PHYS 410/555: Computational Physics Fall 2004 Homework 4
DUE: Thursday, November 25, 10:00 AM Report bugs to choptuik@physics.ubc.ca

The following assignment involves writing, testing, running and analyzing three Fortran 77 programs which use LAPACK routines to solve various linear systems. Follow the instructions carefully and do all program development on the `lnx` *machines. All files required by the assignment must reside in the correct places on your* `lnx` *account for the homework to be considered complete. Note that you are free—and in fact encouraged—to use routines covered in class and available via the class notes as well as routines that you may have coded in previous homeworks.*

Problem 1: In directory `~/hw4/a1` on your `lnx` account, create a source file `lsolve.f` and corresponding executable `lsolve` which solves a general linear system. `lsolve` is to accept exactly two arguments:

```
usage: lsolve <matrix file> <rhs file>
```

where `<matrix file>` is the name of a file which contains an $n \times n$ matrix (A) and `<rhs file>` is the name of a file which contains an n–element column vector (b). `lsolve` is to read the matrix and the right-hand-side vector, then compute the solution, x of the linear system

$$A\,x = b$$

using the `LAPACK` routine `DGESV`. If the solution is successful, the solution vector x is to be written to standard output, one number per line. Use the routines `dvfrom` and `dvto` (see Fortran Notes, Lecture 3) to read and write the vectors and `dmfrom` (Fortran Notes, Lecture 6) to read the matrix. It is probably best to include these routines in your `lsolve.f` source file. Recall that all source code covered in class is available in the `phys410` account on `lnx`. Also note that `dmfrom` reads array entries in standard Fortran order; i.e. column-by-column. Your implementation of `lsolve` should do a reasonable amount of error checking, including:

- verification that the input arguments refer to existing files
- verification that the input matrix, A, is square (i.e. $n \times n$ for some n)
- verification that the length of the rhs vector, b, is n, given that A is $n \times n$
- verification that sufficient storage exists to store A

If any of the above conditions are not satisfied, `lsolve` should print an error-message to standard error, then exit. Note that you need not test for file-existence in the main program since both `dvfrom` and `dmfrom` return information which can be used to deduce that the specified file did not exist. Be sure to test your program using, for example, a small–n problem such as that covered in Lecture 1 of the Linear Systems Notes, as well as various types of *invalid* input.

Hint: If your solution to this problem starts to get complicated, particularly in terms of how the matrix and vectors are dealt with, you should assume that there *is* an easier way!

Problem 2: The next two problems concern the solution of a simple one-dimensional diffusion equation using finite-difference techniques. For this problem, all file references are relative to a solution directory `~/hw4/a2`, that you must create as usual.
Specifically, we consider the partial differential equation

$$u_t = u_{xx} \tag{2.1}$$

where $u \equiv u(t, x)$ and we have adopted a standard "subscript" notation for partial differentiation, so that e.g., $u_t \equiv \partial u/\partial t$. [1]

[1] Those of you who are more familiar with the diffusion equation in the form $u_t = a^2 u_{xx}$ for some (squared) diffusion constant, a^2, should note that there is no loss in generality in solving the "non-dimensionalized" form (2.1), since we can always cast the more general diffusion equation into the specific form (2.1) through a change of variables that amounts to a choice of "natural" units for the problem. Let me know, or ask a fellow student, if this is *not* clear to you.

We will solve (2.1) on the domain

$$0 \le x \le 1 \qquad 0 \le t \le t_{\max} \tag{2.2}$$

subject to initial conditions

$$u(0,x) = u_0(x)\,, \tag{2.3}$$

where $u_0(x)$ is some prescribed function, and (Dirichlet) boundary conditions

$$u(t,0) = u(t,1) = 0\,. \tag{2.4}$$

Note that the initial conditions (2.3) need to be compatible with the boundary conditions (2.4), i.e. we should ensure that $u_0(0) = u_0(1) = 0$.
We will hereafter refer to the system (2.1–2.4) as *the* diffusion equation.

Now consider the specific initial data

$$u(0,x) \equiv u_0(x) = \sin(2\pi x)\,. \tag{2.5}$$

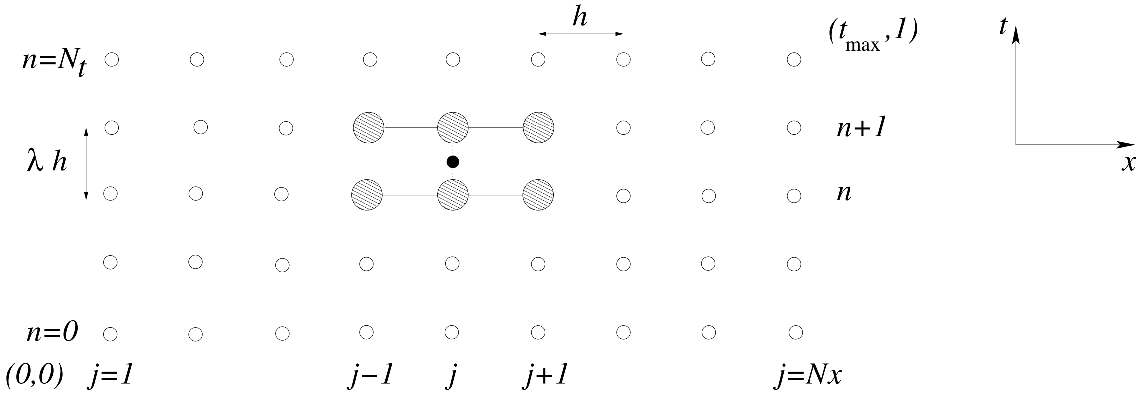
Problem 2a)

For the initial data (2.5) the diffusion equation has an exact solution. What is that exact solution? Answer in a `Maple` worksheet `diffusion2.mws`. Note that `~phys410/hw4/a2` has a "template" `diffusion2.mws` that you are encouraged to modify in answering.
Hint: Assume that the solution is separable, i.e. that $u(t,x) = f(t)g(x)$.

We now proceed to the solution of the system (2.1)–(2.4) via finite difference techniques. As usual, the first step of this process is the replacement of the continuum domain with a discrete domain (*grid*, *mesh*, or *lattice* of points), and the identification of the discrete "degrees of freedom" (discrete unknowns). Also, as usual we choose to work with a *uniform* discretization, so that the spacing, Δx, between any two spatially-adjacent grid points is a constant, as is the spacing Δt between any two temporally-adjacent grid points. In fact, we will go one step further and demand that as we vary, e.g., Δx and Δt for fixed initial data in order to investigate the convergence of our scheme, that the ratio $\Delta t/\Delta x$ be held fixed. We can then write

$$\begin{aligned} \Delta x &\equiv h & (2.6)\\ \Delta t &\equiv \lambda h & (2.7) \end{aligned}$$

where $\lambda \equiv \Delta t/\Delta x$ is often known as the *Courant number.*



With this preamble, we now replace the continuum domain (2.2) with the discrete set

$$(t^n, x_j) = (n\lambda h, (j-1)h) \quad n = 0,1,\ldots N_t \quad j = 1,2\ldots N_x \tag{2.8}$$

(see above figure) where we define the spatial mesh spacing, h, in terms of the reciprocal number of spatial grid points

$$h \equiv \frac{x_{\max} - x_{\min}}{N_x - 1} = \frac{1}{N_x - 1} \tag{2.9}$$

$$\tag{2.10}$$

and the number of time steps, N_t is defined implicitly via

$$N_t \lambda h \equiv t_{\max} \,. \tag{2.11}$$

Note that the above condition requires that λh and $t_{\max}$ be commensurate, i.e. that $\mathrm{mod}(t_{\max}, \lambda h) = 0$. Also note that we number the discrete spatial unknowns from 1 to N_x, since the spatial index, j, *will* be identified with a `Fortran` array index, and it is thus convenient to use the default `Fortran` index convention. On the other hand, our `Fortran` code will *NOT* store all time levels of data, so n will *NOT* be identified as an array index, and thus it is convenient and more natural to number the time levels from 0 so that $t^n = n\Delta t = n\lambda h$, rather than $t^n = (n-1)\Delta t$. This is particularly the case when it comes time to determine whether periodic output should be performed, which can be computed via a simple use of `mod()` when the temporal indexing starts from 0. Observe that all of these details, while they are *details*, need to be fully understood, so be sure that you *do* understand them. Finally, we note that for the sample grid displayed in the above figure, we have $N_x = 9$ and $N_t = 4$, which means that there are 5 (not 4!), discrete time steps in the complete solution.

Having defined our discrete domain (t^n, x_j), we adopt a standard finite difference notation, u_j^n, for the discrete unknowns (grid function values) where

$$u_j^n \longrightarrow u(n\Delta t, (j-1)\Delta x) = u(n\lambda h, (j-1)h) \tag{2.12}$$

and where we have used the $\longrightarrow$ instead of $=$, since we want to continue to distinguish between the finite difference solution u_j^n and the *continuum* solution $u(t^n, x_j)$. In particular, since we will (in one instance) be testing our implementation with a known solution $u(t,x)$ it is convenient to define the (absolute) solution error [2], e_j^n,

$$e_j^n \equiv u_j^n - u(t^n, x_j) \tag{2.13}$$

as well was the RMS (or ℓ_2-norm) of that quantity which we denote, $e(t^n)$

$$e(t^n) \equiv \|e_j^n\|_2 \equiv \left(\frac{\sum_{j=1}^{j=N_x} \left(e_j^n\right)^2}{N_x} \right)^{1/2} \tag{2.14}$$

We now consider the discretization of the diffusion equation (2.1)–(2.4) using the so called *Crank-Nicholson scheme*, which is based on the finite difference stencil illustrated in the above figure using the large shaded circles. The Crank-Nicholson scheme is centred at a "fictitious" half-time step, $t = t^{n+1/2} = t^n + \Delta t/2$, and at $x = x_j$, as indicated by the small black circle in the figure. Specifically, the finite difference approximation to (2.1) becomes

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = \frac{1}{2}\left[\frac{u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}}{\Delta x^2} + \frac{u_{j+1}^{n} - 2u_j^{n} + u_{j-1}^{n}}{\Delta x^2} \right], \quad j = 2, 3, \ldots, N_x - 1\,, \quad n+1 = 1, 2, \ldots N_t$$

or

$$\frac{u_j^{n+1} - u_j^n}{\lambda h} = \frac{1}{2}\left[\frac{u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}}{h^2} + \frac{u_{j+1}^{n} - 2u_j^{n} + u_{j-1}^{n}}{h^2} \right], \quad j = 2, 3, \ldots, N_x - 1\,, \quad n+1 = 1, 2, \ldots N_t \tag{2.15}$$

[2]It is convenient in this case to define the error as "computed" - "exact" in view of the definition of the Richardson expansion (2.23)

This is an $O(\Delta t^2, \Delta x^2) = O(h^2)$ approximation to the diffusion equation as can be verified by performing Taylor series expansion about the point $(t, x) = (t^{n+1/2}, x_j)$, or by inspection once one develops some intuition for such things.

Note that above formula can only be applied at the *interior* (spatial) grid points, $x_j = x_2, x_3, \ldots, x_{N_x-1}$. The boundary values at $x = x_1$ and $x = x_{N_x}$ are fixed by the obvious "discretization" of the continuum boundary conditions (2.4)

$$u_1^{n+1} = 0 \qquad (2.16)$$

$$u_{N_x}^{n+1} = 0\,. \qquad (2.17)$$

Similarly, the initial conditions (2.3) are discretized as

$$u_j^0 = u(0, x_j) = u_0(x_j) \qquad (2.18)$$

Problem 2b)

Show that, with the assumption that the values $u_j^n, j = 1, 2, \ldots N_x$ for any n are *given*, that (2.15)–(2.17) defines a *tridiagonal linear system* for the N_x advanced-time unknowns, $u_j^{n+1}, j = 1, 2, \ldots N_x$. That is, show that (2.15-2.17) can be written in the form

$$c_0 u_1^{n+1} + c_1 u_2^{n+1} = b_1 \qquad (2.19)$$

$$c_{-1} u_{j-1}^{n+1} + c_0 u_j^{n+1} + c_1 u_{j+1}^{n+1} = b_j\,, \quad j = 2, 3, \ldots, N_x - 1 \qquad (2.20)$$

$$c_{-1} u_{N_x-1}^{n+1} + c_0 u_{N_x}^{n+1} = b_{N_x} \qquad (2.21)$$

where the c_i are *constants*, but which comprise *different* sets of constants for each of the three subcases. (i.e. (i) $j = 1$, (ii) $j = 2, 3, \cdots, N_x - 1$ and (iii) $j = N_x$, and the b_j are likewise *known* quantities for all j. Answer by filling out the appropriate template in `diffusion2.mws`.

Thus, we observe that once the $t = 0$ finite difference solution has been initialized via (2.18), the solution can be advanced to $t = t^2, t^3, \ldots, t^{N_t} = t_{\max}$ via the repeated solution of the triagonal system (2.19)–(2.21).

Problem 2c)

Your task in this subproblem is to *complete* the implementation of a `Fortran` program, `diffusion2`, which you are to copy from ~`phys410/hw4/a2`; i.e. you should first execute

```
% cd; cd hw4/a2
% cp ~phys410/hw4/a2/* .
```

to ensure that *all* files in that directory are copied to your solution directory.

~`phys410/hw4/a2` contains files as follows

1. `diffusion2-pub.f`: Defines the main program for `diffusion2` as well as some utility and support routines. You should *not* change this program (except, for example, the enabling/disabling of tracing statements) during the course of solving this problem. You should, however, study the code defined therein carefully enough to fully understand the operation of the program.
2. `diffusion2-priv.f`: Defines "template" subroutines `update2` and `exact` which will ultimately be responsible for advancing the finite difference solution via the solution of the tridiagonal system (using `DGTSV`) defined above, and for computing the exact solution for the case of initial data given by (2.5). As supplied, these two routines do nothing but print a diagnostic message and return. *It is your job to finish the implementation of these routines so that*:
   (a) The overall code, `diffusion2`, *does* solve the diffusion equation.
   (b) When supplied with the initial condition (2.5), the code also calculates the error in the computed solution.
3. `Makefile`: The makefile for the application.

4. `demo`: A script that builds the application if necessary, and then puts the "template" `diffusion2` through its paces. The script also provides some clues for using `gnuplot` to visualize the output of `diffusion2`.

Refer to the source code files for complete details on the argument sequence etc. for the two routines whose implementation you must finish.

We now discuss the command line arguments to `diffusion2` as well as the output that the program produces.

Command line arguments:

`diffusion2` has usage

```
usage: diffusion2 <level> <olevel> <ostride> <tmax> <lambda> <id type> [<out type>]
```

where the various arguments have the following interpretation/semantics:

1. `<level>`: Defines the *discretization level*; i.e. the spatial mesh will have a number of points, N_x given by
$$N_x = 2^{\texttt{level}} + 1$$
2. `<olevel>`: Combined with `<level>`, defines the frequency of output. Must satisfy `<olevel>` $\le$ `<level>`. Output is performed every
$$\texttt{ofreq} = 2^{\texttt{level}-\texttt{olevel}}$$
timesteps, starting from step 0 (the initial time, $t = 0$).
3. `<ostride>`: Positive integer ≥ 1, which controls amount of output performed at output times. Specifically, grid function values at every `<ostride>`-th spatial grid point are output, so if `<ostride> = 1`, all values are output, if `ostride = 2`, every second value is output etc. Useful for "thinning" output prior to plotting via `gnuplot` for large values of `<level>` (large N_x).
4. `<tmax>`: Maximum integration time, $t_{\max}$.
5. `<lambda>`: Courant factor, $\lambda \equiv \Delta t / \Delta x$
6. `<id type>`: Controls what type of initial data is used. Implemented values are
   - `<id type> = 0`: Initialize using $u(0, x) = \sin(2\pi x)$ (Solution is known in this case, e^n_j and $e(t^n)$ can be computed.)
   - `<id type> = 1`: Initialize using $u(0, x) = \exp\left(-\left((x - 0.4)\,/0.07\right)^2\right)$ (Gaussian initial data. [3])
7. `<out type>`: Controls what type of output is performed *to standard output*. Implemented values are
   - `<out type> = 0`: Grid function values $u^n_j, j = 1,$ `<ostride>` $+ 1, 2$ `<ostride>` $+ 1, \ldots, N_x$ are output in a form suitable for subsequent plotting using `gnuplot`'s *surface plotting* facility (i.e. $f(x, y)$ is rendered as a surface/terrain where the surface/terrain height is $z = f(x, y)$). If necessary, see the `demo` script, the `gnuplot` help facility, google `gnuplot splot tutorial`, ask the instructor, ask a fellow student or `gnuplot` guru (the instructor is not one!) etc. for more details, since part of the solution to this exercise requires the generation of a "good looking" surface plot!
   - `<out type> = 1`: Provided that `<id type>` $= 0$, e^n_j and $e(t^n)$ and computed at output times and t^n and $e(t^n)$ are output, two numbers per line.

[3] Note that strictly speaking, the gaussian initial data does *not* satisfy the boundary conditions, $u(t, 0) = u(t, 1) = 0$. However the small level of violation of the boundary conditions is negligible in comparison to the level of truncation (discretization) error inherent in the solution at the resolutions at which you will be running `diffusion2`.

To summarize, there are two possible types of output from `diffusion`. The default (`<out type> = 0`) is relatively voluminous dumping of $u_j^{\tilde{n}}$ where the output times $t^{\tilde{n}}$ and output spatial points $x^{\tilde{j}}$ have been defined implicitly above (i.e. $t^{\tilde{n}}$ and $x_{\tilde{j}}$ are the subsets of t^n and x_j, respectively, at which output is performed). This output, which except for very small lattices (values of $N_x = 2^{\texttt{<level>}} + 1$), should be redirected to a file, can then be processed via `gnuplot` to produce a surface plot. The second type of output (`<out type> = 1`), is more modest in volume, *and is only valid when* `<id type> = 0`. In this case, `diffusion2` computes $e(t^{\tilde{n}})$ (the ℓ_2 norm of the solution error) and outputs $t^{\tilde{n}}$ and $e(t^{\tilde{n}})$ (two numbers per line) to standard output.

Additional Program Output

`diffusion2` also uses the `xvs` interface described in the Course Related Software page to output `.sdf` files containing the approximate solution, $u_j^{\tilde{n}}$, and, where appropriate, the exact solution, $u(t^{\tilde{n}}, x_j)$ and the solution error, $e_j^{\tilde{n}}$. Note that although the "time step filtering" *is* used for `.sdf` output (i.e. `diffusion2` only writes to `.sdf` files every `ofreq` time steps), grid function values for *all* spatial values, x_j are written at each output time since, unlike `gnuplot`'s `splot` facility, `xvs`'s rendering appearance and performance does not deteriorate as rapidly with increasing N_x. The `.sdf` files that are generated can be sent to the visualization server `xvs` for analysis (using the `sdftoxvs` command), provided you are working at the console of a machine on which `xvs` is installed (this includes all of the course machines and X-terms in the computer lab). You may find it quite useful to use `xvs` in your solution of this portion of the exercise as well as the corresponding part of the next problem. If you have a home `Linux` machine, note that you can install `xvs` and related software on your home machine. See the course Software page for more details.

Numerical Experiments

Once you've coded `update2` and `exact` properly, the fun can start, of course. As good computational scientists and finite differencers you will first want to convergence test your code, particularly since you have an exact solution with which to compute the *actual error* in the computed approximation.

First, note that a properly implemented `diffusion2` generates a near-rigorously [4] $O(h^2)$ solution, i.e. changing grid function notation to u^h so that

$$u^h(t^n, x_j) \equiv u_j^n \tag{2.22}$$

we can, following Richardson, expect that as $h \to 0$ (suppressing the time and space dependence)

$$u^h = u + h^2 e_2 + h^4 e_4 + O(h^6) \tag{2.23}$$

where, as discussed in class, the error functions $e_2 = e_2(t,x)$, $e_4 = e_4(t,x)$ etc. are h-independent, with smoothness comparable to the continuum solution $u = u(t,x)$ itself.

From (2.23) we can immediately express the error e^h (defined analogously to u^h) as

$$e^h = u^h - u = h^2 e_2 + h^4 e_4 + O(h^6) = h^2 e_2 + O(h^4) \tag{2.24}$$

and its RMS value/ℓ_2-norm

$$\|e^h\|_2(t^n) = h^2 \|e_2\|(t^n) + O(h^4)\,. \tag{2.25}$$

From this last expression we can quickly see that considering a pair of calculations with the same initial conditions, but two distinct resolutions, $2h$ and h, in a 2:1 ratio, then if we plot $4e^h(t^n)$ and $e^{2h}(t^n)$, and *if the solution IS converging as* $O(h^2)$, then the curves should be *increasingly coincident* as $h \to 0$ (up to the point where effects due to loss of precision start to dominate).

We make yet another change of notation (!), and now denote our finite difference solution as u^ℓ where ℓ is to be identified with `<level>`. Thus u^ℓ and $u^{\ell+1}$ are identical to u^{2h} and u^h respectively in the above (note that larger ℓ means *smaller* h). Similarly, we define $e^\ell(t^{\tilde{n}})$ via $e^\ell = \|u^\ell(t^{\tilde{n}}, x_j) - u(t^{\tilde{n}}, x_j)\|_2$.

[4] And I'm not going to further define this phrase.

Problem 2d)

Using data from the following invocations of `diffusion2`

```
diffusion2  6 6 1 0.25 0.5 0 1
diffusion2  7 6 1 0.25 0.5 0 1
diffusion2  8 6 1 0.25 0.5 0 1
diffusion2  9 6 1 0.25 0.5 0 1
diffusion2 10 6 1 0.25 0.5 0 1
```

demonstrate the convergence of your solution by graphing e^6, $4e^7$, $16e^8$, $64e^9$, $256e^{10}$ as a function of $t^{\tilde{n}}$ in a single plot. Save a postscript version of your plot in the file `esin.ps`.

Problem 2e) (555 CREDIT ONLY, *OPTIONAL* FOR 410 STUDENTS)

Using data from the following invocations of `diffusion2`

```
diffusion2  6 6 1 0.25 0.5 1 0
diffusion2  7 6 1 0.25 0.5 1 0
diffusion2  8 6 1 0.25 0.5 1 0
diffusion2  9 6 1 0.25 0.5 1 0
diffusion2 10 6 1 0.25 0.5 1 0
```

demonstrate the convergence of your solution, *at the single specific time* $t = 0.25$, using gaussian initial data by graphing $u^6 - u^7$, $4(u^7 - u^8)$, $16(u^8 - u^9)$ and $64(u^9 - u^{10})$ as a function of $\tilde{x}$ (where $\tilde{x}$ are the spatial coordinates for the level-6 solution) on a single plot. Save a postscript version of your plot in the file `dgaussian.ps`.

You may find the following Unix commands and instructor supplied scripts useful in answering this sub-problem: `paste`, `nth`, `nf` and `lines`. You should also observe that `gnuplot` also has extensive facilities for manipulating data.

Problem 2f)

Using data from the following invocation of `diffusion2`

```
diffusion2 8 6 4 0.25 0.5 1 0
```

create a postscript file `gaussian8.ps` that contains a `gnuplot` surface plot of the solution u_j^n

You may (or rather, you *should!*) find it convenient to write one or more scripts to run `diffusion2`, massage the output, and generate the `gnuplot` plots.

Checklist of required files: When you are done this problem you should have modified and/or created the following files

1. `diffusion2.mws`: Contains "analytic" work related to FDA solution of the diffusion equation. Note that for this problem you can view `diffusion2.mws` as merely an electronic version of your answers to the questions posed in 2a) and 2b); i.e. there is no need to show any derivation of your results (*Just the facts, ma'am!*).
2. `diffusion2-priv.f`: Defines `update2` and `exact` subroutines.
3. `esin.ps`: Illustrates second order convergence of `diffusion2` with sinusoidal initial data.
4. `gaussian8.ps`: Surface plot of level-8 `diffusion2` solution with gaussian initial data.

PHYS 555 registrants must additionally create

5. `dgaussian.ps`: Illustrates second order convergence of `diffusion2` with gaussian initial data.

Whew!!

Remember that at the core of `diffusion2` lies a *tridiagonal linear solve* and that you are saving your computations from crippling inefficiency through the use of a routine such as `DGTSV`.

Problem 3: For this problem, all file references are relative to a solution directory `~/hw4/a3`, and as in the previous problem once you have created this solution directory, you should first copy the complete contents of `~phys410/hw4/a3` to *your* solution directory prior to working on the problem.

This exercise largely parallels the previous one, except that we now modify the spatial part of the FDA approximation so that an $O(h^4)$ approximation of the second derivative, u_{xx}, is used where possible. This then results in a *pentadiagonal* linear system for the advanced-time unknowns that can be solved with the `LAPACK` routine `DGBSV`.

Many of you will have heard of the technique of Richardson extrapolation, named after the same Richardson who hired boys to do some of his computing for him. Fundamentally, Richardson extrapolation is a general method for improving the accuracy-order of a finite difference equation, or finite difference operator, based on the existence of (asymptotic) series such as (2.23). Here, you must "invent" a Richardson extrapolation procedure to generate a centred $O(h^4)$ approximation to u_{xx} from the $O(h^2)$ formula.

As you can easily verify using your implementation of `fdaeval` from Homework 2, the $O(h^2)$ approximation to u_{xx} that we derived in class for a uniform spatial mesh

$$\frac{u_{j+1} - 2u_j + u_{j-1}}{h^2} = [u_{xx}]_j + O(h^2) \tag{3.26}$$

has a leading error term given by

$$\frac{u_{j+1} - 2u_j + u_{j-1}}{h^2} = [u_{xx}]_j + h^2 \frac{1}{12} [u_{xxxx}]_j + O(h^4) \,. \tag{3.27}$$

Now observe that, we can use next-to-nearest-neighbors, rather than nearest-neighbors to generate *another* $O(h^2)$ approximation, simply by substituting $h \to 2h$ in the above:

$$\frac{u_{j+2} - 2u_j + u_{j-2}}{4h^2} = [u_{xx}]_j + 4h^2 \frac{1}{12} [u_{xxxx}]_j + O(h^4) \,. \tag{3.28}$$

Problem 3a)

Consider a general linear combination of the left and right hand sides of the two formulae (3.27) and (3.28). Show that there is one *specific* linear combination that will eliminate the $O(h^2)$ error term, thus deriving the $O(h^4)$ formulae

$$\frac{-u_{j+2} + 16u_{j+1} - 30u_j + 16u_{j-1} - u_{j-2}}{12h^2} = [u_{xx}]_j + O(h^4) \,. \tag{3.29}$$

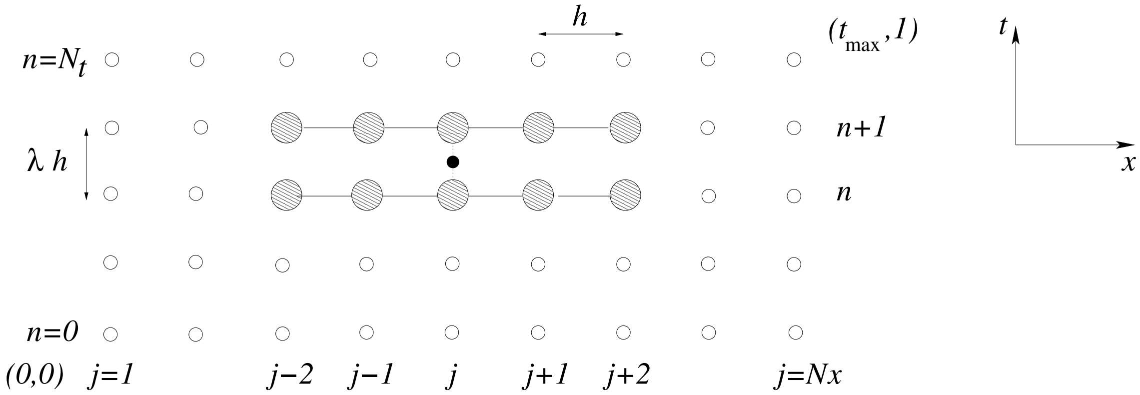
Document your calculations in the maple worksheet `diffusion4.mws`, a template of which is provided in `~phys410/hw4/a3`.

We again use a "Crank-Nicholson" scheme, but, where possible, we now use the stencil shown in the above figure. Clearly, since the stencil requires next-to-nearest neighbors on both the left and the right, as well as nearest neighbors on both sides, we can apply the formulae for $j = 3, 4, \ldots, N_x - 1, N_x - 2$. At the penultimate points $j = 2$ and $j = N_x - 1$ we will use the $O(h^2)$ formula used in the previous problem and illustrated in the first figure. Finally, we again use the Dirichlet boundary conditions to fix our solution at $j = 1$ and $j = N_x$.

Thus, our complete set of difference equations for the mixed $O(h^2, h^4)$ FDA of the diffusion equation is

$$u_1^{n+1} = 0 \tag{3.30}$$

$$\frac{u_j^{n+1} - u_j^n}{\lambda h} = \frac{1}{2}\left[\frac{u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}}{h^2} + \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{h^2}\right], \quad j = 2 \tag{3.31}$$



$$\frac{u_j^{n+1}-u_j^n}{\lambda h} = \frac{1}{2}\left[\frac{-u_{j+2}^{n+1}+16u_{j+1}^{n+1}-30u_j^{n+1}+16u_{j-1}^{n+1}-u_{j-2}^{n+1}}{12h^2}+\right.$$
$$\left.\frac{-u_{j+2}^{n}+16u_{j+1}^{n}-30u_j^{n}+16u_{j-1}^{n}-u_{j-2}^{n}}{12h^2}+\right], \quad j=3,4,\ldots N_x-2 \tag{3.32}$$
$$\frac{u_j^{n+1}-u_j^n}{\lambda h} = \frac{1}{2}\left[\frac{u_{j+1}^{n+1}-2u_j^{n+1}+u_{j-1}^{n+1}}{h^2}+\frac{u_{j+1}^{n}-2u_j^{n}+u_{j-1}^{n}}{h^2}\right], \quad j=N_x-1 \tag{3.33}$$
$$u_{N_x}^{n+1} = 0 \tag{3.34}$$

for $n+1 = 1,2,3,\ldots N_t$, which must be supplemented as before with the initial conditions

$$u_j^0 = u(0,x_j) = u_0(x_j)\,. \tag{3.35}$$

Problem 3b)

Show that, with the assumption that the values $u_j^n, j = 1,2,\ldots N_x$ for any n are *given*, that (3.30)–(3.35) define a *pentadiagonal linear system* for the N_x advanced-time unknowns, $u_j^{n+1}, j = 1,2,\ldots N_x$. That is, show that (3.30)–(3.34) can be written in the form

$$c_0u_1^{n+1}+c_1u_2^{n+1}+c_2u_3^{n+1} = b_1 \tag{3.36}$$
$$c_{-1}u_1^{n+1}c_0u_2^{n+1}+c_1u_3^{n+1}+c_2u_4^{n+1} = b_2 \tag{3.37}$$
$$c_{-2}u_{j-2}^{n+1}+c_{-1}u_{j-1}^{n+1}+c_0u_j^{n+1}+c_1u_{j+1}^{n+1}+c_2u_{j+2}^{n+1} = b_j \quad j=3,4,\ldots,N_x-2 \tag{3.38}$$
$$c_{-2}u_{N_x-3}^{n+1}+c_{-1}u_{N_x-2}^{n+1}+c_0u_{N_x-1}^{n+1}+c_1u_{N_x}^{n+1} = b_{N_x-1} \tag{3.39}$$
$$c_{-2}u_{N_x-2}^{n+1}+c_{-1}u_{N_x-1}^{n+1}+c_0u_{N_x}^{n+1} = b_{N_x} \tag{3.40}$$

where the c_i are *constants*, but which comprise *different* sets of constants for each of the five subcases. (i.e. (i) $j=1$, (ii) $j=2$, (iii) $j=3,4,\cdots,N_x-2$, (iv) $j=N_x-1$ and (v) $j=N_x$, and, again, the b_j are also known quantities. Answer by filling out the appropriate template in `diffusion4.mws`.

Thus, in complete analogy with the previous problem, once we have initialized the finite difference solution, we then advance the finite difference solution from discrete time to discrete time ($t^n \to t^{n-1}$) via the solution of a pentadiagonal system, which, as has already been mentioned, you will do using the `LAPACK` banded solver, `DGBSV`.

Problem 3c)

Your task in this subproblem is to *complete* the implementation of a `Fortran` program, `diffusion4`, which you should have already copied in template form from ~`phys410/hw4/a3`.

Again, the main program has been written for you. You need only modify the file `diffusion4-priv.f`, completing, as before the routine `update4` which advances the solution in time by setting up and solving the pentadiagonal system defined above, and the routine `exact` which can be an exact duplicate of your implementation from the previous problem.

You will find the that the usage (command line arguments) and output of `diffusion2` and `diffusion4` are identical.

Numerical Experiments

Problem 3d)

Once you have `diffusion4` completed and debugged, perform numerical experiments and generate plots as defined by the following pseudo-code

for $\lambda = 1.0, 0.1, 0.01$ **do**
  for `<level>` $= 6, 7, 8, 9, 10$ **do**
    execute `diffusion2 <level> 6 1 0.25` λ `0 1`
    execute `diffusion4 <level> 6 1 0.25` λ `0 1`
  od
  plot standard output from all 10 runs as single plot `esin-lam-<lambda>.ps`
  if $\lambda = 0.01$ **then**
    plot standard output from all 5 `diffusion4` runs as single plot `esin-lam-0.01-4.ps`
  fi
od

Write a short synopsis of your observations concerning the four plots produced in `README`. Do the results surprise you at all? What can you say about the relative superiority of `diffusion2` and `diffusion4`? Again, answer in `README`

Checklist of required files: When you are done this problem you should have modified and/or created the following files

1. `diffusion4.mws`: Contains "derivation" of $O(h^4)$ approximation to u_{xx}, as well as expressions (no derivation required) for the pentadiagonal system.
2. `diffusion4-priv.f`: Defines `update4` and `exact` subroutines.
3. `esin-lam-1.0.ps`: Comparison of `diffusion[24]` errors for $\lambda = 1.0$
4. `esin-lam-0.1.ps`: Comparison of `diffusion[24]` errors for $\lambda = 0.1$
5. `esin-lam-0.01.ps`: Comparison of `diffusion[24]` errors for $\lambda = 0.01$
6. `esin-lam-0.01-4.ps`: Illustration of `diffusion4` error convergence for $\lambda = 0.01$
7. `README`: Contains discussion of your observations re the relative convergence of `diffusion2` and `diffusion4`.

... and that's it!!

Remember that at the core of `diffusion4` lies a *banded linear solve* and that you are saving your computations from crippling inefficiency through the use of a routine such as `DGBSV`.