zumz181b

March 10, 2018

0.1 Uczenie maszynowe UMZ 2017/2018

1 1. Wprowadzenie. Regresja liniowa

1.1 1.4 Funkcja kosztu

1.1.1 Zadanie

Znajc x – ludno miasta (w dziesitkach tysicy mieszkaców), naley przewidzie y – dochód firmy transportowej (w dziesitkach tysicy dolarów).

(Dane pochodz z kursu Machine Learning", Andrew Ng, Coursera).

```
In [2]: # Nagowki, mona zignorowa

        import numpy as np
        import matplotlib
        import matplotlib.pyplot as pl
        import ipywidgets as widgets

        %matplotlib inline
        %config InlineBackend.figure_format = 'svg'

        from IPython.display import display, Math, Latex
```

1.1.2 Dane

```
In [3]: with open('data01.csv') as data:
            for line in data.readlines()[:10]:
                print(line)
```

```
6.1101,17.592

5.5277,9.1302

8.5186,13.662

7.0032,11.854

5.8598,6.8233
```

```
8.3829,11.886

7.4764,4.3483

8.5781,12

6.4862,6.5987

5.0546,3.8166
```

1.1.3 Wczytanie danych

```
In [4]: import csv

        reader = csv.reader(open('data01.csv'), delimiter=',')

        x = list()
        y = list()
        for xi, yi in reader:
            x.append(float(xi))
            y.append(float(yi))

        print('x = {}'.format(x[:10]))
        print('y = {}'.format(y[:10]))
```

```
x = [6.1101, 5.5277, 8.5186, 7.0032, 5.8598, 8.3829, 7.4764, 8.5781, 6.4862, 5.0546]
y = [17.592, 9.1302, 13.662, 11.854, 6.8233, 11.886, 4.3483, 12.0, 6.5987, 3.8166]
```

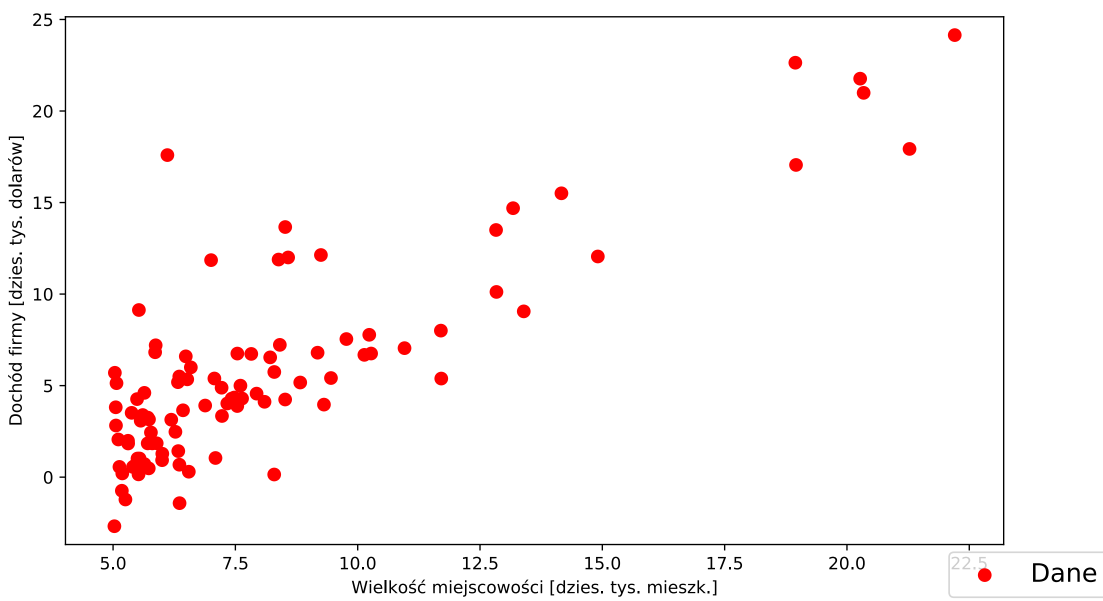
1.1.4 Hipoteza i parametry modelu

$$h_\theta(x) = \theta_0 + \theta_1 x$$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

```
In [5]: # Funkcje rysujce wykres kropkowy oraz prost regresyjn

        def regdots(x, y):
            fig = pl.figure(figsize=(16*.6, 9*.6))
            ax = fig.add_subplot(111)
            fig.subplots_adjust(left=0.1, right=0.9, bottom=0.1, top=0.9)
            ax.scatter(x, y, c='r', s=50, label='Dane')

            ax.set_xlabel(u'Wielko miejscowoci [dzies. tys. mieszk.]')
            ax.set_ylabel(u'Dochód firmy [dzies. tys. dolarów]')
```

```
    ax.margins(.05, .05)
    pl.ylim(min(y) - 1, max(y) + 1)
    pl.xlim(min(x) - 1, max(x) + 1)
    return fig

def regline(fig, fun, theta, x):
    ax = fig.axes[0]
    x0, x1 = min(x), max(x)
    X = [x0, x1]
    Y = [fun(theta, x) for x in X]
    ax.plot(X, Y, linewidth='2',
            label=(r'$y={theta0}{op}{theta1}x$'.format(
                theta0=theta[0],
                theta1=(theta[1] if theta[1] >= 0 else -theta[1]),
                op='+' if theta[1] >= 0 else '-')))

def legend(fig):
    ax = fig.axes[0]
    handles, labels = ax.get_legend_handles_labels()
    # try-except block is a fix for a bug in Poly3DCollection
    try:
        fig.legend(handles, labels, fontsize='15', loc='lower right')
    except AttributeError:
        pass

fig = regdots(x,y)
legend(fig)
```



```
In [6]: # Hipoteza: funkcja liniowa jednej zmiennej

        def h(theta, x):
            return theta[0] + theta[1] * x
```

```
In [7]: # Przygotowanie interaktywnego wykresu

        sliderTheta01 = widgets.FloatSlider(min=-10, max=10, step=0.1, value=0, description=r'$
        sliderTheta11 = widgets.FloatSlider(min=-5, max=5, step=0.1, value=0, description=r'$\

        def slide1(theta0, theta1):
            fig = regdots(x, y)
            regline(fig, h, [theta0, theta1], x)
            legend(fig)
```

```
In [8]: widgets.interact_manual(slide1, theta0=sliderTheta01, theta1=sliderTheta11)

interactive(children=(FloatSlider(value=0.0, description='$\\theta_0$', max=10.0, min=-10.0), F

Out[8]: <function __main__.slide1>
```

1.1.5 Funkcja kosztu

Bdziemy szuka takich parametrów $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$, które minimalizuj fukcj kosztu $J(\theta)$:

$$\hat{\theta} = \mathop{\arg\min}_{\theta} J(\theta)$$

$$\theta \in \mathbb{R}^2, \quad J \colon \mathbb{R}^2 \to \mathbb{R}$$

1.1.6 Bd redniokwadratowy

(metoda najmniejszych kwadratów)

$$J(\theta) \ = \frac{1}{2m} \sum_{i=1}^{m} \left(h_{\theta} \left(x^{(i)} \right) - y^{(i)} \right)^2$$

$$J(\theta_0, \theta_1) \ = \frac{1}{2m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)} \right)^2$$

```
In [9]: def J(h, theta, x, y):
            m = len(y)
            return 1.0 / (2 * m) * sum((h(theta, x[i]) - y[i])**2 for i in range(m))
```

```
In [11]: # Oblicz warto funkcji kosztu i poka na wykresie

         def regline(fig, fun, theta, x, y):
             ax = fig.axes[0]
```

```
    x0, x1 = min(x), max(x)
    X = [x0, x1]
    Y = [fun(theta, x) for x in X]
    cost = J(fun, theta, x, y)
    ax.plot(X, Y, linewidth='2',
            label=(r'$y={theta0}{op}{theta1}x, \; J(\theta)={cost:.3}$'.format(
                theta0=theta[0],
                theta1=(theta[1] if theta[1] >= 0 else -theta[1]),
                op='+' if theta[1] >= 0 else '-',
                cost=cost)))

sliderTheta02 = widgets.FloatSlider(min=-10, max=10, step=0.1, value=0, description=r
sliderTheta12 = widgets.FloatSlider(min=-5, max=5, step=0.1, value=0, description=r'$

def slide2(theta0, theta1):
    fig = regdots(x, y)
    regline(fig, h, [theta0, theta1], x, y)
    legend(fig)
```

```
In [12]: widgets.interact_manual(slide2, theta0=sliderTheta02, theta1=sliderTheta12)
```

```
interactive(children=(FloatSlider(value=0.0, description='$\\theta_0$', max=10.0, min=-10.0), F
```

```
Out[12]: <function __main__.slide2>
```

1.1.7 Funkcja kosztu jako funkcja zmiennej θ

```
In [14]: # Wykres funkcji kosztu dla ustalonego theta_1=1.0

         def costfun(fun, x, y):
             return lambda theta: J(fun, theta, x, y)

         def costplot(hypothesis, x, y, theta1=1.0):
             fig = pl.figure(figsize=(16*.6, 9*.6))
             ax = fig.add_subplot(111)
             fig.subplots_adjust(left=0.1, right=0.9, bottom=0.1, top=0.9)
             ax.set_xlabel(r'$\theta_0$')
             ax.set_ylabel(r'$J(\theta)$')
             j = costfun(hypothesis, x, y)
             fun = lambda theta0: j([theta0, theta1])
             X = np.arange(-10, 10, 0.1)
             Y = [fun(x) for x in X]
             ax.plot(X, Y, linewidth='2', label=(r'$J(\theta_0, {theta1})$'.format(theta1=theta
             return fig

         def slide3(theta1):
             fig = costplot(h, x, y, theta1)
             legend(fig)
```

```
sliderTheta13 = widgets.FloatSlider(min=-5, max=5, step=0.1, value=1.0, description=r
```

In [15]:
```
widgets.interact_manual(slide3, theta1=sliderTheta13)
```

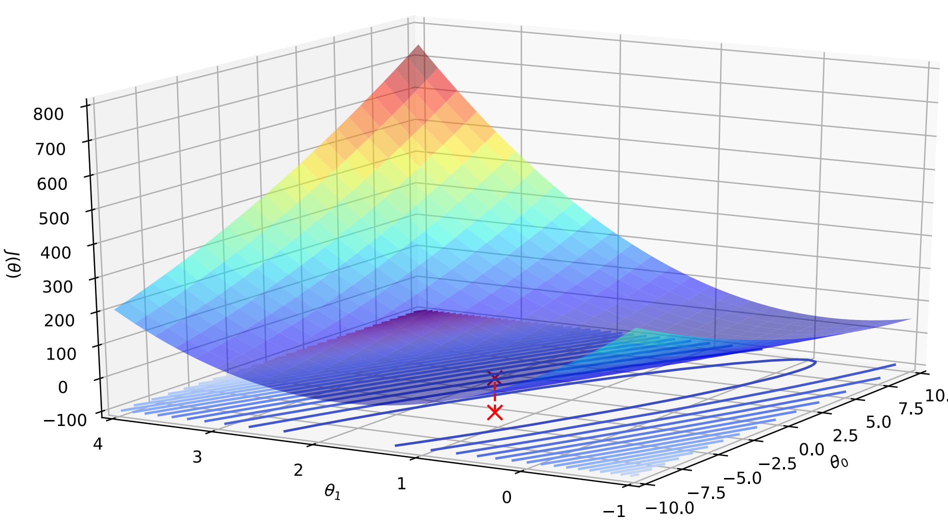
```
interactive(children=(FloatSlider(value=1.0, description='$\\theta_1$', max=5.0, min=-5.0), Bu
```

Out[15]:
```
<function __main__.slide3>
```

In [27]:
```
# Wykres funkcji kosztu wzgldem theta_0 i theta_1

from mpl_toolkits.mplot3d import Axes3D
import pylab

%matplotlib inline

def costplot3d(hypothesis, x, y, show_gradient=False):
    fig = pl.figure(figsize=(16*.6, 9*.6))
    ax = fig.add_subplot(111, projection='3d')
    fig.subplots_adjust(left=0.0, right=1.0, bottom=0.0, top=1.0)
    ax.set_xlabel(r'$\theta_0$')
    ax.set_ylabel(r'$\theta_1$')
    ax.set_zlabel(r'$J(\theta)$')

    j = lambda theta0, theta1: costfun(hypothesis, x, y)([theta0, theta1])
    X = np.arange(-10, 10.1, 0.1)
    Y = np.arange(-1, 4.1, 0.1)
    X, Y = np.meshgrid(X, Y)
    Z = np.matrix([[J(hypothesis, [theta0, theta1], x, y)
                    for theta0, theta1 in zip(xRow, yRow)]
                   for xRow, yRow in zip(X, Y)])

    ax.plot_surface(X, Y, Z, rstride=2, cstride=8, linewidth=0.5,
                    alpha=0.5, cmap='jet', zorder=0,
                    label=r"$J(\theta)$")
    ax.view_init(elev=20., azim=-150)

    ax.set_xlim3d(-10, 10);
    ax.set_ylim3d(-1, 4);
    ax.set_zlim3d(-100, 800);

    N = range(0, 800, 20)
    pl.contour(X, Y, Z, N, zdir='z', offset=-100, cmap='coolwarm', alpha=1)

    ax.plot([-3.89578088] * 2,
            [ 1.19303364] * 2,
            [-100, 4.47697137598],
            color='red', alpha=1, linewidth=1.3, zorder=100, linestyle='dashed',
```

```
                label=r'minimum: $J(-3.90, 1.19) = 4.48$')
        ax.scatter([-3.89578088] * 2,
                   [ 1.19303364] * 2,
                   [-100, 4.47697137598],
                   c='r', s=80, marker='x', alpha=1, linewidth=1.3, zorder=100,
                   label=r'minimum: $J(-3.90, 1.19) = 4.48$')

        if show_gradient:
            ax.plot([3.0, 1.1],
                    [3.0, 2.4],
                    [263.0, 125.0],
                    color='green', alpha=1, linewidth=1.3, zorder=100)
            ax.scatter([3.0],
                       [3.0],
                       [263.0],
                       c='g', s=30, marker='D', alpha=1, linewidth=1.3, zorder=100)

        ax.margins(0,0,0)
        fig.tight_layout()
```

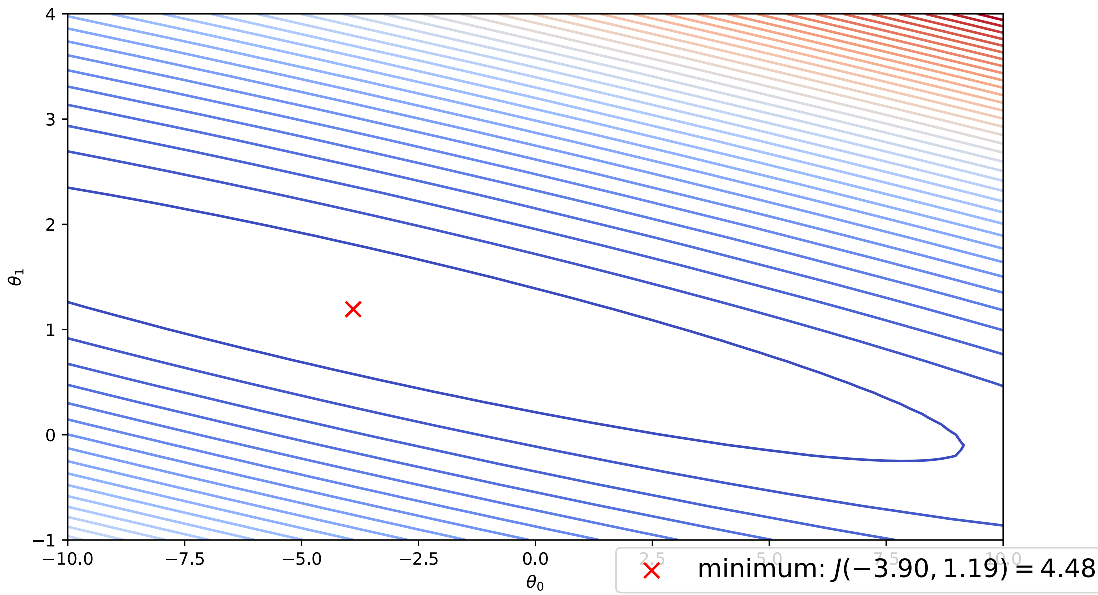
```
In [28]: costplot3d(h, x, y)
```




```
In [29]: def costplot2d(hypothesis, x, y, gradient_values=[], nohead=False):
             fig = pl.figure(figsize=(16*.6, 9*.6))
             ax = fig.add_subplot(111)
             fig.subplots_adjust(left=0.1, right=0.9, bottom=0.1, top=0.9)
             ax.set_xlabel(r'$\theta_0$')
```

```
    ax.set_ylabel(r'$\theta_1$')

    j = lambda theta0, theta1: costfun(hypothesis, x, y)([theta0, theta1])
    X = np.arange(-10, 10.1, 0.1)
    Y = np.arange(-1, 4.1, 0.1)
    X, Y = np.meshgrid(X, Y)
    Z = np.matrix([[J(hypothesis, [theta0, theta1], x, y)
                    for theta0, theta1 in zip(xRow, yRow)]
                   for xRow, yRow in zip(X, Y)])

    N = range(0, 800, 20)
    pl.contour(X, Y, Z, N, cmap='coolwarm', alpha=1)

    ax.scatter([-3.89578088], [1.19303364], c='r', s=80, marker='x',
               label=r'minimum: $J(-3.90, 1.19) = 4.48$')

    if len(gradient_values) > 0:
        prev_theta = gradient_values[0][1]
        ax.scatter([prev_theta[0]], [prev_theta[1]],
                   c='g', s=30, marker='D', zorder=100)
        for cost, theta in gradient_values[1:]:
            dtheta = [theta[0] - prev_theta[0], theta[1] - prev_theta[1]]
            ax.arrow(prev_theta[0], prev_theta[1], dtheta[0], dtheta[1],
                     color='green',
                     head_width=(0.0 if nohead else 0.1),
                     head_length=(0.0 if nohead else 0.2),
                     zorder=100)
            prev_theta = theta

    return fig
```

```
In [31]: fig = costplot2d(h, x, y)
         legend(fig)
```



1.1.8 Cechy funkcji kosztu

- $J(\theta)$ jest funkcj wypuk
- $J(\theta)$ posiada tylko jedno minimum lokalne

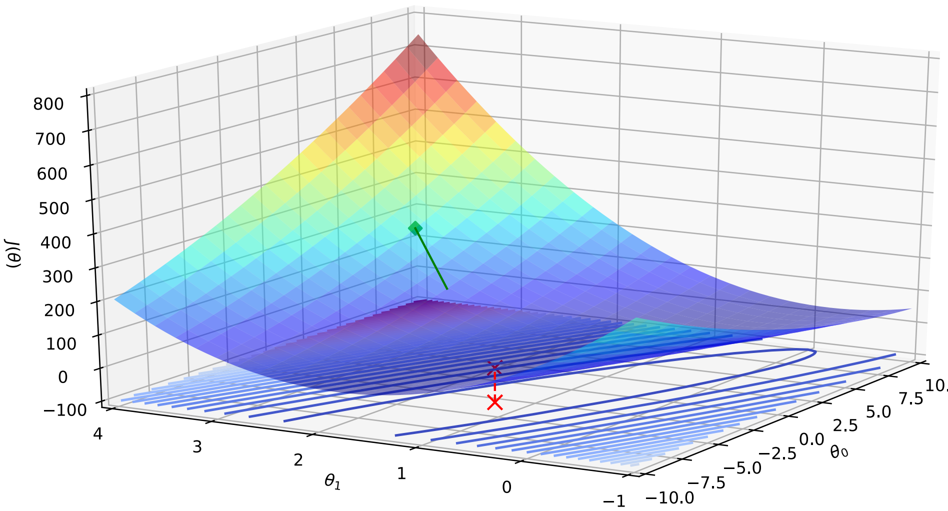
1.2 1.5. Metoda gradientu prostego

1.2.1 Metoda gradientu prostego

Metoda znajdowania minimów lokalnych.

Idea: * Zacznijmy od dowolnego θ. * Zmieniajmy powoli θ tak, aby zmniejsza $J(\theta)$, a w kocu znajdziemy minimum.

```
In [33]: costplot3d(h, x, y, show_gradient=True)
```




```
In [34]: # Przykadowe wartoci kolejnych przyblie (sztuczne)

         gv = [[_, [3.0, 3.0]], [_, [2.6, 2.4]], [_, [2.2, 2.0]], [_, [1.6, 1.6]], [_, [0.4, 1

         # Przygotowanie interaktywnego wykresu

         sliderSteps1 = widgets.IntSlider(min=0, max=3, step=1, value=0, description='kroki', w

         def slide4(steps):
             costplot2d(h, x, y, gradient_values=gv[:steps+1])
```

```
In [35]: widgets.interact(slide4, steps=sliderSteps1)
```

```
interactive(children=(IntSlider(value=0, description='kroki', max=3), Output()), _dom_classes=
```

```
Out[35]: <function __main__.slide4>
```

1.2.2 Metoda gradientu prostego

W kadym kroku bdziemy aktualizowa parametry θ_j:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \quad \text{dla kadego } j$$

Wspóczynnik α nazywamy *dugoci kroku* lub *wspóczynnikiem szybkoci uczenia* (*learning rate*).

$$
\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta \left(x^{(i)} \right) - y^{(i)} \right)^2 \\
&= 2 \cdot \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta \left(x^{(i)} \right) - y^{(i)} \right) \cdot \frac{\partial}{\partial \theta_j} \left(h_\theta \left(x^{(i)} \right) - y^{(i)} \right) \\
&= \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta \left(x^{(i)} \right) - y^{(i)} \right) \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^{n} \theta_i x_i^{(i)} - y^{(i)} \right) \\
&= \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta \left(x^{(i)} \right) - y^{(i)} \right) x_j^{(i)}
\end{aligned}
$$

Czyli dla regresji liniowej jednej zmiennej:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

w kadym kroku bdziemy aktualizowa:

$$
\begin{aligned}
\theta_0 &:= \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \\
\theta_1 &:= \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x^{(i)}
\end{aligned}
$$

Uwaga!

- W kadym kroku aktualizujemy *jednoczenie* θ_0 i θ_1
- Kolejne kroki wykonujemy a uzyskamy zbieno

1.2.3 Metoda gradientu prostego – implementacja

In [37]:
```
# Wywietlanie macierzy w LaTeX-u

def LatexMatrix(matrix):
    ltx = r'\left[\begin{array}'
    m, n = matrix.shape
    ltx += '{' + ("r" * n) + '}'
    for i in range(m):
        ltx += r" & ".join([('%.4f' % j.item()) for j in matrix[i]]) + r" \\ "
    ltx += r'\end{array}\right]'
    return ltx
```

In [38]:
```
def gradient_descent(h, cost_fun, theta, x, y, alpha, eps):
    current_cost = cost_fun(h, theta, x, y)
    log = [[current_cost, theta]]  # log przechowuje wartoci kosztu i parametrów
    m = len(y)
    while True:
        new_theta = [
            theta[0] - alpha/float(m) * sum(h(theta, x[i]) - y[i]
                                            for i in range(m)),
            theta[1] - alpha/float(m) * sum((h(theta, x[i]) - y[i]) * x[i]
```

```
                                          for i in range(m))]
            theta = new_theta  # jednoczesna aktualizacja - uywamy zmiennej tymaczasowej
            try:
                current_cost, prev_cost = cost_fun(h, theta, x, y), current_cost
            except OverflowError:
                break
            if abs(prev_cost - current_cost) <= eps:
                break
            log.append([current_cost, theta])
        return theta, log
```

In [67]:
```
best_theta, log = gradient_descent(h, J, [0.0, 0.0], x, y, alpha=0.02, eps=0.0001)

display(Math(r'\large\textrm{Wynik:}\quad \theta = ' +
             LatexMatrix(np.matrix(best_theta).reshape(2,1)) +
            (r' \quad J(\theta) = %.4f' % log[-1][0])
            + r' \quad \textrm{po %d iteracjach}' % len(log)))
```

$$\textrm{Wynik:}\quad \theta = \begin{bmatrix} -3.5074 \\ 1.1540 \end{bmatrix} \quad J(\theta) = 4.4908 \quad \textrm{po 644 iteracjach}$$

In [68]:
```
# Przygotowanie interaktywnego wykresu

sliderSteps2 = widgets.IntSlider(min=0, max=500, step=1, value=1, description='kroki'

def slide5(steps):
    costplot2d(h, x, y, gradient_values=log[:steps+1], nohead=True)
```

In [69]:
```
widgets.interact_manual(slide5, steps=sliderSteps2)
```

```
interactive(children=(IntSlider(value=1, description='kroki', max=500), Button(description='Ru
```

Out[69]:
```
<function __main__.slide5>
```

1.2.4 Wspóczynnik α (dugo kroku)

- Jeeli dugo kroku jest zbyt maa, algorytm moe dziaa zbyt wolno.
- Jeeli dugo kroku jest zbyt dua, algorytm moe nie by zbieny.

1.3 1.6. Regresja liniowa wielu zmiennych

1.3.1 Przykad – ceny mieszka

In [25]:
```
reader = csv.reader(open('data02.tsv'), delimiter='\t')
for i, row in enumerate(list(reader)[:10]):
    if i == 0:
        print(' '.join(['{}: {:8}'.format('x' + str(j) if j > 0 else 'y ', entry)
```

```
                             for j, entry in enumerate(row)]))
            else:
                print(' '.join(['{:12}'.format(entry) for entry in row]))
```

```
y : price     x1: isNew    x2: rooms    x3: floor    x4: location x5: sqrMetres
476118.0      False        3            1            Centrum       78
459531.0      False        3            2            Soacz        62
411557.0      False        3            0            Soacz        15
496416.0      False        4            0            Soacz        14
406032.0      False        3            0            Soacz        15
450026.0      False        3            1            Naramowice    80
571229.15     False        2            4            Wilda         39
325000.0      False        3            1            Grunwald      54
268229.0      False        2            1            Grunwald      90
```

$$x^{(2)} = (\text{"False"}, 3, 2, \text{"Soacz"}, 62), \quad x_3^{(2)} = 2$$

1.3.2 Hipoteza

W naszym przypadku:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4 + \theta_5 x_5$$

W ogólnoci:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$$

Jeeli zdefiniujemy $x_0 = 1$:

$$\begin{aligned} h_\theta(x) &= \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n \\ &= \sum_{i=0}^{n} \theta_i x_i \\ &= \theta^T x \\ &= x^T \theta \end{aligned}$$

1.3.3 Metoda gradientu prostego – notacja macierzowa

$$X = \begin{bmatrix} 1 & \left(\vec{x}^{(1)}\right)^T \\ 1 & \left(\vec{x}^{(2)}\right)^T \\ \vdots & \vdots \\ 1 & \left(\vec{x}^{(m)}\right)^T \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(1)} & \cdots & x_n^{(1)} \\ 1 & x_1^{(2)} & \cdots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & \cdots & x_n^{(m)} \end{bmatrix} \quad \vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

In [26]:
```
# Wczytywanie danych z pliku za pomoc numpy
# Wersje macierzowe funkcji rysowania wykresów punktowych oraz krzywej regresyjnej

data = np.loadtxt('data01.csv', delimiter=',')
```

```
m, n_plus_1 = data.shape
n = n_plus_1 - 1
Xn = data[:, 0:n].reshape(m, n)

# Dodaj kolumn jedynek do macierzy
XMx = np.matrix(np.concatenate((np.ones((m, 1)), Xn), axis=1)).reshape(m, n_plus_1)
yMx = np.matrix(data[:, 1]).reshape(m, 1)

def hMx(theta, X):
    return X * theta

def regdotsMx(X, y):
    fig = pl.figure(figsize=(16*.6, 9*.6))
    ax = fig.add_subplot(111)
    fig.subplots_adjust(left=0.1, right=0.9, bottom=0.1, top=0.9)
    ax.scatter([X[:, 1]], [y], c='r', s=50, label='Dane')

    ax.set_xlabel('Populacja')
    ax.set_ylabel('Zysk')
    ax.margins(.05, .05)
    pl.ylim(y.min() - 1, y.max() + 1)
    pl.xlim(np.min(X[:, 1]) - 1, np.max(X[:, 1]) + 1)
    return fig

def reglineMx(fig, fun, theta, X):
    ax = fig.axes[0]
    x0, x1 = np.min(X[:, 1]), np.max(X[:, 1])
    L = [x0, x1]
    LX = np.matrix([1, x0, 1, x1]).reshape(2, 2)
    ax.plot(L, fun(theta, LX), linewidth='2',
            label=(r'$y={theta0:.2}{op}{theta1:.2}x$'.format(
                theta0=float(theta[0][0]),
                theta1=(float(theta[1][0]) if theta[1][0] >= 0 else float(-theta[1][0
                op='+' if theta[1][0] >= 0 else '-')))
```

1.3.4 Funkcja kosztu – notacja macierzowa

$$J(\theta) = \frac{1}{2|\vec{y}|} \left(X\theta - \vec{y} \right)^T \left(X\theta - \vec{y} \right)$$

In [27]:
```
# Wersja macierzowa funkcji kosztu

def JMx(theta,X,y):
    m = len(y)
    J = 1.0 / (2.0 * m) * ((X * theta - y) . T * ( X * theta - y))
    return J.item()

thetaMx = np.matrix([-5, 1.3]).reshape(2, 1)
```

```
cost = JMx(thetaMx,XMx,yMx)
display(Math(r'\Large J(\theta) = %.4f' % cost))
```

$$J(\theta) = 4.5885$$

1.3.5 Gradient – notacja macierzowa

$$\nabla J(\theta) = \frac{1}{|\vec{y}|} X^T \left(X\theta - \vec{y} \right)$$

```
In [28]: # Wersja macierzowa gradientu funkcji kosztu

         def dJMx(theta,X,y):
             return 1.0 / len(y) * (X.T * (X * theta - y))

         thetaMx2 = np.matrix([-5, 1.3]).reshape(2, 1)

         display(Math(r'\large \theta = ' + LatexMatrix(thetaMx2) +
                      r'\quad' + r'\large \nabla J(\theta) = '
                      + LatexMatrix(dJMx(thetaMx2,XMx,yMx))))
```

$$\theta = \begin{bmatrix} -5.0000 \\ 1.3000 \end{bmatrix} \quad \nabla J(\theta) = \begin{bmatrix} -0.2314 \\ -0.3027 \end{bmatrix}$$

1.3.6 Algorytm gradientu prostego – notacja macierzowa

$$\theta := \theta - \alpha \, \nabla J(\theta)$$

```
In [29]: # Implementacja algorytmu gradientu prostego za pomoc numpy i macierzy

         def GDMx(fJ, fdJ, theta, X, y, alpha=0.1, eps=10**-3):
             current_cost = fJ(theta, X, y)
             log = [[current_cost, theta]]
             while True:
                 theta = theta - alpha * fdJ(theta, X, y) # implementacja wzoru
                 current_cost, prev_cost = fJ(theta, X, y), current_cost
                 if abs(prev_cost - current_cost) <= eps:
                     break
                 log.append([current_cost, theta])
             return theta, log

         thetaStartMx = np.matrix([0, 0]).reshape(2, 1)

         # Zmieniamy wartoci alpha (rozmiar kroku) oraz eps (kryterium stopu)
         thetaBestMx, log = GDMx(JMx, dJMx, thetaStartMx,
                                 XMx, yMx, alpha=0.01, eps=0.00001)
```

```
########################################################################
display(Math(r'\large\textrm{Wynik:}\quad \theta = ' +
             LatexMatrix(thetaBestMx) +
             (r' \quad J(\theta) = %.4f' % log[-1][0])
             + r' \quad \textrm{po %d iteracjach}' % len(log)))
```

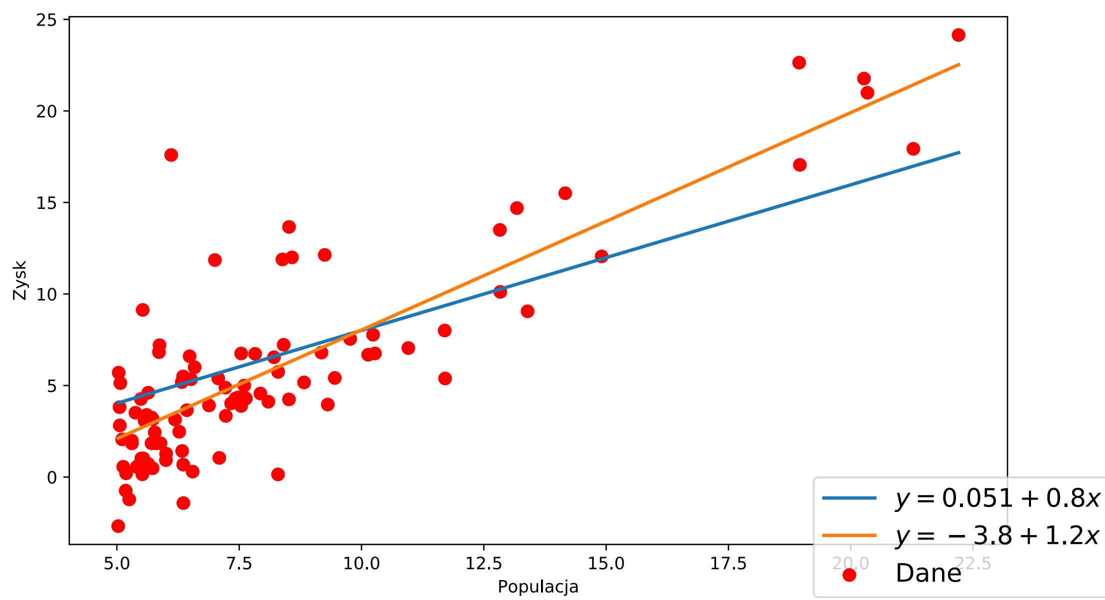
$$\text{Wynik:}\quad \theta = \begin{bmatrix} -3.7217 \\ 1.1755 \end{bmatrix} \quad J(\theta) = 4.4797 \quad \text{po 1734 iteracjach}$$

1.4 1.7. Metoda gradientu prostego w praktyce

1.4.1 Kryterium stopu

Na wykresie zobaczymy porównanie regresji dla rónych wartoci eps

```
In [30]: fig = regdotsMx(XMx, yMx)
         theta_e1, log1 = GDMx(JMx, dJMx, thetaStartMx, XMx, yMx, alpha=0.01, eps=0.01)
         reglineMx(fig, hMx, theta_e1, XMx)
         theta_e2, log2 = GDMx(JMx, dJMx, thetaStartMx, XMx, yMx, alpha=0.01, eps=0.000001)
         reglineMx(fig, hMx, theta_e2, XMx)
         legend(fig)
```




```
In [31]: display(Math(r'\theta_{10^{-2}} = ' + LatexMatrix(theta_e1) +
                      r'\quad\theta_{10^{-6}} = ' + LatexMatrix(theta_e2)))
```

$$\theta_{10^{-2}} = \begin{bmatrix} 0.0511 \\ 0.7957 \end{bmatrix} \quad \theta_{10^{-6}} = \begin{bmatrix} -3.8407 \\ 1.1875 \end{bmatrix}$$

1.4.2 Dugo kroku (α)

```
In [32]: # Jak zmienia si koszt w kolejnych krokach w zalenoci od alfa

         def costchangeplot(logs):
             fig = pl.figure(figsize=(16*.6, 9*.6))
             ax = fig.add_subplot(111)
             fig.subplots_adjust(left=0.1, right=0.9, bottom=0.1, top=0.9)
             ax.set_xlabel('krok')
             ax.set_ylabel(r'$J(\theta)$')

             X = np.arange(0, 500, 1)
             Y = [logs[step][0] for step in X]
             ax.plot(X, Y, linewidth='2', label=(r'$J(\theta)$'))
             return fig

         def slide7(alpha):
             best_theta, log = gradient_descent(h, J, [0.0, 0.0], x, y, alpha=alpha, eps=0.000
             fig = costchangeplot(log)
             legend(fig)

         sliderAlpha1 = widgets.FloatSlider(min=0.01, max=0.03, step=0.001, value=0.02, descri
```

```
In [33]: widgets.interact_manual(slide7, alpha=sliderAlpha1)
```

```
interactive(children=(FloatSlider(value=0.02, description='$\\alpha$', max=0.03, min=0.01, ste
```

```
Out[33]: <function __main__.slide7>
```

1.5 1.8 Regresja liniowa – dodatek

1.5.1 Regresja liniowa za pomoc macierzy normalnej

Zamiast korzysta z algorytmu gradientu prostego moemy bezporednio obliczy minimum $J(\theta)$ dla regresji liniowej ze wzoru:

$$\theta = \left(X^T X\right)^{-1} X^T \vec{y}$$