Replication_code_section3

September 20, 2024

1 Replication code for Section 3, Song et al. (2024)

This file replicates the output in Section 3 of

"PyTimeVar: A Python Package for Trending Time-Varying Time Series Models."

Authors: Mingxuan Song, Bernhard van der Sluis, and Yicong Lin.

Date current version: September 2024

Section 3 illustrates our package using the Temperature dataset.

1.0.1 Load temperature data and set seed

[1]:
```
# Load data
from PyTimeVar.datasets import temperature
import numpy as np
data = temperature.load(
    regions=['World'], start_date='1961', end_date='2023')
vY = data.values
mX = np.ones_like(vY)

# set seed
np.random.seed(123)
```

1.0.2 Illustation of local linear regression

[2]:
```
# illustrate LLR
from PyTimeVar import LocalLinear
model = LocalLinear(vY=vY, mX=mX)
betaHatLLR = model.fit()
```

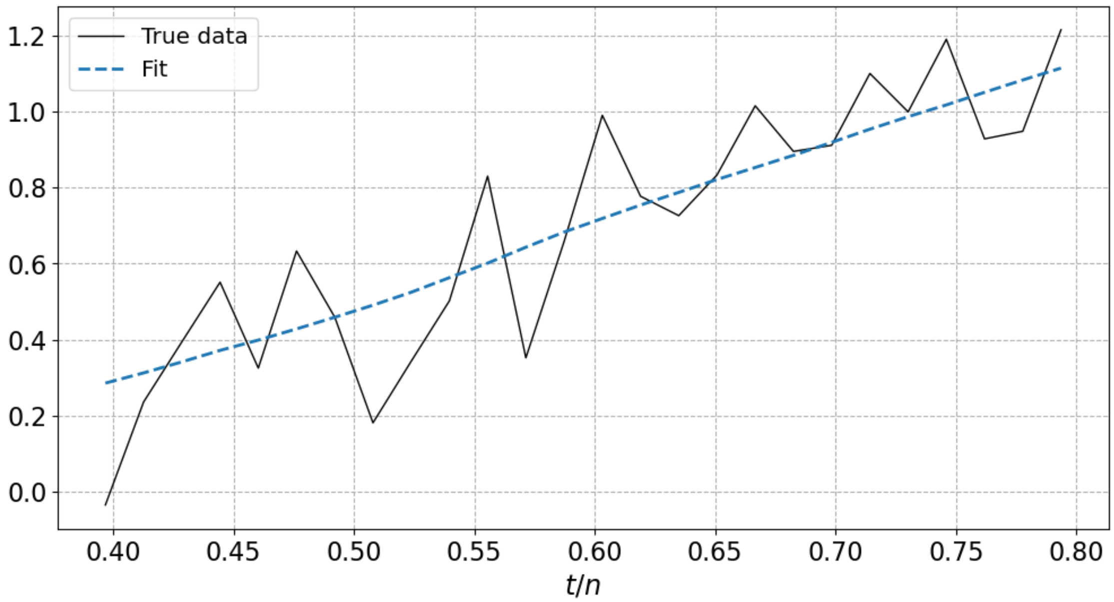
```
No bandwidth or selection method is specified.
Progress LMCV-0:

100%|      | 29/29 [00:00<00:00, 190.59it/s]

Progress LMCV-2:

100%|      | 29/29 [00:00<00:00, 154.26it/s]

Progress LMCV-4:
```

```
100%|      | 29/29 [00:00<00:00, 193.76it/s]

Progress LMCV-6:

100%|      | 29/29 [00:00<00:00, 182.13it/s]


============================================================
Optimal bandwidth selected by individual method:
- AIC method: 0.4286
- GCV method:  0.4286
- LMCV-0 method:  0.0600
- LMCV-2 method:  0.1150
- LMCV-4 method:  0.0600
- LMCV-6 method:  0.0600
============================================================
Optimal bandwidth used is the avg. of all methods:  0.1920
============================================================
Note: (1) For constructing confidence intervals/bands using the residual-based
bootstrap method, the avg. bandwidth of all methods  0.1920 is used; (2) For
constructing confidence intervals/bands using the MB method, the GCV bandwidth
0.4286 is used.
```

[3]:
```python
# print summary
model.summary()
```

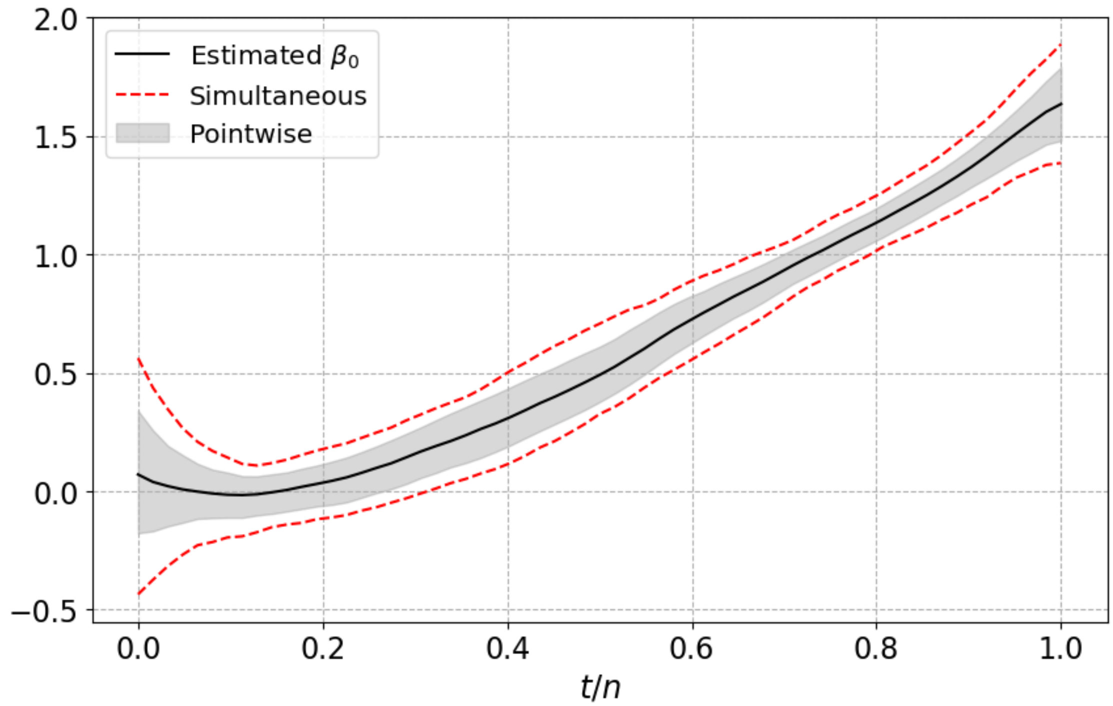
```
Local Linear Regression Results
============================================================
Kernel: epanechnikov
Bandwidth selection method: AVG
Bandwidth used for estimation:  0.1920
Number of observations: 63
Number of predictors: 1
============================================================
Beta coefficients (shape: (1, 63)):
Use the 'plot_betas()' method to plot the beta coefficients.
============================================================
Use the 'confidence_bands()' method to obtain the confidence bands and plots.
You can choose out of 6 types of Bootstrap to construct confidence bands:
SB, WB, SWB, MB, LBWB, AWB
============================================================
Use the 'plot_residuals()' method to plot the residuals.
```

[4]:
```python
# plot trend and data
model.plot_predicted(tau=[0.4,0.8])
```



[5]:
```
# plot confidence bands using LBWB
S_LB, S_UB, P_LB, P_UB = model.confidence_bands(bootstrap_type='LBWB',␣
↪Gsubs=None, plots=True)
```

```
Calculating LBWB Bootstrap Samples
100%|          | 1299/1299 [00:05<00:00, 228.51it/s]
```



1.0.3 Illustration of parameter specifications for local linear regression

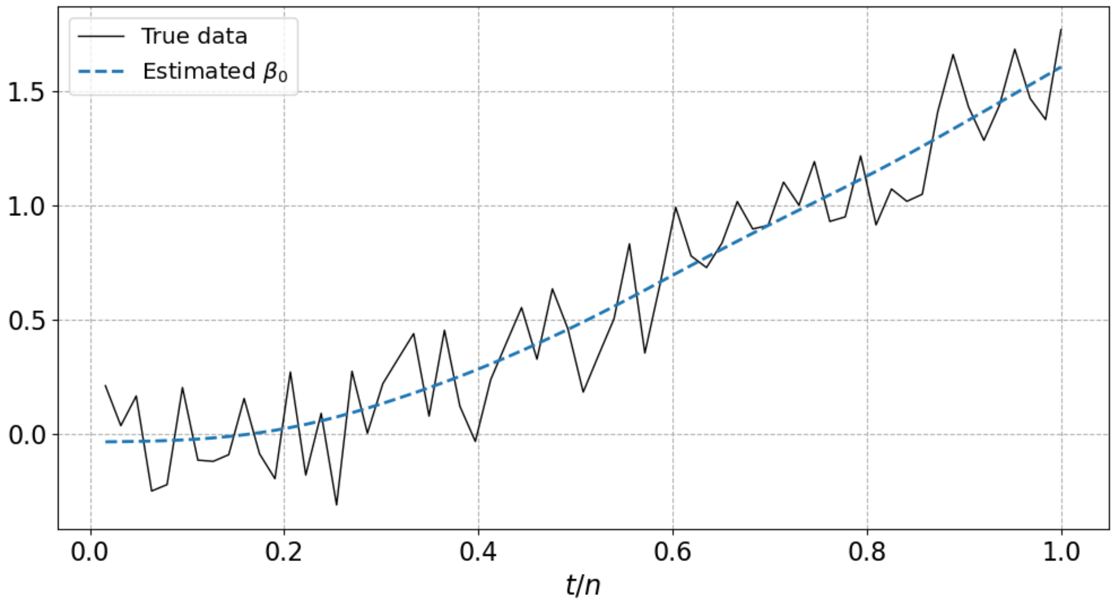
[6]:
```
# auxiliary LLR model to illustrate kernel, bandwidth selection, and tau
model2LLR = LocalLinear(vY=vY, mX=mX, kernel='Gaussian', bw_selection='lmcv_8')
beta_hat_model2 = model2LLR.fit()
```

```
Progress LMCV-8:

100%|     | 29/29 [00:00<00:00, 215.74it/s]



- LMCV-8 method:  0.0750
Optimal bandwidth used is lmcv_8:  0.0750
============================================================
Note: For constructing confidence intervals/bands using the MB method, a GCV
bandwidth is recommended.
```

1.0.4 Illustation of boosted HP filter

[7]:
```
# illustrate boosted HP filter
from PyTimeVar import BoostedHP
bHPmodel = BoostedHP(vY=vY, dLambda=1600, iMaxIter=100)
bHPtrend, bHPresiduals = bHPmodel.fit(
    boost=True, stop="adf", dAlpha=0.05, verbose=False)
bHPmodel.summary()
bHPmodel.plot()
```

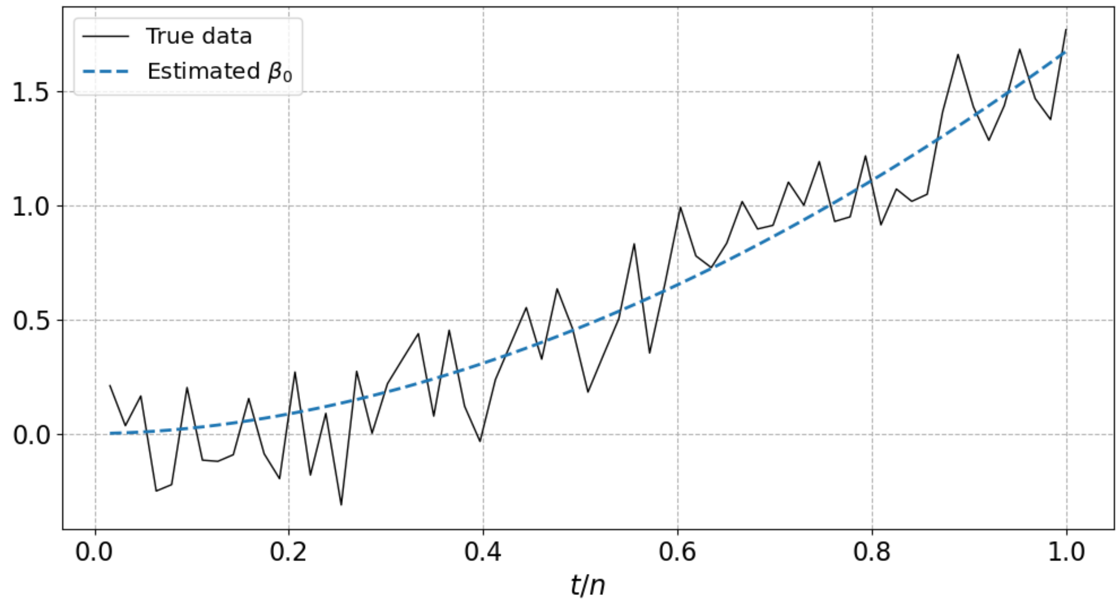
```
Boosted HP Filter Results
============================================================
Stopping Criterion: adf
Max Iterations: 100
Iterations Run: 1
============================================================
Lambda: 1600
Alpha: 0.05
Information Criteria Values: [3.54412124e-12]
============================================================
```



1.0.5 Illustration of power-law model

[8]:
```
# illustrate power-law trend
from PyTimeVar import PowerLaw
PwrLaw = PowerLaw(vY=vY, n_powers=1)
pwrTrend, pwrGamma = PwrLaw.fit()
PwrLaw.summary()
PwrLaw.plot()
```

```
Power-Law Trend Results:
==============================
yhat= 0.001 t^1.848
```



1.0.6 Illustration of parameter specifications for power-law model

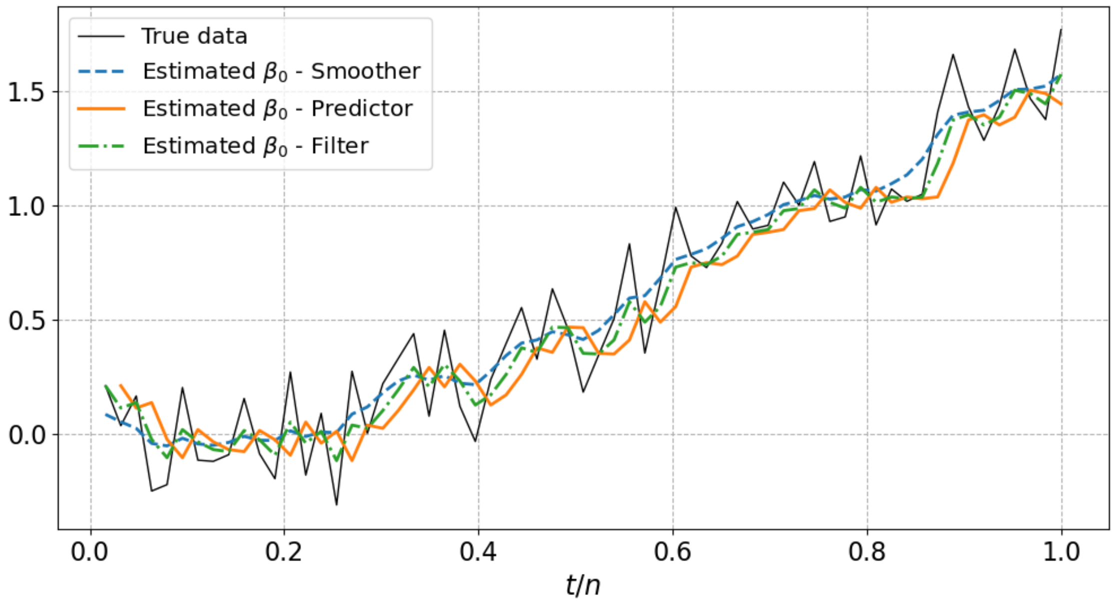
[9]:
```
# auxiliary power-law model to illustrate options
vgamma0 = np.arange(0, 0.1, 0.05)
options = {'maxiter': 1E3, 'disp': False}
bounds = ((0,0),(0.1, 5), )
auxPwr = PowerLaw(vY=vY, n_powers=2, vgamma0=vgamma0, bounds=bounds,␣
↪options=options)
auxPwrTrend, auxPwrGamma = auxPwr.fit()
auxPwr.summary()
```

```
Power-Law Trend Results:
==============================
yhat= -4.586 + 3.745 t
```

1.0.7 Illustration of state-space model

[10]:
```
# illustrate Kalman smoother
from PyTimeVar import Kalman
kalmanmodel = Kalman(vY=vY)
[kl_filter, kl_predictor, kl_smoother] = kalmanmodel.fit('all')
kalmanmodel.plot(individual=False)
```

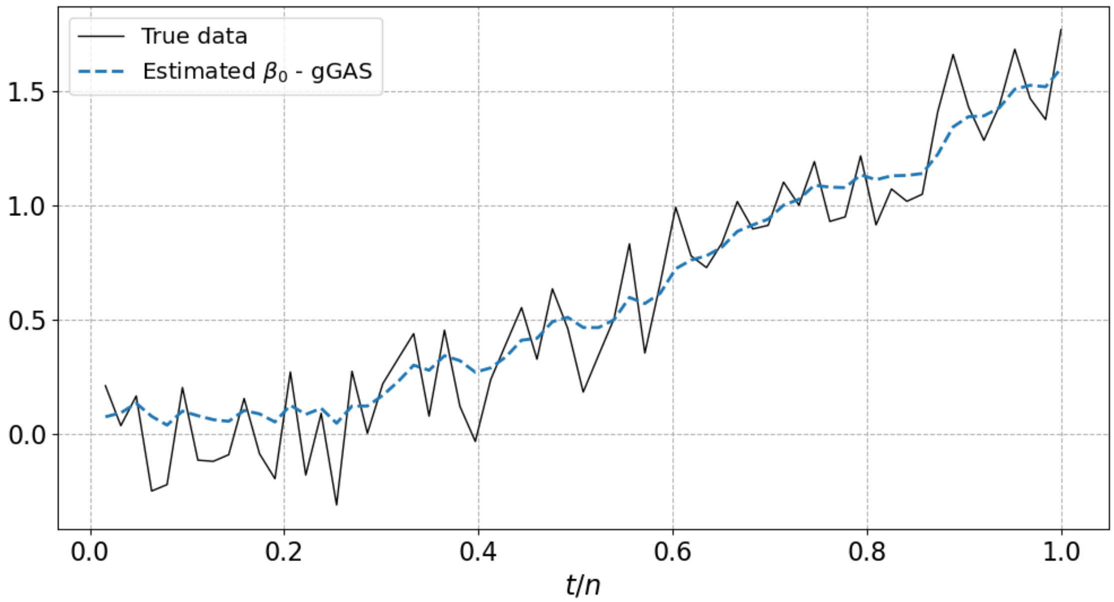
```
Optimization success
```



1.0.8 Illustration of score-driven (GAS) models

[11]:
```
# illustrate GAS model
from PyTimeVar import GAS
N_gasmodel = GAS(vY=vY, mX=mX, method='gaussian', niter=10)
N_GAStrend, N_GASparams = N_gasmodel.fit()
N_gasmodel.plot()
```

```
Time taken: 0.93 seconds
```


True data
Estimated β_0 - gGAS
1.5
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0
t/n