Numerisk lösning av PDE: Comsol Multiphysics

I denna lab ska du bekanta dig med programmet Comsol Multiphysics för numerisk lösning av PDE med finita element. Programmet har många faciliteter och vi kommer bara att använda en liten del.

Hjälp

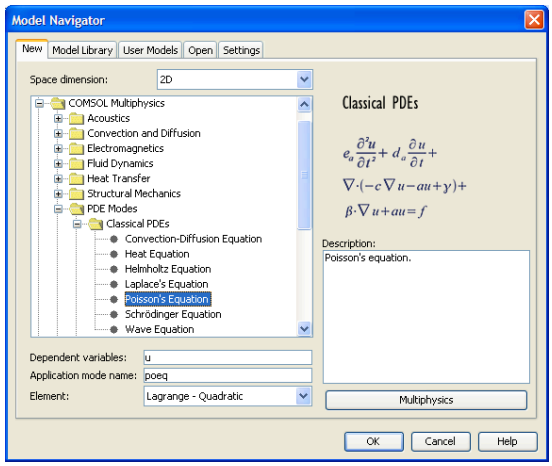
I menyraden överst finns längst till höger en hjälp-knapp ? som ger tillgång till all dokumentation (om man installerat alla filer). Lämpligt att starta med *Quick Start*, men den är ganska omfattande, och GUI-funktionerna är förhoppningsvis ganska intuitiva.

Exempel 1, Poissons ekvation på en ellips.

Starta comsol.

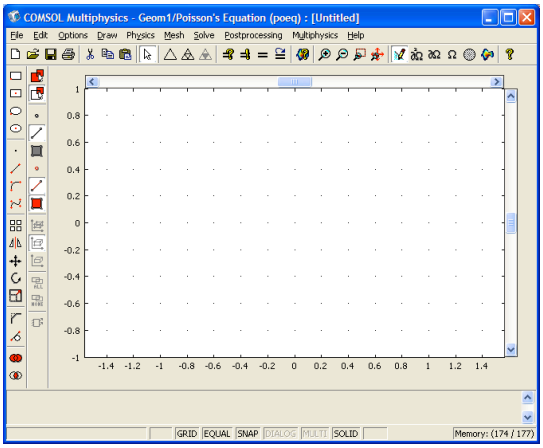
I den första skärmbilden **`ModelNavigator`** väljer vi 2D och `ApplicationModes>ComsolMultiphysics>PDEModes>ClassicalPDEs>PoissonäsEquation.`

Här syns också att elementtypen är **`Lagrange P2`**-trianglar, dvs., lösningen approximeras med styckevis andragradspolynom, ett över varje triangel. Det går att ändra på sedan.



Nästa skärmbild visar arbetsytan och man arbetar oftast från vänster till höger i menyraden:

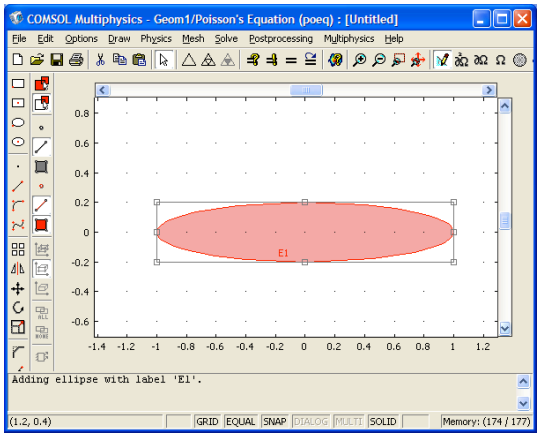
1. Skapa geometri (**`Draw`**)
2. Definiera PDEn (**`Physics`**)
3. Sätt randvillkor (**`Physics`**)
4. Gör elementnät (**`Mesh`**)
5. Lös (**`Solve`**)
6. Plotta (**`Postprocessing`**)
7. Spara modellen (**`File`**)



`Options` ger faciliteter som att definiera konstanter och uttryck. Praktiskt att ha namn på värdena på modellparametrar som värmeledningstal, specifikt värme, etc.

Geometri

Man bygger geometrin i ett rit-program från elementar-kroppar, i 2D rektanglar och ellipser, eller ytor som begränsas av Bezier-kurvor. Delkropparna sätts ihop med mängdoperationer. Vi tar en ellips, centrum i origo och halvaxlar 1 och 0.2. Observera att objekten snäpps mot rutnätet, ändra rutnätet under **`Options>GridSettings`** om



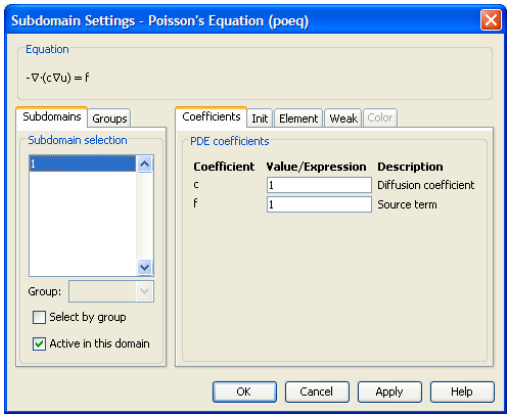
det behövs. Om objektet som här består av en enda delkropp kan vi direkt gå vidare.

Definiera PDE

`Physics>SubdomainSettings` (det finns boundary också, strax…)
Här står PDEn:

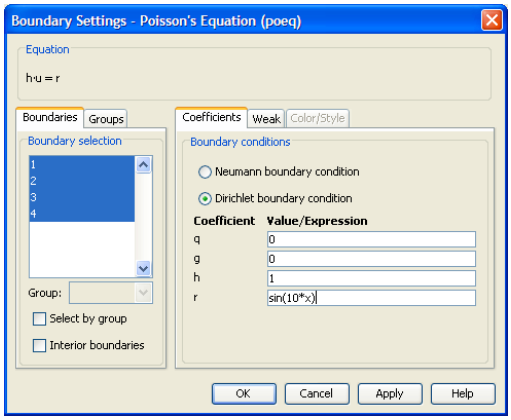
$$-\text{div}\,(c\,\text{grad}u) = f$$

och vi ska definiera koefficienten c och högerledet f. Man kan ge funktioner av $x,y,$ $u,$ $ux,$ uy och (om tidsberoende) t. Som synes finns bara en subdomän, ellipsen. Välj den och acceptera $c = f = 1$ genom OK.



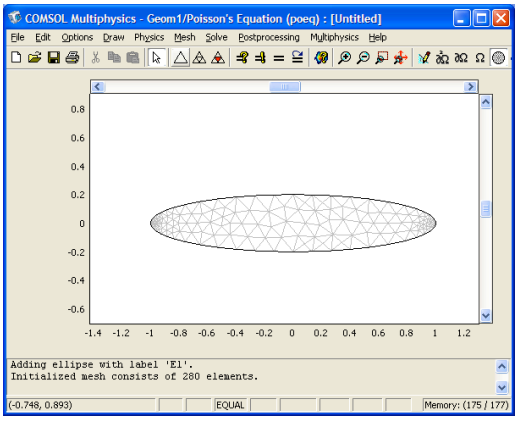
Randvillkor

`Physics>BoundarySettings`
Varje ellips ger fyra ränder. Vi väljer alla med ctrl-a; Det går att peka på dem också på arbetsytan. Default-randvillkoret är uppenbarligen Dirichlet: $h\,u = r$ med $h = 1$ och $r = 0$ och vi byter till $r = \sin(10x)$ för att få lite variation, OK.



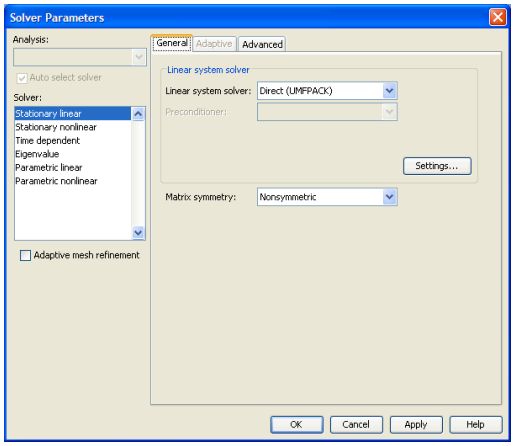
Elementnät

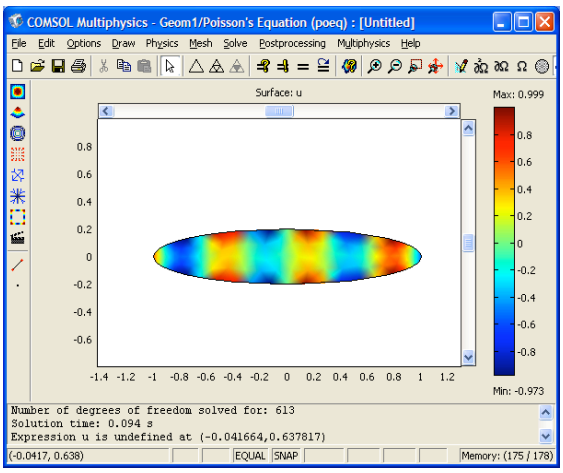
Ikonen triangel ger ett nät. Det ser bra ut, OK. Man kan noga styra element-genereringen med parametrar som sätts under **`Mesh>MeshParameters`** .



Lösning

Man kan lösa de genererade ekvationerna med ikonen =. Vi går först in under **`Solve>SolverParameters`** och ser vilka möjligheter som bjuds. Med vårt val av c, f och randvillkor är problemet linjärt, så förvalet **`stationary linear`** är bra, OK. Detaljer som kan vara nog så viktiga för stora problem är t ex val av numerisk lösare för de linjära ekvationssystemen kan styras här. Visserligen står det **`Unsymmetric`** om matrisen (och den är symmetrisk) men vi accepterar det, OK, och löser med =.





Efter 0.1 sekunder har systemet med 613 obekanta ställts upp och lösts och lösningen målats som färgade iso-ytor på arbetsytan, och log-raden säger

```
Number of degrees of freedom
solved for: 613
Solution time: 0.093 s
```

Spara modellen
Som `.mph`-fil med all information, ett binär-format som bara Multiphysics förstår, eller som `.m`-fil som är text-fil och förstås av matlab och matlab-klonen Comsol Script. Där saknas dock detaljer i nätgenereringen; nät och lösningar sparas på separat binärfil (som bara Multiphysics förstår).
Båda filformaten öppnar man i Multiphysics under **`File`** med **`Open`** …. Man kan också (överkurs) starta Comsol Script och köra comsol-modeller sparade som `m`-filer, och exportera resultat från Multiphysics till Script för vidare bearbetning, etc.

Konvergensstudie
Vi vill se hur lösningen konvergerar och beräknar *u* i centrum . Välj
`Postprocessing>DataDisplay>Subdomain`
där vi ger koordinaterna för centrum och ser

```
Value: 0.020201, Expression: u, Position: (0,0)
```

i log-raden.
Förfina nätet reguljärt med fyrtriangelikonen, lös igen, skriv ut u(centrum) igen:

```
Number of degrees of freedom solved for: 2345
Solution time: 0.188 s
Value: 0.019069, Expression: u, Position: (0,0)
```

och igen, och igen, ... Prova upp till ca en miljon frihetsgrader, sedan tar minnet slut.
Konvergens:

u(0,0)	**diff.**
1201	
69	-1132
231	162
234	3
231	-3

Konvergensordningen ska för reguljära lösningar bli 2 i energi-norm och 3 i max-norm. Här tycks det i alla fall gå fort, tills avrundningsfelen stör bilden. I allmänhet kan vi inte vänta oss samma mycket regelbundna konvergens som man ser med t ex trapetsregeln för kvadratur på snälla funktioner, eftersom elementnätet är så oregelbundet. Inte heller blir det alltid *precis* samma nät vid två likadana körningar! I nätgenereringsprocessen ingår val som styrs av (pseudo-)slumptal.

Prova nu med linjära ansatsfunktioner istället. **`Lagrange P1`** element väljs under
`Physics>SubdomainSettings>Element`

Experimentet upprepas: Långsammare konvergens, ordning runt 2.

```
#dof     tid     u(0.0)    diff.
   613   0.094   0.019103
  2345   0.156   0.019115  12
  9169   0.547   0.019205  90
 36257   2.172   0.019224  19
144193  11.204   0.019230   6
```

Exempel 2 – Instationär värmeledning
Starta ny modell genom att välja **`File>New`**, välj PDE:
`ApplicationModes>ComsolMultiphysics>PDEModes>ClassicalPDEs>Heat Equation`.

Geometri
Samma som ovan: en ellips.

Definiera PDE
`Physics>SubdomainSettings`
Välj den enda domänen, sätt $f = 0$ och acceptera $c = 1$. Välj $u = 1$ som begynnelsevillkor under fliken **`init`**, därefter OK.

Randvillkor
`Physics>BoundarySettings`
Vi väljer alla med ctrl-a; default-randvillkoret är Dirichlet: Sätt $r = \sin(10x)\sin(10t)$, acceptera $h = 1$, OK.

Elementnät
Triangel-ikonen ger ett nät. Förfina reguljärt en gång med fyrtriangelikonen, OK.

Lösning
Se under **`Solver Parameters`**: Spar lösningen för t = 0:0.1:1, OK, feltoleranser 0.001 (absolut) och 0.01 (relativt), varför inte. Lös med =.

Plottning
`Plot Parameters`: Välj att visa u som färgfält (**`surface`**, default) och som z-data (under fliken `height data`, klicka i checkbox), `apply`. Eftersom randvillkoren är periodiska med period $2\pi/10$ svänger också lösningen in mot en periodisk lösning. Nu räcker inte 10 snapshots till för snygg animering, så på **`Animation`**-fliken väljs **`interpolated`**, t ex **`0:0.025:1`** i stället för förvalet 0:0.1:1.

Parametervariation
Byt till $c = 0.1$, kör igen. Hur lösningen ändras när c väljs liten kan man se genom att beräkna periodiska tvungna svängningen: $u = \mathrm{e}^{i\omega t}\, U(x,y)$, $\omega = 10$, ger Helmholtz ekvation,

$$-\frac{i\omega}{c}U = \Delta U$$

$U = \sin(10x)$ på randen.

där alltså lösningen kommer att variera över en längdskala $L = \sqrt{\frac{c}{\omega}}$. Om L << lilla halvaxeln kommer lösningen längre än L från randen att bli liten (0). Man behöver

alltså också elementstorlek << L för att lösa upp lösningens variation. Du kan förstås också hastigt lösa Helmholtz-problemet med Multiphysics som förstår komplexa tal, men poängen är att utifrån parametervärdena dra slutsatser om hur lösningen ska se ut.