Résolution de l'équation de la chaleur par la méthode des différences finies

Classe .NET 2.0

SimulationDiffusionThermique2D

Sommaire

1. Présentation

1.1) Langage de programmation

Cette classe est programmée en ***Visual Basic 2005*** de Microsoft et est basée sur le .NET Framework version 2.0.

Elle peut être intégrée dans tout programme écrit en VB, C#, C++, Python… tant que celui-ci utilise le .NET Framework 2.0 ou supérieur.

1.2) Description du phénomène physique

Il y a trois modes de transfert thermique : la conduction, la convection et le rayonnement.

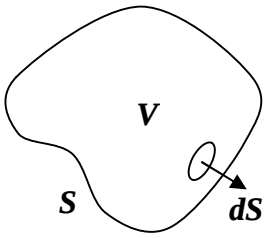
Cette classe ne s'intéresse qu'au phénomène de **conduction thermique**.

A l'échelle microscopique, les atomes sont agités à cause de la température (agitation thermique) et communiquent une partie de cette agitation à leurs voisins. A l'échelle macroscopique, cela engendre un échange énergétique sous forme de quantité de chaleur. Concrètement, la chaleur va des régions chaudes vers les régions froides, et la température tend à s'uniformiser.

Cet échange d'énergie se fait sans déplacement de matière, sinon il s'agit de convection.

1.3) Mise en équation

Remarque : Ce paragraphe établit l'équation de la chaleur. Il nécessite certains pré-requis en physique, du niveau de 1er cycle universitaire.



Considérons une surface fermée ***S*** fixe et indéformable délimitant un volume ***V.***

Dans le cas de la conduction, aucune matière n'entre ou ne sort. Cette surface définit donc un système fermé.

Appliquons le 1er principe de la thermodynamique en termes de puissance :

$$\frac{\partial \boldsymbol{U}}{\partial \boldsymbol{t}} = \boldsymbol{P_Q} + 0$$

avec $\boldsymbol{U}$ l'énergie interne à $\boldsymbol{S}$ et $\boldsymbol{P_Q}$ la puissance thermique échangée.

Comme $\boldsymbol{S}$ est fixe et indéformable :

$$\iiint_V \frac{\partial \boldsymbol{u}}{\partial \boldsymbol{t}}.\boldsymbol{dV} = -\oiint_S \vec{\boldsymbol{j}}_{\boldsymbol{th}}.\boldsymbol{d\vec{S}}$$

avec $\boldsymbol{u}$ l'énergie interne volumique et $\overrightarrow{\boldsymbol{j}_{\boldsymbol{th}}}$ le vecteur densité de flux de chaleur.

(le signe – provient de l'orientation sortante de $\boldsymbol{S}$)

$$\iiint_V \rho.\boldsymbol{C}.\frac{\partial \boldsymbol{T}}{\partial \boldsymbol{t}}.\boldsymbol{dV} = -\oiint_S \vec{\boldsymbol{j}}_{\boldsymbol{th}}.\boldsymbol{d\vec{S}}$$

en supposant ρ (masse volumique) et $\boldsymbol{C}$ (capacité thermique massique) constantes.

Le théorème de Green-Ostrogradski donne :

$$\iiint_V \rho.\boldsymbol{C}.\frac{\partial \boldsymbol{T}}{\partial \boldsymbol{t}}.\boldsymbol{dV} = -\iiint_V \boldsymbol{div}(\overrightarrow{\boldsymbol{j}_{\boldsymbol{th}}}).\boldsymbol{dV}$$

$$\iiint_V (\rho.\boldsymbol{C}.\frac{\partial \boldsymbol{T}}{\partial \boldsymbol{t}} + \boldsymbol{div}(\overrightarrow{\boldsymbol{j}_{\boldsymbol{th}}})).\boldsymbol{dV} = 0$$

Comme cette relation est vraie $\forall\, \boldsymbol{S}$, alors nécessairement :

$$\rho.\boldsymbol{C}.\frac{\partial \boldsymbol{T}}{\partial \boldsymbol{t}} + \boldsymbol{div}(\overrightarrow{\boldsymbol{j}_{\boldsymbol{th}}}) = 0$$

La loi phénoménologique de Fourier stipule que $\overrightarrow{\boldsymbol{j}_{\boldsymbol{th}}} = -\lambda.\overrightarrow{\boldsymbol{grad}}(\boldsymbol{T})$, où λ est la conductivité thermique du matériau.

On a alors :

$$\rho.\boldsymbol{C}.\frac{\partial \boldsymbol{T}}{\partial \boldsymbol{t}} = \lambda.\Delta \boldsymbol{T}$$

$$\boxed{\frac{\partial \boldsymbol{T}}{\partial \boldsymbol{t}} = \boldsymbol{K}.\Delta \boldsymbol{T}} \quad \text{avec} \quad \boxed{\boldsymbol{K} = \frac{\lambda}{\rho.\boldsymbol{C}}} \quad \text{la diffusivité thermique du matériau.}$$

Cette équation est appelée « équation de la chaleur »

La classe SimulationDiffusionThermique2D ne travaille, comme son nom l'indique, qu'en 2D. La température ***T*** ne dépend donc que de ***x*** et ***y*** (dans un repère cartésien).

L'équation de la chaleur est donc :

$$\frac{\partial \boldsymbol{T}}{\partial \boldsymbol{t}} = \boldsymbol{K}.(\frac{\partial^2 \boldsymbol{T}}{\partial \boldsymbol{x}^2} + \frac{\partial^2 \boldsymbol{T}}{\partial \boldsymbol{y}^2})$$

La classe SimulationDiffusionThermique2D permet de résoudre numériquement cette équation aux dérivées partielles.

1.4) Possibilités de la classe

La classe SimulationDiffusionThermique2D permet de résoudre les problèmes bidimensionnels ou pouvant se ramener à 2 dimensions.

Les **conditions initiales** peuvent être quelconques (température pas nécessairement uniforme).

Les **conditions aux limites** sont du type :

- Température imposée
- Température variable
- Paroi adiabatique
- Flux thermique imposé
- Flux conducto-convectif

Les conditions aux limites doivent être présentes sur toute la périphérie de la zone de travail, mais elles peuvent aussi être placées en tout point de celle-ci. On peut par exemple imposer une température au centre de la zone de travail.

Les conditions aux limites peuvent également être modifiées entièrement en cours de calcul. On peut par exemple chauffer un endroit de la zone de travail pendant un certain temps, puis observer l'évolution de la température.

Les **limitations** de la classe sont les suivantes :

- Matériau nécessairement homogène. Les problèmes de mise en contact de deux matériaux différents ne sont pas supportés.
- Les conditions ‘Température variable’ (*) , ‘Flux thermique imposé’ et ‘Flux conducto-convectif’ ont les mêmes paramètres pour l’ensemble des points où elles sont appliquées. On ne peut pas définir deux valeurs de flux imposés différentes.

(*) On peut contourner cette limitation en modifiant des températures fixes entre chaque pas de calcul.

2. Méthode des différences finies

2.1) Méthodes de résolution numériques

L'équation de la chaleur ne peut être résolue analytiquement que dans des cas simples. Si le problème devient plus complexe, on ne peut trouver de solution que sous forme de série de Fourier, voire pas du tout. Une résolution numérique fonctionne quelque soit la complexité géométrique du problème (avec cependant les limitations exposées précédemment).

Le principe de toutes les résolutions numériques est de discrétiser l'espace et le temps. Dans un domaine continu, la température est définie en tout point. Dans un domaine discrétisé (sous forme de grille), la température n'est définie qu'en un nombre fini de points. Les valeurs intermédiaires sont obtenues par interpolation.

Il convient donc de garder à l'esprit qu'une résolution numérique n'est qu'une **approximation** de la véritable solution.

2.1) Principe de la méthode des différences finies

La méthode des différences finies discrétise l'espace (le plan dans notre cas) en une grille de points régulièrement espacés d'une distance $\boldsymbol{h}$.

Le terme de ***pixel*** sera abusivement utilisé dans ce document ; les points précédemment mentionnés correspondent au centre de chaque pixel.

De même, le temps est également discrétisé. La solution est calculée en un nombre fini de ***pas*** espacés d'une durée τ.

La température est ici une fonction de trois variables : ***T(x,y,t)***

La méthode des différences finies est basée sur le développement de Taylor. La dérivée partielle par rapport au temps s'obtient par :

$$\boldsymbol{T}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t}+\tau) = \boldsymbol{T}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t}) + \tau.\frac{\partial \boldsymbol{T}}{\partial \boldsymbol{t}}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t}) + \boldsymbol{O}(\tau^2)$$

d'où : $$\frac{\partial \boldsymbol{T}}{\partial \boldsymbol{t}}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t}) = \frac{\boldsymbol{T}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t}+\tau) - \boldsymbol{T}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t})}{\tau} + \boldsymbol{O}(\tau)$$

On néglige alors le terme en *O(τ)* (c'est l'erreur de troncature). On a donc la dérivée partielle par rapport au temps :

$$\boxed{\frac{\partial \boldsymbol{T}}{\partial \boldsymbol{t}}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t}) = \frac{\boldsymbol{T}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t}+\tau) - \boldsymbol{T}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t})}{\tau}}$$

On trouve de la dérivée seconde en effectuant un développement de Taylor à l'ordre 2 :

$$\begin{cases} \boldsymbol{T}(\boldsymbol{x}+\boldsymbol{h},\boldsymbol{y},\boldsymbol{t}) = \boldsymbol{T}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t}) + \boldsymbol{h}.\frac{\partial \boldsymbol{T}}{\partial \boldsymbol{x}}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t}) + \frac{\boldsymbol{h}^2}{2}.\frac{\partial^2 \boldsymbol{T}}{\partial \boldsymbol{x}^2}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t}) + \boldsymbol{O}(\boldsymbol{h}^3) \\ \boldsymbol{T}(\boldsymbol{x}-\boldsymbol{h},\boldsymbol{y},\boldsymbol{t}) = \boldsymbol{T}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t}) - \boldsymbol{h}.\frac{\partial \boldsymbol{T}}{\partial \boldsymbol{x}}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t}) + \frac{\boldsymbol{h}^2}{2}.\frac{\partial^2 \boldsymbol{T}}{\partial \boldsymbol{x}^2}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t}) + \boldsymbol{O}(\boldsymbol{h}^3) \end{cases}$$

En faisant la somme membre à membre :

$$\boldsymbol{T}(\boldsymbol{x}+\boldsymbol{h},\boldsymbol{y},\boldsymbol{t}) + \boldsymbol{T}(\boldsymbol{x}-\boldsymbol{h},\boldsymbol{y},\boldsymbol{t}) = 2\boldsymbol{T}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t}) + \boldsymbol{h}^2.\frac{\partial^2 \boldsymbol{T}}{\partial \boldsymbol{x}^2}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t}) + \boldsymbol{O}(\boldsymbol{h}^3)$$

d'où : $$\frac{\partial^2 \boldsymbol{T}}{\partial \boldsymbol{x}^2}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t}) = \frac{\boldsymbol{T}(\boldsymbol{x}+\boldsymbol{h},\boldsymbol{y},\boldsymbol{t}) + \boldsymbol{T}(\boldsymbol{x}-\boldsymbol{h},\boldsymbol{y},\boldsymbol{t}) - 2\boldsymbol{T}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t})}{\boldsymbol{h}^2} + \boldsymbol{O}(\boldsymbol{h})$$

On néglige de même le terme en ***O(h)*** :

$$\boxed{\frac{\partial^2 \boldsymbol{T}}{\partial \boldsymbol{x}^2}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t}) = \frac{\boldsymbol{T}(\boldsymbol{x}+\boldsymbol{h},\boldsymbol{y},\boldsymbol{t}) + \boldsymbol{T}(\boldsymbol{x}-\boldsymbol{h},\boldsymbol{y},\boldsymbol{t}) - 2\boldsymbol{T}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t})}{\boldsymbol{h}^2}}$$

On remplace alors dans l'équation aux dérivées partielles :

$$\frac{\boldsymbol{T}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t}+\tau) - \boldsymbol{T}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t})}{\tau} = \frac{\boldsymbol{K}}{\boldsymbol{h}^2}\left[\boldsymbol{T}(\boldsymbol{x}+\boldsymbol{h},\boldsymbol{y},\boldsymbol{t}) + \boldsymbol{T}(\boldsymbol{x}-\boldsymbol{h},\boldsymbol{y},\boldsymbol{t}) + \boldsymbol{T}(\boldsymbol{x},\boldsymbol{y}+\boldsymbol{h},\boldsymbol{t}) + \boldsymbol{T}(\boldsymbol{x},\boldsymbol{y}-\boldsymbol{h},\boldsymbol{t}) - 4\boldsymbol{T}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t})\right]$$

En posant $\boxed{\alpha = \frac{\boldsymbol{K}.\tau}{\boldsymbol{h}^2}}$ on trouve alors :

$$\boldsymbol{T}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t}+\tau) - \boldsymbol{T}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t}) = \alpha.\left[\boldsymbol{T}(\boldsymbol{x}+\boldsymbol{h},\boldsymbol{y},\boldsymbol{t}) + \boldsymbol{T}(\boldsymbol{x}-\boldsymbol{h},\boldsymbol{y},\boldsymbol{t}) + \boldsymbol{T}(\boldsymbol{x},\boldsymbol{y}+\boldsymbol{h},\boldsymbol{t}) + \boldsymbol{T}(\boldsymbol{x},\boldsymbol{y}-\boldsymbol{h},\boldsymbol{t}) - 4\boldsymbol{T}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t})\right]$$

D'où :

$$\boxed{\boldsymbol{T}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t}+\tau) = (1-4\alpha).\boldsymbol{T}(\boldsymbol{x},\boldsymbol{y},\boldsymbol{t}) + \alpha.\left[\boldsymbol{T}(\boldsymbol{x}+\boldsymbol{h},\boldsymbol{y},\boldsymbol{t}) + \boldsymbol{T}(\boldsymbol{x}-\boldsymbol{h},\boldsymbol{y},\boldsymbol{t}) + \boldsymbol{T}(\boldsymbol{x},\boldsymbol{y}+\boldsymbol{h},\boldsymbol{t}) + \boldsymbol{T}(\boldsymbol{x},\boldsymbol{y}-\boldsymbol{h},\boldsymbol{t})\right]}$$

On a donc trouvé la température à l'instant ***t*** en fonction des températures à l'instant ***t-τ***. La classe SimulationDiffusionThermique2D met en œuvre cet algorithme.

Le calcul de la température en un point de la grille nécessite de connaître sa valeur aux points adjacents. On ne peut donc pas calculer aux bords de la zone de travail, d'où la présence impérative de conditions aux limites.

Cet algorithme est obtenu en effectuant une approximation. Il est donc impératif d'avoir τ et ***h*** suffisamment petits, sous peine de voir diverger la solution.

On peut montrer que la solution est convergente si $\boxed{\alpha < \frac{1}{4}}$

3. Eléments de la classe

3.1) Enumérations

- Enum **CouleurDégradé** As **Integer**

  Résumé :

  Enumère les différents types de dégradé utilisés pour afficher les températures

  Membres :

  - ***TempératureMulti*** = 0 
  - ***NoirEtRouge*** = 1 
  - ***NoirEtBlanc*** = 2 
  - ***BleuEtRouge*** = 3 
  - ***DégradéPerso*** = 4 D'après Bitmap

- Enum **ConstanteCL** As **Integer**

  Résumé :

  Enumère les différents types de conditions aux limites possibles

  Membres :

  - ***CLRien*** = -9990
  - ***CLTempVariable*** = -9991
  - ***CLAdiabatique*** = -9992
  - ***CLFluxImposé*** = -9993
  - ***CLFluxCondConv*** = -9994

3.2) Paramètres physiques

- **CLFluxCondConvh** As **Double**

  Résumé :

  Paramètre 'h' du flux conducto-convectif (en W/m²/°K)

- **CLFluxCondConvText** As **Double**

  Résumé :

  Paramètre 'Température extérieure' du flux conducto-convectif (en °C)

- **CLFluxImposé** As **Double**

  Résumé :

  Valeur du flux imposé (en W/m²)

- **CLTempératureVariable** As **Double**

  Résumé :

  Température instantanée des zones 'Conditions aux limites variables' (en °C)

- **K** As **Double**

  Résumé :

  Diffusivité thermique (en m²/s)

  Notes :

  K = Lambda / (ρ . C)

- **Lambda** As **Double**

  Résumé :

  Conductivité (en W/m/°K)

- **PasSpatial** As **Double**

  Résumé :

  Distance entre deux pixels (en m)

- **PasTemporel** As **Double**

  Résumé :

  Temps entre deux pas (en s)

3.3) Propriétés

- Property **ConserverHistorique**() As **Boolean**

  Résumé :

  Indique si le composant mémorise l'ensemble les étapes ou seulement l'étape finale (recommandé pour minimiser l'occupation mémoire)

  Valeur par défaut :

  True

  Notes :

  Si False, les étapes intermédiaires ne sont pas conservées en mémoire. L'appel aux fonctions suivantes est alors interdit :

  - ObtenirHistoriqueTempérature
  - ObtenirImage
  - ObtenirTableau
  - ObtenirTempérature

- Property **CouleurAdiabatique**() As **System**.**Drawing**.**Color**

  Résumé :

  Couleur d'affichage d'un pixel adiabatique

  Valeur par défaut :

  Color.Black

- Property **CouleurFluxCondConv**() As **System**.**Drawing**.**Color**

  Résumé :

  Couleur d'affichage du flux conducto-convectif

  Valeur par défaut :

  Color.Purple

- Property **CouleurFluxImposé**() As **System**.**Drawing**.**Color**

  Résumé :

  Couleur d'affichage du flux imposé

  Valeur par défaut :

  Color.Purple

- Property **DégradéPersoBitmap**() As **System**.**Drawing**.**Bitmap**

  Résumé :

  Obtient ou définit le bitmap utilisé pour le dégradé personnalisé

  Notes :

  Le Bitmap contenant les couleurs du dégradé personnalisé doit mesurer 100 pixels de large

- Property **DégradéType**() As **SimulationDiffusionThermique2D**.**CouleurDégradé**

  Résumé :

  Obtient ou définit le dégradé utilisé pour les bitmaps

  Valeur par défaut :

  TempératureMulti

- ReadOnly Property **NombreDePasEffectués**() As **Integer**

  Résumé :

  Retourne le nombre de pas effectués

- ReadOnly Property **Précision**() As **Double**

  Résumé :

  Retourne l'écart maximal de température entre les 2 derniers pas

- ReadOnly Property **SolutionConvergente**() As **Boolean**

  Résumé :

  Indique si la solution converge

- ReadOnly Property **Taille**() As **System**.**Drawing**.**Size**

  Résumé :

  Retourne les dimensions du tableau

- Property **TempératureMaxi**() As **Double**

  Résumé :

  Retourne la température maxi (température correspondant au rouge pour le bitmap initial)

- Property **TempératureMini**() As **Double**

  Résumé :

  Retourne la température mini (température correspondant au noir pour le bitmap initial)

3.4) Méthodes de calcul

- Sub **AvancerPlusieursPas**(ByVal ***NombreDePas*** As **Integer**)

  Résumé :

  Avance d'un nombre défini de pas

  Paramètres :

  NombreDePas: Nombre de pas

- Sub **AvancerUnPas**()

  Résumé :

  Avance d'un pas (c'est-à-dire d'un temps égal au pas temporel)

- Sub **Initialiser**(ByVal ***TableauConditionsInitiales***(,) As **Double**, ByVal ***TableauConditionsLimites***(,) As **Double**)

  Résumé :

  Initialise le composant avec des tableaux contenant les conditions initiales et aux limites

  Paramètres :

  TableauConditionsInitiales: Tableau contenant les températures initiales

  TableauConditionsLimites: Tableau contenant les conditions aux limites

  Notes :

  Les deux tableaux doivent avoir des dimensions identiques

- Sub **Initialiser**(ByVal ***imgConditionsInitiales*** As **System**.**Drawing**.**Bitmap**, ByVal ***imgConditionsLimites*** As **System**.**Drawing**.**Bitmap**, ByVal ***TemperatureMini*** As **Double**, ByVal ***TemperatureMaxi*** As **Double**)

  Résumé :

  Initialise le composant avec les images et les paramètres

  Paramètres :

  imgConditionsInitiales: Bitmap contenant les températures à l'instant initial

  imgConditionsLimites: Bitmap contenant les conditions aux limites

  TemperatureMaxi: Température correspondant au rouge pur dans les bitmaps

  TemperatureMini: Température correspondant au noir dans les bitmaps

  Notes :

  Les deux tableaux doivent avoir des dimensions identiques

  TempératureMaxi doit être supérieure à ***TempératureMini***

- Sub **ModifierConditionsLimites**(ByVal ***TableauConditionsLimites***(**,**) As **Double**)

  Résumé :

  Modifie les conditions aux limites

  Paramètres :

  TableauConditionsLimites: Tableau contenant les nouvelles conditions aux limites

- Sub **ModifierConditionsLimites**(ByVal ***imgConditionsLimites*** As **System**.**Drawing**.**Bitmap**, ByVal ***TemperatureMini*** As **Double**, ByVal ***TemperatureMaxi*** As **Double**)

  Résumé :

  Modifie les conditions aux limites

  Paramètres :

  imgConditionsLimites: Bitmap contenant les nouvelles conditions aux limites

  TemperatureMini: Température correspondant au noir dans le bitmap

  TemperatureMaxi: Température correspondant au rouge pur dans le bitmap

  Notes :

  imgConditionsLimites doit avoir les mêmes dimensions que l'image initiale

  TempératureMaxi doit être supérieure à ***TempératureMini***

- Sub **Reset**()

  Résumé :

  Réinitialise le pas pour un éventuel nouveau calcul

  Notes :

  N'affecte pas les conditions initiales ni les conditions aux limites

  Déjà appelée par la méthode Initialiser

- Function **InitialiserPasTemporelMaxi**(ByVal ***_PasSpatial*** As **Double**, ByVal ***_K*** As **Double**) As **Double**

  Résumé :

  Initialise PasSpatial, PasTemporel et K avec la plus grande valeur du pas temporel pour que la solution reste convergente et retourne cette valeur de PasTemporel

  Notes :

  Choisit PasTemporel tel que $\alpha = 0.24$

3.5) Méthodes d'obtention des résultats

- Function **DégradéObtenirImage**() As **System**.**Drawing**.**Bitmap**

  Résumé :

  Retourne une image de 100 pixels par 1 pixel contenant le dégradé de températures en cours

- Function **ObtenirHistoriqueTempérature**(ByVal ***x*** As **Integer,** ByVal ***y*** As **Integer**) As **Double**()

  Résumé :

  Retourne un tableau unidimensionnel contenant la température d'un point donné au cours du temps

  Paramètres :

  x: Abscisse du point

  y: Ordonnée du point

- Function **ObtenirHistoriqueTempérature**(ByVal ***x*** As **Integer,** ByVal ***y*** As **Integer,** ByVal ***PasInitial*** As **Integer,** ByVal ***PasFinal*** As **Integer**) As **Double**()

  Résumé :

  Retourne un tableau unidimensionnel contenant la température d'un point donné au cours du temps entre deux instants donnés

  Paramètres :

  x: Abscisse du point

  y: Ordonnée du point

  PasInitial: Instant initial (nombre de pas temporels)

  PasFinal: Instant final (nombre de pas temporels)

- Function **ObtenirImage**(ByVal ***Pas*** As **Integer**) As **System**.**Drawing**.**Bitmap**

  Résumé :

  Retourne une image affichant les températures après un nombre de pas temporels donnés

  Paramètres :

  Pas: Nombre de pas temporels

  Notes :

  Utilise les températures d'initialisation pour les valeurs extrêmes du dégradé

- Function **ObtenirImage**(ByVal ***Pas*** As **Integer**, ByVal ***TemperatureMini*** As **Double**, ByVal ***TemperatureMaxi*** As **Double**) As **System**.**Drawing**.**Bitmap**

  Résumé :

  Retourne une image affichant les températures après un nombre de pas temporels donnés

  Paramètres :

  Pas: Nombre de pas temporels

  TemperatureMini: Température minimale du dégradé de couleurs, en dessous de laquelle les couleurs sont identiques

  TemperatureMaxi: Température maximale du dégradé de couleurs, au dessus de laquelle toutes les couleurs sont identiques

- Function **ObtenirImageFinale**() As **System**.**Drawing**.**Bitmap**

  Résumé :

  Retourne une image affichant les températures au dernier instant calculé

  Notes :

  Utilise les températures d'initialisation pour les valeurs extrêmes du dégradé

- Function **ObtenirImageFinale**(ByVal ***TemperatureMini*** As **Double**, ByVal ***TemperatureMaxi*** As **Double**) As **System**.**Drawing**.**Bitmap**

  Résumé :

  Retourne une image affichant les températures au dernier instant calculé

  Paramètres :

  TemperatureMini: Température minimale du dégradé de couleurs, en dessous de laquelle les couleurs sont identiques

  TemperatureMaxi: Température maximale du dégradé de couleurs, au dessus de laquelle toutes les couleurs sont identiques

- Function **ObtenirTableau**(ByVal ***Pas*** As **Integer**) As **Double**(,)

  Résumé :

  Retourne le tableau des températures après un nombre de pas temporels donné

  Paramètres :

  Pas: Instant auquel on veut obtenir le tableau (en nombre de pas temporels effectués)

- Function **ObtenirTableauFinal**() As **Double**(,)

  Résumé :

  Retourne le tableau des températures au dernier instant calculé

- Function **ObtenirTempérature**(ByVal ***Pas*** As **Integer,** ByVal ***x*** As **Integer,** ByVal ***y*** As **Integer**) As **Double**

  Résumé :

  Retourne la température d'un point donné à un instant donné

  Paramètres :

  Pas: Instant auquel on veut obtenir le tableau (en nombre de pas temporels effectués)

  x: Abscisse du point

  y: Ordonnée du point

- Function **ObtenirTempératureFinale**(ByVal ***x*** As **Integer,** ByVal ***y*** As **Integer**) As **Double**

  Résumé :

  Retourne la température d'un point donné au dernier instant calculé

  Paramètres :

  x: Abscisse du point

  y: Ordonnée du point

4. Utilisation de la classe

4.1) Vue d'ensemble

La classe SimulationDiffusionThermique2D fournit au programmeur toutes les méthodes nécessaires à la résolution de l'équation de la chaleur, sans que celui-ci n'ait à s'inquiéter de leur mise en œuvre.

Pour cela, créer une instance de cette classe, et effectuer les étapes suivantes :

- Définir les paramètres physiques. Les paramètres ***PasSpatial***, ***PasTemporel*** et ***K*** doivent obligatoirement être complétés. Les autres ne peuvent l'être que si nécessaires. Veiller à vérifier que la solution converge !
- Initialiser les conditions initiales et aux limites
- Effectuer le nombre désiré de pas de calcul
- Récupérer les résultats (température en un point donné, tableau de températures ou image)

Il est possible d'effectuer les pas de calcul uns par uns, et de récupérer les résultats à chaque fois.

En cas de conditions aux limites variables, le calcul doit être mené pas-à-pas, et les conditions aux limites doivent être mises à jour entre chaque pas.

4.2) Code de couleurs

Les conditions initiales et aux limites peuvent être entrées de deux manières.

- Sous forme de **tableau** :

  Créer un tableau de **Double** contenant les températures initiales, et un autre contenant les conditions aux limites. Entrer les températures fixées, et utiliser l'énumération ***ConstanteCL*** pour les autres conditions aux limites.

- Sous forme d'**image :**

  Utiliser deux images **Bitmap** contenant les températures initiales et les conditions aux limites.

Pour les conditions initiales :

L'ensemble de l'image doit être remplie.

La température est définie par une nuance de rouge. Le noir correspond à ***TempératureMini*** et le rouge pur à ***Températuremaxi***.

TempératureMini ***TempératureMaxi***

Pour les conditions aux limites :

La périphérie de l'image doit intégralement contenir des conditions aux limites.

Elles sont définies avec le code de couleurs suivant :

Enumération ***ConstanteCL***	**Description**	**Couleur**
CLRien	Pas de condition aux limites : la température évolue librement (conduction thermique)	Blanc
-	Température fixée (code de couleurs idem conditions initiales)	Nuance de rouge
CLTempVariable	Température fixée, mais variable dans le temps	Jaune
CLAdiabatique	Paroi adiabatique : aucun échange énergétique à travers ce pixel	Bleu
CLFluxImposé	Le vecteur densité de flux thermique est fixé	Vert
CLFluxCondConv	Flux conducto-convectif	Gris

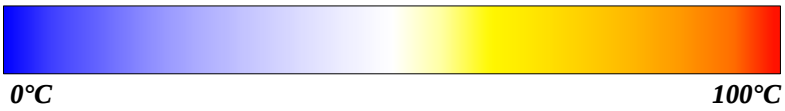
4.3) Exemples

Tous les exemples ont en commun les paramètres suivants :

- PasSpatial = 0.01 m
- PasTemporel = 0.1 s
- K = 0.0001 m²/s

Les températures extrêmes sont 0 et 100°C. Le code de couleurs pour les Bitmap est donc :

Les résultats sont obtenus en utilisant le dégradé ***TempératuresMulti*** (Enumération ***CouleurDégradé)***, avec l'échelle de températures par défaut :



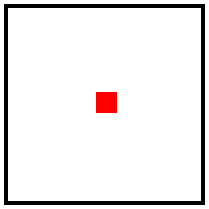
Chaque exemple présente les Bitmaps utilisés, un extrait de code en ***Visual Basic*** et les résultats retournés par l'instance de la classe.

Exemple 1 :

Une plaque carrée a ses bords maintenus à 0°C, mais son centre est chauffé en permanence à 100°C. Sa température à l'instant initial est de 0°C.

Les Bitmap mesurent 50 pixels × 50 pixels :

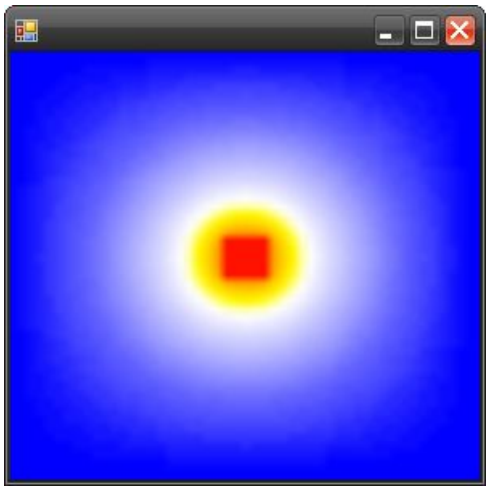
ImageCI ***ImageCL***



Extrait de code :

```
'Créé une instance de la classe
Dim SDT As New SimulationDiffusionThermique2D

'Définit les paramètres physiques
SDT.PasSpatial = 0.01
SDT.PasTemporel = 0.1
SDT.K = 0.0001

'Initialise avec les conditions initiales et aux limites
SDT.Initialiser(ImageCI, ImageCL, 0, 100)

'Calcule 1000 pas
SDT.AvancerPlusieursPas(1000)

'Récupère l'image du dernier pas calculé
Dim bmpRésultat As Bitmap = SDT.ObtenirImageFinale()

'Affiche cette image
Dim frmRésultat As New Windows.Forms.Form
frmRésultat.BackgroundImage = bmpRésultat
frmRésultat.BackgroundImageLayout = ImageLayout.Stretch
frmRésultat.Show()
```

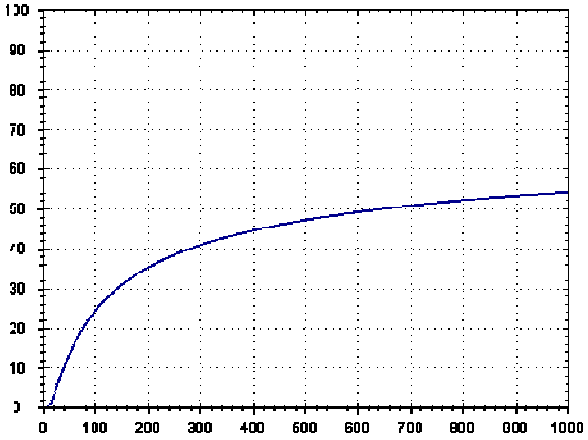
Voici le résultat de ce code :



On peut obtenir l'historique d'un point, par exemple le point de coordonnées (20,20) :

```
Dim Historique As Double() = SDT.ObtenirHistoriqueTempérature(20, 20)
```

La variable ***Historique*** contient alors les températures suivantes :



Exemple 2 :

Une plaque rectangulaire de 50cm × 30cm a les bords isolés. Sa température initiale est non uniforme : elle vaut 0°C sur toute la surface, sauf sur une portion de 10cm de côté où elle vaut 100°C. On veut connaître la température finale.

Les Bitmaps mesurent donc 50 × 30 pixels :

ImageCI

ImageCL

Pour avoir la valeur finale, il faut laisser le temps à la température de bien s'uniformiser. On va donc calculer un grand nombre de pas (10 000). Pour ne pas saturer la mémoire, on ne va pas conserver les valeurs intermédiaires.

Extrait de code :

```
'Créé une instance de la classe
Dim SDT As New SimulationDiffusionThermique2D

'Ne conserve pas les étapes intermédiaires en mémoire
SDT.ConserverHistorique = False

'Définit les paramètres physiques
SDT.PasSpatial = 0.01
SDT.PasTemporel = 0.1
SDT.K = 0.0001

'Initialise avec les conditions initiales et aux limites
SDT.Initialiser(ImageCI, ImageCL, 0, 100)

'Calcule 10000 pas
SDT.AvancerPlusieursPas(10000)

'Récupère la température au centre de la zone de travail
Dim TempératureFinale As Double = SDT.ObtenirTempératureFinale(25, 15)

'Affiche cette température
MsgBox("Température finale : " & _
Math.Round(TempératureFinale, 2).ToString & "°C")
```

Résultat de ce code, après environ 2 secondes de calcul :

Exemple 3 :

Cet exemple traite le phénomène d'effet de peau. Le bord gauche de la plaque est soumis à une température variable, le bord droit est maintenu à 50°C, et la température initiale est de 50°C. On veut connaître la pénétration de l'onde thermique dans le matériau.

On peut résoudre un problème unidimensionnel en utilisant une zone de travail rectangulaire, et en plaçant des parois adiabatiques de part et d'autre.

On utilise des Bitmaps de 50 × 10 pixels :

ImageCI ***ImageCL***

Le calcul est mené pas-à-pas, car il faut modifier entre chaque pas la température du bord gauche. Cette température est sinusoïdale, et vaut $T(\boldsymbol{t}) = 50 + 50 \sin(2\pi\boldsymbol{t}/100)$. Sa période est de 100 secondes, soit 1000 pas temporels.

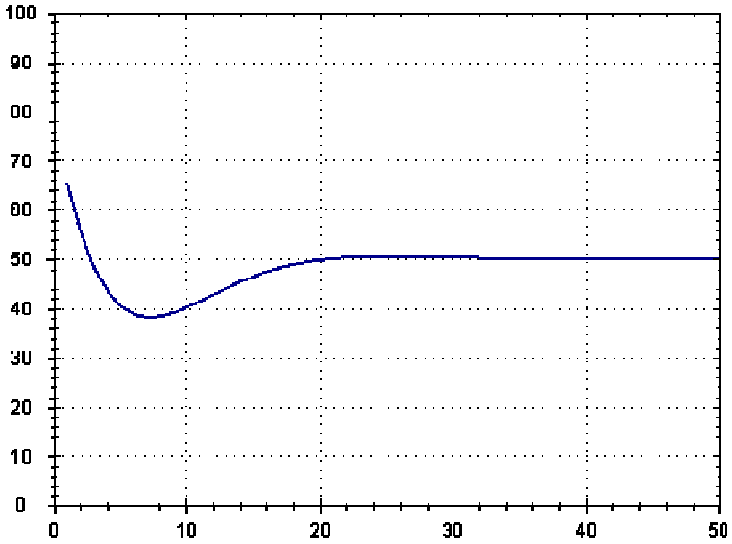
Extrait de code :

```
'Créé une instance de la classe
Dim SDT As New SimulationDiffusionThermique2D

'Définit les paramètres physiques
SDT.PasSpatial = 0.01
SDT.PasTemporel = 0.1
SDT.K = 0.0001

'Initialise avec les conditions initiales et aux limites
SDT.Initialiser(ImageCI, ImageCL, 0, 100)


'Calcule 5000 pas, soit 5 périodes
For i As Integer = 1 To 5000

    'Calcule un nouveau pas
    SDT.AvancerUnPas()

    'Modifie la température des conditions aux limites variables
    SDT.CLTempératureVariable = 50 + 50 * _
    Math.Sin(2 * Math.PI * 0.1 * i / 100)

Next i
```

On veut alors afficher le profil de températures à l'instant ***t*** = 480s.

```
'Tableau des températures au pas 4800
Dim Tableau As Double(,) = SDT.ObtenirTableau(4800)
'Tableau qui va contenir la "coupe" de températures
Dim ProfilTempératures(49) As Double

'On complète le tableau
For i As Integer = 0 To 49

    'On prend la 3ème ligne de pixels (par exemple)
    ProfilTempératures(i) = Tableau(i, 3)

Next i
```

Le tableau ***ProfilTempératures*** contient alors les données suivantes :



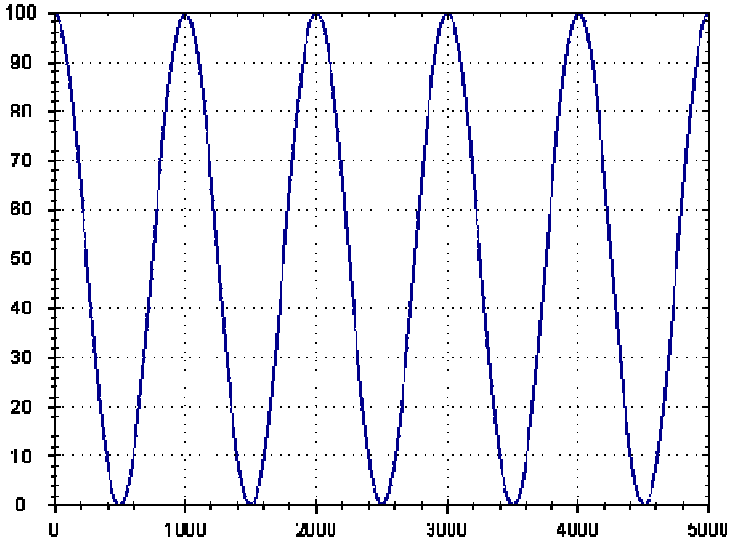
On peut récupérer le profil de températures à différents instants et réaliser une animation pour visualiser la progression (et l'atténuation) de l'onde. On observe qu'il s'agit alors d'une onde évanescente.

On peut également comparer l'évolution des températures à différents endroits, par exemple au bord et à 15cm du bord :

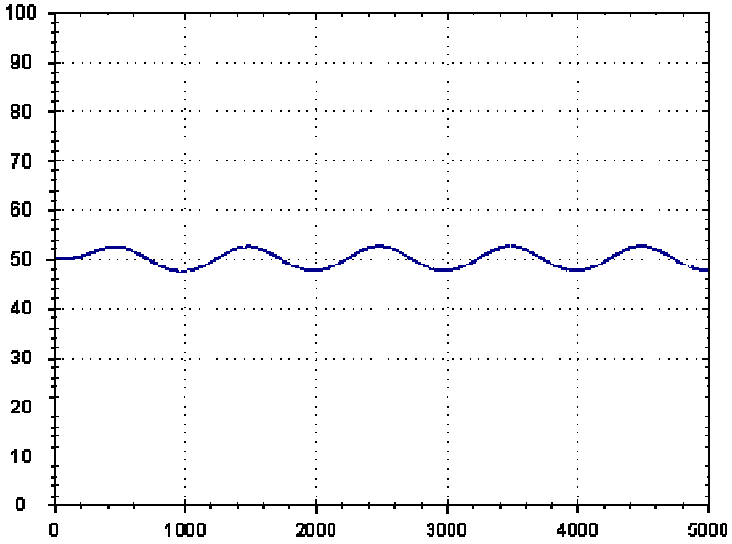
```
'Température au bord
Dim Temp1 As Double() = SDT.ObtenirHistoriqueTempérature(0, 3)

'Température à 15cm du bord
Dim Temp2 As Double() = SDT.ObtenirHistoriqueTempérature(15, 3)
```

Ces deux tableaux contiennent :

Température au bord :



Température à 15cm du bord :



La température à 15cm du bord a une amplitude fortement atténuée, mais elle est également déphasée. Elle est, à cette distance, en opposition de phase par rapport à la température du bord.

Si le matériau est le sol, ce phénomène est appelé « effet de cave ».

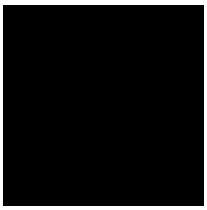
Exemple 4 :

Cet exemple simple va servir à illustrer les effets d'une non-convergence de la solution.

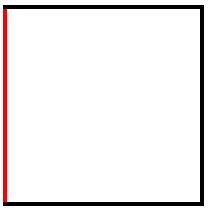
Une plaque de 50cm × 50cm a son bord gauche maintenu à 100°C, et les autres à 0°C. Sa température initiale est de 0°C.

Les Bitmaps sont donc :

ImageCI



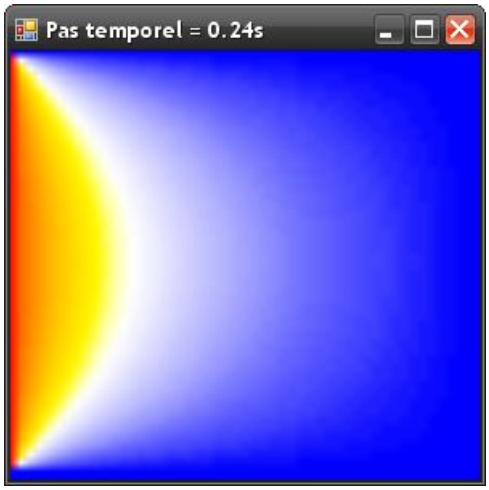
ImageCL



Premier calcul avec la solution convergente :

```
'Créé une instance de la classe
Dim SDT As New SimulationDiffusionThermique2D

'Définit les paramètres physiques en choisissant automatiquement le pas
temporel
Dim PasTemporel As Double = SDT.InitialiserPasTemporelMaxi(0.01, 0.0001)

'Initialise avec les conditions initiales et aux limites
SDT.Initialiser(ImageCI, ImageCL, 0, 100)

'Calcule 2000 pas
SDT.AvancerPlusieursPas(2000)

'Récupère l'image du dernier pas calculé
Dim bmpRésultat As Bitmap = SDT.ObtenirImageFinale()

'Affiche cette image, avec le pas temporel dans la barre de titre
Dim frmRésultat As New Windows.Forms.Form
frmRésultat.BackgroundImage = bmpRésultat
frmRésultat.BackgroundImageLayout = ImageLayout.Stretch
frmRésultat.Text = "Pas temporel = " & PasTemporel.ToString & "s"
frmRésultat.Show()
```

Résultat :



Deuxième calcul, en choisissant un pas temporel trop grand :

```
'Créé une instance de la classe
Dim SDT As New SimulationDiffusionThermique2D

'Définit les paramètres physiques
SDT.PasSpatial = 0.01
SDT.PasTemporel = 0.26
SDT.K = 0.0001

'Vérifie si cette solution est convergente
Dim TestConvergence As Boolean = SDT.SolutionConvergente

'Initialise avec les conditions initiales et aux limites
SDT.Initialiser(ImageCI, ImageCL, 0, 100)

'Calcule 400 pas
SDT.AvancerPlusieursPas(400)

'Récupère l'image après 180 pas
Dim bmpRésultat1 As Bitmap = SDT.ObtenirImage(180)
'Récupère l'image du dernier pas calculé
Dim bmpRésultat2 As Bitmap = SDT.ObtenirImageFinale()

'Affiche ces image, en indiquant la convergence dans la barre de titre
Dim frmRésultat1 As New Windows.Forms.Form
frmRésultat1.BackgroundImage = bmpRésultat1
frmRésultat1.BackgroundImageLayout = ImageLayout.Stretch
frmRésultat1.Text = "Convergence = " & TestConvergence.ToString
frmRésultat1.Show()

Dim frmRésultat2 As New Windows.Forms.Form
frmRésultat2.BackgroundImage = bmpRésultat2
frmRésultat2.BackgroundImageLayout = ImageLayout.Stretch
frmRésultat2.Show()
```

Ce code affiche les deux fenêtres suivantes :

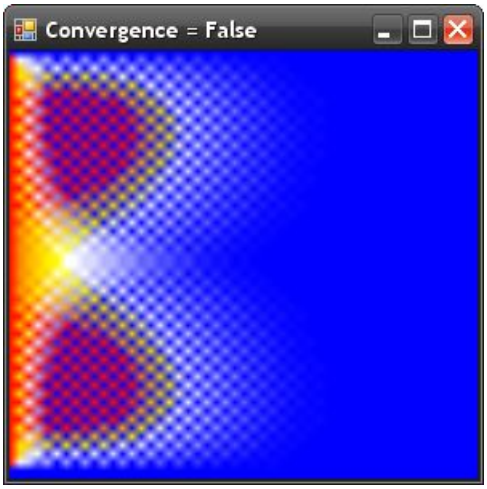


Image après 180 pas

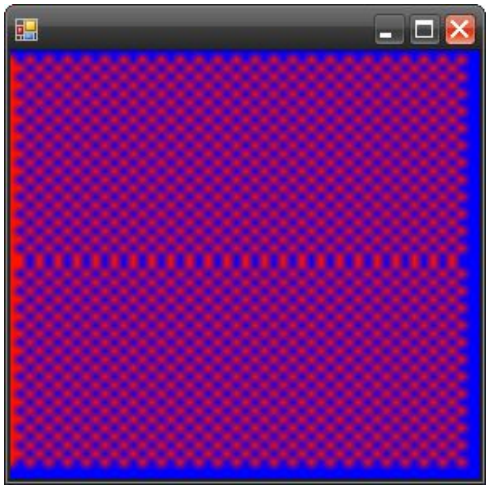
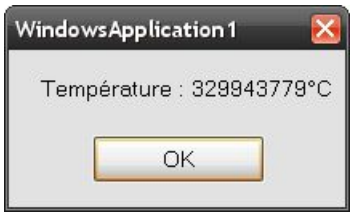


Image après 400 pas

La divergence apparaît dans les zones de fort gradient de température, puis se propage à l'ensemble de la zone de travail.

La croissance des températures est exponentielle. Par exemple, la température au point de coordonnées (9,10) vaut, après 400 pas :

```
Dim Température As Double = SDT.ObtenirTempératureFinale(9, 10)
MsgBox("Température : " & Math.Round(Température).ToString & "°C")
```



La température atteint donc rapidement des valeurs importantes. Si un trop grand nombre de pas est calculé, il y a un risque de dépassement de capacité.

Attention : Aucune protection n'est prévue contre les exceptions du type ***Overflow***. Le programmeur doit veiller à ce que la solution converge avant de lancer le calcul.