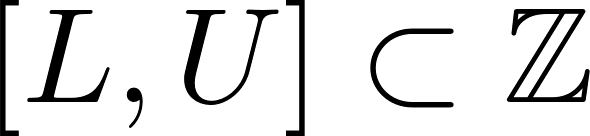
**LEZIONE 3: STRUTTURA DEL SISTEMA FLOATING-POINT**

In questa lezione studieremo la struttura del sistema floating-point, ovvero dell’insieme dei reali-macchina e dei reali rappresentabili per arrotondamento tramite i reali-macchina

Tratteremo un sistema astratto in base b generica (come fatto finora, con esempi in base 10), per poi discutere un modello semplificato dello standard a 64 bit (base 2), utilizzato in Matlab e in tutti i principali linguaggi di calcolo.

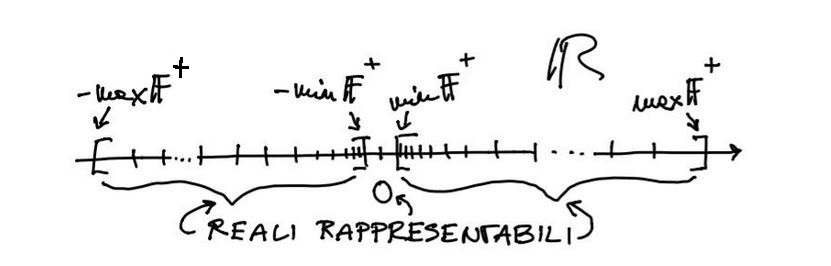
Ricordiamo che i reali macchina in base b e t cifre di mantissa e con range esponenti [](https://www.codecogs.com/eqnedit.php?latex=%5BL%2CU%5D%20%5Csubset%20%5Cmathbb%7BZ%7D#0) [sono definiti da](http://www.texrendr.com/?eqn=%5BL%2CU%5D%20%5Cin%20%5Cmathbb%7BZ%7D#0)

[](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbb%7BF%7D(b%2Ct%2CL%2CU)%3D%5C%7B%5Cmu%20%5Cin%20%5Cmathbb%7BQ%7D%3A%20%5Cmu%3Dsgn(%5Cmu)%5Ccdot%20(0.%5Cmu_1%5Cmu_2%20%5Cdotso%20%5Cmu_t)%20%5Ccdot%20b%5Ep%2C%20%5Cmu_j%20%5Cin%20%5C%7B0%2C1%2C...b-1%5C%7D%2C%20%5Cmu_1%20%5Cneq%200%2C%20p%20%5Cin%20%5BL%2CU%5D%20%5Csubset%20%5Cmathbb%7BZ%7D%20%5C%7D#0)

dove [](https://www.codecogs.com/eqnedit.php?latex=%5Cmu_j#0) con j=1, ..., t sono le cifre della mantissa e p l’esponente intero della potenza della base che sposta la virgola, L<0 e U>0, quindi p appartiene all’intervallo di interi L,L+1,...,-1, 0 ,1,….,U-1,U

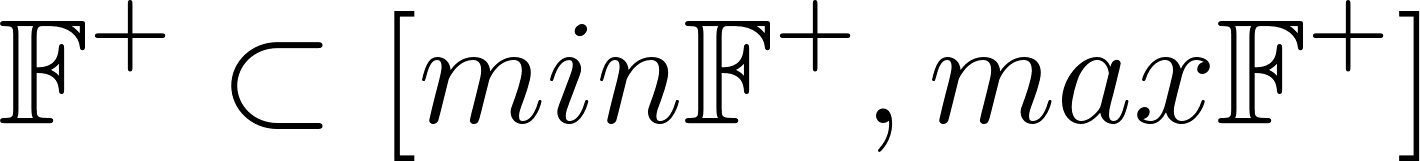
Risponderemo ad alcune domande di base:

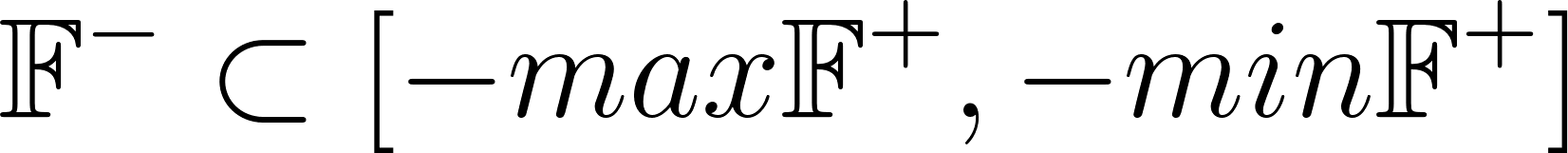
* quanti sono i reali-macchina?
* quali reali sono approssimabili per arrotondamento con i reali-macchina?
* come sono distribuiti sull’asse reale i reali-macchina?

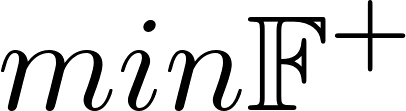
In modo molto schematico, sintetizziamo con un disegno

in questo schemino grafico i reali-macchina corrispondono all’insieme finito di tacche

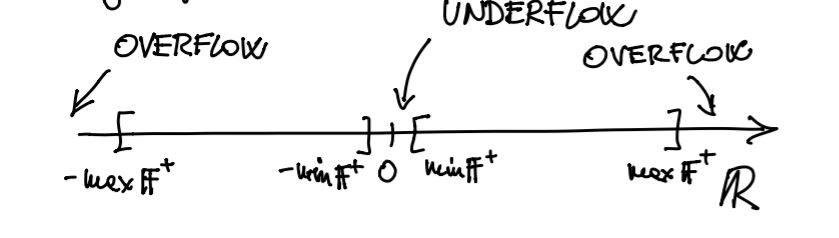
sull’asse reale e a parte lo 0 che è un reale-macchina (con una rappresentazione speciale, visto che la mantissa nulla non è ammessa) stanno nell’unione di due intervalli simmetrici, infatti detti F+ i reali-macchina>0 e F- quelli <0, abbiamo che F- = -F+ (cioè si cambia segno a tutti i positivi, F+ e F- sono simmetrici rispetto a 0)

[](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbb%7BF%7D%5E%2B%20%5Csubset%20%5Bmin%5Cmathbb%7BF%7D%5E%2B%2C%20max%5Cmathbb%7BF%7D%5E%2B%5D#0) (\*1)

[](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbb%7BF%7D%5E-%20%5Csubset%20%5B-max%5Cmathbb%7BF%7D%5E%2B%2C%20-min%5Cmathbb%7BF%7D%5E%2B%5D#0) (\*2)

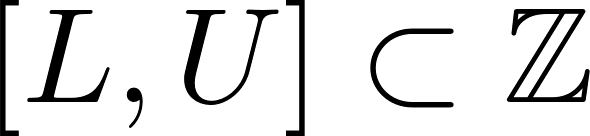
questi due intervalli (\*1) e (\*2) nel continuo sono proprio i reali approssimabili tramite i reali-macchina per arrotondamento a t cifre della mantissa, con un errore relativo<= εM=b1-t/2 (la precisione di macchina) invece i reali che in modulo sono >[](http://www.texrendr.com/?eqn=max%5Cmathbb%7BF%7D%5E%2B#0)e quelli che in modulo sono <[](https://www.codecogs.com/eqnedit.php?latex=min%5Cmathbb%7BF%7D%5E%2B#0), sono “troppo grandi” o “troppo piccoli” per essere approssimati e costituiscono quello che si chiama l’OVERFLOW e l’UNDERFLOW nel sistema floating-point.

Graficamente:



L'UNDERFLOW è un intorno di 0

L'OVERFLOW è un intorno di infinito

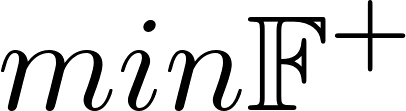
Il problema è che l'esponente p di tali numeri è fuori dal range di esponenti ammissibili ovvero [](https://www.codecogs.com/eqnedit.php?latex=%5BL%2CU%5D%20%5Csubset%20%5Cmathbb%7BZ%7D#0) (cioe p<L (underflow) oppure p>U (overflow)).

La comparsa di numeri in overflow/underflow durante un processo di calcolo altera il processo stesso facendogli tendenzialmente perdere di significato

Nei vecchi linguaggi di programmazione (ad esempio il Fortran) la comparsa di un overflow determinava addirittura l'arresto del processo di calcolo con un messaggio di errore.

In matlab invece (e in altri linguaggi) un numero in overflow, generato ad esempio dal prodotto di reali-macchina molto grandi in modulo, viene indicato col simbolo "Inf" (infinito) ed entra nel processo di calcolo alternandolo, generando altri "Inf" o forme indeterminate (ad es. Inf/Inf) che vengono indicate col simbolo NaN (Not a Number).

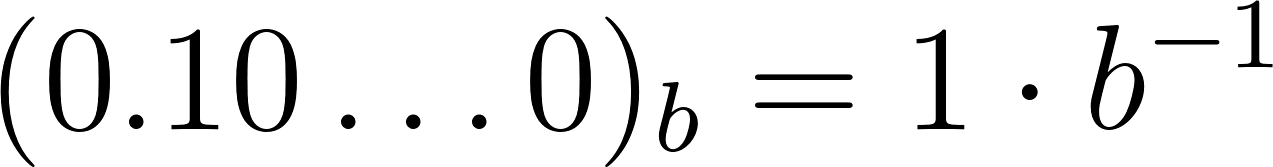
Per questo negli algoritmi numerici è importante cercare di prevenire overflow/underflow.

A questo punto conviene calcolare [](https://www.codecogs.com/eqnedit.php?latex=min%5Cmathbb%7BF%7D%5E%2B#0) e [](https://www.codecogs.com/eqnedit.php?latex=max%5Cmathbb%7BF%7D%5E%2B#0), in modo da quantificare gli estremi degli intervalli di rappresentazione.

Visto che un reale-macchina ha la struttura

[](https://www.codecogs.com/eqnedit.php?latex=%5Cmu%3Dsegno%20%5Ccdot%20mantissa%20%5Ccdot%20b%5Ep#0)

il minimo positivo si otterrà moltiplicando la minima mantissa per la potenza della base con il minimo esponente

Mantissa minima = [](https://www.codecogs.com/eqnedit.php?latex=(0.10%20%5Cdotso%200)_b%20%3D%201%20%5Ccdot%20b%5E%7B-1%7D#0)

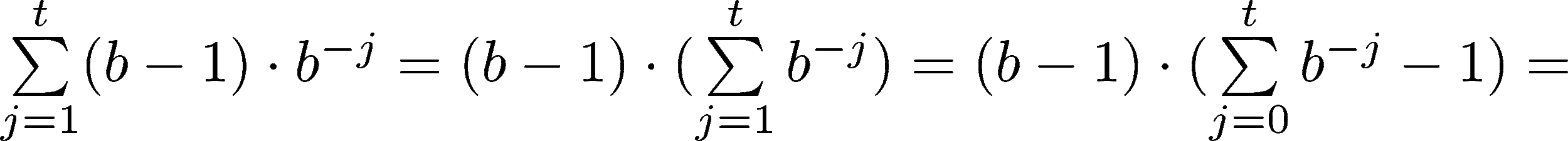
Esponente minimo = L  
Quindi 

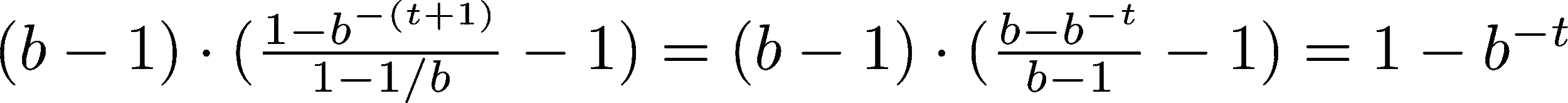
Invece il massimo positivo si otterrà moltiplicando la mantissa massima per la potenza della base col massimo esponente

Mantissa massima = 

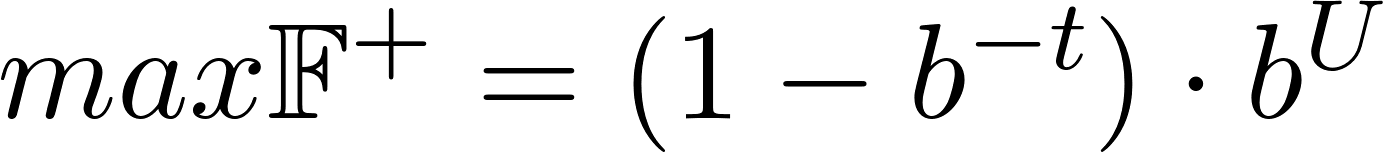
Cioè tutte le t cifre sono uguali alla cifra massima che è *b-1* ad esempio  in base 10

Calcoliamo usando la somma geometrica di ragione b-1 la mantissa massima.

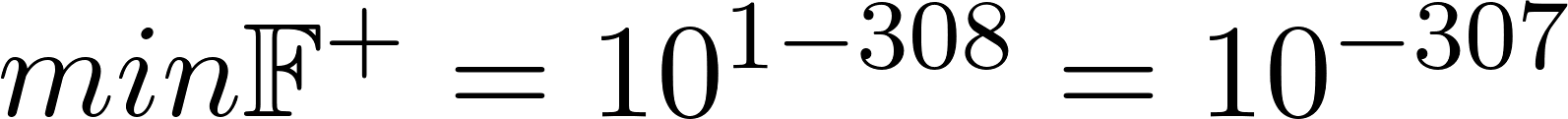
[](http://www.texrendr.com/?eqn=%5Csum%5Climits_%7Bj%3D1%7D%5Et%20(b-1)%20%5Ccdot%20b%5E%7B-j%7D%20%3D%20(b-1)%5Ccdot%20(%5Csum%5Climits_%7Bj%3D1%7D%5Et%20b%5E%7B-j%7D)%20%3D%20(b-1)%5Ccdot%20(%5Csum%5Climits_%7Bj%3D0%7D%5Et%20b%5E%7B-j%7D-1)%20%3D%20#0)

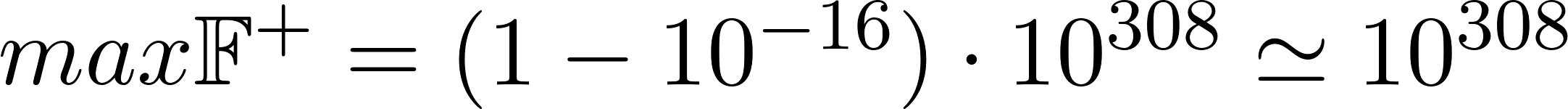
[](http://www.texrendr.com/?eqn=(b-1)%5Ccdot%20(%5Cfrac%7B1-b%5E%7B-(t%2B1)%7D%7D%7B1-1%2Fb%7D-1)%20%3D%20(b-1)%5Ccdot%20(%5Cfrac%7Bb-b%5E%7B-t%7D%7D%7Bb-1%7D-1)%20%3D%201-b%5E%7B-t%7D#0)

Ora esponente\_max = U

Quindi [](https://www.codecogs.com/eqnedit.php?latex=max%5Cmathbb%7BF%7D%5E%2B%20%3D%20(1-b%5E%7B-t%7D)%5Ccdot%20b%5EU%20#0)

Ad esempio in base 10 on 16 cifre di mantissa con esponenti minimo L=-308 e massimo U = +308 (modello che vedremo corrisponde in sostanza all’interfaccia del MatLab, ben sapendo che la rappresentazione interna del calcolatore è in base 2 e non 10)

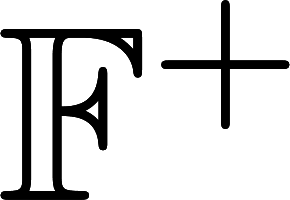
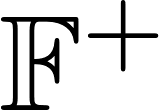
[](https://www.codecogs.com/eqnedit.php?latex=min%5Cmathbb%7BF%7D%5E%2B%3D10%5E%7B1-308%7D%20%3D%2010%5E%7B-307%7D#0)

[](https://www.codecogs.com/eqnedit.php?latex=max%5Cmathbb%7BF%7D%5E%2B%3D(1-%2010%5E%7B-16%7D)%5Ccdot%2010%5E%7B308%7D%20%5Csimeq%20%2010%5E%7B308%7D%20#0)

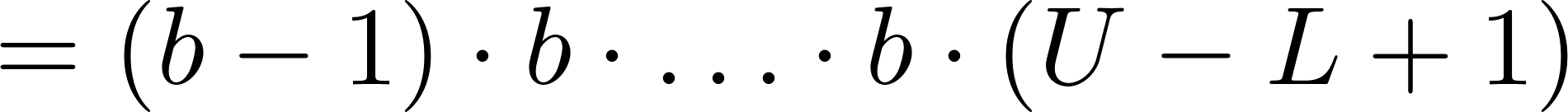
i reali rappresentabili tramite reali-macchina in matlab sono quindi intervalli

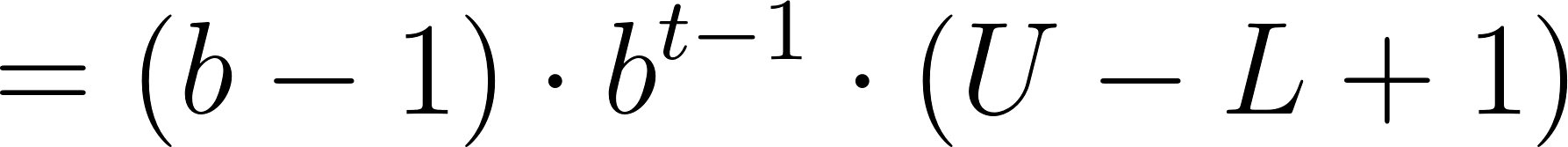
Estremamente ampi, con un estremo "grandissimo" e un estremo "piccolissimo" (in modulo, considerando anche l'intervallo dei negativi)

È importante però ricordare ancora una volta che la rappresentazione avviene per arrotondamento della mantissa e che gli unici numeri effettivamente memorizzabili nel calcolatore sono i reali-macchina, un insieme finito.

Ma qual è la cardinalità (numero di elementi) di [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbb%7BF%7D#0) ? il calcolo è facile, osservando che basta contare gli elementi di [](http://www.texrendr.com/?eqn=%5Cmathbb%7BF%7D%5E%2B#0) e raddoppiare, tenendo poi conto dello 0 che ha una rappresentazione speciale card([](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbb%7BF%7D%5E%2B#0))

= numero possibili mantisse x numero di possibili esponenti.

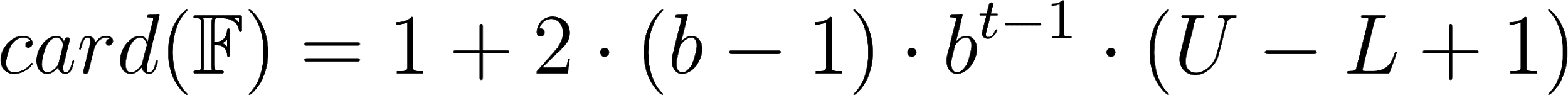
[](https://www.codecogs.com/eqnedit.php?latex=%3D(b-1)%5Ccdot%20b%20%5Ccdot%20%5Cdotso%20%5Ccdot%20b%20%5Ccdot%20(U-L%2B1)#0)

[](https://www.codecogs.com/eqnedit.php?latex=%3D(b-1)%5Ccdot%20b%5E%7Bt-1%7D%5Ccdot%20(U-L%2B1)#0)

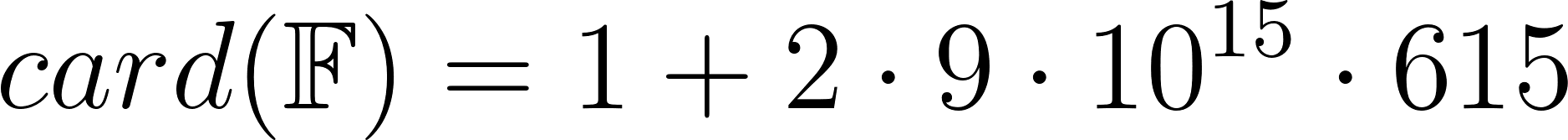
in questo conto si osservi che per la prima cifra di mantissa ci sono b-1 scelte (non può essere 0) mentre ci sono b scelte per tutte le altre cifre(ad es. in base 10 la prima cifra può essere 1, 2, .. ,9 cioè 9 possibili cifre, mentre tutte le altre 0,1,2, … , 9 cioè 10 possibili cifre)

il numero di elementi dell’intervallo di interi [L,U] è U-L+1

alla fine si ottiene

[](https://www.codecogs.com/eqnedit.php?latex=card(%5Cmathbb%7BF%7D)%3D1%2B2%5Ccdot%20(b-1)%20%5Ccdot%20b%5E%7Bt-1%7D%5Ccdot%20(U-L%2B1)#0)

nel modello simil-matlab visto prima con b=10, t=16, L=-308, U= +308 si ottiene

[](https://www.codecogs.com/eqnedit.php?latex=card(%5Cmathbb%7BF%7D)%3D1%2B2%5Ccdot%209%20%5Ccdot%2010%5E%7B15%7D%5Ccdot%20615#0)

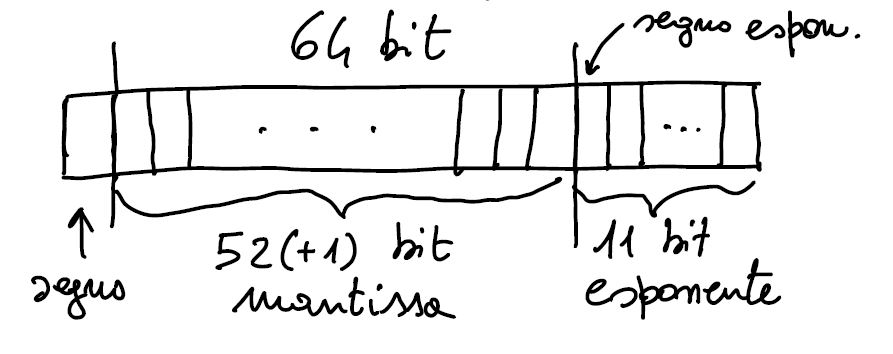
che un numero dell’ordine di 1019

da dove vengono i parametri che abbiamo usato per il modello simil-matlab?

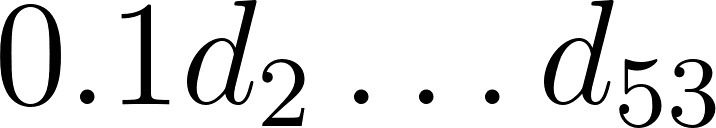
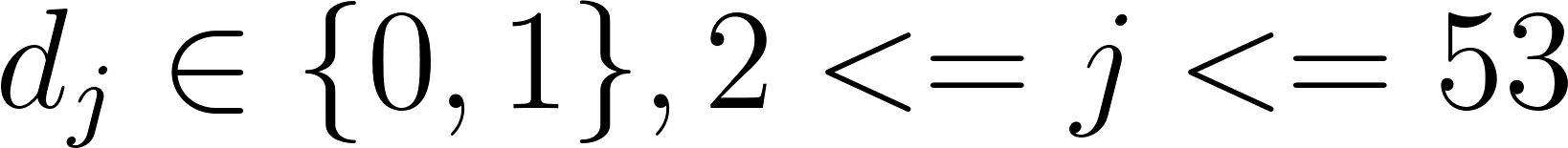
del fatto che il Matlab usa uno standard di rappresentazione

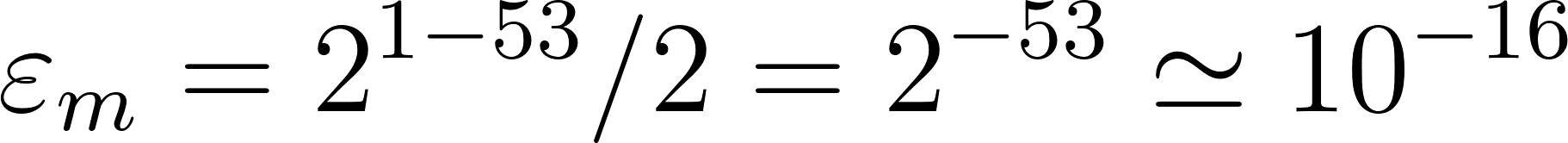
per i reali macchina(lo standard IEEE) in cui ad una variabile reale è dedicata una sequenza di 64 bit (8 byte), in cui 1 bit è riservato al segno, 52 bit della mantissa e 11 bit all’esponente

senza entrare in dettaglio nello standard di rappresentazione, facciamo un semplice modello

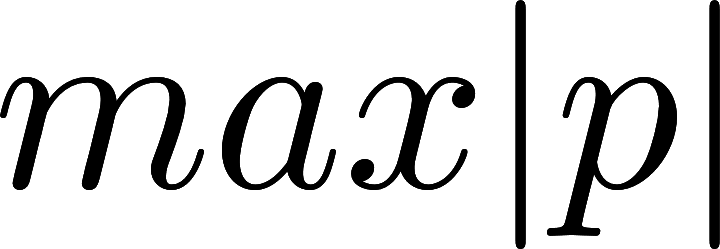


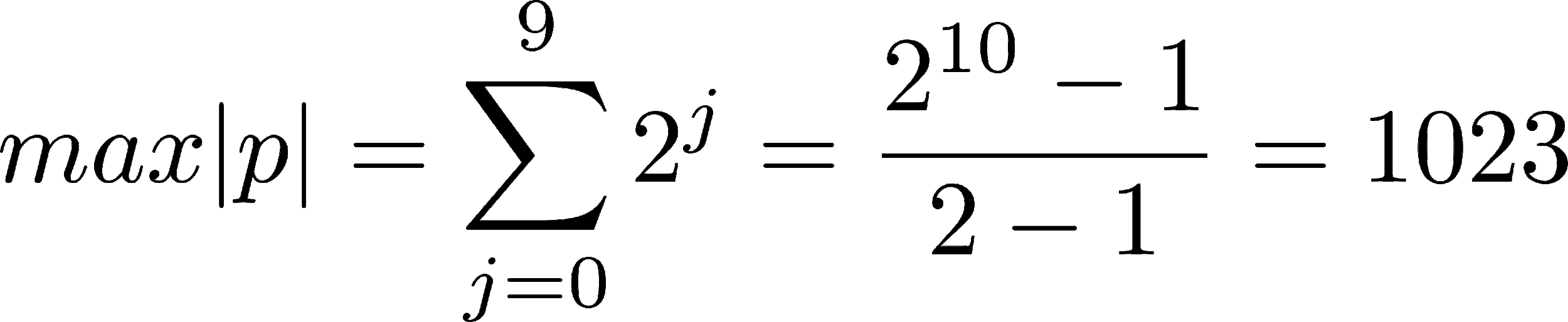
in ciascuna delle caselline (bit) si può scrivere 0 oppure 1 (per il segno uno dei due rappresenta “+” e l’altro “-”)

per quanto riguarda la mantissa i bit sono 52 ma è come se fossero 53, perchè la prima cifra di mantissa deve sempre essere non nulla e quindi per forza 1 (ovvero il processore tratta i numeri come se avessero mantissa [](https://www.codecogs.com/eqnedit.php?latex=0.1d_2%5Cdotso%20d_%7B53%7D#0) con [](https://www.codecogs.com/eqnedit.php?latex=d_j%20%5Cin%20%5C%7B0%2C1%5C%7D%2C%202%3C%3Dj%3C%3D53#0))

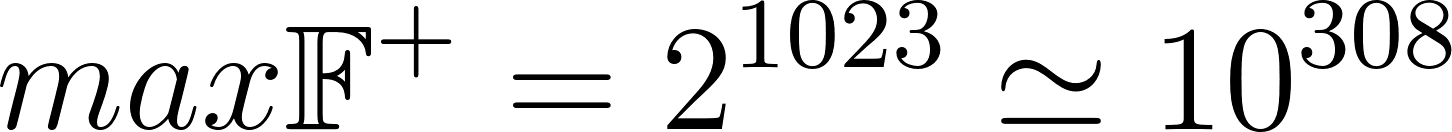
quindi la precisione di macchina è [](https://www.codecogs.com/eqnedit.php?latex=%5Cvarepsilon_m%20%3D%202%5E%7B1-53%7D%2F2%3D2%5E%7B-53%7D%20%5Csimeq%2010%5E%7B-16%7D#0)

per quanto riguarda il range di esponenti, in questo modello semplificato degli 11 bit a disposizione ne teniamo uno per il segno

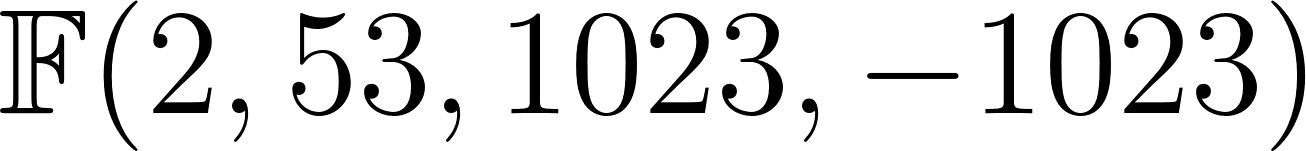
quindi [](http://www.texrendr.com/?eqn=max%20%7Cp%7C#0) si ottiene con la sequenza (1111111111)2 (10 volte 1) ovvero

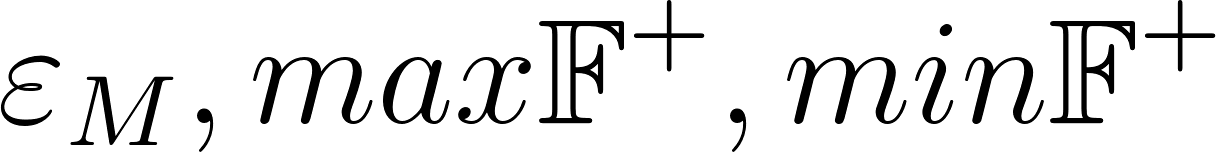
[](http://www.texrendr.com/?eqn=max%20%7Cp%7C%20%3D%20%5Csum%5Climits_%7Bj%3D0%7D%5E9%202%5Ej%20%3D%20%5Cfrac%7B2%5E%7B10%7D-1%7D%7B2-1%7D%3D1023#0)

ricordando le proprietà della somma geometrica(qui con ragione 2 ) viste nella lezione 1 allora U =+1023, L=-1023

e [](https://www.codecogs.com/eqnedit.php?latex=max%5Cmathbb%7BF%7D%5E%2B%3D2%5E%7B1023%7D%20%5Csimeq%2010%5E%7B308%7D#0)

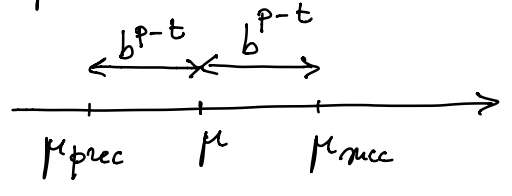
[](http://www.texrendr.com/?eqn=min%5Cmathbb%7BF%7D%5E%2B%3D2%5E%7B-1023%7D%20%5Csimeq%2010%5E%7B-308%7D#0)

(In realtà la codifica dell’esponente usa una tecnica un po’ più sofisticata ma la sostanza cambia poco) alla fine possiamo dire che in MatLab (e molti altri linguaggi) i reali macchina sono sequenze a 64 bit (la cosiddetta precisione doppia) corrispondenti sostanzialmente a [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbb%7BF%7D(2%2C53%2C1023%2C-1023)#0).

Siccome l’interfaccia dei linguaggi con l’utente è in base 10 (con conversione automatica dell’input/output) per quanto visto sopra possiamo dire che lavorando in Matlab è essenzialmente ( ma non esattamente) come se utilizzassimo  nel senso che l’ordine di grandezza dei parametri chiave  è corretto (ma scopriremo che talvolta bisogna tener presente che la base “vera” è 2).

Finora abbiamo risposto a due delle tre domande iniziali sulle strutture del sistema floating-point, resta da analizzare come sono distribuiti i reali-macchina. Il concetto chiave è che non sono distribuiti uniformemente ma bensi “a densità variabile”, i reali-macchina “piccoli” sono vicini, i reali-macchina “grandi” sono distanti tra di loro, la densità cresce andando verso , decresce andando verso .

Questo si può capire bene calcolando la distanza tra reali-macchina consecutivi (con lo stesso esponente p) graficamente (ad esempio in )



infatti due reali-macchina consecutivi differiscono di 1 nella t-esima cifra di mantissa, ovvero

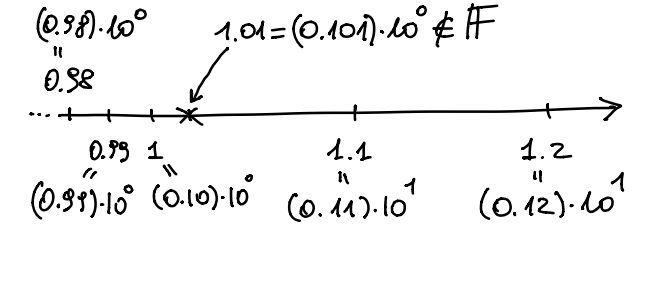
mantissa  - mantissa 

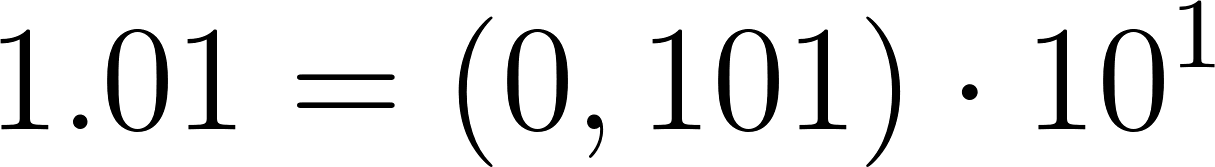
e 

la distanza (assoluta) è quindi variabile con p, cioè con l’ordine di grandezza del numero in base b.

Questa distanza varia (di un fattore b) quando si passa per una potenza della base.

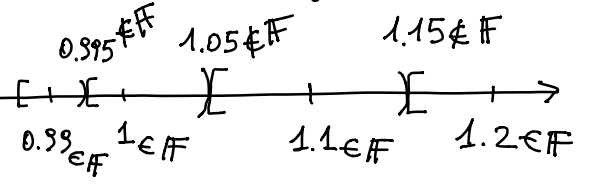
Ad esempio con b=10 e t=2 (due cifre di mantissa)

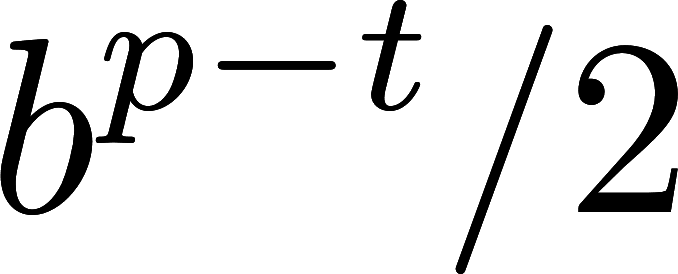


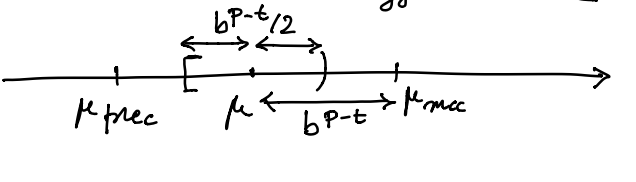
la domanda è: chi è il reale-macchina successivo ad 1? Pensando a 2 cifre decimali, verrebbe da rispondere 1.01 ma NON è così, perché [](https://www.codecogs.com/eqnedit.php?latex=1.01%3D(0%2C101)%5Ccdot%2010%5E1#0) ha 3 cifre di mantissa e in questo esempio t=2

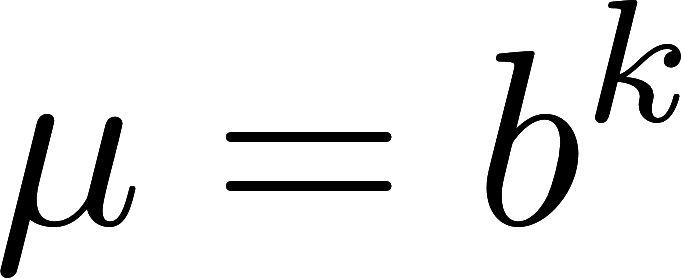
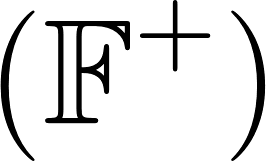
invece il reale-macchina successivo ad 1 è  passando per 1=100 la distanza è cresciuta di un fattore 10, da 10-1 a 10-1

Pensando invece a quali reali sono approssimati da 0.99, 1 e 1.1 abbiamo graficamente



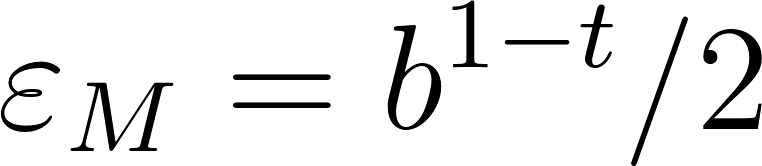
vediamo che ogni reale-macchina è il centro di un intorno di approssimazione a t cifre di mantissa (qui t=2): questi intorni sono generalmente simmetrici e di raggio [](http://www.texrendr.com/?eqn=b%5E%7Bp-t%7D%2F2#0)



tranne per [](http://www.texrendr.com/?eqn=%5Cmu%20%3D%20b%5Ek#0) , nel qual caso l’intorno è asimmetrico e l’intorno destro in [](https://www.codecogs.com/eqnedit.php?latex=(%5Cmathbb%7BF%7D%5E%2B)#0) si allunga di un fattore b.

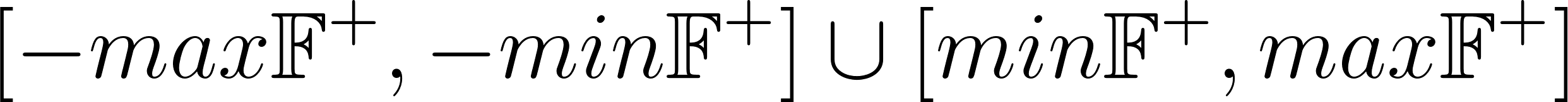
Tutti i reali di questi intorni vengono approssimati del reale-macchina centro dell’intorno per arrotondamento a t cifre di mantissa, con

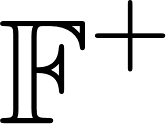
errore assoluto <= (b^(p-t))/2 = raggio

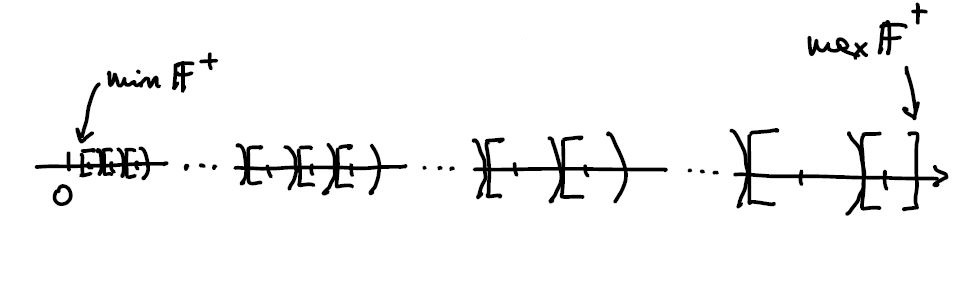
ma se andiamo a stimare l’errore relativo, sappiamo dalla lezione 2 che questo non può superare la precisione di macchina [](https://www.codecogs.com/eqnedit.php?latex=%5Cvarepsilon_M%20%3D%20b%5E%7B1-t%7D%2F2#0)

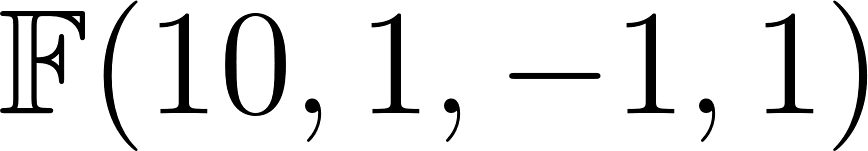
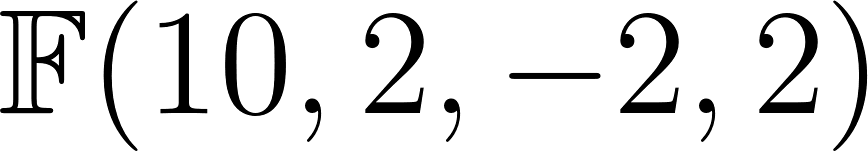
che è indipendente da p.

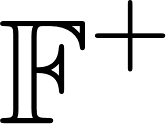
L’unionione di tutti gli intorni di approssimazione per arrotondamento copre tutti gli intervalli dei reali rappresentabili, ovvero

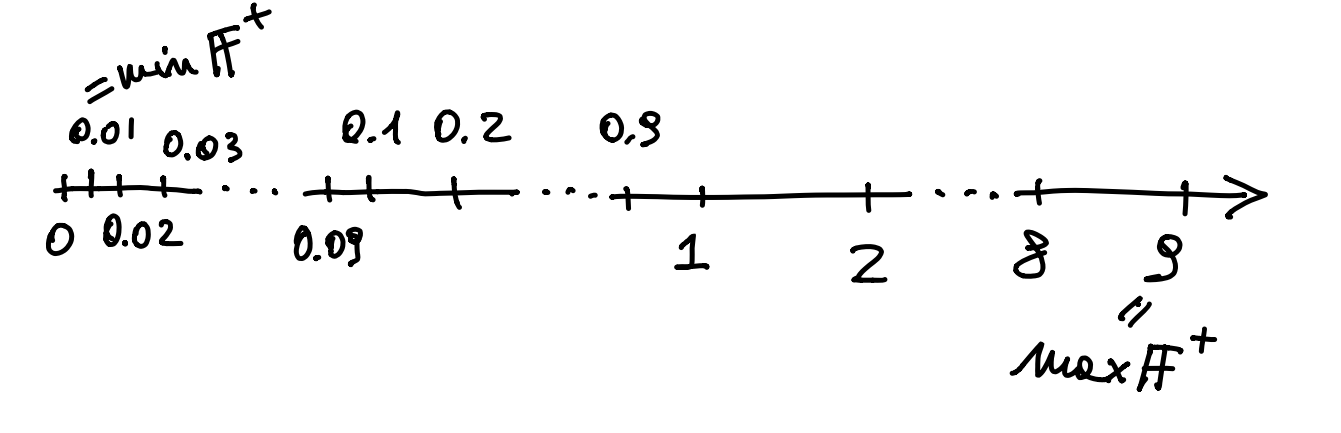
[](https://www.codecogs.com/eqnedit.php?latex=%5B-max%5Cmathbb%7BF%7D%5E%2B%2C%20-min%5Cmathbb%7BF%7D%5E%2B%5D%20%5Ccup%20%5Bmin%5Cmathbb%7BF%7D%5E%2B%2C%20max%5Cmathbb%7BF%7D%5E%2B%5D#0)

come descritto in modo un po’ ingenuo da questo disegno (per [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbb%7BF%7D%5E%2B#0) )

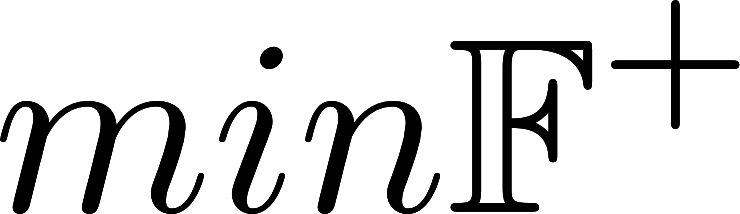


Concludiamo questa lezione con due esempi semplici in base 10: disegnare [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbb%7BF%7D(10%2C%201%2C%20-1%2C%201)#0) e [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbb%7BF%7D(10%2C%202%2C%20-2%2C%202)#0).

Nel primo caso abbiamo una sola cifra di mantissa ( il minimo possibile) ed esponenti p= -1, 0, 1 disegniamo [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbb%7BF%7D%5E%2B#0) (scrivendo per semplicità i reali-macchina in notazione standard)



vediamo che si parte dall’ordine di grandezza dei centesimi con   
min = 0.1 ⋅ 10-1 = 0.01 (tra i centesimi non ci sono reali-macchina perché c’è una sola cifra di mantissa), si passa ai decimi, poi alle unità sempre senza reali-macchina intermedi, fino a max  = (0.9) ⋅ 101 = 9 la precisione di macchina è si tratta di un sistema floating-point molto “povero”, perché c’è una sola cifra di mantissa e un’estensione molto limitata dal piccolo range di esponenti. Lascio per esercizio di provare a disegnare (10 , 2 , -2 , 2) (converrà fare degli opportuni “zoom” sui vari ordini di grandezza) il sistema è più “ricco” del precedente.

Abbiamo [](http://www.texrendr.com/?eqn=min%5Cmathbb%7BF%7D%5E%2B#0)= bL-1 = 10-3

[](https://www.codecogs.com/eqnedit.php?latex=max%5Cmathbb%7BF%7D%5E%2B#0) = (1 - b-t) ⋅ bu

= (1-10-2) ⋅ 102 = 99

εM= b1-t / 2 = 10-1 / 2 = 5%

si parte dai millesimi, ma fra questi ci sono i decimillesimi perché abbiamo 2 cifre di mantissa, poi ai centesimi con i millesimi, i decimi con i centesimi, le unità con i decimi e infine le decine con le unità. Questo dovrebbe far capire quanto ricca è la struttura del sistema floating-point a 64 bit (precisione doppia) è il caso di ricordare che esistono altri standard, la precisione singola (a 32 bit) con e la precisione quadrupla (a 128 bit) con

Quale sia la precisione di default dipende dal linguaggio, alcuni permettono di scegliere tra singola, doppia e quadrupla. Queste precisioni corrispondono ad operazioni aritmetiche implementate a livello hardware (circuiti) nel processore, quindi estremamente veloci. In alcuni linguaggi è possibile aumentare la precisione (cioè aumentare il numero di cifre di mantissa), ma allora le operazioni aritmetiche sono implementate a livello software (c’è un programma che calcola somma, moltiplicazione, e divisione, non un’unità hardware), quindi il costo delle operazioni cresce e cresce ovviamente l’occupazione di memoria, soprattutto elaborando masse di dati. Questo è il motivo per cui si è fatto un compromesso e lo standard attuale è la precisione doppia con , un massimo errore relativo di arrotondamento che è di solito molto più piccolo degli errori di misura sperimentali dei dati nelle applicazioni. Ciononostante, vedremo che la propagazione degli errori in algoritmi instabili può far perdere anche tutta questa precisione nel processo di calcolo. Questo conduce in modo naturale allo studio della STABILITÀ degli algoritmi numerici, che cominceremo già nella prossima lezione sulle operazioni aritmetiche con numeri approssimati.