201700949 설재혁

In [11]:

```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

from collections import Counter
#from linear_algebra import distance, vector_subtract, scalar_multiply

#original version
def squared_distance(v, w):
    return sum_of_squares(vector_subtract(v, w))

def distance(v, w):
   return math.sqrt(squared_distance(v, w))

def vector_subtract(v, w):
    """subtracts two vectors componentwise"""
    return [v_i - w_i for v_i, w_i in zip(v,w)]

def scalar_multiply(c, v):
    return [c * v_i for v_i in v]

def vector_add(v, w):
    """adds two vectors componentwise"""
    return [v_i + w_i for v_i, w_i in zip(v,w)]

def vector_sum(vectors):
    return reduce(vector_add, vectors)

def vector_mean(vectors):
    """compute the vector whose i-th element is the mean of the
    i-th elements of the input vectors"""
    n = len(vectors)
    return scalar_multiply(1/n, vector_sum(vectors))
```

경사 하강법의 숨은 의미

경사 하강법의 핵심은

- 미분한 결과가 양수면 찾고자 하는 대상의 값을 빼줘야 최소점으로 다가가고
- 미분한 결과가 음수면 찾고자 하는 대상의 값을 더해줘야 최소점으로 다가간다.

In [12]:

```
from functools import reduce
import math, random

import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline

def sum_of_squares(v):
    """computes the sum of squared elements in v"""
    return sum(v_i ** 2 for v_i in v)

vector = [i for i in range(10)]
sum_of_squares(vector)

np.sum(np.square(vector))
```

Out[12]:

```
285
```

Out[12]:

```
285
```

함수 변화율(미분값) 근사법

1. f가 단변수 함수인 경우

In [13]:

```
def difference_quotient(f, x, h):
    return (f(x + h) - f(x)) / h

def square(x: float) -> float:
    return x * x

def derivative(x: float) -> float:
    return 2 * x

xs = range(-10,11)
actuals = [derivative(x) for x in xs]
estimates = [difference_quotient(square, x, h=0.0001) for x in xs]

# 두 계산식의 결괏값이 거의 비슷함을 보여 주기 위한 그래프
# plot to show they're basically the same
import matplotlib.pyplot as plt
plt.title("actual Derivatives vs. Estimates")
plt.plot(xs, actuals, 'rx', label='Actual')      # red  x
plt.plot(xs, estimates, 'b+', label='Estimate')  # blue +
plt.legend(loc=9)
plt.show()                                       # purple *, hopefully
```

Out[13]:

```
Text(0.5, 1.0, 'actual Derivatives vs. Estimates')
```

Out[13]:

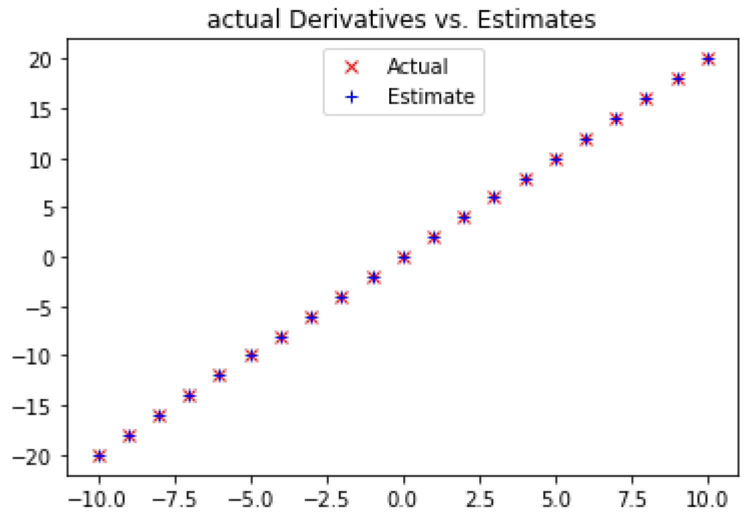
```
[<matplotlib.lines.Line2D at 0x7fe79ac5c790>]
```

Out[13]:

```
[<matplotlib.lines.Line2D at 0x7fe79ac5c940>]
```

Out[13]:

```
<matplotlib.legend.Legend at 0x7fe79ab7ab20>
```



그래디언트 적용하기

경사 하강법을 이용해서 3차원 벡터의 최소값을 구해본다

- 임의의 시작점을 잡고 그래디언트가 아주 작아질 때까지 경사의 반대 방향으로 조금씩 이동하면 된다.

In [14]:

```
def step(v, direction, step_size):
    """move step_size in the direction from v"""
    return [v_i + step_size * direction_i
            for v_i, direction_i in zip(v, direction)]

def sum_of_squares_gradient(v):
    return [2 * v_i for v_i in v]

# 임의의 시작점을 선택
v = [random.randint(-10,10) for i in range(3)]
tolerance = 0.0000001    # 아주 작은 e값

while True:
    #print v, sum_of_squares(v)
    gradient = sum_of_squares_gradient(v)   # compute the gradient at v
    next_v = step(v, gradient, -0.0001)         # take a negative gradient step
    if distance(next_v, v) < tolerance:         # stop if we're converging
        break
    v = next_v                                      # continue if we're not
    #print(v)

print("minimum v", v)
print("minimum value", sum_of_squares(v))
```

```
minimum v [0.0, -0.0004949540745547053, 7.070772493638706e-05]
minimum value 2.4997911828398457e-07
```

적절한 이동 거리 정하기

얼마만큼 이동해야 하는지(=학습률) 정하는 일반적인 옵션

1. 이동 거리를 고정
2. 시간에 따라 이동 거리를 점차 줄임
3. 이동할 때마다 목적 함수를 최소화 하는 이동 거리로 정함(가장 이상적이나 계산 비용이 큼)

경사 하강법으로 모델 학습

In [15]:

```
# Using gradient descent to fit models

def gradient_step(v, gradient, step_size):
    """Moves `step_size` in the `gradient` direction from `v`"""
    assert len(v) == len(gradient)
    step = scalar_multiply(step_size, gradient)
    return vector_add(v, step)

# x ranges from -50 to 49, y is always 20 * x + 5
inputs = [(x, 20 * x + 5) for x in range(-50, 50)]
print(inputs)

#def linear_gradient(x: float, y: float, theta: Vector) -> Vector:
def linear_gradient(x, y, theta):
    slope, intercept = theta
    predicted = slope * x + intercept
    error = (predicted - y)
    squared_error = error ** 2
    grad = [2 * error * x, 2 * error]
    return grad
```

```
[(-50, -995), (-49, -975), (-48, -955), (-47, -935), (-46, -915), (-4
5, -895), (-44, -875), (-43, -855), (-42, -835), (-41, -815), (-40, -7
95), (-39, -775), (-38, -755), (-37, -735), (-36, -715), (-35, -695),
(-34, -675), (-33, -655), (-32, -635), (-31, -615), (-30, -595), (-29,
-575), (-28, -555), (-27, -535), (-26, -515), (-25, -495), (-24, -47
5), (-23, -455), (-22, -435), (-21, -415), (-20, -395), (-19, -375),
(-18, -355), (-17, -335), (-16, -315), (-15, -295), (-14, -275), (-13,
-255), (-12, -235), (-11, -215), (-10, -195), (-9, -175), (-8, -155),
(-7, -135), (-6, -115), (-5, -95), (-4, -75), (-3, -55), (-2, -35), (-
1, -15), (0, 5), (1, 25), (2, 45), (3, 65), (4, 85), (5, 105), (6, 12
5), (7, 145), (8, 165), (9, 185), (10, 205), (11, 225), (12, 245), (1
3, 265), (14, 285), (15, 305), (16, 325), (17, 345), (18, 365), (19, 3
85), (20, 405), (21, 425), (22, 445), (23, 465), (24, 485), (25, 505),
(26, 525), (27, 545), (28, 565), (29, 585), (30, 605), (31, 625), (32,
645), (33, 665), (34, 685), (35, 705), (36, 725), (37, 745), (38, 76
5), (39, 785), (40, 805), (41, 825), (42, 845), (43, 865), (44, 885),
(45, 905), (46, 925), (47, 945), (48, 965), (49, 985)]
```

In [16]:

```
#from linear_algebra import vector_mean

# Start with random values for slope and intercept
theta = [random.uniform(-1, 1), random.uniform(-1, 1)]

learning_rate = .001

for epoch in range(5000):
    # Computer the mean of the gradients
    grad = vector_mean([linear_gradient(x, y, theta) for x, y in inputs])
    # Take a step in that direction
    theta = gradient_step(theta, grad, -learning_rate)
    print(epoch, theta)
slope, intercept = theta
assert 19.9 < slope < 20.1, "slope should be about 20"
assert 4.9 < intercept < 5.1, "intercept should be about 5"
```

```
0 [33.333952857709576, 0.3880890185679627]
1 [11.10164153292628, 0.4106467933885363]
2 [25.93061574433156, 0.41092714133468555]
3 [16.03969022567218, 0.4260359027963477]
4 [22.636952655379453, 0.43122352121642715]
5 [18.23658380238312, 0.4429980268293737]
6 [21.17164160183729, 0.45034861457809805]
7 [19.213965400189107, 0.46061955895077916]
8 [20.519745697632818, 0.4689122852330668]
9 [19.648798531964143, 0.4784942063602335]
10 [20.229729873386276, 0.4871860164794772]
11 [19.842257360467833, 0.49644137431990454]
12 [20.100710781942276, 0.5052907489317325]
13 [19.928331199193433, 0.5143808782158114]
14 [20.043317471016195, 0.5232804476585732]
15 [19.966630527279857, 0.5322772042342723]
16 [20.01778971550857, 0.5411792803530836]
17 [19.983675439036137, 0.550114711507886]
18 [20.006438596874403, 0.5589981575239065]
```

201700949 설재혁