

Національний Технічний Університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-Науковий Комплекс
«Інститут прикладного системного аналізу»

Лабораторна робота № 3
з дисципліни «Моделювання складних систем»

Виконали:
студенти гр. КА-41
Мельничук Валентин
Лочман Ярослава
Снігірьова Валерія

Прийняв:
професор кафедри ММСА,
д.т.н. Степашко В.С.

Київ 2018

Зміст

1	Пошук моделі оптимальної складності	2
1.1	Задання конфігурацій. Створення вибірок	2
1.1.1	Тест 1 (вибірка аналогічна вибірці з лабораторної роботи №2)	2
1.1.2	Тест 2 (велика вибірка з великою кількістю регресорів)	2
1.1.3	Тест 3 (власні задачі)	3
1.2	Пошук моделі. Послідовне включення (метод з лабораторної роботи №2)	5
1.2.1	Тест 1	5
1.2.2	Тест 2	7
1.2.3	Тест 3	9
1.3	Пошук моделі. Метод кореляційного включення	11
1.3.1	Тест 1	11
1.3.2	Тест 2	12
1.3.3	Тест 3	14
1.4	Пошук моделі. Метод випадкового включення	15
1.4.1	Тест 1	15
1.4.2	Тест 2	17
1.4.3	Тест 3	19
1.5	Пошук моделі. Метод перебірного включення	23
1.5.1	Тест 1	23
1.5.2	Тест 2	24
1.5.3	Тест 3	27
2	Висновки	31
3	Код програми	32
3.1	Імпорт необхідних бібліотек, налаштування	32
3.2	Реалізація МНКО	32
3.3	Реалізація класу моделювання і усіх необхідних методів	34

1 Пошук моделі оптимальної складності

Метою роботи є пошук моделі оптимальної складності, причому наперед не відомо, яку саме підмножину всіх регресорів/аргументів містить така модель. Тому необхідно сконструювати і перевірити алгоритми для її пошуку.

1.1 Задання конфігурацій. Створення вибірок

1.1.1 Тест 1 (вибірка аналогічна вибірці з лабораторної роботи №2)

```
In [240]: config1 = ModelConfig(a=0, b=0.5)
          config1.compile(n=1000)
          config1.generate_noise_and_output()
          config1.show()
```

Regressors: m = 5

True parameters: $\theta = [3 \ -2 \ 1 \ 0 \ 0]$

$y_0 = (3) * x_1 + (-2) * x_2 + (1) * x_3 + (0) * x_4 + (0) * x_5$

Noise generation: $\sigma = 0.3$

Sample length: n = 1000

X[:10]:

```
[0.41874236 0.12456156 0.28066172 0.15910143 0.46122701]
[0.18105384 0.37716676 0.27709214 0.4888375  0.44187549]
[0.39936274 0.38647222 0.15799907 0.23559202 0.17128673]
[0.37188136 0.0734651  0.4625002  0.4584715  0.46049963]
[0.16993313 0.00577501 0.10517818 0.41488992 0.02465976]
[0.4015307  0.10460762 0.01333114 0.121193  0.45949378]
[0.04337896 0.04571534 0.09617543 0.36648325 0.33861697]
[0.43594373 0.13017585 0.14422878 0.09106459 0.30376701]
[0.08004885 0.45564509 0.42793286 0.25046363 0.11175226]
[0.24495675 0.28230842 0.13494233 0.00910952 0.08285733]
```

y[:10]:

```
[ 1.54708203  0.37647857  0.09577151  1.52671753  0.3847499  0.94908614
 0.05493975  0.96470431 -0.44266497 -0.01847096]
```

1.1.2 Тест 2 (велика вибірка з великою кількістю регресорів)

```
In [275]: config2 = ModelConfig(m=15, s0=10, a=0, b=0.5, theta={'random': [1, 10]})
          config2.compile(n=1000)
          config2.generate_noise_and_output()
          config2.show(5)
```

Regressors: m = 15

True parameters: $\theta = [1.08631304 \ 7.56699925 \ 9.87829854 \ 1.23698103 \ 1.95130363 \ 7.51601044$
 $3.30060728 \ 9.37353548 \ 8.89118059 \ 5.43780306 \ 0. \quad 0.$
 $0. \quad 0. \quad 0. \quad]$

$y_0 = (1.0863130375942172) * x_1 + (7.5669992531306685) * x_2 + (9.878298535051083) * x_3 +$
 $(1.2369810327923685) * x_4 + (1.9513036347108885) * x_5 + (7.516010436428849) * x_6 +$

$(3.3006072776149766) * x_7 + (9.373535481004593) * x_8 + (8.891180588599674) * x_9 +$
 $(5.437803059763569) * x_{10} + (0.0) * x_{11} + (0.0) * x_{12} + (0.0) * x_{13} + (0.0) * x_{14} + (0.0) * x_{15}$

Noise generation: $\sigma = 0.3$

Sample length: $n = 1000$

X[:10]:

```

[[0.1705477  0.11570771 0.04892584 0.13072365 0.16437218 0.31504024
  0.14840706 0.11924657 0.09960202 0.22359233 0.43944381 0.23732165
  0.19519143 0.10417676 0.37984204]
[0.0331949  0.40150537 0.25136752 0.44019061 0.45945587 0.21892163
  0.1898278  0.05924368 0.29950877 0.17212511 0.13291328 0.27488848
  0.40227747 0.21418697 0.207846  ]
[0.48098721 0.09015798 0.1225068  0.33779223 0.17132909 0.29822888
  0.29883653 0.19555494 0.17036849 0.38940055 0.25217936 0.22981511
  0.11896126 0.4501579  0.22419701]
[0.40599318 0.38085013 0.26163438 0.35836486 0.24301073 0.25737834
  0.21272382 0.40589793 0.25593534 0.29699694 0.07434101 0.32349888
  0.30363923 0.37264665 0.36211302]
[0.06235834 0.23418489 0.43466777 0.34663551 0.10996189 0.19710446
  0.30886481 0.09920457 0.13226504 0.43110287 0.08626634 0.31482844
  0.33328672 0.11005692 0.07750284]]

```

y[:10]:

```
[ 8.30431374 13.50238474 12.2071761  17.13535375 13.8499516 ]
```

1.1.3 Тест 3 (власні задачі)

Вміст октану При дослідженні виробничого процесу нафтопереробному заводі вимірюється вміст октану в нафті залежно від кількості 3-х сировинних матеріалів та змінної, що характеризує умови виробництва. Вхідні змінні (5):

- Кількість матеріалу 1
- Кількість матеріалу 2
- Кількість матеріалу 3
- Кількісний показник умов виробництва
- Одиниця, відповідає вільному члену

Вихідна змінна:

- Вміст октану

```

In [242]: dataframe_path = 'data/octane-rating.csv'
df = pd.read_csv(dataframe_path)
config3 = ModelConfig(theta='unknown',
                       X=df.loc[:, 'One':'Condition'],
                       y=df['Octane number'])

config3.show()

```

Regressors: $m = 5$

True parameters: $\theta = \text{unknown}$

Sample length: n = 82

X[:10]:

```
[[ 1.      55.33      1.72      54.      1.66219]
 [ 1.      59.13      1.2       53.      1.58399]
 [ 1.      57.39      1.42      55.      1.61731]
 [ 1.      56.43      1.78      55.      1.66228]
 [ 1.      55.98      1.58      54.      1.63195]
 [ 1.      56.16      2.12      56.      1.68034]
 [ 1.      54.85      1.17      54.      1.58206]
 [ 1.      52.83      1.5       58.      1.54998]
 [ 1.      54.52      0.87      57.      1.5623 ]
 [ 1.      54.12      0.88      57.      1.57818]]
```

y[:10]:

```
[92.19 92.74 91.88 92.8  92.56 92.61 92.33 92.22 91.56 92.17]
```

Ціни на нерухомість Вхідні змінні (36):

- Тип житла (номер)
- Дистанція від вулиці до фасаду в футах
- Розмір ділянки в квадратних футах
- Загальна якість матеріалів та обробки
- Загальний стан житла
- Рік початку будівництва
- Рік переробки
- Площа фанери в квадратних футах
- Площа підвалу 1 в квадратних футах
- Площа підвалу 2 в квадратних футах
- Площа підвалу (незробленого) в квадратних футах
- Загальна площа підвалу в квадратних футах
- Перший поверх, площа в квадратних футах
- Другий поверх, площа в квадратних футах
- Інші ділянки, площа в квадратних футах
- Житлова площа квадратних футів
- Підвал: кількість великих ванних кімнат
- Підвал: кількість маленьких ванних кімнат
- Вище рівня фундаменту: кількість великих ванних кімнат
- Вище рівня фундаменту: кількість маленьких ванних кімнат
- Кількість спалень вище рівня фундаменту
- Кількість кухонь вище рівня фундаменту
- Загальна кількість кімнат вище рівня фундаменту (не включає ванні кімнати)
- Кількість камінів
- Рік побудови гаража
- Розмір гаража відносно автомобіля
- Розмір гаража в квадратних футах
- Площа деревини в квадратних метрах
- площа відкритого веранди в квадратних футах
- Закрита площа під'їзду в квадратних футах

- три сезонних ганок площі в квадратних метрах
- площа екранної ганку в квадратних футах
- Площа басейну в квадратних футах
- Змішаний показник
- Місяць продажу (номер)
- Рік продажу

Вихідна змінна:

- Ціна на нерухомість в долларах

```
In [243]: dataframe_path = 'data/house-prices-cleaned.csv'
df = pd.read_csv(dataframe_path)
config3 = ModelConfig(theta='unknown',
                      X=df.loc[:, 'MSSubClass':'YrSold'],
                      y=df['SalePrice'])

config3.show(1)
```

Regressors: m = 36

True parameters: θ = unknown

Sample length: n = 1460

X[:10]:

```
[[4.11087386e+00 4.18965474e+00 9.04204006e+00 7.00000000e+00
 5.00000000e+00 2.00300000e+03 2.00300000e+03 5.28320373e+00
 6.56103067e+00 0.00000000e+00 5.01727984e+00 6.75343792e+00
 6.75343792e+00 6.75110147e+00 0.00000000e+00 7.44483327e+00
 1.00000000e+00 0.00000000e+00 2.00000000e+00 1.00000000e+00
 3.00000000e+00 6.93147181e-01 8.00000000e+00 0.00000000e+00
 2.00300000e+03 2.00000000e+00 5.48000000e+02 0.00000000e+00
 4.12713439e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 2.00000000e+00 2.00800000e+03]]
```

y[:10]:

```
[12.24769912]
```

Далі в якості Теста 3 розглядатимемо саме ціни на нерухомість, оскільки ця задача є цікавішою. При захисті роботи продемонструємо дослідження також і для першої вибірки.

1.2 Пошук моделі. Послідовне включення (метод з лабораторної роботи №2)

1.2.1 Тест 1

```
In [244]: df1 = config1.run_single_LSBM_model_selection(p=None,
                                                    criteria=['Cp', 'FPE', 'RSS'],
                                                    plot=True)

print()
df2 = config1.run_single_LSBM_model_selection(p='reverse',
                                                    criteria=['Cp', 'FPE', 'RSS'],
                                                    plot=True)
```

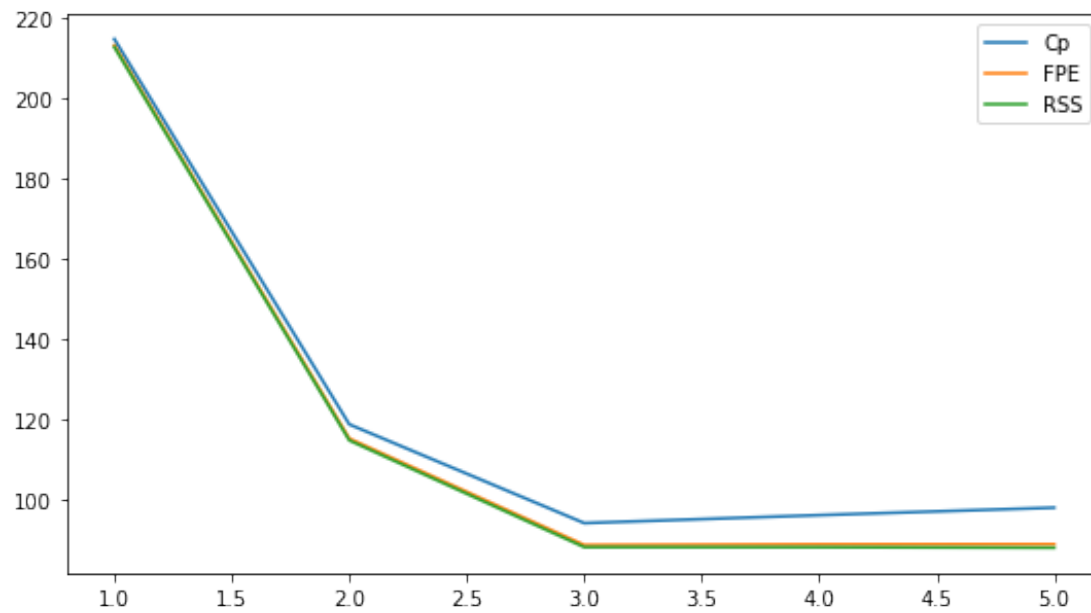
=====

DIRECT

=====

Regressors order:

[1 2 3 4 5]



Optimal:

s* = 3

regressors = [1, 2, 3]

theta* = [2.93417679 -1.95807657 0.96119887 0. 0.]

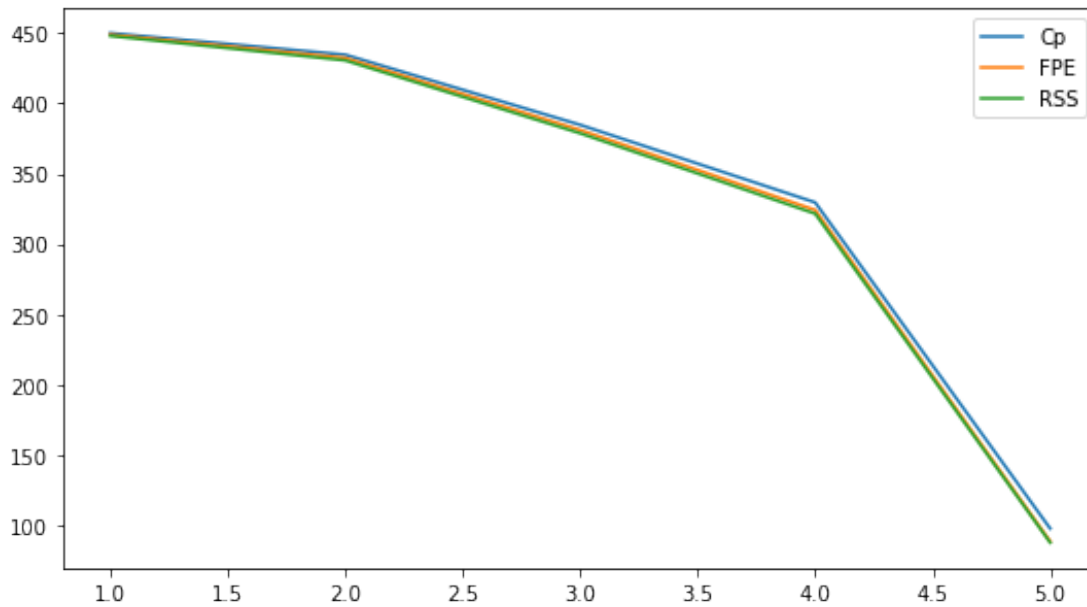
=====

REVERSE

=====

Regressors order:

[5 4 3 2 1]



Optimal:

```
s* = 5
regressors = [1, 2, 3, 4, 5]
theta* = [ 2.92528656 -1.96404659  0.94802332 -0.04141331  0.07425203]
```

Як видно, при прямому порядку критерій C_p має мінімум в точці $s = 3$. А при зворотньому - в точці $s = 5$. Алгоритм МНКО діє послідовно в порядку розташування регресорів, а в другому випадку найбільш впливові регресори були в як раз кінці. Тому перші, ті що не впливові, залишаються за критерієм, хоча, насправді, їх треба вилучити.

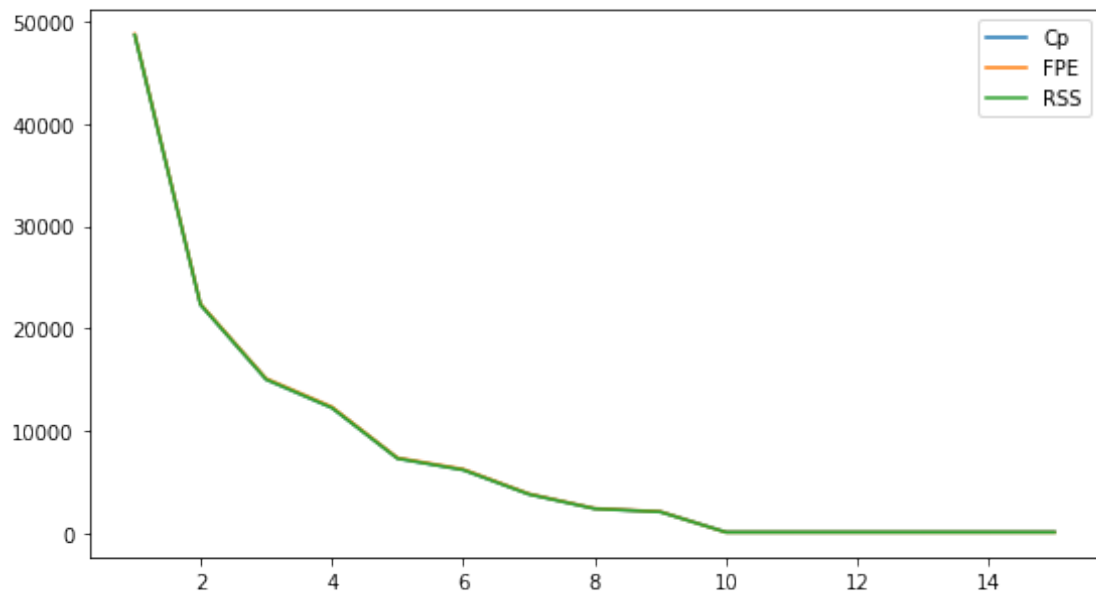
1.2.2 Тест 2

```
In [264]: df1 = config2.run_single_LSMB_model_selection(p=None,
                                                    criteria=['Cp', 'FPE', 'RSS'],
                                                    plot=True)

print()
df2 = config2.run_single_LSMB_model_selection(p='reverse',
                                                    criteria=['Cp', 'FPE', 'RSS'],
                                                    plot=True)
```

```
=====
DIRECT
=====
```

```
Regressors order:
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15]
```

Optimal:

```
s* = 10
regressors = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
theta* = [5.34957983 7.68519352 3.23045789 1.43320723 8.99113426 4.04264347
8.02605224 6.64152218 2.25299834 9.43014176 0. 0.
0. 0. 0. ]
```

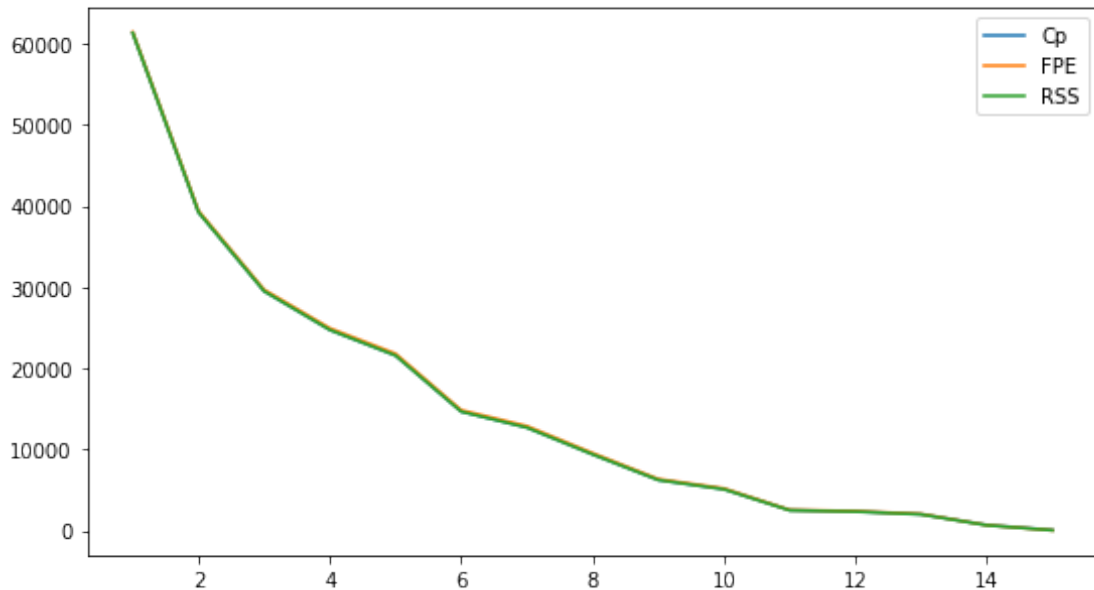
=====

REVERSE

=====

Regressors order:

```
[15 14 13 12 11 10 9 8 7 6 5 4 3 2 1]
```



Optimal:

```
s* = 15
regressors = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
theta* = [ 5.35884827  7.69388952  3.22576623  1.43850351  8.99698342  4.05267537
 8.03636612  6.64606609  2.26220489  9.4353677  0.0189138  -0.07486503
-0.03273382  0.05130706 -0.02752136]
```

Ситуація дуже схожа на Тест 1: при прямому порядку критерій C_p має мінімум в точці $s = 10$. А при зворотньому - в точці $s = 15$. Тут (в другому випадку) також найбільш впливові регресори в кінці, а не впливові на початку, і алгоритм пропонує враховувати всі регресори, що є не дуже добре.

1.2.3 Тест 3

```
In [256]: df1 = config3.run_single_LSMB_model_selection(p=None,
               criteria=['Cp', 'FPE', 'RSS'],
               plot=True)

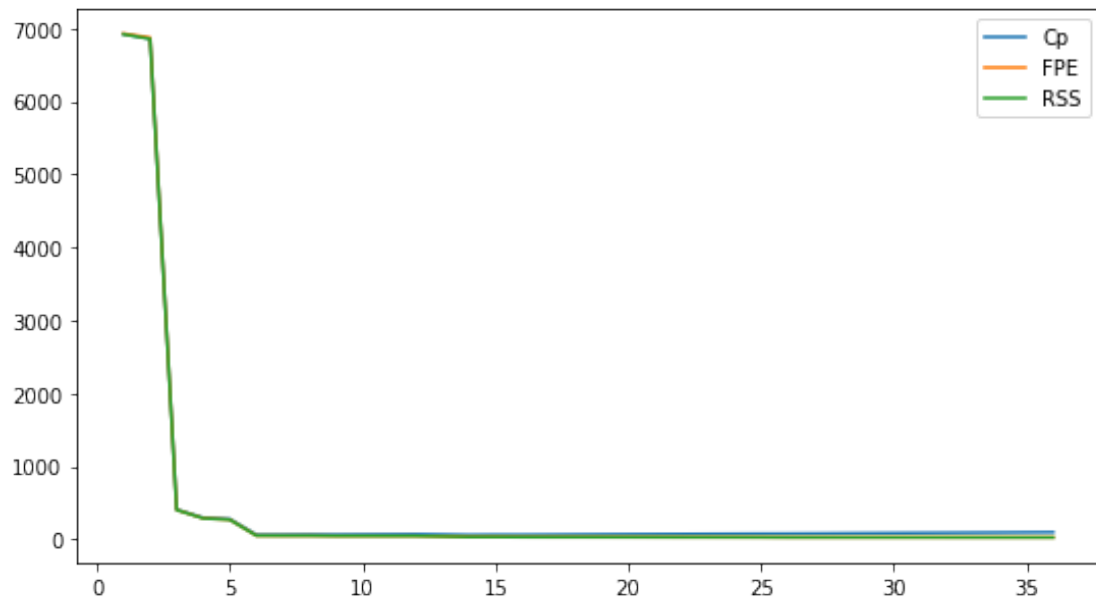
print()
df2 = config3.run_single_LSMB_model_selection(p='reverse',
               criteria=['Cp', 'FPE', 'RSS'],
               plot=True)
```

```
=====
DIRECT
=====
```

Regressors order:

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24]
```

25 26 27 28 29 30 31 32 33 34 35 36]



Optimal:

```
s* = 14
regressors = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
theta* = [-1.28216774e-02  1.35851253e-05  1.01151515e-01  1.25308956e-01
4.12106912e-02  3.12536597e-03  3.92878638e-04 -4.94563644e-06
1.29231644e-02 -2.45073557e-03 -3.51982786e-03  1.69078657e-02
4.24867319e-01  3.14435771e-02  0.00000000e+00  0.00000000e+00
0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
```

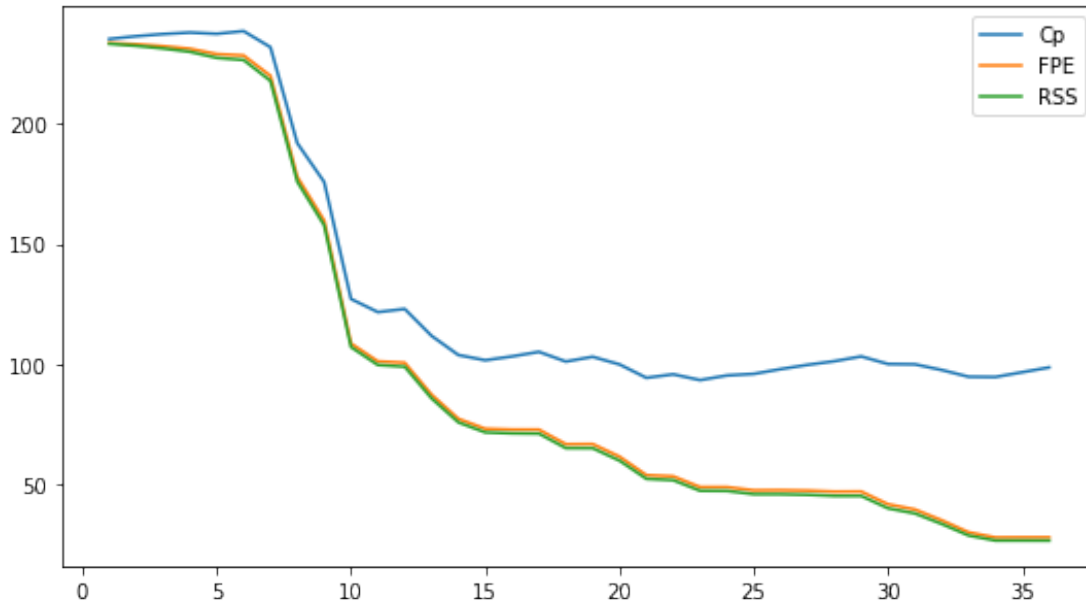
=====

REVERSE

=====

Regressors order:

```
[36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13
12 11 10 9 8 7 6 5 4 3 2 1]
```



Optimal:

```
s* = 23
regressors = [14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32]
theta* = [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 0.00000000e+00 -2.61098624e-02 -2.95063243e-02  6.48993849e-01
 9.19192746e-02  6.26520803e-02  1.20066421e-01  6.97063260e-02
-4.17975867e-02 -7.73319356e-01  1.55498563e-02  3.57490733e-02
-5.79498604e-07  9.78581656e-02  9.61967394e-05  9.14427971e-03
 1.01267236e-02 -8.23456848e-03  7.53719402e-03  7.69962763e-03
-1.80806774e-02 -4.87035069e-03  1.07276179e-03  3.69006574e-03]
```

При прямому порядку критерій C_p має мінімум в точці $s = 14$. А при зворотньому - в точці $s = 23$. Виходить, результати алгоритмів перетинаються тільки в одному регресорі - №14. Це повинно бути спричиненим тим, що впливові регресори знаходяться як на початку, так і в середині, так і в кінці; а неінформативні розташовані серед них довільно.

1.3 Пошук моделі. Метод кореляційного включення

1.3.1 Тест 1

```
In [257]: print('True parameters: {}'.format(config1.theta))
          df1 = config1.run_single_LSMB_model_selection(p='correlation',
              criteria=['Cp', 'FPE', 'RSS'],
              plot=True)
```

```
True parameters: [ 3 -2  1  0  0]
```

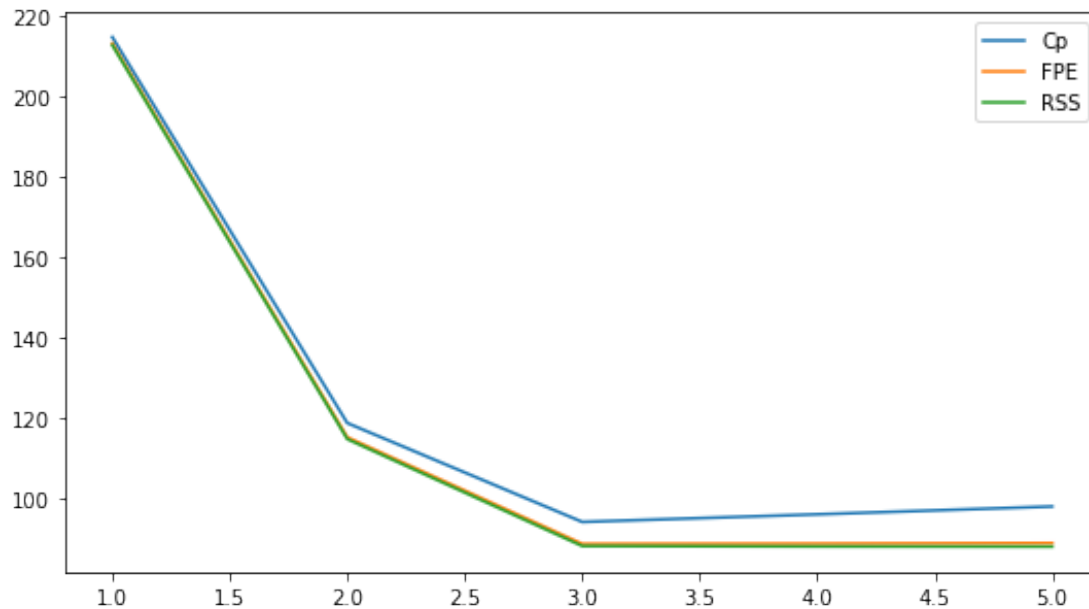
```
=====
CORRELATION INCLUDING
=====
```

```
Correlations with target:
```

```
[0.06608656 0.04732197 0.0214448  0.00308982 0.00320026]
```

```
Regressors order:
```

```
[1 2 3 5 4]
```



```
Optimal:
```

```
s* = 3
```

```
regressors = [1, 2, 3]
```

```
theta* = [ 2.93417679 -1.95807657  0.96119887  0.          0.          ]
```

Як видно, кореляції з вихідною величиною останніх двох регресорів на порядок менші за перших трьох, тому вони знаходяться в кінці, і критерій має мінімум в точці $s = 3$, що означає - не враховувати ці два останні регресори, а це є якраз оптимально.

1.3.2 Тест 2

```
In [277]: print('True parameters: {}'.format(config2.theta))
          df1 = config2.run_single_LSMB_model_selection(p='correlation',
                                                         criteria=['Cp', 'FPE', 'RSS'],
                                                         plot=True)
```

True parameters: [1.08631304 7.56699925 9.87829854 1.23698103 1.95130363 7.51601044
3.30060728 9.37353548 8.89118059 5.43780306 0. 0.
0. 0. 0.]

=====

CORRELATION INCLUDING

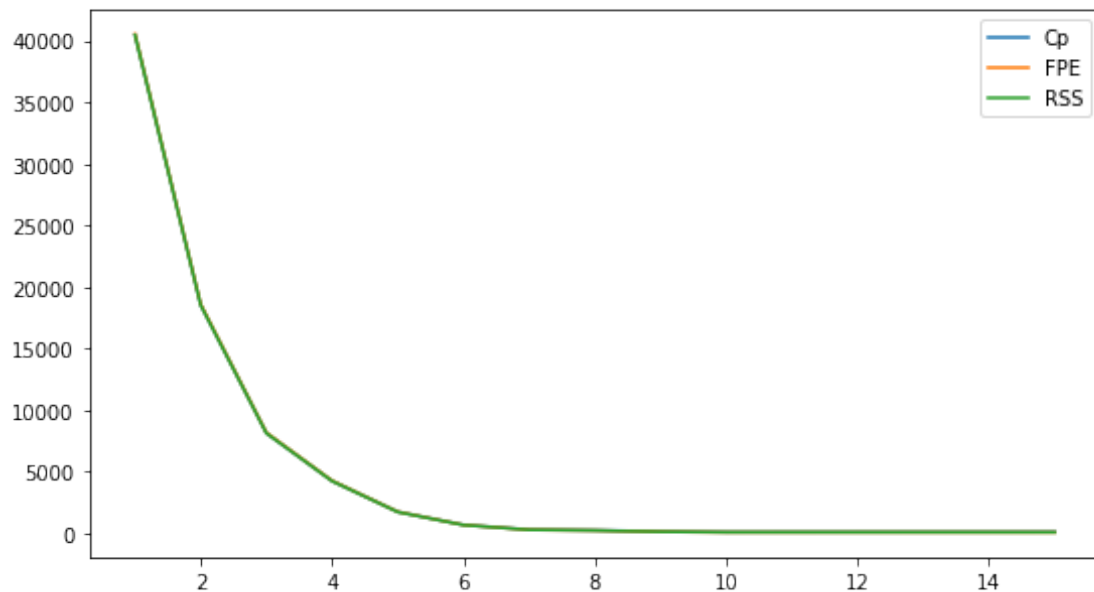
=====

Correlations with target:

[3.83305152e-02 1.55625374e-01 2.09276844e-01 1.39829913e-02
2.83244475e-02 1.57539656e-01 6.13691829e-02 1.90083208e-01
2.06724138e-01 1.23777841e-01 1.24969261e-02 6.35983396e-04
1.01440860e-04 3.86404508e-03 1.67161737e-03]

Regressors order:

[3 9 8 6 2 10 7 1 5 4 11 14 15 12 13]



Optimal:

s* = 10

regressors = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

theta* = [1.05876253 7.50327434 9.85888761 1.3377948 1.96155001 7.46867561
3.35366087 9.27692936 9.01995245 5.40978014 0. 0.
0. 0. 0.]

Кореляції всіх неінформативних регресорів дуже малі - порядку -3 , -4 , що спричинене шумами. В результаті, розмістивши їх в кінці, критерій має мінімум в точці $s = 10$, що якраз відповідає врахуванню тільки впливових регресорів.

1.3.3 Tect 3

```
In [279]: print('True parameters: {}'.format(config3.theta))
          df1 = config3.run_single_LSMB_model_selection(p='correlation',
                                                    criteria=['Cp', 'FPE', 'RSS'],
                                                    plot=True)
```

True parameters: unknown

=====

CORRELATION INCLUDING

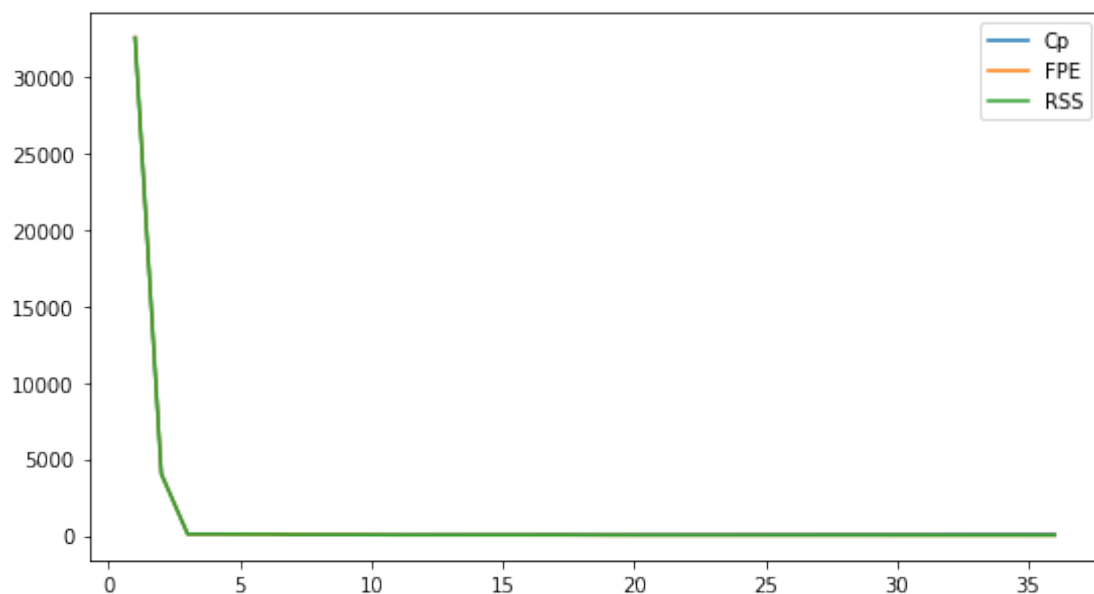
=====

Correlations with target:

```
[5.57596614e-03 4.96374698e+00 8.26491953e-02 4.51442989e-01
 1.63882816e-02 7.07669256e+00 4.66445015e+00 1.63966386e+00
 2.48714016e-01 2.23758820e-02 1.54214723e-01 1.70630669e-01
 7.72140589e-02 2.37815311e-01 1.62738556e-02 9.72242664e-02
 4.89641813e-02 3.17601408e-04 1.30886967e-01 6.30718701e-02
 6.81193001e-02 5.14465628e-03 3.46980247e-01 1.26038896e-01
 4.16687853e+01 2.03176351e-01 5.55885342e+01 3.55780446e-01
 3.95449211e-01 1.43006510e-01 1.56705081e-02 5.93336132e-02
 1.22572971e-02 3.28385950e-02 6.19136590e-02 1.97682209e-02]
```

Regressors order:

```
[27 25 6 2 7 8 4 29 28 23 9 14 26 12 11 30 19 24 16 3 13 21 20 35
 32 17 34 10 36 5 15 31 33 1 22 18]
```



Optimal:

s* = 11

```

regressors = [2, 4, 6, 7, 8, 9, 23, 25, 27, 28, 29]
theta* = [ 0.00000000e+00  6.16508069e-05  0.00000000e+00  1.17978290e-01
0.00000000e+00  1.62350673e-03  3.69267563e-03 -3.26930295e-06
2.14658894e-02  0.00000000e+00  0.00000000e+00  0.00000000e+00
0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
0.00000000e+00  0.00000000e+00  6.47731113e-02  0.00000000e+00
5.15876051e-05  0.00000000e+00  2.72399131e-04  5.69459418e-03
5.16573136e-03  0.00000000e+00  0.00000000e+00  0.00000000e+00
0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]

```

Інформації щодо істинного впливу регресорів на вихідну змінну в цій задачі ми не маємо, тому можемо робити висновки тільки на основі розроблених методів пошуку оптимальної складності. Даний алгоритм показує, що з вихідною величиною дуже корелюють два регресори, для яких значення кореляції мають порядок 1: це 27 і 25; чотири регресори мають кореляції порядку 0: це 6, 2, 7 і 8. Далі 12 регресорів, для яких кореляції порядку -1, і для інших ще менше. Розташували регресори відповідно до кореляцій і виконавши МНК, маємо мінімум критерія в точці $s = 11$, і, дійсно, алгоритм пропонує регресори як з початку (відносно розташування в оригінальній матриці X), так і з кінця, тобто гіпотези, описані вище, підтверджуються.

1.4 Пошук моделі. Метод випадкового включення

1.4.1 Тест 1

```

In [282]: Ks = [10, 20, 50, 100]
          for K in Ks:
              config1.run_single_random_LSMB_model_selection(K=K, criteria=['Cp'],
                                                             main_criterion='Cp')

```

```

=====
RANDOM INCLUDING WITH K = 10
=====

```

	s*	regressors*	Cp
0	4	[1, 2, 3, 5]	95.907950
1	4	[1, 2, 3, 4]	96.001499
2	5	[1, 2, 3, 4, 5]	97.865019
3	5	[1, 2, 3, 4, 5]	97.865019
4	5	[1, 2, 3, 4, 5]	97.865019
...			
	s*	regressors*	Cp
5	5	[1, 2, 3, 4, 5]	97.865019
6	4	[1, 2, 3, 5]	95.907950
7	4	[1, 2, 3, 5]	95.907950
8	5	[1, 2, 3, 4, 5]	97.865019
9	5	[1, 2, 3, 4, 5]	97.865019

Optimal:


```

s* = 4
regressors = [1, 2, 3, 5]
=====
RANDOM INCLUDING WITH K = 20
=====

```

	s*	regressors*	Cp
0	4	[1, 2, 3, 5]	95.907950
1	4	[1, 2, 3, 4]	96.001499
2	4	[1, 2, 3, 4]	96.001499
3	5	[1, 2, 3, 4, 5]	97.865019
4	5	[1, 2, 3, 4, 5]	97.865019
...			
	s*	regressors*	Cp
15	5	[1, 2, 3, 4, 5]	97.865019
16	5	[1, 2, 3, 4, 5]	97.865019
17	4	[1, 2, 3, 5]	95.907950
18	4	[1, 2, 3, 5]	95.907950
19	4	[1, 2, 3, 4]	96.001499

Optimal:

```

s* = 3
regressors = [1, 2, 3]
=====
RANDOM INCLUDING WITH K = 50
=====

```

	s*	regressors*	Cp
0	4	[1, 2, 3, 5]	95.907950
1	4	[1, 2, 3, 5]	95.907950
2	3	[1, 2, 3]	94.017191
3	5	[1, 2, 3, 4, 5]	97.865019
4	5	[1, 2, 3, 4, 5]	97.865019
...			
	s*	regressors*	Cp
45	4	[1, 2, 3, 4]	96.001499
46	4	[1, 2, 3, 5]	95.907950
47	5	[1, 2, 3, 4, 5]	97.865019
48	3	[1, 2, 3]	94.017191
49	5	[1, 2, 3, 4, 5]	97.865019

Optimal:

```

s* = 3
regressors = [1, 2, 3]
=====
RANDOM INCLUDING WITH K = 100
=====

```

	s*	regressors*	Cp
0	5	[1, 2, 3, 4, 5]	97.865019
1	4	[1, 2, 3, 4]	96.001499
2	4	[1, 2, 3, 5]	95.907950
3	5	[1, 2, 3, 4, 5]	97.865019
4	4	[1, 2, 3, 5]	95.907950
...			
	s*	regressors*	Cp
95	5	[1, 2, 3, 4, 5]	97.865019
96	5	[1, 2, 3, 4, 5]	97.865019
97	5	[1, 2, 3, 4, 5]	97.865019
98	5	[1, 2, 3, 4, 5]	97.865019
99	5	[1, 2, 3, 4, 5]	97.865019

Optimal:

```
s* = 3
regressors = [1, 2, 3]
```

При $K \geq 20$ маємо стабільно правильний вибір регресорів. А при меншій кількості випадкових формувань послідовностей метод все ще може враховувати неінформативні регресори. Тож робимо висновок, що $K \approx 20$ буде достатньо для побудови правильної моделі.

1.4.2 Тест 2

```
In [289]: Ks = [50, 100, 200, 500]
          for K in Ks:
              config2.run_single_random_LSMB_model_selection(K=K, criteria=['Cp'],
                                                              main_criterion='Cp')
```

```
=====
RANDOM INCLUDING WITH K = 50
=====
```

	s*	regressors*	Cp
0	13	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 15]	109.052547
1	14	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15]	111.077234
2	14	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15]	111.077234
3	15	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...]	113.012982
4	15	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...]	113.012982
...			
	s*	regressors*	Cp
45	15	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...]	113.012982
46	15	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...]	113.012982
47	15	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...]	113.012982
48	15	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...]	113.012982
49	13	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 13, 14, 15]	109.051216

Optimal:

```

s* = 12
regressors = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
=====
RANDOM INCLUDING WITH K = 100
=====

s* regressors* Cp
0 15 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 113.012982
1 15 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 113.012982
2 15 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 113.012982
3 15 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 113.012982
4 14 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15] 111.050814
...
s* regressors* Cp
95 14 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15] 111.013007
96 14 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15] 111.013640
97 14 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14] 111.073273
98 15 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 113.012982
99 14 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15] 111.013640

```

Optimal:

```

s* = 12
regressors = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13]
=====
RANDOM INCLUDING WITH K = 200
=====

s* regressors* Cp
0 15 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 113.012982
1 15 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 113.012982
2 14 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14] 111.073273
3 15 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 113.012982
4 15 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 113.012982
...
s* regressors* Cp
195 15 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 113.012982
196 15 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 113.012982
197 15 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 113.012982
198 15 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 113.012982
199 15 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 113.012982

```

Optimal:

```

s* = 10
regressors = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
=====
RANDOM INCLUDING WITH K = 500
=====

```

	s*	regressors*	Cp
0	15	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...	113.012982
1	14	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15]	111.050814
2	15	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...	113.012982
3	15	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...	113.012982
4	15	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...	113.012982
...			
	s*	regressors*	Cp
495	15	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...	113.012982
496	14	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15]	111.013007
497	15	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...	113.012982
498	14	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15]	111.077234
499	15	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...	113.012982

Optimal:

```
s* = 11
regressors = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15]
```

В даній задачі за результатами дослідження видно, що алгоритм працює дуже добре при досить великих $K \geq 200$. При меншій кількості випадкових формувань результат роботи менш стабільний (хоча і при великих K не завжди стабільно правильний вибір), і менш точний, хоча і набагато швидший. Тож $K \approx 200$ буде достатньо.

1.4.3 Тест 3

```
In [296]: Ks = [10, 20, 50, 100, 300, 500, 1000]
          for K in Ks:
              config3.run_single_random_LSMB_model_selection(K=K, criteria=['Cp'],
                                                              main_criterion='Cp')
```

```
=====
RANDOM INCLUDING WITH K = 10
=====
```

	s*	regressors*	Cp
0	13	[4, 7, 8, 13, 17, 18, 21, 24, 27, 28, 31, 33, 34]	68.367063
1	19	[1, 5, 7, 8, 9, 10, 11, 13, 14, 16, 18, 19, 22...	79.815037
2	16	[4, 5, 8, 10, 11, 13, 15, 19, 21, 23, 24, 26, ...	75.638691
3	15	[2, 5, 7, 9, 14, 15, 16, 17, 18, 23, 25, 30, 3...	89.511266
4	15	[2, 6, 10, 15, 17, 19, 20, 21, 23, 24, 26, 28,...	92.807331
...			
	s*	regressors*	Cp
5	17	[1, 2, 3, 4, 8, 9, 10, 13, 17, 19, 22, 25, 26,...	74.259286
6	10	[1, 2, 4, 5, 6, 16, 18, 26, 31, 34]	56.133457
7	19	[2, 3, 6, 7, 9, 11, 13, 15, 16, 18, 21, 22, 23...	76.762696
8	10	[2, 10, 14, 15, 16, 22, 26, 27, 28, 36]	77.153497
9	19	[1, 3, 4, 6, 8, 9, 13, 15, 16, 17, 18, 22, 25,...	71.940356

Optimal:

s* = 10

regressors = [1, 2, 4, 5, 6, 16, 18, 26, 31, 34]

=====

RANDOM INCLUDING WITH K = 20

=====

	s*	regressors*	Cp
0	11	[3, 6, 8, 9, 15, 16, 21, 22, 27, 29, 30]	69.034028
1	16	[1, 2, 3, 4, 6, 7, 11, 13, 14, 21, 22, 25, 28, ...]	68.706982
2	27	[2, 3, 4, 5, 7, 8, 9, 11, 12, 14, 15, 16, 17, ...]	84.964975
3	17	[1, 3, 5, 6, 7, 11, 12, 14, 17, 18, 23, 25, 26, ...]	87.369001
4	24	[2, 3, 5, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, ...]	89.806780
...			
	s*	regressors*	Cp
15	21	[1, 3, 4, 7, 10, 11, 13, 14, 15, 17, 18, 19, 2, ...]	76.975123
16	6	[2, 4, 9, 16, 22, 36]	61.096358
17	16	[2, 3, 4, 5, 6, 8, 11, 12, 14, 16, 20, 24, 29, ...]	63.854962
18	19	[2, 3, 5, 6, 7, 8, 10, 11, 15, 16, 17, 18, 19, ...]	77.577881
19	13	[1, 6, 9, 15, 16, 20, 21, 22, 23, 26, 30, 33, 35]	71.786257

Optimal:

s* = 6

regressors = [2, 4, 9, 16, 22, 36]

=====

RANDOM INCLUDING WITH K = 50

=====

	s*	regressors*	Cp
0	18	[2, 6, 7, 8, 9, 12, 13, 14, 15, 17, 18, 21, 23, ...]	84.004745
1	21	[3, 4, 8, 9, 14, 16, 17, 18, 21, 22, 23, 24, 2, ...]	77.665279
2	20	[3, 4, 5, 7, 8, 11, 15, 16, 17, 18, 19, 21, 25, ...]	74.815123
3	23	[3, 7, 9, 12, 13, 14, 15, 17, 18, 19, 21, 23, ...]	90.804127
4	13	[3, 4, 6, 12, 13, 14, 18, 19, 22, 23, 24, 30, 32]	63.870565
...			
	s*	regressors*	Cp
45	16	[3, 4, 7, 9, 10, 13, 16, 17, 18, 19, 21, 23, 2, ...]	65.603009
46	11	[2, 4, 5, 7, 13, 17, 18, 23, 24, 25, 34]	64.474391
47	15	[3, 4, 7, 8, 13, 14, 15, 16, 20, 22, 26, 28, 3, ...]	65.337222
48	29	[1, 3, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, ...]	89.063137
49	10	[3, 5, 6, 19, 23, 24, 28, 29, 32, 35]	82.323554

Optimal:

s* = 11

regressors = [1, 2, 4, 5, 6, 7, 13, 14, 16, 17, 18]

=====

RANDOM INCLUDING WITH K = 100

=====

	s*	regressors*	Cp
0	14	[2, 4, 7, 8, 11, 12, 13, 17, 19, 26, 28, 32, 3...]	71.596149
1	23	[1, 2, 3, 4, 5, 7, 10, 11, 14, 15, 16, 17, 20,...]	78.035155
2	8	[4, 6, 7, 13, 19, 23, 24, 26]	57.358204
3	18	[2, 4, 5, 7, 8, 11, 12, 15, 16, 19, 22, 23, 24...]	74.717910
4	15	[3, 4, 9, 11, 12, 15, 17, 19, 20, 28, 30, 31, ...]	74.383302
...			
	s*	regressors*	Cp
95	15	[2, 4, 5, 7, 13, 14, 17, 18, 19, 25, 26, 27, 2...]	68.784865
96	6	[2, 4, 17, 18, 26, 36]	71.431696
97	25	[1, 2, 3, 4, 6, 8, 9, 10, 12, 13, 14, 15, 16, ...]	83.179062
98	21	[1, 4, 5, 6, 8, 9, 10, 13, 14, 16, 17, 18, 21,...]	73.345661
99	11	[4, 7, 9, 10, 13, 14, 19, 22, 26, 32, 36]	58.853253

Optimal:

s* = 10

regressors = [3, 4, 6, 7, 10, 16, 22, 24, 26, 30]

=====

RANDOM INCLUDING WITH K = 300

=====

	s*	regressors*	Cp
0	10	[1, 3, 4, 7, 12, 16, 17, 26, 30, 36]	55.471972
1	14	[1, 4, 12, 13, 15, 17, 18, 20, 26, 27, 28, 30,...]	72.489937
2	13	[1, 3, 4, 5, 6, 8, 22, 23, 24, 25, 27, 34, 35]	65.088863
3	9	[3, 6, 7, 9, 11, 15, 16, 21, 26]	65.193806
4	15	[3, 4, 5, 9, 12, 13, 15, 17, 18, 19, 21, 27, 3...]	70.587435
...			
	s*	regressors*	Cp
295	11	[1, 2, 3, 4, 8, 12, 13, 14, 27, 29, 36]	66.259638
296	16	[3, 4, 5, 6, 7, 10, 13, 14, 20, 24, 26, 27, 29...]	63.975051
297	4	[6, 16, 17, 26]	63.032251
298	23	[3, 4, 5, 7, 9, 10, 13, 14, 15, 16, 18, 20, 21...]	76.886979
299	19	[3, 6, 7, 8, 9, 11, 12, 14, 15, 16, 17, 19, 20...]	78.850904

Optimal:

s* = 6

regressors = [3, 4, 7, 16, 19, 27]

=====

RANDOM INCLUDING WITH K = 500

=====

	s*	regressors*	Cp
0	10	[3, 5, 6, 16, 19, 21, 23, 24, 28, 33]	72.510077
1	16	[2, 3, 4, 5, 6, 11, 13, 16, 20, 21, 24, 25, 26...]	63.010999
2	22	[4, 6, 7, 8, 9, 11, 14, 16, 17, 20, 21, 22, 23...]	76.964328
3	18	[1, 2, 4, 7, 10, 12, 15, 17, 18, 22, 23, 24, 2...]	76.563448

```

4 22 [1, 3, 4, 6, 8, 11, 12, 13, 15, 16, 17, 18, 19... 77.953536
...
s*                                regressors*                Cp
495 29 [2, 5, 6, 8, 9, 10, 11, 12, 14, 15, 16, 17, 18... 92.359643
496 20 [1, 5, 6, 7, 9, 10, 11, 12, 13, 16, 18, 21, 22... 79.857308
497 14 [1, 7, 8, 9, 10, 12, 15, 16, 17, 23, 26, 31, 3... 77.205777
498 19 [2, 3, 5, 6, 10, 11, 12, 13, 15, 16, 17, 19, 2... 79.754996
499 15 [1, 3, 4, 6, 11, 13, 14, 15, 17, 18, 21, 22, 2... 66.134954

```

Optimal:

```

s* = 9
regressors = [3, 4, 6, 7, 9, 12, 13, 14, 26]

```

=====

RANDOM INCLUDING WITH K = 1000

=====

```

s*                                regressors*                Cp
0 10 [1, 4, 11, 17, 21, 22, 24, 27, 35, 36] 71.556395
1 24 [1, 3, 4, 6, 7, 8, 9, 10, 11, 14, 15, 16, 17, ... 78.606533
2 27 [2, 4, 5, 6, 8, 9, 10, 11, 15, 16, 17, 18, 19,... 85.401718
3 14 [4, 5, 7, 8, 10, 12, 16, 17, 18, 25, 27, 30, 3... 67.357648
4 18 [1, 7, 8, 11, 13, 14, 16, 20, 21, 22, 23, 24, ... 84.101391

```

```

...
s*                                regressors*                Cp
995 21 [1, 2, 4, 5, 6, 8, 9, 10, 13, 16, 18, 19, 23, ... 73.994861
996 21 [2, 3, 4, 6, 8, 11, 13, 14, 15, 17, 18, 19, 20... 77.655959
997 27 [2, 3, 4, 5, 7, 9, 10, 11, 12, 13, 14, 15, 17,... 86.254461
998 15 [4, 5, 8, 12, 13, 14, 19, 21, 22, 27, 29, 30, ... 72.028169
999 23 [1, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 17, 18,... 77.208763

```

Optimal:

```

s* = 6
regressors = [4, 5, 6, 16, 17, 29]

```

```

In [297]: Ks = [2000]
          for K in Ks:
              config3.run_single_random_LSMB_model_selection(K=K, criteria=['Cp'],
                                                              main_criterion='Cp')

```

=====

RANDOM INCLUDING WITH K = 2000

=====

```

s*                                regressors*                Cp
0 20 [1, 4, 6, 8, 12, 13, 14, 15, 17, 18, 19, 27, 2... 77.097628
1 27 [1, 2, 3, 4, 5, 8, 10, 13, 15, 16, 17, 18, 19,... 88.701951
2 25 [1, 2, 4, 5, 7, 8, 9, 10, 12, 13, 15, 17, 18, ... 84.275121

```

```

3 14 [2, 4, 5, 7, 8, 10, 16, 17, 18, 23, 24, 27, 31... 67.502929
4 13 [1, 2, 6, 17, 19, 22, 23, 24, 25, 27, 28, 30, 34] 86.912595
...
      s*                      regressors*          Cp
1995 17 [4, 5, 6, 10, 14, 17, 18, 19, 21, 22, 26, 28, ... 80.121412
1996 25 [1, 2, 3, 4, 7, 10, 11, 12, 13, 14, 15, 17, 19... 83.557161
1997 23 [3, 7, 8, 9, 10, 11, 14, 15, 16, 17, 18, 19, 2... 86.346675
1998 24 [1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, ... 77.661793
1999 27 [2, 3, 4, 5, 6, 8, 9, 11, 12, 14, 17, 19, 21, ... 87.589707

```

Optimal:

```

s* = 7
regressors = [4, 5, 6, 16, 17, 27, 31]

```

Для цієї задачі будемо аналізувати метод, порівнюючи його результати з результатами попереднього алгоритму. Видно, що алгоритм нестабільний, повертає різні комбінації регресорів. Оскільки в цій задачі маємо дуже багато регресорів (36), то припускаємо, що для побудови хорошої моделі потрібно виконати дуже багато випадкових генерувань послідовностей, наприклад $K \approx 1000$. Враховуючи, що попередній метод повертає такий набір: [2, 4, 6, 7, 8, 9, 23, 25, 27, 28, 29], бачимо, що ці результати перетинаються приблизно наполовину. За цим методом завжди або часто вважаються впливовими регресори 4, 6, 7, 16, 17 - перші три мають велику кореляцію з вихідною величиною, а два останні - не дуже велику, проте їх точно потрібно включати в модель. Нестабільність роботи алгоритма також може бути спричинена маленьким розміром вибірки (відносно кількості регресорів) - приблизно 1000, чого скоріш за все недостатньо.

1.5 Пошук моделі. Метод перебірного включення

1.5.1 Тест 1

```

In [300]: total_df, best_df = config1.run_single_picking_LSMB_model_selection(\
                                                criteria=['Cp'],
                                                main_criterion='Cp')

```

```

=====
PICKING INCLUDING
=====

```

Optimal:

```

s* = 3
regressors = [1, 2, 3]

```

Таблиця, яка добре демонструє роботу алгоритма (порядок зберігається), наведена нижче:

```

In [301]: total_df

```

```

Out[301]:      s      regressors      Cp
           0      1      [1]  214.848610

```


1	1	[2]	569.137599
2	1	[3]	389.569791
3	1	[4]	467.699667
4	1	[5]	450.006820
5	2	[1, 2]	118.663149
6	1	[1]	214.848610
7	2	[1, 3]	216.599313
8	2	[1, 4]	203.582136
9	2	[1, 5]	210.770049
10	3	[1, 2, 3]	94.017191
11	3	[1, 2, 4]	119.255285
12	3	[1, 2, 5]	116.878326
13	3	[1, 2, 3]	94.017191
14	4	[1, 2, 3, 4]	96.001499
15	2	[1, 2]	118.663149
16	1	[1]	214.848610
17	4	[1, 2, 3, 5]	95.907950
18	3	[1, 2, 3]	94.017191
19	4	[1, 2, 3, 4]	96.001499
20	5	[1, 2, 3, 4, 5]	97.865019
21	2	[1, 2]	118.663149
22	1	[1]	214.848610

Таблиця, яка містить глобально кращий результат на кожній ітерації підвищення складності

In [302]: best_df

```
Out[302]:  s* regressors*      Cp
0  1          [1]  214.848610
1  2        [1, 2]  118.663149
2  3    [1, 2, 3]   94.017191
3  3    [1, 2, 3]   94.017191
4  3    [1, 2, 3]   94.017191
```

Отже, для даної задачі алгоритм спрацював добре, він повернув оптимальну модель.

1.5.2 Тест 2

```
In [361]: total_df, best_df = config2.run_single_picking_LSMB_model_selection(\
        criteria=['Cp'],
        main_criterion='Cp')
```

```
=====
```

```
PICKING INCLUDING
```

```
=====
```

Optimal:

```
s* = 10
```

```
regressors = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Таблиця, яка добре демонструє роботу алгоритма (порядок зберігається), наведена нижче:

In [362]: total_df

```
Out [362]:
```

	s	regressors	Cp
0	1	[1]	51604.048344
1	1	[2]	43532.540113
2	1	[3]	40484.057161
3	1	[4]	58348.468954
4	1	[5]	54122.151523
5	1	[6]	48234.238030
6	1	[7]	49261.682679
7	1	[8]	41262.597861
8	1	[9]	42697.150718
9	1	[10]	46688.880859
10	1	[11]	52877.477822
11	1	[12]	58507.010273
12	1	[13]	55668.546912
13	1	[14]	58233.789009
14	1	[15]	58075.530065
15	2	[1, 3]	24296.622435
16	1	[3]	40484.057161
17	2	[2, 3]	18772.260468
18	2	[3, 4]	26296.626841
19	2	[3, 5]	24204.462362
20	2	[3, 6]	20248.008097
21	2	[3, 7]	22516.747018
22	2	[3, 8]	15974.764777
23	2	[3, 9]	18544.806985
24	2	[3, 10]	21234.875517
25	2	[3, 11]	24842.104058
26	2	[3, 12]	26383.747773
27	2	[3, 13]	24859.853694
28	2	[3, 14]	25188.913591
29	2	[3, 15]	26605.828375
..
109	10	[2, 3, 4, 5, 6, 7, 8, 9, 10, 12]	127.224293
110	10	[2, 3, 4, 5, 6, 7, 8, 9, 10, 13]	127.227733
111	10	[2, 3, 4, 5, 6, 7, 8, 9, 10, 14]	127.043925
112	10	[2, 3, 4, 5, 6, 7, 8, 9, 10, 15]	126.643167
113	11	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]	105.158353
114	11	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12]	105.125329
115	11	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 13]	105.103888
116	11	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 14]	105.157423
117	11	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15]	105.121343
118	10	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]	103.158799
119	11	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]	105.158353
120	12	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]	107.125305

```

121  9          [2, 3, 4, 5, 6, 7, 8, 9, 10]      125.228342
122  8          [2, 3, 5, 6, 7, 8, 9, 10]      178.976926
123  7          [2, 3, 6, 7, 8, 9, 10]      297.908708
124  6          [2, 3, 6, 8, 9, 10]      684.680793
125  5          [2, 3, 6, 8, 9]      1740.698164
126  4          [3, 6, 8, 9]      4250.629045
127  3          [3, 8, 9]      8138.785038
128  2          [3, 8]      15974.764777
129  1          [3]      40484.057161
130 12          [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13]      107.103842
131 12          [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 14]      107.157069
132 12          [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 15]      107.120060
133 13          [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]      109.073276
134 13          [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14]      109.124952
135 13          [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 15]      109.079281
136 14          [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]      111.073273
137 14          [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15]      111.013640
138 15          [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...]      113.012982

[139 rows x 3 columns]

```

Таблиця, яка містить глобально кращий результат на кожній ітерації підвищення складності

In [363]: best_df

```

Out[363]:      s*      regressors*      Cp
0      1          [3]      40484.057161
1      2          [3, 8]      15974.764777
2      3          [3, 8, 9]      8138.785038
3      4          [3, 6, 8, 9]      4250.629045
4      5          [2, 3, 6, 8, 9]      1740.698164
5      6          [2, 3, 6, 8, 9, 10]      684.680793
6      7          [2, 3, 6, 7, 8, 9, 10]      297.908708
7      8          [2, 3, 5, 6, 7, 8, 9, 10]      178.976926
8      9          [2, 3, 4, 5, 6, 7, 8, 9, 10]      125.228342
9     10          [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]      103.158799
10    10          [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]      103.158799
11    10          [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]      103.158799
12    10          [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]      103.158799
13    10          [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]      103.158799
14    10          [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]      103.158799

```

Для даної задачі також алгоритм спрацював добре і повернув оптимальну модель. Час роботи алгоритма тут звісно більший за перший метод (кореляційного включення), проте менший за другий метод (випадкового включення) і при цьому повертає правильну структуру, тож в цьому плані є кращим.

1.5.3 Тест 3

```
In [311]: total_df, best_df = config3.run_single_picking_LSMB_model_selection(\
                                                criteria=['Cp'],
                                                main_criterion='Cp')
```

```
=====
PICKING INCLUDING
=====
Optimal:
    s* = 7
    regressors = [3, 4, 5, 6, 9, 16, 26]
```

Таблиця, яка добре демонструє роботу алгоритма (порядок зберігається), наведена нижче:

```
In [312]: total_df
```

```
Out [312]:
```

	s	regressors	Cp
0	1	[1]	6922.622699
1	1	[2]	171607.168945
2	1	[3]	595.423285
3	1	[4]	8072.403830
4	1	[5]	8415.507144
5	1	[6]	158.358196
6	1	[7]	175.164569
7	1	[8]	208501.298773
8	1	[9]	69222.239836
9	1	[10]	187787.682459
10	1	[11]	19886.729830
11	1	[12]	5277.995979
12	1	[13]	280.697357
13	1	[14]	119107.026909
14	1	[15]	207716.568406
15	1	[16]	208.827496
16	1	[17]	124823.714740
17	1	[18]	199572.038985
18	1	[19]	20872.240938
19	1	[20]	131690.187233
20	1	[21]	15268.403104
21	1	[22]	3751.889765
22	1	[23]	10822.918605
23	1	[24]	107593.708684
24	1	[25]	4139.687219
25	1	[26]	28801.198215
26	1	[27]	32587.576839
27	1	[28]	109136.079786
28	1	[29]	95121.691000
29	1	[30]	182670.791806

```

.. ..
672 31 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 89.009264
673 31 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 89.051761
674 31 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 89.120734
675 31 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 89.106388
676 32 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 90.880558
677 32 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 91.006471
678 32 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 91.048445
679 32 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 91.117755
680 32 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 91.103769
681 33 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 92.767377
682 33 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 92.800002
683 33 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 92.878469
684 33 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 92.855271
685 34 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 94.695102
686 34 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 94.766422
687 34 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 94.739615
688 35 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 96.693792
689 35 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 96.668793
690 25 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 77.229872
691 26 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 79.226699
692 27 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 81.226596
693 28 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 83.149919
694 29 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 85.148968
695 30 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 87.124184
696 31 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 89.120961
697 32 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 90.880558
698 33 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 92.767377
699 34 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 94.695102
700 35 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 96.693792
701 36 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14... 98.667221

```

[702 rows x 3 columns]

Таблиця, яка містить глобально кращий результат на кожній ітерації підвищення складності

In [313]: best_df

```

Out[313]:      s*      regressors*      Cp
0      1              [6]  158.358196
1      2             [6, 16]   78.361004
2      3            [4, 6, 16]   54.822413
3      4           [3, 4, 6, 16]  49.481128
4      5          [3, 4, 5, 6, 16]  47.128611
5      6         [3, 4, 5, 6, 9, 16]  45.398392
6      7      [3, 4, 5, 6, 9, 16, 26]  44.912894
7      7      [3, 4, 5, 6, 9, 16, 26]  44.912894

```

8	7	[3, 4, 5, 6, 9, 16, 26]	44.912894
9	7	[3, 4, 5, 6, 9, 16, 26]	44.912894
10	7	[3, 4, 5, 6, 9, 16, 26]	44.912894
11	7	[3, 4, 5, 6, 9, 16, 26]	44.912894
12	7	[3, 4, 5, 6, 9, 16, 26]	44.912894
13	7	[3, 4, 5, 6, 9, 16, 26]	44.912894
14	7	[3, 4, 5, 6, 9, 16, 26]	44.912894
15	7	[3, 4, 5, 6, 9, 16, 26]	44.912894
16	7	[3, 4, 5, 6, 9, 16, 26]	44.912894
17	7	[3, 4, 5, 6, 9, 16, 26]	44.912894
18	7	[3, 4, 5, 6, 9, 16, 26]	44.912894
19	7	[3, 4, 5, 6, 9, 16, 26]	44.912894
20	7	[3, 4, 5, 6, 9, 16, 26]	44.912894
21	7	[3, 4, 5, 6, 9, 16, 26]	44.912894
22	7	[3, 4, 5, 6, 9, 16, 26]	44.912894
23	7	[3, 4, 5, 6, 9, 16, 26]	44.912894
24	7	[3, 4, 5, 6, 9, 16, 26]	44.912894
25	7	[3, 4, 5, 6, 9, 16, 26]	44.912894
26	7	[3, 4, 5, 6, 9, 16, 26]	44.912894
27	7	[3, 4, 5, 6, 9, 16, 26]	44.912894
28	7	[3, 4, 5, 6, 9, 16, 26]	44.912894
29	7	[3, 4, 5, 6, 9, 16, 26]	44.912894
30	7	[3, 4, 5, 6, 9, 16, 26]	44.912894
31	7	[3, 4, 5, 6, 9, 16, 26]	44.912894
32	7	[3, 4, 5, 6, 9, 16, 26]	44.912894
33	7	[3, 4, 5, 6, 9, 16, 26]	44.912894
34	7	[3, 4, 5, 6, 9, 16, 26]	44.912894
35	7	[3, 4, 5, 6, 9, 16, 26]	44.912894

Для цієї задачі алгоритм працює довго, (хоча знову не довше ніж другий метод випадкового включення) і повертає такий набір регресорів [3, 4, 5, 6, 9, 16, 26]. Багато з них (3, 4, 6, 16, 26) дійсно перетинаються з результатами попередніх методів.

Якщо об'єднати результати трьох методів для цієї задачі, то отримаємо такий набір регресорів і такі ваги для них:

```
In [365]: regressors1 = {2, 4, 6, 7, 8, 9, 23, 25, 27, 28, 29}
          regressors2 = {4, 5, 6, 16, 17, 27, 31}
          regressors3 = {3, 4, 5, 6, 9, 16, 26}
          total = list(a.union(b).union(c))

          theta, _, RSS_next, df = LSMB(config3.X[:, total], config3.y,
                                         verbose=False, create_dataframe=True)

          print('{} regressors: {}'.format(len(total), total))
          print('  with theta: {}'.format(theta.flatten()))

17 regressors: [2, 3, 4, 5, 6, 7, 8, 9, 16, 17, 23, 25, 26, 27, 28, 29, 31]
   with theta: [ 1.58490768e-01  1.25658013e-01  3.24787641e-02  2.18965046e-03
```

2.52196152e-03 2.70591730e-06 1.24677093e-02 -7.81374595e-04
9.09810152e-03 -1.72335314e-02 7.60034785e-02 7.21612391e-02
7.66623610e-05 5.78935734e-03 8.60323900e-03 7.36291655e-03
1.28642815e-02]

2 Висновки

Досліджуючи три методи пошуку моделі оптимальної складності, ми з'ясували - якщо говорити про ефективність алгоритму в контексті правильної структури і швидкодії, то вона залежить від умов і задачі, великий вплив на неї мають розмірності вибірок (кількість екземплярів та кількість регресорів) та наявність шумів. Конкретно для наших трьох задач: Тест 1 є найпростішим з усіх, і всі алгоритми ефективно побудували модель; Тест 2 є більш великою задачею, і для нього найефективнішими виявилися метод кореляційного включення та метод перебірного включення (хоча останній трохи менше через трохи повільнішу роботу), а метод випадкового включення також працює добре, проте потребує великої кількості випадкових генерацій послідовностей; Тест 3 є реальною задачею, для нього невідомі істинні параметри моделі, але ефективність алгоритмів очевидна, оскільки з 36 регресорів ми зупинилися на 17ти найвпливовіших.

3 Код програми

3.1 Імпорт необхідних бібліотек, налаштування

```
In [237]: import numpy as np
import pandas as pd
from scipy.integrate import odeint
from itertools import permutations
import matplotlib.pyplot as plt
%matplotlib inline

from pylab import rcParams
rcParams['figure.figsize'] = 9, 5
```

3.2 Реалізація МНКО

До реалізації додано обрахування W - розширеної матриці нормальної системи рівнянь для повної моделі на початку роботи МНКО, і подальше її використання. Як наслідок, маємо більш оптимізований алгоритм, оскільки в ньому тепер замість зайвих операцій над частинами однієї й тієї ж матриці відбувається звернення до частин матриці W .

```
In [238]: def LSMB(X, y, W=None, s=None, started=True, sigma_estimation=None,
                verbose=False, deep_verbose=False, create_dataframe=False):
    assert X.ndim == 2 and X.shape[1] > 0
    m = X.shape[1]
    if started:
        W = np.hstack((X.T @ X, (X.T @ y).reshape(-1, 1)))
        started = False
    assert W.ndim == 2 and W.shape[0] == m
    if m > 1:
        if create_dataframe:
            w, H_inv, RSS, df = LSMB(X[:, :-1], y, W[:, :-1, :], s,
                                     started, sigma_estimation,
                                     verbose, deep_verbose, create_dataframe)

            if s and m > s:
                return w, H_inv, RSS, df
        else:
            w, H_inv, RSS = LSMB(X[:, :-1], y, W[:, :-1, :], s,
                                started, sigma_estimation,
                                verbose, deep_verbose, create_dataframe)

            if s and m > s:
                return w, H_inv, RSS
    # w is of shape = [m-1, 1]; H_inv is of shape = [m-1, m-1]
    h = W[:, m-1, m-1].reshape(-1, 1) # shape = [m-1, 1]
    eta = W[m-1, m-1] # shape = [1, 1]
    alpha = H_inv @ h # shape = [m-1, 1]
    beta = eta - h.T @ alpha # shape = [1, 1]
    beta_inv = 1 / beta # shape = [1, 1]
    gamma = W[m-1, -1] # shape = [1, 1]
```

```

nu = beta_inv * (gamma - h.T @ w) # shape = [1, 1]
w = np.vstack((w - nu * alpha, nu)) # shape = [m, 1]
H_next_inv = np.vstack((np.hstack((H_inv + beta_inv * alpha @ alpha.T,
                                   (- beta_inv * alpha).reshape(-1, 1))),
                        np.hstack((-beta_inv * alpha.T, beta_inv))))
RSS_next = (RSS - nu.flatten() ** 2 * beta.flatten())[0]

else: # 1
    H_inv = np.array([[0]])
    eta = beta = W[0,0] #  $X[:, -1].T @ X[:, -1]$ 
    beta_inv = 1 / beta
    alpha = h = np.array([0])
    gamma = W[0, -1] #  $X[:, -1].T @ y$ 
    nu = np.array([beta_inv * gamma])
    w = np.array([nu])
    H_next_inv = np.array(beta_inv).reshape(1, 1)
    RSS_next = (y.T @ y - W[0, -1:] @ w)[0]
    if create_dataframe:
        if sigma_estimation is None:
            df = pd.DataFrame(columns=['s', 'RSS', 'Cp', 'FPE', 'theta'])
        else:
            df = pd.DataFrame(columns=['s', 'RSS', 'Cp_simple',
                                      'Cp', 'FPE', 'theta'])

if verbose:
    print('=====')
    print('\tStep {}'.format(m))
    print('=====')
    if deep_verbose:
        print('h_{}: \t\t{}'.format(m, h.reshape(-1,1)[: ,0]))
        print('eta_{}: \t\t{}'.format(m, eta))
        print('alpha_{}: \t\t{}'.format(m, alpha.reshape(-1,1)[: ,0]))
        print('beta_{}: \t\t{}'.format(m, beta))
        print('gamma_{}: \t\t{}'.format(m, gamma))
        print('nu_{}: \t\t{}'.format(m, nu))
        print('=====')
    print('>  $\theta$ _: {}'.format(m, w[:, 0]))
    print('> H_{}_inv: \n{}'.format(m, H_next_inv))
    print('> RSS_{}: {}'.format(m, RSS_next))
if create_dataframe:
    n = y.shape[0]
    FPE = (n + m) / (n - m) * RSS_next
    Cp_simple = RSS_next + 2 * m
    if sigma_estimation is None:
        df = df.append({'s': m, 'RSS': RSS_next,
                       'Cp': Cp_simple, 'FPE': FPE,
                       'theta': w[:, 0]},
                      ignore_index=True)

```

```

else:
    Cp = RSS_next + 2 * sigma_estimation * m
    df = df.append({'s': m, 'RSS': RSS_next,
                    'Cp_simple': Cp_simple,
                    'Cp': Cp, 'FPE': FPE,
                    'theta': w[:, 0]},
                  ignore_index=True)
    return w, H_next_inv, RSS_next, df
return w, H_next_inv, RSS_next

```

3.3 Реалізація класу моделювання і усіх необхідних методів

```

In [ ]: class ModelConfig():
    m = 5
    n = 10
    n_grid = [10, 30, 100]
    theta = np.array([3, -2, 1, 0, 0])
    a = 0
    b = 2
    sigma = 0.3
    s0 = 3
    s = m

    def __init__(self, m=None, s0=None, theta=None, a=None, b=None,
                  X=None, y=None):
        if m:
            self.m = m
            self.s = m
        if s0:
            self.s0 = s0
        if theta:
            if isinstance(theta, dict) and theta['random']:
                self.theta = np.zeros(self.m)
                self.theta[:self.s0] = np.random.uniform(*theta['random'],
                                                            size=self.s0)

            elif theta == 'unknown':
                assert X is not None, 'Please provide data (X)'
                assert y is not None, 'Please provide target values as well (y)'
                self.theta = theta
            else:
                self.theta = theta
        if a:
            self.a = a
        if b:
            self.b = b
        if X is not None:
            assert y is not None, 'Please provide target values as well (y)'
            self.X = np.array(X)

```

```

        self.y = np.array(y)
        (self.n, self.m) = X.shape
        self.s = self.m
    else:
        self.compile()

def generate_noise_and_output(self):
    self.ksi = np.random.normal(0, self.sigma, size=self.n)
    self.y = self.X @ self.theta + self.ksi

def compile(self, n=None, sigma=None):
    if n:
        self.n = n
    if sigma:
        self.sigma = sigma
    self.X = np.random.uniform(self.a, self.b, size=(self.n, self.m))
    self.generate_noise_and_output()

def show(self, n_limit=10):
    print('Regressors: m = {}'.format(self.m))
    print('True parameters:  $\theta = \{\}$ '.format(self.theta))
    if not isinstance(self.theta, str):
        equation_str = ''
        for i, theta_i in enumerate(self.theta):
            equation_str += ' + ({})*x{}'.format(theta_i, i+1)
        equation_str = 'y0 = ' + equation_str[3:]
        print(equation_str)
        print('Noise generation:  $\sigma = \{\}$ '.format(self.sigma))
    print('Sample length: n = {}'.format(self.n))
    print('X[:10]:\n{}'.format(self.X[:n_limit]))
    print('y[:10]:\n{}'.format(self.y[:n_limit]))

def show_estimations(self):
    print('RSS(m) = {:.5f}'.format(self.RSS))
    print('σ* = {:.5f}'.format(self.sigma_hat ** .2))

def estimate_sigma(self):
    _, _, self.RSS = RMNK(self.X, self.y, s=self.s,
                          verbose=False, create_dataframe=False)
    self.sigma_hat = self.RSS ** 2 / (self.X.shape[0] - self.X.shape[1])

def run_grid_LSMB_model_selection(self):
    self.estimate_sigma()
    for i, n in enumerate(self.n_grid):
        for j, sigma in enumerate(self.sigma_grid):
            self.compile(n, sigma)
            print('-----')
            print('\t\t\tSAMPLE #{}'.format(i * len(self.n_grid) + j + 1))

```

```

print('-----')
print('\t\tCONFIGURATIONS & DATA')
self.show()
print('\n\t\tRLSM ITERATIONS')
theta_pred, _, _, df = LSMB(self.X, self.y, s=self.s,
                             verbose=True,
                             create_dataframe=True)

print('\n\t\tRESULTS')
print('\nPARAMETERS')
print('True values:\t $\theta$ : {}'.format(self.theta))
print('Estimates:\t $\theta^*$ : {}'.format(theta_pred[:,0]))
plt.plot(df['s'], df['RSS'], label='RSS')
plt.plot(df['s'], df['Cp'], label='Cp')
plt.plot(df['s'], df['FPE'], label='FPE')
plt.legend()
plt.show()
print(df)
print('s* by Cp: {}'.format(np.array(df['Cp']).argmin()+1))
print('s* by FPE: {}'.format(np.array(df['FPE']).argmin()+1))
print()

def run_single_LSMB_model_selection(self, p=None, plot=False,
                                    criteria=['Cp', 'FPE', 'RSS']):
    """Single LSMB

    p : list or str
        if list: permutation indices
        if str: one of 'direct', 'reverse', 'correlation'
        defines how to create permutation
    """
    if p == 'reverse':
        print('===== \n REVERSE \n =====')
        p = np.flip(np.arange(self.m), axis=0)
    elif p == 'correlation':
        print('===== \n CORRELATION INCLUDING \n =====')
        correlations = np.abs(np.cov(self.X.T, self.y.T)[-1, :-1])
        p = np.argsort(-correlations)
        print('Correlations with target: \n \t {}'.format(correlations))
    elif isinstance(p, list):
        print('===== \n CUSTOM \n =====')
    else:
        print('===== \n DIRECT \n =====')
        p = np.arange(self.m)
    print('Regressors order: \n \t {}'.format(p+1))
    theta_pred, _, _, df = LSMB(self.X[:,p], self.y, s=self.s,
                                verbose=False, create_dataframe=True)

    df['regressors'] = [str(sorted(p[:int(s)]+1)) for s in df.s]

```

```

for criterion in criteria:
    df[criterion] = np.round(df[criterion], 6)
if plot:
    for criterion in criteria:
        plt.plot(df['s'], df[criterion], label=criterion)
        plt.legend()
    plt.show()
df = df.sort_values(by=criteria).reset_index()\
      [['s', 'regressors', 'theta'] + criteria]
self.s_opt, self.regressors_opt, \
self.theta_opt = df.loc[0, ['s', 'regressors', 'theta']]
self.theta_opt = np.hstack((self.theta_opt,
                             np.zeros(self.m -
                                         len(self.theta_opt))))[np.argsort(p)]

print('Optimal:')
print('\ts* = {}'.format(self.s_opt))
print('\tregressors = {}'.format(self.regressors_opt))
print('\ttheta* = {}'.format(self.theta_opt))
return df

def run_single_full_LSMB_model_selection(self,
                                         criteria=['Cp', 'FPE', 'RSS']):
    print('=====\nBRUT FORCE\n=====')
    total_df = pd.DataFrame()
    for p in permutations(range(self.m)):
        p = np.array(p)
        theta_pred, _, _, df = LSMB(self.X[:,p], self.y, s=self.s,
                                     verbose=False, create_dataframe=True)
        df = df.drop(columns=['theta'])
        df['regressors'] = [str(sorted(p[:int(s)]+1)) for s in df.s]
        total_df = pd.concat([total_df, df], axis=0)
    for criterion in criteria:
        total_df[criterion] = np.round(total_df[criterion], 6)
    total_df = total_df.drop_duplicates()
    total_df = total_df.sort_values(by=criteria).reset_index()\
      [['s', 'regressors'] + criteria]
    return total_df

def run_single_random_LSMB_model_selection(self, K=20,
                                           criteria=['Cp', 'FPE', 'RSS'],
                                           main_criterion='Cp'):
    print('=====\nRANDOM INCLUDING WITH K = {}\n=====')
    permutations = [np.random.permutation(self.m) for k in range(K)]
    total_df = pd.DataFrame()
    best_df = pd.DataFrame()
    self.main_criterion_value = np.Inf
    for p in permutations:
        p = np.array(p)

```

```

        theta_pred, _, _, df = LSMB(self.X[:,p], self.y, s=self.s,
                                   verbose=False, create_dataframe=True)
    df = df.drop(columns=['theta'])
    df['regressors'] = [str(sorted(p[:int(s)]+1)) for s in df.s]
    df = df.sort_values(by=criteria).reset_index()
    main_criterion_value = df.loc[0, main_criterion]
    if main_criterion_value < self.main_criterion_value:
        self.main_criterion_value = main_criterion_value
        self.theta_pred = theta_pred
        self.s_opt, self.regressors_opt = df.loc[0,
                                                ['s', 'regressors']]

    total_df = pd.concat([total_df, df], axis=0)
    best_df = pd.concat([best_df, df.loc[0:0]], axis=0)
    for criterion in criteria:
        total_df[criterion] = np.round(total_df[criterion], 6)
        best_df[criterion] = np.round(best_df[criterion], 6)
    total_df = total_df.drop_duplicates()
    total_df = total_df.sort_values(by=criteria).reset_index()\
        [['s', 'regressors'] + criteria]
    best_df = best_df.rename(columns={'s': 's*',
                                     'regressors': 'regressors*'}).reset_index()\
        [['s*', 'regressors*'] + criteria]

    print()
    print(best_df[:5])
    print('...')
    print(best_df[-5:])
    print()
    print('Optimal:')
    print('\ts* = {}\n\tregressors = {}'.format(self.s_opt,
                                                self.regressors_opt))

    return total_df, best_df

def run_single_picking_LSMB_model_selection(self,
                                           criteria=['Cp', 'FPE', 'RSS'],
                                           main_criterion='Cp'):
    print('=====\\nPICKING INCLUDING\\n=====')
    total_df = pd.DataFrame()
    best_df = pd.DataFrame()
    self.main_criterion_value = np.Inf
    remained_indices = list(range(self.m))
    regressors_indices = []
    while len(remained_indices) > 0:
        local_main_criterion_value = np.Inf
        for i in remained_indices:
            p = np.array(regressors_indices + [i])
            theta_pred, _, _, df = LSMB(self.X[:,p], self.y, s=len(p),
                                         verbose=False,
                                         create_dataframe=True)

```

```

df = df.drop(columns=['theta'])
df['regressors'] = [str(sorted(p[:int(s)]+1)) for s in df.s]
df = df.sort_values(by=criteria).reset_index()
main_criterion_value = df.loc[0, main_criterion]
if main_criterion_value < local_main_criterion_value:
    local_main_criterion_value = main_criterion_value
    local_df = df[:1]
    local_i = i
if main_criterion_value < self.main_criterion_value:
    self.main_criterion_value = main_criterion_value
    self.theta_pred = theta_pred
    self.s_opt, \
    self.regressors_opt = df.loc[0, ['s', 'regressors']]
total_df = pd.concat([total_df, df], axis=0)
regressors_indices += [local_i]
remained_indices.remove(local_i)
best_df = pd.concat([best_df, local_df], axis=0)
for criterion in criteria:
    total_df[criterion] = np.round(total_df[criterion], 6)
    best_df[criterion] = np.round(best_df[criterion], 6)
total_df = total_df.drop_duplicates().reset_index()\
    [['s', 'regressors'] + criteria]
#     total_df = total_df.sort_values(by=criteria).reset_index()\
#     [['s', 'regressors'] + criteria]
best_df = best_df.rename(columns={'s': 's*',
                                'regressors': 'regressors*'}).reset_index()\
    [['s*', 'regressors*'] + criteria]
print('Optimal:')
print('\ts* = {}\n\tregressors = {}'.format(self.s_opt,
                                             self.regressors_opt))
return total_df, best_df

```