Solving a Toeplitz linear system

Contents

Toeplitz matrices

Toeplitz matrices are matrices with constant diagonal elements; they are completely determined by the first column and row, and indeed there is a MATLAB function that generates the full matrix starting by this information. We can generate an example by constructing a random column and row as follows:

```
n = 6;
c = randn(n, 1);
r = randn(1, n);
c(1) = r(1);

T = toeplitz(c, r)
```

```
T =

    0.5171   -1.0689    0.8193    0.2185    0.6074    0.3708
    0.0129    0.5171   -1.0689    0.8193    0.2185    0.6074
   -0.1707    0.0129    0.5171   -1.0689    0.8193    0.2185
   -0.7938   -0.1707    0.0129    0.5171   -1.0689    0.8193
    1.7793   -0.7938   -0.1707    0.0129    0.5171   -1.0689
   -0.3672    1.7793   -0.7938   -0.1707    0.0129    0.5171
```

Note that we have forced the first element of the vectors `c` and `r` to be equal, to make sure that matrix is well-defined. Suppose that we want to solve a linear system with a Toeplitz matrix: $Tx = b$.

For small scale systems, we can rely on the MATLAB backslash operator:

```
b = randn(n, 1);
x = T \ b;
```

Unfortunately, this approach scales cubically, and given the particular structure of the matrices it is natural to ask if a more efficient method might be developed.

This example presents the construction of the superfast Toeplitz solver introduced in [2].

Displacement equation

The first step in this direction is to note that a Toeplitz matrix T satisfies the displacement equation $Z_1 T - T Z_{-1} = F$ where Z_x is the matrix defined as follows:

$$Z_x := \begin{bmatrix} & & & x \\ 1 & & & \\ & \ddots & & \\ & & 1 & \end{bmatrix}$$

and F is a matrix with just the first row and the last column which are non-zero. We can verify this directly:

```
Z1 = eye(n); Z1 = Z1(:,[2:n,1]);
Zm1 = Z1; Zm1(1,end) = -1;
F = Z1 * T - T * Zm1
```

```
F =

    0.7017    0.9600   -1.0123   -0.7781   -0.3578    1.0343
         0         0         0         0         0    0.3837
         0         0         0         0         0    0.4367
         0         0         0         0         0   -0.5754
         0         0         0         0         0    2.5986
         0         0         0         0         0   -1.4360
```

Given the special structure of F, it has rank (at most) 2, and we can represent it as UV^T where U, V are the $n \times 2$ matrices defined as follows:

```
U = [ 1 , 2*c(1) ; zeros(n-1,1) , c(2:n) + r(n:-1:2).' ];
V = [ c(n:-1:2) - r(2:n).' , zeros(n-1,1) ; 0 , 1 ];

norm(F - U*V.')
```

```
ans =

     0
```

We can modify the displacement relation to transform the Toeplitz matrix in another matrix that has a particular structure. Let Ω_n denote the matrix of the Fourier transform, scaled to be unitary. Then, since Z_1 is circulant, $\Omega_n Z_1 \Omega_n^*$ is diagonal, and in particular it has the n-th roots of the unity on the diagonal. We call such matrix D_1.

Here we use the fact that the Fourier matrix as implemented in MATLAB can be obtained by properly flipping a Vandermonde matrix with the roots of the unity as nodes.

```
Omega = sqrt(n) \ rot90(vander(exp(2i*pi/n.*(1:n))), 2);
D1 = Omega * Z1 * Omega';
```

In a similar fashion, if we define $D_0 = \mathrm{diag}(1, \omega_{2n}, \ldots, \omega_{2n}^{n-1})$, where ω_{2n} is $e^{\frac{i\pi}{n}}$, we have the relation

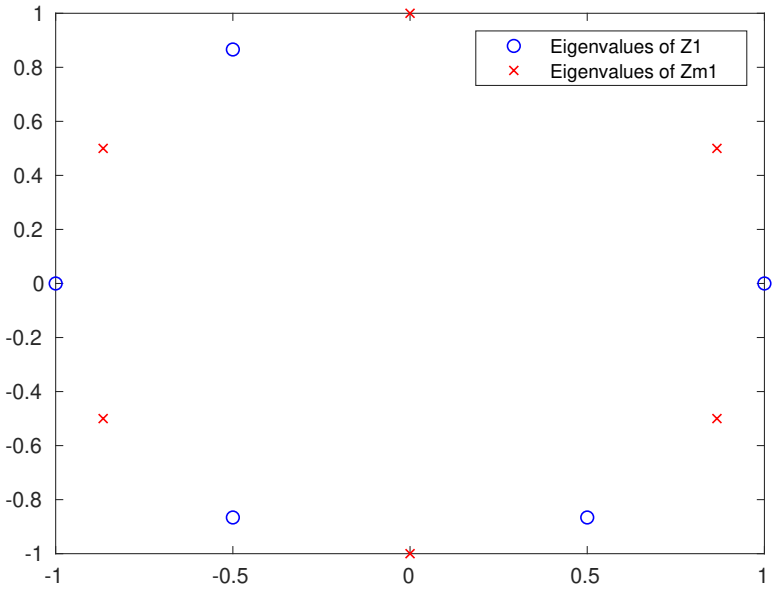
$$\Omega_n D_0 Z_{-1} D_0^* \Omega_n^* = D_{-1},$$

where $D_{-1} = \omega_{2n} D_1$.

```
om = exp(1i * pi / n);
D0 = diag(om.^(0:n-1));
Dm1 = Omega * D0 * Zm1 * D0' * Omega';
```

Let us check the eigenvalues of the matrices Z_1 and Z_{-1}. Together, they constitute the $2n$-th roots of the unity.

```
plot(real(diag(D1)),  imag(diag(D1)),  'bo', ...
 real(diag(Dm1)), imag(diag(Dm1)), 'rx');
axis([-1 1 -1 1]);
legend('Eigenvalues of Z1', 'Eigenvalues of Zm1');
```



A superfast algorithm

We now exploit the displacement relation to develop a superfast algorithm. To test it, we are going to need a larger example, so we regenerate our data with a larger n, but not too large so we will be able to verify our computations.

```
n = 4096;
c = randn(n, 1); r = randn(1, n); c(1) = r(1);
T = toeplitz(c, r);
b = randn(n, 1);

d0 = exp(1i * pi / n .* (0 : n - 1));
d1  = d0.^2;
dm1 = exp(1i * pi / n) * d1;

U = [ 1 , 2*r(1) ; zeros(n-1, 1), r(end:-1:2).' + c(2:end) ];
V = [ conj(c(end:-1:2)) - r(2:end)' , zeros(n-1,1) ; 0 1 ];
```

Multiplying the displacement relation from the left by Ω_n and from the right by $D_0^*\Omega_n^*$ yields the new relation

$$D_1 C - C D_{-1} = G F^T$$

where G, F can be defined as $G = \Omega_n U$ and $F = \overline{\Omega_n} D_0^* V$, and $C = \Omega_n T D_0^* \Omega_n^*$.

The previous relation tells us that the matrix C has a *Cauchy-like* structure; since the coefficients of the displacement equation are diagonal, we can explicitly write its entries as:

$$C_{ij} = \frac{G_i H_j^T}{\omega_{2n}^{2(i-1)} - \omega_{2n}^{(2j-1)}}$$

It turns out that the off-diagonal blocks of C are Cauchy-like matrices involving nodes contained in separated domains, so we expect them to be numerically low-rank [1].

Since its entries are explicitly available, and we can use the expression $C = \Omega_n T D_0 \Omega_n^*$ to implement a fact vector product with C and C^T relying on the fast toeplitz product using the FFT, we can use the HSS constructor to build the HSS representation of C.

For simplicity, in this example we also rely on the HODLR constructor that, using Adaptive Cross Approximation, only requires the matrix evaluation at an index (i, j).

```
Gh = ifft(U) * sqrt(n);
Fh = ifft((d0.' * ones(1, 2)) .* V) * sqrt(n);

C = hodlr2hss(hodlr('handle', ...
@(i,j) (Gh(i, :) * Fh(j, :)') ./ (d1(i).' - dm1(j)), ...
n, n));
```

Now, we can solve the linear system with T by inverting the relation and writing $T = \Omega_n^* C \Omega_n D_0$, which yields

```
z = ifft(b);
y = C \ z;
x = d0' .* fft(y);
```

As a final test, we check the accuracy of the solution (that should be compared with $\|T\|_2$, which we do not compute because it might be expensive).

```
norm(T * x - b)
```

```
ans =

   3.0482e-10
```

The algorithm described here is implemented using the randomized constructor in place of the Adaptive Cross in the function `toeplitz_solve`, which is included the toolbox.

References

[1] Beckermann, Bernhard, and Alex Townsend. "Bounds on the Singular Values of Matrices with Displacement Structure." SIAM Review 61.2 (2019): 319-344.

[2] Xia, Jianlin, Yuanzhe Xi, and Ming Gu. "A superfast structured solver for Toeplitz linear systems via randomized sampling." SIAM Journal on Matrix Analysis and Applications 33.3 (2012): 837-858.