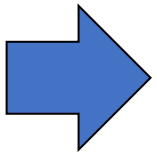
Introduction to Bayesian Statistics - 10

Edoardo Milotti

Università di Trieste and INFN-Sezione di Trieste

Our next important topic: Bayesian estimates often require complex numerical integrals. How do we confront this problem?



enter the Monte Carlo methods!

1. acceptance-rejection sampling
2. importance sampling
3. statistical bootstrap
4. Bayesian methods in a sampling-resampling perspective
5. Introduction to Markov chains and to Random Walks (RW)
6. Detailed balance and Boltzmann's H-theorem
7. The Gibbs sampler
8. Simulated annealing and the Traveling Salesman Problem (TSP)
9. The Metropolis algorithm
10. More on Gibbs sampling
11. Image restoration and Markov Random Fields (MRF)
12. The Metropolis-Hastings algorithm and Markov Chain Monte Carlo (MCMC)
13. The efficiency of MCMC methods
14. Affine-invariant MCMC algorithms (emcee)

Monte Carlo Methods in Statistical Mechanics: Foundations and New Algorithms

Alan D. Sokal

Department of Physics
New York University
4 Washington Place
New York, NY 10003
USA

E-mail: SOKAL@NYU.EDU

Lectures at the Cargèse Summer School on
"Functional Integration: Basics and Applications"
September 1996

Before embarking on 9 hours of lectures on Monte Carlo methods, let me offer a warning:

> *Monte Carlo is an extremely* bad *method; it should be used only when all alternative methods are worse.*

Why is this so? Firstly, *all* numerical methods are potentially dangerous, compared to analytic methods; there are more ways to make mistakes. Secondly, as numerical methods go, Monte Carlo is one of the least efficient; it should be used only on those intractable problems for which all other numerical methods are even *less* efficient.

Let me be more precise about this latter point. Virtually all Monte Carlo methods have the property that the statistical error behaves as

$$\text{error} \sim \frac{1}{\sqrt{\text{computational budget}}}$$

(or worse); this is an essentially universal consequence of the central limit theorem. It may be possible to improve the *proportionality constant* in this relation by a factor of 10^6 or more — this is one of the principal subjects of these lectures — but the overall $1/\sqrt{n}$ behavior is inescapable. This should be contrasted with traditional deterministic numerical methods whose rate of convergence is typically something like $1/n^4$ or e^{-n} or e^{-2^n}. Therefore, Monte Carlo methods should be used only on those extremely difficult problems in which all alternative numerical methods behave even *worse* than $1/\sqrt{n}$.

Consider, for example, the problem of numerical integration in d dimensions, and let us compare Monte Carlo integration with a traditional deterministic method such as Simpson's rule. As is well known, the error in Simpson's rule with n nodal points behaves asymptotically as $n^{-4/d}$ (for smooth integrands). In low dimension ($d < 8$) this is much better than Monte Carlo integration, but in high dimension ($d > 8$) it is much worse. So it is not surprising that Monte Carlo is the method of choice for performing high-dimensional integrals. It is still a bad method: with an error proportional to $n^{-1/2}$, it is difficult to achieve more than 4 or 5 digits accuracy. But numerical integration in high dimension is very difficult; though Monte Carlo is bad, all other known methods are worse.[1]

In summary, Monte Carlo methods should be used only when neither analytic methods nor deterministic numerical methods are workable (or efficient). One general domain of application of Monte Carlo methods will be, therefore, to systems with *many degrees of freedom, far from the perturbative regime*. But such systems are precisely the ones of greatest interest in statistical mechanics and quantum field theory!

It is appropriate to close this introduction with a general methodological observation, ably articulated by Wood and Erpenbeck [3]:

> ... these [Monte Carlo] investigations share some of the features of ordinary experimental work, in that they are susceptible to both statistical and systematic errors. With regard to these matters, we believe that papers should meet much the same standards as are normally required for experimental error, descriptions of experimental conditions (i.e. parameters of the calculation), relevant details of apparatus (program) design, comparisons with previous investigations, discussion of systematic errors, etc. Only if these are provided will the results be trustworthy guides to improved theoretical understanding.

13. The efficiency of MCMC methods

The effective use of MCMC programs requires the fine-tuning of many aspects

- Duration of burn-in (initialization)
- Number of parallel chains (random walkers)
- Selection of jumping rule (proposal function)
- Selection of convergence rule
- …

Statistical Science
1992, Vol. 7, No. 4, 457–511

Inference from Iterative Simulation Using Multiple Sequences

Andrew Gelman and Donald B. Rubin

Abstract. The Gibbs sampler, the algorithm of Metropolis and similar iterative simulation methods are potentially very helpful for summarizing multivariate distributions. Used naively, however, iterative simulation can give misleading answers. Our methods are simple and generally applicable to the output of any iterative simulation; they are designed for researchers primarily interested in the science underlying the data and models they are analyzing, rather than for researchers interested in the probability theory underlying the iterative simulations themselves. Our recommended strategy is to use several independent sequences, with starting points sampled from an overdispersed distribution. At each step of the iterative simulation, we obtain, for each univariate estimand of interest, a distributional estimate and an estimate of how much sharper the distributional estimate might become if the simulations were continued indefinitely. Because our focus is on applied inference for Bayesian posterior distributions in real problems, which often tend toward normality after transformations and marginalization, we derive our results as normal-theory approximations to exact Bayesian inference, conditional on the observed simulations. The methods are illustrated on a random-effects mixture model applied to experimental measurements of reaction times of normal and schizophrenic patients.

Key words and phrases: Bayesian inference, convergence of stochastic processes, EM, ECM, Gibbs sampler, importance sampling, Metropolis algorithm, multiple imputation, random-effects model, SIR.

1.3 Our Approach

Our method is composed of two major steps. First, an estimate of the target distribution is created, centered about its mode (or modes, which are typically found by an optimization algorithm) and "overdispersed" in the sense of being more variable than the target distribution. The approximate distribution is then used to start several independent sequences of the iterative simulation. The second major step is to analyze the multiple sequences to form a distributional estimate of what is known about the target random variable, given the simulations thus far. This distributional estimate, which is in the form of a Student's t distribution for each scalar estimand, is somewhere between its starting and target distributions and provides the basis for an estimate of how close the simulation process is to convergence – that is, how much sharper the distributional estimate might become if the simulations were run longer.

The G-R monitoring index

Given any individual sequence, and if approximate convergence has been reached, an assumption is made that inferences about any quantity of interest is made by computing the sample mean and variance from the simulated draws. Thus, the m chains yield m possible inferences; to answer the question of whether these inferences are similar enough to indicate approximate convergence, Gelman and Rubin (1992a) suggested comparing these to the inference made by mixing together the mn draws from all the sequences.

from Brooks and Gelman, 1998

- *m* chains (walkers) and *n* samples/chain
- mean value of chain j

$$\bar{x}_j = \frac{1}{n}\sum_{i=1}^{n} x_i^{(j)}$$

- mean of the means of all chains

$$\bar{x}_* = \frac{1}{m}\sum_{j=1}^{m} \bar{x}_j$$

- variance of the means of all chains

$$\frac{B}{n} = \frac{1}{m-1}\sum_{j=1}^{m} (\bar{x}_j - \bar{x}_*)^2$$

Between-sequence variance

- averaged variances of individual chains averaged over all chains

$$W = \frac{1}{m}\sum_{j=1}^{m}\left[\frac{1}{n-1}\sum_{i=1}^{n}\left(x_i^{(j)} - \bar{x}_j\right)^2\right]$$

Within-sequence variance

- the variance

$$\hat{\sigma}^2 = \frac{n-1}{n}W + \frac{B}{n}$$

would be an unbiased estimate of the true variance if the starting points were drawn from the target distribution, but it is an overestimate if the starting distribution is overdispersed

- taking the variability of the mean into account yields a different estimate of the variance

$$\hat{V} = \hat{\sigma}^2 + \frac{B}{mn}$$

- the so-called potential scale reduction factor (PSRF) can be interpreted as a convergence diagnostic – when large it can be further decreased by continuing the simulation, **when close to 1 it shows that the set of simulations is close to the target distribution**

$$\hat{R} = \frac{\hat{V}}{W} = \frac{m+1}{m}\frac{\hat{\sigma}^2}{W} - \frac{n-1}{mn}$$

Gelman-Rubin statistic

An Introduction to Probability Theory and Its Applications

WILLIAM FELLER

Eugene Higgins Professor of Mathematics

Princeton University

VOLUME I

THIRD EDITION

CHAPTER III*

Fluctuations in Coin Tossing and Random Walks

This chapter digresses from our main topic, which is taken up again only in chapter V. Its material has traditionally served as a first orientation and guide to more advanced theories. Simple methods will soon lead us to results of far-reaching theoretical and practical importance. We shall encounter theoretical conclusions which not only are unexpected but actually come as a shock to intuition and common sense. They will reveal that commonly accepted notions concerning chance fluctuations are without foundation and that the implications of the law of large numbers are widely misconstrued. For example, in various applications it is assumed that observations on an individual coin-tossing game during a long time interval will yield the same statistical characteristics as the observation of the results of a huge number of independent games at one given instant. This is not so. Indeed, using a currently popular jargon we reach the conclusion that in a population of normal coins the majority is necessarily maladjusted. [For empirical illustrations see section 6 and example (4.*b*).]

6. AN EXPERIMENTAL ILLUSTRATION

Figure 4 represents the result of a computer experiment simulating 10,000 tosses of a coin; the same material is tabulated in example I, (6.*c*). The top line contains the graph of the first 550 trials; the next two lines represent the entire record of 10,000 trials the scale in the horizontal direction being changed in the ratio 1:10. The scale in the vertical direction is the same in the two graphs.

When looking at the graph most people feel surprised by the length of the intervals between successive crossings of the axis. As a matter of fact, the graph represents a rather mild case history and was chosen as the mildest among three available records. A more startling example is obtained by looking at the same graph in the *reverse* direction; that is, reversing the order in which the 10,000 trials actually occurred (see section 8). Theoretically, the series as graphed and the reversed series are equally legitimate as representative of an ideal random walk. The reversed random walk has the following characteristics. Starting from the origin

the path stays on the	
negative side	*positive side*
for the first 7804 *steps*	*next* 8 *steps*
next 2 *steps*	*next* 54 *steps*
next 30 *steps*	*next* 2 *steps*
next 48 *steps*	*next* 6 *steps*
next 2046 *steps*	
Total of 9930 steps	*Total of* 70 steps
Fraction of time: 0.993	*Fraction of time:* 0.007

This *looks* absurd, and yet the probability that in 10,000 tosses of a perfect coin the lead is at one side for more than 9930 trials and at the other for fewer than 70 exceeds $\frac{1}{10}$. In other words, on the average *one record out of ten will look worse than the one just described.* By contrast, the probability of a balance better than in the graph is only 0.072.

The original record of figure 4 contains 78 changes of sign and 64 other returns to the origin. The reversed series shows 8 changes of sign and 6 other returns to the origin. Sampling of expert opinion revealed that even trained statisticians expect much more than 78 changes of sign in 10,000 trials, and nobody counted on the possibility of only 8 changes of sign. Actually the probability of not more than 8 changes of sign exceeds 0.14, whereas the probability of more than 78 changes of sign is about 0.12. As far as the number of changes of sign is concerned the two records stand on a par and, theoretically, neither should cause surprise. If they seem startling, this is due to our faulty intuition and to our having been exposed to too many vague references to a mysterious "law of averages."

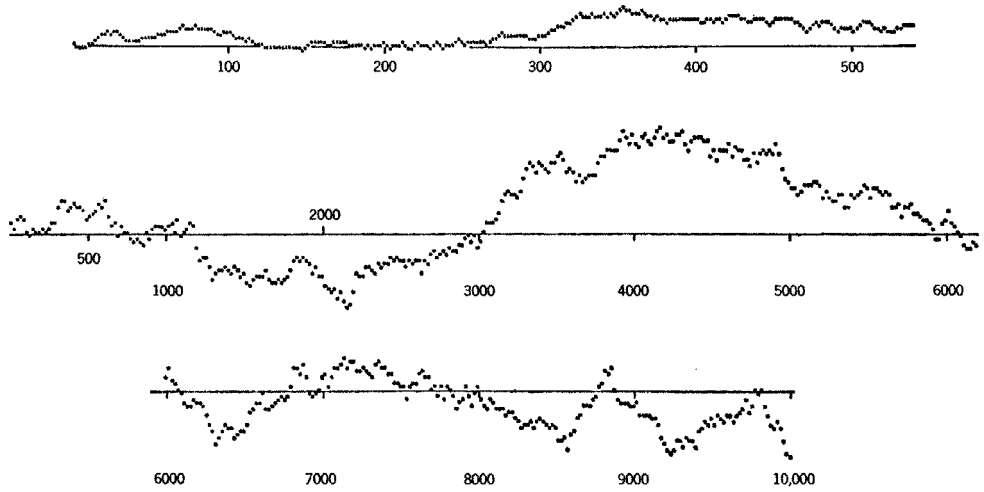
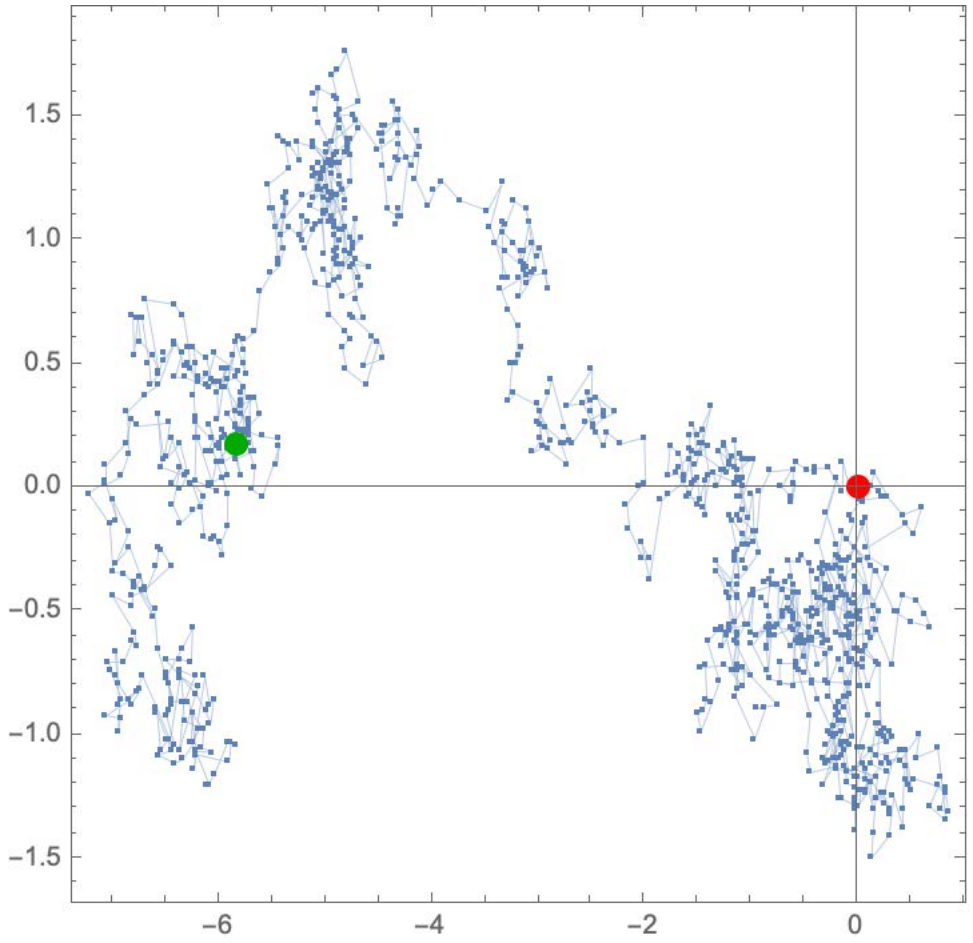
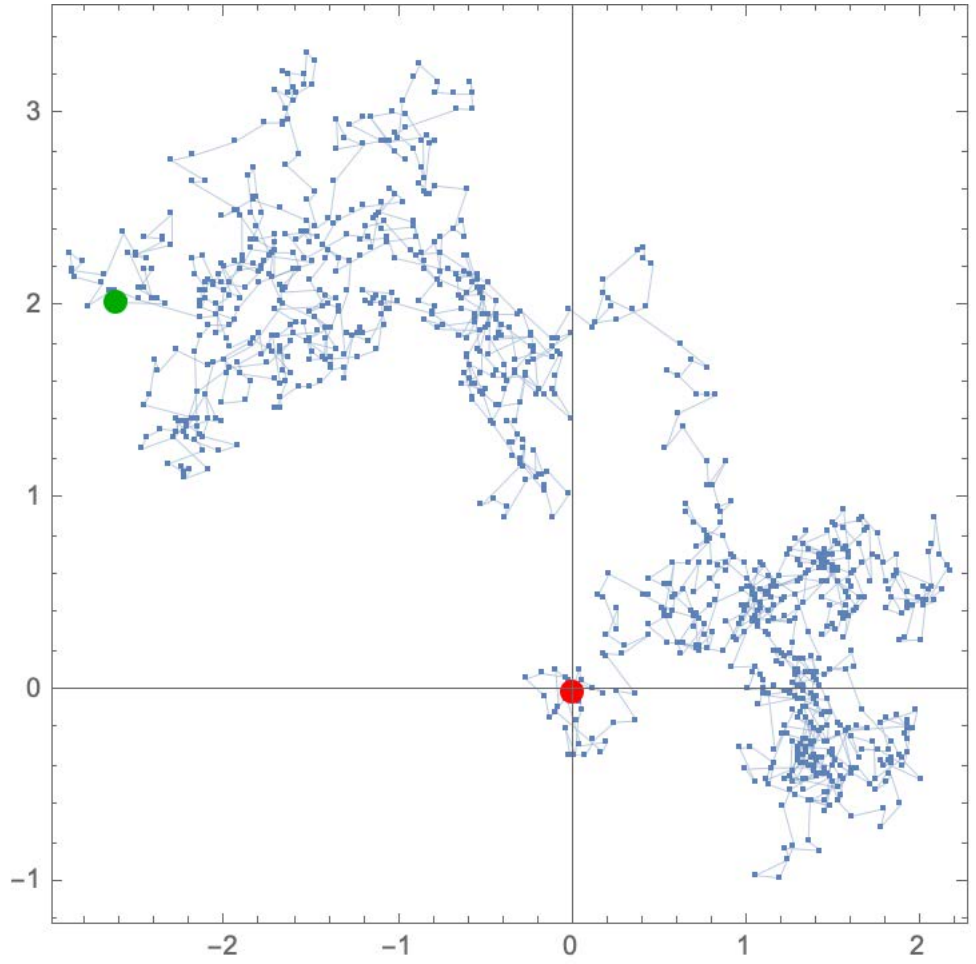
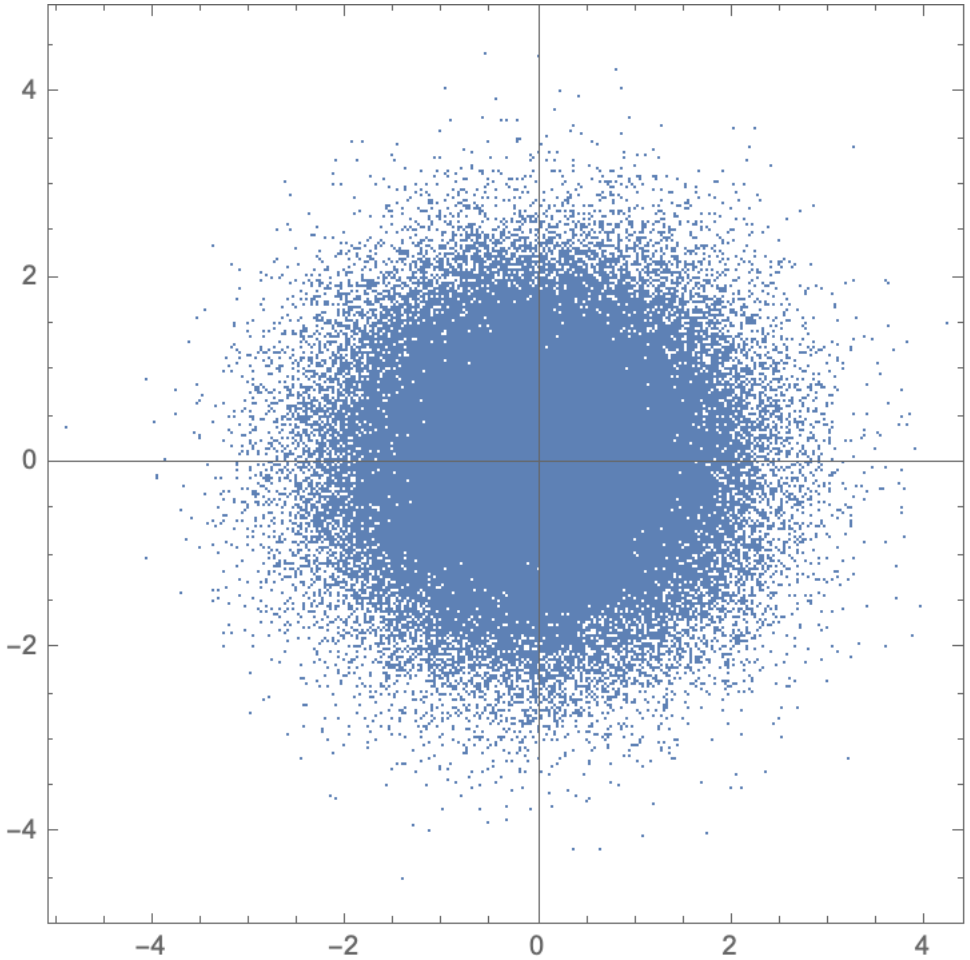


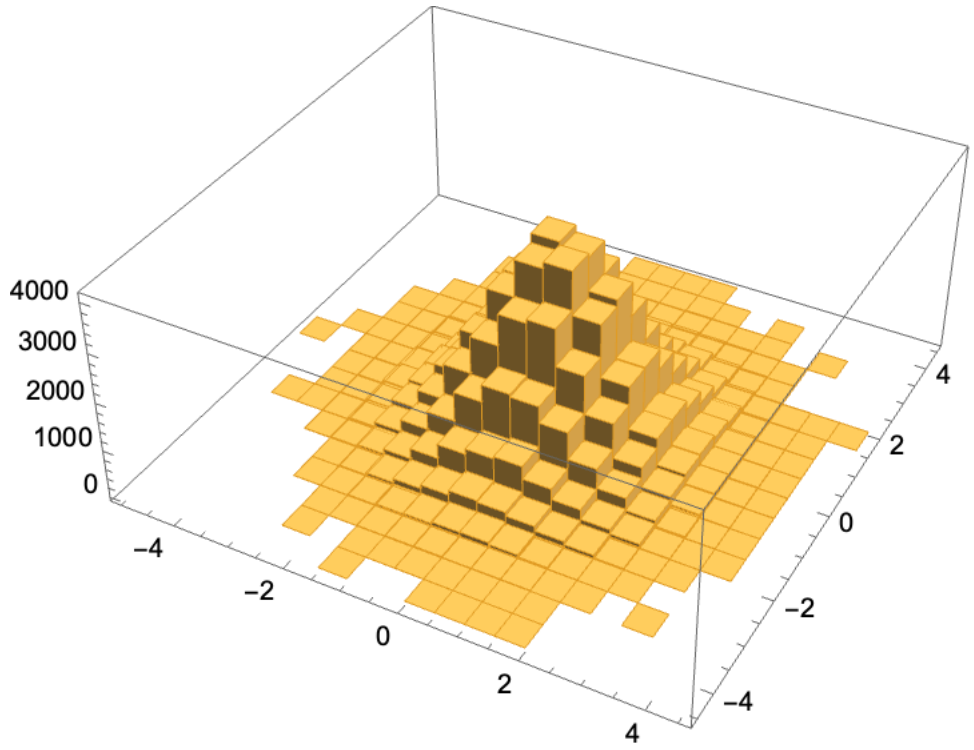
Figure 4. The record of 10,000 tosses of an ideal coin (described in section 6).

- start
- end






4
2
0
−2
−4
−4
−2
0
2
4


4000
3000
2000
1000
0
−4
−2
0
2
4
4
2
0
−2
−4

14. Affine-invariant MCMC algorithms

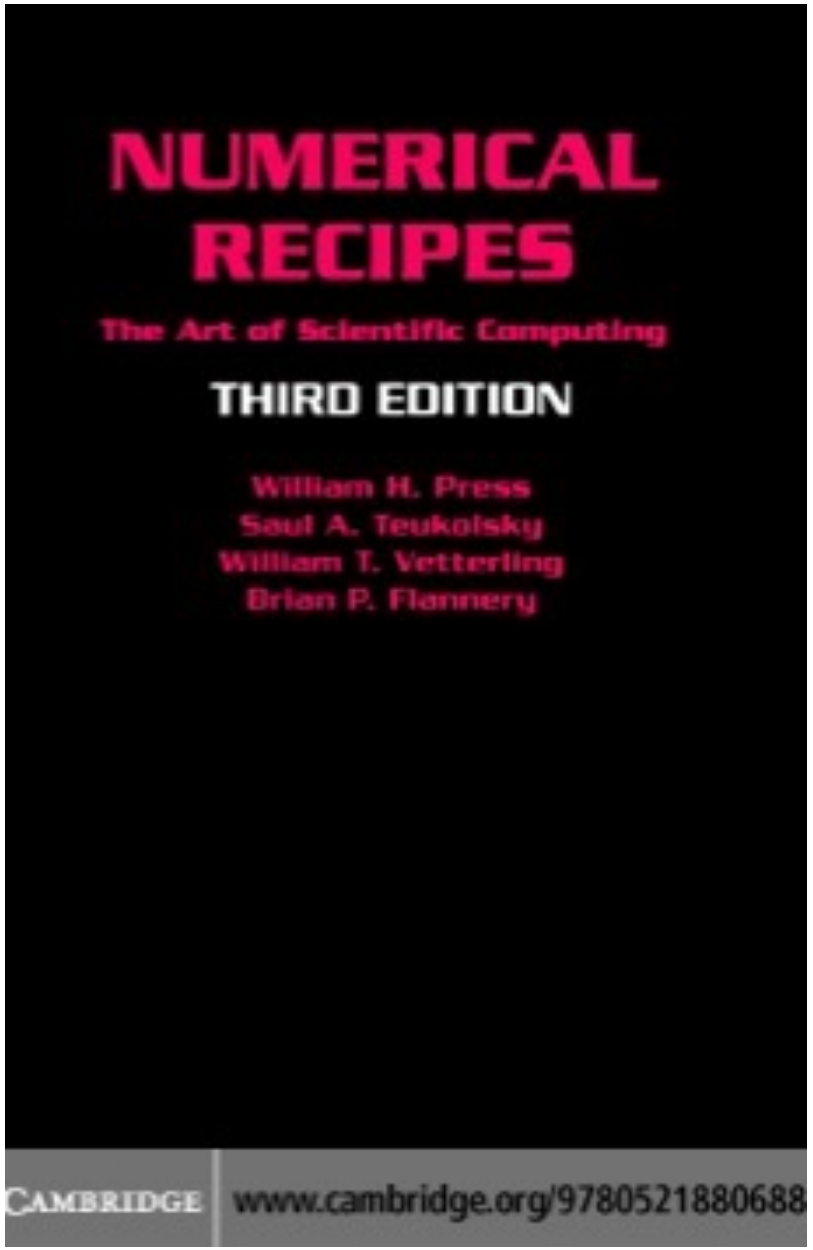
PUBLICATIONS OF THE ASTRONOMICAL SOCIETY OF THE PACIFIC, **125**:306–312, 2013 March

`emcee`: The MCMC Hammer

DANIEL FOREMAN-MACKEY,[1] DAVID W. HOGG,[1,2] DUSTIN LANG,[3,4] AND JONATHAN GOODMAN[5]

Received 2013 January 09; accepted 2013 January 30; published 2013 February 25

ABSTRACT. We introduce a stable, well tested Python implementation of the affine-invariant ensemble sampler for Markov chain Monte Carlo (MCMC) proposed by Goodman & Weare (2010). The code is open source and has already been used in several published projects in the astrophysics literature. The algorithm behind `emcee` has several advantages over traditional MCMC sampling methods and it has excellent performance as measured by the autocorrelation time (or function calls per independent sample). One major advantage of the algorithm is that it requires hand-tuning of only 1 or 2 parameters compared to $\sim N^2$ for a traditional algorithm in an N-dimensional parameter space. In this document, we describe the algorithm and the details of our implementation. Exploiting the parallelism of the ensemble method, `emcee` permits *any* user to take advantage of multiple CPU cores without extra effort. The code is available online at http://dan.iel.fm/emcee under the GNU General Public License v2.



10.5 Downhill Simplex Method in Multidimensions

With this section we begin consideration of multidimensional minimization, that is, finding the minimum of a function of more than one independent variable. This section stands apart from those that follow, however: All of the algorithms after this section will make explicit use of a one-dimensional minimization algorithm as a part of their computational strategy. This section implements an entirely self-contained strategy, in which one-dimensional minimization does not figure.

The *downhill simplex method* is due to Nelder and Mead [1]. The method requires only function evaluations, not derivatives. It is not very efficient in terms of the number of function evaluations that it requires. Powell's method (§10.7) or the DFP method with finite differences (§10.9) is almost surely faster in all likely applications. However, the downhill simplex method may frequently be the *best* method to use if the figure of merit is "get something working quickly" for a problem whose computational burden is small.

The method has a geometrical naturalness about it that makes it delightful to describe or work through:

A *simplex* is the geometrical figure consisting, in N dimensions, of $N+1$ points (or vertices) and all their interconnecting line segments, polygonal faces, etc. In two dimensions, a simplex is a triangle. In three dimensions, it is a tetrahedron, not necessarily the regular tetrahedron. (The *simplex method* of linear programming, described in §10.10, also makes use of the geometrical concept of a simplex. Otherwise it is completely unrelated to the algorithm that we are describing in this section.) In general we are only interested in simplexes that are nondegenerate, i.e., that enclose a finite inner N-dimensional volume. If any point of a nondegenerate simplex is taken as the origin, then the N other points define vector directions that span the N-dimensional vector space.

In one-dimensional minimization, it was possible to bracket a minimum, so that the success of a subsequent isolation was guaranteed. Alas! There is no analogous procedure in multidimensional space. For multidimensional minimization, the best we can do is give our algorithm a starting guess, that is, an N-vector of independent variables as the first point to try. The algorithm is then supposed to make its own way downhill through the unimaginable complexity of an N-dimensional topography, until it encounters a (local, at least) minimum.

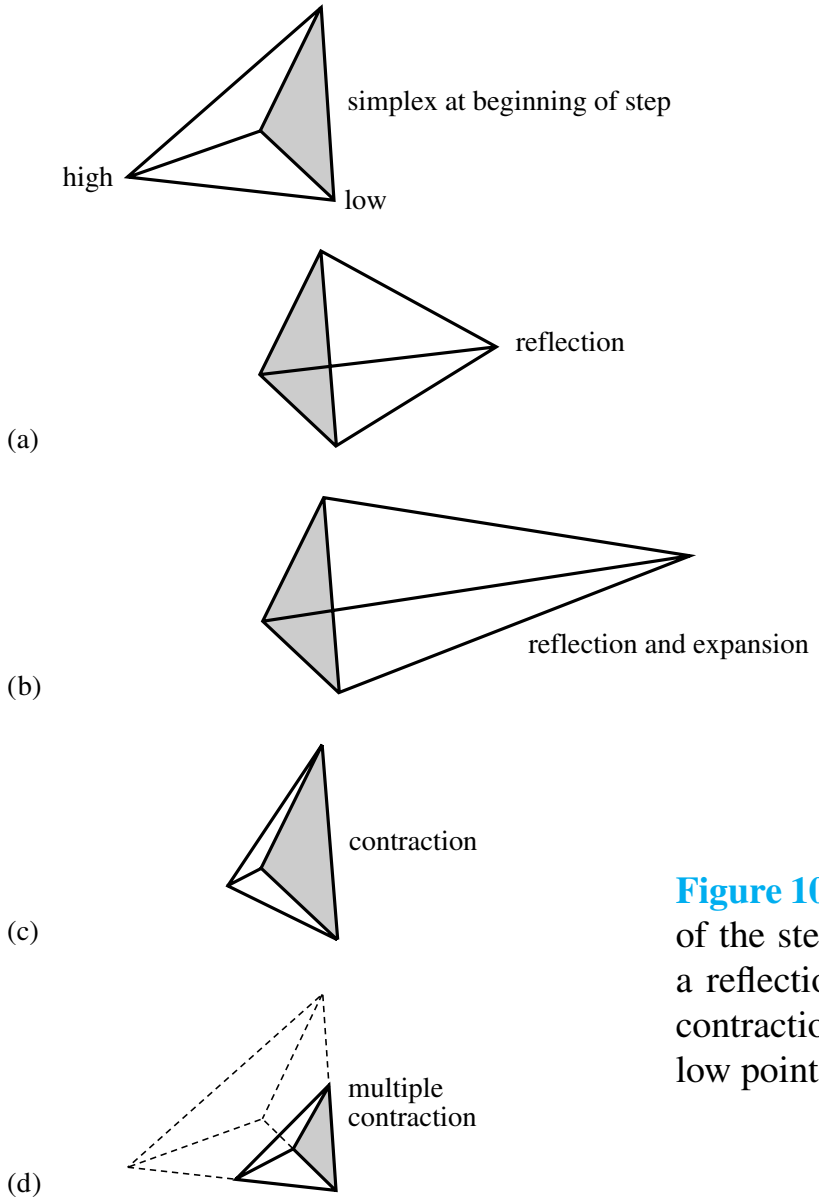
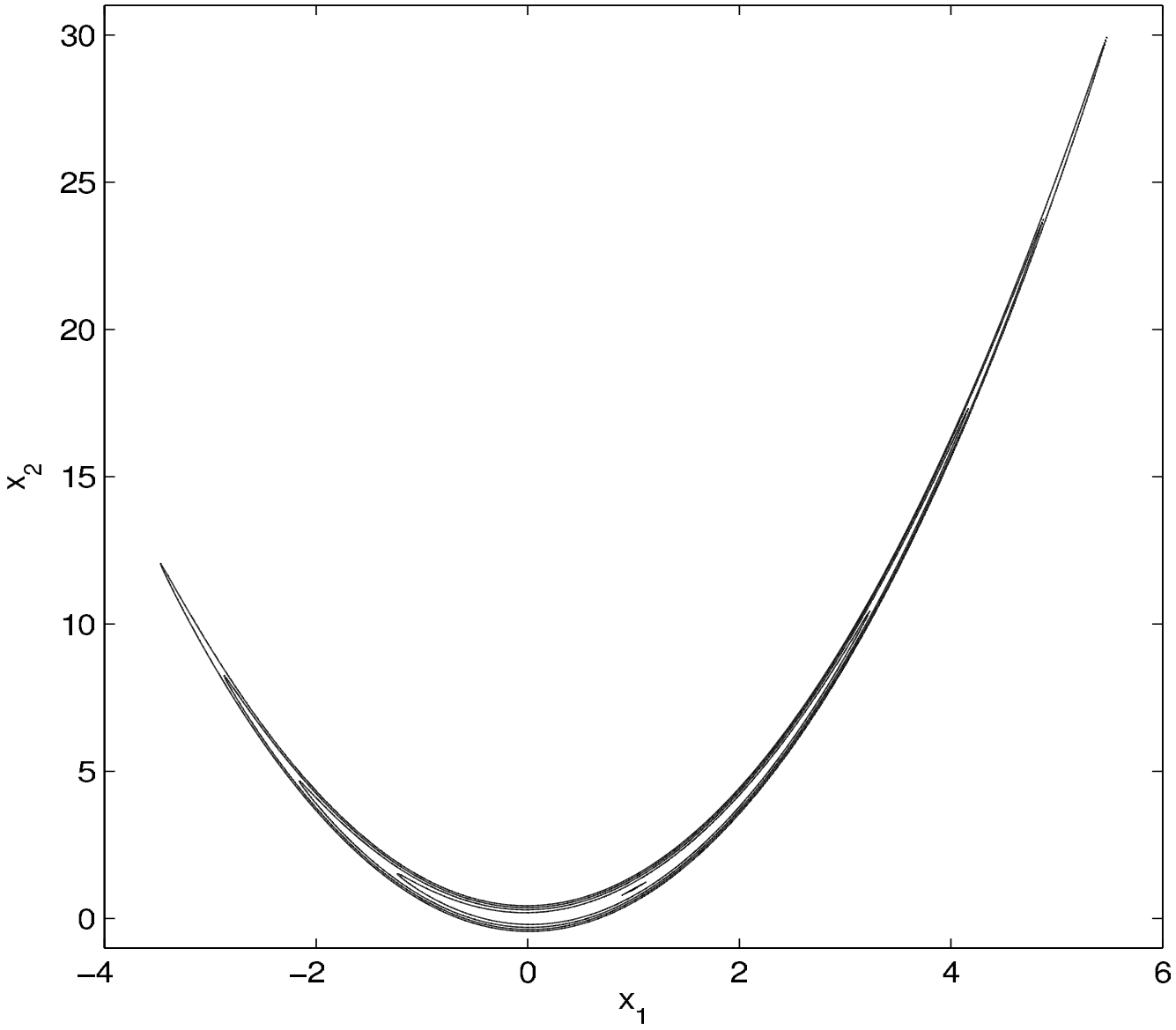


Figure 10.5.1. Possible outcomes for a step in the downhill simplex method. The simplex at the beginning of the step, here a tetrahedron, is shown, top. The simplex at the end of the step can be any one of (a) a reflection away from the high point, (b) a reflection and expansion away from the high point, (c) a contraction along one dimension from the high point, or (d) a contraction along all dimensions toward the low point. An appropriate sequence of such steps will always converge to a minimum of the function.



An example of hard problem: the Rosenbrock density

$$\pi(x_1, x_2) \propto \exp\left(-\frac{100(x_2 - {x_1}^2)^2 + (1 - x_1)^2}{20}\right)$$

- Optimizing a MCMC in a given parameter space often means that we use a proposal distribution that is tuned to the target distribution.

- This proposal distribution is often a multivariate Gaussian with an *n* x *n* covariance matrix that must be tuned accordingly

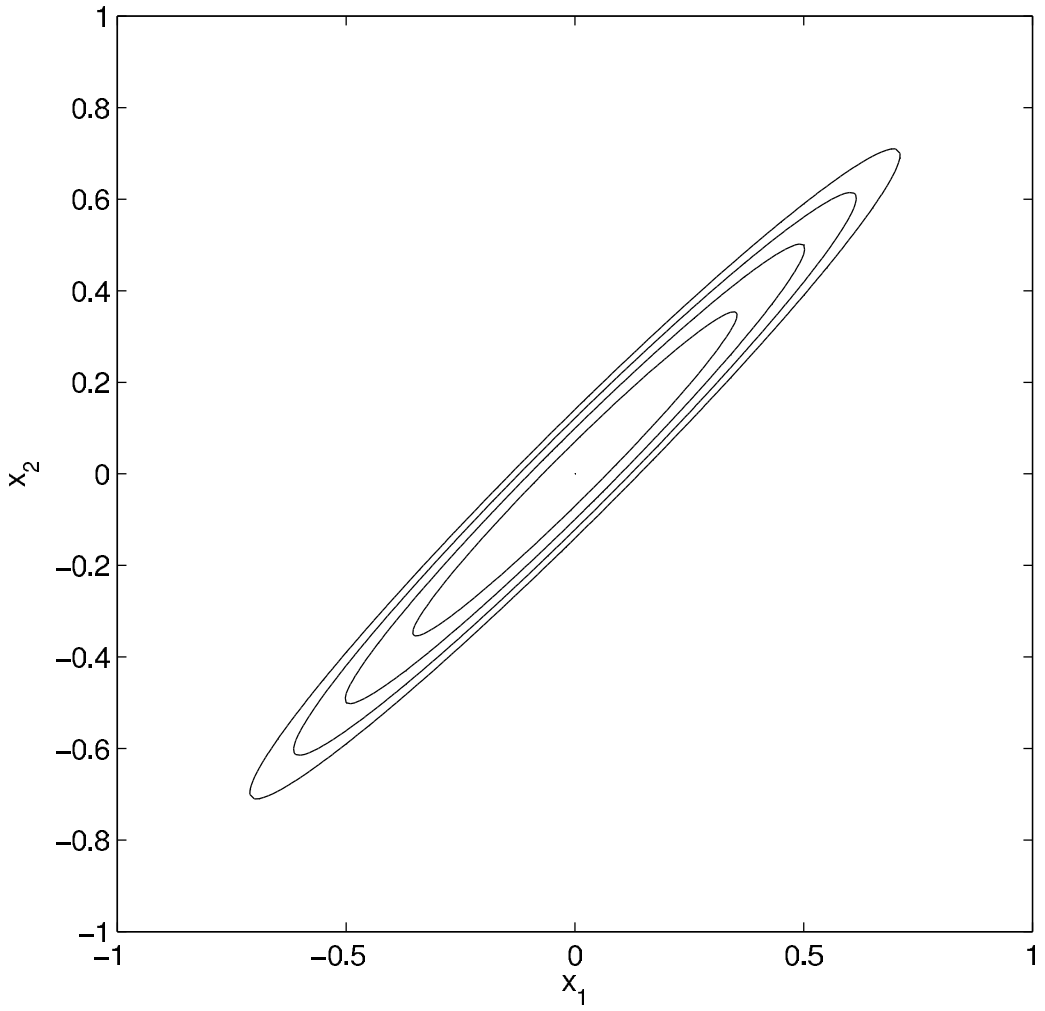
$$p(x_1, \ldots, x_n) = \frac{1}{(2\pi)^{n/2} |\det V|^{1/2}} \exp\left(-\frac{1}{2}\mathbf{x}^T V^{-1} \mathbf{x}\right)$$

with

$$V = \begin{pmatrix} \sigma_1^2 & \rho_{1,2}\sigma_1\sigma_2 & \cdots & \rho_{1,n}\sigma_1\sigma_n \\ \rho_{1,2}\sigma_1\sigma_2 & \sigma_2^2 & \cdots & \rho_{2,n}\sigma_2\sigma_n \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{1,n}\sigma_1\sigma_n & \rho_{2,n}\sigma_2\sigma_n & \cdots & \sigma_n^2 \end{pmatrix}$$

- The covariance matrix has $\frac{1}{2}n(n+1)$ indipendent elements;

tuning the proposal distribution means tuning these independent elements (hyperparameters) with a long (and computationally expensive) burn-in phase.



Consider the following highly anisotropic pdf

$$p(\mathbf{x}) \propto \exp\left(-\frac{(x_1 - x_2)^2}{2\sqrt{\epsilon}} - \frac{(x_1 + x_2)^2}{2}\right)$$

With the variable transformation

$$y_1 = \frac{x_1 - x_2}{\sqrt{\epsilon}}, \quad y_2 = x_1 + x_2$$

which has the Jacobian

$$J(\mathbf{y}, \mathbf{x}) = \begin{vmatrix} \frac{1}{\sqrt{\epsilon}} & -\frac{1}{\sqrt{\epsilon}} \\ 1 & 1 \end{vmatrix} = \frac{2}{\sqrt{\epsilon}}$$

Then, we find that this affine transformation transforms the original Gaussian into the simpler Gaussian

$$p(\mathbf{y}) \propto \exp\left(-\frac{y_1^2}{2} - \frac{y_2^2}{2}\right)$$

In the n-dimensional parameter space there are only 2 hyperparameters to tune (mean, variance) instead of $\frac{1}{2}n(n+1)$

Affine-invariance

Here, we consider MCMC proposal moves of the form

$$X(t+1) = R\left[X(t), \xi(t), \pi\right]$$

proposal function

vector of i.i.d. random variables

target distribution

These proposals are affine-invariant if the following condition holds

$$R\left[Ax + b, \xi(t), \pi\right] = A\ R\left[x, \xi(t), \pi\right] + b$$

for every x.

In this paper we propose a family of affine invariant *ensemble* samplers. An ensemble, $\vec{X}$, consists of L *walkers*[2] $X_k \in \mathbb{R}^n$. Since each walker is in $\mathbb{R}^n$, we may think of the ensemble $\vec{X} = (X_1, \ldots, X_L)$ as being in $\mathbb{R}^{nL}$. The target probability density for the ensemble is the one in which the walkers are independent and drawn from π, that is,

$$\Pi(\vec{x}) = \Pi(x_1, \ldots, x_L) = \pi(x_1)\,\pi(x_2)\cdots\pi(x_L). \tag{5}$$

An ensemble MCMC algorithm is a Markov chain on the state space of ensembles. Starting with $\vec{X}(1)$, it produces a sequence $\vec{X}(t)$. The ensemble Markov chain can preserve the product density (5) without the individual walker sequences $X_k(t)$ (as functions of t) being independent, or even being Markov. This is because the distribution of $X_k(t+1)$ can depend on $X_j(t)$ for $j \neq k$.

We apply an affine transformation to an ensemble by applying it separately to each walker:

$$\vec{X} = (X_1, \ldots, X_L) \xrightarrow{A,b} (AX_1 + b, \ldots, AX_L + b) = (Y_1, \ldots, Y_L) = \vec{Y}. \tag{6}$$

[2]Here x_k is walker k in an ensemble of L walkers.

Goodman and Weare, 2010

The affine transformation is implemented with the following proposal move (*stretch move*), with an ensemble of K walkers $\{X_k\}$

$$X_k(t) \rightarrow Y = X_j + Z[X_k(t) - X_j]$$

where the index *j* is drawn at random from the set of all the all the indexes excluding *k*, and Z is a random variable from a distribution *g* such that

$$g\left(\frac{1}{z}\right) = zg(z) \quad \Rightarrow \quad g(z) \propto \begin{cases} \frac{1}{\sqrt{z}} & \text{if } z \in \left[\frac{1}{a}, a\right] \\ 0 & \text{otherwise} \end{cases}$$

Goodman and Weare (2010)

and acceptance probability

$$\alpha = \min\left(1, Z^{N-1}\frac{\pi(Y)}{\pi(X_k(t))}\right)$$

which satisfies detailed balance. For derivations of the formulas given here, see Roberts and Gilks, J. Multivariate Analysis **49** (1994) 287.

In addition to *stretch moves*, affine-invariant MCMCs also incorporate *walk moves* and *replacement moves* (Goodman and Weare, 2010)

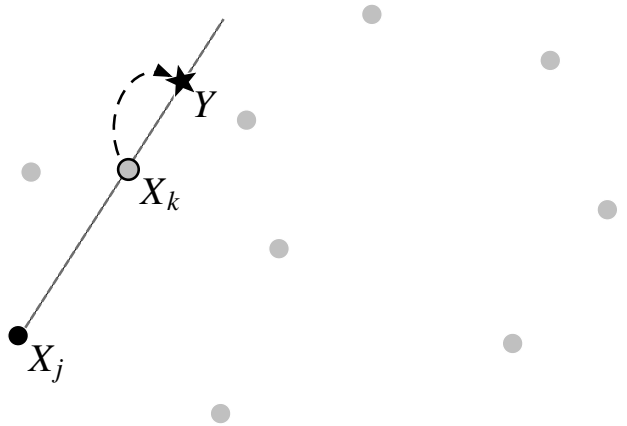


Figure 2. A stretch move. The light dots represent the walkers not participating in this move. The proposal is generated by stretching along the straight line connecting X_j to X_k.

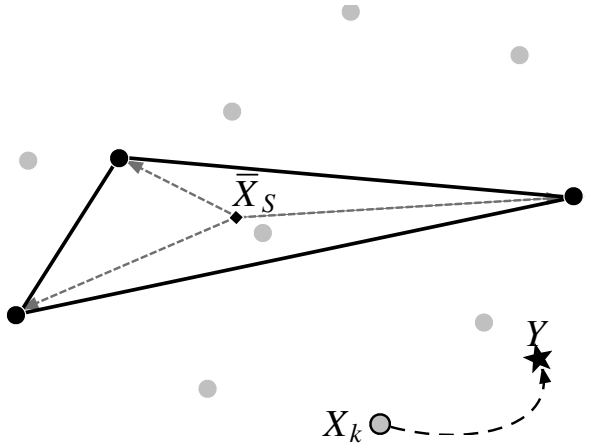


Figure 3. A walk move. The dots represent the ensemble of particles. The dark ones represent the walkers in $\vec{X}_S$. The diamond inside the triangle represents the sample mean $\bar{X}_S$. The proposed perturbation has covariance equal to the sample covariance of the three dark dots. The perturbation is generated by summing random multiples of the arrows from $\bar{X}_S$ to the vertices of the triangle.

https://emcee.readthedocs.io/en/stable/

emcee

emcee is an MIT licensed pure-Python implementation of Goodman & Weare's Affine Invariant Markov chain Monte Carlo (MCMC) Ensemble sampler and these pages will show you how to use it.

This documentation won't teach you too much about MCMC but there are a lot of resources available for that (try this one). We also published a paper explaining the emcee algorithm and implementation in detail.

emcee has been used in quite a few projects in the astrophysical literature and it is being actively developed on GitHub.

Basic Usage

If you wanted to draw samples from a 5 dimensional Gaussian, you would do something like:

```
import numpy as np
import emcee

def log_prob(x, ivar):
    return -0.5 * np.sum(ivar * x ** 2)

ndim, nwalkers = 5, 100
ivar = 1. / np.random.rand(ndim)
p0 = np.random.randn(nwalkers, ndim)

sampler = emcee.EnsembleSampler(nwalkers, ndim, log_prob, args=[ivar])
sampler.run_mcmc(p0, 10000)
```

A more complete example is available in the Quickstart tutorial.

https://github.com/dfm/emcee

README | Code of conduct | MIT license

Documentation

Read the docs at emcee.readthedocs.io.

Attribution

Please cite Foreman-Mackey, Hogg, Lang & Goodman (2012) if you find this code useful in your research. The BibTeX entry for the paper is:

```
@article{emcee,
   author = {{Foreman-Mackey}, D. and {Hogg}, D.~W. and {Lang}, D. and {Goodman}, J.},
    title = {emcee: The MCMC Hammer},
  journal = {PASP},
     year = 2013,
   volume = 125,
    pages = {306-312},
   eprint = {1202.3665},
      doi = {10.1086/670067}
}
```

License

Quickstart

▶ Show code cell content

The easiest way to get started with using emcee is to use it for a project. To get you started, here's an annotated, fully-functional example that demonstrates a standard usage pattern.

How to sample a multi-dimensional Gaussian

We're going to demonstrate how you might draw samples from the multivariate Gaussian density given by:

$$p(\vec{x}) \propto \exp\left[-\frac{1}{2}(\vec{x}-\vec{\mu})^{\mathrm{T}}\,\Sigma^{-1}\,(\vec{x}-\vec{\mu})\right]$$

where $\vec{\mu}$ is an N-dimensional vector position of the mean of the density and Σ is the square N-by-N covariance matrix.

The first thing that we need to do is import the necessary modules:

```
import numpy as np
```

Then, we'll code up a Python function that returns the density $p(\vec{x})$ for specific values of $\vec{x}$, $\vec{\mu}$ and Σ^{-1}. In fact, emcee actually requires the logarithm of p. We'll call it `log_prob`:

```
def log_prob(x, mu, cov):
    diff = x - mu
    return -0.5 * np.dot(diff, np.linalg.solve(cov, diff))
```