Enkla PDE med Δu i planet

Håkan T. Johansson, F96

1 juni 1998

1 Sammanfattning

För att testa finita elementmetoden har jag valt att skriva ett program som löser Poissions ekvation i planet givet en färdig, men godtycklig, triangulering av det aktuella området. För att hålla nere arbetet har jag begränsat mig till ett homogent Dirichlet randvillkor.

För att begränsa rapportens omfång, hänvisas läsaren angående tekniska detaljer till
`http://www.dd.chalmers.se/~f96hajo/pde3/home.html`
där bland annat den använda MATLAB-koden finns[1].

2 Skapa noder och trianglar

För att programmet ska ha något att räkna på, måste jag först göra en triangulering av ett område. Trianguleringsfunktionerna har syntaxen

```
function [xy,trinodes,neighbours,trisofnodes,innernodes]=
   some_nodes_and_triangles(radius,h)
```

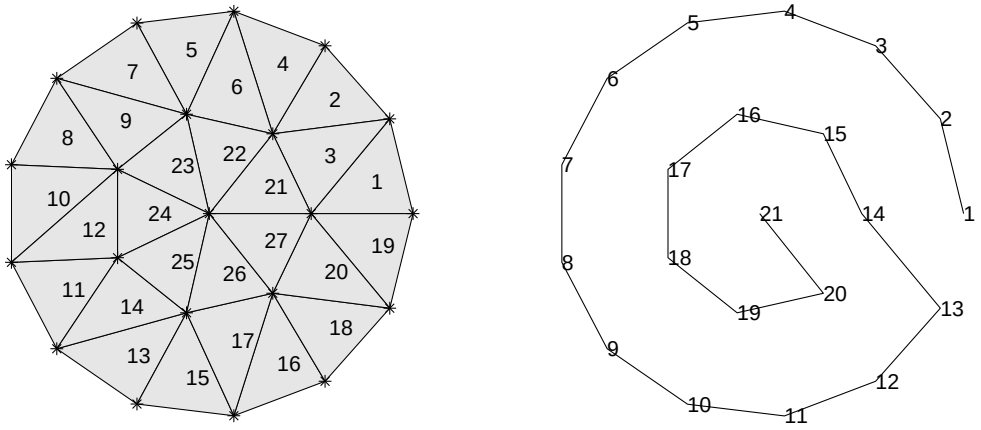
med följande in- och utdata:

`radius`	Karakteristisk längd.
`h`	Typisk kantlängd.
`xy`	Nodernas koordinater.
`trinodes`	Trianglarnas hörn som nodnummer.
`neighbours`	Grannmatris för noderna.
	Element i, j är 1 om nod i och j har någon triangel gemensam.
`trisofnodes`	Matris som kan användas som korsrefens mellan noder och trianglar.
	Element i, j är 1 om nod i är hörn till triangel j.
`innernodes`	Lista över nodnummer som ej är kantnoder.

[1] Därmed menar jag inte att koden är lättläst.

2.1 Cirkulärt område

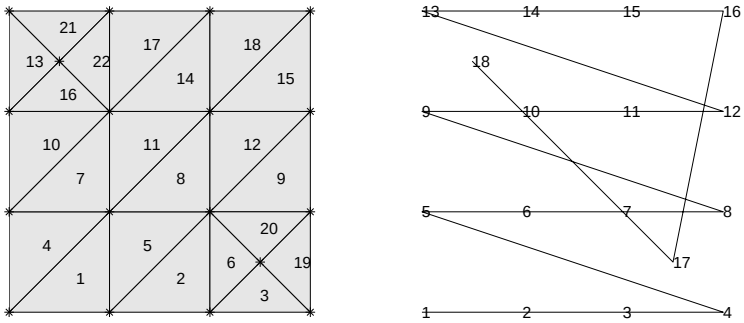
Som figur 1 visar arbetar rutinen genom att varv efter varv, utifrån och in, placera ut noder med avståndet $\approx$ `h`. Radierna i varven varierar också med $\approx$ `h`. När två varv noder placerats ut, läggs också trianglar ut. Trianglar med två noder i det yttre varvet läggs runt hela vägen, så att varje triangels tredje hörn blir den nod i inre varvet som kommer närmast. Sedan läggs även trianglar med två noder i inre varvet och en nod i yttre varvet där detta behövs för att fylla segmentet mellan varven helt. Detta förfarande upprepas tills centrumnoden har agerat innervarv.



Figur 1: Trianglar och noder i ett cirkulärt område, med kantlängd $\approx$ halva radien.

2.2 Kvadratiskt område

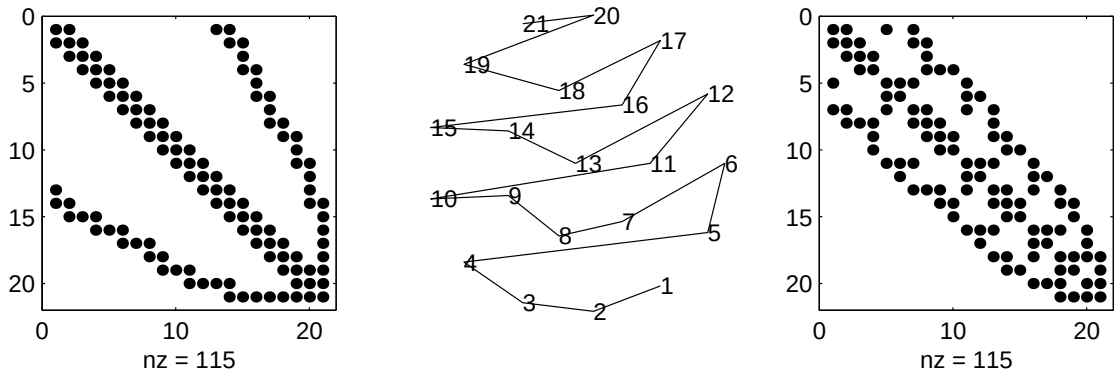
För att kunna testa andra underprogram i projektet mot kända resultat finns även en algoritm som genererar noder och trianglar i ett kvadratiskt område, med en standard-numrering av noderna.



Figur 2: Trianglar och noder i ett rektangulärt område, med kantlängd $= \frac{1}{3}$ sidlängd. Noderna 17 och 18 och motsvarande trianglar har lagts till eftersom vissa metoder senare i programmet mår direkt dåligt av trianglar med alla tre hörnen på områdets rand.

2.3 Omnumrering av noder

Vid den senare lösningen av ekvationssystemen som uppkommer m.h.a. variationsformuleringen är det fördelaktigt om noder som är grannar också har nummer som ligger nära varandra, d.v.s. S i $Sx = b$ har en liten bandbredd. Eftersom S kommer att likna grannmatrisen kan jag på ett tidigt stadium använda MATLAB-kommandot `symrcm`[2] på grannmatrisen, och numrera om noderna enligt den permutationsvektor som `symrcm` ger.



Figur 3: Grannmatrisen till vänster är före omnumrering av noderna i figur 1. Genom omnumreringen som visas i mitten har grannmatrisens bandbredd krympt drastiskt, vilket syns till höger.

3 Uppställning av ekvationssystem

Den efterfrågade Poissons ekvation med ett enkelt randvillkor:

$$\begin{cases} -\Delta u = f, & \Omega \\ u = 0, & \partial\Omega \end{cases} \tag{1}$$

Variationsformulera genom att multiplicera med testfunktion v och integrera över området. Eftersom vi vill ha $u = 0$ på randen, kräver vi samma sak av v, så att randtermerna går bort.

$$\int_\Omega -\Delta u \cdot v = \int_\Omega f \cdot v \qquad \Rightarrow \qquad \int_\Omega \nabla u \cdot \nabla v = \int_\Omega f \cdot v \tag{2}$$

[2]Ur hjälpen till MATLAB:

```
SYMRCM Symmetric reverse Cuthill-McKee permutation.
p = SYMRCM(S) returns a permutation vector p such that S(p,p) tends to have its
diagonal elements closer to the diagonal than S.
```

Låt nu v vara endast bilinjära funktioner på Ω och beteckna en uppsättning basfunktioner med φ_k, där φ_k är tältfunktioner med tältpinne av längd 1 i nod k och fast förankrad i markplanet $= 0$ i alla andra noder.

Ekvationen 2 kan nu skrivas om som ett linjärt ekvationssystem $Su = b$, där S är styvhetsmatrisen som innehåller alla integrerade skalärprodukter mellan tältgradienterna, u är lösningsvektorn (u_k är lösningens värde i nod k), och b är lastvektorn, d.v.s. tälten multiplicerade med lasten f och integrerade över Ω.

3.1 Styvhetsmatrisen

För att beräkna styvhetsmatrisen S av storlek $n \times n$ används en funktion som för varje i och j, med $j > i$ och i granne med j listar ut vilka trianglar som har både i och j som hörn. För var och en av dessa trianglar beräknas koefficienterna i planets ekvation dels för det plan som är 1 i nod i och 0 i de båda andra hörnen, och för det plan som är 1 i nod j och 0 i de två andra hörnen. Med hjälp av dessa koefficienter kan gradienterna för tältfunktionerna φ_i och φ_j bestämmas på den aktuella triangeln. Det är sedan en enkel match att ta skalärprodukten mellan gradienterna, multiplicera med triangelns area och lägga svaret till $S_{i,j}$ och $S_{j,i}$ (men bara en gång om $i = j$).

Genom att använda meshgeneratorn för det kvadratiska området ser man att ovan beskrivna funktion ger den karakteristiska matrisen S med 4:or i huvuddiagonalen och 1:or för dess grannar.

3.2 Lastvektorn

För att beräkna lastvektorns komponenter b_k loopar en funktion över de trianglar som har hörn i nod k, och beräknar planets ekvation för tältfunktionen φ_k på varje triangel. Med hjälp av planets ekvation kan sedan tältfunktionens värde beräknas i ett antal strategiskt placerade punkter och multpliceras med motsvarande värden för funktionen f. Summan av produkterna multipliceras sedan med triangelns area och delas med antalet strategiska punkter och läggs till b_k.

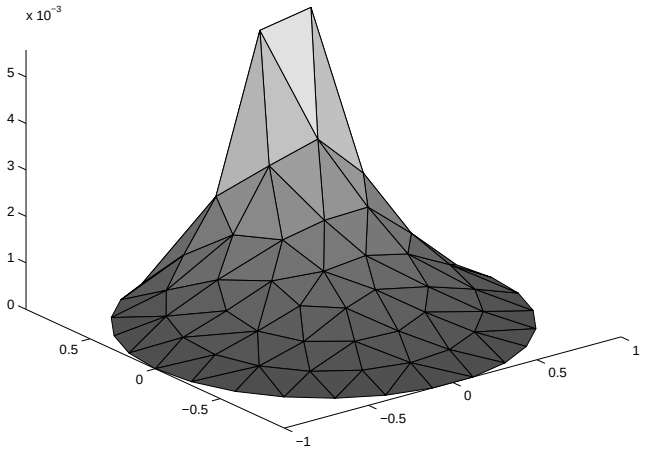
4 Beräkning av lösningen

Eftersom allt jobb redan är gjort, räcker det nu att skriva

```
u = S \ b
```

i MATLAB. Eftersom S är gles och ett rimligt antal noder har använts går lösningen snabbt och det är bara att plotta lösningen med någon lämplig rutin[3]. I figur 4 demonstreras en lösning av ekvation 1.

[3]Eftersom noderna är godtyckligt utplacerade biter inget av standardplot-kommandona i MATLAB, men detta är ett klart tekniskt problem, varför jag hänvisar till koden.



Figur 4: Här har ekvation 1 lösts med $f = 1$ inom $|(x, y) - (.3, .7)| < .1$ och 0 annars.

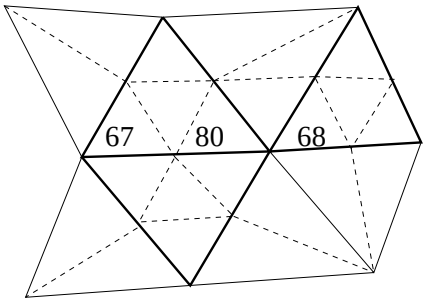
5 Adaptivitet

Eftersom figur 4 blev relativt ful blev det nödvändigt att söka lösa problemet med betydligt fler noder. Att generera noder och trianglar med mindre kantlängd är i sig inget problem med den i avsnitt 2.1 beskrivna metoden, men med den ökning av antalet noder som krävs för en vacker bild blir tiden för uppställning och lösning av ekvationssystemet snabbt oacceptabelt lång. Problemet är alltså att placera ut noderna på ett slugare sätt.

5.1 Förfiningskriterium

Då jag definierar en ful bild som en bild med skarpa kanter är det naturligt att kräva att programmet förfinar lösningen just där det är gott om skarpa kanter. Dessutom är det enkelt att detektera skarpa kanter.

Förfiningsalgoritmen som anropas efter att en lösning beräknats börjar således med att beräkna enhetsnormaler till alla triangelplan på lösningsytan. Sedan går den helt enkelt igenom alla kanter mellan två trianglar och markerar dessa två trianglar för uppdatering om vinkeln mellan normalerna överstiger ett av förutbestämt (tyvärr problemberoende) värde. De trianglar som markerats för uppdatering delas i fyra nya genom att noder placeras ut mitt på dess tre kanter. Alla trianglar som ligger granne med uppdaterade trianglar drabbas även de av delning. Figur 5 visar ett exempel.

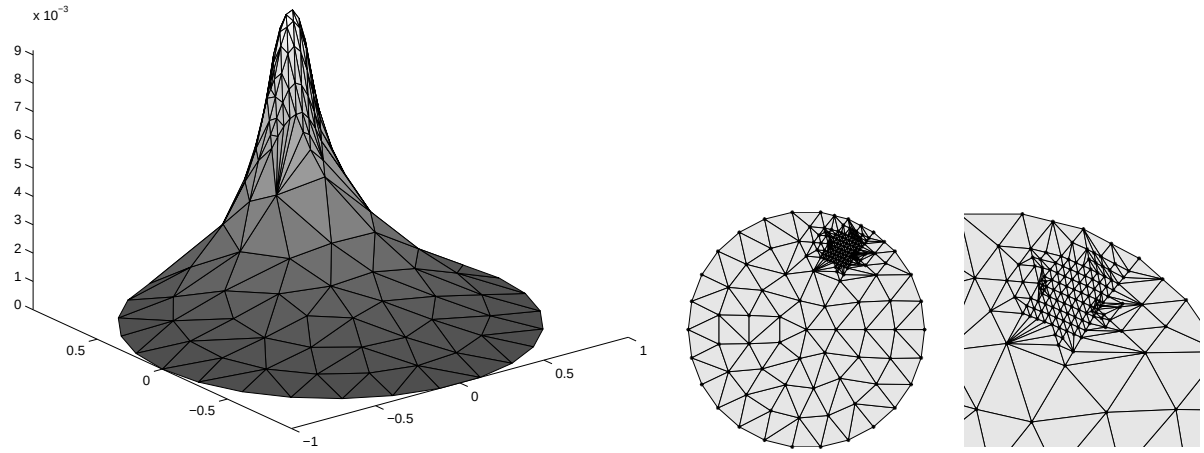


Figur 5: Delning av trianglar markerade för uppdatering och de oskyldiga grannar som drabbas.

När förfiningen är färdig beräknas helt enkelt styvhetsmatrisen och lastvektorn igen för den nya kombinationen av noder och trianglar, varefter $Su = b$ löses igen och lösningen plottas. Förfiningsalgoritmen anropas igen tills dess att alla vinklar mellan grannnormaler understiger det förutbestämda värdet.

5.2 Iterativ lösningsmetod

Eftersom lösningen efter ett förfiningssteg inte skiljer sig så mycket från lösningen innan förfingen, är det dumt att lösa $Su = b$ från början efter varje förfining. Istället används en iterativ lösningsmetod för lösning av ekvationssystemet. Som startapproximation används den gamla lösningen, utökad med värden i de nya noderna satta till medelvärdet av lösningen i de noder som varje ny nod har separerat. Detta innebär tex att i figur 5 blir $u_{80} = \frac{u_{67}+u_{68}}{2}$



Figur 6: Här är lösningen till samma problem som i figur 4, men med ett förfinat nät.