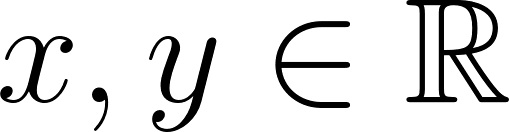
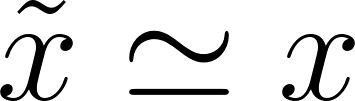
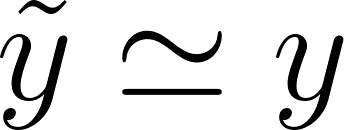
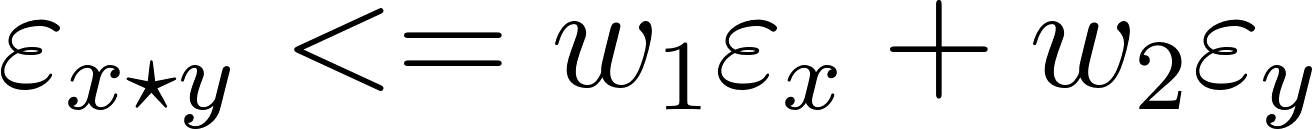
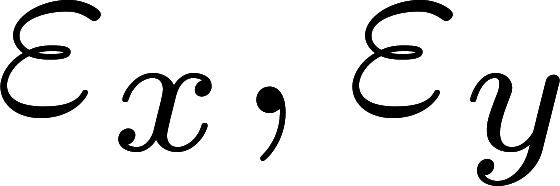
**LEZIONE 5: ESEMPI DI INSTABILITÀ DELLA SOTTRAZIONE**

Nella scorsa lezione abbiamo analizzato la risposta delle operazioni aritmetiche agli errori sui dati. In particolare, dati [](https://www.codecogs.com/eqnedit.php?latex=x%2Cy%20%5Cin%20%5Cmathbb%7BR%7D#0) e [](https://www.codecogs.com/eqnedit.php?latex=%5Ctilde%7Bx%7D%20%5Csimeq%20x#0), [](https://www.codecogs.com/eqnedit.php?latex=%5Ctilde%7By%7D%20%5Csimeq%20y#0), i risultati ottenuti si possono sintetizzare con la disuguaglianza [](https://www.codecogs.com/eqnedit.php?latex=%5Cvarepsilon_%7Bx%5Cstar%20y%7D%20%3C%3D%20w_1%5Cvarepsilon_x%2Bw_2%5Cvarepsilon_y#0) dove [](http://www.texrendr.com/?eqn=%5Cvarepsilon_%7Bx%20%5Cstar%20y%7D#0) è l’errore relativo il risultato dell’operazione [](https://www.codecogs.com/eqnedit.php?latex=%5Cstar#0) e [](http://www.texrendr.com/?eqn=%5Cvarepsilon_x%2C%20%5Cvarepsilon_y#0) gli errori relativi sui dati. I PESI w1,w2>0 possono dipendere da x,y (ma non dagli errori).

Nel caso di moltiplicazione, divisione e addizione si ha w1,w2oppure w1,w2<=1 quindi tali operazioni risultano STABILI.

Nel caso della sottrazione i pesi w1=|x|/|x+y|, w2=|y|/|x+y| con sgn(x)=-sgn(y)

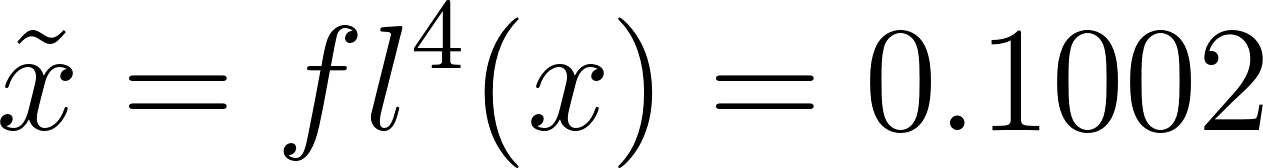
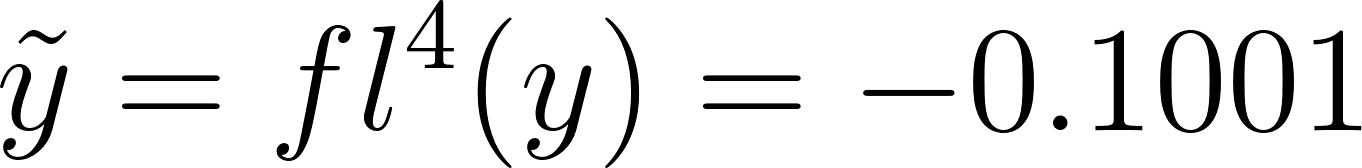
possono essere invece grandi.

Questo accade in particolare se x e y sono “vicini” in termini relativi, ovvero |x+y|<<|x|,|y| quindi la SOTTRAZIONE è POTENZIALMENTE INSTABILE e in grado di diminuire in modo consistente nel risultato la precisione dei dati e anche di distruggerla completamente, rendendo il risultato praticamente privo di significato (quando w1,w2> max{1/x,1/y} per cui ci si può attendere un errore > 100%).

Faremo ora alcuni esempi significativi di perdita di precisione dovuta ad instabilità della sottrazione. Lavoreremo come sempre per semplicità in sistemi floating-point virtuali in base 10.

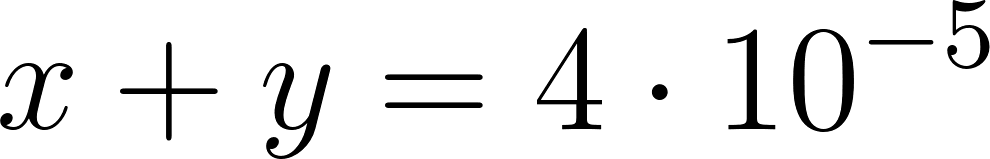
**ESEMPIO 1**

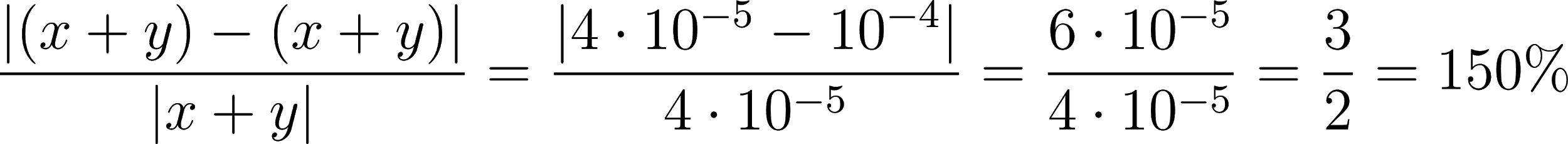
Consideriamo F(10,4,L,U) (con L,U sufficienti per rappresentare i numeri che ci interessano) e x=0.10016 y=-0.10012

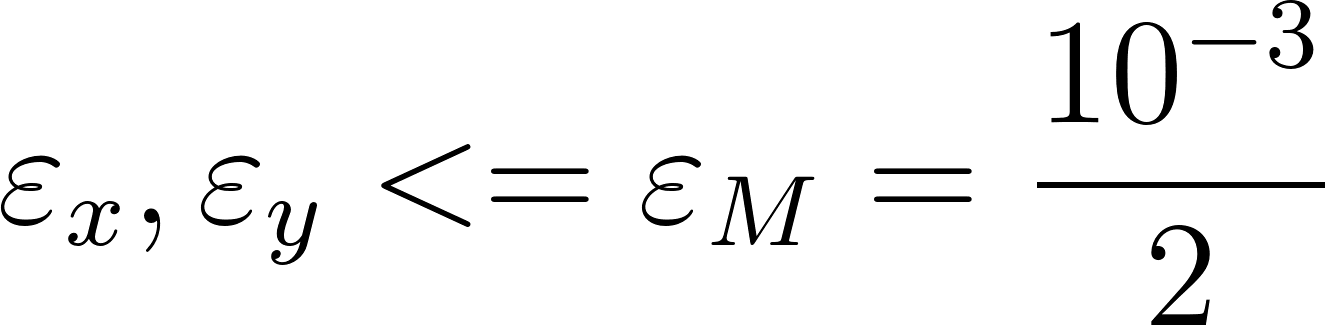
allora [](https://www.codecogs.com/eqnedit.php?latex=%5Ctilde%7Bx%7D%3Dfl%5E4(x)%3D0.1002#0) e [](https://www.codecogs.com/eqnedit.php?latex=%5Ctilde%7By%7D%3Dfl%5E4(y)%3D-0.1001#0)

eseguendo l’operazione-macchina di somma algebrica (che è una sottrazione visto che x e y hanno segno opposto) si ottiene

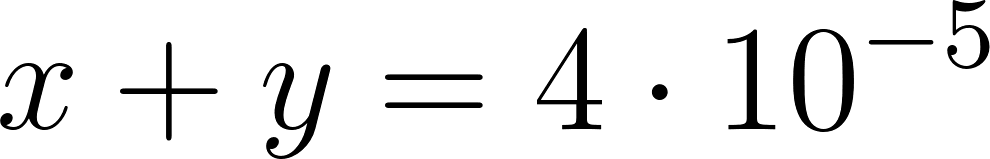
444=4-4

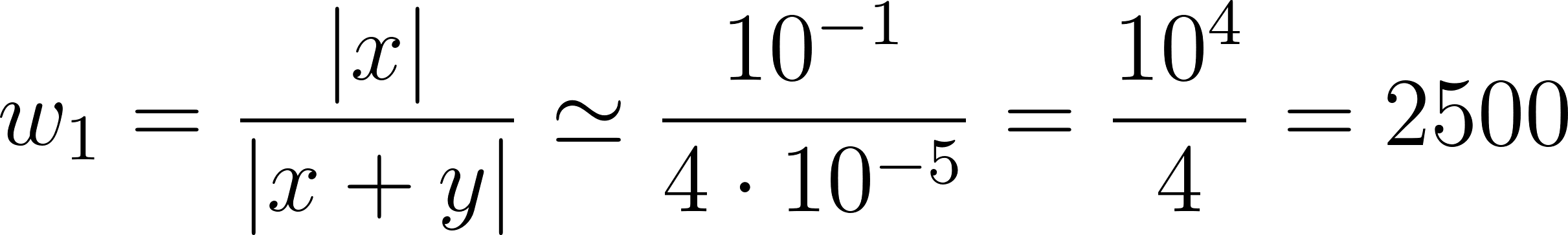
(scriveremo spesso i numeri in notazione standard per comodità) invece [](https://www.codecogs.com/eqnedit.php?latex=x%2By%3D4%5Ccdot%2010%5E%7B-5%7D#0) quindi l’errore relativo nel risultato è

[](https://www.codecogs.com/eqnedit.php?latex=%5Cfrac%7B%7C(x%2By)-(x%2By)%7C%7D%7B%7Cx%2By%7C%7D%20%3D%20%5Cfrac%7B%7C4%5Ccdot%2010%5E%7B-5%7D%20-%2010%5E%7B-4%7D%7C%7D%7B4%5Ccdot%2010%5E%7B-5%7D%7D%20%3D%20%5Cfrac%7B6%5Ccdot%2010%5E%7B-5%7D%7D%7B4%5Ccdot%2010%5E%7B-5%7D%7D%20%3D%20%5Cfrac%7B3%7D%7B2%7D%20%3D%20150%5C%25#0)

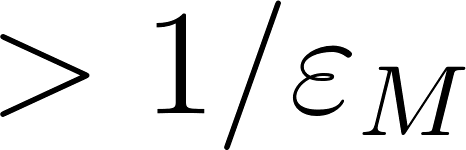
a fronte di [](https://www.codecogs.com/eqnedit.php?latex=%5Cvarepsilon_x%2C%20%5Cvarepsilon_y%20%3C%3D%20%5Cvarepsilon_M%20%3D%20%5Cfrac%7B10%5E%7B-3%7D%7D%7B2%7D#0)

Abbiamo un errore del 150% e una perdita di precisione di ben tre ordini di grandezza rispetto alla precisione di macchina.

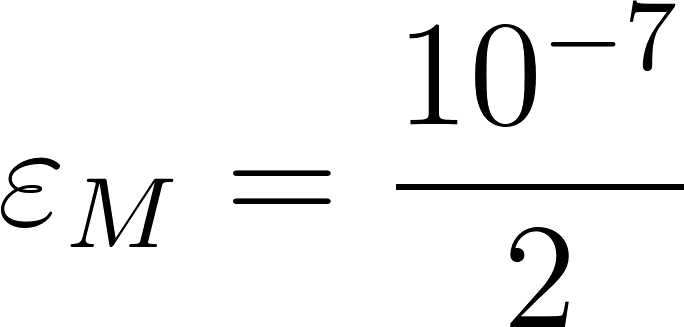
Qui il problema sta nella sottrazione tra numeri vicini, visto che [](https://www.codecogs.com/eqnedit.php?latex=x%2By%3D4%5Ccdot%2010%5E%7B-5%7D#0) ma |x|,|y|-1

Infatti se calcoliamo i pesi [](https://www.codecogs.com/eqnedit.php?latex=w_1%20%3D%20%5Cfrac%7B%7Cx%7C%7D%7B%7Cx%2By%7C%7D%20%5Csimeq%20%5Cfrac%7B10%5E%7B-1%7D%7D%7B4%5Ccdot%2010%5E%7B-5%7D%7D%20%3D%20%5Cfrac%7B10%5E4%7D%7B4%7D%20%3D%202500#0)

e analogamente w22500

Questi fattori di amplificazione degli errori sui dati sono dell’ordine di 103 e spiegano come si arrivi ad un errore finale>100%, che rende inaccettabile in pratica il risultato in questo caso i fattori di amplificazione non sono enormi, ma sono comunque [](https://www.codecogs.com/eqnedit.php?latex=%3E%201%2F%5Cvarepsilon_M#0) osserviamo che bastava una 1 cifra di mantissa in più per avere il risultato esatto, perchè con t=5 non ci sarebbe stato bisogno di arrotondare x e y e quindi [](https://www.codecogs.com/eqnedit.php?latex=%5Cvarepsilon_x%3D%5Cvarepsilon_y%3D0#0) questo ci apre la strada all’analisi del prossimo esempio, un po’ più sofisticato.

**ESEMPIO 2**

Consideriamo F(10,8,L,U) (con L e U appropriati) qui t=8 e [](https://www.codecogs.com/eqnedit.php?latex=%5Cvarepsilon_M%3D%5Cfrac%7B10%5E%7B-7%7D%7D%7B2%7D#0)si tratta di calcolare in questa aritmetica floating-point la somma algebrica a+b+c

dove 





osserviamo che 

vedremo ora che in questo esempio non vale la proprietà associativa, infatti

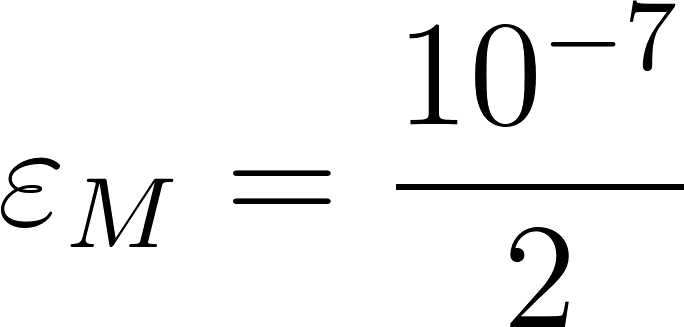


invece

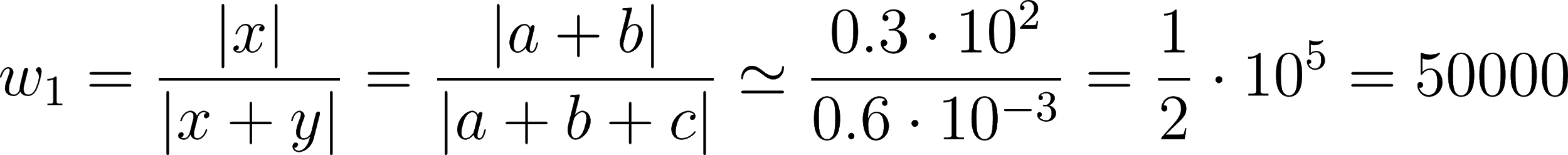


quindi ; inoltre si verifica (ad esempio in Matlab cioè sostanzialmente a 16 cifre decimali) che cioè il risultato  è il meglio che si può ottenere in questa aritmetica floating-point, mentre l’errore relativo di 1 è

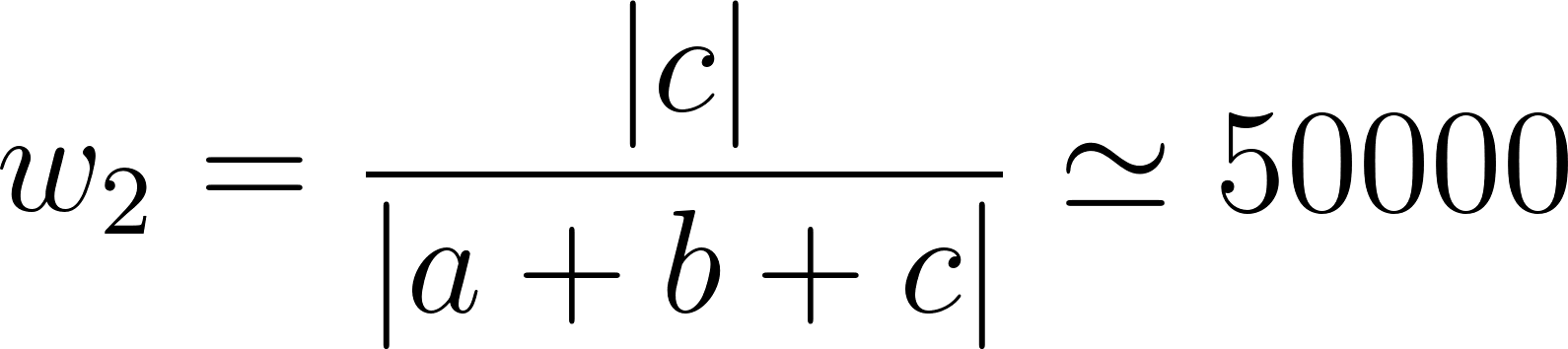


qui riusciamo subito a spiegare la perdita di precisione di 4 ordini di grandezza rispetto a [](https://www.codecogs.com/eqnedit.php?latex=%5Cvarepsilon_M%3D%5Cfrac%7B10%5E%7B-7%7D%7D%7B2%7D#0), calcolando i pesi di  e  associati alla sottrazione

(a+b)+c, visto che a+b e c sono vicini in termini relativi (hanno le prime 4 cifre significative coincidenti)

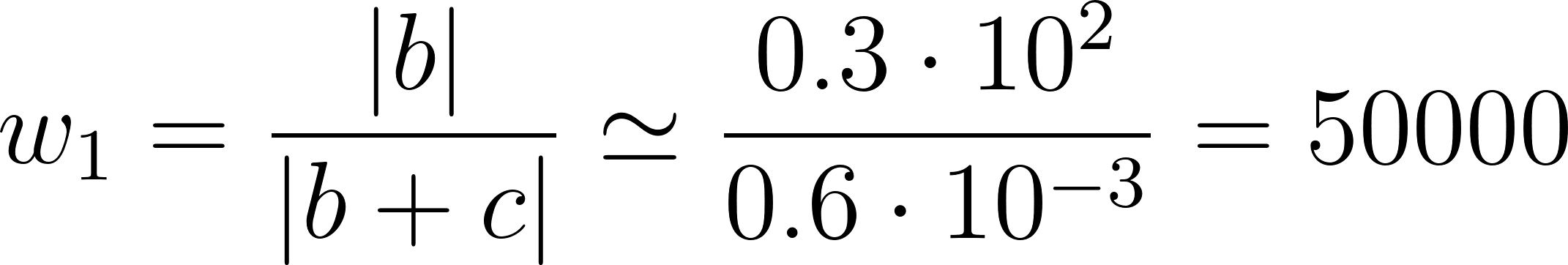
[](https://www.codecogs.com/eqnedit.php?latex=w_1%20%3D%20%5Cfrac%7B%7Cx%7C%7D%7B%7Cx%2By%7C%7D%20%3D%20%5Cfrac%7B%7Ca%2Bb%7C%7D%7B%7Ca%2Bb%2Bc%7C%7D%20%5Csimeq%20%5Cfrac%7B0.3%20%5Ccdot%2010%5E2%7D%7B0.6%5Ccdot%2010%5E%7B-3%7D%7D%20%3D%20%5Cfrac%7B1%7D%7B2%7D%5Ccdot%2010%5E5%20%3D%2050000#0)

analogamente

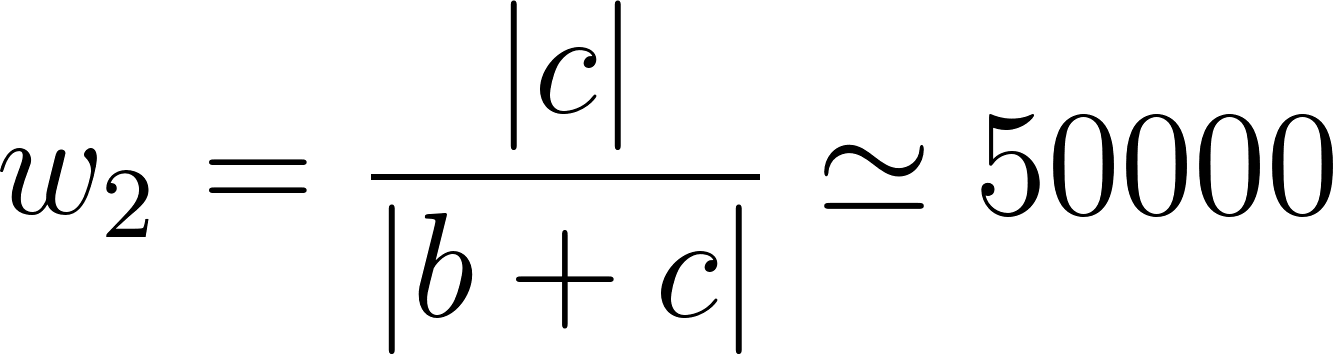
[](https://www.codecogs.com/eqnedit.php?latex=w_2%20%3D%20%5Cfrac%7B%7Cc%7C%7D%7B%7Ca%2Bb%2Bc%7C%7D%20%5Csimeq%2050000#0)

si osservi che in queste stime abbiamo lavorato con 1 cifra significativa, sia con gli errori che con i pesi, perché di queste quantità non ci interessa il valore accurato ma solo l’ordine di grandezza.

Naturalmente, anche nel caso II c’è una sottrazione che viene fatta subito, ovvero b+c. Posto x=b, y=c si ha

[](https://www.codecogs.com/eqnedit.php?latex=w_1%20%3D%20%5Cfrac%7B%7Cb%7C%7D%7B%7Cb%2Bc%7C%7D%20%5Csimeq%20%5Cfrac%7B0.3%5Ccdot%2010%5E2%7D%7B0.6%5Ccdot%2010%5E%7B-3%7D%7D%3D50000#0)

analogamente

[](https://www.codecogs.com/eqnedit.php?latex=w_2%20%3D%20%5Cfrac%7B%7Cc%7C%7D%7B%7Cb%2Bc%7C%7D%20%5Csimeq%2050000#0)

ma allora, perché con l’espressione di calcolo II non c’è perdita di precisione? La spiegazione è sottile e sta nel fatto che i numeri a, b, c entrano con 8 cifre significative e non occorre arrotondarli; cioè  quindi la sottrazione b+c non perde precisione (invece ne perderebbe se b oppure c fossero arrotondati)

Invece nell’espressione di calcolo  la sottrazione avviene con uno dei due dati arrotondato, ovvero a+b, che è un’addizione il cui risultato viene comunque arrotondato perché ha più di 8 cifre significative (cifre di mantissa) e quell’errore di arrotondamento viene amplificato da peso  (mentre il peso  moltiplica  e quindi non ha effetto).

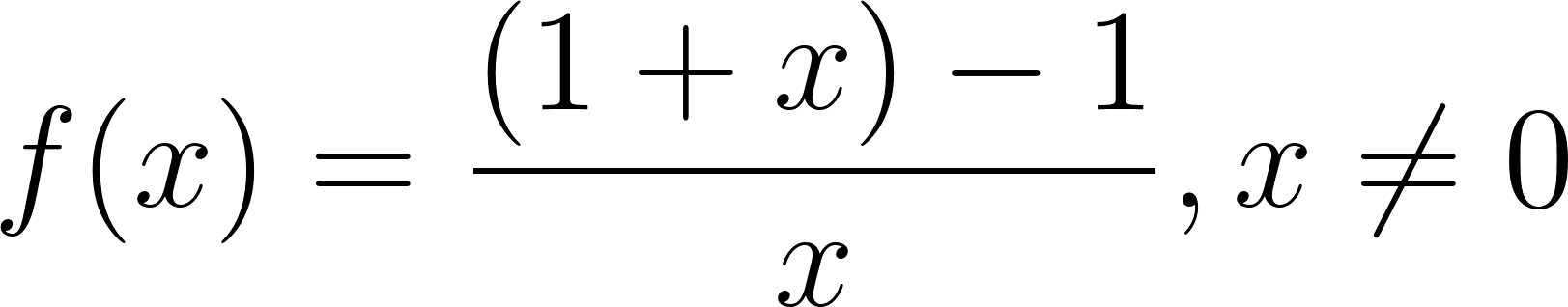
Osserviamo che in una sottrazione instabile basta che uno dei due dati sia affetto da errore per vedere la perdita di precisione

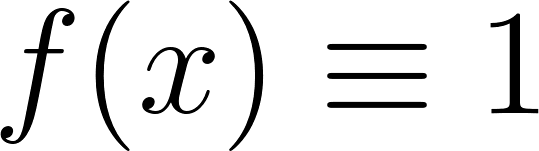
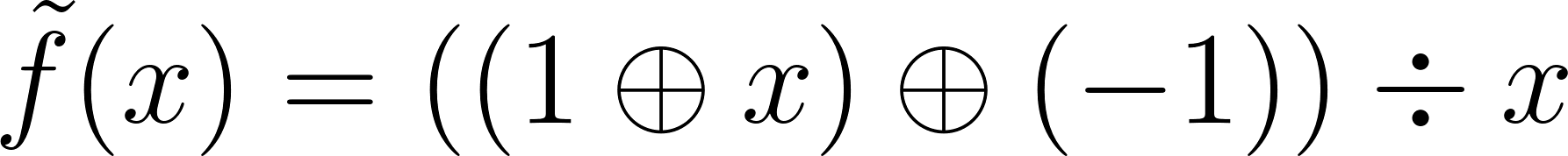
Siamo in grado a questo punto di discutere l’esempio portato alla fine della lezione 4, rispondendo alle domande:

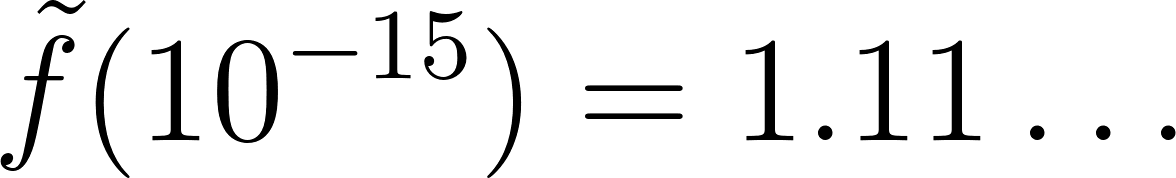
perchè in Matlab  ha errore >11% e  è “esatto”?

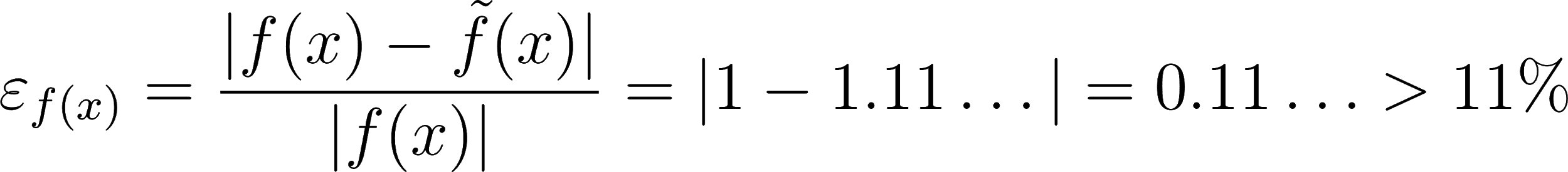
**ESEMPIO 3**

si consideri il calcolo in Matlab della funzione

[](https://www.codecogs.com/eqnedit.php?latex=f(x)%20%3D%20%5Cfrac%7B(1%2Bx)-1%7D%7Bx%7D%2C%20x%20%5Cneq%200#0)

Chiaramente [](https://www.codecogs.com/eqnedit.php?latex=f(x)%20%5Cequiv%201#0) (è la funzione costante 1), in Matlab però con le operazioni macchina si calcola [](https://www.codecogs.com/eqnedit.php?latex=%5Ctilde%7Bf%7D(x)%20%3D%20((1%20%5Coplus%20x)%20%5Coplus%20(-1))%20%5Cdiv%20x#0) (divisione macchina, simbolo cerchiato)

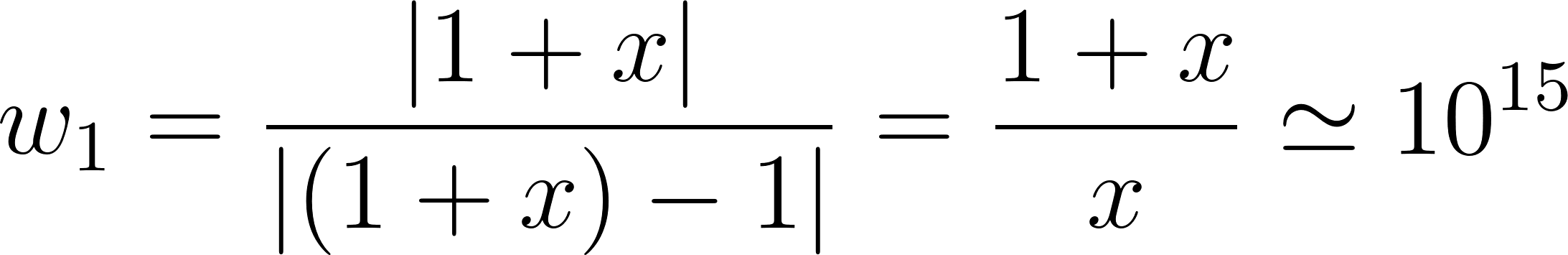
e si ha [](https://www.codecogs.com/eqnedit.php?latex=%5Ctilde%7Bf%7D(10%5E%7B-15%7D)%20%3D%201.11%20%5Cdotso#0) cioè l’errore relativo nel calcolo di f è

[](https://www.codecogs.com/eqnedit.php?latex=%5Cvarepsilon_%7Bf(x)%7D%20%3D%20%5Cfrac%7B%7Cf(x)-%5Ctilde%7Bf%7D(x)%7C%7D%7B%7Cf(x)%7C%7D%20%3D%20%7C1-1.11%20%5Cdotso%7C%20%3D%200.11%5Cdotso%20%3E%2011%5C%25#0)

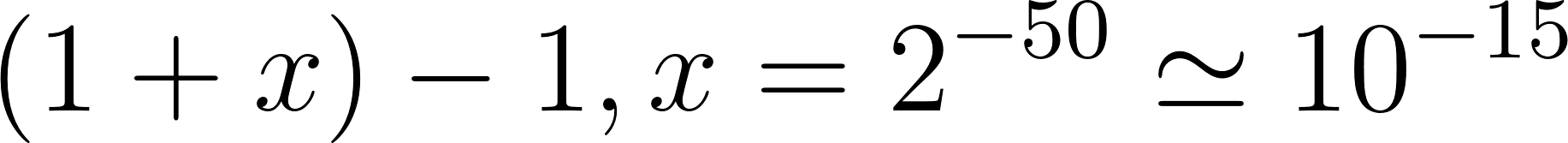
la spiegazione sta nella sottrazione che avviene tra due numeri estremamente vicini, cioè 1 + x e 1 (invece le altre operazioni, un’addizione e una divisione, sono stabili) mentre 1 è un reale-macchina e non viene quindi arrotondato, x=10^-15 viene arrotondato (non lo sarebbe se la base fosse 10, ma non bisogna mai dimenticare che in Matlab la base vera è 2), così come 1 + 10-15.

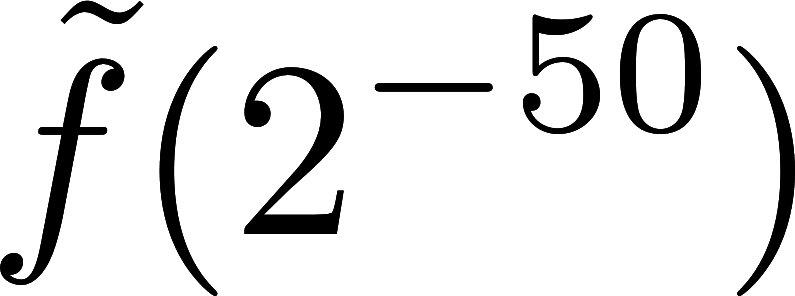
Questi piccolissimi errori di arrotondamento (ricordiamo che sono <= 2-53 circa 10-16) vengono però amplificati dal peso w1 nella sottrazione (1 + x) - 1 , x = 10-15

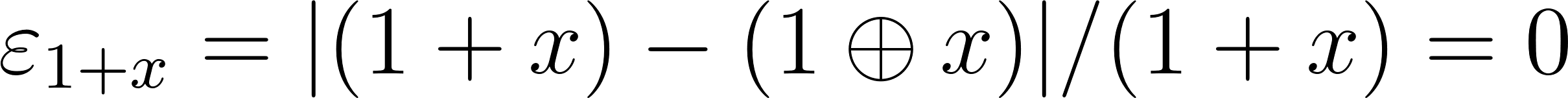
Calcoliamo

[](https://www.codecogs.com/eqnedit.php?latex=w_1%20%3D%20%5Cfrac%7B%7C1%2Bx%7C%7D%7B%7C(1%2Bx)-1%7C%7D%20%3D%20%5Cfrac%7B1%2Bx%7D%7Bx%7D%20%5Csimeq%2010%5E%7B15%7D#0)

che spiega perfettamente come si siano persi 15 ordini di grandezza nel calcolo di f.

Dall’altra parte il peso w1 nel caso della sottrazione [](https://www.codecogs.com/eqnedit.php?latex=(1%2Bx)-1%2C%20x%3D2%5E%7B-50%7D%20%5Csimeq%2010%5E%7B-15%7D#0) è sempre dell’ordine di 1015.

Per quale motivo allora [](http://www.texrendr.com/?eqn=%5Ctilde%7Bf%7D(2%5E%7B-50%7D)#0), cioè in questo caso il risultato è esatto?

Perchè 2-50 e 1+2-50 sono entrambi reali-macchina in base 2 e perciò non vengono arrotondati, cioè [](https://www.codecogs.com/eqnedit.php?latex=%5Cvarepsilon_%7B1%2Bx%7D%20%3D%20%7C(1%2Bx)-(1%5Coplus%20x)%7C%20%2F%20(1%2Bx)%20%3D%200#0)

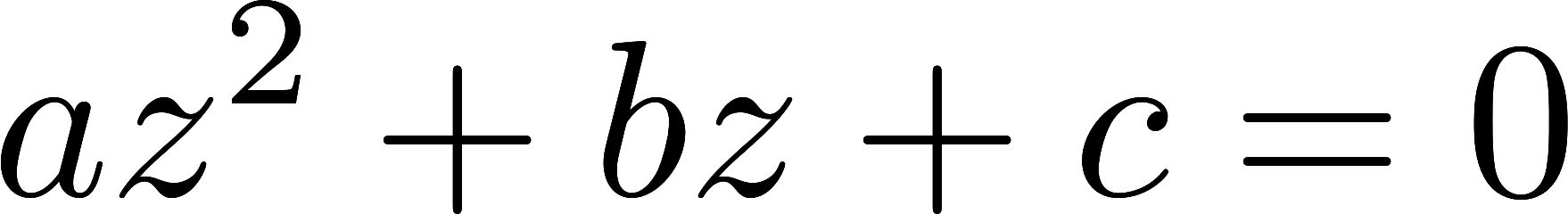
e il fattore di amplificazione non ha effetto.

Come ultimo esempio di possibile instabilitá della sottrazione ci occuperemo ora della formula risolutiva per le equazioni di 2º grado, formula che siamo abituati ad usare senza farci problemi.

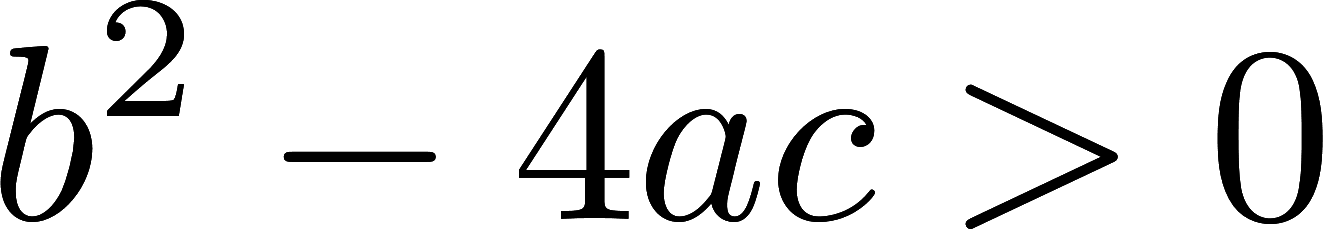
Scopriremo invece che in certe situazioni questa formula in aritmetica floating-point può perdere moltissima precisione, fino a fare perdere del tutto di significato al risultato vedremo però anche che la formula può essere convenientemente “stabilizzata” eliminando la sottrazione potenzialmente instabile

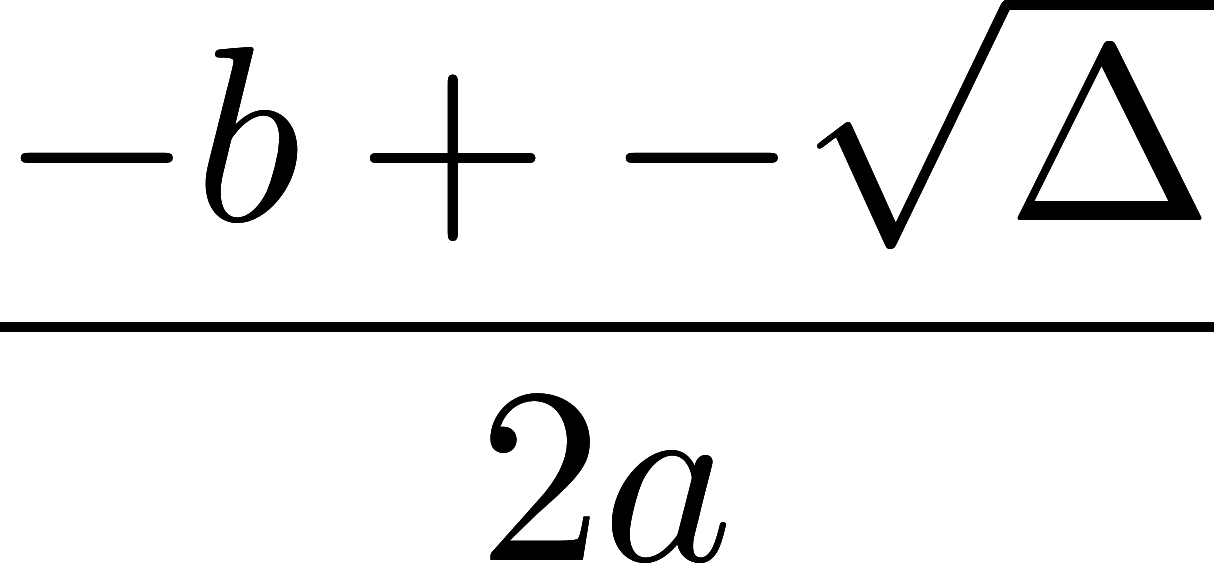
**ESEMPIO 4**

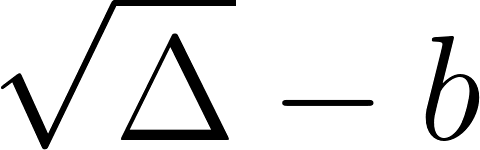
Consideriamo un’equazione di grado effettivo 2

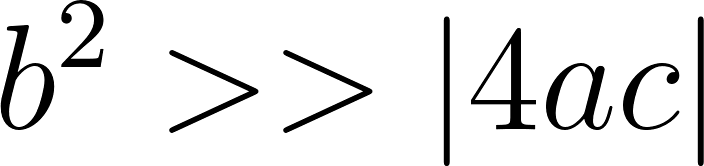
[](http://www.texrendr.com/?eqn=az%5E2%2Bbz%2Bc%3D0#0), a≠0

nel caso di discriminante

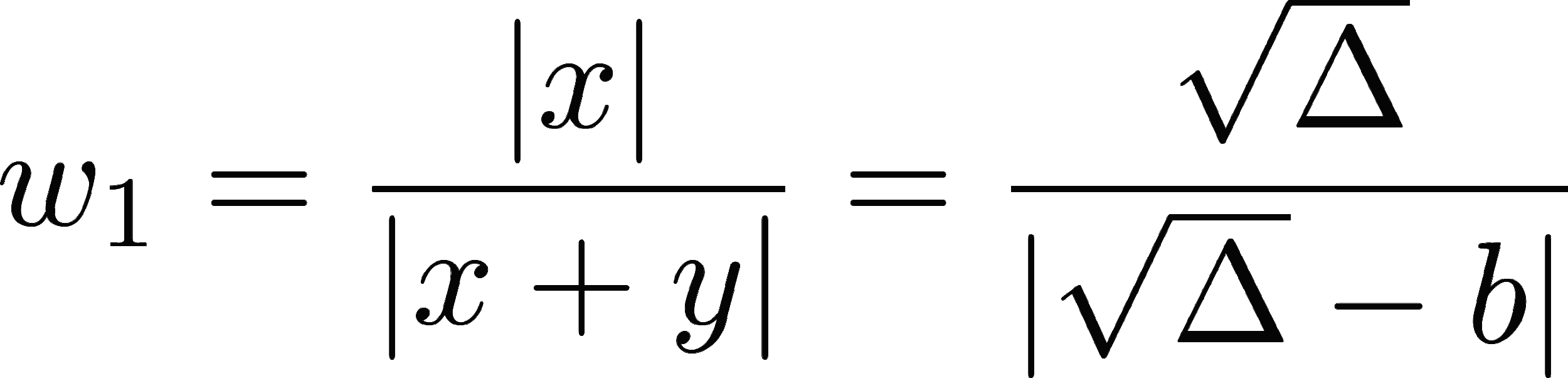
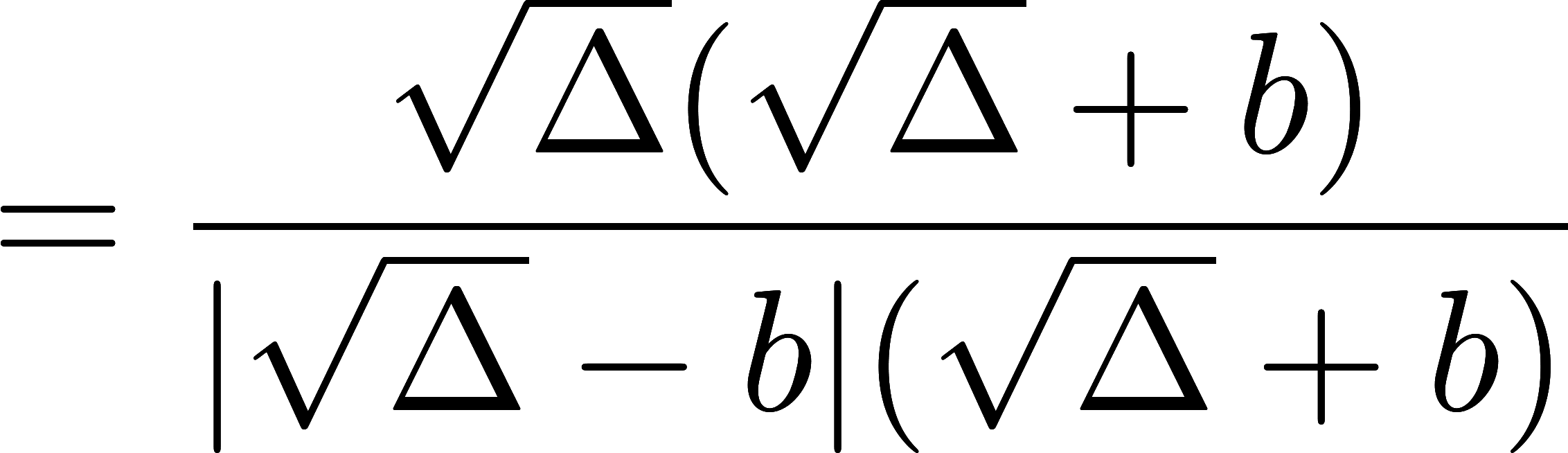
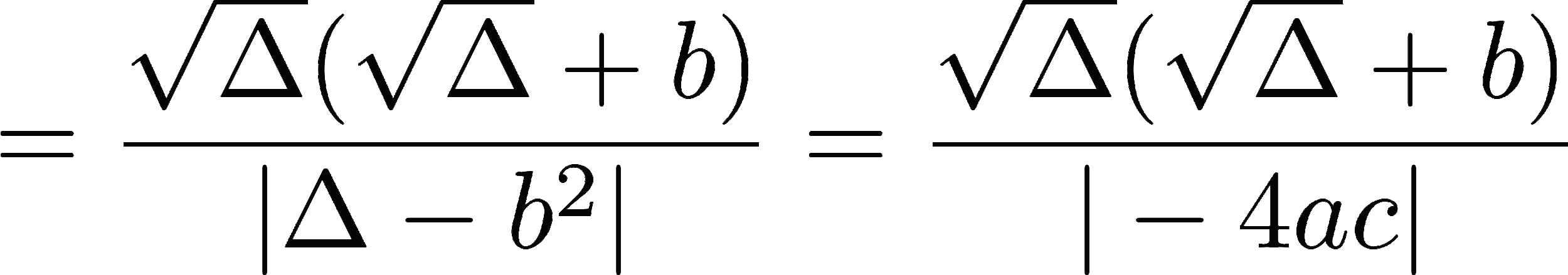
Δ = [](http://www.texrendr.com/?eqn=b%5E2%20-4ac%20%3E%200#0), le cui soluzioni si scrivono abitualmente

come z ± = [](http://www.texrendr.com/?eqn=%5Cfrac%7B-b%20%2B-%5Csqrt%7B%5CDelta%7D%7D%7B2a%7D#0)

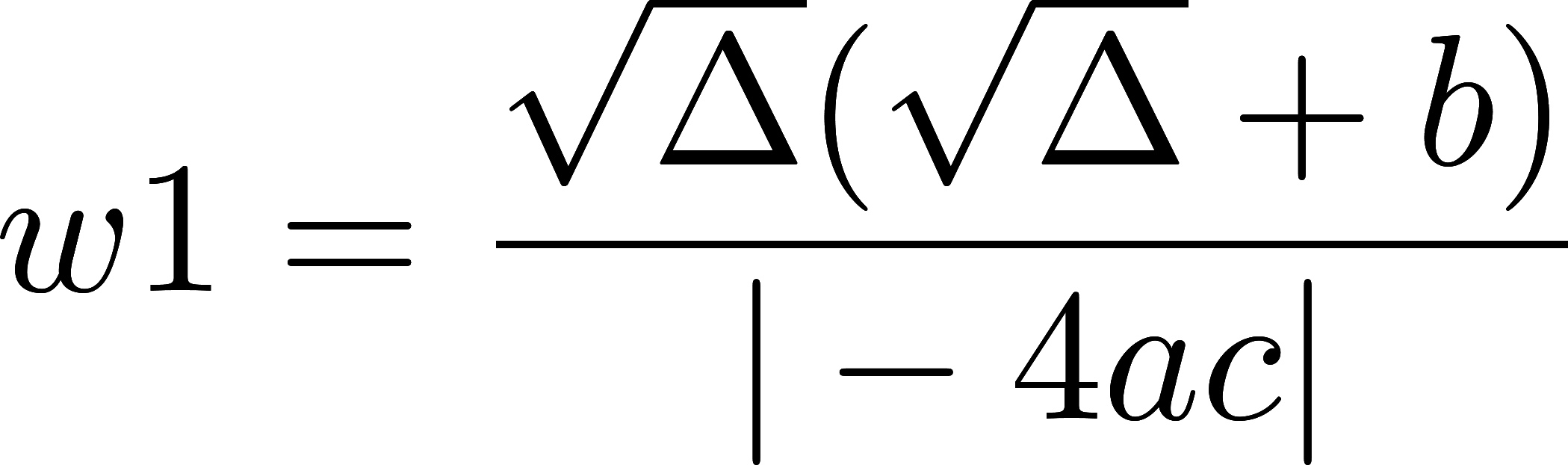
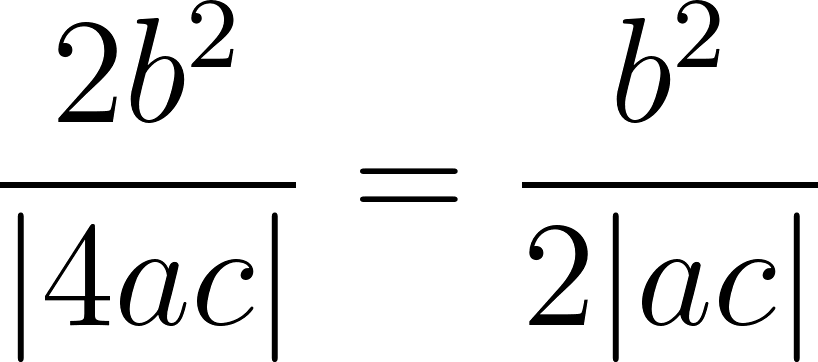
ora, una di queste due soluzioni richiede una sottrazione ovvero z+ per b>0 e z- per b <0 :prendiamo per semplicità b>0, essendo l’analisi dell’altro caso del tutto analoga osserviamo che in aritmetica floating-point [](https://www.codecogs.com/eqnedit.php?latex=%5Csqrt%7B%5CDelta%7D%20#0) sarà quasi sempre arrotondamento, d’altra parte la funzione [](https://www.codecogs.com/eqnedit.php?latex=%5Csqrt%7B%7D#0) viene calcolata alla precisione di macchina in tutti i linguaggi di calcolo (vedremo nelle prossime lezioni per calcolare [](https://www.codecogs.com/eqnedit.php?latex=%5Csqrt%7B%5Calpha%7D#0), [](http://www.texrendr.com/?eqn=%5Calpha#0) > 0 come soluzione dell’equazione [](http://www.texrendr.com/?eqn=x%5E2%20-%20%5Calpha%20%3D%200#0), il metodo delle tangenti detto anche metodo di Newton) quindi nella sottrazione [](https://www.codecogs.com/eqnedit.php?latex=%5Csqrt%7B%5CDelta%7D%20-b#0) il primo dato è sicuramente affetto da un errore al massimo dell’ordine della precisione di macchina. Quando ci si aspetta che questa sottrazione possa far perdere precisione?

quando [](https://www.codecogs.com/eqnedit.php?latex=%5Csqrt%7B%5CDelta%7D%20#0) è vicina a b in termini relativi, cioè per [](https://www.codecogs.com/eqnedit.php?latex=b%5E2%20%3E%3E%20%7C4ac%7C#0) => [](https://www.codecogs.com/eqnedit.php?latex=%5Csqrt%7B%5CDelta%7D%20#0)≈ b

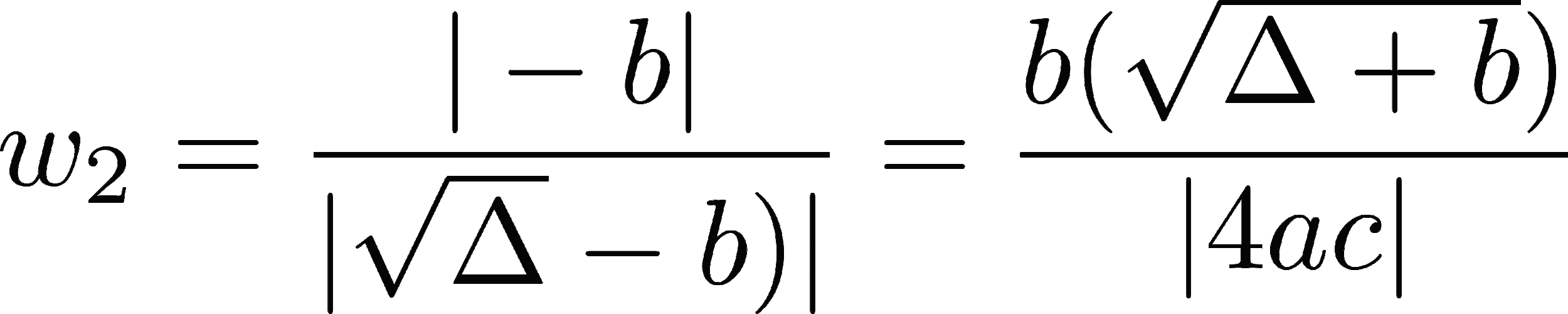
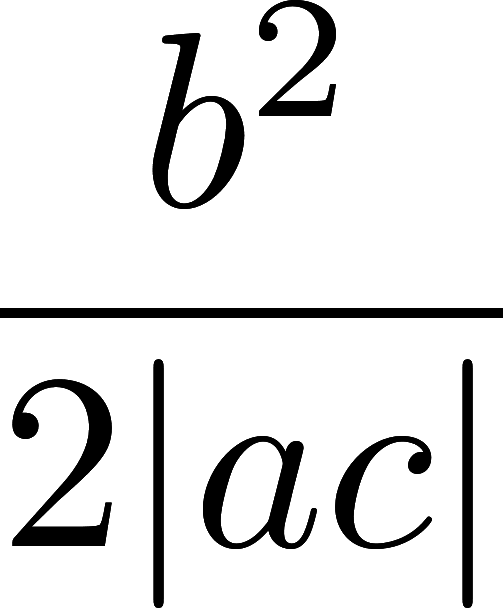
infatti posto x = [](https://www.codecogs.com/eqnedit.php?latex=%5Csqrt%7B%5CDelta%7D%20#0), y = -b i pesi sono

[](http://www.texrendr.com/?eqn=w_1%3D%5Cfrac%7B%7Cx%7C%7D%7B%7Cx%2By%7C%7D%3D%5Cfrac%7B%5Csqrt%7B%5CDelta%7D%7D%7B%7C%5Csqrt%7B%5CDelta%7D-b%7C%7D#0) [](http://www.texrendr.com/?eqn=%3D%5Cfrac%7B%5Csqrt%7B%5CDelta%7D(%5Csqrt%7B%5CDelta%7D%2Bb)%7D%7B%7C%5Csqrt%7B%5CDelta%7D-b%7C(%5Csqrt%7B%5CDelta%7D%2Bb)%7D#0) [](http://www.texrendr.com/?eqn=%3D%5Cfrac%7B%5Csqrt%7B%5CDelta%7D(%5Csqrt%7B%5CDelta%7D%2Bb)%7D%7B%7C%5CDelta-b%5E2%7C%7D%3D%5Cfrac%7B%5Csqrt%7B%5CDelta%7D(%5Csqrt%7B%5CDelta%7D%2Bb)%7D%7B%7C-4ac%7C%7D#0) (a den si sostituisce il delta con la sua definizione e si toglie il b2)

Ma [](https://www.codecogs.com/eqnedit.php?latex=%5Csqrt%7B%5CDelta%7D%20#0)≈ b quindi

[](http://www.texrendr.com/?eqn=w1%3D%5Cfrac%7B%5Csqrt%7B%5CDelta%7D(%5Csqrt%7B%5CDelta%7D%2Bb)%7D%7B%7C-4ac%7C%7D#0) ≈ [](https://www.codecogs.com/eqnedit.php?latex=%5Cfrac%7B2b%5E2%7D%7B%7C4ac%7C%7D%3D%5Cfrac%7Bb%5E2%7D%7B2%7Cac%7C%7D#0)

e analogamente

[](http://www.texrendr.com/?eqn=w_2%3D%5Cfrac%7B%7C-b%7C%7D%7B%7C%5Csqrt%7B%5CDelta%7D-b)%7C%7D%3D%5Cfrac%7Bb(%5Csqrt%7B%5CDelta%2Bb%7D)%7D%7B%7C4ac%7C%7D#0) ≈ [](http://www.texrendr.com/?eqn=%5Cfrac%7Bb%5E2%7D%7B2%7Cac%7C%7D#0)

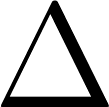
Siccome il rapporto ottenuto da w1 e w2 può diventare arbitrariamente grande (dipende dagli ordini di grandezza dei coefficienti a, b, c) la soluzione z+ può subire una fortissima perdita di precisione in aritmetica floating-point, fino a renderla priva di significato in pratica facciamo a questo proposito un paio di esempi

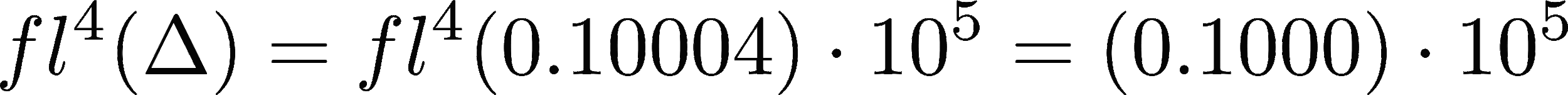
**ESEMPIO 4.1**

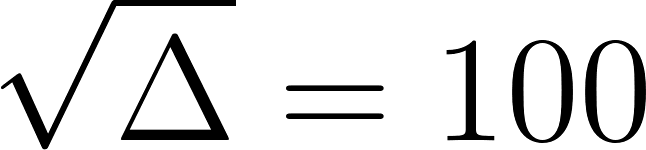
consideriamo l’equazione [](https://www.codecogs.com/eqnedit.php?latex=z%5E2%2B100z-1%3D0#0)

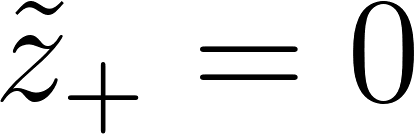
cioè a = 1, b= 100, c =-1 con delta = 1002+4= 10004

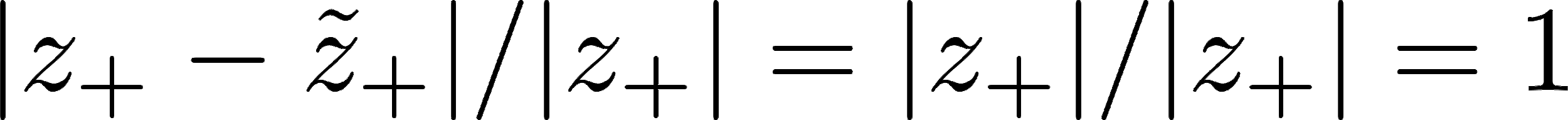
nel sistema floating-point virtuale F(10, 4, L, U) con L ed U opportuni (qui il parametro chiave è #cifre\_mantissa = 4)

con t=4 cifre di mantissa [](https://www.codecogs.com/eqnedit.php?latex=%5CDelta#0) viene arrotondato a 10000

[](http://www.texrendr.com/?eqn=fl%5E4(%5CDelta)%20%3D%20fl%5E4(0.10004)%5Ccdot%2010%5E5%20%3D%20(0.1000)%5Ccdot%2010%5E5#0)

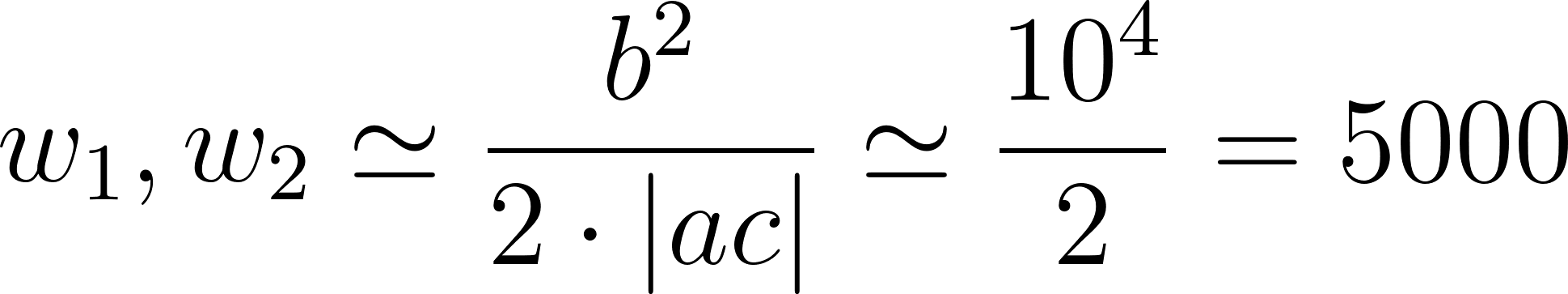
quindi viene calcolato [](https://www.codecogs.com/eqnedit.php?latex=%5Csqrt%7B%5CDelta%7D%20%3D%20100#0) (come sempre, usiamo il meno possibile la notazione floating-point per semplicità)

quindi in questo sistema viene calcolata [](https://www.codecogs.com/eqnedit.php?latex=%5Ctilde%7Bz%7D_%2B%3D0#0) con un errore relativo del 100%

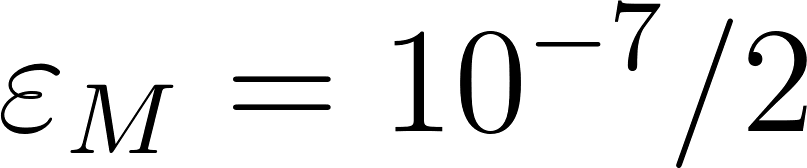
[](http://www.texrendr.com/?eqn=%7Cz_%2B%20-%20%5Ctilde%7Bz%7D_%2B%7C%20%2F%20%7Cz_%2B%7C%20%3D%20%7Cz_%2B%7C%2F%7Cz_%2B%7C%20%3D%201#0)

In un certo senso questo è un caso limite, per la scarsità di cifre di mantissa si è costretti ad approssimare con zero una quantità che non è nulla, facendo direttamente un errore relativo del 100%.

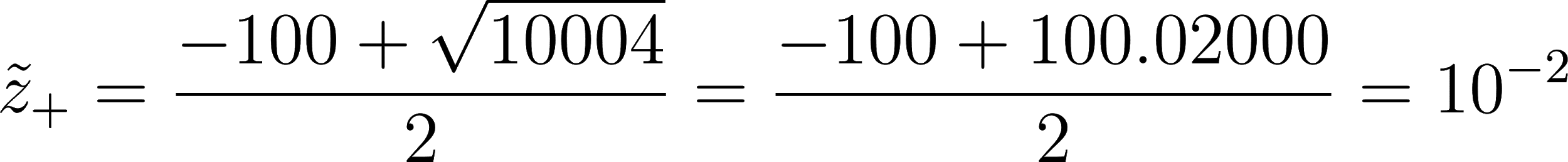
Per vedere l’effetto dei coefficienti di amplificazione

[](https://www.codecogs.com/eqnedit.php?latex=w_1%2C%20w_2%20%5Csimeq%20%5Cfrac%7Bb%5E2%7D%7B2%5Ccdot%20%7Cac%7C%7D%20%5Csimeq%20%5Cfrac%7B10%5E4%7D%7B2%7D%20%3D%205000#0)

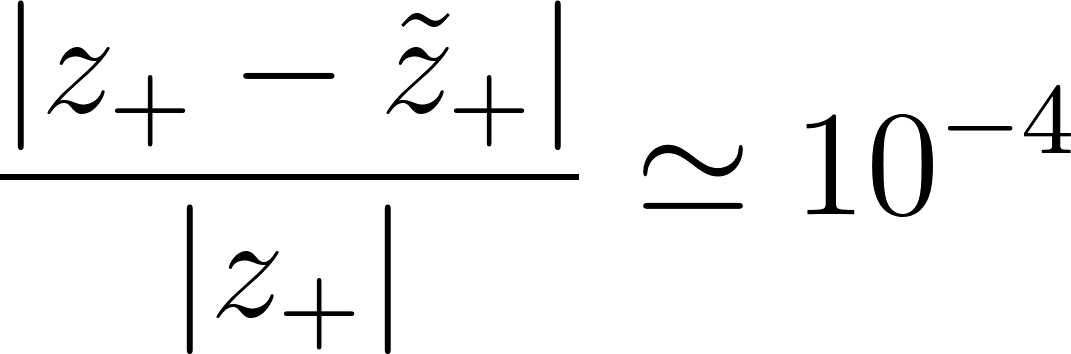
consideriamo F (10,8,L,U)

cioè passiamo a t=8 cifre di mantissa con [](https://www.codecogs.com/eqnedit.php?latex=%5Cvarepsilon_M%20%3D%2010%5E%7B-7%7D%2F2#0)

In questo caso si può verificare che

[](https://www.codecogs.com/eqnedit.php?latex=%5Ctilde%7Bz%7D_%2B%20%3D%20%5Cfrac%7B-100%2B%5Csqrt%7B10004%7D%7D%7B2%7D%20%3D%20%5Cfrac%7B-100%2B100.02000%7D%7B2%7D%20%3D%2010%5E%7B-2%7D#0)

mentre il valore “esatto” è z+= 0.0099990002; per cui abbiamo un errore relativo

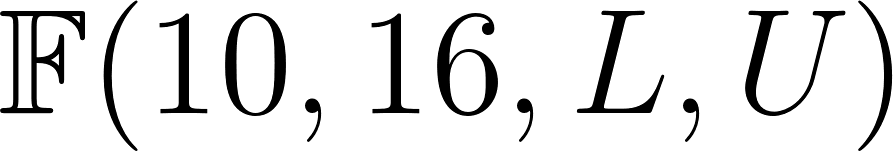
[](https://www.codecogs.com/eqnedit.php?latex=%5Cfrac%7B%7Cz_%2B%20-%20%5Ctilde%7Bz%7D_%2B%7C%7D%7B%7Cz_%2B%7C%7D%20%5Csimeq%2010%5E%7B-4%7D#0)

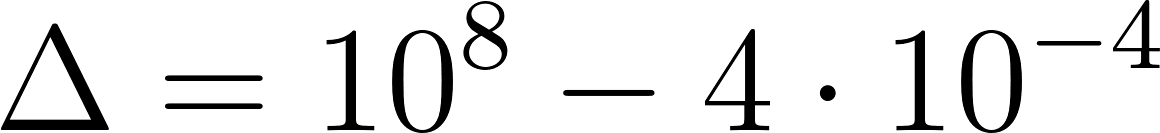
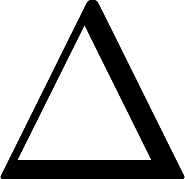
abbiamo quindi perso tre ordini di precisione rispetto alla precisione di macchina, il che è ben spiegato dai pesi w1, w2 che sono dell’ordine di 103. Osserviamo che l’errore commesso è ≈10-4, cioè circa dello 0.01%. D’altra parte tale errore, che potrebbe essere comunque accettabile in molti contesti applicativi, è 2000 volte più grande della precisione di macchina (poteva esserlo fino a 5000 volte, ma va tenuto presente che la precisione di macchina è una soglia massima e che gli errori relativi sono di solito più piccoli di essa). Se avessimo lavorato con t=16 cifre di mantissa, ci saremmo aspettati un’amplificazione degli errori di arrotondamento fino a

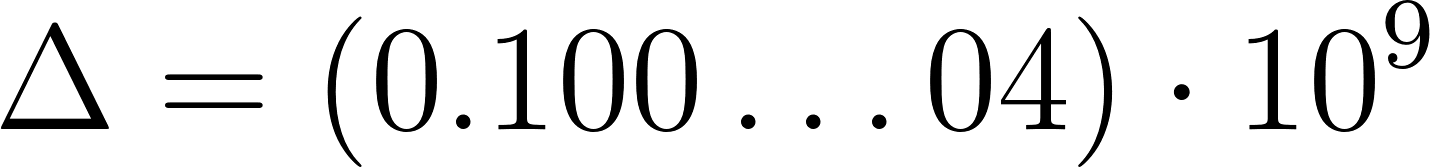


e quindi avremmo ottenuto una precisione  ma comunque molto elevata ma non è difficile convincersi che ci vuole poco per mettere in crisi qualsiasi sistema floating-point, basta che cambino i rapporti tra gli ordini di grandezza dei coefficienti.

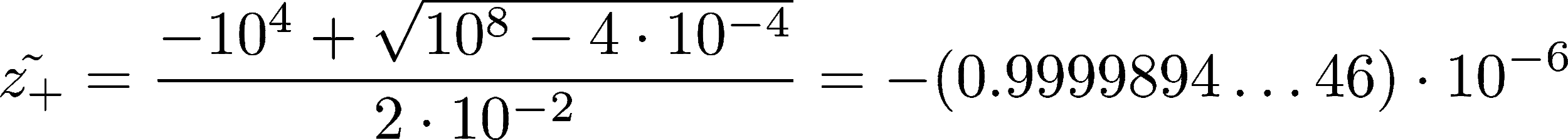
**ESEMPIO 4.2**

Consideriamo infatti [](http://www.texrendr.com/?eqn=10%5E%7B-2%7Dz%5E2%20%2B%2010%5E4z%20%2B%2010%5E%7B-2%7D%20%3D%200#0) in [](https://www.codecogs.com/eqnedit.php?latex=%5Cmathbb%7BF%7D(10%2C16%2CL%2CU)#0)

(sostanzialmente la precisione dell’interfaccia Matlab). Si osservi che qui con t=8 cifre di mantissa saremmo stati di nuovo nella situazione di approssimare [](https://www.codecogs.com/eqnedit.php?latex=%5CDelta%20%3D%2010%5E8-4%5Ccdot%2010%5E%7B-4%7D#0) con 108, perché [](http://www.texrendr.com/?eqn=%5CDelta#0) ha 12 cifre significative

[](https://www.codecogs.com/eqnedit.php?latex=%5CDelta%20%3D%20(0.100%20%5Cdotso%2004)%5Ccdot%2010%5E9#0)

facendo quindi un errore relativo del 100% .Con t=16 cifre di mantissa si calcola

[](http://www.texrendr.com/?eqn=%5Ctilde%7Bz_%2B%7D%20%3D%20%5Cfrac%7B-10%5E4%20%2B%20%5Csqrt%7B10%5E8-4%5Ccdot%2010%5E%7B-4%7D%7D%7D%7B2%5Ccdot%2010%5E%7B-2%7D%7D%20%3D%20-(0.9999894%20%5Cdotso%2046)%20%5Ccdot%2010%5E%7B-6%7D#0)

mentre la soluzione “esatta” (arrotondata a 16 cifre) è



con errore relativo



e una perdita di precisione di 10 ordini di grandezza rispetto a ,

che è spiegabile con i fattori di amplificazione

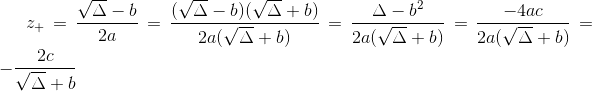


questi esempi mostrano che la formula risolutiva classica delle equazioni di 2°grado è molto instabile quando

 ovvero 

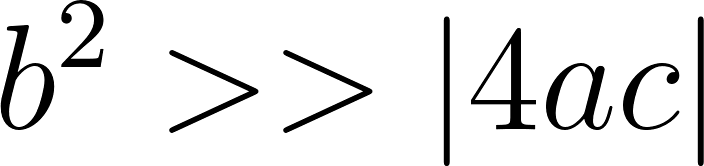
a causa di una sottrazione intrinseca nel modo in cui è scritta.

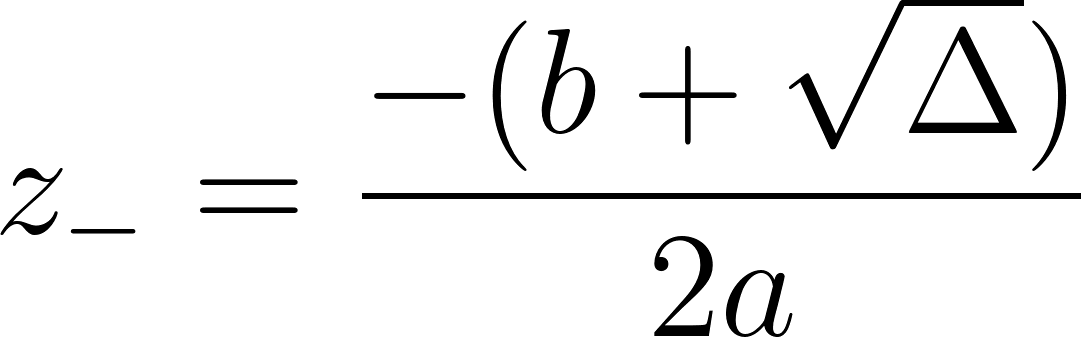
In questo caso però il problema si può risolvere riscrivendo la formula con un “trucchetto” algebrico. Considerando nuovamente il caso b>0



in  le 2 formule sono equivalenti; in  invece, la formula



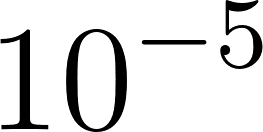
diventa stabile perché è stata eliminata la sottrazione (che come sappiamo è instabile per [](https://www.codecogs.com/eqnedit.php?latex=b%5E2%20%3E%3E%7C4ac%7C#0)). Dall’altra parte,

[](https://www.codecogs.com/eqnedit.php?latex=z_-%20%3D%20%5Cfrac%7B-(b%2B%5Csqrt%7B%5CDelta%7D)%7D%7B2a%7D#0)

è stabile perché non contiene sottrazioni.

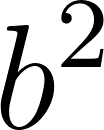
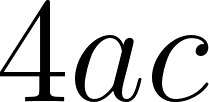
Possiamo a questo punto scrivere una formula risolvibile “STABILIZZATA” che tiene conto del segno di b, ovvero

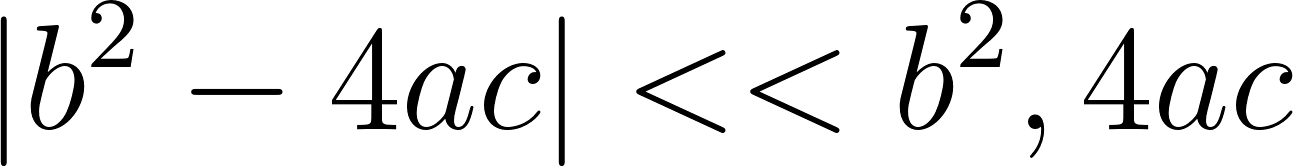


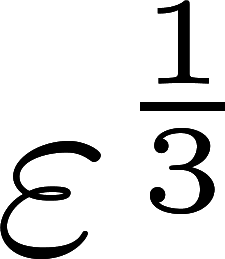
Tornando all’esempio 4.2 , si ha che  fa un errore relativo ≈[](https://www.codecogs.com/eqnedit.php?latex=10%5E%7B-5%7D#0)su  come abbiamo visto, mentre detto  il valore di  calcolato in , si ha



cioè il calcolo di  è stato completamente stabilizzato.

qualcuno potrebbe osservare che anche il calcolo di può contenere una sottrazione, precisamente quando sgn(a)=sgn(c) e che questa può diventare instabile quando [](https://www.codecogs.com/eqnedit.php?latex=b%5E2#0)≈[](https://www.codecogs.com/eqnedit.php?latex=4ac#0) o meglio quando

[](http://www.texrendr.com/?eqn=%7Cb%5E%7B2%7D-4ac%7C%3C%3Cb%5E%7B2%7D#0)

[questa situazione è molto più difficile da trattare](http://www.texrendr.com/?eqn=%7Cb%5E%7B2%7D-4ac%7C%3C%3Cb%5E%7B2%7D#0) e non ce ne occuperemo. Diciamo solo che qui la perdita di precisione non è completamente eliminabile, ma esiste un algoritmo grazie al quale l’errore relativo sulle due soluzioni per Δ>0 con b2≈4ac è dell’ordine di [M (dove εM è la precisione di macchina)](http://www.texrendr.com/?eqn=%5Cvarepsilon%5E%7B%5Cfrac%7B1%7D%7B3%7D%7D#0) cioè con perdita di precisione consistente ma controllata.