1.4 relative error

3.14

abs error ≤ 0.01

rel. error

"3 digits"

In computations we often cannot determine the exact value of some real value $x \in \mathbb{R}$. Instead, some value $\tilde{x} \in \mathbb{R}$ is computed which hopefully is close to x in some sense. This closeness or accuracy of $\tilde{x}$ we describe with statements like $\tilde{x}$ *is accurate to three (decimal) digits*. This concept of accuracy is modelled by the concept of *relative error*.

Definition: An approximation $\tilde{x}$ of a real number x has a relative error δ if

$$\boxed{\tilde{x} = (1 + \delta)x.}$$

The value of δ is thus

$$\delta = \frac{\tilde{x} - x}{x}$$

abs. error

exact value

in the case where $x \neq 0$. Note that δ is well defined for any number $\tilde{x}$.

$|\delta| \leq \frac{0.01}{\pi}$ rel. error $\leq \frac{1}{300}$

module

For example, consider the approximation of 3.14 for π.

```
from math import pi
delta = (pi - 3.14)/pi
print("relative error of 3.14: {}".format(delta))
```

$\frac{3.14 - pi}{pi}$

```
relative error of 3.14: 0.0005069573828972128
```

$\frac{1}{300}$

$\approx \frac{1}{2000}$

In practice, we will not know the (exact) value x. Thus the value of x is uncertain. In error analysis we aim to determine bounds for relative error δ of $\tilde{x}$ based on the properties of the computations performed.

For example, we know that the floating point arithmetic used in Python has a 53 bit mantissa (t=53) and a base $B = 2$. From this one can see that the relative error occurring in optimal rounding satisfies

$$|\delta| \le \epsilon = \frac{1}{2B^{t-1}}$$

$$\frac{m}{2^t} \cdot 2^e$$

$$m = 2^{t-1} \ldots 2^t - 1$$

which in our case is

```
B = 2.0
t = 53
epsilon = 0.5/B**(t-1)
print("bound of relative error {}".format(epsilon))
```

```
bound of relative error 1.1102230246251565e-16
```

We now check how well a number we input is rounded in Python. We consider $x = 3.45$ and use decimal arithmetic. With this we get the exact value of the difference $\tilde{x} - x$.

```
from decimal import Decimal
x = Decimal("3.45")
xtilde = 3.45
delta = (Decimal(xtilde) - x)/x
print("relative error delta = {}".format(delta))
```

```
relative error delta = 5.148860404058696999066117881E-17
```

As this is less than the bound, we may also conclude that the rounding is optimal in this case.

Given a relative error, one can now determine the number of (significant) digits of $\tilde{x}$ are accurate by using the 10 based logarithm. For our example of π we get

```
from math import pi, log
delta = (pi - 3.14)/pi
digits = round(-log(abs(delta))/log(10))
print("accurate digits of 3.14: {}".format(digits))
```

```
accurate digits of 3.14: 3
```

For the rounding error bound we get the number of accurate digits we expect in that case to be

```
B = 2.0
t = 53
epsilon = 0.5/B**(t-1)
digits = round(-log(abs(epsilon))/log(10))
print("accurate digits after rounding: {}".format(digits))
```

```
accurate digits after rounding: 16
```