JARINGAN DAN PENGOLAHAN DATA PARALEL

LAPORAN PRAKTIKUM

Mohammad Rizka Fadhli

Ikang

20921004@mahasiswa.itb.ac.id

31 December 2021

Contents

1 *INTRODUCTION* 6

1.1 Definisi . . . 6

1.2 Perbedaan *Serial Processing* dan *Parallel Processing* . . . 6

1.3 Cara Kerja *Parallel Processing* . . . 8

1.3.1 **SISD** . . . 8

1.3.2 **SIMD** . . . 9

1.3.3 **MISD** . . . 9

1.3.4 **MIMD** . . . 9

1.4 Tugas Praktikum . . . 9

1.5 *Server* yang Digunakan . . . 10

1.6 MPI . . . 13

2 *METHOD* 14

2.0.1 *Broadcast* . . . 14

2.0.2 *Scatter* . . . 14

2.0.3 *Gather* . . . 15

2.0.4 *Point to Point* . . . 15

2.1 Metode Integral Numerik . . . 15

2.1.1 Metode Titik Tengah (*Midpoint*) . . . 15

2.1.2 Simulasi Monte Carlo . . . 16

2.2 Menghitung π . . . 16

2.3 Penjumlahan dan Perkalian Matriks . . . 16

3 *RESULT AND DISCUSSION* 17

3.1 Soal I . . . 17

3.1.1 Jawab . . . 17

3.2 Soal II . . . 18

3.3 Soal III . . . 19

4 *CONCLUSION* 20

REFERENCES **21**

List of Figures

1 Ilustrasi Perbedaan Serial dan Parallel Processing 7
2 Ilustrasi SISD 8
3 Ilustrasi SIMD 9
4 Spesifikasi Server yang Digunakan 11
5 Tampilan Awal Setelah Login ssh 12
6 lscpu dari Server 12
7 htop dari Server 13
8 Versi MPI yang Digunakan 13
9 Flowchart Simulasi Monte Carlo untuk Integral Numerik 16
10 Rekap Runtime Soal I 19

List of Tables

1 *INTRODUCTION*

1.1 Definisi

Parallel processing adalah metode komputasi untuk menggunakan dua atau lebih *processors* untuk menjalankan beberapa tugas secara terpisah atau secara keseluruhan. Setiap komputer yang memiliki lebih dari satu *CPUs* atau memiliki *processor multi cores* bisa melakukan *parallel processing.*[1]

1.2 Perbedaan *Serial Processing* dan *Parallel Processing*

Perbedaan mendasar dari *serial processing* dan *parallel processing* adalah dari segi bagaimana komputer melakukan proses komputasi. *Serial processing* berarti komputer melakukan tugasnya secara sekuensial (berurutan) menggunakan satu *processor*. Akibatnya adalah saat melakukan suatu proses yang kompleks, *runtime* yang diperlukan lebih lama karena *processor* harus memproses data satu-persatu.

Berbeda halnya dengan *parallel processing*. Tugas yang dilakuan komputer didistribusikan kepada sejumlah *processors* untuk diolah secara bersamaan. Konsekuensinya adalah *runtime* komputasi lebih singkat. Namun perlu diperhatikan dengan seksama bahwa tidak semua tugas bisa kita buat paralelisasinya dan cara kita menulis algoritma atau *coding* harus disesuaikan.

Kenapa tidak semua tugas bisa diparalelisasi?

Beberapa tugas sekuensial yang tidak bisa dihindari tidak bisa diparalelisasi.

Sebagai contoh:

1. *Looping* yang prosesnya tidak saling bergantung bisa diparalelisasi. Misalkan ada suatu fungsi untuk menghitung suatu *array* bisa diparalelisasi dengan cara memecah *array* tersebut untuk diproses bersamaan di beberapa *processors*.
2. *Looping* yang prosesnya saling bergantung tidak bisa diparalelisasi. Misalkan suatu *looping* ke i nilainya bergantung pada proses *looping* ke $i-1$.

[1] https://searchdatacenter.techtarget.com/definition/parallel-processing

Berikut adalah ilustrasi perbedaan serial dan *parallel processing*:

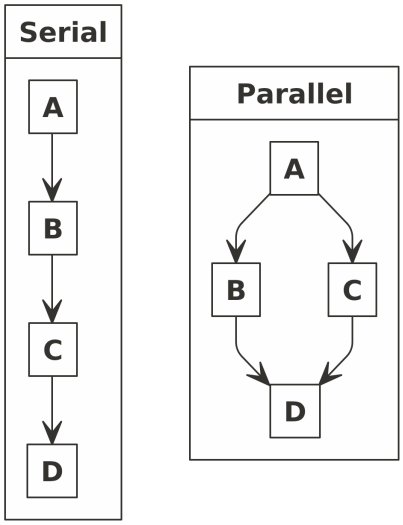

Figure 1: Ilustrasi Perbedaan Serial dan Parallel Processing

1.3 Cara Kerja *Parallel Processing*

Untuk melakukan *parallel processing*, dibutuhkan *hardware* dan *software* yang mendukung hal tersebut. Secara *hardware* dibutuhkan komputer dengan *multiple cores processors* atau dibutuhkan beberapa komputer yang digabung menjadi satu kesatuan. Secara *software* dibutuhkan tidak hanya `Python` tapi juga *middleware* bernama `Open MPI`. Bagian *hardware* dan *software* ini akan dibahas pada bagian selanjutnya.

Pada sistem *parallel processing* terdiri dari beberapa unit *processors* dan beberapa unit *memory*. Ada dua teknik berbeda yang digunakan untuk mengakses data di unit *memory*, yaitu: *shared memory address* dan *message passing*.

Berdasarkan cara mengorganisasikan memori ini komputer bisa dibedakan menjadi *shared memory parallel machine* dan *distributed memory parallel machine*.

Ada empat model komputasi yang dikenal dalam taksonomi Flynn, yaitu:

1. **SISD** (*Single Instruction, Single Data*)
2. **SIMD** (*Single Instruction, Multiple Data*)
3. **MISD** (*Multiple Instruction, Single Data*)
4. **MIMD** (*Multiple Instruction, Multiple Data*)

1.3.1 SISD

Komputer ini adalah tipikal komputer konvensional yang hanya memiliki satu *processor* dan satu instruksi yang dieksekusi secara serial. Komputer jenis ini tidak bisa melakukan *parallel processing*.

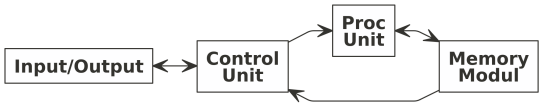

Figure 2: Ilustrasi SISD

1.3.2 SIMD

Komputer ini memiliki lebih dari satu *processor* tapi hanya mengeksekusi satu instruksi secara paralel pada data yang berbeda pada level *lock-step*. Contohnya adalah komputer vektor.

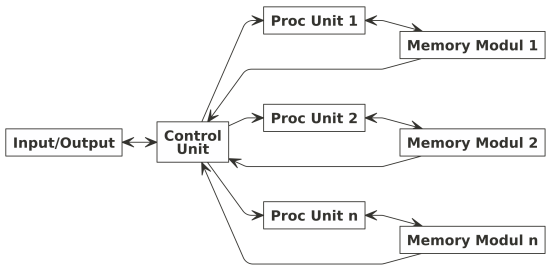

Figure 3: Ilustrasi SIMD

1.3.3 MISD

Komputer jenis ini belum diciptakan karena secara arsitekturnya tidak mudah dipahami. Secara teori komputer ini memiliki satu *processor* dan mengeksekusi beberapa instruksi secara paralel.

1.3.4 MIMD

Komputer berarsitektur ini paling banyak digunakan untuk membangun *super computer*. Komputer ini memiliki lebih dari satu *processors* dan mengeksekusi lebih dari satu instruksi secara paralel.

1.4 Tugas Praktikum

Pada praktikum ini, saya akan mengerjakan:

- Dua buah tugas terkait penyelesaian integral secara numerik memanfaatkan metode dikritisasi nilai tengah (*midpoint*) dan simulasi Monte Carlo.
- Satu buah tugas terkait penjumlahan dan perkalian matriks $n \times n$.

Ketiga tugas tersebut akan diselesaikan menggunakan serial dan *parallel processing*.

1.5 *Server* yang Digunakan

Pada praktikum kali ini, saya tidak bisa menggunakan *server* **HPC** yang disediakan oleh ITB karena masalah koneksi. Oleh karena itu, saya menggunakan *server* lain agar bisa menduplikasi apa yang seharusnya dikerjakan di *server* ITB.

Saya menggunakan *server virtual machine* milik *Google Cloud*[2]. *Server* ini memiliki *processor* **Intel Xeon 8 *cores***. *Hostname* dari *server* ini saya beri nama `praktikum`.

Server ini disewa menggunakan *free credit* yang kita dapatkan saat mengaktifkan layanan *Google Cloud* menggunakan akun *Google*.

[2] https://ikanx101.com/blog/vm-cloud/

Berikut adalah spesifikasinya:

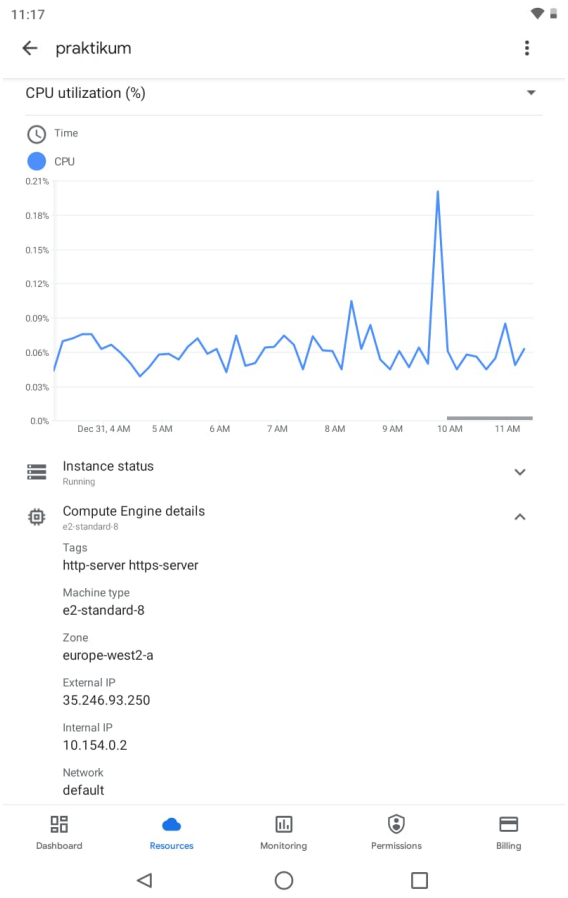

Figure 4: Spesifikasi Server yang Digunakan

Server ini bisa diakses menggunakan *command line* menggunakan `ssh` langsung ke *IP Public* yang diberikan *Google*.

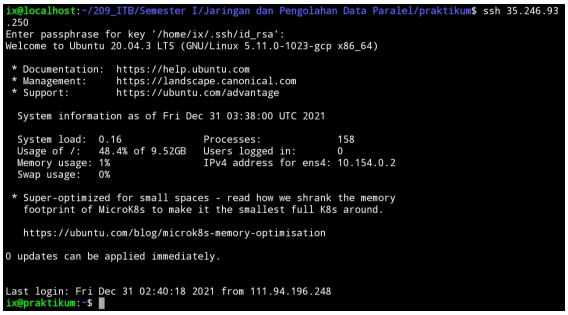

Figure 5: Tampilan Awal Setelah Login ssh

Server ini berjalan di *operating system* `Ubuntu Linux 20.04 LTS`.

Berikut adalah tampilan hasil `lscpu`:

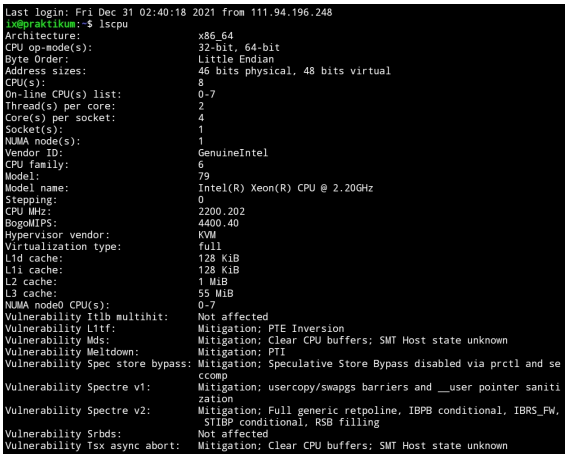

Figure 6: lscpu dari Server

Berikut adalah tampilan hasil `htop`:

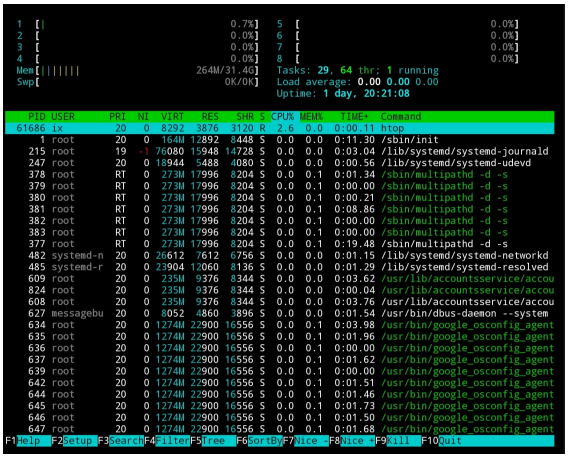

Figure 7: htop dari Server

1.6 MPI

Secara *default* `Python` sudah ter-*install* di *server* namun tidak untuk `Open MPI`. Oleh karena itu, salah satu langkah yang perlu dilakukan adalah meng-*install*-nya terlebih dahulu (Jarno Rantaharju and Bennet 2019).

Pada `Ubuntu`, proses instalasinya bisa kita lakukan dengan mengetikkan perintah berikut ini di *command line*:

```
sudo apt install openmpi-bin openmpi-dev openmpi-common openmpi-doc libopenmpi-dev
```

Setelah proses instalasi selesai, kita bisa mengecek versi `Open MPI` yang berjalan di *server* sebagai berikut:

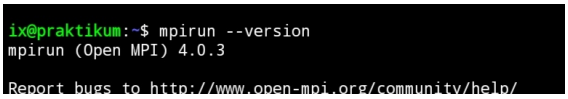

Figure 8: Versi MPI yang Digunakan

Pada `Python`, saya perlu meng-*install library* `mpi4py` dengan perintah:

```
sudo apt install python3-mpi4py
```

Oleh karena saya menjalankan program `Python` praktikum di *server* sendiri (tanpa ada *user* lain), maka saya tidak melakukan instalasi `SLURM`.

2 *METHOD*

Pada praktikum ini, kita akan melakukan *parallel processing* menggunakan `Python` versi `3.8.10` di *server* berbasis `Linux Ubuntu OS`. Ada beberapa metode *parallel processing* yang hendak dilakukan, yakni:

1. *Broadcast*,
    1. *Broadcast-gather*,
    2. *Broadcast-reduce*,
2. *Scatter*,
    1. *Scatter-reduce*,
    2. *Scatter-gather*,
3. *Gather*,
4. *Reduce*,
5. *Multi-processing*,
6. *Multi-thread*,
7. *Point to point*.

Kemudian semua metode *parallel processing* ini akan dibandingkan *runtime*-nya dengan *serial processing*.

2.0.1 *Broadcast*

Merupakan metode komunikasi kolektif di mana satu proses mengirim data yang sama ke proses lainnya.

2.0.2 *Scatter*

Secara fungsi, *broadcast* dan *scatter* hanya memiliki perbedaan pada perintah yang digunakan. Selain itu *scatter* dapat mengirim potongan data dalam *array* ke proses yang berbeda. Perlu

diperhatikan bahwa jumlah *tasks* yang dibagikan tidak boleh melebihi julah *processors*.

2.0.3 *Gather*

Fungsi ini melakukan kebalikan dari *scatter*, yakni dengan mengumpulkan semua data yang diterima.

2.0.4 *Point to Point*

Operasi *point to point* (**P2P**) terdiri dari pertukaran pesan antara dua proses. Setiap operasi pengiriman akan disinkronkan secara sempurna dengan operasi penerimaan (*send - receive*).

2.1 Metode Integral Numerik

Ada beberapa metode numerik yang bisa digunakan untuk menghitung suatu integral dari fungsi kontinu. Pada praktikum ini, saya akan menggunakan metode titik tengah dan simulasi Monte Carlo untuk mengerjakan dua soal integral numerik.

2.1.1 Metode Titik Tengah (*Midpoint*)

Metode titik tengah merupakan salah satu cara perhitungan integral numerik dari fungsi kontinu melalui dikritisasi fungsi (MathLibretexts 2021). Prinsip yang digunakan adalah penjumlahan deret **Riemann**.

2.1.1.1 Definisi

Misalkan $f(x)$ kontinu di selang $[a, b]$. Jika diambil suatu n bilangan bulat positif, kita bisa membagi selang tersebut menjadi partisi-partisi kecil berikut: $\Delta x = \frac{b-a}{n}$.

Tuliskan $m_i = x_i + \frac{\Delta}{2}$ sebagai titik tengah di partisi ke- i, maka: $\int_a^b f(x)dx$ bisa didekati dengan $n \sum_{i=1}^{n} f(m_i)\Delta x_i$.

2.1.2 Simulasi Monte Carlo

Berbeda dengan metode titik tengah, ide dasar dari simulasi Monte Carlo adalah dengan melakukan *generating* sejumlah titik secara *random* pada selang integral dan menghitung nilai fungsinya[3]. Berikut adalah *flowchart*-nya:

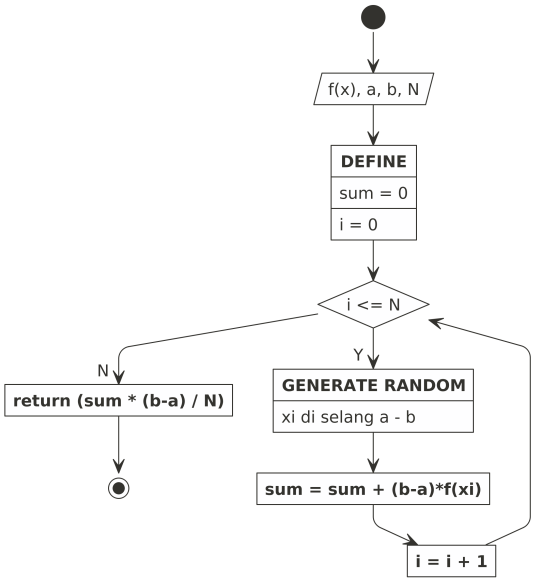

Figure 9: Flowchart Simulasi Monte Carlo untuk Integral Numerik

2.2 Menghitung π

Untuk menghitung nilai π pada soal kedua, saya akan menggunakan fungsi berikut:

$$\int_0^1 4\sqrt{1-x^2}$$

2.3 Penjumlahan dan Perkalian Matriks

Pada soal ketiga, saya menggunakan matriks berukuran $n \times n$ dengan $n = numCores^2 = 8^2 = 64$. Dengan isi dari matriks A dan B di-*generate* secara *random* melalui `numpy`.

[3] https://ikanx101.com/blog/integral-tentu/

3 *RESULT AND DISCUSSION*

Pada tugas praktikum ini, kita akan membandingkan *runtime* antara berbagai metode *parallel processing* dengan *serial processing*. Perlu saya ingatkan kembali bahwa *server Google Cloud* yang saya pakai memiliki **8 *cores***.

3.1 Soal I

Hitung integral $\int_0^1 x^2 dx$ dengan $n = 10^8$.

3.1.1 Jawab

Pada soal I ini saya menggunakan $n = 10^8$. Berikut adalah program *serial processing* yang dibuat:

```
import time
# hitung waktu mulai
mulai = time.time()
# bikin fungsi f(x)
def fx (x) :
   return(x**2)
# bikin fungsi untuk integral numerik
def int_numeric (a,b,n):
   # set terlebih dahulu agar tidak error pada tipe variabel
   n = int(n)
   a = float(a)
   b = float(b)
   sum = float(0)
   # hitung selang integrasi diksritisasi
   h = (b-a) / n
   # mulai iterasi untuk menghitung penjumlahan
   for i in range(n):
```

```
    xi = a + h/2 + i*h
    sum = sum + fx(xi)
  # kalikan dengan h untuk menjadi full integral
  sum = h * sum
  return(sum)
# initial condition
a = 0
b = 1
n = 10**8
# hitung soal
nilai = int_numeric(a,b,n)
print("Nilai integral f(x) dx adalah: ",nilai)
# hitung waktu selesai
end = time.time()
waktu = end - mulai
print(waktu)
# selesai
print("DONE")
```

Hasil *runtime* program di atas adalah `27.43908143043518` detik.

Berikut adalah rekap dan perbandingannya:

```
## Warning in Ops.factor(reorder(metode, runtime)): '-' not meaningful for factors
```

```
## Warning in Ops.factor(reorder(metode, runtime)): '-' not meaningful for factors
```

```
## Warning in Ops.factor(reorder(metode, runtime)): '-' not meaningful for factors
```

3.2 Soal II

Hitung nilai π dengan menyelesaikan: $\int_0^1 4\sqrt{1-x^2}$ dengan $n = 10^8$.

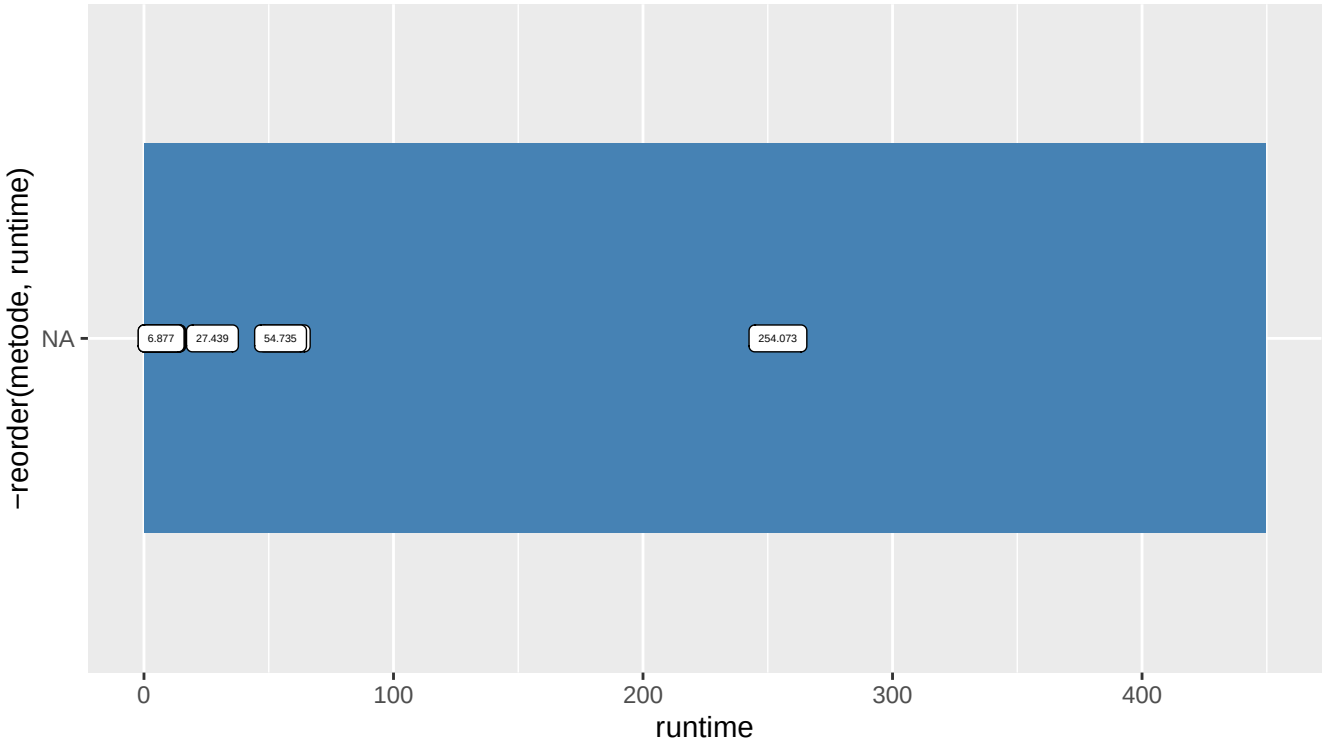

Figure 10: Rekap Runtime Soal I

3.3 Soal III

Buatlah program `Python` untuk menyelesaikan persoalan terkait dengan penjumlahan dan perkalian matriks.

4 *CONCLUSION*

REFERENCES

Jarno Rantaharju, Seyong Kim, and Ed Bennet. 2019. “Introduction to Mpi.” https://github.com/rantahar/introduction-to-mpi10.5281/zenodo.3274680.

MathLibretexts, Contributor. 2021. “Numerical Integration - Midpoint, Trapezoid, Simpson’s Rule.” https://math.libretexts.org/@go/page/10269.