Задача линейной регрессии в режиме онлайн

Введение

Пусть есть данные в виде $(x_t, y_t),\ \ t = 1, \dots, T$. Причем подается этот кортеж из двух чисел в онлайн режиме(всего T итераций). Нужно по x_t уметь предсказывать $\mathbf{y_t = ?}$, зная предыдущие кортежи: $(x_1, y_1), \dots, (x_{t-1}, y_{t-1})$, т.е. хотим предсказать 2ой элемент кортежа: $(x_t, \mathbf{y_t = ?})$. Предсказывать будем, задавая модель предсказываемой функции - ее параметрический вид, когда параметры неизвестны, но оцениваются по данным.

Наша задача - на каждой итерации t иметь модель, которая будет максимально хорошо предсказывать точку y_t в кортеже (x_t, y_t)

Отметим, что основная особенность этой работы заключается в том, что данные нам будут подаваться в онлайн режиме, а для таких задач разрабона целая теоретическая база: **Online Convex Optimization**. Из этой разработанной теории возьмем алгоритм **Vovk-Azoury-Warmuth Forecaster(VOW)**. Поговорим о линейной регрессии, разберемся с теоретической идеей алгоритма, выведем несколько содержательных вещей из алгоритма *VAW*. Реализуем на языке `python` и применим на разных примерах.

И так, давайте теперь подробнее, формализируем наши желания и сразу разберемся с обозначениями, чтобы вдальнейшем не было путаницы. Даны нам $t - 1 \in \mathbb{N}$ упорядоченных пар и значение $x_t \in \mathbb{R}^d$:

$$(x_1, y_1), \dots, (x_{t-1}, y_{t-1}),\ \text{где}\ \ x_i = \begin{pmatrix} x_i^{(1)} \\ x_i^{(2)} \\ \vdots \\ x_i^{(d)} \end{pmatrix} \in \mathbb{R}^d, y_i \in \mathbb{R}$$

$$x_t = \begin{pmatrix} x_t^{(1)} \\ x_t^{(2)} \\ \vdots \\ x_t^{(d)} \end{pmatrix} \in \mathbb{R}^d$$

Зная все эти данные, алгоритм должен нам предъявить $\tilde{y}_t$

d - порядок модели.

---

1. Подготовка среды для работы

Подключение библиотек

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
sns.set(rc={'figure.figsize':(18.7,8.27)})
```

2. Линейная регрессия

Давайте вначале опустим факт об онлайн режиме и поговорим о линейной регрессии, рассмотрев простой пример.

Пускай нам даны данные заболеваемости людей короновирусом.

Тогда в картеже (x_i, y_i): x_i - номер дня, а y_i - количество заболевших в этот день. Частный случай, где $n = 1$

Для примера будем использовать данные, которые поддаются следующему закону:

$$y_i = 1.5 + 0.3x_i + \xi$$

In [2]:
```
T = 100
```

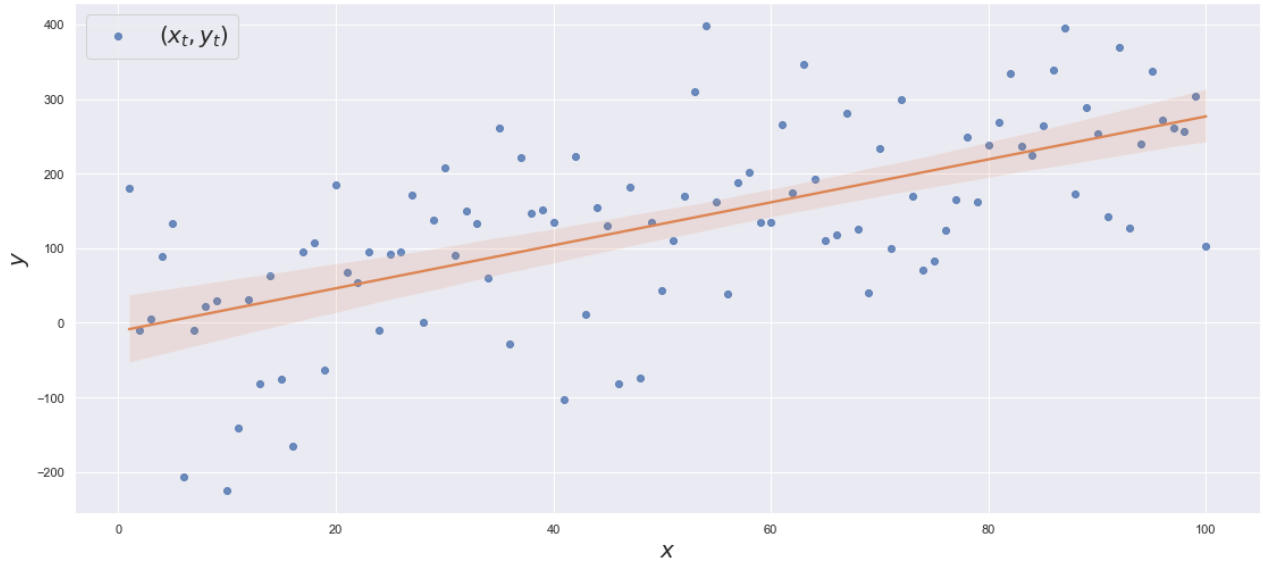
In [3]:
```
x = np.arange(1, T+1, dtype = 'float')
y = 3*x + 100*np.random.normal(0,1, T)
```

In [4]:
```
data_set = pd.DataFrame({'x': x, 'y': y})
data_set.head()
```

Out[4]:

	x	y
0	1.0	180.212195
1	2.0	-9.943453
2	3.0	6.013143
3	4.0	88.645627
4	5.0	133.015924

In [5]:
```
sns.regplot(data = data_set, x = 'x', y = 'y', line_kws={"color": "C1"}, label = '$(x_t, y_t)$')
plt.xlabel('$x$', fontsize=20)
plt.ylabel('$y$', fontsize=20)
plt.legend(fontsize=20);
```



На этом импровизированном графике, на оси абцисс - дни, а на оси ординат - количество заболевших

По этим данным мы хотим построить линейную модель, т.е. просто провести линию, которая будет максимально хорошо характеризовывать точки(и как следствие, если данные линейны, то сможем предсказывать). Искать ее будем с помощью метода наименьших квадратов. Т.е. вычислять расстояние между точкой и проведенной линией по формуле: $(\tilde{y}_i - y_i)^2$

Где $\tilde{y}_i = ax + b$

Таким образом, если мы будем минимизировать сумму всех "потерь" (сумму квадратов), то сможем найти коэффициенты, при которых достигается минимизация.

$$(a, b) = \text{argmin}_{(a,b)} \sum_{i=1}^{T} (y_i - \tilde{y}_i)^2$$

С помощью частных производных, вывели явную формулу для нахождения коэффициентов

$$\text{coeffs} = (a, b) = (X^T X)^{-1} X^T Y$$

$$X = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_T \end{pmatrix}$$

3. Реализация вспомогательных функций

Функция linear_reg_coefs(x, y)

Принимает данные (x_i, y_i) на основе которых мы хотим построить линейную функцию

Возвращает кортеж из двух элементов(коэффициентов): (a, b)

In [6]:
```
def poly1(x, y) -> np.array: #Выход: вектор коэффициентов (a,b)
    X = np.vstack((x**0, x)).T
    poptop = np.linalg.inv(np.dot(X.T, X))
    return np.flip(poptop.dot(X.T).dot(y))
```

Функция poly_create(koeffs)(z)

Принимает коэффициенты полинома: $\text{coeffs} = [a, b, c, d, \dots]$

Возвращает функцию-полином $\text{poly\_create}(\text{coeffs})(\text{z}) = az^n + bz^n \dots$

In [7]:
```
def poly_create(koeffs: np.array) -> np.array:
    def sub_func(z: np.array) -> np.array:
        y = np.zeros(len(z))
        for i, j in enumerate(np.flip(koeffs)):
            y += j*z**i
        return y
    return sub_func
```

Вспомогательная функция linear_reg_func(x, y)(z)

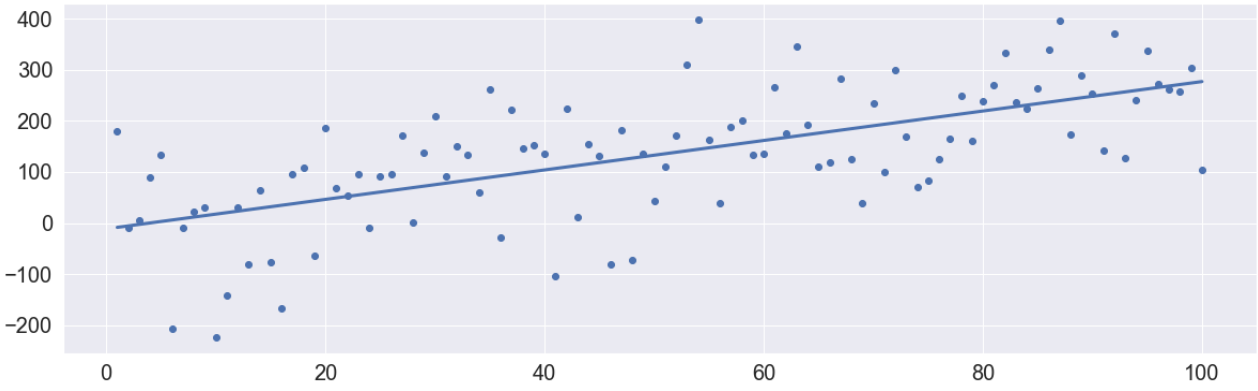
Принимает данные (x_i, y_i)

Возвращает функцию-полином linear_reg_func(x, y)(z) = $az^n + bz^{n-1} + \dots$

In [8]:
```
def linear_reg_func(x, y): # Вход: (x_i, y_i) Выход: линейная функция
    return poly_create(poly1(x, y))#poly_create(linear_reg_coefs(x, y))
```

Визуализация линейной регрессии

In [9]:
```
z = np.linspace(x[0], x[-1], 1000)

plt.figure(figsize=(20, 6))
plt.tick_params(axis='both', which='major', labelsize = 20);
plt.scatter(x, y);

plt.plot(z, linear_reg_func(x, y)(z), linewidth = 3);
```



Теперь давайте вернемся к концепции онлайн режима

Буду следовать источникам:

1. Joulani, A György, C Szepesvári 2020 A modular analysis of adaptive (non-)convex optimization Optimism, composite objectives, variance reduction, and variational bounds
2. Orabona 2020 v5 A Modern Introduction to Online Learning

4. FTRL (Алгоритм следования за регуляризованным лидером)

В процессе доказательства оценки для алгоритма *VAW* используется очень много достаточно содержательных выводов из алгоритма *FTRL*, поэтому невозможно говорить о *VAW* в отрыве от *FTRL*

Введем две последовательности регуляризаторов

$$p_1, \dots, p_T; q_0, \dots, q_T : \mathbb{R}^d \to \mathbb{R} \cup \{+\infty\}$$

Которые могут зависить от $w_1, \dots, w_T; g_1, \dots g_T$. Регуляризаторы p_t предполагаются проксимальными.

т.е. такими, что

$$p_t(w_t) = \inf_{w \in K} p_t(w)$$

Алгоритм заключается в том, что на следующей итерации $t+1$ будем вычислять w_{t+1} по следующему правилу:

$$w_{t+1} \in \underset{w \in K}{\operatorname{argmin}} \left\{ < g_{1:t}, w > + p_{1:t}(w) + q_{0:t}(w) \right\}, \quad t = 0, \dots T$$

где $g_{1:t} = g_1 + \dots + g_t$

Так же введем обозначение:

$$r_t = p_t + q_{t-1}, \quad t = 1, \dots, T$$

Лемма (оценка сожаления FTRL)

Пусть $r_{1:t}$ яаляются 1-сильно выпуклыми относительно нормы $\| \cdot \|_{(t)}$ и $\| \cdot \|_{(t),*}$

Тогда

$$R_T(w) \leq \sum_{t=0}^{T} (q_t(w) - q_t(w_{t+1})) + \sum_{t=0}^{T} (p_t(w) - p_t(w_t)) + \sum_{t=0}^{T} \frac{1}{2} \|g_t\|^2_{(t),*}$$

$\| . \|_{(t),*}$ — это двойственная норма:

$$\|g\|_{(t),*} = \sup \left\{ \langle g, w \rangle \mid \|w\|_{(t)} \leq 1 \right\}$$

Дважды дифференцируемая функция $f : K \subset \mathbb{R}^d \to \mathbb{R}$ называется μ-сильно выпуклой относительно $\| \cdot \|$, если

$$\langle \nabla^2 f(v) w, w \rangle \geq \frac{\mu}{2} \|w\|^2, \quad v \in K, \quad w \in \mathbb{R}^d$$

---

5. Алгоритм Vovk-Azoury-Warmuth

На вход к алгоритму поступает регуляризирущий коэфициент $\lambda > 0$, который влияет на точность предсказания.

Основная суть алгоритма заключается в том, что мы каждой терации t, будем вычислять коэффициенты $\text{coeffs} = (a, b, c \dots)$ модели по следующему принципу:

$$\text{coeffs}_t = \underset{\text{coeffs} \in \mathbb{R}^d}{\operatorname{argmin}} \left(\frac{\lambda}{2} \| \text{ coeffs } \|_2^2 + \frac{1}{2} \sum_{i=1}^{t-1} \left(< \text{coeffs}, x_i > - y_i \right)^2 + \frac{1}{2} < \text{coeffs}, x_t >^2 \right)$$

Примечание: `Заметим, что первое слагаемое соответствует регулязатору от гребневой регрессии, второе - сумме всех потерь до текущей итерации, а третье - просто значение y_t` Т.е. алгоритм будет пытаться минимизировать сумму всех предыдущих потерь и минимизировать текущее значение y_t, а также вносить гребневое измененение, чтобы избежать оверфиттинга

Мы можем переписать алгоритм FTRL в виде:

$$\text{coeffs}_{t+1} = \underset{\text{coeffs} \in \mathbb{R}^d}{\operatorname{argmin}} \left(- \sum_{i=1}^{t} < \text{coeffs}, y_t x_t >^2 + \frac{1}{2} \sum_{i=1}^{t+1} < \text{coeffs}, x_i >^2 + \frac{\lambda}{2} \| \text{ coeffs } \|_2^2 \right)$$

Можно заметить, что он соответствует общей схеме FTRL

$$\text{coeffs}_{t+1} \in \underset{\text{coeffs} \in K}{\operatorname{argmin}} \left\{ < g_{1:t}, \text{coeffs} > + p_{1:t}(\text{coeffs}) + q_{0:t}(\text{coeffs}) \right\}, \quad t = 0, \dots T$$

примененной к последовательности линейных функций

$$\tilde{l}_t(w) = -y_t < x_t, w >$$

с регуляризаторами:

$$p_1 = \cdots = p_T = 0$$
$$q_0 = \frac{1}{2} < \text{coeffs}, x_1 >^2 + \frac{\lambda}{2} \| \text{coeffs} \|_2^2$$
$$q_1 = \frac{1}{2} < \text{coeffs}, x_2 >^2$$
$$\cdots$$
$$q_t = \frac{1}{2} < \text{coeffs}, x_{t+1} >^2, \quad t = 1, \ldots, T-1$$
$$\cdots$$
$$q_T = 0$$

Введу обозначение матрицы:

$$(A_t)_{i,j} = x_t^{(i)} x_t^{(j)}$$

например, если $x_t = \begin{pmatrix} x_t^{(1)} \\ x_t^{(2)} \\ x_t^{(3)} \end{pmatrix}$, то

$$A_t = \begin{pmatrix} x^{(1)}x^{(1)} & x^{(1)}x^{(2)} & x^{(1)}x^{(3)} \\ x^{(2)}x^{(1)} & x^{(2)}x^{(2)} & x^{(2)}x^{(3)} \\ x^{(3)}x^{(1)} & x^{(3)}x^{(2)} & x^{(3)}x^{(3)} \end{pmatrix}$$

Заметим, что мы можем переписать $\langle \text{coeffs}, x_1 \rangle^2 = \langle A_1 \, \text{coeffs}, \text{coeffs} \rangle$:

$$p_1 = \cdots = p_T = 0$$
$$q_0 = \frac{1}{2} < \text{coeffs}, x_1 >^2 + \frac{\lambda}{2} \| \text{coeffs} \|_2^2 = \frac{1}{2} \langle A_1 \, \text{coeffs}, \text{coeffs} \rangle + \frac{\lambda}{2} \| \text{coeffs} \|_2^2$$
$$q_1 = \frac{1}{2} < \text{coeffs}, x_2 >^2 = \frac{1}{2} \langle A_2 \, \text{coeffs}, \text{coeffs} \rangle$$
$$\cdots$$
$$q_t = \frac{1}{2} < \text{coeffs}, x_{t+1} >^2 = \frac{1}{2} \langle A_{t+1} \, \text{coeffs}, \text{coeffs} \rangle, \quad t = 1, \ldots, T-1$$
$$\cdots$$
$$q_T = 0$$

Перепишем через r_t:

$$r_{1:t} = p_{1:t} + q_{0:t-1} = \frac{1}{2} \sum_{i=1}^{t} < \text{coeffs}, x_i >^2 + \frac{\lambda}{2} \| \text{coeffs} \|_2^2$$

$$r_{1:t} = \frac{1}{2} \sum_{i=1}^{t} < \text{coeffs}, \text{coeffs} \cdot A_t >^2 + \frac{\lambda}{2} \| \text{coeffs} \|_2^2$$

Можем легко взять гессиан функции:

$$\nabla^2 r_{1:t} = \lambda I_d + \sum_{i=1}^{t} A_i$$

И как следствие: функция $r_{1:t}$ - является 1-сильно выпуклой относительно нормы $\|w\|_{(t)}$

Обозначим $S_t = \lambda I_d + \sum_{i=1}^{t} A_i$

Можем переписать алгоритм VAW в виде:

$$\text{coeffs}_{t+1} = \underset{\text{coeffs} \in \mathbb{R}^d}{\text{argmin}} \left(-\sum_{i=1}^{t} \langle y_i x_i, \text{coeffs} \rangle + \frac{1}{2} \langle S_{t+1} \, \text{coeffs}, \text{coeffs} \rangle \right) = S_{t+1}^{-1} (\sum_{i=1}^{t} y_i x_i)$$

Т.е. получили явную формулу для вычисления коэффициента на каждой итерации

$$\text{coeffs}_t = \left(\lambda E_d + \sum_{i=1}^{t} x_i x_i^T \right)^{-1} \sum_{i=1}^{t-1} y_i x_i$$

Давайте, все-таки, для удобства, перепишем эту формулу на язык матриц, как в случае с линейной регрессией. Т.к. с ними удобнее работать.

$$\text{coeffs}_t = \left(\lambda E_d + X_t^T X_t\right)^{-1} X_{t-1}^T Y_{t-1}$$

$$X_t = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_t \end{pmatrix} \quad X_{t-1} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_{t-1} \end{pmatrix}$$

Теорема

О

Если $\forall t \in 1, \dots, T \quad \|x_t\|_2 \le R, \quad |y_t| \le Y$, то для алгоритма VAW справедлива оценка сожаления:

$$R_T(\text{coeffs}) \le \frac{\lambda}{2} + \frac{dY^2}{2} \ln(1 + \frac{R^2 T}{\lambda})$$

О

Доказательство

Воспользуемся оценкой сожаления

$$R_T(w) = -\sum_{t=1}^{T} y_t \langle x_t, w_t \rangle + \sum_{t=1}^{T} y_t \langle x_t, w \rangle \le$$
$$\le \sum_{t=0}^{T} \left(q_t(w) - q_t(w_{t+1})\right) + \sum_{t=1}^{T} \frac{1}{2} \|g_t\|_{t,*}^2 =$$
$$= \frac{1}{2}\langle S_T w, w \rangle - \frac{\lambda}{2}\|w_1\|_2^2 - \frac{1}{2}\sum_{t=1}^{T} \langle A_t w_t, w_t \rangle + \frac{1}{2}\sum_{t=1}^{T} \|y_t x_t\|_{S_t^{-1}}$$

Группируем слагаемые

$$\frac{1}{2}\sum_{t=1}^{T} \langle A_t w_t, w_t \rangle - y_t \langle x_t, w_t \rangle - \frac{1}{2}\sum_{t=1} T y_t \langle x_t, w \rangle \le$$
$$\frac{\lambda}{2}\|w\|_2^2 - \frac{\lambda}{2}\|w_1\| - \frac{1}{2}\sum_{t=1}^{T} y_t \langle S_t x_t, x_t \rangle$$

Или иначе:

$$\frac{1}{2}\sum_{t=1}^{T} \langle x_t, w_t \rangle^2 - \sum_{t=1}^{T} y_t \langle x_t, w_t \rangle - \frac{1}{2}\sum_{t=1}^{T} \langle x_t, w \rangle^2 + \sum_{t=1}^{T} y_t \langle x_t, w_t \rangle =$$
$$\frac{1}{2}\sum_{t=1}^{T} (\langle x_t, w \rangle - y_t)^2 - \frac{1}{2}\sum_{t=1}^{T} (\langle x_t, w \rangle - y_t) =$$
$$R_T(w) \le \frac{\lambda}{2}\|w\|_2^2 - \frac{\lambda}{2}\|w_1\|_2^2 + \frac{1}{2}\sum_{t=1}^{T} y_t^2 \langle S_t x_t, x_t \rangle$$

Лемма

Пусть $A \succeq B \succeq 0$, тогда

$$A^{-1} * (A - B) \le \ln \frac{\det A}{\det B} \quad A * B := \sum_{i,j=1}^{d} A_{ij} B_{ij} = Tr(AB^T)$$

Поскольку $S_t - S_{t-1} = A_t$, то используя лемму, находим

$$\sum_{t=1}^{T} y_t^2 \langle S_t^{-1} g_t, g_t \rangle \leq Y^2 \sum_{t=1}^{T} S_t^{-1} * A_t = Y^2 \sum_{t=1}^{T} S_t^{-1} * (S_t - S_{t-1}) \leq Y^2 \sum_{t=1}^{T} \ln \frac{\det S_t}{\det S_{t-1}} = Y^2 \ln \frac{\det S_T}{\det S_0}$$

Из оценки квадратичной формы

$$\langle S_T w, w \rangle = \lambda \|w\|_2^2 + \sum_{t=1}^{T} \langle A_t w, w \rangle = \lambda \|w\|_2^2 + \sum_{t=1}^{T} \langle x_t w, x_t w \rangle \leq \lambda \|w\|_2^2 + \gamma \sum_{t=1}^{T} \|x_t\|_2^2 \|w\|_2^2 \leq (\lambda + R^2 T) \|w\|_2^2$$

Вытекает, что собственные числа S_T не превосходят $\lambda + R^2 T$ и значит

$$\det S_T \leq (\lambda + R^2 T)^d, \quad \det S_0 = \lambda^d$$

Таким образом,

$$R_T(w) \leq \frac{\lambda}{2} \|w\|_2^2 - \frac{\lambda}{2} \|w_1\|_2^2 + \frac{1}{2} \sum_{t=1}^{T} y_t^2 \langle S_t x_t, x_t \rangle \leq \frac{\lambda}{2} \|w\|_2^2 + \frac{Y^2}{2} \ln \frac{(\lambda + R^2 T)^d}{\lambda^d} = \frac{\lambda}{2} \|w\|_2^2 + \frac{dY^2}{2} \ln \left(1 + \frac{R^2 T}{\lambda}\right)$$

Что и требовалось доказать

Обозначения:

Где $R_T(\text{coeffs})$ - функция сожаления:

$$R_T(\text{coeffs}) = \sum_{t=1}^{T} l(\text{coeffs}^*) - \sum_{t=1}^{T} l(\text{coeffs}_\text{t}))$$

Где l - функция потери

$$l(\text{coeffs}_\text{t}) = (< x_t, \text{coeffs}_\text{t} > - y_t)^2 = (\tilde{y}_t - y_t)^2$$

coeffs^* - самое лучшее решение, если бы мы знали заранее всю информацию, которое бы максимально минимизировало все потери на всех итерациях

$$\text{coeffs}^* = \underset{\text{coeffs} \in \mathbb{R}^d}{\text{argmin}} \sum_{t=1}^{T} l(\text{coeffs}))$$

6. Реализация алгоритма VAW

In [10]:
```
def VAW(data_set, l):
    x, y       = data_set['x'], data_set['y']
    T          = len(x)
    coeffs     = np.zeros((T, 4))
    loss       = np.zeros(T)
    y_pred_lst = np.zeros(T)
    for t in range(2, T):
        x_t1 = x[:t] #вектор иксов до x_{t-1} значения
        y_t1 = y[:t] #вектор игриков до x_{t-1} значения
        x_t  = x[:t+1] #вектор иксов до x_t значения
        X_t1 = np.vstack((x_t1**0, x_t1, x_t1**2, x_t1**3)).T #матрица исксов до x_{t-1} значения
        X_t  = np.vstack((x_t**0, x_t, x_t**2, x_t**3)).T #матрица исксов до x_{t-1} значения

        poptop = np.linalg.inv(l*np.eye(4) + np.dot(X_t.T, X_t)) #Обратная матрица

        #финальное вычисление:
        coeffs[t] = poptop.dot(X_t1.T).dot(y_t1)
        y_pred_lst[t] = coeffs[t].dot(X_t[t])
        loss[t] = np.sum((y_pred_lst[:t+1] - y[:t+1])**2)

    return coeffs, loss, y_pred_lst #возвращает коэффициенты coeffs = (a, b, c, ...), где a + bx + cx^2 + ...
```

Для начала пусть параметр будет $\lambda = 0.5$:

In [11]:
```
l = 0.5
```

In [12]:
```
coeffs_lst, loss_lst, y_pred_lst = VAW(data_set, l)
```

In [13]:
```
data_set['$loss_1$'], data_set['$\widetilde y_1$'] = loss_lst, y_pred_lst
```

In [14]:
```
data_set
```

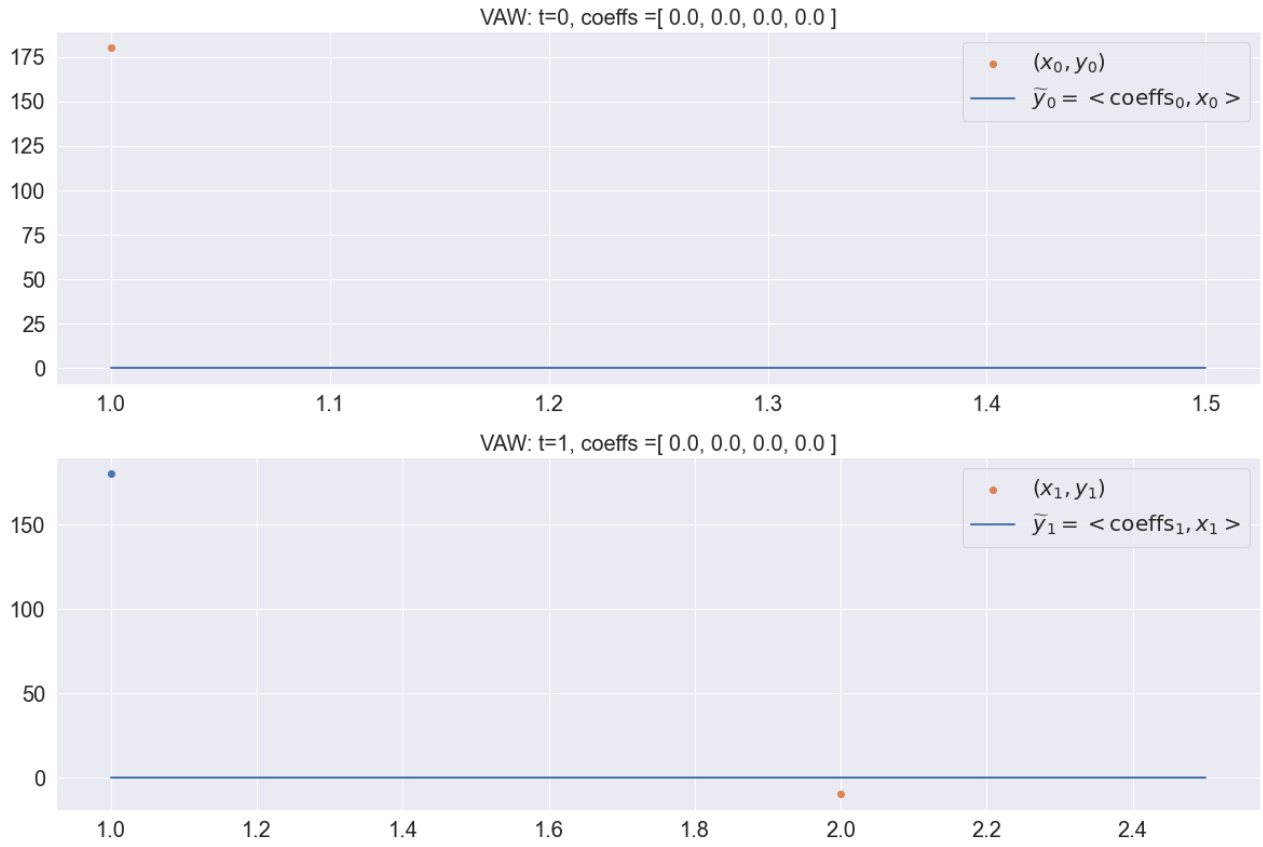
Out[14]:

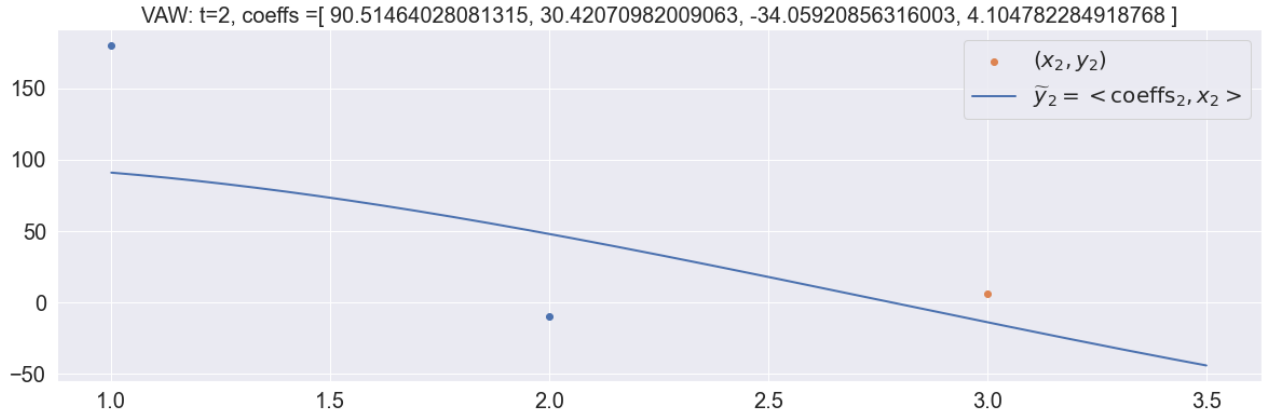
	x	y	$loss_1$	$\tilde{y}_1$
0	1.0	180.212195	0.000000e+00	0.000000
1	2.0	-9.943453	0.000000e+00	0.000000
2	3.0	6.013143	3.297292e+04	-13.926986
3	4.0	88.645627	4.122241e+04	-2.181112
4	5.0	133.015924	5.354994e+04	21.986505
...	...	...	...	...
95	96.0	271.723388	1.407031e+06	230.339538
96	97.0	261.961141	1.407846e+06	233.418939
97	98.0	257.321591	1.408357e+06	234.712653
98	99.0	303.255048	1.413001e+06	235.109700
99	100.0	103.292687	1.432138e+06	241.629933

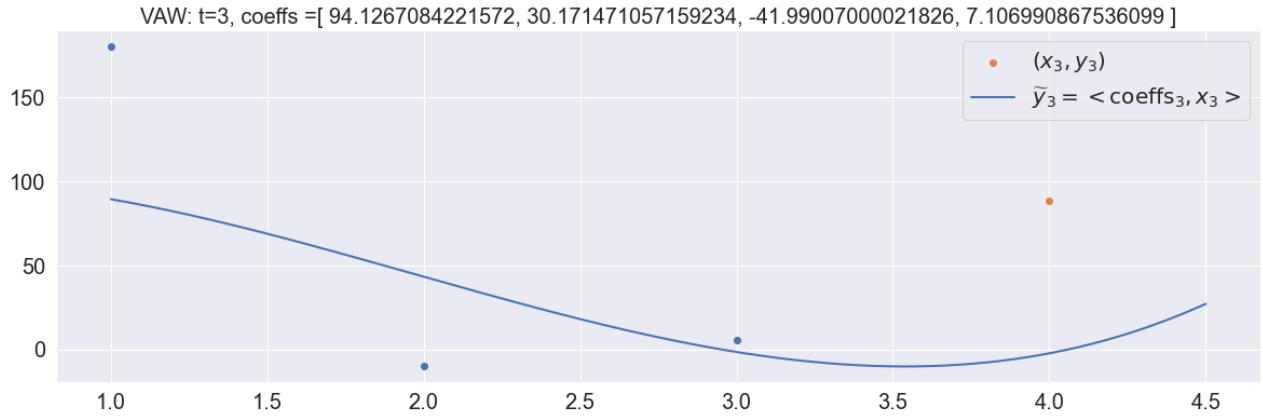
100 rows × 4 columns

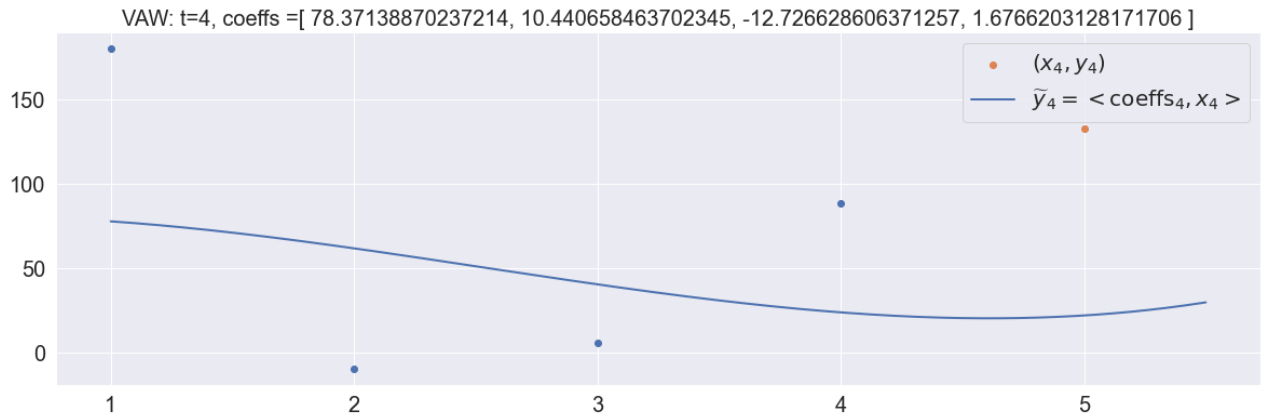
7. Графическое представление работы алгоритма VAW:

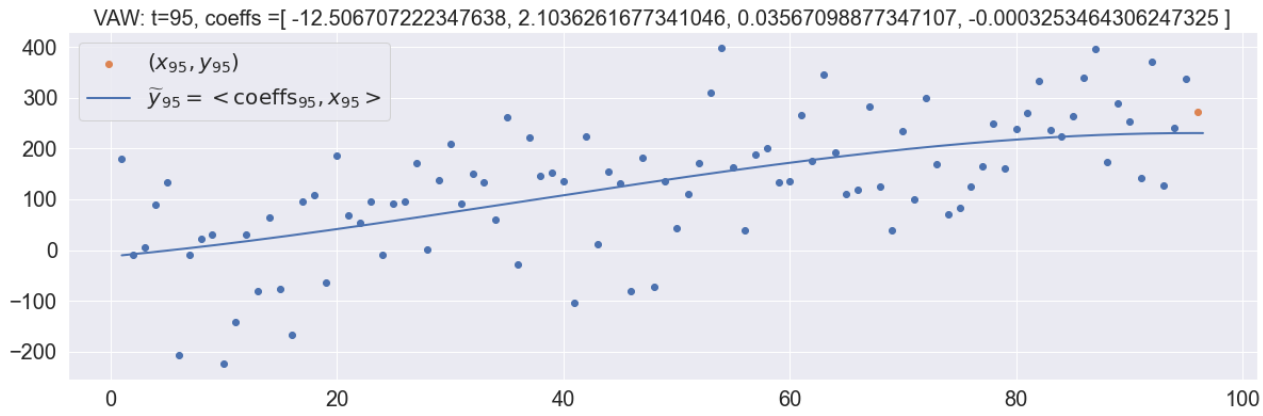
In [15]:
```
for j, i in enumerate(coeffs_lst):
    if not 4<j<95:
        z = np.linspace(x[0], x[j]+0.5, 1000)
        plt.figure(figsize=(20, 6))
        plt.tick_params(axis='both', which='major', labelsize = 20);
        plt.scatter(x[:j], y[:j]);
        plt.scatter(x[j], y[j], label = f'$(x_{ {j} }, y_{ {j} })$');
        plt.plot(z, i[0]+i[1]*z+i[2]*z**2+i[3]*z**3,
                 linewidth = 2,
                 label = f'$\widetilde y_{ {j} } = <\operatorname{{coeffs}}_{ {j} }, x_{ {j} }>$');

        plt.legend(fontsize=20)
        plt.title(f'VAW: t={j}, coeffs =[ {i[0]}, {i[1]}, {i[2]}, {i[3]} ]', size = 20)
```

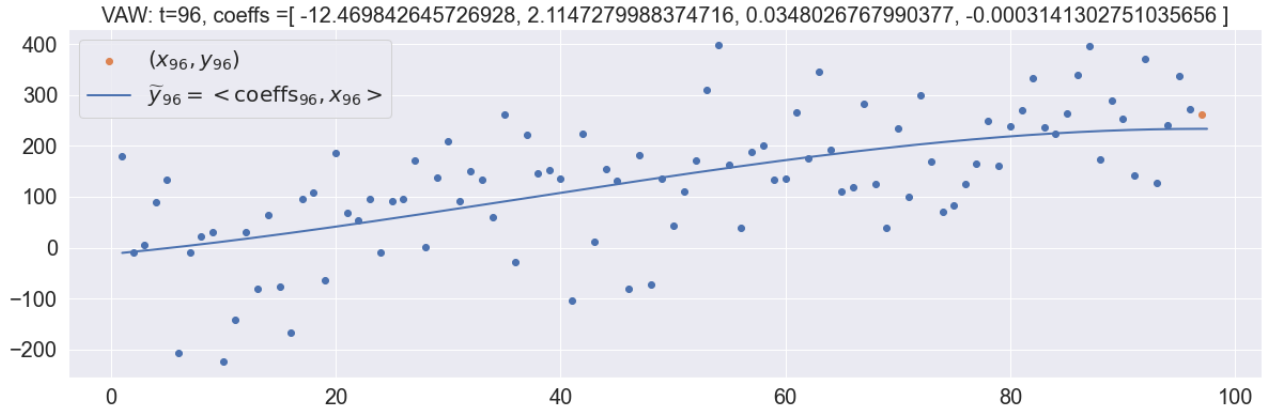


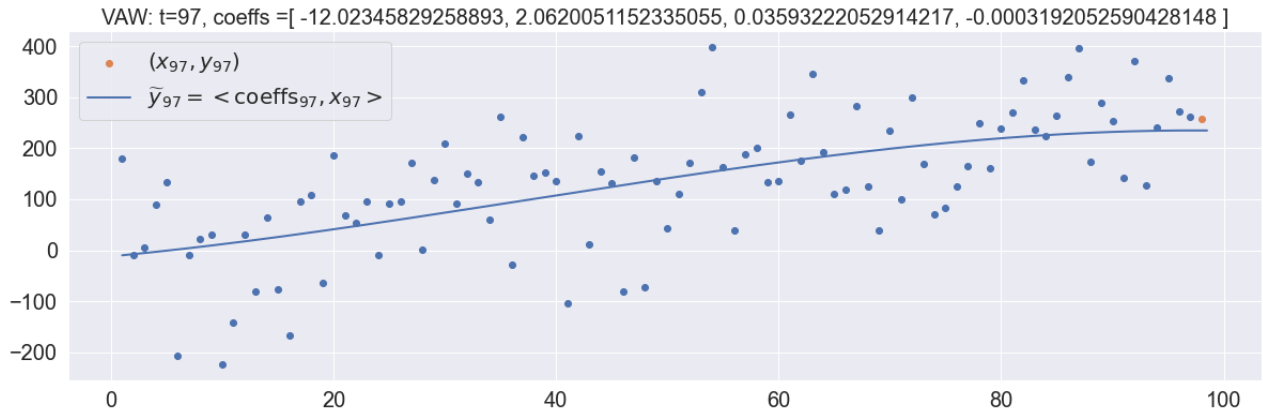

VAW: t=2, coeffs =[90.51464028081315, 30.42070982009063, -34.05920856316003, 4.104782284918768]
(x_2, y_2)
$\widetilde{y}_2 = <\text{coeffs}_2, x_2>$

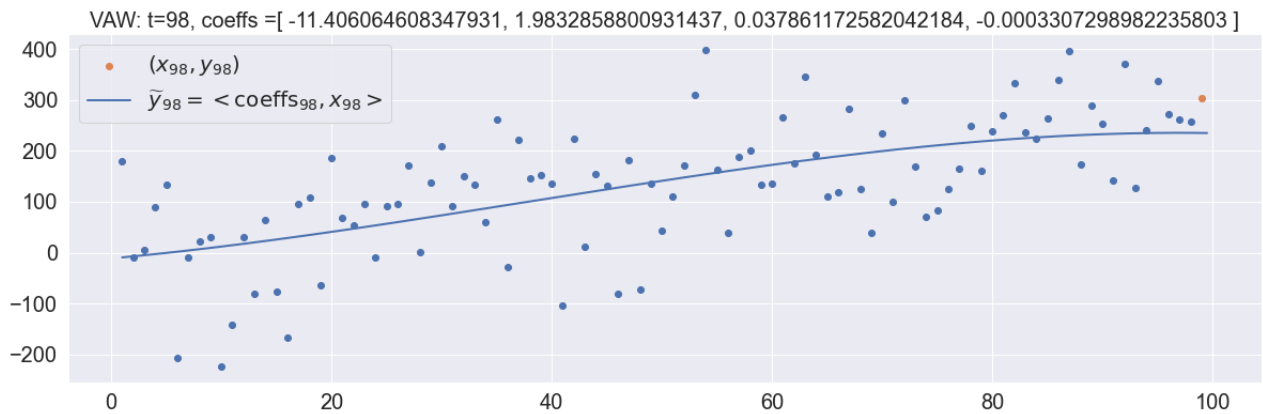

VAW: t=3, coeffs =[94.1267084221572, 30.171471057159234, -41.99007000021826, 7.106990867536099]
(x_3, y_3)
$\widetilde{y}_3 = <\text{coeffs}_3, x_3>$

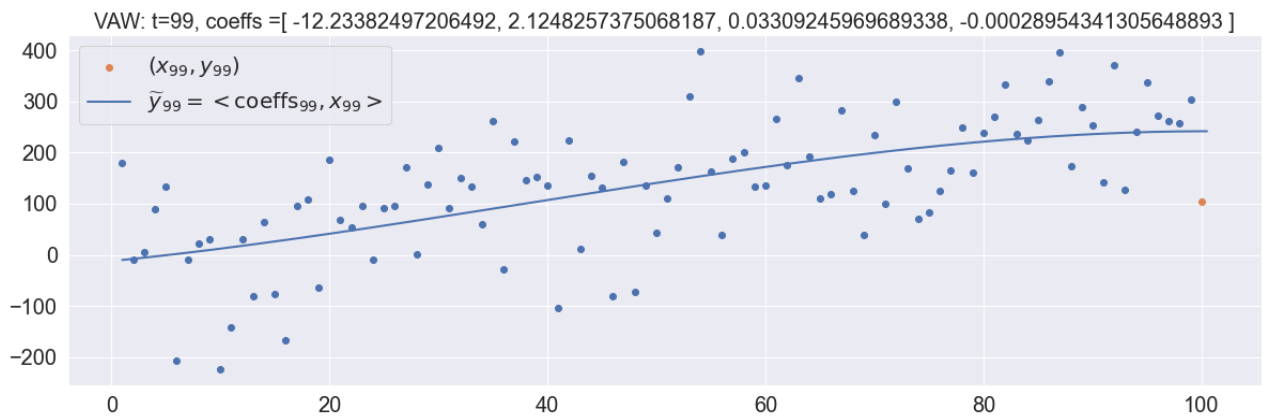

VAW: t=4, coeffs =[78.37138870237214, 10.440658463702345, -12.726628606371257, 1.6766203128171706]
(x_4, y_4)
$\widetilde{y}_4 = <\text{coeffs}_4, x_4>$


VAW: t=95, coeffs =[-12.506707222347638, 2.1036261677341046, 0.03567098877347107, -0.0003253464306247325]
(x_{95}, y_{95})
$\widetilde{y}_{95} = <\text{coeffs}_{95}, x_{95}>$









Покажем, что увеличении параметра λ функция будет все больше и больше стремиться быть похожей на горизонтальную прямую т.к. коэффициенты модели будут уменьшаться

8. Влияние параметра λ на оверфиттинг

Например, если $\lambda = 10^{15}$, то полином будет почти, как прямая:

In [16]:
```
l = 10**14
```

In [17]:
```
coeffs_lst, loss_lst, y_pred_lst = VAW(data_set, l)
```

In [18]:
```
data_set['$loss_2$'], data_set['$\widetilde y_2$'] = loss_lst, y_pred_lst
```

Таблица с вычисленными потерями и вычисленными значениями

In [19]:
```
data_set
```

Out[19]:

	x	y	$loss_1$	$\tilde{y}_1$	$loss_2$	$\tilde{y}_2$
0	1.0	180.212195	0.000000e+00	0.000000	0.000000e+00	0.000000e+00
1	2.0	-9.943453	0.000000e+00	0.000000	0.000000e+00	0.000000e+00
2	3.0	6.013143	3.297292e+04	-13.926986	3.261147e+04	4.633134e-11
3	4.0	88.645627	4.122241e+04	-2.181112	4.046951e+04	2.083589e-10
4	5.0	133.015924	5.354994e+04	21.986505	5.816275e+04	7.852943e-09
...	...	...	...	...	...	...
95	96.0	271.723388	1.407031e+06	230.339538	3.042478e+06	3.525967e+01
96	97.0	261.961141	1.407846e+06	233.418939	3.092608e+06	3.806220e+01
97	98.0	257.321591	1.408357e+06	234.712653	3.139430e+06	4.093758e+01
98	99.0	303.255048	1.413001e+06	235.109700	3.206685e+06	4.391967e+01
99	100.0	103.292687	1.432138e+06	241.629933	3.209805e+06	4.743320e+01

100 rows × 6 columns

Коэффициенты:

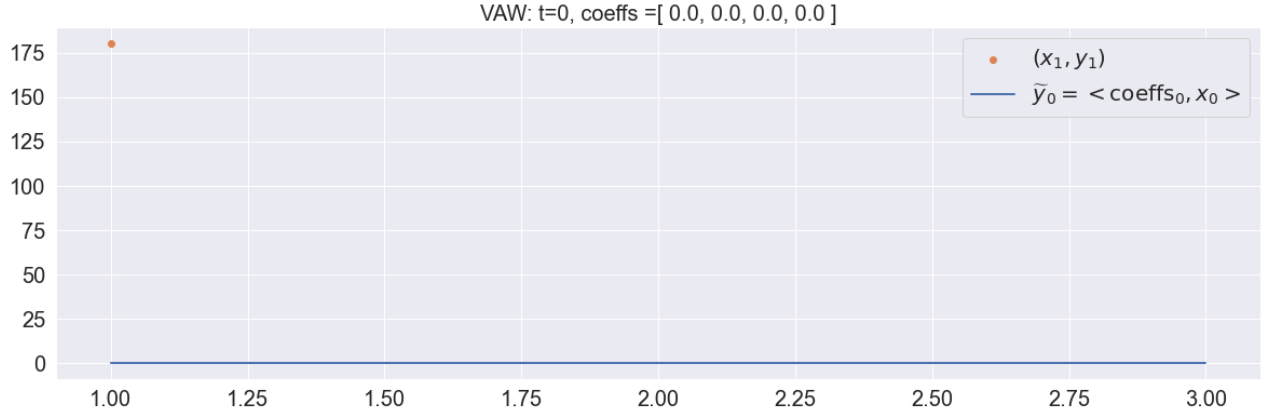
In [20]:
```
pd.DataFrame(coeffs_lst)
```

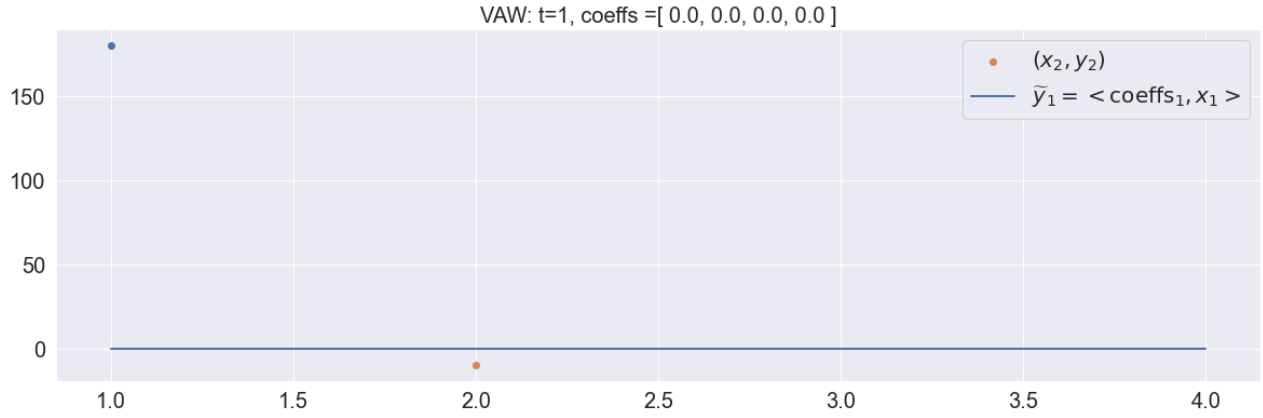
Out[20]:

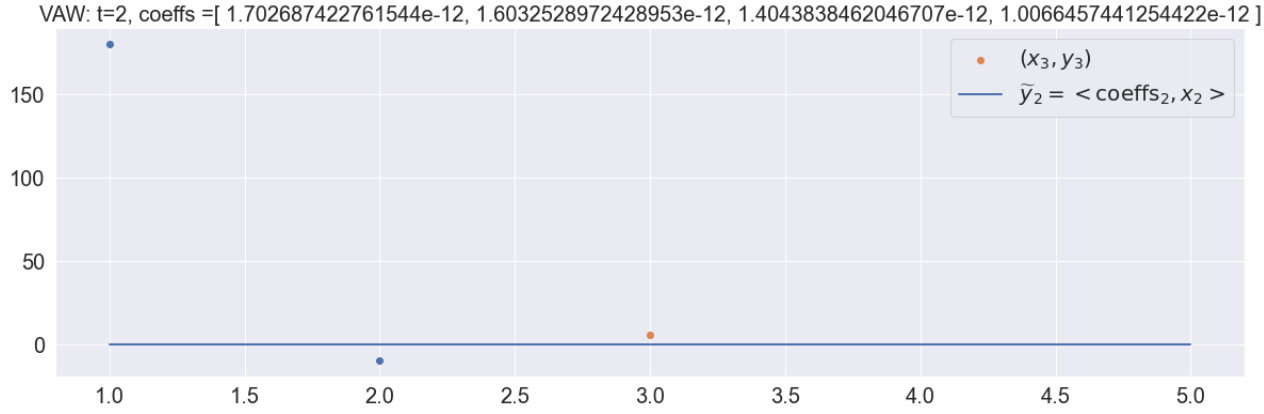
	0	1	2	3
0	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
1	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
2	1.702687e-12	1.603253e-12	1.404384e-12	1.006646e-12
3	1.762819e-12	1.783647e-12	1.945567e-12	2.630194e-12
4	2.649275e-12	5.329472e-12	1.612887e-11	5.936340e-11
...	...	...	...	...
95	1.135537e-10	7.340025e-09	5.240682e-07	3.984786e-05
96	1.154891e-10	7.532984e-09	5.430382e-07	4.169850e-05
97	1.172946e-10	7.715401e-09	5.611898e-07	4.348973e-05
98	1.190124e-10	7.891300e-09	5.789065e-07	4.525820e-05
99	1.210391e-10	8.101781e-09	6.003773e-07	4.742720e-05

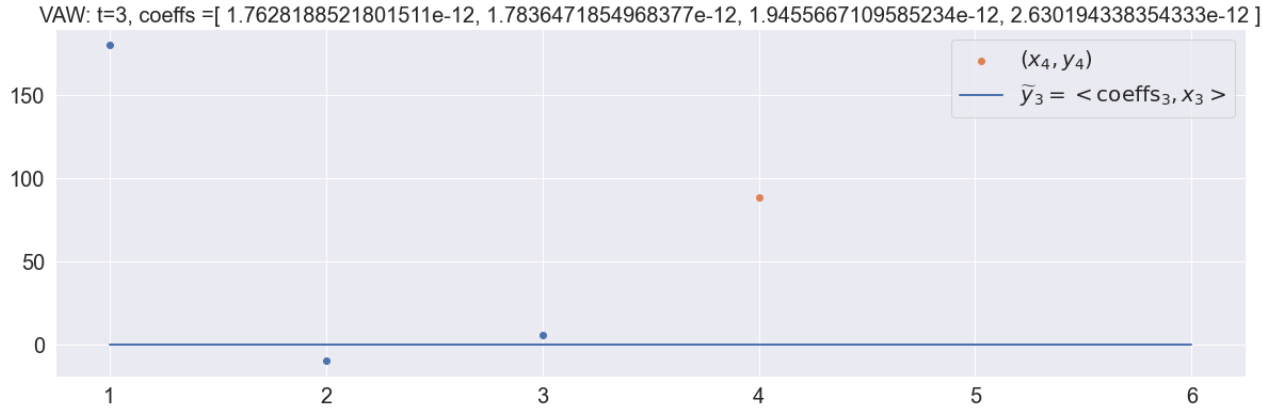
100 rows × 4 columns

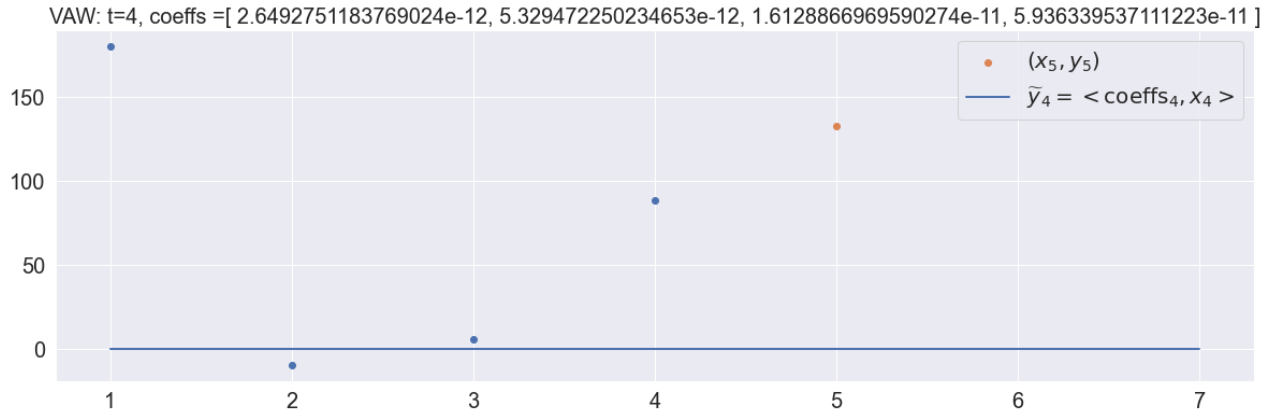
In [21]:
```
for j, i in enumerate(coeffs_lst):
    if not 4<j<95:
        z = np.linspace(x[0], x[j]+2, 1000)
        plt.figure(figsize=(20, 6))
        plt.tick_params(axis='both', which='major', labelsize = 20);
        plt.scatter(x[:j], y[:j]);
        plt.scatter(x[j], y[j], label = f'$(x_{ {j+1} }, y_{ {j+1} })$');
        plt.plot(z, i[0]+i[1]*z+i[2]*z**2+i[3]*z**3, linewidth = 2,
                 label = f'$\widetilde y_{ {j} } = <\operatorname{{coeffs}}_{ {j} }, x_{ {j} }>$');

        plt.legend(fontsize=20)
        plt.title(f'VAW: t={j}, coeffs =[ {i[0]}, {i[1]}, {i[2]}, {i[3]} ]', size = 20)
```

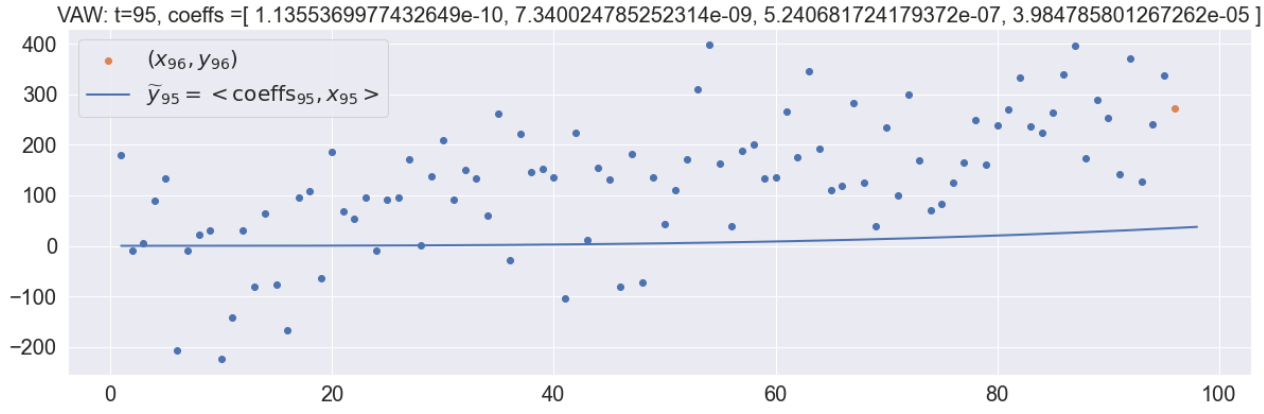


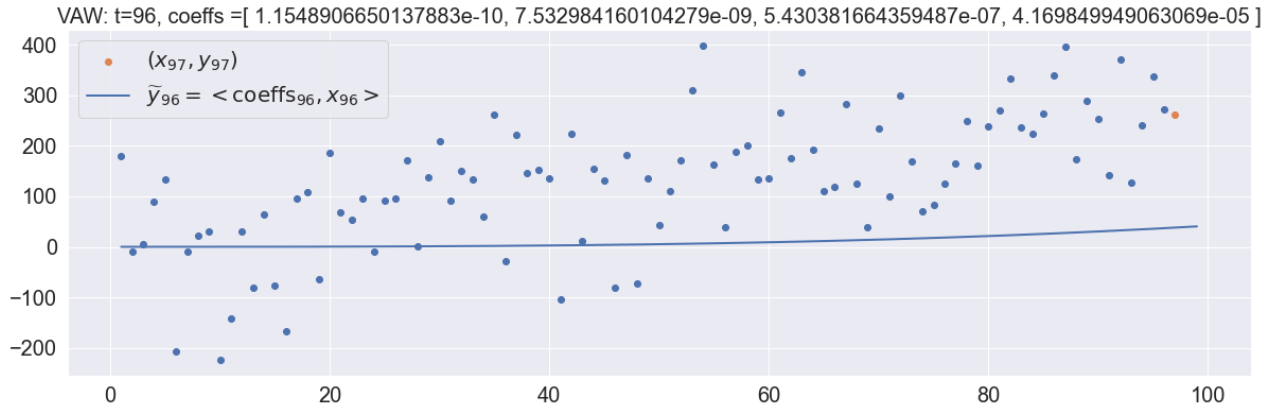

VAW: t=1, coeffs =[0.0, 0.0, 0.0, 0.0]
(x_2, y_2)
$\widetilde{y}_1 = <\text{coeffs}_1, x_1>$
150
100
50
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0

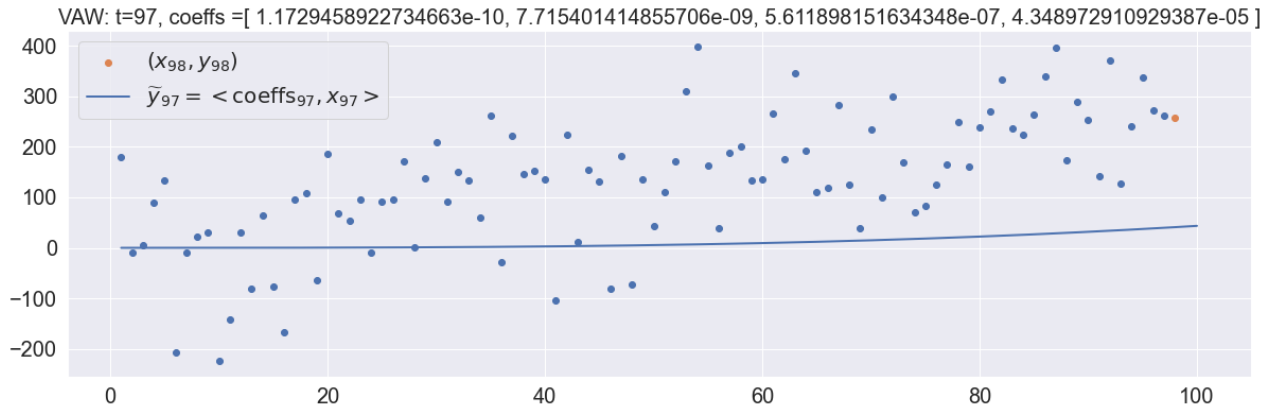

VAW: t=2, coeffs =[1.702687422761544e-12, 1.6032528972428953e-12, 1.4043838462046707e-12, 1.0066457441254422e-12]
(x_3, y_3)
$\widetilde{y}_2 = <\text{coeffs}_2, x_2>$
150
100
50
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

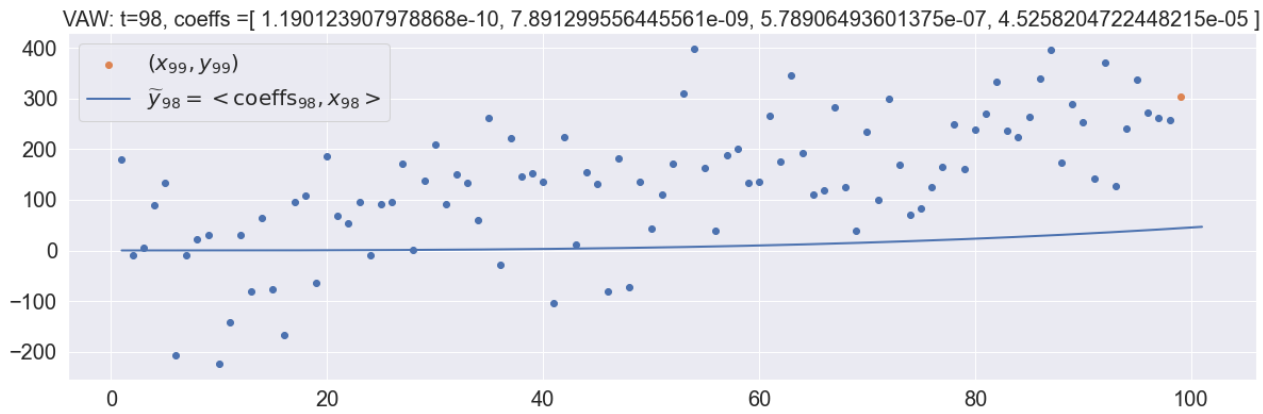

VAW: t=3, coeffs =[1.7628188521801511e-12, 1.7836471854968377e-12, 1.9455667109585234e-12, 2.630194338354333e-12]
(x_4, y_4)
$\widetilde{y}_3 = <\text{coeffs}_3, x_3>$
150
100
50
0
1
2
3
4
5
6

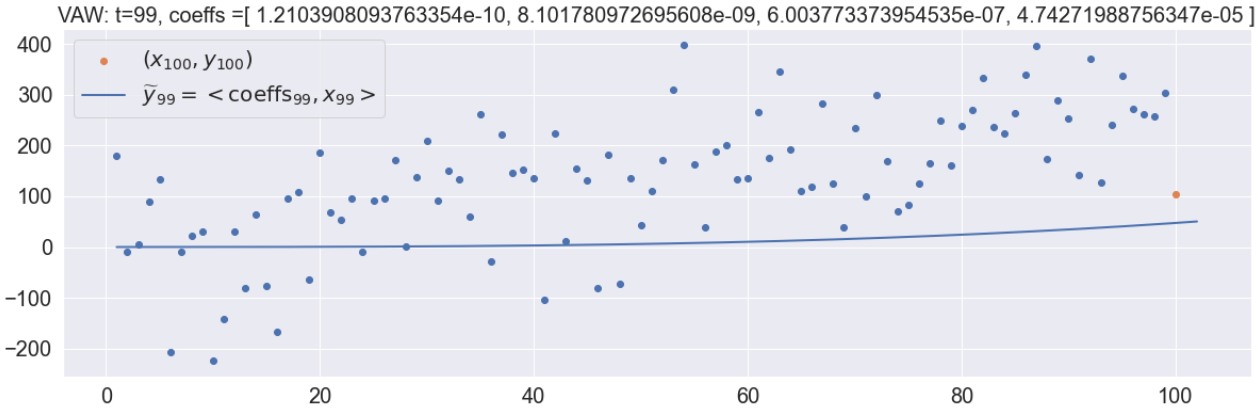

VAW: t=4, coeffs =[2.6492751183769024e-12, 5.329472250234653e-12, 1.6128866969590274e-11, 5.936339537111223e-11]
(x_5, y_5)
$\widetilde{y}_4 = <\text{coeffs}_4, x_4>$
150
100
50
0
1
2
3
4
5
6
7


VAW: t=95, coeffs =[1.1355369977432649e-10, 7.340024785252314e-09, 5.240681724179372e-07, 3.984785801267262e-05]
(x_{96}, y_{96})
$\widetilde{y}_{95} = <\text{coeffs}_{95}, x_{95}>$


VAW: t=96, coeffs =[1.1548906650137883e-10, 7.532984160104279e-09, 5.430381664359487e-07, 4.169849949063069e-05]
(x_{97}, y_{97})
$\widetilde{y}_{96} = <\text{coeffs}_{96}, x_{96}>$


VAW: t=97, coeffs =[1.1729458922734663e-10, 7.715401414855706e-09, 5.611898151634348e-07, 4.348972910929387e-05]
(x_{98}, y_{98})
$\widetilde{y}_{97} = <\text{coeffs}_{97}, x_{97}>$


VAW: t=98, coeffs =[1.190123907978868e-10, 7.891299556445561e-09, 5.78906493601375e-07, 4.5258204722448215e-05]
(x_{99}, y_{99})
$\widetilde{y}_{98} = <\text{coeffs}_{98}, x_{98}>$



Как и ожидалось, если $\lambda = 10^{15}$, то полином будет почти, как прямая

9. Метод экспертов для нахождения наилучшего λ в онлайн режиме

$$w_{i,t} = \frac{w_{i,\,t-1}e^{-\eta \cdot l_{i,\,t}}}{\sum\limits_{j=1}^{N} w_{j,\,t-1}e^{-\eta \cdot l_{j,\,t-1}}}$$

Суть в том, что будет запущено сразу несколько алгоритмов VAW с разными значениями $\lambda_1, \dots \lambda_N$. Предварительно на каждую λ_i равномерно распределим стартовые веса, которые впоследствии, в онлайн режиме будут изменяться. Чем больше вес - тем больше мы будем доверять коэффициенту λ_i и наоборот. На практике будем просто выбирать ту λ_i, которой будет соответствовать больший вес

Зафиксируем интересующий нас набор $\lambda_1, \dots \lambda_N$

В данной формуле $w_{1,t}, \dots w_{N,t}$ это веса для $\lambda_1, \dots \lambda_N$ соответственно, где $t \in 1, \dots T$

l_t это функция потерь

η это коэффиент метода экспертов

Начальные веса: $\frac{1}{N}$

Теорема (об оценке)

$$\hat{L}_n - \min_{i=1,\dots,N} L_{i,n} \leq \frac{\ln N}{\eta} + \frac{n\eta}{8}$$

Где $\hat{L}_n$ - сумма потерь $\min_{i=1,\dots,N} L_{i,n}$ - самая маленькая потеря при наилучшем выборе парметра λ

Так как тестировать будем всего на двух парамтерах, то и распределим веса по 0.5 на каждый параметр

10. Реализация предсказания с советами экспертов для задачи линейной регрессии

In [22]:
```
data_set['$w_1$'] = 0.5
data_set['$w_2$'] = 0.5
```

In [23]:
```
data_set
```

Out[23]:

	x	y	$loss_1$	$\tilde{y}_1$	$loss_2$	$\tilde{y}_2$	w_1	w_2
0	1.0	180.212195	0.000000e+00	0.000000	0.000000e+00	0.000000e+00	0.5	0.5
1	2.0	-9.943453	0.000000e+00	0.000000	0.000000e+00	0.000000e+00	0.5	0.5
2	3.0	6.013143	3.297292e+04	-13.926986	3.261147e+04	4.633134e-11	0.5	0.5
3	4.0	88.645627	4.122241e+04	-2.181112	4.046951e+04	2.083589e-10	0.5	0.5
4	5.0	133.015924	5.354994e+04	21.986505	5.816275e+04	7.852943e-09	0.5	0.5
...	...	...	...	...	...	...	...	...
95	96.0	271.723388	1.407031e+06	230.339538	3.042478e+06	3.525967e+01	0.5	0.5
96	97.0	261.961141	1.407846e+06	233.418939	3.092608e+06	3.806220e+01	0.5	0.5
97	98.0	257.321591	1.408357e+06	234.712653	3.139430e+06	4.093758e+01	0.5	0.5
98	99.0	303.255048	1.413001e+06	235.109700	3.206685e+06	4.391967e+01	0.5	0.5
99	100.0	103.292687	1.432138e+06	241.629933	3.209805e+06	4.743320e+01	0.5	0.5

100 rows × 8 columns

Подсчет потерь

In [24]:
```
data_set['$loss_1$'].sum()
```

Out[24]:
```
76081908.76125334
```

In [25]:
```
data_set['$loss_2$'].sum()
```

Out[25]:
```
116590401.70522618
```

In [26]:
```
w_0 = data_set['$w_1$']
w_1 = data_set['$w_2$']
loss_0 = data_set['$loss_1$']
loss_1 = data_set['$loss_2$']
```

In [27]:
```
eta = 0.00001 #коэффициент метода экспертов
N = 2 #Количество экспертов
T #Количество итераций

w_lst = [w_0, w_1]
loss_lst = [loss_0, loss_1]

for t in range(2, T-1):
    sum_of_w = sum([w_lst[i][t]*np.exp(-eta*loss_lst[i][t]) for i in range(N)])
    for i in range(N):
        w_lst[i][t+1] = w_lst[i][t]*np.exp(-eta*loss_lst[i][t])/sum_of_w
```

В таблице видно, что со временем веса $w_{0.5}$, которые относятся к параметру $\lambda = 0.5$ становятся близкими к 1.0, а веса $w_{10^{14}} \to 0.0$

In [28]:
```
data_set
```

Out[28]:

	x	y	$loss_1$	$\tilde{y}_1$	$loss_2$	$\tilde{y}_2$	w_1	w_2
0	1.0	180.212195	0.000000e+00	0.000000	0.000000e+00	0.000000e+00	0.500000	5.000000e-01
1	2.0	-9.943453	0.000000e+00	0.000000	0.000000e+00	0.000000e+00	0.500000	5.000000e-01
2	3.0	6.013143	3.297292e+04	-13.926986	3.261147e+04	4.633134e-11	0.500000	5.000000e-01
3	4.0	88.645627	4.122241e+04	-2.181112	4.046951e+04	2.083589e-10	0.499096	5.009036e-01
4	5.0	133.015924	5.354994e+04	21.986505	5.816275e+04	7.852943e-09	0.497214	5.027858e-01
...	...	...	...	...	...	...	...	...
95	96.0	271.723388	1.407031e+06	230.339538	3.042478e+06	3.525967e+01	1.000000	3.322536e-139
96	97.0	261.961141	1.407846e+06	233.418939	3.092608e+06	3.806220e+01	1.000000	2.623115e-146
97	98.0	257.321591	1.408357e+06	234.712653	3.139430e+06	4.093758e+01	1.000000	1.264701e-153
98	99.0	303.255048	1.413001e+06	235.109700	3.206685e+06	4.391967e+01	1.000000	3.837361e-161
99	100.0	103.292687	1.432138e+06	241.629933	3.209805e+06	4.743320e+01	1.000000	6.225314e-169

100 rows × 8 columns

Теперь рассмотрим побольше значений параметров λ

N = 15

$\lambda_i = 0.5, 1, 3, 6, 10, 50, 80, 170, 400, 1000, 10^4, 10^5, 10^6, 10^7, 10^8, \quad i = 1, \dots 15$

In [29]:
```
eta = 0.00001 #коэффициент метода экспертов
N = 15 #Количество экспертов
T #Количество итераций
```

Out[29]:
```
100
```

Составим список интересующих нас параметров

In [30]:
```
l_lst = [0.5, 1, 3, 6, 10, 50, 80, 170, 400, 1000, 10**4, 10**5, 10**6, 10**7, 10**8]
```

Создадим новый DataSet:

In [31]:
```
data_set = pd.DataFrame({'x': x, 'y': y})
```

Заполним пока что веса начальными значениями $\frac{1}{N}$, а впоследствии будем менять их

In [32]:
```
for l in l_lst:
    data_set[f'$w_{ {l} }$'] = 1/N
```

Промежуточный DataSet до применения метода экспертов

In [37]:
```
data_set.iloc[:,:10]
```

Out[37]:

	x	y	$w_{0.5}$	w_{1}	w_{3}	w_{6}	w_{10}	w_{50}	w_{80}	w_{170}
0	1.0	180.212195	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667
1	2.0	-9.943453	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667
2	3.0	6.013143	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667
3	4.0	88.645627	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667
4	5.0	133.015924	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667
...	...	...	...	...	...	...	...	...	...	...
95	96.0	271.723388	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667
96	97.0	261.961141	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667
97	98.0	257.321591	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667
98	99.0	303.255048	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667
99	100.0	103.292687	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667

100 rows × 10 columns

In [39]:
```
data_set.iloc[:,10:]
```

Out[39]:

	w_{400}	w_{1000}	w_{10000}	w_{100000}	$w_{1000000}$	$w_{10000000}$	$w_{100000000}$
0	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667
1	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667
2	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667
3	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667
4	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667
...	...	...	...	...	...	...	...
95	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667
96	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667
97	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667
98	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667
99	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667	0.066667

100 rows × 7 columns

Теперь реализуем метод экспертов

In [40]:
```
w_lst = [data_set[f'$w_{ {l} }$'] for l in l_lst]
loss_lst = np.zeros((15,T))
for l in l_lst:
    coeffs_lst, data_set[f'${{loss}}_{ {l} }$'], data_set[f'$\widetilde y_{ {l} }$'] = VAW(data_set, l)

loss_lst = [data_set[f'${{loss}}_{ {l} }$'] for l in l_lst]

for t in range(2, T-1):
    sum_of_w = sum([w_lst[i][t]*np.exp(-eta*loss_lst[i][t]) for i in range(N)])
    for i in range(N):
        w_lst[i][t+1] = w_lst[i][t]*np.exp(-eta*loss_lst[i][t])/sum_of_w
```

Теперь отобразим последние 60 значений, первые 40 особо смысла нету смотреть из-за дефицита информации

In [41]:
```
data_set.iloc[:, 2:17][40:]
```

Out[41]:

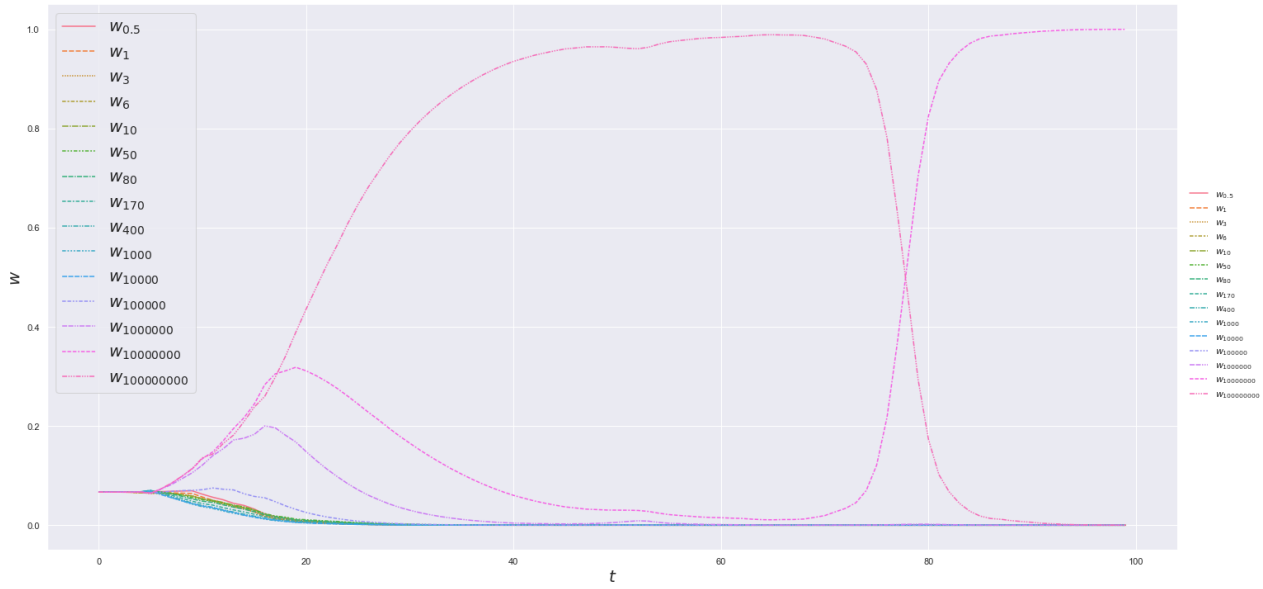
	$w_{0.5}$	w_1	w_3	w_6	w_{10}	w_{50}	w_{80}	w_{170}	w_{400}	w_{1000}	w_{10000}	w_{100000}	v
40	2.844054e-07	3.589431e-07	1.552275e-06	5.867629e-06	1.536642e-05	6.861462e-05	6.306372e-05	4.407115e-05	2.924485e-05	2.229396e-05	1.790537e-05	1.441267e-04	4.20
41	1.748281e-07	2.378164e-07	1.190865e-06	4.907881e-06	1.357814e-05	6.747803e-05	6.273416e-05	4.388262e-05	2.855020e-05	2.115944e-05	1.631052e-05	1.318242e-04	3.78
42	6.967466e-08	1.040784e-07	6.267891e-07	2.880767e-06	8.540306e-06	4.930049e-05	4.714560e-05	3.411029e-05	2.263764e-05	1.685542e-05	1.287066e-05	1.067927e-04	3.23
43	3.528270e-08	5.776403e-08	4.162687e-07	2.125616e-06	6.734583e-06	4.473784e-05	4.381685e-05	3.246963e-05	2.165731e-05	1.595195e-05	1.184832e-05	9.955906e-05	2.99
44	1.440443e-08	2.612250e-08	2.302245e-07	1.323316e-06	4.517551e-06	3.536339e-05	3.579723e-05	2.766222e-05	1.895743e-05	1.407020e-05	1.034437e-05	8.893774e-05	2.73
45	5.273921e-09	1.066951e-08	1.166819e-07	7.615247e-07	2.816289e-06	2.637624e-05	2.775336e-05	2.261625e-05	1.614308e-05	1.220782e-05	8.972829e-06	7.933689e-05	2.52
46	3.164817e-09	7.095514e-09	9.485186e-08	6.958747e-07	2.769321e-06	3.039294e-05	3.295294e-05	2.777784e-05	2.006655e-05	1.500201e-05	1.063221e-05	9.449977e-05	2.78
47	1.283930e-09	3.229863e-09	5.420008e-08	4.545049e-07	1.967229e-06	2.609207e-05	2.950481e-05	2.638386e-05	1.996164e-05	1.523887e-05	1.077843e-05	9.810543e-05	2.87
48	8.181170e-10	2.299456e-09	4.796150e-08	4.565866e-07	2.139526e-06	3.380966e-05	3.963469e-05	3.708811e-05	2.876465e-05	2.188847e-05	1.497981e-05	1.370889e-04	3.63
49	3.920158e-10	1.242106e-09	3.283541e-08	3.593023e-07	1.837334e-06	3.538514e-05	4.338367e-05	4.328052e-05	3.532269e-05	2.748998e-05	1.872302e-05	1.746500e-04	4.41
50	1.981956e-10	7.098405e-10	2.391819e-08	3.018840e-07	1.688240e-06	3.985143e-05	5.120380e-05	5.467812e-05	4.718484e-05	3.768848e-05	2.559207e-05	2.432869e-04	5.78
51	8.412139e-11	3.424572e-10	1.489125e-08	2.185330e-07	1.343433e-06	3.943418e-05	5.339378e-05	6.176139e-05	5.733741e-05	4.784143e-05	3.304457e-05	3.230240e-04	7.48
52	2.586705e-11	1.204874e-10	6.862588e-09	1.182545e-07	8.042631e-07	2.988977e-05	4.294650e-05	5.463943e-05	5.584876e-05	4.991828e-05	3.616869e-05	3.687821e-04	8.84
53	4.247949e-12	2.284073e-11	1.738983e-09	3.566950e-08	2.708006e-07	1.306869e-05	2.012327e-05	2.878407e-05	3.352717e-05	3.338871e-05	2.671503e-05	2.911507e-04	8.12
54	3.748339e-13	2.342562e-12	2.422379e-10	5.978924e-09	5.103311e-08	3.262873e-06	5.426152e-06	8.879282e-06	1.211947e-05	1.389757e-05	1.284304e-05	1.530578e-04	5.57
55	3.309772e-14	2.405281e-13	3.381638e-11	1.005043e-09	9.649011e-09	8.183055e-07	1.470362e-06	2.755261e-06	4.413471e-06	5.837341e-06	6.243592e-06	8.145765e-05	3.88
56	3.614643e-15	3.051667e-14	5.820016e-12	2.079538e-10	2.243215e-09	2.515911e-07	4.878941e-07	1.044188e-06	1.954214e-06	2.964136e-06	3.637327e-06	5.167765e-05	3.12
57	3.641755e-16	3.576323e-15	9.279889e-13	3.994469e-11	4.847912e-10	7.216850e-08	1.512451e-07	3.708124e-07	8.148163e-07	1.425867e-06	2.024416e-06	3.144423e-05	2.45
58	3.353555e-17	3.835206e-16	1.357709e-13	7.053821e-12	9.644056e-11	1.912022e-08	4.335810e-08	1.221187e-07	3.165407e-07	6.428124e-07	1.065051e-06	1.815820e-05	1.86
59	3.269664e-18	4.355299e-17	2.104348e-14	1.319924e-12	2.033269e-11	5.370878e-09	1.318031e-08	4.265809e-08	1.304870e-07	3.076208e-07	5.948965e-07	1.113164e-05	1.50
60	3.359004e-19	5.212691e-18	3.439440e-15	2.605542e-13	4.523370e-12	1.592946e-09	4.231339e-09	1.574413e-08	5.687400e-08	1.557733e-07	3.518886e-07	7.227927e-06	1.28
61	2.779321e-20	5.033270e-19	4.553659e-16	4.178338e-14	8.190808e-13	3.865314e-10	1.113418e-09	4.782907e-09	2.055408e-08	6.605396e-08	1.771280e-07	4.021850e-06	9.67
62	2.314552e-21	4.893060e-20	6.074619e-17	6.755208e-15	1.495836e-13	9.468673e-11	2.958767e-10	1.468525e-09	7.517763e-09	2.839812e-08	9.065082e-08	2.278105e-06	7.46
63	1.403101e-22	3.468019e-21	5.930864e-18	8.015411e-16	2.008696e-14	1.714125e-11	5.820866e-11	3.351805e-10	2.059097e-09	9.237991e-09	3.574692e-08	1.002728e-06	4.68
64	8.713379e-24	2.518228e-22	5.933643e-19	9.747291e-17	2.764781e-15	3.181516e-12	1.174215e-11	7.846256e-11	5.786740e-10	3.085223e-09	1.448621e-08	4.538194e-07	3.02
65	6.394954e-25	2.160573e-23	7.010525e-20	1.399243e-17	4.490947e-16	6.963564e-13	2.792514e-12	2.164006e-11	1.913772e-10	1.210381e-09	6.871895e-09	2.399316e-07	2.25
66	5.469065e-26	2.159658e-24	9.645194e-21	2.338161e-18	8.489388e-17	1.772545e-13	7.721641e-13	6.935577e-12	7.347226e-11	5.503360e-10	3.765239e-09	1.462703e-07	1.91
67	3.862996e-27	1.784793e-25	1.100004e-21	3.245041e-19	1.334609e-17	3.764984e-14	1.783642e-13	1.861620e-12	2.373314e-11	2.120023e-10	1.771331e-09	7.697507e-08	1.44
68	3.182933e-28	1.720344e-26	1.462609e-22	5.249089e-20	2.444878e-18	9.313461e-15	4.797390e-14	5.815742e-13	8.915005e-12	9.483996e-11	9.647938e-10	4.683023e-08	1.24
69	3.645685e-29	2.303595e-27	2.697077e-23	1.176042e-20	6.197982e-19	3.181017e-15	1.780275e-14	2.502455e-13	4.597705e-12	5.795388e-11	7.099817e-10	3.831396e-08	1.40
70	3.649841e-30	2.698862e-28	4.362802e-24	2.315862e-21	1.382843e-19	9.593592e-16	5.839513e-15	9.539760e-14	2.109712e-12	3.171726e-11	4.742453e-10	2.858660e-08	1.48
71	4.450960e-31	3.850779e-29	8.589934e-25	5.548379e-22	3.752547e-20	3.516290e-16	2.327254e-15	4.416022e-14	1.174166e-12	2.101299e-11	3.816661e-10	2.564686e-08	1.85
72	4.036543e-32	4.091808e-30	1.264214e-25	9.964426e-23	7.648231e-21	9.726769e-17	7.010329e-16	1.550249e-14	4.986952e-13	1.073093e-11	2.420208e-10	1.827154e-08	1.90

	$w_{0.5}$	w_1	w_3	w_6	w_{10}	w_{50}	w_{80}	w_{170}	w_{400}	w_{1000}	w_{10000}	w_{100000}	w
73	3.863441e-33	4.589377e-31	1.964649e-26	1.890142e-23	1.646782e-21	2.843820e-17	2.232275e-16	5.754825e-15	2.241148e-13	5.804057e-12	1.628533e-10	1.382211e-08	2.07
74	4.826306e-34	6.716273e-32	3.980314e-27	4.671070e-24	4.617296e-22	1.081420e-17	9.241700e-17	2.775134e-15	1.306168e-13	4.059564e-12	1.407056e-10	1.338610e-08	2.84
75	7.446796e-35	1.213715e-32	9.951672e-28	1.423875e-24	1.596325e-22	5.066241e-18	4.712350e-17	1.647215e-15	9.358693e-14	3.483273e-12	1.482904e-10	1.577695e-08	4.73
76	1.214928e-35	2.319517e-33	2.632299e-28	4.593188e-25	5.841546e-23	2.513338e-18	2.544799e-17	1.035780e-15	7.107186e-14	3.170112e-12	1.659529e-10	1.974874e-08	8.34
77	1.786017e-36	3.996138e-34	6.284678e-29	1.338721e-25	1.932702e-23	1.129187e-18	1.245166e-17	5.907444e-16	4.905208e-14	2.630689e-12	1.706565e-10	2.277325e-08	1.36
78	1.918637e-37	5.035577e-35	1.100125e-29	2.866165e-26	4.703477e-24	3.743920e-19	4.500550e-18	2.494138e-16	2.516439e-14	1.634121e-12	1.337272e-10	2.013862e-08	1.74
79	1.836807e-38	5.655856e-36	1.717295e-30	5.474154e-27	1.021391e-24	1.108389e-19	1.452768e-18	9.408708e-17	1.154473e-14	9.091047e-13	9.419728e-11	1.603038e-08	2.01
80	1.398840e-39	5.055439e-37	2.135679e-31	8.336855e-28	1.769732e-25	2.622386e-20	3.749564e-19	2.841075e-17	4.249011e-15	4.073318e-13	5.400452e-11	1.042917e-08	1.92
81	8.753752e-41	3.713982e-38	2.184312e-32	1.044702e-28	2.523982e-26	5.112142e-21	7.976675e-20	7.077183e-18	1.292310e-15	1.512882e-13	2.588933e-11	5.695908e-09	1.56
82	4.510735e-42	2.246599e-39	1.839247e-33	1.077672e-29	2.963089e-27	8.203994e-22	1.397160e-20	1.452202e-18	3.241157e-16	4.642884e-14	1.031604e-11	2.595206e-09	1.08
83	2.207365e-43	1.290330e-40	1.469695e-34	1.054531e-30	3.298772e-28	1.247755e-22	2.319034e-21	2.823352e-19	7.700373e-17	1.349267e-14	3.888912e-12	1.119164e-09	7.18
84	1.046663e-44	7.179744e-42	1.137227e-35	9.988551e-32	3.553978e-29	1.835445e-23	3.722458e-22	5.307496e-20	1.768440e-17	3.788513e-15	1.414507e-12	4.657129e-10	4.60
85	4.714582e-46	3.793911e-43	8.349713e-37	8.971226e-33	3.628862e-30	2.556151e-24	5.655838e-23	9.440812e-21	3.840755e-18	1.004971e-15	4.846169e-13	1.825210e-10	2.80
86	1.960808e-47	1.849868e-44	5.646804e-38	7.411027e-34	3.404491e-31	3.263360e-25	7.874000e-24	1.537463e-21	7.626176e-19	2.431123e-16	1.502341e-13	6.465301e-11	1.55
87	7.561328e-49	8.353893e-46	3.526260e-39	5.639005e-35	2.936636e-32	3.815119e-26	1.002963e-24	2.287216e-22	1.379392e-19	5.330043e-17	4.152307e-14	2.034134e-11	7.81
88	2.963527e-50	3.835985e-47	2.241760e-40	4.372344e-36	2.583085e-33	4.555260e-27	1.305210e-25	3.478414e-23	2.553360e-20	1.198229e-17	1.183620e-14	6.609906e-12	4.03
89	1.139370e-51	1.727132e-48	1.395756e-41	3.316958e-37	2.221392e-34	5.308760e-28	1.657267e-26	5.157970e-24	4.602856e-21	2.617217e-18	3.252050e-15	2.065699e-12	2.01
90	4.358532e-53	7.736121e-50	8.641284e-43	2.501160e-38	1.898275e-35	6.143545e-29	2.089217e-27	7.591521e-25	8.231175e-22	5.665065e-19	8.819570e-16	6.364208e-13	9.93
91	1.706487e-54	3.548445e-51	5.486573e-44	1.936535e-39	1.667087e-36	7.321088e-30	2.713263e-28	1.151968e-25	1.519764e-22	1.269384e-19	2.497749e-16	2.052549e-13	5.10
92	6.359895e-56	1.548254e-52	3.307391e-45	1.421256e-40	1.386141e-37	8.237638e-31	3.325168e-29	1.647743e-26	2.639749e-23	2.665693e-20	6.539813e-17	6.096107e-14	2.44
93	2.427910e-57	6.923847e-54	2.046975e-46	1.072466e-41	1.186248e-38	9.562813e-32	4.206450e-30	2.435189e-27	4.745600e-24	5.812649e-21	1.798172e-17	1.907862e-14	1.22
94	9.220994e-59	3.080182e-55	1.259945e-47	8.046568e-43	1.009222e-39	1.103166e-32	5.287516e-31	3.575511e-28	8.473210e-25	1.258062e-21	4.895419e-18	5.907159e-15	6.07
95	3.383824e-60	1.323428e-56	7.480801e-49	5.817484e-44	8.267119e-41	1.223078e-33	6.385123e-32	5.039593e-29	1.450275e-25	2.603143e-22	1.260901e-18	1.724588e-15	2.86
96	1.232020e-61	5.640564e-58	4.403616e-50	4.167945e-45	6.708597e-42	1.342201e-34	7.630571e-33	7.027066e-30	2.454133e-26	5.318537e-23	3.190150e-19	4.936352e-16	1.32
97	4.464778e-63	2.392551e-59	2.578853e-51	2.969776e-46	5.412746e-43	1.463641e-35	9.060274e-34	9.732907e-31	4.123227e-27	1.077899e-23	7.974435e-20	1.393838e-16	6.07
98	1.612610e-64	1.011358e-60	1.504618e-52	2.107657e-47	4.349061e-44	1.588710e-36	1.070715e-34	1.341452e-31	6.891005e-28	2.171445e-24	1.974797e-20	3.893681e-17	2.75
99	5.759449e-66	4.226217e-62	8.671333e-54	1.476511e-48	3.447530e-45	1.699225e-37	1.246472e-35	1.820372e-32	1.132830e-28	4.294620e-25	4.761988e-21	1.055734e-17	1.21

Итог: Связи с высокой степенью у полинома - нам больше будут подходить достаточно большие значения параметра λ, на конкретном примере лучшее результат дает $w_{10000000} = 0.330875$ на самоей последней итерации

11. Визуализация работы предсказания с советами экспертов

In [42]:
```
sns.relplot(data = data_set.iloc[:, 2:17], kind = 'line',height=10, aspect=2);
plt.xlabel('$t$', fontsize=20)
plt.ylabel('$w$', fontsize=20)
plt.legend(fontsize=20);
```



На графике также видно, что начиная с некоторой итерации, а именно: с $t = 70$ вес начал стремительно увеличиваться относительно других

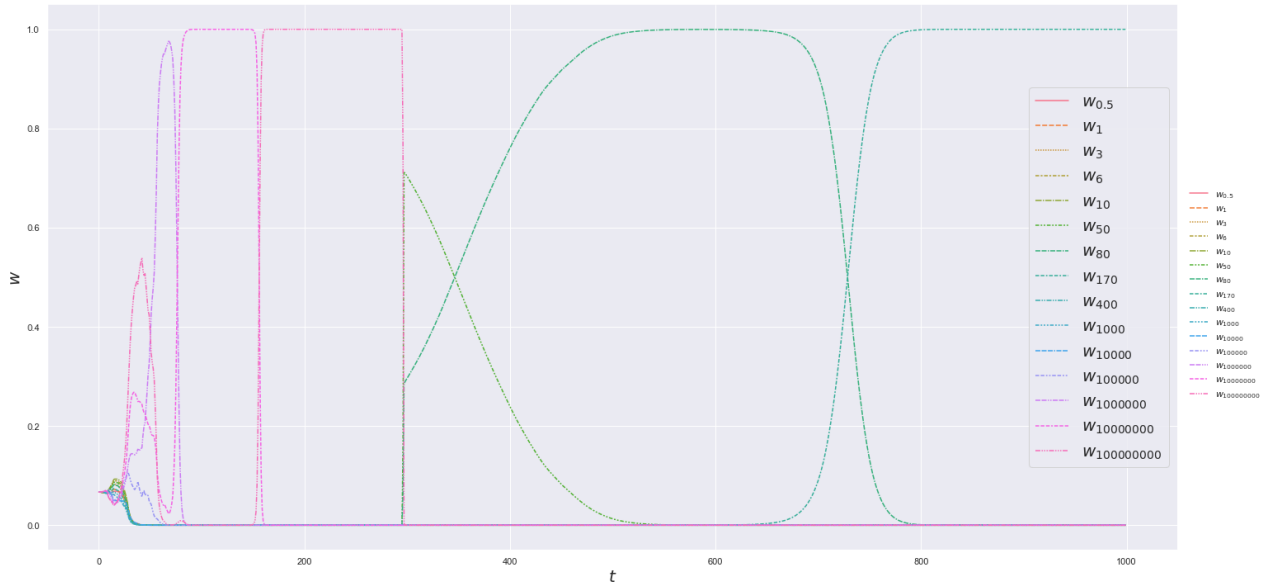
12. Влияние количества итераций на предсказание с советами экспертов

Если увеличить количество итераций, то пронаблюдаем другую картину:

In [43]:
```
eta = 0.00001 #коэффициент метода экспертов
N = 15        #Количество экспертов
T = 1000      #Количество итераций

x = np.arange(1, T+1, dtype = 'float')
y = 3*x + 100*np.random.normal(0,1, T)

data_set = pd.DataFrame({'x': x, 'y': y})
for l in l_lst:
    data_set[f'$w_{ {l} }$'] = 1/N
```

In [44]:
```
w_lst = [data_set[f'$w_{ {l} }$'] for l in l_lst]
loss_lst = np.zeros((15,T))
for l in l_lst:
    coeffs_lst, data_set[f'${{loss}}_{ {l} }$'], data_set[f'$\widetilde y_{ {l} }$'] = VAW(data_set, l)

loss_lst = [data_set[f'${{loss}}_{ {l} }$'] for l in l_lst]

for t in range(2, T-1):
    sum_of_w = sum([w_lst[i][t]*np.exp(-eta*loss_lst[i][t]) for i in range(N)])
    for i in range(N):
        w_lst[i][t+1] = w_lst[i][t]*np.exp(-eta*loss_lst[i][t])/sum_of_w
```

In [45]:
```
sns.relplot(data = data_set.iloc[:, 2:17], kind = 'line',height=10, aspect=2);
plt.xlabel('$t$', fontsize=20)
plt.ylabel('$w$', fontsize=20)
plt.legend(fontsize=20);
```



Заметим, что при увеличении итераций в 10 раз: $T = 1000$. Видим, что очень большая регуляризация после $t = 300$ начинает сильно проигрывать и более малая регуляризация дает более хороший результат

13. Применение задачи линейной регрессии на реальных данных

Возьмем данные по короновирусу https://github.com/CSSEGISandData/COVID-19/tree/master/csse_covid_19_data/

Использован файл: time_series_covid19_confirmed_global.csv

In [61]:
```
df = pd.read_csv('time_series_covid19_confirmed_global.csv')
pd.read_csv('time_series_covid19_confirmed_global.csv').iloc[:,1:]
```

Out[61]:

	Country/Region	Lat	Long	1/22/20	1/23/20	1/24/20	1/25/20	1/26/20	1/27/20	1/28/20	...	2/28/23	3/1/23	3/2/23	3/3/23
0	Afghanistan	33.939110	67.709953	0	0	0	0	0	0	0	...	209322	209340	209358	209362
1	Albania	41.153300	20.168300	0	0	0	0	0	0	0	...	334391	334408	334408	334427
2	Algeria	28.033900	1.659600	0	0	0	0	0	0	0	...	271441	271448	271463	271469
3	Andorra	42.506300	1.521800	0	0	0	0	0	0	0	...	47866	47875	47875	47875
4	Angola	-11.202700	17.873900	0	0	0	0	0	0	0	...	105255	105277	105277	105277
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
284	West Bank and Gaza	31.952200	35.233200	0	0	0	0	0	0	0	...	703228	703228	703228	703228
285	Winter Olympics 2022	39.904200	116.407400	0	0	0	0	0	0	0	...	535	535	535	535
286	Yemen	15.552727	48.516388	0	0	0	0	0	0	0	...	11945	11945	11945	11945
287	Zambia	-13.133897	27.849332	0	0	0	0	0	0	0	...	343012	343012	343079	343079
288	Zimbabwe	-19.015438	29.154857	0	0	0	0	0	0	0	...	263921	264127	264127	264127

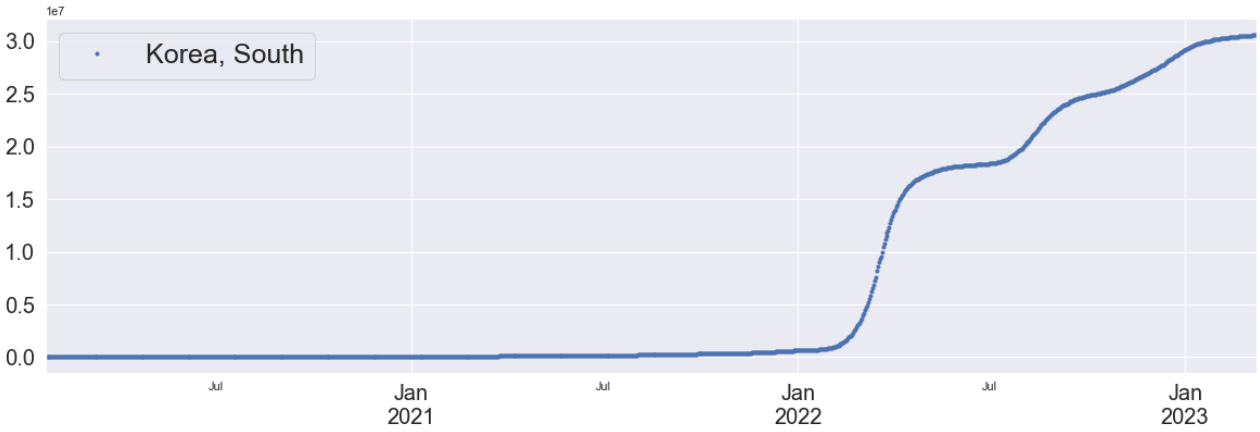
289 rows × 1146 columns

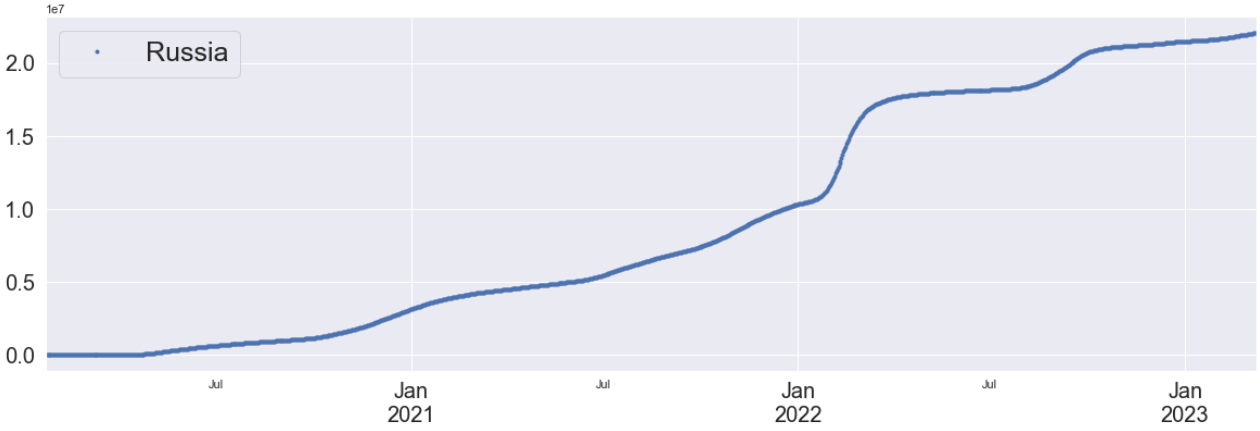
Локации взяты: Австрия, Южная Корея 163 "Korea, South", Россия 220

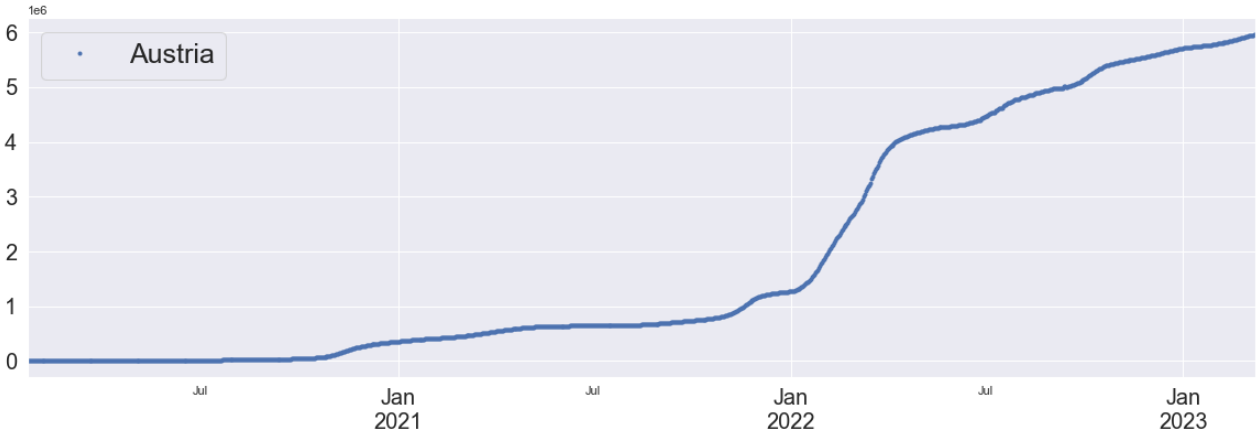
In [47]:
```
countries = ['Korea, South', 'Russia', 'Austria']
```

In [48]:
```
# Найдем индексы стран и вырежем из таблицы нужные данные
indices = [ ]
for ind in countries:
    indices.append(df[df['Country/Region'] == ind].index[0])

ts_lst = [df.iloc[i][4:] for i in indices] #ts = time series
for i in ts_lst:
    i += 1
    i.index = pd.to_datetime(i.index)
```

In [49]:
```
for j, i in enumerate(ts_lst):
    plt.figure(figsize=(20, 6));
    plt.tick_params(axis='both', which='major', labelsize = 20);
    i.plot(style = '.', grid = True, label = countries[j]);
    plt.legend(loc = 2, prop={'size': 25});
```







Для рассмотрения возьмем Южную Корею

Использовать будем полином 4ой степени

In [50]:
```
l_lst = [0.5, 1, 3, 6, 10, 50, 80, 170, 400, 1000, 10**4, 10**5, 10**6, 10**7, 10**8]
```

In [51]:
```
ts_k = ts_lst[0][:100]
```

In [52]:
```
T = len(ts_k) #Количество итераций
N = 15 #Количество экспертов
eta = 10**(-7) #коэффициент метода экспертов
```

In [53]:
```
x = np.arange(1, T+1)
y = np.array(ts_k)
```

После обработки данных получили следующую таблицу

In [54]:
```
ts_k = pd.DataFrame({'x': x, 'y': y})
ts_k
```

Out[54]:

	x	y
0	1	2
1	2	2
2	3	3
3	4	3
4	5	4
...	...	...
95	96	10739
96	97	10753
97	98	10762
98	99	10766
99	100	10775

100 rows × 2 columns

Сразу применим обучение с экспертами:

In [55]:
```
for l in l_lst:
    ts_k[f'$w_{ {l} }$'] = 1/N
```

In [56]:
```
w_lst = [ts_k[f'$w_{ {l} }$'] for l in l_lst]
loss_lst = np.zeros((N,T))
for l in l_lst:
    coeffs_lst, ts_k[f'${{loss}}_{ {l} }$'], ts_k[f'$\widetilde y_{ {l} }$'] = VAW(ts_k, l)

loss_lst = [ts_k[f'${{loss}}_{ {l} }$'] for l in l_lst]
```

In [57]:
```
for t in range(2, T-1):
    sum_of_w = np.sum([w_lst[i][t]*np.exp(-eta*loss_lst[i][t]) for i in range(N)])
    for i in range(N):
        w_lst[i][t+1] = w_lst[i][t]*np.exp(-eta*loss_lst[i][t])/sum_of_w
```

Составим таблицу весов

In [58]:
```
ts_k.iloc[:, 2:13]
```

Out[58]:

	$w_{0.5}$	w_1	w_3	w_6	w_{10}	w_{50}	w_{80}	w_{170}	w_{400}	w_{1000}	w_{10000}
0	0.066667	6.666667e-02	6.666667e-02	6.666667e-02	6.666667e-02	6.666667e-02	6.666667e-02	6.666667e-02	0.066667	0.066667	0.066667
1	0.066667	6.666667e-02	6.666667e-02	6.666667e-02	6.666667e-02	6.666667e-02	6.666667e-02	6.666667e-02	0.066667	0.066667	0.066667
2	0.066667	6.666667e-02	6.666667e-02	6.666667e-02	6.666667e-02	6.666667e-02	6.666667e-02	6.666667e-02	0.066667	0.066667	0.066667
3	0.066667	6.666666e-02	6.666667e-02	6.666667e-02	6.666667e-02	6.666668e-02	6.666668e-02	6.666668e-02	0.066667	0.066667	0.066667
4	0.066667	6.666664e-02	6.666666e-02	6.666667e-02	6.666668e-02	6.666671e-02	6.666671e-02	6.666671e-02	0.066667	0.066667	0.066667
...	...	...	...	...	...	...	...	...	...	...	...
95	1.000000	1.793661e-08	4.184961e-35	2.464449e-66	7.180108e-98	4.575864e-217	6.650130e-246	1.661043e-282	0.000000	0.000000	0.000000
96	1.000000	1.265256e-08	8.599224e-36	1.112653e-67	6.503015e-100	4.922833e-222	1.218534e-251	3.116196e-289	0.000000	0.000000	0.000000
97	1.000000	8.986880e-09	1.823442e-36	5.340420e-69	6.453622e-102	6.467501e-227	2.789887e-257	7.488312e-296	0.000000	0.000000	0.000000
98	1.000000	6.427010e-09	3.989439e-37	2.724487e-70	7.017605e-104	1.040233e-231	8.010068e-263	2.316630e-302	0.000000	0.000000	0.000000
99	1.000000	4.627480e-09	9.003046e-38	1.476726e-71	8.357857e-106	2.051411e-236	2.890946e-268	9.261078e-309	0.000000	0.000000	0.000000

100 rows × 11 columns

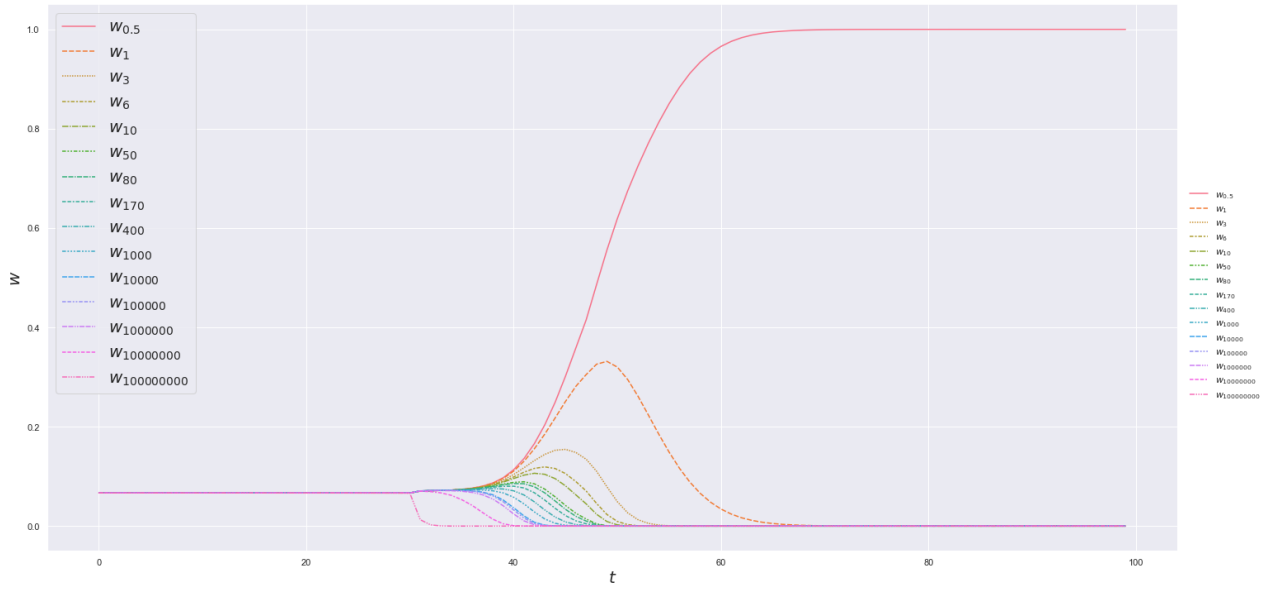
In [59]:
```
ts_k.iloc[:, 13:17]
```

Out[59]:

	w_{100000}	$w_{1000000}$	$w_{10000000}$	$w_{100000000}$	
0	0.066667	0.066667	0.066667	0.066667	
1	0.066667	0.066667	0.066667	0.066667	
2	0.066667	0.066667	0.066667	0.066667	
3	0.066667	0.066667	0.066667	0.066667	
4	0.066667	0.066667	0.066667	0.066667	
...	...	...	...	...	
95	0.000000	0.000000	0.000000	0.000000	
96	0.000000	0.000000	0.000000	0.000000	
97	0.000000	0.000000	0.000000	0.000000	
98	0.000000	0.000000	0.000000	0.000000	
99	0.000000	0.000000	0.000000	0.000000	

100 rows × 4 columns

In [60]:
```
sns.relplot(data = ts_k.iloc[:, 2:17], kind = 'line',height=10, aspect=2);
plt.xlabel('$t$', fontsize=20)
plt.ylabel('$w$', fontsize=20)
plt.legend(fontsize=20);
```



Так как это данные короновируса, то из-за особенности сбора данных - шумов не очень много, поэтому нужно выбирать достаточно малый параметр регуляризации

Заключение

Заключение: В работе были изучены основы теории Online Convex Optimization, алгоритм VAW, подбор оптимальных параметров в онлайн режиме для алгоритма VAW с помощью советов экспертов. Применены алгоритмы на реальном датасете. Все было реализовано на языке Python.

Список литературы

[1] Cesa-Bianchi, Gabor Lugosi-Prediction, learning, and games-Cambridge University Press (2006)

[2] Joulani, A György, C Szepesvári 2020 A modular analysis of adaptive (non-)convex optimization Optimism, composite objectives, variance reduction, and variational bounds

[3] Orabona 2020 v5 A Modern Introduction to Online Learning