UNIVERSIDAD NACIONAL DE ASUNCIÓN

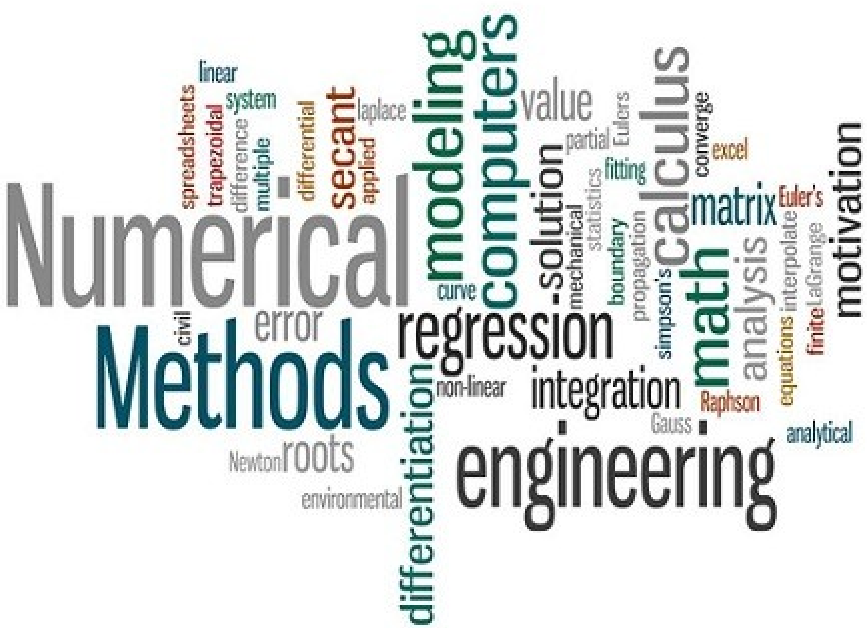
FACULTAD POLITÉCNICA

DEPARTAMENTO DE INVESTIGACIÓN, POSTGRADO Y EXTENSIÓN

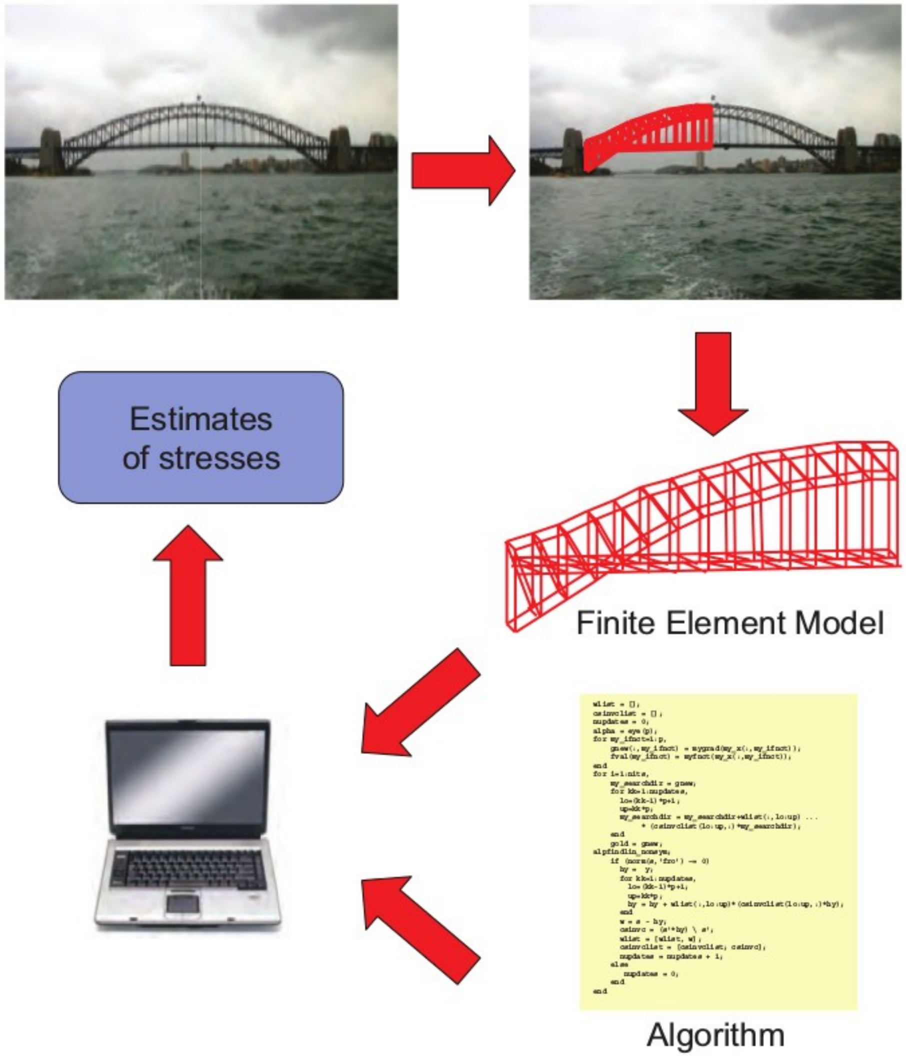
Postgrado en Ciencias de la Computación

METODOS NUMÉRICOS

Prof. Diego Stalder



Fuentes de Error



- Medida.
- Modelo.
- Truncamiento.
- Redondeo.

Como el computador hace los cálculos?

- Aritmética de punto fijo.

Each word (storage location) in a machine contains a fixed number of digits. (Base 10 o Base 2?, Signos?)

0	0	1	9	8	5

- Aritmética de punto flotante.

If we wanted to store 15×2^{11} , we would need 16 bits:

0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

Instead, let's agree to code numbers as **two** fixed point binary numbers:

$z \times 2^{p}$,   with $z = 15$ saved as 01111 and $p = 11$ saved as 01011.

Aritmética de Punto Flotante

Single-precision numbers, 24 digits are used to represent the mantissa, and the exponent is restricted to the range $-126 \leq p \leq 127$.

This allows us to represent numbers as close to zero as $2^{-126} \approx 1.18 \times 10^{-38}$ and as far as almost $2^{128} \approx 3.40 \times 10^{3}$

Double-precision numbers, stored in two words, using 53 digits for the mantissa, with an exponent $-1022 \leq p \leq 1023$.

If we perform a computation in which the exponent of the answer is outside the allowed range, we have a more or less serious error.

- Overflow.
- Underflow.
- Not-a-number.

Errores de truncamiento y redondeo

For normalized floating-point numbers, this proportionality can be expressed, for cases where chopping is employed, as

$$\frac{|\Delta x|}{|x|} \leq \mathscr{E}$$

and, for cases where rounding is employed, as

$$\frac{|\Delta x|}{|x|} \leq \frac{\mathscr{E}}{2}$$

$$\mathscr{E} = b^{1-t}$$

Como medimos el error?

Absolute error in c as an approximation to x:

$$|x - c|$$

Relative error in c as an approximation to nonzero x:

$$\frac{|x - c|}{|x|}$$

Propagación de Errores

función	Error	Error de la función
$q = x \pm y$	$x \pm \Delta x,$ $y \pm \Delta y$	$\Delta q = \Delta x + \Delta y$ $\Delta q = \sqrt{\Delta x^2 + \Delta y^2}$
$q = xy$	$x \pm \Delta x,$ $y \pm \Delta y$	$\frac{\Delta q}{\lvert q\rvert} = \frac{\Delta x}{\lvert x\rvert} + \frac{\Delta y}{\lvert y\rvert}$
$q = \frac{x}{y}$	$x \pm \Delta x,$ $y \pm \Delta y$	$\frac{\Delta q}{\lvert q\rvert} = \frac{\Delta x}{\lvert x\rvert} + \frac{\Delta y}{\lvert y\rvert}$
$q = Ax$	$x \pm \Delta x$	$\Delta q = \lvert A\rvert \Delta x$
$q = x^n$	$x \pm \Delta x$	$\frac{\Delta q}{\lvert q\rvert} = n\frac{\Delta x}{\lvert x\rvert}$
$q = Ax^n y^m$	$x \pm \Delta x$ $y \pm \Delta y$	$\frac{\Delta q}{\lvert q\rvert} = \sqrt{\left(n\frac{\Delta x}{\lvert x\rvert}\right)^2 + \left(m\frac{\Delta y}{\lvert y\rvert}\right)^2}$
$q = f(x)$	$x \pm \Delta x$	$\Delta q = \left\lvert\frac{df(x)}{dx}\right\rvert \Delta x$
$q = f(x,y)$	$x \pm \Delta x,$ $y \pm \Delta y$	$\Delta q = \left\lvert\frac{\partial f}{\partial x}\right\rvert \Delta x + \left\lvert\frac{\partial f}{\partial y}\right\rvert \Delta y$
$q = f(x, y, \dots, w)$	$x \pm \Delta x,$ $y \pm \Delta y,$ $w \pm \Delta w$	$\delta q = \sqrt{\left(\frac{\partial f}{\partial x}\Delta x\right)^2 + \left(\frac{\partial f}{\partial y}\Delta y\right)^2 + \left(\frac{\partial f}{\partial w}\Delta w\right)^2}$

https://www.uv.es/zuniga/3.2_Propagacion_de_errores.pdf

Como medimos el error?

- Cuando sumamos o restamos los errores absolutos se suman.
- Cuando multiplicamos o dividimos, los errores relativos se suman(aproximadamente)
- Pero también podemos tener un error adicional, por ejemplo, al cortar o redondear la respuesta.
- Los límites de error son útiles, pero pueden ser muy pesimistas.

Como medimos el error?

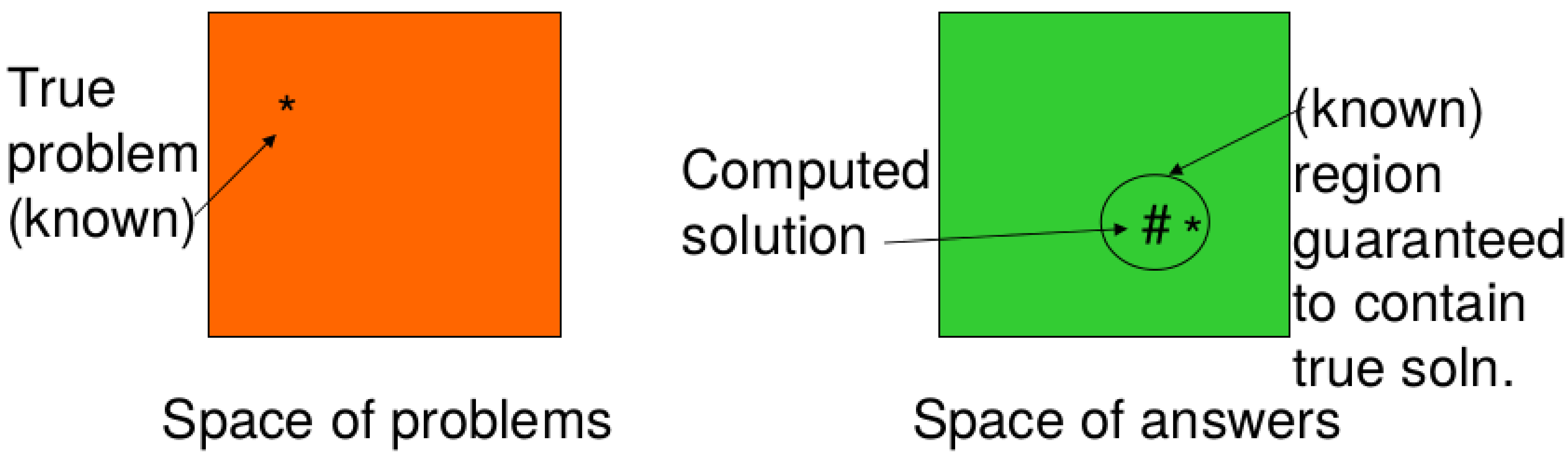
Error analysis determines the cumulative effects of error.

Two approaches:

- Forward error analysis
- Backward error analysis

Como medimos el error?

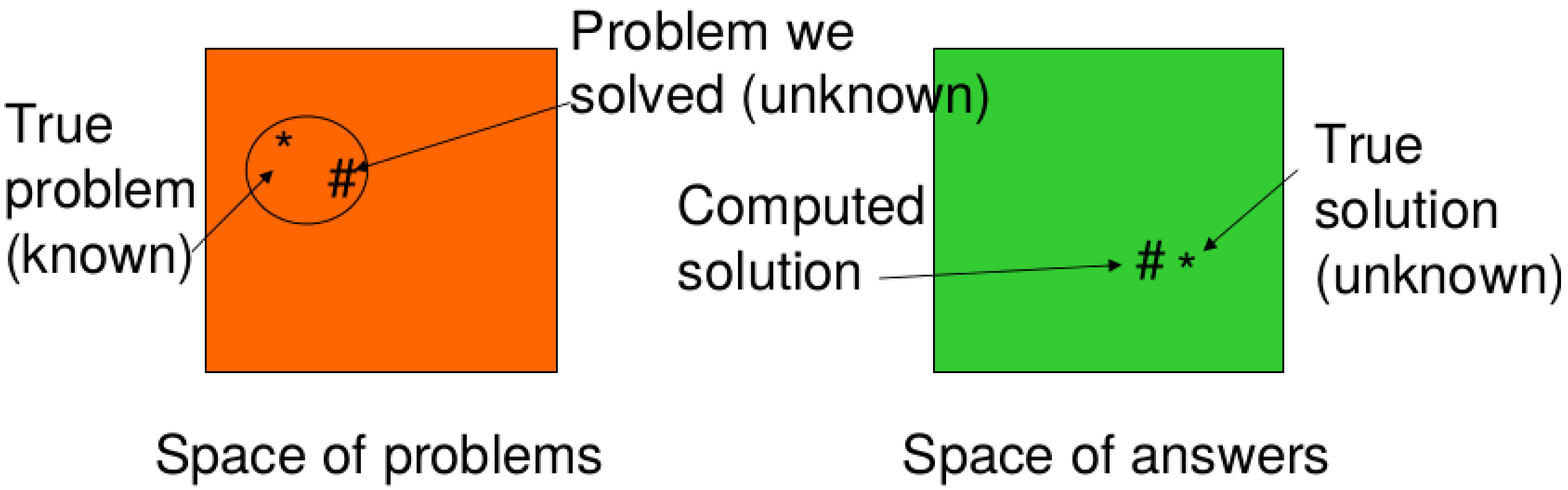
Forward Error analysis



$$\|x_{true} - x_c\|$$

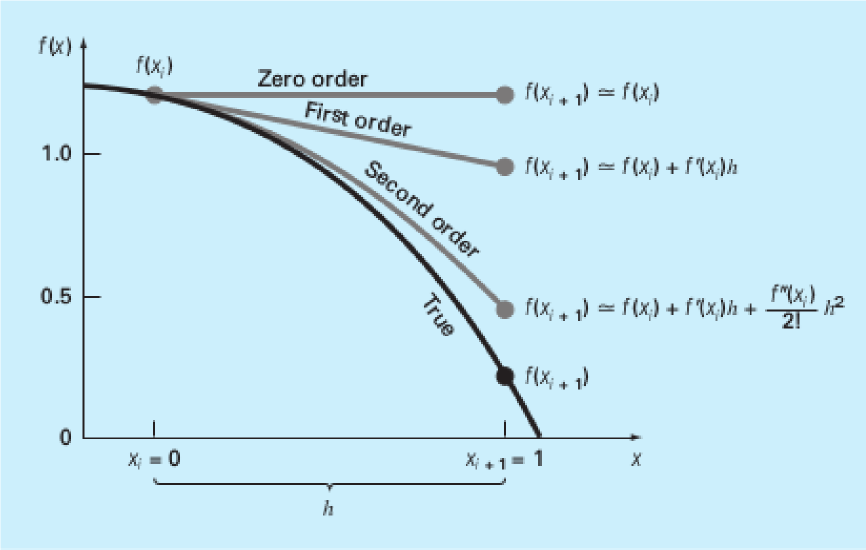
Como medimos el error?

Backward Error analysis



$$r = b - Ax_c.$$

Errores de truncamiento



$$f(x_{i+1}) = f(x_i) + f'(x_i)h + \frac{f''(x_i)}{2!}h^2 + \frac{f^{(3)}(x_i)}{3!}h^3 + \cdots + \frac{f^{(n)}(x_i)}{n!}h^n + R_n \tag{4.7}$$

where the remainder term is now

$$R_n = \frac{f^{(n+1)}(\xi)}{(n+1)!}h^{n+1} \tag{4.8}$$

Introducción a Python

Python is an object-oriented language that was developed in the late 1980s as a scripting language (the name is derived from the British television series, Monty Python’s Flying Circus)

Python programs are not compiled into machine code, but are run by an interpreter.

The great advantage of an interpreted language is that programs can be tested and debugged quickly, allowing the user to concentrate more on the principles behind the program and less on the programming itself.

Python is an open-source software, which means that it is free.

Aritmética de Punto Flotante

Data type	Description
`bool_`	Boolean (True or False) stored as a byte
`int_`	Default integer type (same as C `long`; normally either `int64` or `int32`)
intc	Identical to C `int` (normally `int32` or `int64`)
intp	Integer used for indexing (same as C `ssize_t`; normally either `int32` or `int64`)
int8	Byte (-128 to 127)
int16	Integer (-32768 to 32767)
int32	Integer (-2147483648 to 2147483647)
int64	Integer (-9223372036854775808 to 9223372036854775807)
uint8	Unsigned integer (0 to 255)
uint16	Unsigned integer (0 to 65535)
uint32	Unsigned integer (0 to 4294967295)
uint64	Unsigned integer (0 to 18446744073709551615)
`float_`	Shorthand for `float64`.
float16	Half precision float: sign bit, 5 bits exponent, 10 bits mantissa
float32	Single precision float: sign bit, 8 bits exponent, 23 bits mantissa
float64	Double precision float: sign bit, 11 bits exponent, 52 bits mantissa
`complex_`	Shorthand for `complex128`.
complex64	Complex number, represented by two 32-bit floats (real and imaginary components)
complex128	Complex number, represented by two 64-bit floats (real and imaginary components)

Additionally to `intc` the platform dependent C integer types `short`, `long`, `longlong` and their unsigned versions are defined.

Variables

```
>>> b = 2          # b is integer type
>>> print(b)
2
>>> b = b*2.0   # Now b is float type
>>> print(b)
4.0
```

Cadenas

```
>>> string1 = 'Press return to exit'
>>> string2 = 'the program'
>>> print(string1 + ' ' + string2)  # Concatenation
Press return to exit the program
>>> print(string1[0:12])            # Slicing
Press return
```

Tuplas

```
>>> rec = ('Smith','John',(6,23,68))    # This is a tuple
>>> lastName,firstName,birthdate = rec  # Unpacking the tuple
>>> print(firstName)
John
>>> birthYear = birthdate[2]
>>> print(birthYear)
68
>>> name = rec[1] + ' ' + rec[0]
>>> print(name)
John Smith
>>> print(rec[0:2])
('Smith', 'John')
```

Listas

```
>>> a = [1.0, 2.0, 3.0]         # Create a list
>>> a.append(4.0)               # Append 4.0 to list
>>> print(a)
[1.0, 2.0, 3.0, 4.0]
>>> a.insert(0,0.0)             # Insert 0.0 in position 0
>>> print(a)
[0.0, 1.0, 2.0, 3.0, 4.0]
>>> print(len(a))               # Determine length of list
5
>>> a[2:4] = [1.0, 1.0, 1.0] # Modify selected elements
>>> print(a)
[0.0, 1.0, 1.0, 1.0, 1.0, 4.0]
```

```
>>> a = [1.0, 2.0, 3.0]
>>> b = a               # 'b' is an alias of 'a'
>>> b[0] = 5.0          # Change 'b'
>>> print(a)
[5.0, 2.0, 3.0]         # The change is reflected in 'a'
>>> c = a[:]            # 'c' is an independent copy of 'a'
>>> c[0] = 1.0          # Change 'c'
>>> print(a)
[5.0, 2.0, 3.0]         # 'a' is not affected by the change
```

Operadores aritméticos

+	Addition
−	Subtraction
*	Multiplication
/	Division
**	Exponentiation
%	Modular division

a += b	a = a + b
a -= b	a = a - b
a *= b	a = a*b
a /= b	a = a/b
a **= b	a = a**b
a %= b	a = a%b

```
>>> s = 'Hello '
>>> t = 'to you'
```

```
>>> a = [1, 2, 3]
>>> print(3*s)              # Repetition
Hello Hello Hello
>>> print(3*a)              # Repetition
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> print(a + [4, 5])       # Append elements
[1, 2, 3, 4, 5]
>>> print(s + t)            # Concatenation
Hello to you
>>> print(3 + s)            # This addition makes no sense
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    print(3 + s)
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Operadores aritméticos

<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to

Bloque Condicional

```
def sign_of_a(a):
    if a < 0.0:
        sign = 'negative'
    elif a > 0.0:
        sign = 'positive'
    else:
        sign = 'zero'
    return sign

a = 1.5
print('a is ' + sign_of_a(a))
```

Running the program results in the output

```
a is positive
```

Bloque repetitivo

```
nMax = 5
n = 1
a = []                # Create empty list
while n < nMax:
    a.append(1.0/n)   # Append element to list
    n = n + 1
print(a)
```

The output of the program is

```
[1.0, 0.5, 0.33333333333333331, 0.25]
```

```
nMax = 5
a = []
for n in range(1,nMax):
    a.append(1.0/n)
print(a)
```

```
list = ['Jack', 'Jill', 'Tim', 'Dave']
name = eval(input('Type a name: '))  # Python input prompt
for i in range(len(list)):
    if list[i] == name:
        print(name,'is number',i + 1,'on the list')
        break
else:
    print(name,'is not on the list')
```

Here are the results of two searches:

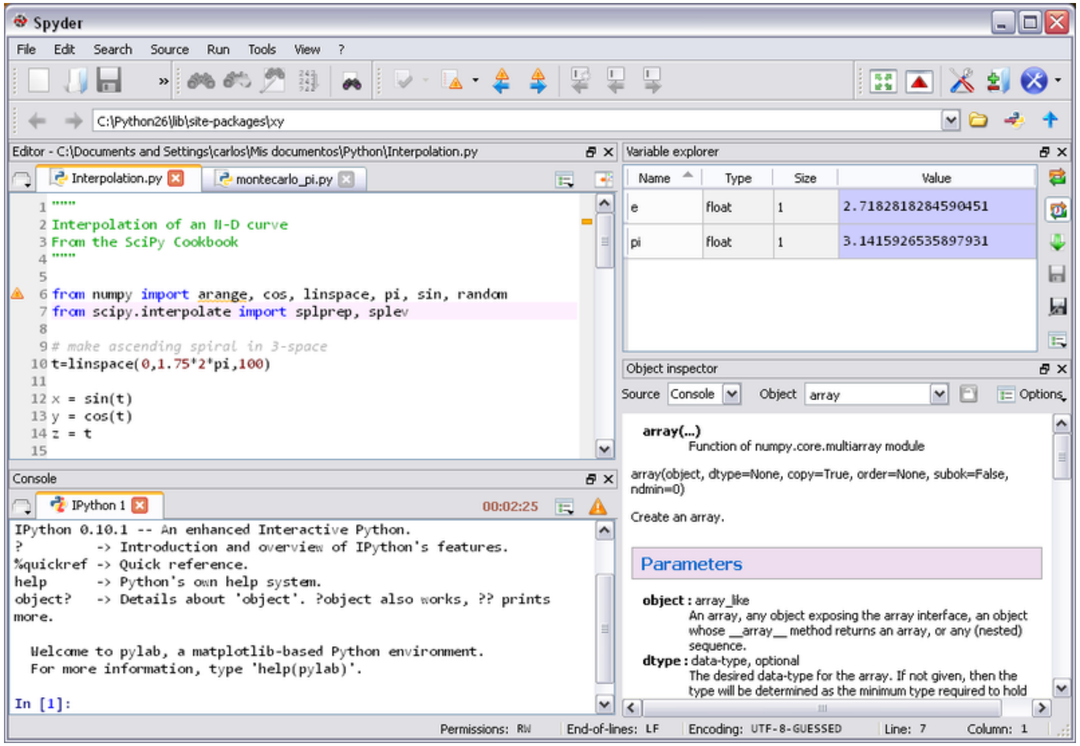
```
Type a name: 'Tim'
Tim is number 3 on the list

Type a name: 'June'
June is not on the list
```

Conversiones de Tipo

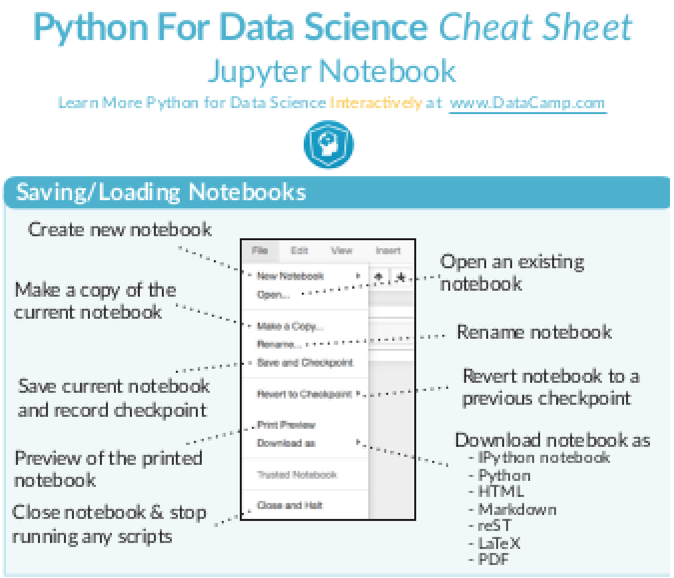
```
>>> a = 5
>>> b = -3.6
>>> d = '4.0'
>>> print(a + b)
1.4
>>> print(int(b))
-3
>>> print(complex(a,b))
(5-3.6j)
>>> print(float(d))
4.0
>>> print(int(d))  # This fails: d is a string
Traceback (most recent call last):
  File "<pyshell#30>", line 1, in <module>
    print(int(d))
ValueError: invalid literal for int() with base 10: '4.0'
```

Ambiente de programación-IDE

https://www.datacamp.com/community/tutorials/data-science-python-ide



Ambiente de programación



https://www.datacamp.com/community/blog/jupyter-notebook-cheat-sheet

Bibliotecas

Python For Data Science *Cheat Sheet*

NumPy Basics

Learn Python for Data Science **Interactively** at www.DataCamp.com

NumPy

The **NumPy** library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

```
>>> import numpy as np
```

NumPy

Bibliotecas

Python For Data Science *Cheat Sheet*

Matplotlib

Learn Python **Interactively** at www.DataCamp.com

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

Bibliotecas

Python For Data Science *Cheat Sheet*

SciPy - Linear Algebra

Learn More Python for Data Science Interactively at www.datacamp.com

SciPy

The **SciPy** library is one of the core packages for scientific computing that provides mathematical algorithms and convenience functions built on the NumPy extension of Python.