PLECS
Tutorial

PLECS XML-RPC Interface and Controller Design in Python

PythonによるPLECS XML-RPCインタフェースとコントローラの設計

- Python Controls Libraryを使用した3相グリッド接続VSIの電流制御設計 -

Tutorial Version 1.1

KESCO KEISOKU ENGINEERING SYSTEM
計測エンジニアリングシステム株式会社
https://kesco.co.jp

1 はじめに

PLECS StandaloneのXML-RPCインタフェースを使用すると、外部プログラムを使用してPLECS Standaloneとの間でデータを送受信できます。Python、Java、C++、Rubyなどの多くのプログラミング言語では、XML-RPCクライアントを設定するための標準のXML-RPCライブラリが提供されています。このチュートリアルでは、Python 3を使用してPLECSシミュレーションを起動し、後処理したシミュレーション結果を取得します。このチュートリアルでは、次の内容を学習します:

- XML-RPCインタフェースをPLECS Standaloneにどのように組み込むか、またシミュレーションフローについての説明。
- Python Control System Libraryを使用して簡単な電流制御図を設計する方法。
- XML-RPCインタフェースを使用してPLECSと対話する方法(シミュレーションの開始とデータの読み込み)。

このチュートリアルを開始する前に、添付ファイルが以下の作業ディレクトリにあることを確認してください:

- `tutorial_functions_library.py`
- `xmlrpc_controller_design.plecs`
- `xmlrpc_controller_design_1.py`
- `xmlrpc_controller_design_2.py`
- `xmlrpc_controller_design.py`

1.1 Python環境

XML-RPCクライアントライブラリ(`xmlrpc.client`)はPython 3にデフォルトで同梱されているため、XML-RPCを介してPythonからPLECSへの接続を確立するために追加のソフトウェアをインストールする必要はありません。このチュートリアルでは、コントローラの設計に重点を置いているため、追加のライブラリが必要になります。これらのライブラリは以下のとおりです:

- **os**: オペレーティング システムに依存する機能を移植可能な方法で使用できるように提供するモジュール。これは Python 3 ディストリビューションにデフォルトで含まれています。
- **matplotlib**: numpyをベースにしたPython用プロットライブラリ。
- **control**: フィードバック制御システムの分析と設計のための基本的な操作を実装するパッケージ。
- **scipy.io**: さまざまなファイル形式からデータを読み取ったり、さまざまなファイル形式にデータを書き込んだりするためのScipyモジュール。
- **io**: さまざまな種類のI/Oを処理するための主要な機能を提供するモジュール。

matplotlibライブラリとcontrolライブラリ(ネイティブのPython 3ライブラリではありません)は、Python 3パッケージ管理ツール`pip3`を使用してインストールできます。

```
pip3 install matplotlib control
```

このチュートリアルでは、`tutorial_functions_library.py`が上記ライブラリをインポートすることに注意してください。

```
import matplotlib.pyplot as plt
import control as ct
import os
import scipy.io as sio
import io
```

1.2 PLECS XML-RPCインタフェース

PLECSには、特定のXML-RPCポート上のクライアントからのシミュレーションコマンドをリッスンするXML-RPCサーバが組み込まれています。利用可能なXML-RPCコマンドの全セットは、PLECSユーザマニュアルに記載されています。

PLECSシミュレーションを特定のパラメータセットで開始するための主なコマンドは、以下のとおりです:

```
plecs.simulate('mdlName', optStruct)
```

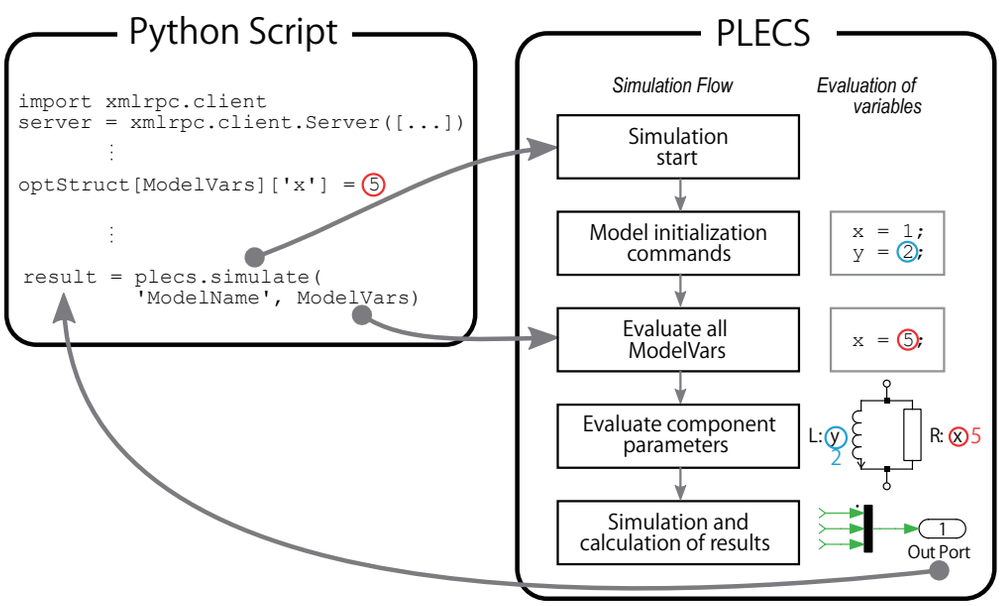
オプションの引数*optStruct*は、PLECSモデルで定義されたパラメータを上書きすることができます。これには、個々のシミュレーションパラメータが含まれるフィールド*ModelVars*が含まれます。値は、数値スカラ、ベクトル、行列、3D配列、または文字列にすることができます。さらに、ソルバパラメータも調整する必要がある場合は、オプションフィールド*SolverOpts*をシミュレーションに渡すことができます(このチュートリアルでは実装していません)。*optStruct*を使用してモデルパラメータを上書きすると、モデルファイルを変更することなく、異なるパラメータで複数のシミュレーションを連続して実行できます。

パラメータ値の上書きとは、モデル初期化コマンドが最初に実行され、次に*optStruct*内で指定されたパラメータが適用され、最後にコンポーネントパラメータが評価されることを意味します。

例

次の図1は、シミュレーションフローの簡単な例を示しています。

図1: PLECS Standalone XML-RPCインタフェースの概要と、パラメータ値の設定とシミュレーション結果の読み戻しの簡単な例



1 Pythonスクリプトでは、構造体`ModelVars`はパラメータ`x`と値`5`で定義されます。

2 XML-RPCコマンドの`simulate`を使用して、指定されたモデルのPLECSシミュレーションを開始し、`ModelVars`構造体に指定されたパラメータをPLECSに送信します。

3 PLECSシミュレーションは、変数`x`と`y`が定義されているモデルの初期化コマンドを評価することから始まります。

4 次に、`ModelVars`構造体が評価され、変数`x`に割り当てられた値`1`は、外部のPythonスクリプトで定義されている値`5`に上書きされます。`y`値は変更されません。

5 次に、すべての回路パラメータが評価されます。インダクタンス値は2H、抵抗の値は5Ωです。

6 最後にシミュレーションが実行されます。シミュレーションが終了すると、Out Portに渡されたすべてのデータが XML-RPC 経由でPythonスクリプトに返送されます。このシミュレーションデータは、戻り値の`result`で利用できます。

この例では、PLECSシミュレーションのパラメータをXML-RPC上で定義する方法を示しています。このチュートリアルでは同じ概念を使用しますが、変数名と値は異なります。

XML-RPCインタフェースの設定

あなたのタスク: モジュールを読み込み、XML-RPC 続を設定します。

1 まず、選択したエディタまたはIDEで新しいPythonスクリプトを作成する必要があります。次に、チュートリアルで使用するモジュールと関数をインポートするために次のコードを追加します:

```
1 from tutorial_functions_library import *
```

2 Pythonスクリプトを実行する前にPLECSを起動しておいてください。PLECSの組み込みXML-RPCインタフェースを有効にするには、**PLECS設定**ウィンドウの**General**タブに移動します。**RPCインターフェイスポート番号**のチェックボックスをクリックして、標準ポート1080でXML-RPCインタフェースを有効にします。ポート1080がすでに別のアプリケーションによって使用している場合、ポート番号を別の値に変更してください。XML-RPCサーバを起動するためには、PLECSを再起動する必要があります。

PLECSへのXML-RPC接続は、PLECSと同じマシン上で動作するクライアントからのみ許可されること、つまり、Pythonスクリプトと PLECSシミュレーションは同じマシン上で実行する必要があります。したがって、XML-RPC接続は、常にサーバ URL で`localhost`を使用して開始する必要があります。

3 XML-RPCモジュールをインポートし、PLECSとのクライアントサーバ接続を開始します。

```
2 import xmlrpc.client
3 server = xmlrpc.client.Server("http://localhost:1080/RPC2")
```

2 Python Control System Libraryによる電流制御設計

このチュートリアルの技術的な内容は[1]に示されている解析に従います。この解析では、RLフィルタを介したグリッド接続の3相電圧源インバータのモデリングを示しています。このシステムを図2に示します。モデリングが実行されると、dq同期参照フレームの電流コントローラーが設計されます。このチュートリアルでは、[1]で提供されている設計ガイドラインとPython Control System Libraryを使用して、電流コントローラを設計し、指定したパラメータセットに対するシステムの予想される動的応答を取得します。

図2: RLフィルタによる系統連系3相電圧源インバータのモデル化システム ([1]の図6)

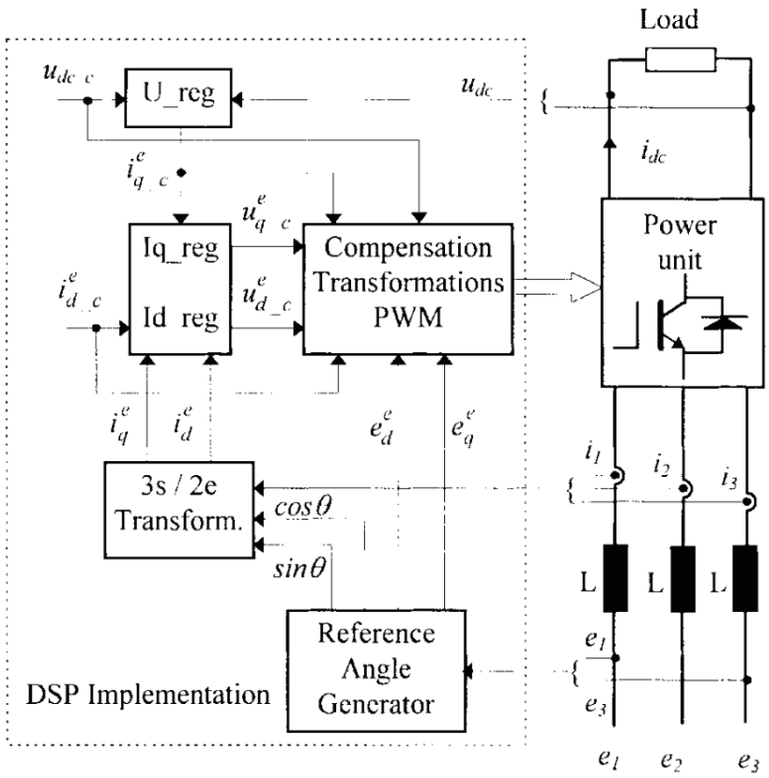
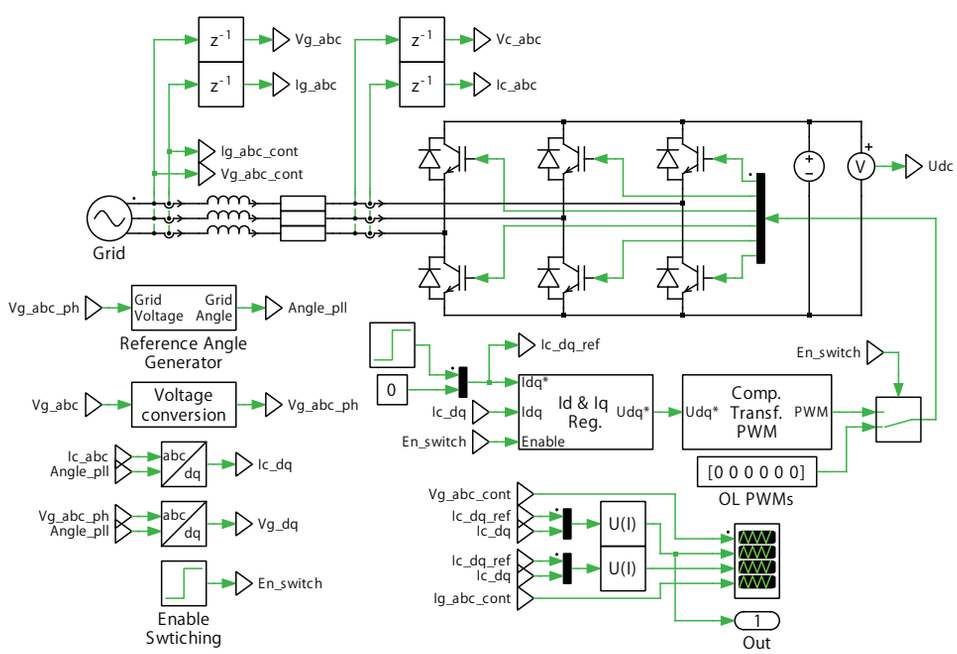


図3: RLフィルタによる系統連系3相電圧源インバータのPLECSモデルのメイン回路図



このチュートリアルでは、図2に示すシステムを実装したPLECSモデルをファイル`xmlrpc_controller_design.plecs`で提供しています。[1]では、DC電圧レギュレータの制御設計の解析も行いますが、これはチュートリアルの範囲外です。PLECSモデルを簡素化するため、DCリンクのキャパシタンスとDC負荷は、DC電圧源に置き換えられています。

あなたのタスク: あなたのタスクは、Python Control System Libraryを使用して、システムのダイナミクスをモデリングする伝達関数(TF)を計算し、予想される周波数応答と過渡応答を取得することです。

1 システムで使用するパラメータを定義する必要があります。このために、必要なすべてのパラメータが定義されているユーザ定義関数`SystemParameters()`を使用します:

```
4 Param = SystemParameters()
```

`tutorial_functions_library.py`ファイル内の`SystemParameters()`関数を変更することで、値を変更できます。この関数は、次のパラメータ値を含む辞書を返します:

- T_s = 5e-5はスイッチング周期です。
- $T_{ei} = 1.5 \cdot T_s$はサンプルアンドホールドとPWM変調器のグループ化されたTFの近似時定数です。
- $T_{rl} = L / R$はプラントの時定数です。このシステムのプラントはRLグリッドフィルタです。
- $K_{rl} = 1/R$はプラントゲインです。
- $K_{PWM} = 1$はPWM変調器のゲインです。
- $D = \sqrt{2}/2$は、閉ループ伝達関数の減衰係数です。
- $K_i = T_{rl} /(4D^2 \cdot T_{ei} \cdot K_{PWM} \cdot K_{rl})$はPI比例ゲインです。
- T_iはPI積分定数です。

[1]で説明されているように、設計目的に応じて2つの異なるT_i値を選択できます。$T_i = T_{rl}$は最適な電流指令値追従に使用し、$T_i = 10\ T_{ei}$は、外乱除去を改善します。

2 設計された閉ループ電流制御の動的性能を得るためには、システムの伝達関数を計算する必要があります。伝達関数の定義を容易にするために、伝達関数変数を作成します。次のコードを使用します:

```
5 s = ct.tf('s')
```

3 サンプルアンドホールド、PWM遅延、およびRLフィルタの伝達関数を定義できるようになりました。[1]に示すように、遅延はシステムの最小の時定数です。簡略化のため、これらをグループ化して等価な時定数T_{ei}で一次近似を使用できます。RLフィルタの伝達関数は、$Plant_TF(s) = \frac{i_L(s)}{v_L(s)} = \frac{1}{L*s+R} = \frac{K_{rl}}{T_{rl}*s+1}$として計算できます。これらの計算には次のコードを使用します:

```
6 Delays_TF = 1/(1+s*Param['Tei'])
7 Plant_TF = Param['Krl']/(1+s*Param['Trl'])
```

4 PIレギュレータの2つのバリエーション、つまり最適な電流指令値追従用に設計されたものと、外乱除去を改善したものを使用して、システムのダイナミクスを解析します。つまり、3つのTFの2つの異なるセットを計算する必要があります: 閉ループTF $H_{ci}(s) = i_c(s)/i^*_c(s)$、開ループTF $H_{oi}(s) = i_c(s)/i^*_c(s)$、および外乱除去TF $H_{di} = i_c(s)/v_g(s)$です。PIゲインを入力として伝達関数を計算するPython関数(`LoopTFsCalculation_PIgains()`)と、ループ遅延とプラントの伝達関数を開発しました。この関数は、`PI`、`Hoi`、`Hci`、`Hdi`、`Plant`、`Delays`のエントリと、計算された関連する伝達関数を含む辞書を返します。次のコードを使用します:

```
8 TFs = LoopTFsCalculation_PIgains(Param['Ki'], Param['Ti'], Plant_TF, Delays_TF)
```

5 [1]で説明されているように、$H_{ci}(s) \approx \frac{1}{s \cdot T_{et}+1}$と$T_{et} = 2 \cdot T_{ei}$の方程式で与えられる閉ループ伝達関数の1次近似を計算することもできます:

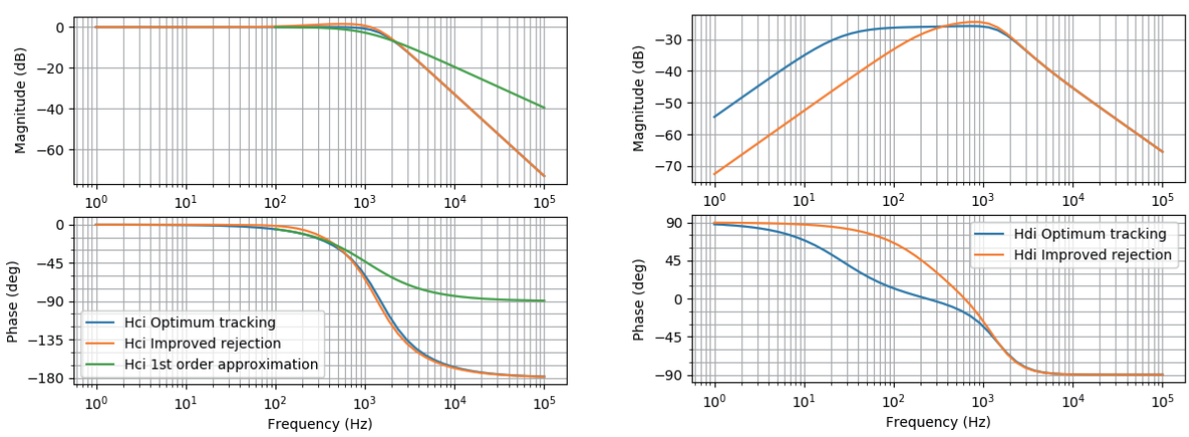
```
Param['Tet'] = 2*Param['Tei']
TFs['Hci'].append(1/(1+s*Param['Tet']))
```

6 システムの伝達関数が計算されたので、閉ループ電流制御の周波数応答を取得できます。これを行うには、Control System Libraryに含まれる関数`bode()`を使用します。提供された伝達関数のステップ応答を含むプロットを作成する Python関数`PlotFreqResponses()`を開発しました。この関数では、図のタイトルを文字列として、および凡例に表示される名前を含む文字列のリストも必要です。このこれを行うには次のコードを使用します:

```
FigNameFreq1 = 'Fig: Frequency response of closed-loop current control transfer functions'
LegendFreq1 = ['Hci Optimum tracking','Hci Improved rejection','Hci 1st order approx.']
PlotFreqResponses(TFs['Hci'],FigNameFreq1,LegendFreq1)
FigNameFreq2 = 'Fig: Frequency response of disturbance rejection transfer functions'
LegendFreq2 = ['Hdi Optimum tracking','Hdi Improved rejection']
PlotFreqResponses(TFs['Hdi'],FigNameFreq2,LegendFreq2)
```

周波数応答を図4に示します。どちらのシステムも約1000Hz(スイッチング周波数20kHz)の同様の電流制御帯域を達成しているのに対し、外乱除去の改善のためにPIを調整したシステムでは、オーバーシュートが大きくなることがわかります。ただし、この劣る電流追従性能は、大幅に優れた外乱除去能力によって補償されます。外乱除去の改善に調整された電流レギュレータは、最適な追従のために調整された電流レギュレータよりも8倍速く電圧摂動を解消できると期待できます。

図4: 異なる電流制御器設計における閉ループ$H_{ci}(s)$と$H_{di}(s)$の周波数応答の比較

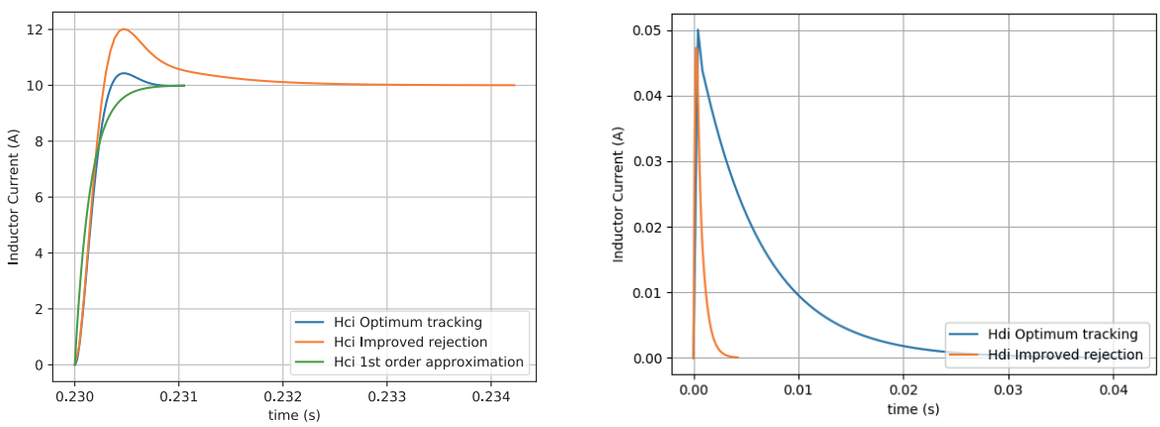


7 解析を続けると、伝達関数から、コントローラの電流基準の変化に対するシステムの過渡応答やグリッド電圧も取得できます。これを行うには、Control System Libraryに含まれている関数`step_response()`を使用します。同様に、前の演習と同様に、提供された伝達関数のステップ応答を含むプロットを作成する Python関数`PlotFreqResponses()`を開発しました。この関数では、図のタイトルを文字列として、凡例に表示される名前を含む文字列のリスト、および2つの値(ステップの発生時刻と入力ステップの変化の大きさ)のリストも必要です。さらに、この関数は、リスト内のプロットに使用されたデータを返します。このデータは、後でこの結果をPLECSのシミュレーション結果と比較するために必要になります。これを行うには次のコードを使用します:

```
17 TitleStep1 = 'Fig: Step response of closed-loop current control transfer functions'
18 LegendStep1 = ['Hci Optimum tracking','Hci Improved rejection','Hci 1st order approx.']
19 XYLabels1 = ['time (s)','Inductor Current (A)']
20 StepData_Hci = PlotStepResponses(TFs['Hci'],TitleStep1,LegendStep1,XYLabels1, ...
21  [Param['t_stepRef'],Param['Istep']])
22 TitleStep2 = 'Fig: Step response of disturbance rejection transfer functions'
23 LegendStep2 = ['Hdi Optimum tracking','Hdi Improved rejection']
24 XYLabels2 = ['time (s)','Inductor Current (A)']
25 StepData_Hdi = PlotStepResponses(TFs['Hdi'],TitleStep2,LegendStep2,XYLabels2,[0,1])
```

設計された電流コントローラの予想されるステップ応答を図5に示します。ここでは、外乱除去に最適化されたPIレギュレータは電流追従性能が低いものの、外乱の軽減に関しては他の設計に比べて大幅に改善されていることがより明確にわかります。

図5: 異なる電流制御器設計における閉ループ$H_{ci}(s)$と$H_{di}(s)$のステップ応答の比較



この段階では、コードは`xmlrpc_controller_design_1.py`と同じになっているはずです。

3 PLECSシミュレーションを起動してPythonでデータを取得

このセクションでは、システムのPLECSモデルのシミュレーションを起動し、後処理用のシミュレーションデータを取得する方法を説明します。

あなたのタスク: タスクは、シミュレーション対象のPLECSモデルの読み込み、使用するシミュレーションパラメータの定義、PLECSモデルのシミュレーション、PLECSモデルの終了から構成されます。

1 XML-RPCサーバが起動し、接続が確立されると、シミュレーションを行うPLECSモデルを読み込むことができます。現在動作しているPythonスクリプトとPLECSモデルが同じフォルダにある場合は、`os`モジュールを使用することができます。現在の作業ディレクトリを取得するのに必要な関数は、`path.abspath()`と`getcwd()`です。次に、現在の作業ディレクトリとファイル名を使用してPLECSファイルパスをビルドします:

```
26 full_path = os.path.abspath(os.getcwd())
27 file_name = 'xmlrpc_controller_design'
28 server.plecs.load(full_path+'/'+file_name)
```

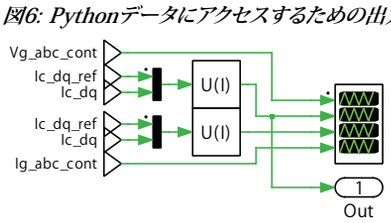
2 シミュレーションを開始する前に、使用するシミュレーションパラメータを定義する必要があります。T_iゲインが異なる2つのPIレギュレータを設計しました。したがって、両方のレギュレータをシミュレーションするために、同じPLECSモデルを使用しますが、T_iについては異なる値をシミュレーションに渡します。Pythonの関数`LoadSimParameters()`を使用して、モデルパラメータ("`Param`")を使用して構造体変数を作成します。これらは、以前に定義され、理論分析に使用されたものと同じ値です。この関数は、セクション1.2で定義された変数構造体をXML-RPCシミュレーション コマンドで使用する辞書を返します:

```
29 opts = LoadSimParameters(Param)
```

3 シミュレーション モデルの最上位の回路図に出力ポートが存在する場合、シミュレーションコマンドは"Time"と"Values"の2つのフィールドで構成される構造体を返します。"Time"は、各シミュレーションステップのシミュレーション時間値(秒単位)を含むベクトルです。配列"Values"の行は、出力ポートに表示される信号値の大きさで構成されます。信号の順序はポート番号によって決まります。図6に示すように、PLECSモデルの**出力信号ポート**ブロックには、d軸の電流制御リファレンスの信号と、測定されたd軸の電流が含まれています。信号と時間情報の両方のシミュレーション結果を格納する変数を定義する必要があります。そのために、次のコードを使用します:

```
30 Time = []
31 Ic_d_ref_sim = []
32 Ic_d_ref_sim = []
```

図6: Pythonデータにアクセスするための出力ポートを含むシミュレーションモデルのトップレベルの概略図



4 シミュレーションは、コマンド`Data = server.plecs.simulate('mdlName', optStruct)`で起動します。このプロセスを2回繰り返す必要があり、PLECSモデルには、以前に計算したT_iの異なる値を含む異なる*optStruct*を渡します。このコマンドでは、*mdlName*はシミュレーションするモデルの名前、*optStruct*はモデルパラメータを含む変数構造体、およびシミュレーションデータの形式タイプです。データ転送のプロセスがより高速であるため、"MatFile"データ形式を使用します。`Data_sim1`および`Data_sim2`はPLECSシミュレーションの結果を辞書に保存します。コードは次のようになります:

```
opts['ModelVars']['Ti'] = Param['Ti'][0]
opts['OutputFormat'] = 'MatFile'
Data_raw = server.plecs.simulate(file_name, opts).data
Data_sim1 = sio.loadmat(io.BytesIO(Data_raw))
Time.append( Data_sim1['Time'][0] )
Ic_d_ref_sim.append( Data_sim1['Values'][0] )
Ic_d_sim.append( Data_sim1['Values'][1] )
opts['ModelVars']['Ti'] = Param['Ti'][1]
Data_raw = server.plecs.simulate(file_name, opts).data
Data_sim2 = sio.loadmat(io.BytesIO(Data_raw))
Time.append( Data_sim2['Time'][0] )
Ic_d_ref_sim.append( Data_sim2['Values'][0] )
Ic_d_sim.append( Data_sim2['Values'][1] )
```

5 シミュレーション結果が得られたので、次のコマンドを使用してシミュレーションモデルを閉じることができます:

```
server.plecs.close('mdlName')
```

ここで、*mdlName*は閉じたいモデルの名前です。このスクリプトでは、名前は変数`file_name`に格納されます:

```
server.plecs.close(file_name)
```

この段階では、コードは`xmlrpc_controller_design_2.py`と同じになっているはずです。

あなたのタスク: シミュレーション結果が得られたので、PLECSのシミュレーション結果と以前に得られた理論応答を比較し、システムモデリングとコントローラ設計の精度を評価することができます。

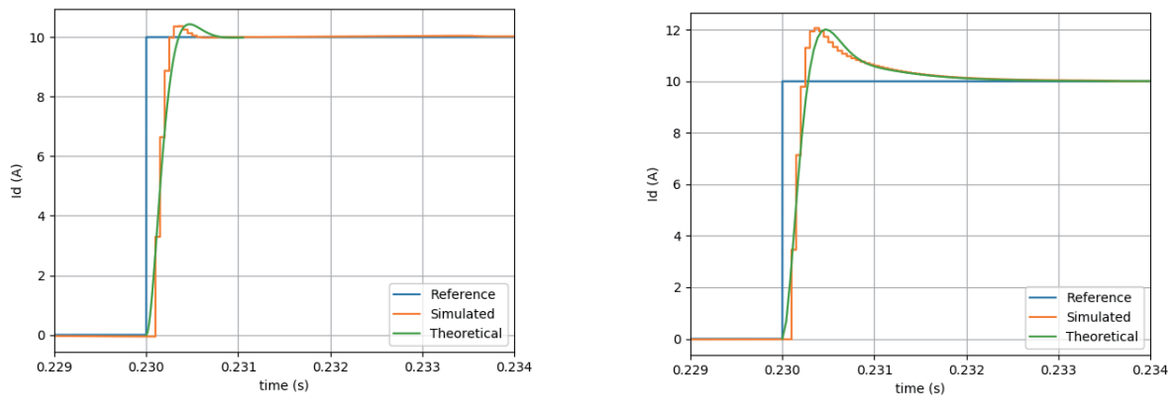
1 理論上の過渡応答とシミュレートした過渡応答を比較するには、プロットするデータを含む[X,Y]ペア値のリストを必要とする関数`PlotSimResults()`を使用します。さらに、図のタイトルを文字列として指定し、凡例に表示する名前を含む文字列のリスト、およびx軸とy軸のラベルを含む文字列のリストを指定する必要があります。この関数は、図形のハンドルを返します。

これで、設計したPI電流レギュレータの理論的なステップ応答とシミュレーションによるステップ応答を比較するプロットを取得する準備が整いました:

```
Plot_XY_Pairs = [[Time[0],Ic_d_ref_sim[0]],[Time[0],Ic_d_sim[0]],StepData_Hci[0]]
TitleResults = 'Fig: Theoretical responses Vs Simulated model, PI reg. 1'
LegendResults = ['Reference','Simulated','Theoretical']
LabelResults = ['time (s)','Id (A)']
plt1 = PlotSimResults(Plot_XY_Pairs,TitleResults,LegendResults,LabelResults,Param)
Plot_XY_Pairs = [[Time[1],Ic_d_ref_sim[1]],[Time[1],Ic_d_sim[1]],StepData_Hci[1]]
TitleResults = 'Fig: Theoretical responses Vs Simulated model, PI reg. 2'
LegendResults = ['Reference','Simulated','Theoretical']
LabelResults = ['time (s)','Id (A)']
plt2 =PlotSimResults(Plot_XY_Pairs,TitleResults,LegendResults,LabelResults,Param)
plt2.show()
```

得られた数値と応答の比較を図7に示します。

図7: 10Aの電流指令値のステップに対する過渡応答との比較(両設計の電流レギュレータ)。左は最適な電流追従を実現するPIレギュレータ設計で、右は外乱除去を向上させるために設計したPIレギュレータ。



この段階では、コードは`xmlrpc_controller_design.py`と同じになっているはずです。

4 参考文献

[1]Blasko, V., & Kaura, V., “A new mathematical model and control of a three-phase AC-DC voltage source converter,” *IEEE Transactions on Power Electronics*, Jan. 1997, pp. 116–123.

改訂履歴:

Tutorial Version 1.0　初版

Tutorial Version 1.1　バイナリ形式でPLECSにデータを転送

Plexim

Pleximへの連絡方法:

☎ +41 44 533 51 00　Phone

+41 44 533 51 01　Fax

✉ Plexim GmbH　Mail

Technoparkstrasse 1

8005 Zurich

Switzerland

@ info@plexim.com　Email

http://www.plexim.com　Web

KESCO KEISOKU ENGINEERING SYSTEM

計測エンジニアリングシステム株式会社

https://kesco.co.jp

PLECS Tutorial