

TikZ manent – Appunti per comprendere TikZ

Dott. Giovanni Pagliarini*

22 maggio 2023

Sommario

Questo documento contiene diversi esempi per comprendere le basi del motore di grafica vettoriale TikZ, ed è pensato per essere confrontato con il suo codice sorgente. Il documento è stato scritto per un corso di laboratorio di L^AT_EX avanzato, tenutosi tra Aprile e Maggio 2023 presso il Dipartimento di Matematica e Informatica dell'Università di Ferrara.

1 Intro

- PGF/TikZ è il package L^AT_EX più utilizzato per grafica vettoriale;
- Creato da Till Tantau, PhD Thesis (2003);
- PGF = Portable Graphics Format rappresenta il livello base (*basic layer*);
- TikZ = TikZ ist kein Zeichenprogramm (not a drawing software) rappresenta il livello superiore (*high-level macros*);
- Pacchetto L^AT_EX: `\usepackage{tikz}`;
- Alternative: Inkscape, Cabri, Matlab.

PRO:

- Vettoriale;
- Tanti livelli di personalizzazione;
- Permette portabilità e uniformità stilistica.

CONTRO:

- Non immediato, curva di apprendimento ripida (o, meglio, piana);
- Come L^AT_EX, non è WYSIWYG.

2 Riferimenti al manuale ufficiale


- Versione web: <https://tikz.dev/>;
- Versione pdf: <https://pgf-tikz.github.io/pgf/pgfmanual.pdf>.

Questo documento rimanda più volte a sezioni della parte III del manuale, che spiega le basi di questo sottolinguaggio del L^AT_EX.

*email: giovanni.pagliarini@aol.com, website: <https://giopaglia.github.io/>

2.1 Ambiente `tikzpicture`

L'ambiente `tikzpicture` crea un canovaccio dove si possono digitare comandi `TikZ`. Una volta compilati i comandi, una scatola viene ritagliata attorno a ciò che è stato disegnato, e l'immagine viene prodotta. Per

esempio `\draw (0,0) - (1,1);` produce:  Per maggiore chiarezza, al posto di `tikzpicture`, in questo documento si utilizzerà un ambiente personalizzato che internamente usa `tikzpicture`. L'ambiente si chiama `giotikzpicture`, ed è definito nel preambolo del codice sorgente di questo documento; centra, racchiude la `tikzpicture` in una scatola e in una Figura come in:

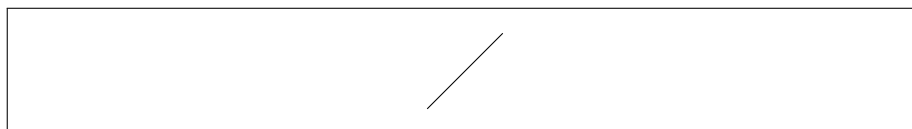


Figura 1

3 `TikZ`

3.1 Unità di misura

1pt	1mm	1cm	1ex	1em	1bp	1dd	1pc	1in	
1	2.84526	28.45274	4.30554	10.00002	1.00374	1.07	12	72.26999	pt
0.35146	1	10.00005	1.51323	3.51462	0.35277	0.37607	4.21754	25.40013	mm
0.03514	0.1	1	0.15132	0.35146	0.03528	0.03761	0.42175	2.54	cm
0.23225	0.66084	6.6084	1	2.32259	0.23312	0.24852	2.78711	16.78534	ex
0.1	0.28453	2.84528	0.43056	1	0.10037	0.107	1.2	7.22699	em
0.99628	2.83467	28.34677	4.2895	9.96277	1	1.06602	11.9553	72.00082	bp
0.93457	2.6591	26.59117	4.02385	9.34575	0.93806	1	11.21487	67.54158	dd
0.08333	0.2371	2.37106	0.3588	0.83333	0.08365	0.08917	1	6.0225	pc
0.01384	0.03937	0.3937	0.05957	0.13837	0.01389	0.0148	0.16605	1	in

Figura 2 Unità di misura ammissibili in \LaTeX



Figura 3



Figura 4



Figura 5

3.1.1 Esercizio

Disegnare due linee concentriche e ortogonali, lunghe 3cm.

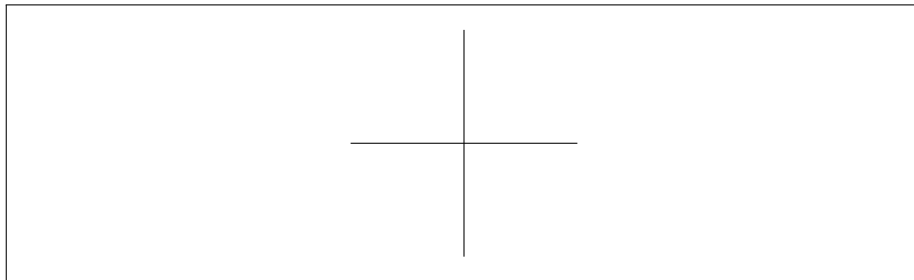


Figura 6

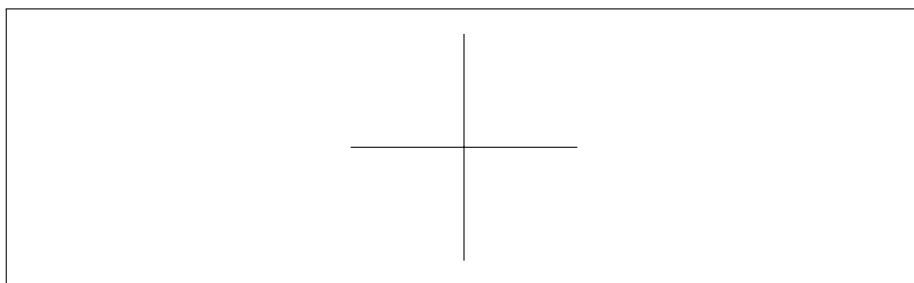


Figura 7

3.2 Esempi di Move-to, Line-to

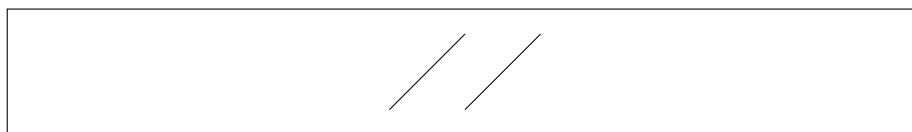


Figura 8

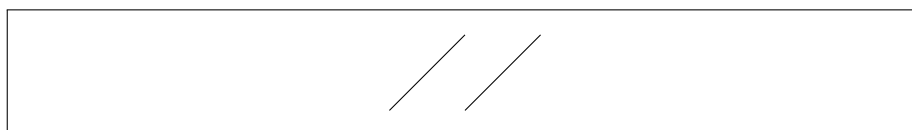


Figura 9

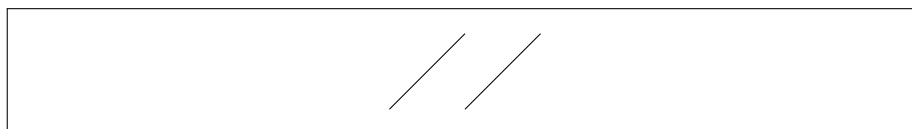


Figura 10



Figura 11



Figura 12



Figura 13

3.2.1 Esercizio

Disegnare un quadrato di lato 1cm .

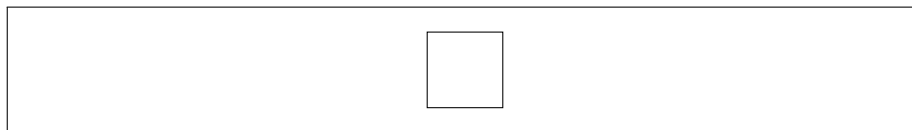


Figura 14

3.3 Coordinate polari

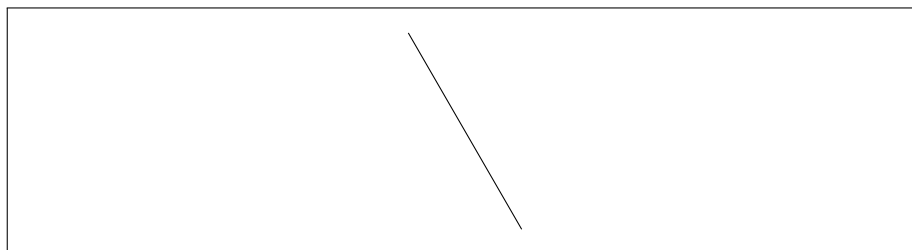


Figura 15

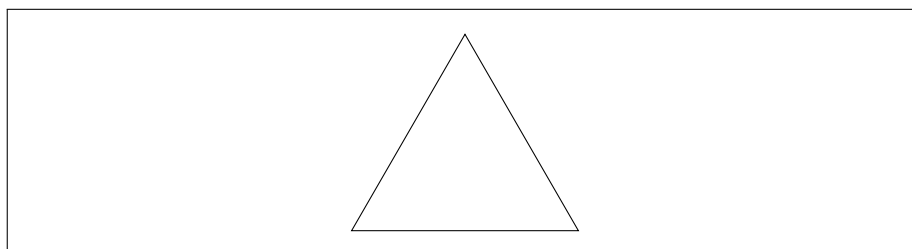


Figura 16

3.4 Movimento relativo (+ e ++)

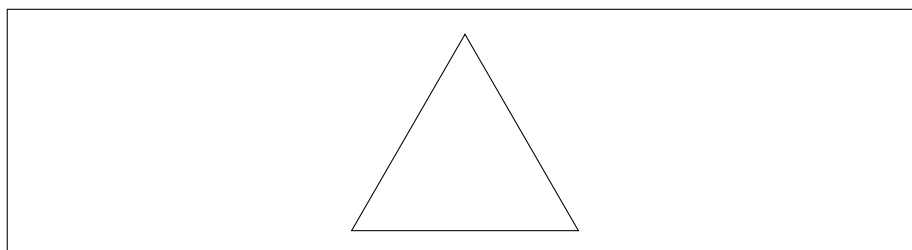


Figura 17

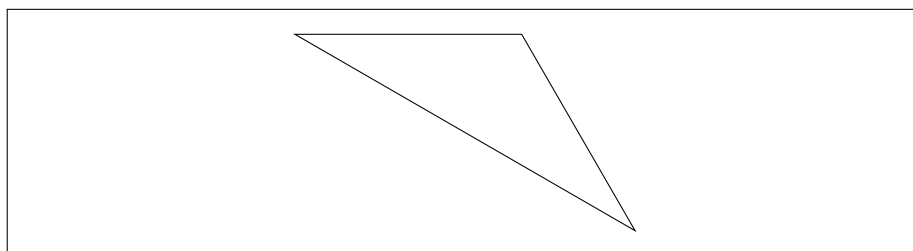


Figura 18

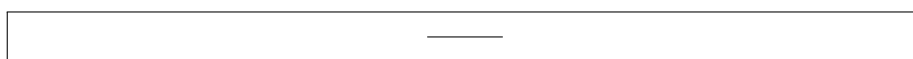


Figura 19

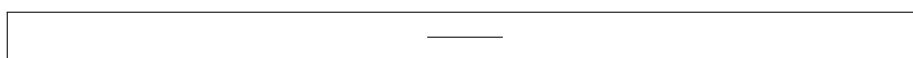


Figura 20

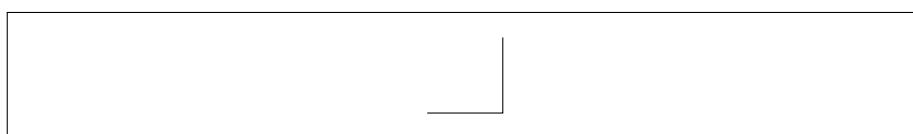


Figura 21

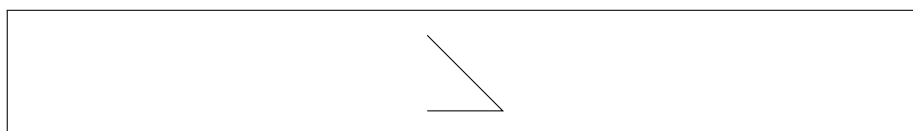


Figura 22



Figura 23

3.4.1 Esercizio

Una scaletta con scalini alti 0.2cm e lunghi 1cm.

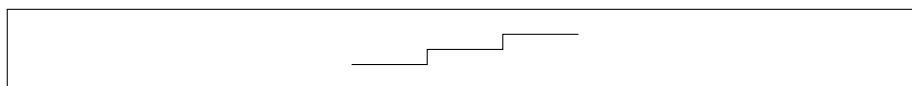


Figura 24

3.4.2 Esercizio

Disegnare un esagono di lato $1.2cm$.

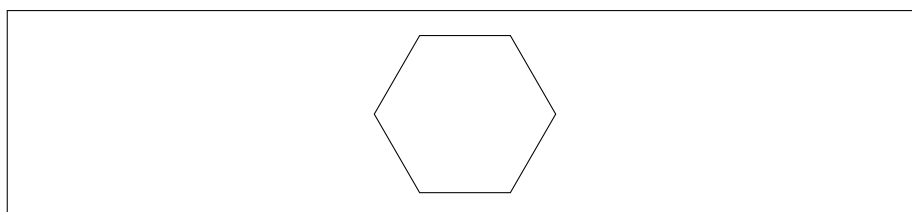


Figura 25

3.5 Opzioni

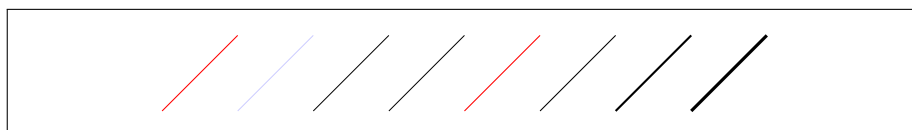


Figura 26

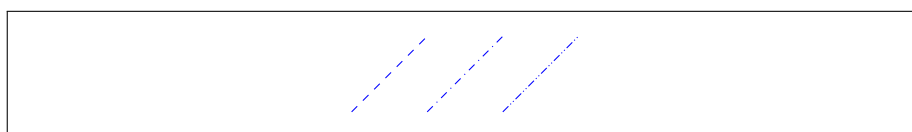


Figura 27

Si possono cambiare le opzioni mentre si disegna:

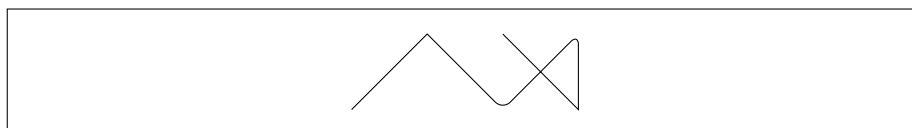


Figura 28

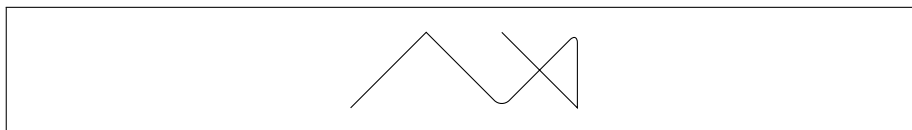


Figura 29

Nota: ci sono alcune eccezioni. Ad esempio, il colore non si può cambiare mentre si disegna un path:

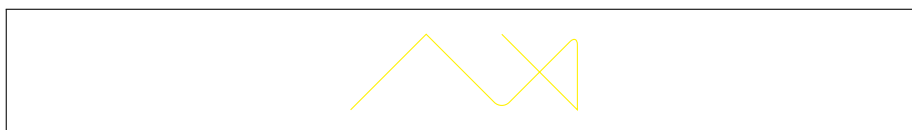


Figura 30

3.6 Frecce ↻

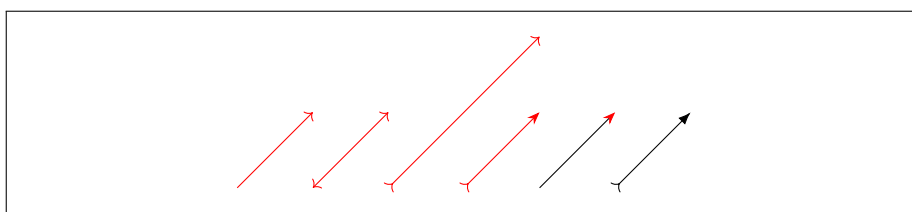


Figura 31

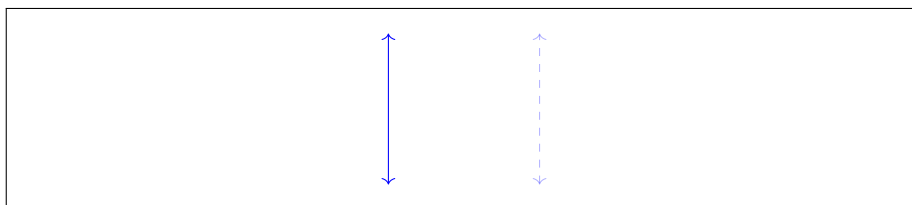


Figura 32

3.6.1 Esercizio

Considerando l'esagono di prima, renderlo spesso $0.5cm$ e tratteggiato come in figura:

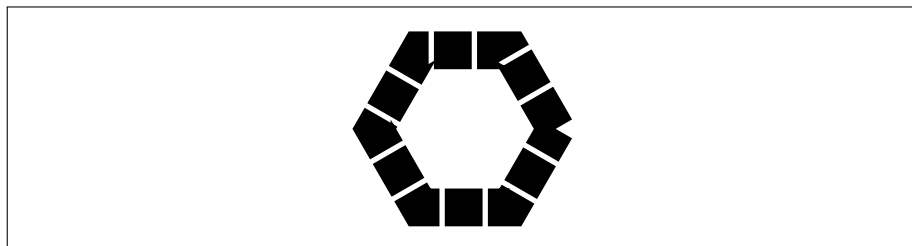


Figura 33

3.6.2 Esercizio

Raffigurare la regola del parallelogramma:

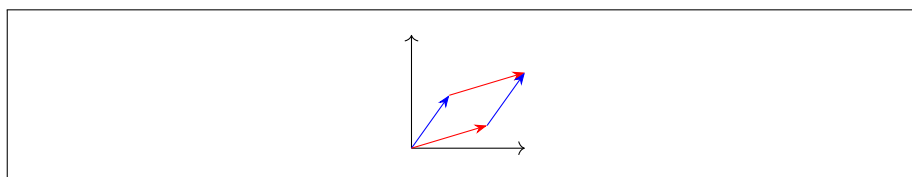


Figura 34

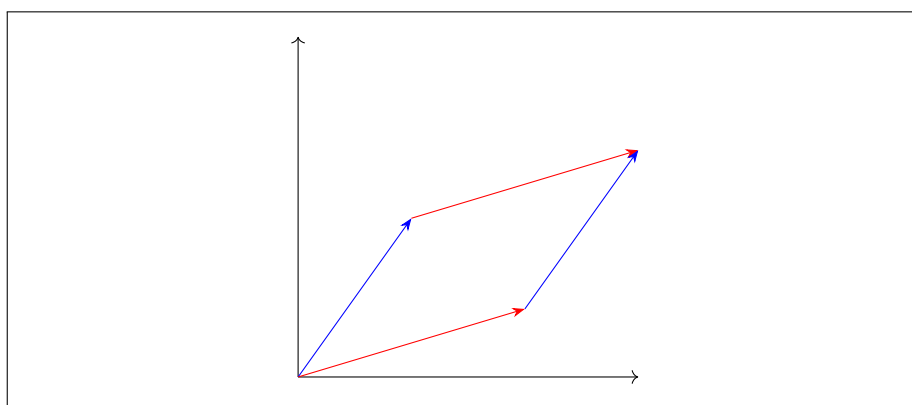


Figura 35

3.7 Coordinate nominate

Rendo dinamica l'immagine usando le macro:

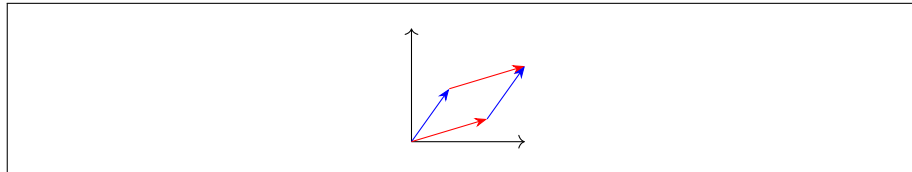


Figura 36

Principale vantaggio: posso accedere alle macro anche qui: “*Regola del parallelogramma applicata ai vettori $(0.5, 0.7)$ e $(1.0, 0.3)$* ”.

Principale svantaggio: ho limiti “sconvenienti” sui nomi da assegnare alle coordinate.

Alternativa: definisco una coordinata tramite il comando `\coordinate`.

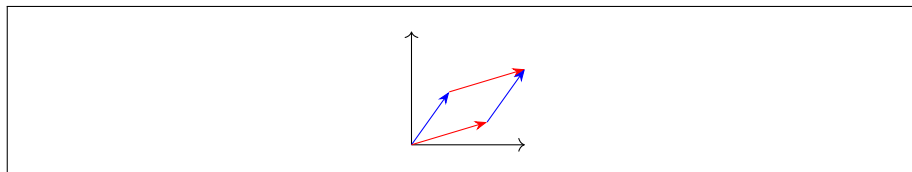



Figura 37

Principale vantaggio: più leggibile e “appropriato”.

Principale svantaggio: non posso riferire alla coordinata fuori dalla `tikzpicture`.

Le coordinate nominate aprono a un mondo di possibilità. Ad esempio, con la semplice opzione `label`  possiamo mostrare il nome di un punto del piano:

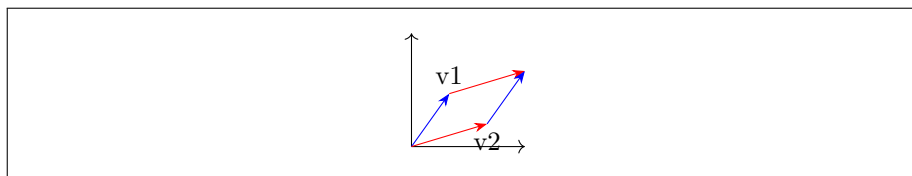


Figura 38

3.8 Coordinate calcolate

Senza librerie esterne, Si possono fare alcuni semplici calcoli nel specificare le coordinate:

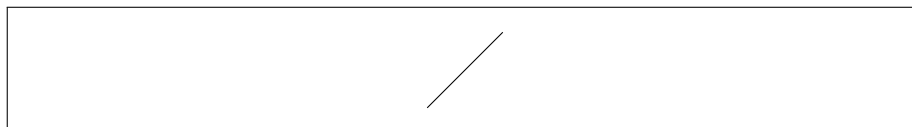


Figura 39

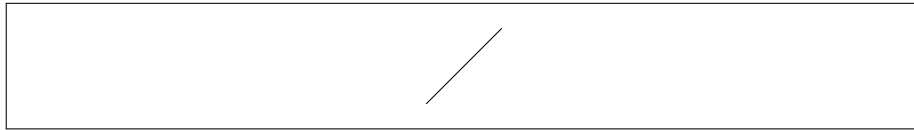


Figura 40

Ma sono limitato, e a volte *TikZ* non mi avverte dei suoi limiti. Tramite la libreria `calc`, posso indicare a *TikZ* come calcolare le coordinate in modi più complessi:

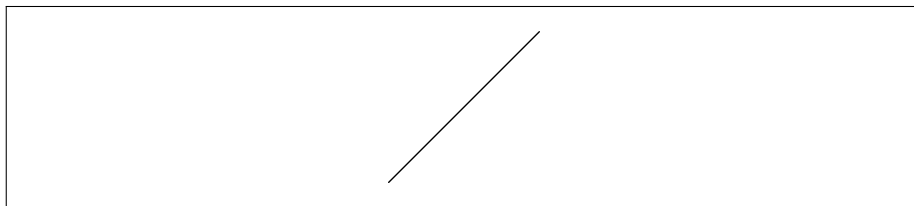


Figura 41

Si ricordino i \$, perché se mancano *TikZ* potrebbe sbagliare silenziosamente. I calcoli possono essere complessi. Esempio:

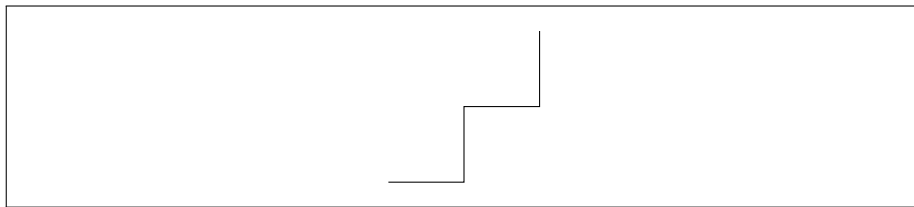


Figura 42

3.8.1 Path-comando `coordinate`

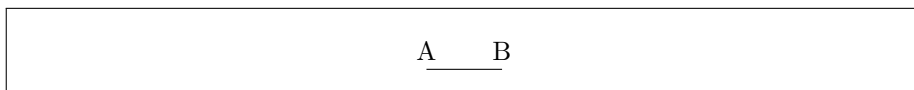


Figura 43

3.8.2 Esercizio

Provare a usare il path-comando `coordinate` per rendere dinamico il path rosso:

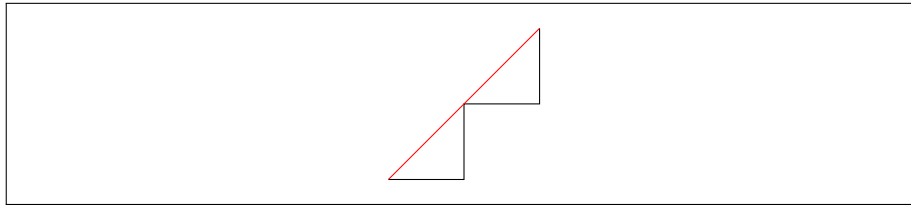


Figura 44

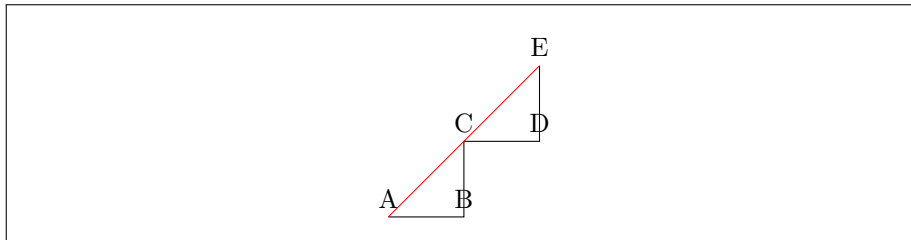



Figura 45

3.8.3 Alcuni calcoli interessanti: partway, distance, projection modifiers

Sintassi comune: **prima coordinata**!modificatore!**seconda coordinata**. A seconda della natura del modificatore, si interpreta questa operazione in maniera diversa.

Partway modifier  una *media pesata* tra due coordinate.

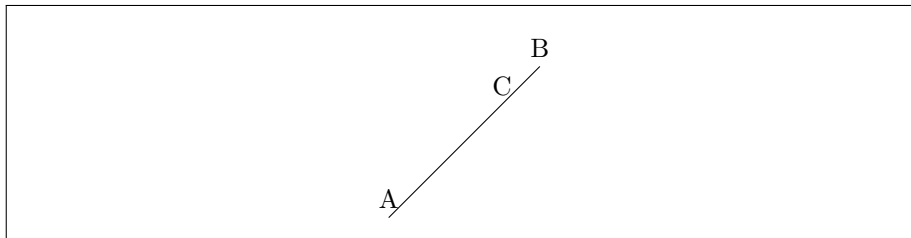


Figura 46

3.8.4 Esercizio

Date due coordinate, e la linea che le congiunge, progettare un ciclo `\foreach` che produca delle frecce; il risultato deve essere una serie di frecce che indichino i punti a 0%, 20%, 40%, ..., lungo la linea, proseguendo fino a ricoprire il doppio della distanza tra le due coordinate:

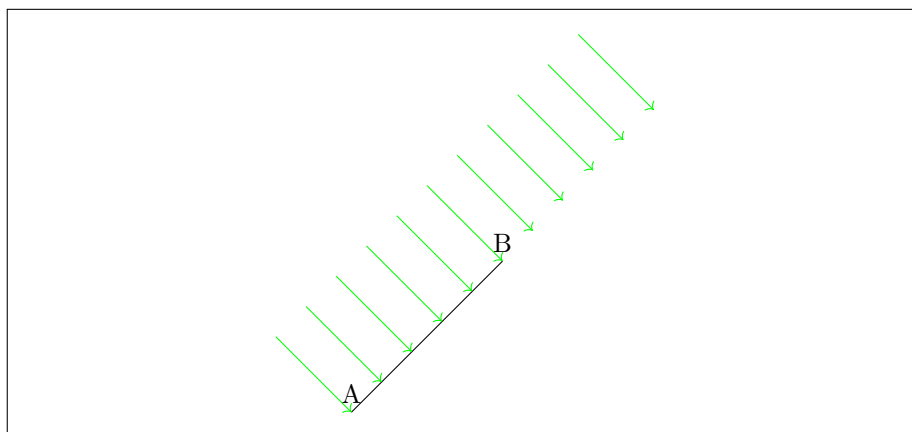



Figura 47

Distance modifier  movimento relativo rispetto al primo punto, in direzione di un secondo punto.

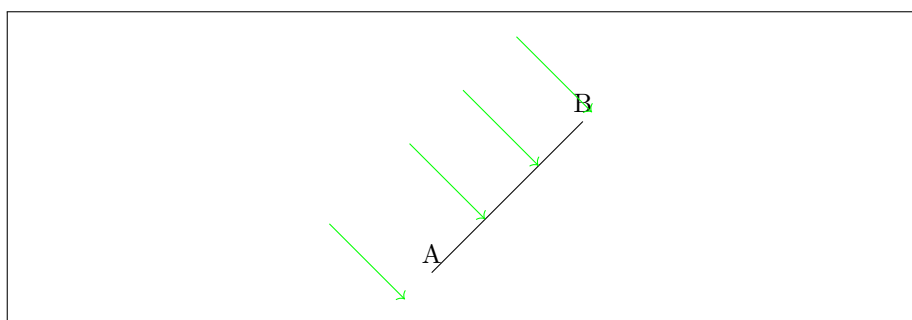



Figura 48

Projection modifier  proiezione di un terzo punto lungo la linea che congiunge i due punti di riferimento.

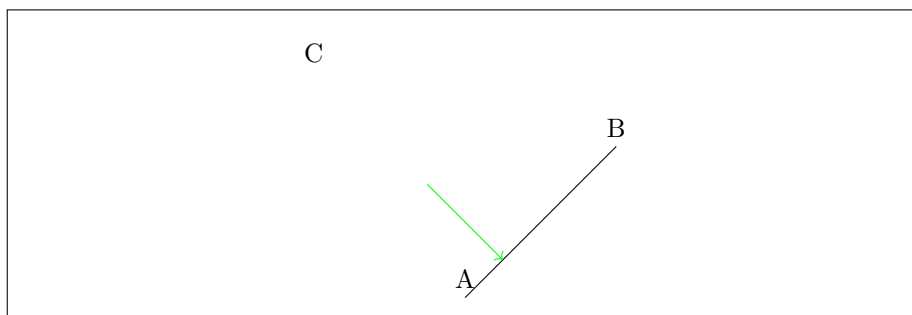


Figura 49

3.9 Linee verticali/orizzontali , Curve-To , archi

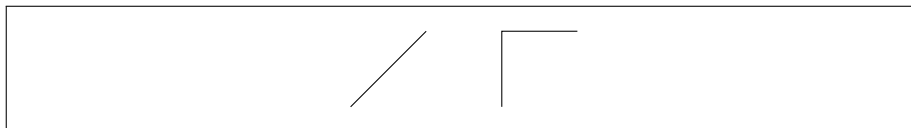


Figura 50

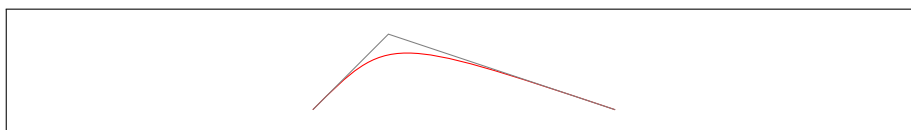


Figura 51

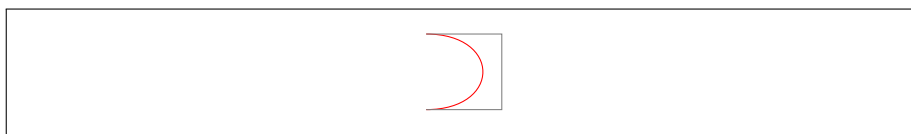


Figura 52

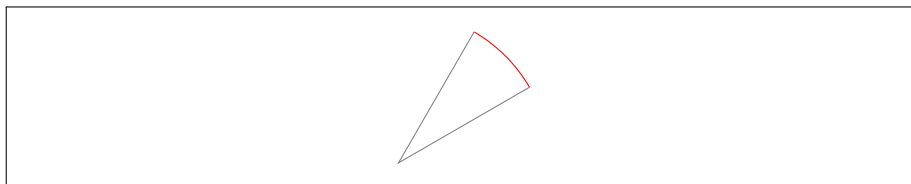


Figura 53

3.9.1 Esercizio

Disegnare degli assi lunghi 1.5cm come in figura, ma usando una sola draw e il comando $|$ —:

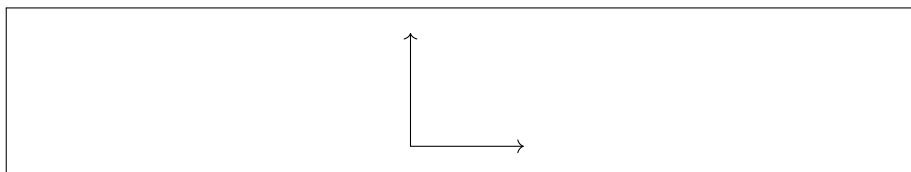


Figura 54

3.10 Coordinate all'intersezione a linee perpendicolari

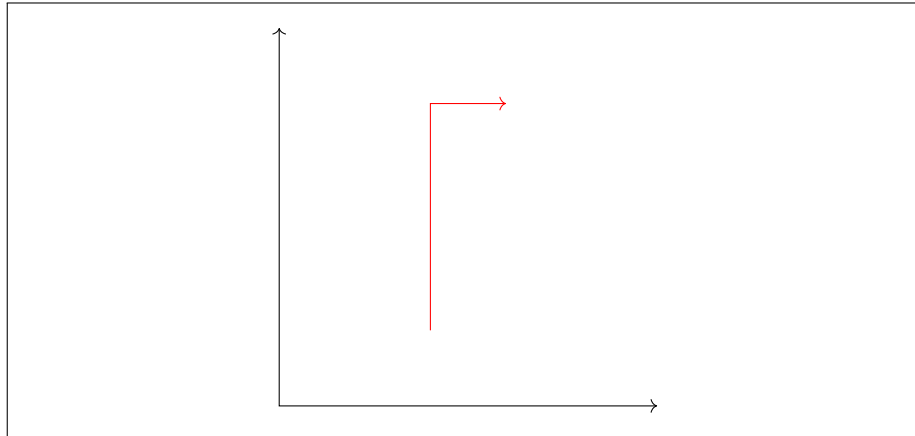


Figura 55

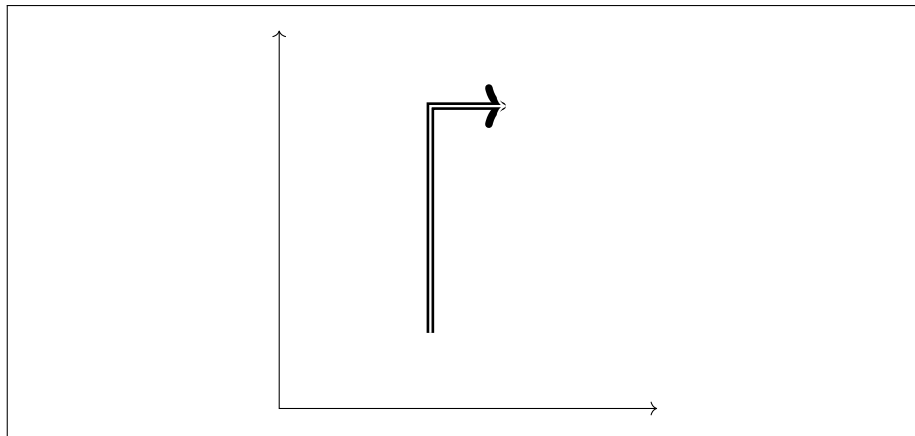


Figura 56

Utile per disegnare le proiezioni di un punto:

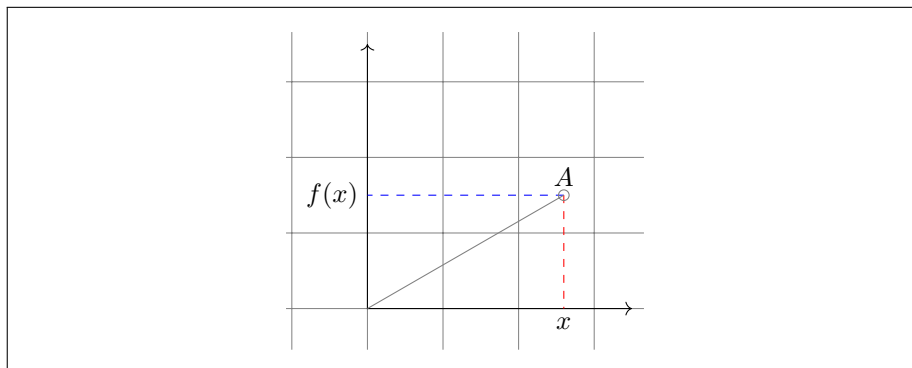



Figura 57

È anche possibile calcolare le coordinate a intersezioni tra path complessi .

3.11 Rettangoli , cerchi, ellissi

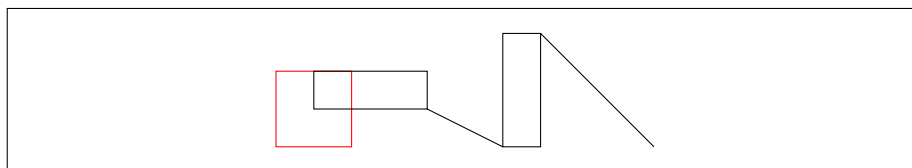


Figura 58

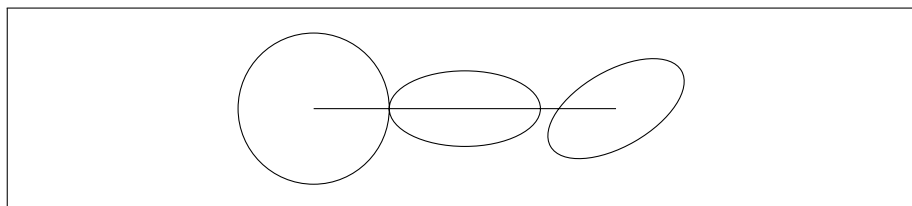


Figura 59

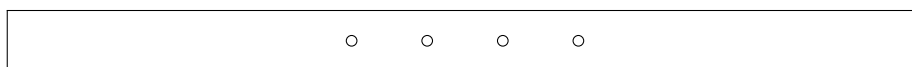



Figura 60

Nota: anche `draw` ha una opzione `rotate` , diversa dalla `rotate` di `circle`; la prima ruota il tracciato attorno all'origine, la seconda disegna il cerchio già ruotato attorno al proprio centro:

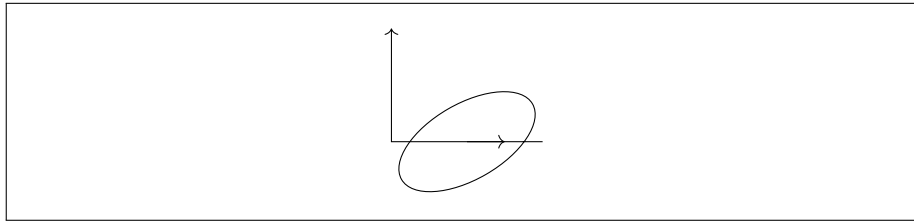


Figura 61

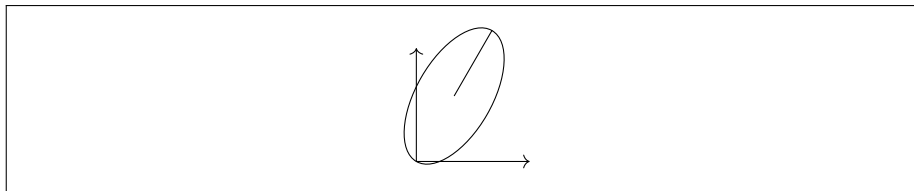


Figura 62

3.12 Comando `\path`: bordi e riempimenti

`\path` è il principale comando *TikZ* per disegnare tracciati. A differenza di `\draw`, di default, questo comando crea il tracciato ma non disegna nulla:

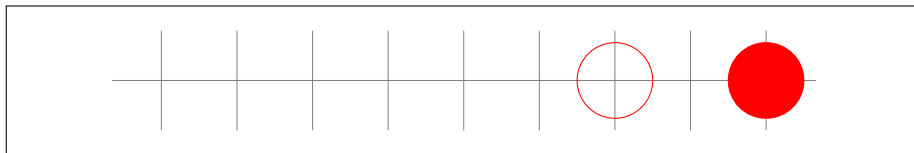


Figura 63

Per disegnare il `path`, bisogna specificare l'opzione `draw`:

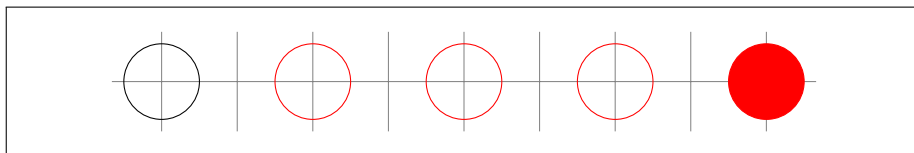


Figura 64

In realtà, il comando `\draw`, così come altri comandi è una istanza di `\path`:

- `\draw = \path[draw];`
- `\fill = \path[fill];`

- `\filldraw = \path[fill,draw];`
- ...

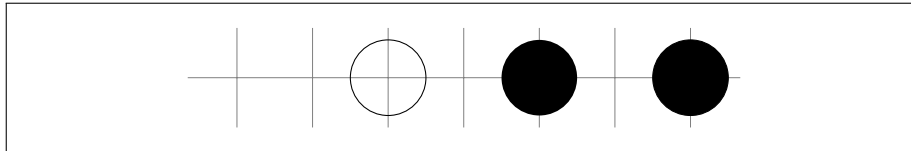


Figura 65

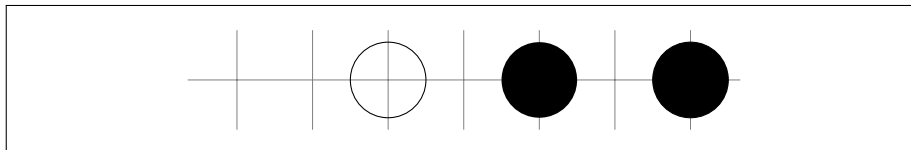


Figura 66

Di default, il colore specificato con `color` viene usato sia per il riempimento che per il bordo del tracciato:

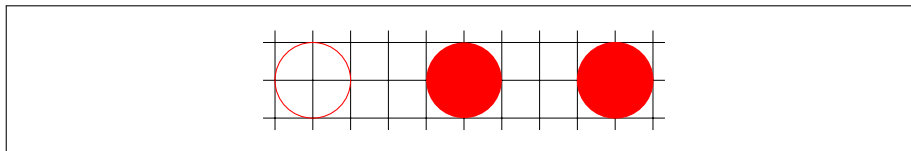


Figura 67

Nota: il colore può anche essere passato direttamente come opzione (e.g., `thin`), senza specificare `color=`:

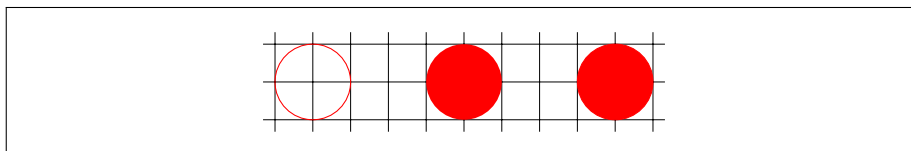


Figura 68

Tuttavia, si possono usare colori diversi per `fill`, `draw`:

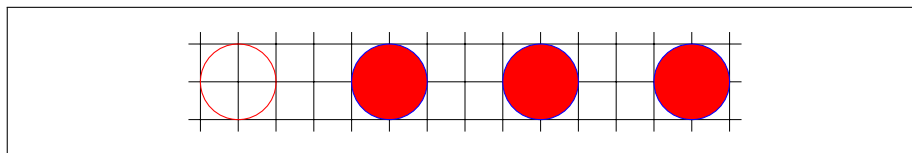


Figura 69

A proposito, si possono controllare indipendentemente i colori per fill, draw, e per il testo generato nella costruzione path. Quando si specifica un colore come chiave (e.g., `[red]`) o tramite `color` (e.g., `[color=red]`), vengono settati automaticamente le tre chiavi:

- `fill color`;
- `draw color`;
- `text color`.

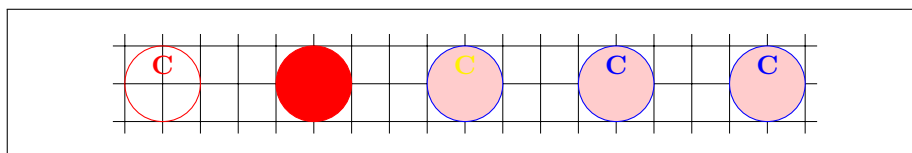


Figura 70

Un discorso simile può essere fatto per la chiave `opacity`, che controlla insieme `fill opacity`, `draw opacity` e `text opacity`.

3.12.1 Riempimento con `fill` , `pattern` , `shade`

La chiave `fill` controlla il riempimento dell'interno del path.

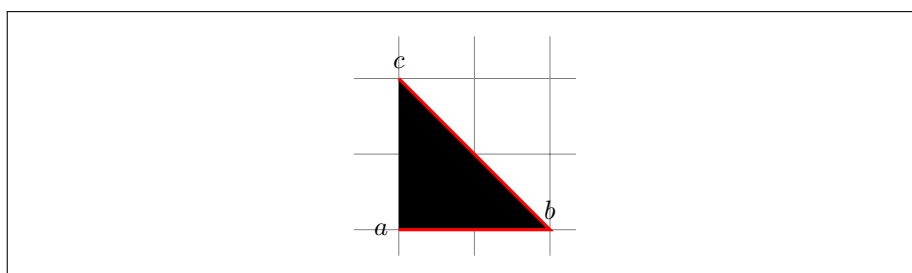


Figura 71

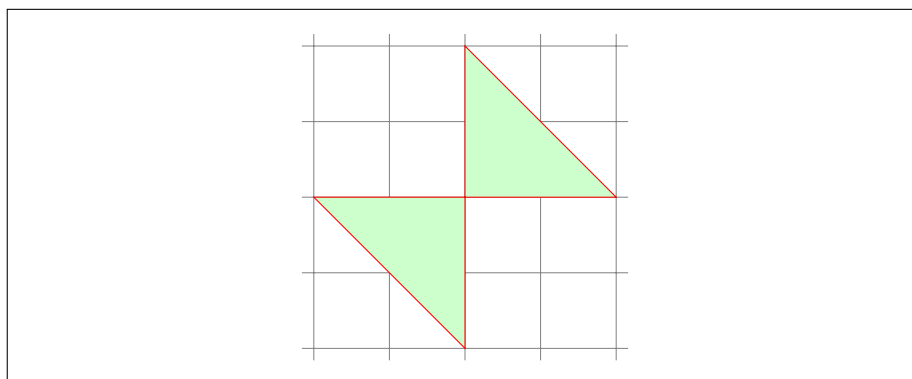


Figura 72

Ma cos'è... l'interno? Dipende dalla **fill rule** :

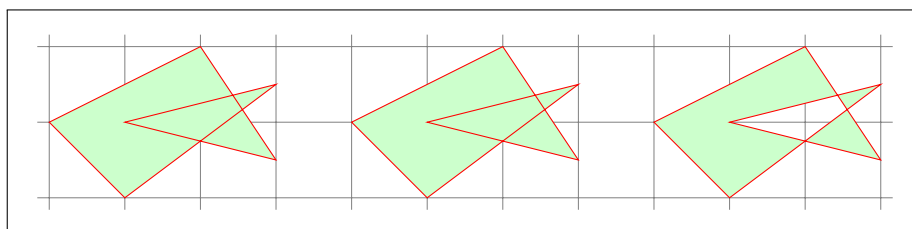


Figura 73

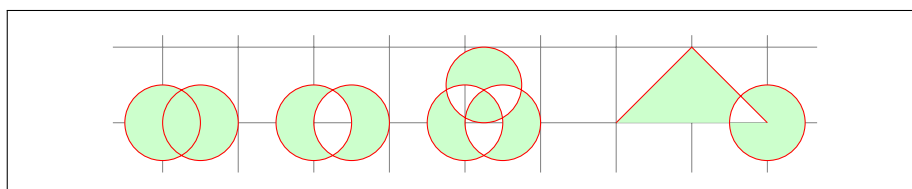


Figura 74

Esempio di fill con arco di circonferenza:

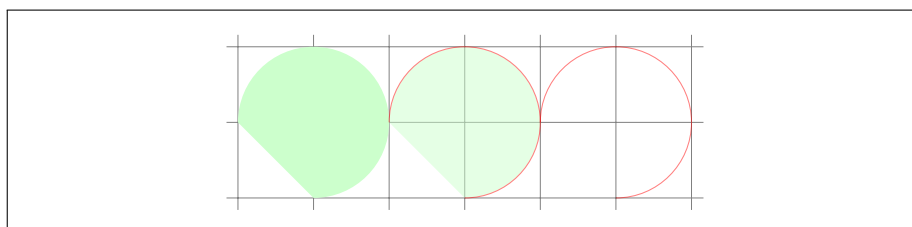


Figura 75

Nota: la presenza di oggetti diversi da linee può influenzare il significato del riempimento.

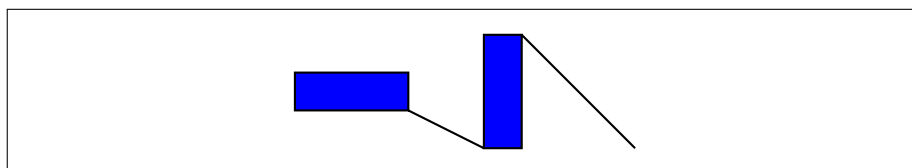


Figura 76

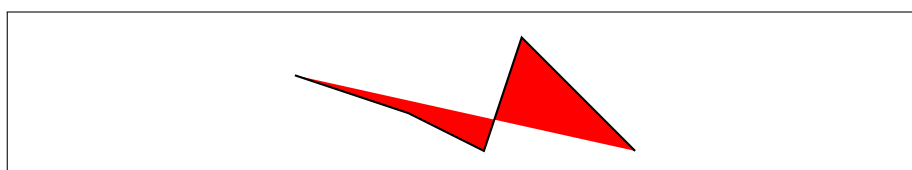


Figura 77

Riempimento con pattern  .

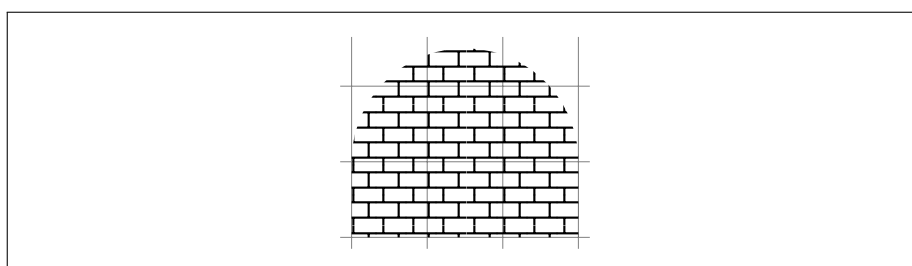


Figura 78

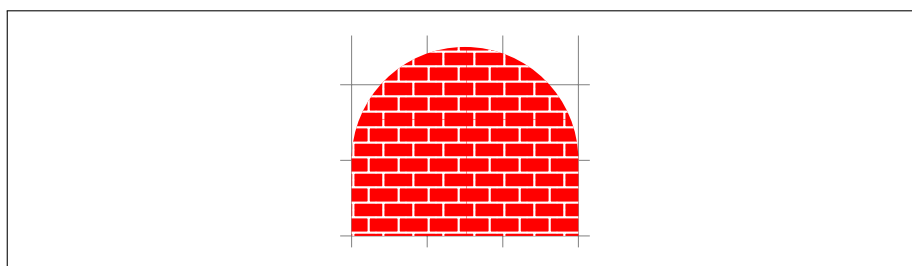



Figura 79

Riempimento con shade .

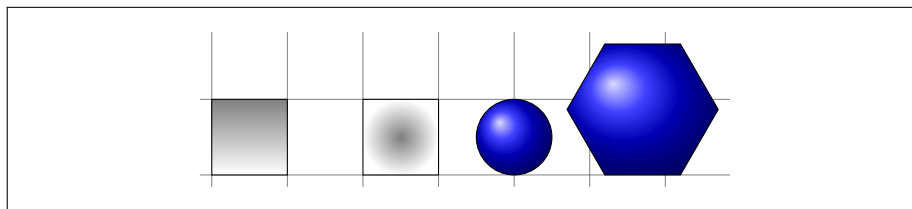


Figura 80

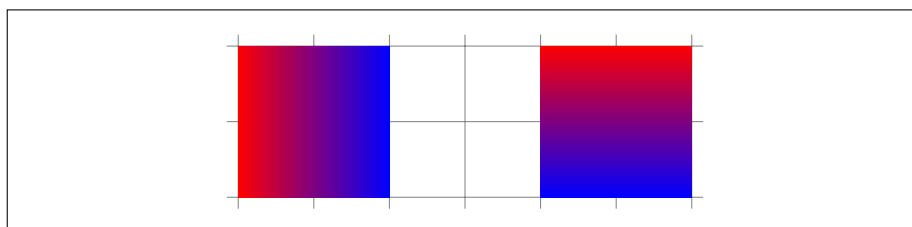


Figura 81

3.12.2 Esercizio

Disegnare questa girandola tramite comandi `\filldraw` e `\pattern`:

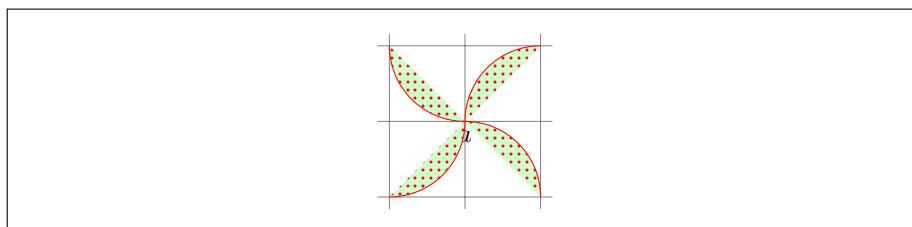


Figura 82

3.13 Un po' di programmazione!

3.13.1 Ciclare con `\foreach`

Disegno 4 ellissi ruotando sistematicamente ciascuno di essi:

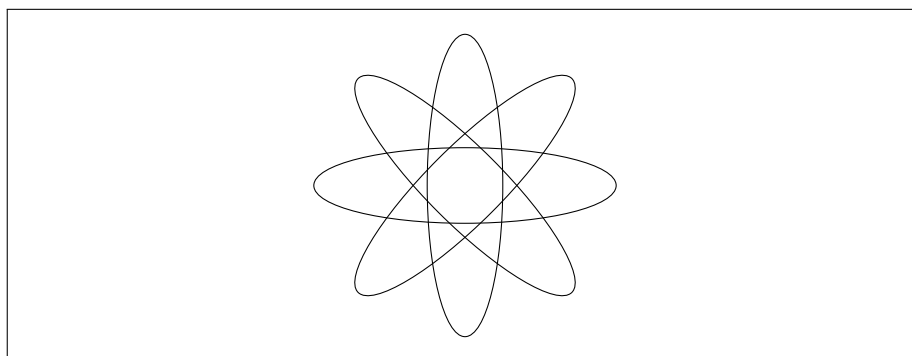


Figura 83

Può essere usato anche per generare path-comandi (e.g., dentro una `\draw`):

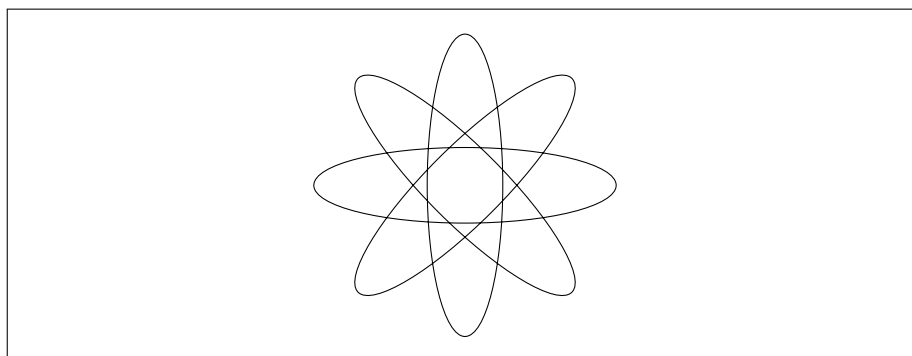


Figura 84



Figura 85

3.13.2 Esercizio

Disegnare 30 ellissi equicentrati, ruotati in modo da distribuirsi uniformemente:

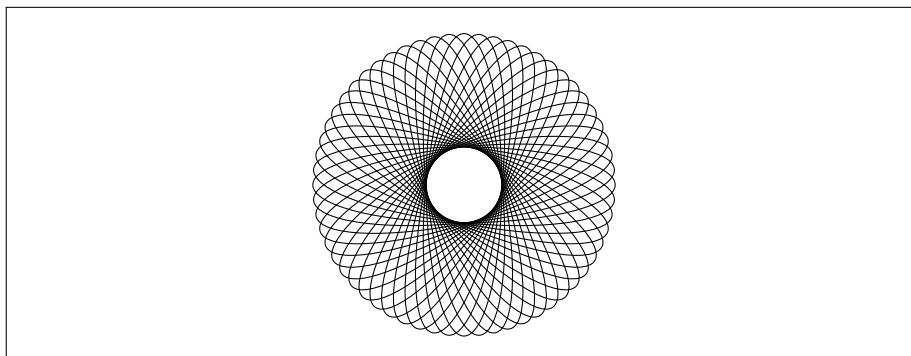


Figura 86

3.13.3 Esercizio

Trovare una soluzione all'esercizio precedente, generalizzata a n ellissi:

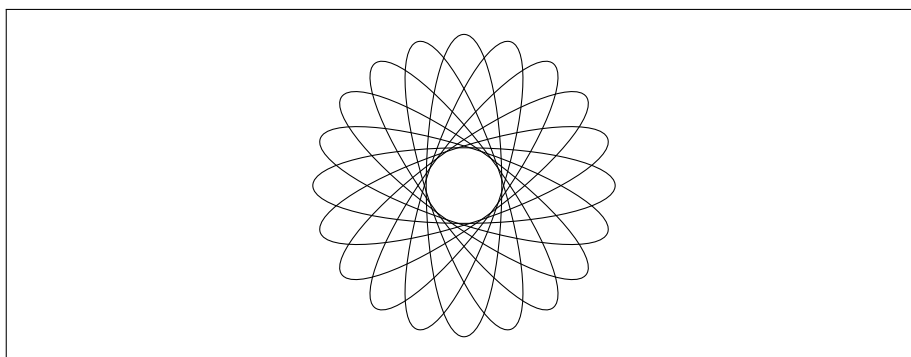


Figura 87

3.13.4 Esercizio

Riprendiamo la nostra scaletta con scalini alti 0.2cm e lunghi 1cm:

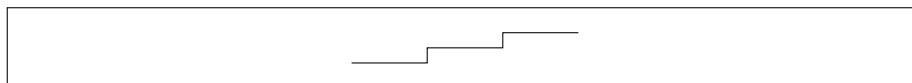


Figura 88

E rendiamola parametrica nel numero di scalini, con uno dei due modi di ciclare, numerando ciascun scalino.

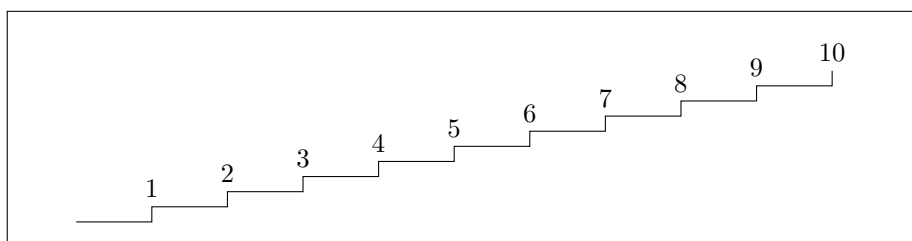


Figura 89

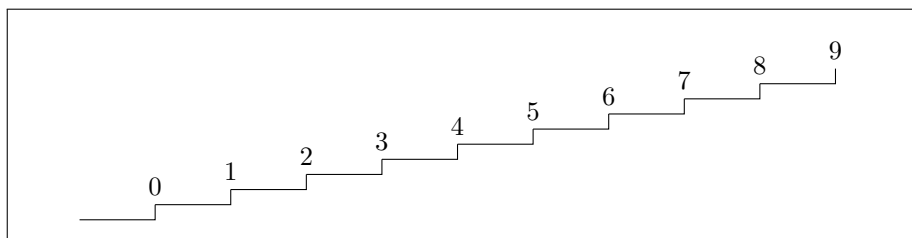


Figura 90

Posso indicare a `\foreach` di salvarmi in una macro contatore il numero dell'iterazione (`count`):

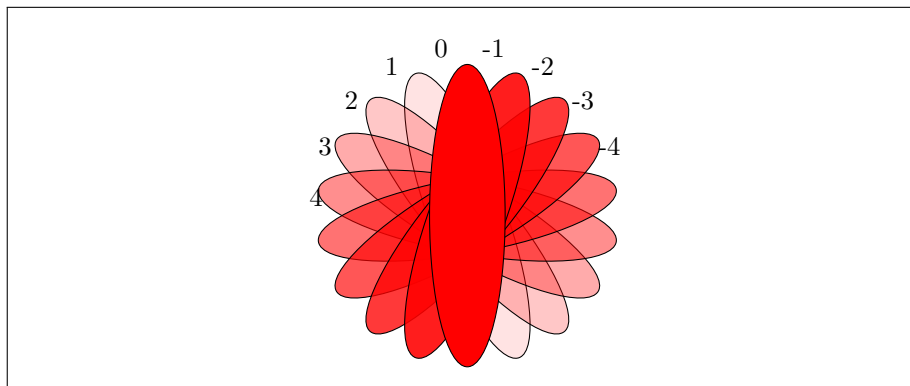


Figura 91

3.13.5 Esercizio

Nella figura precedente, è stata variata la `fill opacity`, che controlla la trasparenza/opacità del riempimento. Cambiare il codice della figura precedente in modo che, anziché l'opacità, sia il colore stesso del riempimento a variare, e che questo vari tra rosso e bianco. Ricordiamo la sintassi per mischiare i colori: `colore1!valorepercentuale!colore2`.

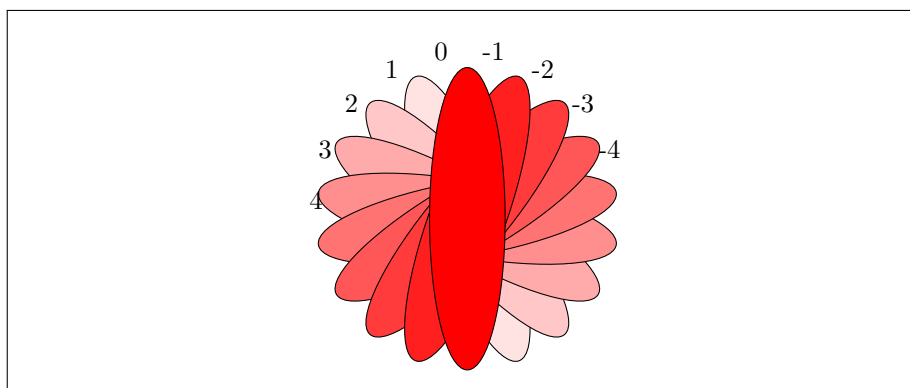


Figura 92

Possiamo usare la variabile del `\foreach` per creare coordinate in maniera sistemica:

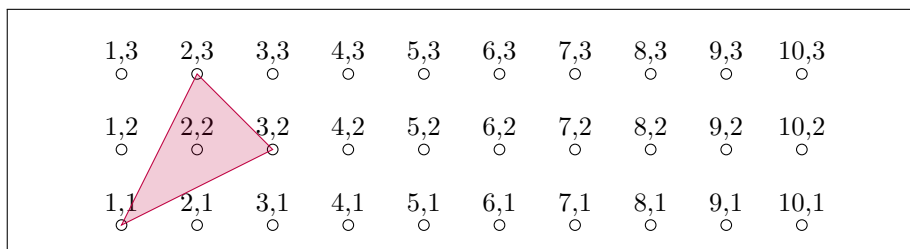


Figura 93

Tecniche avanzate per il `\foreach`:

- Dot notation nel specificare le liste:
Esempio 1: $2^1, 2^2, 2^3, 2^4, 2^5, 2^6, 2^7$,
Esempio 2: $A_1, B_1, C_1, D_1, E_1, F_1, G_1, H_1$,
- Opzione `evaluate-as-using` per assegnare a una macro una valutazione matematica sulla variabile:
Esempio 3: $2^0 = 1.0, 2^1 = 2.0, 2^2 = 4.0, 2^3 = 8.0, 2^4 = 16.0, 2^5 = 32.0, 2^6 = 64.0, 2^7 = 128.0, 2^8 = 256.0$,
- Opzione `parse=true` per valutare matematicamente il limite superiore della lista di iterazione:
Esempio 4: 0.09999, 0.2, 0.3, 0.40001, 0.50002, 0.60002, 0.70003, 0.80003, 0.90001
- Comando `\breakforeach` per terminare l'iterazione (utile se associato ai *condizionali*).

3.13.6 Condizionali

- `\ifnum` si può usare per verificare relazioni di uguaglianza, minoranza o maggioranza ($=$, $<$, $>$) tra numeri *interi*.
Esempio 5: YES
Esempio 6: NO
Esempio 7: YES
- `\ifdim` è l'equivalente che funziona per numeri *reali*. È pensato per dimensioni L^AT_EX, pertanto si aspetta unità di misura.
Esempio 8:
Esempio 9: YES
- `\ifodd` testa la parità di un numero intero:
Esempio 10: 5 è un numero dispari.
- Per confronti tra stringhe (i.e., sequenze di caratteri), consiglio il pacchetto `xifthen`, che mette a disposizione `\ifthenelse`, unito a:
 - `\equal{str1}{str2}`: controlla l'uguaglianza tra stringhe;
 - `\isempty{str}`: controlla se una stringa è vuota;
 - `\isin{str1}{str2}`: controlla se una stringa appare come sottostringa di un'altra;

- `\endswith{str1}{str2}`: controlla che una stringa sia suffissata da un'altra.

Esempio 11: TRUE FALSE

Esempio 12: TRUE FALSE

Esempio 13: TRUE TRUE

Naturalmente, la potenza dei condizionali sta nel fatto di poter espandere macro nelle condizioni:

Esempio 14: Il mio numero è **dispari** e **maggiore** di 4.2.

3.13.7 Esercizio live!

Ho definito tre comandi `\equalitycheck`, `\minorecheck`, `\maggiorecheck` che confrontano due interi. Come li espando per accettare anche valori reali?

Esempio 15:

`\disparicheck`: NO YES

`\equalitycheck`: NO YES

`\maggiorecheck`: YES NO

`\minorecheck`: NO NO

3.13.8 Esempio

Condizionali innestati che controllano il colore:

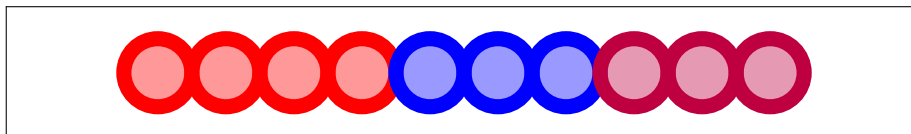


Figura 94

3.13.9 Esercizio

Disegnare una griglia 10×10 di righe verticali e orizzontali, con righe di spessore alternato `very thin`/`very thick`:

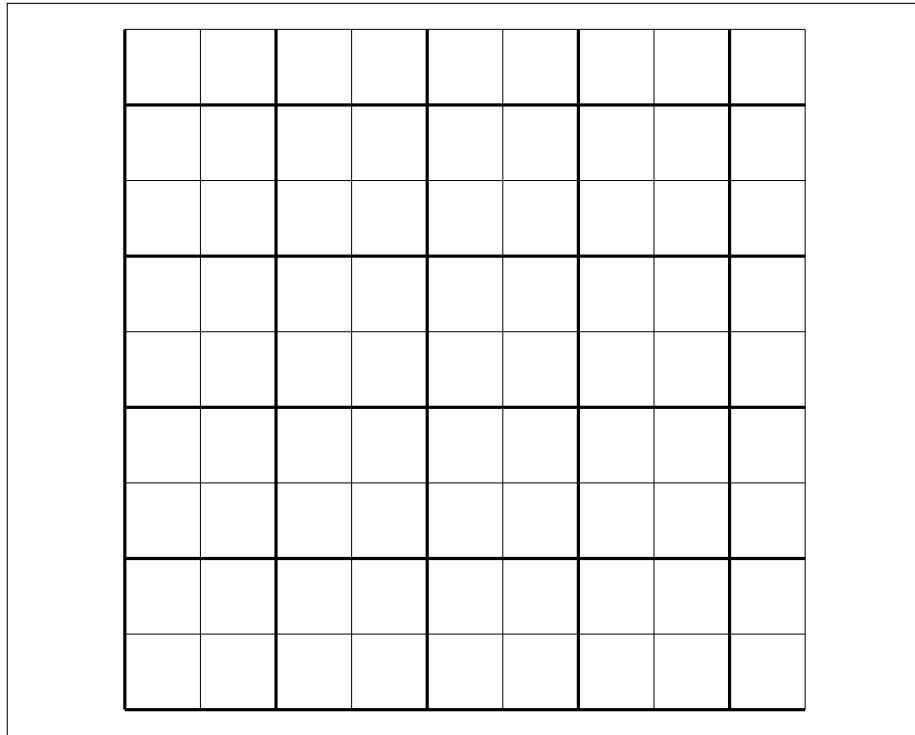


Figura 95

3.13.10 Esercizio

Disegnare una matrice di 12×12 cerchi di raggio 10pt, riempiendo quelli sulla diagonale di rosso (o usando un `ball shading`):

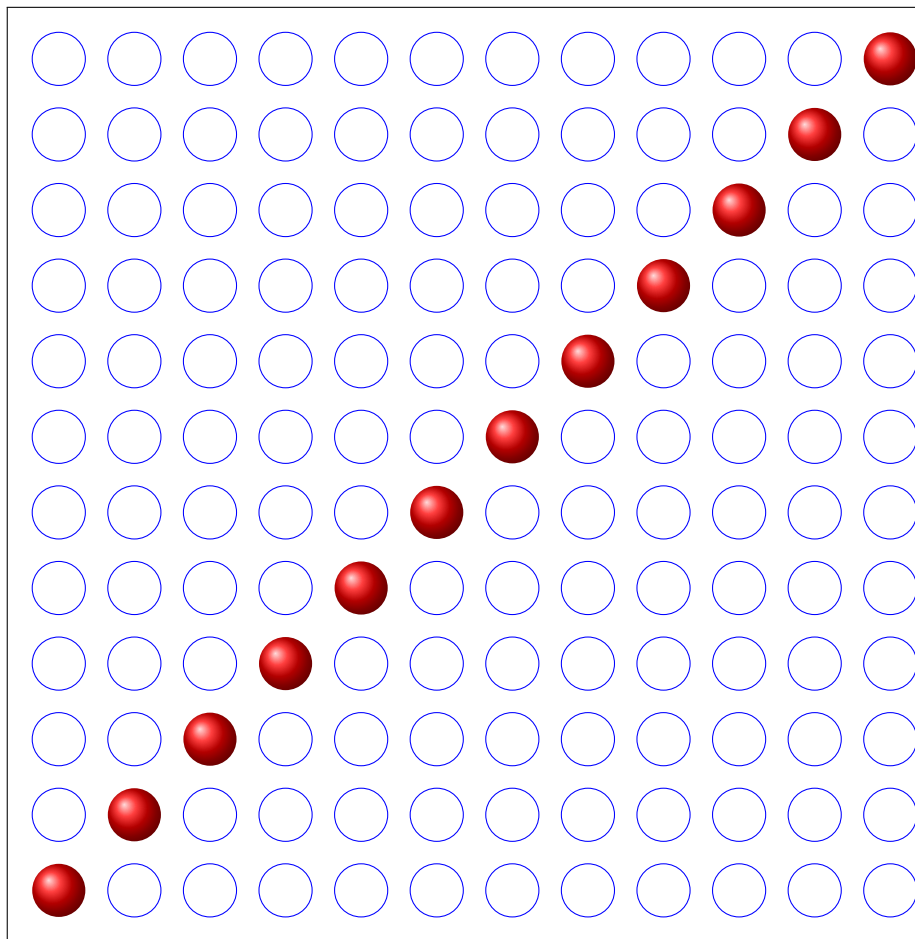


Figura 96

3.13.11 Calcoli

- `\pgfmthparse` e `\pgfmthresult` si possono usare per calcoli aritmetici generici:

Esempio 16: `2.25NO`

Esempio 17: `5*2 > 10.1 non è vero`

- `\pgfmthsetmacro` e `\pgfmthtruncatemacro` assegnano il valore calcolato da una espressione aritmetica ad una macro specificata (similmente a `\newcommand`).

`\pgfmthsetmacro` si usa per calcoli tra numeri *reali*,

`\pgfmthtruncatemacro` tronca il risultato, e si usa tra numeri *interi*.

Esempio 18: `1.50000`

Esempio 19: `1`

Esempio 20: `1`

3.13.12 Esercizio

Cambiare la politica di riempimento della matrice di prima, riempiendo i cerchi che stanno sulla parabola

$$y = \left\lceil \frac{x^2}{10} \right\rceil :$$

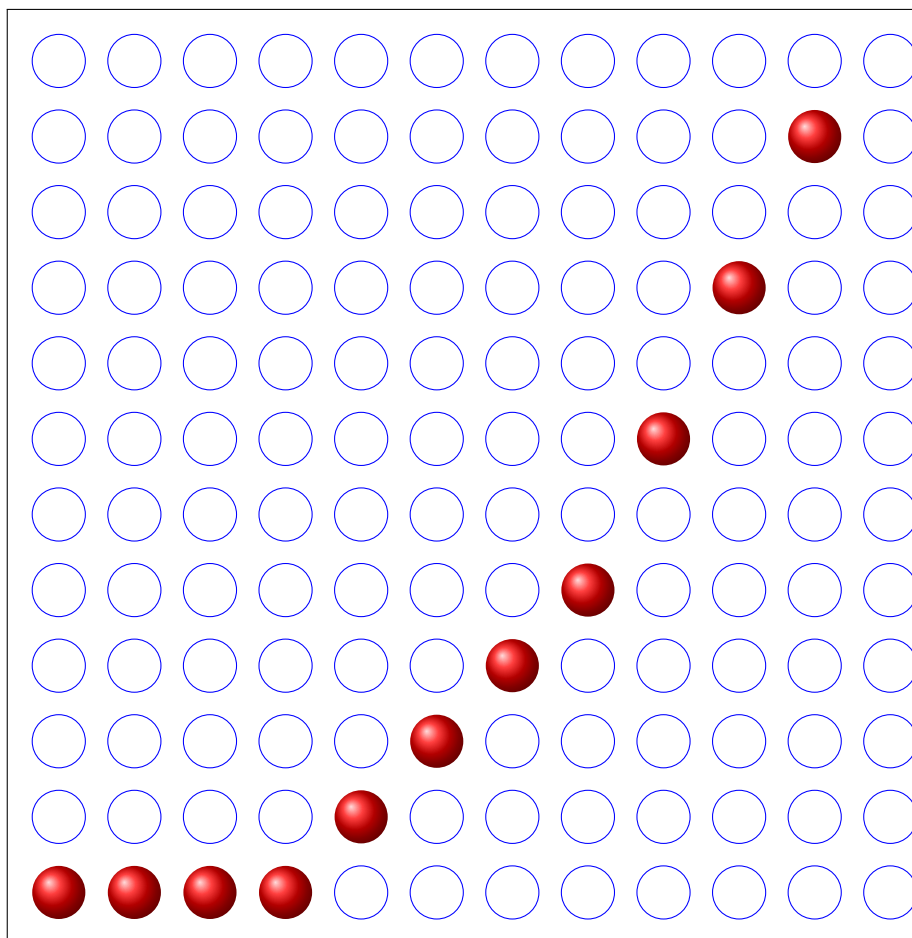


Figura 97

3.14 Esercizio

Creare un comando `\girandola`, che disegni n quadrati strutturati nella seguente maniera:

3.15 Nodi

Un nodo si comporta come una coordinata, ma ha una *estensione*, una *forma* e un *contenuto*. Similmente alle coordinate, si crea con il comando `\node`, o con il path-comando `node` dentro un `\path`; il *contenuto* viene specificato in coda al comando, dentro a graffe:

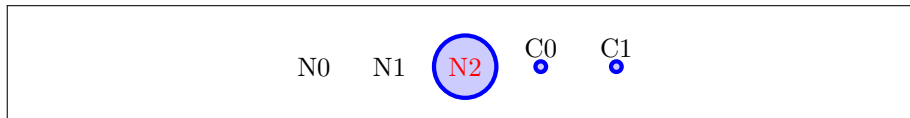


Figura 99

Si noti la differenza tra nodo con testo e coordinata con etichetta. L'etichetta è sempre apposta a lato della coordinata, anche quando si specifica una distanza nulla. In realtà, la coordinata è un tipo particolare di nodo, senza forma né contenuto:

```
\coordinate[opzioni] ... ≡ \node[coordinate,opzioni] ... {};
```

La forma di un nodo si può specificare tramite l'opzione `shape` (oppure anche elidendo l'assegnazione `shape=`). Di default, sono disponibili le forme *circle*, *rectangle*, e, appunto, *coordinate*:

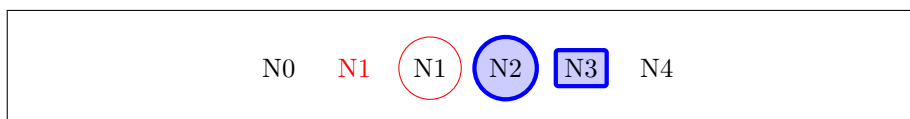


Figura 100

Altre forme per i nodi (e.g., diamante, ellisse, trapezio, ...) sono disponibili nella libreria **shapes** .

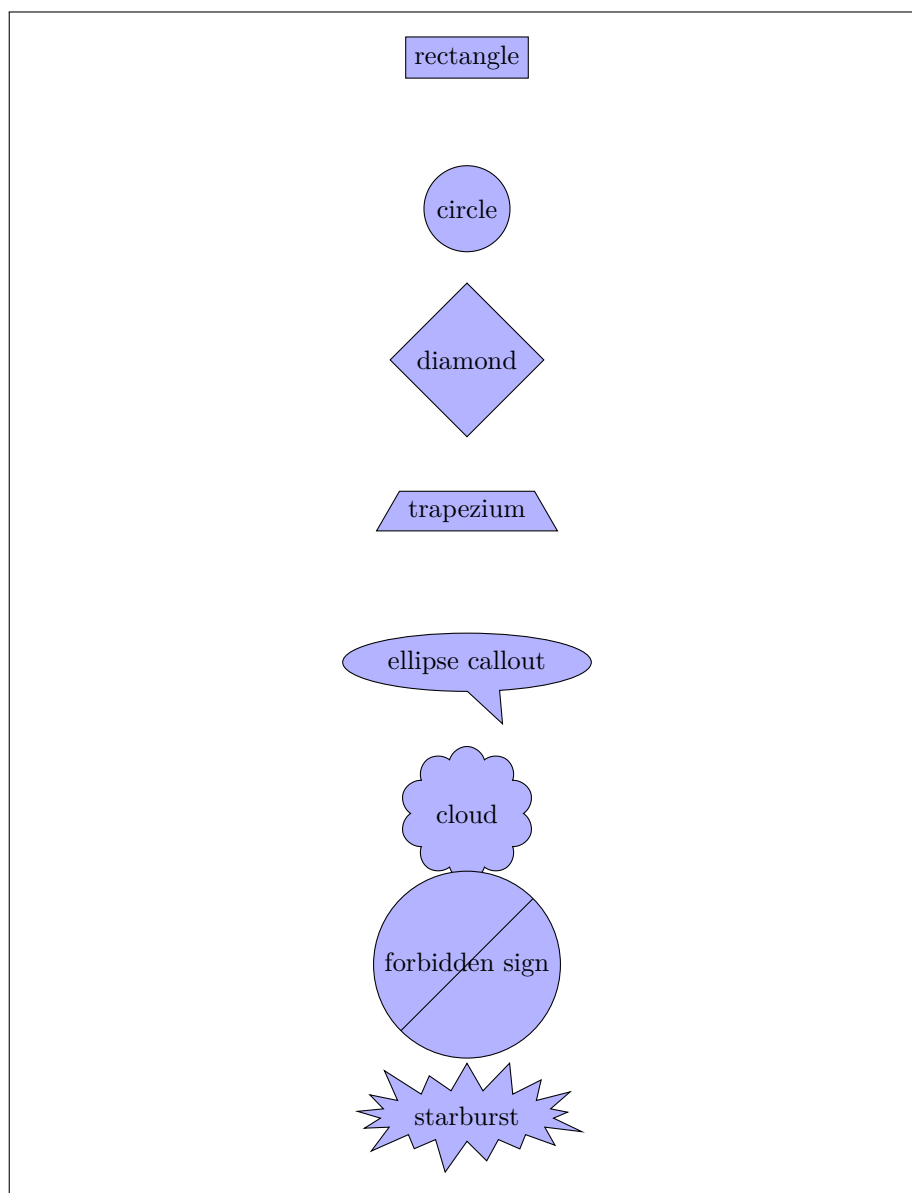


Figura 101

Per esempio: **regular polygon**:

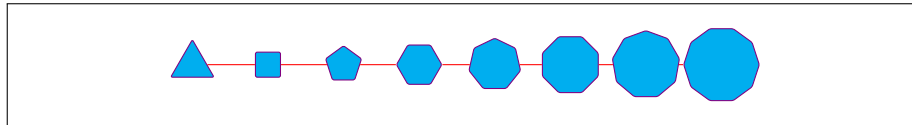


Figura 102

Uno dei punti forti dei nodi sta nelle *line shortening rules*, che permettono di trattare il nodo come un *oggetto geometrico* dotato di una area interna ed una esterna. Il suo uso risulta, di conseguenza, sostanzialmente diverso rispetto a quello delle coordinate; per esempio, un path che collega nodi con estensione (ovvero nodi che non sono coordinate) non può essere riempito:

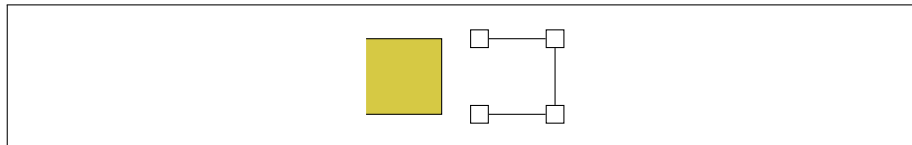


Figura 103

3.15.1 Geometria del nodo

La struttura geometrica di un nodo può essere semplificata così:

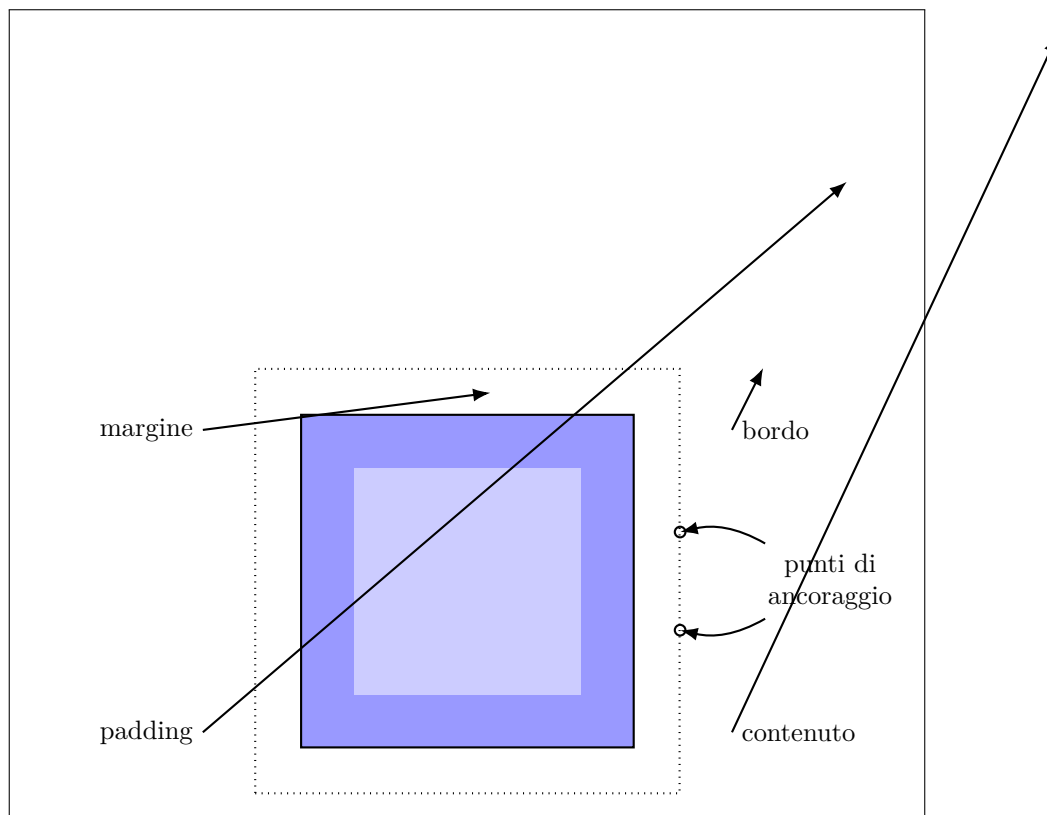


Figura 104

Partendo dall'interno:

- La dimensione del contenuto viene gestita in automatico;
- I padding (o *spaziature interne*) si posson specificare con ***inner sep/xsep/ysep***;
- La dimensione del bordo si può specificare con ***line width***;
- I margini (o *spaziature esterne*) si posson specificare con ***outer sep/xsep/ysep***;

Inoltre, la *dimensione minima* dell'area di filling (contenuto+padding) si posson specificare con ***minimum size/width/height***. Di default, un nodo ha un poco di spaziatura interna, ma spaziatura esterna nulla.

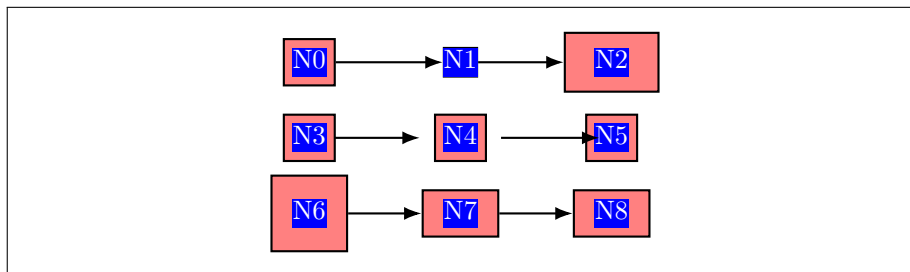


Figura 105

Si noti l'uso di `every path/.style` e `every node/.style`, che permette l'assegnazione di opzioni a tutti i nodi o tutti i path all'interno di una `tikzpicture` [🔗](#).

Cogliamo l'occasione per presentare le opzioni per controllare lo scaling della figura: `scale`/`xscale`/`yscale`. Queste riscalano lo spazio di disegno, ricalcolando le coordinate dei nodi prima di disegnarli, ma lasciando le dimensioni dei nodi invariate. La stessa figura senza `xscale=2.0` si presenta come:

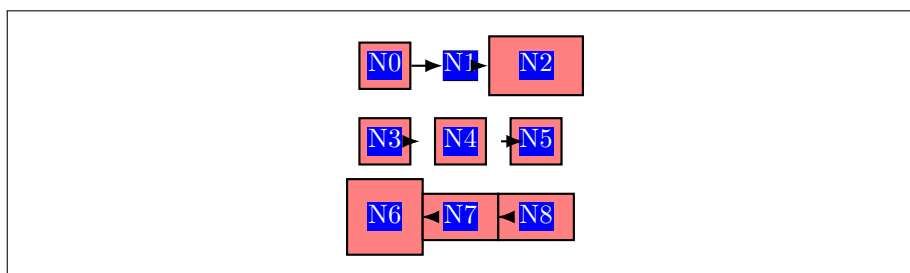


Figura 106

La stessa figura con `scale=2.0` si presenta come:

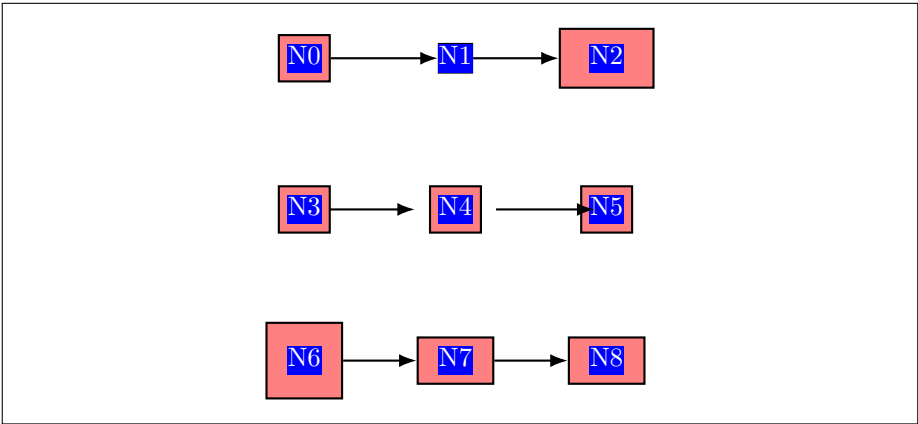
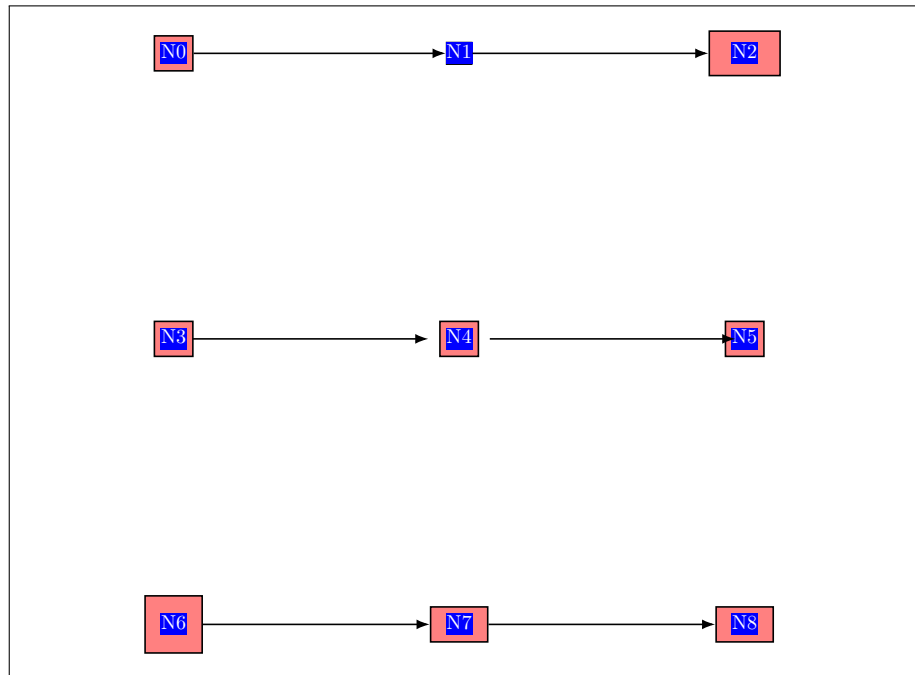


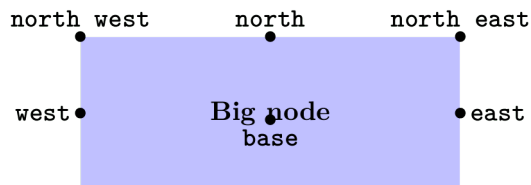
Figura 107

Invece, con `scale=5.0` e un `\resizebox` esterno l'immagine si presenta così:



3.15.2 Posizionamento (in breve)

I *punti di ancoraggio* di un nodo sono utilizzati anche per fissare la posizione del nodo in una data coordinata.



Di default, un nodo è ancorato al centro di una coordinata; altrimenti, si può specificare un punto di ancoraggio specifico tramite l'opzione *anchor*:

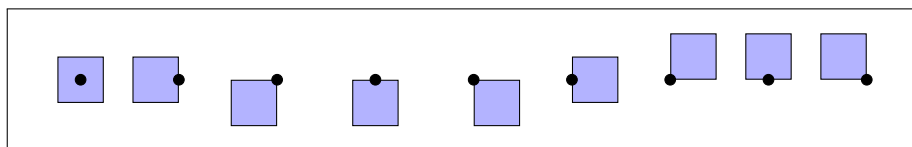


Figura 108

... Oppure tramite le opzioni di posizionamento *above*, *below*, *left*, *right* (e le opzioni combinate, e.g., *above right*):

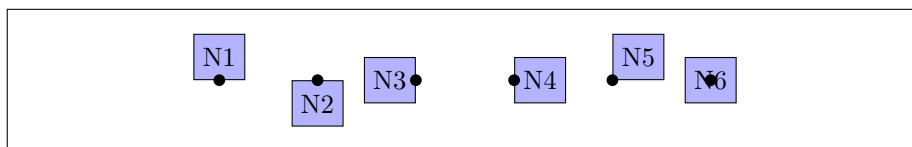


Figura 109

3.15.3 Archi ↻

Una linea tra due nodi si chiama tipicamente *arco*. In TikZ, gli archi si possono disegnare con `\path`, usando le tipiche operazioni per linee dritte, curve, etc. Come anticipato, gli archi uscenti da, e entranti in un certo nodo vengono disegnati non a partire dalla coordinata di posizione del nodo, ma dai **punti di ancoraggio**:

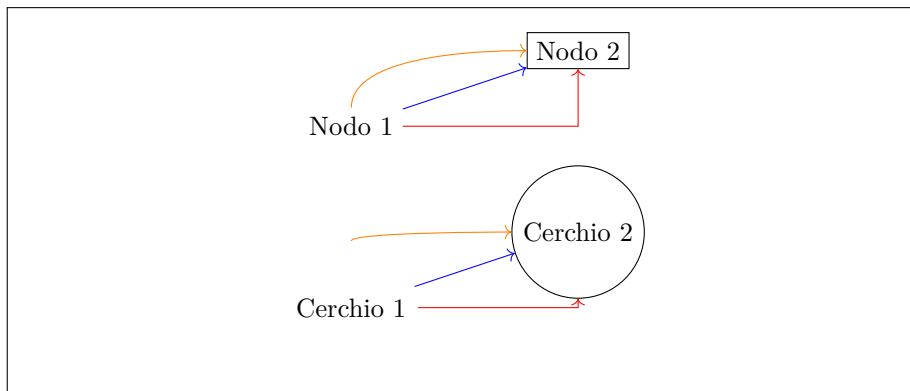


Figura 110

Oltre ai path-comandi che già conosciamo, introduciamo il conveniente path-comando `to ↻`. Funziona come una `Line-to`, ma permette di specificare curvature in maniera più elastica:

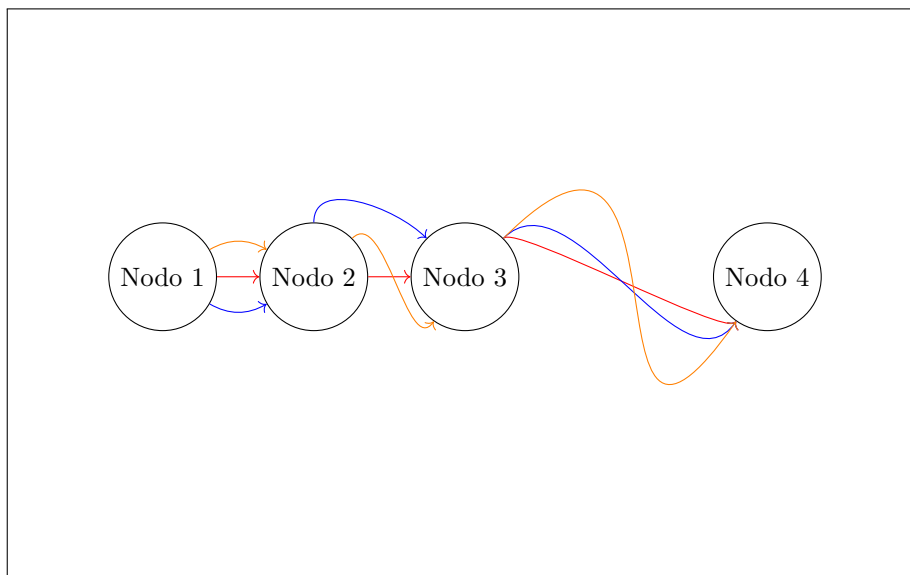



Figura 111

Gestione particolare dei *loop* :

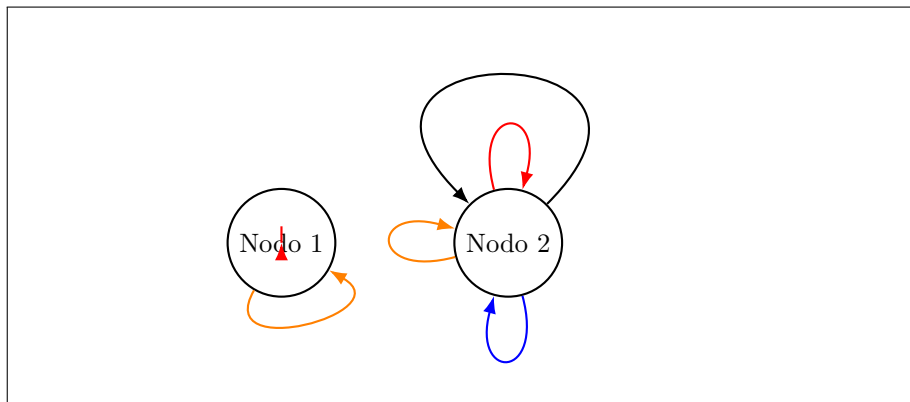


Figura 112

Una caratteristica interessante dei path-comandi è la possibilità di creare nodi *lungo* la linea disegnata:

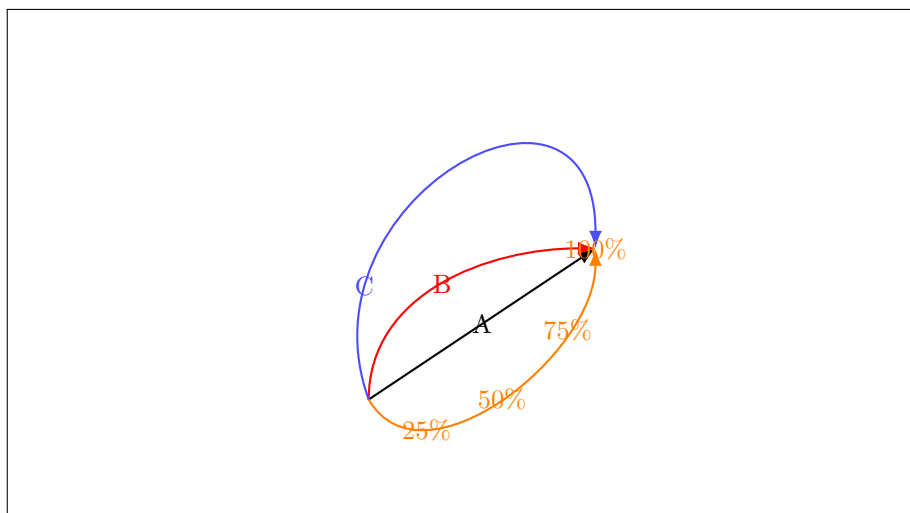


Figura 113

Si possono ottenere risultati eleganti controllando il posizionamento dei nodi. Per esempio, combinando le opzioni *sloped*, *above*, e *below*:

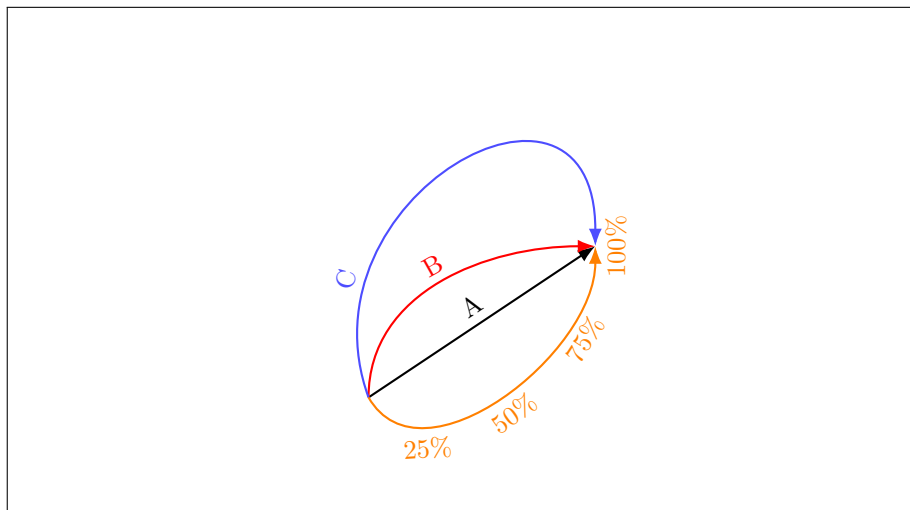


Figura 114

Più nodi su una linea possono essere creati sistematicamente con `node` `foreach` :

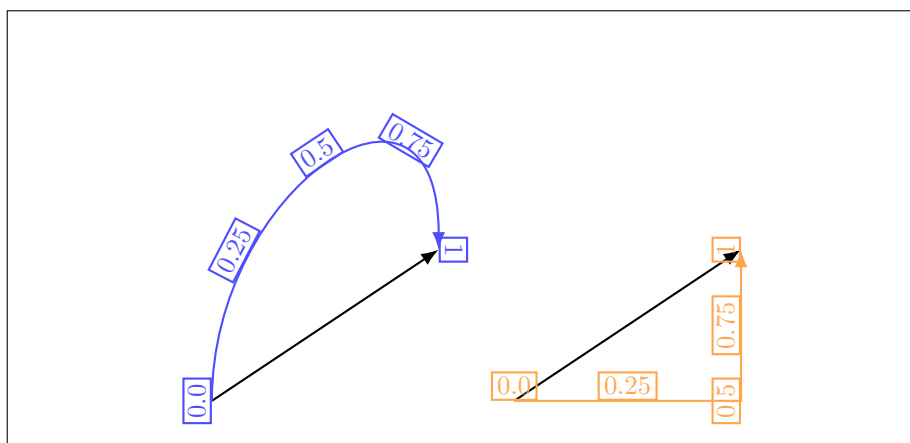



Figura 115

3.15.4 Accenni allo scoping, chiavi `every` e stili

Come anticipato, l'environment `scope`  permette di limitare l'effetto di alcune opzioni a un blocco specifico di comandi. Questo è molto utile se utilizzato in coppia con chiavi come `every node/.style`, che permette di assegnare stili a tutti i nodi nello stesso scope dove viene applicato:

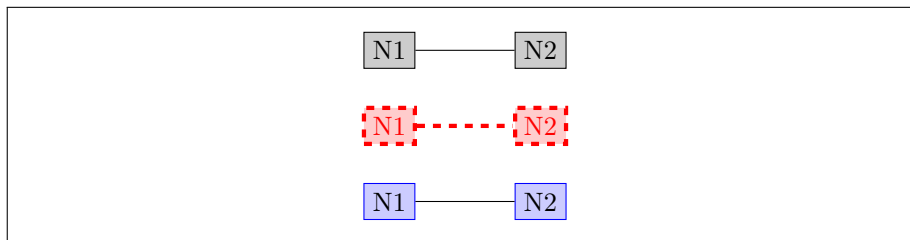



Figura 116

Si noti che, in realtà, gli scope limitano anche la visibilità esterna dei nodi definiti al loro interno (nella figura precedente N1 e N2 sono stati definiti più volte a diversi livelli). Simili a `every node` e `every path`, esistono anche `every to` e `every loop` e `every scope` e `every picture`.

Una funzionalità molto utile è quella di definire degli stili personalizzati. Questo si può fare tramite `tikzstyle` , e gli stili possono anche essere parametrici, come in:

```
\tikzstyle{my style}=[draw=red,fill=red!20]
\tikzstyle{my style}[red]=[draw=#1,fill=#1!20]
```

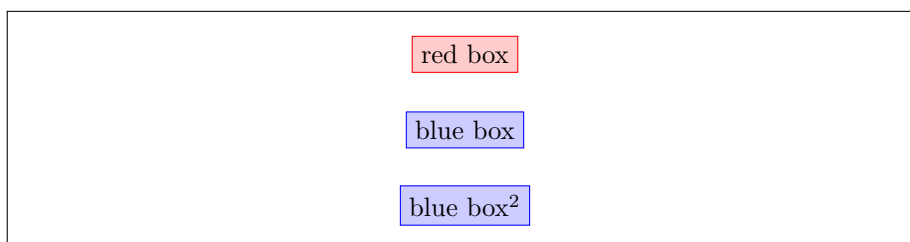


Figura 117

Usando gli stili è possibile limitare ripetizioni e rindondanze nel codice, e si può rendere un codice TikZ contemporaneamente più leggibile e facile da modificare.

3.15.5 Esercizio

Riprodurre il seguente diagramma:

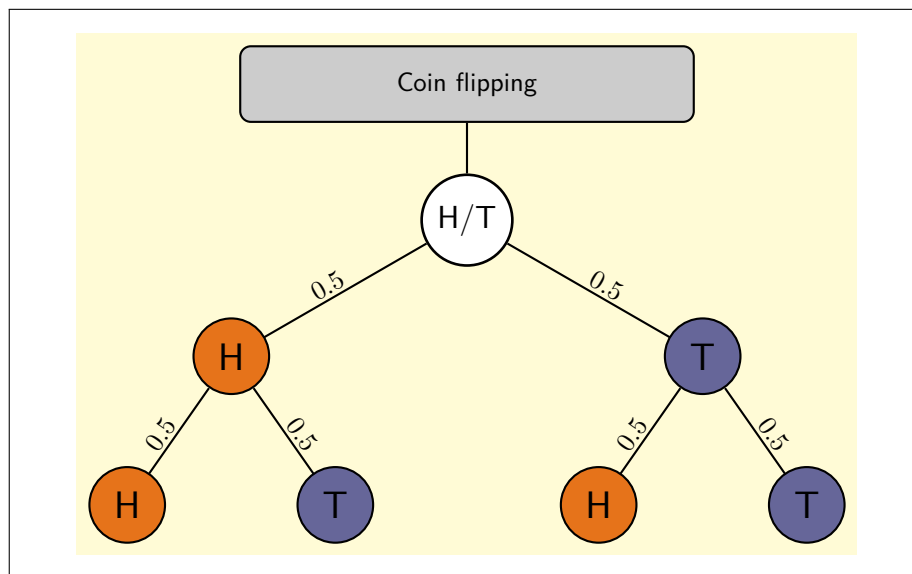


Figura 118

3.15.6 Esercizio

Riprodurre il seguente diagramma:

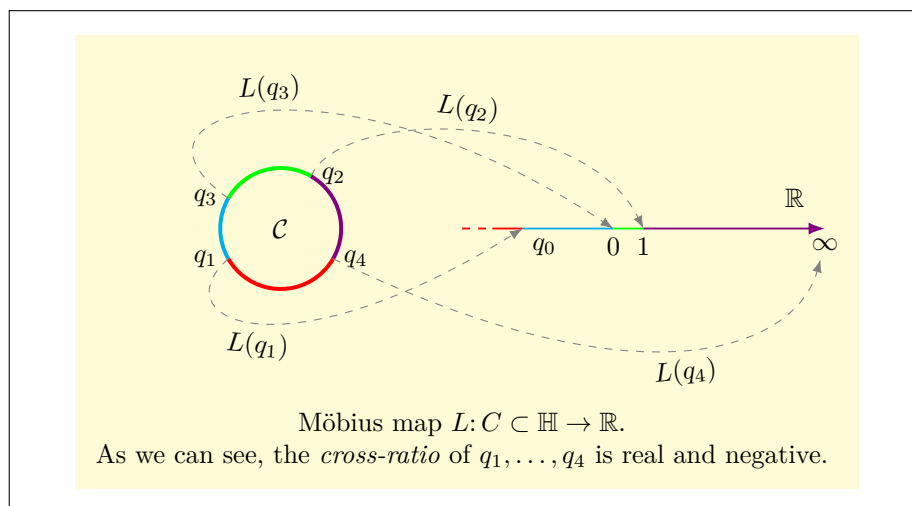


Figura 119

Lecture aggiuntive:

- Nodi di testo ☞;
- osizionamento auto e swap ☞;

- Grafi \mathcal{G} ;
- Grafi ad albero \mathcal{G} .

3.15.7 Esercizio

Riprodurre il diagramma che mostra la struttura di un nodo. Sapendo che dentro ad un nodo può andare una `tikzpicture`! Se si stilizza appropriatamente la geometria dei nodi, è possibile farlo usando tre o quattro nodi *innestati* in `tikzpicture`.

3.15.8 Doppio bordo

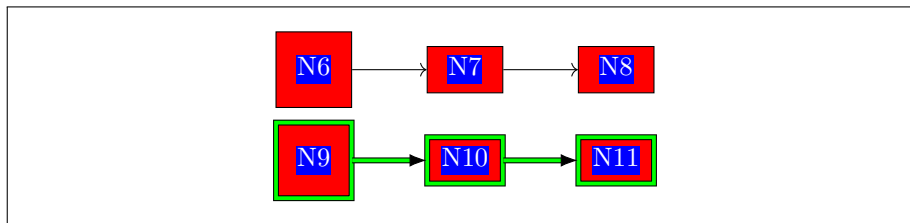


Figura 120

3.16 Plot di funzione

Il path-comando `plot` si usa per plottare funzioni. L'opzione `domain` controlla il dominio della variabile indipendente:

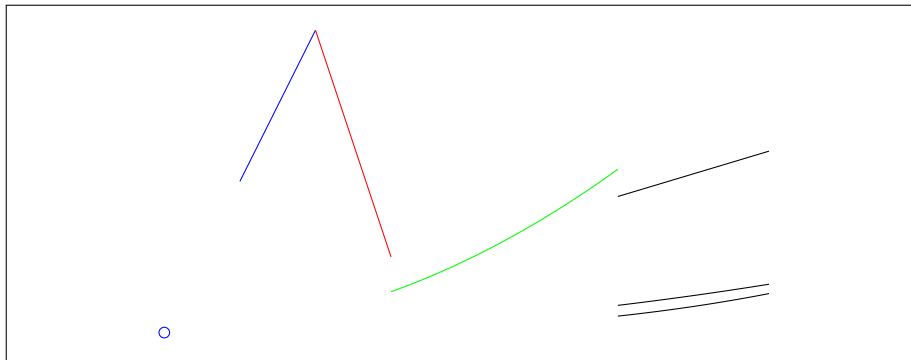


Figura 121

Si possono usare funzioni del motore matematico (nota: quando si usano parentesi tonde, è meglio chiudere l'espressione in un gruppo con `{ e }` per preservare le parentesi tonde):

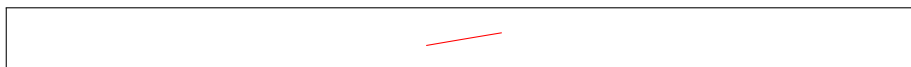


Figura 122

Per funzioni trigonometriche (e.g., `sin(\x)`) la variabile viene interpretata in gradi sessagesimali. Si può, però, indicare che l'angolo è da interpretare in radianti: `sin(\x r)`.

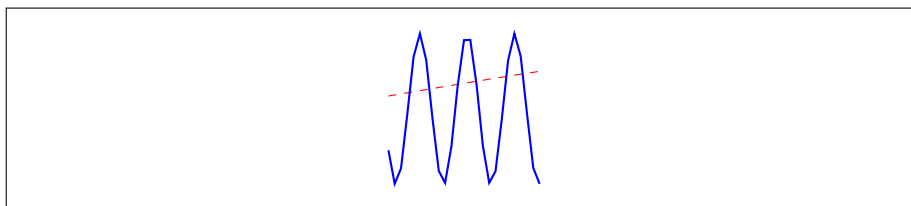


Figura 123

Internamente, `plot` fa un certo numero di iterazioni per calcolare le coppie di coordinate, e poi esegue delle Line-to per unirle. Di default, vengono fatte 15 iterazioni equispaziate, ma questo a volte risulta troppo approssimativo; si può, allora, specificare un sampling più fine tramite l'opzione `samples`.

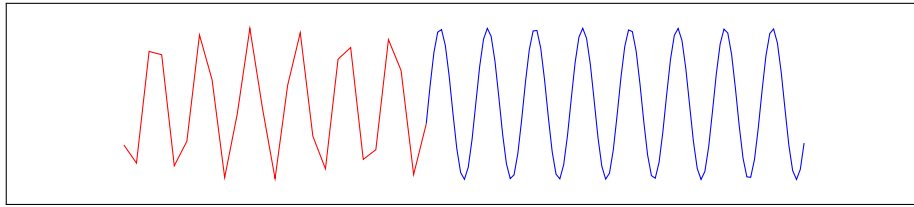


Figura 124

Sempre in ottica di ottenere **plot** meno spezzati, l'opzione **smooth** indica a draw di unire i samples non con spezzate ma con linee curve polinomiche. Attenzione, l'interpolazione curva può dare risultati inaspettati:

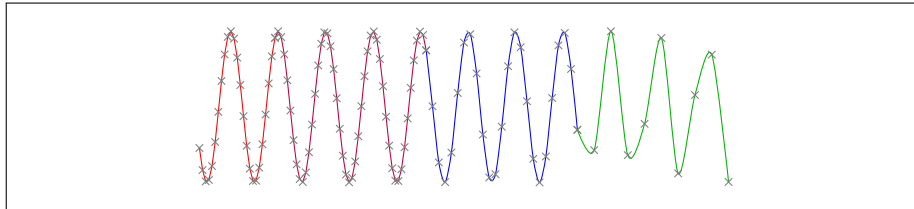


Figura 125

Come si vede nella precedente figura, è possibile aggiungere dei segni (o *mark*) in corrispondenza di ciascun sample del plot \mathcal{P} . Le opzioni per controllare lo stile dei segni sono: **mark**, **mark size**, **mark options**. Inoltre, si può limitare quali sample segnare con **mark repeat** e **mark phase** o, alternativamente, **mark indices**.

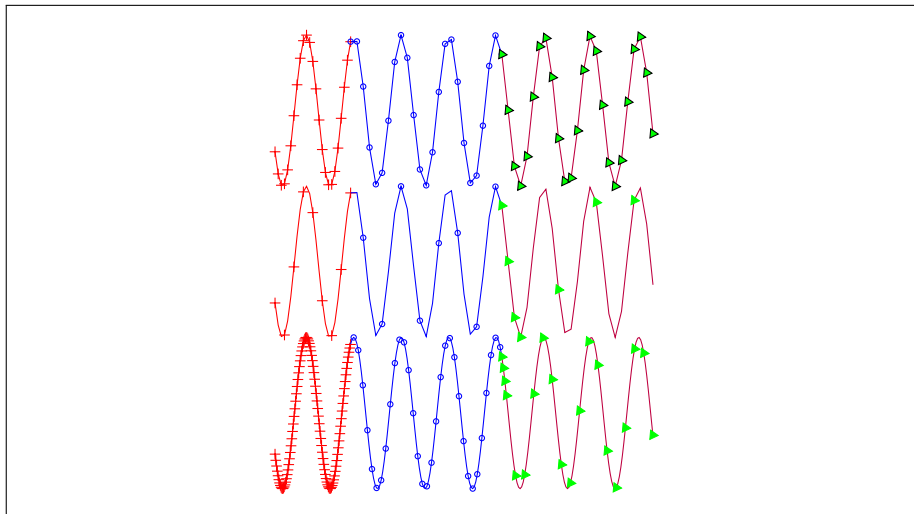



Figura 126

Diversi tipi di marks sono disponibili con `\usetikzlibrary{plotmarks}` .

Di default, la variabile indipendente è assegnata a `\x`, ma il nome di questa macro si può controllare con l'opzione `variable`, per maggiore leggibilità. Questo è particolarmente utile quando si vogliono plottare funzioni parametriche:

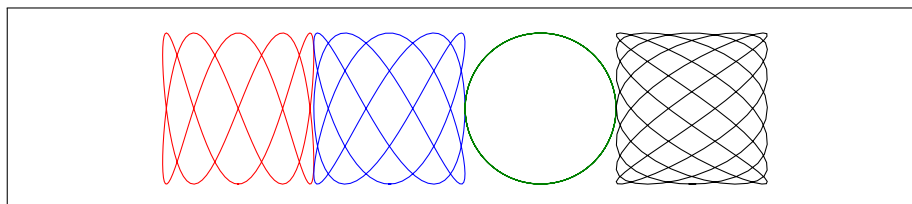


Figura 127

Oltre a **smooth**, esistono altri tipi di giunture tra samples \mathcal{S} :

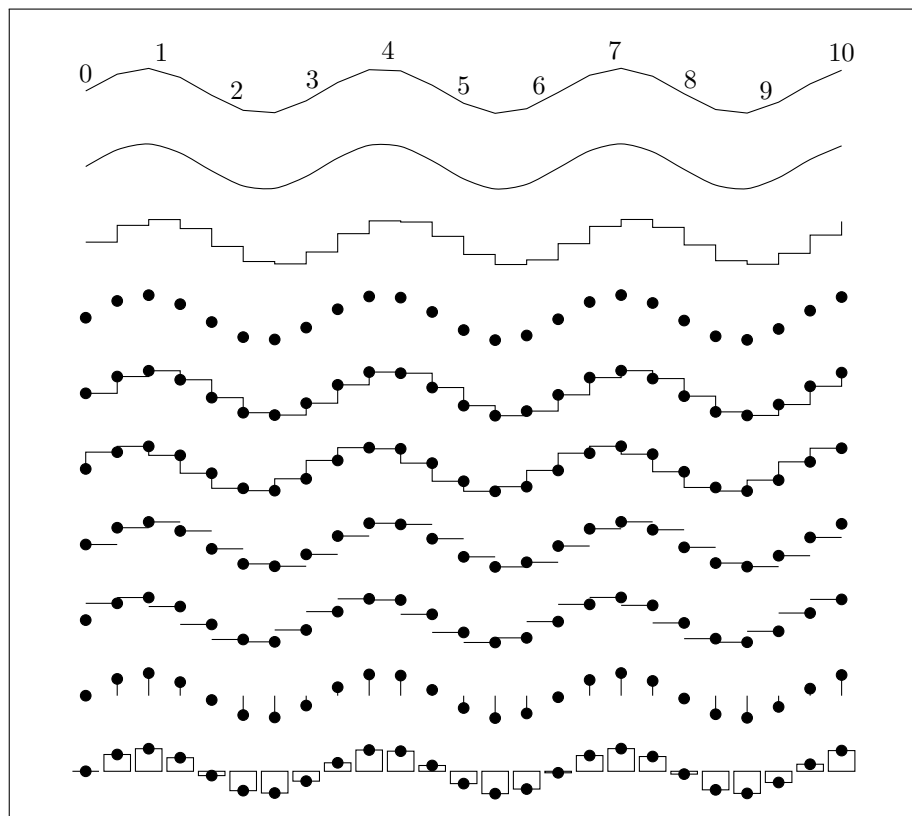


Figura 128

Come per altri path-comandi, alla fine di un `plot` si può costruire un `node` o una `coordinate` tramite i path-comandi appositi.

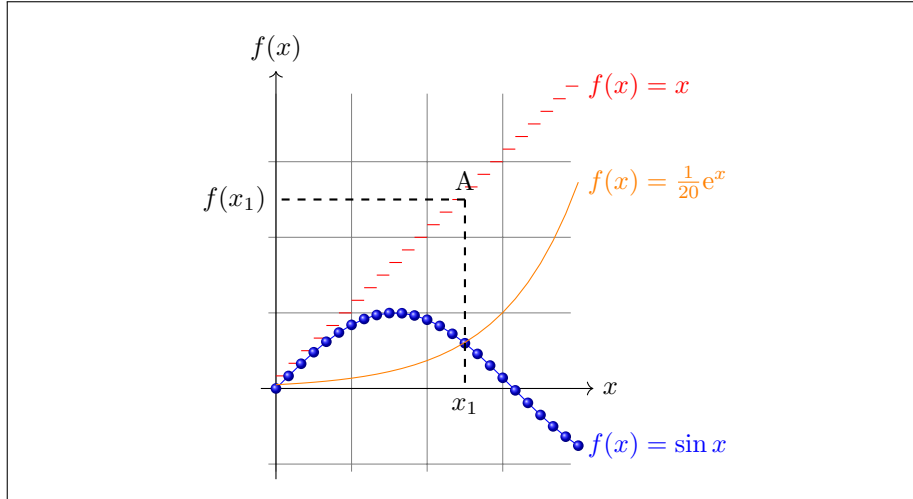


Figura 129

3.16.1 Esercizio

Replicare il seguente plot della funzione $f(x) = 0.2x^3 - 2.4x$ per $x \in [-3, 4]$, facendo uso di nodi ancorati (e.g., `node[left,...]`), `foreach`, e `declare function`:

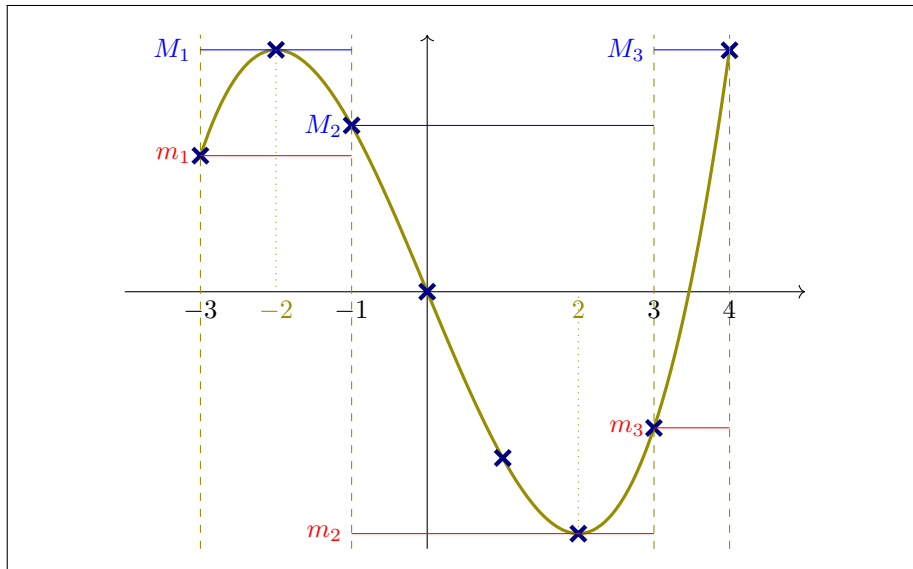


Figura 130

3.16.2 Esercizio

Replicare la curva

```
\draw (1, 1.7)
  .. controls (1.6, 2.4) .. (1.8, 2.4)
  .. controls (2.3, 2.4) and (2.4, 0.5) .. (3, 0.5)
  .. controls (3.5, 0.5) and (4.1, 3.1) .. (4.6, 3.1)
  .. controls (5, 3.1) and (5, 1.5) .. (5.4, 1.5)
  .. controls (5.6, 1.5) .. (6, 2.1);
```

come raffigurato nel seguente plot:

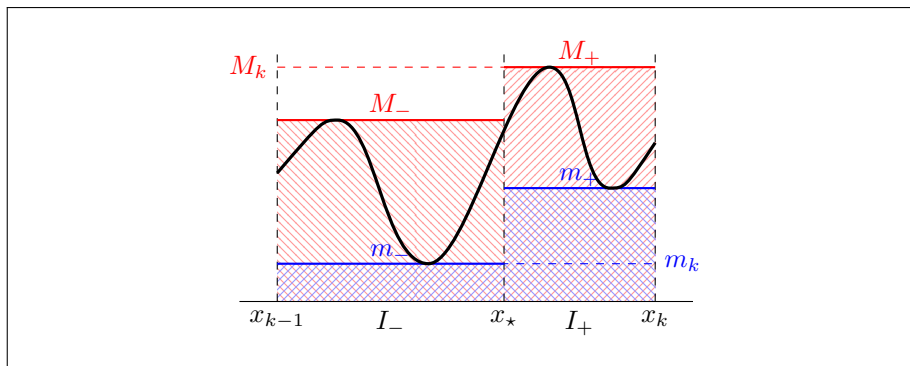


Figura 131

3.16.3 Esercizio

Replicare il seguente grafico raffigurante una distribuzione gaussiana:

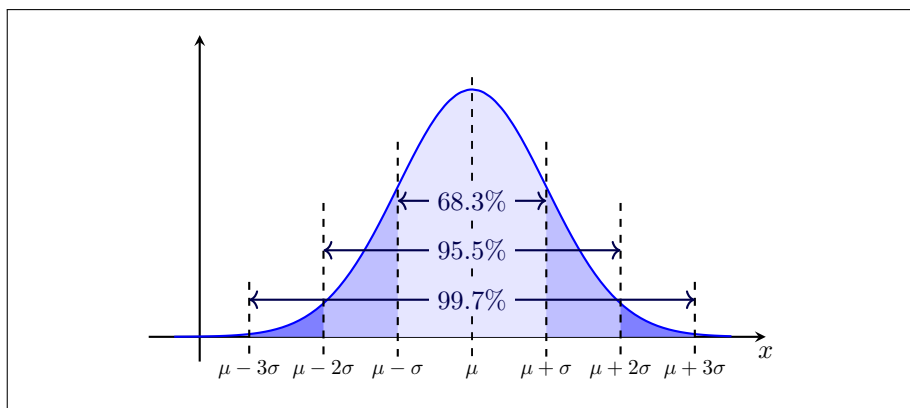


Figura 132

4 Ringraziamenti

Ringrazio il Dr. Jonathan Franceschi, autore di parte del materiale da cui ho attinto nella stesura, e il Prof. Damiano Foschi per l'aiuto nell'organizzazione del corso.