David Aceituno 2015-05-20

Fast Model fitting with Evolutionary Monte Carlo algorithm

1 INTRODUCTION

This report describes how the technique of Evolutionary Monte Carlo (EMC) can be used to efficiently sample large parameter spaces to find "a good enough combination". Such problems often arise in optimization problems and in model fitting of experimental data to target distributions. After describing the EMC approach, two examples of applications of this technique are presented; first, for model fitting of experimental data from drop size measurements obtained in a testing facility simulating conditions in nuclear reactors, and second, for finding PID parameters for a control system balancing the inverted pendulum (Figure 1, without the glass). The EMC approach as used here is described in the article by Liang and Wong (2000)[i].

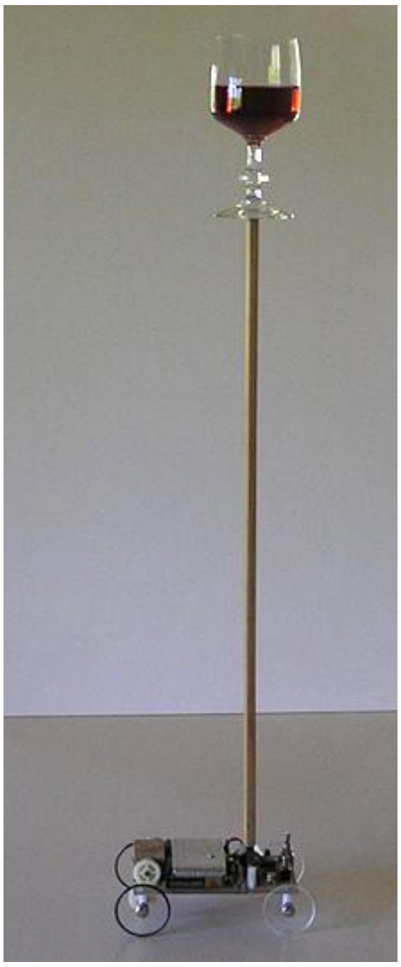


Figure 1: An application of EMC. (Picture from Wikipedia).

2 THE EMC ALGORITHM

We shall begin with a qualitative description of EMC.

Let $\bar{x}$ be a vector containing the genome of one individual, in this case coded in binary bits. Usually, $\bar{x}$ is called a *solution* or *chromosome*. A collection of N such individuals, each with their own $\bar{x}$, is called a *population*. The algorithm starts with all individuals in the population having their bits chosen randomly.

Let $H(\bar{x})$ denote the *fitness* of one chromosome. In the language of optimization, $H(\bar{x})$ is the cost function that we wish to minimize. In the language of "regular" Markov chain Monte Carlo algorithms (MCMC), we recognize $H(\bar{x})$ as the energy; in EMC it takes the place of the energy when calculating the acceptance probabilities with Boltzmann distribution. Computing the fitness from a given set of parameters is the most delicate step for successfully carrying out an EMC simulation. The details of fitness design are discussed later.

Similarly to parallel tempering simulations, each chromosome has a different temperature assigned to it. The temperature is taken from a *temperature ladder*. Well performing chromosomes will move towards low temperatures during the simulation, while bad performers will move up to high temperatures. At high temperatures the mutation probability will be higher, which in turn will introduce new DNA, meaning that these individuals will explore the parameter space faster. Well performing individuals at low temperatures will be less exposed to mutation, but prone to spreading their traits to lesser individuals. The distribution of the temperature ladder greatly affects the convergence of the simulation. Choosing a logarithmically spaced ladder has stood the test of time; it is desirable to have smaller spacing towards the lower end of the temperature ladder, since the exchange probability (movement on the ladder) is high when the temperature difference is low.

To evolve the population we apply operators on the current population. The three main operators used are the *mutation*, *crossover* and *exchange* operators. These are described in great detail in the article by Liang and Wong, and a short description will have to suffice here.

2.1 Genetics

In this report, each gene consists of 24 bits. At the start of a simulation, each chromosome is initialized with random bits. Each chromosome $\bar{x}_i$ is assigned a temperature T_i from the temperature ladder described earlier. With 24 bits we can make integers of size up to $2^{24} - 1 = 1.6 \cdot 10^{14}$. The set of integers $[0, 2^{24} - 1]$ can be mapped down to some parameter range of interest [a, b] where $b - a \ll 2^{24} - 1$. This allows the simulation to explore the parameter space with great resolution, usually many times greater than is allowed by sequential grid search methods.

Below we see an example of a solution with 2 genes of 24 bits each. Each coloured chunk is a gene, which is the binary representation an integer.

$$\bar{x} = 001011010101100100010101\ 010111010010001010110 10$$

Thus to obtain a real number parameter, call it μ, one must first convert from binary to decimal, i.e. base 2 to base 10, and then use the following formula:

$$\mu = \mu_{dec} \cdot \left(\frac{\mu_{\text{MAX}} - \mu_{\text{MIN}}}{2^{24} - 1} + \mu_{\text{MIN}} \right)$$

In this formula, μ_{dec} is the decimal representation of the gene, and μ_{MAX} and μ_{MIN} are the largest and smallest feasible numbers that μ could be, respectively.

2.2 The EMC operators

All operators are applied to the population N times at each generation.

The ***mutation*** operator is meant to introduce new genetics into the population, or in other words, allow the population to explore the parameter space. An individual chosen for mutation will suffer a k *point mutation* wherein k random bits are flipped in its DNA. The number k is also chosen at random. The mutated chromosome $\bar{y}$ is put to test and obtains a new fitness score $H(\bar{y})$. The Metropolis-Hastings algorithm is used to decide if the mutations are accepted or rejected. The individuals are chosen randomly for mutation N times independently in each generation.

Old: $\bar{x}_i = 0010110101\mathbf{0}1100100010101\ 01011\mathbf{1}010010001010110\mathbf{0}10$
New: $\bar{y}_i = 0010110101\mathbf{1}1100100010101\ 01011\mathbf{0}010010001010111\mathbf{1}10$

$$\text{Accept if} \qquad U(0,1) < \exp\left(-\frac{H(\bar{y}_i) - H(\bar{x}_i)}{T_i}\right) \cdot \frac{T(\boldsymbol{x}|\boldsymbol{y})}{T(\boldsymbol{y}|\boldsymbol{x})}$$

Here $U(0,1)$ is a real number chosen randomly from a uniform distribution in the range [0, 1]. The $T(\cdot\,|\,\cdot)$'s in the quotient are transition probabilities determined by the selection strategy used to pick $\bar{x}_i$ from the population. In this case they cancel since $\bar{x}_i$ is chosen randomly from the population with uniform distribution. More details on these transitions probabilities can be found in the references.

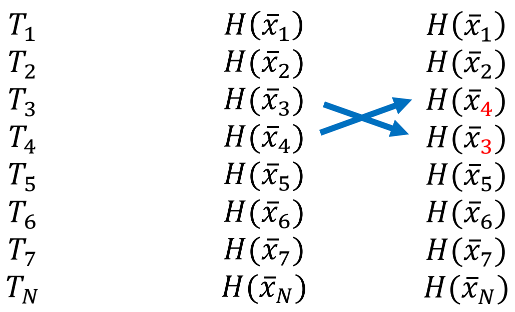
The ***crossover*** operator mimics the process of mating. A good chromosome is chosen with probability proportional to its fitness, to mate with *any other* chromosome. These two parent chromosomes create a pair of offspring by splitting their DNA at some randomly chosen point so that each offspring can copy one segment from each parent. The point where the genome is cut is called the *crossover point*. The new offspring is tested for fitness and is accepted or rejected in a similar fashion as with the mutation operator. If they are accepted they replace the parents place on the temperature ladder, and if they are rejected the parents remain in place. The crossover operator is performed around $N/5$ times on each generation, such that around 40 % of the population gets to mate. The example below illustrates how the crossover splices the parent DNA.

$$\text{Parents}\ \left\{\begin{array}{l|l} \bar{x}_i = 001011010101100 & 100010101\ 01011101001000101011010 \\ \bar{x}_j = 110001111010110 & 000010110\ 10001010000101111101010 \end{array}\right.$$

$$\text{Children}\ \left\{\begin{array}{l|l} \bar{y}_i = 001011010101100 & 000010110\ 10001010000101111101010 \\ \bar{y}_j = 110001111010110 & 100010101\ 01011101001000101011010 \end{array}\right.$$

$$\text{Accept if} \qquad U(0,1) < \exp\left(-\frac{\Delta H_i}{T_i} - \frac{\Delta H_j}{T_j}\right) \cdot \frac{T(\boldsymbol{x}|\boldsymbol{y})}{T(\boldsymbol{y}|\boldsymbol{x})}$$

The ***exchange*** operator attempts to move well performing chromosomes down the temperature ladder in order to retain their good qualities and at the same time forcing poor performers up on the ladder. High up on the temperature ladder mutations are accepted more easily, so the bad traits are easily eliminated. The exchange operator is also performed N times on each generation.


T_1 $H(\bar{x}_1)$ $H(\bar{x}_1)$
T_2 $H(\bar{x}_2)$ $H(\bar{x}_2)$
T_3 $H(\bar{x}_3)$ $H(\bar{x}_4)$
T_4 $H(\bar{x}_4)$ $H(\bar{x}_3)$
T_5 $H(\bar{x}_5)$ $H(\bar{x}_5)$
T_6 $H(\bar{x}_6)$ $H(\bar{x}_6)$
T_7 $H(\bar{x}_7)$ $H(\bar{x}_7)$
T_N $H(\bar{x}_N)$ $H(\bar{x}_N)$

$$\text{Accept the swap if} \qquad U(0,1) < \exp\left[\left(H(\bar{x}_i) - H(\bar{x}_j)\right)\left(\frac{1}{T_i} - \frac{1}{T_j}\right)\right] \cdot \frac{T(\boldsymbol{x}|\boldsymbol{x}')}{T(\boldsymbol{x}'|\boldsymbol{x})}$$

Thus the EMC algorithm for one generation reads as follows:

1. Apply the *mutation* or the *crossover* operator to the current population with probabilities q_m and $1 - q_m$, respectively (q_m is called the mutation rate).
2. Try to *exchange* N pairs of neighbouring chromosomes $\bar{x}_i$ and $\bar{x}_j$.

In addition to these operators, there are a host of variations to improve convergence. One such is the method of "elitism", wherein a list is kept of the best performers throughout the simulation and pick these genes for mutation or crossover, with some low probability. Another operator, described in the article, is the adaptive crossover operator that copies bits from good parents and reverses them from bad parents. Both of these have been used in the examples that follow.

2.3 FITNESS

Designing a fitness function takes patience and practice. One often marvels at the algorithms apparent creativity as it finds new ways of cheating, i.e., getting a good score by performing badly. One has to clearly define the desired behaviour by some scoring procedure that punishes bad behaviour and rewards the good.

As a general rule of thumb, the fitness $H(\bar{x})$ should be a function that decreases very rapidly when individuals perform well. A usable measure is often a sum of squared distances in LSQ-fitting or some integral that measures the total error of an individual's performance. Ideally, that integral should vanish for perfect behaviour. In those circumstances it has been very fruitful to use the following expression

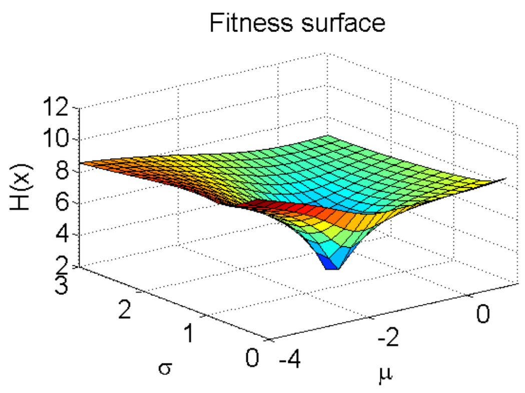
$$H(\bar{x}) = -\frac{1}{\log(M(\bar{x})+\epsilon)} + \log(M(\bar{x}) + \epsilon),$$

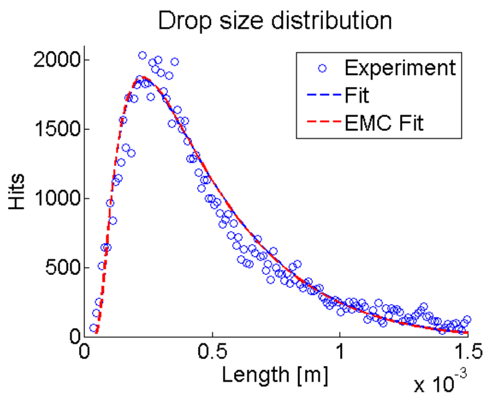
where M is said measure, and $\epsilon \in (0,1]$ is used to prevent overflow if Mbecomes very small. This way of defining $H(\bar{x})$ has proven very efficient in several applications.

3 MODEL FITTING

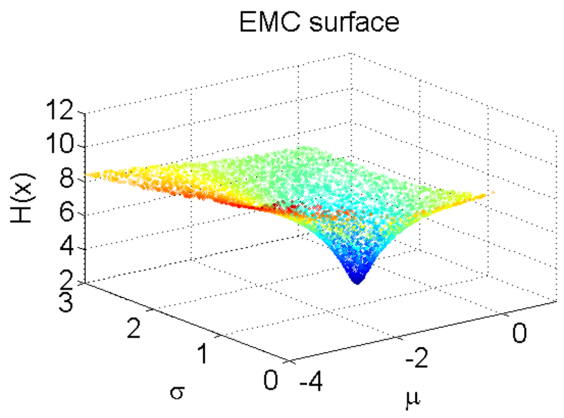
To showcase the strengths of the EMC algorithm we can use it to find parameters of some distribution that best fit experimental data.

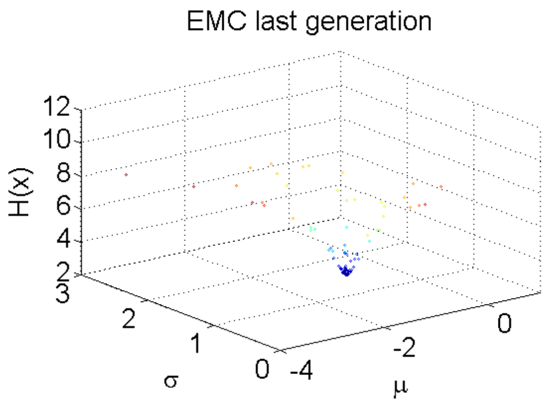
At a testing facility that simulates the conditions in the core of a nuclear reactor, the drop sizes in the cooling loop are measured directly above the core. For reasons we will not go into, it is of interest to identify the drop size distribution with some model, in this case the *Upper limit log-normal* distribution (ULLN). This distribution is characterized by two parameters(μ, σ) which one needs to find. This problem is easy enough to solve with least squares optimization or similar methods, but can take considerable time with more parameters, for instance if more than one ULLN-curve are superimposed, i.e. if there is more than one peak on the same experimental data. In the figures below, we can see the fitness as a function of the parameters (μ, σ). The EMC algorithm finds the minima on this surface in about 0.3 seconds on a laptop pc. On the distribution plot we can see the fit as the red dashed curve. The blue dashed curve, found through much slower sequential search is barely visible directly underneath.





In the figure to the left, below, we can see the points on the surface that the population has visited during the simulation. To the right, we see the last generation of the population. They have clearly found something interesting at the bottom of the pit. Notice also how some individuals remain high up on the surface, still exploring the surface. If any of them were to find another, possibly better minima, they would drag the rest of the population along with them.





The population size on this run was 100 individuals, and temperature scale from 0.1 to 10 for about 150 generations.

4 THE INVERTED PENDULUM

The choice of PID parameters for a control system can be quite tricky with many degrees of freedom. The Monte Carlo approach is well suited for the higher dimensional problems and this is no exception.

In the work leading to this report, the first step was to correctly simulate a pendulum attached to a cart. The pendulum has mass m and length l from the base to its centre of mass. The cart has mass M and applies the force F to the base of the pendulum.

We start by defining the Lagrangian $L = T - V$ and then applying the Euler-Lagrange equations of motion. For the terms in the Lagrangian we have:

$$T = \frac{Mv_1^2}{2} + \frac{mv_2^2}{2} + \frac{I\dot{\theta}^2}{2}$$
$$V = -mgl\cos\theta$$

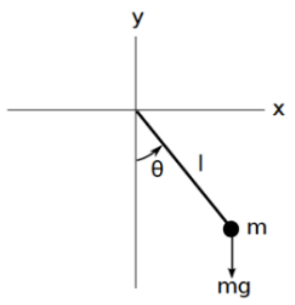


Figure 2: The pendulum base moves freely in the x-direction. The target angle is π.

To continue we need to determine the velocities:

$$v_1^2 = \dot{x}^2$$
$$v_2^2 = \left(\frac{d}{dt}(x + l\sin\theta)\right)^2 + \left(\frac{d}{dt}(l\cos\theta)\right)^2 = \dot{x}^2 + 2\dot{\theta}\dot{x}l\cos\theta + \dot{\theta}^2l^2$$

Putting all of this together gives us the Lagrangian

$$L = \frac{1}{2}M\dot{x}^2 + \frac{1}{2}m\left(\dot{x}^2 + 2\dot{\theta}\dot{x}l\cos\theta + \dot{\theta}^2l^2\right) + \frac{1}{2}I\dot{\theta}^2 + mgl\cos\theta$$

Now we get the equations of motion from the Euler-Lagrange equations:

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{x}} - \frac{\partial L}{\partial x} = F - b\dot{x}$$

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}} - \frac{\partial L}{\partial \theta} = 0$$

Here b is a friction coefficient. Solving for $\ddot{x}$ and $\ddot{\theta}$ finally gives us:

$$\ddot{x} = \frac{F - b\dot{x} - ml(\ddot{\theta}\cos\theta - \dot{\theta}^2\sin\theta)}{M + m}$$
$$\ddot{\theta} = -\frac{\ddot{x}\cos\theta + gl\sin\theta}{l + \frac{I}{ml}}$$

Integrator

The second equation of motion is highly nonlinear and the 4th order Runge-Kutta integrator was used to integrate it with step length $dt = 0.002$. The simpler Verlet algorithm would grow unbounded and crash the program regularly. The first equation of motion was integrated in the same way for convenience.

PID controller parameters

The PID regulator is used to minimize the deviation from a desired reference value $r(t)$ of some system $y(t)$ we wish to control. The deviation, or error, is defined as $e(t) = r(t) - y(t)$. The PID controller takes the error as input and outputs a control signal $u(t)$ that is fed back into the system in order to control it. For a PID control system, we have:

$$u(t) = K_p e(t) + K_i\int_0^t e(t)dt + K_d\frac{d}{dt}e(t)$$

Here we have 3 parameters K_p, K_i and K_d that amplify (or diminish) the role of each term. The first term serves to react to present conditions. The second term makes small remaining errors disappear, displaying the qualities of a memory. It also increases instability at high values. The third term sees ahead and helps with overshooting.

In this case of the inverted pendulum a PID controller was used to determine the force applied from the cart to the rod:

$$F(t) = K_{p\theta}e_\theta(t) + K_{i\theta}\int_0^t e_\theta(t)dt + K_{d\theta}\frac{d}{dt}e_\theta(t)$$
$$+K_{px}e_x(t) + K_{ix}\int_0^t e_x(t)dt + K_{dx}\frac{d}{dt}e_x(t)$$

Notice here that we have 3 new terms to control the horizontal position of the car. These three terms are active *iff* the angle error and angular velocity are small enough.

The 6 K's are the parameters that we seek to optimize in the simulation.

Fitness

Measuring the fitness of rod balancers is not trivial. The main objective is to severely punish ill-behaved chromosomes by giving them very unfavourable fitness. This way the offending chromosomes are sent up on the temperature ladder to learn some manners by mutation.

In the application at hand, the chromosomes came up with some ingenious ways of performing poorly; some would oscillate in place (much like waving a finger *no, no, no!*), others would balance the rod perfectly except traveling in the x-direction in a hurry (I call them *voyagers*). Some even tried to topple over as quickly as possible such as to make the integrals measuring their performance vanish.

In the end it was most fruitful to measure integral of the errors, as well as the total travelled distances, that is:

$$M_1 = \int_0^t |e_\theta(t)|\,dt$$
$$M_2 = \int_0^t |e_x(t)|\,dt$$
$$M_3 = \frac{1}{t}\int_0^t \sqrt{1+\dot{e}_\theta^2}\,dt$$
$$M_4 = \frac{1}{t}\int_0^t \sqrt{1+\dot{e}_x^2}\,dt$$

$$M = M_1 + M_2 + M_3 + M_4$$

$$H(\bar{x}) = -\frac{1}{\log(M+1)} + \log(M+1),$$

Here e_θ is the deviation of the angle from some reference value. In this case $r_\theta = \pi$. Similarly, $e_x = r_x - x(t)$ where $r_x = 0$ and x obviously refers to the sideways displacement of the cart, and not a chromosome. Naturally, perfect chromosomes would have no error, and therefore the integrals would vanish.

Measuring M_3 and M_4 accomplishes to punish both the *voyager* and *oscillator* types.

Results

The simulation with 150 individuals for about 200 generations takes 30 seconds on a laptop PC. Of that time, 0.3-0.5 seconds are used for the EMC algorithm while the rest is spent computing the equations of motion. It is difficult to present the results further on this format, so I invite you to see my YouTube video, where you can see the rod balancers in action (hint: it worked!).

https://youtu.be/oYsTTtz3ixw

5 Discussion

This approach to sampling parameter spaces is very promising. The computation time increases exponentially for the sequential trial of all parameter combinations, but the EMC algorithm handles it very efficiently. Compared with steepest descent optimization algorithms it has the added benefit of guaranteeing the convergence to a global minimum independently of starting conditions because of the ergodic properties of MCMC simulations. Judging by the fact that the convergence time is about the same with 2, 6 or even 8 dimensional parameter spaces, a coarse estimation is that the EMC algorithm scales much better than linearly with number of parameters. In the end, the main difficulty lies in defining an appropriate fitness function, which is typically the case with genetic algorithms. On the other hand, for the case of least squares fitting the fitness formula from section 2.3 works quite generally for any number of parameters.

6 References

Liang, Faming, and Wing Hung Wong. "Evolutionary Monte Carlo: Applications to Cp model sampling and change point problem." *Statistica sinica* 10.2 (2000): 317-342.

Goswami, Gopi, and Jun S. Liu. "On learning strategies for evolutionary Monte Carlo." *Statistics and Computing* 17.1 (2007): 23-38.

Ren, Yuan, Yu Ding, and Faming Liang. "Adaptive evolutionary Monte Carlo algorithm for optimization with applications to sensor placement problems."*Statistics and Computing* 18.4 (2008): 375-390.