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

Finite difference solution of the diffusion equation

The purpose of this worksheet is to discuss the various finite difference stencils one can use to solve the diffusion equation

$$\frac{\partial}{\partial t} u(t,x) = d \frac{\partial^2}{\partial x^2} u(t,x).$$

Here, d is the diffusion constant. This is a parabolic, linear and homogeneous PDE in two dimensions. In this equation, t plays the role of a time variable and x is a spatial variable; that is, we are going to solve the PDE as an initial value problem. In order to obtain a numeric solution, we will need to supplement the PDE with boundary and initial conditions. Here are our choices:

$$u(t=0,x) = f(x), \qquad u(t,x=-1) = u(t,x=1) = 0, \qquad t \in [0,\tau].$$

Here, $f(x)$ is an arbitrary function and τ is a positive constant. That is, we seek a numeric solution for $u(t,x)$ over the region $\Omega = \{(t,x) \in [0,\tau] \times [-1,1]\}$ subject to $u(t,x)$ vanishing at the endpoints of the spatial interval. These are known as Dirichlet boundary conditions. [An alternative would be to have the derivative of the function with respect to x vanishing at the boundary

$$\left.\frac{\partial}{\partial x} u(t,x)\right|_{x=-1} = \left.\frac{\partial}{\partial x} u(t,x)\right|_{x=+1} = 0;$$

these are Neumann boundary conditions.]

Higher order derivative stencils

The key step in solving our PDE numerically using finite difference methods is to replace the derivatives with so-called "finite difference stencils". Here is an example of the a centered finite difference stencil for the second derivative $\frac{\partial^2}{\partial x^2} u(t,x)$ with $(2N+1)$ terms and a step-size of h:

```
> N := 1;
```

```
stencil[1] := diff(u(t,x),x,x) = add(beta[i]*u(t,x+i*h),i=-N..N)+Error;
```

$$N := 1$$

$$stencil_1 := \frac{\partial^2}{\partial x^2} u(t,x) = \beta_{-1}\, u(t,x-h) + \beta_0\, u(t,x) + \beta_1\, u(t,x+h) + Error \qquad \textbf{(1.1}$$

The stencil is "centered" because u is evaluated at an equal number of points to the right and left of the point where we want to approximate the derivative. The coefficients β_i are specified such that **Error** is $O\left(h^{2N+2}\right)$. That is, we solve for **Error**, expand in a Taylor series in h, and then choose coefficients such that the first $(2N+1)$ terms in the Taylor series vanish. Here is the code:

```
> Err[1] := solve(stencil[1],Error);
  Err[2] := series(Err[1],h,2*N+1);
  Err[3] := convert(convert(Err[2],polynom),D);
  vars := indets(Err[3]) minus {h,t,x,seq(beta[i],i=-N..N)};
  Err[4] := collect(Err[3],vars,'distributed');
  sys := {coeffs(Err[4],vars)};
  ans := solve(sys,{seq(beta[i],i=-N..N)});
  stencil[2] := subs(ans,stencil[1]);
```

$$Err_1 := -\beta_{-1}\, u(t,x-h) - \beta_0\, u(t,x) - \beta_1\, u(t,x+h) + \frac{\partial^2}{\partial x^2} u(t,x)$$

$$Err_2 := -\beta_{-1}\, u(t,x) - \beta_1\, u(t,x) - \beta_0\, u(t,x) + \frac{\partial^2}{\partial x^2} u(t,x) + \left(-\beta_1\, D_2(u)(t,x) + \beta_{-1}\, D_2(u)(t,x)\right) h + \left(-\frac{1}{2}\beta_1\, D_{2,2}(u)(t,x) - \frac{1}{2}\beta_{-1}\, D_{2,2}(u)(t,x)\right) h^2 + O\left(h^3\right)$$

$$Err_3 := -\beta_{-1}\, u(t,x) - \beta_1\, u(t,x) - \beta_0\, u(t,x) + D_{2,2}(u)(t,x) + \left(-\beta_1\, D_2(u)(t,x) + \beta_{-1}\, D_2(u)(t,x)\right) h + \left(-\frac{1}{2}\beta_1\, D_{2,2}(u)(t,x) - \frac{1}{2}\beta_{-1}\, D_{2,2}(u)(t,x)\right) h^2$$

$$vars := \left\{u(t,x), D_2(u)(t,x), D_{2,2}(u)(t,x)\right\}$$

$$Err_4 := \left(-\beta_{-1} - \beta_1 - \beta_0\right) u(t,x) + \left(-\beta_1 + \beta_{-1}\right) h\, D_2(u)(t,x) + \left(1 + \left(-\frac{1}{2}\beta_1 - \frac{1}{2}\beta_{-1}\right) h^2\right) D_{2,2}(u)(t,x)$$

$$sys := \left\{\left(-\beta_1 + \beta_{-1}\right) h,\ 1 + \left(-\frac{1}{2}\beta_1 - \frac{1}{2}\beta_{-1}\right) h^2,\ -\beta_{-1} - \beta_1 - \beta_0\right\}$$

$$ans := \left\{\beta_{-1} = \frac{1}{h^2}, \beta_0 = -\frac{2}{h^2}, \beta_1 = \frac{1}{h^2}\right\}$$

$$stencil_2 := \frac{\partial^2}{\partial x^2} u(t,x) = \frac{u(t,x-h)}{h^2} - \frac{2\,u(t,x)}{h^2} + \frac{u(t,x+h)}{h^2} + Error \qquad \textbf{(1.2}$$

After solving for the β_i coefficients, we can also get an explicit equation for the error:

```
> Err[5] := Error = series(`leadterm`(convert(solve(stencil[2],Error),D)),h,20);
  stencil[3] := subs(convert(Err[5],diff),stencil[2]);
```

$$Err_5 := Error = -\frac{1}{12} D_{2,2,2,2}(u)(t,x)\, h^2$$

$$stencil_3 := \frac{\partial^2}{\partial x^2} u(t,x) = \frac{u(t,x-h)}{h^2} - \frac{2\,u(t,x)}{h^2} + \frac{u(t,x+h)}{h^2} + \left(-\frac{1}{12}\frac{\partial^4}{\partial x^4} u(t,x)\, h^2\right) \qquad \textbf{(1.3}$$

Note that the last term on the RHS is just the leading order term in the error; terms with higher powers of h have been omitted. The power of h in the leading error term is called the order of the stencil. So if you execute the above code with $N = 1$, will have obtained a second order accurate centered stencil for $\frac{\partial^2 u}{\partial x^2}$. In addition to centered stencils, we will also make use of one-sided stencil where all of the points are to left or the right of the point where we want to approximate the derivative. For example, here is a stencil using N points to the right (or future) of (t,x) to approximate $\frac{\partial}{\partial t} u(t,x)$ with stepsize s (we will generally be denoting stepsize in the time direction as s and in the spatial direction as h, with s not necessarily equal to h):

```
> N := 2;
  stencil[1] := diff(u(t,x),t) = add(beta[i]*u(t+s*i,x),i=0..N)+Error;
```

$$N := 2$$

$$stencil_1 := \frac{\partial}{\partial t} u(t,x) = \beta_0\, u(t,x) + \beta_1\, u(t+s,x) + \beta_2\, u(t+2s,x) + Error \qquad \textbf{(1.4}$$

A very similar algorithm to before will yield the coefficients:

```
> Err[1] := solve(stencil[1],Error);
  Err[2] := series(Err[1],s,N+1);
  Err[3] := convert(convert(Err[2],polynom),D);
  vars := indets(Err[3]) minus {s,t,x,seq(beta[i],i=0..N)};
  Err[4] := collect(Err[3],vars,'distributed');
  sys := {coeffs(Err[4],vars)};
  ans := solve(sys,{seq(beta[i],i=0..N)});
```

```
stencil[2] := subs(ans,stencil[1]);
Err[5] := Error = series(`leadterm`(convert(solve(stencil[2],Error),D)),s,20);
stencil[3] := subs(convert(Err[5],diff),stencil[2]);
```

$$Err_1 := -\beta_0\, u(t,x) - \beta_1\, u(t+s,x) - \beta_2\, u(t+2\,s,x) + \frac{\partial}{\partial t}\, u(t,x)$$

$$Err_2 := -\beta_0\, u(t,x) - \beta_1\, u(t,x) - \beta_2\, u(t,x) + \frac{\partial}{\partial t}\, u(t,x) + \left(-\beta_1\, D_1(u)(t,x) - 2\,\beta_2\, D_1(u)(t,x)\right) s + \left(-\frac{1}{2}\,\beta_1\, D_{1,1}(u)(t,x) - 2\,\beta_2\, D_{1,1}(u)(t,x)\right) s^2 + O\left(s^3\right)$$

$$Err_3 := -\beta_0\, u(t,x) - \beta_1\, u(t,x) - \beta_2\, u(t,x) + D_1(u)(t,x) + \left(-\beta_1\, D_1(u)(t,x) - 2\,\beta_2\, D_1(u)(t,x)\right) s + \left(-\frac{1}{2}\,\beta_1\, D_{1,1}(u)(t,x) - 2\,\beta_2\, D_{1,1}(u)(t,x)\right) s^2$$

$$vars := \left\{u(t,x), D_1(u)(t,x), D_{1,1}(u)(t,x)\right\}$$

$$Err_4 := \left(-\beta_0 - \beta_1 - \beta_2\right) u(t,x) + \left(-\frac{1}{2}\,\beta_1 - 2\,\beta_2\right) s^2\, D_{1,1}(u)(t,x) + \left(1 + \left(-\beta_1 - 2\,\beta_2\right) s\right) D_1(u)(t,x)$$

$$sys := \left\{\left(-\frac{1}{2}\,\beta_1 - 2\,\beta_2\right) s^2,\ 1 + \left(-\beta_1 - 2\,\beta_2\right) s,\ -\beta_0 - \beta_1 - \beta_2\right\}$$

$$ans := \left\{\beta_0 = -\frac{3}{2\,s},\ \beta_1 = \frac{2}{s},\ \beta_2 = -\frac{1}{2\,s}\right\}$$

$$stencil_2 := \frac{\partial}{\partial t}\, u(t,x) = -\frac{3}{2}\,\frac{u(t,x)}{s} + \frac{2\,u(t+s,x)}{s} - \frac{1}{2}\,\frac{u(t+2\,s,x)}{s} + Error$$

$$Err_5 := Error = \frac{1}{3}\, D_{1,1,1}(u)(t,x)\, s^2$$

$$stencil_3 := \frac{\partial}{\partial t}\, u(t,x) = -\frac{3}{2}\,\frac{u(t,x)}{s} + \frac{2\,u(t+s,x)}{s} - \frac{1}{2}\,\frac{u(t+2\,s,x)}{s} + \left(\frac{1}{3}\,\frac{\partial^3}{\partial t^3}\, u(t,x)\, s^2\right) \qquad \textbf{(1.5}$$

The whole process of obtaining these stencils can be automated; here is a procedure that does the job:

```
> GenerateStencil := proc(F,N,{orientation:=center,stepsize:=h,showorder:=true,showerror:=
  false})
    local vars, f, ii, Degree, stencil, Error, unknowns, Indets, ans, Phi, r, n, phi;
```

```
Phi := convert(F,D);
vars := op(Phi);
n := PDEtools[difforder](Phi);
f := op(1,op(0,Phi));
if (nops([vars])<>1) then:
  r := op(1,op(0,op(0,Phi)));
else:
  r := 1;
fi:
phi := f(vars);
if (orientation=center) then:
  if (type(N,odd)) then:
    ii := [seq(i,i=-(N-1)/2..(N-1)/2)];
  else:
    ii := [seq(i,i=-(N-1)..(N-1),2)];
  fi;
elif (orientation=left) then:
  ii := [seq(i,i=-N+1..0)]:
elif (orientation=right) then:
  ii := [seq(i,i=0..N-1)]:
fi;
stencil := add(a[ii[i]]*subsop(r=op(r,phi)+ii[i]*stepsize,phi),i=1..N);
Error := D[r$n](f)(vars) - stencil;
Error := convert(series(Error,stepsize,N),polynom);
unknowns := {seq(a[ii[i]],i=1..N)};
Indets := indets(Error) minus {vars} minus unknowns minus {stepsize};
Error := collect(Error,Indets,'distributed');
ans := solve({coeffs(Error,Indets)},unknowns);
if (ans=NULL) then:
   print(`Failure: try increasing the number of points in the stencil`);
   return NULL;
fi:
stencil := subs(ans,stencil);
Error := convert(series(`leadterm`(D[r$n](f)(vars) - stencil),stepsize,N+20),polynom);
Degree := degree(Error,stepsize);
if (showorder) then:
   print(cat(`This stencil is of order `,Degree));
fi:
if (showerror) then:
   print(cat(`This leading order term in the error is `,Error));
fi:
```

```
    convert(D[r$n](f)(vars) = stencil,diff);

  end proc:
```

(N.B.: the details of this procedure may be considered to be somewhat "advanced" Maple code and beyond the scope of this course.) The procedure has 2 mandatory arguments: the derivative to be approximated and the number of points N in the stencil. The default behaviour is to return a centered N point approximation. For example, here is the centered 3 point approximation to $\frac{\partial^2}{\partial x^2} u(t,x)$ we have already calculated:

```
> GenerateStencil(diff(u(t,x),x,x),3);
```

This stencil is of order 2

$$\frac{\partial^2}{\partial x^2} u(t,x) = \frac{u(t,x-h)}{h^2} - \frac{2\,u(t,x)}{h^2} + \frac{u(t,x+h)}{h^2} \qquad \textbf{(1.6}$$

The output is the required stencil, and the procedure prints out a message indicating the order of the stencil. The procedure also works for higher derivatives:

```
> GenerateStencil(diff(zeta(x,y,z),z$4),7);
```

This stencil is of order 4

$$\frac{\partial^4}{\partial z^4} \zeta(x,y,z) = -\frac{1}{6}\,\frac{\zeta(x,y,z-3h)}{h^4} + \frac{2\,\zeta(x,y,z-2h)}{h^4} - \frac{13}{2}\,\frac{\zeta(x,y,z-h)}{h^4} + \frac{28}{3}\,\frac{\zeta(x,y,z)}{h^4} - \frac{13}{2}\,\frac{\zeta(x,y,z+h)}{h^4} \qquad \textbf{(1.7}$$
$$+ \frac{2\,\zeta(x,y,z+2h)}{h^4} - \frac{1}{6}\,\frac{\zeta(x,y,z+3h)}{h^4}$$

Generally speaking, we will need at least $(N+1)$ terms in the stencil to approximate a derivative of order N. If we choose N to be too small for the derivative we're approximating, the procedure will return an error:

```
> GenerateStencil(diff(q(t),t$3),3);
```

Failure: try increasing the number of points in the stencil **(1.8**

To fix this, we increase the number of points in the stencil from 3 to 4:

```
> GenerateStencil(diff(q(t),t$3),4);
```

This stencil is of order 2

$$\frac{d^3}{dt^3} q(t) = -\frac{1}{8}\,\frac{q(t-3h)}{h^3} + \frac{3}{8}\,\frac{q(t-h)}{h^3} - \frac{3}{8}\,\frac{q(t+h)}{h^3} + \frac{1}{8}\,\frac{q(t+3h)}{h^3} \qquad \textbf{(1.9}$$

There are some optional arguments to the procedure as well. Setting `orientation = center`, `right`, or `left` will return a centered, left-biased or right-biased stencil, respectively:

```
> GenerateStencil(diff(u(t,x),t),2,orientation=center);
  GenerateStencil(diff(u(t,x),t),2,orientation=left);
  GenerateStencil(diff(u(t,x),t),2,orientation=right);
```

This stencil is of order 2

$$\frac{\partial}{\partial t}\,u(t,x) = -\frac{1}{2}\,\frac{u(t-h,x)}{h} + \frac{1}{2}\,\frac{u(t+h,x)}{h}$$

This stencil is of order 1

$$\frac{\partial}{\partial t}\,u(t,x) = -\frac{u(t-h,x)}{h} + \frac{u(t,x)}{h}$$

This stencil is of order 1

$$\frac{\partial}{\partial t}\,u(t,x) = -\frac{u(t,x)}{h} + \frac{u(t+h,x)}{h} \qquad \textbf{(1.10}$$

Changing the **stepsize** variable will use a different notation for the stepsize:

```
> GenerateStencil(diff(y(x),x$4),5,stepsize=m);
```

This stencil is of order 2

$$\frac{d^4}{dx^4}\,y(x) = \frac{y(x-2\,m)}{m^4} - \frac{4\,y(x-m)}{m^4} + \frac{6\,y(x)}{m^4} - \frac{4\,y(x+m)}{m^4} + \frac{y(x+2\,m)}{m^4} \qquad \textbf{(1.11}$$

Setting **showorder = false** will supress the printout of the error:

```
> GenerateStencil(diff(y(x),x$4),5,showorder=false);
```

$$\frac{d^4}{dx^4}\,y(x) = \frac{y(x-2\,h)}{h^4} - \frac{4\,y(x-h)}{h^4} + \frac{6\,y(x)}{h^4} - \frac{4\,y(x+h)}{h^4} + \frac{y(x+2\,h)}{h^4} \qquad \textbf{(1.12}$$

Conversely, setting **showerror = true** will print out the leading order term in the error:

```
> GenerateStencil(diff(y(x),x$4),5,showerror=true);
```

This stencil is of order 2

This leading order term in the error is $\| \left(-\frac{1}{6}\,D^{(6)}(y)(x)\,h^2 \right)$

$$\frac{d^4}{dx^4}\,y(x) = \frac{y(x-2\,h)}{h^4} - \frac{4\,y(x-h)}{h^4} + \frac{6\,y(x)}{h^4} - \frac{4\,y(x+h)}{h^4} + \frac{y(x+2\,h)}{h^4} \qquad \textbf{(1.13}$$

Forward-time centered-space (FTCS) stencil

We now wish to use the stencils derived in the previous section to write down an approximate form of the diffusion equation:

```
> pde := diff(u(t,x),t) - d*diff(u(t,x),x,x);
```

$$pde := \frac{\partial}{\partial t} u(t,x) - d \left(\frac{\partial^2}{\partial x^2} u(t,x) \right) \tag{2.1}$$

Let us take (t, x) to be the point at which we are approximating the PDE. The time derivative will be replaced with a "forward time" right-oriented stencil with 2 points:

```
> forward_time := GenerateStencil(diff(u(t,x),t),2,orientation=right,stepsize=s);
```

This stencil is of order 1

$$forward\_time := \frac{\partial}{\partial t} u(t,x) = -\frac{u(t,x)}{s} + \frac{u(t+s,x)}{s} \tag{2.2}$$

The "forward time" label comes from the fact that we are approximating the time derivative at (t, x) using a point to the future. Notice that we have denoted the stepsize in the time direction by s. The spatial derivative will be approximated with a centered 3 point stencil with stepsize h:

```
> centered_space := GenerateStencil(diff(u(t,x),x,x),3,orientation=center,stepsize=h);
```

This stencil is of order 2

$$centered\_space := \frac{\partial^2}{\partial x^2} u(t,x) = \frac{u(t,x-h)}{h^2} - \frac{2\,u(t,x)}{h^2} + \frac{u(t,x+h)}{h^2} \tag{2.3}$$

Putting the "forward time" and "centered space" stencils in the PDE results in the "forward-time centered-space (FTCS)" stencil of the diffusion equation:

```
> FTCS_stencil := subs(forward_time,centered_space,pde);
```

$$FTCS\_stencil := -\frac{u(t,x)}{s} + \frac{u(t+s,x)}{s} - d \left(\frac{u(t,x-h)}{h^2} - \frac{2\,u(t,x)}{h^2} + \frac{u(t,x+h)}{h^2} \right) \tag{2.4}$$

Recalling the errors for the spatial and temporal sub-stencils, we say that the aggregate stencil is first order in time and second order in space. Note that the stencil can be re-arranged to yield $u(t+s, x)$ as an explicit function of the other three values of u:

```
> FTCS_stencil := expand(isolate(FTCS_stencil,u(t+s,x)));
```

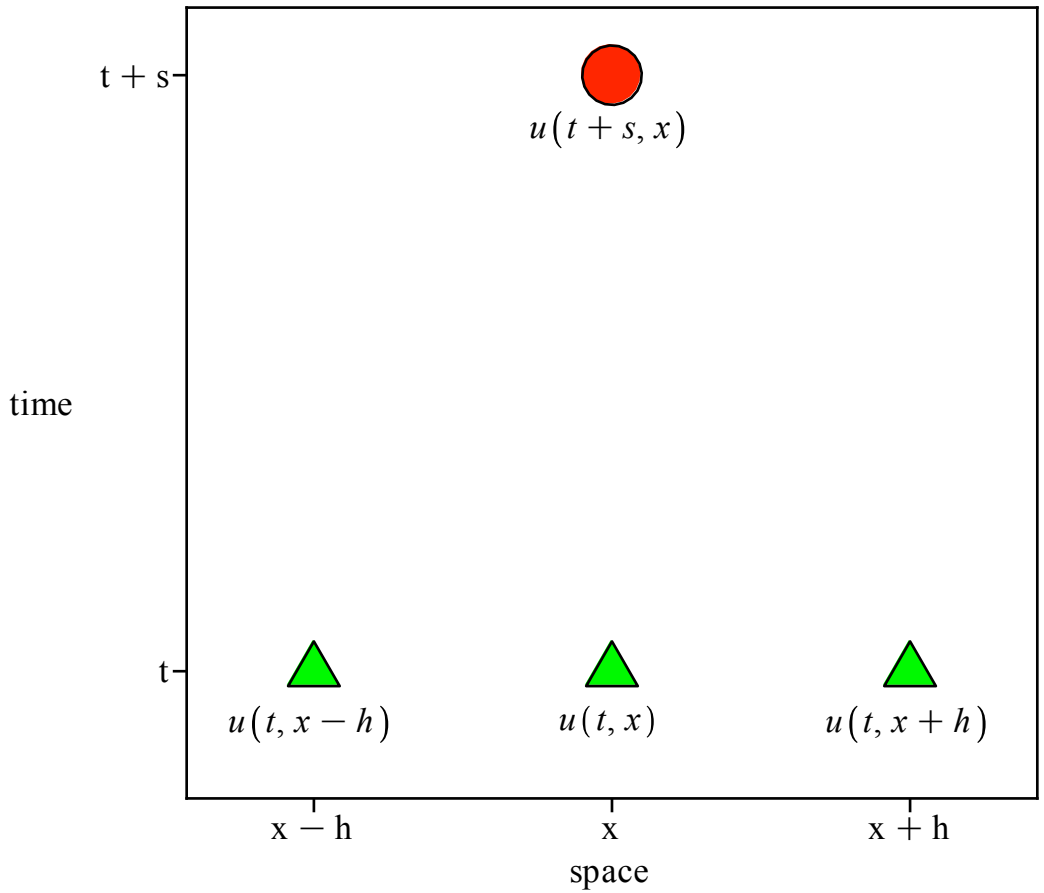
$$FTCS\_stencil := u(t+s,x) = u(t,x) + \frac{s\,d\,u(t,x-h)}{h^2} - \frac{2\,s\,d\,u(t,x)}{h^2} + \frac{s\,d\,u(t,x+h)}{h^2} \tag{2.5}$$

We can interpret this by using the following sketch:

```
> ngon := (n,x,y,r,phi) -> [seq([x+r*cos(2*Pi*i/n+phi), y+r*sin(2*Pi*i/n+phi)], i = 1 .. n)]
```

```
:
display([seq(polygonplot(ngon(3,i,0,0.1,Pi/2),color=green),i=-1..1),polygonplot(ngon(20,0,
2,0.1,Pi/8),color=red),textplot([seq([i,-0.1,typeset(u(t,x+i*h))],i=-1..1),[0,1.9,typeset
(u(t+s,x))]],align={below})],view=[-1.4..1.4,-0.4..2.2],tickmarks=[[-1=x-h,0=x,1=x+h],[0=
t,2=t+s]],axes=boxed,scaling=constrained,labels=["space","time"],title="Information flow
in the FTCS stencil");
```

Information flow in the FTCS stencil



Equation **(2.5)** gives us an approximation to u at the red circle given knowledge of u at the three past positions represented by the green triangles. This equation form the basis of the numerical method used in this section. To implement the method, recall that we are seeking a

solution of **(2.1)** over the region $\Omega = \{(t, x) \in [0, \tau] \times [-1, 1]\}$. We introduce a lattice over Ω consisting of a number of points (t_i, x_j). There will be $(N+1)$ points in the time direction and $(M+2)$ points in the spatial direction such that:

$$t_i = \frac{i\tau}{N}, \quad x_j = \frac{2j}{M+1} - 1, \quad i = 0 \ldots N, \quad j = 0 \ldots (M+1).$$

This choice of lattice automatically implies that

$$t_0 = 0, \quad t_N = \tau, \quad x_0 = -1, \quad x_{M+1} = +1.$$

We will use the following mappings **X** and **T** to generate the various points on the lattice via $t_i = T(i)$ and $x_j = X(j)$.

```
> N := 'N':
  M := 'M':
  X := j -> -1 + j*2/(M+1);
  T := i -> tau*i/N;
```

$$X := j \rightarrow -1 + \frac{2j}{M+1}$$

$$T := i \rightarrow \frac{\tau i}{N} \qquad \textbf{(2.6}$$

Notice that these relations also define the timestep and spacestep:

```
> s_def := s = T(1)-T(0);
  h_def := h = X(1)-X(0);
```

$$s\_def := s = \frac{\tau}{N}$$

$$h\_def := h = \frac{2}{M+1} \qquad \textbf{(2.7}$$

Here is a visualization of our lattice for specific choices of N, M and τ:

```
> M := 3;
  N := 4;
  tau := 1.6;
  r := 1/M/6;
  display([seq(polygonplot(ngon(3,X(0),T(i),r,Pi/2),color=magenta),i=0..N),seq(polygonplot
  (ngon(3,X(M+1),T(i),r,Pi/2),color=magenta),i=0..N),seq(polygonplot(ngon(4,X(j),T(0),r,0),
```

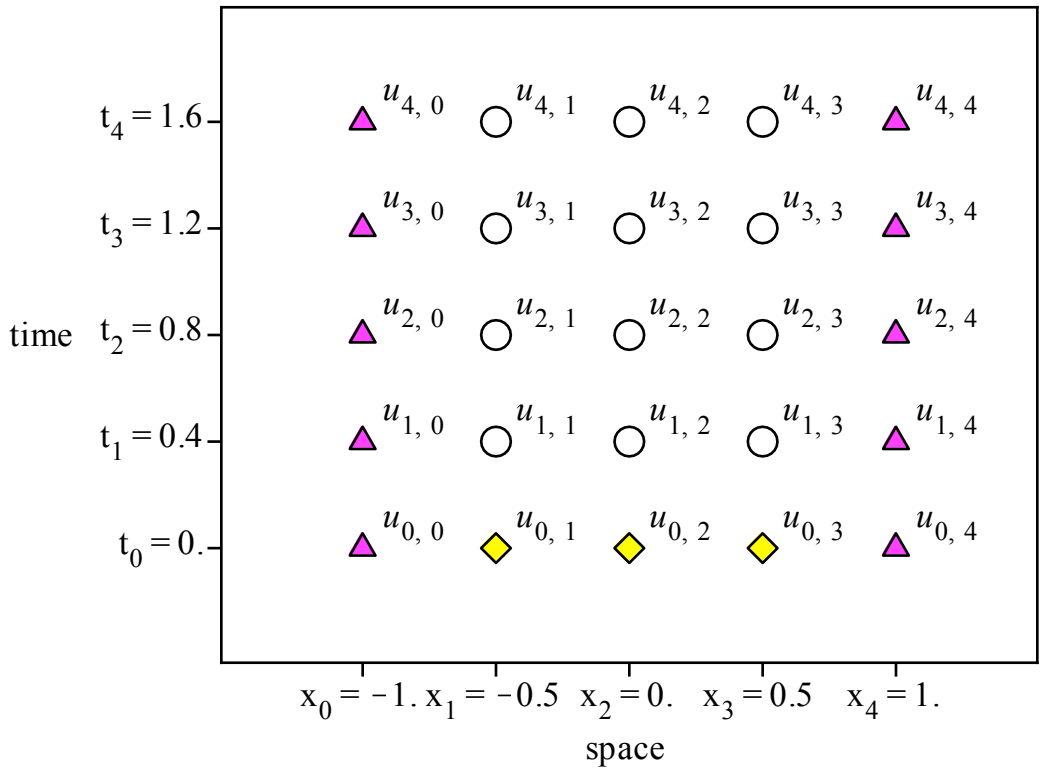
```
color=yellow),j=1..M),seq(seq(polygonplot(ngon(20,X(j),T(i),r,0),color=white),i=1..N),j=1.
.M),textplot([seq(seq([X(j+0.1),T(i),typeset(u[i,j])],i=0..N),j=0..M+1)],align={above,
right})],view=[X(-1)..X(M+2),T(-1)..T(N+1)],tickmarks=[[seq(X(j)=typeset(x[j]=evalf[2](X
(j))),j=0..M+1)],[seq(T(i)=typeset(t[i]=evalf[2](T(i))),i=0..N)]],axes=boxed,scaling=
constrained,labels=["space","time"]);
```

$$M := 3$$

$$N := 4$$

$$\tau := 1.6$$

$$r := \frac{1}{18}$$



Our numerical solution of **(2.1)** will consist of an approximation to the field at each of the lattice points, which we will denote by $u_{i,j} \approx u(t_i, x_j)$. But we do not need to calculate $u_{i,j}$ for every lattice point: The boundary conditions fix the value of $u_{i,j}$ at each node corresponding to $x = -1$ or $x = +1$, which are denoted by the purple triangles in the above plot:

$$u(t, x = -1) = u(t, x = +1) = 0 \quad \Rightarrow \quad u_{i,0} = u_{i,M+1} = 0.$$

Also, the initial conditions fix the value of $u_{i,j}$ for nodes at $t = 0$, which are denoted by yellow diamonds in the above plot:

$$u(t=0,x) = f(x) \quad \Rightarrow \quad u_{0,j} = f\left(x_j\right).$$

Hence, we only need to find approximations to the field at each of the $N \times M$ lattice nodes represented by white circles in the above plot; i.e. all the nodes not covered by the boundary conditions and the initial data.

Aside: There is a potential inconsistency at the bottom corners of our grid if the initial data does not satisfy the boundary condition; i.e., $f\left(x_0 = -1\right) \neq 0$ or $f\left(x_{M+1} = 1\right) \neq 0$. In practice this is not a big deal: if we assume the boundary conditions supercede the initial conditions, it actually implies that our numerical method is solving the PDE with slightly modified initial data. In the $h \to 0$ limit this will b inconsequential.

Now, we can use our stencil **(2.5)** to give an algorithm to find $u_{i+1,j}$ in terms of $\left(u_{i,j-1}, u_{i,j}, u_{i,j+1}\right)$:

$$u_{i+1,j} = \Phi\left(u_{i,j-1}, u_{i,j}, u_{i,j+1}\right).$$

The explicit form of Φ is:

```
> Subs := [u(t,x)=u_middle,u(t,x-h)=u_right,u(t,x+h)=u_left,u(t+s,x)=u_future];
  Phi := subs(Subs,FTCS_stencil);
  Phi := unapply(rhs(Phi),u_left,u_middle,u_right,h,s,d);
```

$$Subs := [u(t,x) = u\_middle, u(t,x-h) = u\_right, u(t,x+h) = u\_left, u(t+s,x) = u\_future]$$

$$\Phi := u\_future = u\_middle + \frac{s\,d\,u\_right}{h^2} - \frac{2\,s\,d\,u\_middle}{h^2} + \frac{s\,d\,u\_left}{h^2}$$

$$\Phi := (u\_left, u\_middle, u\_right, h, s, d) \to u\_middle + \frac{s\,d\,u\_right}{h^2} - \frac{2\,s\,d\,u\_middle}{h^2} + \frac{s\,d\,u\_left}{h^2} \qquad \textbf{(2.8}$$

Given this Φ, we can then obtain all the field values on the first row $u_{1,j}$ $(j = 1 \ldots M)$ since we have knowledge of the field values on the zeroth row $u_{0,j}$ $(j = 0 \ldots M+1)$ from the initial data and boundary conditions. After calculating $u_{1,j}$ we can then obtain $u_{2,j}$, and so on until we have filled in all the white circles. Here is a procedure that implements this routine:

```
> FTCS := proc(tau,N,M,d,f) local s, h, XX, X, T, PlotOptions, Title, u_past, u_future, p,
  i, j:

      # We first determine the time and space steps
```

```
X := j -> -1 + j*2/(M+1);
T := i -> tau*i/N;
s := evalf(T(1)-T(0));
h := evalf(X(1)-X(0));
```

We define an Array containing the x-coordinates of the spatial lattice

```
XX := Array(0..M+1,[seq(X(j),j=0..M+1)],datatype=float):
```

The scheme will be based on two Arrays `u_past` and `u_future`:
`u_past` corresponds to the field values on a given time step
`u_future` corresponds to the field values at the next time step
To begin, we fix `u_past` from the initial data.

```
u_past := Array(0..M+1,[seq(f(XX[j]),j=0..M+1)],datatype=float):
```

Our goal will be a movie whose frames are plots of u at each time step
The frames of our movie will be generated by a separate procedure that follows this one

```
p[0] := Frame[1](XX,u_past,s,h,T(0));
```

Now we start the main calculation loop
`i` will run over timesteps while `j` runs over spacesteps

```
for i from 1 to N do:
```

    # We initialize the `u_future` array and fix its values
    # at either end based on the boundary conditions

```
    u_future := Array(0..M+1,datatype=float):
    u_future[0] := 0:
    u_future[M+1] := 0:
```

    # The rest of the values of `u_future` are obtained by using the `Phi` procedure
    # Notice the arguments of `Phi` are the values of `u_past`

```
    for j from 1 to M do:
        u_future[j] := Phi(u_past[j-1],u_past[j],u_past[j+1],h,s,d):
    od:
```

```
            # Now, we get ready for the next time step by making u_future into the new u_past
            ArrayTools[Copy](u_future,u_past):
            # Finally, another frame of our movie is generated:
            p[i] := Frame[1](XX,u_past,s,h,T(i));
        od:
        # After the loop is over, we have N plots p[i] that are the pictures of u at each time slice
        # The display command assembles these into a movie, which is the output of the procedure
        display(convert(p,list),insequence=true):
end proc:

# This procedure generates each frame of the movie output by the previous routine
# It inputs x and y data and outouts a plot

Frame[1] := proc(xdata,ydata,s,h,T) local Title, PlotOptions:
        Title := typeset(`timestep = `,evalf[2](s),`, spacestep = `,evalf[2](h),`, `,t=
evalf[2](T));
        PlotOptions := axes=boxed, labels=[x,u(t,x)], legend=["FTCS"], color=red;
        plot(Matrix([[xdata],[ydata]])^%T,title=Title,PlotOptions);

end proc:
```

Notice how this procedure does not actually store all of the unknown values of $u_{i,j}$ in the lattice; it only holds two rows in memory at a time. This is purely to preserve RAM, large simulations can easily deplete the physical memory of typical computers. Here is an example of the movie output from our code:

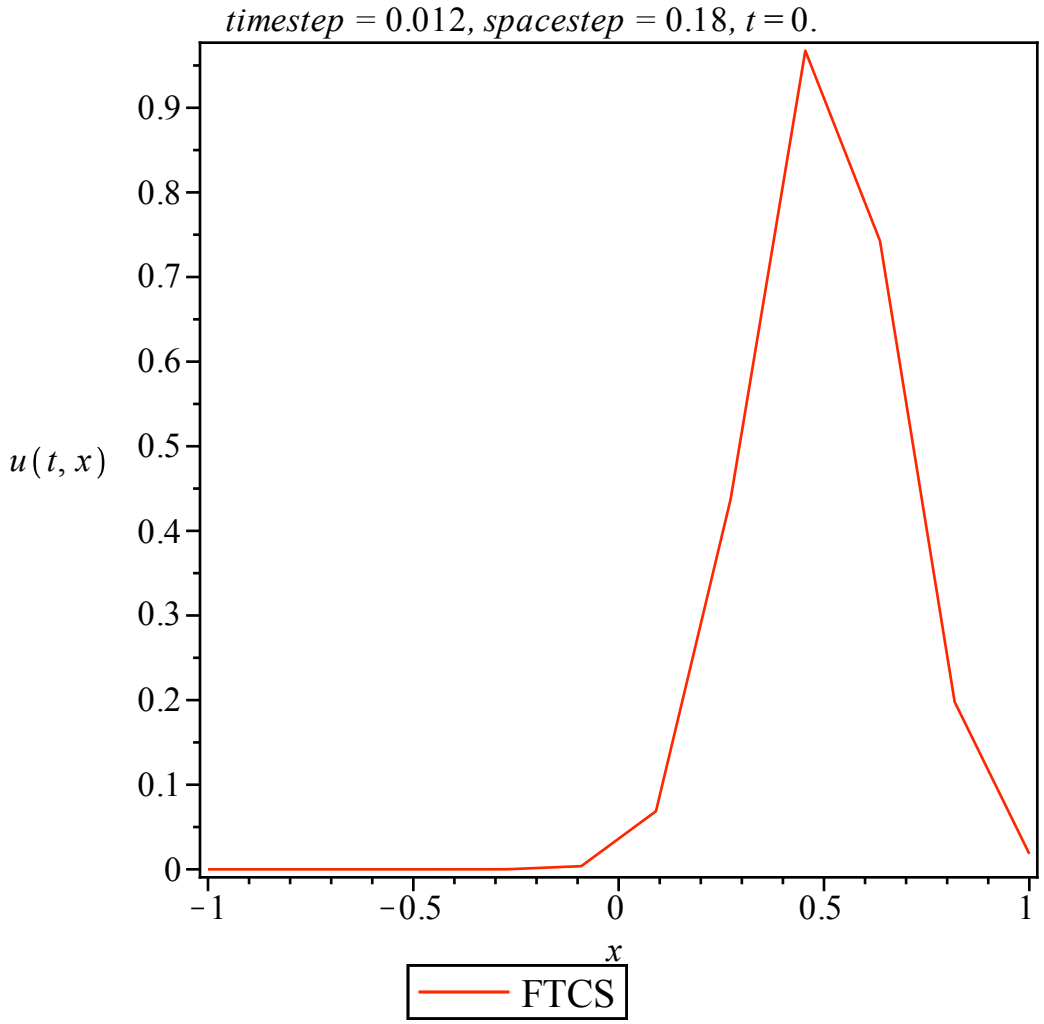
```
> f := x-> exp(-(4*(x-1/2))^2):
  N := 80;
  M := 10;
  d := 1;
  tau := 1;
  movie[1] := FTCS(tau,N,M,d,f):
  movie[1];
```

$N := 80$

$$M := 10$$

$$d := 1$$

$$\tau := 1$$



Here is a comparison of the output of our **FTCS** procedure and **pdsolve/numeric** solution of the problem using the same time- and spacesteps:

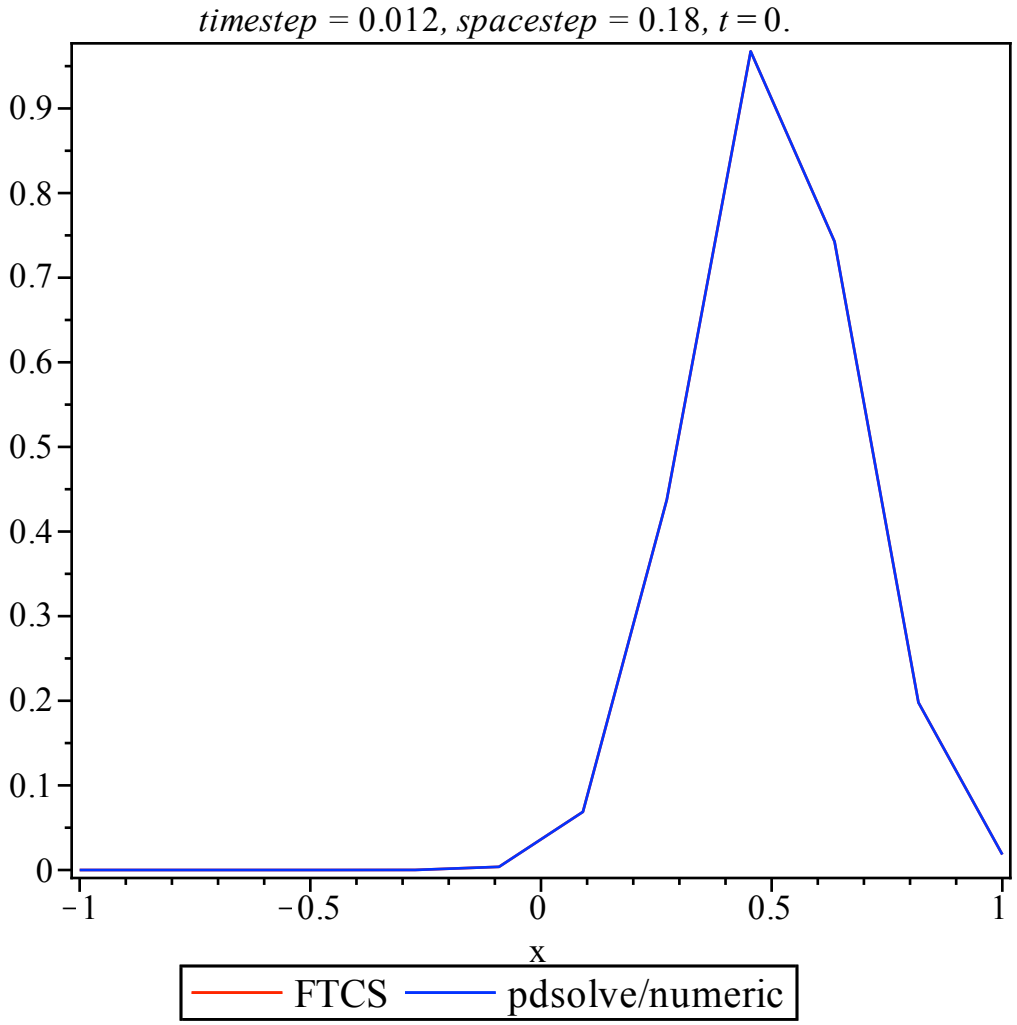
```
> S := evalf(subs(s_def,s));
```

```
H := evalf(subs(h_def,h));
IBC := [u(t,-1)=0,u(t,+1)=0,u(0,x)=f(x)];
pds := pdsolve(pde,IBC,numeric,spacestep=H,timestep=S):
movie[2] := pds:-animate(t=0..tau,frames=N+1,axes=boxed,color=blue,legend=
["pdsolve/numeric"]):
display([movie[1],movie[2]]);
```

$$S := 0.01250000000$$

$$H := 0.1818181818$$

$$IBC := \left[u(t, -1) = 0, u(t, 1) = 0, u(0, x) = e^{-(4x-2)^2}\right]$$



Matrix formulation (method of lines)

```
> j := 'j':
  d := 'd':
```

Before going on to discuss other types of stencils for the diffusion equation, it is useful to reformulate the FTCS stencil in a slightly differer way. Let us recall the PDE we are trying to solve:

```
> pde;
```

$$\frac{\partial}{\partial t} u(t,x) - d\left(\frac{\partial^2}{\partial x^2} u(t,x)\right) \tag{3.1}$$

Now, let us substitute the centered space stencil **(2.3)** that we used above into the PDE, but we'll refrain from putting in the forward time **(2.2)** one:

```
> eq[1] := subs(centered_space,pde);
```

$$eq_1 := \frac{\partial}{\partial t} u(t,x) - d\left(\frac{u(t,x-h)}{h^2} - \frac{2\,u(t,x)}{h^2} + \frac{u(t,x+h)}{h^2}\right) \tag{3.2}$$

Let's evaluate this equation at the j^{th} spatial node:

```
> eq[2] := eval(eq[1],x=x[j]);
```

$$eq_2 := \frac{\partial}{\partial t} u(t,x_j) - d\left(\frac{u(t,x_j-h)}{h^2} - \frac{2\,u(t,x_j)}{h^2} + \frac{u(t,x_j+h)}{h^2}\right) \tag{3.3}$$

We are now going to view $u(t,x_j-h) = u(t,x_{j-1})$, $u(t,x_j)$ and $u(t,x_j+h) = u(t,x_{j+1})$ as separate functions of time. We use the notation $U_j(t) = u(t,x_j)$:

```
> Subs := [seq(u(t,x[j]+jj*h)=U[j+jj](t),jj=-1..1)];
```

$$Subs := \left[u(t,x_j-h) = U_{j-1}(t),\ u(t,x_j) = U_j(t),\ u(t,x_j+h) = U_{j+1}(t)\right] \tag{3.4}$$

Putting these definitions into **(3.3)** yields:

```
> eq[3] := isolate(subs(Subs,eq[2]),diff(U[j](t),t));
```

$$eq_3 := \frac{\mathrm{d}}{\mathrm{d}t} U_j(t) = d\left(\frac{U_{j-1}(t)}{h^2} - \frac{2\,U_j(t)}{h^2} + \frac{U_{j+1}(t)}{h^2}\right) \tag{3.5}$$

Now, the boundary condition will imply that $U_0(t) = u(t,x_0) = 0$ and $U_{M+1}(t) = u(t,x_{M+1}) = 0$. This means that **(3.5)** actually represents a set of M coupled ODEs for M unknown functions $\{U_j(t)\}_{j=1}^{M}$. Here is a particular realization of this system of equations for a given M:

```
> U := 'U':
  sys := 'sys':
  M := 4;
  U[0] := t -> 0;
  U[M+1] := t -> 0;
  for j from 1 to M do:
     sys[j] := rhs(eq[3])=lhs(eq[3]);
```

```
od;
j := 'j':
```

$$M := 4$$

$$U_0 := t \to 0$$

$$U_5 := t \to 0$$

$$sys_1 := d\left(-\frac{2\,U_1(t)}{h^2} + \frac{U_2(t)}{h^2}\right) = \frac{\mathrm{d}}{\mathrm{d}t}U_1(t)$$

$$sys_2 := d\left(\frac{U_1(t)}{h^2} - \frac{2\,U_2(t)}{h^2} + \frac{U_3(t)}{h^2}\right) = \frac{\mathrm{d}}{\mathrm{d}t}U_2(t)$$

$$sys_3 := d\left(\frac{U_2(t)}{h^2} - \frac{2\,U_3(t)}{h^2} + \frac{U_4(t)}{h^2}\right) = \frac{\mathrm{d}}{\mathrm{d}t}U_3(t)$$

$$sys_4 := d\left(\frac{U_3(t)}{h^2} - \frac{2\,U_4(t)}{h^2}\right) = \frac{\mathrm{d}}{\mathrm{d}t}U_4(t) \qquad \textbf{(3.6}$$

This kind of transformation of a single PDE into a number of coupled ODEs is called the "method of lines". The basic idea is that the system can now be solved using any ODE algorithm you wish (this is the default method of handling PDE initial value problems in MATLAB). For the case of the diffusion equation, we see that the system of ODEs is linear in the unknown functions, so we can re-express this as a matrix equation:

$$\mathbf{X}(t) = \begin{bmatrix} U_1(t) \\ U_2(t) \\ \vdots \\ U_M(t) \end{bmatrix}, \quad \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{X}(t) = \mathbf{A}\mathbf{X}(t),$$

where **A** is an $M \times M$ square constant matrix. We can extract the form of the coefficient matrix using **GenerateMatrix**:

```
> X := [seq(U[j](t),j=1..M)];
  A := GenerateMatrix(convert(sys,list),X)[1];
```

$$X := [U_1(t), U_2(t), U_3(t), U_4(t)]$$

$$A := \begin{bmatrix} -\frac{2\,d}{h^2} & \frac{d}{h^2} & 0 & 0 \\ \frac{d}{h^2} & -\frac{2\,d}{h^2} & \frac{d}{h^2} & 0 \\ 0 & \frac{d}{h^2} & -\frac{2\,d}{h^2} & \frac{d}{h^2} \\ 0 & 0 & \frac{d}{h^2} & -\frac{2\,d}{h^2} \end{bmatrix} \qquad \textbf{(3.7}$$

The matrix **A** has a pretty simple structure: It is merely dh^{-2} times a square matrix with -2's along the main diagonal and 1's along each of the adjacent subdiagonals. Here is a routine that automatically generates **A** for any number of spatial lattice points:

```
> _A := (M,d,h) -> d/h^2*BandMatrix([1,-2,1],1,M);
  _A(M,d,h);
```

$$\_A := (M, d, h) \rightarrow \frac{d\ LinearAlgebra{:}\text{-}BandMatrix([1, -2, 1], 1, M)}{h^2}$$

$$\begin{bmatrix} -\frac{2\,d}{h^2} & \frac{d}{h^2} & 0 & 0 \\ \frac{d}{h^2} & -\frac{2\,d}{h^2} & \frac{d}{h^2} & 0 \\ 0 & \frac{d}{h^2} & -\frac{2\,d}{h^2} & \frac{d}{h^2} \\ 0 & 0 & \frac{d}{h^2} & -\frac{2\,d}{h^2} \end{bmatrix} \qquad \textbf{(3.8}$$

A forward Euler type solution of the matrix ODE $\frac{\mathrm{d}}{\mathrm{d}\,t}\mathbf{X}(t) = \mathbf{A} \cdot \mathbf{X}(t)$ would proceed as follows: We define a time lattice by $t_i = is$, where s is the timestep. Then, we denote our numeric solution to the ODE at the i^{th} time by $\mathbf{X}_i \approx \mathbf{X}(t_i)$. Then the forward Euler stencil for the ODE relies on the following approximation for the time derivative:

$$\frac{\mathrm{d}}{\mathrm{d}\,t}\mathbf{X}(t) \approx \frac{\mathbf{X}(t+s)-\mathbf{X}(t)}{s}.$$

Evaluating at $t = t_i$ and putting this into the matrix ODE gives:

$$\mathbf{X}_{i+1} = \mathbf{X}_i + s\mathbf{A}\mathbf{X}_i = (\mathbf{I} + s\mathbf{A})\mathbf{X}_i = \mathbf{P}\mathbf{X}_i, \qquad \mathbf{P} = \mathbf{I} + s\mathbf{A}.$$

It is not terribly difficult to convince oneself that this is exactly the same algorithm as the FTCS stencil of the previous section. To see this explicitly, we consider the forward Euler stencil of **(3.5)**. Let's evaluate this at $t = t_i$:

```
> eq[4] := eval(convert(eq[3],D),t=t[i]);
```

$$eq_4 := \mathrm{D}\left(U_j\right)\left(t_i\right) = d\left(\frac{U_{j-1}\left(t_i\right)}{h^2} - \frac{2\,U_j\left(t_i\right)}{h^2} + \frac{U_{j+1}\left(t_i\right)}{h^2}\right) \qquad \textbf{(3.9}$$

A forward Euler type approximation of the time derivative is:

```
> eq[5] := D(U[j])(t[i]) = (U[j](t[i+1])-U[j](t[i]))/s;
```

$$eq_5 := \mathrm{D}\left(U_j\right)\left(t_i\right) = \frac{U_j\left(t_{i+1}\right) - U_j\left(t_i\right)}{s} \qquad \textbf{(3.10}$$

Subbing this in:

```
> Forward_Euler := subs(eq[5],eq[4]);
  Forward_Euler := expand(isolate(Forward_Euler,U[j](t[i+1])));
```

$$Forward\_Euler := \frac{U_j\left(t_{i+1}\right) - U_j\left(t_i\right)}{s} = d\left(\frac{U_{j-1}\left(t_i\right)}{h^2} - \frac{2\,U_j\left(t_i\right)}{h^2} + \frac{U_{j+1}\left(t_i\right)}{h^2}\right)$$

$$Forward\_Euler := U_j\left(t_{i+1}\right) = \frac{d\,s\,U_{j-1}\left(t_i\right)}{h^2} - \frac{2\,d\,s\,U_j\left(t_i\right)}{h^2} + \frac{d\,s\,U_{j+1}\left(t_i\right)}{h^2} + U_j\left(t_i\right) \qquad \textbf{(3.11}$$

This exactly matches the FTCS stencil **(2.5)** with the identifications:

```
> Subs := [seq(seq(U[j+jj](t[i+ii])=u(t+ii*s,x+jj*h),jj=-1+ii..1-ii),ii=0..1)];
  FTCS_stencil-subs(Subs,Forward_Euler);
```

$$Subs := \left[U_{j-1}\left(t_i\right) = u(t, x-h),\ U_j\left(t_i\right) = u(t,x),\ U_{j+1}\left(t_i\right) = u(t,x+h),\ U_j\left(t_{i+1}\right) = u(t+s,x)\right]$$

$$0 = 0 \qquad \textbf{(3.12}$$

Now, we construct another implementation of the FTCS stencil based on the matrix version of the forward Euler algorithm: $\mathbf{X}_{i+1} = \mathbf{P}\mathbf{X}_i$,

where **P** = **I** + *s***A**:

```
> FTCS_matrix := proc(tau,N,M,d,f)
          local X, T, s, h, XX, u_past, u_future, Title, PlotOptions, p, A, P, i:

          # This is exactly the same as the FTCS procedure above
          X := j -> -1 + j*2/(M+1);
          T := i -> tau*i/N;
          s := evalf(T(1)-T(0));
          h := evalf(X(1)-X(0));

          # Because we are formulating the stencil using matricies this time,
          # we define our data structures in terms of Vector's, not Array's
          XX := Vector([seq(X(j),j=1..M)],datatype=float);
          u_past := map(z->f(z),XX);

          # The plotting  routine has to be slightly modified to take into account that we are
          # using a different data structure for XX and u_past:
          p[0] := Frame[2](XX,u_past,s,h,T(0));

          # The key difference between this procedture and FTCS from above is that
          # we obtain u_future by multiplying u_past by a matrix P, which is defined
          # in terms of the _A procedure we defined above (3.8):
          A := _A(M,d,h):
          P := 1 + s*A:

          # Here is the loop over timesteps.  Notice the matrix multiplication:
          for i from 1 to N do:
               u_future := P.u_past:
               u_past := LinearAlgebra[Copy](u_future):
               p[i] := Frame[2](XX,u_past,s,h,T(i));
          od:

          # Here's the output movie:
          display(convert(p,list),insequence=true);

   end proc:

   # This is our new plotting routine, which assumes that xdata and ydata are Vector's
```

```
Frame[2] := proc(xdata,ydata,s,h,T) local Title, PlotOptions:

        Title := typeset(`timestep = `,evalf[2](s),`, spacestep = `,evalf[2](h),`, `,t=
evalf[2](T));
        PlotOptions := axes=boxed, labels=[x,u(t,x)], legend=["FTCS (matrix)"], color=
green;
        plot(Matrix([xdata,ydata]),title=Title,PlotOptions);

end proc:
```

We can compare the results of this procedure with the one from the previous section. Of course, they match perfectly:

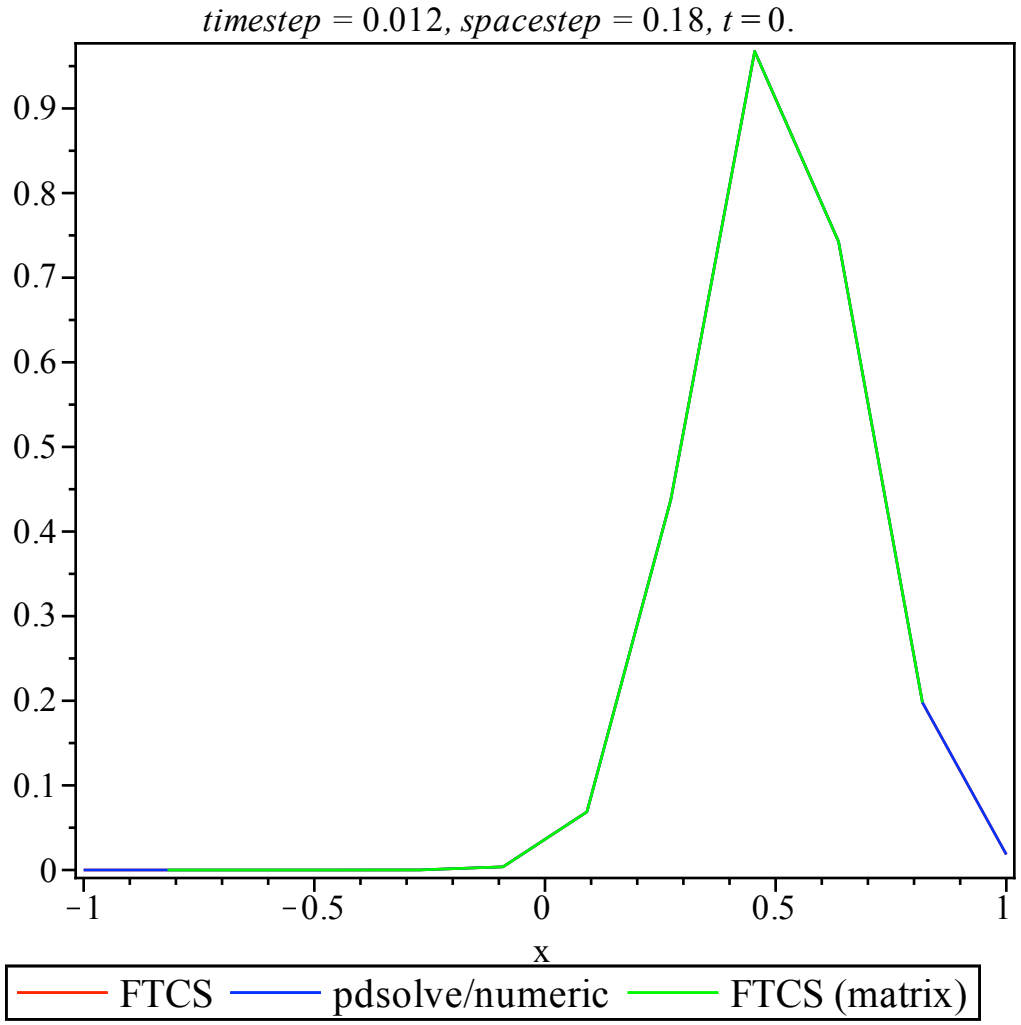
```
> f := x-> exp(-(4*(x-1/2))^2):
N := 80;
M := 10;
d := 1;
tau := 1;
movie[3] := FTCS_matrix(tau,N,M,d,f):
display(convert(movie,list));
```

$$N := 80$$

$$M := 10$$

$$d := 1$$

$$\tau := 1$$



BTCS and Crank-Nicholson implicit stencils

```
> d := 'd':
  s := 's':
  h := 'h':
```

We now construct some other stencils for solving the diffusion equation:

```
> pde;
```

$$\frac{\partial}{\partial t} u(t,x) - d\left(\frac{\partial^2}{\partial x^2} u(t,x)\right) \tag{4.1}$$

We construct a backward time stencil for the time derivative (that is, a left-oriented two-point stencil):

```
> backward_time := GenerateStencil(diff(u(t,x),t),2,orientation=left,stepsize=s);
```

This stencil is of order 1

$$backward\_time := \frac{\partial}{\partial t} u(t,x) = -\frac{u(t-s,x)}{s} + \frac{u(t,x)}{s} \tag{4.2}$$

Compare this to the forward time stencil **(2.2)**. Subbing this and the centered space stencil **(2.3)** into the diffusion equation results in the "backward-time centered=space (BTCS)" stencil:

```
> BTCS_stencil := subs(backward_time,centered_space,pde);
```

$$BTCS\_stencil := -\frac{u(t-s,x)}{s} + \frac{u(t,x)}{s} - d\left(\frac{u(t,x-h)}{h^2} - \frac{2\,u(t,x)}{h^2} + \frac{u(t,x+h)}{h^2}\right) \tag{4.3}$$

It is more common to write this stencil involving the time levels t and $t+s$ (rather than $t-s$ and t):

```
> BTCS_stencil := subs(t=t+s,BTCS_stencil);
```

$$BTCS\_stencil := -\frac{u(t,x)}{s} + \frac{u(t+s,x)}{s} - d\left(\frac{u(t+s,x-h)}{h^2} - \frac{2\,u(t+s,x)}{h^2} + \frac{u(t+s,x+h)}{h^2}\right) \tag{4.4}$$

We re-arrange things such that all the quantities evaluted at the future time $t+s$ are on the left and everything evaluated at the present time t are on the right:

```
> BTCS_stencil := expand(isolate(BTCS_stencil,u(t,x))):
  BTCS_stencil := rhs(BTCS_stencil) = lhs(BTCS_stencil);
```

$$BTCS\_stencil := u(t+s,x) - \frac{s\,d\,u(t+s,x-h)}{h^2} + \frac{2\,s\,d\,u(t+s,x)}{h^2} - \frac{s\,d\,u(t+s,x+h)}{h^2} = u(t,x) \tag{4.5}$$

This can now be directly compared to the FTCS stencil:

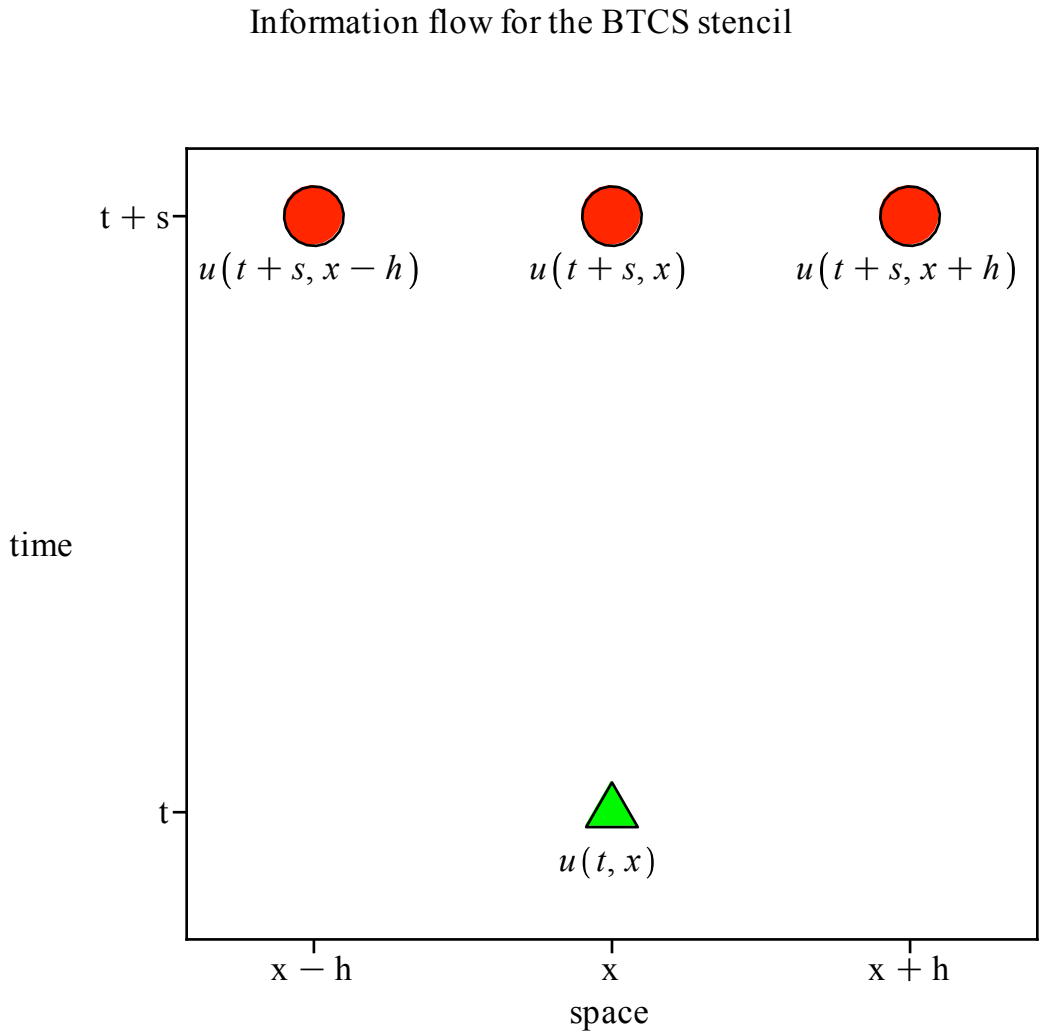
```
> FTCS_stencil;
```

$$u(t+s,x) = u(t,x) + \frac{s\,d\,u(t,x-h)}{h^2} - \frac{2\,s\,d\,u(t,x)}{h^2} + \frac{s\,d\,u(t,x+h)}{h^2} \tag{4.6}$$

Here is a diagram showing how the information flows for the BTCS stencil:

```
> display([seq(polygonplot(ngon(20,i,2,0.1,Pi/8),color=red),i=-1..1),polygonplot(ngon(3,0,0,
  0.1,Pi/2),color=green),textplot([seq([i,1.9,typeset(u(t+s,x+i*h))],i=-1..1),[0,-0.1,
  typeset(u(t,x))]],align={below})],view=[-1.4..1.4,-0.4..2.2],tickmarks=[[-1=x-h,0=x,1=x+
```

```
h],[0=t,2=t+s]],axes=boxed,scaling=constrained,labels=["space","time"],title="Information
flow for the BTCS stencil");
```



Equation **(4.5)** relates the field values at three future nodes (red circles) to the value at one past node (green triangle). Hence, unlike the FTCS stencil, we cannot use the BTCS stencil directly to calculate a future field value based on the past values. Stated another way: if we know $u(t, x)$ the BTCS stencil **(4.5)** is a single equation for three unknowns $\{u(t+s, x-h), u(t+s, x), u(t+s, x+h)\}$; i.e., the problem is underdetermined. The stencil is far from useless, however, as can be seen be examining the *system* of equation for $j = 1 \ldots M$. Here is an example of what the system looks like:

```
> sys := 'sys':
  Subs := [seq(seq(u(t+ii*s,x+jj*h)=u[i+ii,j+jj],jj=-1..1),ii=0..1)]:
  FTCS_general := subs(Subs,FTCS_stencil): #we will need this later
  BTCS_general := subs(Subs,BTCS_stencil);

  M := 4;

  u[i+1,0] := 0;
  u[i+1,M+1] := 0;

  for jj from 1 to M do:
     sys[jj] := eval(BTCS_general,j=jj);
  od;
  u := 'u':
```

$$BTCS\_general := u_{i+1,j} - \frac{s\,d\,u_{i+1,j-1}}{h^2} + \frac{2\,s\,d\,u_{i+1,j}}{h^2} - \frac{s\,d\,u_{i+1,j+1}}{h^2} = u_{i,j}$$

$$M := 4$$

$$u_{i+1,0} := 0$$

$$u_{i+1,5} := 0$$

$$sys_1 := u_{i+1,1} + \frac{2\,s\,d\,u_{i+1,1}}{h^2} - \frac{s\,d\,u_{i+1,2}}{h^2} = u_{i,1}$$

$$sys_2 := u_{i+1,2} - \frac{s\,d\,u_{i+1,1}}{h^2} + \frac{2\,s\,d\,u_{i+1,2}}{h^2} - \frac{s\,d\,u_{i+1,3}}{h^2} = u_{i,2}$$

$$sys_3 := u_{i+1,3} - \frac{s\,d\,u_{i+1,2}}{h^2} + \frac{2\,s\,d\,u_{i+1,3}}{h^2} - \frac{s\,d\,u_{i+1,4}}{h^2} = u_{i,3}$$

$$sys_4 := u_{i+1,4} - \frac{s\,d\,u_{i+1,3}}{h^2} + \frac{2\,s\,d\,u_{i+1,4}}{h^2} = u_{i,4} \qquad \textbf{(4.7}$$

We see **sys** is a system of M linear equations for the M unknowns $u_{i+1,j}$. We can recast this a matrix equation:

$$\mathbf{X}_i = \begin{bmatrix} u_{i,1} \\ u_{i,2} \\ \vdots \\ u_{i,M} \end{bmatrix}, \qquad \mathbf{P}\mathbf{X}_{i+1} = \mathbf{X}_i.$$

The explicit form of the **P** matrix can be found using **GenerateMatrix**:

```
> P := GenerateMatrix(convert(sys,list),[seq(u[i+1,jj],jj=1..M)])[1];
```

$$P := \begin{bmatrix} 1 + \frac{2\,s\,d}{h^2} & -\frac{s\,d}{h^2} & 0 & 0 \\ -\frac{s\,d}{h^2} & 1 + \frac{2\,s\,d}{h^2} & -\frac{s\,d}{h^2} & 0 \\ 0 & -\frac{s\,d}{h^2} & 1 + \frac{2\,s\,d}{h^2} & -\frac{s\,d}{h^2} \\ 0 & 0 & -\frac{s\,d}{h^2} & 1 + \frac{2\,s\,d}{h^2} \end{bmatrix} \qquad \textbf{(4.8}$$

So, in order to actually use the BTCS method to solve the diffusion equation, we need to solve a linear system $\mathbf{P} \cdot \mathbf{X}_{i+1} = \mathbf{X}_i$ at each timestep to determine the new field values from the old ones. Because the BTCS stencil does not give $\mathbf{X}_{i+1}$ explicitly, we call this an implicit method. Notice that the matrix in the linear system is tridiagonal, so it is not particularly difficult to solve for $\mathbf{X}_{i+1}$. This can actually be done rather efficiently in $O(M)$ time.

It is interesting to compare the BTCS stencil to the backward Euler solution of the matrix ODE

$$\frac{d}{dt}\mathbf{X}(t) = \mathbf{A}\mathbf{X}(t)$$

arising in the method of lines. Writing $\mathbf{X}_i \approx \mathbf{X}(t_i)$ as before, the backward Euler stencil for this equation is

$$\mathbf{X}_{i+1} = \mathbf{X}_i + s\mathbf{A} \cdot \mathbf{X}_{i+1} \qquad \Rightarrow \qquad (\mathbf{I} - s\mathbf{A})\mathbf{X}_{i+1} = \mathbf{X}_i.$$

The equation on the right will match what we found for the BTCS stencil if **P** = **I** - *s***A**. We can explicitly confirm that this is true with the _A procedure **(3.8)**:

```
> P - (1-s*_A(M,d,h));
```

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad \textbf{(4.9}$$

Hence, we see that the BTCS stencil is nothing more that the backward Euler solution of the matrix equation arising from the method of line (just as the FTCS stencil corresponded to the forward Euler method). What is the PDE analog of the trapezoidal method applied to the method of lines matrix ODE? Recall that the trapezoidal method was the average of the forward Euler and backward Euler scheme. Hence we define a new stencil, called the "Crank-Nicholson" stencil after its authors, to be the average of the FTCS and BTCS stencils:

```
> CN_stencil := (FTCS_stencil+BTCS_stencil)/2;
```

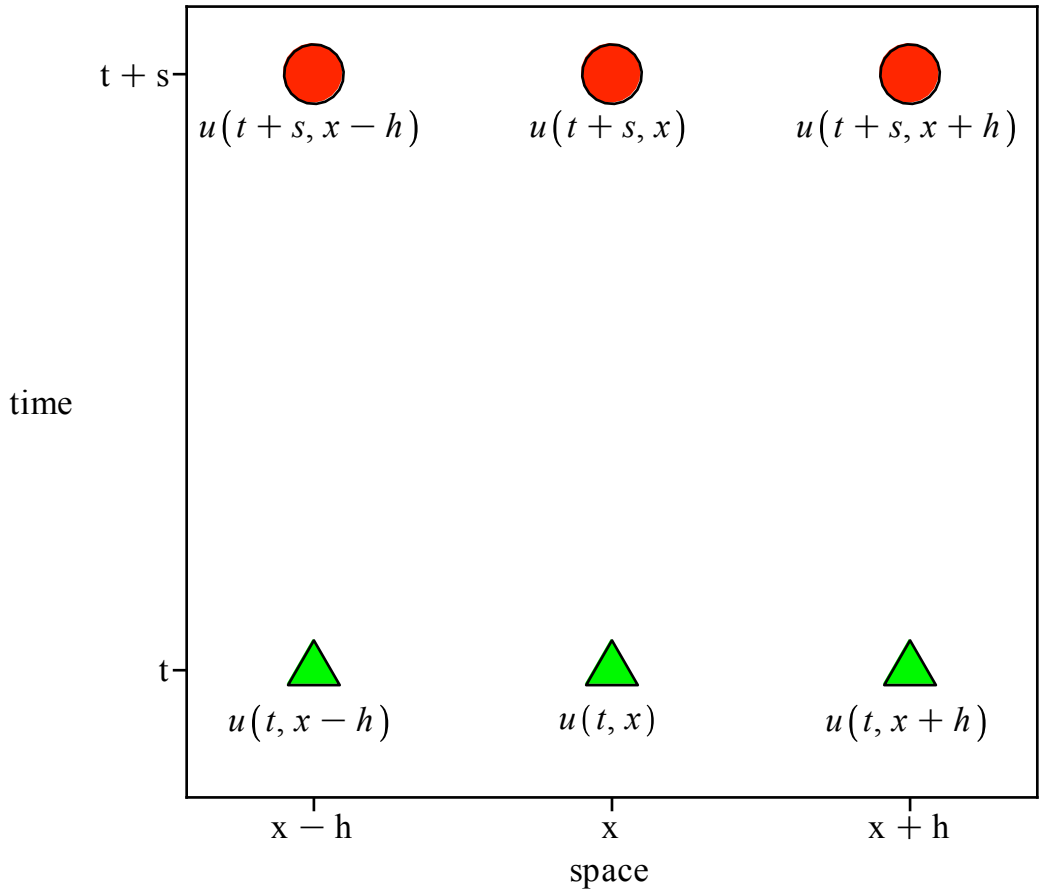
$$CN\_stencil := u(t+s,x) - \frac{1}{2}\frac{s\,d\,u(t+s,x-h)}{h^2} + \frac{s\,d\,u(t+s,x)}{h^2} - \frac{1}{2}\frac{s\,d\,u(t+s,x+h)}{h^2} = u(t,x) \qquad \textbf{(4.10}$$

$$+ \frac{1}{2}\frac{s\,d\,u(t,x-h)}{h^2} - \frac{s\,d\,u(t,x)}{h^2} + \frac{1}{2}\frac{s\,d\,u(t,x+h)}{h^2}$$

Here is a sketch of the information flow:

```
> display([seq(polygonplot(ngon(20,i,2,0.1,Pi/8),color=red),i=-1..1),seq(polygonplot(ngon(3,
  i,0,0.1,Pi/2),color=green),i=-1..1),textplot([seq([i,1.9,typeset(u(t+s,x+i*h))],i=-1..1),
  seq([i,-0.1,typeset(u(t,x+i*h))],i=-1..1)],align={below})],view=[-1.4..1.4,-0.4..2.2],
  tickmarks=[[-1=x-h,0=x,1=x+h],[0=t,2=t+s]],axes=boxed,scaling=constrained,labels=["space",
  "time"],title="Information for for the Crank-Nicholson stencil");
```

Information for for the Crank-Nicholson stencil



The Crank-Nicholson stencil relates three future values to three past values. This means that, as for the BTCS stencil, **(4.10)** is an undetermined equation for the future field values if the past values are known. However, just as for the BTCS case, if we write out all the equations for $j = 1 \ldots M$ we see we have a linear system of equations to solve:

```
> sys := 'sys':
  Subs := [seq(seq(u(t+ii*s,x+jj*h)=u[i+ii,j+jj],jj=-1..1),ii=0..1)]:
  CN_general := subs(Subs,CN_stencil);
```

```
M := 5;

u[i+1,0] := 0;
u[i+1,M+1] := 0;
u[i,0] := 0;
u[i,M+1] := 0;
for jj from 1 to M do:
   sys[jj] := eval(CN_general,j=jj);
od;
u := 'u':
```

$$CN\_general := u_{i+1,j} - \frac{1}{2}\,\frac{s\,d\,u_{i+1,j-1}}{h^2} + \frac{s\,d\,u_{i+1,j}}{h^2} - \frac{1}{2}\,\frac{s\,d\,u_{i+1,j+1}}{h^2} = u_{i,j} + \frac{1}{2}\,\frac{s\,d\,u_{i,j-1}}{h^2} - \frac{s\,d\,u_{i,j}}{h^2} + \frac{1}{2}\,\frac{s\,d\,u_{i,j+1}}{h^2}$$

$$M := 5$$

$$u_{i+1,0} := 0$$

$$u_{i+1,6} := 0$$

$$u_{i,0} := 0$$

$$u_{i,6} := 0$$

$$sys_1 := u_{i+1,1} + \frac{s\,d\,u_{i+1,1}}{h^2} - \frac{1}{2}\,\frac{s\,d\,u_{i+1,2}}{h^2} = u_{i,1} - \frac{s\,d\,u_{i,1}}{h^2} + \frac{1}{2}\,\frac{s\,d\,u_{i,2}}{h^2}$$

$$sys_2 := u_{i+1,2} - \frac{1}{2}\,\frac{s\,d\,u_{i+1,1}}{h^2} + \frac{s\,d\,u_{i+1,2}}{h^2} - \frac{1}{2}\,\frac{s\,d\,u_{i+1,3}}{h^2} = u_{i,2} + \frac{1}{2}\,\frac{s\,d\,u_{i,1}}{h^2} - \frac{s\,d\,u_{i,2}}{h^2} + \frac{1}{2}\,\frac{s\,d\,u_{i,3}}{h^2}$$

$$sys_3 := u_{i+1,3} - \frac{1}{2}\,\frac{s\,d\,u_{i+1,2}}{h^2} + \frac{s\,d\,u_{i+1,3}}{h^2} - \frac{1}{2}\,\frac{s\,d\,u_{i+1,4}}{h^2} = u_{i,3} + \frac{1}{2}\,\frac{s\,d\,u_{i,2}}{h^2} - \frac{s\,d\,u_{i,3}}{h^2} + \frac{1}{2}\,\frac{s\,d\,u_{i,4}}{h^2}$$

$$sys_4 := u_{i+1,4} - \frac{1}{2}\,\frac{s\,d\,u_{i+1,3}}{h^2} + \frac{s\,d\,u_{i+1,4}}{h^2} - \frac{1}{2}\,\frac{s\,d\,u_{i+1,5}}{h^2} = u_{i,4} + \frac{1}{2}\,\frac{s\,d\,u_{i,3}}{h^2} - \frac{s\,d\,u_{i,4}}{h^2} + \frac{1}{2}\,\frac{s\,d\,u_{i,5}}{h^2}$$

(4.11

$$sys_5 := u_{i+1,5} - \frac{1}{2}\,\frac{s\,d\,u_{i+1,4}}{h^2} + \frac{s\,d\,u_{i+1,5}}{h^2} = u_{i,5} + \frac{1}{2}\,\frac{s\,d\,u_{i,4}}{h^2} - \frac{s\,d\,u_{i,5}}{h^2} \qquad \textbf{(4.11}$$

This system is moderately more complicated than the BTCS case **(4.7)**, but it can still be cast in matrix form:

$$\mathbf{X}_i = \begin{bmatrix} u_{i,1} \\ u_{i,2} \\ \vdots \\ u_{i,M} \end{bmatrix}, \qquad \mathbf{P}_L\mathbf{X}_{i+1} = \mathbf{P}_R\mathbf{X}_i.$$

Explicit forms of $\mathbf{P}_L$ and $\mathbf{P}_R$ are again obtained using **GenerateMatrix**:

```
> P := 'P':
  sys := convert(sys,list):
  future := [seq(u[i+1,jj],jj=1..M)];
  past := [seq(u[i,jj],jj=1..M)];
  P[L] := GenerateMatrix(sys,future)[1];
  _sys := map(x->rhs(x)=lhs(x),sys):
  P[R] := GenerateMatrix(_sys,past)[1];
```

$$future := \left[u_{i+1,1}, u_{i+1,2}, u_{i+1,3}, u_{i+1,4}, u_{i+1,5}\right]$$

$$past := \left[u_{i,1}, u_{i,2}, u_{i,3}, u_{i,4}, u_{i,5}\right]$$

$$P_L := \begin{bmatrix} 1+\frac{s\,d}{h^2} & -\frac{1}{2}\frac{s\,d}{h^2} & 0 & 0 & 0 \\ -\frac{1}{2}\frac{s\,d}{h^2} & 1+\frac{s\,d}{h^2} & -\frac{1}{2}\frac{s\,d}{h^2} & 0 & 0 \\ 0 & -\frac{1}{2}\frac{s\,d}{h^2} & 1+\frac{s\,d}{h^2} & -\frac{1}{2}\frac{s\,d}{h^2} & 0 \\ 0 & 0 & -\frac{1}{2}\frac{s\,d}{h^2} & 1+\frac{s\,d}{h^2} & -\frac{1}{2}\frac{s\,d}{h^2} \\ 0 & 0 & 0 & -\frac{1}{2}\frac{s\,d}{h^2} & 1+\frac{s\,d}{h^2} \end{bmatrix}$$

$$P_R := \begin{bmatrix} 1-\frac{s\,d}{h^2} & \frac{1}{2}\frac{s\,d}{h^2} & 0 & 0 & 0 \\ \frac{1}{2}\frac{s\,d}{h^2} & 1-\frac{s\,d}{h^2} & \frac{1}{2}\frac{s\,d}{h^2} & 0 & 0 \\ 0 & \frac{1}{2}\frac{s\,d}{h^2} & 1-\frac{s\,d}{h^2} & \frac{1}{2}\frac{s\,d}{h^2} & 0 \\ 0 & 0 & \frac{1}{2}\frac{s\,d}{h^2} & 1-\frac{s\,d}{h^2} & \frac{1}{2}\frac{s\,d}{h^2} \\ 0 & 0 & 0 & \frac{1}{2}\frac{s\,d}{h^2} & 1-\frac{s\,d}{h^2} \end{bmatrix} \quad \textbf{(4.12}$$

Now the trapezoidal stencil for

$$\frac{\mathrm{d}}{\mathrm{d}\,t}\mathbf{X}(t) = \mathbf{A}\mathbf{X}(t)$$

is

$$\mathbf{X}_{i+1} = \mathbf{X}_i + \frac{s}{2}\left(\mathbf{A}\mathbf{X}_{i+1} + \mathbf{A}\mathbf{X}_{i+1}\right) \quad \Rightarrow \quad \left(\mathbf{I} - \frac{s}{2}\mathbf{A}\right)\mathbf{X}_{i+1} = \left(\mathbf{I} + \frac{s}{2}\mathbf{A}\right)\mathbf{X}_i.$$

We can confirm $\mathbf{P}_L = \left(\mathbf{I} - \frac{1}{2}s\mathbf{A}\right)$ and $\mathbf{P}_R = \left(\mathbf{I} + \frac{1}{2}s\mathbf{A}\right)$, which means that this is the same linear system as the one arising from the Crank-Nicolson stencil **(4.11)**:

```
> P[L] - (1-1/2*s*_A(M,d,h)), P[R] - (1+1/2*s*_A(M,d,h));
```

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad \textbf{(4.13}$$

Here is some code using the Crank-Nicholson stencil to find the solution of the diffusion equation. Note we use the `LinearSolve` command to solve $\mathbf{P}_L\mathbf{X}_{i+1} = \mathbf{P}_R\mathbf{X}_i$ at each time level. (Note: coding the BTCS is similar, so we don't do this here):

```
> CN := proc(tau,N,M,d,f)
          local X, T, s, h, XX, u_past, u_future, Title, PlotOptions, p, A, P, i:

          # Defining the lattice parameters/functions as before:
          X := j -> -1 + j*2/(M+1);
          T := i -> tau*i/N;
          s := evalf(T(1)-T(0));
          h := evalf(X(1)-X(0));

          # Set the initial data
          XX := Vector([seq(X(j),j=1..M)],datatype=float);
          u_past := map(z->f(z),XX);

          # Plot the first frame
          p[0] := Frame[3](XX,u_past,s,h,T(0)):

          # At each time step, we will solve the linear system P[1].u_future = P[2].u_past
          # for u_future. Here we define the coefficient matrices:
          A := _A(M,d,h):
          P[1] := 1 + s/2*A:
```

```
        P[2] := 1 - s/2*A:

        # Here is the main calculation loop (notice the LinearSolve command)
        for i from 1 to N do:
             u_future := LinearSolve(P[2],P[1].u_past):
             u_past := LinearAlgebra[Copy](u_future):
             p[i] := Frame[3](XX,u_past,s,h,T(i)):
        od:

        # Assemble the output movie
        display(convert(p,list),insequence=true);

end proc:

# Here is the plotting subroutine

Frame[3] := proc(xdata,ydata,s,h,T) local Title, PlotOptions:

        Title := typeset(`timestep = `,evalf[2](s),`, spacestep = `,evalf[2](h),`, `,t=
evalf[2](T));
        PlotOptions := axes=boxed, labels=[x,u(t,x)], legend=["Crank-Nicholson"], color=
magenta;
        plot(Matrix([xdata,ydata]),title=Title,PlotOptions);

end proc:
```

Here is a movie of the Crank-Nicholson output compared to the other methods we have studied:

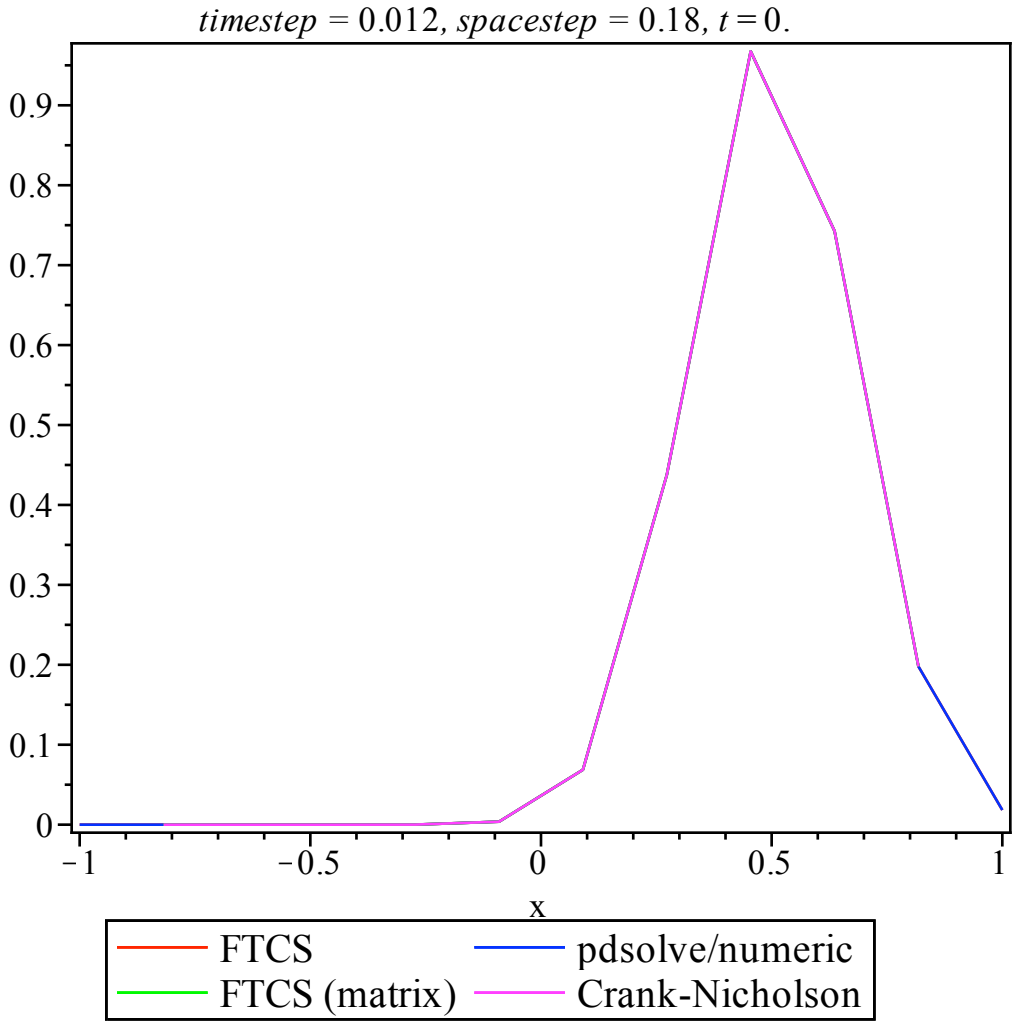
```
> f := x-> exp(-(4*(x-1/2))^2):
  N := 80;
  M := 10;
  d := 1;
  tau := 1;
  movie[4] := CN(tau,N,M,d,f):
  display(convert(movie,list));
```

$$N := 80$$

$$M := 10$$

$$d := 1$$

$$\tau := 1$$



Numeric examples of unstable behaviour

```
> movie := 'movie':
```

A legitimate question is: why bother with stencils such as BTCS or Crank-Nicholson (which involve the solution of a potentially large system of equations at each timestep) over the simpler FTCS method? The answer is the stability of the various schemes. Let's compare the output from the FTCS and Crank-Nicholson schemes for the following choice of parameters:

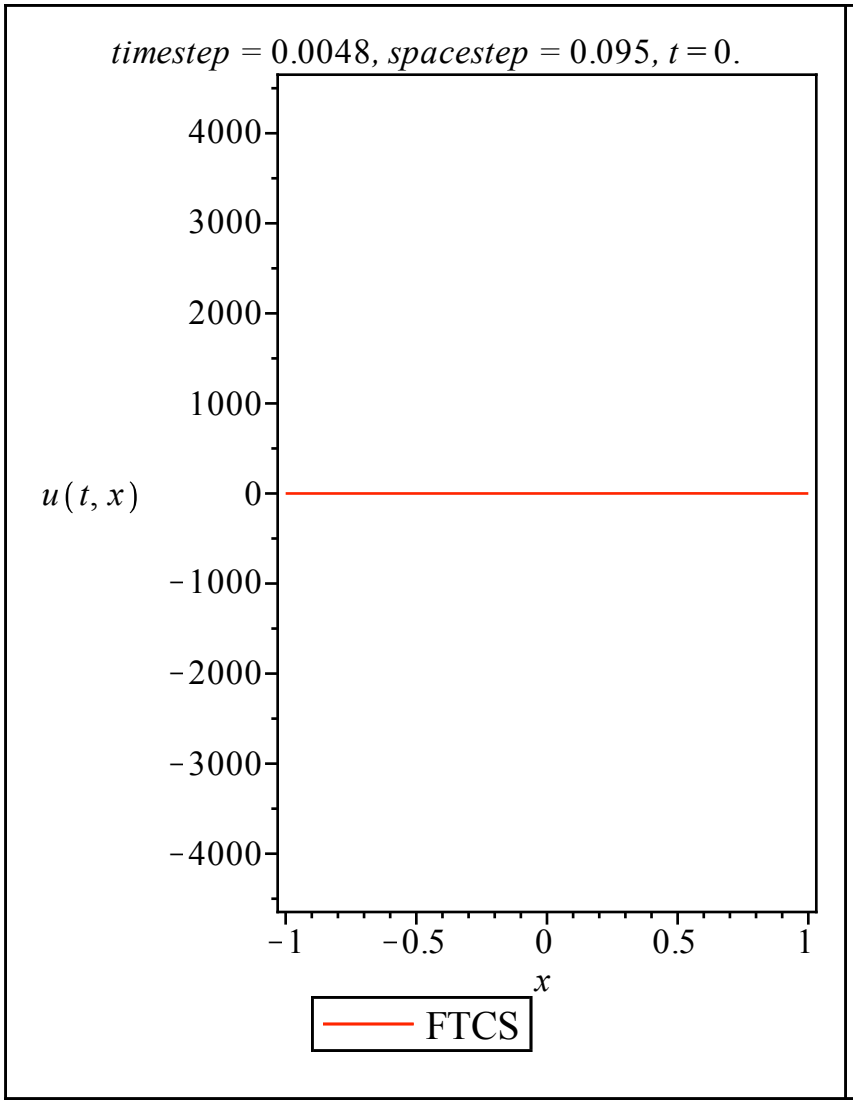
```
> f := x-> exp(-(4*(x-1/2))^2):
  N := 210;
  M := 20;
  d := 1;
  tau := 1;
  movie[1] := FTCS(tau,N,M,d,f):
  movie[2] := CN(tau,N,M,d,f):
  display(Array([movie[1],movie[2]]));
```

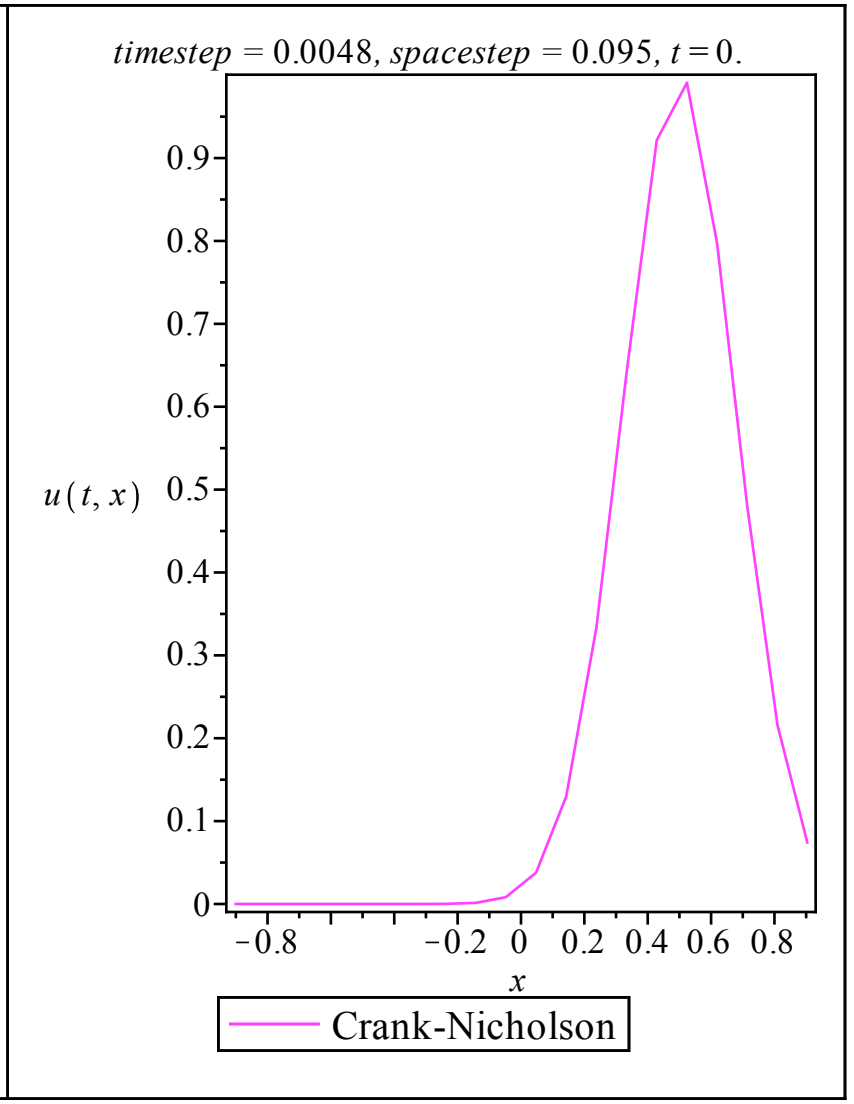
$$N := 210$$

$$M := 20$$

$$d := 1$$

$$\tau := 1$$





Clearly, something is going very wrong with the FTCS method, while the CN one is returning reasonable results. Just a small increase in the spatial stepsize seems to cure the problem:

```
> f := x-> exp(-(4*(x-1/2))^2):
  N := 210;
  M := 19;
  d := 1;
```

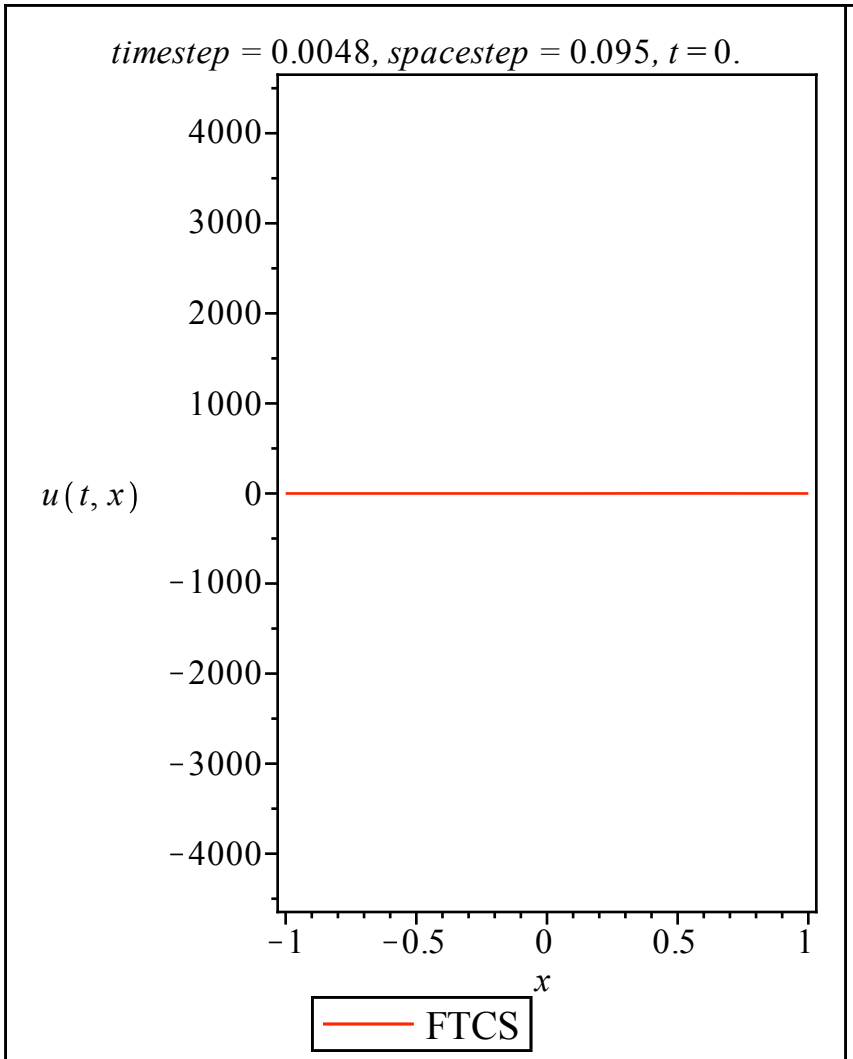
```
tau := 1;
movie[3] := FTCS(tau,N,M,d,f):
movie[4] := CN(tau,N,M,d,f):
display(Array([[movie[1],movie[2]],[movie[3],movie[4]]]));
```

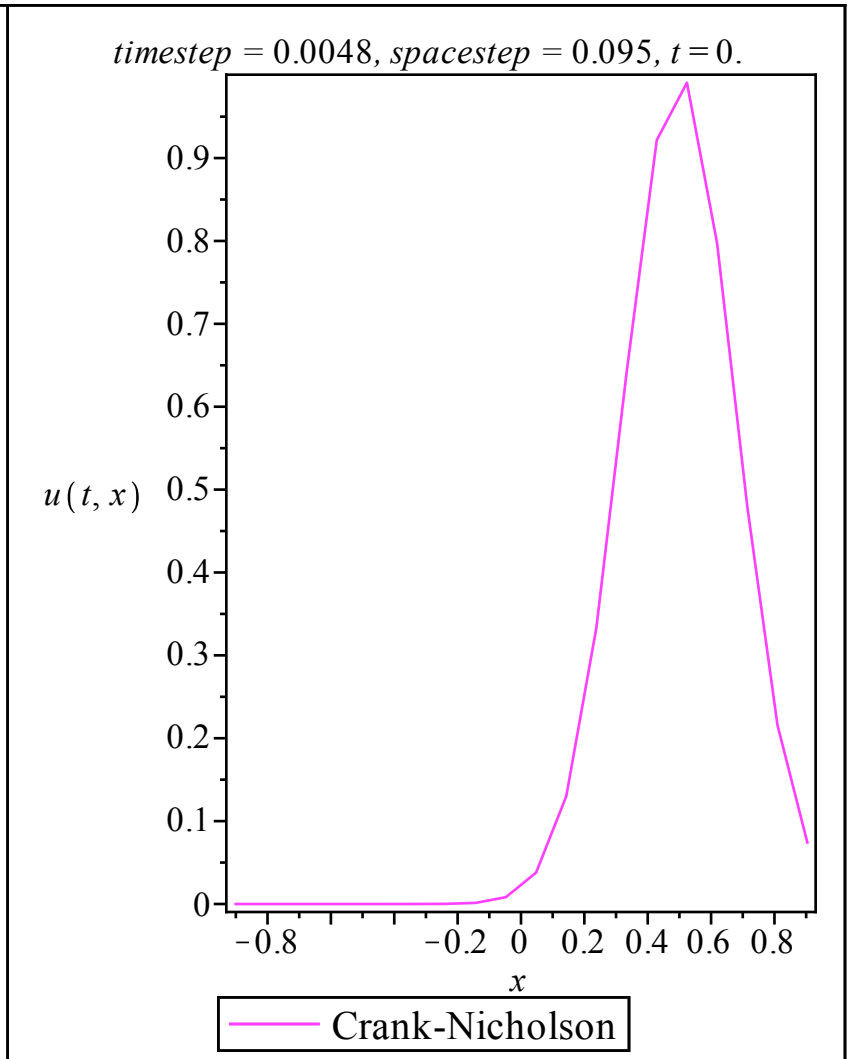
$$N := 210$$

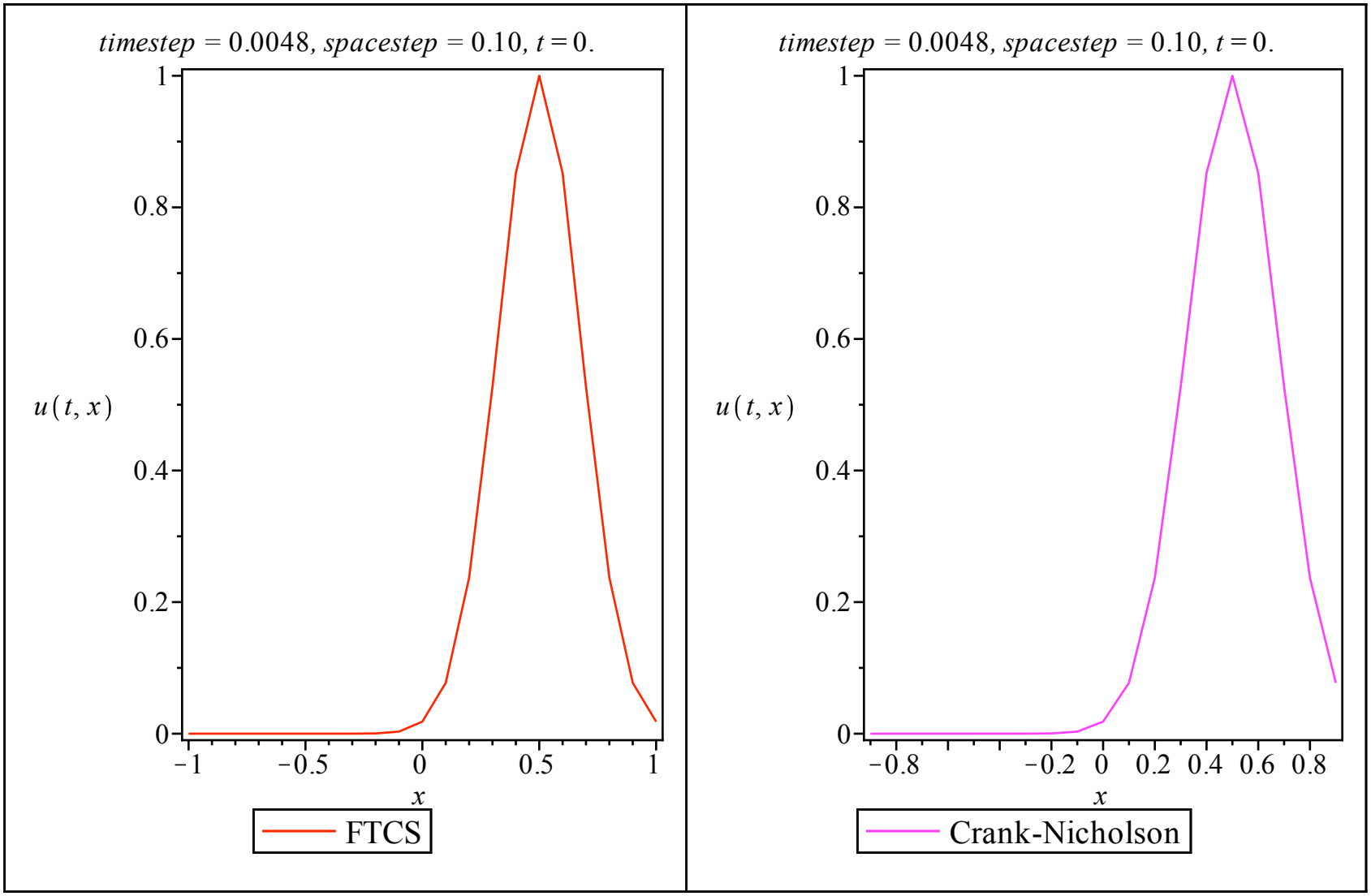
$$M := 19$$

$$d := 1$$

$$\tau := 1$$


timestep = 0.0048, spacestep = 0.095, t = 0.
4000
3000
2000
1000
0
−1000
−2000
−3000
−4000
u(t, x)
−1
−0.5
0
0.5
1
x
FTCS


timestep = 0.0048, spacestep = 0.095, t = 0.
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
u(t, x)
−0.8
−0.2
0
0.2
0.4
0.6
0.8
x
Crank-Nicholson



It seems that the FTCS method is *conditionally stable*; it seems to give sensible results for some choices of parameters but not others. Experimenting with different N and M for the Crank-Nicholson code will convince you that its output for $u(t, x)$ is always bounded and not exponentially diverging; that is, we have some empirical evidence that this method is *unconditionally stable*. We will justify these empirical conclusions in the next section.

Method of lines stability analysis

```
> M := 'M':
  d := 'd':
  h := 'h':
  A := 'A':
  lambda := 'lambda':
```

To understand the empirical stability results of the previous section, we consider the method of lines formulation of the diffusion equation:

$$\frac{\mathrm{d}}{\mathrm{d}\,t}\mathbf{X}(t) = \mathbf{A}\mathbf{X}(t).$$

All of the stencils introduced so far can be rewritten as

$$\mathbf{X}_{i+1} = \mathbf{P}\mathbf{X}_i, \qquad \mathbf{X}_i = \mathbf{X}\left(t_i\right) + \mathrm{O}\left(s^p\right)$$

where we follow our usual notation that $\mathbf{X}(t)$ represents the true solution of the matrix ODE. We define the error in the usual way $\mathbf{E}_i = \mathbf{X}_i - \mathbf{X}\left(t_i\right)$, hence

$$\mathbf{E}_{i+1} = \mathbf{P}\mathbf{E}_i + \mathrm{O}\left(s^p\right).$$

Now, for the particular case of the FTCS stencil $\mathbf{P} = \mathbf{I} + s\mathbf{A}$, so we obtain $\mathbf{E}_{i+1} \approx \mathbf{E}_i + s\mathbf{A}\mathbf{E}_i$ (dropping the local error term). Now, consider a matrix Λ whose columns are the eigenvectors of $\mathbf{A}$. It follows that $\mathbf{B} = \Lambda^{-1}\mathbf{A}\Lambda$ is a diagonal matrix whose entries are the eigenvalues $\left\{\lambda_j\right\}_{j=1}^{M}$ of $\mathbf{A}$. If we define $\mathbf{F}_i = \Lambda^{-1}\mathbf{E}_i$, it follows that

$$\mathbf{F}_{i+1} \approx \mathbf{F}_i + s\Lambda^{-1}\mathbf{A}\Lambda\mathbf{F}_i = (\mathbf{I} + s\mathbf{B})\mathbf{F}_i = \begin{bmatrix} P\left(s\lambda_1\right) & & & \\ & P\left(s\lambda_2\right) & & \\ & & \ddots & \\ & & & P\left(s\lambda_4\right) \end{bmatrix}\mathbf{F}_i,$$

where $P(z) = 1 + z$ is the absolute stability function for the forward Euler ODE stencil. If any of the entries of the diagonal matrix $\mathbf{I} + s\mathbf{B}$

have absolute value greater than 1, at least one component of $\mathbf{F}_i$ will exponentially diverge as $i \to \infty$. Since the error is linearly related to $\mathbf{F}_i$ via $\mathbf{E}_i = \Lambda \mathbf{F}_i$, then we conclude that if one component of $\mathbf{F}_i$ diverges then the error in the stencil will also diverge. Hence, we have the following definition

Definition: The FTCS stencil is *absolutely stable* if $|P(s\lambda_j)| \leq 1$ for $j = 1 \ldots M$, where $|P(z)| = |1 + z| \leq 1$ is the absolute stability criterion for the forward Euler ODE stencil and the λ_j are the eigenvalues of **A**.

It is not difficult to repeat this analysis for the BTCS and Crank-Nicholson stencils, which lead to the following definitions:

Definition: The BTCS stencil is *absolutely stable* if $|P(s\lambda_j)| \leq 1$ for $j = 1 \ldots M$, where $\left|P(z)\right| = \left|\frac{1}{1 - z}\right| \leq 1$ is the absolute stability criterion for the backward Euler ODE stencil and the λ_j are the eigenvalues of **A**.

Definition: The Crank-Nicholson stencil is *absolutely stable* if $|P(s\lambda_j)| \leq 1$ for $j = 1 \ldots M$, where $\left|P(z)\right| = \left|\frac{2 + z}{2 - z}\right| \leq 1$ is the absolute stability criterion for the trapeziodal ODE stencil and the λ_j are the eigenvalues of **A**.

Now, in order to actually use these stability criteria, we need to know the eigenvalues of **A**. Recall our procedure to calculate this matrix:

```
> M := 5;
  _A(M,d,h);
```

$$M := 5$$

(6.1

$$\begin{bmatrix} -\frac{2\,d}{h^2} & \frac{d}{h^2} & 0 & 0 & 0 \\ \frac{d}{h^2} & -\frac{2\,d}{h^2} & \frac{d}{h^2} & 0 & 0 \\ 0 & \frac{d}{h^2} & -\frac{2\,d}{h^2} & \frac{d}{h^2} & 0 \\ 0 & 0 & \frac{d}{h^2} & -\frac{2\,d}{h^2} & \frac{d}{h^2} \\ 0 & 0 & 0 & \frac{d}{h^2} & -\frac{2\,d}{h^2} \end{bmatrix} \quad \textbf{(6.1}$$

Here are the eigenvalues:

```
> _lambda := simplify(Eigenvalues(_A(M,d,h)));
```

$$\_\lambda := \begin{bmatrix} -\frac{2\,d}{h^2} \\ -\frac{d}{h^2} \\ -\frac{3\,d}{h^2} \\ \frac{\left(-2+\sqrt{3}\right)d}{h^2} \\ -\frac{\left(2+\sqrt{3}\right)d}{h^2} \end{bmatrix} \quad \textbf{(6.2}$$

Notice that the eigenvalues are all real (this is true for all $M > 1$). Here are the maximum and minimum eigenvalues:

```
> evalf(max(_lambda)) assuming(d/h^2>0);
  evalf(min(_lambda)) assuming(d/h^2>0);
```

$$-\frac{0.267949192\,d}{h^2}$$

$$-\frac{3.732050808\,d}{h^2} \qquad \textbf{(6.3}$$

So we see that the eigenvalues are also all negative (again this is true for all $M > 1$). Here are procedures that find the maximum and minimum eigenvalues (in units of dh^{-2}):

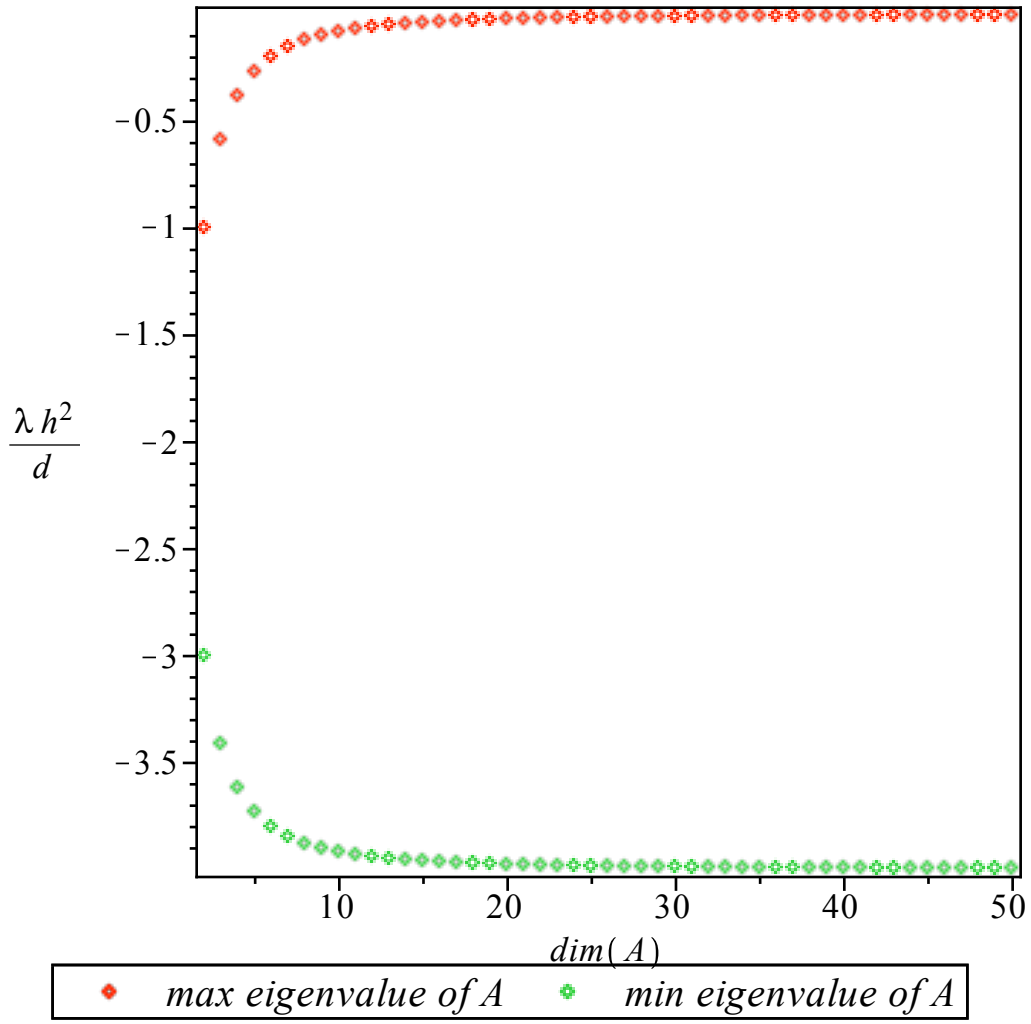
```
> lambda[max] := M -> max(Eigenvalues(evalf(_A(M,1,1))));
  lambda[min] := M -> min(Eigenvalues(evalf(_A(M,1,1))));
```

$$\lambda_{max} := M \rightarrow \max(\mathit{LinearAlgebra}\text{:-}\mathit{Eigenvalues}(\mathit{evalf}(\_A(M, 1, 1))))$$

$$\lambda_{min} := M \rightarrow \min(\mathit{LinearAlgebra}\text{:-}\mathit{Eigenvalues}(\mathit{evalf}(\_A(M, 1, 1)))) \qquad \textbf{(6.4}$$

And here is a plot of the extremal eigenvalues as a function of M:

```
> max_data := [seq([M,lambda[max](M)],M=2..50)]:
  min_data := [seq([M,lambda[min](M)],M=2..50)]:
  plot([max_data,min_data],axes=boxed,labels=[typeset(dim(A)),typeset(lambda/d*h^2)],legend=
  [`max eigenvalue of A`,`min eigenvalue of A`],style=point);
```



It appears from this plot that in the limit of $M \to \infty$ the eigenvalues lie in the interval $\lambda_n \in \left(-\frac{4\,d}{h^2}, 0 \right)$. This empirical observation is actually true; one can show analytically (but not that easily) that the eigenvalues of **A** are given by ($n = 1 \ldots M$):

$$\lambda_n = \frac{2\,d}{h^2}\left[\cos\left(\frac{n\pi}{M+1}\right) - 1\right] \quad \Rightarrow \quad \lambda_n \in \left(-\frac{4\,d}{h^2}, 0 \right).$$

Now, for the FTCS method to be stable we require $|P(s\lambda_n)| = |1 + s\lambda_n| \leq 1$ for all n. The above bounds on λ_n mean that the inequality is satisfied if $\frac{2\,sd}{h^2} \leq 1$. This is the stability condition for the FTCS stencil of the diffusion equation.

For the BTCS stencil, stability requires $|P(s\lambda_n)| = \left|\frac{1}{1 - s\lambda_n}\right| \leq 1$, which is identially true for all the eigenvalues (since $\lambda_n < 0$). Hence, the BTCS method will be unconditionally stable.

For the Crank-Nicholson, we would need $|P(s\lambda_n)| = \left|\frac{2 + s\lambda_n}{2 - s\lambda_n}\right| \leq 1$, which is again identically true for all λ_n. Hence, the Crank-Nicholson stencil is unconditionally stable.

von Neumann stability analysis

```
> i := 'i':
  Error := 'Error':
```

The above stability analysis based on the method of lines is only rigorous if we have analytical knowledge of the eigenvalues of **A**. Fortunately, we had such knowledge for the matrix associated with the diffusion equation, but we won't have that luxury for the coefficient matrices arising from different PDEs with different types of boundary conditions or finite differencing schemes. For this reason, it is useful to have an alternate, more user-friendly, procedure for determining stability. This is the *von Neumann stability analysis* which we describe below. Here are the stencils we will analyze:

```
> Labels := [`FTCS`,`BTCS`,`Crank-Nicholson`];

  Stencil[1] := FTCS_general;
  Stencil[2] := BTCS_general;
  Stencil[3] := CN_general;
```

$$Labels := [FTCS, BTCS, Crank\text{-}Nicholson]$$

$$Stencil_1 := u_{i+1,j} = u_{i,j} + \frac{s\,d\,u_{i,j-1}}{h^2} - \frac{2\,s\,d\,u_{i,j}}{h^2} + \frac{s\,d\,u_{i,j+1}}{h^2}$$

$$Stencil_2 := u_{i+1,j} - \frac{s\,d\,u_{i+1,j-1}}{h^2} + \frac{2\,s\,d\,u_{i+1,j}}{h^2} - \frac{s\,d\,u_{i+1,j+1}}{h^2} = u_{i,j}$$

(7.1

$$Stencil_3 := u_{i+1,j} - \frac{1}{2}\frac{s\,d\,u_{i+1,j-1}}{h^2} + \frac{s\,d\,u_{i+1,j}}{h^2} - \frac{1}{2}\frac{s\,d\,u_{i+1,j+1}}{h^2} = u_{i,j} + \frac{1}{2}\frac{s\,d\,u_{i,j-1}}{h^2} - \frac{s\,d\,u_{i,j}}{h^2} + \frac{1}{2}\frac{s\,d\,u_{i,j+1}}{h^2} \quad \textbf{(7.1}$$

The above stencil are exact relations between our numeric approximation $u_{i,j} \approx u(t_i, x_j)$ at various nodes. Replacing the numerical solutio with the exact solution means that the relations are only true up to the one-step error in the stencil, denoted by **one_step_error** here:

```
> Exact[1] := applyop(x->x+one_step_error,2,FTCS_stencil);
  Exact[2] := applyop(x->x+one_step_error,2,BTCS_stencil);
  Exact[3] := applyop(x->x+one_step_error,2,CN_stencil);
```

$$Exact_1 := u(t+s,x) = u(t,x) + \frac{s\,d\,u(t,x-h)}{h^2} - \frac{2\,s\,d\,u(t,x)}{h^2} + \frac{s\,d\,u(t,x+h)}{h^2} + one\_step\_error$$

$$Exact_2 := u(t+s,x) - \frac{s\,d\,u(t+s,x-h)}{h^2} + \frac{2\,s\,d\,u(t+s,x)}{h^2} - \frac{s\,d\,u(t+s,x+h)}{h^2} = u(t,x) + one\_step\_error$$

$$Exact_3 := u(t+s,x) - \frac{1}{2}\frac{s\,d\,u(t+s,x-h)}{h^2} + \frac{s\,d\,u(t+s,x)}{h^2} - \frac{1}{2}\frac{s\,d\,u(t+s,x+h)}{h^2} = u(t,x) + \frac{1}{2}\frac{s\,d\,u(t,x-h)}{h^2} - \frac{s\,d\,u(t,x)}{h^2} + \frac{1}{2}\frac{s\,d\,u(t,x+h)}{h^2} + one\_step\_error \quad \textbf{(7.2}$$

We define the error at the (i,j) node in the usual way: $E_{i,j} = u_{i,j} - u(t_i, x_j)$. This will allow us to replace the true solution in the above equations with the numeric solution and the error using the following substitutions:

```
> Subs := [seq(seq(u(t+ii*s,x+jj*h)=u[i+ii,j+jj]-E[i+ii,j+jj],ii=0..1),jj=-1..1)];
```

$$Subs := \left[u(t,x-h) = u_{i,j-1} - E_{i,j-1}, u(t+s,x-h) = u_{i+1,j-1} - E_{i+1,j-1}, u(t,x) = u_{i,j} - E_{i,j}, u(t+s,x) = u_{i+1,j} - E_{i+1,j}, u(t,x+h) = u_{i,j+1} - E_{i,j+1}, u(t+s,x+h) = u_{i+1,j+1} - E_{i+1,j+1}\right] \quad \textbf{(7.3}$$

Subtracting the stencils evaluated on the exact solution **(7.2)** from the stencils evaluated on the numeric solutions **(7.1)** results in the following equations of motion for the error:

```
> for ii from 1 to 3 do:
      EOM[ii] := expand(simplify(subs(Subs,Stencil[ii]-Exact[ii]))):
  od;
```

$$EOM_1 := E_{i+1,j} = E_{i,j} + \frac{s\,d\,E_{i,j-1}}{h^2} - \frac{2\,s\,d\,E_{i,j}}{h^2} + \frac{s\,d\,E_{i,j+1}}{h^2} - one\_step\_error$$

$$EOM_2 := E_{i+1,j} - \frac{s\,d\,E_{i+1,j-1}}{h^2} + \frac{2\,s\,d\,E_{i+1,j}}{h^2} - \frac{s\,d\,E_{i+1,j+1}}{h^2} = E_{i,j} - one\_step\_error$$

$$EOM_3 := E_{i+1,j} - \frac{1}{2}\,\frac{s\,d\,E_{i+1,j-1}}{h^2} + \frac{s\,d\,E_{i+1,j}}{h^2} - \frac{1}{2}\,\frac{s\,d\,E_{i+1,j+1}}{h^2} = E_{i,j} + \frac{1}{2}\,\frac{s\,d\,E_{i,j-1}}{h^2} - \frac{s\,d\,E_{i,j}}{h^2} \quad \textbf{(7.4}$$

$$+ \frac{1}{2}\,\frac{s\,d\,E_{i,j+1}}{h^2} - one\_step\_error$$

Notice that these are identical to the numeric stencils **(7.1)** with the switch $u \mapsto E$ up to one-step error term. If we neglect the latter, these represent linear difference equations for the errors. This means that solutions will obey a superposition principle; that is, if we have two distinct solutions for $E_{i,j}$, any linear combination of these solutions will also be a solution. Because of this linearity, we can attempt a Fourier decomposition of $E_{i,j}$. This is motivated from the fact that the original PDE can be converted to an ODE via a Fourier decomposition:

```
> pde;
  fourier := u(t,x) = Xi(t)*exp(I*k*x);
  isolate(dsubs(fourier,pde),diff(Xi(t),t));
```

$$\frac{\partial}{\partial t}\,u(t,x) - d\left(\frac{\partial^2}{\partial x^2}\,u(t,x)\right)$$

$$fourier := u(t,x) = \quad (t)\,e^{I\,k\,x}$$

$$\frac{d}{dt}\quad (t) = -d\quad (t)\,k^2 \qquad \textbf{(7.5}$$

The ODE in the last line is easily solved, which gives us a particular solution for $u(t,x)$. Of course, the general solutions is a superposition of all of these "mode" functions labelled by the parameter k. We are going to attempt a similar type of decomposition of the errors:

```
> ansatz1 := E[i,j] = xi^i*exp(I*k*x[j]);
```

$$ansatz1 := E_{i,j} = \xi^i\,e^{I\,k\,x_j} \qquad \textbf{(7.6}$$

Here, ξ is a (possibly complex) constant that we call the *amplification factor*. This is what we are trying to solve for. Notice that the "time part" of the separation of variables isn't a general function of t_i; this is motivated from the exact solution of **(7.5)** evaluated at $t_i = is$:

```
> eval(dsolve((7.5)),t=is);
```

$$(is) = \_C1\,e^{-d\,k^2\,is} \qquad \textbf{(7.7}$$

This is essentially $(t = is) = (\text{constant})^{i} = \xi^{i}$, hence the form of $E_{i,j}$. We will need this ansatz evaluated at several different nodes before subbing into **(7.4)**:

```
> ansatz2 := [seq(seq(eval(ansatz1,[i=i+ii,j=j+jj]),ii=0..1),jj=-1..1)]:
  ansatz3 := eval(ansatz2,[seq(x[j+jj]=x[j]+jj*h,jj=-1..1)]);
```

$$ansatz3 := \left[E_{i,j-1} = \xi^{i} e^{I k \left(x_j - h\right)}, E_{i+1,j-1} = \xi^{i+1} e^{I k \left(x_j - h\right)}, E_{i,j} = \xi^{i} e^{I k x_j}, E_{i+1,j} = \xi^{i+1} e^{I k x_j}, E_{i,j+1} = \xi^{i} e^{I k \left(x_j + h\right)}, E_{i+1,j+1} = \xi^{i+1} e^{I k \left(x_j + h\right)}\right] \quad \textbf{(7.8}$$

We go ahead and sub this ansatz into the equations of motion for each stencil and solve for the amplification factor (neglecting the one-step error). Note that **(7.4)** always has the trivial solution $E_{i,j} = 0 \Rightarrow \xi = 0$ if the one-step error is neglected, so we need to subtract `{0}` from the output of the `solve` command:

```
> for ii from 1 to 3 do:
    xi_sol[ii] := factor(expand({solve(subs(ansatz3,one_step_error=0,EOM[ii]),xi)} minus
  {0})):
  od;
```

$$xi\_sol_1 := \left\{\frac{h^2 e^{I k h} + s\,d - 2\,s\,d\,e^{I k h} + s\,d\left(e^{I k h}\right)^2}{h^2 e^{I k h}}\right\}$$

$$xi\_sol_2 := \left\{-\frac{h^2 e^{I k h}}{-h^2 e^{I k h} + s\,d - 2\,s\,d\,e^{I k h} + s\,d\left(e^{I k h}\right)^2}\right\}$$

$$xi\_sol_3 := \left\{-\frac{2\,h^2 e^{I k h} + s\,d - 2\,s\,d\,e^{I k h} + s\,d\left(e^{I k h}\right)^2}{-2\,h^2 e^{I k h} + s\,d - 2\,s\,d\,e^{I k h} + s\,d\left(e^{I k h}\right)^2}\right\} \quad \textbf{(7.9}$$

Putting each of these solutions for the amplification factor into the ansatz **(7.6)** will give us a valid solution for $E_{i,j}$ for the FTCS, BTCS and Crank-Nicholson stencils, respectively. Of course, the total error will be a linear superposition of modes corresponding to all wavenumbers k. Now, if there exists some values of k for which $|\xi| > 1$, the form of **(7.6)** implies that the associated error will exponentially diverge as the simulation progresses. Since the actual error will generally be a superposition of errors for each k, the existence of modes with $|\xi| > 1$ implies that it will diverge as time progresses. That is, the stencil is unstable. The essence of the von Neumann stability analysis is that, given the ansatz **(7.6)**, determining whether or not there are any solutions for the amplification factor ξ with magnitude greater than unity. In such cases, the stencil is said to be unstable. To apply the analysis to the specific stencils we have here, we need to calculate the absolute values of ξ :

```
> for ii from 1 to 3 do:
```

```
   abs_xi[ii] := map(x->combine(expand(abs(x)),trig),xi_sol[ii]) assuming(h>0,s>0,d>0,k>0)
;
od;
```

$$abs\_xi_1 := \left\{ \frac{\sqrt{6\,s^2\,d^2 + h^4 - 8\,s^2\,d^2\cos(k\,h) + 2\,s^2\,d^2\cos(2\,k\,h) - 4\,h^2\,s\,d + 4\,h^2\cos(k\,h)\,s\,d}}{h^2} \right\}$$

$$abs\_xi_2 := \left\{ \frac{h^2}{\sqrt{4\,h^2\,s\,d - 4\,h^2\cos(k\,h)\,s\,d + h^4 - 8\,s^2\,d^2\cos(k\,h) + 2\,s^2\,d^2\cos(2\,k\,h) + 6\,s^2\,d^2}} \right\}$$

$$abs\_xi_3 := \left\{ \frac{\sqrt{-8\,h^2\,s\,d + 8\,h^2\cos(k\,h)\,s\,d - 8\,s^2\,d^2\cos(k\,h) + 2\,s^2\,d^2\cos(2\,k\,h) + 6\,s^2\,d^2 + 4\,h^4}}{\sqrt{8\,h^2\,s\,d - 8\,h^2\cos(k\,h)\,s\,d - 8\,s^2\,d^2\cos(k\,h) + 2\,s^2\,d^2\cos(2\,k\,h) + 6\,s^2\,d^2 + 4\,h^4}} \right\} \quad \textbf{(7.10}$$

These expressions are greatly simplified if we make some identifications:

```
> Subs := [theta=k*h,z=-4*s*d/h^2];
  for ii from 1 to 3 do:
     abs_xi[ii] := simplify(subs(solve(Subs,{s,k}),abs_xi[ii])) assuming(h>0,z<0,theta>0);
  od;
```

$$Subs := \left[\theta = k\,h,\ z = -\frac{4\,s\,d}{h^2} \right]$$

$$abs\_xi_1 := \left\{ \frac{1}{2}\,|\cos(\theta)\,z - z - 2| \right\}$$

$$abs\_xi_2 := \left\{ \frac{2}{\cos(\theta)\,z - z + 2} \right\}$$

$$abs\_xi_3 := \left\{ \frac{|\cos(\theta)\,z - z - 4|}{\cos(\theta)\,z - z + 4} \right\} \quad \textbf{(7.11}$$

A particular stencil will be unstable if there are any values of k such that $|\xi| > 1$. But since $\theta = kh$ and each of the above expressions for $|\xi|$ are periodic in θ with period $2\,\pi$, the condition for an unstable stencil is equivalent to saying that the maximum value of $|\xi|$ for $\theta \in [0, 2\,\pi]$ is greater than one. We can find these maximum values using the **maximize** command:

```
> for ii from 1 to 3 do:
     max_abs_xi[ii] := maximize(op(abs_xi[ii]),theta=0..2*Pi) assuming(z<0);
  od;
```

$$max\_abs\_xi_1 := \frac{1}{2}\max(2, |2\,z + 2|)$$

$$max\_abs\_xi_2 := 1$$

$$max\_abs\_xi_3 := 1 \qquad \textbf{(7.12}$$

The maximum value of $|\xi|$ is 1 for the BTCS and Crank-Nicholson stencils, so these stencils are unconditionally stable. The maximum value of $|\xi|$ for FTCS depends on the value of z, which means that that stencil is conditionally stable. Explicitly, the stability criterion $|\xi| \leq 1$ is

```
> FTCS_stability := max_abs_xi[1] <= 1;
```

$$FTCS\_stability := \frac{1}{2}\max(2, |2z+2|) \leq 1 \qquad \textbf{(7.13}$$

Maple can actually solve the inequality to indicate which values of z correspond to stability:

```
> FTCS_stability := z in solve(FTCS_stability);
```

$$FTCS\_stability := z \in RealRange(-2, 0) \qquad \textbf{(7.14}$$

Convert this into inequalities:

```
> FTCS_stability := convert(FTCS_stability,relation);
```

$$FTCS\_stability := \mathrm{And}(-2 \leq z, z \leq 0) \qquad \textbf{(7.15}$$

Express this in terms of the time- and spacesteps, as well as the diffusion constant:

```
> FTCS_stability := subs(Subs,FTCS_stability);
```

$$FTCS\_stability := \mathrm{And}\left(-2 \leq -\frac{4sd}{h^2}, -\frac{4sd}{h^2} \leq 0\right) \qquad \textbf{(7.16}$$

Convert this into conditions on the timestep s:

```
> FTCS_stability := solve(FTCS_stability,s) assuming(h>0,d>0);
```

$$FTCS\_stability := \left\{0 \leq s, s \leq \frac{1}{2}\frac{h^2}{d}\right\} \qquad \textbf{(7.17}$$

This is the explicit criterion for the stability of the FTCS stencil. Here is a procedure that given a choice of parameters tells us if the stencil i stable:

```
> IsStable := proc(tau,N,M,d) local X, T, s, h:
          X := j -> -1 + j*2/(M+1);
          T := i -> tau*i/N;
          s := evalf(T(1)-T(0));
          h := evalf(X(1)-X(0));
          if (2*s*d/h^2 <= 1) then:
             "this simulation is stable":
          else:
```

```
            "this simulation will be unstable":
        fi:
end proc:
```

The following example illustrate that we've got the stability criterion right:

```
> f := x-> exp(-(4*(x-1/2))^2):
  N := 30;
  M := 4;
  d := 1;
  tau := 2;
  IsStable(tau,N,M,d);
  FTCS(tau,N,M,d,f);
```

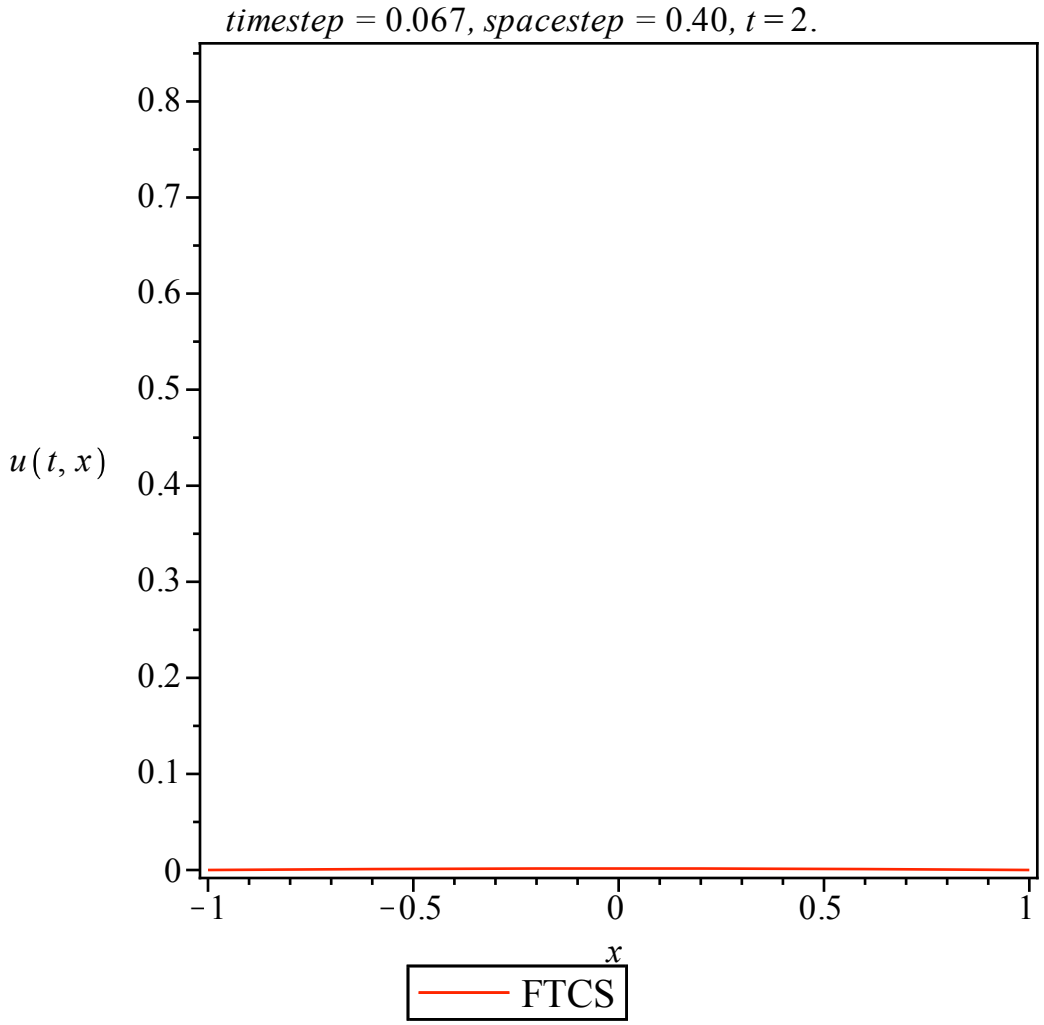
$$N := 30$$

$$M := 4$$

$$d := 1$$

$$\tau := 2$$

"this simulation is stable"




```
> f := x-> exp(-(4*(x-1/2))^2):
  N := 20;
  M := 4;
  d := 1;
  tau := 2;
  IsStable(tau,N,M,d);
  FTCS(tau,N,M,d,f);
```

$N := 20$

$M := 4$

$d := 1$

$\tau := 2$

"this simulation will be unstable"