The One Dimensional Heat Equation - Implicit Schemes

```
> restart; with(plots): with(LinearAlgebra):
```

The one dimensional heat conduction

If u(x,t) is the temperature at position x and time t the one dimensional **heat equation** is given by:

```
> heat := diff(u(x,t),t)=diff(u(x,t),x,x);
```

$$heat := \frac{\partial}{\partial t} u(x,t) = \frac{\partial^2}{\partial x^2} u(x,t) \tag{1.1}$$

In the lectures, we have found the particular solution

```
> u_part := (x,t) -> exp( -(k*Pi)^2 *t) * sin(k*Pi*x);
```

$$u\_part := (x,t) \rightarrow e^{-k^2 \pi^2 t} \sin(k \pi x) \tag{1.2}$$

If k is an integer,

```
> k := 1;
```

$$k := 1 \tag{1.3}$$

we can see that u_part satifies the heat equation

```
> subs(u(x,t)=u_part(x,t),heat);
> simplify(%);
```

$$\frac{\partial}{\partial t}\left(e^{-\pi^2 t} \sin(\pi x)\right) = \frac{\partial^2}{\partial x^2}\left(e^{-\pi^2 t} \sin(\pi x)\right)$$

$$-\pi^2 e^{-\pi^2 t} \sin(\pi x) = -\pi^2 e^{-\pi^2 t} \sin(\pi x) \tag{1.4}$$

and the following boundary conditions:

```
> u_0 := u_part(0,t); u_1 := u_part(1,t);
```

$$u\_0 := 0$$

$$u\_1 := 0 \tag{1.5}$$

and initial conditions:

```
> u_part(x,0);
> u_init := unapply( eval(u_part(x,0)), x);
```

$$\sin(\pi x)$$

$$u\_init := x \rightarrow \sin(\pi x) \tag{1.6}$$

Numerical solution using an implicit scheme

Similar to the explicit scheme, we need to define the number of unknowns, the mesh size, and the size and number of the time steps:

```
> n := 20; h := 1/n;
```

$$n := 20$$

$$h := \frac{1}{20} \tag{2.1}$$

```
> tau := 1/40; steps := 10;
```

$$\tau := \frac{1}{40}$$

$$steps := 10 \tag{2.2}$$

```
> v := array(0..n,0..steps);
```

$$v := array(0..20, 0..10, [\,]) \tag{2.3}$$

Then, we initialize our numerical solution:

```
> for j from 1 to n-1 do
     v[j,0] := u_init(j*h)
  end do:
```

and set the boundary conditions:

```
> for m from 0 to steps do
     v[0,m] := 0;
     v[n,m] := 0;
  end do:
```

Now, before we can start to compute the numerical solution, we need to set up the iteration matrix.

```
> tridiag := (i,j) -> `if`(i=j,2,
                          `if`(i=j+1 or i+1=j, -1,0));
  A := Matrix(n-1,tridiag);
```

$$tridiag := (i, j) \rightarrow if\,(i = j, 2, if\,(i = j + 1 \ \textbf{or}\ i + 1 = j, -1, 0))$$

$$A := \begin{bmatrix} 19\ x\ 19\ Matrix \\ Data\ Type:\ anything \\ Storage:\ rectangular \\ Order:\ Fortran\_order \end{bmatrix} \tag{2.4}$$

```
> M := Matrix(n-1,shape=identity) + tau/h^2*A;
```

$$M := \begin{bmatrix} 19\ x\ 19\ Matrix \\ Data\ Type:\ anything \\ Storage:\ rectangular \\ Order:\ Fortran\_order \end{bmatrix} \tag{2.5}$$

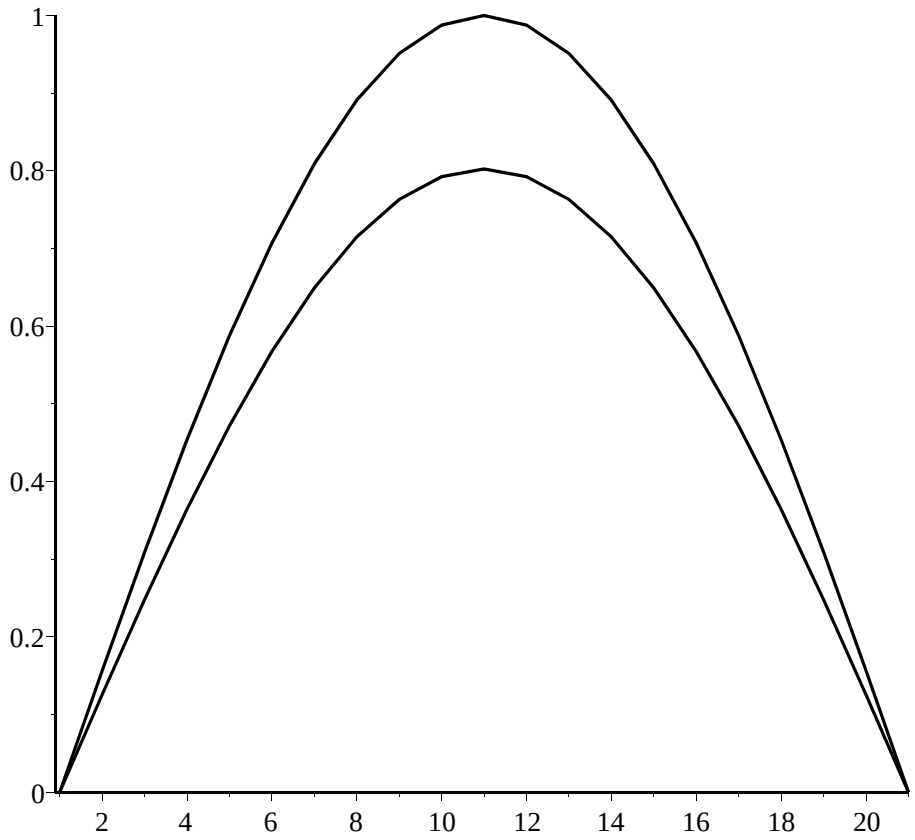
To compute the numerical solution, we can then use the loop

```
> tmp := LinearSolve(M, <seq(v[j,0],j=1..n-1)>);
  for j from 1 to n-1 do
     v[j,1] := tmp[j]:
  end do:
```

$$tmp := \begin{bmatrix} 1\ ..\ 19\ Vector_{column} \\ Data\ Type:\ anything \\ Storage:\ rectangular \\ Order:\ Fortran\_order \end{bmatrix} \tag{2.6}$$

We may take a little time to plot the results:

```
> plot1 := listplot( [seq( v[j,0], j=0..n)] ):
  plot2 := listplot( [seq( v[j,1], j=0..n)] ):
  display(plot1,plot2);
```

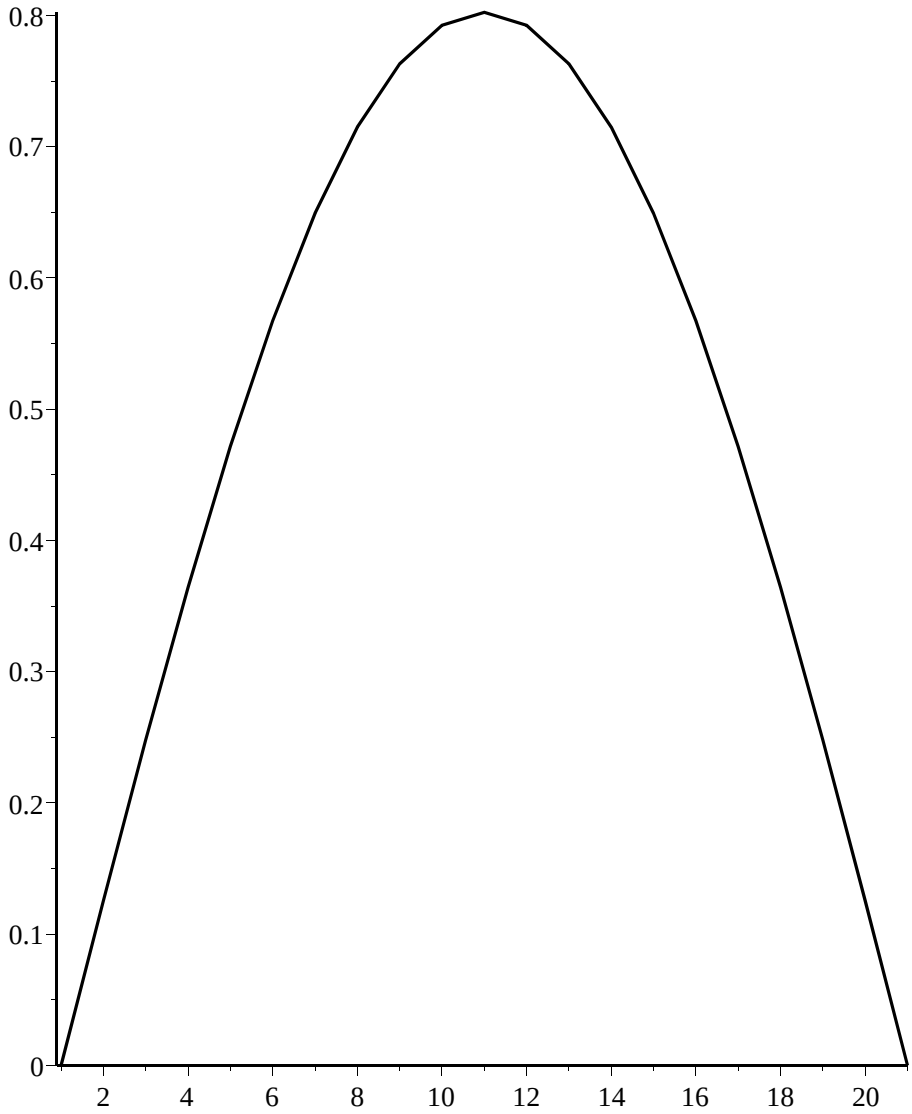


To compute the solution for all time steps, we use the following loop:

```
> for m from 0 to steps-1 do
     tmp := LinearSolve(M, <seq(v[j,m],j=1..n-1)>);
     for j from 1 to n-1 do
        v[j,m+1] := tmp[j]:
     end do:
  end do:
```

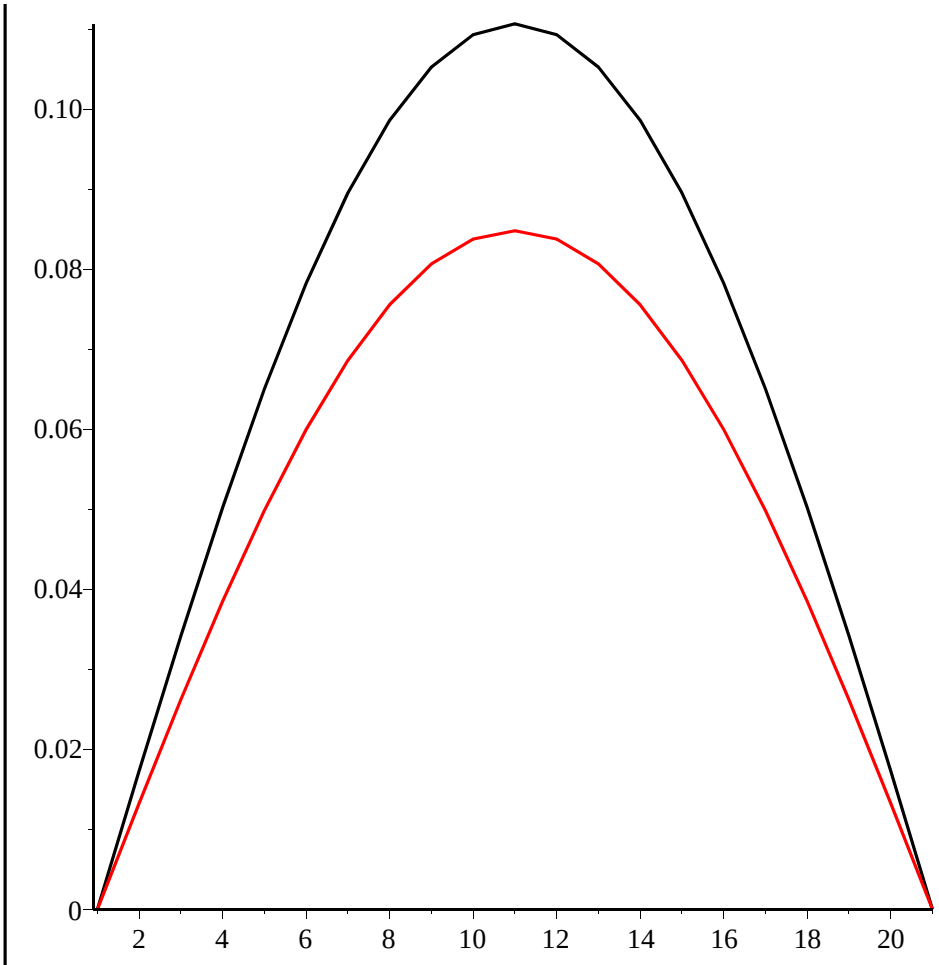
And plot the results again:

```
> resplot := [seq( listplot( [seq( v[j,m], j=0..n)] ), m=1..
  steps)]:
> display(resplot,insequence=true);
```



Of course, we would like to compare the numerical solution with the anayltic one. The following plot shows, that we can come pretty close ...

```
> exact_plot := listplot( [seq( u_part(j*h, steps*tau), j=0..
  n)], color=red ):
> display(resplot[steps],exact_plot);
```



A Second Order Implicit Scheme

```
> n := 20; h := 1/n;
  tau := 1/40; steps := 10;
  v := array(0..n,0..steps);
```

$$n := 20$$

$$h := \frac{1}{20}$$

$$\tau := \frac{1}{40}$$

$$steps := 10$$

$$v := array(0..20, 0..10, [\ ]) \qquad \textbf{(3.1)}$$

Then, we initialize our numerical solution:

```
> for j from 1 to n-1 do
```

```
    v[j,0] := u_init(j*h)
  end do:
```

and set the boundary conditions:

```
> for m from 0 to steps do
    v[0,m] := 0;
    v[n,m] := 0;
  end do:
```

The iteration matrix is similar to that of the simple implicit scheme

```
> tridiag := (i,j) -> `if`(i=j,2,
                           `if`(i=j+1 or i+1=j, -1,0));
  A := Matrix(n-1,tridiag);
```

$$tridiag := (i, j) \rightarrow if\,(i = j, 2, if\,(i = j + 1 \ \textbf{or}\ i + 1 = j, -1, 0))$$

$$A := \begin{bmatrix} 19\ x\ 19\ Matrix \\ Data\ Type:\ anything \\ Storage:\ rectangular \\ Order:\ Fortran\_order \end{bmatrix} \tag{3.2}$$

However, we require an additional weighting parameter α:

```
> alpha := 1/2;
  M := Matrix(n-1,shape=identity) + alpha*tau/h^2*A;
```

$$\alpha := \frac{1}{2}$$

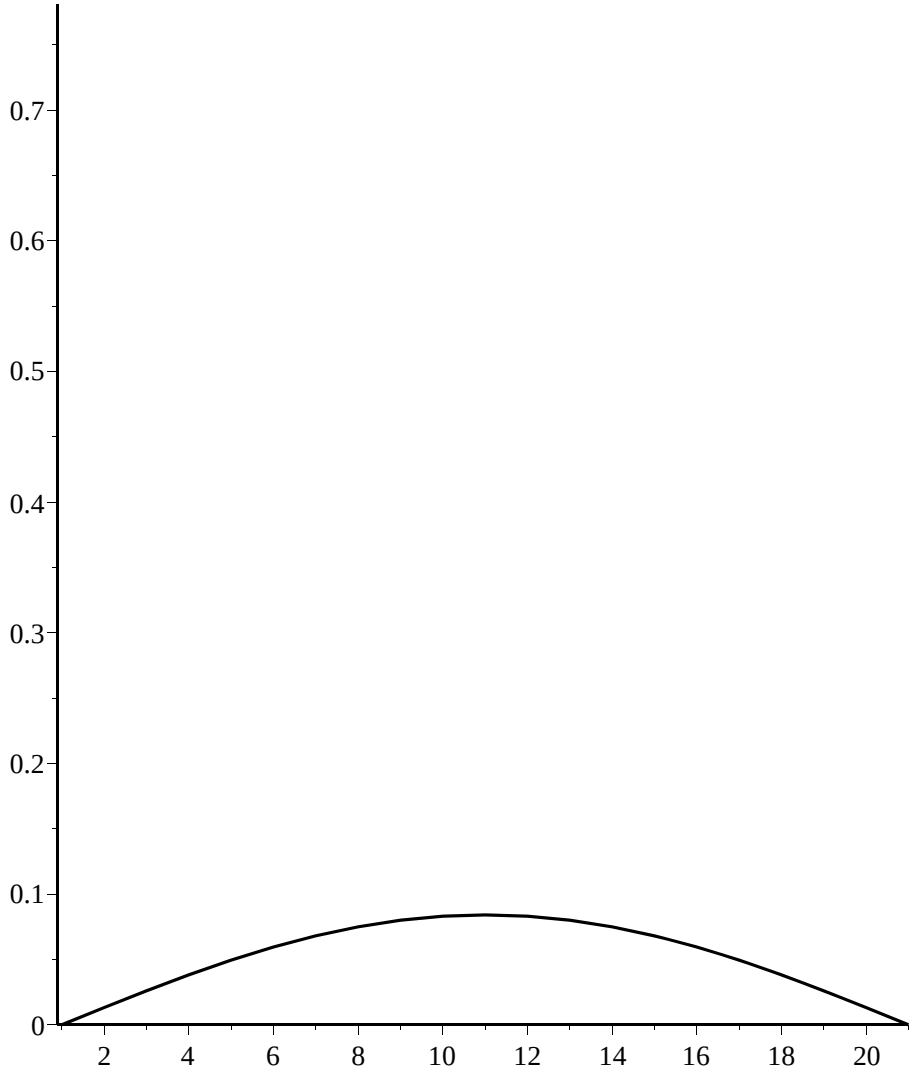
$$M := \begin{bmatrix} 19\ x\ 19\ Matrix \\ Data\ Type:\ anything \\ Storage:\ rectangular \\ Order:\ Fortran\_order \end{bmatrix} \tag{3.3}$$

To compute the solution for all time steps, we use the following loop:

```
> for m from 0 to steps-1 do
    tmp := LinearSolve(M,
             <seq( v[j,m] + (1-alpha)*tau/h^2*( v[j-1,m] - 2*v
  [j,m] + v[j+1,m]) ,j=1..n-1)>);
    for j from 1 to n-1 do
       v[j,m+1] := tmp[j]:
    end do:
  end do:
```

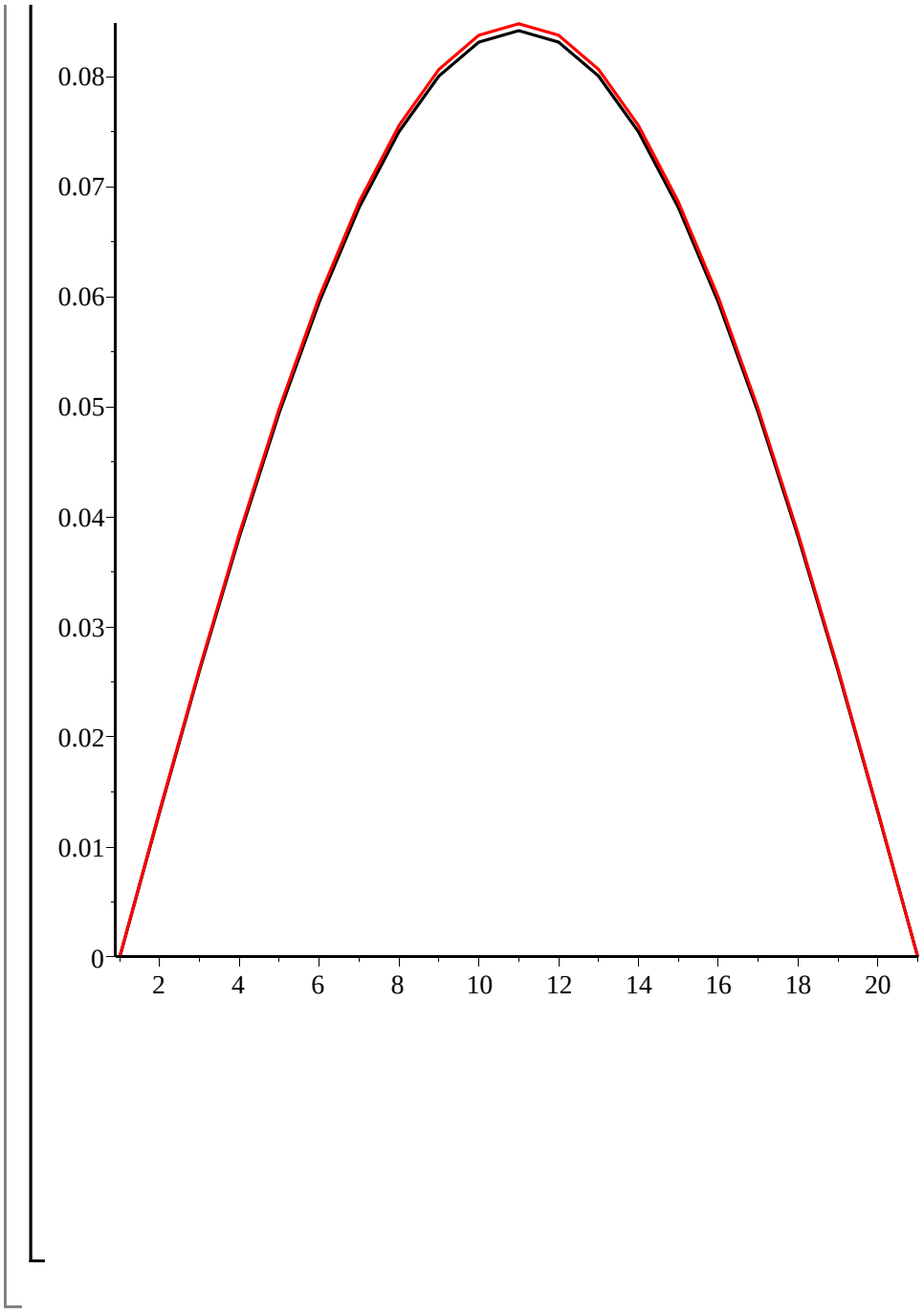
And plot the results again:

```
> resplot := [seq( listplot( [seq( v[j,m], j=0..n)] ), m=1..
  steps)]:
> display(resplot,insequence=true);
```



Of course, we would like to compare the numerical solution with the anayltic one. The following plot shows, that we can come pretty close ...

```
> exact_plot := listplot( [seq( u_part(j*h, steps*tau), j=0..
  n)], color=red ):
> display(resplot[steps],exact_plot);
```


0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
2
4
6
8
10
12
14
16
18
20