Math 660: Problem Set 4

Matthew Grasinger

April 5, 2017

Contents

1 C1: Crank-Nicolson Scheme, Hyperbolic equations 1
1.1 Source Code 3

2 C2: Viscous Burgers' Equation 6
2.1 Constant a, b, and c. Varied h and k. 6
2.2 Constant a, c, h and k. Varied b. 9
2.3 Source Code 13

3 Shared Code 16

1 C1: Crank-Nicolson Scheme, Hyperbolic equations

In the figures that follow this brief discussion, the Crank-Nicolson discrete approximation using the Thomas algorithm applied to the hyperbolic equation, $u_t + u_x = 0$, is plotted against the exact solution for $h = \frac{1}{10}, \frac{1}{20}$ and $\frac{1}{40}$. The relative L_∞ errors at $t = 1.0$ were 0.271, 0.104, and 0.045 for the $h = \frac{1}{10}, \frac{1}{20}$ and $\frac{1}{40}$ approximations, respectively. The error was primarily due to truncation error in the Crank-Nicolson scheme, as opposed to round off error in solution of the tridiagonal linear system of equations using the Thomas algorithm. This conclusion was made by comparing the solution using the Thomas algorithm to another solution using Gaussian elimination with partial pivoting. The difference between the Thomas algorithm solution and the Gaussian elimination with partial pivoting solutions were negligible (order of 10^{-15}). As expected, the error of the Crank-Nicolson scheme decreases with decreasing grid spacing. Also of note, the wave speed seems to be artificially slower for the coarser grid spacings ($h = \frac{1}{10}$ and $\frac{1}{20}$), but the discrete wave "catches up" to the exact solution as the grid is refined. As a result, the Crank-Nicolson approximation was quite accurate for $h = \frac{1}{40}$.

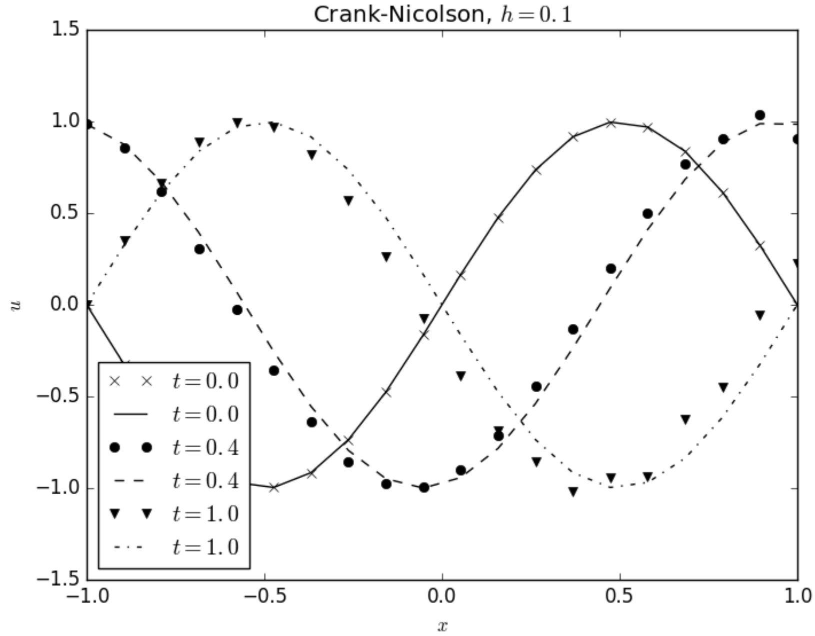

Figure 1: Crank-Nicolson scheme applied to the hyperbolic equation $u_t + u_x = 0$. The discretization is such that $h = \frac{1}{10}$ and $\lambda = 1.0$. The discrete solution (given by markers) is plotted with the exact solution (given by lines) for three different snapshots in time.

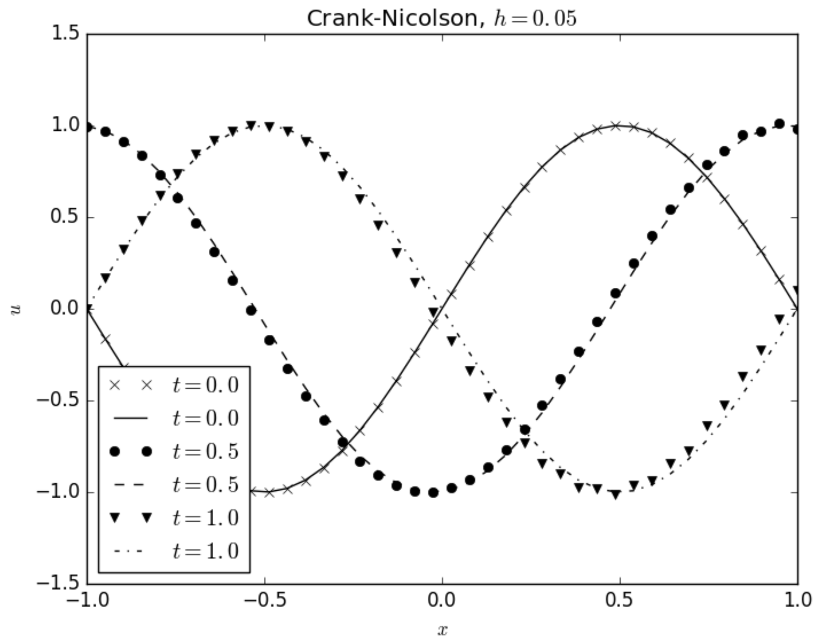

Figure 2: Crank-Nicolson scheme applied to the hyperbolic equation $u_t + u_x = 0$. The discretization is such that $h = \frac{1}{20}$ and $\lambda = 1.0$. The discrete solution (given by markers) is plotted with the exact solution (given by lines) for three different snapshots in time.

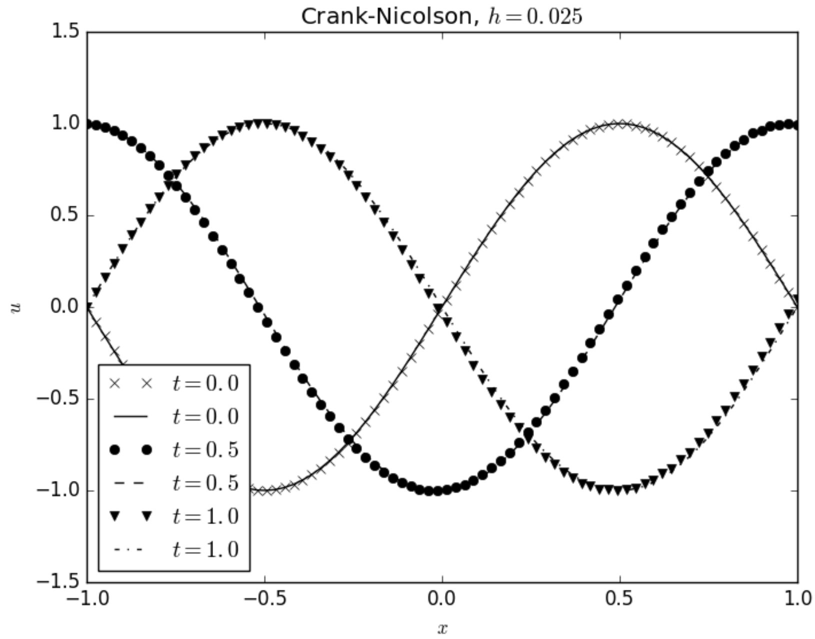

Figure 3: Crank-Nicolson scheme applied to the hyperbolic equation $u_t + u_x = 0$. The discretization is such that $h = \frac{1}{40}$ and $\lambda = 1.0$. The discrete solution (given by markers) is plotted with the exact solution (given by lines) for three different snapshots in time.

1.1 Source Code

```julia
include("hw4_helpers.jl");

const λ = 1.0;
const a = 1.0;

asoln(x, t) = sin(π * (x - t));

# debugging flag
const test_with_gauss_elim = false;

for h in [1/10; 1/20; 1/40]
  println("Crank-Nicolson, h = $h");
  const k = λ * h;
  const aa = -a * λ / 4;
  const bb = 1.0;
  const cc = -aa;

  xs = linspace(-1.0, 1.0, Int(round((2.0) / h)));
  ts = linspace(0.0, 1.0, Int(round(1.0 / k)));
  const M, K = length(xs), length(ts);

  u = zeros(M, K);
  u[:, 1] = map(x -> asoln(x, 0.0), xs);
  # used for debugging
  u_debug = (test_with_gauss_elim) ? zeros(M, K) : zeros(1, 1);
  if test_with_gauss_elim
    u_debug[:, 1] = map(x -> asoln(x, 0.0), xs);
  end

  @time for n in 1:K-1
    # used for debugging
    A_debug = (test_with_gauss_elim) ? zeros(M, M) : zeros(1, 1);
    b_debug = (test_with_gauss_elim) ? zeros(M) : zeros(1);
    if test_with_gauss_elim
      A_debug[1, 1] = 1.0;
      b_debug[1] = asoln(-1.0, ts[n+1]);
      for m=2:M-1
        A_debug[m, m-1] = aa;
        A_debug[m, m] = bb;
        A_debug[m, m+1] = cc;
        b_debug[m] = u_debug[m, n] - cc * u_debug[m+1, n] - aa * u_debug[m-1, n];
      end
      A_debug[M, M-1] = -λ;
      A_debug[M, M] = 1+λ;
      b_debug[M] = u_debug[M, n];
    end

    # calculate pi and qi for Thomas' algorithm
    p = zeros(M);
    q = zeros(M);
    p[2], q[2] = 0.0, asoln(-1.0, ts[n+1]);
    for m=2:M-1
      dd = u[m, n] - a * λ * (u[m+1, n] - u[m-1, n]) / 4;
      denom = aa * p[m] + bb;
      p[m+1] =  -cc / denom;
      q[m+1] = (dd - aa * q[m]) / denom;
    end
    u[M, n+1] = (u[M, n] + q[M]*λ) / (1 + λ - p[M]*λ);
    for m=M-1:-1:1
      u[m, n+1] = p[m+1] * u[m+1, n+1] + q[m+1];
    end

    if test_with_gauss_elim
      u_debug[:, n+1] = A_debug \ b_debug;
```

```julia
      @show norm(u[:, n+1] - u_debug[:, n+1], Inf);
    end
  end

  plot_solution(xs, ts, u, asoln; t="Crank-Nicolson, \$h = $h\$",
                show_plot=false, fname="cn_thomas_M-$M.png");
  if test_with_gauss_elim
    plot_solution(xs, ts, u_debug, asoln; t="Crank-Nicolson, \$h = $h\$",
                  show_plot=false, fname="cn_gauss_M-$M.png");
  end
  u_a = map(x -> asoln(x, 1.0), xs);
  println("Relative L2 error:  ", norm(u[:, end] - u_a, 2) / norm(u_a, 2));
  println("Relative L∞ error:  ", norm(u[:, end] - u_a, Inf) / norm(u_a, Inf));
  println();
end
```

2 C2: Viscous Burgers' Equation

2.1 Constant a, b, and c. Varied h and k.

In order to investigate the effect of grid spacing and time step size on the stability and accuracy of the forward time central space scheme applied to the viscous Burgers' equation, a, b, and c were held constant while h and k were varied. In the first three figures $h = 0.1$ and k is varied ($k = 0.001, 0.005$ and 0.01). The discrete solution agrees well for $k = 0.001$ and $k = 0.005$. However, for $k = 0.01$ the discrete solution quickly diverges (becomes NaN on the interior of the domain).

The last figure is for $h = 0.2$ and $k = 0.01$. The stability of the solution in the figure shows that $k = 0.005$ is not some special time step size above which the solution diverges, but instead that the stability of the discrete solution is sensitive to the ratio between the time step size and the grid spacing. It appears as though the discrete solution becomes divergent when $\lambda > 0.05$ where $\lambda = \frac{k}{h}$.

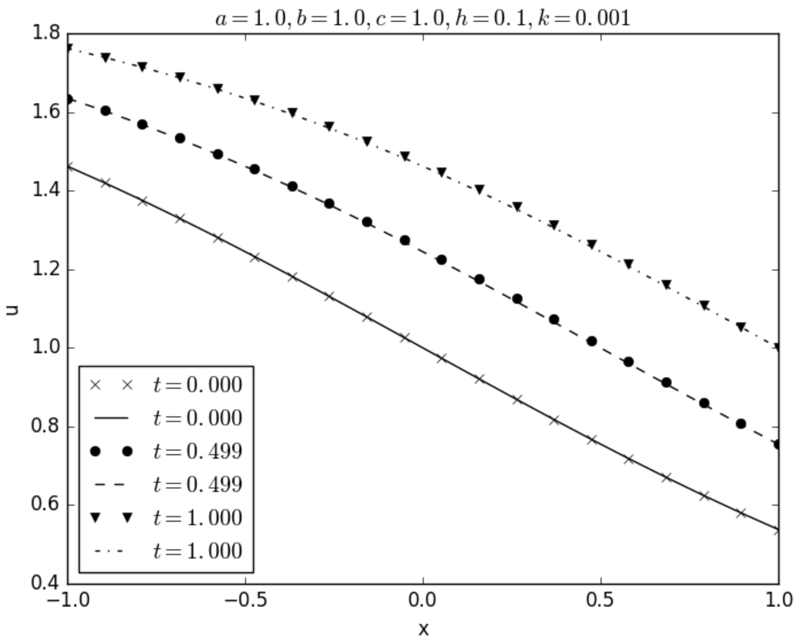

Figure 4: Forward time central space scheme applied to the viscous Burgers' equation $u_t + \left(\frac{u^2}{2}\right)_x = bu_{xx}$. The discrete solution (given by markers) is plotted with the exact solution, $u(x,t) = a - c\tanh\left(\frac{c}{2b}(x - at)\right)$ where $a = 1.0$, $b = 1.0$, $c = 1.0$, $h = 0.1$ and $k = 0.001$ (given by lines) for three different snapshots in time.

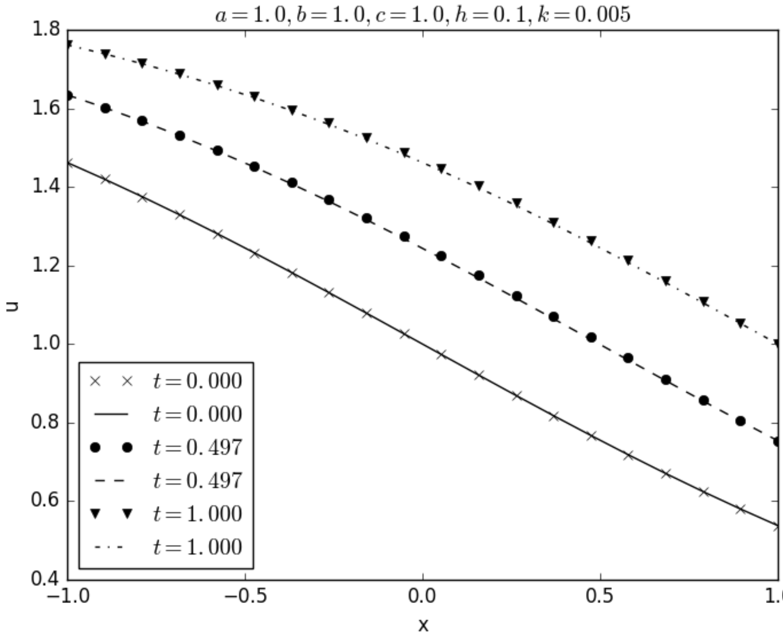

Figure 5: Forward time central space scheme applied to the viscous Burgers' equation $u_t + \left(\frac{u^2}{2}\right)_x = bu_{xx}$. The discrete solution (given by markers) is plotted with the exact solution, $u(x,t) = a - c\tanh\left(\frac{c}{2b}(x-at)\right)$ where $a = 1.0$, $b = 1.0$, $c = 1.0$, $h = 0.1$ and $k = 0.005$ (given by lines) for three different snapshots in time.

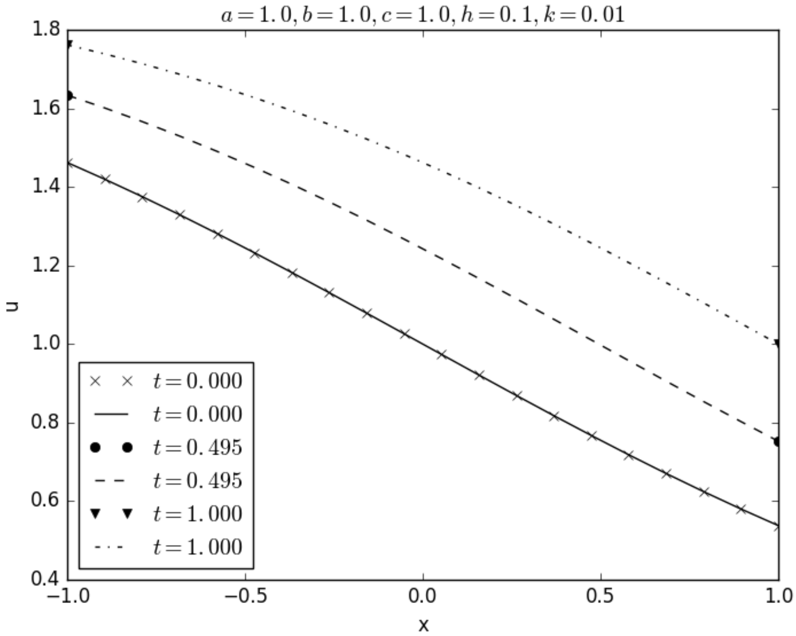

Figure 6: Forward time central space scheme applied to the viscous Burgers' equation $u_t + \left(\frac{u^2}{2}\right)_x = bu_{xx}$. The discrete solution (given by markers) is plotted with the exact solution, $u(x,t) = a - c\tanh\left(\frac{c}{2b}(x - at)\right)$ where $a = 1.0$, $b = 1.0$, $c = 1.0$, $h = 0.1$ and $k = 0.01$ (given by lines) for three different snapshots in time.

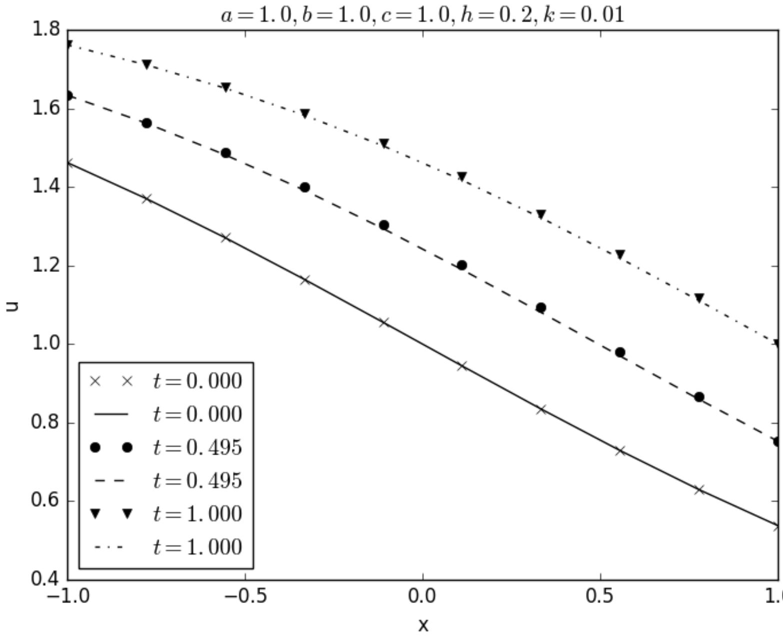

Figure 7: Forward time central space scheme applied to the viscous Burgers' equation $u_t + \left(\frac{u^2}{2}\right)_x = bu_{xx}$. The discrete solution (given by markers) is plotted with the exact solution, $u(x,t) = a - c\tanh\left(\frac{c}{2b}(x - at)\right)$ where $a = 1.0$, $b = 1.0$, $c = 1.0$, $h = 0.2$ and $k = 0.01$ (given by lines) for three different snapshots in time.

2.2 Constant a, c, h and k. Varied b.

In order to investigate the effect of the diffusion coefficient on the stability and accuracy of the forward time central space scheme applied to the viscous Burgers' equation, a, c, h and k were held constant while b was varied; $b = 0.5, 0.1, 0.01, 0.001$. As b is decreased, the solution becomes more nonlinear. For $b = 0.5$ and 0.1 the discrete solution agrees well with the exact solution. However, for smaller values of b ($b = 0.01, 0.001$) large oscillations manifest in the discrete solution.

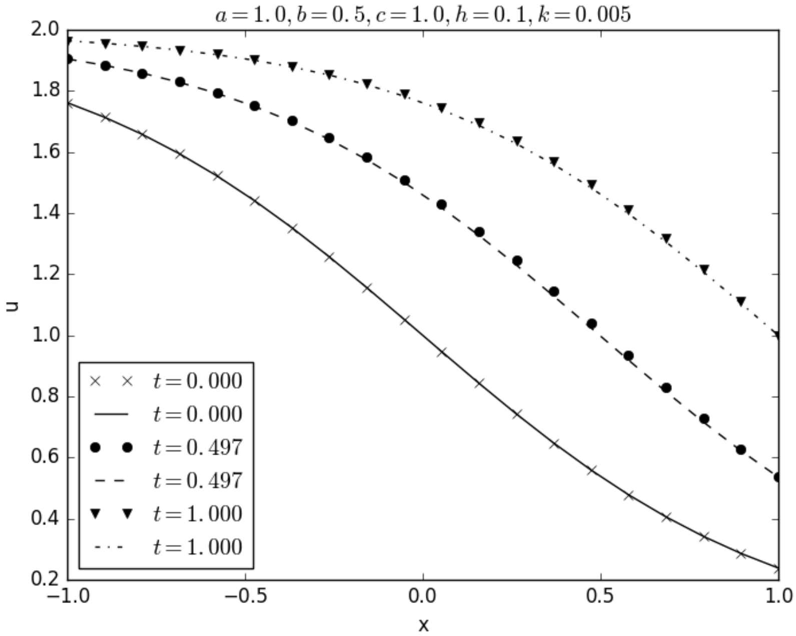

Figure 8: Forward time central space scheme applied to the viscous Burgers' equation $u_t + \left(\frac{u^2}{2}\right)_x = bu_{xx}$. The discrete solution (given by markers) is plotted with the exact solution, $u\,(x,t) = a - c\tanh\left(\frac{c}{2b}\,(x - at)\right)$ where $a = 1.0$, $b = 0.5$, $c = 1.0$, $h = 0.1$ and $k = 0.005$ (given by lines) for three different snapshots in time.

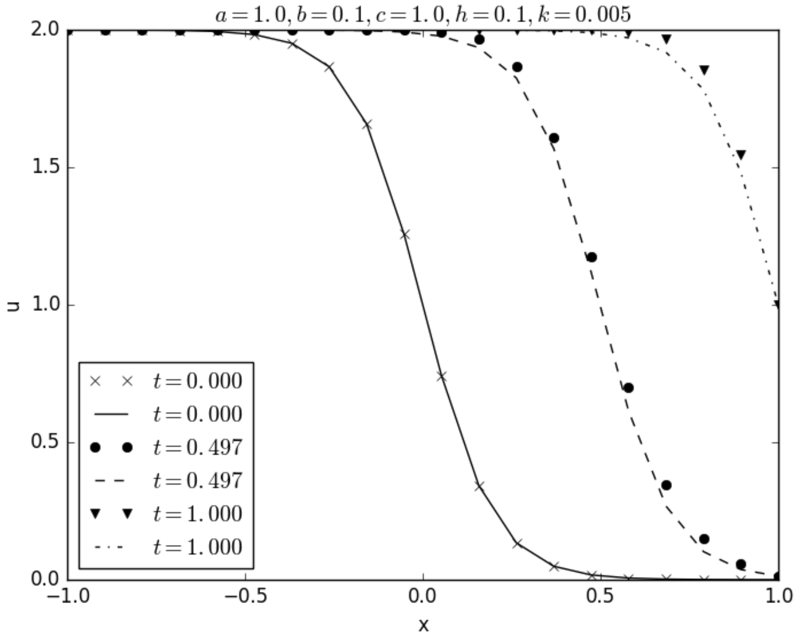

Figure 9: Forward time central space scheme applied to the viscous Burgers' equation $u_t + \left(\frac{u^2}{2}\right)_x = bu_{xx}$. The discrete solution (given by markers) is plotted with the exact solution, $u(x,t) = a - c\tanh\left(\frac{c}{2b}(x - at)\right)$ where $a = 1.0$, $b = 0.1$, $c = 1.0$, $h = 0.1$ and $k = 0.005$ (given by lines) for three different snapshots in time.

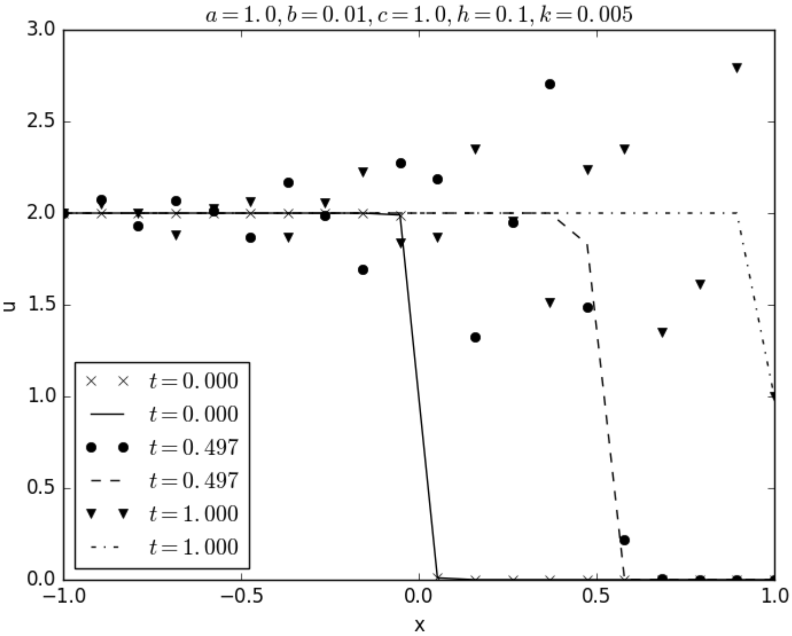

Figure 10: Forward time central space scheme applied to the viscous Burgers' equation $u_t + \left(\frac{u^2}{2}\right)_x = bu_{xx}$. The discrete solution (given by markers) is plotted with the exact solution, $u(x,t) = a - c\tanh\left(\frac{c}{2b}(x-at)\right)$ where $a = 1.0$, $b = 0.01$, $c = 1.0$, $h = 0.1$ and $k = 0.005$ (given by lines) for three different snapshots in time.

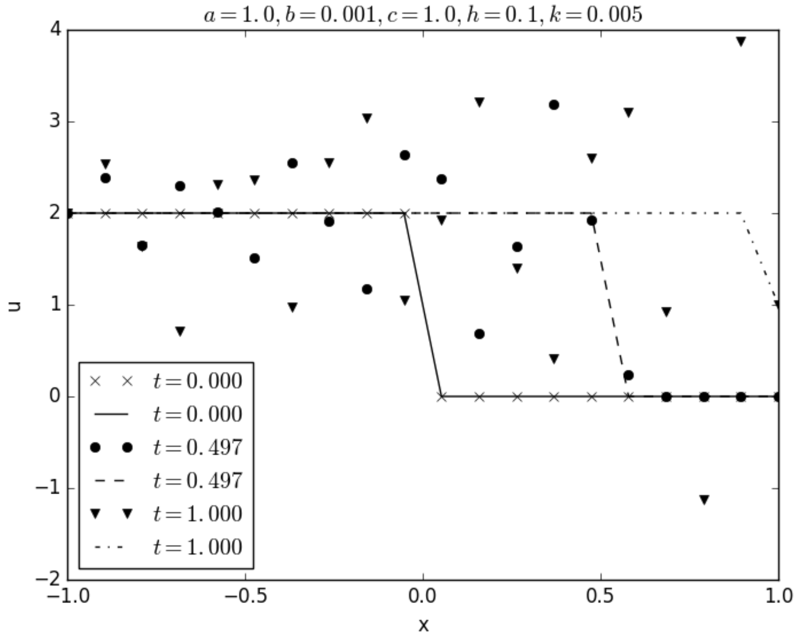

Figure 11: Forward time central space scheme applied to the viscous Burgers' equation $u_t + \left(\frac{u^2}{2}\right)_x = bu_{xx}$. The discrete solution (given by markers) is plotted with the exact solution, $u(x,t) = a - c\tanh\left(\frac{c}{2b}(x - at)\right)$ where $a = 1.0$, $b = 0.001$, $c = 1.0$, $h = 0.1$ and $k = 0.005$ (given by lines) for three different snapshots in time.

2.3 Source Code

```julia
include("hw4_helpers.jl");

using ArgParse;

s = ArgParseSettings();
@add_arg_table s begin
  "-A"
    help = "Coefficient of 1/2 d(u^2)/dx, like a wave speed"
    arg_type = Float64
    default = 1.0
  "-B"
    help = "Diffusion coefficient"
    arg_type = Float64
    default = 1.0
  "-C"
    help = "Coefficient of du/dt"
    arg_type = Float64
    default = 1.0
  "-H"
    help = "Grid spacing"
    arg_type = Float64
    default = 0.2
  "-K"
    help = "Time step size"
    arg_type = Float64
    default = 0.01
  "--show-plot", "-P"
    help = "show plot of solution"
    action = :store_true
  "--fname", "-f"
    help = "file name of plot"
    default = ""
  "--show-error", "-E"
    help = "show L2 error"
    action = :store_true
end

pa = parse_args(s);

const a = pa["A"];
const b = pa["B"];
const c = pa["C"];
const h = pa["H"];
const k = pa["K"];

# analytical solution
asoln(x, t) = a - c * tanh(c / (2*b) * (x - a*t));

xs = linspace(-1.0, 1.0, Int(round((2.0) / h)));
ts = linspace(0.0, 1.0, Int(round(1.0 / k)));
const M, K = length(xs), length(ts);

u = SharedArray(Float64, (M, K); init = x -> 0);
u[:, 1] = map(x -> asoln(x, 0.0), xs);

const α1 = (k * a) / (4 * h);
const α2 = (k * b) / (h^2);

for n in 1:K-1
  for m=2:M-1
    u[m, n+1] = u[m, n] + (-α1 * (u[m+1, n]*u[m+1, n] - u[m-1, n]*u[m-1, n])
                    + α2 * (u[m+1, n] - 2 * u[m, n] + u[m-1, n])) / c;
  end
  u[1, n+1] = asoln(xs[1], ts[n+1]);
```

```julia
    u[end, n+1] = asoln(xs[end], ts[n+1]);
  end

  if pa["show-plot"] || pa["fname"] != ""
    plot_solution(xs, ts, u, asoln; t="\$a = $a, b = $b, c = $c, h = $h, k = $k\$",
                  show_plot=pa["show-plot"], fname=pa["fname"]);
  end

  if pa["show-error"]
    println("Relative L2 error");
    for (t, n) in zip(ts, 1:K)
      u_analytical = map(x -> asoln(x, t), xs);
      println(@sprintf("%5.4lf %5.4lf", t,
                       norm(u[:, n] - u_analytical, 2) / norm(u_analytical, 2)));
    end
  end
```

3 Shared Code

Some visualization code was shared between the files for Section 1 on page 1 and Section 2 on page 6. For the sake of completeness, it is included below.

```julia
using PyPlot;

# plot a solution at three instances in time
function plot_solution(xs, ts, us::AbstractMatrix{Float64}; t::String="",
                       fname::String="", show_plot::Bool=false)

  plot(xs, us[:, 1], "k-"; label=@sprintf("\$t = %.1f\$", ts[1]));
  plot(xs, us[:, div(length(ts), 2)], "k--"; label=@sprintf("\$t = %.1f\$",
                                                    ts[div(length(ts), 2)]));
  plot(xs, us[:, end], "k-."; label=@sprintf("\$t = %.1f\$", ts[end]));
  legend(; loc=3);
  if t != ""
    title(t);
  end
  xlabel("\$x\$");
  ylabel("\$u\$");
  if show_plot
    show();
  end
  if fname != ""
    savefig(fname);
  end
  clf();
end

# plot solution at three instances in time vs. an analytical solution
function plot_solution(xs, ts, us::AbstractMatrix{Float64}, asoln::Function;
                       t::String="", fname::String="", show_plot::Bool=false)

  plot(xs, us[:, 1], "kx"; label=@sprintf("\$t = %.1f\$", ts[1]));
  plot(xs, map(x -> asoln(x, 0), xs), "k-"; label=@sprintf("\$t = %.1f\$", ts[1]));
  plot(xs, us[:, div(length(ts), 2)], "ko"; label=@sprintf("\$t = %.1f\$",
                                                    ts[div(length(ts), 2)]));
  plot(xs, map(x -> asoln(x, ts[div(length(ts), 2)]), xs), "k--";
              label=@sprintf("\$t = %.1f\$", ts[div(length(ts), 2)]));
  plot(xs, us[:, end], "kv"; label=@sprintf("\$t = %.1f\$", ts[end]));
  plot(xs, map(x -> asoln(x, ts[end]), xs), "k-.";
          label=@sprintf("\$t = %.1f\$", ts[end]));
  legend(; loc=3);
  if t != ""
    title(t);
  end
  xlabel("\$x\$");
  ylabel("\$u\$");
  if show_plot
    show();
  end
  if fname != ""
    savefig(fname);
  end
  clf();
end
```