9 Solving Tridiagonal Systems

Matrices of the form

$$A = \begin{bmatrix} a & b & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ b & a & b & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & b & a & b & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & b & a & b \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & b & a \end{bmatrix}$$

occur frequently in solving partial differential equations. They are *tridiagonal* matrices with a on the main diagonal and b on the diagonals immediately above and below the main diagonal. All other elements are zero.

Let $a = 1.2$, $b = -0.1$. Consider the system

$$A\mathbf{x} = \mathbf{d}$$

where $\mathbf{d}$ is a *random* vector (of appropriate length).

1.1 Solve this system with the MATLAB backslash operator when A is a matrix of size 100×100, 1000×1000 and 5000×5000. Measure the (cpu) time it takes MATLAB to solve each of these cases (if you don't know how to do this, check `doc cputime`). Footnote: When timing a routine you should run it a number of times and then average the time taken to get a reliable estimate.

SOLUTION:

```
function t= TDtimer(n,a,b,m)
% n is the size of A, a is the diagonal element, b is the
% off-diagonal element and m is the number of runs
% TDtimer computes the averarge time to compute solution over m runs.

A = a*eye(n) + b*(diag(ones(n-1,1),-1) + diag(ones(n-1,1),1));
d= rand(n,1);
tt=cputime;
for ii=1:m
    A\d;
end
t=(cputime-tt)/m;

end
```

1.2 MATLAB has a `sparse` function (see `doc sparse`). The use of this function reduces storage (only non-zero elements are stored). Moreover backslash will use specialised routines to exploit the *structure* of A. Repeat your timings when A is "sparsified".

SOLUTION:

```
function t= STDtimer(n,a,b,m)
% n is the size of A, a is the diagonal element, b is the
% off-diagonal element and m is the number of runs
% TDtimer computes the averarge time to compute solution over m runs.

A = sparse(a*eye(n) + b*(diag(ones(n-1,1),-1) + diag(ones(n-1,1),1)));
d= rand(n,1);
tt=cputime;
for ii=1:m
    A\d;
end
t=(cputime-tt)/m;

end
```

Systems of this type arise when the heat equation

$$u_t = u_{xx}$$

is solved using backward central differences. In that case $a = 1 + 2r$ and $b = -r$ where

$$r = \frac{\Delta t}{(\Delta x)^2}.$$

Δt and Δx are the time increment and spacing in the spatial grid respectively. The values of the temperature at time $t = (k+1)\,\Delta t$ is then given by

$$A\,\mathbf{u}_{k+1} = \mathbf{u}_k.$$

The moral of this example is that **any structure that A possesses should be exploited by the algorithm chosen to solve the system.**