AMATH 586 SPRING 2020
HOMEWORK 4 — DUE MAY 29 ON GITHUB BY 11PM

SHANNON DOW

Be sure to do a `git pull` to update your local version of the `amath-586-2020` repository.

Problem 1: Consider solving the following heat equation with "linked" boundary conditions

$$\begin{cases} u_t = \frac{1}{2}u_{xx} \\ u(0,t) = su(1,t) \\ u_x(0,t) = u_x(1,t), \\ u(x,0) = \eta(x), \end{cases}$$

where $s \neq -1$. Recall that the MOL discretization with the standard second-order stencil can be written as

$$U'(t) = \frac{1}{2h^2}AU(t) + \begin{bmatrix} \frac{U_0(t)}{2h^2} \\ \vdots \\ \frac{U_{m+1}(t)}{2h^2} \end{bmatrix}, \quad A = \begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & 1 & -2 & 1 & \\ & & \ddots & \ddots & \ddots \\ & & & 1 & -2 \end{bmatrix}.$$

The first boundary condition is naturally enforced via $U_0(t) = sU_{m+1}(t)$. Show that if we suppose

$$\frac{U_1(t) - U_0(t)}{h} = \frac{U_{m+1}(t) - U_m(t)}{h},$$

then the MOL system becomes

$$U'(t) = \frac{1}{2h^2}BU(t), \quad B = \begin{bmatrix} -2 + \frac{s}{1+s} & 1 & & & & \frac{s}{1+s} \\ 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -2 & 1 \\ \frac{1}{1+s} & & & & 1 & -2 + \frac{1}{1+s} \end{bmatrix}. \tag{1}$$

Solution Using the standard second order stencil, we can see that:

$$U_i'(t) = \frac{U_{i-1}(t) - 2U_i(t) + U_{i+1}(t)}{2h^2}$$

If we assume:

$$\frac{U_1(t) - U_0(t)}{h} = \frac{U_{m+1}(t) - U_m(t)}{h}$$

Then, using $U_0(t) = sU_{m+1}(t)$

$$\begin{aligned}
U_0(t) &= -U_{m+1}(t) + U_m(t) + U_1(t) \\
U_0(t) + U_{m+1}(t) &= +U_m(t) + U_1(t) \\
U_0(t) + \frac{1}{s}U_0(t) &= U_m(t) + U_1(t) \\
U_0(t) &= \frac{s}{1+s}U_m(t) + \frac{s}{1+s}U_1(t)
\end{aligned}$$

The second order approximation for $U_1'(t)$ becomes:

$$\begin{aligned}
U_1'(t) &= \frac{U_0(t) - 2U_1(t) + U_2(t)}{2h^2} \\
U_1'(t) &= \frac{\frac{s}{1+s}U_m(t) + \frac{s}{1+s}U_1(t) - 2U_1(t) + U_2(t)}{2h^2} \\
U_1'(t) &= \frac{(-2 + \frac{s}{1+s})U_1(t) + U_2(t) + \frac{s}{1+s}U_m(t)}{2h^2}
\end{aligned}$$

If we assume:

$$\frac{U_1(t) - U_0(t)}{h} = \frac{U_{m+1}(t) - U_m(t)}{h}$$

Then, using $U_0(t) = sU_{m+1}(t)$

$$\begin{aligned}
U_{m+1}(t) &= U_1(t) - U_0(t) + U_m(t) \\
U_0(t) + U_{m+1}(t) &= +U_m(t) + U_1(t) \\
sU_{m+1}(t) + U_{m+1}(t) &= U_m(t) + U_1(t) \\
U_{m+1}(t) &= \frac{1}{1+s}U_m(t) + \frac{1}{1+s}U_1(t)
\end{aligned}$$

The second order approximation for $U_m'(t)$ becomes:

$$\begin{aligned}
U_m'(t) &= \frac{U_{m-1}(t) - 2U_m(t) + U_{m+1}(t)}{2h^2} \\
U_m'(t) &= \frac{U_{m-1}(t) - 2U_m(t) + \frac{1}{1+s}U_m(t) + \frac{1}{1+s}U_1(t)}{2h^2} \\
U_m'(t) &= \frac{\frac{1}{1+s}U_1(t) + U_{m-1}(t) + (-2 + \frac{1}{1+s})U_m(t)}{2h^2}
\end{aligned}$$

Thus, it can be written as above.

Problem 2: Apply the backward Euler method to (1) to give

$$\left(I - \frac{k}{2h^2}B\right)U^{n+1} = U^n, \quad U^n = \begin{bmatrix} U_1^n \\ U_2^n \\ \vdots \\ U_m^n \end{bmatrix}, \tag{2}$$

and write a routine to solve the system (1) with initial condition

$$\eta(x) = e^{-20(x-1/2)^2},$$

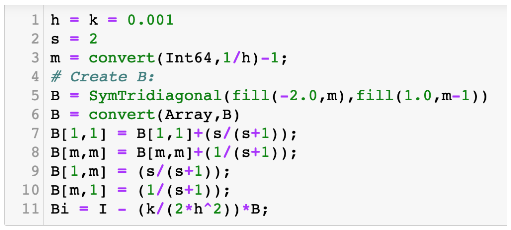
using $k = h$ and $h = 0.001$ with $s = 2$. Plot the solution at times $t = 0.001, 0.01, 0.1$. Note: One could use trapezoid to solve this problem but it wouldn't preserve some important features that we care about. See the last extra credit problem.

Solution Applying Backward Euler:
Replace the time derivative with the forward difference, and it implicit:

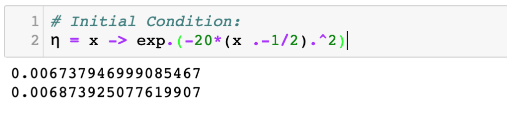
$$U_t = U_{xx}$$
$$\frac{U_i^{n+1} - U_i^n}{k} = \frac{U_{i-1}^{n+1} - 2U_i^{n+1} + U_{i+1}^{n+1}}{2h^2}$$

Applying the boundary conditions, we know that we can replace the right hand side with B:

$$\frac{U_i^{n+1} - U_i^n}{k} = \frac{1}{2h^2}BU^{n+1}$$
$$U_i^{n+1} - U_i^n = \frac{k}{2h^2}BU^{n+1}$$
$$U^n = (I - \frac{k}{2h^2}B)U^{n+1}$$

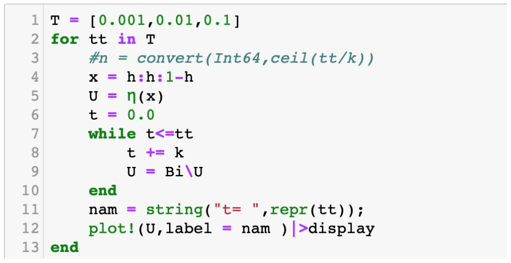


```
h = k = 0.001
s = 2
m = convert(Int64,1/h)-1;
# Create B:
B = SymTridiagonal(fill(-2.0,m),fill(1.0,m-1))
B = convert(Array,B)
B[1,1] = B[1,1]+(s/(s+1));
B[m,m] = B[m,m]+(1/(s+1));
B[1,m] = (s/(s+1));
B[m,1] = (1/(s+1));
Bi = I - (k/(2*h^2))*B;
```

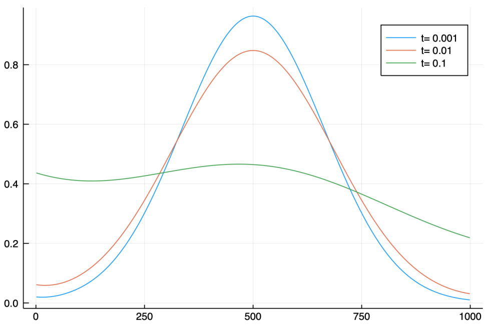


```
# Initial Condition:
η = x -> exp.(-20*(x .-1/2).^2)
```

```
0.006737946999085467
0.006873925077619907
```



```
T = [0.001,0.01,0.1]
for tt in T
    #n = convert(Int64,ceil(tt/k))
    x = h:h:1-h
    U = η(x)
    t = 0.0
    while t<=tt
        t += k
        U = Bi\U
    end
    nam = string("t= ",repr(tt));
    plot!(U,label = nam )|>display
end
```



Problem 3: In the next two problems you will use the heat equation to assist with a statistics problem.

- Consider data points $X_1, X_2, \dots, X_N, \dots$ each being a real number arising from a repeated experiment. We may want to know what probability distribution (if any) they come from. One way of coming up with an approximation to the density is to use

$$\frac{1}{N}\sum_{j=1}^{N}\frac{1}{\sqrt{2\pi t}}\exp\left(-\frac{(x-X_j)^2}{2t}\right), \quad t>0. \tag{3}$$

  Use normally distributed random data (`X = randn(n)` in `Julia`, `X = randn(n,1)` in `Matlab` and `X = numpy.random.randn(n,1)` in `Python`) with $n = 10000$ and plot this function for $t = 0.001, 0.01, 0.1, 1, 10$ and compare it with the true probabilty density function for the data: $\rho(x) = \frac{1}{\sqrt{2\pi}}e^{-x^2/2}$. Visually, which "time" t gives the best approximation?

  Note: The solution of the heat equation $u_t = \frac{1}{2}u_{xx}$ with initial condition $u(x,0) = \delta(x)$ where δ is the standard Dirac delta function is given by $u(x,t) = \frac{1}{\sqrt{2\pi t}}\exp\left(-\frac{x^2}{2t}\right)$. So (3) can be seen as the solution of $u_t = \frac{1}{2}u_{xx}$ with

$$u(x,0) = \frac{1}{N}\sum_{j=1}^{N}\delta(x-X_j).$$

- The previous approach works well if the underlying distribution is smooth and decays exponentially in both directions. But there physical situations within cell biology, in particular, where the density should only be non-zero on a finite interval $[0,1]$ and satisfy some natural boundary conditions:

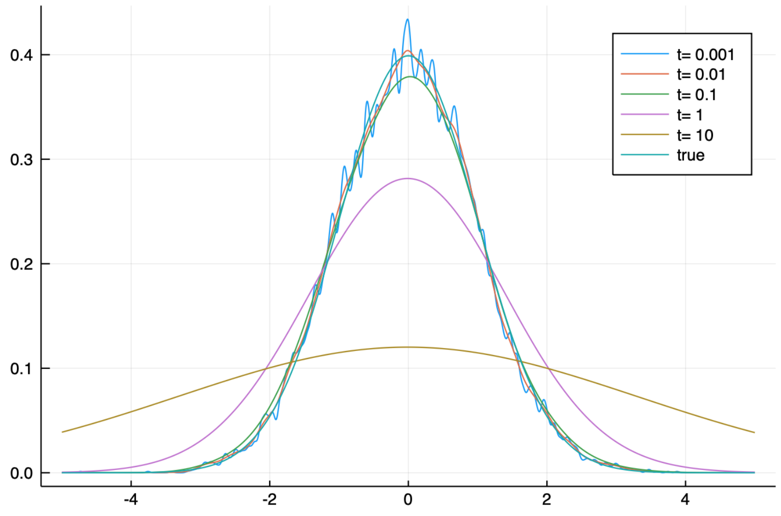
$$\rho(0) = s\rho(1), \quad \rho'(0) = \rho'(1).$$

  An example of such a function for $s = 2$ is given by

$$\rho(x) = -\frac{2}{3}x + \frac{4}{3} + \frac{1}{2}\sin(2\pi x).$$

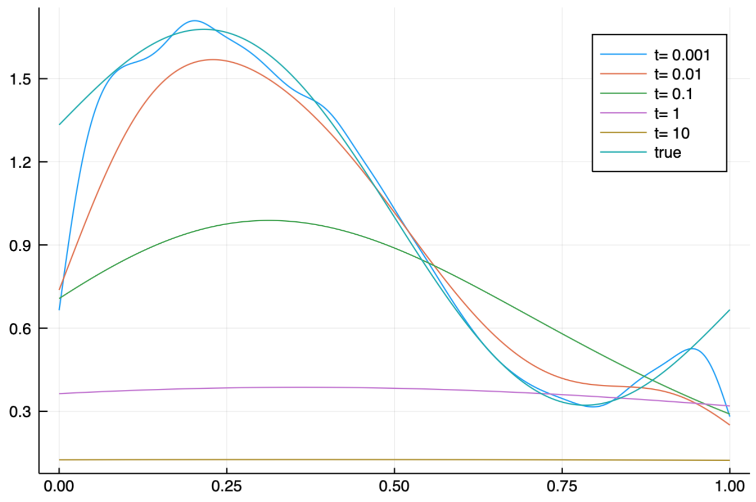
  Code to generate $X_1, X_2, \dots, X_N, \dots$ with this probability density in our three languages is given at the end of the homework. Repeat the calculation in the prevous part with this data, $X_1, X_2, \dots$.

Solution
Part (a) :



The t = 0.01 looks like the best approximation, visually, with t = 0.1 as a close second.

Part (b):



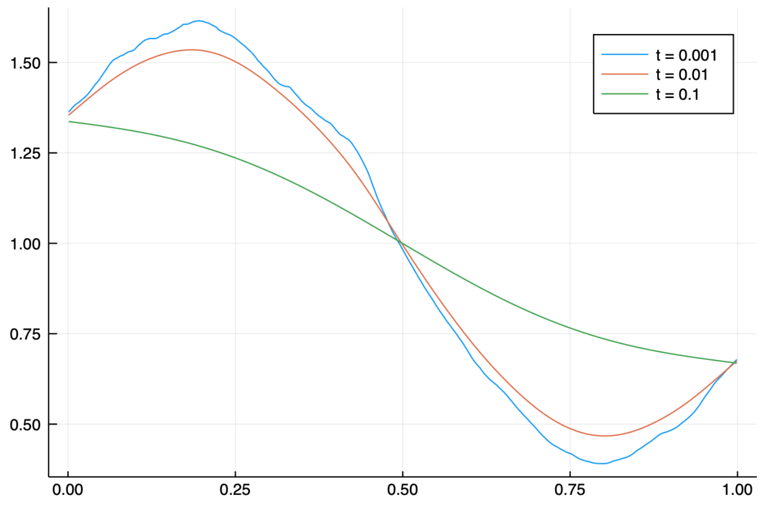
Using the prand function provided on the domain [0,1], this shape is produced. The closest approximation is when t = 0.001, with 001 relatively close.

Problem 4: Consider binning data $X_1, X_2, \dots, X_N$, $X_j \in (0, 1)$ as follows:

- Find Y_i so that Y_i is the number of data points X_j that lie in the interval $[ih, (i+1)h) = [x_i, x_{i+1})$.
- Set $U_i^0 = \frac{Y_i}{hN}$.

With $N = m$, $h = 0.0001$, $k = 10h$, $s = 2$, generate $X_1, \dots, X_N$ using the `prand` function, and bin the data to get the initial condition U_i^0, $i = 1, 2, \dots, m$ for the MOL discretization (1). Solve with this initial condition using your code from **Problem 2** to times $t = 0.001, 0.01, 0.1$. Compare with Part 2 of Problem 3.

Solution



```
m = convert(Int64,1/h)-1;
y = zeros(m)
plot(y)
XX = prand(m)|> Array
x = LinRange(0,1,m)|> Array
for i = 1:m
    for j = 1:m
        if XX[j]>= x[i] && XX[j]<x[i+1]
            y[i]+=1;
        end
    end
end
B = SymTridiagonal(fill(-2.0,m),fill(1.0,m-1))
B = convert(Array,B)
B[1,1] = B[1,1]+(s/(s+1));
B[m,m] = B[m,m]+(1/(s+1));
B[1,m] = (s/(s+1));
B[m,1] = (1/(s+1));
Bi = I - (k/(2*h^2))*B;

T = [0.001,0.01,0.1]
x = h:h:1-h
function backEul(x,uo,B,tt)
    U = uo
    t = 0.0
    while t<=tt
        t += k
        U = B\U
    end
    return U
end
```

The shape of these is comparable to the ones from the second part of problem 3. Julia for some reason could not take h = 0.0001, so h= 0.001 was used.

Problem 5: It can be shown that all the eigenvalues of B are simple, real and non-positive. Explain why this proves that the method (2) is Lax-Richtmyer stable.

Solution First, we must put the method (2) into the form $U^{n+1}U^n + g^n(k)$

$$(I - \frac{k}{2h}B)U^{n+1} = U^n$$

$$U^{n+1} = (I - \frac{k}{2h}B)^{-1}U^n$$

$$B(k) = (I - \frac{k}{2h}B)^{-1}$$

Now, to be Lax-Richtmyer stable, for each time t, there is a constant $C_t > 0$ such that $\|B(k)^n\| < C_T \forall$ k sufficiently small and integers n satisfying $kn \leq t$
Additionally, if λ is an eigenvalue of B(k), then we know that $\frac{1}{\lambda}$ is an eigenvalue of $B(k)^{-1}$
Since the eigenvalues of B are known to be simple, real and non-positive, the eigenvalues of $(I - \frac{k}{2h}B)$ must be greater than 1. Therefore, the eigenvalues of B(k) must be less than 1. This means that we can bound $\|B(k)^n\| < 1$ or smaller.

Problem 6: A challenge (extra credit): For $s > 0$, establish:

- $\begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix} B = 0$ and therefore $\sum_j U_j^n = \sum_j U_j^0$ for all n.
- If y is a vector with non-negative entries and $\left(I - \frac{k}{2h^2} B\right) x = y$ then x has non-negative entries.

Explain why this shows that if $\sum_j U_j^0 = 1$ then at each step n we can interpret U_j^n as the evolution of a probability distribution.

```
## Julia
function prand(m)
  p = x -> -(2.0/3)*x.+4.0/3 .+ .5sin.(2*pi*x)
  B = 1.7
  out = fill(0.,m)
  for j = 1:m
    u = 10.
    y = 0.
    while u >= p(y)/B
      y = rand()
      u = rand()
    end
    out[j] = y
  end
  out
end

 %% Matlab
 function out = prand(m)
    p = @(x) -(2/3)*x + 4/3 + .5*sin(2*pi*x);
    B = 1.7;
    out = zeros(m,1);
    for j = 1:m
        u = 10.;
        y = 0.;
        while u >= p(y)/B
            y = rand();
            u = rand();
        end
        out(j) = y;
    end
end

## Python
import numpy as np

def psamp(m):
```

```
p = lambda x: -(2.0/3)*x +4.0/3 + 0.5*np.sin(2*np.pi*x)
B = 1.7
out = np.zeros(m)
for j in np.arange(m):
  u = 10.
  y = 0.
  while u >= p(y)/B:
    y = np.random.rand()
    u = np.random.rand()
  out[j] = y
return out
```