CS319: Scientific Computing (with MATLAB)

Direct and Iterative Solvers (draft)

Niall Madden

Week 10, **9am**, and **4pm**, 15 March 2023

Useful reading:

- Learning MATLAB, Sections 2.5 (brief overview), 7.1 and 7.2
  https://doi-org.nuigalway.idm.oclc.org/10.1137/1.9780898717662
- The MATLAB Guide, Chapters 9 (Linear Algebra) and 15 (Sparse Matrices) of
  https://doi-org.nuigalway.idm.oclc.org/10.1137/1.9781611974669

This week...

1. 1. Projects (again)

2. 2: Linear Solvers in MATLAB
   - An example problem

3. 3: Direct solvers

4. 4. Iterative Solvers
   - minres

1. Projects (again)

The slides for this section are at https://www.niallmadden.ie/2223-CS319/2223-CS319-Projects.pdf

2: Linear Solvers in MATLAB

(This slide is from Week 9 notes). In this section, we study the details of solving linear systems of equations in MATLAB.
Solvers can be classified as one of two types:

1. **Direct solvers**, which preform a fixed number of steps and give back the true answer (or, as close as possible, given that we have finite precision). Gaussian Elimination is the most famous example of this.
2. **Iterative solvers**, which take an initial guess and repeatedly try to improve it, until some tolerance is reached, or a maximum number of iterations is reached. The Jacobi method of Lab 5 is an example.

Both methods have their advantages and disadvantages:

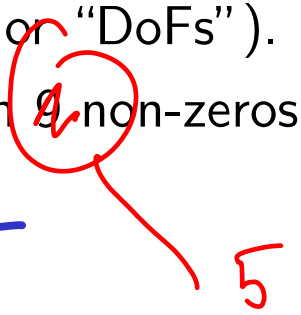
See last week's notes.

To tests some methods, we need a good test problem. We need the problem to be

- Scalable: we can make up a version of any size we choose. In particular, we should be able to make it large.
- Not accidentally simple. For example, solving $Ax = b$ can be much easier when A is a tri-diagonal matrix, than when not.
- Sparse: most entries of A are zero.

For each matrix we choose some number n,

- The matrix will be $n^2 \times n^2$ (we usually say that there are n^2 "degrees of freedom", or "DoFs").
- It will be **banded**, with 9 non-zeros per row.
- The **band-width** is n.



diagonal matrix of 1's (Identity matrix!)

MakeTestProblem.m

```
function [A,b]=MakeTestProblem(n)
A1 = sparse(1:n, 1:n, 1) ...
   + sparse(2:n, 1:n-1, -0.5, n,n) ...
   + sparse(1:n-1, 2:n, 1/4, n, n);
A = kron(A1,speye(n)) + kron(speye(n),A1) ;
x = ones(length(A),1);
b = A*x;
```

Line 2: sparse(I, J, v) returns a matrix where the non-zeros are in entries (I(1), J(1)), (I(2), J(2)) ... (I(NNZ), J(NNZ)) and all have value v (ie I, J are integer vectors of same length, v is a scalar).

sparse(I, J, v) will have max(I) rows and max(J) columns.

MakeTestProblem.m

```
function [A,b]=MakeTestProblem(n)
A1 = sparse(1:n, 1:n, 1) ...
   + sparse(2:n, 1:n-1, -0.5, n,n) ...
   + sparse(1:n-1, 2:n, 1/4, n, n);
A = kron(A1,speye(n)) + kron(speye(n),A1) ;
x = ones(length(A),1);
b = A*x;
```

sets A1(1,2) = 1/4

A1(2,3) = 1/4

sets A1(2,1) = -0.5, A1(3,2) = -0.5, etc

$$\begin{pmatrix} 1 & 1/4 & & & \\ -0.5 & 1 & 1/4 & & \\ & -0.5 & 1 & 1/4 & \\ & & -0.5 & 1 & \ddots \\ & & & \ddots & \ddots \end{pmatrix}$$

So A1 is a tridiagonal matrix.

MakeTestProblem.m

```
function [A,b]=MakeTestProblem(n)
A1 = sparse(1:n, 1:n, 1) ...
   + sparse(2:n, 1:n-1, -0.5, n,n) ...
   + sparse(1:n-1, 2:n, 1/4, n, n);
A = kron(A1,speye(n)) + kron(speye(n),A1) ;
x = ones(length(A),1);
b = A*x;
```

kron(A,B) computes the Kronecker product of A & B. That is, if A & B are both n×n matrices, the output is $n^2 \times n^2$ and of the form

e.g., n = 3

$$\begin{pmatrix} A(1,1)B & A(1,2)B & A(1,3)B \\ A(2,1)B & A(2,2)B & A(2,3)B \\ A(3,1)B & A(3,2)B & A(3,3)B \end{pmatrix}$$

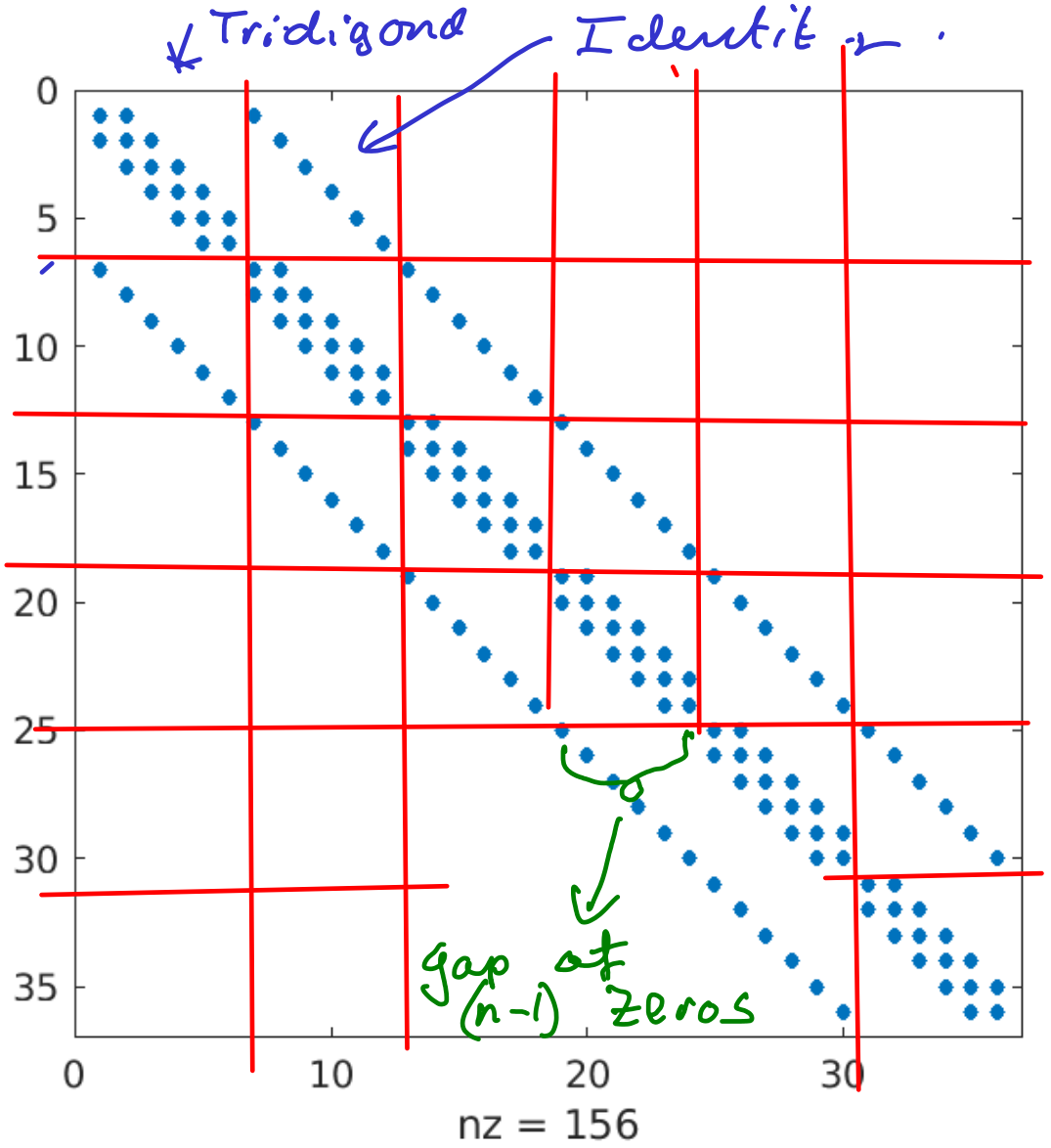
← Each is a 3×3 block.

MakeTestProblem.m

```
function [A,b]=MakeTestProblem(n)
A1 = sparse(1:n, 1:n, 1) ...
   + sparse(2:n, 1:n-1, -0.5, n,n) ...
   + sparse(1:n-1, 2:n, 1/4, n, n);
A = kron(A1,speye(n)) + kron(speye(n),A1) ;
x = ones(length(A),1);
b = A*x;
```

speye( ) = sparse identity matrix.

$$\begin{pmatrix} A_1(1,1)I & A_1(1,2)I & 0 \\ A_1(2,1)I & A_1(2,2)I & A_1(2,3)I \\ 0 & A_1(3,2)I & A_1(3,3)I \end{pmatrix} + \begin{pmatrix} A_1 & 0 & 0 \\ 0 & A_1 & 0 \\ 0 & 0 & A_1 \end{pmatrix}$$


Tridigonal
Identity
gap of (n-1) zeros
0
5
10
15
20
25
30
35
0
10
20
30
nz = 156

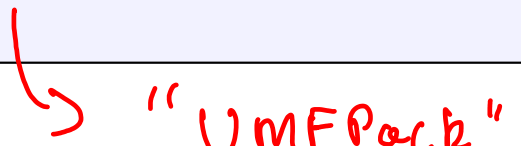
3: Direct solvers

The simplest way to solve a linear system in MATLAB is with the backslash operator:

```
>> x=A\b
```

To find out what is *really* happening we can turn on some diagnostics:

```
>> spparms('spumoni', 2)
>> x=A\b
sp\: bandwidth = 6+1+6.
sp\: is A diagonal? no.
sp\: is band density (0.366197) > bandden (0.500000) to try
     banded solver? no.
sp\: is A triangular? no.
sp\: is A morally triangular? no.
sp\: is A a candidate for Cholesky (symmetric, real
    positive or negative diagonal)? no.
sp\: use Unsymmetric MultiFrontal PACKagewith automatic
    reordering.
```



3: Direct solvers

In that example, we use the (somewhat obscure) function `spparms()` which sets parameters for sparse matrix routines.
The `spumoni` option turns on monitoring.
If we set it to Level 2, we get more output (usually way too much). But with some effort, we can use the data to test the algorithms efficiency.

Unfortunately, it is not easy to record this data in a systematic way. But we will look at the solve times...

Finished here at 10am