Appendix

1. FOMCON vs FOMCONpy

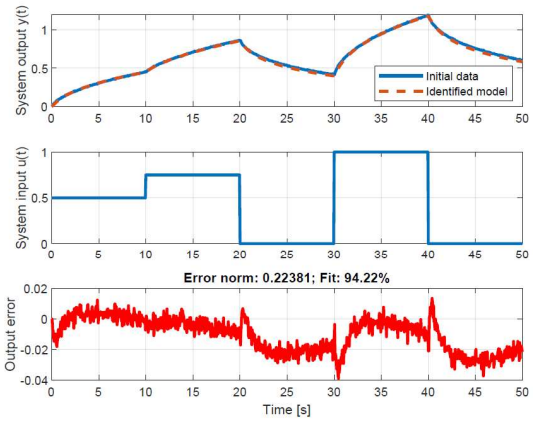
To certify that *FOMCONpy* performance is as good as its counterpart FOMCON, The author did performance tests with time domain identification. Using same data as used in section 5.3. The performance criteria was percentage fitness (3.8) discussed in section 3.3. Using Grunwald Letnikov simulation ,comparison was done for Levenberg Marquardt and Trust Region Reflective optimization method. For each method we compared the outputs percentage fitness for different optimized parameter: Coefficients only, Exponents only and Both (Coefficients + Exponents).

To obtain the FOMCONpy identification results and plots, the user can run the `test()` function in the `fomconoptimize` module which has the following commands:

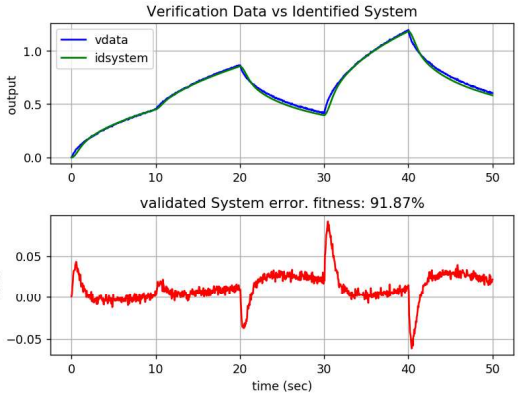
```
def test():
    result, counter = [], 1
    guessset = newfotf('-2s^{0.63}+4', '2s^{3.501}+3.8s^{2.42}+2.6s
   ^{1.798}+2.5s^{1.31}+1.5', 0)
    guessset.numberOfDecimal = 5
    for j in [optAlgo.LevenbergMarquardt]:
        for k in [optFix.Free, optFix.Coeff,optFix.Exp]:
            for l in [simMethod.grunwaldLetnikov]:
                polyfixset = [0, 0]
                optiset = opt(guessset, l, j, k, polyfixset)
                print('\n{0}: Computing settings: {1}, {4}, {2}, {3}\n'
   .format(counter,  j, k, polyfixset,l))
```

```
                res = fid('dataFiles\idenData.xlsx', 'dataFiles\
ValiData.xlsx', optiset, plot=[False, False], plotid=[False, True],
 cleanDelay=[True,2.5])
                res.G.numberOfDecimal = 5
                result.append(res)
                print(res.G, "\n\n")
                counter+=1

 guessset = newfotf('2s^{0.63}+4', '2s^{3.501}+3.8s^{2.42}+2.6s
^{1.798}+2.5s^{1.31}+1.5', 0)
 guessset.numberOfDecimal = 5
 for j in [optAlgo.TrustRegionReflective]:
     for k in [optFix.Free, optFix.Coeff,optFix.Exp]:
         for l in [simMethod.grunwaldLetnikov]:
             polyfixset = [0, 0]
             optiset = opt(guessset, simMethod.grunwaldLetnikov, j,
k, polyfixset)
             print('\n{0}: Computing settings: {1}, {4}, {2}, {3}\n'
.format(counter,  j, k, polyfixset,l))
             res = fid('dataFiles\idenData.xlsx', 'dataFiles\
ValiData.xlsx', optiset, plot=[False, False], plotid=[False, True],
 cleanDelay=[True,2.5])
             res.G.numberOfDecimal = 5
             result.append(res)
             print(res.G, "\n\n")
             counter+=1

 return result
```

Listing 1.1. Using the `fid()` function to generate identification models and plots for comparison with MATLAB



(a) Matlab LM Both: 94.22% fitness (better).



(b) Python LM Both: 91.87% fitness.

1.1 Identification using Levenberg Marquardt method

Initial Guess used for bench-marking purpose:

$$G(s) = \frac{-2s^{0.63} + 4}{2s^{3.501} + 3.8s^{2.42} + 2.6s^{1.798} + 2.5s^{1.31} + 1.5}$$

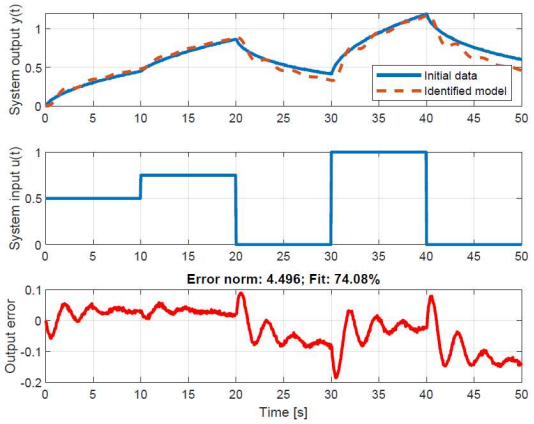
1.1.1 Both – Coefficients + Exponents

1. Matlab LM Both: 94.22% fitness (better).

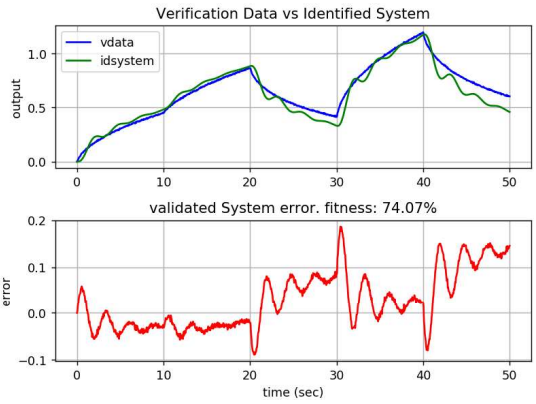
$$\frac{4.1559s^{0.050734} - 2.0996s^{1.2942\times10^{-07}}}{2.256s^{1.6514} + 2.3271s^{0.8797} + 3.7516s^{0.6457} + 2.8353s^{0.6421} + 0.22734s^{5.2447\times10^{-10}}}$$

2. Python LM Both: 91.87% fitness.

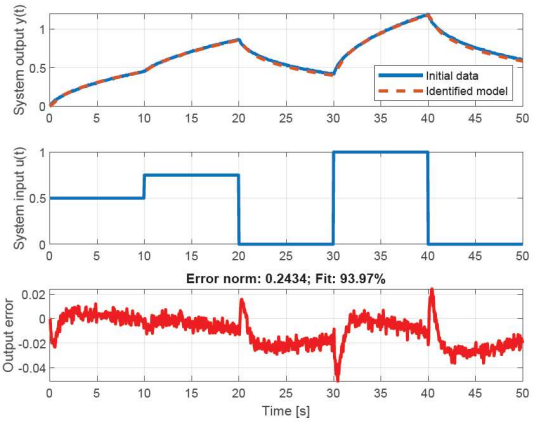
$$\frac{-2.91799s^{-0.04681} + 5.35592}{1.15126s^{2.5615} + 4.74487s^{1.59725} + 3.97537s^{0.62262} + 5.76595s^{0.61868} + 0.30441}$$



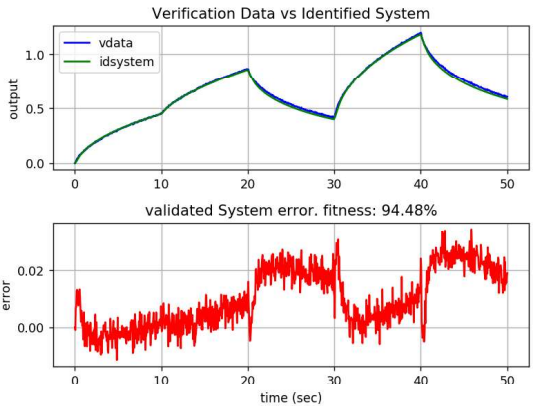
(a) Matlab LM Exponents only: 74.08% fitness (better).



(b) Python LM Exponents only: 74.07% fitness.



(a) Matlab LM Coefficients only: 93.97% fitness.



(b) Python LM Coefficients only: 94.48% fitness (better).

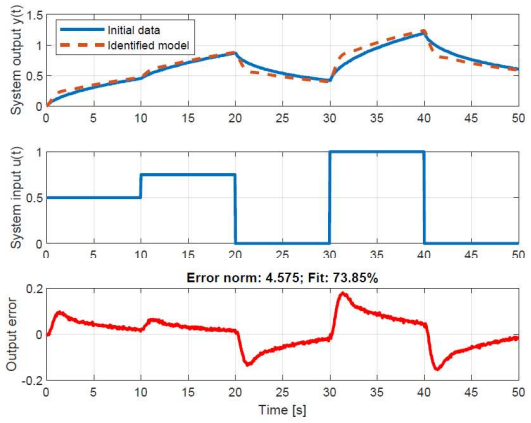
1.1.2 Exponents only

1. Matlab LM Exponents only: 74.08% fitness better.

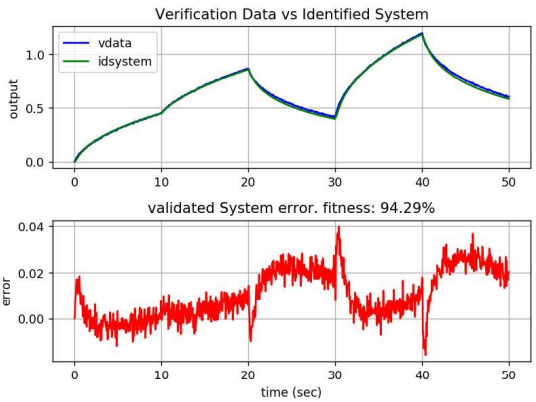
$$\frac{2s^{0.24299} + 4s^{6.2041\times10^{-17}}}{2s^{2.4914} + 3.8s^{0.63126} + 2.5s^{0.63125} + 2.6s^{0.63123} + 1.5}$$

2. Python LM Exponents only: 74.07% fitness.

$$\frac{-2.0s^{0.24309} + 4.0}{2.0s^{2.49852} + 3.8s^{0.63103} + 2.6s^{0.63134} + 2.5s^{0.63131} + 1.5}$$



(a) Matlab TRR Both: 73.85% fitness.



(b) Python TRR Both: 94.29% fitness (better).

1.1.3 Coefficients only

1. Matlab LM Coefficients only: 93.97% fitness.

$$\frac{1.0406s^{0.63} - 0.080038}{-0.024459s^{3.501} + 1.3468s^{2.42} - 0.90976s^{1.798} + 5.0451s^{1.31} - 0.024465}$$

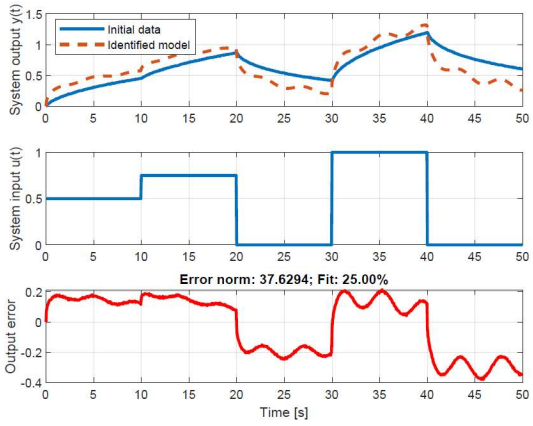
2. Python LM Coefficients only: 94.48% fitness (better).

$$\frac{206.27943s^{0.63} - 16.53132}{-2.95129s^{3.501} + 159.90228s^{2.42} - 107.60438s^{1.798} + 972.42517s^{1.31} - 5.20082}$$

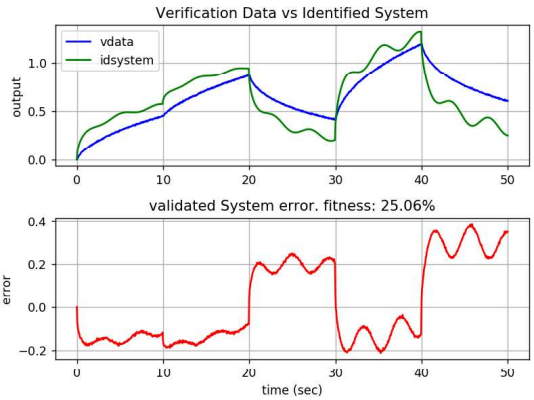
1.2 Identification using Trust Region Reflective method

Initial Guess used for bench-marking purpose:

$$G(s) = \frac{2s^{0.63} + 4}{2s^{3.501} + 3.8s^{2.42} + 2.6s^{1.798} + 2.5s^{1.31} + 1.5}$$



(a) Matlab TRR Exponent only: 25.00% fitness.



(b) Python TRR Exponents only: 25.06% fitness (better).

1.2.1 Both – Coefficients + Exponents

1. Matlab: 73.85% fitness (but does not have a steady state, because the last exponent of s that is, $\alpha_0 \neq \beta_0 \neq 0$).

$$\frac{8.4326s^{1.0176} + 2.2713s^{0.43631}}{7.9863s^{2.6433} + 30.791s^{1.1911} - 2.3643s^{0.5283} + 1.5097s^{0.35477} - 0.027909s^{0.25957}}$$

2. Python: 94.29% fitness (better).

$$\frac{-0.53277s^{0.4587} + 2.58579}{5e-05s^{3.71577} - 0.13814s^{2.40318} + 1.25029s^{1.76607} + 9.66293s^{0.59257} + 0.63314}$$

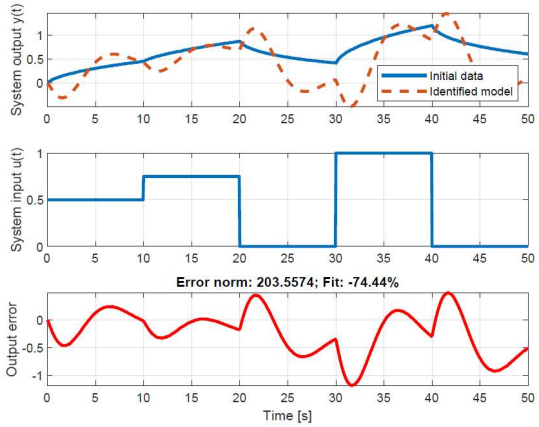
1.2.2 Exponents only

1. Matlab: 25.00% fitness.

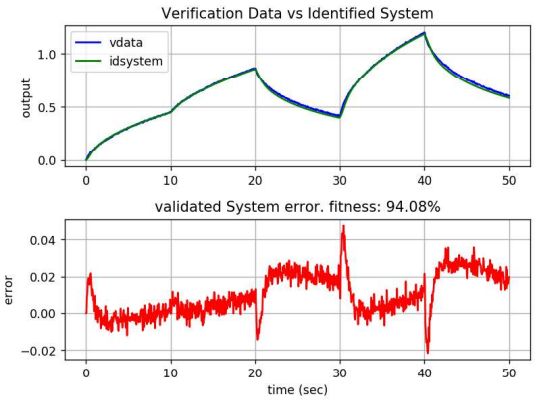
$$\frac{2s^{1.9976} + 4s^{2.4281\times10^{-14}}}{3.8s^{2.4129} + 2.5s^{0.44366} + 2.6s^{0.44362} + 2s^{0.4436} + 1.5s^{9.9998\times10^{-10}}}$$

2. Python: 25.06% fitness (better).

$$\frac{2.0s^{1.99175} + 4.0}{2.0s^{0.44364} + 3.8s^{2.41274} + 2.6s^{0.44364} + 2.5s^{0.44364} + 1.5}$$



(a) MATLAB TRR Coefficients only: -74.44% fitness.



(b) Python TRR Coefficients only: 94.08% fitness (better).

1.2.3 Coefficients only

1. Matlab: -74.4% fitness.

$$\frac{-6.6881s^{0.63}+3.3329}{-0.029477s^{3.501}+1.2118s^{2.42}+3.4482s^{1.798}+3.0363s^{1.31}+3.1005}$$

2. Python: 94.08% fitness (better).

$$\frac{2.70797s^{0.63}-0.21049}{-0.05962s^{3.501}+3.26931s^{2.42}-2.17784s^{1.798}+13.04912s^{1.31}-0.0649}$$

1.3 Remarks

From the comparison, similar results are obtained from Python as compared to MATLAB with variance in some cases possible dues to allowed number of iterations and no bound restriction was applied.