2011-04-18 作業解答

$$\begin{array}{c|cc} 0 & & \\ c & c & \\ \hline & a & b \end{array}$$

1. 把 Runge-Kutta 二次的中點法改成取任意間點 c （Butcher 表如右）： 試求 a 與 b 係數的值。

For the differential equation $dy/dt = f(t, y)$, we consider the Taylor expansions:

$$y = y_1 t + y_2 t^2 + O(t^3) \text{ and } f(t, y) = f_{00} + f_{10} t + f_{01} y + f_{11} t\, y + O(t^2) + O(y^2).$$

Substitute the expansions in to the differential equation, we get

$$y_1 = f_{00} \text{ and } y_2 = (f_{10} + f_{01} f_{00})/2.$$

Instead of the mid-point, we evaluate the intermediate step for f_1 at $t = c\tau$, which expands to

$$f_1 = f(c\tau,\ f_{00} c\tau) = f_{00} + c(f_{10} + f_{01} f_{00})\tau + O(\tau^2).$$

The estimate of y is, therefore,

$$y(\tau) = (a\, f_0 + b\, f_1)\tau = (a+b)\, f_{00}\tau + b\ c(f_{10} + f_{01} f_{00})\tau^2 + O(\tau^3).$$

Matching the coefficient of the Taylor expansion for y, we get

$$a = 1 - 1/(2c) \text{ and } b = 1/(2c).$$

2. 寫一程式模擬前頁的隨機漫遊，統計在 Random Seed = 1 到 1000 中的前頁問題的數值解。

[前頁問題]
給定 p, q, L 及初始 x_0， 求
存活率 r
平均到家時間 τ_h
平均墜崖時間 τ_f

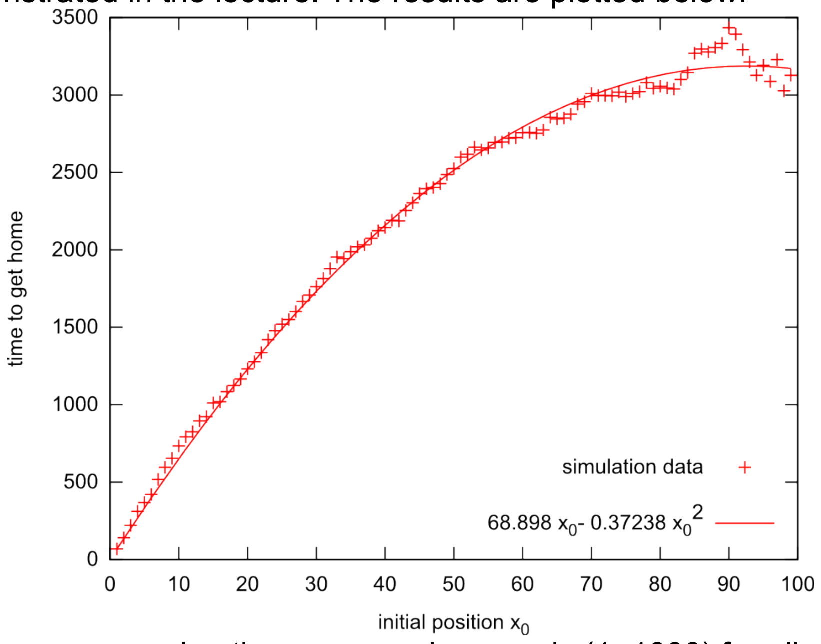
Different random seeds to the (pseudo-)random number generator result in different random sequences, which can be used to model different realizations of experiments and allow us to do an ensemble average. We start the walker at the assigned initial position and the perform the walks until it either arrives home or falls off the cliff. Note that there is no upper bound on the time this algorithm will run. However, the chance that it will keep running decays exponentially in time. This kind of algorithm can be categorized as Las Vegas algorithm in that the resource required is not bounded but it is also a Monte Carlo algorithm in that the resulted answer is only statistically correct.
C++:

```
#include <iostream>
#include "ran_nr.hh"
using namespace std;
int main()
{
	int x0;
	int L;
	double p;
	double q;
	size_t n_sample = 1000;
	cout << "x0 L p q: ";
	cin >> x0 >> L >> p >> q;
	cout << "got " << x0 << ' ' << L << ' ' << p << ' ' << q << '\n';
```

```
      size_t c_home = 0;
      size_t c_fall = 0;
      double t_home = 0;
      double t_fall = 0;
      for (size_t i = 0; i < n_sample; i ++) {
            RanNR rng(i);
            double nstep = 0;
            int x = x0;
            while (x > 0 && x < L) {
                  if (rng.uniform() < p / (p + q)) x --;
                  else x ++;
                  nstep ++;
            }
            if (x == 0) {
                  c_home ++;
                  t_home += nstep;
            }
            else {
                  c_fall ++;
                  t_fall += nstep;
            }
      }
      cout << "s_rate=" << double(c_home) / n_sample << '\n';
      cout << "t_home=" << t_home / c_home << '\n';
      cout << "t_fall=" << t_fall / c_fall << '\n';
      return 0;
}
```

3. 用 2.的程式計算 $L = 100, p = q = 0.5$ 時 $x_0 = 1 \sim 99$ 的平均到家時間 τ_h（作圖），假定 $\tau_h = a + b\, x_0 + c\, x_0^2$, $a = 0$,以最小方差法求 b 及 c 的最佳值。

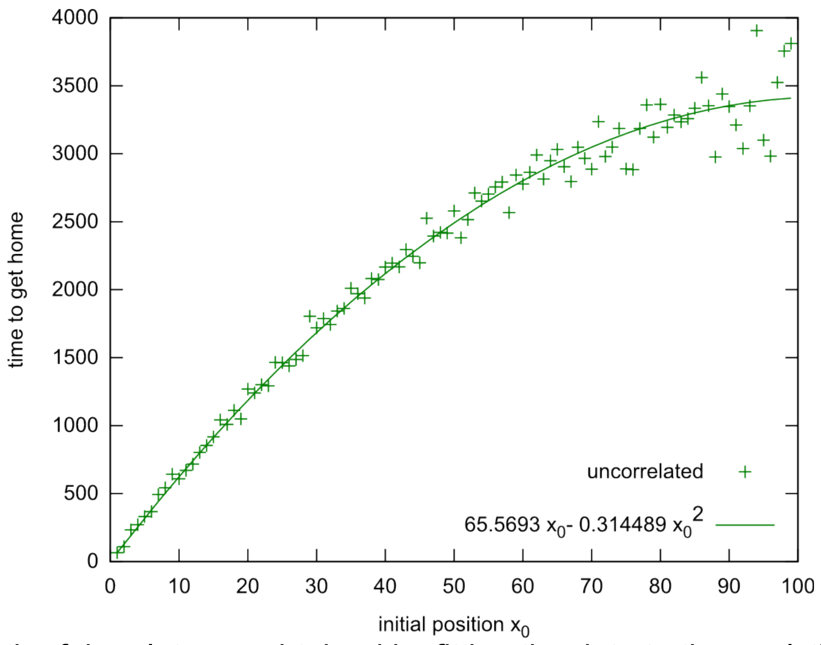
With the executable hw7_rw from problem 2, we can use a BASH script to obtain the answer as demonstrated in the lecture. The results are plotted below:



We note that since we are using the same random seeds (1~1000) for all x_0, the data points are not independent with each other. The adjacent points tend to be close to each other. To generate independent data points we need to use different sets of random seeds for different data points. Or, we can use different parts of a single random sequence. The code for generating the entire data set using one random seed is as follows. The random number generator is initialized once and the sequence is used continuously to generate all

data in the set.
C++:
```cpp
#include <iostream>
#include "ran_nr.hh"
using namespace std;
int main()
{
      RanNR rng(123);
      int L = 100;
      double p = 0.5;
      size_t n_sample = 1000;
      for (int x0 = 1; x0 < L; x0 ++) {
            size_t c_home = 0, c_fall = 0;
            double t_home = 0, t_fall = 0;
            for (size_t i = 0; i < n_sample; i ++) {
                  double nstep = 0;
                  int x = x0;
                  while (x > 0 && x < L) {
                        if (rng.uniform() < p) x --;
                        else x ++;
                        nstep ++;
                  }
                  if (x == 0) {
                        c_home ++;
                        t_home += nstep;
                  }
                  else {
                        c_fall ++;
                        t_fall += nstep;
                  }
            }
            cout << x0;
            cout << '\t' << double(c_home) / n_sample;
            cout << '\t' << t_home / c_home;
            cout << '\t' << t_fall / c_fall;
            cout << '\n';
      }
      return 0;
}
```

In general application, it is usually preferable to have independent data points in the data set. Since, the effectiveness of each measurement is reduced when the data are correlated and the statistical error will no longer go down as fast with the increase of number of measurements. As seen in the following plot, the uncorrelated data points are much more scattered:



The curves in both of the plots are obtained by fitting the data to the analytic form:

$$\tau_h = b\, x_0 + c\, x_0^2$$

with the coefficients b and c obtained by minimizing the mean-square error

$$\varepsilon(b, c) \equiv \sum_i [\tau_i - \tau_h(x_i)]^2.$$

The minimizations are carried out alternating in b and c until they both approach their stationary values. The data are read from "cin" and stored in dynamic "vector" arrays. The "find_min" function(al) uses the simple minimization algorithm described in lecture. We alternatingly pass the "find_min" functional the to-be-minimized functions, "fix_c" and "fix_b", which are specializations of the "error2" function. For each iteration of the process, we keep track of the total square change of b and c and stop when this change is acceptably small.

C++:

```
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;
vector<double> x;
vector<double> t;
void read_data(istream & st)
{
      do {
            double x0, s_rate, t_home, t_fall;
            st >> x0 >> s_rate >> t_home >> t_fall;
            if (! st.good()) break;
            x.push_back(x0);
            t.push_back(t_home);
      } while (true);
}
double error2(double b, double c)
{
      double err2 = 0;
      for (size_t i = x.size(); i --;) {
            double d = x[i] * (b + c * x[i]) - t[i];
            err2 += d * d;
      }
      return err2;
}
double fixed_val;
double fix_c(double b) {return error2(b, fixed_val);}
double fix_b(double c) {return error2(fixed_val, c);}
double find_min(double a, double b, double (*func)(double))
{
      double fa = (*func)(a), fb = (*func)(b);
      double c = (a + b) / 2, fc = (*func)(c);
      double lsz;
      do {
            lsz = fabs(a - b);
            double d = (b + c) / 2;
            double fd = (*func)(d);
            if (fd < fc) {
                  a = c; fa = fc;
                  c = d; fc = fd;
                  continue;
            }
            double e = (c + a) / 2;
            double fe = (*func)(e);
            if (fe < fc) {
                  b = c; fb = fc;
                  c = e; fc = fe;
                  continue;
            }
            b = d; fb = fd;
            a = e; fa = fe;
      } while (lsz > fabs(a - b));
      return c;
}
int main()
{
      read_data(cin);
      double d2 = 1;
      double b = 1, c = 1;
      while (d2 > 1e-12) {
            d2 = 0;
            fixed_val = b;
            double new_c = find_min(c + 10, c - 10, &fix_b);
            d2 += (c - new_c) * (c - new_c);
            c = new_c;
            fixed_val = c;
            double new_b = find_min(b + 10, b - 10, & fix_c);
            d2 += (b - new_b) * (b - new_b);
            b = new_b;
      }
      cout << "b=" << b << '\n';
      cout << "c=" << c << '\n';
      return 0;
}
```

* Note that the average time to get home can be solved analytically with the exact solution:

$$\tau_h = (2L\,x_0 - x_0^2)/3.$$