A kétdimenziós hővezetési egyenlet megoldása négyzetlapon

Modern numerikus módszerek a fizikában beadandó projekt

Novák Martin István

Eötvös Loránd Tudományegyetem

Fizikus MSc - Kutatófizikus szakirány

2017. június 27.

1. Bevezetés

A fizika a világban végbemenő jelenségek leírásával foglalkozó tudomány. Mint a legtöbb természettudománynak, a fizika nyelve is a matematika, a jelenségeket legtöbbször differenciálegyenletekkel írjuk le. Ha egy ilyen egyenletben csak egy darab változó deriváltja szerepel, akkor azt közönséges differenciálegyenletnek, ha több változóé, akkor pedig parciális differenciálegyenletnek *(PDE)* nevezzük.

A differenciálegyenletek *(főképp a parciálisaké)* megoldása teljes általánosságukban általában rendkívül nehéz feladat, sőt, sokszor nincs is zárt alakban megoldásuk. Többek között emiatt sokszor numerikus módszerekhez kell folyamodni. Jelen projektben én is egy ilyen PDE, a kétdimenziós hővezetési egyenlet megoldását kísérlem meg egy általam, C++ nyelven fejlesztett programmal egy közönséges négyzetrácson.

A hővezetési egyenlet az úgynevezett parabolikus PDE-k családjába tartozik, melyek másodrendű PDE-k, általános formájuk pedig[1]:

$$A\frac{\partial^2 u}{\partial x^2}(x,y) + 2B\frac{\partial^2 u}{\partial x \partial y}(x,y) + C\frac{\partial^2 u}{\partial y^2}(x,y) + D\frac{\partial u}{\partial x}(x,y) + E\frac{\partial u}{\partial y}(x,y) + E = 0 \tag{1}$$

Az eq. (1) egyenletben az A, B, és a C együtthatóknak ki kell elégítenie a következő relációt is:

$$B^2 - AC = 0 \tag{2}$$

Az $y = t$, $A = k$, $E = 1$ illetve $B = C = D = F = 0$ választással az egydimenziós hővezetési egyenletet kapjuk vissza:

$$k\frac{\partial^2 u}{\partial x^2}(x,t) = \frac{\partial u}{\partial t}(x,t) \tag{3}$$

Ehhez nagyon hasonló a kétdimenziós egyenlet, ahol megjelenik még egy térbeli koordináta második deriváltja:

$$k\left(\frac{\partial^2 u}{\partial x^2}(x,y,t) + \frac{\partial^2 u}{\partial y^2}(x,y,t)\right) = \frac{\partial u}{\partial t}(x,y,t) \tag{4}$$

A parabolikus PDE-k numerikus megoldására kifejlesztett egyik legnépszerűbb eljárás az úgynevezett Foward-Time Central-Space *(FTCS)* séma. Munkám során én is ezt használtam. Ez az eljárás egy elsőrendű közelítést felhasználó eljárás. Az idő változóban ahogy a neve is mutatja az Euler-féle "előre léptetett" differenciahányados lép a derivált helyébe, míg a térbeli változókban egy középponti differenciahányadossal dolgozunk a derivált helyett. Példaképp lássuk az egydimenziós majd a kétdimenziós hővezetési egyenlet diszkretizált változatát:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{k}{\Delta x^2}\left(u_{i+1}^n - 2u_i^n + u_{i-1}^n\right) \tag{5}$$

Az eq. (5) egyenletben az u függvény felső indexe az n. időlépést jelöli, míg az alsó index az i. rácsponton vett értéket. Az algoritmus stabil *(konvergálni fog)*, ha a

$$\frac{k\Delta t}{\Delta x^2} \leq \frac{1}{2} \tag{6}$$

kifejezés teljesül. Teljesen analóg módon kétdimenzióban a következőt írhatjuk:

$$\frac{u_{i,j}^{n+1} - u_{i,j}^n}{\Delta t} = \frac{k_x}{\Delta x^2}\left(u_{i+1,j}^n - 2u_{i,j}^n + u_{i-1,j}^n\right) + \frac{k_y}{\Delta y^2}\left(u_{i,j+1}^n - 2u_{i,j}^n + u_{i,j-1}^n\right) \tag{7}$$

Itt i az x térbeli irányt, j pedig az y térbeli irányt indexeli. A stabilitás feltétele pedig:

$$\frac{k_x\Delta t}{\Delta x^2} + \frac{k_y\Delta t}{\Delta y^2} \leq \frac{1}{2} \tag{8}$$

Ha izotrop hővezetést tételezünk fel, azaz $k_x = k_y = k$ valamint a használt rácsot négyzetrácsnak választjuk azaz $\Delta x = \Delta y$, akkor a kritérium a következő formára redukálódik:

$$\frac{k\Delta t}{\Delta x} \leq \frac{1}{4} \tag{9}$$

2. A tényleges feladat

A projekt során a konkrét feladatom az volt, hogy oldjam meg egy négyzetlap hővezetési problémáját*(emiatt használok sima négyzetrácsot, mint diszkretizációt, hiszen ez tökéletesen illeszkedik a probléma geometriájához)* a következő kitételekkel:

- A négyzetlap bal szélén egy lineáris hőmérsékletprofil van jelen
- A négyzetlap alján egy állandó hőáram van
- A négyzetlap jobb széle és teteje tökéletes hőszigetelő
- A kezdőfeltétel a lap mentén mindkét koordinátában lineáris és a bal oldali határfeltételt is kielégíti*(az ábrákon ez nem teljesül egy már ismert hiba miatt, a kódban javítottam, de túl sok munka lett volna újra elkészíteni az összes ábrát egy ilyen apró hiba miatt)*

3. A program felépítése, működése

A program forráskódja három fájlból áll: *main.cpp*, *Grid.h*, *Grid.cpp*. A *main.cpp* végzi a "globális" *(nem a szokásos értelemben globális, de a megoldás szempontjából igen)* paraméterek kezelését, mint pl. annak a megadása, hogy mennyi időegységig fusson a szimuláció, vagy hővezetési együttható és az alsó oldalon levő állandó hőáram megadása is itt történik.

A program a megadott t_{fin}-re kiszámolja a stabilitási kritérium által megszabott maximális időlépésközt, ha Δx-et megadjuk. Mivel a program animációt is tud készíteni, ezért bevezettem továbbá egy saját "időskálát" is, hogy a különböző hővezetési együtthatóval futtatott szimulációkat ugyanolyan időskálán tudjam animálni. Erre azért van szükség, mivel ha növeljük a hővezetési együtthatót, az a Δt csökkenéséhez vezet, ami pedig végső soron azt eredményezi, hogy a t_fin idő elérése csak jóval több lépésben történhet meg. Emiatt viszont az a naív animálási technika, hogy pl. minden 25. lépést animálok, és egy másodperc alatt 25 képkockát mutatok be azt fogja eredményezni, hogy a nagyobb hővezetési együtthatóval futtatott szimuláció vizualizációja sokkal lassabb folyamatot prezentál*(mivel több az animálandó lépés)*, mint amilyen a folyamat valójában, így ha nem gondolunk bele, akkor a valójában gyorsabb folyamat lassabbnak fog tűnni.

Ennek kiküszöbölésére az egységnyi időlépésköznek, Δt_0-nak a $k = 1$ hővezetési együtthatóhoz tartozó lépésközt vettem*(ez $\Delta x = 1$ esetén $\Delta t_0 = 0.25$)*. Ezt leosztottam az aktuális futtatáshoz szükséges ténylegesen használt Δt-vel, majd megszoroztam 50-el*(így pl. a $k = 1$-es futtatás során minden 50. lépés került az animációba)*. Ez meghatározza azt, hogy egy futtatás mennyivel tartalmaz több vagy kevesebb lépést, mint a standard $k = 1$-es futtatás, így azt is, hogy mennyivel kell felgyorsítani az animációt. Tehát például, ha $k = 10$, akkor:

$$s = \frac{\Delta t_0}{\Delta t}50 = 50\frac{0.25}{\frac{\Delta x^2}{4*k}} = 50\frac{0.25}{\frac{1}{40}} = 50 \cdot 0.25 \cdot 40 = 500 \tag{10}$$

Tehát ahhoz, hogy ugyanolyan időskálán animáljunk, a $k = 10$-es esetben minden 50. helyett csak minden 500. lépést kell az animációhoz fűzni.*(most így leírva ezt eléggé elbonyolítottam...)*

Ezután a program beállítja a rácson a kezdőfeltételt, amit hogy kielégítse a bal szélső határfeltételt

$$u^0_{i,j} = i + j \tag{11}$$

módon választottam meg. Itt újra felhívom a figyelmet, hogy az ábráimon ez nem teljesül, mivel a program tesztelése közben a bal szélső határfeltételt $u^n_{i,0} = 2 \cdot i$-re állítottam, amit csak utólag vettem észre, hogy így maradt. Ha simán csak $u^n_{i,0} = i$ lenne, akkor minden szép és folytonos

lenne a négyzetlapon, így azonban kezdetben itt egy hatalmas hőmérséklet gradiens van, de ahogy megfigyeltem, ez nem okozott műtermékeket illetve instabilitásokat a szimulációban, mivel utólag próbáltam validálni az eredményeimet a COMSOL Multiphysics végeselem modellező programmal, és kvalitatíve megegyeztek az eredmények*(kvantitatívan természetesen nem hiszen a COMSOL-ban tényleges, valós anyagi paramétereket használtam, mint pl. a réz hővezetési együtthatója)*.

A program lelke a *Grid.cpp*-ben nyugszik, ugyanis minden lényeges függvény ebben van. Ennek az osztálynak a példányosításakor kell megadnunk a fent említett változók mellett még a rács méretét is, majd az objektum *solve* függvényével elindíthatjuk a megoldást a definiált négyzetrácson, az *anim* függvénnyel pedig egyrészt készíthetünk egy MPEG formátumú videót a rendszer időfejlődéséről*(itt a színskála maximuma a teljes időfejlődés alatti globális hőmérséklet maximum)*, másrészt az animációba bekerülő állapotok kiíródnak egy szövegfájlba, amiket később tetszésünk szerint ábrázolhatunk a kedvenc vizualizációs módszerünkkel.

A megoldás során létre kell hozni egy segédrácsot is és ennek a rácsnak az értékeit frissíteni az eredeti rácson számított deriváltak alapján, majd ha végigértünk a teljes rácson, akkor az eredeti rács értékeit egyenlővé tenni a segédrács értékeivel. Erre azért van szükség, hogy elkerüljük azt az effektust, hogy a differenciahányados számításakor az egyik oldali érték már frissített*(természetesen pont emiatt a hatás miatt nem jól)*, a másik oldalról származó érték pedig még az adott lépésben frissítetlen.

A programban a *solve* függvény az osztály két privát tagfüggvényét használja, a *stepInside* és a *stepBoundary* függvényt. Ezeknek a neve magáért beszél: az első a rács belsejében számolja a differenciahányadosokat, míg a második a határfeltételek kezelését biztosítja. A függvény az megadott animációs sebességgel egy vektor változót feltölt az animálni kívánt lépésekkel, amit később az *anim* függvény meghívásával ténylegesen látható eredménnyé alakíthatunk.

4. Eredmények

Négy darab futtatást csináltam, a következő paraméterpárokkal:

- $(k = 1, q = 1)$
- $(k = 10, q = 1)$

Valójában futtattam $q = 10$ értékkel is szimulációkat, de az annyira nem tűnt érdekesnek. Annyiban volt más, hogy idővel teljesen az alsó hőáram kezdte dominálni a rendszer hőmérséklet eloszlását, kivéve természetesen a bal szélen.

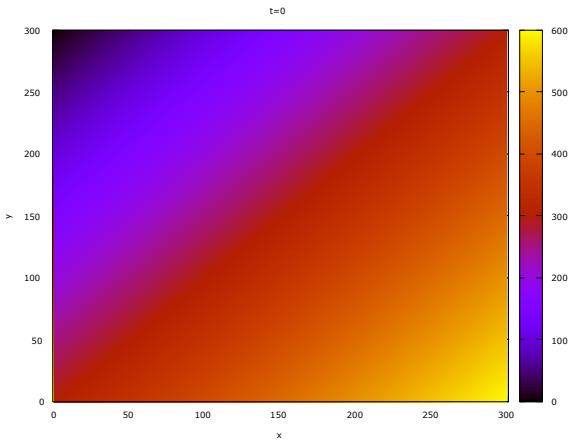
Minden szimuláció $t_{fin} = 2000$ időegységig futott, az animációk pedig a fent említett időskálán készültek. A jegyzőkönyvben való prezentálhatóság miatt néhány állapotot gnuplottal ábrázoltam is, ezek az ábrajegyzékben találhatók. A kezdeti állapotot csak egyszer tettem az ábrajegyzékbe, hiszen az minden futásnál ugyanaz volt.

A fig. 2 ábrákon azt látjuk, hogy $k = 1$ mellett, a folyamat nagyon lassan, de tart valamiféle egyensúlyi állapot felé. Kezdetben a "hideg" rész még erősen "V" alakú, aztán ez lekerekedik és elkezd felmelegedni a rendszer bal oldala is, amennyire a határfeltétel engedi.

A fig. 3 ábrákon pedig azt láthatjuk, mikor a hővezetési együtthatót tízszeresére növeltem. Látszik, hogy ebben az esetben már nagyjából 50 időegység alatt beáll az az állapot, amit az előző szimulációban csak 500 időegység körül értünk el. Ez persze nem is meglepő, hiszen az együttható lineárisan szerepel az egyenletben. Alapvetően itt is megfigyelhetjük azt, hogy a rendszer törekszik egy egyensúlyi hőmérséklet eloszlás felé, hiszen az utolsó 1000 időlépés alatt már alig változott a hőmérséklet.

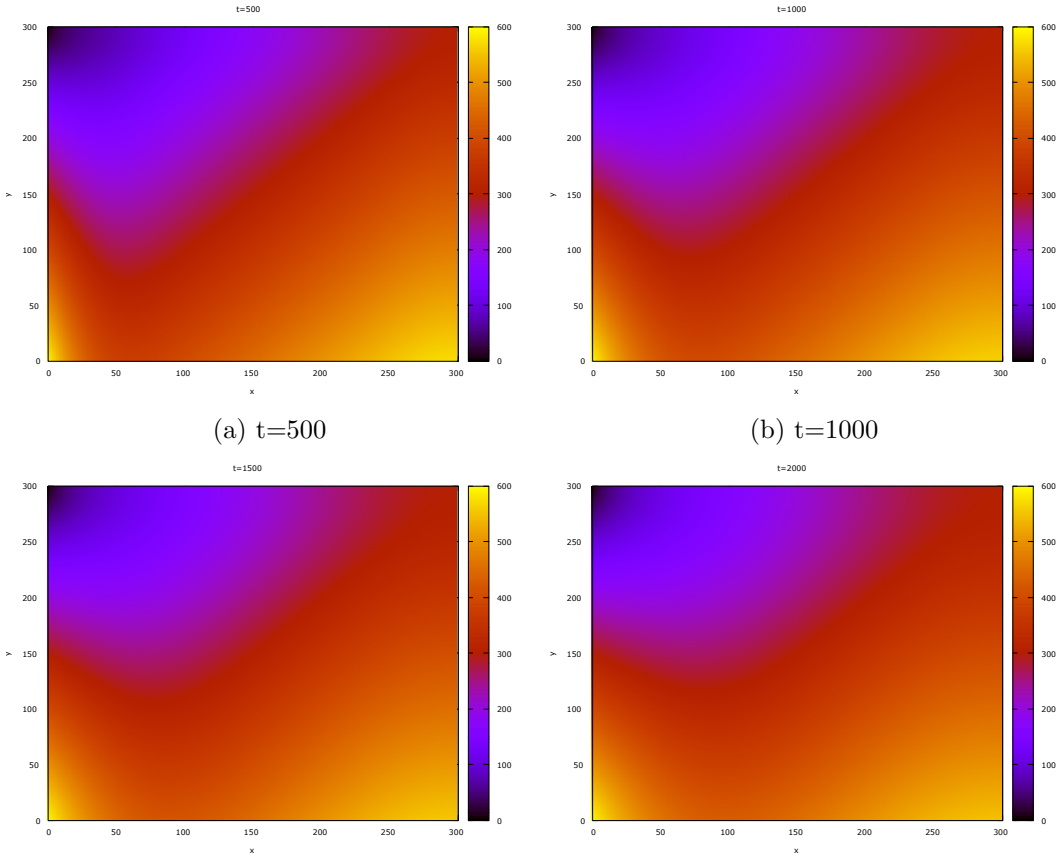
5. Összefoglalás

A projekt elkészítése során gyakorlatban is megismerkedtem a parabolikus PDE-k egy megoldási módszerével. Felelevenítettem C++ ismereteimet, mivel ezt a programnyelvet régen egész kifinomultan tudtam használni, de az idők során áttértem más nyelvre, így sajnos a C++ tudásom és az alacsonyabb szintű programnyelvekhez való szemléletmódom eléggé beporosodott. Valamennyire ezt sikerült a projekt során visszanyernem, továbbá hasznos ismeretekkel gazdagodtam a vizualizációs megoldásom elkészítése során.



1. ábra. t = 0

6. Ábrajegyzék



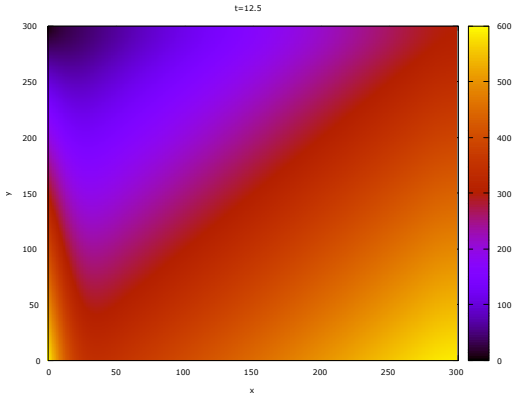
(a) t=500

(b) t=1000

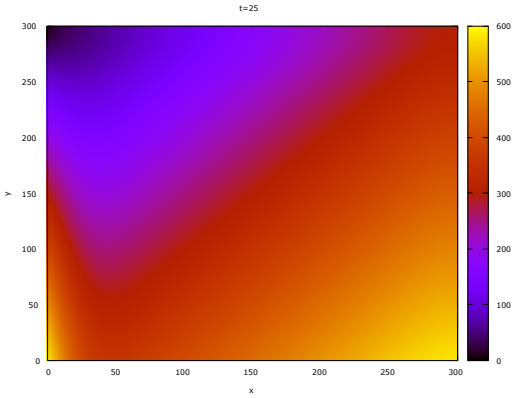
(c) t=1500

(d) t=2000

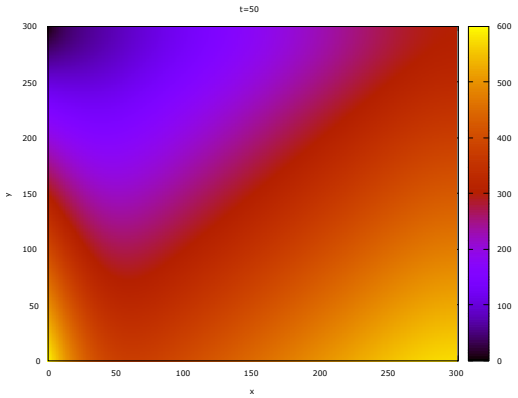
2. ábra. $k = 1$, $q = 1$



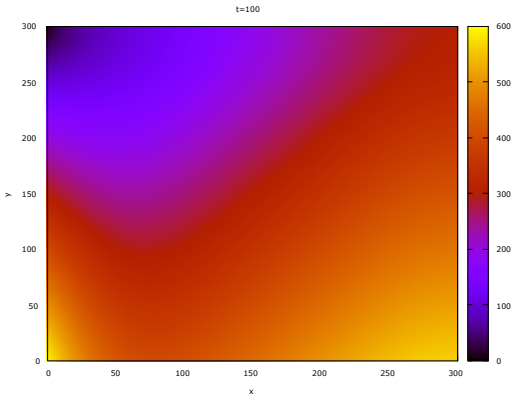
(a) t=12.5



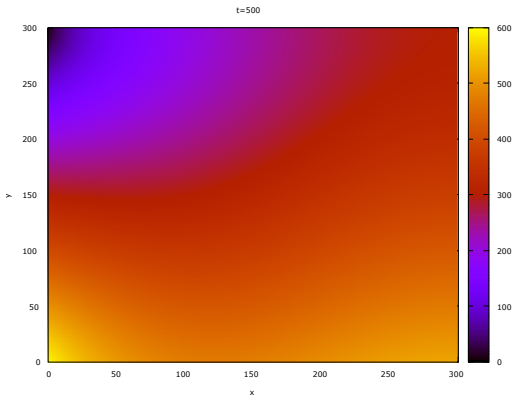
(b) t=25



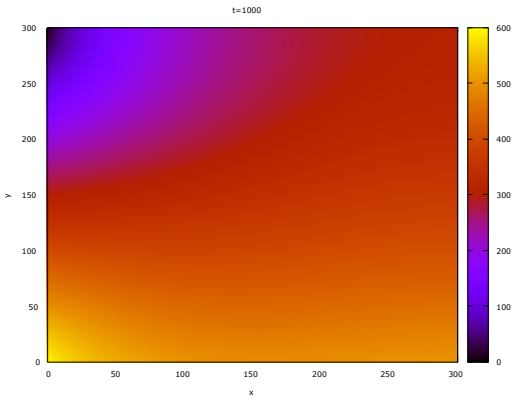
(c) t=50



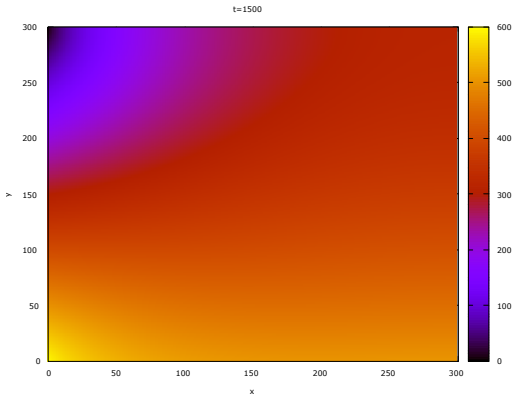
(d) t=100



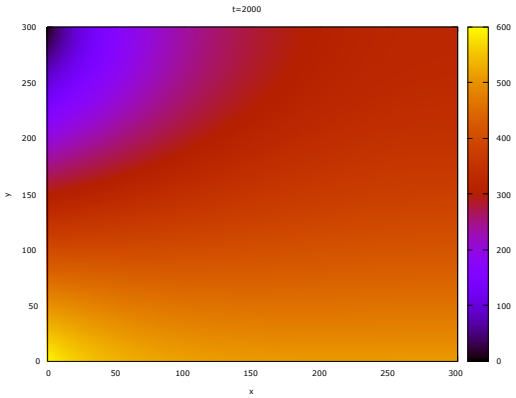
(e) t=500



(f) t=1000



(g) t=1500



(h) t=2000

3. ábra. $k = 10$, $q = 1$

Hivatkozások

[1] Parabolic partial differential equation. Retrieved from https://en.wikipedia.org/wiki/Parabolic_partial_differential_equation (2017. 06. 27.).