Задание 3

Генерация данных

$-a_2 = a_1 = a_0 = 1$ (в коде - массив a)

$\sigma^2 = 0.1$ (в коде - var)

In [90]:

```
import numpy as np
from scipy.stats import norm, uniform, randint
from scipy.special import expit
from scipy.optimize import linprog
import matplotlib.pyplot as plt
%matplotlib inline
```

In [153]:

```
def get_sample(M, var, origin):
    return origin + norm.rvs(size=M, scale=np.sqrt(var))


M = 200
t = np.array(range(M)) * 10 / M

a = np.array([-1, 1, 1])

fun = lambda t, a: (a * np.array([np.sin(t), t, 1])).sum()

var = 0.1

origin = fun(t, a)
sample = get_sample(M, var, origin)
```

Решение задач оптимизации

Просят решить три задачи оптимизации.

- Минимизировать сумму квадратов ошибок. Эту задачу решает метод наименьших квадратов:

Пусть $T = \begin{pmatrix} sin(t_0) & t_0 & 1 \\ sin(t_1) & t_1 & 1 \\ \vdots & \vdots & \vdots \\ sin(t_{m-1}) & t_{m-1} & 1 \end{pmatrix}, b = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{m-1} \end{pmatrix}, a = \begin{pmatrix} a_2 \\ a_1 \\ a_0 \end{pmatrix}$

Тогда задача минимизации записывается как $|T\vec{a} - \vec{b}| \to min$ и имеет решение $\vec{a} = (T^T T)^{-1} T^T \vec{b}$

- Минимизировать сумму модулей ошибок. Это решается с помощью следующей задачи ЛП:

Переменные: $\{v_2, u_2, v_1, u_1, v_0, u_0, \delta_0, \delta_1, \ldots, \delta_{m-1}\}$

$$\vec{a} = \begin{pmatrix} v_2 - u_2 \\ v_1 - u_1 \\ v_0 - u_0 \end{pmatrix}$$

$$T\vec{a} - \vec{\delta} \le \vec{b}$$

$$-T\vec{a} - \vec{\delta} \le -\vec{b}$$

$$\sum \delta_i \to min$$

- Минимизировть максимум ошибки. Это та же задача ЛП, только вместо $\delta_0, \delta_1, \delta_2$ считаем, что за ошибку отвечает только одна переменная δ и минимизируем её:

Переменные: $\{v_2, u_2, v_1, u_1, v_0, u_0, \delta\}$

$$\vec{a} = \begin{pmatrix} v_2 - u_2 \\ v_1 - u_1 \\ v_0 - u_0 \end{pmatrix}$$

$$\vec{\Delta} = \begin{pmatrix} \delta \\ \vdots \\ \delta \end{pmatrix}$$

$$T\vec{a} - \vec{\Delta} \le \vec{b}$$

$$-T\vec{a} - \vec{\Delta} \le -\vec{b}$$

$$\delta \to min$$

PS: Утверждается, что scipy.optimize.linprog позволяет проводить симплекс-метод без условия неотрицательности переменных. Решается это так, как показано выше, а в коде будет напрямую искаться вектор $\vec{a}$

In [92]:

```
# Метод наименьших квадратов
def get_lst2_a(t, sample):
    T = np.matrix([np.sin(t), t, np.array([1] * M)]).T

    return np.linalg.lstsq(T, sample)[0] # Да, именно он реализован в этой функции
```

In [118]:

```
# Первая задача ЛП
def get_lst1_a(t, sample):
    E = np.matrix(np.identity(len(t)))
    A = np.concatenate((T, -E), axis=1)
    A = np.concatenate((A, np.concatenate((-T, -E), axis=1)), axis=0)

    bounds = [(None, None)] * 3 + [(0, None)] * len(t) # Задаем границы значений пе
    return linprog(c=([0] * 3 + [1] * len(t)), A_ub=A, b_ub=list(sample) + list(-sa
                   bounds=bounds, options={'disp': False,
                                           'bland': False,
                                           'tol': 1e-6,
                                           'maxiter': 1000})['x'][:3]
```

In [143]:

```
# Вторая задача ЛП
def get_lstinf_a(t, sample):
    E = np.matrix(np.ones(len(t))).T # Вместо разных - всегда одна ошибка
    A = np.concatenate((T, -E), axis=1)
    A = np.concatenate((A, np.concatenate((-T, -E), axis=1)), axis=0)

    bounds = [(None, None)] * 3 + [(0, None)] # Задаем границы значений переменных.
    return linprog(c=([0] * 3 + [1]), A_ub=A, b_ub=list(sample) + list(-sample),
                   bounds=bounds, options={'disp': False,
                                           'bland': False,
                                           'tol': 1e-3,
                                           'maxiter': 10000})['x'][:3]
```

In [162]:

```
a_2 = get_lst2_a(t, sample)
a_1 = get_lst1_a(t, sample)
a_inf = get_lstinf_a(t, sample)
print(a_2, a_1, a_inf, sep='\n')
```

```
[-1.19795885  1.29567591 -0.44169124]
[-1.0288916   1.00793451  0.95352662]
[ -1.16568772   6.01751838 -23.98680129]
```

Пункт а)

In [163]:

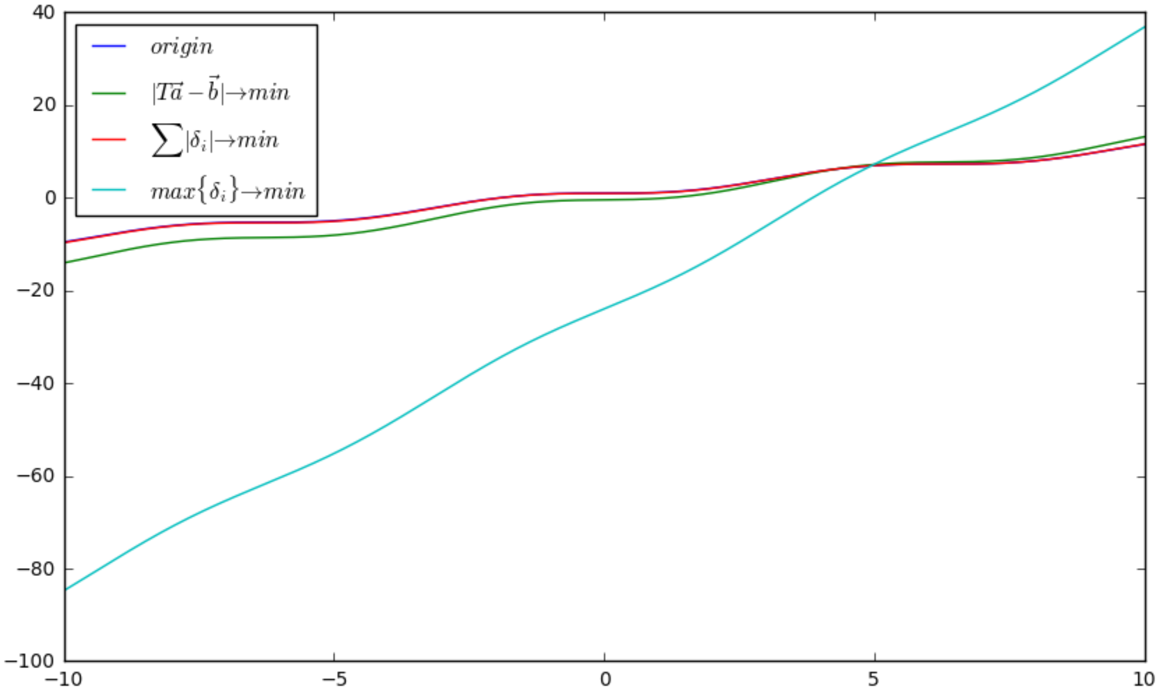
```
def graph_by_a(a_2, a_1, a_inf):
    t_all = np.linspace(-10, 10, 1000)

    plt.figure(figsize=(10, 6))

    plt.plot(t_all, fun(t_all, a), label=r'$origin$')
    plt.plot(t_all, fun(t_all, a_2), label=r'$|T \vec{a} - \vec{b}| \to min$')
    plt.plot(t_all, fun(t_all, a_1), label=r'$\sum{|\delta_i|} \to min$')
    plt.plot(t_all, fun(t_all, a_inf), label=r'$max \{\delta_i\} \to min$')

    plt.legend(loc='upper left')
    plt.show()


graph_by_a(a_2, a_1, a_inf)
```



Пункт b)
В пункте b) логично для $a^{(2)}$ посчитать модуль разности с истинным значением, для $a^{(1)}$ - сумму модулей разности, для $a^{(\infty)}$ - максимальную ошибку

In [166]:

```
def test_a_by_sample(sample):
    a_2 = get_lst2_a(t, sample)
    a_1 = get_lst1_a(t, sample)
    a_inf = get_lstinf_a(t, sample)

    print(a_2, a_1, a_inf, sep='\n')

    print(np.sqrt(((a - a_2) ** 2).sum()))
    print(np.sum(np.abs(a - a_1)))
    print(np.max(np.abs(a - a_inf)))
    return a_2, a_1, a_inf


def test_a_by_var(var):
    sample = get_sample(M, var, origin)
    return test_a_by_sample(sample)
```

In [171]:

```
a_2, a_1, a_inf = test_a_by_var(0.01)
```

```
[-0.99695773  1.00148822  0.99541893]
[-0.9912974   1.00290681  0.96750772]
[-0.98665303  0.99657609  1.01703069]
0.00569705177522
0.0441016922472
0.0170306942428
```

In [172]:

```
a_2, a_1, a_inf = test_a_by_var(1)
```

```
[-0.78357664  0.99228722  0.96472931]
[-0.70117859  0.98796754  1.01770205]
[-0.72808059  0.95469073  1.34390821]
0.21941417534
0.328555916211
0.343908207638
```

In [173]:

```
a_2, a_1, a_inf = test_a_by_var(10)
```

```
[-1.05660087  1.01561866  1.04362036]
[-1.11865486  1.00836018  1.14187697]
[-4.10984293  0.75470671  3.24453566]
0.0731459962745
0.268892010279
3.10984292973
```

Видно, что с увеличением дисперсии нужно больше итераций для сходимости симплекс-метода. Метод наименьших квадратов всегда прекрасен.

Пункт c)

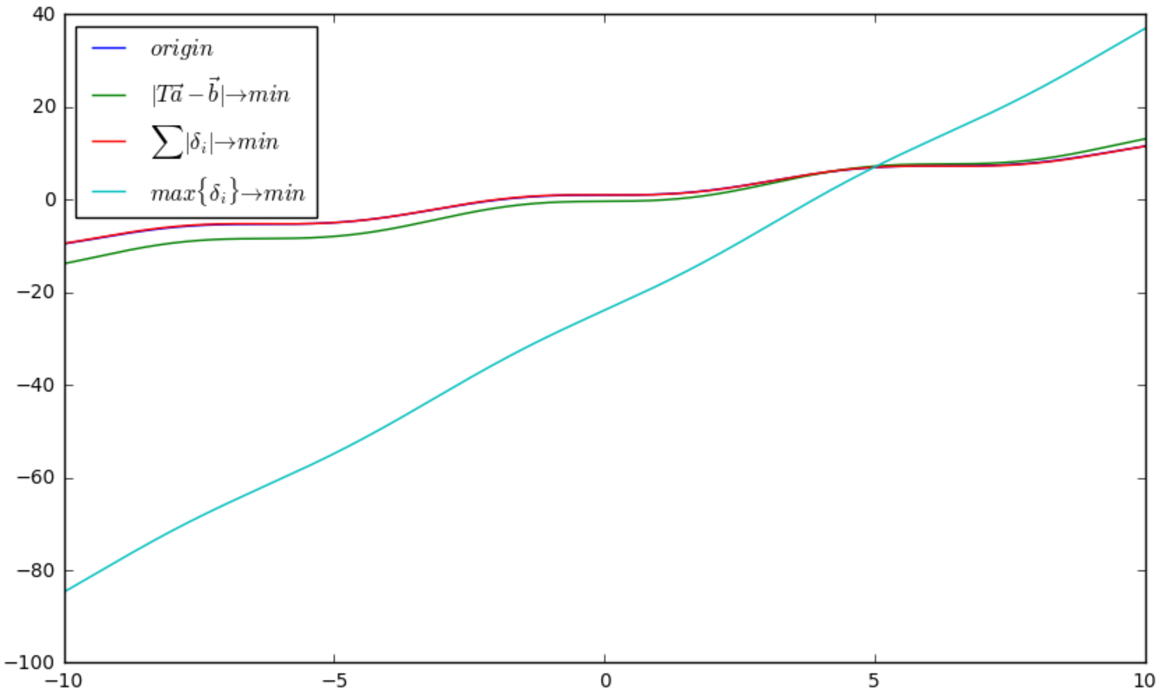
In [170]:

```
var = 0.1
sample = get_sample(M, var, origin)

sample[-1] += 50
sample[0] -= 50

a_2, a_1, a_inf = test_a_by_sample(sample)
graph_by_a(a_2, a_1, a_inf)
```

```
[-1.22904299  1.2803327  -0.35479032]
[-1.06417745  0.99208154  1.02314459]
[ -0.74543352   6.04377494 -23.91071122]
1.40232090193
0.0952405010014
24.9107112171
```



Видим, что наиболее устойчивой к выбросам оказался второй способ оптимизации (по мере $\sum|\delta_i| \to min$)