Convergence rate of gradient descent

(a) M smooth

$$\Rightarrow \quad f(x_{k+1}) \le f(x_k) + \langle \nabla f(x_k), x_{k+1} - x_k \rangle + \frac{M}{2} \|x_{k+1} - x_k\|^2$$

substitute that $x_{k+1} - x_k = -\alpha \nabla f(x_k)$

$$\Rightarrow \quad (\alpha - \frac{M}{2}\alpha^2) \|\nabla f(x_k)\|^2 \le f(x_k) - f(x_{k+1})$$

notice that $\alpha \le \frac{1}{M} \Rightarrow \alpha - \frac{M}{2}\alpha^2 \ge \alpha - \frac{1}{2}\frac{1}{\alpha}\alpha^2 = \frac{1}{2}\alpha$

$$\Rightarrow \quad \frac{\alpha}{2} \|\nabla f(x_k)\|^2 \le f(x_k) - f(x_{k+1})$$

$$\Rightarrow \quad \|\nabla f(x_k)\|^2 \le \frac{2}{\alpha}\left(f(x_k) - f(x_{k+1})\right)$$

(b) $$\sum_{l=0}^{k-1} \|\nabla f(x_l)\|^2 \le \frac{2}{\alpha} \sum_{l=0}^{k-1} \left(f(x_0) - f(x_{l+1})\right)$$

$$LHS \ge k \min_{l=0,\cdots,k-1} \|\nabla f(x_l)\|^2 = k \left(\min_{l=0,\cdots,k-1} \|\nabla f(x_l)\|\right)^2$$

$$RHS = \frac{2}{\alpha}\left(f(x^0) - f(x^k)\right) \le \frac{2}{\alpha}\left(f(x^0) - z^*\right)$$

$$\Rightarrow \quad k \left(\min_{l=0,\cdots,k-1} \|\nabla f(x_l)\|\right)^2 \le \frac{2}{\alpha}\left(f(x^0) - z^*\right)$$

$$\Rightarrow \quad \min_{l=0,\cdots,k-1} \|\nabla f(x_l)\| \le \sqrt{\frac{2}{\alpha k}\left(f(x^0) - z^*\right)}$$

c) i.e. $\sqrt{\frac{2}{\alpha k}\left(f(x^0) - z^*\right)} = \varepsilon$

$$\Rightarrow \quad k = \frac{1}{\varepsilon^2} \frac{2\left(f(x^0) - z^*\right)}{\alpha}$$

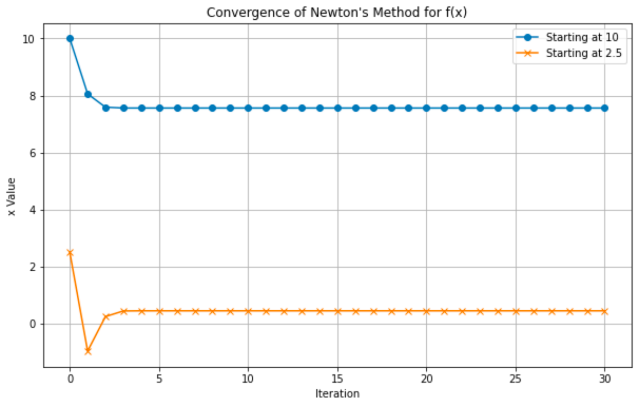
because $\lim_{k \to \infty} \frac{\varepsilon_{k+1}}{\varepsilon_k} = \lim_{k \to \infty} \sqrt{\frac{k}{k+1}} = 1$

it is sub linear convergence

The convergence rate is the same as $f(x^k) - z^*$, which is also sub-linear

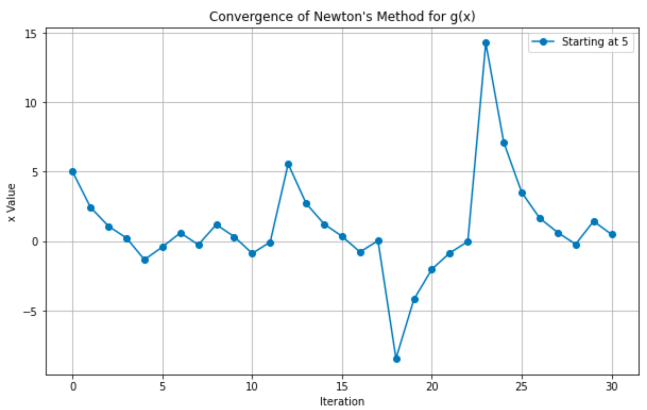
Newton's method

a



when start at 10, it converges at a local minimal,
when start at 2.5, it converges at a local maximal.
This shows that Newton's method is attractive
to a $\nabla f(x) = 0$ point.

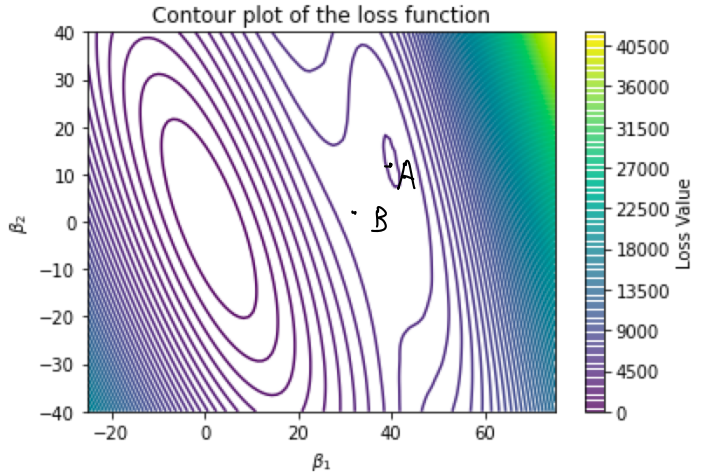
b.



it oscillates around $x=0$, because $g(x)$ is strictly monotonically increasing, $g(x)$ has no $g'(x) = 3x^2 + 2$ has no $g'(x) = 0$ point. And $0 = \arg\min |g'(x)|$, this suggest that when a landscape has no $\nabla g(x) = 0$ point, then it will oscillate around a point satisfying $\min_x \|\nabla g(x)\|$

Gradient descent v.s. stochastic gradient descent

a) the Contour plot:



this is a non-convex shape, there are local minimum (A), saddle point (B), and the slope varies largely in different region.

implications: 1. because slope varies, we'd better use adaptive learning rate

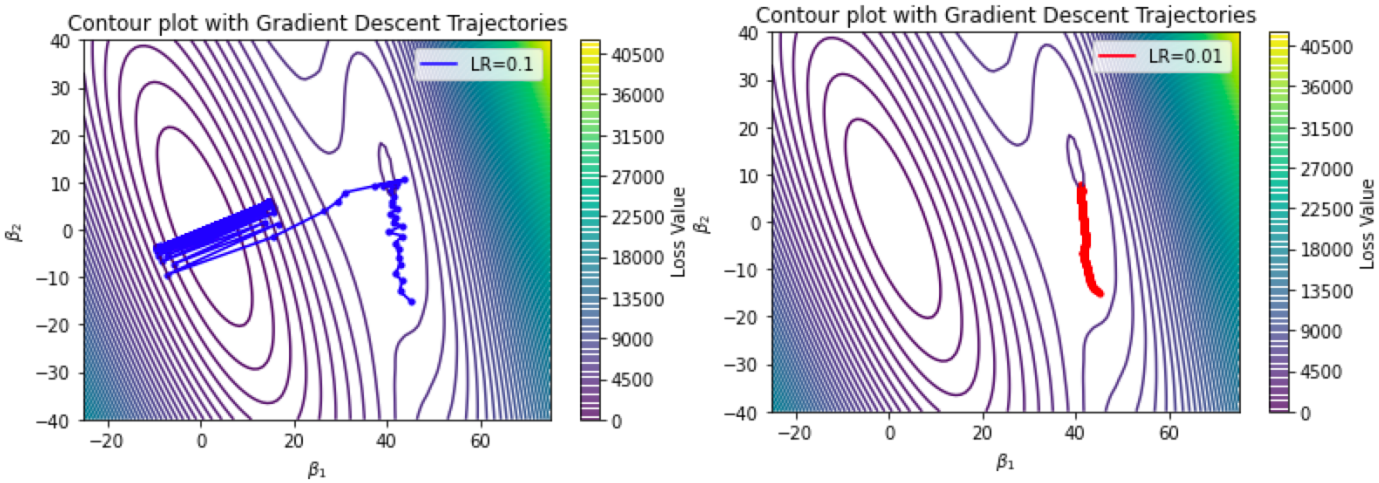
2. because there are local minimal, initial point is sensitive

3. stochastic method might be helpful for escape local minimum and saddle points.

b) key codes:

```python
def gradient(beta1, beta2, XA, yA, XB, yB):
    # Compute residuals
    res_A = yA - beta1 * XA[:, 0] - beta2 * XA[:, 1]
    res_B = yB - beta1 * XB[:, 0] - beta2 * XB[:, 1]

    # Calculate squared residuals
    loss_A = res_A ** 2
    loss_B = res_B ** 2

    # Identify which dataset contributes to the gradient for each sample
    mask_A = loss_A <= loss_B
    mask_B = np.logical_not(mask_A)

    # Calculate gradients
    grad_beta1 = -np.sum(res_A[mask_A] * XA[mask_A, 0]) - np.sum(res_B[mask_B] * XB[mask_B, 0])
    grad_beta2 = -np.sum(res_A[mask_A] * XA[mask_A, 1]) - np.sum(res_B[mask_B] * XB[mask_B, 1])

    return np.array([grad_beta1 / len(yA), grad_beta2 / len(yB)])

# Gradient Descent function
def gradient_descent(XA, yA, XB, yB, learning_rate, iterations):
    beta = np.array([45, -15])  # starting point
    trajectory = [beta]

    for _ in range(iterations):
        grad = gradient(beta[0], beta[1], XA, yA, XB, yB)
        print(grad)
        beta = beta - learning_rate * grad
        trajectory.append(beta)

    return np.array(trajectory)
```

results:



for learning rate = 0.1, it shows zig zag behavior. It was attracted by local minimum first, and the escape it, go through saddle point and fall into global minimum. But at around global minimum, it shows large oscillation and cannot converge.

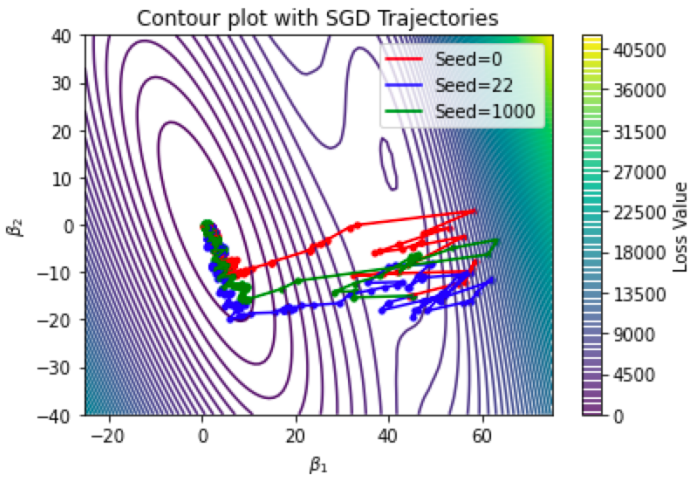
for learning rate = 0.01, it slowly converge to the local minimum.

This shows the importance of tuning learning rate. Too large or too small would both lead to problems.

c. key codes:

```python
def stochastic_gradient(beta1, beta2, x1, x2, yA, x1B, x2B, yB):
    # Compute residuals
    res_A = yA - beta1 * x1 - beta2 * x2
    res_B = yB - beta1 * x1B - beta2 * x2B

    # Calculate squared residuals
    loss_A = res_A ** 2
    loss_B = res_B ** 2

    # Decide which dataset contributes to the gradient
    if loss_A <= loss_B:
        return np.array([-res_A * x1, -res_A * x2])
    else:
        return np.array([-res_B * x1B, -res_B * x2B])

def stochastic_gradient_descent(XA, yA, XB, yB, learning_rate, iterations, seed=None):
    np.random.seed(seed)
    beta = np.array([45, -15])  # starting point
    trajectory = [beta]

    for _ in range(iterations):
        # Randomly select a sample
        idx = np.random.randint(0, len(yA))
        x1, x2 = XA[idx, 0], XA[idx, 1]
        x1B, x2B = XB[idx, 0], XB[idx, 1]
        grad = stochastic_gradient(beta[0], beta[1], x1, x2, yA[idx], x1B, x2B, yB[idx])

        beta = beta - learning_rate * grad
        trajectory.append(beta)

    return np.array(trajectory)
```

result:



the stochastic nature helps escaping local minimum. Though different random seed leads to different path, they all convergent to the same minimum point.