

Laporan praktikum DAA (latihan Algoritma Aproksimasi)

Nama : Yohanes Yeningga

Nim : 20220047

Matkul : Prak. DAA

A. Algoritma aproksimasi adalah algoritma yang digunakan untuk menyelesaikan masalah optimasi dengan memberikan solusi yang mendekati solusi optimal. Tujuan dari algoritma aproksimasi adalah mencapai solusi yang cukup baik dalam waktu yang efisien, walaupun tidak menjamin solusi yang optimal.

Perbedaan utama antara algoritma aproksimasi dan algoritma eksak adalah sebagai berikut:

1. Solusi: Algoritma eksak berusaha mencari solusi yang optimal secara eksak, yaitu solusi yang memberikan nilai yang paling baik atau optimal. Algoritma aproksimasi, di sisi lain, memberikan solusi yang mendekati solusi optimal tetapi tidak menjamin solusi yang tepat.
2. Kinerja: Algoritma eksak seringkali memerlukan waktu eksekusi yang lebih lama karena harus memeriksa semua kemungkinan solusi untuk menemukan yang terbaik. Algoritma aproksimasi dirancang untuk memberikan solusi yang cukup baik dengan waktu eksekusi yang lebih cepat daripada algoritma eksak.
3. Kompleksitas: Kompleksitas algoritma eksak cenderung lebih tinggi karena harus mempertimbangkan semua kemungkinan solusi. Di sisi lain, algoritma aproksimasi seringkali lebih sederhana dan memiliki kompleksitas yang lebih rendah.
4. Keandalan: Algoritma eksak dapat memberikan solusi yang optimal dan dijamin benar. Algoritma aproksimasi, meskipun tidak menjamin solusi yang optimal, sering kali memberikan solusi yang cukup baik dan dapat diterima dalam banyak kasus.

B. Dalam prakteknya, algoritma aproksimasi sering digunakan ketika mencari solusi yang cukup baik sudah lebih penting daripada mencari solusi yang tepat. Hal ini terjadi dalam kasus-kasus di mana waktu eksekusi dan kompleksitas algoritma menjadi faktor yang kritis, atau ketika mencari solusi optimal terbukti sangat sulit atau tidak efisien.

Dalam mengembangkan algoritma aproksimasi, terdapat beberapa strategi dasar yang umum digunakan. Berikut adalah beberapa strategi dasar yang sering digunakan:

1. Greedy Approach (Pendekatan Rakus): Strategi ini memilih langkah terbaik pada setiap tahap untuk mencapai solusi yang optimal secara lokal. Pada setiap langkah, algoritma akan memilih langkah yang paling menguntungkan atau menghasilkan keuntungan terbesar pada saat itu. Meskipun strategi ini cenderung memberikan solusi yang cepat, tidak menjamin solusi yang optimal secara global.
2. Randomized Algorithms (Algoritma Acak): Strategi ini menggunakan elemen acak dalam proses algoritma untuk mencapai solusi yang mendekati solusi optimal. Algoritma acak dapat menjelajahi ruang solusi dengan cara yang berbeda-beda, menghindari kemungkinan terjebak pada solusi lokal yang suboptimal.
3. Rounding Techniques (Teknik Pembulatan): Strategi ini melibatkan pembulatan atau pengubahan nilai solusi yang ditemukan menjadi solusi yang lebih mudah dicapai atau lebih dekat dengan solusi yang optimal. Teknik ini sering digunakan dalam masalah optimasi yang melibatkan bilangan pecahan atau kontinu.
4. Approximation by Sampling (Aproksimasi dengan Sampel): Strategi ini melibatkan penggunaan sampel acak dari data atau ruang solusi untuk mencapai solusi yang mendekati solusi optimal. Dengan menggunakan sampel, algoritma dapat memberikan perkiraan yang baik tentang solusi sebenarnya.
5. Divide and Conquer (Pecah dan Kuasai): Strategi ini memecah masalah besar menjadi submasalah yang lebih kecil, kemudian mencari solusi untuk setiap submasalah, dan menggabungkan solusi submasalah tersebut untuk mencapai solusi yang lebih baik secara keseluruhan. Teknik ini sering digunakan dalam algoritma aproksimasi untuk mengatasi masalah yang kompleks.

Strategi-strategi di atas dapat digunakan secara mandiri atau dikombinasikan dalam pengembangan algoritma aproksimasi. Pilihan strategi yang tepat tergantung pada jenis masalah yang ingin diselesaikan dan ketersediaan sumber daya yang ada.

- C. Contoh kasus di mana algoritma aproksimasi dapat diterapkan dalam kehidupan sehari-hari adalah:

1. Perencanaan Rute Perjalanan: Ketika merencanakan rute perjalanan, algoritma aproksimasi dapat digunakan untuk mencari rute yang mendekati rute optimal dengan mempertimbangkan waktu tempuh, jarak, dan kemacetan lalu lintas. Algoritma ini memberikan solusi yang cukup baik dalam waktu yang efisien, meskipun tidak menjamin rute yang benar-benar optimal.
2. Packing Masalah (Masalah Penyusunan Barang): Ketika harus memilih dan menyusun barang-barang dalam koper atau wadah dengan kapasitas terbatas, algoritma aproksimasi dapat digunakan untuk mencari solusi yang mendekati solusi optimal. Algoritma ini dapat membantu dalam mengoptimalkan penggunaan ruang dan memastikan barang-barang tertata dengan efisien.
3. Penjadwalan Kegiatan: Dalam mengatur jadwal kegiatan sehari-hari, algoritma aproksimasi dapat digunakan untuk mencari penjadwalan yang mendekati penjadwalan optimal dengan mempertimbangkan waktu, prioritas, dan keterbatasan lainnya. Algoritma ini membantu mengoptimalkan penggunaan waktu dan memastikan kegiatan-kegiatan dapat diselesaikan dengan efisien.
4. Pemilihan Restoran: Ketika mencari restoran atau tempat makan, algoritma aproksimasi dapat digunakan untuk menyaring dan mengurutkan pilihan berdasarkan faktor-faktor seperti jarak, rating, dan preferensi pribadi. Algoritma ini dapat membantu dalam mencari pilihan yang memadai secara cepat dan efisien, meskipun tidak menjamin pilihan yang benar-benar optimal.
5. Pengaturan Jadwal Penerbangan: Dalam merencanakan penerbangan, algoritma aproksimasi dapat digunakan untuk mencari kombinasi penerbangan yang mendekati jadwal optimal dengan mempertimbangkan faktor-faktor seperti waktu perjalanan, harga tiket, dan ketersediaan penerbangan. Algoritma ini membantu dalam mencari opsi perjalanan yang memadai dalam waktu yang efisien.

Ini hanya beberapa contoh dari banyak kasus di mana algoritma aproksimasi dapat diterapkan dalam kehidupan sehari-hari. Dalam situasi di mana mencari solusi yang optimal sulit atau tidak efisien, algoritma aproksimasi dapat memberikan solusi yang cukup baik dan dapat diterima.

- D. Algoritma aproksimasi dapat membantu dalam menyelesaikan masalah optimasi linier dengan memberikan solusi yang mendekati solusi optimal. Dalam masalah optimasi linier, tujuan utamanya adalah mencari nilai maksimum atau minimum dari fungsi linier yang terkait dengan batasan-batasan tertentu.

Berikut adalah beberapa cara di mana algoritma aproksimasi dapat digunakan dalam menyelesaikan masalah optimasi linier:

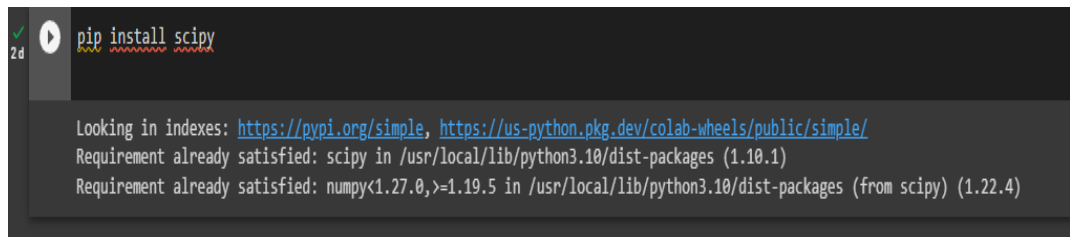
- 1). Metode Penurunan Dimensi: Algoritma aproksimasi dapat menggunakan metode penurunan dimensi untuk mengatasi masalah optimasi linier yang kompleks. Metode ini mengurangi jumlah variabel dalam masalah dengan menggantikan variabel-variabel tertentu dengan estimasi atau pendekatan yang lebih sederhana. Dengan mengurangi dimensi masalah, algoritma dapat mencapai solusi yang lebih cepat dan mendekati solusi optimal.
- 2). Pendekatan Heuristik: Algoritma aproksimasi dapat menggunakan pendekatan heuristik untuk mencari solusi yang mendekati solusi optimal. Pendekatan heuristik mengikuti aturan-aturan yang tidak terlalu ketat untuk menemukan solusi yang cukup baik. Contohnya, algoritma greedy dapat digunakan untuk memilih variabel yang memberikan kontribusi terbesar dalam meningkatkan fungsi objektif, atau algoritma randomisasi dapat digunakan untuk menjelajahi ruang solusi secara acak. Meskipun tidak menjamin solusi yang optimal, pendekatan heuristik seringkali memberikan solusi yang cukup dekat dengan solusi optimal dalam waktu yang lebih cepat.
- 3). Teknik Pembulatan: Algoritma aproksimasi dapat menggunakan teknik pembulatan untuk menghasilkan solusi yang mendekati solusi optimal. Dalam beberapa kasus, solusi optimal dari masalah optimasi linier dapat berupa bilangan pecahan atau kontinu. Dalam algoritma aproksimasi, solusi tersebut dapat dibulatkan menjadi bilangan bulat atau nilai yang lebih sederhana untuk mencapai solusi yang lebih mudah dicapai. Meskipun solusi yang dibulatkan tidak optimal secara tepat, mereka dapat memberikan solusi yang cukup dekat dan dapat diterima.

Penerapan algoritma aproksimasi dalam masalah optimasi linier tergantung pada karakteristik masalah yang dihadapi dan ketersediaan sumber daya yang ada. Penting untuk diingat bahwa algoritma aproksimasi bertujuan untuk memberikan solusi yang cukup baik dalam waktu yang efisien, meskipun tidak menjamin solusi yang optimal.

- E. Implementasi strategi algoritma aproksimasi pada masalah optimasi nonlinear dengan Python dapat dilakukan dengan menggunakan pustaka atau library yang mendukung pemodelan dan penyelesaian masalah optimasi. Salah satu pustaka populer yang dapat digunakan adalah SciPy. Berikut adalah langkah-langkah umum untuk

mengimplementasikan strategi algoritma aproksimasi dalam masalah optimasi nonlinear menggunakan Python dan SciPy:

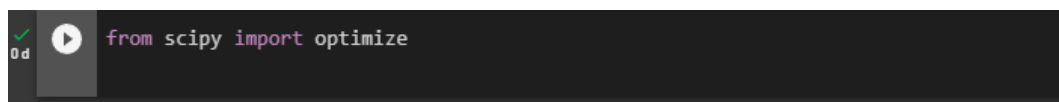
1. Instalasi Pustaka: Pastikan Anda memiliki Python yang terinstal di komputer Anda. Kemudian, instal pustaka SciPy dengan menggunakan perintah pip atau conda dalam command prompt atau terminal:



```
2d pip install scipy

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (1.10.1)
Requirement already satisfied: numpy<1.27.0,>=1.19.5 in /usr/local/lib/python3.10/dist-packages (from scipy) (1.22.4)
```

2. Impor Modul: Di dalam skrip Python Anda, impor modul yang diperlukan dari pustaka SciPy. Untuk masalah optimasi nonlinear, kita akan menggunakan modul optimize:



```
0d from scipy import optimize
```

3. Tentukan Fungsi Objektif dan Batasan: Tentukan fungsi objektif yang ingin dioptimalkan dan batasan-batasan yang relevan dalam bentuk persamaan atau ketidaksetaraan. Fungsi objektif adalah fungsi yang ingin kita maksimalkan atau minimalkan, sedangkan batasan bisa berupa batasan kesetaraan atau ketidaksetaraan terkait variabel yang terlibat dalam masalah. Pastikan fungsi objektif dan batasan diimplementasikan sebagai fungsi Python.
4. Gunakan Algoritma Aproksimasi: Pilih dan gunakan algoritma aproksimasi yang tepat dari pustaka SciPy. SciPy menyediakan beberapa metode numerik yang berguna untuk menyelesaikan masalah optimasi, seperti BFGS, Nelder-Mead, Powell, dan lain-lain. Metode yang cocok akan tergantung pada karakteristik masalah yang dihadapi. Contoh di bawah ini menggunakan metode BFGS:

```
result = optimize.minimize(fungsi_objektif, x0, method='BFGS',
                           constraints=batasan)
```

Di sini, fungsi_objektif adalah fungsi objektif yang telah Anda tentukan, x0 adalah tebakan awal (tebakan pendekatan awal), dan batasan adalah batasan yang telah Anda tentukan.

5. Analisis Hasil: Setelah algoritma selesai dijalankan, Anda dapat menganalisis hasilnya. Misalnya, Anda dapat mengakses nilai optimal yang ditemukan melalui `result.x` dan nilai minimum atau maksimum fungsi objektif melalui `result.fun`. Anda juga dapat memeriksa apakah optimasi berhasil atau tidak melalui `result.success`.

Perhatikan bahwa langkah-langkah ini hanyalah panduan umum dan dapat disesuaikan dengan kebutuhan dan kompleksitas masalah optimasi nonlinear yang Anda hadapi. Penting juga untuk mempelajari dokumentasi pustaka yang digunakan dan memahami metode yang Anda pilih agar dapat mengoptimalkan implementasi algoritma aproksimasi.