Анализ данных

Хашин С.И.
http://math.ivanovo.ac.ru/dalgebra/Khashin/index.html

Ивановский университет

.

Линейные рекуррентные последовательности

.

Иваново-2023

План

Определение

Рассмотрим последовательность чисел $x_0, x_1, \ldots,$ удовлетворяющую условию (при $n \geq k$):

$$x_n = a_{k-1}x_{n-1} + a_{k-2}x_{n-2} + \cdots + a_0 x_{n-k}.$$

Например, при $k = 2$ и $F_0 = F_1 = 1$:

$$F_n = F_{n-1} + F_{n-2}.$$

Таким образом:

$$\{F_i\} = \{1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots\}$$

— числа Фибоначчи. Как получить явную формулу для F_n?

Явная формула

Рассмотрим последовательность чисел $x_n = \lambda^n$ при некотором $\lambda \in \mathbb{R}$. При каком λ она будет удовлетворять условию $x_n = x_{n-1} + x_{n-2}$?

$$\lambda^n = \lambda^{n-1} + \lambda^{n-2},$$

или

$$\lambda^2 = \lambda + 1.$$

— характеристическое уравнение. Отсюда получаем:

$$\lambda_{1,2} = \frac{1 \pm \sqrt{5}}{2}.$$

Поэтому последовательность $x_n = a_1\lambda_1^n + a_2\lambda_2^n$ будет удовлетворять условию при любых a_1, a_2. Осталось подобрать a_1, a_2 так, чтобы $x_0 = x_1 = 1$.

Явная формула

Находим:

$$F_n = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}.$$

Поэтому последовательность $x_n = a_1\lambda_1^n + a_2\lambda_2^n$ будет удовлетворять условию при любых a_1, a_2. Осталось подобрать a_1, a_2 так, чтобы $x_0 = x_1 = 1$. Если подставить приближённые значения, получим

$$F_n \approx \frac{1.618033989^n - (-0.6180339887)^n}{2.236067977}.$$

В общем виде

$$x_n = a_{k-1}x_{n-1} + a_{k-2}x_{n-2} + \cdots + a_0 x_{n-k}.$$

Будем искать решения в виде $x_n = \lambda^n$.

$$\lambda^n = a_{k-1}\lambda^{n-1} + a_{k-2}\lambda^{n-2} + \cdots + a_0\lambda^{n-k}.$$

или

$$\lambda^k = a_{k-1}\lambda^{k-1} + a_{k-2}\lambda^{k-2} + \cdots + a_0.$$

или

$$\lambda^k - a_{k-1}\lambda^{k-1} - a_{k-2}\lambda^{k-2} - \cdots - a_0 = 0.$$

— характеристическое уравнение. Пусть $\{\lambda_1, \ldots, \lambda_k\}$ — его корни. Решение, для любых a_j:

$$x_n = a_1\lambda_1^n + \cdots + a_k\lambda_k^n \ .$$

Комплексный корень

Предположим, что среди корней есть пара комплексно сопряженных корней $\lambda_{1,2} = a \pm b \cdot i$ или, в тригонометрической форме

$$\lambda_{1,2} = r(\cos\phi \pm i \cdot \sin\phi) = re^{\pm i\phi}$$

Тогда

$$a_1\lambda_1^n + a_2\lambda_2^n = r(a_1 re^{i\cdot n\phi} + a_2 e^{-i\cdot n\phi}) =$$

$$= r((a_1 + a_2)\cos n\phi + (a_1 - a_2)i\sin n\phi)$$

или, подобрав подходящие b_1, b_2:

$$x_n = r(b_1\cos n\phi + b_2\sin n\phi)$$

Пример

Рассмотрим две последовательности x_n, y_n, обе начинаются с 1,1 и удовлетворяют формулам:

$$x_n = 1.5x_{n-1} - 1.05x_{n-2},$$

$$y_n = 1.5y_{n-1} - 0.95y_{n-2},$$

Их характеристические уравнения:

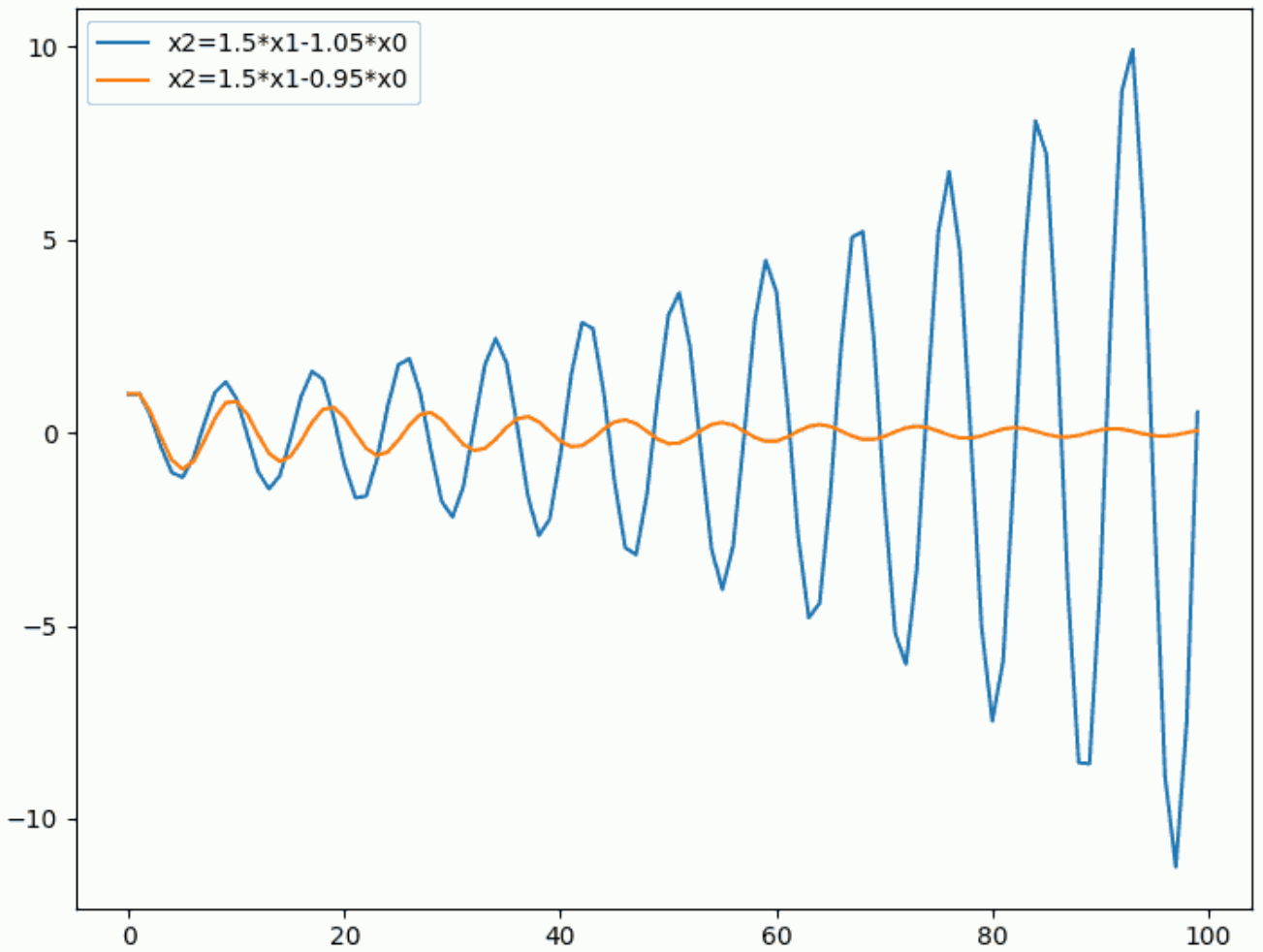
$$\lambda^2 - 1.5\lambda + 1.05, \quad \lambda^2 - 1.5\lambda + 0.95$$

и корни:

$$0.75 \pm 0.6225 * i, \quad 0.75 \pm 0.6982 * i \ .$$

Пример

А вот графики x_n, y_n при $n = 0, \ldots, 100$:



Простая регрессия

Пусть дан временной ряд: $x_0, \ldots, x_{N-1}$. Будем искать аппроксимацию каждого элемента через предыдущие ($M = 4$):

$$x_4 \approx a_3 x_3 + a_2 x_2 + a_1 x_1 + a_0 x_0,$$

$$x_5 \approx a_3 x_4 + a_2 x_3 + a_1 x_2 + a_0 x_1,$$

$$x_6 \approx a_3 x_5 + a_2 x_4 + a_1 x_3 + a_0 x_2,$$

$$\ldots$$

Уравнения для простой регрессии

```
def recurr_equations(x,M):
    '''
    уравнения для простой рекурсии
    :param x: временной ряд
    :param M: через сколько шагов предсказываем
    :return: (A,b) - система уравнений
    '''
    N = len(x)
    A = np.zeros((N-M, M))
    b = x[M:]
    for i in range(N-M):
        A[i] = x[i:M+i]
    return A,b
```

Проверка

```
def tst_recurr_equations():
    x = np.arange(10)
    A,b = recurr_equations(x, 2)
    print(A, '=A\n')
    print(b, '=b\n')
    w = lin_reg1(b,A)
    print('w=', w) # w=[-1,2]
```

Рекуррентный прогноз временного ряда

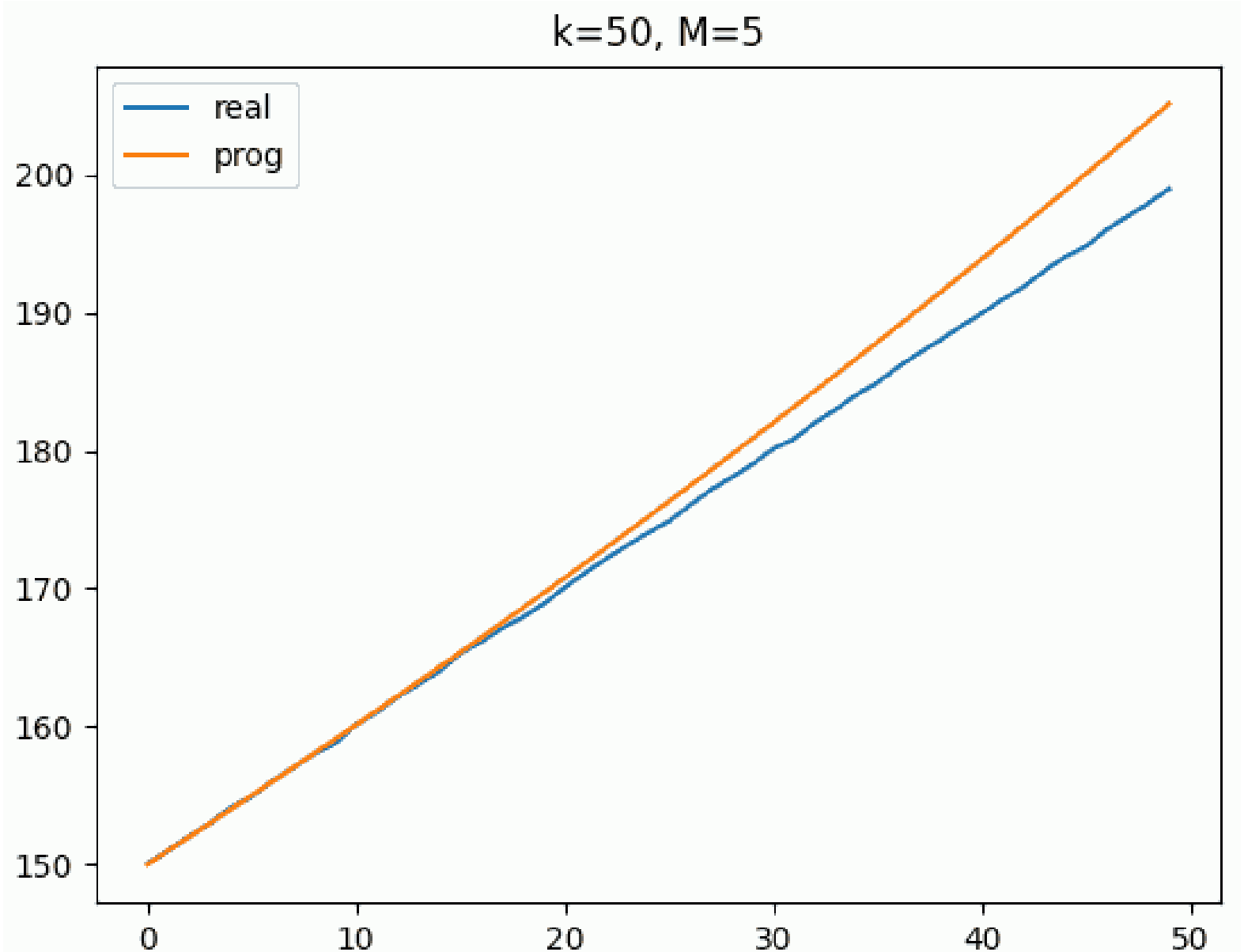
Временного ряда X на k шагов вперёд

```
prognoz_recurr_M = 4      # глобальная переменная
def prognoz_recurr(X, k):
    global prognoz_recurr_M
    M = prognoz_recurr_M  # через сколько шагов предсказывае
    A,b = recurr_equations(X, M)
    w = lin_reg1(b,A) # коэффициенты прогноза
    XX = np.zeros(k+M)
    XX[:M] = X[-M:]
    for i in range(M, M+k):
        XX[i] = np.sum(XX[i-M:i]*w)
    return XX[-k:]
```

Проверка

```
def tst_prognoz_recurr():
    N  = 200                    # длина исходного вектора
    k  = 50                     # на сколько шагов вперед прогно
    prognoz_recurr_M = 5        # сколько предыдущих значений ис
    X = np.arange(N).astype(float)          # временной ряд
    X += np.random.normal(loc=0, scale=0.1, size=N)
    x_real = X[-k:]             # последние реальные k значений
    x_prog = prognoz_recurr(X[:-k], k)   # их же прогноз
    plt.plot(np.arange(k), x_real, label='real')
    plt.plot(np.arange(k), x_prog, label='prog')
    plt.legend()
    plt.title(f'k={k}, M={prognoz_recurr_M}')
    plt.show()
    print(x_real)
    print(x_prog)
```

Проверка



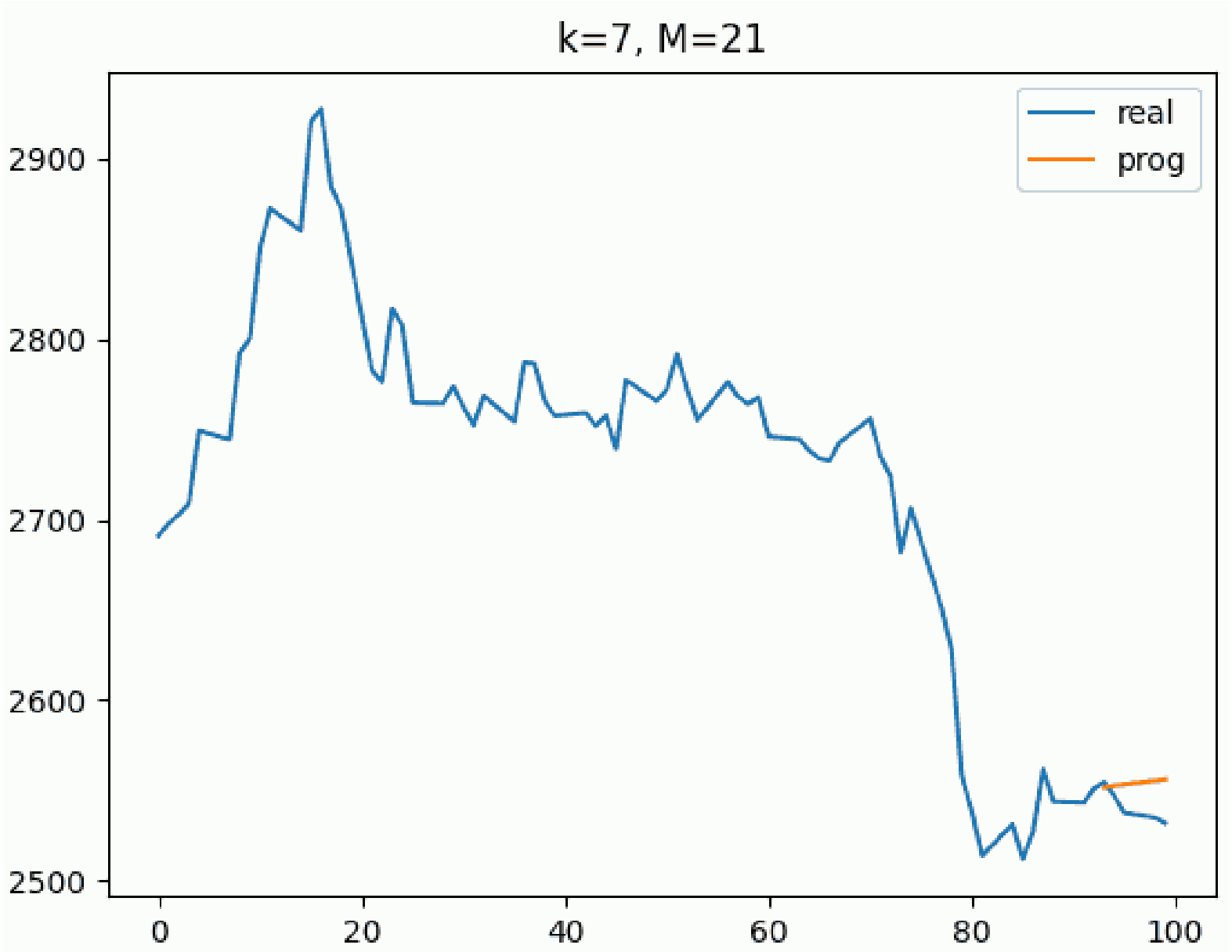
Проверка

```
def tst_prognoz_recurr():
    N  = 200                    # длина исходного вектора
    k  = 10                     # на сколько шагов вперед прогно
    prognoz_recurr_M = 5        # сколько предыдущих значений ис
    X = np.arange(N).astype(float)          # временной ряд
    X += np.random.normal(loc=0, scale=0.1, size=N)
    x_real = X[-k:]             # последние реальные k значений
    x_prog = prognoz_recurr(X[:-k], k)   # их же прогноз
    print(x_real)
    print(x_prog)
    > [189.9826 191.1423 192.1739 193.0329 194.0209  ...]
    > [190.006  190.9406 191.9324 192.9216 193.8935  ...]
    err = quality_prognoz(X, k, prognoz_recurr)
    print(f'err={err:.3f}')
    > [0.0707 0.0786 0.1041 0.0362 0.054 ] 60% 70% 80% 90% 1
    > err=0.069
```

На примере цен на золото

```
def prognoz_recurr_gold(): # Рекуррентный прогноз цен на зо
    X = np.load("time_series\\gold_sb.npz")['data'][:, 0]
    print(X.shape)
    k  = 7                      # на сколько шагов вперед прогноз
    prognoz_recurr_M = 21  # сколько предыдущих значений ис
    sz = 100
    x_real = X[-sz:]          # последние реальные k значений
    x_prog = prognoz_recurr(X[:-k], k)   # их же прогноз
    plt.plot(np.arange(sz), x_real, label='real')
    plt.plot(np.arange(k)+(sz-k), x_prog, label='prog')
    plt.legend()
    plt.title(f'k={k}, M={prognoz_recurr_M}')
    plt.show()
    err0 = quality_prognoz(X, k, prognoz_0)
    err1 = quality_prognoz(X, k, prognoz_recurr)
    print(f'k={k}, M={prognoz_recurr_M}, err0={err0:8.2f}, e
```

На примере цен на золото



Как улучшить?

Пусть дан временной ряд: $x_0, \ldots, x_{N-1}$. Будем искать аппроксимацию каждого элемента через предыдущие ($M = 4$):

$$w_0 x_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 \approx x_4,$$

$$w_0 x_1 + w_1 x_2 + w_2 x_3 + w_3 x_4 \approx x_5,$$

$$w_0 x_2 + w_1 x_3 + w_2 x_4 + w_3 x_5 \approx x_6,$$

$$\ldots$$

Или в матричном виде: $A \cdot w = b$.

Как улучшить?

Раньше мы находили коэффициенты w для прогноза на один шаг вперёд:

$$x_{n+1} = w_0 x_{n-k} + w_1 x_{n-k+1} + \cdots + w_k x_n.$$

Но точно также мы можем находить коэффициенты на два, три шага вперёд:

$$x_{n+3} = w_{3,0} x_{n-k} + w_{3,1} x_{n-k+1} + \cdots + w_{3,k} x_n.$$

Как реализовать?

Напомню, как устроена линейная регрессия. Пусть мы хотим решить систему линейный уравнений $Aw = b$, причем уравнений намного больше, чем переменных, например, 1000 уравнений от 3 переменных. Решений почти наверняка нет. Будем искать приближенное решение: $Aw \approx b$. Или, более точно, ищем вектор w минимизирующий отклонение

$$||Aw - b||^2.$$

Для этого, оказывается, уравнение $Aw = b$ надо умножить на транспонированную матрицу A^t:

$$(A^t \cdot A) \cdot w = A^t \cdot b.$$

В нашем случае матрица $(A^t \cdot A)$ будет иметь размер $3 * 3$ и вектор $(A^t \cdot b)$ длину 3, то есть мы получаем систему 3 уравнений от 3-х переменных.

Как реализовать?

Для прогноза на один шаг мы имели переопределённую систему линейных уравнений $Aw_1 = b_1$. Для прогноза на два шага получим систему $Aw_2 = b_2$ с той же левой частью и так далее. Их можно объединить в одном матричное уравнение:

$$A \cdot W = B,$$

W будет уже не вектор длины M, а набор из k таких векторов, то есть матрицей размера $M \times k$. Точно так же и B будет матрицей того же размера.
Как и ранее, умножив на транспонированную матрицу получим:

$$(A^t \cdot A) \cdot W = A^t \cdot B.$$

Тогда

$$W = (A^t \cdot A)^{-1} A^t \cdot B$$

Как реализовать?

Итак:

$$W = (A^t \cdot A)^{-1} A^t \cdot B$$

Матрица $(A^t \cdot A)^{-1} A^t$ называется псевдообратной к матрице A и обозначается A^+. Она определена даже в случае, когда матрица $(A^t \cdot A)$ вырождена и обратная к ней не существует!
На Питоне, обратная матрица (inverse):

```
A1 = np.linalg.inv(A)
```

псевдообратная матрица (pseudo inverse):

```
A2 = np.linalg.pinv(A)
```

то есть

```
W = np.linalg.pinv(A).dot(B)
```

На Питоне, уравнения

```
def regress_equations(x,M, k):
    '''
    уравнения для простой рекурсии
    :param x: временной ряд
    :param M: через сколько шагов предсказываем
    :param k: на сколько вперёд шагов предсказываем
    :return: (A,B) - система уравнений
    '''
    N = len(x)
    A = np.zeros((N-M-k+1, M))
    B = np.zeros((N-M-k+1, k))
    for i in range(N-M-k+1):
        A[i] = x[i:M+i]
        B[i] = x[M+i:M+i+k]
        #print(i, A[i], B[i])
    return A,B
```

Проверка уравнений

```
def tst_regress_equations():  # уравнения для простой рекурс
    x = np.arange(10)
    M = 2   # через сколько шагов предсказываем
    k = 3   # на сколько вперёд шагов предсказываем
    A,B = regress_equations(x, M, k)
    print(A, '=A\n')
    print(B, '=B\n')
    print(np.hstack((A, B)), '=A,B')
```

Проверка уравнений

№	A	B
0	[0. 1.]	[2. 3. 4.]
1	[1. 2.]	[3. 4. 5.]
2	[2. 3.]	[4. 5. 6.]
3	[3. 4.]	[5. 6. 7.]
4	[4. 5.]	[6. 7. 8.]
5	[5. 6.]	[7. 8. 9.]

Прогноз через регрессию

```
def prognoz_regress(X, k):
    '''
    прогноз через регрессию временного ряда X
    :param X: временной ряд
    :param k: на сколько шагов предсказываем
    :return: прогноз: вектор длины k
    '''
    global prognoz_recurr_M
    M = prognoz_recurr_M # через сколько шагов предсказываем
    A,B = regress_equations(X, M, k)
    W = np.linalg.pinv(A).dot(B)
    return X[-M:].dot(W)
```

Метео прогноз

```
def prognoz_regress_meteo():   # прогноз через регрессию
    X = np.load("time_series\\mpi_roof_2008.npz")['data'][:
    print(X.shape)
    k  = 144                   # на сколько шагов вперед про
    prognoz_recurr_M = k*7     # сколько предыдущих значений

    sz = 500
    x_real = X[-sz:]           # последние реальные k значени
    x_recu = prognoz_recurr(X[:-k], k)    # их же прогноз
    x_regr = prognoz_regress(X[:-k], k)   # их же прогноз
    plt.plot(np.arange(sz), x_real, label='real')
    plt.plot(np.arange(k)+(sz-k), x_recu, label='recurr')
    plt.plot(np.arange(k)+(sz-k), x_regr, label='regr')
    plt.legend()
    plt.title(f'k={k}, M={prognoz_recurr_M}')
    plt.show()
```

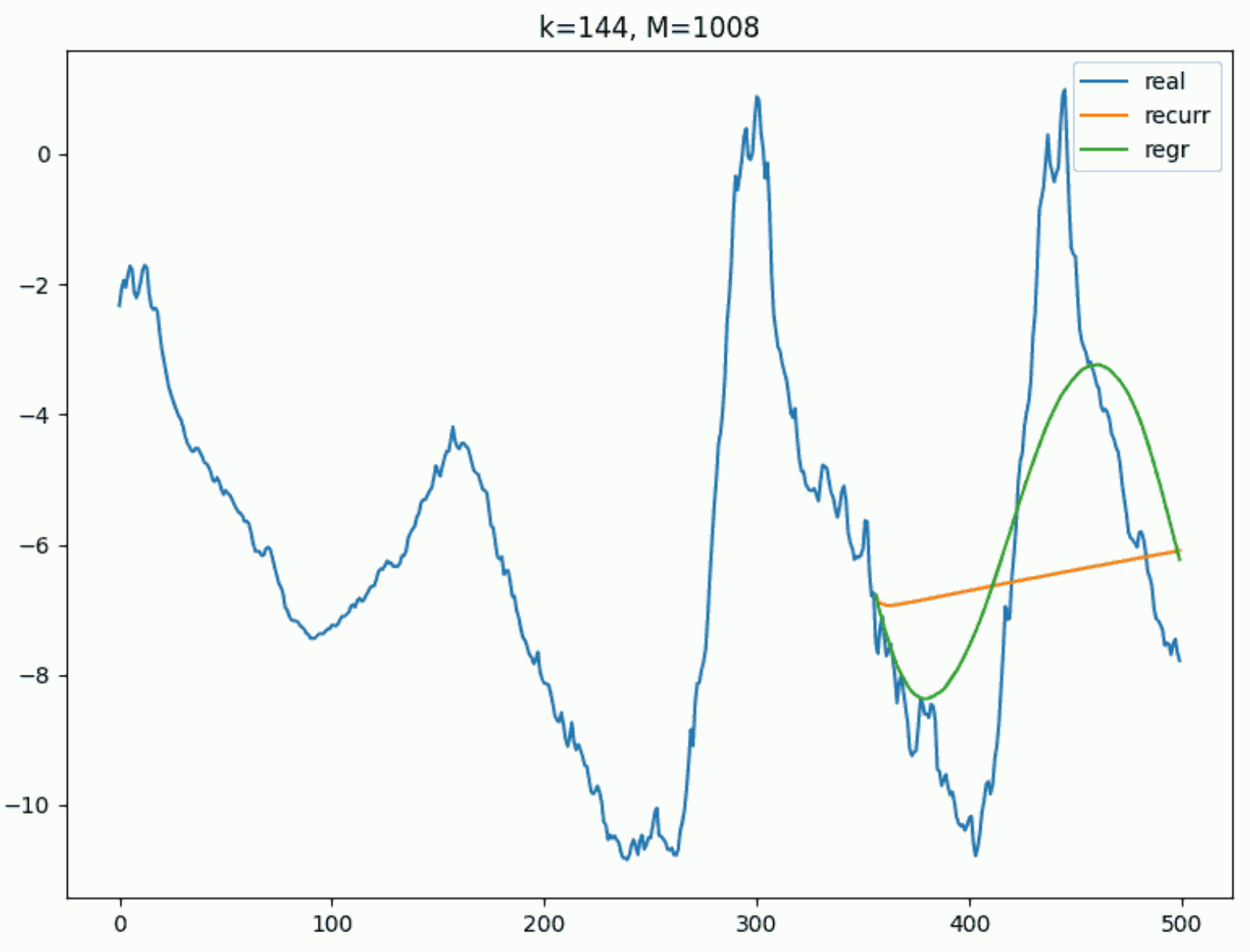
Метео прогноз

```
def prognoz_regress_meteo():   # прогноз через регрессию
    X = np.load("time_series\\mpi_roof_2008.npz")['data'][:
    print(X.shape)
    k  = 144                   # на сколько шагов вперед про
    prognoz_recurr_M = k*7     # сколько предыдущих значений

    sz = 500
    x_real = X[-sz:]           # последние реальные k значени
    x_recu = prognoz_recurr(X[:-k], k)    # их же прогноз
    x_regr = prognoz_regress(X[:-k], k)   # их же прогноз

    err0 = quality_prognoz(X, k, prognoz_0)
    err1 = quality_prognoz(X, k, prognoz_recurr)
    err2 = quality_prognoz(X, k, prognoz_regress)
    print(f'k={k}, M={prognoz_recurr_M}, '\
          f'err0={err0:8.2f}, err1={err1:8.2f}, err2={err2:8
```

Метео прогноз



Метео прогноз

```
[0.5664 0.1952 0.365  0.2515 0.2667] err0
[0.5264 0.217  0.3764 0.3046 0.2549] err1
[0.4191 0.2819 0.2696 0.2227 0.1676] err2
k=144, M=1008,
err0=    0.33, err1=    0.34, err2=    0.27
```

Потребление ЭЭ в Германии

```
X = np.load("time_series\\GermanE.npz")['data'][:, 0]
X = X[1000:-240]
print(X.shape)
k  = 30          # на сколько шагов вперед прогнозируем
prognoz_recurr_M = k*7 # сколько предыдущих значений
```

```
[24.1575 29.902  34.13   30.2496 35.6959] err0
[24.7002 27.0668 27.527  25.5673 30.3332] err1
[23.9151 29.2687 30.1312 12.0079 21.1271] err2
k=30, M=210,
err0=   30.83, err1=   27.04, err2=   23.29
```

Метео прогноз