Introduzione alla programmazione per le scuole superiori

basics

27 nov 2024

Introduzione alla programmazione

Questo libro fa parte del materiale pensato per le scuole superiori. E" disponibile la versione in .pdf scaricabile.

Obiettivi generali. Questo lavoro punta ad essere un'opera di *formazione alla* **tirchieria**, **prigrizia** *e* **onestà**, almeno in ambito informatico. I più benvolenti potranno riassumere questo obiettivo come il desiderio di *non buttare nel WC soldi, tempo, e pazienza*.

Oltre alle nozioni minime, l'obiettivo principale di questo libro è la formazione a:

- **indipendenza** in ambito informatico: evitare di pagare per qualcosa che non serve; evitare di pagare per qualcosa di inutile o dannoso in presenza di alternative libere; meglio dedicare le risorse a ciò che vale la pena pagare
- **ordine**: i moderni strumenti informatici permettono di lavorare in maniera ordinata, risparmiando tempo, soldi e pazienza
- **trasparenza** e **onestà**: anche se spesso in maniera non lineare, la conoscenza procede seguendo il metodo scientifico: i risultati mostrati e le tesi proposte devono essere supportate da dati e logica; i dati e le analisi svolte per poter produrre risultati devono essere disponibili, controllabili e soggetti a critica. Tutto il resto, almeno qui, almeno nell'ambito della conoscenza che procede con il metodo scientifico, sarà considerata confusione nella migliore delle ipotesi o direttamente *merda*.

Questo stesso libro è scritto seguendo questi criteri: oltre al dispositivo elettronico usato per consultare il materiale (online o offline, una volta scaricato), non è necessaria la spesa per nessun altro dispositivo o infrastruttura informatica; i sorgenti del materiale è sviluppato localmente, ospitato e disponibile su Github all'indirizzo https://github.com/Basics2022/bbooks-programming-hs.

necessità di una connessione internet, se non si porta il progetto su un sistema locale, con tutti gli strumenti necessari - non tanti, e standard, ma comunque devono esserci «per funzionare»

Metodo.

- **Impostazione degli strumenti necessari. todo**…
- **Linguaggio di programmazione.** In questa introduzione si sceglie di usare **Python** come linguaggio di programmazione. Un approccio più generale all'informatica e alla programmazione prevederebbe l'utilizzo di altri linguaggio di programmazione (come C). Considerata la **diffusione** di Python, la quantità di **librerie disponibili** (con eventuali binding a librerie sviluppate usando linguaggi di programmazione più efficienti) e strumenti per la **programmazione collaborativa** e remota (**Colab**, **Jupyter**,…), Python risulta comunque una buona scelta per un corso improntato alla presentazione delle basi di programmazione dirette a un'applicazione abbastanza immediata.

  Verranno posta attenzione sulla sintassi «particolare» di Python quando si discosta maggiormente dagli altri linguaggi di programmazione.
- **Argomenti. todo**…

Parte I

Introduzione alla programmazione

CAPITOLO 1

Introduzione alla programmazione - in Python

L'introduzione alla programmazione in Python userà le risorse messe a disposizione da Google con il progetto Jupyter per lo sviluppo di codice open-source, con open-standard e servizi interattivi utilizzabili su diversi dispositivi usando diversi linguaggi di programmazione, come Python, Julia o R

Oltre al dispositivo elettronico utilizzato per consultare al materiale, non è necessario nessun altro dispositivo informatico: un account Google personale permette l'accesso libero ai servizi base di cloud computing di Colab

CAPITOLO 2

Variabili, tipi e funzioni elementari built-in

- tipi
- variabili
  - …
  - by-reference o by-value

2.1 Commenti

```
# I commenti permettono di aggiungere brevi descrizione al codice

# In Python, è possibile aggiungerre commenti al codice con il carattere #: tutto␣
 ↪quello
# che viene dopo il carattere # su una riga è considerato un commento, e non codice da
# eseguire

# E' buona regola aggiungere qualche commento al codice, e sarebbe bene iniziare a␣
 ↪farlo
# in lingua inglese:
# - il codice non è auto-esplicativo!
# - il codice potrebbe essere usato da altri in giro per il mondo, ed è più probabile␣
 ↪che
#   si conosca l'inglese invece dell'italiano

# So let's switch to English for scripts, both for comments and "for variable names"

# Comments are not documentation! **todo** Add some paragraph about documentation!
```

2.2 Tipi built-in

- numero: intero, reale, complesso
- booleano
- stringa
- bytes
- lista
- tupla
- insieme
- dizionario, `dict`

2.2.1 Numeri

```
"""
Numbers

in Python, variables are not delcared. Thus, a number variable is not defined
as an integer, a real or a complex variable, but it's type is inferred by its
initialization
"""

# Numbers
a_int = 1
a_real = 1.
a_complex = 1.+ 0.j

# Strings
a_str = '1.0'

print(f"type(a_int)    : {type(a_int    )}")
print(f"type(a_real)   : {type(a_real   )}")
print(f"type(a_complex): {type(a_complex)}")
print(f"type(a_str    ): {type(a_str    )}")
```

```
type(a_int)    : <class 'int'>
type(a_real)   : <class 'float'>
type(a_complex): <class 'complex'>
type(a_str    ): <class 'str'>
```

2.2.2 Booleani - logici

2.2.3 Stringhe

```
"""
Strings

strings are character
"""
```

```
'\nStrings\n\nstrings are character \n'
```

2.2.4 Liste, tuple e insiemi

2.2.5 Dizionari

CAPITOLO 3

Controllo del flusso

Nei paradigmi di *programmazione imperativa* (**todo** fare riferimento ai paradigmi di programmazione. Ha senso questa distinzione?), vengono usate delle strutture di controllo del flusso di esecuzione di un programma.

Si possono distinguere due categorie delle strutture di controllo:

- condizionale ed alternativa: if, if-then, if-then-else
- iterazione: for, while, …

3.1 Alternativa

3.1.1 `if-then` statement

```
""" if-then example """
# Try this script changing the user input

# User input
a = 2         # a is initialize as an integer
# a = 2.1       # if a is initialized as a real, a % 2 perform automatic casting,␣
 ↪int(a) % 2
                # uncomment previous line and try!

word = 'odd'
if ( a % 2 == 0 ):    # automatic casting into an int -> a % 2 = floor(a) % 2
    word = 'even'

print(f"User input ({a}) is an {word} number")
```

```
User input (2) is an even number
```

3.1.2 `if-then-else` statement

```
""" if-then-else example"""
# Try this script changing the user input

# User input
a = 15

if ( a % 3 == 1 ):      # First condition
    reminder = 1
elif ( a % 3 == 2 ):    # Other condition
    reminder = 2
else:                   # All the other conditions
    reminder = 0

print(f"User input ({a}). {a} % 3 = {reminder}")
```

```
User input (15). 15 % 3 = 0
```

3.2 Iterazione

3.2.1 for loop

```
""" for loop examples

Loops over:
- elements of a list
- elements in range
- keys, values of a dict
- ...
"""

# Loop over elements of a list
seq = ['a', 3, 4. , {'key': 'value'}]

print("\nLoop over elements of the list: seq = {seq}")
for el in seq:
    print(f"element {el} has type {type(el)}")

# Loop over elements of a tuple
# ...

# Loop over elements of a range
n_el = 5
range_el = range(5)
print("\nLoop over elements of the list, seq = {seq}")
print(f"range({n_el}) has type: {type(range_el)}")
print(f"range({n_el}): {range_el}")

for i in range_el:
    print(i)
```

(continues on next page)

(continua dalla pagina precedente)

```
# Loop over keys, values of a dict
d = {'a': 1., 'b': 6, 'c': {'c1': 1, 'c2': True}}
print(f"\nLoop over elements of the dict, d = {d}")

for i,k in d.items():
    print(i, k)
```

```
Loop over elements of the list: seq = {seq}
element a has type <class 'str'>
element 3 has type <class 'int'>
element 4.0 has type <class 'float'>
element {'key': 'value'} has type <class 'dict'>

Loop over elements of the list, seq = {seq}
range(5) has type: <class 'range'>
range(5): range(0, 5)
0
1
2
3
4

Loop over elements of the dict, d = {'a': 1.0, 'b': 6, 'c': {'c1': 1, 'c2': True}}
a 1.0
b 6
c {'c1': 1, 'c2': True}
```

3.2.2 while loop

```
""" while loop example """

a = 3

while ( a < 5 ):
    a += 1
    print(f">> in while loop, a: {a}")

print(f"after while loop, a: {a}")
```

```
>> in while loop, a: 4
>> in while loop, a: 5
after while loop, a: 5
```

3.2.3 altri cicli

todo

CAPITOLO 4

Funzioni

```
#> Define a function to tell if a number is positive or not
def is_positive(x):
    """ Function returning True if x>0, False if x<=0 """
    return x > 0

#> User input to test the function: tuple of numbers
n_tuple = [ -1, 2., .003, 1./3., -2.2, 0, -7.4 ]

#> Test function on all the elements in the user-defined tuple
for n in n_tuple:
    if ( is_positive(n) ):
        string = ''
    else:
        string = 'not '

    print(f'{n} is '+string+'positive')
```

```
-1 is not positive
2.0 is positive
0.003 is positive
0.3333333333333333 is positive
-2.2 is not positive
0 is not positive
-7.4 is not positive
```

4.1 Default value of function arguments

```
#> Define a user function to tell is the first argument is greater than the second.
# If no second argument is given, it's set = 0 (default value, defined in the␣
↪function)
def is_greater_than(x, y=0):
    """  """
    return is_positive(x-y)

a, b = -2, -3
print(f"Is {a} greater than {b}? is_greater_than({a}, {b}):"+str(is_greater_than(a,␣
↪b)))
print(f"Is {a} greater than  0 ? is_greater_than({a},    ):"+str(is_greater_than(a)))
```

```
Is -2 greater than -3? is_greater_than(-2, -3):True
Is -2 greater than  0 ? is_greater_than(-2,    ):False
```

CAPITOLO 5

Classi

- classi, metodi, oggetti
- ereditarietà, overloading

CAPITOLO 6

Librerie

- Definizione del concetto
- librerie «standard», scritte da qualcun'altro
- scrivere una libreria

Parte II

Introduzione al calcolo scientifico

CAPITOLO 7

Introduzione al calcolo scientifico

In questa introduzione al calcolo numerico, vengono presentati alcuni algoritmi. Dove sensato, viene implementata la versione elementare di alcuni di questi algoritmi. Per usi non didattici, e quando possibile, si raccomanda l'uso di algoritmi implementati in librerie disponibili, per questioni di tempo ed efficienza: è lavoro già fatto, da persone che lo sanno fare meglio di noi, controllato, migliorato nel corso degli anni, ottimizzato per ogni sistema e spesso in linguaggi di programmazione diversi da Python, come C o Fortran.

In questa introduzione viene fatto affidamento e uso di alcune librerie disponibili per Python:

- librerie con algoritmi e strumenti matematici per il calcolo numerico: NumPy, SciPy,…
- librerie per la creazione di grafici: Matplotlib, Plotly,…
- librerie per l'analisi dati e la statistica: pandas,…
- librerie per il machine learning: sci-kit, PyTorch,…
- …

Introduzione al calcolo numerico

- Equazioni lineari
- Equazioni non lineari
- Approssimazione di funzioni
- Derivate
- Integrali
- Equazioni differenziali ordinarie:
  - problema di Cauchy ai valori iniziali
  - problema ai valori al contorno
- Ottimizzazione, vincolata e non

Metodi per la statistica

Introduzione al machine learning

CAPITOLO 8

Sistemi lineari

La soluzione di sistemi lineari è un problema che compare in molte altre applicazioni di calcolo numerico.

Formalismo matriciale. Con il formalismo matriciale, un sistema di equazioni lineari può essere scritto come

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

Classificazione. In generale, i sistemi di equazioni lineari possono essere classificati:

- in base al numero di incognite n_u ed equazioni indipendenti n_e: $n_e = n_u$ sistemi determinati con un'unica soluzione; $n_e > n_u$ sistemi sovradeterminati: con nessuna soluzione in generale; $n_e < n_u$ sistemi indeterminati, con infinite soluzioni in generale
- in base alla «struttura» del sistema:
  - diagonale, tridiagonale, …
- in base al numero di coefficienti non-nulli della matrice **A**: sistemi con matrice **A** piena o **sparsa**; questa distinzione non è netta, ma il più delle volte risulta chiara dalla particolare applicazione/metodo.

Algoritmi. Esistono due grandi classi di metodi/algoritmi per la soluzione di sistemi lineari:

- i **metodi diretti**, che si basano su una fattorizzazione della matrice
- i **metodi indiretti**, che si basano sul calcolo di prodotti matrice-vettore

8.1 Sistemi lineari quadrati con matrici piene

In questa sezione si discute la soluzione di sistemi lineari quadrati con matrici piene con le funzoni disponibili nella libreria NumPy.

8.1.1 Esempio 1. Sistema quadrato determinato

Il sistema lineare

$$\begin{cases} x_1 + 2\,x_2 = 0 \\ x_1 + x_3 = -1 \\ x_1 + x_2 + x_3 = 1 \end{cases}$$

può essere riscritto con il formalismo matriciale nella forma $\mathbf{Ax} = \mathbf{b}$,

$$\underbrace{\begin{bmatrix} 1 & 2 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix}}_{\mathbf{b}}$$

e risolto grazie alla funzione `solve(A, b)` della libreria `numpy.linalg`.

```
"""
Linear systems with full square non-singular matrices
"""

import numpy as np

A = np.array([[1.,2.,0.], [1.,0.,1.], [1.,1.,1.]])
b = np.array([0.,-1.,1.])

x = np.linalg.solve(A, b)

print(f"Sol, x: {x}")
print(f"Proof : Ax = {A @ x}")    # Check that Ax = b
print(f"         b = {b}")
```

```
Sol, x: [-4.  2.  3.]
Proof : Ax = [ 0. -1.  1.]
         b = [ 0. -1.  1.]
```

8.1.2 Esempio 2. Sistemi quadrati non determinati

I sistemi lineari

$$\begin{cases} x_1 + 2\,x_2 = 1 \\ x_1 + x_3 = -1 \\ 2x_1 + 2\,x_2 + x_3 = 1 \end{cases} \quad , \quad \begin{cases} x_1 + 2\,x_2 = 0 \\ x_1 + x_3 = -1 \\ 2x_1 + 2\,x_2 + x_3 = 1 \end{cases}$$

sono due sistemi quadrati non determinati. Il primo sistema non ha soluzioni, mentre il secondo ne ha infinite della forma

$$(x_1, x_2, x_3) = (-2, 1, 1) + \alpha(2, -1, -2)\,, \quad \alpha \in \mathbb{R}\,.$$

Dopo aver riscritto i sistemi lineari con il formalismo matriciale, si può provare a risolverli usando la funzione `solve(A, b)` della libreria `numpy.linalg`. In entrambi i casi, la funzione `solve(A,b)` resitiuisce un errore, segnalando che la matrice del sistema lineare è singolare, definizione equivalente di sistemi non determinati.

- **todo** *dare interpretazione geometrica, fare grafico?*
- **todo** *spiegare motivo?*

- *Esistono algoritmi che trovano almeno una soluzione nel caso in cui ne esistano infinite?*: discutere gli algoritmi implementati nella funzione $numpy.linalg.solve()$ e rimandare alla documentazione della libreria; discutere altri algoritmi che rendono possibile trovare una soluzione
- *Esistono algoritmi che trovano una soluzione approssimata nel caso in cui non ne esistano?*: **minimi quadrati**, minimizzano l'errore, dare un'interpretazione geometrica

```
"""
Linear systems with full square singular matrices
"""

import numpy as np

A = np.array([[1.,2.,0.], [1.,0.,1.], [2.,2.,1.]])
b = np.array([1.,-1.,1.])
# b = np.array([0.,-1.,1.])

x = np.linalg.solve(A, b)

print(f"Sol, x: {x}")
print(f"Proof : Ax = {A @ x}")    # Check that Ax = b
print(f"         b = {b}")
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
Cell In[2], line 11
      8 b = np.array([1.,-1.,1.])
      9 # b = np.array([0.,-1.,1.])
---> 11 x = np.linalg.solve(A, b)
     13 print(f"Sol, x: {x}")
     14 print(f"Proof : Ax = {A @ x}")    # Check that Ax = b

File <__array_function__ internals>:200, in solve(*args, **kwargs)

File ~/.local/lib/python3.8/site-packages/numpy/linalg/linalg.py:386, in solve(a,␣
 ↪b)
    384 signature = 'DD->D' if isComplexType(t) else 'dd->d'
    385 extobj = get_linalg_error_extobj(_raise_linalgerror_singular)
--> 386 r = gufunc(a, b, signature=signature, extobj=extobj)
    388 return wrap(r.astype(result_t, copy=False))

File ~/.local/lib/python3.8/site-packages/numpy/linalg/linalg.py:89, in _raise_
 ↪linalgerror_singular(err, flag)
     88 def _raise_linalgerror_singular(err, flag):
---> 89     raise LinAlgError("Singular matrix")

LinAlgError: Singular matrix
```

8.2 Matrici sparse

Una matrice sparsa ha un elevato numero di elementi nulli. Una matrice sparsa viene definita in maniera efficiente salvando in memoria solo gli elementi non nulli (**limiti di memoria**); gli algoritmi per le matrici sparse risultano spesso efficienti perché evitano un molte operazioni che darebbero risultati parziali nulli (**velocità**).

- **todo** dire due parole sui formati
- **todo** fare esempio di calcolo del prodotto matrice vettore per matrici sparse

8.2.1 Esempio 1 - Matrice di rigidezza di elementi finiti

Il sistema lineare

$$\begin{bmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix},$$

è descritto da una matrice, $N = 5$, $N \times N = 25$, che ha $N + (N-1) + (N-1) = 13$ elementi non nulli. Il rapporto tra il numero di elementi non nulli e il numero di elementi totali è $\frac{3N-2}{N^2} \sim \frac{3}{N}$. Al crescere della dimensione del problema, la matrice **A** diventa sempre più sparsa e diventa sempre più conveniente definirla come matrice sparsa, ed usare gli algoritmi pensati per questo tipo di matrici.

```
"""
Linear systems with square non-singular matrices, in sparse format
"""

from scipy import sparse

# Printout level: the higher the number, the more verbose the script
printout_level = 1

n_nodes = 5
i_nodes = list(np.arange(5))

# Build sparse stiffness matrix, I: row indices, J: col indices, E: matrix elems
I = np.array(i_nodes+i_nodes[:-1]+i_nodes[ 1:])
J = np.array(i_nodes+i_nodes[ 1:]+i_nodes[:-1])
E = np.array(n_nodes*[2]+(n_nodes-1)*[-1]+(n_nodes-1)*[-1])

A = sparse.coo_array((E, (I,J))).tocsr()

if ( printout_level > 50 ):  # print matrix in sparse format
    print(f" I: {I}\n J: {J}\n E: {E}")
    print(f" A:\n {A}")

if ( printout_level > 60 ):  # convert and primt matrix in full format
    print(f" A.todense(): {A.todense()}")

# RHS
b = np.array(5*[1])

# Solve linear system
x = sparse.linalg.spsolve(A, b)
```

(continues on next page)

(continua dalla pagina precedente)

```
print(f"Sol, x: {x}")
```

```
Sol, x: [2.5 4.  4.5 4.  2.5]
```

CAPITOLO 9

Equazioni algebriche non lineari

Questa sezione si occupa della soluzione delle equazioni algebriche non lineari, distinguendo le equazioni non lineari con una sola incognita $x \in \mathbb{R}$

$$f(x) = 0 \, ,$$

e i sistemi di equazioni non lineari con un numero di incognite pari al numero di equazione,

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \, .$$

9.1 Equazioni non lineari

Vengono presentati i metodi di bisezione e di Newton per la soluzione di un'equazione non lineare,

$$f(x) = 0 \, ,$$

e applicati alla soluzione del problema con $f(x) = e^x + x$, la cui derivata è nota e immediata da calcolare $f'(x) = e^x + 1$. L'espressione della derivata verrà utilizzata nel metodo di Newton.

```
""" Import libaries """

import numpy as np
from time import time
```

```
"""
Example. f(x) = e^x + x
"""

# Function f and its derivative
f  = lambda x: np.exp(x) + x
df = lambda x: np.exp(x) + 1
```

9.1.1 Metodo di bisezione

Il metodo di bisezione per la ricerca degli zeri di una funzione continua $F(x)$ si basa sul teorema dei valori intermedi per le funzioni continue.

Dati due numeri reali a, b tali che $f(a)\,f(b) < 0$, allora esiste un punto $c \in (a, b)$ tale che $f(c) = 0$.

```
"""
Define bisection_method_scalar() function to solve nonlinear scalar equations with␣
 ↪bisection method
"""

def bisection_method_scalar(f, a, b, tol=1e-6, max_niter=100):
    """ Function implementing the bisection method for scalar equations """

    niter = 0

    if ( not f(a) * f(b) < 0 ):
        print("Bisection algorithm can't start, f(a)f(b)>= 0")
    else:
        x = .5 * (a+b)
        fx = f(x)
        while ( np.abs(fx) > tol and niter < max_niter ):

            if ( f(x) * f(a) <= 0 ):  # new range [a,c]
                b = x
            else:  # new range [a,b]
                a = x

            # Update solution and residual
            x = .5 * (a+b)
            fx = f(x)

            # Update n.iter
            niter += 1

    return x, np.abs(fx), niter, max_niter
```

```
""" Use bisection_method_scalar() function to solve the example """

# Find 2 values so that $f(a) f(b) < 0$
a, b = -2., 0.

t1 = time()
x, res, niter, max_niter = bisection_method_scalar(f, a, b,)

print("Bisection method summary: ")
if ( niter < max_niter ):
    print(f"Convergence reached")
    print(f"Sol, x = {x}")
else:
    print(f"max n.iter reached without convergence")

print(f"residual     : {f(x)}")
print(f"n. iterations: {niter}")
print(f"elapsed time : {time()-t1}")
```

```
Bisection method summary:
Convergence reached
Sol, x = -0.567143440246582
residual     : -2.348157265297246e-07
n. iterations: 20
elapsed time : 0.0006110668182373047
```

9.1.2 Metodo di Newton

Per trovare la soluzione del problema non lineare

$$f(x) = 0\,,$$

il metodo di Newton sfrutta l'espansione in serie troncata al primo grado della funzione $f(x)$, per scrivere

$$0 = f(x^n + \Delta x) \approx f(x^n) + f'(x^n)\Delta x$$

e ottenere l'incremento della soluzione Δx come soluzione del sistema lineare

$$f'(x^n)\Delta x = -f(x^n)$$

e aggiornare la soluzione $x^{n+1} = x^n + \Delta x$.

```
"""
Define newton_method_scalar() function to solve nonlinear scalar equations with Newton
 ↪'s method
"""

def newton_method_scalar(f, df, x=.0, tol=1e-6, max_niter=100):
    """ Function implementing Newton's method for scalar equations """

    res = f(x)
    niter = 0

    # Newton algorithm
    while ( np.abs(res) > tol and niter < max_niter ):
        # Solve linear approximation step, and update solution
        dx = - res / df(x)
        x += dx

        #> Evaluate new residual and n. of iter
        res = f(x)
        niter += 1

    return x, res, niter, max_niter
```

```
""" Use newton_method_scalar() function to solve the example """

# import numpy as np   # already imported

# Parameters of the Newton method, for stopping criteria
# tol = 1e-6          # tolerance on the residual |f(x)| < tol
# max_niter = 10      # max n. of iterations      niter > max_niter
x0 = -1.
```

(continues on next page)

(continua dalla pagina precedente)

```
t1 = time()
x, res, niter, max_niter = newton_method_scalar(f, df, x=x0)

print("Newton's method summary: ")
if ( niter < max_niter ):
    print(f"Convergence reached")
    print(f"Sol, x = {x}")
else:
    print(f"max n.iter reached without convergence")

print(f"residual     : {f(x)}")
print(f"n. iterations: {niter}")
print(f"elapsed time : {time()-t1}")
```

```
Newton's method summary:
Convergence reached
Sol, x = -0.567143285989123
residual     : 6.927808993140161e-09
n. iterations: 3
elapsed time : 0.00047278404235839844
```

9.2 Sistemi di equazioni non lineari

9.2.1 Metodo di Newton

Il metodo di Newton sfrutta l'espansione lineare della funzione $\mathbf{f}(\mathbf{x})$ nell'intorno di un valore $\mathbf{x}$,

$$\mathbf{0} = \mathbf{f}(\mathbf{x} + \mathbf{h}) \simeq \mathbf{f}(\mathbf{x}) + \mathbf{f}'(\mathbf{x})\,\mathbf{h}$$

per costruire un metodo iterativo composto da due passi a ogni iterazione:

- ricerca dell'incremento:

$$\mathbf{f}'(\mathbf{x}^n)\,\mathbf{h}^{(n)} = -\mathbf{f}(\mathbf{x}^{(n)})$$

- aggiornamento della soluzione

$$\mathbf{x}^{(n+1)} = \mathbf{x}^{(n)} + \mathbf{h}^{(n)}\ .$$

Esempio. Il metodo di Newton viene applicato al sistema non lineare

$$\begin{cases} x_0 - x_1 = 0 \\ -x_0^2 + x_1 = -1 \end{cases}$$

che può essere scritto con il formalismo matriciale come

$$\mathbf{0} = \mathbf{f}(\mathbf{x}) = \begin{bmatrix} x_0 - x_1 \\ -x_0^2 + x_1 - 1 \end{bmatrix}\ .$$

La derivata della funzione $\mathbf{f}(\mathbf{x})$, rispetto alla variabile indipendente $\mathbf{x}$,

$$\mathbf{f}'(\mathbf{x}) = \begin{bmatrix} 1 & -1 \\ -2x_0 & 1 \end{bmatrix}$$

può essere rappresentata come una matrice che ha come elemento alla riga i e alla colonna j la derivata della funzione $f_i(\mathbf{x})$ rispetto alla variabile x_j, $[\mathbf{f}']_{ij} = \frac{\partial f_i}{\partial x_j}$, così che l'approssimazione al primo ordine dell'incremento della funzione può essere scritto come

$$\mathbf{f}(\mathbf{x}+\mathbf{h}) - \mathbf{f}(\mathbf{x}) = \mathbf{f}'(\mathbf{x})\,\mathbf{h} + o(||\mathbf{h}||) \,.$$

```
"""
Example. f(x) = np.array([  x[0]    - x[1]
                          -x[0]**2 + x[1] - 1 ])
"""

# Function f and its derivative
f  = lambda x: np.array([ x[0] - x[1], -x[0]**2 + x[1] + 1])
df = lambda x: np.array([[1, -1], [-2*x[0], 1]])
```

```
"""
Define newton_method_scalar() function to solve nonlinear systems of equations with␣
 ↪Newton's method
"""

def newton_method_system(f, df, x=.0, tol=1e-6, max_niter=100):
    """ Function implementing Newton's method for systems of equations """

    res = f(x)
    niter = 0

    # Newton algorithm
    while ( np.linalg.norm(res) > tol and niter < max_niter ):
        # Solve linear approximation step, and update solution
        dx = - np.linalg.solve(df(x), res)
        x += dx

        #> Evaluate new residual and n. of iter
        res = f(x)
        niter += 1

    return x, res, niter, max_niter
```

```
""" Use newton_method_scalar() function to solve the example """

# import numpy as np   # already imported

# Parameters of the Newton method, for stopping criteria
# tol = 1e-6          # tolerance on the residual |f(x)| < tol
# max_niter = 10      # max n. of iterations      niter > max_niter
x0 = np.array([ -1. , 1 ])

t1 = time()
x, res, niter, max_niter = newton_method_system(f, df, x=x0)

print("Newton's method summary: ")
if ( niter < max_niter ):
    print(f"Convergence reached")
    print(f"Sol, x = {x}")
else:
    print(f"max n.iter reached without convergence")
```

(continues on next page)

(continua dalla pagina precedente)

```
print(f"residual     : {f(x)}")
print(f"n. iterations: {niter}")
print(f"elapsed time : {time()-t1}")
```

```
Newton's method summary:
Convergence reached
Sol, x = [-0.61803399 -0.61803399]
residual     : [ 0.00000000e+00 -2.10942375e-13]
n. iterations: 4
elapsed time : 0.0007443428039550781
```

- **todo** L'algoritmo di Newton trova solo una soluzione del problema. Cercare le altre soluzioni cambiando il tentativo iniziale.
- **todo** … altro?

CAPITOLO 10

Approssimazione di funzioni

10.1 Interpolazione

10.2 Regressione

CAPITOLO 11

Derivate di funzioni

11.1 Differenze finite

Il calcolo della derivata di una funzione $f(x)$ derivabile in un punto x_0 può essere svolto utilizzando l'espansione locale in serie di Taylor di una funzione.

11.1.1 Derivata prima

Usando le espansioni

$$f(x+h) = f(x) + f'(x)\,h + f''(x)\frac{h^2}{2} + f'''(x)\frac{h^3}{3!} + o(h^3)$$
$$f(x-h) = f(x) - f'(x)\,h + f''(x)\frac{h^2}{2} - f'''(x)\frac{h^3}{3!} + o(h^3)$$

si possono ricavare:

- gli schemi del **primo ordine**

$$f'(x) = \frac{f(x+h) - f(x)}{h} + O(h)$$
$$f'(x) = \frac{f(x) - f(x-h)}{h} + O(h)$$

- lo schema **centrato del secondo ordine**

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2)$$

- lo schema non centrato del secondo ordine:

$$f'(x) = \frac{-3\,f(x) + 4\,f(x+h) - f(x+2h)}{2\,h} + O(h^2)\,.$$

Dimostrazione

Usando le espansioni in serie,

$$f(x+h) = f(x) + f'(x)\,h + f''(x)\frac{h^2}{2} + f'''(x)\frac{h^3}{3!} + o(h^3)$$
$$f(x+2h) = f(x) + f'(x)\,2h + 2f''(x)\,h^2 + f'''(x)\frac{4}{3}h^3 + o(h^3)$$

si cerca una coppia di coefficienti della combinazione lineare $a_1 f(x+h) + a_2 f(x+2h)$ che annullano il termine di secondo grado.

In particolare è facile dimostrare che una di queste scelte è $\alpha_1 = 4$, $\alpha_2 = -1$, e la combinazione lineare per questi valori diventa,

$$4f(x+h) - f(x+2h) = 3f(x) + 2\,hf'(x) + O(h^3)\,.$$

A questo punto è semplice isolare $f'(x)$ per trovare lo schema numerico desiderato,

$$f'(x) = \frac{-3\,f(x) + 4\,f(x+h) - f(x+2h)}{2\,h} + O(h^2)\,.$$

- …
- schemi di ordine superiore…

11.1.2 Derivata seconda

Usando le stesse espansioni in serie, si può ottenere uno schema del secondo ordine per la derivata seconda

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + O(h^2)$$

Dimostrazione

Usando le espansioni in serie

$$f(x+h) = f(x) + f'(x)\,h + f''(x)\frac{h^2}{2} + f'''(x)\frac{h^3}{3!} + f^{iv}(x)\frac{h^4}{4!} + O(h^5)$$
$$f(x-h) = f(x) - f'(x)\,h + f''(x)\frac{h^2}{2} - f'''(x)\frac{h^3}{3!} + f^{iv}(x)\frac{h^4}{4!} + O(h^5)$$

si può notare che nella somma che compare al numeratore dello schema numerico si annullano sia il termine di primo grado sia il termine di terzo grado,

$$f(x+h) - 2f(x) + f(x-h) = f''(x)\,h^2 + O(h^4)\,,$$

e quindi lo schema numerico proposto è del secondo ordine,

$$f'(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + O(h^2)\,.$$

```
""" Numerical schemes for derivatives """

df_first_order_left    = lambda f, x, h: (f(x) - f(x-h)) / h
df_first_order_right   = lambda f, x, h: (f(x+h) - f(x)) / h
df_second_order_center = lambda f, x, h: (f(x+h) - f(x-h)) / ( 2. * h )
df_second_order_left   = lambda f, x, h: (  3*f(x) - 4*f(x-h) + f(x-2*h)) / ( 2. * h )
df_second_order_right  = lambda f, x, h: (- 3*f(x) + 4*f(x+h) - f(x+2*h)) / ( 2. * h )

df_fun_dict = {
    '1_left'  : df_first_order_left,
    '1_right' : df_first_order_right,
    '2_center': df_second_order_center,
    '2_left'  : df_second_order_left,
    '2_right' : df_second_order_right
}
```

```
""" Example """
import numpy as np

f = lambda x: 2. * np.exp(- x**2)

x, h = 1., .01

for df_label, df_fun in df_fun_dict.items():
    print(f"Scheme: {df_label.ljust(10)}, value: {df_fun(f,x,h)}" )
```

```
Scheme: 1_left    , value: -1.4788256893130014
Scheme: 1_right   , value: -1.4641117378933477
Scheme: 2_center  , value: -1.4714687136031745
Scheme: 2_left    , value: -1.4716195509633268
Scheme: 2_right   , value: -1.4716121945026028
```

CAPITOLO 12

Integrali

```
""" Import libraries """

import numpy as np
```

12.1 Integrazione di Newton-Cotes

Valutazioni di integrali definiti di funzioni $f : [a, b] \in \mathbb{R} \to \mathbb{R}$, continue sull'intervallo. Si veda l'*esempio* per intuire la necessità della continuità della funzione.

- Formula del punto medio
- Formula del trapezio

12.1.1 Metodo di integrazione del punto medio

Il metodo di integrazione del punto medio ricorda la definizione dell'integrale di Riemann (**todo** aggiungere link). Data la funzione $f : [a, b] \in \mathbb{R} \to \mathbb{R}$, e una partizione $P = \{a = x_0 < x_1 < \cdots < x_n = b\}$ dell'intervallo $[a, b]$, l'integrale viene calcolato come la somma dell'area dei rettangoli elementari di lati $\Delta x_k := x_k - x_{k-1}$ e $f(\xi_k)$, con $\xi_k \in [x_{k-1}, x_k]$,

$$\int_{x=a}^{b} f(x)dx \simeq \sum_{k=1:n} f(\xi_k)\, \Delta x_k \, .$$

```
""" Mid-point method """

def integral_rect(f, a, b, n):
    """
    Inputs:
    - f: function
    - a, b: extreme points of the range
```

(continues on next page)

(continua dalla pagina precedente)

```
    - n: n. of intervals of the range
    """
    # Partition of the range [a,b]
    # uniform partition here; refined algorithms may use non-uniform partitions
    xv = a + ( b - a ) * np.arange(n+1) / n;  xv[-1] = b
    xc = .5 * ( xv[1:] + xv[:-1] )
    dx = xv[1:] - xv[:-1]

    return np.sum( dx * f(xc) )
```

12.1.2 Metodo di integrazione del trapezio

Data la funzione $f : [a,b] \in \mathbb{R} \to \mathbb{R}$, e una partizione $P = \{a = x_0 < x_1 < \cdots < x_n = b\}$ dell'intervallo $[a,b]$, l'integrale viene calcolato come la somma dell'area dei trapezi rettangoli elementari con basi $f(x_{k-1})$, $f(x_k)$ e altezza $\Delta x_k := x_k - x_{k-1}$,

$$\int_{x=a}^{b} f(x)dx \simeq \sum_{k=1:n} \frac{1}{2} \left(f(x_{k-1}) + f(x_k)\right) \Delta x_k \, .$$

```
""" Trapezoid method """

def integral_trapz(f, a, b, n):
    """
    Inputs:
    - f: function
    - a, b: extreme points of the range
    - n: n. of intervals of the range
    """
    # Partition of the range [a,b]
    # uniform partition here; refined algorithms may use non-uniform partitions
    xv = a + ( b - a ) * np.arange(n+1) / n;  xv[-1] = b
    dx = xv[1:] - xv[:-1]

    return np.sum( .5 * dx * ( f(xv[1:]) + f(xv[:-1]) ) )
```

12.1.3 Esempi

Esempio 1.

Si valuta l'integrale

$$\int_{x=0}^{1} x^2 \, dx = \frac{1}{3} \, ,$$

con i metodi del punto medio e del trapezio. La funzione $f(x) = x^2$ è continua ovunque e quindi è continua nell'intervallo $[0,1]$.

```
f = lambda x: x**2

a, b = 0., 1.
n = 10

# Evaluate integral with the rect and trapz methods
val_rect  = integral_rect(f, a, b, n)
val_trapz = integral_trapz(f, a, b, n)

print(f"Value of the integral, x \in [{a}, {b}] with {n} intervals")
print(f"- mid-point method: {val_rect}")
print(f"- trapezoid method: {val_trapz}")
```

```
Value of the integral, x \in [0.0, 1.0] with 10 intervals
- mid-point method: 0.3325
- trapezoid method: 0.33499999999999996
```

Esempio 2. Necessità della continuità della funzione.

12.2 Integrazione di Gauss

L'integrazione di Gauss permette di calcolare in **maniera esatta** l'integrale di una **funzione polinomiale** $p_n(x)$ su un intervallo $[a, b]$, come somma pesata della funzione valuatata in alcuni punti dell'intervallo,

$$\int_a^b p^{(n)}(x)dx = \sum_g w_g f(x_g) \,,$$

per un numero di nodi di Gauss, n_{Gauss} che soddisfa la relazione

$$n < 2n_{Gauss} - 1 \,.$$

Per motivi di generalizzazione dell'algoritmo, nella definizione dei **pesi** w_i e dei **nodi di Gauss** x_i, l'integrale viene riportato all'integrale su un intervallo di riferimento, tramite una trasformazione di coordinate. Per domini 1D, l'intervallo di riferimento per la quadratura di Gauss è l'intervallo $\xi = [-1, 1]$ e il cambio di variabili è

$$x = \frac{a+b}{2} + \frac{b-a}{2}\,\xi \,,$$

così che l'integrale originale può essere scritto come

$$\int_{x=a}^{b} p^{(n)}(x)dx = \int_{\xi=-1}^{1} p^(n)(x(\xi))\frac{dx}{d\xi}d\xi = \\ = \frac{b-a}{2}\int_{\xi=-1}^{1} p^{(n)}(x(\xi))d\xi = \frac{b-a}{2}\sum_g w_g\, p^{(n)}\left(x(\xi_g)\right)$$

```
""" Gauss integration """

# Dict of Gauss weights and nodes on the reference interval [-1,1] for 1D integration
gauss_nw = { 1: {'nodes'   : [ 0. ],
                 'weights': [ 2. ]},
             2: {'nodes'   : [ -1./np.sqrt(3.), 1./np.sqrt(3.) ],
```

(continues on next page)

(continua dalla pagina precedente)

```
                'weights': [ 1., 1.]},
            3: {'nodes'  : [ 0., -np.sqrt(3./5.), np.sqrt(3./5.) ],
                'weights': [ 8./9., 5./9., 5./9. ]},
}

def gauss_int_1(f, n, a=-1, b=1, gauss_nw=gauss_nw):
    """
    Integral of the function f(x)
    over a physical domain, x \in [a, b],
    using n number of Gauss nodes
    """
    # Cap n.nodes to the max n.nodes stored in gauss_nw dict
    n = np.min([n, np.max(list(gauss_nw.keys()))])

    # Gauss nodes (from reference to actual domain) and weights
    xg = .5 * ( a + b + ( b - a ) * np.array(gauss_nw[n]['nodes']) )
    wg = np.array(gauss_nw[n]['weights'])

    return .5 * (b-a) * np.sum( wg * f(xg) )


def gauss_int_n(f, n_gauss, a, b, n_elems):
    """
    Integral of the function f(x)
    over a physical domain, x \in [a, b], (uniformly) splitted in n_elems
    using n number of Gauss nodes per each elem
    """
    # Partition of the interval [a,b]
    xv = a + ( b - a ) * np.arange(n_elems+1) / n_elems;  xv[-1] = b

    return np.sum([ gauss_int_1(f, n_gauss, xv[i], xv[i+1]) for i in np.arange(n_
↪elems) ])
```

12.2.1 Esempio 1 - integrazione di Gauss.

Si valuta l'integrale

$$\int_{x=0}^{1} x^2 \, dx = \frac{1}{3} \, ,$$

con il metodo di integrazione di Gauss. Si chiede di:

- osservare che l'integrazione è esatta, a meno degli arrotondamenti dovuti all'aritmetica finita, poiché la funzione integranda è di grado 2, e il numero di nodi di Gauss è $n_{Gauss} = 3$; questo è vero indipendentemente dal numero di sotto-intervalli;
- cambiare le funzioni e il numero di nodi usati nelle funzioni per l'integrazione di Gauss per verificare che l'integrazione è esatta per funzioni polinomiali di grado $2n_{Gauss} - 1$.

```
""" Evaluate integrals using Gauss integration """

f = lambda x: x**2
a, b = 0., 1.

# Gauss nodes, and n.of elements
```

(continues on next page)

(continua dalla pagina precedente)

```
n_gauss = 3
n_elems = 10

val_1 = gauss_int_1(f, n_gauss, 0., 1.)
val_n = gauss_int_n(f, n_gauss, 0., 1., n_elems)

print(f"Value of the integral, x \in [{a}, {b}] using Gauss integration methods with
 ↪{n_gauss} nodes per elem")
print(f"- using one elem: {val_1}")
print(f"- using {n_elems} elems: {val_n}")
```

```
Value of the integral, x \in [0.0, 1.0] using Gauss integration methods with 3␣
 ↪nodes per elem
- using one elem: 0.33333333333333337
- using 10 elems: 0.33333333333333337
```

CAPITOLO 13

Equazioni differenziali ordinarie

13.1 Problemi di Cauchy ai valori iniziali

13.1.1 Approccio a un problema di Cauchy di ordine n

Un problema di Cauchy di ordine n

$$\begin{cases} F(y^{(n)}(x), y^{(n-1)}(x), \dots, y'(x), y(x), x) = 0 \\ y(x_0) = y^0 \\ y'(x_0) = y'^0 \\ \dots \\ y^{(n-1)}(x_0) = y^{(n-1),0} \end{cases}$$

con funzione incognita $y(x) : D \in \mathbb{R} \to \mathbb{R}$, può essere riscritto come un problema di *«ordine 1»* per la funzione incognita $\mathbf{z}(x) : D \in \mathbb{R} \to \mathbb{R}^n$, definita come

$$\mathbf{z}(x) = (z_0(x), z_1(x), \dots z_{n-1}(x)) := (y(x), y'(x), \dots, y^{(n-1)}(x)) \, .$$

Esplicitando le relazioni tra le componenti di $\mathbf{z}(x)$ e le derivate della funzione $y(x)$, $z_k(x) = y^{(k)}(x) = y^{(k-1)'}(x) = z'_{k-1}(x)$, il problema di Cauchy può essere riformulato come

$$\begin{cases} z'_0 - z_1 = 0 \\ z'_1 - z_2 = 0 \\ \dots \\ z'_{n-2} - z_{n-1} = 0 \\ F(z'_{n-1}(x), z_{n-1}(x), \dots, z_1(x), z_0(x)) = 0 \, , \end{cases} \quad , \quad \text{i.c.} \quad \begin{cases} z_0(x_0) = y^0 \\ z_1(x_0) = y'^0 \\ \dots \\ z_{n-1}(x_0) = y^{(n-1),0} \end{cases}$$

che può essere riscritto con il formalismo vettoriale come

$$\begin{cases} \mathbf{F}(\mathbf{z}'(x), \mathbf{z}) = \mathbf{0} \\ \mathbf{z}(x_0) = \mathbf{z}_0 \end{cases}$$

13.1.2 Caratteristiche (cenni)

- accuratezza, consistenza, convergenza
- stabilità: 0-, A- condizionata e incondizionata

13.1.3 Schemi numerici

Schemi numerici a un passo

Schemi numerici multi-step

13.2 Problemi al contorno

13.2.1 Differenze finite

13.2.2 Elementi finiti

13.2.3 Volumi finiti

CAPITOLO 14

Ottimizzazione

Le tecniche di ottimizzazione sono alla base di molti metodi di interesse, dall'approssimazione di funzioni, alla regolazione e controllo, agli algoritmi usati in intelligenza artificiale

Parte III

Statistica

CAPITOLO 15

Metodi per la statistica

La statistica si occupa dello studio dei fenomeni in condizioni di incertezza o di non determinismo, determinate da una conoscenza incompleta del modello.

I contenuti seguono l'ordine logico di un approccio alla statistica: dopo aver introdotto la statistica descrittiva e la rappresnetazione dei dati, vengono fornite delle basi di teoria della probabilità necessarie alla deduzione e alle conclusioni tipiche di un'analisi statistica inferenziale.

La statistica è applicabile in **diversi (tutti?) gli ambiti**: dalla scienze naturali, alle scienze sociali,… Secondo uno studio, i 12 metodi statistici (cos'è un metodo statistico?) più usati sono: ANOVA, χ^2-test, t-test, regressione lineare, coefficiente di correlazione di Pearson, Mann-Whitney U-test, Kruskal-Wallis test, Shannon's diversity index, Turkey's range test, cluster analysis, Spearman's rank correlation coefficient, and PCA.

La statistica è uno strumento fondamentale del **metodo scientifico**, e si occupa dell'intero processo di:

- progetto dell'esperimento
- svolgimento esperimento e raccolta dati
- analisi dati

A seconda dell'uso dei dati raccolti, si riconoscono due approcci alla statistica:

- **statistica descrittiva**: riassume i dati raccolti in statistiche riassuntive, senza voler dedurre conclusioni riguardo alla popolazione della quale è stato analizzato il campione; una **rappresentazione grafica** dei dati e delle statistiche sintetiche risulta molto efficiente per la mente umana

  todo *dati di sintesi, non ci sono ipotesi sulle dimensioni di campione e popolazione; una volta disponibili i dati, quelli sono e quelli si riassumono*

- **statistica inferenziale**: usa i dati raccolti per dedurre proprietà di una popolazione ampia della quale è stato analizzato un campione ristretto. Le deduzioni si possono spesso riassumere in:
  - **stima** di distribuzioni di probabilità o valori
  - **test di ipotesi**

todo *uniformare diverse applicazioni allo stesso metodo statistico, ad esempio come diverse interpretazioni che possono essere date allo stesso metodo*

- stime: regressione lineare e lineare generalizzata, minimi quadrati,…
- …

todo Statistica e ML:

- SL: regressione e classificazione
- UL: clustering, dimension reduction
- ML: otpimization and control

CAPITOLO 16

Statistica descrittiva

La statistica descrittiva si occupa di descrivere e riassumere le informazioni contenute nei dati disponibili, con lo scopo di:

- comunicare il maggior numero di informazioni rilevanti
- nella maniera più semplice e sintetica possibile

Questo, in accordo con la *percezione umana* **todo**, avviene tramite indicazioni di:

- **posizione**, come la media, la mediana, la moda, o loro variazioni sul tema
- **dispersione**, come la deviazione standard, la varianza, lo scarto interquartile,…
- **forma**, come skewness o curtosi, o momenti di ordine superiore
- **correlazione** o **dipendenza todo** *correlazione* $\neq$ *dipendenza, fare un riferimento; è necessario dedurre una dipendenza? E" tra i compiti della statistica descrittiva?*

16.1 Dimensione dei dati

- variabili 1-dimensionali
- variabili 2-dimensionali o di dimenione piccola
- variabili di grandi dimensioni: possono essere necessarie tecniche di riduzione dei dati per far emergere le proprietà significative contenute nei dati raccolti

16.2 Rappresentazione grafica

EDA: exploratory data analysis. Può essere utilizzata in applicazioni di statistica descrittiva o nelle prime fasi di applicazioni di statistica inferenziale, «per vedere che faccia hanno i dati raccolti» e per fare una prima selezione dei metodi da applicare nelle fasi successive dell'indagine.

Attualmente, sono disponibili i mezzi informatici e un gran numero di librerie per l'analisi dati, dall'elaborazione alla rappresentazione.

16.3 Variabili 1-dimensionali

16.4 Variabili 2-dimensionali

16.5 Variabili di grandi dimensioni

()=

CAPITOLO 17

Introduzione alla probabilità

- Variabili casuali
- Processi casuali

17.1 Variabili casuali

Definizione. Una variabile casuale…

17.1.1 Variabili casuali discrete

…

Valore medio

$$\mu_X = E[X] = \sum_k p(x_k) x_k$$

Varianza

$$\sigma_X^2 = E[(X - \mu_X)^2] = \sum_k p(x_k)(x_k - \mu_X)^2$$

media, moda.

Variabili casuali multi-dimensionali

- **Probabilità congiunta.** $p(x, y)$
- **Probabilità condizionale.** $p(x|y)$
- **Probabilità marginale.** $p(x)$

Probabilità congiunta e teorema di Bayes

$$p(x, y) = p(x|y)p(y) = p(y|x)p(x)$$

Variabili indipendenti

Due variabili casuali sono indipendenti se la probabilità condizionale di una variabile coincide con la sua probabilità marginale,

$$p(x|y) = p(x) \ ,$$

così che la probabilità congiunta di due variabili casuali indipendenti è il prodotto delle probabilità marginali,

$$p(x, y) = p(x)p(y) \ .$$

Covarianza

$$\sigma_{ij}^2 = E[(X_i - \mu_i)(X_j - \mu_j)] = R_{ij} - \mu_i \mu_j$$

Correlazione

$$\rho_{XY} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{E[(X - \mu_X)^2]^{1/2} \, E[(X - \mu_X)^2]^{1/2}} = \frac{\sigma_{XY}^2}{\sigma_X \, \sigma_Y}$$

17.1.2 Variabili casuali continue

…

17.1.3 Trasformazione di probabilità di densità

Variabile singola

Cambio di variabile

Traslazione

Scalatura

Multi-variabile

Combinazione di variabili

Somma

Date due variabili casuali X, Y, con probabilità congiunta $p_{XY}(x, y)$, la loro somma $Z = X + Y$ è una variabile casuale dipendente con distribuzione di probabilità

$$p(z) = \int_y p_{XY}(z - y, y)\, dy = \int_x p_{XY}(x, z - x)\, dx \; .$$

Se le due variabili sono tra di loro *statisticamente indipendenti*, la densità di probabilità congiunta è uguale al prodotto delle densità di probabilità delle singole variabili, $p_{XY}(x, y) = p_X(x)p_Y(y)$, e quindi la densità di probabilità della somma è uguale alla **convoluzione** tra le densità di probabilità delle due variabili,

$$p(z) = \int_y p_X(z - y)\, p_Y(y)\, dy \; .$$

Il **valore atteso** della somma è quindi

$$E[Z] = \int_z z\, p(z)\, dz = \int_{y,z} z\, p_{XY}(z - y, y)\, dy\, dz = \int_{x,y} (x + y)\, p_{XY}(x, y)\, dx\, dy \; ,$$

e, nel caso in cui le due variabili siano tra di loro *statisticamente indipendenti*,

$$\begin{aligned} E[Z] &= \int_{x,y} (x + y)\, p_X(x)\, p_Y(y)\, dx\, dy = \\ &= \int_{x,y} x\, p_X(x)\, p_Y(y)\, dx\, dy + \int_{x,y} y\, p_X(x)\, p_Y(y)\, dx\, dy = \\ &= \int_x x\, p_X(x)\, dx + \int_y y\, p_Y(y)\, dy = E[X] + E[Y] \; . \end{aligned}$$

La **varianza** della somma è

$$\sigma_Z^2 = E\left[(Z - E[Z])^2\right] = E\left[Z^2\right] - E[Z]^2 = \int_z z^2 p(z)\, dz - E[Z]^2$$

e nel caso le due variabili siano tra di loro *statisticamente indipendenti*,

$$\begin{aligned} \sigma_Z^2 &= \int_z z^2 p(z)\, dz - E[Z]^2 = \\ &= \int_{x,y} (x + y)^2\, p_X(x)\, p_Y(y)\, dx\, dy - \left(E[X] + E[Y]\right)^2 = \\ &= \int_x x^2\, pX(x)\, dx + \int_y y^2\, p_Y(y)\, dy + 2E[X]E[Y] - (E[X]^2 + 2E[X]E[Y] + E[Y]^2) = \\ &= \int_x x^2\, pX(x)\, dx - E[X]^2 + \int_y y^2\, p_Y(y)\, dy - E[Y]^2 = \\ &= \sigma_X^2 + \sigma_Y^2 \; . \end{aligned}$$

Prodotto

Date due variabili casuali X, Y con probabilità congiunta $p_{XY}(x, y)$, il loro prodotto $Z = X \cdot Y$ è una variabile casuale dipendente con distribuzione di probabilità

$$p(z) = \int_y p_{XY}\left(\frac{z}{y}, y\right)\, dy = \int_x p_{XY}\left(x, \frac{z}{x}\right)\, dx$$

Se le due variabili sono *statisticamente indipendenti*,…**todo**

Il **valore atteso** del prodotto è

$$E[Z] = \int_z z\, p(z)\, dz = \int_{y,z} z\, p_{XY}\left(\frac{z}{y}, y\right)\, dy\, dz = \int_{x,y} xy\, p_{XY}(x, y)\, dx\, dy\ ,$$

e, nel caso in cui le due variabili siano tra di loro *statisticamente indipendenti*,

$$\begin{aligned} E[Z] &= \int_{x,y} xy\, p_X(x)\, p_Y(y)\, dx\, dy = \\ &= \int_x x\, p_X(x)\, dx \cdot \int_y y\, p_Y(y)\, dy = E[X] \cdot E[Y]\ . \end{aligned}$$

La **varianza** del prodotto è

$$\sigma_Z^2 = ...$$

e nel caso le due variabili siano tra di loro *statisticamente indipendenti*,

$$\begin{aligned} \sigma_Z^2 &= \int_{x,y} x^2 y^2 p_X(x)\, p_Y(y)\, dx\, dy - \left(E[X]\, E[Y]\right)^2 = \\ &= R_X^2\, R_Y^2 - E[X]^2\, E[Y]^2 = \\ &= \left(\sigma_X^2 + E[X]^2\right)\left(\sigma_Y^2 + E[Y]^2\right) - E[X]^2\, E[Y]^2 = \\ &= \sigma_X^2\, \sigma_Y^2 + \sigma_X^2\, E[Y]^2 + \sigma_Y^2\, E[X]^2 \end{aligned}$$

In termini di correlazione, $R_X^2 = \sigma_X^2 + E[X]^2$,

$$R_Z^2 = R_X^2\, R_Y^2\ .$$

17.1.4 Esempi di funzioni densità di probabilità

Variabili casuali discrete

Variabili casuali continue

Distribuzione gaussiana o normale, $\mathcal{N}$

$\mathcal{N}(\mu, \sigma^2)$

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \ \sim\ e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Distribuzione chi-quadrato, χ^2

Date n variabili casuali X_k indipendenti con distribuzione normale $\mathcal{N}(0,1)$, la somma dei loro quadrati,

$$\chi^2_n = \sum_{k=1}^{n} X_k^2 \,,$$

è una variabile casuale con densità di probabilità χ^2_n, con n definito come il numero di gradi di libertà. La distribuzione χ^2_n ha una pdf

$$f(x) = \frac{1}{2^{\frac{n}{2}}\Gamma\left(\frac{n}{2}\right)} x^{\frac{n}{2}-1} e^{-\frac{x}{2}} \;\sim\; x^{\frac{n}{2}-1} e^{-\frac{n}{2}}$$

Distribuzione t-Student

La t di Student è la distribuzione di probabilità che governa il rapporto tra due variabili casuali indipendenti,

$$t_\nu = \frac{Z}{\sqrt{\frac{K}{\nu}}} \,,$$

con il numeratore con distribuzione normale, $Z \sim \mathcal{N}(0,1)$, e il denominatore con distribuzione chi quadrato, $K \sim \chi^2_n$. La pdf è

$$f(x) = \frac{\Gamma\left(\frac{n+1}{2}\right)}{\sqrt{n\,\pi}\,\Gamma\left(\frac{n}{2}\right)} \left(1 + \frac{x^2}{n}\right)^{-\frac{n+1}{2}} \;\sim\; \left(1 + \frac{x^2}{n}\right)^{-\frac{n+1}{2}} \,.$$

Distrubuzione gaussiana come limite della distribuzione t-Student

Ricordandosi la definizione di e^x come limite della successione $\lim_{n\to\infty}\left(1+\frac{x}{n}\right)^n =: e^x$, e usando l'approssimazione asintotica per $n \to \infty$ della funzione Gamma $\Gamma(n+\alpha) \sim \Gamma(n) n^\alpha$, si può dimostrare che

$$\left(1 + \frac{x^2}{n}\right)^{-\frac{n+1}{2}} = \left(1 + \frac{x^2}{n}\right)^{-\frac{1}{2}} \left[\left(1 + \frac{x^2}{n}\right)^{n}\right]^{-\frac{1}{2}} \quad \to \quad 1 \cdot e^{-\frac{x^2}{2}}$$

$$\frac{\Gamma\left(\frac{n+1}{2}\right)}{\sqrt{n}\,\Gamma\left(\frac{n}{2}\right)} \sim \frac{\Gamma\left(\frac{n}{2}\right)\left(\frac{n}{2}\right)^{\frac{1}{2}}}{\sqrt{n}\,\Gamma\left(\frac{n}{2}\right)} = \frac{1}{\sqrt{2}} \,,$$

e quindi la funzione t-Student tende alla distribuzione normale con valore atteso 0 e varianza unitaria,

$$f(x) \sim \frac{1}{\sqrt{2\,\pi}} e^{-\frac{x^2}{2}} \,.$$

17.1.5 Variabili casuali indipendenti e identicamente distribuite, iid.

Due variabili stochastiche X, Y sono

- statisticamente indipendenti se la loro probabilità congiunta è uguale al prodotto delle probabilità marginali,

$$p_{XY}(x,y) = p_X(x)\, p_Y(y) \,,$$

- identicamente distribuite se hanno la stessa densità di probabilità,

$$p_X(x) = p_Y(x)$$

Teoremi sulle variabili casuali i.i.d.

Dato un insieme di N variabili casuali iid $\{X_n\}_{n=1:N}$, con valore atteso $E[X_n] = \mu$ e varianza $E[(X_n - \mu)^2] = \sigma$, allora la sua media campionaria

$$\bar{X}_N = \frac{1}{N}\sum_{n=1}^{N} X_n$$

- per il **teorema dei grandi numeri**, converge al valore atteso μ della distribuzione di probabilità.
- per il **teorema del limite centrale**, è una variabile casuale che converge in distribuzione a una variabile casuale gaussiana con valore atteso μ e varianza $\frac{\sigma^2}{n}$,

$$\bar{X}_N \to \mathcal{N}\left(\mu, \frac{\sigma^2}{N}\right) \qquad , \qquad \text{as } N \to \infty$$

Convergenza in statistica, todo

- **quasi certamente**. Esempio: teorema dei grandi numeri in forma forte. Il limite della media campionaria è diverso da una variabile casuale μ (**todo** *controllare*!) solo nel caso di eventi di probabilità nulla,

$$P\left(\lim_{N\to\infty} \bar{X}_N = \mu\right) = 1 \, .$$

- **in probabilità**. Esempio: teorema dei grandi numeri in forma debole. Per ogni valore di $\varepsilon > 0$,

$$P\left(\left|\lim_{N\to\infty} \bar{X}_N - \mu\right| < \varepsilon\right) = 1 \, .$$

- **convergenza in distribuzione**. Esempio: teorema del limite centrale…

Esempio: teoremi dei grandi numeri e del limite centrale

Librerie e funzioni

Dimensione campione

Definizione delle variabili casuali

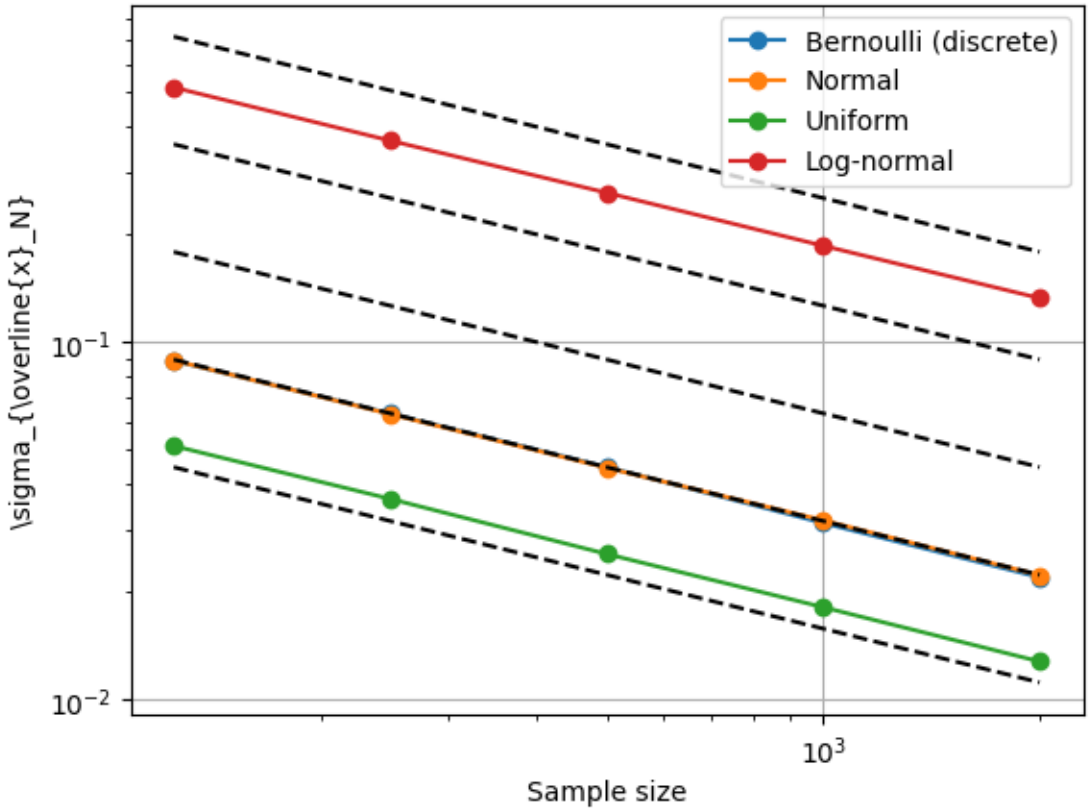
Stima della distribuzione di probabilità della media campionaria

Per tutte le popolazioni definite sopra, vengono

- estratti N_s campioni,
- valuata la media campionaria $\{\overline{X}_s\}_{s=1:N_s}$,
- stimata la densità di probabilità $f_{\overline{X}}(x)$,
- calcolate le statistiche: media $\mu_{\overline{x}}$, e variazione standard $\sigma^2_{\overline{x}}$

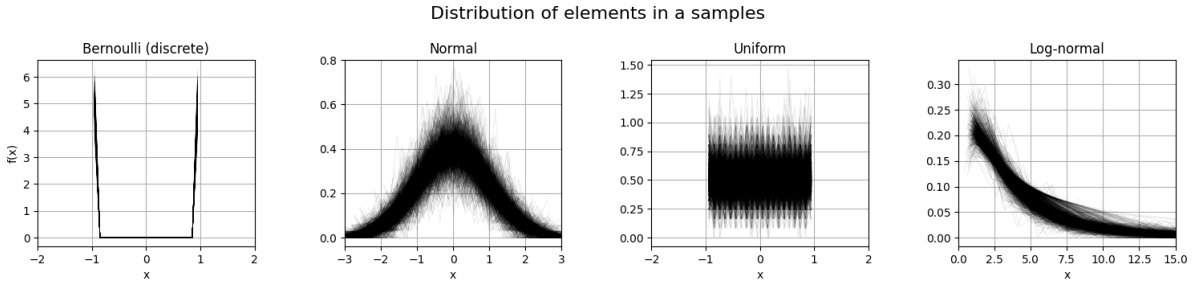
Grafici

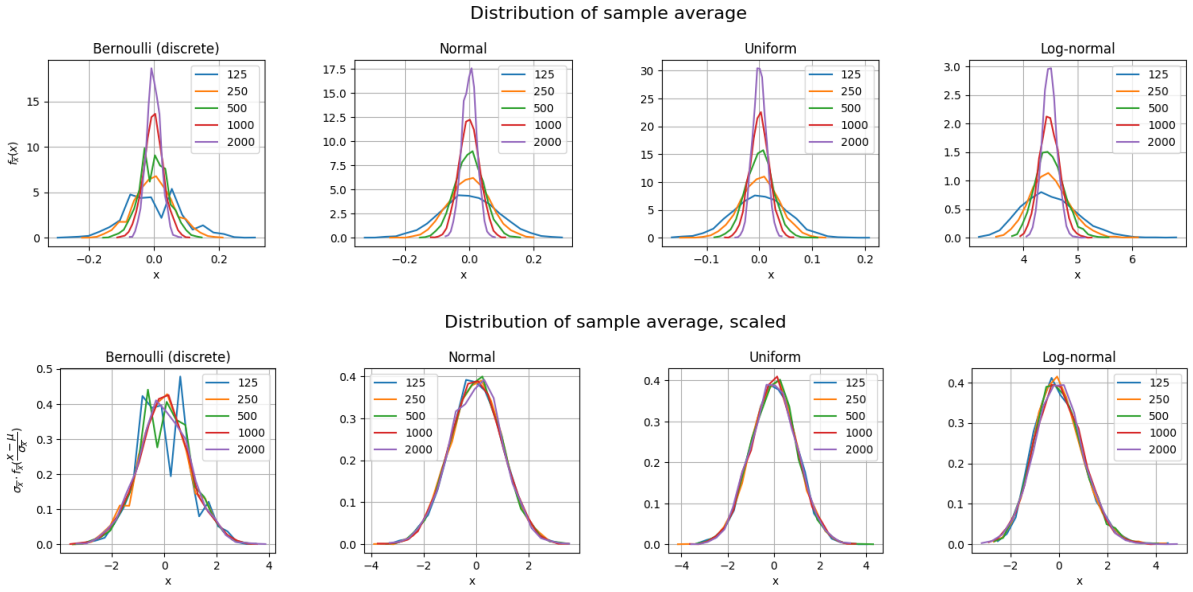
Deviazione standard della media campionaria



Distribuzione di probabilità

```
Text(0.5, 0.98, 'Distribution of sample average, scaled')
```





17.1.6 Campionamento

17.1.7 Media campionaria

La media campionaria di N campioni indipendenti identicametne distribuiti,

$$\bar{X}_N = \frac{1}{N}\sum_{n=1}^{N} X_n \,,$$

è una variabile casuale, che può essere usata come **stimatore senza bias della media** della popolazione. Uno **stimatore senza bias della varianza** è definito

$$\hat{\sigma}_N^2 = \frac{1}{N-1}\sum_{n=1}^{N}(X_n - \bar{X}_N)^2 \,.$$

17.1.8 Dimensione del campione, teorema del limite centrale e distribuzione t-Student

La media di N variabili iid con media μ e varianza σ^2 è una variabile casuale, la cui distribuzione di probabilità tende alla distribuzione normale $\mathcal{N}\left(\mu, \frac{\sigma^2}{N}\right)$ per $N \to \infty$.

Per un numero di campioni ridotti, nel caso la popolazione sia formata da variabili iid con distribuzione gaussiana, la distribuzione della variabile

$$T_N = \frac{\bar{X} - \hat{X}}{\frac{\hat{\sigma}}{\sqrt{N}}} \,,$$

costruita con gli stimatori non-biased della media e della varianza, è una variabile casuale con distribuzione t**-Student** con $N-1$ gradi di libertà. Al tendere di $N \to \infty$, gli stimatori senza bias tendono ai valori veri delle statistiche della popolazione, e la variabile

$$\frac{\bar{X} - \hat{X}}{\frac{\hat{\sigma}}{\sqrt{N}}} \sim \frac{\bar{X} - \mu}{\frac{\sigma}{\sqrt{N}}} \,,$$

tende alla distribuzione normale $\mathcal{N}(0, 1)$

```
#

import numpy as np
import scipy as sp
import matplotlib.pyplot as plt


def sample_to_pdf(x, bins=100, density=True):
  """ evaluate pdf, from histogram, and normalizing with uniform integration rule """
  hist, bin_edges = np.histogram(x, bins=bins, density=True)    # get data from␣
 ↪histogram
  bin_cen = 0.5 * ( bin_edges[:-1] + bin_edges[1:] )            # bin center
  d_bin = bin_edges[1:] - bin_edges[:-1]                        # bin widt
  pdf = hist / np.sum(hist * d_bin)                            # normalization to get␣
 ↪int pdf = 1
  return pdf, bin_cen
```

```
#> Population size
pop_size = 1000000

#> Random number generators
pop_distribution = 'normal'

if ( pop_distribution == 'uniform'):
    rng = np.random.default_rng().uniform   # Uniform distribution
    rng_params = { 'low': -1. , 'high': 1., 'size': pop_size }
else: # 'normal by default'
    rng = np.random.default_rng().normal    # Normal distribution
    rng_params = { 'loc': 0. , 'scale': 1., 'size': pop_size }

#> Generate population
pop = rng(**rng_params)
```

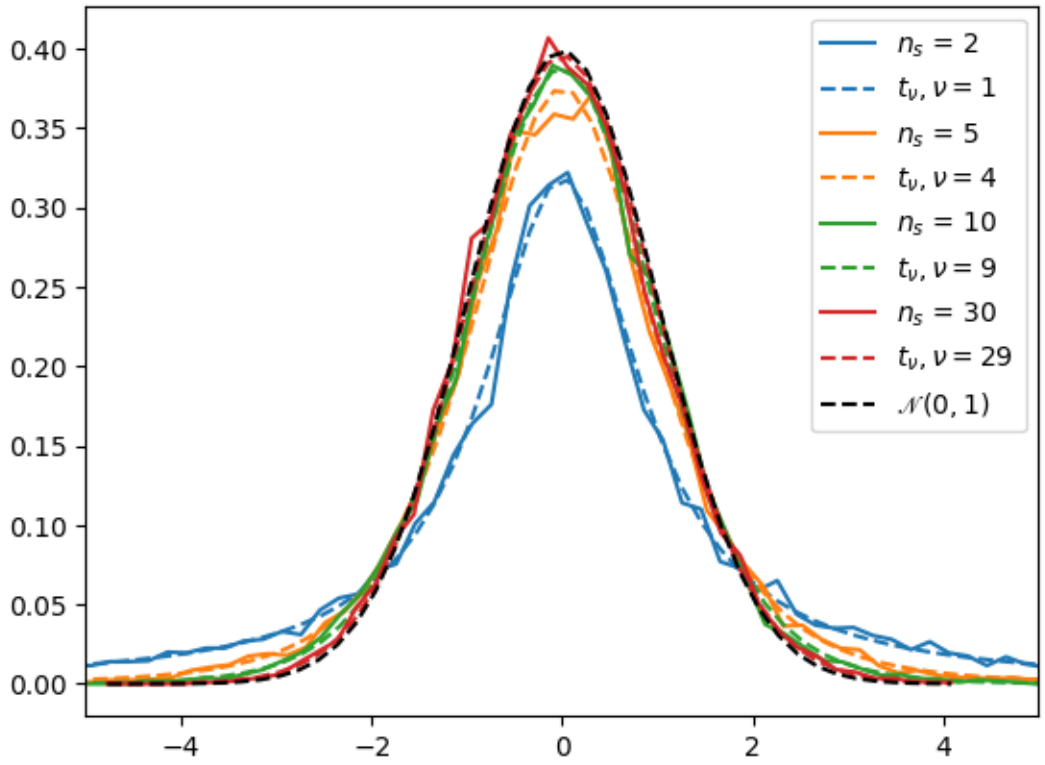
```
sample_size = [ 2, 5, 10, 30 ]
n_samples   = 10000

sample_avgs = []
sample_vars = []
for s_s in sample_size:
    sample_avg = []
    sample_var = []
    for i_s in np.arange(n_samples):
        sample_avg += [ np.mean(pop[i_s*s_s:(i_s+1)*s_s]) ]
        sample_var += [ np.var(pop[i_s*s_s:(i_s+1)*s_s], ddof=1) ]

    sample_avgs += [ sample_avg ]
    sample_vars += [ sample_var ]

# print(sample_avgs)

plt.figure()
for is_s in np.arange(len(sample_size)):
    ns = sample_size[is_s]
    var = np.array( sample_vars[is_s] ) / ns
```

(continues on next page)

(continua dalla pagina precedente)

```
    scaled = sample_avgs[is_s]/var**.5
    bin_width = .2; n_bins = int((np.max(scaled)-np.min(scaled))/bin_width)
    f, x = sample_to_pdf(scaled, bins=n_bins)
    plt.plot(x,f, color=plt.cm.tab10(is_s), label=f"$n_s$ = {ns}")
    plt.plot(x, sp.stats.t.pdf(x, ns-1, loc=0, scale=1), '--', color=plt.cm.tab10(is_
↪s), label=f"$t_\\nu, \\nu ={ns-1}$")
plt.plot(x, sp.stats.norm.pdf(x, loc=0, scale=1), '--', color='black', label="$\
↪mathscr{N}(0,1)$")
plt.legend()
plt.xlim(-5,5)
```

```
(-5.0, 5.0)
```



17.2 Processi casuali

Un processo casuale può essere definito come una variabile casuale che dipende dalla variabile tempo t, $X(t)$.

todo *è probabile che i dettagli di questa sezione richiedano una conoscenza abbastanza approfondita della materia, e di alcuni strumenti matematici come le trasformate*

todo

- definizione
- statistiche (in tempo e in frequenza) e definizioni (stazionarietà, ergodicità,…)
- esempi:

- discreti: Markov
- continui (e discretizzati al computer):
  - white noise, Wiener
  - random walk, Brown
  - applicazioni:
    - diffusione
    - white noise nei sistemi dinamici (risposte a forzanti stocastiche)

CAPITOLO 18

Statistica inferenziale

18.1 Stima

18.2 Test di verifica delle ipotesi

Popper. Filosofia della scienza e principio di falsificabilità.

Fisher. Viene formulata un'ipotesi falsificabile, chiamata **ipotesi nulla**, H_0, che viene ritenuta vera fino a prova contraria. Il test di verifica delle ipotesi ha l'obiettivo di verificare se i dati smentiscono l'ipotesi.

Vengono presentati alcuni esempi:

- moneta truccata
- moneta truccata, revisited
- coniglio ludopatico e bosone di Higgs
- …

Più in particolare, viene scelta come **statistica test** x un parametro disponibile, rilavante per il fenomeno indagato e di cui è nota la distribuzione di probabilità - o una sua ragionevole approssimazione/attesa - sotto l'ipotesi nulla,

$$p(x|\mathrm{H}_0) \ .$$

In base alle caratteristiche del fenomeno indagato vengono scelte le caratteristiche del test; in base all'evidenza richiesta contro l'ipotesi H_0 viene scelto il **livello di significatività**, α, del test che - insieme alle caratteristiche del test - determina le **regioni di rifiuto** e, per differenza, **di accettazione** dell'ipotesi.

Una volta raccolti i dati, si calcola la statistica scelta con i dati disponibili, si verifica se il suo valore appartiene alle regioni di rifiuto o di accettazione dell'ipotesi, per concludere rispettivamente se l'ipotesi H_0 è stata falsificata o non è stata falsificata e quindi va ritenuta ancora valida.

A partire dalla formulazione generale del test di verifica delle ipotesi secondo Fisher, vengono poi definiti diversi test:

- Z-test: test di Fisher sulla media del campione

- t-test: test di Fisher sulla media del campione di una popolazone con media e varianza sconosciuti che, sotto opportune ipotesi, è una variabile casuale che segue una distribuzione di probabilità t-Student
- $ANOVA$

18.2.1 Test di verifica d'ipotesi - Fisher

Il test di verifica di un'ipotesi di Fisher si ispira al principio di falsificabilità di Popper, e può essere riassunto nei seguenti passaggi:

1. formulazione di un'ipotesi falsificabile, definita **ipotesi nulla** H_0, da verificare e che viene ritenuta vera fino a prova contraria
2. scelta di una variabile esplicativa, o **statistica test**, x, un parametro disponibile, rilevante per il fenomeno indagato e di cui è nota - o approssimabile, sotto ipotesi ragionevoli sul fenomeno - la distribuzione di probabilità, $$p(x|\mathrm{H}_0)$$
3. scelta del **test statistico** (es. una coda o due code,…), in base anche alle caratteristiche del fenomeno indagato
4. scelta del **livello di significatività**, α, del test; il valore del livello di significatività traduce «il livello di evidenza richiesto» per falsificare l'ipotesi e - insieme alle caratteristiche del fenomeno e del test - determina le *regioni di rifiuto e di accettazione dell'ipotesi*, gli intervalli di valori della statistica test x che determinano se l'ipotesi è stata falsificata o meno;
5. raccolta dati e **calcolo statistica test sul campione**
6. **confronto** del valore calcolato della statistica test con gli intervalli di rifiuto e accettazione della variabile soggetta all'ipotesi nulla, e **verdetto sull'ipotesi**

Esempio: moneta truccata o no?

Dati i risultati di n lanci di una moneta, si vuole stabilire con una certa probabilità se la moneta è truccata o meno.

Il lancio di una moneta viene modellato come una variabile casuale X di Bernoulli, con due possibili uscite testa, $X = 0$, o croce, $X = 1$. La forma generale della distribuzione di probabilità di una variabile casuale di Bernoulli $B(p)$ è

$$p(X) = \begin{cases} p & , \quad X = H : \text{Head} \\ 1 - p & , \quad X = T : \text{Tail} \end{cases} ,$$

essendo $p \in [0, 1]$ la probabilità associata al valore $X = H$ e $1 - p$ quella associata al valore $X = T$.

Per la verifica dell'ipotesi si organizza una campagna sperimentale di n lanci e si sceglie come **statistica test** x il numero di volte che il risultato del lancio è testa, $X = H$. Per una variabile casuale con distribuzione di probabilità di Bernoulli $B(p)$, il numero x di risultati $X = H$ in n ripetizioni indipendenti dell'evento è a sua volta una variabile casuale, con distribuzione di probabilità binomiale $\mathcal{B}(n, p)$

$$p_n(x) = \binom{n}{x} p^x (1 - p)^{n-x}$$

Import librerie

```
# import libraries
%reset -f
import numpy as np
import scipy as sp

import matplotlib.pyplot as plt
```

Campione generato dal processo casuale

Viene lanciata una moneta n_{flips} volte. Dati

```
#> Random process of interest, unknown execpt for the sample.
p_head = .5
p_tail = 1. - p_head

# Coin flip as a Bernoulli probability with outcomes: a = [0, 1] with prob p = [p_
↪head, p_tail]
flip_rng = np.random.default_rng(42).choice
n_flips = 30
flip_params = { 'a': [0,1], 'p': [p_head, p_tail], 'size': n_flips }

#> Running coin flip experiment
ov = flip_rng(**flip_params)

coin_dict = [ 'head', 'tail']
# print([ coin_dict[o] for o in ov])
print("\n> Experiment")
print(f"n.samples: {n_flips}")
print(f"heads:tails {np.sum(ov)}:{n_flips - np.sum(ov)}")
print()
```

```
> Experiment
n.samples: 30
heads:tails 17:13
```

```
# ! Approach for low-dimensional problems, with no memory or performance issues

#> Null hypotesis, H0: the coin is fair
# The outcome of a single flip is a r.v. distributed like a Bernoulli variable with p_
↪head = .5
# Every flip is statistically independent from the other
ixv = np.arange(n_flips+1)
xv = ixv / n_flips

#> Test statistics: average n.of heads in flip_n samples
# H0 hypotesis implies that the outcome of flip_n is a r.v. with Binomial pdf
x_H0_fv = sp.stats.binom.pmf(ixv, n_flips, p_head)

#> Test characteristics: symmetric
test_type = 'value'  # 'symmetric', 'right', 'left', 'value'
```

(continues on next page)

(continua dalla pagina precedente)

```
#> Significance level, alpha = .05 ("default")
alpha = .05
```

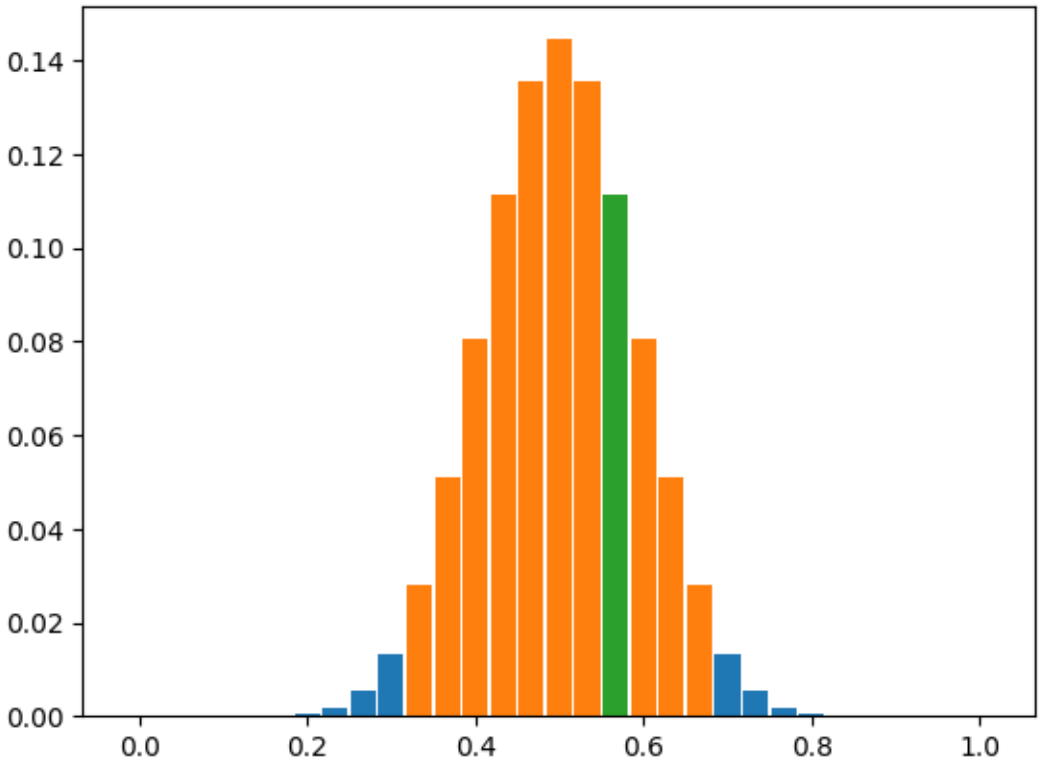
```
#> Acceptance and rejection regions, for discrete pdf
# Starting from the value of the test statistics x_max = max(x_H0_fv), expand

def find_acceptance_region(p, alpha, test_type='value'):
    """ Find acceptance region for a discrete pdf, supposed to be unimodal """
    ix_max = np.argmax(p)
    nx = len(p)
    threshold = 1. - alpha

    # Initialization
    p_acc, ixl, ixr = p[ix_max], ix_max, ix_max

    # if ( test_type == 'value' ):
    while ( p_acc < threshold ):
        if ( p[ixl-1] >= p[ixr+1] ):
            ixl -= 1;  p_acc += p[ixl]
        else:
            ixr += 1;  p_acc += p[ixr]

    # else:
    return ixl, ixr
```

```
#> Compute acceptance region
ixl, ixr = find_acceptance_region(x_H0_fv, alpha,)

#> Evaluate test statistics on the samples
ixs = np.sum(ov)
xs  = ixs / n_flips
```

```
# Compare test statistics on the samples with the acceptance region
if ( ixl <= ixs <= ixr ):
    print("H0 accepted")
else:
    print("H0 rejected")

ix_acc = np.arange(ixl, ixr+1)
x_acc = xv[ix_acc]

dx_bar = (xv[1] - xv[0])*.90
plt.figure()
plt.bar(   xv, x_H0_fv        , width=dx_bar)
plt.bar(x_acc, x_H0_fv[ix_acc], width=dx_bar)
plt.bar(   xs, x_H0_fv[ixs]   , width=dx_bar)

# print(xv)
```

```
H0 accepted
```

```
<BarContainer object of 1 artists>
```



18.2.2 Moneta truccata

Si vuole valutare se una moneta è truccata. L'ipotesi nulla H_0 da falsificare è «la moneta non è truccata». Si sceglie come statistica test la percentuale di risultati «testa» sul numero di lanci.

L'esperimento viene condotto per un numero incrementale di esperimenti.

Import librerie e funzioni utili

Import librerie

Reset delle variabili e import librerie.

Funzione per lo svolgimento di un esperimento

La funzione per lo svolgimento di un esperimento qui definita prende come argomenti un generatore di numeri casuali e i suoi parametri, e restituisce i campioni prodotti secondo la distribuzione data. Vengono definiti dei parametri di default, corrispondenti a una distribuzione di Bernoulli uniforme

Funzione che restituisce la pdf $p(x|\mathrm{H}_0)$

Funzioni per la ricerca degli intervalli di accettazione dell'ipotesi

Vengono qui definite due funzoni per la ricerca degli intervalli di accettazione, dato il livello di significatività del test richiesto. La prima funzione ricerca un unico intervallo di accettazione riferito a un unico livello di significatività; la seconda funzione ricerca tanti intervalli di accettazione quanti sono i livelli di significatività cercati: ad esempio, si possono cercare simultaneamente i livellidi significatività associati a σ, 2σ, 3σ…

Esperimento

Livelli di accettazione

Svolgimento esperimento

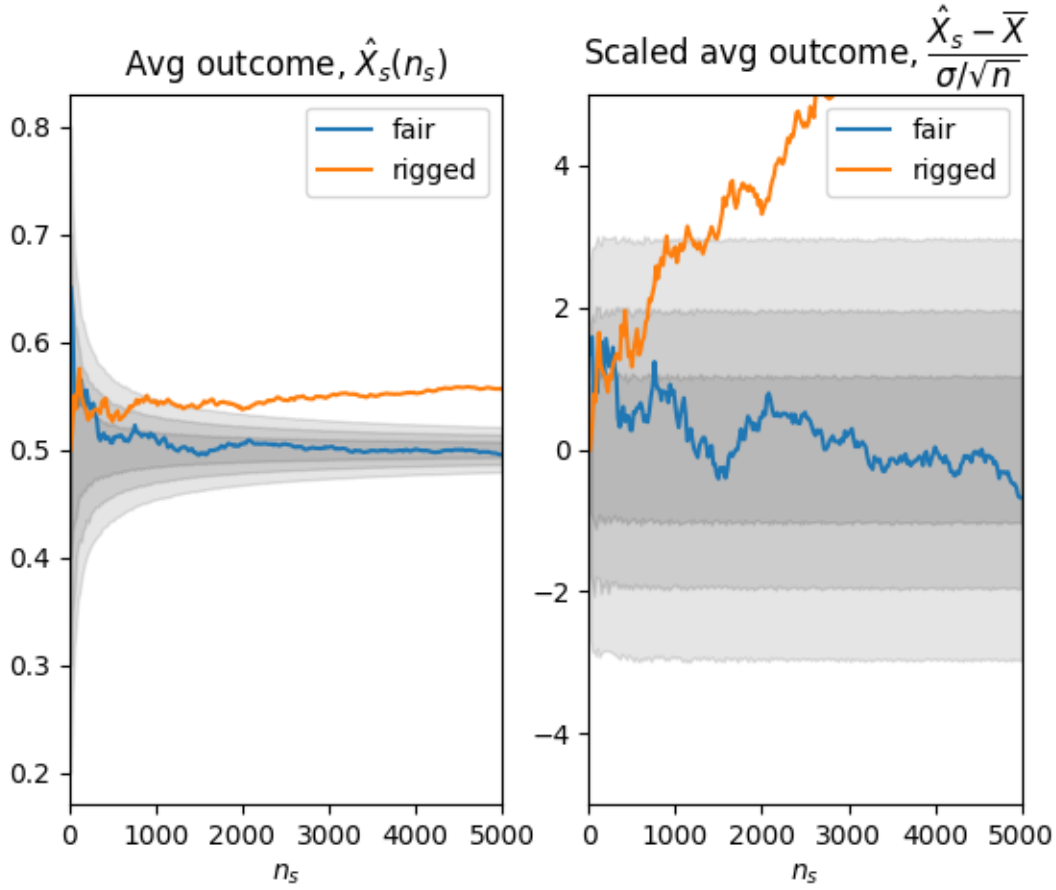
L'esperimento viene svolto usando due monete, la cui natura è a priori incognita, che possono essere modellate con variabili casuali di Bernoulli: una moneta non truccata produce testa o croce con la probaiblità uniforme 0.5; una moneta truccata che produce testa o croce con probabilità $0.45 - 0.55$

```
/home/davide/.local/lib/python3.8/site-packages/scipy/stats/_discrete_distns.
↪py:77: RuntimeWarning: divide by zero encountered in _binom_pdf
  return _boost._binom_pdf(x, n, p)
```

Grafici

Il valore medio delle volte che è uscita croce viene usata come statistica test. Il valore della statistica test per gli esperimenti condotti con le due monete viene confrontato con la funzione di distribuzione $p(x|\mathrm{H}_0)$ per una moneta non truccata. Viene mostrato l'andamento delle statistiche test e degli intervalli di accettazione in funzione del numero di lanci della moneta. Vengono mostrati i dati raccolti e successivamente depurati della media attesa e scalati per la varianza della media campionaria $\frac{\sigma}{\sqrt{n_s}}$, per mettere in evidenza i livelli di significatività $[0.3, 0.05, 0.003]$

```
<matplotlib.legend.Legend at 0x7f8b05018ee0>
```



18.2.3 Gambler rabbit

18.2.4 t-test

Un t-test è un test di verifica delle ipotesi nel quale la **statistica test** ha una distribuzione **t di Student** sotto l'ipotesi nulla H_0,

$$p(x|\mathrm{H}_0) \sim t_\nu$$

t-test per un campione

Il t-test per un campione è un test di verifica del valore di un parametro, per la media di una variabile casuale sotto un'opportuna ipotesi nulla H_0. Il parametro t,

$$t = \frac{\hat{x} - \mu_0}{\frac{\hat{s}}{\sqrt{n}}} \,,$$

confronta la distanza della media $\bar{x}$ del campione dal valore medio μ_0 della distribuzione $p(x|\mathrm{H}_0)$, con la deviazione standard del campione s opportunamente scalata della grandezza del campione n.

Se le osservazioni della variabile casuale sono indipdendenti tra di loro, allora t è una variabile casuale che tende a una variabile normale $\mathcal{N}(0, 1)$ per il teorema del limite centrale **todo** *link a una sezione sul campionamento*

```
# Example

import numpy as np
from scipy.stats import ttest_1samp, t

import matplotlib.pyplot as plt

# Test statistics has expected value mu0 under H0 hypotesis
mu0 = 86.          # Expeceted value of the test statistics under H0
sigma = 0.05       # Significance level


# Sample data: test scores from two classes
sample = [88, 92, 85, 91, 87]
n_s = len(sample)
mu_s = np.sum(sample)/n_s
s_s = ( np.sum((sample-mu_s)**2)/(n_s-1) )**.5

print("Sample statistics:")
print(f" sample avg    : {mu_s}")
print(f" sample std.dev: {s_s}")


# Perform independent t-test
t_statistic, p_value = ttest_1samp(sample, mu0)

print(f"\n1-sample t-test")
print(f" t-Statistic: {t_statistic}")
print(f" p-Value    : {p_value}")


# Interpretation
if p_value < sigma:
    print("\n> Reject the null hypothesis, H0: the means of the two classes are␣
 ↪significantly different.")
else:
    print("\n> Fail to reject the null hypothesis, H0: no significant difference in␣
 ↪means between the classes.")

x_plot = np.arange(80, 100, .01)
t_pdf_plot = t.pdf(x_plot, n_s, mu_s, s_s)

plt.figure()
plt.plot(x_plot, t_pdf_plot, color=plt.cm.tab10(0))
plt.plot(sample, np.zeros(n_s), 'o', color=plt.cm.tab10(0))
plt.plot(mu_s, 0., 'x', color=plt.cm.tab10(0), markersize=10, markeredgewidth=2)
plt.plot(mu0 , 0., 'x', color='black'        , markersize=10, markeredgewidth=2)
```

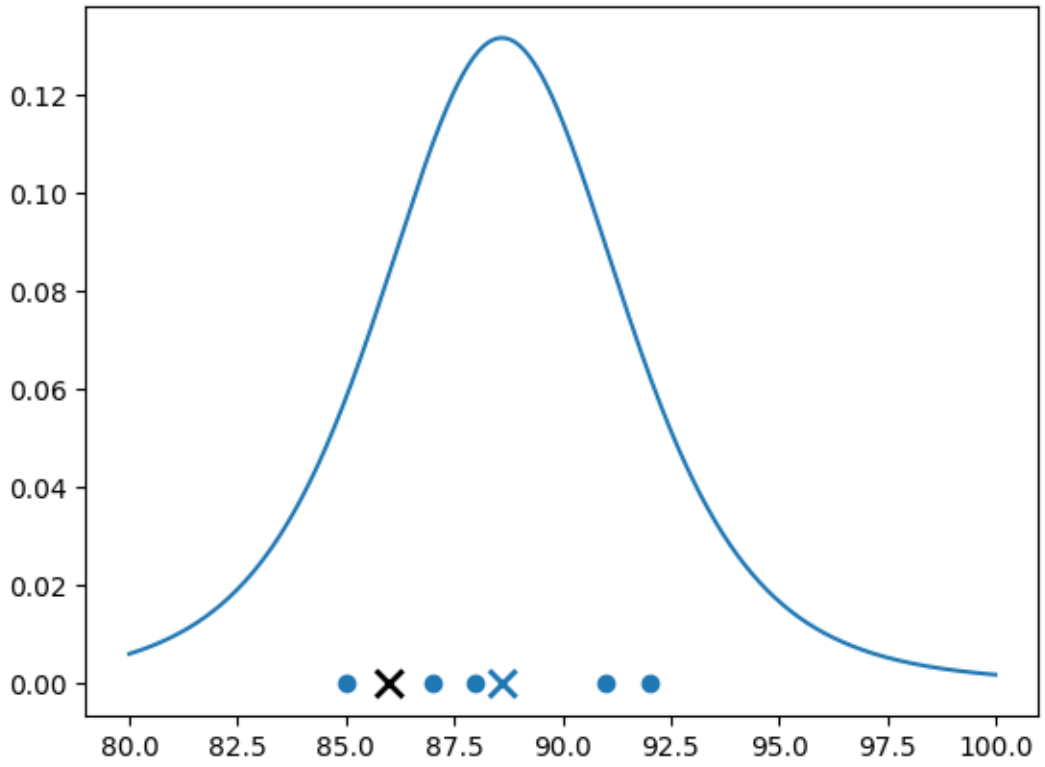
```
Sample statistics:
 sample avg    : 88.6
 sample std.dev: 2.8809720581775866

1-sample t-test
 t-Statistic: 2.017991366836461
 p-Value    : 0.11375780482862627
```

(continues on next page)

(continua dalla pagina precedente)

```
> Fail to reject the null hypothesis, H0: no significant difference in means␣
↪between the classes.
```

```
[<matplotlib.lines.Line2D at 0x7f3e55ff4760>]
```



t-test per due campioni

- test con varianza uguale (o simile), t-test/test con varianza diversa, *Welch*
- campioni indipendenti/campioni dipendenti (paired samples)

t-test per due variabili indipendenti, con varianza simile e stesso numero di osservazioni

Nell'ipotesi che i campioni siano ottenuti da due variabili indipendenti, i risultati ottenuti nella sezione sulla *combinazione di variabili casuali*

- il **valore atteso** della somma/differenza di variabili casuali indipendenti è uguale alla somma/differenza dei valori attesi delle singole variabili
- la **varianza** della somma/differenza di variabili casuali indipendenti è uguale alla somma delle varianze delle singole variabili.

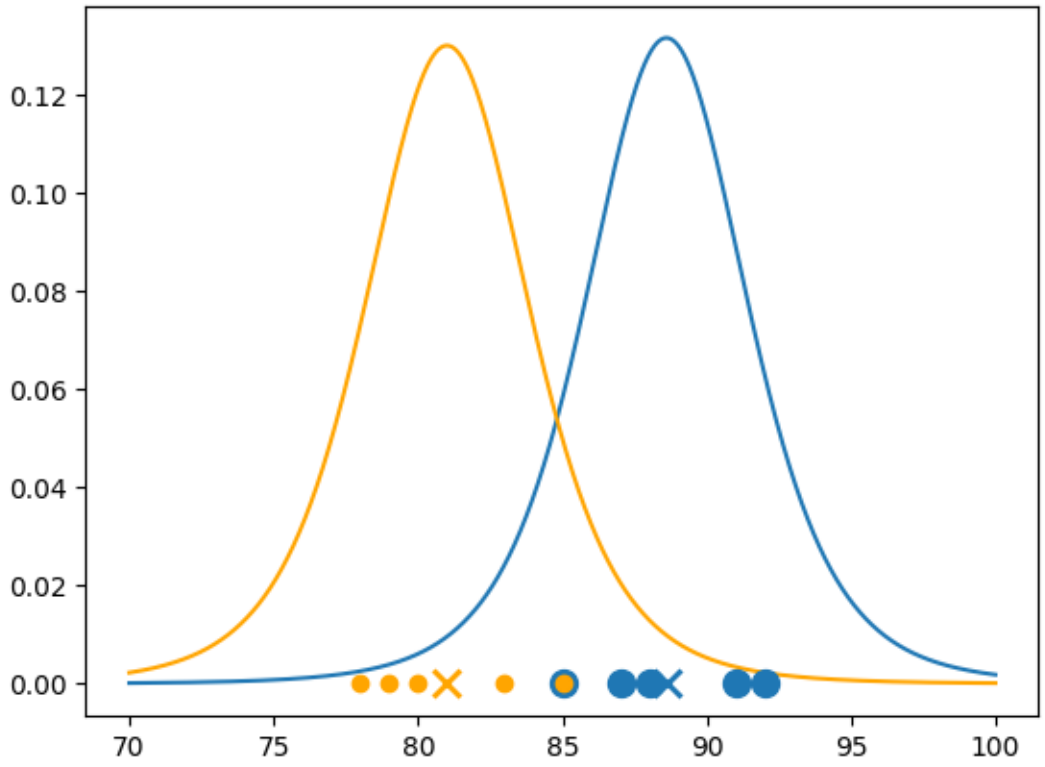
Il t-test per due campioni equivale al t-test per un campione uguale alla differenza delle osservazioni nei due campioni,

$$z_i = x_i - y_i \, .$$

```
# Example

import numpy as np
from scipy.stats import ttest_ind

# Significance level
sigma = 0.05

# Sample data: test scores from two classes
class_A = [88, 92, 85, 91, 87]
class_B = [78, 85, 80, 83, 79]

nA = len(class_A); mu_A = np.sum(class_A)/nA; sA = ( np.sum((class_A-mu_A)**2)/(nA-1)␣
↪)**.5
nB = len(class_B); mu_B = np.sum(class_B)/nB; sB = ( np.sum((class_B-mu_B)**2)/(nB-1)␣
↪)**.5

print("Sample statistics:")
print("Sample A")
print(f" sample avg    : {mu_A}")
print(f" sample std.dev: {sA}")
print("Sample B")
print(f" sample avg    : {mu_B}")
print(f" sample std.dev: {sB}")


# Perform independent t-test
t_statistic, p_value = ttest_ind(class_A, class_B)


print(f"\ns-sample t-test")
print(f" t-Statistic: {t_statistic}")
print(f" p-Value    : {p_value}")

print((mu_A-mu_B)/((sA**2 + sB**2)/nA)**.5)

# Interpretation
if p_value < sigma:
    print("\nReject the null hypothesis: The means of the two classes are␣
↪significantly different.")
else:
    print("\nFail to reject the null hypothesis: No significant difference in means␣
↪between the classes.")

x_plot = np.arange(70, 100, .01)
t_pdf_A = t.pdf(x_plot, nA, mu_A, sA)
t_pdf_B = t.pdf(x_plot, nB, mu_B, sB)

plt.figure()
plt.plot(x_plot, t_pdf_A, color=plt.cm.tab10(0))
plt.plot(x_plot, t_pdf_B, color='orange')
plt.plot(class_A, np.zeros(nA), 'o', color=plt.cm.tab10(0), markersize=10)
plt.plot(class_B, np.zeros(nB), 'o', color='orange'       )
plt.plot(mu_A, 0., 'x', color=plt.cm.tab10(0), markersize=10, markeredgewidth=2)
plt.plot(mu_B, 0., 'x', color='orange'       , markersize=10, markeredgewidth=2)
```

```
Sample statistics:
Sample A
 sample avg    : 88.6
 sample std.dev: 2.8809720581775866
Sample B
 sample avg    : 81.0
 sample std.dev: 2.9154759474226504

s-sample t-test
 t-Statistic: 4.146139914483853
 p-Value    : 0.0032260379191180397
4.146139914483853

Reject the null hypothesis: The means of the two classes are significantly␣
 ↪different.
```

```
[<matplotlib.lines.Line2D at 0x7f3e55f11c70>]
```




```
#
```

18.2.5 ANOVA - Analysis of variance

```
# ANOVA - Analysis of variance

import numpy as np
from scipy.stats import f_oneway

# Sample data: test scores from three teaching methods
method_A = [88, 92, 85, 91, 87]
method_B = [78, 85, 80, 83, 79]
method_C = [94, 89, 91, 96, 90]

# Perform one-way ANOVA
f_statistic, p_value = f_oneway(method_A, method_B, method_C)

# Display results
print(f"F-Statistic: {f_statistic}")
print(f"P-Value: {p_value}")

# Interpretation
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant difference between the␣
 ↪groups.")
else:
    print("Fail to reject the null hypothesis: No significant difference between the␣
 ↪groups.")
```

```
F-Statistic: 18.806324110671927
P-Value: 0.00020023178541394456
Reject the null hypothesis: There is a significant difference between the groups.
```

Parte IV

Introduzione ai metodi in statistica e AI

CAPITOLO 19

Introduzione ai metodi in AI

In questa sezione si descriveranno alcuni metodi utilizzati nelle applicazioni di **machine learning**. I contenuti sono organizzati in 3 sezioni, seguendo una delle più comuni classificazioni in machine learning.

Approccio. Alla completezza e al rigore dello sviluppo teorico dei metodi, viene preferita una descrizione dei metodi tramite esempi e applicazioni.

Benché questo possa essere un approccio ad alto rischio di creazione di utenti acritici di strumenti che non comprendono, la speranza è di mitigare questo rischio con continui moniti a prestare attenzione e un supporto del lettore da parte di gente con un minimo di esperienza.

Argomenti. In questa sezione verranno presentati alcuni metodi e applicazioni del **machine learning**, che possono essere classificate in tre grandi classi di apprendimento:

- SL, supervised learning: regression and classification
- UL, unsupervised learning: clustering
- ML, machine learning: control

Dopo aver presentato le tecniche classiche, verranno introdotte le reti neurali, gli algoritmi fondamentali che hanno permesso un uso pratico ed efficiente di reti profonde, e alcune architetture fondamentali di reti neurali.

Le applicazioni verranno affrontate inizialmente con approcci classici, affidandoci alla libreria sci-kit, e successivamente con tecniche «deep» grazie alla libreria PyTorch.

CAPITOLO 20

Supervised Learning

CAPITOLO 21

Unsupervised Learning

Le principali attività dell'applrendimento non supervisionato sono:

- il clustering: raggruppamento dei dati in classi più o meno omogenee
- la riduzione delle dimensioni: sintesi delle dinamiche/informazioni «principali» contenute nei dati. Una caratteristica che permette di distinguere gli algoritmi è la definizione di «principali»; così ad esempio:
  - PCA (o la cara vecchia SVD): seleziona le dinamiche «più energetiche» tra di loro ortogonali contenute in un segnale
  - ICA: seleziona le dinamiche «maggiormente indipendenti» contenute nel segnale

  Un confronto tra ICA e PCA può chiarire il significato delle diverse definizioni e le differenze nei risultati prodotti.

21.1 PCA

21.1.1 Esempio: compressione di immagini

Un'immagine in scala di grigio può essere rappresentata con una matrice **A**, i cui elementi rappresentano il valore di grigio di ogni pixel.

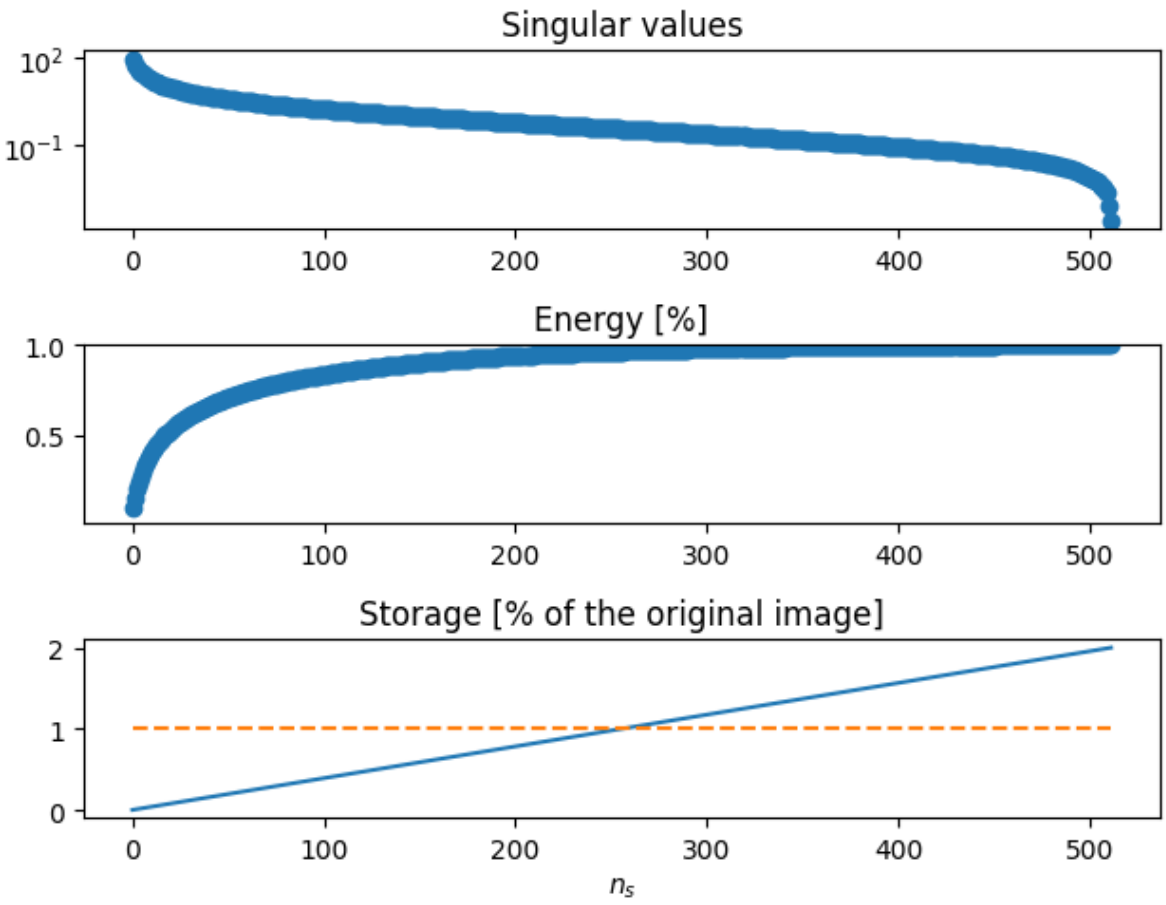
```
(-0.5, 511.5, 511.5, -0.5)
```

Preprocessing e SVD

Viene rimosso il valore medio dall'immagine, prima di valutare la SVD della matrice **A**

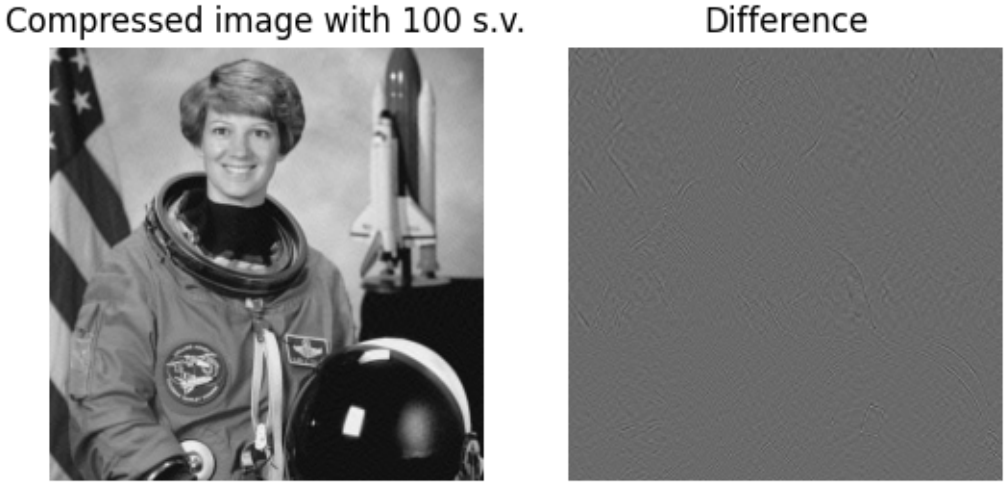
Risultati della SVD

I valori singolari rappresentano il contenuto energetico dei singoli modi. La somma cumulativa dei valori singolari rappresenta il contenuto energetico della



Ricostruzione dell'immagine compressa

```
Image reconstruction with 100 singular values:
> Energy error [% original]: 0.1647581016503445
> Storage      [% original]: 0.3870964050292969
```



21.2 ICA e PCA

21.2.1 Riferimenti

scikit

- https://scikit-learn.org/dev/modules/generated/sklearn.decomposition.FastICA.html

Carten Klein

- https://github.com/akcarsten/
- https://github.com/akcarsten/Independent_Component_Analysis

21.2.2 FastICA on 2D point clouds

Esempio disponibile sul sito di scikit-learn.

Librerie e funzioni di comodo

Import librerie

```
%reset -f

import numpy as np
from sklearn.decomposition import PCA, FastICA

import matplotlib.pyplot as plt
```

Funzioni utili

```
### utils

def plot_samples(S, axis_list=None):
    """ util function for plotting scatter plots """
    plt.scatter(
        S[:,0], S[:,1], s=2, marker="o", zorder=10, color="steelblue", alpha=0.5
    )
    if axis_list is not None:
        for axis, color, label in axis_list:
            x_axis, y_axis = axis / axis.std()
            plt.quiver(
                (0, 0),
                (0, 0),
                x_axis,
                y_axis,
                zorder=11,
                width=0.01,
                scale=6,
                color=color,
                label=label,
```

(continues on next page)

(continua dalla pagina precedente)

```
            )

    plt.hlines(0, -3, 3, color="black", linewidth=0.5)
    plt.vlines(0, -3, 3, color="black", linewidth=0.5)
    plt.xlim(-3, 3)
    plt.ylim(-3, 3)
    plt.gca().set_aspect("equal")
    plt.xlabel("x")
    plt.ylabel("y")
```

Generazione campione

La formulazione più comune dei metodi usa l'espressione

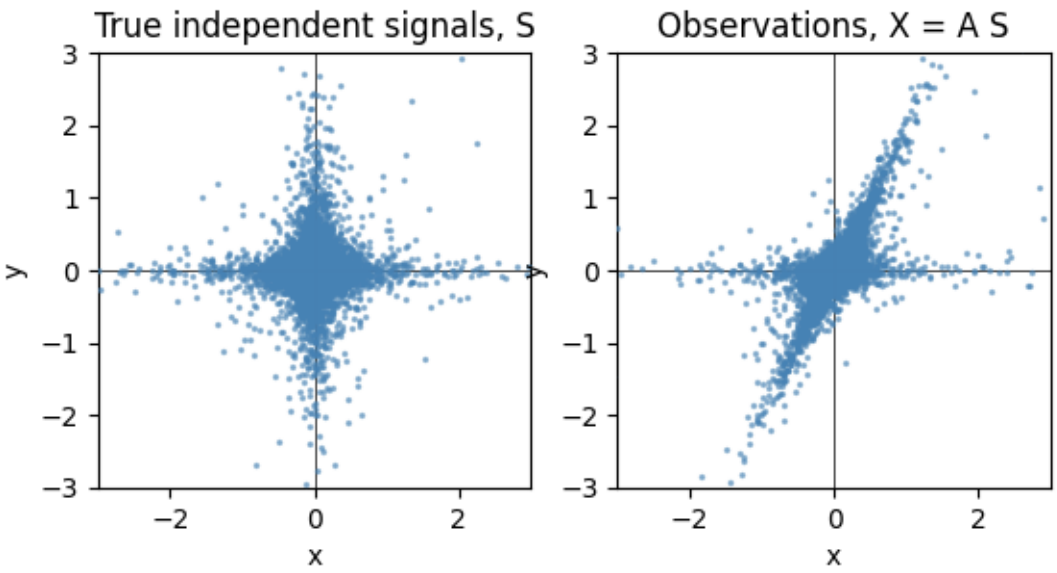
$$\mathbf{X} = \mathbf{A}\,\mathbf{S}\,,$$

che lega osservazioni X e segnali S tramite la matrice di mixing A. Per le strutture dati di Python, è conveniente (**todo** *provare! O è dovuto solo alla forma mentis di chi usa Python?*) scrivere la relazione trasposta,

$$\mathbf{X}^T = \mathbf{S}^T\,\mathbf{A}^T\,.$$

```
#> Sample data generation
# Initialize a default random number generator
rng = np.random.default_rng(42)
# rng = np.random.RandomState(42)
# np.random.RandomState() is deprecated! Use random.default_rng(),
# but FastICA needs a np.random.RandomSstate as an optional input random_state

# Create data: non-isotropic mixing of 2 t-Student variables:
# 1. Create signals S, as 2 t-Student distributions
# 1. sampling from a t-Student distribution with degrees of freedom df
# matrix S has dimensions (n_rows, n_cols) = (n_samples, n_dim), interpreting
# each row as a sample, and each row as a dimension of the data
df, n_samples, n_dims = 1.5, 10000, 2
S = rng.standard_t(df, size=(n_samples, n_dims))

# 2. scale one component
S[0,:] *= 2.0
plt.subplot(1, 2, 1)
plot_samples(S / S.std())
plt.title("True independent signals, S")

# 3. Mix components
A = np.array([[1, 1], [0, 2]])  # Mixing matrix
X = np.dot(S, A.T)              # Generate observations

plt.subplot(1, 2, 2)
plot_samples(X / X.std())
plt.title("Observations, X = A S")
```

```
Text(0.5, 1.0, 'Observations, X = A S')
```



ICA e PCA

```
#> PCA
pca = PCA()
S_pca_ = pca.fit(X).transform(X)

#> ICA
ica = FastICA(random_state=3, whiten="arbitrary-variance")
# ica = FastICA(random_state=np.random.RandomState(42), whiten="arbitrary-variance")
S_ica_ = ica.fit(X).transform(X)  # Estimate the sources
```

Risultati

```
#> Print results:
# Normalization: principal and independent components are usually defined up to a␣
 ↪multiplicative factor;
# PCA and ICA provides information abouth the "shape" of the main components in a␣
 ↪signal; usually, it's a
# good practice to have normalized info/results, that contains only shape info and no␣
 ↪other arbitrary (non)-info
# - PCA results are usually normalized
# - ICA results seem to be not normalized by the algorithm; easy to perform␣
 ↪normalization, as done below,
#   to have unit-norm vectors to be easily compared with PCA.
#   If normalization done outside functions, remember to normalize both signals and␣
 ↪mixing matrix

print("\nPCA, principal components (rows of the matrix)")
print(pca.components_ )
print("\nICA. indepdent components (rows of the matrix)")
print(ica.mixing_.T)                                                    # non␣
 ↪normalized
```

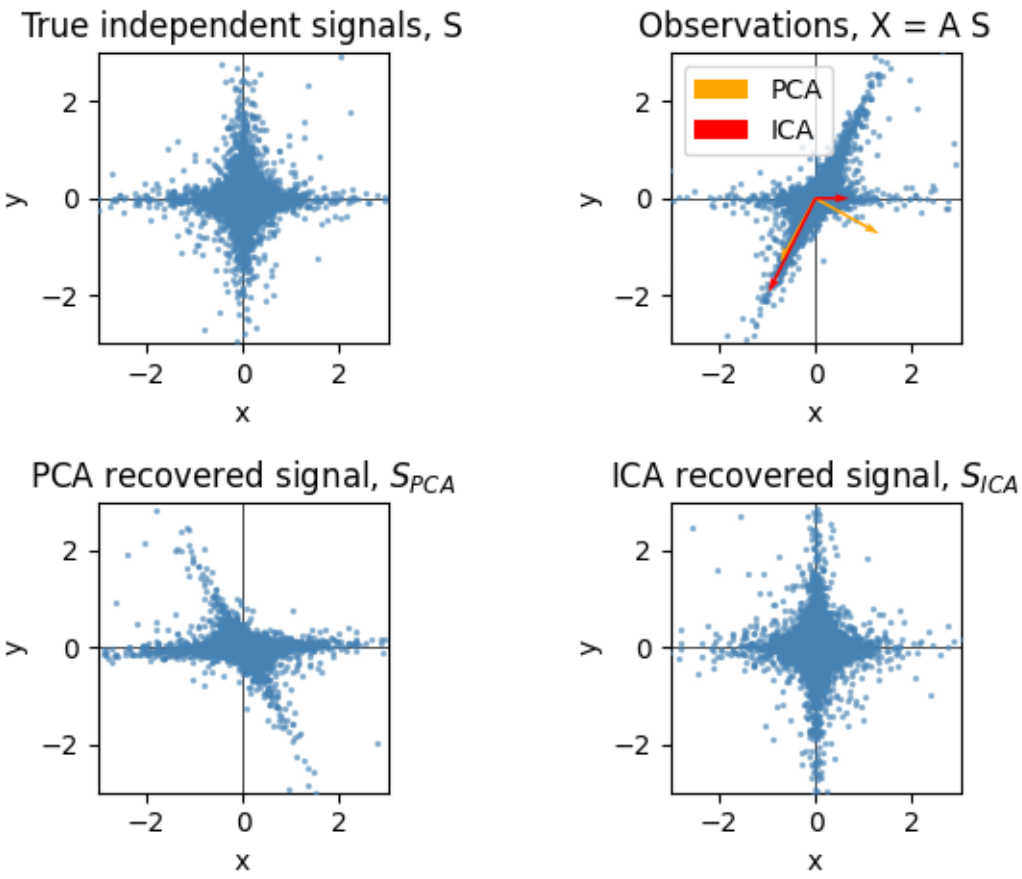
(continues on next page)

(continua dalla pagina precedente)

```
# print(ica.mixing_.T / np.linalg.norm(ica.mixing_.T,axis=1)[:, np.newaxis])  #␣
 ↪normalized
print()
```

```
PCA, principal components (rows of the matrix)
[[-0.48647086 -0.8736968 ]
 [ 0.8736968  -0.48647086]]

ICA. indepdent components (rows of the matrix)
[[-9.46908181e+02 -1.89462935e+03]
 [ 6.85936578e+02 -8.07940837e-01]]
```

```
#> Plots
plt.figure()
plt.subplot(2, 2, 1)
plot_samples(S / S.std())
plt.title("True independent signals, S")

axis_list = [(pca.components_.T, "orange", "PCA"), (ica.mixing_, "red", "ICA")]
plt.subplot(2, 2, 2)
plot_samples(X / np.std(X), axis_list=axis_list)
legend = plt.legend(loc="upper left")
legend.set_zorder(100)

plt.title("Observations, X = A S")

plt.subplot(2, 2, 3)
plot_samples(S_pca_ / np.std(S_pca_))
plt.title("PCA recovered signal, $S_{PCA}$")

plt.subplot(2, 2, 4)
plot_samples(S_ica_ / np.std(S_ica_))
plt.title("ICA recovered signal, $S_{ICA}$")

plt.subplots_adjust(0.09, 0.04, 0.94, 0.94, 0.26, 0.36)
plt.tight_layout()
plt.show()
```



21.2.3 ICA e PCA: segnali in tempo

21.2.4 ICA e PCA - dettagli

Prendendo spunto da quanto fatto da A.K.Carsten si discutono alcuni dettagli dell'algoritmo ICA,[1] applicandolo a un problema di riconoscimento delle componenti indipendenti di segnali in tempo.

In particolare, si vuole discutere **todo**:

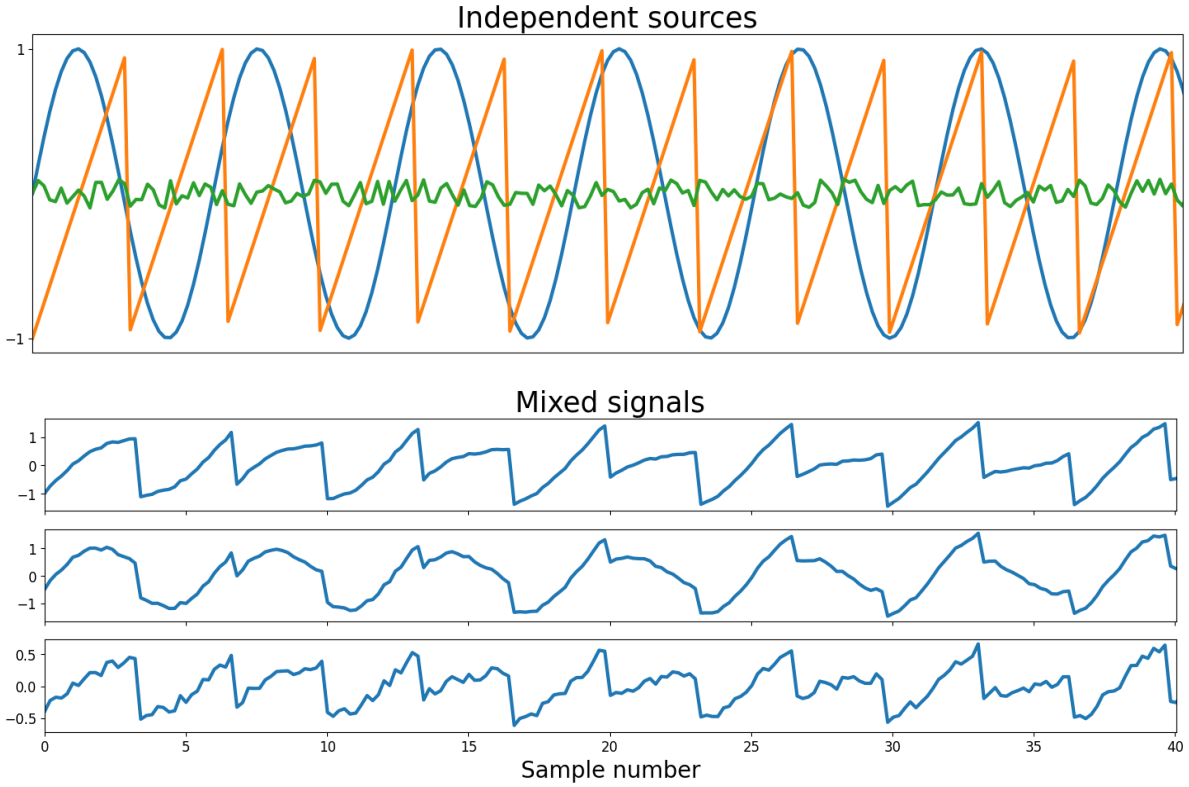
- i dettagli dell'algoritmo, prestando attenzione a
  - la relazione «non-gaussianità» $\sim$ «indipendenza»
  - la misura di non gaussianità tramite neg-entropia, e le sue approssimazioni
  - l'espressione dell'iterazione di Newton nell'ottimizzazione della neg-entropia, che dà vita a un problema di punto fisso
- la necessità della non-gaussianità dei segnali

[1] A.Hyvarinen, E.Oja *Independent component analysis: algorithms and applications»

Librerie e funzioni utili

Generazione segnale

Viene generato un campione test con 200 campioni di 3 osservazioni, come risultato del mix di 3 segnali indipendenti, da ricostruire con l'algoritmo ICA.



Algoritmo

Pre-processing

L'osservazione $\mathbf{X}$ vengono **depurate dalla media** e viene usata una trasformazione per combinare le osservazioni originali e ottenere un nuovo segnale $\mathbf{X}_w$ con **componenti non correlate**. Il procedimento viene illustrato per i dati organizzati nella matrice,

$$\mathbf{X} = [\mathbf{x}_1 | \mathbf{x}_2 | \dots | \mathbf{x}_s] \ ,$$

in cui la colonna j-esima contiene le n_x osservazioni relative all'indice j (istante di tempo, paziente,…), mentre la riga i-esima contiene l'osservazione della quantità identificata dall'indice i per ogni valore di j (istante di tempo, paziente,…).

Teoria

Vengono svolte due operazioni:

- rimozione della media dalle osservazioni di ogni quantità, quindi rimozione del valore medio di ogni riga, $\$X_{ij} \leftarrow X_{ij} - \frac{1}{n_s}\sum_{j=1}^{n_s} X_{ij}$,$ in modo da avere i nuovi segnali a media nulla;
- ricerca di una trasformazione di coordinate (combinazione delle osservazioni) che renda la nuove componenti non correlate. Una stima senza bias della correlazione è

$$\hat{\mathbf{R}} = \frac{1}{n_s - 1}\mathbf{X}\,\mathbf{X}^* \,.$$

  In generale, questa è una matrice piena di dimensioni (n_x, n_x). Si vuole cercare la trasformazione di coordinate $\mathbf{X}_w = \mathbf{W}_w\mathbf{X}$, che garantisce che la nuove osservazioni siano non correlate con varianza unitaria (come *white noise*, e da qui il nome *whitening*), cioè

$$\mathbf{I} = \hat{\mathbf{R}}_w = \frac{1}{n_s - 1}\mathbf{X}_w\,\mathbf{X}_w^* = \frac{1}{n_s - 1}\mathbf{W}_w\,\mathbf{X}\,\mathbf{X}^*\,\mathbf{W}_w^* = \mathbf{W}_w\,\hat{\mathbf{R}}\,\mathbf{W}_w^*$$

  E" possibile trovare la matrice desiderata $\mathbf{W}_w$ usando una tecnica di scomposizione della matrice di correlazione $\hat{\mathbf{R}}$, simmetrica (semi)definita positiva, come ad esempio la

  – scomposizione agli autovalori, $\hat{\mathbf{R}} = \mathbf{E}\,\mathbf{\Lambda}\,\mathbf{E}^{-1}$, con $\mathbf{\Lambda}$ diagonale; sfruttando le proprietà delle matrici sdp, si può scrivere $\hat{\mathbf{R}} = \mathbf{E}\,\mathbf{\Lambda}\,\mathbf{E}^*$, con la matrice $\mathbf{E}$ ortogonale tale che $\mathbf{E}\,\mathbf{E}^* = \mathbf{I}$.

  – la SVD, $\hat{\mathbf{R}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$, con $\mathbf{\Sigma}$ diagonale; nel caso di matrice di partenza sdp, si può scrivere si può scrivere $\hat{\mathbf{R}} = \mathbf{U}\mathbf{\Sigma}\mathbf{U}^*$, con la matrice $\mathbf{E}$ ortogonale tale che $\mathbf{U}\,\mathbf{U}^* = \mathbf{I}$.

  Sfruttando quindi la proprietà delle scomposizioni presentate sopra, osservando l'analogia del risultato delle due scomposizioni nel caso di matrice sdp, e definendo $\mathbf{\Sigma}^{1/2}$ la matrice diagonale con elementi le radici quadrate di $\mathbf{\Sigma}$, si può scrivere

$$\begin{aligned}
\mathbf{I} &= \mathbf{W}_w\,\hat{\mathbf{R}}\,\mathbf{W}_w^* = \\
&= \mathbf{W}_w\,\mathbf{U}\mathbf{\Sigma}^{\frac{1}{2}}\mathbf{\Sigma}^{\frac{1}{2}}\mathbf{U}^*\,\mathbf{W}_w^* = \\
&= \mathbf{W}_w\,\mathbf{U}\mathbf{\Sigma}^{\frac{1}{2}}\mathbf{U}^*\mathbf{U}\mathbf{\Sigma}^{\frac{1}{2}}\mathbf{U}^*\,\mathbf{W}_w^* = \\
&= (\mathbf{W}_w\,\mathbf{U}\mathbf{\Sigma}^{\frac{1}{2}}\mathbf{U}^*)\,(\mathbf{W}_w\,\mathbf{U}\mathbf{\Sigma}^{\frac{1}{2}}\mathbf{U}^*)^* \,,
\end{aligned}$$

  e quindi (**todo** *gisutificare la comparsa di* $\mathbf{U}^*\mathbf{U}$ *per ottenere una matrice quadrata, giustificare la scelta di* $\mathbf{W}_w\,\mathbf{U}\mathbf{\Sigma}^{\frac{1}{2}}\mathbf{U}^* = \mathbf{I}$)

$$\mathbf{W}_w\,\mathbf{U}\mathbf{\Sigma}^{\frac{1}{2}}\mathbf{U}^* = \mathbf{I} \qquad \rightarrow \qquad \mathbf{W}_w = \mathbf{U}^*\mathbf{\Sigma}^{-\frac{1}{2}}\mathbf{U}$$

FastICA

Viene qui illustrato l'algoritmo FastICA[Pag. 96, 1]

todo

```
def fastIca(signals,  alpha = 1, thresh=1e-8, iterations=5000):
    m, n = signals.shape

    # Initialize random weights
    W = np.random.rand(m, m)
```

(continues on next page)

(continua dalla pagina precedente)

```
    for c in range(m):
            w = W[c, :].copy().reshape(m, 1)
            w = w / np.sqrt((w ** 2).sum())

            i = 0
            lim = 100
            while ((lim > thresh) & (i < iterations)):

                # Dot product of weight and signal
                ws = np.dot(w.T, signals)

                # Pass w*s into contrast function g
                wg = np.tanh(ws * alpha).T

                # Pass w*s into g prime
                wg_ = (1 - np.square(np.tanh(ws))) * alpha

                # Update weights
                wNew = (signals * wg.T).mean(axis=1) - wg_.mean() * w.squeeze()

                # Decorrelate weights
                wNew = wNew - np.dot(np.dot(wNew, W[:c].T), W[:c])
                wNew = wNew / np.sqrt((wNew ** 2).sum())

                # Calculate limit condition
                lim = np.abs(np.abs((wNew * w).sum()) - 1)

                # Update weights
                w = wNew

                # Update counter
                i += 1

            W[c, :] = w.T
    return W
```

Applicazione del metodo al segnale

Il metodo viene applicato al problema considerato. In questo caso (segnali non gaussiani,…) il metodo riesce a ricostruire bene (**todo** *è un bene «a occhio», qualitativo. Quantificare!*) i 3 segnali indipendenti di partenza **a meno di un fattore moltiplicativo**, arbitrarietà propria del metodo.

todo Dire qualcosa sulle componenti principali indipendenti, discutendo le componenti della matrice **W**

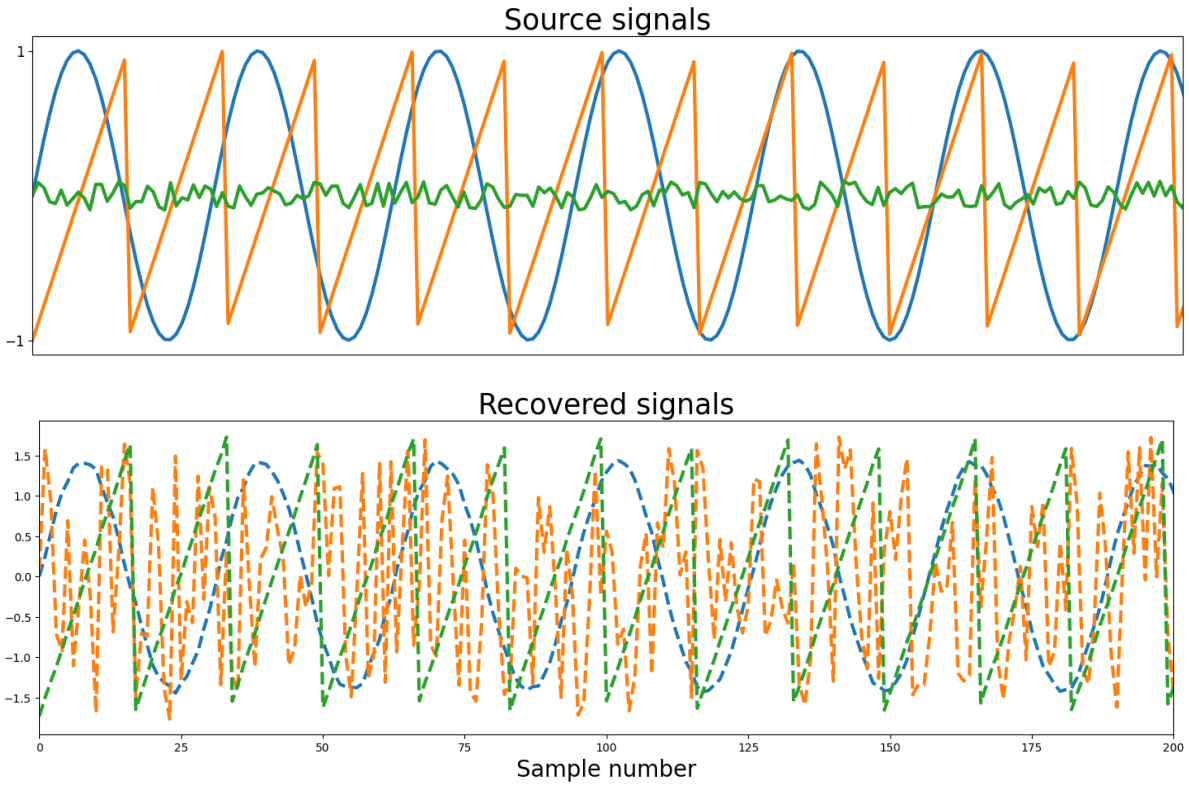
```
#> Preprocessing
# Center signals
Xc, meanX = center(X)

# Whiten mixed signals
Xw, whiteM = whiten(Xc)
```

```
#> FastICA
W = fastIca(Xw,  alpha=1)
```

(continues on next page)

(continua dalla pagina precedente)

```
#> Find unmixed signals using
unMixed = Xw.T.dot(W.T)
```




```
#> Comparison of matrices involved, A, whiteM, W
# todo...
```

CAPITOLO 22

Reinforcement Learning

Parte V

Supporto tecnico

CAPITOLO 23

Supporto tecnico

- git
- Servizi Google
- TeX