

# Introduction aux Équations Différentielles Partielles

*Programmation Orientée Objet - C++*

Réalisé par:  
Elsa Catteau  
Charlotte Prouzet

# I. Equation de diffusion et convection

*Formulation générale :*

$$f(x, t) = \frac{\partial u}{\partial t} - D \frac{\partial^2 u}{\partial x^2} + C \frac{\partial u}{\partial x},$$

avec :

- D : le coefficient de diffusion
- C : le coefficient de convection
- f (x,t) : le terme source
- u (x,t) : la solution recherchée

## II. Approximation numérique

*Méthode des différences finies :*

$$u_i^{n+1} = u_i^n + \gamma (u_{i+1}^n - 2u_i^n + u_{i-1}^n) - \nu (u_i^n - u_{i-1}^n) + f(x_i, t^n) \Delta t,$$

$$\gamma = \frac{D \Delta t}{\Delta x^2}, \quad \nu = \frac{C \Delta t}{\Delta x}.$$

avec :  $\gamma$  correspondant aux contributions diffusives

$\nu$  correspondant aux contributions convectives

## Implémentation en C++ : *transportDiffusion.h*

```
#ifndef TRANSPORTDIFFUSION_H
#define TRANSPORTDIFFUSION_H

#include <vector>
#include <string>

class TransportDiffusion {
private:
    double C, D, L;
    int Nx, Nt;
    double dx, dt, T;
    double gamma, v;
    std::vector<double> x;
    std::vector<double> t;
    std::vector<std::vector<double>> u_calcule;

public:
    // Constructeur et destructeur
    TransportDiffusion(double C_, double D_, double L_, int Nx_, int Nt_);
    ~TransportDiffusion();

    // Methodes
    double u_exacte(double xi, double ti);
    double f(double xi, double ti);
    void calculer();
    double erreur_L2();
    double erreur_Linf();
    void exporter_csv(const std::string& filename);
    void exportData(const std::string& filename);
};

#endif
```

# Implémentation en C++ : *transportDiffusion.cpp*

## A) Les directives

```
#include "TransportDiffusion.h"
#include <cmath>
#include <iostream>
#include <fstream>
#include <algorithm>
```

## B) Constructeur

```
TransportDiffusion::TransportDiffusion(double C_, double D_, double L_, int Nx_,
    int Nt_)
: C(C_), D(D_), L(L_), Nx(Nx_), Nt(Nt_)
```

1

Initialisation du  
constructeur

Création d'un objet  
transportDiffusion

2

Calcul des  
paramètres  
dérivés

- Pas spatial / de temps limite
- Coeff de transport / diffusion

3

Initialisation des  
vecteurs et de la  
matrice

- vecteurs x et t
- matrice u\_calcule

4

Conditions  
initiales

- vecteurs x et t
- solution exacte pour t=0

## C) Le destructeur

```
TransportDiffusion::~TransportDiffusion() {}
```

## D) Calcul numérique

```
void TransportDiffusion::calculer() {  
    for (int n = 0; n < Nt-1; ++n) {  
        for (int i = 1; i < Nx-1; ++i) {  
            double du = (C > 0) ? (u_calcule[i][n] - u_calcule[i-1][n])  
                                : (u_calcule[i+1][n] - u_calcule[i][n]);  
            u_calcule[i][n+1] = u_calcule[i][n]  
                + gamma * (u_calcule[i+1][n] - 2*u_calcule[i][n] +  
                    u_calcule[i-1][n])  
                - v * du  
                + f(x[i], t[n]) * dt;  
        }  
        u_calcule[0][n+1] = 0.0;  
        u_calcule[Nx-1][n+1] = 0.0;  
    }  
}
```

- > La méthode `calculer()` applique un schéma numérique explicite
- > solution `u_calcule`

## E) Calcul des erreurs

### 1. Erreur L2

```
double TransportDiffusion::erreur_L2() {  
    double sum = 0.0;  
    for (int i = 0; i < Nx; ++i) {  
        double e = std::abs(u_calcule[i][Nt-1] - u_exacte(x[i], T));  
        sum += e * e;  
    }  
    return std::sqrt(dx * sum);  
}
```

> Calcule la différence entre la solution numérique et la solution exacte

### 2. Erreur L<sub>inf</sub>

```
double TransportDiffusion::erreur_Linf() {  
    double max_err = 0.0;  
    for (int i = 0; i < Nx; ++i) {  
        double e = std::abs(u_calcule[i][Nt-1] - u_exacte(x[i], T));  
        if (e > max_err) max_err = e;  
    }  
    return max_err;  
}
```

> Plus grande erreur ponctuelle entre les deux solutions



## Implémentation en C++ : main.cpp

```
#include "TransportDiffusion.h"
#include <iostream>

int main() {
    double C = 1.0, D = 1.0, L = 1.0;
    int Nx = 10000, Nt = 50;
    TransportDiffusion td(C, D, L, Nx, Nt);
    td.calculer();
    std::cout << "Erreur L2 : " << td.erreur_L2() << std::endl;
    std::cout << "Erreur Linf : " << td.erreur_Linf() << std::endl;
    return 0;
}
```

1

Définition des  
constantes

2

Création d'un objet  
transportDiffusion

3

Appel de la  
méthode  
calculer()

4

Affichage des  
erreurs L2 et  
Linf

## *Les tests unitaires*

### Création de la fonction runAllTests dans le fichier main.cpp

```
void runAllTests() {  
    // Test 1: Solution exacte  
    {  
        TransportDiffusion td(1.0, 1.0, 1.0, 100, 100);  
        assert(std::abs(td.u_exacte(0.0, 0.0)) < 1e-12);  
        assert(std::abs(td.u_exacte(1.0, 0.0)) < 1e-12);  
        assert(std::abs(td.u_exacte(0.5, 0.0) - std::sin(M_PI*0.5)) < 1e-12);  
    }  
    std::cout << "Test de la solution exacte reussi" << std::endl;  
  
    // Test 2: Terme source  
    {  
        TransportDiffusion td(1.0, 1.0, 1.0, 100, 100);  
        double f_val = td.f(0.5, 1.0);  
        assert(std::isfinite(f_val));  
    }  
    std::cout << "Test du terme source reussi" << std::endl;  
  
    // Test 3: Calcul  
    {  
        TransportDiffusion td(1.0, 1.0, 1.0, 100, 100);  
        td.calculer();  
        assert(std::isfinite(td.erreur_L2()));  
        assert(std::isfinite(td.erreur_Linf()));  
    }  
    std::cout << "Tests calcul et erreurs reussis" << std::endl;  
}
```



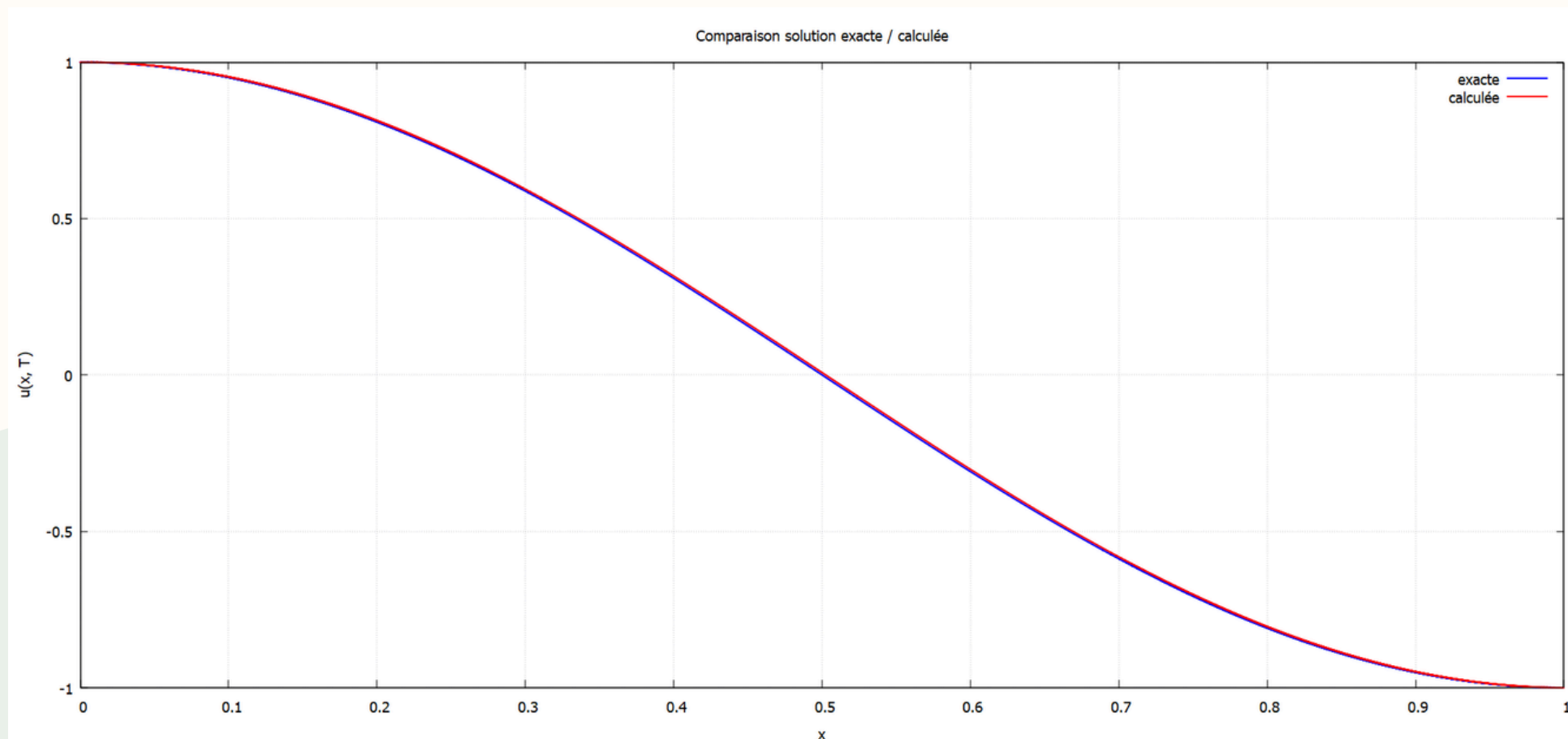
## Implémentation en C++ : Cas\_test\_1.cpp

Solution analytique :  $u(x,t)=\cos(\pi x)*(1 + t)$

Condition initiale (t=0) :  $u(x,0)=\cos(\pi x)$

Conditions aux limites :  $u(0,t) = 1 + t$  ;  $u(1,t) = -(1 + t)$

Terme source :  $f(x, t) = \cos(\pi x) + D(\pi^2 * \cos(\pi x)*(1 + t)) - C (\pi*\sin(\pi x)*(1 + t))$



à l'aide de gnuplot : `td.exportData("C:/Users/elsac/solution.dat")`

Erreur L2	Erreur Linf
0.00544224	0.00769649

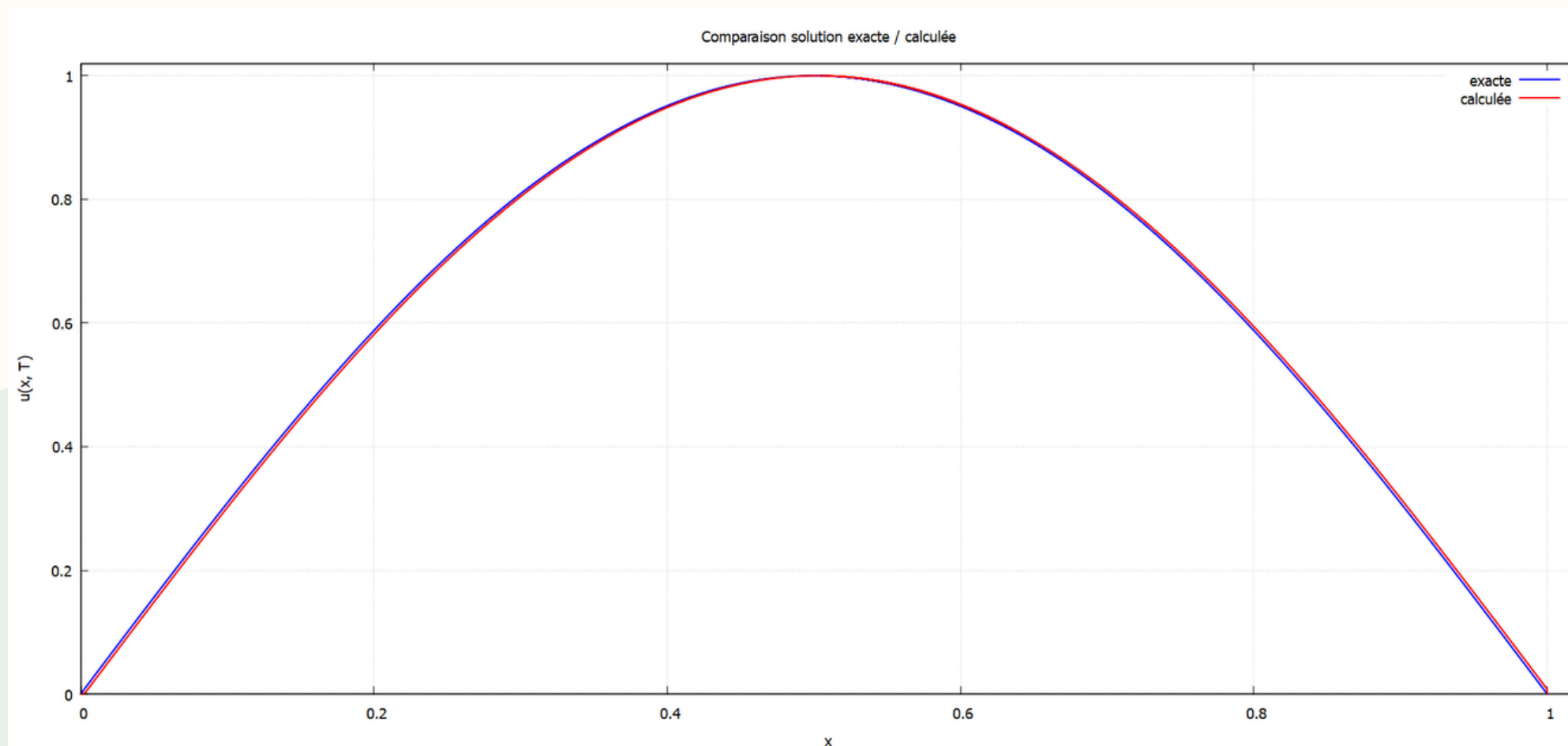
## Implémentation en C++ : Cas\_test\_2.cpp

Solution analytique :  $u(x, t) = \sin(\pi x) \exp(-t)$

Condition initiale (t=0) :  $u(x, 0) = \sin(\pi x)$

Conditions aux limites :  $u(0, t) = 0$  ;  $u(1, t) = 0$

Terme source :  $f(x, t) = -\cos(\pi x) \exp(-t) + D \pi^2 \sin(\pi x) \exp(-t) + C \pi \cos(\pi x) \exp(-t)$



Erreur L2	Erreur Linf
0.00543136	0.00769624

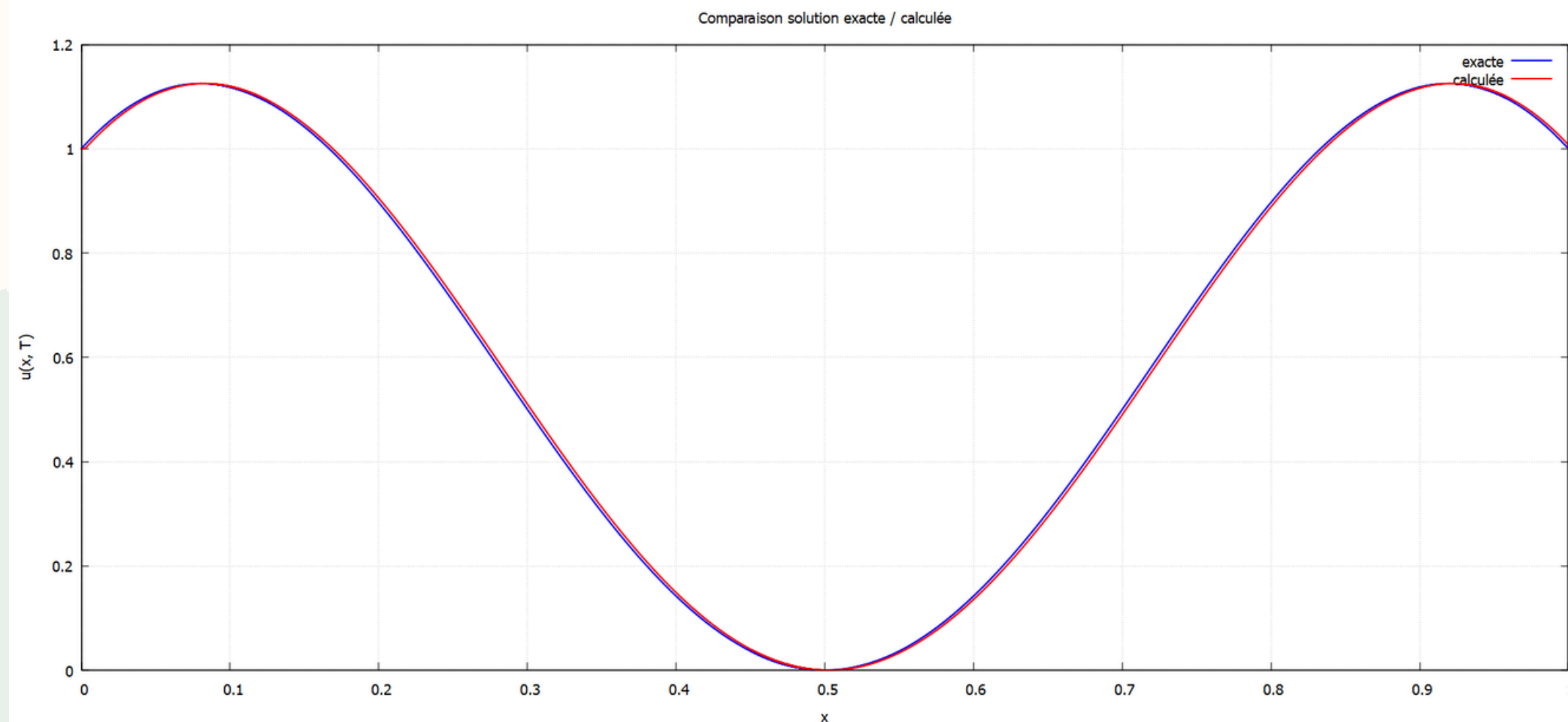
## Implémentation en C++ : Cas\_test\_3.cpp

**Solution analytique :**  $u(x, t) = (\sin(\pi x) + \cos(2\pi x)) \exp(-\pi^2 t)$

**Condition initiale (t=0) :**  $u(x, 0) = \sin(\pi x) + \cos(2\pi x)$

**Conditions aux limites :**  $u(0, t) = \exp(-\pi^2 t)$  ;  $u(1, t) = \exp(-\pi^2 t)$

**Terme source :**  $f(x, t) = -\pi^2(\sin(\pi x) + \cos(2\pi x)) \exp(-\pi^2 t) + D \cdot \pi^2(\sin(\pi x) + 4 \cos(2\pi x)) \exp(-\pi^2 t) + C(\pi \cos(\pi x) - 2\pi \sin(2\pi x)) \exp(-\pi^2 t)$



Erreur L2


0.00688547

Erreur Linf

0.0102145

# Conclusion

- Mettre en œuvre l'encapsulation à travers des classes C++.
- Utiliser des constructeurs et destructeurs pour gérer les objets.
- Implémenter un schéma numérique explicite.
- Comparaison entre les résultats exacts et ceux calculés.
- Calculer les erreurs L2 et Linfini.



**Merci pour votre attention !**