

プラントモデル検証手法

公開 2024 年

作成 JMAAB プラントモデルワーキンググループ

改訂履歴

Revision	日付	内容	作成者
1.0	24/11/29	初版	プラントモデルワーキンググループ

■ 著作権について

- 本ドキュメントの著作権は、JMAAB に帰属します。
- JMAAB は、本文書の内容に関し、いかなる保証もするものではありません。万一本文書を利用して不具合等があった場合でも、JMAAB は一切責任を負いかねます。また、本文書に記載されている事項は予告なしに変更または廃止されることがありますので、あらかじめご了承ください。

■ 本ドキュメントの取扱いについて

- 本文書は、非営利目的、または利用者内部で使用する場合に限り、複製が可能です。また、本文書を引用する場合は、本文書からの引用であることを明示し、引用された著作物の題号や著作者名を明示する等の引用の要件を満たす必要があります。
- 本成果物については、JMAAB ホームページ（<https://jmaab.jp/>）を参照下さい。
- その他のお問い合わせは、JMAAB 事務局（jmaab-office@mathworks.co.jp）へご連絡下さい。

目次

1. 背景	4
2. 活動のフォーカスと進め方	5
3. ユースケースの定義	6
4. プロセスとツールチェーンの仮説	9
5. 仮説の有効性検証	11
5.1 題材としたモデルについて	11
5.2 トライアル結果	12
6. まとめ	29
7. 理想のツールチェーン実現のための MATLAB®への要望	30
8. 最後に	33

1. 背景

昨今、CASE に代表される社会的要求の変化に伴い、モビリティに関わるシステムが大規模複雑化しており、個社や単一組織での開発が難しくなっている。要求変化に対する迅速な対応に向け、大規模システムの効率的な開発には組織や会社を跨いだ制御モデル/プラントモデルの流通が求められている。

このような状況下で、モデル流通促進に向けた活動は様々な団体で進められており、JMAAB プラントモデル WG でも関心を持つ企業が多い。2019 年度には、プラントモデル流通時に必要な外部仕様書・内部仕様書フォーマットを検討し、JMAAB HP 上に公開した。

一方、プラントモデルの検証に関する記載事項に関しては詳細な検討ができておらず、課題として残っていた。2022 年度以降は、「プラントモデル開発における検証プロセスやツールチェーン」と「流通を想定したうえで検証結果として示すべき事項」について議論してきた。

本ドキュメントには、議論の結果から得られたプラントモデル検証のベストプラクティスを記している。

2. 活動のフォーカスと進め方

プラントモデル検証は、本 WG で定義したモデル流通プロセスの「依頼先がモデル開発に着手」の一部に該当する。

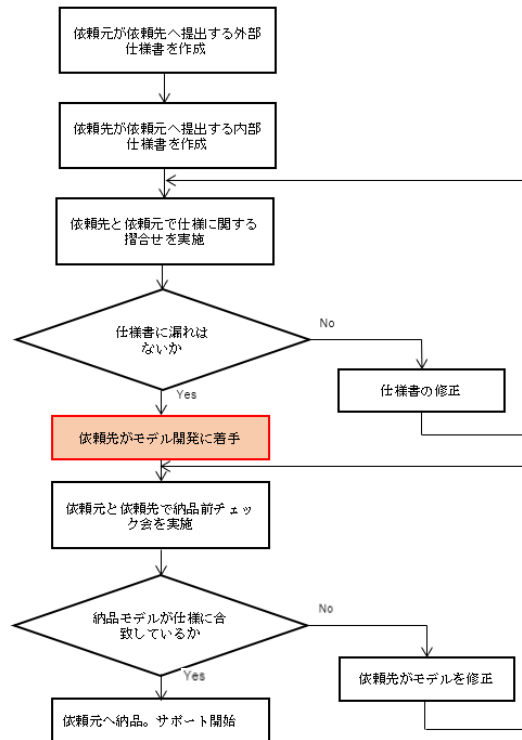


図 1 モデル流通プロセス

プラントモデル開発における検証プロセスとツールチェーンを定義するため、本活動を以下の Step で進めた。

- Step1：プラントモデル検証のユースケースの定義
- Step2：ユースケースに沿ったプロセスとツールチェーンの仮説
- Step3：仮説の有効性検証

初めに、開発時の環境制約を考慮した現実的なユースケースを定義した。次に、ユースケースを実現するためのプロセスと、MATLAB および Simulink®を中心としたツールチェーンの仮説を立てた。最後に、仮説を立てたツールチェーンで検証プロセスを回してみ、その有効性を検証した。修正が必要な部分は修正し、残課題を整理した。また、明確になったツールチェーンの課題については、改善要望として MathWorks 社へフィードバックを行った。

3. ユースケースの定義

モデル開発依頼元が抱える制約を考慮したユースケースを定義する。ユースケースの観点としては、車種が特定されているか、テストパターンが決まっているか、リファレンスデータが存在するか、合否基準が決まっているか、モデルの秘匿化は必要か、燃費届け出などで特別な制約が必要か、などが考えられる。

今回はよくある以下のユースケースを定義した。

- 車種：特定車種が対象
- テストパターン：特定モード走行が対象
- リファレンスデータ：実機で計測したデータあり
- 合否基準：6つの合否基準あり（後述）
- モデル秘匿化：不要
- 特別な制約：なし

尚、WG メンバの経験から、プラントモデルに求める合否基準として、以下の6つを設定した。

- ① 最大瞬間誤差
- ② RMSE（二乗平均平方根誤差）
- ③ 相関係数
- ④ 計算時間
- ⑤ パワースペクトル密度
- ⑥ 周波数特性

以下に、各合否基準の数値とイメージを記載する（合否基準の妥当性を議論する活動ではないため、数値は仮設定である）。

- ・ 最大瞬間誤差： ± 20 以内（単位は比較データの物理量に依存）

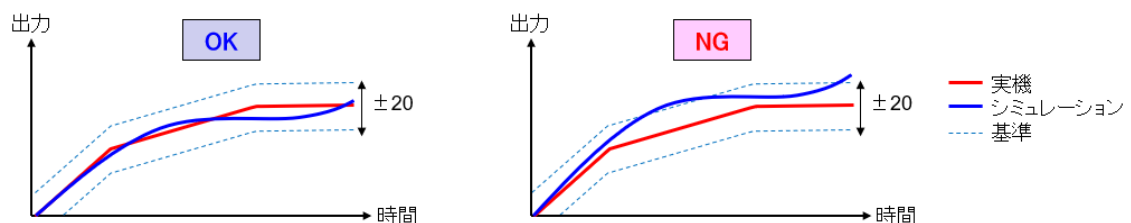


図 2 最大瞬間誤差

- ・ RMSE（二乗平均平方根誤差）：5 以内（単位は比較データの物理量に依存）

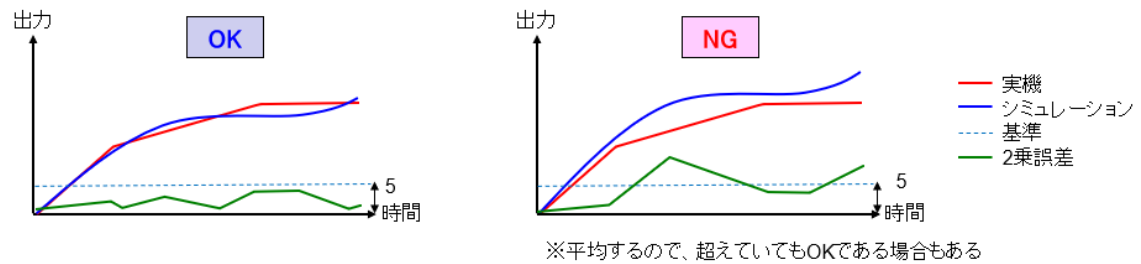


図 3 RMSE（二乗平均平方根誤差）

- ・ 相関係数：0.7 以上

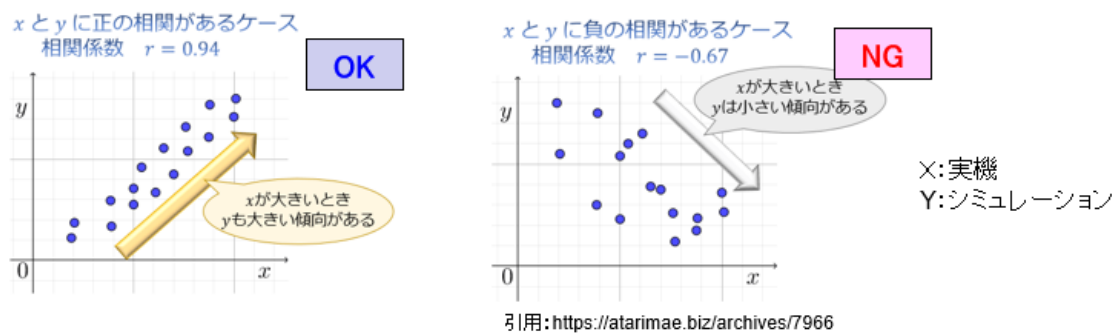


図 4 相関係数

- ・ 計算時間：実時間の 5 倍速以上（例：10sec 間のテストを 2sec 以内に完了）

- ・ パワースペクトル密度：250Hz 以上の領域で、最大パワーの 10%以下

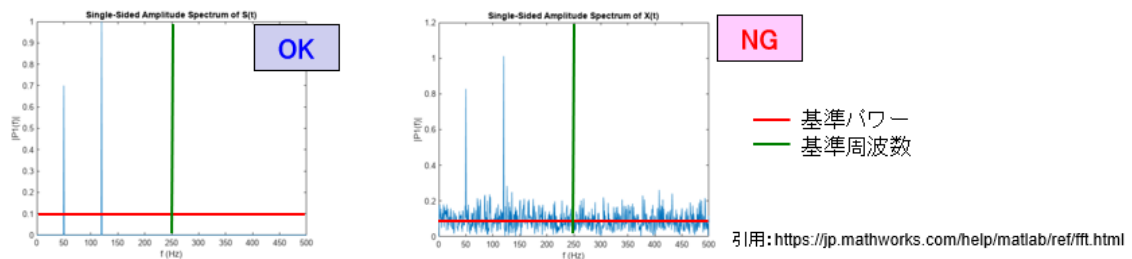


図 5 パワースペクトル密度

- ・ 周波数特性：ゲイン線図の共振点の数が等しい

ゲイン線図の共振点の周波数ズレが $\pm 10\%$ 以内

位相線図の $\pm 180^\circ$ の周波数ズレが 10% 以内

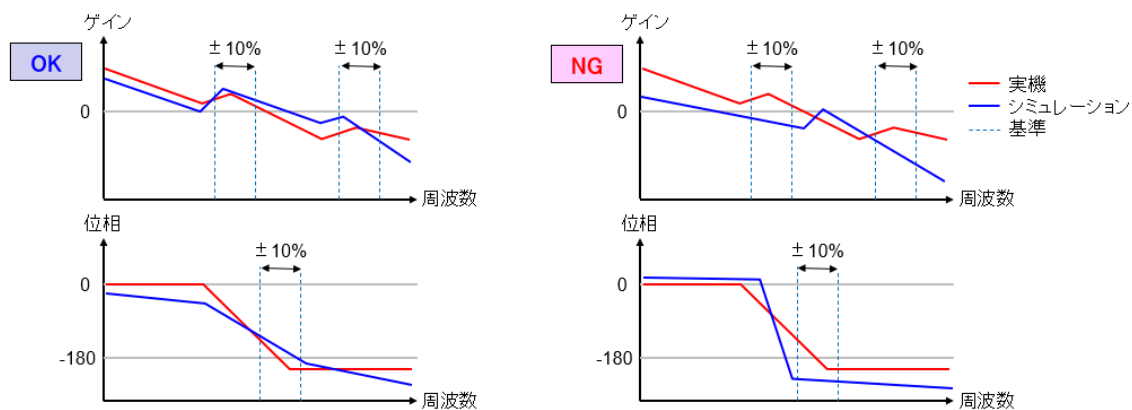


図 6 周波数特性

4. プロセスとツールチェーンの仮説

3章で定義したユースケースに対して、モデル検証のプロセスとツールチェーンの仮説を立てる。

図 7 に示すように、モデル検証の大まかなプロセスを「データ準備」⇒「実行管理」⇒「合否判定」と定義した（詳細なプロセスは、5章に記載する）。「データ準備」では、内部仕様書に記載された検証シナリオに必要なテストデータを準備する。「実行管理」では、複数のテストシナリオを管理しながらテスト実行を行う。そして、「合否判定」では、内部仕様書の判断基準に従ってモデルが意図した動作をしているか判定を行う。

この検証プロセスに Simulink Test™ を適用することで、内部仕様書やテストシナリオとのトレースを容易にする Requirements Toolbox™ と、モデル本体である Simulink や Simscape™ とトレースをとりながら開発を進めることができると考える。

テスト結果は、Simulink Test のレポート機能で出力することで、内部仕様書の付属ドキュメントとすることができる。

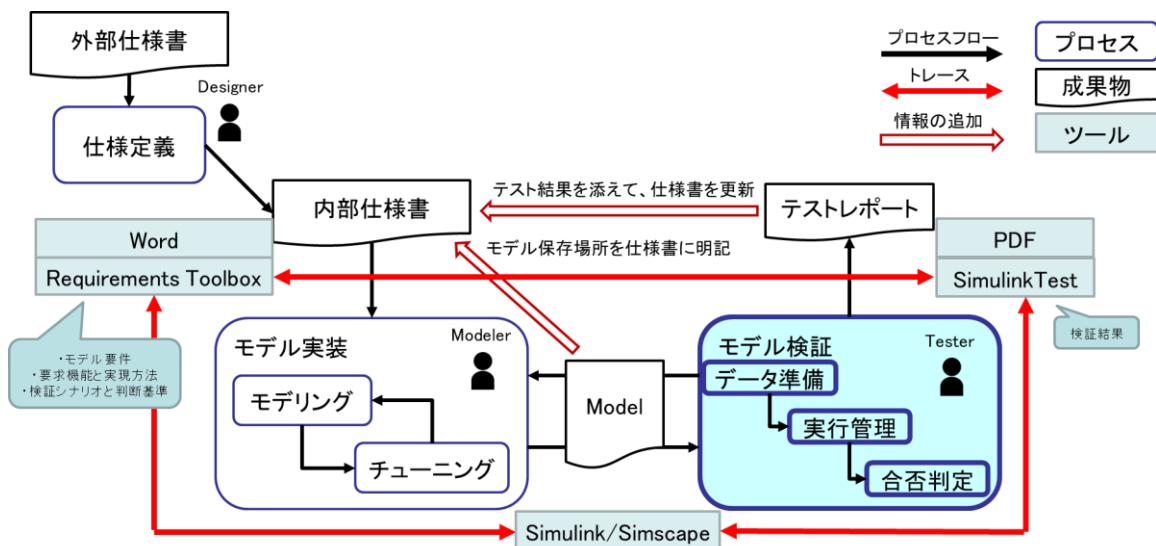


図 7 プロセスとツールチェーンの仮説

しかしながら、後述する「仮説の有効性検証」を行う中で、Simulink Test だけでは検証プロセスに対応できないことが分かった。そこで、Simulink Test の機能不足をライブスクリプトで補完することで、検証プロセスに対応できるツールチェーン（図 8）を構築した。

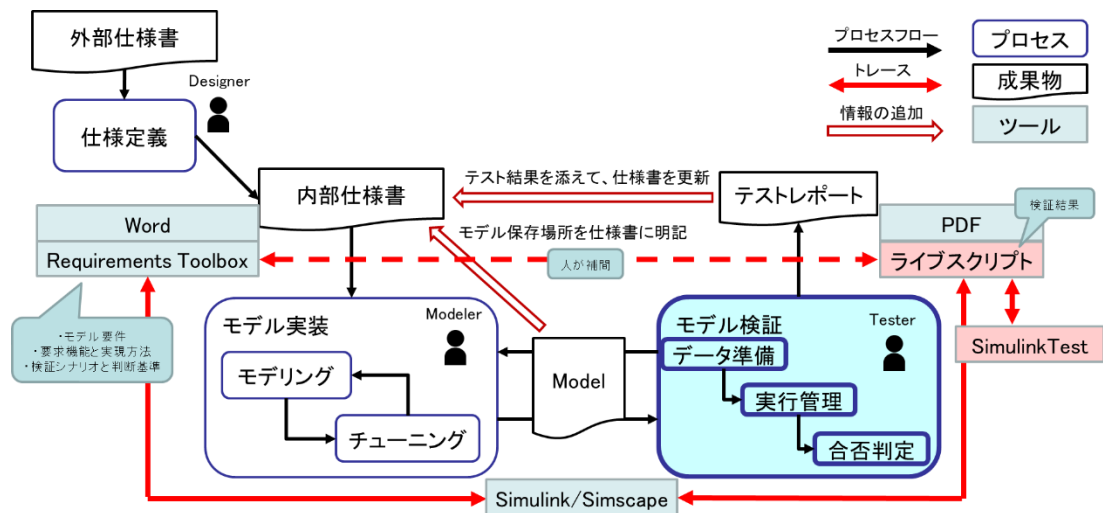


図 8 現実的なプロセスとツールチェーン

Simulink Test では、「データ準備として検証データを加工する」、「合否判定として多様な合否基準で判定する」、「テストレポート生成のためのテスト情報を成形する」（これらの詳細プロセスは、5 章に記載する）ができない。そのため、Simulink Test はテスト実行、テスト結果データ収集、①最大瞬間誤差に関する合否判定をするスレイブ機能として動かす。一方マスターとしてライブスクリプトを活用することで、テストレポート生成を前提としたドキュメント構成を作成しながら、テストデータ加工、合否判定ロジックの構築、テスト実行のトリガ生成、①以外の合否判定、テスト結果プロット、レポート生成の役割を担うようにした。

ただし、ライブスクリプトと Requirements Toolbox のトレースが取れなくなるため、人が補間をしながらテスト設計をする必要がある。

尚、明らかになった Simulink Test の課題は、「7 章 理想のツールチェーン実現のための MATLAB への要望」にまとめている。

5. 仮説の有効性検証

4 章で仮説を立てた、検証プロセスとツールチェーンの有効性検証を行った。検証の結果得られたベストプラクティスを「5.2 章 トライアル結果」にまとめて記載している。

尚、図 8 で示した検証プロセスをさらに詳細に定義して、以下のプロセスでトライアルを実施している。

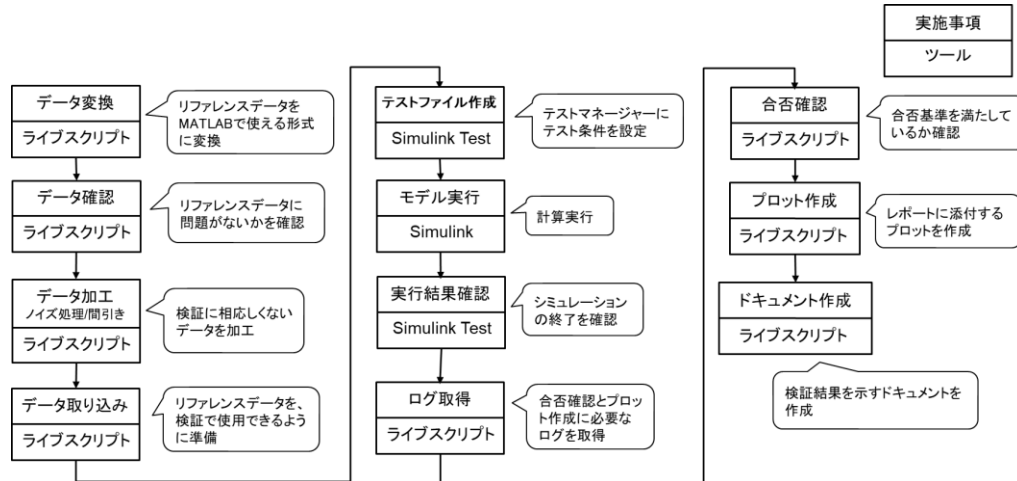


図 9 モデル検証の詳細プロセス

5.1 題材としたモデルについて

トライアルでは、2017 年に JMAAB で公開した HEV (Hybrid Electric Vehicle) モデルの中から、バッテリー、モータ、トルクコンバータのサブシステムを題材として検討を進めた。「5.2 章 トライアル結果」では、代表してモータについて記載している。

尚、モデルに設定している車両諸元（パラメータ）は、設計工程で問題ないことを確認している前提とする。

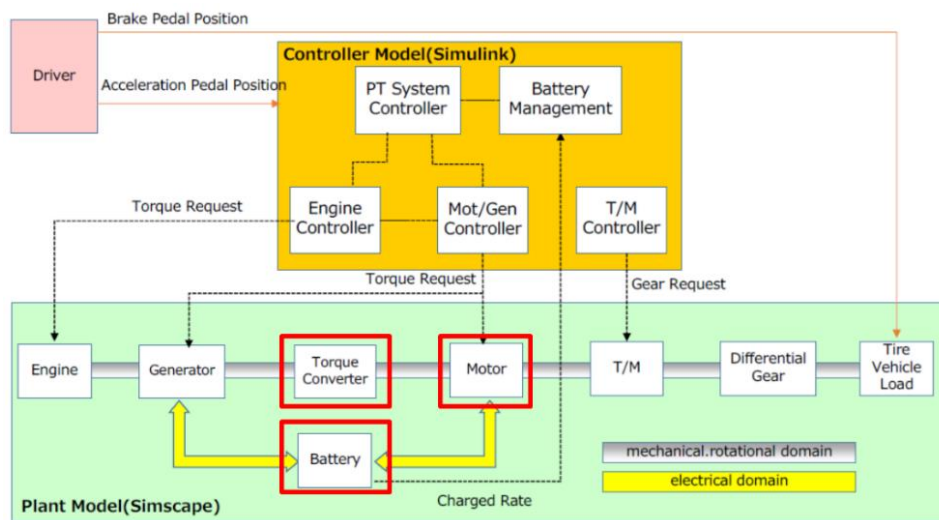


図 10 トライアルの題材モデル

5.2 トライアル結果

<ライブスクリプトの新規作成>

MATLAB を起動して、新規のライブスクリプトを作成する。
尚、本トライアルでは MATLAB R2023b を使用している。

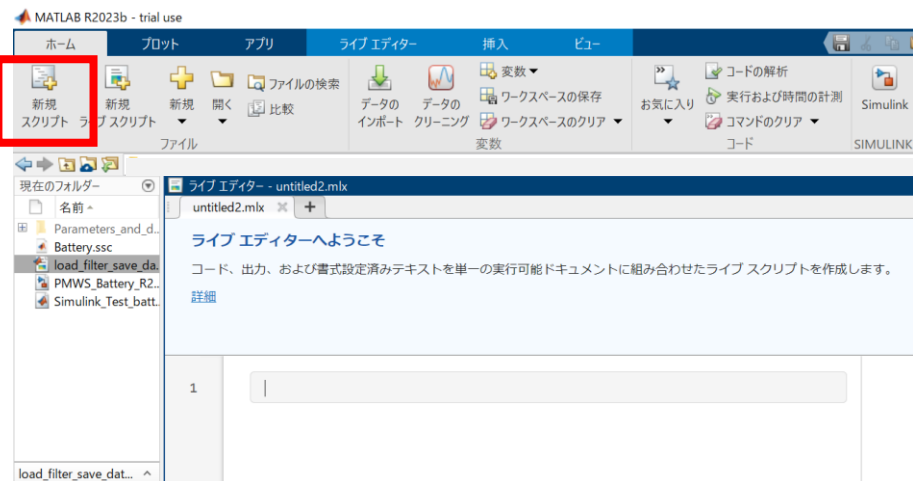


図 11 ライブスクリプト

<モデル実行準備>

モータモデルを実行するための準備をする。今回の場合、関連ファイルを保存したフォルダのパス設定と、Simscape ライブラリのビルドを、以下のスクリプトで実施する。（図 12 にスクリプトに対応するフォルダ構成例を示す）

※スクリプト例

```
% 関連ファイルのパス設定
addpath('PMWS_Simscape_Library')
addpath(genpath('Parameters_and_data'))

% Simscape ライブラリのビルド
cd('PMWS_Simscape_Library')
ssc_build PMWS_Simscape_Library;
cd('../')
```

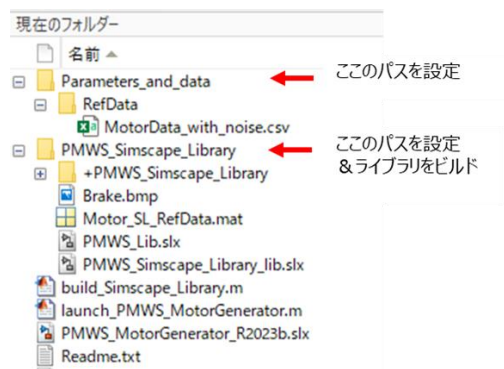


図 12 フォルダ構成例

<データ変換>

まずは、検証に使用する実機データをシミュレーションで使える形式に変換する。今回は、CSV 形式の実機データファイルがあると仮定して、以下のスクリプトで MAT ファイルに変換する。

尚、ファンクション“convert_datafile_to_matfile”は、別途定義して呼び出している。

※スクリプト例

```
% ファイル名と区切り文字の指定して、MAT ファイル形式のデータを作成
datafilename = 'MotorData_with_noise.csv';
Delimiter = ',';
MATfilename = 'MotorRefData.mat';

convert_datafile_to_matfile(datafilename, Delimiter, MATfilename)

load(MATfilename);
```

※スクリプト例 (convert_datafile_to_matfile)

```
function convert_datafile_to_matfile(loadfilename, Delimiter, MATfilename)
% MAT-ファイル変換

% ファイルからのデータの取り込み
MotorExpData = readmatrix(loadfilename, 'NumHeaderLines', 1, 'Delimiter',
Delimiter); % 区切りが異なる場合は, Delimiter の指定を変更

% MAT-ファイルに保存するデータを作成
MotorData_time = MotorExpData(:,1);
MotorData_TrqDemmand = MotorExpData(:,2);
MotorData_Current = MotorExpData(:,3);
MotorData_Voltage = MotorExpData(:,4);
MotorData_Trq = MotorExpData(:,5);
MotorData_AngVel = MotorExpData(:,6);
MotorData_InertiaTrq = MotorExpData(:,7);
MotorData_InertiaAngVel = MotorExpData(:,8);
MotorData_TrqOut = MotorExpData(:,9);

% 加工前データの保存
save(['./Parameters_and_data/', MATfilename],
'MotorData_time', 'MotorData_TrqDemmand', 'MotorData_Current', 'MotorData_Voltage', 'MotorData_Trq', 'MotorData_AngVel', 'MotorData_TrqOut')

end
```

<データ確認>

続いて、使用するデータに問題が無いかを確認する。以下のスクリプトで無効値（NaN）有無の確認や、プロットしてノイズ等の有無の確認を行う。

※無効値検出スクリプト例

```
% 各データの無効値の有無を検出
TF1 = anynan(MotorData_time);
TF2 = anynan(MotorData_TrqDemmand);
TF3 = anynan(MotorData_Current);
TF4 = anynan(MotorData_Voltage);
TF5 = anynan(MotorData_TrqOut);
TF6 = anynan(MotorData_AngVel);
```

※データプロットスクリプト例

```
% リファレンスデータをまとめてプロット
figure
subplot(5,1,1);
plot(MotorData_time, MotorData_TrqDemmand) ;
title('要求トルク')
subplot(5,1,2);
plot(MotorData_time, MotorData_Current) ;
title('電流')
subplot(5,1,3);
plot(MotorData_time, MotorData_Voltage) ;
title('電圧')
subplot(5,1,4);
plot(MotorData_time, MotorData_TrqOut) ;
title('出力トルク')
subplot(5,1,5);
plot(MotorData_time, MotorData_AngVel) ;
title('角速度')
```

<データ加工>

データ確認で問題データが存在した場合、フィルター処理や特定区間の切り出しを行い、シミュレーションに使えるデータに加工する。今回は、データにノイズが乗っていたと想定して、以下のスクリプトで除去する。FFT 解析によって周波数特性を把握した上で、通過周波数“fpass”を変更してフィルタリングをするスクリプトになっている。

尚、ファンクション“DataFiltering”、“fft_analysis_display”は、別途定義して呼び出している。

※スクリプト例

```
%% FFT 解析 / フィルター適応
MATfilename = 'MotorRefData.mat';
[MotorTrqDemmand_filt, MotorCurrent_filt, MotorVoltage_filt, MotorTrq_filt,
MotorAngVel_filt] = DataFiltering(MATfilename);
```

※スクリプト例 (DataFiltering)

```
function [MotorTrqDemmand_lpf, MotorCurrent_lpf, MotorVoltage_lpf, MotorTrq_lpf,
MotorAngVel_lpf] = DataFiltering(MATfilename)

% 解析用データの読み込み
eval(['load ', MATfilename])
L = length(MotorData_time); % Length of signal
fs = 1/(MotorData_time(2) - MotorData_time(1)); % Sampling frequency [Hz]

% シミュレーションデータの FFT 解析
fft_analysis_display(MotorData_Voltage, fs, L)

%% ローパスフィルター適用
fpass = 50; % 通過周波数 [Hz]
[MotorTrqDemmand_lpf, dtd_lpf] = lowpass(MotorData_TrqDemmand, fpass, fs);
[MotorCurrent_lpf, dc_lpf] = lowpass(MotorData_Current, fpass, fs);
[MotorVoltage_lpf, dv_lpf] = lowpass(MotorData_Voltage, fpass, fs);
[MotorTrq_lpf, dt_lpf] = lowpass(MotorData_Trq, fpass, fs);
[MotorAngVel_lpf, dav_lpf] = lowpass(MotorData_AngVel, fpass, fs);

%% 位相遅れなし(No Phase Delay)フィルター
MotorData_Voltage_npd = firlfilt(dv_lpf, MotorData_Voltage);

%% 移動平均
k = 5; % 平均するタップ数を指定
MotorData_Voltage_mean = movmean(MotorData_Voltage, [k 0], 'Endpoints', 'fill');

% サンプルとして電圧データで各フィルターの性能を比較
figure
plot([repmat(MotorData_time, 1, 4)], [MotorData_Voltage, MotorVoltage_lpf,
MotorData_Voltage_npd, MotorData_Voltage_mean])
legend('Signal with noise', 'LowPassFilter', 'Zero-Phase-Filter', 'move mean')
xlabel('Time [sec]')
ylabel('Motor Voltage [V]')
ylim([198 208])

end
```


※スクリプト例 (fft_analysis_display)

```
function fft_analysis_display(data, fs, L)
% 周波数応答確認

%% Plot Power Spectrum Density
figure
[pxx,f] = pwelch(data,[],[],[],fs);
plot(f,pow2db(pxx))
xlabel('Frequency (Hz)')
ylabel('PSD (dB/Hz)')

%% Plot Bode diagram
% data_FFT = fft(data);
% f = fs*(0:floor(L/2))/L;
% sys = frd(data(1:floor(L/2)+1),f,1/fs);
% bode(sys)

end
```

<データ取り込み>

続いて、モデルにインプットする構造にデータを書き換える。また、モデルに入力できるようにコンフィギュレーションパラメータの設定を行う。今回は以下のスクリプトで、タイムシリーズ形式のデータをデータセット形式に変換し、あわせてモデルのコンフィギュレーションパラメータにデータセット形式のデータを設定する。

尚、ファンクション”data_setting_to_simulate”は、別途定義して呼び出している。

※スクリプト例

```
% Simulink Test で活用可能なシミュレーション用 MAT-ファイル(Motor_SL_RefData.mat)と、Simulink モデルにマッピングするための Dataset オブジェクト作成
data_setting_to_simulate(MotorData_time, MotorTrqDemmand_filt, MotorCurrent_filt,
MotorVoltage_filt, MotorTrq_filt, MotorAngVel_filt, MotorData_TrqOut)

% 検証対象モデルのコンフィギュレーションパラメータを設定
model = 'PMWS_MotorGenerator_R2023b';
open_system(model)
activeConfigObj = getActiveConfigSet(model);
set_param(activeConfigObj, 'ZeroCrossControl', 'EnableAll');
set_param(activeConfigObj, 'LoadExternalInput', 'on', 'ExternalInput', 'ds');
set_param(activeConfigObj, 'SaveTime', 'on', 'TimeSaveName', 'tout');
set_param(activeConfigObj, 'SaveOutput', 'on', 'DSMLLogging', 'off', 'ReturnWorkspaceOutputs', 'off');
set_param(activeConfigObj, 'SaveFormat', 'Dataset');
```

※スクリプト例 (data_setting_to_simulate)

```
function data_setting_to_simulate(MotorData_time,
MotorData_TrqDemmand,MotorData_Current, MotorData_Voltage, MotorData_Trq,
MotorData_AngVel, MotorData_TrqOut)
% データ設定

% Simulink に取り込むために入力ファレンスデータを Timeseries オブジェクトで保存
Motor_TrqDemmand_RefData = timeseries(MotorData_TrqDemmand,
MotorData_time,'Name','TrqDemmand');
Motor_Voltage_RefData =
timeseries(MotorData_Voltage,MotorData_time,'Name','MotorVoltage');
Motor_AngVel_RefData = timeseries(MotorData_AngVel,
MotorData_time,'Name','MotorAngVel');

% SimulationData オブジェクトを作成し、データとポート番号を設定
% (PortIndex は検証モデルの Inport ブロックの端子番号に該当するように設定)
Motor_TrqDemmand_RefData_Input = Simulink.SimulationData.Signal;
Motor_TrqDemmand_RefData_Input.Values = Motor_TrqDemmand_RefData; % 要求トルクのリ
ファレンスデータを設定
Motor_TrqDemmand_RefData_Input.PortIndex = 1; % 端子番号 1 の Inport ブロックに設定
Motor_Voltage_RefData_Input = Simulink.SimulationData.Signal;
Motor_Voltage_RefData_Input.Values = Motor_Voltage_RefData;
Motor_Voltage_RefData_Input.PortIndex = 2;
Motor_AngVel_RefData_Input = Simulink.SimulationData.Signal;
Motor_AngVel_RefData_Input.Values = Motor_AngVel_RefData;
Motor_AngVel_RefData_Input.PortIndex = 3;

% Simulink モデルにマッピングするための Dataset オブジェクト作成
ds = Simulink.SimulationData.Dataset;
ds =
ds.addElement(Motor_TrqDemmand_RefData_Input,'Motor_TrqDemmand_RefData_Input');
ds = ds.addElement(Motor_Voltage_RefData_Input,'Motor_Voltage_RefData_Input');
ds = ds.addElement(Motor_AngVel_RefData_Input,'Motor_AngVel_RefData_Input');

% 出力ファレンスデータを timeseries オブジェクトで作成
% (ベースラインテストで信号を比較するため, Simulink モデルの信号名と timeseries の信号名を一致
させる。ここでは「MoterTrq」としている。)
Motor_Current_RefData =
timeseries(MotorData_Current,MotorData_time,'Name','MotorCurrent');
Motor_Trq_RefData =
timeseries(MotorData_Trq,MotorData_time,'Name','MotorTrq');
Motor_TrqOut_RefData =
timeseries(MotorData_TrqOut,MotorData_time,'Name','MotorTrqOut');

% シミュレーション用 MAT-ファイルにデータを保存
save './Parameters_and_data/Motor_SL_RefData.mat' ds Motor_TrqDemmand_RefData
Motor_Current_RefData Motor_Voltage_RefData Motor_Trq_RefData Motor_AngVel_RefData
Motor_TrqOut_RefData
end
```

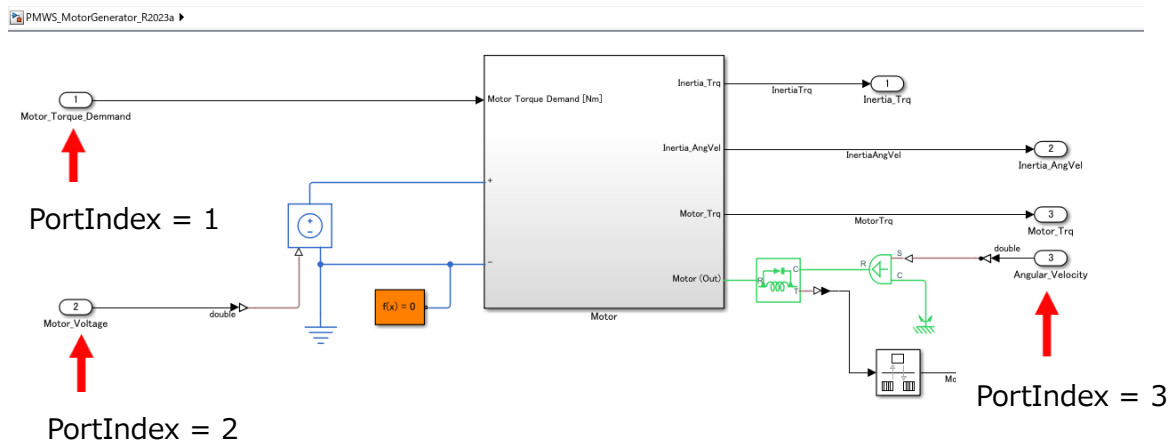


図 13 PortIndex と端子番号の対応

<テストファイル作成>

Simulink Test のファイル「Sample_Simulink_Test_motor.mldatx」を作成し、シミュレーションの設定を行う。今回は以下のスクリプトで設定を実施する。

尚、テストファイルを作成、変更するための MATLAB 関数は、下記のページに一覧化されている。

<https://jp.mathworks.com/help/sltest/referencelist.html>

※スクリプト例

```
% Simulink Test マネージャーの初期化
sltest.testmanager.clear
sltest.testmanager.clearResults
sltest.testmanager.close

% Simulink Test ファイル作成
tf_obj = sltest.testmanager.TestFile('Sample_Simulink_Test_motor.mldatx');
ts_obj = createTestSuite(tf_obj, 'Motor Test Suite');
tc_obj = createTestCase(ts_obj, 'baseline', 'Baseline Motor Test Case');

% デフォルトのテストスイート削除
tsDel = getTestSuiteByName(tf_obj, '新規テスト スイート 1');
remove(tsDel);

% 「テスト対象システム」の設定
setProperty(tc_obj, 'Model', 'PMWS_MotorGenerator_R2023b');

% 「入力」の設定
Input = addInput(tc_obj, 'Motor_SL_RefData.mat'); % 「入力ファイル」の設定
map(Input, 'Mode', 3, 'CompileModel', true); % 「マッピングモード」を"端子の順序"に設定

% 「シミュレーション出力」の設定
lgset = tc_obj.addLoggedSignalSet; % 「信号セット」の作成
blkpath = ['PMWS_MotorGenerator_R2023b/PS-Simulink', newline, 'Converter'];
bPath = Simulink.BlockPath(blkpath);
sig = lgset.addLoggedSignal(bPath, 1); % "Motor"サブシステムの 1 番目の出力信号を「シミュレーション出力」に設定

% 「ベースライン基準」の設定
baseline = addBaselineCriteria(tc_obj, 'Motor_SL_RefData.mat') % 「基準データファイル」の設定
sc = getSignalCriteria(baseline); % ベースラインクライテリア情報取得
% ベースラインの信号線評価の有効無効設定 ...今回は Motor_Trq のみ有効とする
sc(1).Enabled = false; % TrqDemmand 無効
sc(2).Enabled = false; % MotorVoltage 無効
sc(3).Enabled = false; % MotorAngVel 無効
sc(4).Enabled = false; % TrqDemmand 無効
sc(5).Enabled = false; % MotorVoltage 無効
sc(6).Enabled = false; % MotorAngVel 無効
sc(7).Enabled = false; % MotorCurrent 無効
sc(8).Enabled = false; % MotorTrq 無効
sc(9).Enabled = true; % MotorTrqOut 有効
% Motor_Trq のクライテリア設定
Val_AbsTol = 20; % 絶対誤差
Val_RelTol = 1; % 相対誤差
sc(9).AbsTol = Val_AbsTol; % 絶対許容誤差設定
sc(9).RelTol = Val_RelTol; % 相対許容誤差
```

<モデル実行>

続いて、テストファイルから、対象モデルのシミュレーションを実行する。以下のスクリプトで実施する。

※スクリプト例

```
% モデル実行
simstart=tic;
baselineObj = sltest.testmanager.run;
simend=toc(simstart);
```

<実行結果確認>

Simulink Test のテストマネージャーで、シミュレーション終了の確認と、合否確認に進めるかどうかを判断する。例えば、以下の 2 点について確認し、問題があればその時点でモデル実装にフィードバックする必要がある。

- ・シミュレーション結果に想定以上の振動（※）が発生していないこと
※マップの切り替え等の不連続性が原因となった振動が発生している、定常状態なのに振動している、など
- ・ベースライン基準をクリアしていること（①の合否基準）

結果確認のためにライブスクリプトで作成した「Sample_Simulink_Test_motor.mldatx」を開く。以下のスクリプトでファイルを開くことができる。テストマネージャーのウィンドウで、「結果とアーティファクト」を選択し、左側のツリーを展開していき、「ベースライン基準の結果」の配下の「MotorTrqOut」にチェックを入れる。

※スクリプト例

```
% Simulink Test ファイルを開く
open('./Sample_Simulink_Test_motor.mldatx');
```

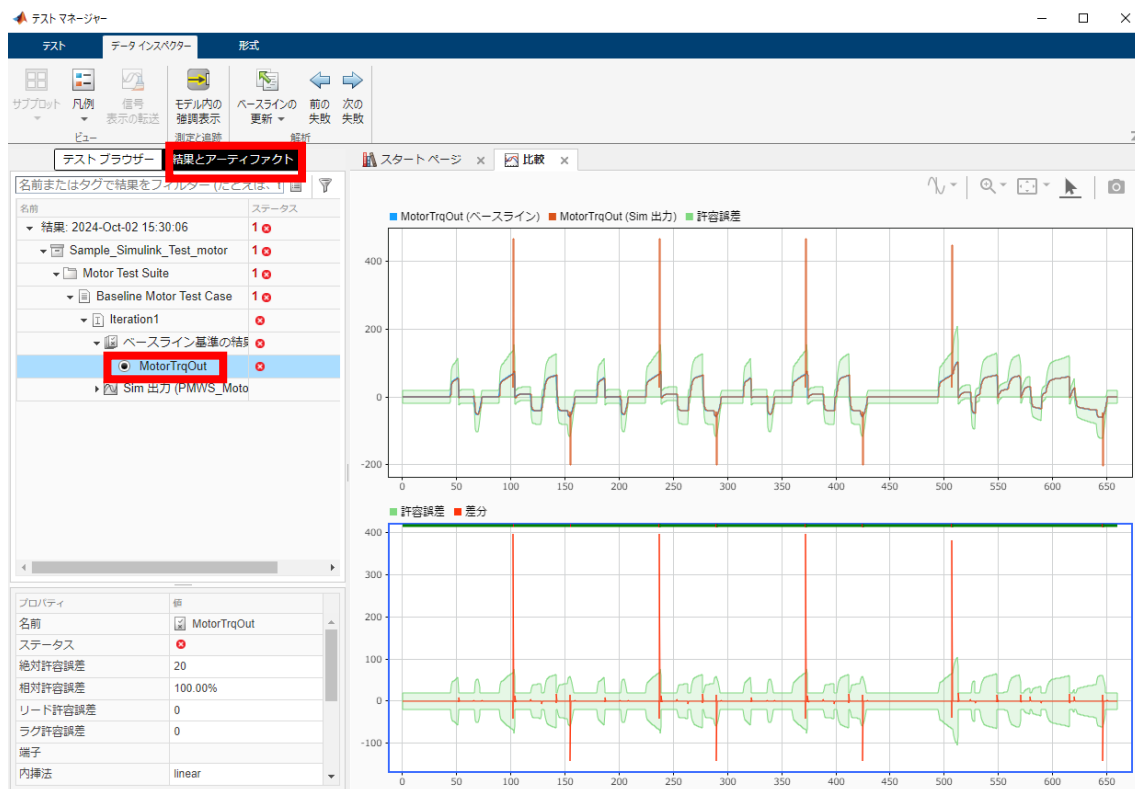


図 14 実行結果確認

右側に対象データのプロットが表示されるため、想定以上の高周波振動をしていないことを確認する。また、左側のツリーで、合否判定する全てのデータのステータスが緑になっていることを確認することで、ベースライン基準をクリアしていることを確認できる。

図 14 では、今回の題材モデルでの結果を示している。赤線の Sim 出力において、数か所の高周波振動が見られる。そのため、合否判定のステータスが赤になり、ベースライン基準をクリアできていないことが分かる。本来は、この時点で合否基準を満たせないことが明らかであるため、モデル実装に戻る必要があるが、今回は説明のためにプロセスを進める。

<ログ取得>

ライブスクリプトにシミュレーション結果をまとめるために、シミュレーション結果のデータを取得する。以下のスクリプトで Simulink Test からログデータを取得する。

※スクリプト例

```
% ログデータ取得
TestFileName = 'Sample_Simulink_Test_motor.mldatx'; % ファイル名を指定
TestFile_obj = sltest.testmanager.load(TestFileName);
TestCase_obj = TestFile_obj.getAllTestCases;
TestCaseResult_obj = TestCase_obj.getTestCaseResults; % テストケース結果の情報を取得
TestResultData = TestCaseResult_obj(end); % 最後にシミュレーションした時の結果を取得
TestResultData_iter = TestResultData.getIterationResults; % 結果が "反復" の中に含まれているため、getIterationResults で情報取得

% TestCaseResult オブジェクトの内容確認(下記コードの ';' を外すと、保存されている情報が確認できる)
TestResultData_iter.Baseline;
TestResultData_iter.SimulationMetadata;

% Simulink Test で実施したシミュレーションについて、getOutputRuns メソッドで Run オブジェクトを取得
Output_runObj = TestResultData_iter.getOutputRuns;

% Data Inspector の API を使用し、信号名を指定して信号情報を取得
MotorTrq_sigObj = getSignalsByName(Output_runObj, 'MotorTrq');
MotorTrq_out_sigObj = getSignalsByName(Output_runObj, 'MotorTrqOut');

Ref_MotorData_Trq_jdg = MotorData_Trq;%比較するデータ
MotorData_Trq_jdg = MotorTrq_out_sigObj.Values.Data;%Sim 出力
MotorData_time_jdg = MotorData_time;%時間軸
MoterData_TargetTRQ_jdg = MotorData_TrqDemmand;%目標トルク
```

参考として、テストマネージャーの UI からログデータを取得できるため、その方法を記す。テストマネージャーの左側のツリーで、出力したい信号を右クリックし、「エクスポート」を選択する。エクスポートのダイアログが表示されるため、任意の変数名を入力して「エクスポート」ボタンを押す。



図 15 テストマネージャーからのデータエクスポート

上記の方法でエクスポートしたデータは、Simulink の実行周期に従ったサンプリングになっている。リファレンスデータとサンプリングが合っていない場合は、resample 関数を使ってリサンプリングする。

<https://jp.mathworks.com/help/matlab/ref/timeseries.resample.html>

<合否確認>

シミュレーションデータとリファレンスデータを使い、合否基準の判定を行う。合否基準の⑤パワースペクトル密度、⑥周波数特性は、判定の自動化が難しいため、後述のプロット作成をした上で目視確認を行う。ここでは、合否基準①～④に対して、以下のスクリプトで合否を判定する。（Simulink Test 上で、他の合否判定も「カスタム基準」機能を使って実現可能だと思われるが、現時点ではライブスクリプトの方が使い勝手が良い。）

※スクリプト例

```
% 合否判定
% 最大瞬間誤差
err = max(abs(MotorData_Trq_jdg - Ref_MotorData_Trq_jdg))
if err>20
    disp(['最大瞬間誤差判定 NG、現在の誤差は', num2str(err), 'です'])
else
end

% RMSE(2乗平均平方根誤差)
RMSE = sqrt(sum((MotorData_Trq_jdg-
Ref_MotorData_Trq_jdg).^2)/length(Ref_MotorData_Trq_jdg))
if RMSE > 5
    disp(['RMSE 判定 NG、現在の判定値は', num2str(RMSE), 'です'])
else
end

% 相関係数
Y = MotorData_Trq_jdg; %sim 値; % 例として 1 つ目のデータを指定
X = Ref_MotorData_Trq_jdg; %ref 値; % 例として 2 つ目のデータを指定
correlation = corrcoef(Y, X); % 相関係数の計算
fprintf('相関係数: %.4f\n', correlation(1, 2));
if correlation(1,2)<0.7
    disp(strcat(['相関係数 NG、現在の相関係数は', num2str(correlation(1,2)), 'です']))
else
end

% 計算時間
simtime = simend; %tic toc で計算して、sim 時間算出
TimeRate = MotorData_time_jdg(end)/simtime
if TimeRate < 5
    disp(strcat(['計算時間 NG、現在の計算倍率は', TimeRate, '倍です']))
else
end
```

スクリプトを実行すると以下のような合否結果が得られる。合否判定のために算出した数値と、判定結果 NG の場合はメッセージが表示されるスクリプトとしている。


```

err = 396.5115
最大瞬間誤差判定NG、現在の誤差は396.5115です
RMSE = 2.7347
相関係数: 0.9967
TimeRate = 14.7005

```

図 16 合否結果

<プロット作成>

取得したデータを使って、レポート用のプロットを作成する。以下のスクリプトで合否基準①③⑤⑥のプロットを作成する。（合否基準②RMSE と、合否基準④計算時間はプロットできない。）

※スクリプト例（①最大瞬間誤差）

```

% 最大瞬間誤差
% % 使用変数
% % MotorData_time_jdg:時間軸
% % Ref_MotorData_Trq_jdg:実測 Motorトルク
% % MotorData_Trq_jdg:SimMotorトルク
% % ref_upper:Motorトルクの許容範囲(上限)
% % ref_lower:Motorトルクの許容範囲(下限)

ref_upper = Ref_MotorData_Trq_jdg+20; %許容範囲(上限側)
ref_lower = Ref_MotorData_Trq_jdg-20; %許容範囲(下限側)

figure
plot(MotorData_time_jdg , Ref_MotorData_Trq_jdg , '-r' , ...
     MotorData_time_jdg , MotorData_Trq_jdg , '-b' , ...
     MotorData_time_jdg , ref_upper , '--g' , ...
     MotorData_time_jdg , ref_lower , '--g')
xlabel('時間')
ylabel('トルク(Nm)')

```

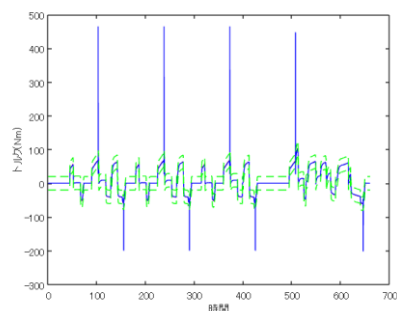


図 17 最大瞬間誤差 プロット結果

※スクリプト例（③相関係数）

```
% 相関係数
figure
scatter(MotorData_Trq_jdg,Ref_MotorData_Trq_jdg, ".")
xlabel('モータトルク(Nm)sim')
ylabel('モータトルク(Nm)実機')
lsline
```

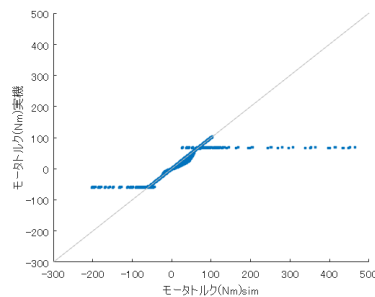


図 18 相関係数 プロット結果

※スクリプト例（⑤パワースペクトル密度）

```
% パワースペクトル密度
% データの読み込みや生成
data = MotorData_Trq;%[1, 2, 3, 4, 5]; % 例としてデータを指定

% パワースペクトル密度の計算
N = length(MotorData_time); % Length of signal
Fs = 1/(MotorData_time(2) - MotorData_time(1)); % Sampling frequency[Hz]

%% Plot Power Spectrum Density
[pxx,f] = pwelch(data,[],[],[],Fs);
figure
plot(f,pow2db(pxx))
xlabel('Frequency (Hz)')
ylabel('PSD (dB/Hz)')
```

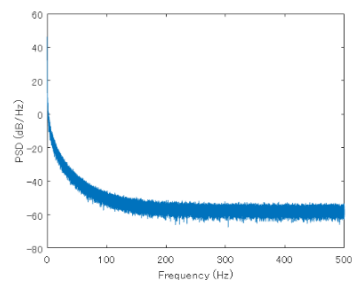


図 19 パワースペクトル密度 プロット結果

※スクリプト例 (⑥周波数特性)

```
%%% 周波数応答確認(ボード線図)シミュレーションデータ
Ts = 0.001; % データのサンプリング時間
% 窓関数適応
N = length(MoterData_TargetTRQ_jdg); % Number of samples
f = (0:N-1)*(1/(N*Ts)); % Frequency vector
% Define the window function (Hamming window in this case)
window = hamming(N);
% Apply the window to the input signal
windowed_input = MoterData_TargetTRQ_jdg .* window;
% Apply the window to the output signal
windowed_output = MotorData_Trq_jdg .* window;
% データの FFT 解析
% 入力信号
input_fft = fft(windowed_input);
input_fft_shifted = fftshift(input_fft);
input_fft_magnitude = abs(input_fft_shifted/N);
% 出力信号
output_fft = fft(windowed_output);
output_fft_shifted = fftshift(output_fft);
output_fft_magnitude = abs(output_fft_shifted/N);
% ボード線図表示
outdata = iddata(output_fft_magnitude, input_fft_magnitude, Ts); % Y, U, Ts,
create IDdata object
frd_data = etfe(outdata); % estimate
figure
bh = bodeplot(frd_data); % plot freq. response
setoptions(bh, 'FreqUnits', 'Hz') % convert the unit, (rad/s) -> (Hz)
title('ボード線図(シミュレーション)')

%%% 周波数応答確認(ボード線図)実データ
% Apply the window to the input signal
windowed_input = MoterData_TargetTRQ_jdg .* window;
% Apply the window to the output signal
windowed_output = Ref_MotorData_Trq_jdg .* window;
% データの FFT 解析
% 入力信号
input_fft = fft(windowed_input);
input_fft_shifted = fftshift(input_fft);
input_fft_magnitude = abs(input_fft_shifted/N);
% 出力信号
output_fft = fft(windowed_output);
output_fft_shifted = fftshift(output_fft);
output_fft_magnitude = abs(output_fft_shifted/N);
% ボード線図表示
outdata = iddata(output_fft_magnitude, input_fft_magnitude, Ts); % Y, U, Ts,
create IDdata object
frd_data = etfe(outdata); % estimate
figure
bh = bodeplot(frd_data); % plot freq. response
setoptions(bh, 'FreqUnits', 'Hz') % convert the unit, (rad/s) -> (Hz)
title('ボード線図(実機)')
```

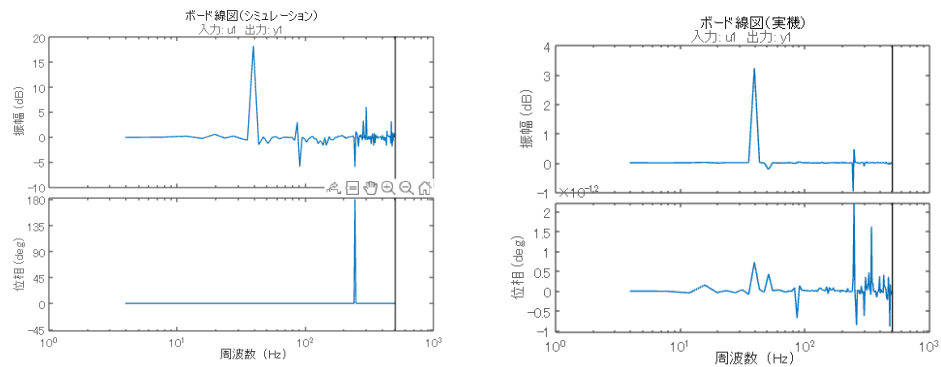


図 20 周波数特性 プロット結果

<ドキュメント作成>

Simulink Test とライブスクリプトからテストレポートをエクスポートして、内部仕様書の付属ドキュメントとして活用する。エクスポートは以下のスクリプトで実施する。

※スクリプト例

```
% Simulink Test の結果の PDF ファイル化
% ファイル名, タイトル, 作成者 の指定
report_file_path = './baselineReport.pdf';
report_title = 'MotorGenerator Test';
report_Author = 'PMWG';
% 同名ファイルを削除
if isfile(report_file_path)
    delete(report_file_path);
end
% レポート出力
sltest.testmanager.report(baselineObj, report_file_path, ...
    'Author', report_Author, 'Title', report_title, ...
    'IncludeTestResults', 0, 'IncludeComparisonSignalPlots', true, ...
    'IncludeSimulationSignalPlots', true, 'NumPlotRowsPerPage', 3);

% ライブスクリプトを PDF 化
file_name_temp = matlab.desktop.editor.getActive;
MyFileName = file_name_temp.FileName;
export(MyFileName);
```

これまで例として紹介したスクリプトの全てをまとめた、ライブスクリプトのサンプルファイルを添付する。



SampleScript.ml

x

6. まとめ

プラントモデルの検証において、特定車種の実機データが存在するユースケースに対する検証プロセスとツールチェーンの定義を行い、トライアルで妥当性の検証を行った。トライアルでは、テストデータの準備から、テストを実行、レポートを生成するまでの一連の流れを、ライブスクリプトと Simulink Testを活用して実現できることを確認した。同時に、ツール（ライブスクリプト、及びSimulink Test）の課題も明確になったため、今後の改善を期待して 7 章に要望としてまとめる。特に、Simulink Test 単独で検証プロセスが完結できるようになることを期待したい。

7. 理想のツールチェーン実現のための MATLAB への要望

<データ加工>

No.1 : Simulink Test にテストデータを読み込んだ上で、データを確認しながらフィルター/間引き処理等の加工を行いたい(Simulink Test)

<データ取り込み>

No.2 : Simscape 信号を「signal」形式に変換せずに、Simulink Test で扱いたい(Simulink Test)

<テストファイル作成>

No.3 : 登録したい信号を複数一度に登録できるようにしてほしい(Simulink Test)

※本トライアルではライブスクリプトを併用しているため、ライブスクリプト内で一括設定している

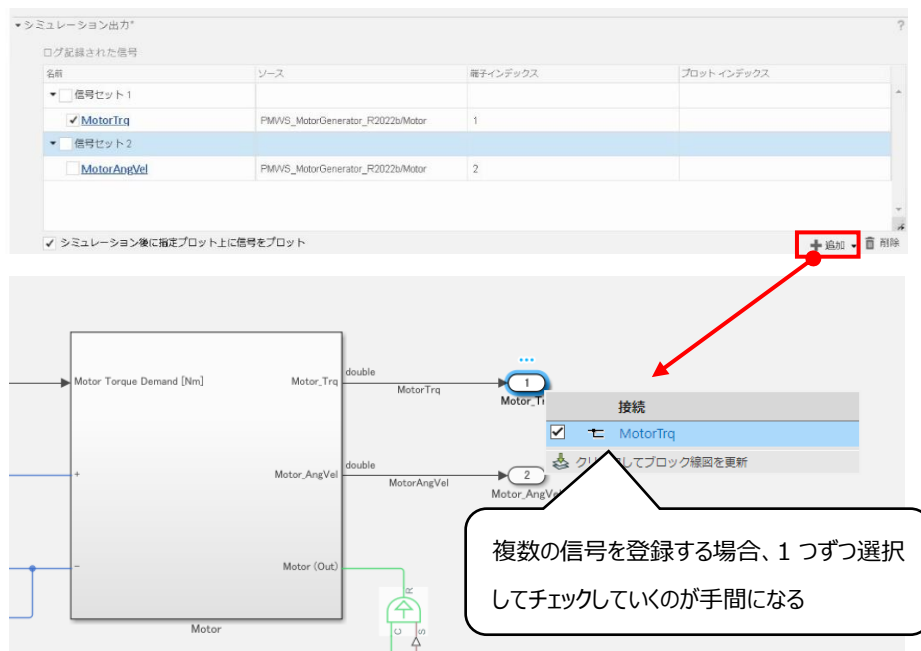


図 21 信号の登録

No.4 : ベースライン基準で検証するデータの選択・解除を簡単にしたい(Simulink Test)

※本トライアルではライブスクリプトを併用しているため、ライブスクリプト内で一括設定している

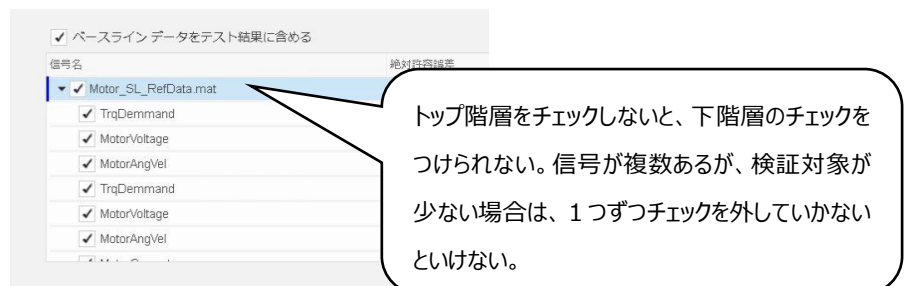


図 22 検証データの選択

No.5 : カスタム基準のスクリプトを、MATLAB スクリプトのようにデバッグできるようにしてほしい
 Simulink Test 特有のメソッドは、MATLAB スクリプト上でデバッグできないため、「ブレイクポイント
 を置ける」、「仮のデータを流せる」などができるようになってほしい(Simulink Test)

<実行結果確認>

No.6 : テスト結果の信号一覧で、モデルパス情報を常時、もしくはポップアップ表示するなどして、モ
 デルのどの信号が簡単に分かるようにしてほしい(Simulink Test)

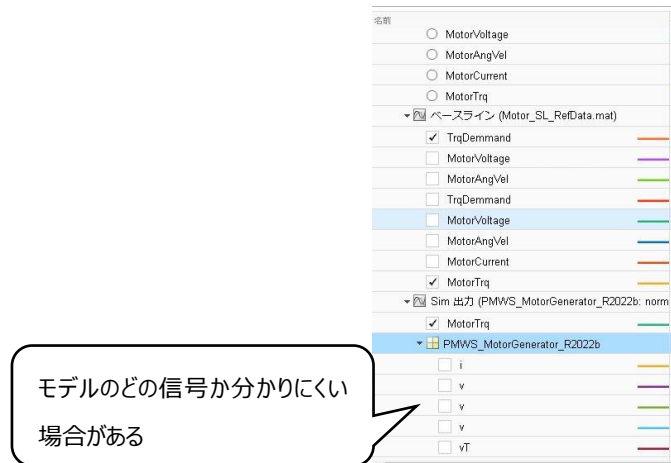


図 23 テスト結果

No.7 : シミュレーション後に、区間指定をして合否確認したい(Simulink Test)

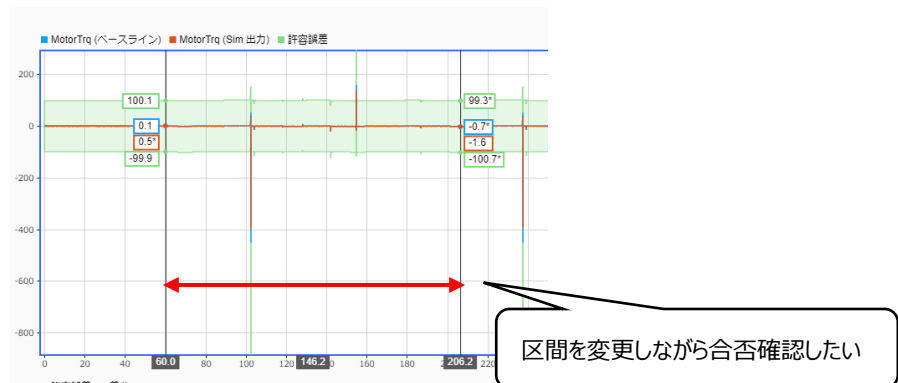


図 24 合否判定区間の指定

<ログ取得>

No.8 : Simulink Test 上のプロット結果や合否のステータス情報を、ライブスクリプト等に流用した
 い(Simulink Test)

※本トライアルではライブスクリプトを併用せざるを得なかったため、上記要望が発生した

<合否確認>

No.9 : Simulink Test のデフォルトクライテリアを増やしてほしい(Simulink Test)

※本トライアルで使った②～⑥がほしい

No.10 : ベースライン基準のステータスが「合格」となる条件を、相対/絶対許容誤差の AND 条件と OR 条件を切り替えられるようにしてほしい(Simulink Test)

<ドキュメント作成>

No.11 : Simulink Test のレポート出力時に、非出力項目をコントロールしたい(Simulink Test)

No.12 : ライブスクリプトのレポート出力時に、非出力項目をコントロールしたい(ライブスクリプト)

<その他>

No.13 : ライブスクリプトのテーブルを編集しやすくしてほしい

セルの配色、Microsoft® Excel®ファイルからのコピー & ペースト、MATLAB 関数で Excel を読み込んで自動作成、などがしたい(ライブスクリプト)

No.14 : Microsoft® Word, PDF などの外部ファイルからのインポートをしたい(ライブスクリプト)

No.15 : 仕様書をライブスクリプトで記載することを想定した場合、ライブスクリプト→Requirements Toolbox へインポートしたい(Requirements Toolbox)

8. 最後に

ご協力いただきましたメンバの皆様に感謝いたします。今後このような取り組みをされる皆様の参考になればとの思いでメンバー同、活動を進めてきました。トライアル結果は一例であり、ご覧になる皆様がそれぞれの環境に合ったさらに良い方法を見つける一助になれば幸いです。

活動参加企業（順不同）

- ・JATCO
- ・キャタピラージャパン
- ・三菱ふそう
- ・マツダ
- ・ミツバ
- ・アドビックス
- ・TTDC
- ・ダイハツ工業
- ・MathWorks
- ・デンソー
- ・デンソーテクノ