Structured matrices

We will discuss three types of structured matrices, although plenty of other types exist:

- identity+low rank
- circulant
- Toeplitz

In each of these cases, the structure of the system allows us to do a solve in much better than $O(N^3)$ operations.

Identity + low rank

Consider a system of the form

$$(\gamma \mathbf{I} + \boldsymbol{B}\boldsymbol{B}^{\mathrm{T}})\boldsymbol{x} = \boldsymbol{b},$$

where $\gamma > 0$ is some scalar, and $\boldsymbol{B}$ is a $N \times R$ matrix with $R < N$. These types of systems are prevalent in array signal processing and machine learning. We will see that if $R \ll N$, this system can be solved in (much) faster than $O(N^3)$ time.

Note that while $\boldsymbol{B}\boldsymbol{B}^{\mathrm{T}}$ is not at all invertible (since it is rank deficient), $\gamma\mathbf{I} + \boldsymbol{B}\boldsymbol{B}^{\mathrm{T}}$ will be. To see this, set

$$\boldsymbol{z} = \boldsymbol{B}^{\mathrm{T}}\boldsymbol{x}, \qquad \boldsymbol{z} \in \mathbb{R}^R.$$

Then we can solve the system by jointly solving for $\boldsymbol{x}$ and $\boldsymbol{z}$:

$$\gamma\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z} = \boldsymbol{b} \tag{1}$$
$$\boldsymbol{B}^{\mathrm{T}}\boldsymbol{x} - \boldsymbol{z} = \mathbf{0}. \tag{2}$$

Solving the first equations (1) yields

$$\boldsymbol{x} = \gamma^{-1}(\boldsymbol{b} - \boldsymbol{B}\boldsymbol{z}),$$

and then plugging this into (2) gives us

$$\begin{aligned} \gamma^{-1}\boldsymbol{B}^{\mathrm{T}}(\boldsymbol{b} - \boldsymbol{B}\boldsymbol{z}) - \boldsymbol{z} &= \boldsymbol{0} \\ \Rightarrow \quad (\gamma\mathbf{I} + \boldsymbol{B}^{\mathrm{T}}\boldsymbol{B})\boldsymbol{z} &= \boldsymbol{B}^{\mathrm{T}}\boldsymbol{b} \end{aligned}$$

and so

$$\boldsymbol{z} = (\gamma\mathbf{I} + \boldsymbol{B}^{\mathrm{T}}\boldsymbol{B})^{-1}\boldsymbol{B}^{\mathrm{T}}\boldsymbol{b}.$$

But notice that this is an $R \times R$ system of equations.

So it takes $O(NR^2)$ to construct $\gamma\mathbf{I} + \boldsymbol{B}^{\mathrm{T}}\boldsymbol{B}$,
then $O(R^3)$ to solve for $\boldsymbol{z}$,
then $O(NR)$ to calculate $\boldsymbol{B}\boldsymbol{z}$ (and hence find $\boldsymbol{x}$).

The dominant cost in all of this is $O(NR^2)$, which is much less than $O(N^3)$ if $R \ll N$.

Circulant systems.

A circulant matrix has the form

$$\boldsymbol{H} = \begin{bmatrix} h_0 & h_{N-1} & h_{N-2} & \cdots & h_1 \\ h_1 & h_0 & h_{N-1} & \cdots & h_2 \\ h_2 & h_1 & h_0 & & \vdots \\ \vdots & \vdots & & \ddots & h_{N-1} \\ h_{N-1} & & & h_1 & h_0 \end{bmatrix}$$

For $\boldsymbol{H}$ symmetric, we have $h_k = h_{N-k}$ for $k = 1, \ldots, N-1$, although symmetry does not play too big a role in exploiting this structure.

Circulant matrices have two very nice properties:

- We know their eigenvectors already — they are the discrete harmonic sinusoids (i.e. the columns of the $N \times N$ DFT matrix).
- Transforming into the eigenbasis is **fast** thanks to the FFT (which is $O(N \log N)$).

We can write

$$\boldsymbol{H} = \boldsymbol{F}\boldsymbol{\Lambda}\boldsymbol{F}^{\mathrm{H}}, \qquad F[m,n] = \frac{1}{\sqrt{N}} e^{j2\pi mn/N}$$

and

$$\boldsymbol{H}^{-1} = \boldsymbol{F}\boldsymbol{\Lambda}^{-1}\boldsymbol{F}^{\mathrm{H}},$$

so

$$\boldsymbol{H}^{-1}\boldsymbol{b} = \underbrace{\boldsymbol{F}}_{\text{FFT, } O(N\log N);} \quad \underbrace{\boldsymbol{\Lambda}^{-1}}_{\text{diagonal weighting, } O(N);} \quad \underbrace{\boldsymbol{F}^{\mathrm{H}}\boldsymbol{b}}_{\text{FFT, } O(N\log N)}$$

$\Rightarrow$ solving an $N \times N$ system of equations can be done in $O(N \log N)$ time!

This is **fast** compared to $O(N^3)$ — on my computer, I can solve a system like this in $N = 20000$ in 800μs (compare to 24 seconds for the general case).

Toeplitz systems.

Toeplitz matrices, which are matrices that are constant along their diagonals, arise in many different signal processing applications, as they are fundamental in describing the action of linear time-invariant systems. For example, suppose we observe the discrete convolution of an unknown signal $\boldsymbol{x}$ of length N and a known sequence[1] $a_0, \ldots, a_{L-1}$ of length L. We can write the corresponding matrix equation as

$$\begin{bmatrix} a_0 & 0 & \cdots & & 0 \\ a_1 & a_0 & 0 & \cdots & 0 \\ a_2 & a_1 & a_0 & \cdots & 0 \\ \vdots & & & & \\ a_{N-1} & a_{N-2} & \cdots & & a_0 \\ \vdots & & & & \\ a_{L-1} & a_{L-2} & \cdots & & a_{L-N} \\ 0 & a_{L-1} & \cdots & & a_{L-N+1} \\ \vdots & & & & \\ 0 & 0 & \cdots & \cdots & a_{L-1} \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ \vdots \\ x[N-1] \end{bmatrix} = \begin{bmatrix} y[0] \\ y[1] \\ y[2] \\ \vdots \\ y[N-1] \\ \vdots \\ y[L-1] \\ y[L] \\ \vdots \\ y[L+N-2] \end{bmatrix}$$

If we recover $\boldsymbol{x}$ from $\boldsymbol{y}$ using least-squares, $\widehat{\boldsymbol{x}} = \boldsymbol{A}^{\dagger}\boldsymbol{y} = (\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})^{-1}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y}$, then the $N \times N$ system we need to invert, $\boldsymbol{H} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ is also Toeplitz (and is of course symmetric and non-negative definite):

$$\boldsymbol{H} = \begin{bmatrix} h_0 & h_1 & \cdots & & h_{N-1} \\ h_1 & h_0 & h_1 & \cdots & h_{N-2} \\ h_2 & h_1 & h_0 & \cdots & h_{N-3} \\ \vdots & & & & \\ h_{N-1} & \cdots & & \cdots & h_0 \end{bmatrix}.$$

[1]We are going to save some space in this section by using subscript notation to index signals that are going in a matrix; i.e. a_k instead of $a[k]$.

Here is a quick example in MATLAB:

```
  >> A = toeplitz([1; randn(5,1); zeros(3,1)], [1 zeros(1,3)])

    A =

       1.0000         0         0         0
      -0.4336    1.0000         0         0
       0.3426   -0.4336    1.0000         0
       3.5784    0.3426   -0.4336    1.0000
       2.7694    3.5784    0.3426   -0.4336
      -1.3499    2.7694    3.5784    0.3426
            0   -1.3499    2.7694    3.5784
            0         0   -1.3499    2.7694
            0         0         0   -1.3499


  >> H = A'*A

    H =

      23.6023    6.8156   -5.0905    1.9151
       6.8156   23.6023    6.8156   -5.0905
      -5.0905    6.8156   23.6023    6.8156
       1.9151   -5.0905    6.8156   23.6023
```

Symmetric Toeplitz systems appear frequently in linear prediction, array processing, adaptive filtering, and other areas of statistical signal processing. An $N \times N$ Toeplitz system $\boldsymbol{H}$ can be inverted in $O(N^2)$ time using the **Levinson-Durbin** algorithm. The algorithm is relatively easy to derive, and even easier to implement. The increase in efficiency it offers is significant, as the difference between $O(N^3)$, the cost of solving the system using a general linear solver, and $O(N^2)$ is enormous even for moderate N.

Iterative methods for solving least-squares

When $\boldsymbol{A}$ has full column rank, our least-squares estimate is

$$\widehat{\boldsymbol{x}} = (\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})^{-1}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y}.$$

If $\boldsymbol{A}$ is $M \times N$, then constructing $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ costs $O(MN^2)$ computations, and solving the $N \times N$ system $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{y}$ costs $O(N^3)$ computations. (Note that for $M \geq N$, the cost of constructing the matrix actually exceeds the cost to solve the system.)

This cost can be prohibitive for even moderately large M and N. But inverse problems with large M and N are common in the modern world. For example, a typical 3D MRI scan will try to reconstruct a $128 \times 128 \times 128$ cube of voxels from about 5 million non-uniformly spaced samples in the spatial Fourier domain. In this case, the matrix $\boldsymbol{A}$, which models the MRI machine, is $M = 5 \cdot 10^6$ by $N = 2.1 \cdot 10^6$.

With those values, MN^2 is huge ($\sim 10^{19}$); even storing the matrix $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ in memory would require terabytes of RAM.

Beginning next time, we will consider approaches that reformulate

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{y}$$

as an optimization program and then solve it by an iterative descent method. Each iteration is simple, requiring one application of $\boldsymbol{A}$ and one application of $\boldsymbol{A}^{\mathrm{T}}$.

If $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ is well-conditioned, then these methods can converge in very few iterations (especially conjugate gradients). This makes the cost of solving a least-squares problem dramatically smaller — about the cost of a few hundred applications of $\boldsymbol{A}$.

Moreover, we will not need to construct $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ or even $\boldsymbol{A}$ explicitly. All we need is a "black box" which takes a vector $\boldsymbol{x}$ and returns $\boldsymbol{A}\boldsymbol{x}$. This is especially useful if it takes $\ll O(MN)$ operations to apply $\boldsymbol{A}$ or $\boldsymbol{A}^{\mathrm{T}}$.

In the MRI example above, it takes about one second to apply $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$, and the conjugate gradients method converges in about 50 iterations, meaning that the problem can be solved in less than a minute. Also, the storage requirement is on the order of $O(M+N)$, rather than $O(MN)$.

In such a case we can take an alternative approach. Specifically, recall that the least squares estimate is the solution to the optimization problem

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \;\; \|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{y}\|_2^2.$$

Note that we can write this equivalently as

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \;\; \boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} - 2\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y} + \boldsymbol{y}^{\mathrm{T}}\boldsymbol{y}.$$

We can ignore terms that do not depend on $\boldsymbol{x}$, and can also rescale the objective function by a constant (for convenience) to obtain

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \;\; \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} - \boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y}. \tag{3}$$

We have previously shown that a necessary and sufficient condition for $\widehat{\boldsymbol{x}}$ to be the the minimizer of (3) is to satisfy

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{y}.$$

More generally, for any $\boldsymbol{H}$ which is symmetric and positive definite and any vector $\boldsymbol{b}$, we can consider the optimization problem

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \ \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x} - \boldsymbol{x}^{\mathrm{T}}\boldsymbol{b}, \tag{4}$$

and by the same argument we can show that $\widehat{\boldsymbol{x}}$ is the solution to (4) if and only if

$$\boldsymbol{H}\widehat{\boldsymbol{x}} = \boldsymbol{b}.$$

What remains is to show how we can actually solve an optimization problem of the form (4) without directly solving the system $\boldsymbol{H}\boldsymbol{x} = \boldsymbol{b}$. Next time we will consider two iterative methods — **steepest descent** and **conjugate gradients** — that do exactly this.

Technical Details: Solving Toeplitz systems and the Levinson-Durbin algorithm

We start by looking at how to solve $\boldsymbol{Hv} = \boldsymbol{y}$ for a very particular right-hand side. Consider

$$\begin{bmatrix} h_0 & h_1 & \cdots & & h_{N-1} \\ h_1 & h_0 & h_1 & \cdots & h_{N-2} \\ h_2 & h_1 & h_0 & \cdots & h_{N-3} \\ \vdots & & & & \\ h_{N-1} & \cdots & & \cdots & h_0 \end{bmatrix} \begin{bmatrix} v[1] \\ v[2] \\ \vdots \\ \vdots \\ v[N] \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_{N-1} \\ h_N \end{bmatrix}. \qquad (5)$$

Here the first $N-1$ entries of the "observation" vector $\boldsymbol{y}$ match the last $N-1$ entries in the first column of $\boldsymbol{H}$. This system is specialized, but not at all contrived — (5) are called the **Yule-Walker equations**, and appear in many different places in statistical signal processing. Moreover, solving systems with general right-hand sides solve systems of the form (5) as an intermediate step.

The crux of the Levinson-Durbin algorithm relies upon a seemingly innocuous fact. Let $\mathbf{J}$ be the $N \times N$ *exchange matrix* (also called the *counter identity*):

$$\mathbf{J} = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 & 1 \\ 0 & 0 & \cdots & 0 & 1 & 0 \\ 0 & 0 & \cdots & 1 & 0 & 0 \\ \vdots & & & & & \vdots \\ 1 & 0 & \cdots & & & 0 \end{bmatrix}.$$

Applying $\mathbf{J}$ to a vector $\boldsymbol{x}$ reverses the entries in $\boldsymbol{x}$. For example,

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 5 \\ -1 \\ 7 \end{bmatrix} = \begin{bmatrix} 7 \\ -1 \\ 5 \end{bmatrix}.$$

It should be clear that $\mathbf{J}^{\mathrm{T}} = \mathbf{J}$ and $\mathbf{J}^2 = \mathbf{J}\mathbf{J} = \mathbf{I}$.

Applying $\mathbf{J}$ to the left of a matrix reverses all of its columns, while applying $\mathbf{J}$ to the right reverses the rows. In particular, if $\boldsymbol{H}$ is a symmetric Toeplitz matrix, then

$$\begin{aligned}
\mathbf{J}\boldsymbol{H} &= \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 & 1 \\ 0 & 0 & \cdots & 0 & 1 & 0 \\ 0 & 0 & \cdots & 1 & 0 & 0 \\ \vdots & & & & & \vdots \\ 1 & 0 & \cdots & & & 0 \end{bmatrix} \begin{bmatrix} h_0 & h_1 & \cdots & & h_{N-1} \\ h_1 & h_0 & h1 & \cdots & h_{N-2} \\ h_2 & h_1 & h_0 & \cdots & h_{N-3} \\ \vdots & & & & \\ h_{N-1} & \cdots & & \cdots & h_0 \end{bmatrix} \\
&= \begin{bmatrix} h_{N-1} & h_{N-2} & \cdots & & h_0 \\ h_{N-2} & h_{N-3} & h_{N-4} & \cdots & h_1 \\ h_{N-3} & h_{N-4} & h_{N-5} & \cdots & h_2 \\ \vdots & \vdots & & & \\ h_0 & h_1 & & \cdots & h_{N-1}, \end{bmatrix}
\end{aligned}$$

and

$$\begin{aligned}
\mathbf{J}\boldsymbol{H}\mathbf{J} &= \begin{bmatrix} h_{N-1} & h_{N-2} & \cdots & & h_0 \\ h_{N-2} & h_{N-3} & h_{N-4} & \cdots & h_1 \\ h_{N-3} & h_{N-4} & h_{N-5} & \cdots & h_2 \\ \vdots & \vdots & & & \\ h_0 & h_1 & & \cdots & h_{N-1}, \end{bmatrix} \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 & 1 \\ 0 & 0 & \cdots & 0 & 1 & 0 \\ 0 & 0 & \cdots & 1 & 0 & 0 \\ \vdots & & & & & \vdots \\ 1 & 0 & \cdots & & & 0 \end{bmatrix} \\
&= \begin{bmatrix} h_0 & h_1 & \cdots & & h_{N-1} \\ h_1 & h_0 & h1 & \cdots & h_{N-2} \\ h_2 & h_1 & h_0 & \cdots & h_{N-3} \\ \vdots & & & & \\ h_{N-1} & \cdots & & \cdots & h_0 \end{bmatrix} \\
&= \boldsymbol{H}.
\end{aligned}$$

So symmetric Toeplitz matrices obey the identity

$$\begin{aligned} \boldsymbol{H} &= \mathbf{J}\boldsymbol{H}\mathbf{J} \\ \Rightarrow \mathbf{J}\boldsymbol{H} &= \boldsymbol{H}\mathbf{J} \qquad (\text{since } \mathbf{J}\mathbf{J} = \mathbf{I}) \end{aligned}$$

That is, $N \times N$ symmetric Toeplitz matrices **commute** with $\mathbf{J}$.

Also note that

$$\begin{aligned} \boldsymbol{H}^{-1} &= (\mathbf{J}\boldsymbol{H}\mathbf{J})^{-1} \\ &= \mathbf{J}^{-1}\boldsymbol{H}^{-1}\mathbf{J}^{-1} \\ &= \mathbf{J}\boldsymbol{H}^{-1}\mathbf{J} \quad (\text{since } \mathbf{J}^{-1} = \mathbf{J}), \end{aligned}$$

and so $\boldsymbol{H}^{-1}$ commutes with $\mathbf{J}$ as well:

$$\boldsymbol{H}^{-1}\mathbf{J} = \mathbf{J}\boldsymbol{H}^{-1}.$$

Important note: Even though $\boldsymbol{H}^{-1}$ does commute with $\mathbf{J}$, it is not necessarily Toeplitz. Matrices which commute with $\mathbf{J}$ are called **persymmetric**. So what the calculations above are telling us is that all Toeplitz matrices are persymmetric, all inverses of Toeplitz matrices are persymmetric, but inverses of Toeplitz matrices are not (in general) Toeplitz.

Let's see how to take advantage of these properties in solving the Yule-Walker equations. Start by parititioning off the last row and column:

$$\left[\begin{array}{cccc|c} h_0 & h_1 & h_2 & \cdots & h_{N-1} \\ h_1 & h_0 & h_1 & \cdots & h_{N-2} \\ h_2 & & \ddots & & \vdots \\ \vdots & & & \ddots & h_1 \\ \hline h_{N-1} & h_{N-2} & \cdots & & h_0 \end{array}\right] \left[\begin{array}{c} v[1] \\ v[2] \\ \vdots \\ v[N-1] \\ \hline v[N] \end{array}\right] = \left[\begin{array}{c} h_1 \\ h_2 \\ \vdots \\ h_{N-1} \\ \hline h_N \end{array}\right]$$

which we re-write as

$$\left[\begin{array}{c|c} \boldsymbol{H}_{N-1} & \mathbf{J}\boldsymbol{h}_{N-1} \\ \hline (\mathbf{J}\boldsymbol{h}_{N-1})^{\mathrm{T}} & h_0 \end{array}\right] \left[\begin{array}{c} \boldsymbol{z} \\ \hline \beta \end{array}\right] = \left[\begin{array}{c} \boldsymbol{h}_{N-1} \\ \hline h_N \end{array}\right]$$

where

- $\boldsymbol{H}_{N-1}$ consists of the first $N-1$ rows and columns of $\boldsymbol{H}$ (this is also Toeplitz)
- $\boldsymbol{h}_{N-1} \in \mathbb{R}^{N-1}$ contains $h_1, \ldots, h_{N-1}$

Now we would like to solve for the vector $\boldsymbol{z} \in \mathbb{R}^{N-1}$ and the scalar β. We have[2]

$$\boldsymbol{H}_{N-1}\boldsymbol{z} + \beta\mathbf{J}\boldsymbol{h}_{N-1} = \boldsymbol{h}_{N-1} \tag{6}$$
$$\boldsymbol{h}_{N-1}^{\mathrm{T}}\mathbf{J}\boldsymbol{z} + \beta h_0 = h_N. \tag{7}$$

Solving the first equation yields

$$\begin{aligned} \boldsymbol{z} &= \boldsymbol{H}_{N-1}^{-1}(\boldsymbol{h}_{N-1} - \beta\mathbf{J}\boldsymbol{h}_{N-1}) \\ &= \boldsymbol{H}_{N-1}^{-1}\boldsymbol{h}_{N-1} - \beta\boldsymbol{H}_{N-1}^{-1}\mathbf{J}\boldsymbol{h}_{N-1} \\ &= \boldsymbol{H}_{N-1}^{-1}\boldsymbol{h}_{N-1} - \beta\mathbf{J}\boldsymbol{H}_{N-1}^{-1}\boldsymbol{h}_{N-1} \quad (\text{since } \boldsymbol{H}_{N-1}^{-1} \text{ commutes with } \mathbf{J}). \end{aligned}$$

Suppose we already had the solution to the smaller system

$$\boldsymbol{v}_{N-1} = \boldsymbol{H}_{N-1}^{-1}\boldsymbol{h}_{N-1}$$

in hand. Then we could compute $\boldsymbol{z}$ using

$$\boldsymbol{z} = \boldsymbol{v}_{N-1} - \beta\mathbf{J}\boldsymbol{v}_{N-1},$$

[2]Here we use the fact that $\mathbf{J}^{\mathrm{T}} = \mathbf{J}$.

and then plugging this into (7) gives us the **scalar** equation

$$\boldsymbol{h}_{N-1}^{\mathrm{T}}\mathbf{J}(\boldsymbol{v}_{N-1} - \beta\mathbf{J}\boldsymbol{v}_{N-1}) + \beta h_0 = h_N$$
$$\Rightarrow \quad \beta = \frac{h_N - \boldsymbol{h}_{N-1}^{\mathrm{T}}\mathbf{J}\boldsymbol{v}_{N-1}}{h_0 - \boldsymbol{h}_{N-1}^{\mathrm{T}}\boldsymbol{v}_{N-1}},$$

and so we take

$$\boldsymbol{z} = \boldsymbol{v}_{N-1} - \beta\mathbf{J}\boldsymbol{v}_{N-1},$$

and set

$$\boldsymbol{v}_N = \begin{bmatrix}\boldsymbol{z}\\ \beta\end{bmatrix} = \boldsymbol{H}_N^{-1}\boldsymbol{h}_N.$$

Moral: Given the solution to

$$\boldsymbol{H}_{N-1}\boldsymbol{v}_{N-1} = \boldsymbol{h}_{N-1}$$

the solution to

$$\boldsymbol{H}_N\boldsymbol{v}_N = \boldsymbol{h}_N$$

can be computed in $O(N)$ time (a few inner products).

So to solve the $N \times N$ system of equations $\boldsymbol{H}\boldsymbol{v}_N = \boldsymbol{h}_N$, we work "from the ground up", first solving the 1×1 system

$$\boldsymbol{H}_1\boldsymbol{v}_1 = \boldsymbol{h}_1,$$

then using the solution of this to solve the 2×2 system

$$\boldsymbol{H}_2\boldsymbol{v}_2 = \boldsymbol{h}_2,$$

and then using the solution to $\boldsymbol{H}_{N-1}\boldsymbol{v}_{N-1} = \boldsymbol{h}_{N-1}$ to solve the $N \times N$ system[3]

$$\boldsymbol{H}_N \boldsymbol{v}_N = \boldsymbol{h}_N.$$

Adding together the computational costs at each stage:

$$\begin{aligned}\text{Total cost} &= (\text{some constant})(1 + 2 + \cdots + N - 1 + N) \\ &= O(N^2).\end{aligned}$$

General right-hand sides

Solving for a general right-hand side is not much harder — it just takes twice the work. (And $2N^2$ still beats N^3 every day of the week.)

To solve

$$\boldsymbol{H}\boldsymbol{x} = \boldsymbol{y}$$

we again subdivide it into sections:

$$\left[\begin{array}{c|c} \boldsymbol{H}_{N-1} & \mathbf{J}\boldsymbol{h}_{N-1} \\ \hline (\mathbf{J}\boldsymbol{h}_{N-1})^{\mathrm{T}} & h_0 \end{array}\right] \left[\begin{array}{c} \boldsymbol{w} \\ \hline \alpha \end{array}\right] = \left[\begin{array}{c} \boldsymbol{y}_{N-1} \\ \hline y_N \end{array}\right],$$

and so

$$\begin{aligned} \boldsymbol{H}_{N-1}\boldsymbol{w} + \alpha \mathbf{J}\boldsymbol{h}_{N-1} &= \boldsymbol{y}_{N-1} \\ \boldsymbol{h}_{N-1}^{\mathrm{T}}\mathbf{J}\boldsymbol{w} + \alpha h_0 &= y_N. \end{aligned}$$

Solving the first equation:

$$\boldsymbol{w} = \boldsymbol{H}_{N-1}^{-1}\boldsymbol{y}_{N-1} - \alpha \mathbf{J}\boldsymbol{H}_{N-1}^{-1}\boldsymbol{h}_{N-1}.$$

[3]By definition, $\boldsymbol{H}_N = \boldsymbol{H}$.

Now suppose we have the following solutions in hand:

$$\begin{aligned}\boldsymbol{x}_{N-1} &= \boldsymbol{H}_{N-1}^{-1}\boldsymbol{y}_{N-1}, \quad \text{and} \\ \boldsymbol{v}_{N-1} &= \boldsymbol{H}_{N-1}^{-1}\boldsymbol{h}_{N-1}.\end{aligned}$$

Then we can again quickly solve for $\boldsymbol{w}$ and α:

$$\boldsymbol{w} = \boldsymbol{x}_{N-1} - \alpha\mathbf{J}\boldsymbol{v}_{N-1},$$

with

$$\alpha = \frac{y_N - \boldsymbol{h}_{N-1}^{\mathrm{T}}\mathbf{J}\boldsymbol{x}_{N-1}}{h_0 - \boldsymbol{h}_{N-1}^{\mathrm{T}}\boldsymbol{v}_{N-1}}.$$

So given the solutions to

$$\begin{aligned}\boldsymbol{H}_{N-1}\boldsymbol{x}_{N-1} &= \boldsymbol{y}_{N-1} \\ \boldsymbol{H}_{N-1}\boldsymbol{v}_{N-1} &= \boldsymbol{h}_{N-1},\end{aligned}$$

the solution to

$$\boldsymbol{H}_N\boldsymbol{x}_N = \boldsymbol{y}_N,$$

(and also the solution to $\boldsymbol{H}_N\boldsymbol{v}_N = \boldsymbol{h}_N$) can be computed in $O(N)$ time.

Moral: Solving the $N \times N$ symmetric Toeplitz system

$$\boldsymbol{H}\boldsymbol{x} = \boldsymbol{y}$$

can be done in $O(N^2)$ time.

What we have done above is easily extended to non-symmetric Toeplitz systems as well.