Nama : febi rahmadia putri

NIM :20220021

Mata Kuliah : Praktikum Desain Analisis & Algoritma

Laporan praktikum algoritma aproksimasi

1. implementasi strategi algoritma aproksimasi pada masalah penjadwalan tugas dengan bahasa pemrograman Python

```
import numpy as np

def scheduling_approx(tasks, deadlines, processing_times):
    n = len(tasks)
    order = []
    time_left = deadlines - rocessing_times

    while len(order) < n:
        ratios = time_left / processing_times
        i = np.argmax(ratios)
        order.append(tasks[i])
        time_left = np.delete(time_left, i)
        tasks = np.delete(tasks, i, axis=0)
        deadlines = np.delete(deadlines, i, axis=0)
        processing_times = np.delete(processing_times, i, axis=0)

    return order

tasks = np.array(['A', 'B', 'C', 'D', 'E'])[:, np.newaxis]
deadlines = np.array([7, 6, 8, 10, 11])[:, np.newaxis]
processing_times = np.array([3, 4, 2, 5, 6])[:, np.newaxis]

order = scheduling_approx(tasks, deadlines, processing_times)
print("Urutan tugas yang direkomendasikan:", order)
```

Script di atas mengimplementasikan pemecahan masalah pemrograman linier menggunakan algoritma simplex dengan menggunakan fungsi linprog dari modul scipy.optimize dalam Python.

Langkah-langkah utama yang dilakukan dalam script ini adalah sebagai berikut:

a. Impor modul numpy dan linprog dari scipy.optimize.

b. Bentuk matriks koefisien batasan A, vektor batasan b, dan koefisien fungsi tujuan c menggunakan array NumPy.
c. Gunakan fungsi linprog dengan argumen c sebagai koefisien fungsi tujuan, A_ub sebagai matriks koefisien batasan (dalam bentuk ≤), dan b_ub sebagai vektor batasan.
d. Hasil dari pemecahan masalah linier disimpan dalam variabel res.
e. Cetak nilai maksimum fungsi tujuan dengan -res.fun, di mana -res.fun adalah negasi dari nilai minimum yang dihasilkan oleh algoritma simplex.
f. Cetak nilai variabel x dengan res.x, yang merupakan solusi optimum yang ditemukan oleh algoritma.

Dengan menggunakan algoritma simplex, script ini mencari solusi optimum yang memaksimalkan fungsi tujuan dengan memenuhi batasan linier yang diberikan. Output dari script ini adalah nilai maksimum fungsi tujuan dan nilai variabel x yang merupakan solusi optimum dari masalah linier yang diberikan.

2. implementasi strategi algoritma aproksimasi pada masalah LP dengan bahasa pemrograman Python menggunakan algoritma simplex

```
import numpy as np
from scipy.optimize import linprog

# Membentuk sistem persamaan linier dengan numpy
A = np.array([[1, 1], [-1, 2], [2, 1]])
b = np.array([6, 2, 5])
c = np.array([1, -2])

# Menyelesaikan sistem persamaan linier menggunakan algoritma simplex
res = linprog(c, A_ub=A, b_ub=b, method='simplex')

print('Nilai minimum fungsi tujuan:', res.fun)
print('Nilai variabel x:', res.x)
```

Script di atas mengimplementasikan pemecahan masalah pemrograman linier menggunakan algoritma simplex dengan menggunakan fungsi linprog dari modul scipy.optimize dalam Python.

Langkah-langkah utama yang dilakukan dalam script ini adalah sebagai berikut:

a. Impor modul numpy dan linprog dari scipy.optimize.
b. Bentuk matriks koefisien batasan A, vektor batasan b, dan koefisien fungsi tujuan c menggunakan array NumPy.
c. Gunakan fungsi linprog dengan argumen c sebagai koefisien fungsi tujuan, A_ub sebagai matriks koefisien batasan (dalam bentuk ≤), dan b_ub sebagai vektor batasan.

d. Hasil dari pemecahan masalah linier disimpan dalam variabel res.
e. Cetak nilai minimum fungsi tujuan dengan res.fun, yang merupakan nilai minimum yang dihasilkan oleh algoritma simplex.
f. Cetak nilai variabel x dengan res.x, yang merupakan solusi optimum yang ditemukan oleh algoritma.

Dengan menggunakan algoritma simplex, script ini mencari solusi optimum yang meminimalkan fungsi tujuan dengan memenuhi batasan linier yang diberikan. Output dari script ini adalah nilai minimum fungsi tujuan dan nilai variabel x yang merupakan solusi optimum dari masalah linier yang diberikan.

3. implementasi strategi algoritma aproksimasi pada masalah pengalokasian sumber daya dengan batasan menggunakan bahasa pemrograman Python

```
import numpy as np
from scipy.optimize import linprog

# Membentuk sistem persamaan linier dengan numpy
A = np.array([[-3, -2, -4, 0], [-1, -4, 0, -3], [-2, -3, -5, -4]])
b = np.array([-15, -10, -20])
c = np.array([-10, -8, -15, 0])

# Menyelesaikan sistem persamaan linier menggunakan algoritma simplex
res = linprog(c, A_ub=A, b_ub=b, method='simplex')

print('Nilai maksimum fungsi tujuan:', -res.fun)
print('Nilai variabel x:', res.x)
```

Script di atas mengimplementasikan pemecahan masalah pemrograman linier menggunakan algoritma simplex dengan menggunakan fungsi linprog dari modul scipy.optimize dalam Python.

Langkah-langkah utama yang dilakukan dalam script ini adalah sebagai berikut:

a. Impor modul numpy dan linprog dari scipy.optimize.
b. Bentuk matriks koefisien batasan A, vektor batasan b, dan koefisien fungsi tujuan c menggunakan array NumPy.
c. Gunakan fungsi linprog dengan argumen c sebagai koefisien fungsi tujuan, A_ub sebagai matriks koefisien batasan (dalam bentuk ≤), dan b_ub sebagai vektor batasan.
d. Hasil dari pemecahan masalah linier disimpan dalam variabel res.
e. Cetak nilai maksimum fungsi tujuan dengan -res.fun, di mana -res.fun adalah negasi dari nilai minimum yang dihasilkan oleh algoritma simplex.

f. Cetak nilai variabel x dengan res.x, yang merupakan solusi optimum yang ditemukan oleh algoritma.

Dalam masalah ini, script mencari solusi optimum yang memaksimalkan fungsi tujuan dengan memenuhi batasan linier yang diberikan. Output dari script ini adalah nilai maksimum fungsi tujuan dan nilai variabel x yang merupakan solusi optimum dari masalah linier yang diberikan.

Soal :

1. Apa itu algoritma aproksimasi dan apa perbedaannya dengan algoritma eksak?
   Jawab : Algoritma aproksimasi adalah jenis algoritma yang digunakan untuk mencari solusi yang mendekati solusi optimal dalam masalah optimisasi yang kompleks atau NP-sulit. Algoritma ini memberikan solusi yang dapat diterima dalam waktu yang wajar, meskipun tidak menjamin solusi yang tepat atau optimal.
   Perbedaan utama antara algoritma aproksimasi dan algoritma eksak adalah sebagai berikut:
   a. Solusi yang diberikan: Algoritma eksak bertujuan untuk memberikan solusi yang optimal atau tepat sesuai dengan kriteria yang ditentukan. Artinya, algoritma eksak mencari solusi yang benar-benar optimal untuk masalah yang diberikan. Di sisi lain, algoritma aproksimasi memberikan solusi yang mendekati solusi optimal, namun tidak menjamin solusi yang tepat. Solusi yang diberikan oleh algoritma aproksimasi mungkin memiliki kualitas yang lebih rendah dibandingkan solusi optimal, tetapi tetap memenuhi batasan waktu atau sumber daya yang ada.
   b. Kompleksitas waktu: Algoritma eksak cenderung memiliki kompleksitas waktu yang lebih tinggi karena mencoba semua kemungkinan solusi untuk mencari solusi optimal. Dalam beberapa kasus, algoritma eksak dapat memerlukan waktu yang sangat lama atau bahkan tidak memungkinkan untuk mencari solusi optimal dalam waktu yang terbatas. Di sisi lain, algoritma aproksimasi dirancang untuk bekerja dalam waktu yang wajar atau dapat dibatasi. Algoritma aproksimasi sering kali lebih efisien dalam hal kompleksitas waktu dibandingkan dengan algoritma eksak.
   c. Ketepatan solusi: Algoritma eksak memberikan solusi yang tepat dan optimal sesuai dengan kriteria yang ditentukan. Dalam banyak kasus, algoritma eksak dapat memberikan solusi yang terbaik. Namun, dalam beberapa masalah yang kompleks, mencari solusi yang tepat secara eksak mungkin tidak mungkin dalam waktu yang terbatas. Algoritma aproksimasi memberikan solusi yang mendekati solusi optimal, tetapi tidak menjamin solusi yang tepat. Ketepatan solusi yang diberikan oleh algoritma aproksimasi dapat bervariasi tergantung pada masalah yang diberikan.

   Dalam beberapa situasi, algoritma aproksimasi merupakan pilihan yang baik ketika solusi yang tepat atau optimal tidak dapat ditemukan dalam waktu yang terbatas atau ketika kompleksitas waktu menjadi faktor yang kritis. Meskipun solusi yang diberikan oleh algoritma aproksimasi tidak sempurna, mereka sering kali cukup baik dan dapat digunakan dalam banyak kasus praktis.

2. Jelaskan strategi dasar dalam mengembangkan algoritma aproksimasi.
   Jawab : Strategi dasar dalam mengembangkan algoritma aproksimasi melibatkan pendekatan sistematis untuk mendekati solusi optimal atau memperoleh solusi yang cukup baik dalam waktu yang wajar. Berikut adalah beberapa strategi dasar yang sering digunakan dalam mengembangkan algoritma aproksimasi:
   a. Relaksasi Masalah: Strategi ini melibatkan relaksasi atau pengurangan batasan pada masalah yang kompleks sehingga menjadi lebih mudah untuk mencari solusi yang mendekati solusi optimal. Dalam beberapa kasus, batasan yang ketat atau sulit dipecahkan dapat diubah atau dilemahkan untuk mencapai solusi yang lebih mudah dicari. Dengan memperoleh solusi relaksasi yang lebih cepat, kemudian diperbaiki atau diperkaya, dapat ditemukan solusi yang mendekati solusi optimal.
   b. Pendekatan Greedy: Pendekatan ini melibatkan pengambilan keputusan berdasarkan langkah-langkah lokal yang optimal pada setiap langkah dalam algoritma. Pada setiap tahap, keputusan diambil dengan mempertimbangkan keuntungan terbesar atau biaya terkecil pada saat itu, tanpa mempertimbangkan konsekuensi jangka panjang secara keseluruhan. Pendekatan ini sering digunakan dalam algoritma aproksimasi karena sederhana dan dapat memberikan solusi yang cukup baik dalam waktu yang cepat.
   c. Pendekatan Pembungkus (Envelope Methods): Pendekatan ini melibatkan pembentukan suatu "pembungkus" (envelope) atau batasan atas dan batasan bawah pada solusi optimal yang diketahui. Dengan memperoleh batasan atas dan batasan bawah, dapat dilakukan pencarian yang lebih efisien dalam ruang solusi. Algoritma aproksimasi dapat berfokus pada mencari solusi di dalam batasan atas dan batasan bawah tersebut untuk mendekati solusi optimal.
   d. Pendekatan Pembagian dan Penaklukan: Pendekatan ini melibatkan pembagian masalah yang kompleks menjadi submasalah yang lebih kecil dan lebih mudah ditangani. Setiap submasalah kemudian dipecahkan secara terpisah, dan solusi dari setiap submasalah digabungkan untuk memperoleh solusi akhir. Dengan membagi masalah menjadi submasalah yang lebih sederhana, algoritma aproksimasi dapat memperoleh solusi yang mendekati solusi optimal dengan cara yang lebih efisien.
   e. Metode Heuristik: Metode heuristik melibatkan penggunaan aturan atau strategi berbasis pengalaman atau pengetahuan domain untuk mencari solusi yang memadai. Heuristik dapat memberikan petunjuk atau pedoman dalam pencarian solusi, meskipun tidak menjamin solusi yang optimal. Metode heuristik sering digunakan dalam algoritma aproksimasi untuk mencapai solusi yang memadai dalam waktu yang wajar.

3. Berikan contoh kasus di mana algoritma aproksimasi dapat diterapkan dalam kehidupan sehari-hari.
   Jawab : Rute Perjalanan: Dalam perencanaan perjalanan, algoritma aproksimasi dapat digunakan untuk mencari rute perjalanan yang efisien. Misalnya, dalam kasus Traveling Salesman Problem (TSP) di mana salesman harus mengunjungi beberapa kota dengan jarak terpendek, algoritma aproksimasi dapat memberikan rute yang

mendekati rute optimal dalam waktu yang wajar.

Berikut adalah contoh implementasi algoritma aproksimasi untuk kasus Traveling Salesman Problem (TSP) dalam Python menggunakan pendekatan Greedy:

```
import numpy as np

def calculate_distance(city1, city2):
    # Fungsi ini menghitung jarak antara dua kota menggunakan metode Euclidean
    x1, y1 = city1
    x2, y2 = city2
    return np.sqrt((x2 - x1)**2 + (y2 - y1)**2)

def find_nearest_neighbor(current_city, unvisited_cities):
    # Fungsi ini mencari kota terdekat dari kota saat ini
    nearest_city = None
    min_distance = float('inf')

    for city in unvisited_cities:
        distance = calculate_distance(current_city, city)
        if distance < min_distance:
            min_distance = distance
            nearest_city = city

    return nearest_city

def tsp_approximation(cities):
    start_city = cities[0]  # Memilih kota awal
    unvisited_cities = cities[1:]  # Daftar kota yang belum dikunjungi
    current_city = start_city
    tour = [current_city]  # Inisialisasi tur dengan kota awal

    while unvisited_cities:
        nearest_city = find_nearest_neighbor(current_city, unvisited_cities)
        tour.append(nearest_city)
        unvisited_cities.remove(nearest_city)
        current_city = nearest_city

    tour.append(start_city)  # Kembali ke kota awal untuk melengkapi tur

    return tour

# Contoh penggunaan
cities = [(0, 0), (1, 5), (2, 3), (5, 2), (6, 4)]
tour = tsp_approximation(cities)

print("Tur Perjalanan:", tour)
```

Dengan hasil

```
Tur Perjalanan: [(0, 0), (2, 3), (1, 5), (5, 2), (6, 4), (0, 0)]
```

4. Bagaimana algoritma aproksimasi dapat membantu dalam menyelesaikan masalah optimasi linier?
   Jawab : Algoritma aproksimasi dapat membantu dalam menyelesaikan masalah optimasi linier dengan memberikan solusi yang mendekati solusi optimal dalam waktu yang terbatas. Masalah optimasi linier melibatkan mencari nilai maksimum atau minimum dari fungsi linier yang terkait dengan sejumlah variabel, dengan mempertimbangkan batasan linier.
   Berikut adalah beberapa cara algoritma aproksimasi dapat digunakan dalam masalah

optimasi linier:

a. Pendekatan Heuristik: Algoritma aproksimasi dapat menggunakan pendekatan heuristik untuk mencari solusi yang memadai dalam waktu yang wajar. Pendekatan heuristik menggunakan aturan atau strategi berbasis pengalaman atau pengetahuan domain untuk memandu pencarian solusi. Meskipun tidak menjamin solusi optimal, pendekatan heuristik dapat memberikan solusi yang cukup baik dan dapat diterima dalam banyak kasus.
b. Metode Relaksasi: Dalam beberapa kasus, masalah optimasi linier yang kompleks dapat dipecahkan dengan menggunakan metode relaksasi. Metode ini melibatkan relaksasi atau pengurangan batasan yang ketat pada masalah, sehingga memungkinkan pencarian solusi yang lebih efisien. Misalnya, dengan mengabaikan beberapa batasan yang tidak kritis atau mengizinkan toleransi kesalahan tertentu dalam batasan, algoritma aproksimasi dapat mencari solusi yang mendekati solusi optimal dengan waktu yang lebih singkat.
c. Metode Pembagian dan Penaklukan: Algoritma aproksimasi dapat menggunakan metode pembagian dan penaklukan untuk menyelesaikan masalah optimasi linier yang kompleks. Metode ini melibatkan pembagian masalah menjadi submasalah yang lebih kecil dan lebih mudah ditangani. Setiap submasalah kemudian dipecahkan secara terpisah, dan solusi dari setiap submasalah digabungkan untuk memperoleh solusi akhir. Dengan membagi masalah menjadi submasalah yang lebih sederhana, algoritma aproksimasi dapat memperoleh solusi yang mendekati solusi optimal secara keseluruhan.

Pendekatan Greedy: Pendekatan Greedy juga dapat diterapkan dalam masalah optimasi linier. Algoritma aproksimasi menggunakan pendekatan Greedy dengan memilih langkah-langkah lokal yang optimal pada setiap tahap dalam algoritma. Pada setiap tahap, algoritma aproksimasi memilih langkah yang memberikan peningkatan terbesar dalam fungsi objektif atau pemenuhan batasan, tanpa mempertimbangkan konsekuensi jangka panjang secara keseluruhan. Pendekatan Greedy dapat memberikan solusi yang memadai dalam waktu yang cepat, meskipun tidak menjamin solusi optimal.

5. Jelaskan bagaimana implementasi strategi algoritma aproksimasi dapat dilakukan pada masalah optimasi nonlinear dengan Python

   Jawab : Berikut adalah contoh sederhana untuk mengoptimalkan fungsi objektif nonlinear menggunakan metode optimasi BFGS dalam pustaka SciPy:

```python
from scipy.optimize import minimize

# Fungsi objektif
def objective_function(x):
    return (x[0]**2 + x[1]**2)

# Batasan
def constraint(x):
    return (x[0] + x[1] - 1)

# Memasukkan fungsi objektif dan batasan ke dalam objek optimization
problem = {
    'fun': objective_function,
    'constraints': [{'type': 'eq', 'fun': constraint}]
}

# Menjalankan optimisasi menggunakan metode BFGS
result = minimize(**problem, method='SLSQP', x0=[0, 0])

# Mencetak hasil
print("Nilai optimum:", result.x)
print("Nilai fungsi objektif:", result.fun)
print("Status konvergensi:", result.success)
```

Dengan hasil

```
Nilai optimum: [0.5 0.5]
Nilai fungsi objektif: 0.5
Status konvergensi: True
```

Perubahan yang dilakukan adalah:

1. Mengubah 'objective' menjadi 'fun' dalam objek problem untuk mencocokkan argumen yang diterima oleh fungsi minimize.
2. Mengubah metode optimasi menjadi 'SLSQP' karena metode 'BFGS' tidak mendukung batasan.
3. Mengganti 'x' dengan 'x0' sebagai argumen yang mengindikasikan tebakan awal (initial guess) dalam fungsi minimize.

Dengan perubahan tersebut, kode sekarang akan berfungsi dengan baik untuk mengoptimalkan fungsi objektif nonlinear dengan batasan menggunakan metode optimasi SLSQP.