Nested Sampling con Python

Calcolo dell’evidenza usando come likelihood una distribuzione multivariata normale di dimensione arbitraria

Federico V. Mastellone, 583122

Evidenza

Stima dei parametri di un modello

Supponiamo di dover stimare i parametri $\vec{\theta}_1$ di un modello M_1 avendo raccolto un set di dati d, si definiscono le seguenti quantità:

1. *Prior probability* $\longrightarrow P\left(\vec{\theta}_1 | M_1\right)$
2. *Posterior probability* $\longrightarrow P\left(\vec{\theta}_1 | d, M_1\right)$
3. *Likelihood* $\longrightarrow P\left(d | \vec{\theta}_1, M_1\right)$
4. *Evidence* $P\left(d | M_1\right) = \int_{\Theta_1} d\vec{\theta}_1 P\left(d | \vec{\theta}_1, M_1\right) P\left(\vec{\theta}_1 | M_1\right)$

$$P\left(\vec{\theta}_1 | d, M_1\right) = \frac{P\left(d | \vec{\theta}_1, M_1\right) P\left(\vec{\theta}_1 | M_1\right)}{P\left(d | M_1\right)}$$

Evidenza

Convalida di un modello

Cosa succede se ho due modelli $M_1\left(\vec{\theta}_1\right)$ e $M_2\left(\vec{\theta}_2\right)$? Quale dei due è più plausibile in base ai dati che ho raccolto?
Interviene il *fattore di Bayes*:

$$F = \frac{P\left(d|M_1\right)}{P\left(d|M_2\right)} = \frac{\int_{\Theta_1} d\vec{\theta}_1 P\left(d|\vec{\theta}_1, M_1\right) P\left(\vec{\theta}_1|M_1\right)}{\int_{\Theta_2} d\vec{\theta}_2 P\left(d|\vec{\theta}_2, M_2\right) P\left(\vec{\theta}_2|M_2\right)}$$

Se $F > 1$ allora M_1 è più plausibile. Compare quindi ancora una volta l'evidenza Z, come può essere riscritta?

$$Z = \int_{\Theta} d\vec{\theta} P\left(d|\vec{\theta}, M\right) P\left(\vec{\theta}|M\right) = \int_{\Theta} \mathscr{L}(\vec{\theta})\pi(\vec{\theta})d\vec{\theta}$$

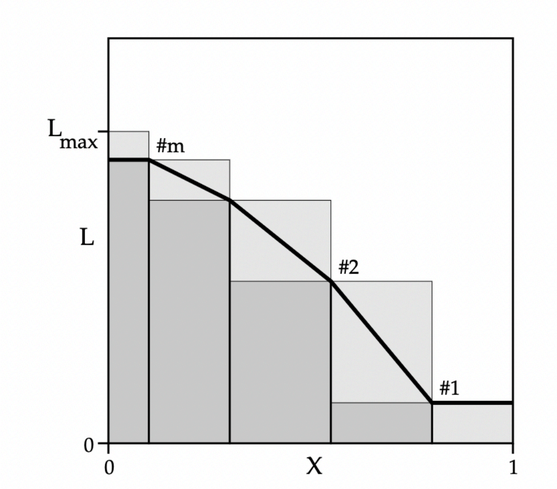
Dove con $\mathscr{L}(\vec{\theta})$ si intende la *likelihood* e con $\pi(\vec{\theta})$ si intende la *prior probability*.

Prior mass

Si definisce quindi la *Prior mass* come: $X(\lambda) = \int_{\mathscr{L}(\vec{\theta})>\lambda} \pi(\vec{\theta}) d\vec{\theta} \Rightarrow X(\lambda) : \mathbb{R} \rightarrow [0,1]$

Scegliendo due valori $\lambda_1 > \lambda_2$ si ha che $X(\lambda_1) < X(\lambda_2)$

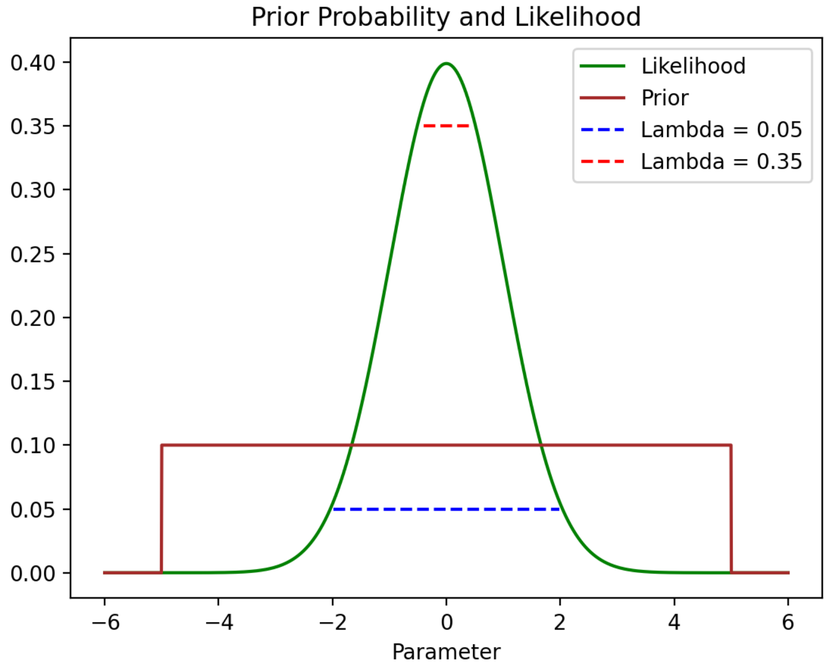
Possiamo quindi riscrivere $Z \longrightarrow Z = \int_0^1 \mathscr{L}(X) dX$



2. Plot di $\mathscr{L}(X)$

Dove $\mathscr{L}(X)$ è monotona decrescente!
Riscrivendo l'integrale utilizzando le somme di Riemann si ottiene:

$$Z \approx \sum_{i=1}^{m} (X_{i-1} - X_i)\mathscr{L}_i(X_i)$$



1. Likelihood e Prior per uno spazio dei parametri unidimensionale

Stima dell'evidenza

La (non) necessaria conoscenza di X_i

Se conosco un set di punti X_i della prior mass allora è semplice valutare la somma si Riemann. Conoscere un set di punti X_i, tuttavia, può non essere semplice. Occorre una stima statistica.

Cosa posso fare?

1. Il valore di Z sarà dominato dalla regione dove si concentra la *posterior mass*, questa regione è una frazione della regione occupata dalla *prior mass* $X(\lambda)$. Questa frazione è determinata dall'informazione $H = \int \ln\left(dP/dX\right) dP \rightarrow e^{-H}$

   H misura quindi il fattore di compressione da prior a posterior e può assumere valori molto elevati, ci si aspetta quindi che la maggior parte della *posterior mass* verrà trovata per piccoli valori di X.

2. È quindi necessaria una stima uniforme in $\ln X \longrightarrow X_m = \prod_{i=1}^{m} t_i$ con $t_i \in U(0,1)$

C'è un modo però per stimare X_i senza conoscerlo!

Stima dell'evidenza

Partendo dai parametri $\vec{\theta}$

1. Supponiamo di partire dal valore della likelihood $\mathscr{L}_0 = 0$ e consideriamo un punto $\vec{\theta}_c \in \pi(\vec{\theta}) \,|\, \mathscr{L}(\vec{\theta}_c) > \mathscr{L}_0$ allora, essendo $\mathscr{L}(X)$ monotòna decrescente so già di aver trovato un punto $X_i < X_0 \,|\, X_i = t_i X_0 \leftarrow t_i \in U(0,1)$.

2. Supponendo di poter prendere N punti $\vec{\theta}_i$ campionati dalla $\pi(\vec{\theta})$ tali da generare un set ordinato di coppie $\{\mathscr{L}(\vec{\theta}), \vec{\theta}\}$, qual è la probabilità di rimpicciolire la *prior mass* di un certo t_i nel momento in cui vado a generare un $\bar{\vec{\theta}}$ tale da restituire un valore della *likelihood* che sia $\mathscr{L}(\vec{\theta})_{worst} < \mathscr{L}(\bar{\vec{\theta}}) < \mathscr{L}(\vec{\theta}_i)\ \forall i \in [0,N]$? È $P(t_i) = N t_i^{N-1}$.

3. Posso quindi andare a definire il *valor medio* di $\ln t$ come $< \ln t > = -1/N$, se ripeto questo iter per i volte allora mi aspetto che la *prior mass* si rimpicciolisca fino a $\ln X_i \approx -i/N$. Ho quindi trovato un modo per stimare il logaritmo naturale dei punti X_i e questo mi consente di andare a calcolare ogni volta ΔX

Posso quindi stimare $\ln Z$ sfruttando la somma di Riemann vista nelle slide 4
$\longrightarrow \ln Z_{i+1} = \ln(\exp(\ln Z_i) + \exp(\ln(\mathscr{L}_i) + \ln(\Delta X_i))$

Stima dell'evidenza

L'errore su $\ln Z$

1. Ad ogni iterazione incrementa l'errore sulla stima di $\ln Z$, questo errore è dominante negli step necessari a raggiungere la *posterior* ed è legato al rapporto tra l'information H e il numero di punti estratti dalla *prior,* N.

$$\longrightarrow \ln Z \pm \sqrt{\frac{H}{N}}$$

2. Come riscrivere H in modo da implementarla nel codice?

$$H = \int_0^1 dX \frac{\mathscr{L}(X)}{Z} \ln\left(\frac{\mathscr{L}(X)}{Z}\right)$$

Da cui si ricava il valore di H allo step $i+1 \longrightarrow H_{i+1} = H_i + \frac{L_i \Delta X_i}{Z_i}(\ln L_i - \ln Z_i)$ convenientemente espresso in logaritmi in tal modo:

$$\ln H_{i+1} = \ln\{\exp(\ln H_i) + \exp(\ln L_i + \ln \Delta X_i - \ln Z_i + \ln(\ln L_i - \ln Z_i))\}$$

Stima dell’evidenza

L’algoritmo

Si delinea quindi il seguente algoritmo:

I. Inizializza i valori di $\ln Z$ e di $\ln H$

II. Inizializza un set di N punti da $\pi(\overrightarrow{\theta})$

III. Itera i seguenti step:

1. Calcola i valori della $\ln \mathscr{L}$ da questi punti e trovane il minimo

2. Genera un nuovo punto dalla $\pi(\overrightarrow{\theta})$

2.1. Se la $\ln \mathscr{L}$ calcolata in questo punto è maggiore della più piccola $\ln \mathscr{L}$ presente nel set allora aggiorna il valore di $\ln \Delta X$ e successivamente quello di $\ln Z$ e di $\ln H$

2.2. Sostituisci al punto di minimo per $\ln \mathscr{L}(\overrightarrow{\theta})$ il punto precedentemente generato dalla $\pi(\overrightarrow{\theta})$

Quando fermare il ciclo?
Quando l’incremento su $\ln Z$ dato dalla più grande $\ln \mathscr{L}$ tra quelle calcolate è minore di una frazione $\ln f$:

$$\ln(\max(\mathscr{L}_i)) - \ln(Z_i) < \ln f + \frac{i+\sqrt{i}}{N}$$

Stima dell’evidenza

Proposal functions per il sampling su $\pi(\vec{\theta})$

Durante l’esecuzione dell’algoritmo è necessario generare un nuovo punto dalla prior distribution $\pi(\vec{\theta})$.
Questo processo viene diviso in due step:

1. Si parte dal punto $\vec{\theta}'$ tale per cui $\mathscr{L}(\vec{\theta}')$ sia un minimo di $\mathscr{L}(\vec{\theta})$. Questo punto viene usato come parametro che definisce la media della distribuzione di probabilità da usare per generare un punto $\vec{\theta}''$ nei dintorni di $\vec{\theta}'$.
   Si usa invece come parametro di scaling della suddetta distribuzione il termine $k\sigma$ dove σ rappresenta la deviazione standard calcolata sugli N punti del sample di partenza da $\pi(\vec{\theta})$ mentre k è un fattore moltiplicativo per ampliare o ridurre il range di ricerca.

2. Dopo aver generato il punto $\vec{\theta}''$ si verifica che quest’ultimo non oltrepassi i boundaries definiti dalla prior distribution $\pi(\vec{\theta})$

Nel codice presentato in seguito vengono usate principalmente due distribuzioni continue del pacchetto SciPy:

A. Anglit

B. Semicircular

Distribuzioni Semicircular, Anglit e Normal

Confronto con una Gaussiana in una dimensione

A. Normal distribution:

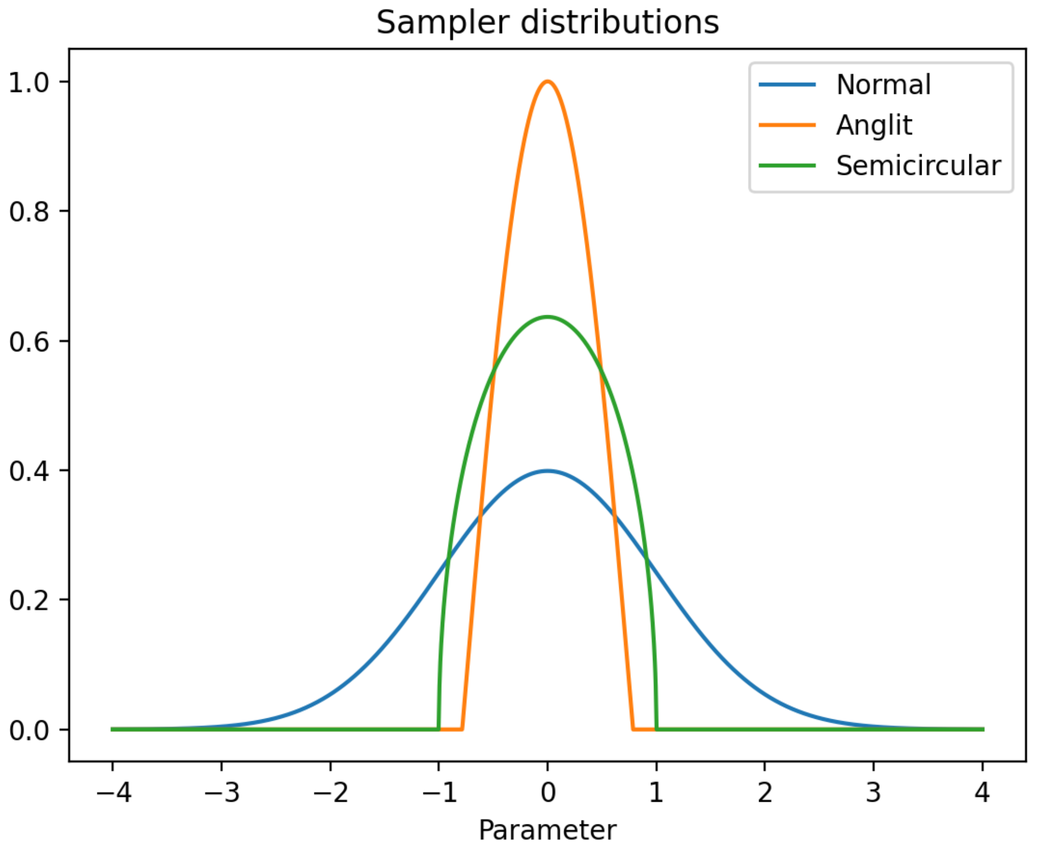
$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

B. Anglit distribution:

$$p(x) = \cos\left(2\frac{x-\mu}{\sigma}\right) = \cos(2y(x)) \leftarrow y \in \left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$$

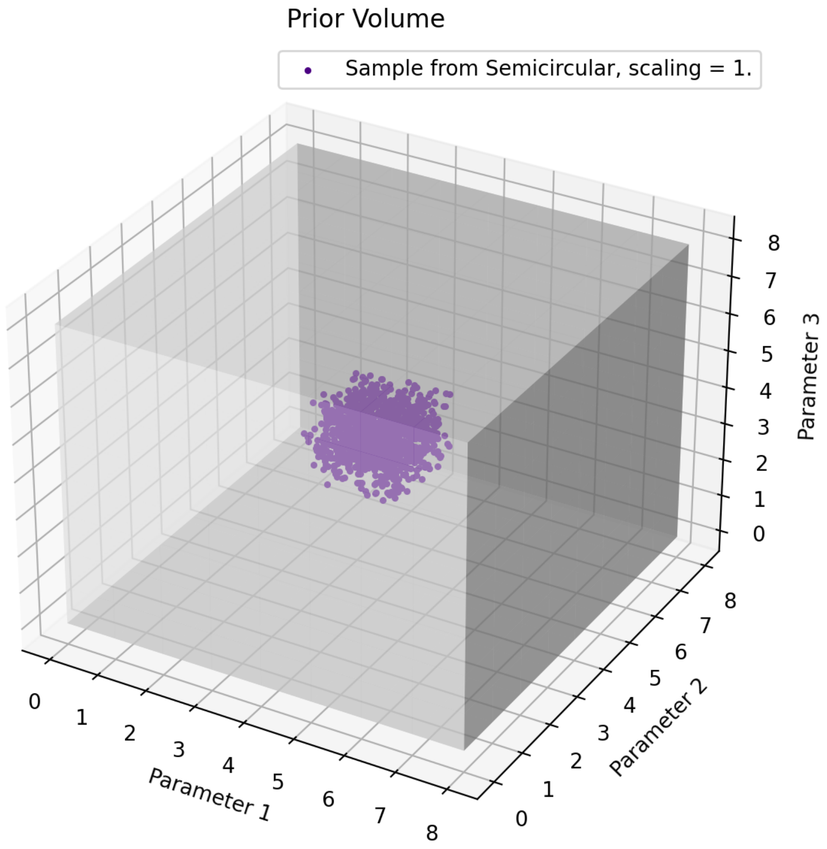
C. Semicircular distribution:

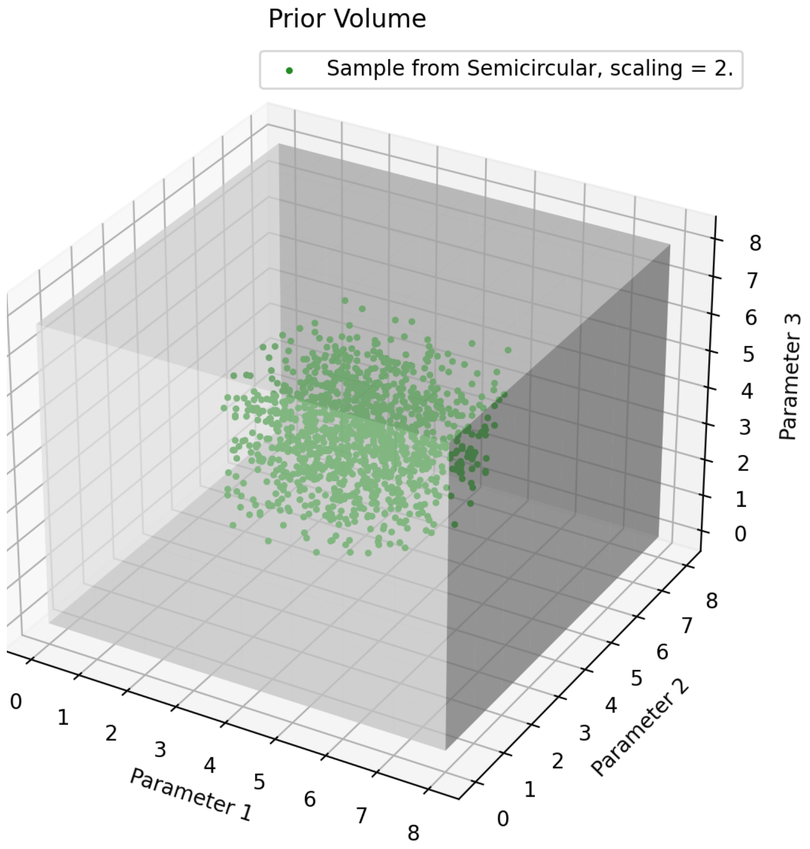
$$p(x) = \frac{2}{\pi}\sqrt{1 - \frac{(x-\mu)^2}{\sigma^2}} = \frac{2}{\pi}\sqrt{1 - y(x)^2} \leftarrow y \in [-1,1]$$



Distribuzioni Semicircular, Anglit e Normal

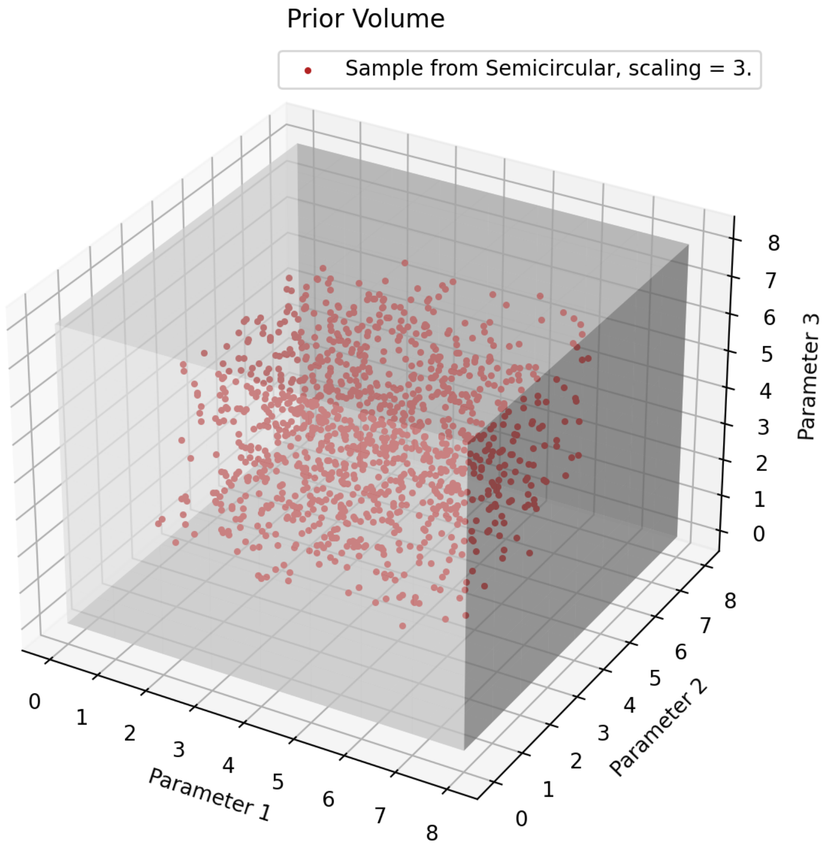
Semicircular in uno spazio dei parametri 3D

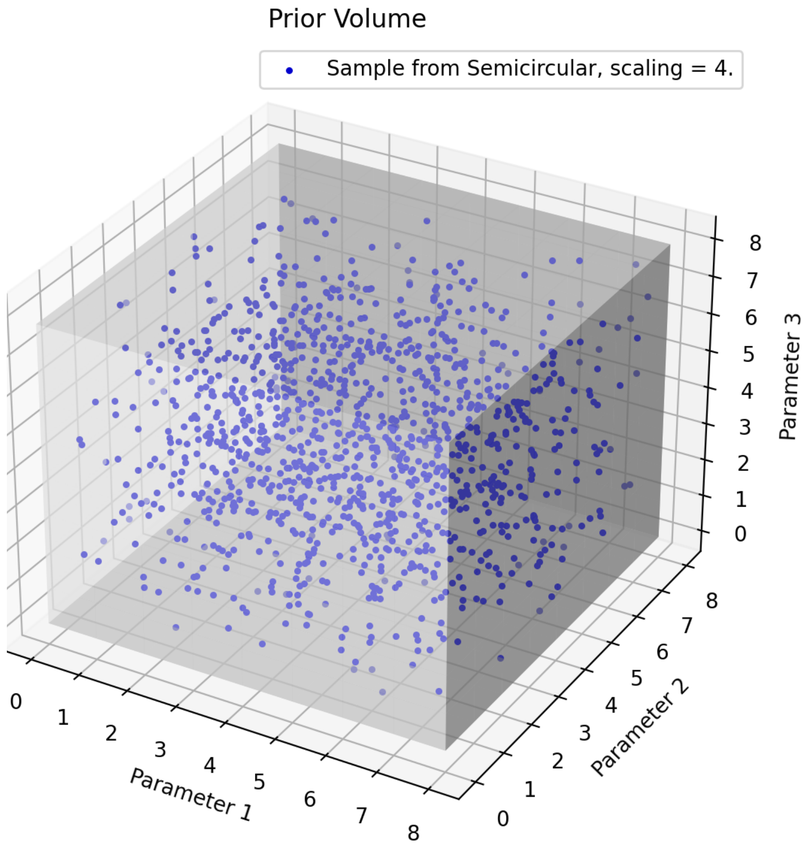




Distribuzioni Semicircular, Anglit e Normal

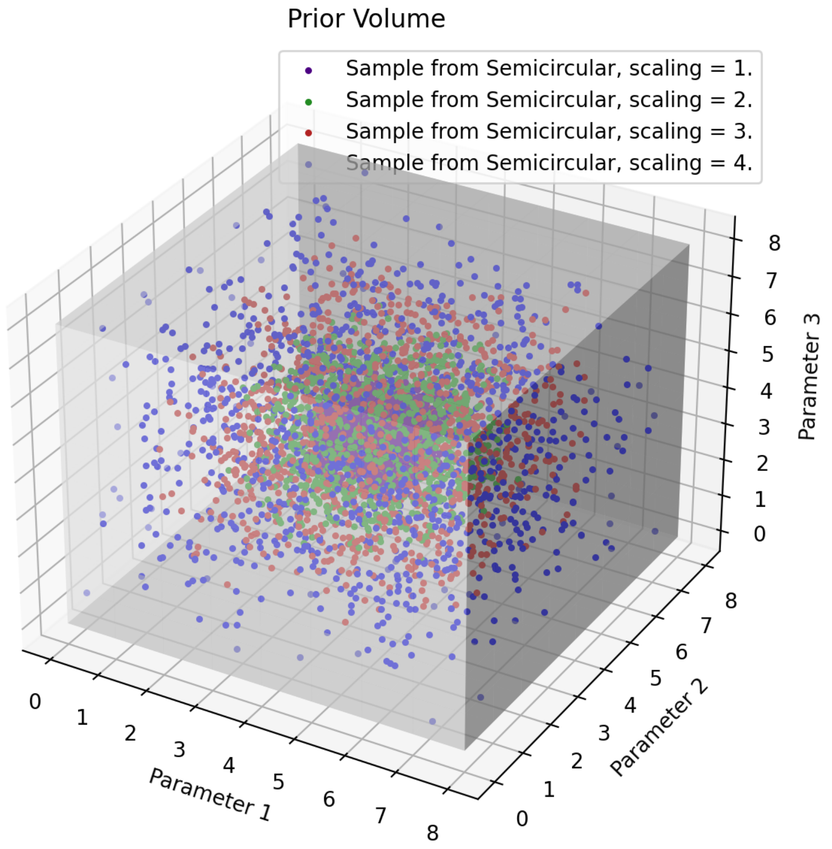
Semicircular in uno spazio dei parametri 3D





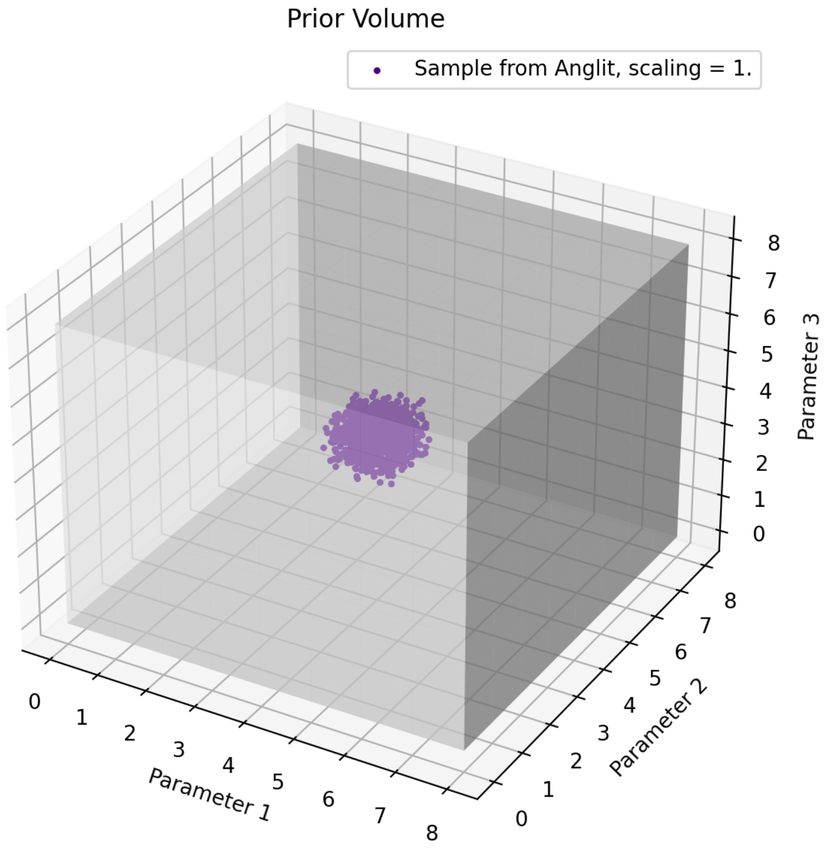
Distribuzioni Semicircular, Anglit e Normal

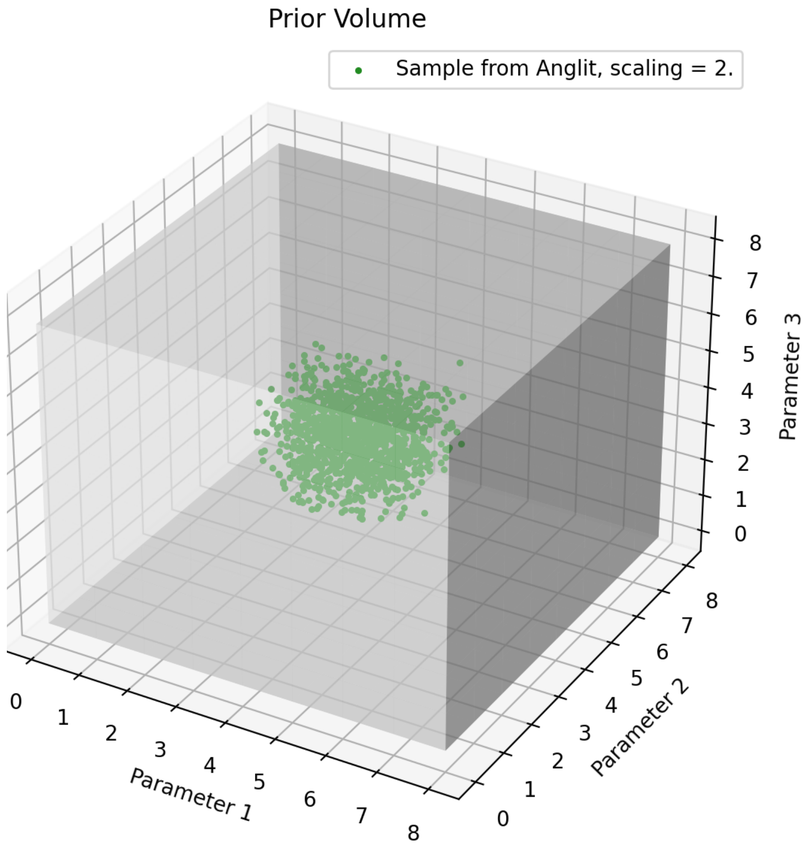
Semicircular in uno spazio dei parametri 3D



Distribuzioni Semicircular, Anglit e Normal

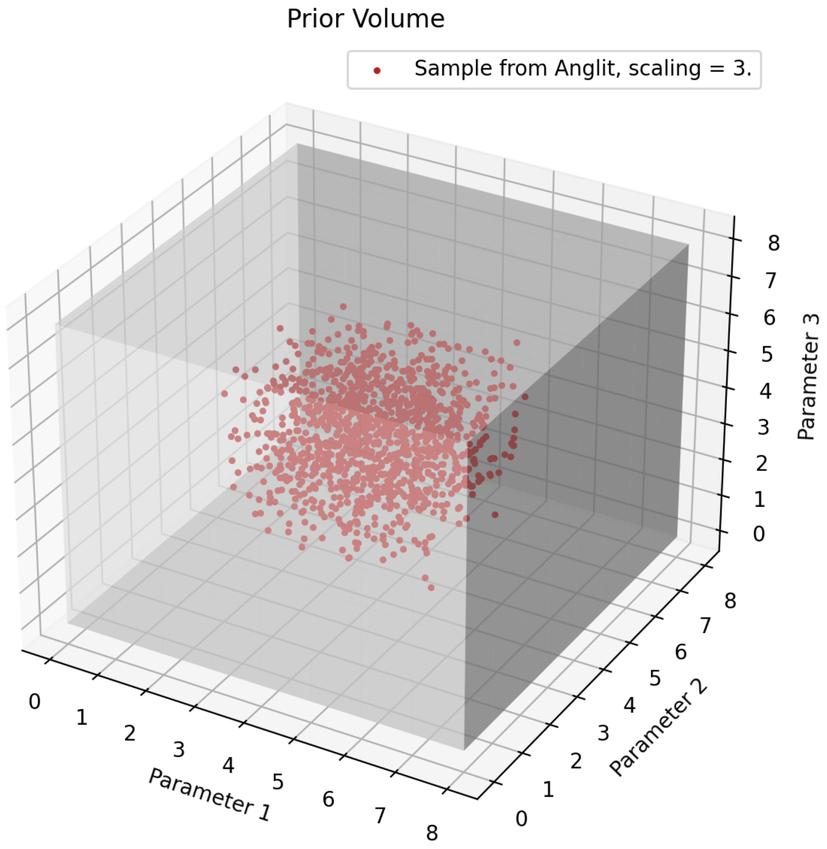
Anglit in uno spazio dei parametri 3D

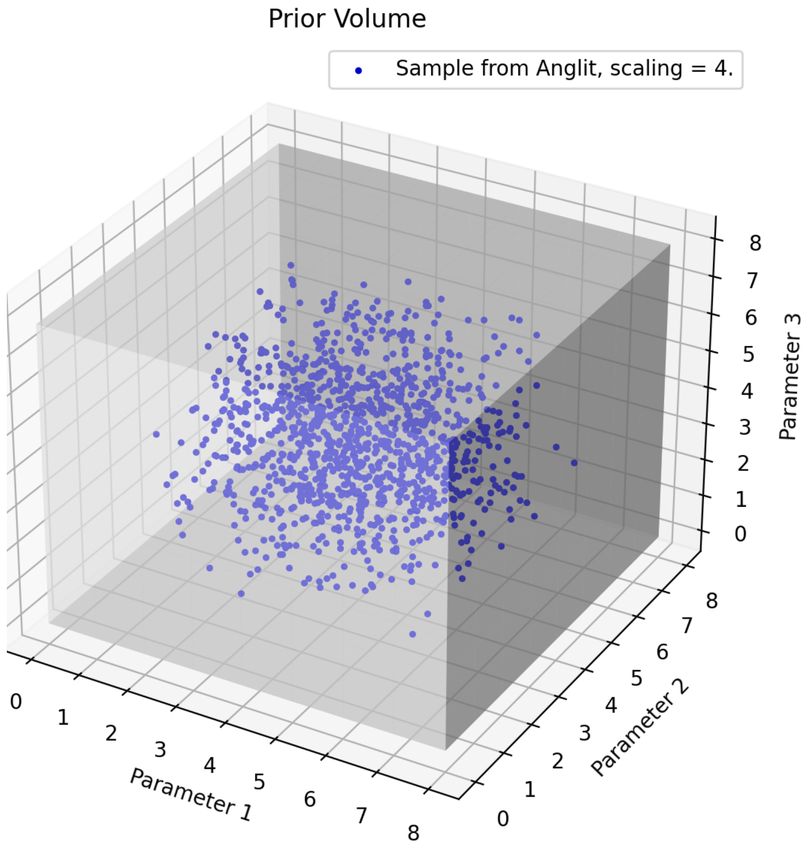




Distribuzioni Semicircular, Anglit e Normal

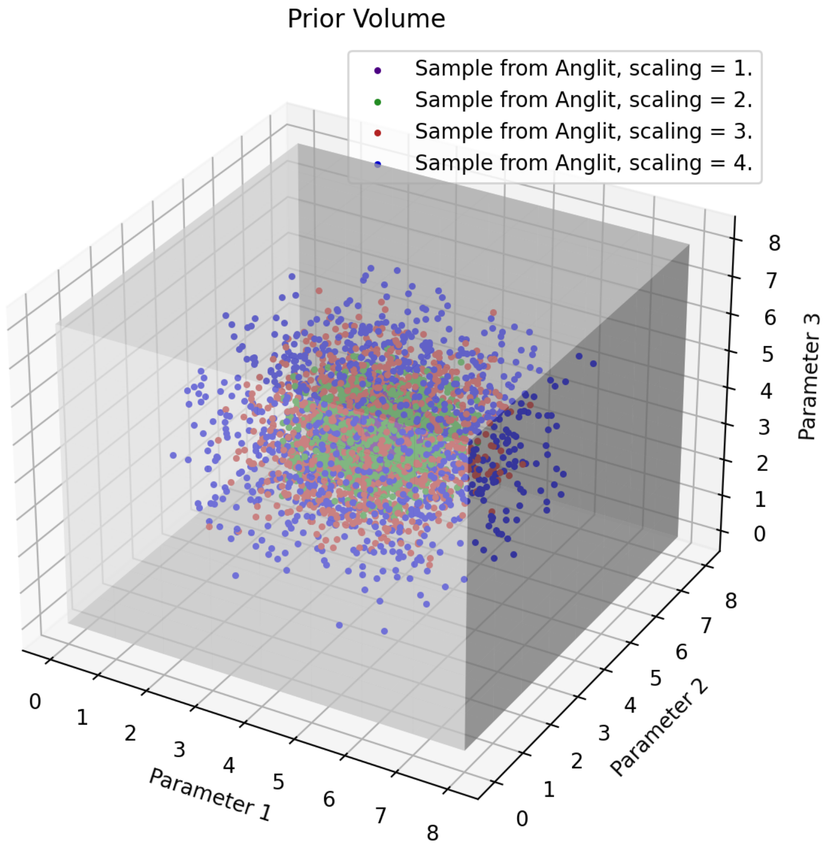
Anglit in uno spazio dei parametri 3D





Distribuzioni Semicircular, Anglit e Normal

Anglit in uno spazio dei parametri 3D



Struttura del codice

Calcolo del punto della *log likelihood*

```python
def log_multivariate_point(array, dimensions):
    """
    Given a set of parameters, calculates the logarithm of the
    multivariate normal distribution.

    Args:
        array (float): array of parameters' values
        dimensions (int): dimension of the parameters' space

    Returns:
        point (float): the logarithm of the multivariate normal distribution given the initial array
    """
    mean_array = np.zeros(shape=dimensions, dtype=float)
    covariance_matrix = np.identity(dimensions)
    exponent = np.dot((array - mean_array), np.dot(covariance_matrix, (array - mean_array)))
    point = - 0.5*exponent - (dimensions/2)*np.log(2*np.pi)
    return point
```

Struttura del codice

Proposal function

```python
def proposal(generator, distrib_function,  dimensions, mult_par, loc_value, parameters):
    """
    Generates a n-dimensional point given any continous distribution from the SciPy package.
    The mean value of the distribution can be chosen to be any and the scaling parameter is
    defined as the standard deviation of a set of points already existing in the paramters space
    times a multiplicative parameter.

    Args:
        generator: None or Generator to block the random generator seed
        distrib_function: one of the continous distributions from the SciPy package
        dimensions (int): the dimensions of the parameters' space
        mult_par (float): the multiplicative parameter
        loc_value (array or float): the mean value of the chosen distribution
        parameters (array): set of points in the parameters space

    Returns:
        proposed_point (array): the point generated from the chosen distribution
    """
    scaling = mult_par*np.std(parameters)
    proposed_point = distrib_function.rvs(loc=loc_value, scale=scaling, size=(1,dimensions), random_state=generator)
    return proposed_point
```

Struttura del codice

Prior sampling

```python
def prior_uniform_sampling(generator, dimensions, parameters, mult_par, loc_value, lb_val, hb_val, distrib_function):
    """
    Using the proposal function, generates a point and checks if it stays within the
    boundaries of the hypercube representing the prior volume. If it does, returns such
    point, if it doesn't generates a new point and do the check again.

    Args:
        generator: None or Generator to block the random generator seed
        dimensions (int): the dimensions of the parameters' space
        parameters (array): set of points in the parameters space
        mult_par (float): see the proposal function documentation
        loc_value (array): see the proposal function documentation
        lb_val (float): the low boundary value of the hypercube, e.g. lb_val = -1. --> x = [-1., -1., ..., -1.]
        hb_val (float): the high boundary value of the hypercube, e.g. hb_val = 1. --> x = [1., 1., ..., 1.]
        distrib_function: see the proposal function documentation
    """
    new_point = proposal(generator, distrib_function, dimensions, mult_par, loc_value, parameters)

    # Check if the proposed point is in the boundary
    cond1 = any(axis > hb_val for axis in new_point[0])
    cond2 = any(axis < lb_val for axis in new_point[0])

    while (cond1 or cond2) == True:
        # print("Out of boundary")
        new_point = proposal(generator, distrib_function, dimensions, mult_par, loc_value, parameters)
        cond1 = any(axis > hb_val for axis in new_point[0])
        cond2 = any(axis < lb_val for axis in new_point[0])

    return new_point
```

Struttura del codice

Calcolo di $\min(\mathscr{L}_i)$

```python
def lowest_likelihood_point(log_like_points):
    """
    After calculating the log_multivariate_points checks which one is the lowest.

    Args:
        log_like_points (array): the log_multivariate_points

    Returns:
        min_index (int): the index "i" such that log_multivariate_point(parameters[i], dimensions) is the lowest value
        min_likelihood_value (float): the aforementioned lowest value
    """
    min_index = np.argmin(log_like_points)
    min_likelihood_value = log_like_points[min_index]
    return min_index, min_likelihood_value
```

Struttura del codice

Control Board per dim = 1

```python
def update_controlboard(parameters, min_index, log_mult_points, mult_param):
    """
    Updates the content of the Control Board during the code execution.

    Args:
        parameters (array): set of points in the parameters space
        min_index (int): the index "i" such that log_multivariate_point(parameters[i], dimensions) is the lowest value
        log_mult_points (array): the set of log_mult_points calulated using the log_multivariate_point function
        mult_par (float): the multiplicative parameter needed to plot the proposal function
    """
    plt.suptitle("Control board")

    line, = axs[0].plot(parameters, np.exp(log_mult_points), 'x', ms=3.5, label="L values")
    vertline = axs[0].axvline(parameters[min_index], color='r', ls='--', label='Min L value')
    axs[0].set_title("Likelihood")
    axs[0].legend(loc=8)

    line = axs[1].hist(parameters, bins=15)
    scal = mult_param*np.std(parameters)
    x = np.linspace(min(parameters), max(parameters), 2000)
    sampler = proposal_function.pdf(x, loc=parameters[min_index], scale=scal)
    line, = axs[1].plot(x, sampler, label="Proposal")
    axs[1].legend(loc=1)
    axs[1].set_xlabel("Parameters")
    axs[1].set_title("Live-points")

    plt.show()
    plt.pause(0.00001)

    axs[0].cla()
    axs[1].cla()
```

```python
def define_controlboard():
    """
    Creates the figure that will show some useful plots, called Control Board.
    In the upper part of the figure one can see the normal distribution and the selected lowest value point.
    In the lower part of the figure one can find the samples representing the prior volume and the proposal
    function centered in the parameter which generated the lowest valued log_multivariate_point.

    Pay attention: this works only in dimension 1.

    Returns:
        fig, axs: figure and axis without content
    """
    plt.ion()

    fig = plt.figure(num="CB", figsize=(5,7))
    axs = fig.subplots(nrows=2, ncols=1, sharex=True)

    return fig, axs
```

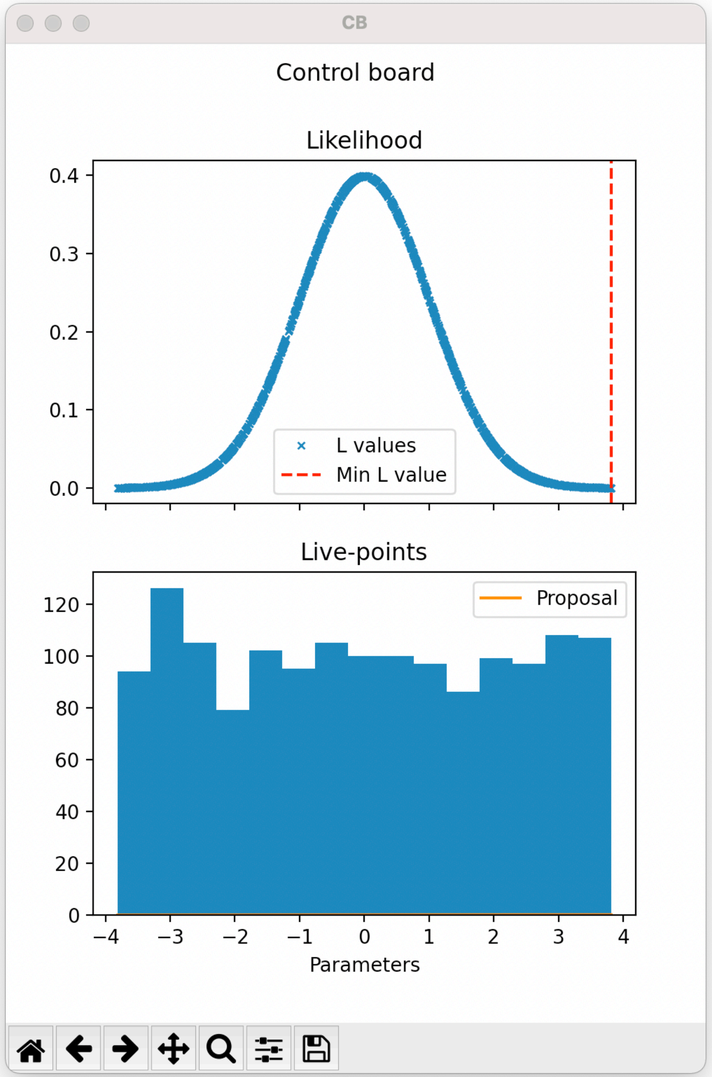
Struttura del codice

Inizializzazione dei parametri

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Welcome. This is a benchmark test for a Nested Sampling algorithm. Its
                                                  purpose is to integrate a Multivariate Normal Distribution with zero
                                                  mean and the identity matrix as covariance matrix.')

    parser.add_argument('dim', metavar='Dimensions', type=int,
                        help='How many dimensions should have the parameters\' space?')
    parser.add_argument('pts', metavar='Points', type=int,
                        help='Choose the number of points for the Prior Sample')
    parser.add_argument('distr', metavar='Sampling Distribution',
                        help='Which distribution would you like to use as a Sampler?')
    parser.add_argument('plot', metavar='Control Board', choices=['True', 'False'],
                        help='Select True if you want an over time updated plot of the Prior Sample and of the Likelihood!
                              Warning: This works only for Dimension = 1!')

    args = parser.parse_args()

    term_size = os.get_terminal_size()
    towel = np.random.Generator(np.random.PCG64(42))

    plots = args.plot

    dimensions   = args.dim
    samples_size = args.pts
    proposal_function = eval(args.distr)

    list_of_mult_params = np.arange(1., 5.)
    proposals_param_iterable = it.cycle(list_of_mult_params)

    log_evidence       = -np.inf
    logh               = -np.inf
    constant_increment = np.log(1 - np.exp(-1/samples_size))
    fraction = .1

    low_bound_value   =  -4.
    high_bound_value  =   4.

    low_bound   = np.full(shape=dimensions, fill_value = low_bound_value, dtype=float)
    high_bound  = np.full(shape=dimensions, fill_value = high_bound_value, dtype=float)

    parameters = towel.uniform(low_bound, high_bound, size=(samples_size, dimensions))

    t1 = time.time()
```

Struttura del codice

Ciclo while




```
if plots == 'True' and dimensions == 1:
    fig, axs = define_controlboard()

i = 0

while True:
    log_mult_points = [log_multivariate_point(parameters[j], dimensions) for j in range(samples_size)]

    min_index, min_likelihood_value = lowest_likelihood_point(log_mult_points)

    mult_param = next(proposals_param_iterable)

    if plots == 'True' and dimensions == 1:
        update_controlboard(parameters, min_index, log_mult_points, mult_param)

    a_new_point = prior_uniform_sampling(
        towel, dimensions, parameters,
        mult_param, parameters[min_index],
        low_bound_value, high_bound_value,
        proposal_function
        )

    if (log_multivariate_point(a_new_point[0], dimensions) > min_likelihood_value) == True:
        t2 = time.time()

        log_delta_X = -i/samples_size + constant_increment

        #                          This is decreasing                    This is increasing
        if max(log_mult_points) - log_evidence < np.log(fraction) + (i + i**.5)/samples_size:
            print("The mean increment is now too small to make a difference!\n")
            break

        print("—"*term_size.columns)
        print("This loop will end when max(ln(L)) - log(Z) is lesser than log(f) + (i + sqrt(i))/N")
        print(f"max(ln(L)) - log(Z)            --> {max(log_mult_points) - log_evidence}")
        print(f"log(f)     + (i + sqrt(i))/N   --> {np.log(fraction) + (i + i**0.5)/samples_size}\n")
        print(f"I found a new likelihood minimum after {(t2 - t1)} seconds!")
        parameters[min_index] = a_new_point[0]

        log_evidence = np.logaddexp(log_evidence, (log_delta_X + min_likelihood_value))

        loghplus = min_likelihood_value - log_evidence + np.log(min_likelihood_value - log_evidence) + log_delta_X
        logh = np.logaddexp(logh, loghplus)
        error = np.sqrt(np.exp(logh)/samples_size)

        print(f"The logarithm of evidence at iteration {i+1} is ------> {log_evidence} +- {error}")
        print("—"*term_size.columns + "\n")

        i += 1

    else:
        parameters[min_index] = parameters[min_index]
```

Struttura del codice

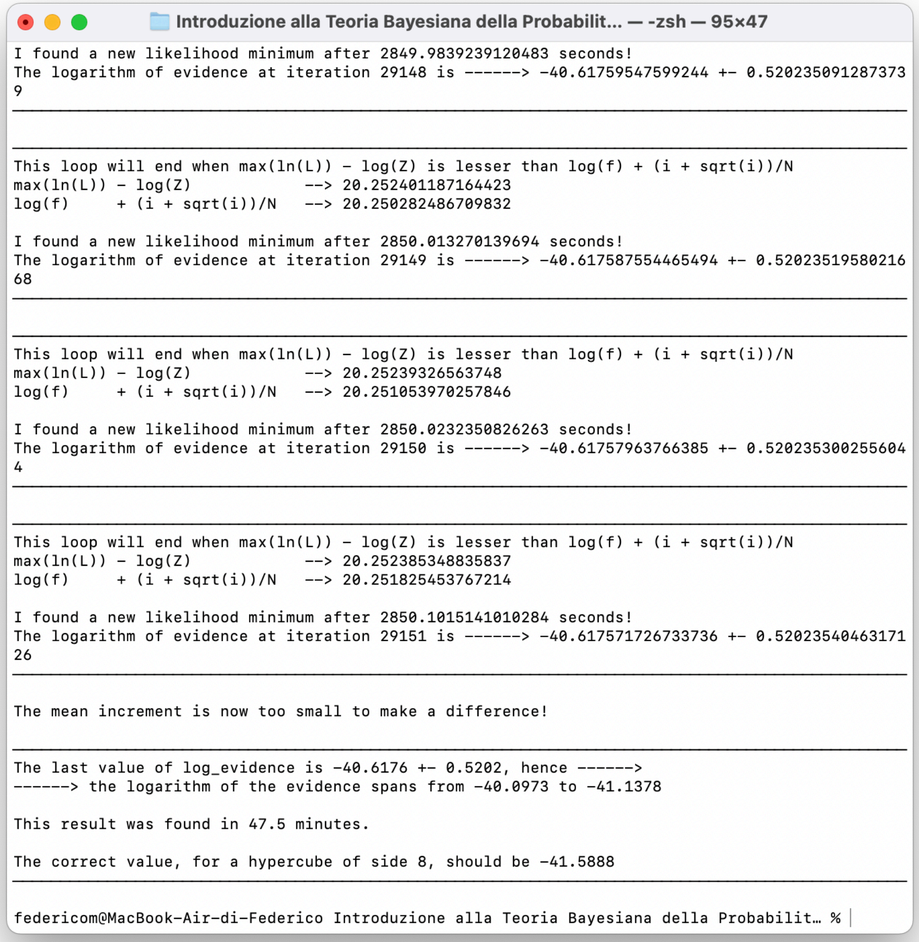
Risultato finale e codice ausiliario

```
    t3 = time.time()

    fin_time = round((t3-t1)/60, 2)
    fin_log_ev = round(log_evidence, 4)
    fin_err = round(error, 4)
    min_log_ev_val = round(log_evidence + error, 4)
    max_log_ev_val = round(log_evidence - error, 4)
    theor_value = round(-dimensions*np.log(high_bound_value - low_bound_value), 4)

    print("—"*term_size.columns)
    print(f"The last value of log_evidence is {fin_log_ev} +- {fin_err}, hence ------>")
    print(f"------> the logarithm of the evidence spans from {min_log_ev_val} to {max_log_ev_val}\n ")
    print(f"This result was found in {fin_time} minutes.\n")

    print(f"The correct value, for a hypercube of side {2*high_bound_value}, should be {theor_value}")
    print("—"*term_size.columns + "\n")
```

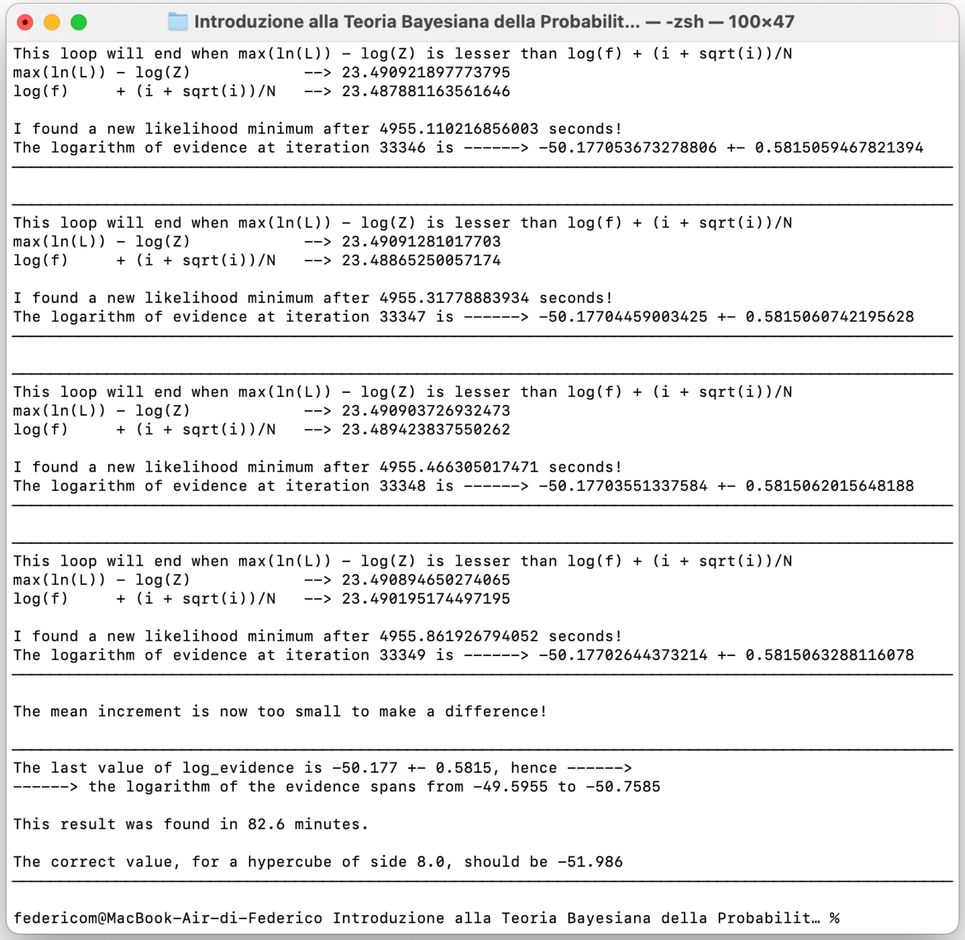
```
from scipy.stats import norm, anglit, semicircular
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-5., 5., 5000)
gauss1 = norm.pdf(x, loc=0., scale=1.)
gauss2 = norm.pdf(x, loc=0., scale=2.)
gauss3 = norm.pdf(x, loc=0., scale=3.)
gauss3 = norm.pdf(x, loc=0., scale=4.)

anglit1 = anglit.pdf(x, scale=1.)
anglit2 = anglit.pdf(x, scale=2.)
anglit3 = anglit.pdf(x, scale=3.)
anglit4 = anglit.pdf(x, scale=4.)

semicircular4 = semicircular.pdf(x, scale=4.)
semicircular3 = semicircular.pdf(x, scale=3.)
semicircular2 = semicircular.pdf(x, scale=2.)
semicircular1 = semicircular.pdf(x, scale=1.)


fig, ax = plt.subplots()

ax.plot(x, anglit1, 'b', label='Anglit 1')
ax.plot(x, anglit2, 'g', label='Anglit 2')
ax.plot(x, anglit3, 'r', label='Anglit 3')
ax.plot(x, anglit4, 'm', label='Anglit 4')

ax.plot(x, semicircular1, 'b', label='SemiCirc 1', ls='--')
ax.plot(x, semicircular2, 'g', label='SemiCirc 2', ls='--')
ax.plot(x, semicircular3, 'r', label='SemiCirc 3', ls='--')
ax.plot(x, semicircular4, 'm', label='SemiCirc 4', ls='--')

ax.set_xlabel('parameters')
ax.legend()
plt.show()
```

Alcuni risultati

Output



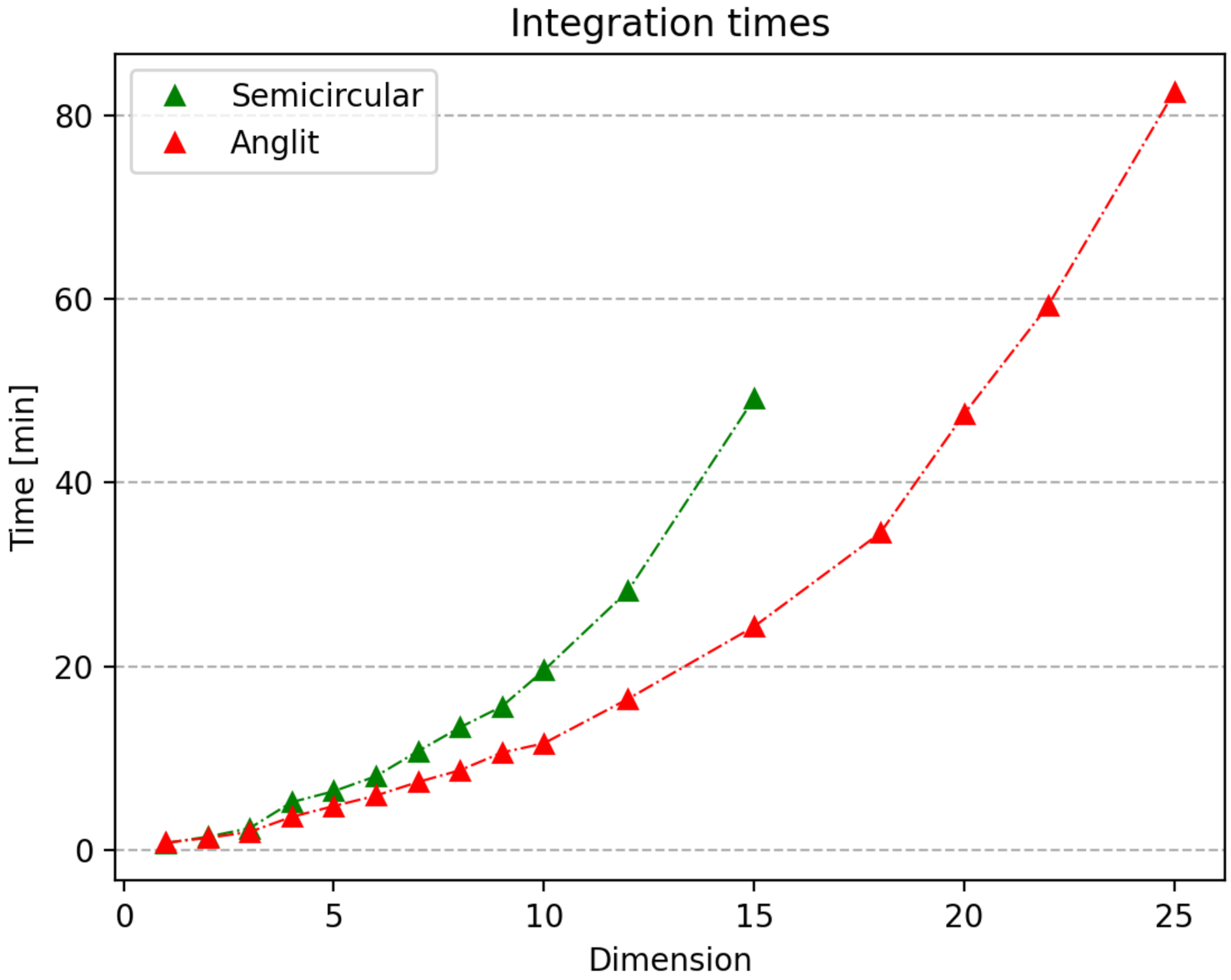
3. Risultato dell'integrazione di una distribuzione normale multivariata di dimensione 20



4. Risultato dell'integrazione di una distribuzione normale multivariata di dimensione 25

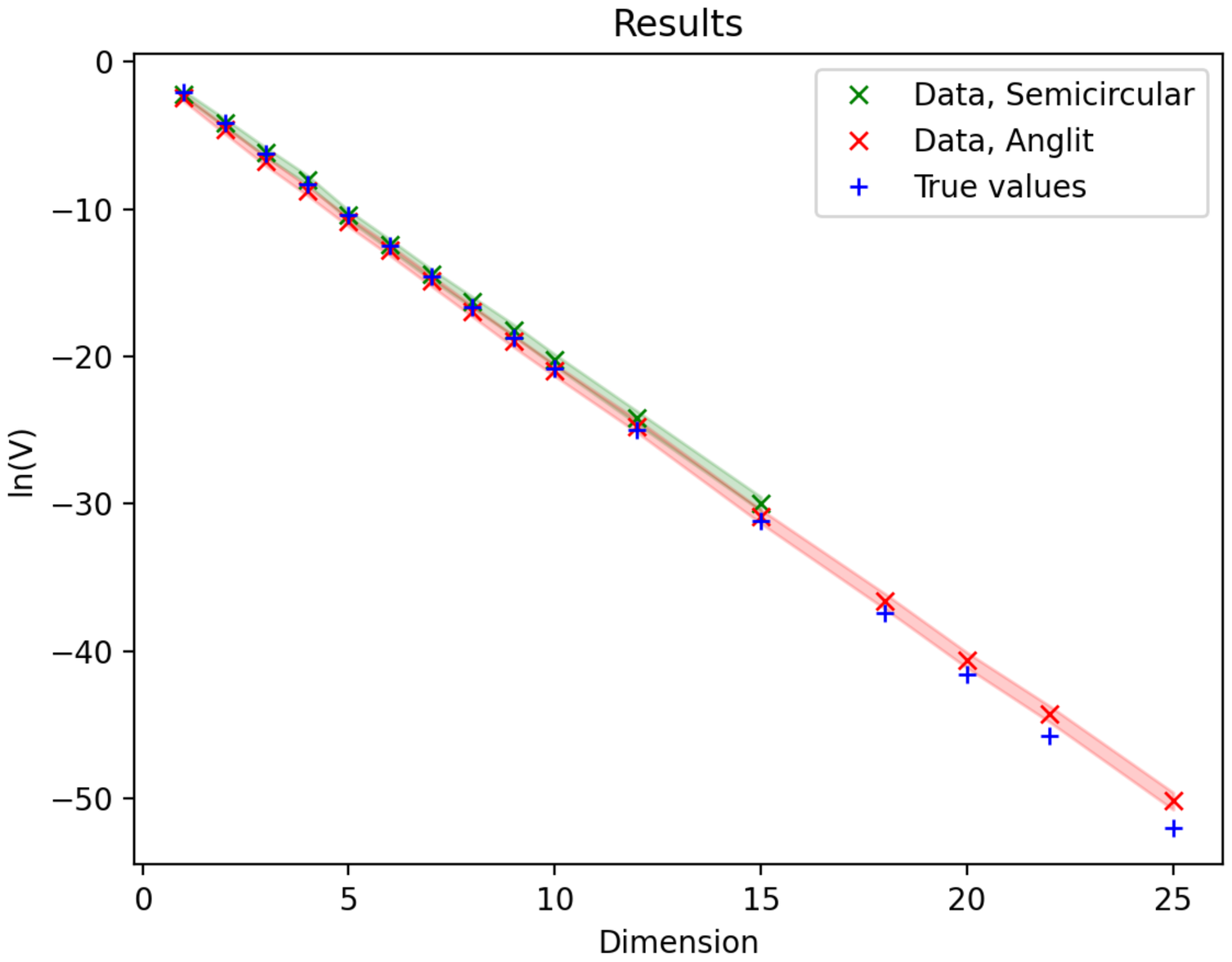
Alcuni risultati

Tempi di integrazione



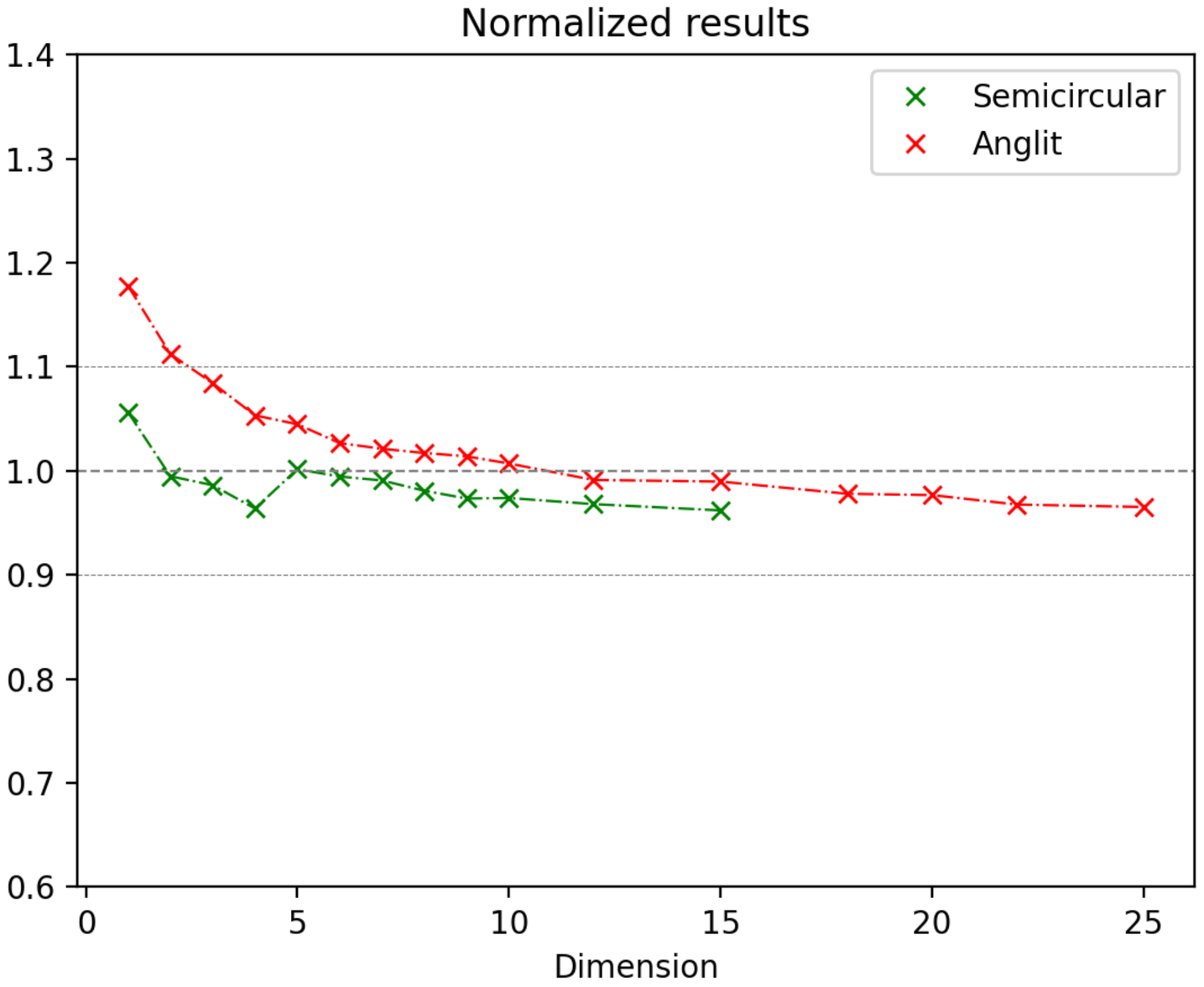
Alcuni risultati

Risultati non normalizzati



Alcuni risultati

Risultati normalizzati



Bibliografia

- Skilling J. (2004). Nested Sampling
- Pritchard J. *Nested Sampling*
- Skilling J. (2006). Nested Sampling for General Bayesian Computation. *Bayesian Analysis 1*(4), 833-860. 10.1214/06-BA127