



Nested Sampling Algorithm

Theory and Python implementation

Daniele Maria Di Nosse (605391)

Nested Sampling Algorithm

❖ Summary

- Introduction
- Outline of the algorithm
- Details
- Python implementation

❖ References

- John Skilling, Nested sampling for general Bayesian computation (2006)
<https://doi.org/10.1214/06-BA127>
- John Skilling, Nested Sampling (2004)
<https://ui.adsabs.harvard.edu/abs/2004AIPC..735..395S>
- Front image from
http://helper.ipam.ucla.edu/publications/elws2/elws2_14353.pdf

❖ Code

[https://github.com/DanieleMDiNosse/Nested Sampling](https://github.com/DanieleMDiNosse/Nested_Sampling)

Introduction

- The core of the bayesian probability is of course the Bayes' theorem.
- It has the ability to handle problems that the frequentist approach to probability can not.

$$P(H|DI) = \frac{P(H|I)P(D|HI)}{P(D|I)} = \frac{P(H|I)P(D|HI)}{\sum_i P(H_i|I)P(D|H_iI)}$$

$$Posterior = \frac{Prior \times Likelihood}{Evidence}$$

- It can be used to compute the so-called Odds Ratio between two set of parameters (in general between two models).

$$\frac{P(H_1|DI)}{P(H_2|DI)} = \frac{P(H_1|I)}{P(H_2|I)} \frac{P(D|H_1I)}{P(D|H_2I)} = Prior\ Odds \times Bayes\ Factor$$

Introduction

- Without any particular information in favor of one or the other model, Prior Odds are set to 1.
- If $H_1 \equiv H_1(\vec{\theta}_1)$, $H_2 \equiv H_2(\vec{\theta}_2)$ Bayes factor reduces to the ratio of the evidence for $\vec{\theta}_1$ and $\vec{\theta}_2$

$$B_{12} = \frac{\int_{\Theta_1} d\vec{\theta}_1 P(\vec{\theta}_1|H_1I)P(D|\vec{\theta}_1H_1I)}{\int_{\Theta_2} d\vec{\theta}_2 P(\vec{\theta}_2|H_2I)P(D|\vec{\theta}_2H_2I)}$$

- For the evidence to be used in the Bayes factor, we need to compute

$$Z = \int \int \dots \int L(\vec{\theta})\pi(\vec{\theta})d\vec{\theta}$$

- MCMC algorithms are used to compute the posterior.
- Using such methods to evaluate the evidence requires a lot of extra work (generalization of thermodynamic integration)
- Nested sampling puts the evidence as the prime target and the posterior is just a by-product of it.

Nested sampling - Outline of the algorithm

- The main idea is to image to change variable

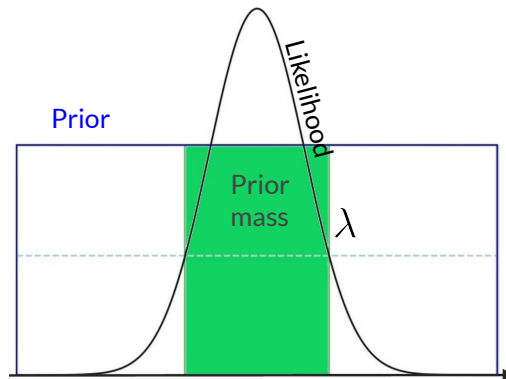
$$\pi(\vec{\theta})d\vec{\theta} = d\xi$$

- Where ξ is called the **prior mass** and represents the cumulative prior over a specific value of the likelihood

$$\xi(\lambda) = \int \int \dots \int_{L(\vec{\theta}) > \lambda} \pi(\vec{\theta}) d\vec{\theta}$$

- We constructed so a 1-dim integral over $[0,1]$ from a N-dim one:

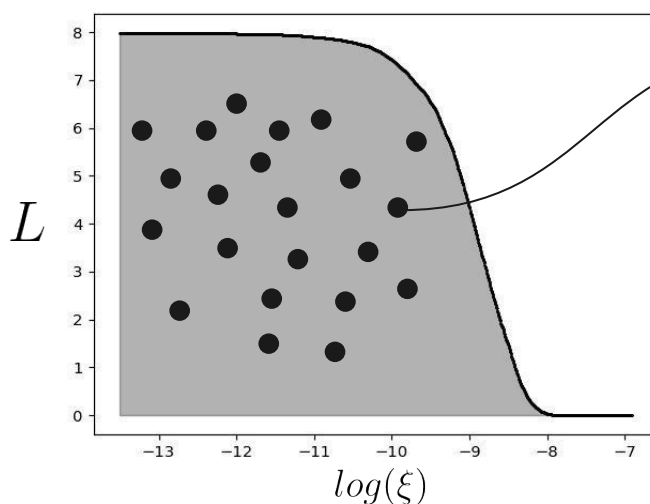
$$\int_0^1 L(\xi) d\xi$$



- If each of the N components of $\vec{\theta}$ is stored with an accuracy of 1 part in R, ξ should be stored as 1 part in R^N : information remains $N \log_2 R$

Nested sampling - Outline of the algorithm

- Image now to sort likelihoods in ascending order $L_{min} < L_1 < L_2 < \dots < L_{max}$
- At every value of the likelihood we can associate a value of the prior mass, starting from $\xi = 0$ ($L = L_{min}$) and going up until we reach the maximum of L , where the prior mass is $\xi = 1$



As by-product we have
sample from the posterior

$$P(\xi) = \frac{L(\xi)}{Z}$$

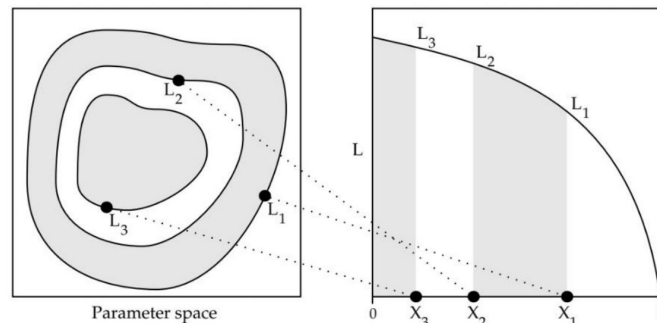
- The idea is to approximate the integral as the usual Riemann sum $Z \approx \sum_i L_i \Delta \xi_i$

Nested sampling - Details of the algorithm

- How do we compute $\Delta\xi_i$? We do not know the transformation from $\vec{\theta} \rightarrow \xi$!
- Nested Sampling tries to find it statistically.
- NS uses N objects randomly sampled from the prior that each time are updated with an evolving constraint regarding the growing of the likelihood .
- Each time we consider the object with smallest likelihood we have, L^* , remove it from the sample and replace it by a new one satisfying $L(\vec{\theta}_{new}) > L^*$
- In terms of the prior mass, this object are such that $\xi_{i+1} < \xi_i$, so we are shrinking the prior mass domain.
- Reason as follows:
 - At a certain iteration we have an upper bound ξ^* for the prior mass.
 - Choosing the worst L implies choosing a certain ξ
 - ξ is the largest number out of N uniformly distributed in $[0, \xi^*]$.
 - The shrinkage ratio $t = \xi/\xi^* \in [0, 1]$ is distributed as $p(t) = nt^{n-1}$ with mean and standard deviation such that

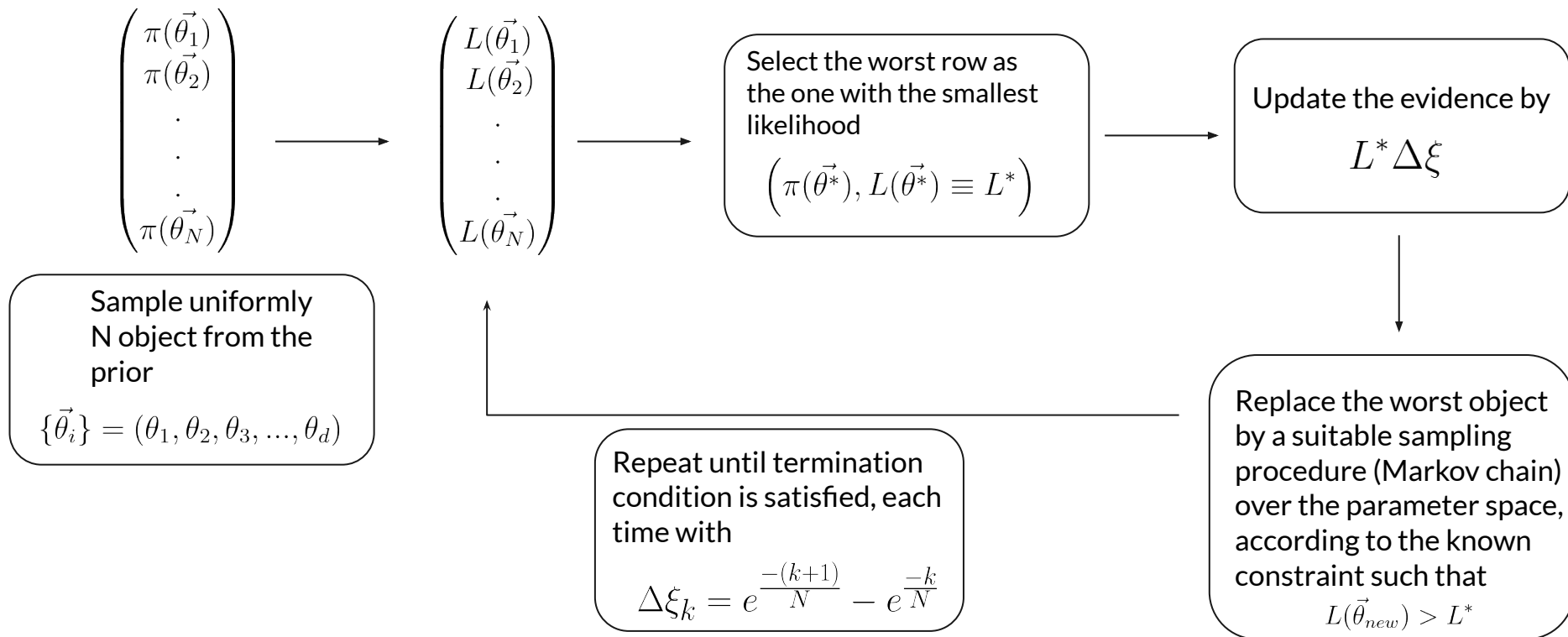
$$\log t = \frac{-1 \pm 1}{n}$$
 - After k iteration

$$\xi_k = \prod_{j=1}^k t_j \quad \text{with} \quad \log t \approx \frac{-1}{n}$$
 - Now we are able to compute the sum.



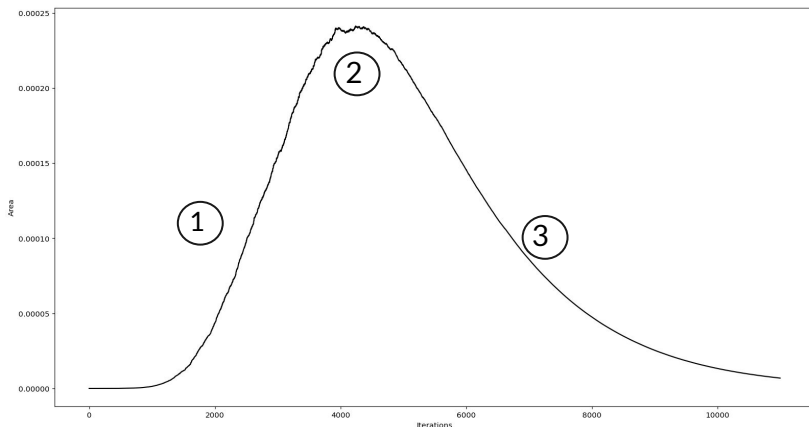
Nested sampling - Details of the algorithm

- Overview of the algorithm steps.



Nested sampling - Termination

- How do we decide when terminate the algorithm?
- The usual behaviour of the areas element is the following



1. Increase of L is dominant on the decrease of the widths
2. Balance
3. Decrease of the width take over the increase of L

- We want to stop the algorithm when $L(\vec{\theta}^*)\Delta\xi$ becomes so small that it does not contribute significantly to the evidence.
- Qualitatively this happens when the number of iterations exceeds significantly nH .
- More quantitatively this condition will occur when

$$\max\{L(\vec{\theta})\}_i \xi_i < fZ_i$$

Nested sampling - Error

- The peak is to be found in the region $\xi \approx e^{-H}$ where $H = \int P(\xi) \log [P(\xi)] d\xi \approx \sum_k \frac{\Delta \xi_k L_k}{Z} \log \left[\frac{L_k}{Z} \right]$ is the information. It is a measure of the prior-to-posterior shrinkage in logarithmic form.

- Remember that

$$\log t = \frac{-1 \pm 1}{n} \quad \text{and} \quad \xi_k = \prod_{j=1}^k t_j \quad \longrightarrow \quad \log \xi_k = \frac{-k \pm \sqrt{k}}{n}$$

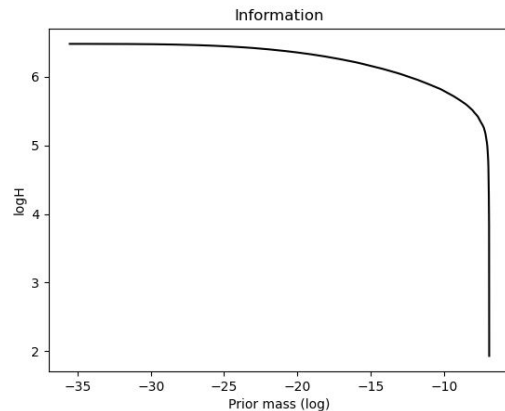
- The nH iterations required induces an error over the evidence such that

$$\log Z \approx \sum_k \log (L_k \xi_k) \pm \sqrt{\frac{H}{n}}$$

- More accurate estimate of the error can be done using the distribution of t $p(t) = nt^{n-1}$

$$t \implies \xi \implies \Delta \xi \implies Z \equiv Z(t)$$

- We can sample several t from the distribution, obtain the values of the evidence and compute mean and standard deviation from them



Nested sampling - Python implementation

- Compute numerically

$$\int_{-A}^A \left(\frac{1}{\sqrt{2\pi\Sigma}} \right)^d e^{-\frac{1}{2}(\vec{x}-\vec{\mu})^t \Sigma^{-1}(\vec{x}-\vec{\mu})} d\vec{x}$$

where

$$\Sigma = \mathbb{I} \quad ; \quad \vec{\mu} = 0 \\ d \in [1, 2, \dots, 50] \quad ; \quad A = 5$$

```
7 def log_likelihood(x, dim, init):
8     ''' Return the logarithm of a N-dimensional gaussian likelihood.
9     It is set in such a way that the integral of the product with the
10    prior over the parameter space is 1.
11
12    Parameters
13    -----
14    x : numpy.array
15        If init is set to True, x should be a MxN matrix whose M rows (number of points)
16        are random N-dimensional vectors. In this case it is used to initialize the
17        likelihood of the live points. If init is set to False, x should be just a
18        random N dimensional vector.
19    dim : int
20        Dimension of the parameter space.
21    init: bool
22        You can choose to use the function to initialize the likelihood of the live
23        points (True) or to generate just a new likelihood value (False)
24
25    Returns
26    -----
27    Likelihood : list or float
28        Likelihood values or single likelihood value
29    '''
30
31    if init:
32        likelihood = [- 0.5*dim*np.log(2*np.pi) - 0.5*v.T.dot(v) for v in x]
33    else:
34        likelihood = - 0.5*dim*np.log(2*np.pi) - 0.5*x.T.dot(x)
35
36    return likelihood
37
```

Live Points

$$\begin{pmatrix} \theta_1^1, \theta_2^1, \dots, \theta_d^1, L(\vec{\theta}_1) \\ \theta_1^2, \theta_2^2, \dots, \theta_d^2, L(\vec{\theta}_2) \\ \vdots \\ \theta_1^N, \theta_2^N, \dots, \theta_d^N, L(\vec{\theta}_N) \end{pmatrix}$$

Nested sampling - Python implementation

- The idea is to tune the average jump of the points in two ways:

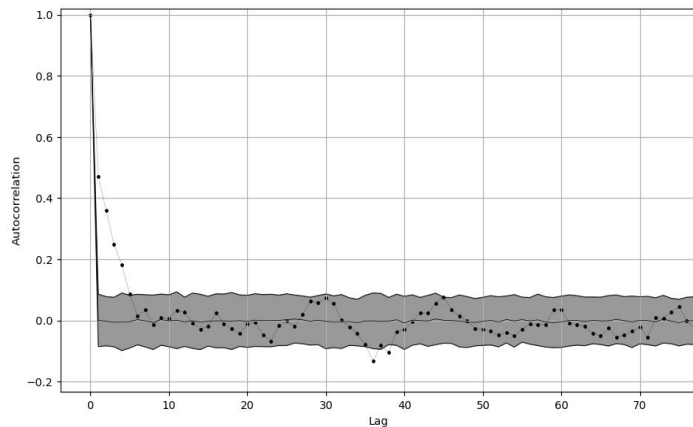
→ as a function of the dimension of the parameter space for the normal proposal

→ keeping the acceptance ratio around 50% in the uniform case. If there are more accepted points than rejected we can enlarge the jump, otherwise we have to shrink it

```
03 def proposal(x, dim, boundary, std, distribution):
04     """ Sample a new object from the prior subject to the constrain  $L(x_{new}) > L_{worst\_old}$  """
05     ...
06
07     logLmin = x[dim]
08     boundary_point = x[:dim]
09
10
11     start = time.time()
12     accepted = 0
13     rejected = 0
14     n = 0
15     k_n = 1/(2*np.log(dim+1))
16
17     while True:
18         new_line = np.zeros(dim+1, dtype=np.float64)
19         for i in range(len(new_line[:dim])):
20
21             if distribution == 'uniform':
22                 new_line[:dim][i] = boundary_point[i] + np.random.uniform(-std, std)
23                 while np.abs(new_line[:dim][i]) > boundary:
24                     new_line[:dim][i] = boundary_point[i] + np.random.uniform(-std, std)
25
26             if distribution == 'normal':
27                 new_line[:dim][i] = np.random.normal(boundary_point[i], k_n*std)
28                 while np.abs(new_line[:dim][i]) > boundary:
29                     new_line[:dim][i] = np.random.normal(boundary_point[i], k_n*std)
30
31         new_line[dim] = log_likelihood(new_line[:dim], dim, init=False)[0]
32
33         if new_line[dim] < logLmin:
34             rejected += 1
35         if new_line[dim] > logLmin:
36             accepted += 1
37             boundary_point[:dim] = new_line[:dim]
38             if n > 10:
39                 end = time.time()
40                 t = end - start
41                 break
42
43         if distribution == 'uniform':
44             if accepted != 0 and rejected != 0:
45                 if accepted > rejected: std *= np.exp(1.0/accepted)
46                 if accepted < rejected: std /= np.exp(1.0/rejected)
47
48     return new_line, t, accepted, rejected
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
```

Nested sampling - Python implementation

- New object must be independent from the starting point of the Markov chain



```
103 def proposal(x, dim, logLmin, boundary_point, boundary, std, distribution):
104     """ Sample a new object from the prior subject to the constrain  $L(x_{new}) > L_{worst\_old}$ 
105     """
106     start = time.time()
107     accepted = 0
108     rejected = 0
109     n = 0
110     k_n = 1/(2*np.log(dim+1))
111
112     while True:
113         new_line = np.zeros(dim+1, dtype=np.float64)
114         for i in range(len(new_line[:dim])):
115
116             if distribution == 'uniform':
117                 new_line[:dim][i] = boundary_point[i] + np.random.uniform(-std, std)
118                 while np.abs(new_line[:dim][i]) > boundary:
119                     new_line[:dim][i] = boundary_point[i] + np.random.uniform(-std, std)
120
121             if distribution == 'normal':
122                 new_line[:dim][i] = np.random.normal(boundary_point[i], k_n*std)
123                 while np.abs(new_line[:dim][i]) > boundary:
124                     new_line[:dim][i] = np.random.normal(boundary_point[i], k_n*std)
125
126         new_line[dim] = log_likelihood(new_line[:dim], dim, init=False)[0]
127
128         if new_line[dim] < logLmin:
129             rejected += 1
130         if new_line[dim] > logLmin:
131             n += 1
132             accepted += 1
133             boundary_point[:dim] = new_line[:dim]
134             if n > 10:
135                 end = time.time()
136                 t = end - start
137                 break
138
139         if distribution == 'uniform':
140             if accepted != 0 and rejected != 0:
141                 if accepted > rejected: std *= np.exp(1.0/accepted)
142                 if accepted < rejected: std /= np.exp(1.0/rejected)
143
144     return new_line, t, accepted, rejected
```

Nested sampling - Python implementation

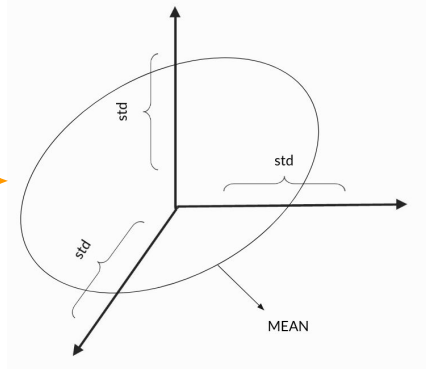
```
43 def autocorrelation(x, max_lag, bootstrap=False):
44     '''Simple implementation for the autocorrelation function.
45     Autocorrelation function'''
46
47     x = np.array(x)
48     x_mean = np.mean(x)
49     auto_corr = []
50     for d in range(max_lag):
51         ac = 0
52         for i in range(len(x)-d):
53             ac += (x[i] - x_mean) * (x[i+d] - x_mean)
54         ac = ac / np.sqrt(np.sum((x - x_mean)**2) * np.sum((x - x_mean)**2))
55         auto_corr.append(ac)
56     auto_corr = np.array(auto_corr)
57     plt.figure()
58     plt.plot(auto_corr, 'k--', linewidth=0.4, alpha=0.5)
59     plt.scatter(np.arange(len(auto_corr)), auto_corr, s=5, color='black')
60     plt.grid()
61     plt.xlabel('Lag')
62     plt.ylabel('Autocorrelation')
63
64     if bootstrap:
65         auto_corr_bootstrap = []
66         for i in range(200):
67             xs = x
68             np.random.shuffle(xs)
69             xs_mean = np.mean(xs)
70             auto_corr_bootstrap_i = []
71             for d in range(max_lag):
72                 ac = 0
73                 for i in range(len(x)-d):
74                     ac += (x[i] - x_mean) * (x[i+d] - x_mean)
75                 ac = ac / np.sqrt(np.sum((x - x_mean)**2) * np.sum((x - x_mean)**2))
76                 auto_corr_bootstrap_i.append(ac)
77             auto_corr_bootstrap.append(auto_corr_bootstrap_i)
78         meanac = np.mean(np.array(auto_corr_bootstrap), axis=0)
79         stdac = np.std(np.array(auto_corr_bootstrap), axis=0)
80         plt.plot(meanac - 2*stdac, 'black', lw=0.5)
81         plt.plot(meanac, 'black', lw=0.5)
82         plt.plot(meanac + 2*stdac, 'black', lw=0.5)
83         plt.fill_between(np.arange(0, max_lag), meanac + 2*stdac, meanac - 2*stdac, color='black',
84             alpha=0.4)
85     plt.show()
86
87     return auto_corr
```

The random shuffling of time series ensures all the temporal relations to be lost

Nested sampling - Python implementation

```
def nested_sampler(live_points, dim, boundary, proposal_distribution, verbose=False):  
    '''Nested Sampling by Skilling (2004)'''  
  
    N = live_points.shape[0]  
    f = np.log(0.01)  
    area = []; Zlog = [[]]; logL_worst = []; T = []; prior_mass = []; logH_list = []  
  
    logZ = -np.inf  
    logH = -np.inf  
    parameters = np.random.uniform(-boundary, boundary, size=(N, dim))  
    live_points[:, :dim] = parameters  
    live_points[:, dim] = log_likelihood(parameters, dim, init=True)  
    logwidth = np.log(1.0 - np.exp(-1.0/N))  
  
    steps = 0  
    rejected = 0  
    accepted = 0  
    while True:  
        steps += 1  
        prior_mass.append(logwidth)  
  
        Lw_idx = np.argmin(live_points[:, dim])  
        logLw = live_points[Lw_idx, dim]  
        logL_worst.append(logLw)  
  
        logZnew = np.logaddexp(logZ, logwidth+logLw)  
        logZ = logZnew  
  
        logH = np.logaddexp(logH, logwidth + logLw - logZ + np.log(logLw - logZ))  
        logH_list.append(logH)  
        error = np.sqrt(np.exp(logH)/N)  
        Zlog[0].append(logZ + error)  
        Zlog[1].append(logZ - error)  
  
        survivors = np.delete(live_points, Lw_idx, axis=0)  
        std = np.mean(np.std(survivors[:,dim], axis = 0))
```

Initialization



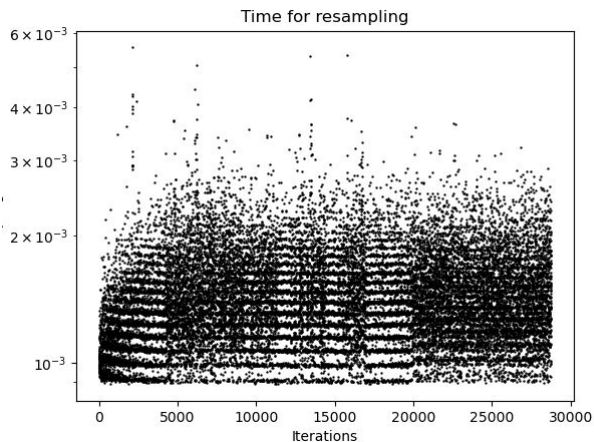
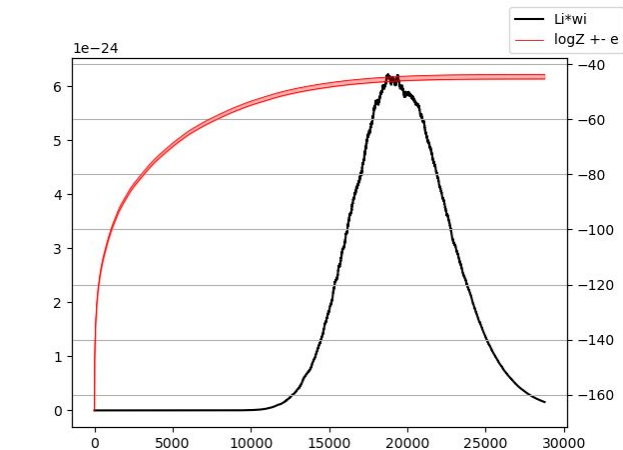
```
90     area.append(logwidth+logLw)  
91  
92     if verbose:  
93         print("i:{0} d={1} log(Lw)={2:.2f} term={3:.2f} log(Z)={4:.2f} std={5:.2f}  
94             e={6:.2f} H={7:.2f} prop={8}").format(steps, dim, logLw,  
95             (max(live_points[:,dim]) - steps/N - f - logZ), logZ, std, error,  
96             np.exp(logH), proposal_distribution)  
97  
98     new_sample, t, acc, rej = proposal(live_points[Lw_idx], dim, boundary, std,  
99     proposal_distribution)  
100     accepted += acc  
101     rejected += rej  
102     T.append(t)  
103  
104     live_points[Lw_idx] = new_sample  
105     logwidth -= 1.0/N  
106  
107     if max(live_points[:,dim]) - (steps+steps**0.5)/N < f + logZ:  
108         break  
109  
110     final_term = np.log(np.exp(live_points[:,dim]).sum()*np.exp(-steps/N)/N)  
111     logZ = np.logaddexp(logZ, final_term)  
112     area = np.exp(area)  
113  
114     return area, Zlog, logL_worst, prior_mass, logZ, T, steps, accepted, rejected,  
115     logH_list, error
```

Sample new object

Terminal condition

Adding final term

Nested sampling - Python implementation (results)



===== SUMMARY =====

Dimension of the integral = 20

Number of steps required = 28628

Evidence = -44.46 +- 0.81

Theoretical value = -46.05170185988092

Information = 649.47

Maximum of the likelihood = -18.38

Proposal chosen: normal

Last area value = 0.00

Last worst Likelihood = -21.50

Accepted and rejected points: 314908, 169792

Mass prior sum = 1.00

Total time: 56.25 s

=====

Nested sampling - Python implementation (results)

- Both in terms of accuracy and computational time, the normal proposal with the implementation used to handle its standard deviation performed better than the uniform one.

