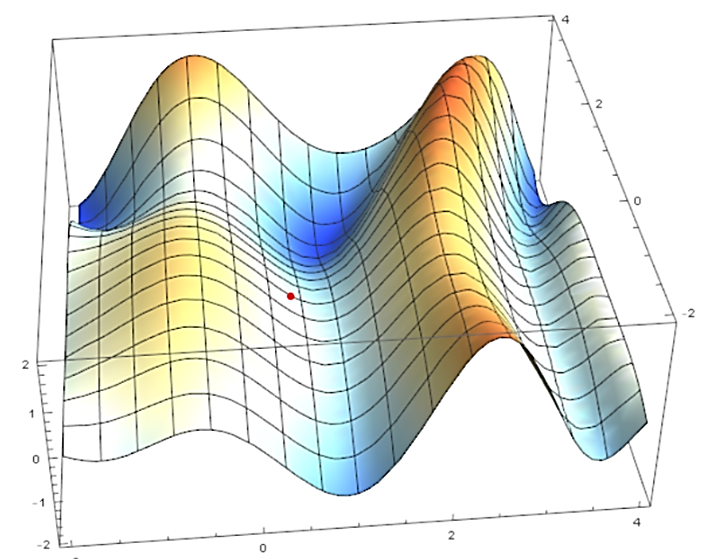
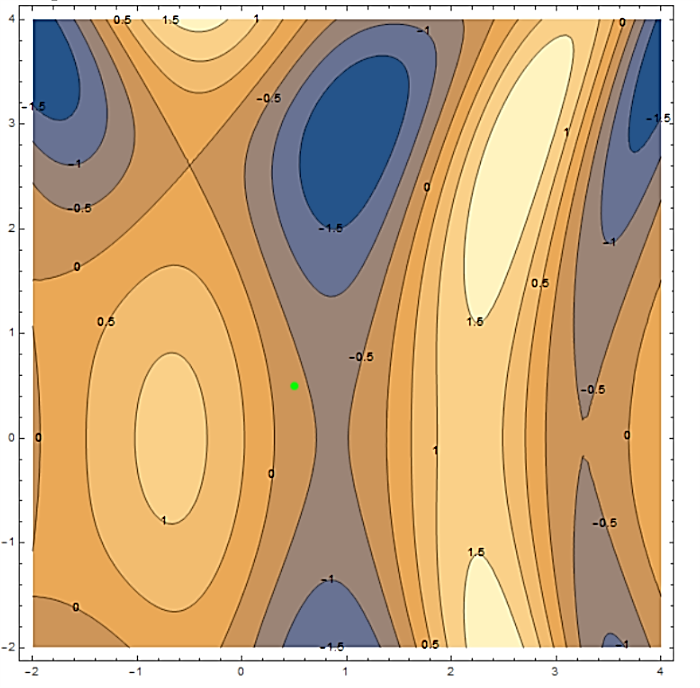
**Андрій Дідус, Олександр Коровій  
(Київ, Україна)**

**ВИКОРИСТАННЯ ГРАДІЄНТНОГО МЕТОДУ В**

**МАШИННОМУ НАВЧАННІ**

Градієнтні методи – це широкий клас оптимізаційних алгоритмів, які використовуються не тільки в машинному навчанні, а й в інших наукових сферах таких, як математична статистика, економіка та іншi. В нашій статті ми розглянемо стохастичний градієнтний метод оптимізації похибки.

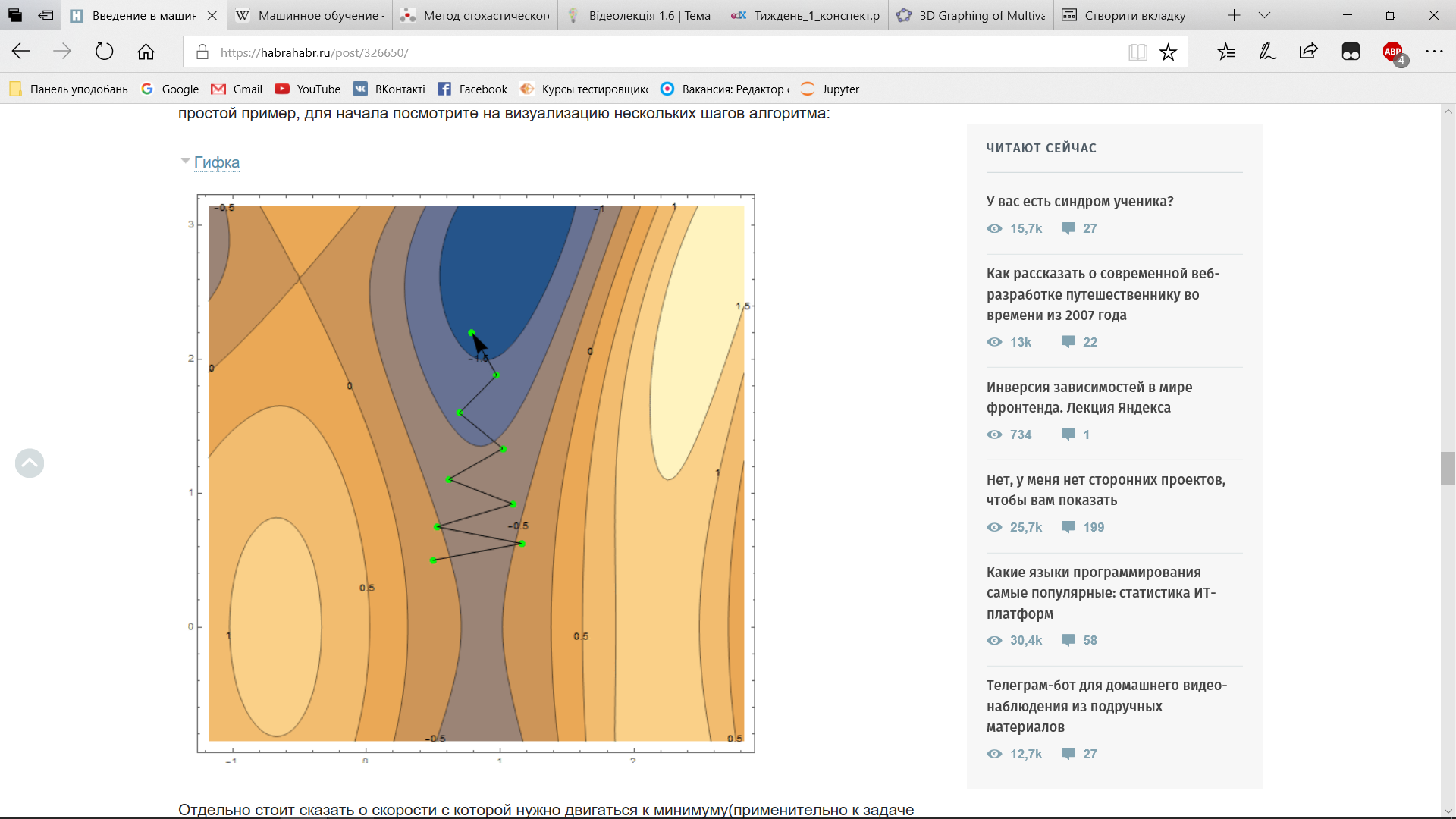
Нижче представлені два варіанти графіка .



Задача методу – знайти локальний мінімум, тобто з взятої випадково точки (наприклад (, як на нашому графіку) потрапити в заглиблення (синя зона на графіках).



Суть методу полягає в тому, щоб йти у напрямку протилежному градієнту функції в даній точці. Градієнт – це вектор, який вказує напрям найбільшого росту функції. Математично −це вектор із похідних по всіх аргументах:grad(f)=∇f=(∂f/∂x,∂f/∂y).Розглянемо покрокову роботу алгоритму на випадковій функції:



Окремо варто сказати про швидкість, з якою потрібно рухатись до мінімума (якщо говорити про застосування у машинному навчанні, то це матиме назву «швидкість навчання»). Для отримання перших результатів, нам буде достатньо підібрати фіксовану швидкість. Але часто застосовується ідея зменшувати швидкiсть по мірі виконання алгоритму, тобто рухатися все меншими і меншими кроками.

У наступному прикладі ми спробуємо відновити значення функції «2х-3» на інтервалі [-2;2] за 50-тьма точками з нормально розподіленим шумом. Навчати модель будемо пакетами по 5 точок за допомогою стохастичного градієнтного спуску (англ. SGD — StochasticGradientDescent). Перейдемо до програмної реалізації мовою Python, адже саме Python став загальновизнаною мовою для багатьох сфер застосування науки про дані (datascience). Він поєднує в собі потужність мов програмування з простотою використання предметно-орієнтованих скриптових мов, як MATLAB або R. А також в Python вже є бібліотеки для завантаження даних, візуалізації та статистичних обчислень.

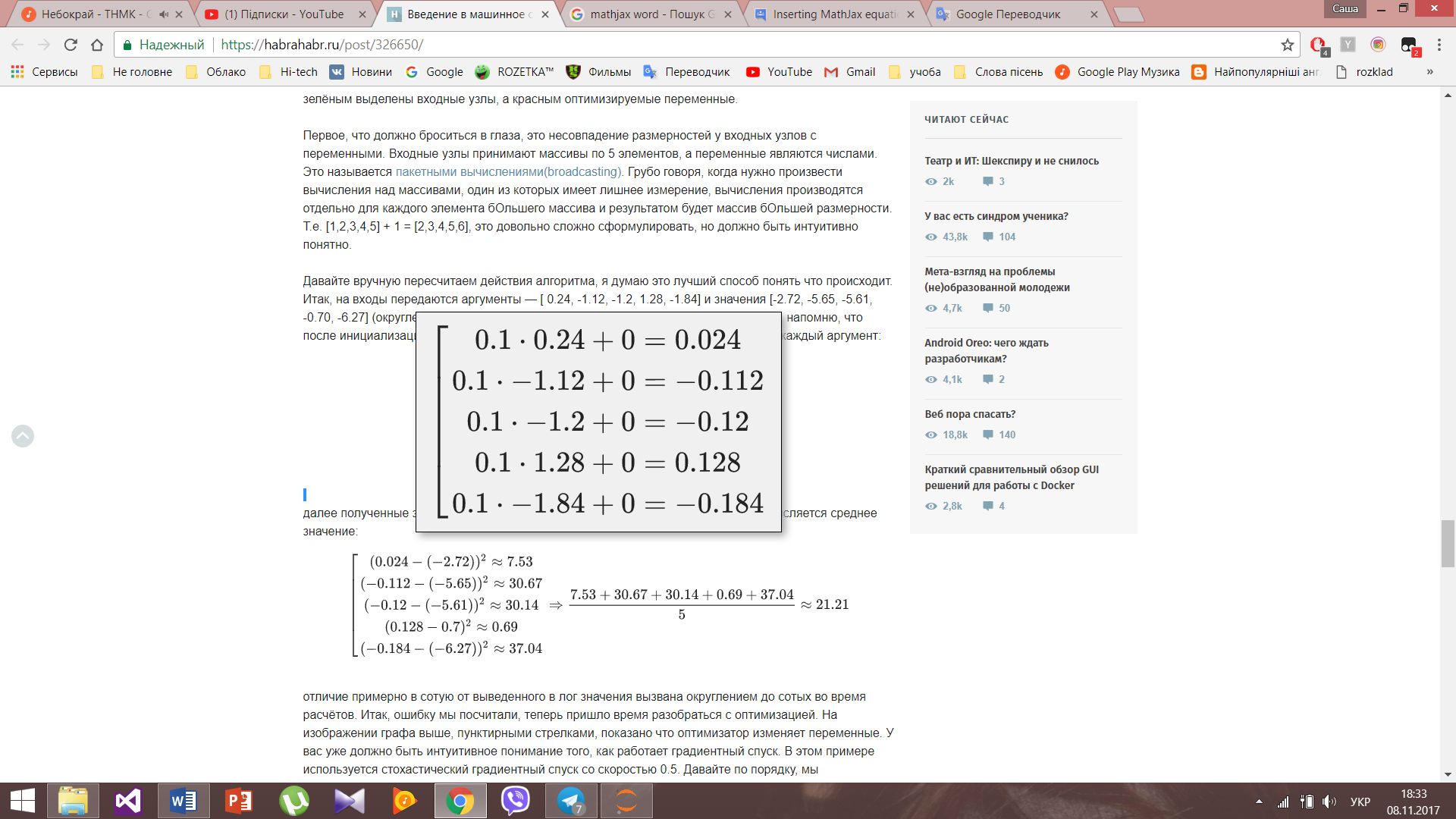
|  |
| --- |
| import numpy as np  import tensorflow as tf  %matplotlib inline  import matplotlib.pyplot as plt  samples = 50 # кількість точок  packetSize = 5 # розмір пакета  def f(x): return 2\*x-3 # шуканна функція  x\_0 = -2 # початок інтервалу  x\_l = 2 # кінець інтервалу  sigma = 0.5 # середньоквадтратичне відхилення шуму  np.random.seed(0) # робимо випадковість передбачуваною (щоб всі бажаючі могли повторити обчислення на цьому ж наборі даних)  data\_x = np.arange(x\_0,x\_l,(x\_l-x\_0)/samples) # массив [-2, -1.92, -1.84, ..., 1.92, 2]  np.random.shuffle(data\_x) #перемішати, але не збовтувати  data\_y = list(map(f, data\_x)) + np.random.normal(0, sigma, samples) # массив значень функції з шумом  print(",".join(list(map(str,data\_x[:packetSize])))) # перший пакет іксів  print(",".join(list(map(str,data\_y[:packetSize])))) # и первый пакет ігриків  tf\_data\_x = tf.placeholder(tf.float32, shape=(packetSize,)) # вузол на який будемо подавати аргументи функції  tf\_data\_y = tf.placeholder(tf.float32, shape=(packetSize,)) # вузол на який будемо подавати значення функції  #коефіцієнти які ми будемо шукати за допомогою Градієнтного спуску  weight = tf.Variable(initial\_value=0.1, dtype=tf.float32, name="a")  bias = tf.Variable(initial\_value=0.0, dtype=tf.float32, name="b")  model = tf.add(tf.multiply(tf\_data\_x, weight), bias)#наша модель(функція y=W\*x+b)  loss = tf.reduce\_mean(tf.square(model-tf\_data\_y)) # функція втрат, про неї нижче  optimizer = tf.train.GradientDescentOptimizer(0.5).minimize(loss) # Сам метод оптимізації  with tf.Session() as session:  tf.global\_variables\_initializer().run()#ініціалізуєм всі змінні  for i in range(samples//packetSize):  #подаєм наші данні пакетами  feed\_dict={tf\_data\_x: data\_x[i\*packetSize:(i+1)\*packetSize], tf\_data\_y: data\_y[i\*packetSize:(i+1)\*packetSize]}  # запускаєм наш метод оптимізації і знаходим наші "втрати"  \_, l = session.run([optimizer, loss], feed\_dict=feed\_dict)  print("Помилка: %f" % (l, ))  print("a = %f, b = %f" % (weight.eval(), bias.eval()))  plt.plot(data\_x, list(map(lambda x: weight.eval()\*x+bias.eval(), data\_x)), data\_x, data\_y, 'ro') |

**Вивід:**

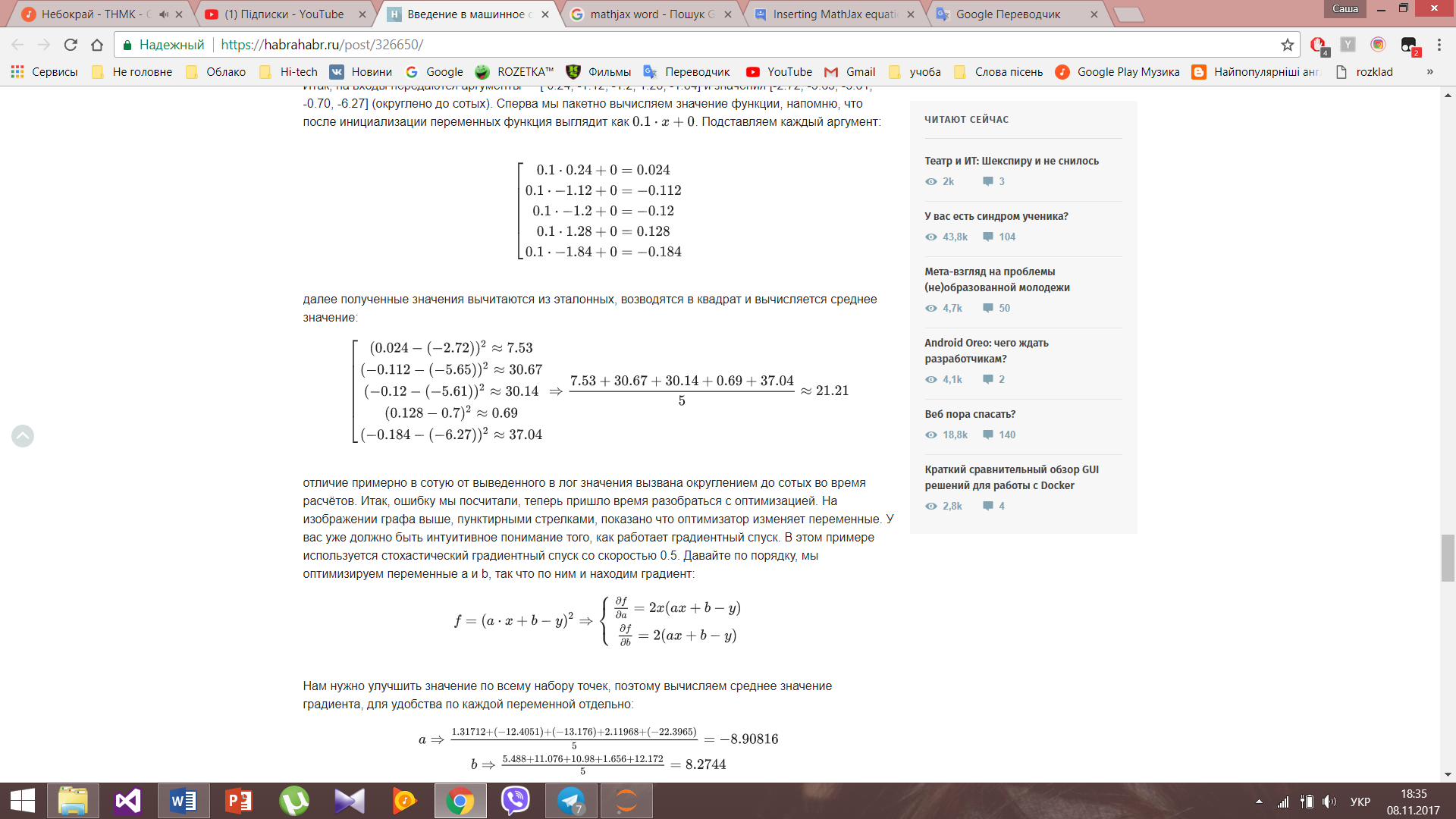
|  |
| --- |
| 0.24,-1.12,-1.2,1.28,-1.84  -2.71939308668,-5.65359826674,-5.60787235079,-0.702256096866,-6.27344936408  Помилка: 21.224598  a = 4.555096, b = -4.138514  Помилка: 6.941454  a = 2.484084, b = -2.752421  Помилка: 0.399143  a = 2.223793, b = -3.131071  Помилка: 0.238383  a = 2.134463, b = -2.752734  Помилка: 1.017416  a = 1.630625, b = -3.502803  Помилка: 0.928532  a = 1.964050, b = -2.789906  Помилка: 0.096641  a = 2.248643, b = -3.046779  Помилка: 0.340161  a = 2.146654, b = -2.895813  Помилка: 0.134945  a = 1.902377, b = -2.817182  Помилка: 0.528846  a = 2.276279, b = -3.158037 |

Вхідні вузли приймають масиви по 5 елементів, а змінні є числами. Це називається пакетними обчисленнями (broadcasting). Iншими словами, коли потрібно зробити обчислення над масивами, один з яких має зайвий вимір, обчислення проводяться окремо для кожного елемента більшого масиву і результатом буде масив бiльшої розмірності. Тобто [1,2,3,4,5] + 1 = [2,3,4,5,6].

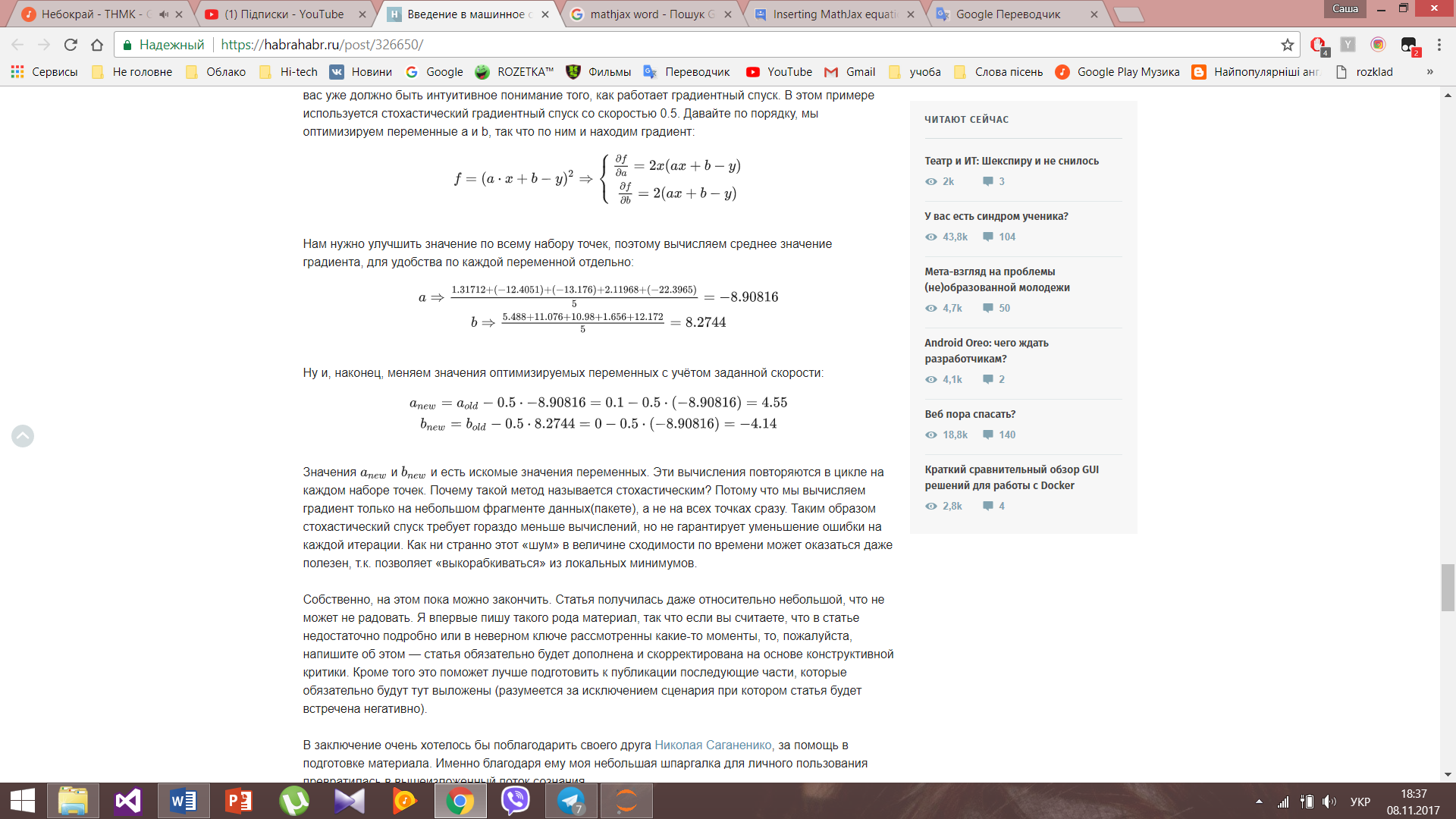
Давайте вручну перерахуємо дії алгоритму, щоб краще зрозуміти, що відбувається. Отже, на входи передаються аргументи - [0.24, -1.12, -1.2, 1.28, -1.84] і значення [-2.72, -5.65, -5.61, -0.70, -6.27] (округлено до сотих). Спершу ми пакетно обчислюємо значення функції(після ініціалізації змінних функція виглядає як 0.1⋅x + 0).Підставляємо кожен аргумент:



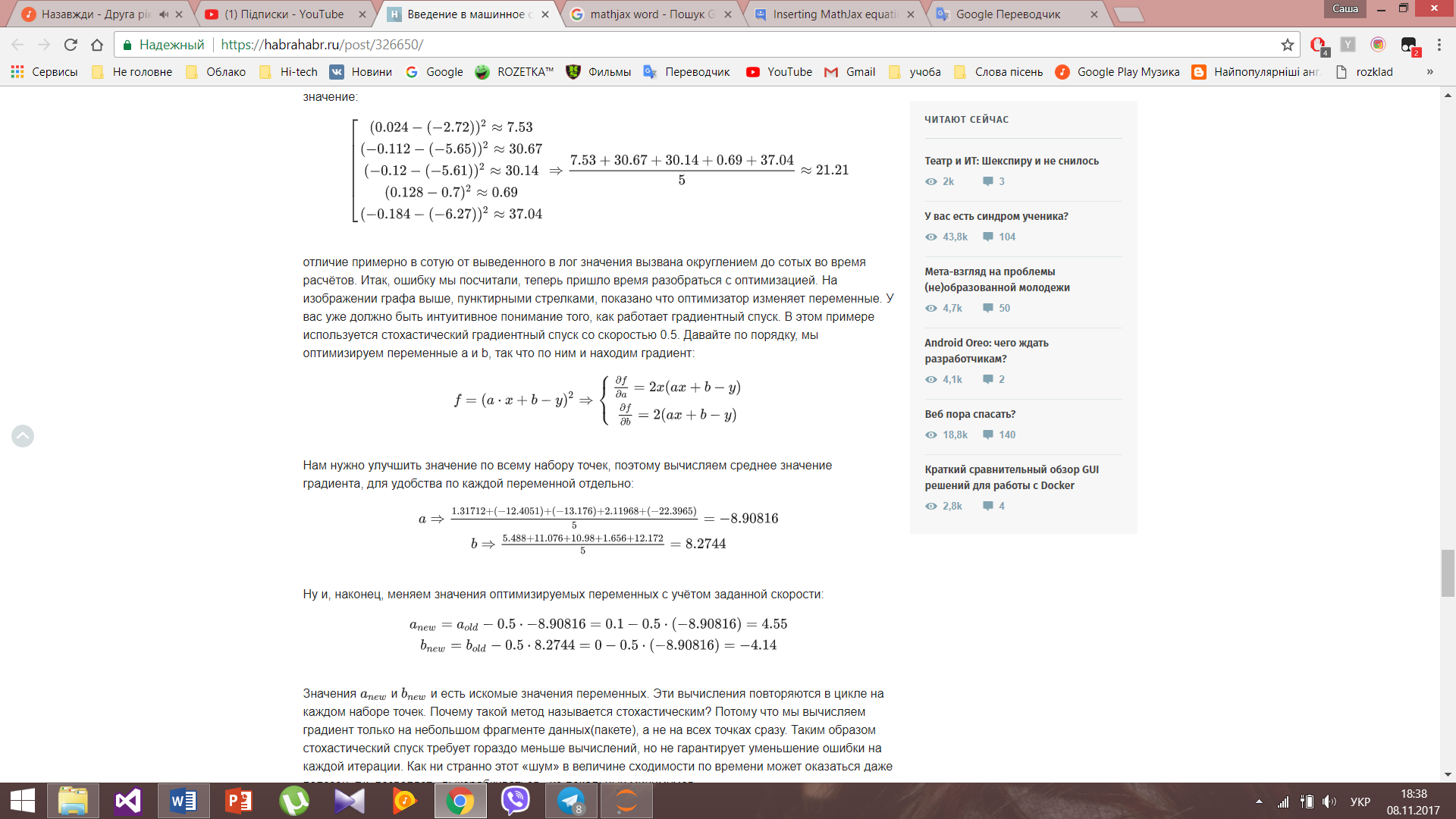
далі отримані значення віднімаються з еталонних, зводяться до квадрату і обчислюється середнє значення:



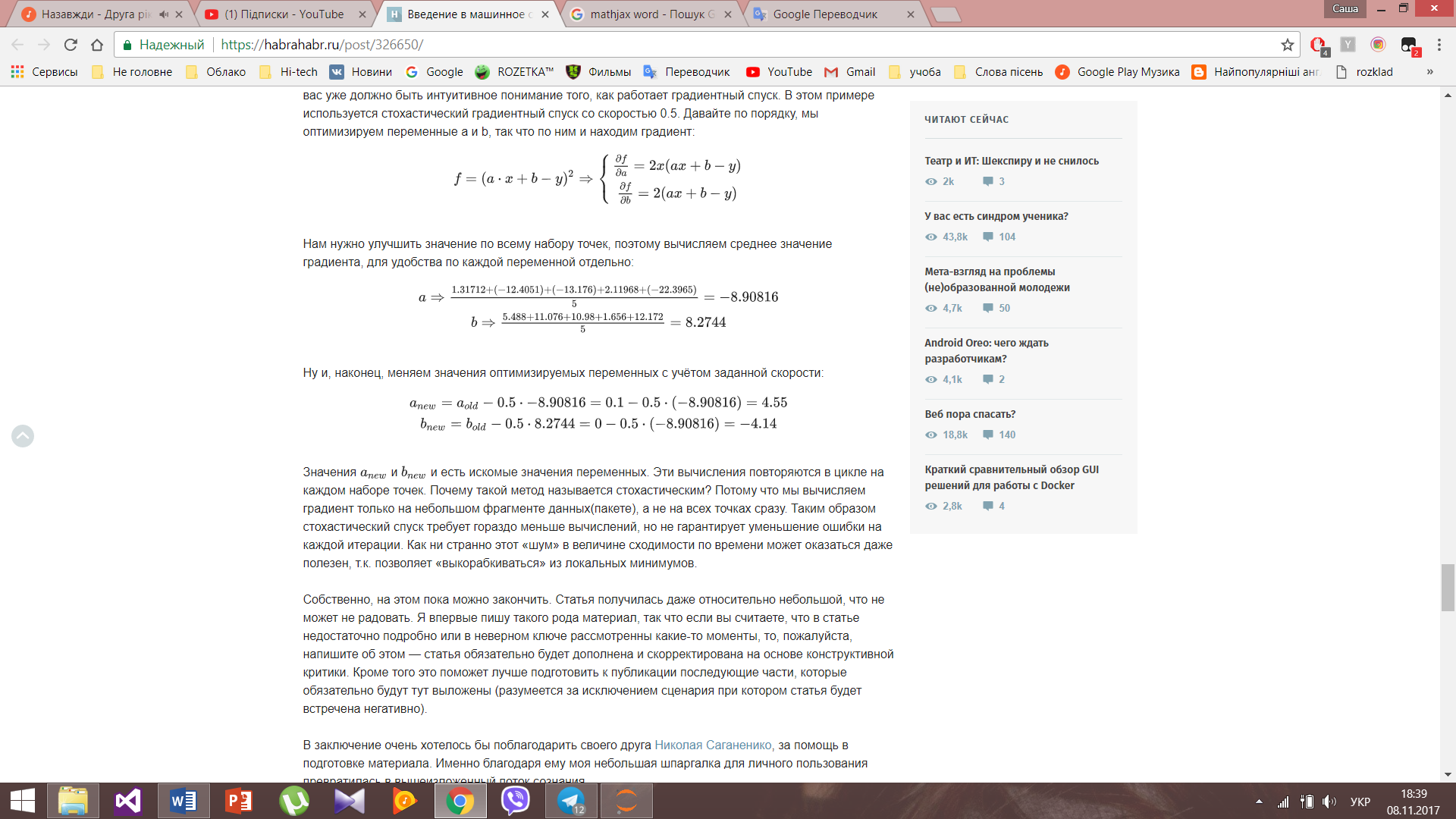
Відмінність приблизно в соту від виведеного в лог значення обумовлена округленням до сотих під час розрахунків. Отже, похибку ми порахували, тепер розглянемо оптимізацію. На зображенні графа вище, пунктирними стрілками показано, що оптимізатор змінює змінні. У вас вже має бути інтуїтивне розуміння того, як працює градієнтний спуск. У цьому прикладі використовується стохастичний градієнтний спуск зі швидкістю 0.5. Ми оптимізуємо змінні *a* і *b* так, що по ниx і знаходимо градієнт:



Нам потрібно поліпшити значення по всьому набору точок, тому обчислюємо середнє значення градієнта, для зручності по кожній змінній окремо:



Ну і, нарешті, міняємо значення оптимізуються змінних з урахуванням заданої швидкості:



Значення ***anew*** і ***bnew*** і є шукані значення змінних. Ці обчислення повторюються в циклі на кожному наборі точок. Чому такий метод називається стохастичним? Тому що ми обчислюємо градієнт тільки на невеликому фрагменті даних (пакеті), а не на всіх точках відразу. Таким чином, стохастический спуск вимагає набагато менше обчислень, але не гарантує зменшення помилки на кожній ітерації. Як не дивно цей «шум» в величині збіжності за часом може виявитися навіть корисний, тому що дозволяє «викарабкуватися» з локальних мінімумів.

Варто зауважити, що градієнтний спуск – це метод оптимізації першого порядку, оскільки беретьсяпохідна першого порядку. Також існує багато методів другуго порядку, в яких знаходиться друга похідна, наприклад,спряжені градієнти, метод Ньютона та інші. Загалом, оптимізація – це величезний і окремий дуже широкий пласт проблем. Хоча багато науковців і практиків, які працюють в цій області, вважають, що можна не ускладнювати і просто використовувати градієнтні методи. Навіть зараз багато сучасних ШНМ використовують градієнтні методи.

**Література**:

1.Carl E. Rasmussen, Christopher K. I. Williams. Gaussian Processes for Machine Learning. –The MIT Press, 2005. – 266 p.

2. Hal Daumé III. A Course in Machine Learning. – ciml.info, 2012. – 189 p.

3. Tariq Rashid.Make Your Own Neural Network. –Paperback, 2016. – 222 p.

4. Андреас Мюллер, Сара Гвидо.Введение в машинное обучение с помощью Python. Руководство для специалистов по работе с данными. – Самиздат 2016. – 390 стр.

5. Бринк Х., Ричардс Д., Феверолф М. Машинное обучение. – Издательский дом «Питер». – 336 стр.

**Науковий керівник:**

кандидат фізико-математичних наук, доцент Барановська Леся Валеріївна.