dsb-testing

April 17, 2024

1 Differential spectrum balance - Testing

This notebook is used to test this fitting method using different models and data. Who knowns what we'll find out here!

Tests need to be visualized against original data or other models as well as measured using main set of parameters: - median - variance - standard deviation - linear deviation - covariance - concordance

If need be add some stuff here, but please keep most of the pluming and junk in separate files somewhere we can't see them and just import functions or classes here… Idk why I'm writing a reminder like that to myself, but who knows.

Gloria in excelsis Stella Caesa!

[ ]:

```python
# Main imports
import pandas as pd
import numpy as np

from modules.extra import poly_fit, dsb_fit
from modules.utils import multi_plot, statistics, timed
from modules.models import apply_noise
```

1.1 Exponential model

Yeah, let's use our old pal for testing at first. Since it is an ideal model, it is defined in it's own file and need to be imported. General model parameters: - form: f(x) = 0.0000005 * x**2 - size: 10000 - abn: 3

[ ]:

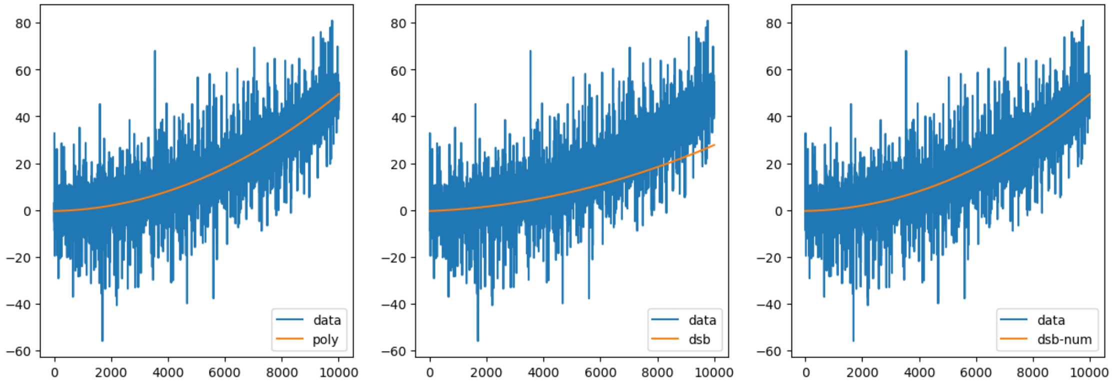
```python
# Model import
from modules.models import exponential

# Create
ideal = exponential()
model = apply_noise(ideal)

# Fitting
fitted_poly = timed(lambda: poly_fit(model, 4), label="poly")
fitted_dsb = timed(
    lambda: dsb_fit("a0 + a1*t + a2*exp(a3*t)", "t", model, numeric=False),␣
  ↪label="dsb"
)
fitted_dsb_numeric = timed(
    lambda: dsb_fit("a0 + a1*t + a2*exp(a3*t)", "t", model), label="dsb_num"
)

# Display
results = (fitted_poly, "poly"), (fitted_dsb, "dsb"), (fitted_dsb_numeric, "dsb-num")
multi_plot(model, *results)
statistics(ideal, *results)
```

```
[poly] took 9.32390ms to complete.
[dsb] took 174.80070ms to complete.
[dsb_num] took 166.79000ms to complete.
```



```
             median    variance  std. div.     lin. div.  covariance  concordance
poly      12.535914  220.267011  14.841395  48622.038185  220.267011     0.900525
dsb        7.787406   69.302228   8.324796  89714.879981  123.489883     0.673328
dsb-num   12.535785  220.265166  14.841333  48621.729686  220.266045     0.900524
data      13.170270  268.527358  16.386804      0.000000  268.527358     1.000000
```

1.2 Real data: USD Exchange

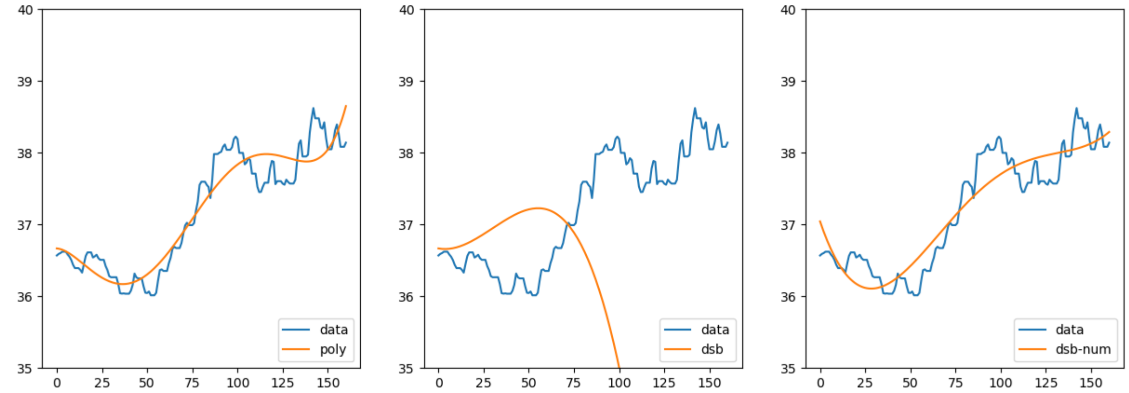
This dataset is fairly small, but it is good enough to test nonlinear stuff, since previous model was just fine even with polynomial. - form: ??? - size: 161

```
[ ]: # Read data
     raw_data = pd.read_excel("data/usd_2023-2024_1.xlsx", "USD")
     usd_data = np.array(raw_data.iloc[0:161, 6].values)

     # Fitting
     fitted_poly = timed(lambda: poly_fit(usd_data, 6), label="poly")
     fitted_dsb = timed(
         lambda: dsb_fit(
             "a0 + a1*t + a2*sin(a3*t) + a4*cos(a5*t)", "t", usd_data, numeric=False
         ),
         label="dsb",
     )
     fitted_dsb_numeric = timed(
         lambda: dsb_fit("a0 + a1*t + a2*sin(a3*t) + a4*cos(a5*t)", "t", usd_data),
         label="dsb_num",
     )

     # Display
     results = (fitted_poly, "poly"), (fitted_dsb, "dsb"), (fitted_dsb_numeric,␣
      ↪"dsb-num")
     multi_plot(usd_data, *results, ylims=(35, 40))
     statistics(usd_data, *results)
```

```
[poly] took 10.31760ms to complete.
[dsb] took 223.60680ms to complete.
[dsb_num] took 312.38210ms to complete.
```



```
          median   variance  std. div.   lin. div.  covariance  concordance
poly     37.260237   0.561493   0.749328   34.292776    0.561493     0.912883
dsb      36.658856  24.555325   4.955333  653.894273   -2.821073    -0.218549
dsb-num  37.267693   0.547409   0.739871   39.678951    0.547419     0.902390
data     37.452100   0.631848   0.794889    0.000000    0.631848     1.000000
```

1.2.1 4 rank stuff

Let’s try the same model but without cosine part, so it should be 4th rank.

commented out to avoid errors

[ ]:
```
# # Fitting
# fitted_poly = timed(lambda: poly_fit(usd_data, 5), label="poly")
# fitted_dsb = timed(
#     lambda: dsb_fit(
#         "a0 + a1*t + a2*sin(a3*t)", "t", usd_data, numeric=False
#     ),
#     label="dsb",
# )
# fitted_dsb_numeric = timed(
#     lambda: dsb_fit("a0 + a1*t + a2*sin(a3*t)", "t", usd_data),
#     label="dsb_num",
# )

# # Display
# results = (fitted_poly, "poly"), (fitted_dsb, "dsb"), (fitted_dsb_numeric, "dsb-num")
# multi_plot(usd_data, *results, ylims=(35, 40))
# statistics(usd_data, *results)
```

Aaand it doesn't really work, cause DSB has a weakness - it the system of nonlinear equations is unsolvable when one or more of the series partitions are nullified.

Here is where it happens:

[ ]:
```
def failing_model_test():
    # Eah, i didn't want to import this, yet here we are
    import sympy as sp
    t, a0, a1, a2, a3 = sp.symbols("t, a0, a1, a2, a3")
    # Model definition
    failing_model = a0 + a1*t + a2 *sp.sin(a3*t)
    display("Failing model:", failing_model)
    # Series expansion
    series = sp.series(failing_model, t).removeO()
    display("Series expansion:", series)

failing_model_test()
```

```
'Failing model:'
```

$$a_0 + a_1 t + a_2 \sin(a_3 t)$$

```
'Series expansion:'
```

$$a_0 + \frac{a_2 a_3^5 t^5}{120} - \frac{a_2 a_3^3 t^3}{6} + t\left(a_1 + a_2 a_3\right)$$

No coeffs for t^2 and for t^4, as one can see, so no DSB here.

1.2.2 Extrapolation

Let's try extra for this model since i have future data lying around.

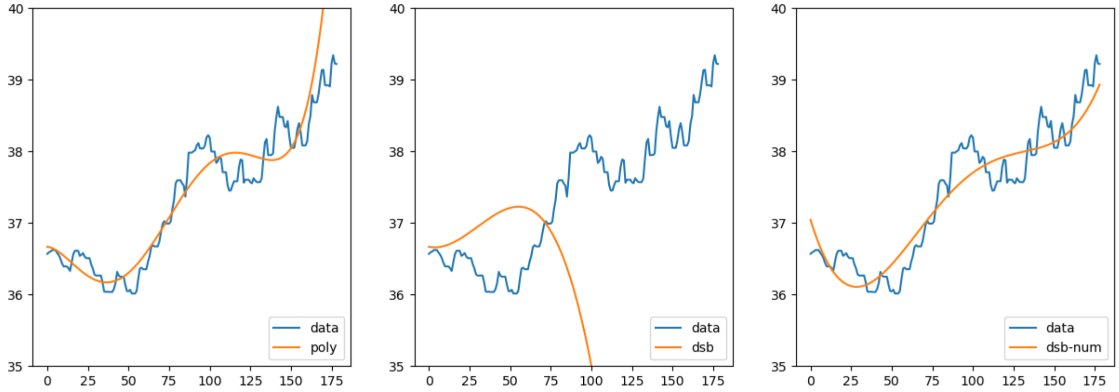
[ ]:
```
# Read extra data
raw_data_extra = pd.read_excel("data/usd_2023-2024_2.xlsx", "USD")
```

```
usd_data_extra = np.array(raw_data_extra.iloc[0:179, 6].values)
extra = usd_data_extra.size


# Fitting
fitted_poly = timed(lambda: poly_fit(usd_data, 6, length=extra), label="poly")
fitted_dsb = timed(
    lambda: dsb_fit(
        "a0 + a1*t + a2*sin(a3*t) + a4*cos(a5*t)",
        "t",
        usd_data,
        numeric=False,
        length=extra,
    ),
    label="dsb",
)
fitted_dsb_numeric = timed(
    lambda: dsb_fit(
        "a0 + a1*t + a2*sin(a3*t) + a4*cos(a5*t)", "t", usd_data, length=extra
    ),
    label="dsb_num",
)

# Display
results = (fitted_poly, "poly"), (fitted_dsb, "dsb"), (fitted_dsb_numeric,␣
 ↪"dsb-num")
multi_plot(usd_data_extra, *results, ylims=(35, 40))
statistics(usd_data_extra, *results)
```

```
[poly] took 4.22110ms to complete.
[dsb] took 207.05650ms to complete.
[dsb_num] took 308.72950ms to complete.
```



	median	variance	std. div.	lin. div.	covariance	concordance

```
poly     37.560950   1.367871   1.169560    55.644263    0.979259     0.885912
dsb      36.086632  57.351572   7.573082  1100.181898   -5.639062    -0.186742
dsb-num  37.488539   0.670034   0.818556    45.671735    0.712453     0.937794
data     37.569800   0.842787   0.918034     0.000000    0.842787     1.000000
```

1.3 Transcendental model

This is another one of the ideal models, so needs to be imported. Uses trigonometry, so may be more difficult for DSB, let's see.

Model parameters: - form: 0.2*sin(0.005*t) + 2.12*cos(0.005*t) - size: 10000 - sigma: 1

[ ]:

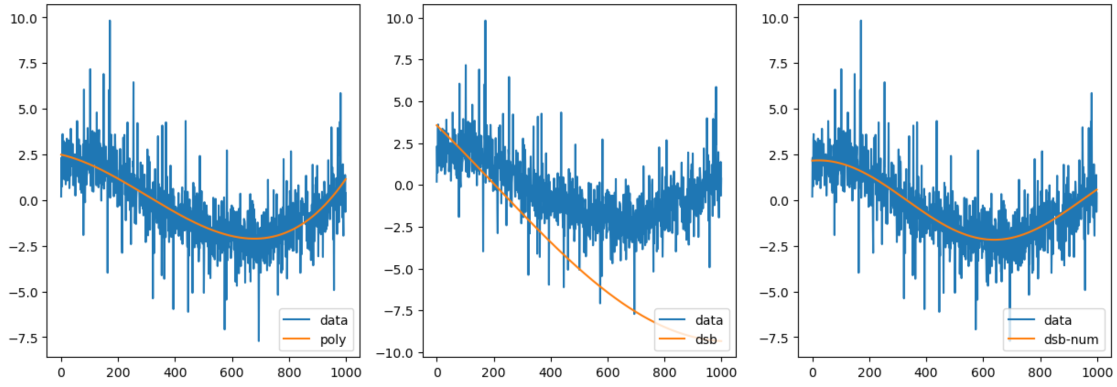
```
# Import model
from modules.models import transcendental1

ideal = transcendental1()
model = apply_noise(ideal, sigma=1)

# Fitting
fitted_poly = timed(lambda: poly_fit(model, 4), label="poly")
fitted_dsb = timed(
    lambda: dsb_fit("a0*sin(a2*t) + a3*cos(a2*t)", "t", model, numeric=False),␣
  ↪label="dsb"
)
fitted_dsb_numeric = timed(
    lambda: dsb_fit("a0*sin(a2*t) + a3*cos(a2*t)", "t", model), label="dsb_num"
)

# Display
results = (fitted_poly, "poly"), (fitted_dsb, "dsb"), (fitted_dsb_numeric,␣
  ↪"dsb-num")
multi_plot(model, *results)
statistics(ideal, *results)
```

```
[poly] took 7.04380ms to complete.
[dsb] took 102.03930ms to complete.
[dsb_num] took 80.42050ms to complete.
```



```
           median   variance  std. div.    lin. div.  covariance  concordance
poly    -0.664581   2.139559   1.462723   954.005818    2.139559     0.702619
dsb     -5.010142  16.252462   4.031434  4178.393884    4.716586     0.233876
dsb-num -0.634037   2.165476   1.471556   942.316651    2.166530     0.709370
data    -0.521095   3.930089   1.982445     0.000000    3.930089     1.000000
```

Looks fine here, let's try another form. - form: 21.5*sin(0.02*t) - 12.3*cos(0.02*t)

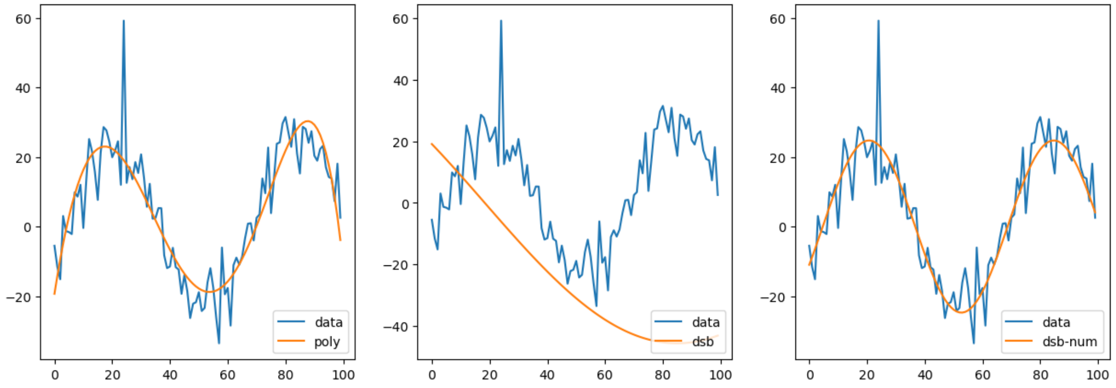
```
[ ]: from modules.models import transcendental2

     ideal = transcendental2()
     model = apply_noise(ideal)

     # Fitting
     fitted_poly = timed(lambda: poly_fit(model, 6), label="poly")
     fitted_dsb = timed(
         lambda: dsb_fit("a0*sin(a2*t) + a3*cos(a2*t)", "t", model, numeric=False),␣
       ↪label="dsb"
     )
     fitted_dsb_numeric = timed(
         lambda: dsb_fit("a0*sin(a2*t) + a3*cos(a2*t)", "t", model, maxfev=1000),␣
       ↪label="dsb_num"
     )

     # Display
     results = (fitted_poly, "poly"), (fitted_dsb, "dsb"), (fitted_dsb_numeric,␣
       ↪"dsb-num")
     multi_plot(model, *results)
     statistics(ideal, *results)
```

```
[poly] took 10.49230ms to complete.
[dsb] took 101.91430ms to complete.
[dsb_num] took 81.47730ms to complete.
```



```
            median    variance  std. div.    lin. div.  covariance  concordance
poly      6.911345  253.756893  15.929749   541.377836  253.756893     0.898307
dsb     -30.658375  434.786899  20.851544  3278.218432   11.673271     0.010613
dsb-num   8.343063  274.819497  16.577681   470.747052  270.972773     0.924089
data      7.475274  310.891928  17.632128     0.000000  310.891928     1.000000
```

Hm, need to trim maxfev to prevent overfitting, otherwise looks fine. If dataset captures more than a couple of function's periods, than numeric part of the fitting fails, so need to be careful.

1.3.1 5 rank stuff

Let's try higher rank again, even tho it didn't work out previously. - form: -7.22*cos(0.015*t) - 0.54*sin(0.015*t) + 1.9*cos(0.015*t) - size: 500

It sure didn't work here as well

[ ]:

```
# from modules.models import transcendental3

# ideal = transcendental3()
# model = apply_noise(ideal)

# # Fitting
# fitted_poly = timed(lambda: poly_fit(model, 6), label="poly")
# fitted_dsb = timed(
#     lambda: dsb_fit("a0*cos(a1*t) + a2*sin(a1*t) + a3*cos(a1*t)", "t", model,␣
 ↪numeric=False, rank=5), label="dsb"
# )
# fitted_dsb_numeric = timed(
#     lambda: dsb_fit("a0*cos(a1*t) + a2*sin(a1*t) + a3*cos(a1*t)", "t", model,␣
 ↪maxfev=1000, rank=5), label="dsb_num"
# )

# # Display
# results = (fitted_poly, "poly"), (fitted_dsb, "dsb"), (fitted_dsb_numeric,␣
 ↪"dsb-num")
# multi_plot(model, *results)
# statistics(ideal, *results)
```

The reason here is there is no solutions for the balance even for different ranks (I've tried 4th, 5th and 6th, which is max). What to do here I have no idea tbh.