# Laboratorio Informatico per l'Ingegneria Civile

Amabile Tatone

Elena Antonacci e Felice Fusco

4 maggio 2008

# Introduzione

Il carattere peculiare che di solito si attribuisce all'ingegnere è l'attitudine al calcolo, che consiste nell'utilizzare modelli astratti applicandoli a casi concreti. Si potrebbe dire: nel tradurre in una forma concreta l'astrazione. Questa attitudine, naturalmente, deve essere accompagnata, a diversi livelli, ad una capacità di motivare, giustificare e formulare modelli.

La relazione tra l'astratto e il concreto propria del *calcolo* e della *visualizzazione* rende questi anche dei potenti strumenti per lo studio. Uno stimolo essenziale alla comprensione di concetti astratti viene sempre fornito dall'esame di casi particolari. Spesso il risultato di un semplice calcolo ripetuto per casi diversi e accompagnato da qualche forma di visualizzazione fornisce anche ai più esperti occasioni di stupore.

Il Laboratorio Informatico vuole sviluppare, proprio all'inizio del percorso di studio, le abilità di calcolo e visualizzazione con cui realizzare quell'attività di sperimentazione che è essenziale nel processo di apprendimento. Come peraltro lo è un gioco, in cui l'eliminazione di ogni finalità utilitaristica giova alla creatività. Da questo punto di vista i *poligoni orari* e i *poligoni antiorari*, che si imparerà a creare e a deformare a piacimento, si trovebbero sicuramente a loro agio nel mondo fantastico di *Flatlandia*[1].

L'attività di laboratorio si svolge seguendo un percorso guidato che, attraverso esempi legati in qualche modo agli argomenti di studio del primo anno, fornisce gli elementi di base di `Scilab`. L'uso del linguaggio è finalizzato alla realizzazione di grafici di funzioni, al disegno di curve e di poligoni.

Non è richiesto alcun prerequisito per frequentare il laboratorio. Anche coloro che non hanno mai osato toccare una tastiera riceveranno l'assistenza necessaria a diventare indipendenti, almeno nello svolgimento degli esercizi.

Un test finale darà la possibilità di acquisire 6 crediti nella tipologia F, come indicato nel Manifesto degli Studi.

[1] Edwin A. Abbott, *Flatland: a Romance of Many Dimensions*, Princeton University Science Library 1991 (Dover 1953), traduzione italiana: *Flatlandia*, Adelphi, 1966.

# Autori

I moduli `Scilab` riportati nella parte I sono stati ideati e realizzati da Amabile Tatone. Rispetto alla versione originaria sono stati modificati più volte anche in base ai suggerimenti e alle osservazioni di Elena Antonacci e di Felice Fusco che hanno redatto la parte II e curato l'attività di laboratorio. Questa consiste nella presentazione e nella illustrazione dei moduli, nell'introduzione ai concetti generali e ai costrutti principali del linguaggio `Scilab`, nell'assistenza individuale agli studenti nelle esercitazioni. Questa attività si riflette sul contenuto del corso la cui evoluzione, oltre che dall'interazione con gli studenti, è indotta anche dagli stimoli provenienti dal contesto didattico.

Il corso si è svolto per la prima volta nel periodo Settembre-Dicembre 2005, primo quadrimestre dell'anno accademico 2005-2006.

# Indice

# Parte I

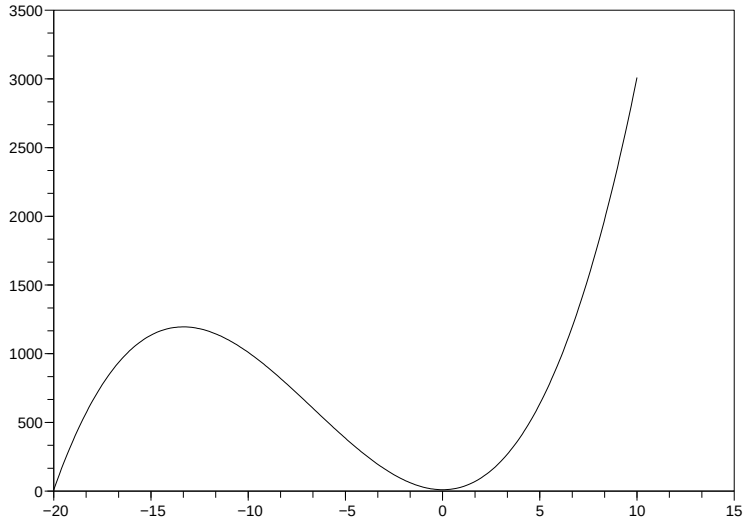
# Visualizzazione e calcolo con Scilab

# Capitolo 1

# Grafici di funzioni, secanti e tangenti

In questo capitolo si vede come disegnare il grafico di una funzione, come tracciare una retta secante e come ottenere l'approssimazione di una retta tangente in un punto qualunque del grafico.

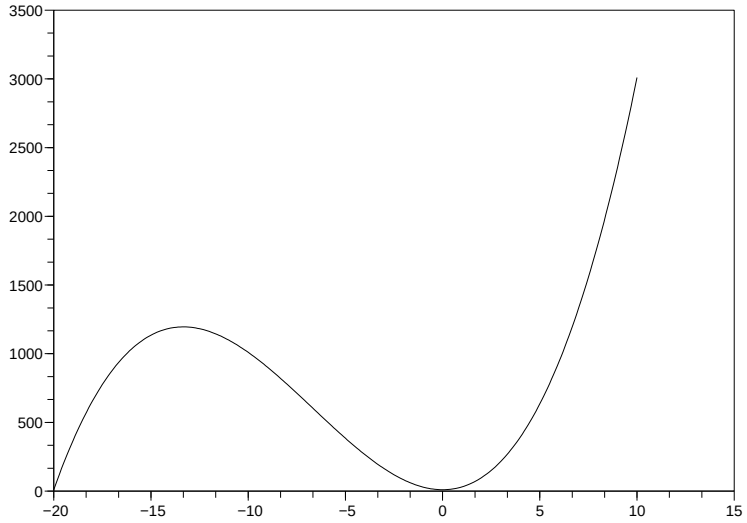
## 1.1 Grafico di una funzione (1)




```
//:grf0.sce
//
// Grafico di una funzione (1)
//
x=[-20:0.1:10];
y=10+20*x^2+x^3;
clf();
plot2d(x,y);
//.
```

[Download]

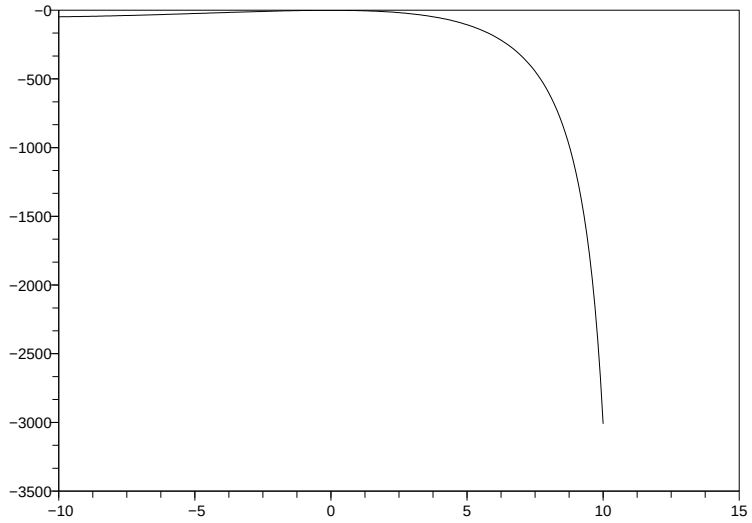
## 1.2 Grafico di una funzione (2)




```
//:grf1.sce
//
// Grafico di una funzione (2)
//
function y=f(x)
      y=10+20*x^2+x^3
endfunction;
x=[-20:0.1:10];
clf();
y=f(x);
plot2d(x,y);
//.
```

[Download]

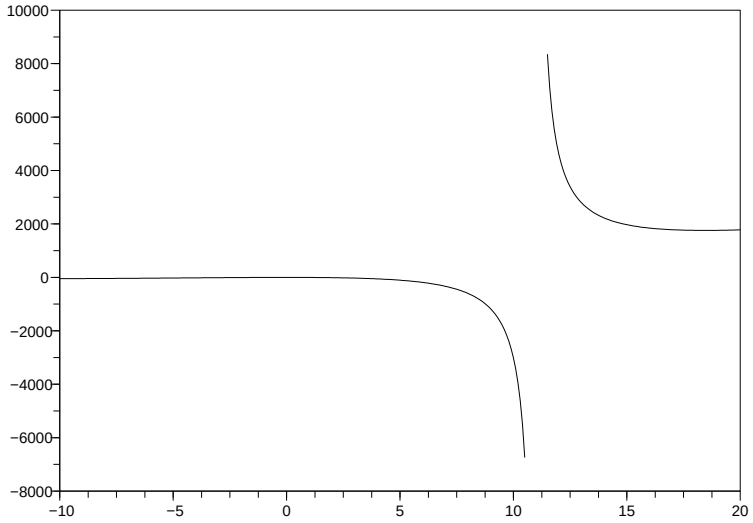
## 1.3 Grafico di una funzione (3)




```
//:grf2.sce
//
// Grafico di una funzione (3)
//
function y=f(x)
      y=(10+20*x^2+x^3)./(x-11)
endfunction;
x=[-10:0.1:10];
clf();
y=f(x);
plot2d(x,y);
//.
```

[Download]

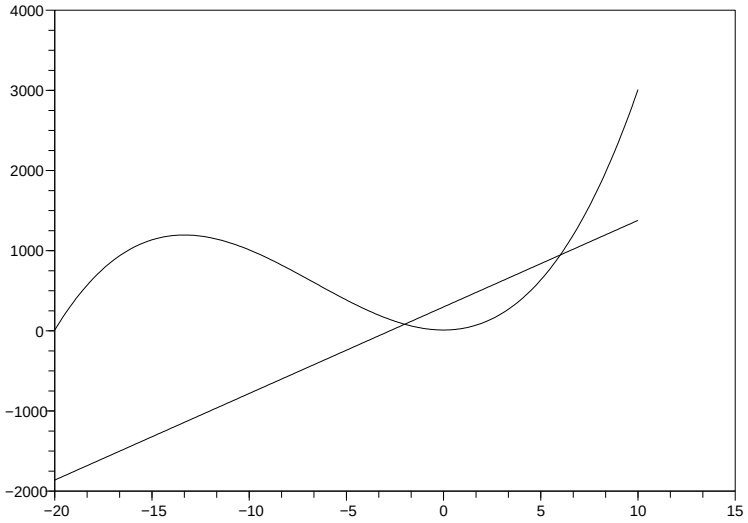
## 1.4 Grafico di una funzione (4)




```
//:grf3.sce
//
// Grafico di una funzione (4)
//
function y=f(x)
      y=(10+20*x^2+x^3)./(x-11)
endfunction;
x1=[-10:0.1:10.5];
x2=[11.5:0.1:20];
clf();
plot2d(x1,f(x1));
plot2d(x2,f(x2));
//.
```

[Download]

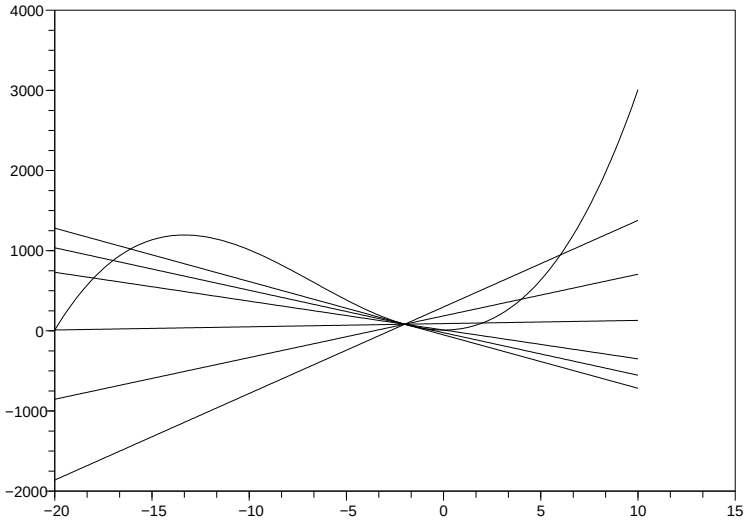
## 1.5 Secante al grafico di una funzione (1)




```
//:grsec1.sce
//
// Secante al grafico di una funzione (1)
//
function y=f(x)
      y=10+20*x^2+x^3,
endfunction;
clf();
x=[-20:0.1:10];
x0=-2;
x1=6
plot2d(x,f(x));
plot2d(x,f(x0)+((f(x1)-f(x0))/(x1-x0))*(x-x0));
//.
```

[Download]

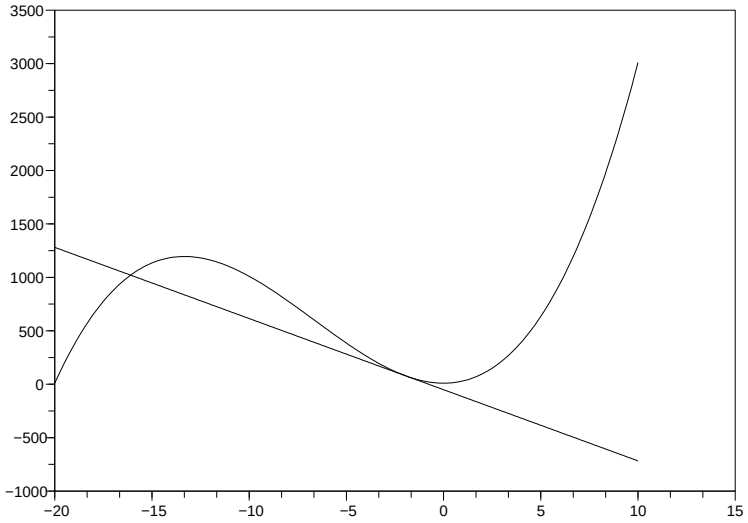
## 1.6 Secante al grafico di una funzione (2)




```
//:grsec2.sce
//
// Secante al grafico di una funzione (2)
//
function y=f(x)
      y=10+20*x^2+x^3,
endfunction;
function y=fsec(x,x0,x1)
      y=f(x0)+((f(x1)-f(x0))/(x1-x0))*(x-x0),
endfunction;
clf();
x=[-20:0.1:10];
x0=-2;
plot2d(x,f(x));
plot2d(x,fsec(x,x0,x0+8));
plot2d(x,fsec(x,x0,x0+6));
plot2d(x,fsec(x,x0,x0+4));
plot2d(x,fsec(x,x0,x0+2));
plot2d(x,fsec(x,x0,x0+1));
plot2d(x,fsec(x,x0,x0+0.1));
//.
```

[Download]

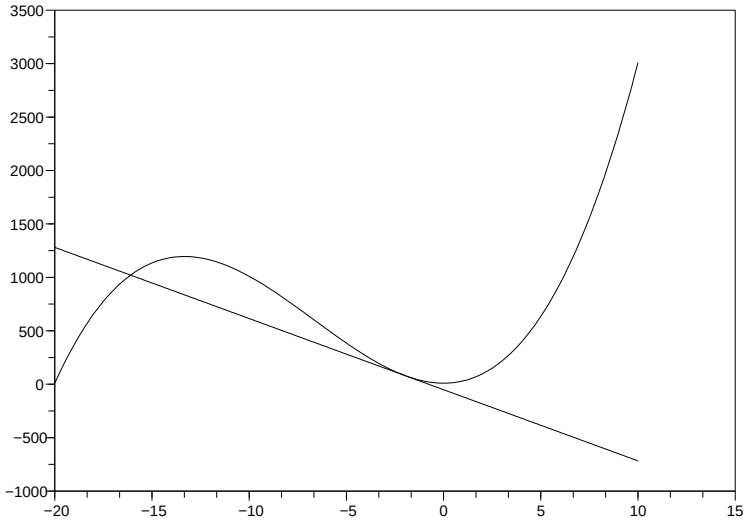
## 1.7 Tangente approssimata al grafico di una funzione (1)




```
//:grtan1.sce
//
// Tangente approssimata al grafico di una funzione (1)
//
function y=f(x)
      y=10+20*x^2+x^3,
endfunction;
function y=fsec(x,x0,x1)
      y=f(x0)+((f(x1)-f(x0))/(x1-x0))*(x-x0),
endfunction;
clf();
x=[-20:0.1:10];
x0=-2;
plot2d(x,f(x));
plot2d(x,fsec(x,x0,x0+0.1));
//.
```

[Download]

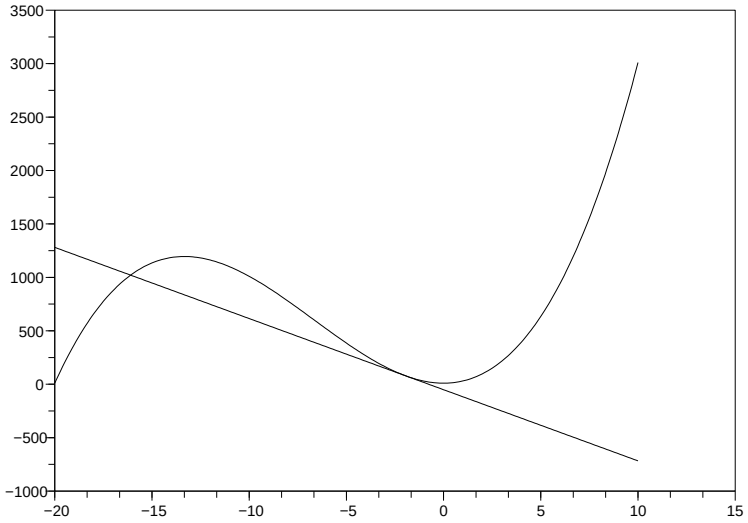
## 1.8 Tangente approssimata al grafico di una funzione (2)




```
//:grtan2.sce
//
// Tangente approssimata al grafico di una funzione (2)
//
function y=f(x)
      y=10+20*x^2+x^3,
endfunction;
function y=ftan(x,x0)
      h=0.1,
      y=f(x0)+((f(x0+h)-f(x0))/h)*(x-x0),
endfunction;
clf();
x=[-20:0.1:10];
x0=-2;
plot2d(x,f(x));
plot2d(x,ftan(x,x0));
//.
```

[Download]

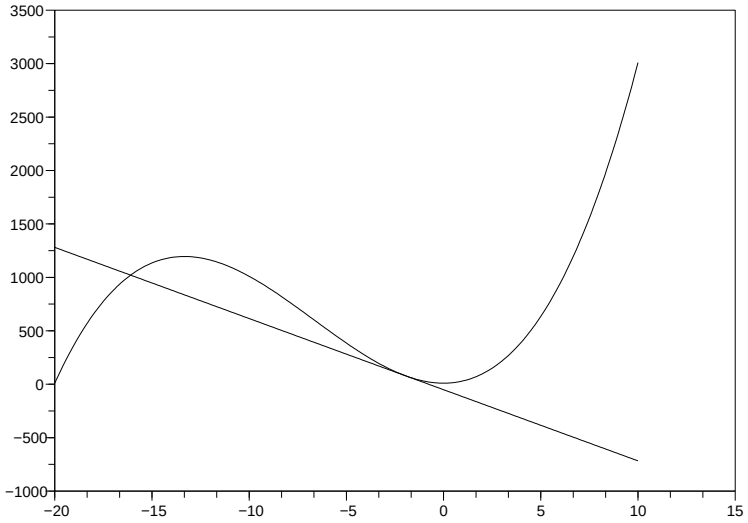
## 1.9 Tangente approssimata al grafico di una funzione (3)




```
//:grtan3.sce
//
// Tangente approssimata al grafico di una funzione (3)
//
function y=f(x)
      y=10+20*x^2+x^3,
endfunction;
function yp=fp(x0)
      h=0.1,
      yp=((f(x0+h)-f(x0))/h),
endfunction;
function y=ftan(x,x0)
      y=f(x0)+fp(x0)*(x-x0),
endfunction;
clf();
x=[-20:0.1:10];
x0=-2;
plot2d(x,f(x));
plot2d(x,ftan(x,x0));
//.
```

[Download]

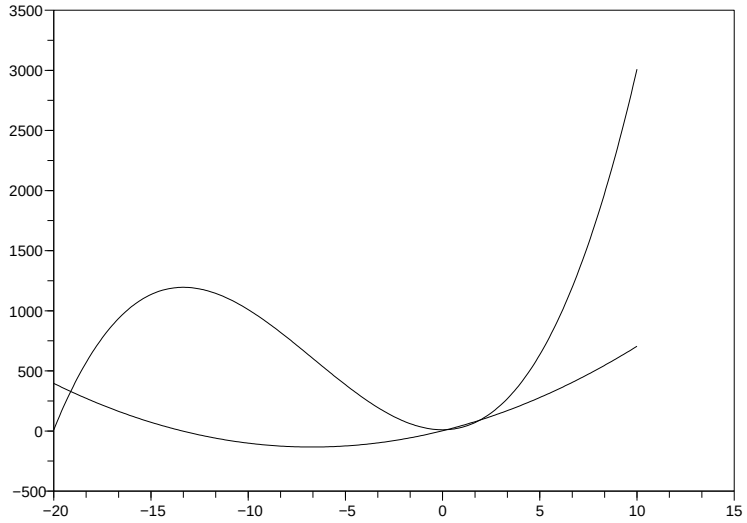
## 1.10 Tangente approssimata al grafico di una funzione (3a)




```
//:grtan3a.sce
//
// Tangente approssimata al grafico di una funzione (3a)
//
function y=f(x), y=10+20*x^2+x^3, endfunction;
function yp=fp(x0), h=0.1, yp=((f(x0+h)-f(x0))/h), endfunction;
function y=ftan(x,x0), y=f(x0)+fp(x0)*(x-x0), endfunction;
clf();
x=[-20:0.1:10];
x0=-2;
plot2d(x,f(x));
plot2d(x,ftan(x,x0));
//.
```

[Download]

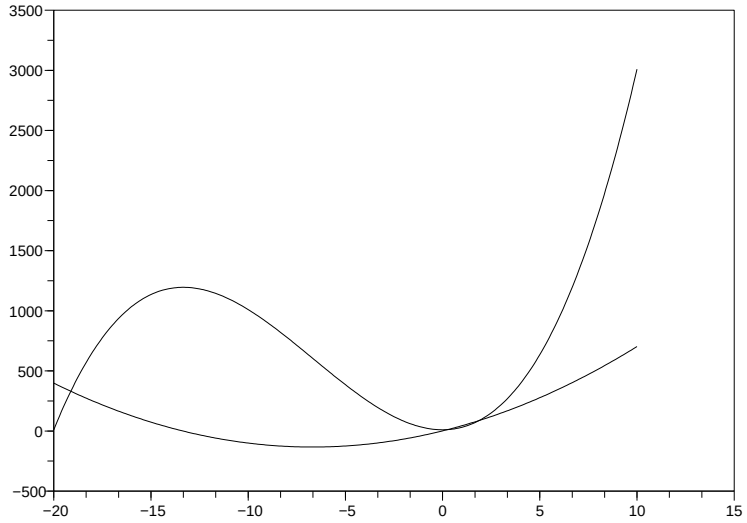
## 1.11 Grafico della derivata approssimata di una funzione (1)




```
//:fder1.sce
//
// Grafico della derivata approssimata di una funzione (1)
//
function y=f(x), y=10+20*x^2+x^3, endfunction;
function yp=fp(x), h=0.1, yp=((f(x+h)-f(x))/h), endfunction;
clf();
x=[-20:0.1:10];
plot2d(x,f(x));
plot2d(x,fp(x));
//.
```

[Download]

## 1.12 Grafico della derivata approssimata di una funzione (2)




```
//:fder2.sce
//
// Grafico della derivata approssimata di una funzione (2)
//
function y=f(x), y=10+20*x^2+x^3, endfunction;
function yp=fp(x), h=d/2, yp=((f(x+h)-f(x))/h), endfunction;
clf();
d=0.1;
x=[-20:d:10];
plot2d(x,f(x));
plot2d(x,fp(x));
//.
```

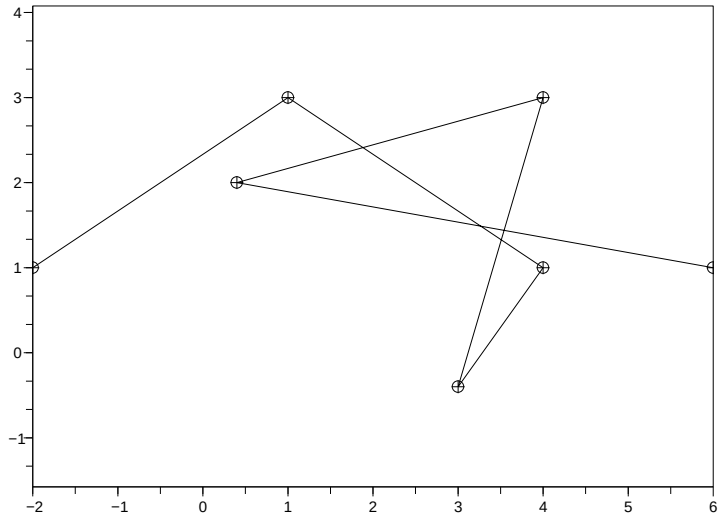
[Download]

# Capitolo 2

# Curve, secanti e tangenti

Il grafico di una funzione è solo un caso particolare di curva nel piano. Una curva in generale può avere un aspetto molto più vario. Si vede come disegnare una curva qualsiasi e come tracciare la retta tangente in un punto e anche la retta ortogonale a questa.

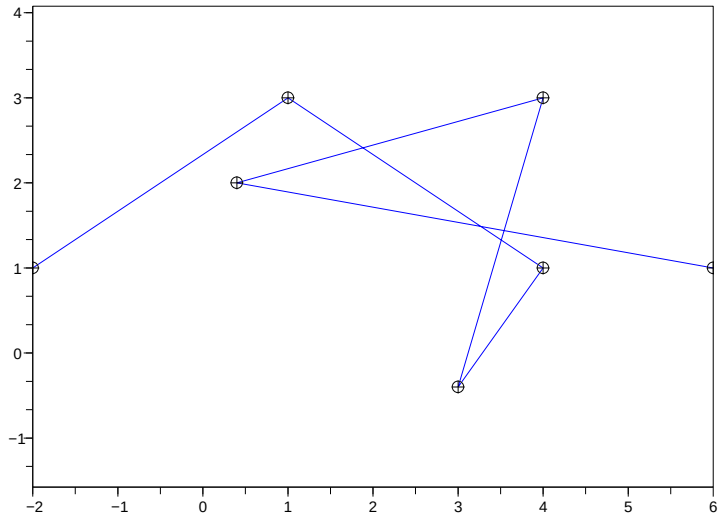
## 2.1 Poligonale




```
//:polgl.sce
//
// Poligonale
//
x=[-2  1  4  3   4  0.4  6]
y=[ 1  3  1 -0.4 3  2    1]

clf();
plot2d(0,0,1,"031");
plot2d(x,y,-3);
plot2d(x,y);
//.
```

[Download]

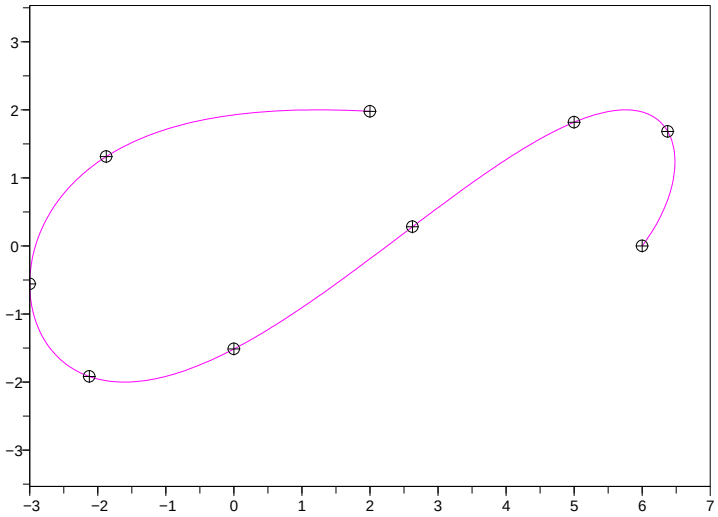
## 2.2 Poligonale (m)




```
//:polgl-m.sce
//
// Poligonale (m)
//
x=[-2  1  4  3   4  0.4  6]
y=[ 1  3  1 -0.4 3  2    1]

clf();
plot2d(0,0,1,"031");
plot2d(x,y,2);
for i=[1:7]
  xclick;
  plot2d(x(i),y(i),-3);
end
//.
```

[Download]

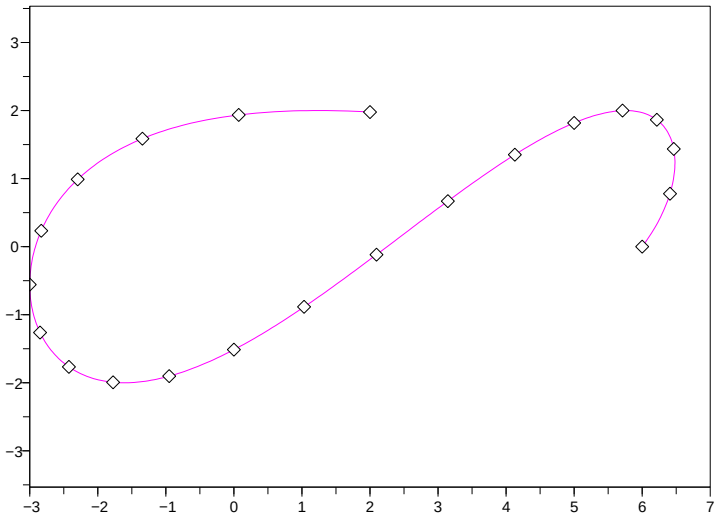
## 2.3 Traiettoria




```
//:tra1.sce
//
// Traiettoria
//
function x=fx(t), x=6+3*t-5*t^2+t^3, endfunction;
function y=fy(t), y=2*sin(2*t), endfunction;

t=[0:0.5:4]; // suddivisione rada dell'intervallo di tempo
clf();
plot2d(0,0,1,"031");
plot2d(fx(t),fy(t),-3);
plot2d(fx(t),fy(t),4);

t=[0:0.01:4]; // suddivisione fitta dell'intervallo di tempo
plot2d(fx(t),fy(t),6);

clf();
plot2d(0,0,1,"031");
plot2d(fx(t),fy(t),6);
// Tacche su una suddivisione rada dell'intervallo di tempo
for ti=[0:0.5:4]
      plot2d(fx(ti),fy(ti),-3);
end
//.
```

[Download]

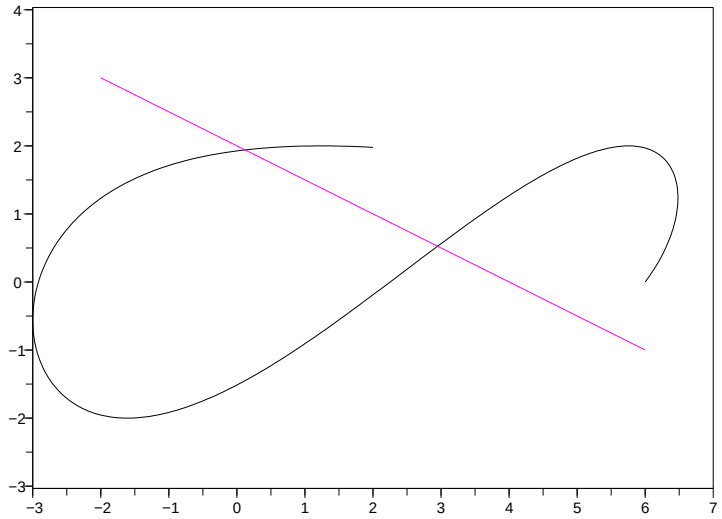
## 2.4 Traiettoria (m)




```
//:tra1-m.sce
//
// Traiettoria (m)
//
function x=fx(t), x=6+3*t-5*t^2+t^3, endfunction;
function y=fy(t), y=2*sin(2*t), endfunction;

t=[0:0.01:4];

clf();
plot2d(0,0,1,"031");
plot2d(fx(t),fy(t),6);
// Generazione delle tacche scandita con il mouse
for ti=[t(1):0.2:t($)]
      xclick();
      plot2d(fx(ti),fy(ti),-5);
end
//.
```

[Download]

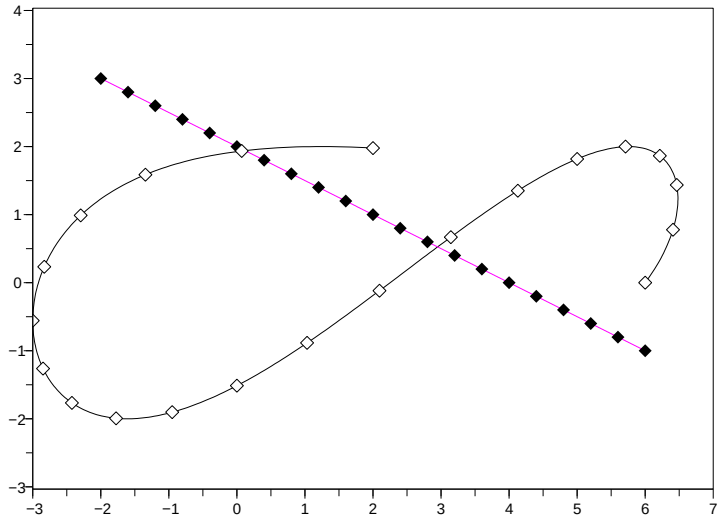
## 2.5 Traiettoria rettilinea




```
//:tra2.sce
//
// Traiettoria rettilinea
//
function x=fx(t), x=6+3*t-5*t^2+t^3, endfunction;
function y=fy(t), y=2*sin(2*t), endfunction;

function x=rx(t), x=6-2*t, endfunction;
function y=ry(t), y=-1+t, endfunction;

t=[0:0.01:4];

clf();
plot2d(0,0,1,"031");
plot2d(fx(t),fy(t));
plot2d(rx(t),ry(t),6);
//.
```

[Download]

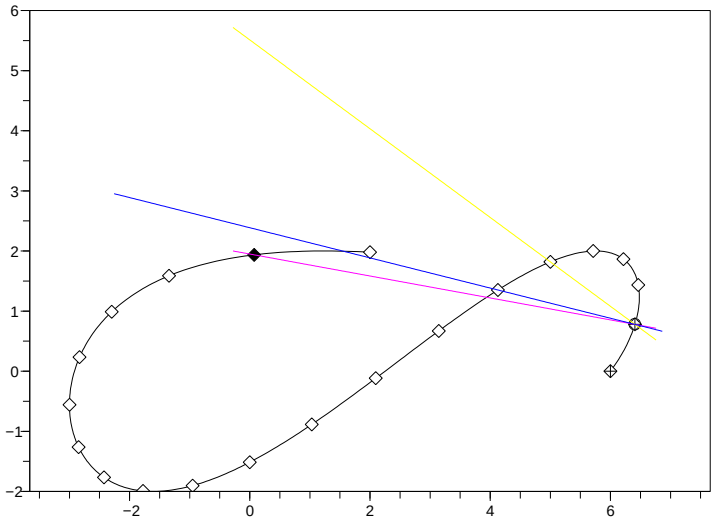
## 2.6 Traiettoria rettilinea (m)




```
//:tra2-m.sce
//
// Traiettoria rettilinea (m)
//
function x=fx(t), x=6+3*t-5*t^2+t^3, endfunction;
function y=fy(t), y=2*sin(2*t), endfunction;

function x=rx(t), x=6-2*t, endfunction;
function y=ry(t), y=-1+t, endfunction;

t=[0:0.01:4];

clf();
plot2d(0,0,1,"031");
plot2d(fx(t),fy(t));
plot2d(rx(t),ry(t),6);
// Tacche sincronizzate sulle due traiettorie
for ti=[t(1):0.2:t($)]
      xclick();
      plot2d(fx(ti),fy(ti),-5);
      plot2d(rx(ti),ry(ti),-4);
end
//.
```

[Download]

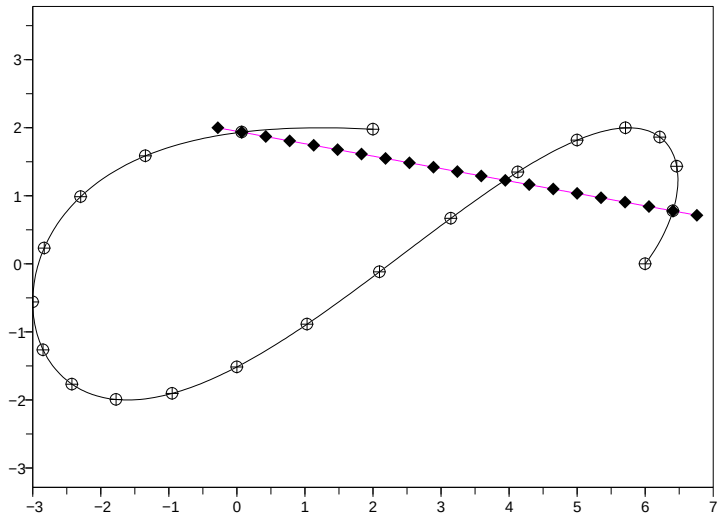
## 2.7 Traiettoria rettilinea secante




```
//:trasec1.sce
//
// Traiettoria rettilinea secante
//
function x=fx(t), x=6+3*t-5*t^2+t^3, endfunction;
function y=fy(t), y=2*sin(2*t), endfunction;

function x=fxsec(t,t0,t1), x=fx(t0)+((fx(t1)-fx(t0))/(t1-t0))*(t-t0), endfunction;
function y=fysec(t,t0,t1), y=fy(t0)+((fy(t1)-fy(t0))/(t1-t0))*(t-t0), endfunction;

t=[0:0.01:4];

clf();
plot2d(0,0,1,"031");
plot2d(fx(t),fy(t));
for ti=[t(1):0.2:t($)]
      plot2d(fx(ti),fy(ti),-5);
end
plot2d(fx(t(1)),fy(t(1)),-1);

t0=t(1)+1*0.2;
t1=t(1)+19*0.2;
plot2d(fx(t0),fy(t0),-3);
plot2d(fx(t1),fy(t1),-4);

// Fascio di traiettorie secanti
plot2d(fxsec(t,t0,t1),fysec(t,t0,t1),6);
plot2d(fxsec(t,t0,t0+4*0.2),fysec(t,t0,t0+4*0.2),7);
plot2d(fxsec(t,t0,t0+5*0.2),fysec(t,t0,t0+5*0.2),2);
//.
```

[Download]

## 2.8 Traiettoria rettilinea secante (m)

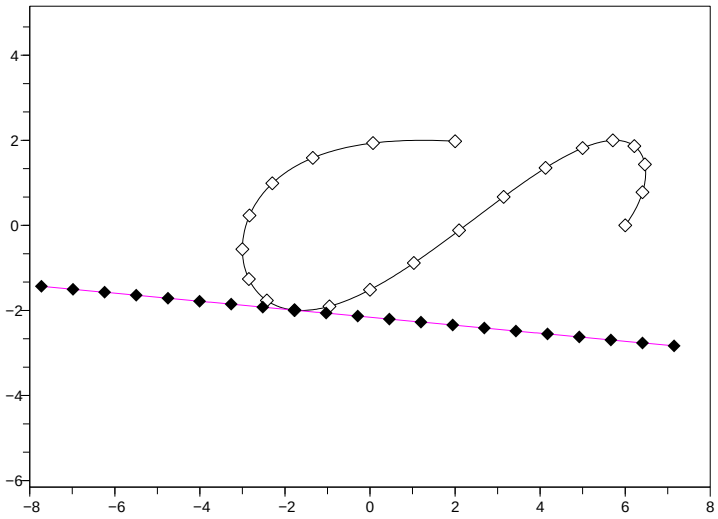



```
//:trasec1-m.sce
//
// Traiettoria rettilinea secante (m)
//
function x=fx(t), x=6+3*t-5*t^2+t^3, endfunction;
function y=fy(t), y=2*sin(2*t), endfunction;

function x=fxsec(t,t0,t1), x=fx(t0)+((fx(t1)-fx(t0))/(t1-t0))*(t-t0), endfunction;
function y=fysec(t,t0,t1), y=fy(t0)+((fy(t1)-fy(t0))/(t1-t0))*(t-t0), endfunction;

t=[0:0.01:4];

clf();
plot2d(0,0,1,"031");
plot2d(fx(t),fy(t));
// Tacche ad intervalli di tempo uguali
for ti=[t(1):0.2:t($)]
      plot2d(fx(ti),fy(ti),-3);
end
// Un segno nel punto iniziale
plot2d(fx(t(1)),fy(t(1)),-5);

// Tacche negli istanti t0 e t1
t0=t(1)+1*0.2;
t1=t(1)+19*0.2;
plot2d(fx(t0),fy(t0),-4);
plot2d(fx(t1),fy(t1),-4);
// Traiettoria secante negli istanti t0 e t1
plot2d(fxsec(t,t0,t1),fysec(t,t0,t1),6);

clf();
plot2d(0,0,1,"031");
plot2d(fx(t),fy(t));
plot2d(fxsec(t,t0,t1),fysec(t,t0,t1),6);
```

```
// Tacche sincronizzate
for ti=[t(1):0.2:t($)]
      xclick();
      plot2d(fx(ti),fy(ti),-3);
      plot2d(fxsec(ti,t0,t1),fysec(ti,t0,t1),-4);
end
//.
```

[Download]

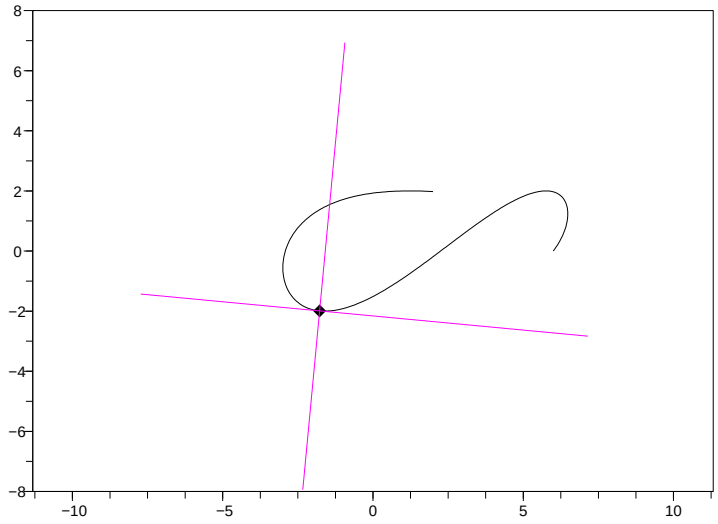
## 2.9 Traiettoria tangente approssimata (m)




```
//:tratan1-m.sce
//
// Traiettoria tangente approssimata (m)
//
function x=fx(t), x=6+3*t-5*t^2+t^3, endfunction;
function y=fy(t), y=2*sin(2*t), endfunction;

function vx=vfx(t0), dt=0.0001, vx=(fx(t0+dt)-fx(t0))/dt, endfunction
function vy=vfy(t0), dt=0.0001, vy=(fy(t0+dt)-fy(t0))/dt, endfunction

function x=fxtan(t,t0), x=fx(t0)+vfx(t0)*(t-t0), endfunction;
function y=fytan(t,t0), y=fy(t0)+vfy(t0)*(t-t0), endfunction;

t=[0:0.01:4];

clf();
plot2d(0,0,1,"031");
plot2d(fx(t),fy(t));

t0=t(1)+12*0.2;
plot2d(fx(t0),fy(t0),-4);
plot2d(fxtan(t,t0),fytan(t,t0),6);

// Tacche sincronizzate
for ti=[t(1):0.2:t($)]
      xclick();
      plot2d(fx(ti),fy(ti),-5);
      plot2d(fxtan(ti,t0),fytan(ti,t0),-4);
end
//.
```

[Download]

## 2.10 Traiettoria ortogonale approssimata

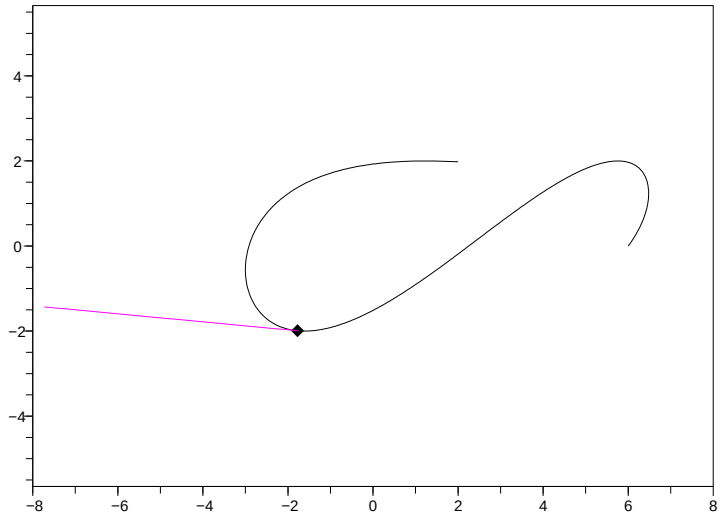



```
//:traort1-m.sce
//
// Traiettoria ortogonale approssimata
//
function x=fx(t), x=6+3*t-5*t^2+t^3, endfunction;
function y=fy(t), y=2*sin(2*t), endfunction;

function vx=vfx(t0), dt=0.0001, vx=(fx(t0+dt)-fx(t0))/dt, endfunction
function vy=vfy(t0), dt=0.0001, vy=(fy(t0+dt)-fy(t0))/dt, endfunction

function x=fxtan(t,t0), x=fx(t0)+vfx(t0)*(t-t0), endfunction;
function y=fytan(t,t0), y=fy(t0)+vfy(t0)*(t-t0), endfunction;

function x=fxort(t,t0), x=fx(t0)-vfy(t0)*(t-t0), endfunction;
function y=fyort(t,t0), y=fy(t0)+vfx(t0)*(t-t0), endfunction;

t=[0:0.01:4];

clf();
plot2d(0,0,1,"031");
plot2d(fx(t),fy(t));

t0=t(1)+12*0.2;
plot2d(fx(t0),fy(t0),-4);
plot2d(fxtan(t,t0),fytan(t,t0),6);
plot2d(fxort(t,t0),fyort(t,t0),6);

// Tacche lungo la traiettoria tangente
// e la traiettoria ortogonale in un punto
for ti=[t(1):0.4:t($)]
      xclick();
      plot2d(fx(ti),fy(ti),-5);
      plot2d(fxtan(ti,t0),fytan(ti,t0),-4);
      plot2d(fxort(ti,t0),fyort(ti,t0),-4);
end
```

```
//.
```

[Download]

## 2.11 Traiettoria tangente parziale

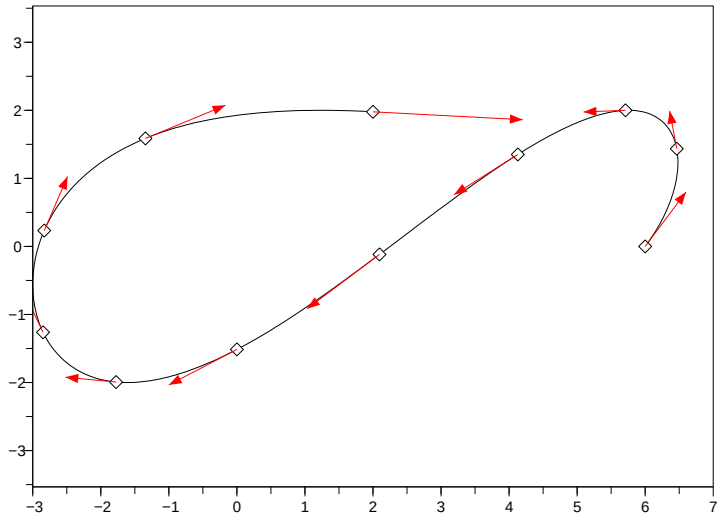



```
//:tratan2-m.sce
//
// Traiettoria tangente parziale
//
function x=fx(t), x=6+3*t-5*t^2+t^3, endfunction;
function y=fy(t), y=2*sin(2*t), endfunction;

function vx=vfx(t0), dt=0.0001, vx=(fx(t0+dt)-fx(t0))/dt, endfunction
function vy=vfy(t0), dt=0.0001, vy=(fy(t0+dt)-fy(t0))/dt, endfunction

function x=fxtan(t,t0), x=fx(t0)+vfx(t0)*(t-t0), endfunction;
function y=fytan(t,t0), y=fy(t0)+vfy(t0)*(t-t0), endfunction;

t=[0:0.01:4];

clf();
plot2d(0,0,1,"031");
plot2d(fx(t),fy(t));

t0=t(1)+12*0.2;
plot2d(fx(t0),fy(t0),-4);

// Traiettoria tangente in t0, per t da t0 in poi
tp=[t0:0.01:t($)];
plot2d(fxtan(tp,t0),fytan(tp,t0),6);

clf();
plot2d(0,0,1,"031");
plot2d(fx(t),fy(t));
// Traiettoria tangente in t0, per t da t0 a t0+0.2
tp=[t0,t0+0.2];
plot2d(fx(t0),fy(t0),-4);
plot2d(fxtan(tp,t0),fytan(tp,t0),6);

clf();
```

```
plot2d(0,0,1,"031");
plot2d(fx(t),fy(t));
// Traiettorie tangenti in punti diversi
for ti=[t(1):0.4:t($)]
      xclick();
      plot2d(fx(ti),fy(ti),-5);
      tp=[ti,ti+0.2];
      plot2d(fxtan(tp,ti),fytan(tp,ti),5);
end
//.
```

[Download]

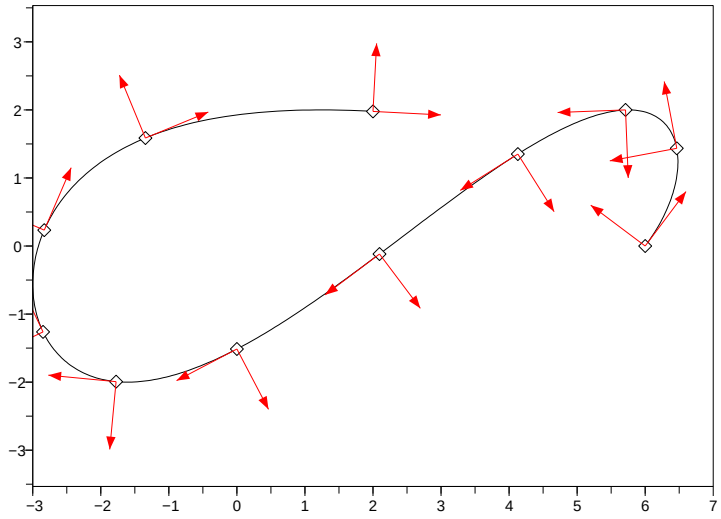
## 2.12 Vettori velocita' lungo una traiettoria




```
//:travel.sce
//
// Vettori velocita' lungo una traiettoria
//
function x=fx(t), x=6+3*t-5*t^2+t^3, endfunction;
function y=fy(t), y=2*sin(2*t), endfunction;

function vx=vfx(t0), dt=0.0001, vx=(fx(t0+dt)-fx(t0))/dt, endfunction
function vy=vfy(t0), dt=0.0001, vy=(fy(t0+dt)-fy(t0))/dt, endfunction

function x=fxtan(t,t0), x=fx(t0)+vfx(t0)*(t-t0), endfunction;
function y=fytan(t,t0), y=fy(t0)+vfy(t0)*(t-t0), endfunction;

t=[0:0.01:4];

clf();
plot2d(0,0,1,"031");
plot2d(fx(t),fy(t));
for ti=[t(1):0.4:t($)]
      plot2d(fx(ti),fy(ti),-5);
      tp=[ti,ti+0.2];
      xarrows(fxtan(tp,ti),fytan(tp,ti),4,5);
end
// (per ogni freccia sono specificati la grandezza e il colore)
//.
```

[Download]

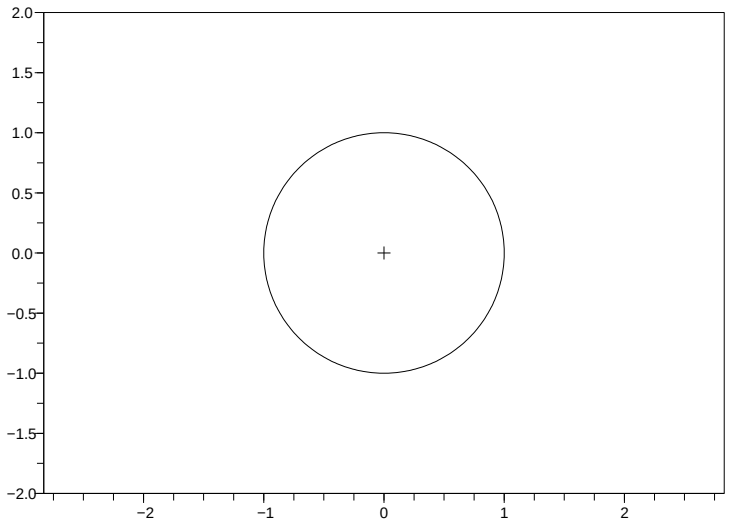
## 2.13 Versore tangente e versore normale




```
//:travers.sce
//
// Versore tangente e versore normale
//
function x=fx(t), x=6+3*t-5*t^2+t^3, endfunction;
function y=fy(t), y=2*sin(2*t), endfunction;

function vx=vfx(t0), dt=0.0001, vx=(fx(t0+dt)-fx(t0))/dt, endfunction;
function vy=vfy(t0), dt=0.0001, vy=(fy(t0+dt)-fy(t0))/dt, endfunction;

function x=fxtan(t,t0), x=fx(t0)+vfx(t0)*(t-t0), endfunction;
function y=fytan(t,t0), y=fy(t0)+vfy(t0)*(t-t0), endfunction;

function x=fxort(t,t0), x=fx(t0)-vfy(t0)*(t-t0), endfunction;
function y=fyort(t,t0), y=fy(t0)+vfx(t0)*(t-t0), endfunction;

// versore tangente
function vx=vtx(t0), vx=vfx(t0)/sqrt(vfx(t0)^2+vfy(t0)^2), endfunction;
function vy=vty(t0), vy=vfy(t0)/sqrt(vfx(t0)^2+vfy(t0)^2), endfunction;

// versore normale
function vx=vnx(t0), vx=-vty(t0), endfunction;
function vy=vny(t0), vy=vtx(t0), endfunction;

clf();
t=[0:0.01:4];
plot2d(0,0,1,"031");
plot2d(fx(t),fy(t));
d=[0,1];
for ti=[t(1):0.4:t($)]
      plot2d(fx(ti),fy(ti),-5);
      xarrows(fx(ti)+vtx(ti)*d,fy(ti)+vty(ti)*d,4,5);
      xarrows(fx(ti)+vnx(ti)*d,fy(ti)+vny(ti)*d,4,5);
end
//.
```

[Download]

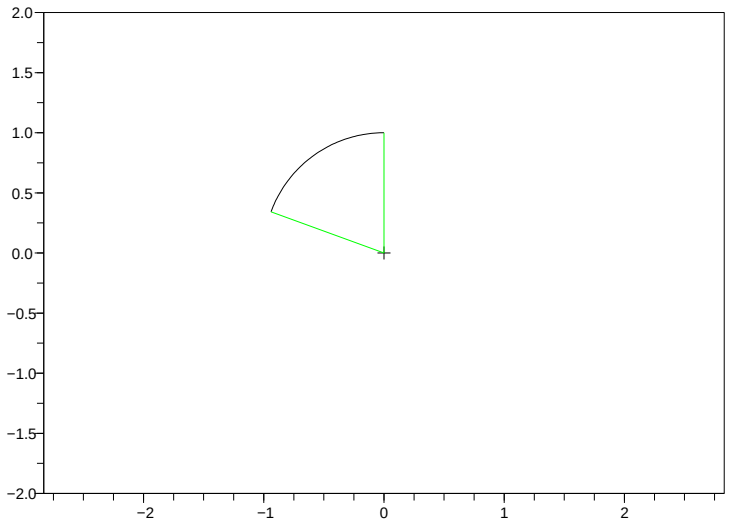
## 2.14 Cerchio




```
//:crc.sce
//
// Cerchio
//
function x=fx(a), x=cos(a), endfunction;
function y=fy(a), y=sin(a), endfunction;
clf();
a=[0:360]/360*2*%pi;
plot2d(0,0,-1,"031"," ",[-2,-2,2,2]);
plot2d(fx(a),fy(a));
//.
```

[Download]

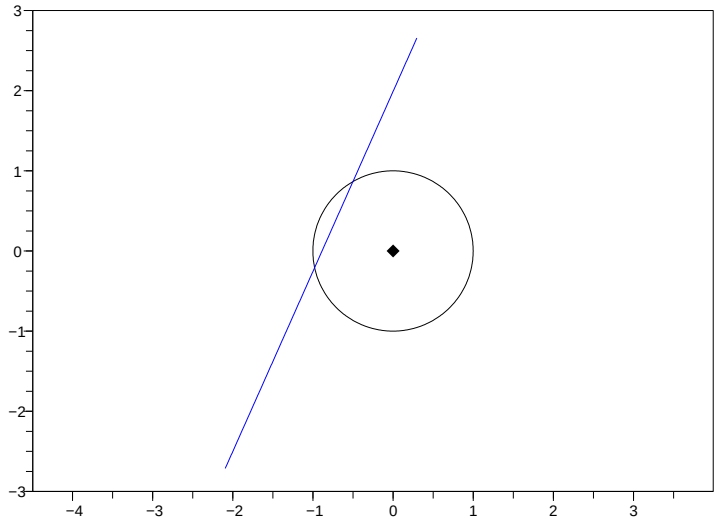
## 2.15 Arco di cerchio




```
//:crarc.sce
//
// Arco di cerchio
//
function x=fx(a), x=cos(a), endfunction;
function y=fy(a), y=sin(a), endfunction;
clf();
a=[90:160]/360*2*%pi;
// arco
plot2d(0,0,-1,"031",rect=[-2,-2,2,2]);
plot2d(fx(a),fy(a));
// raggi
plot2d([0,fx(a(1))],[0,fy(a(1))],3);
plot2d([0,fx(a($))],[0,fy(a($))],3);
//.
```

[Download]

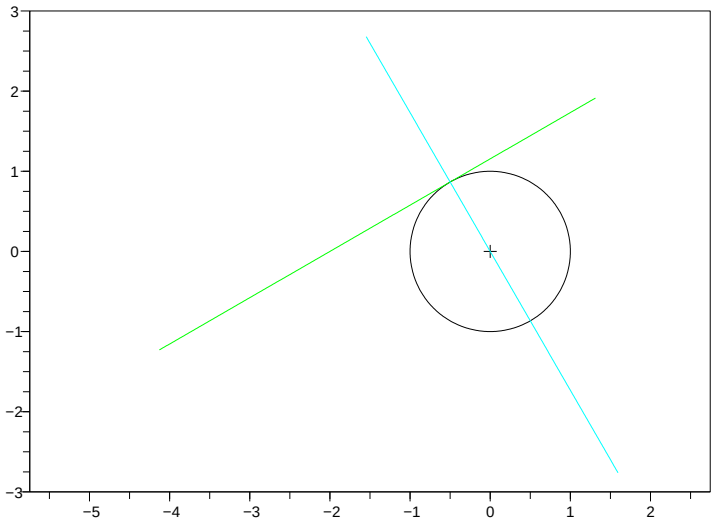
## 2.16 Rette secanti un cerchio




```
//:crsec1.sce
//
// Rette secanti un cerchio
//
function x=fx(a), x=cos(a), endfunction;
function y=fy(a), y=sin(a), endfunction;

function x=fxsec(a,a0,a1), x=fx(a0)+((fx(a1)-fx(a0))/(a1-a0))*(a-a0), endfunction;
function y=fysec(a,a0,a1), y=fy(a0)+((fy(a1)-fy(a0))/(a1-a0))*(a-a0), endfunction;

clf();
a=[0:360]/360*2*%pi;
plot2d(0,0,-4,"031"," ",[-2,-2,2,2]);
plot2d(fx(a),fy(a));
a0=2*%pi/3;
a1=a0+4*%pi/10;
plot2d(fxsec(a,a0,a1),fysec(a,a0,a1),2);
//.
```

[Download]

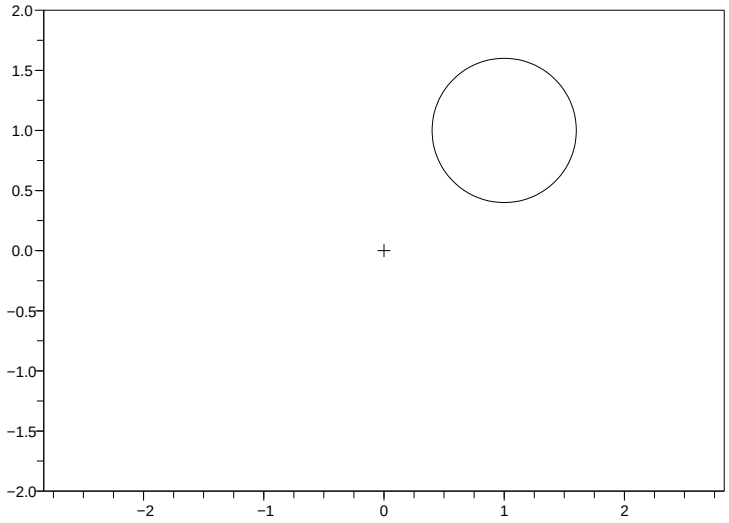
## 2.17 Rette tangenti e rette ortogonali ad un cerchio




```
//:crtan1.sce
//
// Rette tangenti e rette ortogonali ad un cerchio
//
function x=fx(a), x=cos(a), endfunction;
function y=fy(a), y=sin(a), endfunction;

function vx=vfx(a0), da=0.0001, vx=(fx(a0+da)-fx(a0))/da, endfunction;
function vy=vfy(a0), da=0.0001, vy=(fy(a0+da)-fy(a0))/da, endfunction;

function x=fxtan(a,a0), x=fx(a0)+vfx(a0)*(a-a0), endfunction;
function y=fytan(a,a0), y=fy(a0)+vfy(a0)*(a-a0), endfunction;

function x=fxort(a,a0), x=fx(a0)-vfy(a0)*(a-a0), endfunction;
function y=fyort(a,a0), y=fy(a0)+vfx(a0)*(a-a0), endfunction;

clf();
a=[0:360]/360*2*%pi;
plot2d(0,0,-1,"031"," ",[-2,-2,2,2]);
plot2d(fx(a),fy(a));
a0=2*%pi/3;
plot2d(fxtan(a,a0),fytan(a,a0),3);
plot2d(fxort(a,a0),fyort(a,a0),4);
//.
```

[Download]

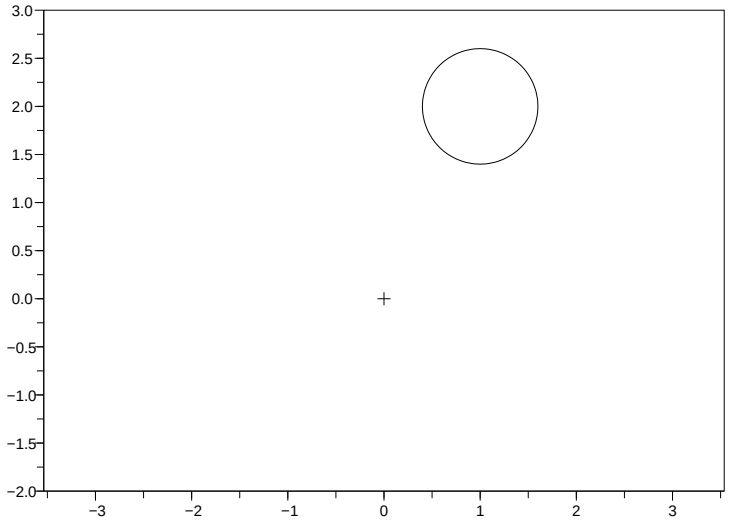
## 2.18 Cerchio con centro in (x0,y0) e di raggio r (1)




```
//:crp1.sce
//
// Cerchio con centro in (x0,y0) e di raggio r (1)
//
function x=fx(a), x=r*cos(a), endfunction;
function y=fy(a), y=r*sin(a), endfunction;
clf();
a=[0:360]/360*2*%pi;
plot2d(0,0,-1,"031"," ",[-2,-2,2,2]);
r=0.6;
x0=1;
y0=1;
plot2d(x0+fx(a),y0+fy(a));
//.
```

[Download]

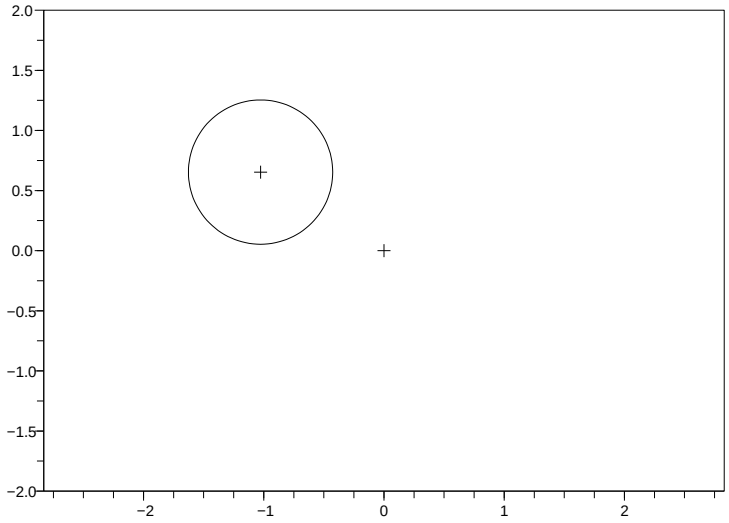
## 2.19 Cerchio con centro in (x0,y0) e di raggio r (2)




```
//:crp2.sce
//
// Cerchio con centro in (x0,y0) e di raggio r (2)
//
function x=fxc(x0,r,a), x=x0+r*cos(a), endfunction;
function y=fyc(y0,r,a), y=y0+r*sin(a), endfunction;
clf();
a=[0:360]/360*2*%pi;
plot2d(0,0,-1,"031"," ",[-2,-2,2,2]);
r=0.6;
x0=1;
y0=2;
plot2d(fxc(x0,r,a),fyc(y0,r,a));
//.
```

[Download]

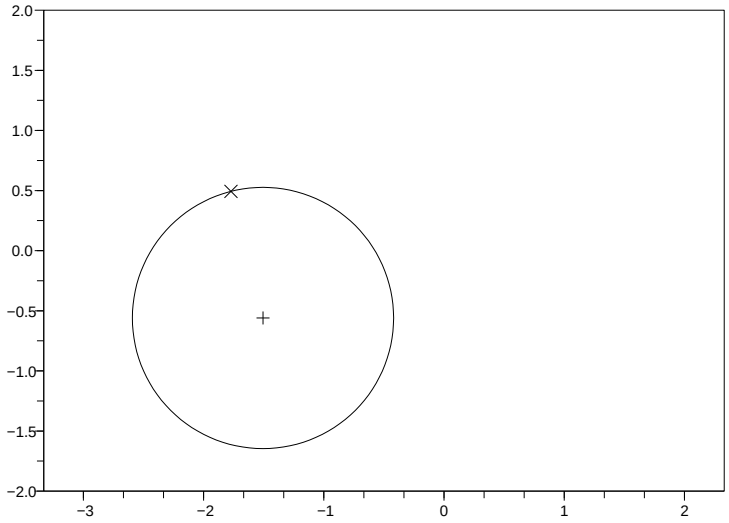
## 2.20 Cerchio con centro in (x0,y0) e di raggio r (m)




```
//:crp2-m.sce
//
// Cerchio con centro in (x0,y0) e di raggio r (m)
//
function x=fxc(x0,r,a), x=x0+r*cos(a), endfunction;
function y=fyc(y0,r,a), y=y0+r*sin(a), endfunction;
clf();
a=[0:360]/360*2*%pi;
plot2d(0,0,-1,"031"," ",[-2,-2,2,2]);
r=0.6;
[b,x0,y0]=xclick();
plot2d(x0,y0,-1);
plot2d(fxc(x0,r,a),fyc(y0,r,a));
//.
```

[Download]

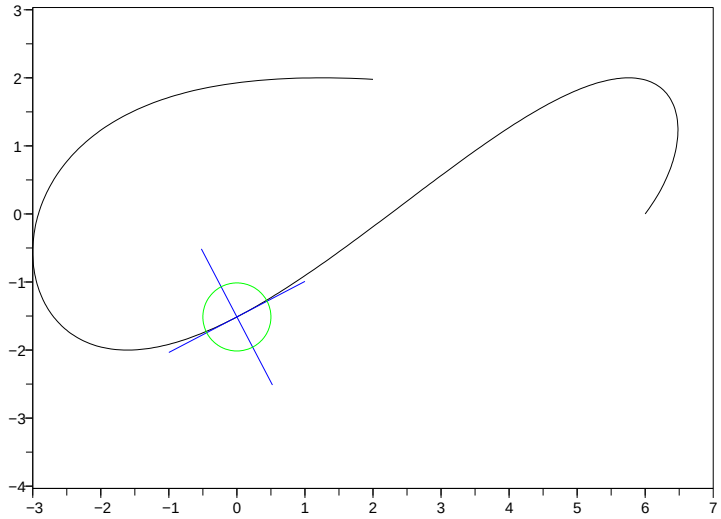
## 2.21 Cerchio con centro in (x0,y0) e passante per (x1,y1) (m)




```
//:crp3-m.sce
//
// Cerchio con centro in (x0,y0) e passante per (x1,y1) (m)
//
function x=fxc(x0,r,a), x=x0+r*cos(a), endfunction;
function y=fyc(y0,r,a), y=y0+r*sin(a), endfunction;
clf();
a=[0:360]/360*2*%pi;
plot2d(0,0,1,"031"," ",[-2,-2,2,2]);

[b,x0,y0]=xclick();
plot2d(x0,y0,-1);
[b,x1,y1]=xclick();
plot2d(x1,y1,-2);
r=sqrt((x1-x0)^2+(y1-y0)^2);
plot2d(fxc(x0,r,a),fyc(y0,r,a));
//.
```

[Download]

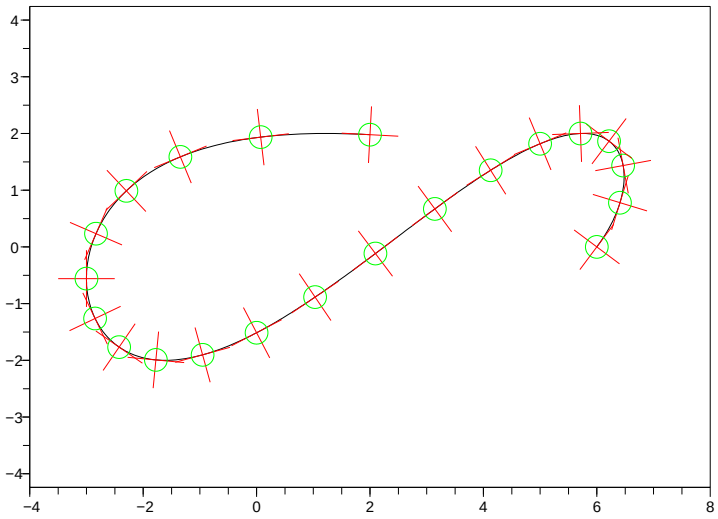
## 2.22 Cerchio con centro su una curva




```
//:crtra1.sce
//
// Cerchio con centro su una curva
//
function x=fx(t), x=6+3*t-5*t^2+t^3, endfunction;
function y=fy(t), y=2*sin(2*t), endfunction;

function vx=vfx(t0), dt=0.0001, vx=(fx(t0+dt)-fx(t0))/dt, endfunction;
function vy=vfy(t0), dt=0.0001, vy=(fy(t0+dt)-fy(t0))/dt, endfunction;

function x=fxtan(t,t0), x=fx(t0)+vfx(t0)*(t-t0), endfunction;
function y=fytan(t,t0), y=fy(t0)+vfy(t0)*(t-t0), endfunction;

function x=fxort(t,t0), x=fx(t0)-vfy(t0)*(t-t0), endfunction;
function y=fyort(t,t0), y=fy(t0)+vfx(t0)*(t-t0), endfunction;

function xc=fxc(x0,r,a), xc=x0+r*cos(a), endfunction;
function yc=fyc(y0,r,a), yc=y0+r*sin(a), endfunction;

clf();
t=[0:0.01:4];
a=[0:360]/360*2*%pi;
plot2d(0,0,1,"031"," ");
plot2d(fx(t),fy(t));
t0=2;
tp=[t0-0.2,t0+0.2];
plot2d(fxtan(tp,t0),fytan(tp,t0),2);
plot2d(fxort(tp,t0),fyort(tp,t0),2);

plot2d(fxc(fx(t0),0.5,a),fyc(fy(t0),0.5,a),3);
//.
```

[Download]

## 2.23 Cerchi con centro su una curva (m)

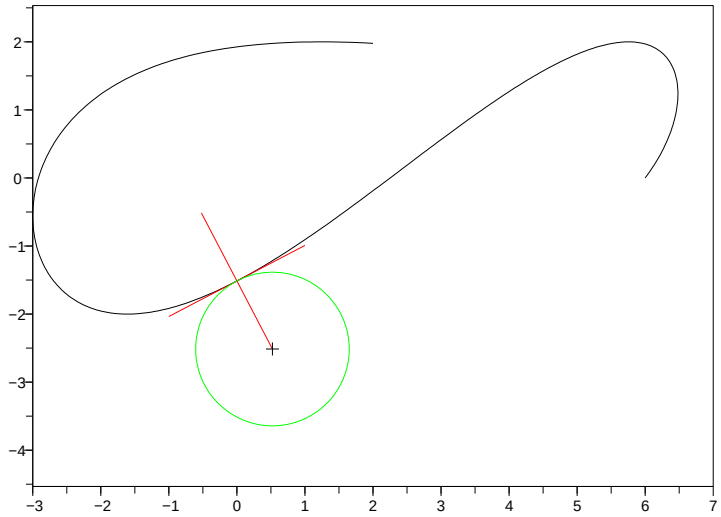



```
//:crtra1-m.sce
//
// Cerchi con centro su una curva (m)
//
function x=fx(t), x=6+3*t-5*t^2+t^3, endfunction;
function y=fy(t), y=2*sin(2*t), endfunction;

function vx=vfx(t0), dt=0.0001, vx=(fx(t0+dt)-fx(t0))/dt, endfunction;
function vy=vfy(t0), dt=0.0001, vy=(fy(t0+dt)-fy(t0))/dt, endfunction;

function x=fxtan(t,t0), x=fx(t0)+vfx(t0)*(t-t0), endfunction;
function y=fytan(t,t0), y=fy(t0)+vfy(t0)*(t-t0), endfunction;

function x=fxort(t,t0), x=fx(t0)-vfy(t0)*(t-t0), endfunction;
function y=fyort(t,t0), y=fy(t0)+vfx(t0)*(t-t0), endfunction;

// versore tangente
function vx=vtx(t0), vx=vfx(t0)/sqrt(vfx(t0)^2+vfy(t0)^2), endfunction;
function vy=vty(t0), vy=vfy(t0)/sqrt(vfx(t0)^2+vfy(t0)^2), endfunction;

// versore normale
function vx=vnx(t0), vx=-vty(t0), endfunction;
function vy=vny(t0), vy=vtx(t0), endfunction;

// cerchio
function xc=fxc(x0,r,a), xc=x0+r*cos(a), endfunction;
function yc=fyc(y0,r,a), yc=y0+r*sin(a), endfunction;

clf();
t=[0:0.01:4];
a=[0:360]/360*2*%pi;
plot2d(0,0,1,"031"," ");
plot2d(fx(t),fy(t));
for ti=[t(1):0.2:t($)]
      xclick;
```

```
      tp=[ti-0.2,ti+0.2];
      d=[-0.5,0.5];
      // segmenti ortogonali di lunghezza data
      plot2d(fx(ti)+vtx(ti)*d,fy(ti)+vty(ti)*d,5);
      plot2d(fx(ti)+vnx(ti)*d,fy(ti)+vny(ti)*d,5);
      // cerchio
      plot2d(fxc(fx(ti),0.2,a),fyc(fy(ti),0.2,a),3);
end
//.
```

[Download]

## 2.24 Cerchio tangente ad una curva

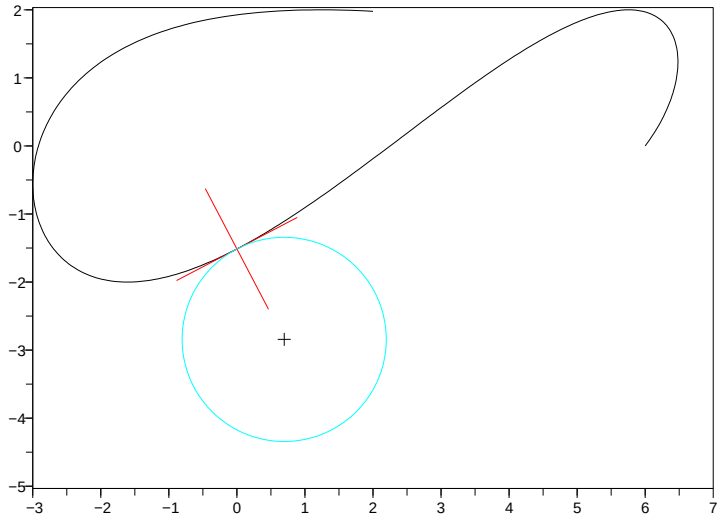



```
//:crtra2.sce
//
// Cerchio tangente ad una curva
//
function x=fx(t), x=6+3*t-5*t^2+t^3, endfunction;
function y=fy(t), y=2*sin(2*t), endfunction;

function vx=vfx(t0), dt=0.0001, vx=(fx(t0+dt)-fx(t0))/dt, endfunction;
function vy=vfy(t0), dt=0.0001, vy=(fy(t0+dt)-fy(t0))/dt, endfunction;

function x=fxtan(t,t0), x=fx(t0)+vfx(t0)*(t-t0), endfunction;
function y=fytan(t,t0), y=fy(t0)+vfy(t0)*(t-t0), endfunction;

function x=fxort(t,t0), x=fx(t0)-vfy(t0)*(t-t0), endfunction;
function y=fyort(t,t0), y=fy(t0)+vfx(t0)*(t-t0), endfunction;

// cerchio
function xc=fxc(x0,r,a), xc=x0+r*cos(a), endfunction;
function yc=fyc(y0,r,a), yc=y0+r*sin(a), endfunction;

clf();
t=[0:0.01:4];
a=[0:360]/360*2*%pi;
plot2d(0,0,1,"031"," ");
plot2d(fx(t),fy(t));

t0=2;
tp=[t0-0.2,t0+0.2];
plot2d(fxtan(tp,t0),fytan(tp,t0),5);
plot2d(fxort(tp,t0),fyort(tp,t0),5);

// dato t1, si calcola il raggio
t1=t0+0.2;
plot2d(fxort(t1,t0),fyort(t1,t0),-1);
x1=fxort(t1,t0);
```

```
y1=fyort(t1,t0);
x0=fx(t0);
y0=fy(t0);
r=sqrt((x1-x0)^2+(y1-y0)^2);
plot2d(fxc(x1,r,a),fyc(y1,r,a),3);
//.
```

[Download]

## 2.25 Cerchio di raggio r tangente ad una curva

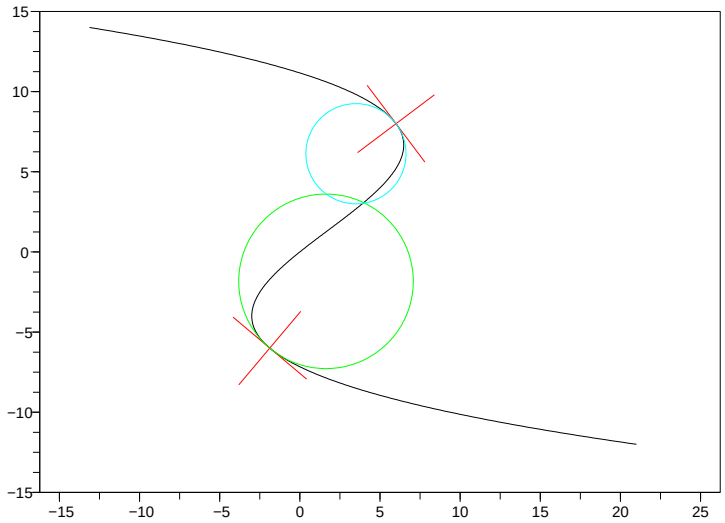



```
//:crtra3.sce
//
// Cerchio di raggio r tangente ad una curva
//
function x=fx(t), x=6+3*t-5*t^2+t^3, endfunction;
function y=fy(t), y=2*sin(2*t), endfunction;

function vx=vfx(t0), dt=0.0001, vx=(fx(t0+dt)-fx(t0))/dt, endfunction;
function vy=vfy(t0), dt=0.0001, vy=(fy(t0+dt)-fy(t0))/dt, endfunction;

function x=fxtan(t,t0), x=fx(t0)+vfx(t0)*(t-t0), endfunction;
function y=fytan(t,t0), y=fy(t0)+vfy(t0)*(t-t0), endfunction;

function x=fxort(t,t0), x=fx(t0)-vfy(t0)*(t-t0), endfunction;
function y=fyort(t,t0), y=fy(t0)+vfx(t0)*(t-t0), endfunction;

// versore tangente
function vx=vtx(t0), vx=vfx(t0)/sqrt(vfx(t0)^2+vfy(t0)^2), endfunction;
function vy=vty(t0), vy=vfy(t0)/sqrt(vfx(t0)^2+vfy(t0)^2), endfunction;

// versore normale
function vx=vnx(t0), vx=-vty(t0), endfunction;
function vy=vny(t0), vy=vtx(t0), endfunction;

// cerchio
function xc=fxc(x0,r,a), xc=x0+r*cos(a), endfunction;
function yc=fyc(y0,r,a), yc=y0+r*sin(a), endfunction;

clf();
t=[0:0.01:4];
a=[0:360]/360*2*%pi;
plot2d(0,0,1,"031"," ");
plot2d(fx(t),fy(t));

t0=2;
```

```
// segmenti ortogonali di lunghezza data
d=[-1,1];
plot2d(fx(t0)+vtx(t0)*d,fy(t0)+vty(t0)*d,5);
plot2d(fx(t0)+vnx(t0)*d,fy(t0)+vny(t0)*d,5);

// dato il raggio, si calcolano le coordinate del centro
r=1.5;
x1=fx(t0)+r*vnx(t0);
y1=fy(t0)+r*vny(t0);
plot2d(x1,y1,-1)
plot2d(fxc(x1,r,a),fyc(y1,r,a),4);
//.
```

[Download]

## 2.26 Cerchio osculatore ad una curva




```
//:crvosc.sce
//
// Cerchio osculatore ad una curva
//
function x=fx(t), x=-5*t+t^2+t^3, endfunction;
function y=fy(t), y=-4*t, endfunction;

function vx=vfx(t0), dt=0.0001, vx=(fx(t0+dt)-fx(t0))/dt, endfunction;
function vy=vfy(t0), dt=0.0001, vy=(fy(t0+dt)-fy(t0))/dt, endfunction;

function x=fxtan(t,t0), x=fx(t0)+vfx(t0)*(t-t0), endfunction;
function y=fytan(t,t0), y=fy(t0)+vfy(t0)*(t-t0), endfunction;

function x=fxort(t,t0), x=fx(t0)-vfy(t0)*(t-t0), endfunction;
function y=fyort(t,t0), y=fy(t0)+vfx(t0)*(t-t0), endfunction;

// versore tangente
function vx=vtx(t0), vx=vfx(t0)/sqrt(vfx(t0)^2+vfy(t0)^2), endfunction;
function vy=vty(t0), vy=vfy(t0)/sqrt(vfx(t0)^2+vfy(t0)^2), endfunction;

// versore normale
function vx=vnx(t0), vx=-vty(t0), endfunction;
function vy=vny(t0), vy=vtx(t0), endfunction;

// cerchio
function xc=fxc(x0,r,a), xc=x0+r*cos(a), endfunction;
function yc=fyc(y0,r,a), yc=y0+r*sin(a), endfunction;

// derivata del versore tangente rispetto all'ascissa curvilinea
function vx=vtxp(t0), dt=0.0001, vx=((vtx(t0+dt)-vtx(t0))/dt)/sqrt(vfx(t0)^2+vfy(t0)^2), endfuncti
function vy=vtyp(t0), dt=0.0001, vy=((vty(t0+dt)-vty(t0))/dt)/sqrt(vfx(t0)^2+vfy(t0)^2), endfuncti

t=[-3.5:0.01:3];
a=[0:360]/360*2*%pi;
clf();
```

```
plot2d(0,0,1,"031"," ");
plot2d(fx(t),fy(t));

t0=1.5;
d=[-3,3];
plot2d(fx(t0)+vtx(t0)*d,fy(t0)+vty(t0)*d,5);
plot2d(fx(t0)+vnx(t0)*d,fy(t0)+vny(t0)*d,5);
rc=1/(vtxp(t0)*vnx(t0)+vtyp(t0)*vny(t0)); // raggio di curvatura
x1=fx(t0)+rc*vnx(t0);
y1=fy(t0)+rc*vny(t0);
plot2d(fxc(x1,rc,a),fyc(y1,rc,a),3);

t0=-2;
d=[-3,3];
plot2d(fx(t0)+vtx(t0)*d,fy(t0)+vty(t0)*d,5);
plot2d(fx(t0)+vnx(t0)*d,fy(t0)+vny(t0)*d,5);
rc=1/(vtxp(t0)*vnx(t0)+vtyp(t0)*vny(t0)); // raggio di curvatura
x1=fx(t0)+rc*vnx(t0);
y1=fy(t0)+rc*vny(t0);
plot2d(fxc(x1,rc,a),fyc(y1,rc,a),4);
//.
```

[Download]

## 2.27 Funzioni (Curve, secanti e tangenti)

```
//:lib_curve.sce
//
// Funzioni (Curve, secanti e tangenti)
//
// ------<funzioni>-<inizio>----------------------------------------

// curva (traiettoria)
function x=fx(t), x=6+3*t-5*t^2+t^3, endfunction;
function y=fy(t), y=2*sin(2*t), endfunction;

// retta (traiettoria secante)
function x=fxsec(t,t0,t1), x=fx(t0)+((fx(t1)-fx(t0))/(t1-t0))*(t-t0), endfunction;
function y=fysec(t,t0,t1), y=fy(t0)+((fy(t1)-fy(t0))/(t1-t0))*(t-t0), endfunction;

// vettore tangente (velocita')
function vx=vfx(t0), dt=0.0001, vx=(fx(t0+dt)-fx(t0))/dt, endfunction;
function vy=vfy(t0), dt=0.0001, vy=(fy(t0+dt)-fy(t0))/dt, endfunction;

// retta tangente
function x=fxtan(t,t0), x=fx(t0)+vfx(t0)*(t-t0), endfunction;
function y=fytan(t,t0), y=fy(t0)+vfy(t0)*(t-t0), endfunction;

// retta ortogonale
function x=fxort(t,t0), x=fx(t0)-vfy(t0)*(t-t0), endfunction;
function y=fyort(t,t0), y=fy(t0)+vfx(t0)*(t-t0), endfunction;

// versore tangente
function vx=vtx(t0), vx=vfx(t0)/sqrt(vfx(t0)^2+vfy(t0)^2), endfunction;
function vy=vty(t0), vy=vfy(t0)/sqrt(vfx(t0)^2+vfy(t0)^2), endfunction;

// versore normale
function vx=vnx(t0), vx=-vty(t0), endfunction;
function vy=vny(t0), vy=vtx(t0), endfunction;

// cerchio
function xc=fxc(x0,r,a), xc=x0+r*cos(a), endfunction;
function yc=fyc(y0,r,a), yc=y0+r*sin(a), endfunction;

// derivata del versore tangente rispetto all'ascissa curvilinea
function vx=vtxp(t0),
      dt=0.0001,
      vx=((vtx(t0+dt)-vtx(t0))/dt)/sqrt(vfx(t0)^2+vfy(t0)^2),
endfunction;
function vy=vtyp(t0),
      dt=0.0001,
      vy=((vty(t0+dt)-vty(t0))/dt)/sqrt(vfx(t0)^2+vfy(t0)^2),
endfunction;

// ------<funzioni>-<fine>-----------------------------------------
//.
```

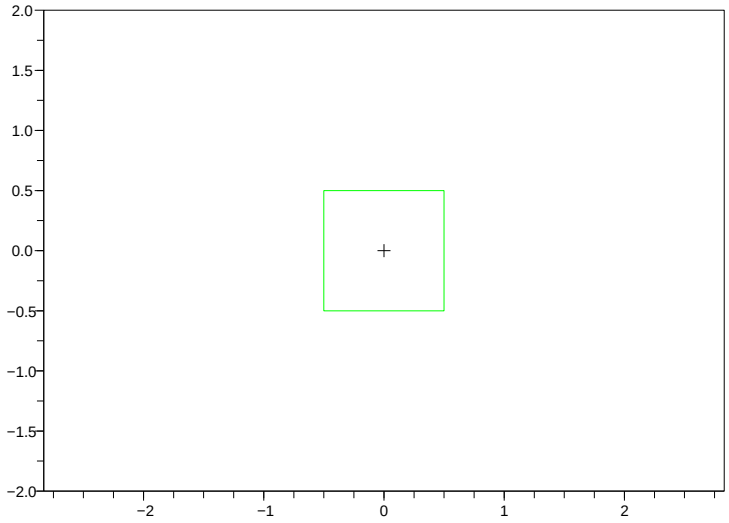
[Download]

# Capitolo 3

# Parallelogrammi, segmenti e vettori

Partendo dalla costruzione di un parallegramma si distinguono i segmenti dai vettori e le componenti dei vettori dalle coordinate dei vertici. Si utilizzano questi concetti costruendo il rettangolo di area uguale a quella di un parallelogramma e nel calcolo della distanza di un punto da una retta, attraverso le proiezioni di un vettore su una base ortonormale. Il calcolo dell'area viene poi reinterpretato come calcolo di un determinante.

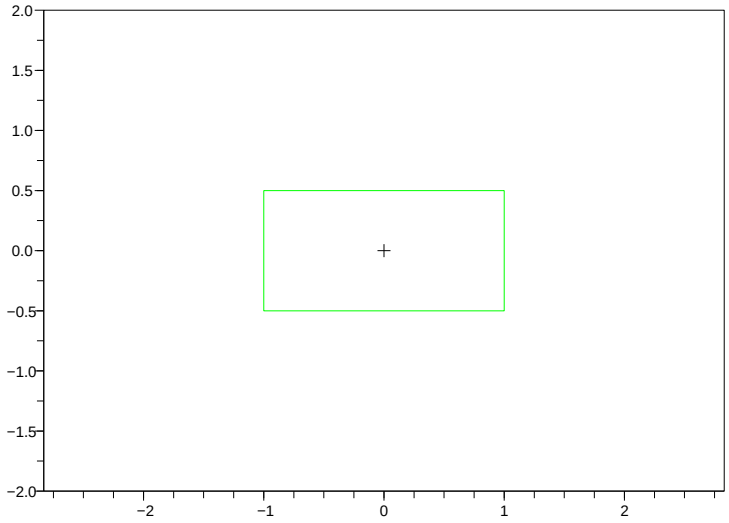
## 3.1 Quadrato con centro in (x0,y0) e di lato h




```
//:quad1.sce
//
// Quadrato con centro in (x0,y0) e di lato h
//
function x=fxq(x0,h), x=x0+[-1, 1, 1,-1,-1]*h/2, endfunction;
function y=fyq(y0,h), y=y0+[-1,-1, 1, 1,-1]*h/2, endfunction;
clf();
plot2d(0,0,1,"031",rect=[-2,-2,2,2]);
x0=0;
y0=0;
plot2d(x0,y0,-1)
plot2d(fxq(x0,1),fyq(x0,1),3);
//.
```

[Download]

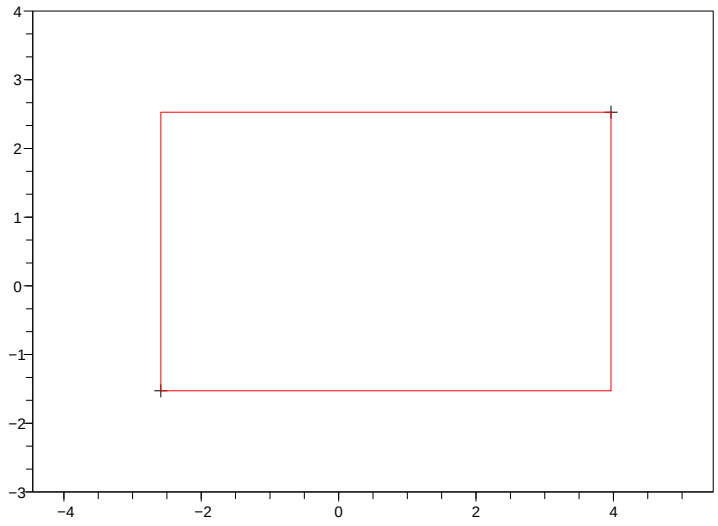
## 3.2 Rettangolo con centro in (x0,y0) e lati hx, hy




```
//:rett1.sce
//
// Rettangolo con centro in (x0,y0) e lati hx, hy
//
function x=fxr(x0,hx),x=x0+[-1, 1, 1,-1,-1]*hx/2,  endfunction;
function y=fyr(y0,hy),y=y0+[-1,-1, 1, 1,-1]*hy/2,  endfunction;
clf();
plot2d(0,0,1,"031",rect=[-2,-2,2,2]);
x0=0;
y0=0;
plot2d(x0,y0,-1)
plot2d(fxr(x0,2),fyr(x0,1),3);
//.
```

[Download]

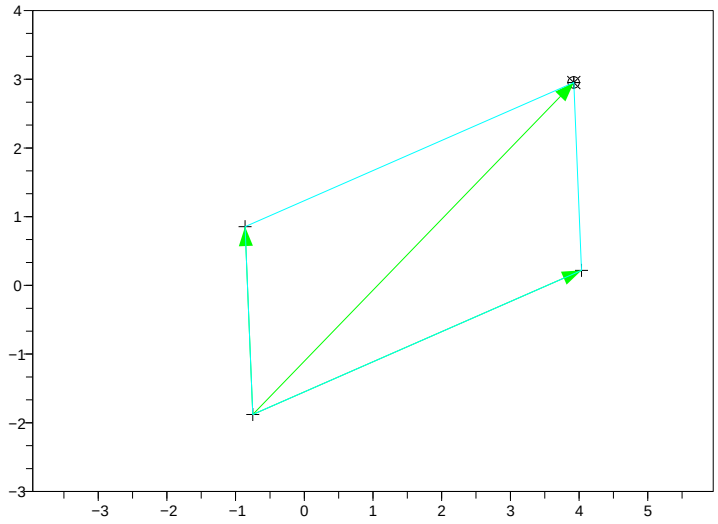
## 3.3 Rettangolo con un vertice in (x0,y0) e vertice opposto in (x1,y1)




```
//:rett2-m.sce
//
// Rettangolo con un vertice in (x0,y0) e vertice opposto in (x1,y1)
//
function x=fxrv(x0,hx),x=x0+[0,hx,hx,0 ,0], endfunction;
function y=fyrv(y0,hy),y=y0+[0,0 ,hy,hy,0], endfunction;

clf();
plot2d(0,0,1,"031",rect=[-3,-3,4,4]);

// un vertice
[b,x0,y0]=xclick();
plot2d(x0,y0,-1)

// il vertice opposto
[b,x1,y1]=xclick();
plot2d(x1,y1,-1)

hx=(x1-x0);
hy=(y1-y0);
plot2d(fxrv(x0,hx),fyrv(y0,hy),5);
//.
```

[Download]

## 3.4 Parallelogrammi e somme di vettori

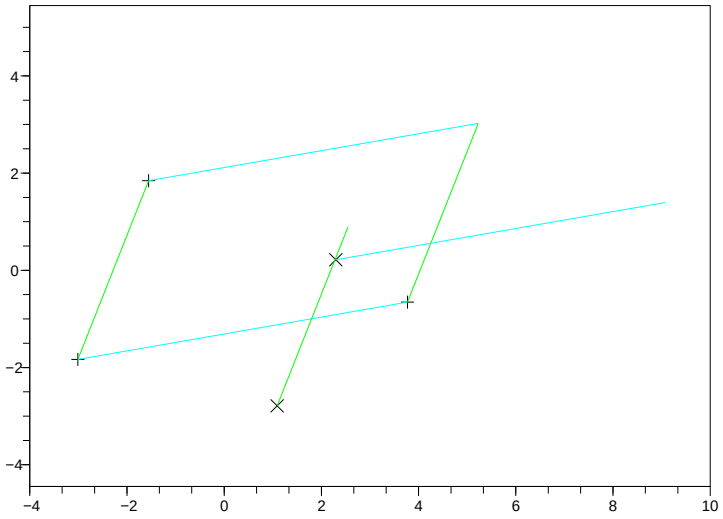



```
//:par1-m.sce
//
// Parallelogrammi e somme di vettori
//
clf();
plot2d(0,0,1,"031",rect=[-3,-3,4,4]);

[b,x0,y0]=xclick();
plot2d(x0,y0,-1)

[b,x1,y1]=xclick();
plot2d(x1,y1,-1)
xarrows([x0,x1],[y0,y1],6,3);

[b,x2,y2]=xclick();
plot2d(x2,y2,-1)
xarrows([x0,x2],[y0,y2],6,3);

// vettore (componenti)
v1x=(x1-x0);
v1y=(y1-y0);

// vettore (componenti)
v2x=(x2-x0);
v2y=(y2-y0);

// vertice (coordinate)
x3=x1+v2x;
y3=y1+v2y;
plot2d(x3,y3,-1)

// vertice (coordinate)
x3=x2+v1x;
y3=y2+v1y;
plot2d(x3,y3,-2)
```

```
// vertice (coordinate)
x3=x0+(v1x+v2x);
y3=y0+(v1y+v2y);
plot2d(x3,y3,-3)

xclick();
xarrows([x0,x3],[y0,y3],6,3);

xclick();
plot2d([x0,x1,x3,x2,x0],[y0,y1,y3,y2,y0],4);
//.
```

[Download]

## 3.5 Segmenti paralleli

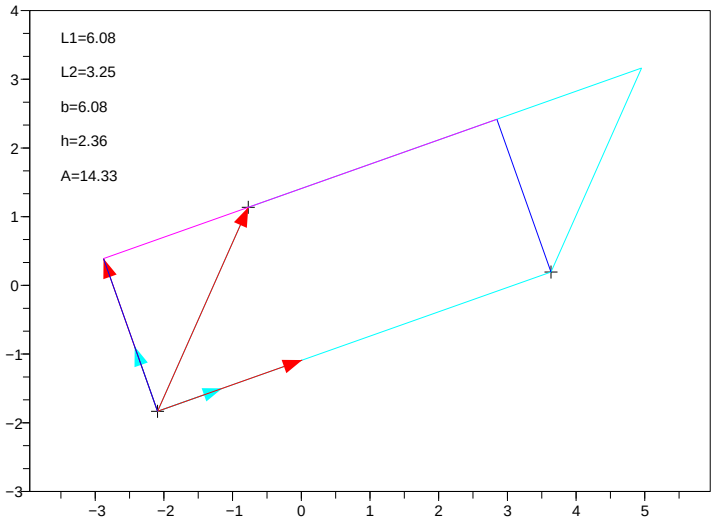



```
//:par3-m.sce
//
// Segmenti paralleli
//
clf();
plot2d(0,0,1,"031",rect=[-3,-3,4,4]);

// vertice (x0,y0)
[b,x0,y0]=xclick();
plot2d(x0,y0,-1)

// vertice (x1,y1)
[b,x1,y1]=xclick();
plot2d(x1,y1,-1)

// vertice (x2,y2)
[b,x2,y2]=xclick();
plot2d(x2,y2,-1)

// calcolo del primo vettore differenza
v1x=(x1-x0);
v1y=(y1-y0);

// calcolo del secondo vettore differenza
v2x=(x2-x0);
v2y=(y2-y0);

// disegno del primo lato
plot2d([x0,x0+v1x],[y0,y0+v1y],4);

// disegno del secondo lato
plot2d([x0,x0+v2x],[y0,y0+v2y],3);

// disegno del terzo lato
plot2d([(x0+v2x),(x0+v2x)+v1x],[(y0+v2y),(y0+v2y)+v1y],4);
```

```
// disegno del quarto lato
plot2d([(x0+v1x),(x0+v1x)+v2x],[(y0+v1y),(y0+v1y)+v2y],3);

// Altri segmenti paralleli ai lati

[b,xi,yi]=xclick();
plot2d(xi,yi,-2)
plot2d([xi,xi+v1x],[yi,yi+v1y],4);

[b,xj,yj]=xclick();
plot2d(xj,yj,-2)
plot2d([xj,xj+v2x],[yj,yj+v2y],3);
//.
```

[Download]

## 3.6 Costruzione del rettangolo di uguale area

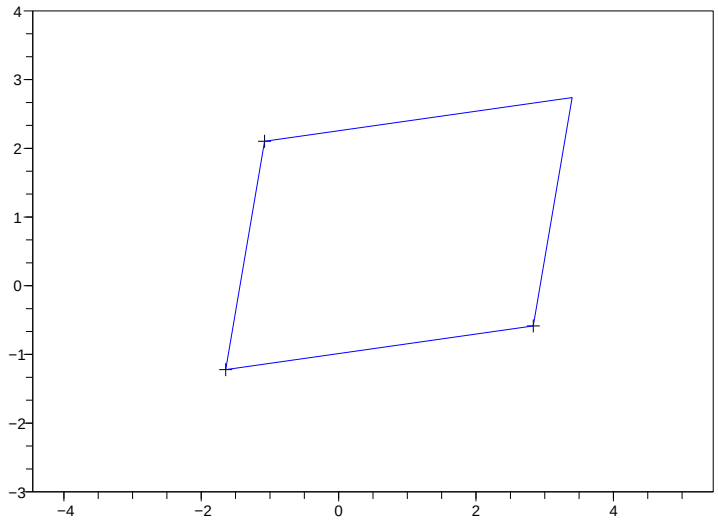



```
//:rettpar-m.sce
//
// Costruzione del rettangolo di uguale area
//
clf();
plot2d(0,0,1,"031",rect=[-3,-3,4,4]);

// vertice (x0,y0)
[b,x0,y0]=xclick();
plot2d(x0,y0,-1)

// vertice (x1,y1)
[b,x1,y1]=xclick();
plot2d(x1,y1,-1)

// vertice (x2,y2)
[b,x2,y2]=xclick();
plot2d(x2,y2,-1)

// primo vettore differenza
v1x=(x1-x0);
v1y=(y1-y0);

// secondo vettore differenza
v2x=(x2-x0);
v2y=(y2-y0);

// primo lato
plot2d([x0,x0+v1x],[y0,y0+v1y],4);
// secondo lato
plot2d([x0,x0+v2x],[y0,y0+v2y],4);
// terzo lato
plot2d([(x0+v2x),(x0+v2x)+v1x],[(y0+v2y),(y0+v2y)+v1y],4);
// quarto lato
plot2d([(x0+v1x),(x0+v1x)+v2x],[(y0+v1y),(y0+v1y)+v2y],4);
```

```
// Lunghezze dei lati

L1=sqrt(v1x^2+v1y^2);
L2=sqrt(v2x^2+v2y^2);

xstring(-3.5,3.5,"L1="+string(round(L1*100)/100))
xstring(-3.5,3.0,"L2="+string(round(L2*100)/100))

// Base ortonormale

vtx=v1x/L1;
vty=v1y/L1;

vnx=-vty;
vny=vtx;

xarrows([x0,x0+vtx],[y0,y0+vty],6,4);
xarrows([x0,x0+vnx],[y0,y0+vny],6,4);

// Calcolo delle proiezioni ortogonali del secondo vettore differenza

h=v2x*vnx+v2y*vny;
s=v2x*vtx+v2y*vty;
xarrows([x0,x0+v2x],[y0,y0+v2y],6,5);
xarrows([x0,x0+s*vtx],[y0,y0+s*vty],6,5);
xarrows([x0,x0+h*vnx],[y0,y0+h*vny],6,5);

// Rettangolo

plot2d([x0,x0+h*vnx],[y0,y0+h*vny],2);
plot2d([(x0+v1x),(x0+v1x)+h*vnx],[(y0+v1y),(y0+v1y)+h*vny],2);
plot2d([(x0+h*vnx),(x0+h*vnx)+v1x],[(y0+h*vny),(y0+h*vny)+v1y],6);

// Area

b=L1;
A=b*h;
det([v1x v2x; v1y v2y])

xstring(-3.5,2.5,"b="+string(round(b*100)/100))
xstring(-3.5,2.0,"h="+string(round(h*100)/100))
xstring(-3.5,1.5,"A="+string(round(A*100)/100))
//.
```

[Download]

## 3.7 Parallelogramma generato da due vettori con vertice in (x0,y0)

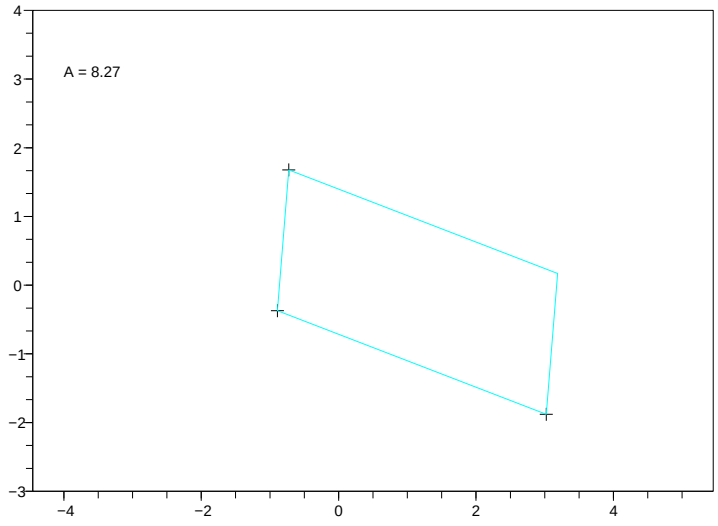



```
//:par4-m.sce
//
// Parallelogramma generato da due vettori con vertice in (x0,y0)
//
function x=fxp(x0,v1x,v2x),  x=x0+[0, v1x, v1x+v2x, v2x, 0], endfunction;
function y=fyp(y0,v1y,v2y),  y=y0+[0, v1y, v1y+v2y, v2y, 0], endfunction;

clf();
plot2d(0,0,1,"031",rect=[-3,-3,4,4]);

// vertice (x0,y0)
[b,x0,y0]=xclick();
plot2d(x0,y0,-1)

// vertice (x1,y1)
[b,x1,y1]=xclick();
plot2d(x1,y1,-1)

// vertice (x2,y2)
[b,x2,y2]=xclick();
plot2d(x2,y2,-1)

// primo vettore differenza
v1x=(x1-x0);
v1y=(y1-y0);

// secondo vettore differenza
v2x=(x2-x0);
v2y=(y2-y0);

plot2d(fxp(x0,v1x,v2x),fyp(y0,v1y,v2y),2);

// area
A=det([v1x,v2x;v1y,v2y])
```

```
//.
```

[Download]

## 3.8 Coppie di coordinate e coppie di componenti come matrici colonna

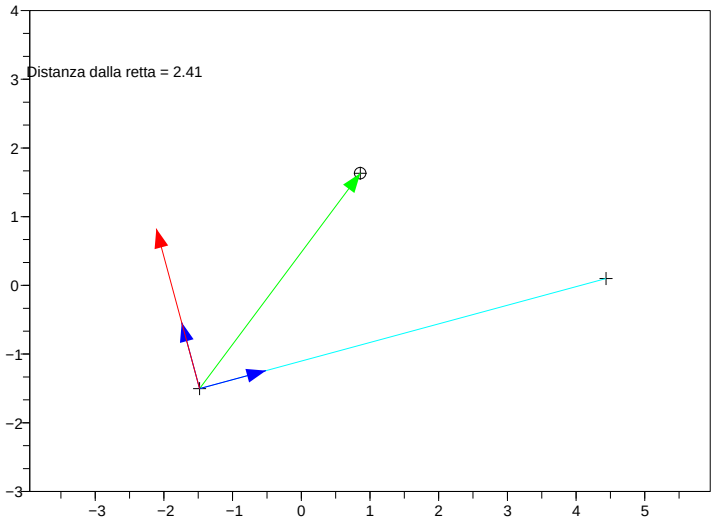



```
//:coppie-m.sce
//
// Coppie di coordinate e coppie di componenti come matrici colonna
//
clear
clf();
plot2d(0,0,1,"031",rect=[-3,-3,4,4]);

// vertice p0
[b,x0,y0]=xclick();
plot2d(x0,y0,-1)
p0=[x0;y0];

// vertice p1
[b,x1,y1]=xclick();
plot2d(x1,y1,-1)
p1=[x1;y1];

// vertice p2
[b,x2,y2]=xclick();
plot2d(x2,y2,-1)
p2=[x2;y2];

// vettore p1-p0
v1=p1-p0;

// vettore p2-p0
v2=p2-p0;

p3=p0+v1+v2;

par=[p0,p1,p3,p2,p0];

plot2d(par(1,:),par(2,:),4)
```

```
A=det([v1 v2]);
xstring(-4,3,"A = "+string(round(A*100)/100))
//.
```

[Download]

## 3.9 Distanza di un punto p0 da una retta per p1 e p2

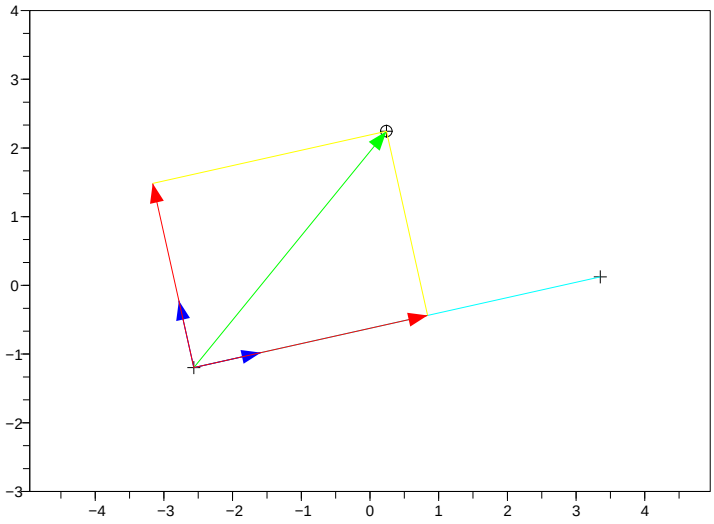



```
//:dist-m.sce
//
// Distanza di un punto p0 da una retta per p1 e p2
//
clf();
plot2d(0,0,1,"031",rect=[-3,-3,4,4]);

// un punto della retta (x1,y1)
[b,x1,y1]=xclick();
plot2d(x1,y1,-1)
p1=[x1;y1];

// un altro punto della retta (x2,y2)
[b,x2,y2]=xclick();
plot2d(x2,y2,-1)
p2=[x2;y2];

// vettore tangente alla retta
vr=p2-p1;

// segmento da p1 a p2 (solo un tratto della retta)
seg=[p1,p2];
plot2d(seg(1,:),seg(2,:),4);

// il punto (x0,y0) di cui calcolare la distanza dalla retta
[b,x0,y0]=xclick();
plot2d(x0,y0,-3)
p0=[x0;y0];

// vettore differenza
vd=p0-p1;

xclick();
fr1=[p1,p0]
xarrows(fr1(1,:),fr1(2,:),6,3);
```

```
// Base ortonormale

vt=vr/norm(vr);
vn=[-vt(2);vt(1)];

xclick();
frt=[p1,p1+vt];
frn=[p1,p1+vn];
xarrows(frt(1,:),frt(2,:),6,2);
xarrows(frn(1,:),frn(2,:),6,2);

// Calcolo delle proiezioni ortogonali del vettore differenza

xclick();
// prodotto scalare come prodotto riga per colonna
h=(vd')*vn;
s=(vd')*vt;
frh=[p1,p1+h*vn];
xarrows(frh(1,:),frh(2,:),6,5);

// h puo' essere < 0
//(il segno distingue i punti "a sinistra" da quelli "a destra")
xstring(-4,3,"Distanza dalla retta = "+string(round(100*h)/100));
//.
```

[Download]

## 3.10 Proiezione ortogonale

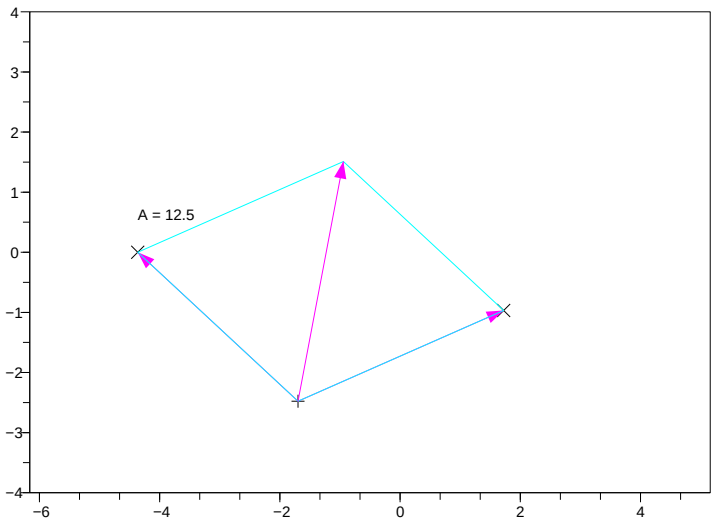



```
//:prort-m.sce
//
// Proiezione ortogonale
//
clf();
plot2d(0,0,1,"031",rect=[-3,-3,4,4]);

// due punti individuano una retta
[b,x1,y1]=xclick();
plot2d(x1,y1,-1)
p1=[x1;y1];

[b,x2,y2]=xclick();
plot2d(x2,y2,-1)
p2=[x2;y2];

// vettore tangente alla retta
vr=p2-p1;

// segmento da p1 a p2 (solo un tratto della retta)
sr=[p1,p2];
plot2d(sr(1,:),sr(2,:),4);

// un punto fuori dalla retta
[b,x0,y0]=xclick();
plot2d(x0,y0,-3)
p0=[x0;y0];

// vettore p0-p1
v=p0-p1;

xclick();
s0=[p1,p0]
xarrows(s0(1,:),s0(2,:),6,3);
```

```
// Base ortonormale

et=vr/norm(vr);
en=[-et(2);et(1)];

xclick();
st=[p1,p1+et];
sn=[p1,p1+en];
xarrows(st(1,:),st(2,:),6,2);
xarrows(sn(1,:),sn(2,:),6,2);

// Calcolo delle proiezioni ortogonali del vettore p0-p1

xclick();
// prodotto scalare come prodotto riga per colonna
hn=(v')*en;
ht=(v')*et;
// proiezioni
pron=[p1,p1+hn*en];
xarrows(pron(1,:),pron(2,:),6,5);
prot=[p1,p1+ht*et];
xarrows(prot(1,:),prot(2,:),6,5);

// hn puo' essere < 0
//(il segno distingue i punti "a sinistra" da quelli "a destra")

// completamento della figura
ret=[p1+ht*et,p0,p1+hn*en];
plot2d(ret(1,:),ret(2,:),7)

//.
```

[Download]

## 3.11 Parallelogramma generato da due vettori con vertice in p0

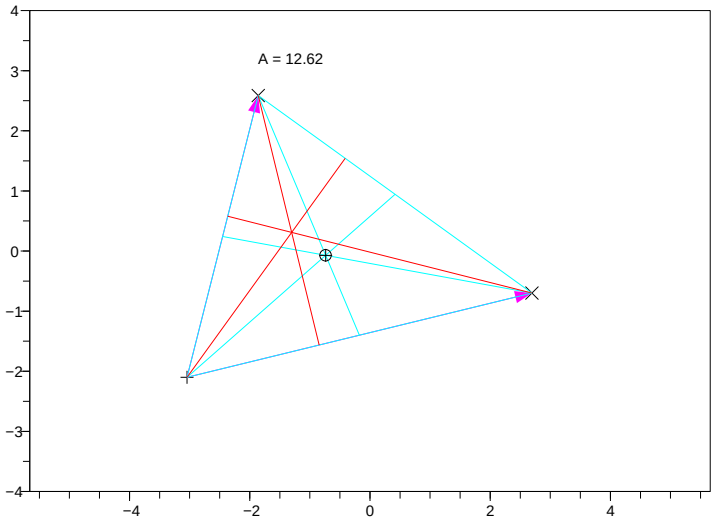



```
//:parall-m:sce
//
// Parallelogramma generato da due vettori con vertice in p0
//
function p=parallelogramma(p0,u,v),
      p=[p0,p0+u,p0+u+v,p0+v,p0]
endfunction;

clf();
plot2d(0,0,1,"031",rect=[-4,-4,4,4]);

[b,x0,y0]=xclick();
p0=[x0;y0];
plot2d(x0,y0,-1);

[b,x1,y1]=xclick();
plot2d(x1,y1,-2);
p1=[x1;y1];

v1=p1-p0;
fr=[p0,p1];
xarrows(fr(1,:),fr(2,:),6,6);

[b,x2,y2]=xclick();
plot2d(x2,y2,-2);
p2=[x2;y2];

v2=p2-p0;
fr=[p0,p2];
xarrows(fr(1,:),fr(2,:),6,6);

xclick;
fig=parallelogramma(p0,v1,v2);
plot2d(fig(1,:),fig(2,:),4)
```

```
xclick;
fr=[p0,p1+v2];
xarrows(fr(1,:),fr(2,:),6,6);

// area
A=det([v1,v2])
xstring(x2,y2+0.5,"A = "+string(round(A*100)/100))
//.
```

[Download]

## 3.12 Triangolo, mediane e altezze

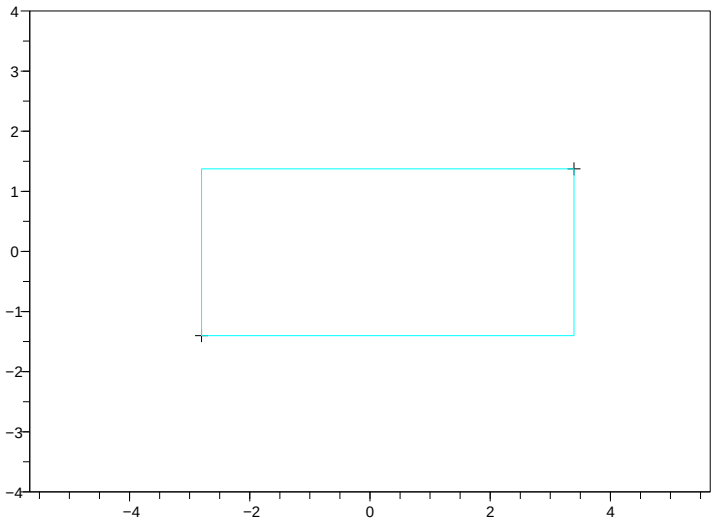



```
//:tria-m.sce
//
// Triangolo, mediane e altezze
//
function p=triangolo(p0,u,v),
      p=[p0,p0+u,p0+v,p0]
endfunction;

clf();
plot2d(0,0,1,"031",rect=[-4,-4,4,4]);

[b,x0,y0]=xclick();
p0=[x0;y0];
plot2d(x0,y0,-1);

[b,x1,y1]=xclick();
plot2d(x1,y1,-2);
p1=[x1;y1];

v1=p1-p0;
fr=[p0,p1];
xarrows(fr(1,:),fr(2,:),6,6);

[b,x2,y2]=xclick();
plot2d(x2,y2,-2);
p2=[x2;y2];

v2=p2-p0;
fr=[p0,p2];
xarrows(fr(1,:),fr(2,:),6,6);

fig=triangolo(p0,v1,v2);
plot2d(fig(1,:),fig(2,:),4)
//
// Area
```

```
//
A=det([v1,v2])/2
xstring(x2,y2+0.5,"A = "+string(round(A*100)/100))
//
// Mediane
//
xclick;
p0m=p1+(p2-p1)/2;
mediana=[p0,p0m];
plot2d(mediana(1,:),mediana(2,:),4);

xclick;
p1m=p0+(p2-p0)/2;
mediana=[p1,p1m];
plot2d(mediana(1,:),mediana(2,:),4);

xclick;
p2m=p0+(p1-p0)/2;
mediana=[p2,p2m];
plot2d(mediana(1,:),mediana(2,:),4);
//
// Baricentro
//
xclick;
p0m=(p1+p2)/2;
g=p0+(p0m-p0)*2/3;
plot2d(g(1),g(2),-3)
//
// Altezze
//
xclick;
h2=det([v1,v2])/norm(v1);
vn=[-v1(2);v1(1)]/norm(v1);
segmento=[p2,p2-h2*vn];
plot2d(segmento(1,:),segmento(2,:),5)

xclick;
h1=det([v1,v2])/norm(v2);
vn=[-v2(2);v2(1)]/norm(v2);
segmento=[p1,p1+h1*vn];
plot2d(segmento(1,:),segmento(2,:),5)

xclick;
v3=v2-v1;
h0=det([v1,v2])/norm(v3);
vn=[-v3(2);v3(1)]/norm(v3);
segmento=[p0,p0-h0*vn];
plot2d(segmento(1,:),segmento(2,:),5)
//.
```

[Download]

## 3.13 Rettangolo con lati orizzontali e verticali




```
//:retoriz-m.sce
//
// Rettangolo con lati orizzontali e verticali
//
function p=rettangolo(p0,p1),
      vd=p1-p0;
      u=[vd(1);0];
      v=[0;vd(2)];
      p=[p0,p0+u,p1,p0+v,p0]
endfunction;

clf();
plot2d(0,0,1,"031",rect=[-4,-4,4,4]);

// un vertice
[b,x0,y0]=xclick();
p0=[x0;y0];
plot2d(x0,y0,-1);

// il vertice opposto
[b,x1,y1]=xclick();
plot2d(x1,y1,-1);
p1=[x1;y1];

fig=rettangolo(p0,p1);
plot2d(fig(1,:),fig(2,:),4)
//.
```

[Download]

## 3.14 Funzioni (Parallelogrammi)

```
//:lib_parallelogrammi.sce
//
// Funzioni (Parallelogrammi)
//
// ------<funzioni>-<inizio>----------------------------------------

// Parallelogramma generato da due vettori con vertice in p0
function p=parallelogramma(p0,u,v),
      p=[p0,p0+u,p0+u+v,p0+v,p0]
endfunction;

// Triangolo generato da due vettori con vertice in p0
function p=triangolo(p0,u,v),
      p=[p0,p0+u,p0+v,p0]
endfunction;

// Rettangolo con lati orizzontali e verticali
function p=rettangolo(p0,p1),
      vd=p1-p0;
      u=[vd(1);0];
      v=[0;vd(2)];
      p=[p0,p0+u,p1,p0+v,p0]
endfunction;
// ------<funzioni>-<fine>-----------------------------------------
//.
```

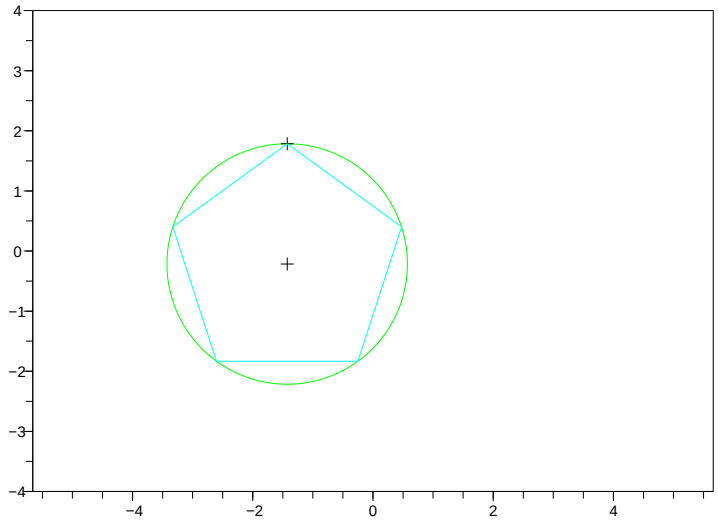
[Download]

# Capitolo 4

# Area e baricentro di poligoni

Un poligono è descritto dalla $n$-pla delle coppie di coordinate dei vertici. Queste possono essere generate, come nel caso dei poligoni regolari, o più in generale assegnate, ad esempio utilizzando il mouse. L'area di un poligono si può calcolare come somma delle aree dei triangoli formati da ciascun lato e un polo scelto ad arbitrio. Questo calcolo può essere esteso al calcolo delle coordinate del baricentro.

## 4.1 Poligono regolare con centro in p0

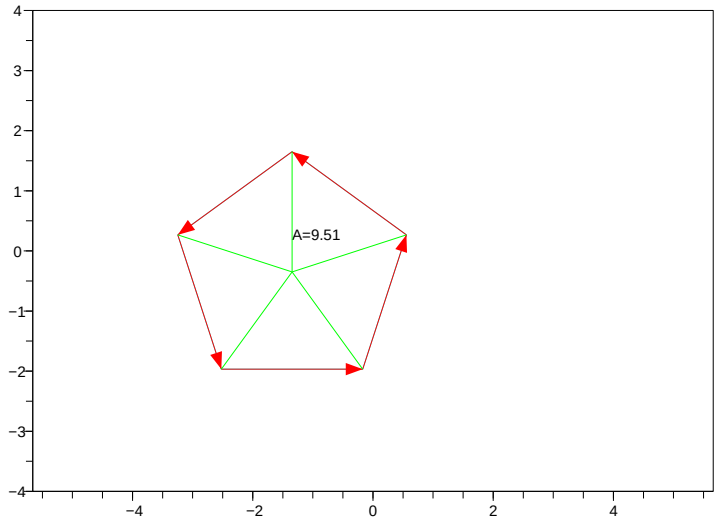



```
//:poli-m.sce
//
// Poligono regolare con centro in p0
//
// Circonferenza di verso antiorario
function p=cira(c0,r)
      n=360; a=%pi/2+[0:n]/n*2*%pi;
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;
// Circonferenza di verso orario
function p=ciro(c0,r)
      n=360; a=%pi/2-[0:n]/n*2*%pi;
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;
// Poligono regolare (antiorario)
function p=poliga(n,c0,r)
      a=%pi/2+[0:n]/n*2*%pi;
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;
// Poligono regolare (orario)
function p=poligo(n,c0,r)
      a=%pi/2-[0:n]/n*2*%pi;
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;
clf();
plot2d(0,0,1,"031",rect=[-4,-4,4,4]);

[b,x0,y0]=xclick();
p0=[x0;y0];
plot2d(x0,y0,-1);

r=2;
cerchio=cira(p0,r);
plot2d(cerchio(1,:),cerchio(2,:),3);
n=5;
```

```
pol=poliga(n,p0,r);
plot2d(pol(1,1),pol(2,1),-1);
plot2d(pol(1,:),pol(2,:),4);
//
// Si mette in evidenza il verso del poligono
for i=1:n
      xarrows(pol(1,i:i+1),pol(2,i:i+1),6,5);
end
//.
```

[Download]

## 4.2 Area di un poligono regolare (verso antiorario)

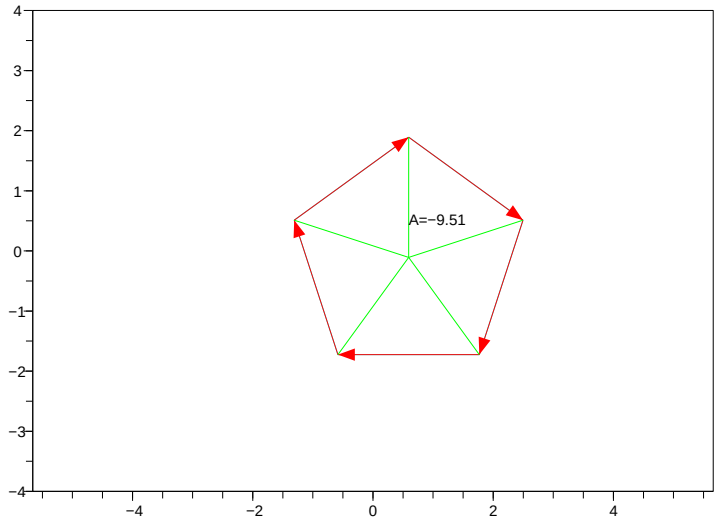



```
//:apa-m.sce
//
// Area di un poligono regolare (verso antiorario)
//
// Poligono regolare (antiorario)
function p=poliga(n,c0,r)
      a=%pi/2+[0:n]/n*2*%pi;
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;
// Poligono regolare (orario)
function p=poligo(n,c0,r)
      a=%pi/2-[0:n]/n*2*%pi;
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;
// Area di un triangolo
function area=areatriangolo(p0,p1,p2)
      area=det([p1-p0,p2-p0])/2
endfunction
//
clf();
plot2d(0,0,1,"031",rect=[-4,-4,4,4]);

[b,x0,y0]=xclick();
p0=[x0;y0];

r=2;
n=5;
pol=poliga(n,p0,r);
plot2d(pol(1,:),pol(2,:),4);
//
// Si mette in evidenza il verso del poligono
for i=1:n
      xarrows(pol(1,i:i+1),pol(2,i:i+1),6,5);
end
//
```

```
A=0;
for i=1:n
      p1=pol(:,i);
      p2=pol(:,i+1);
      A=A+areatriangolo(p0,p1,p2);
end
//
xstring(p0(1),p0(2)+0.5,"A="+string(round(A*100)/100));

for i=1:n
      raggio=[p0,pol(:,i)];
      plot2d(raggio(1,:),raggio(2,:),3)
end
//.
```

[Download]

## 4.3 Area di un poligono regolare (verso orario)

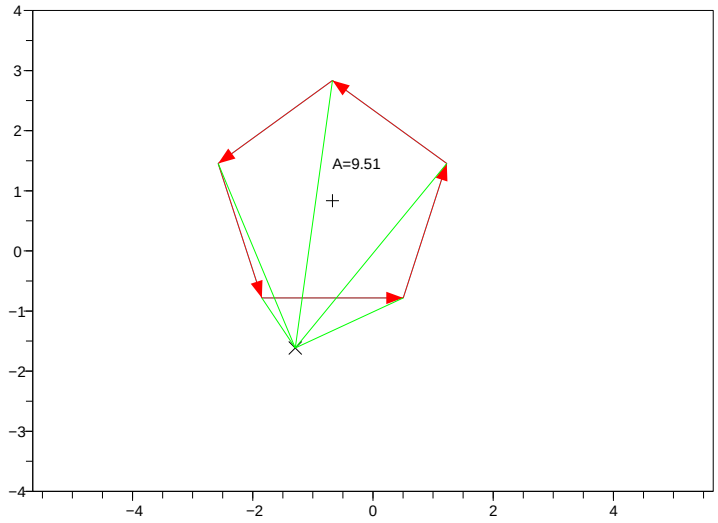



```
//:apo-m.sce
//
// Area di un poligono regolare (verso orario)
//
// Poligono regolare (antiorario)
function p=poliga(n,c0,r)
      a=%pi/2+[0:n]/n*2*%pi;
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;
// Poligono regolare (orario)
function p=poligo(n,c0,r)
      a=%pi/2-[0:n]/n*2*%pi;
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;
// Area di un triangolo
function area=areatriangolo(p0,p1,p2)
      area=det([p1-p0,p2-p0])/2
endfunction
//
clf();
plot2d(0,0,1,"031",rect=[-4,-4,4,4]);

[b,x0,y0]=xclick();
p0=[x0;y0];

r=2;
n=5;
pol=poligo(n,p0,r);
plot2d(pol(1,:),pol(2,:),4);
//
// Si mette in evidenza il verso del poligono
for i=1:n
      xarrows(pol(1,i:i+1),pol(2,i:i+1),6,5);
end
// Si disegna la suddivisione in triangoli
```

```
for i=1:n
      raggio=[p0,pol(:,i)];
      plot2d(raggio(1,:),raggio(2,:),3)
end
//
A=0;
for i=1:n
      p1=pol(:,i);
      p2=pol(:,i+1);
      A=A+areatriangolo(p0,p1,p2);
end
//
xstring(p0(1),p0(2)+0.5,"A="+string(round(A*100)/100));
//.
```

[Download]

## 4.4 Area di un poligono regolare (polo diverso dal centro)

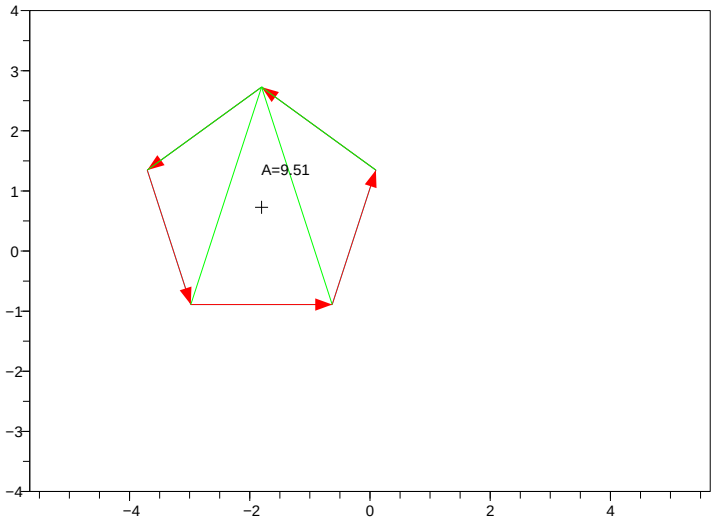



```
//:apolo-m.sce
//
// Area di un poligono regolare (polo diverso dal centro)
//
// Poligono regolare (antiorario)
function p=poliga(n,c0,r)
      a=%pi/2+[0:n]/n*2*%pi;
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;
// Poligono regolare (orario)
function p=poligo(n,c0,r)
      a=%pi/2-[0:n]/n*2*%pi;
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;
// Area di un triangolo
function area=areatriangolo(p0,p1,p2)
      area=det([p1-p0,p2-p0])/2
endfunction
//
clf();
plot2d(0,0,1,"031",rect=[-4,-4,4,4]);

[b,cx0,cy0]=xclick();
c0=[cx0;cy0];
plot2d(cx0,cy0,-1);

r=2;
n=5;
pol=poliga(n,c0,r);
plot2d(pol(1,:),pol(2,:),4);
//
// Si mette in evidenza il verso del poligono
for i=1:n
      xarrows(pol(1,i:i+1),pol(2,i:i+1),6,5);
end
```

```
// Si assegna il polo della suddivisione in triangoli
[b,x0,y0]=xclick();
p0=[x0;y0];
plot2d(x0,y0,-2);
// Si disegna la suddivisione in triangoli
for i=1:n
      raggio=[p0,pol(:,i)];
      plot2d(raggio(1,:),raggio(2,:),3)
end
//
A=0;
for i=1:n
      p1=pol(:,i);
      p2=pol(:,i+1);
      A=A+areatriangolo(p0,p1,p2);
end
//
xstring(c0(1),c0(2)+0.5,"A="+string(round(A*100)/100));
//.
```

[Download]

## 4.5 Area di un poligono regolare

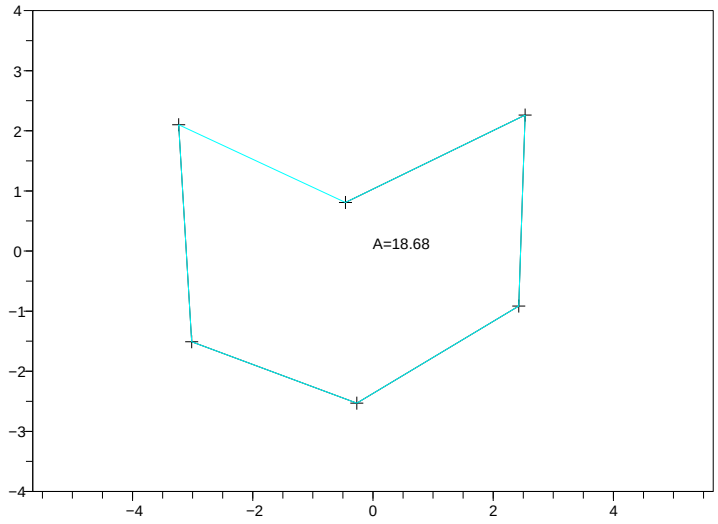



```
//:apol-m.sce
//
// Area di un poligono regolare
//
// Poligono regolare (antiorario)
function p=poliga(n,c0,r)
      a=%pi/2+[0:n]/n*2*%pi;
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;
// Poligono regolare (orario)
function p=poligo(n,c0,r)
      a=%pi/2-[0:n]/n*2*%pi;
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;
// Area di un triangolo
function area=areatriangolo(p0,p1,p2)
      area=det([p1-p0,p2-p0])/2
endfunction
// Area di un poligono suddiviso in triangoli
function area=areapol(pol)
      [nr,nc]=size(pol);
      p0=pol(:,1);
      area=0
      for i=1:nc-1
            p1=pol(:,i);
            p2=pol(:,i+1);
            area=area+areatriangolo(p0,p1,p2)
      end
endfunction
//
clf();
plot2d(0,0,1,"031",rect=[-4,-4,4,4]);

[b,cx0,cy0]=xclick();
c0=[cx0;cy0];
```

```
plot2d(cx0,cy0,-1);

r=2;
n=5;
pol=poliga(n,c0,r);
plot2d(pol(1,:),pol(2,:),4);
//
// Si mette in evidenza il verso del poligono
for i=1:n
      xarrows(pol(1,i:i+1),pol(2,i:i+1),6,5);
end
// Si disegna la suddivisione in triangoli
p0=pol(:,1);
for i=1:n
      raggio=[p0,pol(:,i)];
      plot2d(raggio(1,:),raggio(2,:),3)
end
//
A=areapol(pol);
//
xstring(c0(1),c0(2)+0.5,"A="+string(round(A*100)/100));

//.
```

[Download]

## 4.6 Area di un poligono generico di n vertici

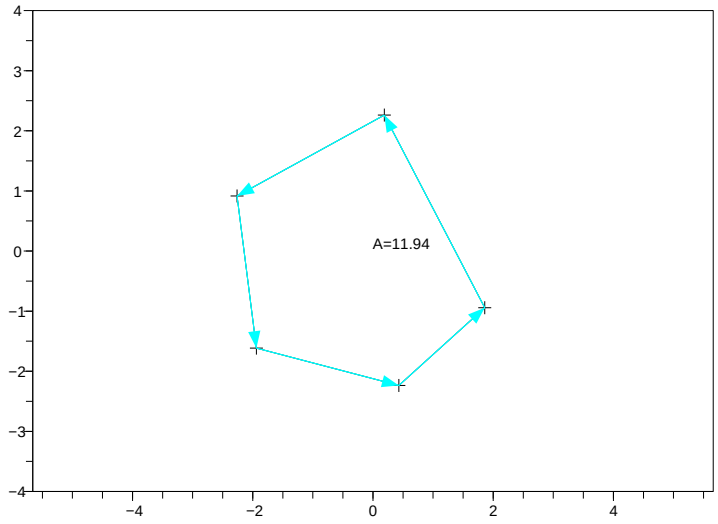



```
//:apolg-m.sce
//
// Area di un poligono generico di n vertici
//
// Area di un triangolo
function area=areatriangolo(p0,p1,p2)
      area=det([p1-p0,p2-p0])/2
endfunction
// Area di un poligono suddiviso in triangoli
function area=areapol(pol)
      [nr,nc]=size(pol);
      p0=pol(:,1);
      area=0
      for i=1:nc-1
            p1=pol(:,i);
            p2=pol(:,i+1);
            area=area+areatriangolo(p0,p1,p2)
      end
endfunction
//
clf();
plot2d(0,0,1,"031",rect=[-4,-4,4,4]);

n=6;
pol=[0;0];
for i=1:n
      [b,xi,yi]=xclick();
      pol(:,i)=[xi;yi];
      plot2d(xi,yi,-1)
      if i>1 then
            plot2d(pol(1,i-1:i),pol(2,i-1:i)),
      end
end
pol(:,n+1)=pol(:,1);
```

```
plot2d(pol(1,:),pol(2,:),4);
//
// Si mette in evidenza il verso del poligono
xclick();
for i=1:n
      xarrows(pol(1,i:i+1),pol(2,i:i+1),6,5);
end
// Si disegna la suddivisione in triangoli
p0=pol(:,1);
for i=1:n
      raggio=[p0,pol(:,i)];
      plot2d(raggio(1,:),raggio(2,:),3)
end
//
A=areapol(pol);
//
xstring(0,0,"A="+string(round(A*100)/100));
//.
```

[Download]

## 4.7 Area di un poligono generico

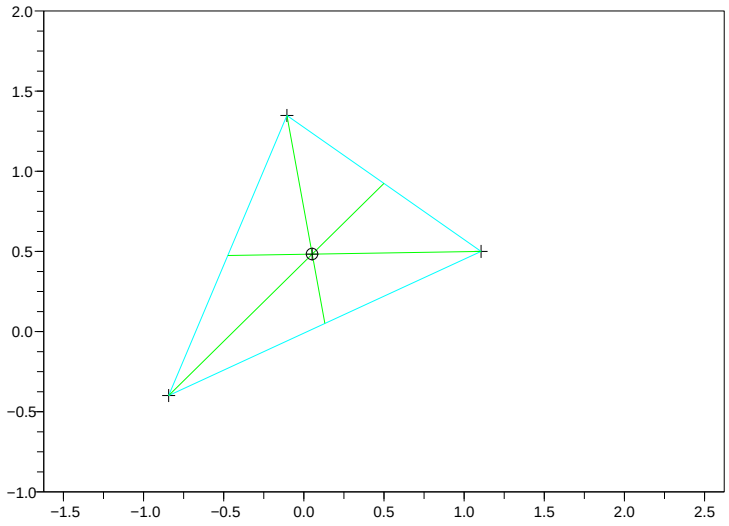



```
//:apolge-m.sce
//
// Area di un poligono generico
//
// Area di un triangolo
function area=areatriangolo(p0,p1,p2)
      area=det([p1-p0,p2-p0])/2
endfunction
// Area di un poligono suddiviso in triangoli
function area=areapol(pol)
      [nr,nc]=size(pol);
      p0=pol(:,1);
      area=0
      for i=1:nc-1
            p1=pol(:,i);
            p2=pol(:,i+1);
            area=area+areatriangolo(p0,p1,p2)
      end
endfunction
//
clf();
plot2d(0,0,1,"031",rect=[-4,-4,4,4]);

// Assegnare i vertici con il tasto sinistro
// Terminare con un tasto diverso dal sinistro
n=0;
pol=[];
[b,xv,yv]=xclick();
while (b==0)|(b==3)
      n=n+1;
      pol(:,n)=[xv;yv];
      plot2d(xv,yv,-1)
      if n>1 then plot2d(pol(1,n-1:n),pol(2,n-1:n)), end
      [b,xv,yv]=xclick();
end
```

```
if n>1 then
      pol(:,n+1)=pol(:,1);
      plot2d(pol(1,n:n+1),pol(2,n:n+1))
end
plot2d(pol(1,:),pol(2,:),4);
//
// Si mette in evidenza il verso del poligono
xclick();
for i=1:n
      xarrows(pol(1,i:i+1),pol(2,i:i+1),6,4);
end
//
A=areapol(pol);
//
xstring(0,0,"A="+string(round(A*100)/100));
//
// Si disegna la suddivisione in triangoli
p0=pol(:,1);
for i=1:n
      raggio=[p0,pol(:,i)];
      plot2d(raggio(1,:),raggio(2,:),3)
end
//.
```

[Download]

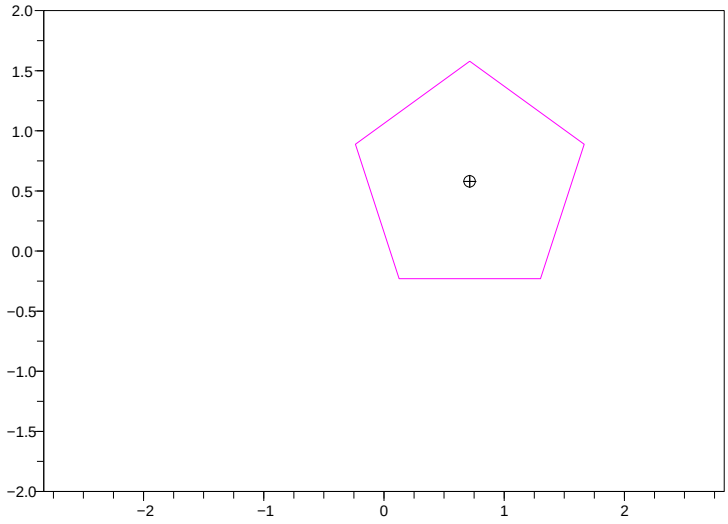
## 4.8 Baricentro di un triangolo generico




```
//:batri-m.sce
//
// Baricentro di un triangolo generico
//
clf();
plot2d(0,0,1,"031"," ",rect=[-1,-1,2,2]);

[b,x1,y1]=xclick();
plot2d(x1,y1,-1);
p1=[x1;y1];

[b,x2,y2]=xclick();
plot2d(x2,y2,-1);
p2=[x2;y2];

[b,x3,y3]=xclick();
p3=[x3;y3];
plot2d(x3,y3,-1);

tri=[p1,p2,p3,p1];
plot2d(tri(1,:),tri(2,:),4);

// mediane
p3m=p1+(p2-p1)/2; med=[p3,p3m]; plot2d(med(1,:),med(2,:),3)
p1m=p2+(p3-p2)/2; med=[p1,p1m]; plot2d(med(1,:),med(2,:),3)
p2m=p3+(p1-p3)/2; med=[p2,p2m]; plot2d(med(1,:),med(2,:),3)

// baricentro
g=p1+(p1m-p1)*2/3;
plot2d(g(1),g(2),-3);
//.
```

[Download]

## 4.9 Baricentro di un poligono regolare

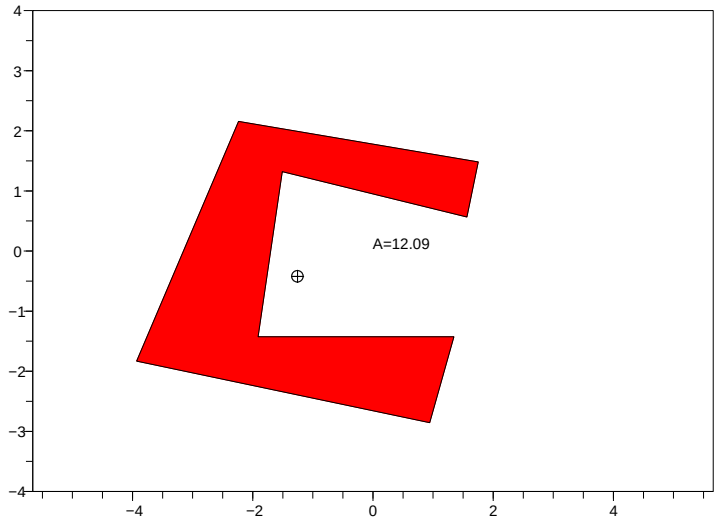



```
//:bapol-m.sce
//
// Baricentro di un poligono regolare
//
// Poligono regolare (antiorario)
function p=poliga(n,c0,r)
      a=%pi/2+[0:n]/n*2*%pi;
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;
// Poligono regolare (orario)
function p=poligo(n,c0,r)
      a=%pi/2-[0:n]/n*2*%pi;
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;
// Area di un triangolo
function area=areatriangolo(p0,p1,p2)
      area=det([p1-p0,p2-p0])/2
endfunction
// Baricentro di un triangolo
function g=bartriangolo(p0,p1,p2)
      p0m=p1+(p2-p1)/2;
      g=p0+(p0m-p0)*2/3;
endfunction
//

clf();
plot2d(0,0,1,"031"," ",[-2,-2,2,2]);
[b,cx0,cy0]=xclick();
c0=[cx0;cy0];
plot2d(cx0,cy0,-1);
pol=poligo(5,c0,1);
//
[nr,nc]=size(pol);
p0=pol(:,1);
area=0;
```

```
rg=[0;0];
p0=pol(:,1);
for i=1:nc-1
      p1=pol(:,i);
      p2=pol(:,i+1);
      areai=areatriangolo(p0,p1,p2);
      area=area+areai;
      rgi=bartriangolo(p0,p1,p2)-p0;
      rg=rg+areai*rgi;
end
g=p0+rg/area;

plot2d(pol(1,:),pol(2,:),6);
plot2d(g(1),g(2),-3);
//.
```

[Download]

## 4.10 Baricentro di un poligono generico




```
//:bapolg-m.sce
//
// Baricentro di un poligono generico
//
// Area di un triangolo
function area=areatriangolo(p0,p1,p2)
      area=det([p1-p0,p2-p0])/2
endfunction
// Area di un poligono suddiviso in triangoli
function area=areapol(pol)
      [nr,nc]=size(pol);
      p0=pol(:,1);
      area=0
      for i=1:nc-1
            p1=pol(:,i);
            p2=pol(:,i+1);
            area=area+areatriangolo(p0,p1,p2)
      end
endfunction
// Baricentro di un triangolo
function g=bartriangolo(p0,p1,p2)
      p0m=p1+(p2-p1)/2;
      g=p0+(p0m-p0)*2/3;
endfunction
// Baricentro di un poligono
function g=barpol(pol)
      [nr,nc]=size(pol);
      p0=pol(:,1);
      area=0;
      rg=[0;0];
      for i=1:nc-1
            p1=pol(:,i);
            p2=pol(:,i+1);
            areai=areatriangolo(p0,p1,p2);
            area=area+areai;
```

```
            rgi=bartriangolo(p0,p1,p2)-p0;
            rg=rg+areai*rgi;
      end
      g=p0+rg/area
endfunction
// Poligono assegnato con il mouse
function pol=poligono_m()
// Assegnare i vertici con il tasto sinistro
// Terminare con un tasto diverso dal sinistro
      n=0;
      pol=[];
      [b,xv,yv]=xclick();
      while (b==0)|(b==3)
            n=n+1;
            pol(:,n)=[xv;yv];
            plot2d(xv,yv,-1)
            if n>1 then plot2d(pol(1,n-1:n),pol(2,n-1:n)), end
            [b,xv,yv]=xclick();
      end
      if n>1 then
            pol(:,n+1)=pol(:,1);
            plot2d(pol(1,n:n+1),pol(2,n:n+1))
      end
endfunction
//
clf();
plot2d(0,0,1,"031",rect=[-4,-4,4,4]);

pol=poligono_m();

plot2d(pol(1,:),pol(2,:),4);
//
A=areapol(pol);
//
xstring(0,0,"A="+string(round(A*100)/100));
//
xclick();
g=barpol(pol);
//
// Il baricentro viene riportato su un nuovo disegno
clf();
plot2d(0,0,0,"031",rect=[-4,-4,4,4]);
plot2d(pol(1,:),pol(2,:),5);
plot2d(g(1),g(2),-3);
// Il poligono viene riempito di colore
xfpoly(pol(1,:),pol(2,:),5);
plot2d(g(1),g(2),-3);
// Il valore dell'area viene scritto vicino al baricentro
xstring(g(1),g(2)+0.5,"A="+string(round(A*100)/100));

//.
```

[Download]

## 4.11 Funzioni (Cerchi e poligoni)

```
//:lib_poligoni.sce
//
// Funzioni (Cerchi e poligoni)
//
// ------<funzioni>-<inizio>--------------------------------------

// Poligono (verso antiorario)
function p=poliga(n,c0,r)
      a=[0:n]/n*2*%pi+%pi/2;
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;

// Poligono (verso orario)
function p=poligo(n,c0,r)
      a= -[0:n]/n*2*%pi+%pi/2;
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;

// Circonferenza (verso antiorario)
function p=cira(c0,r)
      n=120; a= [0:n]/n*2*%pi+%pi/2;
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;

// Circonferenza (verso orario)
function p=ciro(c0,r)
      n=120; a= -[0:n]/n*2*%pi+%pi/2;
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;

// Numerazione dei vertici di un poligono
function numerapol(pol,c0)
      [nr,nc]=size(pol);
      for i=1:nc-1
            ag=[0.03;0.05]; // aggiustamento posizione del numero
            pn=c0+((pol(:,i)-c0)-ag)*1.1;
            xstring(pn(1),pn(2),string(i));
      end
endfunction

// Poligono assegnato con il mouse
function pol=poligono_m()
// Assegnare i vertici con il tasto sinistro
// Terminare con un tasto diverso dal sinistro
      n=0;
      pol=[];
      [b,xv,yv]=xclick();
      while (b==0)|(b==3)
            n=n+1;
            pol(:,n)=[xv;yv];
            plot2d(xv,yv,-1)
            if n>1 then plot2d(pol(1,n-1:n),pol(2,n-1:n)), end
            [b,xv,yv]=xclick();
      end
      if n>1 then
            pol(:,n+1)=pol(:,1);
```

```
            plot2d(pol(1,n:n+1),pol(2,n:n+1))
      end
endfunction
// ------<funzioni>-<fine>----------------------------------------
//.
```

[Download]

## 4.12 Funzioni (Area e baricentro)

```
//:lib_area.sce
//
// Funzioni (Area e baricentro)
//
// ------<funzioni>-<inizio>--------------------------------------

// Calcolo dell'area di un triangolo
//
function area=areatriangolo(p1,p2,p3)
      area=det([p2-p1,p3-p1])/2
endfunction

// Calcolo dell'area di un poligono suddiviso in triangoli di vertice p0
//
function area=areapolp0(pol,p0)
      [nr,nc]=size(pol);
      area=0
      for i=1:nc-1
            p1=pol(:,i);
            p2=pol(:,i+1);
            area=area+areatriangolo(p0,p1,p2)
      end
endfunction

// Calcolo dell'area di un poligono suddiviso in triangoli
//
function area=areapol(pol)
      [nr,nc]=size(pol);
      p0=pol(:,1);
      area=0
      for i=1:nc-1
            p1=pol(:,i);
            p2=pol(:,i+1);
            area=area+areatriangolo(p0,p1,p2)
      end
endfunction

// Calcolo del baricentro di un poligono suddiviso in triangoli di vertice p0
//
function pg=barpolp0(pol,p0)
      [nr,nc]=size(pol);
      area=0;
      rg=[0;0];
      for i=1:nc-1
            p1=pol(:,i);
            p2=pol(:,i+1);
            pm=(p1+p2)/2;
            areai=areatriangolo(p0,p1,p2);
            area=area+areai;
            rg=rg+areai*(pm-p0)*2/3
      end
      pg=rg/area + p0
endfunction

// Calcolo del baricentro di un poligono
//
```

```
function pg=barpol(pol)
      [nr,nc]=size(pol);
      p0=pol(:,1);
      area=0;
      rg=[0;0];
      for i=1:nc-1
            p1=pol(:,i);
            p2=pol(:,i+1);
            pm=(p1+p2)/2;
            areai=areatriangolo(p0,p1,p2);
            area=area+areai;
            rg=rg+areai*(pm-p0)*2/3
      end
      pg=rg/area + p0
endfunction
// ------<funzioni>-<fine>----------------------------------------
//.
```

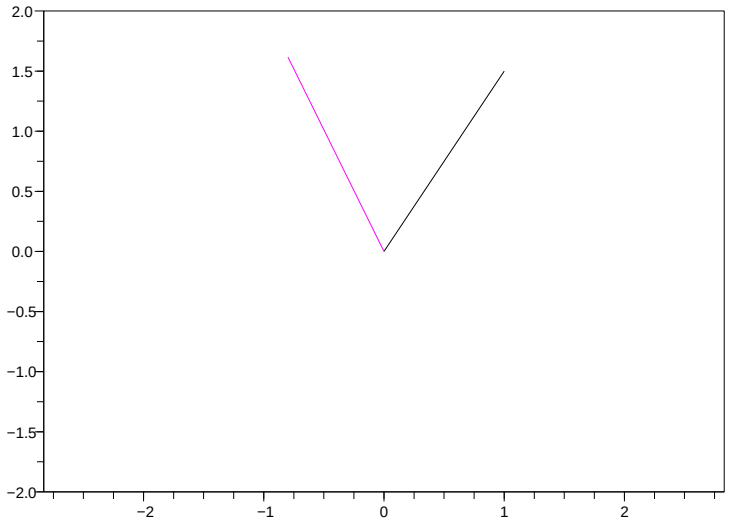
[Download]

# Capitolo 5

# Rotazioni di segmenti

Si vede come tracciare un segmento e costruirne altri attraverso delle rotazioni. Questo servirà poi a ottenere rotazioni di poligoni.

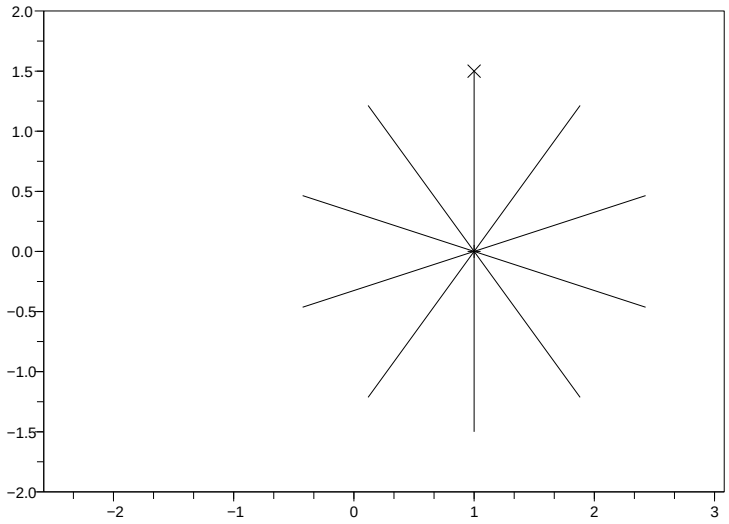
## 5.1 Rotazione di un segmento




```
//:seg1.sce
//
// Rotazione di un segmento
//
function mat=R(a)
      mat=[cos(a),-sin(a);
         sin(a), cos(a)];
endfunction;
p0=[0;0];
p1=[1;1.5];
a=%pi/3;
p1r=p0+R(a)*(p1-p0);
s=[p0,p1]; // segmento
sr=[p0,p1r]; // segmento ruotato
clf();
plot2d(0,0,1,"031"," ",[-2,-2,2,2]);
plot2d(s(1,:),s(2,:));
plot2d(sr(1,:),sr(2,:),6);
//.
```

[Download]

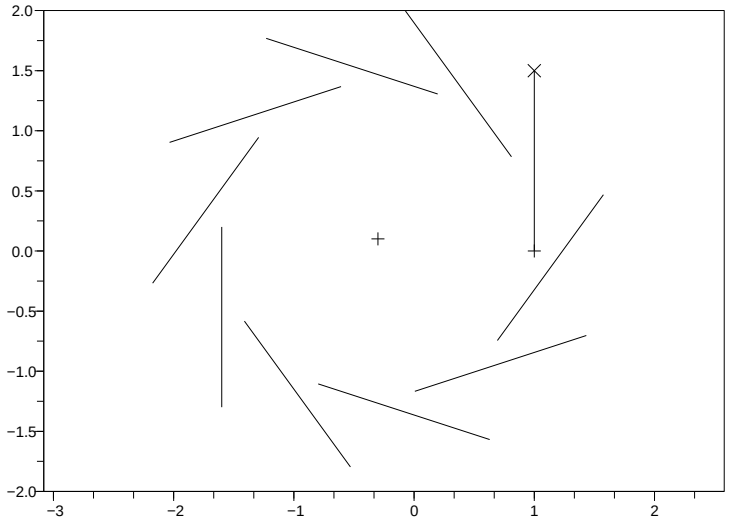
## 5.2 Rotazioni di un segmento




```
//:seg2.sce
//
// Rotazioni di un segmento
//
function mat=R(a)
      mat=[cos(a),-sin(a);
         sin(a), cos(a)];
endfunction;
//
p0=[1;0];
p1=[1;1.5];
s=[p0,p1]; // segmento
//
clf();
plot2d(0,0,1,"031"," ",[-2,-2,2,2]);
plot2d(p0(1),p0(2),-1)
plot2d(p1(1),p1(2),-2)
//
// Rotazioni del segmento
//
n=10;
a=2*%pi/n;
for i=1:n
      p1r=p0+R(a*i)*(p1-p0);
      sr=[p0,p1r];
      plot2d(sr(1,:),sr(2,:));
end
//.
```

[Download]

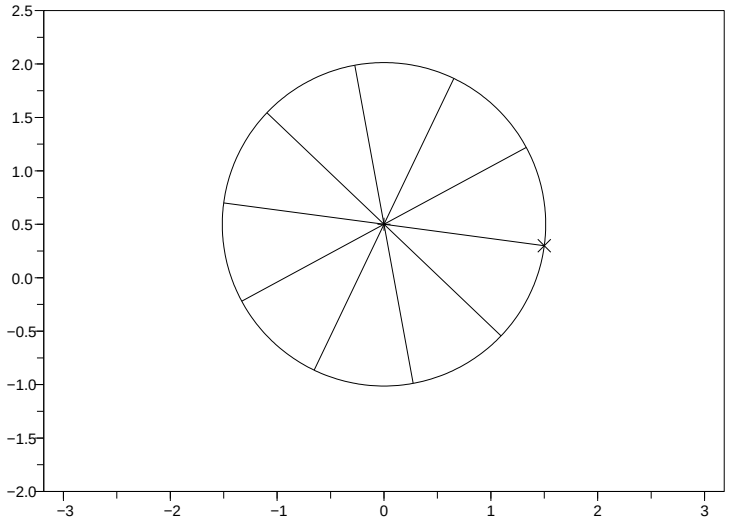
## 5.3 Rotazioni attorno a c0




```
//:seg3.sce
//
// Rotazioni attorno a c0
//
function mat=R(a)
      mat=[cos(a),-sin(a);
         sin(a), cos(a)];
endfunction;
//
c0=[-0.3;0.1];
p0=[1;0];
p1=[1;1.5];
s=[p0,p1]; // segmento
//
clf();
plot2d(0,0,1,"031"," ",[-2,-2,2,2]);
plot2d(c0(1),c0(2),-1)
plot2d(p0(1),p0(2),-1)
plot2d(p1(1),p1(2),-2)
//
// Rotazioni del segmento
//
n=10;
a=2*%pi/n;
for i=1:n
      p0r=c0+R(a*i)*(p0-c0);
      p1r=c0+R(a*i)*(p1-c0);
      sr=[p0r,p1r];
      plot2d(sr(1,:),sr(2,:));
end
//.
```

[Download]

## 5.4 Un segmento che ruota in un cerchio di centro p0

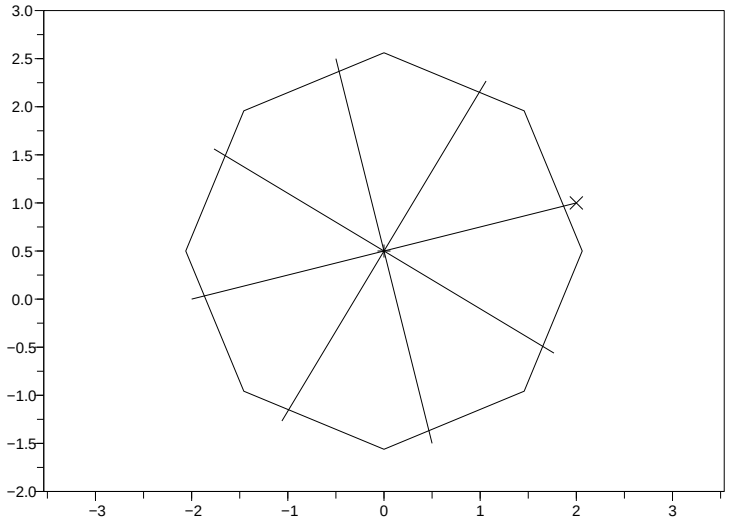



```
//:seg4.sce
//
// Un segmento che ruota in un cerchio di centro p0
//
// Circonferenza di verso antiorario
function p=cira(c0,r)
      n=360; a=%pi/2+[0:n]/n*2*%pi;
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;
// Circonferenza di verso orario
function p=ciro(c0,r)
      n=360; a=%pi/2-[0:n]/n*2*%pi;
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;
// Poligono regolare (antiorario)
function p=poliga(n,c0,r)
      a=%pi/2+[0:n]/n*2*%pi;
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;
// Poligono regolare (orario)
function p=poligo(n,c0,r)
      a=%pi/2-[0:n]/n*2*%pi;
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;
// Rotazione
function mat=R(a)
      mat=[cos(a),-sin(a);
         sin(a), cos(a)];
endfunction;
//
p0=[0;0.5];
p1=[1.5;0.3];
s=[p0,p1];
//
clf();
```

```
plot2d(0,0,1,"031"," ",[-2,-2,2,2]);
plot2d(p0(1),p0(2),-1)
plot2d(p1(1),p1(2),-2)
//
// Rotazioni del segmento
//
n=10;
a=2*%pi/n;
for i=1:n
      p1r=p0+R(a*i)*(p1-p0);
      sr=[p0,p1r];
      plot2d(sr(1,:),sr(2,:));
end
//
// Cerchio di centro p0
//
r=norm(p1-p0,2);
fig=cira(p0,r);
plot2d(fig(1,:),fig(2,:));
//.
```

[Download]

## 5.5 Un segmento che ruota e un poligono di centro p0




```
//:seg5.sce
//
// Un segmento che ruota e un poligono di centro p0
//
// Circonferenza di verso antiorario
function p=cira(c0,r)
      n=360; a=%pi/2+[0:n]/n*2*%pi;
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;
// Circonferenza di verso orario
function p=ciro(c0,r)
      n=360; a=%pi/2-[0:n]/n*2*%pi;
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;
// Poligono regolare (antiorario)
function p=poliga(n,c0,r)
      a=%pi/2+[0:n]/n*2*%pi;
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;
// Poligono regolare (orario)
function p=poligo(n,c0,r)
      a=%pi/2-[0:n]/n*2*%pi;
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;
// Rotazione
function mat=R(a)
      mat=[cos(a),-sin(a);
         sin(a), cos(a)];
endfunction;
//
p0=[0;0.5];
p1=[2;1];
s=[p0,p1];
clf();
//
```

```
plot2d(0,0,1,"031"," ",[-2,-2,2,2]);
plot2d(p0(1),p0(2),-1)
plot2d(p1(1),p1(2),-2)
//
// Rotazioni del segmento
//
n=8;
a=2*%pi/n;
for i=1:n
      p1r=p0+R(a*i)*(p1-p0);
      sr=[p0,p1r];
      plot2d(sr(1,:),sr(2,:));
end
//
// Poligono di centro p0
//
r=norm(p1-p0,2);
fig=poliga(n,p0,r)
plot2d(fig(1,:),fig(2,:));
//.
```

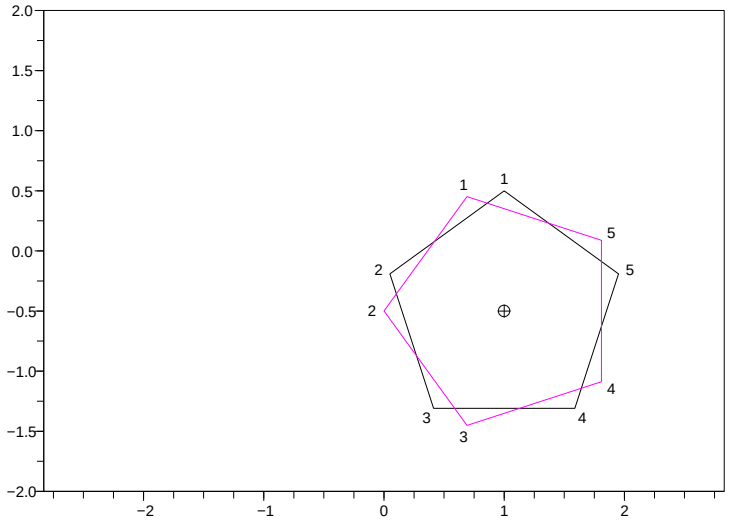
[Download]

# Capitolo 6

# Deformazioni affini di poligoni

Un poligono è traslato, ruotato, dilatato. Si mette in evidenza come si modifica l'area.

## 6.1 Rotazione di un poligono (1)

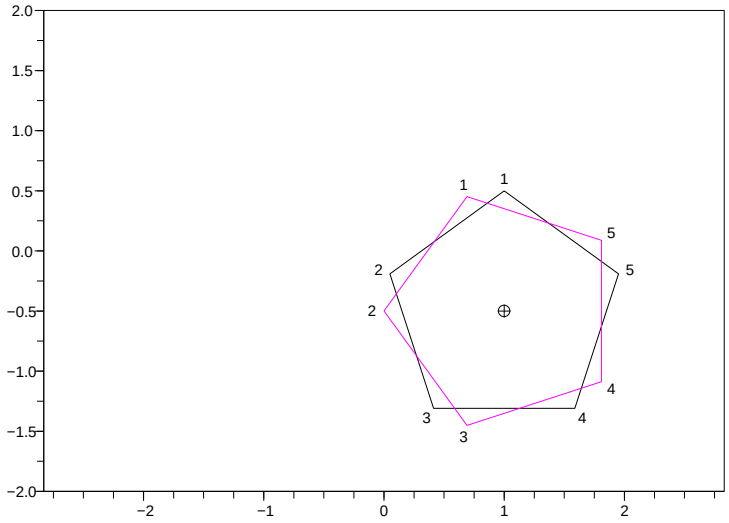



```
//:pol1.sce
//
// Rotazione di un poligono (1)
//
// ------<funzioni>-<inizio>----------------------------------------
function p=poliga(n,c0,r)
      a=[0:n]/n*2*%pi+%pi/2; // verso antiorario
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;
function numerapol(pol,c0)
      [nr,nc]=size(pol);
      for i=1:nc-1
            ag=[0.03;0.05]; // aggiustamento posizione del numero
            pn=c0+((pol(:,i)-c0)-ag)*1.1;
            xstring(pn(1),pn(2),string(i));
      end
endfunction
function mat=R(a),
       mat=[cos(a),-sin(a);
            sin(a),cos(a)];
endfunction
// ------<funzioni>-<fine>-----------------------------------------
//
n=5;
c0=[1;-0.5];
fig=poliga(n,c0,1);
//
clf();
plot2d(0,0,1,"031"," ",[-2,-2,2,2]);
plot2d(c0(1),c0(2),-3)
plot2d(fig(1,:),fig(2,:));
numerapol(fig,c0);
//
// Rotazione
//
```

```
a=%pi/10;
figr=fig;
for j=1:n+1
      pj=fig(:,j);
      pjr=c0+R(a)*(pj-c0);
      figr(:,j)=pjr;
end
plot2d(figr(1,:),figr(2,:),6);
numerapol(figr,c0);
//.
```

[Download]

## 6.2 Rotazione di un poligono (2)

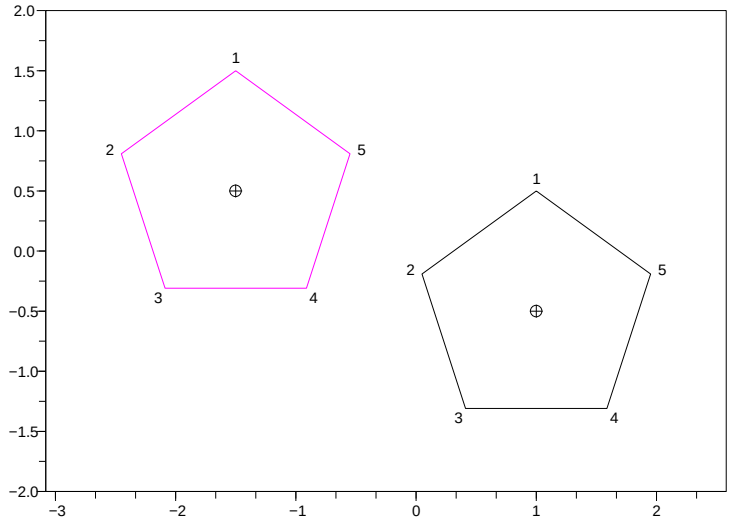



```
//:pol2.sce
//
// Rotazione di un poligono (2)
//
// ------<funzioni>-<inizio>----------------------------------------
function p=poliga(n,c0,r)
      a=[0:n]/n*2*%pi+%pi/2;
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;
function numerapol(pol,c0)
      [nr,nc]=size(pol);
      for i=1:nc-1
            ag=[0.03;0.05]; // aggiustamento posizione del numero
            pn=c0+((pol(:,i)-c0)-ag)*1.1;
            xstring(pn(1),pn(2),string(i));
      end
endfunction
function mat=R(a),
       mat=[cos(a),-sin(a);
            sin(a),cos(a)];
endfunction
function figr=ruota(fig,c0,a)
      figr=fig;
      [nr,nc]=size(fig);
      for j=1:nc
            pj=fig(:,j);
            pjr=c0+R(a)*(pj-c0);
            figr(:,j)=pjr;
      end
endfunction
// ------<funzioni>-<fine>-----------------------------------------
//
n=5;
c0=[1;-0.5];
fig=poliga(n,c0,1);
```

```
//
clf();
plot2d(0,0,1,"031"," ",[-2,-2,2,2]);
plot2d(c0(1),c0(2),-3)
plot2d(fig(1,:),fig(2,:));
numerapol(fig,c0);
//
// Rotazione
//
a=%pi/10;
figr=ruota(fig,c0,a);
plot2d(figr(1,:),figr(2,:),6);
numerapol(figr,c0);
//.
```

[Download]

## 6.3 Traslazione di un poligono

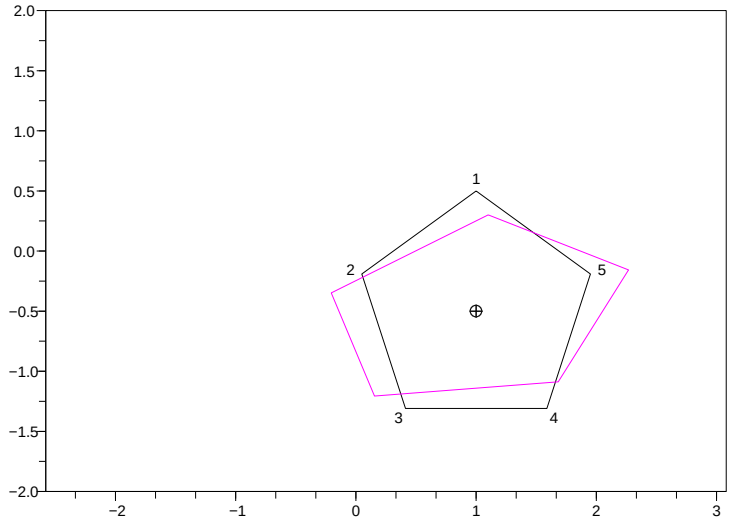



```
//:pol3.sce
//
// Traslazione di un poligono
//
// ------<funzioni>-<inizio>----------------------------------------
function p=poliga(n,c0,r)
      a=[0:n]/n*2*%pi+%pi/2;
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;
function numerapol(pol,c0)
      [nr,nc]=size(pol);
      for i=1:nc-1
            ag=[0.03;0.05]; // aggiustamento posizione del numero
            pn=c0+((pol(:,i)-c0)-ag)*1.1;
            xstring(pn(1),pn(2),string(i));
      end
endfunction
function mat=R(a),
       mat=[cos(a),-sin(a);
            sin(a),cos(a)];
endfunction
function figr=ruota(fig,c0,a)
      figr=fig;
      [nr,nc]=size(fig);
      for j=1:nc
            pj=fig(:,j);
            pjr=c0+R(a)*(pj-c0);
            figr(:,j)=pjr;
      end
endfunction
function figt=trasla(fig,u)
      figt=fig;
      [nr,nc]=size(fig);
      for j=1:nc
            pj=fig(:,j);
```

```
            pjt=pj+u;
            figt(:,j)=pjt;
      end
endfunction
// ------<funzioni>-<fine>----------------------------------------
//
n=5;
c0=[1;-0.5];
fig=poliga(n,c0,1);
//
clf();
plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
plot2d(c0(1),c0(2),-3)
plot2d(fig(1,:),fig(2,:));
numerapol(fig,c0);
//
// Traslazione
//
u0=[-2.5;1];
figt=trasla(fig,u0);
plot2d(c0(1)+u0(1),c0(2)+u0(2),-3)
plot2d(figt(1,:),figt(2,:),6);
numerapol(figt,c0+u0);
//.
```

[Download]

## 6.4 Dilatazione di un poligono

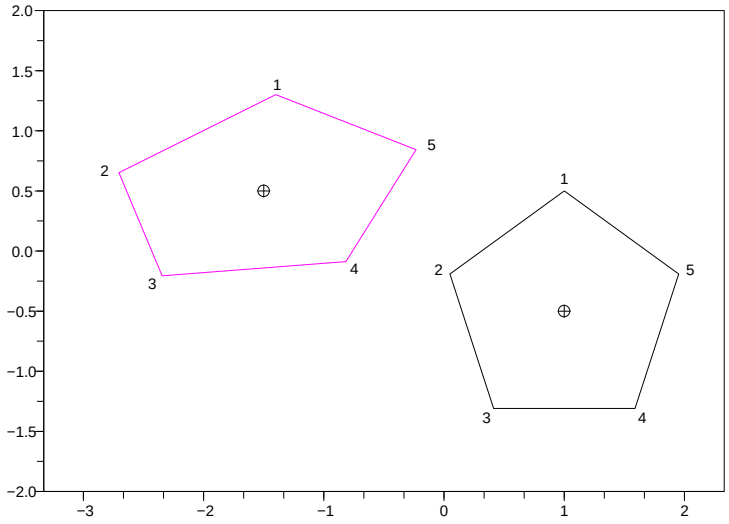



```
//:pol4.sce
//
// Dilatazione di un poligono
//
// ------<funzioni>-<inizio>----------------------------------------
function p=poliga(n,c0,r)
      a=[0:n]/n*2*%pi+%pi/2;
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;
function numerapol(pol,c0)
      [nr,nc]=size(pol);
      for i=1:nc-1
            ag=[0.03;0.05]; // aggiustamento posizione del numero
            pn=c0+((pol(:,i)-c0)-ag)*1.1;
            xstring(pn(1),pn(2),string(i));
      end
endfunction
function mat=R(a),
       mat=[cos(a),-sin(a);
            sin(a),cos(a)];
endfunction
function figr=ruota(fig,c0,a)
      figr=fig;
      [nr,nc]=size(fig);
      for j=1:nc
            pj=fig(:,j);
            pjr=c0+R(a)*(pj-c0);
            figr(:,j)=pjr;
      end
endfunction
function figt=trasla(fig,u)
      figt=fig;
      [nr,nc]=size(fig);
      for j=1:nc
            pj=fig(:,j);
```

```
            pjt=pj+u;
            figt(:,j)=pjt;
      end
endfunction
function figd=dilata(fig,c0,U)
      figd=fig;
      [nr,nc]=size(fig);
      for j=1:nc
            pj=fig(:,j);
            pjd=c0+U*(pj-c0);
            figd(:,j)=pjd;
      end
endfunction
// ------<funzioni>-<fine>----------------------------------------
//
n=5;
c0=[1;-0.5];
fig=poliga(n,c0,1);
//
clf();
plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
plot2d(c0(1),c0(2),-3)
plot2d(fig(1,:),fig(2,:));
numerapol(fig,c0);
//
// Dilatazione
//
U=[1.3 0.1;0.1 0.8]; // simmetrica e def.pos.
figd=dilata(fig,c0,U);
plot2d(c0(1),c0(2),-3)
plot2d(figd(1,:),figd(2,:),6);
//.
```

[Download]

## 6.5 Traslazione e dilatazione di un poligono

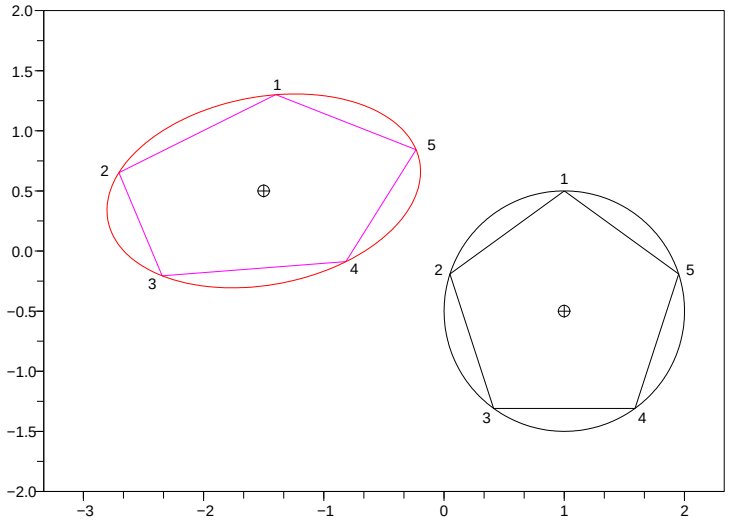



```
//:pol5.sce
//
// Traslazione e dilatazione di un poligono
//
// ------<funzioni>-<inizio>----------------------------------------
function p=poliga(n,c0,r)
      a=[0:n]/n*2*%pi+%pi/2;
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;
function numerapol(pol,c0)
      [nr,nc]=size(pol);
      for i=1:nc-1
            ag=[0.03;0.05]; // aggiustamento posizione del numero
            pn=c0+((pol(:,i)-c0)-ag)*1.1;
            xstring(pn(1),pn(2),string(i));
      end
endfunction
function mat=R(a),
       mat=[cos(a),-sin(a);
            sin(a),cos(a)];
endfunction
function figr=ruota(fig,c0,a)
      figr=fig;
      [nr,nc]=size(fig);
      for j=1:nc
            pj=fig(:,j);
            pjr=c0+R(a)*(pj-c0);
            figr(:,j)=pjr;
      end
endfunction
function figt=trasla(fig,u)
      figt=fig;
      [nr,nc]=size(fig);
      for j=1:nc
            pj=fig(:,j);
```

```
            pjt=pj+u;
            figt(:,j)=pjt;
      end
endfunction
function figd=dilata(fig,c0,U)
      figd=fig;
      [nr,nc]=size(fig);
      for j=1:nc
            pj=fig(:,j);
            pjd=c0+U*(pj-c0);
            figd(:,j)=pjd;
      end
endfunction
// ------<funzioni>-<fine>----------------------------------------
//
n=5;
c0=[1;-0.5];
fig=poliga(n,c0,1);
//
clf();
plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
plot2d(c0(1),c0(2),-3)
plot2d(fig(1,:),fig(2,:));
numerapol(fig,c0);
//
// Dilatazione e traslazione
//
u0=[-2.5;1];
U=[1.3 0.1;0.1 0.8]; // simmetrica e def.pos.
figd=trasla(dilata(fig,c0,U),u0);
plot2d(c0(1)+u0(1),c0(2)+u0(2),-3)
plot2d(figd(1,:),figd(2,:),6);
numerapol(figd,c0+u0);
//.
```

[Download]

## 6.6 Traslazione e dilatazione di un poligono e di un cerchio

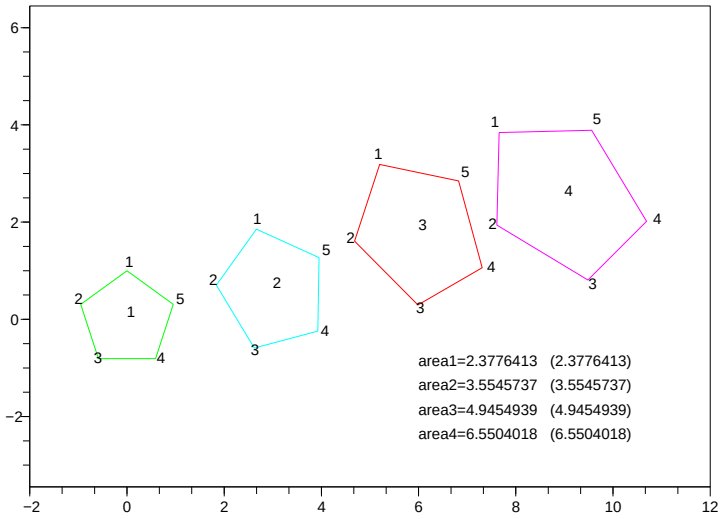



```
//:pol6.sce
//
// Traslazione e dilatazione di un poligono e di un cerchio
//
// ------<funzioni>-<inizio>----------------------------------------
function p=poliga(n,c0,r)
      a=[0:n]/n*2*%pi+%pi/2;
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;
function p=cira(c0,r)
      n=360; a= [0:n]/n*2*%pi+%pi/2;
      p=[c0(1)+r*cos(a); c0(2)+r*sin(a)];
endfunction;
function numerapol(pol,c0)
      [nr,nc]=size(pol);
      for i=1:nc-1
            ag=[0.03;0.05]; // aggiustamento posizione del numero
            pn=c0+((pol(:,i)-c0)-ag)*1.1;
            xstring(pn(1),pn(2),string(i));
      end
endfunction
function mat=R(a),
       mat=[cos(a),-sin(a);
            sin(a),cos(a)];
endfunction
function figr=ruota(fig,c0,a)
      figr=fig;
      [nr,nc]=size(fig);
      for j=1:nc
            pj=fig(:,j);
            pjr=c0+R(a)*(pj-c0);
            figr(:,j)=pjr;
      end
endfunction
function figt=trasla(fig,u)
```

```
      figt=fig;
      [nr,nc]=size(fig);
      for j=1:nc
            pj=fig(:,j);
            pjt=pj+u;
            figt(:,j)=pjt;
      end
endfunction
function figd=dilata(fig,c0,U)
      figd=fig;
      [nr,nc]=size(fig);
      for j=1:nc
            pj=fig(:,j);
            pjd=c0+U*(pj-c0);
            figd(:,j)=pjd;
      end
endfunction
// ------<funzioni>-<fine>----------------------------------------
//
c0=[1;-0.5];
n=5;
fig1=poliga(n,c0,1);
r=norm(fig1(:,1)-c0,2);
fig2=cira(c0,r);
//
clf();
plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
plot2d(c0(1),c0(2),-3)
plot2d(fig1(1,:),fig1(2,:));
numerapol(fig1,c0);
plot2d(fig2(1,:),fig2(2,:));
//
// Dilatazione e traslazione
//
u0=[-2.5;1];
U=[1.3 0.1;0.1 0.8]; // simmetrica e def.pos.
fig1d=trasla(dilata(fig1,c0,U),u0);
fig2d=trasla(dilata(fig2,c0,U),u0);
plot2d(c0(1)+u0(1),c0(2)+u0(2),-3)
plot2d(fig1d(1,:),fig1d(2,:),6);
numerapol(fig1d,c0+u0);
plot2d(fig2d(1,:),fig2d(2,:),5);
//.
```

[Download]

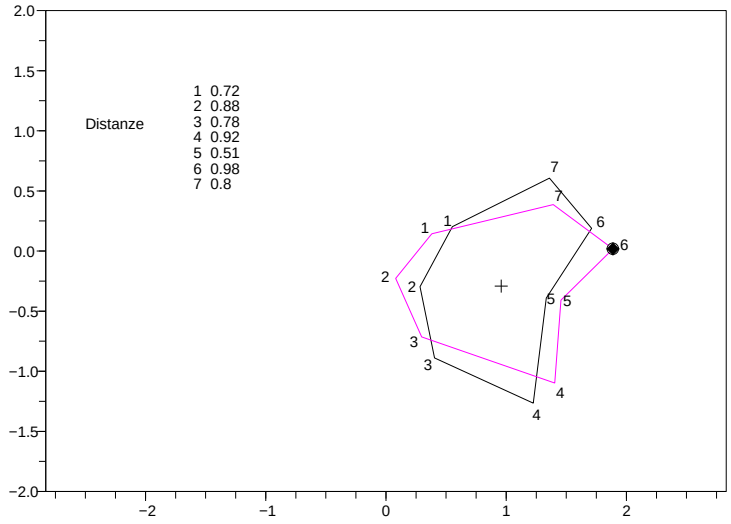
## 6.7 Area di un pentagono deformato




```
//:pol7.sce
//
// Area di un pentagono deformato
//
getf('lib_poligoni.sce')
getf('lib_area.sce')
getf('lib_deformazioni.sce')
c0=[0;0];
fig1=poliga(5,c0,1);
clf();
plot2d(0,0,1,"031"," ",[-2,-2,12,5]);
u1=[0;0]; u2=[3;0.6]; u3=[6;1.8]; u4=[9;2.5];
u=[u1,u2,u3,u4];
a=0;
epsx=0.15;
epsy=0.3;
A1=areapol(fig1);
for m=1:4
      U=[1+epsx*(m-1),0;0,1+epsy*(m-1)];
      u0=u(:,m);
      fig2=trasla(ruota(dilata(fig1,c0,U),c0,a),u0);
      plot2d(fig2(1,:),fig2(2,:),m+2);
      c2=c0+u0;
      numerapol(fig2,c2);
      xstring(c2(1),c2(2),string(m));
      a=a+%pi/12;
      area=areapol(fig2); //area figura dilatata
      aread=det(U)*A1;    //amplificazione area iniziale
      xstring(6,-0.5-m/2,"area"+string(m)+"="+string(area)+"   ("+string(aread)+")")
end;
//.
```

[Download]

## 6.8 Distanze dal centro dei vertici di un poligono dilatato

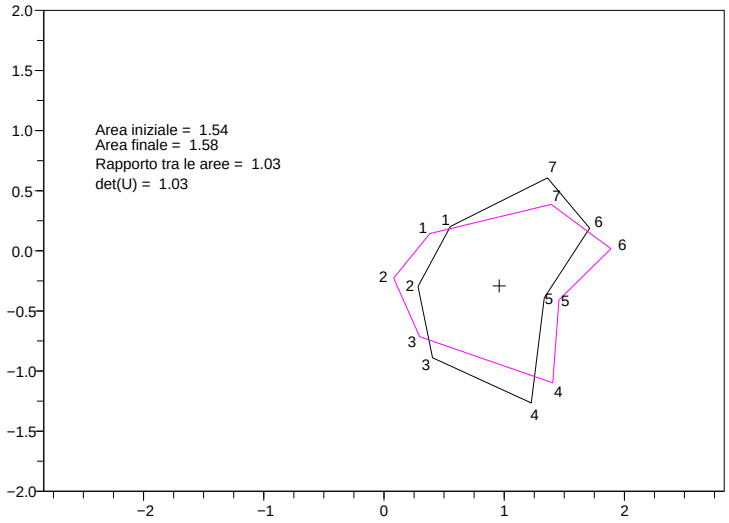



```
//:pol8.sce
//
// Distanze dal centro dei vertici di un poligono dilatato
//
// ------<funzioni>-<inizio>----------------------------------------
getf('lib_poligoni.sce');
getf('lib_area.sce');
getf('lib_deformazioni.sce');
// ------<funzioni>-<fine>-----------------------------------------
//
clf();
plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
fig=poligono_m();
clf();
plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
plot2d(fig(1,:),fig(2,:));
c0=barpol(fig);
plot2d(c0(1),c0(2),-1)
numerapol(fig,c0);
//
// Dilatazione
//
U=[1.3 -0.1;-0.1 0.8]; // simmetrica e def.pos.
figd=dilata(fig,c0,U);
plot2d(figd(1,:),figd(2,:),6);
numerapol(figd,c0);

// Calcolo delle distanze dei vertici dal centro
d=[];
[nr,nc]=size(fig); n=nc-1;
for i=1:n
      ui=figd(:,i)-c0;
      d(i)=norm(ui);
end
xstring(-2.5,1,"Distanze")
```

```
xstring(-1.6,0.5,string([1:n]')+"  "+string(round(d*100) /100))

// Numero di un vertice (k) di massima distanza
[m,k]=max(d) // (massimo di una n-pla e posizione)
plot2d(figd(1,k),figd(2,k),-3);

// (In alternativa)
// Numero di un vertice (k) di massima distanza
dmax=0;
for i=1:n
      ui=figd(:,i)-c0;
      di=norm(ui);
      if di>dmax then k=i; dmax=di; end
end
plot2d(figd(1,k),figd(2,k),-4);
//.
```

[Download]

## 6.9 Rapporto tra le aree e determinante della matrice della dilatazione




```
//:pol9.sce
//
// Rapporto tra le aree e determinante della matrice della dilatazione
//
// ------<funzioni>-<inizio>----------------------------------------
getf('lib_poligoni.sce');
getf('lib_area.sce');
getf('lib_deformazioni.sce');
// ------<funzioni>-<fine>-----------------------------------------
//
clf();
plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
fig=poligono_m();
clf();
plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
plot2d(fig(1,:),fig(2,:));
c0=barpol(fig);
plot2d(c0(1),c0(2),-1)
numerapol(fig,c0);
//
// Dilatazione
//
U=[1.3 -0.1;-0.1 0.8]; // simmetrica e def.pos.
figd=dilata(fig,c0,U);
plot2d(figd(1,:),figd(2,:),6);
numerapol(figd,c0);

// Area
area=areapol(fig);
aread=areapol(figd);
rapporto=aread/area;
xstring(-2.4,0.5, ...
["Area iniziale = ", string(round(area*100)/100); ...
 "Area finale = ", string(round(aread*100)/100); ...
```

```
 "Rapporto tra le aree = ", string(round(rapporto*100)/100); ...
 "det(U) = ", string(round(det(U)*100)/100) ...
])
//.
```

[Download]

## 6.10 Funzioni (Deformazioni affini)

```
//:lib_deformazioni.sce
//
// Funzioni (Deformazioni affini)
//
// ------<funzioni>-<inizio>----------------------------------------

// Matrice della rotazione
function mat=R(a),
       mat=[cos(a),-sin(a);
            sin(a),cos(a)];
endfunction

// Rotazione
function figr=ruota(fig,c0,a)
      figr=fig;
      [nr,nc]=size(fig);
      for j=1:nc
            pj=fig(:,j);
            pjr=c0+R(a)*(pj-c0);
            figr(:,j)=pjr;
      end
endfunction

// Traslazione
function figt=trasla(fig,u)
      figt=fig;
      [nr,nc]=size(fig);
      for j=1:nc
            pj=fig(:,j);
            pjt=pj+u;
            figt(:,j)=pjt;
      end
endfunction

// Dilatazione
function figd=dilata(fig,c0,U)
      figd=fig;
      [nr,nc]=size(fig);
      for j=1:nc
            pj=fig(:,j);
            pjd=c0+U*(pj-c0);
            figd(:,j)=pjd;
      end
endfunction
// ------<funzioni>-<fine>-----------------------------------------
//.
```

[Download]

# Capitolo 7

# Intersezioni e composizioni di poligoni

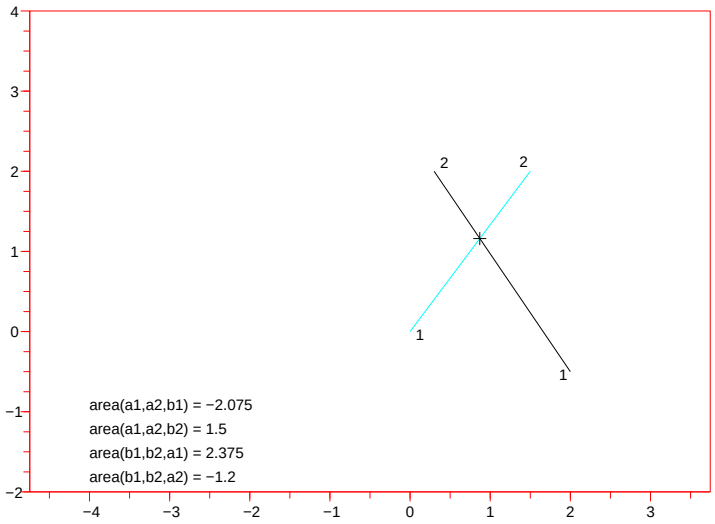
Da una coppia di poligoni si può costruire un nuovo poligono attraverso le operazioni di unione e di intersezione. A questo fine è utile saper calcolare l'intersezione di due segmenti.

## 7.1 Funzioni (Intersezione di segmenti)

```
//:lib_intersezioni.sce
//
// Funzioni (Intersezione di segmenti)
//
// ------<funzioni>-<inizio>----------------------------------------
//
// Calcolo del punto di intersezione
// (solo per segmenti intersecanti)
function p=intersezione(sa,sb)
      a1=sa(:,1); a2=sa(:,2);
      b1=sb(:,1); b2=sb(:,2);
      xa1=a1(1); ya1=a1(2);
      xa2=a2(1); ya2=a2(2);
      xb1=b1(1); yb1=b1(2);
      xb2=b2(1); yb2=b2(2);
      nax=ya2-ya1; nay=-(xa2-xa1); nbx=yb2-yb1; nby=-(xb2-xb1);
      N=[nax nay;nbx nby]; tn=[0;-((xa1-xb1)*nbx+(ya1-yb1)*nby)];
      p=inv(N)*tn + a1;
endfunction
//
// Aggiunge ad un poligono un nuovo vertice p0 dopo il vertice n0
function npol=aggiungivertice(pol,p0,n0)
      [nr,nc]=size(pol);
      npol=[pol(:,1:n0),p0,pol(:,n0+1:nc)]
endfunction
// ------<funzioni>-<fine>-----------------------------------------
//.
```

[Download]

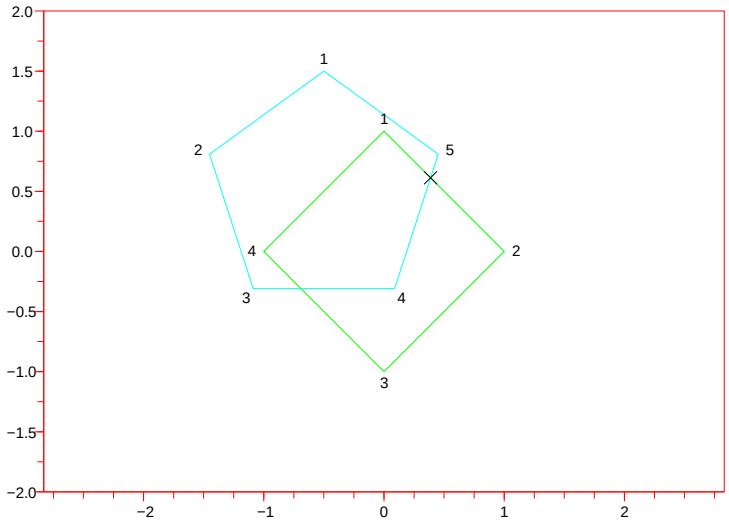
## 7.2 Intersezione di due segmenti




```
//:ints0.sce
//
// Intersezione di due segmenti
//
getf('lib_poligoni.sce');
getf('lib_poligonali.sce');
getf('lib_area.sce');
getf('lib_intersezioni.sce');
//
a1=[2;-0.5]; a2=[0.3;2];
b1=[0;0]; b2=[1.5;2];
//
sa=[a1,a2];
sb=[b1,b2];
//
clf();
xset("color",5)
plot2d(0,0,0,"031"," ",[-4,-2,3,4]);
//
plot2d(sa(1,:),sa(2,:))
numera(sa)
//
plot2d(sb(1,:),sb(2,:),4)
numera(sb)
//
xstring(-4,-1.0,"area(a1,a2,b1) = "+string(areatriangolo(a1,a2,b1)))
xstring(-4,-1.3,"area(a1,a2,b2) = "+string(areatriangolo(a1,a2,b2)))
xstring(-4,-1.6,"area(b1,b2,a1) = "+string(areatriangolo(b1,b2,a1)))
xstring(-4,-1.9,"area(b1,b2,a2) = "+string(areatriangolo(b1,b2,a2)))
//
p0=intersezione(sa,sb);
plot2d(p0(1),p0(2),-1)
//.
```

[Download]

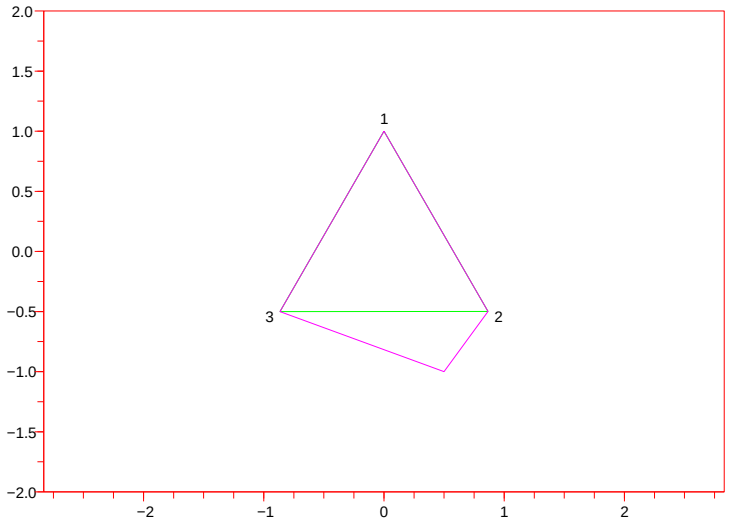
## 7.3 Due poligoni con dei lati che si intersecano




```
//:intp0.sce
//
// Due poligoni con dei lati che si intersecano
//
getf('lib_poligoni.sce');
getf('lib_poligonali.sce');
getf('lib_area.sce');
getf('lib_intersezioni.sce');
//
n1=4;c1=[0;0];
pol1=poligo(n1,c1,1);
n2=5;c2=[-0.5;0.5];
pol2=poliga(n2,c2,1);
//
clf();
xset("color",5)
plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
plot2d(pol1(1,:),pol1(2,:),3)
plot2d(pol2(1,:),pol2(2,:),4)
numerapol(pol1,c1)
numerapol(pol2,c2)
//
// Calcolo dell'intersezione di due lati
//
sa=[pol1(:,1),pol1(:,2)];
sb=[pol2(:,4),pol2(:,5)];
p1=intersezione(sa,sb);
plot2d(p1(1),p1(2),-2)
//.
```

[Download]

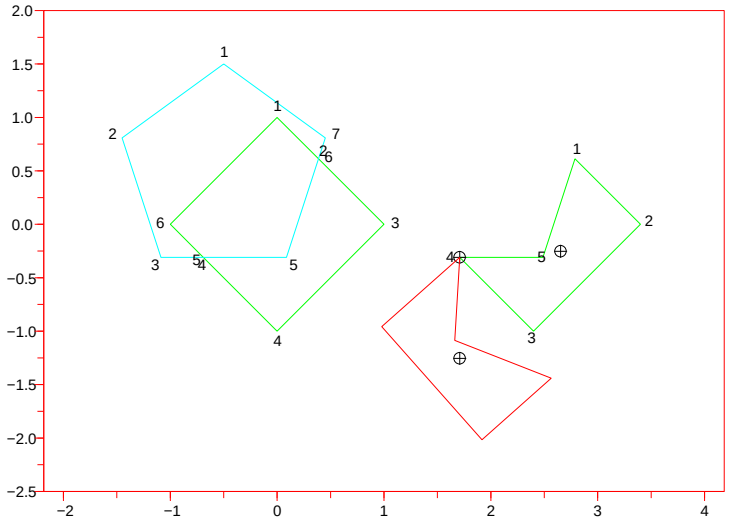
## 7.4 Aggiunta di un vertice ad un poligono




```
//:insv0.sce
//
// Aggiunta di un vertice ad un poligono
//
getf('lib_poligoni.sce');
getf('lib_poligonali.sce');
getf('lib_area.sce');
getf('lib_intersezioni.sce');
//
n1=3;c1=[0;0];
pol1=poligo(n1,c1,1);
//
clf();
xset("color",5)
plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
plot2d(pol1(1,:),pol1(2,:),3)
numerapol(pol1,c1)
//
// Aggiunge un vertice dopo il vertice n0
n0=2;
p0=[0.5;-1];
npol1=aggiungivertice(pol1,p0,n0);
plot2d(npol1(1,:),npol1(2,:),6)
//.
```

[Download]

## 7.5 Composizione di due poligoni




```
//:comp0.sce
//
// Composizione di due poligoni
//
getf('lib_poligoni.sce');
getf('lib_poligonali.sce');
getf('lib_area.sce');
getf('lib_intersezioni.sce');
//
n1=4;c1=[0;0];
pol1=poligo(n1,c1,1);
n2=5;c2=[-0.5;0.5];
pol2=poliga(n2,c2,1);
//
clf();
xset("color",5)
plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
plot2d(pol1(1,:),pol1(2,:),3)
plot2d(pol2(1,:),pol2(2,:),4)
numerapol(pol1,c1)
numerapol(pol2,c2)
npol1=pol1;
npol2=pol2;
//
// Calcolo dell'intersezione di due lati
//
i=1;sa=[npol1(:,i),npol1(:,i+1)];
j=4;sb=[npol2(:,j),npol2(:,j+1)];
p1=intersezione(sa,sb);
npol1=aggiungivertice(npol1,p1,i);
npol2=aggiungivertice(npol2,p1,j);
plot2d(p1(1),p1(2),-2)
// Nuova numerazione
clf();
xset("color",5)
```

```
plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
plot2d(npol1(1,:),npol1(2,:),3)
plot2d(npol2(1,:),npol2(2,:),4)
numerapol(npol1,c1)
numerapol(npol2,c2)
//
// Calcolo dell'intersezione di altri due lati
//
i=4;sa=[npol1(:,i),npol1(:,i+1)];
j=3;sb=[npol2(:,j),npol2(:,j+1)];
p2=intersezione(sa,sb);
npol1=aggiungivertice(npol1,p2,i);
npol2=aggiungivertice(npol2,p2,j);
plot2d(p2(1),p2(2),-2)
// Nuova numerazione
clf();
xset("color",5)
plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
plot2d(npol1(1,:),npol1(2,:),3)
plot2d(npol2(1,:),npol2(2,:),4)
numerapol(npol1,c1)
numerapol(npol2,c2)
//
// Costruzione di un nuovo poligono
//
pol=[npol1(:,2:4),npol2(:,4:6)];
u0=[2.4;0];
polt=trasla(pol,u0);
plot2d(polt(1,:),polt(2,:),3)
//
// Calcolo del baricentro
//
pg=barpol(polt);
plot2d(pg(1),pg(2),-3);
numerapol(polt,pg)
//
// Il nuovo poligono viene appeso ad un vertice
//
fig=polt;
c0=trasla(p2,u0);
vg=pg-c0;
alfa=-(atan(vg(2),vg(1))+%pi/2);
figr=ruota(fig,c0,alfa);
plot2d(c0(1),c0(2),-3);
plot2d(figr(1,:),figr(2,:),5);
pgr=barpolp0(figr,c0);
plot2d(pgr(1),pgr(2),-3);
//.
```

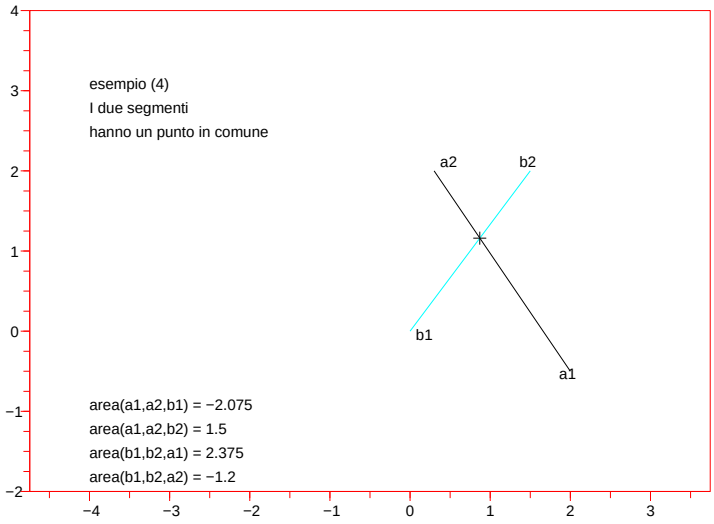
[Download]

# Capitolo 8

# Come generare, modificare, ritagliare poligoni

Si vedono qui maggiori dettagli sul test di intersezione tra segmenti. Sono inoltre descritti algoritmi per individuare i punti di intersezione del bordo di due poligoni, per costruire l'unione o l'intersezione, per ottenere un poligono convesso da uno che non lo è (convessificazione).

## 8.1 Intersezione di due segmenti



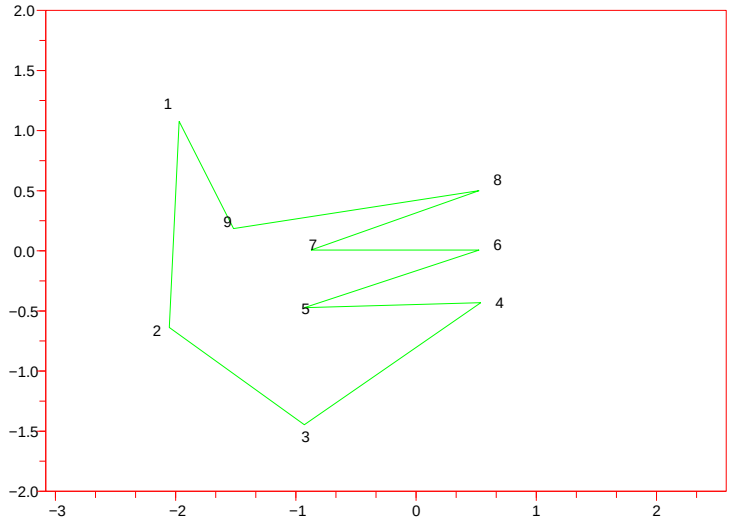

```
//:ints.sce
//
// Intersezione di due segmenti
//
getf('lib_poligoni.sce');
getf('lib_area.sce');
getf('lib_intersezioni-e.sce');
//
// ------<funzioni>-<inizio>----------------------------------------
//
// Numerazione, abbastanza ben posizionata, dei vertici di una poligonale
// con un carattere identificatore prima del numero
//
function numeraid(sp,id)
      [nr,nc]=size(sp);
      ag=[0.03;0.05];
      for i=1:nc
            vpre=[0;0]; vpost=[0;0];
            if i>1  then vpre=sp(:,i)-sp(:,i-1), end
            if i<nc then vpost=sp(:,i+1)-sp(:,i), end
            vsec=(vpre+vpost)/norm(vpre+vpost);
            vort=[-vsec(2);vsec(1)];
            vest=-vsec/2+vpre;
            if (vort' * vest > 0) then vpos=vort, else vpos=-vort, end
            if ((i==1) | (i==nc)) & (norm(sp(:,nc)-sp(:,1)) < 3*norm(ag)) & (nc>3) then
                  vest=sp(:,i)-(sp(:,2)+sp(:,nc-1))/2
                  vpos=vest/norm(vest)+1.5*(vpost-vpre)/norm(vpost-vpre)
            end
            pos=sp(:,i)+vpos*0.13-ag;
            xstring(pos(1),pos(2),id+string(i))
      end;
endfunction
// ------<funzioni>-<fine>-----------------------------------------
//
nesempi=4;
```

```
for esempio=1:nesempi
      select esempio
      case 01 then titolo="esempio ("+string(esempio)+")"; // nessun punto in comune
            a1=[2;-0.5]; a2=[0.3;2];
            b1=[-1;1]; b2=[0.5;3];
      case 02 then titolo="esempio ("+string(esempio)+")"; // nessun punto in comune
            a1=[2;-0.5]; a2=[0.3;2];
            b1=[-2.2;-0.2]; b2=[-0.7;1.8];
      case 03 then titolo="esempio ("+string(esempio)+")"; // intersecanti
            a1=[2;-0.5]; a2=[0.3;2];
            b1=[0;0]; b2=[0.5;3];
      case 04 then titolo="esempio ("+string(esempio)+")"; // intersecanti
            a1=[2;-0.5]; a2=[0.3;2];
            b1=[0;0]; b2=[1.5;2];
      case 05 then titolo="esempio ("+string(esempio)+")"; // paralleli
            a1=[2;-0.5]; a2=[0.3;2];
            b1=a1+[1;0]; b2=a2+[1;0];
      case 06 then titolo="esempio ("+string(esempio)+")"; // nessun punto in comune
            a1=[2;-0.5]; a2=[0.3;2];
            b1=[0;0]+[1.5;1]; b2=[1.5;2]+[1;1];
      case 07 then titolo="esempio ("+string(esempio)+")"; // intersecanti
            a1=[2;-0.5]; a2=[0.3;2];
            b2=a1+(a2-a1)*0.5; b1=[2;3];
      case 08 then titolo="esempio ("+string(esempio)+")"; // intersecanti adiacenti
            a1=[2;-0.5]; a2=[0.3;2];
            b1=a2; b2=[0.3;-1];
      case 09 then titolo="esempio ("+string(esempio)+")"; // tangenti adiacenti
            a1=[2;-0.5]; a2=[0.3;2];
            b1=a1-(a2-a1)*0.4; b2=a1;
      case 10 then titolo="esempio ("+string(esempio)+")"; // tangenti
            a1=[2;-0.5]; a2=[0.3;2];
            b1=a1; b2=a1+(a2-a1)*0.4;
      case 11 then titolo="esempio ("+string(esempio)+")"; // tangenti
            a1=[2;-0.5]; a2=[0.3;2];
            b1=a2-(a2-a1)*0.2; b2=a2-(a2-a1)*0.7;
      case 12 then titolo="esempio ("+string(esempio)+")"; // tangenti
            a1=[2;-0.5]; a2=[0.3;2];
            b1=a1+(a2-a1)*0.6; b2=a2+(a2-a1)*0.4;
      case 13 then titolo="esempio ("+string(esempio)+")"; // nessun punto in comune
            a1=[2;-0.5]; a2=[0.3;2];
            b1=a2+(a2-a1)*0.2; b2=a2+(a2-a1)*0.6;
      end
      //
      sa=[a1,a2];
      sb=[b1,b2];
      //
      clf();
      xset("color",5)
      plot2d(0,0,0,"031"," ",[-4,-2,3,4]);
      //
      plot2d(sa(1,:),sa(2,:))
      numeraid(sa,"a")
      //
      plot2d(sb(1,:),sb(2,:),4)
      numeraid(sb,"b")
      //
      xstring(-4,-1.0,"area(a1,a2,b1) = "+string(areatriangolo(a1,a2,b1)))
      xstring(-4,-1.3,"area(a1,a2,b2) = "+string(areatriangolo(a1,a2,b2)))
```

```
      xstring(-4,-1.6,"area(b1,b2,a1) = "+string(areatriangolo(b1,b2,a1)))
      xstring(-4,-1.9,"area(b1,b2,a2) = "+string(areatriangolo(b1,b2,a2)))
      //
      xstring(-4,3,titolo)
      xstring(-4,2.7,"I due segmenti")
      if intersecanti(sa,sb) then
            xstring(-4,2.4,"hanno un punto in comune")
            p0=intersezione(sa,sb);
            plot2d(p0(1),p0(2),-1)
      elseif tangenti(sa,sb) then
            xstring(-4,2.4,"sono tangenti")
      else
            xstring(-4,2.4,"non hanno alcun punto in comune")
      end
      if esempio<nesempi then
            xstring(-4,1.8,"Clic per continuare")
            xclick();
      end
      //
end
//.
```

[Download]

## 8.2 Come costruire o modificare un poligono utilizzando il mouse

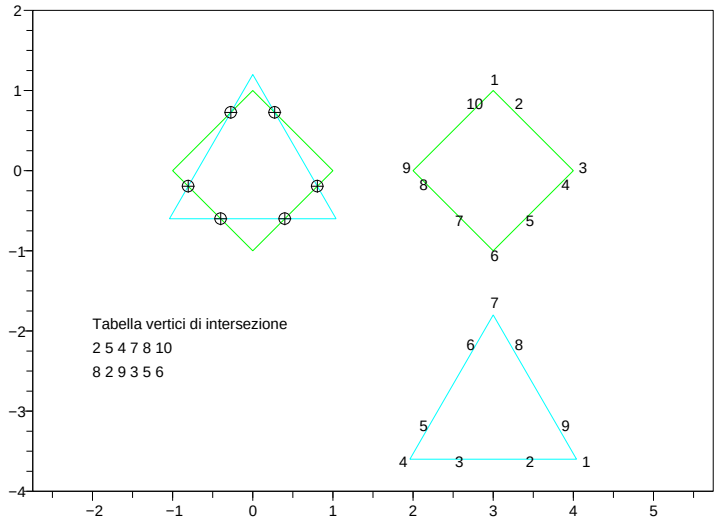



```
//:disegno.sce
//
// Come costruire o modificare un poligono utilizzando il mouse
//
getf('lib_poligoni.sce');
getf('lib_area.sce');
getf('lib_intersezioni-e.sce');
getf('lib_riordina-vertici.sce');
getf('lib_sposta-vertici.sce');
//
// Generazione di un poligono regolare
//
n=4
pol=poligo(4,[0;0],1.2);
//
xset("color",5)
plot2d(0,0,1,"031","",[-2,-2,2,2]);
plot2d(pol(1,:),pol(2,:),3)
numerapol(pol,barpol(pol))
//
// Generazione di un poligono con il mouse
//
pol=dispol_m();
//
// Spostamento dei vertici di un poligono con il mouse
//
pol=spostavertici_m(pol);
//
// Aggiunta di un nuovo vertice con il mouse
//
pol=aggiungivertici_m(pol);
//
// La matrice pu essere scritta su un file (per essere poi riletta)
//
```

```
fprintfMat('poligono.dat',pol)
pol1=fscanfMat('poligono.dat');
//
// La matrice pu anche essere visualizzata e copiata qui:
//
pol1=[
  -1.9706512  -2.0529135  -0.9286632   0.5383462  -0.9286632 ..
   0.5246358  -0.8738218   0.5246358  -1.5182091  -1.9706512
   1.0762639  -0.6375321  -1.4464439  -0.4318766  -0.4730077 ..
   0.0068552   0.0068552   0.5004284   0.1850900   1.0762639]
//
scf(1);
clf(1);
xset("color",5)
plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
plot2d(pol1(1,:),pol1(2,:),3)
numerapol(pol1,barpol(pol1))
//.
```

[Download]

## 8.3 Visualizzazione del calcolo automatico dei punti di intersezione dei lati



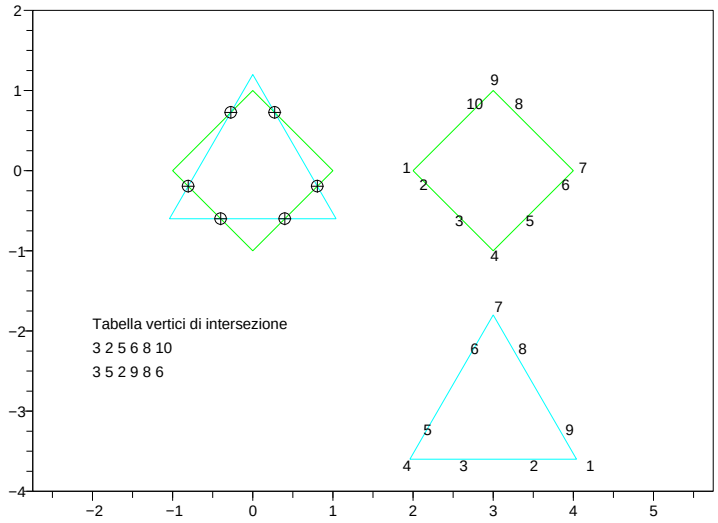

```
//:insva.sce
//
// Visualizzazione del calcolo automatico dei punti di intersezione dei lati
//
getf('lib_poligoni.sce');
getf('lib_area.sce');
getf('lib_deformazioni.sce');
getf('lib_intersezioni-e.sce');
getf('lib_riordina-vertici.sce');
getf('lib_sposta-vertici.sce');
//
// ------<funzioni>-<inizio>----------------------------------------
//
// Test di appartenza di un vertice ad un poligono
// Restituisce il numero del vertice, oppure 0
function n=vertice(pol,p0)
      eps=10^(-6);
      [nr,nc]=size(pol);
      i=1;n=0;
      while (i<=nc)& (n==0) do
            if norm(pol(:,i)-p0)<eps then n=i;
            end
            i=i+1;
      end
endfunction
// ------<funzioni>-<fine>------------------------------------------
//
n1=4;c1=[0;0];
pol1=poligo(n1,c1,1);
n2=3;c2=[-0.5;0.5];
u2=[0.5;-0.5];
pol2=trasla(poligo(n2,c2,1.2),u2); c2=c2+u2;
pol2=riordina(pol2,2);
//
```

```
clf();
xset("color",5)
plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
plot2d(pol1(1,:),pol1(2,:),3)
plot2d(pol2(1,:),pol2(2,:),4)
numerapol(pol1,c1)
numerapol(pol2,c2)
xclick();
//
npol1=pol1;
npol2=pol2;
//
lines(0);
n1=length(pol1(1,:))-1;
n2=length(pol2(1,:))-1;
nn1=n1;
nn2=n2;
insv1=[1:nn1]*0;
insv2=[1:nn2]*0;
i=1;ninc=0;
// Calcola le intersezioni dei lati, le numera, le aggiunge come vertici
while i<=nn1 do
      a1=npol1(:,i);a2=npol1(:,i+1);sa=[a1,a2];
      j=1;
      while (j<=nn2) do
            clf();
            xset("color",5)
            plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
            b1=npol2(:,j);b2=npol2(:,j+1);sb=[b1,b2];
            if  intersecanti(sa,sb) then
                  p0=intersezione(sa,sb);
                  vea=vertice(sa,p0);veb=vertice(sb,p0);
                  nonvea=(vea==0);nonveb=(veb==0);
                  vint=(nonvea|nonveb);
                  if vint then ninc=ninc+1
                  end
                  if nonvea then
                        npol1=aggiungivertice(npol1,p0,i);
                        nn1=nn1+1;
                        insv1=aggiungivertice(insv1,ninc,i);insv1
                        a1=npol1(:,i);a2=npol1(:,i+1);sa=[a1,a2];
                        xstring(1.5,1.7,"intersezione: "+string(i)+"--"+string(j))
                        plot2d(p0(1),p0(2),-3)
                  elseif vint then
                        insv1(i+vea-1)=ninc
                  end
                  if nonveb then
                        npol2=aggiungivertice(npol2,p0,j);
                        nn2=nn2+1;
                        insv2=aggiungivertice(insv2,ninc,j);insv2
                        xstring(1.5,1.7,"intersezione: "+string(i)+"--"+string(j))
                        plot2d(p0(1),p0(2),-3)
                  elseif vint then
                        insv2(j+veb-1)=ninc
                  end
            end
            xstring(1.5,1.3,"n1="+string(nn1)+", n2="+string(nn2))
            xstring(1.5,1.0,"i="+string(i)+", j="+string(j))
```

```
            plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
            plot2d(npol2(1,:),npol2(2,:),4);plot2d(sb(1,:),sb(2,:),6)
            plot2d(npol1(1,:),npol1(2,:),3);plot2d(sa(1,:),sa(2,:),5)
            xclick();
            j=j+1;
      end
      i=i+1;
end
// Crea una tabella di corrispondenza dei numeri dei vertici di intersezione
inc1=insv1;n=0;
for i=1:nn1
      ni=insv1(i);
      if ni>0 then inc1(ni)=i;n=n+1; end
end
inc2=insv2;n=0;
for i=1:nn2
      ni=insv2(i);
      if ni>0 then inc2(ni)=i;n=n+1; end
end
inc=[inc1(1:n);inc2(1:n)];
//
clf();
plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
plot2d(npol1(1,:),npol1(2,:),3);
plot2d(npol2(1,:),npol2(2,:),4);
for i=1:length(inc(1,:))
      plot2d(npol1(1,inc(1,i)),npol1(2,inc(1,i)),-3)
end
npol1t=trasla(npol1,[3;0]);
plot2d(npol1t(1,:),npol1t(2,:),3)
numerapol(npol1t,c1+[3;0])
npol2t=trasla(npol2,[3;-3]);
plot2d(npol2t(1,:),npol2t(2,:),4)
numerapol(npol2t,c2+[3;-3])
xstring(-2,-2.0,"Tabella vertici di intersezione")
xstring(-2,-2.3,string(inc(1,:)))
xstring(-2,-2.6,string(inc(2,:)))
//.
```

[Download]

## 8.4 Calcolo delle intersezioni dei lati e aggiunta automatica di nuovi vertici

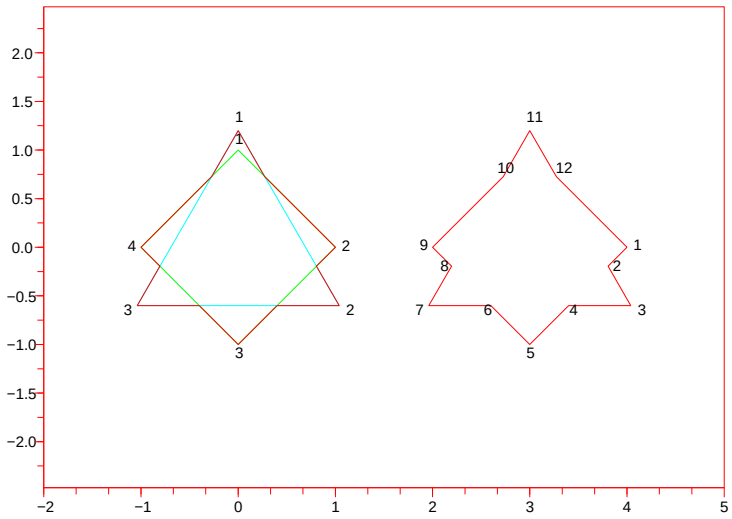



```
//:intrp.sce
//
// Intersezione dei lati di due poligoni e aggiunta automatica di nuovi vertici
//
getf('lib_poligoni.sce');
getf('lib_area.sce');
getf('lib_deformazioni.sce');
getf('lib_intersezioni-e.sce');
getf('lib_riordina-vertici.sce');
getf('lib_sposta-vertici.sce');
getf('lib_incroci.sce');
//
n1=4;c1=[0;0];
pol1=poliga(n1,c1,1);
n2=3;c2=[-0.5;0.5];
u2=[0.5;-0.5];
pol2=trasla(poligo(n2,c2,1.2),u2);
pol2=riordina(pol2,2);
pol1=riordina(pol1,2);
//
clf();
xset("color",5)
plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
plot2d(pol1(1,:),pol1(2,:),3)
plot2d(pol2(1,:),pol2(2,:),4)
numerapol(pol1,c1)
numerapol(pol2,c2)
//
[npol1,npol2,inc]=incroci(pol1,pol2);
//
xclick();
clf();
plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
plot2d(npol1(1,:),npol1(2,:),3);
```

```
plot2d(npol2(1,:),npol2(2,:),4);
for i=1:length(inc(1,:))
      plot2d(npol1(1,inc(1,i)),npol1(2,inc(1,i)),-3)
end
npol1t=trasla(npol1,[3;0]);
plot2d(npol1t(1,:),npol1t(2,:),3)
numerapol(npol1t,c1+[3;0])
npol2t=trasla(npol2,[3;-3]);
plot2d(npol2t(1,:),npol2t(2,:),4)
numerapol(npol2t,c2+[3;-3])
xstring(-2,-2.0,"Tabella vertici di intersezione")
xstring(-2,-2.3,string(inc(1,:)))
xstring(-2,-2.6,string(inc(2,:)))
//.
```

[Download]

## 8.5 Generazione di un percorso chiuso sul bordo di due poligoni



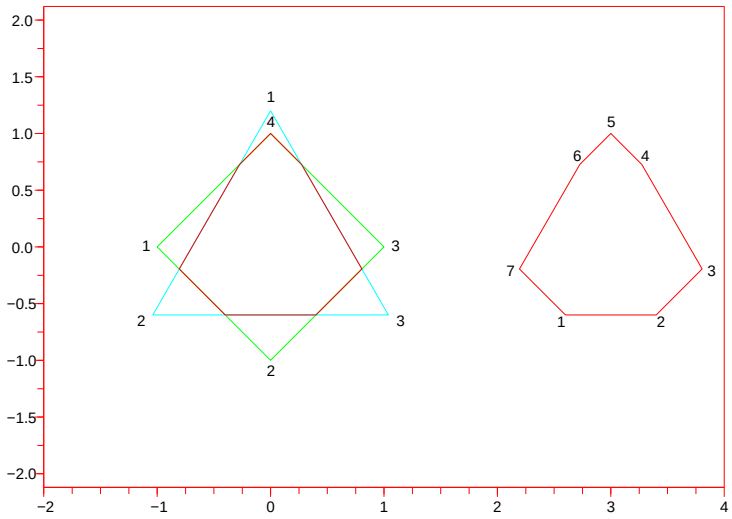

```
//:compc0.sce
//
// Generazione di un percorso chiuso sul bordo di due poligoni
//
getf('lib_poligoni.sce');
getf('lib_area.sce');
getf('lib_deformazioni.sce');
getf('lib_intersezioni-e.sce');
getf('lib_riordina-vertici.sce');
getf('lib_sposta-vertici.sce');
getf('lib_incroci.sce');
//
n1=4;c1=[0;0];
pol1=poligo(n1,c1,1);
n2=3;c2=[-0.5;0.5];
u2=[0.5;-0.5];
pol2=trasla(poligo(n2,c2,1.2),u2); c2=c2+u2;
//
clf();
xset("color",5)
plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
plot2d(pol1(1,:),pol1(2,:),3)
plot2d(pol2(1,:),pol2(2,:),4)
numerapol(pol1,c1)
numerapol(pol2,c2)
//
[npol1,npol2,inc]=incroci(pol1,pol2);
//
// Percorso chiuso a partire da un vertice n0
n0=3;
n=length(inc(1,:));
polC=[];
nn1=length(npol1(1,:))-1;
nn2=length(npol2(1,:))-1;
```

```
//
n0=1+pmodulo(n0-1,nn1);
//
// Ordina le coppie di vertici di intersezione
// i denota un incrocio da cui si prosegue sul poligono 1
// j denota un incrocio da cui si prosegue sul poligono 2
incor=[];
passi=[];
[nvi,i]=min(pmodulo(inc(1,:)-n0,nn1));
vip=inc(:,i);//primo incrocio
incor=[incor,vip];
passi=[passi,nvi];
inc2=[inc(2,1:i-1),inc(2,i+1:n)];
[nvi,i]=max(pmodulo(inc(1,:)-n0,nn1));
viu=inc(:,i);//ultimo incrocio
k=1;vi=[];vj=[];
while ~(vj==viu)&~((vj==vip)&(vip(1)==n0)) do
      //
      [nvj,j]=min(pmodulo(inc2-incor(2,k),nn2));
      jx=find(inc(2,:)==inc2(j));
      vj=inc(:,jx);
      incor=[incor,inc(:,jx)];k=k+1;
      passi=[passi,nvj];
      inc1=[inc(1,1:jx-1),inc(1,jx+1:n)];
      //
      [nvi,i]=min(pmodulo(inc1-incor(1,k),nn1));
      ix=find(inc(1,:)==inc1(i));
      vi=inc(:,ix);
      incor=[incor,inc(:,ix)];k=k+1;
      passi=[passi,nvi];
      inc2=[inc(2,1:ix-1),inc(2,ix+1:n)];
end;
nx=k;
incor=incor(:,1:nx-1);
passi=[passi(:,1:nx-1),pmodulo(n0-incor(1,nx-1),nn1)];
// Costruisce il percorso partendo da n0
polC=[npol1(:,n0)];
ix=n0;
// proseguendo a zig zag
for k=1:2:nx-1
      for i=1:passi(k)
            polC=[polC,npol1(:,1+pmodulo(ix+i-1,nn1))];
      end
      jx=1+pmodulo(incor(2,k)-1,nn2);
      for j=1:passi(k+1)
            polC=[polC,npol2(:,1+pmodulo(jx+j-1,nn2))];
      end
      ix=1+pmodulo(incor(1,k+1)-1,nn1);
end
// tornando a n0
for i=1:passi(nx)
      polC=[polC,npol1(:,1+pmodulo(ix+i-1,nn1))];
end
//
plot2d(polC(1,:),polC(2,:),5)
xclick();
//
polCt=trasla(polC,[3;0]);
```

```
plot2d(polCt(1,:),polCt(2,:),5)
numerapol(polCt,barpol(polCt))
//.
```

[Download]

## 8.6 Composizione di due poligoni (1)

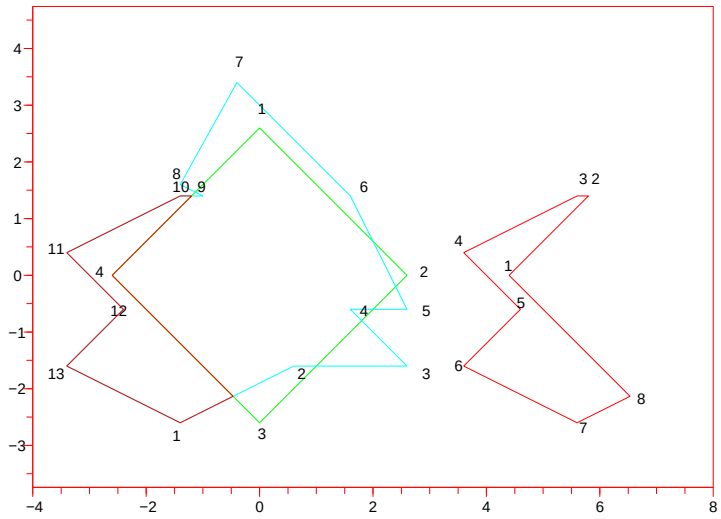



```
//:compc1.sce
//
// Composizione di due poligoni (1)
//
getf('lib_poligoni.sce');
getf('lib_area.sce');
getf('lib_deformazioni.sce');
getf('lib_intersezioni-e.sce');
getf('lib_riordina-vertici.sce');
getf('lib_sposta-vertici.sce');
getf('lib_incroci.sce');
getf('lib_ritagli.sce');
//
// ------<funzioni>-<inizio>----------------------------------------
//
n1=4;c1=[0;0];
pol1=poliga(n1,c1,1);
n2=3;c2=[-0.5;0.5];
u2=[0.5;-0.5];
pol2=trasla(poliga(n2,c2,1.2),u2); c2=c2+u2;
pol1=riordina(pol1,2);
//
//
clf();
xset("color",5)
plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
plot2d(pol1(1,:),pol1(2,:),3)
plot2d(pol2(1,:),pol2(2,:),4)
numerapol(pol1,c1)
numerapol(pol2,c2)
//
// Composizione attraverso la generazione di un percorso chiuso
// a partire da un vertice n0
//
[npol1,npol2,inc]=incroci(pol1,pol2);
```

```
polC=zigzac(npol1,npol2,inc,3);
//
plot2d(polC(1,:),polC(2,:),5)
xclick();
//
polCt=trasla(polC,[3;0]);
plot2d(polCt(1,:),polCt(2,:),5)
numerapol(polCt,barpol(polCt))
//.
```

[Download]

## 8.7 Composizione di due poligoni (2)

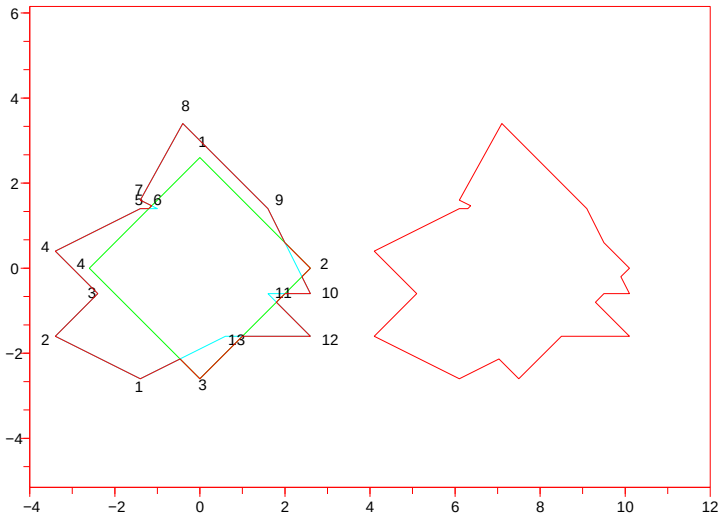



```
//:compc2.sce
//
// Composizione di due poligoni (2)
//
getf('lib_poligoni.sce');
getf('lib_area.sce');
getf('lib_deformazioni.sce');
getf('lib_intersezioni-e.sce');
getf('lib_riordina-vertici.sce');
getf('lib_sposta-vertici.sce');
getf('lib_incroci.sce');
getf('lib_ritagli.sce');
//
n1=4;c1=[0;0];
pol1=poligo(n1,c1,2.6);
pol2=[[3;-2],[1;-2],[-1;-3],[-3;-2],[-2;-1],[-3;0],[-1;1],[-0.6;1], ..
[-1;1.2],[0;3],[2;1],[3;-1],[2;-1],[3;-2]];
c2=[0;0];
n2=length(pol2(1,:))-1;
u2=-[0.4;-0.4];
pol2=trasla(pol2,u2); c2=c2+u2;
pol2=riordina(pol2,3);
pol2=inverti(pol2);
//
clf();
xset("color",5)
plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
plot2d(pol1(1,:),pol1(2,:),3)
plot2d(pol2(1,:),pol2(2,:),4)
numerapol(pol1,c1)
numerapol(pol2,c2)
//
// Composizione attraverso la generazione di un percorso chiuso
// a partire da un vertice n0
//
```

```
[npol1,npol2,inc]=incroci(pol1,pol2);
polC=zigzac(npol1,npol2,inc,10);
//
plot2d(polC(1,:),polC(2,:),5)
xclick();
//
polCt=trasla(polC,[7;0]);
plot2d(polCt(1,:),polCt(2,:),5)
numerapol(polCt,barpol(polCt))
//.
```

[Download]

## 8.8 Composizione di due poligoni (3)

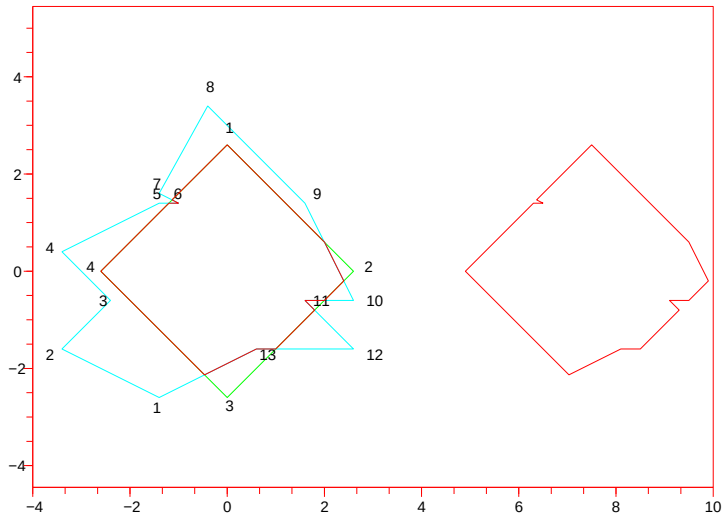



```
//:compc3.sce
//
// Composizione di due poligoni (3)
//
getf('lib_poligoni.sce');
getf('lib_area.sce');
getf('lib_deformazioni.sce');
getf('lib_intersezioni-e.sce');
getf('lib_riordina-vertici.sce');
getf('lib_sposta-vertici.sce');
getf('lib_incroci.sce');
getf('lib_ritagli.sce');
//
n1=4;c1=[0;0];
pol1=poligo(n1,c1,2.6);
pol2=[[3;-2],[1;-2],[-1;-3],[-3;-2],[-2;-1],[-3;0],[-1;1],[-0.6;1], ..
[-1;1.2],[0;3],[2;1],[3;-1],[2;-1],[3;-2]];
c2=[0;0];
n2=length(pol2(1,:))-1;
u2=-[0.4;-0.4];
pol2=trasla(pol2,u2); c2=c2+u2;
pol2=riordina(pol2,3);
//
clf();
xset("color",5)
plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
plot2d(pol1(1,:),pol1(2,:),3)
plot2d(pol2(1,:),pol2(2,:),4)
numerapol(pol1,c1)
numerapol(pol2,c2)
//
// Composizione attraverso la generazione di un percorso chiuso
// a partire da un vertice n0
//
[npol1,npol2,inc]=incroci(pol1,pol2);
```

```
polC=zigzac(npol1,npol2,inc,3);
//
plot2d(polC(1,:),polC(2,:),5)
xclick();
//
polCt=trasla(polC,[7.5;0]);
plot2d(polCt(1,:),polCt(2,:),5)
//.
```

[Download]

## 8.9 Composizione di due poligoni (4)

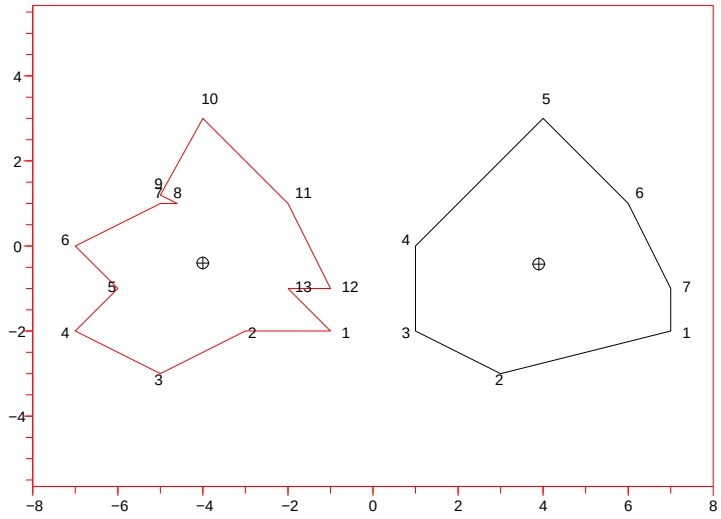



```
//:compc4.sce
//
// Composizione di due poligoni (4)
//
getf('lib_poligoni.sce');
getf('lib_area.sce');
getf('lib_deformazioni.sce');
getf('lib_intersezioni-e.sce');
getf('lib_riordina-vertici.sce');
getf('lib_sposta-vertici.sce');
getf('lib_incroci.sce');
getf('lib_ritagli.sce');
//
n1=4;c1=[0;0];
pol1=poligo(n1,c1,2.6);
pol2=[[3;-2],[1;-2],[-1;-3],[-3;-2],[-2;-1],[-3;0],[-1;1],[-0.6;1], ..
[-1;1.2],[0;3],[2;1],[3;-1],[2;-1],[3;-2]];
c2=[0;0];
n2=length(pol2(1,:))-1;
u2=-[0.4;-0.4];
pol2=trasla(pol2,u2); c2=c2+u2;
pol2=riordina(pol2,3);
//
clf();
xset("color",5)
plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
plot2d(pol1(1,:),pol1(2,:),3)
plot2d(pol2(1,:),pol2(2,:),4)
numerapol(pol1,c1)
numerapol(pol2,c2)
//
// Composizione attraverso la generazione di un percorso chiuso
// a partire da un vertice n0
//
[npol1,npol2,inc]=incroci(pol1,pol2);
```

```
polC=zigzac(npol1,npol2,inc,10);
//
plot2d(polC(1,:),polC(2,:),5)
xclick();
//
polCt=trasla(polC,[7.5;0]);
plot2d(polCt(1,:),polCt(2,:),5)
//.
```

[Download]

## 8.10 Convessificazione di un poligono irregolare



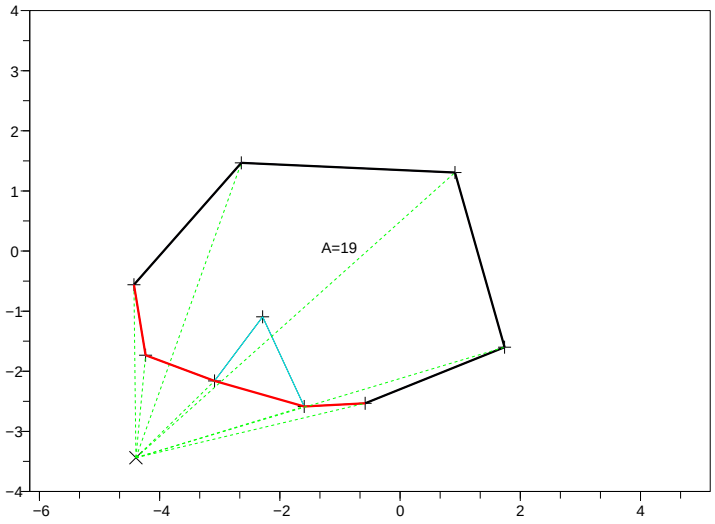

```
//:convp.sce
//
// Convessificazione di un poligono irregolare
// (gira intorno eliminando gli avvallamenti)
//
getf('lib_poligoni.sce');
getf('lib_area.sce');
getf('lib_deformazioni.sce');
getf('lib_intersezioni-e.sce');
getf('lib_riordina-vertici.sce');
getf('lib_sposta-vertici.sce');
//
// ------<funzioni>-<inizio>----------------------------------------
//
// Elimina un vertice
function npol=elimina(pol,n0)
      [nr,nc]=size(pol);
      npol=pol;
      if n0==1 then
            npol=[pol(:,2:nc-1),pol(:,2)]
      elseif n0<nc then
            npol=[pol(:,1:n0-1),pol(:,n0+1:nc)]
      end
endfunction
//
// ------<funzioni>-<fine>-----------------------------------------
//
pol0=[[3;-2],[1;-2],[-1;-3],[-3;-2],[-2;-1],[-3;0],[-1;1],[-0.6;1], ..
[-1;1.2],[0;3],[2;1],[3;-1],[2;-1],[3;-2]];
c0=[0;0];
clf();
xset("color",5)
plot2d(0,0,0,"031"," ",[-4,-4,4,4]);
plot2d(pol0(1,:),pol0(2,:),3)
numerapol(pol0,c0)
```

```
//
lines(0);
pol=pol0;
[nr,nc]=size(pol);
nv=nc-1;
ap=areapol(pol);
pg=barpol(pol);
//
// Conta delle insenature (test di convessita')
//
for i=1:nv+1
      i1=1+pmodulo(i-1,nv);
      i2=1+pmodulo(i,nv);
      i3=1+pmodulo(i+1,nv);
      at1=areatriangolo(pol(:,i1),pol(:,i2),pol(:,i3));
      at1m(i)=at1;
      if i==1 then baie=0;at2=at1;
      elseif (at2*at1<0) then
            baie=baie+1;
      end
      at2=at1;
      //
      clf();
      plot2d(0,0,0,"031"," ",[-4,-4,4,4]);
      plot2d(pol(1,:),pol(2,:),6);
      plot2d(pol(1,i1),pol(2,i1),-3);
      pg=barpol(pol);
      numerapol(pol,pg);xclick();
      //
end
baie=baie/2;
xstring(2,3,"ci sono "+string(baie)+" insenature");
xclick();
//
// Convessificazione
//
i=1; cons=1; test=0;
while (cons<nv+1) & (nv>3) do
      i1=1+pmodulo(i-1,nv);
      i2=1+pmodulo(i,nv);
      i3=1+pmodulo(i+1,nv);
      //
      clf();
      plot2d(0,0,0,"031"," ",[-4,-4,4,4]);
      plot2d(pol(1,:),pol(2,:),6);
      plot2d(pol(1,i1),pol(2,i1),-3);
      pg=barpol(pol);
      numerapol(pol,pg);xclick();
      //
      test=test+1;
      if (areatriangolo(pol(:,i1),pol(:,i2),pol(:,i3))*ap<0) then
            pol=elimina(pol,i2); nv=nv-1; cons=1; i=i-2;
      else i=i+1; cons=cons+1;
      end
end
clf();
xset("color",5)
plot2d(0,0,0,"031"," ",[-8,-4,8,4]);
```

```
pol0t=trasla(pol0,[-4;0]);
polt=trasla(pol,[4;0]);
plot2d(pol0t(1,:),pol0t(2,:),5)
plot2d(polt(1,:),polt(2,:))
pg0t=barpol(pol0t);
pgt=barpol(polt);
numerapol(pol0t,pg0t)
numerapol(polt,pgt)
plot2d(pg0t(1),pg0t(2),-3)
plot2d(pgt(1),pgt(2),-3)
//.
```

[Download]

## 8.11 Illuminazione da una sorgente




```
//:ombre.sce
//
// Illuminazione da una sorgente
//
getf('lib_poligoni.sce');
getf('lib_area.sce');
getf('lib_deformazioni.sce');
getf('lib_intersezioni-e.sce');
getf('lib_riordina-vertici.sce');
getf('lib_sposta-vertici.sce');
getf('lib_convesso.sce');
//
clf();
plot2d(0,0,1,"031",rect=[-4,-4,4,4]);

pol=poligono_m();
[nr,nc]=size(pol);
n=nc-1;
plot2d(pol(1,:),pol(2,:),4);
pol0=pol;

// Costruisce il poligono convesso circoscritto
pol=convessifica(pol0);xclick;
plot2d(pol(1,:),pol(2,:),6);
//
[nr,nc]=size(pol);n=nc-1;
// Si assegna il polo della suddivisione in triangoli
[b,x0,y0]=xclick();
p0=[x0;y0];
plot2d(x0,y0,-2);

// Suddivisione in triangoli e calcolo dell'area
A=0;
for i=1:n
      p1=pol(:,i);
```

```
      raggio=[p0,p1];
      p2=pol(:,i+1);
      lato=[p1,p2];
      At=areatriangolo(p0,p1,p2);
      A=A+At;
      if At>0 then col=1; else col=5; end
      plot2d(raggio(1,:),raggio(2,:),3)
      h=gce();h.children.line_style=3;
      plot2d(lato(1,:),lato(2,:),col);
      h=gce();h.children.thickness=3;
end
//
c0=barpol(pol);
xstring(c0(1),c0(2)+0.5,"A="+string(round(A*100)/100));
//.
```

[Download]

## 8.12 Funzioni (Intersezioni di segmenti)

```
//:lib_intersezioni-e.sce
//
// Funzioni (Intersezioni di segmenti)
//
// ------<funzioni>-<inizio>--------------------------------------
//
// Test di intersezione (due segmenti hanno un punto in comune ?)
// (esclude il caso di due segmenti tangenti)
// (comprende il caso di due segmenti adiacenti non allineati)
function v=intersecanti0(sa,sb)
      a1=sa(:,1); a2=sa(:,2);
      b1=sb(:,1); b2=sb(:,2);
      tab1=areatriangolo(a1,a2,b1);
      tab2=areatriangolo(a1,a2,b2);
      tba1=areatriangolo(b1,b2,a1);
      tba2=areatriangolo(b1,b2,a2);
      casoI1=(tab1*tab2<0)  & (tba1*tba2<0);
      casoI2=(tab1*tab2==0) & (tba1*tba2<0);
      casoI3=(tba1*tba2==0) & (tab1*tab2<0);
      casoI4=(tab1*tab2==0) & (tba1*tba2==0);
      caso0=(tab1==0) & (tab2==0);
      v= ~caso0 & (casoI1 | casoI2 | casoI3 | casoI4);
endfunction
//
// Test di intersezione con tolleranza
// (esclude il caso di due segmenti tangenti)
// (comprende il caso di due segmenti adiacenti non allineati)
function v=intersecanti(sa,sb)
      a1=sa(:,1); a2=sa(:,2);
      b1=sb(:,1); b2=sb(:,2);
      tab1=areatriangolo(a1,a2,b1);
      tab2=areatriangolo(a1,a2,b2);
      tba1=areatriangolo(b1,b2,a1);
      tba2=areatriangolo(b1,b2,a2);
      eps=10^(-6)*norm(a2-a1);
      casoI1=(tab1*tab2<-eps)  & (tba1*tba2<-eps);
      casoI2=(abs(tab1*tab2)<eps) & (tba1*tba2<-eps);
      casoI3=(abs(tba1*tba2)<eps) & (tab1*tab2<-eps);
      casoI4=(abs(tab1*tab2)<eps) & (abs(tba1*tba2)<eps);
      caso0=(abs(tab1)<eps) & (abs(tab2)<eps)
      v= ~caso0 & (casoI1 | casoI2 | casoI3 | casoI4);
endfunction
//
// Test di tangenza (due segmenti hanno una parte in comune ?)
// (comprende il caso di due segmenti adiacenti allineati)
function v=tangenti(sa,sb)
      a1=sa(:,1); a2=sa(:,2);
      b1=sb(:,1); b2=sb(:,2);
        tab1=areatriangolo(a1,a2,b1);
      tab2=areatriangolo(a1,a2,b2);
      eps=10^(-6)*norm(a2-a1);
      caso0=(abs(tab1)<eps) & (abs(tab2)<eps);
      xa1=a1(1); ya1=a1(2); xa2=a2(1); ya2=a2(2);
      xb1=b1(1); yb1=b1(2); xb2=b2(1); yb2=b2(2);
      na=[-(ya2-ya1);(xa2-xa1)]/10;
      nb=[-(yb2-yb1);(xb2-xb1)]/10;
```

```
      casoT1=intersecanti([a1+na,a2],[b1,b2]);
      casoT2=intersecanti([a1,a2+na],[b1,b2]);
      casoT3=intersecanti([a1,a2],[b1+nb,b2]);
      casoT4=intersecanti([a1,a2],[b1,b2+nb]);
      v= caso0 & (casoT1 | casoT2 | casoT3 | casoT4);
endfunction
//
// Calcolo del punto di intersezione
// (solo per segmenti intersecanti)
function p=intersezione(sa,sb)
      a1=sa(:,1); a2=sa(:,2);
      b1=sb(:,1); b2=sb(:,2);
      xa1=a1(1); ya1=a1(2);
      xa2=a2(1); ya2=a2(2);
      xb1=b1(1); yb1=b1(2);
      xb2=b2(1); yb2=b2(2);
      nax=ya2-ya1; nay=-(xa2-xa1); nbx=yb2-yb1; nby=-(xb2-xb1);
      N=[nax nay;nbx nby]; tn=[0;-((xa1-xb1)*nbx+(ya1-yb1)*nby)];
      p=inv(N)*tn + a1;
endfunction
// ------<funzioni>-<fine>----------------------------------------
//.
```

[Download]

## 8.13 Funzioni (Intersezioni di poligoni)

```
//:lib_incroci.sce
//
// Funzioni (Intersezioni di poligoni)
//
// ------<funzioni>-<inizio>--------------------------------------
//
// Test di appartenza di un vertice ad un poligono
// Restituisce il numero del vertice, oppure 0
function n=vertice(pol,p0)
      eps=10^(-6);
      [nr,nc]=size(pol);
      i=1;n=0;
      while (i<=nc)& (n==0) do
            if norm(pol(:,i)-p0)<eps then n=i;
            end
            i=i+1;
      end
endfunction
//
// Calcola le intersezioni dei lati di due poligoni e le aggiunge come vertici
function [npol1,npol2,inc]=incroci(pol1,pol2)
      //
      npol1=pol1;
      npol2=pol2;
      n1=length(pol1(1,:))-1;nn1=n1;
      n2=length(pol2(1,:))-1;nn2=n2;
      insv1=[1:nn1]*0;
      insv2=[1:nn2]*0;
      i=1;ninc=0;
      // Calcola le intersezioni dei lati, le numera, le aggiunge come vertici
      while i<=nn1 do
            a1=npol1(:,i);a2=npol1(:,i+1);sa=[a1,a2];
            j=1;
            while (j<=nn2) do
                  b1=npol2(:,j);b2=npol2(:,j+1);sb=[b1,b2];
                  if  intersecanti(sa,sb) then
                        p0=intersezione(sa,sb);
                        vea=vertice(sa,p0);veb=vertice(sb,p0);
                        nonvea=(vea==0);nonveb=(veb==0);
                        vint=(nonvea|nonveb);
                        if vint then ninc=ninc+1;
                        end
                        if nonvea then
                              npol1=aggiungivertice(npol1,p0,i);
                              nn1=nn1+1;
                              insv1=aggiungivertice(insv1,ninc,i);
                              a1=npol1(:,i);a2=npol1(:,i+1);sa=[a1,a2];
                        elseif vint then
                              insv1(i+vea-1)=ninc;
                        end
                        if nonveb then
                              npol2=aggiungivertice(npol2,p0,j);
                              nn2=nn2+1;
                              insv2=aggiungivertice(insv2,ninc,j);
                        elseif vint then
                              insv2(j+veb-1)=ninc;
```

```
                        end
                    end
                    j=j+1;
              end
              i=i+1;
        end
        // Crea una tabella di corrispondenza dei numeri dei vertici di intersezione
        inc1=insv1;n=0;
        for i=1:nn1
              ni=insv1(i);
              if ni>0 then inc1(ni)=i;n=n+1; end
        end
        inc2=insv2;n=0;
        for i=1:nn2
              ni=insv2(i);
              if ni>0 then inc2(ni)=i;n=n+1; end
        end
        inc=[inc1(1:n);inc2(1:n)];
endfunction
// ------<funzioni>-<fine>----------------------------------------
//.
```

[Download]

## 8.14 Funzioni (Riordinamento dei vertici)

```
//:lib_riordina-vertici.sce
//
// Funzioni (Riordinamento dei vertici)
//
// ------<funzioni>-<inizio>----------------------------------------
// Inverte l'ordine dei vertici
function npol=inverti(pol)
      [nr,nc]=size(pol);
      npol=pol;
      for i=1:nc
            j=nc-i+1;
            npol(:,i)=pol(:,j);
      end
endfunction
//
// Riordina i vertici mettendo al primo posto il vertice n0
function npol=riordina(pol,n0)
      [nr,nc]=size(pol);
      npol=pol;
      if (n0>1) & (n0<nc) then
            npol=[pol(:,n0:nc-1),pol(:,1:n0-1),pol(:,n0)]
      end
endfunction
//
// ------<funzioni>-<fine>-----------------------------------------
//.
```

[Download]

## 8.15 Funzioni (Modifiche dei vertici)

```
//:lib_sposta-vertici.sce
//
// Funzioni (Modifiche dei vertici)
//
// ------<funzioni>-<inizio>--------------------------------------
//
// Disegno di un poligono con il mouse
// -indcare la posizione di ciascun vertice con il tasto sinistro
// -terminare con tasto destro
//
function pol=dispol_m()
      b=3;
      n=0;
      pol=[];
      clf();
      xset("color",5)
      plot2d(0,0,1,"031","",[-2,-2,2,2]);
      while (b==0)|(b==3) do
            [b,x0,y0]=xclick();
            if (b==0)|(b==3) then
                  p0=[x0;y0];
                  pol=[pol,p0];
                  n=n+1;
                  plot2d(x0,y0,-1)
                  plot2d(pol(1,:),pol(2,:),3)
            end
      end
      pol=[pol,pol(:,1)];
      clf();
      xset("color",5)
      plot2d(0,0,1,"031","",[-2,-2,2,2]);
      plot2d(pol(1,:),pol(2,:),3)
      numerapol(pol,barpol(pol))
endfunction
//
// Modifica della posizione di un vertice con il mouse
// -indicare vertice da spostare con tasto sinistro
// -indicare nuova posizione con tasto sinistro
// -confermare con tasto centrale
// -terminare con tasto destro
//
function polm=spostavertici_m(pol)
      polm=pol;
      clf();
      xset("color",5)
      plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
      plot2d(polm(1,:),polm(2,:),3)
      numerapol(polm,barpol(polm))
      b=1;
      n=length(polm(1,:))-1;
      [b,x0,y0]=xclick();
      while (b<>2)&(b<>5) do
            n=length(polm(1,:))-1;
            p0=[x0;y0];
            d0=[];
            for i=1:n
```

```
            d0=[d0,norm(polm(:,i)-p0)];
      end
      [d,n0]=min(d0);
      plot2d(polm(1,n0),polm(2,n0),-2)
      //
      while (b<>1)&(b<>4) do
            clf();
            xset("color",5)
            plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
            plot2d(polm(1,:),polm(2,:),3)
            plot2d(polm(1,n0),polm(2,n0),-1)
            numerapol(polm,barpol(polm))
            [b,x0,y0]=xclick();
            p0=[x0;y0];
            if (b==0)|(b==3) then
                  if n0==1 then
                        polm=[p0,polm(:,2:n),p0];
                  else
                        polm=[polm(:,1:n0-1),p0,polm(:,n0+1:n+1)];
                  end
                  plot2d(x0,y0,-1)
            end
      end
      clf();
      xset("color",5)
      plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
      plot2d(polm(1,:),polm(2,:),3)
      numerapol(polm,barpol(polm))
      [b,x0,y0]=xclick();
end
endfunction
//
// Aggiunta di nuovi vertici con il mouse
// -indicare vertice precedente con tasto sinistro
// -indicare posizione nuovo vertice con tasto sinistro
// -confermare con tasto centrale
// -terminare con tasto destro
//
function polm=aggiungivertici_m(pol)
      polm=pol;
      clf();
      xset("color",5)
      plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
      plot2d(polm(1,:),polm(2,:),3)
      numerapol(polm,barpol(polm))
      b=1;
      n=length(polm(1,:))-1;
      [b,x0,y0]=xclick();
      while (b<>2)&(b<>5) do
            n=length(polm(1,:))-1;
            p0=[x0;y0];
            d0=[];
            for i=1:n
                  d0=[d0,norm(polm(:,i)-p0)];
            end
            [d,n0]=min(d0);
            plot2d(polm(1,n0),polm(2,n0),-2)
            //
```

```
            polm0=polm;
            while (b<>1)&(b<>4) do
                  clf();
                  xset("color",5)
                  plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
                  plot2d(polm0(1,:),polm0(2,:),3)
                  plot2d(polm0(1,n0),polm0(2,n0),-1)
                  numerapol(polm0,barpol(polm0))
                  [b,x0,y0]=xclick();
                  p0=[x0;y0];
                  if (b==0)|(b==3) then
                        plot2d(x0,y0,-1)
                        polm0=aggiungivertice(polm,p0,n0);
                  end
            end
            polm=polm0;
            clf();
            xset("color",5)
            plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
            plot2d(polm(1,:),polm(2,:),3)
            numerapol(polm,barpol(polm))
            [b,x0,y0]=xclick();
      end
endfunction
//
// Aggiunge ad un poligono un nuovo vertice p0 dopo il vertice n0
function npol=aggiungivertice(pol,p0,n0)
      [nr,nc]=size(pol);
      npol=[pol(:,1:n0),p0,pol(:,n0+1:nc)]
endfunction
// ------<funzioni>-<fine>----------------------------------------
//.
```

[Download]

## 8.16 Funzioni (Composizioni di poligoni)

```
//:lib_ritagli.sce
//
// Funzioni (Composizioni di poligoni)
//
// ------<funzioni>-<inizio>--------------------------------------
//
// Ritaglio di due poligoni:
// partendo dal vertice n0 del primo poligono (nella
// numerazione con i punti di intersezione)
// salta da uno all'altro bordo ad ogni intersezione
// seguendo il verso di ciascun poligono.
//
function polC=zigzac(npol1,npol2,inc,n0)
      //
      n=length(inc(1,:));
      polC=[];
      if n==0 then
            return(polC);
      end
      nn1=length(npol1(1,:))-1;
      nn2=length(npol2(1,:))-1;
      //
      n0=1+pmodulo(n0-1,nn1);
      // Ordina le coppie di vertici di intersezione
      // i denota un incrocio da cui si prosegue sul poligono 1
      // j denota un incrocio da cui si prosegue sul poligono 2
      incor=[];
      passi=[];
      [nvi,i]=min(pmodulo(inc(1,:)-n0,nn1));
      vip=inc(:,i);//primo incrocio
      incor=[incor,vip];
      passi=[passi,nvi];
      inc2=[inc(2,1:i-1),inc(2,i+1:n)];
      [nvi,i]=max(pmodulo(inc(1,:)-n0,nn1));
      viu=inc(:,i);//ultimo incrocio
      k=1;vi=[];vj=[];
      while ~(vj==viu)&~((vj==vip)&(vip(1)==n0)) do
            //
            [nvj,j]=min(pmodulo(inc2-incor(2,k),nn2));
            jx=find(inc(2,:)==inc2(j));
            vj=inc(:,jx);
            incor=[incor,inc(:,jx)];k=k+1;
            passi=[passi,nvj];
            inc1=[inc(1,1:jx-1),inc(1,jx+1:n)];
            //
            [nvi,i]=min(pmodulo(inc1-incor(1,k),nn1));
            ix=find(inc(1,:)==inc1(i));
            vi=inc(:,ix);
            incor=[incor,inc(:,ix)];k=k+1;
            passi=[passi,nvi];
            inc2=[inc(2,1:ix-1),inc(2,ix+1:n)];
      end;
      nx=k;
      incor=incor(:,1:nx-1);
      passi=[passi(:,1:nx-1),pmodulo(n0-incor(1,nx-1),nn1)];
      // Costruisce il percorso partendo da n0
```

```
      polC=[npol1(:,n0)];
      ix=n0;
      // proseguendo a zig zag
      for k=1:2:nx-1
            for i=1:passi(k)
                  polC=[polC,npol1(:,1+pmodulo(ix+i-1,nn1))];
            end
            jx=1+pmodulo(incor(2,k)-1,nn2);
            for j=1:passi(k+1)
                  polC=[polC,npol2(:,1+pmodulo(jx+j-1,nn2))];
            end
            ix=1+pmodulo(incor(1,k+1)-1,nn1);
      end
      // tornando a n0
      for i=1:passi(nx)
            polC=[polC,npol1(:,1+pmodulo(ix+i-1,nn1))];
      end
endfunction
//
// Stessa funzione con selezione del vertice iniziale con il mouse
// e visualizzazione passo passo
//
function polC=zigzac_m(npol1,npol2,inc)
      //
      n=length(inc(1,:));
      polC=[];
      if n==0 then
            return(polC);
      end
      nn1=length(npol1(1,:))-1;
      nn2=length(npol2(1,:))-1;
      // Determina il numero del vertice n0
      clf();
      xset("color",5)
      plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
      plot2d(npol1(1,:),npol1(2,:),3)
      plot2d(npol2(1,:),npol2(2,:),4)
      b=1;
      [b,x0,y0]=xclick();
      while (b<>2)&(b<>5) do
            p0=[x0;y0];
            d0=[];
            for i=1:nn1
                  d0=[d0,norm(npol1(:,i)-p0)];
            end
            [d,n0]=min(d0);
            //
            clf();
            xset("color",5)
            plot2d(0,0,0,"031"," ",[-2,-2,2,2]);
            plot2d(npol1(1,:),npol1(2,:),3)
            plot2d(npol2(1,:),npol2(2,:),4)
            plot2d(npol1(1,n0),npol1(2,n0),-2)
            [b,x0,y0]=xclick();
      end
      // Ordina le coppie di vertici di intersezione
      // i denota un incrocio da cui si prosegue sul poligono 1
      // j denota un incrocio da cui si prosegue sul poligono 2
```

```
      incor=[];
      passi=[];
      [nvi,i]=min(pmodulo(inc(1,:)-n0,nn1));
      vip=inc(:,i);//primo incrocio
      incor=[incor,vip];
      passi=[passi,nvi];
      inc2=[inc(2,1:i-1),inc(2,i+1:n)];
      [nvi,i]=max(pmodulo(inc(1,:)-n0,nn1));
      viu=inc(:,i);//ultimo incrocio
      k=1;vi=[];vj=[];
      while ~(vj==viu)&~((vj==vip)&(vip(1)==n0)) do
            //
            [nvj,j]=min(pmodulo(inc2-incor(2,k),nn2));
            jx=find(inc(2,:)==inc2(j));
            vj=inc(:,jx);
            incor=[incor,inc(:,jx)];k=k+1;
            passi=[passi,nvj];
            inc1=[inc(1,1:jx-1),inc(1,jx+1:n)];
            //
            [nvi,i]=min(pmodulo(inc1-incor(1,k),nn1));
            ix=find(inc(1,:)==inc1(i));
            vi=inc(:,ix);
            incor=[incor,inc(:,ix)];k=k+1;
            passi=[passi,nvi];
            inc2=[inc(2,1:ix-1),inc(2,ix+1:n)];
      end;
      nx=k;
      incor=incor(:,1:nx-1);
      passi=[passi(:,1:nx-1),pmodulo(n0-incor(1,nx-1),nn1)];
      // Costruisce il percorso partendo da n0
      polC=[npol1(:,n0)];
      ix=n0;
      // proseguendo a zig zag
      for k=1:2:nx-1
            for i=1:passi(k)
                  polC=[polC,npol1(:,1+pmodulo(ix+i-1,nn1))];
                  plot2d(polC(1,:),polC(2,:),5);xclick();
            end
            jx=1+pmodulo(incor(2,k)-1,nn2);
            for j=1:passi(k+1)
                  polC=[polC,npol2(:,1+pmodulo(jx+j-1,nn2))];
                  plot2d(polC(1,:),polC(2,:),5);xclick();
            end
            ix=1+pmodulo(incor(1,k+1)-1,nn1);
      end
      // tornando a n0
      for i=1:passi(nx)
            polC=[polC,npol1(:,1+pmodulo(ix+i-1,nn1))];
            plot2d(polC(1,:),polC(2,:),5);xclick();
      end
endfunction
// ------<funzioni>-<fine>----------------------------------------
//.
```

[Download]

## 8.17 Funzioni (Convessificazione)

```
//:lib_convesso.sce
//
// Funzioni (Convessificazione)
//
// ------<funzioni>-<inizio>--------------------------------------
//
// Elimina un vertice
function npol=elimina(pol,n0)
      [nr,nc]=size(pol);
      npol=pol;
      if n0==1 then
            npol=[pol(:,2:nc-1),pol(:,2)]
      elseif n0<nc then
            npol=[pol(:,1:n0-1),pol(:,n0+1:nc)]
      end
endfunction
//
// Convessificazione
//
function pol=convessifica(pol0)
pol=pol0;
[nr,nc]=size(pol);nv=nc-1;
ap=areapol(pol);
//
// Conta delle insenature (test di convessita')
//
for i=1:nv+1
      i1=1+pmodulo(i-1,nv);
      i2=1+pmodulo(i,nv);
      i3=1+pmodulo(i+1,nv);
      at1=areatriangolo(pol(:,i1),pol(:,i2),pol(:,i3));
      at1m(i)=at1;
      if i==1 then baie=0;at2=at1;
      elseif (at2*at1<0) then
            baie=baie+1;
      end
      at2=at1;
end
baie=baie/2;
//
// Convessificazione
//
i=1; cons=1; test=0;
while (cons<nv+1) & (nv>3) do
      i1=1+pmodulo(i-1,nv);
      i2=1+pmodulo(i,nv);
      i3=1+pmodulo(i+1,nv);
      //
      test=test+1;
      if (areatriangolo(pol(:,i1),pol(:,i2),pol(:,i3))*ap<0) then
            pol=elimina(pol,i2); nv=nv-1; cons=1; i=i-2;
      else i=i+1; cons=cons+1;
      end
end
endfunction
// ------<funzioni>-<fine>-----------------------------------------
```

```
//.
```

[Download]

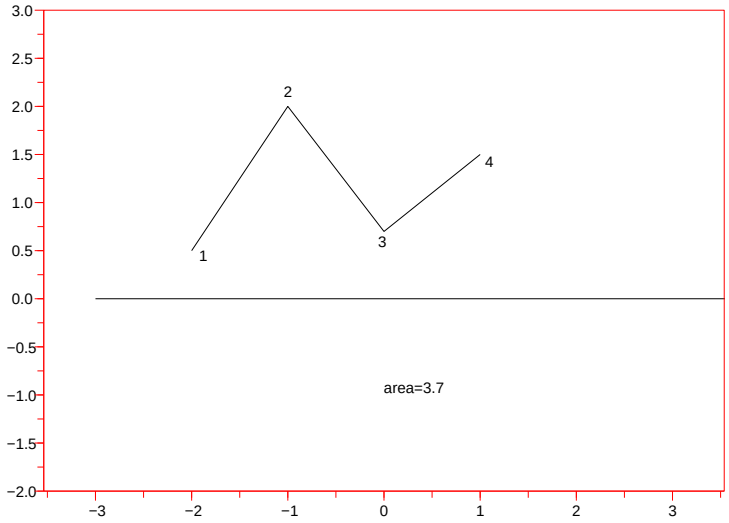
# Capitolo 9

# Area di trapezi e poligoni

Si calcola l'area della figura delimitata dal grafico di una funzione lineare a tratti come somma di aree di trapezi. All'area cosìcalcolata è attribuito in modo naturale un segno. Questo permette di estendere il calcolo dell'area ad un qualsiasi poligono.

Un'applicazione del calcolo delle aree consiste nell'approssimazione dell'integrale definito di una funzione.

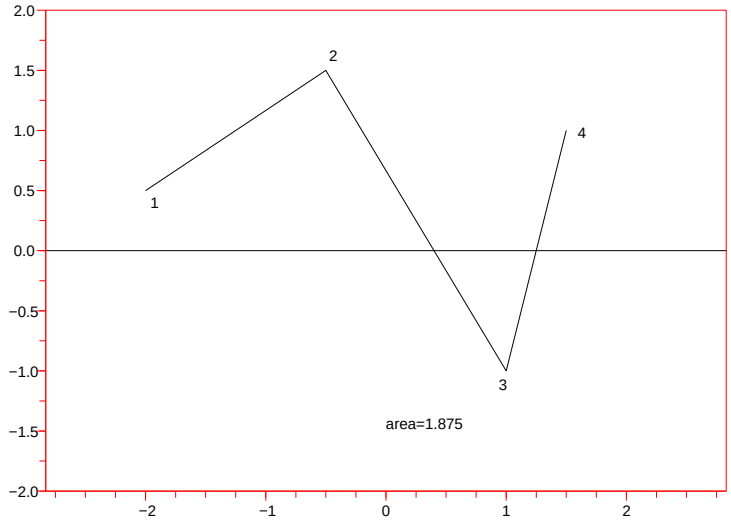
## 9.1 Area del grafico di una spezzata (1)




```
//:asp1.sce
//
// Area del grafico di una spezzata (1)
//
getf('lib_poligonali.sce')
clf();
p1=[-2;0.5]; p2=[-1;2]; p3=[0;0.7]; p4=[1;1.5];
sp=[p1,p2,p3,p4];
xset("color",5);
plot2d([-3,4],[0,0],1,"031"," ",[-2,-2,2,3]);
plot2d(sp(1,:),sp(2,:));
numera(sp);
area=areaf(sp);
xstring(0,-1,"area="+string(area))
//.
```

[Download]

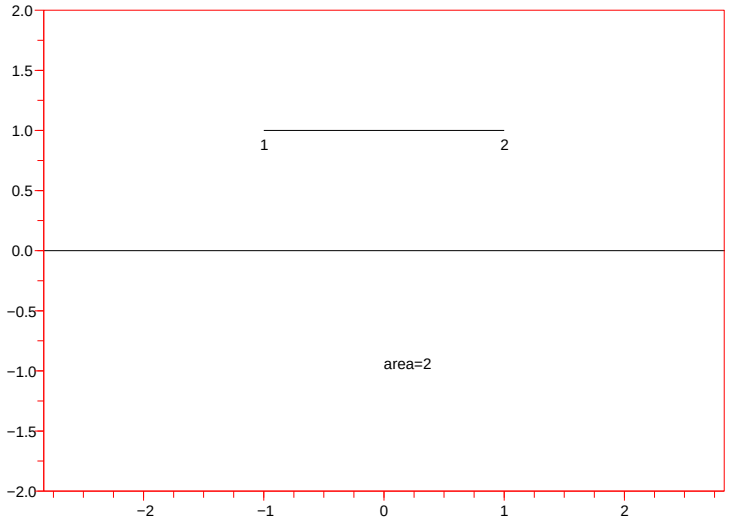
## 9.2 Area del grafico di una spezzata (2)




```
//:asp2.sce
//
// Area del grafico di una spezzata (2)
//
getf('lib_poligonali.sce')
clf();
p1=[-2;0.5]; p2=[-0.5;1.5]; p3=[1;-1]; p4=[1.5;1];
sp=[p1,p2,p3,p4];
xset("color",5);
plot2d([-3,4],[0,0],1,"031"," ",[-2,-2,2,2]);
plot2d(sp(1,:),sp(2,:));
numera(sp);
xstring(0,-1.5,"area="+string(areaf(sp)))
//.
```

[Download]

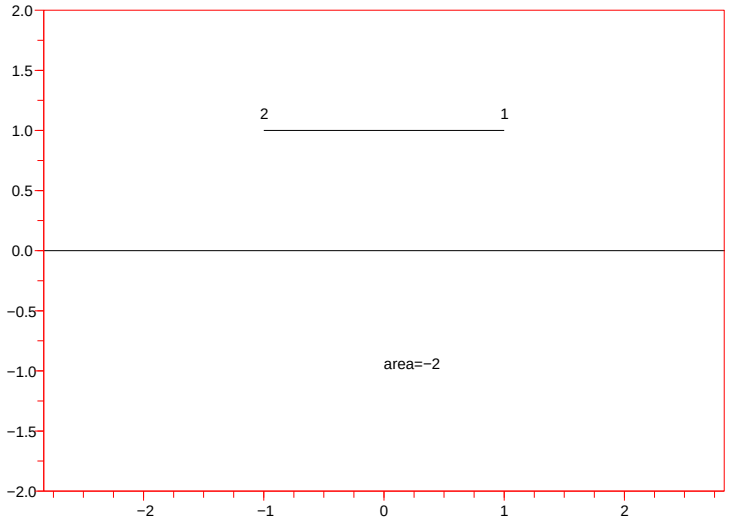
## 9.3 Area del grafico di un segmento (1)




```
//:asg1.sce
//
// Area del grafico di un segmento (1)
//
getf('lib_poligonali.sce')
clf();
p1=[-1;1]; p2=[1;1];
s=[p1,p2];
xset("color",5);
plot2d([-3,3],[0,0],1,"031"," ",[-2,-2,2,2]);
plot2d(s(1,:),s(2,:));
numera(s);
xstring(0,-1,"area="+string(areaf(s)))
//.
```

[Download]

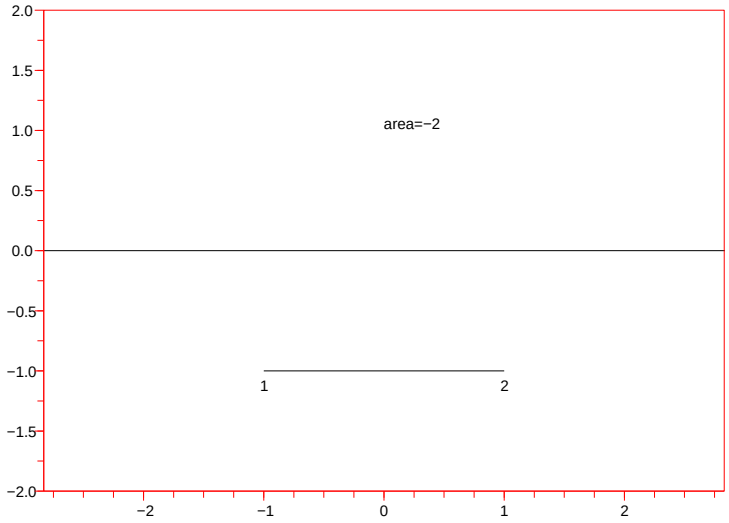
## 9.4 Area del grafico di un segmento (2)




```
//:asg2.sce
//
// Area del grafico di un segmento (2)
//
getf('lib_poligonali.sce')
clf();
p2=[-1;1]; p1=[1;1];
s=[p1,p2];
xset("color",5);
plot2d([-3,3],[0,0],1,"031"," ",[-2,-2,2,2]);
plot2d(s(1,:),s(2,:));
numera(s);
xstring(0,-1,"area="+string(areaf(s)))
//.
```

[Download]

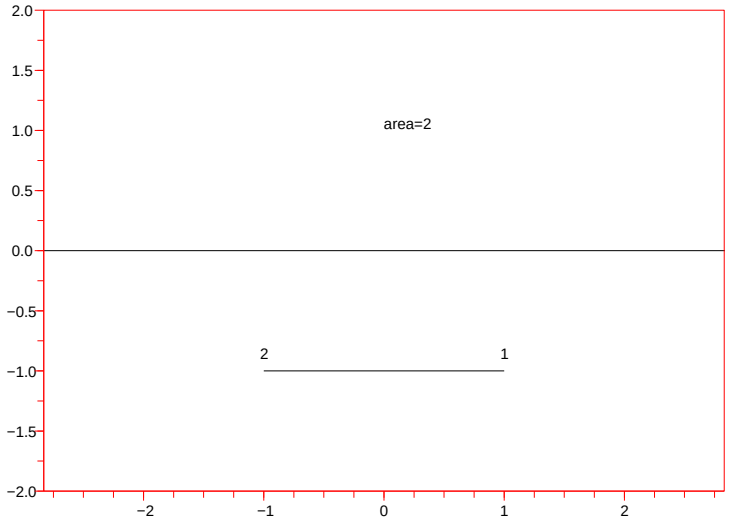
## 9.5 Area del grafico di un segmento (3)




```
//:asg3.sce
//
// Area del grafico di un segmento (3)
//
getf('lib_poligonali.sce')
clf();
p1=[-1;-1]; p2=[1;-1];
s=[p1,p2];
xset("color",5);
plot2d([-3,3],[0,0],1,"031"," ",[-2,-2,2,2]);
plot2d(s(1,:),s(2,:));
numera(s);
xstring(0,1,"area="+string(areaf(s)))
//.
```

[Download]

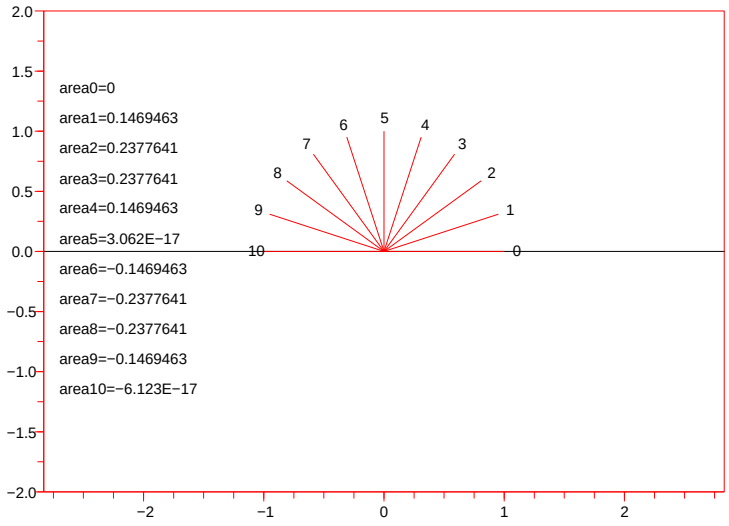
## 9.6 Area del grafico di un segmento (4)




```
//:asg4.sce
//
// Area del grafico di un segmento (4)
//
getf('lib_poligonali.sce')
clf();
p1=[1;-1]; p2=[-1;-1];
s=[p1,p2];
xset("color",5);
plot2d([-3,3],[0,0],1,"031"," ",[-2,-2,2,2]);
plot2d(s(1,:),s(2,:));
numera(s);
xstring(0,1,"area="+string(areaf(s)))
//.
```

[Download]

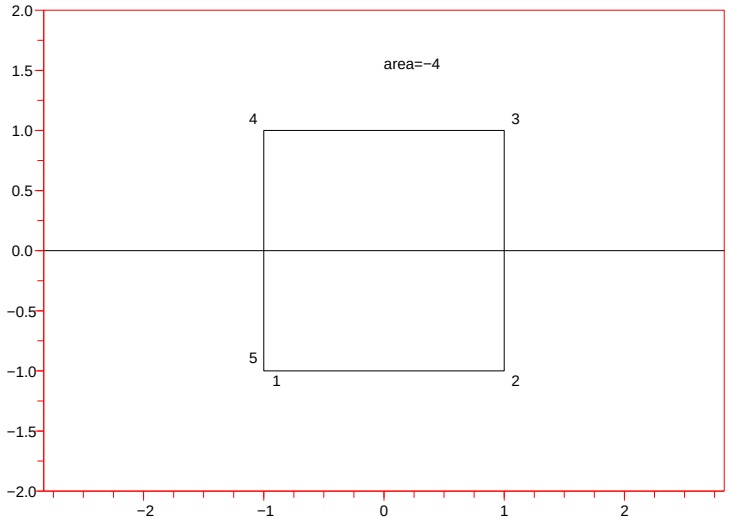
## 9.7 Area del grafico di un segmento che ruota




```
//:asg5.sce
//
// Area del grafico di un segmento che ruota
//
getf('lib_poligoni.sce')
getf('lib_deformazioni.sce')
getf('lib_poligonali.sce')
a=%pi/10;
p1=[0;0]; p2=[1;0];
s=[p1,p2];
clf();
xset("color",5);
plot2d([-3,3],[0,0],1,"031"," ",[-2,-2,2,2]);
c0=[0;0];
for i=0:10
      sr=ruota(s,c0,a*i);
      plot2d(sr(1,:),sr(2,:),5);
      area=areaf(sr);
      xstring(sr(1,2)*1.1-0.03,sr(2,2)*1.1-0.05,string(i));
      xstring(-2.7,1.3-i/4,"area"+string(i)+"="+string(area))
end;
//.
```

[Download]

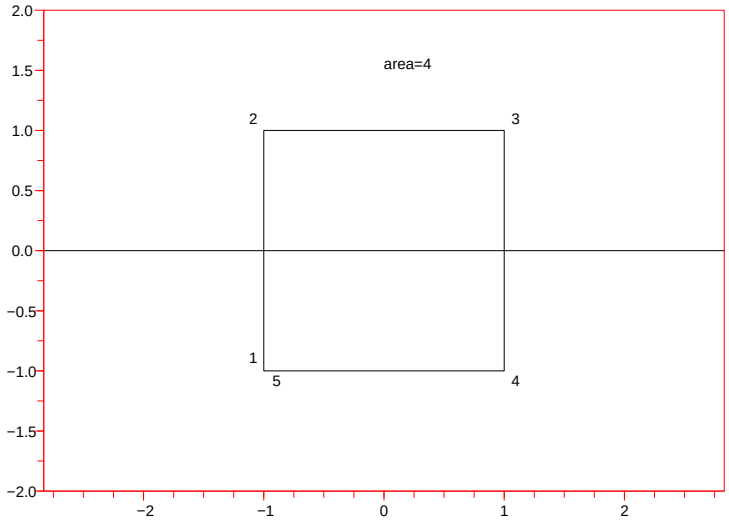
## 9.8 Area di un quadrato (verso antiorario)




```
//:aqu1.sce
//
// Area di un quadrato (verso antiorario)
//
getf('lib_poligoni.sce')
getf('lib_poligonali.sce')
p1=[-1;-1]; p2=[1;-1]; p3=[1;1]; p4=[-1;1];
sp=[p1,p2,p3,p4,p1];
clf();
xset("color",5);
plot2d([-3,3],[0,0],1,"031"," ",[-2,-2,2,2]);
plot2d(sp(1,:),sp(2,:));
numera(sp);
area=areaf(sp);
xstring(0,1.5,"area="+string(area))
//.
```

[Download]

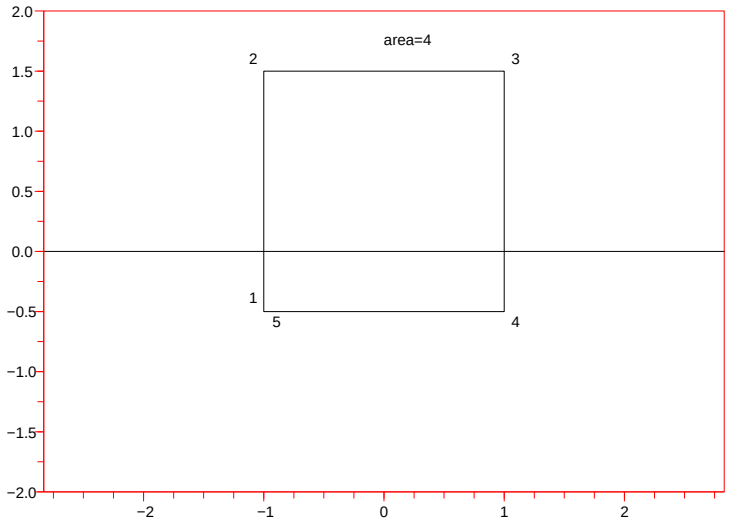
## 9.9 Area di un quadrato (verso orario)




```
//:aqu2.sce
//
// Area di un quadrato (verso orario)
//
getf('lib_poligoni.sce')
getf('lib_poligonali.sce')
p1=[-1;-1]; p4=[1;-1]; p3=[1;1]; p2=[-1;1];
sp=[p1,p2,p3,p4,p1];
clf();
xset("color",5);
plot2d([-3,3],[0,0],1,"031"," ",[-2,-2,2,2]);
plot2d(sp(1,:),sp(2,:));
numera(sp);
area=areaf(sp);
xstring(0,1.5,"area="+string(area))
//.
```

[Download]

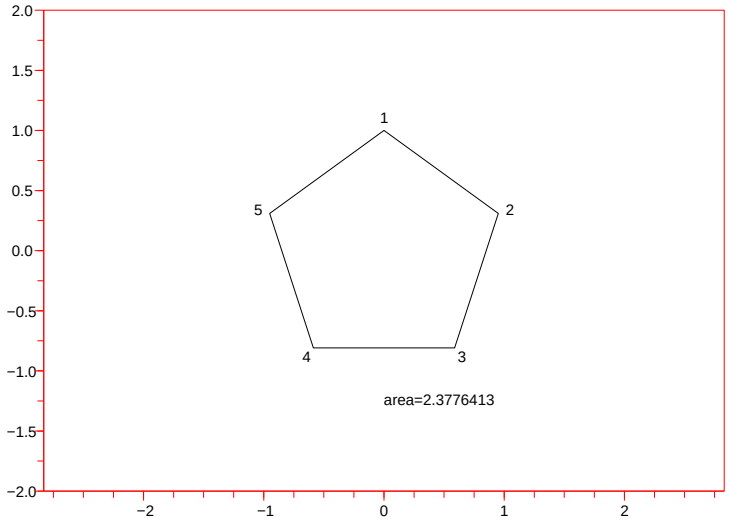
## 9.10 Area di un quadrato traslato




```
//:aqu3.sce
//
// Area di un quadrato traslato
//
getf('lib_poligoni.sce')
getf('lib_deformazioni.sce')
getf('lib_poligonali.sce')
u0=[0;0.5];
p1=[-1;-1]; p4=[1;-1]; p3=[1;1]; p2=[-1;1];
sp=trasla([p1,p2,p3,p4,p1],u0);
clf();
xset("color",5);
plot2d([-3,3],[0,0],1,"031"," ",[-2,-2,2,2]);
plot2d(sp(1,:),sp(2,:));
numera(sp);
area=areaf(sp);
xstring(0,1.7,"area="+string(area))
//.
```

[Download]

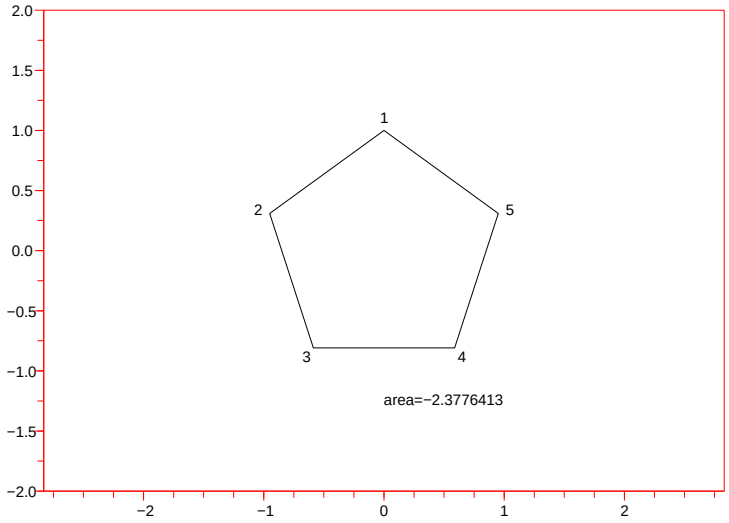
## 9.11 Area di un pentagono (verso orario)




```
//:ape1.sce
//
// Area di un pentagono (verso orario)
//
getf('lib_poligoni.sce')
getf('lib_poligonali.sce')
c0=[0;0];
fig=poligo(5,c0,1);
clf();
xset("color",5);
plot2d(0,0,1,"031"," ",[-2,-2,2,2]);
plot2d(fig(1,:),fig(2,:));
numerapol(fig,c0);
area=areaf(fig);
xstring(0,-1.3,"area="+string(area))
//.
```

[Download]

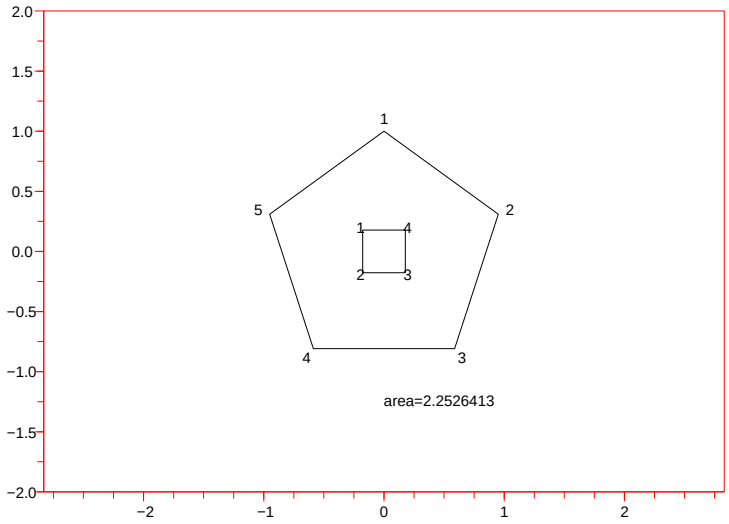
## 9.12 Area di un pentagono (verso antiorario)




```
//:ape2.sce
//
// Area di un pentagono (verso antiorario)
//
getf('lib_poligoni.sce')
getf('lib_poligonali.sce')
c0=[0;0];
fig=poliga(5,c0,1);
clf();
xset("color",5);
plot2d(0,0,1,"031"," ",[-2,-2,2,2]);
plot2d(fig(1,:),fig(2,:));
numerapol(fig,c0);
area=areaf(fig);
xstring(0,-1.3,"area="+string(area))
//.
```

[Download]

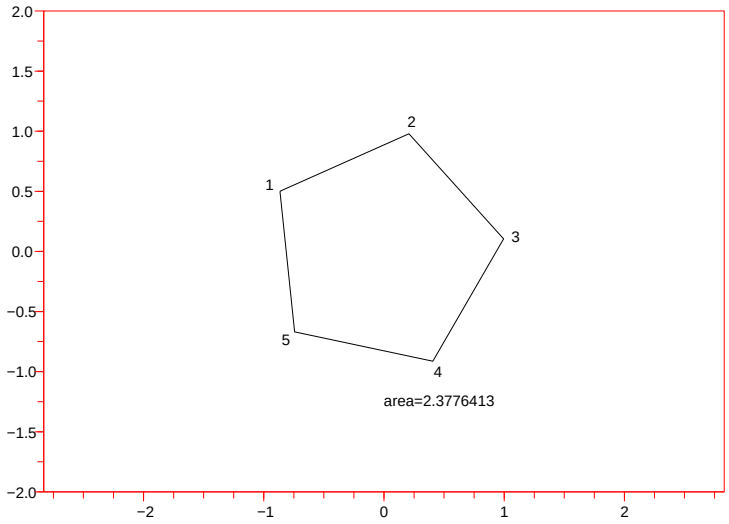
## 9.13 Area di un pentagono con un foro




```
//:ape3.sce
//
// Area di un pentagono con un foro
//
getf('lib_poligoni.sce')
getf('lib_deformazioni.sce')
getf('lib_poligonali.sce')
c0=[0;0];
fig=poligo(5,c0,1);
foro=ruota(poliga(4,c0,1/4),c0,%pi/4);
clf();
xset("color",5);
plot2d(0,0,1,"031"," ",[-2,-2,2,2]);
plot2d(fig(1,:),fig(2,:));
numerapol(fig,c0);
plot2d(foro(1,:),foro(2,:));
numerapol(foro,c0);
area=areaf(fig)+areaf(foro);
xstring(0,-1.3,"area="+string(area))
//.
```

[Download]

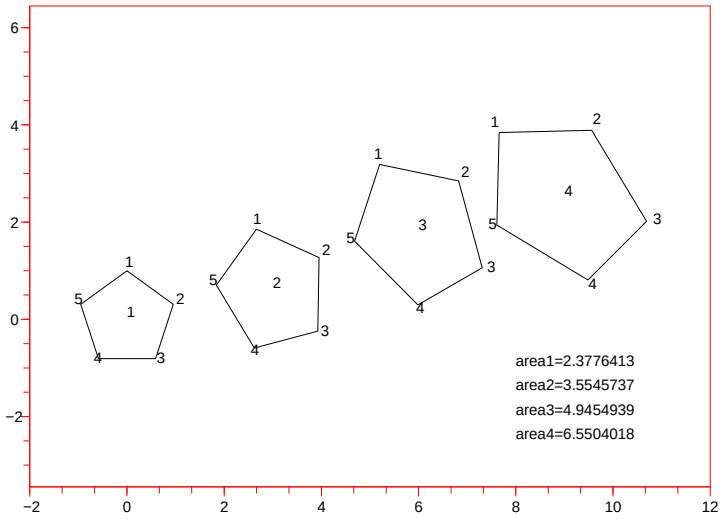
## 9.14 Area di un pentagono ruotato




```
//:ape4.sce
//
// Area di un pentagono ruotato
//
getf('lib_poligoni.sce')
getf('lib_deformazioni.sce')
getf('lib_poligonali.sce')
c0=[0;0];
fig=ruota(poligo(5,c0,1),c0,%pi/3);
clf();
xset("color",5);
plot2d(0,0,1,"031"," ",[-2,-2,2,2]);
plot2d(fig(1,:),fig(2,:));
numerapol(fig,c0);
area=areaf(fig);
xstring(0,-1.3,"area="+string(area))
//
//.
```

[Download]

## 9.15 Area di un pentagono deformato




```
//:ape5.sce
//
// Area di un pentagono deformato
//
getf('lib_poligoni.sce')
getf('lib_deformazioni.sce')
getf('lib_poligonali.sce')
c0=[0;0];
fig1=poligo(5,c0,1);
clf();
xset("color",5);
plot2d(0,0,1,"031"," ",[-2,-2,12,5]);
u1=[0;0]; u2=[3;0.6]; u3=[6;1.8]; u4=[9;2.5];
u=[u1,u2,u3,u4];
a=0;
epsx=0.15;
epsy=0.3;
fig2=fig1;
for m=1:4
      U=[1+epsx*(m-1),0;0,1+epsy*(m-1)];
      u0=u(:,m);
      fig2=trasla(ruota(dilata(fig1,c0,U),c0,a),u0);
      plot2d(fig2(1,:),fig2(2,:));
      c2=c0+u0;
      numerapol(fig2,c2);
      xstring(c2(1),c2(2),string(m));
      a=a+%pi/12;
      area=areaf(fig2);
      xstring(8,-0.5-m/2,"area"+string(m)+"="+string(area))
end;
//.
```

[Download]

## 9.16 Funzioni (Poligonali e aree di trapezi)

```
//:lib_poligonali.sce
//
// Funzioni (Poligonali e aree di trapezi)
//
// ------<funzioni>-<inizio>----------------------------------------

// Numerazione, essenziale, dei vertici di una poligonale
function numera0(sp)
      [nr,nc]=size(sp);
      ag=[0.03;0.05]; // aggiustamento posizione del numero
      for i=1:nc
            pos=sp(:,i)-ag;
            xstring(pos(1),pos(2),string(i))
      end;
endfunction

// Numerazione, ben posizionata, dei vertici di una poligonale
function numera(sp)
      [nr,nc]=size(sp);
      ag=[0.03;0.05]; // aggiustamento posizione del numero
      for i=1:nc
            vpre=[0;0]; vpost=[0;0];
            if i>1  then vpre=sp(:,i)-sp(:,i-1), end
            if i<nc then vpost=sp(:,i+1)-sp(:,i), end
            vsec=(vpre+vpost)/norm(vpre+vpost);
            vort=[-vsec(2);vsec(1)];
            vest=-vsec/2+vpre;
            if (vort' * vest > 0) then vpos=vort, else vpos=-vort, end
            if ((i==1) | (i==nc)) & (norm(sp(:,nc)-sp(:,1)) < 3*norm(ag)) & (nc>3) then
                  vest=sp(:,i)-(sp(:,2)+sp(:,nc-1))/2
                  vpos=vest/norm(vest)+1.5*(vpost-vpre)/norm(vpost-vpre)
            end
            pos=sp(:,i)+vpos*0.13-ag;
            xstring(pos(1),pos(2),string(i))
      end;
endfunction

// Calcolo dell'area con la regola dei trapezi
function area=areaf(sp)
      x=sp(1,:), y=sp(2,:)
      [nr,nc]=size(sp);
      area=0
      for i=1:nc-1
            area=area+(x(i+1)-x(i))*(y(i+1)+y(i))/2
      end
endfunction
// ------<funzioni>-<fine>-----------------------------------------
//.
```

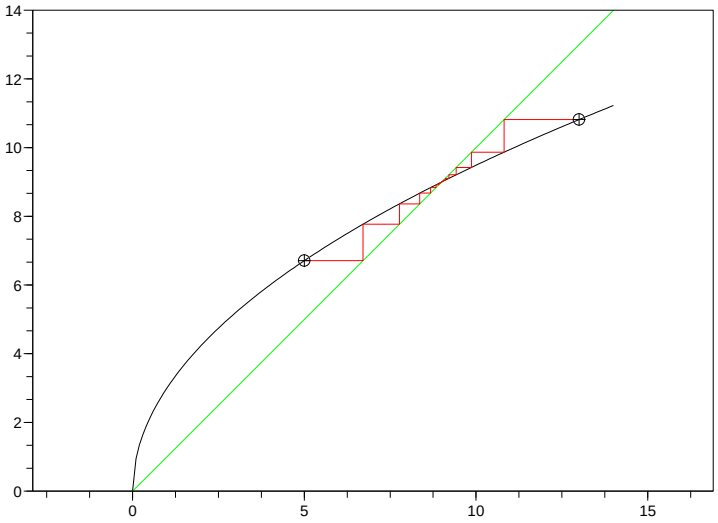
[Download]

# Capitolo 10

# Successioni e punti fissi

In questo capitolo si vede come costruire successioni definite per ricorrenza attraverso una funzione $f : \mathbb{R} \to \mathbb{R}$ e come visualizzarne il comportamento nelle vicinanze dei punti fissi di $f$.

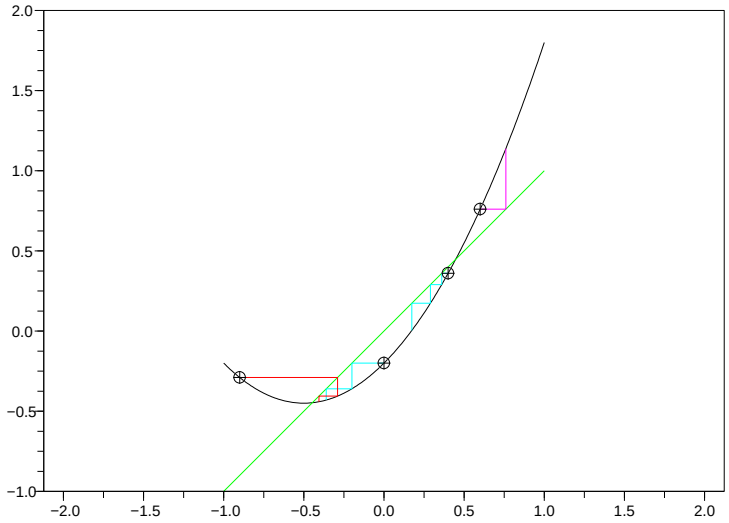
## 10.1 Successione generata da una funzione (1)




```
//:succ01.sce
//
// Successione generata da una funzione (1)
//
function y=f(x), y=3*sqrt(x), endfunction
function yp=fp(x), dx=0.01, yp=(f(x+dx)-f(x))/dx, endfunction
x=[0:0.1:14];
clf();
plot2d(0,0,1,"031")
plot2d(x,f(x))
plot2d(x,x,3)
N=30;
a0=13;
a=[a0 0];
for i=1:N-1
      a(i+1)=f(a(i));
end
plot2d2(a,f(a),5)
plot2d(a(1),a(2),-3)
N=30;
a0=5;
a=[a0 0];
for i=1:N-1
      a(i+1)=f(a(i));
end
plot2d2(a,f(a),5)
plot2d(a(1),a(2),-3)
//.
```

[Download]

## 10.2 Successione generata da una funzione (2)

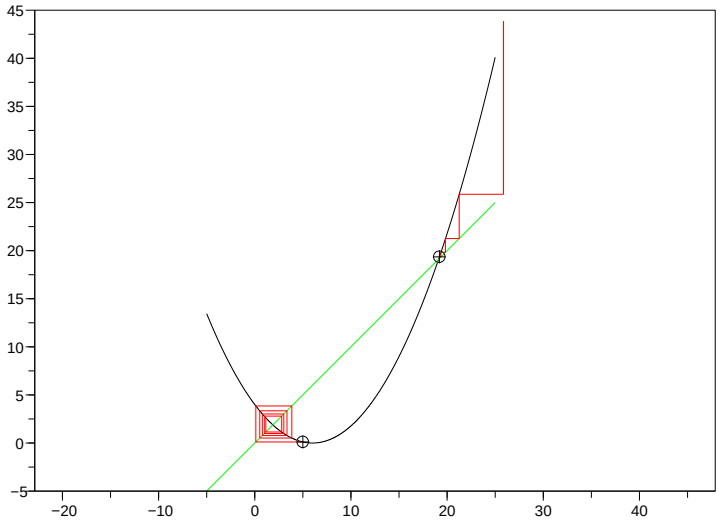



```
//:succ02.sce
//
// Successione generata da una funzione (2)
//
function y=f(x), y=(x^2)+x-0.2, endfunction
function yp=fp(x), dx=0.0001, yp=(f(x+dx)-f(x))/dx, endfunction
x=[-1:0.01:1];
clf();
plot2d(0,0,1,"031")
plot2d(x,f(x))
plot2d(x,x,3)
//
N=3;
a0=0;
a=[a0 0];
for i=1:N-1
      a(i+1)=f(a(i));
end
plot2d2(a,f(a),4)
plot2d(a(1),a(2),-3)
//
N=3;
a0=-0.9;
a=[a0 0];
for i=1:N-1
      a(i+1)=f(a(i));
end
plot2d2(a,f(a),5)
plot2d(a(1),a(2),-3)
//
N=2;
a0=0.6;
a=[a0 0];
for i=1:N-1
      a(i+1)=f(a(i));
```

```
end
plot2d2(a,f(a),6)
plot2d(a(1),a(2),-3)
//
N=4;
a0=0.4;
a=[a0 0];
for i=1:N-1
      a(i+1)=f(a(i));
end
plot2d2(a,f(a),4)
plot2d(a(1),a(2),-3)
//.
```

[Download]

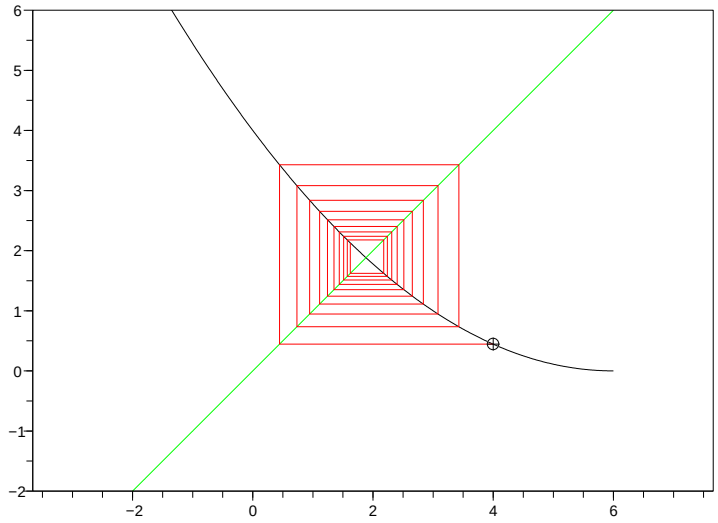
## 10.3 Successione generata da una funzione (3)




```
//:succ03.sce
//
// Successione generata da una funzione (3)
//
function y=f(x), y=(x/3-2)^2, endfunction
function yp=fp(x), dx=0.01, yp=(f(x+dx)-f(x))/dx, endfunction
x=[-5:0.01:25];
clf();
plot2d(0,0,1,"031")
plot2d(x,f(x))
plot2d(x,x,3)
//
N=10;
a=[5 0];
for i=1:N-1
      a(i+1)=f(a(i));
end
plot2d2(a,f(a),5)
plot2d(a(1),a(2),-3)
//
N=5;
a0=19.2;
a=[a0 0];
for i=1:N-1
      a(i+1)=f(a(i));
end
plot2d2(a,f(a),5)
plot2d(a(1),a(2),-3)
//.
```

[Download]

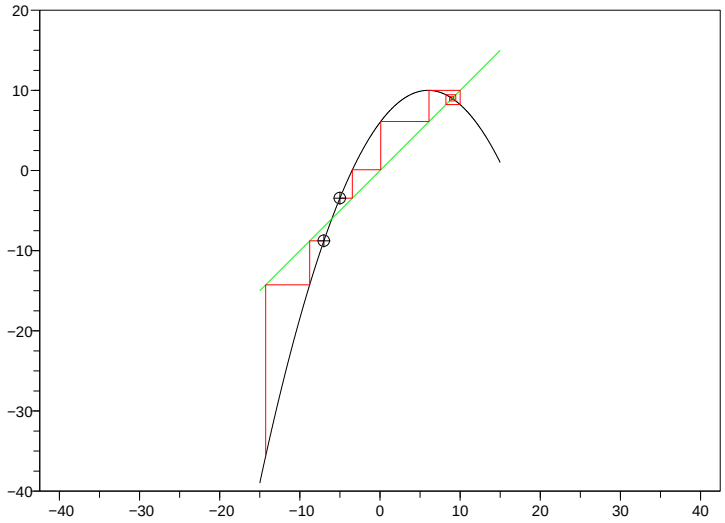
## 10.4 Successione generata da una funzione (4)




```
//:succ04.sce
//
// Successione generata da una funzione (4)
//
function y=f(x), y=(x/3-2)^2, endfunction
function yp=fp(x), dx=0.01, yp=(f(x+dx)-f(x))/dx, endfunction
x=[-2:0.01:6];
clf();
plot2d(x,f(x))
plot2d(0,0,1,"031"))
plot2d(x,x,3)
N=20;
a0=4;
a=[a0 0];
for i=1:N-1
      a(i+1)=f(a(i));
end
plot2d2(a,f(a),5)
plot2d(a(1),a(2),-3)
//.
```

[Download]

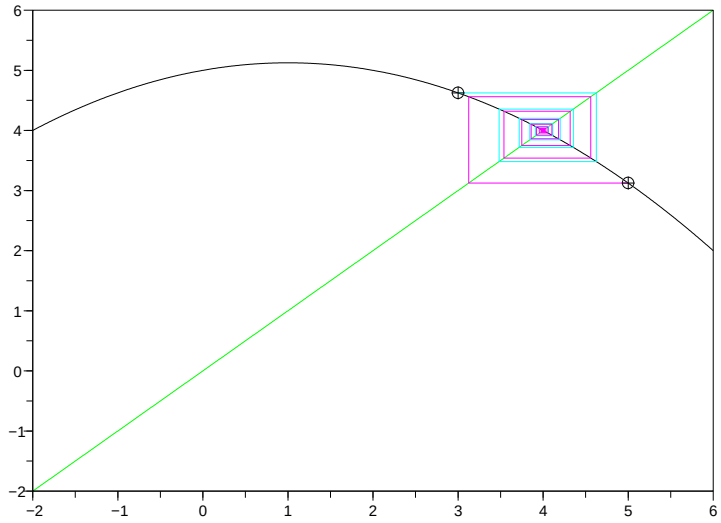
## 10.5 Successione generata da una funzione (5)




```
//:succ05.sce
//
// Successione generata da una funzione (5)
//
function y=f(x), y=-(x/3-2)^2+10, endfunction
function yp=fp(x), dx=0.01, yp=(f(x+dx)-f(x))/dx, endfunction
x=[-15:0.01:15];
clf();
plot2d(0,0,1,"031")
plot2d(x,f(x))
plot2d(x,x,3)
N=10;
a0=-5;
a=[a0 0];
for i=1:N-1
      a(i+1)=f(a(i));
end
plot2d2(a,f(a),5)
plot2d(a(1),a(2),-3)
N=3;
a0=-7;
a=[a0 0];
for i=1:N-1
      a(i+1)=f(a(i));
end
plot2d2(a,f(a),5)
plot2d(a(1),a(2),-3)
//.
```

[Download]

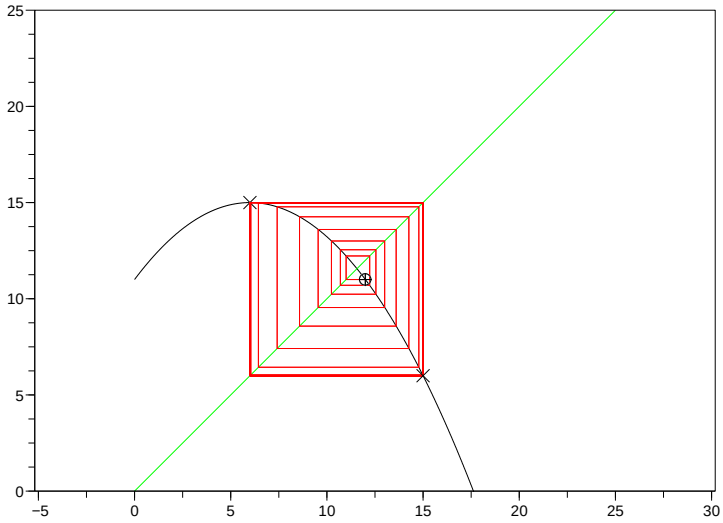
## 10.6 Successione generata da una funzione (6)




```
//:succ06.sce
//
// Successione generata da una funzione (6)
//
function y=f(x), y=(-x^2 +2*x +40)/8, endfunction
function yp=fp(x), dx=0.01, yp=(f(x+dx)-f(x))/dx, endfunction
x=[-2:0.01:6];
clf();
//plot2d(0,0,1,"031")
plot2d(x,f(x))
plot2d(x,x,3)
//
N=10;
a0=3;
a=[a0 0];
for i=1:N-1
      a(i+1)=f(a(i));
end
plot2d2(a,f(a),4)
plot2d(a(1),a(2),-3)
//
N=20;
a0=5;
a=[a0 0];
for i=1:N-1
      a(i+1)=f(a(i));
end
plot2d2(a,f(a),6)
plot2d(a(1),a(2),-3)
//.
```

[Download]

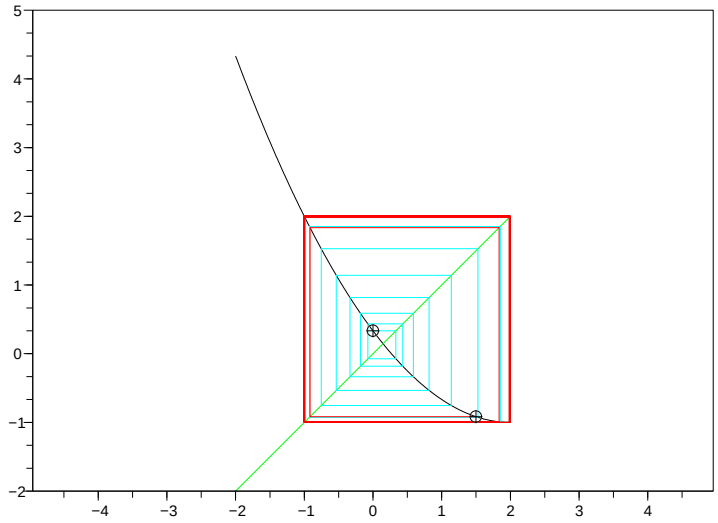
## 10.7 Successione generata da una funzione (7)




```
//:succ07.sce
//
// Successione generata da una funzione (7)
//
function y=f(x), y=-(x/3-2)^2+15, endfunction
function yp=fp(x), dx=0.01, yp=(f(x+dx)-f(x))/dx, endfunction
x=[0:0.01:25];
clf();
plot2d(x,f(x))
plot2d(0,0,1,"031")
plot2d(x,x,3)
//
N=30; // pari
a0=12;
a=[a0 0];
for i=1:N-1
      a(i+1)=f(a(i));
end
plot2d2(a,f(a),5)
plot2d(a(1),a(2),-3)
plot2d(a(N),f(a(N)),-2)
//
N=31; // dispari
a0=12;
a=[a0 0];
for i=1:N-1
      a(i+1)=f(a(i));
end
plot2d2(a,f(a),5)
plot2d(a(1),a(2),-3)
plot2d(a(N),f(a(N)),-2)
//.
```

[Download]

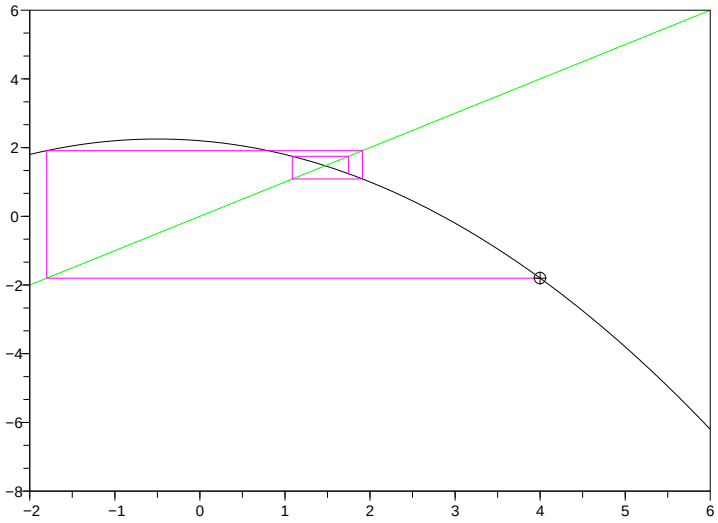
## 10.8 Successione generata da una funzione (8)




```
//:succ08.sce
//
// Successione generata da una funzione (8)
//
function y=f(x), y=((x-2)^2)/3-1, endfunction
function yp=fp(x), dx=0.01, yp=(f(x+dx)-f(x))/dx, endfunction
x=[-2:0.01:2];
clf();
plot2d(0,0,1,"031")
plot2d(x,f(x))
plot2d(x,x,3)
//
N=30;
a0=0;
a=[a0 0];
for i=1:N-1
      a(i+1)=f(a(i));
end
plot2d2(a,f(a),4)
plot2d(a(1),a(2),-3)
//
N=20;
a0=1.5;
a=[a0 0];
for i=1:N-1
      a(i+1)=f(a(i));
end
plot2d2(a,f(a),5)
plot2d(a(1),a(2),-3)
//.
```

[Download]

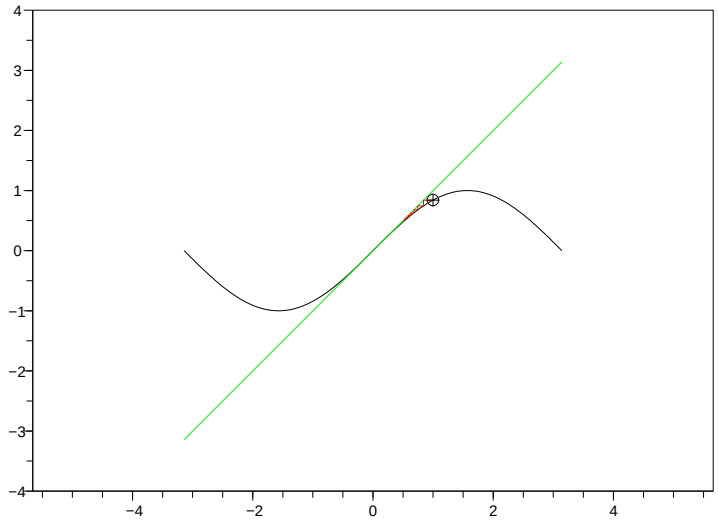
## 10.9 Successione generata da una funzione (9)




```
//:succ09.sce
//
// Successione generata da una funzione (9)
//
function y=f(x), y=-(x+3)^2/5+x+4, endfunction
function yp=fp(x), dx=0.01, yp=(f(x+dx)-f(x))/dx, endfunction
x=[-2:0.01:6];
clf();
//plot2d(0,0,1,"031")
plot2d(x,f(x))
plot2d(x,x,3)
//
N=5;
a0=4;
a=[a0 0];
for i=1:N-1
      a(i+1)=f(a(i));
end
plot2d2(a,f(a),6)
plot2d(a(1),a(2),-3)
//.
```

[Download]

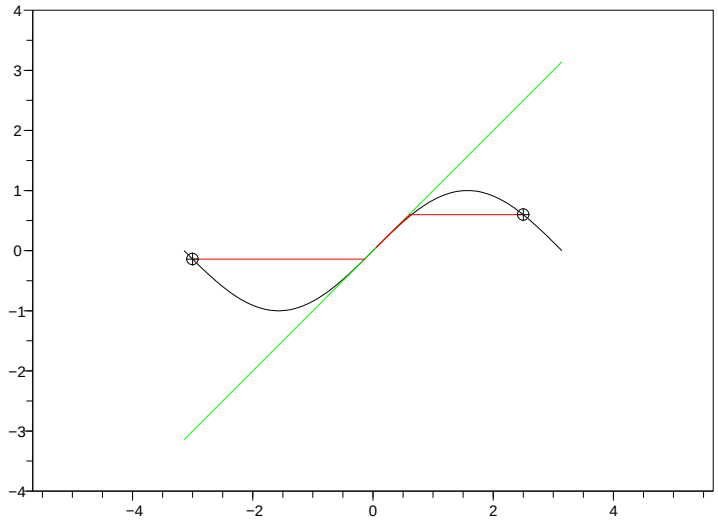
## 10.10 Successione generata da una funzione (10)




```
//:succ10.sce
//
// Successione generata da una funzione (10)
//
function y=f(x), y=sin(x), endfunction
function yp=fp(x), dx=0.01, yp=(f(x+dx)-f(x))/dx, endfunction
x=[-60:60]/120*2*%pi;
clf();
plot2d(0,0,1,"031")
plot2d(x,f(x))
plot2d(x,x,3)
N=8;
a0=1;
a=[a0 0];
for i=1:N-1
      a(i+1)=f(a(i));
end
plot2d2(a,f(a),5)
plot2d(a(1),a(2),-3)
//.
```

[Download]

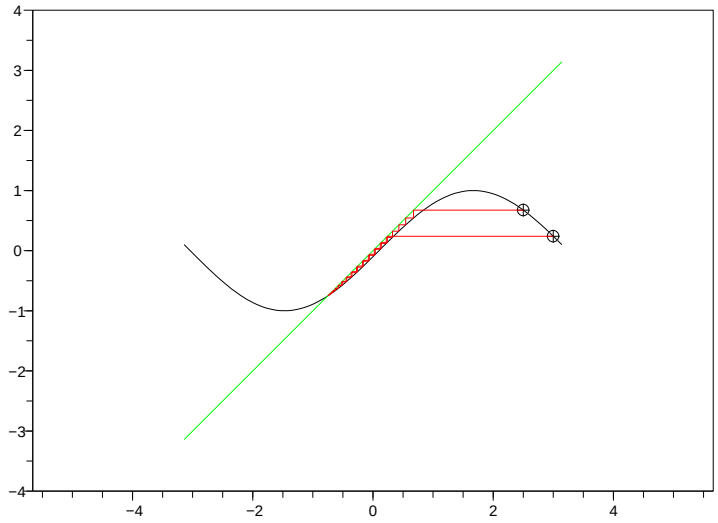
## 10.11 Successione generata da una funzione (11)




```
//:succ11.sce
//
// Successione generata da una funzione (11)
//
function y=f(x), y=sin(x), endfunction
function yp=fp(x), dx=0.01, yp=(f(x+dx)-f(x))/dx, endfunction
x=[-60:60]/120*2*%pi;
clf();
plot2d(0,0,1,"031")
plot2d(x,f(x))
plot2d(x,x,3)
N=1000;
a0=2.5;
a=[a0 0];
for i=1:N-1
      a(i+1)=f(a(i));
end
plot2d2(a,f(a),5)
plot2d(a(1),a(2),-3)
N=10;
a0=-3;
a=[a0 0];
for i=1:N-1
      a(i+1)=f(a(i));
end
plot2d2(a,f(a),5)
plot2d(a(1),a(2),-3)
//.
```

[Download]

## 10.12 Successione generata da una funzione (12)




```
//:succ12.sce
//
// Successione generata da una funzione (12)
//
function y=f(x), y=sin(x-0.1), endfunction
function yp=fp(x), dx=0.01, yp=(f(x+dx)-f(x))/dx, endfunction
x=[-60:60]/120*2*%pi;
clf();
plot2d(0,0,1,"031")
plot2d(x,f(x))
plot2d(x,x,3)
//
N=20;
a0=2.5;
a=[a0 0];
for i=1:N-1
      a(i+1)=f(a(i));
end
plot2d(a(1),a(2),-3)
plot2d2(a,f(a),5)
//
N=20;
a0=3;
a=[a0 0];
for i=1:N-1
      a(i+1)=f(a(i));
end
plot2d(a(1),a(2),-3)
plot2d2(a,f(a),5)
//.
```

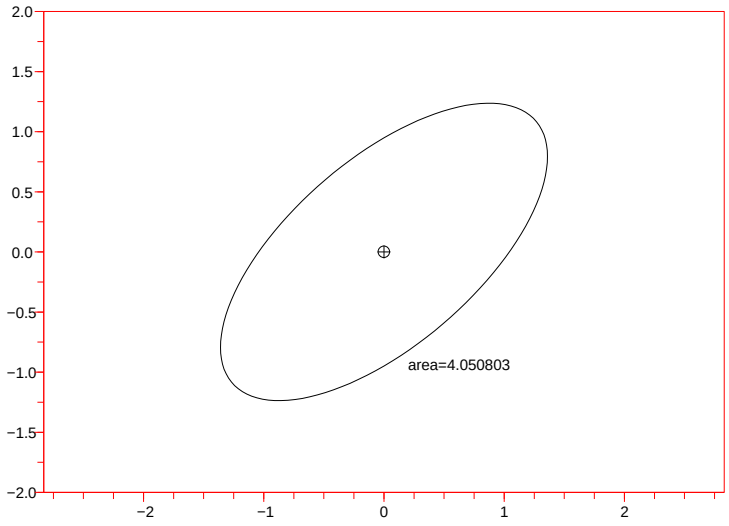
[Download]

# Capitolo 11

# Dischi deformati

Si possono ottenere figure bizzarre semplicemente deformando un cerchio. Si vede come fare questo con una funzione che trasforma le coordinate. Dopo aver generato una nuova figura se ne calcola l'area e il baricentro.

## 11.1 Deformazione affine di un disco

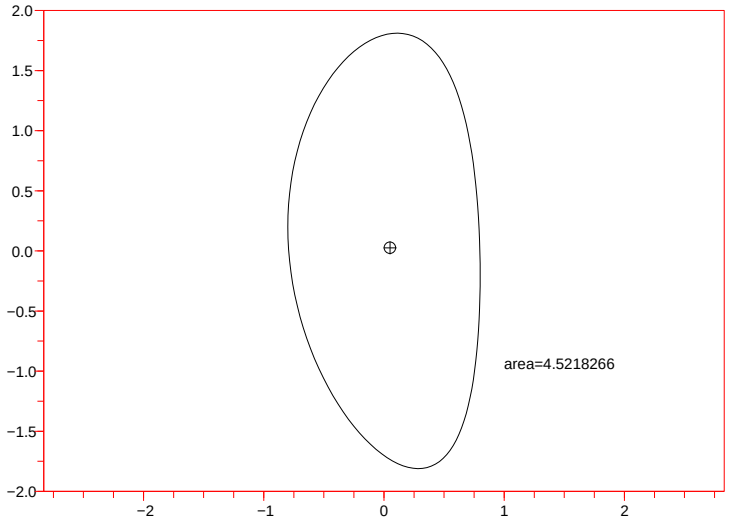



```
//:defda.sce
//
// Deformazione affine di un disco
//
function x=fx(t), x=cos(t), endfunction;
function y=fy(t), y=sin(t), endfunction;
function pt=phi(p), pt= [1.1, -0.8 ; 1.2, 0.3]*p, endfunction;
t=-[0:120]/120*2*%pi;
fig1=[fx(t);fy(t)];
fig2=fig1;
for i=1:length(fig1(1,:)), fig2(:,i)=phi(fig1(:,i)); end;
//
// Area e baricentro (tratto da baC1.sce)
//
x=fig2(1,:);
y=fig2(2,:);
area=0;
xg=0;
yg=0;
for i=1:(length(x)-1)
      i2=i+1;
      i1=i;
      dx=x(i2)-x(i1);
      dy=y(i2)-y(i1);
      xm=(x(i2)+x(i1))/2;
      ym=(y(i2)+y(i1))/2;
      area=area+(ym*dx-xm*dy)/2;
      xg=xg +(xm*ym*dx-xm^2*dy)/3;
      yg=yg +(ym^2*dx-xm*ym*dy)/3;
end;
xg=xg/area
yg=yg/area
clf();
xset("color",5);
plot2d(xg,yg,-3,"031"," ",[-2,-2,2,2]);
```

```
plot2d(fig2(1,:),fig2(2,:));
xstring(0.2,-1,"area="+string(area))
//.
```

[Download]

## 11.2 Deformazione non affine di un disco (1)

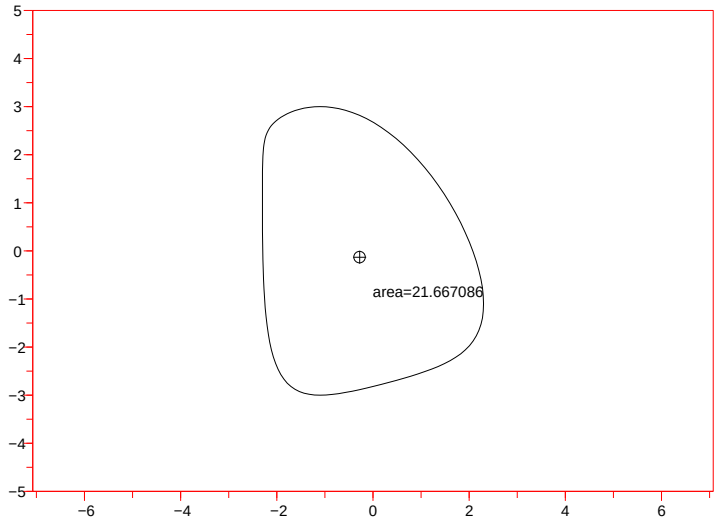



```
//:defd1.sce
//
// Deformazione non affine di un disco (1)
//
function x=fx(t), x=cos(t), endfunction;
function y=fy(t), y=sin(t), endfunction;
function pt=phi(p), pt= [0.2*p(1), -0.8 ; 1.8, 0.2]*p, endfunction;
t=-[0:120]/120*2*%pi;
fig1=[fx(t);fy(t)];
fig2=fig1;
for i=1:length(fig1(1,:)), fig2(:,i)=phi(fig1(:,i)); end;
//
// Area e baricentro
//
x=fig2(1,:);
y=fig2(2,:);
area=0;
xg=0;
yg=0;
for i=1:(length(x)-1)
      i2=i+1;
      i1=i;
      dx=x(i2)-x(i1);
      dy=y(i2)-y(i1);
      xm=(x(i2)+x(i1))/2;
      ym=(y(i2)+y(i1))/2;
      area=area+(ym*dx-xm*dy)/2;
      xg=xg +(xm*ym*dx-xm^2*dy)/3;
      yg=yg +(ym^2*dx-xm*ym*dy)/3;
end;
xg=xg/area
yg=yg/area
clf();
xset("color",5);
plot2d(xg,yg,-3,"031"," ",[-2,-2,2,2]);
```

```
plot2d(fig2(1,:),fig2(2,:));
xstring(1,-1,"area="+string(area))
//.
```

[Download]

## 11.3 Deformazione non affine di un disco (2)

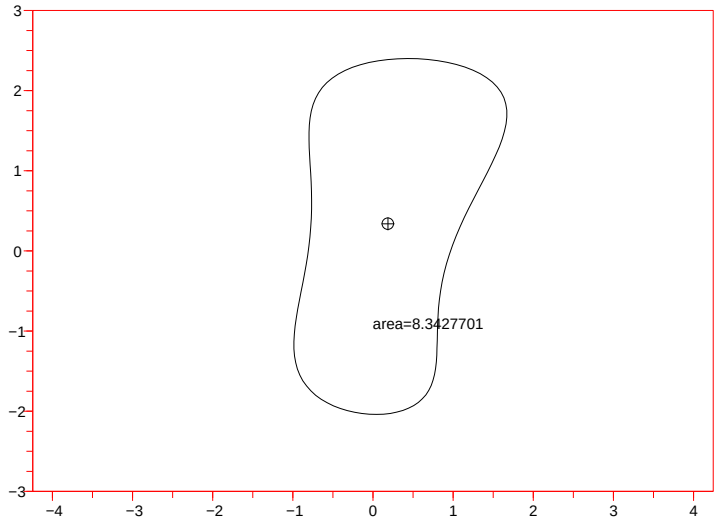



```
//:defd2.sce
//
// Deformazione non affine di un disco (2)
//
function x=fx(t), x=cos(t), endfunction;
function y=fy(t), y=sin(t), endfunction;
function pt=phi(p), pt= [-1.1*p(1), -2.3 ;3, 1.1*p(2)^2]*p, endfunction;
t=-[0:120]/120*2*%pi;
fig1=[fx(t);fy(t)];
fig2=fig1;
for i=1:length(fig1(1,:)), fig2(:,i)=phi(fig1(:,i)); end;
//
// Area e baricentro
//
x=fig2(1,:);
y=fig2(2,:);
area=0;
xg=0;
yg=0;
for i=1:(length(x)-1)
      i2=i+1;
      i1=i;
      dx=x(i2)-x(i1);
      dy=y(i2)-y(i1);
      xm=(x(i2)+x(i1))/2;
      ym=(y(i2)+y(i1))/2;
      area=area+(ym*dx-xm*dy)/2;
      xg=xg +(xm*ym*dx-xm^2*dy)/3;
      yg=yg +(ym^2*dx-xm*ym*dy)/3;
end;
xg=xg/area
yg=yg/area
clf();
xset("color",5);
plot2d(xg,yg,-3,"031"," ",[-5,-5,5,5]);
```

```
plot2d(fig2(1,:),fig2(2,:));
xstring(0,-1,"area="+string(area))
//.
```

[Download]

## 11.4 Deformazione non affine di un disco (3)

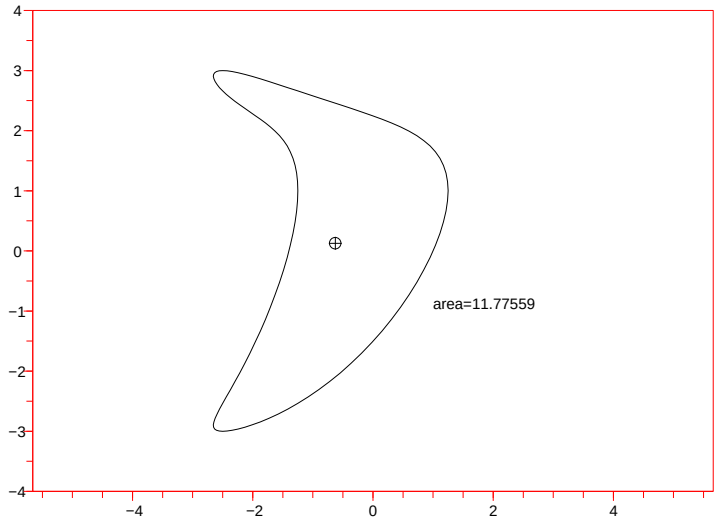



```
//:defd3.sce
//
// Deformazione non affine di un disco (3)
//
function x=fx(t), x=cos(t), endfunction;
function y=fy(t), y=sin(t), endfunction;
function pt=phi(p), pt= [1.2*p(1)^2, -0.8 ; 1.8, 1.3]*p, endfunction;
t=-[0:120]/120*2*%pi;
fig1=[0.1+fx(t);fy(t)];
fig2=fig1;
for i=1:length(fig1(1,:)), fig2(:,i)=phi(fig1(:,i)); end;
//
// Area e baricentro (tratto da baC1.sce)
//
x=fig2(1,:);
y=fig2(2,:);
area=0;
xg=0;
yg=0;
for i=1:(length(x)-1)
      i2=i+1;
      i1=i;
      dx=x(i2)-x(i1);
      dy=y(i2)-y(i1);
      xm=(x(i2)+x(i1))/2;
      ym=(y(i2)+y(i1))/2;
      area=area+(ym*dx-xm*dy)/2;
      xg=xg +(xm*ym*dx-xm^2*dy)/3;
      yg=yg +(ym^2*dx-xm*ym*dy)/3;
end;
xg=xg/area
yg=yg/area
clf();
xset("color",5);
plot2d(xg,yg,-3,"031"," ",[-3,-3,3,3]);
```

```
plot2d(fig2(1,:),fig2(2,:));
xstring(0,-1,"area="+string(area))
//.
```

[Download]

## 11.5 Deformazione non affine di un disco (4)

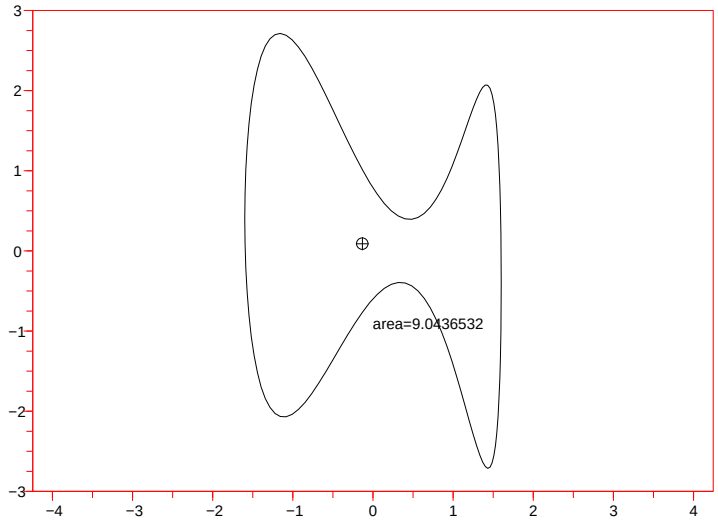



```
//:defd4.sce
//
// Deformazione non affine di un disco (4)
//
function x=fx(t), x=cos(t), endfunction;
function y=fy(t), y=sin(t), endfunction;
function pt=phi(p), pt= [-5/2*p(1), -5/4 ; 3, p(2)^3]*p, endfunction;
t=-[0:120]/120*2*%pi;
fig1=[fx(t);fy(t)];
fig2=fig1;
for i=1:length(fig1(1,:)), fig2(:,i)=phi(fig1(:,i)); end;
//
// Area e baricentro
//
x=fig2(1,:);
y=fig2(2,:);
area=0;
xg=0;
yg=0;
for i=1:(length(x)-1)
      i2=i+1;
      i1=i;
      dx=x(i2)-x(i1);
      dy=y(i2)-y(i1);
      xm=(x(i2)+x(i1))/2;
      ym=(y(i2)+y(i1))/2;
      area=area+(ym*dx-xm*dy)/2;
      xg=xg +(xm*ym*dx-xm^2*dy)/3;
      yg=yg +(ym^2*dx-xm*ym*dy)/3;
end;
xg=xg/area
yg=yg/area
clf();
xset("color",5);
plot2d(xg,yg,-3,"031"," ",[-4,-4,4,4]);
```

```
plot2d(fig2(1,:),fig2(2,:));
xstring(1,-1,"area="+string(area))
//.
```

[Download]

## 11.6 Deformazione non affine di un disco (5)

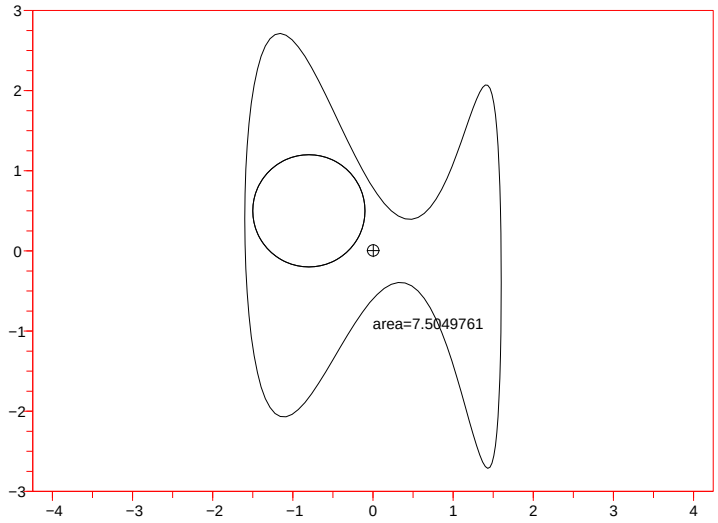



```
//:defd5.sce
//
// Deformazione non affine di un disco (5)
//
function x=fx(t), x=cos(t), endfunction;
function y=fy(t), y=sin(t), endfunction;
function pt=phi(p), pt= [0.2*p(1), -0.8; 0.2+2.8*p(2)^2, 0.2]*2*p, endfunction;
t=-[0:120]/120*2*%pi;
fig1=[fx(t);fy(t)];
fig2=fig1;
for i=1:length(fig1(1,:)), fig2(:,i)=phi(fig1(:,i)); end;
//
// Area e baricentro
//
x=fig2(1,:);
y=fig2(2,:);
area=0;
xg=0;
yg=0;
for i=1:(length(x)-1)
      i2=i+1;
      i1=i;
      dx=x(i2)-x(i1);
      dy=y(i2)-y(i1);
      xm=(x(i2)+x(i1))/2;
      ym=(y(i2)+y(i1))/2;
      area=area+(ym*dx-xm*dy)/2;
      xg=xg +(xm*ym*dx-xm^2*dy)/3;
      yg=yg +(ym^2*dx-xm*ym*dy)/3;
end;
xg=xg/area
yg=yg/area
clf();
xset("color",5);
plot2d(xg,yg,-3,"031"," ",[-3,-3,3,3]);
```

```
plot2d(fig2(1,:),fig2(2,:));
xstring(0,-1,"area="+string(area))
//.
```

[Download]

## 11.7 Deformazione non affine di un disco (6)




```
//:defd6.sce
//
// Deformazione non affine di un disco (6)
//
function x=fx(t), x=cos(t), endfunction;
function y=fy(t), y=sin(t), endfunction;
function pt=phi(p), pt= [0.2*p(1), -0.8; 0.2+2.8*p(2)^2, 0.2]*2*p, endfunction;
function xc=fxc(x0,r,t), xc=x0+r*cos(t), endfunction;
function yc=fyc(y0,r,t), yc=y0+r*sin(t), endfunction;
t=-[0:120]/120*2*%pi;
fig1=[fx(t);fy(t)];
fig2=fig1;
for i=1:length(fig1(1,:)), fig2(:,i)=phi(fig1(:,i)); end;
foro=[fxc(-0.8,0.7,-t);fyc(0.5,0.7,-t)] // l'orientamento  opposto
//
// Area e baricentro
//
x=fig2(1,:);
y=fig2(2,:);
area=0;
xg=0;
yg=0;
for i=1:(length(x)-1)
      i2=i+1;
      i1=i;
      dx=x(i2)-x(i1);
      dy=y(i2)-y(i1);
      xm=(x(i2)+x(i1))/2;
      ym=(y(i2)+y(i1))/2;
      area=area+(ym*dx-xm*dy)/2;
      xg=xg +(xm*ym*dx-xm^2*dy)/3;
      yg=yg +(ym^2*dx-xm*ym*dy)/3;
end;
// foro 1
x=foro(1,:);
```

```
y=foro(2,:);
plot2d(x,y);
for i=1:(length(x)-1)
      i2=i+1;
      i1=i;
      dx=x(i2)-x(i1);
      dy=y(i2)-y(i1);
      xm=(x(i2)+x(i1))/2;
      ym=(y(i2)+y(i1))/2;
      area=area+(ym*dx-xm*dy)/2;
      xg=xg +(xm*ym*dx-xm^2*dy)/3;
      yg=yg +(ym^2*dx-xm*ym*dy)/3;
end;
xg=xg/area
yg=yg/area
//clf();
xset("color",5);
plot2d(xg,yg,-3,"031"," ",[-3,-3,3,3]);
plot2d(fig2(1,:),fig2(2,:));
plot2d(foro(1,:),foro(2,:));
xstring(0,-1,"area="+string(area))
//.
```

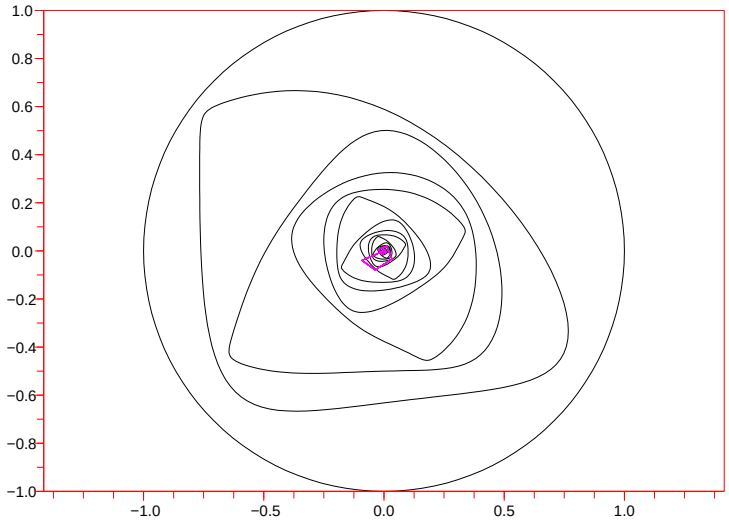
[Download]

# Capitolo 12

# Mappe di contrazione

Si applica ripetutamente una mappa (deformazione) ad un disco e si cerca di intuire se esiste un punto fisso costruendo la successione di dischi deformati. Si genera anche la successione delle aree e dei baricentri.

## 12.1 Mappa di contrazione di un disco (1) (convergente)

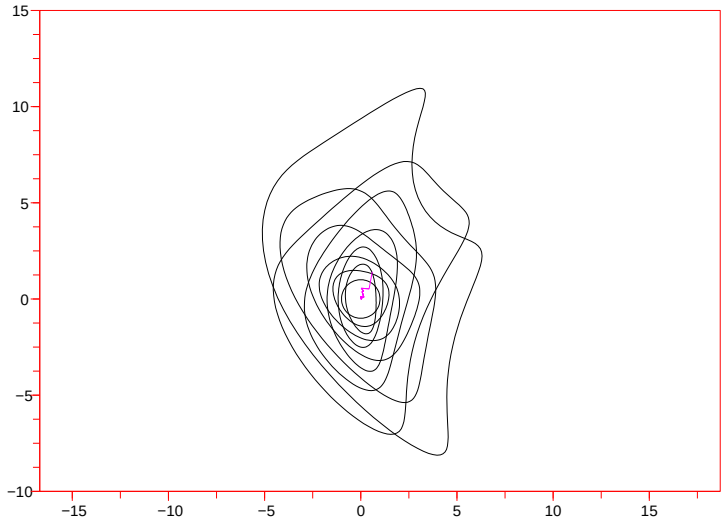



```
//:contr01.sce
//
// Mappa di contrazione di un disco (1) (convergente)
//
function x=fx(t), x=cos(t), endfunction;
function y=fy(t), y=sin(t), endfunction;
function pt=phi(p), pt= [-1.1*p(1), -2.3 ;2, p(2)^2]*p/3; endfunction;
function fig2=fdef(fig1)// Deformazione
      for i=1:length(fig1(1,:)),
            fig2(:,i)=phi(fig1(:,i));
      end;,
endfunction;
function [xg,yg,area]=ga(fig)// Area e baricentro
      x=fig(1,:);
      y=fig(2,:);
      area=0;
      xg=0;
      yg=0;
      for i=1:(length(x)-1)
            i2=i+1;
            i1=i;
            dx=x(i2)-x(i1);
            dy=y(i2)-y(i1);
            xm=(x(i2)+x(i1))/2;
            ym=(y(i2)+y(i1))/2;
            area=area+(ym*dx-xm*dy)/2;
            xg=xg +(xm*ym*dx-xm^2*dy)/3;
            yg=yg +(ym^2*dx-xm*ym*dy)/3;
      end;
      xg=xg/area;
      yg=yg/area;
endfunction
t=-[0:120]/120*2*%pi;
fig1=[fx(t);fy(t)];
clf();
```

```
xset("color",5);
plot2d(0,0,1,"031"," ");
plot2d(fx(t),fy(t))
k=1;
[xg, yg, area]=ga(fig1);
xgc=[xg 0];
ygc=[yg 0];
areac=[area 0];
fig2=fig1;
//
for n=1:20
// begin loop
xclick();
//
fig2=fdef(fig1);
[xg, yg, area]=ga(fig2);
k=k+1;
xgc(k)=xg;
ygc(k)=yg;
areac(k)=area;
plot2d(fig2(1,:),fig2(2,:));
plot2d(xgc,ygc,6)
fig1=fig2;
//
// end loop
end
//
[xgc',ygc']
areac'
//.
```

[Download]

## 12.2 Mappa di contrazione di un disco (2) (divergente)

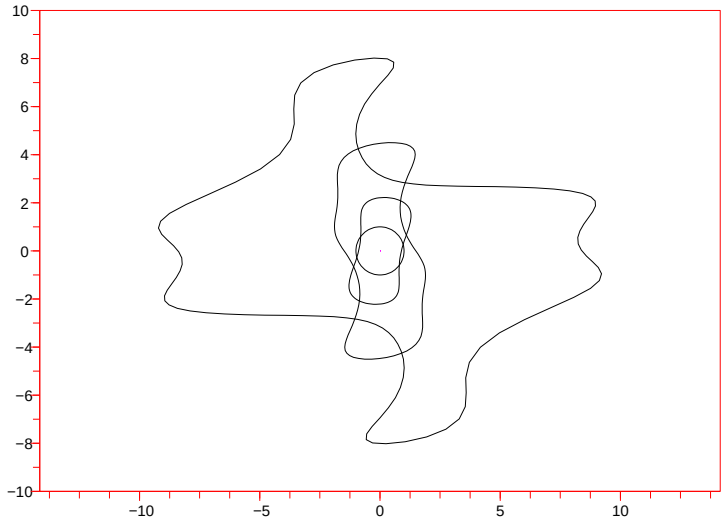



```
//:contr02.sce
//
// Mappa di contrazione di un disco (2) (divergente)
//
function x=fx(t), x=cos(t), endfunction;
function y=fy(t), y=sin(t), endfunction;
function pt=phi(p), pt= [0.2*p(1), -0.8 ; 1.8, 0.2]*p, endfunction;
function fig2=fdef(fig1)// Deformazione
      for i=1:length(fig1(1,:)),
            fig2(:,i)=phi(fig1(:,i));
      end;,
endfunction;
function [xg,yg,area]=ga(fig)// Area e baricentro
      x=fig(1,:);
      y=fig(2,:);
      area=0;
      xg=0;
      yg=0;
      for i=1:(length(x)-1)
            i2=i+1;
            i1=i;
            dx=x(i2)-x(i1);
            dy=y(i2)-y(i1);
            xm=(x(i2)+x(i1))/2;
            ym=(y(i2)+y(i1))/2;
            area=area+(ym*dx-xm*dy)/2;
            xg=xg +(xm*ym*dx-xm^2*dy)/3;
            yg=yg +(ym^2*dx-xm*ym*dy)/3;
      end;
      xg=xg/area;
      yg=yg/area;
endfunction
t=-[0:120]/120*2*%pi;
fig1=[fx(t);fy(t)];
clf();
```

```
xset("color",5);
plot2d(0,0,1,"031"," ");
plot2d(fx(t),fy(t))
k=1;
[xg, yg, area]=ga(fig1);
xgc=[xg 0];
ygc=[yg 0];
areac=[area 0];
fig2=fig1;
//
for n=1:10
// begin loop
xclick();
//
fig2=fdef(fig1);
[xg, yg, area]=ga(fig2);
k=k+1;
xgc(k)=xg;
ygc(k)=yg;
areac(k)=area;
plot2d(fig2(1,:),fig2(2,:));
plot2d(xgc,ygc,6)
fig1=fig2;
//
// end loop
end
//
[xgc',ygc']
areac'
//.
```

[Download]

## 12.3 Mappa di contrazione di un disco (3) (divergente)

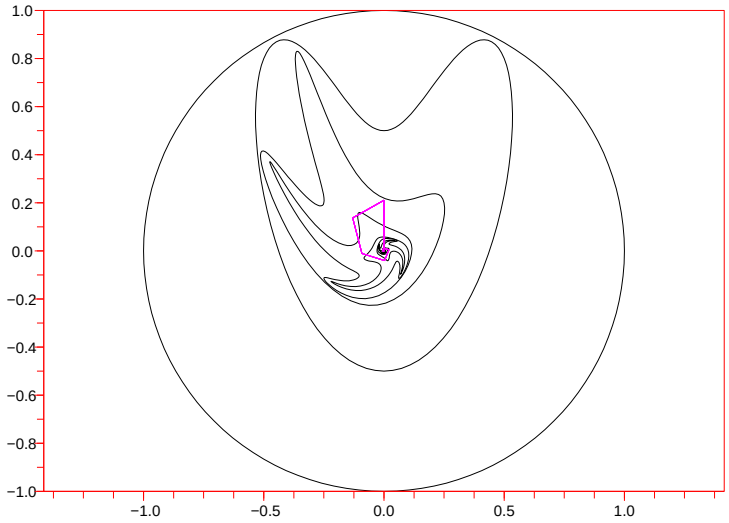



```
//:contr03.sce
//
// Mappa di contrazione di un disco (3) (divergente)
//
function x=fx(t), x=cos(t), endfunction;
function y=fy(t), y=sin(t), endfunction;
function pt=phi(p), pt= [1.2*p(1)^2, -0.8 ; 1.8, 1.3]*p, endfunction;
function fig2=fdef(fig1)// Deformazione
      for i=1:length(fig1(1,:)),
            fig2(:,i)=phi(fig1(:,i));
      end;,
endfunction;
function [xg,yg,area]=ga(fig)// Area e baricentro
      x=fig(1,:);
      y=fig(2,:);
      area=0;
      xg=0;
      yg=0;
      for i=1:(length(x)-1)
            i2=i+1;
            i1=i;
            dx=x(i2)-x(i1);
            dy=y(i2)-y(i1);
            xm=(x(i2)+x(i1))/2;
            ym=(y(i2)+y(i1))/2;
            area=area+(ym*dx-xm*dy)/2;
            xg=xg +(xm*ym*dx-xm^2*dy)/3;
            yg=yg +(ym^2*dx-xm*ym*dy)/3;
      end;
      xg=xg/area;
      yg=yg/area;
endfunction
t=-[0:120]/120*2*%pi;
fig1=[fx(t);fy(t)];
clf();
```

```
xset("color",5);
plot2d(0,0,1,"031"," ");
plot2d(fx(t),fy(t))
k=1;
[xg, yg, area]=ga(fig1);
xgc=[xg 0];
ygc=[yg 0];
areac=[area 0];
fig2=fig1;
//
for n=1:3
// begin loop
xclick();
//
fig2=fdef(fig1);
[xg, yg, area]=ga(fig2);
k=k+1;
xgc(k)=xg;
ygc(k)=yg;
areac(k)=area;
plot2d(fig2(1,:),fig2(2,:));
plot2d(xgc,ygc,6)
fig1=fig2;
//
// end loop
end
//
[xgc',ygc']
areac'
//.
```

[Download]

## 12.4 Mappa di contrazione di un disco (4) (convergente)




```
//:contr04.sce
//
// Mappa di contrazione di un disco (4) (convergente)
//
function x=fx(t), x=cos(t), endfunction;
function y=fy(t), y=sin(t), endfunction;
function pt=phi(p), pt= [p(1)*p(2)/5- p(2)/2 ;0.5*p(1)+ 0.8*p(2)^2], endfunction;
function fig2=fdef(fig1)// Deformazione
      for i=1:length(fig1(1,:)),
            fig2(:,i)=phi(fig1(:,i));
      end;,
endfunction;
function [xg,yg,area]=ga(fig)// Area e baricentro
      x=fig(1,:);
      y=fig(2,:);
      area=0;
      xg=0;
      yg=0;
      for i=1:(length(x)-1)
            i2=i+1;
            i1=i;
            dx=x(i2)-x(i1);
            dy=y(i2)-y(i1);
            xm=(x(i2)+x(i1))/2;
            ym=(y(i2)+y(i1))/2;
            area=area+(ym*dx-xm*dy)/2;
            xg=xg +(xm*ym*dx-xm^2*dy)/3;
            yg=yg +(ym^2*dx-xm*ym*dy)/3;
      end;
      xg=xg/area;
      yg=yg/area;
endfunction
t=-[0:120]/120*2*%pi;
fig1=[fx(t);fy(t)];
clf();
```

```
xset("color",5);
plot2d(0,0,1,"031"," ");
plot2d(fx(t),fy(t))
k=1;
[xg, yg, area]=ga(fig1);
xgc=[xg 0];
ygc=[yg 0];
areac=[area 0];
fig2=fig1;
//
for n=1:10
// begin loop
xclick();
//
fig2=fdef(fig1);
[xg, yg, area]=ga(fig2);
k=k+1;
xgc(k)=xg;
ygc(k)=yg;
areac(k)=area;
plot2d(fig2(1,:),fig2(2,:));
plot2d(xgc,ygc,6)
fig1=fig2;
//
// end loop
end
//
[xgc',ygc']
areac'
//.
```

[Download]

# Parte II

# Piccolo manuale Scilab

# Capitolo 13

# Scilab

Sono qui illustrati i concetti fondamentali e i costrutti essenziali del linguaggio con l'obiettivo di permettere la comprensione dei moduli presentati nella parte I e di facilitare l'apprendimento autonomo. `Scilab` è un linguaggio complesso che permette applicazioni di calcolo e visualizzazione anche in ambiti molto diversi da quello di questo corso.

## 13.1 Come introdurre i comandi

Supponiamo di aver avviato `Scilab`. Nella finestra `scilex` dovrebbe comparire a sinistra una freccia

```
-->
```

a fianco della quale lampeggia il cursore. Qui va introdotto un comando a cui `Scilab` risponde in genere con

```
ans =
```

Un comando può consistere semplicemente in una espressione numerica. Se si digita ad esempio

```
2*5
```

si ottiene in risposta

```
ans =
   10.
```

## 13.2 Le espressioni numeriche

L'espressione

$$7\frac{2}{3}+\sqrt{5}-2^3$$

si scrive

```
7*2/3+sqrt(5)-2^3
```

Si ottiene in risposta

```
ans =
 - 1.0972654
```

Le parentesi tonde servono in generale, oltre che a racchiudere gli argomenti di una funzione, a delimitare parti di espressioni, anche a più livelli:

```
(b+sqrt(b^2-4*a*c))/(2*a)
```

In generale è consigliabile usare delle parentesi anche quando non sembrano necessarie ma non si è sicuri che l'ordine di valutazione sarà quello voluto. Non è possibile utilizzare a questo scopo parentesi quadre o parentesi graffe.

## 13.3 Variabili e assegnazioni

Il nome di una variabile può essere una parola, con una lettera come primo carattere, formata da lettere e numeri. Ad una variabile si assegna un valore nel seguente modo:

```
a=5
b1=3*a+2
```

Il valore assegnato deve essere un numero o un'espressione che abbia un valore numerico. Non è ammesso ad esempio assegnare alla variabile `b1` l'espressione:

```
-->b1=3*c+2
       !--error 4
undefined variable : c
```

senza aver prima assegnato un valore alla variabile `c`.

Il comando `clear` cancella tutte le assegnazioni fatte rendendo non definite tutte le variabili introdotte.

Occorre fare attenzione a non confondere lo zero con la o maiuscola (corrispondono infatti a tasti diversi sulla tastiera), come pure a distinguere lettere maiuscole e lettere minuscole.

## 13.4 Costanti predefinite

Esistono delle costanti predefinite, quali:

| | |
|---|---|
| `%pi` | $\pi$ |
| `%e` | la costante di Nepero |
| `%i` | l'unità immaginaria |

che possono essere utilizzate in una qualsiasi espressione, come ad esempio:

```
-->a=2*%pi
 a  =
    6.2831853
```

## 13.5 Stringhe di caratteri

In una variabile è possibile memorizzare anche delle parole o delle brevi frasi. L'assegnazione avviene scrivendo una succesione di caratteri (*stringa*) tra apici o doppi apici:

```
str='Punto'
```

Due, o più, stringhe possono essere composte utilizzando il segno `+` come simbolo dell'operazione di *concatenazione*, nel seguente modo:

```
str1='Retta'
str2=str1+' tangente'
```

ottenendo

```
str2  =
Retta tangente
```

Occorre ricordare che una stringa è una cosa diversa dal nome di una variabile. Le stringhe sono utili, ad esempio, per scrivere didascalie sui grafici.

## 13.6 Le n-ple

Un insieme ordinato di numeri si indica racchiudendo tra parentesi quadre un elenco di numeri separati da spazi o virgole:

```
[2 5 3 2 1]
[2, 5, 3, 2, 1]
```

Una $n$-pla può essere assegnata ad una variabile:

```
a=[1 2 3 4 5 6]
```

### 13.6.1 Generazione automatica

Si può generare automaticamente una $n$-pla incrementando di una unità l'estremo sinistro di un intervallo assegnato. Ad esempio con

```
[1:6]
```

si ottiene una $n$-pla uguale alla precedente. Si può assegnare un incremento diverso da 1, ad esempio 0.5, con

```
[1:0.5:6]
```

ottenendo

```
 ans  =
    1.    1.5    2.    2.5    3.    3.5    4.    4.5    5.    5.5    6.
```

Si noti che con un incremento 0.6 si ottiene, nell'intervallo $[1, 6]$,

```
-->[1:0.6:6]
 ans  =
    1.    1.6    2.2    2.8    3.4    4.    4.6    5.2    5.8
```

### 13.6.2 Operazioni

Una $n$-pla può essere modificata attraverso un'operazione con un numero:

```
x=[2 5 3 2 1]
x*2
x+3
x/5
x^2
```

Queste operazioni sono distribuite su ciascun elemento della $n$-pla. Si possono eseguire le operazioni di addizione e sottrazione tra due $n$-ple *termine a termine* come in:

```
x1=[2 5 3 2 1]
x2=[1 2 3 1 2]
x1+x2
x1-x2
```

Naturalmente le due $n$-ple devono avere lo stesso numero di termini. Per eseguire le operazioni di moltiplicazione e divisione *termine a termine* tra due $n$-ple occorre far precedere i simboli `*` e `/` da un punto:

```
x1=[2 5 3 2 1]
x2=[1 2 3 1 2]
x1+x2
x1-x2
x1.*x2
x1./x2
```

Altri esempi di operazioni *termine a termine* sono i seguenti:

```
x1=[2 5 3 2 1]
x2=[1 2 3 1 2]
(x1+3)+x2
x1^2-x2
(x1^2 +x1 -1).*x2
1 ./ x2
```

Notare in particolare l'ultima espressione. È facile commettere l'errore di scriverla senza lo spazio prima del punto:

```
1./ x2
```

In tal caso il punto viene interpretato come punto decimale di `1` e l'espressione indicata è equivalente a:

```
1.0/ x2
```

che ha un significato completamente diverso che non ha interesse considerare qui.

### 13.6.3 I singoli elementi

Un elemento di una $n$-pla è individuato dalla sua posizione:

```
-->x1=[2 5 3 2 1]
 x1  =
    2.    5.    3.    2.    1.
-->x1(2)
 ans  =
    5.
```

È possibile modificare il valore di un solo elemento di una $n$-pla:

```
-->x1=[2 5 3 2 1]
 x1  =
    2.    5.    3.    2.    1.
-->x1(2)=7
 x1  =
    2.    7.    3.    2.    1.
```

Una parte di una $n$pla è individuata da una coppia di indici separati dai due punti, come in questo esempio:

```
-->x1=[2 5 3 2 1]
 x1  =
    2.    5.    3.    2.    1.
-->x1(2:4)
 ans  =
    5.    3.    2.
```

Come indice dell'ultimo elemento di una $n$-pla si può usare un \$, secondo l'esempio seguente

```
-->x1($)
 ans  =
    1.
-->x1(2:$)
 ans  =
    5.    3.    2.    1.
```

## 13.7 Matrici e vettori

Matrici in $\mathbb{R}^{m\times n}$ e vettori in $\mathbb{R}^n$, come

$$\begin{pmatrix} 1 & 2 \\ -2 & 10 \end{pmatrix}, \begin{pmatrix} 3 \\ 7 \end{pmatrix}$$

si rappresentano nel seguente modo:

```
mat=[1 2; -2 10]
vec1=[3; 7]
```

Il punto e virgola separa le righe di una matrice. Gli elementi di una riga possono essere separati da spazi o da virgole. Notare che un vettore è rappresentato come una matrice con una sola colonna (*questa struttura è utilizzata nei moduli della parte I sia per le coppie di coordinate che per le coppie di componenti di vettori*).

### 13.7.1 Operazioni

Il prodotto tra matrici

$$\begin{pmatrix} 1 & 2 & 1 \\ -2 & 10 & 3 \end{pmatrix} \begin{pmatrix} 3 & 2 \\ 1 & 4 \\ 5 & 2 \end{pmatrix}$$

si indica così:

```
-->mat1=[1 2 1; -2 10 3]
 mat1  =
    1.    2.     1.
  - 2.    10.    3.
-->mat2=[3 2; 1 4; 5 2]
 mat2  =
    3.    2.
    1.    4.
    5.    2.
```

```
-->mat=mat1*mat2
 mat  =
    10.    12.
    19.    42.
```

### 13.7.2 I singoli elementi

Un elemento di una matrice è individuato da una coppia di indici separati da una virgola:

```
-->mat1(2,1)
 ans  =
  - 2.
```

Un'intera riga o un'intera colonna sono individuate sostituendo uno degli indici con i due punti:

```
-->mat1(2,:)
 ans  =
  - 2.    10.    3.
-->mat1(:,1)
 ans  =
    1.
  - 2.
```

Notare che è sbagliato separare gli indici con un punto e virgola:

```
-->mat1(2;3)
         !--error 3
waiting for right parenthesis
```

### 13.7.3 Matrice trasposta

```
La matrice trasposta si ottiene con un apice:
-->mat1
 mat1  =
    1.    2.     1.
  - 2.    10.    3.

-->mat1'
 ans  =
    1.  - 2.
    2.    10.
    1.    3.
```

Funziona anche per una matrice di una sola colonna:

```
-->vec1
 vec1  =
    3.
    7.

-->vec1'
 ans  =
    3.    7.
```

## 13.8 Grafici

Il modo più semplice per creare un grafico è quello di definire due $n$-ple, ad esempio `x` e `y`, contenenti rispettivamente le ascisse e le ordinate dei punti da rappresentare, e utilizzare poi la funzione `plot2d`:

```
x=[1 2 3 5]
y=[-1 1 0 2]
plot2d(x,y)
```

Si ottiene il grafico della spezzata che unisce i punti di coordinate $(x, y)$. Le coordinate possono essere disposte a formare le righe di una matrice. In tal caso:

```
sp=[1 2 3 5;-1 1 0 2]
plot2d(sp(1,:),sp(2,:))
```

Un modo ancora diverso per ottenere lo stesso grafico consiste nel costruire la matrice precedente come $n$-pla di coppie di coordinate disposte ciascuna a formare una colonna:

```
sp=[[1;-1],[2;1],[3;0],[5;2]]
plot2d(sp(1,:),sp(2,:))
```

## 13.9 Il comando `plot2d` in dettaglio

Il comando:

```
plot2d(x,y,1,"031"," ",[-2,-2,2,2])
```

genera il grafico della spezzata che unisce i punti le cui coordinate sono in `x` e `y`. Genera inoltre una cornice lungo il bordo del rettangolo con il vertice in basso a sinistra di coordinate $(-2, -2)$ e il vertice in alto a destra di coordinate $(2, 2)$, descritte nella $n$-pla all'ultimo posto, detta `rect`. Il valore del terzo argomento, se è un numero positivo, definisce il colore della linea:

| | |
|---|---|
| 1 | nero |
| 2 | blu |
| 3 | verde |
| 4 | celeste |
| 5 | rosso |
| 6 | viola |
| ... | ... |

Se il valore è zero il grafico è descritto con semplici punti; se è negativo invece dei punti sono utilizzati dei simboli. Ad esempio con:

```
plot2d(x,y,-1,"031"," ",[-2,-2,2,2])
```

il grafico è descritto utilizzando il segno `+`.

Il disegno generato da un comando `plot2d` si sovrappone al precedente. Questo permette, ad esempio, di costruire prima una cornice e poi di generare altre parti di una figura complessa, come dei grafici di diverse spezzate. Per cancellare un disegno già fatto si usa il comando:

```
clf
```

La stringa `"031"` può essere modificata per ottenere diversi effetti. I due caratteri oltre lo zero si chiamano rispettivamente `frameflag` e `axesflag` e possono avere i seguenti valori:

`frameflag`

| | |
|---|---|
| 0 | impiega la scala già definita nel disegno precedente; |
| 1 | l'intervallo è quello definito da `rect`; |
| 2 | l'intervallo coincide con l'intervallo dei valori delle $x$ e delle $y$; |
| 3 | l'intervallo è quello definito da `rect`, la scala è isometrica; |
| 4 | l'intervallo coincide con l'intervallo dei valori delle $x$ e delle $y$, la scala è isometrica. |

`axesflag`

| | |
|---|---|
| 0 | non viene disegnata né la cornice né gli assi; |
| 1 | viene disegnata la cornice `rect` con l'asse delle $y$ a sinistra; |
| 2 | viene disegnata la cornice `rect` senza tacche di graduazione; |
| 3 | sono disegnati gli assi e quello delle $y$ sta a destra; |
| 4 | gli assi passano per il punto centrale di `rect`; |
| 5 | gli assi passano per il punto di coordinate $(0, 0)$. |

Un modo più esplicito di scrivere il comando precedente è:

```
plot2d(x,y,style=-1,frameflag=3,axesflag=1,rect=[-2,-2,2,2])
```

Per controllare le tacche sugli assi si può aggiungere `nax=[nx,Nx, ny,Ny]` in cui `Nx` e `Ny` rappresentano il numero di segni di graduazione principali, rispettivamente sull'asse delle $x$ e delle $y$, mentre `nx` e `ny` sono le suddivisioni secondarie lungo l'asse delle $x$ e delle $y$, cioè il numero di graduazioni secondarie tra due graduazioni principali contigue.

Una legenda si costruisce con:

```
legends(["curva (a)";"curva (b)";"curva (c)"],[5,2 3],opt="lr")
```

La posizione della legenda dipende dal valore di `opt`:

| | |
|---|---|
| `"ur"` | in alto a destra; |
| `"ul"` | in alto a sinistra; |
| `"lr"` | in basso a destra; |
| `"ll"` | in basso a sinistra; |
| `"?"` | il posizionamento è fatto con il mouse. |

Con `xset` si possono definire varie caratteristiche del disegno. Ad esempio con:

```
xset("line style",2);
plot2d(x,y,style=5);
```

si costruisce un grafico utilizzando una linea tratteggiata e colorata. Con

```
xset("color",4)
```

si assegna il colore della cornice e degli assi.

## 13.10 Il costrutto `function` per funzioni e procedure

Nell'esempio seguente:

```
function y=f(x)
   y=x^2+3
endfunction
```

con `f` si indica il *nome* della funzione, con `x` l'*argomento* e con `y` il *valore*. L'unica cosa che conta in questa definizione è il nome della funzione e l'espressione, o le espressioni, che ne forniscono il valore. Invece il nome utilizzato per l'argomento e il nome utilizzato per il valore sono del tutto fittizi. La definizione precedente è infatti equivalente alla seguente:

```
function val=f(arg)
   val=arg^2+3
endfunction
```

La funzione definita può essere valutata in una qualunque espressione:

```
-->a=f(3)
 a  =
    12.
```

Il nome della funzione può essere scelto a piacimento.

Nel caso in cui la variabile `x` sia una $n$-pla, con `f(x)` si genera la $n$-pla dei valori di `f` in corrispondenza di ciascun elemento della $n$-pla `x`:

```
-->x=[1:5]
 x  =
    1.    2.    3.    4.    5.
-->f(x)
 ans  =
    4.    7.    12.    19.    28.
```

Se si vuole utilizzare una funzione in questo modo (applicandola ad una $n$-pla) occorre che le espressioni siano corrette anche nel caso di operazioni termine a termine, come nell'esempio seguente:

```
function y=rec(x)
   y=1 ./ x
endfunction

-->rec(x)
 ans  =
    1.    0.5    0.3333333    0.25    0.2
```

## 13.11 La struttura `if`

Nell'esempio seguente:

```
a=10;
if a>0 then
   ar=sqrt(a)
end
```

il calcolo della radice di `a` viene eseguito solo nel caso in cui `a` abbia un valore positivo. In generale la condizione indicata in questo costrutto può essere espressa in vari modi, ad esempio usando un'espressione

```
a>1+c
a-1>0
```

oppure usando le relazioni $\leq$ e $\geq$

```
a<=2
b>=a
```

La relazione di uguaglianza si scrive così:

```
b==a
```

Le condizioni si possono comporre usando il segno `|` (che sta per la congiunzione *oppure*) o il segno `&` (che sta per la congiunzione *e*):

```
(b==a)&(a>1)
(b<0)|(a>0)
```

Si può anche usare la negazione:

```
~(b==a)
```

(Il carattere `~` si può ottenere in Windows con `AltGr 126`, in Linux con `AltGr ^`).

## 13.12 La struttura `for`

Nell'esempio seguente:

```
for i=1:5
   i^2+i
end
```

l'espressione sulla seconda riga viene calcolata più volte con tutti i valori di `i` appartenenti alla $n$-pla generata da `[1:5]`. Nel caso:

```
for j=1:2:5
   j^2+j
end
```

vengono considerati tutti i valori di `j` appartenenti alla $n$-pla generata da `[1:2:5]`. Si può scrivere tutto su una linea separando i termini con una virgola:

```
-->for j=1:2:5, j^2+j, end
 ans  =
    2.
 ans  =
    12.
 ans  =
    30.
```

Notare che la $n$-pla di valori da prendere in considerazione può essere qualunque:

```
-->x=[0 -1 2 -5 ]
 x  =
    0.  - 1.    2.  - 5.
-->for xi=x, xi^2+xi, end
 ans  =
    0.
 ans  =
    0.
 ans  =
    6.
 ans  =
    20.
```

## 13.13 La struttura `while`

Nell'esempio seguente:

```
n=4;
fat=1;
while n>1 do
   fat=n*fat;
   n=n-1;
end
fat
```

viene calcolato il fattoriale di 4.

Il punto e virgola alla fine di un'espressione serve ad impedire che il valore venga visualizzato. È particolarmente utile all'interno delle strutture cicliche `for` e `while` in cui di solito ha interesse il valore finale di una o più variabili.

## 13.14 Come riutilizzare le definizioni di funzioni

Le definizioni di funzioni non occasionali possono essere conservate in un file, ad esempio di nome `mie_funzioni.sce`. Per utilizzare tali funzioni basta inserire, nella sezione di definizione delle funzioni al posto di tutte quelle conservate nel file, il comando:

```
getf('mie_funzioni.sce')
```

Per ricordare come si chiamano le funzioni e quali sono i loro argomenti, basta aprire con `scipad` il file `mie_funzioni.sce`.

Se si ricorda il nome di una funzione (ad esempio `ruota`) con:

```
disp(ruota)
```

si visualizza la prima riga della definizione.

## 13.15 Come cercare gli errori

La prima cosa da fare è tentare di interpretare le indicazioni di errore. Se queste non sono sufficienti o incomprensibili:

1. salvare una copia del file aperto con `scipad`
2. riorganizzare il file aperto con `scipad` in modo che compaiano prima tutte e sole le definizioni di funzioni da utilizzare e poi i comandi nell'ordine in cui dovranno essere eseguiti
3. pulire la finestra dei comandi con `clc`
4. pulire la finestra grafica con `clf`
5. cancellare tutte le definizioni già date con `clear`

Dopo aver fatto tutto questo:

1. rieseguire dall'inizio solo poche righe di comandi per volta
2. controllare le indicazioni che compaiono nella finestra dei comandi
3. eventualmente visualizzare il valore di qualche variabile (scrivendone il nome, ad esempio `pol`, seguito dal tasto invio)
4. fare le eventuali modifiche e ripetere l'esecuzione

Eseguendo i comandi un po' per volta occorre tener presente che un ciclo deve essere eseguito per intero, ma si può:

1. fare in modo che venga eseguito poche volte, preferibilmente una sola
2. escludere provvisoriamente dall'esecuzione i comandi interni al ciclo inserendo all'inizio di ciascuna linea i due caratteri //

In generale la regola principale da seguire è procedere nello sviluppo a piccoli passi, aggiungendo poche righe per volta (anche una sola) e controllando l'effetto da esse prodotto nell'esecuzione.

Capitolo 14

# Come creare, modificare e organizzare i moduli scilab

# Capitolo 15

# Come installare Scilab

Per avere una versione aggiornata di `Scilab` e le istruzioni per l'installazione andare al sito ufficiale:
`[Download Scilab]`