PIDゲインスケジューリングをAutotunerを用いて設計する

初期化

```
open_system(system_model_name);
controller_PID_name = 'PID_Controller';
controller_name = 'PID_AutoTuning_tester';
set_param([system_model_name, '/Controller'], 'ModelName', controller_PID_name);
```

設計課題の確認

Simulinkモデルの設計を確認する。「Controller」サブシステムと「DCDC_plant_model」サブシステムの内部の構造を理解すること。

モデルを実行し、動作を確認する。

```
sim(system_model_name);
```

```
Simulink.sdi.clearAllSubPlots;      % シミュレーションデータインスペクターのチェックを全て外す
plot_results_in_SDI;
```

シミュレーションデータインスペクターで結果を確認する。

電流負荷が+/-20[A]に変化した時と、+/-20[A]から0[A]に戻った時の応答の仕方が異なっていることがわかる。

ここで、1[A]の電流負荷のステップ変動に対する応答を確認する。

```
set_param([system_model_name, '/Reference/Signal Editor'], 'ActiveScenario', 'Simple_step');
sim(system_model_name);
```

シミュレーションデータインスペクターで結果を確認する。

このように、負荷の大きさによって応答が大きく異なる理由は、スイッチング回路による不連続性によって生じる非線形性にある。そこで今回は、負荷に流れる電流の大きさによってゲインを切り替える「ゲインスケジューリング制御器」を設計する。

ゲイン調整の準備

```
set_param([system_model_name, '/Reference/Signal Editor'], 'ActiveScenario', 'Step_current_work
load_system(controller_name);
set_param([system_model_name, '/Controller'], 'ModelName', controller_name);
```

調整の実行中、プラントモデルのEDLCの電圧が変化しないようにしたい。そのために、EDLCの容量を十分大きな値に設定する。

```
set_slddVal('system_data.sldd', 'EDLC_Capacitance', 100);
```

モデルをゲイン調整用に設定する。

```
Iout_ref = 20;
open_system([system_model_name, '/Reference/dist_cur_swith']);
```

操作点のゲインを求める

電流I_{out}の値が以下の値にあるときを調べる。

```
Iout_op = [
    -20; -16; -12; -8; -4; -2; -1;
    1; 2; 4; 8; 12; 16; 20];
```

Iout_opのパターン数分、シミュレーション設定変数を作成する。

```
for i = 1:numel(Iout_op)
    simin(i) = Simulink.SimulationInput(system_model_name);

    % 必要に応じてアクセラレータ、ラピッドアクセラレータモードを使用する。
    % 使用する場合は、以下のどちらかをコメントアウトする。
%     simin(i) = simin(i).setModelParameter('SimulationMode', 'accelerator');
%     simin(i) = simin(i).setModelParameter('SimulationMode', 'rapid-accelerator');

    simin(i) = simin(i).setVariable('Iout_ref',Iout_op(i));
end
```

パターン数分のシミュレーションをまとめて実行する。Parallel Computing Toolboxがインストールされているとき、parsimは自動的に並列ワーカーを立ち上げて並列実行を行う。

```
save_system(system_model_name, [], 'OverwriteIfChangedOnDisk', true);
simout = parsim(simin, 'ShowProgress', 'on');
```

並列のparsim実行中、Windowsのタスクマネージャーなどを起動し、CPUやメモリの消費量を確認すること。

調整結果の確認と保存

得られた比例、積分ゲインをP_gain_table、I_gain_tableに格納する。

```
P_gain_table = zeros(size(Iout_op));
I_gain_table = zeros(size(Iout_op));

for i = 1:numel(Iout_op)
    dataSet = simout(i).logsout;
    pid_gains_sig = dataSet.getElement('pid_gains');
    P_gain_table(i) = pid_gains_sig.Values.P.Data(1);
```

```
    I_gain_table(i) = pid_gains_sig.Values.I.Data(1);
end
```

テーブル値をグラフで確認する。

```
plot(P_gain_table)
plot(I_gain_table)
```

結果をファイルに保存する。

```
save(fullfile(pjObj.RootFolder, filesep, 'data', filesep, 'scheduled_gain_data.mat'), ...
    "P_gain_table", "I_gain_table", "Iout_op");
```

モデルの変更を戻す。

```
set_slddVal('system_data.sldd', 'EDLC_Capacitance', 0.1);
open_system([system_model_name, '/Reference/dist_cur_swith']);
```

ゲインスケジューリング制御

設計した制御器の動作を確認する。

```
set_param([system_model_name, '/Controller'], 'ModelName', 'GainScheduled_PID_Controller');
sim(system_model_name);
Simulink.sdi.clearAllSubPlots;
plot_results_in_SDI;
```