PhasorArray Toolbox

A MATLAB Toolbox for Harmonic Domain Modeling and Control of Periodic Systems

Maxime Gruss
University of Lorraine

October 22, 2025

Contents

1 Introduction and Motivation

The harmonic modeling of systems with periodic coefficients, whether arising from AC/DC converter control, vibratory system modeling, or more generally from the analysis of Linear Time Periodic (LTP) systems, relies on the systematic use of truncated Fourier series.

1.1 Mathematical Foundation

For any scalar time-signal $x \in L^2_{\text{loc}}(\mathbb{R}, \mathbb{R})$ and a given period $T = 2\pi/\omega$, we define the *phasors* $X = \mathcal{F}(x)$ associated to x, by the time-varying sequence $X(t) = (X_k(t))_{k\in\mathbb{Z}}$ whose components result from a sliding Fourier decomposition:

$$X_k(t) = \frac{1}{T}\int_{t-T}^{t} x(\tau)e^{-\mathrm{j}k\omega\tau}d\tau \tag{1}$$

$\mathcal{F}(a)$ is then denoted

$$\mathcal{F}(a) = \begin{bmatrix} \vdots \\ a_{-2} \\ a_{-1} \\ a_0 \\ a_1 \\ a_2 \\ \vdots \end{bmatrix} \tag{2}$$

For a time vector-valued function $x \in L^2_{\text{loc}}(\mathbb{R}, \mathbb{R}^n)$ (or matrix-valued), this extends to:

$$X = \mathcal{F}(x) = (\mathcal{F}(x_1)^T, \cdots, \mathcal{F}(x_n)^T)^T \tag{3}$$

The Toeplitz transformation $\mathcal{T}$ of a scalar function $a \in L^\infty$ with phasors $(a_k)_{k\in\mathbb{Z}} = \mathcal{F}(a)$ is defined by the infinite dimensional Toeplitz operator bounded on ℓ^2:

$$\mathcal{T}(a) = \begin{bmatrix} \ddots & \vdots & \vdots & \vdots & \\ \cdots & a_0 & a_{-1} & a_{-2} & \cdots \\ \cdots & a_1 & a_0 & a_{-1} & \cdots \\ \cdots & a_2 & a_1 & a_0 & \cdots \\ & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{4}$$

A matrix-valued function $A = (a_{ij})_{i\in\{1,n\},j\in\{1,m\}}$ is represented by the Toeplitz-Block (TB) operator:

$$\mathcal{A} = (\mathcal{T}(a_{ij}))_{i\in\{1,n\},j\in\{1,m\}} \tag{5}$$

$$= \begin{bmatrix} \mathcal{T}(a_{11}) & \cdots & \mathcal{T}(a_{1m}) \\ \vdots & \ddots & \vdots \\ \mathcal{T}(a_{n1}) & \cdots & \mathcal{T}(a_{nm}) \end{bmatrix} \tag{6}$$

Equivalently, the Block-Toeplitz (BT) representation can be used, where each block is the matricial phasor of the T-periodic $A(t) = \sum_k A_k e^{\mathrm{j}k\omega t}$, organised in a toeplitz manner.

$$\mathcal{T}_{\text{BT}}(A) = \begin{bmatrix} \ddots & \vdots & \vdots & \vdots & \\ \cdots & A_0 & A_{-1} & A_{-2} & \cdots \\ \cdots & A_1 & A_0 & A_{-1} & \cdots \\ \cdots & A_2 & A_1 & A_0 & \cdots \\ & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{7}$$

Similarly for Fourier operation, both formalisme exists for matrix-valued functions or vector.

$$\mathcal{F}_{\mathrm{TB}} = \begin{bmatrix} \mathcal{F}(a_{11}) & \cdots & \mathcal{F}(a_{1m}) \\ \vdots & \ddots & \vdots \\ \mathcal{F}(a_{n1}) & \cdots & \mathcal{F}(a_{nm}) \end{bmatrix} \qquad \mathcal{F}_{\mathrm{BT}} = \begin{bmatrix} \vdots \\ A_{-1} \\ A_0 \\ A_1 \\ \vdots \end{bmatrix} \tag{8}$$

Both formalism are strictly equivalent and interchangeable via permutations, and all theorems hold in both cases. Consistency is although required : one should not mix TB matrix and BT Fourier for instance. In the toolbox `_bt` and `_tb` suffixes are used to differentiate the two representations.

Fundamental Properties: The Toeplitz structure enable representation of convolution operations (time domain product) as simple matrix-vector products in the harmonic domain. For a signal $x \in L^2_{\mathrm{loc}}$ and L^∞-matrix valued functions A and B:

$$\mathcal{F}(Ax) = \mathcal{T}(A)\mathcal{F}(x) = \mathcal{A}X \tag{9}$$

$$\mathcal{T}(AB) = \mathcal{T}(A)\mathcal{T}(B) = \mathcal{A}\mathcal{B} \tag{10}$$

Truncation Challenges: Due to the infinite-dimensional nature of harmonic representations, truncation is essential for numerical computations. However, naive truncation of products introduces errors:

$$\mathcal{T}(AB)_h \neq \mathcal{T}(A)_h\mathcal{T}(B)_h \tag{11}$$

where the subscript h denotes truncation to order h. This error can be corrected by adding appropriate boundary terms, making both approaches equivalent:

$$\mathcal{T}(AB)_h = \mathcal{T}(A)_h\mathcal{T}(B)_h + \text{correction terms} \tag{12}$$

The key insight is that the product of truncated operators is incorrect without these corrections, which can lead to convergence issues in numerical algorithms.

The harmonic modeling of systems with periodic coefficients, whether arising from AC/DC converter control, vibratory system modeling, or more generally from the analysis of Linear Time Periodic (LTP) systems, relies on the systematic use of truncated Fourier series.

In this context, many operations become both:

- **Tedious** to implement for each new project: handling harmonic indices, convolutions, matrix size alignment, Toeplitz constructions.
- **Systematic** and mathematically well-defined: sum, product, inversion, harmonic projections, temporal representations.

These characteristics make manual programming laborious, error-prone, and poorly readable. A simple addition of two periodic matrices or displaying their temporal realization can require between 5 and 40 lines of MATLAB code written by hand.

Toolbox Objective: *Provide an abstraction layer allowing users to focus solely on control logic, modeling, or analysis, while hiding the heavy operations related to harmonic signals.*

Benefits include:

- Code readability and conciseness are greatly improved.
- Development time is drastically reduced (observed examples: reduction by a factor of 3 to 5 on simulation projects).
- Code maintenance, robustness, and portability are facilitated.

Philosophy: A `PhasorArray` object is a *periodic signal or matrix* encapsulated in a 3D array. Classical operations are redefined:

- $A + B$ performs the temporal sum of two periodic signals/matrices.
- $A * B$ performs the appropriate *harmonic convolution*.
- `inv(A)` computes the phasors of $A(t)^{-1}$.
- `mreal(A)` gives the temporal real part $\Re(A(t))$, different from `real(`$A_{3Darray}$`)`.

All usual operations (addition, multiplication, inversion, Toeplitz conversion, conjugation) are thus redefined to respect the underlying harmonic structure.

Example: In just 5 lines, we perform an operation that would manually require about 30 lines of index manipulation, convolutions, and temporal reconstructions. For comparison, the underlying functions `plot`, `times`, and `random` represent 567, 112, and 120 lines of code respectively in the toolbox.

```
A = PhasorArray.eye(3,3,4); % Periodic identity
% I + 2 sum_{k=1}^4 cos(kwt)
B = PhasorArray.random(3,3,5); % Random periodic matrix
C = A * B; % Harmonic product
C.plot(); % Temporal display
C.value % Display raw 3D harmonic array
```

> **Note:** The command `C.value` allows access to the raw harmonic coefficients stored in 3D, with generally complex values. To obtain the temporal values of a matrix A realization over one period, also in 3D array form, use `evalTime` or `plot`.

2 Toolbox Structure

The project organization is structured as follows:

- **Core functions** `pArrayBasicOperations`: direct 3D array manipulation.
- **Main class** `@PhasorArray`: encapsulation, operator overloading, new methods.
- **Dynamic systems** `@PhasorSS`: periodic system modeling.
- **Sparse version** `@sparsePhasorArray`: sparse storage (experimental).
- **Examples**: demonstration scripts from simple to advanced usage.

3 Main Methods of the `PhasorArray` Class

The `PhasorArray` class provides a comprehensive set of methods for manipulating periodic objects in the harmonic domain. These methods are organized according to the following main categories:

3.1 Creation and Initialization

- Constructor: `PhasorArray`
- Standard generators: `eye`, `ones`, `zeros`, `empty`, `scalar`, `random`, `randomPhasorArrayWithPole`, `ndsdpvar`, `sym`

3.2 Analysis and Information Retrieval

- **Dimensions and size**: `size`, `dim`, `h`, `end`, `length`, `ndims`, `numel` (operation on array), `numelt` (operation on temporal matrix, only first two dimensions), `isempty`
- **Energy properties** (allows evaluation of phasor importance): `energy`, `realEnergy`, `imagEnergy`, `pageEnergy`, `tolReal`
- **Structural properties**: `isreal`, `iscomplex`, `ishermitian`, `issymmetric`, `issquare`, `isvector`, `isrow`, `iscolumn`, `ismatrix`, `isscalar`, `isnumeric`, `islogical`, `isimag`, `iszero`
- **General information**: `info`

3.3 Standard Mathematical Operations

- **Elementary algebra**: `plus`, `minus`, `times`, `rdivide`, `ldivide`, `power`, `mpower`, `mtimes`, `mldivide`, `mrdivide`
- **Special functions**: `det`, `expm`, `logm` (the m denotes that the method acts on the temporal version, we expect harmonics of $e^{A(t)}$ not $e^{A_{3Darray}}$), `trace`, `inv`, `kron`
- **Transpositions**: `transpose`, `ctranspose`, `mtranspose`, `mctranspose`
- **Specific temporal operations**: `antiD`, `retro`, `trretro`, `PhaseShift`, `pmax`

3.4 Specific Harmonic Manipulation

- **Harmonic management**: `expandBase`, `reduce`, `extract`, `pad`, `squishBase`, `trunc`, `neglect`
- **Operations on 3D array pages** (harmonics): `pageplus`, `pagetimes`, `pagepower`, `pagerdivide`, `pageldivide`, `pagectranspose`, `pagetranspose`, `pageimag`, `pagereal`, `pageabs`, `pageconj`

3.5 Conversions and Alternative Representations

- **General conversion**: `double`, `sdpval`, `value`, `Value`, `squeeval`
- **Alternative representation forms**: `AngleAmpForm`, `CosSinForm`, `ImagRealForm`, `RealImagForm`, `SinCosForm`
- **Conversion from other formats**: `fromTBMatrix`, `fromTFTB`, `funcToPhasorArray`, `time2Phasor`, `cqt2ScalarPhasor`

3.6 Temporal Domain Evaluation

- **Temporal generation**: `evalTime`, `evalp`, `initial`
- **Simulation**: `lsim`, `sim`

3.7 Advanced Harmonic Analysis

- **Spectral analysis**: `HmqEig` (spectrum of $[\mathcal{A}]_m$), `HmqNEig` (spectrum of $[\mathcal{A}-\mathcal{N}]_m$)
- **Pole placement**: `place`
- **Toeplitz form and associated operations**: `BT`, `BTHankel`, `TB`, `TBHankel`, `spBT`, `spTB`, `spBTHankel`, `spTBHankel`, `spTBmtimes`
- **Fourier vector construction**: `FvTB`, `TF_BT`, `TF_TB`

3.8 Standard Operator Overloading

- **Advanced indexing**: `parenAssign`, `parenReference`, `parenDelete`, `parenListLength`, `braceAssign`, `braceReference`, `braceListLength`
- **Note**: Parentheses allow access to the 3D array and manipulate it as if it were one, often requiring three-entry indexing `A(i,j,k)`
- Braces allow access to the phasor corresponding to $A_{i,j}$ via `A{i,j}`

3.9 Visualization

- **Graphics and display**: `plot` (temporal simulation), `plot3D` (same along two real and imaginary axes), `barsurf` (3D histogram visualization of A's harmonic intensity), `stem` (histogram visualization of A's harmonic intensity)

4 Simple Periodic System Simulation

In this section, we progressively present the use of basic functionalities of the `PhasorArray` class. The goal is to show how to create, manipulate, and easily simulate periodic matrices without manually handling underlying harmonic operations.

4.1 Creating and Displaying a Periodic Matrix

Creation from Fourier Coefficients The most direct way to create a `PhasorArray` object is to start from a 3D array whose third dimension encodes the harmonic coefficients:

```
A_0 = eye(2);
A_1 = [1 1i; -1i 2];
A_2 = [2i, exp(1i*1.89)/4; 0.5, -1];

% Array construction
A_ar = cat(3, conj(A_2), conj(A_1), A_0, A_1, A_2);

% PhasorArray construction
A = PhasorArray(A_ar);

% Visualization
figure;
plot(A,2,0:0.001:4,"title","A(t)");
figure;
stem(A,"display","real","scale","linear");
figure;
barsurf(A);
```

Three types of visualization are proposed here: *plot*, *stem*, and *barsurf*.

Creation from a Temporal Signal From uniform sampling of a periodic signal, the `time2Phasor` method automatically reconstructs the harmonic decomposition:

```
T = 2; % Period
sig = @(t) sawtooth(2*pi*t/T,0.5);
dt = 1/2^6;
t = (0:dt:(1-dt))*T;
y = sig(t);

Y = PhasorArray.time2Phasor(y, "timeDim", 2);

% Visualization
```

```
figure;
plot(Y,T,0:0.001:4,"title","Reconstruction of y(t)");
```

4.2 Elementary Manipulations

Addition, Product, and Temporal Operations Usual operations on periodic matrices become immediate:

```
T = 2
tt = 0:0.001:4;

% Random matrix with 5 harmonics (2 x 2 x 11)
B = PhasorArray.random(2,2,5);
C = A + B;     % Temporal addition
D = A * B;     % Temporal product (harmonic convolution)

dA = d(A, T); % Temporal derivative
Ap = dephase(A, pi/2); % Phase shift of +pi/2 rad

% Visualization
figure; plot(B,T,tt,"title","B(t)");
figure; plot(C,T,tt,"title","Sum A + B");
figure; plot(D,T,tt,"title","Product A*B");

figure; plot(dA,T,tt,"title","Derivative of A");
hold on; plot(A,T,tt);
legend('A derivative', 'A')

figure; plot(Ap,T,tt,"title","A(t+T/4)");
hold on; plot(A,T,tt);
legend('A phase shifted', 'A')
```

4.3 Harmonic Reduction

Explicit Truncation To limit the number of harmonics used:

```
A_trunc = A.trunc(1); % Keep only -1 to +1
figure;
stem(A,"marker",'*')
hold on
stem(A_trunc,"display","abs");
```

Automatic Reduction by Thresholding Intelligent reduction according to harmonic energy, truncates to order k such that all phasors of order $> k$ are smaller than *threshold*.

```
A_red = A.reduce('reduceMethod','absolute','reduceThreshold',1e-2);
figure; stem(A_red,"display","abs");
```

Zero Thresholding Reduction Intelligent reduction according to harmonic energy, sets to 0 any harmonics smaller than the threshold.

```
A_red = A.neglect(1,'reduceMethod','absolute');
figure;
stem(A,"marker",'*')
hold on
stem(A_red);
sgtitle('Threshold at 1')
```

4.4 Simple Periodic System Simulation

System Simulation The following code simulates the autonomous system $\dot{x} = S(t)x$:

```
S = PhasorArray.random(2,2,5);
x0 = rand(2,1); % Initial condition
T = 2; % Period
[y,t] = lsim(S, 10*T, x0, T, "plot", true);
figure;
plot(t,y)
title('Simulation')
```

Observations

- Temporal behavior is correctly reconstructed.
- The periodic nature of the system is entirely managed by the class.

5 Application Example: Harmonic Sylvester Equation Resolution and Harmonic Pole Placement

In this section, we illustrate the use of the `PhasorArray` toolbox to solve a **harmonic pole placement** problem on a periodic system.

5.1 Problem Statement

Consider the periodic dynamics:

$$\dot{x}(t) = A(t)x(t) + B(t)u(t) \tag{13}$$

where:

- $A(t)$ is a periodic matrix with period T.
- $B(t)$ is assumed constant here for simplification.
- $u(t)$ is the control to be designed.

We wish to determine a state feedback law:

$$u(t) = -K(t)x(t) \tag{14}$$

such that the closed-loop dynamics:

$$\dot{x}(t) = (A(t) - B(t)K(t))x(t) \tag{15}$$

possesses a targeted harmonic spectrum.

5.2 Strategy

The method relies on solving the periodic differential Sylvester equation:

$$\frac{d}{dt}P(t) - A(t)P(t) + P(t)\Lambda + B(t)G(t) = 0 \tag{16}$$

Equivalently, we solve the algebraic Sylvester equation in the harmonic domain:

$$(\mathcal{A} - \mathcal{N})\mathcal{P} - \mathcal{P}(\Lambda \otimes \mathcal{I} - \mathcal{N}) - \mathcal{B}\mathcal{G} = 0 \tag{17}$$

where:

- Λ is a diagonal matrix containing the targeted poles.
- $G(t)$ is an auxiliary gain matrix.

Once $P(t)$ is obtained, the state feedback is simply given by:

$$K(t) = G(t)P^{-1}(t) \tag{18}$$

5.3 Implementation with the `PhasorArray` Toolbox

The resolution of this problem is facilitated by using the `PhasorArray` class:

```
T = 1/50; % Fundamental period
nx = 3;   % State dimension

% Generation of a harmonic matrix A(t)
AA = PhasorArray.random(nx,nx,5);

% Definition of target eigenvalues
La = PhasorArray(diag([-50, -25, -125]));

% Generation of a positive definite G(t) matrix
G = PhasorArray.random(nx,nx,5,"time_structure","sdp");

% Resolution of harmonic Sylvester equation, truncated to order 200
P = PhasorArray(Sylv_harmonique(-AA, La, eye(nx)*G, 200, 2*pi/T));

% State feedback calculation
K = G / P;

% Closed-loop matrix construction
newA = AA - eye(nx)*K;

% Visualizations
figure; plot(P); sgtitle('Solution P(t)');
figure; plot(K); sgtitle('Harmonic gain K(t)');
figure; plot(newA); sgtitle('Closed matrix A(t) - BK(t)');
figure; plot(AA.HmqNEig(50,T),'*')
hold on; plot(newA.HmqNEig(50,T),'o'); hold on; grid on
sgtitle('Old and new eigenvalues')
```

Important Remarks:

- The function `Sylv_harmonique` numerically solves the Sylvester equation on a truncated basis of 200 harmonics, thus controlling the error.
- The specific structure of B is simply taken as identity for simplicity.
- The operator `/` in `PhasorArray` corresponds to an adapted temporal harmonic inversion.

5.4 Results Validation

Validation is performed in two steps:

- Verification of the Sylvester equation residual:

$$R(t) = \frac{d}{dt}P(t) - A(t)P(t) + P(t)\Lambda + B(t)G(t) \tag{19}$$

This residual must be negligible.

- Analysis of the closed-loop system's harmonic spectrum via the `HmqNEig` function, allowing visualization of dominant harmonics of the system $A(t) - B(t)K(t)$.

```
% Residual
res = P.d(T) + (-AA)*P + P*La + eye(nx)*G;
figure; stem(res); sgtitle('Sylvester equation residual');
```

6 Advanced Harmonic Analysis and Control Design

6.1 Lyapunov Equations

For a periodic system $\dot{x} = A(t)x$, asymptotic stability can be analyzed via the existence of a T-periodic, symmetric and positive definite solution to the periodic differential Lyapunov equation:

$$\dot{P}(t) + A(t)^T P(t) + P(t)A(t) + Q(t) = 0, \quad P(0) = P(T) \tag{20}$$

where $Q(t) = Q(t)^T > 0$ is T-periodic and belongs to L^∞.

In the harmonic domain, this is equivalent to solving the harmonic Lyapunov equation:

$$(\mathcal{A} - \mathcal{N})^* \mathcal{P} + \mathcal{P}(\mathcal{A} - \mathcal{N}) + \mathcal{Q} = 0 \tag{21}$$

The system is stable if and only if the solution $\mathcal{P}$ is Hermitian positive definite. A consistent numerical scheme is implemented in the toolbox:

```
% Create periodic system matrix
A = PhasorArray.funcToPhasorArray(@(t) [1+sawtooth(2*pi*t), cos(2*pi*t);
                                        sin(4*pi*t), -0.5+square(2*pi*t)/2], 1,
                                            6);
T = 1; % Period

% Define positive definite Q(t)
Q = PhasorArray.eye(2,2,0)*10; % Constant positive definite matrix

% Solve Lyapunov equation
P = lyap(A, Q, "T", T);

% Verify solution and analyze stability
figure; plot(P); sgtitle('Lyapunov solution P(t)');
figure; plot(eig(P.TB(20)), '*'); title('Eigenvalues of P (should be positive)')
    ;

% Check system stability via harmonic eigenvalues
lambda = A.HmqNEig(20, T, "fundamental");
fprintf('System eigenvalues: %.4f, %.4f\n', real(lambda(1)), real(lambda(2)));
if all(real(lambda) < 0)
    fprintf('System is asymptotically stable\n');
else
    fprintf('System is unstable\n');
end
```

6.2 Riccati Equations and LQR Control

For the classical LQR problem, the T-periodic feedback gain $K(t) = R(t)^{-1}B(t)^T S(t)$ is computed by solving the periodic differential Riccati equation:

$$\dot{S} + A^T S + SA - SBR^{-1}B^T S + Q = 0, \quad S(0) = S(T) \tag{22}$$

In the harmonic domain, this becomes an algebraic Riccati equation. The toolbox implements an iterative Kleinman-like algorithm that guarantees consistency:

```
% Define system matrices
A = PhasorArray.funcToPhasorArray(@(t) [1+sawtooth(2*pi*t), cos(2*pi*t);
                                        sin(4*pi*t), -0.5], 1, 6);
B = PhasorArray([1; 0]) + PhasorArray([0; 1]) * PhasorArray.sin;
T = 1;

% LQR weights
Q = PhasorArray.eye(2,2,0) * 10;
R = PhasorArray.eye(1,1,0);

% Initial stabilizing gain
K0 = PhasorArray([10, 10]);

% Solve Riccati equation
[K_final, S_final] = RicHarmonicKlein(A, B, Q, R, K0, T, ...
    "autoUpdateh", true, "max_iter", 50, "residualThreshold", 1e-6, ...
    "htrunc", 6, "hmax", 500);

% Verify closed-loop stability
A_cl = A - B * K_final;
lambda_cl = A_cl.HmqNEig(20, T, "fundamental");
fprintf('Closed-loop eigenvalues: %.4f, %.4f\n', real(lambda_cl(1)), real(
    lambda_cl(2)));

% Visualize results
figure; plot(K_final); sgtitle('Optimal feedback gain K(t)');
figure; plot(S_final); sgtitle('Riccati solution S(t)');
```

6.3 Harmonic Eigenvalue Analysis

The `HmqNEig` function computes the Floquet exponents (harmonic eigenvalues) of a periodic system. The spectrum of $\mathcal{A}-\mathcal{N}$ consists of the fundamental eigenvalues λ_i replicated at $\lambda_i + jk\omega$ for all $k \in \mathbb{Z}$, where $\omega = 2\pi/T$. For stability analysis, it suffices to examine only the fundamental eigenvalues:

```
% Stability analysis of periodic system
A = PhasorArray.random(3, 3, 5);
T = 1;
h = 15; % Truncation order

% Compute all harmonic eigenvalues (truncated to order h)
% These are: {lambda_i + jk*omega, i=1:n, k=-h:h}
lambda_all = A.HmqNEig(h, T);

% Compute only fundamental eigenvalues (the lambda_i)
% These determine stability: system stable iff Re(lambda_i) < 0 for all i
lambda_fund = A.HmqNEig(h, T, "fundamental");

% Visualize in complex plane
figure;
subplot(1,2,1);
plot(lambda_all, 'x', 'MarkerSize', 8);
title('All Harmonic Eigenvalues \lambda_i + jk\omega');
xlabel('Real part'); ylabel('Imaginary part');
grid on; axis equal;

subplot(1,2,2);
plot(lambda_fund, 'ro', 'MarkerSize', 10, 'LineWidth', 2);
title('Fundamental Eigenvalues \lambda_i');
xlabel('Real part'); ylabel('Imaginary part');
grid on;
```

```
27
28 % Stability check (only fundamental eigenvalues matter)
29 if all(real(lambda_fund) < 0)
30     fprintf('System is asymptotically stable (all fundamental eigenvalues have
       negative real parts)\n');
31 else
32     fprintf('System is unstable\n');
33 end
```

7 Linear Matrix Inequalities (LMI) Resolution

We illustrate in this section how to use the `PhasorArray` class to formulate and solve stability problems via periodic LMIs, exploiting YALMIP. The toolbox supports PhasorArrays whose 3D array consists of ndsdpvar, allowing construction of optimization problems.

7.1 Problem Statement

Consider a periodic system:

$$\dot{x}(t) = A(t)x(t) \tag{23}$$

where $A(t)$ is a periodic matrix with period T.
The objective is to find a function $P(t)$, periodic and positive definite, such that:

$$\dot{P}(t) + A(t)^T P(t) + P(t)A(t) \leq 0 \tag{24}$$

which certifies exponential stability of the origin for the system.

7.2 Method 1: Direct Formulation in Harmonic Domain

We construct the PhasorArray consisting of a well-chosen ndsdpvar to represent harmonics of a symmetric matrix:

```
1  hp = 10;
2  P = PhasorArray.ndsdpvar(nx,nx,hp);
3
4  ATP = (A.') * P;
5  PA = P * A;
6  ATPpPA = ATP + PA;
7
8  Q = PhasorArray.eye(nx,nx,0)*10
9
10 hlmi = 20;
11 F = [P.T_tb(hlmi) >= 0, ...
12 ATPpPA.T_tb(hlmi) + T_tb(P.d(T),hlmi) <= -Q.T_tb(hlmi)];
13 optimize(F,phas(P.trace,0));
14
15 Pf = sdpval(P);
16 figure; plot(Pf); sgtitle('P(t) solution')
17 figure; barsurf(Pf); title('P harmonics')
18 figure; plot(eig(Pf.T_tb(hlmi)),'*'); title('P eigenvalues')
```

Note the property for periodic matrices : $\mathcal{T}(\dot{P}) = -\mathcal{N}^*\mathcal{P} - \mathcal{P}\mathcal{N}$, hence the use of `T_tb(P.d(T),hlmi)` in the LMI formulation. `d` being the derivative operator in the toobox.

7.3 Method 2: Hankel Correction Approach

An alternative and numerically stable approach uses Hankel corrections to properly handle truncation effects. This method explicitly accounts for the coupling between positive and negative harmonics through Hankel matrices:

```
hp = 10;
P = PhasorArray.ndsdpvar(nx,nx,hp,"PhasorType",'symmetric',"real",true);

PT = P.T_tb(hp);
ATB = A.T_tb(hp);

% Extract Hankel correction matrices
[AHpJ,~,~,AHm] = A.TBHankel(hp);
[~,AtJHm,AtHp,~] = TBHankel(A.',hp);
[PHpJ,PJHm,PHp,PHm] = P.TBHankel(hp);

% LMI formulation with Hankel corrections
F = [PT >= 0;
     (ATB' + N_tb(A,hp))*PT + PT*(ATB - N_tb(A,hp)) + ...
     AtHp*PHm + PHp*AHm + ...
     AtJHm*PHpJ + PJHm*AHpJ <= -Q.T_tb(hlmi)];

optimize(F,phas(P.trace,0));

Pf = sdpval(P);
figure; plot(Pf); sgtitle('P(t) solution with Hankel correction')
figure; barsurf(Pf); title('P harmonics with Hankel correction')
```

The `TBHankel` method returns four matrices: `AHpJ`, `AJHm`, `AHp`, `AHm` representing different harmonic coupling terms that must be included in the LMI constraints for mathematical rigor.

8 Linear Time-Periodic Systems with PhasorSS

The `PhasorSS` class extends the PhasorArray framework to provide a complete representation of Linear Time-Periodic (LTP) systems. This class enables simulation and analysis using familiar MATLAB Control System Toolbox commands while maintaining the harmonic structure.

8.1 PhasorSS System Construction

A `PhasorSS` object represents the periodic state-space system:

$$\dot{x}(t) = A(t)x(t) + B(t)u(t) \tag{25}$$

$$y(t) = C(t)x(t) + D(t)u(t) \tag{26}$$

where all matrices are periodic with period T.

```
% Define periodic system matrices
T = 0.1; % Period in seconds
nx = 3; nu = 2; ny = 2;

% Create periodic state matrix with prescribed poles
A = PhasorArray.randomPhasorArrayWithPole(3, [-15, -10, -5], T, 'h', 10);

% Input and output matrices
B = PhasorArray.random(nx, nu, 5);
C = PhasorArray.random(ny, nx, 5);
D = PhasorArray.zeros(ny, nu); % Feedthrough (often zero)

% Create LTP system
sys = PhasorSS(A, B, C, D, T, "InputName", {'U1', 'U2'}, ...
               'OutputName', {'Y1', 'Y2'}, 'StateName', {'x1', 'x2', 'x3'});

% Display system information
figure;
sys.plot(); % Plot all system matrices
```

8.2 System Analysis and Simulation

The `PhasorSS` class supports standard MATLAB simulation commands:

```
% Analyze system stability
poles = sys.A.HmqNEig(15, T, "fundamental");
fprintf('System poles: '); disp(poles');

% Initial condition response
x0 = ones(3,1);
t_sim = 0:T/100:20*T;

figure;
initial(sys, x0, t_sim);
title('Initial Condition Response');

% Step response (constant input applied to each channel)
figure;
step(sys, t_sim);
title('Step Response');

% Impulse response
figure;
impulse(sys, t_sim);
title('Impulse Response');
```

8.3 Periodic Input Simulation

For periodic inputs, the toolbox provides specialized simulation functions:

```
% Periodic step input (constant in harmonic domain)
UW = PhasorArray.random(2, 1, 5); % Periodic input signal
figure;
plot(UW, T, 0:T/50:3*T);
title('Periodic Input Signal');

% Simulate with periodic input
figure;
stepu(sys, t_sim, UW);
title('Response to Periodic Input');

% General time-varying input using lsim
t = 0:T/100:20*T-T/100;
uw = [t*25-70; atan((t-4*T)/0.3)*30]; % Time-varying input
figure;
plot(t, uw);
title('Time-Varying Input');

figure;
lsim(sys, t, uw, ones(3,1));
title('System Response to Time-Varying Input');
```

Note:

- `step` applies a constant step to each input
- `stepu` takes a PhasorArray U as input and applies a periodic input (constant in the harmonic domain)
- `lsim` handles general time-varying inputs
- `initial` simulates free response from initial conditions
- `impulse` computes impulse response

9 Correspondences Between Temporal and Harmonic Domains

To conclude this documentation, we summarize the fundamental properties and correspondences between the temporal domain and its harmonic equivalent.

Table 1: Correspondences between temporal and harmonic domains.

Temporal Domain	**Harmonic Domain**	***Notes***
Scalar or vector signal $x \in L^2_{\text{loc}}(\mathbb{R}, \mathbb{C}^n)$	Phasor sequence $X \in \mathcal{H}_T \subset C^a(\mathbb{R}, \ell^2(\mathbb{C}^n))$	*Correspondence is guaranteed only if X belongs to space $\mathcal{H}_T$.*
Multiplication by a function $a(t)x(t)$	Product by a Toeplitz operator $\mathcal{F}(ax) = \mathcal{A}X = \mathcal{T}(a)\mathcal{F}(x)$ $\mathcal{T}(ax) = \mathcal{T}(a)\mathcal{T}(x)$	*This is the fundamental property that linearizes products. $\mathcal{A}$ is a block Toeplitz operator.*
Matrix inversion $a^{-1}(t)$	Toeplitz inversion $\mathcal{T}(a^{-1}) = \mathcal{T}(a)^{-1}$	*This is the product property applied to $\mathcal{I} = \mathcal{T}(a^{-1}a)$*
Temporal derivation $\dot{x}(t)$	Algebraic operation $\dot{X} = \mathcal{F}(\dot{x}) - \omega\mathcal{N}X$ $\dot{\mathcal{X}} = \mathcal{T}(\dot{x}) + \omega\mathcal{N}\mathcal{X} - \mathcal{X}\omega\mathcal{N}$	*The term $\mathcal{N}$ is the diagonal operator* $\text{diag}(jk)$*, $X = \mathcal{F}(x)$ and $\mathcal{X} = \mathcal{T}(X)$*
T-periodic signal $x(t) = x(t+T)$	Constant harmonic signal $\dot{X}(t) = 0$	*Phasors of a periodic signal are constant.*
Real signal $x(t) \in \mathbb{R}^n$	Phasor conjugation $X_{-k}(t) = \overline{X_k}(t)$	*This property allows reducing the number of independent variables by half.*
Bounded function $a(t) \in L^\infty([0,T])$	Bounded operator on ℓ^2 $\|\mathcal{A}\|_{\ell^2} =$ $\sup_{\|X\|_{\ell^2}=1} \|\mathcal{A}X\|_{\ell^2} =$ $\|a(t)\|_{L^\infty} < \infty$	*Crucial condition for realizability of gains obtained in synthesis and stability.*
Positive definite symmetric periodic function $A(t) = A(t)^T \succ 0$ a.e.	Positive definite Hermitian operator $\mathcal{A} = \mathcal{A}^* \succ 0$	*Essential for Lyapunov functions and LMIs.*

10 Conclusion

This documentation has presented the foundations and capabilities of the PhasorArray Toolbox for harmonic domain modeling and control of periodic systems. The toolbox provides:

- A comprehensive abstraction layer for harmonic signal manipulation with mathematically rigorous foundations
- Intuitive operator overloading that respects harmonic structure and handles truncation challenges automatically
- Advanced analysis and synthesis capabilities including Lyapunov, Riccati, and Sylvester equation solvers

- Integration with optimization toolboxes like YALMIP for LMI formulations with both direct and Hankel correction methods
- Extensive visualization and simulation capabilities through the PhasorSS class
- Complete workflow support from system modeling through controller design to performance validation

The toolbox addresses the key challenge in harmonic methods: the conversion from infinite-dimensional theoretical results to consistent finite-dimensional numerical implementations. By encapsulating the required mathematical expertise within an object-oriented framework, it makes advanced harmonic control methods accessible to the broader engineering community.

10.1 Key Contributions

Mathematical Rigor: The toolbox implements consistent truncation schemes that ensure convergence to the infinite-dimensional solution, addressing the fundamental challenge that $\mathcal{T}(AB)_h \neq \mathcal{T}(A)_h\mathcal{T}(B)_h$.

Practical Implementation: Complex harmonic operations that would typically require 20-40 lines of manual coding are reduced to simple, intuitive MATLAB commands through operator overloading.

Advanced Control Capabilities: The toolbox provides state-of-the-art solvers for harmonic Lyapunov, Riccati, and LMI problems, enabling sophisticated control design for periodic systems.

System Integration: The PhasorSS class bridges the gap between harmonic theory and practical system simulation, supporting all standard MATLAB Control System Toolbox functions.

10.2 Applications and Impact

The examples presented demonstrate the toolbox's effectiveness across various application domains:

- **Power Electronics:** AC/DC converter control, harmonic mitigation in power systems
- **Vibration Control:** Analysis and control of systems with periodic coefficients
- **Aerospace:** Helicopter rotor dynamics, satellite attitude control with periodic disturbances
- **Process Control:** Periodic batch processes, cyclic manufacturing systems
- **Renewable Energy:** Wind turbine control, solar tracking systems

The significant reduction in code complexity and development time (observed factors of 3-5 in real projects) makes this toolbox an essential tool for researchers and engineers working with periodic systems.

10.3 Future Developments

The toolbox continues to evolve with ongoing research in harmonic control theory. Future enhancements may include:

- Sparse harmonic representations for very high-order systems
- Distributed and parallel algorithms for large-scale problems
- Integration with model predictive control frameworks
- Real-time implementation tools for embedded systems
- Extended support for time-varying periodic systems

For additional examples, advanced usage patterns, and the latest developments, refer to the example scripts and documentation provided with the toolbox distribution at `github.com/mxmGrss/phasorArra`