

ME 201/MTH 281/ME400/CHE400 Newton's Law of Cooling *Mathematica 7*

■ 1. INTRODUCTION

This notebook uses *Mathematica* to solve the problem presented in class, in section 5.1 of the notes. The problem is one of transient heat conduction in a slab of finite width L . The boundary condition on the left face of the slab is zero heat flux, and the condition on the right face of the slab is Newton's law of cooling, with an ambient temperature T_a . The initial temperature is the function $T_0(x)$. The relevant material properties are the thermal conductivity k , the thermal diffusivity D_f , and the heat transfer coefficient h . The specific expressions for all functions and parameters for this problem are given in the section 2. The mathematical formulation of the problem is given below.

$$\frac{\partial T}{\partial t} = D_f \frac{\partial^2 T}{\partial x^2}, \quad 0 < x < L, \quad t > 0,$$
$$\text{with } \frac{\partial T}{\partial x}(0, t) = 0, \quad k \frac{\partial T}{\partial x}(L, t) + h[T(L, t) - T_a] = 0, \quad (1)$$
$$\text{and } T(x, 0) = T_0(x).$$

■ 2. PARAMETER VALUES

In this section, we define the parameter values and the initial function. This is the only place in the notebook where these quantities are given specifically. This makes it possible to change a value for the entire calculation by just changing it here. The material properties are appropriate for granite. The initial temperature is taken to be a constant.

$$\mathbf{h} = 22.4; \quad (** \text{ W/m}^2 \cdot \mathbf{K} **)$$

$$\mathbf{L} = 0.5; \quad (** \text{ m} **)$$

$$\mathbf{k} = 2.80; \quad (** \text{ W/m} \cdot \mathbf{K} **)$$

$$\mathbf{Df} = 1.37 * 10^{-6}; \quad (** \text{ m}^2/\mathbf{s} **)$$

$$\mathbf{TA} = 10.0; \quad (** \text{ }^\circ\mathbf{C} **)$$

$$\mathbf{T0[x_]} = 60.0; \quad (** \text{ }^\circ\mathbf{C} **)$$

■ 3. STEADY STATE SOLUTION

As in all other problems of this sort, we must first find the steady-state solution $T_s(x)$, and then reformulate the problem in terms of the transient solution T_r . The steady-state solution is obtained by setting the time derivative to zero in the original equation for T . The solution of the resulting equation is a linear function of x , and we require that particular linear function which satisfies the boundary conditions at $x = 0$ and L . The result (which is intuitively obvious) is that $T_s(x)$ is constant and equal to the ambient temperature:

$$\mathbf{T_s [x_]} := \mathbf{T_A}$$

■ 4. FORMULATION OF PROBLEM FOR TRANSIENT

Now that we have the steady-state solution, we decompose the full solution into a steady-state part $T_s(x)$ and a transient part $T_r(x,t)$:

$$T(x,t) = T_s(x) + T_r(x,t) . \quad (2)$$

By substituting this decomposition into the original equation for T , we find the following problem for the transient:

$$\begin{aligned} \frac{\partial T_r}{\partial t} &= D_f \frac{\partial^2 T_r}{\partial x^2}, \quad 0 < x < L, \quad t > 0, \\ \text{with } \frac{\partial T_r}{\partial x}(0, t) &= 0, \quad k \frac{\partial T_r}{\partial x}(L, t) + h T_r(L, t) = 0, \end{aligned} \quad (3)$$

$$\text{and } T_r(x, 0) = T_0(x) - T_s(x) .$$

This is the problem that we solve by separation of variables.

■ 5. SEPARATION OF VARIABLES IN TRANSIENT PROBLEM

Following the standard approach in separation of variables, we seek functions which satisfy the equation for T_r and the homogeneous boundary conditions at $x = 0, L$. We assume a form $F(x)G(t)$ for such solutions. The two separated equations then have the form

$$\begin{aligned} F''(x) + \lambda F(x) &= 0, \quad \text{with } F'(0) = 0 \text{ and } kF'(L) + hF(L) = 0, \\ \text{and } G'(t) + \lambda DG(t) &= 0 . \end{aligned} \quad (4)$$

The general solution for the equation for $F(x)$ is a linear combination of $\sin(\sqrt{\lambda} x)$ and $\cos(\sqrt{\lambda} x)$. Applying the two homogeneous boundary conditions to the general solution, and simplifying the resulting eigenvalue equation, we get for the eigenvalues

$$\mathbf{lam [n_]} := (\mathbf{z / L})^2 / . \mathbf{z} \rightarrow \mathbf{z [n]}$$

where the values $z[1], z[2], \dots, z[n], \dots$ are the roots of $\text{eig1} = \text{eig2}$, where

$$\mathbf{eig1} = \mathbf{Tan [z]};$$

```
eig2 = Bi / z;
```

with

```
Bi := (h * L) / k
```

The generic eigenfunction is

```
geneig[x_] := Cos[(z * x) / L]
```

and the nth eigenfunction is given by

```
F eig[x_, n_] := geneig[x] /. z -> z[n]
```

The generic normalization integral is

```
nor = Integrate[(geneig[x])^2, {x, 0, L}]
```

$$0.25 + \frac{0.125 \sin[2. z]}{z}$$

and the nth normalization integral is

```
norm[n_] := nor /. z -> z[n]
```

■ 6. DETERMINATION OF EIGENVALUES

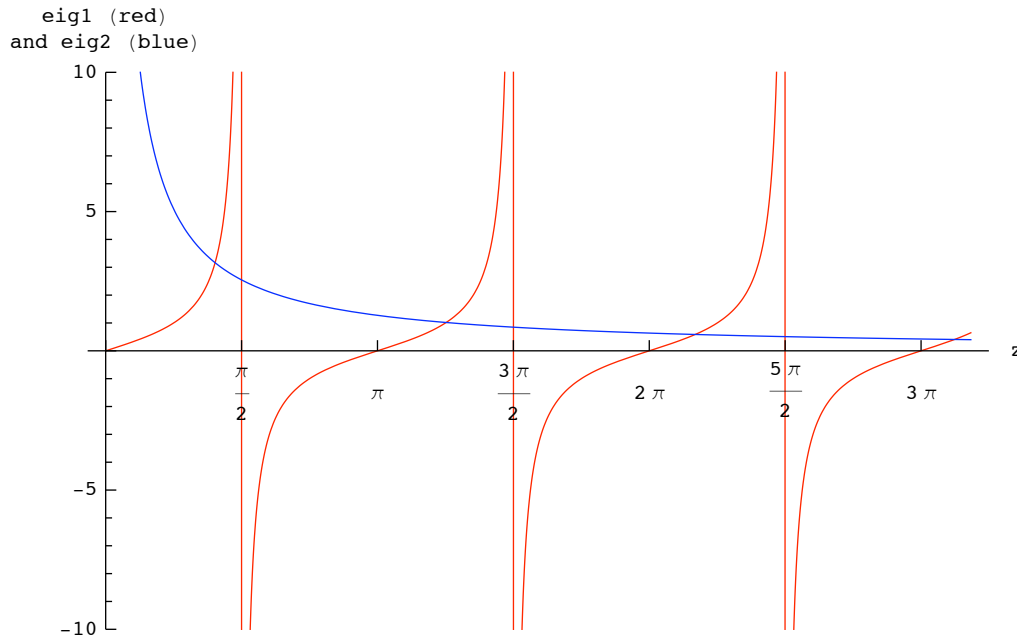
We now determine numerically the eigenvalues. We begin by plotting the expressions eig1 and eig2. At each crossing of these, there is an eigenvalue.

```
SetOption[Plot, ImageSize -> 250];
```

```

Plot[{eig1, eig2}, {z, 0, 10}, PlotRange -> {-10, 10},
  PlotStyle -> {RGBColor[1, 0, 0], RGBColor[0, 0, 1]},
  AxesLabel -> {"z", "eig1 (red)
and eig2 (blue)"}, Ticks -> {{0, Pi/2, Pi, 3 Pi/2, 2 Pi, 5 Pi/2, 3 Pi}}]

```



We see that the first root is between 0 and $\pi/2$, and each subsequent root is in an interval of length π . It is also clear that for large n , the n th root is very close to $(n-1)\pi$. We now use a Do loop to calculate and display in a table the first 40 roots. This will suffice for any calculation requiring up to 40 terms in the eigenfunction series. We also display the eigenvalue λ , and the approximate value $z = (n-1)\pi$.

```

zroot[n_] := FindRoot[eig1 == eig2, {z, (n - 0.5) * Pi - 0.1}]

```

```

zasymp[n_] := N[(n - 1) * Pi]

```

```

Do[z[n] = z /. zroot[n], {n, 1, 40}];

```

```

TableForm[
  Table[{n, PaddedForm[z[n], {10, 4}], PaddedForm[zasympt[n], {10, 4}],
    PaddedForm[lam[n], {10, 4}]}, {n, 1, 40}], TableHeadings ->
  {None, {"n", "      z[n]", "      zasympt[n]", "      lam[n]"}}]

```

n	z[n]	zasympt[n]	lam[n]
1	1.2646	0.0000	6.3968
2	3.9352	3.1416	61.9420
3	6.8140	6.2832	185.7229
4	9.8119	9.4248	385.0918
5	12.8678	12.5664	662.3165
6	15.9536	15.7080	1018.0727
7	19.0565	18.8496	1452.5939
8	22.1697	21.9911	1965.9744
9	25.2896	25.1327	2558.2574
10	28.4142	28.2743	3229.4647
11	31.5421	31.4159	3979.6082
12	34.6724	34.5575	4808.6949
13	37.8045	37.6991	5716.7291
14	40.9381	40.8407	6703.7134
15	44.0728	43.9823	7769.6496
16	47.2084	47.1239	8914.5391
17	50.3448	50.2655	10138.3827
18	53.4817	53.4071	11441.1809
19	56.6192	56.5487	12822.9343
20	59.7571	59.6903	14283.6432
21	62.8954	62.8319	15823.3079
22	66.0339	65.9734	17441.9285
23	69.1728	69.1150	19139.5052
24	72.3119	72.2566	20916.0381
25	75.4512	75.3982	22771.5274
26	78.5907	78.5398	24705.9731
27	81.7303	81.6814	26719.3752
28	84.8701	84.8230	28811.7339
29	88.0100	87.9646	30983.0491
30	91.1500	91.1062	33233.3210
31	94.2902	94.2478	35562.5495
32	97.4304	97.3894	37970.7346
33	100.5707	100.5310	40457.8764
34	103.7111	103.6726	43023.9750
35	106.8516	106.8142	45669.0302
36	109.9921	109.9557	48393.0422
37	113.1327	113.0973	51196.0109
38	116.2733	116.2389	54077.9364
39	119.4140	119.3805	57038.8186
40	122.5547	122.5221	60078.6576

We see that the asymptotic formula for $z[n]$ has an error of less than 0.5% for any n equal to or greater than 10. It is possible to develop even more accurate, but still simple, asymptotic formulas.

7. REPRESENTATION OF THE INITIAL CONDITION

We are finally ready to begin solving the boundary value for the transient $T_r(x,t)$. The form of the solution for the transient is

$$T_r(x,t) = \sum_{n=1}^{\infty} C(n) \text{Exp}(-\lambda(n)D_f t) \cos[(z(n)x/L], \quad (5)$$

where the coefficients $C(n)$ are determined by the initial conditions satisfied by T_r . The generic formula for the n th coefficient is

$$\mathbf{coeff}[\mathbf{n}_] := \text{Integrate}[\mathbf{Feig}[\mathbf{x}, \mathbf{n}] * (\mathbf{T0}[\mathbf{x}] - \mathbf{Ts}[\mathbf{x}]), \{\mathbf{x}, \mathbf{0}, \mathbf{L}\}] / \mathbf{norm}[\mathbf{n}]$$

We now use a Do loop to construct the first 40 coefficients, and then print them out in a table. The n th numerical value is assigned to $c[n]$.

$$\mathbf{Do}[\mathbf{c}[\mathbf{n}] = \mathbf{coeff}[\mathbf{n}], \{\mathbf{n}, \mathbf{1}, \mathbf{40}\}];$$

```
TableForm[Table[{n, PaddedForm[c[n], {8, 4}]}, {n, 1, 40}],
  TableHeadings -> {None, {"n", "      c[n]"}}]
```

n	c[n]
1	61.4354
2	-16.0732
3	6.9821
4	-3.7151
5	2.2572
6	-1.5022
7	1.0667
8	-0.7947
9	0.6140
10	-0.4882
11	0.3973
12	-0.3295
13	0.2776
14	-0.2370
15	0.2047
16	-0.1785
17	0.1571
18	-0.1393
19	0.1243
20	-0.1116
21	0.1008
22	-0.0915
23	0.0834
24	-0.0763
25	0.0701
26	-0.0646
27	0.0598
28	-0.0554
29	0.0516
30	-0.0481
31	0.0449
32	-0.0421
33	0.0395
34	-0.0371
35	0.0350
36	-0.0330
37	0.0312
38	-0.0296
39	0.0280
40	-0.0266

As a check on all that we have done, we see how well our series represents the initial condition, by plotting both the initial condition (in blue) and its representation by the first 40 terms of our series (in red).

```
exactinit[x_] := T0[x] - Ts[x]
```

```

seriesinit[x_] := Sum[c[n] * Feig[x, n], {n, 1, 40}]

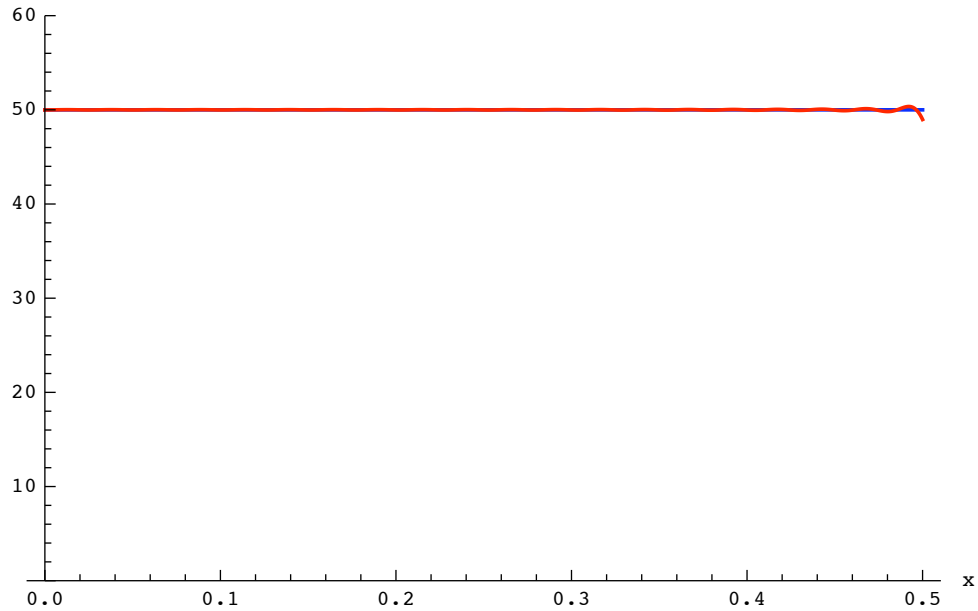
seriesgraph =
  Plot[{exactinit[x], seriesinit[x]}, {x, 0, L}, PlotRange -> {0, 60},
    PlotStyle -> {{RGBColor[0, 0, 1], Thickness[0.004]},
      {RGBColor[1, 0, 0], Thickness[0.004]}}},
    AxesLabel -> {"x", "Exact (blue)",
  Series (red)"}]

```

```

Exact (blue),
Series (red)

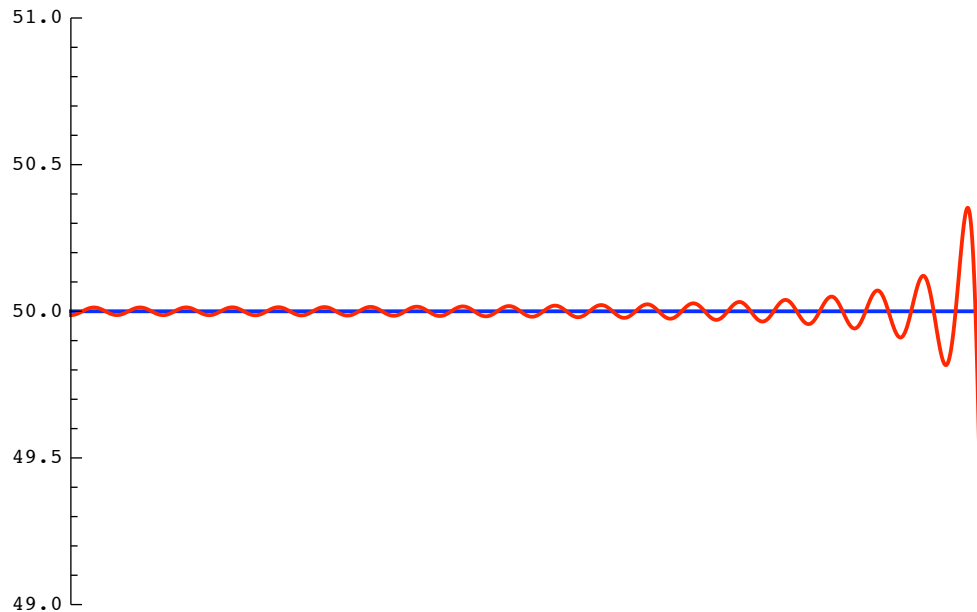
```



The agreement looks excellent, and is good evidence that our calculations are correct so far. To get a little closer view of the accuracy, we replot with a much narrower range in y .

```
Show[seriesgraph, PlotRange -> {49, 51}]
```

```
Exact (blue),  
Series (red)
```



Now we can see a little overshoot at the right end of the interval. This is not the Gibbs phenomenon, however, because the magnitude of the overshoot decreases to zero as the number of terms is increased. We haven't shown that here, but it can be shown. With some algebraic labor, it also can be shown that the coefficients $c[n]$ diminish like $1/n^2$, and not like the $1/n$ that would be associated with the Gibbs overshoot.

■ 8. SOLUTION OF THE INITIAL-BOUNDARY VALUE PROBLEM

Now that we have the eigenvalues and series coefficients, we can construct the series solution of the problem. We define $\text{Tran}[x,t,n]$ as the n th partial sum of the series solution, and we define $\text{term}[x,t,n]$ as the n th term in the series. As a special case of importance, we define $\text{firstterm}[x,t]$ to be the first term in the series.

```
term[x_, t_, n_] := c[n] * Feig[x, n] * Exp[-lam[n] * Df * t]
```

```
Tran[x_, t_, n_] := Sum[term[x, t, k], {k, 1, n}]
```

```
firstterm[x_, t_] := term[x, t, 1]
```

Let's look at the first term:

```
firstterm[x, t]
```

```
61.4354 e-8.76357×10-6 t Cos[2.52918 x]
```

We may calculate the e-folding time for this mode -- call it τ_1 -- as the reciprocal of the coefficient of t in the exponent:

$$\tau_1 = 1 / (\text{lam}[1] * \text{Df})$$

114 109.

This time, which is in seconds, equals about 31.7 hours. We compare this with the diffusion time $L^2 / (\pi^2 D_f)$ we derived earlier for simpler boundary conditions:

$$L^2 / (\pi^2 \text{Df})$$

18 489.3

It is interesting that the first decay time is considerably longer than our earlier estimate of diffusion time. This happens because in the present problem there is thermal resistance at the boundary (characterized by the reciprocal of the heat transfer coefficient h) whereas in the earlier problem in which the temperature was specified on the boundary, there was no resistance to transfer from the surface to the surroundings. The extra resistance at the boundary in the present case gives rise to a longer decay time.

Let's look at the decay times of the second and third modes, converting them to hours as we calculate them

$$\tau_2 = 1 / (\text{lam}[2] * \text{Df} * 3600)$$

3.27335

$$\tau_3 = 1 / (\text{lam}[3] * \text{Df} * 3600)$$

1.09172

Knowledge of these decay times will help us choose time values for plotting the solution. To make this a little easier, we construct a table of the first 40 decay times.

$$\tau[\mathbf{n_}] := 1 / (\text{lam}[\mathbf{n}] * \text{Df})$$

```

TableForm[
  Table[{n, PaddedForm[ $\tau$ [n], {10, 2}], PaddedForm[ $\tau$ [n] / 3600, {10, 5}]},
    {n, 1, 40}], TableHeadings ->
  {None, {"Mode", "Decay Time (s)", "Decay Time (hr)"}}]

```

Mode	Decay Time (s)	Decay Time (hr)
1	114108.73	31.69687
2	11784.04	3.27335
3	3930.19	1.09172
4	1895.46	0.52652
5	1102.08	0.30613
6	716.97	0.19916
7	502.50	0.13958
8	371.28	0.10313
9	285.32	0.07926
10	226.02	0.06278
11	183.42	0.05095
12	151.79	0.04216
13	127.68	0.03547
14	108.88	0.03025
15	93.95	0.02610
16	81.88	0.02274
17	72.00	0.02000
18	63.80	0.01772
19	56.92	0.01581
20	51.10	0.01420
21	46.13	0.01281
22	41.85	0.01162
23	38.14	0.01059
24	34.90	0.00969
25	32.05	0.00890
26	29.54	0.00821
27	27.32	0.00759
28	25.33	0.00704
29	23.56	0.00654
30	21.96	0.00610
31	20.53	0.00570
32	19.22	0.00534
33	18.04	0.00501
34	16.97	0.00471
35	15.98	0.00444
36	15.08	0.00419
37	14.26	0.00396
38	13.50	0.00375
39	12.80	0.00355
40	12.15	0.00337

■ 9. GRAPHS OF THE SOLUTION

We define here graphs of the solution for T versus x , for various values of time. We test the code with a few graphs, and then we construct a sequence to be animated to show the evolution of the system with time. The function which produces the graph at time t is named `snapshot[t]`. We use all 40 terms that we have calculated, although this is somewhat extravagant in that fewer terms would be sufficient for the longer times.

```
Trans[x_, t_] := Tran[x, t, 40]
```

```
Temp[x_, t_] := Trans[x, t] + Ts[x]
```

Now we are ready to define `snapshot[t]`, which gives us a plot of temperature versus x at time t . From the ambient temperature and the initial distribution, we can get a range of variation of temperature. We extend this range slightly on either side, and call the extended range `plrange`:

```
plrange = {0, 70};
```

Now we define `snapshot[t]`, giving a special definition for `snapshot[0]` -- namely the initial distribution. For convenience, the function `snapshot` takes an argument in hours, but converts it internally to seconds, to be consistent with the units of the problem.

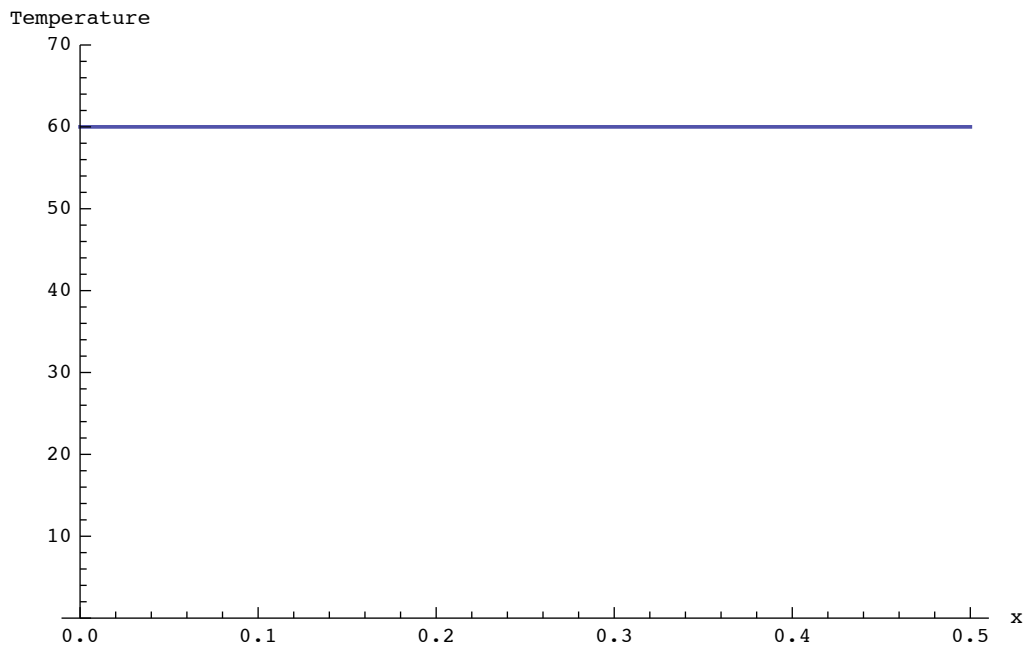
```
snapshot[t_] := Module[{time}, time = 3600 * t;
  Plot[Temp[x, time], {x, 0, L}, PlotStyle → Thickness[0.004],
  PlotRange -> plrange, AxesLabel -> {"x", "Temperature"},
  PlotLabel → Row["t = ", PaddedForm[t, {4, 2}], " hr"]]]]

snapshot[0] := Plot[T0[x], {x, 0, L}, PlotStyle → Thickness[0.004],
  PlotRange -> plrange, AxesLabel -> {"x", "Temperature"},
  PlotLabel → Row["t = ", PaddedForm[0, {4, 2}], " hr"]]]]
```

We try it for the initial time and for 1, 10, 20, and 30 hr.

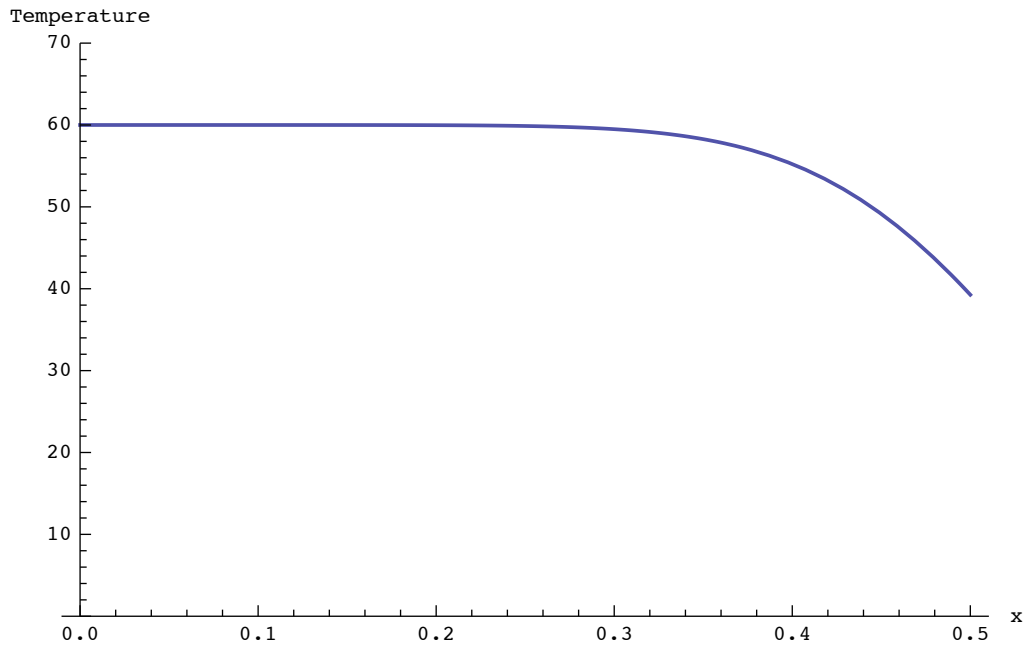
snapshot [0]

t = 0.00 hr



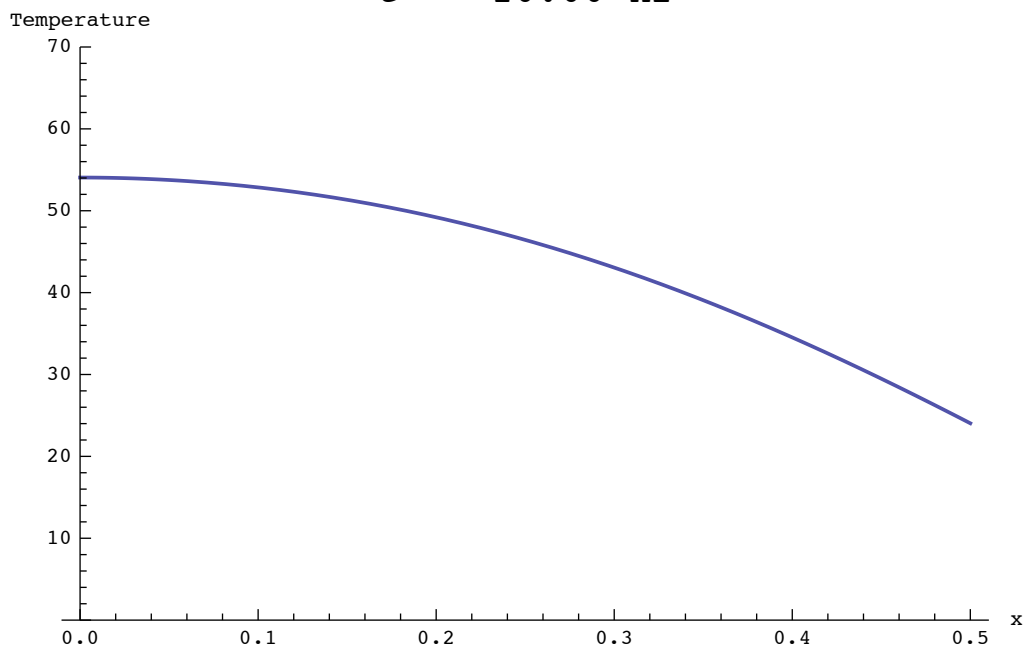
snapshot [1]

t = 1.00 hr



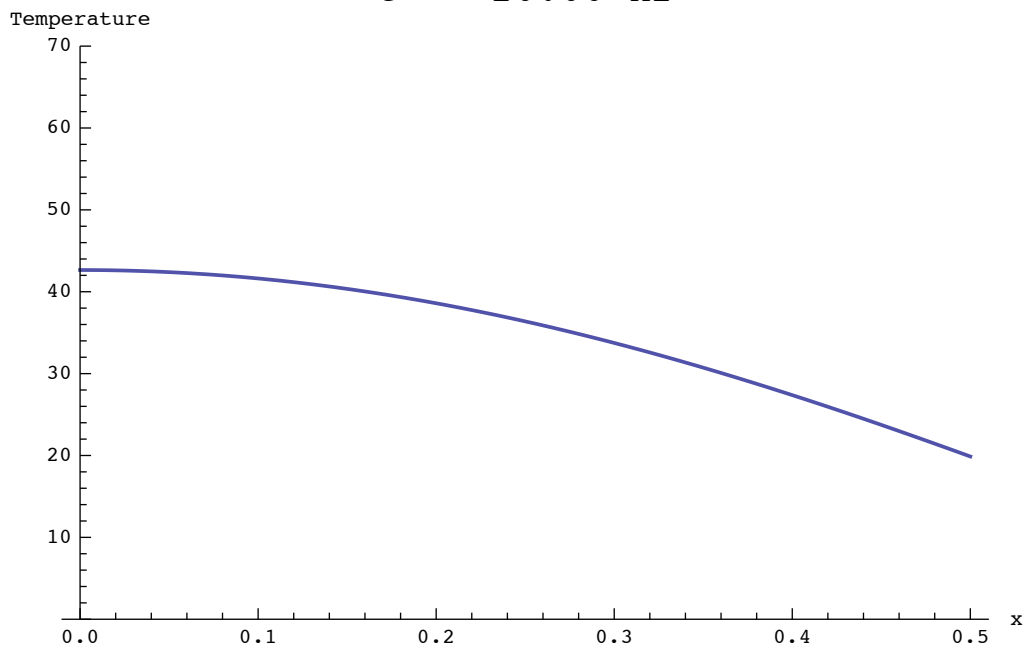
snapshot [10]

$t = 10.00$ hr



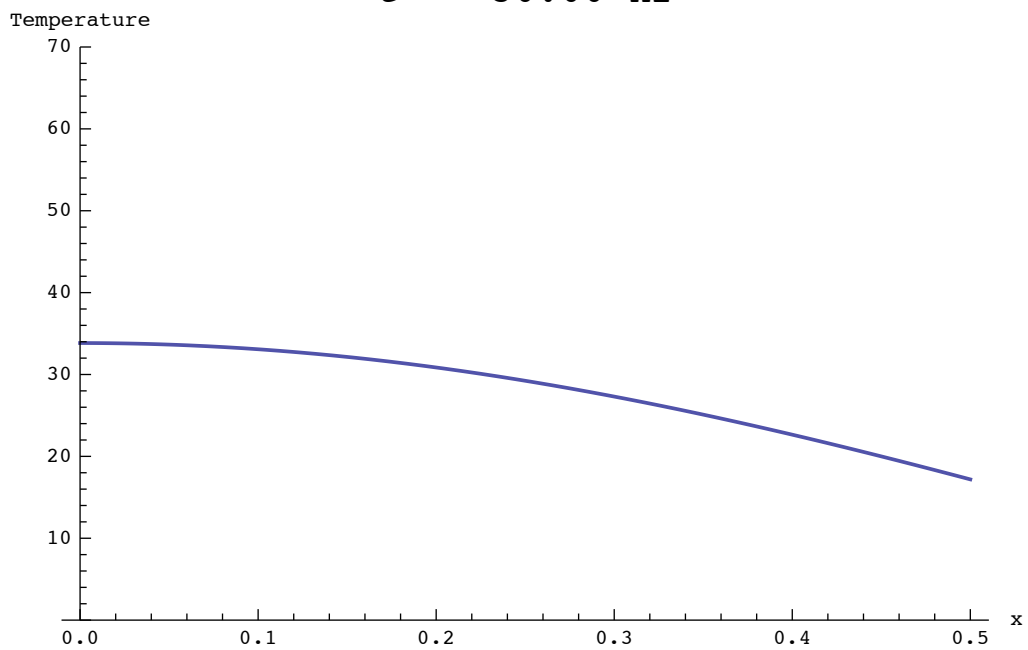
snapshot [20]

$t = 20.00$ hr



snapshot [30]

$t = 30.00$ hr



As a final task, we produce a sequence of graphs that can be animated to show the dynamics of the heat loss process. We produce a sequence of 101 graphs at 0.5 hr intervals, running from 0 hr to 50 hr. In the printed version of the notebook, only one graph in the sequence is shown.

```
Do[Print[snapshot[0.5 * i]], {i, 0, 100}];
```

$t = 0.00$ hr

