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

Лабороторна робота № 2
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

1 Модель Фергюльста

1.1 Рівняння моделі

$$N' = \mu N(k - N) \;\mid\; N_0$$

1.2 Різницеве рівняння

$\Delta t = 1$

$$N'(t) \approx N(t+1) - N(t) - \text{різниця вперед}$$

$$N'(t) \approx N(t) - N(t-1) - \text{різниця назад}$$

$$N'(t) \approx \frac{N(t+1) - N(t-1)}{2} - \text{центральна різниця}$$

Розглядатимемо для різниці вперед:

$$\begin{cases} N(t+1) - N(t) = \mu N(t)[k - N(t)] \\ N(0) = N_0 \end{cases}$$

$$\iff \begin{cases} N(t+1) = (\mu k + 1)N(t) - \mu N^2(t) \\ N(0) = N_0 \end{cases}$$

Отже, лінійна регресійна залежність виглядає таким чином:

$$\begin{cases} y_i = \theta_1 x_{i1} + \theta_2 x_{i2} + \xi_i \\ \theta_1 = \mu k + 1 \\ \theta_2 = -\mu \\ E\xi = 0_n; \;\; cov(\xi) = \sigma^2 I_n \end{cases}$$

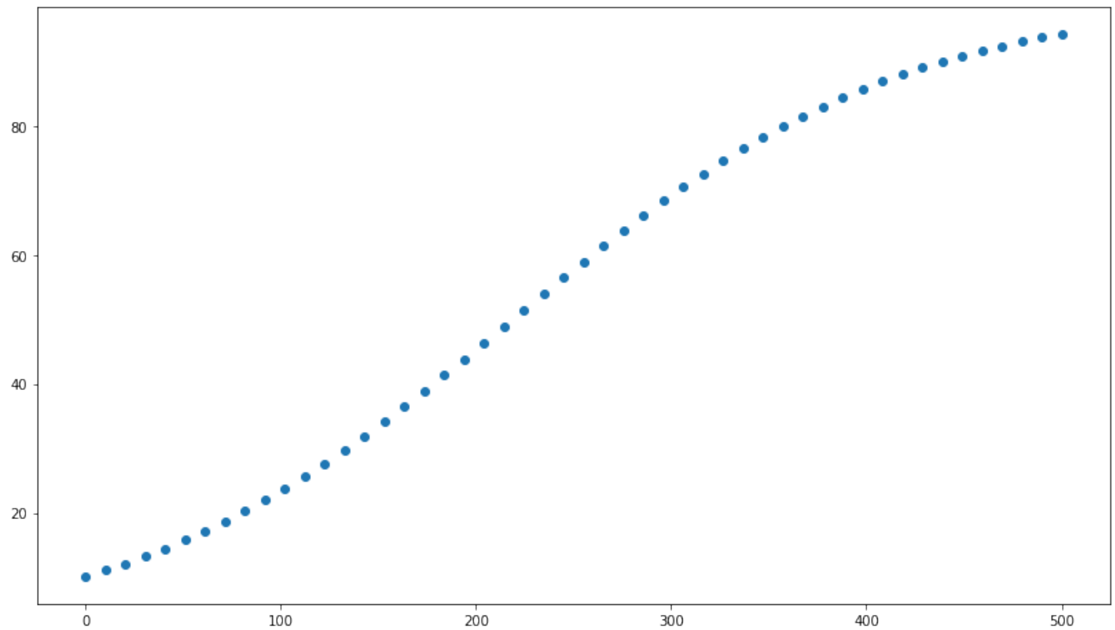
$$\Rightarrow \begin{cases} \mu = -\theta_2 \\ k = (1 - \theta_1)/\theta_2 \end{cases}$$

1.3 Генерування вибірки

In [41]:
```
config = VerhulstModelConfig()
config.show()

plt.scatter(config.t, config.y)
plt.show()
print('Intermediate parameters values: θ_1 = {}\tθ_2 = {}'.format(*config.theta))
print('Regression model: y = ({}) * x1 + ({}) * x2'.format(*config.theta))
config.df.head(10)
```

```
Initial parameters:        μ = 0.0001
                                  k = 100
                      N0 = 10
```

```
Noise generation: C = 3
Sample length: n = 50
Time starting from 0 to 500                with discretization frequency 10
```




```
Intermediate parameters values: θ_1 = 1.01         θ_2 = -0.0001
Regression model: y = (1.01) * x1 + (-0.0001) * x2
```

Out[41]:

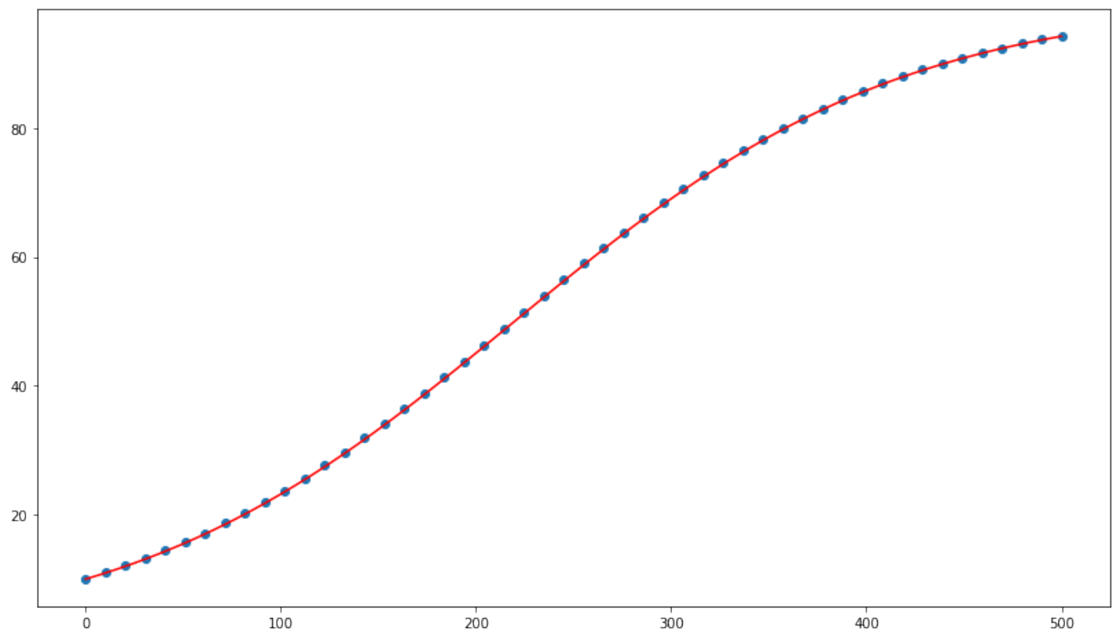
	i	t	N(t)	N^2(t)	N(t+1)
0	1	0	10.000000	100.000000	10.090
1	2	10	10.956582	120.046684	11.054
2	3	20	11.992475	143.819461	12.098
3	4	30	13.111886	171.921546	13.226
4	5	40	14.318785	205.027597	14.441
5	6	51	15.616808	243.884680	15.749
6	7	61	17.009139	289.310822	17.150
7	8	71	18.498393	342.190549	18.649
8	9	81	20.086481	403.466709	20.247
9	10	91	21.774481	474.128027	21.945

1.4 Робота алгоритму МНКО

In [25]:
```
config.run_single_RMNK(verbose=True, deep_verbose=True)
```

```
Recurrent Least Squares Method
===============================================
```

```
        Step 1
================================================
h_1:                [0]
eta_1:                192021.11695382808
alpha_1:        [0]
beta_1:                192021.11695382808
gamma_1:        192454.57239914758
nu_1:                [1.00225733]
================================================
> θ_1: [1.00225733]
> H_1_inv:
[[5.20776056e-06]]
> RSS_1: 0.5984206103312317
================================================
        Step 2
================================================
h_2:                [14865979.31260227]
eta_2:                1210738873.994198
alpha_2:        [77.41846078]
beta_2:                [[59837637.56477976]]
gamma_2:        14893552.79759949
nu_2:                [[-0.0001]]
================================================
> θ_2: [ 1.00999944e+00 -1.00003420e-04]
> H_2_inv:
[[ 1.05372445e-04 -1.29380878e-06]
 [-1.29380878e-06  1.67118897e-08]]
> RSS_2: 3.3057016820547958e-06
==============================================

INTERMEDIATE PARAMETERS
True values:        θ_1 = 1.01        θ_2 = -0.0001
Estimates:        θ_1* = 1.0099994430189054        θ_2* = -0.00010000341994466655

INITIAL PARAMETERS
True values:        μ = 0.0001        k = 100
Estimates:        μ* = 0.00010000341994466655        k* = 99.9910105518219
```



1.5 Таблиця залежності оцінок від рівня шуму

In [26]:
```
config.run_grid_RMNK(verbose=False)
```

Out[26]:

	C	num_samples	θ_1	θ_1*	θ_2	θ_2*	μ	μ*	\
0	0.0	10.0	1.01	0.998957	-0.0001	0.000044	0.0001	-0.000044	
1	0.0	50.0	1.01	1.010265	-0.0001	-0.000106	0.0001	0.000106	
2	0.0	100.0	1.01	1.009688	-0.0001	-0.000097	0.0001	0.000097	
3	2.0	10.0	1.01	1.009857	-0.0001	-0.000098	0.0001	0.000098	
4	2.0	50.0	1.01	1.009994	-0.0001	-0.000100	0.0001	0.000100	
5	2.0	100.0	1.01	1.009956	-0.0001	-0.000099	0.0001	0.000099	
6	5.0	10.0	1.01	1.010000	-0.0001	-0.000100	0.0001	0.000100	
7	5.0	50.0	1.01	1.010000	-0.0001	-0.000100	0.0001	0.000100	
8	5.0	100.0	1.01	1.010000	-0.0001	-0.000100	0.0001	0.000100	

	k	k*
0	100.0	23.826999
1	100.0	96.768521
2	100.0	100.083587
3	100.0	100.559471
4	100.0	100.105598
5	100.0	100.113571
6	100.0	100.000233
7	100.0	100.000141
8	100.0	100.000091

2 Рівняння згасаючих коливань

2.1 Модель рівняння

$$x'' + 2\delta x' + \omega_0^2 x = 0 \mid x_0, x'_0$$

2.2 Різницеве рівняння

$\Delta t = 1$

$$x'(t) \approx x(t+1) - x(t) - \text{різниця вперед}$$

$$x'(t) \approx x(t) - x(t-1) - \text{різниця назад}$$

$$x'(t) \approx \frac{x(t+1) - x(t-1)}{2} - \text{центральна різниця}$$

$$x''(t) \approx x(t+1) - 2x(t) + x(t-1)$$

Розглянемо для апроксимації різницею вперед:

$$\begin{cases} x(t+1) - 2x(t) + x(t-1) + 2\delta[x(t+1) - x(t)] + \omega_0^2 x(t) = 0 \\ x(0) = x_0 \\ x(1) = x(0) + x'(0) = x_0 + x'_0 \end{cases}$$

$$\iff \begin{cases} x(t+2) = \frac{2+2\delta-\omega_0^2}{1+2\delta} x(t+1) - \frac{1}{1+2\delta} x(t) \\ x(0) = x_0 \\ x(1) = x_0 + x'_0 \end{cases}$$

Отже, лінійна регресійна залежність виглядає таким чином:

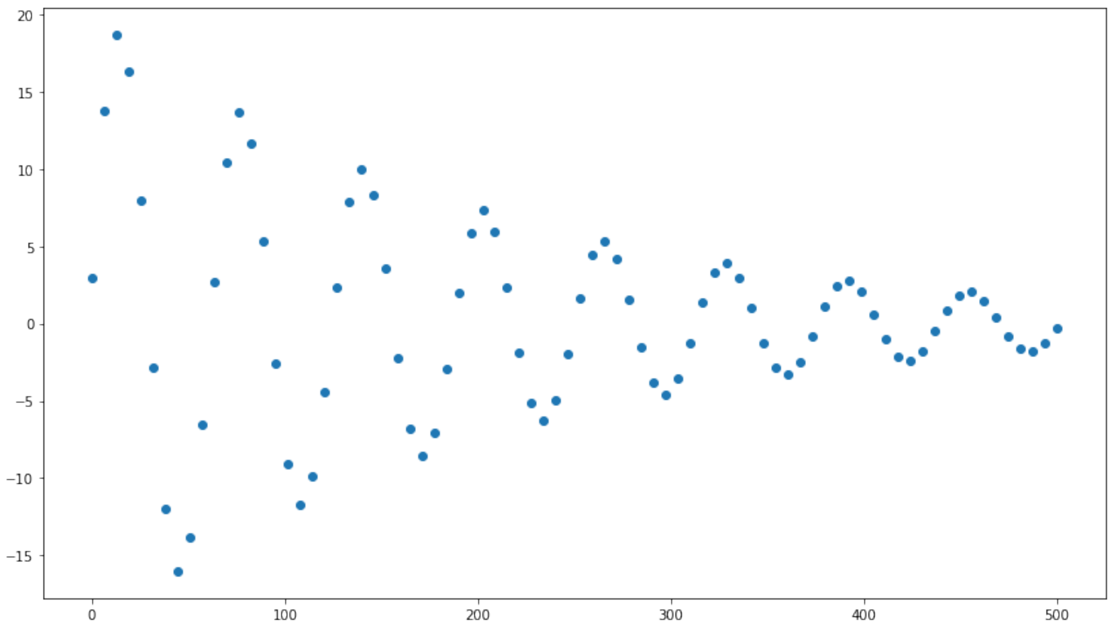
$$\begin{cases} y_i = \theta_1 x_{i1} + \theta_2 x_{i2} + \xi_i \\ \theta_1 = \frac{2+2\delta-\omega_0^2}{1+2\delta} \\ \theta_2 = -\frac{1}{1+2\delta} \\ E\xi = 0_n; \ cov(\xi) = \sigma^2 I_n \end{cases}$$

2.3 Генерування вибірки

In [53]:
```
config = OscillationModelConfig()
config.show()

plt.scatter(config.t, config.y)
plt.show()
print('Intermediate parameters values: θ_1 = {}\tθ_2 = {}'.format(*config.theta))
print('Regression model: y = ({}) * x1 + ({}) * x2'.format(*config.theta))
config.df.head(10)
```

Initial parameters: δ = 0.005
ω0^2 = 0.01
x0 = 5
x00 = 2

Noise generation: C = 2
Sample length: n = 80
Time starting from 0 to 500 with discretization frequency 6



Intermediate parameters values: θ_1 = 1.98019801980198 θ_2 = -0.9900990099009901
Regression model: y = (1.98019801980198) * x1 + (-0.9900990099009901) * x2

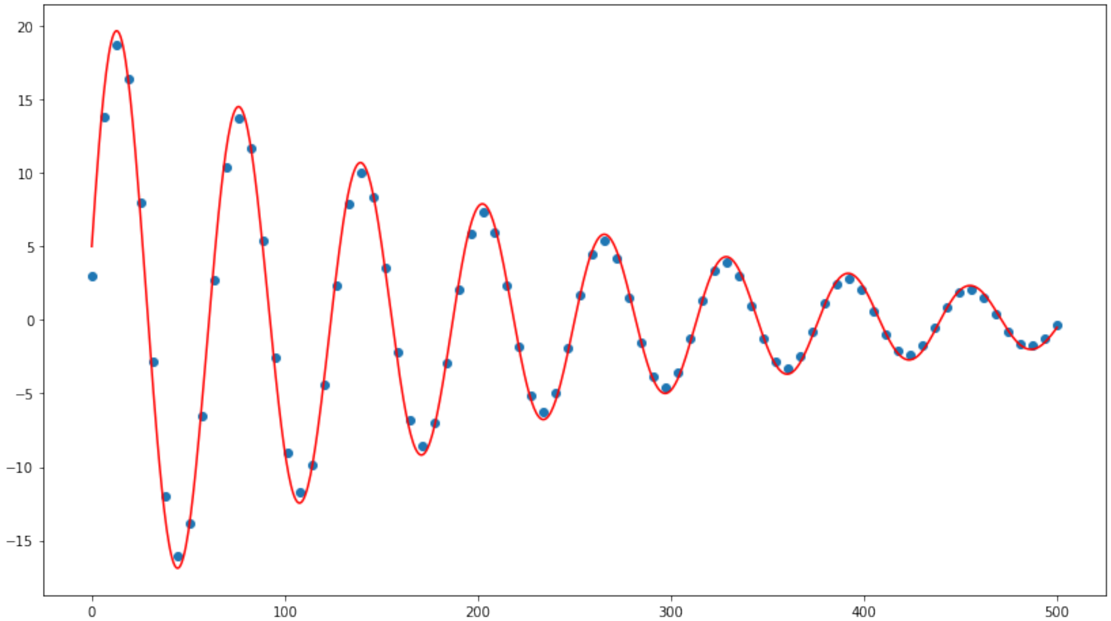
Out[53]:

	i	t	x(t)	x(t+1)	x(t+2)
0	1	0	5.000000	7.000000	2.97
1	2	6	15.514879	17.135487	13.76
2	3	12	19.560926	20.217068	18.72
3	4	18	16.016072	15.520598	16.35
4	5	25	6.676531	5.286058	7.99
5	6	31	-4.596447	-6.305073	-2.86
6	7	37	-13.452646	-14.818532	-11.97
7	8	44	-16.715857	-17.247299	-16.02
8	9	50	-13.504159	-13.052839	-13.82
9	10	56	-5.420258	-4.215867	-6.56

2.4 Робота алгоритму МНКО

```
In [43]: config.run_single_RMNK(verbose=True, deep_verbose=True)
```

```
Recurrent Least Squares Method
===============================================
        Step 1
===============================================
h_1:                [0]
eta_1:                3496.0743024036183
alpha_1:        [0]
beta_1:                3496.0743024036183
gamma_1:        3349.3120437748094
nu_1:                [0.95802084]
===============================================
> θ_1: [0.95802084]
> H_1_inv:
[[0.00028604]]
> RSS_1: 29.453858680093163
===============================================
        Step 2
===============================================
h_2:                [3609.56221018]
eta_2:                3756.7578218698554
alpha_2:        [1.03246153]
beta_2:                [[30.02371322]]
gamma_2:        3428.2987274811694
nu_2:                [[-0.99045368]]
===============================================
> θ_2: [ 1.98062616 -0.99045368]
> H_2_inv:
[[ 0.03579053 -0.0343882 ]
 [-0.0343882   0.03330701]]
> RSS_2: 0.0006414646992354278
=============================================

INTERMEDIATE PARAMETERS
True values:        θ_1 = 1.98019801980198        θ_2 = -0.9900990099009901
Estimates:        θ_1* = 1.9806261560117218        θ_2* = -0.9904536768605774

INITIAL PARAMETERS
True values:        δ = 0.005                        ω0^2 = 0.01
Estimates:        δ* = 0.004819166894145632        ω0^2* = 0.009922241775108143
```



2.5 Таблиця залежності оцінок від рівня шуму

Розміри вибірки: [30, 80, 150]
Округлення (кількість знаків після коми) до: [0, 2, 5]

In [44]:
```
forward_df = config.run_grid_RMNK(verbose=False)
forward_df
```

Out[44]:

	C	num_samples	θ_1	θ_1*	θ_2	θ_2*	δ	δ*	\
0	0.0	30.0	1.980198	2.092809	-0.990099	-1.100300	0.005	-0.045579	
1	0.0	80.0	1.980198	1.959643	-0.990099	-0.966105	0.005	0.017542	
2	0.0	150.0	1.980198	1.985392	-0.990099	-0.991847	0.005	0.004110	
3	2.0	30.0	1.980198	1.980402	-0.990099	-0.990257	0.005	0.004919	
4	2.0	80.0	1.980198	1.980626	-0.990099	-0.990454	0.005	0.004819	
5	2.0	150.0	1.980198	1.980007	-0.990099	-0.989910	0.005	0.005096	
6	5.0	30.0	1.980198	1.980197	-0.990099	-0.990098	0.005	0.005001	
7	5.0	80.0	1.980198	1.980199	-0.990099	-0.990100	0.005	0.005000	
8	5.0	150.0	1.980198	1.980197	-0.990099	-0.990098	0.005	0.005000	

	ω0_sqr	ω0_sqr*
0	0.01	0.006809
1	0.01	0.006689
2	0.01	0.006507
3	0.01	0.009952
4	0.01	0.009922
5	0.01	0.010005
6	0.01	0.010000

```
7     0.01   0.010000
8     0.01   0.010000
```

Розглянемо для апроксимації центральною різницею:

$$\Rightarrow \begin{cases} x(t+1) - 2x(t) + x(t-1) + \delta[x(t+1) - x(t-1)] + \omega_0^2 x(t) = 0 \\ x(0) = x_0 \\ x(1) = x(0) + x'(0) = x_0 + x'_0 \end{cases}$$

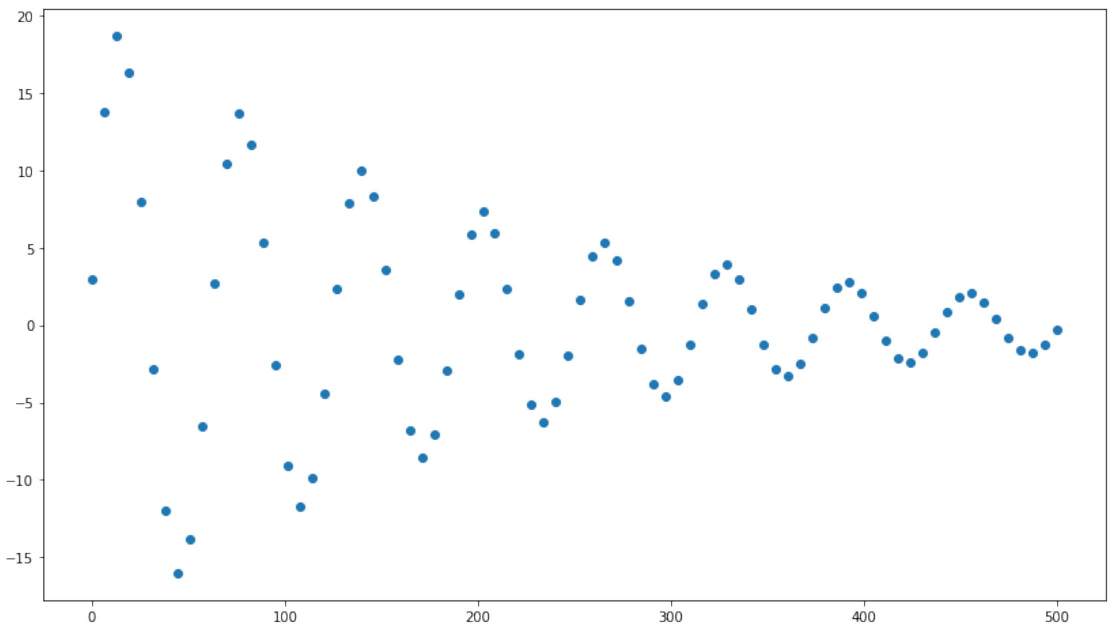
$$\iff \begin{cases} x(t+2) = \frac{2-\omega_0^2}{1+\delta} x(t+1) - \frac{1-\delta}{1+\delta} x(t) \\ x(0) = x_0 \\ x(1) = x_0 + x'_0 \end{cases}$$

$$\begin{cases} y_i = \theta_1 x_{i1} + \theta_2 x_{i2} + \xi_i \\ \theta_1 = \frac{2-\omega_0^2}{1+\delta} \\ \theta_2 = -\frac{1-\delta}{1+\delta} \\ E\xi = 0_n; \ \ cov(\xi) = \sigma^2 I_n \end{cases}$$

2.6 Генерування вибірки

In [50]:
```
config = OscillationModelConfig(difference='center')
config.show()

plt.scatter(config.t, config.y)
plt.show()
print('Intermediate parameters values: θ_1 = {}\tθ_2 = {}'.format(*config.theta))
print('Regression model: y = ({}) * x1 + ({}) * x2'.format(*config.theta))
config.df.head(10)
```

```
Initial parameters:        δ = 0.005
                       ω0^2 = 0.01
                       x0 = 5
                       x00 = 2

Noise generation: C = 2
Sample length: n = 80
Time starting from 0 to 500               with discretization frequency 6
```



```
Intermediate parameters values: θ_1 = 1.9800995024875623        θ_2 = -0.9900497512437811
Regression model: y = (1.9800995024875623) * x1 + (-0.9900497512437811) * x2
```

Out[50]:

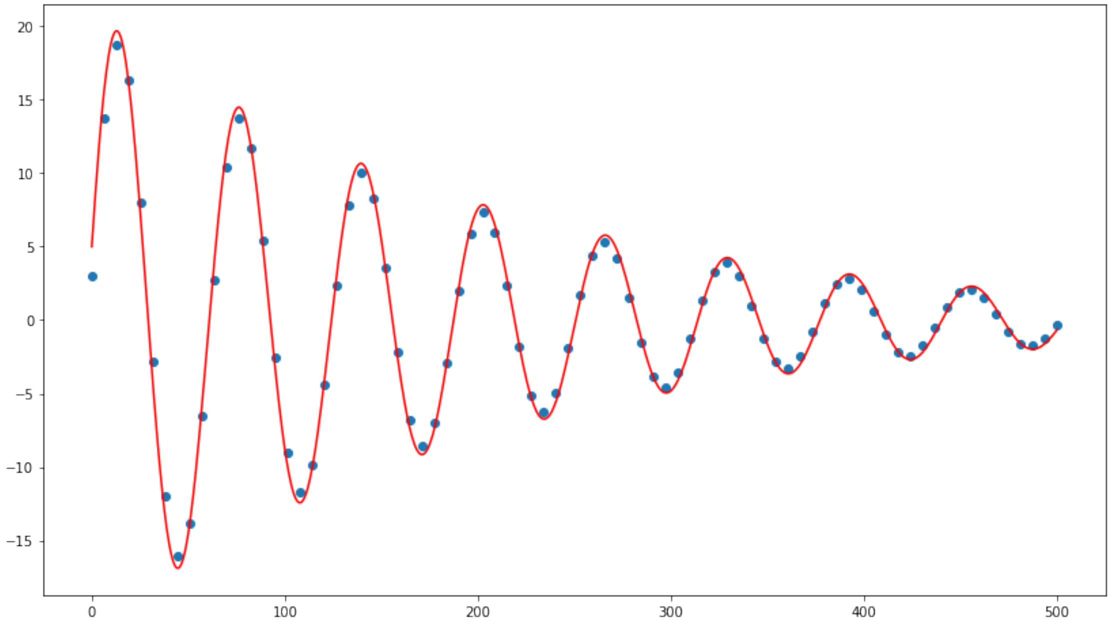
	i	t	x(t)	x(t+1)	x(t+2)
0	1	0	5.000000	7.000000	2.97
1	2	6	15.514879	17.135487	13.76
2	3	12	19.560926	20.217068	18.72
3	4	18	16.016072	15.520598	16.35
4	5	25	6.676531	5.286058	7.99
5	6	31	-4.596447	-6.305073	-2.86
6	7	37	-13.452646	-14.818532	-11.97
7	8	44	-16.715857	-17.247299	-16.02
8	9	50	-13.504159	-13.052839	-13.82
9	10	56	-5.420258	-4.215867	-6.56

2.7 Робота алгоритму МНКО

In [33]:
```
config.run_single_RMNK(verbose=True, deep_verbose=True)
```

```
Recurrent Least Squares Method
===============================================
        Step 1
===============================================
h_1:                    [0]
eta_1:                    3496.0743024036183
alpha_1:            [0]
```

```
beta_1:                3496.0743024036183
gamma_1:        3349.2782321079894
nu_1:                [0.95801117]
================================================
> θ_1: [0.95801117]
> H_1_inv:
[[0.00028604]]
> RSS_1: 29.447942916061038
================================================
        Step 2
================================================
h_2:                [3609.56221018]
eta_2:                3756.7578218698554
alpha_2:        [1.03246153]
beta_2:                [[30.02371322]]
gamma_2:        3428.266809326686
nu_2:                [[-0.99035405]]
================================================
> θ_2: [ 1.98051363 -0.99035405]
> H_2_inv:
[[ 0.03579053 -0.0343882 ]
 [-0.0343882   0.03330701]]
> RSS_2: 0.0006504761399703796
==============================================

INTERMEDIATE PARAMETERS
True values:        θ_1 = 1.9800995024875623        θ_2 = -0.9900497512437811
Estimates:        θ_1* = 1.980513626457381        θ_2* = -0.9903540525860997

INITIAL PARAMETERS
True values:        δ = 0.005                            ω0^2 = 0.01
Estimates:        δ* = 0.004846347513582927        ω0^2* = 0.009888116253420117
```



2.8 Таблиця залежності оцінок від рівня шуму

Розміри вибірки: [30, 80, 150]
Округлення (кількість знаків після коми) до: [0, 2, 5]

In [34]:
```
center_df = config.run_grid_RMNK(verbose=False)
center_df
```

Out[34]:

	C	num_samples	θ_1	θ_1*	θ_2	θ_2*	δ	δ*
0	0.0	30.0	1.9801	2.092809	-0.99005	-1.100300	0.005	-0.047755
1	0.0	80.0	1.9801	1.959643	-0.99005	-0.966105	0.005	0.017240
2	0.0	150.0	1.9801	1.985392	-0.99005	-0.991847	0.005	0.004093
3	2.0	30.0	1.9801	1.980023	-0.99005	-0.989908	0.005	0.005071
4	2.0	80.0	1.9801	1.980514	-0.99005	-0.990354	0.005	0.004846
5	2.0	150.0	1.9801	1.979936	-0.99005	-0.989861	0.005	0.005095
6	5.0	30.0	1.9801	1.980099	-0.99005	-0.990050	0.005	0.005000
7	5.0	80.0	1.9801	1.980099	-0.99005	-0.990050	0.005	0.005000
8	5.0	150.0	1.9801	1.980100	-0.99005	-0.990050	0.005	0.005000

\

	ω0_sqr	ω0_sqr*
0	0.01	0.007134
1	0.01	0.006574
2	0.01	0.006481
3	0.01	0.009935
4	0.01	0.009888
5	0.01	0.009975
6	0.01	0.010000

```
7    0.01  0.010000
8    0.01  0.010000
```

2.9 Об'єднана таблиця для двох видів апроксимацій

In [35]:
```
center_df['difference'] = 'center'
forward_df['difference'] = 'forward'
pd.concat([center_df, forward_df], axis=0).sort_values(by=['C',
                                                             'num_samples',
                                                             'difference'])
```

Out[35]:

	C	num_samples	θ_1	θ_1*	θ_2	θ_2*	δ	δ*
0	0.0	30.0	1.980100	2.092809	-0.990050	-1.100300	0.005	-0.047755
0	0.0	30.0	1.980198	2.092809	-0.990099	-1.100300	0.005	-0.045579
1	0.0	80.0	1.980100	1.959643	-0.990050	-0.966105	0.005	0.017240
1	0.0	80.0	1.980198	1.959643	-0.990099	-0.966105	0.005	0.017542
2	0.0	150.0	1.980100	1.985392	-0.990050	-0.991847	0.005	0.004093
2	0.0	150.0	1.980198	1.985392	-0.990099	-0.991847	0.005	0.004110
3	2.0	30.0	1.980100	1.980023	-0.990050	-0.989908	0.005	0.005071
3	2.0	30.0	1.980198	1.980402	-0.990099	-0.990257	0.005	0.004919
4	2.0	80.0	1.980100	1.980514	-0.990050	-0.990354	0.005	0.004846
4	2.0	80.0	1.980198	1.980626	-0.990099	-0.990454	0.005	0.004819
5	2.0	150.0	1.980100	1.979936	-0.990050	-0.989861	0.005	0.005095
5	2.0	150.0	1.980198	1.980007	-0.990099	-0.989910	0.005	0.005096
6	5.0	30.0	1.980100	1.980099	-0.990050	-0.990050	0.005	0.005000
6	5.0	30.0	1.980198	1.980197	-0.990099	-0.990098	0.005	0.005001
7	5.0	80.0	1.980100	1.980099	-0.990050	-0.990050	0.005	0.005000
7	5.0	80.0	1.980198	1.980199	-0.990099	-0.990100	0.005	0.005000
8	5.0	150.0	1.980100	1.980100	-0.990050	-0.990050	0.005	0.005000
8	5.0	150.0	1.980198	1.980197	-0.990099	-0.990098	0.005	0.005000

\

	ω0_sqr	ω0_sqr*	difference
0	0.01	0.007134	center
0	0.01	0.006809	forward
1	0.01	0.006574	center
1	0.01	0.006689	forward
2	0.01	0.006481	center
2	0.01	0.006507	forward
3	0.01	0.009935	center
3	0.01	0.009952	forward
4	0.01	0.009888	center
4	0.01	0.009922	forward
5	0.01	0.009975	center
5	0.01	0.010005	forward
6	0.01	0.010000	center
6	0.01	0.010000	forward
7	0.01	0.010000	center
7	0.01	0.010000	forward

```
8    0.01  0.010000     center
8    0.01  0.010000    forward
```

3 Дослідження закономірностей селекції оптимальних моделей за різними критеріями

Розглянемо настпуні критерії
$RSS(s)$
RSS(s) (як функція дискретного аргумента s) є строго спадною. Тобто при підвищенні складності (число аргументів або регресорів) моделі, наприклад, за рахунок шумів, значення цієї функції зменшиться. Тому її не можна використовувати в якості критерія оптимальності моделі.

Тож введемо такі два критерії, які будемо використовувати для селекції оптимальних моделей
$C\_p(s) = RSS(s) + 2s$ – спрощений критерій Меллоуза
$FPE(s) = \frac{n+s}{n-s}RSS(s)$ – критерій фінальної помилки передбачення Акаіке

3.1 Генерування вибірки

```
In [49]: config = ModelConfig()

         config.generate_noise_and_output()
         config.show()
```

```
Sample length: n = 10
Noise generation: σ = 0.01
X[:10]:
[[0.8256075  0.96538628 0.98689577 0.34211039 0.73125083]
 [1.99661548 1.10312016 0.47063081 0.6743167  0.32270018]
 [0.50521783 1.94305153 0.2321996  1.57475096 1.7151454 ]
 [1.45231104 0.84137512 1.6251626  1.55947169 1.65454129]
 [0.30540542 0.59276023 0.81413665 1.63787871 0.78084218]
 [0.35927949 1.59431491 0.59784151 0.89101901 0.11862545]
 [1.66405542 0.4946509  1.41582645 0.42319989 0.14946643]
 [0.25641774 0.39958445 0.1393855  1.55680484 1.58562454]
 [0.16804735 0.58465338 0.66805314 0.25416042 0.67679176]
 [0.52214368 0.30785052 0.81917318 1.88955457 1.13773672]]
y[:10]:
[ 1.53935532e+00  4.24655357e+00 -2.13879892e+00  4.28630293e+00
  5.42699716e-01 -1.53415571e+00  5.41475184e+00  1.17818043e-01
 -2.70343421e-04  1.78361145e+00]
```

3.2 Результати роботи МНКО для кожної складності моделі

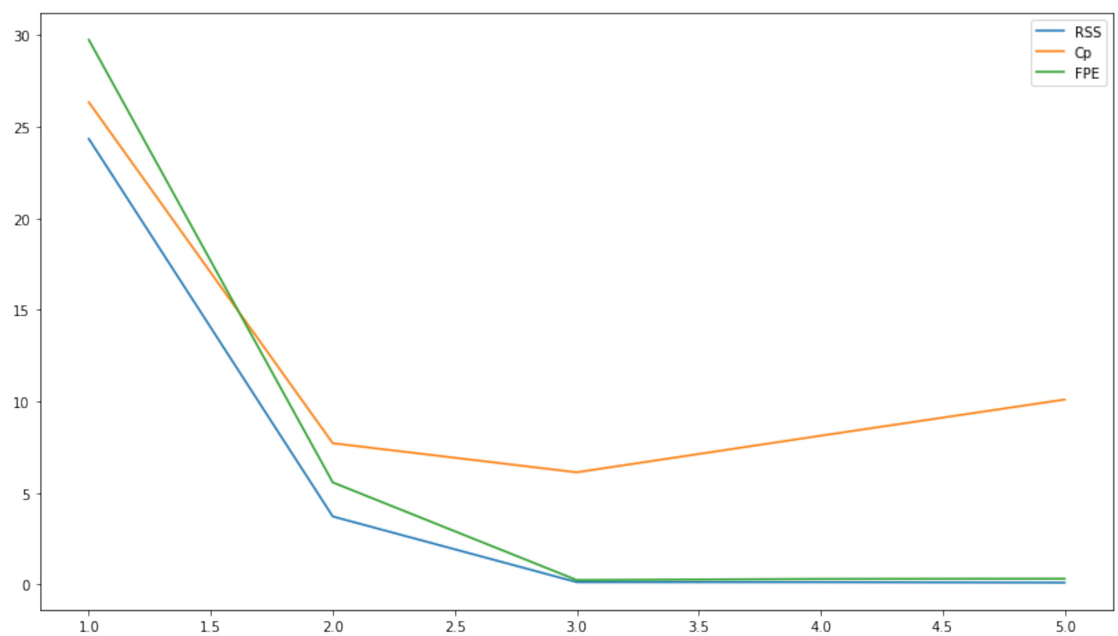
Розміри вибірки: [10, 30, 100]
Дисперсії: [0.1, 0.5, 1]

```
In [47]: config.run_grid_RMNK_model_selection()
```

```
--------------------------------------------------------------
                        SAMPLE #1
```

```
-------------------------------------------------------------
                CONFUGURATIONS & DATA
Sample length: n = 10
Noise generation: σ = 0.1
X[:10]:
[[1.74499128 1.36387688 0.97317271 0.91202505 0.82077243]
 [0.25024059 0.71549269 0.67900453 1.56559033 1.60674223]
 [0.78587479 1.4332687  0.72735019 0.80508076 1.44407207]
 [0.92653795 1.16807282 1.81264013 1.87331243 1.26766465]
 [1.37688235 1.55669921 0.41909229 1.71869654 0.44395493]
 [0.06463555 1.91239844 1.86896445 1.56800728 1.53465654]
 [1.6610133  0.98512173 1.79487906 1.51341071 1.51509083]
 [1.41194792 1.36280981 0.38997994 1.53333186 0.61675108]
 [0.03745953 1.65573554 0.41078422 1.58074669 1.24010989]
 [0.50865276 1.78581802 0.58324822 1.6045627  1.52076612]]
y[:10]:
[ 3.48591953 -0.0823742   0.35078648  2.32588991  1.68130005 -1.81033725
  4.69829369  1.89906112 -2.76304196 -1.67492179]

                RLSM ITERATIONS
===============================================
        Step 1
===============================================
> θ_1: [1.75888448]
> H_1_inv:
[[0.08698404]]
> RSS_1: 24.34948235855881
===============================================
        Step 2
===============================================
> θ_2: [ 3.28244712 -1.51445048]
> H_2_inv:
[[ 0.19950471 -0.1118477 ]
 [-0.1118477   0.11117876]]
> RSS_2: 3.720000469205509
===============================================
        Step 3
===============================================
> θ_3: [ 3.04616316 -1.98965346  0.9481994 ]
> H_3_inv:
[[ 0.21506781 -0.08054793 -0.0624542 ]
 [-0.08054793  0.17412736 -0.1256049 ]
 [-0.0624542  -0.1256049   0.25062656]]
> RSS_3: 0.13266281282578296
===============================================
        Step 4
===============================================
> θ_4: [ 3.04862082 -1.97302879  0.95465723 -0.02234266]
```

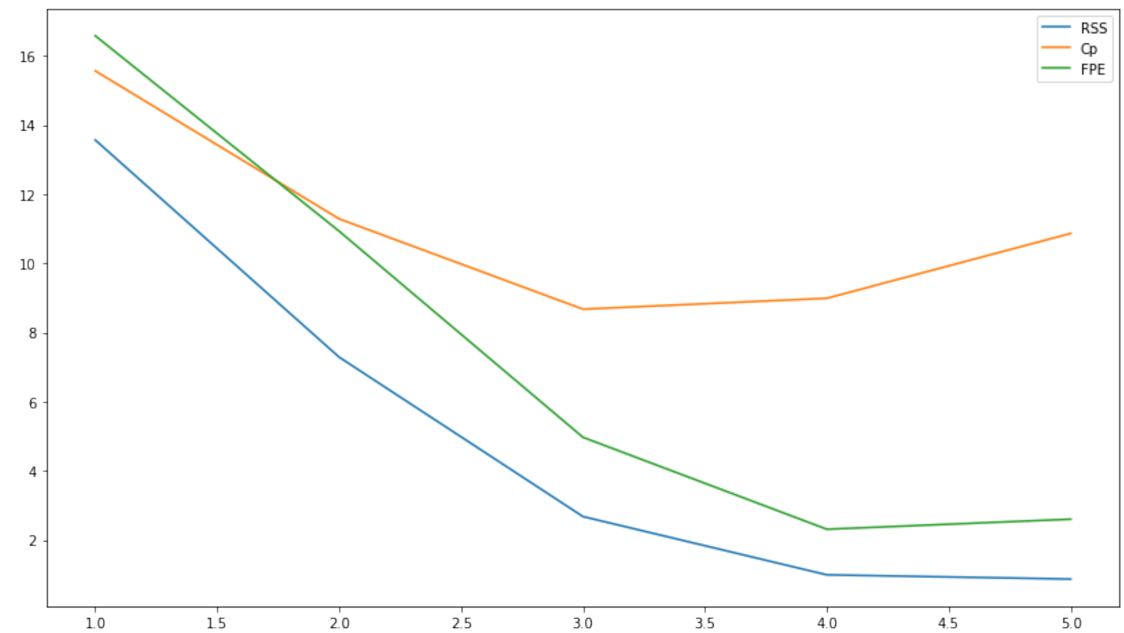
```
> H_4_inv:
[[ 0.2217556  -0.03530892 -0.04488117 -0.06079878]
 [-0.03530892  0.48014312 -0.00673353 -0.41126856]
 [-0.04488117 -0.00673353  0.29680197 -0.15975667]
 [-0.06079878 -0.41126856 -0.15975667  0.55272261]]
> RSS_4: 0.1317596574980914
===============================================
        Step 5
===============================================
> θ_5: [ 3.01994177 -1.93297116  1.01152629  0.02558417 -0.13454104]
> H_5_inv:
[[ 0.25394333 -0.08026732 -0.10870776 -0.11458912  0.15100118]
 [-0.08026732  0.54293903  0.08241662 -0.33613661 -0.21091176]
 [-0.10870776  0.08241662  0.42336674 -0.05309326 -0.2994274 ]
 [-0.11458912 -0.33613661 -0.05309326  0.642614   -0.2523447 ]
 [ 0.15100118 -0.21091176 -0.2994274  -0.2523447   0.70838647]]
> RSS_5: 0.10620681153214041

                        RESULTS

PARAMETERS
True values:        θ: [ 3 -2  1  0  0]
Estimates:         θ*: [ 3.01994177 -1.93297116  1.01152629  0.02558417 -0.13454104]
```




```
     s        RSS         Cp        FPE
0  1.0  24.349482  26.349482  29.760478
```

```
1  2.0   3.720000   7.720000   5.580001
2  3.0   0.132663   6.132663   0.246374
3  4.0   0.131760   8.131760   0.307439
4  5.0   0.106207  10.106207   0.318620
s* by Cp:  3
s* by FPE: 3

--------------------------------------------------------------
                        SAMPLE #2
--------------------------------------------------------------
                CONFUGURATIONS & DATA
Sample length: n = 10
Noise generation: σ = 0.5
X[:10]:
[[0.33492291 1.08563438 1.68194974 0.29033474 1.78802504]
 [0.2159277  0.81205598 0.53006163 1.53257466 1.88128202]
 [0.39104558 0.91036983 0.36540506 1.12260893 1.52395021]
 [1.22387016 0.962584   0.69919347 1.47873658 1.06353278]
 [1.98865213 1.88135532 1.62673745 1.07394478 0.38639772]
 [0.21440056 0.00601832 0.63623142 0.72337703 0.98327312]
 [0.71403784 1.08255883 0.15211974 0.74150085 0.19634336]
 [1.82350366 1.98757221 0.10940998 1.41456159 1.19650657]
 [1.00166412 1.39490568 0.49255994 0.91280459 1.37797455]
 [0.7371997  1.43040815 0.65326362 1.36619623 1.98564815]]
y[:10]:
[ 0.12993849 -0.54116239  0.15820152  3.12991112  3.43588371  2.54449243
 -0.1174249   1.47193416 -0.00622622  0.01589706]

                RLSM ITERATIONS
===============================================
        Step 1
===============================================
> θ_1: [1.23329701]
> H_1_inv:
[[0.08934883]]
> RSS_1: 13.568114398236023
===============================================
        Step 2
===============================================
> θ_2: [ 3.28209475 -1.8105654 ]
> H_2_inv:
[[ 0.75766193 -0.59060225]
 [-0.59060225  0.52192756]]
> RSS_2: 7.287267551556573
===============================================
        Step 3
===============================================
> θ_3: [ 3.42991111 -2.54163368  1.20115625]
```

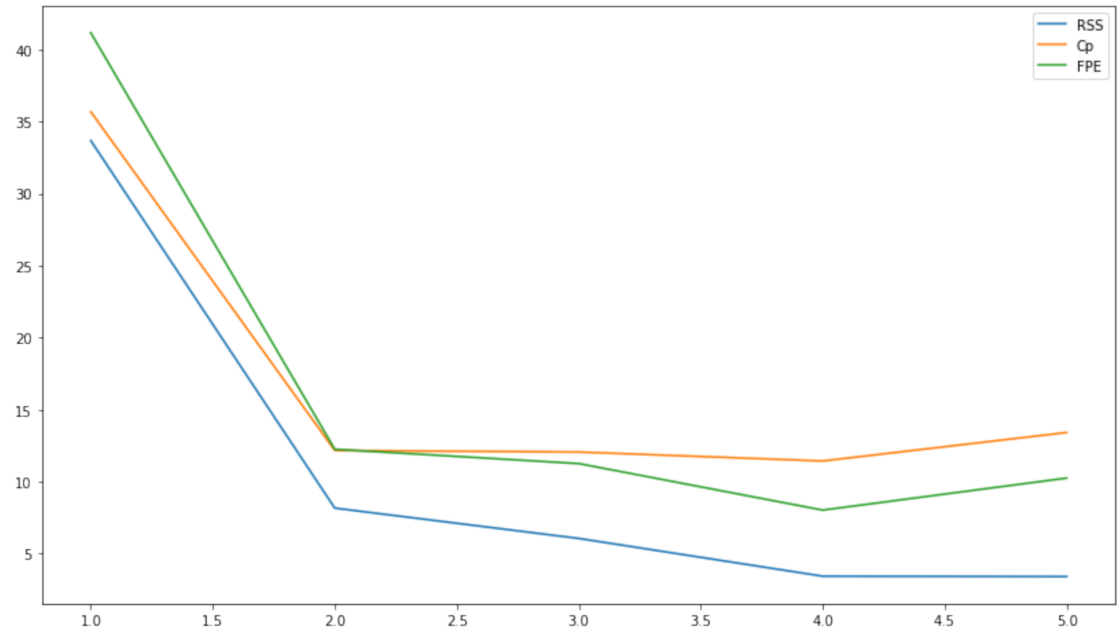
```
> H_3_inv:
[[ 0.76239897 -0.61403067  0.03849325]
 [-0.61403067  0.63779955 -0.19037944]
 [ 0.03849325 -0.19037944  0.3127963 ]]
> RSS_3: 2.674756662545904
===============================================
        Step 4
===============================================
> θ_4: [ 3.45280477 -3.20182555  1.21173671  0.80394412]
> H_4_inv:
[[ 0.7627099  -0.622997    0.03863695  0.01091868]
 [-0.622997    0.89636443 -0.19452329 -0.31486561]
 [ 0.03863695 -0.19452329  0.31286271  0.00504615]
 [ 0.01091868 -0.31486561  0.00504615  0.38342543]]
> RSS_4: 0.989093427444625
===============================================
        Step 5
===============================================
> θ_5: [ 3.89786176 -3.50690387  1.06884173  0.47639856  0.33843127]
> H_5_inv:
[[ 2.37527968 -1.7283837  -0.47911276 -1.17587332  1.22623406]
 [-1.7283837   1.65408652  0.16038454  0.49865831 -0.84056072]
 [-0.47911276  0.16038454  0.47909723  0.38609088 -0.39370844]
 [-1.17587332  0.49865831  0.38609088  1.25686066 -0.90246313]
 [ 1.22623406 -0.84056072 -0.39370844 -0.90246313  0.93245575]]
> RSS_5: 0.8662610858480779

                        RESULTS

PARAMETERS
True values:        θ: [ 3 -2  1  0  0]
Estimates:         θ*: [ 3.89786176 -3.50690387  1.06884173  0.47639856  0.33843127]
```




```
     s        RSS         Cp        FPE
0  1.0  13.568114  15.568114  16.583251
1  2.0   7.287268  11.287268  10.930901
2  3.0   2.674757   8.674757   4.967405
3  4.0   0.989093   8.989093   2.307885
4  5.0   0.866261  10.866261   2.598783
s* by Cp:  3
s* by FPE: 4

--------------------------------------------------------------
                        SAMPLE #3
--------------------------------------------------------------
                CONFUGURATIONS & DATA
Sample length: n = 10
Noise generation: σ = 1
X[:10]:
[[0.14696984 1.64225686 0.81527383 0.5052387  1.74258384]
 [0.54192376 1.77129056 0.69364413 1.91389457 1.11587571]
 [1.46345848 0.70656644 0.33467826 1.80626046 1.15677286]
 [0.15388681 1.80944155 0.56569552 0.16313333 1.6271434 ]
 [1.55113002 0.30217504 0.42268223 1.36097742 0.80769278]
 [0.90139168 1.54823504 0.90174186 0.59907987 0.66142068]
 [0.87465941 1.822172   1.23629605 1.23502435 1.11140353]
 [0.06916039 0.20130832 1.52169335 0.59426913 0.98419139]
 [1.42907423 0.32435217 1.16714399 1.15158231 1.01927989]
 [1.51590501 1.6155413  1.6922702  1.25023024 0.92448734]]
```

```
y[:10]:
[-1.35546965 -2.93986301  2.09830818 -2.84221758  2.75526638  0.02831423
  0.59059142 -0.01914911  4.41352945  2.51628973]

                RLSM ITERATIONS
===============================================
        Step 1
===============================================
> θ_1: [1.45961255]
> H_1_inv:
[[0.09251363]]
> RSS_1: 33.684904369399405
===============================================
        Step 2
===============================================
> θ_2: [ 2.72218064 -1.53371291]
> H_2_inv:
[[ 0.15498937 -0.07589281]
 [-0.07589281  0.09219129]]
> RSS_2: 8.169747929934228
===============================================
        Step 3
===============================================
> θ_3: [ 2.38566268 -1.88274339  0.847052  ]
> H_3_inv:
[[ 0.20876806 -0.02011451 -0.13536676]
 [-0.02011451  0.15004357 -0.14040001]
 [-0.13536676 -0.14040001  0.34073274]]
> RSS_3: 6.064000510738701
===============================================
        Step 4
===============================================
> θ_4: [ 3.27963753 -1.61892154  0.89849639 -1.1196245 ]
> H_4_inv:
[[ 0.5134597   0.06980336 -0.11783308 -0.38159936]
 [ 0.06980336  0.17657932 -0.13522563 -0.11261419]
 [-0.11783308 -0.13522563  0.34174173 -0.0219594 ]
 [-0.38159936 -0.11261419 -0.0219594   0.47791947]]
> RSS_4: 3.44105023141445
===============================================
        Step 5
===============================================
> θ_5: [ 3.27075458 -1.57050896  0.93236985 -1.08868603 -0.11081723]
> H_5_inv:
[[ 0.51689518  0.0510798  -0.13093363 -0.39356481  0.04285855]
 [ 0.0510798   0.27862387 -0.0638268  -0.04740175 -0.23358169]
 [-0.13093363 -0.0638268   0.39169827  0.02366863 -0.16343312]
 [-0.39356481 -0.04740175  0.02366863  0.51959403 -0.14927237]
```

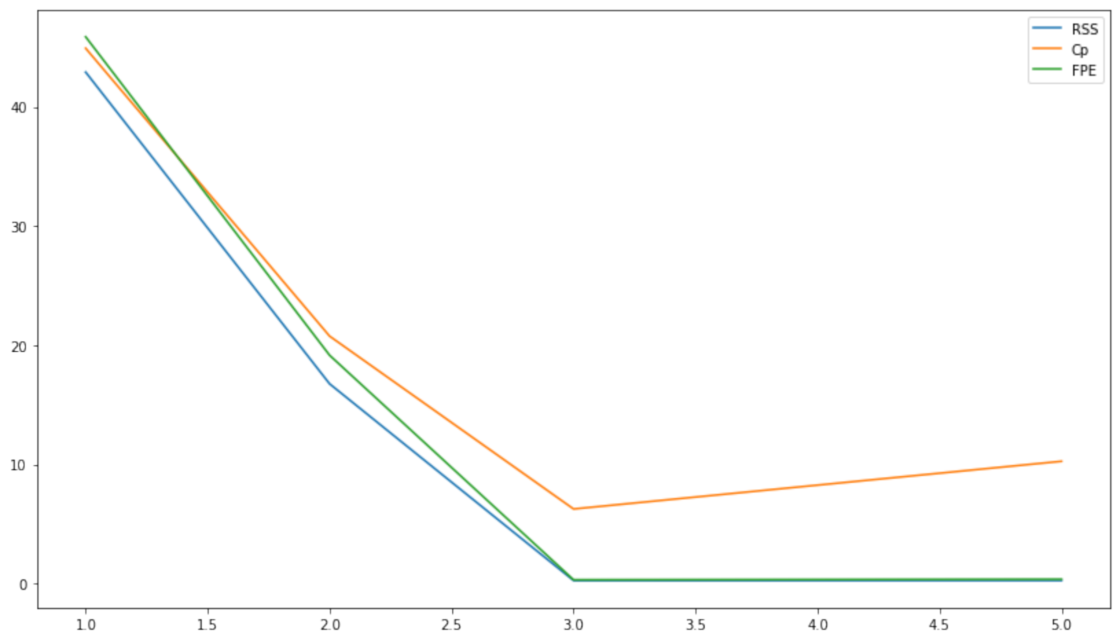
```
 [ 0.04285855 -0.23358169 -0.16343312 -0.14927237  0.53467246]]
> RSS_5: 3.4180820437301587

                        RESULTS

PARAMETERS
True values:        θ: [ 3 -2  1  0  0]
Estimates:        θ*: [ 3.27075458 -1.57050896  0.93236985 -1.08868603 -0.11081723]
```




```
     s        RSS         Cp        FPE
0  1.0  33.684904  35.684904  41.170439
1  2.0   8.169748  12.169748  12.254622
2  3.0   6.064001  12.064001  11.261715
3  4.0   3.441050  11.441050   8.029117
4  5.0   3.418082  13.418082  10.254246
s* by Cp:  4
s* by FPE: 4

--------------------------------------------------------------
                        SAMPLE #4
--------------------------------------------------------------
                CONFUGURATIONS & DATA
Sample length: n = 30
Noise generation: σ = 0.1
X[:10]:
[[0.1487495  0.39253727 0.94876846 0.84020432 1.02141893]
```

```
 [1.03237769 1.14966652 1.10723391 0.16075337 0.2289113 ]
 [1.55214301 0.974513   0.66598849 1.86118306 0.22885745]
 [1.65397925 1.45360209 0.83021713 0.78597958 0.43489001]
 [0.71856424 1.28416975 0.73402926 1.3018575  0.94595914]
 [0.80133343 0.43329017 1.09414303 0.98373604 0.62182217]
 [0.10070248 1.77904068 1.78430279 0.09961824 0.16469113]
 [0.59460095 0.28739308 1.33864153 1.84248439 0.03749578]
 [1.48062364 1.05328261 1.36069382 0.56938633 1.91588621]
 [0.5420078  0.63760454 1.89696076 0.86427491 1.31056505]]
y[:10]:
[ 0.49487737  1.96984176  3.44538533  3.02811936  0.23413413  2.54153192
 -1.56549911  2.5680147   3.57818677  2.40551953]

                RLSM ITERATIONS
================================================
        Step 1
================================================
> θ_1: [2.16725506]
> H_1_inv:
[[0.02894723]]
> RSS_1: 42.911961631672995
================================================
        Step 2
================================================
> θ_2: [ 3.27016325 -1.34135586]
> H_2_inv:
[[ 0.07547652 -0.05658888]
 [-0.05658888  0.06882334]]
> RSS_2: 16.769149956764263
================================================
        Step 3
================================================
> θ_3: [ 3.02755153 -2.03001379  1.00549493]
> H_3_inv:
[[ 0.07904502 -0.04645962 -0.01478951]
 [-0.04645962  0.09757541 -0.0419803 ]
 [-0.01478951 -0.0419803   0.06129455]]
> RSS_3: 0.2746963575911856
================================================
        Step 4
================================================
> θ_4: [ 3.02592015e+00 -2.02963720e+00  1.00441252e+00  2.61162670e-03]
> H_4_inv:
[[ 0.11719665 -0.05526666  0.01052376 -0.06107561]
 [-0.05526666  0.09960845 -0.04782369  0.01409888]
 [ 0.01052376 -0.04782369  0.07808968 -0.04052313]
 [-0.06107561  0.01409888 -0.04052313  0.09777381]]
> RSS_4: 0.2746265986852164
```

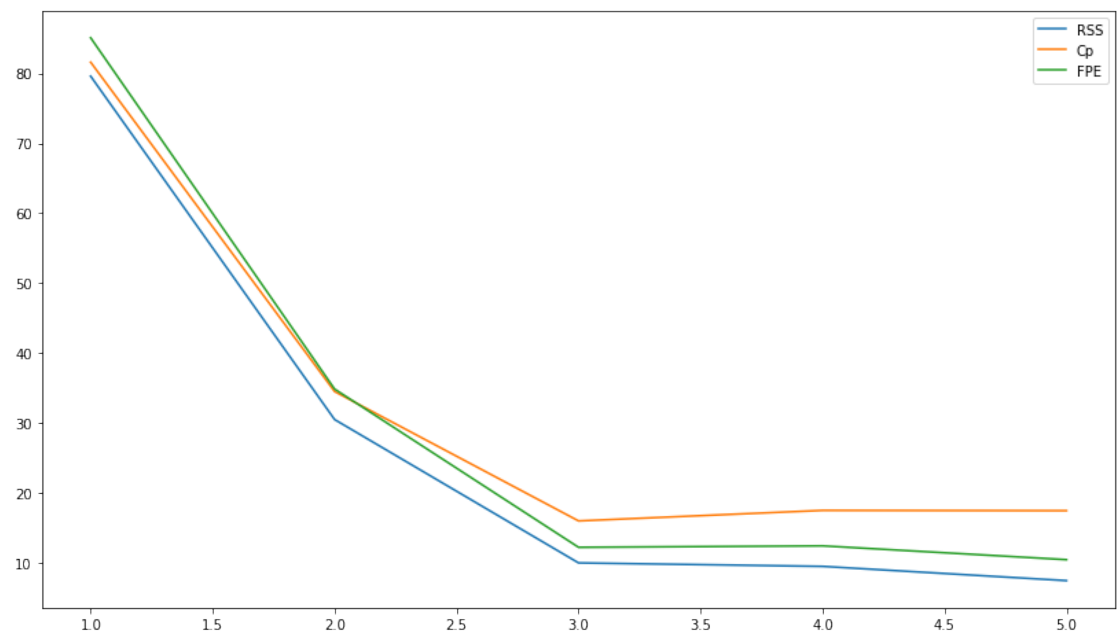
```
===============================================
        Step 5
===============================================
> θ_5: [ 3.02427775e+00 -2.02995134e+00  1.00108958e+00  2.33265179e-03
  6.88853109e-03]
> H_5_inv:
[[ 0.12210533 -0.05432778  0.0204551  -0.06024183 -0.02058789]
 [-0.05432778  0.09978804 -0.04592412  0.01425836 -0.00393785]
 [ 0.0204551  -0.04592412  0.09818297 -0.03883621 -0.04165386]
 [-0.06024183  0.01425836 -0.03883621  0.09791543 -0.00349702]
 [-0.02058789 -0.00393785 -0.04165386 -0.00349702  0.0863494 ]]
> RSS_5: 0.2740770655149688

                        RESULTS

PARAMETERS
True values:         θ: [ 3 -2  1  0  0]
Estimates:         θ*: [ 3.02427775e+00 -2.02995134e+00  1.00108958e+00  2.33265179e-03
  6.88853109e-03]
```



	s	RSS	Cp	FPE
0	1.0	42.911962	44.911962	45.871407
1	2.0	16.769150	20.769150	19.164743
2	3.0	0.274696	6.274696	0.335740
3	4.0	0.274627	8.274627	0.359127
4	5.0	0.274077	10.274077	0.383708

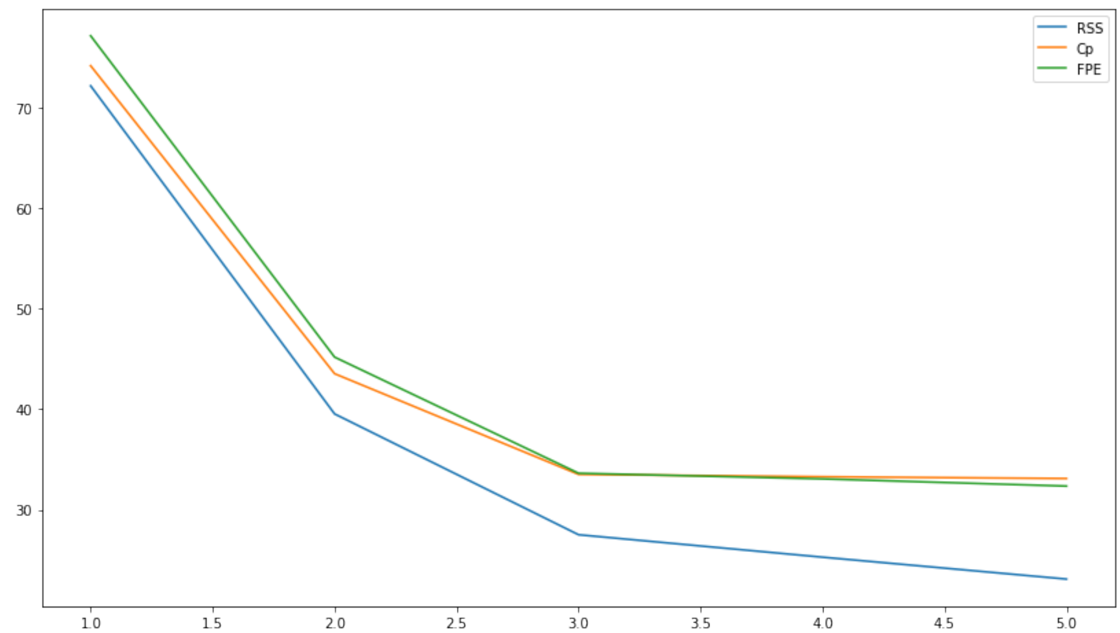
```
s* by Cp:  3
s* by FPE: 3

--------------------------------------------------------------
                        SAMPLE #5
--------------------------------------------------------------
                CONFUGURATIONS & DATA
Sample length: n = 30
Noise generation: σ = 0.5
X[:10]:
[[0.72876001 1.31939899 0.25131668 1.02587873 1.08236842]
 [1.6790284  1.49552647 1.82319386 0.0662499  1.77054865]
 [0.24189569 0.06920569 0.11676017 0.36094835 1.96565189]
 [0.03083241 0.48892639 1.62124848 0.57922692 1.94939807]
 [0.8968224  1.35714991 1.36761615 1.21381602 1.48231496]
 [0.62585452 0.74419252 1.41248933 1.82981646 0.6278084 ]
 [1.87687817 0.45745942 0.14204097 1.2282496  1.40724115]
 [0.74047996 1.79276388 0.18282914 0.33575151 1.06089481]
 [0.34882251 0.07366257 1.75406866 1.3220241  0.92124879]
 [0.12071026 1.89096654 1.05274132 0.58485607 1.80056392]]
y[:10]:
[-0.72240974  3.2231523   0.62399833  0.62839286  0.97950829  2.15486126
  6.06103113 -1.59945271  3.02280988 -3.09663769]

                RLSM ITERATIONS
===============================================
        Step 1
===============================================
> θ_1: [2.45358863]
> H_1_inv:
[[0.02866141]]
> RSS_1: 79.59086964249369
===============================================
        Step 2
===============================================
> θ_2: [ 3.90447742 -1.86582164]
> H_2_inv:
[[ 0.07155604 -0.05516186]
 [-0.05516186  0.07093734]]
> RSS_2: 30.515298339401106
===============================================
        Step 3
===============================================
> θ_3: [ 3.24605609 -2.29954684  1.17928989]
> H_3_inv:
[[ 0.09271279 -0.04122516 -0.03789359]
 [-0.04122516  0.08011793 -0.02496184]
 [-0.03789359 -0.02496184  0.06787072]]
```

```
> RSS_3: 10.024508583050252
===============================================
        Step 4
===============================================
> θ_4: [ 3.12715944 -2.32002488  1.12147583  0.21910323]
> H_4_inv:
[[ 0.12126723 -0.03630712 -0.02400886 -0.05262023]
 [-0.03630712  0.08096498 -0.02257041 -0.00906299]
 [-0.02400886 -0.02257041  0.07462223 -0.02558683]
 [-0.05262023 -0.00906299 -0.02558683  0.09696876]]
> RSS_4: 9.529439633710565
===============================================
        Step 5
===============================================
> θ_5: [ 3.1684059  -2.2741488   1.27557734  0.30493557 -0.35785053]
> H_5_inv:
[[ 0.12210233 -0.03537828 -0.02088882 -0.05088241 -0.00724528]
 [-0.03537828  0.08199808 -0.01910017 -0.00713011 -0.00805851]
 [-0.02088882 -0.01910017  0.08627907 -0.01909414 -0.02706921]
 [-0.05088241 -0.00713011 -0.01909414  0.1005851  -0.01507716]
 [-0.00724528 -0.00805851 -0.02706921 -0.01507716  0.06285941]]
> RSS_5: 7.492242706240905

                        RESULTS

PARAMETERS
True values:        θ: [ 3 -2  1  0  0]
Estimates:         θ*: [ 3.1684059  -2.2741488   1.27557734  0.30493557 -0.35785053]
```



```
     s        RSS         Cp        FPE
0  1.0  79.590870  81.590870  85.079895
1  2.0  30.515298  34.515298  34.874627
2  3.0  10.024509  16.024509  12.252177
3  4.0   9.529440  17.529440  12.461575
4  5.0   7.492243  17.492243  10.489140
s* by Cp:  3
s* by FPE: 5

--------------------------------------------------------------
                        SAMPLE #6
--------------------------------------------------------------
                CONFUGURATIONS & DATA
Sample length: n = 30
Noise generation: σ = 1
X[:10]:
[[1.4657479  1.0203736  0.27119653 0.76962363 0.12997583]
 [0.12688183 0.14651378 1.5920262  0.65060901 0.41573806]
 [0.39620312 1.51504911 1.95252342 1.38204029 0.33692889]
 [1.30746237 1.12286779 0.00974265 0.85186273 1.39939848]
 [0.36114552 0.02900568 0.24237525 0.37378936 1.51242997]
 [0.14718973 0.55690455 0.63306301 0.6750009  1.61196702]
 [0.561286   1.13888002 0.81831695 0.72452641 1.3125748 ]
 [1.6350668  1.38261912 0.66248552 1.09468795 0.14239637]
 [1.96966048 1.37709031 1.24864897 0.07716315 1.53212933]
 [1.67362771 1.12044411 1.36239841 1.73962138 0.13950526]]
y[:10]:
[ 2.23790939  3.28962769  0.66891754  1.28786737  1.87640715 -1.05213968
 -1.13407385  3.14801966  4.0742737   2.62623902]

                RLSM ITERATIONS
===============================================
        Step 1
===============================================
> θ_1: [1.95439287]
> H_1_inv:
[[0.02786828]]
> RSS_1: 72.16315009756227
===============================================
        Step 2
===============================================
> θ_2: [ 3.18672401 -1.67285668]
> H_2_inv:
[[ 0.0744196  -0.06319218]
 [-0.06319218  0.0857817 ]]
```

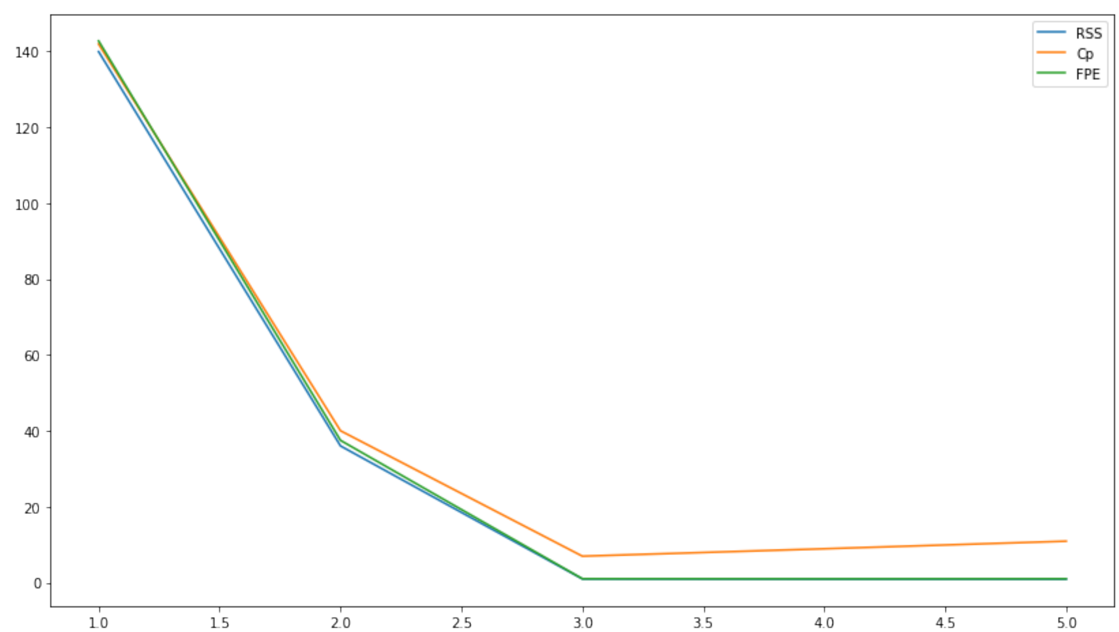
```
> RSS_2: 39.54022982150166
===============================================
        Step 3
===============================================
> θ_3: [ 2.84257131 -2.09676554  0.88739358]
> H_3_inv:
[[ 0.0842777  -0.05104949 -0.02541901]
 [-0.05104949  0.10073841 -0.03130977]
 [-0.02541901 -0.03130977  0.0655426 ]]
> RSS_3: 27.525641486335502
===============================================
        Step 4
===============================================
> θ_4: [ 2.82363745 -2.44083945  0.75675816  0.48181414]
> H_4_inv:
[[ 0.0844384  -0.04812916 -0.02431024 -0.0040894 ]
 [-0.04812916  0.15380786 -0.01116076 -0.0743143 ]
 [-0.02431024 -0.01116076  0.07319263 -0.0282151 ]
 [-0.0040894  -0.0743143  -0.0282151   0.10406392]]
> RSS_4: 25.294850332210924
===============================================
        Step 5
===============================================
> θ_5: [ 2.80245265 -2.3222004   0.85858036  0.60379904 -0.37542031]
> H_5_inv:
[[ 0.08464449 -0.04928331 -0.02530078 -0.0052761   0.00365216]
 [-0.04928331  0.16027129 -0.0056135  -0.06766858 -0.02045284]
 [-0.02530078 -0.0056135   0.07795357 -0.0225114  -0.01755369]
 [-0.0052761  -0.06766858 -0.0225114   0.11089706 -0.02102965]
 [ 0.00365216 -0.02045284 -0.01755369 -0.02102965  0.06472077]]
> RSS_5: 23.11718154095141

                        RESULTS

PARAMETERS
True values:        θ: [ 3 -2  1  0  0]
Estimates:         θ*: [ 2.80245265 -2.3222004   0.85858036  0.60379904 -0.37542031]
```




```
     s        RSS         Cp        FPE
0  1.0  72.163150  74.163150  77.139919
1  2.0  39.540230  43.540230  45.188834
2  3.0  27.525641  33.525641  33.642451
3  4.0  25.294850  33.294850  33.077881
4  5.0  23.117182  33.117182  32.364054
s* by Cp:  5
s* by FPE: 5

--------------------------------------------------------------
                        SAMPLE #7
--------------------------------------------------------------
                CONFUGURATIONS & DATA
Sample length: n = 100
Noise generation: σ = 0.1
X[:10]:
[[1.98504115e+00 1.37682418e-01 1.29076862e+00 1.61146647e+00
  1.55773586e+00]
 [1.68041062e-01 1.02261171e+00 6.52525865e-01 1.79307661e+00
  7.81053943e-01]
 [1.44758733e+00 5.79668061e-01 9.21768252e-01 1.75864705e+00
  1.40988989e+00]
 [1.60422920e+00 1.86154158e+00 6.18633827e-01 1.38958277e-03
  8.07922782e-01]
 [1.63708292e+00 5.86510715e-01 1.06373771e+00 1.48776152e+00
  1.20121708e-01]
```

```
 [9.66599051e-01 4.68436160e-01 2.89297431e-01 5.11833060e-01
  1.31440094e+00]
 [4.81831012e-02 1.34767400e+00 7.61509565e-01 6.08325805e-01
  9.66134925e-02]
 [1.71562470e+00 1.83542193e+00 7.19104070e-01 1.13391063e+00
  9.80065132e-01]
 [1.08834028e+00 1.29009480e+00 1.92487873e+00 7.87675476e-01
  1.24867257e+00]
 [1.90783230e+00 1.57581923e+00 1.20548848e+00 1.24757961e+00
  1.16952879e+00]]
y[:10]:
[ 7.0468222  -0.91133898  3.94671901  1.74501699  4.79035547  2.10071719
 -1.80283288  2.28376996  2.54127111  3.74845277]

                RLSM ITERATIONS
===============================================
        Step 1
===============================================
> θ_1: [2.20023113]
> H_1_inv:
[[0.00827928]]
> RSS_1: 139.90400552191068
===============================================
        Step 2
===============================================
> θ_2: [ 3.27649717 -1.32743988]
> H_2_inv:
[[ 0.0194286  -0.0137513 ]
 [-0.0137513   0.01696051]]
> RSS_2: 36.009931867380544
===============================================
        Step 3
===============================================
> θ_3: [ 2.9906726  -1.96198542  0.97913501]
> H_3_inv:
[[ 0.02175901 -0.00857766 -0.00798317]
 [-0.00857766  0.02844625 -0.01772306]
 [-0.00798317 -0.01772306  0.02734756]]
> RSS_3: 0.9535903845829736
===============================================
        Step 4
===============================================
> θ_4: [ 3.00435631 -1.95577587  0.9914165  -0.03379706]
> H_4_inv:
[[ 0.0257926  -0.00674726 -0.00436293 -0.00996246]
 [-0.00674726  0.02927688 -0.01608023 -0.00452088]
 [-0.00436293 -0.01608023  0.03059683 -0.00894156]
 [-0.00996246 -0.00452088 -0.00894156  0.02460603]]
```

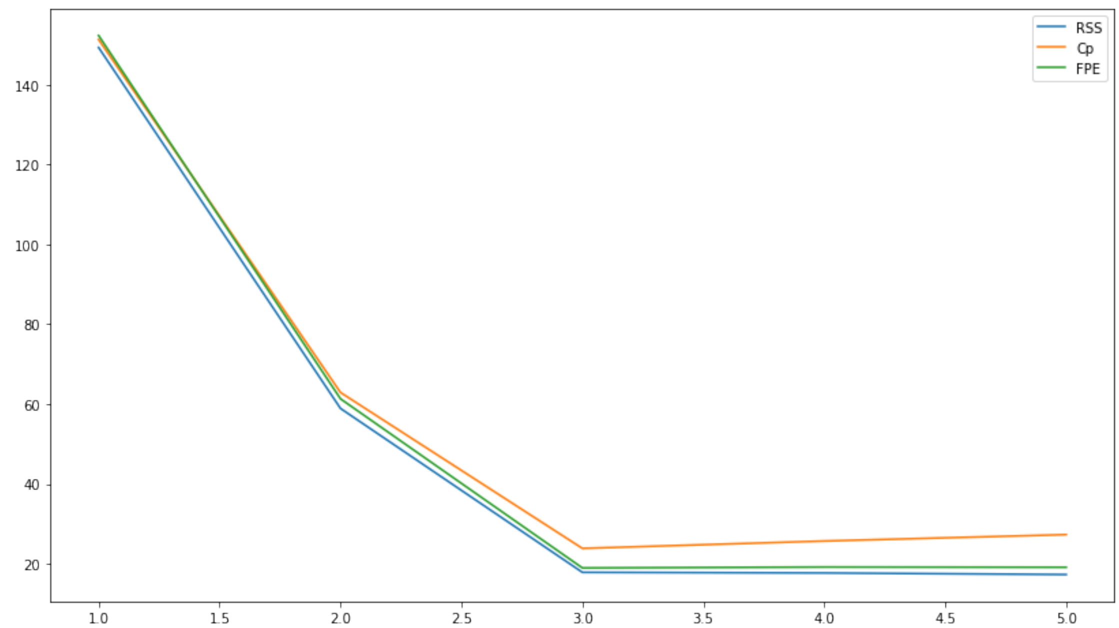
```
> RSS_4: 0.9071691999350311
==============================================
        Step 5
==============================================
> θ_5: [ 3.00447526e+00 -1.95562053e+00  9.91512807e-01 -3.37209128e-02
 -5.39683672e-04]
> H_5_inv:
[[ 0.02728664 -0.00479622 -0.00315327 -0.00900608 -0.00677842]
 [-0.00479622  0.03182468 -0.01450057 -0.00327198 -0.00885175]
 [-0.00315327 -0.01450057  0.03157622 -0.00816724 -0.00548814]
 [-0.00900608 -0.00327198 -0.00816724  0.02521822 -0.00433901]
 [-0.00677842 -0.00885175 -0.00548814 -0.00433901  0.03075334]]
> RSS_5: 0.9071597291441771

                        RESULTS

PARAMETERS
True values:        θ: [ 3 -2  1  0  0]
Estimates:        θ*: [ 3.00447526e+00 -1.95562053e+00  9.91512807e-01 -3.37209128e-02
 -5.39683672e-04]
```




```
     s         RSS          Cp         FPE
0  1.0  139.904006  141.904006  142.730349
1  2.0   36.009932   40.009932   37.479725
2  3.0    0.953590    6.953590    1.012575
3  4.0    0.907169    8.907169    0.982767
```

```
4  5.0    0.907160   10.907160    1.002650
s* by Cp:  3
s* by FPE: 4

--------------------------------------------------------------
                        SAMPLE #8
--------------------------------------------------------------
                CONFUGURATIONS & DATA
Sample length: n = 100
Noise generation: σ = 0.5
X[:10]:
[[1.16187113 0.56692258 0.88095381 0.32026532 0.09746003]
 [0.38195709 1.42179186 1.01824668 0.34410347 1.71823047]
 [1.46259151 0.68407143 0.29287657 1.28169498 1.92523134]
 [1.97185322 1.18134425 1.79891701 0.12145096 0.1565097 ]
 [0.95352219 0.65448288 0.0711204  0.71785963 0.33788788]
 [1.82159754 0.20021885 0.88754426 0.08819382 0.1003117 ]
 [0.07789955 1.81320805 0.38136005 0.43157208 0.93967719]
 [1.95729242 1.14355876 1.26371677 0.27375364 1.20363225]
 [0.62060307 1.0264235  0.58194768 0.69663503 1.41678501]
 [1.354261   0.39651202 1.62760213 1.56285985 0.4112288 ]]
y[:10]:
[ 2.38814537 -0.266042    3.83982479  5.04186083  2.25968846  6.26543951
 -3.28538265  4.52274475  0.4938657   5.23045345]

                RLSM ITERATIONS
===============================================
        Step 1
===============================================
> θ_1: [2.40017533]
> H_1_inv:
[[0.00709572]]
> RSS_1: 149.33523273005187
===============================================
        Step 2
===============================================
> θ_2: [ 3.34070496 -1.31207733]
> H_2_inv:
[[ 0.01687949 -0.01364876]
 [-0.01364876  0.01904058]]
> RSS_2: 58.92059074839193
===============================================
        Step 3
===============================================
> θ_3: [ 2.98275274 -1.91011335  1.00409455]
> H_3_inv:
[[ 0.02000003 -0.00843524 -0.00875343]
 [-0.00843524  0.02775088 -0.01462449]
```

```
 [-0.00875343 -0.01462449  0.02455432]]
> RSS_3: 17.860373938172174
===============================================
        Step 4
===============================================
> θ_4: [ 2.96394998 -1.92331992  0.98658008  0.06439653]
> H_4_inv:
[[ 0.0223063  -0.00681537 -0.00660517 -0.00789865]
 [-0.00681537  0.02888864 -0.01311561 -0.0055478 ]
 [-0.00660517 -0.01311561  0.0265554  -0.00735746]
 [-0.00789865 -0.0055478  -0.00735746  0.02705165]]
> RSS_4: 17.70707778387631
===============================================
        Step 5
===============================================
> θ_5: [ 2.95898496 -1.9600944   0.96301441  0.04528211  0.09973274]
> H_5_inv:
[[ 0.0223677  -0.0063606  -0.00631375 -0.00766227 -0.00123333]
 [-0.0063606   0.03225696 -0.01095713 -0.00379704 -0.00913492]
 [-0.00631375 -0.01095713  0.02793858 -0.00623555 -0.0058538 ]
 [-0.00766227 -0.00379704 -0.00623555  0.02796165 -0.00474809]
 [-0.00123333 -0.00913492 -0.0058538  -0.00474809  0.02477398]]
> RSS_5: 17.305583185460527

                        RESULTS

PARAMETERS
True values:        θ: [ 3 -2  1  0  0]
Estimates:        θ*: [ 2.95898496 -1.9600944   0.96301441  0.04528211  0.09973274]
```




```
     s         RSS          Cp         FPE
0  1.0  149.335233  151.335233  152.352106
1  2.0   58.920591   62.920591   61.325513
2  3.0   17.860374   23.860374   18.965139
3  4.0   17.707078   25.707078   19.182668
4  5.0   17.305583   27.305583   19.127224
s* by Cp:  3
s* by FPE: 3

-------------------------------------------------------------
                        SAMPLE #9
-------------------------------------------------------------
                CONFUGURATIONS & DATA
Sample length: n = 100
Noise generation: σ = 1
X[:10]:
[[0.16317167 1.04009727 0.61903509 1.41726038 0.1041959 ]
 [0.03426191 1.54064845 1.38433454 1.95238531 1.57416181]
 [0.05619171 0.67891817 0.82471112 1.37210371 0.05593376]
 [1.4317724  0.92900862 0.03548929 1.2426427  0.01113482]
 [0.93923039 0.6533068  0.70056497 0.607301   0.71847807]
 [0.97213425 0.14949815 1.82198766 1.43888765 1.94579019]
 [0.64720489 0.85166612 1.14092488 0.22591394 1.77904608]
 [1.55753117 1.50877538 1.94006306 1.67828737 1.39720275]
 [0.81905904 0.07663614 1.13281294 0.28873397 0.3659255 ]
 [0.18543698 1.66253315 0.45416154 0.10617749 0.08200263]]
```

```
y[:10]:
[-1.38886961 -1.10470176  0.02462339  1.37963429  0.8109831   5.42763679
  1.96177076  2.97733318  5.03682273 -2.54133726]

                RLSM ITERATIONS
===============================================
        Step 1
===============================================
> θ_1: [2.29777755]
> H_1_inv:
[[0.00924745]]
> RSS_1: 321.732971393194
===============================================
        Step 2
===============================================
> θ_2: [ 3.62354553 -1.61403333]
> H_2_inv:
[[ 0.02015025 -0.01327342]
 [-0.01327342  0.0161595 ]]
> RSS_2: 160.5210677852021
===============================================
        Step 3
===============================================
> θ_3: [ 3.20833974 -2.08013516  1.01265985]
> H_3_inv:
[[ 0.02407731 -0.00886498 -0.00957786]
 [-0.00886498  0.02110833 -0.01075191]
 [-0.00957786 -0.01075191  0.02335977]]
> RSS_3: 116.62165726531734
===============================================
        Step 4
===============================================
> θ_4: [ 3.14722556 -2.12966522  0.9258072   0.20096952]
> H_4_inv:
[[ 0.02596771 -0.0073329  -0.00689131 -0.00621643]
 [-0.0073329   0.02235001 -0.0085746  -0.00503812]
 [-0.00689131 -0.0085746   0.02717776 -0.00883451]
 [-0.00621643 -0.00503812 -0.00883451  0.02044229]]
> RSS_4: 114.64591220959169
===============================================
        Step 5
===============================================
> θ_5: [ 3.14978672 -2.12652568  0.929833    0.20372165 -0.01407788]
> H_5_inv:
[[ 0.02684447 -0.00625814 -0.00551315 -0.00527429 -0.00481931]
 [-0.00625814  0.02366748 -0.00688521 -0.00388321 -0.00590764]
 [-0.00551315 -0.00688521  0.02934404 -0.00735358 -0.00757532]
 [-0.00527429 -0.00388321 -0.00735358  0.02145468 -0.00517866]
```

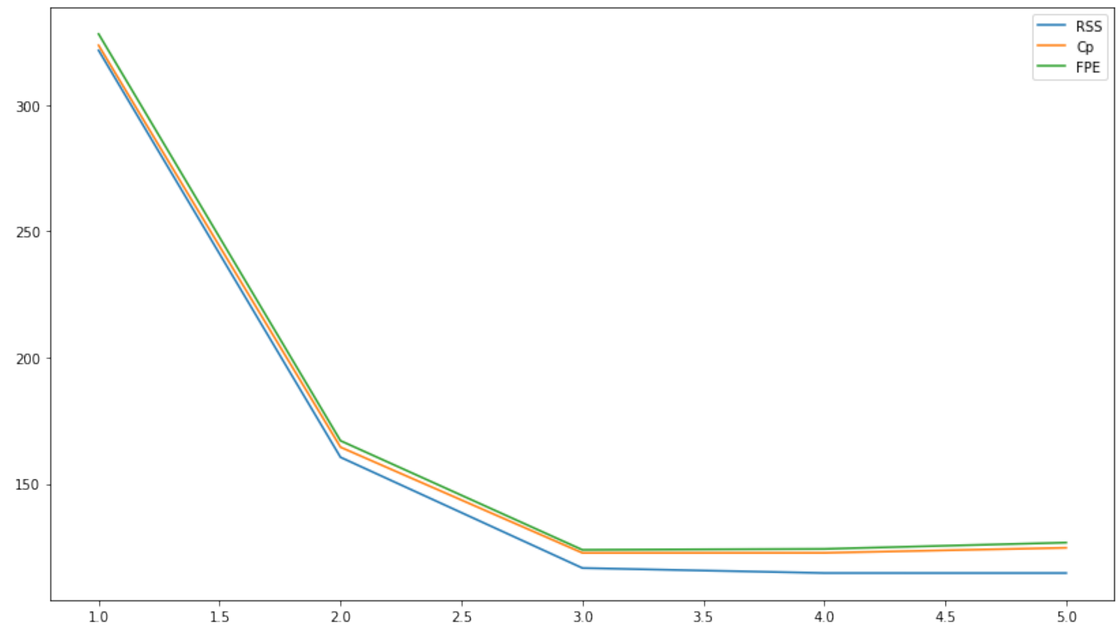
```
 [-0.00481931 -0.00590764 -0.00757532 -0.00517866  0.02649024]]
> RSS_5: 114.63843070995833
```

RESULTS

PARAMETERS

True values: θ: [3 -2 1 0 0]

Estimates: θ*: [3.14978672 -2.12652568 0.929833 0.20372165 -0.01407788]



	s	RSS	Cp	FPE
0	1.0	321.732971	323.732971	328.232627
1	2.0	160.521068	164.521068	167.072948
2	3.0	116.621657	122.621657	123.835368
3	4.0	114.645912	122.645912	124.199738
4	5.0	114.638431	124.638431	126.705634

s* by Cp: 3

s* by FPE: 3

3.3 Результати роботи МНКО для кожної складності моделі з перестановками

Оскільки алгоритм МНКО діє послідовно в порядку розташування регресорів, має сенс будувати моделі для перестановок (найвпливовіший регресор, наприклад, може бути розташований в кінці, а найменш впливовий - на початку; це потрібно відслідковувати).

Тож виконаємо МНКО для однієї вибірки довжини n = 100, з дисперсією шуму $\sigma = 1$ з перестановкою регресорів.

In [48]:
```
config.run_single_full_RMNK_model_selection(sort_values_by=['Cp', 'FPE'])
# ['s', 'Cp', 'FPE']
```

Out[48]:

	s	regressors	RSS	Cp	FPE
0	3.0	[1, 2, 3]	116.621657	122.621657	123.835368
1	4.0	[1, 2, 3, 4]	114.645912	122.645912	124.199738
2	4.0	[1, 2, 3, 5]	116.572858	124.572858	126.287262
3	5.0	[1, 2, 3, 4, 5]	114.638431	124.638431	126.705634
4	4.0	[1, 2, 4, 5]	144.102310	152.102310	156.110836
5	3.0	[1, 2, 4]	146.183425	152.183425	155.225699
6	3.0	[1, 2, 5]	153.827996	159.827996	163.343130
7	2.0	[1, 2]	160.521068	164.521068	167.072948
8	3.0	[1, 3, 5]	306.719568	312.719568	325.691912
9	2.0	[1, 5]	309.379084	313.379084	322.006802
10	4.0	[1, 3, 4, 5]	305.707017	313.707017	331.182602
11	3.0	[1, 4, 5]	309.249047	315.249047	328.377854
12	2.0	[1, 4]	318.070515	322.070515	331.052985
13	3.0	[1, 3, 4]	317.575349	323.575349	337.219185
14	1.0	[1]	321.732971	323.732971	328.232627
15	2.0	[1, 3]	321.610000	325.610000	334.736939
16	4.0	[2, 3, 4, 5]	484.217522	492.217522	524.568983
17	3.0	[2, 3, 4]	496.082280	502.082280	526.767782
18	3.0	[2, 3, 5]	517.356037	523.356037	549.357442
19	2.0	[2, 3]	544.137976	548.137976	566.347689
20	2.0	[3, 4]	572.017780	576.017780	595.365444
21	3.0	[3, 4, 5]	571.494826	577.494826	606.845021
22	3.0	[2, 4, 5]	572.337972	578.337972	607.740321
23	2.0	[3, 5]	584.595152	588.595152	608.456178
24	1.0	[3]	589.415868	591.415868	601.323260
25	2.0	[4, 5]	616.452583	620.452583	641.613913
26	2.0	[2, 4]	618.422055	622.422055	643.663771
27	1.0	[4]	634.770928	636.770928	647.594583
28	2.0	[2, 5]	665.920341	669.920341	693.100763
29	1.0	[5]	675.015687	677.015687	688.652368
30	1.0	[2]	812.130057	814.130057	828.536725

В даному випадку за другим критерієм оптимальною є модель складності 3 з регресорами 1, 2, 3.

4 Висновки

При побудові регресійних моделей для різницевого рівняння моделі Фергюльста та різницевого рівнянна згасаючих коливань ми показали, що алгоритм МНКО є ефективним при оцінюванні параметрів моделі. У порівнянні з МНК він є набагато менш затратним через те, що не виконує пряме обертання матриці, а рекурентно виконує апроксимацію оберненої матриці, і при цьому дає досить точні оцінки, залежність точності яких від шуму була показана в роботі. При відносно невеликих шумах алгоритм показує дуже хороші результати.

В ході проведення статистичних експериментів ми з'ясували, що критерії Маллоуза і критерій фінальної помилки передбачення Акаіке є досить стійкими до зростаючого рівня шуму, і при додаванні нерелевантних регресорів моделі, вони не обирають переускладнені моделі, оскільки в них наявний штраф за складність. За теоретичними оцінками відомо, що за умови наявності точної оцінки дисперсії (рівня) шуму критерій Маллоуза є оптимальним з точки зору теорії завадостійкого моделювання. Ми підтвердили це експериментально.

5 Код програми

5.1 Імпорт необхідних бібліотек; налаштування

In [20]:
```
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'] = 14, 8

import plotly.offline as py
import plotly.graph_objs as go

py.offline.init_notebook_mode(connected=True)
```

5.2 Реалізація МНКО

In [21]:
```
def RMNK(X, y, s=None, sigma_estimation=1
          verbose=False, deep_verbose=False, create_dataframe=False):
    assert X.ndim == 2 and X.shape[1] > 0
    m = X.shape[1]
    if m > 1:
        if create_dataframe:
            w, H_inv, RSS, df = RMNK(X[:,:-1], y, s, sigma_estimation
                                     verbose, deep_verbose, create_dataframe)
            if s is not None and m > s:
                return w, H_inv, RSS, df
        else:
            w, H_inv, RSS = RMNK(X[:,:-1], y, s, sigma_estimation,
                                 verbose, deep_verbose, create_dataframe)
            if s is not None 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 = (X[:,:-1].T @ X[:,-1]).reshape(-1,1) # shape = [m-1, 1]
        eta = X[:,-1].T @ X[:,-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 = X[:,-1].T @ y # 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 = X[:,-1].T @ X[:,-1]
            beta_inv = 1 / beta
            alpha = h = np.array([0])
            gamma = 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 - y.T @ X[:,-1].reshape(-1, 1) @ w)[0]
            if create_dataframe:
                df = pd.DataFrame(columns=['s', 'RSS', 'Cp', 'FPE'])

        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{}'.format(m, alpha.reshape(-1,1)[:,0]))
                print('beta_{}:\t\t{}'.format(m, beta))
                print('gamma_{}:\t{}'.format(m, gamma))
                print('nu_{}:\t\t{}'.format(m, nu))
                print('===============================================')
            print('> θ_{}: {}'.format(m, w[:, 0]))
            print('> H_{}_inv:\n{}'.format(m, H_next_inv))
            print('> RSS_{}: {}'.format(m, RSS_next))
        if create_dataframe:
            Cp = RSS_next + 2 * sigma_estimation * m
            n = y.shape[0]
            FPE = (n + m) / (n - m) * RSS_next
            df = df.append({'s': m, 'RSS': RSS_next, 'Cp': Cp, 'FPE': FPE},
                           ignore_index=True)
            return w, H_next_inv, RSS_next, df
        return w, H_next_inv, RSS_next
```

5.3 Модель Фергюльста: реалізація класу, методів, функцій

```
In [22]: def Verhulst_model_equation(N, t, μ, k):
             return μ * N * (k - N)

         class VerhulstModelConfig():
             k = 100
             μ = 0.0001
             N0 = 10
             t_start = 0
```

```
t_end = 500
num_samples = 50
num_samples_grid = [10, 50, 100]
C = 3
C_grid = [0, 2, 5]

def __init__(self):
    self.theta = self.init_to_inter_params()
    self.compile()

def compile(self):
    self.h = int((self.t_end - self.t_start) / (self.num_samples - 1))
    self.t = np.linspace(self.t_start, self.t_end, num=self.num_samples)
    self.N = odeint(Verhulst_model_equation,
                    self.N0, self.t, (self.μ, self.k))
    self.create_data_sample()

def recompile(self, C, num_samples):
    self.C = C
    self.num_samples = num_samples
    self.compile()

def show(self):
    print('Initial parameters:\tμ = {}\n\t\t\t\
           k = {}\n\t\t\tN0 = {}\n'.format(self.μ, self.k, self.N0))
    print('Noise generation: C = {}'.format(self.C))
    print('Sample length: n = {}'.format(self.num_samples))
    print('Time starting from {} to {} \
           with discretization frequency {}\n'.format(self.t_start,
                                                       self.t_end,
                                                       self.h))


def init_to_inter_params(self):
    w1 = self.μ * self.k + 1
    w2 = -self.μ
    return np.array([w1, w2])

def inter_to_init_params(self, w1, w2):
    μ = - w2
    k = (1 - w1) / w2
    return μ, k

def create_data_sample(self):
    self.df = pd.DataFrame()
    self.df['i'] = range(1, self.num_samples+1)
    self.df['t'] = list(map(int, self.t))
    self.df['N(t)'] = self.N.flatten()
    self.df['N^2(t)'] = np.square(self.N.flatten())
```

```
        self.df['N(t+1)'] = np.array(self.df[['N(t)','N^2(t)']]) @ self.theta
        self.df['N(t+1)'] = np.round(self.df['N(t+1)'], self.C)
        self.X = np.array(self.df[['N(t)', 'N^2(t)']])
        self.y = np.array(self.df['N(t+1)'])

    def plot_3D(self):
        trace1 = go.Scatter3d(
            x=self.df['N(t)'],
            y=self.df['N^2(t)'],
            z=self.df['N(t+1)'],
            mode='markers',
            marker=dict(
                size=12,
                line=dict(
                    color='rgba(217, 217, 217, 0.14)',
                    width=0.5
                ),
                opacity=0.8
            )
        )

        data = [trace1]
        layout = go.Layout(
            margin=dict(
                l=0,
                r=0,
                b=0,
                t=0
            )
        )
        fig = go.Figure(data=data, layout=layout)
        py.iplot(fig, filename='simple-3d-scatter')

    def run_single_RMNK(self, verbose=True, deep_verbose=True):
        print('Recurrent Least Squares Method')
        self.theta_pred = RMNK(self.X, self.y, verbose=verbose,
                               deep_verbose=deep_verbose)[0][:,0]
        self.μ_pred, self.k_pred = \
        self.inter_to_init_params(*self.theta_pred)
        print('=============================================')
        print('\nINTERMEDIATE PARAMETERS')
        print('True values:\tθ_1 = {}\tθ_2 = {}'.format(*self.theta))
        print('Estimates:\tθ_1* = {}\tθ_2* = {}'.format(*self.theta_pred))
        print('\nINITIAL PARAMETERS')
        print('True values:\tμ = {}\tk = {}'.format(self.μ, self.k))
        print('Estimates:\tμ* = {}\tk* = {}'.format(self.μ_pred, self.k_pred))
        plt.scatter(self.t, self.y)
        t_for_plot = np.linspace(self.t_start, self.t_end,
```

```
                              num=self.num_samples * 10)
    plt.plot(t_for_plot, odeint(Verhulst_model_equation,
                                self.N0, t_for_plot,
                                (self.μ_pred, self.k_pred)), 'r')
    plt.show()

def run_grid_RMNK(self, verbose=False):
    intermediate_estimates_df = pd.DataFrame(columns=['C', 'num_samples',
                                                      'θ_1', 'θ_1*',
                                                      'θ_2', 'θ_2*'])
    initial_estimates_df = pd.DataFrame(columns=['C', 'num_samples',
                                                 'μ', 'μ*', 'k', 'k*'])
    for C in self.C_grid:
        for num_samples in self.num_samples_grid:
            self.recompile(C, num_samples)
            theta_pred = RMNK(self.X, self.y, verbose=False)[0][:,0]
            μ_pred, k_pred = self.inter_to_init_params(*theta_pred)
            intermediate_estimates_df = \
            intermediate_estimates_df.append({'C': self.C,
                                              'num_samples': self.num_samples,
                                              'θ_1': self.theta[0],
                                              'θ_1*': theta_pred[0],
                                              'θ_2': self.theta[1],
                                              'θ_2*': theta_pred[1]},
                                             ignore_index=True)
            initial_estimates_df = \
            initial_estimates_df.append({'C': self.C,
                                         'num_samples': self.num_samples,
                                         'μ': self.μ,
                                         'μ*': μ_pred,
                                         'k': self.k,
                                         'k*': k_pred},
                                        ignore_index=True)
            if verbose:
                print('=============================================================
                print('C: {}\t num_samples: {}'.format(self.C,
                                                       self.num_samples))
                print('\nINTERMEDIATE PARAMETERS')
                print('Estimates:\tθ_1* = {: 12.8}\tθ_2* = \
                {:12.8}'.format(*theta_pred))
                print('\nINITIAL PARAMETERS')
                print('Estimates:\tμ* = {: 12.8}\tk* = \
                {: 12.8}'.format(μ_pred, k_pred))

    return pd.concat([intermediate_estimates_df,
                      initial_estimates_df[['μ', 'μ*', 'k', 'k*']]],
                     axis=1)
```

5.4 Модель згасаючих коливань: реалізація класу, методів, функцій

```
In [27]: def Oscillation_model_equation(x, t, δ, ω0_sqr):
             #x_0' = x_1 = x'
             #x_1' = x'' = - 2 * δ * x[1] - (ω0 ** 2) * x[0]
             return [x[1],  - 2 * δ * x[1] - ω0_sqr * x[0]]

         class OscillationModelConfig():
             δ = 0.005
             ω0_sqr = 0.01
             x0 = 5
             x00 = 2
             t_start = 0
             t_end = 500
             num_samples = 80
             num_samples_grid = [30, 80, 150]
             C = 2
             C_grid = [0, 2, 5]

             def __init__(self, difference='forward'):
                 self.difference = difference
                 self.theta = self.init_to_inter_params()
                 self.compile()

             def compile(self):
                 self.h = int((self.t_end - self.t_start) / (self.num_samples - 1))
                 self.t = np.linspace(self.t_start, self.t_end, num=self.num_samples)
                 self.x = odeint(Oscillation_model_equation,
                                 np.array([self.x0, self.x00]),
                                 self.t, (self.δ, self.ω0_sqr))
                 self.x1 = self.x0 + self.x00
                 self.x11 = self.x00
                 self.x_1 = odeint(Oscillation_model_equation,
                                   np.array([self.x1, self.x11]), self.t+1,
                                   (self.δ, self.ω0_sqr))
                 self.create_data_sample()

             def recompile(self, C, num_samples):
                 self.C = C
                 self.num_samples = num_samples
                 self.compile()

             def show(self):
                 print('Initial parameters:\tδ = {}\n\t\t\tω0^2 = {}\
                       \n\t\t\tx0 = {}\n\t\t\tx00 = {}\n'.format(self.δ, self.ω0_sqr,
                                                                 self.x0, self.x00))
                 print('Noise generation: C = {}'.format(self.C))
                 print('Sample length: n = {}'.format(self.num_samples))
```

```
    print('Time starting from {} to {} \
           with discretization frequency {}\n'.format(self.t_start,
                                                      self.t_end,
                                                      self.h))

def init_to_inter_params(self):
    if self.difference == 'center':
        divider = 1 + self.δ
        w1 = (2 - self.ω0_sqr) / divider
        w2 = - (1 - self.δ) / divider
    elif self.difference == 'forward':
        divider = 1 + 2 * self.δ
        w1 = (2 + 2 * self.δ - self.ω0_sqr) / divider
        w2 = - 1 / divider
    return np.array([w1, w2])

def inter_to_init_params(self, w1, w2):
    if self.difference == 'center':
        δ = (1 + w2) / (1 - w2)
        ω0_sqr = (2 - 2 * w1 - 2 * w2) / (1 - w2)
    elif self.difference == 'forward':
        δ = - (1 / w2 + 1) / 2
        ω0_sqr = 1 - 1 / w2 + w1 / w2
    return δ, ω0_sqr

def create_data_sample(self):
    self.df = pd.DataFrame()
    self.df['i'] = range(1, self.num_samples+1)
    self.df['t'] = list(map(int, self.t))
    self.df['x(t)'] = self.x[:,0].flatten()
    self.df['x(t+1)'] = self.x_1[:,0].flatten()
    self.df['x(t+2)'] = np.array(self.df[['x(t)','x(t+1)']]) @ self.theta
    self.df['x(t+2)'] = np.round(self.df['x(t+2)'], self.C)
    self.X = np.array(self.df[['x(t)', 'x(t+1)']])
    self.y = np.array(self.df['x(t+2)'])

def plot_3D(self):
    trace1 = go.Scatter3d(
        x=self.df['x(t)'],
        y=self.df['x(t+1)'],
        z=self.df['x(t+2)'],
        mode='markers',
        marker=dict(
            size=12,
            line=dict(
                color='rgba(217, 217, 217, 0.14)',
                width=0.5
            ),
```

```
            opacity=0.8
        )
    )

    data = [trace1]
    layout = go.Layout(
        margin=dict(
            l=0,
            r=0,
            b=0,
            t=0
        )
    )
    fig = go.Figure(data=data, layout=layout)
    py.iplot(fig, filename='simple-3d-scatter')

def run_single_RMNK(self, verbose=True, deep_verbose=False):
    print('Recurrent Least Squares Method')
    self.theta_pred = RMNK(self.X, self.y, verbose=verbose,
                           deep_verbose=deep_verbose)[0][:,0]
    self.δ_pred, self.ω0_sqr_pred = \
    self.inter_to_init_params(*self.theta_pred)
    print('=============================================')
    print('\nINTERMEDIATE PARAMETERS')
    print('True values:\tθ_1 = {}\tθ_2 = {}'.format(*self.theta))
    print('Estimates:\tθ_1* = {}\tθ_2* = {}'.format(*self.theta_pred))
    print('\nINITIAL PARAMETERS')
    print('True values:\tδ = {}\t\t\tω0^2 = {}'.format(self.δ,
                                                       self.ω0_sqr))
    print('Estimates:\tδ* = {}\tω0^2* = {}'.format(self.δ_pred,
                                                   self.ω0_sqr_pred))
    plt.scatter(self.t, self.y)
    t_for_plot = np.linspace(self.t_start, self.t_end,
                             num=self.num_samples * 10)
    plt.plot(t_for_plot, odeint(Oscillation_model_equation,
                                np.array([self.x0, self.x00]),
                                t_for_plot,
                                (self.δ_pred,
                                 self.ω0_sqr_pred))[:,0], 'r')
    plt.show()

def run_grid_RMNK(self, verbose=True):
    intermediate_estimates_df = pd.DataFrame(columns=['C', 'num_samples',
                                                      'θ_1', 'θ_1*',
                                                      'θ_2', 'θ_2*'])
    initial_estimates_df = pd.DataFrame(columns=['C', 'num_samples',
                                                 'δ', 'δ*',
                                                 'ω0_sqr', 'ω0_sqr*'])
```

```
        for C in self.C_grid:
            for num_samples in self.num_samples_grid:
                self.recompile(C, num_samples)
                intermediate_estimates_df
                theta_pred = RMNK(self.X, self.y, verbose=False)[0][:,0]
                δ_pred, ω0_sqr_pred = self.inter_to_init_params(*theta_pred)
                intermediate_estimates_df = \
                intermediate_estimates_df.append({'C': self.C,
                                                  'num_samples': self.num_samples,
                                                  'θ_1': self.theta[0],
                                                  'θ_1*': theta_pred[0],
                                                  'θ_2': self.theta[1],
                                                  'θ_2*': theta_pred[1]},
                                                 ignore_index=True)
                initial_estimates_df = \
                initial_estimates_df.append({'C': self.C,
                                             'num_samples': self.num_samples,
                                             'δ': self.δ,
                                             'δ*': δ_pred,
                                             'ω0_sqr': self.ω0_sqr,
                                             'ω0_sqr*': ω0_sqr_pred},
                                             ignore_index=True)
                if verbose:
                    print('==========================================================
                    print('C: {}\t num_samples: {}'.format(self.C,
                                                           self.num_samples))
                    print('\nINTERMEDIATE PARAMETERS')
                    print('Estimates:\tθ_1* = {: 12.8}\tθ_2* = \
                    {:12.8}'.format(*theta_pred))
                    print('\nINITIAL PARAMETERS')
                    print('Estimates:\tδ* = {: 12.8}\tk* = \
                    {: 12.8}'.format(δ_pred, ω0_sqr_pred))

        return pd.concat([intermediate_estimates_df,
                          initial_estimates_df[['δ', 'δ*', \
                                                'ω0_sqr', 'ω0_sqr*']]],
                         axis=1)
```

5.5 Селекція оптимальних моделей: реалізація класу, методів, функцій

```
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 = 0.01
```

```
    sigma_grid = [0.1, 0.5, 1]
    s = 5
    s_grid = [1, 2, 3, 4, 5]

    def __init__(self):
        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):
        self.X = np.random.uniform(self.a, self.b, size=(self.n, self.m))
        self.generate_noise_and_output()

    def recompile(self, n, sigma):
        self.n = n
        self.sigma = sigma
        self.compile()

    def show(self):
#         print('Regressors: m = {}'.format(self.m))
#         print('True parameters: θ = {}'.format(self.theta))
        print('Sample length: n = {}'.format(self.n))
        print('Noise generation: σ = {}'.format(self.sigma))
        print('X[:10]:\n{}'.format(self.X[:10]))
        print('y[:10]:\n{}'.format(self.y[:10]))

    def run_grid_RMNK_model_selection(self):
        for i, n in enumerate(self.n_grid):
            for j, sigma in enumerate(self.sigma_grid):
                self.recompile(n, sigma)
                print('---------------------------------------------------------------')
                print('\t\t\tSAMPLE #{}'.format(i * len(self.n_grid) + j + 1))
                print('---------------------------------------------------------------')
                print('\t\tCONFUGURATIONS & DATA')
                self.show()
                print('\n\t\tRLSM ITERATIONS')
                theta_pred, _, _, df = RMNK(self.X, self.y, s=self.s,
                                            verbose=True,
                                            create_dataframe=True)
                print('\n\t\t\tRESULTS')
                print('\nPARAMETERS')
                print('True values:\tθ: {}'.format(self.theta))
                print('Estimates:\tθ*: {}'.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_full_RMNK_model_selection(self,
                                              sort_values_by=['Cp', 'FPE']):
        total_df = pd.DataFrame()
        for p in permutations(range(self.X.shape[1])):
            p = np.array(p)
            theta_pred, _, _, df = RMNK(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]
            total_df = pd.concat([total_df, df], axis=0)

        total_df['RSS'] = np.round(total_df['RSS'], 6)
        total_df['Cp'] = np.round(total_df['Cp'], 6)
        total_df['FPE'] = np.round(total_df['FPE'], 6)
        total_df = total_df.drop_duplicates()
        total_df = total_df.sort_values(by=sort_values_by).reset_index()\
                   [['s', 'regressors', 'RSS', 'Cp', 'FPE']]
        return total_df
```