Monte Carlo Computer Practical

Monte Carlo methods in a nutshell: A short course

October 2012
M. Sambridge

Each of the following problems illustrate an aspect of Monte Carlo methods, namely numerical integration, error analysis and parameter search, and Bayesian sampling. Each requires you to write some computer code. Feel free to use whatever programming language you feel most comfortable with. We'll focus on doing them with MATLAB and python in the practical class, but you could try on your own using another programming language if you prefer.

Problem 1: Monte Carlo integration. The mass of an irregularly shaped object

With Monte Carlo integration any type of integral can be evaluated numerically. Figure 1 shows the example of a set of circular objects inside a square box B (with side $= 2m$ and area $= 4m^2$). If the thickness of the plate is a constant, $z = 1m$, then the mass is given by the integral of its density over the volume of the disc. The purpose of this exercise is to evaluate the mass of various irregularly shaped objects using numerical integration.

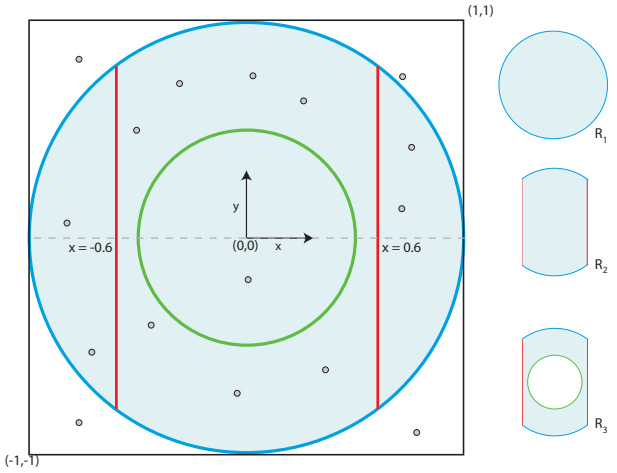

Figure 1: Monte Carlo integration in complex domains by throwing random darts.

1. If density is a constant (1kg/m^3) the mass, M, of the circular plate with blue outline (region R_1) is given by an integral over the square box, B,

$$M = \int_B d(x, y) dx dy \tag{1}$$

where $d(x, y)$ is the density as a function of position and given by

$$\begin{aligned} d(x, y) &= 1 \quad \text{if (x,y) lies inside chosen region} \\ &= 0 \quad \text{Otherwise} \end{aligned}$$

From simple geometry the mass of the circular plate is equal to $\pi r^2 z$. With thickness $z = 1$ and radius $r = 1$ this gives a mass of π kg. Write a program to evaluate the mass using Monte Carlo integration and see how quickly it converges to this value.

Procedure:

(a) First generate N random points inside the box B and count those that fall inside the circular region R_1, call this value h, i.e. $h = \sum_{i=1}^{N} d(x_i, y_i)$.

(b) Recall that the Monte Carlo estimate of the integral (1) is given by

$$M_{mc} = \sum_{i=1}^{N} \frac{d(x_i, y_i)}{\rho_i} \tag{2}$$

where (x_i, y_i) is the position of the ith random sample and ρ_i is the sampling density at that point. Since the box has volume $= 4m^3$ and there are N samples the density is uniform we have $\rho_i = N/4$, which gives

$$M_{mc} = \frac{4h}{N} \tag{3}$$

Evaluate the Monte Carlo approximation to the mass as a function of the number of points, N, and see how quickly this converges to the correct answer.

2. Repeat the Monte Carlo integration to find the mass of the irregular objects contained within the blue circle and red vertical lines at $x = \pm 0.6m$ (region R_2 in figure 1),

3. Now do the same again within the blue circle and red vertical lines but outside the green circle with radius $r = 0.5m$ (region R_3 in figure 1).

4. Repeat the integral for region R_3 but with density varying inversely with radius, $d(x, y) = 0.1(x^2 + y^2)^{1/2}$

5. You could also plot the 1 σ error for each integral using the formula in the notes. Recall we have the general formula

$$\sigma_{MC} = \left[\sum_{i=1}^{N} \frac{d(x_i, y_i)^2}{\rho_i^2} - \frac{1}{N} \left(\sum_{i=1}^{N} \frac{d(x_i, y_i)}{\rho_i} \right)^2 \right]^{1/2} \tag{4}$$

Substituting from above we have

$$\sigma_{MC} = \frac{4}{N} \left[h(1 - \frac{h}{N}) \right]^{1/2} \tag{5}$$

Even though there is no simple analytical solution for the integrals over shapes R_2 and R_3, they are just as easy to evaluate with Monte Carlo integration by rejecting the random points which fall outside the required region.

Problem 2: Error propagation with the Bootstrap: The cannonball problem

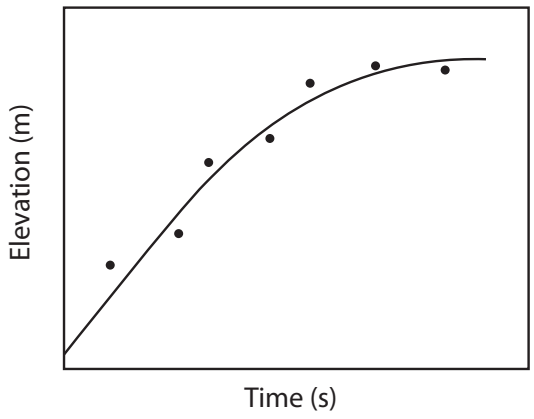

Figure 2: Cannon heights as a function of time.

A cannon ball is fired directly upwards from an unknown starting height above the surface, m_1, with unknown initial velocity, m_2 and unknown gravitational acceleration, m_3 . Newton's laws of motion tell us that the relationship between position and time follows

$$y(t) = m_1 + m_2 t - \frac{1}{2} m_3 t^2. \tag{6}$$

An experiment has been performed and heights, y_i, $(i = 1, \ldots, 8)$ are collected at fixed time intervals of a second. We obtain the data

y	t
26.94	1.0
33.45	2.0
40.72	3.0
42.32	4.0
44.30	5.0
47.19	6.0
43.33	7.0
40.13	8.0

To find the unknowns (m_1, m_2, m_3) we must fit a quadratic curve (equation 2) to the observed data (see figure 2). This can be achieved by evaluating the following expression

$$\begin{pmatrix} m_1 \\ m_2 \\ m_3 \end{pmatrix} = \begin{pmatrix} \mathrm{N} & \sum t_i & -1/2 \sum t_i^2 \\ \sum t_i & \sum t_i^2 & -1/2 \sum t_i^3 \\ -1/2 \sum t_i^2 & -1/2 \sum t_i^3 & 1/4 \sum t_i^4 \end{pmatrix}^{-1} \begin{pmatrix} \sum y_i \\ \sum t_i y_i \\ -1/2 \sum t_i^2 y_i \end{pmatrix} \tag{7}$$

where $N = 8$ is the number of data, (y_i, t_i) represent the ith data pair and each summation is over all N data. All terms on the right hand side of equation (7) are known and so its a simple case of plugging in values to determine the best fit estimates of (m_1, m_2, m_3). **Write a computer code** to calculate the best

fit values of the three unknowns (height, velocity and gravitational acceleration). We call these values (m_1^0, m_2^0, m_3^0) our **solution**. By the way can you guess where this experiment took place ?

The problem now is to use the **bootstrap** to determine how error in the data propagate into the estimated unknowns. We do not know the size of errors in the data but we can apply the bootstrap. Since the data are associated with increasing time it does not make sense to directly resample the data (because we could end up with two heights of the same value associated with different times). The data are not IID, since they belong to a trend. However we can still proceed by applying the bootstrap principle to the data residuals produced by the best fit solution., i.e. we have 8 residuals, r_i, where

$$r_i = y_i - m_1^0 - m_2^0 t_i + \frac{1}{2} m_3^0 t_i^2. \quad (i = 1, \ldots, 8).$$

If we assume that the residuals are IID they can be re-sampled with replacement in the usual way to form multiple sets of 8 residual values $r_j^*, (j = 1, \ldots, 8)$ and new bootstrap data are constructed using this set of residuals by

$$y_j^* = r_j^* + m_1^0 + m_2^0 t_j - \frac{1}{2} m_3^0 t_j^2. \quad (j = 1, \ldots, 8).$$

Using this approach the residuals are mixed between different data, and so each y values does not simply get its own residual back. **Write a computer code** to build bootstrap data sets and for each of these insert them into the above expressions to calculate the bootstrap estimates of the unknowns. Lets call these $(m_1^i, m_2^i, m_3^i), (i = 1, \ldots, B)$. The number of bootstrap samples B is your choice but it should be at least 100.

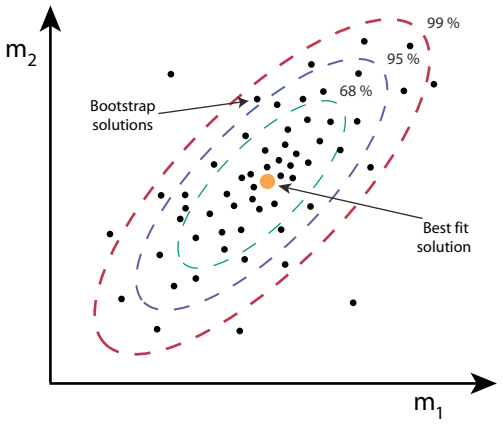

Figure 3: Bootstrap solutions and confidence intervals distributed about the best fit solution.

It can be instructive to **plot the bootstrap samples** as a scatter plot for the three pair of variables, i.e. (m_1^i, m_2^i), (m_2^i, m_3^i) and (m_1^i, m_3^i), $(i = 1, \ldots, B)$. They should look something like figure 3.

From the bootstrap output samples $(m_1^i, m_2^i, m_3^i), (i = 1, \ldots, B)$ and the formulae in your handouts calculate the i) **the mean**, ii) **the variance**, iii) **the bias corrected solution**, and iv) **the 95% confidence intervals** for each of the three unknowns . The mean should look similar to the best fit values and the bias should be small. The variance and confidence intervals characterize the error in the estimated values of the unknowns.

Problem 3: Parameter search with nested grids

Suppose we have two unknowns, (x_1, x_2). We measure x_1 directly and obtain a value of $1 \pm \sigma$ (where the measurement error is $\sigma = 10$). Suppose also that we know the following property should hold, $x_1^2 = x_2$. We wish to find the best compromise values of (x_1, x_2) which both fit the data and the extra constraint. We could solve this problem by finding (x_1, x_2) which minimizes the least squares misfit function

$$f(x_1, x_2) = \frac{(1 - x_1)^2}{100} + (x_2 - x_1^2)^2. \tag{8}$$

This is hardly necessary because the solution is fairly obvious, i.e. $x_1 = x_2 = 1$, and indeed this solution corresponds to a global minimum in the misfit function $f(1, 1) = 0$. The misfit surface is plotted in figure 4 and is known as the Rosenbrock function[1]. Although its global minimum is known it can be quite difficult to locate with an optimization algorithm (because it lies at the end of a long valley).

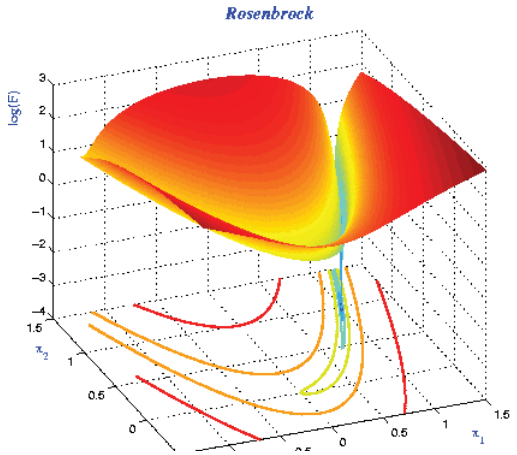

Figure 4: The Rosenbrock data misfit function for a two parameter problem.

1. Find the global minimum of this function using simple uniform Monte Carlo random search. **Write a computer program** to generate random points in a plane in the ranges $-1.5 \leq x_1 \leq 1.5$, and $-1.5 \leq x_2 \leq 1.5$. Calculate $f(x_1, x_2)$ for each of your points and monitor the minimum as a function of the number of points. How many do you need to generate to get a good solution? It can be instructive to plot the minimum in $f(x_1, x_2)$ as a function of the number of points you generate.

2. Now repeat the exercise but stop the process after a fixed number of samples, say N_s (you choose N_s). Then centre a new smaller box with side length L (you choose L, e.g. 1/2 or 3/4 of the length of the original) at your current best solution and repeat the process. You have built a **nested grid parameter search scheme**. Repeat this several times and see if the solution is more accurate and the convergence any quicker than with the single stage approach.

3. If you want a challenge then repeat the problem with multiple unknowns and data. The N variable Rosenbrock function is

$$f(x_1, x_2, \ldots, x_N) = \sum_{i=1}^{N-1} (1 - x_i)^2 + 100(x_{i+1} - x_i^2)^2. \tag{9}$$

[1] Actually the Rosenbrock function is a factor of one hundred larger, i.e. $100 f(x, y)$.

But beware this function has local minima as well as a global minimum.

Problem 4: How much profit can you make from Lotto?

The question refers to how we can use published information about the number of Lotto winners and the likelihood of winning to determine how much profit the owners (and not you) make from a given Lotto game.

The unknown in the problem is the total number of entries (i.e. sets of 6 numbers) sold, which we will call n, from knowledge of the number of winners of each division $d_i, (i = 1, \ldots, N_{div})$, where N_{div} is 6 for the example below. The values of $d_i, (i = 1, \ldots, N_{div})$ are our data and are assumed to be a known set of integers without error. Of course, since both the cost per entry and the total prize money are published then an estimate of n, allows a direct inference on the total profit before costs made by the lotto organisation. (In reality the real value of n is never made public!)

We will attack the problem both from a Frequentist viewpoint, using the data to make a single estimate of n, and also a Bayesian inference viewpoint where we use the data to construct a probability distribution for n.

Background theory

The probability of winning each division, is independent of the total number of entries n, so these may be treated as a set of known constants, $p_i, (i = 1, \ldots, N_{div})$, the value of which depends on the details of the game.

i	1	2	3	4	5	6
d_i	14	169	3059	149721	369543	802016
p_i^{-1}	8145060	678756	36696	732	300	144

Table 1: Tattslotto dividend results for draw number 3253 on 29/09/2012. Total prize pool of \$49.92m, with division 1 prize of \$22m. The cost of a single entry is about \$0.65.

1. A Frequentist solution might be to take the number of winners of each division and divide by the probability of winning to get multiple estimates of n. These estimates are independent and we could average them. Do this for the data above to get an estimate for n. By how much do these estimates vary ?

2. A Bayesian inference approach requires us to find the Likelihood and prior and then multiply them together. Lets assume our prior is uniform between $1 < n < 3 \times 10^8$ which is a safe assumption. The likelihood is the probability of the data given the model, i.e. the probability that there would be d_i winners of division i and $n - d_i$ non winners when there are n tickets sold. The binomial theorem tells us that this probability, $p(d_i|n)$, is given by

$$p(d_i|n) = \frac{n!}{d_i!(n-d_i)!} \times p_i^{d_i}(1-p_i)^{n-d_i} \tag{10}$$

All values in this expression are known except the single unknown n. Since the number of winners in each division provides independent data, the total likelihood is the product of similar terms for each division, i.e.

$$p(\mathbf{d}|n) = \prod_{i=1}^{N_{div}} p(d_i|n) \tag{11}$$

Bayes' theorem says that to find the *a posteriori* probability distribution for the unknown n we just multiply the likelihood by the prior. Since the prior is a constant the result is

$$p(n|\mathbf{d}) \propto \prod_{i=1}^{N_{div}} \frac{n!}{(n-d_i)!} \times (1-p_i)^{n-d_i} \tag{12}$$

which holds for $1 \leq n \leq 3 \times 10^8$. Outside this range the posterior PDF is zero because the prior is zero. Our only interest is in the unknown n and so the constant of proportionality is used to absorb all quantities independent of n.

Your task is to use the values of $(d_i, p_i), i = 1, \ldots, 6$ from the table and plot the probability distribution as a function of n. Do this in the range 112.5m - 114.5m and your figure should look similar to figure 5. Compare this curve to the single frequentist estimate of n you obtained in part 1, what do you notice?

`[Hint: In any computer program it is always best to calculate` $\log p(n|\mathbf{d})$ `first and then take an exponent to evaluate the curve as a function of` n`. Stirling's formulae for the approximation to` $n!$ `may be useful. This is what is done in the example solution script` **lotto.py**`]`

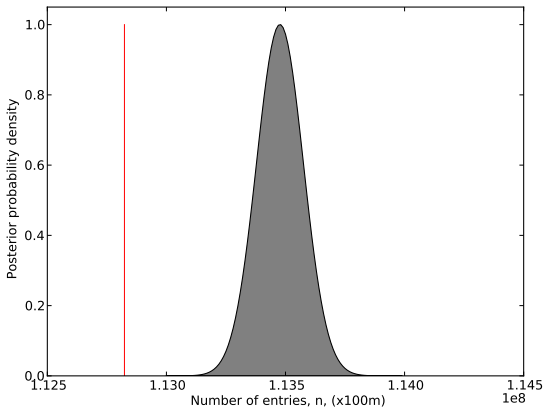

Figure 5: Bayesian posterior PDF for number of tickets sold in Lotto game 3253. The red line is the mean of the estimates obtained from table .

3. Repeat the problem using the Maximum Likelihood (ML) approach. This is done by finding the value of n which maximises the the likelihood expression in eqn. (11). Since the prior is a constant for this problem the likelihood is proportional to the curve you produced in part 2. You could probably do it visually. Plot the average estimate you obtained in part 1 on top of the curve from part 2. How does the ML solution compare to the solution from part 1?

Problem 5: Bayesian inference: Regression

Suppose we have a 2-D data set of noisy (x, y) values and have no idea about the underlying function which produced it (Figure 6). The task is to recover information about the (red) function from the observations. In this exercise you can do this without fixing the complexity (polynomial order) of the curve in advance. Instead the data is used to constrain the number of degrees of freedom in the curve using (transdimensional) Bayesian sampling.

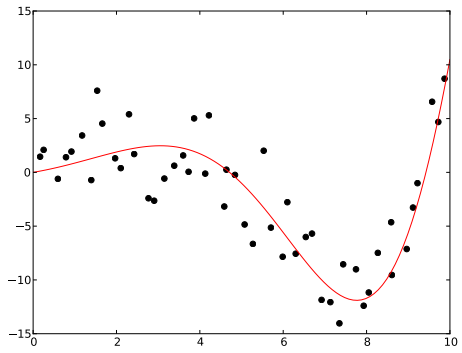

Figure 6: 2-D data set. Red curve is the real function, dots are observed data.

This exercise makes use of Bayesian Markov chain Monte Carlo sampling software contained in the *AuScope-ilab* inversion software library. The library is available as Fortran or C source code with a python interface. A tutorial to the python routines is available on the course website. The tutorial can be used as a guide to carry out the following exercises. In fact you can either try and write python scripts yourself to carry out the exercises below, or just load the solutions, run them and examine what they do.

The task is to estimate the red curve, as well as its uncertainty using the Partition Modelling algorithm described in the lectures. This is implemented in the python software library *rjmcmc*.

Lets assume that the errors are independent and only in the y-co-ordinate and have a Gaussian distribution, with variance σ_i^2. If the data are $y_i, (i = 1, \ldots, n)$ and the model predictions at the same locations are $y_i, (i = 1, \ldots, n)$, then the Likelihood function which measures the success of the model in fitting the data is given by

$$p(\mathbf{d}|\mathbf{m}) = \frac{1}{(2\pi)^{n/2} \prod_{i=1}^{n} \sigma_i} e^{-\sum_{i=1}^{n} [y_i^{obs} - y_i(\mathbf{m})]^2 / \sigma_i^2}. \tag{13}$$

1. Follow instructions in the *rjmcmc library tutorial guide* (file **rjmcmc_tutorial_single.pdf**) to load the given data set of (x_i^{obs}, y_i^{obs}) values and plot the data. (This is done for you by running the script `ch1-loading.py`.)

2. In this exercise we assume a polynomial representation for the unknown function (red curve) with maximum order 5 and a uniform prior PDF.

$$p(\mathbf{m}|\mathbf{d}) \propto \frac{1}{(2\pi)^{n/2} \prod_{i=1}^{n} \sigma_i} e^{-\sum_{i=1}^{n} [y_i^{obs} - y_i(\mathbf{m})]^2 / \sigma_i^2}. \tag{14}$$

   Follow instructions in the *rjmcmc library tutorial guide* to sample from the Bayesian posterior PDF

using an McMC algorithm. Use the 1-D Partition modelling software to generate 50000 curves and take the mean. (This is done for you by running the script `ch2-analyse.py`.)

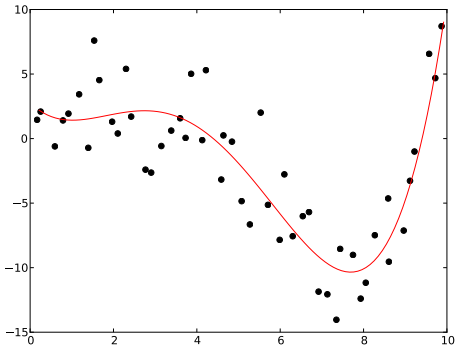

Figure 7: 2-D data set with mean reconstructed model from 50000 McMC samples.

You should get a figure similar to figure 7.

3. In this example the maximum order of the polynomial has been fixed at 5. Use script `ch3-orderanalysis.p` to adjust the maximum order between 0 and 5 and plot the posterior distribution of the order. Plots the two figures showing the mean predicted curve for each case (Figure 8a)and the posterior PDF on the order of the polynomial (Figure 8b). The shows how the data support has detected the degree of

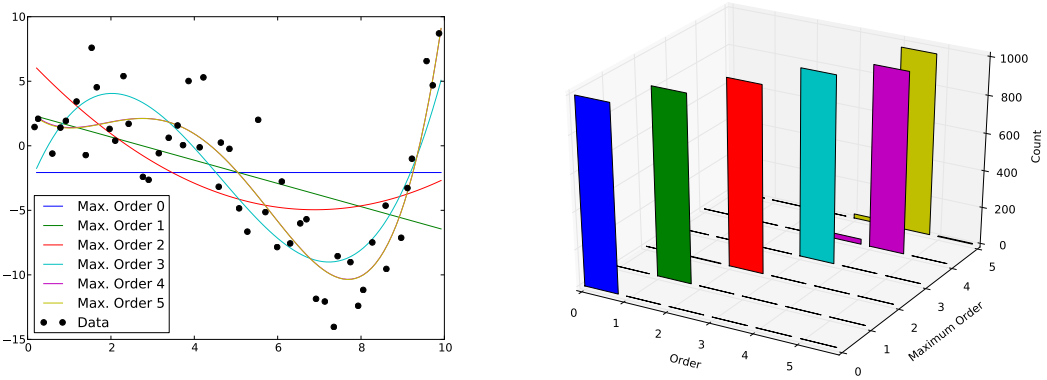

Figure 8: a) (left) Means models for 6 separate McMC runs with different maximum order of polynomial, 0 to 5; b) (right) Posterior probability distribution showing the support of the data for different polynomial orders as a function of maximum order.

4. Repeat the run with maximum order 5 to generate an ensemble of solutions, just as in exercise 2 above, only this time use the library routines to plot a density model of the entire ensemble. In this way we get a visual impression of the error in the predicted curve. You can use script Use script `ch4-confidence.py` to do this and you should get a plot similar to Figure 9.

5. Use the script `ch5-hierarchical.py` to try and estimate the standard deviation of the noise in the data. In fact we invert for a parameter λ which is the ratio of the estimated noise (i.e. σ in eqn. 10) to the true noise. Plot a histogram of the results and see how well the Bayesian sampling

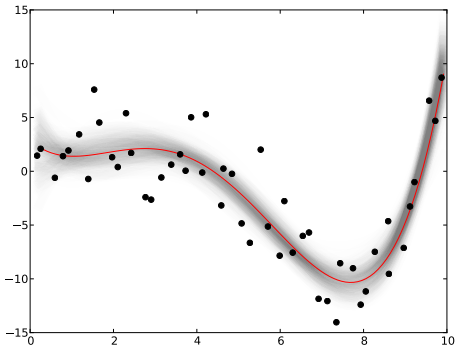

Figure 9: Grayscale image of probability density of all curves in the ensemble when assuming maximum polynomial order equal to 5.

is able to constrain the level of noise in the data. Your results should be similiar to Figure 10. If the data were estimated with $\sigma = 3$, what do you think the true value was ?

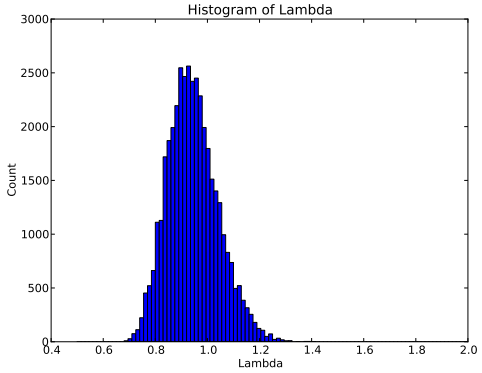

Figure 10: Posterior PDF of the data noise parameter λ values found from sampling.

Problem 6: Bayesian inference: Regression with discontinuities

Now consider a new data set of noisy (x, y) values only this time with discontinuities evident (See figure 11). The task is again to recover information about the (red) function from the observations, but this time the data must also be used to constrain the number and position of discontinuities.

This exercise is an expanded version of the previous one only now we allow for multiple polynomial functions separated into partitions along the x axis. We do not know where the discontinuities occur, nor how many there are. Algorithms for constructing solutions using the Partition modelling approach are in the python software library *rjmcmc*. To carry out the exercise you need to follow the *rjmcmc library tutorial guide* (file **rjmcmc_tutorial_multi.pdf**).

The same Likelihood function and prior is assumed as in problem 4, only now we introduce the number of partitions as unknowns and this also has a flat prior.

1. Follow instructions in the *rjmcmc library tutorial guide* to load the new data set of (x_i^{obs}, y_i^{obs}) values and plot the data (similar to Figure 11). (This is done for you by running the script `ch1-loading.py`.)

2. In this exercise we assume a polynomial representation for the unknown function (red curve) with maximum order 1 and a uniform prior PDF, and also sample over the number of partitions and the locations of the discontinuities.

   Follow instructions in the *rjmcmc library tutorial guide* to sample from the Bayesian posterior PDF using an McMC algorithm. Use the 1-D Partition modelling software to generate 50000 curves and take the mean. (This is done for you by running the script `ch2-analyse.py`.) Plot the a) the mean curve, b) the posterior PDF of the discontinuity locations and c) the posterior PDF of the number of partitions. Your figures should be similar to Figure 12.

   Looking at the results of the Bayesian sampling try and answer the following: How many partitions have been detected ? Where are the most likely location of the partition boundaries/discontinuities ? Can you see a difference in how well the data is able to detect boundaries of each partition ? What would you estimate as the likely position and error of the discontinuities ?

3. In the previous example linear polynomials were used in each partition. We now increase the maximum order of the polynomial to 5, meaning that up to quintic polynomials are used within

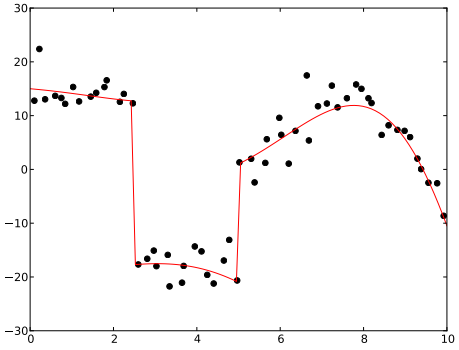

Figure 11: 2-D data set with discontinuities. Red curve is the real function, dots are observed data.

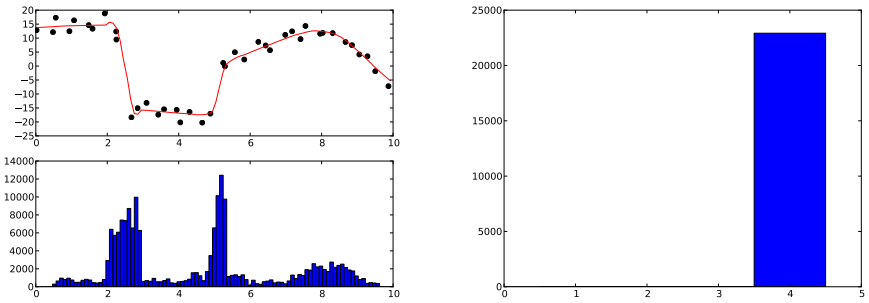

Figure 12: a) (Top left) 2-D data set with mean reconstructed model from 50000 McMC samples; b) (bottom left) the posterior PDF of the location of discontinuities in the data (notice the two peaks); c) the posterior PDF of the number of partitions detected in the data.

each partition. Script `ch3-orderanalysis.py` does this for you are reruns the Markov chain. Plot the same figures as in the previous exercise showing a) the mean curve, b) the posterior PDF of the discontinuity locations and c) the posterior PDF of the number of partitions. See how they have changed at the inference process now picks out the discontinuity number and location much better. Your figures should be similar to Figure 13. Using these probabilistic sampling results we might draw different conclusions than in Q2, but remember here the maximum polynomial order provided as prior information to the Bayesian procedure is different from in Q2.

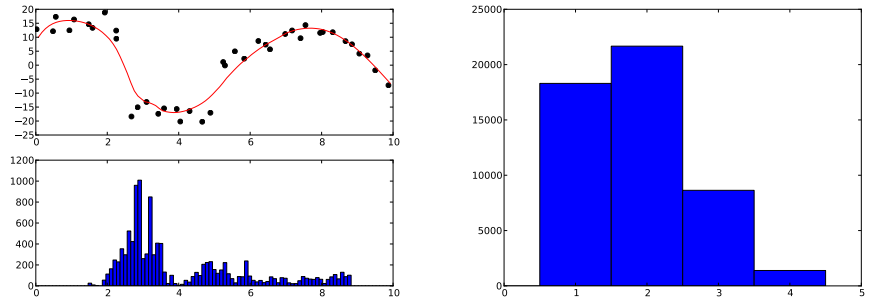

Figure 13: a) (Top left) 2-D data set with mean reconstructed model from 50000 McMC samples, with maximum order of polynomial set to 5; b) (bottom left) the posterior PDF of the location of discontinuities in the data (notice the two peaks); c) the posterior PDF of the number of partitions detected in the data.

4. Repeat the run, only this time use the library routines to plot a density model of the entire ensemble of curves and 95% confidence intervals at each point along the axes. In this way we get a visual impression of the error in the predicted curve. You can use script Use script `ch4-confidence.py` to do this and you should get a plot similar to Figure 14. This gives an indication of the confidence in the predictive capability of the ensemble of curves.

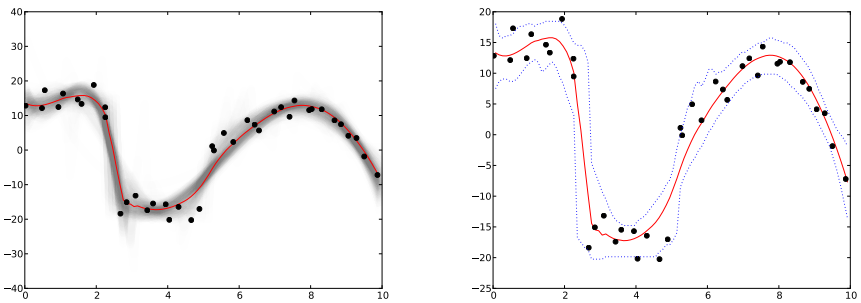

Figure 14: a) (left) Grayscale image of probability density of all curves in the ensemble when maximum polynomial order equal to 5 inside each partition; b) (right) 95% (point by point) confidence intervals from the ensemble of solutions.