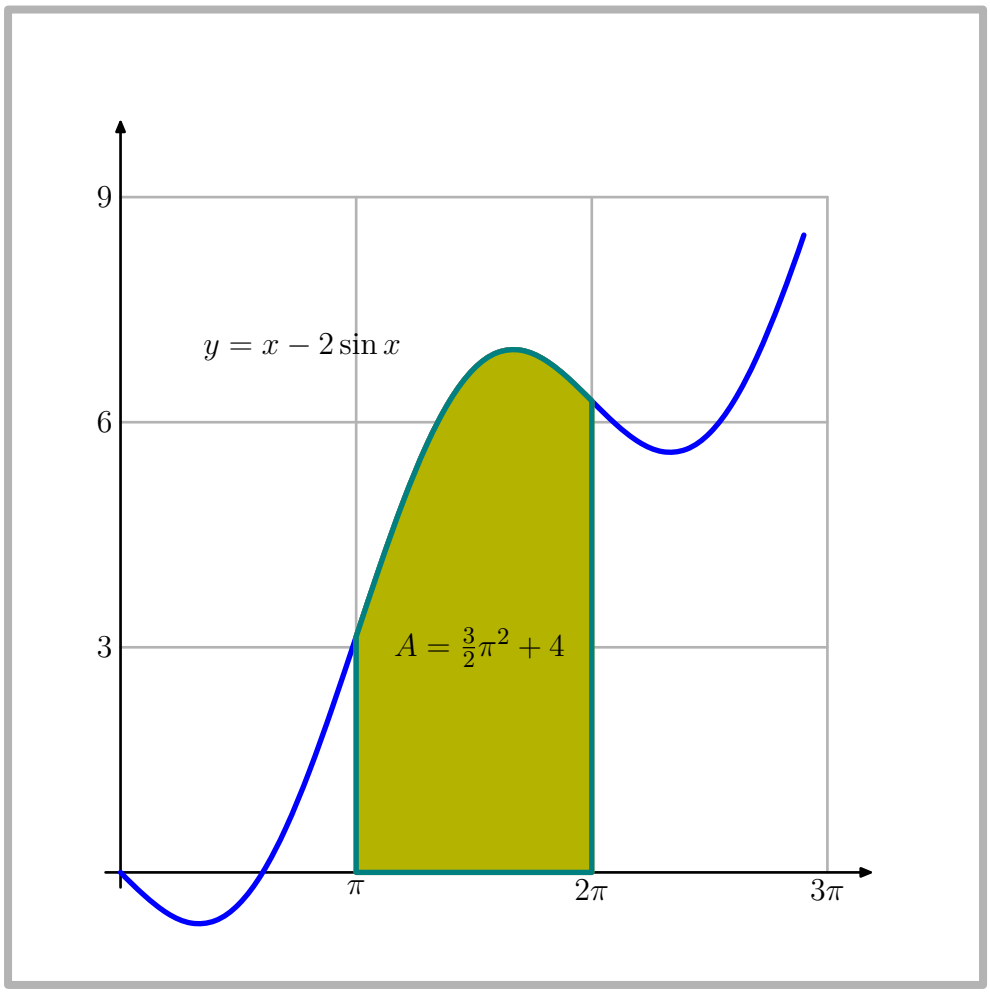

Fig. 15 La funzione $y = x - 2\sin x$.

7.3 Una clip su $y = x - 2\sin x$.

Il sorgente potrebbe differire dal precedente per la sola riga `clip currentpicture to r3;`. In effetti in questo caso l'equazione della funzione da tracciare è assegnato dalla definizione `vardef f(expr x)=x-2*sin(x) enddef;` che verrà invocata dal ciclo `for` ed anche il ciclo `for` che la invoca è leggermente diverso. Vengono anche aggiunte delle righe ad evidenziare la `clip`, ricostruito un riferimento cartesiano che aiuta a collocare la `clip` in cui sono inserite due righe di commento, `Fig.16`.
Il sorgente si limiterà alle parti non presenti nel precedente.

```
%----
xmin:=0;xmax:=3;xinc:=.1;
vardef f(expr x)=x-2*sin(x) enddef;
%-
p:= (xmin,f(xmin))*u for x=xmin+xinc step xinc until xmax:
.. (x*pi,f(x*pi))*u endfor;
%- righe oblique
pickup pencircle scaled .5;
r5= 0*dir(45)-- 300*dir(45);
for i=-20 upto 20:
draw r5 shifted (0,8*i)  withcolor .7white;
endfor
%- clip
clip currentpicture to r3;
%- assi cartesiani
pickup pencircle scaled 1;
drawarrow (-.2,0)--(10,0)  transformed t withcolor .7white;
for i=0 step pi*u until 3*pi*u:
    draw (i,u/10)--(i,-u/10) withcolor .7white;
    label.bot(btex $i*\pi$ etex,(i*pi,0) transformed t);
    endfor
drawarrow ((0,-.2)--(0,10)) transformed t withcolor .7white;
for i=1 step 3*u until 9*u:
    draw (u/10,i)--(-u/10,i) withcolor .7white;
    label.lft(btex $i*3$ etex,(0,i*3) transformed t);
    endfor;

label.rt(btex \vbox{\ttx\hbox {Il comando clip taglia tutto ci\`o che \`e esterno al}
 \hbox {ciclo; per visualizzare gli assi cartesiani occorre}
 \hbox {tracciarli successivamente al comando clip.}} etex, (1,8.5) transformed t );
%----
```

7.4 Iperbole in coordinate parametriche.

In questo esempio si vuole tracciare un'iperbole a partire dalle sue coordinate parametriche gudermanniane $(\frac{a}{\cos\alpha}, b\tan\alpha)$. Si è pensato ad un codice che possa tracciare l'iperbole con i fuochi sia sulle ascisse che sulle ordinate. Non c'è la possibilità di immettere il valore dei semiassi dall'esterno, ma intervenendo sulla `i` che sta per immaginario e su `r`, che sta per reale in effetti si può decidere quale caso dovrà essere rappresentato. Indicando come reale la seconda `vardef`, quella con tan, l'asse reale è quello delle ordinate. E viceversa. Scambiando tra loro i valori assegnati ai parametri portandoli da `a=5` e `b=8` a `a=8` e `b=5` la forma dell'iperbole cambia pur rimanendo invariato l'asse reale, `Fig.17`. I valori di `a` e `b` possono variare in un intervallo ragionevole; i numeri ad una cifra sono compatibili con il sorgente.
Si può notare anche che il valore di scala viene rappresentati sempre sull'asse immaginario e che la maglia può cambiare unità.
Per non vedere linee esterne alla maglia si è provveduto con `clip`.

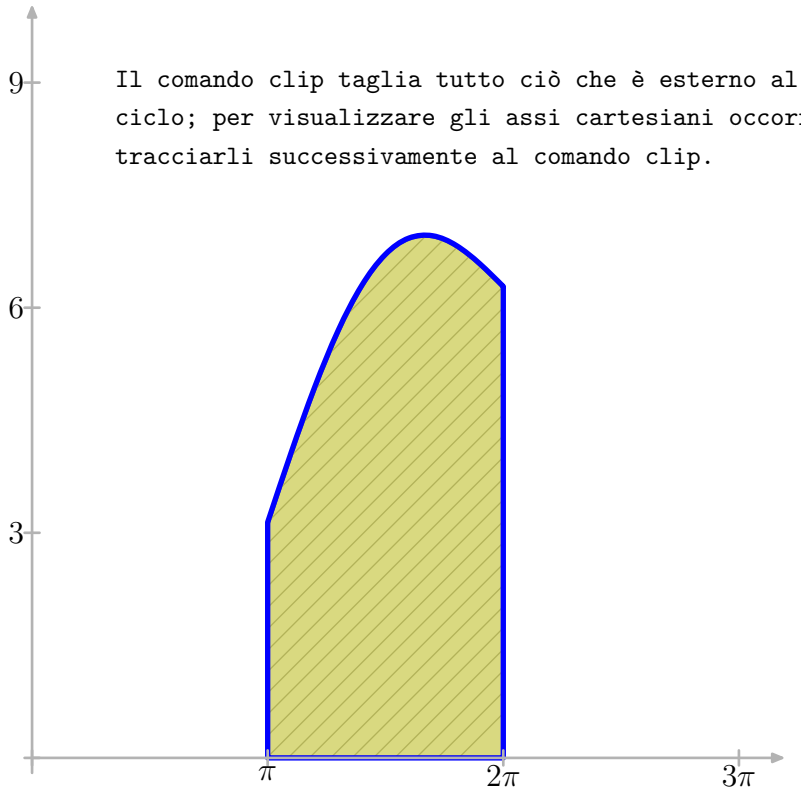

Fig. 16 Una clip sulla funzione $y = x - 2\sin x$.

```
%----
l:=360pt;
a:=8; b:=5; c:=sqrt(a*a+b*b);
d:=ceiling(c);
if ( odd(d)):
d:=d+5;
else: d:=d+4;
fi;
e:=d+1; u:=l/(2*e); alt :=2e*u; larg:=2e*u;
z0=.5(larg,alt);
path p[];
transform t;
t:=identity scaled u shifted z0;
%- la maglia
pickup pencircle scaled 1;
for i=-d step 2 until d :
 draw((-d,i)--(d,i)) transformed t  withcolor .8white;
 draw((i,-d)--(i,d)) transformed t  withcolor .8white;
endfor

%- definizione di variabili per calcolare le funzioni
vardef i(expr x)=a/cosd(x) enddef;
vardef r(expr x)=b*sind(x)/cosd(x) enddef;

%- Comportamento condizionato
if (r(0)= a) or (r(0)=b):
 p0:=(r(-70),i(-70))
for x=-69 step 1 until 70:
 ..(r(x),i(x))
```

```
 endfor;
 draw p0 transformed t  withcolor .7blue;
 draw ((-2*a,-2*b)-- (2*a,2*b)) transformed t;
 draw ((-2*a,2*b)-- (2*a,-2*b)) transformed t;
 p1:= p0 reflectedabout ((0,-2.b*u), (0,2*b*u));
 draw p1 transformed t withcolor .7blue;
 dotlabel.top("F", ((c,0)*u+z0));
 dotlabel.top("F'", (-c,0)*u+z0);
 dotlabel.ulft("V", (a,0)*u+z0);
 dotlabel.urt("V'", (-a,0)*u+z0);
 dotlabel.rt(decimal(a), (0,a)*u+z0);
 dotlabel.rt(decimal(-a), (0,-a)*u+z0);
else:
 if (i(0)= a) or (i(0)=b):
  p0:=(r(-70),i(-70))
  for x=-69 step 1 until 70:
   ..(r(x),i(x))
   endfor;
  draw p0 transformed t  withcolor .7blue;
  draw ((-2*b,-2*a)-- (2*b,2*a)) transformed t;
  draw ((-2*b,2*a)-- (2*b,-2*a)) transformed t;
  p1:= p0 reflectedabout ((-2*b,0),(2*b,0));
  draw p1 transformed t  withcolor .7blue;
  dotlabel.urt("F", (0,c)*u+z0);
  dotlabel.lrt("F'", (0,-c)*u+z0);
  dotlabel.lrt("V", (0,a)*u+z0);
  dotlabel.urt("V'", (0,-a)*u+z0);
  dotlabel.bot(decimal(a), (a,0)*u+z0);
  dotlabel.bot(decimal(-a), (-a,0)*u+z0);
 fi;
fi;
%- Fine del comportamento condizionato

p1:= (-d,-d)*u --(-d,d)*u --(d,d)*u--(d,-d)*u--cycle;
clip currentpicture to p1 shifted z0;
%
%------ gli assi cartesiani -----
pickup pencircle scaled 1;
drawarrow ((-d-.3,0)--(d+.3,0)) transformed t;
drawarrow ((0,-d-.3)--(0,d+.3)) transformed t;
%
%------ il riquadro -----
pickup pencircle scaled 3;
draw ((0,0)--(larg,0)--(larg,alt)--(0,alt)--cycle)  withcolor .7white;
%----
```

8 Soluzioni approssimate

Il controllo che le macro METAPOST esercitano sulle curve di Bézier rende semplice determinare gli zeri di queste curve approssimando l'equazione delle funzioni con una funzione lineare, la tangente. Iterando si ottengono risultati via via più precisi. Ma si tratta di un fare ridondante in quanto METAPOST con una sola riga di codice può trovare l'intersezione curva asse ascisse.
Si parte da un valore del parametro `t`, si determina il punto corrispondente della curva, si traccia la tangente determinandone l'intersezione col l'asse delle ascisse. La verticale per questo punto taglia la curva determinando il nuovo valore di `t`, e così via.

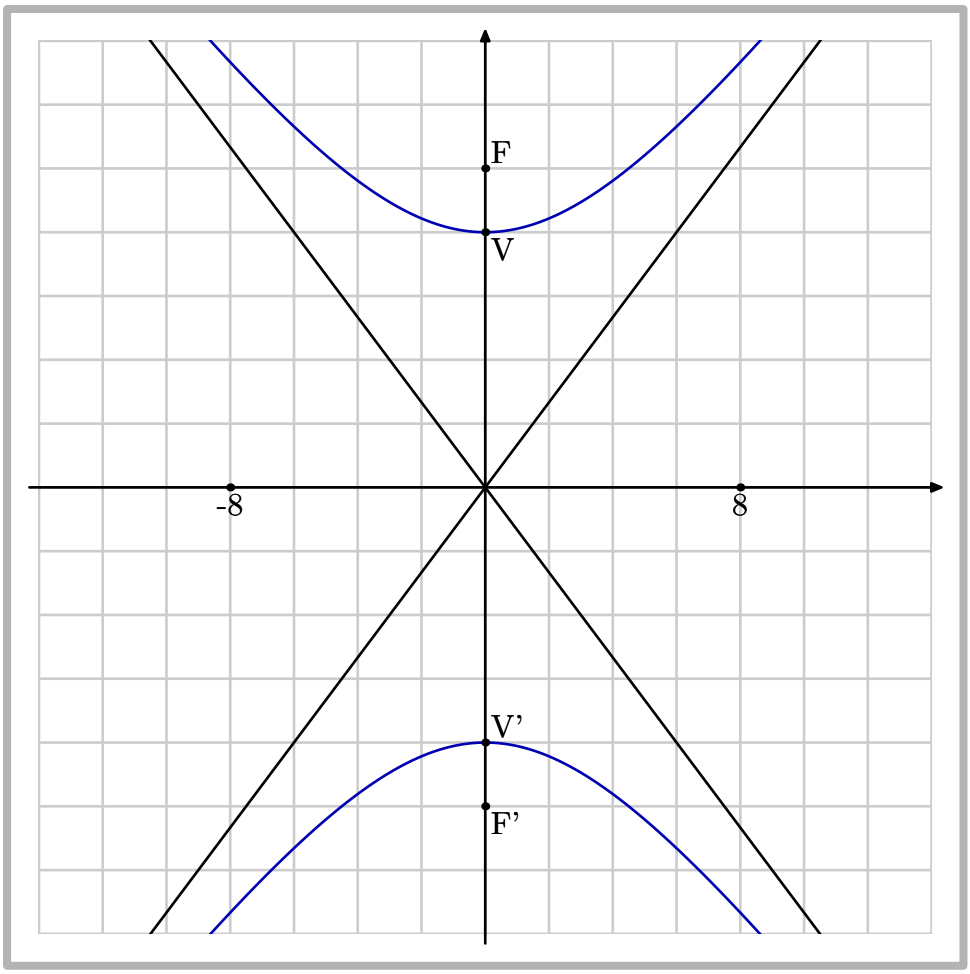

Fig. 17 Un'iperbole e i suoi parametri.

8.1 Curva di Bézier e valore approssimato dell'intersezione con l'asse x

Nel primo esempio si sviluppa un ciclo che con il metodo delle tangenti calcola la soluzione approssimata (x5). METAPOST nel caso di una curva di archi di Bézier può calcolare anche direttamente quella soluzione (x8), `Fig.18`.

Il codice visualizza gli incrementi e traccia la tangente nel punto `Po` a partire dal valore 0.8 del parametro `t`. L'unità di misura in `pt` è scelta per omogeneità con le risposte di METAPOST sulle intersezioni.

```
%--------------
u:=1pt; path c;
c = (0,-10){dir 0} .. (180,100);
draw c;
drawarrow (0,0) -- (185,0);
drawarrow (20,-30) -- (20,120);
numeric x[],t[];
t0 = .8;
pair p[];
p0=point t0 of c;
dotlabel.ulft("Po",p0);
for i=0 upto 5:
(x[i+1],0)=p[i]+whatever*direction t[i] of c;
p[i]=point t[i] of c;
```

```
(t[i+1],whatever)= c intersectiontimes ((x[i+1],-infinity)--(x[i+1], infinity));
fill fullcircle scaled 2pt shifted (x[i+1],0) withcolor red;
draw (xpart p[i] ,0)-- p[i] --(x[i+1],0);
endfor;

% La soluzione x5 = sesta iterazione, x8 l'intersezione curva asse x.
defaultscale := .8 ;
label.bot("x5="&decimal x5, (80,90));
(t8,whatever)= c intersectiontimes ((-infinity,0)--(infinity,0));
p8=point t8 of c;
label.bot("x8="&decimal xpart(p8), (80,98));
label.bot(btex $x_1$ etex, (x1,0));
label.bot(btex $x_2$ etex, (x2,0));
%--------------
```

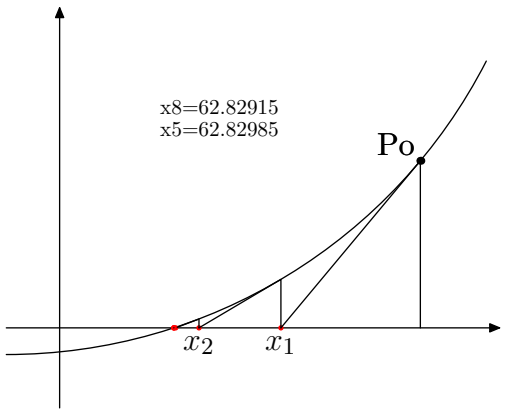

Fig. 18 Le intersezioni approssimate di una curva con l'asse x.

8.2 Funzione e valore approssimato dell'intersezione con l'asse x

Ciò premesso il metodo iterativo per la ricerca degli zeri di una curva approssimando con la retta tangente diventa più interessante se condotto su una funzione di equazione nota. La tangente in x_o ad una funzione f ha equazione $f(x)-f(x_o) = f'(x_o)(x-x_o)$ e taglia l'asse x in $x = x_o - \frac{f(x)}{f'(x_o)}$. Ma poiché stiamo facendo in modo che sia il grafico a dare le risposte, e non gli strumenti dell'analisi matematica, il coefficiente angolare della retta tangente verrà calcolato con il rapporto incrementale riferito ad un incremento della variabile indipendente *piccolo* invece che con la derivata. Si pone $f'(x_o) \approx \frac{f(x_o+h)-f(x_o)}{h}$, `Fig.19`. Le macro METAPOST per l'intersezione e per la tangenza possono ben visualizzare il senso di questa approssimazione.
L'incremento h della variabile indipendente è visualizzato da P_oL quello della funzione f da HL, quello della retta tangente da KL. Se h è *piccolo*, tanto da confondere funzione e tangente il rapporto incrementale, $\frac{\overline{HL}}{\overline{PoL}}$, potrà sostituire, graficamente parlando, la derivata. È quello che verrà fatto nell'ultimo esempio.
Seguono alcune righe di codice che rappresentano gli elementi del rapporto incrementale.

```
%--------------
u:=1pt;h:=40;path c;
c = (0,-10){dir 0} .. (150,100);
draw c;
drawarrow (0,0) -- (160,0);
drawarrow (20,-30) -- (20,120);
```

```
pair p[],L,H,K; numeric t[];
t0:=.6;
p0 = point t0 of c; L:=p0+(h,0);
(t1,whatever)= c intersectiontimes ((xpart L,-infinity)--(xpart L, infinity));
H=point t1 of c;
(xpart L, y1) = p0+whatever * direction t0 of c;
K = (xpart L, y1);
(x1, 0) = p0+whatever * direction t0 of c;
dotlabel.ulft("Po",p0); dotlabel.rt("L",L);
dotlabel.rt("K",K); dotlabel.rt("H",H);
draw p0--L--H; draw (x1,0)-- K;
%--------------
```

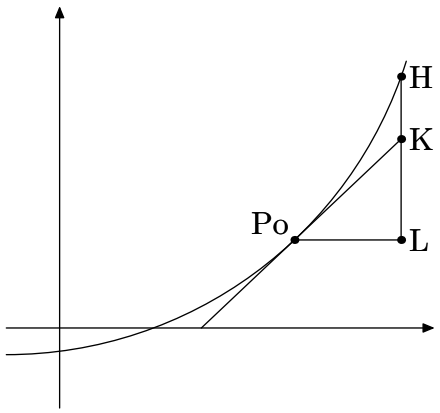

Fig. 19 La tangente e le soluzioni approssimate

Infine si considera la funzione $y = x - 1.2 \sin x$ che nell'intervallo $[0, \pi]$ interseca l'asse x. L'equazione associata non è direttamente risolvibile e lo zero è determinabile solo agendo con approssimazioni successive. Che vengono ottenute con tangenti iterativamente condotte, tangenti la cui direzione non è data dalla derivata nel punto ma, per restare in ambito grafico, calcolata dal valore del rapporto incrementale riferito all'incremento $h = 0.01pt$ della variabile indipendente, che METAPOST gestisce bene, `Fig.20`. Che nel tracciare le tangenti alle curve controllate dal parametro `t` si può pensare faccia qualcosa del genere.

```
%--------------
 u :=28.125*2; pi:=3.1415926;
 xmin:=0; xmax:=pi; xinc:=.1;
 transform t;
 t:=identity scaled u;
 path p;
 %
 vardef f(expr x) = x-1.2*sin(x) enddef;
 %
 drawarrow ( (xmin-.2,0) -- (xmax,0) ) transformed t;
 drawarrow ( (0,xmin-.2) -- (0,xmax) ) transformed t;
 %
 p:=(xmin,f(xmin))*u for x=xmin+xinc step xinc until xmax:
 .. (x,f(x))*u endfor;
 pickup pencircle scaled 1;
 draw p withcolor blue;
 %
 numeric x,x';
 x:=3; h:=0.01;
pickup pencircle scaled .5;
dotlabel.bot("",(x,f(x))*u) withcolor red;
```

```
for i=1 upto 5:
   x' := x - f(x)*h/( (f(x+h) - f(x)));
   draw ((x,f(x)) -- (x',0) -- (x',f(x'))) transformed t;
   x := x';
   dotlabel.bot("",(x,f(x))*u) withcolor red;
   dotlabel.bot(btex etex, (x,0)*u) withcolor red;
 endfor;
 defaultscale := .9 ;
 label.bot("x="&decimal x, (1.2,2)*u) ;
 label.bot(btex $y=x-\frac65\sin(x), [0,\pi]$etex, (1.2,2.5)*u) ;
%--------------
```

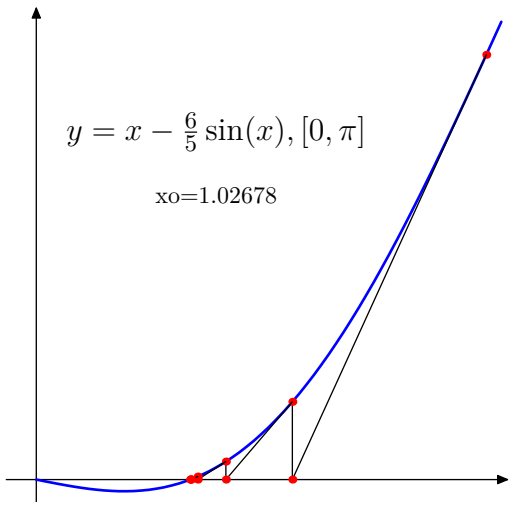

Fig. 20 Una funzione e la sua intersezione xo con l'asse x calcolata su basi grafiche.

8.3 Iterazioni e soluzioni approssimate

Nella sez. 7.3, `Fig.15`, è stata rappresentata la funzione $y = x - 2\sin x$ che interseca l'asse `x`, oltre che nell'origine, in un punto del primo quadrante la cui ascissa non è stata però presa in considerazione. Per determinarla occorrerebbe risolvere l'equazione $x - 2\sin x = 0$, che al contrario non è esplicitamente risolvibile. Poteva essere risolta nella sez. 8.2, `Fig.20`, ma trattandosi di un metodo per approssimazioni successive basato sulla grafica, l'equazione mancava di una base grafica soddisfacente tanto che è stata risolta un'equazione poco diversa, $x - \frac{6}{5}\sin x = 0$, ma graficamente meglio adatta allo scopo.
Del caso introdotto è possibile dare una soluzione oltre che con il metodo visto in `8.2` anche pensando che l'equazione ha per soluzione il punto comune alle funzioni $y = x$ e $y = 2\sin x$ che vengono rappresentate nella `Fig.21`. La loro intersezione può essere calcolata solo con approssimazioni successive che qui viene visualizzata sotto forma di una sorta di spirale formata da segmenti rettilinei e che tende a convergere verso il punto in questione. Scegliendo come valore iniziale $x = \frac{\pi}{2}$, a partire dal punto $(x, 2\sin x)$ si crea un percorso formato dalla spezzata che si sviluppa appoggiando i vertici di due lati consecutivi e perpendicolari uno sulla sinusoide, l'altro sulla bisettrice e il terzo di nuovo sulla sinusoide e via così. Si potrà notare che si tratta di un percorso iterativo ottenibile ripetendo -aggiornata- la cellula dei primi due segmenti. Alla base dell'iterazione è posta l'assegnazione $x := 2\sin x$ che x fa a se stesso assegnandosi un valore calcolato a partire dal proprio valore iniziale. Un po' il ragionamento ricorsivo di Zenone; una iterazione che non ha un termine, un ragionamento al *limite* (in questo esempio il limite esiste di fatto, dato dalla capacità di calcolo di METAPOST).
Sulla curva abbastanza ingrandita, `u=10cm`, si può notare come i punti della spirale si dispongono a forcella dell'intersezione, avvicinandosi ad essa man mano che si procede. L'ultima intersezione calcolata

alla ventesima iterazione ha i valori dell'ascissa e dell'ordinata che coincidono fino alla terza cifra decimale. La soluzione che per essere tale deve trovarsi anche sulla bisettrice dovrà avere ascissa e ordinata coincidenti; si può dire che il punto trovato ha millesimi certi e decimillesimi incerti. Guardando alla decima iterazione sono certi i centesimi e alla quinta le unità.
Segue il codice delle etichette per rappresentare nella `Fig.21` il valore delle coordinate della intersezione calcolato alle iterazioni n.ro 5, 10, 20:

```
%----------
 defaultscale := .8 ;
 label.rt (textext("{\switchtobodyfont[8bp]Coordinate 20mo punto iterazione:}"), (xpart
r[nit]-.26,1.54)*u);
 label.rt ("r.x[20]="&decimal(xpart r[nit]), (xpart r[nit],1.5)*u);
 label.rt ("r.y[20]="&decimal(ypart r[nit]), (xpart r[nit],1.46)*u);

 label.rt (btex  \rmxx Coordinate 10mo punto iterazione:etex, (xpart r[nit/2]-.26,1.41)*u);
 label.rt ("r.x[10]="&decimal(xpart r[nit/2]), (xpart r[nit/2],1.37)*u);
 label.rt ("r.y[10]="&decimal(ypart r[nit/2]), (xpart r[nit/2],1.33)*u);

label.rt (btex \rmxx Coordinate 5.to punto iterazione: etex, (xpart r[nit/4]-.26,1.28)*u);
label.rt ("r.x[5]="&decimal(xpart r[nit/4]), (xpart r[nit/4],1.24)*u);
label.rt ("r.y[5]="&decimal(ypart r[nit/4]), (xpart r[nit/4],1.20)*u);%----------
```

Prima di procedere col ciclo iterativo si mostra come si dovrebbe procedere per determinare le coordinate dei vertici della spirale, che verranno mostrati come punti ed etichette. Occorre anche premettere che per il calcolo della sinusoide viene definita la macro: `vardef sen primary x=(2*sind(x*rad)) enddef;` che permette di invocare direttamente la funzione seno e che, in quanto definizione primaria, può evitare di chiudere l'argomento della funzione tra parentesi.
Quanto alle etichette le coordinate, proposte tenendo conto della `vardef` ma non della iteratività, mostrano come cresce rapidamente e pericolosamente il loro codice:

```
%----------
 rad:= 180/pi;
 defaultscale := .7 ;
 x:=pi/2;
 dotlabel.top("Ao", (x,sen x)*u);
 dotlabel.rt("B1", (sen x,sen x)*u);
 dotlabel.urt("A1", (sen x,sen(sen x))*u);
 dotlabel.ulft("B2", (sen(sen x),sen(sen x))*u);
 dotlabel.top("A2", (sen(sen x),sen(sen(sen x)))*u);
 dotlabel.rt("B3", (sen(sen(sen x)),sen(sen(sen x)))*u);
 dotlabel.urt("A3", (sen(sen(sen x)),sen(sen(sen(sen x))))*u);
%----------
```

Al contrario con l'iterazione si parte dalla prima coppia di segmenti che, come le altre, viene direttamente disegnata a partire dai punti `(x,sen x), (sen x, sen x), (sen x, sen(sen x))` dopo di che si effettua l'assegnazione che aggiorna i dati per il ciclo successivo. La necessità dell'uso di `draw` nel codice dell'iterazione è dovuta al fatto che nella iterazione successiva le coordinate dei tre punti saranno sostituite da quelle degli estremi della nuova coppia di segmenti. Il codice del ciclo prevede anche la memorizzazione nell'array `r[]` delle coordinate del punto terminale di ciascuna iterazione, così che possano essere evidenziate successivamente, come fa il grafico.
Guardando al codice si nota che, a partire dalla seconda iterazione, per via dell'assegnazione `x:= senx` ogni elemento di un ciclo è definito in termini di quello precedente. Proprio come recita la definizione di successione ricorsiva. Ma nel nostro caso la ripetizione del ciclo, comandata da `for ... endfor`, è esterna al ciclo stesso mentre in un algoritmo ricorsivo, che in METAPOST sarebbe espresso da una `vardef ... enddef`, è la funzione a invocare se stessa. Inoltre un procedimento ricorsivo si caratterizza per immettere in uno stack, a partire dal basso e lasciando in sospeso le operazioni previste, gli elementi dell'intero processo iterativo che saranno elaborati una volta esaurite le istruzioni. Pertanto il procedimento in questione, per il fatto che i dati di un ciclo vengono sostituiti e quindi persi e che il ciclo è guidato dall'esterno, è iterativo.

I numeri usati esprimono la misura angolare dell'argomento del seno in radianti. Vista la scala della rappresentazione tipografica, u=10cm, si crea la corrispondenza 1rad =1dm per cui le coordinate del punto di intersezione sono certe fino al decimo di millimetro. Per un riscontro nel grafico delle ascisse occorre guardare all'origine che è posta a $\frac{\pi}{3}$ (1.0472dm).

```
%----------
nit:=20;
for i=1 upto nit:
 draw (x,sen x)*u -- (sen x, sen x)*u -- (sen x, sen(sen x))*u
  withpen pencircle scaled .3pt withcolor blue;
 x:= sen x;
 r[i]:=(x, sen x);
endfor;
%----------
```

Manca il codice del preambolo, quello dei risultati e quello delle coordinate degli assi cartesiani.

```
%----------
 u := 10cm;
 numeric pi; pi := 3.1415926;
 pair r[];
 numeric rad; rad:= 180/pi;
 numeric xmin, xmax, ymin, ymax, xinc;
 xmin := pi/3; xmax :=5/6*pi;
 ymin := 1;ymax := 2;
 xinc := 0.01;
 path p,q;
%--
 q:= ((xmin,xmin)--(xmax,xmax)*.8) scaled u;
 draw q;

 drawarrow ((xmin,1) -- (xmin,2.1))scaled u  ;
 for i= pi/3,ymin, sqrt 3,2:
 draw ((xmin,i)--(xmin-.01,i)) scaled u;
 endfor;

 drawarrow ((xmin,1) -- (xmax+.1,1))scaled u  ;
 for i= pi/3,pi/2,1.895,5*pi/6:
 draw ((i,1)--(i,.99)) scaled u;
 endfor;
%--
vardef sen primary x = (2*sind(x*rad)) enddef;
%--
 z0=(xmin,sen xmin)*u;

 p:= z0 for i= xmin+xinc step xinc until xmax:
     .. (i,sen i) scaled u
 endfor;
 draw p withcolor .625 red;

 ...

%--
labeloffset := 0.25cm;
label.bot(btex $\frac\pi 3$ etex, (xmin,ymin)*u);
label.bot(btex $\frac 56\pi$ etex, (5/6*pi,ymin)*u);
label.lft(decimal(ymin), (xmin,ymin)*u);
label.lft(decimal(ymax), (xmin,ymax)*u);
label.bot(btex $\frac\pi 2$ etex, (pi/2,ymin)*u);
```

```
label.lft(decimal(pi/3), (xmin,pi/3)*u);
label.lft(btex $\scriptstyle\sqrt 3$ etex, (xmin,sqrt 3)*u);
label.rt(btex $y=x$ etex, ((xmax,xmax)*.8)*u);
label.rt(btex $y=2\sin x$ etex, (.7*pi,1.65)*u);
label.bot(decimal(1.895), (1.895,ymin)*u);
%----------
```

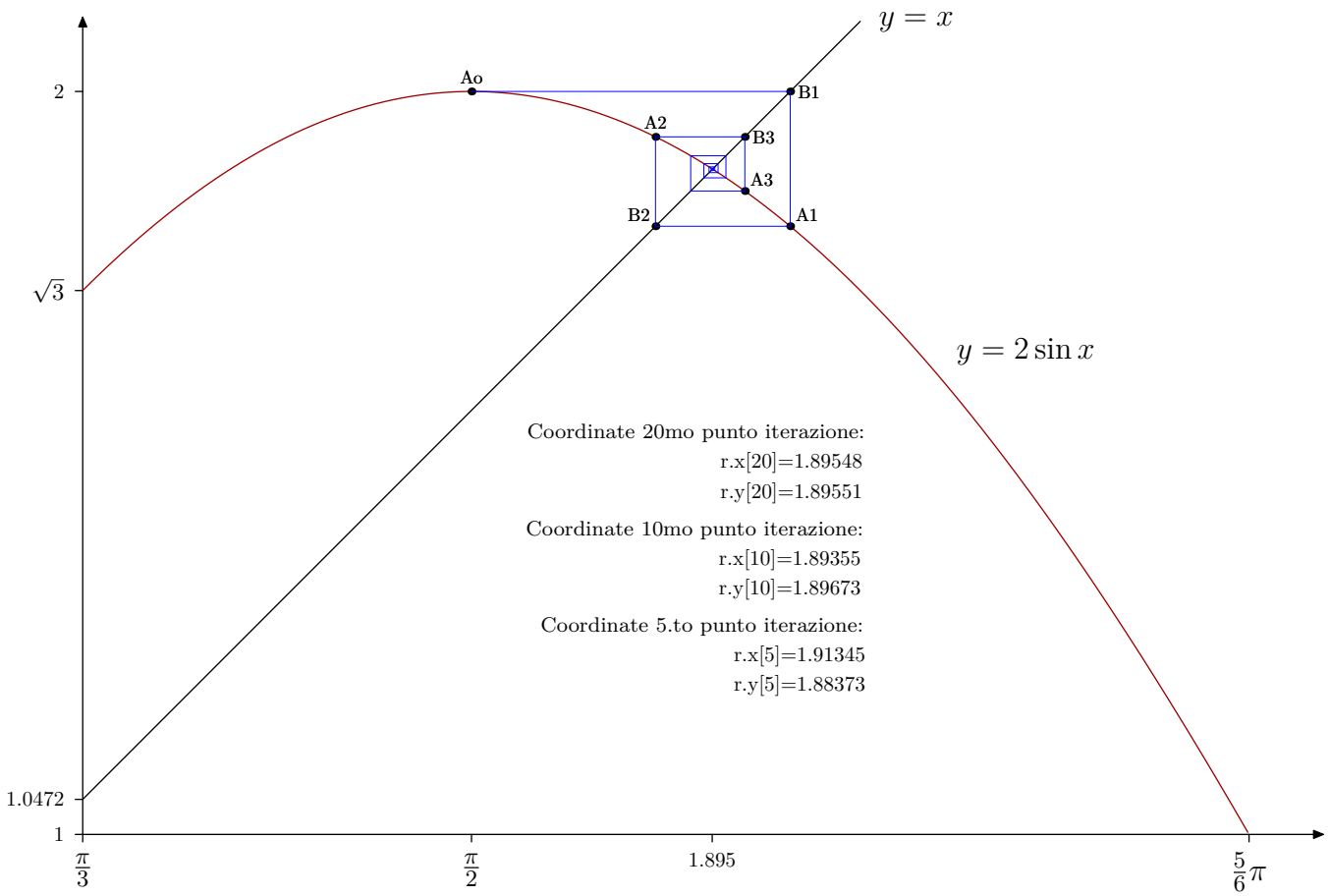

Fig. 21 Un ciclo iterativo per le soluzioni approssimate della equazione $x - 2\sin x = 0$.

9 Procedure ricorsive

Una tipica funzione ricorsiva ha almeno un riferimento esplicito a se stessa. Il limite al numero delle ricorsioni è dato dalla memoria disponibile. Il compilatore quando incontra un procedimento ricorsivo salva i dati intermedi e le istruzioni non risolte in uno stack, una pila in cui il primo dato è alla base della pila stessa.

9.1 Calcolo del fattoriale di n

Posto che la funzione ricorsiva sia `fatt(k)`, nel caso del calcolo di `4!` alla base dello stack avremo `4*fatt(3)` che non è calcolabile in quanto `fatt(3)` non è noto. E proprio `fatt(3)` invoca a sua volta la procedura in cui è inserito dando luogo all'espressione `3*fatt(2)`, anch'essa non calcolabile, che il compilatore posizione nella locazione di memoria immediatamente sopra a quella precedente. Con la stessa logica avranno luogo ancora due invocazione che creano nelle locazioni di memoria disponibili lo

statement `2*fatt(1)` e quello `1*fatt(0)` che essendo calcolabile chiude lo stack. Andando a ritroso il valore dell'ultima locazione serve a calcolare fatt(1)=1 e scendendo fatt(2)=2, fatt(3)=6 e fatt(4)=24. Il valore finale di `f`, posizionato come ultima istruzione della procedura, è 24 e può essere comunicato all'esterno della procedura con il comando `label.rt` vuotando lo stack.

```
%--
vardef fatt(expr k)=
 numeric f;
 if k=0:
  f:=1
 else:
  f:=k*fatt(k-1);
 fi;
 f
enddef;
label.rt("fatt(4)="&decimal fatt(4), (10,40));
%--
```

L'etichetta restituisce l'output:
Fattoriale(4)=24

9.2 MCD(n,m)

Una funzione ricorsiva meno nota è quella per il calcolo del MCD di due interi non negativi n e m. Se d è un divisore di n e m, q il quoziente di (n, m) e $r = n \bmod m$, essendo $r = n - q \cdot m$, d sarà divisore anche di r. E questo permette di risolvere il calcolo ricorsivamente.
$MCD(102, 136)$, con $n = 102$ e $m = 136$, invoca la funzione che scende alla prima istruzione creando $MCD(136, 102)$, che invoca nuovamente la funzione, le variabili sono $n = 136$ e $m = 102$, che scende fino all'ultima istruzione creando $MCD(102, 34)$, che invoca la funzione, le variabili $n = 102$ e $m = 34$, che scende all'ultima istruzione creando $MCD(34, 0)$, $n = 34$ e $m = 0$, che scende alla seconda istruzione trovando $MCD = 34$.

```
%--
vardef MCD(expr n,m)=
 numeric d;
 if m>n:
  d:=MCD(m,n)
 elseif m=0:
  d:=n;
  elseif n>=m:
  d:=MCD(m, n mod m)
 fi;
 d
enddef;
label.rt("MCD(102,136)="&decimal MCD(24,15), (10,20));
%--
```

L'etichetta restituisce l'output:
MCD(102,136)=34

9.3 Doppia ricorsione

L'uso della ricorsione per effettuare calcoli non è però centrale in METAPOST che ha una specifica vocazione per la grafica. E in questo campo ho voluto considerare l'esempio non molto noto neanche particolarmente complesso del tracciamento di un righello. Il mio riferimento sono state le pp. 54-58 di Algorithms, di Robert Sedgewick, Second Edition, Addison-Wesley, 1988. La procedura `righello` riportata sotto e adattata per METAPOST ha due successive istruzioni ricorsive che costruiscono un albero rovesciato che ad ogni ricorsione raddoppia il numero delle diramazioni. Il parametro `h` indica

il numero delle ricorsioni, `r` delle diramazioni dell'ultima ricorsione. Scopo principale del programma è la creazione di marcatori verticali (in numero dispari, uno per ogni nodo operativo) atti a dividere un dato righello in un determinato numero di parti. Siamo in un Paese di cultura anglosassone e il sistema metrico è fortemente influenzato da 2^n, il numero di rami prodotti dall'albero binario nella ennesima ricorsione, che decide anche il numero di parti in cui il segmento unitario sarà diviso. L'invocazione `righello(0,16,4)` applicata ad 1 inch lo divide in sedicesimi, la stessa divisione dei righelli in commercio. I marcatori sono 15, la marcatura centrale è la più alta, seguono quelle dei quarti e degli ottavi di inch fino ai sedicesimi; la procedura non traccia le marcature iniziale e finale che si deve pensare tracciate separatamente o, in un ideale righello materiale unitario, dall'inizio e dal termine del righello stesso.

```
vardef righello(expr l, r, h)=
  if h>0:
   m:=(l+r)div 2;
   mark(m,h);
   righello(l,m,h-1);
   righello(m,r,h-1);
  fi;
  enddef;
```

Per tracciare i segmentini divisori:

```
def mark(expr m,h)=
  draw(m,0)*u -- (m,h/4)*u withpen pencircle scaled .2pt ;
enddef;
```

Viste le cinque pagine dedicate alla procedura non è il caso di pensare di aggiungere alcunché sull'argomento. Ma con alcune restrizioni e piccoli cambiamenti quel procedimento è applicabile al nostro sistema metrico decimale per tracciare le divisioni in centimetri e in millimetri in un righello di 1 decimetro.
Le potenze di 2 non sono un ambito coerente con il sistema metrico decimale, ad esclusione delle prime due. Con `righello(0,4,2)`, vedi l'albero di `Fig.22`, che produce tre marcatori si dividerà il segmento lungo un centimetro in due parti marcandone il centro e i due estremi e questo sarà ripetuto altre nove volte. Separatamente e aggiungendosi alle marcature per il mezzo centimetro saranno tracciate le marcature dei millimetri. Ovviamente si tratta di una forzatura, ma dato che le ricorsioni sono algoritmi intriganti, si può prendere l'eredità anglosassone come spunto per accennare alla doppia ricorsione.
Il parametro i aggiunto a quelli della procedura originaria consente di iterare fino al decimetro.
Ma prima guardiamo al funzionamento della doppia procedura così come accade se la chiamata alla macro che traccia i marcatori precede entrambe le ricorsioni, al momento senza considerare affatto il quarto parametro.
Per poter rappresentare anche lo sviluppo temporale della logica della doppia ricorsione ci si può appoggiare all'albero di flusso binario della `Fig.22`.
Quando la `vardef` viene inizialmente invocata con l'invio di tre valori, nel nostro caso quelli di `righello(0,4,2)`, questi vanno ai tre parametri della `vardef` nell'ordine in cui sono argomenti. Nel momento in cui la `vardef` attiva il proprio procedimento per la prima volta le sue variabili saranno `l=0, r=4, h=2`. E guardando alla procedura si dice che la `vardef` percorre le successive istruzioni scorrendo verso il basso, facendo assumere ad `m` il valore 2, predisponendo lo statement `mark(2,2)` e rendendo disponibili le variabili `l=0, r=4, m=2, h=2` , che fungono da riferimento e nodo. A questo punto i due *comandi* `righello` costituiscono due possibilità da percorrere. Quella del primo righello rappresentata nel diagramma di flusso dal percorso di sinistra, quella del secondo righello dal percorso a destra.
È attivato per primo il `righello(l,m,h-1)` che assume i valori `(0,2,1)` e invoca la `vardef` facendo assumere alle sue variabili i nuovi valori `l=0, r=2, h=1` con cui la `vardef` percorrerà nuovamente verso il basso le istruzioni della procedura, nuovo valore per `m=1`, predisposizione della macro `mark(1,1)` rendendo disponibili le variabili `l=0, r=2, m=1, h=1` che permettono l'attivazione del primo `righello` nella forma `(0,1,0)`, che non supera la verifica `h>0` e si ferma.
Ora che il primo `righello` ha terminato il percorso, il nodo delle variabili `l=0, r=2, m=1, h=1` attiva il secondo `righello(m,r,h-1)` che, nella forma `(1,2,0)`, termina il percorso; lo stack si attiva, rende operative `mark(1,1)` e poi `mark(2,2)` e torna al livello delle variabili del nodo `l=0, r=4, m=2, h=2` che attivano il secondo `righello(m,r,h-1)`. Questo nella forma `righello(2,4,1)`, invoca la vardef che scende ponendo `m=3`, predisponendo `mark(3,1)`, attivando il primo `righello` nella forma `righello(2,3,0)`, che termina il percorso, e poi in secondo `righello` che, nella forma `righello(3,4,0)`, si ferma. Lo stack si attiva e rende operativa la macro `mark(3,1)`.

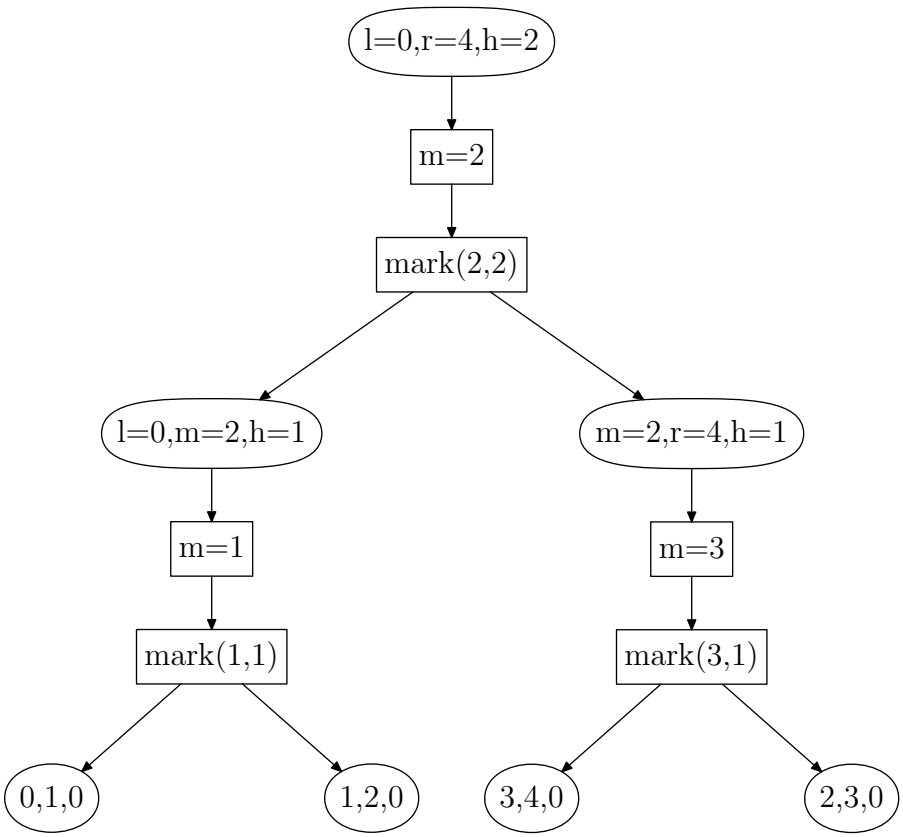

Fig. 22 Diagramma di flusso di una doppia ricorsione. L'ovale evidenzia l'elemento ricorsivo.

In una situazione di ripetizioni e di sovrapposizioni come quella vista considerare che gli eventi descritti avvengono in tempi successivi, con la variabile tempo vista come una freccia verso il basso, ci permette di rappresentare separatamente eventi che avvengono nello stesso luogo (stesso statement). È quanto fa il diagramma di flusso delle prime due ricorsioni.
Guardando al diagramma e sintetizzando la procedura ricorsiva appare evidente che l'obiettivo è quello di collocare nella posizione 2 il marcatore più alto e nelle posizioni 1 e 3, gli altri due di minore altezza. Considerato che nel caso del decimetro che si vuole costruire, `Fig.23`, l'unità di misura, u=0.5cm, il segmento in questione ha lunghezza 1cm e origine a 0.5cm. Inoltre, diversamente da quanto previsto nel righello di 1 `inch`, qui si tratta di posizionare i marcatori più alti agli estremi e quello di minor altezza al centro e questo avviene nella definizione `mark(expr m,h,i)` all'interno della istruzione condizionata. Poichè l'azione sul righello centimetrico va ripetuta, alle macro `mark` e `righello` viene aggiunto il parametro `i`. La ripetizione di `righello` avviene in uno specifico ciclo `for`, quello per il tracciamento della scala millimetrica in un altro.
Si può ancora osservare che le operazioni della ricorsione dipendono strettamente da quelle precedenti le quali debbono necessariamente essere memorizzate. Infatti l'esecuzione delle operazioni inizia dall'ultima operazione impostata, mentre la prima, alla base dello stack, sarà risolta per ultima. Lo spazio-tempo non ammette sovrapposizioni e la sua struttura è rappresentabile da un diagramma di flusso.

```
u := .5cm;
def mark(expr m,h,i)=
 if (m=2): k:=.2 elseif (m=1)or (m=3): k:=.4  fi;
 draw(m+i,0)*u -- (m+i,k)*u withpen pencircle scaled .2pt ;
enddef;

vardef righello(expr l,r,h,i)=
  if h>0:   %lo zero per la barretta iniziale
```

```
   m:=(l+r)div2;
   mark(m,h,i);
   righello(l,m,h-1,i);
   righello(m,r,h-1,i);
  fi;
  enddef;
 % Divide il centimetro in millimetri
 for k= 1 step 2 until 19:
 for i=.2,.4,.6,.8,1.2,1.4,1.6, 1.8:
 draw (i+k,0)*u -- (i+k,.1)*u withpen pencircle scaled .2pt withcolor .0white;
 endfor;
 endfor;
 % Invoca 10 volte la vardef che con una ricorsione
 % crea un segmento di 1 cm diviso a meta'
 for i=0 step 2 until 18:
  righello(0,5,2,i);
 endfor;
```

Fig. 23 Righello di 10cm, divisioni in mm.

```
%
% Il codice per il diagramma di flusso
%
vardef cuta(suffix a, b)=
 drawarrow a.c -- b.c cutbefore bpath.a cutafter bpath.b;
enddef;
input boxes
beginfig(001);
circleit.aa(btex \strut 0,4,2 etex);aa.dx=aa.dy;
boxit.bb(btex \strut m=2 etex);bb.dx=bb.dy;
boxit.a(btex \strut mark(2,2) etex);
circleit.ab(btex \strut 0,2,1 etex);ab.dx=ab.dy;
circleit.ac(btex \strut 2,4,1 etex);ac.dx=ac.dy;
boxit.bc(btex \strut m=1 etex);bc.dx=bc.dy;
boxit.b(btex \strut mark(1,1) etex);
boxit.cc(btex \strut m=3 etex);cc.dx=cc.dy;
boxit.c(btex \strut mark(3,1) etex);
circleit.abc(btex \strut 0,1,0 etex);abc.dx=abc.dy;
circleit.abb(btex \strut 1,2,0 etex);abb.dx=abb.dy;
circleit.bbc(btex \strut 3,4,0 etex);bbc.dx=bbc.dy;
circleit.acc(btex \strut 2,3,0 etex);acc.dx=acc.dy;

aa.s-bb.n=(0,20);
bb.s-a.n=(0,20);
a.s-ab.n=(90,40);
a.s-ac.n=(-90,40);
ab.s-bc.n=(0,20);
bc.s-b.n=(0,20);
ac.s-cc.n=(0,20);
cc.s-c.n=(0,20);
b.s-abc.n=(60,30);
```

```
b.s-abb.n=(-60,30);
c.s-bbc.n=(60,30);
c.s-acc.n=(-60,30);

drawboxed(aa,bb,a,ab,ac,bc,cc,b,c,abc,abb,bbc,acc);
cuta(aa,bb);
cuta(bb,a);
cuta(a,ab);
cuta(a,ac);
cuta(ab,bc);
cuta(bc,b);
cuta(b,abc);
cuta(b,abb);
cuta(ac,cc);
cuta(cc,c);
cuta(c,bbc);
cuta(c,acc);
endfig;
end.
```

10 Conclusioni

Al termine dell'esperienza la sensazione è che ConTeXt sia un sistema di composizione tipografica non so se ampio come LaTeX ma rispetto al quale offre un quadro di maggiore unitarietà; l'inserimento dei grafici che avviene direttamente, senza bisogno di moduli o pacchetti esterni semplifica non poco. Un documento semplice come il mio si ottiene utilizzando un terzo della sua capacità di strutturazione . Al contrario in certe operazioni, come ad esempio la bibliografia, in effetti ci si sente un po' troppo soli; gli esempi non sono il punto di forza di ConTeXt. Un altro aspetto molto positivo è che METAPOST è perfettamente integrato. E per un utente come me interessato (come autodidatta) ai linguaggi di programmazione non è poco. Le mie pagine in effetti si riducono ad un excursus in METAPOST, che mi si è proposto come un linguaggio snello e intelligente nel senso che in certi casi sembra prevenire i bisogni dell'utente. In particolare con le sue equazioni, le trasformazioni del piano e con il controllo dei percorsi curvi. Ad esempio, nel caso di sez. 3.4 stavo provando un'equazione per far determinare ad una macro le coordinate dell'incentro di un triangolo quando mi sono accorto che essendone presente anche un'altra METAPOST aveva di sua iniziativa risolto il sistema e questo mi sembra lo faccia normalmente. E poi il controllo che ha su un percorso curvo di Bézier! Lavorandoci ne sono diventato un tifoso. Forse un po' di questo piacere riesce a trasparire dalle presentazioni dei miei piccoli programmi.