

Numerical Computing Practical 1

Stuart Golodetz

March 5, 2007

1. The matrices (for $J = 10, \theta = 1, \nu = 2$) are:

$$D = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -2 & 5 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2 & 5 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 5 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2 & 5 & -2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 5 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -2 & 5 & -2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -2 & 5 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 5 & -2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 5 & -2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$B = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Since the θ -scheme is an implicit method, we get lots of equations relating the various U_j^{n+1} 's to the various U_j^n 's. Specifically, we get one equation for each value of j , i.e. $J + 1$ equations. Writing this in matrix form, we can say

$$A(U_1^{n+1}, U_2^{n+1}, \dots, U_{J+1}^{n+1})^T = B(U_1^n, U_2^n, \dots, U_{J+1}^n)^T$$

for every value of n and some $(J + 1) \times (J + 1)$ matrices A and B.

Clearly the matrices we need are the ones the code is calculating. In particular, we note that there are two different types of equation - those at the boundary, and normal ones. At the boundary, we have:

$$\begin{aligned} U_1^{n+1} &= 0 \\ U_{J+1}^{n+1} &= 0 \end{aligned}$$

These correspond to the first and last rows of A and B, respectively. For the normal case, we have the scheme given, namely

$$(1 - \theta\nu\delta_x^2)U_j^{n+1} = (1 + (1 - \theta)\nu\delta_x^2)U_j^n.$$

This is what each of the other rows of A and B corresponds to. Each row relates certain U_j^{n+1} 's to certain U_j^n 's. Considering the left-hand-side, we note that when we construct A, we start with a diagonal matrix of ones (corresponding to the $1 \times U_j^{n+1}$ on the left-hand-side of the scheme) and subtract $\theta\nu D$ from it (corresponding to the $-\theta\nu\delta_x^2 U_j^{n+1}$ term). We do the same sort of thing for the other side of the equation to get B.

Note that D has zero entries in its first and last rows - that's because those rows correspond to the boundary cases.

2. For stability, the theory requires either that

$$\theta \geq \frac{1}{2}$$

or that

$$\nu \leq \frac{1}{2(1 - 2\theta)}$$

$\theta = 0$
According to theory, we need $\nu \leq 1/2$. Accordingly, we'd expect the largest value (to 2DP) for which the scheme is stable to be 0.50. If we try 0.51, we'd expect it not to work.
Result: As expected, it's stable for 0.50 and unstable for 0.51.

$\theta = 0.2$
Here we need $\nu \leq 1/(2(1 - 0.4)) = 1/1.2 = 0.83$ (2DP).
Result: As expected, it's stable for 0.83 and unstable for 0.84.

$\theta = 0.4$
Here we need $\nu \leq 1/(2(1 - 0.8)) = 1/0.4 = 2.5$.
Result: It's quite hard to see that it doesn't work for 2.51, but if you make the window larger then it becomes clearer. If you start using 2.52, 2.53, etc., it starts to become really obvious. It's not quite as clear as for smaller values of θ , though.

$\theta = 0.6$
We'd expect this to be stable for any value of ν since $\theta \geq 1/2$.
Result: I tried it with $\nu = 50.0$ and it was stable, though it looked a bit funny in the middle because the step was ridiculously large. With smaller values of ν , like 5.0, it was fine.

$\theta = 0.8$
We'd expect this to be stable for any value of ν for the same reason.
Result: I tried it with $\nu = 100.0$ and it was fine. With $\nu = 500.0$, it started to look funny in the middle, but was nevertheless stable.

$\theta = 1.0$
Once again, we'd expect this to be stable for any value of ν .
Result: I tried it with $\nu = 500.0$ and it was fine.

These results were all consistent with the Fourier stability theory given in lectures, as noted above. The reason it sometimes looked a bit funny in the middle for extremely large values of ν was to do with an accuracy problem, not a stability one.

3. The analytic solution is:

$$u(x, t) = e^{-\pi^2 t} \sin(\pi x)$$

The new version of the program is:

```

%
% Numerical Computing -- practical 1
%

% parameters
nu    = 100;
theta = 1.0;
J      = 100;           % number of points
dx     = 1/J;          % space step
dt     = nu*dx^2;      % time step

%
% difference operator matrices
%

col = ones(J-1,1);    % column vector of ones

D =  diag([ col; 0],-1) ...
    - 2*diag([0; col; 0], 0) ...
    +  diag([0; col ], 1); % second difference matrix
A =  diag([1; col; 1], 0) - theta *nu*D; % LHS matrix
B =  diag([0; col; 0], 0) + (1-theta)*nu*D; % RHS matrix

% SMG
%A = sparse(A);
%B = sparse(B);

%
% initialisation
%

x = (0:dx:1)'; % grid points
%u = min(2*x,2-2*x); % initial data

u = sin(pi*x); % initial data for Question 3

%
% main solver loop
%

plot(x,u); % plot initial data
pause(0); % force it to plot now
hold on; % keep plot
axis manual; % keep axes scales fixed
tic % initialise timer

oldt = 0;
for t = dt:dt:0.5
    u = A \ (B*u); % update solution

    % calculate the value of the analytic solution
    v = exp(-pi^2*t)*sin(pi*x);

    if mod(t,0.05) < dt % at intervals of 0.05
        plot(x,u,'b',x,v,'--r'); pause(0); % plot current solution
    end

    %if t == 0.2
    if oldt < 0.2 & t >= 0.2
        % calculate the r.m.s. difference
        w = u - v;
        norm(w)/sqrt(length(w))
    end
    oldt = t;
end

disp(sprintf('Execution time: %d\n',toc)) % report execution time

hold off

```

As derived in lectures, the truncation error for the θ -method is given by:

$$\tau_j^n = \frac{1}{2}(1 - 2\theta)\Delta t \frac{\partial^2 u}{\partial t^2} + \frac{1}{6}(1 - 3\theta + 3\theta^2)\Delta t^2 \frac{\partial^3 u}{\partial t^3} - \frac{1}{12}\Delta x^2 \frac{\partial^4 u}{\partial x^4} + \dots$$

In the examples below, we'll plug the values of θ and ν in and see what we're expecting to get, then compare that with our actual results.

For $\theta = 0, \nu = 0.5$, the r.m.s. differences are:

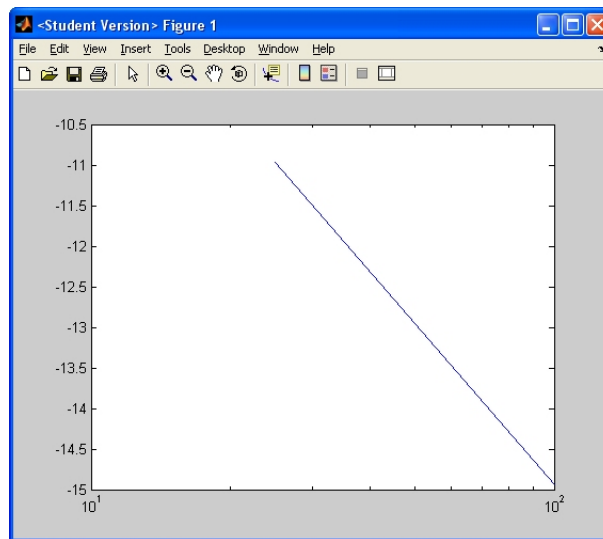
J	R.M.S. Difference
25	5.0119×10^{-4}
50	1.2637×10^{-4}
100	3.1738×10^{-5}

```
>> x = [25, 50, 100];
>> y = log2([5.0119e-04, 1.2637e-04, 3.1738e-05])

y =

-10.9624 -12.9501 -14.9434

>> semilogx(x, y)
```



When $\theta = 0$ and $\nu = 0.5$ (thus $\Delta t = 0.5\Delta x^2$), we have:

$$\tau_j^n = \frac{1}{4}\Delta x^2 \frac{\partial^2 u}{\partial t^2} + \frac{1}{24}\Delta x^2 \frac{\partial^3 u}{\partial t^3} - \frac{1}{12}\Delta x^2 \frac{\partial^4 u}{\partial x^4} + \dots = O(\Delta x^2)$$

Thus we expect the spatial convergence to be second-order. When doubling the number of grid points, therefore, we'd expect the error to go down by roughly a factor of 4, and hence $\log_2(\text{error})$ to go down by about 2. This is what we observe in practice, as the above results show.

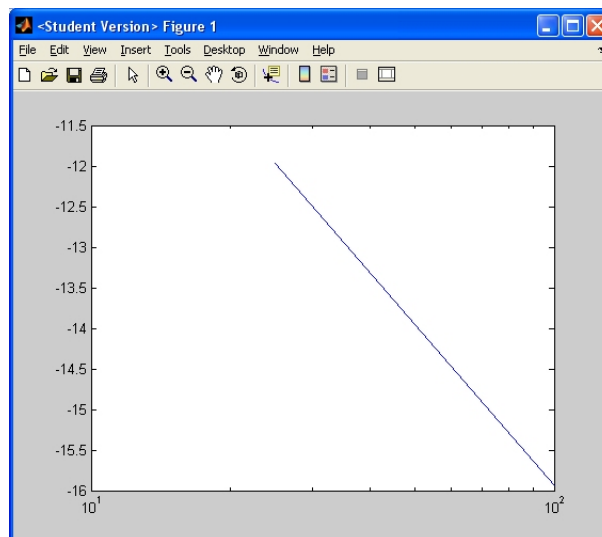
For $\theta = 0.5$, $\nu = 0.5$, the r.m.s. differences are:

J	R.M.S. Difference
25	2.4940×10^{-4}
50	6.3108×10^{-5}
100	1.5864×10^{-5}

```
>> x = [25, 50, 100];
>> y = log2([2.4940e-04, 6.3108e-05, 1.5864e-05])
```

```
y =
-11.9693 -13.9518 -15.9439
```

```
>> semilogx(x, y)
```



When $\theta = 0.5$ and $\nu = 0.5$ (thus $\Delta t = 0.5\Delta x^2$), we have:

$$\tau_j^n = \frac{1}{96}\Delta x^2 \frac{\partial^3 u}{\partial t^3} - \frac{1}{12}\Delta x^2 \frac{\partial^4 u}{\partial x^4} + \dots = O(\Delta x^2)$$

Thus we expect the spatial convergence to be second-order. When doubling the number of grid points, therefore, we'd again expect the error to go down by roughly a factor of 4, and hence $\log_2(\text{error})$ to go down by about 2. This is once again the case in practice.

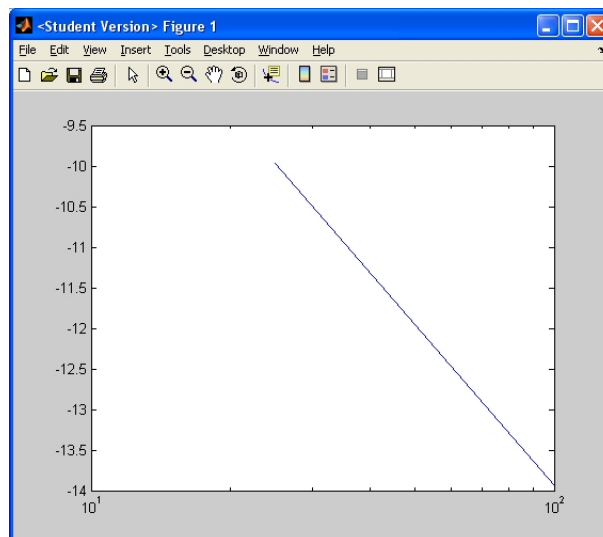
For $\theta = 1, \nu = 0.5$, the r.m.s. differences are:

J	R.M.S. Difference
25	9.9991×10^{-4}
50	2.5258×10^{-4}
100	6.3467×10^{-5}

```
>> x = [25, 50, 100];
>> y = log2([9.9991e-04, 2.5258e-04, 6.3467e-05])
```

```
y =
-9.9659 -11.9510 -13.9436
```

```
>> semilogx(x, y)
```



When $\theta = 1$ and $\nu = 0.5$ (thus $\Delta t = 0.5\Delta x^2$), we have:

$$\tau_j^n = -\frac{1}{4}\Delta x^2 \frac{\partial^2 u}{\partial t^2} + \frac{1}{24}\Delta x^2 \frac{\partial^3 u}{\partial t^3} - \frac{1}{12}\Delta x^2 \frac{\partial^4 u}{\partial x^4} + \dots = O(\Delta x^2)$$

Thus we expect the spatial convergence to be second-order. When doubling the number of grid points, therefore, we'd again expect the error to go down by roughly a factor of 4, and hence $\log_2(\text{error})$ to go down by about 2. This is once again the case in practice.

For $\theta = 0.5$, $\nu = 1/\Delta x$ ($= J$):

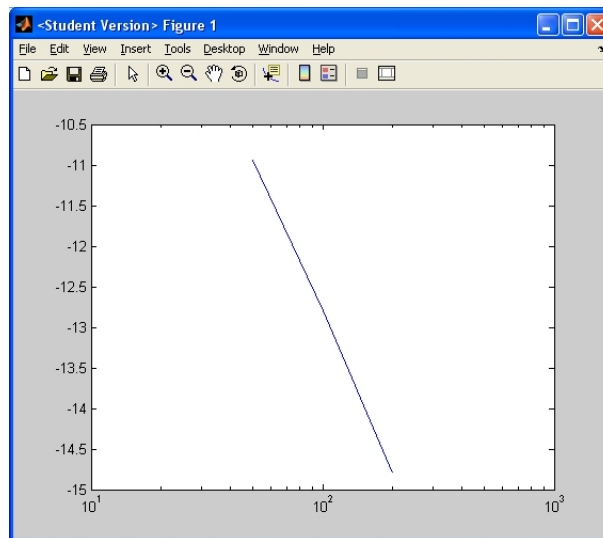
J	R.M.S. Difference
50	5.0697×10^{-4}
100	1.4083×10^{-4}
200	3.5278×10^{-5}

```
>> x = [50, 100, 200];  
>> y = log2([5.0697e-04, 1.4083e-04, 3.5278e-05])
```

```
y =
```

```
-10.9458 -12.7938 -14.7909
```

```
>> semilogx(x, y)
```



When $\theta = 0.5$ and $\nu = 1/\Delta x$ (thus $\Delta t = \Delta x$), we have:

$$\tau_j^n = \frac{1}{24} \Delta x^2 \frac{\partial^3 u}{\partial t^3} - \frac{1}{12} \Delta x^2 \frac{\partial^4 u}{\partial x^4} + \dots = O(\Delta x^2)$$

Thus we expect the spatial convergence to be second-order. When doubling the number of grid points, therefore, we'd again expect the error to go down by roughly a factor of 4, and hence $\log_2(\text{error})$ to go down by about 2. This is almost the case in practice, but this time it's not quite as close as before because (particularly for the $J = 50$ case) we're not getting the value at exactly $t = 0.2$ each time (see code). If we were getting the value accurately, it would be much closer.

For $\theta = 1, \nu = 1/\Delta x (= J)$:

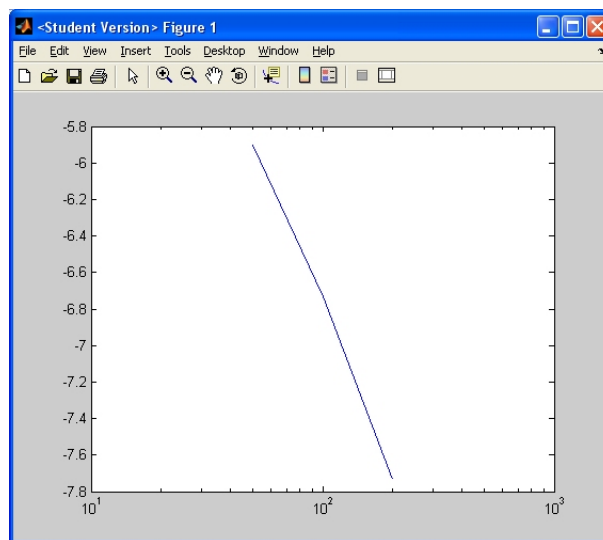
J	R.M.S. Difference
50	0.0167
100	0.0094
200	0.0047

```
>> x = [50, 100, 200];
>> y = log2([0.0167, 0.0094, 0.0047])
```

```
y =
```

```
-5.9040    -6.7331    -7.7331
```

```
>> semilogx(x, y)
```



When $\theta = 1$ and $\nu = 1/\Delta x$ (thus $\Delta t = \Delta x$), we have:

$$\tau_j^n = -\frac{1}{2}\Delta x \frac{\partial^2 u}{\partial t^2} + \frac{1}{6}\Delta x^2 \frac{\partial^3 u}{\partial t^3} - \frac{1}{12}\Delta x^2 \frac{\partial^4 u}{\partial x^4} + \dots = O(\Delta x)$$

Thus we expect the spatial convergence to be first-order. (In fact, when $\Delta t = \Delta x$, we always expect the spatial convergence to be first-order except for the special case when $\theta = 0.5$, when the term in Δt (alternatively, in Δx) cancels. This is presumably why the previous example was chosen!) When doubling the number of grid points, therefore, we'd expect the error to go down by roughly a factor of 2, and hence $\log_2(\text{error})$ to go down by about 1. Again, this isn't quite the case in practice because of the way we're obtaining the value at $t = 0.2$ when $J = 50$, but it's close enough.

4.

J	Before	After	Ratio
100	1.488636	1.449758	0.9739
200	1.814365	1.583170	0.8726
500	11.60640	1.501286	0.1293
1000	133.0422	3.593943	0.0270

Explanation:

I'm not sure I can explain the values themselves, but I can comment on the improvement when using the sparse matrix representation. Essentially, the point is that we're using $O(J^2)$ space when storing A and B normally, but only $O(J)$ space when storing them using the sparse representation, since they only have three non-zero diagonals - we're using roughly $3J$ space to store them instead of J^2 . The difference between J^2 and $3J$ gets bigger the larger J gets, so we see more of an improvement for large values of J than we do for small ones. When $J = 1000$, the difference is absolutely massive. (Specifically, you can fit a whole cup of coffee in while waiting for the normal version, whereas you barely have time to leave your seat with the sparse representation. This may or may not be a good thing...)