**Contoh Soal:**

Sama seperti sebelumnya, kita ingin memprediksi keuntungan harian tempat wisata berdasarkan faktor-faktor: harga tiket ((h)), jumlah mainan anak ((m)), luas wahana air ((w)), indeks lokasi ((l)), dan ketersediaan transportasi umum ((t)).

**Persamaan Keuntungan (Contoh yang Lebih Sederhana untuk Demonstrasi Gradien):**

Untuk mempermudah perhitungan gradien, kita akan menggunakan bentuk persamaan keuntungan yang lebih sederhana, meskipun dalam praktiknya bisa jauh lebih kompleks:

K(h,m,w,l,t)=P(h,m,w,l,t)−C(m,w)

Di mana:

* Pendapatan Harian ((P)): P(h,m,w,l,t)=(k1​⋅h+k2​⋅m​+k3​⋅ln(w+1)+k4​⋅l+k5​⋅t)⋅(N0​−α⋅h+β⋅m+γ⋅w+δ⋅l+ϵ⋅t) Di sini, bagian pertama dalam kurung merepresentasikan pendapatan rata-rata per pengunjung yang dipengaruhi oleh faktor-faktor, dan bagian kedua adalah perkiraan jumlah pengunjung.
* Biaya Operasional Harian ((C)): C(m,w)=c1​⋅m+c2​⋅w

**Tujuan:**

Kita ingin menemukan nilai (h, m, w, l, t) yang memaksimalkan (K) menggunakan metode berbasis gradien.

**Penggunaan Gradien:**

Metode gradient descent (atau ascent untuk maksimasi) adalah algoritma optimasi iteratif yang bergerak menuju minimum (atau maksimum) suatu fungsi dengan mengambil langkah proporsional terhadap negatif (atau positif) dari gradien pada titik saat ini. Gradien adalah vektor dari turunan parsial fungsi terhadap masing-masing variabel.

Dalam kasus ini, kita ingin memaksimalkan (K), jadi kita akan bergerak ke arah gradien positif:

∇K=(∂h∂K​,∂m∂K​,∂w∂K​,∂l∂K​,∂t∂K​)

Setiap langkah iterasi akan memperbarui nilai variabel sebagai berikut:

xi+1​=xi​+η⋅∇K(xi​)

di mana (x) adalah vektor ((h, m, w, l, t)) dan (\eta) adalah *learning rate* (langkah pembelajaran).

**Kode Python untuk Menyelesaikan Kasus dengan Gradien Ascent:**

Python

import numpy as np

import matplotlib.pyplot as plt

# Definisi fungsi pendapatan

def pendapatan(params, k1=1.5, k2=2, k3=0.8, k4=10, k5=20, n0=1000, alpha=5, beta=3, gamma=2, delta=150, epsilon=40):

h, m, w, l, t = params

pendapatan\_per\_pengunjung = k1 \* h + k2 \* np.sqrt(m) + k3 \* np.log(w + 1) + k4 \* l + k5 \* t

jumlah\_pengunjung = n0 - alpha \* h + beta \* m + gamma \* w + delta \* l + epsilon \* t

return pendapatan\_per\_pengunjung \* jumlah\_pengunjung

# Definisi fungsi biaya

def biaya(m, w, c1=8, c2=4):

return c1 \* m + c2 \* w

# Definisi fungsi keuntungan

def keuntungan(params):

h, m, w, l, t = params

if h < 0 or m < 0 or w < 0 or l < 0 or l > 1 or t not in [0, 1]:

return -np.inf # Mengembalikan nilai negatif tak hingga untuk parameter tidak valid

return pendapatan(params) - biaya(m, w)

# Definisi gradien fungsi keuntungan (turunan parsial)

def gradien\_keuntungan(params, delta=1e-6):

h, m, w, l, t = params

grad = np.zeros\_like(params, dtype=float)

# Turunan parsial terhadap h

params\_h\_plus = np.array([h + delta, m, w, l, t])

params\_h\_minus = np.array([h - delta, m, w, l, t])

grad[0] = (keuntungan(params\_h\_plus) - keuntungan(params\_h\_minus)) / (2 \* delta)

# Turunan parsial terhadap m

params\_m\_plus = np.array([h, m + delta, w, l, t])

params\_m\_minus = np.array([h, m - delta, w, l, t])

grad[1] = (keuntungan(params\_m\_plus) - keuntungan(params\_m\_minus)) / (2 \* delta)

# Turunan parsial terhadap w

params\_w\_plus = np.array([h, m, w + delta, l, t])

params\_w\_minus = np.array([h, m, w - delta, l, t])

grad[2] = (keuntungan(params\_w\_plus) - keuntungan(params\_w\_minus)) / (2 \* delta)

# Turunan parsial terhadap l

params\_l\_plus = np.array([h, m, w, l + delta, t])

params\_l\_minus = np.array([h, m, w, l - delta, t])

grad[3] = (keuntungan(params\_l\_plus) - keuntungan(params\_l\_minus)) / (2 \* delta)

# Turunan parsial terhadap t

params\_t\_plus = np.array([h, m, w, l, t + delta if t == 0 else t - delta])

params\_t\_minus = np.array([h, m, w, l, t - delta if t == 1 else t + delta])

grad[4] = (keuntungan(params\_t\_plus) - keuntungan(params\_t\_minus)) / (2 \* delta)

return grad

# Inisialisasi parameter

params = np.array([50.0, 10.0, 20.0, 0.7, 1.0])

learning\_rate = 0.001

n\_iterations = 1000

keuntungan\_history = []

params\_history = []

# Gradient Ascent

for i in range(n\_iterations):

grad = gradien\_keuntungan(params)

params = params + learning\_rate \* grad

# Memastikan batasan variabel

params[0] = np.clip(params[0], 10, 100) # Batas harga tiket

params[1] = np.clip(params[1], 0, 50) # Batas jumlah mainan

params[2] = np.clip(params[2], 0, 100) # Batas luas wahana air

params[3] = np.clip(params[3], 0, 1) # Batas indeks lokasi

params[4] = np.round(np.clip(params[4], 0, 1)) # Batas transportasi (0 atau 1)

keuntungan\_current = keuntungan(params)

keuntungan\_history.append(keuntungan\_current)

params\_history.append(params.copy())

# Hasil optimal

h\_optimal, m\_optimal, w\_optimal, l\_optimal, t\_optimal = params

keuntungan\_optimal = keuntungan(params)

print("Hasil Optimal (dengan Gradient Ascent):")

print(f"Harga Tiket Optimal: ${h\_optimal:.2f}")

print(f"Jumlah Mainan Anak Optimal: {int(m\_optimal)}")

print(f"Luas Wahana Air Optimal: {w\_optimal:.2f} unit")

print(f"Indeks Lokasi Optimal: {l\_optimal:.2f}")

print(f"Ketersediaan Transportasi Umum Optimal: {int(t\_optimal)}")

print(f"Keuntungan Harian Optimal: ${keuntungan\_optimal:.2f}")

# Visualisasi Konvergensi Keuntungan

plt.figure(figsize=(10, 6))

plt.plot(range(n\_iterations), keuntungan\_history)

plt.xlabel("Iterasi")

plt.ylabel("Keuntungan Harian ($)")

plt.title("Konvergensi Keuntungan selama Gradient Ascent")

plt.grid(True)

plt.show()

# Visualisasi Perubahan Parameter (Contoh Harga Tiket)

plt.figure(figsize=(10, 6))

harga\_tiket\_history = [p[0] for p in params\_history]

plt.plot(range(n\_iterations), harga\_tiket\_history)

plt.xlabel("Iterasi")

plt.ylabel("Harga Tiket ($)")

plt.title("Perubahan Harga Tiket selama Gradient Ascent")

plt.grid(True)

plt.show()

# ... (Anda dapat menambahkan visualisasi untuk parameter lainnya)

**Penjelasan Kode:**

1. **Definisi Fungsi:**
   * pendapatan(): Menghitung pendapatan harian berdasarkan parameter.
   * biaya(): Menghitung biaya operasional harian.
   * keuntungan(): Menghitung keuntungan harian.
2. **Definisi Gradien:**
   * gradien\_keuntungan(): Menghitung perkiraan gradien fungsi keuntungan secara numerik menggunakan metode selisih pusat (central difference). Ini mengestimasi turunan parsial terhadap setiap variabel.
3. **Inisialisasi:**
   * params: Nilai awal untuk variabel optimasi.
   * learning\_rate: Ukuran langkah yang diambil pada setiap iterasi. Nilai yang terlalu besar dapat menyebabkan osilasi, sedangkan nilai yang terlalu kecil dapat memperlambat konvergensi.
   * n\_iterations: Jumlah iterasi algoritma.
   * keuntungan\_history dan params\_history: List untuk menyimpan nilai keuntungan dan parameter pada setiap iterasi untuk visualisasi.
4. **Gradient Ascent:**
   * Loop utama melakukan iterasi sebanyak n\_iterations.
   * Pada setiap iterasi, gradien dihitung.
   * Nilai parameter diperbarui dengan bergerak ke arah gradien positif (untuk maksimasi).
   * Batasan untuk setiap variabel diterapkan setelah pembaruan.
5. **Hasil Optimal:**
   * Setelah iterasi selesai, nilai parameter terakhir dianggap sebagai nilai optimal yang ditemukan.
6. **Visualisasi:**
   * Grafik pertama menunjukkan bagaimana keuntungan berubah selama iterasi, yang idealnya akan meningkat dan kemudian mungkin mencapai plateau (konvergensi).
   * Grafik kedua menunjukkan bagaimana nilai salah satu parameter (dalam contoh ini, harga tiket) berubah selama iterasi. Anda dapat membuat grafik serupa untuk parameter lainnya.

**Penting untuk Diperhatikan:**

* **Gradien Numerik:** Metode gradien numerik adalah pendekatan perkiraan. Untuk fungsi yang lebih kompleks, perhitungan gradien analitik (menurunkan persamaan secara langsung) bisa lebih akurat dan efisien, tetapi mungkin lebih sulit diimplementasikan.
* **Learning Rate:** Pemilihan *learning rate* sangat penting. Nilai yang tepat dapat mempercepat konvergensi, sementara nilai yang tidak tepat dapat menyebabkan masalah.
* **Konvergensi:** Metode gradient ascent tidak menjamin untuk menemukan maksimum global, terutama untuk fungsi yang tidak cembung. Hasilnya dapat bergantung pada nilai awal parameter.
* **Kompleksitas Model:** Model keuntungan yang lebih realistis mungkin memiliki bentuk yang lebih rumit, sehingga perhitungan gradien dan optimasi bisa menjadi lebih menantang.