ALGORITMI E STRUTTURE DI DATI

Capitoli

Varia

Algebra lineare

Trigonometria piana

Geometria analitica

Trigonometria sferica

Teoria dei numeri

R

Python

Algoritmi elementari

Compressione di dati

Esercizi per gli scritti

I. INTRODUZIONE

Che cos'è la matematica?

Dividiamo questa domanda in due sottodomande, cercando di indicare prima i costituenti elementari della matematica, poi come la matematica deve essere usata.

I componenti elementari del ragionamento matematico sono le coppie ipotesi-tesi; in questo senso la matematica non conosce affermazioni assolute, ma soltanto proposizioni che si compongono ogni volta di un preciso elenco delle ipotesi che vengono fatte, e poi di una altrettanto precisa specificazione dell'enunciato che secondo quella proposizione ne consegue. A questo punto non è detto che la proposizione sia valida, bisogna ancora dimostrarla, e ciò significa, nella matematica, dimostrare che la tesi segue dalle ipotesi unite agli assiomi e ai risultati già ottenuti e alle regole logiche che dobbiamo applicare. Gli assiomi sono enunciati che vengono messi all'inizio di una teoria, senza dimostrazione; ogni altro enunciato deve essere invece dimostrato.

*È importante che bisogna sempre dimostrare una proposizione - che è sempre nella forma ipotesi-tesi! - nella sua interezza, cioè che si tratta di dimostrare la validità dell'implicazione e non la validità della tesi. L'enunciato **A implica B** può essere vero, anche se **B** non e vero. Ad esempio in logica si impara che, se l'ipotesi **A** è falsa, allora la proposizione **A implica B** è sempre vera. Quindi l'affermazione **se la luna è un cubo, allora io mi chiamo Marco** è sempre vera, indipendentemente da come mi chiamo io. Nella pratica matematica ciò significa che da una premessa errata si può, con un po' di pazienza, dedurre qualunque cosa.*

*La validità si riferisce quindi sempre a tutta la proposizione **A implica B**, cioè alla coppia ipotesi-tesi.*

Mentre il matematico puro cerca soprattutto di arricchire l'edificio delle teorie matematiche con nuovi concetti o con dimostrazioni, talvolta assai difficili, di teoremi, il matematico applicato deve anche saper usare la matematica. Nelle scienze naturali e sociali, le quali pongono problemi molto complessi, uno dei compiti più importanti è spesso la separazione degli elementi essenziali di un fenomeno dagli aspetti marginali. In queste scienze le informazioni disponibili sono quasi sempre incomplete, cosicché possiamo ogni volta descrivere soltanto una piccola parte della realtà. Anche quando disponiamo di conoscenze dettagliate, queste si presentano in grande quantità, sono complesse e multiformi e richiedono concetti ordinatori per poterle interpretare. Ciò significa che bisogna estrarre e semplificare.

Un modello matematico di un fenomeno ha soprattutto lo scopo di permettere di comprendere meglio quel fenomeno, quindi di metterne in evidenza cause e effetti e comportamenti quantitativi, di individuarne i tratti essenziali e i meccanismi fondamentali. In un certo senso la matematica consiste di tautologie, e nel modello matematico si tenta di evidenziare le tautologie contenute nel fenomeno studiato. La teoria cerca di comprendere i processi e legami funzionali di un campo del sapere.

La mente umana pensa in modelli. Anche quando non facciamo matematica della natura, cerchiamo di comprendere la natura mediante immagini semplificate. La teoria inizia già nell'istante in cui cominciamo a porci la domanda quali siano gli aspetti essenziali di un oggetto o di un fenomeno. La matematica non è dunque altro che un modo sistematico e controllato di eseguire questi processi di astrazione e semplificazione costantemente attivi nel nostro intelletto.

Il modello matematico, una volta concepito, se sviluppato correttamente, si mostra poi di una esattezza naturale che spesso impressiona l'utente finale che è tentato di adottare gli enunciati matematici come se essi corrispondessero precisamente ai fenomeni modellati. Ma ciò non è affatto vero: La precisione del modello matematico è soltanto una precisione interna, tautologica, e la semplificazione, quindi verità approssimata e parziale, che sta all'inizio del modello, si conserva, e più avanza lo sviluppo matematico, maggiore è il pericolo che iterando più volte l'errore, questo sia cresciuto in misura tale da richiedere un'interpretazione estremamente prudente dei risultati matematici. Proprie le teorie più avanzate, più belle quindi per il matematico puro, sono spesso quelle più lontane dalla realtà. Questo automatismo della matematica può essere però anche fonte di nuovi punti di vista e svelare connessioni nascoste.

Un modello matematico è perciò solo un ausilio per la riflessione, per controllare il contenuto e la consistenza logica di un pensiero o di una ricerca. In altre parole, modelli sono strumenti intellettuali e non si possono da essi aspettare descrizioni perfette della realtà. Essi non forniscono risposte complete, ma indicano piuttosto quali siano le domande che bisogna porre.

L'astrattezza intrinseca della matematica comporta da un lato che essa rimanga sempre diversa dalla realtà, offre però dall'altro lato la possibilità di generalizzare i risultati ottenuti nelle ricerche in un particolare campo applicativo o anche uno strumento della matematica pura a problemi apparentemente completamente diversi, se questi hanno proprietà formali in comune con il primo campo.

L'alfabeto greco

alfa	α	A
beta	β	B
gamma	γ	Γ
delta	δ	Δ
epsilon	ε	E
zeta	ζ	Z
eta	η	H
theta	θ	Θ
iota	ι	I
kappa	$\varkappa$	K
lambda	λ	Λ
mi	μ	M
ni	ν	N
xi	ξ	Ξ
omikron	o	O
pi	π	Π
rho	ρ	P
sigma	σ,ς	Σ
tau	τ	T
ypsilon	υ	Υ
fi	φ	Φ
chi	χ	X
psi	ψ	Ψ
omega	ω	Ω

Sono 24 lettere. Per ogni lettera sono indicati il nome italiano, la forma minuscola e quella maiuscola. La sigma minuscola ha due forme: alla fine della parola si scrive ς, altrimenti σ. In matematica si usa solo la σ.

Mikros ($\mu\iota\varkappa\rho\acute{o}\varsigma$) significa piccolo, megas ($\mu\acute{\varepsilon}\gamma\alpha\varsigma$) grande, quindi la omikron è la *o* piccola e la omega la *o* grande.

Le lettere greche vengono usate molto spesso nella matematica, ad esempio $\sum_{k=0}^{3} a_k$ è un'abbreviazione per la somma $a_0+a_1+a_2+a_3$ e $\prod_{k=0}^{3} a_k$ per il prodotto $a_0a_1a_2a_3$, mentre A^i_α è un oggetto con due indici, per uno dei quali abbiamo usato una lettera greca.

Fusione di dizionari

Con `tab.update(tab1)` il dizionario `tab` viene fuso con il dizionario `tab1`. Ciò significa più precisamente che alle voci di `tab` vengono aggiunte, con i rispettivi valori, le voci di `tab1`; voci già presenti in `tab` vengono sovrascritte. Possiamo così in particolare unire due dizionari, come faremo nella funzione `calbin` a pagina 43.

```
tab=dict(a=1, b=2)
tab1=dict(b=300, c=4, d=5)

tab.update(tab1)
print tab
# {'a': 1, 'c': 4, 'b': 300, 'd': 5}
```

dict

La funzione `dict` può essere usata, con alcune varianti di sintassi, per generare dizionari da sequenze già esistenti:

```
d = dict(); print d
# {} - dizionario vuoto

a=[('u',1), ('v',2)]
d=dict(a); print d
# {'u': 1, 'v': 2}
```

Problemi semplici difficili

L'uso di modelli nelle scienze naturali oppure in ingegneria e economia comporta due difficoltà: In primo luogo bisogna concepire modelli adatti e interpretare i risultati matematici ottenuti nel modo adeguato. Ma le difficoltà possono essere anche di carattere matematico. Daremo adesso alcuni esempi di problemi puramente matematici di semplice formulazione, in cui non si pongono sottili questioni di interpretazione, la cui soluzione è però talmente difficile che i matematici non sanno nemmeno da che parte cominciare per risolverli. Di questi problemi ne esistono migliaia. Alcuni sono poco più che giochetti, ma problemi altrettanto difficili appaiono in molti modelli matematici della fisica e della biologia.

Il problema dei tre corpi

Il problema dei tre corpi, cioè il comportamento di tre corpi sufficientemente vicini e di massa approssimativamente uguali sotto l'influsso delle forze gravitazionali che essi esercitano l'uno sull'altro, è matematicamente estremamente difficile e ha più di una soluzione - pochi anni fa sono state trovate nuove soluzioni. Sole, terra e luna non sono un esempio, perché il sole ha massa molto più grande degli altri due, cosicché il sistema può essere descritto dal moto del baricentro di terra e luna attorno al sole, e poi della luna attorno alla terra, si decompone quindi in due problemi di due corpi, e questi obbediscono alle leggi di Kepler.

Numeri primi

Definizione 2.1. Un numero naturale diverso da 1 si dice *primo*, se non è divisibile (senza resto) da nessun numero naturale tranne 1 e se stesso.

Ad esempio i primi ≤ 40 sono $2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37$.

Lemma 2.2. *Ogni numero naturale $n > 1$ è diviso da un numero primo.*

Dimostrazione. Infatti i divisori di n sono tutti compresi tra 1 e n, e certamente tra quelli che sono maggiori di 1 ne esiste uno più piccolo, che chiamiamo p. Ogni divisore di p però è anche divisore di n, quindi p non può avere un divisore d con $1 < d < p$, altrimenti d sarebbe un divisore di n più piccolo di p e maggiore di 1. Ciò significa che p è primo.

Teorema 2.3. *Esiste un numero infinito di numeri primi.*

Dimostrazione. Ecco la dimostrazione, risalente a Euclide e quindi vecchia circa 2300 anni!

Assumiamo che esista solo un numero finito di numeri primi e dimostriamo che si arriva a una contraddizione:

In quella ipotesi possiamo formare il prodotto di tutti questi primi che chiamiamo P. $P + 1$ è certamente maggiore di 1, quindi per il lemma 2.2 esiste anche un numero primo p che divide $P + 1$, ad esempio $P + 1 = px$. D'altra parte però p, essendo primo, deve essere uno dei fattori di P (che è appunto il prodotto di tutti i primi), e quindi p divide anche P, ad esempio $P = py$.

Ciò implica $1 = P + 1 - P = px - py = p(x - y)$, quindi p divide 1, ma allora $p \leq 1$, una contraddizione.

Il problema dei primi gemelli

Due numeri primi che si distinguono di 2, come 3 e 5, 11 e 13, 17 e 19, 29 e 31, 41 e 43, si chiamano primi gemelli. Nessun matematico è finora riuscito a dimostrare che esistono infiniti primi gemelli, anche se, come si vede dagli esempi, sembra che ce ne siano parecchi e si è tutti convinti che devono essere infiniti.

Numeri perfetti

Un numero naturale si dice perfetto, se è uguale alla somma dei suoi divisori diversi da se stesso.

Ad esempio $6 = 1 + 2 + 3$ e $28 = 1 + 2 + 4 + 7 + 14$ sono numeri perfetti. Esistono infiniti numeri perfetti? Esiste almeno un numero perfetto dispari? Non si sa.

Il problema del $3x + 1$

Questo è sicuramente il problema più inutile della matematica. Sembra incredibile che, da quando è stato posto più di 60 anni fa, nessuno sia riuscito a risolverlo.

Partiamo con un numero naturale maggiore di 1 qualsiasi. Se è pari, lo dividiamo per 2, altrimenti lo moltiplichiamo per 3 e aggiungiamo 1. Con il numero così ottenuto ripetiamo l'operazione, e ci fermiamo solo quando arriviamo a 1. Ad esempio partendo con 7 otteniamo $22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1$.

In genere si arriva a 1 molto presto; però se si parte con 27, il ciclo è molto lungo e bisogna effettuare più di 100 iterazioni. Provare.

Sembra che alla fine l'algoritmo si fermi sempre, cioè che prima o poi si arrivi sempre ad 1. Ma nessuno riesce a dimostrarlo.

Questi esempi, di cui l'ultimo è particolarmente semplice, mostrano che anche quando si riesce a trasformare un problema scientifico, ingegneristico o economico in un problema matematico, non è affatto detto che la matematica riesca a fornire una soluzione. Spesso si renderanno necessarie ulteriori semplificazioni perché il modello più realistico può essere formulato ma non risolto matematicamente.

La congettura di Fermat

Altri problemi semplici difficili sono stati risolti recentemente. Il più famoso di questi era la congettura di Fermat: Se n è un numero naturale > 2, non è possibile risolvere in numeri naturali $x, y, z \neq 0$ l'equazione $x^n + y^n = z^n$. Questa congettura è stata dimostrata (dopo secoli di tentativi) attorno al 1995 da Andrew Wiles.

Il teorema di Green-Tao

Ma il risultato più sensazionale della matematica ottenuto negli ultimi decenni è il teorema di Green-Tao:

Per ogni $n \in \mathbb{N} + 1$ esistono n primi equidistanti, esistono cioè $p \in \mathbb{N}$ e $d \in \mathbb{N} + 1$ tale che $p, p + d, p + 2d, \ldots, p + (n - 1)d$ sono tutti primi.

La dimostrazione fornita nel 2004 da Ben Green (nato 1977) e Terence Tao (nato 1975, detto il Mozart della matematica, medaglia Fields nel 2006) è complicata e consiste in una sofisticata combinazione di strumenti della teoria dei numeri, del calcolo combinatorio, dell'analisi armonica, della teoria delle probabilità e della teoria ergodica. Le tecniche usate sono molto potenti e potrebbero portare addirittura a una dimostrazione della congettura dei numeri primi gemelli.

Il test di primalità appartiene a P

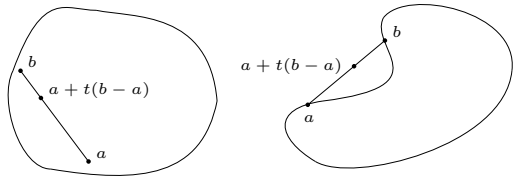
Un problema pratico importante, ad esempio in crittografia, è determinare se un numero naturale n è primo. Per n molto grande (più di 100 cifre decimali) ciò non è banale e solo nel 2002 Manindra Agrawal (nato 1966) con due studenti ha trovato un algoritmo efficiente per il test di primalità. Questo algoritmo è sorprendentemente elementare e facile da programmare.

I matematici distinguono algoritmi (teoricamente) efficienti (detti di classe P) e algoritmi non efficienti (detti di classe NP). Agrawal, Kayal e Saxena hanno dimostrato che il loro algoritmo è di classe P.

In verità è un problema aperto se veramente vale P≠NP, come molti esperti pensano, anche se non è detto che sia così.

La quadratura insiemistica del cerchio

Definizione 3.1. V sia uno spazio vettoriale (ad esempio $V = \mathbb{R}^m$) ed $A \subset V$. A si dice *convesso* se per ogni $a, b \in A$ tutto il segmento di retta tra a e b è contenuto in A, cioè se $a + t(b - a) \in A$ per ogni $t \in [0, 1]$.

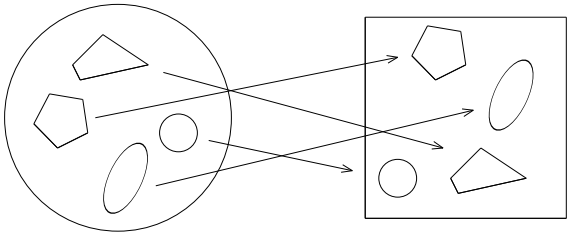


Osservazione 3.2. Per ogni sottoinsieme convesso $A \subset \mathbb{R}^m$ è definito il *volume* $\lambda(A)$, detto anche *misura di Lebesgue* di A.

Per $m = 2$ $\lambda(A)$ si chiama anche l'*area* di A.

Teorema 3.3. *Siano $m \geq 1$ ed A, B sottoinsiemi convessi e limitati di $\mathbb{R}^m$ tali che $\lambda(A) = \lambda(B) > 0$. Allora esistono $A_1, \ldots, A_s \subset A$ e $B_1, \ldots, B_s \subset B$ tali che $A = A_1 \sqcup \ldots \sqcup B_s$, e traslazioni $g_1, \ldots, g_s$ di $\mathbb{R}^m$ tali che $B_1 = g_1(A_1), \ldots, B_s = g_s(A_s)$.*

Dimostrazione. Nel 1988 questo teorema è stato dimostrato da Miklos Laczkovich per il caso speciale che A sia un cerchio e B un quadrato *della stessa area* di A. Il problema era rimasto irrisolto per più di 50 anni ed era noto come il problema della *quadratura insiemistica del cerchio*. La sua soluzione era un'autentica sensazione. La dimostrazione è molto lunga, ma piuttosto concreta:



Per trovare queste traslazioni, Laczkovich ha usato considerazioni molto complicate che appartengono alla geometria dei numeri e alla teoria dell'uniforme distribuzione, un altro ramo della teoria dei numeri con molte applicazioni, ad esempio nella generazione di successioni casuali di numeri, nell'integrazione multidimensionale (metodi di Monte Carlo), nella crittografia.

La versione più generale del teorema è stata dimostrata dallo stesso autore pochi anni dopo.

Equazioni lineari in una incognita

Sono due gli algoritmi elementari fondamentali della matematica, l'*algoritmo di eliminazione di Gauss* per la risoluzione di sistemi di equazioni lineari e l'*algoritmo euclideo* per il calcolo del massimo comune divisore di due numeri interi.

Siano dati numeri reali a e b. Cercare di risolvere l'equazione

$$ax = b$$

nell'incognita x significa cercare tutti i numeri reali x per i quali $ax = b$. Per $a \neq 0$ la soluzione è unica e si ha $x = b/a$.

Cosa succede per $a = 0$? In tal caso bisogna cercare tutti gli x per cui $0 \cdot x = b$. Se anche $b = 0$, allora ogni x va bene. Se invece $b \neq 0$, allora nessun x può soddisfare l'equazione. Abbiamo quindi tre possibili situazioni:

coefficienti	insieme delle soluzioni
$a \neq 0$	$\{b/a\}$
$a = 0 = b$	$\mathbb{R}$
$a = 0 \neq b$	$\emptyset$

Alcune nozioni astratte

Definizione 3.4. Siano dati un insieme X ed una funzione $f : X \longrightarrow \mathbb{R}$. Denotiamo con $(f = 0)$ l'insieme degli zeri di f:

$$(f = 0) := \{x \in X \mid f(x) = 0\}$$

Date m funzioni $f_1, \ldots, f_m : X \longrightarrow \mathbb{R}$, denotiamo inoltre con $(f_1 = 0, \ldots, f_m = 0)$ oppure anche con $(f_1 = \ldots = f_m = 0)$ l'insieme $(f_1 = 0) \cap \cdots \cap (f_m = 0)$ degli zeri comuni di queste funzioni. Questa notazione è molto usata in calcolo delle probabilità.

Soprattutto quando una funzione f è data da una formula, la scriviamo talvolta nella forma $f = \bigcirc\limits_x f(x)$.

La funzione $\bigcirc\limits_x \sin(x^3 + 1)$ è quindi quella funzione che manda x nel seno di $x^3 + 1$, mentre $\bigcirc\limits_a \bigcirc\limits_x x^2 + a^2$ manda a nella funzione che manda x in $x^2 + a^2$.

Osservazione 3.5. *X sia un insieme. Con $\mathbb{R}^X$ si denota l'insieme di tutte le funzioni a valori reali definite su X. Possiamo formare somme di funzioni, moltiplicare funzioni con un numero reale o con un'altra funzione, e costruire combinazioni lineari di funzioni in questo spazio nel modo seguente.*

Siano $f, g, f_1, \ldots, f_m \in \mathbb{R}^X$ ed $\alpha, \alpha_1, \ldots, \alpha_m \in \mathbb{R}$. Allora:

$$f + g := \bigcirc_x f(x) + g(x)$$

$$\alpha f := \bigcirc_x \alpha f(x)$$

$$\alpha_1 f_1 + \cdots + \alpha_m f := \bigcirc_x \alpha_1 f_1(x) + \cdots + \alpha_m f_m(x)$$

Teorema 3.6. *Siano dati un insieme X ed m funzioni $f_1, \ldots, f_m : X \longrightarrow \mathbb{R}$. Siano $\alpha_1, \ldots, \alpha_m$ numeri reali con $\alpha_1 \neq 0$. Allora gli insiemi*

$$(f_1 = 0, \ldots, f_m = 0)$$

e

$$(\alpha_1 f_1 + \alpha_2 f_2 + \cdots + \alpha_m f_m = 0, f_2 = 0, \ldots, f_m = 0)$$

coincidono.

Dimostrazione. Per dimostrare l'uguaglianza tra i due insiemi, dobbiamo dimostrare che ogni elemento del primo insieme è anche elemento del secondo, e che ogni elemento del secondo insieme è elemento del primo.

Sia quindi x un elemento fissato di X.

(1) Sia $f_1(x) = 0, \ldots, f_m(x) = 0$. È chiaro che allora anche

$$\begin{aligned} &\alpha_1 f_1(x) + \alpha_2 f_2(x) + \cdots + \alpha_m f_m(x) = 0 \\ &f_2(x) = 0 \\ &\cdots \\ &f_m(x) = 0 \end{aligned}$$

(2) Sia viceversa

$$\begin{aligned} &\alpha_1 f_1(x) + \alpha_2 f_2(x) + \cdots + \alpha_m f_m(x) = 0 \\ &f_2(x) = 0 \\ &\cdots \\ &f_m(x) = 0 \end{aligned}$$

Dobbiamo dimostrare che $f_1(x) = 0$. Ma se sostituiamo $f_2(x) = 0, \ldots, f_m(x) = 0$ nella prima equazione, vediamo che $\alpha_1 f_1(x) = 0$.

Qui possiamo adesso applicare la nostra ipotesi che $\alpha_1 \neq 0$. Essa implica $f_1(x) = 0$.

Attenzione: Il ragionamento vale soltanto se sappiamo che $\alpha_1 \neq 0$!

Nota 3.7. Nel seguito avremo $X = \mathbb{R}^2$ oppure, più in generale, $X = \mathbb{R}^n$. Nel primo caso scriveremo gli elementi di X nella forma (x, y), cosicché x denoterà la prima coordinata di un elemento e non l'elemento stesso.

Gli elementi di $\mathbb{R}^3$ saranno scritti nella forma (x, y, z), gli elementi di $\mathbb{R}^n$ nella forma $x = (x_1, \ldots, x_n)$. Questo passaggio da una notazione all'altra è frequente e diventerà presto familiare.

II. SISTEMI LINEARI

Due equazioni lineari in due incognite

Siano dati numeri reali $a_1, b_1, c_1, a_2, b_2, c_2$. Risolvere il sistema

$$\begin{aligned} a_1x + b_1y &= c_1 \\ a_2x + b_2y &= c_2 \end{aligned}$$

significa trovare tutte le coppie (x, y) di numeri reali che soddisfano entrambe le equazioni. Per poter applicare il teorema 3.6 introduciamo le funzioni $f_1, f_2 : \mathbb{R}^2 \longrightarrow \mathbb{R}$ definite da

$$\begin{aligned} f_1(x, y) &:= a_1x + b_1y - c_1 \\ f_2(x, y) &= a_2x + b_2y - c_2 \end{aligned}$$

cosicché l'insieme delle soluzioni cercate coincide con $(f_1 = f_2 = 0)$. Lasciamo come esercizio il caso molto facile che $a_1 = 0$.

Assumiamo quindi che $a_1 \neq 0$ e definiamo la funzione

$$f_3 := a_1f_2 - a_2f_1$$

Per il teorema 3.6 abbiamo $(f_1 = f_2 = 0) = (f_1 = f_3 = 0)$, perché è f_2 che sicuramente appare con un coefficiente $\neq 0$ in f_3.

Scritta per esteso l'equazione $f_3 = 0$ diventa

$$\underline{a_1a_2x} + a_1b_2y - a_1c_2 - \underline{a_2a_1x} - a_2b_1y + a_2c_1 = 0$$

cioè

$$(a_1b_2 - a_2b_1)y = a_1c_2 - a_2c_1$$

e quindi le soluzioni del sistema originale coincidono con le soluzioni del sistema

$$\begin{aligned} a_1x + b_1y &= c_1 \\ (a_1b_2 - a_2b_1)y &= a_1c_2 - a_2c_1 \end{aligned}$$

Il numero $a_1b_2 - a_2b_1$ si chiama il *determinante* del sistema; lasciamo ancora come esercizio il caso che il determinante si annulli; se è invece $\neq 0$, allora la seconda equazione significa che

$$y = \frac{a_1c_2 - a_2c_1}{a_1b_2 - a_2b_1}.$$

Se per numeri reali a, b, c, d poniamo $\begin{vmatrix} a & b \\ c & d \end{vmatrix} := ad - bc$, possiamo scrivere

$$y = \frac{\begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}}$$

Vediamo che anche il numeratore ha la forma di un determinante; infatti si ottiene dal denominatore sostituendo per la seconda colonna la colonna che costituisce il lato destro del sistema.

A questo punto possiamo calcolare anche x. Ricordando che $a_1 \neq 0$, otteniamo

$$\begin{aligned} x &= \frac{c_1 - b_1y}{a_1} = \frac{c_1 - b_1\dfrac{a_1c_2 - a_2c_1}{a_1b_2 - a_2b_1}}{a_1} \\ &= \frac{a_1b_2c_1 - \underline{a_2b_1c_1} - b_1a_1c_2 + \underline{\underline{b_1a_2c_1}}}{a_1(a_1b_2 - a_2b_1)} \\ &= \frac{a_1b_2c_1 - b_1a_1c_2}{a_1(a_1b_2 - a_2b_1)} = \frac{b_2c_1 - b_1c_2}{a_1b_2 - a_2b_1} = \frac{\begin{vmatrix} c_1 & b_1 \\ c_2 & b_2 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}} \end{aligned}$$

Quindi nel caso che il determinante del sistema sia $\neq 0$, il sistema possiede un'unica soluzione data da

$$x = \frac{\begin{vmatrix} c_1 & b_1 \\ c_2 & b_2 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}} \qquad y = \frac{\begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}}$$

Il numeratore di x si ottiene quindi anch'esso dal determinante del sistema, sostituendo la prima colonna con il lato destro del sistema. Questo risultato è molto importante per l'algebra lineare e può essere generalizzato a più dimensioni; è noto come *regola di Cramer*.

Esempi

Risolviamo con la regola di Cramer il sistema

$$\begin{aligned} 3x - 2y &= 8 \\ x + 6y &= 5 \end{aligned}$$

Il determinante del sistema è $\begin{vmatrix} 3 & -2 \\ 1 & 6 \end{vmatrix} = 18 + 2 = 20$, quindi diverso da 0, per cui

$$x = \frac{\begin{vmatrix} 8 & -2 \\ 5 & 6 \end{vmatrix}}{20} = \frac{48 + 10}{20} = \frac{58}{20}$$

$$y = \frac{\begin{vmatrix} 3 & 8 \\ 1 & 5 \end{vmatrix}}{20} = \frac{15 - 8}{20} = \frac{7}{20}$$

Esercizio. Risolvere da soli

$$\begin{aligned} 4x + 3y &= 10 \\ 2x + 9y &= 7 \end{aligned}$$

Esercizio. Perché non si può applicare la regola di Cramer al sistema seguente?

$$\begin{aligned} x + 3y &= 1 \\ 2x + 6y &= 4 \end{aligned}$$

Eppure non è difficile trovare "tutte" le soluzioni. Perché ho messo "tutte" tra virgolette? E perché è anche (quasi) facile trovare tutte le soluzioni di

$$\begin{aligned} x + 3y &= 2 \\ 2x + 6y &= 4 \end{aligned}$$

La forma generale della regola di Cramer

Sia dato un sistema di n equazioni lineari in n incognite (quindi il numero delle equazioni è uguale al numero delle incognite):

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= c_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= c_2 \\ &\cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= c_n \end{aligned}$$

Anche in questo caso più generale si può definire il *determinante* del sistema, un numero che viene denotato con

$$D := \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ & \cdots & & \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix}$$

e si dimostrerà nel corso di Geometria I che questo determinante è $\neq 0$ se e solo se il sistema possiede un'unica soluzione che in tal caso è data da

$$x_1 = \frac{\begin{vmatrix} c_1 & a_{12} & \dots & a_{1n} \\ c_2 & a_{22} & \dots & a_{2n} \\ & \cdots & & \\ c_n & a_{n2} & \dots & a_{nn} \end{vmatrix}}{D}$$

$$x_2 = \frac{\begin{vmatrix} a_{11} & c_1 & \dots & a_{1n} \\ a_{21} & c_2 & \dots & a_{2n} \\ & \cdots & & \\ a_{n1} & c_n & \dots & a_{nn} \end{vmatrix}}{D}$$

$$\cdots$$

$$x_n = \frac{\begin{vmatrix} a_{11} & a_{12} & \dots & c_1 \\ a_{21} & a_{22} & \dots & c_2 \\ & \cdots & & \\ a_{n1} & a_{n2} & \dots & c_n \end{vmatrix}}{D}$$

x_i è quindi un quoziente il cui numeratore si ottiene dal determinante del sistema, sostituendo la i-esima colonna con il lato destro del sistema.

Determinanti

Conosciamo già i determinanti 2×2: $\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix} = a_1 b_2 - a_2 b_1$.

Definizione 5.1. Per induzione definiamo i determinanti di ordine superiore:

$$\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix} := a_1 \begin{vmatrix} b_2 & c_2 \\ b_3 & c_3 \end{vmatrix} - a_2 \begin{vmatrix} b_1 & c_1 \\ b_3 & c_3 \end{vmatrix} + a_3 \begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix}$$

$$\begin{vmatrix} a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \\ a_4 & b_4 & c_4 & d_4 \end{vmatrix} := a_1 \begin{vmatrix} b_2 & c_2 & d_2 \\ b_3 & c_3 & d_3 \\ b_4 & c_4 & d_4 \end{vmatrix} - a_2 \begin{vmatrix} b_1 & c_1 & d_1 \\ b_3 & c_3 & d_3 \\ b_4 & c_4 & d_4 \end{vmatrix} + a_3 \begin{vmatrix} b_1 & c_1 & d_1 \\ b_2 & c_2 & d_2 \\ b_4 & c_4 & d_4 \end{vmatrix} - a_4 \begin{vmatrix} b_1 & c_1 & d_1 \\ b_2 & c_2 & d_2 \\ b_3 & c_3 & d_3 \end{vmatrix}$$

e così via. Si noti l'alternanza dei segni. I determinanti hanno molte proprietà importanti che verranno studiate nel corso di Geometria. Per determinanti 2×2 e 3×3 dimostriamo alcune semplici regole, valide anche per determinanti $n \times n$, se riformulate in modo naturale.

Lemma 5.2. *Se in un determinante 2×2 scambiamo tra di loro due righe o due colonne, il determinante si moltiplica con -1.*

Dimostrazione. Immediata.

Lemma 5.3. *Un determinante 3×3 può essere calcolato anche secondo la regola*

$$\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix} := a_1 \begin{vmatrix} b_2 & c_2 \\ b_3 & c_3 \end{vmatrix} - b_1 \begin{vmatrix} a_2 & c_2 \\ a_3 & c_3 \end{vmatrix} + c_1 \begin{vmatrix} a_2 & b_2 \\ a_3 & b_3 \end{vmatrix}$$

Dimostrazione. Le due espansioni si distinguono in

$$-a_2 \begin{vmatrix} b_1 & c_1 \\ b_3 & c_3 \end{vmatrix} + a_3 \begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix} = -a_2 b_1 c_3 + a_2 b_3 c_1 + a_3 b_1 c_2 - a_3 b_2 c_1$$

e

$$-b_1 \begin{vmatrix} a_2 & c_2 \\ a_3 & c_3 \end{vmatrix} + c_1 \begin{vmatrix} a_2 & b_2 \\ a_3 & b_3 \end{vmatrix} = -b_1 a_2 c_3 + b_1 a_3 c_2 + c_1 a_2 b_3 - c_1 a_3 b_2$$

che però, come vediamo, danno lo stesso risultato.

Lemma 5.4. *Se si scambiano due righe o due colonne in una matrice 3×3, il determinante si moltiplica per -1.*

Dimostrazione. Ciò, per il lemma 5.2, è evidente per lo scambio della seconda e della terza colonna e, per il lemma 5.3, anche per lo scambio della seconda e della terza riga. Se invece scambiamo la prima e la seconda colonna, otteniamo il determinante

$$\begin{vmatrix} b_1 & a_1 & c_1 \\ b_2 & a_2 & c_2 \\ b_3 & a_3 & c_3 \end{vmatrix} := b_1 \begin{vmatrix} a_2 & c_2 \\ a_3 & c_3 \end{vmatrix} - a_1 \begin{vmatrix} b_2 & c_2 \\ b_3 & c_3 \end{vmatrix} + c_1 \begin{vmatrix} b_2 & a_2 \\ b_3 & a_3 \end{vmatrix}$$

uguale, come si vede subito, al determinante originale moltiplicato per -1. Gli altri casi seguono adesso applicando le regole già dimostrate.

Lemma 5.5. *Se in un determinante appaiono due righe o due colonne uguali, allora il determinante è uguale a 0.*

Dimostrazione. Ciò per un determinante 2×2 è ovvio, e se ad esempio sono uguali le ultime due colonne, l'enunciato segue (usando il caso 2×2) dalla formula di espansione anche per i determinanti 3×3, e poi dal caso 3×3 anche per i determinanti 4×4 ecc.

Esempio. Verificare con calcoli a mano che

$$\begin{vmatrix} a_1 & a_1 & c_1 \\ a_2 & a_2 & c_2 \\ a_3 & a_3 & c_3 \end{vmatrix} = 0$$

e che

$$\begin{vmatrix} a_1 & b_1 + a_1 & c_1 \\ a_2 & b_2 + a_2 & c_2 \\ a_3 & b_3 + a_3 & c_3 \end{vmatrix} = \begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix}$$

L'ultima uguaglianza è un caso speciale di un'altra proprietà fondamentale dei determinanti, contenuta nella proposizione 5.7.

Multilinearità del determinante

Siano dati n vettori

$$a_1 = (a_1^1, \dots, a_1^n), \dots, a_n = (a_n^1, \dots, a_n^n)$$

di $\mathbb{R}^n$. Allora la matrice

$$A := \begin{pmatrix} a_1^1 & \dots & a_n^1 \\ & \dots & \\ a_1^n & \dots & a_n^n \end{pmatrix}$$

è quadratica e possiamo formare il suo determinante

$$|A| := \begin{vmatrix} a_1^1 & \dots & a_n^1 \\ & \dots & \\ a_1^n & \dots & a_n^n \end{vmatrix}$$

che denotiamo anche con $\det(a_1, \dots, a_n)$.

Si noti che nella matrice abbiamo, come d'uso nel calcolo tensoriale (un calcolo multilineare sistematico molto importante nella geometria differenziale e nella fisica matematica), scritto gli indici di riga in alto, così come spesso anche al di fuori di una matrice gli indici dei componenti di un vettore v di $\mathbb{R}^n$ vengono posti in alto: $v = (v^1, \dots, v^n)$. È quasi sempre chiaro dal contesto se si tratta di indici o di esponenti di potenze.

Proposizione 5.6. *Il determinante è una funzione multilineare delle colonne (e delle righe) di una matrice.*

Ciò significa che, se anche $b = (b^1, \dots, b^n)$ è un vettore di $\mathbb{R}^n$, allora, per $\lambda, \mu \in \mathbb{R}$,

$$\det(\lambda a_1 + \mu b, a_2, \dots, a_n) = \\ = \lambda \det(a_1, a_2, \dots, a_n) + \mu \det(b, a_2, \dots, a_n) \\ \dots$$

$$\det(a_1, \dots, a_{n-1}, \lambda a_n + \mu b) = \\ = \lambda \det(a_1, \dots, a_{n-1}, a_n) + \mu \det(a_1, \dots, a_{n-1}, b)$$

Dimostrazione. Ciò, per $n = 2$, è evidente (verificare da soli) e, per l'ultima colonna, segue poi per induzione, come si vede dalle formule di espansione date. Dai lemmi di scambio (che valgono per ogni n, anche se li abbiamo dimostrato solo per $n = 2, 3$) ciò implica che il determinante è lineare in ogni colonna (e quindi anche in ogni riga, perché, come si vedrà nel corso di Geometria, il determinante della matrice trasposta di A, cioè della matrice che cui colonne sono le righe di A, è uguale al determinante di A).

Proposizione 5.7. *Se in un determinante a una colonna (o a una riga) aggiungiamo un multiplo di una delle altre colonne (o righe), il determinante non cambia:*

$$\det(\dots, a_i + \lambda a_k, \dots) = \det(\dots, a_i, \dots)$$

per $k \neq i$.

Dimostrazione. Assumiamo $i < k$. Allora

$$\det(\dots, a_i + \lambda a_k, \dots, a_k, \dots) = \\ = \det(\dots, a_i, \dots, a_k, \dots) + \lambda \det(\dots, a_k, \dots, a_k, \dots) \\ = \det(\dots, a_i, \dots, a_k, \dots)$$

usando la multilinearità e il lemma 5.5.

Esercizio. Verificare con calcolo a mano che

$$\begin{vmatrix} 1 & a & a^2 \\ 1 & b & b^2 \\ 1 & c & c^2 \end{vmatrix} = (b - a)(c - a)(c - b)$$

Questa regola può essere generalizzata a matrici $n \times n$; determinanti di questa forma si chiamano *determinanti di Vandermonde*. Qual'è la forma generale?

Esercizio. Verificare a mano che per $x, a, b, c \in \mathbb{R}$ vale sempre

$$\begin{vmatrix} x & -1 & 0 \\ 0 & x & -1 \\ c & b & x + a \end{vmatrix} = x^3 + ax^2 + bx + c$$

L'esercizio è molto semplice, ma anche questo è un determinante molto importante nella teoria; lo si incontra nello studio degli *autovalori* e *autovettori* di una matrice.

L'algoritmo di eliminazione di Gauss

La teoria dei determinanti e la regola di Cramer hanno una grandissima importanza teorica, ma non possono essere utilizzate se non per sistemi in due o al massimo tre incognite. Inoltre la regola di Cramer si applica solo al caso di un sistema quadratico. Esiste invece un metodo molto efficiente (anche nel calcolo a mano) per la risoluzione di sistemi di equazioni lineari, che viene detto *algoritmo di eliminazione di Gauss* e che consiste nella sistematica applicazione del teorema 3.6.

Esempio 6.1. Consideriamo il sistema

$$\begin{aligned} 3x + 2y - z &= 10 \ldots f_1 \\ 4x - 9y + 2z &= 6 \quad \ldots f_2 \\ x + y - 14z &= 2 \quad \ldots f_3 \end{aligned}$$

In analogia a quanto abbiamo fatto a pagina 4 per il sistema 2×2 le funzioni f_1, f_2 ed f_3 sono definite da

$$\begin{aligned} f_1(x, y, z) &:= 3x + 2y - z - 10 \\ f_2(x, y, z) &:= 4x - 9y + 2z - 6 \\ f_3(x, y, z) &:= x + y - 14z - 2 \end{aligned}$$

Le indichiamo alla destra delle equazioni corrispondenti. Con la notazione che abbiamo introdotto nell'osservazione 3.5 poniamo

$$f_4 := 4f_1 - 3f_2$$

Per il teorema 3.6 allora

$$(f_1 = 0, f_2 = 0, f_3 = 0) = (f_4 = 0, f_2 = 0, f_3 = 0)$$

perché il coefficiente con cui f_1 appare in f_4 è diverso da 0. Esplicitamente $f_4 = 0$ equivale a

$$4(3x + 2y - z) - 3(4x - 9y + 2z) = 4 \cdot 10 - 3 \cdot 6$$

cioè a

$$35y - 10z = 22.$$

Se chiamiamo due sistemi *equivalenti* quando hanno le stesse soluzioni, possiamo dire che il sistema originale è equivalente al sistema

$$\begin{aligned} 4x - 9y + 2z &= 6 \quad \ldots f_2 \\ x + y - 14z &= 2 \quad \ldots f_3 \\ 35y - 10z &= 22 \ldots f_4 \end{aligned}$$

Nell'ultima equazione la variabile x in f_4 è sparita, è stata *eliminata*. Ripetiamo questa operazione sostituendo la funzione f_2 con

$$f_5 := f_2 - 4f_3$$

Ciò è possibile perché in f_5 la f_2 appare con un coefficiente $\neq 0$. Esplicitamente $f_5 = 0$ significa

$$4x - 9y + 2z - 4(x + y - 14z) = 6 - 4 \cdot 2$$

cioè

$$-13y + 58z = -2.$$

Perciò il sistema originale ha le stesse soluzioni come il sistema

$$\begin{aligned} x + y - 14z &= 2 \quad \ldots f_3 \\ 35y - 10z &= 22 \quad \ldots f_4 \\ -13y + 58z &= -2 \ldots f_5 \end{aligned}$$

Adesso formiamo $f_6 := 13f_4 + 35f_5$ che può sostituire sia la f_4 che la f_5. Possiamo togliere la f_5. $f_6 = 0$ è equivalente a

$$13(35y - 10z) + 35(-13y + 58z) = 13 \cdot 22 + 35 \cdot (-2),$$

cioè a

$$1900z = 216.$$

Otteniamo così il sistema

$$\begin{aligned} x + y - 14z &= 2 \quad \ldots f_3 \\ 35y - 10z &= 22 \quad \ldots f_4 \\ 1900z &= 216 \ldots f_6 \end{aligned}$$

che è ancora equivalente a quello originale. Ma adesso vediamo che nell'ultima equazione è stata eliminata anche la y ed è rimasta solo la z che possiamo così calcolare direttamente:

$$z = \frac{216}{1900} = 0.11368,$$

poi, usando $f_4 = 0$, otteniamo

$$y = \frac{22 + 10z}{35} = 0.66105,$$

e infine dal $f_3 = 0$

$$x = -y + 14z + 2 = 2.930526.$$

Nella pratica si userà uno schema in cui vengono scritti, nell'ordine indicato dall'ordine delle variabili, solo i coefficienti. Nell'esempio appena trattato i conti verrebbero disposti nel modo seguente:

3	2	−1	10	f_1	✓
4	−9	2	6	f_2	✓
1	1	−14	2	f_3	
0	35	−10	22	$f_4 = 4f_1^* - 3f_2$	
0	−13	58	−2	$f_5 = f_2^* - 4f_3$	✓
0	0	1900	216	$f_6 = 13f_4 + 35f_5^*$	

L'asterisco indica ogni volta l'equazione cancellata in quel punto; l'uncino a destra di un'equazione significa che questa equazione è stata cancellata. Nei conti a mano spesso si preferirà forse cancellare la riga con un tratto orizzontale piuttosto di usare l'uncino.

Come si vede, nell'algoritmo cerchiamo prima di ottenere un sistema equivalente all'originale in cui tutti i coefficienti tranne al massimo uno nella prima colonna sono $= 0$, poi, usando le equazioni rimaste, applichiamo lo stesso procedimento alla seconda colonna (non modificando più però quella riga a cui corrisponde quell'eventuale coefficiente $\neq 0$ nella prima colonna), ecc. È chiaro che il procedimento termina sempre: alle m equazioni iniziali si aggiungono prima $m - 1$, poi $m - 2$, poi $m - 3$, ecc.

L'insieme delle soluzioni rimane sempre lo stesso; le equazioni cancellate naturalmente sono superflue e non vengono più usate. Quindi, se il sistema non ha soluzioni o più di una soluzione, riusciamo a scoprire anche questo.

Esempio 6.2. Consideriamo il sistema

$$\begin{aligned} 2x_1 - 5x_2 + 3x_3 - x_4 &= 1 \\ x_1 + 4x_2 - x_3 + 2x_4 &= 3 \\ 3x_1 - x_2 + 2x_3 + x_4 &= 7 \end{aligned}$$

Applichiamo il nostro schema:

2	−5	3	−1	1	f_1	✓
1	4	−1	2	3	f_2	
3	−1	2	1	7	f_3	✓
0	−13	5	−5	−5	$f_4 = f_1^* - 2f_2$	✓
0	−13	5	−5	−2	$f_5 = f_3^* - 3f_2$	
0	0	0	0	−3	$f_6 = f_4^* - f_5$	

Il sistema dato è quindi equivalente al sistema

$$\begin{aligned} x_1 + 4x_2 - x_3 + 2x_4 &= 3 \\ -13x_2 + 5x_3 - 5x_4 &= -2 \\ 0 &= -3 \end{aligned}$$

In particolare siamo arrivati alla contraddizione $0 = -3$, quindi il sistema non ha soluzione.

Sistemi con più di una soluzione

Consideriamo il sistema

$$\begin{aligned} 4x - y + 3z &= 5 \\ x + 2y - 10z &= 4 \\ 6x + 3y - 17z &= 13 \end{aligned}$$

Usiamo di nuovo il nostro schema di calcolo:

4	−1	3	5	f_1	✓
1	2	−10	4	f_2	
6	3	−17	13	f_3	✓
0	−9	43	−11	$f_4 = f_1^* - 4f_2$	✓
0	9	−43	11	$f_5 = 6f_2 - f_3^*$	
0	0	0	0	$f_6 = f_4^* + f_5$	

Stavolta non abbiamo una contraddizione, ma un'ultima equazione $0 = 0$ superflua, quindi siamo rimasti con due equazioni per tre incognite:

$$\begin{aligned} x + 2y - 10z &= 4 \\ 9y - 43z &= 11 \end{aligned}$$

Per ogni valore t di z possiamo risolvere

$$\begin{aligned} y &= \frac{11 + 43t}{9} \\ x &= -2y + 10t + 4 = \frac{-22 - 86t}{9} + 10t + 4 \\ &= \frac{-22 - 86t + 90t + 36}{9} = \frac{14}{9} + \frac{4}{9}t \end{aligned}$$

e vediamo che l'insieme delle soluzioni è una retta nello spazio $\mathbb{R}^3$ con la rappresentazione parametrica

$$\begin{aligned} x(t) &= \frac{14}{9} + \frac{4}{9}t \\ y(t) &= \frac{11}{9} + \frac{43}{9}t \\ z(t) &= t \end{aligned}$$

Per ogni numero reale t si ottiene un punto $(x(t), y(t), z(t)) \in \mathbb{R}^3$ che è una soluzione del nostro sistema, e viceversa ogni soluzione è di questa forma.

L'insieme delle soluzioni di un sistema lineare

Negli esempi visti finora abbiamo trovato sistemi che non avevano soluzioni, oppure un'unica soluzione (descriventi cioè un unico punto nello spazio), oppure, nell'ultimo esempio, una retta di soluzioni.

Ciò vale per ogni sistema di equazioni lineari: l'insieme delle soluzioni è sempre o vuoto (nessuna soluzione), oppure un solo punto, oppure una retta, oppure un piano, oppure uno spazio affine tridimensionale ecc., e viceversa ogni insieme di questa forma può essere descritto da un sistema di equazioni lineari. La dimostrazione di questo teorema e la definizione precisa del concetto di spazio affine verranno date nel corso di Geometria.

Nonostante l'efficienza dell'algoritmo di eliminazione che permette la risoluzione abbastanza agevole di sistemi lineari non troppo grandi (con un po' di pazienza si possono risolvere anche sistemi 10×10 a mano) la pratica è più complicata. Nelle applicazioni reali si affrontano sistemi con decine di migliaia di equazioni e variabili e non solo il tempo di calcolo, ma anche l'accumularsi di errori di arrotondamento nei calcoli approssimati che il software normalmente utilizza possono creare grandi problemi.

Piccoli errori, spesso inevitabili, nei dati in entrata (ad esempio nei coefficienti a_{ij} e b_i del nostro sistema) possono provocare in taluni casi, che bisogna riconoscere e controllare, grandi cambiamenti nelle soluzioni. Così il sistema

$$\begin{aligned} 4x + 1.99y &= 2.01 \\ 2x + y &= 1 \end{aligned}$$

possiede un'unica soluzione $x = 1, y = -1$, ma se lo cambiamo di poco,

$$\begin{aligned} 4x + 2y &= 2 \\ 2x + y &= 1 \end{aligned}$$

il determinante si annulla e l'insieme delle soluzioni è dato da $y = 1 - 2x$, e quindi le soluzioni non sono più univocamente determinate e possono essere arbitrariamente distanti dalla soluzione $(1, -1)$ del primo sistema.

Esercizi per gli scritti

1. Sia $f := \bigcirc_x 3x^4 - 5x^2 + 10x - 1 : \mathbb{R} \longrightarrow \mathbb{R}$. Calcolare $f(2)$.

2. Siano $f := \bigcirc_x 3x^2 + 5$ e $g := \bigcirc_x 7x - 1$ considerate come funzioni $\mathbb{R} \longrightarrow \mathbb{R}$. Calcolare $f \circ g$ e $g \circ f$.

3. Siano $f := \bigcirc_{(x,y)} x + y^3 : \mathbb{R}^2 \longrightarrow \mathbb{R}$ e $g := \bigcirc_x x^2 : \mathbb{R} \longrightarrow \mathbb{R}$. Calcolare $g \circ f$.

4. Risolvere con la regola di Cramer il sistema

$$\begin{aligned} 3x - 2y &= 26 \\ 2x + 7y &= 9 \end{aligned}$$

5. Calcolare il determinante

$$\begin{vmatrix} 3 & 8 & 2 \\ 6 & 2 & 1 \\ 4 & 2 & 5 \end{vmatrix}$$

6. Calcolare il determinante

$$\begin{vmatrix} 2 & 8 & 12 \\ 5 & 1 & 11 \\ 2 & -5 & -1 \end{vmatrix}$$

Risolvere i sistemi con l'algoritmo di Gauß usando lo schema.

7.
$$\begin{aligned} 2x_1 - 4x_2 + x_3 - x_4 &= -1 \\ x_1 + 5x_2 - x_3 + 2x_4 &= 0 \\ 2x_1 - x_2 + 4x_3 + x_4 &= 24 \\ 4x_1 + x_2 - x_3 + 3x_4 &= 5 \end{aligned}$$

8.
$$\begin{aligned} x_1 + 2x_2 + 3x_3 + 2x_4 &= 20 \\ 4x_1 + x_2 + 2x_3 + x_4 &= 16 \\ 4x_1 + 2x_2 + 3x_3 + 4x_4 &= 27 \\ x_1 - 2x_2 - 2x_3 - 2x_4 &= -13 \end{aligned}$$

9.
$$\begin{aligned} 8x + 7y + 15z &= 10 \\ 2x + 3y + 3z &= 4 \\ 3x + 2y + 6z &= 3 \end{aligned}$$

10.
$$\begin{aligned} 2x_1 + x_2 + x_3 + x_4 + 2x_5 &= 29 \\ 2x_1 + x_2 + x_3 + x_4 + x_5 &= 24 \\ x_1 - x_2 + 2x_3 + x_4 + 2x_5 &= 23 \\ x_1 + 2x_2 + 2x_3 + x_4 + 3x_5 &= 34 \\ 3x_1 + x_2 + x_4 + 2x_5 &= 32 \end{aligned}$$

11.
$$\begin{aligned} 4x_1 + 8x_2 + 7x_3 + x_4 + 2x_5 &= 20 \\ 3x_1 + 5x_2 + 2x_3 + 3x_4 + 4x_5 &= 31 \\ 2x_2 + x_3 + x_4 + 2x_5 &= 14 \\ 2x_1 + 3x_2 + x_3 + 6x_4 + 5x_5 &= 36 \\ 4x_1 + 2x_2 + x_3 + 7x_4 + 2x_5 &= 20 \end{aligned}$$

12.
$$\begin{aligned} x_1 + 5x_2 + x_3 + 4x_4 &= -8 \\ 2x_1 + x_2 + 3x_3 + 2x_4 &= -9 \\ x_1 + 2x_2 + x_3 + 7x_4 &= 1 \\ 2x_1 + x_2 + 3x_3 + 4x_4 &= -7 \end{aligned}$$

13.
$$\begin{aligned} x_1 + 2x_2 + x_3 + x_4 + 2x_5 + x_6 &= 28 \\ 2x_1 + x_2 + x_3 + x_4 + x_5 &= 16 \\ x_1 + 2x_2 + x_3 + x_5 + 2x_6 &= 25 \\ 2x_1 + x_3 + x_4 + x_5 + 4x_6 &= 38 \\ 4x_1 + x_2 + 2x_5 + x_6 &= 22 \\ 2x_1 - x_3 + x_4 + x_5 + 4x_6 &= 32 \end{aligned}$$

III. TRIGONOMETRIA PIANA

Trigonometria oggi

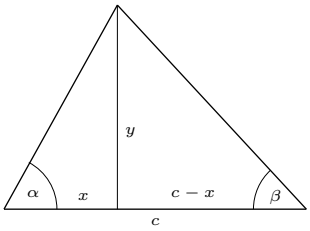
Dai piani di studio, soprattutto nell'università, la trigonometria è sparita da molto tempo. Ma questa disciplina, una delle più antiche della matematica, è ancora oggi una delle più importanti.

Mentre almeno gli elementi della trigonometria piana vengono insegnati nelle scuole, la trigonometria sferica è ormai conosciuta pochissimo anche tra i matematici di professione. Eppure le applicazioni sono tantissime: nautica, cartografia, geodesia e geoinformatica, astronomia, cristallografia, classificazione dei movimenti nello spazio, cinematica e quindi robotica e costruzione di macchine, grafica al calcolatore.

Problemi di geodesia piana

Sia dato, come nella figura, un triangolo con base di lunghezza nota c e in cui anche gli angoli α e β siano noti e tali che $0 < \alpha, \beta < 90°$.

Vogliamo calcolare x ed y.



Per le nostre ipotesi $\tan\alpha$ e $\tan\beta$ sono numeri ben definiti e > 0 (cfr. pag. 10). Inoltre abbiamo

$$\tan\alpha = \frac{y}{x}$$
$$\tan\beta = \frac{y}{c-x}$$

Queste equazioni possono essere riscritte come sistema lineare di due equazioni in due incognite:

$$x\tan\alpha - y = 0$$
$$x\tan\beta + y = c\tan\beta$$

Il determinante $\begin{vmatrix} \tan\alpha & -1 \\ \tan\beta & 1 \end{vmatrix}$ di questo sistema è uguale a

$$\tan\alpha + \tan\beta$$

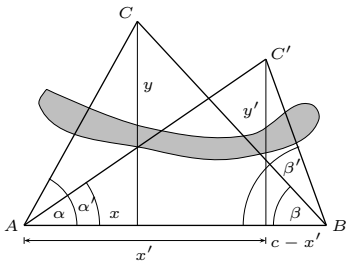
e quindi > 0. Possiamo perciò applicare la regola di Cramer e otteniamo

$$x = \frac{\begin{vmatrix} 0 & -1 \\ c\tan\beta & 1 \end{vmatrix}}{\tan\alpha + \tan\beta} = \frac{c\tan\beta}{\tan\alpha + \tan\beta}$$

mentre per y possiamo, se calcoliamo prima x, usare direttamente la relazione $y = x\tan\alpha$.

Esercizio. Prendendo il centimetro come unità di misura e con l'uso di un goniometro verificare le formule con le distanze nella figura.

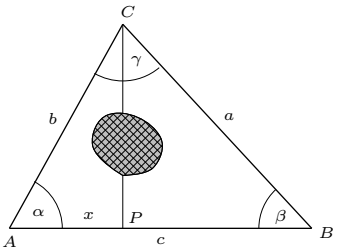
Con questo metodo possiamo risolvere due compiti elementari ma frequenti di geodesia piana.



Assumiamo di conoscere la distanza tra i punti A e B e, mediante un teodolite, di essere in grado di misurare gli angoli α, β, α' e β'. Vorremmo conoscere la distanza tra i punti C e C', ai quali però non possiamo accedere direttamente, ad esempio perché da essi ci separa un fiume che non riusciamo ad attraversare o perché si trovano in mezzo a una palude. Se le distanze sono molto grandi (maggiori di 50 km), dovremo appellarci alla trigonometria sferica, per distanze sufficientemente piccole invece possiamo utilizzare la tecnica vista sopra che ci permette di calcolare x, y, x' e y', da cui la distanza tra C e C' si ottiene come

$$|C - C'| = \sqrt{(x-x')^2 + (y-y')^2}$$

Consideriamo di nuovo un triangolo



Stavolta assumiamo di trovarci nel punto C e di voler determinare la proiezione P di C sulla retta tra i punti A e B che ci sono visibili, mentre un ostacolo visivo ci impedisce di determinare P in modo più diretto. Mirando ad A e B da C misuriamo le distanze a e b e l'angolo γ, da cui con il teorema del coseno calcoliamo prima c e poi α e β. Da $x = \dfrac{c\tan\beta}{\tan\alpha + \tan\beta}$ troviamo facilmente P.

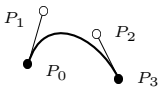
Grafica al calcolatore e geometria

La geometria viene utilizzata in molti campi della tecnologia moderna: nella tomografia computerizzata, nella pianificazione di edifici, nella creazione di animazioni per film e pubblicità, nell'analisi dei movimenti di robot e satelliti.

La grafica al calcolatore e le discipline affini come la geometria computazionale e l'elaborazione delle immagini si basano sulla matematica. È importante separare gli algoritmi dalla loro realizzazione mediante un linguaggio di programmazione. È importante separare la rappresentazione matematica delle figure nello spazio dalle immagini che creiamo sullo schermo di un calcolatore.

Il matematico è molto avvantaggiato in questo. Già semplici nozioni di trigonometria e di geometria (traslazioni, rotazioni, riflessioni, coordinate baricentriche, i vari tipi di proiezioni) e algebra lineare possono rendere facili o immediate costruzioni e formule di trasformazione (e quindi gli algoritmi che da esse derivano) che senza questi strumenti matematici risulterebbero difficoltose o non verrebbero nemmeno scoperte.

La geometria proiettiva, apparentemente una vecchia teoria astratta e filosofica, diventa di sorpresa una tecnica molto utile per trasformare compiti di proiezione in semplici calcoli con coordinate omogenee.

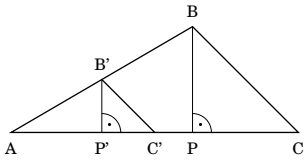


I concetti dell'analisi e della geometria differenziale portano all'introduzione e allo studio delle curve e superficie di Bézier, largamente utilizzate nei programmi di disegno al calcolatore (CAD, computer aided design).

Molte figure possono essere descritte mediante equazioni algebriche; per questa ragione la geometria algebrica assume notevole importanza nella grafica al calcolatore moderna. Curve e superficie possono essere date in forme parametrica oppure mediante un sistema di equazioni; le basi di Gröbner forniscono uno strumento per passare da una rappresentazione all'altra.

La topologia generale, una disciplina tra la geometria, l'analisi e l'algebra, è la base della morfologia matematica, mentre la topologia algebrica e la geometria algebrica reale possiedono applicazioni naturali in robotica.

Il triangolo



In questa figura i segmenti BC e $B'C'$ sono paralleli. Nella geometria elementare si dimostra che le *proporzioni* del triangolo più piccolo $AB'C'$ sono uguali alle proporzioni del triangolo grande ABC. Cio significa che, se $\overline{AB}$ denota la lunghezza del segmento AB, allora

$$\frac{\overline{AB'}}{\overline{AB}} = \frac{\overline{AC'}}{\overline{AC}} = \frac{\overline{B'C'}}{\overline{BC}}$$

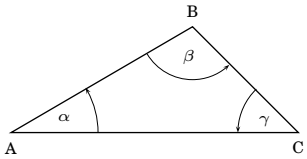
Se il valore comune di queste tre frazioni viene denotato con λ, abbiamo quindi

$$\begin{aligned}\overline{AB'} &= \lambda \cdot \overline{AB}\\ \overline{AC'} &= \lambda \cdot \overline{AC}\\ \overline{B'C'} &= \lambda \cdot \overline{BC}\end{aligned}$$

Una relazione analoga vale anche per le altezze:

$$\overline{B'P'} = \lambda \cdot \overline{BP}$$

Dati tre punti A, B, C denotiamo con $\sphericalangle(AC, AB)$ l'angolo α tra i segmenti AC e AB:



Evidentemente $0 < \alpha < 180^\circ$.

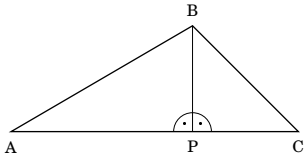
Con β e γ indichiamo gli altri due angoli come nella figura; spesso serve solo la grandezza assoluta degli angoli, allora si lasciano via le punte di freccia.

Nella prima figura il triangolo piccolo e il triangolo grande hanno gli stessi angoli, cioè

$$\begin{aligned}\sphericalangle(AC, AB) &= \sphericalangle(AC', AB')\\ \sphericalangle(BA, BC) &= \sphericalangle(B'A, B'C')\\ \sphericalangle(CB, CA) &= \sphericalangle(C'B', C'A)\end{aligned}$$

Si può dimostrare ed è chiaro intuitivamente che, dati due triangoli con gli stessi angoli, essi possono essere sovrapposti in maniera tale che si ottenga una figura simile alla nostra.

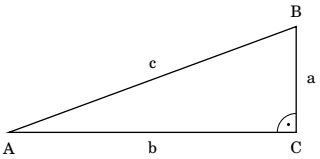
Ogni triangolo può essere considerato (talvolta anche in più modi - quando?) come unione di due triangoli rettangoli.



Le formule per i triangoli rettangoli sono particolarmente semplici; conviene quindi studiare separatamente i triangoli APB e PBC.

Il triangolo rettangolo

Il triangolo ABC sia rettangolo, ad esempio $\sphericalangle(CA, CB) = 90^\circ$.



Il lato più lungo è quello opposto all'angolo retto, cioè AB, e si chiama *ipotenusa*, i due altri lati sono più brevi e sono detti *cateti*.

La somma dei tre angoli α, β, γ di un triangolo è sempre uguale a 180°:

$$\alpha + \beta + \gamma = 180^\circ$$

Ciò implica che un triangolo può avere al massimo un angolo retto (se ce ne fossero due, il terzo dovrebbe essere zero e non avremmo più un triangolo).

Teorema 9.1 (teorema di Pitagora). *Dato un triangolo rettangolo e posto $a := \overline{BC}$, $b := \overline{AC}$ e $c := \overline{AB}$ come nella figura, si ha*

$$a^2 + b^2 = c^2$$

Il teorema di Pitagora (dimostrato a pagina 10) implica che l'ipotenusa è veramente più lunga di ciascuno dei due cateti (perché $a, b > 0$). La relazione $c^2 = a^2 + b^2$ può essere anche usata per il calcolo di uno dei lati di un triangolo rettangolo dagli altri due:

$$\begin{aligned}c &= \sqrt{a^2 + b^2}\\ a &= \sqrt{c^2 - b^2}\\ b &= \sqrt{c^2 - a^2}\end{aligned}$$

Triple pitagoree

Una tripla pitagorea è una tripla (a, b, c) di numeri naturali positivi tali che $a^2 + b^2 = c^2$. La tripla pitagorea si chiama *primitiva*, se a, b e c sono relativamente primi tra di loro. Diamo una tavola delle prime triple pitagoree primitive in ordine crescente di c.

3	4	5
5	12	13
8	15	17
7	24	25
20	21	29
12	35	37
9	40	41
28	45	53
11	60	61
33	56	65
16	63	65

Gli arabi possedevano già nel 972 tavole simili.

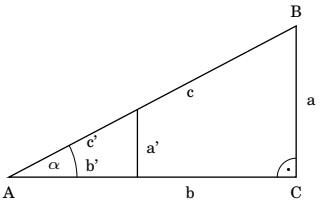
Come abbiamo già osservato, per $n > 2$ non esistono invece soluzioni dell'equazione

$$a^n + b^n = c^n$$

con a, b, c interi > 0. La dimostrazione di questo teorema (detto *ultimo teorema di Fermat*) è stata molto difficile; Andrew Wiles ha utilizzato strumenti molto avanzati della geometria algebrica. Pierre de Fermat (circa 1607-1665) sostenne di conoscere una dimostrazione, ma non è mai stata trovata e si dubita molto che sia esistita.

Le funzioni trigonometriche

Consideriamo la seguente figura,



in cui a, b, c sono come prima i lati del triangolo rettangolo più grande e a', b' e c' sono i lati del triangolo più piccolo, che è ancora rettangolo. Le proporzioni nella figura dipendono solo dall'angolo α, si ha cioè

$$\frac{c'}{c} = \frac{b'}{b} = \frac{a'}{a}$$

e da ciò anche

$$\begin{aligned}\frac{a'}{c'} &= \frac{a}{c}\\ \frac{b'}{a'} &= \frac{b}{a}\\ \frac{c'}{b'} &= \frac{c}{b}\end{aligned}$$

Questi rapporti sono perciò funzioni dell'angolo α che vengono dette funzioni trigonometriche e denotate come segue:

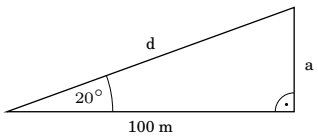
$$\begin{aligned}\sin\alpha &:= \frac{a}{c} \ldots \text{seno di } \alpha\\ \cos\alpha &:= \frac{b}{c} \ldots \text{coseno di } \alpha\\ \tan\alpha &:= \frac{a}{b} \ldots \text{tangente di } \alpha\\ \cot\alpha &:= \frac{b}{a} \ldots \text{cotangente di } \alpha\end{aligned}$$

Dalle definizioni seguono le relazioni

$$\begin{aligned}a &= c\sin\alpha = b\tan\alpha\\ b &= c\cos\alpha = a\cot\alpha\\ c &= \frac{a}{\sin\alpha} = \frac{b}{\cos\alpha}\end{aligned}$$

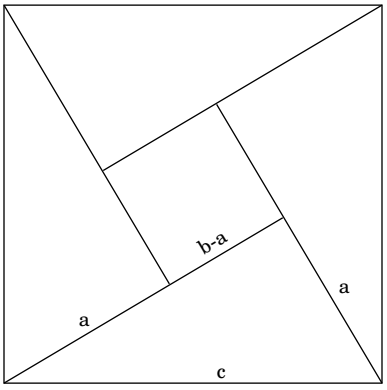
Esercizio 9.2. Calcolare $\sin 45^\circ$, $\cos 45^\circ$, $\tan 45^\circ$, $\cot 45^\circ$.

Esercizio 9.3. I valori delle funzioni trigonometriche si trovano in tabelle oppure possono essere calcolati con la calcolatrice tascabile oppure con una semplice istruzione in quasi tutti i linguaggi di programmazione. Ricavare in uno di questi modi i necessari valori per calcolare la distanza d e l'altezza a nella seguente figura:



La dimostrazione indiana

In una fonte indiana del dodicesimo secolo si trova il seguente disegno, con una sola parola in sanscrito: *guarda!*



Da esso si deduce immediatamente il teorema di Pitagora:

Il nostro triangolo rettangolo abbia i lati a, b, c con $a < b < c$. Allora l'area del quadrato grande è uguale a quella del quadrato piccolo più quattro volte l'area del triangolo, quindi

$$c^2 = (b-a)^2 + 4\frac{ab}{2}$$

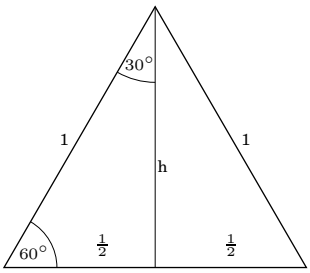
cioè

$$c^2 = b^2 - 2ab + a^2 + 2ab = b^2 + a^2$$

Esercizio: Disegnare la figura nel caso che $a = b$ e convincersi che la dimostrazione rimane ancora valida.

Il triangolo isolatero

Consideriamo adesso un triangolo isolatero di lato 1. In esso anche gli angoli devono essere tutti uguali, quindi, dovendo essere la somma degli angoli $180°$, ogni angolo è uguale a $60°$.



Dalla figura otteniamo

$$h = \sqrt{1-\tfrac{1}{4}} = \frac{\sqrt{3}}{2}$$

$$\sin 60° = \cos 30° = \frac{\sqrt{3}}{2}$$

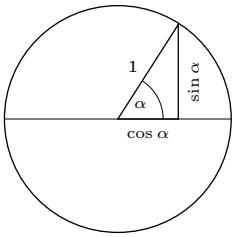
$$\sin 30° = \cos 60° = \frac{1}{2}$$

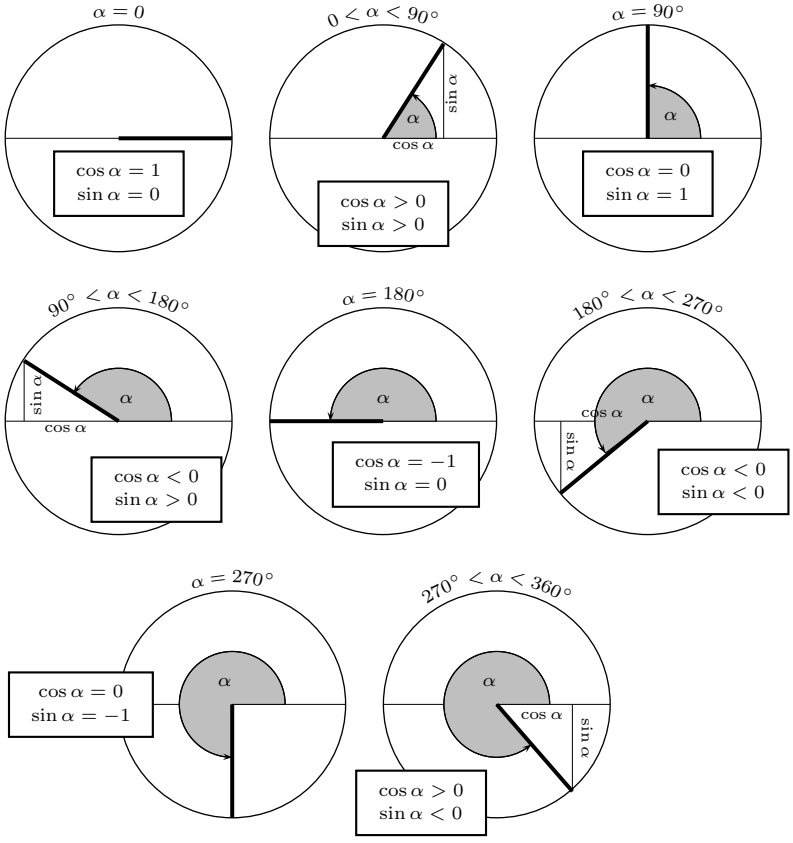
$$\tan 60° = \frac{2h}{} = \sqrt{3}$$

$$\tan 30° = \frac{1}{2h} = \frac{\sqrt{3}}{3}$$

Angoli sul cerchio

Siccome le lunghezze assolute non sono importanti, possiamo assumere che l'ipotenusa del triangolo rettangolo considerato sia di lunghezza 1 e studiare le funzioni trigonometriche sulla circonferenza di raggio 1. Questo ci permette inoltre di estendere la definizione delle funzioni trigonometriche a valori arbitrari di α, non necessariamente sottoposti, come finora, alla condizione $0 < \alpha < 90°$. Definiamo prima $\sin\alpha$ e $\cos\alpha$ per ogni α con $0 \le \alpha \le 360°$ come nelle seguenti figure:





Definiamo poi ogni volta

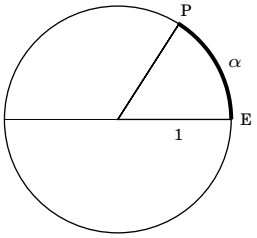
$$\tan\alpha := \frac{\sin\alpha}{\cos\alpha} \qquad \cot\alpha := \frac{\cos\alpha}{\sin\alpha}$$

quando $\cos\alpha \neq 0$ risp. $\sin\alpha \neq 0$. Si vede subito che questa definizione coincide con quella data a pag. 9, quando $0 < \alpha < 90°$.

Quindi $\tan\alpha = \dfrac{1}{\cot\alpha}$ quando entrambi i valori sono definiti.

Se α è infine un numero reale qualsiasi (non necessariamente compreso tra 0 e $360°$), esiste sempre un numero intero n tale che $\alpha = n \cdot 360° + \alpha_0$ con $0 \le \alpha_0 < 360°$ e possiamo definire $\cos\alpha := \cos\alpha_0$, $\sin\alpha := \sin\alpha_0$, $\tan\alpha := \tan\alpha_0$, $\cot\alpha := \cot\alpha_0$.

In matematica si identifica l'angolo con la lunghezza dell'arco descritto sulla circonferenza tra i punti E e P della figura a lato, aggiungendo però multipli del perimetro della circonferenza se l'angolo è immaginato ottenuto dopo essere girato più volte attorno al centro. Se il centro del cerchio è l'origine $(0,0)$ del piano, possiamo assumere che $E = (1,0)$. Siccome il perimetro della circonferenza di raggio 1 è 2π, si ha $360° = 2\pi$.



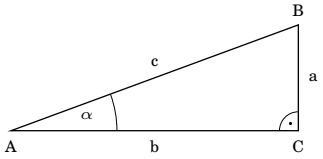
È chiaro che un angolo di $g°$ è uguale a $\dfrac{g}{360}2\pi$,

in altre parole $g° = \dfrac{2\pi g}{360}$, e viceversa $\alpha = \alpha\dfrac{360°}{2\pi}$ per ogni $\alpha \in \mathbb{R}$.

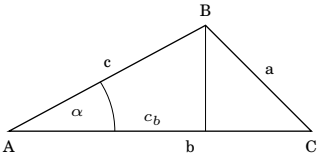
Infatti $1 = \dfrac{360°}{2\pi} \sim 57.29577951°$.

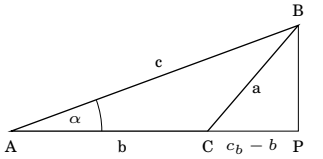
Il teorema del coseno

Dato un triangolo con i vertici A, B, C poniamo ancora $a := \overline{BC}$, $b := \overline{AC}$ e $c := \overline{AB}$. Denotiamo inoltre con c_b la lunghezza della proiezione di AB su AC misurando a partire da A. In modo analogo sono definite le grandezze c_a, b_a ecc. Se l'angolo α è ottuso, c_b sarà negativo. Sono possibili quattro situazioni:

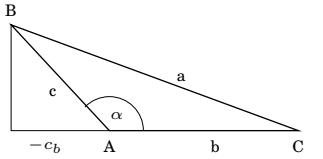


In questo caso $c_b = b$.





Si osservi che qui c_b è la lunghezza di tutto il segmento AP.



Teorema 11.1. *In tutti i casi, quindi in ogni triangolo, vale la relazione*

$$a^2 = b^2 + c^2 - 2bc_b$$

Per simmetria vale anche

$$c^2 = a^2 + b^2 - 2ab_a$$

Dimostrazione. Quando $c_b = b$, la formula diventa $a^2 = c^2 - b^2$ e segue direttamente dal teorema di Pitagora.

Nei rimanenti tre casi calcoliamo l'altezza del triangolo con il teorema di Pitagora in due modi.
Nella seconda figura abbiamo

$$c^2 - c_b^2 = a^2 - (b - c_b)^2$$

cioè

$$c^2 - c_b^2 = a^2 - b^2 + 2bc_b - c_b^2$$

per cui

$$c^2 = a^2 - b^2 + 2bc_b$$

Similmente nella terza figura

$$c^2 - c_b^2 = a^2 - (c_b - b)^2$$

la stessa equazione di prima.

Nella quarta figura infine abbiamo

$$c^2 - (-c_b)^2 = a^2 - (b - c_b)^2$$

che è ancora la stessa equazione.

Teorema 11.2 (teorema di Pitagora inverso). *Un triangolo è rettangolo con l'ipotenusa c se e solo se*

$$c^2 = a^2 + b^2$$

Dimostrazione. Dalla figura in alto a destra a pag. 9 si vede che il triangolo è rettangolo con ipotenusa c se e solo se $b_a = 0$ (oppure, equivalentemente, $a_b = 0$). L'enunciato segue dal teorema precedente.

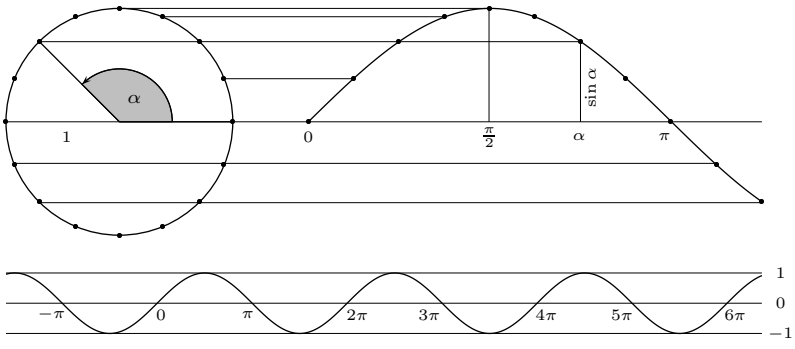
Teorema 11.3 (teorema del coseno).

$$a^2 = b^2 + c^2 - 2bc\cos\alpha$$

Dimostrazione. $c_b = c\cos\alpha$ in tutti e quattro i casi del precedente teorema (cfr. le definizioni degli angoli sul cerchio a pagina 10).

Il grafico della funzione seno

Facendo percorrere α l'asse reale e riportando $\sin\alpha$ come ordinata, otteniamo il grafico della funzione seno.



Come si vede dalla figura e come sarà dimostrato rigorosamente nel corso di Analisi, la funzione seno è iniettiva sull'intervallo chiuso $[-\frac{\pi}{2}, \frac{\pi}{2}]$ e assume su questo intervallo tutti i valori possibili per il seno, cioè tutti i valori tra -1 e 1. Possiamo quindi definire una funzione biiettiva $\bigcirc_{\alpha} \sin\alpha : [-\frac{\pi}{2}, \frac{\pi}{2}] \longrightarrow [-1, 1]$. L'inversa di questa funzione viene denotata con $\arcsin$. In modo analogo si definiscono l'inversa $\arccos$ della funzione biiettiva $\bigcirc_{\alpha} \cos\alpha : [0, \pi] \longrightarrow [-1, 1]$ e l'inversa $\arctan$ della funzione biiettiva $\bigcirc_{\alpha} \tan\alpha : (-\frac{\pi}{2}, \frac{\pi}{2}) \longrightarrow (-\infty, \infty)$.

La periodicità di seno e coseno

$$\cos(\alpha + 360^\circ) = \cos\alpha$$
$$\sin(\alpha + 360^\circ) = \sin\alpha$$

per ogni numero reale α, come segue dalle definizioni date a pagina 10. Invece di 360° possiamo anche scrivere 2π, quindi

$$\cos(\alpha + 2\pi) = \cos\alpha$$
$$\sin(\alpha + 2\pi) = \sin\alpha$$

per ogni numero reale α. Le funzioni $\sin$ e $\cos$ sono quindi funzioni periodiche con periodo 2π.

Altre proprietà di seno e coseno

$$\cos(-\alpha) = \cos\alpha$$
$$\sin(-\alpha) = -\sin\alpha$$

per ogni numero reale α, come si vede dai disegni a pagina 10. Il coseno è quindi una funzione *pari*, il seno una funzione *dispari*.

Teorema 11.4 (teorema di addizione).

$$\sin(\alpha + \beta) = \sin\alpha \cdot \cos\beta + \sin\beta \cdot \cos\alpha$$
$$\cos(\alpha + \beta) = \cos\alpha \cdot \cos\beta - \sin\alpha \cdot \sin\beta$$

Dimostrazione. Corsi di Analisi.

Esercizio 11.5. Dimostrare le formule

$$\sin(\alpha + \pi/2) = \cos\alpha$$
$$\cos(\alpha + \pi/2) = -\sin\alpha$$
$$\sin(\pi/2 - \alpha) = \cos\alpha$$
$$\cos(\pi/2 - \alpha) = \sin\alpha$$
$$\sin(\pi - \alpha) = \sin\alpha$$
$$\cos(\pi - \alpha) = -\cos\alpha$$

Teorema 11.6. $\sin^2\alpha + \cos^2\alpha = 1$.

Dimostrazione. Ciò segue direttamente dalle definizioni geometriche.

Mentre queste proprietà algebriche delle funzioni trigonometriche rimangono valide anche per un argomento α complesso, ciò non è più vero per le disuguaglianze $|\sin\alpha| \leq 1$ e $|\cos\alpha| \leq 1$. Infatti, se dall'analisi complessa anticipiamo le formule

$$\cos z = \frac{e^{iz} + e^{-iz}}{2} \qquad \sin z = \frac{e^{iz} - e^{-iz}}{2i}$$

valide per ogni numero complesso z, vediamo che ad esempio $\cos ai = \dfrac{e^{-a} + e^{a}}{2}$, quindi per a reale e tendente ad infinito (in questo caso e^{-a} tende a 0) $\cos ai$ si comporta come $\dfrac{e^a}{2}$ e tende quindi fortemente ad infinito.

arcsin, arccos e arctan

Queste funzioni, definite a sinistra, sono determinate dalle seguenti relazioni:

$$\arcsin x = \alpha \iff \sin\alpha = x$$

per $-1 \leq x \leq 1$ e $-\frac{\pi}{2} \leq \alpha \leq \frac{\pi}{2}$

$$\arccos x = \alpha \iff \cos\alpha = x$$

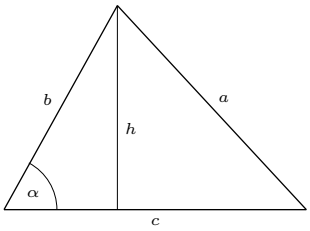
per $-1 \leq x \leq 1$ e $0 \leq \alpha \leq \pi$

$$\arctan x = \alpha \iff \tan\alpha = x$$

per $-\infty < x < \infty$ e $-\frac{\pi}{2} \leq \alpha \leq \frac{\pi}{2}$

L'area di un triangolo

Consideriamo un triangolo come nella figura:



Come mostrano le illustrazioni a pagina 10, l'altezza h è sempre uguale ad $h = b|\sin\alpha|$, anche quando α non è compreso tra 0 e $90°$. D'altra parte l'area del triangolo è uguale a $ch/2$, cosicché otteniamo la formula fondamentale per l'area di un triangolo:

$$\text{area} = \frac{1}{2}bc|\sin\alpha|$$

Proposizione 12.1. *Con le stesse notazioni l'area di un triangolo può essere calcolata anche con le formule*

$$\begin{aligned}\text{area} &= \frac{1}{4}\sqrt{2(a^2b^2+b^2c^2+c^2a^2)-(a^4+b^4+c^4)}\\ &= \sqrt{s(s-a)(s-b)(s-c)}\end{aligned}$$

dove abbiamo posto $s := \dfrac{a+b+c}{2}$.

La seconda formula è spesso citata come formula di Erone.

Dimostrazione. Dalla formula fondamentale abbiamo prima

$$4\,\text{area}^2 = b^2c^2\sin^2\alpha = b^2c^2(1-\cos^2\alpha)$$

(1) Per il teorema del coseno

$$a^2 = b^2 + c^2 - 2bc\cos\alpha$$

cosicché

$$\cos\alpha = \frac{b^2+c^2-a^2}{2bc} \quad \text{e} \quad \cos^2\alpha = \frac{(b^2+c^2-a^2)^2}{4b^2c^2}$$

quindi

$$4\,\text{area}^2 = b^2c^2 - \frac{1}{4}(b^2+c^2-a^2)^2$$

da cui

$$\begin{aligned}16\,\text{area}^2 &= 4b^2c^2 - (b^2+c^2-a^2)^2\\ &= 4b^2c^2 - b^4 - c^4 - a^4 - 2b^2c^2 + 2a^2b^2 + 2c^2a^2\\ &= 2a^2b^2 + 2b^2c^2 + 2c^2a^2 - (a^4+b^4+c^4)\end{aligned}$$

Abbiamo così ottenuto la prima formula.

(2) Possiamo però anche scrivere, usando più volte la formula $x^2 - y^2 = (x+y)(x-y)$,

$$\begin{aligned}4b^2c^2 - (b^2+c^2-a^2)^2 &= (2bc + b^2 + c^2 - a^2)(2bc - b^2 - c^2 + a^2)\\ &= ((b+c)^2 - a^2)(a^2 - (b-c)^2)\\ &= (b+c+a)(b+c-a)(a+b-c)(a-b+c)\end{aligned}$$

cosicché

$$16\,\text{area}^2 = (b+c+a)(b+c-a)(a+b-c)(a+c-b)$$

Posto, come nell'enunciato, $a+b+c = 2s$, abbiamo però

$$b+c-a = 2s-2a = 2(s-a)$$

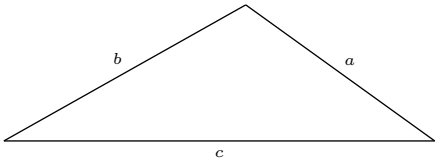
e nello stesso modo

$$a+b-c = 2(s-c) \quad \text{e} \quad a+c-b = 2(s-b)$$

cosicché $16\,\text{area}^2 = 2s\cdot 2(s-a)\cdot 2(s-b)\cdot 2(s-c)$ e quindi

$$\text{area}^2 = s(s-a)(s-b)(s-c)$$

Esempio 12.2. Misurando con la riga (in mm) i lati del triangolo



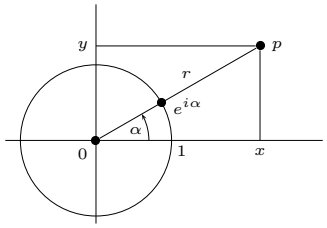
abbiamo $a = 30, b = 37, c = 57, s = 62$, per cui

$$\text{area} = \sqrt{62\cdot 32\cdot 25\cdot 5} = \sqrt{248000} \simeq 498$$

l'area è quindi con buona approssimazione uguale a 4.98 cm^2.

Coordinate polari nel piano

Sia $p = (x, y)$ un punto del piano reale.



Si vede che, se $p \neq (0,0)$, allora

$$\boxed{\begin{aligned}x &= r\cos\alpha\\ y &= r\sin\alpha\end{aligned}} \qquad (*)$$

dove $r = \sqrt{x^2+y^2}$, mentre l'angolo α è univocamente determinato se chiediamo $0 \le \alpha < 2\pi$.

Nel caso $p = (0,0)$ la rappresentazione (*) rimane valida con $r = 0$ e qualsiasi α, la biiettività della (*) viene quindi meno nel punto $p = (0,0)$.

Scriviamo adesso $e^{i\alpha} := (\cos\alpha, \sin\alpha)$ come abbiamo già fatto nel disegno; allora la relazione (*) può anche essere scritta nella forma

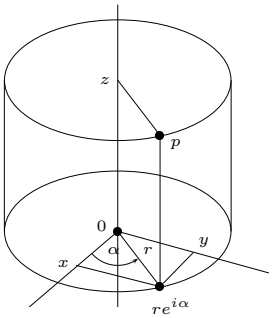
$$p = re^{i\alpha}$$

Questo prodotto di r con $e^{i\alpha}$ può essere interpretato come prodotto dello scalare reale r con il vettore $e^{i\alpha}$ di $\mathbb{R}^2$ ed è allo stesso tempo il prodotto dei numeri complessi r ed $e^{i\alpha}$ come si dimostra nei corsi di Analisi.

Coordinate cilindriche nello spazio

Dalla figura si vede che un punto $p = (x, y, z)$ dello spazio può essere rappresentato nella forma

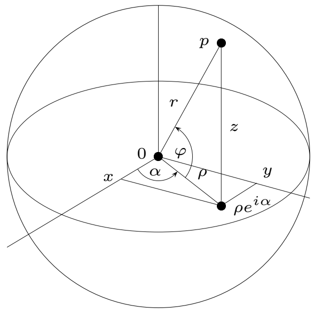
$$\boxed{\begin{aligned}x &= r\cos\alpha\\ y &= r\sin\alpha\\ z &= z\end{aligned}}$$



La rappresentazione è univoca per quei punti per cui $(x, y) \neq (0, 0)$, quindi per tutti i punti che non si trovano sull'asse z.

Coordinate polari (o sferiche) nello spazio

Un punto $p = (x, y, z)$ dello spazio tridimensionale può anche essere rappresentato come nella figura seguente:



Avendo $\rho = r\cos\varphi$, si vede che

$$\boxed{\begin{aligned} x &= r\cos\alpha\cos\varphi \\ y &= r\sin\alpha\cos\varphi \\ z &= r\sin\varphi \end{aligned}}$$

con

$$r \geq 0$$
$$0 \leq \alpha < 2\pi$$
$$-\frac{\pi}{2} \leq \varphi \leq \frac{\pi}{2}$$

Questa rappresentazione è quella che si usa nelle coordinate geografiche di un punto della terra o della sfera celeste:

$\alpha =$ longitudine, $\varphi =$ latitudine.

Anche in questo caso la corrispondenza non è biiettiva, perché non solo per $p = (0, 0, 0)$ la rappresentazione è valida per $r = 0$ e valori arbitrari di α e φ, ma anche per ogni altro punto $\neq (0, 0, 0)$ dell'asse z bisogna porre $\varphi = 90^\circ$ e quindi $\cos\varphi = 0$ e $\sin\varphi = 1$ se $z > 0$ oppure $\varphi = -90^\circ$ e quindi $\cos\varphi = 0$ e $\sin\varphi = -1$ se $z < 0$, e allora ogni α va bene. Quindi su tutta l'asse z le coordinate polari non sono univocamente determinate.

Spesso al posto di φ si usa $\theta := 90^\circ - \varphi$, quindi

$$\cos\varphi = \sin\theta \ \text{ e } \ \sin\varphi = \cos\theta$$

Molte funzioni della matematica e della fisica presentano *simmetrie*. A una funzione $f = f(x, y, z)$ definita su $\mathbb{R}^3$ (per semplicità, ma spesso bisognerà studiare bene il più adatto dominio di definizione) possiamo associare la funzione $g = g(r, \alpha, \varphi)$ definita da

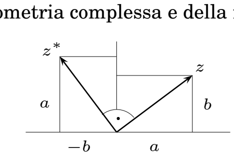
$$g(r, \alpha, \varphi) := f(r\cos\alpha\cos\varphi, r\sin\alpha\cos\varphi, r\sin\varphi)$$

che in caso di simmetrie può avere una forma analitica molto più semplice della f.

$f(x, y, z) = x^2 + y^2 + z^2$ ad esempio diventa così $g(r, \alpha, \varphi) = r^2$, una funzione di una sola variabile notevolmente più semplice. Altre volte una funzione dipende solo dalla direzione e quindi non da r; in questo caso g è una funzione di sole due variabili e anche questa è una importante semplificazione. Nello stesso modo si usano le coordinate cilindriche e le coordinate polari piane.

Il vettore magico

Per un vettore $z = (a, b)$ del piano il vettore $z^* := (-b, a)$ che si ottiene da z per rotazione di 90° in senso antiorario si chiama il *vettore magico* di z. Questo vettore è molto importante: riapparirà continuamente non solo nella geometria elementare del piano, ma anche in molti contesti della geometria complessa e della meccanica!

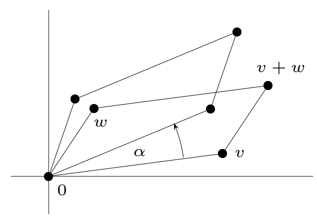


Rotazioni nel piano

Consideriamo l'applicazione f_α da $\mathbb{R}^2$ in $\mathbb{R}^2$ che consiste nel ruotare un punto $v = (v_1, v_2)$ per l'angolo fissato α in senso antiorario.
È chiaro che $f_\alpha(\lambda v) = \lambda f_\alpha(v)$ per ogni $\lambda \in \mathbb{R}$ e dal disegno si vede che anche

$$f_\alpha(v + w) = f_\alpha(v) + f_\alpha(w)$$

per $v, w \in \mathbb{R}^2$. Una rotazione è quindi un'applicazione lineare.



Sia e_1, e_2 la base canonica di $\mathbb{R}^2$. Allora $v = v_1e_1 + v_2e_2$, perciò $f_\alpha(v) = v_1f_\alpha(e_1) + v_2f_\alpha(e_2)$.

Ma $f_\alpha(e_1) = \begin{pmatrix} \cos\alpha \\ \sin\alpha \end{pmatrix}$ e $f_\alpha(e_2) = \begin{pmatrix} -\sin\alpha \\ \cos\alpha \end{pmatrix}$.

$f_\alpha(e_2)$ è il vettore magico di $f_\alpha(e_1)$!

Quindi

$$f_\alpha(v) = v_1\begin{pmatrix} \cos\alpha \\ \sin\alpha \end{pmatrix} + v_2\begin{pmatrix} -\sin\alpha \\ \cos\alpha \end{pmatrix} = \begin{pmatrix} v_1\cos\alpha - v_2\sin\alpha \\ v_1\sin\alpha + v_2\cos\alpha \end{pmatrix}$$

Se per una matrice $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \mathbb{R}^2_2$

definiamo $Av = \begin{pmatrix} av_1 + bv_2 \\ cv_1 + dv_2 \end{pmatrix}$,

vediamo che possiamo prendere

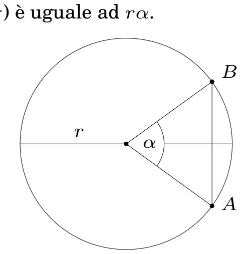
$$A = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix}$$

Notiamo anche che le colonne di A sono proprio $f_\alpha(e_1)$ e $f_\alpha(e_2)$.

Esercizi per gli scritti

14. $e^{i(\alpha+\beta)} = e^{i\alpha} \cdot e^{i\beta}$ per $\alpha, \beta \in \mathbb{R}$.

15. Dalla figura si vede che la lunghezza c della corda (cioè del segmento di retta) tra A e B è uguale a $2r\sin(\alpha/2)$, mentre la distanza (cioè l'arco) d sul cerchio tra A e B (nell'ipotesi $0 \leq \alpha < \pi$) è uguale ad $r\alpha$.



La funzione cordale $\underset{x}{\bigcirc}\, 2\sin\frac{x}{2}$ è probabilmente la più antica funzione trigonometrica e venne tabulata già da Ipparco da Nikaia nel secondo secolo prima di Cristo (*tavola delle corde*).

Calcolare la differenza $d - c$, che corrisponde all'errore che si commette usando c al posto di d per misurare la distanza tra i punti A e B sulla sfera terrestre, che possiede un raggio medio r di 6371 km, per $c = 1$ km, 10 km, 50 km, 100 km, 500 km, 1000 km.

Attenzione: Se $\sin\beta = u$, come si calcola β?

IV. PYTHON

Installazione

Python è in questo momento forse il miglior linguaggio di programmazione: per la facilità di apprendimento e di utilizzo, per le caratteristiche di linguaggio ad altissimo livello che realizza i concetti sia della programmazione funzionale che della programmazione orientata agli oggetti, per il recente perfezionamento della libreria per la programmazione insiemistica, per il supporto da parte di numerosi programmatori, per l'ottima documentazione disponibile in rete e la ricerca riuscita di meccanismi di leggibilità, per la grafica con Tkinter, per la semplicità dell'aggancio ad altri linguaggi, di cui il modulo RPy per il collegamento con R è un esempio meraviglioso.

Dal sito del corso installare nell'ordine indicato:

Enthought Python 2.4.3
R 2.2.1
pywin32-209.win32-py2.4.exe
rpy-0.4.6-R-2.0.0-to-2.2.1-py24.win32.exe

Attualmente per Windows sono queste le versioni compatibili con l'utilizzo di RPy anche se di R esiste già la versione 2.3.

Fahrenheit e Celsius

Scriviamo due semplici funzioni per la conversione tra Fahrenheit e Celsius. Se f è la temperatura espressa in gradi Fahrenheit e c la stessa temperatura espressa in gradi Celsius, allora vale la relazione

$$c = (f - 32)/1.8$$

e quindi $f = 1.8c + 32$.

```
def celsiodafahrenheit (f):
  return (f-32)/1.8

def fahrenheitdacelsio (c):
  return 1.8*c+32
```

Queste funzioni vengono utilizzate nel modo seguente.

```
cdf=celsiodafahrenheit
fdc=fahrenheitdacelsio

for n in (86,95,104): print n,cdf(n)

print

for n in (20,30,35,40): print n,fdc(n)
```

Otteniamo l'output

```
86 30.0
95 35.0
104 40.0

20 68.0
30 86.0
35 95.0
40 104.0
```

Esecuzione di un programma in Python

In una cartella *Python* (oppure, per progetti più importanti, in un'apposita sottocartella per quel progetto) creiamo i files sorgente come files di testo puro con l'estensione *.py*, utilizzando l'editor incorporato di Python. Con lo stesso editor, piuttosto comodo, scriviamo anche, usando l'estensione *.r*, le sorgenti in R che vogliamo affiancare ai programmi in Python. I programmi possono essere eseguiti mediante il tasto *F5* nella finestra dell'editor. Soprattutto in fase di sviluppo sceglieremo questa modalità di esecuzione, perché così vengono visualizzati anche i messaggi d'errore.

Successivamente i programmi possono essere eseguiti anche tramite il clic sull'icona del file oppure, in un terminale (prompt dei comandi) e se il file si chiama *alfa.py*, con il comando `python alfa.py`.

Teoricamente i programmi possono essere scritti con un qualsiasi editor che crea files in formato testo puro, ad esempio il *Blocco note* di Windows, ma preferiamo utilizzare l'editor di Python per una più agevole correzione degli errori, per l'indentazione automatica e perché prevede la possibilità di usare combinazioni di tasti più comode di quelle disponibili per il Blocco note.

Elenchiamo alcune combinazioni di tasti:

Ctrl Q	uscire
F5	esecuzione
Ctrl N	nuovo file
Ctrl O	aprire un file
Alt G	trova una riga
Ctrl A	seleziona tutto
Ctrl C	copia il testo selezionato
Ctrl V	incolla
Ctrl X	cancella il testo selezionato
Ctrl K	cancella il resto della riga
Inizio	inizio della riga
Fine	fine della riga
Ctrl E	fine della riga
Alt P	lista dei comandi dati: indietro
Alt N	lista dei comandi dati: avanti

Se una riga contiene, al di fuori di una stringa, il carattere `#`, tutto il resto della riga è considerato un *commento*, compreso il carattere `#` stesso.

Molti altri linguaggi interpretati (Perl, R, la shell di Unix) usano questo simbolo per i commenti. In C e C++ una funzione analoga è svolta dalla sequenza `//`.

Per importare le istruzioni contenute in un file `beta.py` si usa il comando `import beta`, tralasciando l'estensione. Con lo stesso comando si importano anche i moduli delle librerie incorporate o prelevate in rete:

```
import os, math, scipy
```

Enthought Python

Questa è una raccolta molto ricca (di 124 MB compressi) per Windows che non comprende soltanto il linguaggio Python (attualmente nella versione 2.4.3), ma anche numerose librerie scientifiche:

wxPython. Un'interfaccia alla libreria grafica *wxWidgets*.

PIL. La *Python Imaging Library*, una libreria per l'elaborazione delle immagini.

VTK. Il *3D Visualization Toolkit*, un sofisticato pacchetto di grafica 3-dimensionale.

MayaVi. Una raccolta di programmi per la visualizzazione 3-dimensionale di dati scientifici basata su VTK.

Numeric. Un pacchetto di calcolo numerico, soprattutto matriciale.

ScientificPython. Libreria per il calcolo numerico scientifico.

SciPy. Una nuova libreria per il calcolo scientifico.

Matplotlib. Funzioni di visualizzazione matematica in 2 dimensioni.

Chaco. Una libreria per la grafica scientifica sviluppata dalla Enthought.

Traits. Uno strumento di sviluppo per la programmazione orientata agli oggetti.

PyCrust. Una shell per Python che fa parte di *wxPython*.

ZODB3. Due sistemi di basi di dati orientate agli oggetti.

Gadfly. Un sistema di basi di dati relazionali.

PySQLite. Un'estensione per SQLite.

PyXML. Strumenti per XML.

ctypes. Un pacchetto per la creazione di tipi di dati per il C.

Per il matematico sono particolarmente utili i pacchetti scientifici e grafici.

Primi esempi in Python

```
a=range(5,13,2)
print a
# output: [5, 7, 9, 11]

a=range(5,13)
print a
# output: [5, 6, 7, 8, 9, 10, 11, 12]

a=range(11)
print a
# output:
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Si noti che il limite destro non viene raggiunto.

```
a=xrange(5,13,2)
print a
# output: xrange(5, 13, 2)

for x in a: print x,
# output: 5 7 9 11
```

La differenza tra `range` e `xrange` è questa: Mentre `range(1000000)` genera una lista di un milione di elementi, `xrange(1000000)` è un *iteratore* (pagina 16) che crea questi elementi uno allo volta in ogni passaggio di un ciclo in cui il comando viene utilizzato.

Si noti il doppio punto (`:`) alla fine del comando `for`.

Sono possibili assegnazioni e confronti simultanei:

```
if 3<5<9: print 'o.k.'
# output: o.k.

a=b=c=4
for x in [a,b,c]: print x,
print
# output: 4 4 4
```

Funzioni in Python:

```
def f(x): return 2*x+1

def g(x):
  if (x>0): return x
  else: return -x

for x in xrange(0,10): print f(x),
print
# output: 1 3 5 7 9 11 13 15 17 19

for x in xrange(-5,5): print g(x),
print
# output: 5 4 3 2 1 0 1 2 3 4
```

A differenza da R e Lisp, il `return` è obbligatorio.

La virgola alla fine di un comando `print` fa in modo che la stampa continui sulla stessa riga. Come si vede nella definizione di g, Python utilizza l'indentazione per strutturare il programma. Anche le istruzioni `if` ed `else` richiedono il doppio punto.

Una funzione di due variabili:

```
import math

def raggio (x,y):
  return math.sqrt(x**2+y**2)

print raggio(2,3)
# output: 3.60555127546

print math.sqrt(13)
# output: 3.60555127546
```

Funzioni possono essere non solo argomenti, ma anche risultati di altre funzioni:

```
def sommax (f,g,x): return f(x)+g(x)

def compx (f,g,x): return f(g(x))

def u(x): return x**2

def v(x): return 4*x+1

print sommax(u,v,5)
# output: 46

print compx(u,v,5)
# output: 441
```

Possiamo però anche definire

```
def somma (f,g):
  def s(x): return f(x)+g(x)
  return s

def comp (f,g):
  def c(x): return f(g(x))
  return c

def u(x): return x**2

def v(x): return 4*x+1

print somma(u,v)(5)
# output: 46

print comp(u,v)(5)
# output: 441
```

Queste costruzioni significano che Python appartiene (come R, Perl e Lisp) alla famiglia dei potenti linguaggi *funzionali*.

Stringhe sono racchiuse tra apici o virgolette, stringhe su più di una riga tra triplici apici o virgolette:

```
print 'Carlo era bravissimo.'
# output: Carlo era bravissimo.

print "Carlo e' bravissimo."
# output: Carlo e' bravissimo.

print '''Stringhe a piu' righe si
usano talvolta nei commenti.'''
# output:
# Stringhe a piu' righe si
# usano talvolta nei commenti.
```

Funzioni con un numero variabile di argomenti: Se una funzione è dichiarata nella forma `def f(x,y,*a):`, l'espressione `f(2,4,5,7,10,8)` viene calcolata in modo che gli ultimi argomenti vengano riuniti in una tupla `(5,7,10,8)` che nel corpo del programma può essere vista come se questa tupla fosse a.

```
def somma (*a):
  s=0
  for x in a: s+=x
  return s

print somma(1,2,3,10,5)
# output: 21
```

Lo schema di Horner per il calcolo dei valori $f(\alpha)$ di un polinomio $f = a_0x^n + \ldots + a_n$ consiste nella ricorsione

$$b_{-1} = 0$$
$$b_k = \alpha b_{k-1} + a_k$$

per $k = 0, \ldots, n$. Possiamo quindi definire

```
def horner (alfa,*a):
  alfa=float(alfa); b=0
  for t in a: b=b*alfa+t
  return b

print horner(10,6,2,0,8)
# output: 6208.0
```

Vettori associativi (dizionari o tabelle di hash) vengono definiti nel modo seguente:

```
latino = {'casa': 'domus',
  'villaggio': 'pagus',
  'nave':'navis', 'campo':'ager'}
voci=sorted(latino.keys())
for x in voci:
  print '%-9s = %s' %(x,latino[x])
# output:
# campo     = ager
# casa      = domus
# nave      = navis
# villaggio = pagus
```

Scambi simultanei:

```
a=5; b=3; a,b=b,a; print [a,b]
# output: [3, 5]
```

Uguaglianza:

```
a==b # a e b hanno lo stesso valore
a!=b # a e b hanno valori diversi
a=b # assegnamento
```

Input dalla tastiera

Per l'input dalla tastiera sono previsti le funzioni `raw_input` e `input`. Entrambe accettano un argomento facoltativo che, quando è presente, viene visualizzato sullo schermo prima dell'input dell'utente. Mentre `input` aspetta dalla tastiera una espressione valida in Python che viene calcolata come se facesse parte di un programma, `raw_input` tratta la risposta dell'utente come una stringa. Se ad esempio vogliamo immettere una stringa con `input`, la dobbiamo racchiudere tra apici, mentre con `raw_input` gli apici verrebbero considerati come lettere della stringa.

```
>>> x=input('nome: ')
nome: Giacomo
...
NameError: name 'Giacomo' is not defined
>>> # Giacomo non e' una stringa.

>>> x=input('nome: ')
nome: 'Giacomo'
>>> print x
Giacomo

>>> x=raw_input('nome: ')
nome: Giacomo
>>> print x
Giacomo

>>> x=raw_input('nome: ')
nome: 'Giacomo'
>>> print x
'Giacomo'

>>> def f(x): return x**2

>>> x=input('espressione: ')
espressione: f(9)
>>> print x
81

>>> x=raw_input('espressione: ')
espressione: f(9)
>>> print x
f(9)
```

R ed S-Plus

R è un linguaggio di programmazione ad altissimo livello orientato soprattutto all'uso in statistica. In verità lo sbilanciamento verso la statistica non deriva dalla natura del linguaggio, ma dalla disponibilità di grandi raccolte di funzioni statistiche e dagli interessi dei ricercatori che lo hanno inventato e lo mantengono. R è gratuito e molto simile a un linguaggio commerciale, S, creato negli anni '80 e anch'esso molto usato. S viene commercializzato come sistema S-Plus. Le differenze non sono grandissime se non sul piano della programmazione, dove R aderisce a una impostazione probabilmente più maneggevole.

R ed S-Plus sono particolarmente popolari nella statistica medica, ma vengono anche usati nella statistica economica o sociale, in geografia, nella matematica finanziaria. L'alto livello del linguaggio permette di creare facilmente librerie di funzioni per nuove applicazioni. Il punto debole è la velocità di esecuzione in calcoli numerici in grandi dimensioni, mentre sono ricchissime le capacità grafiche.

Benché così indirizzato verso la statistica, R non deve essere considerato un pacchetto di statistica. È un vero linguaggio di programmazione, anzi un linguaggio di programmazione molto avanzato, e ciò permette di adattarlo ad ogni compito informatico. Nella stessa statistica questa flessibilità è molto importante proprio oggi, dove continuamente si scoprono nuovi bisogni applicativi, nuove necessità di tradurre metodi matematici, ad esempio nella statistica di complessi dati clinici o geografici, in strumenti informatici.

Un'introduzione alla programmazione in R si trova nel corso di Fondamenti di informatica 2004/05 e, per quanto riguarda la grafica, nel corso di Algoritmi e strutture di dati 2004/05.

Utilizzo di RPy

Il modulo RPy è un piccolo miracolo e permette una quasi perfetta e semplicissima collaborazione tra Python ed R.

Sotto Linux il pacchetto va installato nel modo seguente: In primo luogo è necessario che R sia stato creato in modo che si possano utilizzare le librerie condivise:

```
./configure --enable-R-shlib
make
make install
```

Successivamente va aggiunta la riga

```
/usr/local/lib/R/lib
```

nel file */etc/ld.so.conf* ed eseguito il comando `ldconfig`.

A questo punto, per installare RPy stesso, è sufficiente

```
/usr/local/bin/python setup.py install
```

Per importare il pacchetto scriviamo

```
from rpy import r
```

nel programma in Python. Le funzioni di R possono allora essere usate anteponendo il prefisso `r.`, come in `r.fun(argomenti)`.

Sotto Linux bisogna (per un piccolo errore contenuto nel modulo) inserire l'istruzione `r.q()` alla fine del programma.

Definiamo ad esempio una funzione in Python che utilizza la funzione `mean` di R per calcolare la media di un vettore:

```
def Media (x): return r.mean(x)
```

Per provare la funzione usiamo

```
print Media([1,5,8,6,3,1])
# output: 4.0
```

In particolare possiamo usare la funzione `source` di R. Ciò significa che possiamo creare una raccolta di funzioni in R da noi programmate; queste funzioni possono a loro volta utilizzare (come se fossimo in una libreria creata per R) le altre funzioni di quella raccolta e allo stesso tempo essere usate, nella sintassi indicata, nei programmi in Python! Creiamo ad esempio un file *funz.r*:

```
# funz.r

cubo = function (x): x^2
```

In uno script di Python scriviamo poi

```
r.source('funz.r')

print r.cubo(13)
# output: 2197.0
```

Liste

Liste sono successioni finite modificabili di elementi non necessariamente dello stesso tipo. Python fornisce numerose funzioni per liste che non esistono per le tuple. Come le tuple anche le liste possono essere annidate, la lunghezza di una lista `v` la si ottiene con `len(v)`, l'`i`-esimo elemento è `v[i]`. A differenza dalle tuple, la lista che consiste solo di un singolo elemento `x` è denotata con `[x]`.

```
v=[1,2,5,8,7]

for i in xrange(5): print v[i],
print
# 1 2 5 8 7

for x in v: print x,
print
# 1 2 5 8 7

a=[1,2,3]; b=[4,5,6]
v=[a,b]

print v
# [[1, 2, 3], [4, 5, 6]]

for x in v: print x
# [1, 2, 3]
# [4, 5, 6]

print len(v)
# 2
```

Sequenze

Successioni finite in Python vengono dette *sequenze*, di cui i tipi più importanti sono liste, tuple e stringhe. Tuple e stringhe sono sequenze non modificabili. Esistono alcune operazioni comuni a tutte le sequenze che adesso elenchiamo. `a` e `b` siano sequenze:

`x in a`	Vero, se `x` coincide con un elemento di `a`.
`x not in a`	Vero, se `x` non coincide con nessun elemento di `a`.
`a + b`	Concatenazione di `a` e `b`.
`a * k`	Concatenazione di `k` copie di `a`.
`a[i]`	`i`-esimo elemento di `a`.
`a[-1]`	Ultimo elemento di `a`.
`a[i:j]`	Sequenza che consiste degli elementi `a[i]`, ..., `a[j-1]` di `a`.
`a[:]`	Copia di `a`.
`len(a)`	Lunghezza di `a`.
`min(a)`	Più piccolo elemento di `a`. Per elementi non numerici viene utilizzato l'ordine alfabetico.
`max(a)`	Più grande elemento di `a`.
`sorted(a)`	Una *lista* che contiene gli elementi di `a` in ordine crescente.
`reversed(a)`	*iteratore* che corrisponde agli elementi di `a` elencati in ordine invertito.
`list(a)`	Converte la sequenza in una lista con gli stessi elementi.

Come abbiamo visto, l'espressione `x in a` può apparire anche in un ciclo `for`.

Iteratori sono oggetti che forniscono uno dopo l'altro tutti gli elementi di una sequenza senza creare questa sequenza in memoria. Ad esempio sono iteratori gli oggetti che vengono creati tramite l'istruzione `xrange`.

La funzione `list` può essere applicata anche agli iteratori, per cui per generare la lista `b` che si ottiene da una sequenza `a` invertendo l'ordine in cui sono elencati i suoi elementi, possiamo utilizzare `b=list(reversed(a))`. Esempi:

```
if 'a' in 'nave': print "Si'."
if 7 in [3,5,1,7,10]: print "Si'."
if 2 in [3,5,1,7,10]: print "Si'."
else: print "No."

a=[1,2,3,4]; b=[5,6,7,8,9]
print a+b
# [1, 2, 3, 4, 5, 6, 7, 8, 9]

a='Mario'; b='Rossi'; print a+' '+b
# Mario Rossi

print [1,2,3]*4
# [1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]

a='==.==.'*3
print a
# ==.====.====.==

a=[10,11,12,13,14,15,16,17,18,19,20]
for i in xrange(0,11,3): print a[i],
print # 10 13 16 19

a='01234567'; print a[2:5] # 234

a=[3,5,3,1,0,1,2,1]
b=sorted(a)
print b # [0, 1, 1, 1, 2, 3, 3, 5]

a=[1,2,3,4,5]
print reversed(a)
# <listreverseiterator object at ...>
# L'indirizzo dopo at cambia ogni volta.

d=list(reversed(a))
print d # [5, 4, 3, 2, 1]

a='MARIO'; print list(a)
# ['M', 'A', 'R', 'I', 'O']
```

Funzioni per liste

Per le liste sono disponibili alcune funzioni speciali che non possono essere utilizzate per tuple o stringhe. `v` sia una lista.

`v.append(x)`	`x` viene aggiunto alla fine di `v`. Equivalente a `v=v+[x]`, ma più veloce.
`v.extend(w)`	Aggiunge la lista `w` a `v`. Equivalente a `v=v+w`, ma più veloce.
`v.count(x)`	Il risultato indica quante volte `x` appare in `v`.
`v.index(x)`	Indice della prima posizione in cui `x` appare in `v`; provoca un errore se `x` non è elemento di `v`.
`v.insert(i,x)`	Inserisce l'elemento `x` come `i`-esimo elemento della lista.
`v.remove(x)`	Elimina `x` nella sua prima apparizione in `v`; provoca un errore se `x` non è elemento di `v`.
`v.pop()`	Toglie dalla lista il suo ultimo elemento che restituisce come risultato; errore, se il comando viene applicato alla lista vuota.
`v.sort()`	Ordina la lista che viene modificata.
`v.reverse()`	Inverte l'ordine degli elementi in `v`. La lista viene modificata.

Esempi:

```
v=[1,2,3,4,5,6]
v.append(7)
print v
# [1, 2, 3, 4, 5, 6, 7]

v=[2,8,2,7,2,2,3,3,5,2]
print v.count(2)
# 5
print v.count(77)
# 0

print v.index(7)
# 3

v=[10,11,12,13,14,15,16]
v.insert(4,99)
print v
# [10, 11, 12, 13, 99, 14, 15, 16]

v=[2,3,8,3,7,6,3,9]
v.remove(3)
print v
# [2, 8, 3, 7, 6, 3,9]

v.pop()
print v
# [2, 8, 3, 7, 6, 3]

v.sort()
print v
# [2, 3, 3, 6, 7, 8]

v=[7,0,1,0,2,3,3,0,5]
v.sort()
print v
# [0, 0, 0, 1, 2, 3, 3, 5, 7]

v=[0,1,2,3,4,5,6,7]
v.reverse()
print v
# [7, 6, 5, 4, 3, 2, 1, 0]
```

Tuple

Tuple sono successioni finite *non modificabili* di elementi non necessariamente dello stesso tipo che sono elencati separati da virgole e possono facoltativamente essere incluse tra parentesi tonde. Per ragioni misteriose una tupla con un solo elemento deve essere scritta nella forma `(x,)` perché `(x)` è lo stesso come `x`. Ciò non vale per le liste: `[x]` è una lista. Tuple possono essere annidate.

Esempi:

```
x=3,5,8,9
print x
# (3, 5, 8, 9)

y=(3,5,8,9)
print y
# (3, 5, 8, 9)

s=(7)
print s
# 7

t=(7,)
print t
# (7,)

z=(x)
print z
# (3, 5, 8, 9)

u=(x,)
print u
# ((3, 5, 8, 9),)

v=(x,y)
print v
# ((3, 5, 8, 9), (3, 5, 8, 9))

w=1,2,(3,4,5),6,7
print w
# (1, 2, (3, 4, 5), 6, 7)

vuota=()
print vuota
# ()
```

La lunghezza di una tupla `v` la otteniamo con `len(v)`; l'`i`-esimo elemento di `v` è `v[i]`, contando (come in C e a differenza da R!) cominciando da 0.

```
x=3,5,8,9
print len(x)
# 4

for i in xrange(0,4): print x[i],
print
# 3 5 8 9

for a in x: print a,
# 3 5 8 9
```

Tuple vengono elaborate più velocemente e consumano meno spazio in memoria delle liste; per sequenze molto grandi (con centinaia di migliaia di elementi) o sequenze che vengono usate in migliaia di operazioni le tuple sono perciò talvolta perferibili alle liste. Liste d'altra parte non solo possono essere modificate, ma prevedono anche molte operazioni flessibili che non sono disponibili per le tuple. Le liste costituiscono una delle strutture fondamentali di Python. Con dati molto grandi comunque l'utilizzo di liste, soprattutto nei calcoli intermedi, può effettivamente rallentare notevolmente l'esecuzione di un programma.

Nomi ed assegnamento

A differenza dal C, il Python non distingue tra il nome `a` di una variabile e il suo indirizzo (che in C viene denotato con `&a`). Ciò ha implicazioni a prima vista sorprendenti nelle assegnazioni `b=a` in cui `a` è un oggetto mutabile (ad esempio una lista o un dizionario), mentre nel caso che `a` non sia mutabile (ad esempio un numero, una stringa o una tupla) non si avverte una differenza con quanto ci si aspetta.

Dopo `b=a` infatti `a` e `b` sono nomi diversi per lo stesso oggetto e se modifichiamo l'oggetto che corrisponde ad `a`, senza però assegnare il nome a ad un altro oggetto, lo troviamo cambiato anche quando usiamo il nome `b` proprio perché si tratta sempre dello stesso oggetto.

```
a=7; b=a; b=3; print a # 7
a=7; b=a; a=3; print b # 7

a=[1,5,0,2]; b=a; b.sort(); print a
# [0, 1, 2, 5]

b[2]=17; print a
# [0, 1, 17, 5]

b=a[:].sort(); print a
# [0, 1, 17, 5]
# a non e' cambiata.

b=a[:].reverse(); print a
# [0, 1, 17, 5]
# a non e' cambiata.
```

Se gli elementi di `a` sono immutabili, è sufficiente, come sopra, creare una copia `b=a[:]` oppure `b=list(a)` affinché cambiamenti in `b` non influenzino `a`. Ciò non basta più quando gli elementi di `a` sono mutabili, come ad esempio nel caso che `a` sia una lista annidata:

```
a=[[1,2],[3,4]]
b=a[:] # oppure b=list(a)
b[1][0]=17; print a
# [[1, 2], [17, 4]]
```

In questo caso bisogna creare una *copia profonda* utilizzando la funzione `copy.deepcopy` che naturalmente richiede il modulo `copy`:

```
import copy

a=[[1,2],[3,4]]
b=copy.deepcopy(a)
b[1][0]=17; print a
# [[1, 2], [3, 4]]
# a non e' cambiata.
```

Per verificare se due nomi denotano lo stesso oggetto, si può usare la funzione `is`, mentre l'operatore di uguaglianza `==` controlla soltanto l'uguaglianza degli elementi, non degli oggetti che corrispondono ai nomi `a` e `b`:

```
a=[1.5,0,2]; b=a
print b is a
# True

a=[[1,2],[3,4]]; b=a[:]
print b is a
# False
print b[0] is a[0]
# True

a=[[1,2],[3,4]]
b=copy.deepcopy(a)
print b is a
# False
print b[0] is a[0]
# False
print b==a
# True
```

Valori di verità

Vero e falso in Python possono essere rappresentati dai valori `True` e `False`. In un contesto logico, cioè nei confronti o quando sono argomenti degli operatori logici, anche ad altri oggetti è attribuito un valore di verità; a differenza dal C però essi conservano il loro valore originale e il risultato di un'espressione logica in genere non è un valore di verità, ma uno degli argomenti da cui si partiva.

Con

```
a="Roma"; b="Torino"
for x in (3<5, 3<5<7, 3<5<4,
  6==7, 6==6,a=='Roma',a<b):
  print x,
```

otteniamo

```
True True False False True True True
```

perché la stringa `"Roma"` precede alfabeticamente `"Torino"`. Con

```
for x in (0,1,0.0,[],(),[0],
  None,'','alfa'):
  print "%-6s%s" %(x, bool(x))
```

otteniamo

```
0     False
1     True
0.0   False
[]    False
()    False
[0]   True
None  False
      False
alfa  True
```

Vediamo così che il numero 0, l'oggetto `None`, la stringa vuota, e la lista o la tupla vuota hanno tutti il valore di verità falso, numeri diversi da zero, liste, tuple e stringhe non vuote il valore di verità vero.

In un contesto numerico i valori di verità vengono trasformati in 1 e 0:

```
print (3<4)+0.7
# 1.7

v=[3<4,3<0]
for x in v: print x>0.5,
# True False

print
from rpy import r

a=[True,True,False,True,False]
print r.sum(a)
# 3
```

for

La sintassi di base del `for` ha, come sappiamo, la forma

```
for x in v: istruzioni
```

dove `v` è una sequenza o un iteratore.

Dal `for` si esce con `break` (o naturalmente, dall'interno di una funzione, con `return`), mentre `continue` interrompe (come il `next` di R) il passaggio corrente di un ciclo e porta all'immediata esecuzione del passaggio successivo, cosicché

```
for x in xrange(0,21):
  if x%2>0: continue
  print x,
# 0 2 4 6 8 10 12 14 16 18 20
```

stampa sullo schermo i numeri pari compresi tra 1 e 20.

Il `for` può essere usato anche nella forma

```
for x in v: istruzioni
else: istruzionefinale
```

Quando le istruzioni nel `for` stesso non contengono un `break`, questa sintassi è equivalente a

```
for x in v: istruzioni
```

Un `break` invece *salta* la parte `else` che quindi viene eseguita solo se tutti i passaggi previsti nel ciclo sono stati effettuati. Questa costruzione viene utilizzata quando il percorso di tutti i passaggi è considerato come una condizione di eccezione: Assumiamo ad esempio che cerchiamo in una sequenza un primo elemento con una determinata proprietà - una volta trovato, usciamo dal ciclo e continuiamo l'elaborazione con questo elemento; se invece un elemento con quella proprietà non si trova, abbiamo una situazione diversa che trattiamo nel `else`. Nei due esempi che seguono cerchiamo il primo elemento positivo di una tupla di numeri:

```
for x in (-1,0,0,5,2,-3,4):
  if x>0: print x; break
else: print 'Nessun elemento positivo.'
# 5

for x in (-1,0,0,-5,-2,-3,-4):
  if x>0: print x; break
else: print 'Nessun elemento positivo.'
# Nessun elemento positivo.
```

Cicli `for` possono essere annidati; con

```
for i in xrange(4):
  for j in xrange(5):
    print i+j,
  print
```

otteniamo

```
0 1 2 3 4
1 2 3 4 5
2 3 4 5 6
3 4 5 6 7
```

Se gli elementi della sequenza che viene percorsa dal `for` sono a loro volta sequenze tutte della stessa lunghezza, nel `for` ci possiamo riferire agli elementi di queste sequenze con nomi di variabili:

```
u=[[1,10],[2,10],[3,10],[4,20]]
for x,y in u: print x+y,
print
# 11 12 13 24

v=['Aa','Bb','Cc','Dd','Ee']
for x,y in v: print y+'.'+x,
print
# a.A b.B c.C d.D e.E

w=[[1,2,5],[2,3,6],[11,10,9]]
for x,y,z in w: print x*y+z,
# 7 12 119
```

while

Il `while` controlla cicli più generali del `for` e gli è molto simile nella sintassi:

```
while A: istruzioni
```

oppure

```
while A: istruzioni
else: istruzionefinale
```

`break` e `continue` vengono utilizzati come nel `for`. Se è presente un `else`, l'istruzione finale viene eseguita se l'uscita dal ciclo è avvenuta perché la condizione `A` non era più soddisfatta, mentre viene saltata se si è usciti con un `break` o un `return`.

```
x=0; v=[]
while not x in v:
  v.append(x)
  x=(7*x+13)%17

for x in v: print x,
print
# 0 13 2 10 15 16 6 4 7 11 5 14 9 8 1 3

while 1:
  nome=raw_input('Come ti chiami? ')
  if nome=='': break
  print 'Ciao, %s!' %(nome)
```

Espressioni lambda

L'espressione `lambda x: f(x)` corrisponde all'espressione `function (x) f(x)` di R; con più variabili diventa `lambda x,y: f(x,y)`. A differenza da R però in Python `f(x)` deve essere un'espressione unica che soprattutto non può contenere istruzioni di assegnamento o di controllo; manca anche il `return`. Ciononostante il valore di un'espressione lambda è una funzione che può essere usata come valore di un'altra funzione:

```
def u(x): return x**2
def v(x): return 4*x+1

def comp (f,g): return lambda x: f(g(x))

print comp(u,v)(5) # 441
```

Qui abbiamo ridefinito la funzione `comp` vista a pagina 15. Possiamo anche assegnare un nome alla funzione definita da un'espressione lambda:

```
f=lambda x,y: x+y-1
print f(6,2) # y
```

I `lambda` possono essere annidati. Un'espressione lambda senza variabile è una funzione costante:

```
def costante (x): return lambda : x

f=costante(10); print f() # 10
```

Per uno spazio vettoriale V possiamo definire un'iniezione canonica

$$j := \bigcirc_{\alpha} \alpha(v) : V \longrightarrow V''$$

di V nel suo doppio duale; se V è di dimensione finita, j è un isomorfismo. Possiamo imitare j in Python con

```
def incan (v): return lambda a: a(v)
```

Il modulo math

Questo modulo contiene le seguenti funzioni e costanti, corrispondenti per la maggior parte ad analoghe funzioni del C.

Funzioni trigonometriche: `math.cos`, `math.sin` e `math.tan`.

Funzioni trigonometriche inverse: `math.acos`, `math.asin`, `math.atan` e `math.atan2`.

`math.atan2(y,x)` calcola l'angolo nella rappresentazione polare di un punto $(x, y) \neq (0, 0)$ nel piano. Attenti all'ordine degli argomenti!

```
print math.degrees(math.atan2(1,0))
# 90

print math.degrees(math.atan2(1,-1))
# 135.0

print math.degrees(math.atan2(-1,-1))
# -135.0
```

Funzioni per la conversione di angoli a gradi: `math.degrees(alfa)` esprime l'angolo `alfa` in gradi, `math.radians(gradi)` calcola il valore in radianti di un angolo espresso in gradi.

```
for x in (90,135,180,270,360):
  print math.radians(x)
# Output:

1.57079632679
2.35619449019
3.14159265359
4.71238898038
6.28318530718

for x in (1.6,2.4,3.1,4.7,6.3):
  print math.degrees(x)
# Output:
91.6732472209
137.509870831
177.616916491
269.290163711
360.963410932
```

Funzioni iperboliche: `math.cosh`, `math.sinh` e `math.tanh`.

Queste importanti funzioni sono così definite:

$$\cosh x := \frac{e^x + e^{-x}}{2}$$

$$\sinh x := \frac{e^x - e^{-x}}{2}$$

$$\tanh x := \frac{\sinh x}{\cosh x} = \frac{e^{2x} - 1}{e^{2x} + 1}$$

Esponenziale e logaritmi: `math.exp`, `math.pow`, `math.log`, `math.log10`, `math.ldexp`, `math.sqrt` e `math.hypot`.

Il logaritmo naturale di x lo si ottiene con `math.log(x)`, il logaritmo in base b con `math.log(x,b)`, il logaritmo in base 10 anche con `math.log10(x)`. `ldexp(a,n)` è uguale ad $a2^n$ e difficilmente ne avremo bisogno. Abbiamo già visto che `math.sqrt` calcola la radice di un numero reale > 0. Con `math.hypot(x,y)` otteniamo $\sqrt{x^2 + y^2}$. Invece di `math.pow` possiamo usare `pow`.

```
for x in (0,1,2,math.log(7)):
  print math.exp(x),
# 1.0 2.71828182846 7.38905609893 7.0

print
print math.ldexp(1,6)
# 64.0
print math.ldexp(2.5,6)
# 160.0

for b in (2,math.e,7,10):
  print round(math.log(10,b),4),
# 3.3219 2.3026 1.1833 1.0

print math.log(1024,2)
# 10.0
```

Alcune funzioni aritmetiche: `math.floor`, `math.ceil`, `math.fmod`.

`math.floor(x)` restituisce la parte intera del numero reale x, cioè il più vicino intero alla sinistra di x, `math.ceil` il più vicino intero alla destra di x. Entrambe le funzioni restituiscono valori reali e funzionano correttamente anche per argomenti negativi:

```
for x in (-1.3,-4,0.5,2,2.7):
  print math.floor(x), math.ceil(x)
# Output:

-2.0 -1.0
-4.0 -4.0
 0.0  1.0
 2.0  2.0
 2.0  3.0
```

Le funzioni `math.modf`, `math.fmod` e `math.frexp` non funzionano correttamente.

`math.fabs(x)` è il valore assoluto di x, ma possiamo usare la funzione `abs`.

Le costanti `math.e` e `math.pi` rappresentano i numeri e e π:

```
print math.e
# 2.71828182846

print math.pi
# 3.14159265359
```

In entrambi i casi l'ultima cifra è arrotondata in alto.

Il modulo cmath

Questo modulo fornisce (con il prefisso `cmath.`) le funzioni `cos`, `sin`, `tan`, `acos`, `asin`, `atan`, `cosh`, `sinh`, `tanh`, `acosh`, `asinh`, `atanh`, `exp`, `log`, `log10` e `sqrt` per argomenti complessi:

```
for k in xrange(1,8):
  print cmath.sin(k*math.pi*1j)
# Output:

11.5487393573j
267.744894041j
6195.82386361j
143375.656567j
3317811.99967j
76776467.6977j
1776660640.42j
```

Si vede chiaramente che sulla retta $\mathbb{R}i$ il seno cresce in modo esponenziale.

apply

f sia una funzione di n argomenti, dove n può essere anche variabile, e v una sequenza di lunghezza n. Allora `apply(f,v)` è uguale ad $f(v_1, \ldots, v_n)$. Esempi:

```
def somma (*a):
  s=0
  for x in a: s+=x
  return s

v=[1,2,4,9,2,8]
s=apply(somma,v)
print s # 26
```

---

reduce

f sia una funzione di due argomenti. Come in algebra scriviamo $a \cdot b := f(a, b)$. Per una sequenza $v = (a_1, \ldots, a_n)$ di lunghezza $n \geq 1$ l'espressione $\mathrm{reduce}(f, v)$ è definita come il prodotto (in genere non associativo) da sinistra verso destra degli elementi di v:

$$\begin{aligned}
\mathrm{reduce}(f, [a_1]) &= a_1 \\
\mathrm{reduce}(f, [a_1, a_2]) &= a_1 \cdot a_2 \\
\mathrm{reduce}(f, [a_1, a_2, a_3]) &= (a_1 \cdot a_2) \cdot a_3 \\
&\ldots \\
\mathrm{reduce}(f, [a_1, \ldots, a_{n+1}]) &= (a_1 \cdots a_n) \cdot a_{n+1}
\end{aligned}$$

`reduce` può essere anche usata con un argomento iniziale a_0; in questo caso il valore è definito semplicemente mediante

$$\mathrm{reduce}(f, [a_1, \ldots, a_n], a_0) := \mathrm{reduce}(f, [a_0, a_1, \ldots, a_n])$$

Nella genetica algebrica (che esprime le leggi di Mendel) si usa la composizione non associativa $a \cdot b := \dfrac{a+b}{2}$:

```
def f(x,y): return (x+y)/2.0

print reduce(f,[3]),
print reduce(f,[3,4]),
print reduce(f,[3,4,5]),
print reduce(f,[3,4,5,8])
# 3 3.5 4.25 6.125
```

Se, fissato x, nello schema di Horner definiamo $b \cdot a := bx + a$, possiamo riprogrammare l'algoritmo mediante `reduce`:

```
def hornerr (a,x):
  def f(u,v): return x*u+v
  return reduce(f,a,0)

a=[7,2,3,5,4]
print hornerr(a,10)
# 72354
```

Se per numeri reali $a, b > 0$ definiamo $a \cdot b := \dfrac{1}{a} + b$, otteniamo le frazioni continue, con gli argomenti invertiti nell'ordine:

```
def frazcont (v):
  def f(x,y): return y+1.0/x
  w=v[:]; w.reverse()
  return reduce(f,w)

print frazcont([2,3,1,5])
# 2.26086956522 = 52/23.0
```

filter

Questa utilissima funzione con la sintassi `filter(f,v)` estrae da una sequenza `v` tutti gli elementi `x` per cui `f(x)` è vera e restituisce come risultato la lista che consiste di tutti questi elementi. In R filtri sono realizzati mediante vettori di indici logici.

Dopo aver definito

```
def pari (x): return x%2==0
```

possiamo estrarre tutti i numeri pari da un vettore:

```
v=xrange(1,21)
print filter(pari,v)
# [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

Similmente da un elenco di parole possiamo estrarre le parole che hanno lunghezza ≥ 5:

```
v=['Roma','Ferrara','Bologna','Pisa']
print filter(lambda x: len(x)>4,v)
# ['Ferrara', 'Bologna']
```

Il crivello di Eratostene

Questo antico metodo per trovare i numeri primi $\leq n$ può essere programmato molto facilmente con Python:

```
def Eratostene (n):
  v=xrange(2,n+1); u=[]
  r=math.sqrt(n); p=2
  while p<=r:
    p=v[0]; u.append(p)
    v=filter(lambda x: x%p>0,v)
  return u+v

v=Eratostene(100); m=len(v)
for i in xrange(0,m):
  if i%10==0: print
  print v[i],

# 2 3 5 7 11 13 17 19 23 29
# 31 37 41 43 47 53 59 61 67 71
# 73 79 83 89 97
```

map semplice

f sia una funzione (o un'espressione `lambda`) di un singolo argomento, a una sequenza o un iteratore con gli elementi $a_0, \ldots, a_n$. Allora `map(f,a)` è la lista $[f(a_0), \ldots, f(a_n)]$.

```
a='ABCDabcd'
print map(ord,a)
# [65, 66, 67, 68, 97, 98, 99, 100]
```

Infatti, per un carattere `x` si ottiene con `ord(x)` il suo codice ASCII.

```
u=map(lambda x: x**2, xrange(10))
print u
# [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

L'esempio che segue esprime in poche righe la potenza di Python:

```
def vocale (x):
  return x.lower() in 'aeiou'

print map(vocale,'Crema')
# [False, False, True, False, True]
```

Usiamo `map` e `zip` (pagina 21) per costruire il grafico di una funzione:

```
def grafico (f,a):
  return zip(a,map(f,a))

a=(0,1,2,3)
print grafico(lambda x: x**4,a)
# [(0, 0), (1, 1), (2, 16), (3, 81)]
```

La funzione `str` trasforma un oggetto in una stringa e, utilizzata insieme a `join`, permette ad esempio di trasformare una lista di numeri in una stringa i cui i numeri appaiono separati da spazi:

```
a=(88,20,17,4,58)
v=map(str,a)
print v
# ['88', '20', '17', '4', '58']

print ' '.join(v)
# 88 20 17 4 58
```

Alcune regolarità del codice genetico possono essere studiate meglio se le lettere G,A,C,T vengono sostituite con 0,1,2,3:

```
def dnanumerico (x):
  x=x.lower()
  sost={'g': 0, 'a': 1, 'c': 2, 't': 3}
  return sost[x]

g='TGAATGCTAC'
print map(dnanumerico,g)
# [3, 0, 1, 1, 3, 0, 2, 3, 1, 2]
```

Cesare codificava talvolta i suoi messaggi sostituendo ogni lettera con la lettera che si trova a tre posizioni dopo la lettera originale, calcolando le posizioni in maniera ciclica. Assumiamo di avere un alfabeto di 26 lettere (senza minuscole, spazi o interpunzioni):

```
def cesare (a):
  o=ord('A')
  def codifica(x):
    n=(ord(x)-o+3)%26
    return chr(o+n)
  v=map(codifica,a)
  return ''.join(v)

a='CRASCASTRAMOVEBO'
print cesare(a)
# FUDVFDVWUDPRYHER
```

I `map` e i `lambda` possono essere annidati. Per costruire una matrice identica possiamo usare (non è questo il modo più efficiente) la seguente costruzione:

```
a=map(lambda i:
  map (lambda j:
    (i==j)+0,xrange(4)), xrange(4))
for x in a: print x
```

con output

```
[1, 0, 0, 0]
[0, 1, 0, 0]
[0, 0, 1, 0]
[0, 0, 0, 1]
```

Qui abbiamo usato che addizionando 0 a `True` o `False` otteniamo il valore 1 o 0 corrispondente.

map multivariato

$a = (a_0, \ldots, a_n)$ e $b = (b_0, \ldots, b_n)$ siano due sequenze o iteratori della stessa lunghezza ed f una funzione di 2 argomenti. Allora `map(f,u,v)` è la lista

$$[f(a_0, b_0), \ldots, f(a_n, b_n)]$$

In modo simile sono definite le espressioni `map(f,u,v,w)` per funzioni di tre argomenti ecc. Esempi:

```
def somma (*a): # Come a pagina 15.
  s=0
  for x in a: s+=x
  return s

print map(somma,
  (0,1,5,4),(2,0,3,8),(6,2,2,7))
# [8, 3, 10, 19]
```

`None`, pur non essendo una funzione, nel `map` è considerata come la funzione identica:

```
print map(None,(1,2,3))
# [1, 2, 3]

print map(None,(11,12,13),(21,22,23))
# [(11, 21), (12, 22), (13, 23)]

v=map(None,
  (11,12,13),(21,22,23),(31,32,33))
for r in v: print r
# Output:
(11, 21, 31)
(12, 22, 32)
(13, 23, 33)
```

Vediamo che `map(None,...)` è equivalente all'utilizzo di `zip` (pagina 21). Un altro esempio:

```
a=(2,3,5,0,2); b=(1,3,8,0,4)

def f(x,y): return (x+y,x-y)

for x in map(f,a,b): print x,
# (3,1) (6,0) (13,-3) (0,0) (6,-2)
```

map implicito

Il Python prevede una costruzione che permette spesso di sostituire il `map` con un `for`. Se `f` è una funzione in una variabile,

```
[f(x) for x in a]
```

è equivalente a `map(f,a)`; similmente per una funzione di due variabili

```
[f(x,y) for x,y in ((a1,b1),(a2,b2),...)]
```

corrisponde a `map(f,a,b)`. Questa forma è spesso più intuitiva e talvolta più breve:

```
a=('Roma','Pisa','Milano','Trento')
print [x[0] for x in a]
# ['R', 'P', 'M', 'T']

a=[x*x+3 for x in xrange(8)]
print a
# [3, 4, 7, 12, 19, 28, 39, 52]

a=[x+y for x,y in ((3,2),(5,6),(1,9))]
print a
# [5, 11, 10]
```

Variabili globali

Variabili *alla sinistra di assegnazioni* all'interno di una funzione e non riferite esplicitamente a un modulo sono *locali*, se non vengono definite globali con `global`. Non è invece necessario dichiarare `global` variabili esterne che vengono solo lette senza che gli venga assegnato un nuovo valore.

```
x=7

def f(): x=2
f(); print x
# 7

def g(): global x; x=2

g(); print x
# 2
```

Nell'esempio

```
u=[8]

def f(): u[0]=5

f(); print u
# [5]
```

non è necessario dichiarare `u` come variabile globale, perché viene usata solo in lettura. Infatti non viene cambiata la `u`, ma solo un valore in un indirizzo a cui `u` punta.

Quindi si ha ad esempio anche

```
u=[8]

def aumenta (u): u[0]=u[0]+1

aumenta(u); print u
# [9]
```

Invece di dichiarare una variabile come globale, la si può anche riferire esplicitamente a un modulo. Assumiamo prima che la variabile `u` appartenga a un modulo esterno, ad esempio `mat`, in cui abbia inizialmente il valore 7. Allora possiamo procedere come nel seguente esempio:

```
import mat

x=7

def f ():
  mat.x+=1

f(); print x
# 8
```

Se la variabile appartiene invece allo stesso modulo come la funzione che la dovrebbe modificare, possiamo usare `sys.modules` per identificare il modulo:

```
x=7

def f ():
  sys.modules[__name__].x+=1

f(); print x
# 8
```

A questo scopo possiamo anche importare il modulo stesso in cui ci troviamo e usare la tecnica del penultimo esempio.

Si dovrebbe cercare di utilizzare variabili globali solo quando ciò si rivela veramente necessario e, in tal caso, di non definirle nel file che contiene il programma principale, ma in moduli appositi.

---

zip

Questa funzione utilissima corrisponde essenzialmente alla formazione della trasposta di una matrice. Esempio:

```
a=[1,2,3]; b=[11,12,13]
c=[21,22,23]; d=[31,32,33]
for x in zip(a,b,c,d): print x
# Output:

(1, 11, 21, 31)
(2, 12, 22, 32)
(3, 13, 23, 33)
```

Il risultato di `zip` è sempre una lista i cui elementi sono tuple. Quando gli argomenti non sono tutti della stessa lunghezza, viene usata la lunghezza minima, come in

```
a=[1,2,3,4]; b=[11,12]
c=[21,22,23,24]
for x in zip(a,b,c): print x
# Output:

(1, 11, 21)
(2, 12, 22)
```

Il tipo dei dati non ha importanza:

```
a=[0,1,2,3]; b=['a',7,'geo',[9,10]]
c=[11,12,13,14]
d=['A','B','C',['E','F']]
for x in zip(a,b,c,d): print x
# Output:

(0, 'a', 11, 'A')
(1, 7, 12, 'B')
(2, 'geo', 13, 'C')
(3, [9, 10], 14, ['E', 'F'])
```

Uso di `zip` con `for x,y`:

```
nomi=('Verdi','Rossi','Bianchi')
stipendi=(2000,1800,2700)
for x,y in zip(nomi,stipendi):
  print x,y
# Output:

Verdi 2000
Rossi 1800
Bianchi 2700
```

---

eval

Se `E` è una stringa che contiene un'espressione valida di Python (ma non ad esempio un'assegnazione), `eval(E)` è il *valore* di questa espressione. Esempi:

```
u=4
print eval('u*u+7')
# 23

def f (x): return x+5

print eval('f(2)+17') # 24

print eval('f(u+1)') # 10

def sommaf (F,x,y):
  f=eval(F); return f(x)+f(y)

def cubo (x): return x*x*x

print sommaf('cubo',2,5)
# 133
```

---

exec

Se la stringa `a` contiene istruzioni di Python valide, queste vengono eseguite con `exec(a)`. `exec`, a differenza da `eval`, non restituisce un risultato. Esempi:

```
a='x=8; y=6; print x+y'

exec(a)
# 14
```

`exec` è utilizzato naturalmente soprattutto per eseguire comandi che vengono costruiti durante l'esecuzione di un programma; usato con raziocinio permette metodi avanzati e, se si vuole, lo sviluppo di meccanismi di intelligenza artificiale.

Consideriamo le istruzioni

```
x=0; comando='x=17'

def f ():
  global x
  exec comando

f(); print x
# 0
```

Come mai - nonostante che `x` in `f` sia `global`? La ragione è questa: `global` è una (anzi l'unica) direttiva per il compilatore e riguarda solo l'esecuzione in cui viene chiamata. `exec` fa ripartire il compilatore e quindi bisogna ripetere il `global` nell'espressione a cui si applica `exec`. Dovremmo quindi scrivere:

```
x=0; comando='global x; x=17'

def f ():
  exec comando

f(); print x
# 17
```

---

execfile

`nomef` sia il nome di un file che contenga istruzioni di Python. Allora con `execfile(nomef)` queste istruzioni vengono eseguite. La differenza pratica principale con `import` è che i nomi in `nomef` valgono come se fossero nomi del file chiamante, mentre con `import` (l'istruzione che viene utilizzata per importare uno o più moduli) bisogna aggiungere il prefisso corrispondente al modulo definito dal file. Essenzialmente l'effetto di `execfile` è come se avessimo letto il contenuto del file in una stringa `a` ed effettuato il comando `exec(a)`.

Talvolta `execfile` viene usato per leggere parametri di configurazione da un file. Questi parametri possono poi essere utilizzati come variabili globali dal programma.

sort

Sappiamo dall'elenco a pagina 17 che il metodo `sort` permette di ordinare una lista. Per una lista `a` la sintassi completa è

```
a.sort(cmp=None,key=None,reverse=False)
```

Se non reimpostiamo il parametro `cmp`, il Python utilizza una funzione di confronto naturale, essenzialmente l'ordine alfabetico inteso in senso generale, ad esempio nel caso di liste annidate; infatti in Python praticamente tutti gli oggetti sono confrontabili. `key` è una funzione che viene applicata ad ogni elemento della lista prima dell'ordinamento; lasciando `key=None`, gli elementi vengono ordinati così come sono. `reverse=True` implica un ordinamento in senso decrescente.

Come esempio dell'uso del parametro `key` assumiamo che vogliamo ordinare una lista di numeri tenendo conto dei valori assoluti:

```
a=[1,3,-5,0,-3,7,-10,3]
a.sort(key=abs)
print a
# [0, 1, 3, -3, 3, -5, 7, -10]
```

Similmente possiamo usare come elementi di confronto i resti modulo 100; ciò è equivalente naturalmente a considerare solo le ultime due cifre di un numero naturale:

```
a=[3454,819,99,4545,716,310]
a.sort(key=lambda x: x%100)
print a
# [310, 716, 819, 4545, 3454, 99]
```

Utilizzando `key=string.lower` possiamo ordinare una lista di stringhe alfabeticamente, prescindendo dalla distinzione tra minuscole e maiuscole.
Impostando `cmp=f` è anche possibile utilizzare una propria funzione di confronto `f`. Questa deve essere una funzione di due variabili `x`,`y` che restituisce -1 (o un numero negativo) se consideriamo `x` minore di `y`, 0 se i due numeri sono considerati uguali, 1 (o un numero positivo) se consideriamo `x` maggiore di `y`. La funzione dovrebbe soddisfare la condizione `f(x,x)==0` e indurre una relazione transitiva.

Assumiamo ad esempio che consideriamo una lista `a` minore di una lista `b` se `a` ha meno elementi di `b`:

```
def conflun (a,b):
  if len(a)<len(b): return -1
  if len(a)==len(b): return 0
  return 1

v=[[6,2,0],[9,1],[4,5,8,3],[3,1,7]]
v.sort(cmp=conflun)
print v
# [[9,1],[6,2,0],[3,1,7],[4,5,8,3]]
```

Naturalmente in questo caso lo stesso effetto l'avremmo potuto ottenere con

```
v.sort(key=len)
```

Infatti molto spesso nella pratica (e teoricamente sempre) è più conveniente impostare il parametro `key` piuttosto di impostare `cmp`.

Dizionari

Abbiamo incontrato esempi di dizionari alle pagine 15 e 20. Come voci (o chiavi) si possono usare oggetti non mutabili (ad esempio numeri, stringhe oppure tuple i cui elementi sono anch'essi non mutabili). Un dizionario, una volta definito, può essere successivamente modificato:

```
stipendi = {('Rossi','Trento') : 4000,
('Rossi','Roma') : 8000,
('Gardini','Pisa') : 3400}

stipendi[('Neri','Roma')]=4500

voci=list(stipendi.keys())
voci.sort()
for x in voci:
  print '%-s %7s %d' \
    %(x[0],x[1],stipendi[x])

# Output:

Gardini Pisa    3400
Neri    Roma    4500
Rossi   Roma    8000
Rossi   Trento  4000
```

Si osservi il modo in cui le voci sono state ordinate alfabeticamente, prima rispetto alla prima colonna, poi, in caso di omonimia, rispetto alla seconda colonna.

Se `d` è un dizionario, `d.keys()` è una lista che contiene le chiavi di `d`. L'ordine in cui queste chiavi appaiono nella lista non è prevedibile per cui spesso, ad esempio nell'output, essa viene ordinata come abbiamo fatto in alcuni degli esempi precedenti.

`d.has_key(x)` indica (mediante un valore di verità) se `d` possiede una voce `x`. Con `del d[x]` la voce `x` viene cancellata dal dizionario. Questa istruzione provoca un errore, se la voce `x` non esiste.

`diz.items()` è la lista delle coppie di cui consiste il dizionario.

```
diz = {'a' : 1, 'b' : 2, 'c' : 3,
  'd' : 4, 'x' : 5, 'y' : 6, 'z' : 7}

print diz.items()
# Output che riscriviamo su
# piu' righe:

[('a', 1), ('c', 3), ('b', 2),
('d', 4), ('y', 6), ('x', 5),
('z', 7)]
```

if ... elif ... else

Le istruzioni condizionali in Python vengono utilizzate con la sintassi

```
if A: alfa()

if A: alfa()
else: beta()

if A:
  if B: alfa()
  else: beta()
else: gamma()
```

L'indentazione determina in ogni caso a quale `if` un `else` si riferisce. Non dimenticare i doppi punti.

Spesso si incontrano diramazioni della forma

```
if A: alfa()
else:
  if B: beta()
  else:
    if C: gamma()
    else: delta()
```

In questi casi i doppi punti e la necessità delle indentazioni rendono la composizione del programma difficoltosa; è prevista perciò in Python l'abbreviazione `elif` per un `else ... if` come nell'ultimo esempio che può essere riscritto in modo più semplice:

```
if A: alfa()
elif B: beta()
elif C: gamma()
else: delta()
```

Esempio:

```
def segno (x):
  if x>0: return 1
  elif x==0: return 0
  return -1
```

Purtroppo il Python non prevede la costruzione `switch ... case` del C. Con un po' di attenzione la si può comunque emulare con l'impiego di una serie di `elif` oppure, come nel seguente esempio, mediante l'impiego adeguato di un dizionario e di `exec`:

```
operazioni = {'Roma' : 'print "Lazio"',
  'Ferrara' : 'print "Romagna"',
  'Cremona' : 'x=5; print x*x'}
for x in ['Roma','Ferrara','Cremona']:
  exec(operazioni[x])
```

Esercizi per gli scritti

16. Definire prima una funzione `tangradi` e poi una funzione che calcola x ed y da α, β e c come nel primo esempio a pagina 8. Il risultato deve essere la lista $[x, y]$.

17. Definire una funzione P con gli argomenti a, b e γ per l'ultimo esempio a pagina 8. Assumere che il sistema di coordinate (cartesiano) sia scelto in modo tale che C si trovi nell'origine ed A sull'asse delle x. Il risultato è la lista che contiene le coordinate di P rispetto a questo sistema di coordinate.

Fare un disegno e procedere con molta cura. Se necessario inserire `float` nelle divisioni. γ è dato in gradi, mentre `math.acos` fornisce l'angolo in radianti. Misurare il triangolo dell'esempio per controllare il risultato.

18. Definire una funzione che utilizza la formula di Erone per calcolare l'area di un triangolo.

Usare le funzioni

```
def cosgradi (g):
  return math.cos(g*math.pi/180)
def singradi (g):
  return math.sin(g*math.pi/180)
```

V. L'ALGORITMO EUCLIDEO

Multipli e divisori di un numero intero

Situazione 23.1. Tutti i numeri considerati $a, b, c, \ldots$ sono interi, cioè elementi di $\mathbb{Z}$.

Usiamo l'abbreviazione $\mathbb{Z}d := \{nd \mid n \in \mathbb{Z}\}$.

Definizione 23.2. Diciamo che a è un *multiplo* di d, se $a \in \mathbb{Z}d$, cioè se esiste $n \in \mathbb{Z}$ tale che $a = nd$. In questo caso diciamo anche che d *divide* a o che d è un *divisore* di a e scriviamo $d|a$. Quindi

$$d \text{ divide } a \iff a \text{ è multiplo di } d \quad (*)$$

In formule: $d|a \iff a \in \mathbb{Z}d$.

Nota 23.3. Questo è un esempio di *dualità*, cioè lo stesso concetto ha due aspetti che possono essere formulati in modo diverso benché siano tra di loro del tutto equivalenti. La relazione (*) significa che la teoria T_1 che per ogni $a \in \mathbb{Z}$ studia l'insieme dei suoi divisori e la teoria T_2 che per ogni $d \in \mathbb{Z}$ studia l'insieme dei suoi multipli sono soltanto formulazioni diverse della stessa teoria T.

In questi casi è spesso molto vantaggioso lavorare con entrambe le teorie senza cercare di limitarsi a una delle due, anche se ciò teoricamente sarebbe possibile. Quindi anche noi useremo sia il linguaggio della divisibilità e dei divisori che quello dei multipli che a sua volta, come vedremo brevemente quando parleremo dei sottogruppi di $\mathbb{Z}$, porta alla teoria degli *ideali*.

Osservazione 23.4. *Si dimostra facilmente che valgono i seguenti enunciati:*

(1) $d|a \iff d|(-a) \iff -d|a$.
(2) $d|a \iff d||a|$.
(3) $1|a$ *per ogni* a.
(4) $a|1 \iff a \in \{1, -1\}$.
(5) $a|0$ *per ogni* a.
(6) $d|a \Longrightarrow d|ka$ *per ogni* k.
(7) $d|a$ *e* $d|b \Longrightarrow d|(a+b)$ *e* $d|(a-b)$.

Definizione 23.5. Per $(a, b) \neq (0, 0)$ il *massimo comune divisore* di a e b, denotato con $\mathrm{mcd}(a, b)$, è il più grande $d \in \mathbb{N}$ che è un comune divisore di a e b, cioè tale che $d|a$ e $d|b$. Poniamo invece $\mathrm{mcd}(0, 0) := 0$. In questo modo $\mathrm{mcd}(a, b)$ è definito per ogni coppia (a, b) di numeri interi.

Osservazione 23.6. Perché esiste $\mathrm{mcd}(a, b)$? Per $(a, b) = (0, 0)$ esso è uguale a 0 per definizione. Assumiamo che $(a, b) \neq (0, 0)$. Adesso $1|a$ e $1|b$ e se $d|a$ e $d|b$ e ad esempio $a \neq 0$, allora $d \leq |a|$, per cui vediamo che esiste solo un numero finito (al massimo $|a|$) di divisori comuni ≥ 1, tra cui uno ed uno solo deve essere il più grande. $\mathrm{mcd}(a, b)$ è quindi univocamente determinato e uguale a 0 se e solo se $a = b = 0$. Abbiamo visto che $d|a \iff -d|a$, per cui possiamo senza perdita in generalità assumere che $d \in \mathbb{N}$.

Definizione 23.7. a e b si chiamano *relativamente primi*, se il loro massimo comune divisore è uguale a 1.

L'algoritmo euclideo

Questo algoritmo familiare a tutti e apparentemente a livello solo scolastico, è uno dei più importanti della matematica ed ha numerose applicazioni: in problemi pratici (ad esempio nella grafica al calcolatore), in molti campi avanzati della matematica (teoria dei numeri e analisi complessa), nell'informatica teorica. L'algoritmo euclideo si basa sulla seguente osservazione (*lemma di Euclide*):

Lemma 23.8. *Siano* a, b, c, q, d *numeri interi e* $a = qb + c$. *Allora*

$$(d|a \text{ e } d|b) \iff (d|b \text{ e } d|c)$$

Quindi i comuni divisori di a *e* b *sono esattamente i comuni divisori di* b *e* c. *In particolare le due coppie di numeri devono avere lo stesso massimo comune divisore:* $\mathrm{mcd}(a, b) = \mathrm{mcd}(b, c)$.

Dimostrazione. Se $d|a$ e $d|b$, cioè $dx = a$ e $dy = b$ per qualche x, y, allora $c = a - qb = dx - qdy = d(x - qy)$ e vediamo che $d|c$.

E viceversa.

Esempio 23.9. Calcoliamo $d := \mathrm{mcd}(7464, 3580)$:

$$\begin{aligned}
7464 &= 2{\cdot}3580 + 304 &&\Longrightarrow d = \mathrm{mcd}(3580, 304)\\
3580 &= 11{\cdot}304 + 236 &&\Longrightarrow d = \mathrm{mcd}(304, 236)\\
304 &= 1{\cdot}236 + 68 &&\Longrightarrow d = \mathrm{mcd}(236, 68)\\
236 &= 3{\cdot}68 + 32 &&\Longrightarrow d = \mathrm{mcd}(68, 32)\\
68 &= 2{\cdot}32 + 4 &&\Longrightarrow d = \mathrm{mcd}(32, 4)\\
32 &= 8{\cdot}4 + 0 &&\Longrightarrow d = \mathrm{mcd}(4, 0) = 4
\end{aligned}$$

Si vede che il massimo comune divisore è l'ultimo resto diverso da 0 nell'algoritmo euclideo. Che l'algoritmo può sempre essere eseguito e che termina sempre verrà dimostrato nella prop. 24.4.

La traduzione in Python dell'algoritmo euclideo è molto semplice (dobbiamo però prima convertire i numeri negativi in positivi):

```
def mcd (a,b):
  if a<0: a=-a
  if b<0: b=-b
  while b: a,b=b,a%b
  return a
```

Altrettanto semplice è la versione ricorsiva:

```
def mcd (a,b):
  if a<0: a=-a
  if b<0: b=-b
  if b==0: return a
  return mcd(b,a%b)
```

dove usiamo la relazione

$$\mathrm{mcd}(a, b) = \begin{cases} a & \text{se } b = 0 \\ \mathrm{mcd}(b, a\%b) & \text{se } b > 0 \end{cases}$$

Osservazione 23.10. Sia $d = \mathrm{mcd}(a, b)$. Si può dimostrare che esistono sempre $x, y \in \mathbb{Z}$ tali che $d = ax + by$, seguendo ad esempio il seguente ragionamento ricorsivo. Se abbiamo

$$a = \alpha b + c \quad \text{e} \quad d = bx' + cy'$$

allora

$$d = bx' + (a - \alpha b)y' = ay' + b(x' - \alpha y')$$

per cui $d = ax + by$ con $x = y'$ ed $y = x' - \alpha y'$.

In Python l'algoritmo euclideo esteso restituisce la tupla (d, x, y):

```
def mcde (a,b):
  if a<0: a=-a
  if b<0: b=-b
  if b==0: return a,1,0
  d,x,y=mcde(b,a%b); alfa=a/b
  return d,y,x-alfa*y
```

Daremo fra poco (nel teorema 25.8) un'altra dimostrazione di questo importante enunciato, utilizzando la teoria degli ideali.

Esempio 23.11. Nell'esempio 23.9 abbiamo visto che $d := \mathrm{mcd}(7464, 3580) = 4$. Per l'oss. 23.10 deve quindi esistere una rappresentazione $d = 7464x + 3580y$ con $x, y \in \mathbb{Z}$. Seguiamo il procedimento dell'oss. 23.10; dobbiamo cominciare con la penultima riga dell'algoritmo euclideo:

$$\begin{aligned}
4 &= 68 - 2 \cdot 32 = 68 - 2 \cdot (236 - 3 \cdot 68)\\
&= 7 \cdot 68 - 2 \cdot 236 = 7 \cdot (304 - 1 \cdot 236) - 2 \cdot 236\\
&= 7 \cdot 304 - 9 \cdot 236 = 7 \cdot 304 - 9 \cdot (3580 - 11 \cdot 304)\\
&= 106 \cdot 304 - 9 \cdot 3580 = 106 \cdot (7464 - 2 \cdot 3580) - 9 \cdot 3580\\
&= 106 \cdot 7464 - 221 \cdot 3580
\end{aligned}$$

Questa è la rappresentazione cercata con $x = 106$ ed $y = -221$. Si noti che x ed y non sono univocamente determinati; infatti se $ax_0 + by_0 = d$, allora $a(x_0 - nb) + b(y_0 + na) = d$ per ogni $n \in \mathbb{Z}$.

L'algoritmo può essere reso più sistematico e più trasparente e fa parte di un ramo molto importante della teoria dei numeri, la teoria delle *frazioni continue*.

Frazioni continue

Nota 24.1. Accenniamo solo brevemente al legame tra l'algoritmo euclideo (o le frazioni continue) e la teoria dei linguaggi formali, cioè lo studio delle parole su un alfabeto finito. Consideriamo ancora i coefficienti dello schema di calcolo nell'esempio 23.9: $[2, 11, 1, 3, 2, 8]$.

Lavorando con un alfabeto a due lettere (distinte) A e B, associamo a questa sequenza la parola

$$\underbrace{AA}_{2}\underbrace{BBBBBBBBBBB}_{11}\underbrace{A}_{1}\underbrace{BBB}_{3}\underbrace{AA}_{2}\underbrace{BBBBBBB}_{7=8-1}$$

Si noti che alla fine abbiamo diminuito l'ultimo numero della sequenza di 1. Altri esempi per questa corrispondenza:

$$[3, 1, 2, 4] \ldots AAABAABBB$$
$$[0, 2, 1, 2, 3] \ldots BBABBAA$$
$$[7, 3, 6, 2, 5, 4] \ldots AAAAAAABBBAAAAAABBAAAAABBB$$

Dalla sequenza si può ricostruire il quoziente a/b dei due numeri dei quali mediante l'algoritmo euclideo abbiamo calcolato il massimo comune divisore:

		2	11	1	3	2	8
0	1	2	23	25	98	221	1866
1	0	1	11	12	47	106	895

Osserviamo che $1866 = 7464/4, 895 = 3580/4$ e quindi

$$\frac{7464}{3580} = \frac{1866}{895}$$

L'ultima colonna fornisce perciò la frazione continua ridotta dei due numeri a e b; infatti si può dimostrare che per ogni colonna i due numeri che in essa appaiono sono relativamente primi; cfr. nota 24.3 e teorema 25.10.

Nella penultima colonna appaiono invece i coefficienti 106 e -221 nella rappresentazione $ax + by = d$ che devono essere presi con i segni giusti. Tutto ciò vale in generale e rientra, come osservato, nella teoria delle frazioni continue.

Proviamo con $[3, 1, 2, 4]$:

		3	1	2	4
0	1	3	4	11	48
1	0	1	1	3	13
		+	−	+	−

Verifichiamo che i numeri dati corrispondono all'algoritmo euclideo per 48 e 13:

$$48 = 3 \cdot 13 + 9$$
$$13 = 1 \cdot 9 + 4$$
$$9 = 2 \cdot 4 + 1$$
$$4 = 4 \cdot 1$$

Vediamo che è così. Inoltre possiamo spreare di poter usare 11 e 3 per la rappresentazione del $\mathrm{mcd}(48, 13) = 1$; per il segno giusto usiamo i simboli $+$ e $-$ che abbiamo aggiunto sotto la tabella:

$$3 \cdot 48 - 11 \cdot 13 = 144 - 143 = 1$$

Perciò $1 = 48x + 13y$ con $x = 3$ e $y = -11$.

Perché si parla di *frazioni continue*? Perché

$$\frac{48}{13} = 3 + \cfrac{1}{1 + \cfrac{1}{2 + \cfrac{1}{4}}}$$

come si verifica facilmente. Possiamo identificare questa frazione continua o il numero razionale $48/13$ con la parola $AAABAABBB$.

$SL(2, \mathbb{N})$

Definizione 24.2. $SL(2, \mathbb{N}) := \{A \in \mathbb{N}_2^2 \mid \det(A) = 1\}$ sia l'insieme delle matrici 2×2 con coefficienti naturali e determinante 1.

Nota 24.3. Mediante la nota 24.1 è molto facile trovare elementi di $SL(2, \mathbb{N})$. Infatti si può dimostrare che nelle tabelle che abbiamo costruito per ogni coppia di colonne adiacenti il determinante della matrice 2×2 che si può formare con gli elementi di quelle colonne è uguale a ± 1 (con segno alternante).

Nella prima tabella ad esempio

$$\begin{vmatrix} 0 & 1 \\ 1 & 0 \end{vmatrix} = \begin{vmatrix} 2 & 23 \\ 1 & 11 \end{vmatrix} = \begin{vmatrix} 25 & 98 \\ 12 & 47 \end{vmatrix} = \begin{vmatrix} 221 & 1866 \\ 106 & 895 \end{vmatrix} = -1$$

$$\begin{vmatrix} 1 & 2 \\ 0 & 1 \end{vmatrix} = \begin{vmatrix} 23 & 25 \\ 11 & 12 \end{vmatrix} = \begin{vmatrix} 98 & 221 \\ 47 & 106 \end{vmatrix} = 1$$

Per trovare un elemento di $SL(2, \mathbb{N})$ è quindi sufficiente indicare una lista $[a_1, \ldots, a_k]$ di numeri naturali $\neq 0$. Si può anche dimostrare che ogni matrice in $SL(2, \mathbb{N})$ può essere ottenuta in questo modo.

Divisione con resto

Proposizione 24.4. *Siano dati $a, b \in \mathbb{R}$ con $b \neq 0$. Allora esistono univocamente determinati $q \in \mathbb{Z}$ e $r \in \mathbb{R}$ tali che*

$$a = qb + r \text{ e } 0 \leq r < |b|.$$

La penultima relazione implica che, se $a, b \in \mathbb{Z}$, anche $r \in \mathbb{Z}$ (e quindi $r \in \mathbb{N}$).

Dimostrazione. Sia $b > 0$. Allora

$$\mathbb{R} = \ldots \sqcup [-2b, -b) \sqcup [-b, 0) \sqcup [0, b) \sqcup [b, 2b) \sqcup [2b, 3b) \sqcup \ldots,$$

cioè

$$\mathbb{R} = \bigsqcup_{q \in \mathbb{Z}} [qb, qb + b) = \bigsqcup_{q \in \mathbb{Z}} [qb, qb + |b|)$$

Se invece $b < 0$, allora

$$\mathbb{R} = \cdots \sqcup [2b, b) \sqcup [b, 0) \sqcup [0, -b) \sqcup [-b, -2b) \sqcup [-2b, -3b) \sqcup \ldots,$$

cioè

$$\mathbb{R} = \bigsqcup_{q \in \mathbb{Z}} [qb, qb - b) = \bigsqcup_{q \in \mathbb{Z}} [qb, qb + |b|)$$

Quindi in entrambi i casi

$$\mathbb{R} = \bigsqcup_{q \in \mathbb{Z}} [qb, qb + |b|)$$

Le unioni sono disgiunte; questo implica che per ogni $a \in \mathbb{R}$ esiste esattamente un $q \in \mathbb{Z}$ per il quale $a \in [qb, qb + |b|)$.

E questo è esattamente l'enunciato della proposizione: trovato q, possiamo porre r uguale a $a - qb$.

Osservazione 24.5. Nella prop. 24.4 r si chiama il *resto* nella divisione di a per b o il resto di a *modulo* b. Nella matematica si scrive spesso $r = a \mod b$. In Python (e in C) il resto viene calcolato dall'espressione *a%b*, che dà però risultati corretti solo per $a \in \mathbb{N}$ e $b \in \mathbb{N} + 1$. Per questa ragione nelle funzioni `mcd` e `mcde` a pagina 23 abbiamo prima convertito eventuali argomenti negativi in positivi.

$\mathbb{N}$ è un insieme ben ordinato

Lemma 24.6. *Ogni insieme non vuoto di numeri naturali possiede un elemento più piccolo.*

Dimostrazione. Sia $A \subset \mathbb{N}$ ed $A \neq \emptyset$. Assumiamo, per assurdo, che A non abbia un elemento più piccolo. Dimostriamo che $A = \emptyset$ (una contraddizione all'ipotesi), dimostrando per induzione su n che $n \notin A$ per ogni $n \in \mathbb{N}$.

$n = 0$: Se fosse $0 \in A$, 0 sarebbe l'elemento più piccolo di A.

$n \to n+1$: Sia $n+1 \in A$. Per l'ipotesi di induzione k non appartiene ad A per ogni $k \leq n$. Ciò implicherebbe che $n + 1$ è l'elemento più piccolo di A.

Sottogruppi di $\mathbb{Z}$

Definizione 25.1. Un *sottogruppo* di $\mathbb{Z}$ è un sottoinsieme $H \subset \mathbb{Z}$ con le seguenti proprietà:

(1) $0 \in H$.
(2) $a, b \in H \Longrightarrow a - b \in H$.

Queste condizioni implicano che per $a, b \in H$ anche $-b \in H$, perché $-b = 0 - b$, ed $a + b \in H$, perché $a + b = a - (-b)$.

I sottogruppi di $\mathbb{Z}$ e diversi da $\mathbb{Z}$ sono anche detti *ideali* di $\mathbb{Z}$. In strutture algebriche più generali invece ideali e sottogruppi propri in genere non sono più la stessa cosa.

Osservazione 25.2. Per ogni $d \in \mathbb{Z}$ l'insieme $\mathbb{Z}d$ dei multipli di d è evidentemente un sottogruppo di $\mathbb{Z}$. Vogliamo adesso dimostrare che non ci sono altri sottogruppi di $\mathbb{Z}$.

Osservazione 25.3. $a|b \iff \mathbb{Z}b \subset \mathbb{Z}a$.

Infatti $a|b \iff$ ogni multiplo di b è anche multiplo di a.

Nonostante che si tratti di una quasi ovvia riformulazione del concetto di divisibilità, questa osservazione trasforma un problema aritmetico in un problema insiemistico.

Osservazione 25.4. $\mathbb{Z}a = \mathbb{Z}b \iff a = \pm b$.

Dimostrazione. $\Longrightarrow$: Per $a = 0$ si ha $b \in \mathbb{Z}a = 0$, quindi $b = 0 = a$.
Sia quindi $a \neq 0$. L'ipotesi implica $a = bx, b = ay$ per qualche $x, y \in \mathbb{Z}$, per cui $a = bx = axy$, quindi $a(1 - xy) = 0$ ed, essendo $a \neq 0$, si ha $xy = 1$, perciò $x = \pm 1$.

$\Longleftarrow$: Chiaro.

Teorema 25.5. *Ogni sottogruppo H di $\mathbb{Z}$ è della forma $H = \mathbb{Z}d$ per un numero naturale d che è univocamente determinato.*
Infatti, se $H \neq \{0\}$, allora

$$d = \min\{a > 0 \mid a \in H\}$$

mentre naturalmente d deve essere uguale a 0 se $H = \{0\}$.

Dimostrazione. (1) *Unicità:* Per l'osservazione 25.4 oltre a d solo $-d$ può andar bene. Ma se $d > 0$, allora $-d \notin \mathbb{N}$.

(2) *Esistenza:* Sia $H \neq \{0\}$. Allora H possiede un elemento $a \neq 0$. Se $a < 0$, anche $-a \in H$ e vediamo che possiamo sempre trovare un elemento $a > 0$ con $a \in H$.
Sia $A := \{a \in H \mid a > 0\}$. Come abbiamo appena osservato, $A \neq \emptyset$, e dal lemma 24.6 segue che A possiede un minimo che chiamiamo d. Dimostriamo che $H = \mathbb{Z}d$.
In primo luogo $d \in H$, perciò $d, 2d = d + d, 3d = d + d + d, \ldots, -d, -2d = -(d + d), \ldots \in H$ perché, per ipotesi, H è un sottogruppo di $\mathbb{Z}$. In altre parole $\mathbb{Z}d \subset H$.

Dobbiamo ancora dimostrare che $H \subset \mathbb{Z}d$, cioè che ogni elemento di H è un multiplo del più piccolo elemento positivo di H.
Sia $a \in H$. Per la prop. 24.4 abbiamo $a = qd + r$ con $0 \leq r < d$, per cui $r = a - qd \in H$. Ma $0 \leq r < d$, quindi per la minimalità di d vediamo che r deve essere uguale a 0. Ciò significa $a = qd \in \mathbb{Z}d$.

Osservazione 25.6. *H e K siano sottogruppi di $\mathbb{Z}$. Allora anche*

$$H + K := \{a + b \mid a \in H, b \in K\}$$

è un sottogruppo di $\mathbb{Z}$.

Dimostrazione. (1) $0 = 0 + 0 \in H + K$.
(2) Siano $a = h + k$ e $b = h' + k'$ con $h, h' \in H$ e $k, k' \in K$. Allora

$$a - b = h + k - (h' + k') = \underbrace{h - h'}_{\in H} + \underbrace{k - k'}_{\in K} \in H + K$$

Teorema 25.7. *Dati a e b, esiste un unico numero naturale d tale che $\mathbb{Z}a + \mathbb{Z}b = \mathbb{Z}d$.*

Dimostrazione. Per l'osservazione 25.6 $\mathbb{Z}a + \mathbb{Z}b$ è un sottogruppo di $\mathbb{Z}$. L'enunciato segue dal teorema 25.5.

Teorema 25.8. *Dati a e b, sia d l'unico numero naturale d per il quale $\mathbb{Z}a + \mathbb{Z}b = \mathbb{Z}d$. Allora $d = \mathrm{mcd}(a, b)$.*
Infatti d ha le seguenti proprietà (più forti della condizione richiesta nella def. 23.5):

(1) $d|a$ e $d|b$.
(2) Se f è un altro divisore comune di a e b, allora $f|d$.

Dimostrazione. (1) $\mathbb{Z}a \subset \mathbb{Z}a + \mathbb{Z}b = \mathbb{Z}d$, per cui dall'oss. 25.3 segue che $d|a$. Nello stesso modo si vede che $d|b$.

(2) Ancora per l'oss. 25.3 abbiamo $\mathbb{Z}a \subset \mathbb{Z}f$ e $\mathbb{Z}b \subset \mathbb{Z}f$, per cui

$$\mathbb{Z}d = \mathbb{Z}a + \mathbb{Z}b \subset \mathbb{Z}f + \mathbb{Z}f = \mathbb{Z}f$$

e quindi $f|d$.
Siccome nel punto (2) necessariamente $f \leq d$, vediamo che d è veramente il massimo comune divisore di a e b nel senso della definizione 23.5.

Teorema 25.9. *Sia $d = \mathrm{mcd}(a, b)$. Allora esistono $x, y \in \mathbb{Z}$ tali che $d = ax + by$.*

Dimostrazione. Ciò è una conseguenza immediata del teorema precedente.

Teorema 25.10. $\mathrm{mcd}(a, b) = 1$ *se e solo se esistono $x, y \in \mathbb{Z}$ tali che $ax + by = 1$.*

Dimostrazione. $\Longrightarrow$: Teorema 25.9.

$\Longleftarrow$: Sia $d = \mathrm{mcd}(a, b)$. Se $ax + by = 1$, allora $1 \in \mathbb{Z}a + \mathbb{Z}b = \mathbb{Z}d$, per cui $d|1$ e quindi $d = \pm 1$. Ma $d \in \mathbb{N}$, quindi $d = 1$.

Equazioni diofantee lineari

Definizione 25.11. Un'*equazione diofantea lineare* in due variabili è un'equazione $ax + by = h$ con $a, b, h \in \mathbb{Z}$. Si cercano numeri interi x, y che soddisfano l'equazione.

Teorema 25.12. *L'equazione diofantea $ax + by = h$ ammette soluzione se e solo se $\mathrm{mcd}(a, b)|h$.*

Dimostrazione. Ciò segue direttamente dal teorema 25.8!

Nota 25.13. Sia $d := \mathrm{mcd}(a, b)$. Per trovare le soluzioni di un'equazione diofantea $ax + by = h$ in cui (necessariamente) $d|h$ si trova prima una soluzione (u, v) dell'equazione diofantea $ax + by = d$ con il metodo della nota 24.1.
A questo punto (x_0, y_0) è una soluzione dell'equazione $ax + by = h$, se poniamo $x_0 := uh/d, y_0 := vh/d$. Si può dimostrare (non è difficile) che tutte le soluzioni (intere) sono esattamente i punti

$$x = x_0 - nb/d$$
$$y = y_0 + na/d$$

con $n \in \mathbb{Z}$. Si osservi che a/d e b/d sono sempre interi, perché $d|a$ e $d|b$.

Osservazione 25.14. A pagina 13 per un vettore reale $z = (a, b)$ abbiamo definito il *vettore magico* $z^* = (-b, a)$. Esso è ortogonale ad (a, b) ed anche alla retta reale $ax + by = h$ (per ogni $h \in \mathbb{R}$). Vediamo quindi che, una volta trovata una soluzione (x_0, y_0) di un'equazione diofantea $ax + by = h$ (adesso per $a, b, h \in \mathbb{Z}$), le altre si ottengono aggiungendo a questa soluzione un multiplo del vettore magico di $(a/d, b/d)$.

Esercizi per gli scritti

19. $\mathrm{mcd}(4635, 1250)$.

20. $\mathrm{mcd}(71356, 18444)$.

21. Calcolare la parola che corrisponde al numero razionale $77/30$. Attenzione alla fine della parola!

22. Calcolare il numero razionale a/b che corrisponde alla parola $AABBBBAABABBAAA$.

Trovare tutte le soluzioni delle seguenti equazioni diofantee. Disegnare in ogni caso la retta reale corrispondente all'equazione e alcuni dei punti interi su di essa.

23. $6x + 13y = 1$.

24. $6x + 13y = 10$.

25. $45x + 20y = 10$.

VI. ALGORITMI ELEMENTARI

L'algoritmo del contadino russo

Esiste un algoritmo leggendario del contadino russo per la moltiplicazione di due numeri, uno dei quali deve essere un numero naturale. Nella formulazione matematica ricorsiva questo algoritmo si presenta nel modo seguente. Sia f la funzione di due variabili definita da $f(x,n) := nx$. Allora

$$f(x,n) = \begin{cases} 0 & \text{se } n = 0 \\ f(2x, n/2) & \text{se } n \text{ è pari } \neq 0 \\ x + f(x, n-1) & \text{se } n \text{ è dispari} \end{cases}$$

In Python ciò corrisponde alla funzione

```
# Moltiplicazione russa.
def f (x,n):
  if n==0: return 0
  if n%2==0: return f(x+x,n/2)
  return x+f(x,n-1)
```

Se questa funzione la inseriamo nel file *russo.py*, nel file *alfa* che funge da programma principale possiamo scrivere

```
import russo

x=10; n=87
print russo.f(x,n)
# Output: 870
```

Naturalmente il prodotto nx die due numeri in Python lo otteniamo più semplicemente con `n*x`. L'algoritmo può però essere utile in un contesto di calcolo (in un $\mathbb{N}$-*modulo*, come si dice in algebra, ad esempio in un *gruppo abeliano*) per il quale il Python non fornisce direttamente una funzione per la moltiplicazione con fattori interi. Lo stesso vale per il calcolo di potenze x^n (che, per numeri, in Python si ottengono con `x**n`). Anche qui esiste un algoritmo del contadino russo che può essere formulato così: Sia g la funzione di 2 variabili definita da $g(x,n) := x^n$. Allora

$$g(x,n) = \begin{cases} 1 & \text{se } n = 0 \\ g(x^2, n/2) & \text{se } n \text{ è pari } \neq 0 \\ x \cdot g(x, n-1) & \text{se } n \text{ è dispari} \end{cases}$$

In Python definiamo la funzione nel modo seguente:

```
# Potenza russa.
def g(x,n):
  if n==0: return 1
  if n%2==0: return g(x*x,n/2)
  return x*g(x,n-1)
```

Se questa funzione la inseriamo nel file *russo.py*, nel file *alfa* che funge da programma principale possiamo scrivere

```
x=2; n=16
print russo.g(x,n)
# Output: 65536

x=2; n=32
print russo.g(x,n)
# Output: 4294967296
```

Confrontando i due casi, ci si accorge che l'algoritmo è sempre lo stesso - cambia soltanto l'operazione di $\mathbb{N}$ sugli argomenti!

Rappresentazione binaria

Ogni numero naturale n possiede una rappresentazione binaria, cioè una rappresentazione della forma

$$n = a_k 2^k + a_{k-1} 2^{k-1} + \cdots + a_1 2 + a_0$$

con coefficienti (o cifre binarie) $a_i \in \{0,1\}$. Per $n = 0$ usiamo $k = 0$ ed $a_0 = 0$; per $n > 0$ chiediamo che $a_k \neq 0$. Con queste condizioni k e gli a_i sono univocamente determinati. Sia $r_2(n) = (a_k, \ldots, a_0)$ il vettore i cui elementi sono queste cifre. Dalla rappresentazione binaria si deduce la seguente relazione ricorsiva:

$$r_2(n) = \begin{cases} (n) & \text{se } n \leq 1 \\ (r_2(\frac{n}{2}), 0) & \text{se } n \text{ è pari} \\ (r_2(\frac{n-1}{2}), 1) & \text{se } n \text{ è dispari} \end{cases}$$

È molto facile tradurre questa idea in una funzione di Python. Siccome Python per operandi interi esegue una divisione intera, anche per n dispari possiamo scrivere `n/2`, in tal caso automaticamente uguale a `(n-1)/2`.

```
def rapp2pv (n): # Prima versione.
  if n<=1: v=[n]
  else:
    v=rapp2pv(n/2)
    if n%2==0: v.append(0)
    else: v.append(1)
  return v
```

La versione definitiva della funzione prevede un secondo parametro facoltativo `cifre`; quando questo è maggiore del numero di cifre necessarie per la rappresentazione binaria di n, i posti iniziali vuoti vengono riempiti con zeri.

```
def rapp2 (n,cifre=0):
  if n<=1: v=[n]
  else:
    v=rapp2(n/2)
    if n%2==0: v.append(0)
    else: v.append(1)
  d=cifre-len(v)
  if d>0: v=[0]*d+v
  return v
```

Per provare la funzione usiamo la possibilità di costruire una stringa da una lista di numeri mediante la funzione `str`, come abbiamo visto a pagina 20, ottenendo l'output

```
  0  0 0 0 0 0 0 0 0
  1  0 0 0 0 0 0 0 1
  2  0 0 0 0 0 0 1 0
  3  0 0 0 0 0 0 1 1
  4  0 0 0 0 0 1 0 0
  5  0 0 0 0 0 1 0 1
  6  0 0 0 0 0 1 1 0
  7  0 0 0 0 0 1 1 1
  8  0 0 0 0 1 0 0 0
  9  0 0 0 0 1 0 0 1
 10  0 0 0 0 1 0 1 0
 11  0 0 0 0 1 0 1 1
 12  0 0 0 0 1 1 0 0
 19  0 0 0 1 0 0 1 1
 48  0 0 1 1 0 0 0 0
 77  0 1 0 0 1 1 0 1
106  0 1 1 0 1 0 1 0
135  1 0 0 0 0 1 1 1
164  1 0 1 0 0 1 0 0
194  1 1 0 0 0 0 1 0
221  1 1 0 1 1 1 0 1
```

con

```
def strdalista (a,sep=' '):
  return sep.join(map(str,a))

for n in range(13)+ \
  [19,48,77,106,135,164,194,221]:
  print "%3d  %s" \
    %(n,strdalista(rapp2(n,cifre=8)))
```

Se usiamo invece

```
for n in xrange(256):
  print "%3d  %s" \
    %(n,strdalista(rapp2(n,cifre=8)))
```

otteniamo gli elementi dell'ipercubo $\{0,1\}^8$.

Una rappresentazione binaria a lunghezza fissa di valori numerici viene anche usata nella trasmissione di segnali, ad esempio dei valori di grigio di un'immagine bianco-nera satellitare.

Numeri esadecimali

Nei linguaggi macchina e assembler molto spesso si usano i numeri esadecimali o, più correttamente, la rappresentazione esadecimale dei numeri naturali, cioè la loro rappresentazione in base 16.

Per $2789 = 10 \cdot 16^2 + 14 \cdot 16 + 5 \cdot 1$ potremmo ad esempio scrivere

`2789 = (10,14,5)`$_{16}$.

In questo senso `10`,`14` e `5` sono le cifre della rappresentazione esadecimale di `2789`. Per poter usare lettere singole per le cifre si indicano le cifre `10`,...,`15` mancanti nel sistema decimale nel modo seguente:

10	A
11	B
12	C
13	D
14	E
15	F

In questo modo adesso possiamo scrivere $2789 = (AE5)_{16}$. In genere si possono usare anche indifferentemente le corrispondenti lettere minuscole. Si noti che `(F)`$_{16}$ `= 15` assume nel sistema esadecimale lo stesso ruolo come il `9` nel sistema decimale. Quindi `(FF)`$_{16}$ `= 255` $= 16^2 - 1 = 2^8 - 1$. Un numero naturale n con $0 \le n \le 255$ si chiama un *byte*, un *bit* è invece uguale a 0 o a 1. Esempi:

	0	(0)$_{16}$
	14	(E)$_{16}$
	15	(F)$_{16}$
	16	(10)$_{16}$
	28	(1C)$_{16}$
2^5	32	(20)$_{16}$
2^6	64	(40)$_{16}$
	65	(41)$_{16}$
	97	(61)$_{16}$
	127	(7F)$_{16}$
2^7	128	(80)$_{16}$
	203	(CB)$_{16}$
	244	(F4)$_{16}$
	255	(FF)$_{16}$
2^8	256	(100)$_{16}$
2^{10}	1024	(400)$_{16}$
2^{12}	4096	(1000)$_{16}$
	65535	(FFFF)$_{16}$
2^{16}	65536	(10000)$_{16}$

Si vede da questa tabella che i byte sono esattamente quei numeri per i quali sono sufficienti due cifre esadecimali.

Lo schema di Horner

Sia dato un polinomio

$$f = a_0 x^n + a_1 x^{n-1} + \cdots + a_n \in A[x]$$

dove A è un qualsiasi anello commutativo.

Per $\alpha \in A$ vogliamo calcolare $f(\alpha)$.
Sia ad esempio $f = 3x^4 + 5x^3 + 6x^2 + 8x + 7$. Poniamo

$$\begin{aligned} b_0 &= 3 \\ b_1 &= b_0\alpha + 5 \;=\; 3\alpha + 5 \\ b_2 &= b_1\alpha + 6 \;=\; 3\alpha^2 + 5\alpha + 6 \\ b_3 &= b_2\alpha + 8 \;=\; 3\alpha^3 + 5\alpha^2 + 6\alpha + 8 \\ b_4 &= b_3\alpha + 7 \;=\; 3\alpha^4 + 5\alpha^3 + 6\alpha^2 + 8\alpha + 7 \end{aligned}$$

e vediamo che $b_4 = f(\alpha)$. Lo stesso si può fare nel caso generale:

$$\begin{aligned} b_0 &= a_0 \\ b_1 &= b_0\alpha + a_1 \\ &\cdots \\ b_k &= b_{k-1}\alpha + a_k \\ &\cdots \\ b_n &= b_{n-1}\alpha + a_n \end{aligned}$$

con $b_n = f(\alpha)$, come dimostriamo adesso.

Consideriamo il polinomio

$$g := b_0 x^{n-1} + b_1 x^{n-2} + \cdots + b_{n-1}.$$

Allora, usando che $\alpha b_k = b_{k+1} - a_{k+1}$ per $k = 0, \ldots, n-1$, abbiamo

$$\begin{aligned} \alpha g &= \alpha b_0 x^{n-1} + \alpha b_1 x^{n-2} + \cdots + \alpha b_{n-1} \\ &= (b_1 - a_1)x^{n-1} + (b_2 - a_2)x^{n-2} + \ldots \\ &\quad + (b_{n-1} - a_{n-1})x + b_n - a_n \\ &= (b_1 x^{n-1} + b_2 x^{n-2} + \cdots + b_{n-1}x + b_n) \\ &\quad - (a_1 x^{n-1} + a_2 x^{n-2} + \ldots a_{n-1}x + a_n) \\ &= x(g - b_0 x^{n-1}) + b_n - (f - a_0 x^n) \\ &= xg - b_0 x^n + b_n - f + a_0 x^n \\ &= xg + b_n - f \end{aligned}$$

quindi

$$\boxed{f = (x - \alpha)g + b_n}$$

e ciò implica $f(\alpha) = b_n$.

$b_0, \ldots, b_{n-1}$ sono perciò i coefficienti del quoziente nella divisione con resto di f per $x - \alpha$, mentre b_n è il resto, uguale a $f(\alpha)$.

Questo algoritmo è detto *schema di Horner* o *schema di Ruffini* ed è molto più veloce del calcolo separato delle potenze di α (tranne nel caso che il polinomio consista di una sola o di pochissime potenze, in cui si userà invece semplicemente l'operazione `x**n`).

Abbiamo menzionato l'algoritmo di Horner già a pagina 15; modifichiamo leggermente la funzione che vogliamo usare in Python:

```
def horner (a,x):
  b=0
  for ak in a: b=b*x+ak
  return b
```

Una frequente applicazione dello schema di Horner è il calcolo del valore corrispondente a una rappresentazione binaria o esadecimale. Infatti otteniamo $(1,0,0,1,1,0,1,1,1)_2$ come

```
horner([1,0,0,1,1,0,1,1,1],2)
```

e $(A, F, 7, 3, 0, 5, E)_{16}$ come

```
horner([10,15,7,3,0,5,14],16):

x=horner([1,0,0,1,1,0,1,1,1],2)
print x # 311

y=horner([10,15,7,3,0,5,14],16)
print y # 183971934
```

Impostare il limite di ricorsione

Se definiamo il fattoriale con

```
def fatt (n):
  if n==0: return 1
  else: return n*fatt(n-1)
```

riusciamo a calcolare con `fatt(998)` il fattoriale $998!$, mentre il programma termina con un errore se proviamo `fatt(999)`. Abbiamo infatti superato il limite di ricorsione, più precisamente dello stack utilizzato per l'esecuzione delle funzioni, inizialmente impostato a 1000. Si può ridefinire questo limite con la funzione `sys.setrecursionlimit`:

```
sys.setrecursionlimit(2000)

print fatt(1998) # Funziona.
```

Per sapere il limite di ricorsione attuale si può usare la funzione `sys.getrecursionlimit`:

```
print sys.getrecursionlimit()
# 1000 (se non reimpostato)
```

Zeri di una funzione continua

Siano $a < b$ e $f : [a, b] \longrightarrow \mathbb{R}$ una funzione continua tale che $f(a) < 0$ e $f(b) > 0$. In analisi si impara che allora la funzione f deve contenere uno zero nell'intervallo (a, b). Da questo fatto deriva un buon metodo elementare e facile da ricordare, detto il metodo del *sorpasso e ritorno*, per la ricerca delle radici di una funzione continua, che in un pseudolinguaggio può essere formulato nel modo seguente:

```
if b − a < ε then return (a, b)
x = (a + b)/2
if f(x) == 0 then return (x, x)
if f(x) > 0 then cerca in (a, x) # ricorsione
else cerca in (x, b) # ricorsione
```

$\varepsilon > 0$ è qui la precisione richiesta nell'approssimazione al valore x della radice; cioè ci fermiamo quando abbiamo trovato un intervallo di lunghezza $< \varepsilon$ al cui interno si deve trovare uno zero della funzione. È chiaro che questo algoritmo teoricamente deve terminare. In pratica però potrebbe non essere così. Infatti, se ε è minore della precisione della macchina, a un certo punto si avrà che il valore effettivamente calcolato come approssimazione di $x = \frac{a+b}{2}$ è uguale a b, e quindi, se al passo (4) dobbiamo sostituire b con x, rimaniamo sempre nella situazione (a, b) e avremo un ciclo infinito.

Assumiamo ad esempio che $a = 3.18$ e $b = 3.19$ e che la macchina arrotondi a due cifre decimali. Allora $a + b = 6.37$ e $x = \frac{a+b}{2} = 3.185$ che viene arrotondato a $3.19 = b$. Se noi avessimo impostato $\varepsilon = 0.001$, il programma possibilmente non termina.

Si può però sfruttare questo fenomeno a nostro favore: l'imprecisione della macchina fa in modo che prima o poi arriviamo a $x = a$ o $x = b$, e in quel momento ci fermiamo, potendo così applicare un criterio di interruzione indipendente dalla macchina.

In Python possiamo formulare questa idea nel modo seguente:

```
def zero (f,a,b):
  x=(a+b)/2.0
  if x==a or x==b: return (a,b)
  y=f(x)
  if y==0: return (x,x)
  if y>0: return zero(f,a,x)
  return zero(f,x,b)
```

Prima di usare la funzione bisogna sempre verificare che veramente $f(a) < 0$ e $f(b) > 0$ e sostituire f con $-f$ quando ciò non accade; non sarebbe difficile migliorare la funzione in modo che essa stessa esegua questa verifica.

Calcoliamo con il nostro algoritmo una delle (al massimo 4) radici del polinomio $f = x^4 + x - 7$. Verifichiamo prima che $f(0) = -7 < 0$ e $f(2) = 16 + 2 - 7 = 11 > 0$. Se la funzione si trova nel file *vari.py*, possiamo scrivere

```
def f (x): return x**4+x-7

print vari.zero(f,0,2)
# Output:
(1.5293593647724706, 1.5293593647724708)
```

Spesso la funzione f è differenziabile; in questo caso l'analisi numerica fornisce metodi molto più veloci e più accurati (in genere varianti del *metodo di Newton*), in particolare si possono in tal caso dare stime per l'errore commesso. Se invece la funzione non è nemmeno continua, si ricorre talvolta ad algoritmi genetici.

Due funzioni per matrici 2×2

Creiamo un file *matrici.py* in cui inseriamo due funzioni per il calcolo del determinante di una matrice 2×2 e per la regola di Cramer in dimensione due.

```
def det2 (A): (a,b)=A[0]; (c,d)=A[1]; return a*d-b*c

def cramer2 (A,c):
  (a1,b1)=A[0]; (a2,b2)=A[1]; pc=[a1,a2]; sc=[b1,b2]
  D=float(det2(A))
  x=det2([c,sc])/D; y=det2([pc,c])/D; return [x,y]
```

Esercizi per gli scritti

26. Una funzione in Python che accetta come argomenti due liste di numeri e restituisce la lista che consiste delle somme dei coefficienti delle liste date. Se le due liste date non hanno lunghezza uguale, il risultato deve essere `None`.

27-28. L'anno solare è lungo 365.24219 giorni. Mediante l'algoritmo euclideo calcolare la frazione continua di 24219/100000 che deve risultare uguale a $[0, 4, 7, 1, 3, 24, 6, 2, 2]$.

La quinta approssimante (il quoziente dei numeri nella quinta colonna dello schema) è $31/128$ (con un errore, come si può dimostrare, minore di $1/128^2$). Siccome $31/128 = 1/4 - 1/128$, si vede che un calendario sensibilmente più preciso di quello in vigore consisterebbe di scegliere come anni bisestili quegli anni n per cui n è multiplo di 4, ma non multiplo di 128.

29. Una funzione in Python che accetta due stringhe e restituisce la stringa che si ottiene eliminando dalla prima tutte le lettere che appaiono nella seconda.

30. Usare il metodo del sorpasso e ritorno per trovare uno zero positivo della funzione f definita da $f(x) = e^x - 10 \cos x$.

31. Trovare $\sqrt[7]{3}$ con sorpasso e ritorno.

32. Trovare $\sqrt{\sqrt[3]{10} + 1}$ con sorpasso e ritorno.

33. La funzione contiene un errore di sintassi. Quale?

```
def f(x):
  u=math.cos(x); if u<0: return f(x+u)
  return 0
```

34. La funzione contiene un errore di sintassi. Quale?

```
def f(x):
return x*x
```

35. La funzione contiene un errore di sintassi. Quale?

```
def f(x,y):
  if x<y: return y
  elif x==y: return 0
  else return x
```

36. La funzione è sintatticamente corretta, ma non restituisce il risultato desiderato.

```
def media (a):
  n=len(a); s=0
  for x in a: s=s+x
  return s/n
```

37. La funzione è sintatticamente corretta, ma non restituisce il risultato desiderato.

```
def cubo (x): x*x*x
```

38. Una funzione `lin` che accetta a e b come argomenti e calcola la soluzione di $ax = b$. Se l'equazione non ha soluzione, il risultato è la lista `[False]` (per non confondere `False` con il numero 0), se ogni x è soluzione, il risultato è `[True]`.

39. Il prodotto di due matrici 2×2 $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ e $B = \begin{pmatrix} x & y \\ z & t \end{pmatrix}$ è definito come

$$AB := \begin{pmatrix} ax + bz & ay + bt \\ cx + dz & cy + dt \end{pmatrix}$$

In Python rappresentiamo A nella forma `A=[[a,b],[c,d]]`. Scrivere una funzione che calcola il prodotto di due matrici 2×2.

40. Dimostrare con un calcolo diretto che per matrici 2×2 si ha $|AB| = |A||B|$. Il determinante di un prodotto è quindi il prodotto dei determinanti. Nel corso di Geometria si dimostra che questa importante regola vale per matrici $n \times n$ per ogni n. Il caso $n = 2$ implica in particolare che il prodotto di due elementi di $SL(2, \mathbb{N})$ appartiene ancora ad $SL(2, \mathbb{N})$.

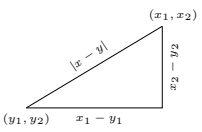
VII. IL PRODOTTO SCALARE

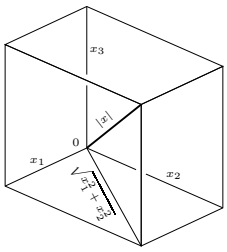
Distanze in $\mathbb{R}^n$

La distanza tra due punti $x = (x_1, x_2)$ e $y = (y_1, y_2)$ del piano reale $\mathbb{R}^2$ si calcola secondo il teorema di Pitagora come

$$|x - y| = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}.$$

La distanza del punto x dall'origine è quindi $|x| = \sqrt{x_1^2 + x_2^2}$ e viceversa la distanza di x e y è proprio la lunghezza del vettore $x - y$.





Formule del tutto analoghe si hanno nello spazio tridimensionale $\mathbb{R}^3$. Calcoliamo prima la lunghezza $|x|$ di un vettore $x = (x_1, x_2, x_3)$ utilizzando la figura a destra, dalla quale si vede che

$$|x|^2 = (\sqrt{x_1^2 + x_2^2})^2 + x_3^2 = x_1^2 + x_2^2 + x_3^2,$$

per cui

$$|x| = \sqrt{x_1^2 + x_2^2 + x_3^2}.$$

Se adesso $y = (y_1, y_2, y_3)$ è un altro punto, la distanza tra x e y sarà uguale alla lunghezza di $x - y$, quindi

$$|x - y| = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + (x_3 - y_3)^2}.$$

Per ogni $n \geq 1$ possiamo definire lunghezze e distanze in $\mathbb{R}^n$ nello stesso modo. Per $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ poniamo

$$|x| := \sqrt{x_1^2 + \cdots + x_n^2},$$

e se $y = (y_1, \ldots, y_n)$ è un altro punto, la distanza tra x e y è la lunghezza di $x - y$, cioè

$$|x - y| = \sqrt{(x_1 - y_1)^2 + \cdots + (x_n - y_n)^2}.$$

Il prodotto scalare

Siano come sopra $x = (x_1, \ldots, x_n)$ e $y = (y_1, \ldots, y_n)$ due punti di $\mathbb{R}^n$. Calcoliamo la lunghezza $|x + y|$ della somma dei due vettori; questo è anche in statistica il punto di partenza per la definizione del *coefficiente di correlazione* che, nonostante il nome prometta molto di più, non è altro che un mezzo per confrontare x, y e $x + y$!

$$\begin{aligned} |x + y|^2 &= \sum_{k=1}^{n} (x_k + y_k)^2 = \sum_{k=1}^{n} x_k^2 + \sum_{k=1}^{n} y_k^2 + 2 \sum_{k=1}^{n} x_k y_k \\ &= |x|^2 + |y|^2 + 2 \sum_{k=1}^{n} x_k y_k \end{aligned}$$

L'espressione $\sum_{k=1}^{n} x_k y_k$ si chiama il *prodotto scalare* dei vettori x ed y. Esso è di fondamentale importanza per tutta la geometria. Introduciamo le abbreviazioni

$$\|x, y\| := (x, y) := x_1 y_1 + \cdots + x_n y_n$$

La seconda è più diffusa della prima, comporta però il pericolo di confusione con la coppia (x, y) che ad esempio nella statistica multidimensionale appare spesso contemporaneamente.

Sostituendo y con $-y$ otteniamo

$$|x - y|^2 = |x|^2 + |y|^2 - 2\|x, y\|.$$

I due punti x ed y formano insieme all'origine 0 un triangolo (eventualmente degenerato) i cui lati hanno le lunghezze $|x|$, $|y|$ e $|x - y|$. Assumiamo che il triangolo non sia degenerato e sia α l'angolo opposto al lato di lunghezza $|x - y|$. Per il teorema del coseno abbiamo

$$|x - y|^2 = |x|^2 + |y|^2 - 2|x||y| \cos\alpha, \text{ da cui } \|x, y\| = |x||y| \cos\alpha$$

In particolare $\|x, y\| = 0 \iff \cos\alpha = 0$.

Combinando il `for` con `zip` possiamo calcolare il prodotto scalare di due vettori:

```
def prodottoscalare (u,v):
  s=0
  for x,y in zip(u,v): s+=x*y
  return s

u=[1,3,4]
v=[6,2,5]

print prodottoscalare(u,v)
# 32
```

La lunghezza $|v|$ di un vettore v è uguale a $\sqrt{\|v, v\|}$, quindi in Python possiamo definire la funzione

```
# Lunghezza di un vettore.
def lun (v): return math.sqrt(prodottoscalare(v,v))
```

Ortogonalità

La formula fondamentale

$$\|x, y\| = |x||y| \cos\alpha$$

rimane valida anche se x e y sono uno un multiplo dell'altro, ad esempio $y = tx$ per $t \in \mathbb{R}$, però entrambi $\neq 0$ (ciò implica $t \neq 0$). In questo caso infatti il triangolo determinato da x, y e 0 è degenerato, ma è naturale assegnare all'angolo tra x e y il valore 0 (per cui $\cos\alpha = 1$) se $t > 0$ e invece il valore $180°$ (cosicché $\cos\alpha = -1$) se $t < 0$.

Inoltre allora $\|x, y\| = \|x, tx\| = t\|x, x\| = t|x|^2$ e $|x||y| = |x||tx| = |t||x||x| = |t||x|^2$. Dimostrare queste relazioni (cfr. es. 53) e concludere da soli, stando attenti ai segni.

Quindi se i due vettori sono diversi da zero (ciò implica che anche $|x| \neq 0$ e $|y| \neq 0$), allora essi sono ortogonali (cioè $\alpha = 90°$ oppure $\alpha = 270°$) se e solo se $\cos\alpha = 0$, cioè se e solo se $\|x, y\| = 0$.

Siccome infine $\|x, 0\| = 0$ per ogni x, è naturale includere anche il vettore 0 tra i vettori ortogonali ad x. Raccogliendo tutto possiamo perciò dire:

Due vettori x ed y di $\mathbb{R}^n$ sono ortogonali se e solo se $\|x, y\| = 0$.

Esempio 29.1. Siano $v = (v_1, v_2) \in \mathbb{R}^2$ e $v^* = (-v_2, v_1)$ il vettore magico di v. Allora $\|v, v^*\| = -v_1 v_2 + v_1 v_2 = 0$; ciò mostra che veramente, come intuitivamente è evidente, v e v^* sono ortogonali tra di loro.

Nell'analisi complessa si impara che la moltiplicazione con i corrisponde a una rotazione di $90°$ (in senso antiorario, come sempre), e infatti $i(a + bi) = ia + i^2 b = -b + ia$.

Disuguaglianze fondamentali

Teorema 29.2 (disuguaglianza di Cauchy-Schwarz). *Siano $x = (x_1, \ldots, x_n)$ e $y = (y_1, \ldots, y_n)$ due punti di $\mathbb{R}^n$. Allora*

$$|\|x, y\|| \leq |x||y|$$

Dimostrazione. Possiamo ricondurre questa fondamentale disuguaglianza al caso $n = 2$. Infatti i due vettori stanno su un piano e il prodotto scalare si esprime mediante l'angolo α che essi formano in questo piano:

$$\|x, y\| = |x||y| \cos\alpha$$

e siccome $|\cos\alpha| \leq 1$ abbiamo

$$|\|x, y\|| = |x||y||\cos\alpha| \leq |x||y|$$

Proposizione 29.3 (disuguaglianza triangolare). *Siano ancora $x = (x_1, \ldots, x_n)$ ed $y = (y_1, \ldots, y_n)$ due punti di $\mathbb{R}^n$. Allora*

$$|x + y| \leq |x| + |y|$$

Dimostrazione. Ciò è una facile conseguenza della formula

$$|x + y|^2 = |x|^2 + |y|^2 + 2\|x, y\|$$

per il prodotto scalare e della disuguaglianza di Cauchy-Schwarz:

$$\begin{aligned} |x + y|^2 &= |x|^2 + |y|^2 + 2\|x, y\| \\ &\leq |x|^2 + |y|^2 + 2|x||y| = (|x| + |y|)^2 \end{aligned}$$

per cui anche $|x + y| \leq |x| + |y|$.

Il segno del prodotto scalare

Nota 30.1. Nella disuguaglianza di Cauchy-Schwarz anche a sinistra dobbiamo mettere il segno di valore assoluto, perché il prodotto scalare può essere negativo.

Infatti il segno del prodotto scalare ha una importantissima interpretazione geometrica: Siano come finora $x = (x_1, \ldots, x_n)$ e $y = (y_1, \ldots, y_n)$ due punti di $\mathbb{R}^n$, entrambi diversi da 0. Come nella dimostrazione della disuguaglianza di Cauchy-Schwarz sia α l'angolo che i due vettori formano in un piano comune (un tale piano esiste sempre ed è univocamente determinato se i due vettori non sono paralleli). Sappiamo che $\|x, y\| = |x||y| \cos\alpha$ e per ipotesi $|x| > 0$ e $|y| > 0$. Ciò implica che $\|x, y\|$ e $\cos\alpha$ hanno lo stesso segno; in particolare

$$\|x, y\| \geq 0 \iff \cos\alpha \geq 0$$

Fissiamo adesso x. Allora i vettori $y \in \mathbb{R}^n$ per i quali vale $\cos\alpha = 0$ sono esattamente i vettori ortogonali ad x. Essi formano l'*iperpiano ortogonale* ad x (una retta ortogonale ad x in $\mathbb{R}^2$, un piano ortogonale ad x in $\mathbb{R}^3$). Come si vede dalle figure a pagina 29, per $y = x$ il coseno di α è uguale ad 1, e se, partendo da $y = x$, avviciniamo y all'iperpiano ortogonale di x, il coseno diventa sempre più piccolo, rimanendo però positivo fino a quando non tocchiamo l'iperpiano ortogonale. Se y passa invece dall'altra parte dell'iperpiano, il coseno di α diventa negativo.

Avendo $\|x, y\|$ e $\cos\alpha$ però lo stesso segno, otteniamo il seguente importante enunciato geometrico:

Teorema 30.2. *Siano $x = (x_1, \ldots, x_n)$ e $y = (y_1, \ldots, y_n)$ due punti di $\mathbb{R}^n$, entrambi diversi da zero. Allora $\|x, y\| > 0$ se e solo se y si trova dalla stessa parte dell'iperpiano ortogonale ad x come x stesso.*

Area di un parallelogramma

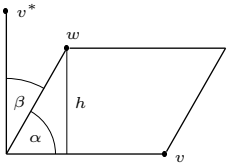
Lemma 30.3. *Siano $v = (v_1, v_2)$ e $w = (w_1, w_2)$ vettori in $\mathbb{R}^2$. $v^* = (-v_2, v_1)$ sia il vettore magico di v. Allora $\|v^*, w\| = \det(v, w)$.*

Dimostrazione. Infatti

$$\|v^*, w\| = -v_2 w_1 + v_1 w_2 = \begin{vmatrix} v_1 & w_1 \\ v_2 & w_2 \end{vmatrix} = \det(v, w)$$

Nota 30.4. Consideriamo due vettori linearmente indipendenti v e w in $\mathbb{R}^n$. Insieme all'origine essi determinano un parallelogramma la cui area, con le notazione nella figura, è uguale a $|v|h = |v||w||\sin\alpha|$, dove abbiamo usato il valore assoluto del seno per non essere obbligati a un particolare orientamento dell'angolo α.

Assumiamo adesso che $v, w \in \mathbb{R}^2$. In questo caso l'area del parallelogramma può essere espressa come valore assoluto del determinante dei due vettori, come adesso vediamo. Questo risultato può essere ottenuto in modo elegante se introduciamo di nuovo il vettore magico $v^* = (-v_2, v_1)$ che si ottiene girando v per 90 gradi in senso antiorario.



In primo luogo $|\sin\alpha| = |\cos\beta|$, per cui

$$h = |w||\sin\alpha| = |w||\cos\beta| = \frac{|w|||v^*, w|||}{|v^*||w|}$$
$$= \frac{|\|v^*, w\||}{|v^*|} = \frac{|\|v^*, w\||}{|v|} = \frac{|\det(v, w)|}{|v|}$$

dove abbiamo usato il lemma 30.3 e il fatto che $|v^*| = |v|$.

L'area del parallelogramma è uguale ad $h|v|$, quindi uguale a

$$|\det(v, w)| = |v_1 w_2 - v_2 w_1|.$$

L'area di un parallelogramma nel piano è perciò uguale al valore assoluto del determinante che si ottiene dai due vettori. Se questi sono linearmente dipendenti, l'area è evidentemente nulla così come il determinante, e vediamo che la formula vale per una coppia qualsiasi di vettori del piano. Una formula analoga vale in $\mathbb{R}^n$ come dimostreremo per $n = 3$.

Esercizi per gli scritti

41. Con $A := \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ e $B := \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ calcolare prima A^n e B^n per $n \in \mathbb{N}$ e poi $A^2B^3AB^4$ e $A^2B^3AB^4A^3$.

Confrontare i risultati con i coefficienti negli schemi per le frazioni continue $[2, 3, 1, 4]$ e $[2, 3, 1, 4, 3]$.

42. La funzione è sintatticamente corretta, ma date due liste $[a_0, \ldots, a_p]$ e $[b_0, \ldots, b_q]$, non restituisce la lista $[a_0, \ldots, a_i, b_{i+1}, \ldots, b_q]$.

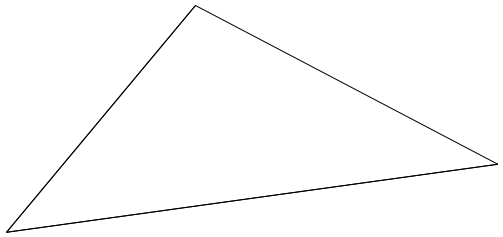
```
def unisci (a,i,b): return a[:i]+b[i+1:]
```

43. Per una sequenza finita $(a_1, \ldots, a_n)$ la *mediana* di a è definita come $(a_{n/2} + a_{n/2+1})/2$, se n e pari, come $a_{(n+1)/2}$ altrimenti. Usare `sort` per scrivere una funzione in Python che calcola la mediana di una lista o di una tupla. Attenti agli indici. La funzione non deve modificare la lista data come argomento!

44. La funzione contiene un errore di sintassi. Quale?

```
def f(x):
  if x==0: return 0
  else if x>0: return math.log(x)
  return math.log(-x)
```

45. Se si dispone solo di una riga ma non di un goniometro, come si fa a misurare gli angoli di un triangolo? Provare con il triangolo nella figura.
Una volta capito come si fa, creare una funzione in Python.



46. Sia $f = 3 + 5x + 2x^2 + 3x^3 + x^4 + 2x^5$. Calcolare a mano $f(2)$, usando lo schema di Horner. Non riscrivere la teoria, ma cercare di utilizzare una disposizione essenziale dei singoli passaggi in righe della forma $26 + 2 = 28$ (tranne la prima).

47. Sia $f = 2x^6 + x^5 + 4x^3 + x^2 + 7$. Calcolare a mano $f(-1)$, usando lo schema di Horner.

48. Calcolare a mano $(1011101101101)_2$.

49. Calcolare a mano $(AE2C0B)_{16}$.

50. Calcolare la distanza di $(3, 1, 5)$ e $(6, 2, 3)$ in $\mathbb{R}^3$.

51. Calcolare la distanza di $(1, 2, 3, 7, 2)$ e $(1, 4, 3, 2, 4)$ in $\mathbb{R}^5$.

52. Sia $v \in \mathbb{R}^n$. Allora $|v| = 0 \iff v = 0$.

53. *Bilinearità del prodotto scalare.* Siano $x, y, z \in \mathbb{R}^n$ ed $\alpha, \beta \in \mathbb{R}$. Allora $\|\alpha x + \beta y, z\| = \alpha\|x, z\| + \beta\|y, z\|$.

54. $v_1, \ldots, v_m$ siano vettori $\neq 0$ di $\mathbb{R}^n$, a due a due ortogonali tra di loro. Allora questi vettori sono linearmente indipendenti.

55. *Sviluppo di Fourier.* $v_1, \ldots, v_m$ siano vettori $\neq 0$ di $\mathbb{R}^n$, a due a due ortogonali tra di loro e tali che $v_k = 1$ per ogni k.
x sia un vettore che si esprime come combinazione lineare dei vettori dati: $x = \lambda_1 v_1 + \ldots + \lambda_m v_m$ con $\lambda_1, \ldots, \lambda_m \in \mathbb{R}$. Calcolare i coefficienti λ_k. Un fatto sorprendente e importante è che λ_k dipende solo da x e v_k e non dagli altri v_j.

VIII. IL PRODOTTO VETTORIALE

Un po' di algebra esterna

Situazione 31.1. V sia uno spazio vettoriale su $\mathbb{R}$ (oppure su un campo K qualsiasi - non ha importanza).

Definizione 31.2. Vettori $v_1, \ldots, v_m \in V$ si dicono *linearmente indipendenti*, se esistono $\lambda_1, \ldots, \lambda_m \in \mathbb{R}$ non tutti uguali a 0, tali che $\lambda_1 v_1 + \ldots + \lambda_m v_m = 0$. Altrimenti diciamo che $v_1, \ldots, v_m$ sono *linearmente indipendenti*.

$v_1, \ldots, v_m$ sono quindi linearmente indipendenti se e solo se

$$\lambda_1 v_1 + \ldots + \lambda_m v_m = 0$$

con $\lambda_1, \ldots, \lambda_m \in \mathbb{R}$ implica $\lambda_1 = \ldots = \lambda_m = 0$.

Definizione 31.3. Per $v_1, \ldots, v_m \in V$ usiamo la seguente abbreviazione:

$v_1 \wedge \ldots \wedge v_m = 0$ se $v_1, \ldots, v_m$ sono linearmente dipendenti.

$v_1 \wedge \ldots \wedge v_m \neq 0$ se $v_1, \ldots, v_m$ sono linearmente indipendenti.

Questa notazione appartiene all'*algebra esterna*, un metodo molto potente che permette di amministrare in modo algebrico e trasparente tutte le formule che riguardano determinanti, sottodeterminanti, prodotto vettoriale ($v \times w$), dipendenza lineare ecc. Nell'algebra lineare un po' un più avanzata per ogni $m \geq 1$ si costruisce uno spazio vettoriale $\Lambda^m V$ a cui $v_1 \wedge \ldots \wedge v_m$ appartiene.

Posto $\Lambda^0 V := \mathbb{R}$ si forma $\Lambda V := \bigcup_{m=0}^{\infty} \Lambda^m V$.

ΛV è in modo naturale uno spazio vettoriale su $\mathbb{R}$ e allo stesso tempo un anello (non commutativo) con il prodotto

$$(v_1 \wedge \ldots \wedge v_m) \wedge (w_1 \wedge \ldots \wedge w_s) := v_1 \wedge \ldots \wedge v_m \wedge w_1 \wedge \ldots \wedge w_s$$

Noi non effettuiamo questa costruzione, ma faremo adesso vedere che già con soli ragionamenti formali riusciamo a ottenere molti risultati utili che permettono un'espressione algebrica di molte situazioni geometriche.

Osservazione 31.4. Il simbolo $\wedge$ nella matematica viene usato in tre contesti diversi con significati diversi:

- ✶ Nell'algebra multilineare per il prodotto esterno come nella def. 31.3;
- ✶ in logica per la congiunzione logica (AND, e);
- ✶ in analisi per denotare il minimo tra due o più elementi.

Teorema 31.5. *Siano $u = (u_1, \ldots, u_n)$ e $v = (v_1, \ldots, v_n)$ due vettori di $\mathbb{R}^n$. Allora i seguenti enunciati sono equivalenti:*

(1) $u \wedge v = 0$.

(2) $\begin{vmatrix} u_i & v_i \\ u_j & v_j \end{vmatrix} = 0$ *per ogni* i, j *con* $1 \leq i, j \leq n$.

(3) $\begin{vmatrix} u_i & v_i \\ u_j & v_j \end{vmatrix} = 0$ *per ogni* i, j *con* $1 \leq i < j \leq n$.

Dimostrazione. È chiaro che (2) e (3) sono equivalenti, perché per $i = j$ questi determinanti sono in ogni caso nulli, mentre, se scambiamo i e j, cambiano solo di segno.

(1) $\Longrightarrow$ (2): Se u e v sono linearmente dipendenti, allora uno dei due, ad esempio v, è un multiplo dell'altro: $v = \lambda u$ con $\lambda \in \mathbb{R}$. Allora

$$\begin{vmatrix} u_i & v_i \\ u_j & v_j \end{vmatrix} = \begin{vmatrix} u_i & \lambda u_i \\ u_j & \lambda u_j \end{vmatrix} = \lambda u_i u_j - \lambda u_i u_j = 0.$$

(2) $\Longrightarrow$ (1): Se $u = 0$, tutti gli enunciati sono banalmente veri. Sia quindi $u \neq 0$, ad esempio $u_1 \neq 0$. Per ipotesi, per ogni j vale

$$\begin{vmatrix} u_1 & v_1 \\ u_j & v_j \end{vmatrix} = 0$$

cioè $u_1 v_j - u_j v_1 = 0$ e quindi $v_j = \dfrac{v_1}{u_1} u_j$ per ogni j.

Cio significa che $v = \lambda u$ con $\lambda = \dfrac{v_1}{u_1}$.

Corollario 31.6. *Siano $u = (u_1, u_2)$ e $v = (v_1, v_2)$ due vettori del piano $\mathbb{R}^2$. Allora i seguenti enunciati sono equivalenti:*

(1) $u \wedge v = 0$.

(2) $\begin{vmatrix} u_1 & v_1 \\ u_2 & v_2 \end{vmatrix} = 0$

Teorema 31.7. *Siano $u = (u_1, \ldots, u_n)$, $v = (v_1, \ldots, v_n)$ e $w = (w_1, \ldots, w_n)$ tre vettori di $\mathbb{R}^n$. Allora i seguenti enunciati sono equivalenti:*

(1) $u \wedge v \wedge w = 0$.

(2) $\begin{vmatrix} u_i & v_i & w_i \\ u_j & v_j & w_j \\ u_k & v_k & w_k \end{vmatrix} = 0$ *per ogni* i, j, k *con* $1 \leq i, j, k \leq n$.

(3) $\begin{vmatrix} u_i & v_i & w_i \\ u_j & v_j & w_j \\ u_k & v_k & w_k \end{vmatrix} = 0$ *per ogni* i, j, k *con* $1 \leq i < j < k \leq n$.

Dimostrazione. Anche questa volta è chiaro che (2) e (3) sono equivalenti, perché se due dei tre indici coincidono, il corrispondente determinante si annulla per il lemma 5.5, e se invece gli indici sono tutti distinti, li possiamo scambiare per ottenere $i < j < k$ con il determinante che cambia solo di segno.

(1) $\Longrightarrow$ (2): Sia $u \wedge v \wedge w = 0$, ad esempio $w = \lambda u + \mu v$ con $\lambda, \mu \in \mathbb{R}$. Allora

$$\begin{vmatrix} u_i & v_i & w_i \\ u_j & v_j & w_j \\ u_k & v_k & w_k \end{vmatrix} = \begin{vmatrix} u_i & v_i & \lambda u_i + \mu v_i \\ u_j & v_j & \lambda u_j + \mu v_j \\ u_k & v_k & \lambda u_k + \mu v_k \end{vmatrix}$$

$$= \lambda \begin{vmatrix} u_i & v_i & u_i \\ u_j & v_j & u_j \\ u_k & v_k & u_k \end{vmatrix} + \mu \begin{vmatrix} u_i & v_i & v_i \\ u_j & v_j & v_j \\ u_k & v_k & v_k \end{vmatrix} = 0$$

come segue dal lemma 5.5.

(2) $\Longrightarrow$ (1): Tutti i determinanti al punto (2) siano nulli. Dobbiamo dimostrare che $u \wedge v \wedge w = 0$. Ciò è sicuramente vero se $u \wedge v = 0$. Possiamo quindi assumere che $u \wedge v \neq 0$. Dal teorema 31.5 segue che allora ad esempio $\begin{vmatrix} u_1 & v_1 \\ u_2 & v_2 \end{vmatrix} \neq 0$, mentre per ipotesi per ogni k con $1 \leq k \leq n$ abbiamo

$$\begin{vmatrix} u_k & v_k & w_k \\ u_1 & v_1 & w_1 \\ u_2 & v_2 & w_2 \end{vmatrix} = 0$$

Sviluppando questo determinante troviamo

$$u_k \begin{vmatrix} v_1 & w_1 \\ v_2 & w_2 \end{vmatrix} - v_k \begin{vmatrix} u_1 & w_1 \\ u_2 & w_2 \end{vmatrix} + w_k \begin{vmatrix} u_1 & v_1 \\ u_2 & v_2 \end{vmatrix} = 0$$

e ciò, valendo per ogni indice k, significa che

$$\begin{vmatrix} v_1 & w_1 \\ v_2 & w_2 \end{vmatrix} u - \begin{vmatrix} u_1 & w_1 \\ u_2 & w_2 \end{vmatrix} v + \begin{vmatrix} u_1 & v_1 \\ u_2 & v_2 \end{vmatrix} w = 0$$

una relazione lineare tra i vettori u, v e w in cui almeno il terzo coefficiente è diverso da zero. I tre vettori sono quindi linearmente dipendenti.

Corollario 31.8. *Siano $u = (u_1, u_2, u_3)$, $v = (v_1, v_2, v_3)$ e $w = (w_1, w_2, w_3)$ tre vettori di $\mathbb{R}^3$. Allora i seguenti enunciati sono equivalenti:*

(1) $u \wedge v \wedge w = 0$.

(2) $\begin{vmatrix} u_1 & v_1 & w_1 \\ u_2 & v_2 & w_2 \\ u_3 & v_3 & w_3 \end{vmatrix} = 0$.

Esercizio 31.9. Verificare che i vettori $(3, 2, 5)$, $(7, 2, 29)$ e $(1, 0, 6)$ sono linearmente dipendenti.

Il prodotto vettoriale

Situazione 32.1. $u, v, w, \ldots \in \mathbb{R}^3$, ove non indicato diversamente.

Definizione 32.2. Siano $v = (v_1, v_2, v_3)$, $w = (w_1, w_2, w_3)$ due vettori di $\mathbb{R}^3$. Allora il vettore

$$v \times w := \left(\begin{vmatrix} v_2 & w_2 \\ v_3 & w_3 \end{vmatrix}, - \begin{vmatrix} v_1 & w_1 \\ v_3 & w_3 \end{vmatrix}, \begin{vmatrix} v_1 & w_1 \\ v_2 & w_2 \end{vmatrix} \right)$$

si chiama il *prodotto vettoriale* dei vettori v e w. Dal teorema 31.5 vediamo che v e w sono linearmente dipendenti se e solo se $v \times w = 0$.

Nota 32.3. Il prodotto vettoriale $v \times w$ permette di rappresentare il prodotto esterno $v \wedge w$ di due vettori di $\mathbb{R}^3$ come un vettore dello stesso $\mathbb{R}^3$. Ciò non è possibile in altre dimensioni perché il prodotto esterno di due vettori di $\mathbb{R}^n$ è un vettore di uno spazio vettoriale reale di dimensione $\binom{n}{2}$ e solo per $n = 3$ si ha $\binom{n}{2} = n$.

Scrivendo il prodotto vettoriale come vettore colonna e calcolando esplicitamente i coefficienti abbiamo

$$v \times w = \begin{pmatrix} v_2 w_3 - v_3 w_2 \\ v_3 w_1 - v_1 w_3 \\ v_1 w_2 - v_2 w_1 \end{pmatrix}$$

Si noti il modo ciclico in cui si susseguono gli indici.

Proposizione 32.4. $\det(u, v, w) = \|u, v \times w\| = \|u \times v, w\|$.

Dimostrazione. Il primo prodotto scalare è uguale a

$$u_1 \begin{vmatrix} v_2 & w_2 \\ v_3 & w_3 \end{vmatrix} - u_2 \begin{vmatrix} v_1 & w_1 \\ v_3 & w_3 \end{vmatrix} + u_3 \begin{vmatrix} v_1 & w_1 \\ v_2 & w_2 \end{vmatrix}$$

Però questo è proprio il determinante $\det(u, v, w)$ secondo la formula di espansione data nella def. 5.1.

Inoltre

$$\begin{aligned} \|u \times v, w\| &= \|w, u \times v\| = \det(w, u, v) \\ &= -\det(u, w, v) = \det(u, v, w) \end{aligned}$$

Corollario 32.5. *Il vettore $v \times w$ è ortogonale sia a v che a w.*

Dimostrazione. Verifichiamo ad esempio che $\|v, v \times w\| = 0$.

Per la prop. 32.4 abbiamo $\|v, v \times w\| = \det(v, v, w)$. Questo determinante però si annulla, perché contiene due righe uguali.

Possiamo utilizzare la prop. 32.4 per creare una funzione in Python per il calcolo del determinante di una matrice 3×3. Usiamo che il determinante di una matrice è uguale al determinante della sua trasposta. La funzione `prodottovettoriale` è definita nella colonna accanto.

```
def det3 (A):
  u=A[0]; v=A[1]; w=A[2]
  return geom.prodottoscalare(u,prodottovettoriale(v,w))
```

Osservazione 32.6. $v \times w = -(w \times v)$.

Dimostrazione. Immediato dalla def. 32.2.

Lemma 32.7 (identità di Grassmann).

$$\begin{aligned} u \times (v \times w) &= \|u, w\| v - \|u, v\| w \\ (u \times v) \times w &= \|u, w\| v - \|v, w\| u \end{aligned}$$

Dimostrazione. Il modo più indolore per verificare queste identità è il calcolo diretto. Scriviamo i vettori come colonne.

$$\begin{aligned} u \times (v \times w) &= \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \times \begin{pmatrix} v_2 w_3 - v_3 w_2 \\ v_3 w_1 - v_1 w_3 \\ v_1 w_2 - v_2 w_1 \end{pmatrix} \\ &= \begin{pmatrix} \begin{vmatrix} u_2 & v_3 w_1 - v_1 w_3 \\ u_3 & v_1 w_2 - v_2 w_1 \end{vmatrix} \\ - \begin{vmatrix} u_1 & v_2 w_3 - v_3 w_2 \\ u_3 & v_1 w_2 - v_2 w_1 \end{vmatrix} \\ \begin{vmatrix} u_1 & v_2 w_3 - v_3 w_2 \\ u_2 & v_3 w_1 - v_1 w_3 \end{vmatrix} \end{pmatrix} \\ &= \begin{pmatrix} u_2 v_1 w_2 - u_2 v_2 w_1 - u_3 v_3 w_1 + u_3 v_1 w_3 \\ u_3 v_2 w_3 - u_3 v_3 w_2 - u_1 v_1 w_2 + u_1 v_2 w_1 \\ u_1 v_3 w_1 - u_1 v_1 w_3 - u_2 v_2 w_3 + u_2 v_3 w_2 \end{pmatrix} \\ &= \begin{pmatrix} v_1(u_2 w_2 + u_3 w_3) - w_1(u_2 v_2 + u_3 v_3) \\ v_2(u_3 w_3 + u_1 w_1) - w_2(u_3 v_3 + u_1 v_1) \\ v_3(u_1 w_1 + u_2 w_2) - w_3(u_1 v_1 + u_2 v_2) \end{pmatrix} \\ &= \begin{pmatrix} v_1(u_2 w_2 + u_3 w_3 + u_1 w_1) - w_1(u_2 v_2 + u_3 v_3 + u_1 v_1) \\ v_2(u_3 w_3 + u_1 w_1 + u_2 w_2) - w_2(u_3 v_3 + u_1 v_1 + u_2 v_2) \\ v_3(u_1 w_1 + u_2 w_2 + u_3 w_3) - w_3(u_1 v_1 + u_2 v_2 + u_3 v_3) \end{pmatrix} \\ &= \|u, w\| v - \|u, v\| w \end{aligned}$$

Questa è la prima delle due identità. Nella seconda usiamo l'antisimmetria del prodotto vettoriale (oss. 32.6):

$$\begin{aligned} (u \times v) \times w &= -w \times (u \times v) \\ &= -(\|w, v\| u - \|w, u\| v) = \|u, w\| v - \|v, w\| u \end{aligned}$$

Corollario 32.8.

$$\begin{aligned} (u \times v) \times (x \times w) &= \det(u, v, w) x - \det(u, v, x) w \\ (u \times v) \times (u \times w) &= \det(u, v, w) u \end{aligned}$$

Dimostrazione. (1) Utilizzando il lemma 32.7 e la prop. 32.4 abbiamo

$$\begin{aligned} (u \times v) \times (x \times w) &= \|u \times v, w\| x - \|u \times v, x\| w \\ &= \det(u, v, w) x - \det(u, v, x) w \end{aligned}$$

(2) Per $x = u$ si ottiene la seconda formula.

Proposizione 32.9.

$$\begin{aligned} \|x \times u, v \times w\| &= \|x, v\| \|u, w\| - \|x, w\| \|u, v\| \\ &= \det \begin{pmatrix} \|x, v\| & \|x, w\| \\ \|u, v\| & \|u, w\| \end{pmatrix} \\ |v \times w|^2 &= |v|^2 |w|^2 - \|v, w\|^2 \end{aligned}$$

Dimostrazione. (1) Usando la prop. 32.4 e la prima identità di Grassmann insieme all'esercizio 53 abbiamo

$$\begin{aligned} \|x \times u, v \times w\| &= \|x, u \times (v \times w)\| = \|x, \|u, w\| v - \|u, v\| w\| \\ &= \|x, v\| \|u, w\| - \|x, w\| \|u, v\| \end{aligned}$$

(2) Per $x = v, u = w$ otteniamo la seconda formula.

Una funzione in Python per il prodotto vettoriale:

```
def prodottovettoriale (v,w):
  (v1,v2,v3)=v; (w1,w2,w3)=w
  return [v2*w3-v3*w2,v3*w1-v1*w3,v1*w2-v2*w1]
```

Significato geometrico di $v \times w$

Nota 32.10. Due vettori linearmente indipendenti v e w in $\mathbb{R}^3$ formano, insieme all'origine, un triangolo non degenere in cui denotiamo con α l'angolo nell'origine.

Per la proposizione 32.9 e le formule a pagina 29 abbiamo

$$\begin{aligned} |v \times w|^2 &= |v|^2 |w|^2 - \|v, w\|^2 = |v|^2 |w|^2 - |v|^2 |w|^2 \cos^2 \alpha \\ &= |v|^2 |w|^2 (1 - \cos^2 \alpha) = |v|^2 |w|^2 \sin^2 \alpha \end{aligned}$$

per cui

$$|v \times w| = |v||w| |\sin \alpha|.$$

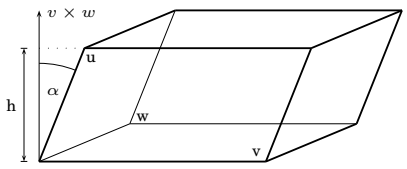
$v \times w$ è quindi un vettore ortogonale a v e w di lunghezza uguale all'area del parallelogramma racchiuso da v e w (come visto nella nota 30.4) ed è orientato in modo che i vettori v, w e $v \times w$ formino un sistema destrogiro come vedremo nella prop. 33.4.

Il volume

Nota 33.1. Un *parallelepipedo*, analogo in dimensioni superiori del parallelogramma, centrato nell'origine di $\mathbb{R}^n$ è determinato da n vettori $v_1, \dots, v_n$ e può essere descritto analiticamente come l'insieme di tutte le combinazioni lineari $\lambda_1 v_1 + \cdots + \lambda_n v_n$ in cui $0 \le \lambda_i \le 1$ per ogni $i = 1, \dots, n$.

Il suo volume è uguale al valore assoluto del determinante della matrice le cui colonne sono i vettori $v_1, \dots, v_n$ come dimostrato nella nota 30.4 per il caso $n = 2$.

Dimostriamo la formula per $n = 3$. Il parallelepipedo sia generato dai vettori u, v e w. Se questi sono linearmente dipendenti, il parallelepipedo è degenere (tutto contenuto in un piano) e il suo volume 3-dimensionale è uguale a zero così come $\det(u, v, w)$. Altrimenti abbiamo una situazione come nella figura.



È chiaro che il volume del parallelepipedo è uguale all'area del parallelogramma di base generato da v e w moltiplicato per l'altezza h. Abbiamo visto nella nota 32.10 che l'area del parallelogramma è uguale a $|v \times w|$, mentre l'altezza è uguale a

$$h = |u||\cos\alpha| = \frac{|u|||u, v \times w|||}{|u||v \times w|} = \frac{|||u, v \times w|||}{|v \times w|}$$

per cui $h|v \times w| = |||u, v \times w||| = |\det(u, v, w)|$.

Il volume del parallelepipedo generato da k vettori $v_1, \dots, v_k$ in $\mathbb{R}^n$ è uguale a

$$|v_1 \wedge \ldots \wedge v_k| = \sqrt{\det \begin{pmatrix} \|v_1, v_1\| & \dots & \|v_1, v_k\| \\ \dots & \dots & \dots \\ \|v_k, v_1\| & \dots & \|v_k, v_k\| \end{pmatrix}}$$

Questa è una delle più importanti formule della matematica! Nel calcolo delle *forme differenziali* diventa il legame tra geometria e analisi nell'analisi sulle varietà differenziabili. Verificarla per $n = k = 2$.

Orientamento

Nota 33.2. Denotiamo con $\mathbb{R}^n_m$ l'insieme delle matrici

$$A := \begin{pmatrix} a^1_1 & \dots & a^1_m \\ & \dots & \\ a^n_1 & \dots & a^n_m \end{pmatrix}$$

con n righe ed m colonne a coefficienti reali, i cui vettori colonna (che sono vettori di $\mathbb{R}^n$) verranno denotati con $a_1, \dots, a_m$ (oppure spesso con $e_1, \dots, e_n$, quando per $n = m$ formano una base).

Per $n = m$ la matrice diventa quadratica e possiamo considerare il suo determinante $\det A = \det(a_1, \dots, a_n)$.

In analogia con quanto visto nei corollari 31.6 e 31.8, questo determinante è diverso da zero se e solo i vettori $a_1, \dots, a_n$ sono linearmente indipendenti e formano quindi una base di $\mathbb{R}^n$. Ciò verrà dimostrato nel corso di Geometria.

Il determinante suddivide quindi le matrici $A \in \mathbb{R}^n_n$ in due classi: quelle matrici il cui determinante è $\neq 0$ e le cui colonne formano quindi una base di $\mathbb{R}^n_n$, e le matrici il cui determinante è uguale a zero e le cui colonne sono linearmente dipendenti.

Questa suddivisione sussiste per ogni campo di scalari al posto di $\mathbb{R}$. In $\mathbb{R}$ però possiamo distinguere le basi ancora più finemente utilizzando l'ordine sulla retta reale. Infatti un numero reale $\neq 0$ o è > 0 oppure è < 0. Il determinante può essere perciò usato per suddividere ulteriormente le matrici con determinante $\neq 0$ in quelle che hanno determinante > 0 e quelle che hanno determinante < 0.

Mentre la lineare indipendenza dei vettori colonna di una matrice non dipende dall'ordine in cui questi vettori compaiono nella matrice, perché se cambiamo l'ordine delle colonne cambiamo solo al massimo il segno del determinante, l'essere il determinante > 0 o < 0 dipende dall'ordine in cui le colonne della matrice sono elencate e se quindi queste colonne le consideriamo come componenti di una base di $\mathbb{R}^n$, dobbiamo parlare di *basi ordinate*. Perciò, quando diciamo che $e_1, \dots, e_n$ è una base ordinata di $\mathbb{R}^n$, intendiamo che fissiamo anche l'ordine in cui i vettori $e_1, \dots, e_n$ sono elencati.

Definizione 33.3. Una base ordinata $e_1, \dots, e_n$ di $\mathbb{R}^n$ si dice *positivamente orientata* se $\det(e_1, \dots, e_n) > 0$ e *negativamente orientata* quando invece $\det(e_1, \dots, e_n) < 0$.

Ogni base ordinata è o positivamente orientata oppure negativamente orientata e, se in una base ordinata scambiamo due dei suoi elementi, otteniamo una base di orientamento opposto.

Proposizione 33.4. *v e w siano due vettori linearmente indipendenti di $\mathbb{R}^3$. Allora i vettori $v, w, v \times w$ sono linearmente indipendenti e formano una base ordinata positivamente orientata.*

Dimostrazione. Abbiamo già osservato nella def. 32.2 che $v \times w \neq 0$ se e solo se, come nella nostra ipotesi, v e w sono linearmente indipendenti. Perciò

$$\det(v, w, v \times w) = \|v \times w, v \times w\| = |v \times w|^2 > 0$$

come si vede applicando la prop. 32.4.

Nota 33.5. Tentiamo adesso di dare, almeno intuitivamente, un'interpretazione geometrica dell'orientamento di una base ordinata e_1, e_2, e_3 di $\mathbb{R}^3$, limitandoci al caso che e_1 si trovi sul lato positivo dell'asse x ed e_3 sul lato positivo dell'asse z, mentre e_2 si trovi nel piano x, y.

Consideriamo prima il caso che e_2 si trovi sul lato positivo dell'asse y. In questo caso $e_1 = (\lambda, 0, 0)$, $e_2 = (0, \mu, 0)$ ed $e_3 = (0, 0, \nu)$ con $\lambda, \mu, \nu > 0$ e

$$\det(e_1, e_2, e_3) = \begin{vmatrix} \lambda & 0 & 0 \\ 0 & \mu & 0 \\ 0 & 0 & \nu \end{vmatrix} = \lambda\mu\nu > 0$$

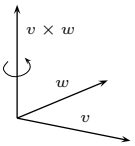
Quindi in questo caso la base ordinata e_1, e_2, e_3 è positivamente ordinata. Facciamo adesso ruotare e_2 nel piano x, y ponendo

$$e_2 = e_2(t) = (\mu\cos t, \mu\sin t, 0)$$

per $0 \le t < 360°$. Allora per il determinante abbiamo

$$\det(e_1, e_2(t), e_3) = \begin{vmatrix} \lambda & \mu\cos t & 0 \\ 0 & \mu\sin t & 0 \\ 0 & 0 & \nu \end{vmatrix} = \lambda\mu\nu\sin t$$

e vediamo che la base ordinata e_1, e_2, e_3 rimane positivamente orientata per $0 < t < 180°$ ed è invece negativamente orientata per $180° < t < 360°$. Per $t = 0$ oppure $t = 180°$ il vettore e_2 è parallelo ad e_1 e quindi non abbiamo più una base.



Questa considerazione, purché incompleta, descrive comunque la situazione nel caso di una base della forma $v, w, v \times w$ che, come abbiamo visto, è positivamente orientata e che, mediante una rotazione di $\mathbb{R}^3$, può sempre essere portata nella posizione appena descritta - si imparerà nel corso di Geometria che una rotazione lascia invariante il determinante di una base.

Proposizione 33.6 (identità di Jacobi).

$$u \times (v \times w) + v \times (w \times u) + w \times (u \times v) = 0$$

Dimostrazione. Esercizio 56.

Osservazione 33.7. Il prodotto vettoriale, considerato come operazione binaria su $\mathbb{R}^3$, è bilineare, antisimmetrico e non associativo, perché dalle identità di Grassmann si vede che in genere

$$u \times (v \times w) \neq (u \times v) \times w.$$

Le identità di Jacobi sono un'importante alternativa alla legge associativa in un anello: $(\mathbb{R}^3, +, \times)$ è un'*algebra di Lie*.

Le idee di Carl Gustav Jacob Jacobi (1804-1851), Hermann Grassmann (1809-1977) e Marius Sophus Lie (1842-1899) sono ancora oggi importanti in molti rami della matematica.

Esercizio per gli scritti

56. Dimostrare l'identità di Jacobi (prop. 33.6).

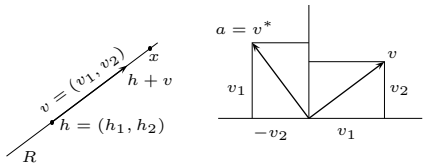
IX. GEOMETRIA ANALITICA CON PYTHON

Rette nel piano

Nota 34.1. Una retta R nel piano reale $\mathbb{R}^2$ possiede una rappresentazione parametrica

$$R = \{h + tv \mid t \in \mathbb{R}\},$$

dove $h, v \in \mathbb{R}^2$ con $v \neq 0$.



Il vettore magico $a = (a_1, a_2) := v^* = (-v_2, v_1)$ che si ottiene ruotando v di $90°$ è anch'esso diverso da zero. Un punto $x = (x_1, x_2)$ appartiene alla retta se e solo se il vettore $x - h$ è parallelo a v, cioè se e solo se $x - h$ è ortogonale ad a. Quindi i punti della retta sono esattamente i punti x che soddisfano l'equazione $\|v^*, x - h\| = 0$, ovvero

$$a_1(x_1 - h_1) + a_2(x_2 - h_2) = 0, \qquad (*)$$

che può essere scritta anche nella forma

$$a_1x_1 + a_2x_2 = a_1h_1 + a_2h_2 \text{ oppure } \|a, x\| = \|a, h\|.$$

Siccome però a_1 e a_2 non sono entrambi zero, per ogni $c \in \mathbb{R}$ si trovano facilmente h_1 e h_2 tali che $a_1h_1 + a_2h_2 = c$. Ciò mostra che ogni equazione della forma $a_1x_1 + a_2x_2 = c$ può essere portata nella forma (*) e descrive perciò una retta ortogonale ad a e quindi parallela a v. La retta $a_1x_1 + a_2x_2 = 0$ è tra queste rette tutte parallele ad R quella che passa per l'origine.

Troviamo ad esempio una rappresentazione parametrica per la retta $3x + 5y = 18$. Per $y = 0$ troviamo $x = 6$, quindi $h = (6, 0)$ è un punto della retta che deve essere parallela a $v = (5, -3)$.

Se x ed y sono due punti distinti di $\mathbb{R}^2$, una rappresentazione parametrica e l'equazione della retta passante per questi due punti si trovano ponendo $v := y - x$.

Raccogliamo nel file *geom.py* le nostre funzioni geometriche. Anche le funzioni `prodottoscalare` e `lun` definite a pagina 29 fanno parte di questo file. Aggiungiamo per prima cosa una funzione che calcola il vettore magico z^* di un punto z del piano:

```
def magico (z): (x,y)=z; return [-y,x]
```

Rette in uno spazio vettoriale

Definizione 34.2. In uno spazio vettoriale reale V, in particolare in $V = \mathbb{R}^n$, usiamo le abbreviazioni, per $h, v \in V$,

$$\mathbb{R}v := \{tv \mid t \in \mathbb{R}\}$$

$$h + \mathbb{R}v := \{h + tv \mid t \in \mathbb{R}\}$$

Gli insiemi della prima forma per cui $v \neq 0$ sono quindi esattamente le rette in $\mathbb{R}^n$ passanti per l'origine, mentre ogni retta può essere scritta nella seconda forma con $v \neq 0$. Per h si può usare un punto qualsiasi della retta.

La retta che congiunge due punti x ed y distinti in $\mathbb{R}^n$ è l'insieme $x + \mathbb{R}(y - x)$. Se i due punti non sono distinti, questo insieme non è una retta, ma contiene soltanto l'unico punto dato.

Se come parametri usiamo solo i valori $t \in [0, 1]$, invece della retta otteniamo il segmento di retta che congiunge i due punti.

Nota 34.3. Una retta R in $\mathbb{R}^n$ può quindi essere rappresentata nella forma

$$R = \{h + tv \mid t \in \mathbb{R}\} = h + \mathbb{R}v$$

con $h, v \in \mathbb{R}^n$ e $v \neq 0$, equivalente alla rappresentazione parametrica (in cui t varia su tutto $\mathbb{R}$)

$$x_1 = h_1 + tv_1$$
$$\dots$$
$$x_n = h_n + tv_n$$

se $v = (v_1, \dots, v_n)$ e $h = (h_1, \dots, h_n)$.

In $\mathbb{R}^2$ possiamo scrivere $R = z_0 + \mathbb{R}w$ e, nella rappresentazione parametrica,

$$x = x_0 + tu$$
$$y = y_0 + tv$$

se $w = (u, v)$ e $z_0 = (x_0, y_0)$.

Osservazione 34.4. Due rette $R = h + \mathbb{R}v$ ed $S = g + \mathbb{R}w$ in uno spazio vettoriale reale sono parallele se e solo se i vettori v e w sono paralleli, cioè se e solo se esiste $\lambda \in \mathbb{R}$ (necessariamente $\neq 0$, perché sia v che w devono essere $\neq 0$) tale che $w = \lambda v$, cioè se e solo se $v \wedge w = 0$. In questo caso le due rette coincidono se e solo se $g \in R$, cioè se e solo se $g - h$ è un multiplo di v e allora $R = S = R \cap S$. Altrimenti $R \cap S = \emptyset$.

Lemma 34.5. *Siano dati tre vettori a, b, c in uno spazio vettoriale reale. Allora:*

(1) Se c è combinazione lineare di a e b, allora $a \wedge b \wedge c = 0$.

(2) Se $a \wedge b \neq 0$ (cioè se a e b sono linearmente indipendenti) e $a \wedge b \wedge c = 0$, allora c è combinazione lineare di a e b.

Dimostrazione. (1) Chiaro.

(2) $a \wedge b \wedge c = 0$ implica che esiste una relazione $\lambda a + \mu b + \nu c = 0$ con coefficienti $\lambda, \mu, \nu \in \mathbb{R}$ non tutti nulli. Se $\nu = 0$, allora $(\lambda, \mu) \neq (0, 0)$ e $\lambda a + \mu b = 0$ (perché in quel caso anche $\nu c = 0$), in contraddizione all'indipendenza lineare di a e b. Quindi $\nu \neq 0$. Allora però possiamo scrivere $c = -\dfrac{\lambda}{\nu}a - \dfrac{\mu}{\nu}b$ e vediamo che c è una combinazione lineare di a e b.

Prepariamo alcune funzioni in Python per le operazioni vettoriali elementari in $\mathbb{R}^n$:

```
# x+y.
def add (x,y): return [a+b for a,b in zip(x,y)]

# x-y.
def diff (x,y): return [a-b for a,b in zip(x,y)]

# tv.
def mul (t,v): return [t*a for a in v]
```

Esse saranno contenute nel file *geom.py*.

Intersezione di due rette nel piano

Nota 34.6. Siano date due rette $R = h + \mathbb{R}v$ ed $S = g + \mathbb{R}w$ nel piano $\mathbb{R}^2$. Se le rette non sono parallele, i vettori $v = (v_1, v_2)$ e $w = (w_1, w_2)$ sono linearmente indipendenti e per trovare l'intersezione $R \cap S$ consideriamo l'equazione vettoriale $h + tv = g + sw$ nelle incognite t ed s, costituita da due equazioni scalari

$$v_1t - w_1s = g_1 - h_1$$
$$v_2t - w_2s = g_2 - h_2$$

il cui determinante $\begin{vmatrix} v_1 & -w_1 \\ v_2 & -w_2 \end{vmatrix}$ è diverso da zero (cor. 31.6) e che quindi possiede esattamente una soluzione (t, s) da cui otteniamo l'unico punto $h + tv$ dell'intersezione delle due rette.

Quindi due rette non parallele nel piano $\mathbb{R}^2$ si intersecano esattamente in un punto.

L'intersezione di due rette date tramite equazioni $a_1x_1 + a_2x_2 = c$ e $b_1x_1 + b_2x_2 = d$ può essere calcolata anche risolvendo direttamente il sistema lineare

$$\begin{aligned} a_1x_1 + a_2x_2 &= c \\ b_1x_1 + b_2x_2 &= d \end{aligned}$$

nelle incognite x_1 e x_2.

Due rette in $\mathbb{R}^3$

Nota 35.1. Siano date due rette $R = h + \mathbb{R}v$ ed $S = g + \mathbb{R}w$ (con $v, w \neq 0$) nello spazio $\mathbb{R}^3$, non necessariamente parallele.

Per trovare l'intersezione $R \cap S$ dobbiamo risolvere l'equazione vettoriale $h + tv = g + sw$ nelle incognite scalari t ed s, equivalente all'equazione $tv - sw = g - h$, dalla quale si vede che $R \cap S \neq \emptyset$ se e solo se il vettore $g - h$ è combinazione lineare di v e w.

(1) v e w siano linearmente dipendenti: In questo caso $g - h$ è combinazione linerare di v e w se e solo se è un multiplo di v, e siccome $h \in R$, ciò accade se e solo $g \in R$.

Quindi $R \cap S = R = S$ oppure $R \cap S = \emptyset$ come già visto prima.

(2) v e w siano linearmente indipendenti: In questo caso, per il lemma 34.5, $g - h$ è combinazione lineare di v e w se e solo se $v \wedge w \wedge (g - h) = 0$ e quindi, per il cor. 31.8, se e solo se

$$\begin{vmatrix} v_1 & w_1 & g_1 - h_1 \\ v_2 & w_2 & g_2 - h_2 \\ v_3 & w_3 & g_3 - h_3 \end{vmatrix} = 0$$

(sempre in questa ipotesi però, che v e w siano linearmente indipendenti).

Assumiamo che ciò accada e che $h + tv = g + sw$ sia un punto dell'intersezione $R \cap S$. Possono essercene altri?

Per una coppia di numeri reali t', s' sono allora equivalenti:

$$\begin{aligned} h + t'v &= g + s'w \\ g + sw - tv + t'v &= g + s'w \\ (t' - t)v &= (s' - s)w \end{aligned}$$

v e w sono linearmente indipendenti, perciò $(t' - t)v = (s' - s)w$ implica $t' = t$ e $s' = s$. Quindi le due rette si intersecano in un solo punto. Il seguente teorema riassume ciò che abbiamo finora dimostrato.

Teorema 35.2. *In $\mathbb{R}^3$ siano date due rette $R = h + \mathbb{R}v$ ed $S = g + \mathbb{R}w$.*

(1) Se i due vettori v e w sono linearmente indipendenti, allora le rette R e S si intersecano se e solo se

$$\begin{vmatrix} v_1 & w_1 & g_1 - h_1 \\ v_2 & w_2 & g_2 - h_2 \\ v_3 & w_3 & g_3 - h_3 \end{vmatrix} = 0$$

e in questo caso si intersecano in un solo punto.

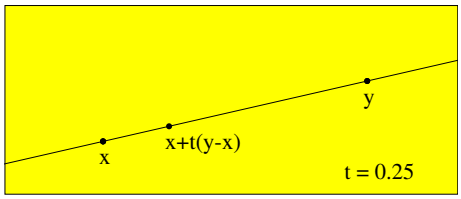
(2) Se i due vettori v e w sono linearmente dipendenti, allora le due rette sono parallele e si intersecano se e solo se $g - h$ è un multiplo di v e in quest'ultimo caso le due rette coincidono.

Coordinate baricentriche su una retta

Abbiamo visto che la retta che congiunge due punti x ed y distinti in $\mathbb{R}^n$ è l'insieme

$$x + \mathbb{R}(y - x) = \{x + t(y - x) \mid t \in \mathbb{R}\}$$

Questo insieme è definito anche nel caso che $y = x$ e coincide in tal caso con il punto x. Per $y \neq x$ il parametro t per ogni punto p della forma $p = x + t(y - x)$ è univocamente determinato e si chiama la *coordinata baricentrica* di p rispetto ad x ed y; viceversa, anche quando $x = y$, ogni $t \in \mathbb{R}$ definisce naturalmente un unico punto $p = x + t(y - x)$.



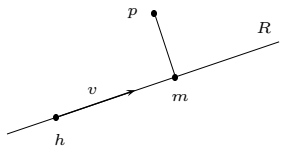
Per $t = 0$ si ottiene il punto x, per $t = 1$ il punto y, per $t = 1/2$ il baricentro $\dfrac{x + y}{2}$ dei punti x ed y.

Per ottenere il punto $x + t(y - x)$ usiamo la seguente funzione in Python:

```
# x+t(y-x).
def bari (x,y,t): return add(x,mul(t,diff(y,x)))
```

Proiezione su una retta in $\mathbb{R}^n$

Nota 35.3. Siano dati una retta $R = h + \mathbb{R}v$ in $\mathbb{R}^n$ (con $v \neq 0$) e un punto $p \in \mathbb{R}^n$. Calcoliamo la proiezione ortogonale m di p su R.



Il punto m deve essere in primo luogo un punto della retta e quindi della forma $m = h + tv$, inoltre il vettore $p - m$ deve essere ortogonale a v, cioè $\|p - m, v\| = 0$, ossia

$$\|p, v\| = \|m, v\| = \|h + tv, v\| = \|h, v\| + t\|v, v\|$$

Siccome $v \neq 0$, ciò è equivalente a $t = \dfrac{\|p, v\| - \|h, v\|}{\|v, v\|}$

e quindi abbiamo la formula fondamentale

$$t = \frac{\|p - h, v\|}{\|v, v\|}$$

Da essa otteniamo la proiezione con $m = h + \dfrac{\|p - h, v\|}{\|v, v\|} v$.

Quando $|v| = 1$ (ciò si può sempre ottenere sostituendo v con $v/|v|$), abbiamo la rappresentazione $m = h + \|p - h, v\|v$.

Dalla derivazione si vede anche che t e con esso m sono univocamente determinati. La distanza di p dalla retta è uguale a $|p - m|$.

Tutto ciò è valido in $\mathbb{R}^n$ per ogni n.

È geometricamente chiaro e facile da dimostrare che $m = p$ se p è un punto della retta.

In Python possiamo realizzare la proiezione con la funzione

```
# Proiezione di p sulla retta h+Rv.
def prosuretta (h,v,p):
  t=prodottoscalare(diff(p,h),v)/float(prodottoscalare(v,v))
  return add(h,mul(t,v))
```

Proiezione su una retta nel piano

Nota 35.4. Nel piano le formule per la proiezione di un punto su una retta possono essere formulate in maniera più esplicita. Come nella nota 34.1 siano $v = (v_1, v_2)$ e $a = (a_1, a_2)$ con $a_1 = -v_2$, $a_2 = v_1$. La retta può essere rappresentata in forma parametrica come $R = h + \mathbb{R}v$ oppure tramite l'equazione

$$a_1x_1 + a_2x_2 = c \text{ oppure } \|a, x\| = \|a, h\|$$

con $c = \|a, h\|$. I vettori a e v sono ortogonali tra di loro e hanno la stessa lunghezza. Possiamo calcolare

$$m = h + \frac{\|p - h, v\|}{|v|^2} v$$

come nel caso generale di $\mathbb{R}^n$; nel piano vediamo però che m deve essere anche della forma $m = p + sa$ per qualche $s \in \mathbb{R}$ il cui valore può essere trovato utilizzando l'equazione $\|a, m\| = c$ che m come punto della retta deve soddisfare. Quindi

$$c = \|a, m\| = \|a, p\| + s\|a, a\| = \|a, p\| + s|a|^2,$$

per cui

$$s = \frac{c - \|a, p\|}{|a|^2} = \frac{\|a, h\| - \|a, p\|}{|a|^2} = \frac{\|a, h - p\|}{|a|^2}$$

e quindi

$$m = p + \frac{c - \|a, p\|}{|a|^2} a$$

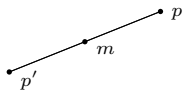
Adesso otteniamo facilmente la distanza di p dalla retta, infatti questa distanza coincide con la lunghezza del vettore $p - m$:

$$|p - m| = \frac{|\|a, p\| - c|}{|a|} = \frac{|a_1p_1 + a_2p_2 - c|}{\sqrt{a_1^2 + a_2^2}}$$

Riflessione in un punto

Nota 36.1. Siano p ed m due punti in $\mathbb{R}^n$. Diciamo che un punto p' è la riflessione di p in m, se m è il punto di dimezzamento del segmento tra p e p', cioè se

$$m = \frac{p + p'}{2}$$



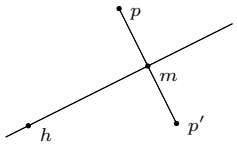
Ciò implica che p' è univocamente determinato con $p' = 2m - p$.

In Python usiamo la funzione

```
# Riflessione di p in m.
def riflinpunto (m,p): return diff(mul(2,m),p)
```

Riflessione in una retta

Siano p un punto ed R una retta in $\mathbb{R}^n$. Diciamo che un punto p' è la riflessione di p in R, se p' è la riflessione di p nella proiezione m di p sulla retta.



Possiamo facilmente combinare le funzioni `prosuretta` e `riflinpunto` per realizzare questa operazione in Python:

```
# Riflessione di p nella retta h+Rv.
def riflinretta (h,v,p):
  return riflinpunto(prosuretta(h,v,p),p)
```

Rotazione attorno a un punto nel piano

Abbiamo visto a pagina 13 che il punto p' che si ottiene da un punto $p = (x, y)$ del piano mediante una rotazione attorno all'origine per l'angolo α (in senso antiorario) è dato da

$$p' = (x\cos\alpha - y\sin\alpha, x\sin\alpha + y\cos\alpha) =: f_\alpha(p)$$

Se la rotazione avviene invece attorno ad un altro centro m, otteniamo il punto $m + f_\alpha(p - m)$.

In Python usiamo la seguente funzione:

```
# Rotazione di p attorno al centro m.
# L'angolo alfa e' indicato in gradi.
def rot (p,alfa,m=None):
  if m==None: m=(0,0)
  c=trigo.cosgradi(alfa); s=trigo.singradi(alfa)
  (x,y)=diff(p,m)
  return add(m,[x*c-y*s,x*s+y*c])
```

Piani nello spazio

Nota 36.2. Un piano P in uno spazio vettoriale reale V possiede una rappresentazione

$$P = h + \mathbb{R}v + \mathbb{R}w$$

con $h, v, w \in V$ e dove v e w sono linearmente indipendenti. h appartiene a P e può essere sostituito da qualsiasi altro punto del piano.

Un punto $x \in V$ appartiene al piano P se e solo se $x - h$ è combinazione lineare di v e w, quindi, essendo v e w linearmente indipendenti, se e solo se

$$(x - h) \wedge v \wedge w = 0$$

come segue dalla nota 35.1.

Assumiamo adesso che $V = \mathbb{R}^3$. In questo caso per il cor. 31.8 la condizione $(x - h) \wedge v \wedge w = 0$ è equivalente a

$$\det(x - h, v, w) = 0$$

Dalla proposizione 32.4 sappiamo però che

$$\det(x - h, v, w) = \|x - h, v \times w\|$$

e quindi x appartiene al piano se e solo se $\|x - h, v \times w\| = 0$. Ciò dal lato geometrico significa che x appartiene a P se e solo se $x - h$ è ortogonale al prodotto vettoriale $v \times w$, mentre dal lato analitico ci fornisce anche un'equazione che descrive il piano:

$$\|x, v \times w\| = \|h, v \times w\|$$

Se rappresentiamo i vettori tramite i loro componenti,

$$\begin{aligned} x &= (x_1, x_2, x_3) \\ h &= (h_1, h_2, h_3) \\ v &= (v_1, v_2, v_3) \\ w &= (w_1, w_2, w_3) \end{aligned}$$

questa equazione è un'equazione scalare nelle tre incognite x_1, x_2, x_3:

$$a_1x_1 + a_2x_2 + a_3x_3 = c$$

con

$$\begin{aligned} a_1 &= v_2w_3 - v_3w_2 \\ a_2 &= v_3w_1 - v_1w_3 \\ a_3 &= v_1w_2 - v_2w_1 \\ c &= \|h, v \times w\| \end{aligned}$$

che possiamo scrivere nella forma

$$\|a, x\| = \|a, h\|$$

in analogia con quanto abbiamo visto nella nota 35.4 per la retta nel piano, con $a := (a_1, a_2, a_3) = v \times w$.

Insieme alle note 32.10 e 33.5 ciò mostra anche che il prodotto vettoriale può essere considerato come l'analogo 3-dimensionale del vettore magico.

Nota 36.3. Sia viceversa data un'equazione della forma

$$a_1x_1 + a_2x_2 + a_3x_3 = c$$

con $a := (a_1, a_2, a_3) \neq (0, 0, 0)$. Anche stavolta è facile trovare un punto $h := (h_1, h_2, h_3)$ tale che $c = a_1h_1 + a_2h_2 + a_3h_3$, e allora l'equazione può essere scritta nella forma $\|a, x\| = \|a, h\|$, ossia

$$\|a, x - h\| = 0$$

L'insieme delle soluzioni di questa equazione consiste quindi di tutti i punti x per cui il vettore $x - h$ è ortogonale al vettore a. Ciò intuitivamente dimostra già che l'insieme delle soluzioni forma un piano, per il quale comunque possiamo facilmente trovare una rappresentazione parametrica:

Sia ad esempio $a_3 \neq 0$, allora l'equazione $a_1x_1 + a_2x_2 + a_3x_3 = c$ è equivalente (cfr. pag. 5) a

$$\frac{a_1}{a_3}x_1 + \frac{a_2}{a_3}x_2 + x_3 = \frac{c}{a_3}$$

cioè a

$$x_3 = \frac{c}{a_3} - \frac{a_1}{a_3}x_1 - \frac{a_2}{a_3}x_2$$

Per ogni scelta di x_1 e x_2 otteniamo un valore di x_3 tale che $x := (x_1, x_2, x_3)$ sia una soluzione dell'equazione e viceversa, quindi le soluzioni sono esattamente i punti rappresentabili nella forma

$$\begin{aligned} x_1 &= t \\ x_2 &= s \\ x_3 &= \frac{c}{a_3} - \frac{a_1}{a_3}t - \frac{a_2}{a_3}s \end{aligned}$$

per $t, s \in \mathbb{R}$ e quindi, se poniamo

$$\begin{aligned} h &:= (0, 0, \frac{c}{a_3}) \\ v &:= (1, 0, -\frac{a_1}{a_3}) \\ w &:= (0, 1, -\frac{a_2}{a_3}) \end{aligned}$$

le soluzioni costituiscono esattamente il piano

$$P = h + \mathbb{R}v + \mathbb{R}w.$$

Proiezione di un punto su un piano

Nota 37.1. Siano dati un piano P e un punto p di $\mathbb{R}^3$. Il piano sia descritto dall'equazione $\|a, x\| = c$.

Vogliamo calcolare la proiezione ortogonale m di p su P. Il vettore $p - m$ deve essere ortogonale al piano e quindi parallelo al vettore a, per cui $m = p + sa$ per qualche valore $s \in \mathbb{R}$ che otteniamo dall'equazione $(a, m) = c$ che m come punto del piano deve soddisfare. Quindi

$$c = \|a, m\| = \|a, p + sa\| = \|a, p\| + s\|a, a\| = \|a, p\| + s|a|^2$$

per cui $s = \dfrac{c - \|a, p\|}{|a|^2}$ e quindi $m = p + \dfrac{c - \|a, p\|}{|a|^2} a$.

Si noti la completa analogia con la formula derivata nella nota 35.4 per la proiezione di un punto su una retta nel piano. Cos'è che hanno in comune le due situazioni?

La ragione è che entrambe le volte calcoliamo la proiezione ortogonale di un punto su un insieme descritto da un'equazione della forma

$$\|a, x\| = c$$

Infatti, assumiamo che siano dati un vettore $a = (a_1, \ldots, a_n) \neq 0$ di $\mathbb{R}^n$ e un'equazione $\|a, x\| = c$ per qualche $c \in \mathbb{R}$.

Nel corso di Geometria si imparerà che una tale equazione descrive un *iperpiano* di $\mathbb{R}^n$, cioè un sottospazio affine E di dimensione $n - 1$. Ciò implica che anche in questo caso, se scegliamo un punto arbitrario h dell'iperpiano (cioè un punto h tale che $\|a, h\| = c$), E consista esattamente di quei punti x per chi il vettore $x - h$ è ortogonale ad a e che viceversa un vettore b è ortogonale all'iperpiano se e solo se è parallelo ad a (qui entra l'ipotesi che la dimensione di E sia $n - 1$). A questo punto il calcolo e le formule coincidono con quanto ottenuto per $\mathbb{R}^2$ e $\mathbb{R}^3$: La proiezione ortogonale m di un punto p su E è data da

$$m = p + \frac{c - \|a, p\|}{|a|^2} a$$

la distanza di p dall'iperpiano, cioè la lunghezza di $p - m$, da

$$|p - m| = \frac{|\|a, p\| - c|}{|a|} = \frac{|a_1 p_1 + \cdots + a_n p_n - c|}{\sqrt{a_1^2 + \cdots + a_n^2}}$$

In Python useremo le seguenti funzioni:

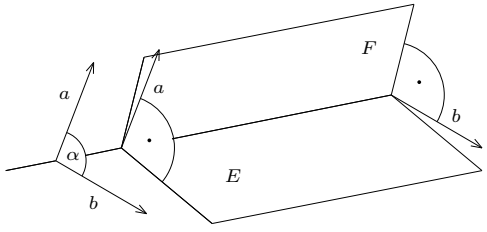
```
# Proiezione di p sull'iperpiano ||a,x||=c.
def prosuiperpiano (a,c,p):
  v=mul((c-prodottoscalare(a,p))/float(prodottoscalare(a,a)),a)
  return add(p,v)

# Distanza dall'iperpiano ||a,x||=c.
def distdaiperpiano (a,c,p):
  return abs(prodottoscalare(a,p)-c)/float(lun(a))
```

L'angolo tra due iperpiani

Nota 37.2. E ed F siano due iperpiani di $\mathbb{R}^n$ che si intersecano in un punto h. Allora esistono vettori $a, b \in \mathbb{R}^n \setminus 0$ tali che

$$E = \{x \in \mathbb{R}^n \mid \|x - h, a\| = 0\} = \{x \in \mathbb{R}^n \mid \|x, a\| = c_1\}$$
$$F = \{x \in \mathbb{R}^n \mid \|x - h, b\| = 0\} = \{x \in \mathbb{R}^n \mid \|x, b\| = c_2\}$$

con $c_1 := \|h, a\|$ e $c_2 := \|h, b\|$. Come angolo tra E ed F possiamo considerare l'angolo α tra a e b o anche $180^\circ - \alpha$.



Il coseno di α non cambia se invece di α usiamo $180^\circ - \alpha$ ed è uguale a $\cos\alpha = \dfrac{\|a, b\|}{|a||b|}$.

Il piano passante per tre punti

Nota 37.3. p, q ed r siano tre punti di uno spazio vettoriale reale V. I vettori $q - p$ ed $r - p$ sono linearmente indipendenti se e solo se i punti p, q ed r non stanno sulla stessa retta (esercizio 57). In questo caso essi generano un piano

$$P = p + \mathbb{R}(q - p) + \mathbb{R}(r - p)$$

a cui appartengono p, $q = p + (q - p)$ ed $r = p + (r - p)$.

Esercizi per gli scritti

57. In uno spazio vettoriale i vettori $q - p$ ed $r - p$ sono linearmente indipendenti se e solo se i punti p, q ed r non stanno sulla stessa retta.

58. Siano $x, y \in \mathbb{R}^n$ ed $s \in \mathbb{R}$. Allora $sx + (1 - s)y = x + t(y - x)$ per un $t \in \mathbb{R}$. Per quale t?

59. Siano a, b e t numeri reali e $t \in [0, 1]$. Allora $ta + (1-t)b \in [a, b]$.

60. Siano $x, y \in \mathbb{R}^n$, $t \in \mathbb{R}$ e $p := 0.3x + 0.7y$. Esprimere la distanza di p da x e la distanza di p da y in termini di $|x - y|$.

61. Trovare l'equazione di una delle due rette parallele alla retta $R = (1, 2) + \mathbb{R}(3, 4)$ che hanno distanza 5 da R.

62. Trovare l'equazione della retta passante per i punti $(2, 5)$ e $(1, 7)$.

63. Trovare l'equazione della retta passante per i punti $(-4, 1)$ e $(11, 4)$.

64. Trovare l'intersezione delle rette $(1, 0) + \mathbb{R}(3, 1)$ e $(2, 5) + \mathbb{R}(1, 4)$ nel piano con il primo metodo indicato nella nota 34.6.

65. Trovare le equazioni per le rette dell'esercizio 64 e determinare il punto di intersezione risolvendo un sistema nelle incognite x_1 ed x_2.

66. Trovare la proiezione ortogonale del punto $(2, 3)$ sulla retta $(-1, 0) + \mathbb{R}(4, 1)$ e calcolare la distanza del punto dalla retta.

67. Calcolare la proiezione ortogonale dell'origine sulla retta $y = 3x + 4$ e la distanza dell'origine dalla retta.

68. Trovare l'equazione del piano passante per $(0, 1, 2)$, $(1, 2, 3)$ e $(4, 8, 5)$.

69. Trovare una forma parametrica per il piano con equazione $3x + 5y + 11z = 7$.

70. Calcolare il volume del parallelepipedo generato da $(1, 2, 3)$, $(2, 1, 5)$, $(4, 1, 2)$.

71. Calcolare in $\mathbb{R}^5$ la proiezione ortogonale di $(8, 6, 5, 11, 2)$ sulla retta $(1, 5, 2, 3, 4) + \mathbb{R}(2, 1, 0, 1, 5)$.

72. Calcolare la proiezione ortogonale di $(4, 1, 7)$ sulla rette che congiunge i punti $(3, 5, 2)$ e $(8, 1, 5)$.

73. Calcolare la riflessione del punto $(3, 5)$ nella retta che congiunge $(7, 3)$ e $(1, 4)$.

74. Proiettare il punto $(3, 0, 5)$ sul piano con equazione $7x + 3y + z = 4$.

75. Calcolare la distanza del punto $(3, 2, 1)$ dal piano con equazione $4x + 2y + 3z = 1$.

76. Proiettare il punto $(4, 1, 3, 1) \in \mathbb{R}^4$ sull'iperpiano con equazione $6x_1 + x_2 + x_3 + 2x_4 = 5$ e calcolare la distanza del punto dall'iperpiano.

77. Ruotare il punto $(6, 2)$ di 50° attorno al punto $(12, 1)$.

78. Calcolare l'angolo tra gli iperpiani con equazioni $2x_1 + x_2 + 7x_3 + 2x_4 = 6$ e $x_1 + 3x_2 + 5x_3 - x_4 = 10$ in $\mathbb{R}^4$.

X. FUNZIONI PER MATRICI

Il prodotto matriciale

Abbiamo già creato (pagina 28) un file *matrici.py* che contiene le nostre funzioni per le matrici. Come finora, matrici vengono rappresentate come liste di liste: Se A è una matrice $n \times m$, essa è rappresentata nella forma $A = [A^1, \ldots, A^n]$, dove $A^i = [A^i_1, \ldots, A^i_m]$ è la i-esima riga. `len(A)` è perciò il numero delle righe di `A`.

Questa notazione, che usiamo in Python, non deve essere confusa con la notazione matematica $A = (v_1, \ldots, v_m)$, in cui $v_1, \ldots, v_m$ sono le colonne di A.

Le funzioni per matrici che qui presentiamo naturalmente non sono ottimizzate per quanto riguarda efficienza e precisione numerica. Nell'analisi numerica si studiano algoritmi molto migliori. D'altra parte è comodo avere disponibili velocemente funzioni per semplici compiti di calcolo matriciale.

Otteniamo la trasposta di una matrice con

```
def trasposta (A): return map(list,apply(zip,A))
```

Definizione 38.1. Il prodotto matriciale fv di un vettore riga $f = (f_1, \ldots, f_n)$ con un vettore colonna $\begin{pmatrix} v^1 \\ \ldots \\ v^n \end{pmatrix}$ è definito come $f_1 v^1 + \ldots + f_n v^n$. Se $f^t = \begin{pmatrix} f_1 \\ \ldots \\ f_n \end{pmatrix}$ è il vettore colonna con gli stessi coefficienti di f (cioè la trasposta di f), allora $fv = \|f^t, v\|$.

Il prodotto AB di due matrici A e B si ottiene mettendo nella i-esima riga di AB i prodotti della i-esima riga di A con le colonne di B. Il prodotto matriciale è un'applicazione $\mathbb{R}^n_p \times \mathbb{R}^p_m \longrightarrow \mathbb{R}^n_m$. Si vede facilmente che $(AB)^i_j = \sum_{k=1}^{p} A^i_k B^k_j$ per ogni i, j.

Per matrici 2×2 il prodotto coincide con quello definito nell'esercizio 39.

In Python possiamo usare la funzione

```
def mul (A,B):
  C=[]; Bt=trasposta(B)
  for r in A:
    rigaC=[]
    for c in Bt: rigaC.append(geom.prodottoscalare(r,c))
    C.append(rigaC)
  return C
```

Nella notazione matematica un vettore riga e un vettore colonna possono essere considerati come casi speciali di matrici. In Python invece matrici sono doppie liste, vettori liste semplici. Per prodotti della forma Av o fA usiamo perciò le seguente funzioni:

```
# Ax.
def mulmatvet (A,x):
  y=[]
  for r in A: y.append(geom.prodottoscalare(r,x))
  return y

# xA.
def mulvetmat (x,A):
  y=[]; At=trasposta(A)
  for c in At: y.append(geom.prodottoscalare(x,c))
  return y
```

Triangolarizzazione con il metodo di Gauss

Nota 38.2. Nella prima parte dell'algoritmo di eliminazione di Gauss una matrice viene portata in forma triangolare superiore. Nel calcolo a mano non lo abbiamo fatto, ma nel calcolo automatico bisogna cercare in ogni sezione il coefficiente massimale (di valore assoluto) tra i coefficienti iniziali delle righe ancora da elaborare. Questo coefficiente è detto *perno* (in inglese *pivot*).

```
def perno (A,j):
  B=A[j:]; A=A[0:j]
  B.sort(key=lambda r: abs(r[j]),reverse=1)
  A.extend(B); return A
```

Nota 38.3. Adesso possiamo programmare la triangolarizzazione. Se nessuno dei coefficienti iniziali rimasti è $\neq 0$, l'algoritmo si ferma e restituisce `None`. Ciò non accade (a parte errori di precisone) se la matrice è invertibile.

```
def triangolare (A):
  A=copy.deepcopy(A); n=len(A)
  for j in xrange(0,n):
    A=perno(A,j)
    Aj=A[j]; ajj=float(Aj[j])
    if ajj==0: return None
    for i in xrange(j+1,n):
      A[i]=geom.diff(A[i],geom.mul(A[i][j]/ajj,Aj))
  return A
```

Nota 38.4. Nella funzione `triangolare`, a differenza dal calcolo a mano, abbiamo usato sostituzioni della forma $A^i \longmapsto A^i - \alpha A^j$. Per la prop. 5.7 il determinante non cambia con queste operazioni. Nel corso di Geometria si impara che il determinante di una matrice (quadratica) triangolare è uguale al prodotto degli elementi nella diagonale principale. Possiamo quindi usare questi ragionamenti per scrivere una funzione per il determinante in Python. Se la triangolarizzazione dà il risultato `None`, ciò significa che un perno è risultato uguale a zero e (come si vede facilmente) ciò implica che il determinante è nullo:

```
def det (A):
  A=triangolare(A)
  if A==None: return 0
  n=len(A); p=1
  for i in xrange(0,n): p*=A[i][i]
  return p
```

Nota 38.5. Per la risoluzione di un sistema triangolare usiamo la seguente funzione:

```
# I coefficienti nella diagonale principale
# devono essere diversi da 0.
def risolvitriangolare (A):
  A=copy.deepcopy(A); n=len(A)
  for i in xrange(0,n): A[i]=geom.mul(1/float(A[i][i]),A[i])
  for j in xrange(n-1,-1,-1):
    Aj=A[j]
    for i in xrange(0,j):
      A[i]=geom.diff(A[i],geom.mul(A[i][j],Aj))
  return A
```

Nell'algoritmo di eliminazione di Gauss la matrice viene prima triangolarizzata, poi si usa la funzione `risolvitriangolare`:

```
def gauss (A):
  n=len(A); A=triangolare(A)
  if A==None: return None
  A=risolvitriangolare(A)
  A=trasposta(A); A=A[n:]; A=trasposta(A); return(A)
```

Il lato destro del sistema può consistere anche di più di un vettore colonna. In particolare possiamo trovare l'inversa di una matrice se a destra scriviamo la matrice identica:

```
def inversa (A):
  B=trasposta(A)
  B.extend(identica(len(A)))
  B=trasposta(B); return gauss(B)
```

Per ottenere una matrice identica usiamo

```
def identica (n):
  A=range(n)
  for i in xrange(n): A[i]=range(n)
  for i in xrange(n):
    for j in xrange(n):
      if i==j: A[i][j]=1
      else: A[i][j]=0
  return A
```

Usate con un po' di attenzione le funzioni che abbiamo creato permettono spesso la soluzione di piccoli sistemi senza dover ricorrere ai pacchetti numerici che Python (in particolare nella distribuzione Enthought) metterebbe a disposizione.

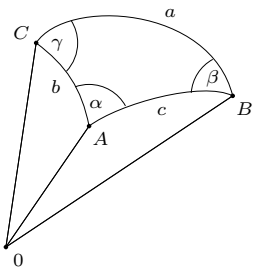
XI. TRIGONOMETRIA SFERICA

Triangoli euleriani

Situazione 39.1. A, B, C siano tre vettori linearmente indipendenti e di lunghezza 1 in $\mathbb{R}^3$. Assumiamo inoltre che la base A, B, C sia positivamente orientata, cioè che $\det(A, B, C) > 0$.

Denotiamo, come d'uso nella matematica, con S^2 la sfera unitaria in $\mathbb{R}^3$; quindi $S^2 = \{x \in \mathbb{R}^3 \mid |x| = 1\}$. Per ipotesi i tre punti A, B e C si trovano su questa sfera.

Nota 39.2. I tre vettori individuano univocamente un *triangolo sferico euleriano* (o *elementare*), se chiediamo che gli angoli $a, b, c, \alpha, \beta, \gamma$ nella figura siano presi tra 0 e $180°$.



Denotiamo il triangolo con ABC.

L'angolo tra B e C può essere identificato con un arco sulla sfera tra B e C con centro nell'origine (l'arco è quindi parte di un *cerchio massimo* o *circolo massimo*) e viene denotato con a. α è invcee l'angolo tra il piano generato da A, B, 0 e il piano generato da A, C, 0. Nello stesso modo sono definiti b, c, β, γ.

A, B e C si chiamano i *vertici,* a, b e c i *lati* ed α, β e γ gli *angoli* del triangolo sferico. Abbiamo

$$\cos a = \|B, C\|$$
$$\cos b = \|A, C\|$$
$$\cos c = \|A, B\|$$

e

$$\cos\alpha = \frac{\|A \times B, A \times C\|}{|A \times B||A \times C|}$$

$$\cos\beta = \frac{\|B \times A, B \times C\|}{|B \times A||B \times C|}$$

$$\cos\gamma = \frac{\|C \times A, C \times B\|}{|C \times A||C \times B|}$$

utilizzando quanto visto nelle note 37.2 e 36.2.

Siccome gli angoli $a, b, c, \alpha, \beta, \gamma$ sono presi tra 0 e $180°$, essi hanno seno > 0. In particolare abbiamo

$$\sin a = |B \times C|$$
$$\sin b = |A \times C|$$
$$\sin c = |A \times B|$$

Formule fondamentali della trigonometria sferica

Lemma 39.3. $\det(A, B, C) = \sin\alpha\ \sin b\ \sin c$.

Dimostrazione. Usando l'ipotesi $\det(A, B, C) > 0$ dal cor. 32.8 abbiamo

$$\begin{aligned}\det(A, B, C) &= |\det(A, B, C)A| = |(A \times B) \times (A \times C)| \\ &= |A \times B||A \times C|\sin\alpha = \sin c\ \sin b\ \sin\alpha\end{aligned}$$

Teorema 39.4 (relazioni di Vieta). $\dfrac{\sin\alpha}{\sin a} = \dfrac{\sin\beta}{\sin b} = \dfrac{\sin\gamma}{\sin c}$.

Dimostrazione. Dal lemma 39.3 abbiamo

$$\frac{\sin\alpha}{\sin a} = \frac{\det(A, B, C)}{\sin a\ \sin b\ \sin c}.$$

Però

$$\frac{\det(A, B, C)}{\sin a\ \sin b\ \sin c} = \frac{\det(B, C, A)}{\sin b\ \sin c\ \sin a} = \frac{\sin\beta}{\sin b}$$

e similmente vediamo che l'espressione è anche uguale a $\dfrac{\sin\gamma}{\sin c}$.

Teorema 39.5 (primo teorema del coseno).

$$\cos a = \cos b\ \cos c + \sin b\ \sin c\ \cos\alpha$$

Dimostrazione. Questo teorema è una conseguenza diretta della prop. 32.9. Infatti da quella proposizione abbiamo

$$\begin{aligned}\cos\alpha &= \frac{\|A \times B, A \times C\|}{|A \times B||A \times C|} = \frac{\|A \times B, A \times C\|}{\sin c\ \sin b} \\ &= \frac{\|A, A\|\|B, C\| - \|A, C\|\|B, A\|}{\sin c\ \sin b} = \frac{\cos a - \cos b\ \cos c}{\sin c\ \sin b}\end{aligned}$$

Teorema 39.6 (primo teorema del seno-coseno).

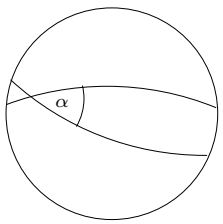
$$\sin a\ \cos b = \sin c\ \cos\beta + \sin b\ \cos a\ \cos\gamma$$

Dimostrazione. Usando la prop. 32.9 e le uguaglianze $\|B, B\| = \|C, C\| = 1$ abbiamo

$$\begin{aligned}&\sin c\ \cos\beta + \sin b\ \cos a\ \cos\gamma \\ &= |A \times B|\frac{\|B \times A, B \times C\|}{|B \times A||B \times C|} + |A \times C|\|B, C\|\frac{\|C \times A, C \times B\|}{|C \times A||C \times B|} \\ &= \frac{\|B, B\|\|A, C\| - \|B, C\|\|A, B\|}{|B \times C|} \\ &\qquad + \frac{\|B, C\|(\|C, C\|\|A, B\| - \|C, B\|\|A, C\|)}{|B \times C|} \\ &= \frac{\|A, C\| - \|B, C\|\|A, B\| + \|B, C\|\|A, B\| - \|B, C\|^2\|A, C\|}{|B \times C|} \\ &= \frac{\|A, C\|}{|B \times C|}(1 - \|B, C\|^2) = \frac{\cos b}{\sin a}\ \sin^2 a = \sin a\ \cos b\end{aligned}$$

Triangoli ausiliari

Nota 39.7. Ogni cerchio massimo sulla sfera determina un piano passante per l'origine, due cerchi massimi non coincidenti quindi due piani passanti per l'origine. L'angolo α tra questi piani lo scegliamo di nuovo in modo che $0 < \alpha < 180°$. I due piani determinano un *biangolo* (scelto in corrispondenza di α) la cui area, essende 4π l'area della sfera, è uguale a $(\alpha/2\pi)4\pi = 2\alpha$.



Definizione 39.8. Scriviamo $\overline{A}$ per il punto antipodale $-A$ di A; similmente sono definiti $\overline{B}$ e $\overline{C}$. I triangoli $AB\overline{C}$, $A\overline{B}C$ e $\overline{A}BC$ si chiamano *triangoli laterali* di ABC. $AB\overline{C}$ è il complemento del biangolo determinato dai lati a e b. I triangoli $\overline{A}BC$, $\overline{A}B\overline{C}$ e $A\overline{B}C$ si chiamano *triangoli verticali* di ABC. Il triangolo $\overline{A}\overline{B}\overline{C}$ si chiama *triangolo antipodale* di ABC.

I *poli* di un circolo massimo sono i due punti che sarebbero considerati come poli della sfera se il circolo massimo dato fosse l'equatore della sfera.

Il triangolo verticale $\overline{A}\overline{B}C$ si ottiene continuando i lati a e b attraverso C in direzione delle proiezioni $\overline{A}$ e $\overline{B}$ di questi lati sul circolo massimo determinato da c.

Con i poli dei circoli massimi passanti per i lati del triangolo si può formare il *triangolo polare* che sta alla base del *principio di dualità* della trigonometria sferica (pagina 40); con il prodotto vettoriale lo studio del triangolo polare risulterà molto semplice.

L'area del triangolo sferico

Esercizio 40.1. Disegnare su una pallina da ping-pong un triangolo sferico ABC con i tre circoli massimi che esso determina, i punti antipodali e i triangoli laterali e verticali. Usare più colori.

Convincersi che l'unione (disgiunta) $ABC \cup \overline{A}BC \cup A\overline{B}C \cup \overline{AB}\overline{C}$ è uguale alla semisfera determinata dal circolo massimo che passa per A e B e che contiene il terzo punto C.

Teorema 40.2. *L'area del triangolo sferico (euleriano) ABC è uguale ad $\alpha + \beta + \gamma - \pi$.*

Dimostrazione. Scriviamo solo in questa dimostrazione $[ABC]$ per l'area di ABC ecc. Per la nota 39.7 abbiamo

$$[ABC] + [\overline{A}BC] = 2\alpha$$
$$[ABC] + [A\overline{B}C] = 2\beta$$
$$[ABC] + [AB\overline{C}] = 2\gamma$$

$AB\overline{C}$ è però il triangolo antipodale di $\overline{AB}C$, per cui $[AB\overline{C}] = [\overline{AB}C]$.

Dalla relazione nell'esercizio 40.1 sappiamo adesso che l'unione disgiunta $ABC \cup \overline{A}BC \cup A\overline{B}C \cup \overline{AB}\overline{C}$ è una semisfera e possiede perciò l'area 2π.

Da ciò segue $2[ABC] + 2\pi = 2(\alpha + \beta + \gamma)$, da cui otteniamo l'enunciato: $[ABC] = \alpha + \beta + \gamma - \pi$.

Nota 40.3. L'area $\alpha + \beta + \gamma - \pi$ si chiama anche *eccesso sferico* del triangolo ABC.

Il teorema 40.2 ha una sorprendente implicazione: La somma degli angoli in un triangolo sferico euleriano è sempre maggiore di $180°$!

Mentre in un triangolo piano la somma degli angoli è sempre uguale (uguale a $180°$), ciò non vale più per i triangoli sferici. Infatti consideriamo sulla sfera terrestre due punti A e B sull'equatore, mentre C sia il polo nord. Se fissiamo A e C e muoviamo B lungo l'equatore, la somma degli angoli sarà uguale a $180° + \delta$, dove δ è differenza delle latitudini di A e B.

Ciò potrebbe far sospettare che la massima somma che si può ottenere sia $360°$ (il valore che si ottiene per $\delta = 180°$). Si può invece dimostrare che sono possibili i valori $180° < \alpha + \beta + \gamma < 540°$.

In particolare si vede che in un triangolo sferico non possiamo calcolare γ conoscendo solo α e β!

Il principio di dualità

Definizione 40.4. I punti $\widehat{A} := \dfrac{B \times C}{|B \times C|}$, $\widehat{B} := \dfrac{C \times A}{|C \times A|}$ e $\widehat{C} := \dfrac{A \times B}{|A \times B|}$ sono detti *poli* del triangolo ABC; $\widehat{A}$ è il polo (sinistro) del circolo massimo orientato passante per $a = BC$ percorso da B verso C ecc. Il triangolo $\widehat{A}\widehat{B}\widehat{C}$ è il triangolo polare (def. 39.8) di ABC.

Il lati del triangolo polare vengono denotati con $\widehat{a}$, $\widehat{b}$ e $\widehat{c}$, gli angoli con $\widehat{\alpha}$, $\widehat{\beta}$, $\widehat{\gamma}$.

Teorema 40.5 (primo teorema di dualità). *Il triangolo polare del triangolo polare di ABC è ABC.*

Più precisamente $\widehat{\widehat{A}} = A, \widehat{\widehat{B}} = B, \widehat{\widehat{C}} = C$.

Dimostrazione. Usiamo, solo in questa dimostrazione, la notazione $v \sim w$ per due vettori v e w, se esiste $\lambda > 0$ con $w = \lambda v$. È chiaro che $\sim$ è una relazione di equivalenza e che due vettori di lunghezza 1 coincidono se sono equivalenti rispetto a $\sim$. È quindi sufficiente dimostrare che $\widehat{\widehat{A}} \sim A$, $\widehat{\widehat{B}} \sim B$, $\widehat{\widehat{C}} \sim C$. Per definizione

$$\begin{aligned}\widehat{\widehat{A}} &\sim \widehat{B} \times \widehat{C} \sim (C \times A) \times (A \times B)\\ &= \det(C, A, B)A - \det(C, A, A)B = \det(C, A, B)A \sim A\end{aligned}$$

perché $\det(C, A, B) = \det(A, B, C) > 0$ e utilizzando il cor. 32.8.

Teorema 40.6 (secondo teorema di dualità). *Le somme $\widehat{a} + \alpha$, $\widehat{b} + \beta$, $\widehat{c} + \gamma$, $a + \widehat{\alpha}$, $b + \widehat{\beta}$ e $c + \widehat{\gamma}$ sono tutte uguali a π.*

Dimostrazione. Per simmetria e per il primo teorema di dualità è sufficiente dimostrare che $\widehat{a} + \alpha = \pi$. Ma

$$\cos \widehat{a} = \|\widehat{B}, \widehat{C}\| = \frac{\|C \times A, A \times B\|}{|C \times A||A \times B|} = -\cos\alpha = \cos(\pi - \alpha)$$

Siccome $0 < \widehat{a} < \pi$ e $0 < \alpha < \pi$ ciò implica $\widehat{a} = \pi - \alpha$.

Osservazione 40.7. I teoremi di dualità implicano che nelle formule per il triangolo sferico possiame scambiare angoli e lati, tenendo conto delle relazioni contenute nel teorema 40.6.

Teorema 40.8 (secondo teorema del coseno).

$$\cos\alpha = -\cos\beta\ \cos\gamma + \sin\beta\ \sin\gamma\ \cos a$$

Dimostrazione. Per i teoremi di dualità dal primo teorema del coseno abbiamo

$$\cos(\pi-\alpha) = \cos(\pi-\beta)\ \cos(\pi-\gamma)+\sin(\pi-\beta)\ \sin(\pi-\gamma)\ \cos(\pi-a)$$

ovvero $-\cos\alpha = \cos\beta\ \cos\gamma - \sin\beta\ \sin\gamma\ \cos a$.

Teorema 40.9 (secondo teorema del seno-coseno).

$$\sin\alpha\ \cos\beta = \sin\gamma\ \cos b - \sin\beta\ \cos\alpha\ \cos c$$

Dimostrazione. Per i teoremi di dualità dal primo teorema del seno-coseno abbiamo

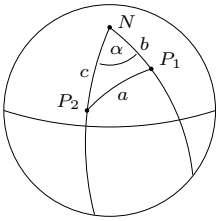
$$\begin{aligned}\sin(\pi-\alpha)\ \cos(\pi-\beta) &= \sin(\pi-\gamma)\ \cos(\pi-b)\\ &\quad + \sin(\pi-\beta)\ \cos(\pi-\alpha)\ cos(\pi-c)\end{aligned}$$

ovvero $-\sin\alpha\ \cos\beta = -\sin\gamma\ \cos b + \sin\beta\ \cos\alpha\ \cos c$.

Distanze sulla sfera

Nota 40.10. Possiamo usare il primo teorema del coseno per calcolare la distanza sulla sfera terrestre tra due luoghi di cui conosciamo le coordinate geografiche. Latitudini meridionali e longitudini ovest corrispondono a valori negativi.

Il luogo P_1 abbia latitudine φ_1 e longitudine λ_1, il luogo P_2 latitudine φ_2 e longitudine λ_2. Consideriamo il triangolo sferico della figura:



$$b = 90° - \varphi_1$$
$$c = 90° - \varphi_2$$
$$\alpha = \lambda_2 - \lambda_1$$

Dal primo teorema del coseno abbiamo

$$\begin{aligned}\cos a &= \cos(90° - \varphi_1)\ \cos(90° - \varphi_2)\\ &\quad + \sin(90° - \varphi_1)\ \sin(90° - \varphi_2)\ \cos(\lambda_2 - \lambda_1)\\ &= \sin\varphi_1\ \sin\varphi_2 + \cos\varphi_1\ \cos\varphi_2\ \cos(\lambda_2 - \lambda_1)\end{aligned}$$

Otteniamo a come $\arccos(\cos a)$; per trovare la distanza dobbiamo moltiplicare a con il raggio medio terrestre che è uguale a 6378 km.

Ferrara, Buenos Aires e Los Angeles hanno le seguenti coordinate:

	lat.	long.
Ferrara	45	12
Buenos Aires	−35	−59
Los Angeles	34	−118

Calcoliamo la distanza tra Ferrara e Buenos Aires:

$$\cos a = -\sin 45°\ \sin 35° + \cos 45°\ \cos 35°\ \cos 71°$$

per cui $\cos a = -0.217$. Siccome $\arccos(-0.217) = 1.7895$, otteniamo la distanza di 11413 km.

Per il calcolo possiamo usare il seguente programma in Python:

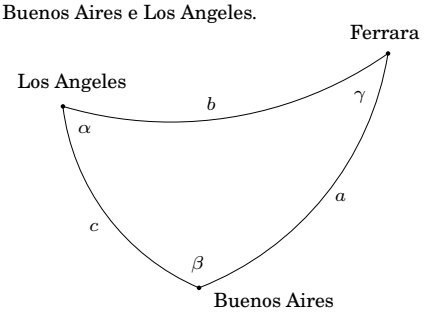
```
# Distanza sulla sfera terrestre.
def dist (lat1,long1,lat2,long2):
  cosa=singradi(lat1)*singradi(lat2)+ \
    cosgradi(lat1)*cosgradi(lat2)*cosgradi(long2-long1)
  return math.acos(cosa)*6378
```

Per calcolare la distanza tra Ferrara e Buenos Aires con questa funzione usiamo l'istruzione

```
print trigo.dist(45,12,-35,-59)
```

Un triangolo sulla sfera terrestre

Vogliamo calcolare l'area del triangolo sferico con i vertici Ferrara, Buenos Aires e Los Angeles.



Sappiamo dal teorema 40.2 che l'area è uguale ad $\alpha + \beta + \gamma - \pi$ moltiplicato per 6371^2 km^2.

Con il procedimento della nota 40.10 possiamo calcolare a, b e c e poi α, β, γ, utilizzando di nuovo il primo teorema del coseno:

$$\cos\alpha = \frac{\cos a - \cos b \ \cos c}{\sin b \ \sin c}$$

ecc. Cio porta alla seguente funzione in Python:

```
# Area del triangolo definito da tre punti
# sulla sfera terrestre.
def area (lat1,long1,lat2,long2,lat3,long3):
  cosa=cosenosferico(lat2,long2,lat3,long3)
  cosb=cosenosferico(lat1,long1,lat3,long3)
  cosc=cosenosferico(lat1,long1,lat2,long2)
  sina=math.sqrt(1-cosa*cosa)
  sinb=math.sqrt(1-cosb*cosb)
  sinc=math.sqrt(1-cosc*cosc)
  cosalfa=(cosa-cosb*cosc)/(sinb*sinc)
  cosbeta=(cosb-cosa*cosc)/(sina*sinc)
  cosgamma=(cosc-cosa*cosb)/(sina*sinb)
  alfa=math.acos(cosalfa); beta=math.acos(cosbeta)
  gamma=math.acos(cosgamma)
  A=alfa+beta+gamma-math.pi; r=6378
  return r*r*A
```

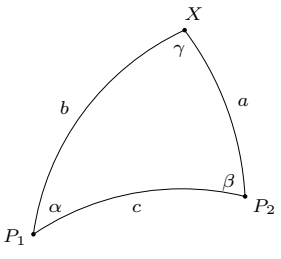
in cui per il calcolo di $\cos a$ secondo il teorema 39.5 abbiamo usato la funzione

```
# cos(a) in funzione di latitudini e longitudini.
def cosenosferico (lat1,long1,lat2,long2):
  return singradi(lat1)*singradi(lat2)+ \
    cosgradi(lat1)*cosgradi(lat2)*cosgradi(long2-long1)
```

Per il triangolo Ferrara, Buenos Aires e Los Angeles troviamo un'area di 70408117 km^2.

Un compito di radiogoniometria

Nella navigazione marittima o aerea ci si trova spesso nella seguente situazione: Del triangolo sferico nella figura sono noti gli angoli α e β, tipicamente rilevati tramite segnali radio (*radiogoniometria*), e il lato c.



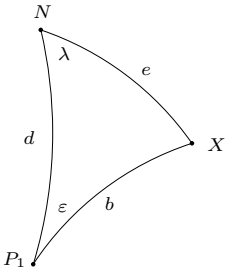
Con il teorema 40.8 calcoliamo prima γ tramite

$$\cos\gamma = -\cos\alpha \ \cos\beta + \sin\alpha \ \sin\beta \ \cos c$$

poi a e b con le relazioni di Vieta (teorema 39.4):

$$\sin a = \sin\alpha \ \frac{\sin c}{\sin\gamma}$$
$$\sin b = \sin\beta \ \frac{\sin c}{\sin\gamma}$$

A questo punto, se sono note le coordinate geografiche di P_1 e P_2, con ulteriori considerazioni non troppo difficili è possibile calcolare la posizione del punto X. All'incirca si procede cosi: Sia N il polo nord.



ε e b sono noti (durante il rilevamento si calcolano infatti sia ε che α) e anche d (perché conosciamo la latitudine di P_1). Troviamo prima

$$\cos e = \cos d \ \cos b + \sin d \ \sin b \ \cos\varepsilon$$

e con e la latitudine di X. Poi con le formule di Vieta otteniamo

$$\sin\lambda = \sin b \ \frac{\sin\varepsilon}{\sin e}$$

cosicché, se λ_1 è la longitudine di P_1, con $\lambda + \lambda_1$ troviamo la longitudine di X.

Bisogna stare un po' attenti per evitare di lavorare con triangoli che non sono euleriani.

Esercizi per gli scritti

79. Dov'è l'errore?

```
def unterzo (x): return float(x/3)
```

80. $\begin{pmatrix} 3 & 2 \\ 4 & 5 \end{pmatrix} \begin{pmatrix} 4 & 2 \\ 1 & 6 \end{pmatrix}$

81. $\begin{pmatrix} 2 & 3 & 2 \\ 1 & 4 & 2 \\ 3 & 5 & 1 \end{pmatrix} \begin{pmatrix} 6 & 2 & 1 \\ 3 & 1 & 5 \\ 5 & 8 & 3 \end{pmatrix}$

82. $\begin{pmatrix} 2 & 3 & 5 & 6 \\ 4 & 1 & 2 & 1 \end{pmatrix} \begin{pmatrix} 7 & 8 & 4 \\ 2 & 4 & 0 \\ 8 & 1 & 1 \\ 3 & 5 & 7 \end{pmatrix}$

83. $\begin{pmatrix} 3 & 4 & 1 & 2 \\ 2 & 1 & 8 & 1 \\ 5 & 7 & 3 & 5 \end{pmatrix} \begin{pmatrix} 6 \\ 2 \\ 3 \\ 5 \end{pmatrix}$

84. $\begin{pmatrix} 1 & 4 & 3 & 6 \end{pmatrix} \begin{pmatrix} 2 & 3 \\ 5 & 1 \\ 1 & 6 \\ 8 & 2 \end{pmatrix}$

85. Sia $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \mathbb{R}^2_2$ e $|A| \neq 0$. Calcolare l'inversa A^{-1}.

86. Calcolare la distanza fra Los Angeles e Buenos Aires.

XII. UN ALGORITMO DI COMPRESSIONE

Codici

Situazione 42.1. Nelle prime considerazioni su questa pagina, ove non indicato diversamente, sia A un insieme finito, ad esempio $A = \{0, 1\}$. E sia un altro insieme finito.

Definizione 42.2. Con A^+ denotiamo l'insieme di tutte le sequenze finite (dette in questo contesto *parole*) di lunghezza ≥ 1 formate con lettere appartenenti ad A. Introducendo la parola vuota ε (di lunghezza 0), poniamo $A^* := A^+ \cup \{\varepsilon\}$. Formalmente:

$$A^* = \bigcup_{n=0}^{\infty} A^n \quad \text{ed} \quad A^+ = \bigcup_{n=1}^{\infty} A^n$$

Identifichiamo A con $A^1 \subset A^*$. Siccome $A^0 = \{\emptyset\}$, si potrebbe identificare ε con l'insieme vuoto $\emptyset$. Se $x \in A^n$, diciamo che x possiede *lunghezza* n e scriviamo $|x| := n$.

Sottoinsiemi di A^* si chiamano *linguaggi formali*; la loro teoria si colloca al confine tra matematica pura e informatica.

A^* diventa un monoide (cioè un semigruppo con elemento neutro, che in questo caso naturalmente coincide con ε) se come prodotto di due parole utilizziamo la loro naturale concatenazione. Nella teoria di A^* una sequenza $a_1, \ldots, a_n$ viene spesso scritta semplicemente nella forma $a_1 \ldots a_n$, tralasciando le parentesi. Il prodotto è quindi definito da

$$a_1 \ldots a_n \cdot b_1 \ldots b_m := a_1 \ldots a_n b_1 \ldots b_m$$

È anche chiaro a questo punto che $a_1 \ldots a_n$ è proprio il prodotto delle lettere a_1,...,a_n, considerate come elementi di A^*.

Il monoide A^* si chiama il *monoide libero* sull'alfabeto A ed è altamente noncommutativo. Con i ragionamenti visti a pagina 24 si può dimostrare che $\{0, 1\}^*$ è isomorfo ad $SL(2, \mathbb{N})$.

Osservazione 42.3. Siano $u, p \in A^*$. Allora p si chiama un *prefisso* di u, se esiste una parola $x \in A^*$ tale che $u = px$.

x può essere la parola vuota, quindi u stesso è prefisso di u. Anche ε è prefisso di u, perché $u = \varepsilon u$.

Definizione 42.4. Per un sottoinsieme $X \subset A^*$ denotiamo con $\text{semigruppo}(X)$ il semigruppo generato da X. Quindi

$$\text{semigruppo}(X) = \{x_1 \cdots x_n \mid n \geq 1 \text{ e } x_1, \ldots, x_n \in X\}$$

Definizione 42.5. Un sottoinsieme $X \subset A^*$ si chiama un *codice*, se ogni elemento u di $\text{semigruppo}(X)$ possiede un'unica fattorizzazione come prodotto di elementi di X, cioè della forma $u = x_1 \cdots x_n$ con $x_1, \ldots, x_n \in X$.

Definizione 42.6. X sia un codice. Una *codifica* di E su X è un'applicazione biiettiva $\tau : E \longrightarrow X$.

Nota 42.7. Siano X un codice e $\tau : E \longrightarrow X$ una codifica. Allora esiste un unico isomorfismo di semigruppi $\tau^+ : E^+ \longrightarrow \text{semigruppo}(X)$ che su E coincide con τ. Ciò significa che da ogni parola che possiamo formare moltiplicando elementi di X otteniamo un'unica parola in E^+. L'applicazione inversa $(\tau^+)^{-1} : \text{semigruppo}(X) \longrightarrow E^+$ è l'applicazione di *decodifica*.

τ^+ è definito semplicemente in questo modo: Siano $e_1, \ldots, e_n \in E$. Allora poniamo $\tau^+(e_1 \ldots e_n) := \tau(e_1) \cdots \tau(e_n)$. È chiaro che così si ottiene un omomorfismo di semigruppi che su E coincide con τ.

È anche chiaro che τ^+ è suriettivo. Infatti ogni elemento u di $\text{semigruppo}(X)$ possiede un'unica fattorizzazione $u = x_1 \cdots x_n$ con $x_1, \ldots, x_n \in X$, perché X è un codice. Siano $e_1 := \tau^{-1}(x_1)$, ..., $e_n := \tau^{-1}(x_n)$. Allora $u = \tau^+(e_1 \ldots e_n)$.

Dimostriamo ancora che τ^+ è iniettivo. Siano quindi $e_1, \ldots, e_n, f_1, \ldots, f_m \in E$ tali che $\tau^+(e_1 \ldots e_n) = \tau^+(f_1 \ldots f_m)$. Ponendo $x_i := \tau(e_i)$ ed $y_j := \tau(f_j)$ per ogni i, j abbiamo allora $x_1 \cdots x_n = y_1 \cdots y_m$. Ma X è un codice, per cui necessariamente $n = m$ ed $x_i = y_i$ per ogni i. Siccome l'applicazione τ è iniettiva, ciò implica $e_i = f_i$ per ogni i.

Definizione 42.8. Un *codice a blocchi* è un sottoinsieme $X \subset A^*$ i cui elementi sono tutti della stessa lunghezza ≥ 1. Esiste quindi un $n \geq 1$ tale che $X \subset A^n$.

È immediato che un codice a blocchi è veramente un codice nel senso della def. 42.5.

Codici a blocchi vengono usati soprattutto nella correzione degli errori durante la trasmissione (*codifica di canale*).

Decodifica per un codice prefisso

Definizione 42.9. Un insieme $X \subset A^*$ si chiama un *codice prefisso* (in inglese *prefix code*), se nessun elemento di X è prefisso di un altro elemento di X.

Formalmente chiediamo quindi che, se $x, y \in X$ e se y è un prefisso di x, allora $y = x$.

Proposizione 42.10. *Un codice prefisso è un codice.*

Dimostrazione. Sia X un codice prefisso. Assumiamo che $x_1 \cdots x_n = y_1 \cdots y_m$ con $x_1, \ldots, x_n, y_1, \ldots, y_m \in X$.

Allora deve valere una delle due disuguaglianze $|x_1| \leq |y_1|$ e $|y_1| \leq |x_1|$. Se ad esempio è vera la prima, x_1 è necessariamente prefisso di y_1 e coincide quindi con y_1. È chiaro che in questo modo otteniamo $n = m$ ed $x_i = y_i$ per ogni i.

Lemma 42.11. *X sia un codice prefisso ed $u \in \text{semigruppo}(X)$. Siano $p \in X$ e $v \in A^*$ tali che $u = pv$. Allora $v = \varepsilon$ oppure $v \in \text{semigruppo}(X)$.*

Dimostrazione. Questo lemma è meno ovvio di quanto sembri, anche se adesso la dimostrazione non sarà difficile. Dobbiamo però esaminare con attenzione le ipotesi e l'enunciato.

u è prodotto di elementi di X, quindi esistono $n \geq 1$ ed elementi $x_1, \ldots, x_n \in X$ tali che $u = x_1 \cdots x_n$. Come nella dimostrazione della prop. 42.10 p deve essere prefisso di x_1 oppure x_1 prefisso di p. Ma siccome X è un codice prefisso, in entrambi i casi necessariamente $p = x_1$. In A^* vale però la legge di cancellazione, per cui necessariamente $v = \varepsilon$ oppure $v = x_2 \cdots x_n \in \text{semigruppo}(X)$.

Nota 42.12. Dal lemma 42.11 otteniamo un algoritmo per trovare la fattorizzazione di un elemento di $\text{semigruppo}(X)$ nel caso che X sia un codice prefisso.

Sia infatti dato $u \in \text{semigruppo}(X)$, ad esempio $u = x_1 \cdots x_n$ con $x_1, \ldots, x_n \in X$. Allora percorriamo tutti gli elementi di X fino a quando ne troviamo uno che sia prefisso di u. Come nella dimostrazione del lemma 42.11 questo elemento deve coincidere con x_1. Poi applichiamo lo stesso ragionamento ad $x_2 \cdots x_n$ per trovare x_2 ecc.

Nota 42.13. Negli algoritmi di compressione il testo t da comprimere è un prodotto di elementi di E, si ha quindi $t \in E^+$. Si cercano un codice prefisso X e una codifica $\tau : E \longrightarrow X$ tali che il risultato $u := \tau^+(t)$ della codifica sia più breve di t, che siano cioè necessari meno *bit* per rappresentare u di quanti ne richieda l'originale t. Il dato compresso u viene trasmesso attraverso un canale telematico o ad esempio segnali satellitari oppure anche solo conservato fino al prossimo utilizzo.

Per riottenere t da u decomponiamo prima u con il metodo della nota 42.12 nella forma $u = x_1 \cdots x_n$ con $x_i \in X$ per ogni i. Poi, come nella nota 42.7, per ogni x_i cerchiamo $e_i := \tau^{-1}(x_i)$. Allora $t = e_1 \ldots e_n$.

In Python, se lavoriamo con un alfabeto le cui lettere appartengono al repertorio dei caratteri comuni, possiamo usare stringhe per rappresentare parole. Per controllare se una stringa a inizia con la stringa x, cioè se x è prefisso di a, si utilizza il metodo `startswith`: L'espressione `a.startswith(x)` è vera se e solo se x è prefisso di a.

L'algoritmo della nota 42.12 è quindi facilmente tradotto:

```
def fattorizza (a,X):
  fattori=[]
  while a:
    for x in X:
      if a.startswith(x):
        fattori.append(x); a=a[len(x):]; break
    else: return None
  return fattori
```

Invertire un dizionario

Per invertire un dizionario invertibile (in cui esiste cioè una biiezione tra voci e valori) usiamo la seguente funzione che a sua volta utilizza il metodo `iteritems` che restituisce un iteratore che corrisponde alle coppie (voce,valore) del dizionario.

```
def inverti (diz):
  return dict([(y,x) for x,y in diz.iteritems()])
```

Una funzione di decodifica

Anche il metodo della nota 42.13 è facilmente realizzato in Python:

```
def decodifica (u,diz):
  dizinv=inverti(diz); fattori=fattorizza(u,dizinv)
  return ''.join(map(lambda x: dizinv[x],fattori))
```

Qui abbiamo usato due piccoli trucchi, uno matematico e uno di programmazione. Quest'ultimo consiste nel fatto che quando un dizionario viene usato in un contesto in cui ci si aspetta una lista, di esso viene utilizzata solo la liste delle voci. Per questo invece di `fattori=fattorizza(u,dizinv.keys())` abbiamo potuto scrivere più brevemente `fattori=fattorizza(u,dizinv)`.

Il trucco matematico è semplicemente l'osservazione che un sottoinsieme di un codice prefisso è ovviamente ancora un codice prefisso.

L'algoritmo di Huffman

Nota 43.1. Questo algoritmo, molto famoso, risale al 1952 ed è tuttora molto usato nei programmi di compressione, spesso con varianti. Si può dimostrare che in un certo senso è ottimale (almeno dal punto di vista teorico), eppure è piuttosto semplice. Assumiamo di avere un testo t che vogliamo comprimere, composto da lettere distinte $A^1, \dots, A^n$; qui gli esponenti sono naturalmente indici e non corrispondono a potenze. Calcoliamo per ogni A^i la sua frequenza f^i, ciò semplicemente il numero delle volte che A^i appare nel testo t. In verità gli f^i possono essere anche *pesi* ottenuti in altro modo. In ogni caso vorremmo che le lettere con frequenza maggiore vengano codificate mediante parole più corte; le parole con cui codifichiamo apparterranno a $\{0,1\}^*$.

Il procedimento di Huffman è il seguente: Formiamo le coppie $A^i_{f^i}$ che poi elenchiamo in modo tale che la successione delle f^i sia decrescente. Per comodità assumiamo che sia già così all'inizio; consideriamo quindi la successione $A^1_{f1}, \dots, A^n_{fn}$.

Il caso $n \le 2$ è banale. Per $n > 2$ raccogliamo A^{n-1}_{fn-1} ed A^n_{fn} in un'espressione $[A^{n-1}, A^n]_{fn-1+fn}$. Questa viene adesso inserita nell'elenco in modo tale che la successione delle frequenze sia ancora decrescente. Ci fermiamo quando sono rimaste solo due espressioni.

Illustriamo questa prima metà dell'algoritmo di Huffman con un esempio in cui anche la notazione, apparentemente complicata, si rivelerà invece semplificatrice nei calcoli a mano.

L'elenco (ancora non ordinato) delle lettere utilizzate e delle loro frequenze sia $A_{30}\ B_{20}\ C_{10}\ D_8\ E_6\ F_7\ G_9\ H_{10}$.

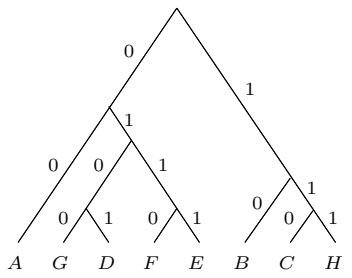
L'elenco ordinato è quindi $A_{30}\ B_{20}\ C_{10}\ H_{10}\ G_9\ D_8\ F_7\ E_6$. Da esso otteniamo successivamente le seguenti successioni:

$$
\begin{aligned}
&A_{30}\ B_{20}\ [F,E]_{13}\ C_{10}\ H_{10}\ G_9\ D_8\\
&A_{30}\ B_{20}\ [G,D]_{17}\ [F,E]_{13}\ C_{10}\ H_{10}\\
&A_{30}\ B_{20}\ [C,H]_{20}\ [G,D]_{17}\ [F,E]_{13}\\
&A_{30}\ [[G,D],[F,E]]_{30}\ B_{20}\ [C,H]_{20}\\
&[B,[C,H]]_{40}\ A_{30}\ [[G,D],[F,E]]_{30}\\
&[A,[[G,D],[F,E]]]_{60}\ [B,[C,H]]_{40}
\end{aligned}
$$

Dall'ultima riga otteniamo la lista annidata

$$[[A,[[G,D],[F,E]]],[B,[C,H]]]$$

Una lista annidata è però la stessa cosa come un albero (nel nostro caso binario) che può essere codificato come nella figura:



Alla codifica dell'albero corrisponde la codifica delle lettere del testo originale:

A 00	B 10	C 110	D 0101
E 0111	F 0110	G 0100	H 111

Questa codifica dell'albero rappresentato dalla lista annidata è la seconda parte dell'algoritmo di Huffman che in Python può essere effettuata con la seguente funzione:

```
def calbin (a,pos=''): # Codifica un albero binario.
  s=a[0]; d=a[1]
  if isinstance(s,list): u=calbin(s,pos+'0')
  else: u={s:pos+'0'}
  if isinstance(d,list): v=calbin(d,pos+'1')
  else: v={d:pos+'1'}
  u.update(v); return u
```

Qui abbiamo usato la funzione `isinstance` per verificare se un oggetto è una lista e il metodo `update` per la fusione dei dizionari rappresentati dai due rami dell'albero, visto a pagina 1.

La funzione `huffman` combina le due parti dell'algoritmo:

```
def huffman (a): # a = lista delle coppie [Ai,fi].
  n=len(a); a=copy.deepcopy(a)
  if n==0: return None
  if n==1: return {a[0][0]:'0'}
  if n==2: return {a[0][0]:'0',a[1][0]:'1'}
  a.sort(reverse=1,key=lambda x: x[1])
  while n>2:
    v=a.pop(); u=a.pop(); peso=u[1]+v[1]; u=[[u[0],v[0]],peso]
    for k in xrange(0,n-2):
      if peso>a[k][1]: a.insert(k,u); break
    else: a.insert(n-2,u)
    n=n-1
  a=[a[0][0],a[1][0]]; return calbin(a)
```

Si può dimostrare che il metodo di Huffman genera un codice prefisso e ciò ci permette di ritrovare il testo originale con il metodo visto a pagina 42: $0001010111111\ldots \longrightarrow ADEH\ldots$

Quanto spazio abbiamo risparmiato nel nostro esempio attraverso la compressione? Le 8 lettere $A, \dots, H$ possono essere rappresentate mediante una codifica a blocchi di 3 bit. Il testo originale era lungo 100 caratteri, corrispondenti a 300 bit. Nella codifica di Huffman il numero di bit di cui abbiamo bisogno è invece uguale a

$$30 \cdot 2 + 20 \cdot 2 + 10 \cdot 3 + 8 \cdot 4 + 6 \cdot 4 + 7 \cdot 4 + 9 \cdot 4 + 10 \cdot 3 = 272$$

In questo caso non abbiamo quindi risparmiato moltissimo. Dobbiamo anche trasmettere la tabella di codifica, anche se ciò in un testo più lungo non fa molta differenza. Il codice generato con il metodo di Huffman è inoltre molto sensibile ad errori di trasmissione. Si usano perciò versioni un po' più sofisticate dell'algoritmo.

Le techniche di compressione fanno parte della *codifica di sorgente*, un campo vasto che comprende tutta la teoria della rappresentazione di informazioni, fondamentale per molti algoritmi di ricerca e di ottimizzazione e in un certo senso per tutta la nostra vita.

Esercizi per gli scritti

87. Verificare che

$$X := \{0100, 0111, 0101, 110, 0110, 10, 111, 00\}$$

è un codice prefisso.

Fattorizzare poi la parola 01110101000011001001111110010, eseguendo a mano l'algoritmo della nota 42.12.

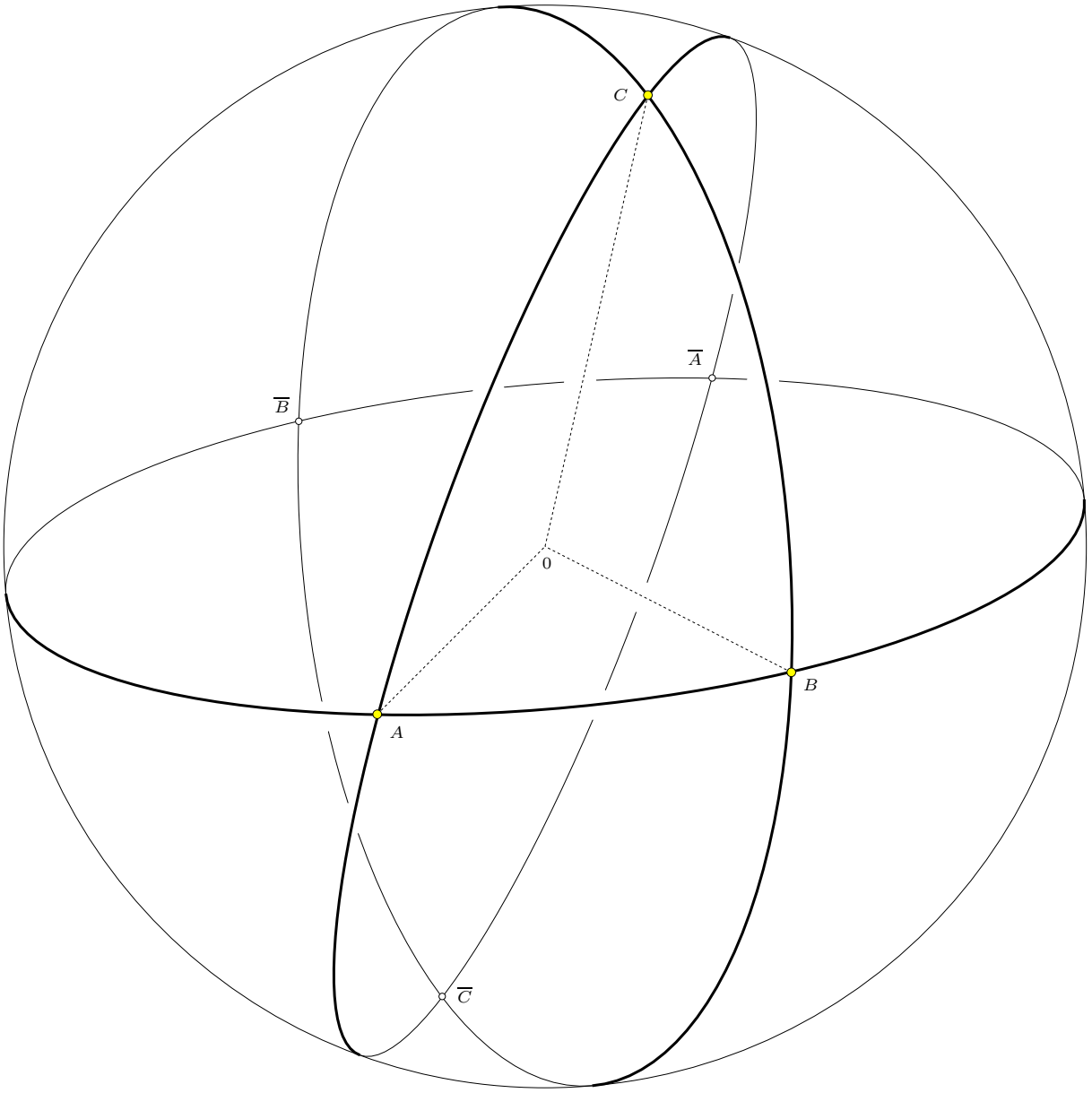
88. Con X come nell'esercizio 87, fattorizzare

01110100100110101001000011101000111010101110100
00100001010100101011101000100011101011101100110

89. Eseguire a mano l'algoritmo di Huffman per la sequenza

$$A_{325}\ B_{210}\ C_{80}\ D_{20}\ E_{70}\ F_{140}\ G_{90}\ H_{65}$$

Stavolta il testo originale consiste di 1000 caratteri a 3 bit. Quanto si risparmia con la compressione?



Fare copie di questo foglio, per poter ripetere gli esercizi più volte.

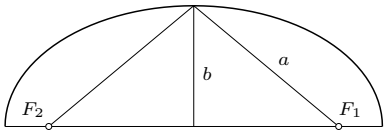
Esercizio 44.1. Indicare i triangoli laterali di ABC.

Esercizio 44.2. Indicare i triangoli verticali di ABC.

Esercizio 44.3. Indicare il triangolo antipodale di ABC.

Esercizio 44.4. Indicare i triangoli ABC, $\overline{A}BC$, A$\overline{B}C$ e $\overline{AB}C$ e convincersi che effettivamente l'unione $ABC \cup \overline{A}BC \cup A\overline{B}C \cup \overline{AB}C$ è uguale alla semisfera determinata dal circolo massimo che passa per A e B e che contiene il terzo punto C - nella figura la semisfera superiore! L'unione è disgiunta, se consideriamo solo gli interni di questi triangoli (perché naturalmente ci sono intersezioni ai bordi, ma ciò è irrilevante per il calcolo dell'area nel teorema 40.2).

Esercizio 44.5. Per ognuna delle tre ellissi determinare (graficamente) prima le due assi e i due fuochi, usando il seguente disegno, valido per ogni ellisse con semidiametro principale uguale ad a:



Nella figura che rappresenta la sfera i due poli corrispondenti a ciascuna delle ellissi si ottengono girando i fuochi per 90 gradi. Indicare quindi il triangolo polare di ABC, i suoi angoli e i suoi lati.