User Manual

Nigel Bess

June 19, 2018

Contents

1 Introduction

Network.m is a tool for solving the transient and equilibrium behavior of thermal networks.

2 Setting up a Network

To begin using Network.m, one must first declare a variable as a network.

```
n = Network;
```

This network must be initialized with some number of nodes. Here n is initialized with 8 nodes:

```
» n = n.Initialize(8);
```

2.1 Connecting and Disconnecting Nodes

2.1.1 Connecting

Nodes of the network are connected via resistances using the `Conn` method. `n.Conn(i,j,r)` connects nodes `i` and `j` with resistance `r`. Here nodes 1 and 2 are connected via a resistance of `150`:

```
» n = n.Conn(1,2,150);
```

One can look up resistance values of a network from the "`r`" property. `n.r(i,j)` will give the resistance between nodes `i` and `j`.

```
» n.r(1,2)
   150;
```

Note: It is highly recommended not to alter the `r` property directly.

By default nodes are connected by infinite resistance:

```
» n.r(2,3)
   Inf;
```

Connecting nodes that have been previously been connected leads to the resistance being added in parallel. Here a resistor with value 300 will be added in parallel to the existing resistor (150) between nodes `1` and `2`.

```
» n = n.Conn(1,2,300);
» n.r(1,2)
   100
```

Connecting nodes without specifying a resistance causes them to be connected with zero resistance.

```
» n = n.Conn(2,4);
» n.r(2,4)
   0
```

2.1.2 Disconnecting

The `DisConn` method is used to disconnect nodes that have previously been connected. `n.DisConn(i,j)` removes the connection between nodes `i` and `j`. Here nodes `2` and `4` will be disconnected:

```
» n = n.DisConn(2,4);
» n.r(2,4);
   Inf
```

By specifying a resistance value, an individual resistor can be removed from a set of parallel resistors. Here the resistor of value 300 will be removed from nodes 1 and 2 (leaving only the resistor of value 150):

```
» n = n.DisConn(1,2,300);
» n.r(1,2)
   150
```

2.2 Isothermal Nodes and Heat Generation

2.2.1 Isothermal Nodes

Isothermal nodes are defined using the `IsoNode` method. Here node `2` will be defined as isothermal with temperature 250:

```
» n = n.IsoNode(2,250);
```

2.2.2 Heat Generation

Nodes can be defined to have constant heat generation using the `HeatGen` method or the `qGen` property. Here node 1 will be defined as having constant heat generation of 25:
Using `HeatGen`:

```
» n = n.HeatGen(1,25);
```

Using `qGen`:

```
» n.qGen(1) = 25;
```

These statements are equivalent.

3 Solving the Network

After setting up all connections and defining isothermal and heat-generating nodes, the network can be solved for temperature and heat flux.

3.1 Equilibrium

To find the equilibrium conditions of the network simply call the `Equilibrium` method:

```
» n = n.Equilibrium;
```

To look up the temperature of any node, use the `"t"` property. `n.t(i)` will give the temperature of node `i`:

```
» n.t(1)

  4000
```

The rate of heat transfer between each node is stored in the `"q"` property. `n.q(i,j)` gives the rate of heat transfer **from** node `i` **to** node `j`

```
» n.q(1,2)

  25
```

3.2 Transient State

To find the transient state of a network, the heat capacity of each node must be defined.
Heat capacity is stored in the `cap` property. `n.cap(i)` contains the heat capacity of node `"i"`. Here node `1` is defined as having a heat capacity of 0.5:

```
» n.cap(1) = 0.5;
```

If heat capacity is not specified, it is given a default value of 0.01.

Once all heat capacities have been set, the transient state of the network is calculated using the `Transient` method. `n.Transient(t)` gives the transient state of the Network at time `t`. Here the transient state of the Network is calculated at time 100:

```
» n = n.Transient(100);
```

Note: the transient state of a network depends on its initial conditions. Make sure to set all initial temperatures by altering the `t` property.

4 Grid Networks

Often times thermal networks take the shape of a grid. Numbering each node, and keeping track of its position can be tricky, so `Network.m` has built in functionality to take care of that for you.

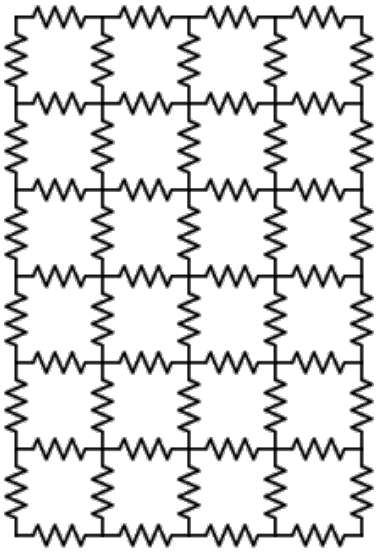


Figure 1: An example of a grid-shaped network

4.1 Node Numbering

In grid networks, nodes are defined by dual-index notation in the form: `[row,column]`

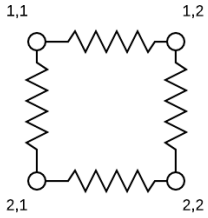


Figure 2: Node numbering for a 2 by 2 grid

Alternately, single-index notation can be used, in which case nodes are numbered left to right, top to bottom.

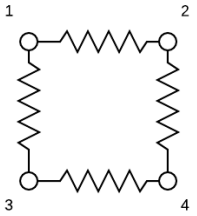


Figure 3: Single-index numbering for a 2 by 2 grid

4.2 Grid Initialization

To demonstrate grid networks, a new variable will be declared as a network:

```
» g = Network;
```

Rather than calling the `Initialize` method, `GridInit` is used to initialize grid networks. `g.GridInit(m,n)` will create a grid with `m` rows and `n` columns. `g.GridInit(m)` will create and `m` by `m` grid. Here `g` is initialized as a 7 by 5 grid:

```
» g = g.GridInit(7,5);
```

4.3 Setting up a Grid Network

The same methods for setting up non-grid networks can still be used. These include `Conn`, `DisConn`, `IsoNode`, and `HeatGen`. The only difference is that dual index notation can now be used.

For example, `g.Conn([i,j],[k,l],r)` connects node `[i,j]` with node `[k,l]`, via resistance `r`. Here node `[2,3]` will be connected with node `[5,4]` via resistance 150:

```
» g = g.Conn([2,3],[5,4],150);
```

This is equivalent to:

```
» g = g.Conn(7,20,150);
```

In the first example, dual-index notation was used, and in the second example single-index was used. Both are valid.

As another example of dual-index notation, node `[4,2]` will be defined as isothermal with temperature 300:

```
» g = g.IsoNode([4,2],300);
```

Note: Dual-index notation can not be used to directly access network properties such as `g.t` or `g.r`. Single-index notation must be used.

4.3.1 Connecting Large portions of a Grid

In order to connect rectangular sections of a grid with the same resistance, the `GridConnect` method can be used. `g.GridConnect(r,[i,j],[k,l])` connects a resistance of `r` between every adjacent node in the rectangle defined by corners `[i,j]` and `[k,l]`.

For example, lets say you wanted to create the following grid of connections in a 7 by 5 grid:

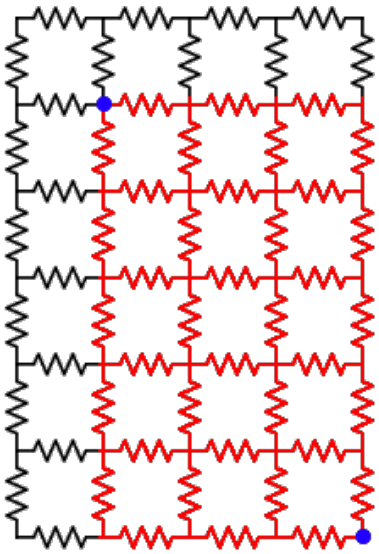


Figure 4: All resistors shown in red have a resistance of 100.

This would be achieved by calling `GridConnect` on the nodes shown in blue.

```
» g = g.GridConnect([2,2],[7,5],100);
```

To connect the entire grid with the same resistance, indices can be left out.

```
» g = g.GridConnect(100);
```

Note: Unlike `Conn`, `GridConnect` overwrites existing connections, rather that connecting in parallel.

To disconnect large portions of the grid, the `GridDisConn` method can be used. `g.GridDisConn([i,j],[k,l])` disconnects all adjacent nodes within the rectangle defined by corners `[i,j]` and `[k,l]`.

4.4 Solving Grid Networks

Grid Networks are solved in the same way as non-grid networks, using the `Equilibrium` and `Transient` methods. However, looking up temperatures, and heat transfer between nodes from the `t` and `q` properties is only possible with single-index notation.

In order to use dual-index notation when looking up temperatures, and to convert the grid of temperatures to matrix form, the `MapGrid` method is used.

```
» g = g.MapGrid;
```

After calling `MapGrid`, the matrix of temperatures can be accessed from the `mappedTemps` property.

Note: Using single-index notation on `mappedTemps` will not return the temperature for the correct node. Matlab's built-in single-index notation for matrices is not the same as the one used in `Network`.

Altering temperatures in `mappedTemps` does not directly change the temperature of the associated nodes. To apply changes made on the `mappedTemps` matrix, use the `DeMapGrid` method.

```
» g.mappedTemps(1) = 100;

» g = g.DeMapGrid;
```

5 Radiative Thermal Networks

Radiative thermal networks can be solved for equilibrium conditions in the same way as regular thermal networks. Transient conditions of radiative networks can not be solved using `Network`.

To solve for equilibrium conditions, simply initiate as a normal network, but use values for black body radiation and radiosity where temperature would normally be used.

There exists a method for initializing radiative networks (`RadInit`), but this should **only** be used for parallel network systems. For radiative networks acting in seclusion use `Initialize`.

6 Parallel Network Systems

In the case of two dependent networks, such as a system undergoing radiative heat transfer as well as convective/conductive heat transfer, the `Equilibrium` and `Transient` methods can no longer be used. Instead, the networks must be iterated simultaneously and information must be passed between them.

Note: Parallel network systems involving radiative heat transfer can not be solved for transient conditions if there is any view factor resistance involved.

Disclaimer: This tool may not be the easiest or most efficient way to solve such networks, but it is possible, and instructions are included.

6.1 Setting Up Parallel Networks

All networks in the system should be set up as outlined in 2 and 4. The only difference is that radiative networks should be initialized using `RadInit`. This takes the same syntax as `Initialize`. Here `r` is initialized as a radiative network with 6 nodes:

```
» r = Network;

» r = r.RadInit(6);
```

The `RadInit` method has the same functionality as `Initialize`, but it sets the `rad` property of the network to `true`. As such, the same can be achieved by normal initialization and altering the `rad` property manually. Here `r` is initialized as a 2 by 3 radiative grid network:

```
» r = Network;

» r = r.GridInit(2,3);

» r.rad = true;
```

The `RadInit` method and the `rad` property should ONLY be used for parallel network systems

6.2 The `Prep` Method

The `Equilibrium` and `Transient` methods automatically find sets of nodes connected by zero resistance (supernodes), and calculate internal time constants for each node, and each supernode. This is done by the `Prep` method, and is absolutely necessary for the network to function.

For parrallel networks, `Prep` must be called manually for each network in the system:

```
» r = r.Prep;
```

This must be called after creating all connections in the network, and before beginning to solve the system.

6.3 Time Constant Calculation

`Prep` chooses an optimal time step for iteration of individual networks, but for a system of networks, the time step must be chosen by the user. After calling `Prep`, a good time step for each network can be found in the `dt` property.

For radiative networks in a system, `dt` is temperature dependent. All initial conditions must be set before calling `Prep`.

The time step should be chosen as large as possible, but small enough that all networks remain stable. A safe choice of time step would be the smallest `dt` of all networks in the system.

Once the time step has been chosen by the user, `dt` should be set for each network. Here `dt` is set to 0.01:

```
» r.dt = 0.01;
```

6.4 Manual Iteration

After all networks have been set up, `Prep` has been called for each network, and an appropriate time step has been chosen, iteration can begin.

Transient heat transfer in each network is calculated with the `CalcHeat` method:

```
» r = r.CalcHeat;
```

After `CalcHeat` has been called for each network, the heat can be applied. This is done using the `ApplyHeat` method:

```
» r = r.ApplyHeat;
```

At this point, the networks are still acing in seclusion. To pass heat between nodes of separate networks, temperature can be altered manually in the `t` property. Make sure to convert black body radiation in radiative networks to temperature, and vice-verse before equilibrating. For usage refer to example 7.3.

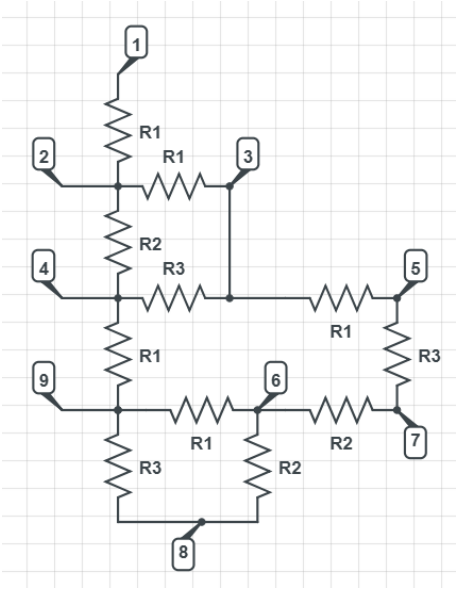
To solve for equilibrium conditions, continue to iterate until temperatures no longer change.

If the system becomes unstable, `dt` might be too large. If the system doesn't change fast enough, `dt` might be too small. If the system contains any radiative networks, increasing `dt` as the network iterates can make it reach equilibrium in less iterations.

7 Examples

7.1 A Conductive Network

Problem: What is the net thermal resistance between node 1 and node 8?



$$
\begin{aligned}
R_1 &= 10\frac{K}{W} \\
R_2 &= 15\frac{K}{W} \\
R_3 &= 50\frac{K}{W}
\end{aligned}
\tag{1}
$$

To solve this problem, node 1 and node 8 will be set to some arbitrary temperatures, and equilibrium heat transfer will be calculated using `Network`. Net resistance, R_{net} will be found from:

$$
Q = \frac{1}{R_{net}}(T_1 - T_8) \tag{2}
$$

T_1 and T_8 can be set to any arbitrary value. In this example, T_1 will be set to 400K and T_8 to 300K.

```
%given
r1 = 10;
r2 = 15;
r3 = 50;

 %arbitrary
t1 = 400;
t8 = 300;

%network setup
n = Network;
n = n.Initialize(9);

%create all connections
n = n.Conn(1,2,r1);
n = n.Conn(2,3,r1);
```

```
n = n.Conn(2,4,r2);
n = n.Conn(4,3,r3);
n = n.Conn(3,5,r1);
n = n.Conn(4,9,r1);
n = n.Conn(5,7,r3);
n = n.Conn(9,6,r1);
n = n.Conn(6,7,r2);
n = n.Conn(9,8,r3);
n = n.Conn(6,8,r2);

%define isothermal nodes
n = n.IsoNode(1,t1);
n = n.IsoNode(8,t8);

%solve for equilibrium
n = n.Equilibrium;

%total heat from node 1 to 8 = heat from node 1 to 2
q = n.q(1,2);
rnet = (t1-t8)/q;
fprintf("Net Thermal Resistance: %f K/W\n\n",rnet)
```

```
Net Thermal Resistance: 42.221551 K/W
```

7.2 A Complex Grid Network

Problem: Find the transient state temperatures of all nodes in the network at time 100s.

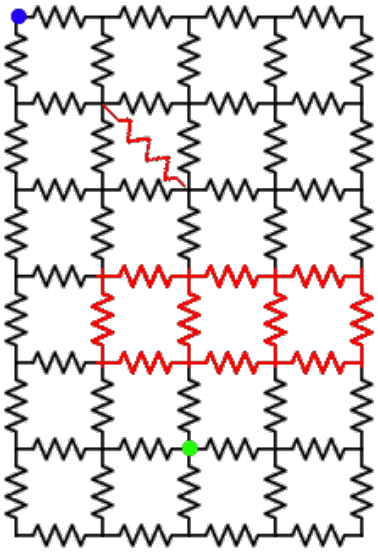


Figure 5: Node `[1,1]` (blue) is isothermal with temperature 300K. Node `[6,3]` (green) generates heat at a rate of 20W

Resistors in black have resistance $R_1 = 10\frac{K}{W}$.
Resistors in red have resistance $R_2 = 50\frac{K}{W}$.

Each node has a heat capacitance of $0.1\frac{J}{K}$.
All nodes have initial temperature 300K.

```
%given
t11 = 300;
q63 = 20;
endTime = 100;
r1 = 10;
r2 = 50;
c = 0.1;

%network setup
g = Network;
g = g.GridInit(7,5);

%initial conditions
g.t(:) = 300;

%heat capacitance
g.cap(:) = c;

%connections
g = g.GridConnect(r1);
g = g.GridConnect(r2,[4 2],[5 5]);
g = g.Conn([2,2],[3,3],r2);

%isothermal node and heat generation
```

```
g = g.IsoNode([1,1],t11);
g = g.HeatGen([6 3],q63);



%solve
g = g.Transient(endTime);
g = g.MapGrid;
disp(g.mappedTemps)
```

$$\begin{pmatrix} 300.0 & 384.44 & 427.62 & 450.79 & 461.14 \\ 395.37 & 425.93 & 448.0 & 464.05 & 471.97 \\ 460.42 & 466.49 & 474.84 & 485.92 & 491.24 \\ 519.81 & 505.23 & 509.19 & 514.08 & 516.36 \\ 594.32 & 624.6 & 682.7 & 660.34 & 647.22 \\ 639.22 & 667.83 & 734.25 & 688.52 & 671.55 \\ 656.23 & 674.0 & 698.73 & 688.72 & 679.73 \end{pmatrix}$$

7.3 A Radiative Network In Parallel With a Convective Network

Problem: Find the steady state temperature of node 2.

Figure 6: Here node 2 conducts to node 1, and undergoes radiative heat transfer with surface 3.

Node 1 has a temperature of $T_1 = 300K$. Node 3 has a black body emission corresponding to 500K. Conductive resistance between node 1 and node 2 is $R_{1,2} = 10\frac{K}{W}$. Radiative pseudo-resistance between node 2 and node 3 is $R_{2,3} = 12m^{-2}$.

```
clc
clear
%given
t1 = 300;
t3 = 500;
r12 = 10;
r23 = 12;

%boltzman constant
sig = 5.67E-8;

%initialize networks
c = Network; % conductive
r = Network; % radiative
c = c.Initialize(2);
r = r.RadInit(2);

%connections
c = c.Conn(1,2,r12);
r = r.Conn(1,2,r23);

%isothermal nodes
c = c.IsoNode(1,t1);
r = r.IsoNode(2,sig*(t3^4));%black body radiation

%set initial conditions to solve for time constants.
%I will choose t2 = 400 as an initial condition. (arbitrary)
c.t(2) = 400;
r.t(1) = sig*(400^4);

%prep
c = c.Prep;
r = r.Prep;

%choose the smallest recommended time step
dt = min([c.dt r.dt]);
c.dt = dt;
r.dt = dt;

for i = 1:10000%manually iterate 10000 times (arbitrary large number)
c = c.CalcHeat;
r = r.CalcHeat;
c = c.ApplyHeat;
r = r.ApplyHeat;
```

```
%convert black body radiation to temperature
t2Rad = (r.t(1)/sig)^(1/4);

%equilibriate temperature between the two networks
tAv = (t2Rad+c.t(2))/2;

c.t(2) = tAv;
r.t(1) = sig*(tAv^4);
end
fprintf("Equilibrium Temperature of Node 2: %f K\n\n",c.t(2))
```

```
Equilibrium Temperature of Node 2:  491.681191 K
```
(3)