ECE5984 – Applications of Machine Learning
Lecture 15 – Gradient-Based Methods

Creed Jones, PhD

BRADLEY DEPARTMENT
OF ELECTRICAL & COMPUTER ENGINEERING

VIRGINIA TECH.

Course update

- Next quiz on Thursday, March 24
- HW4 will be posted next Monday
  - Due April 5
- Project I
  - Hope all is moving along
- Wednesday office hours are changing slightly
  - 1:30 to 3 PM, instead of 1:30 to 3:30

Today’s Objectives

Gradient-based methods

- Linear Regression
- Multivariate Linear Regression
- Function Optimization
- Gradient Descent
- An Example

LINEAR REGRESSION

Machine learning using gradient descent involves finding the optimal set of parameters for a model, generally by iterative means

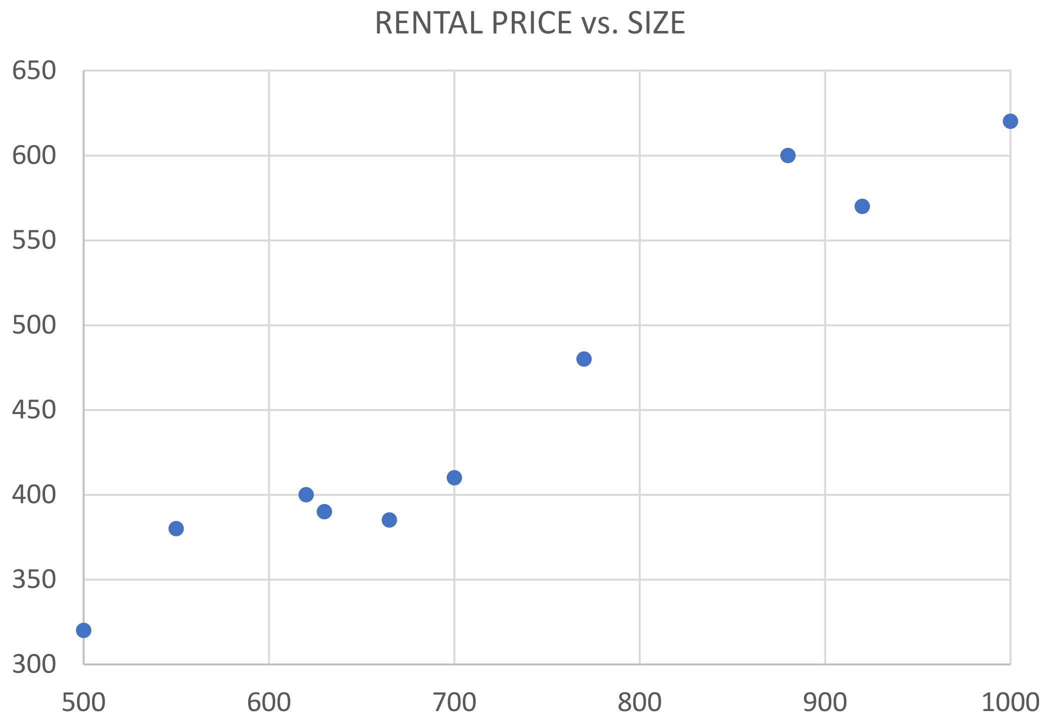
- A parameterized prediction model is initialized with a set of random parameters and an error function is used to judge how well this initial model performs when making predictions for instances in a training dataset.
- Based on the value of the error function the parameters are iteratively adjusted to create a more and more accurate model.
- This can be done for any modeling architecture based on a set of parameters
  - Polynomial regression
  - Support vector machines
  - Artificial neural networks
- Let’s look at simple linear regression to develop the idea

Consider a dataset that includes office rental prices and a number of descriptive features for 10 Dublin city-center offices

ID	Size	Floor	Broadband Rate	Energy Rating	Rental Price
1	500	4	8	C	320
2	550	7	50	A	380
3	620	9	7	A	400
4	630	5	24	B	390
5	665	8	100	C	385
6	700	4	8	B	410
7	770	10	7	B	480
8	880	12	50	A	600
9	920	14	8	C	570
10	1,000	9	24	B	620

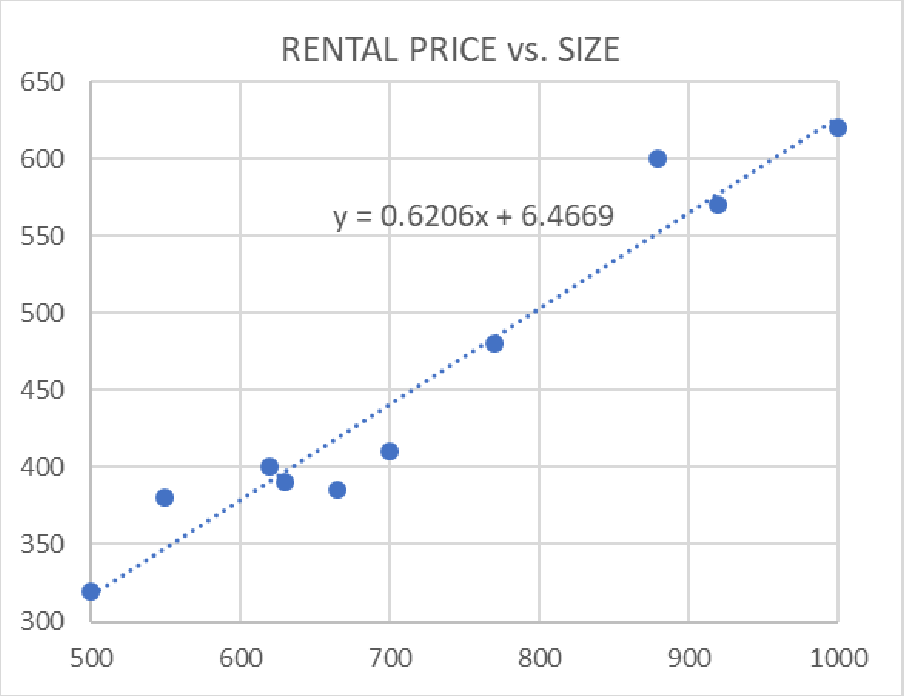
Examine a scatterplot of RENTAL PRICE vs. SIZE – there is obviously an approximate linear relationship

ID	SIZE	FLOOR	BROADBAND RATE	ENERGY RATING	RENTAL PRICE
1	500	4	8	C	320
2	550	7	50	A	380
3	620	9	7	A	400
4	630	5	24	B	390
5	665	8	100	C	385
6	700	4	8	B	410
7	770	10	7	B	480
8	880	12	50	A	600
9	920	14	8	C	570
10	1,000	9	24	B	620



For a simple linear regression model, using MSE, we can calculate the parameters directly

ID	Size	Rental Price
1	500	320
2	550	380
3	620	400
4	630	390
5	665	385
6	700	410
7	770	480
8	880	600
9	920	570
10	1,000	620



The equations for the slope m and intercept b of the best line are (see https://www.itl.nist.gov/div898/handbook/pmd/section4/pmd431.htm):

$$m = \frac{\sum_{i=1}^{n}(x_i - \mu_x)\left(y_i - \mu_y\right)}{\sum_{i=1}^{n}(x_i - \mu_x)^2}$$

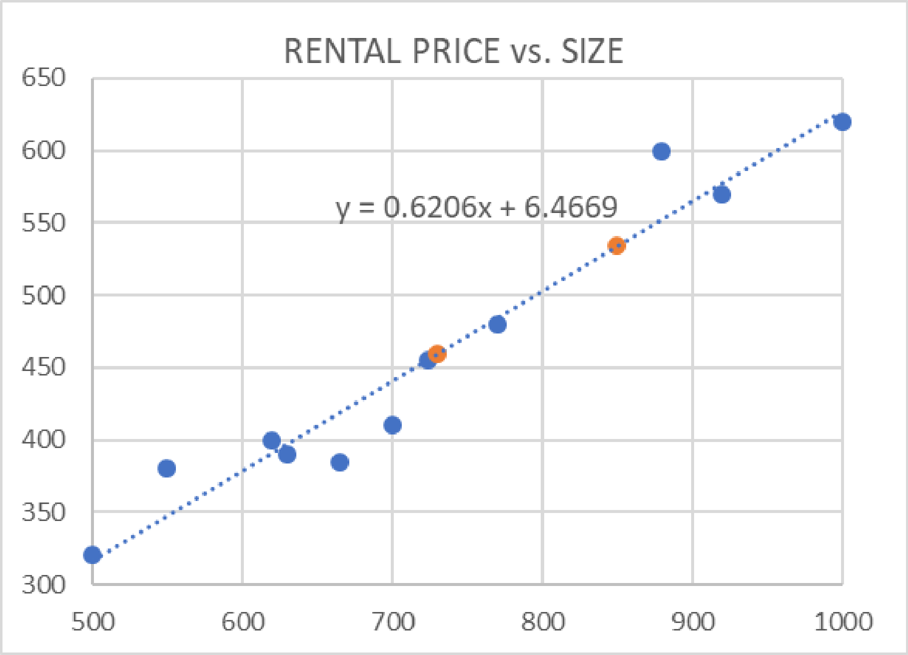
$$= \frac{\sum_{i=1}^{n}(x_i - 723.5)\left(y_i - 455.5\right)}{\sum_{i=1}^{n}(x_i - 723.5)^2} = 0.62064$$

$$b = \mu_y - m\mu_x$$

$$= 455.5 - 0.62064 \cdot 723.5 = 6.4669$$

Now that we have a parametric model, can we estimate the rent for a 730 square foot office?

ID	Size	Rental Price
1	500	320
2	550	380
3	620	400
4	630	390
5	665	385
6	700	410
7	770	480
8	880	600
9	920	570
10	1,000	620



Our model is:

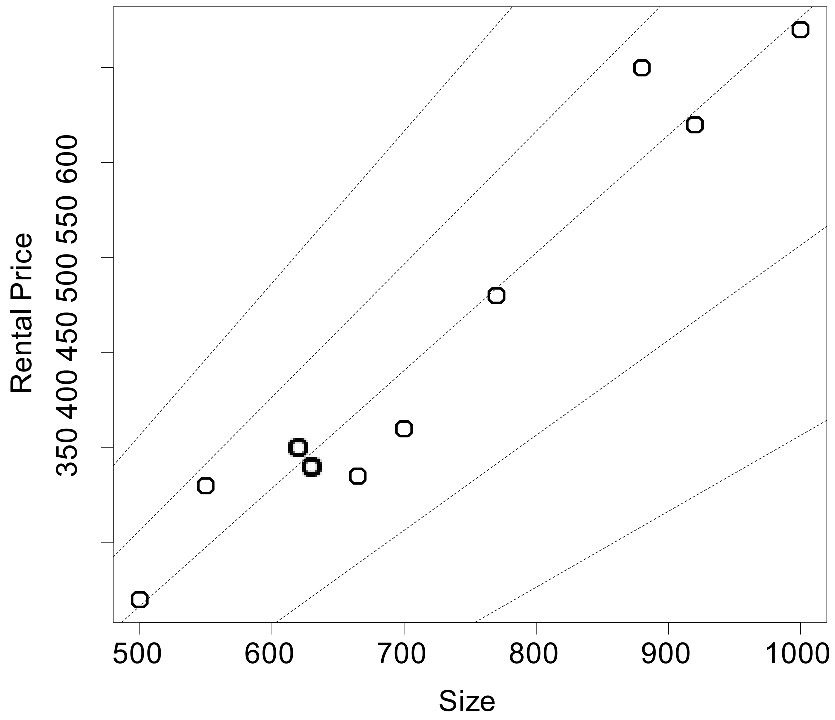
$$0.62064 \cdot SIZE + 6.4669$$

$$0.62064 \cdot (730) + 6.4669 = 459.53$$

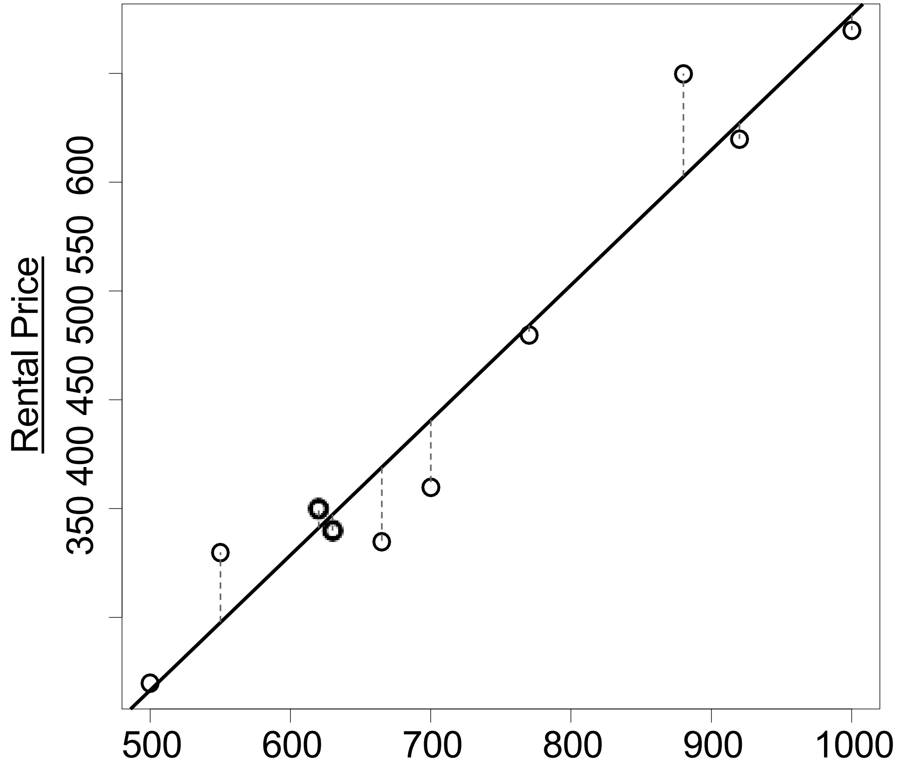
What if *SIZE*=850?

$$0.62064 \cdot (850) + 6.4669 = 534.01$$

Using the minimum square error criterion produces the "best" (in some sense) model from a set of possibilities

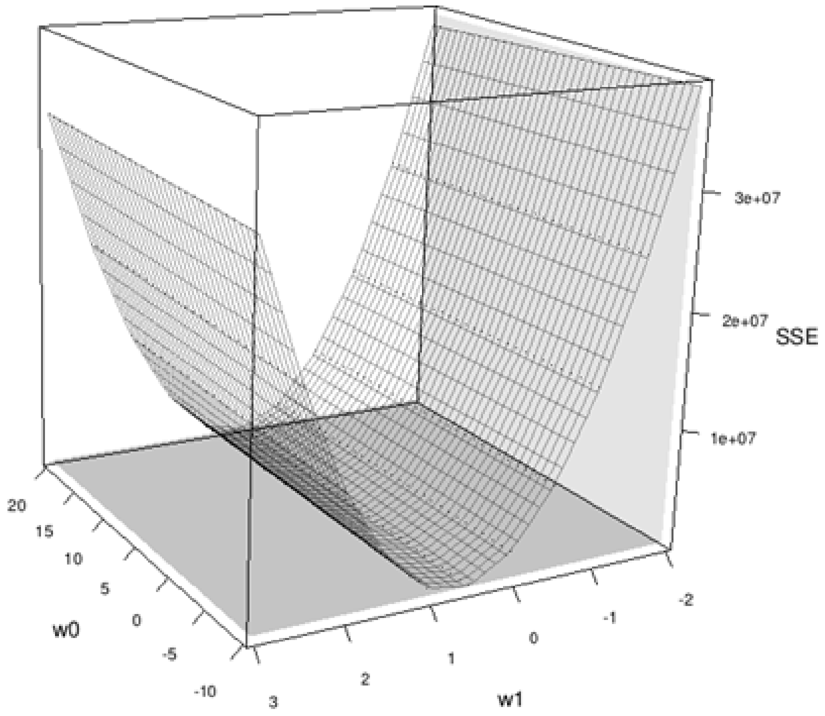


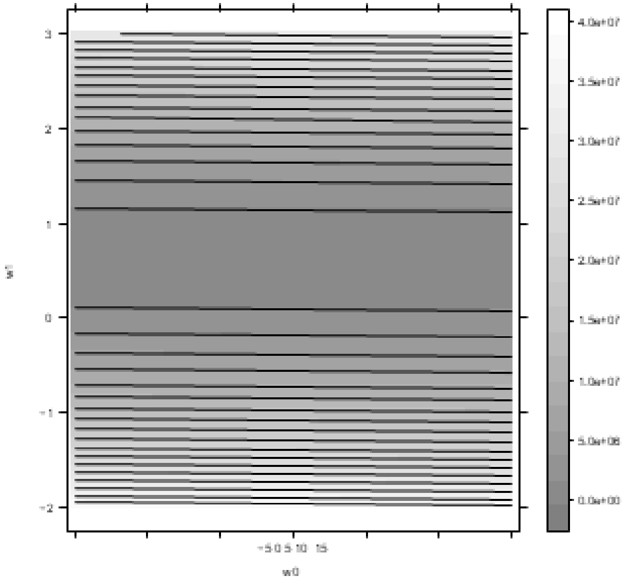
A scatter plot of SIZE and RENTAL PRICE, as well as a collection of possible simple linear regression models. For all models ***b*** is set to 6.47. From top to bottom the models use 0.4, 0.5, 0.62, 0.7 and 0.8 respectively for **m.**



A scatter plot of SIZE and RENTAL PRICE, showing a candidate prediction model (with ***b*** = 6.47 and ***m*** = 0.62) and the resulting errors.

For every possible combination of parameters ***m*** and ***b***, there is a corresponding sum of squared errors value that can be joined together to make a surface – this defines the *Error Surface*





A 3D surface plot and contour plot of the error surface generated by sum of squared errors for the office rentals training set;
-10 < ***b*** < 20
-2 < ***m*** < 3

The *x* -*y* plane is known as a *weight space* and the surface is known as an *error surface*.
The model that best fits the training data is the model corresponding to the lowest point on the error surface.

Finding the optimal point can be considered a minimization problem

- Using Equation 4 in the book, $L_2(\mathbb{M}_W, \mathcal{D}) = \frac{1}{2}\sum_{i=1}^{t}\left(t_i - (md_i + b)\right)^2$, we can formally define this point on the error surface as the point at which:

$$\frac{\partial}{\partial m}\frac{1}{m}\sum_{i=1}^{t}\left(t_i - (md_i + b)\right)^2 = 0 \text{ and } \frac{\partial}{\partial b}\frac{1}{m}\sum_{i=1}^{t}\left(t_i - (md_i + b)\right)^2 = 0$$

- In the general case of a multiparameter model,

$$\frac{\partial}{\partial w_k}\frac{1}{m}\sum_{i=1}^{t}\left(t_i - f(d_i, \boldsymbol{w})\right)^2 = 0 \text{ for all k}$$

- For simple cases such as univariate linear regression this can be found by direct calculation
- In general we use a guided search method, such as *gradient descent*

MULTIVARIATE LINEAR REGRESSION

Let's define a *multivariate linear regression model* as

$$\mathbb{M}_{\boldsymbol{w}}(\boldsymbol{d}) = w[0] + \sum_{i=1}^{m} \boldsymbol{w}[j] \cdot \boldsymbol{d}[j]$$

ID	Size	Floor	Broadband Rate	Energy Rating	Rental Price
1	500	4	8	C	320
2	550	7	50	A	380
3	620	9	7	A	400
4	630	5	24	B	390
5	665	8	100	C	385
6	700	4	8	B	410
7	770	10	7	B	480
8	880	12	50	A	600
9	920	14	8	C	570
10	1,000	9	24	B	620

non-numeric

We usually add an additional feature that is always 1, to carry along the additive constant $\boldsymbol{w}[0]$:

$$\mathbb{M}_W = \boldsymbol{w} \cdot \boldsymbol{d}$$

The MSE or loss function is now:

$$L_2(\mathbb{M}_W, \mathcal{D}) = \frac{1}{2}\sum_{i=1}^{n}\left(t_i - (\boldsymbol{w}\boldsymbol{d}_i)\right)^2$$

Look at the addition of the constant term $d[0]$ a bit more closely – this forms the *augmented feature vector*

ID	Size	Floor	Broadband Rate	Energy Rating	Rental Price
1	500	4	8	C	320
2	550	7	50	A	380
3	620	9	7	A	400
4	630	5	24	B	390
5	665	8	100	C	385
6	700	4	8	B	410
7	770	10	7	B	480
8	880	12	50	A	600
9	920	14	8	C	570
10	1,000	9	24	B	620

non-numeric

$$\mathbb{M}_W = \boldsymbol{w} \cdot \boldsymbol{d}$$

$$\boldsymbol{d} = \begin{bmatrix} d[0] \\ d[1] \\ d[2] \\ d[3] \end{bmatrix} = \begin{bmatrix} 1 \\ SIZE \\ FLOOR \\ BBRATE \end{bmatrix}, \qquad \boldsymbol{d_{ID=1}} = \begin{bmatrix} 1 \\ 500 \\ 4 \\ 8 \end{bmatrix}$$

$$\boldsymbol{w} \cdot \boldsymbol{d_{ID=1}} = \begin{bmatrix} w[0] \\ w[1] \\ w[2] \\ w[3] \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 500 \\ 4 \\ 8 \end{bmatrix}$$

$$= w[0] + 500w[1] + 4w[2] + 8w[3]$$

The regression equations require numeric values, but if we can attribute a numeric value to categoricals, they can be used in regression

ID	Size	Floor	Broadband Rate	Energy Rating	Rental Price
1	500	4	8	C	320
2	550	7	50	A	380
3	620	9	7	A	400
4	630	5	24	B	390
5	665	8	100	C	385
6	700	4	8	B	410
7	770	10	7	B	480
8	880	12	50	A	600
9	920	14	8	C	570
10	1,000	9	24	B	620

non-numeric

$$\mathbb{M}_{W} = \boldsymbol{w} \cdot \boldsymbol{d}$$

$$\boldsymbol{d} = \begin{bmatrix} d[0] \\ d[1] \\ d[2] \\ d[3] \\ d[4] \end{bmatrix} = \begin{bmatrix} 1 \\ SIZE \\ FLOOR \\ BBRATE \\ ENGRAT \end{bmatrix}, \quad \boldsymbol{d_{ID=1}} = \begin{bmatrix} 1 \\ 500 \\ 4 \\ 8 \\ val(C) \end{bmatrix}$$

$$\boldsymbol{w} \cdot \boldsymbol{d_{ID=1}} = \begin{bmatrix} w[0] \\ w[1] \\ w[2] \\ w[3] \\ w[4] \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 500 \\ 4 \\ 8 \\ val(C) \end{bmatrix}$$

$$= w[0] + 500w[1] + 4w[2] + 8w[3] + val(C) \cdot w[4]$$

To minimize a single metric, $\frac{1}{2}\sum_{i=1}^{n}\left(t_i - (\boldsymbol{w}\boldsymbol{d}_i)\right)^2$, of several parameters (the $\boldsymbol{w}$) is a classic function optimization problem

ID	Size	Floor	Broadband Rate	Energy Rating	Rental Price
1	500	4	8	C	320
2	550	7	50	A	380
3	620	9	7	A	400
4	630	5	24	B	390
5	665	8	100	C	385
6	700	4	8	B	410
7	770	10	7	B	480
8	880	12	50	A	600
9	920	14	8	C	570
10	1,000	9	24	B	620

The optimal set of parameters is:

$$\boldsymbol{w} = \begin{bmatrix} -0.1513 \\ 0.6270 \\ -0.1781 \\ 0.0714 \end{bmatrix}$$

$$RentalPrice \cong -0.1513 + 0.6270 \cdot SIZE \\ -0.1781 \cdot FLOOR + 0.1714 \cdot BBRATE$$

For example:

$$RP(600{,}6{,}20) \cong -0.1513 + 0.6270 \cdot 600 \\ - 0.1781 \cdot 6 + 0.1714 \cdot 20 = 378.40$$

FUNCTION OPTIMIZATION

Function optimization using gradient methods is an iterative process based on updated estimates of the extreme point (local maximum or minimum)

VIRGINIA TECH.

BRADLEY DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

- Simple *gradient ascent*
  - Used to find the maximum value of a function
  - $x_{n+1} = x_n + \gamma_n \nabla f(x_n)$
  - In cases of positive slope, the new point is expected to have a higher value

- *Gradient descent* uses an update equation with a negative scale factor
  - Used to find the minimum value of a function
  - $x_{n+1} = x_n - \gamma_n \nabla f(x_n)$
  - In cases of positive slope, the new point will be in the opposite direction, as we attempting to move "down the hill"

Many gradient methods use an adaptive gamma factor – often, start small and increase as the curve flattens out

- $x_{n+1} = x_n + \gamma_n \nabla f(x_n)$,
  $\gamma_n = gfac \cdot \gamma_{n-1}$

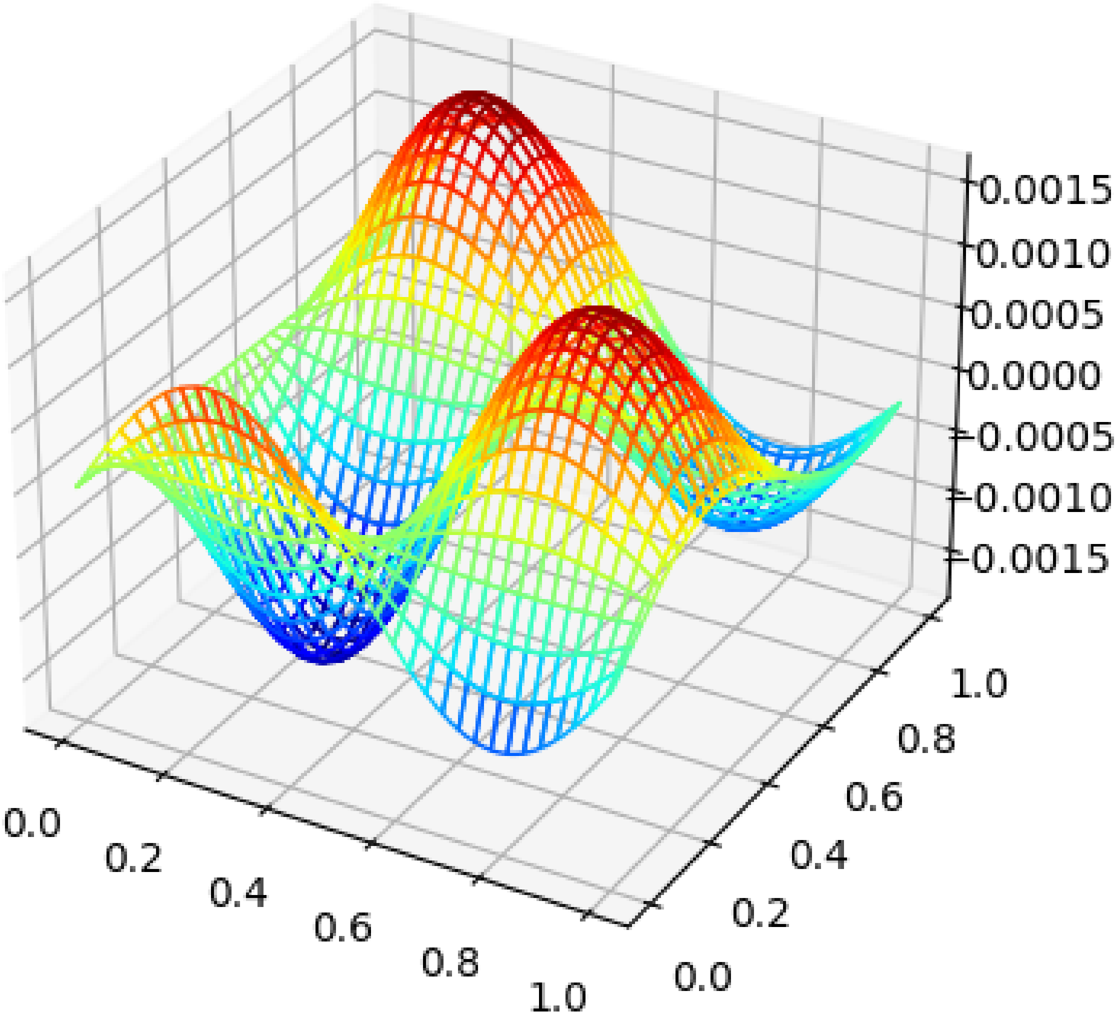
- The values of the start point x_n , the initial gamma γ_0 and the gamma factor $gfac$ are crucial to success

gamma	x	f(x)	f'(x)
0.05	0	-0.004598	0.3344
0.0525	0.01672	0.013993828	1.27405653
0.055125	0.083607968	0.099995584	1.170144748
0.05788125	0.148112197	0.162099472	0.732439864
0.060775313	0.190506732	0.185987043	0.389417735
0.063814078	0.214173716	0.192814936	0.186421421
0.067004782	0.226070028	0.194412594	0.081922569
0.070355021	0.231559231	0.194728717	0.033206891
0.073872772	0.233895503	0.194781982	0.012382175
0.077566411	0.234810209	0.194789573	0.004214361
0.081444731	0.235137101	0.194790473	0.001293434
0.085516968	0.235242445	0.19479056	0.000351928
0.089792816	0.235272541	0.194790566	8.2927E-05
0.094282457	0.235279987	0.194790567	1.637E-05
0.09899658	0.23528153	0.194790567	2.57448E-06
0.103946409	0.235281785	0.194790567	2.96402E-07
0.109143729	0.235281816	0.194790567	2.10111E-08
0.114600916	0.235281818	0.194790567	5.13336E-10
0.120330962	0.235281818	0.194790567	-1.2498E-11
0.12634751	0.235281818	0.194790567	9.44578E-13

The gradient method extends easily into functions of multiple dimensions

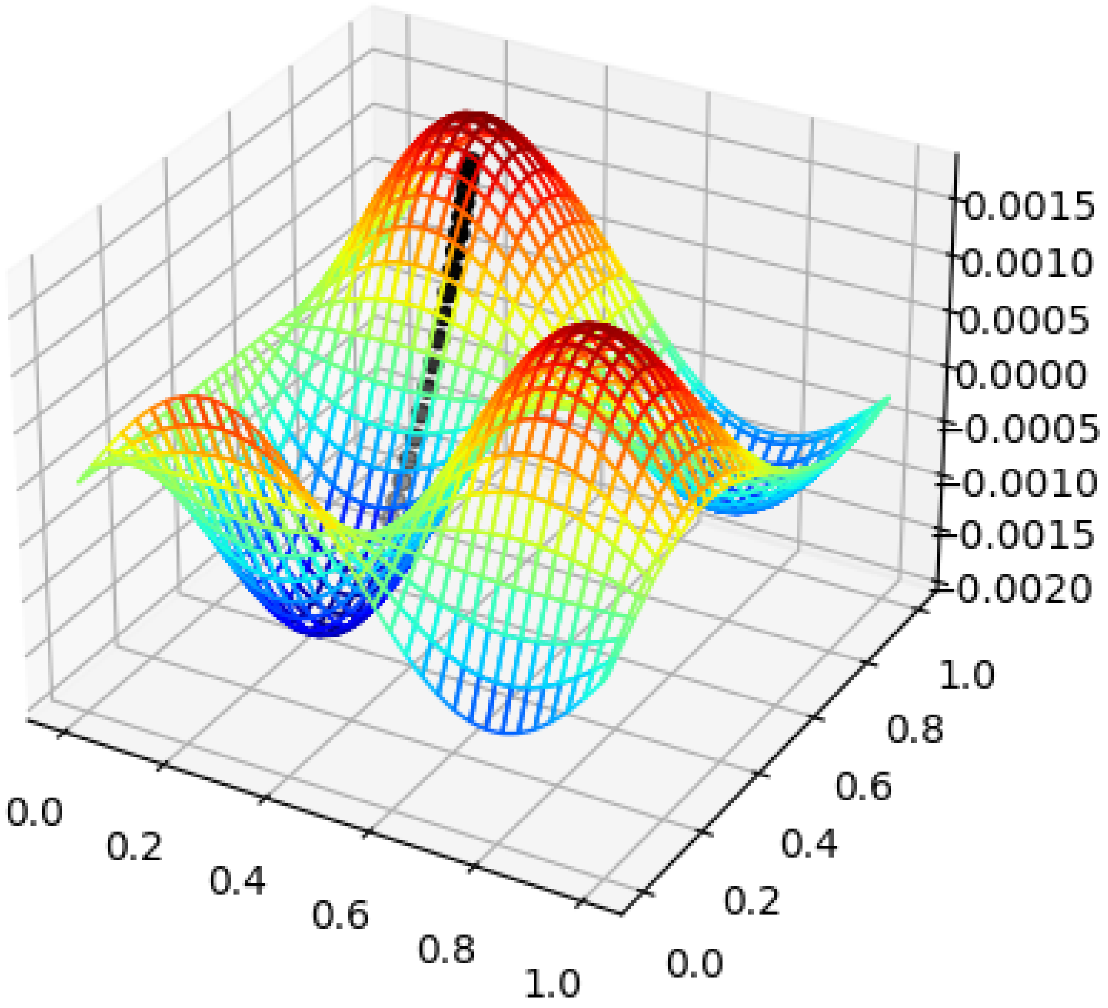
- Write the update equation in terms of vectors
- $x_{n+1} = x_n + \gamma_n \nabla f(x_n) \Longrightarrow \begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \end{bmatrix} + \gamma_n \nabla f(x_n, y_n)$
- recall the definition of the grad operator:
- $\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \end{bmatrix} + \gamma_n \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$

Consider a difficult function of two dimensions…



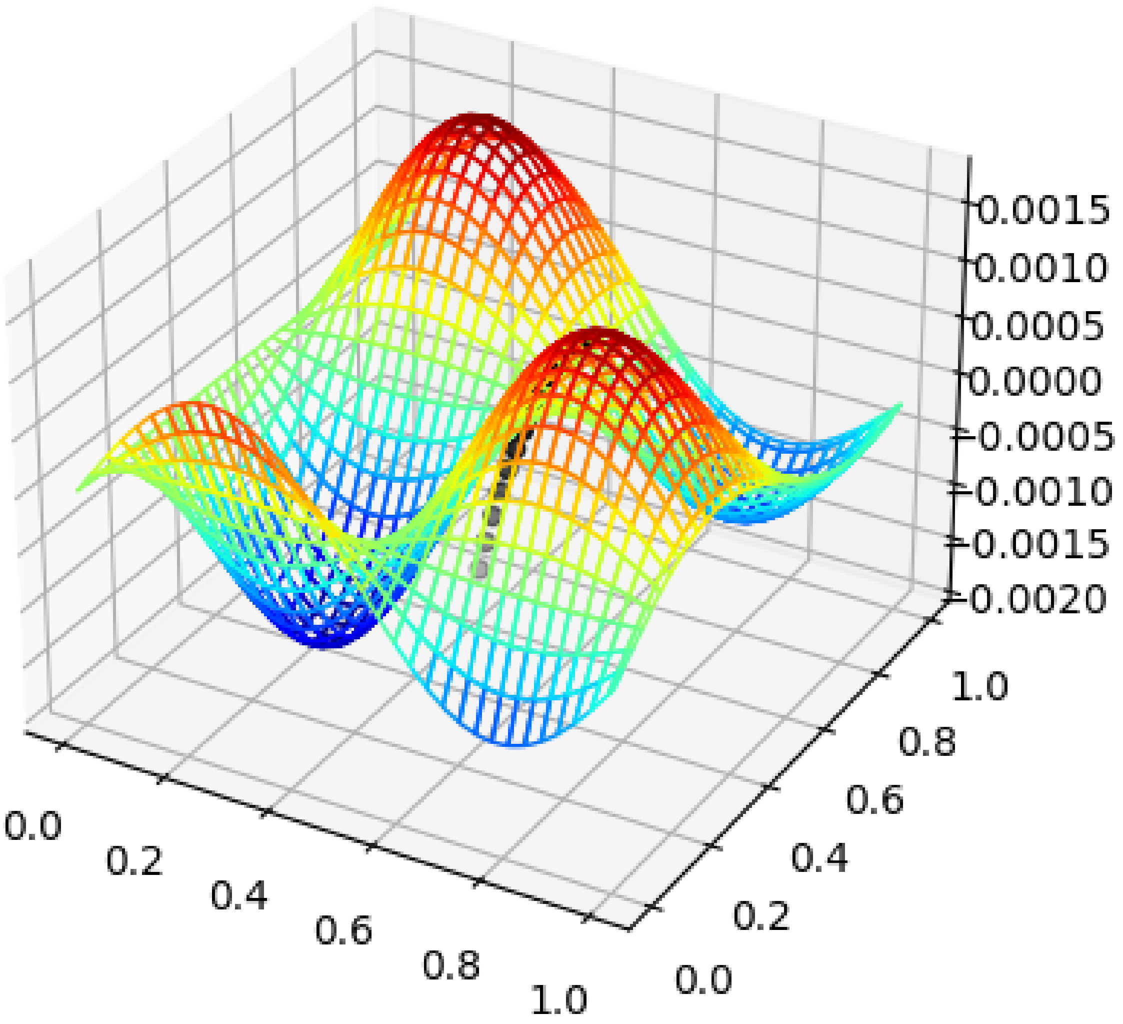
$$= \sin(ax)\cos(by+f)\left(cx+\frac{d}{xy+e}\right)$$

Starting at [0.3, 0.5] with gamma = 2



x, y = (0.3,   0.5), fcn = -0.00111, grad = (0.00155, 0.00769)
x, y = (0.303, 0.515), fcn = -0.000975, grad = (0.00148, 0.00819)
x, y = (0.306, 0.532), fcn = -0.000827, grad = (0.00135, 0.00863)
x, y = (0.309, 0.549), fcn = -0.000667, grad = (0.00116, 0.00901)
x, y = (0.311, 0.567), fcn = -0.000496, grad = (0.000907, 0.00929)
x, y = (0.313, 0.586), fcn = -0.000318, grad = (0.000604, 0.00948)
x, y = (0.314, 0.605), fcn = -0.000136, grad = (0.000264, 0.00956)
x, y = (0.315, 0.624), fcn = 4.73e-05, grad = (-9.31e-05, 0.00953)
x, y = (0.314, 0.643), fcn = 0.000227, grad = (-0.000445, 0.00939)
x, y = (0.314, 0.662), fcn = 0.000401, grad = (-0.000771, 0.00915)
x, y = (0.312, 0.68), fcn = 0.000565, grad = (-0.00105, 0.00883)
x, y = (0.31, 0.697), fcn = 0.000718, grad = (-0.00128, 0.00842)
x, y = (0.307, 0.714), fcn = 0.000857, grad = (-0.00144, 0.00796)
x, y = (0.304, 0.73), fcn = 0.000982, grad = (-0.00154, 0.00745)
x, y = (0.301, 0.745), fcn = 0.00109, grad = (-0.00159, 0.00691)
x, y = (0.298, 0.759), fcn = 0.00119, grad = (-0.00158, 0.00637)
x, y = (0.295, 0.772), fcn = 0.00127, grad = (-0.00154, 0.00582)
x, y = (0.292, 0.783), fcn = 0.00134, grad = (-0.00147, 0.00529)
x, y = (0.289, 0.794), fcn = 0.00139, grad = (-0.00138, 0.00478)
x, y = (0.286, 0.803), fcn = 0.00144, grad = (-0.00128, 0.0043)

Starting at [0.7, 0.1] with gamma = 2



x, y = (0.7,   0.1), fcn = 4.76e-05, grad = (0.000161, 0.00977)
x, y = (0.7, 0.12), fcn = 0.000235, grad = (0.000789, 0.00956)
x, y = (0.702, 0.139), fcn = 0.000413, grad = (0.00136, 0.00927)
x, y = (0.705, 0.157), fcn = 0.00058, grad = (0.00183, 0.0089)
x, y = (0.708, 0.175), fcn = 0.000734, grad = (0.0022, 0.00848)
x, y = (0.713, 0.192), fcn = 0.000875, grad = (0.00245, 0.008)
x, y = (0.718, 0.208), fcn = 0.001, grad = (0.00259, 0.00747)
x, y = (0.723, 0.223), fcn = 0.00111, grad = (0.00263, 0.00692)
x, y = (0.728, 0.237), fcn = 0.00121, grad = (0.0026, 0.00636)
x, y = (0.733, 0.249), fcn = 0.00129, grad = (0.0025, 0.0058)
x, y = (0.738, 0.261), fcn = 0.00135, grad = (0.00236, 0.00525)
x, y = (0.743, 0.272), fcn = 0.0014, grad = (0.00219, 0.00473)
x, y = (0.747, 0.281), fcn = 0.00145, grad = (0.00202, 0.00425)
x, y = (0.751, 0.29), fcn = 0.00148, grad = (0.00184, 0.0038)
x, y = (0.755, 0.297), fcn = 0.0015, grad = (0.00166, 0.00338)
x, y = (0.758, 0.304), fcn = 0.00152, grad = (0.0015, 0.00301)
x, y = (0.761, 0.31), fcn = 0.00154, grad = (0.00134, 0.00267)
x, y = (0.764, 0.315), fcn = 0.00155, grad = (0.0012, 0.00237)
x, y = (0.766, 0.32), fcn = 0.00156, grad = (0.00107, 0.0021)
x, y = (0.769, 0.324), fcn = 0.00156, grad = (0.000951, 0.00186)

How do we find the value of the gradient at the current position?

1. Analytically – find the expression for the derivative(s) and evaluate them
   - not always possible
   - the fastest way

2. Secant method – evaluate the function at two points some (small) distance apart and approximate the point derivative by the slope
   - Twice as many function evaluations
   - What is the best increment size?

Two ways of evaluating the derivative – some example code in Java

```java
private double f(double x) {
  return (3*Math.pow(x,1.3) - 0.38*Math.pow(4*x-0.11, 2));
        }

private double fprime(double x) {         // by direct evaluation
    return (3.9*Math.pow(x, 0.3) - 0.38*(32*x-0.88));
        }

private double dfdx(double x) {
    final double xstep = 0.01;            // by secant method
    return ((f(x)-f(x-xstep))/xstep);
}
```

Many optimization methods exist – usually based on the function gradient in some way

- Gradient
- Adaptive gradient
- Conjugate gradient (see next slide)
- Stochastic hill-climbing uses probability distributions to choose one of several uphill moves
- Simulated annealing – see https://en.wikipedia.org/wiki/Simulated_annealing
- Random-restart hill-climbing – start at some random position and find the maxima; choose many other starting points and select the "best" maxima found
  - this is a very common method; probably my go-to method for a serious problem

The conjugate gradient method finds a set of orthogonal directions that will take us to the maximum

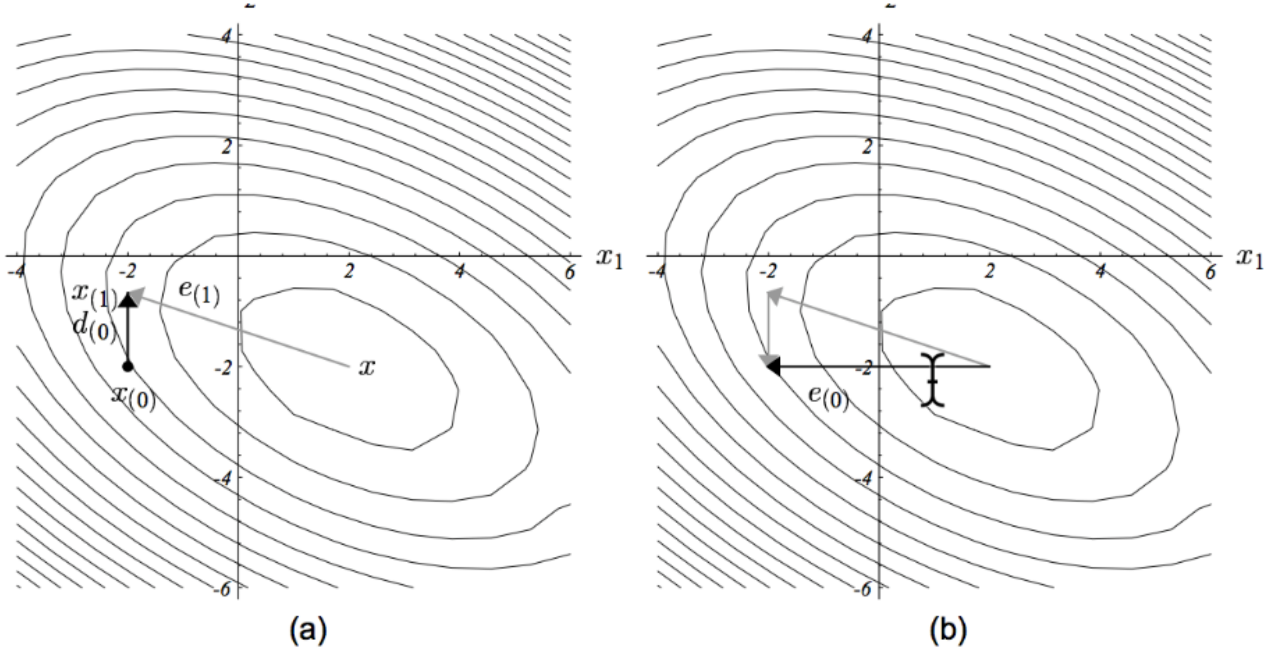
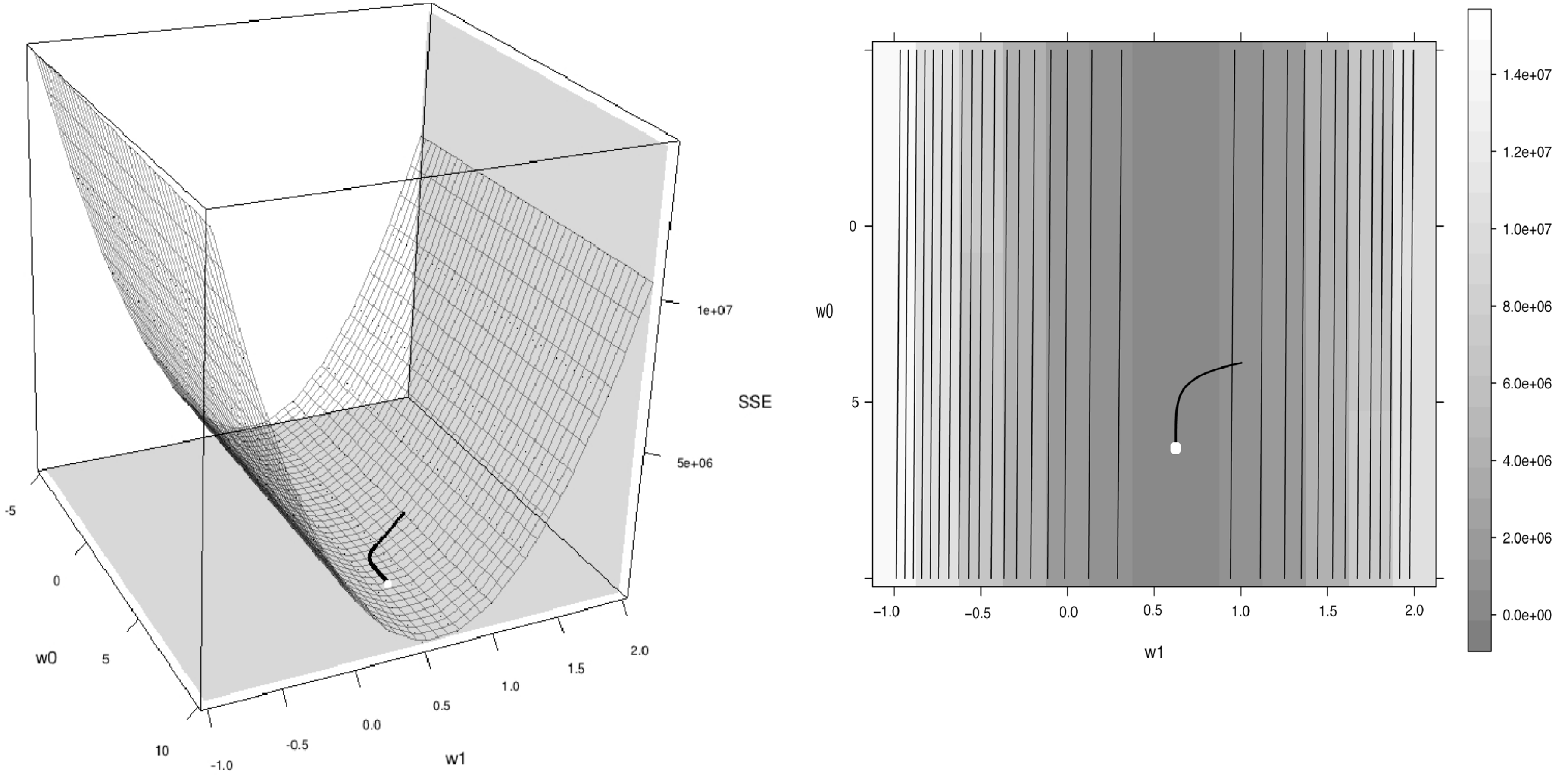


Figure 23: The method of Conjugate Directions converges in n steps. (a) The first step is taken along some direction $d_{(0)}$. The minimum point $x_{(1)}$ is chosen by the constraint that $e_{(1)}$ must be A-orthogonal to $d_{(0)}$. (b) The initial error $e_{(0)}$ can be expressed as a sum of A-orthogonal components (gray arrows). Each step of Conjugate Directions eliminates one of these components.

- from Jonathan Shewchuk, An Introduction to the Conjugate Gradient Method Without the Agonizing Pain,, https://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf

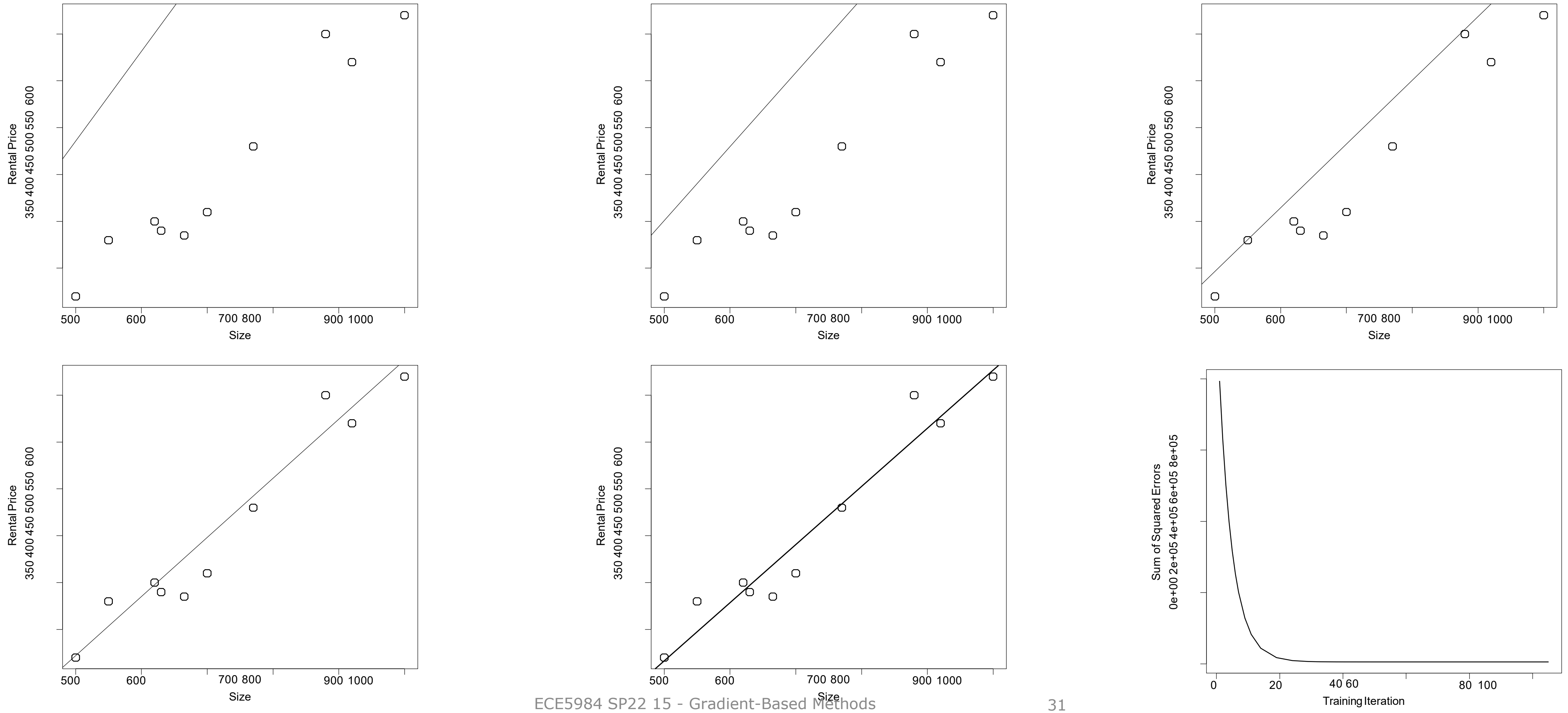
GRADIENT DESCENT FOR REGRESSION

The journey across the error surface of the gradient descent algorithm when training the univariate version of the model - involving only SIZE and RENTAL PRICE.



The successive univariate linear regression models developed during the gradient descent process - the final panel shows the sum of squared error values generated during the gradient descent process

VIRGINIA TECH
ece BRADLEY DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING



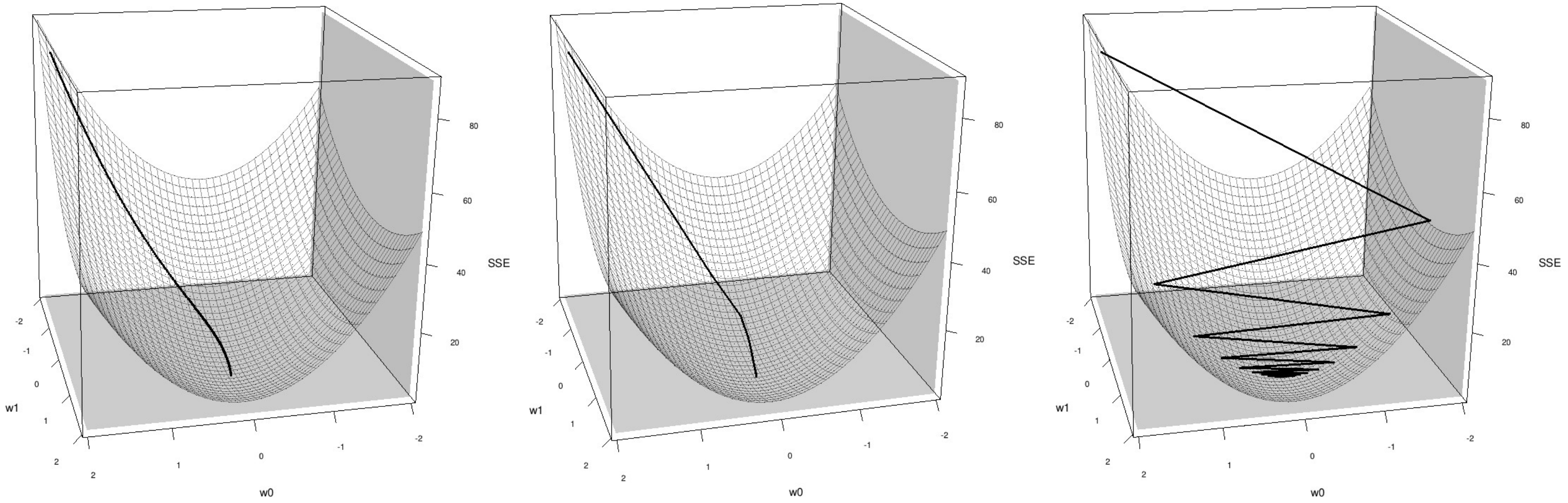
The gradient descent algorithm for training multivariate linear regression models

- Require: set of training instances $\mathcal{D}$
- Require: a learning rate α that controls how quickly the algorithm converges
- Require: a function, errorDelta, that determines the direction in which to adjust a given weight, $\mathbf{w}[j]$, so as to move down the slope of an error surface determined by the dataset, $\mathcal{D}$
- Require: a convergence criterion that indicates that the algorithm has completed

1. Set $\mathbf{w} \leftarrow random\ starting\ point$
2. repeat
    1. for each $\mathbf{w}[j]$ in $\mathbf{w}$ do
        1. Set $\mathbf{w}[j] \leftarrow \mathbf{w}[j] + \alpha \cdot errorDelta(\mathcal{D}, \mathbf{w}[j])$
    2. end for
3. until convergence occurs

The learning rate α determines the size of the adjustment made to each weight at each step in the process

- Unfortunately, choosing learning rates is not a well defined science.
- Most practitioners use rules of thumb and trial and error.



Plots of the learning process on the office rentals prediction problem for different learning rates: (a) a very small learning rate (0.002), (b) a medium learning rate (0.08) and (c) a very large learning rate (0.18).

Some heuristics for multivariate linear regression

- A typical range for learning rates is [0.00001, 10]
- Based on empirical evidence, choosing random initial weights uniformly from the range $[-0.2, 0.2]$ tends to work well in many cases

EXAMPLE

We are now in a position to build a linear regression model that uses all of the continuous descriptive features in the office rentals dataset

The general structure of the model is:

$$RentalPrice \cong -0.1513 + 0.6270 \cdot SIZE - 0.1781 \cdot FLOOR + 0.1714 \cdot BBRATE$$

- For this example let's assume that:

$$\alpha \;=\; 0.00000002$$

- Initial weights (assigned randomly):

$$w[0]\text{: } -0.146, \qquad w[1]\text{: } 0.185, \qquad w[2]\text{: } -0.044, \qquad w[3]\text{: } 0.119$$

ITERATION 1

Rental ID	Price	Pred.	Error	Squared Error	errorDelta(D, w[i]) w[0]	w[1]	w[2]	w[3]
1	320	93.26	226.74	51411.08	226.74	113370.05	906.96	1813.92
2	380	107.41	272.59	74307.70	272.59	149926.92	1908.16	13629.72
3	400	115.15	284.85	81138.96	284.85	176606.39	2563.64	1993.94
4	390	119.21	270.79	73327.67	270.79	170598.22	1353.95	6498.98
5	385	134.64	250.36	62682.22	250.36	166492.17	2002.91	25036.42
6	410	130.31	279.69	78226.32	279.69	195782.78	1118.76	2237.52
7	480	142.89	337.11	113639.88	337.11	259570.96	3371.05	2359.74
8	600	168.32	431.68	186348.45	431.68	379879.24	5180.17	21584.05
9	570	170.63	399.37	159499.37	399.37	367423.83	5591.23	3194.99
10	620	187.58	432.42	186989.95	432.42	432423.35	3891.81	10378.16
			Sum Sum of squared errors (Sum/2)	1067571.59 533785.80	3185.61	2412073.90	27888.65	88727.43

$$\mathbf{w}[j] \leftarrow \mathbf{w}[j] + \alpha \underbrace{\sum_{i=1}^{n} \left(\left(t_i - \mathbb{M}_{\mathbf{w}}(\mathbf{d}_i) \right) \times d_i[j] \right)}_{errorDelta(\mathcal{D}, \mathbf{w}[j])}$$

Initial Weights

w[0]:	-0.146	**w**[1]:	0.185	**w**[2]:	-0.044	**w**[3]:	0.119

Example

$$\mathbf{w}[1] \leftarrow 0.185 + 0.00000002 \times 2,412,074 = 0.23324148$$

New Weights (Iteration 1)

w[0]:	-0.146	**w**[1]:	0.233	**w**[2]:	-0.043	**w**[3]:	0.121

Rental ID	Price	Pred.	Error	Squared Error	errorDelta(D, w[i]) w[0]	w[1]	w[2]	w[3]
1	320	117.40	202.60	41047.92	202.60	101301.44	810.41	1620.82
2	380	134.03	245.97	60500.69	245.97	135282.89	1721.78	12298.44
3	400	145.08	254.92	64985.12	254.92	158051.51	2294.30	1784.45
4	390	149.65	240.35	57769.68	240.35	151422.55	1201.77	5768.48
5	385	166.90	218.10	47568.31	218.10	145037.57	1744.81	21810.16
6	410	164.10	245.90	60468.86	245.90	172132.91	983.62	1967.23
7	480	180.06	299.94	89964.69	299.94	230954.68	2999.41	2099.59
8	600	210.87	389.13	151424.47	389.13	342437.01	4669.60	19456.65
9	570	215.03	354.97	126003.34	354.97	326571.94	4969.57	2839.76
10	620	187.58	432.42	186989.95	432.42	432423.35	3891.81	10378.16
			Sum	886723.04	2884.32	2195615.84	25287.08	80023.74
			Sum of squared errors (Sum/2)	443361.52				

ITERATION 2

$$\mathbf{w}[j] \leftarrow \mathbf{w}[j] + \alpha \underbrace{\sum_{i=1}^{n} \left((t_i - \mathbb{M}_{\mathbf{w}}(\mathbf{d}_i)) \times d_i[j] \right)}_{errorDelta(\mathcal{D}, \mathbf{w}[j])}$$

Initial Weights (Iteration 2)

w[0]:	-0.146	**w**[1]:	0.233	**w**[2]:	-0.043	**w**[3]:	0.121

Exercise

$$\mathbf{w}[1] \leftarrow -0.233 + 0.00000002 \times 2195616.08 = 0.27691232$$

New Weights (Iteration 2)

w[0]:	-0.145	**w**[1]:	0.277	**w**[2]:	-0.043	**w**[3]:	0.123

Iteration continue until completion…

The algorithm then keeps iteratively applying the weight update rule until it converges on a stable set of weights beyond which little improvement in model accuracy is possible.

Determining this stopping point is not always straightforward!

After 100 iterations, the values of the weights – *according to the textbook* - are:

- $w[0] = -0.1513$,
- $w[1] = 0.6270$,
- $w[2] = -0.1781$
- $w[3] = 0.0714$

which results in a residual sum of squared errors value of 2913.5

```
ECE5984 SP20 Multivariate Linear Regression
Created on Thu Feb 20 17:41:33 2020

@author: crjones4
from sklearn import preprocessing as preproc
from sklearn import linear_model as linmod
import pandas as pd
import numpy as np

pathName = "C:\\Data\\"
fileName = "DublinRental.xlsx"      # read from Excel file
targetName = "PRICE"
IDName = "ID"
catName = "ENERGY"
dataFrame = pd.read_excel(pathName + fileName, sheet_name='train')
trainX = dataFrame.drop([IDName, catName, targetName], axis=1).to_numpy()
trainY = dataFrame[targetName].to_numpy()
mlr = linmod.LinearRegression()     # creates the regressor object
mlr.fit(trainX, trainY)
print("R2 is %f" % mlr.score(trainX, trainY))
print("W = ", mlr.intercept_, mlr.coef_)
query = np.array([[600,6,20]])
print("prediction:", query, mlr.predict(query))
```

```
R2 is 0.955209
('W = ', 19.561558897449345, array([ 0.54873985,
4.96354677, -0.06209515]))
('prediction:', array([[600,   6,  20]]), array([377.34484437]))
```

VIRGINIA TECH.
BRADLEY DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

So why does the book's resulting weight vector differ from the one that I have calculated?

- $\boldsymbol{w}_{book} = [-0.1513, 0.6270, -0.1781, 0.0714]$
  - $RMSE \; = \; 2913.5$
  - $M_{book}([600, 6, 20]) = 378.408$

- $\boldsymbol{w}_{code} = [19.56155, 0.54873, 4.96354, -0.0062]$
  - $RMSE = 448.456$
  - $M_{code}([600, 6, 20]) = 378.456$

- I think that the book's authors either:
  - stopped early in training
  - used a different training process
  - made a mistake?

Today’s Objectives

Gradient-based methods

- Linear Regression
- Multivariate Linear Regression
- Function Optimization
- Gradient Descent
- An Example