Generalizing the inverse FFT off the unit circle

Signal-Processing Final Project

Team - 20 - Cloudy Dreamers
Adityakumar Sinha (2019102003)
Bhaiya Vaibhaw Kumar (2019112021)

Overview

The project suggests a O(nlogn) algorithm of computing inverse Chirp Z Transform (ICZT) when the number of points in input and output are exactly the same. Just like Chirp Z Transform (CZT) is viewed as the generalisation of Fast Fourier Transform (FFT) and both are having the same complexity, the ICZT can be viewed as a generalization of the inverse fast Fourier transform (IFFT) off the unit circle in the complex plane.

Goals

1. Our main goal is to find an algorithm for doing inverse chirp Z transform.
2. Check how accurately the algorithm is working.

Problem Description

The M-point CZT for a N-point input is defined as:-

$$X_k = \sum_{j=0}^{N-1} x_j \, A^{-j} \, W^{jk}, \; k = 0, 1, \ldots, M-1.$$

We are given the output vector **X** which contains {X_K for k = 0,1,2,3,4,....,M-1 } and we have to find the input vector **x** which contains {X_n for n = 0,1,2,3,4,....,N-1 }

Calculation/Solution approach

We can write

$$\mathbf{X} = \boldsymbol{W} \, \mathbf{A} \, \mathbf{x}.$$

Where **A** is a diagonal matrix given by

$$\mathbf{A} = \mathrm{diag}(A^{-0}, A^{-1}, A^{-2}, \ldots, A^{-(N-1)})$$

And **W** is a Vandermonde Matrix given by

$$\boldsymbol{W} = \underbrace{\begin{bmatrix} W^{0\cdot 0} & W^{1\cdot 0} & W^{2\cdot 0} & \dots & W^{(N-1)\cdot 0} \\ W^{0\cdot 1} & W^{1\cdot 1} & W^{2\cdot 1} & \dots & W^{(N-1)\cdot 1} \\ W^{0\cdot 2} & W^{1\cdot 2} & W^{2\cdot 2} & \dots & W^{(N-1)\cdot 2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ W^{0\cdot(M-1)} & W^{1\cdot(M-1)} & W^{2\cdot(M-1)} & \dots & W^{(N-1)\cdot(M-1)} \end{bmatrix}}_{\text{Vandermonde matrix}}.$$

We can write

$$jk = \frac{j^2 + k^2 - (k-j)^2}{2}$$

Thus, our initial equation becomes

$$\mathbf{X}_k = \sum_{j=0}^{N-1} \mathrm{x}_j \, A^{-j} \, W^{\frac{j^2}{2}} \, W^{\frac{k^2}{2}} \, W^{-\frac{(k-j)^2}{2}}.$$

We can notice a few things here:-

1. $W^{\frac{k^2}{2}}$ maps to an M-by-M diagonal matrix $\mathbf{P} = \mathrm{diag}(W^{\frac{0^2}{2}}, W^{\frac{1^2}{2}}, \dots, W^{\frac{(M-1)^2}{2}})$
2. $W^{\frac{j^2}{2}}$ maps to a diagonal matrix $\mathbf{Q}$ that has N rows and N columns.

   $\mathbf{Q} = \mathrm{diag}(W^{\frac{0^2}{2}}, W^{\frac{1^2}{2}}, \dots, W^{\frac{(N-1)^2}{2}})$
3.

$W^{-\frac{(k-j)^2}{2}}$ maps to an M-by-N Toeplitz matrix $\hat{W}$:

$$\hat{W} = \underbrace{\begin{bmatrix} W^{-\frac{(0-0)^2}{2}} & W^{-\frac{(0-1)^2}{2}} & \dots & W^{-\frac{(0-(N-1))^2}{2}} \\ W^{-\frac{(1-0)^2}{2}} & W^{-\frac{(1-1)^2}{2}} & \dots & W^{-\frac{(1-(N-1))^2}{2}} \\ \vdots & \vdots & \ddots & \vdots \\ W^{-\frac{((M-1)-0)^2}{2}} & W^{-\frac{((M-1)-1)^2}{2}} & \dots & W^{-\frac{((M-1)-(N-1))^2}{2}} \end{bmatrix}}_{\text{Toeplitz matrix}}.$$

Since $\boldsymbol{W} = \mathbf{P}\,\hat{\boldsymbol{W}}\,\mathbf{Q}$, the CZT algorithm can be viewed as an efficient implementation of the following matrix equation:

$$\mathbf{X} = \mathbf{P}\,\hat{\boldsymbol{W}}\,\mathbf{Q}\,\mathbf{A}\,\mathbf{x}.$$

Now, if M = N then all the matrices involved in the above expressions are invertible.

Thus, we can write

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{Q}^{-1}\hat{\boldsymbol{W}}^{-1}\mathbf{P}^{-1}\mathbf{X}$$

Thus, in this way we can find input vector x.

Apart from $\hat{W}^{-1}$, other matrices are diagonal and their product can be calculated in O(n) time.

We only need to figure out an efficient way of calculating the inverse of a Toeplitz Matrix.
Let **T** be a non-singular 3-by-3 Toeplitz matrix generated by five complex numbers *a*, *b*, *c*, *d*, and *e*.

$$\mathbf{T} = \begin{bmatrix} a & b & c \\ d & a & b \\ e & d & a \end{bmatrix}$$

According to Gohberg–Semencul formula,

$$u_0 \mathbf{T}^{-1} = \underbrace{\begin{bmatrix} u_0 & 0 & 0 \\ u_1 & u_0 & 0 \\ u_2 & u_1 & u_0 \end{bmatrix}}_{\mathcal{A}} \underbrace{\begin{bmatrix} v_2 & v_1 & v_0 \\ 0 & v_2 & v_1 \\ 0 & 0 & v_2 \end{bmatrix}}_{\mathcal{C}} - \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ v_0 & 0 & 0 \\ v_1 & v_0 & 0 \end{bmatrix}}_{\mathcal{B}} \underbrace{\begin{bmatrix} 0 & u_2 & u_1 \\ 0 & 0 & u_2 \\ 0 & 0 & 0 \end{bmatrix}}_{\mathcal{D}},$$

As $\hat{W}^{-1}$ is a symmetric toeplitz matrix,

$$u_0 \widehat{\mathbf{W}}^{-1} = \underbrace{\begin{bmatrix} u_0 & 0 & 0 \\ u_1 & u_0 & 0 \\ u_2 & u_1 & u_0 \end{bmatrix}}_{\mathcal{A}} \underbrace{\begin{bmatrix} u_0 & u_1 & u_2 \\ 0 & u_0 & u_1 \\ 0 & 0 & u_0 \end{bmatrix}}_{\mathcal{A}^T} - \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ u_2 & 0 & 0 \\ u_1 & u_2 & 0 \end{bmatrix}}_{\mathcal{D}^T} \underbrace{\begin{bmatrix} 0 & u_2 & u_1 \\ 0 & 0 & u_2 \\ 0 & 0 & 0 \end{bmatrix}}_{\mathcal{D}}$$

For dimension n x n,

The vector $\mathbf{u} = (u_0, u_1, \ldots, u_{n-1})$

$$u_k = (\widehat{\mathbf{W}}^{-1})_{k+1,1} = (-1)^k \frac{W^{\frac{2k^2-(2n-1)k+n(n-1)}{2}}}{\prod_{s=1}^{n-k-1} (W^s - 1) \prod_{s=1}^{k} (W^s - 1)}$$

and

This is the way, we can calculate $\hat{W}^{-1}$.

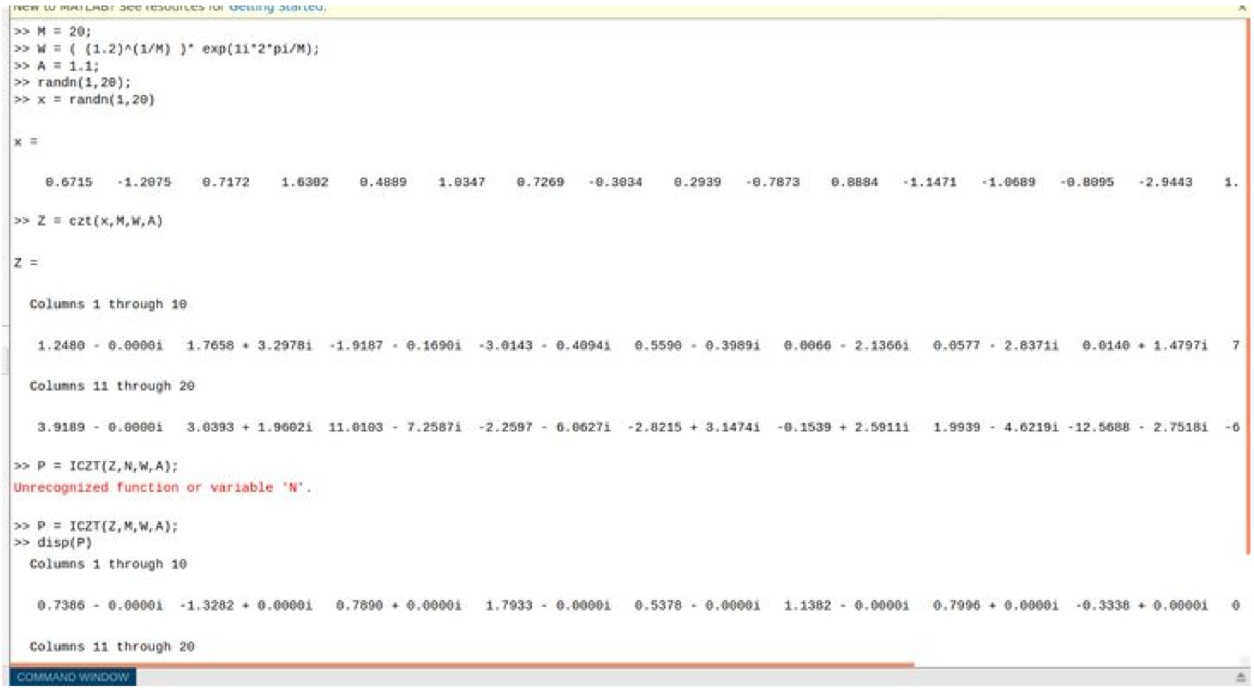
After calculating $\hat{W}^{-1}$, we just have to multiply all the matrices.
The diagonal matrices can be multiplied in O(n).
The multiplication with a symmetric Toeplitz matrix can be carried out using FFT and IFFT in O(nlogn) time.

Thus, the overall complexity of calculating ICZT is O(n logn).

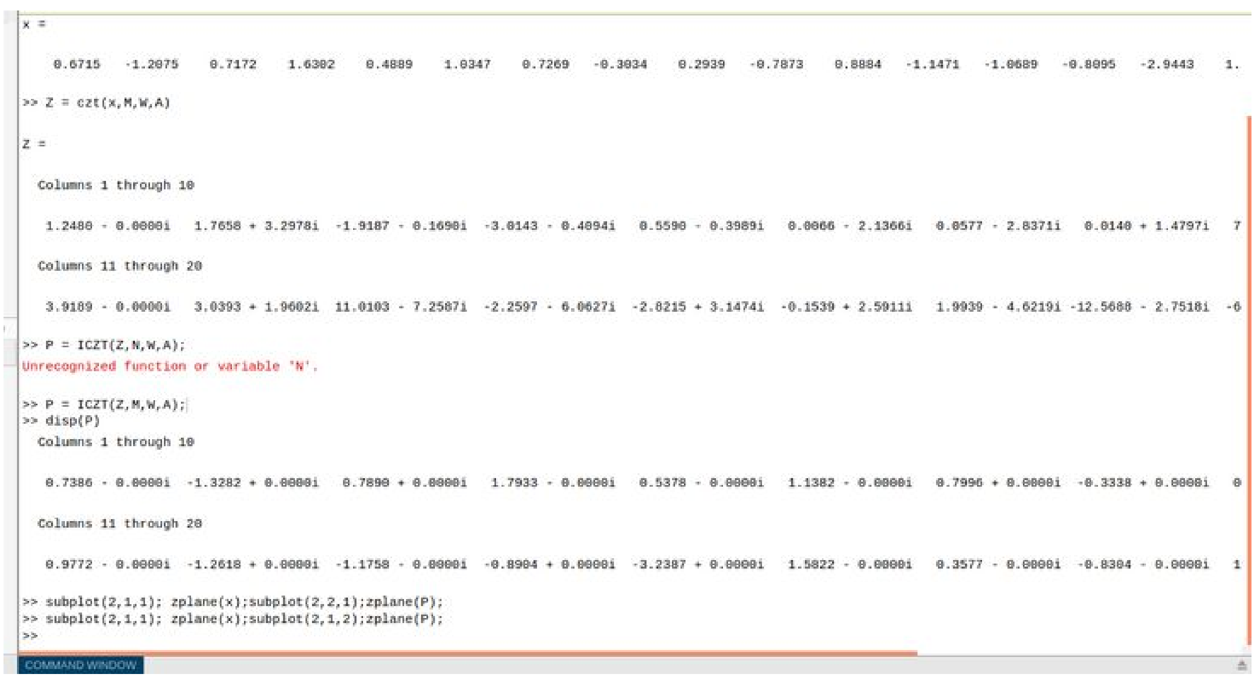
Some extra comments on calculations

One thing to be noted here, that the above algorithm for calculating ICZT is valid only when M = N, i.e when the length of both input and output vectors are the same.

Plot

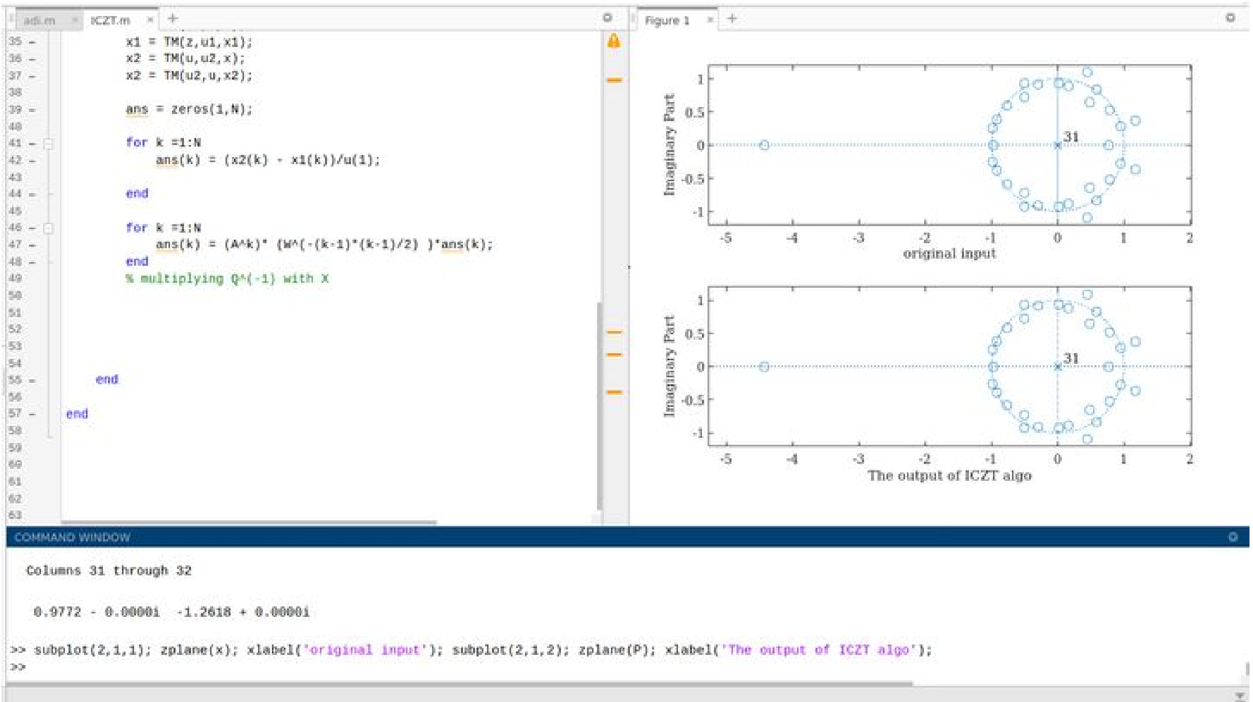


```
New to MATLAB? See resources for Getting Started.
>> M = 20;
>> W = ( (1.2)^(1/M) )* exp(1i*2*pi/M);
>> A = 1.1;
>> randn(1,20);
>> x = randn(1,20)

x =

    0.6715   -1.2075    0.7172    1.6302    0.4889    1.0347    0.7269   -0.3034    0.2939   -0.7873    0.8884   -1.1471   -1.0689   -0.8095   -2.9443    1.

>> Z = czt(x,M,W,A)

Z =

  Columns 1 through 10

   1.2480 - 0.0000i   1.7658 + 3.2978i  -1.9187 - 0.1690i  -3.0143 - 0.4094i   0.5590 - 0.3989i   0.0066 - 2.1366i   0.0577 - 2.8371i   0.0140 + 1.4797i   7

  Columns 11 through 20

   3.9189 - 0.0000i   3.0393 + 1.9602i  11.0103 - 7.2587i  -2.2597 - 6.0627i  -2.8215 + 3.1474i  -0.1539 + 2.5911i   1.9939 - 4.6219i -12.5688 - 2.7518i  -6

>> P = ICZT(Z,N,W,A);
Unrecognized function or variable 'N'.

>> P = ICZT(Z,M,W,A);
>> disp(P)
  Columns 1 through 10

   0.7386 - 0.0000i  -1.3282 + 0.0000i   0.7890 + 0.0000i   1.7933 - 0.0000i   0.5378 - 0.0000i   1.1382 - 0.0000i   0.7996 + 0.0000i  -0.3338 + 0.0000i   0

  Columns 11 through 20
COMMAND WINDOW
```



```
x =

    0.6715   -1.2075    0.7172    1.6302    0.4889    1.0347    0.7269   -0.3034    0.2939   -0.7873    0.8884   -1.1471   -1.0689   -0.8095   -2.9443    1.

>> Z = czt(x,M,W,A)

Z =

  Columns 1 through 10

   1.2480 - 0.0000i   1.7658 + 3.2978i  -1.9187 - 0.1690i  -3.0143 - 0.4094i   0.5590 - 0.3989i   0.0066 - 2.1366i   0.0577 - 2.8371i   0.0140 + 1.4797i   7

  Columns 11 through 20

   3.9189 - 0.0000i   3.0393 + 1.9602i  11.0103 - 7.2587i  -2.2597 - 6.0627i  -2.8215 + 3.1474i  -0.1539 + 2.5911i   1.9939 - 4.6219i -12.5688 - 2.7518i  -6

>> P = ICZT(Z,N,W,A);
Unrecognized function or variable 'N'.

>> P = ICZT(Z,M,W,A);
>> disp(P)
  Columns 1 through 10

   0.7386 - 0.0000i  -1.3282 + 0.0000i   0.7890 + 0.0000i   1.7933 - 0.0000i   0.5378 - 0.0000i   1.1382 - 0.0000i   0.7996 + 0.0000i  -0.3338 + 0.0000i   0

  Columns 11 through 20

   0.9772 - 0.0000i  -1.2618 + 0.0000i  -1.1758 - 0.0000i  -0.8904 + 0.0000i  -3.2387 + 0.0000i   1.5822 - 0.0000i   0.3577 - 0.0000i  -0.8304 - 0.0000i   1

>> subplot(2,1,1); zplane(x);subplot(2,2,1);zplane(P);
>> subplot(2,1,1); zplane(x);subplot(2,1,2);zplane(P);
>>
COMMAND WINDOW
```

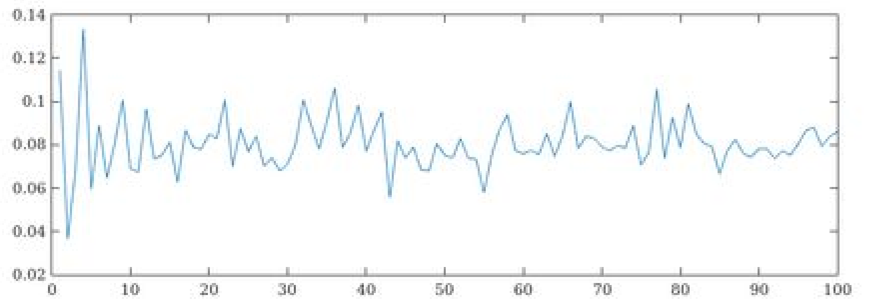
The above photos show the code for a random vector x of size 20. We calculated its czt for A = 1.1 and W = ((1.2)^(1/20)) * exp(i*2*pi/20). Then, we tried getting back the original signal using ICZT function.



Checking for error - The plots look similar. We calculated the error between the two plots.

```
>> err = x-P

err =

  Columns 1 through 10

  -0.0538 + 0.0000i  -0.1834 + 0.0000i   0.2259 - 0.0000i  -0.0862 + 0.0000i  -0.0319 - 0.0000i   0.1308 - 0.0000i   0.0434 - 0.0000i  -0.0343 - 0.0000i  -0

  Columns 11 through 20

   0.1350 - 0.0000i  -0.3035 + 0.0000i  -0.0725 + 0.0000i   0.0063 - 0.0000i  -0.0715 + 0.0000i   0.0205 - 0.0000i   0.0124 - 0.0000i  -0.1490 + 0.0000i  -0

  Columns 21 through 30

  -0.0671 + 0.0000i   0.1207 - 0.0000i  -0.0717 + 0.0000i  -0.1630 - 0.0000i  -0.0489 - 0.0000i  -0.1035 + 0.0000i  -0.0727 - 0.0000i   0.0303 + 0.0000i  -0

  Columns 31 through 32

  -0.0888 + 0.0000i   0.1147 - 0.0000i

>> err = abs(err);
>> error = sum(err)/M;
>> disp(error);
    0.1084

>>
```

The average deviation is 0.1084.

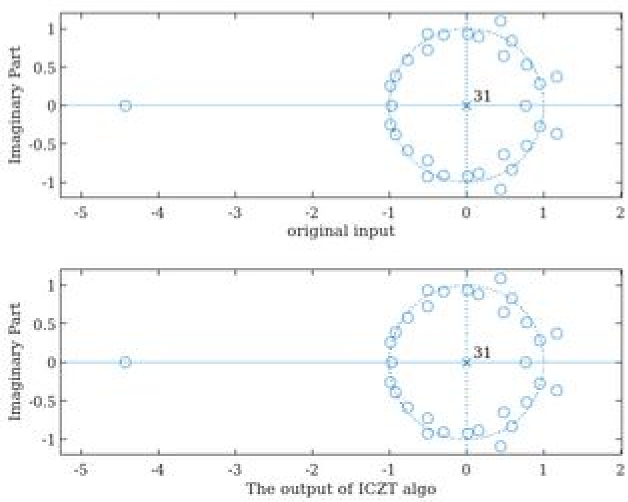


The average deviation of output of ICZT function from the original function x as we vary N from 0 to 100.

We varied the number of points from 0 to 100 and found the absolute error.

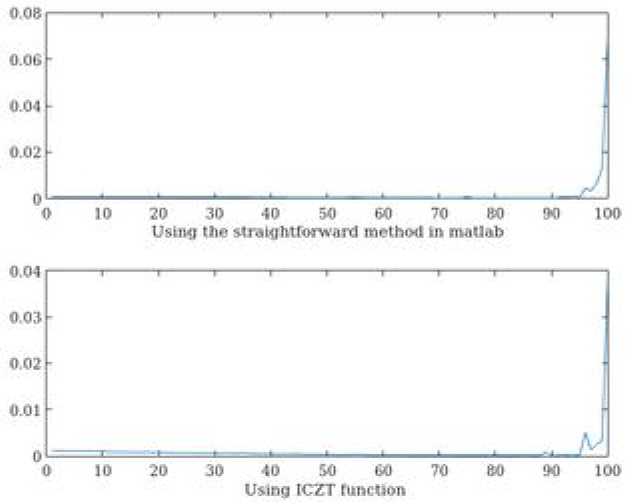
Using the straightforward approach of calculating inverse

We used the ‘inv’ function of matlab to calculate the inverse of W.



We are getting the same output as before.

Let’s compare their time complexity.



Here, we had varied number of elements from 1 to 100 and calculated the time in which the ICZT and iczt_straight_forward (refer the codes folder for these functions) functions worked. We used the tic-toc method.

Final Conclusions

From the above observations, we conclude that our algorithm for finding ICZT works well.

It has a good efficiency over the straight forward approach.

As number of points increase, the absolute error between the original vector x and the one we get as output form ICZT, decreases.

It is a good algorithm for calculating ICZT but the only limitation is that the input vector and the output vector should have the same size.

Special Notes:

References

1. https://www.nature.com/articles/s41598-019-50234-9
2. https://www.nature.com/articles/s41598-020-60878-7
3. https://in.mathworks.com/help/signal/ref/czt.html
4. https://krex.k-state.edu/dspace/bitstream/handle/2097/7844/LD2668R41972S43.pdf