COMS 4115 PROGRAMMING LANGUAGE & TRANSLATOR

Monte Carlo Simulation Language

Proposal

Yunling Wang (yw2291@columbia.edu)
Chong Zhai (cz2191@columbia.edu)
Diego Garcia (dg2275@columbia.edu)
Eita Shuto (es2808@columbia.edu)

Introduction

A computer simulation is an attempt to model a real-life or hypothetical situation on a computer so that it can be studied to see how the system works. By changing variables, predictions may be made about the behavior of the system. It has become a useful part of modeling many natural systems in physics, chemistry and biology, human systems in economics and social science (the computational sociology) as well as in engineering to gain insight into the operation of those systems. Traditionally, the formal modeling of systems has been via a mathematical model, which attempts to find analytical solutions enabling the prediction of the behavior of the system from a set of parameters and initial conditions. Computer simulation is often used as an adjunct to, or substitution for, modeling systems for which simple closed form analytic solutions are not possible. There are many different types of computer simulation; the common feature they all share is the attempt to generate a sample of representative scenarios for a model in which a complete enumeration of all possible states would be prohibitive or impossible. There have been millions of simulations using different popular programming languages. And over 30 simulation programming languages [1] have been designed and implemented. However, from another point of view, most of them are discipline-oriented other than method-oriented which will also explain the considerable number of languages introduced.

When it is infeasible or impossible to compute an exact result with a deterministic algorithm, Monte Carlo methods which rely on repeated computation and random or pseudo-random numbers become most suitable for such calculations. This class of methods has been broadly used in physical, chemical, mathematical and financial systems.

Generality

Ripley (1987, p.119) wrote: "The object of any simulation study is the estimation of one or more expectations of the form E[f (x)]". The generation of the U[0, 1] numbers is the focus of this page and is very important for the quality of the simulation. We allow user to generate pseudo-random sequence of numbers with different schemes or quasi-random numbers.

Monte Carlo simulations vary with problems but share similar processes. First, the generation of (pseudo or quasi) random numbers is a way to generate representative samples (a lot of scenarios), to describe the uncertainties of our physical problem through probabilities distributions. And the Uniform [0, 1] distribution permits to generate all the distributions that we need to perform the simulations. Second, the simulation problem can be viewed as the solution of one (or several) integral, the Monte Carlo Integral. In general, the Monte Carlo method solves multidimensional integrals, and the expression for the Monte Carlo approximation for the multidimensional integral over the unit hypercube is given by this language.

Example

This sample code is from the calculation of Energy of Helium atom which in principle has no analytical solution. Scientists have been using simulations methods to calculate this energy for over 50 years and improvement of precision are still going on. For a simplified variantional Monte Carlo method, which only has one variant, the form of the function is complicated enough, not to mention calculations with more than 10 variants.

Hamiltonian

$$H_0 = -\tfrac{1}{2}\nabla_1^2 - \tfrac{1}{2}\nabla_2^2 - \frac{1}{r_{1L}} - \frac{1}{r_{1R}} - \frac{1}{r_{2L}} - \frac{1}{r_{2R}} + \frac{1}{r_{12}}$$

The wave function is given by:

$$\Psi(r_1, r_2, s) = \varphi(r_1, s) \cdot \varphi(r_2, s) \cdot f(r_{12})$$

With:

$$\varphi(r_1, s) = e^{-\frac{r_{1L}}{a}} + e^{-\frac{r_{1R}}{a}}$$

$$\varphi(r_1, s) = e^{-\frac{r_{2L}}{a}} + e^{-\frac{r_{2R}}{a}}$$

$$f(r_{12}) = e^{\frac{r_{12}}{2(1+\beta r_{12})}}$$

The expectation function to be evaluated is:

$$H\Psi = E\Psi =$$

$$\left[\left(\frac{\mathbf{x}_1 \cdot \mathbf{x}_1 - \mathbf{x}_1 \cdot \mathbf{x}_2 + \frac{1}{2}sx_{12}}{2r_{1L}r_{12}(1+\beta r_{12})^2} + \frac{1}{r_{1L}}\right)e^{-\frac{r_{1L}}{a}} + \left(\frac{\mathbf{x}_1 \cdot \mathbf{x}_1 - \mathbf{x}_1 \cdot \mathbf{x}_2 - \frac{1}{2}sx_{12}}{2r_{1R}r_{12}(1+\beta r_{12})^2} + \frac{1}{r_{1R}}\right)e^{-\frac{r_{1R}}{a}}\right]\frac{\varphi_2 f_{12}}{a} +$$

$$\left[\left(\frac{\mathbf{x}_2 \cdot \mathbf{x}_2 - \mathbf{x}_1 \cdot \mathbf{x}_2 - \frac{1}{2}sx_{12}}{2r_{2L}r_{12}(1+\beta r_{12})^2} + \frac{1}{r_{2L}}\right)e^{-\frac{r_{2L}}{a}} + \left(\frac{\mathbf{x}_2 \cdot \mathbf{x}_2 - \mathbf{x}_1 \cdot \mathbf{x}_2 + \frac{1}{2}sx_{12}}{2r_{2R}r_{12}(1+\beta r_{12})^2} + \frac{1}{r_{2R}}\right)e^{-\frac{r_{2R}}{a}}\right]\frac{\varphi_1 f_{12}}{a} +$$

$$\left(-\frac{1}{a^2} - \frac{1}{r_{12}(1+\beta r_{12})^3} - \frac{1}{4(1+\beta r_{12})^4} - \frac{1}{r_{1L}} - \frac{1}{r_{1R}} - \frac{1}{r_{2L}} - \frac{1}{r_{2R}} + \frac{1}{r_{12}}\right)\varphi_1\varphi_2 f_{12}$$

Instead, we could define a Hamiltonian, or a just a function and automatically calculate the operator acting on wave function. To define built-in types like Hamiltonian operator and wave function (which is basic all needed to get the variational function might be interesting but makes little sense nor applicability. Instead, vector operators like gradient, divergence, curl and Laplacian which generate complexity in the formula will be introduced.

Sample Code

```
/*
   Built-in constant, such as base of the natural logarithm e = 2.71828182845904523536
   Pi = 3.14159265358979323846, based on the precision requirement, we can call these constant
   with a precision digit parameter. For example: e(6) = 2.71828 and Pi(10) = 3.141592653.
   This is some time a must, as some time user doesn't have to type 3.00000 to make the
   compiler understand 6 digits is the required precision. And it's sometimes better than
   just have double or float precision
 */

// Initialization
int steps, walk, cut;
double beta = 0.5; // this is the coefficient to be variated

/*
```

```
  3 is the dimension of the random sample
  walk is the array size, or number of random walks
  we have not decide a set of rand generator scenarios, sometimes choosing between “fine”
  scenario and efficiency is a question, sometimes we want to use quasi-way. Whether to
  include low-discrepancy sequence type other than type ‘RAND’ has not be decided.
 */
rand first(3, walk, scenario);
rand second(3, walk, scenario);

/* <o> means operator by some rules, detailed calculation involved in  this case is
   laplacian
 */
Etmp = hami() <o> wave();

// if the iteration step by default is one, this could be simplified by i=1:step
do i=1:1:step
         do j=1:1:walk
                  If random_walk(fun1)/fun1 < 1 //infinitesimal random walk;
                  // accept or reject this walk via condition
                           walk(first)
                  /* we don’t care about scenario for single random number and by default,
                    1,2 argument takes 1 and 1 correspondently
                   */

                  elif random_walk(fun1)/fun1 > rand a()
                           walk(first);

// Same for second electron
                  If random_walk(fun2)/fun2 < 1
                           walk(second);
                  elif random_walk(fun2)/fun2 > rand a()
                           walk((second)

         if(i>cut)
                  Esum += Etmp(first, second);
/* in which normalized is a built in function and it automatically calculated the number
   of acceptances
*/
Eva = Esum/((step-cut)*walk) or Eva = normalize(Esum)

print Eave, beta

// function to define the Hamiltonian
hami()
         null // still thinking whether to support some symbolic calculations

// Wave function
wave()
         null // still thinking whether to support some symbolic calculations
```