ESF EIROPAS SOCIĀLAIS FONDS IEGULDĪJUMS TAVĀ NĀKOTNĒ

29.11.2012
Raimonds Viļums (raimonds.vilums@lu.lv)
LU Fizikas un matemātikas fakultāte

I. Temperatūra elektriskajā vadā un tā izolācijā

Matemātiskā modelēšana, izmantojot OpenFOAM

Šī dokumenta mērķis ir, izmantojot konkrētu piemēru, palīdzēt apgūt atvērtā koda datorprogrammu paketi OpenFOAM, kas paredzēta fizikālo procesu skaitliskajai modelēšanai. OpenFOAM programmu pakotnei nav visaptveroša apraksta, kas apgrūtina šī atvērtā koda programmprodukta apgūšanu.

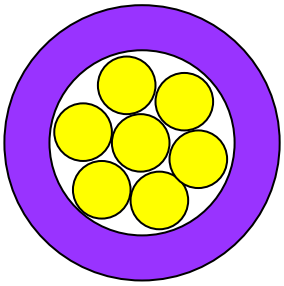
Pirms pildīt šo pamācību, lasītājam vēlams iepazīties ar „User Guide" (UG) un „Programmer's Guide" (PG) dokumentiem angļu valodā, kas atrodas OpenFOAM instalācijas direktorijā *doc/Guides-a4*. UG publicēts arī OpenFOAM oficiālajā mājas lapā. Nepieciešama, protams, pieeja datoram ar uzinstalētu OpenFOAM, kā arī papildus instalējamu rīku swak4foam un programmu ParaView rezultātu vizualizācijai. Šīs pamācības piemēri veidoti ar OpenFOAM versiju 2.1.x un ParaView 3.10.

Papildu resursi:
http://matmod.pbworks.com/w/browse/#view=ViewFolder&param=OpenFOAM-Latviski
http://www.openfoam.org
http://www.openfoamwiki.net
http://www.cfd-online.com/Forums/openfoam

Pirmā saite ved uz lapu, kurā atrodas šī pamācība tās izveidošanas brīdī; otrā – uz oficiālo OpenFOAM fonda lapu. Trešā adrese norāda uz neoficiālu wiki resursu, bet pēdējā – uz aktīvāko forumu, kurā OpenFOAM lietotāji mēģina rast atbildes uz saviem jautājumiem.

1. Viendimensiju matemātiskā problēma izolācijai ar konstantiem koeficientiem un Dirihlē robežnosacījumiem

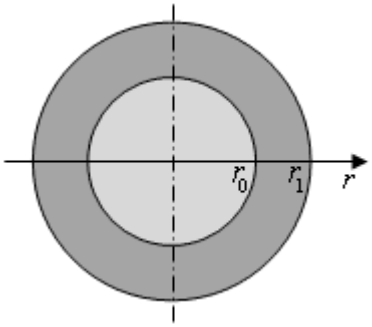
Fizikālā situācija



Dots apaļš elektriskais vads ar izolāciju, kuru uzkarsē cauri plūstoša līdzstrāva. Uzdevums ir — vai nu aprēķināt stacionāro temperatūras sadalījumu vadā un tā izolācijā, vai arī noskaidrot laiku, kādā vads uzkarst līdz kritiskajai temperatūrai, ja strāva ir pārāk liela. Elektriskais vadītājs var sastāvēt no vairākiem metāla vadiņiem.

Problēmas formulējums

Sāksim ar vienkāršu modeli un nākamajās nodaļās pamazām to veidosim sarežģītāku un pilnīgāku. Par pamatu ņemsim vienkāršotu ģeometriju:



Aplūkosim tikai izolācijas apgabalu vienā dimensijā (rādiusa virzienā) un uzskatīsim, ka zinām, kāda temperatūra ir gan izolācijas ārpusē, gan uz iekšējās tās robežas. Ja temperatūru apzīmē ar funkciju $T(x,t)$, tad sekojošs Laplasa vienādojums raksturo siltuma plūsmu materiālā:

$$\frac{\partial T}{\partial t} + D_T \Delta T = 0. \tag{1}$$

Pieņemsim, ka izolācija ir no polivinilhlorīda (PVC), kam augšējā darba temperatūras robeža ir 105 grādi pēc Celsija. Par ārējās vides un vada sākotnējo temperatūru ņemsim 65 grādus. Šīs vērtības izmantosim uz robežām, lai iegūtu temperatūras sadalījumu izolācijā:

$$T(r_0,t) = 105, \qquad T(r_1,t) = T(r,0) = 65. \tag{2}$$

Par piemēru ņemam vadu, kura iekšējais diametrs ir 3.25 *mm*, bet ārējais – 4.15 *mm*, tātad $r_0 = 1.625mm$ un $r_1 = 2.075mm$. Koeficientu D_T aprēķināsim sekojoši:

$$D_T = \frac{k}{\gamma} = \frac{0.17\,\frac{\mathrm{W}}{\mathrm{m}\cdot\mathrm{K}}}{1.4\cdot 10^6\,\frac{\mathrm{J}}{\mathrm{m}^3\cdot\mathrm{K}}} = 1.21429\cdot 10^{-7}\,\frac{\mathrm{m}^2}{\mathrm{s}}. \tag{3}$$

k ir PVC īpatnējais siltumvadīšanas, bet $\gamma = c\rho$ siltumietilpības (pēc tilpuma) koeficients. Realitātē šie lielumi ir nelineāri atkarīgi no temperatūras, bet mēs pašlaik izmantosim aptuvenas konstantas vērtības.

OpenFOAM *case* datnes

Ja tas vēl nav izdarīts, izveidojam savu OpenFOAM programmu mapi, kas parasti saucas *<username>-<nr>/run*, (piemēram, *raimonds-2.1.x/run*). Pie reizes varam iekopēt tajā arī OpenFOAM piemērus, ar kuriem droši eksperimentēt OpenFOAM apgūšanas laikā:

```
mkdir -p $FOAM_RUN
cp -r $FOAM_TUTORIALS $FOAM_RUN
```

Ar komandu *run* varam aiziet uz savu OpenFOAM mapi *$FOAM_RUN*:

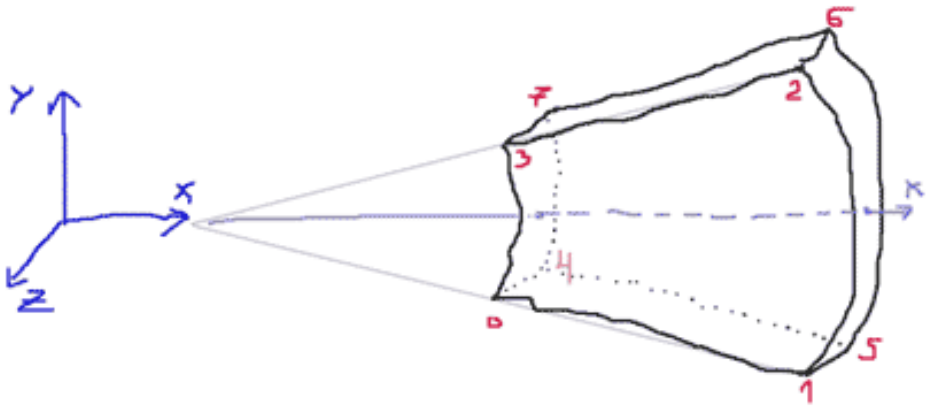
```
run
```

Mūsu izolētajam vadam kā paraugu ņemam datnes no programmu paketei OpenFOAM līdzi nākošā piemēra *basic/laplacianFoam/flange*, izveidojot kopijas mapē *wire/wire-01*. Izdzēšam arī datnes, kas nebūs vajadzīgas:

```
  mkdir -p $FOAM_RUN/wire/wire-01
  cp -r $FOAM_TUTORIALS/basic/laplacianFoam/flange/* $FOAM_RUN/wire/wire-01
  cd $FOAM_RUN/wire/wire-01
  rm Allclean Allrun flange.ans constant/polyMesh/boundary
constant/polyMesh/boundary.org
```

Režģa ģenerēšana

Ģeometriju, izmērus un režģa ģenerēšanas parametrus norādām datnē *constant/polyMesh/blockMeshDict*. Programmai OpenFOAM nepieciešams 3D apgabals, tāpēc *z* virzienā mums būs *empty* plaknes, bet tām plaknēm, ko nosaka leņķis, jāņem tips *wedge* (UG nodaļa 5.2.2); pie tam abas malas nedrīkst apvienot. Ģeometrijai ņemam 1° leņķi. Līklīniju malas nosakām ar atslēgvārdu *arc*. Apgabalu brīvi sanumurējam un pēc tā veidojam režģa aprakstu.




```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 0.001;           //Norādīsim izmērus milimetros

vertices
(                                //Numerācija
    (1.62475 -0.028360  0.01)    //0
    (2.07468 -0.036214  0.01)    //1 z-virzienā 0.01mm uz abām pusēm no ass
    (2.07468  0.036214  0.01)    //2  x = r1 cos1°,   y = r1 sin1°
    (1.62475  0.028360  0.01)    //3  x = r0 cos1°,   y = r0 sin1°
    (1.62475 -0.028360 -0.01)    //4
    (2.07468 -0.036214 -0.01)    //5
    (2.07468  0.036214 -0.01)    //6
    (1.62475  0.028360 -0.01)    //7
);

blocks
(
```

```
//Bloka plaknes (ko nosaka pirmie 4 punkti) normālei jābūt vērstai uz iekšu
 //Ja skatās uz plakni, kas definēta ar punktiem sarindotiem pretēji
 //pulksteņrādītāja virzienam, normāle ir vērsta uz skatītāja seju.
 //Piemēram, hex (0 1 2 3 4 5 6 7).. šeit neder.
    hex (4 5 6 7 0 1 2 3) (20 1 1) simpleGrading (1 1 1)
);

edges
(
    arc 0 3 (1.625 0  0.01)     //Norādām malas, kas nav taisnas un caur kādu
    arc 4 7 (1.625 0 -0.01)     //punktu tās iet cauri
    arc 1 2 (2.075 0  0.01)
    arc 5 6 (2.075 0 -0.01)
);

patches
(
    patch outerSurface   //izolācijas ārpuse
    (
        (1 5 6 2)
    )
    patch innerSurface   //izolācijas iekšpuse
    (
        (0 4 7 3)
    )
    wedge leftSide       //leņķiskā mala pa kreisi no x ass
    (
        (3 2 6 7)
    )
    wedge rightSide      //leņķiskā mala pa labi no x ass
    (
        (0 4 5 1)
    )
    empty frontAndBack   //šķēlums pa y
    (
        (0 1 2 3)
        (4 7 6 5)
    )
);

mergePatchPairs          //nav vajadzības kaut ko apvienot
(
);

// ************************************************************************* //
```

Ģenerējam režģi, piemēra direktorijā *wire-01* izpildot komandu *blockMesh*:

```
cd $FOAM_RUN/wire/wire-01
blockMesh
```

Mapē *constant/polyMesh* tiek izveidotas datnes *boundary*, *faces*, *neighbour*, *owner* un *points*.

Parametru uzstādīšana un uzdevuma risināšana

Uzdevumu rēķināsim ar OpenFOAM aplikāciju *laplacianFoam*, kas risina diferenciālvienādojumu (1). Datnē *constant/transportProperties* norādām parametru D_T:

```
FoamFile
{
    version     2.0;
    format      ascii;
```

```
    class       dictionary;
    location    "constant";
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

DT              DT [ 0 2 -1 0 0 0 0 ] 1.21429e-07;

// ************************************************************************* //
```

Sākuma un robežnosacījumus norādām datnē *0/T*, nomainot tekstu aiz rindiņas *dimensions*. Uz iekšējās malas fiksēsim temperatūru 105°C, uz ārējās: 65°C. Sākuma momentā izolācijai būs 65°C:

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      T;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 1 0 0 0];

internalField   uniform 65;

boundaryField
{
    outerSurface
    {
        type            fixedValue;
        value           uniform 65;
    }
    innerSurface
    {
        type            fixedValue;
        value           uniform 105;
    }
    leftSide
    {
        type            wedge;
    }
    rightSide
    {
        type            wedge;
    }
    frontAndBack
    {
        type            empty;
    }
}

// ************************************************************************* //
```

Datnē *system/controlDict*, kurā tiek norādīti laika iterāciju parametri, mainām beigu laiku un rezultātu pierakstīšanas biežumu:

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
```

```
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     laplacianFoam;

startFrom       latestTime;

startTime       0;

stopAt          endTime;

endTime         1;

deltaT          0.005;

writeControl    runTime;

writeInterval   0.04;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;


// ************************************************************************* //
```

Pārējās divas datnes *fvSchemes* un *fvSolution*, kas atrodas mapē *system*, nemainām:

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;
    grad(T)         Gauss linear;
}

divSchemes
{
    default         none;
}

laplacianSchemes
```

```
{
    default         none;
    laplacian(DT,T) Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         corrected;
}

fluxRequired
{
    default         no;
    T               ;
}

// ************************************************************************* //
```

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    T
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-06;
        relTol          0;
    }
}

SIMPLE
{
    nNonOrthogonalCorrectors  2;
}


// ************************************************************************* //
```

Ejam uz mūsu piemēra sākuma mapi un palaižam rēķināšanas procesu, izpildot komandu *laplacianFoam*:

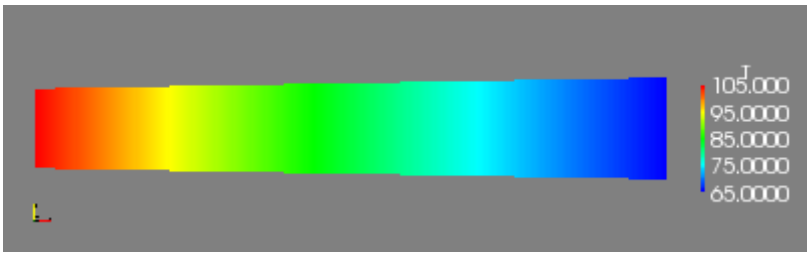
```
cd $FOAM_RUN/wire/wire-01
laplacianFoam
```

Ja viss izdarīts pareizi, programmai vajadzētu uz ekrāna parādīt informāciju par katru izpildīto iterāciju un rēķināšanu noslēgt ar vārdu *End*. Piemēra mapē būs izveidotas laika direktorijas ar mūsu norādīto soli 0.04 sekundes, kas saturēs informāciju par temperatūras sadalījumu izolācijā atbilstošajā laika momentā.

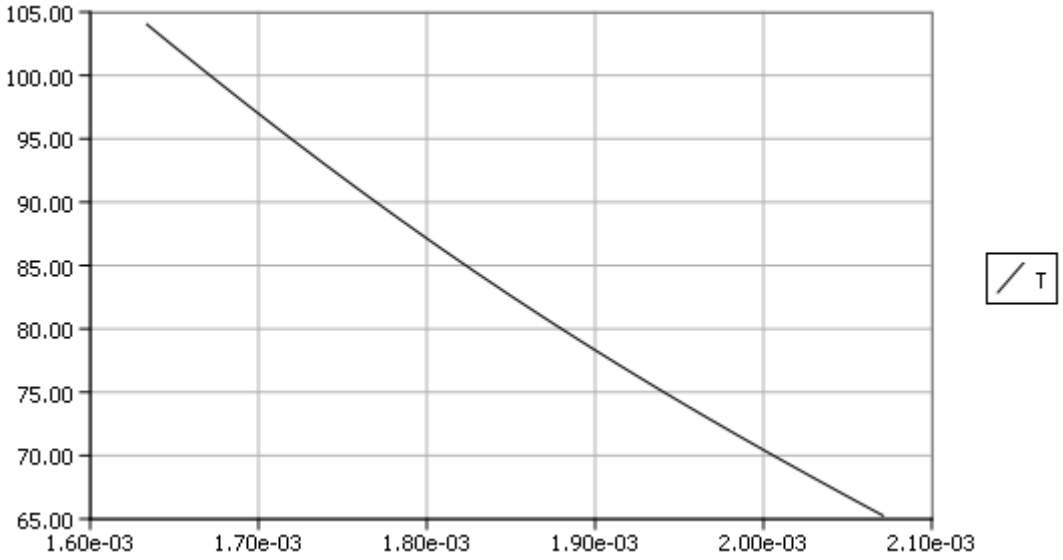
Rezultāti

Datu vizualizācijai izmantosim programmu *ParaView*. Veids, kādā piemēru veiksmīgi ielādēt programmā, dažādiem datoriem var būt atšķirīgs. Viens variants ir izpildīt komandu *paraFoam*, otrs – piemēra direktorijā izveidot tukšu datni ar paplašinājumu *.foam* un atvērt to tieši no programmas *ParaView*. Pēdējo variantu var izmantot arī, ja piemēra datnes tiek pārkopētas pēcapstrādei uz citu datoru (piemēram, no klastera uz Windows mašīnu).

Atverot piemēru programmā *ParaView*, spiežam pogu *Apply*, lapiņā *Display* izvēlamies *Color by: T* un *Represantation: Surface*:



Noklusētā krāsu skala var atšķirties no attēlā redzamā. Tālāk izvēlamies *Filter > Plot Over Line*, tad pogu *X Axis* un *Apply*, lapiņā *Display* nomainām *Component* uz *X* un noņemam ķekšus pie gradientiem:



Līnija nav taisna, bet izliekta, par ko var pārliecināties kaut vai pieliekot lineālu pie attēla. Ja izolācija būtu biezāka, izliekums būtu izteiktāks.

2. Trešā veida jeb Robina robežnosacījums uz ārējās robežas

Uz ārējās izolācijas robežas norādīsim homogēnu trešā veida robežnosacījumu, kas raksturo siltumapmaiņu ar apkārtējo vidi:

$$k\frac{\partial T}{\partial \mathbf{n}}+\alpha\left(T-T_{\infty}\right)=0. \qquad (4)$$

k ir jau minētais siltumvadīšanas koeficients, T_∞ – ārējās vides temperatūra, α – konvekcijas koeficients, kas teorētiski ir nelineāri atkarīgs no temperatūras, bet mēs pašlaik ņemsim konstanti.

Izveidojam mapes *wire-01* kopiju, izņemot sarēķinātās laika direktorijas, un nosaucam to par *wire-02*. Lietosim pakotnē *swak4foam* ietilpstošo *groovyBC* robežnosacījuma veidu, kas ir *mixed* robežnosacījums ar paplašinātām iespējām.

Swak4foam ir trešās puses izstrādāts rīks, kas ļauj veikt daudz dažādu uzdevumu, kuru realizācijai citā gadījumā būtu jāveic risinātājprogrammu, robežnosacījumu vai cita C++ pirmkoda rediģēšana un kompilācija. Ja *swak4foam* bibliotēkas nav instalētas, izdarām to vai arī lūdzam kādas zinošākas personas palīdzību. Papildu informāciju var atrast dokumenta sākumā minētājās wiki un foruma lapās.

Lai lietotu *groovyBC*, datnē *system/controlDict* pirms rindiņas *application* ievietojam sekojošu tekstu:

```
libs ( "libOpenFOAM.so" "libgroovyBC.so" );
```

Pirmo minēto bibliotēku, iespējams, var nenorādīt, ja bez tās programma *ParaView* veiktos aprēķinus atver bez problēmām.

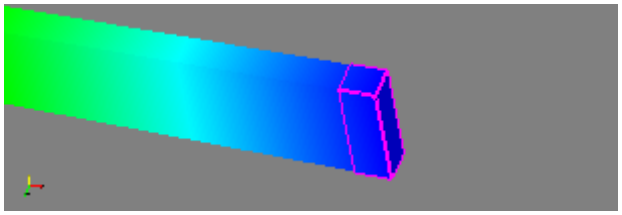
Uz iekšējās robežas norādīsim fiksētu temperatūru: 50 grādus pēc celsija, bet apkārtējās vides temperatūra būs $T_\infty = 0°\mathrm{C}$; Koeficienti: $k = 0.17\,W/mK$, $\alpha = 15\,W/m^2K$. Datnē *0/T* atbilstoši mainām sākuma un robežnosacījumus:

```
internalField   uniform 0;

boundaryField
{
    outerSurface
    {
        type            groovyBC;
        value           uniform 0;
        variables       "k=0.17;alpha=15;Tinf=0;C=k/alpha/mag(delta());";
        valueExpression "Tinf";
        gradientExpression "0";
        fractionExpression "1/(1+C)";
    }
    innerSurface
    {
        type            fixedValue;
        value           uniform 50;
    }
    ...
```

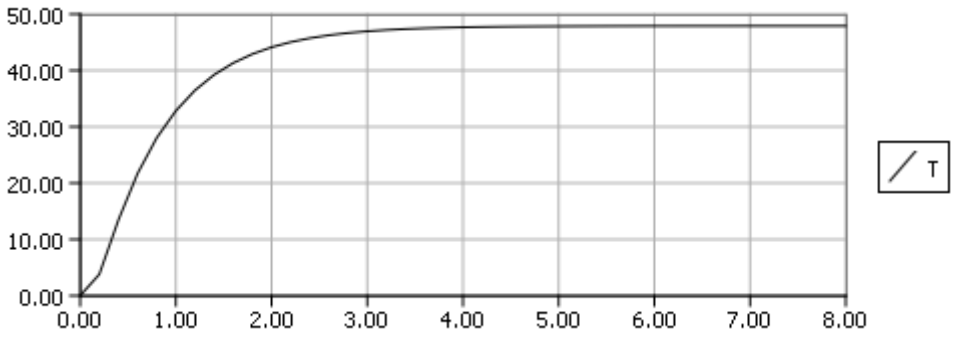
Palielinām aprēķinu periodu, lai sasniegtu stacionāru stāvokli, mainot datnē *system/controlDict* sekojošās rindiņas:

```
endTime         8;

writeInterval   0.2;
```

Palaižam aprēķinus ar komandu *laplacianFoam* un, līdzīgi kā iepriekš, rezultātus apskatāmies ar programmu *ParaView*. Papildus iegūsim grafiku, kā uzkarst izolācija pie ārējās robežas. Izvēlnē ņemam *Edit* > *Select Cells On* (vai attiecīgo pogu rīkjoslā) un 3D attēlā iezīmējam gala elementu:



Tālāk izvēlnē *Filters* lietojam filtru *Plot Selection Over Time* un panelī *Object Inspector* spiežam pogas *Copy Active Selection* un *Apply*:



Trešā veida robežnosacījumu īstenošana ar *groovyBC* nav acīmredzama. C++ klases *groovyBC* pamatā ir robežnosacījums *mixed*. Izpētot funkcijas *evaluate* algoritmu izejas koda datnē *mixedFvPatchField.C* (src » finiteVolume » fields » fvPatchFields » basic » mixed) redzams, ka temperatūra uz robežas T_{face} tiek aprēķināta sekojoši:

$$T_{face} = f \cdot valueExpr + (1-f)\left(T_{center} + gradExpr \cdot \delta\right), \tag{5}$$

kur $f \in [0,1]$ ir proporcija *fractionExpression*, δ ir attālums starp šūnas centru T_{center} un virsmas centru T_{face} (programmā OpenFOAM tas ir vienāds ar $1/$*deltaCoeffs*( ) vai – robežnosacījuma *groovyBC* gadījumā – ar komandu *mag*(*delta*( ))). Ja ņemam robežnosacījumu (4):

$$k\frac{\partial T}{\partial \mathbf{n}} + \alpha\left(T - T_{\infty}\right) = 0, \tag{4}$$

linearizējam tā atvasinājumu:

$$k\frac{T_{face} - T_{center}}{\delta} + \alpha\left(T - T_{\infty}\right) = 0, \tag{6}$$

izdalām ar α un izsakām T_{face}, iegūstam:

$$T_{face} = \frac{1}{1+C}T_{\infty} + \left(1 - \frac{1}{1+C}\right)T_{center}, \qquad C = \frac{k}{\alpha\delta}. \tag{7}$$

Kā redzams, šīs izteiksmes salīdzināšana ar formulu (5) dod:

$f = 1/(1+C)$,

valueExpression = T_{∞},

gradientExpression = 0.

3. Siltuma plūsma uz iekšējās robežas

PVC izolācijas siltumvadītspēja ir aptuveni 2000 reižu mazāka nekā vara vadam, tāpēc var uzskatīt, ka elektriskajā vadītājā viscaur ir vienāda temperatūra. Siltuma daudzumu, kas rodas strāvai plūstot cauri metālam, var raksturot ar sekojošu lineāru sakarību:

$$f_0(T) = \frac{I^2}{A^2}\rho_{20}\left(1+\alpha_{20}\left(T-T_{ref}\right)\right)\ \frac{\mathrm{W}}{\mathrm{m}^3}, \tag{8}$$

kur I – strāvas stiprums, A – vadītāja šķērsgriezuma laukums, ρ_{20} – metāla īpatnējā pretestība atsauces temperatūrā $T_{ref} = 20°\mathrm{C}$, α_{20} – pretestības lineārais temperatūras koeficients.

Ņemsim 50 ampēru stipru strāvu un vara sakausējuma „Cu-ETP" fizikālās īpašības:

$$\rho_{20} = 1.7544\cdot 10^{-8}\,\Omega\mathrm{m}, \qquad \alpha_{20} = 3.83\cdot 10^{-3}\,\frac{1}{\mathrm{K}}. \tag{9}$$

Elektriskais vadītājs sastāv no 84 metāla stieplītēm ar diametru $0.3\mathrm{mm}$, tāpēc šķērsgriezuma laukumu rēķināsim nevis no izolācijas iekšējā rādiusa r_0, bet metāla reālā šķērsgriezuma laukuma:

$$A = n\pi\tilde{r}^2 \approx 5.94\cdot 10^{-6}\,\mathrm{m}^2, \quad \tilde{r} = 0.15\,\mathrm{mm}, \quad n = 84.$$

Stacionārā gadījumā viss ģenerētais siltums nonāk izolācijā caur iekšējo robežu. Lai aprēķinātu jaudu uz vienu robežvirsmas kvadrātmetru, avota funkciju f_0 pareizinām ar metāla šķērsgriezuma laukumu, iegūstot jaudu uz vienu vada metru, un izdalām ar izolācijas iekšējās robežas perimetru. Iegūto izteiksmi izmantojam robežnosacījumā uz izolācijas iekšējās robežas:

$$k\frac{\partial T}{\partial r} + \frac{A}{2\pi r_0}\,f_0(T) = 0, \quad r = r_0. \tag{10}$$

jeb

$$k\frac{\partial T}{\partial r} + \frac{n\tilde{r}^2}{2r_0}\,f_0(T) = 0, \quad r = r_0. \tag{11}$$

Atkal izmantosim *groovyBC* kodu, lai īstenotu šo robežnosacījumu. Veicam līdzīgus pārveidojumus kā iepriekšējā nodaļā:

$$k\frac{T_{face}-T_{center}}{-\delta} + \frac{I^2\rho_{20}}{2\pi r_0 A}\left(1+\alpha_{20}\left(T_{face}-T_{ref}\right)\right) = 0 \tag{12}$$

$$T_{face} = \frac{1}{1+C}\left(T_{ref} - \frac{1}{\alpha_{20}}\right) + \left(1-\frac{1}{1+C}\right)T_{center}, \tag{13}$$

kur

$$T_{ref} = 20, \quad C = -\frac{k}{\delta\alpha_{20}B}, \quad B = \frac{I^2\rho_{20}}{2\pi r_0 A}. \tag{14}$$

Izveidojam mapes *wire-02* kopiju, nosaucam to par *wire-03* un izdzēšam iepriekš sarēķinātās laika mapes. Datnē *0/T* labojam robežnosacījumu uz virsmas *innerSurface*:

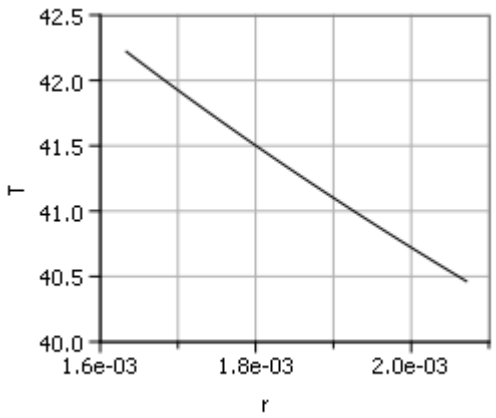
```
    innerSurface
    {
        type            groovyBC;
```

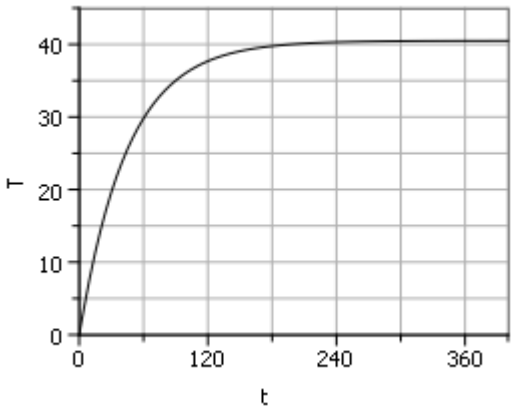
```
        value           uniform 0;
        variables       "I=50;r0=1.625e-3;rMet=0.15e-3;n=84;
A=n*pi*rMet*rMet;k=0.17;rho20=1.724e-8;alpha20=4.04e-3;
B=I*I*rho20/(2*pi*r0*A);C=-k/(B*alpha20*mag(delta()));Tref=20;";
        valueExpression "Tref-1/alpha20";
        gradientExpression "0";
        fractionExpression "1/(1+C)";
    }
```

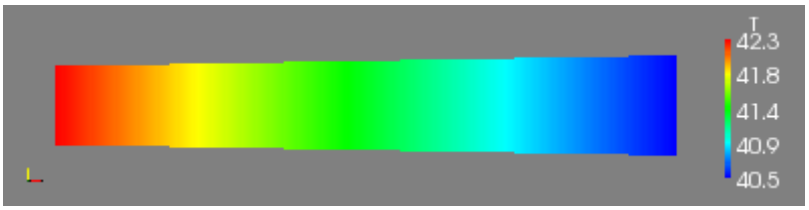
Lai sasniegtu stacionāro stāvokli, palielinām aprēķinu intervālu un beigu laiku, labojot datni *system/controlDict*:

```
endTime         400;

deltaT          0.05;

writeInterval   5;
```

Palaižam aprēķinus ar komandu *laplacianFoam* un pēc tam apskatāmies rezultātus programmā *ParaView*. Attēlos redzams temperatūras sadalījums beigu momentā, kā arī uzkaršanas dinamika.







4. Pilnā 1D problēma ar siltuma avotu un nestacionāriem koeficientiem

Turpinājumā veiksim aprēķinus ne tikai izolācijas apgabalā, bet arī pašā vadītājā, kā arī ņemsim vērā fizikālo koeficientu atkarību no temperatūras. Mums būs nepieciešams labot „solvera" jeb OpenFOAM risinātājprogrammas pirmkodu, pilnveidot modeļa ģeometriju un izmantot papildu „utilītu" koeficientu norādīšanai.

Izejas koda labošana un kompilēšana

Lai īstenotu diferenciālvienādojumu (15), kas atsevišķi apskata siltumietilpību un siltumvadītspēju un satur avota funkciju, labosim iepriekš izmantoto programmu *laplacianFoam*.

$$\frac{\partial c\rho T}{\partial t} - \nabla \cdot k\nabla T = F \tag{15}$$

Izveidojam mapi, kurā glabāsim savas risinātājprogrammas un nokopējam oriģinālo *laplacianFoam* pirmkodu, pārdēvējot to par *laplacianSourceFoam*:

```
  mkdir -p $WM_PROJECT_USER_DIR/applications/solvers/
  cp -r $FOAM_SOLVERS/basic/laplacianFoam $WM_PROJECT_USER_DIR/applications/
solvers/laplacianSourceFoam
```

Pārdēvējam galveno izejas koda datni *laplacianFoam.C* par *laplacianSourceFoam.C*:

```
  cd $WM_PROJECT_USER_DIR/applications/solvers/laplacianSourceFoam
  mv laplacianFoam.C laplacianSourceFoam.C
```

Direktorijas un vārda nomaiņu atspoguļojam kompilācijai nepieciešamajā datnē *Make/files*:

```
  laplacianSourceFoam.C

  EXE = $(FOAM_USER_APPBIN)/laplacianSourceFoam
```

Datni *Make/options* nemainām:

```
  EXE_INC = -I$(LIB_SRC)/finiteVolume/lnInclude

  EXE_LIBS = -lfiniteVolume
```

Datnē *laplacianSourceFoam.C* labosim risināmo diferenciālvienādojumu atbilstoši formulai (15), kā arī ievadīsim koeficientu aprēķināšanas formulas. Siltumietilpības koeficientu norādīsim kā polinomu līdz trešajai pakāpei, avota funkcijai pietiks ar kvadrātisko polinomu, bet siltumvadīšanas koeficients k būs gabaliem konstants, kas ir pietiekami mūsu situācijā. Koeficientus definēsim uz režģa kā skalāros laukus.

```
  #include "fvCFD.H"
  #include "simpleControl.H"

  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

  int main(int argc, char *argv[])
  {

  #include "setRootCase.H"
  #include "createTime.H"
  #include "createMesh.H"
  #include "createFields.H"

  simpleControl simple(mesh)

  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

```
Info<< "\nCalculating temperature distribution\n" << endl;

while (simple.loop())
{
    Info<< "Time = " << runTime.timeName() << nl << endl;

    while (simple.correctNonOrthogonal())
    {
        //siltumietilpības koeficients un avota funkcija
        CRho = CRho0 + CRho1*T + CRho2*sqr(T) + CRho3*pow(T,3);
        F = F0 + F1 * T;

        solve
        (
            //diferenciālvienādojums (15)
            fvm::ddt(CRho, T)
          - fvm::laplacian(K, T)
            ==
            F
        );
    }

#include "write.H"

 Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
     << "  ClockTime = " << runTime.elapsedClockTime() << " s"
     << nl << endl;
    }

    Info<< "End\n" << endl;

    return 0;
}
```

Programmai nepieciešamos parametrus jeb mainīgos definējam datnē *createFields.H*, kur oriģinālais teksts sākot ar rindiņu „`Info<< "Reading diffusivity...`” nebūs vajadzīgs.

```
    Info<< "Reading fields: T, CRho, K, F\n" << endl;

    volScalarField T              //temperatūras lauka definīcija paliek
    (                             //tāda pati kā oriģinālajā datnē
        IOobject
        (
            "T",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    );

    volScalarField K              //siltumvadīšanas koeficients
    (
        IOobject
        (
            "K",                  //datnes nosaukums
            runTime.constant(),   //datne atradīsies mapē constant
            mesh,
            IOobject::MUST_READ,
            IOobject::NO_WRITE    //aprēķinu laikā to uz diska neglabāsim
        ),
        mesh
    );
```

```
    volScalarField CRho          //īpatnējā siltumietilpība, ko programmā
    (                            //aprēķināsim no nākamajiem koeficientiem
        IOobject
        (
            "CRho",
            runTime.timeName(),
            mesh,
            IOobject::NO_READ,
            IOobject::NO_WRITE
        ),
        mesh,
        //Piešķiram 0. Šeit dimensijas sanāk: dimensionSet(1,-1,-2,-1,0,0,0)
        dimensionedScalar("zero", dimSpecificHeatCapacity * dimDensity, 0.0)
    );

    volScalarField CRho0         //siltumietilpības polinoma brīvais loceklis
    (
        IOobject
        (
            "CRho0",
            runTime.constant(),
            mesh,
            IOobject::MUST_READ,
            IOobject::NO_WRITE
        ),
        mesh
    );

    volScalarField CRho1          //siltumietilpības lineārais loceklis
    (
        IOobject
        (
            "CRho1",
            runTime.constant(),
            mesh,
            IOobject::READ_IF_PRESENT, //Ja nav norādīts, piešķiram nulli
            IOobject::NO_WRITE
        ),
        mesh,
        dimensionedScalar("zero", dimensionSet(1,-1,-2,-2,0,0,0), 0.0)
    );

    volScalarField CRho2          //siltumietilpības kvadrātiskais loceklis
    (
        IOobject
        (
            "CRho2",
            runTime.constant(),
            mesh,
            IOobject::READ_IF_PRESENT,
            IOobject::NO_WRITE
        ),
        mesh,
        dimensionedScalar("zero", dimensionSet(1,-1,-2,-3,0,0,0), 0.0)
    );

    volScalarField CRho3          //siltumietilpības kubiskais loceklis
    (
        IOobject
        (
            "CRho3",
            runTime.constant(),
            mesh,
            IOobject::READ_IF_PRESENT,
            IOobject::NO_WRITE
        ),
        mesh,
        dimensionedScalar("zero", dimensionSet(1,-1,-2,-4,0,0,0), 0.0)
```

```
    );

    volScalarField F0            //avota brīvais loceklis
    (
        IOobject
        (
            "F0",
            runTime.constant(),
            mesh,
            IOobject::MUST_READ,
            IOobject::NO_WRITE
        ),
        mesh
    );

    volScalarField F1            //avota lineārais loceklis
    (
        IOobject
        (
            "F1",
            runTime.constant(),
            mesh,
            IOobject::READ_IF_PRESENT,
            IOobject::NO_WRITE
        ),
        mesh,
        dimensionedScalar("zero", dimPower/dimVolume/dimTime, 0.0)
    );

    volScalarField F             //avots
    (
        IOobject
        (
            "F",
            runTime.constant(),
            mesh,
            IOobject::NO_READ,
            IOobject::NO_WRITE
        ),
        F0                       //var veidot no cita mainīgā vai formulas
    );

    //Ja vērtības nevajadzēs saglabāt uz diska, avotu F var definēt arī tā:
    //volScalarField F = F0;
```

Datni *write.H* nemainām:

```
    if (runTime.outputTime())
    {
        volVectorField gradT = fvc::grad(T);

        volScalarField gradTx
        (
            IOobject
            (
                "gradTx",
                runTime.timeName(),
                mesh,
                IOobject::NO_READ,
                IOobject::AUTO_WRITE
            ),
            gradT.component(vector::X)
        );


        volScalarField gradTy
```

```
        (
            IOobject
            (
                "gradTy",
                runTime.timeName(),
                mesh,
                IOobject::NO_READ,
                IOobject::AUTO_WRITE
            ),
            gradT.component(vector::Y)
        );

        volScalarField gradTz
        (
            IOobject
            (
                "gradTz",
                runTime.timeName(),
                mesh,
                IOobject::NO_READ,
                IOobject::AUTO_WRITE
            ),
            gradT.component(vector::Z)
        );

        runTime.write();
    }
```

Pirmkoda datnes ir sagatavotas, un tagad varam tās nokompilēt, izpildot komandu:

```
wmake
```

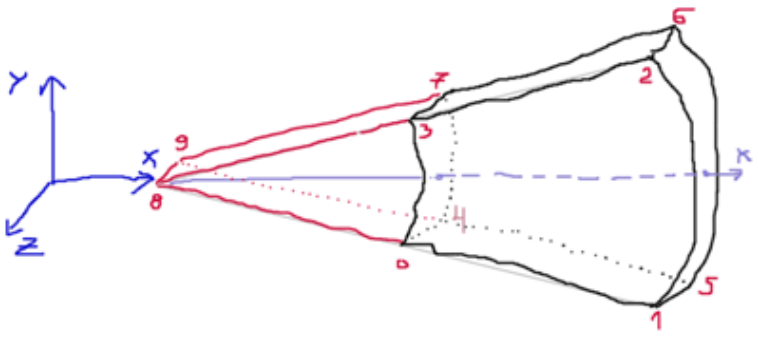
Veiksmīgas kompilācijas gadījumā direktorijā *$FOAM_USER_APPBIN* parādīsies izpildāmā datne *laplacianSourceFoam*.

Piemēra sagatavošana

Vispirms nokopējam iepriekšējā piemēra mapi *wire-03*, nosaucam to par *wire-04* un izdzēšam iepriekš rēķinātās laika mapes. Ja netiek izmantota grafiskā vide, iespējams, ērtāk ir lietot OpenFOAM utilītprogrammu *foamCopySettings*, pēc tam nokopējot arī režģa ģenerēšanai nepieciešamās datnes:

```
cd $FOAM_RUN/wire
mkdir wire-04
foamCopySettings wire-03 wire-04
cp -r wire-03/constant/polyMesh wire-04/constant
```

Ietversim režģī elektriskā vadītāja apgabalu, papildinot datni *constant/polyMesh/blockMeshDict* ar diviem punktiem kā parādīts attēlā un atbilstoši mainot arī pārējās datnes sadaļas.



```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 0.001;

vertices
(
    (1.62475 -0.028360  0.01)
    (2.07468 -0.036214  0.01)
    (2.07468  0.036214  0.01)
    (1.62475  0.028360  0.01)
    (1.62475 -0.028360 -0.01)
    (2.07468 -0.036214 -0.01)
    (2.07468  0.036214 -0.01)
    (1.62475  0.028360 -0.01)
    (0.0      0.0       0.01)          //Punkts nr. 8
    (0.0      0.0      -0.01)          //Punkts nr. 9
);

blocks
(
    hex (4 5 6 7 0 1 2 3) (20 1 1) simpleGrading (1 1 1)
    hex (9 4 7 9 8 0 3 8) (20 1 1) simpleGrading (1 1 1)
);

edges
(
    arc 0 3 (1.625 0  0.01)
    arc 4 7 (1.625 0 -0.01)
    arc 1 2 (2.075 0  0.01)
    arc 5 6 (2.075 0 -0.01)
);

patches
(
    patch outerSurface
    (
        (1 5 6 2)
    )
    symmetryPlane centre      //"patch innerSurface" vietā nāk šīs rindiņas,
    (                         // kas izolācijas iekšējās robežas vietā definē
        (8 9 9 8)             // aksiālās simetrijas robežu
    )
    wedge leftSide
    (
        (3 2 6 7)
        (3 7 9 8)
    )
    wedge rightSide
    (
        (0 4 5 1)
        (0 8 9 4)
    )
    empty frontAndBack
    (
        (0 1 2 3)
        (4 7 6 5)
        (0 3 8 8)
        (4 9 9 7)
     )
);
```

```
mergePatchPairs
(
);
```

No piemēra mapes *wire-04* ģenerējam jauno režģi:

```
blockMesh
```

Uz režģa definējot koeficientu skalāros laukus, sākotnēji izmantosim izolācijas fizikālās īpašības, bet elektrisko vadītāju un avota jaudu norādīsim pēc tam ar utilītprogrammas *setFields* palīdzību. Tā kā šī utilītprogramma maina parametru datnes, vispirms saglabāsim tās ar paplašinājumu „.org", kas nozīmēs *original*.

Izveidojam datni *constant/K.org*, ņemot k vērtību kā formulā (3):

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      K;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [1 1 -3 -1 0 0 0];          // [k] = W/(m·K) = J/(m·s·K) = N/(s·K) = kg·m/(s³·K)

internalField   uniform 0.17;

boundaryField
{
    outerSurface
    {
        type            fixedValue;
        value           uniform 0.17;
    }
    centre
    {
        type            symmetryPlane;
    }
    leftSide
    {
        type            wedge;
    }
    rightSide
    {
        type            wedge;
    }
    frontAndBack
    {
        type            empty;
    }
}
```

Materiālu siltumietilpība dota tabulas veidā[1], ko mēs aproksimēsim ar kubisko polinomu. Koeficientu iegūšanai var izmantot, piemēram, datorprogrammu Maple (izvēlnē: *Tools* > *Assistants* > *Curve Fitting...*). Aprēķiniem nepieciešama tilpumveida siltumietilpība, kas ir blīvuma un „parastās" siltumietilpības reizinājums.

[1] *VDI-Wärmeatlas*. 9. Aufl. Berlin : Springer, 2002

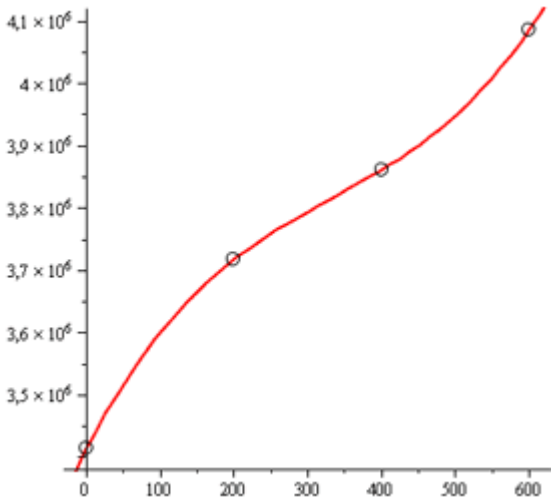
$\rho_{\mathrm{Cu}} = 8960\ \mathrm{kg/m^3}$, $\rho_{\mathrm{PVC}} = 1350\ \mathrm{kg/m^3}$, $[c] = \mathrm{J/(kg \cdot K)}$

	-50 °C	0 °C	50 °C	100 °C	200 °C	400 °C	600 °C
c_{Cu}		381			415	431	456
c_{PVC}	750	920	1040	1530			

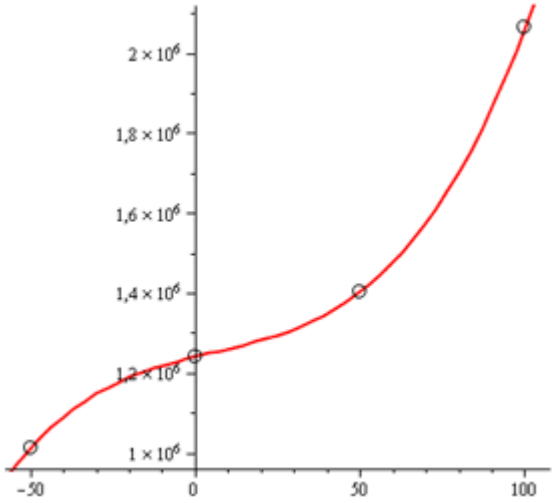
Pareizinot dotos lielumus un aproksimējot, iegūstam sekojošas izteiksmes:

$$c\rho_{\mathrm{Cu}} = 3.414 \cdot 10^6 + 2330T - 5.040T^2 + 5.040 \cdot 10^{-3} T^3 \quad (16)$$

$$c\rho_{\mathrm{PVC}} = 1.242 \cdot 10^6 + 2025T - 13.50T^2 + 0.7560T^3 \quad (17)$$



$c\rho_{\mathrm{Cu}}$



$c\rho_{\mathrm{PVC}}$

Siltumvadīšanas polinoma brīvajam loceklim izveidojam datni *constant/CRho.org*. Saturu ņemam no iepriekš veidotās datnes *K.org*, nomainot objekta nosaukumu, fizikālās dimensijas un pašu vērtību:

```
FoamFile
{
    ...
    object      CRho0;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [1 -1 -2 -1 0 0 0];              // [cρ] = J/(m^3·K) = kg/(m·s^2·K)

internalField   uniform 1.242e6;

boundaryField
{
    outerSurface
    {
        type            fixedValue;
        value           uniform 1.242e6;
    }
...
```

Ņemot vērtības no polinoma (17), līdzīgi izveidojam arī pārējo koeficientu datnes: *CRho1.org*, *CRho2.org* un *CRho3.org*. Lai gala izteiksmei būtu korektas fizikālās

dimensijas, ar katru nākamo pakāpi jānorāda par vienu mazāka temperatūras dimensija, piemēram, kubiskajam loceklim dimensijas ir sekojošas:

```
dimensions      [1 -1 -2 -4 0 0 0];
```

Izolācijā nav siltuma avota, tāpēc funkcijas F visu koeficientu vērtības ir vienādas ar nulli, ko norādām datnēs *F0.org* un *F1.org*. Brīvajam loceklim dimensijas ir sekojošas (pārējiem – attiecīgi par temperatūras dimensiju mazāk)

```
dimensions      [1 -1 -3 0 0 0 0];                  // [F]=W/m^3=kg/(m·s^3)
```

Tagad nokopēsim *org* datnes, atstājot nosaukumu bez šī paplašinājuma, un izmainīsim lauku vērtības tā, lai apgabala vidusdaļā būtu norādītas mūsu elektriskā vadītāja fizikālās īpašības. Lai to automatizētu, piemēra direktorijā izveidojam datni *setValues*:

```
cd constant
cp K.org K
cp CRho0.org CRho0
cp CRho1.org CRho1
cp CRho2.org CRho2
cp CRho3.org CRho3
cp F0.org F0
cp F1.org F1
cp F2.org F2
cd ..
setFields -constant   // norādām, ka lauki ir definēti mapē constant, nevis
                      // kādā laika mapē
```

un piešķiram tai izpildāmās datnes statusu:

```
chmod +x setValues
```

Utilītprogrammai *setFields* veicamās darbības jānorāda datnē *system/setFieldsDict*:

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      setFieldsDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

defaultFieldValues
(
    //volScalarFieldValue K 0.17  //Piemērs, kā izmantot noklusētās vērtības
);

regions
(
    cylinderToCell        // Vērtības tiks mainītas cilindriskā apgabalā,
    {                     // ko nosaka punkti p1 un p2, un rādiuss
        p1 (0 0 -1);
        p2 (0 0 1);
        radius 1.625e-3;  // Izolācijas iekšējais rādiuss
        fieldValues
        (
            volScalarFieldValue K      401
```

```
            volScalarFieldValue CRho0  3.414e6
            volScalarFieldValue CRho1  2330
            volScalarFieldValue CRho2  -5.04
            volScalarFieldValue CRho3  5.04e-3
            volScalarFieldValue F0     8.219e5
            volScalarFieldValue F1     3.409e3
        );
    }
);
```

Vara īpatnējā siltumvadīšana: $k = 401\,\mathrm{W/(m\,K)}$, siltumietilpības vērtības nāk no formulas (16), bet avota funkcijas koeficienti iegūstami pārveidojot formulu (8):

$$f_0(T) = B_0\left(1-\alpha_{20}T_{ref}\right)+B_0\alpha_{20}T = 8.219\cdot 10^5 + 3409T\,, \qquad (18)$$

$$B_0 = \frac{\tilde{r}^2}{r_0^2}\frac{I^2}{A^2}\rho_{20}\,, \qquad I = 50\mathrm{A}\,.$$

Aprēķinot avota funkciju, ņēmām vērā faktu, ka metāla reālais šķērsgriezuma laukums ir mazāks par matemātiskajā modelī izmantoto, tāpēc formulu pareizinājām ar laukumu attiecību, lai ģenerētais siltums būtu tāds pats kā realitātē.

Pabeidzam parametru lauku norādīšanu, no komandrindas piemēra galvenajā mapē izpildot izveidoto skriptu *setValues*:

```
setValues
```

Uz ārējās robežas ņemsim vērā siltumapmaiņas koeficienta atkarību no temperatūras, kurā ietversim arī siltumstarojumu:

$$\alpha = \alpha_{conv} + \alpha_{rad}\,, \qquad \alpha_{rad} = \varepsilon\sigma(\tilde{T}^2+\tilde{T}_\infty^2)(\tilde{T}+\tilde{T}_\infty)\,, \qquad \sigma = 5.6704\cdot 10^{-8}\,\mathrm{W/(m^2\,K^4)}$$

Virsmas starojuma koeficientu ņemam $\varepsilon = 0.93$, temperatūra ar tildes zīmi jārēķina Kelvina grādos, bet konvektīvā siltumapmaiņas koeficienta α_{conv} aprēķināšanai izmantojam sekojošu algoritmu[2]:

$$T = (T_{face}+T_\infty)/2$$
$$l = \pi r_1$$
$$\lambda = 2.434\cdot 10^{-2} + 7.658\cdot 10^{-5}\,T - 4.030\cdot 10^{-8}\,T^2 + 3.101\cdot 10^{-11}\,T^3 - 1.065\cdot 10^{-14}\,T^4$$
$$\nu = 1.352\cdot 10^{-5} + 8.876\cdot 10^{-8}\,T + 1.098\cdot 10^{-10}\,T^2 - 4.035\cdot 10^{-14}\,T^3 + 1.347\cdot 10^{-17}\,T^4$$
$$\mathrm{Pr} = 0.7109 - 1.578\cdot 10^{-4}\,T + 5.885\cdot 10^{-7}\,T^2 - 6.198\cdot 10^{-10}\,T^3 + 2.180\cdot 10^{-13}\,T^4$$
$$\beta = 0.993048/(\mathrm{T}+270.127)$$
$$Gr = g\beta l^3\,|T_{face}-T_\infty|/\nu^2$$
$$Ra = Gr\cdot \mathrm{Pr}$$
$$f = \left(1+\left(0.559/\mathrm{Pr}\right)^{9/16}\right)^{-16/9}$$
$$Nus = \left(0.752+0.387(Ra\cdot f)^{1/6}\right)^2$$
$$\alpha_{conv} = Nus\cdot\lambda/l$$

[2] Ilgevicius A. *Analytical and Numerical Analysis and Simulation of Heat Transfer in Electrical Conductors and Fuses*. Neubiberg : Universitaet der Bundeswehr Muenchen, 2004. Dissertation

Algoritmu implementējam ar *groovyBC* palīdzību temperatūras lauka datnē *0/T*, kurā norādām arī sākotnējo temperatūru iekšējos punktos un parametrus uz pārējām režģa robežām:

```
  FoamFile
  {
      version     2.0;
      format      ascii;
      class       volScalarField;
      object      T;
  }
  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

  dimensions      [0 0 0 1 0 0 0];

  internalField   uniform 0;

  boundaryField
  {
      outerSurface
      {
          type            groovyBC;
          value           uniform 0;
          variables       "r1=2.075e-3;
k=0.17;
epsilon=0.93;
Tinf=0;
TT=(T+Tinf)/2;
g=9.81;
sigma=5.6704e-8;
L=pi*r1;
lambda=2.4340e-2+7.6575e-5*TT-4.0303e-8*pow(TT,2)+3.1098e-11*pow(TT,3)
-1.0648e-14*pow(TT,4);
nu=1.3520e-5+8.8761e-8*TT+1.0979e-10*pow(TT,2)-4.0352e-14*pow(TT,3)
+1.3466e-17*pow(TT,4);
pr=0.71091-1.5775e-4*TT+5.8848e-7*pow(TT,2)-6.1980e-10*pow(TT,3)
+2.1796e-13*pow(TT,4);
beta=0.993048/(TT+270.127);
gr=g*beta*mag(T-Tinf)*pow(L,3)/pow(nu,2);
ra=gr*pr;
f=pow(1+pow(0.559/pr,0.5625),-1.778);
nus=pow(0.752+0.387*pow(ra*f,0.1667),2);
alphaConv=nus*lambda/L;
alphaRad=epsilon*sigma*(sqr(T+273.15)+sqr(Tinf+273.15))*(T+Tinf+2*273.15);
alpha=alphaConv+alphaRad;
C=k/alpha/mag(delta());";
          valueExpression "Tinf";
          fractionExpression "1/(1+C)";
      }
      centre
      {
          type            symmetryPlane;
      }
      leftSide
      {
          type            wedge;
      }
      rightSide
      {
          type            wedge;
      }
      frontAndBack
      {
          type            empty;
      }
  }
```

Uzskatāmības dēļ mainīgie šeit izvietoti viens zem otra, bet datnē visus jāsaraksta aiz atslēgvārda *variables* vienu aiz otra bez atstarpēm.

Datnē *system/fvSchemes* labojam sadaļu *laplacianSchemes*, norādot Laplasa operatoram plūsmas koeficienta nosaukumu:

```
laplacianSchemes
{
    default         none;
    laplacian(K,T)  Gauss linear corrected;
}
```

Lai precīzāk aproksimētu plūsmas nepārtrauktību siltumvadīšanas vienādojumā, Laplasa operatora diskretizācijā būtu jāizmanto koeficienta k interpolēšana, izmantojot harmonisko vidējo vērtību. Tādā gadījumā atslēgvārda *linear* vietā mēs norādītu – *harmonic*. Diemžēl oficiālajā OpenFOAM 2.1.x versijā minētā funkcionalitāte joprojām nav līdz galam pabeigta, bet to iespējams pārnest no *OpenFOAM-Extend* projekta kā pirmkodu, pārkompilējot nepieciešamās datnes.[3]

Tā kā fizikālie koeficienti ir atkarīgi no temperatūras, katrā laika slānī mums jāveic papildu iekšējas iterācijas, kas, patiesībā, jau ir iestrādāts pirmkodā ar rindiņām:

```
   while (simple.correctNonOrthogonal())
   {
      ...
   }
```

Sākotnējās programmas autori to ir veikuši ar mērķi ņemt vērā neortogonālus režģus, bet mūsu vajadzībām šis cikls der bez labojumiem. Papildu iterāciju skaitu palielinām no divām uz piecām, labojot datni *system/fvSolution*:

```
...

SIMPLE
{
    nNonOrthogonalCorrectors  5;
}
```

Vārds *simple* šajā gadījumā norāda nevis uz kaut ko vienkāršu (angļu val. – *simple*), bet algoritmu SIMPLE (Semi-Implicit Method for Pressure-Linked Equations), kas plaši tiek lietots OpenFOAM risinātājprogrammās.

Datni *constant/transportProperties* drīkst izdzēst, jo tā aprēķinos netiek izmantota.

Aprēķini un rezultāti

Datnē *system/controlDict* pamainām aprēķinu soli, beigu laiku un atrādīšanas intervālu, lai temperatūras sadalījums būtu tuvu stacionārai situācijai.

```
FoamFile
{
    version     2.0;
    format      ascii;
```

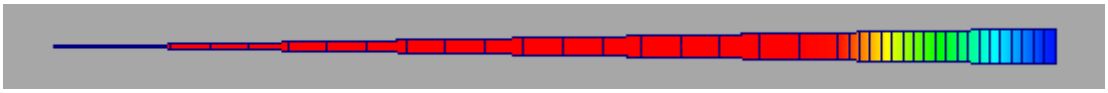
[3] Sk. pamācību http://matmod.pbworks.com/w/browse > OpenFOAM > „Plūsmas nepārtrauktība siltumvadīšanas vienādojumā"

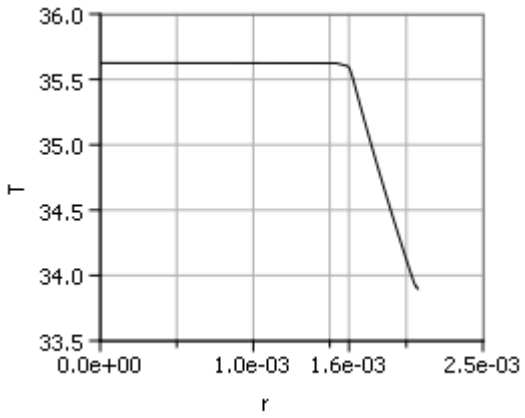
```
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

libs ( "libOpenFOAM.so" "libgroovyBC.so" );

application     laplacianSourceFoam;

startFrom       latestTime;

startTime       0;

stopAt          endTime;

endTime         800;

deltaT          1;

writeControl    runTime;

writeInterval   20;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression uncompressed;

timeFormat      general;

timePrecision   6;

runTimeModifiable yes;
```

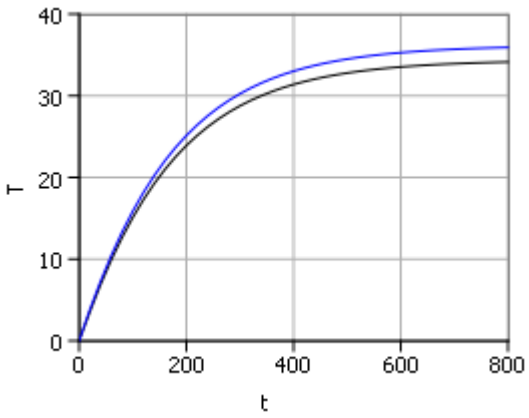
Palaižam aprēķinus, izpildot komandu *laplacianSourceFoam*:

```
laplacianSourceFoam
```

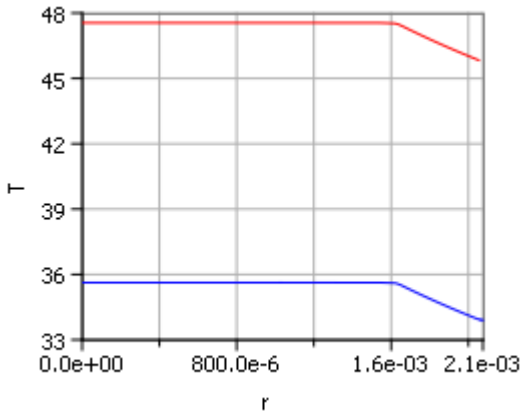
Ar programmu ParaView apskatāmies rezultātus. Temperatūras sadalījums vadā un tā izolācijā parādīts aprēķinu beigu momentā. Pirmajā attēlā redzamas arī režģa līnijas, lai varētu saprast, kur atrodas elektriskā vadītāja un izolācijas robeža. Grafiks, kas attēlo atkarību no laika, rāda temperatūras vērtības gan vada centrā (zilā krāsā), gan uz izolācijas ārējās robežas.

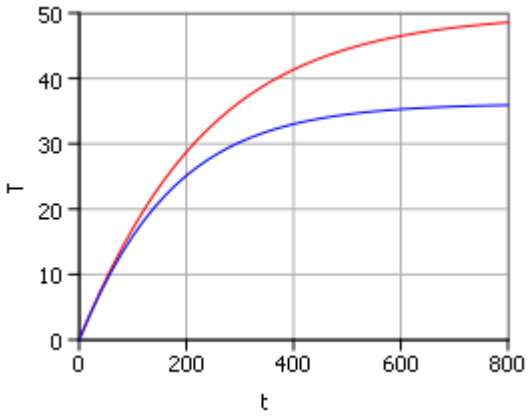






Paskatīsimies, kāda ir atšķirība skaitliskajos rezultātos, ja mēs neņemtu vērā siltumstarojumu. Robežnosacījumos norādām $\varepsilon = 0$ un veicam atkārtotus aprēķinus. Šajā gadījumā elektriskais vads uzkarst vairāk. Abu aprēķinu rezultāti salīdzināti nākamajos grafikos (iepriekšējais rezultāts parādīts ar zilu krāsu, jaunais – ar sarkanu).





Materiāls tapis ar ESF projekta nr. 2009/0223/1DP/1.1.1.2.0/APIA/VIAA/008 atbalstu.