```
> restart;
  with(PDEtools):
  with(plots):
  with(LinearAlgebra):
  Digits := 14:
```

Nonlinear PDEs

The purpose of this worksheet is to discuss the solution of nonlinear PDEs using finite difference methods. For concreteness, we will concentrate on a diffusion equation with a potential:

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

Here, d is the diffusion constant and V is a given (presumably nonlinear) function of $u(t,x)$. We take t as a time variable, hence this is an initia value problem. We'll take Dirichlet boundary conditions

$$u(t,-L) = u(t,L) = 0,$$

where L is a constant to be specified.

Stencils

Here is our PDE:

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

$$pde := \frac{\partial}{\partial t}u(t,x) = d\left(\frac{\partial^2}{\partial x^2}u(t,x)\right) + V(u(t,x)) \qquad (1.1$$

We will make use of the `GenerateStencil` procedure defined elsewhere:

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

```
backward_time := GenerateStencil(diff(u(t,x),t),2,orientation=left,stepsize=s);
centered_time := GenerateStencil(diff(u(t,x),x,x),3,orientation=center,stepsize=h);
```

This stencil is of order 1

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

This stencil is of order 1

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

This stencil is of order 2

$$centered\_time := \frac{\partial^2}{\partial x^2} u(t,x) = \frac{u(t,x-h)}{h^2} - \frac{2\,u(t,x)}{h^2} + \frac{u(t,x+h)}{h^2} \quad \textbf{(1.2}$$

```
> Label[1] := `FTCS`:
  stencil[1] := subs(forward_time,centered_time,pde);
```

$$stencil_1 := -\frac{u(t,x)}{s} + \frac{u(t+s,x)}{s} = d\left(\frac{u(t,x-h)}{h^2} - \frac{2\,u(t,x)}{h^2} + \frac{u(t,x+h)}{h^2}\right) + V(u(t,x)) \quad \textbf{(1.3}$$

```
> BTCS := subs(backward_time,centered_time,t=t+s,pde);
```

$$BTCS := -\frac{u(t,x)}{s} + \frac{u(t+s,x)}{s} = d\left(\frac{u(t+s,x-h)}{h^2} - \frac{2\,u(t+s,x)}{h^2} + \frac{u(t+s,x+h)}{h^2}\right) + V(u(t+s,x)) \quad \textbf{(1.4}$$

```
> Label[2] := `CN`:
  stencil[2] := (stencil[1]+BTCS)/2;
```

$$stencil_2 := -\frac{u(t,x)}{s} + \frac{u(t+s,x)}{s} = \frac{1}{2}\,d\left(\frac{u(t,x-h)}{h^2} - \frac{2\,u(t,x)}{h^2} + \frac{u(t,x+h)}{h^2}\right) + \frac{1}{2}\,V(u(t,x)) + \frac{1}{2}\,d\left(\frac{u(t+s,x-h)}{h^2} - \frac{2\,u(t+s,x)}{h^2} + \frac{u(t+s,x+h)}{h^2}\right) + \frac{1}{2}\,V(u(t+s,x)) \quad \textbf{(1.5}$$

Forward-time centered-space solution

```
> FTCS_stencil := expand(isolate(stencil[1],u(t+s,x)));
```

$$FTCS\_stencil := u(t+s,x) = \frac{s\,d\,u(t,x-h)}{h^2} - \frac{2\,s\,d\,u(t,x)}{h^2} + \frac{s\,d\,u(t,x+h)}{h^2} + s\,V(u(t,x)) + u(t,x) \quad \textbf{(2.1}$$

```
> Subs := u(t,x-h)=u1,u(t,x)=u2,u(t,x+h)=u3;
```

```
Phi := unapply(subs(Subs,rhs(FTCS_stencil)),u1,u2,u3,h,s,d);
```

$$Subs := u(t, x - h) = u1, u(t, x) = u2, u(t, x + h) = u3$$

$$\Phi := (u1, u2, u3, h, s, d) \rightarrow \frac{s\, d\, u1}{h^2} - \frac{2\, s\, d\, u2}{h^2} + \frac{s\, d\, u3}{h^2} + s\, V(u2) + u2 \qquad (2.2$$

```
> FTCS := proc(tau,L,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 -> L*(-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:

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=["numerical solution (FTCS)
"], color=red;
        plot(Matrix([[xdata],[ydata]])^%T,title=Title,PlotOptions);

end proc:
> V := u -> 1-u^2;
tau := 1;
N := 400;
```

```
M := 20;
L := 1;
d := 1;
s := evalf(2*L/(M+1));
h := evalf(tau/N);
f := x -> exp(-(4*x)^2);
movie[1] := FTCS(tau,L,N,M,d,f):
pds := pdsolve(pde,[u(t,-L)=0,u(t,+L)=0,u(0,x)=f(x)],numeric,timestep=s,spacestep=h):
movie[2] := pds:-animate(t=0..tau,frames=N+1,axes=boxed,legend=["pdsolve/numeric"],color=
blue):
```

$$V := u \rightarrow 1 - u^2$$

$$\tau := 1$$

$$N := 400$$

$$M := 20$$

$$L := 1$$

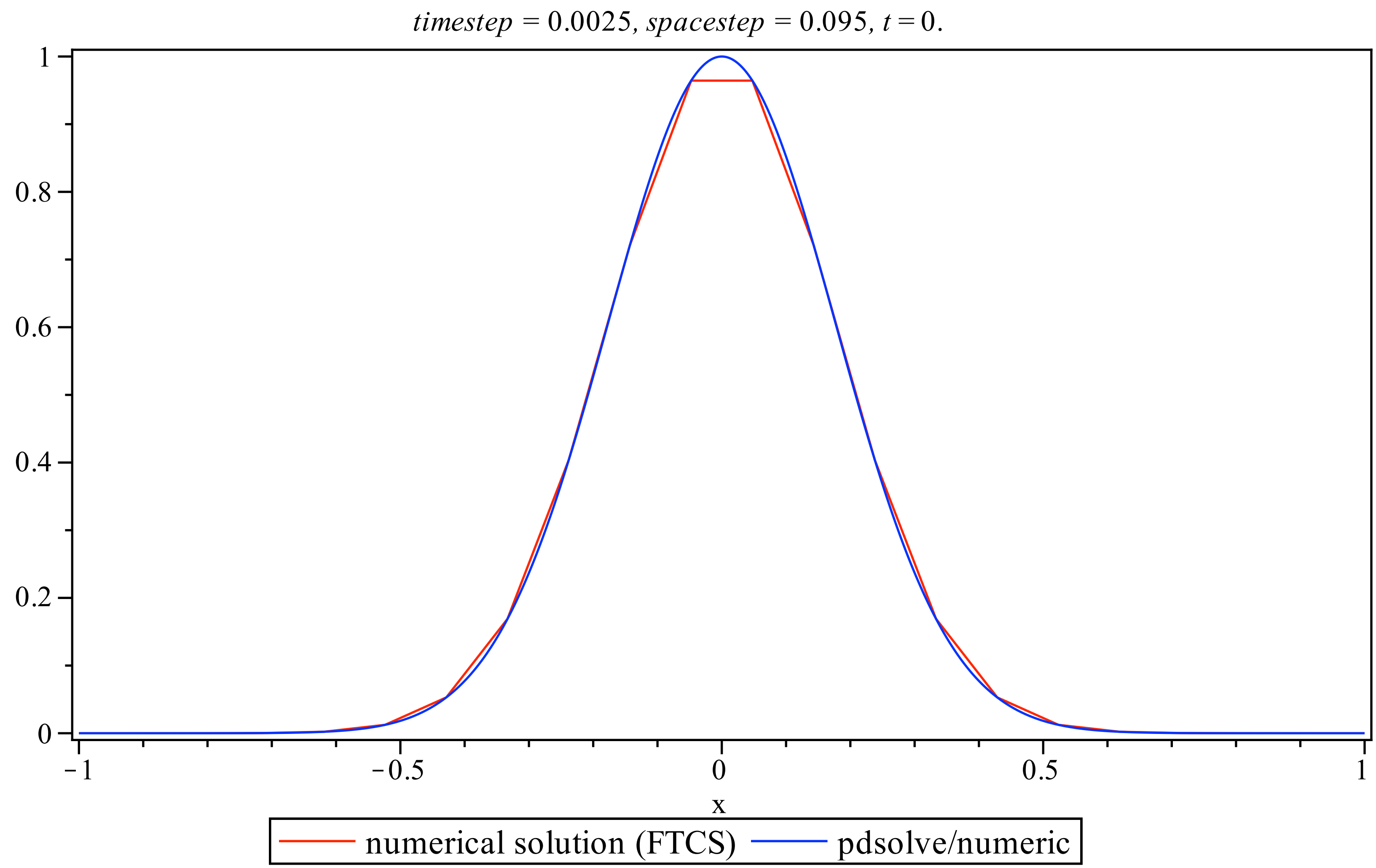
$$d := 1$$

$$s := 0.095238095238095$$

$$h := 0.0025000000000000$$

$$f := x \rightarrow e^{-16 x^2} \quad \textbf{(2.3}$$

```
> display([movie[1],movie[2]]);
```



Review of Newton's method to solve nonlinear algebraic equations

Solving one equation for one unknown

```
> eq1 := F(x) = 0;
```

$$eq1 := F(x) = 0 \quad (3.1.1)$$

```
> eq2 := x = x[guess] + delta;
```

$$eq2 := x = x_{guess} + \delta \qquad \textbf{(3.1.2)}$$

```
> eq3 := subs(eq2,eq1);
```

$$eq3 := F\left(x_{guess} + \delta\right) = 0 \qquad \textbf{(3.1.3)}$$

```
> eq4 := series(lhs(eq3),delta,2)=0;
```

$$eq4 := F\left(x_{guess}\right) + \mathrm{D}(F)\left(x_{guess}\right)\delta + \mathrm{O}\left(\delta^2\right) = 0 \qquad \textbf{(3.1.4)}$$

```
> eq5 := isolate(convert(eq4,polynom),delta);
```

$$eq5 := \delta = -\frac{F\left(x_{guess}\right)}{\mathrm{D}(F)\left(x_{guess}\right)} \qquad \textbf{(3.1.5)}$$

```
> eq6 := x[`new guess`] = x[guess] + delta;
```

$$eq6 := x_{new\ guess} = x_{guess} + \delta \qquad \textbf{(3.1.6)}$$

```
> eq7 := subs(eq5,eq6);
```

$$eq7 := x_{new\ guess} = x_{guess} - \frac{F\left(x_{guess}\right)}{\mathrm{D}(F)\left(x_{guess}\right)} \qquad \textbf{(3.1.7)}$$

```
> x_new := unapply(rhs(eq7),x[guess]);
```

$$x\_new := y1 \rightarrow y1 - \frac{F(y1)}{\mathrm{D}(F)(y1)} \qquad \textbf{(3.1.8)}$$

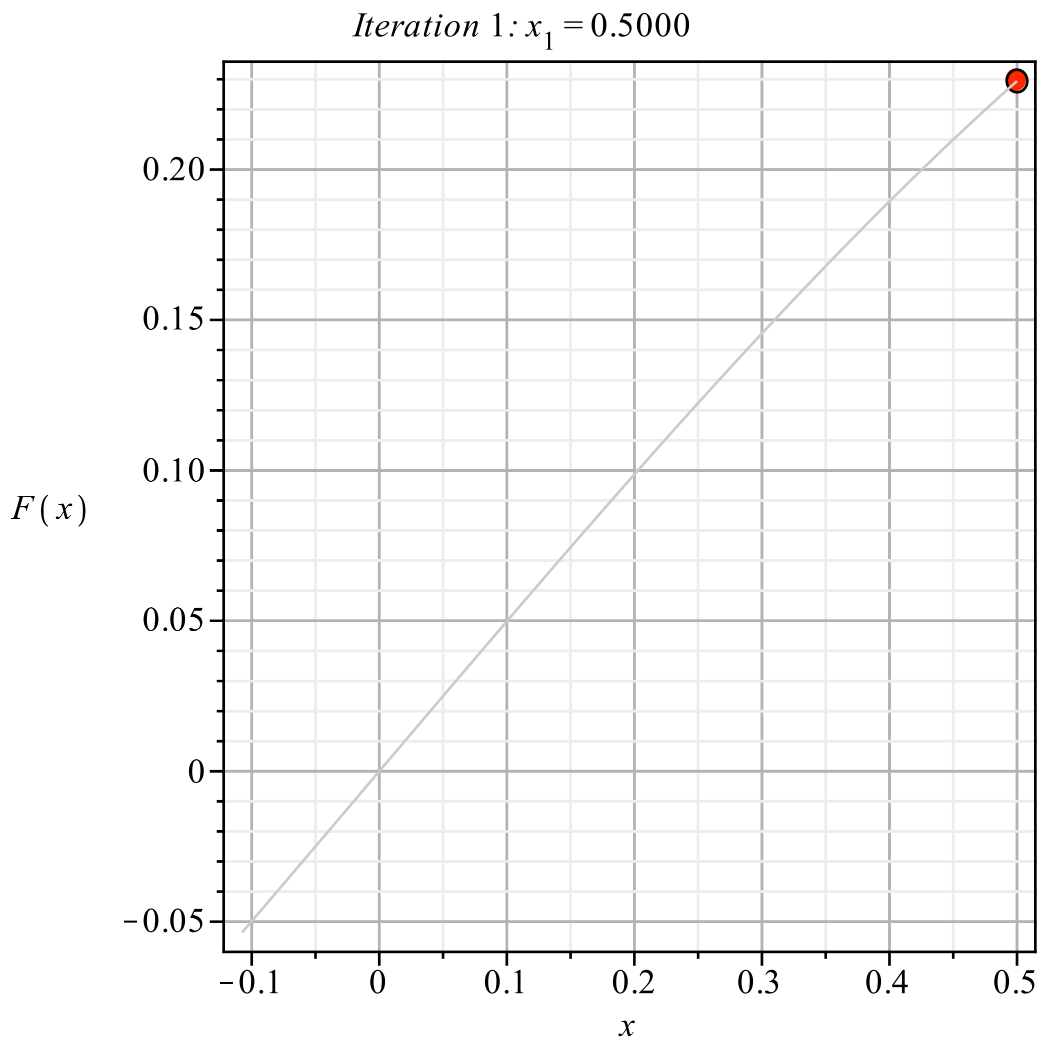
```
> Newton[scalar] := proc(F,x0,eps)
     local maxsteps, CONTINUE, x, i:
     maxsteps := 20:
     CONTINUE := true:
     x[0] := evalf(x0):
     for i from 1 to maxsteps while (CONTINUE) do:
        x[i] := evalf(x_new(x[i-1])):
        if (abs(x[i]-x[i-1])<eps) then CONTINUE := false fi:
     od;
     Matrix([seq([x[j],F(x[j])],j=0..i-1)],datatype=float);
  end proc:
> F := x -> sin(x)-x/2;
  Newton[scalar](F,4,1e-10);
```

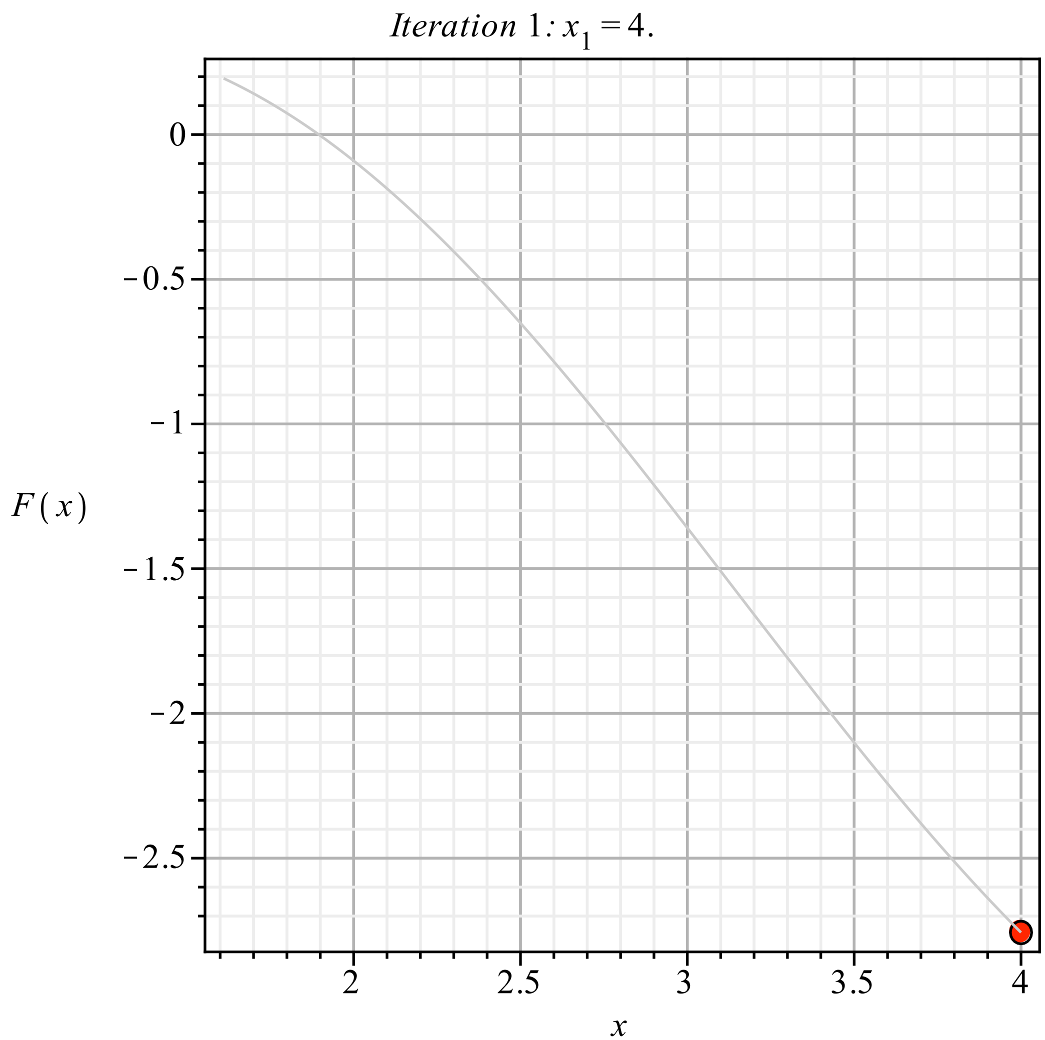
$$F := x \rightarrow \sin(x) - \frac{1}{2}x$$

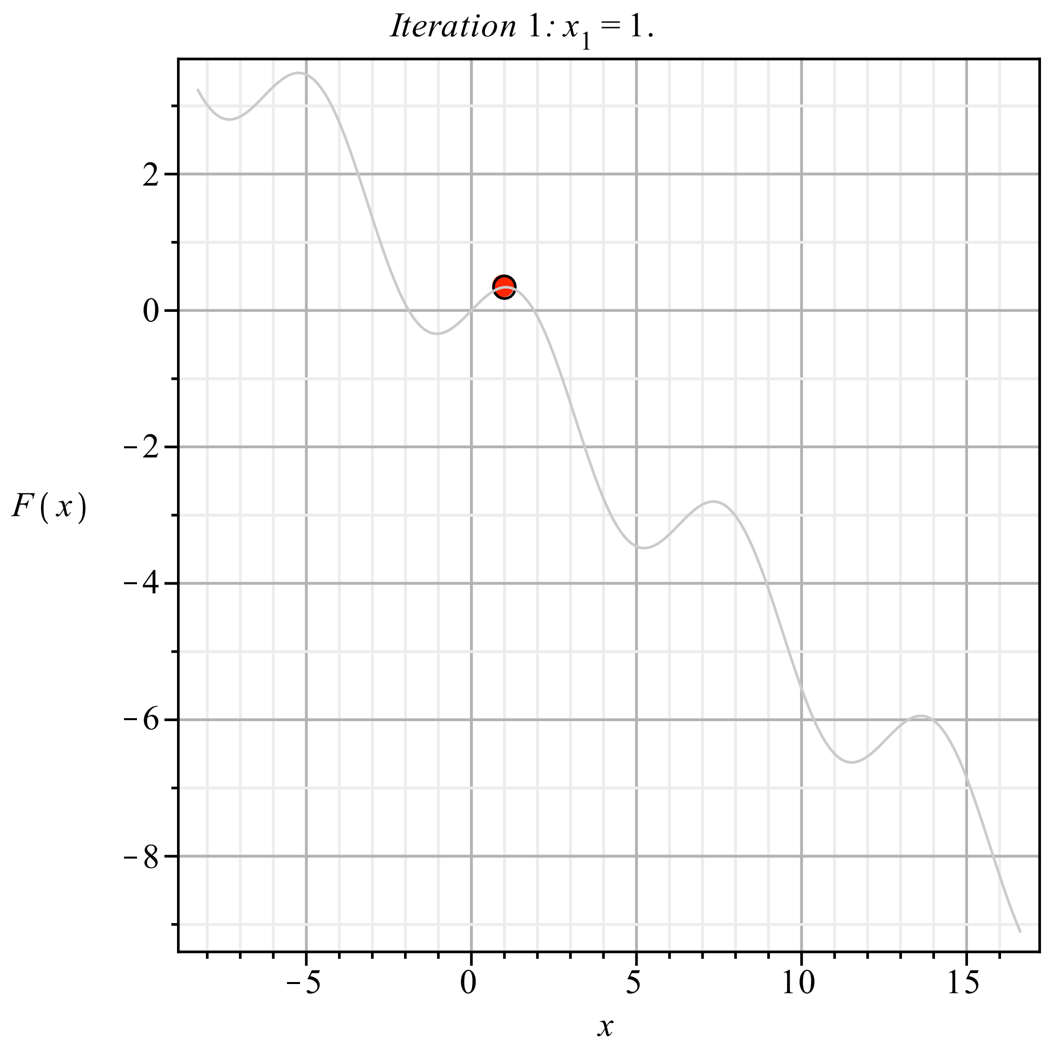
$$\begin{bmatrix} 4. & -2.75680249530790 \\ 1.61035171915210 & 0.194041927889040 \\ 1.96999160043930 & -0.0636217292795800 \\ 1.89840009052060 & -0.00238393487773000 \\ 1.89549913294010 & -0.00000398529793000000 \\ 1.89549426704770 & -1.12400000000000\ 10^{-11} \\ 1.89549426703400 & -2.00000000000000\ 10^{-14} \end{bmatrix} \quad (3.1.9$$

```
> Newton[movie] := proc(F,x0,eps)
    local data, i, r, ngon, q, p:
    data[0] := Newton[scalar](F,x0,eps):
    for i from 1 to 2 do:
       data[i] := Column(data[0],i):
       r[i] := (max(data[i])-min(data[i]))/75:
    od;
    ngon := (n,x,y,rx,ry,phi,Color) -> polygonplot([seq([x+rx*cos(2*Pi*i/n+phi),
            y+ry*sin(2*Pi*i/n+phi)], i = 1 .. n)],color=Color):
    q := plot(F(x),x=min(data[1])..max(data[1]),color=grey,axes=boxed,gridlines=true,
  labels=[x,''F(x)'']):
    for i from 1 to Dimension(data[0])[1] do:
         p[i] := display(ListTools[Reverse]([q,seq(ngon(20,data[0][j,1],data[0][j,2],r
  [1],r[2],0,yellow),j=1..i-1),
                 ngon(20,data[0][i,1],data[0][i,2],r[1],r[2],0,red)]),title=[typeset
  (`Iteration `,i,`: `,
                 x[i]=evalf[4](data[0][i,1]))]);
    od:
    display(convert(p,list),insequence=true);
  end proc:
> F := x -> sin(x)-x/2;
  Newton[movie](F,0.5,1e-6);
  Newton[movie](F,4.0,1e-6);
  Newton[movie](F,1.0,1e-6);
```

$$F := x \rightarrow \sin(x) - \frac{1}{2}x$$


Iteration 1: $x_1 = 0.5000$
F(x)
x
0.20
0.15
0.10
0.05
0
−0.05
−0.1
0
0.1
0.2
0.3
0.4
0.5


Iteration 1: $x_1 = 4.$
F (x)
x
0
-0.5
-1
-1.5
-2
-2.5
2
2.5
3
3.5
4



Solving *n* equations for *n* unknowns

```
> Newton[vector] := proc(Sys,Vars,guess,eps)
        local N, vars, i, sys, linearization, linear_sys,
              A, b, X, CONTINUE, maxsteps, delta, Subs, d:
        N := nops(Sys):
        vars := convert(Vars,list):
```

```
        for i from 1 to N do:
           if (type(Sys[i],equation)) then:
             sys[i] := (lhs-rhs)(Sys[i]):
           else:
             sys[i] := Sys[i]:
           fi;
        od;
        sys := convert(sys,list);
        linearization := map(u->u=u+epsilon*d[u],vars);
        linear_sys := map(u->subs(epsilon=1,convert(series(u,epsilon,2),polynom)),subs
   (linearization,sys));
        A,b := LinearAlgebra[GenerateMatrix](linear_sys,map(u->d[u],vars));
        Subs := seq(vars[i]=q[i],i=1..N);
        A := unapply(subs(Subs,A),q);
        b := unapply(subs(Subs,b),q);
        X[0] := evalf(Vector(guess)):
        maxsteps := 50:
        CONTINUE := true:

        for i from 1 to maxsteps while (CONTINUE) do:
           X[i] := LinearAlgebra[LinearSolve](evalf(A(X[i-1])),evalf(b(X[i-1])))+X[i-1]:
           delta := X[i]-X[i-1]:
           delta := sqrt(delta^%T.delta)/N;
           if (delta<eps) then CONTINUE := false fi:
        od;

        if (CONTINUE) then:
           return `maximum number of iterations exceeded`:
        else:
           return convert(X[i-1],list):
        fi:

   end proc:
> sys := [2*x+y^2-z-1,x-y^3+z^2,z^2-x^2];
   vars := [x,y,z];
   guess := [-4,2,-4];
   Newton[vector](sys,vars,guess,1e-14);
```

$$sys := [2\,x + y^2 - z - 1,\, x - y^3 + z^2,\, z^2 - x^2]$$

$$vars := [x, y, z]$$

$$guess := [-4, 2, -4]$$

$$[-3.76230632494549, 2.18227091007178, -3.76230632494549] \quad (3.2.1)$$

```
> fsolve(sys);
```

$$\{x = -3.7623063249455, y = 2.1822709100718, z = -3.7623063249455\} \quad (3.2.2)$$

Crank-Nicholson solution

```
> V := 'V':
  d := 'd':
  s := 's':
  h := 'h':
  j := 'j':
> expand(stencil[2]/2/h^2);
```

$$-\frac{1}{2}\frac{u(t,x)}{h^2 s}+\frac{1}{2}\frac{u(t+s,x)}{h^2 s}=\frac{1}{4}\frac{d\,u(t,x-h)}{h^4}-\frac{1}{2}\frac{d\,u(t,x)}{h^4}+\frac{1}{4}\frac{d\,u(t,x+h)}{h^4}+\frac{1}{4}\frac{V(u(t,x))}{h^2}+\frac{1}{4}\frac{d\,u(t+s,x-h)}{h^4}-\frac{1}{2}\frac{d\,u(t+s,x)}{h^4}+\frac{1}{4}\frac{d\,u(t+s,x+h)}{h^4}+\frac{1}{4}\frac{V(u(t+s,x))}{h^2} \quad (4.1)$$

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

$$Subs := u(t+s,x-h)=\phi_{j-1}, u(t+s,x)=\phi_j, u(t+s,x+h)=\phi_{j+1}, u(t,x-h)=\psi_{j-1}, u(t,x)=\psi_j, u(t,x+h)=\psi_{j+1} \quad (4.2)$$

```
> stencil[2] := expand(isolate(subs(Subs,stencil[2]),phi));
```

$$stencil_2 := -2\,\phi_j\,h^2+d\,s\,\phi_{j-1}-2\,d\,s\,\phi_j+d\,s\,\phi_{j+1}+V(\phi_j)\,s\,h^2=-2\,h^2\,\psi_j-s\,d\,\psi_{j-1}+2\,s\,d\,\psi_j-s\,d\,\psi_{j+1}-s\,h^2\,V(\psi_j) \quad (4.3)$$

```
> master_CN := unapply(stencil[2],phi,psi,d,s,h,j);
```

$$master\_CN := (\phi,\psi,d,s,h,j)\rightarrow -2\,\phi_j\,h^2+d\,s\,\phi_{j-1}-2\,d\,s\,\phi_j+d\,s\,\phi_{j+1}+V(\phi_j)\,s\,h^2=-2\,h^2\,\psi_j-s\,d\,\psi_{j-1}+2\,s\,d\,\psi_j-s\,d\,\psi_{j+1}-s\,h^2\,V(\psi_j) \quad (4.4)$$

```
> phi := 'phi':
  psi := 'psi':
  M := 5:
  phi[0],psi[0],psi[M+1],phi[M+1] := 0,0,0,0;
  Vector([seq(master_CN(phi,psi,d,s,h,j),j=1..M)]);
```

$$\phi_0, \psi_0, \psi_6, \phi_6 := 0, 0, 0, 0$$

$$\left[\begin{array}{c} -2\,\phi_1\,h^2 - 2\,d\,s\,\phi_1 + d\,s\,\phi_2 + V(\phi_1)\,s\,h^2 = -2\,h^2\,\psi_1 + 2\,s\,d\,\psi_1 - s\,d\,\psi_2 - s\,h^2\,V(\psi_1) \\ -2\,\phi_2\,h^2 + d\,s\,\phi_1 - 2\,d\,s\,\phi_2 + d\,s\,\phi_3 + V(\phi_2)\,s\,h^2 = -2\,h^2\,\psi_2 - s\,d\,\psi_1 + 2\,s\,d\,\psi_2 - s\,d\,\psi_3 - s\,h^2\,V(\psi_2) \\ -2\,\phi_3\,h^2 + d\,s\,\phi_2 - 2\,d\,s\,\phi_3 + d\,s\,\phi_4 + V(\phi_3)\,s\,h^2 = -2\,h^2\,\psi_3 - s\,d\,\psi_2 + 2\,s\,d\,\psi_3 - s\,d\,\psi_4 - s\,h^2\,V(\psi_3) \\ -2\,\phi_4\,h^2 + d\,s\,\phi_3 - 2\,d\,s\,\phi_4 + d\,s\,\phi_5 + V(\phi_4)\,s\,h^2 = -2\,h^2\,\psi_4 - s\,d\,\psi_3 + 2\,s\,d\,\psi_4 - s\,d\,\psi_5 - s\,h^2\,V(\psi_4) \\ -2\,\phi_5\,h^2 + d\,s\,\phi_4 - 2\,d\,s\,\phi_5 + V(\phi_5)\,s\,h^2 = -2\,h^2\,\psi_5 - s\,d\,\psi_4 + 2\,s\,d\,\psi_5 - s\,h^2\,V(\psi_5) \end{array}\right] \quad (4.5$$

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

      # Defining the lattice parameters/functions as before:
      X := j -> L*(-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 := Array(0..M+1,[seq(X(j),j=0..M+1)],datatype=float);
      psi := map(z->f(z),XX);
      # Plot the first frame
      p[0] := Frame[2](XX,psi,s,h,T(0)):

      # Here is the main calculation loop (notice the LinearSolve command)
      for i from 1 to N do:
           phi := 'phi':
           phi[0],psi[0],psi[M+1],phi[M+1] := 0,0,0,0;
           sys := [seq(master_CN(phi,psi,d,s,h,j),j=1..M)];
           vars := [seq(phi[j],j=1..M)];
           guess := [seq(psi[j],j=1..M)];
           phi := Newton[vector](sys,vars,guess,1e-5):
           psi := Array(0..M+1,[0,op(phi),0],datatype=float):
           p[i] := Frame[2](XX,psi,s,h,T(i)):
      od:

      # Assemble the output movie
```

```
        display(convert(p,list),insequence=true);

end proc:

# Here is the plotting subroutine

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=["Crank-Nicholson"], color=
magenta;
        plot(Matrix([[xdata],[ydata]])^%T,title=Title,PlotOptions);

end proc:
> V := u -> 5*u*(1-u);
tau := 5;
N := 400;
M := 100;
L := 5;
d := 1;
f := x -> exp(-(10*x)^2);
s := evalf(2*L/(M+1));
h := evalf(tau/N);
movie[1] := FTCS(tau,L,N,M,d,f);
movie[2] := CN(tau,L,N,M,d,f);
pds := pdsolve(pde,[u(t,-L)=0,u(t,+L)=0,u(0,x)=f(x)],numeric,timestep=s,spacestep=h):
movie[3] := pds:-animate(t=0..tau,frames=N+1,axes=boxed,legend=["pdsolve/numeric"],color=
blue):
```

$$V := u \to 5\, u\, (1-u)$$

$$\tau := 5$$

$$N := 400$$

$$M := 100$$

$$L := 5$$

$$d := 1$$

$$f := x \to e^{-100\, x^2}$$

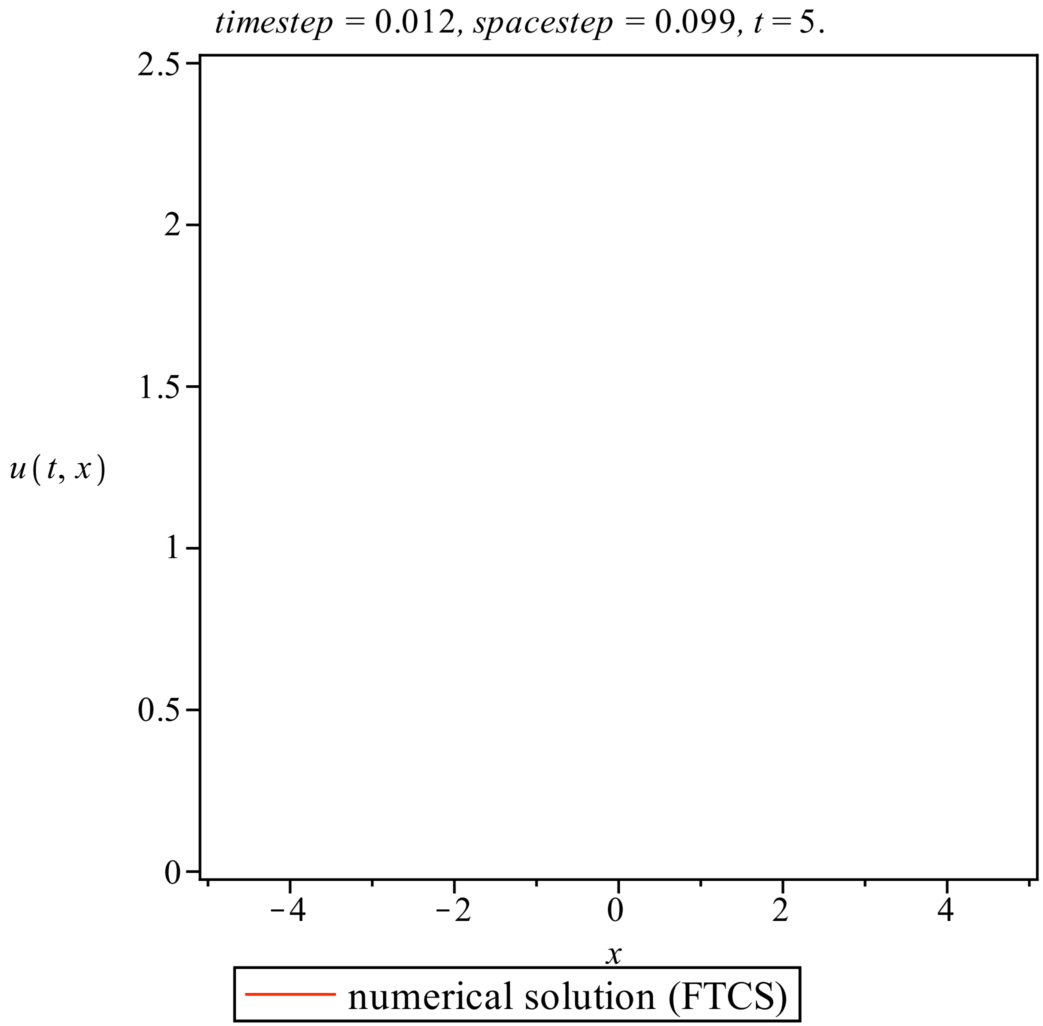
$$s := 0.099009900990099$$

$$h := 0.0125000000000000$$

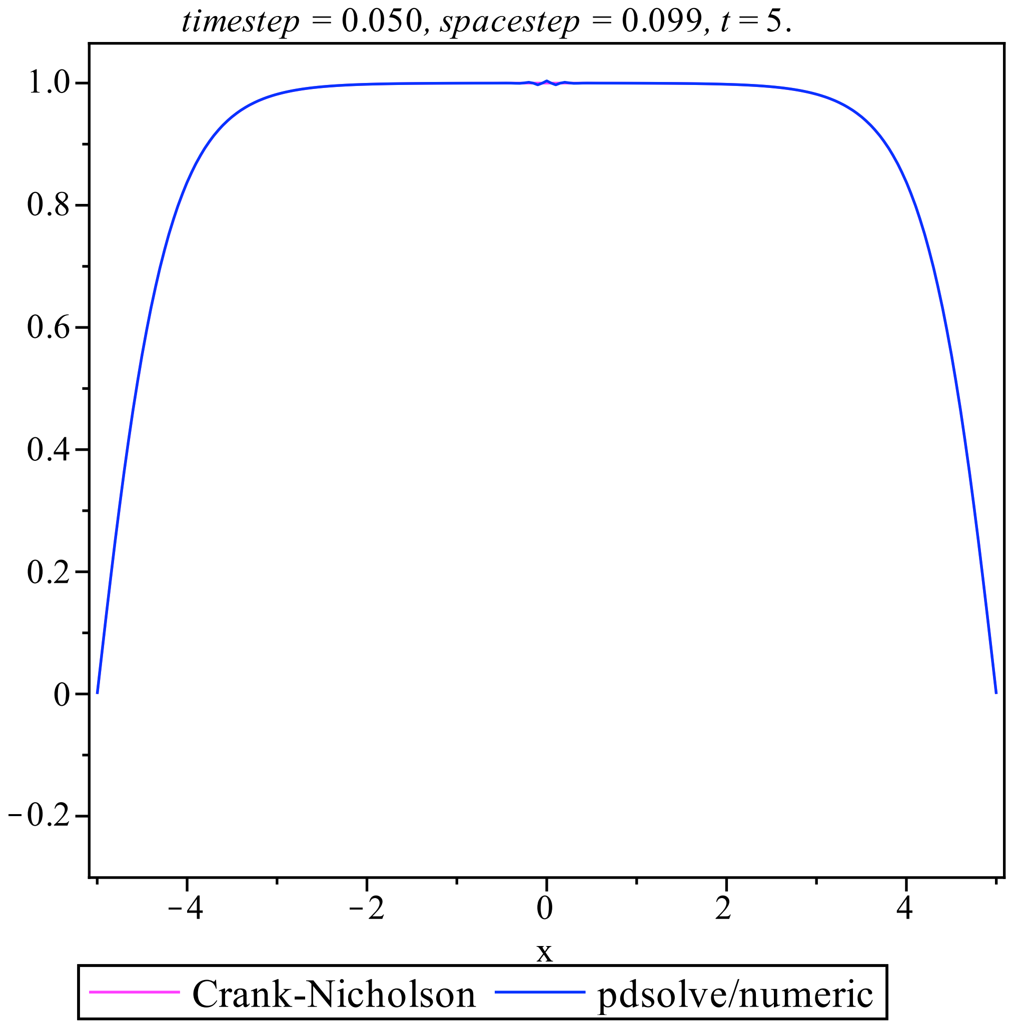
$$movie_1 := PLOT(\ldots)$$

$$movie_2 := PLOT(\ldots) \quad (4.6$$

```
> movie[1];
```




```
> display([movie[2],movie[3]]);
```


timestep = 0.050, spacestep = 0.099, t = 5.
1.0
0.8
0.6
0.4
0.2
0
−0.2
−4
−2
0
2
4
x
Crank-Nicholson
pdsolve/numeric