Examples of solving symmetric linear systems

We first experiment with a symmetric positive definite A with zeros in random locations:

```
>> rng(0);
>> B = sprandsym(4000, 0.001);
>> A = speye(size(B)) - 0.1*B;
>> tic; R = chol(full(A)); toc
Elapsed time is 1.059630 seconds.
>> tic; R = chol(A); toc
Elapsed time is 0.910553 seconds.
>> nnz(R)
ans =
     2360808
>> nnz(A)
ans =
       19984
```

Reordering entries to reduce bandwidth

The instruction `symrcm` reorders the entries of A according to a (sort of) BFS on its adjacency graph. This often reduces the bandwidth and the number of nonzeros in `chol(A)`.

```
>> p = symrcm(A);
>> tic; R = chol(A(p, p)); toc
Elapsed time is 0.374971 seconds.
>> nnz(R)
ans =
     1576535
>> spy(R)
```

All these factorizations can be used to solve linear systems, e.g.,

```
>> x = R \ (R' \ y(p));
>> x(p) = x; %inverse permutation
>> norm(A*x-y)
ans =
   7.8569e-14
```

Iterative methods

The function pcg solves a system with conjugate gradient.

```
>> tic;[x, ~, ~, ~, resvec] = pcg(A, y);toc
Elapsed time is 0.014492 seconds.
>> norm(A*x-y)
ans =
   6.1736e-05
>> semilogy(resvec)
```

A is a reasonably well-conditioned matrix, and CG converges fast on it.

```
>> ev = eig(full(A)); plot(real(ev), imag(ev), 'x');
```

Don't use `inv`

Direct inversion with `inv` is not competitive by any metric.

```
>> tic; P = inv(A); toc
Elapsed time is 2.178261 seconds.
>> tic; x = P*y; toc
Elapsed time is 0.027226 seconds.
>> norm(A*x - y)
ans =
   1.4211e-13
```

Drawbacks:

- `inv(A)` is typically very dense: high cost and storage.
- Even for ignoring sparsity, computing `inv(A)` is more expensive than solving a linear system: it's basically equivalent to solving m linear systems $A\mathbf{x}_j = \mathbf{e}_j$ for $j = 1, 2, \ldots, m$.
- . . . and it's less accurate (basically as the least accurate of all $A\mathbf{x}_j = \mathbf{e}_j$)

A different example

This 'thin-band' matrix comes from discretization of a differential equation; it is more suitable to direct solvers than iterative ones.

```
>> A = delsq(numgrid('S',50));
>> size(A)
ans =
        2304 2304
>> spy(A)
>> b = randn(length(A), 1);
>> tic; R = chol(A); toc;
Elapsed time is 0.028630 seconds.
>> pcg(A, b, 1e-8, 100);
pcg stopped at iteration 100 without converging to the desi
because the maximum number of iterations was reached.
The iterate returned (number 100) has relative residual 8.1
>> [x, ~, ~, ~, resvec] = pcg(A, b, 1e-8, 100); semilogy(re
```

Preconditioning

A powerful idea: preconditioning. Since the performance of CG and GMRES on $A\mathbf{x} = \mathbf{y}$ depends a lot on where the eigenvalues of A are located, we may replace $A\mathbf{x} = \mathbf{y}$ with $PA\mathbf{x} = P\mathbf{y}$ to try to 'improve' its eigenvalues.

What is the ideal P?

Two extreme choices:

- $P = I$: trivial to invert, but does not change eigenvalue location.
- $P = A^{-1}$: changes eigenvalue location to all=1 (perfect!), but it takes too much work to compute: even more than solving a linear system! (Actually, if I know A^{-1}, why am I even using an iterative method?)

Idea: drop some coefficients of A to make it easier to invert. For instance, if A has small off-diagonal entries, we can invert $\mathrm{diag}(A)$.

Preconditioning: practical choices

Some intermediate choices:

- $P = \text{diag}(A)^{-1}$: cheap to compute, but might be ineffective. Use it if A is close to a diagonal matrix.
- $U = \text{triu}(A)^{-1}$: upper triangle of A. Might be effective if A is almost triangular.
- Better preconditioners are obtained with so-called 'incomplete LU': start by computing $A = LU$, but cheat to obtain more zeros: whenever an element is small-ish, replace it with 0!
  `[L,U] = ilu(A)`.
  Then, $A \approx LU$, and hence $PA = (U^{-1}L^{-1})A \approx I$.

  Do not compute P explicitly! Rather, use Arnoldi with a black-box function:

```
[L, U] = ilu(A);
f = @(v) U \ (L \ (A*v))
x = gmres(f, U \ (L \ y))
```

Matlab's `gmres` has syntax to specify a preconditioner (matrix or function) in optional arguments.

Symmetric preconditioners

When $A = A^T$, there is another issue: even with a symmetric P, the matrix PA is not going to be symmetric.

Idea: apply a matrix on both sides of A, getting an equivalent linear system:

$$PAP^T(P^{-T}x) = Pb.$$

Notation: P^{-T} stands for $(P^{-1})^T = (P^T)^{-1}$.
This time, we want PAP^T to be close to the identity matrix, or at least "with favorable eigenvalues".
Example: $P = \text{diag}(A_{11}^{-1/2}, A_{22}^{-1/2}, \ldots, A_{nn}^{-1/2})$, which works well if A is close to a diagonal matrix.

Incomplete Cholesky

Similarly to incomplete LU, we can compute an incomplete Cholesky factorization: `ichol(A)` computes L such that $LL^T \approx A$, and L is sparse (by arbitrarily setting certain entries to zero liberally during the factorization).

If we take $P = L^{-1}$, then PAP^T is more well-conditioned than A:

```
>> L = ichol(A);
>> cond(A)
ans =
   1.4136e+03
>> ev = cond(L \ A / L')
ev =
   2.0379e+02
```

```
>> matvec = @(v) L \ (A*(L' \ v))
matvec =
  function_handle with value:
    @(v)L\(A*(L'\v))
>> x = L' \ pcg(matvec, L \ y, 1e-8, 100);
pcg converged at iteration 51 to a solution
with relative residual 7e-09.
>> norm(A*x - y)
ans =
   2.9117e-07
```

Note that the residual used for stopping is the residual of the preconditioned system $\|L^{-1}AL^{-T}(L^T\mathbf{x}) - L^{-1}\mathbf{y}\|$, which is different from that of $\|A\mathbf{x} - \mathbf{y}\|$.

However, we can still compute the residual $\|A\tilde{\mathbf{x}} - \mathbf{y}\|$ at the end to figure out how close the computed $\tilde{\mathbf{x}}$ is to the true solution:

$$\frac{\|\tilde{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|} \leq \kappa(A)\frac{\|A\tilde{\mathbf{x}} - \mathbf{y}\|}{\|\mathbf{y}\|}.$$

Preconditioners are a dark art

When solving huge linear systems ($m \approx$ millions), finding a good preconditioner can improve performance dramatically.

There are many more techniques than dropping zeros in LU factorizations.

Good preconditioners often come from the structure of the problem: solving the same differential equation problem on coarser grids, considering a simpler subgraph of the graph such as a spanning tree, etc.

Randomization is another technique that is becoming popular recently: for instance, get simpler problems by randomly sampling unknowns/equations.

We don't see much here.

Wrap-up

In this part of the course, we have seen the 'simple' problems in linear algebra:

- Dense least-squares problems:
  - Normal equations
  - QR
  - SVD
  - Accuracy issues
  - Regularization
- Sparse linear systems:
  - Sparse LU / LDL / Cholesky (depending on symmetry/definiteness)
  - GMRES / MINRES / CG (depending on symmetry/definiteness)
  - Preconditioning