Schémas numériques: erreur forte - erreur faible

Exemples en C++

Vincent Lemaire
vincent.lemaire@upmc.fr

Fonctions à nombre variable d'arguments

Comme la fonction `printf` en `C`, une fonction peut prendre un nombre variable d'arguments. Il suffit que le premier argument contienne suffisamment d'information (nombre des arguments suivants ; ou nombre et type des arguments suivants).

Pour cela il existe

- le type `va_list`
- les macros `va_start`, `va_end`, et `va_arg`
- un prototype de fonction de la forme `T f(T1 arg1,...)`.

Ces fonctions sont définies dans la librairie standard du `C` et reprises dans le `C++`.

Diffusions paramétriques

But : Fournir une classe abstraite facile à utiliser pour définir des EDS paramétriques.

```
template <typename T, typename S>
struct sde {
    sde(int n, ...) : p(n) {
        va_list vl;
        va_start(vl, n);
        for (int i = 0; i < n; i++)
            p[i] = va_arg(vl, double);
        va_end(vl);
    };
    virtual T drift(T) = 0;
    virtual S sigma(T) = 0;
    std::vector<double> p;
};
```

En C++11 on peut remplacer ce vieux mécanisme utilisant le préprocesseur (`va_list`, `va_start`, `va_arg`, `va_end`) par des variadic templates.

Classe abstraite scheme_diff_unidim

Par exemple pour le modèle de Black-Scholes en dimension 1, on définit

```
struct BS : public sde<double, double> {
    BS(double r, double sigma) : sde<double, double>(2, r, sigma) {};
    double drift(double x) { return p[0]*x; };
    double sigma(double x) { return p[1]*x; };
};
```

On va maintenant définir une classe abstraite scheme_diff_unidim pour faciliter l'implémentation de schémas de la forme $X_{n+1} = f(X_n, G_{n+1})$ où G_{n+1} est une v.a. Gaussienne. qui contient

- une méthode virtuelle algo qui fait tout le travail *i.e.* qui code f...
- des opérateurs += et ++ qui appellent la fonction algo et mettent à jour l'état du schéma,
- une fonction init() qui reinitialise le schéma : $n = 0$ et $X_n = x_0$,
- un opérateur () qui execute algo de 0 à N,
- des fonctions d'accès à l'état interne du schéma : iter, current, last_alea, pas, nb_iter, eds.

Classe abstraite scheme_diff_unidim

```
struct scheme_diff_unidim {
    typedef double result_type;
    scheme_diff_unidim(double x0, sde<double, double> &X, double h, int N)
      :x0(x0),X(X),h(h),s_h(sqrt(h)),N(N),n(0),state(x0),acc_brownian(0,s_h) {};
    virtual double algo(double acc_brown) = 0;
    scheme_diff_unidim& operator+=(double acc_brown) {
        algo(last_acc = acc_brown);
        ++n;
        return *this;
    };
    scheme_diff_unidim& operator++() { return operator+=(acc_brownian()); };
    scheme_diff_unidim& operator++(int) { /* postfix: a faire ! ... */ };
    void init() { n = 0; state = x0; };
    bool not_end() const { return (n < N); };
    double operator()() { /* au tableau */ };
    // fonctions d'acces...
    protected:
        int n, N;
        double h, s_h, x0, state, last_acc;
        sde<double, double> &X;
        gaussian acc_brownian;
};
```

Rq : que pensez-vous du champ X ?

Schéma d'Euler

On rappelle le schéma d'Euler pour une diffusion :

$$X_{t_{n+1}} = X_{t_n} + b(X_{t_n})h + \sigma(X_{t_n})\sqrt{h}G_{n+1}, \quad X_0 = x_0.$$

Le code sera simplement :

```
struct euler : public scheme_diff_unidim {
    euler(double x0, sde<double, double> &X, double h, int N)
        : scheme_diff_unidim(x0, X, h, N) {};
    double algo(double acc_brown) {
        return state += X.drift(state)*h + X.sigma(state)*acc_brown;
    };
};
```

Rappel : Toutes les méthodes/opérateurs de la classe `scheme_diff_unidim` seront utilisables par un objet de la classe `euler`.

Schéma de Milstein

On rappelle le schéma de Milstein pour une diffusion :

$$X_{t_{n+1}} = X_{t_n} + b(X_{t_n})h + \sigma(X_{t_n})\sqrt{h}G_{n+1} + \frac{1}{2}(\sigma\sigma')(X_{t_n})(G_{n+1}^2 - 1)h.$$

On peut aussi utiliser la version équivalente de Newton (sans σ')

$$X_{t_{n+1}} = X_{t_n} + b(X_{t_n})h - \sigma(X_{t_n})\sqrt{h} + \tilde{\sigma}_{n+1}\sqrt{h}(G_{n+1} - 1),$$

$$\text{avec } \tilde{\sigma}_{n+1} = \sigma\left(X_{t_n} + \frac{1}{2}\sigma(X_{t_n})\sqrt{h}(G_{n+1} - 1)\right).$$

Le code s'écrit :

```
struct milstein : public scheme_diff_unidim {
    milstein(double x0, sde<double, double> &X, double h, int N)
        : scheme_diff_unidim(x0, X, h, N) {};
    double algo(double acc_brown) {
        double z = acc_brown - s_h;
        double y = X.sigma(state + 0.5*X.sigma(state)*z);
        return state += X.drift(state)*h - X.sigma(state)*s_h + y*z;
    };
};
```

Erreur forte d'un schéma

Soit $\left(x_t\right)_{t \geqslant 0}$ le processus de diffusions solution sur $[0, T]$ de

$$\mathrm{d}x_t = b(x_t)\mathrm{d}t + \sigma(x_t)\mathrm{d}W_t, \quad x_0 \in \mathbf{R}^d,$$

avec b et σ Lipschitz, et $\left(X_t^h\right)_{t \geqslant 0}$ un schéma d'approximation de pas $h = \frac{T}{N}$ partant de $X_0^N = x_0$.

On dit que le schéma est d'erreur forte $\mathbf{L}^p$ d'ordre α si

$$\left\| \sup_{t \in [0,T]} \left| x_t - X_t^h \right| \right\|_p = \mathscr{O}(h^\alpha)$$

ou encore

$$\left\| \sup_{k \in \{0,\dots,N\}} \left| x_{t_k} - X_{t_k}^h \right| \right\|_p = \mathscr{O}(h^\alpha)$$

Erreur forte d'un schéma

Soit $(x_t)_{t \geqslant 0}$ le processus de diffusions solution sur $[0, T]$ de

$$\mathrm{d}x_t = b(x_t)\mathrm{d}t + \sigma(x_t)\mathrm{d}W_t, \quad x_0 \in \mathbf{R}^d,$$

avec b et σ Lipschitz, et $(X_t^h)_{t \geqslant 0}$ un schéma d'approximation de pas $h = \frac{T}{N}$ partant de $X_0^N = x_0$.

On dit que le schéma est d'erreur forte $\mathbf{L}^p$ d'ordre α si

$$\left\| \sup_{t \in [0,T]} \left| x_t - X_t^h \right| \right\|_p = \mathscr{O}(h^\alpha)$$

ou encore

$$\left\| \sup_{k \in \{0,\dots,N\}} \left| x_{t_k} - X_{t_k}^h \right| \right\|_p = \mathscr{O}(h^\alpha)$$

- Schéma d'Euler : erreur forte d'ordre $\frac{1}{2}$ dans tout $\mathbf{L}^p$ $(p > 0)$.
- Schéma de Milstein : erreur forte d'ordre 1 dans tout $\mathbf{L}^p$ $(p > 0)$ (dim 1...).

Exemple sur Black-Scholes (dim 1)

Rappel : Pour Black-Scholes **on connaît** la vraie solution

$$\forall t \in [0,T], \quad x_t = x_0 e^{\left(r - \frac{\sigma^2}{2}\right)t + \sigma W_t}.$$

Le schéma d'Euler de pas $h = \frac{T}{N}$ s'écrit

$$X^h_{t_{k+1}} = X^h_{t_k} + rX^h_{t_k} h + \sigma X^h_{t_k} \sqrt{h} G_{k+1}.$$

Le schéma de Milstein de pas $h = \frac{T}{N}$ s'écrit

$$X^h_{t_{k+1}} = X^h_{t_k} + (r - \frac{1}{2}\sigma^2) X^h_{t_k} h + \sigma X^h_{t_k} \sqrt{h} G_{k+1} + \frac{1}{2}\sigma^2 X^h_{t_k} G^2_{k+1}.$$

Exemple sur Black-Scholes (dim 1)

Rappel : Pour Black-Scholes **on connaît** la vraie solution

$$\forall t \in [0,T], \quad x_t = x_0 e^{\left(r-\frac{\sigma^2}{2}\right)t+\sigma W_t}.$$

Le schéma d'Euler de pas $h = \frac{T}{N}$ s'écrit

$$X^h_{t_{k+1}} = X^h_{t_k} + rX^h_{t_k} h + \sigma X^h_{t_k} \sqrt{h} G_{k+1}.$$

Le schéma de Milstein de pas $h = \frac{T}{N}$ s'écrit

$$X^h_{t_{k+1}} = X^h_{t_k} + (r - \frac{1}{2}\sigma^2) X^h_{t_k} h + \sigma X^h_{t_k} \sqrt{h} G_{k+1} + \frac{1}{2}\sigma^2 X^h_{t_k} G^2_{k+1}.$$

Attention : Il faut calculer $\sup_k \left| x_{t_k} - X^h_{t_k} \right|$ sur le même événement ω *i.e.* la même trajectoire Brownienne $\left(W_t\right)_{t \geqslant 0}$.
Pour faciliter le calcul de cette erreur, on définit un schéma `true_BS` qui calcule x_{t_k} aux mêmes instants t_k que le schéma à étudier.

Schéma exact true_BS

```
// ne s'applique qu'avec la sde BS !!
struct true_BS : public scheme_diff_unidim {
    true_BS(double x0, BS &X, double h, int N)
        : scheme_diff_unidim(x0, X, h, N),
          mu(X.p[0]-0.5*X.p[1]*X.p[1]) {};
    double algo(double acc_brown) {
        return state *= exp(mu*h + X.p[1]*acc_brown);
    };
    private:
        double mu;
};
```

Remarque : Première façon de restreindre un schéma à une sde, on verra une seconde façon de faire un peu plus loin...

« Payoff » de l'erreur forte $\mathbf{L}^2$

```
template <typename Scheme1, typename Scheme2>
struct erreur_forte {
    erreur_forte(Scheme1 S1, Scheme2 S2) : S1(S1), S2(S2) {};
    double operator()() {
        double max_err = 0, erreur = 0;
        for (S1.init(), S2.init();
             S1.not_end() && S2.not_end();
             ++S1, S2+=S1.last_alea()) {
            erreur = fabs(S1.current()-S2.current());
            max_err = (erreur > max_err) ? erreur : max_err;
        }
        return max_err*max_err;
    };
    protected:
        Scheme1 S1;
        Scheme2 S2;
};
```

Black-Scholes - Erreur forte du schéma d'Euler

En live...

Black-Scholes - Erreur forte du schéma de Milstein

En live...

Autre estimation

Comment estimer l'erreur forte d'un schéma sans connaître la vraie solution ?

- Si $\left\|\sup_k \left|x_{t_k} - X^h_{t_k}\right|\right\|_p \longrightarrow 0$ alors

$$\left\|\sup_k \left|x_{t_k} - X^h_{t_k}\right|\right\|_p = \mathcal{O}(h^\alpha) \quad \Longleftrightarrow \quad \left\|\sup_k \left|X^h_{t_k} - X^{h/2}_{t_{2k}}\right|\right\|_p = \mathcal{O}(h^\alpha)$$

Autre estimation

Comment estimer l'erreur forte d'un schéma sans connaître la vraie solution ?

- Si $\left\|\sup_k \left|x_{t_k} - X^h_{t_k}\right|\right\|_p \longrightarrow 0$ alors

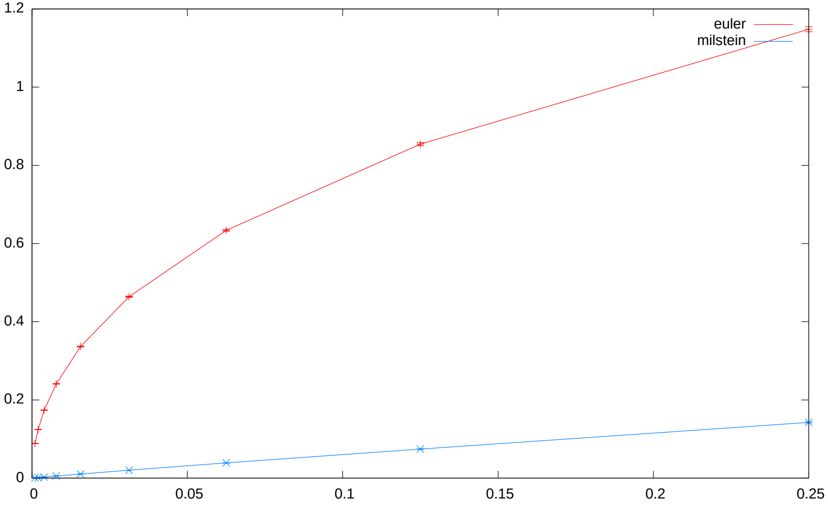
$$\left\|\sup_k \left|x_{t_k} - X^h_{t_k}\right|\right\|_p = \mathscr{O}(h^\alpha) \iff \left\|\sup_k \left|X^h_{t_k} - X^{h/2}_{t_{2k}}\right|\right\|_p = \mathscr{O}(h^\alpha)$$

Il faut donc construire les schémas $(X^h_{kh})_{k=0,\ldots,N}$ et $(X^{h/2}_{\frac{kh}{2}})_{k=0,\ldots,2N}$ en « utilisant » la même trajectoire brownienne...
On fera de même pour mettre en œuvre l'extrapolation de Richardson-Romberg qui permet de gagner un ordre pour l'erreur faible.

« Payoff » de l'erreur forte $\mathbf{L}^2$ approchée

```cpp
template <typename Scheme>
struct erreur_forte_app {
    erreur_forte_app(Scheme S1) : S1(S1),
        S2(S1.current(), S1.eds(), S1.pas()*0.5, S1.nb_iter()*2) {};
    double operator()() {
        double max_err = 0, erreur = 0, acc_B1, acc_B2;
        for (S1.init(), S2.init(); S2.not_end();) {
            acc_B1 = (++S2).last_alea();
            acc_B2 = (++S2).last_alea();
            S1+=(acc_B1 + acc_B2);
            erreur = fabs(S1.current()-S2.current());
            max_err = (erreur > max_err) ? erreur : max_err;
        }
        return max_err*max_err;
    };
protected:
        Scheme S1;
        Scheme S2;
};
```

Black-Scholes - Erreur forte approchée



Erreur faible d'un schéma

Soit $\left(x_t\right)_{t\geqslant 0}$ le processus de diffusions solution sur $[0,T]$ de

$$\mathrm{d}x_t = b(x_t)\mathrm{d}t + \sigma(x_t)\mathrm{d}W_t, \quad x_0 \in \mathbf{R}^d,$$

avec b et σ dans $\mathscr{C}^\infty$ à dérivées bornées, et $\left(X_t^h\right)_{t\geqslant 0}$ un schéma d'approximation de pas $h = \frac{T}{N}$ partant de $X_0^N = x_0$.

On dit que le schéma est d'erreur faible d'ordre α si

$$\forall f \text{ « régulière »}, \quad \mathbf{E}\left[f(x_t)\right] - \mathbf{E}\left[f(X_t^h)\right] = \mathscr{O}(h^\alpha)$$

Erreur faible d'un schéma

Soit $\left(x_t\right)_{t\geqslant 0}$ le processus de diffusions solution sur $[0,T]$ de

$$\mathrm{d}x_t = b(x_t)\mathrm{d}t + \sigma(x_t)\mathrm{d}W_t, \quad x_0 \in \mathbf{R}^d,$$

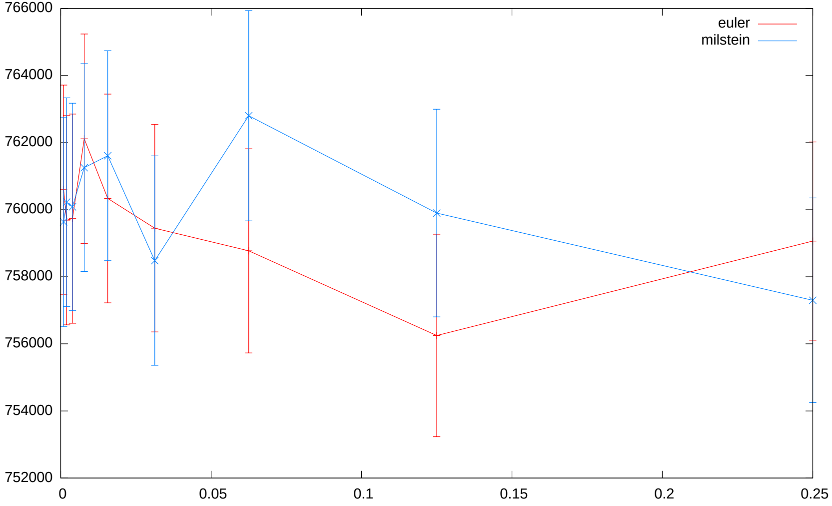
avec b et σ dans $\mathscr{C}^\infty$ à dérivées bornées, et $\left(X_t^h\right)_{t\geqslant 0}$ un schéma d'approximation de pas $h = \frac{T}{N}$ partant de $X_0^N = x_0$.

On dit que le schéma est d'erreur faible d'ordre α si

$$\forall f \text{ « régulière »}, \quad \mathbf{E}\left[f(x_t)\right] - \mathbf{E}\left[f(X_t^h)\right] = \mathscr{O}(h^\alpha)$$

- Schéma d'Euler : erreur faible d'ordre 1.
- Schéma de Milstein : erreur faible d'ordre 1.

Exemple sur Black-Scholes (avec une fonction f donnée)



Comment interpréter ce graphe ?

Rappels sur le CIR

Soit $(\alpha, \lambda, \sigma) \in (\mathbf{R}_+^*)^3$ tel que $\sigma^2 < 2\alpha$. Alors l'EDS

$$\mathrm{d}x_t = (\alpha - \lambda x_t)\mathrm{d}t + \sigma\sqrt{x_t}\mathrm{d}W_t, \quad x_0 > 0,$$

a une unique solution $\left(x_t\right)_{t \geqslant 0}$, et pour tout $t \geqslant 0$, $x_t > 0$.

- Simulation exacte possible mais coûteuse...
  cf. slides sur simulation exacte.

Rappels sur le CIR

Soit $(\alpha, \lambda, \sigma) \in (\mathbf{R}_+^*)^3$ tel que $\sigma^2 < 2\alpha$. Alors l'EDS

$$\mathrm{d}x_t = (\alpha - \lambda x_t)\mathrm{d}t + \sigma\sqrt{x_t}\mathrm{d}W_t, \quad x_0 > 0,$$

a une unique solution $\left(x_t\right)_{t \geqslant 0}$, et pour tout $t \geqslant 0$, $x_t > 0$.

- Simulation exacte possible mais coûteuse...
  cf. slides sur simulation exacte.

Classe CIR dérivée de la classe `sde<double, double>` :

```
struct CIR : public sde<double, double> {
    CIR(double alpha, double lambda, double sigma)
        : sde<double, double>(3, alpha, lambda, sigma) {};
    double drift(double x) { return p[0] - p[1]*x; };
    double sigma(double x) { return p[2]*sqrt(x); };
};
```

Schéma « en valeur absolue »

Schéma proposé par Diop

$$X^h_{t_{k+1}} = \left| X^h_{t_k} + (\alpha - \lambda X^h_{t_k})h + \sigma \sqrt{X^h_{t_k}} \sqrt{h} G_{k+1} \right|, \quad X^h_0 = x0.$$

- Erreur forte d'ordre $\frac{1}{2}$
- Erreur faible d'ordre 1

Schéma « en valeur absolue »

Schéma proposé par Diop

$$X^h_{t_{k+1}} = \left| X^h_{t_k} + (\alpha - \lambda X^h_{t_k})h + \sigma\sqrt{X^h_{t_k}}\sqrt{h}G_{k+1} \right|, \quad X^h_0 = x0.$$

- Erreur forte d'ordre $\frac{1}{2}$
- Erreur faible d'ordre 1

On peut aussi introduire (pour les tests numériques) un schéma de Milstein « en valeur absolue »

$$X^h_{t_{k+1}} = \left| X^h_{t_k} + (\alpha - \lambda X^h_{t_k})h + \sigma\sqrt{X^h_{t_k}}\sqrt{h}G_{k+1} + \frac{\sigma^2}{4}(G^2_{k+1} - 1)h \right|, \quad X^h_0 = x0.$$

Erreur forte, erreur faible ? ?

Schéma implicite

Cf. sujet d'examen mars 2008, d'après un papier d'A. Alfonsi.

Par Itô, on a $y_t = \sqrt{x_t}$ qui vérifie

$$\mathrm{d}y_t = \left(\frac{\alpha - \sigma^2/4}{2y_t} - \frac{\lambda}{2} y_t \right) \mathrm{d}t + \frac{\sigma}{2} \mathrm{d}W_t, \quad y_0 = \sqrt{v_0}.$$

On considère le schéma implicite (coefficient de diffusion constant donc pas de problème de définition...)

$$Y_{t_{k+1}} = Y_{t_k} + \left(\frac{\alpha - \sigma^2/4}{2Y_{t_{k+1}}} - \frac{\lambda}{2} Y_{t_{k+1}} \right) h + \frac{\sigma}{2} \sqrt{h} G_{k+1}, \quad Y_0 = \sqrt{v_0},$$

i.e. pour tout $k \in \{0, \ldots, N-1\}$, $Y_{t_{k+1}}$ solution du polynôme

$$\left(1 + \frac{\lambda}{2} h\right) y^2 - \left(Y_{t_k} + \frac{\sigma}{2} \sqrt{h} G_{k+1}\right) y - \frac{\alpha - \sigma^2/4}{2} h = 0.$$

Schéma implicite -2-

Le code du schéma implicite Y_{t_k} peut s'écrire de la façon suivante :

```
struct implicit_CIR : public scheme_diff_unidim {
    implicit_CIR(double x0, sde<double, double> &X, double h, int N)
      : scheme_diff_unidim(x0, X, h, N)  {
        try { dynamic_cast<CIR&>(X); }
        catch (const std::bad_cast& e) {
        std::cerr << "Schéma correct que pour le CIR !" << std::endl;
        }
    };
    double algo(double acc_brown) {
        double a = (1 + 0.5*X.p[1]*h);
        double b = - (0.5*X.p[2]*acc_brown + sqrt(state));
        double c = - 0.5*(X.p[0] - 0.25*X.p[2]*X.p[2])*h;
        double y = (-b+sqrt(b*b-4*a*c)) / (2*a);
        return state = y*y;
    };
};
```

- Deuxième façon de restreindre un schéma numérique à un type de `sde`. Quel est l'avantage de cette méthode ? (Tester la première méthode *i.e.* avec le constructeur `implicit_CIR(double x0, CIR &X,...)` avec le fonctor `erreur_forte_app`. Quelle est l'erreur ? Pourquoi ?)
- A partir de ce schéma on peut aussi considérer une approximation au premier ordre pour obtenir un schéma explicite (testé dans les simulations qui suivent).

Fonction **dynamic_cast**

La fonction **dynamic_cast** permet de « downcaster » : de convertir l'adresse d'une classe mère en l'adresse d'une classe dérivée.

- si la conversion est impossible :
  - renvoie un pointeur NULL si le **dynamic_cast** se fait sur les pointeurs
  - renvoie une exception : un objet std::**bad_cast** si le **dynamic_cast** se fait sur les références
- mécanisme lent à n'utiliser que si nécessaire

Exemple (version pointeur) :

```
gaussian *G = dynamic_cast<gaussian*>(Y);
```

Pour gérer les erreurs :

```
expo *E = dynamic_cast<expo*>(Y);
if (E == 0) cout << "Erreur: pas une expo" << endl;
```

Fonction **dynamic_cast**

La fonction **dynamic_cast** permet de « downcaster » : de convertir l'adresse d'une classe mère en l'adresse d'une classe dérivée.

- si la conversion est impossible :
  - renvoie un pointeur NULL si le **dynamic_cast** se fait sur les pointeurs
  - renvoie une exception : un objet `std::bad_cast` si le **dynamic_cast** se fait sur les références
- mécanisme lent à n'utiliser que si nécessaire

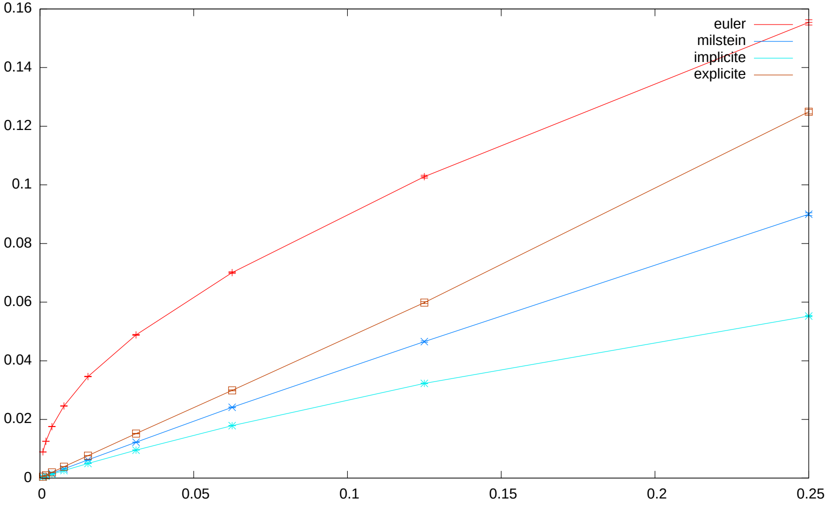
Exemple (version référence) :

```
gaussian &G = dynamic_cast<gaussian&>(X);
```

Pour gérer les erreurs :

```
try {
    expo &E = dynamic_cast<expo&>(X);
}
catch (const std::bad_cast& e) {
    cout << "Erreur: par une expo" << endl;
}
```

Erreur forte : comparaisons des schémas



Erreur faible : comparaisons des schémas