Universitat Autònoma de Barcelona

Grau en Matemàtiques

Pràctica en C
Diferències finites per equacions d'evolució.

Xavier López

26 d'abril de 2016

Índex

1 Introducció

L'objectiu d'aquesta pràctica és resoldre un problema d'evolució mitjançant diferències finites.

El problema triat és un problema mixte per l'equació de la calor sobre una regió rectangular. Concretament,

$$\begin{aligned} u_t - \Delta u &= f \quad \text{a} \quad [0,T]\times\Omega \\ u &= g \quad \text{a} \quad [0,T]\times\partial\Omega \\ u &= h \quad \text{a} \quad 0\times\partial\Omega \end{aligned} \tag{1}$$

on $\Omega = [0, L_x] \times [0, L_y]$.

Discretitzem $[0,T]$ amb pas $\delta_t = T/n_t$, $[0, L_x]$ amb pas $\delta_x = L_x/n_x$ i $[0, L_y]$ amb pas $\delta_y = L_y/n_y$. Per a $k = 0 \div n_t$, $i = 0 \div n_x$, j=$0 \div n_y$. Denotem per $U_{k,i,j}$ les aproximacions que obtindrem de $u(k_t, i\delta_x, j\delta_y)$. Imposarem les condicions de frontera de (1) prenent:

$$\begin{aligned} &U_{0,i,j} = h(i\delta_x, j\delta_y), \qquad i = 0 \div n_x,\ j = 0 \div n_y \\ &U_{k,i,0} = g(k\delta_t, i\delta_x, 0), \quad U_{k,i,n_y} = g(k\delta_t, i\delta_x, L_y), \quad k = 1 \div n_t,\ i = 0 \div n_x, \\ &U_{k,0,j} = g(k\delta_t, 0, j\delta_y), \quad U_{k,i,n_y} = g(k\delta_t, Lx, j\delta_y), \quad k = 1 \div n_t,\ j = 1 \div n_y - 1, \end{aligned} \tag{2}$$

Per a omplir els $U_{k+1,i,j}$ usarem dos mètodes diferents.

- Diferències finites endavant de primer orde en temps i centrades de segon ordre en l'espai.
- Crank-Nicolson amb sobrerelaxació.

El motiu pel que fem servir dos mètodes diferents, és que per a tenir convergència en diferències finites endevant de primer ordre en el temps ens obliga a pendre δ_t molt petits respecte dels passos δ_x, δ_y per a tenir bones aproximacions.

L'aventatge de Crank-Nicolson amb sobrerelaxació que justifica la seva implementació és que podem pendre passos en el temps del mateix ordre que en l'espai

2 Esquema explícit en diferències finites

En prendre diferències finites endavant de primer ordre en el temps i centrades de segon ordre en l'espai a l'equació de la calor (1), obtenim

$$\begin{aligned} U_{k+1,i,j} =& (1 - 2\mu_x - 2\mu_y)U_{k,i,j} \\ &+ \mu_x(U_{k,i+1,j} + U_{k,i-1,j}) + \mu_y(U_{k,i,j+1} + U_{k,i,j-1}) \\ &+ \delta_t f_{k,i,j} \end{aligned} \tag{3}$$

on $\mu_x = \delta_t/\delta_x^2$, $\mu_y = \delta_t/\delta_y^2$, $U_{k,i,j} \sim u(k\delta_t, i\delta_x, j\delta_y)$ i $f_{k,i,j} = f(k\delta_t, i\delta_x, j\delta_y)$.

L'equació anterior és un *esquema explícit en diferències finites* perquè permet obtenir recurrentment les aproximacions $\{U_{k,i,j}\}_{k=0\div T, i=0\div L_x, j=0\div L_y}$ de la següent forma:

Algorisme 1.1

Omplir $\{U_{0,i,j}\}_{i=0\div n_x, j=1\div n_y-1}$ a partir de (2);
for $\forall k = 0 \div n_t - 1$ **do**
  Omplir $\{U_{k+1,i,0}, U_{k+1,i,n_y}, U_{k+1,0,j}, U_{k+1,n_x,j}\}_{i=0\div n_x, j=1\div n_y-1}$ a partir de (2);
  for $\forall i = 1 \div n_x - 1$ **do**
    for $\forall j = 1 \div n_y - 1$ **do**
      Omplir $U_{k+1,i,j}$ a partir de (3).
    end
  end
end

Aquest mètode de resolució numèrica del problem (1) és atractiu per la seva simplicitat, però presenta dos inconvenients:

- Per assegurar convergència de l'aproximació obtinguda cap a la solució exacta, és a dir, perquè
$$U_{k,i,j} - u(k\delta_t, i\delta_x, j\delta_y) \xrightarrow[\delta_t,\delta_x,\delta_y\to 0]{} 0$$
cal imposar la condició:
$$1 - 2\mu_x - 2\mu_y \geq 0 \tag{4}$$
- Suposant que satisfà la condició (4), l'aproximació obtinguda és quaràtica en l'espai però lineal en el temps. Concretament,
$$|U_{k,i,j} - u(k\delta_t, i\delta_x, j\delta_y)| = O(\delta_t + \delta_x^2 + \delta_y^2).$$
Això ens obliga a prendre passos δ_t molt petits respecte dels passos δ_x,δ_y, per a tenir bones aproximacions.

2.1 Implementació

La implementació de l'esquema en diferències finites consisteix en:

- `grRDF.c`
- `grRDF.h`

2.2 Validació

Per tal de verificar la correcció de la implementació anterior, resolem el problema (1) en un cas en què sabem que hem trobat la solució exacta. Per exemple, triem un polinomi de grau 1 en t i grau ≤ 3 en x i y, llavors trobem f, g, i h tals que u sigui solució de (1).

La implementació de la verificació està al fitxer `polExpl.c`, on apliquem l'esquema en diferències finites que he implementat i comprovem que la solució és exacte per al polinomi $u(t,x,y) = (1+x+y)t$.
Nota que per aquest polinomi $f = (1+x+y)$, $g = u$, i $h = 0$.

Per a executar el programa, és a dir, omplir la graella, podem triar per exemple $n_t = 1000$, $n_x = 10$, $n_y = 10$, $T = 1$, $L_x = 1$ i $L_y = 1$, ja que compleixen la condició de convergència (4).

Per a compilar `polExpl.c`, escribim a la terminal, a la direcció on es troba el fitxer:

```
$ make polExpl
```

I per executar

```
$ ./polExpl 1000 10 10 1 1 1
```

La rutina crea un fitxer `polExpl.txt` on guarda la graella per a cada iterat, i ens torna per *standard output* l'error màxim i la condició de convergència per aquests paràmetres:

```
L'error màxim és 0.000000
Passa la condició de convergència 0.6 >0
```

Nota que si canviem el polinomi, l'error no necessàriament és zero, per exemple si usem

$$u(t,x,y) = 1 + x + y + x^2 + y^2 + t(1 + x + y + x^2 + y^2)$$

hi ha error d'arrodoniment, per aquest polinomi u tenim

$$\begin{aligned} f &= 1 + x + y + x^2 + y^2 - 4t - 4 \\ g &= u \\ h &= 1 + x + y + x^2 + y^2 \end{aligned}$$

Per fer la implementació per aquest polinomi cal canviar les funcions `f_pol` , `g_pol`, `h_pol` del fitxer `polExpl.c` (Els canvis necessaris estan comentats dins del fitxer).

Compilant i executant de nou tenim el següent output:

```
L'error màxim és 0.000292
Passa la condició de convergència 0.6 >0
```

L'error és prou petit per a que puguem validar la implementació de l'esquema explícit en diferències finites.

3 Crank-Nicolson mitjançant SOR

El problema de la no convergència es pot solucionar discretitzant l'equació de la calor en el temps mitjançant una diferència finita endarrera de primer ordre, en comptes d'endavant, de manera que obtindríem un *esquema implícit.*

Existeix una variant d'aquesta estratègia, coneguda com mètode de Crank-Nicolson, que consisteix en promitjar les discretitzacions espacials en els instants de temps $k\delta_t$ i $(k+1)\delta_t$. Concretament,

$$\frac{U_{k+1,i,j} - U_{k,i,j}}{\delta_t} - \frac{1}{2}\left(\frac{U_{k+1,i+1,j} - 2U_{k+1,i,j} + U_{k+1,i-1,j}}{\delta_x^2} + \frac{U_{k,i+1,j} - 2U_{k,i,j} + U_{k,i,j-1}}{\delta_y^2}\right) = \frac{1}{2}(f_{k+1,i,j} + f_{k,i,j}) \tag{5}$$

Si ara volguéssim fer l'anàl·leg de l'algorisme 1.1, suposant que som al pas k i coneixem $\{U_{k,i,j}\}_{i=0\div n_x, j=0\div n_y}$, l'equació (5) no ens permet trobar $U_{k+1,i,j}$ a partir dels valors del pas k, donat que apareixen simultàniament els valors $\{U_{k,i,j}, U_{k,i\pm1,j}, U_{k,i,j\pm1}\}$ i $\{U_{k+1,i,j}, U_{k+1,i\pm1,j}, U_{k+1,i,j\pm1}\}$.

Variant $j = 1\div n_y - 1$, $i = 1\div n_x - 1$, l'equació (5) és un sistema lineal en $\{U_{k+1,i,j}\}_{j=1\div n_y-1, i=1\div n_x-1}$. Aleshores, amb les mateixes notacions de (3), el mètode de sobrerelaxació amb paràmetre ω per a aquest sistema es pot descriure com segueix: prenent $\{U^{(0)}_{k+1,i,j}\}_{i,j}$ aproximació inicial

de la solució de (5), i per a $m \geq 0$, calculem $\{U^{(m+1)}_{k+1,i,j}\}_{i,j}$ a partir de $\{U^{(m)}_{k+1,i,j}\}_{i,j}$ mitjançant

for $\forall j = 1, \ldots, n_y - 1$ **do**
  for $\forall i = 1, \ldots, n_x - 1$ **do**

$$\begin{aligned} U^{(m+1)}_{k+1,i,j} = (1-\omega)U^{(m)}_{k+1,i,j} + \frac{\omega}{1+\mu_x+\mu_y}\,[(1-\mu_x-\mu_y)U_{k,i,j} \\ + \frac{\mu_x}{2}\left(U^{(m)}_{k+1,i+1,j} + U^{(m+1)}_{k,i-1,j} + U_{k,i+1,j} + U_{k,i-1,j}\right) \\ + \frac{\mu_y}{2}\left(U^{(m)}_{k+1,i,j+1} + U^{(m+1)}_{k,i,j-1} + U_{k,i,j+1} + U_{k,i,j-1}\right) \\ + \frac{\delta_t}{2}(f_{k+1,i,j} + f_{k,i,j}) \end{aligned} \tag{6}$$

  end
end

Com aproximació inicial $\{U^{(0)}_{k+1,i,j}\}_{i,j}$ podem prendre els valors $\{U_{k,i,j}\}_{i,j}$ determinats al pas de temps anterior.

Amb tot això, podem descriure algorísticament *l'esquema imiplícit de Crank-Nicolson per a l'equació de la calor* com segueix:

Algorisme 1.2:

Omplir $\{U_{0,i,j}\}_{i=0\div n_x, j=1\div n_y-1}$ a partir de (2);
for $\forall k = 0 \div nt - 1$ **do**
  Omplir $\{U_{k+1,i,0}, U_{k+1,i,n_y}, U_{k+1,0,j}, U_{k+1,n_x,j}\}_{i=0\div n_x, j=1\div n_y-1}$ a partir de (2);
  for $\forall i = 1 \div n_x - 1$ **do**
    for $\forall j = 1 \div n_y - 1$ **do**
      Omplir $U_{k+1,i,j}$ a partir de (6).
    end
  end
end

Amb aquest procediment, a més d'evitar la necessitat d'imposar la condició (4) per a tenir convergència, obtenim també que l'error és quadràtic tant en el pas temporal com en els passos espacials. Concretament

$$|u(k\delta_t, i\delta_x) - U_{k,i,j}| = O(\delta_t^2 + \delta_x^2 + \delta_y^2)$$

de manera que podem prendre passos en el temps del mateix ordre que en l'espai.

3.1 Implementació

La implementació és essencialment la mateixa que en l'esquema explícit en diferències finites, la diferència fonamental és que en lloc de la funció `void grRDF_pasCalExpl(...)` usem `int grRDF_pasCalCN(...)`, i per tant necessitem dos paràmetres nous: `maxit` i `tol` per a la sobrerelaxació.

Nota que la funció que usem per a fer un pas en temps és de tipus `int`, en lloc d'una de tipus `void`, ja que ens interessarà saber el nombre de passos a la sobrerelaxació (recorda que en diferències finites no usem sombrerelaxació).

De la mateixa manera, la implementació està a:

- `grRDF.c`
- `grRDF.h`

3.2 Validació

Per a fer la validació ho fem de la mateixa forma que per validar l'implementació en diferències finites, triem un polinomi i comprovem que només hi ha error d'arrodoniment en la graella.

La validació de Crank-Nicolson, usant exl polinomi $u = (1 + x + y)t$, està a `polCN.c`.

Compilem amb:

```
$ make polCN
```

Podem executar així:

```
$ ./polCN 50 50 50 1 1 1 1e-6 100
```

un cop executada, la rutina crea un fitxer `polCN.c`, on guarda la graella i ens torna per *standard output*:

```
L'error màxim és 0.076237
```

Per tant, condiserem la implementació verificada.

4 Placa d'acer sota una font de calor

Suposem que $\Omega = [0,1] \times [0,1]$ representa una placa d'acer molt prima, la qual posem una font de calor que genera $F(\tau, x, y)$ cal/(s · cm^3) a l'instant τ (en s), i al punt (x, y) (en cm), on F està definida per

$$f(n) = \begin{cases} 100 & \text{si } \|(x,y) - (0.5, 0.5)\|_2 < 0.2 \\ 0 & \text{altrament} \end{cases}$$

Suposem també que, per $\tau = 0$, la placa es troba a 0°C, i per tot temps τ la vora de la placa es manté també a $0°C$.

L'evolució de la temperatura sobre el domini ω des de l'instant $\tau = 0$ fins a l'instant $\tau = S$ (en segons) ve donada per la solució $v(\tau, x, y)$ del problema mixte:

$$\begin{aligned} c\rho v_\tau - \kappa \Delta v = F \quad & \text{a} \quad [0, S] \times \Omega \\ v = G \quad & \text{a} \quad [0, S] \times \partial\Omega, \\ v = H \quad & a \quad \{0\} \times \Omega \end{aligned} \tag{7}$$

on c és la calor específica del material, ρ és la seva densitat i κ la seva conductivitat tèrmica, suposant que totes tres són constants. Per a la placa d'acer, els valors són

$$\kappa = 0.13 \frac{\text{cal}}{\text{s} \cdot \text{cm} \cdot^\circ \text{C}}$$

Nota que el canvi de temps donat per

$$u(t, x, y) = v(\tau, x, y) \quad \text{per} \quad t = \frac{\kappa}{c\rho}\tau,$$

permet transformar el problema (7) en el problema (1), per a

$$f(t, x, y) = \frac{1}{\kappa} F(\frac{c\rho}{\kappa}, x, y), \quad g(t, x, y) = G(\frac{c\rho}{\kappa}, x, t), \quad h(x, y) = H(x, y)$$

4.1 Implementació en diferències finites

La implementació en diferències finites de l'evolució de la placa d'acer sota una font de calor consisteix en:

- `grRDF.c`
- `grRDF.h`
- `placaExpl.c`

4.1.1 Resultats

Compilem amb:

```
make placaExpl
```

Nota que si prenem passos δ_t, δ_x, δ_y que no satisfan la condició (4) i apliquem diferències finites per equacions d'evolució, obtenim solucions físicament sense sentit, per exemple, executant amb

```
./placaExpl 50 50 50 1 1 1
```

Ens genera el fitxer `placaExpl.txt`. Nota que en aquest fitxer, on hi ha la solució per a la graella, té solucions sense sentit, ja que té valors negatius de mòdul gran, a més, aquests punts de la graella estan al costat de punts amb valor positiu de mòdul semblant.

Per exemple, algunes línies del fitxer amb solució sense sentit són:

```
0.240000 0.240000 208921466427052.750000
0.240000 0.260000 -4525143665298961.000000
0.240000 0.280000 43172047953805768.000000
0.240000 0.300000 -233391739214530496.000000
```

Així doncs, hem confirmat que no és cap sorpresa obtenir solucions sense sentit físic si no complim la condició de convergència del mètode de diferències finites.

D'altra banda, nota que complir la condició de convergència suposa una complicació rellevant si volem passos petits en x i y, ja que ens obliga a pendre passos molt petits en el temps, i per tant, realitzar molts iterats.

Podem evitar aquesta compliació implementant *Crank-Nicolson* amb sobrerelaxació.

4.2 Implementació en Crank-Nicolson

La implementació de Crank-Nicolson amb sobrerelaxació sobre l'evolució de la placa d'acer sota una font de calor consisteix en:

- `grRDF.c`
- `grRDF.h`
- `placaCN.c`

4.2.1 Resultats

Compilem amb:

```
make placaCN
```

I si volem $n_t = n_x = n_y = 100$ i $\tau = L_x = L_y = 1$ executem:

```
$ ./placaCN 100 100 100 1 1 1 1e-6 1000
```

Que ens genera el fixer `placaCN.txt` on hi ha la solució de la graella per cada pas en temps.

La visualització de la solució de la graella la fem mitjançant *gnuplot*, està implementada al fitxer `placaCN.gnu`. Per a executar la implementació cal obrir *gnuplot* i carregar el fitxer:

```
$ gnuplot

gnuplot> load 'placaCN.gnu'
```

Aquestes comandes generen els fitxers: `t0.eps`, `t006.eps`, `t025.eps`, `t062.eps`, `t1.eps`. Aquets fitxers es corresponenn amb les següents imatges:

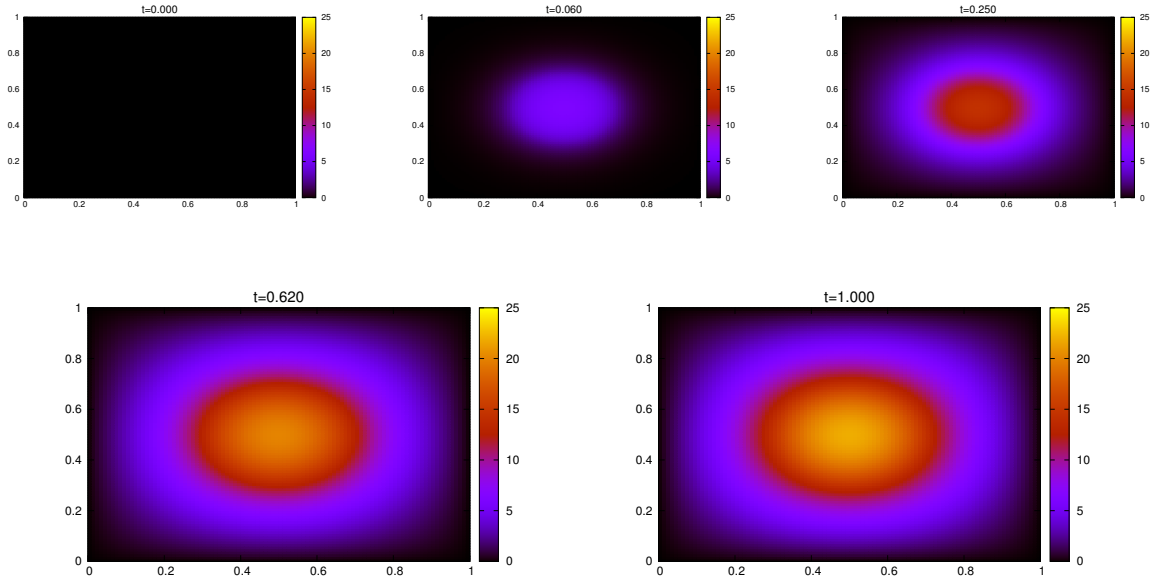

Figura 1: A figure with two subfigures

Nota que per a tenir aquests gràfics cal primer haver creat el fitxer `placaCN.txt` mitjançant la comanda

```
$ ./placaCN 100 100 100 1 1 1 1e-6 1000
```

Pots fer gàfics diferents (de la solució en diferents interval de temps, o per a una execució de `placaCN` amb paràmetres diferents), de la següent manera:

```
$ ./placaCN // crida amb els paràmetres que desitgis
$gnuplot
```

i dins de gnuplot

```
gnuplot> #gràfics projectats amb colors
gnuplot> unset hidden3d
gnuplot> set pm3d
gnuplot> unset surface
gnuplot> set view map

gnuplot> splot 'placaCN.txt' index w l
```

5 Codi

El codi complert d'aquesta pràctica consisteix en:

- `Makefile`
- `grRDF.c`
- `grRDF.h`
- `polExpl.c`
- `polCN.c`
- `placaExpl.c`
- `placaCN.c`
- `placaCN.gnu`

Per a compilar el codi cal escriure a la terminal:

```
$ make all
```

o si volem compilar només un fitxer:

```
$ make <nom_fitxer>
```

5.1 Trampes del C

Formalment aquest apartat no té interés, però pel mal de cap que m'ha causat li dedico la secció final de la pràctica per a recordar l'error d'implementació següent:

En `C`, si volem implementar macros, convé no oblidar la necessitat d'usar parèntesis, per exemple, si definim la macro

```
define U(i,j) gr->u[j*(1+gr->nx)+i]
```

sembla que estigui ben implementada, pero no ho està, ja que si cridem `U(1,2+3)` s'interpreta com `U[2+3*(1+gr->nx)+1]`, pero el que volem és `U[(2+3)*(1+gr->nx)+1]`, per tant la implementació correcta necessita parèntesis:

```
define U(i, j) gr->u[(gr->nx + 1)*(j) + (i)]
```