T-ALGEBRA: AN IMPLEMENTATION OF MATRIX PARADIGM/PACKAGE OVER A NOVEL SEMISIMPLE COMMUTATIVE ALGEBRA

LIANG LIAO, STEPHEN JOHN MAYBANK AND SHEN LIN

LIAOLIANGIS@126.COM (L. LIAO),
SJMAYBANK@DCS.BBK.AC.UK (S. J. MAYABNK), LINS16@126.COM

Contents

1. Introduction

In the big-data deluge era, the canonical matrix and tensor paradigm over an algebraically closed field plays an essential role in many areas, including but not limited to machine learning, computer vision, pattern analysis, and statistic inference. Under the canonical matrix and tensor paradigm, observed data are given in the form of high-order arrays of canonical scalars (i.e., real or complex numbers). For example, an RGB image is a real number array of order three, two orders for the image's spatial measures, and a third for the image's spectral measures. An RGB image is also said to have three modes or three-way. A color video sequence of images is of order four, with three orders for spatial-spectral measures and the fourth-order chronological tempo.

Therefore, it is a natural question of whether there exists an extension of the field $\mathbb{C}$ over which a generalized matrix and tensor paradigm can be established and backward-compatible to the canonical paradigm over a field. Fortunately, the answer is yes, but one had to sacrifice at least one of the axioms of a field to obtain something extended.

Among these efforts trying to generalizing the field of complex numbers, well-known is Hamilton's $\mathbb{H}$ of quaternions, which, up to isomorphism, is a real division subalgebra of the matrix algebra $M_2(\mathbb{C})$ [Ham48, Roz88, HN11]. However, the multiplication of quaternions is not commutative.

Most hypercomplex number systems, including Hamilton's quaternions, are all subalgebras of Clifford algebra, and the fruits of generating complex numbers to obtain something extended. However, Clifford algebra's hypercomplex number systems are not suitable for general data analytics partially because they are either non-commutative or incompatible with many canonical notions such as euclidean norms. These hypercomplex number systems so far only find narrow niches in geometry and geometry-related branches of physics and computer sciences [Hes03, AS+04].

2. Functions for T-scalars

A matrix over C is called a tmatrix. In this article, its canonical counterpart over complex numbers is called a "canonical matrix," or just a "matrix."

The matrix paradigm over t-algebra C generalizes many notions of canonical matrices over complex numbers. In the following, we show how to use these t-matrix generalizations with the their MATLAB implementations.

2.1. t-scalar product: `tproduct`. The following function `tproduct` computes the t-scalar product of two t-scalars `tscalar01` and `tscalar02`.

The multi-way circular convolution can compute the product of two scalars in the spatial domain. The product can be equivalently obtained in the Fourier domain by the Hadamard product. The computation in the Fourier domain is more efficient and is adopted in the following function `tproduct`.

FUNCTION 1. **tproduct.m**: t-scalar product of two sclalars

```
1 function result = tproduct(tscalar01, tscalar02)
2   %This function computes the product of two t-scalars
3   assert(isequal(size(tscalar01), size(tscalar02)));
4 
5   %T-scalar product computed in the fourier domain by the
6   %Hadamard product
7   result = ifftn(fftn(tscalar01) .* fftn(tscalar02));
8 end
```

The input arguments `tscalar01` and `tscalar02` are complex arrays of the same shape. The line confirms their shapes (i.e., "`size`" by the MATLAB colloquialism) are identical.

In line 4, multi-way Fourier transform `fftn` and its inverse transform `ifftn` are invoked. The multi-way Fourier transform and its inverse transform can be equivalently computed by MATLAB's single-way Fourier transform `fft` and is demonstrated as follows.

FUNCTION 2. Demo: multi-way Fourier transform `fftn` implemented by single-way Fourier transforms `fft`

```
function fourier_transformed_tscalar = fftn(tscalar)
  assert(isnumeric(tscalar));
  number_of_index_axes = ndims(tscalar);
  %Compute single-way fourier transform along each index axis.
  fourier_transformed_tscalar = tscalar;
  for i = 1: number_of_index_axes
    fourier_transformed_tscalar = fft(fourier_transformed_tscalar, [], i);
  end
end
```

Similarly, the multi-way inverse transform `ifftn` is computed by single-way inverse transforms `ifft` along different axes as follows.

FUNCTION 3. Demo: multi-way inverse Fourier transform `ifftn` implemented by single-way Fourier transforms `ifft`

```
function tscalar = ifftn(fourier_transformed_tscalar)
  assert(isnumeric(fourier_transformed_tscalar));
  number_of_index_axes = ndims(fourier_transformed_tscalar);
```

```
  %Compute single-way inverse fourier transform along each index axis.
  tscalar = fourier_transformed_tscalar;
  for i = 1: number_of_index_axes
    tscalar = fft(tscalar, [], i);
  end
end
```

SYNTAX

```
C = tproduct(A, B)
```

DESCRIPTION

`C = tproduct(A, B)` returns the t-scalar product of the two t-scalars `A` and `B`. The arguments `A` and `B` and the returned result `C` are all numerical arrays of the same size.

EXAMPLES

```
>> A = [1 1 1 1];
>> B = [2 2 2 2];
>> C = tproduct(A,B)
C =
     8     8     8     8


%---------------------------------------------
% When A and B are two 3*3 random real arrays
>> A = rand(3)
A =
    0.4984    0.5853    0.2551
    0.9597    0.2238    0.5060
    0.3404    0.7513    0.6991

>> B = rand(3)
B =
    0.8909    0.1386    0.8407
    0.9593    0.1493    0.2543
    0.5472    0.2575    0.8143

>> C = tproduct(A, B)
C =
    2.4311    2.5358    2.7127
    2.7564    2.5265    2.6006
```

```
    2.8999    2.4472    2.4721

%------------------------------------------------
% When A and B are two 2*3 random complex arrays
>> A = randn(2, 3) + i * randn(2, 3)
A =
   0.5377 - 0.4336i  -2.2588 + 3.5784i   0.3188 - 1.3499i
   1.8339 + 0.3426i   0.8622 + 2.7694i  -1.3077 + 3.0349i

>> B = randn(2, 3) + i * randn(2, 3)
B =
   0.7254 + 1.4090i   0.7147 + 0.6715i  -0.1241 + 0.7172i
  -0.0631 + 1.4172i  -0.2050 - 1.2075i   1.4897 + 1.6302i

>> C = tproduct(A, B)
C =
  -0.0499 + 6.6934i -15.9137 + 1.0122i  -0.5178 + 0.7970i
  -14.5318 + 6.7276i  -7.0977 - 2.0930i   1.2429 + 6.9517i

%-------------------------------------------
% When A and B are 2*2*3 random real arrays
>> A = randn(2, 2, 3)
A(:,:,1) =
   -0.0068   -0.7697
    1.5326    0.3714



A(:,:,2) =
   -0.2256   -1.0891
    1.1174    0.0326



A(:,:,3) =
    0.5525    1.5442
    1.1006    0.0859



>> B = randn(2, 2, 3)

B(:,:,1) =
   -0.1774    1.4193
```

```
   -0.1961    0.2916


B(:,:,2) =
    0.1978   -0.8045
    1.5877    0.6966


B(:,:,3) =
    0.8351    0.2157
   -0.2437   -1.1658

>> C = tproduct(A, B)

C(:,:,1) =
   -1.3426   -1.0037
    4.3985    5.0231


C(:,:,2) =
    2.0206    1.8076
   -1.9653   -1.3698


C(:,:,3) =
    3.5530   -0.9110
    1.6056   -0.5372
```

Multi-way Fourier Transform and Inverse Transform. Liang Liao and Stephen John Maybank give a rigorous definition of multi-way Fourier transform and inverse transform in [LM20a, LM20b]. For convenience, the definition is organized as follows.

Definition 2.1 (Multi-way Fourier transform)**.** The Fourier transform is a linear isomorphism defined by the N-mode multiplication of tensors, which sends each element $X_T \in C \equiv \mathbb{C}^{I_1 \times \cdots \times I_N}$ to $\tilde{X}_T \in C \equiv \mathbb{C}^{I_1 \times \cdots \times I_N}$ as follows.

$$\tilde{X}_T \doteq F(X_T) \doteq X_T \times_1 W_{mat}^{(I_1)} \cdots \times_k W_{mat}^{(I_n)} \cdots \times_N W_{mat}^{(I_N)} \in C \equiv \mathbb{C}^{I_1 \times \cdots \times I_N} \tag{2.1}$$

where $W_{mat}^{(I_n)} \in \mathbb{C}^{I_n \times I_n}$ denotes the $I_n \times I_n$ Fourier matrix for all $n \in [N]$.

The (m_1, m_2)-th complex entry of the matrix $W_{mat}^{(I_n)}$ is given by

$$\left(W_{mat}^{(I_n)}\right)_{m_1,m_2} = e^{2\pi i\cdot(m_1-1)\cdot(m_2-1)\cdot I_k^{-1}} \in \mathbb{C}\ , \quad \forall\, (m_1, m_2) \in [I_n] \times [I_n]\ . \tag{2.2}$$

The inverse multi-way transform $F^{-1} : \tilde{X}_T \mapsto X_T$ is given by the following N-mode multiplication for tensors as follows.

$$X_T \doteq F^{-1}(\tilde{X}_T) = \tilde{X}_T \times_1 \left(W_{mat}^{(I_1)}\right)^{-1} \cdots \times_n \left(W_{mat}^{(I_n)}\right)^{-1} \cdots \times_N \left(W_{mat}^{(I_N)}\right)^{-1} \in C \equiv \mathbb{C}^{I_1\times\cdots\times I_N} \tag{2.3}$$

where $\left(W_{mat}^{(I_n)}\right)^{-1}$ denotes the inverse of the matrix $W_{mat}^{(I_n)}$ for all $n \in [N]$.

It is possible to employ a different multi-way transform and inverse transform to define a variant t-algebra C. Nevertheless, we only discuss the proposed algebra based on multi-way Fourier transforms or equivalently multi-way circular convolution. Interested readers are referred to [LM20a, LM20b] to connect the multi-way circular convolution and Fourier transform.

2.2. Fourier Matrix: `fourier_matrix`.

For the MATLAB enthusiasts, who only care about getting their applications done quickly, the following function gives the $n \times n$ Fourier matrix.

FUNCTION 4. **fourier_matrix.m**: function returns a $n \times n$ Fourier transform

```
function fourier_matrix_result = fourier_matrix(n)
  % this function computes the nxn Fourier matrix
  assert(isscalar(n));
  assert(n > 0);
  fourier_matrix_result = fft(eye(n), [], 1);
end
```

SYNTAX

```
Y = fourier_matrix(n)
```

DESCRIPTION

The input argument `n` is a positive integer. The result `Y` is a $n \times n$ square complex matrix (i.e., Fourier matrix), whose (m_1, m_2)-th entry is given by equation (2.2).

EXAMPLES

```
%------------------------
% The 3*3 fourier matrix
>> fourier_matrix(3)
ans =
   1.0000 + 0.0000i   1.0000 + 0.0000i   1.0000 + 0.0000i
```

```
   1.0000 + 0.0000i  -0.5000 - 0.8660i  -0.5000 + 0.8660i
   1.0000 + 0.0000i  -0.5000 + 0.8660i  -0.5000 - 0.8660i

%-----------------------
% The 2*2 fourier matrix
>> fourier_matrix(2)
ans =
     1     1
     1    -1
```

Remark 2.2 (Shape of a Fourier matrix)**.** A Fourier matrix must be square.

2.3. Inverse Fourier Matrix: ifourier_matrix.

The inverse Fourier transform is given by the following code.

FUNCTION 5. **ifourier_matrix**: function returns a $n \times n$ Fourier transform

```
function ifourier_matrix_result = ifourier_matrix(n)
  % this function computes the nxn inverse Fourier matrix
  ifourier_matrix_result = conj(fourier_matrix(n)) / n;
end
```

SYNTAX

```
Y = ifourier_matrix(n)
```

DESCRIPTION

The input argument n is a positive integer. The result Y is a $n \times n$ square complex matrix (i.e., inverse Fourier matrix). The result returned by ifourier_matrix(n) is the inverse matrix of the result by fourier_matrix(n).

Let the $n \times n$ Fourier matrix be $X_{mat} \doteq W_{mat}^{(I_n)}$ and its inverse matrix be $Y_{mat} \doteq \left(W_{mat}^{(I_n)}\right)^{-1}$. Besides the equality $X_{mat} \cdot Y_{mat} = Y_{mat} \cdot X_{mat} = I_{mat}$, the following equality holds

$$\begin{aligned}(Y_{mat})_{m_1,m_2} &= (1/n) \cdot \overline{(X_{mat})_{m_1,m_2}} \\ &= (1/n) \cdot e^{-2\pi i \cdot (m_1-1)\cdot(m_2-1)\cdot I_k^{-1}} \quad , \quad \forall (m_1, m_2) \in [I_n] \times [I_n]\end{aligned} \tag{2.4}$$

Thus, FUNCTION 5 is an efficient way of computing an inverse Fourier matrix.

EXAMPLES

```
%--------------------------------
% The 3*3 inverse fourier matrix
>> ifourier_matrix(3)
ans =
```

```
0.3333 + 0.0000i   0.3333 + 0.0000i   0.3333 + 0.0000i
0.3333 + 0.0000i  -0.1667 + 0.2887i  -0.1667 - 0.2887i
0.3333 + 0.0000i  -0.1667 - 0.2887i  -0.1667 + 0.2887i
```

2.4. Identity T-scalar: E_T. The function returns the identity t-scalar of a given size is given as follows.

SYNTAX

```
E = E_T(tsize)
```

DESCRIPTION

The input argument `tsize` is a row array of positive integers. The result is an array whose inceptional entry is equal to 1, and the other entries are equal to 0.

EXAMPLES

```
%------------------------
% E_t with the tsize [3, 3]
>> E_T([3, 3])
ans =
     1     0     0
     0     0     0
     0     0     0

%--------------------------------------------------------
% multi-way Fourer transform of the 3*3 identity t-scalar
>> fftn(E_T([3, 3]))
ans =
     1     1     1
     1     1     1
     1     1     1

%----------------------------
% E_T with the tsize [2, 3]
>> E_T([2, 3])
ans =
     1     0     0
     0     0     0

%--------------------------------------------------------
% multi-way Fourer transform of the 3*3 identity t-scalar
>> fftn(E_T([2, 3]))
ans =
```

```
     1      1      1
     1      1      1
```

2.5. Zero T-scalar: Z_T. This function returns the zero t-scalar of a given size.

SYNTAX

```
Y = Z_T(tsize)
```

DESCRIPTION

The input argument `tsize` is a row array of positive integers. The result is an array of zeros. When `tsize` is a single positive integer, the result is a square array of zeros.

EXAMPLES

```
%-----------------------
% The 3*3 zero t-scalar
>> Z_T([3, 3])
ans =
     0      0      0
     0      0      0
     0      0      0

%-----------------------------------------------------
% When the input argument is a single postive integer,
% the output is a square array of zeros
>> Z_T(2)
ans =
     0      0
     0      0

%-----------------------
% The 1*3 zero t-scalar
>> Z_T([1, 3])
ans =
     0      0      0

%-------------------------
% The 2*3*2 zero t-scalar
>> Z_T([2, 3, 2])
ans(:,:,1) =

     0      0      0
     0      0      0
```

```
ans(:,:,2) =

     0     0     0
     0     0     0
```

2.6. PSEUDO-INVERSE OF A T-SCLAAR: `tpinv_tscalar`. This function returns the pseudo-inverse of a t-scalar.

SYNTAX

```
Y = tpinv_tscalar(tscalar)
```

DESCRIPTION

The input argument `tscalar` is a numeric array. The output t-sclalar `Y` is a numeric array of the same size. The multiplication of `tscalar` and `Y` yields a idempotent t-scalar, which is also the generalized rank of both `tscalar` and `Y`.

EXAMPLES

```
%-------------------------------------------------------
% The pseudo-inverse of the identity t-scalar is itself.
>>  tpinv_tscalar(E_T(3))
ans =
     1     0     0
     0     0     0
     0     0     0

%-------------------------------------------------------
% The pseudo-inverse of the zero t-scalar is still zero.
>> tpinv_tscalar(Z_T(3))
ans =
     0     0     0
     0     0     0
     0     0     0
```

Mathematically, the following equality holds

$$\begin{aligned} E_T^+ \equiv E_T^{-1} \equiv E_T \\ Z_T^+ \equiv Z_T \quad . \end{aligned} \tag{2.5}$$

where $(\cdot)^+$ denotes the psuedo-inverse of a t-scalar, and $(\cdot)^{-1}$ denotes the inverse of a t-scalar.

The zero t-scalar Z_T is non-invertible and the identity t-scalar E_T is invertible. Any t-scalar is pseudo-invertible [LM20a, LM20b].

2.7. NORM AND ANGLE OF A T-SCALAR: `tnorm_angle`. Function `tnorm_angle` returns the norm (i.e., generalized absolute value, a nonnegative t-scalar) and the generalized angle (a nonnegative t-scalar) of the t-scalar.

SYNTAX

```
[tnorm, tangle] = tnorm_angle(tscalar)
```

DESCRIPTION

The input argument `tscalar` and output arguments `tnorm`, `tangle` are also numeric arrays of the same size.

EXAMPLES

```
>> tscalar = randn(3)
tscalar =
    0.5377    0.8622   -0.4336
    1.8339    0.3188    0.3426
   -2.2588   -1.3077    3.5784


>> [tnorm, tangle] = tnorm_angle(tscalar)
tnorm =
    4.5066   -0.9511   -0.9511
   -0.5076   -0.0028    0.9449
   -0.5076    0.9449   -0.0028

tangle =
   0.0000 + 0.0000i   0.0000 + 0.0686i   0.0000 - 0.0686i
   0.0000 - 0.7277i   0.0000 + 0.4253i   0.0000 - 0.6812i
   0.0000 + 0.7277i   0.0000 + 0.6812i   0.0000 - 0.4253i
```

The multi-way Fourier transform of the norm of a t-scalar is an array of nonnegative real numbers. See the following example.

```
>> fftn(tnorm)
ans =
    3.4734    3.5006    3.5006
    2.1698    7.8579    5.0149
    2.1698    5.0149    7.8579
```

The multi-way Fourier transform of the abgle of a t-scalar is an array of real numbers. See the following example.

```
>> fftn(tangle)
ans =
    0.0000 + 0.0000i    2.0351 - 0.0000i   -2.0351 - 0.0000i
   -1.7036 - 0.0000i   -1.8782 + 0.0000i   -0.1993 + 0.0000i
    1.7036 - 0.0000i    0.1993 + 0.0000i    1.8782 + 0.0000i
```

2.8. CONJUGATE OF A T-SCALAR: `tconj_tscalar`. Function `tconj_tscalar` returns the conjugate of a given t-scalar.

SYNTAX

```
conj_tscalar = tconj_tscalar(tscalar)
```

DESCRIPTION

The input argument `tscalar` and output arguments `conj_tscalar` are also numeric arrays of the same size.

EXAMPLES

```
>> X = randn(3, 2) + i * randn(3, 2)
X =
    0.0774 + 0.3714i  -0.0068 - 1.0891i
   -1.2141 - 0.2256i   1.5326 + 0.0326i
   -1.1135 + 1.1174i  -0.7697 + 0.5525i

>> Y = tconj_tscalar(X)
Y =
    0.0774 - 0.3714i  -0.0068 + 1.0891i
   -1.1135 - 1.1174i  -0.7697 - 0.5525i
   -1.2141 + 0.2256i   1.5326 - 0.0326i

>> [fftn(X), fftn(Y)]
ans =
  Columns 1 through 2

   -1.4941 + 0.7592i  -3.0064 + 1.7671i
   -0.7605 - 3.3628i   0.9168 + 3.3881i
    2.4662 + 0.4506i   2.3422 - 0.7739i

  Columns 3 through 4

   -1.4941 - 0.7592i  -3.0064 - 1.7671i
   -0.7605 + 3.3628i   0.9168 - 3.3881i
```

```
  2.4662 - 0.4506i   2.3422 + 0.7739i
```

Remark 2.3. From the above example, it shows `fftn(X)` is the conjugate array of `fftn(Y)`.

In the following example, `A` is a random t-scalar, and `C` is a so-called nonnegative t-scalar.

```
>> A = randn(3) + i * randn(3)
A =
 -0.8095 + 0.3192i   0.3252 - 0.0301i  -1.7115 + 1.0933i
 -2.9443 + 0.3129i  -0.7549 - 0.1649i  -0.1022 + 1.1093i
  1.4384 - 0.8649i   1.3703 + 0.6277i  -0.2414 - 0.8637i

>> C = tproduct(A, tconj_tscalar(A))
C =
 21.4860 - 0.0000i   4.5405 + 6.1719i   4.5405 - 6.1719i
 -4.4911 - 1.2003i   0.0077 + 2.2780i  -3.7334 + 3.5056i
 -4.4911 + 1.2003i  -3.7334 - 3.5056i   0.0077 - 2.2780i

>> fftn(C)
ans =
 14.1333 - 0.0000i  20.2526 + 0.0000i   3.1253 - 0.0000i
 46.7224 - 0.0000i  18.6277 - 0.0000i   6.3444 + 0.0000i
 30.8452 - 0.0000i  44.0261 - 0.0000i   9.2967 - 0.0000i
```

Remark 2.4. The above example shows the Fourier transform of a nonnegative t-scalar is an array of nonnegative real numbers.

2.9. REAL PART OF A T-SCALAR: `treal_part_tscalar`. For all t-scalar $X_T \in C$, the t-scalar in the form of $\frac{X_T + X_T^*}{2}$ is called the real part of the t-scalar X_T.

It is easy to follow that the following equality holds for all $X_T \in C$,

$$\left(\frac{X_T + X_T^*}{2}\right)^* \equiv \frac{X_T + X_T^*}{2} \tag{2.6}$$

The real part of a t-scalar is self-conjugate. Namely, $\frac{X_T + X_T^*}{2} \in C^{sc}$, $\forall\, X_T \in C$, where C^{sc} denotes the subalgebra of C [LM20a, LM20b].

SYNTAX

```
real_part = treal_part_tscalar(tscalar)
```

DESCRIPTION

The input argument `tscalar` and output argument `real_part` are both numeric arrays of the same size, where `tscalar` can be any t-scalar in C and `real_part` is the so-called real part of `tscalar` and a self-conjugate t-scalar in C^{sc}.

EXAMPLES

```
>> A = randn(3)
A =
    1.1006   -1.4916    2.3505
    1.5442   -0.7423   -0.6156
    0.0859   -1.0616    0.7481

>> B = treal_part_tscalar(A)
B =
    1.1006    0.4294    0.4294
    0.8151    0.0029   -0.8386
    0.8151   -0.8386    0.0029

>> C = tconj_tscalar(B)
C =
    1.1006    0.4294    0.4294
    0.8151    0.0029   -0.8386
    0.8151   -0.8386    0.0029

>> fftn(B)
ans =
    1.9182    3.1371    3.1371
    1.9800   -1.8240    0.7005
    1.9800    0.7005   -1.8240
```

Remark 2.5. It follows from the above examples that `C` is identical to `B`, namely, `B` is self-conjugate. The Fourier transform of `B`, (i.e., a self-conjugate t-scalar) is an array of real numbers.

2.10. IMAGINARY PART OF A T-SCALAR: `timg_part_tscalar`. For all t-scalar $X_T \in C$, the t-scalar in the form of $\frac{X_T - X_T^*}{2i}$ is called the imaginary part of the t-scalar X_T.

It is easy to follow that the following equality holds for all $X_T \in C$,

$$\left(\frac{X_T - X_T^*}{2i}\right)^* \equiv \frac{X_T - X_T^*}{2i} \tag{2.7}$$

The imaginary part of a t-scalar is self-conjugate. Namely, $\frac{X_T - X_T^*}{2i} \in C^{sc}$, $\forall\ X_T \in C$, where C^{sc} denotes the subalgebra of C [LM20a, LM20b].

SYNTAX

```
imag_part = timg_part_tscalar(tscalar)
```

DESCRIPTION

The input argument `tscalar` and output argument `imag_part` are both numeric arrays of the same size, where `tscalar` can be any t-scalar in C and `imag_part` is the so-called imaginary part of `tscalar` and a self-conjugate t-scalar in C^{sc}.

EXAMPLES

```
>> A = randn(3) + i * randn(3)
A =
   0.2916 - 1.1480i  -0.8045 + 2.5855i  -0.2437 - 0.0825i
   0.1978 + 0.1049i   0.6966 - 0.6669i   0.2157 - 1.9330i
   1.5877 + 0.7223i   0.8351 + 0.1873i  -1.1658 - 0.4390i

>> B = timg_part_tscalar(A)
B =
  -1.1480 + 0.0000i   1.2515 + 0.2804i   1.2515 - 0.2804i
   0.4136 + 0.6949i  -0.5529 - 0.9312i  -0.8728 + 0.3097i
   0.4136 - 0.6949i  -0.8728 - 0.3097i  -0.5529 + 0.9312i

>> C = tconj_tscalar(B)
C =
  -1.1480 + 0.0000i   1.2515 + 0.2804i   1.2515 - 0.2804i
   0.4136 + 0.6949i  -0.5529 - 0.9312i  -0.8728 + 0.3097i
   0.4136 - 0.6949i  -0.8728 - 0.3097i  -0.5529 + 0.9312i

>> fftn(B)
ans =
   -0.6694   -1.8103    1.5172
    2.4944   -0.7035   -2.8644
    2.2401   -3.2276   -7.3080
```

Remark 2.6. Since `C` is identical to `B`, `B` is self-conjugate, and the Fourier transform of `B` (i.e., a self-conjugate t-scalar) is an array of real numbers.

2.11. POWER OF A T-SCALAR: `tpower`. Function `tpower` returns the p power of a t-scalar, $Y_T = (X_T)^p$. When $p = \frac{1}{2}$, the function returns the arithmetic square root of a t-scalar $Y_T = \sqrt{X_T}$.

SYNTAX

```
tscalar_power = tpower(tscalar, p_value)
```

DESCRIPTION

The input argument `tscalar` and output arguments `tscalar_power` are numeric arrays of the same size. The argument `p_value` is a canonical scalar, i.e., a complex number. Usually, the value of `p_value` is a real number.

EXAMPLES

```
>> A = randn(3) + i * randn(3)
A =
   1.7119 - 1.9609i  -0.8396 + 2.9080i   0.9610 - 1.0582i
  -0.1941 - 0.1977i   1.3546 + 0.8252i   0.1240 - 0.4686i
  -2.1384 - 1.2078i  -1.0722 + 1.3790i   1.4367 - 0.2725i

>> B = tpower(A, 2)
B =
   9.3472 + 4.9465i   5.5966 + 2.8883i -16.1201 - 9.0661i
   6.2115 +10.0242i  18.5029 - 8.5377i -16.8333 - 0.8297i
 -10.9200 +16.9954i  14.1335 - 6.0189i  -8.1149 -10.5458i

>> fftn(A)
ans =

   1.3440 - 0.0535i   4.3826 - 2.3565i  -7.5884 - 7.6893i
   2.3033 - 2.7884i   3.8327 - 1.7446i   5.1228 - 4.2926i
   1.8525 + 2.5087i   7.0427 + 0.1217i  -2.8853 - 1.3535i

>> [fftn(A) .^2, fftn(B)]
ans =
   1.0e+02 *

  Columns 1 through 3
   0.0180 - 0.0014i   0.1365 - 0.2066i  -0.0154 + 1.1670i
  -0.0247 - 0.1285i   0.1165 - 0.1337i   0.0782 - 0.4398i
  -0.0286 + 0.0929i   0.4959 + 0.0171i   0.0649 + 0.0781i

  Columns 4 through 6
   0.0180 - 0.0014i   0.1365 - 0.2066i  -0.0154 + 1.1670i
  -0.0247 - 0.1285i   0.1165 - 0.1337i   0.0782 - 0.4398i
  -0.0286 + 0.0929i   0.4959 + 0.0171i   0.0649 + 0.0781i
```

Remark 2.7. The above example shows that the array `fftn(A).^2` is identical to the array `fftn(B)`.

More algebraically, for all t-scalar $X_T \in C$, let $Y_T \doteq X_T^p$ and $F(X_T) = \tilde{X}_T$ and $F(Y_T) = \tilde{Y}_T$. The following equality holds

$$\left((\tilde{X}_T)_{m_1,m_2}\right)^p = (\tilde{Y}_T)_{m_1,m_2} \ , \quad \forall m_1, m_2 \ . \tag{2.8}$$

When $p = 1/2$, one has the arithmetical square root of a t-scalar, which helps define the norm of a t-scalar or a t-matrix.

See the following examples.

```
>> C = tpower(A, 0.5)
C =
    1.7072 - 0.7392i  -0.5633 + 0.6266i   0.3416 + 0.1000i
    0.0274 - 0.0826i   0.2279 + 0.1667i   0.0453 - 0.0479i
   -0.2045 - 0.3783i  -0.5681 + 0.0536i   0.1459 + 0.2779i

>> [fftn(A) .^0.5, fftn(C)]
ans =
  Columns 1 through 3

    1.1595 - 0.0231i   2.1632 - 0.5447i   1.2678 - 3.0325i
    1.7205 - 0.8104i   2.0055 - 0.4350i   2.4296 - 0.8834i
    1.5766 + 0.7956i   2.6539 + 0.0229i   0.3884 - 1.7425i

  Columns 4 through 6
    1.1595 - 0.0231i   2.1632 - 0.5447i   1.2678 - 3.0325i
    1.7205 - 0.8104i   2.0055 - 0.4350i   2.4296 - 0.8834i
    1.5766 + 0.7956i   2.6539 + 0.0229i   0.3884 - 1.7425i
```

2.12. RANDOM T-SCALAR: `randn_tscalar`. For the convenience of demonstration, we give function `randn_tscalar`, which returns a random t-scalar.

SYNTAX

```
random_tscalar = randn_tscalar(tsize, mode)
```

DESCRIPTION

The input argument `tsize` is a row numeric array. The argument `mode` is either `'real'` or `'complex'`. If only one argument `tsize` is given to function `randn_tscalar`, the second argument is set the default value `'complex'`.

EXAMPLES

```
>> randn_tscalar([2, 3])
```

```
ans =
   1.0984 - 1.5771i   0.7015 + 0.2820i  -0.3538 - 1.3337i
  -0.2779 + 0.5080i  -2.0518 + 0.0335i  -0.8236 + 1.1275i
```

```
>> randn_tscalar(3, 'real')
ans =
   -0.7145   -0.5890   -1.1201
    1.3514   -0.2938    2.5260
   -0.2248   -0.8479    1.6555
```

The following examples show that, for all t-scalar $X_T \in C$, the following equality holds [LM20a, LM20b]

$$X_T \circ X_T^* \equiv Re(X_T)^2 + Im(X_T)^2 \in S^{nonneg} \tag{2.9}$$

```
>> X = randn_tscalar(2)
X =
   0.3075 + 0.7914i  -0.8655 - 2.3299i
  -1.2571 - 1.3320i  -0.1765 - 1.4491i
```

```
>> [tproduct(X, tconj_tscalar(X)); ...
tpower(treal_part_tscalar(X), 2) + ...
tpower(timg_part_tscalar(X), 2)]

ans =
  12.3839 + 0.0000i   0.0841 + 0.0000i
   4.1764 - 0.0000i   5.9805 - 0.0000i
  12.3839 + 0.0000i   0.0841 + 0.0000i
   4.1764 + 0.0000i   5.9805 + 0.0000i
```

The following example shows that, for all $X_T \in C$, the t-scalar $X_T \circ X_T^*$ is nonnegative. In other words, $F(X_T \circ X_T^*)$ returns an array of nonnegative real numbers.

```
>> fftn(tproduct(X, tconj_tscalar(X)))

ans =
  22.6249 + 0.0000i  10.4956 - 0.0000i
   2.3111 + 0.0000i  14.1038 + 0.0000i
```

2.13. GENERALIZED EXPONENTIAL: `texp`. Generalized exponential of a t-scalar.

SYNTAX

```
exp_result = texp(tscalar)
```

DESCRIPTION

The input argument `tscalar` and the result `exp_result` are numeric arrays of the same size.

EXAMPLES

Calculate the exponential of 1, which is Euler's number, e.

```
>> texp(1)
ans =
    2.7183
```

Calculate the exponential of 0, which is equal to 1.

```
>> texp(0)
ans =
    1
```

Calculate the exponential of i, which is equal to $e^i \equiv \cos 1 + i \sin 1$.

```
>> texp(i)
ans =
    0.5403 + 0.8415i
```

Calculate the exponential of E_T of the size 3×3, i.e. $e^{E_T} = e \cdot E_T$, which generalizes Euler's number e.

```
>> texp(E_T(3))
ans =
    2.7183         0         0
         0         0         0
         0         0         0
```

Calculate the exponential of Z_T of the size 3×3, i.e. $e^{Z_T} = E_T$, which generalizes the canonical scalar 1.

```
>> texp(Z_T(3))
ans =
     1     0     0
     0     0     0
     0     0     0
```

Calculate the exponential of a random t-scalar of the size 2×3.

```
>> X = randn_tscalar([2, 3])
X =
```

```
   0.3335 + 0.8620i   0.4517 + 0.4550i   0.1837 - 0.3349i
   0.3914 - 1.3617i  -0.1303 - 0.8487i  -0.4762 + 0.5528i

>> texp(X)
ans =
   0.2069 + 0.4511i  -0.6209 - 0.7939i  -0.1891 + 0.4643i
   1.7240 - 1.0796i   0.8839 - 0.3436i  -0.3464 - 0.0270i
```

Remark 2.8 (Expotential of a t-scalar)**.** For all $X_T \in C$, the expotential of X_T is a t-scalar given by the following series

$$e^{X_T} \doteq \sum_{k=1}^{+\infty} \frac{X_T^{(k-1)}}{k-1} = E_T + X_T + \frac{X_T^2}{2} + \frac{X_T^3}{6} + \frac{X_T^4}{24} + \cdots \in C \ . \tag{2.10}$$

For all $X_T \in C$, let $F(X_T) = \tilde{X}_T$ and $F(e^{X_T}) = \tilde{Y}_T$. By the semisimplicity of C introduced in [LM20a], the following equality holds in the following domain

$$e^{\alpha} = \beta \in \mathbb{C} \tag{2.11}$$

where $\alpha \doteq (\tilde{X}_T)_{m_1,m_2} \in \mathbb{C}$ and $\beta \doteq (\tilde{Y}_T)_{m_1,m_2}$ for all m_1, m_2.

2.14. Natural logarithm of a t-scalar: `tlog`**.** Function `tlog` generalizes the natural logarithm of a canonical scalar.

SYNTAX

```
log_result = tlog(tscalar)
```

DESCRIPTION

The input `tscalar` and the output `log_result` are also numeric arrays of the same size.

References

[AS+04] Rafal Ablamowicz, Garret Sobczyk, et al., *Lectures on clifford (geometric) algebras and applications*, Springer, 2004.

[Ham48] William Rowan Hamilton, *On quaternions; or on a new system of imaginaries in algebra*, The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science **33** (1848), no. 219, 58–60.

[Hes03] David Hestenes, *Oersted medal lecture: Reforming the mathematical language of physics*, American Journal of Physics **71** (2003), no. 2, 104–121.

[HN11] Joachim Hilgert and Karl-Hermann Neeb, *Structure and geometry of lie groups*, ch. 2.3, Springer, 2011.

[LM20a] Liang Liao and Stephen John Maybank, *General data analytics with applications to visual information analysis: A provable backward-compatible semisimple paradigm over t-algebra*, arXiv preprint arXiv:2011.00307 (2020), 1–53.

[LM20b] ______, *Generalized visual information analysis via tensorial algebra*, Journal of Mathematical Imaging and Vision (2020), 560–584.

[Roz88] Boris Abramovich Rozenfel'd, *The history of non-euclidean geometry: Evolution of the concept of a geometric space*, p. 385, Springer, 1988.