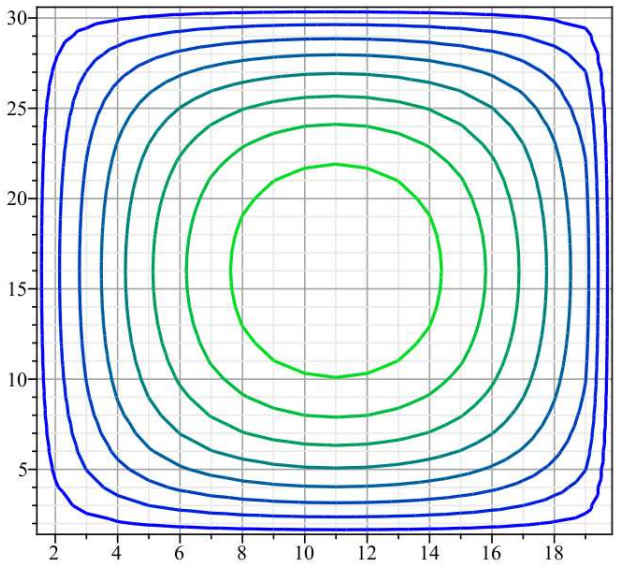
L.MESSAOUDI

# METHODES NUMERIQUES APPLIQUEES

Tome II : Programmes Maple




Edition 2015

Université HL Batna

*Maître de Conférences.*
*Département de Mécanique.*
*Faculté des Technologies.*
*Université Hadj Lakhdar Batna.*

# Table des matières


**1 *Approxiamtions de Taylor*** **1**

1.1 Influence des erreurs d'arrondies . . . . . 1

1.2 Approximations centrées . . . . . 3

1.2.1 Programme AC-01 . . . . . 3

1.2.2 Programme AC-02 . . . . . 4

1.3 Approximations décentrées d'ordre 1 . . . . . 4

1.3.1 Programme AD-01 . . . . . 4

1.3.2 Programme AD-02 . . . . . 5

1.3.3 Programme AD-03 . . . . . 5

1.3.4 Programme AD-04 . . . . . 5

1.4 Approximations décentrées d'ordre 2 . . . . . 6

1.4.1 Programme AD-05 . . . . . 6

1.4.2 Programme AD-06 . . . . . 6

1.4.3 Programme AD-07 . . . . . 7

1.4.4 Programme AD-08 . . . . . 7

**2 *Equation de diffusion de la chaleur*** **9**

2.1 Equation unidimensionnelle . . . . . 9

2.1.1 Programme Diff1D-An . . . . . 9

2.1.2 Programme Diff1D-Num . . . . . 11

2.1.3 Programme Diff1D-Exp . . . . . 14

2.1.4 Programme Diff1D-Imp . . . . . 20

2.1.5 Programme Diff1D-Cr . . . . . 22

2.1.6 Programme Diff1D-ImpMat . . . . . 22

2.1.7 Programme Diff1D-FM-Exp-CD . . . . . 24

2.1.8 Programme Diff1D-FM-01 . . . . . 25

2.1.9 Programme Diff1D-FM-02 . . . . . 26

2.1.10 Programme Diff1D-FM-03 . . . . . 27

2.1.11 Programme Diff1D-CNgd . . . . . 27
2.1.12 Programme Diff1D-FM-04 . . . . . 28
2.1.13 Programme Diff1D-FM-05 . . . . . 29
2.1.14 Programme Diff1D-FM-06 . . . . . 30
2.2 Equation bidimensionnelle . . . . . 31
2.2.1 Programme Diff2D-01 . . . . . 31
2.2.2 Programme Diff2D-02 . . . . . 31
2.3 Equation tridimensionnelle . . . . . 31

**3 *Equations de Laplace et de Poisson* 33**
3.1 Equation bidimensionnelle . . . . . 33
3.1.1 Programme Lap2D-An . . . . . 33
3.1.2 Programme Lap2D-01 . . . . . 35
3.1.3 Programme Lap2D-02 . . . . . 37
3.1.4 Programme Lap2D-03 . . . . . 39
3.1.5 Programme Lap2D-04 . . . . . 42
3.1.6 Programme Lap2D-05 . . . . . 44
3.1.7 Programme Lap2D-06 . . . . . 45
3.1.8 Programme Lap2D-07 . . . . . 46
3.1.9 Programme Lap2D-08 . . . . . 49
3.1.10 Programme Lap2D-09 . . . . . 51
3.1.11 Programme Lap2D-10 . . . . . 51
3.2 Equation Tridimensionnelle . . . . . 51
3.2.1 Programme Lap3D-01 . . . . . 51
3.2.2 Programme Lap3D-02 . . . . . 51

**4 *Equation d'onde* 53**
4.1 Equation unidimensionnelle . . . . . 53
4.1.1 Programme Ond1D-An . . . . . 53
4.1.2 Programme Ond1D-01 . . . . . 55
4.1.3 Programme Ond1D-02 . . . . . 58
4.2 Equation Tridimensionnelle . . . . . 58
4.2.1 Programme Ond2D-01 . . . . . 58
4.2.2 Programme Ond2D-02 . . . . . 58

**5 *Applications utilisant les Maplets* 59**
5.1 Equation de la chaleur . . . . . 59
5.2 Equation de Laplace . . . . . 63
5.3 Equation de Poisson . . . . . 70
5.4 Ecoulements potentiels . . . . . 70

5.5 Ecoulement de Couette généralisé . . . . . 70

**Bibliographie** **79**

Chapitre 1

# *Approxiamtions de Taylor*

## 1.1 Influence des erreurs d'arrondies

Ce premier programme *IErrA* nous montre l'influence des erreurs d'arrondies quand on approxime la fonction $e^x$ par les séries de Taylor. Cette inflence est montrée en traçant sur un même graphe cette fonction ainsi que les différentes approximations.

```
> restart;
```

```
> p7 := convert(taylor(exp(x), x = 0, 7), polynom);
```

$$p7 = 1 + x + 1/2\,x^2 + 1/6\,x^3 + 1/24\,x^4 + \frac{x^5}{120} + \frac{x^6}{720}$$

```
> p6 := convert(taylor(exp(x), x = 0, 6), polynom);
```

$$p6 = 1 + x + 1/2\,x^2 + 1/6\,x^3 + 1/24\,x^4 + \frac{x^5}{120}$$

```
> p5 := convert(taylor(exp(x), x = 0, 5), polynom);
```

$$p5 = 1 + x + 1/2\,x^2 + 1/6\,x^3 + 1/24\,x^4$$

```
> p4 := convert(taylor(exp(x), x = 0, 4), polynom);
```

$$p4 = 1 + x + 1/2\,x^2 + 1/6\,x^3$$

```
> p3 := convert(taylor(exp(x), x = 0, 3), polynom);
```

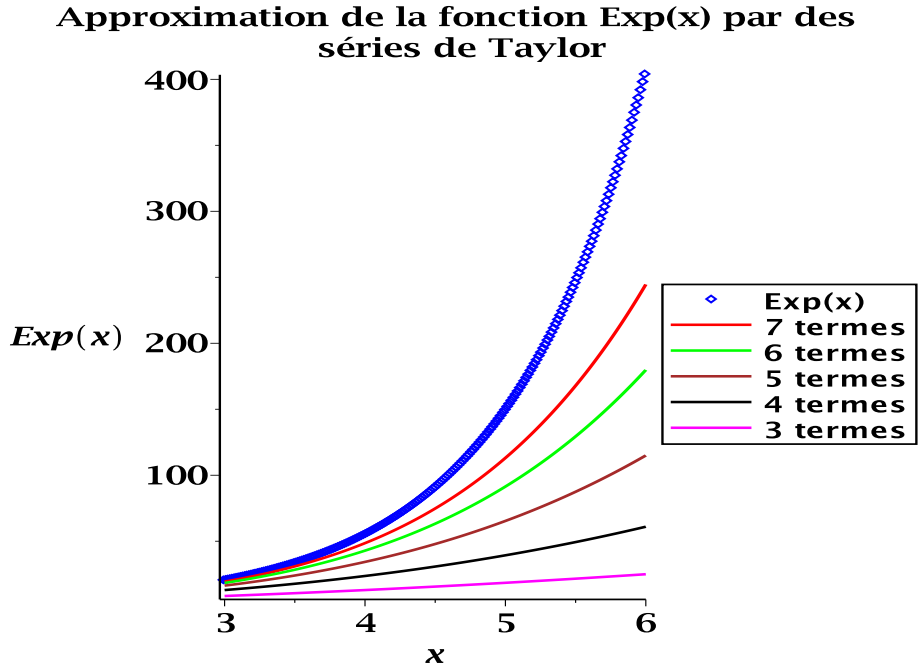
$$p3 = 1 + x + 1/2\,x^2$$

```
> p0 := exp(x);
```

$$p0 = \mathrm{e}^x$$

```
> pp0 := plot(p0, x = 3 .. 6, color = blue, legend = "Exp(x)", style
= point);
pp3 := plot(p3, x = 3 .. 6, color = magenta, legend = "3 termes");
pp4 := plot(p4, x = 3 .. 6, color = black, legend = "4 termes");
```

```
pp5 := plot(p5, x = 3 .. 6, color = brown, legend = "5 termes");
pp6 := plot(p6, x = 3 .. 6, color = green, legend = "6 termes");
pp7 := plot(p7, x = 3 .. 6, color = red, legend = "7 termes");
plots[display]([pp0, pp7, pp6, pp5, pp4, pp3], labels = [x, Exp(x)],
title = "Approximation de la fonction Exp(x) par des séries de Taylor",
font = [TIMES, BOLD, 12], legendstyle = [location = right, font = [HELVETICA,
BOLD, 10]]);
```




Ci-dessous, une autre version plus améliorée de ce programme :

```
> restart;
> ni := 4; ns := 8; a := 3; b := 6;
```

$$ni := 4$$

$$ns := 8$$

$$a := 3$$

$$b := 6$$

```
> f := exp(x);
```

$$f := \mathrm{e}^{x}$$

```
> for i from ni to ns do
p[i] := convert(taylor(f, x = 0, i), polynom)
end do;
```

$$p_4 := 1 + x + 1/2\,x^2 + 1/6\,x^3$$

$$p_5 := 1 + x + 1/2\,x^2 + 1/6\,x^3 + 1/24\,x^4$$

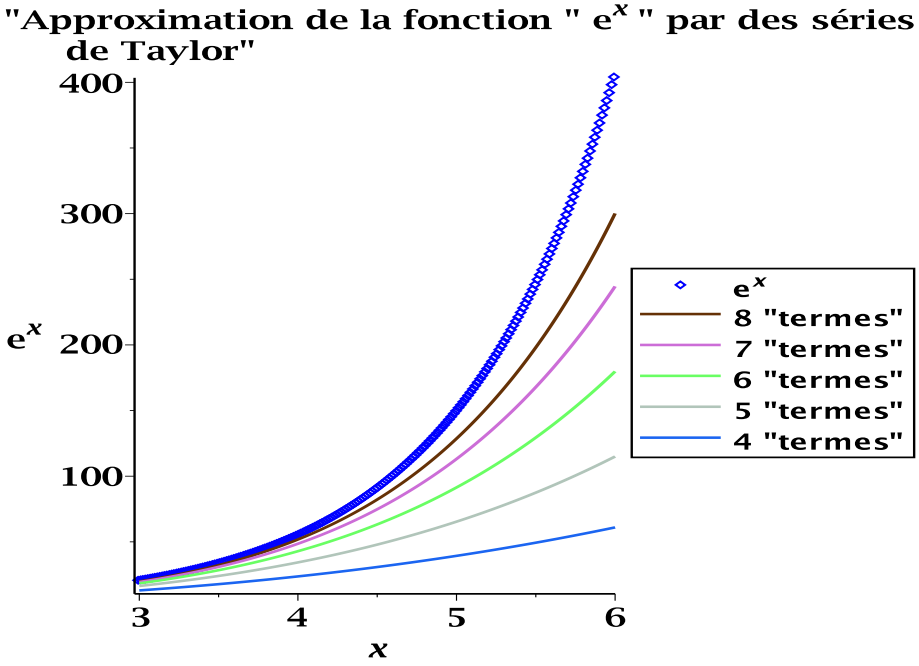
$$p_6 := 1 + x + 1/2\,x^2 + 1/6\,x^3 + 1/24\,x^4 + \frac{x^5}{120}$$

$$p_7 := 1 + x + 1/2\,x^2 + 1/6\,x^3 + 1/24\,x^4 + \frac{x^5}{120} + \frac{x^6}{720}$$

$$p_8 := 1 + x + 1/2\,x^2 + 1/6\,x^3 + 1/24\,x^4 + \frac{x^5}{120} + \frac{x^6}{720} + \frac{x^7}{5040}$$

```
> C0 := plot(f, x = a .. b, color = blue, legend = f, style = point);
for i from ns by -1 to ni do C[i] := plot(p[i], x = a .. b,
color = ColorTools:-Color([rand()/10^12, rand()/10^12, rand()/10^12]),
legend = i*"termes") end do;
plots[display]([C0, seq(C[i], i = ns .. ni, -1)], labels = [x, f],
title = "Approximation de la fonction "*f*" par des séries de Taylor",

font = [TIMES, BOLD, 12], legendstyle = [location = right, font = [HELVETICA,
BOLD, 10]]);
```




## 1.2 Approximations centrées

Les programmes ci-dessous permettent de retrouver toutes les formules des différentes approximations centrées utilisées pour la discrétisation des dérivées d'ordre 1 et 2 :

### 1.2.1 Programme AC-01

Approximation centrée de la dérivée dérivée d'ordre 1 :

```
> restart; N := 2:

> Eq1 := f(x+h) = convert(taylor(f(x+h), h, N), polynom);
```

$$f(x+h) = f(x) + \mathrm{D}(f)(x)\,h$$

```
> Eq2 := f(x-h) = convert(taylor(f(x-h), h, N), polynom);
```

$$f(x-h) = f(x) - \mathrm{D}(f)(x)\,h$$

```
> Eq3 := Eq1 - Eq2;
```

$$f(x+h) - f(x-h) = 2\,\mathrm{D}(f)(x)\,h$$

```
> (D(f))(x) = solve(Eq3, (D(f))(x));
```

$$\mathrm{D}(f)(x) = 1/2\,\frac{f(x+h) - f(x-h)}{h}$$

### 1.2.2 Programme AC-02

Approximation centrée de la dérivée d'ordre 2 :

```
> restart; -1; N := 3:
```

```
> Eq1 := f(x+h) = convert(taylor(f(x+h), h, N), polynom);
```

$$f(x+h) = f(x) + \mathrm{D}(f)(x)\,h + 1/2\,\left(D^{(2)}\right)(f)(x)\,h^2$$

```
> Eq2 := f(x-h) = convert(taylor(f(x-h), h, N), polynom);
```

$$f(x-h) = f(x) - \mathrm{D}(f)(x)\,h + 1/2\,\left(D^{(2)}\right)(f)(x)\,h^2$$

```
> Eq3 := Eq1 + Eq2;
```

$$f(x+h) + f(x-h) = 2\,f(x) + \left(D^{(2)}\right)(f)(x)\,h^2$$

```
> ((D@@2)(f))(x) = solve(Eq3, ((D@@2)(f))(x));
```

$$\left(D^{(2)}\right)(f)(x) = -\frac{2\,f(x) - f(x+h) - f(x-h)}{h^2}$$

## 1.3 Approximations décentrées d'ordre 1

Les programmes ci-dessous permettent de retrouver toutes les formules des différentes approximations décentrées d'ordre 1 utilisées pour la discrétisation des dérivées première et seconde.

### 1.3.1 Programme AD-01

Approximation décentrée avant d'ordre 1 pour la dérivée première.

```
> restart; N := 2:
```

```
> Eq1 := f(x+h) = convert(taylor(f(x+h), h, N), polynom);
```

$$f(x+h) = f(x) + \mathrm{D}(f)(x)\,h$$

```
> (D(f))(x) = solve(Eq1, (D(f))(x));
```

$$\mathrm{D}\left(f\right)\left(x\right) = -\frac{f\left(x\right) - f\left(x+h\right)}{h}$$

### 1.3.2 Programme AD-02

Approximation décentrée avant d'ordre 1 pour la dérivée seconde.

```
> restart; N := 3:
```

```
> Eq1 := f(x+h) = convert(taylor(f(x+h), h, N), polynom);
```

$$f\left(x+h\right) = f\left(x\right) + \mathrm{D}\left(f\right)\left(x\right)h + 1/2\,\left(D^{(2)}\right)\left(f\right)\left(x\right)h^2$$

```
> Eq2 := f(x+2*h) = convert(taylor(f(x+2*h), h, N), polynom);
```

$$f\left(x+2\,h\right) = f\left(x\right) + 2\,\mathrm{D}\left(f\right)\left(x\right)h + 2\,\left(D^{(2)}\right)\left(f\right)\left(x\right)h^2$$

```
> Eq3 := Eq2 - 2*Eq1;
```

$$f\left(x+2\,h\right) - 2\,f\left(x+h\right) = -f\left(x\right) + \left(D^{(2)}\right)\left(f\right)\left(x\right)h^2$$

```
> ((D@@2)(f))(x) = solve(Eq3, ((D@@2)(f))(x))
```

$$\left(D^{(2)}\right)\left(f\right)\left(x\right) = \frac{f\left(x\right) + f\left(x+2\,h\right) - 2\,f\left(x+h\right)}{h^2}$$

### 1.3.3 Programme AD-03

Approximation décentrée arrière d'ordre 1 pour la dérivée première.

```
> restart; N := 2:
```

```
> Eq1 := f(x-h) = convert(taylor(f(x-h), h, N), polynom);
```

$$f\left(x-h\right) = f\left(x\right) - \mathrm{D}\left(f\right)\left(x\right)h$$

```
> (D(f))(x) = solve(Eq1, (D(f))(x));
```

$$\mathrm{D}\left(f\right)\left(x\right) = \frac{f\left(x\right) - f\left(x-h\right)}{h}$$

### 1.3.4 Programme AD-04

Approximation décentrée arrière d'ordre 1 pour la dérivée seconde.

```
> restart; N := 3:
```

```
> Eq1 := f(x-h) = convert(taylor(f(x-h), h, N), polynom);
```

$$f\left(x-h\right) = f\left(x\right) - \mathrm{D}\left(f\right)\left(x\right)h + 1/2\,\left(D^{(2)}\right)\left(f\right)\left(x\right)h^2$$

```
> Eq2 := f(x-2*h) = convert(taylor(f(x-2*h), h, N), polynom);
```

$$f(x-2h) = f(x) - 2\,\mathrm{D}(f)(x)\,h + 2\left(D^{(2)}\right)(f)(x)\,h^2$$

```
> Eq3 := Eq2 - 2*Eq1;
```

$$f(x-2h) - 2f(x-h) = -f(x) + \left(D^{(2)}\right)(f)(x)\,h^2$$

```
> ((D@@2)(f))(x) = solve(Eq3, ((D@@2)(f))(x))
```

$$\left(D^{(2)}\right)(f)(x) = \frac{f(x) + f(x-2h) - 2f(x-h)}{h^2}$$

## 1.4 Approximations décentrées d'ordre 2

Les programmes ci-dessous permettent de retrouver toutes les formules des différentes approximations décentrées d'ordre 2 utilisées pour la discrétisation des dérivées première et seconde.

### 1.4.1 Programme AD-05

Approximation décentrée avant d'ordre 2 pour la dérivée première.

```
> restart; N := 3:
```

```
> Eq1 := f(x+h) = convert(taylor(f(x+h), h, N), polynom);
```

$$f(x+h) = f(x) + \mathrm{D}(f)(x)\,h + 1/2\left(D^{(2)}\right)(f)(x)\,h^2$$

```
> Eq2 := f(x+2*h) = convert(taylor(f(x+2*h), h, N), polynom);
```

$$f(x+2h) = f(x) + 2\,\mathrm{D}(f)(x)\,h + 2\left(D^{(2)}\right)(f)(x)\,h^2$$

```
> Eq3 := Eq2 - 4*Eq1;
```

$$f(x+2h) - 4f(x+h) = -3f(x) - 2\,\mathrm{D}(f)(x)\,h$$

```
> (D(f))(x) = solve(Eq3, (D(f))(x));
```

$$\mathrm{D}(f)(x) = -1/2\,\frac{3f(x) + f(x+2h) - 4f(x+h)}{h}$$

### 1.4.2 Programme AD-06

Approximation décentrée arrière d'ordre 2 pour la dérivée première.

```
> restart; N := 3:
```

```
> Eq1 := f(x-h) = convert(taylor(f(x-h), h, N), polynom);
```

$$f(x-h) = f(x) - \mathrm{D}(f)(x)\,h + 1/2\left(D^{(2)}\right)(f)(x)\,h^2$$

```
> Eq2 := f(x-2*h) = convert(taylor(f(x-2*h), h, N), polynom);
```

$$f(x-2h) = f(x) - 2\,\mathrm{D}(f)(x)\,h + 2\left(D^{(2)}\right)(f)(x)\,h^2$$

```
> Eq3 := Eq2 - 4*Eq1;
```

$$f(x-2h) - 4f(x-h) = -3f(x) + 2\,\mathrm{D}(f)(x)\,h$$

```
> (D(f))(x) = solve(Eq3, (D(f))(x));
```

$$\mathrm{D}(f)(x) = 1/2\,\frac{3f(x) + f(x-2h) - 4f(x-h)}{h}$$

### 1.4.3 Programme AD-07

Approximation décentrée avant d'ordre 2 pour la dérivée seconde.

```
> restart; N := 3:
```

```
> Eq1 := f(x+h) = convert(taylor(f(x+h), h, N), polynom);
```

$$f(x+h) = f(x) + \mathrm{D}(f)(x)\,h + 1/2\,\left(D^{(2)}\right)(f)(x)\,h^2$$

```
> Eq2 := f(x+2*h) = convert(taylor(f(x+2*h), h, N), polynom);
```

$$f(x+2h) = f(x) + 2\,\mathrm{D}(f)(x)\,h + 2\,\left(D^{(2)}\right)(f)(x)\,h^2$$

```
> Eq3 := f(x+3*h) = convert(taylor(f(x+3*h), h, N), polynom);
```

$$f(x+3h) = f(x) + 3\,\mathrm{D}(f)(x)\,h + 9/2\,\left(D^{(2)}\right)(f)(x)\,h^2$$

```
> Eq4 := Eq3 - 4*Eq2 + 5*Eq1;
```

$$f(x+3h) - 4f(x+2h) + 5f(x+h) = 2f(x) - \left(D^{(2)}\right)(f)(x)\,h^2$$

```
> ((D@@2)(f))(x) = solve(Eq4, ((D@@2)(f))(x));
```

$$\left(D^{(2)}\right)(f)(x) = \frac{2f(x) - f(x+3h) + 4f(x+2h) - 5f(x+h)}{h^2}$$

### 1.4.4 Programme AD-08

Approximation décentrée arrière d'ordre 2 pour la dérivée seconde.

```
> restart; N := 3:
```

```
> Eq1 := f(x-h) = convert(taylor(f(x-h), h, N), polynom);
```

$$f(x-h) = f(x) - \mathrm{D}(f)(x)\,h + 1/2\,\left(D^{(2)}\right)(f)(x)\,h^2$$

```
> Eq2 := f(x-2*h) = convert(taylor(f(x-2*h), h, N), polynom);
```

$$f(x-2h) = f(x) - 2\,\mathrm{D}(f)(x)\,h + 2\,\left(D^{(2)}\right)(f)(x)\,h^2$$

```
> Eq3 := f(x-3*h) = convert(taylor(f(x-3*h), h, N), polynom);
```

$$f(x-3h) = f(x) - 3\,\mathrm{D}(f)(x)\,h + 9/2\,\left(D^{(2)}\right)(f)(x)\,h^2$$

```
> Eq4 := Eq3 - 4*Eq2 + 5*Eq1;
```

$$f(x-3h)-4f(x-2h)+5f(x-h)=2f(x)-\left(D^{(2)}\right)(f)(x)h^2$$

```
> ((D@@2)(f))(x) = solve(Eq4, ((D@@2)(f))(x))
```

$$\left(D^{(2)}\right)(f)(x)=\frac{2f(x)-f(x-3h)+4f(x-2h)-5f(x-h)}{h^2}$$

Chapitre 2

# *Equation de diffusion de la chaleur*

## 2.1 Equation unidimensionnelle

Le problème mathématique est défini avec ses conditions aux limites (CL) et sa condition initiale (CI) comme suit :

$$\frac{\partial T(X,\tau)}{\partial \tau} = \frac{\partial^2 T(X,\tau)}{\partial X^2} \qquad 0 \leq X \leq 1\,,\ \tau \geq 0 \tag{2.1}$$

$$\begin{cases} T(X,0) = 1 & 0 \leq X \leq 1 \\ T(0,\tau) = 0 & \tau > 0 \\ T(1,\tau) = 0 & \tau > 0 \end{cases}$$

### 2.1.1 Programme Diff1D-An

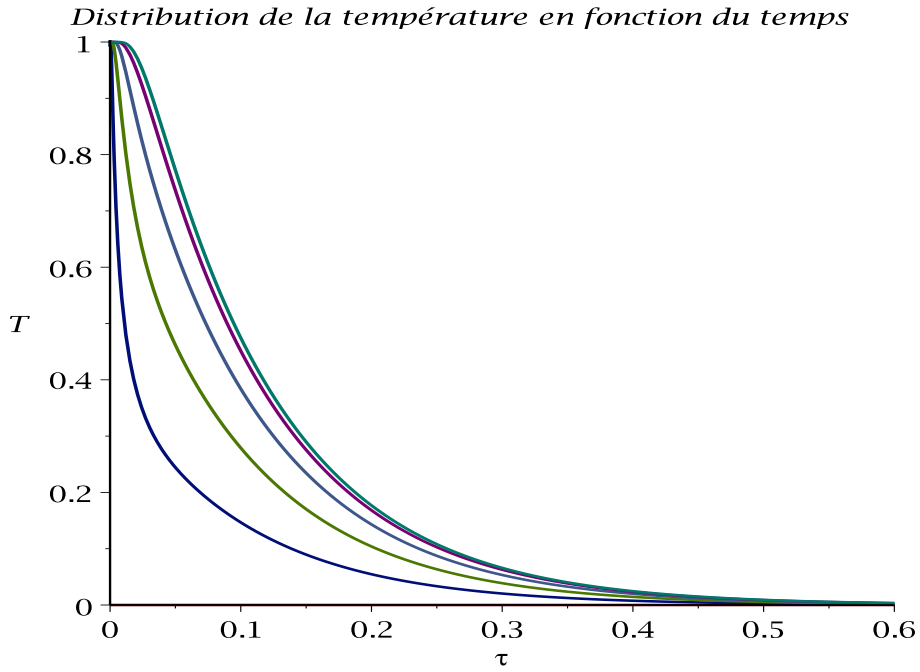
Ce programme donne la solution analytique de l'équation de la chaleur unidimensionnelle.

```
> restart;
```

```
> T(X,τ) := sum(4 sin((2 k+1) π X) e^(-(2 k+1)^2 π^2 τ) / ((2 k+1) π), k = 0 .. 100);
```

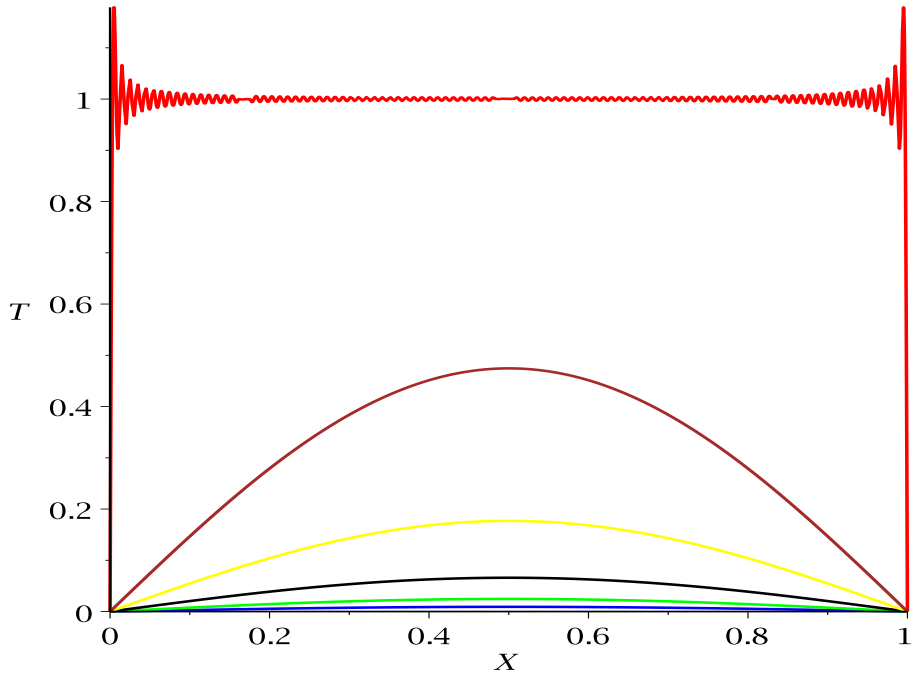
$$T := (X,\tau) \mapsto \sum_{k=0}^{100} 4\,\frac{\sin\left((2\,k+1)\,\pi\,X\right)\mathrm{e}^{-(2\,k+1)^2\pi^2\tau}}{(2\,k+1)\,\pi}$$

```
> plot(T(1,tau), T(.1,tau), T(.2,tau), T(.3,tau), T(.4,tau), T(.5,tau),
tau = 0 .. .6, title = Distribution de la température en fonction du temps,
labels = [‘&tau;‘, T]);
```

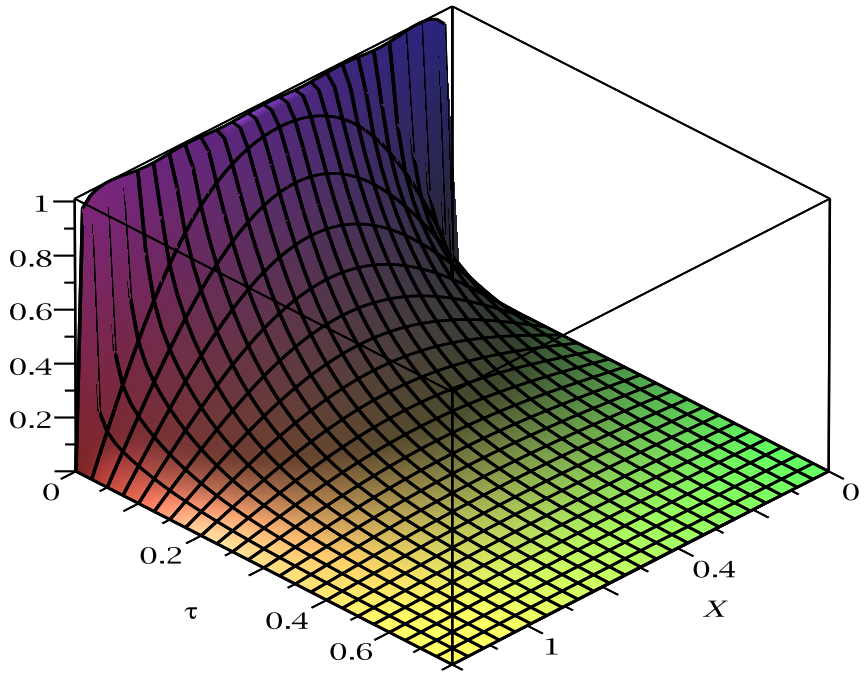



Distribution de la température en fonction de la distance :

```
> plot(T(X,0), T(X,.1), T(X,.2), T(X,.3), T(X,.4), T(X,.5),
X = 0 .. 1, colour = [red, blue, green, black, yellow, brown],
labels = [X,T]);
```




```
> plot3d(T(X,tau), X = 0 .. 1, tau = 0 .. .6, axes = boxed);
```




### 2.1.2 Programme Diff1D-Num

Ce programme permet de déterminer la solution complète de l'équation de diffusion de la chaleur dans une barre de très faible section (1D) et soumise à ses extrémités à des conditions de Dirichlet. Après avoir saisi l'EDP, ses CL et CI (regroupées ici en une seule variable : CLI), la commande *pdsolve* permet de déterminer complètement la solution du problème. Le résultat est donnée sous forme de module regroupant les commandes *plot* (pour les graphiques 2D) , *plot3d* (pour les graphiques 3D), *animate* (pour les graphiques 2D animées) et *value* (pour calculer n'importe quelle valeur de $T(x,t)$ dans le domaine de calcul).

```
> restart: with(plots):
> Eq := diff(T(x,t),t) = diff(diff(T(x,t),x),x);
```

$$Eq := \frac{\partial}{\partial t}T(x,t) = \frac{\partial^2}{\partial x^2}T(x,t)$$

```
> CLI := {T(0,t)=0, T(1,t)=0, T(x,0)=1}
```

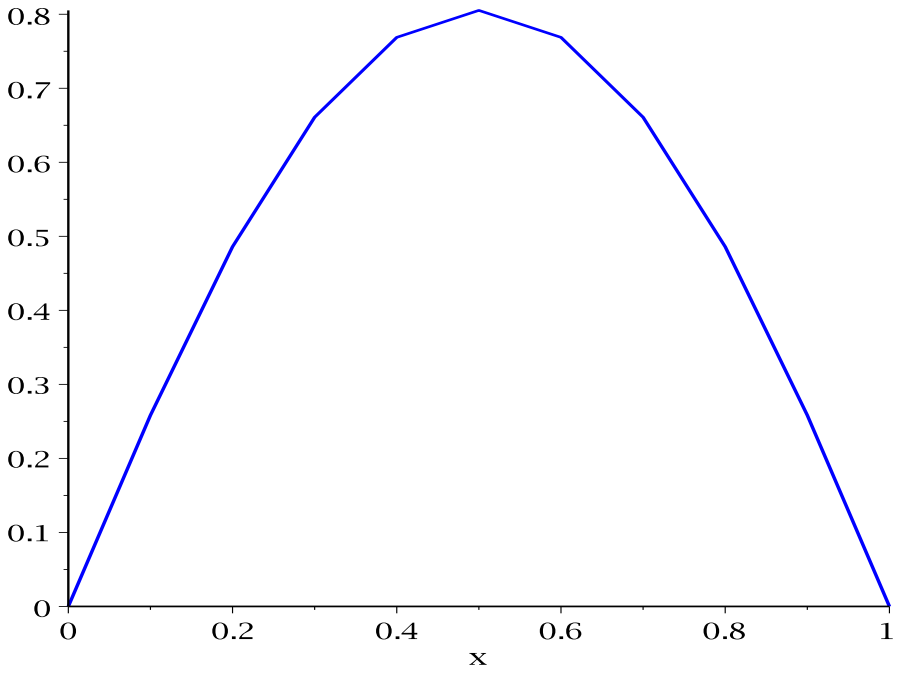
$$CLI := \{T(0,t) = 0, T(1,t) = 0, T(x,0) = 1\}$$

Résolution avec un pas de temps de 0.01 et un pas d'espace de 0.1 et création d'un module de solutions.

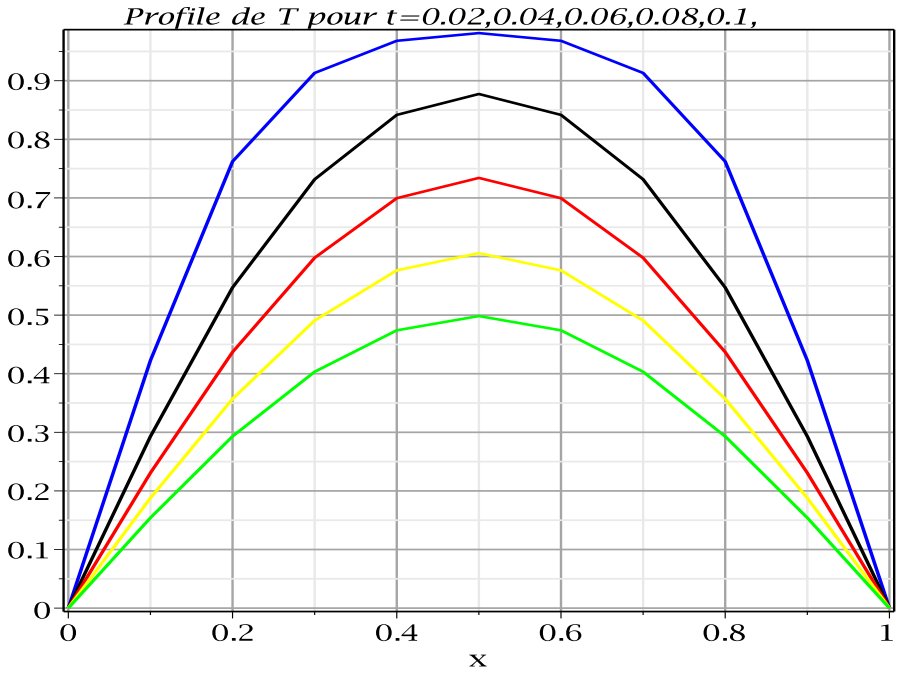
```
> Sol := pdsolve(Eq, CLI, numeric, timestep = 0.01, spacestep = 0.1);
```

$Sol :=$ module () export $plot$, $plot3d$, $animate$, $value$, $settings$; ... endmodule

```
> Sol:-plot(T(x,t), x = 0..1, t=0.05, color = blue);
```



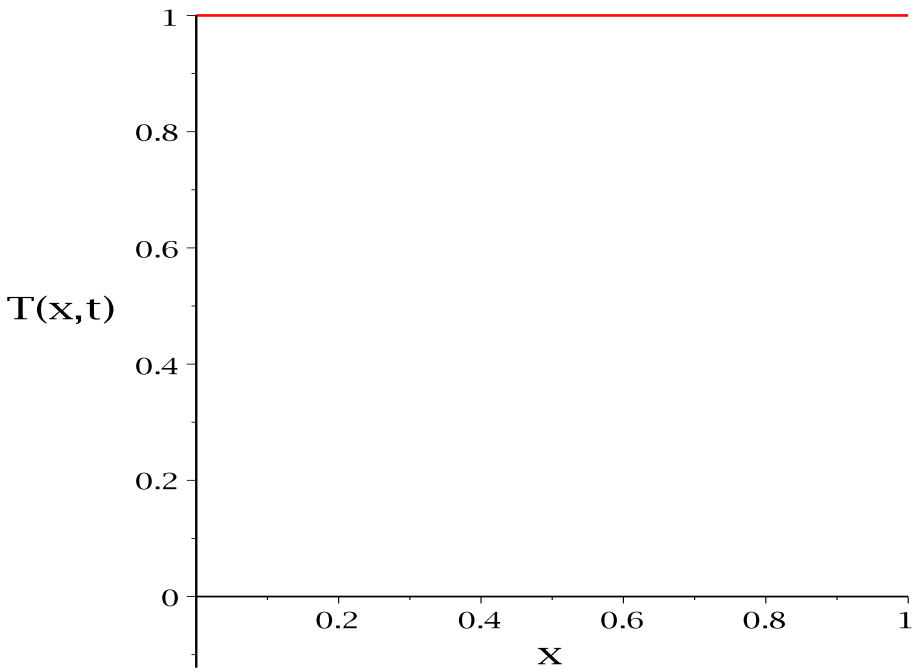

```
> g1 := Sol:-plot(t=0.02, colour=blue);
g2 := Sol:-plot(t=0.04, colour=black);
g3 := Sol:-plot(t=0.06, colour=red);
g4 := Sol:-plot(t=0.08, colour=yellow);
g5 := Sol:-plot(t=0.10, colour=green);
plots[display](g1,g2,g3,g4,g5, gridlines=true, axes=boxed,
title=‘Profile de T pour t=0.02,0.04,0.06,0.08,0.10,‘);
```



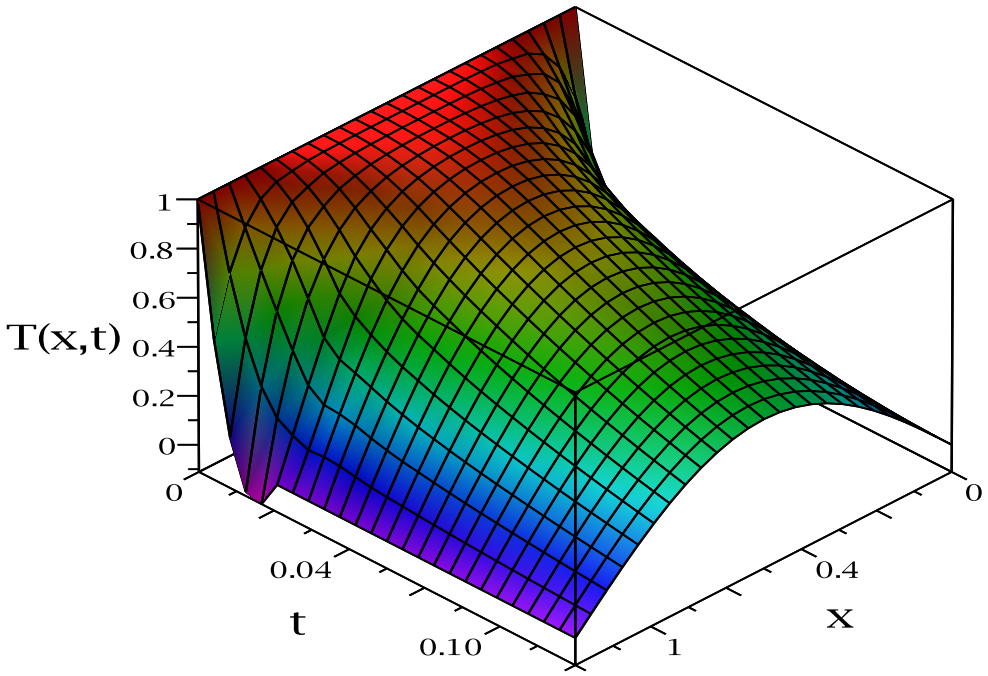

Cette commande permet (en utilisant la barre d'outils) de voir l'animation du refroidissement de la barre jusqu'à l'état stationnaire. La vitesse de défilement est

réglée par le nombre de *frames*.

```
> Sol:-animate(T(x,t),t=0..0.7,frames=50,labels=["x","T(x,t)"]);
```




```
> Sol:-plot3d(T(x,t),t=0..0.1,shading=zhue,axes=boxed,labels=["x","t","T(x,t)"]
```




Pour avoir les valeurs numériques, on crée une procédure utilisant la commande value à partir du module de la solution.

```
> ValTemp := Sol:-value();
```

$$ValTemp := \text{proc}\,()\ldots\text{endproc}$$

Valeur de la température par exemple à la position x=0.1 à l'instant t=0.01 :

```
> ValTemp(0.1, 0.01);
```

$$[x = 0.1, t = 0.01, T(x,t) = 0.732044198895027]$$

### 2.1.3 Programme Diff1D-Exp

Nous pouvons aussi résoudre le même problème en utilisant la programmation standard sous *Maple*. Le programme *Diff1D-Exp* utilise la méthode explicite.

```
> restart; with(plots);
> ‘&Delta;x‘ := .2; ‘&Delta;t‘ := 0.5e-2;
```

$$\Delta x := 0.2$$

$$\Delta t := 0.005$$

```
> lambda := &Delta;t‘/‘&Delta;x‘^2;
```

$$\lambda := 0.1250000000$$

```
> imax := 11;
```

$$imax := 11$$

```
> nmax := 15;
```

$$nmax := 15$$

```
> alpha := 0; beta := 0; sigma := 1;
```

$$\alpha := 0$$

$$\beta := 0$$

$$\sigma := 1$$

Condition initiale :

```
> for i from 2 to imax-1 do T[i, 0] := sigma end do;
```

$$T_{2,0} := 1$$
$$T_{3,0} := 1$$
$$T_{4,0} := 1$$
$$T_{5,0} := 1$$
$$T_{6,0} := 1$$
$$T_{7,0} := 1$$
$$T_{8,0} := 1$$
$$T_{9,0} := 1$$
$$T_{10,0} := 1$$

Condition limite gauche :

```
> for n from 0 to nmax do T[1, n] := alpha end do;
```

$$T_{1,0} := 0$$
$$T_{1,1} := 0$$

$$T_{1,2} := 0$$
$$T_{1,3} := 0$$
$$T_{1,4} := 0$$
$$T_{1,5} := 0$$
$$T_{1,6} := 0$$
$$T_{1,7} := 0$$
$$T_{1,8} := 0$$
$$T_{1,9} := 0$$
$$T_{1,10} := 0$$
$$T_{1,11} := 0$$
$$T_{1,12} := 0$$
$$T_{1,13} := 0$$
$$T_{1,14} := 0$$
$$T_{1,15} := 0$$

Condition limite droite :

```
> for n from 0 to nmax do T[imax, n] := beta end do;
```

Boucle principale :

```
> for n from 0 to nmax do
for i from 2 to imax-1 do
T[i,n+1] := lambda*T[i-1,n]+(1-2*lambda)*T[i,n]+lambda*T[i+1,n]
end do
end do;
```

Affichage de la solution :

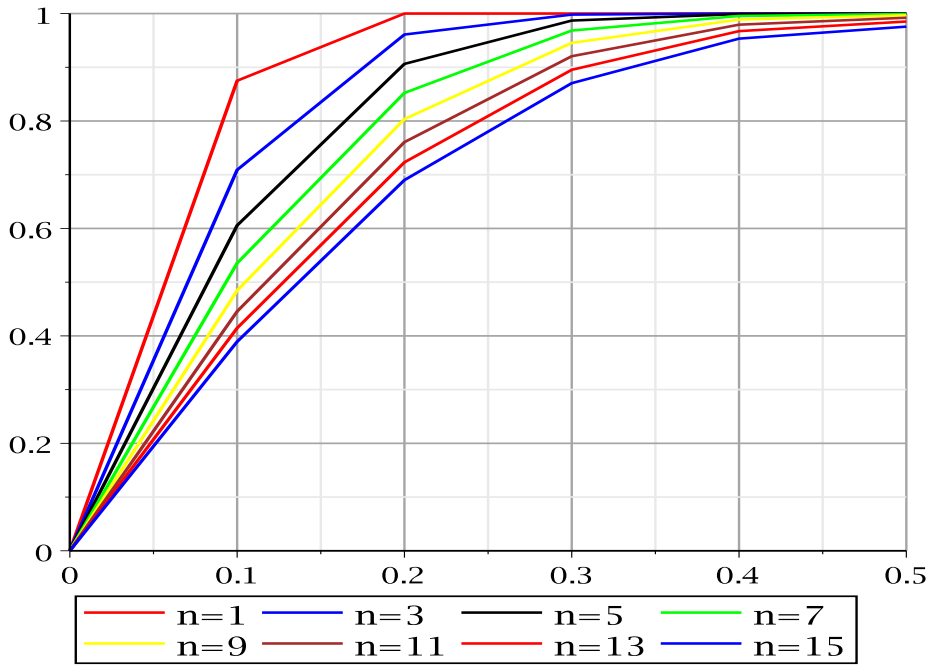
```
> for i from 2 to imax-1 do T[i, nmax] end do;
```

0.3892606696
0.6900617869
0.8704018496
0.9533285673
0.9756458209
0.9533285673
0.8704018496
0.6900617868
0.3892606696

Création des listes pour le tracé sur la première moitié du domaine :

```
> for n to nmax do
liste[n]:=[[0.,T[1,n]],[.1,T[2,n]],[.2,T[3,n]],[.3,T[4,n]],[.4,T[5,n]],[.5,T[6,n]]]
 end do;
```

```
> multiple(listplot, [liste[1], color = red, legend = "n=1"], [liste[3],
color = blue, legend = "n=3"], [liste[5], color = black, legend = "n=5"],
[liste[7], color = green, legend = "n=7"], [liste[9], color = yellow,
legend = "n=9"], [liste[11], color = brown, legend = "n=11"], [liste[13],
color = red, legend = "n=13"], [liste[15], color = blue, legend = "n=15"],
gridlines = true);
```




Création amliorée des listes pour le tracé sur tout le domaine :

```
> for n to nmax do
 liste[n] := [alpha, seq(T[i, n], i = 2 .. imax-1), beta]
 end do
```

$$liste_1 := [0, 0.875, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.875, 0]$$

$$liste_2 := [0, 0.781, 0.984, 1.0, 1.0, 1.0, 1.0, 1.0, 0.984, 0.781, 0]$$

$$liste_3 := [0, 0.709, 0.961, 0.998, 1.0, 1.0, 1.0, 0.998, 0.961, 0.709, 0]$$

$$liste_4 := [0, 0.652, 0.934, 0.994, 1.0, 1.0, 1.0, 0.994, 0.934, 0.652, 0]$$

$$liste_5 := [0, 0.606, 0.906, 0.987, 0.999, 1.0, 0.999, 0.987, 0.906, 0.606, 0]$$

$$liste_6 := [0, 0.568, 0.879, 0.978, 0.998, 1.0, 0.998, 0.978, 0.879, 0.568, 0]$$

$$liste_7 := [0, 0.535, 0.852, 0.968, 0.995, 0.999, 0.995, 0.968, 0.852, 0.535, 0]$$

$$liste_8 := [0, 0.508, 0.827, 0.957, 0.993, 0.998, 0.993, 0.957, 0.827, 0.508, 0]$$

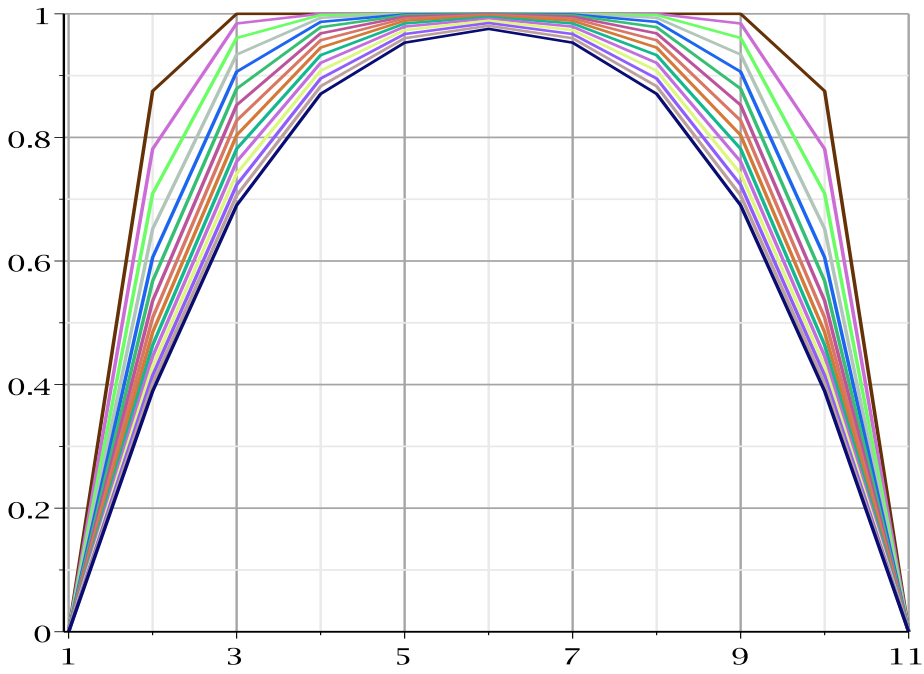
$$
\begin{aligned}
liste_9 &:= [0, 0.485, 0.804, 0.945, 0.989, 0.997, 0.989, 0.945, 0.804, 0.485, 0] \\
liste_{10} &:= [0, 0.464, 0.781, 0.933, 0.984, 0.995, 0.984, 0.933, 0.781, 0.464, 0] \\
liste_{11} &:= [0, 0.446, 0.761, 0.921, 0.979, 0.992, 0.979, 0.921, 0.761, 0.446, 0] \\
liste_{12} &:= [0, 0.429, 0.741, 0.908, 0.974, 0.989, 0.974, 0.908, 0.741, 0.429, 0] \\
liste_{13} &:= [0, 0.415, 0.723, 0.895, 0.967, 0.985, 0.967, 0.895, 0.723, 0.415, 0] \\
liste_{14} &:= [0, 0.401, 0.706, 0.883, 0.961, 0.981, 0.961, 0.883, 0.706, 0.401, 0] \\
liste_{15} &:= [0, 0.389, 0.690, 0.870, 0.953, 0.976, 0.953, 0.870, 0.690, 0.389, 0]
\end{aligned}
$$

Enregistrement dans un fichier pour comparer par la suite avec la solution implicite.

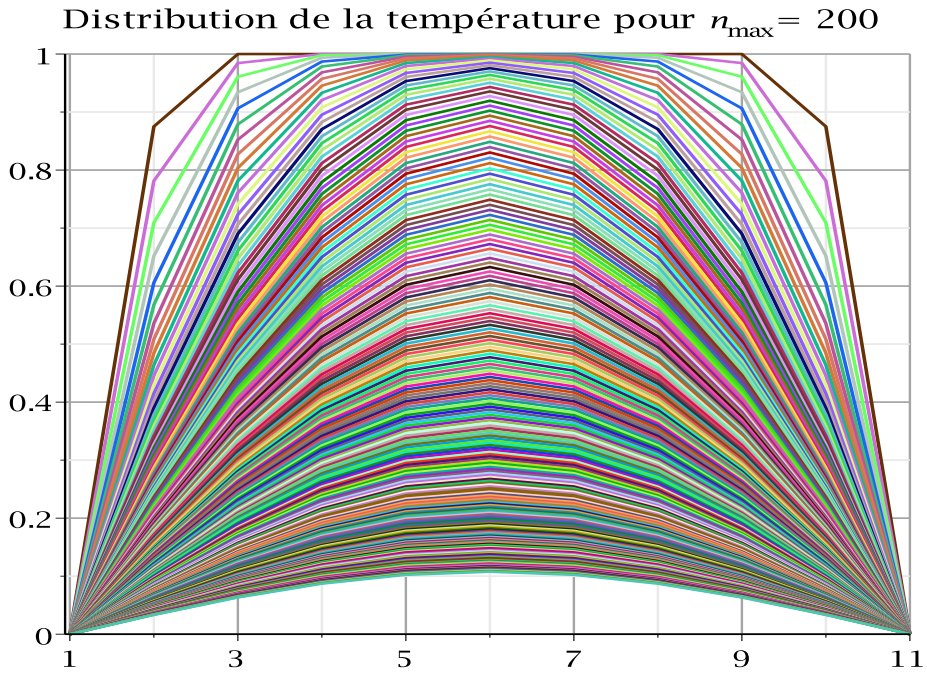
```
> writedata(diffusionExp, liste[nmax])
```

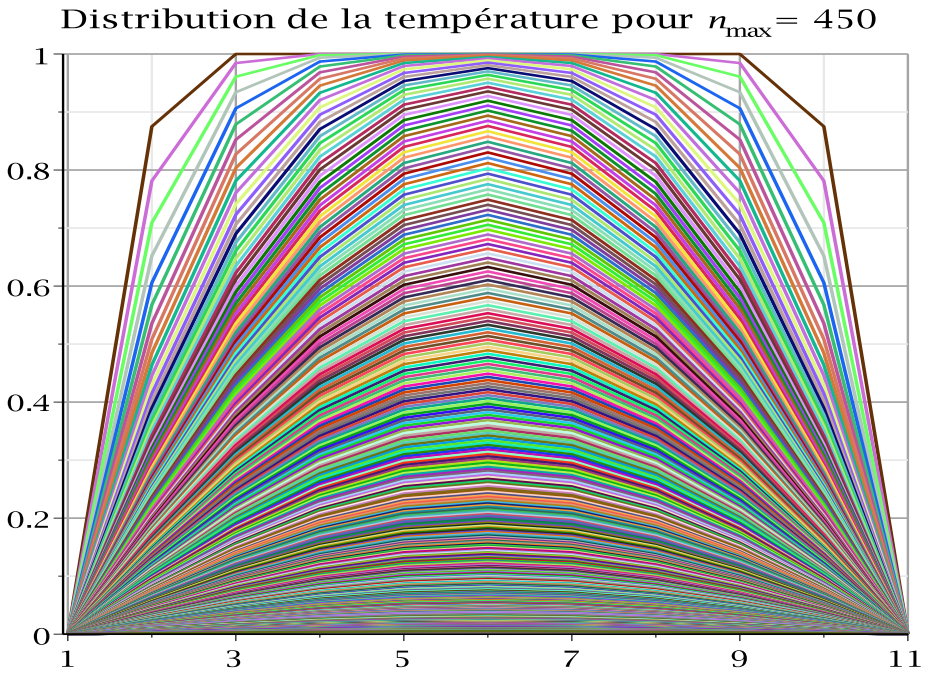
Une autre manière de tracer les courbes avec des couleurs aléatoires :

```
> multiple(listplot, seq([liste[i], color = COLOR(RGB, rand()/10^12,
rand()/10^12, rand()/10^12)], i = 1 .. nmax), gridlines = true);
```



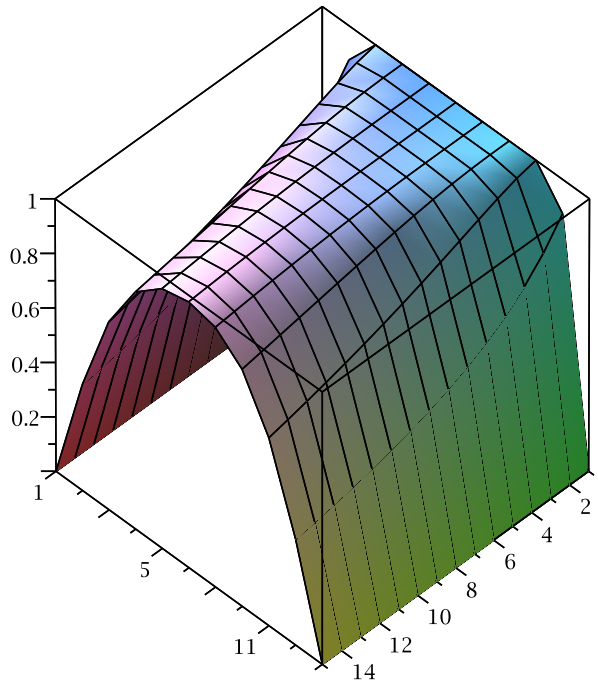

Nous remarquons à partir de ces courbes que l'état stationnaire n'est pas encore atteint. Il peut être déterminé en augmentant *nmax*.



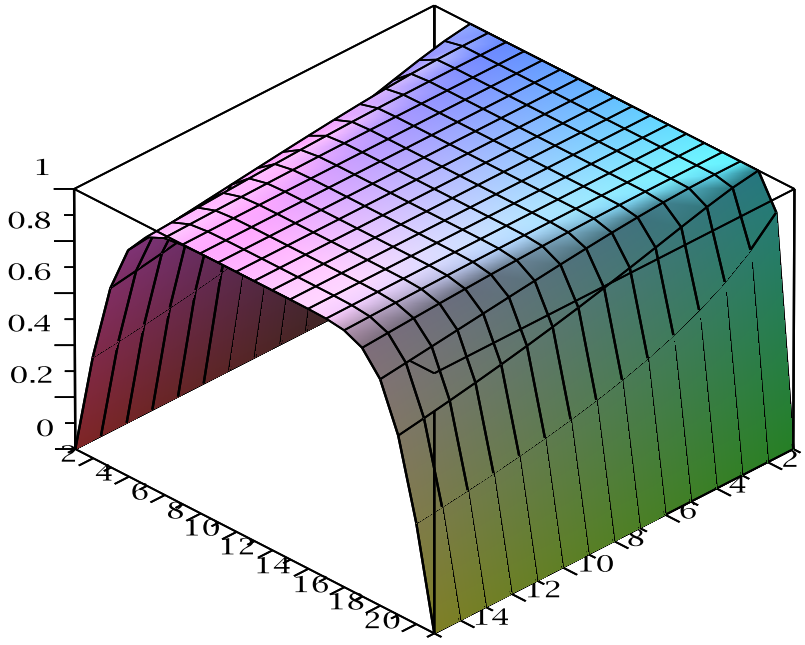

Donc l'état stationnaire est atteint au voisinage de 500 pas de temps c'est à dire après environ 2.5 secondes (Théoriquement, d'après la première partie, ce temps est évalué à environ 6 s). Tracé 3D :

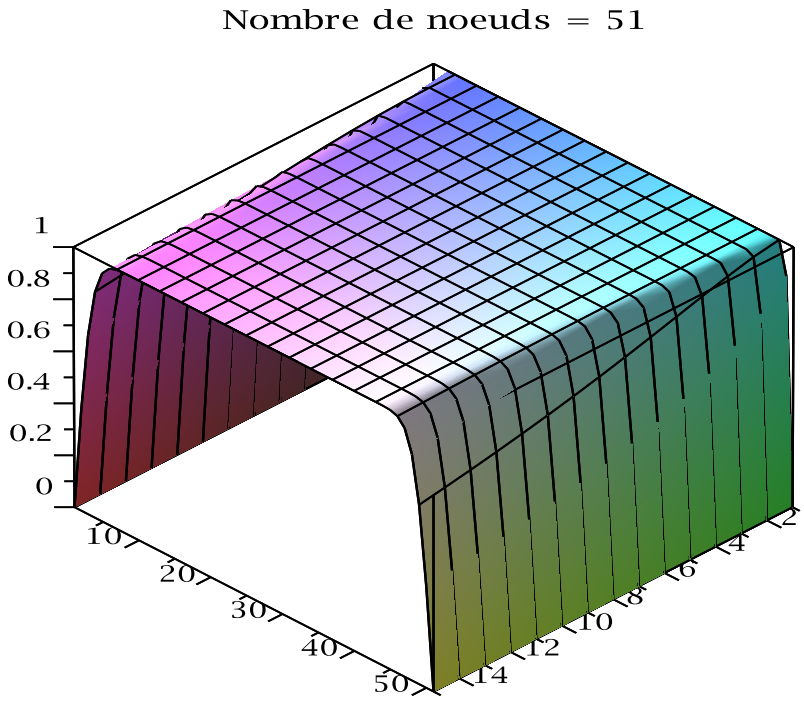
```
> listplot3d([seq([seq(T[i, n], i = 1 .. imax)], n = 1 .. nmax)]);
```




En augmentant le nombre de noeuds à 21 puis à 51 nous aurons :

**Nombre de noeuds = 21**






### 2.1.4 Programme Diff1D-Imp

Nous pouvons aussi résoudre le même problème en utilisant la programmation standard sous *Maple*. Le programme *Diff1D-Imp* utilise la méthode implicite.

```
> restart; with(plots);

> ‘&Delta;x‘ := .2; ‘&Delta;t‘ := 0.5e-2;

> lambda := ‘&Delta;t‘/‘&Delta;x‘^2;

> i_max := 11;

> n_max := 15;

> alpha := 0; beta := 0; sigma := 1;
```

Condition initiale :

```
> for i from 2 to i_max-1 do T[i, 0] := sigma end do;
```

Condition limite gauche :

```
> for n from 0 to n_max do T[1, n] := alpha end do;
```

Condition limite droite :

```
> for n from 0 to n_max do T[i_max, n] := beta end do;
```

Boucle principale :

```
> for n from 0 to n_max do
 k := 1; for i from 2 to i_max-1 do
 Eq[k] := T[i,n] = -lambda*T[i-1,n+1]+(1+2*lambda)*T[i,n+1]-lambda*T[i+1,n+1];
 k := k+1
 end do;
```

```
> for k to i_max-2 do Eq[k] end do;
> Eqs := seq(Eq[k], k = 1 .. i_max-2);
> Temps := [seq(T[i, n+1], i = 2 .. i_max-1)];
> SolTemp := solve(Eqs, Temps);
> for i from 2 to i_max-1 do
T[i, n+1] := rhs(SolTemp[1, i-1])
end do;
```

```
> for n to n_max do
liste[n] := [alpha, seq(T[i, n], i = 2 .. i_max-1), beta]
end do
```

$$liste_1 := [0, 0.899, 0.990, 0.999, 1.0, 1.0, 1.0, 0.999, 0.990, 0.899, 0]$$

$$liste_2 := [0, 0.816, 0.973, 0.996, 1.0, 1.0, 1.0, 0.996, 0.973, 0.816, 0]$$

$$liste_3 := [0, 0.748, 0.953, 0.992, 0.999, 1.0, 0.999, 0.992, 0.953, 0.748, 0]$$

$$liste_4 := [0, 0.692, 0.930, 0.987, 0.998, 0.999, 0.998, 0.987, 0.930, 0.692, 0]$$

$$liste_5 := [0, 0.644, 0.906, 0.979, 0.996, 0.999, 0.996, 0.979, 0.906, 0.644, 0]$$

$$liste_6 := [0, 0.603, 0.882, 0.971, 0.994, 0.998, 0.994, 0.971, 0.882, 0.603, 0]$$

$$liste_7 := [0, 0.569, 0.859, 0.962, 0.991, 0.996, 0.991, 0.962, 0.859, 0.569, 0]$$

$$liste_8 := [0, 0.539, 0.836, 0.952, 0.987, 0.994, 0.987, 0.952, 0.836, 0.539, 0]$$

$$liste_9 := [0, 0.512, 0.814, 0.941, 0.983, 0.992, 0.983, 0.941, 0.814, 0.512, 0]$$

$$liste_{10} := [0, 0.489, 0.793, 0.930, 0.978, 0.989, 0.978, 0.930, 0.793, 0.489, 0]$$

$$liste_{11} := [0, 0.469, 0.774, 0.919, 0.973, 0.986, 0.973, 0.919, 0.774, 0.469, 0]$$

$$liste_{12} := [0, 0.450, 0.755, 0.907, 0.968, 0.982, 0.968, 0.907, 0.755, 0.450, 0]$$

$$liste_{13} := [0, 0.434, 0.737, 0.896, 0.961, 0.978, 0.961, 0.896, 0.737, 0.434, 0]$$

$$liste_{14} := [0, 0.419, 0.720, 0.884, 0.955, 0.974, 0.955, 0.884, 0.720, 0.419, 0]$$

$$liste_{15} := [0, 0.406, 0.703, 0.872, 0.948, 0.968, 0.948, 0.872, 0.703, 0.406, 0]$$

Nous pouvons tracer les même courbes que pour le cas explicite. Enregistrement dans un fichier pour comparer avec la solution explicite.

```
> writedata(diffusionImp, liste[n_max])
```

Lectures des résultats explicite et implicite partir des fichiers :

```
> SolExplicite := readdata(diffusionExp)
```

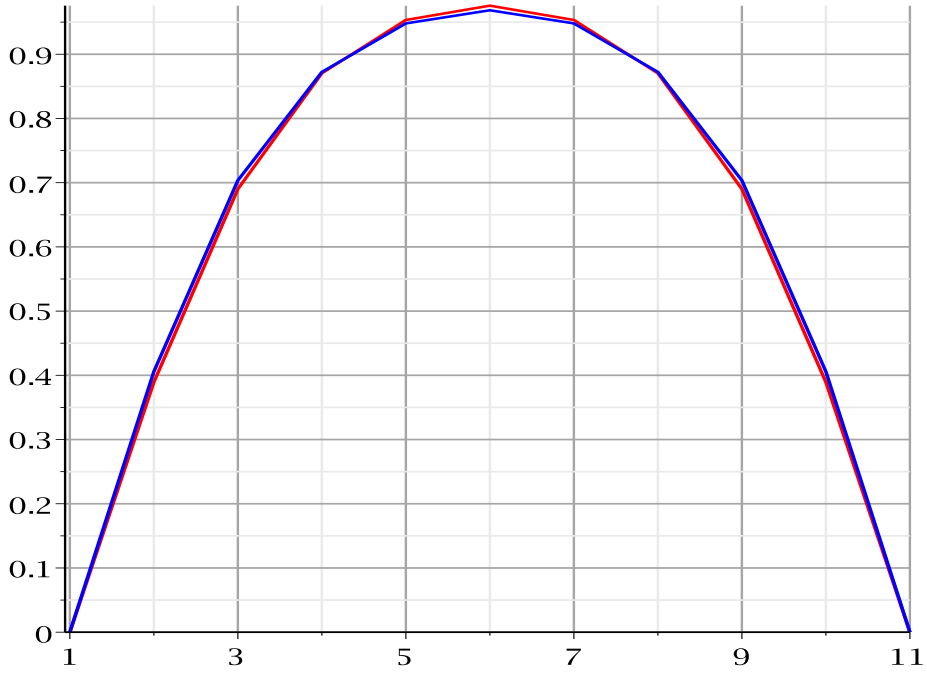
$$SolExplicite := [0.0, 0.389, 0.690, 0.870, 0.953, 0.976, 0.953, 0.870, 0.690, 0.389, 0.0]$$

```
> SolImplicite := readdata(diffusionImp)
```

$$SolImplicite := [0.0, 0.406, 0.703, 0.872, 0.948, 0.968, 0.948, 0.872, 0.703, 0.406, 0.0]$$

Tracé des coubes comparatives :

```
> multiple(listplot, [SolExplicite, color = red],
[SolImplicite, color = blue], gridlines = true)
```




### 2.1.5 Programme Diff1D-Cr

Le même problème pourra être résolu en utilisant la programmation standard sous *Maple* en utilisant le programme *Diff1D-Cr* basé sur la méthode implicite de Crank-Nicholson.

### 2.1.6 Programme Diff1D-ImpMat

Afin de montrer les capacités de *Maple* dans le calcul matriciel, nous pouvons aussi résoudre le problème implicite en utilisant le programme *Diff1D-ImpMat* basé sur la méthode matricielle. Dans ce programme, nous avons préféré utiliser la commande *LinearSolve* au lieu de *MatrixInverse*.

```
> restart; with(plots); with(LinearAlgebra):
> interface(displayprecision = 3);
> `&Delta;x` := .2; `&Delta;t` := 0.5e-2;
> lambda := `&Delta;t`/`&Delta;x`^2;
> i_max := 11;
> n_max := 15;
> alpha := 0; beta := 0; sigma := 1;
```

Condition initiale :

```
> for i from 2 to i_max-1 do T[i, 0] := sigma end do;
```

Condition limite gauche :

```
> for n from 0 to n_max do T[1, n] := alpha end do;
```

Condition limite droite :

```
> for n from 0 to n_max do T[i_max, n] := beta end do;
```

En se référant au cours, nous utiliserons directement les formes matricielles afin de montrer les capacités de Maple dans le calcul matriciel.

```
> a := 1+2*lambda; b := -lambda;
```

$$a := 1.250$$

$$b := -0.125$$

```
> M := BandMatrix([b,a,b], 1, i_max-2, outputoptions = [storage = rectangular])
```

$$M := \begin{bmatrix} 1.250 & -0.125 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.125 & 1.250 & -0.125 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -0.125 & 1.250 & -0.125 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.125 & 1.250 & -0.125 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -0.125 & 1.250 & -0.125 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.125 & 1.250 & -0.125 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -0.125 & 1.250 & -0.125 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -0.125 & 1.250 & -0.125 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.125 & 1.250 \end{bmatrix}$$

```
> R := Matrix(i_max-2, 1, [alpha*lambda+T[2, 0], seq(T[i, 0],
  i = 3 .. i_max-2), beta*lambda+T[i_max-1, 0]]);
```

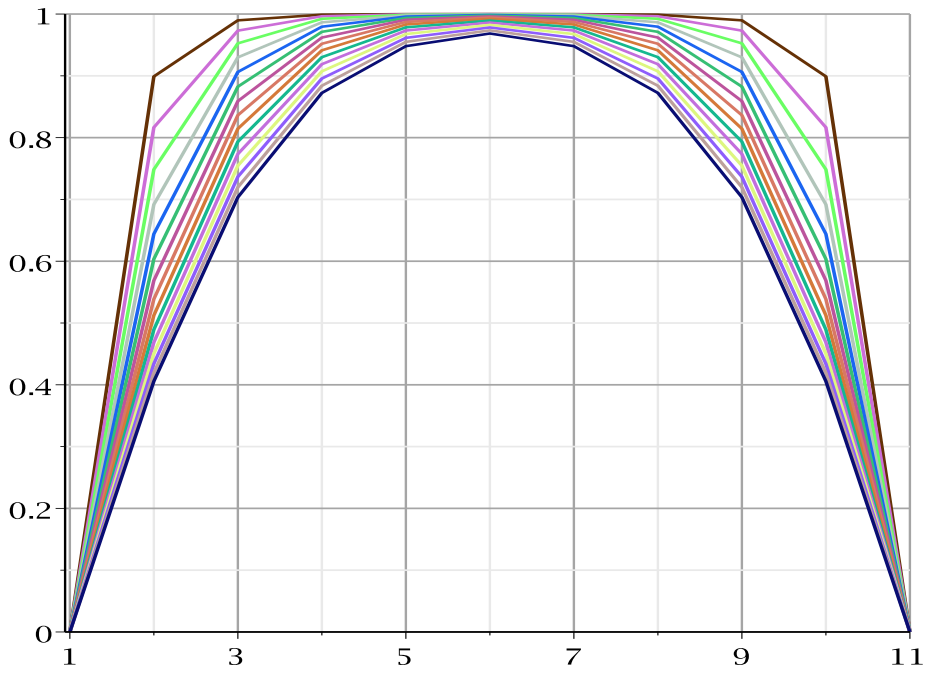
$$R := \begin{bmatrix} 1.0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1.0 \end{bmatrix}$$

Boucle principale :

```
> for n from 0 to n_max do
  T[n+1] := LinearSolve(M, R, method = 'Cholesky');
  R := Matrix(i_max-2, 1, [T[n+1](1)+lambda*alpha, seq(T[n+1](i),
  i = 2 .. i_max-3), T[n+1](i_max-2)+lambda*beta])
  end do;
```

Listes pour le tracé des courbes :

```
> for n to n_max do
 liste[n] := [alpha, seq(T[n](i), i = 1 .. i_max-2), beta]
 end do;
> multiple(listplot, seq([liste[i], color = COLOR(RGB, rand()/10^12,
rand()/10^12, rand()/10^12)], i = 1 .. n_max), gridlines = true);
```




### 2.1.7 Programme Diff1D-FM-Exp-CD

Ce programme donne la forme matricielle $A.T = b$ du schéma explicite de l'équation de diffusion de la chaleur avec des conditions limites gauche et droite de type Dirichlet.

```
> restart; with(LinearAlgebra):
> i_max := 7: n_max := 15: N := i_max-2:
> for i from 2 to i_max-1 do T[i,0] := sigma end do:
> for n from 0 to n_max do T[1,n] := alpha end do:
> for n from 0 to n_max do T[i_max,n] := beta end do:
> n := n_max-1; k := 1:
> for i from 2 to i_max-1 do
Eq[k]:=T[i,n+1]=lambda*T[i-1,n]+(1-2*lambda)*T[i,n]+lambda*T[i+1,n];
k := k+1:
end do:
```

```
> Eqs := [seq(Eq[k], k = 1 .. N)]:

> Tmps := [seq(T[i,n], i = 2 .. i_max-1)]:

> A, b := GenerateMatrix(Eqs, Tmps);
```

$$A,\, b := \begin{bmatrix} 1-2\,\lambda & \lambda & 0 & 0 & 0 \\ \lambda & 1-2\,\lambda & \lambda & 0 & 0 \\ 0 & \lambda & 1-2\,\lambda & \lambda & 0 \\ 0 & 0 & \lambda & 1-2\,\lambda & \lambda \\ 0 & 0 & 0 & \lambda & 1-2\,\lambda \end{bmatrix}, \begin{bmatrix} T_{2,15}-\lambda\,\alpha \\ T_{3,15} \\ T_{4,15} \\ T_{5,15} \\ T_{6,15}-\lambda\,\beta \end{bmatrix}$$

### 2.1.8 Programme Diff1D-FM-01

Ce programme donne la forme matricielle $A\,.\,T = b$ du schéma explicite de l'équation de diffusion de la chaleur avec une condition limite gauche de type Neumann discrétisée par une approximation centrée.

```
> restart; with(LinearAlgebra):

> i_max := 7: n_max := 15:

> N := i_max-1:

> for i from 2 to i_max-1 do T[i,0] := sigma end do:

> for n from 0 to n_max do T[i_max,n] := beta end do:
```

Boucle principale :

```
> n := n_max-1: k := 1:

> T[0,n] := T[2,n] - 2*alpha*‘&Delta;x‘:
for i to i_max-1 do
Eq[k]:=T[i,n+1]= lambda*T[i-1,n]+(1-2*lambda)*T[i,n]+lambda*T[i+1,n];
k := k+1:
end do:

> Eqs := [seq(Eq[k], k = 1 .. N)]:

> Tmps := [seq(T[i,n], i = 1 .. i_max-1)]:

> A, b := GenerateMatrix(Eqs, Tmps);
```

$$A,\, b := \begin{bmatrix} 1-2\,\lambda & 2\,\lambda & 0 & 0 & 0 & 0 \\ \lambda & 1-2\,\lambda & \lambda & 0 & 0 & 0 \\ 0 & \lambda & 1-2\,\lambda & \lambda & 0 & 0 \\ 0 & 0 & \lambda & 1-2\,\lambda & \lambda & 0 \\ 0 & 0 & 0 & \lambda & 1-2\,\lambda & \lambda \\ 0 & 0 & 0 & 0 & \lambda & 1-2\,\lambda \end{bmatrix}, \begin{bmatrix} T_{1,15}+2\,\lambda\,\alpha\,\Delta x \\ -T_{2,15} \\ -T_{3,15} \\ -T_{4,15} \\ -T_{5,15} \\ T_{6,15}-\lambda\,\beta \end{bmatrix}$$

### 2.1.9 Programme Diff1D-FM-02

Ce programme donne la forme matricielle $A\,.\,T = b$ du schéma explicite de l'équation de diffusion de la chaleur avec une condition limite droite de type Neumann discrétisée par une approximation centrée.

```
> restart; with(LinearAlgebra):
> i_max := 7: n_max := 15:
> N := i_max - 1:
> for i from 2 to i_max-1 do T[i,0] := sigma end do:
> for n from 0 to n_max do T[1,n] := alpha end do:
```

Boucle principale :

```
> n := n_max - 1: k := 1:
> T[i_max+1,n] := T[i_max-1,n] + 2*beta*‘&Delta;x‘;
for i from 2 to i_max do
Eq[k]:=T[i,n+1]=lambda*T[i-1,n]+(1-2*lambda)*T[i,n]+lambda*T[i+1,n];
k := k+1:
end do:
> Eqs := [seq(Eq[k], k = 1 .. N)]:
> Tmps := [seq(T[i,n], i = 2 .. i_max)]:
> A, b := GenerateMatrix(Eqs, Tmps);
```

$$A,\, b := \begin{bmatrix} 1-2\,\lambda & \lambda & 0 & 0 & 0 & 0 \\ \lambda & 1-2\,\lambda & \lambda & 0 & 0 & 0 \\ 0 & \lambda & 1-2\,\lambda & \lambda & 0 & 0 \\ 0 & 0 & \lambda & 1-2\,\lambda & \lambda & 0 \\ 0 & 0 & 0 & \lambda & 1-2\,\lambda & \lambda \\ 0 & 0 & 0 & 0 & 2\,\lambda & 1-2\,\lambda \end{bmatrix}, \begin{bmatrix} T_{2,15}-\lambda\,\alpha \\ T_{3,15} \\ T_{4,15} \\ T_{5,15} \\ T_{6,15} \\ T_{7,15}-2\,\lambda\,\beta\,\Delta x \end{bmatrix}$$

### 2.1.10 Programme Diff1D-FM-03

Ce programme donne la forme matricielle $A\,.\,T = b$ du schéma explicite de l'équation de diffusion de la chaleur avec des conditions limites gauche et droite de type Neumann discrétisées par une approximation centrée.

```
> restart; with(LinearAlgebra):
> i_max := 7; n_max := 15:
> N := i_max:
> for i from 2 to i_max-1 do T[i,0] := sigma end do:
```

Boucle principale :

```
> n := n_max-1;
> T[0,n] := T[2,n] - 2*alpha*‘&Delta;x‘:
T[i_max+1,n] :=T[i_max-1,n] + 2*beta*‘&Delta;x‘:
for i to N do
Eq[i]:=lambda*T[i-1,n]+(1-2*lambda)*T[i,n]+lambda*T[i+1,n]=T[i,n+1]
end do:
> Eqs := [seq(Eq[i], i = 1 .. N)]:
> Tmps := [seq(T[i,n], i = 1 .. N)]:
> A, b := GenerateMatrix(Eqs, Tmps)
```

$$A,\,b := \left[\begin{array}{ccccccc} 1-2\,\lambda & 2\,\lambda & 0 & 0 & 0 & 0 & 0 \\ \lambda & 1-2\,\lambda & \lambda & 0 & 0 & 0 & 0 \\ 0 & \lambda & 1-2\,\lambda & \lambda & 0 & 0 & 0 \\ 0 & 0 & \lambda & 1-2\,\lambda & \lambda & 0 & 0 \\ 0 & 0 & 0 & \lambda & 1-2\,\lambda & \lambda & 0 \\ 0 & 0 & 0 & 0 & \lambda & 1-2\,\lambda & \lambda \\ 0 & 0 & 0 & 0 & 0 & 2\,\lambda & 1-2\,\lambda \end{array}\right], \left[\begin{array}{c} T_{1,15}+2\,\lambda\,\alpha\,\Delta x \\ T_{2,15} \\ T_{3,15} \\ T_{4,15} \\ T_{5,15} \\ T_{6,15} \\ T_{7,15}-2\,\lambda\,\beta\,\Delta x \end{array}\right]$$

### 2.1.11 Programme Diff1D-CNgd

Ce programme donne un exemple de calcul du schéma explicite appliqué à l'équation de diffusion avec des conditions limites gauche et droite de type Neumann discrétisées par une approximation centrée.

```
> restart;
> i_max := 11: n_max := 15:
> N := i_max: lambda := .125: alpha := 0: beta := 0: ‘&Delta;x‘ := .1:
> for i from 2 to i_max-1 do T[i,0] := 1 end do:
```

Ce problème nécessite les deux valeurs initiales :

```
> T[1,0] := .25: T[N+1,0] := .25:
> for n from 0 to n_max do
T[0,n] := -2*alpha*‘&Delta;x‘+T[2,n]:
T[i_max+1,n] := 2*beta*‘&Delta;x‘+T[i_max-1,n]:
for i to N do
 T[i,n+1]:=lambda*T[i-1,n]+(1-2*lambda)*T[i,n]+lambda*T[i+1,n]
end do
end do:
> for i to i_max do T[i,n_max] end do
```

0.8438552947
0.8641862362
0.9101150160
0.9539883235
0.9804516147
0.9886623256
0.9804516147
0.9539883235
0.9101150160
0.8641862362
0.8438552947

### 2.1.12 Programme Diff1D-FM-04

Ce programme donne la forme $A\,.\,T = b$ matricielle du schéma explicite de l'équation de diffusion de la chaleur avec une condition limite gauche de type Neumann discrétisée par une approximation décentrée avant d'ordre 1.

```
> restart; with(LinearAlgebra):
> i_max := 7: n_max := 15:
> N := i_max-2:
> for i from 2 to i_max-1 do T[i,0] := sigma end do:
> for n from 0 to n_max do T[i_max,n] := beta end do:
```

Boucle principale :

```
> n := n_max: k := 1:
```

```
> T[1,n] := T[2,n] - alpha*‘&Delta;x‘;
for i from 2 to i_max-1 do
 Eq[k]:=T[i,n+1]=lambda*T[i-1,n]+(1-2*lambda)*T[i,n]+lambda*T[i+1,n]:

k := k+1:
end do:
```

```
> Eqs := [seq(Eq[k], k = 1 .. N)]:
```

```
> Tmps := [seq(T[i,n], i = 2 .. i_max-1)]:
```

```
> A, b := GenerateMatrix(Eqs, Tmps);
```

$$A,\, b := \begin{bmatrix} 1-\lambda & \lambda & 0 & 0 & 0 \\ \lambda & 1-2\,\lambda & \lambda & 0 & 0 \\ 0 & \lambda & 1-2\,\lambda & \lambda & 0 \\ 0 & 0 & \lambda & 1-2\,\lambda & \lambda \\ 0 & 0 & 0 & \lambda & 1-2\,\lambda \end{bmatrix}, \begin{bmatrix} T_{2,16}+\lambda\,\alpha\,\Delta x \\ T_{3,16} \\ T_{4,16} \\ T_{5,16} \\ T_{6,16}-\lambda\,\beta \end{bmatrix}$$

### 2.1.13 Programme Diff1D-FM-05

Ce programme donne la forme matricielle $A\,.\,T = b$ du schéma explicite de l'équation de diffusion de la chaleur avec une condition limite gauche de type Neumann discrétisée par une approximation décentrée avant d'ordre 2.

```
> restart; with(LinearAlgebra):
```

```
> i_max := 7: n_max := 15:
```

```
> N := i_max-2:
```

```
> n := n_max-1: k := 1:
```

```
> T[1,n] := 1/3*(-2*alpha*‘&Delta;x‘+4*T[2,n]-T[3,n]);
```

$$T_{1,14} := -2/3\,\alpha\,\Delta x + 4/3\,T_{2,14} - 1/3\,T_{3,14}$$

```
> T[i_max,n] := 1/3*(2*beta*‘&Delta;x‘-T[i_max-2,n]+4*T[i_max-1,n]);
```

$$T_{7,14} := 2/3\,\beta\,\Delta x - 1/3\,T_{5,14} + 4/3\,T_{6,14}$$

```
> for i from 2 to i_max-1 do
Eq[k]:=T[i,n+1]=lambda*T[i-1,n]+(1-2*lambda)*T[i,n]+lambda*T[i+1,n];
k := k+1
end do:
```

```
> Eqs := [seq(Eq[k], k = 1 .. N)]:
```

```
> Tmps := [seq(T[i,n], i = 2 .. i_max-1)]:
```

```
> A, b := GenerateMatrix(Eqs, Tmps);
```

$$A,\, b := \begin{bmatrix} -2/3\,\lambda+1 & 2/3\,\lambda & 0 & 0 & 0 \\ \lambda & 1-2\,\lambda & \lambda & 0 & 0 \\ 0 & \lambda & 1-2\,\lambda & \lambda & 0 \\ 0 & 0 & \lambda & 1-2\,\lambda & \lambda \\ 0 & 0 & 0 & 2/3\,\lambda & -2/3\,\lambda+1 \end{bmatrix}, \begin{bmatrix} T_{2,15}+2/3\,\lambda\,\alpha\,\Delta x \\ T_{3,15} \\ T_{4,15} \\ T_{5,15} \\ T_{6,15}-2/3\,\lambda\,\beta\,\Delta x \end{bmatrix}$$

### 2.1.14 Programme Diff1D-FM-06

Ce programme donne la forme matricielle $A\,.\,T = b$ du schéma implicite de l'équation de diffusion de la chaleur avec des conditions limites gauche et droite de type Dirichlet.

```
> restart; with(LinearAlgebra):

> i_max := 7: n_max := 15: N := i_max-2:

> for i from 2 to i_max-1 do T[i,0] := sigma end do:

> for n from 0 to n_max do T[1,n] := alpha end do:

> for n from 0 to n_max do T[i_max,n] := beta end do:

> n := n_max-1: k := 1:
> for i from 2 to i_max-1 do
Eq[k] := -lambda*T[i-1,n+1]+(1+2*lambda)*T[i,n+1]-lambda*T[i+1,n+1] =
T[i,n];
k := k+1:
end do:

> Eqs := [seq(Eq[k], k = 1 .. N)]:

> Tmps := [seq(T[i, n+1], i = 2 .. i_max-1)]:

> A, b := GenerateMatrix(Eqs, Tmps);
```

$$A,\, b := \begin{bmatrix} 1+2\,\lambda & -\lambda & 0 & 0 & 0 \\ -\lambda & 1+2\,\lambda & -\lambda & 0 & 0 \\ 0 & -\lambda & 1+2\,\lambda & -\lambda & 0 \\ 0 & 0 & -\lambda & 1+2\,\lambda & -\lambda \\ 0 & 0 & 0 & -\lambda & 1+2\,\lambda \end{bmatrix}, \begin{bmatrix} T_{2,14}+\lambda\,\alpha \\ T_{3,14} \\ T_{4,14} \\ T_{5,14} \\ T_{6,14}+\lambda\,\beta \end{bmatrix}$$

## 2.2 Equation bidimensionnelle

### 2.2.1 Programme Diff2D-01

### 2.2.2 Programme Diff2D-02

## 2.3 Equation tridimensionnelle

Chapitre 3

# *Equations de Laplace et de Poisson*

## 3.1 Equation bidimensionnelle

Le problème mathématique est défini avec ses conditions aux limites (CL) comme suit :

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 \qquad 0 \leq x \leq a \ , 0 \leq y \leq b \tag{3.1}$$

$$\begin{cases} T(0,y) = 0 & , \quad T(x,0) = 0 \\ T(a,y) = 0 & , \quad T(x,b) = T_0 \end{cases}$$

### 3.1.1 Programme Lap2D-An

Ce programme donne la solution analytique de l'équation de Laplace 2D. Lap2D-An détermine la température à travers une plaque rectangulaire dont les extrémités sont maintenues à des tempratures constantes.

```
> restart;
```

```
> T(x,y) := (4*T0/Pi) * sum('sin((2*k-1)*Pi*x/a)*sinh((2*k-1)*Pi*y/a)/((2*k-1)*sinh((2*k-1)*Pi*b/a))', k=1..100);
```

$$T := (x,y) \mapsto 4\,\frac{T0}{\pi}\sum_{k=1}^{100}\frac{1}{2\,k-1}\sin\left(\frac{(2\,k-1)\,\pi\,x}{a}\right)\sinh\left(\frac{(2\,k-1)\,\pi\,y}{a}\right)\left(\sinh\left(\frac{(2\,k-1)\,\pi\,b}{a}\right)\right)^{-1}$$

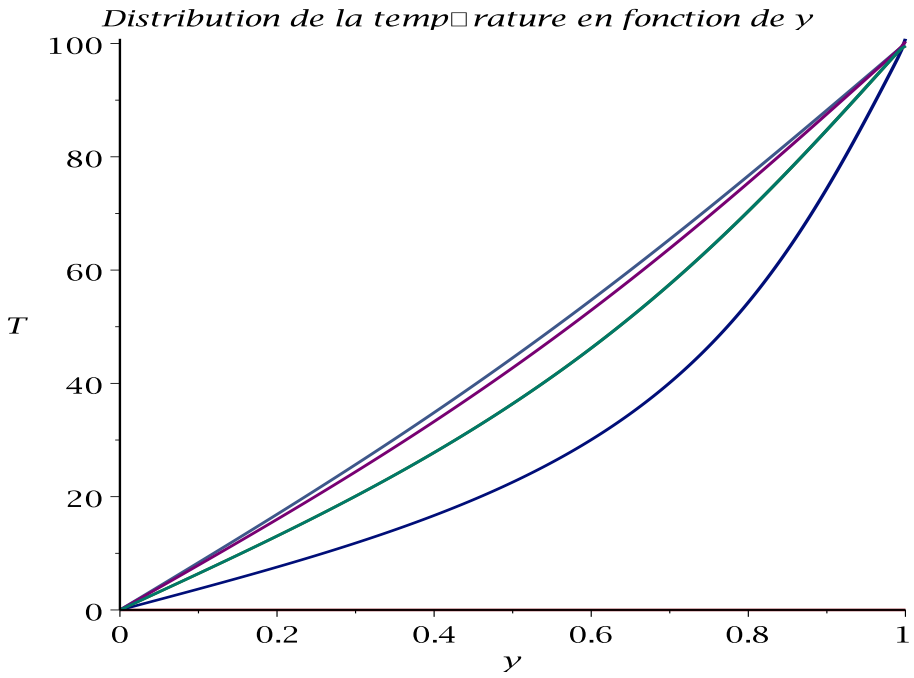
```
> a := 2; b := 1; T0 := 100;
```

$$a := 2$$

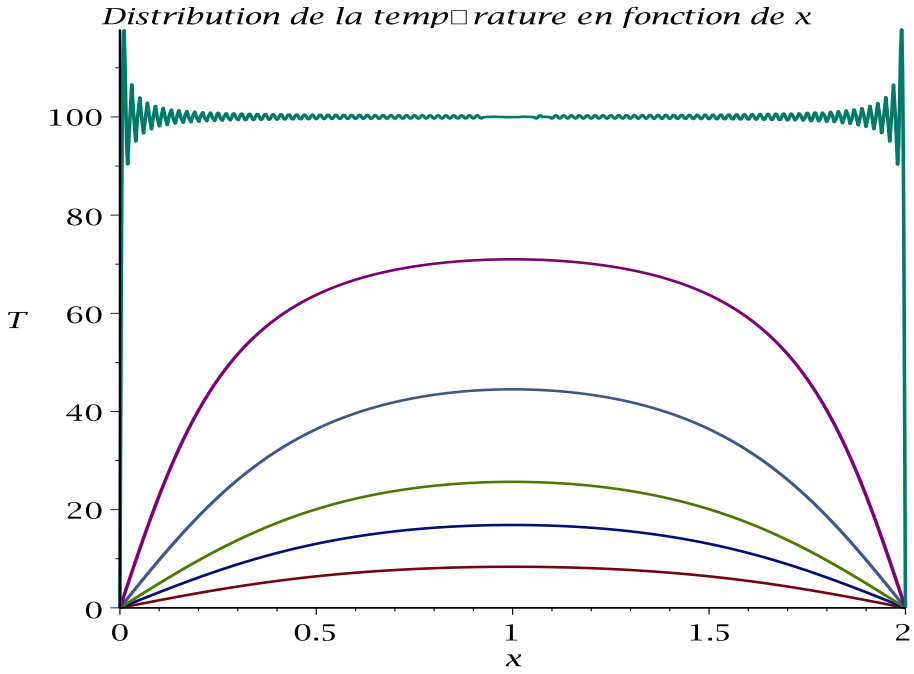
$$b := 1$$

$$T0 := 100$$

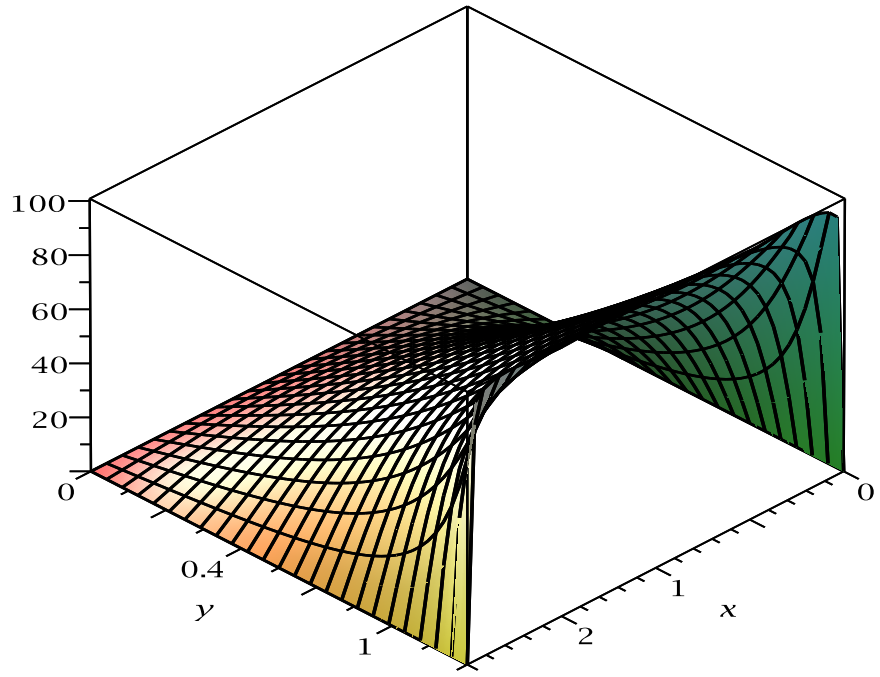
```
> plot(T(2,y), T(.25,y), T(.5,y), T(1.0,y), T(1.25,y), T(1.5,y),
y = 0 .. 1, title = ‘Distribution de la température en fonction de y‘,
labels = [y,T]);
```




```
> plot(T(x,.1), T(x,.2), T(x,.3), T(x,.5), T(x,.75), T(x,1.0),
x = 0 .. 2, title = ‘Distribution de la température en fonction de x‘,
labels = [x,T]);
```




```
> plot3d(T(x,y), x = 0 .. 2, y = 0 .. 1, axes = boxed);
```




### 3.1.2 Programme Lap2D-01

Détermination de la forme matricielle ainsi que la température $T(x, y)$ à travers la surface d'une plaque rectangulaire ($a \times b$ ) dont les extrémités sont soumises à des (C.L) de Dirichlet (Exemple 1, partie Cours). Dans cet exemple on utilisera le schéma à 5 points.

```
> restart;
> with(LinearAlgebra):
> Digits := 3:
> i_max := 5; j_max := 5;
```

$$i_{max} := 5$$

$$j_{max} := 5$$

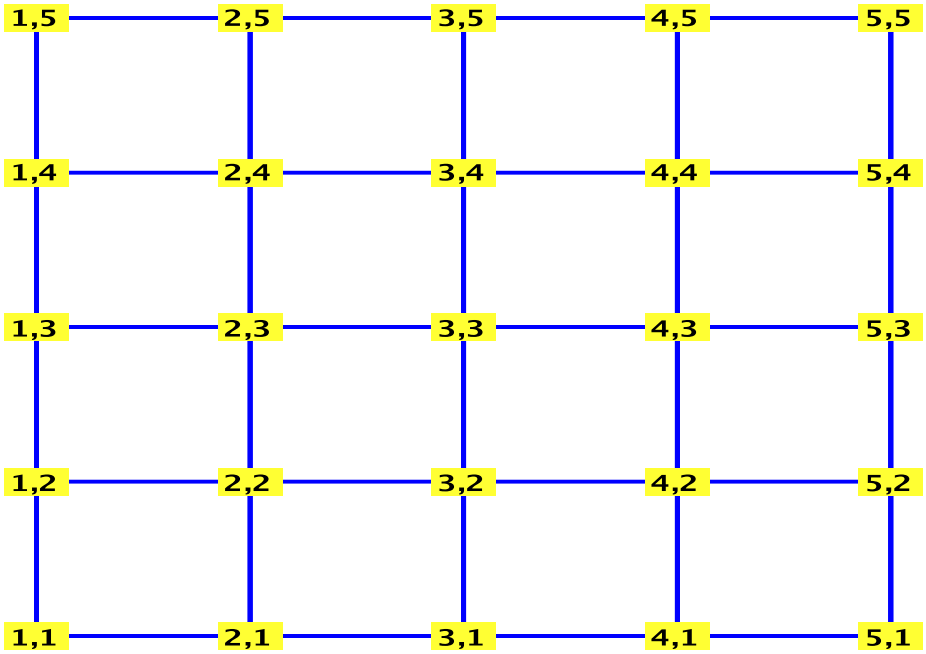
Nombre d'équations :

```
> N := (i_max-2)*(j_max-2)
```

$$N := 9$$

Maillage :

```
> with(GraphTheory):
> with(SpecialGraphs):
> G := GridGraph(i_max, j_max):
> DrawGraph(G);
```




Conditions aux Limites :

```
> for i from 2 to i_max-1 do T[i,1] := 0. end do;
```

$$T_{2,1} := 0.0$$

$$T_{3,1} := 0.0$$

$$T_{4,1} := 0.0$$

```
> for i from 2 to i_max-1 do T[i,j_max] := 100 end do;
```

$$T_{2,5} := 100$$

$$T_{3,5} := 100$$

$$T_{4,5} := 100$$

```
> for j from 2 to j_max-1 do T[1,j] := 75 end do;
```

$$T_{1,2} := 75$$

$$T_{1,3} := 75$$

$$T_{1,4} := 75$$

```
> for j from 2 to j_max-1 do T[i_max,j] := 50 end do;
```

$$T_{5,2} := 50$$

$$T_{5,3} := 50$$

$$T_{5,4} := 50$$

```
> k := 1:
```

Résolution pour les noeuds internes :

```
> for j from 2 to j_max-1 do
for i from 2 to i_max-1 do
Eq[k] := -4*T[i,j]+T[i+1,j]+T[i-1,j]+T[i,j+1]+T[i,j-1] = 0;
Temps[k] := T[i,j];
```

```
k := k+1
end do
end do;
```

Ecriture du système d'équations :

```
> for k from 1 to N do Eq[k] end do;
```

$$-4\,T_{2,2} + T_{3,2} + 75.0 + T_{2,3} = 0$$
$$-4\,T_{3,2} + T_{4,2} + T_{2,2} + T_{3,3} = 0$$
$$-4\,T_{4,2} + 50.0 + T_{3,2} + T_{4,3} = 0$$
$$-4\,T_{2,3} + T_{3,3} + 75 + T_{2,4} + T_{2,2} = 0$$
$$-4\,T_{3,3} + T_{4,3} + T_{2,3} + T_{3,4} + T_{3,2} = 0$$
$$-4\,T_{4,3} + 50 + T_{3,3} + T_{4,4} + T_{4,2} = 0$$
$$-4\,T_{2,4} + T_{3,4} + 175 + T_{2,3} = 0$$
$$-4\,T_{3,4} + T_{4,4} + T_{2,4} + 100 + T_{3,3} = 0$$
$$-4\,T_{4,4} + 150 + T_{3,4} + T_{4,3} = 0$$

Equations et variables :

```
> Eqs := seq(Eq[i], i = 1 .. N);
> Tmps := [seq(Temps[i], i = 1 .. N)];
```

$$Tmps := [T_{2,2}, T_{3,2}, T_{4,2}, T_{2,3}, T_{3,3}, T_{4,3}, T_{2,4}, T_{3,4}, T_{4,4}]$$

Solution du système :

```
> solve(Eqs, Tmps);
```

$$[[T_{2,2} = 42.9, T_{3,2} = 33.3, T_{4,2} = 33.9, T_{2,3} = 63.2, T_{3,3} = 56.2, T_{4,3} = 52.5, T_{2,4} = 78.6, T_{3,4} = 76.1, T_{4,4} = 69.6]]$$

Forme matricielle :

```
> Eqs := [seq(Eq[i], i = 1 .. N)];
> A, B := GenerateMatrix(Eqs,Tmps);
```

$$A,\, B := \begin{bmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{bmatrix}, \begin{bmatrix} -75.0 \\ 0 \\ -50.0 \\ -75 \\ 0 \\ -50 \\ -175 \\ -100 \\ -150 \end{bmatrix}$$

### 3.1.3 Programme Lap2D-02

Le même exemple que celui d'avant mais en utilisant le schéma à 9 points. Il suffit de modifier la boucle principale et de rajouter dans les (C.L) les valeurs des températures

aux quatre coins de la plaque en prenant la moyenne des deux cotés adjacents.

```
> restart;
> with(LinearAlgebra): with(GraphTheory): with(SpecialGraphs):
> Digits := 3:
> i_max := 5: j_max := 5:
> N := (i_max-2)*(j_max-2):

> G := GridGraph(i_max, j_max):

> DrawGraph(G):
> for i from 2 to i_max-1 do T[i,1]:=0. end do; T[1,1]:=0.5*75;
```

$$T_{2,1} := 0.0$$

$$T_{3,1} := 0.0$$

$$T_{4,1} := 0.0$$

$$T_{1,1} := 37.5$$

```
> for i from 2 to i_max-1 do T[i,j_max]:=100 end do; T[5,1]:=0.5*50;
```

$$T_{2,5} := 100$$

$$T_{3,5} := 100$$

$$T_{4,5} := 100$$

$$T_{5,1} := 25.0$$

```
> for j from 2 to j_max-1 do T[1,j]:=75 end do; T[1,5]:=0.5*(75+100);
```

$$T_{1,2} := 75$$

$$T_{1,3} := 75$$

$$T_{1,4} := 75$$

$$T_{1,5} := 87.5$$

```
> for j from 2 to j_max-1 do T[i_max,j]:=50 end do; T[5,5]:=0.5*(50+100);
```

$$T_{5,2} := 50$$

$$T_{5,3} := 50$$

$$T_{5,4} := 50$$

$$T_{5,5} := 75.0$$

```
> k := 1:
> for j from 2 to j_max-1 do
for i from 2 to i_max-1 do
Eq[k] := -20*T[i,j]+T[i-1,j-1]+T[i+1,j-1]+T[i-1,j+1]+T[i+1,j+1] +
 4*(T[i-1,j]+T[i+1,j]+T[i,j-1]+T[i,j+1])=0;
Temps[k] := T[i,j];
k := k+1
end do
end do;
> for k from 1 to N do Eq[k] end do;
```

$$-20\,T_{2,2} + 412.5 + T_{3,3} + 4\,T_{3,2} + 4\,T_{2,3} = 0$$

$$-20\,T_{3,2} + T_{2,3} + T_{4,3} + 4\,T_{2,2} + 4\,T_{4,2} + 4\,T_{3,3} = 0$$

$$-20\,T_{4,2} + 275.0 + T_{3,3} + 4\,T_{3,2} + 4\,T_{4,3} = 0$$

$$-20\,T_{2,3} + 450 + T_{3,2} + T_{3,4} + 4\,T_{3,3} + 4\,T_{2,2} + 4\,T_{2,4} = 0$$
$$-20\,T_{3,3} + T_{2,2} + T_{4,2} + T_{2,4} + T_{4,4} + 4\,T_{2,3} + 4\,T_{4,3} + 4\,T_{3,2} + 4\,T_{3,4} = 0$$
$$-20\,T_{4,3} + T_{3,2} + 300 + T_{3,4} + 4\,T_{3,3} + 4\,T_{4,2} + 4\,T_{4,4} = 0$$
$$-20\,T_{2,4} + 962.0 + T_{3,3} + 4\,T_{3,4} + 4\,T_{2,3} = 0$$
$$-20\,T_{3,4} + T_{2,3} + T_{4,3} + 600 + 4\,T_{2,4} + 4\,T_{4,4} + 4\,T_{3,3} = 0$$
$$-20\,T_{4,4} + T_{3,3} + 825.0 + 4\,T_{3,4} + 4\,T_{4,3} = 0$$

```
> Eqs := [seq(Eq[i], i = 1 .. N)]:
> Tmps := [seq(Temps[i], i = 1 .. N)]:
> solve(Eqs, Tmps);
```

$$[[T_{2,2} = 42.6, T_{3,2} = 32.3, T_{4,2} = 33.5, T_{2,3} = 63.5, T_{3,3} = 56.2, T_{4,3} = 52.4, T_{2,4} = 79.0, T_{3,4} = 76.8, T_{4,4} = 69.9]]$$

Forme matricielle :

```
> A, B := GenerateMatrix(Eqs, Tmps);
```

$$A,\, B := \begin{bmatrix} -20 & 4 & 0 & 4 & 1 & 0 & 0 & 0 & 0 \\ 4 & -20 & 4 & 1 & 4 & 1 & 0 & 0 & 0 \\ 0 & 4 & -20 & 0 & 1 & 4 & 0 & 0 & 0 \\ 4 & 1 & 0 & -20 & 4 & 0 & 4 & 1 & 0 \\ 1 & 4 & 1 & 4 & -20 & 4 & 1 & 4 & 1 \\ 0 & 1 & 4 & 0 & 4 & -20 & 0 & 1 & 4 \\ 0 & 0 & 0 & 4 & 1 & 0 & -20 & 4 & 0 \\ 0 & 0 & 0 & 1 & 4 & 1 & 4 & -20 & 4 \\ 0 & 0 & 0 & 0 & 1 & 4 & 0 & 4 & -20 \end{bmatrix}, \begin{bmatrix} -412.5 \\ 0 \\ -275.0 \\ -450 \\ 0 \\ -300 \\ -962.0 \\ -600 \\ -825.0 \end{bmatrix}$$

### 3.1.4 Programme Lap2D-03

Le même exemple que le premier mais en utilisant une condition limite en haut de type Dirichlet mais variable ($T(x) = 2.10^4.x^2$). Ce programme est écrit d'une manière plus générale en fonction des différentes variables.(Exemple 4, partie Cours).

```
> restart;
> with(LinearAlgebra):
> Digits := 3:
> a := 12; b := 12; ndx := 4; ndy := 4;
```

$$a := 12$$
$$b := 12$$
$$ndx := 4$$
$$ndy := 4$$

```
> f := 2*10^4*x^2;
```

$$f := 20000\,x^2$$

```
> Tb := 0; 1; Tg := 75; 1; Td := 50; 1; Th := unapply(f, x)
```

$$Tb := 0$$

$$Tg := 75$$

$$Td := 50$$

$$Th := x \mapsto 20000\,x^2$$

```
> ‘&Delta;x‘ := a/ndx; ‘&Delta;y‘ := b/ndy; beta := ‘&Delta;x‘/‘&Delta;y‘;
```

$$\Delta x := 3$$

$$\Delta y := 3$$

$$\beta := 1$$

```
> i_max := ndx+1; j_max := ndy+1;
```

$$i_{max} := 5$$

$$j_{max} := 5$$

```
> N := (i_max-2)*(j_max-2);
```

$$N := 9$$

```
> for i from 2 to i_max-1 do T[i,1] := Tb end do;
```

$$T_{2,1} := 0$$

$$T_{3,1} := 0$$

$$T_{4,1} := 0$$

```
> for j from 2 to j_max-1 do T[1,j] := Tg end do;
```

$$T_{1,2} := 75$$

$$T_{1,3} := 75$$

$$T_{1,4} := 75$$

```
> for j from 2 to j_max-1 do T[i_max,j] := Td end do;
```

$$T_{5,2} := 50$$

$$T_{5,3} := 50$$

$$T_{5,4} := 50$$

```
> for i from 2 to i_max-1 do T[i,j_max] := Th(10^(-2)*(i-1)*‘&Delta;x‘)
end do;
```

$$T_{2,5} := 18$$

$$T_{3,5} := 72$$

$$T_{4,5} := 162$$

```
> k := 1:
```

```
> for j from 2 to j_max-1 do
for i from 2 to i_max-1 do
Eq[k]:=-(2*(beta^2+1))*T[i,j]+T[i+1,j]+T[i-1,j]+beta^2*(T[i,j+1]+T[i,j-1])=0;
Temps[k] := T[i,j];
k := k+1
end do
end do;
```

```
> for k from 1 to N do Eq[k] end do;
```

$$-4\,T_{2,2}+T_{3,2}+75+T_{2,3}=0$$
$$-4\,T_{3,2}+T_{4,2}+T_{2,2}+T_{3,3}=0$$
$$-4\,T_{4,2}+50+T_{3,2}+T_{4,3}=0$$
$$-4\,T_{2,3}+T_{3,3}+75+T_{2,4}+T_{2,2}=0$$
$$-4\,T_{3,3}+T_{4,3}+T_{2,3}+T_{3,4}+T_{3,2}=0$$
$$-4\,T_{4,3}+50+T_{3,3}+T_{4,4}+T_{4,2}=0$$
$$-4\,T_{2,4}+T_{3,4}+93+T_{2,3}=0$$
$$-4\,T_{3,4}+T_{4,4}+T_{2,4}+72+T_{3,3}=0$$
$$-4\,T_{4,4}+212+T_{3,4}+T_{4,3}=0$$

```
> Eqs := seq(Eq[i], i = 1 .. N);
```

```
> Tmps := [seq(Temps[i], i = 1 .. N)];
```

```
> evalf(solve(Eqs, Tmps));
```

$$[[T_{2,2}=40.4, T_{3,2}=31.5, T_{4,2}=34.0, T_{2,3}=55.0, T_{3,3}=51.5, T_{4,3}=54.6, T_{2,4}=53.2, T_{3,4}=64.9, T_{4,4}=82.9]]$$

```
> Sys := [seq(Eq[i], i = 1 .. N)];
```

```
> Var := [seq(Temps[i], i = 1 .. N)];
```

```
> A, B := GenerateMatrix(Sys, Var);
```

$$A,\,B := \begin{bmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{bmatrix}, \begin{bmatrix} -75 \\ 0 \\ -50 \\ -75 \\ 0 \\ -50 \\ -93 \\ -72 \\ -212 \end{bmatrix}$$

### 3.1.5 Programme Lap2D-04

Condition Limite gauche de Neumann discrétisée par un schéma centré.

```
> restart;
```

```
> with(LinearAlgebra): with(GraphTheory): with(SpecialGraphs):
```

```
> Digits := 3:
```

```
> L := 20; H := 20; ndx := 3; ndy := 3;
```

$$L := 20$$
$$H := 20$$
$$ndx := 3$$
$$ndy := 3$$

```
> Td := 10; Tb := 10; Th := 30; alpha := 0
```

$$Td := 10$$
$$Tb := 10$$
$$Th := 30$$
$$\alpha := 0$$

```
> ‘&Delta;x‘ := L/ndx; ‘&Delta;y‘ := H/ndy; beta := ‘&Delta;x‘/‘&Delta;y‘
```

$$\Delta x := \frac{20}{3}$$
$$\Delta y := \frac{20}{3}$$
$$\beta := 1$$

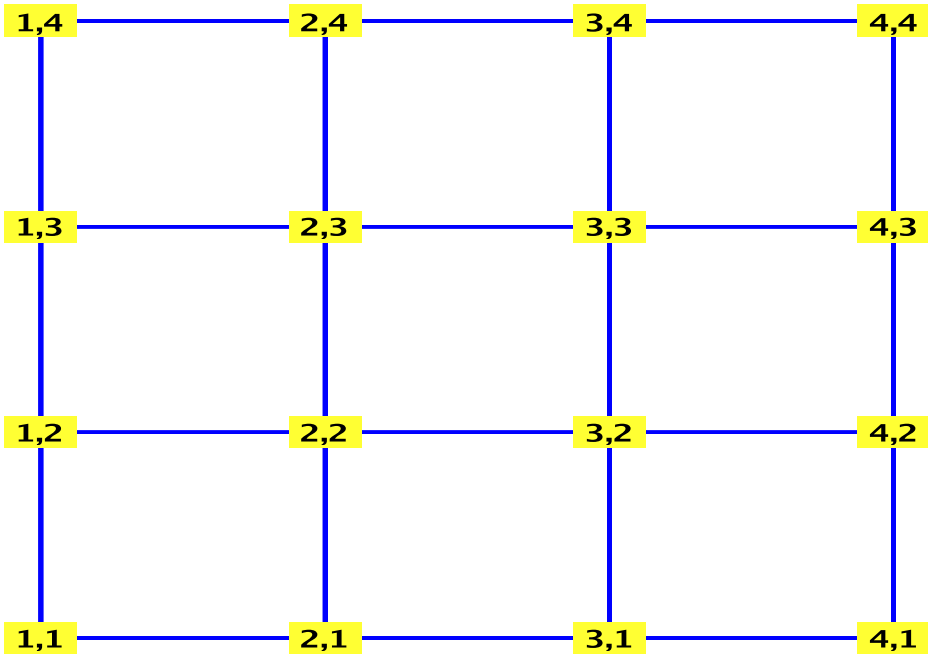
```
> i_max := ndx+1; j_max := ndy+1;
```

$$i_{max} := 4$$
$$j_{max} := 4$$

```
> N := (i_max-2)*(j_max-2) + j_max-2;
```

$$N := 6$$

```
> G := GridGraph(i_max, j_max): DrawGraph(G);
```




```
> for j from 2 to j_max-1 do T[i_max,j] := Td end do;
```

$$T_{4,2} := 10$$

$$T_{4,3} := 10$$

```
> for i from 1 to i_max-1 do T[i,1] := Tb end do;
```

$$T_{1,1} := 10$$

$$T_{2,1} := 10$$

$$T_{3,1} := 10$$

```
> for i from 1 to i_max-1 do T[i,j_max] := Th end do;
```

$$T_{1,4} := 30$$

$$T_{2,4} := 30$$

$$T_{3,4} := 30$$

```
> k := 1:
> for j from 2 to j_max-1 do
T[0,j] := T[2,j] - 2*alpha*‘&Delta;x‘;
Eq[k]:=-(2*(beta^2+1))*T[1,j]+T[2,j]+T[0,j]+beta^2*(T[1,j+1]+T[1,j-1])=0;
Temps[k] := T[1,j];
k := k+1:
for i from 2 to i_max-1 do
Eq[k]:=-(2*(beta^2+1))*T[i,j]+T[i+1,j]+T[i-1,j]+beta^2*(T[i,j+1]+T[i,j-1])=0;
Temps[k] := T[i,j];
k := k+1
end do
```

```
end do;
```

```
> for k from 1 to N do Eq[k] end do;
```

$$-4\,T_{1,2} + 2\,T_{2,2} + T_{1,3} + 10 = 0$$
$$-4\,T_{2,2} + T_{3,2} + T_{1,2} + T_{2,3} + 10 = 0$$
$$-4\,T_{3,2} + 20 + T_{2,2} + T_{3,3} = 0$$
$$-4\,T_{1,3} + 2\,T_{2,3} + 30 + T_{1,2} = 0$$
$$-4\,T_{2,3} + T_{3,3} + T_{1,3} + 30 + T_{2,2} = 0$$
$$-4\,T_{3,3} + 40 + T_{2,3} + T_{3,2} = 0$$

```
> Eqs := seq(Eq[k], k = 1 .. N):
> Tmps := [seq(Temps[k], k = 1 .. N)]:
> evalf(solve(Eqs, Tmps));
```

$$[[T_{1,2} = 15.6, T_{2,2} = 15.2, T_{3,2} = 13.5, T_{1,3} = 22.2, T_{2,3} = 21.5, T_{3,3} = 18.7]]$$

```
> Eqs := [seq(Eq[k], k = 1 .. N)]:
> A, B := GenerateMatrix(Eqs, Tmps);
```

$$A,\, B := \begin{bmatrix} -4 & 2 & 0 & 1 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 \\ 1 & 0 & 0 & -4 & 2 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 1 & 0 & 1 & -4 \end{bmatrix}, \begin{bmatrix} -10 \\ -10 \\ -20 \\ -30 \\ -30 \\ -40 \end{bmatrix}$$

### 3.1.6 Programme Lap2D-05

Le programme précédent peut être nettement amélioré en modifiant la boucle principale comme suit :

```
> for j from 2 to j_max-1 do
T[0,j] := T[2,j] - 2*alpha*‘&Delta;x‘;
for i from 1 to i_max-1 do
Eq[k]:=-(2*(beta^2+1))*T[i,j]+T[i+1,j]+T[i-1,j]+beta^2*(T[i,j+1]+T[i,j-1])=0;
Temps[k] := T[i,j];
k := k+1
end do
end do;
```

Les deux équations supplémentaires sont prises en compte en commençant la boucle $i$ par 1 au lieu de deux. Les incinnues sont remplacées par l'équation discrétisée de la (CLN) dans la boucle $j$.

### 3.1.7 Programme Lap2D-06

Le problème suivant traite le cas de deux (CLN) à droite $\alpha_d$ et à gauche $\alpha_g$. Dans ce cas la boucle $j$ reste inchangée et on modifie la boucle $i$ de 1 à $i_{max}$.

```
> restart;
> with(LinearAlgebra): with(GraphTheory): with(SpecialGraphs):
```

```
> L := 6; H := 6; ndx := 3; ndy := 3;
```

$$L := 6$$
$$H := 6$$
$$ndx := 3$$
$$ndy := 3$$

```
> Tb := 100; Th := 40; alpha[g] := 10; alpha[d] := 20;
```

$$Tb := 100$$
$$Th := 40$$
$$\alpha_g := 10$$
$$\alpha_d := 20$$

```
> ‘&Delta;x‘ := L/ndx; ‘&Delta;y‘ := H/ndy; beta := ‘&Delta;x‘/‘&Delta;y‘
```

$$\Delta x := 2$$
$$\Delta y := 2$$
$$\beta := 1$$

```
> i_max := ndx+1; j_max := ndy+1;
```

$$i_{max} := 4$$
$$j_{max} := 4$$

```
> N := (i_max-2)*(j_max-2) + 2*(j_max-2);
```

$$N := 8$$

```
> G := GridGraph(i_max, j_max): DrawGraph(G):
```

```
> for i from 1 to i_max do T[i,1] := Tb end do;
```

$$T_{1,1} := 100$$
$$T_{2,1} := 100$$
$$T_{3,1} := 100$$
$$T_{4,1} := 100$$

```
> for i from 1 to i_max do T[i,j_max] := Th end do;
```

$$T_{1,4} := 40$$
$$T_{2,4} := 40$$
$$T_{3,4} := 40$$
$$T_{4,4} := 40$$

```
> k := 1:
```

```
> for j from 2 to j_max-1 do
T[0,j] := T[2,j] - 2*‘&Delta;x‘*alpha[g];
for i from 1 to i_max do
Eq[k]:=-(2*(beta^2+1))*T[i,j]+T[i+1,j]+T[i-1,j]+beta^2*(T[i,j+1]+T[i,j-1])=0;
Temps[k] := T[i,j];
k := k+1
end do;
T[i_max+1,j] := T[i_max-1,j] + 2*‘&Delta;x‘*alpha[d]
end do;
```

```
> for k from 1 to N do Eq[k] end do;
```

$$
\begin{aligned}
-4\,T_{1,2} + 2\,T_{2,2} + 60 + T_{1,3} &= 0\\
-4\,T_{2,2} + T_{3,2} + T_{1,2} + T_{2,3} + 100 &= 0\\
-4\,T_{3,2} + T_{4,2} + T_{2,2} + T_{3,3} + 100 &= 0\\
-4\,T_{4,2} + 2\,T_{3,2} + 180 + T_{4,3} &= 0\\
-4\,T_{1,3} + 2\,T_{2,3} + T_{1,2} &= 0\\
-4\,T_{2,3} + T_{3,3} + T_{1,3} + 40 + T_{2,2} &= 0\\
-4\,T_{3,3} + T_{4,3} + T_{2,3} + 40 + T_{3,2} &= 0\\
-4\,T_{4,3} + 2\,T_{3,3} + 120 + T_{4,2} &= 0
\end{aligned}
$$

```
> Eqs := seq(Eq[k], k = 1 .. N):
```

```
> Tmps := [seq(Temps[k], k = 1 .. N)]:
```

```
> solve(Eqs, Tmps);
```

$$[[T_{1,2} = 66, T_{2,2} = 79, T_{3,2} = 91, T_{4,2} = 114, T_{1,3} = 46, T_{2,3} = 59, T_{3,3} = 71, T_{4,3} = 94]]$$

```
> Eqs := [seq(Eq[k], k = 1 .. N)]:
```

```
> A, B := GenerateMatrix(Eqs, Tmps);
```

$$
A,\,B := \begin{bmatrix}
-4 & 2 & 0 & 0 & 1 & 0 & 0 & 0\\
1 & -4 & 1 & 0 & 0 & 1 & 0 & 0\\
0 & 1 & -4 & 1 & 0 & 0 & 1 & 0\\
0 & 0 & 2 & -4 & 0 & 0 & 0 & 1\\
1 & 0 & 0 & 0 & -4 & 2 & 0 & 0\\
0 & 1 & 0 & 0 & 1 & -4 & 1 & 0\\
0 & 0 & 1 & 0 & 0 & 1 & -4 & 1\\
0 & 0 & 0 & 1 & 0 & 0 & 2 & -4
\end{bmatrix},
\begin{bmatrix}
-60\\ -100\\ -100\\ -180\\ 0\\ -40\\ -40\\ -120
\end{bmatrix}
$$

### 3.1.8 Programme Lap2D-07

Le problème suivant traite le cas de deux (CLN) en haut $\alpha_h$ et en bas $\alpha_b$. Dans ce cas la boucle $i$ reste inchangée et on modifie la boucle $j$ de 1 à $j_{max}$.

```
> restart
> with(LinearAlgebra): with(GraphTheory): with(SpecialGraphs):
> Digits := 3:
> L := 20; H := 20; ndx := 3; ndy := 3;
```

$$L := 20$$
$$H := 20$$
$$ndx := 3$$
$$ndy := 3$$

```
> Tg := 10; Td := 30; alpha[b] := 0; alpha[h] := 0;
```

$$Tg := 10$$
$$Td := 30$$
$$\alpha_b := 0$$
$$\alpha_h := 0$$

```
> ‘&Delta;x‘ := L/ndx; ‘&Delta;y‘ := H/ndy; beta := ‘&Delta;x‘/‘&Delta;y‘
```

$$\Delta x := \frac{20}{3}$$
$$\Delta y := \frac{20}{3}$$
$$\beta := 1$$

```
> i_max := ndx+1; j_max := ndy+1
```

$$i_{max} := 4$$
$$j_{max} := 4$$

```
> N := (i_max-2)*(j_max-2) + 2*(i_max-2);
```

$$N := 8$$

```
> G := GridGraph(i_max, j_max): DrawGraph(G):
> for j from 1 to j_max do T[1,j] := Tg end do;
```

$$T_{1,1} := 10$$
$$T_{1,2} := 10$$
$$T_{1,3} := 10$$
$$T_{1,4} := 10$$

```
> for j from 1 to j_max do T[i_max,j] := Td end do;
```

$$T_{4,1} := 30$$
$$T_{4,2} := 30$$
$$T_{4,3} := 30$$
$$T_{4,4} := 30$$

```
> k := 1:
> for i from 2 to i_max-1 do
T[i,0] := T[i,2] - 2*‘&Delta;y‘*alpha[b]
end do;
for j from 1 to j_max do
for i from 2 to i_max-1 do
Eq[k]:=-(2*(beta^2+1))*T[i,j]+T[i+1,j]+T[i-1,j]+beta^2*(T[i,j+1]+T[i,j-1])=0;
Temps[k] := T[i,j];
k := k+1
end do
end do;
for i from 2 to i_max-1 do
T[i,j_max+1] := T[i,j_max-1] + 2*‘&Delta;y‘*alpha[h]
end do;
```

```
> for k from 1 to N do Eq[k] end do;
```

$$
\begin{aligned}
-4\,T_{2,1} + T_{3,1} + 10 + 2\,T_{2,2} &= 0\\
-4\,T_{3,1} + 30 + T_{2,1} + 2\,T_{3,2} &= 0\\
-4\,T_{2,2} + T_{3,2} + 10 + T_{2,3} + T_{2,1} &= 0\\
-4\,T_{3,2} + 30 + T_{2,2} + T_{3,3} + T_{3,1} &= 0\\
-4\,T_{2,3} + T_{3,3} + 10 + T_{2,4} + T_{2,2} &= 0\\
-4\,T_{3,3} + 30 + T_{2,3} + T_{3,4} + T_{3,2} &= 0\\
-4\,T_{2,4} + T_{3,4} + 10 + 2\,T_{2,3} &= 0\\
-4\,T_{3,4} + 30 + T_{2,4} + 2\,T_{3,3} &= 0
\end{aligned}
$$

```
> Eqs := seq(Eq[k], k = 1 .. N):
> Tmps := [seq(Temps[k], k = 1 .. N)]:
> evalf(solve(Eqs, Tmps));
```

$$[[T_{2,1} = 16.7, T_{3,1} = 23.3, T_{2,2} = 16.7, T_{3,2} = 23.3, T_{2,3} = 16.7, T_{3,3} = 23.3, T_{2,4} = 16.7, T_{3,4} = 23.3]]$$

```
> Eqs := [seq(Eq[k], k = 1 .. N)]:
> A, B := GenerateMatrix(Eqs, Tmps);
```

$$
A,\,B := \begin{bmatrix}
-4 & 1 & 2 & 0 & 0 & 0 & 0 & 0\\
1 & -4 & 0 & 2 & 0 & 0 & 0 & 0\\
1 & 0 & -4 & 1 & 1 & 0 & 0 & 0\\
0 & 1 & 1 & -4 & 0 & 1 & 0 & 0\\
0 & 0 & 1 & 0 & -4 & 1 & 1 & 0\\
0 & 0 & 0 & 1 & 1 & -4 & 0 & 1\\
0 & 0 & 0 & 0 & 2 & 0 & -4 & 1\\
0 & 0 & 0 & 0 & 0 & 2 & 1 & -4
\end{bmatrix},
\begin{bmatrix}
-10\\ -30\\ -10\\ -30\\ -10\\ -30\\ -10\\ -30
\end{bmatrix}
$$

### 3.1.9 Programme Lap2D-08

Conditions Limites de Neumann en bas et à gauche discrétisées par un schéma centré

```
> restart
> with(LinearAlgebra): with(GraphTheory): with(SpecialGraphs):
> Digits := 3:
> L := 20; H := 20; ndx := 3; ndy := 3;
```

$$L := 20$$
$$H := 20$$
$$ndx := 3$$
$$ndy := 3$$

```
> Td:=30; Th:=10; alpha[g]:=0; alpha[b]:=0; alpha[m]:=0.5*(alpha[g]+alpha[b]);
```

$$Td := 30$$
$$Th := 10$$
$$\alpha_g := 0$$
$$\alpha_b := 0$$
$$\alpha_m := 0.0$$

```
> ‘&Delta;x‘ := L/ndx; ‘&Delta;y‘ := H/ndy; beta := ‘&Delta;x‘/‘&Delta;y‘;
```

$$\Delta x := \frac{20}{3}$$
$$\Delta y := \frac{20}{3}$$
$$\beta := 1$$

```
> i_max := ndx+1; j_max := ndy+1;
```

$$i_{max} := 4$$
$$j_{max} := 4$$

```
> N := (i_max-2)*(j_max-2) + i_max-2 + (j_max-2)+1;
```

$$N := 9$$

```
> G := GridGraph(i_max, j_max): DrawGraph(G):
> for j from 1 to j_max-1 do T[i_max,j] := Td end do;
```

$$T_{4,1} := 30$$
$$T_{4,2} := 30$$
$$T_{4,3} := 30$$

```
> T[i_max,j_max] := 0.5*(Td+Th);
```

$$T_{4,4} := 20.0$$

```
> for i from 1 to i_max-1 do T[i,j_max] := Th end do;
```

$$T_{1,4} := 10$$
$$T_{2,4} := 10$$
$$T_{3,4} := 10$$

```
> k := 1:

> for i from 1 to i_max-1 do
T[i,0] := T[i,2] - 2*‘&Delta;y‘*alpha[b]
end do;
T[0,1] := T[2, 1] - 2*‘&Delta;x‘*alpha[m];
for j from 1 to j_max-1 do T[0,j] := T[2,j] - 2*‘&Delta;x‘*alpha[g];
for i from 1 to i_max-1 do
Eq[k]:=-(2*(beta^2+1))*T[i,j]+T[i+1,j]+T[i-1,j]+beta^2*(T[i,j+1]+T[i,j-1])=0;
Temps[k] := T[i,j];
k := k+1
end do
end do;

> for k from 1 to N do Eq[k] end do;
```

$$-4\,T_{1,1} + 2\,T_{2,1} + 2\,T_{1,2} = 0$$
$$-4\,T_{2,1} + T_{3,1} + T_{1,1} + 2\,T_{2,2} = 0$$
$$-4\,T_{3,1} + 30 + T_{2,1} + 2\,T_{3,2} = 0$$
$$-4\,T_{1,2} + 2\,T_{2,2} + T_{1,3} + T_{1,1} = 0$$
$$-4\,T_{2,2} + T_{3,2} + T_{1,2} + T_{2,3} + T_{2,1} = 0$$
$$-4\,T_{3,2} + 30 + T_{2,2} + T_{3,3} + T_{3,1} = 0$$
$$-4\,T_{1,3} + 2\,T_{2,3} + 10 + T_{1,2} = 0$$
$$-4\,T_{2,3} + T_{3,3} + T_{1,3} + 10 + T_{2,2} = 0$$
$$-4\,T_{3,3} + 40 + T_{2,3} + T_{3,2} = 0$$

```
> Eqs := seq(Eq[k], k = 1 .. N):

> Tmps := [seq(Temps[k], k = 1 .. N)]:

> evalf(solve(Eqs, Tmps));
```

$$[[T_{1,1} = 20.0, T_{2,1} = 21.2, T_{3,1} = 24.6, T_{1,2} = 18.8, T_{2,2} = 20.0, T_{3,2} = 23.7, T_{1,3} = 15.4, T_{2,3} = 16.3, T_{3,3} = 20.0]]$$

```
> Eqs := [seq(Eq[k], k = 1 .. N)]:

> A, B := GenerateMatrix(Eqs, Tmps);
```

$$A,\, B := \left[\begin{array}{ccccccccc} -4 & 2 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 2 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 2 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 2 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{array}\right], \left[\begin{array}{c} 0 \\ 0 \\ -30 \\ 0 \\ 0 \\ -30 \\ -10 \\ -10 \\ -40 \end{array}\right]$$

### 3.1.10 Programme Lap2D-09

### 3.1.11 Programme Lap2D-10

## 3.2 Equation Tridimensionnelle

### 3.2.1 Programme Lap3D-01

### 3.2.2 Programme Lap3D-02

Chapitre 4

# *Equation d'onde*

## 4.1 Equation unidimensionnelle

Le problème mathématique est défini avec ses conditions aux limites (CL) et initiales (CI) comme suit :

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2} \qquad 0 \leq x \leq L \ , \ t \geq 0 \tag{4.1}$$

$$\begin{cases} u(0,t) = 0 & , \quad u(x,0) = 2\, sin(\frac{\pi}{L}\, x) \\ u(L,t) = 0 & , \quad \frac{\partial u}{\partial t}(x,o) = -sin(\frac{2\pi}{L}\, x) \end{cases}$$

### 4.1.1 Programme Ond1D-An

Ce programme donne la solution analytique de l'équation d'onde 2D.

```
> restart;
```

```
> u(x,t) := 2 sin(π x/L) cos(c π t/L) - L/(2 π c) sin(2 π x/L) sin(2 c π t/L);
```

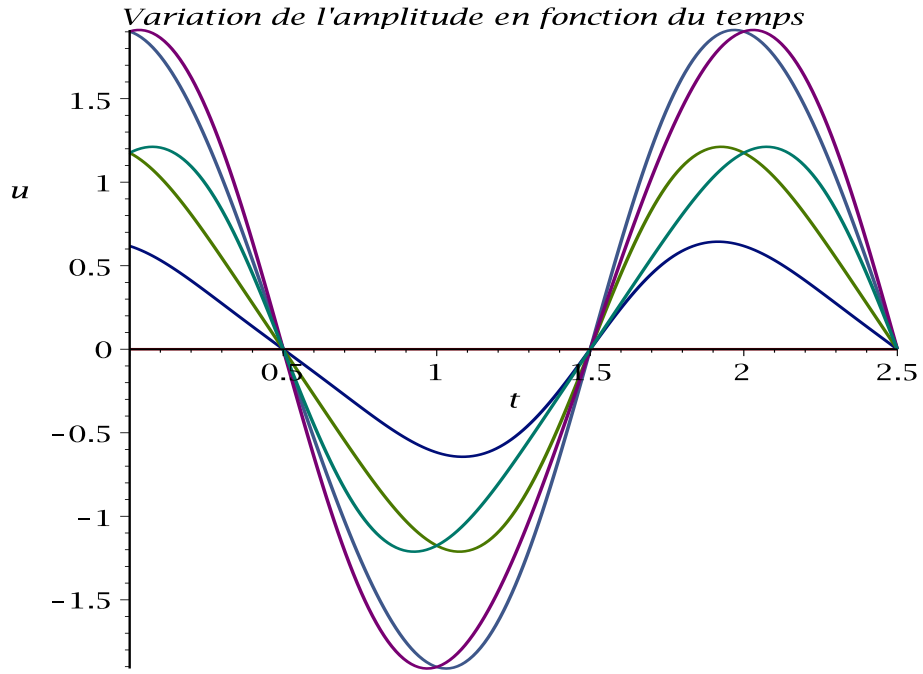
$$u := (x,t) \mapsto 2 \sin\left(\frac{\pi\, x}{L}\right) \cos\left(\frac{c\pi\, t}{L}\right) - 1/2\, \frac{L}{c\pi} \sin\left(2\, \frac{\pi\, x}{L}\right) \sin\left(2\, \frac{c\pi\, t}{L}\right)$$

```
> L := 1; c := 1;
```

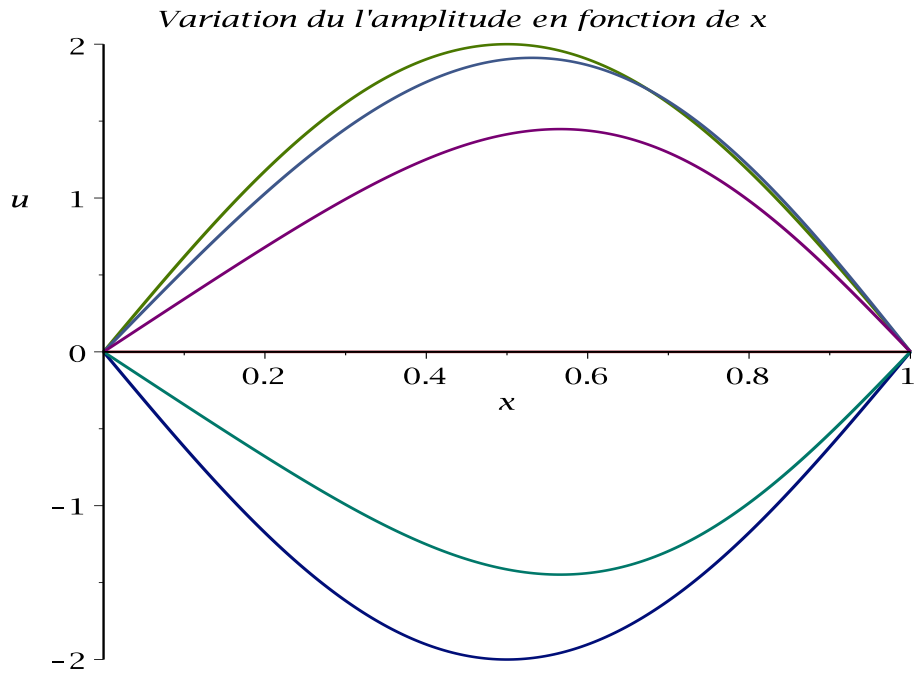
$$L := 1$$

$$c := 1$$

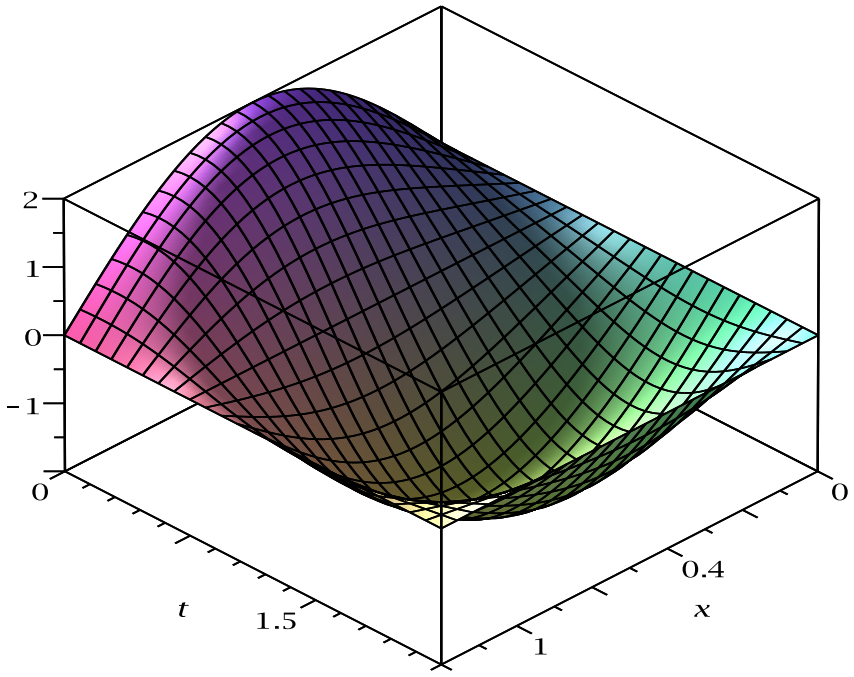
```
> plot(u(1, t), u(.1,t), u(.2,t), u(.4,t), u(.6,t), u(.8,t),
t = 0 .. 2.5, title = 'Variation de l'amplitude en fonction du temps',
labels = [t,u]);
```




```
> plot(u(x,.1), u(x,.25), u(x,.75), u(x,1.0), u(x,1.5), u(x,2.0),
x = 0 .. 1, title = 'Variation du l'amplitude en fonction de x', labels
= [x,u]);
```




```
> plot3d(u(x,t), x = 0 .. 1, t = 0 .. 1.5, axes = boxed);
```




### 4.1.2 Programme Ond1D-01

Le programme Onde1D-Num donne directement la solution numérique du problème défini plus haut.

```
> restart;
```

```
> EDP := diff(u(x,t),t,t) = diff(u(x,t),x,x);
```

$$EDP := \frac{\partial^2}{\partial t^2} u(x,t) = \frac{\partial^2}{\partial x^2} u(x,t)$$

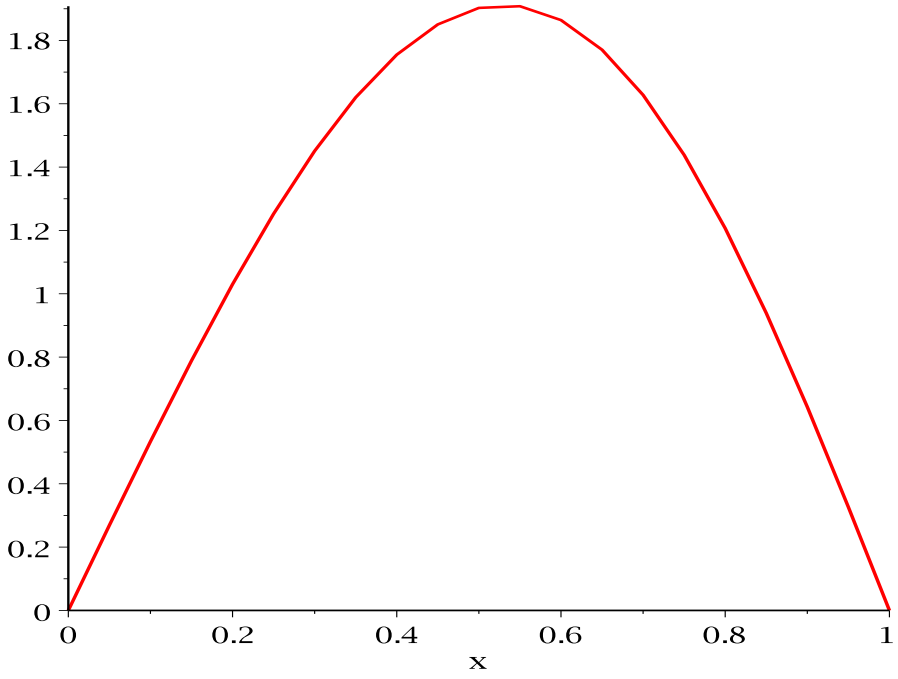
```
> CLI := {u(0,t) = 0, u(1,t) = 0, u(x,0) = 2*sin(Pi*x),
(D[2](u))(x,0) = -sin(2*Pi*x)};
```

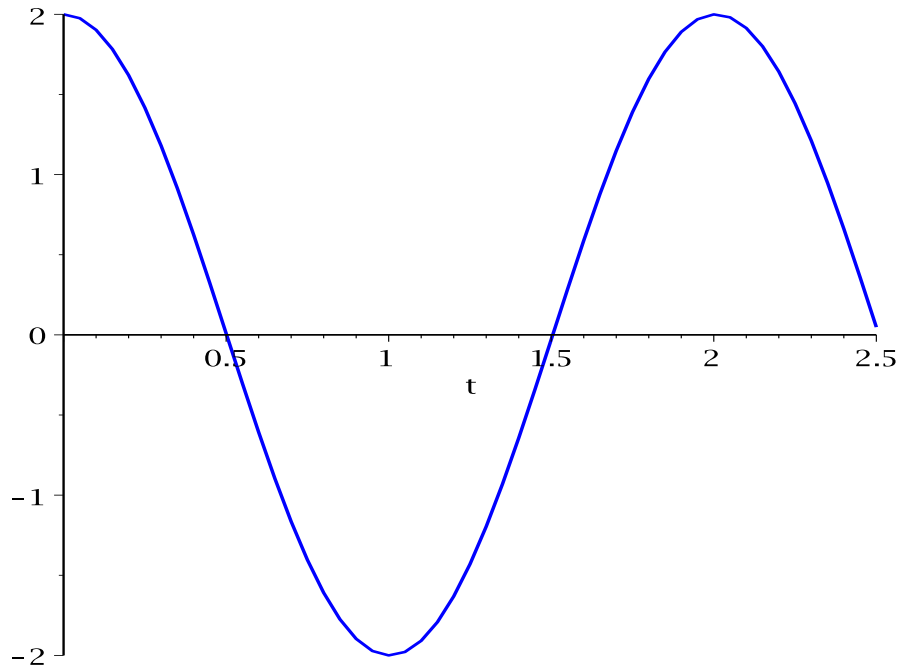
$$CLI := \{u(0,t) = 0, u(1,t) = 0, u(x,0) = 2\sin(\pi x), D_2(u)(x,0) = -\sin(2\pi x)\}$$

```
> Sol := pdsolve(EDP, CLI, numeric, spacestep = 0.5e-1);
```

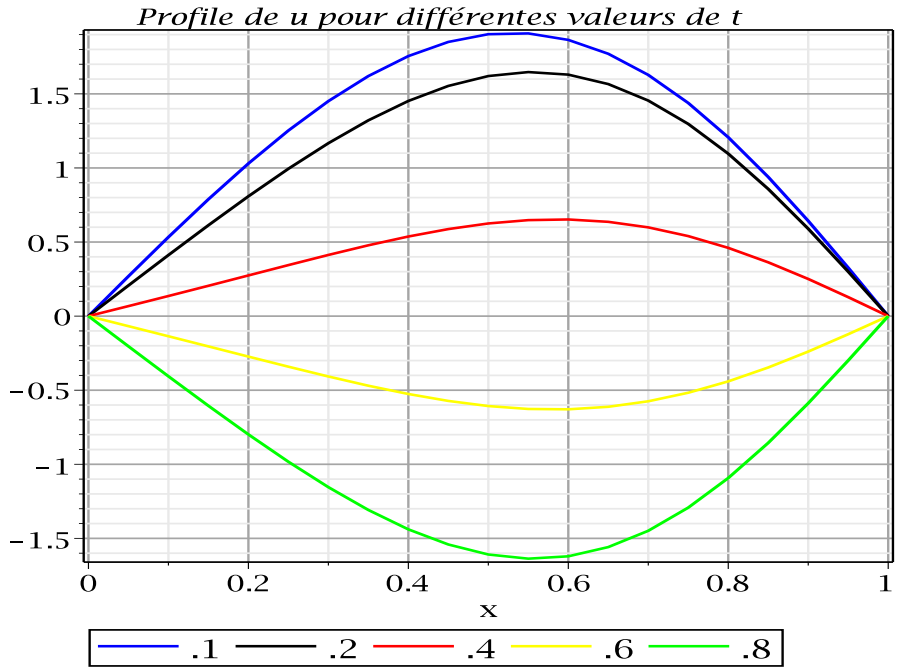
$Sol :=$ module () export *plot*, *plot3d*, *animate*, *value*, *settings*; ... endmodule

```
> Sol:-plot(u(x,t), x = 0 .. 1, t = .1);
Sol:-plot(u(x, t), x = .5, t = 0 .. 2.5, color = blue);
```




```
> g1 := Sol:-plot(t = .1, colour = blue, legend = '.1'):
g2 := Sol:-plot(t = .2, colour = black, legend = '.2'):
g3 := Sol:-plot(t = .4, colour = red, legend = '.4'):
g4 := Sol:-plot(t = .6, colour = yellow, legend = '.6'):
g5 := Sol:-plot(t = .8, colour = green, legend = '.8'):
plots[display](g1, g2, g3, g4, g5, gridlines = true, axes = boxed,
title = `Profile de u pour différentes valeurs de t`);
```



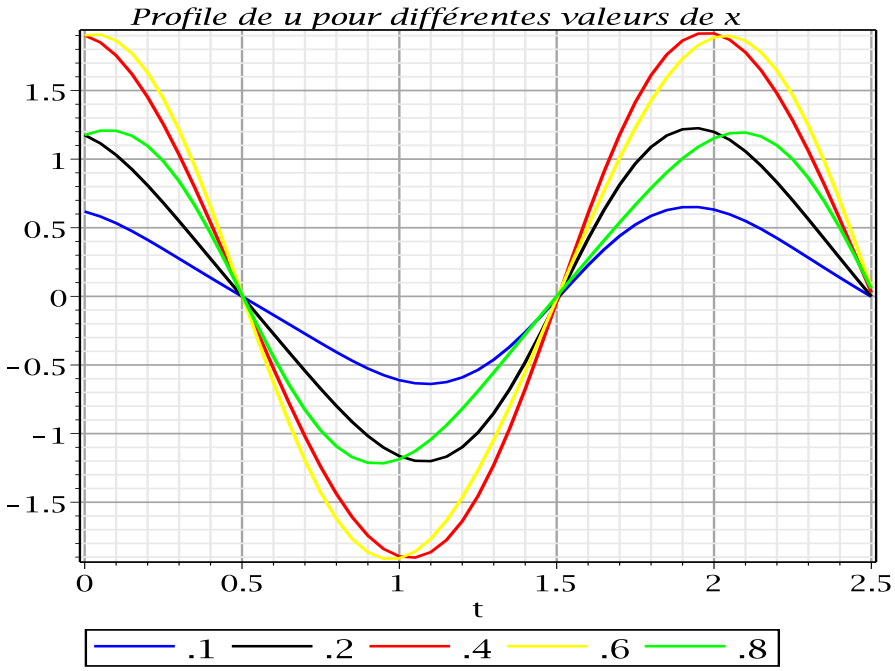

```
> C1 := Sol:-plot(x = .1, t = 0 .. 2.5, colour = blue, legend = '.1'):

C2 := Sol:-plot(x = .2, t = 0 .. 2.5, colour = black, legend = '.2'):

C3 := Sol:-plot(x = .4, t = 0 .. 2.5, colour = red, legend = '.4'):
C4 := Sol:-plot(x = .6, t = 0 .. 2.5, colour = yellow, legend = '.6'):

C5 := Sol:-plot(x = .8, t = 0 .. 2.5, colour = green, legend = '.8'):

plots[display](C1, C2, C3, C4, C5, gridlines = true, axes = boxed,
title = `Profile de u pour différentes valeurs de x`);
```




### 4.1.3 Programme Ond1D-02

## 4.2 Equation Tridimensionnelle

### 4.2.1 Programme Ond2D-01

### 4.2.2 Programme Ond2D-02

Chapitre 5

# *Applications utilisant les Maplets*

Nous pouvons utiliser la programmation avancée de Maple en utilisant les *Maplets*. Cette technique nous permet de créer une interface graphique qui sert de lien entre l'utilisateur et les commandes de *Maple* ou les programmes mis sous forme procédurale.

## 5.1 Equation de la chaleur

Détermination de la température $T(x,t)$ à travers la surface d'une barre de très faible section, de longueur $L$ dont les deux extrémités sont soumises à des (C.L) de type Dirichlet.

Le programme *Diff1D-Map* est basé sur le programme *Diff1D-Num* mais écrit avec les *Maplets* et utilise directement les commandes *Maple* et non pas la programmation standard. Nous pouvons varier la diffusivité thermique du matériau ainsi que le temps d'évaluation afin de voir comment varie la température.

```
> restart: with(plottools): with(plots): with(Maplets[Elements]):
```

Programme écrit sous forme de procédure :

```
> Chaleur := proc (Tg, Td, Ti, k, Ts, vx, vt, Ch)
 local Eq, CL, Sol, g1, g2, g3, g4, g5, p, anim, ValTemp, V;

Eq := diff(T(x,t), t) = k*(diff(diff(T(x,t),x),x));
CL := T(0,t) = Tg, T(1,t) = Td, T(x,0) = Ti;
Sol := pdsolve(Eq, CL, numeric, timestep = 0.1e-1, spacestep = .1);
if Ch = 1 then
g1 := Sol:-plot(t = 0.2e-1, colour = blue);
g2 := Sol:-plot(t = 0.4e-1, colour = black);
g3 := Sol:-plot(t = 0.6e-1, colour = red);
g4 := Sol:-plot(t = 0.8e-1, colour = yellow);
g5 := Sol:-plot(t = .1, colour = green);
plots[display](g1, g2, g3, g4, g5, gridlines = true, axes = boxed,
title = `Profile de T pour t=0.02,0.04,0.06,0.08,0.1,`, labels = [x,T])
elif Ch = 2 then
p := Sol:-animate(T(x,t), t = 0 .. .7, frames = 50,
labels = ["x","T(x,t)"], labelfont = [TIMES, ROMAN, 14]);
anim := Maplet([Plotter[P](p, continuous = false),
[Button("Play", SetOption(P('play') = true)),
Button("Stop", SetOption(P(''stop'') = true)),
Button("pause", SetOption(P('pause') = true))],
["continuous", CheckBox[CONTINUOUS](value = false, onchange = SetOption(target
= P, `option` = 'continuous', Argument(CONTINUOUS)))],
["delay", Slider[DELAY](100 .. 500, 200, 'filled' = true, 'showticks',
'majorticks' = 100, 'minorticks' = 50, onchange = SetOption(target = P,
`option` = 'delay', Argument(DELAY)))],
Button("OK", Shutdown())]);
Maplets:-Display(anim)
elif Ch = 3 then
Sol:-plot3d(T(x, t), t = 0 .. Ts, shading = zhue, axes = boxed,
labels = ["x", "t", "T(x,t)"], labelfont = [TIMES, ROMAN, 16])
elif Ch = 4 then
ValTemp := Sol:-value();
V := ValTemp(vx, vt); `<,>`(V[1], V[2], V[3])
end if
end proc;
```

Déinition des couleurs :

```
> CF1 := "#CCFFFC"; CF2 := "#CCAAAC"; CF3 := "#FFFFCC"; CF4 := "#FACCAF";
CF5 := "#CCAAAC"; CF6 := "#AAFFCC";
```

Interface graphique :

```
> maplette := Maplet(Window('title' = " Equation de la Chaleur 1D ",
width = 1200, height = 750, resizable = 'false',
['background' = CF1, BoxLayout['BL1'](border = true, 'background' = CF2,
```

```
BoxRow([Label("Résolution directe de l'équation de la chaleur 1D par Maple",
'font' = Font("times", bold, 16))]),
[Label("Par: Dr. Laïd MESSAOUDI", 'foreground' = 'blue')]),
BoxLayout['BL2'](border = true, 'background' = CF3,
BoxRow('background' = CF3, BoxColumn('background' = CF3, [border = true,
'caption' = "Données :", 'background' = CF3, ['background' = CF3, ["Température
gauche :", TextField['Tg'](4, 'value' = 0), 'background' = CF3], ["Température
droite  :", TextField['Td'](4, 'value' = 0), 'background' = CF3], ["Condition
initiale       :", TextField['Ti'](4, 'value' = 1), 'background' = CF3]]]),
BoxColumn('background' = CF3, [border = true, 'caption' = "Paramètres à varier
:", 'background' = CF3, ['background' = CF3, ["Diffusivité Thermique :",
TextField['k'](4, 'value' = 1), 'background' = CF3], ["Temps d'évaluation
:", TextField['Ts'](4, 'value' = .1), 'background' = CF3]]]))),
BoxLayout['BL3'](border = true, 'background' = CF5, BoxRow(BoxColumn(border
= false, BoxRow(border = false, [border = true, Label("Tracé des courbes
:", 'font' = Font("times", bold, 14)), 'background' = CF6,
[Button("Courbes instantannées", 'background' = CF5,
Evaluate('PL1' = 'Chaleur(Tg, Td, Ti, k, Ts, vx, vt, 1)')), border = true],

[Button("Animation", 'background' = CF5,
Evaluate('PL1' = 'Chaleur(Tg, Td, Ti, k, Ts, vx, vt, 2)')), border = true],

[Button("Courbe 3D", 'background' = CF5,
Evaluate('PL1' = 'Chaleur(Tg, Td, Ti, k, Ts, vx, vt, 3)')), border = true],

[Button("Valeur de la température", 'background' = CF5,
Evaluate('MV1' = 'Chaleur(Tg, Td, Ti, k, Ts, vx, vt, 4)')), border = true]])),

BoxColumn(["x = ", TextField['vx'](4, 'value' = .1)],
["t = ", TextField['vt'](4, 'value' = 0.1e-1)],
[MathMLViewer['MV1']('background' = CF3, 'width' = 80, 'height' = 40, 'value'
= MathML[Export]("")), 'background' = CF4]),
Plotter['PL1']('background' = CF4, 'width' = 500, 'height' = 400))]]));
```

```
> result := Maplets[Display](maplette);
```

Les figures ci-dessous nous donnent les résultats de ce programme.

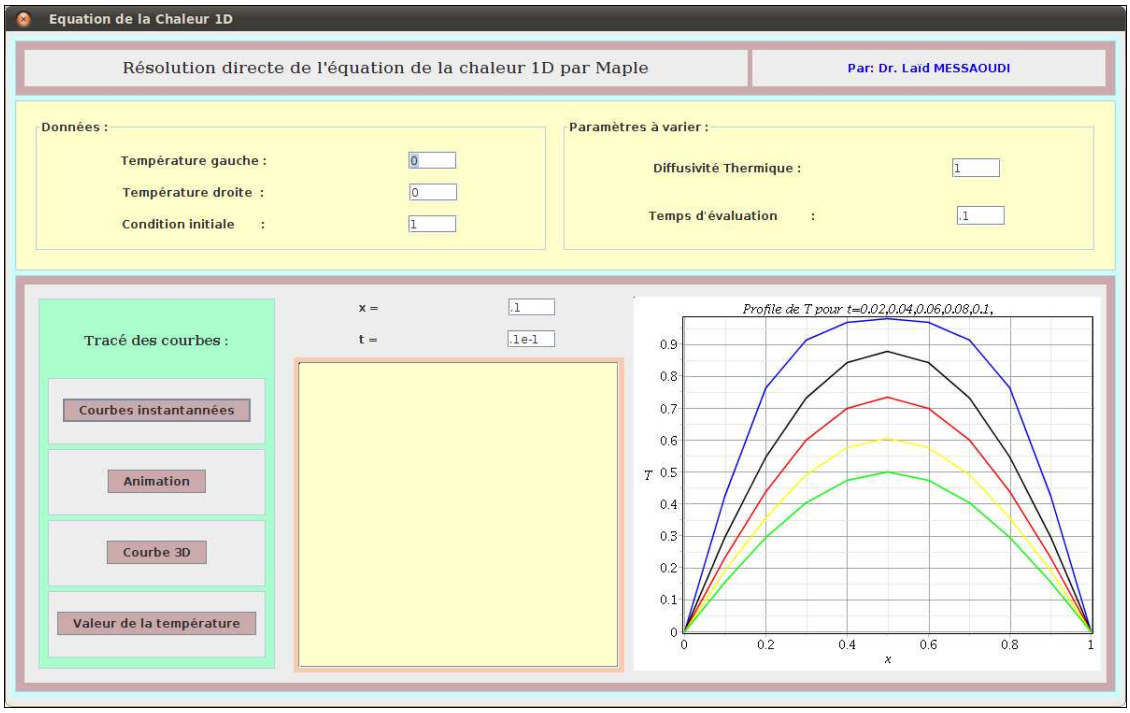



FIGURE 5.1: *Programme Diff1D-Map. Courbes instantannées.*

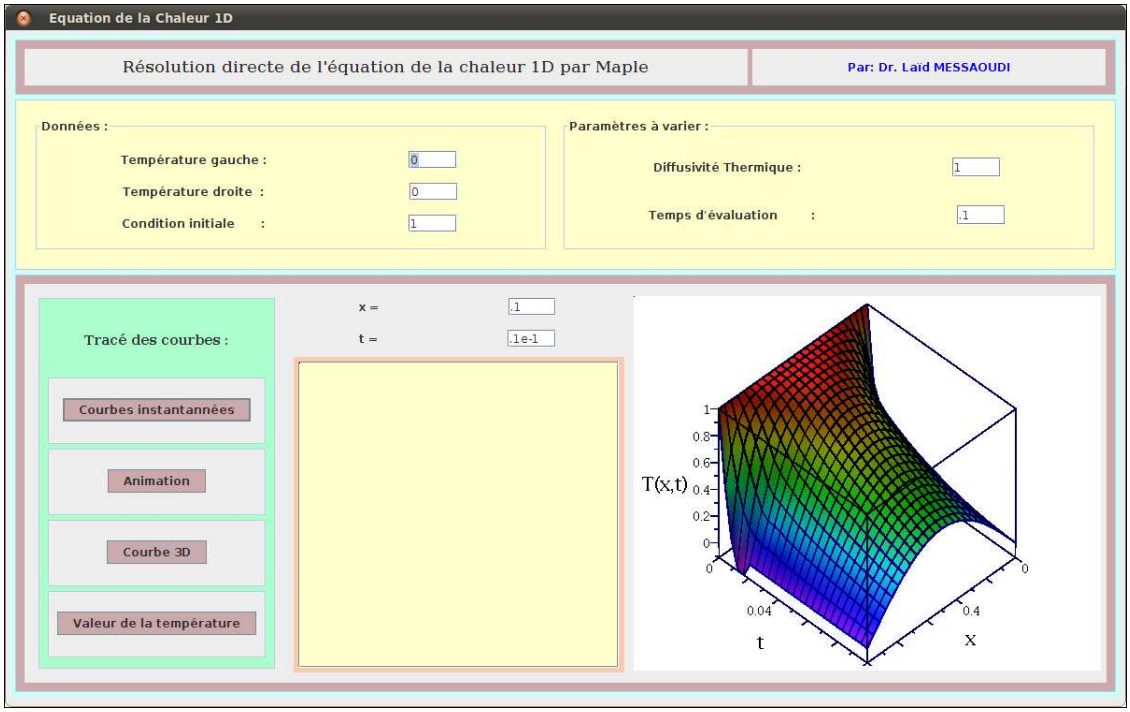



FIGURE 5.2: *Programme Diff1D-Map. Courbe 3D.*

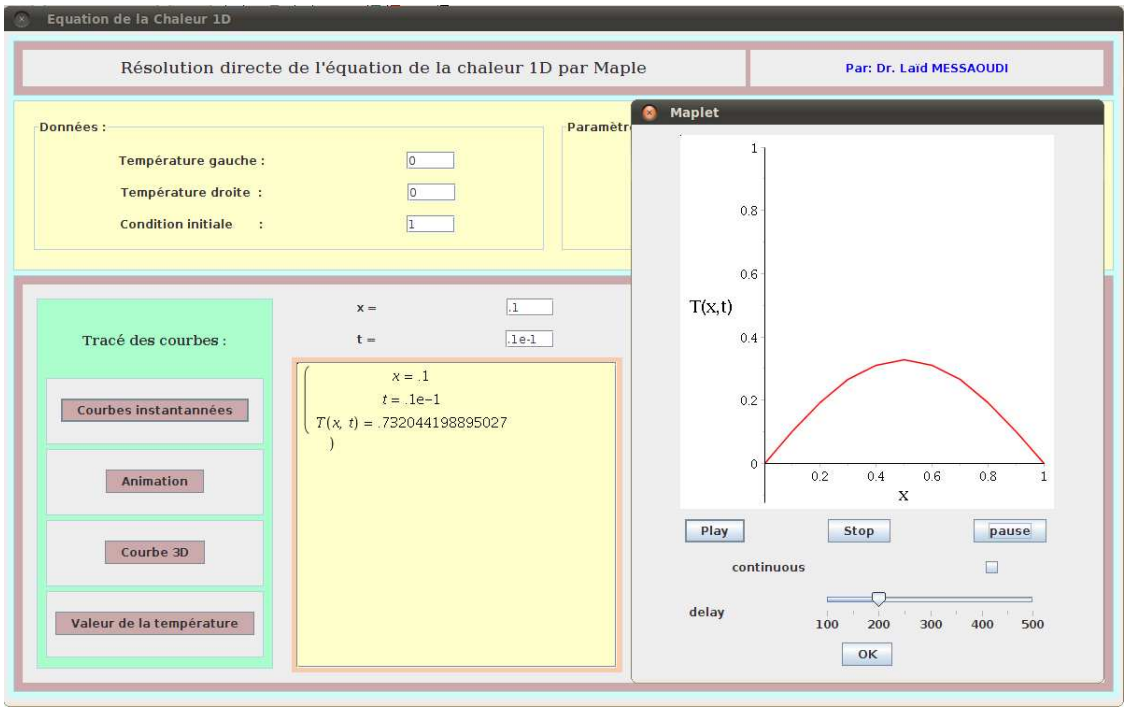



FIGURE 5.3: *Programme Diff1D-Map. Animation et valeur de la température.*

## 5.2 Equation de Laplace

Les programmes *Lap5-Map* et *Lap9-Map* sont de type pédagogiques qui résolvent l'équation de Laplace 2D dans une plaque rectangulaire en utilisant successivement le schéma à 5 points et à 9 points avec des (C.L) de type Dirichlet. Ci-dessous le code Maple du premier programme, le second ne diffère du premier que par l'équation discrétisée de la boucle principale ainsi que la prise en compte des (C.L) des coins qui sont prises égales aux moyennes des deux cotés formant le coin.

```
> restart: with(plottools): with(plots): with(Maplets[Elements]):
> with(GraphTheory): with(SpecialGraphs): with(LinearAlgebra):
```

```
> Maillage := proc (ndx, ndy)
local i_max, j_max, G;
i_max := ndx+1; j_max := ndy+1;
G := GridGraph(i_max, j_max); DrawGraph(G)
end proc:
```

```
> FormeMat := proc (ndx,ndy,Tg,Td,Tb,Th,C)
local a, b, beta, i, j, i_max, j_max, N, T, k, Eq, Temps, Eqs, Tmps, SolT,
Solution, Var, Sys, A, R;
beta := 1:
i_max := ndx+1:
j_max := ndy+1:
N := (i_max-2)*(j_max-2):
for i from 2 to i_max-1 do T[i,1] := Tb end do:
for i from 2 to i_max-1 do T[i,j_max] := Th end do:
for j from 2 to j_max-1 do T[1,j] := Tg end do:
for j from 2 to j_max-1 do T[i_max,j] := Td end do:
k := 1:
for j from 2 to j_max-1 do
for i from 2 to i_max-1 do
Eq[k]:=T[i+1,j]+T[i-1,j]+beta^2*(T[i,j+1]+T[i,j-1])-(2+2*beta^2)*T[i,j]=0:
Temps[k] := T[i,j]:
k := k+1:
end do
end do;
Eqs := seq(Eq[i],i = 1 .. N):
Tmps := [seq(Temps[i],i = 1 .. N)]:
SolT := solve(Eqs,Tmps):
Solution := evalf(SolT):
Sys := [seq(Eq[i],i = 1 .. N)]:
Var := [seq(Temps[i],i = 1 .. N)]:
A, R := GenerateMatrix(Sys,Var):
Var := convert(Var,Vector):
Solution := convert(Solution,Vector):
Sys := convert(Sys,Vector):
if C = 1 then A
elif C = 2 then Var
elif C = 3 then R
elif C = 4 then Solution
elif C = 5 then Sys
end if
end proc:
```

```
> CF1 := "#BFEBFF": CF2 := "#CCAAAC": CF3 := "#FFFFCC": CF4 := "#FACCAF":
CF5 := "#EAB8FB": CF6 := "#BDEBFE": CF7 := "#BDECBA": CF8 := "#FDE9BB":
CF9 := "#8E3557":
```

```
> maplette := Maplet(Window(
'title' = " Equation de Laplace 2D avec conditions de Dirichlet ",
width = 1200, height = 750, resizable = 'false',
['background' = CF5, BoxLayout['BL1'](border = true, 'background' = CF6,
BoxRow('background' = CF7,
[Label("Equation de Laplace 2D avec CL de Dirichlet: Schéma à 5 points",
'font' = Font("times", 18)), 'background' = CF7]),
[Label("Par: Dr. Laïd MESSAOUDI", 'foreground' = 'blue',
'font' = Font("times", 16))]), BoxLayout['BL2'](border = true,
'background' = CF6, BoxRow('background' = CF6, BoxColumn('background'
= CF6, [border = true, 'caption' = "Données", 'background' = CF6,
[border = false, 'background' = CF6, ['background' = CF6,
"Nombre de divisions dx:",TextField['ndx'](4,'value'=5)],['background'=CF6,
"Nombre de divisions dy:",TextField['ndy'](4,'value'=3)],['background'=CF6,
"Température à gauche:",TextField['Tg'](4,'value'=0)],['background'=CF6,
"Température à droite :",TextField['Td'](4,'value'=100)],['background'=CF6,
"Température en bas:",TextField['Tb'](4,'value'=200)],['background'=CF6,
"Température en haut:",TextField['Th'](4,'value'=0)]]]),
Plotter['PL1']('background'=CF8, 'width' = 450, 'height' = 250),
[MathMLViewer['MV0']('background' = CF3, 'width' = 140,
'value' = MathML[Export]([E])), 'background' = CF8])),
BoxLayout['BL3']('background' = CF8,
BoxRow('background' = CF8, BoxColumn('background' = CF8,
BoxRow(border = false, 'background' = CF8, ['background' = CF8,
[Button("Maillage", 'background' = CF9, Evaluate('PL1' = 'Maillage(ndx,
ndy)')), border = true, 'background' = CF6],
[Button("Equations", 'background' = CF9, Evaluate('MV0' = 'FormeMat(ndx,
ndy, Tg, Td, Tb, Th, 5)')), border = true, 'background' = CF6],
[Button("Matrice", 'background' = CF9, Evaluate('MV1' = 'FormeMat(ndx,
ndy, Tg, Td, Tb, Th, 1)')), border = true, 'background' = CF6],
[Button("Variables", 'background' = CF9, Evaluate('MV2' = 'FormeMat(ndx,
ndy, Tg, Td, Tb, Th, 2)')), border = true, 'background' = CF6],
[Button("Second membre", 'background' = CF9, Evaluate('MV3' = 'FormeMat(ndx,
ndy, Tg, Td, Tb, Th, 3)')), border = true, 'background' = CF6],
[Button("Résultats", 'background' = CF9, Evaluate('MV4' = 'FormeMat(ndx,
ndy, Tg, Td, Tb, Th, 4)')), border = true, 'background' = CF6]])),
[MathMLViewer['MV1']('background' = CF3, 'width' = 250, 'value' =
MathML[Export]([A])), 'background' = CF8],
[MathMLViewer['MV2']('background' = CF3, 'width' = 65, 'value' =
MathML[Export]([T])), 'background' = CF8],
[MathMLViewer['MV3']('background' = CF3, 'width' = 90, 'value' =
MathML[Export]([R])), 'background' = CF8],
[MathMLViewer['MV4']('background' = CF3, 'width' = 155, 'value' =
MathML[Export](Solution)), 'background' = CF8]))])):
result := Maplets[Display](maplette):
```

Les figures ci-dessous nous montrent les résultats de ces programmes.

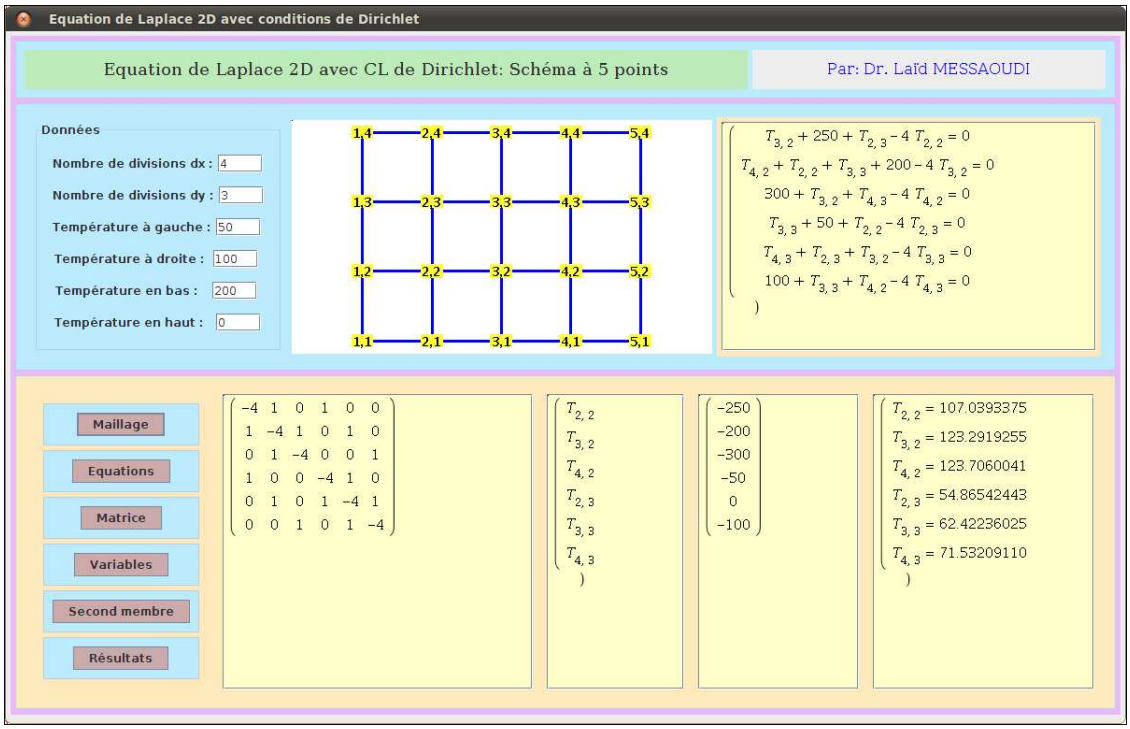



FIGURE 5.4: *Programme Lap-Map. Schéma à 5 points. Exemple 1.*

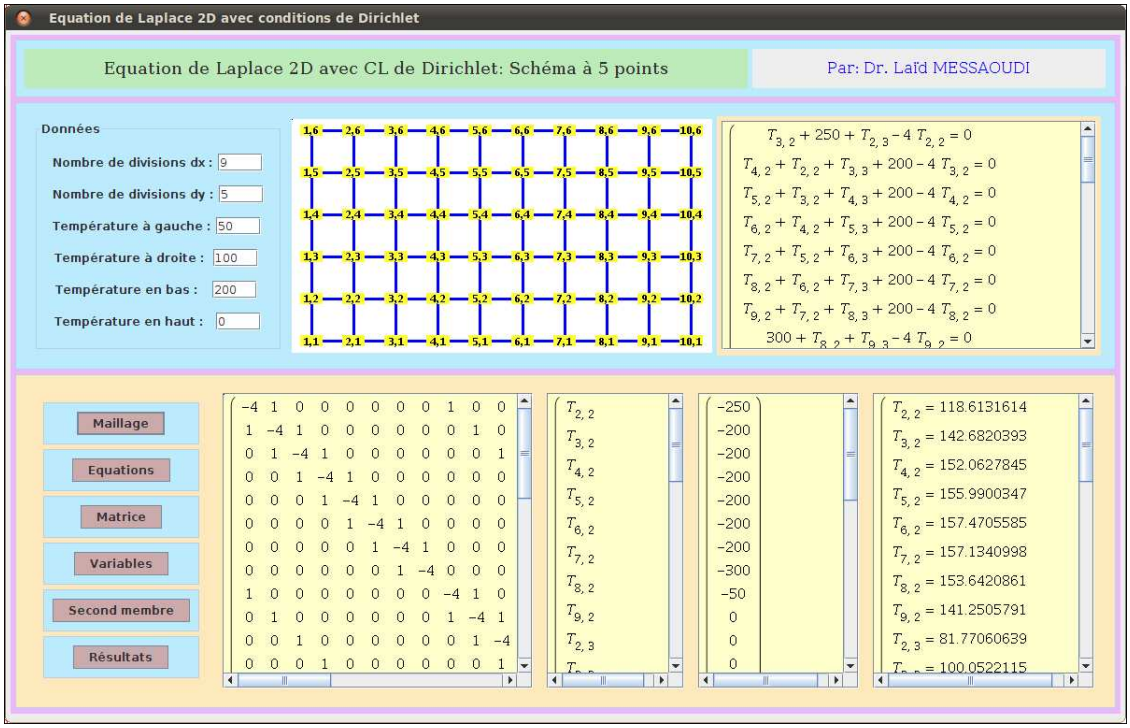



FIGURE 5.5: *Programme Lap-Map. Schéma à 5 points. Exemple 2.*

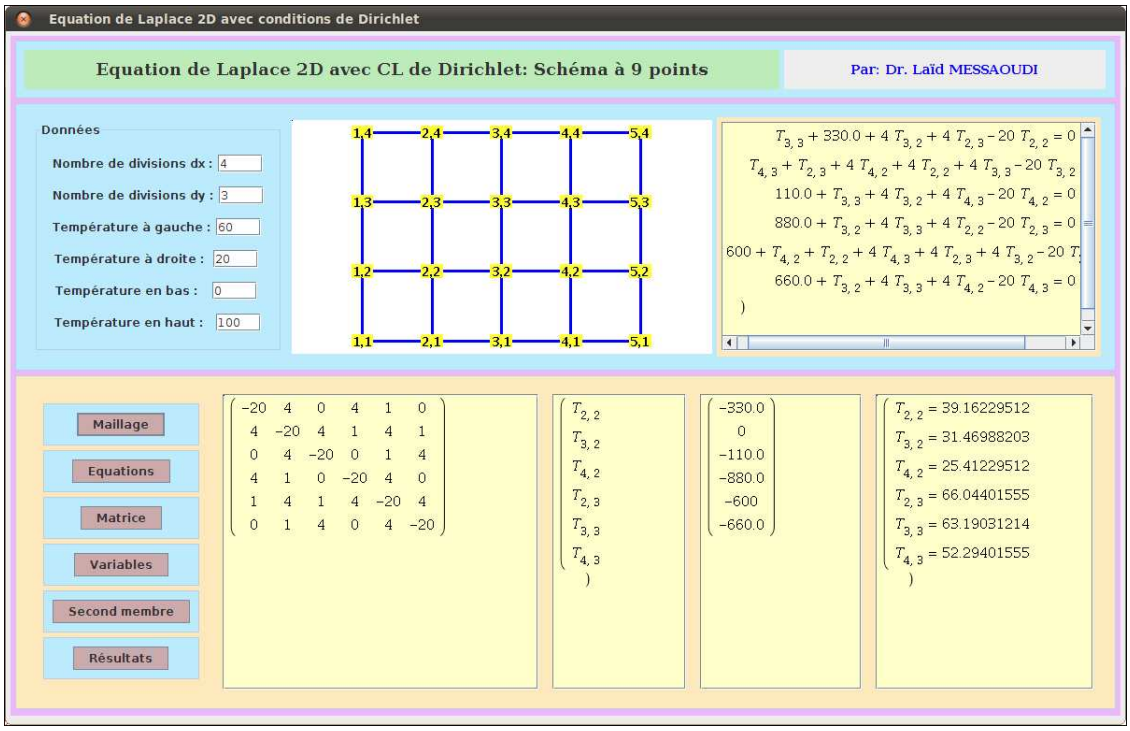



FIGURE 5.6: *Programme Lap-Map. Schéma à 9 points.*

Le programme *Lap5N-Map* ci-dessous donne la répartition de la température à travers la surface d'une plaque rectangulaire dont 3 extrémités sont soumises à des (C.L) de Dirichlet et la quatrième à une conditions de Neumann (C.N).

code

Exemple de résultats :

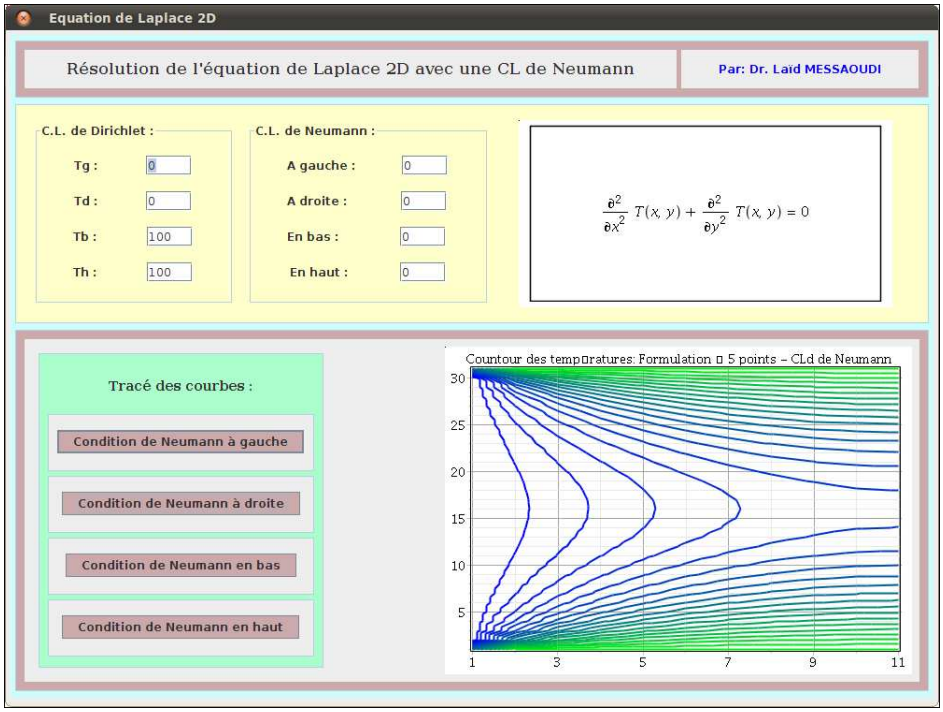



FIGURE 5.7: *Programme Lap5N-Map. Schéma à 5 points. Exemple 1.*

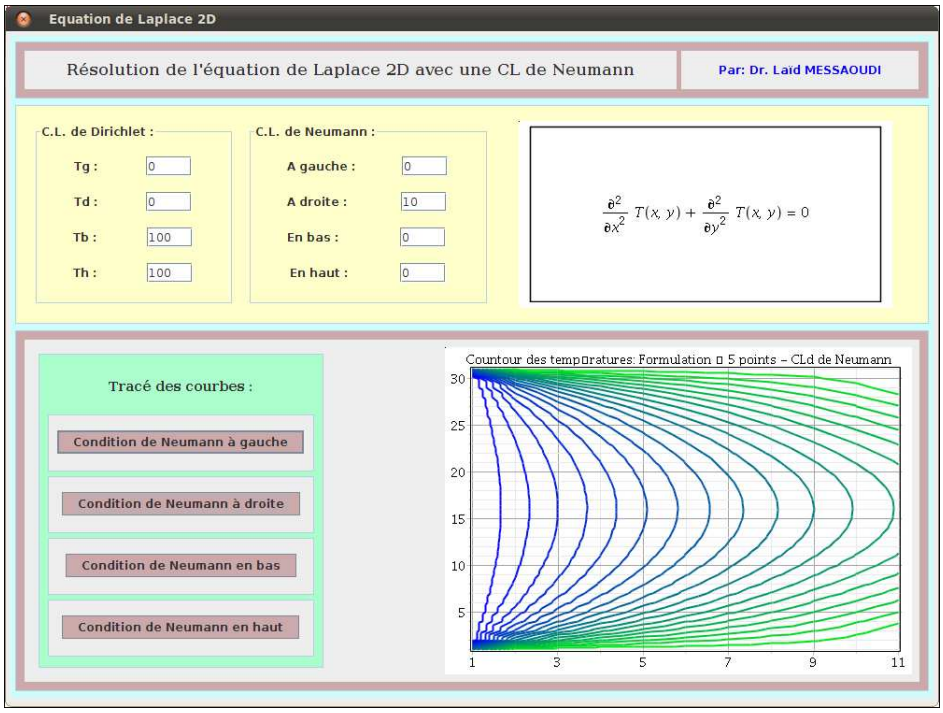



FIGURE 5.8: *Programme Lap5N-Map. Schéma à 5 points. Exemple 2.*

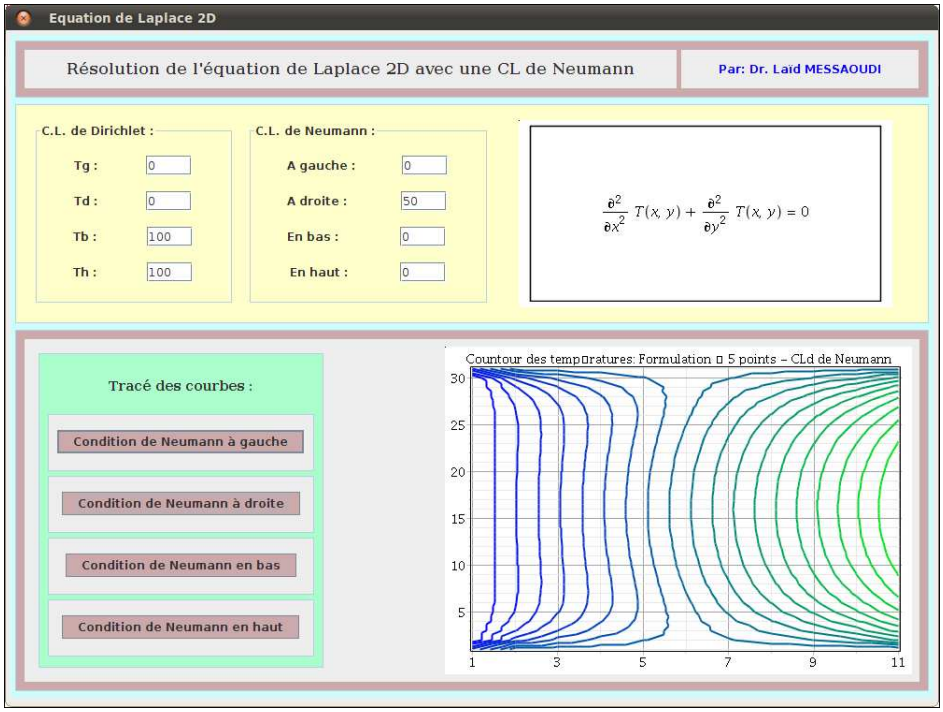



FIGURE 5.9: *Programme Lap5N-Map. Schéma à 5 points. Exemple 3.*

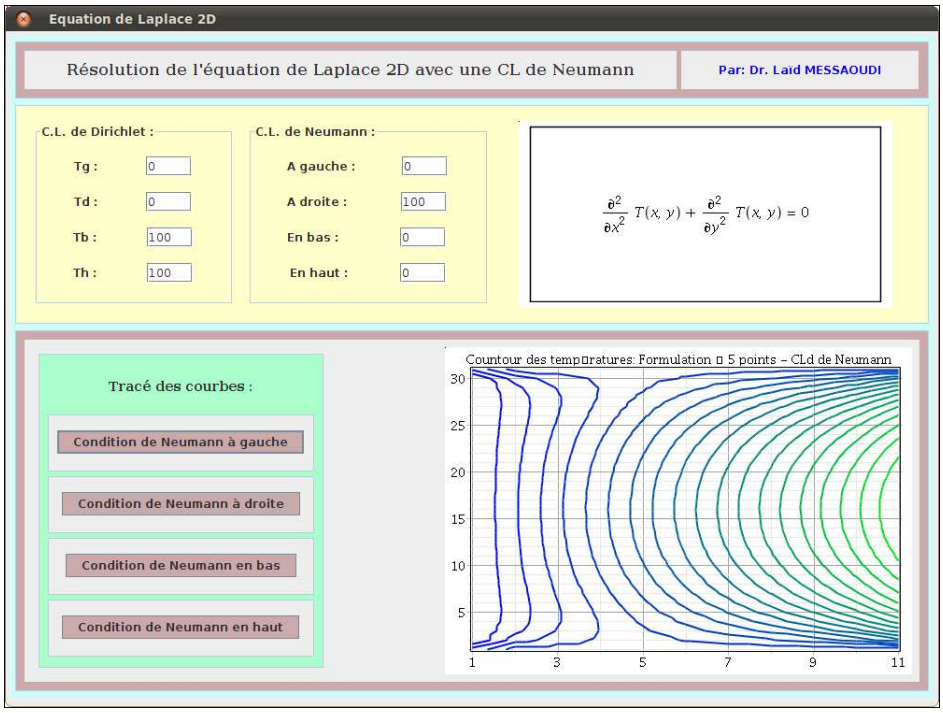



FIGURE 5.10: *Programme Lap5N-Map. Schéma à 5 points. Exemple 4.*

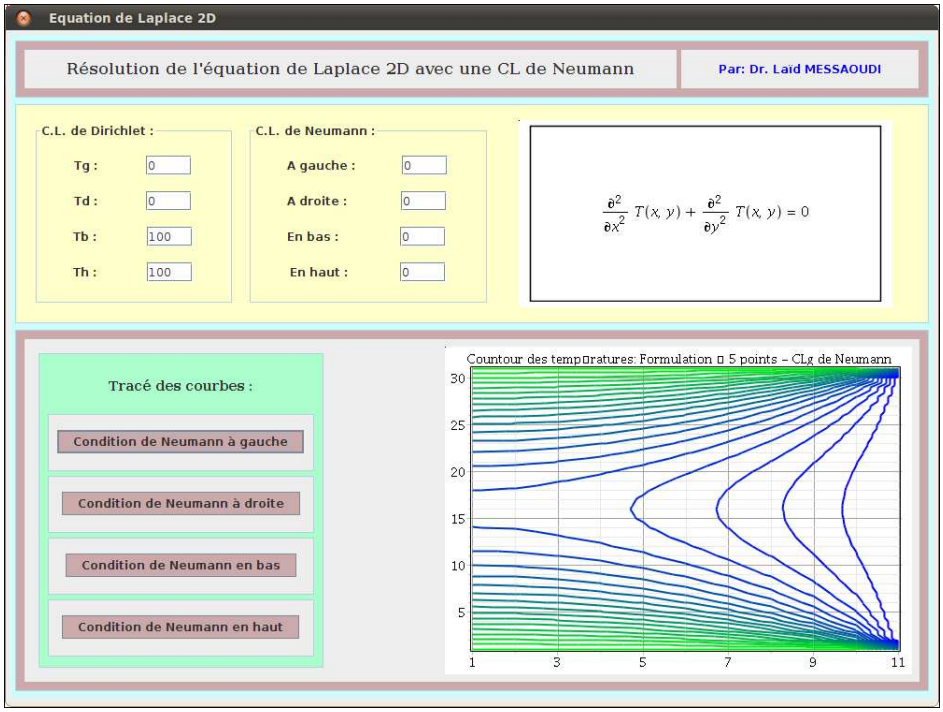



FIGURE 5.11: *Programme Lap5N-Map. Schéma à 5 points. Exemple 5.*

## 5.3 Equation de Poisson

## 5.4 Ecoulements potentiels

## 5.5 Ecoulement de Couette généralisé

Il s'agit dans ce programme de la résolution d'une EDP de type parabolique représentative de l'écoulement de Couette entre deux plaques planes. La solution est recherchée par la discrétisation en différences finies avec les schémas explicite, implicite et de Crank-Nicholson avec ou sans gradient de pression.

L'équation adimensionnelle est de la forme suivante :

$$\frac{\partial u(y,t)}{\partial t} = -\frac{\partial p}{\partial x} + \frac{1}{R}\frac{\partial^2 u(y,t)}{\partial y^2} \qquad 0 \leq y \leq 1\ ,\ t \geq 0 \tag{5.1}$$

$$\begin{cases} u(y,0) = 0 & 0 \leq y \leq 1 \\ u(0,t) = 1 & t > 0 \\ u(1,t) = 0 & t > 0 \end{cases}$$

```
> restart; with(plottools): with(plots): with(Maplets[Elements]):
```

```
> Explicite := proc ()
local R, ‘&Delta;y‘, ‘&Delta;t‘, lambda, i_max, Neq, pas, x, n_max, i,
n, k, u, Eq, Eqs, Vels, SolVel, lp, PlaqInf, PlaqSup, C1, C2, C3, C4,
C5, C6, C7, C8, C9, C10, Titre, TitreAxes, Carac;

R := 2000:   ‘&Delta;y‘ := 0.32e-1:   ‘&Delta;t‘ := 1:
lambda := ‘&Delta;t‘/(R*‘&Delta;y‘^2):
i_max := 31: n_max := 500:   Neq := i_max-2:
pas := evalf(1/(i_max-1),2):
x := seq(p, p = 0 .. 1, pas):
for i to i_max do u[i,0] := 0 end do:
for n to n_max do u[1,n] := 1 end do:
for n to n_max do u[i_max,n] := 0 end do:
for n from 0 to n_max-1 do
k := 1:
for i from 2 to i_max-1 do
Eq[k]:=u[i,n+1]=lambda*u[i-1,n]+(1-2*lambda)*u[i,n]+lambda*u[i+1,n];
k := k+1:
end do:
for k to Neq do Eq[k] end do:
Eqs := seq(Eq[i], i = 1 .. Neq):
Vels := [seq(u[i,n+1], i = 2 .. Neq+1)];
SolVel := solve(Eqs,Vels):
for i from 2 to Neq+1 do u[i,n+1] := rhs(SolVel[1,i-1]) end do:
lp[n] := [seq([u[i,n], x[i]], i = 1 .. i_max)]
end do:
PlaqInf := [[[-.2,0.],[1.2,0.]], color = magenta, thickness = 4]:
PlaqSup := [[[-.2,1.0],[1.2,1.0]], color = magenta, thickness = 4]:
C1 := [lp[10], color = gray, legend = "t = 10 "]:
C2 := [lp[20], color = blue, legend = "t = 20 "]:
C3 := [lp[30], color = orange, legend = "t = 30 "]:
C4 := [lp[40], color = cyan, legend = "t = 40 "]:
C5 := [lp[50], color = yellow, legend = "t = 50 "]:
C6 := [lp[60], color = gold, legend = "t = 60 "]:
C7 := [lp[80], color = green, legend = "t = 80 "]:
C8 := [lp[100], color = brown, legend = "t = 100 "]:
C9 := [lp[150], color = red, legend = "t = 150 "]:
C10 := [lp[n_max-1], color = black, style = point, symbol = circle,
legend = "t = 500"]:
Titre := title = "Profil de vitesse: Schéma Explicite d’Euler",
titlefont = [TIMES,BOLDITALIC,14]:
TitreAxes:=labels=["u/U0","y/d"], labelfont=[TIMES,BOLDITALIC,14]:
Carac:=Titre,TitreAxes,gridlines=false,filled=true,font=[TIMES,ITALIC,10]:
multiple(listplot,PlaqInf,PlaqSup,C1,C2,C3,C4,C5,C6,C7,C8,C9,C10,Carac):
end proc:
```

```
> Implicite := proc ()
local R, `&Delta;y`, `&Delta;t`, lambda, i_max, Neq, pas, x, n_max, i,
n, k, u, Eq, Eqs, Vels, SolVel, lp, PlaqInf, PlaqSup, C1, C2, C3, C4,
C5, C6, C7, C8, C9, C10, Titre, TitreAxes, Carac;

R := 2000:   `&Delta;y` := 0.2e-1:   `&Delta;t` := 1:
lambda := `&Delta;t`/(R*`&Delta;y`^2):
i_max := 31:  n_max := 300:  Neq := i_max-2:
pas := evalf(1/(i_max-1),2):
x := seq(p, p = 0 .. 1, pas):
for i to i_max do u[i,0] := 0 end do:
for n to n_max do u[1,n] := 1 end do:
for n to n_max do u[i_max,n] := 0 end do:
for n from 0 to n_max-1 do
k := 1:
for i from 2 to i_max-1 do
Eq[k]:=-lambda*u[i-1,n+1]+(1+2*lambda)*u[i,n+1]-lambda*u[i+1,n+1]=u[i,n]:
k := k+1:
end do:
for k to Neq do Eq[k] end do:
Eqs := seq(Eq[i], i = 1 .. Neq):
Vels := [seq(u[i,n+1], i = 2 .. Neq+1)]:
SolVel := solve(Eqs,Vels):
for i from 2 to Neq+1 do u[i,n+1] := rhs(SolVel[1,i-1]) end do:
lp[n] := [seq([u[i,n],x[i]],i = 1 .. i_max)]
end do:
PlaqInf := [[[-.2,0.], [1.2,0.]], color = magenta, thickness = 4]:
PlaqSup := [[[-.2,1.0], [1.2,1.0]], color = magenta, thickness = 4]:
C1 := [lp[10], color = gray, legend = "t = 10"]:
C2 := [lp[20], color = blue, legend = "t = 20"]:
C3 := [lp[30], color = orange, legend = "t = 30"]:
C4 := [lp[40], color = cyan, legend = "t = 40"]:
C5 := [lp[50], color = yellow, legend = "t = 50"]:
C6 := [lp[60], color = gold, legend = "t = 60"]:
C7 := [lp[80], color = green, legend = "t = 80"]:
C8 := [lp[100], color = brown, legend = "t = 100"]:
C9 := [lp[150], color = red, legend = "t = 150"]:
C10 := [lp[n_max-1], color = black, style = point, symbol = circle,
legend = "t = 300"]:
Titre := title = "Profil de vitesse: Schéma Implicite d'Euler",
titlefont=[TIMES,BOLDITALIC,14]:
TitreAxes := labels=["u/U0","y/d"],labelfont=[TIMES,BOLDITALIC,14]:
Carac:=Titre,TitreAxes,gridlines=false,filled=true,font=[TIMES,ITALIC,10]:
multiple(listplot,PlaqInf,PlaqSup,C1,C2,C3,C4,C5,C6,C7,C8,C9,C10,Carac):
end proc:
```

```
> CrankNicholson := proc (R, dp)
local ‘&Delta;y‘, ‘&Delta;t‘, lambda, i_max, Neq, pas, x, n_max, i, n,
k, u, Eq, Eqs, Vels, SolVel, lp, PlaqInf, PlaqSup, C1, C2, C3, C4, C5,
C6, C7, C8, C9, C10, Titre, TitreAxes, Carac;

‘&Delta;y‘ := 0.2e-1:  ‘&Delta;t‘ := 1:
lambda := (1/2)*‘&Delta;t‘/(R*‘&Delta;y‘^2):
i_max := 31:  n_max := 300:  Neq := i_max-2:
pas := evalf(1/(i_max-1), 2):
x := seq(p, p = 0 .. 1, pas):
for i to i_max do u[i,0] := 0 end do:
for n to n_max do u[1,n] := 1 end do:
for n to n_max do u[i_max,n] := 0 end do:
for n from 0 to n_max-1 do
k := 1:
for i from 2 to i_max-1 do
Eq[k]:=-lambda*u[i-1,n+1]+(1+2*lambda)*u[i,n+1]-lambda*u[i+1,n+1]=
lambda*u[i-1,n]+(1-2*lambda)*u[i,n]+lambda*u[i+1,n]+dp:
k := k+1
end do:
for k to Neq do Eq[k] end do:
Eqs := seq(Eq[i], i = 1 .. Neq):
Vels := [seq(u[i, n+1], i = 2 .. Neq+1)]:
SolVel := solve(Eqs, Vels):
for i from 2 to Neq+1 do u[i,n+1] := rhs(SolVel[1,i-1]) end do:
lp[n] := [seq([u[i,n], x[i]], i = 1 .. i_max)]
end do:
PlaqInf := [[[-.4,0.],[1.4,0.]], color = magenta, thickness = 4]:
PlaqSup := [[[-.4,1.0],[1.4,1.0]], color = magenta, thickness = 4]:
C1 := [lp[10], color = gray, legend = "t = 10"]:
C2 := [lp[20], color = blue, legend = "t = 20"]:
C3 := [lp[30], color = orange, legend = "t = 30"]:
C4 := [lp[40], color = cyan, legend = "t = 40"]:
C5 := [lp[50], color = yellow, legend = "t = 50"]:
C6 := [lp[60], color = gold, legend = "t = 60"]:
C7 := [lp[80], color = green, legend = "t = 80"]:
C8 := [lp[100], color = brown, legend = "t = 100"]:
C9 := [lp[150], color = red, legend = "t = 150"]:
C10 := [lp[n_max-1], color = black, style = point, symbol = circle,
legend = "t = 300"]:
Titre:=title="Profil de vitesse: Schéma de Crank-Nicholson",
titlefont=[TIMES,BOLDITALIC,14]:
TitreAxes:=labels=["u/U0","y/d"],labelfont=[TIMES,BOLDITALIC,14]:
Carac:=Titre,TitreAxes,gridlines=false,filled=true,font=[TIMES,ITALIC,10]:
multiple(listplot,PlaqInf,PlaqSup,C1,C2,C3,C4,C5,C6,C7,C8,C9,C10,Carac):
end proc:
```

```
> CF1 := "#BFEBFF"; CF2 := "#CCAAAC"; CF3 := "#FFFFCC"; CF4 := "#FACCAF";
```

```
> maplette := Maplet(Window( 'title'= " Superposition des écoulements
élémentaires   ", width=1000, height=750, resizable='false',
['background'= CF1, BoxLayout['BL1'](border=true,'background'=CF2,
BoxRow([Label("Ecoulement de Couette généralisé",'font'=Font("times",20))]),
[Label("Par: Dr. Laïd MESSAOUDI",'foreground'='blue','font'=Font("times",14))]),
BoxLayout['BL2'](border=true, BoxRow(border=true,'background'=CF3,
[Label("Variation de paramètres pour le schéma de Cranck-Nicholson:",

'font'=Font("times",bold,16)),'background'=CF3,['background'=CF3,"Nombre
de Reynolds :",TextField['R'](5,'value'=2000)],["Gradient de pression
: ",'background'=CF3, TextField['dp'](5,'value'=0.01)]])),
BoxLayout['BL3'](border=true,'background'=CF2, BoxRow('background'=CF3,
BoxColumn(border=true,'background'=CF3,BoxRow('background'=CF3, [Label("Tracé
des courbes:", 'font' = Font( "times",bold, 16)),'background'=CF3,
[Button("Shéma explicite d'Euler",'background'='blue',
Evaluate('PL1'='Explicite')),'background'=CF1,border=true],
[Button("Shéma implicite d'Euler",'background'='blue',
Evaluate('PL1'='Implicite')),'background'=CF1,border=true],
[Button("Shéma implicite de Crank-Nicholson sans dp",'background'='blue',
Evaluate('PL1'='CrankNicholson(R,0)')),'background'=CF1,border=true],
[Button("Shéma implicite de Crank-Nicholson avec dp",'background'='blue',
Evaluate('PL1'='CrankNicholson(R,dp)')),'background'=CF1,border=true]])),
[Label("                          "),'background'=CF3],
Plotter['PL1']('background'=CF4,'width'=500,'height'=350))])]):
result := Maplets[Display](maplette):
```

Les figures ci-dessous nous donnent les résultats de ce programme.

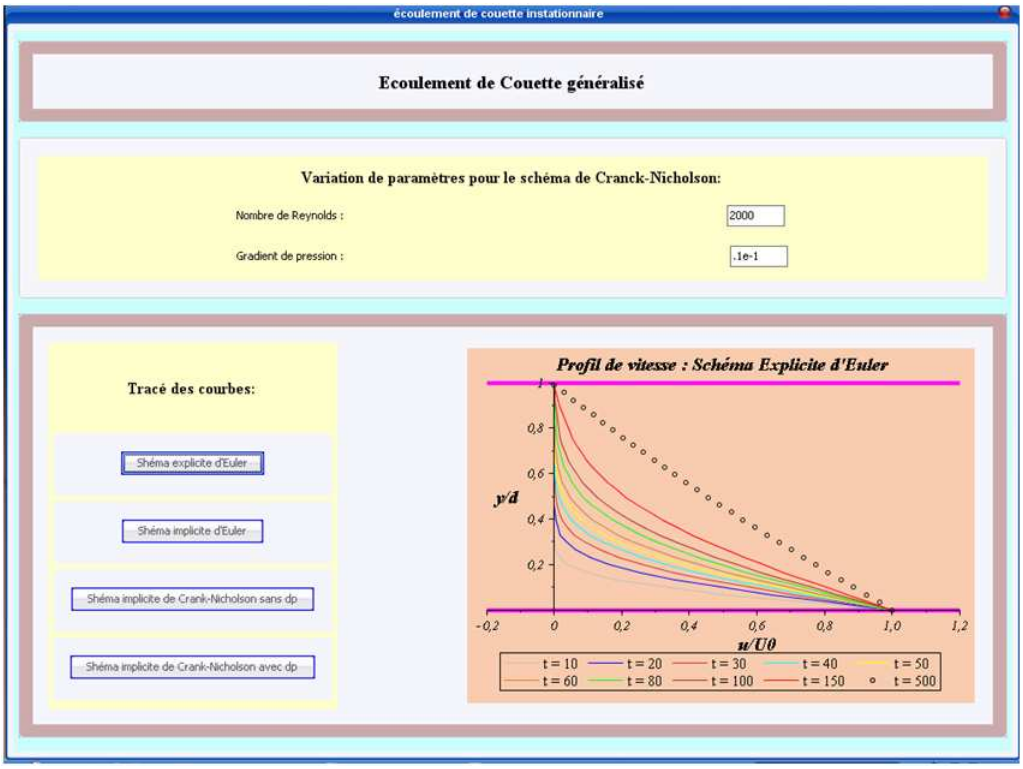



FIGURE 5.12: *Programme ECG-Map. Schéma explicite d'Euler.*

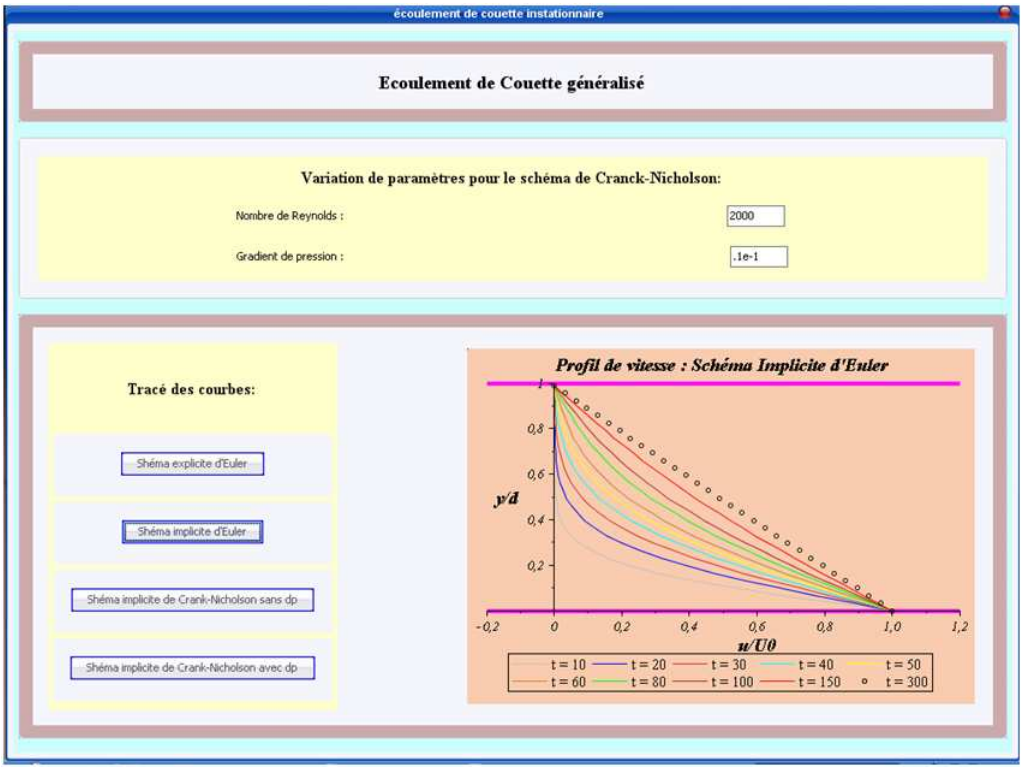



FIGURE 5.13: *Programme ECG-Map. Schéma implicite d'Euler.*

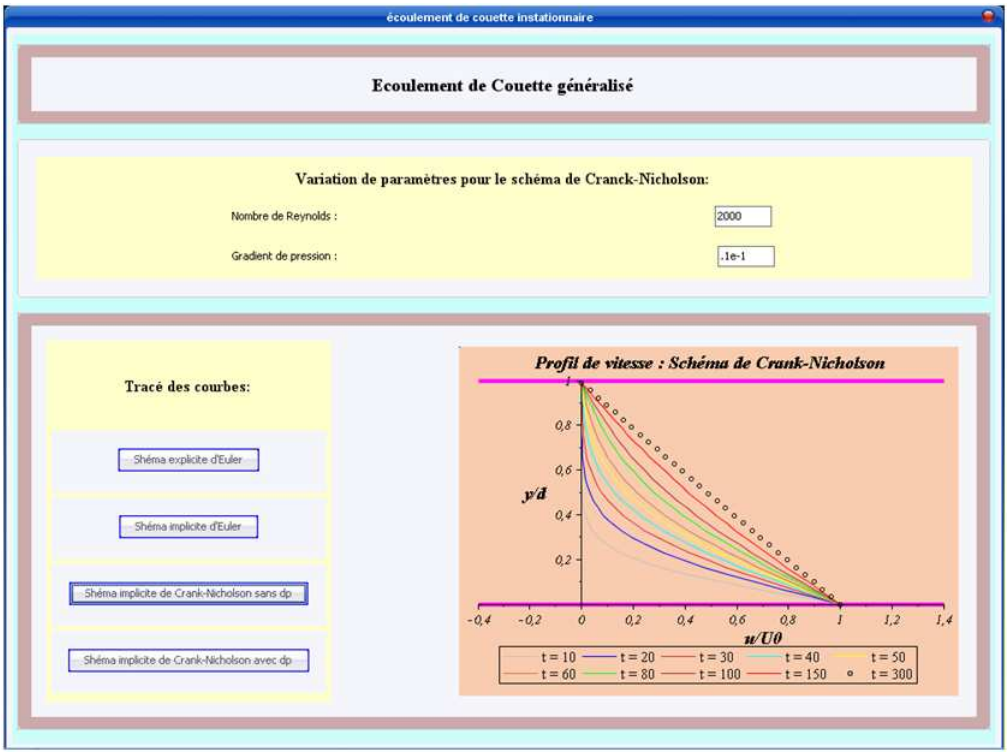



FIGURE 5.14: *Programme ECG-Map. Schéma de Crank-Nocholson sans gradient de pression.*

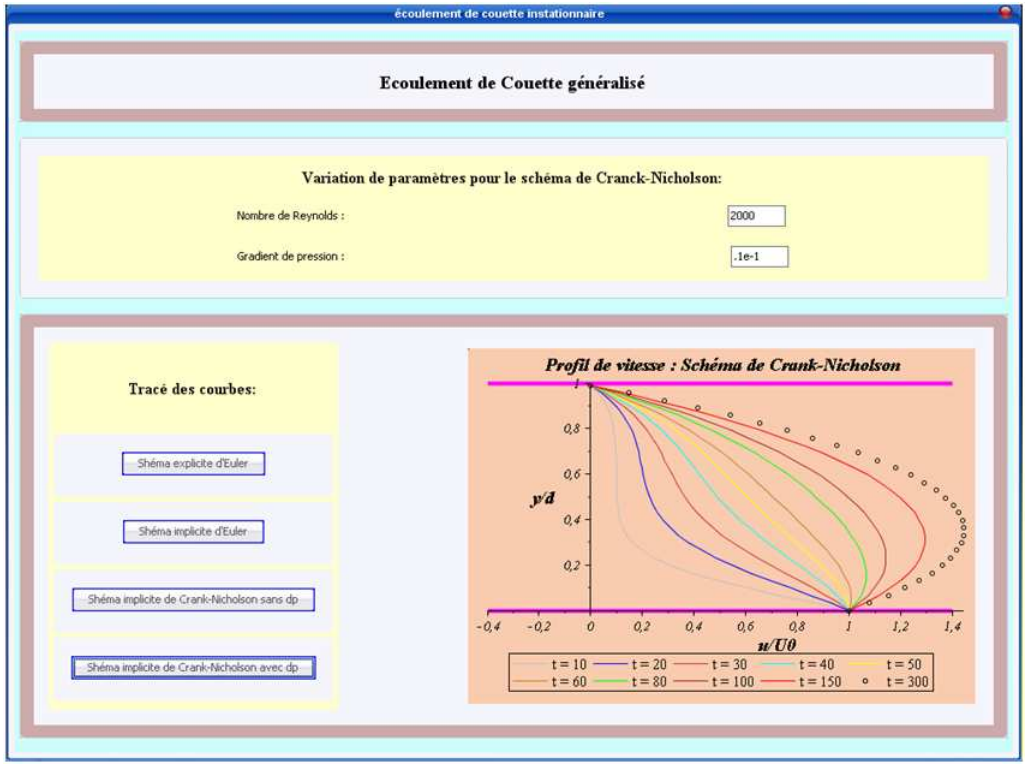



FIGURE 5.15: *Programme ECG-Map. Schéma de Crank-Nocholson avec gradient de pression favorable.*

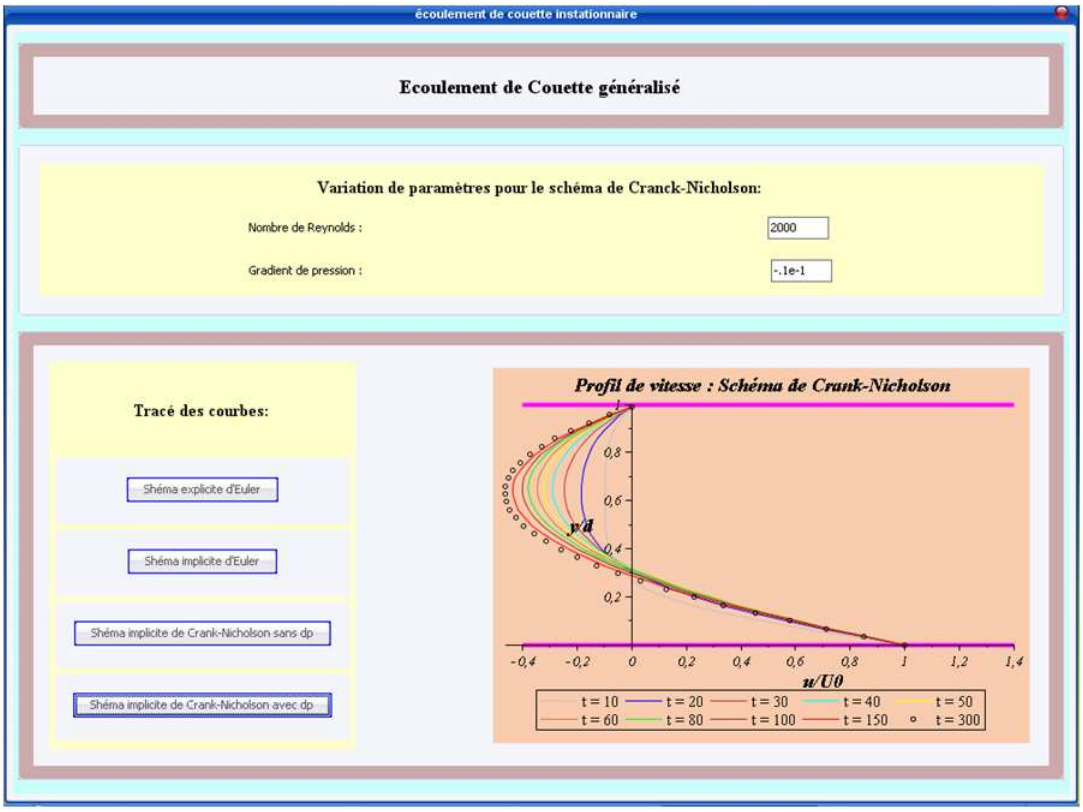



FIGURE 5.16: *Programme ECG-Map. Schéma de Crank-Nocholson avec gradient de pression défavorable.*

# Bibliographie


[1] Joe D. Hoffmann, "*Numerical Methods for Engineers and Scientists*", 2nd Ed., McGraw-Hill, New York, 2001.

[2] Chapra, S. C., and Canale, R. P., "*Numerical Methods for Engineers*", 3rd Ed., McGraw-Hill, New York, 1998.

[3] Aslak Tveito and Ragnar Winther, "*Introduction to Partial Differential Equations : A Computational*", Springer-Verlag, New York, 1998.

[4] Y. Pinchover and J. Rubinstein, "*An Introduction to Partial Differential Equations*", Cambridge University Press, 2005.

[5] Richard hyberman, "*Elementary Applied Partial Differential Equations*", 2nd. Ed., Prentice Hall, USA, 1983.

[6] Klauss A. Hoffmann, Steve T. Chiang, "*Computational Fluid Dynamics*", 4th Ed., Engineering Education System, USA, 2000.

[7] William F. Ames, "*Numerical Methods for Partial Differential Equations*", 2nd Ed., Academic Press, USA, 1977.

[8] H. F. Weinberger, "*A First Cours in Partial Differential Equations with Complexe Variables and Transform Methods* ", Dover Publications, New York, 1965.

[9] David L. Powers, "*Boundary Value Problems and Partial Differential Equations*", 5th Ed., Elsevier Academic Press, USA, 2006.

[10] David R. Croft, David G. Lilley, "*Heat transfer calculations using Finite Difference Equations*", Applied Science Publishers, England, 1977.

[11] Henry J. Ricardo, "*A Modern Introduction to Differential Equations*", 2nd Ed., Elsevier, Canada, 2009.

[12] Jayathi Y. Murthy, "*Numerical Methods in Heat, Mass, and Momentum Transfer*", Purdue University, 2002.

[13] K. W. Morton, David Mayers, "*Numerical Solution of Partial Differential Equations*", 2nd Ed., Cambridge, 2005.

[14] Peter J. Collins, "*Differential and Integral Equations*", Oxford University Press, New York, 2006.

[15] C. Pozrikidis, "*Fluid Dynamics : Theory, Computation, and Numerical Simulation*", 2nd Ed., Springer, New York, 2009.

[16] Aslak Tveito, Ragnar Winther, "*Introduction to Partial Differential Equations : A Computational Approach*", Springer-Verlag, New York, 1998.

[17] John C. Tannehill, Dale A. Anderson, Richard H. Pletcher, "*Computational Fluid Mechanics And Heat Transfer*", 2nd Ed., Taylor&Francis, USA, 1997.

[18] H. K. Versteeg, W. Malalasekera, "*An Introduction to Computational Fluid Dynamics. The Finite Volume Method*", 1st Ed., Longman scientific&Technical, USA, 1995.

[19] Suhas V. Patankar, "*Numerical Heat Transfer and Fluid Flow*", Hemisphere Publishing, USA, 1980.