

Software crítico para sistemas espaciales

2026/27

Master Universitario en Enxeñaría Aeroespacial

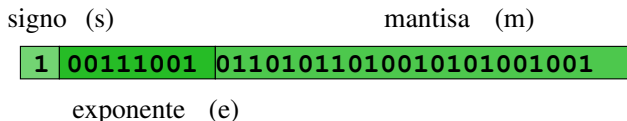
Arno Formella
Miguel Ramón González Castro
Manuel Pérez Cota

Departamento de Informática
Universidade de Vigo

26/27

formato de números flotantes

casi todos los sistemas usan hoy el estándar IEEE 754 para representar números con coma flotante



$$v = (-1)^s \cdot 2^{e-2^{|e|-1}} \cdot (1.m)$$

para precisión simple (32 bits): $|e| = 8$, es decir, 8 bits
 $|m| = 23$, es decir, 23 bits, el signo siempre es un solo bit
(usamos $| \cdot |$ para anotar *longitud de cadena en bits*)

para precisión doble (64 bits, $|e| = 11$, $|m| = 52$):

- si $0 < e < 2047$, entonces $v = (-1)^s \cdot 2^{e-1023} \cdot (1.m)$
- si $e = 0$ y $m \neq 0$, entonces $v = (-1)^s \cdot 2^{-1022} \cdot (0.m)$
- si $e = 0$ y $m = 0$, entonces $v = (-1)^s \cdot 0$
- si $e = 2047$ y $m = 0$, entonces $v = (-1)^s \cdot \text{Inf}$
- si $e = 2047$ y $m \neq 0$, entonces $v = \text{NaN}$

inf significa *infinity* (infinito)

NaN significa *not-a-number* (no es un número)

diferencia con float

```
1  #include <iostream>
2  #include <iomanip>
3
4  using namespace std;
5
6  int main() {
7      const float a11=64919121.0, a12=-159018721.0;
8      const float a21=41869520.5, a22=-102558961.0;
9      const float d = a11*a22 - a12*a21;
10     const float x1= a22 / d;
11     const float x2=-a21 / d;
12     cout
13         <<setw(20)<<setprecision(15)<<x1<<"\n"
14         <<setw(20)<<setprecision(15)<<x2<<"\n"
15         <<setw(20)<<setprecision(15)<< d<<"\n"
16     ;
17     return 0;
18 }
```

diferencia con double

```
1  #include <iostream>
2  #include <iomanip>
3
4  using namespace std;
5
6  int main() {
7      const double a11=64919121.0, a12=-159018721.0;
8      const double a21=41869520.5, a22=-102558961.0;
9      const double d = a11*a22 - a12*a21;
10     const double x1= a22 / d;
11     const double x2=-a21 / d;
12     cout
13         <<setw(20)<<setprecision(15)<<x1<<"\n"
14         <<setw(20)<<setprecision(15)<<x2<<"\n"
15         <<setw(20)<<setprecision(15)<< d<<"\n"
16     ;
17     return 0;
18 }
```

diferencia con long double

```
1  #include <iostream>
2  #include <iomanip>
3
4  using namespace std;
5
6  int main() {
7      const long double a11=64919121.0, a12=-159018721.0;
8      const long double a21=41869520.5, a22=-102558961.0;
9      const long double d = a11*a22 - a12*a21;
10     const long double x1= a22 / d;
11     const long double x2=-a21 / d;
12     cout
13         <<setw(20)<<setprecision(15)<<x1<<"\n"
14         <<setw(20)<<setprecision(15)<<x2<<"\n"
15         <<setw(20)<<setprecision(15)<< d<<"\n"
16     ;
17     return 0;
18 }
```

diferencia con python

```
1  a11=64919121.0
2  a12=-159018721.0
3  a21=41869520.5
4  a22=-102558961.0
5
6  d  =a11*a22 - a12*a21
7  x1 = a22 / d
8  x2 =-a21 / d
9
10 print("{:20.15}".format(x1))
11 print("{:20.15}".format(x2))
12 print("{:20.15}".format(d))
```

diferencia con float

```
1  #include <iostream>
2  #include <iomanip>
3
4  using namespace std;
5
6  int main() {
7      const float a11=64919121.0, a12=-159018721.0;
8      const float a21=41869520.5, a22=-102558961.0;
9      const float d = a11*a22 - a12*a21;
10     const float x1= a22 / d;
11     const float x2=-a21 / d;
12     cout
13         <<setw(20)<<setprecision(15)<<x1<<"\n"
14         <<setw(20)<<setprecision(15)<<x2<<"\n"
15         <<setw(20)<<setprecision(15)<<d<<"\n"
16     ;
17     return 0;
18 }
```

File Edit View Search Terminal Help

```
(base) formella@hmformella:~/tex/teach/scribe/cpp$ g++ -Wall mat_float.cpp
```

```
(base) formella@hmformella:~/tex/teach/scribe/cpp$ ./a.out
```

```
-inf
```

```
-inf
```

```
0
```

```
(base) formella@hmformella:~/tex/teach/scribe/cpp$ █
```

diferencia con double

```
1  #include <iostream>
2  #include <iomanip>
3
4  using namespace std;
5
6  int main() {
7      const double a11=64919121.0, a12=-159018721.0;
8      const double a21=41869520.5, a22=-102558961.0;
9      const double d = a11*a22 - a12*a21;
10     const double x1= a22 / d;
11     const double x2=-a21 / d;
12     cout
13         <<setw(20)<<setprecision(15)<<x1<<"\n"
14         <<setw(20)<<setprecision(15)<<x2<<"\n"
15         <<setw(20)<<setprecision(15)<<d<<"\n"
16     ;
17     return 0;
18 }
```

File Edit View Search Terminal Help

```
(base) formella@hmformella:~/tex/teach/scrise/cpp$ g++ -Wall mat_double.cpp
```

```
(base) formella@hmformella:~/tex/teach/scrise/cpp$ ./a.out
```

```
102558961
```

```
41869520.5
```

```
-1
```

```
(base) formella@hmformella:~/tex/teach/scrise/cpp$ █
```

diferencia con long double

```
1  #include <iostream>
2  #include <iomanip>
3
4  using namespace std;
5
6  int main() {
7      const long double a11=64919121.0, a12=-159018721.0;
8      const long double a21=41869520.5, a22=-102558961.0;
9      const long double d = a11*a22 - a12*a21;
10     const long double x1= a22 / d;
11     const long double x2=-a21 / d;
12     cout
13         <<setw(20)<<setprecision(15)<<x1<<"\n"
14         <<setw(20)<<setprecision(15)<<x2<<"\n"
15         <<setw(20)<<setprecision(15)<<d<<"\n"
16     ;
17     return 0;
18 }
```

File Edit View Search Terminal Help

(base) formella@hmformella:~/tex/teach/scrise/cpp\$ g++ -Wall mat_longdouble.cpp

(base) formella@hmformella:~/tex/teach/scrise/cpp\$./a.out

205117922

83739041

-0.5

(base) formella@hmformella:~/tex/teach/scrise/cpp\$

diferencia con long double

```
1  a11=64919121.0
2  a12=-159018721.0
3  a21=41869520.5
4  a22=-102558961.0
5
6  d  =a11*a22 - a12*a21
7  x1 = a22 / d
8  x2 =-a21 / d
9
10 print("{:20.15}".format(x1))
11 print("{:20.15}".format(x2))
12 print("{:20.15}".format(d))
```

File Edit View Search Terminal Help

```
(base) formella@hmformella:~/tex/teach/scribe/cpp$ cd ../python/
(base) formella@hmformella:~/tex/teach/scribe/python$ python mat.py
      102558961.0
      41869520.5
      -1.0
(base) formella@hmformella:~/tex/teach/scribe/python$
```

resumen:

- en mi sistema los tres tipos se comportan diferentes
- parece que `float` es un flotante on 32 bits, `double` son 64 bit, y `long double` es algo más grande
- el estándar de C solo dice: $|float| \leq |double| \leq |longdouble|$
- en muchos, pero no en todos los sistemas, `long double` es de 80 bits (en MSVC es solamente 64 bits)
- hay que saber como está realizado en el sistema en cuestión
- hay que tener en cuenta posibles incompatibilidades entre sistemas
- python en mi versión 3.12 usa 64 bits en flotantes

- el compilador podría avisar ya que hemos puesto `const` (g++ 11.4 no lo hace)
- con `float` tenemos una división por 0, podríamos captar una excepción, pero por defecto no está activada (y tampoco resuelve el problema)
- existen diferentes modos de redondeo (hacia cero, hacia más próximo, hacia infinito, hacia menos infinito) que complica la cosa aún más...
- **solución:** usar rutinas matemáticamente robustas para el tipo de datos empleados y aceptar que la precisión siempre es limitada

miramos la siguiente secuencia

$$a_0 = \frac{11}{2}, \quad a_1 = \frac{61}{11}, \quad a_{n+1} = 111 - \frac{1130 - 3000/a_{n-1}}{a_n},$$

¿a dónde converge?

secuencia con float

```
1  #include<iomanip>
2  #include<iostream>
3
4  using namespace std;
5
6  int main() {
7      float a=11.0/2.0;
8      float b=61.0/11.0;
9      for(int n=1; n<=25; ++n) {
10         cout<<setprecision(16)<<a<<"\n";
11         const float c=111-(1130-3000/a)/b;
12         a=b;
13         b=c;
14     }
15     return 0;
16 }
```

secuencia con double

```
1  #include<iomanip>
2  #include<iostream>
3
4  using namespace std;
5
6  int main() {
7      double a=11.0/2.0;
8      double b=61.0/11.0;
9      for(int n=1; n<=25; ++n) {
10         cout<<setprecision(16)<<a<<"\n";
11         const double c=111-(1130-3000/a)/b;
12         a=b;
13         b=c;
14     }
15     return 0;
16 }
```

secuencia con long double

```
1  #include<iomanip>
2  #include<iostream>
3
4  using namespace std;
5
6  int main() {
7      long double a=11.0/2.0;
8      long double b=61.0/11.0;
9      for(int n=1; n<=25; ++n) {
10         cout<<setprecision(16)<<a<<"\n";
11         const long double c=111-(1130-3000/a)/b;
12         a=b;
13         b=c;
14     }
15     return 0;
16 }
```

secuencias con diferentes precisiones

```
File Edit View Search Terminal Help
formella@hpformella ~/tex/teac
h/scrise/cpp $ g++ -O3 -Wall s
eq_float.cpp
formella@hpformella ~/tex/teac
h/scrise/cpp $ ./a.out
```

```
5.5
5.545454502
5.590163934
5.633285522
5.672050476
5.667526245
4.941246033
-10.56215668
160.5036926
102.1900253
100.1250763
100.0073166
100.0004272
100.0000229
100
100
100
100
100
100
100
100
100
100
100
100
```

```
formella@hpformella ~/tex/teac
h/scrise/cpp $ █
```

```
File Edit View Search Terminal Help
formella@hpformella ~/tex/teac
h/scrise/cpp $ g++ -O3 -Wall s
eq_double.cpp
formella@hpformella ~/tex/teac
h/scrise/cpp $ ./a.out
```

```
5.5
5.545454545
5.590163934
5.633431085
5.674648621
5.713329052
5.749120921
5.781810945
5.811314669
5.837663962
5.861078485
5.883542934
5.935956717
6.534421641
15.41304318
67.47239836
97.13715118
99.82469415
99.98953969
99.99937614
99.99996276
99.99999778
99.99999987
99.99999999
100
```

```
formella@hpformella ~/tex/teac
h/scrise/cpp $ █
```

```
File Edit View Search Terminal Help
formella@hpformella ~/tex/teac
h/scrise/cpp $ g++ -O3 -Wall s
eq_longdouble.cpp
formella@hpformella ~/tex/teac
h/scrise/cpp $ ./a.out
```

```
5.5
5.545454545
5.590163934
5.633431085
5.674648621
5.713329052
5.74912092
5.781810934
5.811314464
5.837660439
5.86101815
5.882513763
5.918471029
6.240858698
11.1559948
52.79838451
94.69104863
99.66650971
99.98006941
99.99881122
99.99992904
99.99999576
99.99999975
99.99999998
100
```

```
formella@hpformella ~/tex/teac
h/scrise/cpp $ █
```



¿a dónde converge?

$$a_0 = \frac{11}{2}, \quad a_1 = \frac{61}{11}, \quad a_{n+1} = 111 - \frac{1130 - 3000/a_{n-1}}{a_n},$$

adivinamos y comprobamos por inducción:

$$a_n = \frac{6^{n+1} + 5^{n+1}}{6^n + 5^n}$$

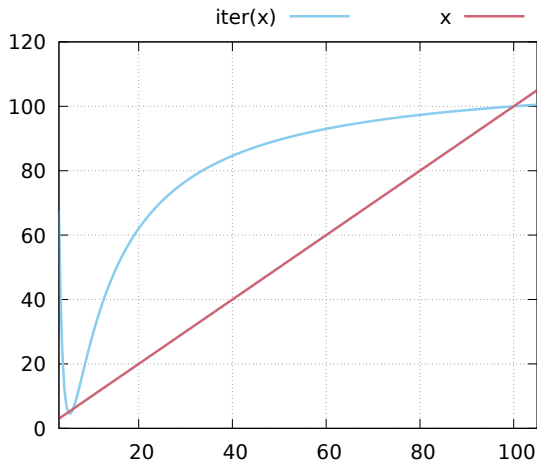
a_0 y a_1 son triviales

$$\begin{aligned} a_{n+1} &= 111 - \frac{1130 - 3000 / ((6^n + 5^n) / (6^{n-1} + 5^{n-1}))}{(6^{n+1} + 5^{n+1}) / (6^n + 5^n)} \\ &= \frac{111 \cdot (6^{n+1} + 5^{n+1}) - 1130 \cdot (6^n + 5^n) + 3000 \cdot (6^{n-1} + 5^{n-1})}{6^{n+1} + 5^{n+1}} \\ &= \frac{666 \cdot 6^n + 555 \cdot 5^n - 1130 \cdot 6^n - 1130 \cdot 5^n + 500 \cdot 6^n + 600 \cdot 5^n}{6^{n+1} + 5^{n+1}} \\ &= \frac{36 \cdot 6^n + 25 \cdot 5^n}{6^{n+1} + 5^{n+1}} = \frac{6 + 5 \cdot 5^{n+1} / 6^{n+1}}{1 + 5^{n+1} / 6^{n+1}} \implies \lim_{n \rightarrow \infty} = 6 \end{aligned}$$



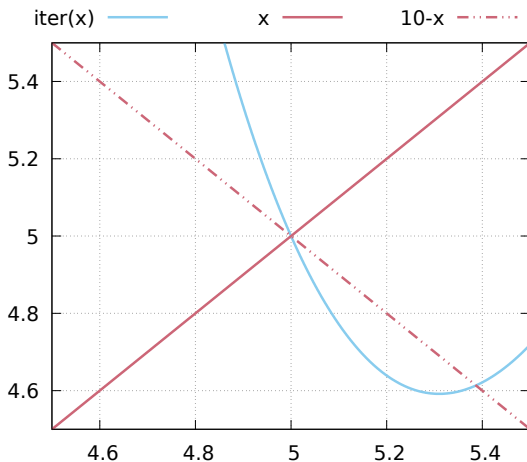
miramos la siguiente función

$$f(x) = 111 - (1130 - 3000/x)/x$$



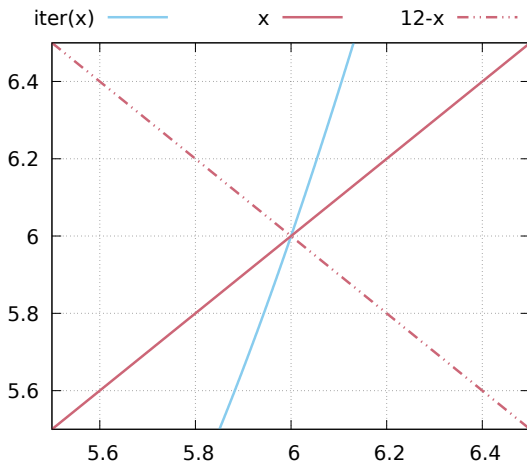
sus puntos fijos

$f(x)$ tiene 3 puntos fijos: el primero (5,5) inestable (diverge)



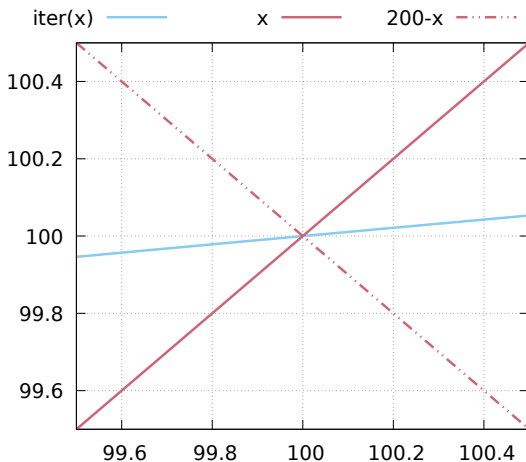
sus puntos fijos

$f(x)$ tiene 3 puntos fijos: el segundo $(6,6)$ inestable (diverge)



sus puntos fijos

$f(x)$ tiene 3 puntos fijos: el tercero $(100, 100)$ estable (converge)



usando función directa en vez de iteración

```
1 def item(n):
2     p5=1
3     p6=1
4     for i in range(1,n+1):
5         p5*=5
6         p6*=6
7     return (p5*5+p6*6) / (p5+p6)
8
9 for i in range(0,21):
10    print("{:2} ".format(i),item(i))
```

- computacionalmente más costoso
- valores intermedios más grandes (peligro de desbordamiento)

```
0  5.5
1  5.545454545454546
2  5.590163934426229
3  5.633431085043989
4  5.674648620510151
5  5.713329052380516
6  5.749120919702638
7  5.781810920485616
8  5.8113142382939955
9  5.837656548958712
10 5.860951522516132
11 5.881377215841419
12 5.899153905790065
13 5.914524950678984
14 5.927741407776796
15 5.939050485461118
16 5.948687492480417
17 5.956870731918221
18 5.963798720819403
19 5.969649144047887
20 5.974579028666723
```

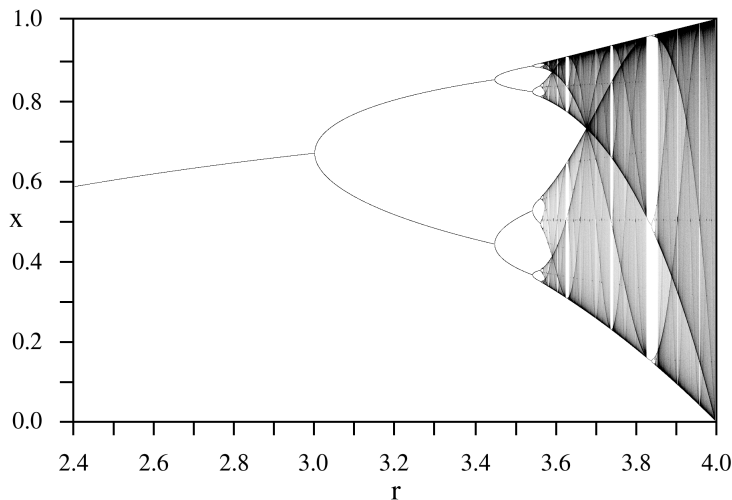
una ecuación logística:

$$\begin{aligned}x_0 &= 0.5 \\ x_{i+1} &= 3.75 \cdot x_i \cdot (1 - x_i)\end{aligned}$$

¿a dónde converge?

dependiendo del factor $r \in [0 : 4]$ (aquí $r = 3.75$) la secuencia converge o muestra comportamiento caótico

diagrama de bifurcación (diagrama de Feigenbaum)



ecuación logística con float

```
1  #include <iostream>
2  #include <iomanip>
3
4  float logis(const float x) {
5      return 3.75*x*(1.0-x);
6  }
7  int main() {
8      float x=0.5;
9      for(int i=0; i<111; ++i) {
10         if(i>50 && i%10==0)
11             std::cout<<"x"<<i<<" = "<<std::setprecision(10)<<x<<"\n";
12         x=logis(x);
13     }
14     return 0;
15 }
```

valores verdaderos:

x60 = 0.7990863343
x70 = 0.4521952998
x80 = 0.8561779966
x90 = 0.7399137486
x100 = 0.8882939922
x110 = 0.7156795292

valores calculados (con float):

x60 = 0.6019204855
x70 = 0.9286298156
x80 = 0.5707449913
x90 = 0.5013405681
x100 = 0.6443744898
x110 = 0.8775931001

ecuación logística con double

```
1  #include <iostream>
2  #include <iomanip>
3
4  double logis(const double x) {
5      return 3.75*x*(1.0-x);
6  }
7  int main() {
8      double x=0.5;
9      for(int i=0; i<111; ++i) {
10         if(i>50 && i%10==0)
11             std::cout<<"x"<<i<<" = "<<std::setprecision(10)<<x<<"\n";
12         x=logis(x);
13     }
14     return 0;
15 }
```

valores verdaderos:

x60 = 0.7990863343
x70 = 0.4521952998
x80 = 0.8561779966
x90 = 0.7399137486
x100 = 0.8882939922
x110 = 0.7156795292

valores calculados (con double):

x60 = 0.7990957295
x70 = 0.4520542524
x80 = 0.8492167613
x90 = 0.8481094254
x100 = 0.8358933995
x110 = 0.8238971883

ecuación logística con long double

```
1  #include <iostream>
2  #include <iomanip>
3
4  long double logis(const long double x) {
5      return 3.75*x*(1.0-x);
6  }
7  int main() {
8      long double x=0.5;
9      for(int i=0; i<111; ++i) {
10         if(i>50 && i%10==0)
11             std::cout<<"x"<<i<<" = "<<std::setprecision(10)<<x<<"\n";
12         x=logis(x);
13     }
14     return 0;
15 }
```

valores verdaderos:

x60 = 0.7990863343
x70 = 0.4521952998
x80 = 0.8561779966
x90 = 0.7399137486
x100 = 0.8882939922
x110 = 0.7156795292

valores calculados (con long double):

x60 = 0.799086337
x70 = 0.4521952586
x80 = 0.8561759906
x90 = 0.7400517104
x100 = 0.9017659679
x110 = 0.2201217854



errores numéricos y comportamiento caótico

- **solución:** usar otros métodos matemáticos que no sean tan puramente numéricos, por ejemplo, cálculo con intervalos, cálculo con números racionales, cálculo algebraico, cálculo simbólico, cálculo con números reales arbitrariamente precisos etc.
- **existen bibliotecas** correspondientes; los valores exactos de la ecuación logística fueron calculado con la biblioteca `iRRAM`
- generalmente problemas **geométricos** suelen tener sus *problemillas* de precisión, especialmente cerca del valor 0
- si se hace numérica se necesita un/a **matemático/a** y un/a **informático/a**
- hay que **analizar complejidad** en tiempo y memoria, se suele necesitar bastante más recursos si se quiere hacer numérica bien



operaciones con números flotantes

- operaciones con números flotantes no se comportan como operaciones matemáticas entre valores reales
- es decir (como ejemplos entre otros casos más):

$$(a + b) + c \neq a + (b + c)$$

$$a * (b + c) \neq a * b + a * c$$

$$a + a + \dots + a \neq n * a$$

comportamiento por redondeo con float

```
1  #include <iomanip>
2  #include <iostream>
3
4  using namespace std;
5
6  int main() {
7      const float small=0.00003;
8      const float large=1000.0;
9      cout<<setprecision(20)
10         <<"          s = "<<small<<"\n"
11         <<"          l = "<<large<<"\n"
12         <<"          l+s = "<<large+small<<"\n\n"
13         <<"          (l+s)+s = "<<large+small+small<<"\n"
14         <<"          l+(s+s) = "<<large+(small+small)<<"\n\n"
15         <<"          ((l+s)+s)+s = "<<large+small+small+small<<"\n"
16         <<"          l+((s+s)+s) = "<<large+(small+small+small)<<"\n\n"
17         <<"          (((l+s)+s)+s)+s = "<<large+small+small+small+small<<"\n"
18         <<"          l+(((s+s)+s)+s) = "<<large+(small+small+small+small)<<"\n\n"
19         <<"          l*((s+s)+s) = "<<large*(small+small+small)<<"\n"
20         <<"          (l*s+l*s)+l*s = "<<large*small+large*small+large*small<<"\n\n"
21     ;
22     if (large+(small+small)==large+(small+small+small)) cout<<"what?\n";
23     else cout<<"ok\n";
24     float l=large;
25     for(int i=0; i<100000; ++i) l+=small;
26     cout<<setprecision(20)<<" sum of l and s'es = "<<l<<"\n";
27     float s=0.0;
28     for(int i=0; i<100000; ++i) s+=small;
29     cout<<setprecision(20)<<" l plus sum of s'es = "<<large+s<<"\n";
30     cout<<setprecision(20)<<"      l + 100000 * s = "<<large+100000*small<<"\n";
31     return 0;
32 }
```

comportamiento por redondeo con float

```
File Edit View Search Terminal Help
formella@hpformella ~/tex/teach/scrise/cpp $ g++ -Wall asso_float.cpp
formella@hpformella ~/tex/teach/scrise/cpp $ ./a.out
    s = 2.9999999242136254907e-05
    l = 1000
    l+s = 1000

    (l+s)+s = 1000
    l+(s+s) = 1000.00006103515625

    ((l+s)+s)+s = 1000
    l+((s+s)+s) = 1000.00006103515625

    (((l+s)+s)+s)+s = 1000
    l+(((s+s)+s)+s) = 1000.0001220703125

    l*((s+s)+s) = 0.090000003576278686523
    (l*s+l*s)+l*s = 0.0899999961256980896

what?
sum of l and s'es = 1000
l plus sum of s'es = 1003.00250244140625
  l + 100000 * s = 1003
formella@hpformella ~/tex/teach/scrise/cpp $
```

comportamiento por redondeo con double

```
1  #include <iomanip>
2  #include <iostream>
3
4  using namespace std;
5
6  int main() {
7      const double small=0.00003;
8      const double large=1000.0;
9      cout<<setprecision(20)
10         <<"          s = "<<small<<"\n"
11         <<"          l = "<<large<<"\n"
12         <<"          l+s = "<<large+small<<"\n\n"
13         <<"          (l+s)+s = "<<large+small+small<<"\n"
14         <<"          l+(s+s) = "<<large+(small+small)<<"\n\n"
15         <<"          ((l+s)+s)+s = "<<large+small+small+small<<"\n"
16         <<"          l+((s+s)+s) = "<<large+(small+small+small)<<"\n\n"
17         <<"          (((l+s)+s)+s)+s = "<<large+small+small+small+small<<"\n"
18         <<"          l+(((s+s)+s)+s) = "<<large+(small+small+small+small)<<"\n\n"
19         <<"          l*((s+s)+s) = "<<large*(small+small+small)<<"\n"
20         <<"          (l*s+l*s)+l*s = "<<large*small+large*small+large*small<<"\n\n"
21     ;
22     if (large+(small+small)==large+(small+small+small)) cout<<"what?\n";
23     else cout<<"ok\n";
24     double l=large;
25     for(int i=0; i<100000; ++i) l+=small;
26     cout<<setprecision(20)<<" sum of l and s'es = "<<l<<"\n";
27     double s=0.0;
28     for(int i=0; i<100000; ++i) s+=small;
29     cout<<setprecision(20)<<" l plus sum of s'es = "<<large+s<<"\n";
30     cout<<setprecision(20)<<"      l + 100000 * s = "<<large+100000*small<<"\n";
31     return 0;
32 }
```

comportamiento por redondeo con double

```
File Edit View Search Terminal Help
formella@hpformella ~/tex/teach/scrise/cpp $ g++ -Wall asso_double.cpp
formella@hpformella ~/tex/teach/scrise/cpp $ ./a.out
    s = 3.000000000000000076e-05
    l = 1000
    l+s = 1000.0000300000000379

    (l+s)+s = 1000.0000600000000759
    l+(s+s) = 1000.0000599999999622

    ((l+s)+s)+s = 1000.0000900000001138
    l+((s+s)+s) = 1000.0000900000000001

    (((l+s)+s)+s)+s = 1000.0001200000001518
    l+(((s+s)+s)+s) = 1000.0001200000000381

    l*((s+s)+s) = 0.090000000000000010547
    (l*s+l*s)+l*s = 0.090000000000000010547

ok
sum of l and s'es = 1003.0000000037944119
l plus sum of s'es = 1003.0000000000051159
    l + 100000 * s = 1003
formella@hpformella ~/tex/teach/scrise/cpp $
```

comportamiento por redondeo con long double

```
1  #include <iomanip>
2  #include <iostream>
3
4  using namespace std;
5
6  int main() {
7      const long double small=0.00003;
8      const long double large=1000.0;
9      cout<<setprecision(20)
10         <<"          s = "<<small<<"\n"
11         <<"          l = "<<large<<"\n"
12         <<"          l+s = "<<large+small<<"\n\n"
13         <<"          (l+s)+s = "<<large+small+small<<"\n"
14         <<"          l+(s+s) = "<<large+(small+small)<<"\n\n"
15         <<"          ((l+s)+s)+s = "<<large+small+small+small<<"\n"
16         <<"          l+((s+s)+s) = "<<large+(small+small+small)<<"\n\n"
17         <<"          (((l+s)+s)+s)+s = "<<large+small+small+small+small<<"\n"
18         <<"          l+(((s+s)+s)+s) = "<<large+(small+small+small+small)<<"\n\n"
19         <<"          l*((s+s)+s) = "<<large*(small+small+small)<<"\n"
20         <<"          (l*s+l*s)+l*s = "<<large*small+large*small+large*small<<"\n\n"
21     ;
22     if (large+(small+small)==large+(small+small+small)) cout<<"what?\n";
23     else cout<<"ok\n";
24     long double l=large;
25     for(int i=0; i<100000; ++i) l+=small;
26     cout<<setprecision(20)<<" sum of l and s'es = "<<l<<"\n";
27     long double s=0.0;
28     for(int i=0; i<100000; ++i) s+=small;
29     cout<<setprecision(20)<<" l plus sum of s'es = "<<large+s<<"\n";
30     cout<<setprecision(20)<<"      l + 100000 * s = "<<large+100000*small<<"\n";
31     return 0;
32 }
```

comportamiento por redondeo con long double

```
File Edit View Search Terminal Tabs Help
formella@pcformella: ~/tex/teach/scrise × formella@pcformella: ~/tex/teach/scrise/cpp ×
formella@pcformella:~/tex/teach/scrise/cpp$ g++ -O3 asso_longdouble.cpp
formella@pcformella:~/tex/teach/scrise/cpp$ ./a.out
s = 3.000000000000000076e-05
l = 1000
l+s = 1000.00003

(l+s)+s = 1000.0000599999999999
l+(s+s) = 1000.00006

((l+s)+s)+s = 1000.0000899999999999
l+((s+s)+s) = 1000.00009

(((l+s)+s)+s)+s = 1000.0001199999999999
l+(((s+s)+s)+s) = 1000.00012

l*((s+s)+s) = 0.09000000000000000228
(l*s+l*s)+l*s = 0.09000000000000000228

ok
sum of l and s'es = 1002.999999999974492
l plus sum of s'es = 1003.0000000000000002
l + 100000 * s = 1003.0000000000000001
formella@pcformella:~/tex/teach/scrise/cpp$
```



método de Kahan para sumas de valores

- operaciones tan aparentemente simples como calcular la media o las raíces de polinomios esconden problemas numéricos
- por ejemplo, el método de Kahan mejora la sumación especialmente si los sumandos son de tamaños de ordenes diferente (y/o la precisión del cálculo es reducido)
- parece que se hace cálculo *inútil*, pero no se puede optimizar pensando en simplificar las operaciones
- existe otro método (si todos los valores están conocidos antemano) que suma en árbol (y por eso es parcialmente paralelizable)

método de Kahan para sumas de valores

```
1  #include <algorithm>
2  #include <iomanip>
3  #include <iostream>
4  #include <random>
5  #include <vector>
6
7  #define N 10000000
8
9  int main() {
10     std::random_device rd {};
11     std::mt19937 gen {rd()};
12     std::lognormal_distribution<> normDisBase {0.0,0.01};
13     std::vector<float> data(N);
14     for(int i=0; i<N; ++i) data[i]=(float)normDisBase(gen);
15
16     float S=0.0;
17     for(int j=0; j<N; ++j) S+=data[j];    // suma simple
18     std::cout<<"S      = "<<std::setprecision(16)<<S/N<<"\n";
19
20     float Sk=0.0;
21     float C=0.0;
22     for(int j=0; j<N; ++j) {                // suma Kahan's method
23         const float Y=data[j]-C;
24         const float T=Sk+Y;
25         C=(T-Sk)-Y;
26         Sk=T;
27     }
28     std::cout<<"Sk      = "<<std::setprecision(16)<<Sk/N<<"\n";
29     return 0;
30 }
```

método de Kahan para sumas de valores

```
File Edit View Search Terminal Tabs Help
formella@pcformella: ~/tex/teach/scrise × formella@pcformella: ~/tex/teach/scrise/cpp ×
formella@pcformella:~/tex/teach/scrise/cpp$ g++ -O3 kahan.cpp
formella@pcformella:~/tex/teach/scrise/cpp$ ./a.out
S      = 1.000004887580872
Sk     = 1.000051736831665
formella@pcformella:~/tex/teach/scrise/cpp$ ./a.out
S      = 1.000003933906555
Sk     = 1.000051140785217
formella@pcformella:~/tex/teach/scrise/cpp$ ./a.out
S      = 1.00000524520874
Sk     = 1.00005316734314
formella@pcformella:~/tex/teach/scrise/cpp$ ./a.out
S      = 1.000003695487976
Sk     = 1.000048995018005
formella@pcformella:~/tex/teach/scrise/cpp$ ./a.out
S      = 1.000004649162292
Sk     = 1.000049471855164
formella@pcformella:~/tex/teach/scrise/cpp$ ./a.out
S      = 1.000004887580872
Sk     = 1.000051617622375
formella@pcformella:~/tex/teach/scrise/cpp$ ./a.out
S      = 1.00000524520874
Sk     = 1.000051736831665
formella@pcformella:~/tex/teach/scrise/cpp$
```



dificultades añadidas

- la entrada y salida de valores de números flotantes muchas veces no es recíproco si se trabaja con decimales (hay que tener cuidado)
- algunos procesadores trabajan internamente con precisión más alta (80 bits, por ejemplo, para algunos procesadores de Intel)
- no todas las operaciones a nivel de ensamblador dan soporte a precisiones más altas
- depende del caso concreto puede ser que resultados intermedios se graban a memoria
- eso dificulta, entre otras cosas, la generación de *foto fija* (*snapshot*) en una simulación numérica
- `long double` muchas veces se implementa con menos de 128 bits en las operaciones (80 bits, por ejemplo)
- hay solamente unos pocos sistemas (*main frames*) que tienen procesadores con 128 bits reales (por ejemplo IBM o NEC)



¿que podemos hacer?

- usar métodos de cálculo **robustos** en relación a la propagación de errores de redondeo
- se tiene que tener **cuidado con simplificar** expresiones
- se tiene que tener **cuidado con procesos de optimización** en los compiladores
- el algoritmo más eficiente, quizá no es el algoritmo más robusto, hay que **elegir bien**
- realizar un banco de pruebas incluyendo **pruebas de robustez**
- evitar creer ciegamente en el **determinismo sugerido**
- si se hace numérica se necesita un/a **matemático/a** y un/a **informático/a**

comparación aproximativa de valores flotantes

```
1  #include <algorithm>
2  #include <iomanip>
3  #include <iostream>
4  #include <random>
5  #include <vector>
6
7  #define N 10000000
8
9  int main() {
10     std::random_device rd {};
11     std::mt19937 gen {rd()};
12     std::lognormal_distribution<> normDisBase {0.0,0.01};
13     std::vector<float> data(N);
14     for(int i=0; i<N; ++i) data[i]=(float)normDisBase(gen);
15
16     float S=0.0;
17     for(int j=0; j<N; ++j) S+=data[j];    // suma simple
18     std::cout<<"S      = "<<std::setprecision(16)<<S/N<<"\n";
19
20     float Sk=0.0;
21     float C=0.0;
22     for(int j=0; j<N; ++j) {                // suma Kahan's method
23         const float Y=data[j]-C;
24         const float T=Sk+Y;
25         C=(T-Sk)-Y;
26         Sk=T;
27     }
28     std::cout<<"Sk      = "<<std::setprecision(16)<<Sk/N<<"\n";
29     return 0;
30 }
```

comparación con tolerancia absoluta

```
1  #include <iomanip>
2  #include <iostream>
3  #include <cmath>
4
5  bool AlmostEqualAbsolute(const float A, const float B, const float eps) {
6      return fabs(A-B) < eps;
7  }
8
9  int main() {
10     const float A=10000;
11     const float B=10000.001;
12     const float eps=0.00001;
13     std::cout<<std::setprecision(20)
14         <<A<<" almost equal to "
15         <<B<<" : "<<(AlmostEqualAbsolute(A,B,eps)?"true":"false")
16         <<" with eps "<<eps<<"\n";
17     return 0;
18 }
```

comparación con tolerancia relativa

```
1  #include <iomanip>
2  #include <iostream>
3  #include <cmath>
4
5  bool AlmostEqualRelative(const float A, const float B, const float eps) {
6      if (A==B) return true;
7      const float relativeError=fabs((A-B)/B);
8      cout<<"rel err "<<relativeError<<"\n";
9      return relativeError<=eps;
10 }
11
12 int main() {
13     const float A=1.0;
14     const float B=0.001;
15     const float eps=1.0;
16
17     std::cout<<std::setprecision(20)
18         <<A<<" almost equal to "
19         <<B<<" : "<<(AlmostEqualRelative(A,B,eps)?"true":"false")
20         <<" with eps "<<eps <<"\n"
21         <<B<<" almost equal to "
22         <<A<<" : "<<(AlmostEqualRelative(B,A,eps)?"true":"false")
23         <<" with eps "<<eps <<"\n"
24     ;
25     return 0;
26 }
```

comparación con tolerancia relativa (simétrica)

```
1  #include <iomanip>
2  #include <iostream>
3  #include <cmath>
4
5  bool AlmostEqualRelative(const float A, const float B, const float eps) {
6      if(A==B) return true;
7      float relativeError;
8      if(fabs(B)>fabs(A))
9          relativeError=fabs((A-B)/B);
10     else
11         relativeError=fabs((A-B)/A);
12     return relativeError<=eps;
13 }
14
15 int main() {
16     const float A=0.00000001;
17     const float B=-0.00000001;
18     const float eps=1.0;
19
20     std::cout<<std::setprecision(20)
21         <<A<<" almost equal to "
22         <<B<<" : "<<(AlmostEqualRelative(A,B,eps)?"true":"false")
23         <<" with eps "<<eps <<"\n"
24         <<B<<" almost equal to "
25         <<A<<" : "<<(AlmostEqualRelative(B,A,eps)?"true":"false")
26         <<" with eps "<<eps <<"\n"
27     ;
28     return 0;
29 }
```

comparación con tolerancia relativa y absoluta

```
1  #include <iomanip>
2  #include <iostream>
3  #include <cmath>
4
5  bool AlmostEqualRelative(const float A, const float B, const float eps, const float eps_abs)
6  {
7      if (fabs(A-B) < eps_abs) return true;
8      float relativeError;
9      if (fabs(B) > fabs(A))
10         relativeError = fabs((A-B)/B);
11     else
12         relativeError = fabs((A-B)/A);
13     return relativeError <= eps;
14 }
15
16 int main() {
17     const float A=0.00000001;
18     const float B=-0.00000001;
19     const float eps=1.0;
20     const float eps_abs=0.0001;
21
22     std::cout << std::setprecision(20)
23         << A << " almost equal to "
24         << B << " : " << (AlmostEqualRelative(A,B,eps,eps_abs) ? "true" : "false")
25         << " with eps " << eps << "\n"
26         << B << " almost equal to "
27         << A << " : " << (AlmostEqualRelative(B,A,eps,eps_abs) ? "true" : "false")
28         << " with eps " << eps << "\n"
29     ;
30     return 0;
31 }
```

comparación con tolerancia respecto a representación

```
1  #include <cassert>
2  #include <iomanip>
3  #include <iostream>
4  #include <cmath>
5
6  bool AlmostEqualRelative(const float A, const float B, const int ulps) {
7      // Make sure maxUlps is non-negative and small enough that the
8      // default NaN won't compare as equal to anything.
9      assert(ulps > 0 && ulps < 4*1024*1024);
10     int aInt = *(int*)&A;
11     // Make aInt lexicographically ordered as a twos-complement int
12     if(aInt<0) aInt=0x80000000-aInt;
13     // Make bInt lexicographically ordered as a twos-complement int
14     int bInt = *(int*)&B;
15     if(bInt<0) bInt=0x80000000-bInt;
16     int intDiff = abs(aInt-bInt);
17     return intDiff<=ulps;
18 }
19 int main() {
20     const float A=1.000001f;
21     const float B=1.0f;
22     const int eps=1;
23
24     std::cout<<std::setprecision(20)
25         <<A<<" almost equal to "
26         <<B<<" : "<<(AlmostEqualRelative(A,B,eps)?"true":"false")
27         <<" with eps "<< eps <<"\n"
28         <<B<<" almost equal to "
29         <<A<<" : "<<(AlmostEqualRelative(B,A,eps)?"true":"false")
30         <<" with eps "<< eps <<"\n"
31     ;
32     return 0;
33 }
```

¿que podemos hacer?

- se tiene que analizar que **tipo de comparación** es la correcta para el algoritmo implementado
- se tiene que tener **cuidado con las tolerancias** escogidas
- se tiene que tener **cuidado con la representación** de números flotantes en el hardware en concreto (especialmente las codificaciones especiales: NaN, Inf, sub-normal, etc.)
- el algoritmo más optimizado, quizá no es el algoritmo más robusto, hay que **elegir bien**
- realizar un banco de pruebas incluyendo **pruebas de tolerancias**
- si se hace numérica se necesita un/a **matemático/a** y un/a **informático/a**

una cosa difícil

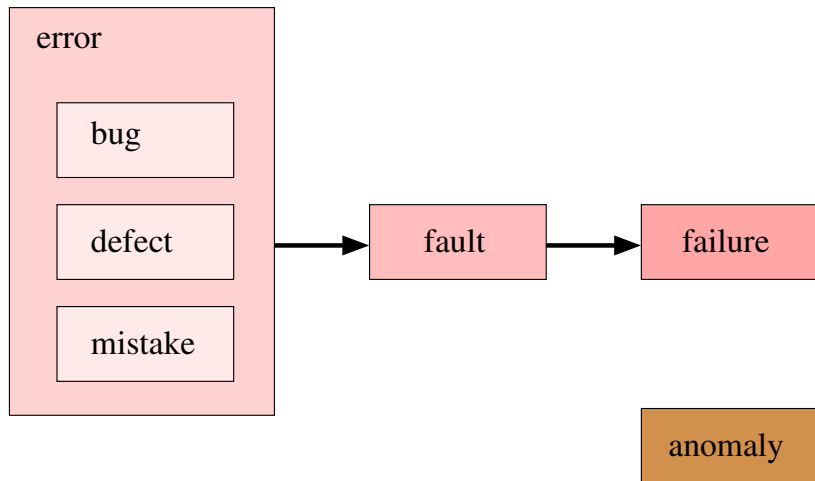
mirando muchas fuentes, parece **difícil concretar** el significado de las siguientes palabras (y sus traducciones a otros idiomas):

English	castellano	Deutsch
<i>bug</i>	bug	Bug
<i>defect</i>	defecto	Defekt/Mangel
<i>mistake</i>	malentendido	Irrtum
<i>error</i>	error	Fehler
<i>fault</i>	fallo	Versagen
<i>failure</i>	malfuncionamiento	Fehlverhalten
<i>anomaly</i>	anomalía	Anomalie

vamos a concretar la semántica para estas clases...

(hay más palabrería: *mishap, incidence, glitch, flaw, ...*)

relación entre los términos



- un bug es un error en el código del software
- un bug se puede encontrar y remediar/corregir
- un bug puede ser la consecuencia de un malfuncionamiento de una herramienta de generación de código automático
- un bug puede provocar grandes desastres
- si hay un bug, sufre el usuario, no el o la programadora

defecto (*defect*)

- un defecto está localizado en el hardware (no en el software)
- un defecto puede existir desde el principio o por rotura durante el uso
- un defecto se puede reparar (por ejemplo, sustituyendo una parte defectuosa)
- existen defectos transitorios, es decir, se manifiestan y desaparecen, o re-aparecen con cierta probabilidad
- existen defectos que se pueden *remediar* en software
- defectos suelen ser raros en sistemas críticos asumiendo que se usan componentes fiables y comprobados (incluyendo los procesos de montaje)
- un defecto se puede (muchas veces) simular en software
- no todos los defectos causan malfuncionamiento, especialmente si hay redundancia implementada, o se usa, por ejemplo, autosaneamiento implementado en hardware

