Funzioni

- Come abbiamo visto, il concetto di *overload* degli operatori di base viene ulteriormente esteso alle funzioni
- `numpy` mette a disposizione tutta una serie di funzioni matematiche che agiscono sugli *array*.
- Queste funzioni introducono il concetto di *broadcast function*.
- Vediamo quali funzioni sono implementate:

Funzioni: trigonometriche

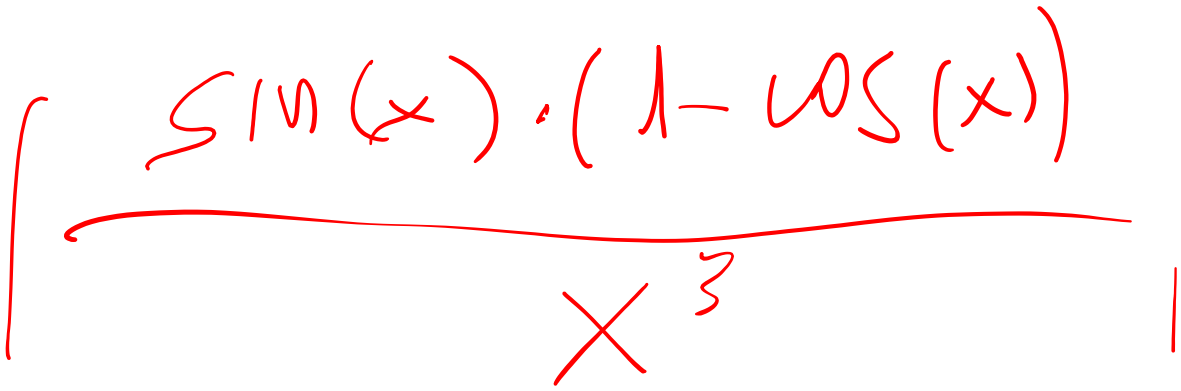
sin()	Trigonometric sine, element-wise.
cos()	Cosine element-wise.
tan()	Compute tangent element-wise.
arcsin()	Inverse sine, element-wise.
arccos()	Trigonometric inverse cosine, element-wise.
arctan()	Trigonometric inverse tangent, element-wise.
hypot(x1, x2)	Given the "legs" of a right triangle, return its hypotenuse.
arctan2(x1, x2)	Element-wise arc tangent of x1/x2 choosing the quadrant correctly.
degrees()	Convert angles from radians to degrees.
radians()	Convert angles from degrees to radians.
deg2rad()	Convert angles from degrees to radians.
rad2deg()	Convert angles from radians to degrees.

Esempio

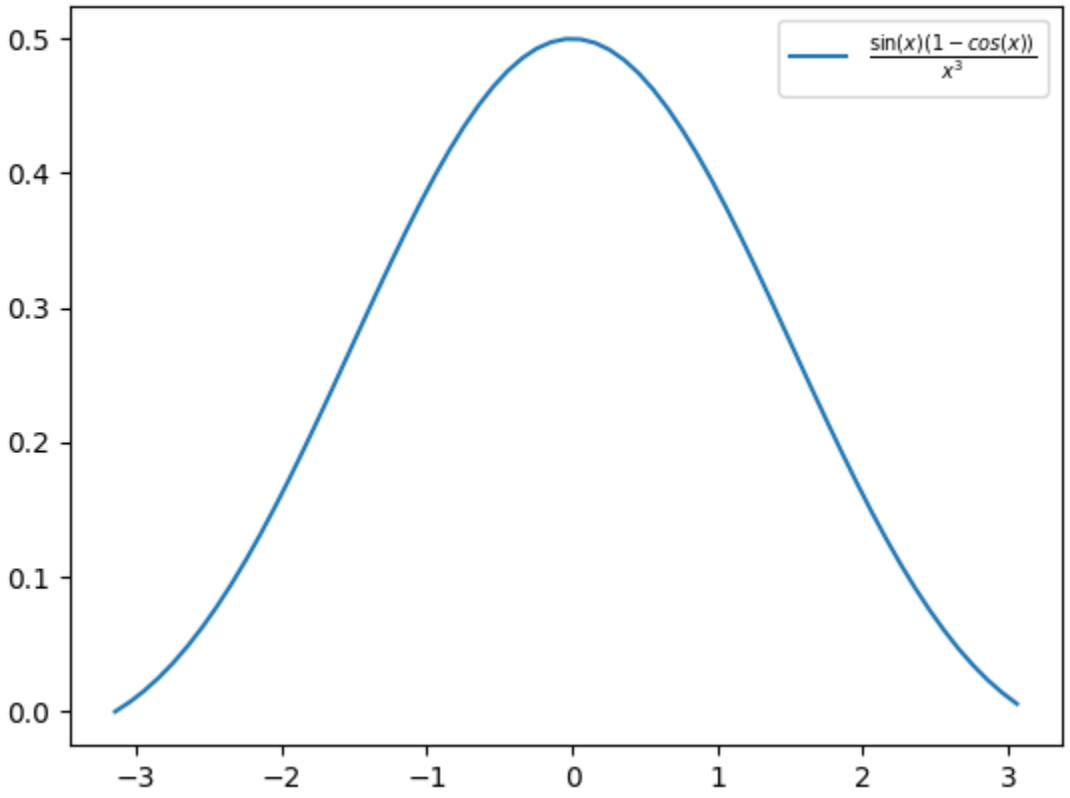
- Grazie a queste speciali funzioni che agiscono all'intero array e all'uso dell'*overload* degli operatori possiamo:

```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(-np.pi, np.pi, .1)
plt.plot(x, (np.sin(x) * (1-np.cos(x)))/(x**3))
plt.legend([r"$\frac{\sin(x)(1-cos(x))}{x^3}$"])
plt.show()
```



Esempio – cont.



Funzioni: rounding

around(a[, decimals])	Evenly round to the given number of decimals.
round_(a[, decimals])	Round an array to the given number of decimals.
rint(x)	Round elements of the array to the nearest integer.
fix(x)	Round to nearest integer towards zero.
floor(x)	Return the floor of the input, element-wise.
ceil(x)	Return the ceiling of the input, element-wise.
trunc(x)	Return the truncated value of the input, element-wise.

Funzioni: somma, prodotto, differenza

prod(a)	Return the product of array elements over a given axis.
sum(a)	Sum of array elements over a given axis.
nanprod(a)	Return the product of array elements over a given axis treating Not a Numbers (NaNs) as ones.
nansum(a)	Return the sum of array elements over a given axis treating Not a Numbers (NaNs) as zero.
cumprod(a)	Return the cumulative product of elements along a given axis.
cumsum(a)	Return the cumulative sum of the elements along a given axis.
nancumprod(a)	Return the cumulative product of array elements over a given axis treating Not a Numbers (NaNs) as one.
nancumsum(a)	Return the cumulative sum of array elements over a given axis treating Not a Numbers (NaNs) as zero.
diff(a, …)	Calculate the n-th discrete difference along the given axis.
cross(a, b, ...)	Return the cross product of two (arrays of) vectors.
trapz(y, ...)	Integrate along the given axis using the composite trapezoidal rule.

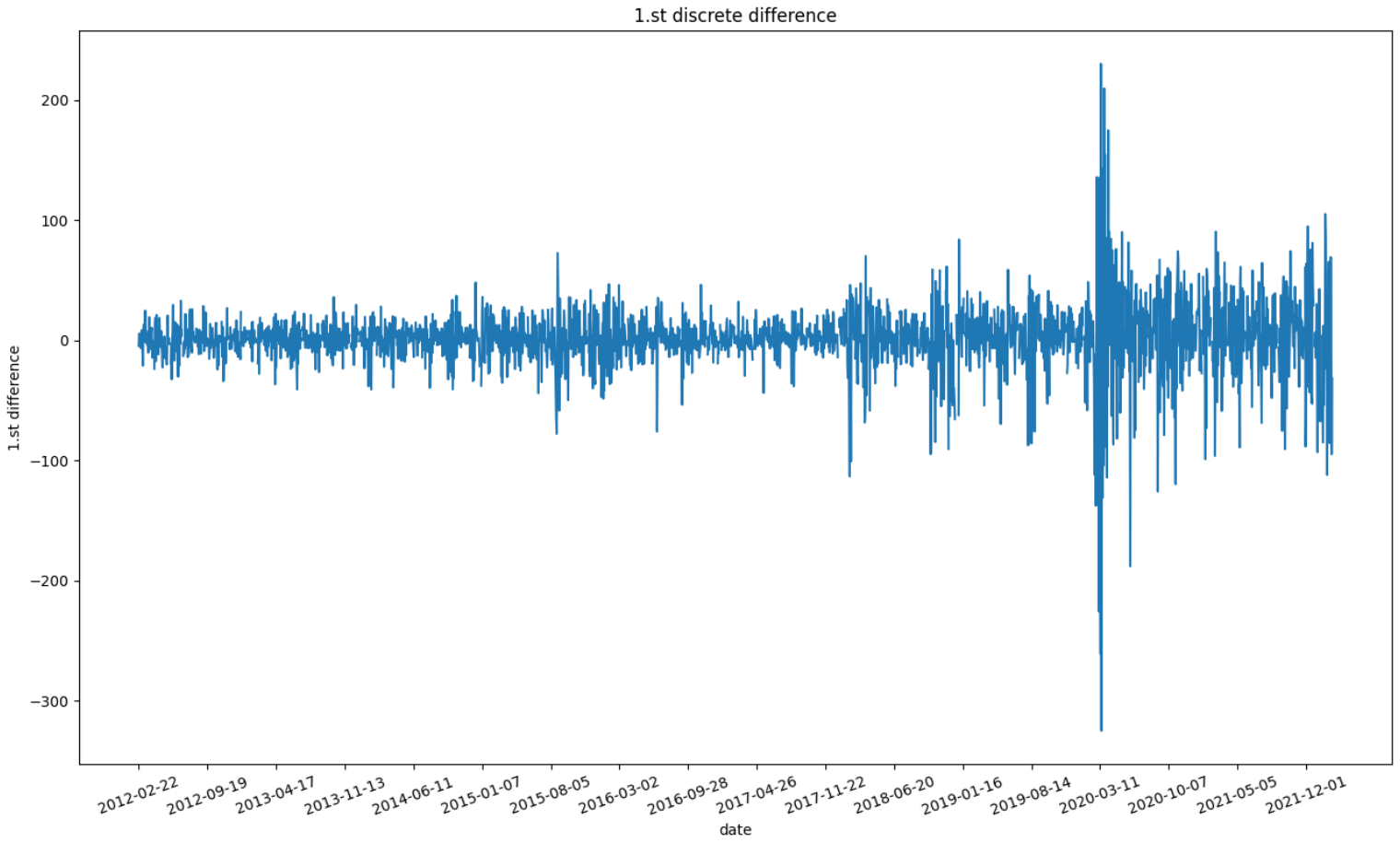
Esempi

```
import numpy as np
import matplotlib.pyplot as plt

dt = np.dtype([('date', 'U10'), ('value', 'f8')])
sp500 = np.genfromtxt('sp500.csv',
                      delimiter=',',
                      dtype=dt,
                      skip_header=1,
                      missing_values={1: "."},
                      filling_values={1: np.nan},
                      names=["Date", "Value"])

plt.figure(figsize=(16, 9))
diff = np.diff(sp500["Value"], 1)
plt.plot(sp500["Date"][1:], diff)
plt.title(r"1.st discrete difference")
plt.xlabel("date")
plt.ylabel("1.st difference")
plt.xticks(range(0, len(sp500), 150), rotation=20)
plt.show()
```

Esempio - 2


1.st discrete difference
1.st difference
200
100
0
−100
−200
−300
2012-02-22
2012-09-19
2013-04-17
2013-11-13
2014-06-11
2015-01-07
2015-08-05
2016-03-02
2016-09-28
2017-04-26
2017-11-22
2018-06-20
2019-01-16
2019-08-14
2020-03-11
2020-10-07
2021-05-05
2021-12-01
date

Funzioni: logaritmiche/esponenziali

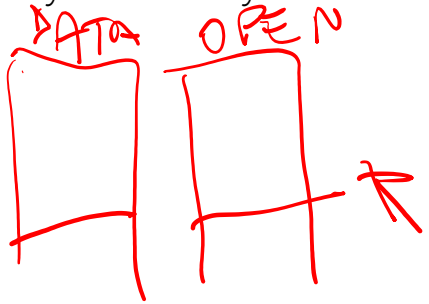
exp(x)	Calculate the exponential of all elements in the input array.
expm1(x)	Calculate exp(x) - 1 for all elements in the array.
exp2(x)	Calculate *2**p* for all *p* in the input array.
log(x)	Natural logarithm, element-wise.
log10(x)	Return the base 10 logarithm of the input array, element-wise.
log2(x)	Base-2 logarithm of *x*.
log1p(x)	Return the natural logarithm of one plus the input array, element-wise.
logaddexp(x1, x2)	Logarithm of the sum of exponentiations of the inputs.
logaddexp2(x1, x2)	Logarithm of the sum of exponentiations of the inputs in base-2.

Funzioni: estremi

maximum(x1, x2)	Element-wise maximum of array elements.
fmax(x1, x2)	Element-wise maximum of array elements.
amax(a)	Return the maximum of an array or maximum along an axis.
nanmax(a)	Return the maximum of an array or maximum along an axis, ignoring any NaNs.
minimum(x1, x2)	Element-wise minimum of array elements.
fmin(x1, x2)	Element-wise minimum of array elements.
amin(a)	Return the minimum of an array or minimum along an axis.
nanmin(a)	Return minimum of an array or minimum along an axis, ignoring any NaNs.

Funzioni: ricerca

argmax(a[, axis, keepdims])	Returns the indices of the maximum values along an axis.
nanargmax(a[, axis, keepdims])	Return the indices of the maximum values in the specified axis ignoring NaNs.
argmin(a[, axis, keepdims])	Returns the indices of the minimum values along an axis.
nanargmin(a[, axis, keepdims])	Return the indices of the minimum values in the specified axis ignoring NaNs.
argwhere(a)	Find the indices of array elements that are non-zero, grouped by element.
nonzero(a)	Return the indices of the elements that are non-zero.
flatnonzero(a)	Return indices that are non-zero in the flattened version of a.
where(condition, [x, y])	Return elements chosen from *x* or *y* depending on *condition*.
searchsorted(a, v[, side, sorter])	Find indices where elements should be inserted to maintain order.
extract(condition, arr)	Return the elements of an array that satisfy some condition.



Esempi

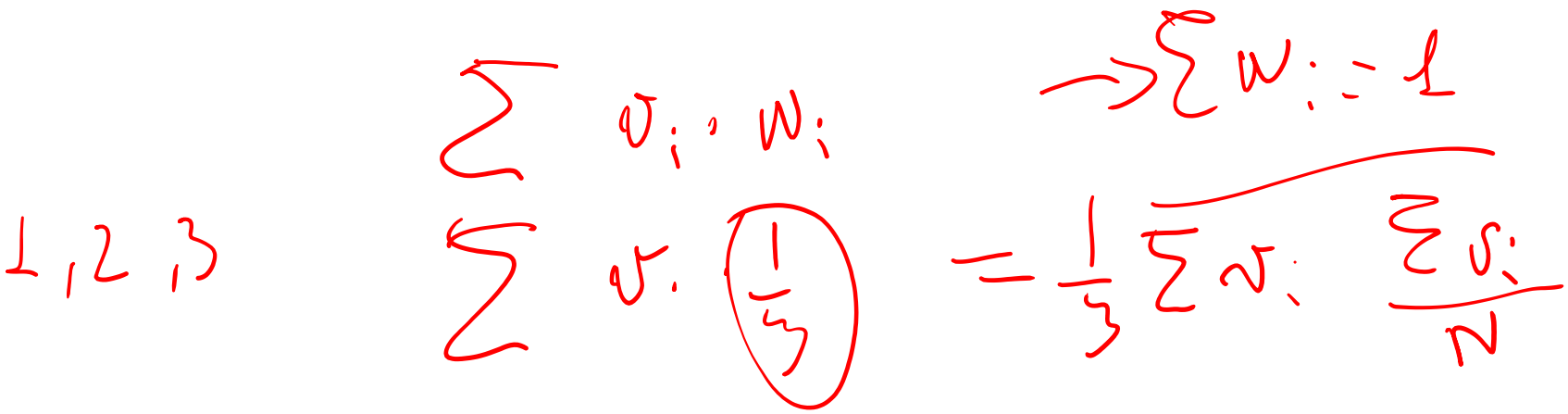
```
vmax = np.nanmax(sp500["Value"])
vmin = np.nanmin(sp500["Value"])
indvmax = np.nanargmax(sp500["Value"])
indvmin = np.nanargmin(sp500["Value"])
dmax = sp500["Date"][indvmax]
dmin = sp500["Date"][indvmin]

print(f"Massimo storico {vmax} il {dmax} ")
print(f"Minimo storico {vmin} il {dmin} ")
```

```
Massimo storico 4796.56 il 2022-01-03
Minimo storico 1278.04 il 2012-06-01
```

Funzioni: Statistiche - 1

median(a[, axis])	Compute the median along the specified axis.
average(a[, axis, weights])	Compute the weighted average along the specified axis.
mean(a[, axis, dtype])	Compute the arithmetic mean along the specified axis.
std(a[, axis, dtype])	Compute the standard deviation along the specified axis.
var(a[, axis, dtype])	Compute the variance along the specified axis.
nanmedian(a[, axis])	Compute the median along the specified axis, while ignoring NaNs.
nanmean(a[, axis])	Compute the arithmetic mean along the specified axis, ignoring NaNs.
nanstd(a[, axis])	Compute the standard deviation along the specified axis, while ignoring NaNs.
nanvar(a[, axis])	Compute the variance along the specified axis, while ignoring NaNs.



Esempio

```
print(np.min(wh['height']),
      np.mean(wh['height']),
      np.max(wh['height']))
male = wh['gender'] == 'Male'
female = wh['gender'] == 'Female'

print(np.min(wh[male]['height']),
      np.mean(wh[male]['height']),
      np.max(wh[male]['height']))
print(np.min(wh[female]['height']),
      np.mean(wh[female]['height']),
      np.max(wh[female]['height']))
```

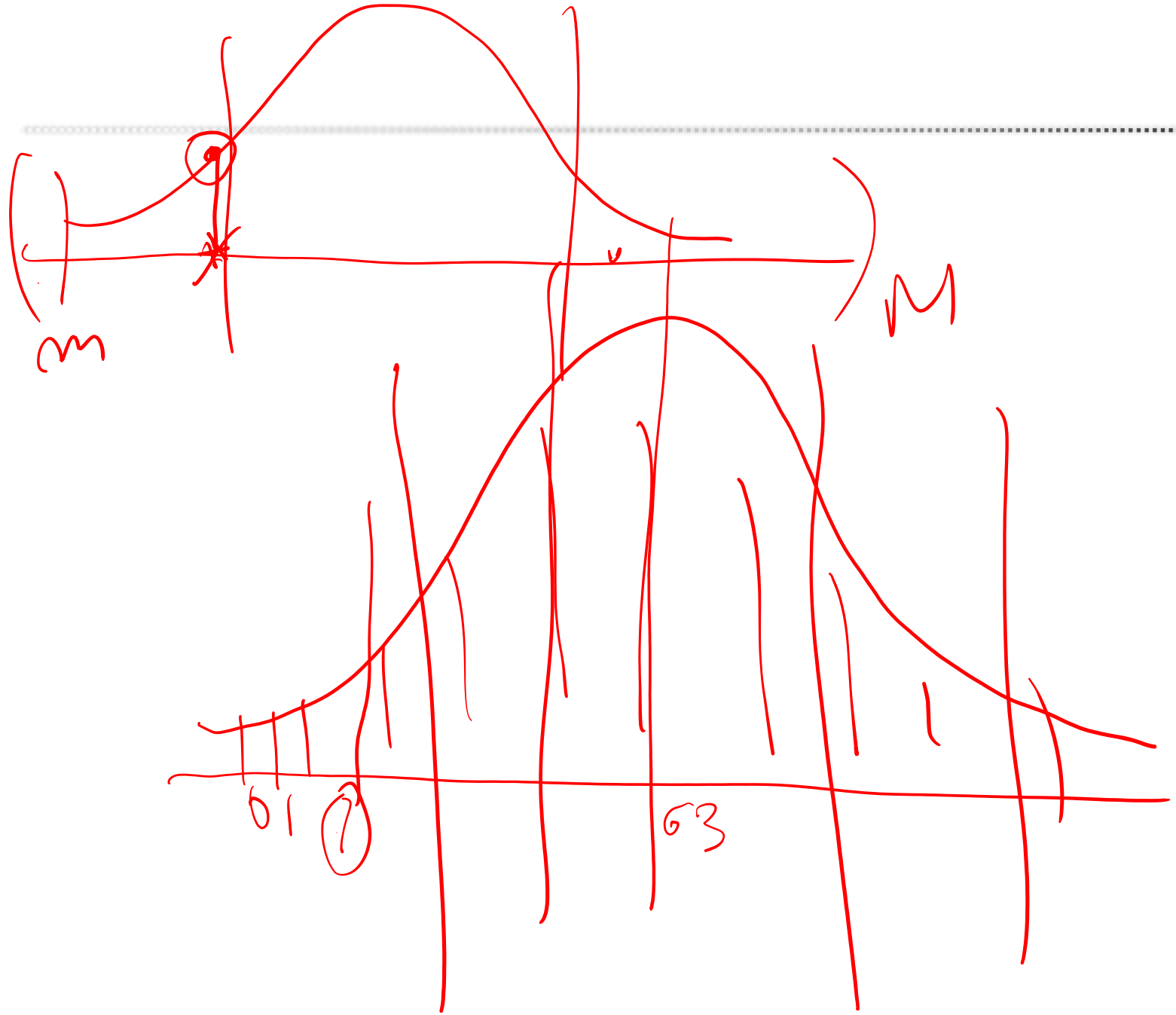
Esempio – cont.

```
1.3782835864574663 1.685736017724594 2.006568055598296
1.4835353852664448 1.7532691860179221 2.006568055598296
1.3782835864574663 1.6182028495269967 1.864095480998170
```

Funzioni: Statistiche - 2

ptp(a[, axis, out, keepdims])	Range of values (maximum - minimum) along an axis.
percentile(a, q[, axis])	Compute the q-th percentile of the data along the specified axis.
nanpercentile(a, q[, axis])	Compute the qth percentile of the data along the specified axis, while ignoring nan values.
quantile(a, q[, axis])	Compute the q-th quantile of the data along the specified axis.
nanquantile(a, q[, axis)	Compute the qth quantile of the data along the specified axis, while ignoring nan values.

corrcoef(x[, y, rowvar, bias, ddof, dtype])	Return Pearson product-moment correlation coefficients.
correlate(a, v[, mode])	Cross-correlation of two 1-dimensional sequences.
cov(m[, y, rowvar, bias, ddof, fweights, ...])	Estimate a covariance matrix, given data and weights.

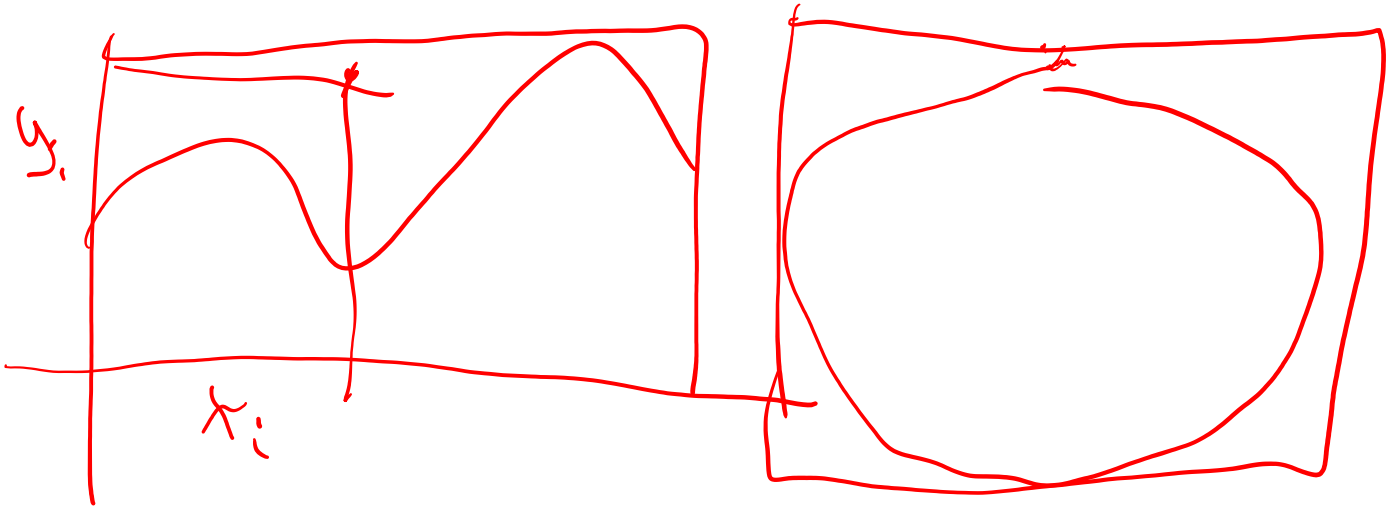

m
M
v
σ1
σ3

Funzioni: Statistiche - 3

histogram(a[, bins, range, normed, weights, ...])	Compute the histogram of a dataset.
histogram2d(x, y[, bins, range, normed, ...])	Compute the bi-dimensional histogram of two data samples.
histogramdd(sample[, bins, range, normed, ...])	Compute the multidimensional histogram of some data.
bincount(x, /[, weights, minlength])	Count number of occurrences of each value in array of non-negative ints.
histogram_bin_edges(a[, bins, range, weights])	Function to calculate only the edges of the bins used by the histogram function.
digitize(x, bins[, right])	Return the indices of the bins to which each value in input array belongs.

Funzioni: random

- `random` è un sotto modulo di `numpy`
- Per accedere, quindi, alle funzioni presenti nel modulo `random` di `numpy` bisogna preporre la stringa `np.random`
- Buona norma, inoltre, usare la classe `Generator` per creare una nuova istanza del generatore (ma non necessario)
- Esistono vari generatori di numeri casuali ma il *Mersenne Twister* è il generatore di default e per i nostri scopi va benissimo



Funzioni: random

integers(low[, high, size, dtype, endpoint])	Return random integers from *low* (inclusive) to *high* (exclusive), or if endpoint=True, *low* (inclusive) to *high* (inclusive).
random([size, dtype, out])	Return random floats in the half-open interval [0.0, 1.0).
choice(a[, size, replace, p, axis, shuffle])	Generates a random sample from a given array
bytes(length)	Return random bytes.

shuffle(x[, axis])	Modify an array or sequence in-place by shuffling its contents.
permutation(x[, axis])	Randomly permute a sequence or return a permuted range.
permuted(x[, axis])	Randomly permute *x* along axis *axis*.

Funzioni: random - 2

beta(a, b[, size])	Draw samples from a Beta distribution.
binomial(n, p[, size])	Draw samples from a binomial distribution.
chisquare(df[, size])	Draw samples from a chi-square distribution.
dirichlet(alpha[, size])	Draw samples from the Dirichlet distribution.
exponential([scale, size])	Draw samples from an exponential distribution.
f(dfnum, dfden[, size])	Draw samples from an F distribution.
gamma(shape[, scale, size])	Draw samples from a Gamma distribution.
geometric(p[, size])	Draw samples from the geometric distribution.
gumbel([loc, scale, size])	Draw samples from a Gumbel distribution.
hypergeometric(ngood, nbad, nsample[, size])	Draw samples from a Hypergeometric distribution.

Funzioni: random - 3

laplace([loc, scale, size])	Draw samples from the Laplace or double exponential distribution with specified location (or mean) and scale (decay).
logistic([loc, scale, size])	Draw samples from a logistic distribution.
lognormal([mean, sigma, size])	Draw samples from a log-normal distribution.
logseries(p[, size])	Draw samples from a logarithmic series distribution.
multinomial(n, pvals[, size])	Draw samples from a multinomial distribution.
multivariate_hypergeometric(colors, nsample)	Generate variates from a multivariate hypergeometric distribution.
multivariate_normal(mean, cov[, size, ...])	Draw random samples from a multivariate normal distribution.

Funzioni: random - 4

negative_binomial(n, p[, size])	Draw samples from a negative binomial distribution.
noncentral_chisquare(df, nonc[, size])	Draw samples from a noncentral chi-square distribution.
noncentral_f(dfnum, dfden, nonc[, size])	Draw samples from the noncentral F distribution.
normal([loc, scale, size])	Draw random samples from a normal (Gaussian) distribution.
pareto(a[, size])	Draw samples from a Pareto II or Lomax distribution with specified shape.
poisson([lam, size])	Draw samples from a Poisson distribution.
power(a[, size])	Draws samples in [0, 1] from a power distribution with positive exponent a - 1.

Esempio

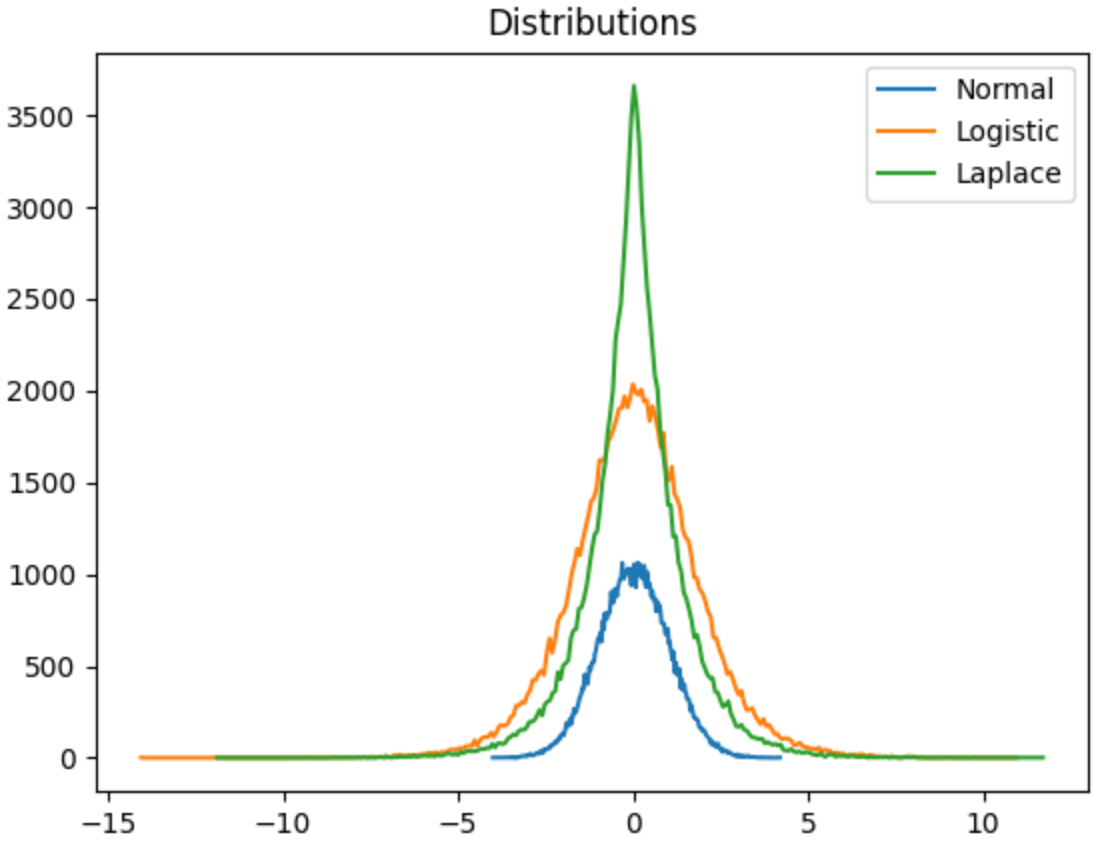
```
import numpy as np
import matplotlib.pyplot as plt

how_many = 100000
norm = np.random.normal(0, 1, how_many)
logi = np.random.logistic(0, 1, how_many)
lapl = np.random.laplace(0, 1, how_many)
hnorm, nint = np.histogram(norm, bins=int(np.sqrt(how_many)))
hlogi, hint = np.histogram(logi, bins=int(np.sqrt(how_many)))
hlapl, lapl = np.histogram(lapl, bins=int(np.sqrt(how_many)))

plt.plot(nint[1:], hnorm)
plt.plot(hint[1:], hlogi)
plt.plot(lapl[1:], hlapl)

plt.title("Distributions")
plt.legend(["Normal", "Logistic", "Laplace"])
plt.show()
```

Esempio – cont.



Funzioni: random - 5

rayleigh([scale, size])	Draw samples from a Rayleigh distribution.
standard_cauchy([size])	Draw samples from a standard Cauchy distribution with mode = 0.
standard_exponential([size)	Draw samples from the standard exponential distribution.
standard_gamma(shape[, size])	Draw samples from a standard Gamma distribution.
standard_normal([size, dtype])	Draw samples from a standard Normal distribution (mean=0, stdev=1).
standard_t(df[, size])	Draw samples from a standard Student's t distribution with *df* degrees of freedom.
triangular(left, mode, right[, size])	Draw samples from the triangular distribution over the interval [left, right].
uniform([low, high, size])	Draw samples from a uniform distribution.
vonmises(mu, kappa[, size])	Draw samples from a von Mises distribution.
wald(mean, scale[, size])	Draw samples from a Wald, or inverse Gaussian, distribution.
weibull(a[, size])	Draw samples from a Weibull distribution.
zipf(a[, size])	Draw samples from a Zipf distribution.