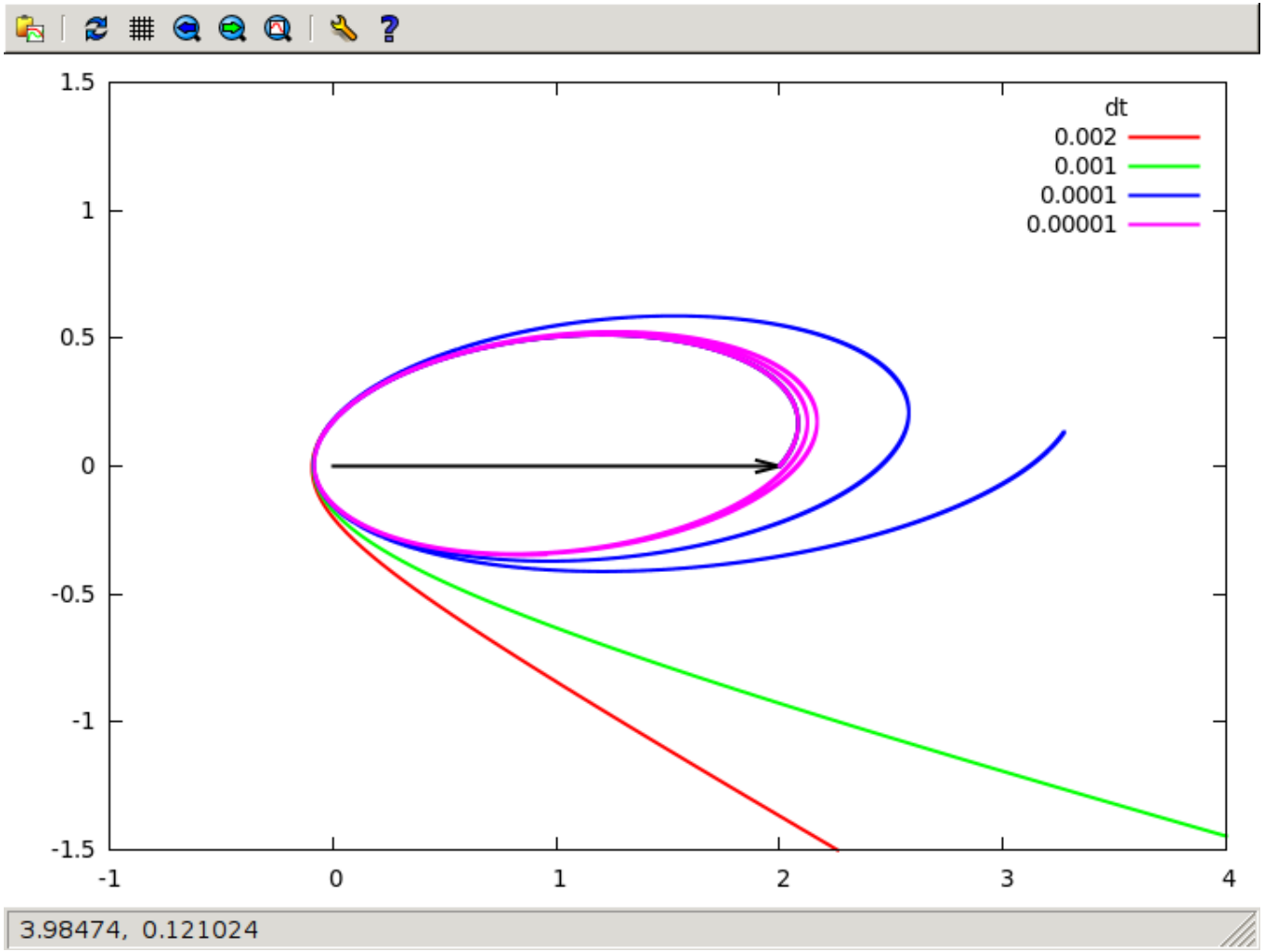
觀念補充

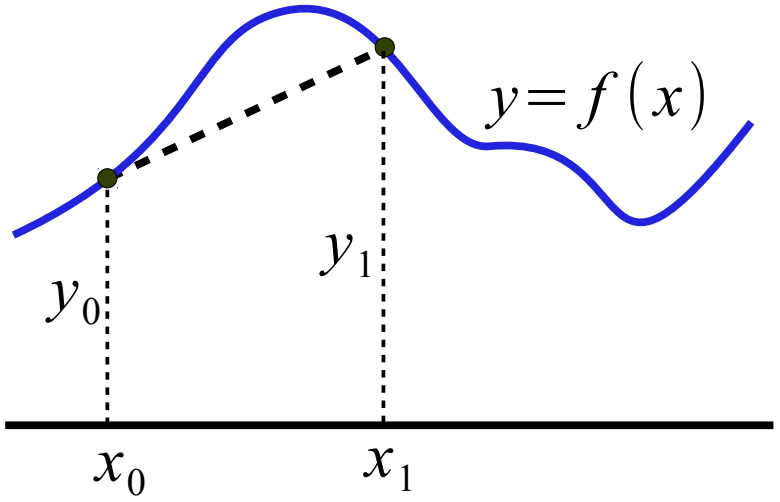
- 多維陣列的實作
- 複製建構子（copy constructor）
- 存取控制（public, private, protected, const）
- 標頭檔和程式庫
- 自由練習補充：在 matrix 程式庫中加入求反矩陣及本徵值的功能。

上週習題例圖更正



請用 hw5-orbit1 來產生範例中的軌跡
（hw5-orbit 產生的為 Leapfrog 法的結果）

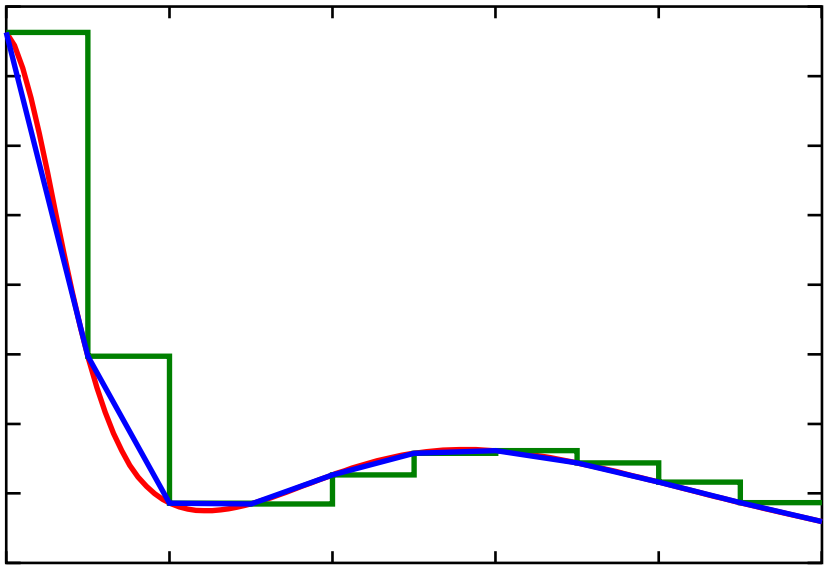
函數的近似



$$y \approx y_0 + (y - y_0)\frac{y_1 - y_0}{x_1 - x_0}$$

- 線性插值 (interpolation)
- 多項式近似： Lagrange 式、 Newton 式

數值積分



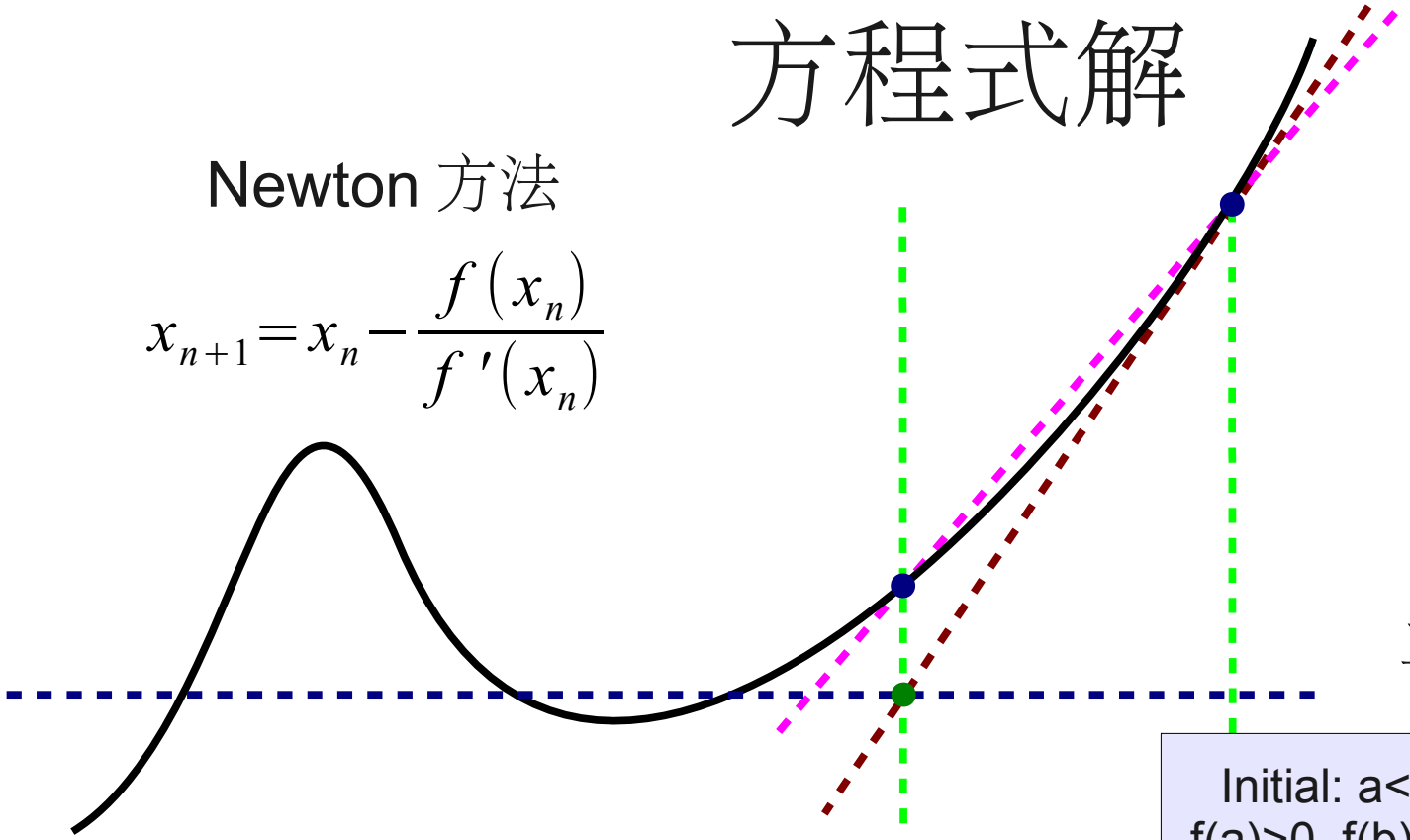
- 矩形法、梯形法
- Simpson 法

$$\frac{x_1 - x_0}{6}\left[f(x_0) + 4f\left(\frac{x_0 + x_1}{2}\right) + f(x_1)\right]$$

方程式解

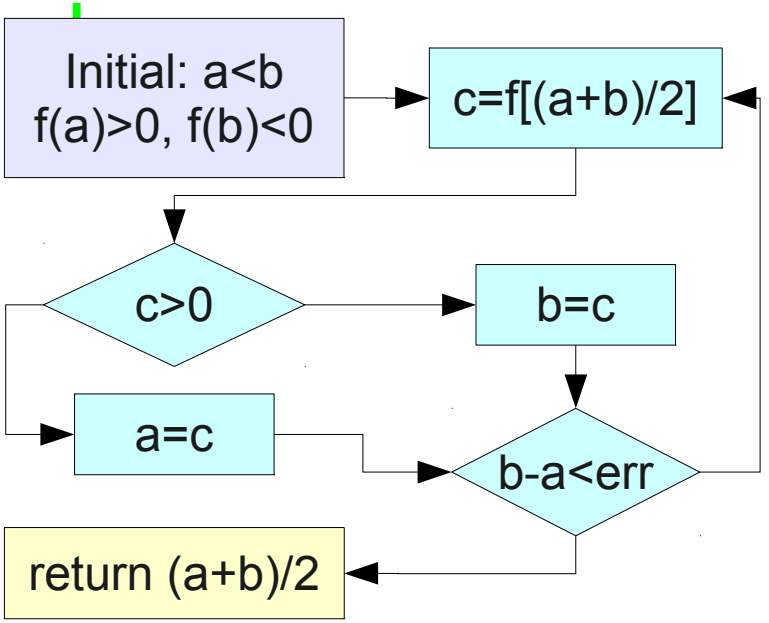
Newton 方法

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$



- 割線法
  （不需微分值）
- 假位法
  （有收斂的保證）

二分逼近法



常微分方程的積分

$$\frac{dy}{dt} = f(y,t)$$

$$y_n, t_n \to y_{n+1}, t_{n+1}$$

$$t_{n+1} - t_n = \tau$$

運動方程

$$\frac{dv}{dt} = a(x) \quad \frac{dx}{dt} = v$$

Euler 方法（最簡單）

$$y_{n+1} = y_n + \tau f(y_n, t_n)$$

Leapfrog 方法

$$x_{n+1} = x_n + \tau v_{n+1/2}$$

$$v_{n+1/2} = v_{n-1/2} + \tau a(x_n)$$

Runge-Kutta 法（RK4）

$$y_{n+1} = y_n + \frac{1}{6}\tau(f_1 + 2f_2 + 2f_3 + f_4)$$

$$f_1 = f(y_n, t_n)$$

$$f_2 = f\left(y_n + \frac{1}{2}\tau f_{1,} t_n + \frac{1}{2}\tau\right)$$

$$f_3 = f\left(y_n + \frac{1}{2}\tau f_{2,} t_n + \frac{1}{2}\tau\right)$$

$$f_4 = f(y_n + \tau f_{3,} t_n + \tau)$$

Verlet 方法（具時間對稱）

$$x_{n+1} = 2x_n - x_{n-1} + \tau^2 a(x_n)$$

具速的 Verlet 法

$$x_{n+1} = 2x_n + v_n + \frac{1}{2}\tau^2 a(x_n)$$

$$v_{n+1} = v_n + \tau\frac{a(x_n) + a(x_{n+1})}{2}$$

亂數（隨機數）產生器

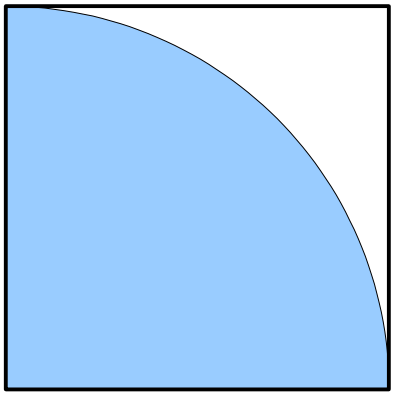
```
// From Numerical Recipe: Ran = [A1_l(C3) + A3_r] ^ B1
class RanNR {
	typedef unsigned long long u64;
	u64 C3, A3, B1;
	static const u64 SeedShift = 4101842887655102017ULL;
	static const u64 B_a = 4294957665ULL;
	static const u64 C_a = 2862933555777941757ULL;
	static const u64 C_c = 7046029254386353087ULL;
public:
	RanNR(u64 seed) { // generate the same initial states as NR
		C3 = (seed ^ SeedShift) * C_a + C_c; A3 = C3 ^ (C3 >> 17); A3 ^= A3 << 31;
		A3 ^= A3 >> 8; C3 = C_a * C3 + C_c; B1 = B_a * (A3 & 0xffffffff) + (A3 >> 32);
		A3 ^= A3 >> 17; A3 ^= A3 << 31; A3 ^= A3 >> 8; C3 = C_a * C3 + C_c;}
	inline u64 int64() {
		C3 = C_a * C3 + C_c; A3 ^= A3 >> 17; A3 ^= A3 << 31; A3 ^= A3 >> 8;
		B1 = B_a * (B1 & 0xffffffff) + (B1 >> 32); u64 a1 = C3 ^ (C3 << 21);
		a1 ^= a1 >> 35; a1 ^= a1 << 4; a1 = (a1 + A3) ^ B1; return a1;}
	inline double uniform() {return 5.42101086242752217E-20 * int64();}
};
```

亂數產生器的使用

```
#include <nr_ran.hh>
#include <iostream>
using namespace std;
int main()
{
    NRRan rng(123);
    size_t in_cir = 0;
    size_t n_pts;
    cout << "number of points: ";
    cin >> n_pts;
    for (size_t i = 0; i < n_pts; i ++) {
        double x = rng.uniform();
        double y = rng.uniform();
        if (x * x + y * y < 1) in_cir ++;
    }
    cout << "pi ≈ " << in_cir * 4.0 / n_pts << '\n';
}
```

初始化 seed

產生亂數



Monte Carlo 法求積分
產生隨機點來
估計面積比例

本週習題

1. 用 leapfrog 法計算上週習題的行星運動軌道：計算給定初始條的軌道週期（可用 y 值回到 0 為完成一週期的條件）；計算並比較在不同 τ 時（$0.01, 0.001, 0.0001$）時 leapfrog 法和上週用的 Euler 法收斂速度的差異。

2. 數值積分：以梯形法及 Simpson 法求下列函數在 0~10 區間的積分

$$f(x)=\sin\left[x^2\exp(-x)+\ln(x+8)\right]$$

計算兩方法在 $\Delta x = 1, 0.1, 0.01, 0.001$ 時的積分值，並以此作圖估計兩方法在 $\Delta x \to 0$ 的收斂速度。

3. 用 Monte Carlo 法求上述的積分，估計在點數 $n \to \infty$ 時的收斂速度。（在給定點數時可用不同的 seed 來作取樣，以估計它的標準差）