1 The MultiNest Algorithm

The MultiNest algorithm is a Bayesian inference tool for parameter space exploration and model selection that has come to widespread use in Astrophysics and Cosmology over the past years.

It builds upon the nested sampling technique producing the Bayesian evidence by integrating the likelihood associated with a given set of parameters over the multidimensional parameter space, and creates samples of the posterior distribution as a by-product.

The novel way in which MultiNest represents the sampled volume by an optimized set of ellipsoids makes it especially powerful for exploring multi-modal posteriors or posteriors with curving degeneracies in high dimensions.

Our implementation follows the full description of the algorithm in Feroz et al. [2009].

2 Performance

2.1 Locate the lighthouse

While implementing our project, we used a simple (unimodal) problem from the literature to create a working infrastructure for all neccessary user and data interactions before integrating the algorithmical modules of MultiNest. The following problem, which uses a likelihood function derived from observed data, was taken from Sivia and Skilling [2006]:

> A lighthouse is somewhere off a piece of straight coastline at position α along the shore and a distance β out to sea. It emits a series of highly collimated flashes at random intervals and hence random azimuths. These pulses are intercepted on the coast by photo-detectors that record only the fact that a flash has occurred, but not the angle from which it came. N flashes have so far been recorded at positions $\{x_k\}$. Where is the lighthouses?

We are told $-2 < \alpha < 2$ and $0 < \beta < 2$, so we assume uniform priors for α and β in these ranges. For a given flash measurement x_k, it can be shown that

$$\text{prob}(x_k|\alpha,\beta) = \frac{\beta}{\beta^2 + (x_k - \alpha)^2},$$

which says the probability that the k^{th} flash will be measured at x_k, given the lighthouse is located at (α, β), follows the Cauchy distribution. Thus, the likelihood function is given by

$$\mathcal{L}(\alpha,\beta) = \prod_{k=1}^{N} \text{prob}(x_k|\alpha,\beta).$$

Simulated data of $N = 64$ pulses are stored in `lighthouse.dat` in the `DataFiles` directory. With our program, we find $\alpha = 1.24 \pm 0.19$, $\beta = 0.97 \pm 0.16$, and $\log(\mathcal{Z}) = -161.49 \pm 0.05$, which is consistent with Sivia and Skilling [2006]. Figure 1 shows the joint posterior distribution for α and β.

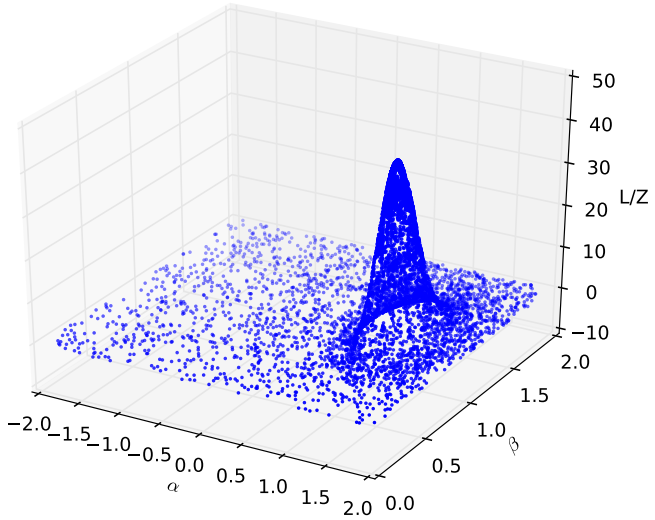

Figure 1: Posterior probability distribution of the lighthouse location. We used 1000 active points.

2.2 Toy model 1: "egg-box"

Although our MultiNest implementation is currently lacking one major feature from Feroz et al. [2009], mode identification (described in section 5.6), we can use the test cases presented there to test our code's performance.

The first toy model is called the egg-box likelihood. It is designed to show MultiNest's performance in highly multimodal problems. Instead of deriving the likelihood function from observed data and a realistic model, it is given analytically:

$$L(\theta_1, \theta_2) = \exp\left[\left\{2 + \cos\left(\frac{\theta_1}{2}\right) + \cos\left(\frac{\theta_2}{2}\right)\right\}^5\right]$$

The likelihood function and its sampling returned by a run of our code with

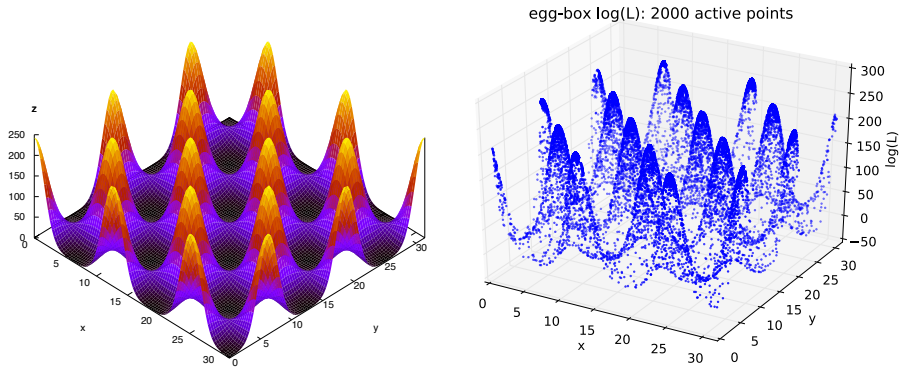

Figure 2: Analytical and sampled likelihood function of the eggbox toy model

2000 points is shown in figure 2.2.

Given the analytical likelihood, the total log-evidence can be numerically integrated to 235.88. Our algorithm gives a result of 235.57 ± 0.06 which is too low. Presumably, a bug either in the uniform sampling or in the final integration after convergenge limits our accuracy. This deviation will be investigated further.

2.3 Toy model 2: "gaussian shells"

The second toy model in Feroz et al. [2009] tests MultiNest's performance with curving degeneracies and high dimensionalities. The analytic likelihood function is given by

$$L(\theta) = \mathrm{circ}(\theta; c_1, r_1, w_1) + \mathrm{circ}(\theta; c_2, r_2, w_2)$$

where

$$\mathrm{circ}(\theta; c, r, w) = \frac{1}{\sqrt{2\pi w^2}} \exp\left[-\frac{(|\theta - c| - r)^2}{2w^2}\right]$$

The two dimensional appearance of this likelihood function is shown in figure 2.3.

The total log-likelihood evaluates to -1.75, while our algorithm returns -1.41 ± 0.03 with 2000 points.

2.4 Increasing Dimensionality and Sampling Efficiency

The gaussian shell-problem can be easily expanded to arbitrary dimensions, and used to show the unparalleled scaling behaviour of the number of likelihood evaluation MultiNest needs to reach a given accuracy and the sampling efficiency (successfully finding a point of higher likelihood value than the current lowest from the set of ellipsoids) it achieves.

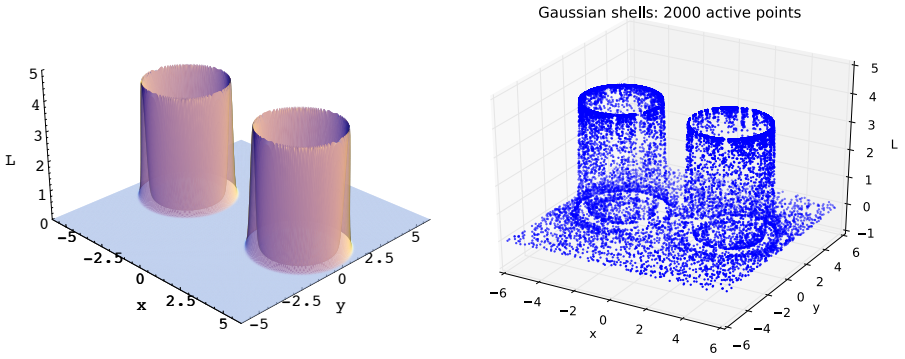

Figure 3: Analytical and sampled likelihood function of the gaussian-shells toy model

	Feroz et al. [2009]		Own Implementation	
D	N_{like}	Efficiency	N_{like}	Efficiency
2	7370	0.7077	75980	0.3422
5	17967	0.5102		
10	52901	0.3428		

Table 1: Comparison between published and own performance of number of likelihood evaluations and sampling efficiency in gaussian-shell-problems of varying dimensionality

2.5 Profiled Run

Using gprof, the following flat profile was found for a MultiNest run of the eggbox problem with 2000 points:

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
35.43	6.23	6.23	388786	0.02	0.02	KMeans(std::vector<Point*, std::alloca
19.48	9.66	3.43	1481594	0.00	0.00	FindEnclosingEllipsoid(std::vector<Poi
16.15	12.50	2.84	218581886	0.00	0.00	Ellipsoid::mdist(Point*)
12.17	14.64	2.14	3128	0.68	5.10	Samplers::EllipsoidalPartitioning(std
6.31	15.75	1.11	1870380	0.00	0.00	SelectFromGrouping(std::vector<Point*,
3.24	16.32	0.57	238084	0.00	0.00	Samplers::get_newcoor()
2.96	16.84	0.52	34052242	0.00	0.00	Ellipsoid::IsMember(gsl_vector*)
1.31	17.07	0.23				main
0.80	17.21	0.14	1677551	0.00	0.00	Ellipsoid::Ellipsoid(int, gsl_vector*,
0.23	17.25	0.04	36676332	0.00	0.00	Ellipsoid::getVol()
0.23	17.29	0.04	3557834	0.00	0.00	std::vector<Point*, std::allocator<Poi
0.11	17.31	0.02	377845	0.00	0.00	unisphere(float*, int)
0.06	17.32	0.01	1677551	0.00	0.00	std::vector<Point*, std::allocator<Poi
0.06	17.33	0.01	377845	0.00	0.00	u_in_hypercube(gsl_vector*, int)
0.06	17.34	0.01	377845	0.00	0.00	Ellipsoid::SampleEllipsoid()
0.06	17.35	0.01	240084	0.00	0.00	Point::transform_prior()
0.00	17.35	0.00	1481594	0.00	0.00	Ellipsoid::~Ellipsoid()
0.00	17.35	0.00	673094	0.00	0.00	Ellipsoid::getEnlFac()
0.00	17.35	0.00	477137	0.00	0.00	Ellipsoid::setEnlFac(double)
0.00	17.35	0.00	240084	0.00	0.00	Data::logL(Point*)
0.00	17.35	0.00	195957	0.00	0.00	Ellipsoid::getCenter()
0.00	17.35	0.00	195957	0.00	0.00	Ellipsoid::getCovMat()
0.00	17.35	0.00	3128	0.00	0.00	Samplers::CalcVtot()
0.00	17.35	0.00	2000	0.00	0.00	Point::set_params(std::vector<std::str
0.00	17.35	0.00	2000	0.00	0.00	Point::hypercube_prior()
0.00	17.35	0.00	2000	0.00	0.00	Point::Point(int)
0.00	17.35	0.00	9	0.00	0.00	std::vector<Ellipsoid*, std::allocator
0.00	17.35	0.00	1	0.00	0.00	global constructors keyed to _Z4distiP
0.00	17.35	0.00	1	0.00	0.00	global constructors keyed to _Z9boxmul
0.00	17.35	0.00	1	0.00	0.00	global constructors keyed to _ZN4Data(
0.00	17.35	0.00	1	0.00	0.00	global constructors keyed to _ZN5Point
0.00	17.35	0.00	1	0.00	0.00	global constructors keyed to _ZN8Sampl
0.00	17.35	0.00	1	0.00	0.00	global constructors keyed to main
0.00	17.35	0.00	1	0.00	0.00	Data::Data(int, int, std::string)

Interestingly, the most time-consuming function is K-means, a simple algorithm used for the preliminary partitioning of the data cloud, before the splitting is optimized using a more sophisticated scheme taking into account the desired ellipsoidal form of the clusters.

3 User Instructions

Compile the code using `make`. This program relies on the GNU Scientific Library (GSL) to do linear algebra, so be sure you have GSL installed and that the compiler knows where to find its source files. Before running the program, run time parameters must be set in a file in the `RunTimeFiles` directory. The format of run time files must be as follows.

`problem/data file`	: `problem or data file name`
$\mathtt{N}_{\mathtt{columns}}$	: #
`Dimension`	: #
$\mathtt{N}_{\mathtt{points}}$	: #
`efficiency factor`	: 1.0 (`default vaule`)
`repartition factor`	: 1.2 (`default value`)
θ_1	: `prior` $\theta_{1,\mathtt{min}}$ $\theta_{1,\mathtt{max}}$
θ_2	: `prior` $\theta_{2,\mathtt{min}}$ $\theta_{2,\mathtt{max}}$

Here, θ_i is the i^{th} parameter, `prior` is the prior probability distribution for a given parameter, and the efficiency and repartition factors are optimization settings that are related to ellipsoidal partitioning and sampling. We have supplied three run time files for the test problems described above. The lighthouse problem is the only test case that requires data for likelihood evaluations, and its data file can be found in the `DataFiles` directory. The current version of the code can only run these three problems, but we plan to adapt it for much more general use.

The program takes the name of the run time file as a command-line argument. From the `GrecoRothe_MultiNest` directory, run the three test cases using the commands below.

```
./multinest eggbox.txt
./multinest gauss_shells.txt
./multinest lighthouse.txt
```

If you would like to run make quick test runs with the code, lower the number of active points (`Npoints`) in the run time files. Although our code can run the egg-box and Gaussian shells problems in an arbitrary number of dimensions, we have focused on 2 dimensions (2D), which allows for easy visualization and requires practical computation times.

If the program runs successfully, it will output `posterior_pdfs.dat`, which has columns θ_1, θ_2, $\log(\mathcal{L})$, and $\mathcal{L}/\mathcal{Z}$. As the name implies, this file contains the necessary data to visualize the desired 2D posterior probability distributions. In addition, the program will output text similar to the following to the command-line.

```
getting runtime parameters
creating 2000 active points in 2 dimensions
running MultiNest algorithm...  this may take a few minutes
job complete!
**** results ****
number iterations = 34001
number reclusters = 2534
information:  H = 8.9353 bits
global evidence:  logZ = 235.683 +/- 0.0556484
***************
```

As promised, the code tells us the evidence $\mathcal{Z}$, plus some additional information about how the algorithm is progressing.

References

F. Feroz, M. P. Hobson, and M. Bridges. MULTINEST: an efficient and robust Bayesian inference tool for cosmology and particle physics. *MNRAS*, 398: 1601–1614, October 2009. doi: 10.1111/j.1365-2966.2009.14548.x.

D. S. Sivia and J. Skilling. *Data Analysis: A Bayesian Tutorial*. Oxford University Press, 2 edition, 2006.