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

The Crank-Nicholson method for a nonlinear diffusion equation

The purpoe of this worksheet is to solve a diffuion equation involving nonlinearities numerically using the Crank-Nicholson stencil. We start with the following PDE, where the potential function is meant to be a nonlinear function of the unknown u(t,x):

```
> potential := u -> u*(u-1)-1;
  pde := diff(u(t,x),t) = diff(u(t,x),x,x)+potential(u(t,x));
```

$$potential := u \to u\,(u-1)-1$$

$$pde := \frac{\partial}{\partial t}\,u(t,x) = \frac{\partial^2}{\partial x^2}\,u(t,x) + u(t,x)\,(u(t,x)-1)-1 \qquad \textbf{(1)}$$

To construct the Crank-Nicholson stencil for this PDE, we will make use of the following procedures, as before:

```
> centered_stencil := proc(r,N,{direction := spatial})
      local n, stencil, vars, beta_sol:
      n := floor(N/2):
      if (direction = spatial) then:
         stencil := D[2$r](u)(t,x) - add(beta[i]*u(t,x+i*h),i=-n..
  n);
         vars := [u(t,x),seq(D[2$i](u)(t,x),i=1..N-1)];
      else:
         stencil := D[1$r](u)(t,x) - add(beta[i]*u(t+i*h,x),i=-n..
  n);
         vars := [u(t,x),seq(D[1$i](u)(t,x),i=1..N-1)];
      fi:
      beta_sol := solve([coeffs(collect(convert(series(stencil,h,
  N),polynom),vars,'distributed'),vars)]):
      stencil := subs(beta_sol,stencil);
      if (direction = spatial) then:
         convert(stencil = convert(series(stencil,h,N+2),polynom),
  diff);
      else:
         subs(h=s,convert(stencil = convert(series(stencil,h,N+2),
  polynom),diff));
      fi:
  end proc:

  onesided_stencil := proc(r,N,{direction := spatial})
      local stencil, vars, beta_sol:
      if (direction = spatial) then:
         stencil := D[2$r](u)(t,x) - add(beta[i]*u(t,x+i*h),i=0..
  N-1);
         vars := [u(t,x),seq(D[2$i](u)(t,x),i=1..N-1)];
      else:
         stencil := D[1$r](u)(t,x) - add(beta[i]*u(t+i*h,x),i=0..
  N-1);
         vars := [u(t,x),seq(D[1$i](u)(t,x),i=1..N-1)];
      fi:
      beta_sol := solve([coeffs(collect(convert(series(stencil,h,
  N),polynom),vars,'distributed'),vars)]):
      stencil := subs(beta_sol,stencil);
      if (direction = spatial) then:
         convert(stencil = convert(series(`leadterm`(stencil),h,
```

```
N+1),polynom),diff);
    else:
       subs(h=s,convert(stencil = convert(series(`leadterm`
(stencil),h,N+1),polynom),diff));
    fi:
end proc:
```

Our basic stencils for the derivatives are as follows:

```
> center_space := isolate(lhs(centered_stencil(2,3)),diff(u(t,x),x,
x));
forward_time := isolate(lhs(onesided_stencil(1,2,direction=
temporal)),diff(u(t,x),t));
backward_time := subs(s=-s,forward_time);
```

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

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

$$backward\_time := \frac{\partial}{\partial t} u(t,x) = \frac{u(t,x)}{s} - \frac{u(t-s,x)}{s} \qquad (2)$$

Subbing these objects into the PDE in various combinations gives us the basic three stencils:

```
> FTCS := subs(center_space,forward_time,pde);
BTCS := subs(center_space,backward_time,t=t+s,pde);
CN := (FTCS+BTCS)/2;
```

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

$$x)\,(u(t,x)-1)-1$$

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

$$+ u(t+s,x)\,(u(t+s,x)-1)-1$$

$$CN := -\frac{u(t,x)}{s} + \frac{u(t+s,x)}{s} = \frac{1}{2}\,\frac{u(t,x-h)}{h^2} - \frac{u(t,x)}{h^2} + \frac{1}{2}\,\frac{u(t,x+h)}{h^2} + \frac{1}{2}\,u(t, \qquad (3)$$

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

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

As before, only the FTCS stencil explicitly gives us the future value of u based on past values. We will be working with the CN stencil, which was shown to be unconditionally stable for the linear problem and is second order in both space and time. We relabel the past values of u as the elements of an array V and the future values as the element of an array U:

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

$$Subs := \left[u(t+s,x+h) = U_{j-1}, u(t+s,x) = U_j, u(t+s,x-h) = U_{j+1}, u(t,x+h)\right. \qquad (4)$$

$$\left. = V_{j-1}, u(t,x) = V_j, u(t,x-h) = V_{j+1}\right]$$

Putting these definitions into the stencil, we get

```
> CN := (lhs-rhs)(subs(Subs,CN));
```

$$CN := -\frac{V_j}{s} + \frac{U_j}{s} - \frac{1}{2}\frac{V_{j+1}}{h^2} + \frac{V_j}{h^2} - \frac{1}{2}\frac{V_{j-1}}{h^2} - \frac{1}{2}V_j\left(V_j - 1\right) + 1 - \frac{1}{2}\frac{U_{j+1}}{h^2} + \frac{U_j}{h^2} - \frac{1}{2}\frac{U_{j-1}}{h^2} - \frac{1}{2}U_j\left(U_j - 1\right) \quad (5)$$

When we use this in an actual algorithn, the V's will be known and the U's will be unknown. Observe that since the potential was nonlinear, the U's appear nonlinearly in this equation. Previously, the U's all appeared linearly, and the above was just one component of the matrix equation A.U=B. We can convert CN into a linear equation by defining:

```
> Linearization := seq(U[j+i] = U0[j+i] + epsilon*dU[j+i],i=-1..1);
```

$$Linearization := U_{j-1} = U0_{j-1} + \varepsilon\, dU_{j-1},\ U_j = U0_j + \varepsilon\, dU_j,\ U_{j+1} = U0_{j+1} + \varepsilon\, dU_{j+1} \quad (6)$$

In this formula, U0[j] is a ***guess*** for the value of U[j] and epsilon*dU[j] is the error in that guess. We'll substitute this into CN, and then expand the resulting expression to linear order in epsilon (after the series has been calculated, we set epsilon = 1):

```
> CN_linear := subs(Linearization,CN):
  CN_linear := series(CN_linear,epsilon,2):
  CN_linear := convert(CN_linear,polynom):
  CN_linear := simplify(subs(epsilon=1,CN_linear)):
  CN_linear := (lhs-rhs)(expand(isolate(CN_linear,dU[j]))):
  CN_linear := (CN_linear=0)-subs(seq(dU[j+i]=0,i=-1..1),CN_linear)
  ;
```

$$CN\_linear := dU_j + \frac{dU_{j+1}\, s}{2\, U0_j\, s\, h^2 - 2\, s - 2\, h^2 - s\, h^2} + \frac{dU_{j-1}\, s}{2\, U0_j\, s\, h^2 - 2\, s - 2\, h^2 - s\, h^2} = \quad (7)$$

$$-\frac{2\, V_j\, h^2}{2\, U0_j\, s\, h^2 - 2\, s - 2\, h^2 - s\, h^2} + \frac{2\, U0_j\, h^2}{2\, U0_j\, s\, h^2 - 2\, s - 2\, h^2 - s\, h^2}$$

$$-\frac{V_j^2\, s\, h^2}{2\, U0_j\, s\, h^2 - 2\, s - 2\, h^2 - s\, h^2} + \frac{V_j\, s\, h^2}{2\, U0_j\, s\, h^2 - 2\, s - 2\, h^2 - s\, h^2}$$

$$-\frac{V_{j+1}\, s}{2\, U0_j\, s\, h^2 - 2\, s - 2\, h^2 - s\, h^2} + \frac{2\, V_j\, s}{2\, U0_j\, s\, h^2 - 2\, s - 2\, h^2 - s\, h^2}$$

$$-\frac{V_{j-1}\, s}{2\, U0_j\, s\, h^2 - 2\, s - 2\, h^2 - s\, h^2} + \frac{2\, U0_j\, s}{2\, U0_j\, s\, h^2 - 2\, s - 2\, h^2 - s\, h^2}$$

$$+\frac{2\, s\, h^2}{2\, U0_j\, s\, h^2 - 2\, s - 2\, h^2 - s\, h^2} - \frac{U0_{j+1}\, s}{2\, U0_j\, s\, h^2 - 2\, s - 2\, h^2 - s\, h^2}$$

$$-\frac{U0_j^2\, s\, h^2}{2\, U0_j\, s\, h^2 - 2\, s - 2\, h^2 - s\, h^2} + \frac{U0_j\, s\, h^2}{2\, U0_j\, s\, h^2 - 2\, s - 2\, h^2 - s\, h^2}$$

$$-\frac{U0_{j-1}\, s}{2\, U0_j\, s\, h^2 - 2\, s - 2\, h^2 - s\, h^2}$$

On the LHS of CN linear are the new unknowns dU[j] and on the right are the knowns V[j] and U0[j]. So, given a guess U0 for the solution U of the CN stencil, this gives us a linear system for the error in the guess dU. After the system is solved for dU, we obtain a better guess for U from U0 + dU. We can then iterate the procedure a number of times to futher refine our solution. This is Newton's method for solving a nonlinear system of M equations with M unknowns. We can illustrate stucture of the linear system for dU by defining two functions q and b:

```
> q_def := q(U0[j],s,h) = coeff(lhs(CN_linear),dU[j+1]);
  CN_linear := simplify(algsubs(isolate(q_def,h^2),CN_linear)):
  b_def := b(U0,V,j,s,h) = rhs(CN_linear);
```

$$q\_def := q(U0_j, s, h) = \frac{s}{2\, U0_j\, s\, h^2 - 2\, s - 2\, h^2 - s\, h^2}$$

$$b\_def := b(U0, V, j, s, h) = \frac{1}{2\, U0_j\, s - 2 - s}\left(V_j\, s + U0_j\, s + 2\, s - 2\, V_j + 2\, U0_j + 4\, s\, q(U0_j, s, h) - V_j^2\, s + 2\, V_{j+1}\, q(U0_j, s, h) - 8\, V_j\, q(U0_j, s, h) + 2\, V_{j-1}\, q(U0_j, s, h) - 2\, V_{j+1}\, q(U0_j, s, h)\, U0_j\, s + 4\, V_j\, q(U0_j, s, h)\, U0_j\, s - 2\, V_{j-1}\, q(U0_j, s, h)\, U0_j\, s - 2\, U0_{j+1}\, q(U0_j, s, h)\, U0_j\, s - 2\, U0_{j-1}\, q(U0_j, s, h)\, U0_j\, s + 2\, U0_{j+1}\, q(U0_j, s, h) - U0_j^2\, s + 2\, U0_{j-1}\, q(U0_j, s, h) - 2\, V_j^2\, s\, q(U0_j, s, h) + V_{j+1}\, q(U0_j, s, h)\, s + V_{j-1}\, q(U0_j, s, h)\, s + 2\, U0_j^2\, q(U0_j, s, h)\, s + U0_{j+1}\, q(U0_j, s, h)\, s + U0_{j-1}\, q(U0_j, s, h)\, s\right) \quad \textbf{(8)}$$

In tern if these two functions, the CN linear stencil looks a little simpler:

```
> CN_linear := lhs(CN_linear) = lhs(b_def);
```

$$CN\_linear := dU_{j+1}\, q(U0_j, s, h) + dU_{j-1}\, q(U0_j, s, h) + dU_j = b(U0, V, j, s, h) \quad \textbf{(9)}$$

Here is what the linear system looks like when there are M = 4 interior nodes:

```
> U0 := 'U0':
  V := 'V':
  dU := 'dU':
  eq := 'eq':
  M := 4:
  V[0] := 0:
  V[M+1] := 0:
  U0[0] := 0:
  U0[M+1] := 0:
  dU[0] := 0:
  dU[M+1] := 0:
  for i from 1 to M do:
     eq[i] := eval(subs(j=i,CN_linear)):
  od;
```

$$eq_1 := dU_2\, q(U0_1, s, h) + dU_1 = b(U0, V, 1, s, h)$$

$$eq_2 := dU_3\, q(U0_2, s, h) + dU_1\, q(U0_2, s, h) + dU_2 = b(U0, V, 2, s, h)$$

$$eq_3 := dU_4\, q(U0_3, s, h) + dU_2\, q(U0_3, s, h) + dU_3 = b(U0, V, 3, s, h)$$

$$eq_4 := dU_3\, q(U0_4, s, h) + dU_4 = b(U0, V, 4, s, h) \quad \textbf{(10)}$$

These M equations form a linear system of M unknowns dU[j]. Viewing this as a matrix equation A.dU = B, we can determine A and B using the GenerateMatrix command from the LinearAlgebra package:

```
> A,B := GenerateMatrix(convert(eq,list),[seq(dU[j],j=1..M)]):
  A;
  B;
```

$$\begin{bmatrix} 1 & q(U0_1, s, h) & 0 & 0 \\ q(U0_2, s, h) & 1 & q(U0_2, s, h) & 0 \\ 0 & q(U0_3, s, h) & 1 & q(U0_3, s, h) \\ 0 & 0 & q(U0_4, s, h) & 1 \end{bmatrix}$$

$$\begin{bmatrix} b(U0, V, 1, s, h) \\ b(U0, V, 2, s, h) \\ b(U0, V, 3, s, h) \\ b(U0, V, 4, s, h) \end{bmatrix} \quad \textbf{(11)}$$

We see that in order to obtain dU, we need to solve a tridiagonal matrix equation. The following procedure takes the array of past values V, a guess for the future values U0, and the time s and spacesteps h and returns the matrix A and

```
> AA := proc(U0,V,s,h)
        local qU0, M , A, B:
        qU0 := map(x->q(x,s,h),U0):
        M := rhs(ArrayDims(U0))-1;
        A := BandMatrix([convert(qU0[2..M],list),[1$M],convert(qU0
  [1..M-1],list)],1,M);
        B := Vector(1..M,[seq(b(U0,V,i,s,h),i=1..M)]):
        A,B:
  end proc:
```

Notice that for this procedure to actually return A and B as floating poing objects, q and b need to be defined as procedures.

```
> q := unapply(rhs(q_def),U0[j],s,h):
  b := unapply(rhs(b_def),U0,V,j,s,h):
```

Here is the procedure that actually implements Newton's method. Taking the past field values V, s, h and the number of spatial nodes, it implements Newton's method to solve the nonlinear system for the future field values U. It decides to stop the iterations when the average magnitude of the elements of the dU vector are less than a small parameter eps. Note that we invoke the LinearSolve command with the SparseIterative method to speed things up a little.

```
> Newton := proc(V,s,h,M) local U, i, dU, CONTINUE, eps, err:
       U := Array(0..M+1,datatype=float):
       U[1..M] := V[1..M]:                              # our initial
  guess is U = V
       CONTINUE := true:
       eps := 1e-3:                                     # the
  parameter that fixes when the Newton iterations stop
```

```
      for i from 1 to 50 while (CONTINUE) do:        # we're going
to do a maximum of 50 iterations
          dU := LinearSolve(AA(U,V,s,h),
                method='SparseIterative'):           # obtain the
dU from the guess U by solving the linear system generated by AA
          err := sqrt((dU.dU)/M):                    # calculate
the root-mean-square value of the components of dU
          if (err < eps) then CONTINUE := false fi: # decide if
err is small enough to stop iterating
          U[1..M] := U[1..M] + dU:                   # update the
guess by adding dU
     od:
     U:
end proc:
```

Here is the main calculation:

```
> CN_evolve := proc(f,X,T,s,h) local N, M, x, XX, V, p, A, i, B, U,
q:
       N := round(T/s):
  # the number of timesteps
       M := round(X/h) - 1:
  # the number of spacesteps
       x := j -> -X/2 + j*h:
       XX := Array(0..M+1,[seq(x(j),j=0..M+1)],datatype=float):
  # define Array of x values for plotting
       V := Array(0..M+1,[seq(f(x(j)),j=0..M+1)],datatype=float):
  # initial data
       p[0] := plot(ListTools[Transpose](convert
               (Concatenate(1,XX,V),listlist)),axes=boxed,style=
line):          # first frame of the movie
       for i from 1 to N do:
  # loop over timesteps
          V[1..M] := Newton(V,s,h,M)[1..M]:
  # Newton iteration to evolve field values
          p[i] := plot(ListTools[Transpose](convert
                  (Concatenate(1,XX,V),listlist)),axes=boxed,
style=line):      # ith frame of the movie
       od:
       display(convert(p,list),insequence=true):
  # prepare movie output
end proc:
```

Choose initial data and parameters to test the code:

```
> f_ := x -> 1/cosh(10*x);
  X_ := 2;
  T_ := 1;
  s_ := 0.01;
  h_ := 0.25;
```

$$f\_ := x \to \frac{1}{\cosh(10\,x)}$$

$$X\_ := 2$$

$$T\_ := 1$$

$$s\_ := 0.01$$

$$h\_ := 0.25 \qquad \textbf{(12)}$$

movie1 is the output of our code, while movie2 is the output of pdsolve/numeric for the same problem:

```
> movie1 := CN_evolve(f_,X_,T_,s_,h_):
  pds := pdsolve(pde,[u(0,x)=f_(x),u(t,-X_/2)=0,u(t,X_/2)=0],
```

```
numeric,timestep=s_,spacestep=h_):
movie2 := pds:-animate(t=0..T_,axes=boxed,frames=round(T_/s_)+1,
style=point,color=blue):
```

We display the movies simultaneously to compare results. The pdsolve answer is in points while our code is the line.

```
> display([movie1,movie2]);
```