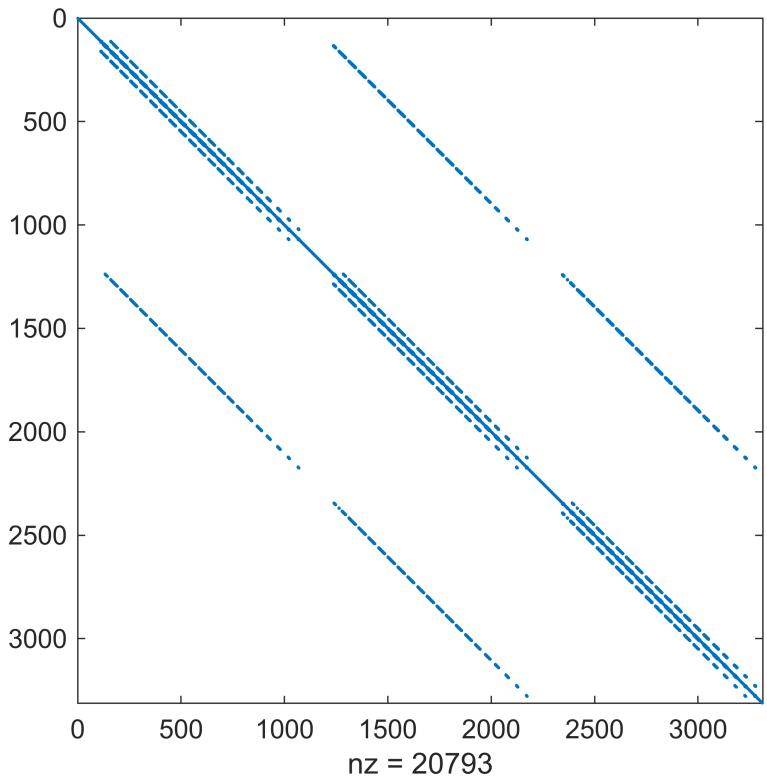
```
clear
rng('default')
load sherman5
figure, spy(A) % the "spy plot" exhibits the locations of the nonzero entries
```



Same idea as always, start with a random vector.

```
b = rand(n,1)
```

```
b = 3312×1
    0.8147
    0.9058
    0.1270
    0.9134
    0.6324
    0.0975
    0.2785
    0.5469
    0.9575
    0.9649
      ⋮
```

```
beta = norm(b)
```

```
beta =
33.3475
```

Let's compare the naive basis $b, Ab, A^2b, \ldots$ with the Arnoldi method.

```
m = 80
```

```
m =
80
```

Naive basis first.

```
K(:,1) = b / beta;
for j = 1:m-1
    w = A * K(:,j);
    K(:,j+1) = w / norm(w);
end
```

Now Arnoldi basis.

```
V(:,1) = b / beta;
for j = 1:m

    % Compute next vector
    w = A * V(:,j);

    % Orthogonalise against previous columns (MGS)
    for i = 1:j
        H(i,j) = dot(w, V(:,i));
        w = w - H(i,j) * V(:,i);
    end

    % Normalise
    H(j+1, j) = norm(w);

    % Record the new vector in V
    V(:, j+1) = w / H(j+1, j);

end
Vm = V(:, 1:m); Hm = H(1:m, 1:m);
```

The difference in the condition number of the two matrices couldn't be starker.

```
cond(Vm), cond(K)
```

```
ans =
1.0000
ans =
2.1460e+17
```

But wait, what if we orthonormalise K using Gram-Schmidt (or Householder, or whatever).

```
[Q,R] = qr(K, "econ");
```

Look at the beautiful conditioning of Q now.

```
cond(Q)
```

```
ans =
1.0000
```

But there's no free lunch here: the ill-conditioning has just migrated to R.

```
cond(R)
```

```
ans =
2.1460e+17
```

Ah, but what if we formulate our calculations to never actually use R? Here's H_m computed directly from the projection $H_m = Q^* A Q$.

```
H2 = Q' * A * Q
```

```
H2 = 80×80
  -14.1240  -37.9523    2.8775    1.4543   -0.1684   -2.9462   22.1405   13.0381 ···
  134.8324  513.2181  -14.1914  167.2011   19.7124  -42.3230 -182.3275 -158.7274
    0.0000 -237.9963  195.8816 -329.3283  -62.5824  191.4550  265.2109  120.4547
    0.0000    0.0000 -136.4856  116.0191 -161.4556  -29.8726  -94.9162  -61.1546
    0.0000    0.0000         0 -166.0327  265.5661  202.0194  -15.2504  -28.2169
   -0.0000         0    0.0000   -0.0000  173.1377  141.4500 -171.9175   64.9155
    0.0000   -0.0000    0.0000   -0.0000   -0.0000 -255.1039   95.1075 -140.1344
   -0.0000    0.0000   -0.0000   -0.0000   -0.0000    0.0000 -201.7173  282.3698
    0.0000    0.0000    0.0000    0.0000    0.0000   -0.0000   -0.0000  163.9518
   -0.0000    0.0000   -0.0000   -0.0000    0.0000   -0.0000   -0.0000   -0.0000
      ⋮
```

It certainly looks upper Hessenberg.

How does it compare to H_m from the Arnoldi method?

```
Hm
```

```
Hm = 80×80
  -14.1240  -37.9523   -2.8775    1.4543    0.1684    2.9462   22.1405  -13.0381 ···
  134.8324  513.2181   14.1914  167.2011  -19.7124   42.3230 -182.3275  158.7274
         0  237.9963  195.8816  329.3283  -62.5824  191.4550 -265.2109  120.4547
         0         0  136.4856  116.0191  161.4556   29.8726  -94.9162   61.1546
         0         0         0  166.0327  265.5661  202.0194   15.2504  -28.2169
         0         0         0         0  173.1377  141.4500  171.9175   64.9155
         0         0         0         0         0  255.1039   95.1075  140.1344
         0         0         0         0         0         0  201.7173  282.3698
         0         0         0         0         0         0         0  163.9518
         0         0         0         0         0         0         0         0
      ⋮
```

Pretty much the same (apart from different sign choices in the QR orthogonalisation)?

Shall we give them both a try then? Let's approximate the first two largest eigenvalues of A. Here are the benchmark values.

```
k = 2;
lambda = eigs(A, k, 'lm')
```

```
lambda = 2×1
  594.5283
  591.6830
```

Using Arnoldi first:

```
[Y,D] = eigs(H(1:m,1:m), k, 'lm'); theta = diag(D)
```

```
theta = 2×1
  594.5283
  591.6830
```

Looking good!

Now the naive approach:

```
[Y2,D2] = eigs(H2, k, 'lm'); theta2 = diag(D2)
```

```
theta2 = 2×1
  594.5271
  591.6716
```

Also quite decent?

What about the Ritz vectors? Arnoldi first:

```
U = Vm * Y; % Ritz vectors
res = A*U - U*diag(theta);  % compute the full residual
norm(res(:,1)), norm(res(:,2))
```

```
ans =
1.1417e-05
ans =
8.9343e-04
```

The Arnoldi Ritz vectors are quite accurate.

And now the naive way:

```
U2 = Q * Y2;
res2 = A*U2 - U2*diag(theta2); % compute the full residual
norm(res2(:,1)), norm(res2(:,2))
```

```
ans =
0.0422
ans =
1.2511
```

Here we see the ill-conditioning contaminating the answer.