

Laporan Pengerjaan Tugas Kelompok 2 Analisis Numerik

Deep Learning Optimizer



Disusun oleh Kelompok A2:

Raden Ahmad Yasin Mahendra (2306215154)

Ischika Afrilla (2306227955)





Yasmine Putri Viryadhani (2206081862)

Fabian Akmal Arkandion (2106750660)

Humam Al Labib (2206081755)

Pakta Integritas

Dengan ini, kami menyatakan bahwa tugas ini adalah hasil pekerjaan kelompok sendiri.

Fabian Akmal Arkandion	Ischika Afrilla	Yasmine Putri V.
		
Humam Al Labib	Raden Ahmad Yasin Mahendra	
		

RANGKUMAN

Tugas ini bertujuan untuk memahami dan membandingkan dua algoritma optimisasi, yaitu Adam (*Adaptive Moment Estimation*) dan Quasi-Newton, dalam menyelesaikan masalah optimisasi fungsi Rosenbrock, yaitu fungsi non-konveks yang umum digunakan sebagai *benchmark*.

Pertama, dilakukan kajian terhadap algoritma Adam, termasuk mekanisme pembaruan parameternya yang menggunakan estimasi momen pertama (rata-rata gradien) dan kedua (rata-rata kuadrat gradien), serta keunggulannya dalam hal kestabilan dan kecepatan konvergensi dibanding metode *first-order* lain seperti *Gradient Descent*.

Selanjutnya, algoritma Adam diimplementasikan dari awal dan digunakan untuk mengoptimasi fungsi Rosenbrock dalam berbagai dimensi ($N = 3, 7, 100, 500, 1000$). Analisis dilakukan untuk melihat pengaruh dimensi terhadap kinerja algoritma serta efektivitas penggunaan *hyperparameter* seperti *learning rate* dan *initial guess*. Kemudian, dikaji kemungkinan penerapan *line search* untuk membuat *learning rate* bersifat adaptif.

Selain itu, dirancang dan diimplementasikan algoritma Quasi-Newton buatan sendiri dengan pendekatan BFGS sebagai metode aproksimasi Hessian. Kompleksitas komputasi masing-masing algoritma dianalisis dalam satuan FLOPs.

Terakhir, dilakukan perbandingan eksperimental antara algoritma Adam dan Quasi-Newton berdasarkan laju konvergensi dan akurasi solusi, dengan mengukur *backward error* pada akhir iterasi. Hasil eksperimen digunakan untuk menentukan metode yang lebih unggul dalam menyelesaikan kasus optimisasi fungsi Rosenbrock.

DAFTAR ISI

1. PENDAHULUAN.....	1
1.1 Optimisasi Deep Learning.....	1
1.2 Adam (Adaptive Moment Estimation) Optimizer.....	1
1.3 Metode Quasi-Newton dan BFGS.....	2
2. PEMBAHASAN.....	3
2.1 Mekanisme Metode Optimisasi Adam.....	3
2.1.1 Algoritma Metode Adam.....	3
2.1.2 Keunggulan Metode Adam.....	5
2.2 Hasil Rancangan Metode Optimisasi Adam.....	6
2.2.1 Analisis Kompleksitas Metode Adam.....	6
2.2.2 Analisis Kompleksitas Metode Adam Berdasarkan Perhitungan.....	8
2.3 Metode Rosenbrock dengan Optimisasi Adam.....	10
2.3.1 Perhitungan Optimisasi Fungsi Rosenbrock dengan metode Adam.....	10
2.3.2 Analisis Optimisasi Fungsi Rosenbrock dengan metode Adam.....	11
2.4 Penggunaan Metode Line Search Untuk Learning Rate.....	12
2.4.1 Penggunaan Line Search di Metode Adam.....	12
2.4.2 Pengaruh Penggunaan Line Search di Metode Adam.....	12
2.5 Metode Quasi-Newton dengan aproksimasi Hessian BFGS (Broyden-Fletcher-Goldfarb-Shanno).....	13
2.5.1 Analisis Kompleksitas.....	13
2.6 Hasil Eksperimen dengan Metode Quasi-Newton Menggunakan Aproksimasi Hessian BFGS.....	15
3. KESIMPULAN.....	19
4. REFERENSI.....	20
LAMPIRAN.....	21

1. PENDAHULUAN

1.1 Optimisasi *Deep Learning*

Dalam bidang kecerdasan artifisial dan pembelajaran mesin, pilihan penggunaan algoritma optimisasi dapat berpengaruh pada kesuksesan suatu model atau agen dalam memprediksi variabel target. Proses optimisasi itu sendiri menjadi bagian inti dari proses pembelajaran model yang kompleks, *hyperparameter tuning*, dan performa sistem kecerdasan buatan dalam berbagai aplikasi, misalnya seperti *computer vision*, *neural network*, dan *reinforcement learning*. Pemilihan algoritma optimisasi ini juga dapat mempengaruhi laju konvergensi, akurasi, dan stabilitas proses pembelajaran.

Permasalahan optimisasi dalam *deep learning* dapat terbilang cukup rumit. Tahap proses pengembangan optimisasi *deep learning* dapat dibagi menjadi tiga tahap. Pertama, menjalankan algoritma sampai konvergen menuju solusi stabil yang dapat diterima seperti titik stasioner. Lalu, yang kedua adalah memastikan algoritma dapat konvergen secepat mungkin. Ketiga dan yang terakhir adalah memastikan konvergensi algoritma menuju ke solusi dengan nilai objektif yang rendah (*global minima*).

1.2 Adam (*Adaptive Moment Estimation*) Optimizer

Optimisasi dengan metode *stochastic gradient* merupakan bagian inti dari perkembangan sains dan teknologi saat ini. Pada fungsi objektif yang dapat diturunkan terhadap parameternya, *gradient descent* menjadi metode yang relatif efisien karena komputasi turunan parsial pertama terhadap semua parameternya memiliki kompleksitas perhitungan yang sama dengan mengevaluasi fungsi objektif itu sendiri.

Sebagai contoh, banyak jenis fungsi objektif yang terdiri atas penjumlahan subfungsi yang dievaluasi pada sub sampel data yang berbeda. Maka, pada kasus ini, optimisasi akan lebih efisien dengan cara melakukan *gradient step* terhadap masing-masing subfungsi. SGD (*Stochastic Gradient Descent*) sudah terbukti menjadi metode yang efektif dan efisien untuk optimisasi fungsi objektif dalam berbagai kasus pembelajaran mesin, seperti contohnya pada kemajuan di bidang *deep learning*. Fungsi objektif mungkin saja memiliki sumber *noise* selain dari *subsampling* data seperti *dropout regularization* (Hinton et al., 2012). Untuk fungsi objektif yang memiliki banyak *noise*, teknik optimisasi stokastik yang efisien sangat diperlukan. Pada kasus optimisasi fungsi objektif stokastik dengan parameter berdimensi banyak, metode dengan *high-order optimization* tidak sesuai digunakan sehingga dibutuhkan metode *first-order optimization*.

Adam (*Adaptive Moment Estimation*), merupakan metode optimisasi stokastik yang efisien dengan cukup menggunakan gradien pertama dengan kebutuhan memori yang kecil. Metode ini menghitung *learning rate* adaptif secara individu untuk parameter yang berbeda-beda dari hasil estimasi momen gradien pertama dan kedua. Metode ini didesain dari kombinasi antara dua metode, yaitu AdaGrad (Duchi et al., 2011) yang unggul ketika berhadapan dengan *sparse gradient* dan RMSProp (Tieleman dan Hinton, 2012) yang unggul

di *on-line* dan *stationary setting*. Kelebihan penggunaan metode Adam adalah bahwa banyaknya *update* pada parameter tidak berubah terhadap *gradient rescaling*, ukuran *step* dibatasi oleh hyperparameter *stepsize*, tidak membutuhkan fungsi objektif yang stasioner, dapat digunakan untuk gradien yang *sparse*, dan secara natural melakukan *step size annealing* (pendekatan metaheuristik untuk aproksimasi global optimum pada *search space* yang luas).

1.3 Metode Quasi-Newton dan BFGS

Metode optimisasi Newton adalah salah satu metode optimisasi *unconstrained* yang menggunakan turunan kedua dari fungsi objektif, berbeda halnya dengan *gradient descent* yang hanya menggunakan turunan pertama. Pada tiap iterasi ke- k , metode ini mengaproksimasi fungsi f pada titik x_k dengan sebuah paraboloid, lalu meminimalisasikan hasil aproksimasi dengan melakukan *step* ke titik minimum pada paraboloid. Dengan begitu, metode Newton dapat konvergen lebih cepat menuju titik minimum. Namun, kekurangan dari metode ini adalah komputasinya yang mahal karena harus menghitung matriks Hessian dan inversnya. Hal ini menjadi masalah yang lebih besar lagi ketika dimensi matriks makin membesar. Untuk itu, metode Quasi-Newton dibuat sebagai solusi terhadap masalah tersebut.

Alhasil, Quasi-Newton dipilih menjadi metode optimisasi yang memanfaatkan (aproksimasi) sifat turunan kedua agar dapat konvergen lebih cepat tanpa menghitung nilai eksak turunan kedua. Pada tiap iterasi ke- $(k+1)$, metode ini membuat aproksimasi invers turunan kedua dengan hanya menggunakan nilai pada iterasi sebelumnya, yaitu turunan pertama dari dua iterasi terakhir (k dan $k-1$).

Dalam satu dimensi, aproksimasi matriks Hessian pada metode Quasi-Newton hanya mengaproksimasi turunan kedua dan menggantikannya dengan aproksimasi selisih *finite* atau juga disebut dengan metode *secant*. Pada dimensi yang lebih dari satu, kondisi Quasi-Newton tidak menspesifikasikan estimasi Hessian B dan diperlukan *constraint* pada aproksimasi B untuk menyelesaikannya. Salah satu metode populer yang dapat menyelesaikan permasalahan ini adalah BFGS yang diambil dari akronim nama penemunya (Broyden, Fletcher, Goldfarb, dan Shanno) yang masing-masing menemukan algoritma tersebut secara independen pada tahun 1970 (Broyden, 1970; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970). Dengan metode BFGS, kita melakukan *update* pada nilai B^{-1} hanya dengan menggunakan nilai dari $\delta x_k = x_{k+1}$ dan $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$ dari iterasi sebelumnya yang bersesuaian dengan formula update aproksimasi Hessian B

$$B_{k+1}^{-1} = \left(I - \frac{\Delta x y^\top}{y^\top \Delta x} \right) B^{-1} \left(I - \frac{y \Delta x^\top}{y^\top \Delta x} \right) + \frac{\Delta x \Delta x^\top}{y^\top \Delta x}$$

Dengan mengestimasi langsung invers Hessian pada tiap iterasi, kita menghindari operasi invers matriks Hessian yang memiliki kompleksitas $O(n^3)$ seperti pada metode Newton.

2. PEMBAHASAN

2.1 Mekanisme Metode Optimisasi Adam

Metode Adam (*Adaptive Moment Estimation*) merupakan salah satu metode *optimizer* yang paling banyak digunakan dalam berbagai model *deep learning*. Metode ini pertama kali diperkenalkan oleh Diederik P. Kingma dan Jimmy Lei Ba pada tahun 2014. Metode Adam menggunakan jumlah nilai gradien dikalikan dengan bobot yang dihitung sebelumnya, yang merupakan gagasan momentum. Hasil perhitungan tersebut disebut momentum pertama. Kemudian, jumlah kuadrat gradien dihitung dengan cara yang sama, menghasilkan momentum kedua. Pada akhirnya, rasio nilai momentum pertama dan kedua dihitung dan nilai minimum dicari menurut rasio tersebut.

2.1.1 Algoritma Metode Adam

Metode Adam dijalankan dengan langkah-langkah algoritma sebagai berikut:

1. Input Awal dan Inisialisasi Variabel

Metode Adam memiliki beberapa input, yaitu γ sebagai *learning rate*, dua hiperparameter berupa β_1 dan β_2 , θ_0 sebagai parameter, $f(\theta)$ sebagai fungsi objektif, λ sebagai *weight decay*, variabel boolean *amsgrad* sebagai penanda ada tidaknya AMSGrad, dan variabel boolean *maximize*.

Sebelum memulai algoritma ini, beberapa variabel juga harus diinisialisasi ke 0. Variabel tersebut terdiri dari m_0 sebagai momen pertama, v_0 sebagai momen kedua, serta $\widehat{v_0}^{max}$ sebagai nilai maksimum dari momen kedua.

2. Komputasi Gradien (g_t)

Untuk setiap iterasi t , gradien dari fungsi objektif $f(\theta)$ akan dihitung terhadap parameter $g_t \leftarrow -\nabla_{\theta} f_t(\theta_{t-1})$ jika *maximize* atau parameter $g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$ jika tidak *maximize*.

3. Penambahan Nilai *Weight Decay* (λ) ke Gradien

Jika nilai *weight decay* $\lambda \neq 0$, kita perlu menambahkan nilai *weight decay* ke gradien yang telah dikomputasi, yaitu dengan proses $g_t \leftarrow -g_t + \lambda \theta_{t-1}$. Proses ini merupakan bentuk regularisasi yang digunakan untuk mencegah *overfitting* pada algoritma yang dijalankan.

4. Pembaruan Momen Pertama (m_t)

Momen pertama diperbarui sebagai rata-rata bergerak eksponensial dari gradien dengan rumus $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1)g_t$. Parameter β_1 digunakan untuk mengontrol seberapa besar pengaruh gradien sebelumnya (m_{t-1}) dibandingkan dengan gradien saat ini (g_t).

5. Pembaruan Momen Kedua (v_t)

Momen kedua diperbarui sebagai rata-rata bergerak eksponensial dari gradien kuadrat (g_t^2) dengan rumus $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2)g_t^2$. Parameter β_2 digunakan untuk mengontrol seberapa besar pengaruh gradien kuadrat sebelumnya.

6. Koreksi Bias untuk Momen Pertama dan Kedua

Momen pertama dan kedua perlu dikoreksi karena nilai estimasi awal dari m_t dan v_t cenderung bias ke arah nol pada iterasi awal. Koreksi bias momen pertama \hat{m}_t dicari menggunakan rumus $\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$, sedangkan koreksi bias momen kedua \hat{v}_t dicari menggunakan rumus $\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$.

7. Pembaruan Parameter θ_t

Algoritma akan mengembalikan hasil pembaruan parameter θ_t , yaitu parameter hasil optimisasi Adam. Nilai parameter θ_t dapat dicari dengan beberapa rumus sesuai nilai boolean *amsgrad* yang tergantung pada ada tidaknya AMSGrad. Secara umum, perhitungan θ_t juga menggunakan standar epsilon ϵ untuk mencegah pembagian dengan nol.

Jika terdapat AMSGrad, perhitungan dilakukan dengan mencari nilai maksimum dari \hat{v}_t yaitu \hat{v}_t^{max} terlebih dahulu di setiap iterasinya untuk mencegah \hat{v}_t mengecil terlalu cepat. Metode perhitungan dengan *amsgrad* dilakukan sebagai berikut:

$$\hat{v}_t^{\max} \leftarrow \max(\hat{v}_t^{\max}, \hat{v}_t)$$

$$\theta_t \leftarrow \theta_{t-1} - \gamma \frac{\hat{m}_t}{\sqrt{\hat{v}_t^{\max} + \epsilon}}$$

Apabila tidak terdapat AMSGrad, perhitungan dilakukan dengan menggunakan standar Adam, seperti berikut:

$$\theta_t \leftarrow \theta_{t-1} - \gamma \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}$$

2.1.2 Keunggulan Metode Adam

Terdapat beberapa alasan di balik Metode Adam yang lebih *robust* dibandingkan dengan algoritma *first-order* lain:

1. Adaptasi Langkah untuk Setiap Parameter

Metode Adam memiliki penyesuaian untuk setiap parameter dengan cara mengoreksi bias momen pertama dan kedua. Hal ini dilakukan untuk mencegah *overshooting* dan mempercepat konvergensi algoritma.

2. Kombinasi Momentum dengan Gradien

Metode Adam menggabungkan momentum dengan skala gradien sehingga membuat metode ini lebih stabil ketika digunakan dalam daerah dengan gradien yang bervariasi, contohnya pada fungsi Rosenbrock yang memiliki lembah yang sempit.

3. Koreksi Bias

Seperti yang dijelaskan pada poin pertama, terdapat koreksi bias momen pertama dan kedua yang membuat langkah-langkah pengerjaan algoritma menjadi lebih akurat dan tidak bias ke nol.

4. Stabilitas terhadap Hiperparameter

Penggunaan hiperparameter β_1 dan β_2 memastikan perhitungan yang dilakukan menjadi adaptif dan stabil.

5. Efisiensi Komputasi

Metode Adam memiliki konvergensi yang lebih cepat dari metode *first-order* yang lain karena terdapat adaptasi untuk setiap parameternya.

2.2 Hasil Rancangan Metode Optimisasi Adam

Sebuah rancangan metode Adam dapat dibuat dalam Octave berdasarkan mekanisme metode optimisasi Adam yang telah dijelaskan pada poin sebelumnya. Misalkan inisialisasi metode Adam dilakukan tanpa AMSGrad dan tidak *maximize*, serta dengan nilai variabel-variabel awal $\gamma = 0.1$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\theta_0 = 10$, $\lambda = 0$, didapatkan nilai parameter θ_t yang dicari dengan jumlah iterasi $T = 100$ yaitu sekitar 2.2445.

```
>> adam_optimizer
warning: function 'adam_optimizer' defined within script file '/Users/yasmineputri/Downloads/tk2_anum/adam_optimizer.m'
Parameter initial:
theta0 = 10
gamma = 0.1
beta1 = 0.9
lambda = 0
amsgrad = false
maximize = false
T = 100
epsilon = 1e-8
Optimized theta: 2.2445
```

Gambar 2.2.1: Hasil optimisasi Adam menggunakan Octave

2.2.1 Analisis Kompleksitas Metode Adam

Kompleksitas metode Adam dapat dihitung berdasarkan tahapan-tahapan yang telah dijelaskan sebelumnya. Analisis kompleksitas metode Adam diukur dalam jumlah operasi *floating point* (FLOPs).

1. Komputasi Gradien

Tahapan ini terdiri dari fungsi pencarian gradien untuk fungsi objektif $f(\theta)$. Baik parameter $g_t \leftarrow -\nabla_{\theta} f_t(\theta_{t-1})$ (digunakan jika *maximize*) ataupun parameter $g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$ (jika tidak *maximize*) sama-sama merupakan operasi perkalian.

Evaluasi fungsi: 1 FLOPs (1 perkalian).

Total FLOPs: $O(N)$.

2. Kondisi *Maximize*

Jika perhitungan yang dilakukan tidak memiliki nilai *maximize*, nilai FLOPs tahap ini menjadi nol atau $O(0N)$. Namun, jika terdapat *maximize*, parameter negasi $g_t \leftarrow -g_t$ digunakan.

Evaluasi fungsi:

- 0 FLOPs (jika tidak *maximize*).
- 1 FLOPs (1 negasi, jika *maximize*).

Total FLOPs:

- $O(0N)$ jika tidak *maximize*.
- $O(N)$ jika *maximize*.

3. *Weight Decay*

Jika perhitungan yang telah dilakukan tidak memiliki *weight decay*, nilai FLOPs tahap ini menjadi nol atau $O(0N)$. Namun, jika terdapat *weight decay*, operasi $g_t \leftarrow -g_t + \lambda \theta_{t-1}$ dilakukan.

Evaluasi fungsi:

- 0 FLOPs (jika tidak ada *weight decay*).
- 2 FLOPs (1 penjumlahan dan 1 perkalian, jika ada *weight decay*).

Total FLOPs:

- $O(0N)$ jika tidak ada *weight decay*.
- $O(2N)$ jika ada *weight decay*.

4. Pembaruan Momen Pertama (m_t)

Tahapan ini menggunakan rumus $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$.

Evaluasi fungsi: 3 FLOPs (2 perkalian dan 1 penjumlahan).

Total FLOPs: $O(3N)$.

5. Pembaruan Momen Kedua (v_t)

Tahapan ini menggunakan rumus $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$.

Evaluasi fungsi: 4 FLOPs (3 perkalian dan 1 penjumlahan).

Total FLOPs: $O(4N)$.

6. Koreksi Bias Momen Pertama (\hat{m}_t)

Tahapan ini menggunakan rumus $\hat{m}_t \leftarrow \frac{m_t}{1 - \beta_1^t}$.

Evaluasi fungsi: 3 FLOPs (1 eksponen, 1 pengurangan, 1 pembagian).

Total FLOPs: $O(3N)$.

7. Koreksi Bias Momen Kedua (\hat{v}_t)

Tahapan ini menggunakan rumus $\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$.

Evaluasi fungsi: 3 FLOPs (1 eksponen, 1 pengurangan, 1 pembagian).

Total FLOPs: $O(3N)$.

8. Pembaruan parameter θ_t

Evaluasi fungsi: 5 FLOPs (1 akar kuadrat, 1 penjumlahan, 1 pembagian, 1 perkalian, 1 pengurangan, jika tidak memakai *amsgrad*).

- 6 FLOPs (1 operasi *max*, 1 akar kuadrat, 1 penjumlahan, 1 pembagian, 1 perkalian, 1 pengurangan, jika memakai *amsgrad*).

Total FLOPs:

- $O(5N)$ jika tidak memakai *amsgrad*.
- $O(6N)$ jika memakai *amsgrad*.

2.2.2 Analisis Kompleksitas Metode Adam Berdasarkan Perhitungan

Berdasarkan contoh perhitungan yang telah dilakukan beserta perhitungan kompleksitas yang telah dijabarkan, jika digunakan parameter $\theta_0 = 10$ sebagai skalar dan jumlah iterasi $T = 100$, maka berikut merupakan perhitungan kompleksitasnya:

1. Komputasi Gradien

Evaluasi fungsi: 1 FLOPs (1 perkalian).

Total FLOPs: $O(N)$.

2. Kondisi *Maximize* (*maximize* = false)

Evaluasi fungsi: 0 FLOPs karena tidak *maximize*.

Total FLOPs: $O(0N)$ karena tidak *maximize*.

3. *Weight Decay* ($\lambda = 0$)

Evaluasi fungsi: 0 FLOPs karena tidak ada *weight decay*.

Total FLOPs: $O(0N)$ karena tidak ada *weight decay*.

4. Pembaruan Momen Pertama (m_t)

Evaluasi fungsi: 3 FLOPs (2 perkalian dan 1 penjumlahan).

Total FLOPs: $O(3N)$.

5. Pembaruan Momen Kedua (v_t)

Evaluasi fungsi: 4 FLOPs (3 perkalian dan 1 penjumlahan).

Total FLOPs: $O(4N)$.

6. Koreksi Bias Momen Pertama (\hat{m}_t)

Tahapan ini menggunakan rumus $\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$.

Evaluasi fungsi: 3 FLOPs (1 eksponen, 1 pengurangan, 1 pembagian).

Total FLOPs: $O(3N)$.

7. Koreksi Bias Momen Kedua (\hat{v}_t)

Tahapan ini menggunakan rumus $\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$.

Evaluasi fungsi: 3 FLOPs (1 eksponen, 1 pengurangan, 1 pembagian).

Total FLOPs: $O(3N)$.

8. Pembaruan parameter θ_t (*amsgrad* = *false*)

Evaluasi fungsi: 5 FLOPs (1 akar kuadrat, 1 penjumlahan, 1 pembagian, 1 perkalian, 1 pengurangan) karena tidak memakai *amsgrad*.

Total FLOPs: $O(5N)$ karena tidak memakai *amsgrad*.

Dari hasil perhitungan kompleksitas untuk setiap tahapannya, kemudian dilakukan penjumlahan untuk mencari total kompleksitas untuk satu iterasi. Jumlah

FLOPs per iterasi yang didapatkan sebesar $1 + 0 + 0 + 3 + 4 + 3 + 3 + 5 = 19$ FLOPs atau dalam kata lain, besar kompleksitas untuk satu iterasi adalah $O(19N)$. Apabila kita kalikan besar kompleksitas tersebut dengan jumlah iterasi yang telah dilakukan, yaitu $T = 100$, total kompleksitas perhitungan menjadi $19 \times 100 = 1900$ FLOPs atau $19NT$ FLOPs, dengan notasi kompleksitas sebesar $O(19NT)$.

2.3 Metode Rosenbrock dengan Optimisasi Adam

Kita dapat menggunakan metode Adam untuk mengoptimisasi berbagai fungsi, salah satunya adalah fungsi Rosenbrock. Fungsi Rosenbrock didefinisikan sebagai

$$f(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2]$$

di mana $x = (x_1, x_2, \dots, x_n) \in \mathbb{R}^n$. Fungsi ini merupakan fungsi *non-convex* yang memiliki lembah yang sempit, dengan *global minimum* di $x_i = 1$ untuk semua i , di mana $f(x) = 0$. Untuk mengoptimisasi fungsi Rosenbrock dengan optimisasi Adam, kita perlu mencari gradiennya terlebih dahulu, kemudian perhitungan metode Adam baru dapat dilakukan.

2.3.1 Perhitungan Optimisasi Fungsi Rosenbrock dengan metode Adam

Sebuah perhitungan optimisasi fungsi Rosenbrock menggunakan metode Adam dengan ukuran vektor $x \in \mathbb{R}^N$ yang bervariasi $N = [3, 7, 100, 500, 1000]$ dilakukan. Misalkan inisialisasi metode Adam dilakukan tanpa AMSGrad dan tidak *maximize*, serta dengan nilai variabel-variabel awal $\gamma = 0.1, \beta_1 = 0.9, \beta_2 = 0.999, \theta_0 = 10, \lambda = 0$, didapatkan nilai parameter θ_t untuk setiap ukuran vektor dengan jumlah iterasi $T = 10000$ yaitu sebagai berikut:

```
Hyperparameter initial:
theta0 = zeros(N, 1)
gamma = 0.001
beta1 = 0.9
beta2 = 0.999
lambda = 0
amsgrad = false
maximize = false
T = 10000
epsilon = 1e-8

N = 3
Nilai akhir fungsi: 0.000000
Jarak ke true minimum: 0.000000
Elapsed time: 2.6547 seconds
---
N = 7
Nilai akhir fungsi: 0.000000
Jarak ke true minimum: 0.000000
Elapsed time: 4.2238 seconds
---
N = 100
Nilai akhir fungsi: 0.000090
Jarak ke true minimum: 0.000317
Elapsed time: 50.4228 seconds
---
N = 500
Nilai akhir fungsi: 249.938453
Jarak ke true minimum: 15.758964
Elapsed time: 193.9439 seconds
---
N = 1000
Nilai akhir fungsi: 744.880603
Jarak ke true minimum: 27.169512
Elapsed time: 254.4483 seconds
---
```

Gambar 2.3.1.1: Hasil optimisasi fungsi Rosenbrock dengan metode Adam di Octave

2.3.2 Analisis Optimisasi Fungsi Rosenbrock dengan metode Adam

Berdasarkan optimisasi fungsi Rosenbrock yang telah dilakukan sebelumnya, analisis terhadap ukuran vektor x (yang ditentukan oleh dimensi N) yang dapat memengaruhi performa metode Adam dalam mengoptimisasi fungsi Rosenbrock.

1. Konvergensi pada Dimensi Rendah ($N = 3$ dan $N = 7$)

Hasil:

- Nilai akhir fungsi: 0.000000
- Jarak ke *global minimum*: 0.000000

Analisis:

- Pada dimensi rendah, metode Adam berhasil mencapai minimum global fungsi Rosenbrock, yaitu $x_i = 1$ untuk semua i , di mana $f(x) = 0$.
 - Jumlah variabel yang kecil memungkinkan algoritma untuk menavigasi lembah Rosenbrock dengan lebih mudah.
 - Kompleksitas ruang pencarian rendah, sehingga gradien dapat mengarahkan optimisasi ke minimum dengan cepat.
- #### 2. Konvergensi pada Dimensi Sedang ($N = 100$)

Hasil:

Nilai akhir fungsi: 0.000090

Jarak ke *global minimum*: 0.0000317

Analisis:

- Pada dimensi sedang, metode Adam mendekati *global minimum*, walau tidak seakurat dimensi kecil.
 - Peningkatan dimensi mulai memengaruhi konvergensi karena lembah menjadi lebih panjang dan sempit, sehingga metode Adam mulai melambat dan memerlukan lebih banyak iterasi untuk sampai ke *global minimum*.
- #### 3. Konvergensi pada Dimensi Tinggi ($N = 500$ dan $N = 1000$)

Hasil:

Nilai akhir fungsi:

- Untuk $N = 500$: 249.938453
- Untuk $N = 1000$: 744.880603

Jarak ke *global minimum*:

- Untuk $N = 500$: 15.758964
- Untuk $N = 1000$: 27.169512

Analisis:

- Pada dimensi tinggi, performa metode Adam menurun drastis karena solusi dari proses optimisasi fungsi belum mendekati *global minimum*.
- Dimensi yang tinggi memengaruhi konvergensi karena lembah menjadi lebih panjang dan sempit, sehingga metode Adam menjadi sangat lambat dan memerlukan jauh lebih banyak iterasi untuk sampai ke *global minimum*.

2.4 Penggunaan Metode Line Search Untuk Learning Rate α

Metode *line search* dapat digunakan untuk membuat *learning rate* α pada optimisasi Adam menjadi adaptif. Secara tradisional, metode Adam menggunakan *learning rate* yang bersifat konstan. Namun, dengan *line search*, α_t dapat ditentukan secara berbeda untuk setiap iterasinya.

2.4.1 Penggunaan Line Search di Metode Adam

Metode Adam akan menghitung arah d seperti biasa menggunakan momen pertama dan kedua. Kemudian, sebuah fungsi $\phi(a) = f(\theta_t + \alpha d_t)$ didefinisikan. Lalu, gunakan kondisi seperti Armijo-Goldstein Rules untuk memastikan penurunan yang cukup, yaitu $f(\theta_t + \alpha_t d_t) \leq f(\theta_t) + c_t \alpha_t \nabla f(\theta_t)^T d_t$ dengan $c_1 \in (0, 1)$. Jika kondisi tidak terpenuhi, kurangi α_t hingga kondisi terpenuhi. Pada akhirnya, gunakan α_t yang telah ditemukan untuk memperbarui parameter θ_t , dengan rumus $\theta_{t+1} = \theta_t + \alpha_t d_t$.

2.4.2 Pengaruh Penggunaan Line Search di Metode Adam

Penggunaan *line search* dalam metode Adam memiliki kelebihan dan kekurangan. Namun, di kebanyakan kasus, *Line search* cenderung lebih baik terutama untuk fungsi *non-convex* seperti fungsi Rosenbrock, di mana langkah adaptif dapat mempercepat konvergensi dan mengatasi masalah langkah konstan yang terlalu kecil. Tetapi, biaya komputasi penggunaan *line search* akan menjadi tinggi apabila dilakukan pada dimensi yang tinggi.

2.5 Metode Quasi-Newton dengan aproksimasi Hessian BFGS (Broyden-Fletcher-Goldfarb-Shanno)

Dalam masalah optimisasi, metode Newton memiliki beberapa keunggulan, contohnya yaitu cepat ketika tebakan awal mendekati nilai eksak. Namun, kelemahan dari metode ini adalah membutuhkan seluruh nilai turunan parsial kedua yang sulit untuk dihitung dan membutuhkan $O(n^3)$ untuk menghitung matriks Hessian dan menyimpannya dengan kompleksitas *space* sebesar $O(n^2)$. Untuk itu, maka muncullah metode optimisasi baru yang diberi nama Quasi-Newton.

Metode Quasi-Newton adalah salah satu metode yang dapat digunakan untuk mengoptimisasi fungsi nonlinear secara *unconstrained*. Metode ini melakukan *line search* secara sekuensial dengan komputasi f dan $\sim f$ pada tiap iterasi. Hal yang membedakan metode ini dengan metode lainnya adalah cara menentukan arah pencarian. Pada iterasi ke- k , metode ini akan melakukan komputasi matriks H_k yang merupakan aproksimasi dari invers matriks Hessian $H^{-1}(x_k)$. Dengan aproksimasi matriks Hessian, kita tidak perlu menyelesaikan sistem persamaan linear $H(x_k)h_k = -\nabla f(x_k)$ dan hanya perlu mengalikan gradien dengan aproksimasi invers Hessian untuk menghasilkan arah pencarian $h_k = -H_k \nabla f(x_k)$.

2.5.1 Analisis Kompleksitas

Pertama-tama, kita dapat membagi algoritma optimasi Quasi-Newton dengan BFGS menjadi 4 bagian, yaitu:

1. Evaluasi Fungsi Rosenbrock dan Gradien

Pada fungsi rosenbrock_n, untuk dimensi n:

$$f(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2]$$

Fungsi dan gradien dihitung dengan *loop* dari 1 sampai n-1, dimana tiap iterasi *loop* akan menjalankan:

- Evaluasi fungsi: 9 FLOPs (3 perpangkatan, 1 perkalian, 2 pengurangan, 3 penjumlahan)
- Gradien (turunan pertama dan kedua): 12 FLOPs (2 perpangkatan, 4 perkalian, 5 pengurangan, 1 penjumlahan)

Jadi, total FLOPs nya adalah $O(21n) \sim O(n)$

2. Komputasi Arah Pencarian

Arah pencarian didapat melalui formula berikut:

$$p = -H \cdot \nabla f$$

Kita melakukan perkalian matriks-vektor dengan $H \in \mathbb{R}^{n \times n}$ dan $\nabla f(x) \in \mathbb{R}^n$ sehingga total flops menjadi $O(n^2)$

3. *Backtracking Line Search (Armijo's rule)*

Pertama kita melakukan uji kondisi Armijo (dengan kompleksitas konstan) berikut:

$$f(x + \alpha p) \leq f(x) + c\alpha \nabla f^T p$$

Lalu, evaluasi fungsi setiap kali α gagal karena terlalu kecil sehingga *worst case* nya menjadi $O(mn)$ (dengan m = jumlah evaluasi *line search*).

4. Update matriks H dengan BFGS

Langkah-langkah dalam aproksimasi matriks Hessian dengan BFGS adalah sebagai berikut:

$$y = \nabla f(x_{k+1}) - \nabla f(x_k) : (\mathcal{O}(n))$$

$$s = x_{k+1} - x_k : (\mathcal{O}(n))$$

$$\rho = \frac{1}{y^\top s} : (\mathcal{O}(n))$$

dan yang terakhir adalah mengupdate matriks Hessian H:

$$H_{k+1} = (I - \rho s y^\top) H (I - \rho y s^\top) + \rho s s^\top$$

yang dimana membutuhkan perkalian matriks $\mathcal{O}(n^2)$

Jadi, total kompleksitas FLOPs yang mendominasi adalah $O(n^2)$

Dari keempat langkah di atas, dapat disimpulkan bahwa total kompleksitas FLOPs adalah $O(n^2) + O(mn) + O(n)$ dan untuk $m \ll n$, maka dapat dianggap bahwa algoritma Quasi Newton dengan aproksimasi Hessian menggunakan BFGS memiliki kompleksitas $O(n^2)$.

2.6 Hasil Eksperimen dengan Metode Quasi-Newton Menggunakan Aproksimasi Hessian BFGS

Dengan menjalankan algoritma optimasi untuk $N = [3, 7, 100, 500, 100]$ seperti pada metode sebelumnya, didapat hasil seperti berikut:

```
Optimisasi fungsi Rosenbrock dengan Quasi Newton BFGS untuk N = 3
Konvergen pada N=3 dalam 24 iterasi
Iterasi: 24
Nilai x:
  1
  1
  1
Nilai grad f(x):
-5.1845e-08
 3.4943e-08
-5.5191e-09
Waktu komputasi: 0.01 detik

Estimasi laju konvergensi: 1.12
```

Gambar 2.6.1: Optimisasi fungsi Rosenbrock untuk $N = 3$

```
Optimisasi fungsi Rosenbrock dengan Quasi Newton BFGS untuk N = 7
Konvergen pada N=7 dalam 45 iterasi
Iterasi: 45
Nilai x:
  1
  1
  1
  1
  1
  1
  1
  1
Nilai grad f(x):
-1.1155e-07
 1.3912e-07
 6.4831e-08
-5.0988e-07
 3.6075e-07
-2.5968e-07
 9.0153e-08
Waktu komputasi: 0.03 detik

Estimasi laju konvergensi: 0.99
```

Gambar 2.6.2: Optimisasi fungsi Rosenbrock untuk $N = 7$

```

Optimisasi fungsi Rosenbrock dengan Quasi Newton BFGS untuk N = 100
Konvergen pada N=100 dalam 211 iterasi
Iterasi: 211
Nilai x:
  1
  1
  1
  1
  1
  1
  1
  1
  1
  1
  1
  1
Nilai grad f(x):
  9.6654e-08
  1.3652e-07
 -9.0109e-08
  1.2194e-07
 -2.2154e-07
  1.1755e-07
  5.2310e-08
 -8.1707e-08
  6.6165e-08
  2.7710e-08
Waktu komputasi: 2.84 detik

Estimasi laju konvergensi: 0.94

```

Gambar 2.6.3: Optimisasi fungsi Rosenbrock untuk $N = 100$ (nilai x dan $\text{grad } f(x)$ di-truncate sampai 10 nilai pertama)

```

Optimisasi fungsi Rosenbrock dengan Quasi Newton BFGS untuk N = 500
Konvergen pada N=500 dalam 585 iterasi
Iterasi: 585
Nilai x:
  1
  1
  1
  1
  1
  1
  1
  1
  1
  1
  1
  1
Nilai grad f(x):
 -9.2207e-09
 -5.5803e-08
  7.2613e-08
 -2.9354e-08
  1.7908e-08
  1.2981e-08
  1.5690e-08
 -3.2384e-08
 -6.4275e-08
 -7.4623e-08
Waktu komputasi: 60.40 detik

Estimasi laju konvergensi: 1.21

```

Gambar 2.6.4: Optimisasi fungsi Rosenbrock untuk $N = 500$ (nilai x dan $\text{grad } f(x)$ di-truncate sampai 10 nilai pertama)

```

Optimisasi fungsi Rosenbrock dengan Quasi Newton BFGS untuk N = 1000
Konvergen pada N=1000 dalam 1247 iterasi
Iterasi: 1247
Nilai x:
  1
  1
  1
  1
  1
  1
  1
  1
  1
  1
  1
  1
Nilai grad f(x):
  6.4674e-09
  5.4938e-08
 -2.1906e-08
  3.9505e-08
 -1.1564e-08
 -7.3017e-08
 -1.3440e-08
  4.2439e-08
  4.6466e-08
 -1.1650e-07
Waktu komputasi: 308.65 detik

Estimasi laju konvergensi: 0.97

```

Gambar 2.6.5: Optimisasi fungsi Rosenbrock untuk $N = 1000$ (nilai x dan $\text{grad } f(x)$ di-truncate sampai 10 nilai pertama)

Program berhasil berjalan untuk mengoptimisasi Fungsi Rosenbrock yang ditandai dengan nilai $\nabla f(x)$ yang sangat kecil dan sudah mendekati nol pada tiap N yang digunakan. Dari hasil di atas, kita juga bisa menyimpulkan bahwa algoritma Quasi Newton dengan BFGS ini memiliki laju konvergensi $c \approx 1$ yang berarti laju konvergensi nya linear.

Laju konvergensi ini dihitung dengan formula seperti berikut:

$$c \approx \frac{\log\left(\frac{\eta_k}{\eta_{k-1}}\right)}{\log\left(\frac{\eta_{k-1}}{\eta_{k-2}}\right)}$$

dengan $\eta_k = \|\nabla f(x_k)\|$ merupakan *backward error* pada iterasi ke- k .

3. KESIMPULAN

Berdasarkan hasil implementasi dan eksperimen yang dilakukan terhadap algoritma Adam dan Quasi-Newton (BFGS) dalam mengoptimasi fungsi Rosenbrock, dapat disimpulkan beberapa hal sebagai berikut:

1. Adam merupakan algoritma optimisasi berbasis *first-order* yang unggul dalam hal kecepatan konvergensi awal dan stabilitas pembaruan parameter, terutama pada dimensi rendah hingga menengah. Penggunaan momen pertama dan kedua secara adaptif membuatnya lebih *robust* terhadap *noise* dan fluktuasi gradien.
2. Ukuran dimensi vektor x berpengaruh signifikan terhadap performa optimisasi. Seiring meningkatnya dimensi, dibutuhkan lebih banyak iterasi dan waktu komputasi untuk mencapai konvergensi, terutama bagi algoritma berbasis *first-order* seperti Adam.
3. Penggunaan *line search* sebagai pengganti *learning rate* tetap pada Adam dapat meningkatkan akurasi dan stabilitas konvergensi, namun dengan tambahan beban komputasi. Hasilnya bersifat kontekstual dan perlu disesuaikan dengan karakteristik fungsi dan sumber daya.
4. Metode Quasi-Newton (BFGS) menunjukkan laju konvergensi yang lebih cepat dan lebih akurat pada dimensi tinggi dibanding Adam, berkat pemanfaatan informasi kurvatur melalui aproksimasi Hessian.
5. Berdasarkan analisis *backward error*, Quasi-Newton dengan BFGS lebih unggul dari segi presisi hasil akhir pada dimensi tinggi, sedangkan pada dimensi kecil sampai menengah, performa kedua metode tersebut dapat dibilang sama. Quasi-Newton dengan BFGS lebih efisien untuk dimensi tinggi karena skalabilitas dan kompleksitas komputasinya yang lebih rendah.

Secara keseluruhan, pilihan algoritma terbaik sangat tergantung pada karakteristik masalah dan ukuran data. Akan tetapi, secara umum, metode Quasi-Newton cocok untuk permasalahan dengan dimensi berapapun (terlebih pada dimensi berskala besar) dengan waktu komputasi yang jauh lebih singkat daripada Adam dengan aproksimasi Hessian menggunakan BFGS. Sementara itu, metode Adam cocok untuk masalah dengan dimensi kecil hingga menengah dengan data yang memiliki *noise* cukup besar (memanfaatkan sifat *robust* dari Adam *optimizer*) untuk mencapai akurasi yang tinggi.

4. REFERENSI

- Basaruddin, T. (2025). Topic 5: Optimization [Lecture slides]. Fakultas Ilmu Komputer, Universitas Indonesia. SCELE Fakultas Ilmu Komputer Universitas Indonesia.
- Broyden, C. G. (1970). The convergence of a class of double-rank minimization algorithms. *Journal of the Institute of Mathematics and Its Applications*, 6, 76–90.
- Bryan, K. (2007). Quasi-Newton Methods. Rose-Hulman Institute of Technology. <https://www.rose-hulman.edu/~bryan/lottamath/quasinewton.pdf>.
- Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12, 2121–2159.
- Fletcher, R. (1970). A new approach to variable metric algorithms. *The Computer Journal*, 13, 317–322.
- Goldfarb, D. F. (1970). A family of variable-metric methods derived by variational means. *Mathematics of Computation*, 24, 23–26.
- Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580. <https://arxiv.org/abs/1207.0580>.
- Kingma, D. P., & Ba, J. (2017). *Adam: A method for stochastic optimization*. arXiv preprint arXiv:1412.6980. <https://arxiv.org/abs/1412.6980>
- Lam, A. (2020, November 26). BFGS in a Nutshell: An Introduction to Quasi-Newton Methods. Towards Data Science. <https://towardsdatascience.com/bfgs-in-a-nutshell-an-introduction-to-quasi-newton-methods-21b0e13ee504>.
- Ngartera, L.. (2024). A Comparative Study of Optimization Techniques on the Rosenbrock Function. *Open Journal of Optimization*. Vol.13. 51-63. <http://dx.doi.org/10.4236/ojop.2024.133004>.
- Shanno, D. (1970). Conditioning of quasi-Newton methods for function minimization. *Mathematics of Computation*, 24, 647–656.
- Sun, R. (2019). Optimization for deep learning: Theory and algorithms. arXiv. <https://arxiv.org/abs/1912.08957>.
- Tieleman, T., & Hinton, G. (2012). Lecture 6.5 — RMSProp: Coursera Neural Networks for Machine Learning. Technical report.

LAMPIRAN

adam_optimizer.m

```
% Inisialisasi parameter
theta0 = 10;
gamma = 0.1;
beta1 = 0.9;
beta2 = 0.999;
lambda = 0;
amsgrad = false;
maximize = false;
T = 100;
epsilon = 1e-8;

% Definisikan fungsi gradien
function gradient = grad_func(theta)
    gradient = 2 * theta;
endfunction

% Definisikan Adam Optimizer
function theta = adam_optimizer(theta0, grad_func, gamma, beta1, beta2, lambda, amsgrad,
maximize, T, epsilon)

    % Inisialisasi variabel
    theta = theta0;
    m = zeros(size(theta0)); % Momen pertama
    v = zeros(size(theta0)); % Momen kedua

    if amsgrad
        vmax = zeros(size(theta0)); % Momen kedua maksimum untuk AMSGrad
    endif

    % Iterasi untuk T steps
    for t = 1:T
        % Hitung gradien theta
        theta_grad = grad_func(theta);

        if maximize
            theta_grad = -theta_grad; % Negasi gradien untuk maximize
        endif

        if lambda ~= 0
            theta_grad = theta_grad + lambda * theta; % Weight decay
        endif

        % Perbarui momen
        m = beta1 * m + (1 - beta1) * theta_grad;
        v = beta2 * v + (1 - beta2) * (theta_grad .^ 2); % Element-wise square

        % Perbaiki momen berdasarkan bias
        m_hat = m / (1 - beta1^t);
        v_hat = v / (1 - beta2^t);

        % Perbarui parameter
        if amsgrad
```



```

        v_max = max(v_max, v_hat); % Element-wise maximum
        theta = theta - gamma * m_hat ./ (sqrt(v_max) + epsilon);
    else
        theta = theta - gamma * m_hat ./ (sqrt(v_hat) + epsilon);
    endif

endfor

endfunction

% Jalankan Adam Optimizer
printf("Parameter initial:\n")
printf("theta0 = 10\n")
printf("gamma = 0.1\n")
printf("beta1 = 0.9\n")
printf("lambda = 0\n")
printf("amsgrad = false\n")
printf("maximize = false\n")
printf("T = 100\n")
printf("epsilon = 1e-8\n")
printf("\n")
optimized_theta = adam_optimizer(theta0, @grad_func, gamma, beta1, beta2, lambda,
amsgrad, maximize, T, epsilon);
disp(['Optimized theta: ', num2str(optimized_theta)]);

```

adam_rosenbrock_optimization.m

```

% Inisialisasi hyperparameter
gamma = 0.001; % Learning rate
beta1 = 0.9; % Decay momen pertama
beta2 = 0.999; % Decay momen kedua
lambda = 0; % Tidak ada decay
amsgrad = false;
maximize = false;
T = 10000; % Jumlah iterasi
epsilon = 1e-8; % Pengstabil

% Inisialisasi vektor N
N_values = [3, 7, 100, 500, 1000];

% Definisikan fungsi gradien dari fungsi Rosenbrock
function gradient = rosenbrock_grad(x)
    N = length(x);
    gradient = zeros(N, 1);

    % Hitung gradien dari x_1
    gradient(1) = -400 * x(1) * (x(2) - x(1)^2) - 2 * (1 - x(1));

    % Hitung gradien dari x_2 hingga x_(N-1)
    for i = 2:(N-1)

```

```

    gradient(i) = -400 * x(i) * (x(i+1) - x(i)^2) - 2 * (1 - x(i)) + 200 * (x(i) -
x(i-1)^2);
    endfor

    % Hitung gradien dari x_N
    gradient(N) = 200 * (x(N) - x(N-1)^2);
endfunction

% Definisikan fungsi Rosenbrock
function value = rosenbrock_func(x)
    N = length(x);
    value = 0;
    for i = 1:(N-1)
        value = value + 100 * (x(i+1) - x(i)^2)^2 + (1 - x(i))^2;
    endfor
endfunction

% Definisikan Adam Optimizer
function theta = adam_optimizer(theta0, grad_func, gamma, beta1, beta2, lambda, amsgrad,
maximize, T, epsilon)
    theta = theta0;
    m = zeros(size(theta0)); % Momen pertama
    v = zeros(size(theta0)); % Momen kedua

    if amsgrad
        vmax = zeros(size(theta0)); % Momen kedua maksimum untuk AMSGrad
    end

    for t = 1:T
        theta_grad = grad_func(theta);

        if maximize
            theta_grad = -theta_grad; % Negasi gradien untuk maximize
        end

        if lambda ~= 0
            theta_grad = theta_grad + lambda * theta; % Weight decay
        end

        m = beta1 * m + (1 - beta1) * theta_grad;
        v = beta2 * v + (1 - beta2) * (theta_grad.^ 2);

        m_hat = m / (1 - beta1^t);
        v_hat = v / (1 - beta2^t);

        if amsgrad
            v_max = max(v_max, v_hat);
            theta = theta - gamma * m_hat ./ (sqrt(v_max) + epsilon);
        else
            theta = theta - gamma * m_hat ./ (sqrt(v_hat) + epsilon);
        end
    end
endfunction

% Tampilkan hiperparameter awal
printf("Hyperparameter initial:\n")

```

```

printf("theta0 = zeros(N, 1)\n") % Diperbarui sesuai kode
printf("gamma = 0.001\n") % Diperbarui sesuai kode
printf("beta1 = 0.9\n")
printf("beta2 = 0.999\n") % Ditambahkan
printf("lambda = 0\n")
printf("amsgrad = false\n")
printf("maximize = false\n")
printf("T = 10000\n")
printf("epsilon = 1e-8\n")
printf("\n")

% Jalankan Adam Rosenbrock Optimization untuk seluruh elemen dari vektor N
for N = N_values
    % Tebakan awal: all zeros
    x0 = zeros(N, 1);

    % Mulai pengukuran waktu
    tic;

    % Jalankan Adam Optimizer
    x_opt = adam_optimizer(x0, @rosenbrock_grad, gamma, beta1, beta2, lambda, amsgrad,
maximize, T, epsilon);

    % Hitung waktu yang telah berlalu
    elapsed_time = toc;

    % Hitung nilai akhir fungsi
    final_value = rosenbrock_func(x_opt);

    % Hitung jarak ke true minimum (all ones)
    true_min = ones(N, 1);
    distance_to_min = norm(x_opt - true_min);

    % Tampilkan hasil
    printf("N = %d\n", N);
    printf("Nilai akhir fungsi: %.6f\n", final_value);
    printf("Jarak ke true minimum: %.6f\n", distance_to_min);
    printf("Elapsed time: %.4f seconds\n", elapsed_time);
    printf("---\n");
endfor

```

rosenbrock_n.m

```

function [f, grad] = rosenbrock_n(x)
    % Fungsi Rosenbrock multivariat dan gradien untuk dimensi N
    n = length(x);
    f = 0;
    grad = zeros(n,1);

    for i = 1:n-1
        f = f + 100*(x(i+1) - x(i)^2)^2 + (1 - x(i))^2;
    end

```

```

        grad(i) = grad(i) - 400*(x(i+1) - x(i)^2)*x(i) - 2*(1 - x(i));
        grad(i+1) = grad(i+1) + 200*(x(i+1) - x(i)^2);
    end
end

```

quasi_newton_bfgs.m

```

function [x, f_val, grad, k, back_errs] = quasi_newton_bfgs(n, x0)
% Optimisasi fungsi Rosenbrock dengan dimensi N
x = x0;
max_iter = 2000;
tol = 1e-6;
H = eye(n);
back_errs = zeros(max_iter,1); % Array penyimpan backward error

for k = 1:max_iter
    [f_val, grad] = rosenbrock_n(x);
    back_errs(k) = norm(grad);
    if back_errs(k) < tol
        fprintf('Konvergen pada N=%d dalam %d iterasi\n', n, k);
        % Truncate array penyimpan backward error sampai ujung iterasi terakhir saja
        back_errs = back_errs(1:k);
        return;
    end

    % Tentukan arah pencarian
    p = -H * grad;

    % === Backtracking line search (Armijo rule) ===
    alpha = 1;
    rho_ls = 0.5;
    c = 1e-4;
    while true
        x_new = x + alpha * p;
        f_new = rosenbrock_n(x_new);
        if f_new <= f_val + c * alpha * grad' * p
            break;
        end
        alpha = alpha * rho_ls;
        if alpha < 1e-8
            warning('Step size terlalu kecil, keluar dari line search');
            break;
        end
    end

    [~, grad_new] = rosenbrock_n(x_new);
    s = x_new - x;
    y = grad_new - grad;
    ys = y' * s;

    if abs(ys) < 1e-10
        x = x_new;
        continue;
    end
end

```

```

        end

        rho = 1 / ys;

        I = eye(n);
        H = (I - rho * (s * y')) * H * (I - rho * (y * s')) + rho * (s * s');

        x = x_new;
    end

    fprintf('Maksimum iterasi tercapai untuk N=%d\n', n);
end

```

run_rosenbrock_optim.m

```

% Daftar ukuran dimensi N
Ns = [3, 7, 100, 500, 1000];

for i = 1:length(Ns)
    N = Ns(i);
    fprintf('=====\n');
    fprintf('Optimisasi fungsi Rosenbrock dengan Quasi Newton BFGS untuk N = %d\n', N);

    % Tebakan awal umum x0 = [-1, -1,..., -1]
    x0 = -ones(N,1);

    tic;
    [x_opt, f_val, grad, iter, back_errs] = quasi_newton_bfgs(N, x0);
    elapsed_time = toc;
    converge_rate = log(back_errs(iter)/back_errs(iter-1))...
        /log(back_errs(iter-1)/back_errs(iter-2));

    fprintf('Iterasi: %d\n', iter);
    fprintf('Nilai x:\n');
    disp(x_opt(1:min(size(x_opt,1),10)));
    fprintf('Nilai grad f(x):\n');
    disp(grad(1:min(size(grad,1),10)));
    fprintf('Waktu komputasi: %.2f detik\n\n', elapsed_time);
    fprintf('Estimasi laju konvergensi: %.2f \n\n', converge_rate);
end

```

Laporan Pengerjaan Tugas Kelompok 2 Analisis Numerik

Old School Animation



Disusun oleh Kelompok A2:

Raden Ahmad Yasin Mahendra (2306215154)

Ischika Afrilla (2306227955)



Yasmine Putri Viryadhani (2206081862)

Fabian Akmal Arkandion (2106750660)

Humam Al Labib (2206081755)

Pakta Integritas

Dengan ini, kami menyatakan bahwa tugas ini adalah hasil pekerjaan kelompok sendiri.

Fabian Akmal Arkandion	Ischika Afrilla	Yasmine Putri V.
		
Humam Al Labib	Raden Ahmad Yasin Mahendra	
		

RANGKUMAN

Laporan ini membahas penerapan metode interpolasi numerik untuk menghasilkan gerakan animasi yang halus dari seekor kuda yang sedang berlari. Fokus utama adalah pada komponen gerakan vertikal (koordinat- y) dari salah satu kaki kuda, berdasarkan serangkaian gambar (*frame*) dalam satu siklus lari.

Laporan terbagi ke dalam empat bagian utama:

1. Pengumpulan dan Pra-pemrosesan Data

Data diambil dari *frame-frame* animasi GIF yang telah dipecah menjadi gambar terpisah. Salah satu titik merah pada kaki kuda dilacak untuk memperoleh nilai koordinat- y pada setiap waktu (*frame*) $t = 1, 2, \dots, N$ dan disusun dalam bentuk tabel sebagai pasangan (t_i, y_i) .

2. Interpolasi Polinomial dan Pengaruh Basis

Empat basis polinomial berbeda dievaluasi (monomial biasa, terpusat, tertranslasi, dan terstandarisasi). Untuk tiap basis, sistem linear $Ac = y$ disusun dan *condition number* dari matriks A dihitung untuk menentukan stabilitas numerik. Basis dengan *condition number* terkecil dipilih untuk membangun polinomial interpolasi $p_y(t)$.

3. Evaluasi Polinomial dan Penambahan Titik

Koefisien polinomial dihitung menggunakan basis pilihan dan ditulis ulang dalam bentuk bersarang agar evaluasi menggunakan metode Horner menjadi efisien. Fungsi interpolasi ini kemudian digunakan untuk memprediksi nilai $y(t)$ di antara dan sedikit di luar *frame* asli. Satu titik tambahan (t_{N+1}, y_{N+1}) juga ditambahkan untuk membentuk interpolasi polinomial berderajat N .

4. Interpolasi *Natural Cubic Spline* dan Perbandingan Metode

Interpolasi *natural cubic spline* diimplementasikan dari nol (tanpa *library*) untuk menyusun sistem tridiagonal dan menyelesaikannya guna memperoleh *spline* $y(t)$. Kurva *spline* dievaluasi di interval halus dan dibandingkan dengan kurva polinomial. Perbandingan fokus pada perilaku kurva di antara dan di luar data asli ($t > N$), serta stabilitas hasil interpolasi.

Akhirnya, hasil interpolasi polinomial (derajat N , $N - 1$), interpolasi *natural spline*, dan evaluasi data ditampilkan dalam bentuk visual (plot 2D $(t, y(t))$) untuk melihat keakuratan dan kestabilan hasil interpolasi.

DAFTAR ISI

1. PENDAHULUAN.....	1
2. PEMBAHASAN.....	2
2.1 Pengumpulan dan Pra-Pemrosesan Data.....	2
2.2 Pengaruh Basis pada Interpolasi Polinomial untuk $p_y(t)$	2
2.2.1 Sistem Persamaan Linear menggunakan Basis Pertama.....	3
2.2.2 Sistem Persamaan Linear menggunakan Basis Kedua.....	4
2.2.3 Sistem Persamaan Linear menggunakan Basis Ketiga.....	5
2.2.4 Sistem Persamaan Linear menggunakan Basis Keempat.....	7
2.2.5 Analisis Condition Number.....	8
2.3 Interpolasi Polinomial $p_y(t)$ dengan Basis Pilihan.....	9
2.3.1 Menentukan Koefisien c	9
2.3.2 Bentuk Bersarang dari Polinomial $p_y(t)$	10
2.3.3 Evaluasi $p_y(t)$ dalam Rentang $[1, N]$	10
2.3.4 Penambahan Data Baru Interpolasi.....	14
2.3.5 Plotting Lintasan 2D.....	16
2.4 Interpolasi Natural Cubic Splines untuk $y(t)$ dan Evaluasi Hasil Interpolasi.....	17
2.4.1 Natural Cubic Splines untuk $y(t)$	17
2.4.2 Perbandingan dan Analisis antara Interpolasi menggunakan Polinomial dan Natural Cubic Spline.....	18
3. KESIMPULAN.....	20
4. REFERENSI.....	21
LAMPIRAN.....	22

1. PENDAHULUAN

Dalam industri animasi, kelancaran gerakan memegang peran kunci dalam menciptakan kesan realistis dan estetis. Salah satu teknik yang umum digunakan adalah *in-betweening* atau interpolasi, di mana *frame* tambahan dihasilkan secara otomatis di antara *keyframe* utama untuk menghasilkan gerakan yang halus. Interpolasi numerik menjadi dasar matematis yang penting dalam memprediksi posisi objek pada waktu atau *frame* tertentu, sehingga memungkinkan pembuatan animasi yang efisien tanpa kehilangan kualitas visual.

Laporan ini berfokus pada penerapan metode interpolasi lanjutan untuk menganalisis gerakan vertikal (y) salah satu kaki kuda dalam sebuah siklus lari yang diberikan dalam bentuk rangkaian *frame* GIF. Data koordinat y dari titik tertentu pada setiap *frame* akan digunakan untuk membentuk suatu fungsi interpolasi guna memperkirakan lintasan gerakan secara lebih halus. Melalui laporan ini, akan dibahas pengumpulan dan pra-pemrosesan data, pengaruh basis pada interpolasi polinomial untuk $p_y(t)$, interpolasi polinomial $p_y(t)$ dengan basis pilihan, serta interpolasi *natural cubic splines* untuk $y(t)$ dan evaluasi hasil interpolasi.

2. PEMBAHASAN

2.1 Pengumpulan dan Pra-Pemrosesan Data

Kita akan melakukan pengumpulan dan pra-pemrosesan data dengan memilih titik kunci pada salah satu kaki kuda (ditandai warna merah di kaki kiri depan kuda dengan *pixel* kiri atas). Lalu, kita mengambil posisi- y titik kunci tersebut (dengan $y = 0$ di sudut kiri atas) pada setiap *frame* di gif kuda. Kita menggunakan alat online ezgif.com dan pixspy.com untuk membantu pengumpulan dan pra-pemrosesan data ini.

Hasilnya terdapat 10 *frame* pada gif tersebut. Ini hasil posisi- y titik kunci tersebut pada *frame* t .

Tabel 2.1.1: Data 10 frame pada gif

t (frame)	y (pixel)
1	132
2	136
3	149
4	152
5	149
6	150
7	143
8	135
9	126
10	127

2.2 Pengaruh Basis pada Interpolasi Polinomial untuk $p_y(t)$

Kita bisa gunakan data koordinat (t_i, y_i) dari 10 *frame* tersebut untuk aproksimasi lintasan vertikal y sebagai fungsi waktu t , $p_y(t)$, menggunakan interpolasi polinomial berderajat 9.

Polinomial interpolasi berderajat 9 dapat direpresentasikan dengan berbagai basis. Jika dipilih basis $\{\Phi_0(t), \Phi_1(t), \dots, \Phi_9(t)\}$, maka polinomial interpolasi dapat direpresentasikan sebagai:

$$p_y(t) = c_0\Phi_0(t) + c_1\Phi_1(t) + \cdots + c_9\Phi_9(t)$$

Untuk menginterpolasi titik-titik data (t_i, y_i) , kita perlu mencari koefisien c_j sehingga $p_y(t_i) = y_i$ untuk $i = 1, 2, \dots, 10$. Ini mengarah pada penyelesaian sistem persamaan linear $Ac = y$, di mana matriks A bergantung pada pilihan basis $\Phi_j(t)$ dan titik-titik t_i .

Pada laporan ini kita akan eksperimen dengan 4 fungsi-fungsi basis berikut.

1. $\Phi_i(t) = t^i$
2. $\Phi_i(t) = (t - t_{avg})^i$
3. $\Phi_i(t) = (t - N)^i$
4. $\Phi_i(t) = \left(\frac{t - t_{avg}}{2}\right)^i$

dengan $N = 10$ (banyaknya data) dan $t_{avg} = \frac{1+N}{2} = \frac{11}{2} = 5.5$

2.2.1 Sistem Persamaan Linear menggunakan Basis Pertama

Kita masukkan titik-titik data (t_i, y_i) pada persamaan polinomial interpolasi menggunakan basis pertama. Dalam kata lain:

$$c_0t_i^0 + c_1t_i^1 + \cdots + c_9t_i^9 = y_i$$

Maka kita akan mendapatkan sistem persamaan linear:

$$\begin{array}{ccccccccc} c_0 & + & c_1t_1 & + & \cdots & + & c_9t_1^9 & = & y_1 \\ c_0 & + & c_1t_2 & + & \cdots & + & c_9t_2^9 & = & y_2 \\ \vdots & + & \vdots & + & \ddots & + & \vdots & = & \vdots \\ c_0 & + & c_1t_{10} & + & \cdots & + & c_9t_{10}^9 & = & y_{10} \end{array}$$

Dengan ini kita bisa merepresentasikan sistem persamaan linear dalam bentuk $Ac = y$:

$$\begin{bmatrix} 1 & t_1 & \cdots & t_1^9 \\ 1 & t_2 & \cdots & t_2^9 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & t_{10} & \cdots & t_{10}^9 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_9 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{10} \end{bmatrix}$$

Secara eksplisit, ini adalah isi elemen-elemen dari matriks A .

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2^1 & 2^2 & 2^3 & 2^4 & 2^5 & 2^6 & 2^7 & 2^8 & 2^9 \\ 1 & 3^1 & 3^2 & 3^3 & 3^4 & 3^5 & 3^6 & 3^7 & 3^8 & 3^9 \\ 1 & 4^1 & 4^2 & 4^3 & 4^4 & 4^5 & 4^6 & 4^7 & 4^8 & 4^9 \\ 1 & 5^1 & 5^2 & 5^3 & 5^4 & 5^5 & 5^6 & 5^7 & 5^8 & 5^9 \\ 1 & 6^1 & 6^2 & 6^3 & 6^4 & 6^5 & 6^6 & 6^7 & 6^8 & 6^9 \\ 1 & 7^1 & 7^2 & 7^3 & 7^4 & 7^5 & 7^6 & 7^7 & 7^8 & 7^9 \\ 1 & 8^1 & 8^2 & 8^3 & 8^4 & 8^5 & 8^6 & 8^7 & 8^8 & 8^9 \\ 1 & 9^1 & 9^2 & 9^3 & 9^4 & 9^5 & 9^6 & 9^7 & 9^8 & 9^9 \\ 1 & 10^1 & 10^2 & 10^3 & 10^4 & 10^5 & 10^6 & 10^7 & 10^8 & 10^9 \end{bmatrix}$$

2.2.2 Sistem Persamaan Linear menggunakan Basis Kedua

Kita masukkan titik-titik data (t_i, y_i) pada persamaan polinomial interpolasi menggunakan basis kedua. Dalam kata lain:

$$c_0(t_i - 5.5)^0 + c_1(t_i - 5.5)^1 + \cdots + c_9(t_i - 5.5)^9 = y_i$$

Maka kita akan mendapatkan sistem persamaan linear:

$$\begin{array}{ccccccccc}
 c_0 & + & c_1(t_1 - 5.5) & + & \cdots & + & c_9(t_1 - 5.5)^9 & = & y_1 \\
 c_0 & + & c_1(t_2 - 5.5) & + & \cdots & + & c_9(t_2 - 5.5)^9 & = & y_2 \\
 \vdots & + & \vdots & + & \ddots & + & \vdots & = & \vdots \\
 c_0 & + & c_1(t_{10} - 5.5) & + & \cdots & + & c_9(t_{10} - 5.5)^9 & = & y_{10}
 \end{array}$$

Dengan ini kita bisa merepresentasikan sistem persamaan linear dalam bentuk $Ac = y$:

$$\begin{bmatrix}
 1 & (t_1 - 5.5) & \cdots & (t_1 - 5.5)^9 \\
 1 & (t_2 - 5.5) & \cdots & (t_2 - 5.5)^9 \\
 \vdots & \vdots & \ddots & \vdots \\
 1 & (t_{10} - 5.5) & \cdots & (t_{10} - 5.5)^9
 \end{bmatrix}
 \begin{bmatrix}
 c_0 \\
 c_1 \\
 \vdots \\
 c_9
 \end{bmatrix}
 =
 \begin{bmatrix}
 y_1 \\
 y_2 \\
 \vdots \\
 y_{10}
 \end{bmatrix}$$

Secara eksplisit, ini adalah isi elemen-elemen dari matriks A .

$$A = \begin{bmatrix}
 1 & (-4.5)^1 & (-4.5)^2 & (-4.5)^3 & (-4.5)^4 & (-4.5)^5 & (-4.5)^6 & (-4.5)^7 & (-4.5)^8 & (-4.5)^9 \\
 1 & (-3.5)^1 & (-3.5)^2 & (-3.5)^3 & (-3.5)^4 & (-3.5)^5 & (-3.5)^6 & (-3.5)^7 & (-3.5)^8 & (-3.5)^9 \\
 1 & (-2.5)^1 & (-2.5)^2 & (-2.5)^3 & (-2.5)^4 & (-2.5)^5 & (-2.5)^6 & (-2.5)^7 & (-2.5)^8 & (-2.5)^9 \\
 1 & (-1.5)^1 & (-1.5)^2 & (-1.5)^3 & (-1.5)^4 & (-1.5)^5 & (-1.5)^6 & (-1.5)^7 & (-1.5)^8 & (-1.5)^9 \\
 1 & (-0.5)^1 & (-0.5)^2 & (-0.5)^3 & (-0.5)^4 & (-0.5)^5 & (-0.5)^6 & (-0.5)^7 & (-0.5)^8 & (-0.5)^9 \\
 1 & (0.5)^1 & (0.5)^2 & (0.5)^3 & (0.5)^4 & (0.5)^5 & (0.5)^6 & (0.5)^7 & (0.5)^8 & (0.5)^9 \\
 1 & (1.5)^1 & (1.5)^2 & (1.5)^3 & (1.5)^4 & (1.5)^5 & (1.5)^6 & (1.5)^7 & (1.5)^8 & (1.5)^9 \\
 1 & (2.5)^1 & (2.5)^2 & (2.5)^3 & (2.5)^4 & (2.5)^5 & (2.5)^6 & (2.5)^7 & (2.5)^8 & (2.5)^9 \\
 1 & (3.5)^1 & (3.5)^2 & (3.5)^3 & (3.5)^4 & (3.5)^5 & (3.5)^6 & (3.5)^7 & (3.5)^8 & (3.5)^9 \\
 1 & (4.5)^1 & (4.5)^2 & (4.5)^3 & (4.5)^4 & (4.5)^5 & (4.5)^6 & (4.5)^7 & (4.5)^8 & (4.5)^9
 \end{bmatrix}$$

2.2.3 Sistem Persamaan Linear menggunakan Basis Ketiga

Kita masukkan titik-titik data (t_i, y_i) pada persamaan polinomial interpolasi menggunakan basis ketiga. Dalam kata lain:

$$c_0(t_i - 10)^0 + c_1(t_i - 10)^1 + \cdots + c_9(t_i - 10)^9 = y_i$$

Maka kita akan mendapatkan sistem persamaan linear:

$$\begin{array}{ccccccccc} c_0 & + & c_1(t_1 - 10) & + & \cdots & + & c_9(t_1 - 10)^9 & = & y_1 \\ c_0 & + & c_1(t_2 - 10) & + & \cdots & + & c_9(t_2 - 10)^9 & = & y_2 \\ \vdots & + & \vdots & + & \ddots & + & \vdots & = & \vdots \\ c_0 & + & c_1(t_{10} - 10) & + & \cdots & + & c_9(t_{10} - 10)^9 & = & y_{10} \end{array}$$

Note: Asumsikan $(t_{10} - 10)^0 = (10 - 10)^0 = 0^0 = 1$

Dengan ini kita bisa merepresentasikan sistem persamaan linear dalam bentuk $Ac = y$:

$$\begin{bmatrix} 1 & (t_1 - 10) & \cdots & (t_1 - 10)^9 \\ 1 & (t_2 - 10) & \cdots & (t_2 - 10)^9 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & (t_{10} - 10) & \cdots & (t_{10} - 10)^9 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_9 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{10} \end{bmatrix}$$

Secara eksplisit, ini adalah isi elemen-elemen dari matriks A .

$$A = \begin{bmatrix} 1 & (-9)^1 & (-9)^2 & (-9)^3 & (-9)^4 & (-9)^5 & (-9)^6 & (-9)^7 & (-9)^8 & (-9)^9 \\ 1 & (-8)^1 & (-8)^2 & (-8)^3 & (-8)^4 & (-8)^5 & (-8)^6 & (-8)^7 & (-8)^8 & (-8)^9 \\ 1 & (-7)^1 & (-7)^2 & (-7)^3 & (-7)^4 & (-7)^5 & (-7)^6 & (-7)^7 & (-7)^8 & (-7)^9 \\ 1 & (-6)^1 & (-6)^2 & (-6)^3 & (-6)^4 & (-6)^5 & (-6)^6 & (-6)^7 & (-6)^8 & (-6)^9 \\ 1 & (-5)^1 & (-5)^2 & (-5)^3 & (-5)^4 & (-5)^5 & (-5)^6 & (-5)^7 & (-5)^8 & (-5)^9 \\ 1 & (-4)^1 & (-4)^2 & (-4)^3 & (-4)^4 & (-4)^5 & (-4)^6 & (-4)^7 & (-4)^8 & (-4)^9 \\ 1 & (-3)^1 & (-3)^2 & (-3)^3 & (-3)^4 & (-3)^5 & (-3)^6 & (-3)^7 & (-3)^8 & (-3)^9 \\ 1 & (-2)^1 & (-2)^2 & (-2)^3 & (-2)^4 & (-2)^5 & (-2)^6 & (-2)^7 & (-2)^8 & (-2)^9 \\ 1 & (-1)^1 & (-1)^2 & (-1)^3 & (-1)^4 & (-1)^5 & (-1)^6 & (-1)^7 & (-1)^8 & (-1)^9 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

2.2.4 Sistem Persamaan Linear menggunakan Basis Keempat

Kita masukkan titik-titik data (t_i, y_i) pada persamaan polinomial interpolasi menggunakan basis keempat. Dalam kata lain:

$$c_0(0.5t_i - 2.75)^0 + c_1(0.5t_i - 2.75)^1 + \cdots + c_9(0.5t_i - 2.75)^9 = y_i$$

Maka kita akan mendapatkan sistem persamaan linear:

$$\begin{array}{cccccccccc} c_0 & + & c_1(0.5t_1 - 2.75) & + & \cdots & + & c_9(0.5t_1 - 2.75)^9 & = & y_1 \\ c_0 & + & c_1(0.5t_2 - 2.75) & + & \cdots & + & c_9(0.5t_2 - 2.75)^9 & = & y_2 \\ \vdots & + & \vdots & + & \ddots & + & \vdots & = & \vdots \\ c_0 & + & c_1(0.5t_{10} - 2.75) & + & \cdots & + & c_9(0.5t_{10} - 2.75)^9 & = & y_{10} \end{array}$$

Dengan ini kita bisa merepresentasikan sistem persamaan linear dalam bentuk $Ac = y$:

$$\begin{bmatrix} 1 & (0.5t_1 - 2.75) & \cdots & (0.5t_1 - 2.75)^9 \\ 1 & (0.5t_2 - 2.75) & \cdots & (0.5t_2 - 2.75)^9 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & (0.5t_{10} - 2.75) & \cdots & (0.5t_{10} - 2.75)^9 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_9 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{10} \end{bmatrix}$$

Secara eksplisit, ini adalah isi elemen-elemen dari matriks A .

$$A = \begin{bmatrix} 1 & (-2.25)^1 & (-2.25)^2 & (-2.25)^3 & (-2.25)^4 & (-2.25)^5 & (-2.25)^6 & (-2.25)^7 & (-2.25)^8 & (-2.25)^9 \\ 1 & (-1.75)^1 & (-1.75)^2 & (-1.75)^3 & (-1.75)^4 & (-1.75)^5 & (-1.75)^6 & (-1.75)^7 & (-1.75)^8 & (-1.75)^9 \\ 1 & (-1.25)^1 & (-1.25)^2 & (-1.25)^3 & (-1.25)^4 & (-1.25)^5 & (-1.25)^6 & (-1.25)^7 & (-1.25)^8 & (-1.25)^9 \\ 1 & (-0.75)^1 & (-0.75)^2 & (-0.75)^3 & (-0.75)^4 & (-0.75)^5 & (-0.75)^6 & (-0.75)^7 & (-0.75)^8 & (-0.75)^9 \\ 1 & (-0.25)^1 & (-0.25)^2 & (-0.25)^3 & (-0.25)^4 & (-0.25)^5 & (-0.25)^6 & (-0.25)^7 & (-0.25)^8 & (-0.25)^9 \\ 1 & (0.25)^1 & (0.25)^2 & (0.25)^3 & (0.25)^4 & (0.25)^5 & (0.25)^6 & (0.25)^7 & (0.25)^8 & (0.25)^9 \\ 1 & (0.75)^1 & (0.75)^2 & (0.75)^3 & (0.75)^4 & (0.75)^5 & (0.75)^6 & (0.75)^7 & (0.75)^8 & (0.75)^9 \\ 1 & (1.25)^1 & (1.25)^2 & (1.25)^3 & (1.25)^4 & (1.25)^5 & (1.25)^6 & (1.25)^7 & (1.25)^8 & (1.25)^9 \\ 1 & (1.75)^1 & (1.75)^2 & (1.75)^3 & (1.75)^4 & (1.75)^5 & (1.75)^6 & (1.75)^7 & (1.75)^8 & (1.75)^9 \\ 1 & (2.25)^1 & (2.25)^2 & (2.25)^3 & (2.25)^4 & (2.25)^5 & (2.25)^6 & (2.25)^7 & (2.25)^8 & (2.25)^9 \end{bmatrix}$$

2.2.5 Analisis *Condition Number*

Setelah mendapatkan empat matriks dari basis-basisnya, kita analisis kestabilan permasalahannya menggunakan *condition number*.

Condition number didapatkan dengan perkalian dari *norm* matriks A dengan *norm* invers matriks A . Dalam kata lain,

$$\|A\| \cdot \|A^{-1}\|$$

Maka dari itu, kita bisa dapatkan *condition number* pada matriks basis pertama, kedua, ketiga, dan keempat dengan *norm* 1 sebagai berikut:

Tabel 2.2.5.1: *Condition number masing-masing basis dengan norm 1*

Matriks basis ke-	Norm A	Norm A^{-1}	Condition Number
1	1.5743×10^9	2310	3.6366×10^{12}
2	1.6787×10^6	3.4554	5.8005×10^6
3	5.7430×10^8	252	1.4472×10^{11}
4	3278.7	21.914	7.1849×10^4

Dari tabel tersebut, dapat disimpulkan bahwa menggunakan basis keempat yang paling stabil secara numerik dibanding basis-basis lainnya karena memiliki *condition number* terkecil. Tetapi itu menggunakan *norm* 1, apakah dengan menggunakan *norm* lain, *norm infinity*, dapat menghasilkan kesimpulan yang berbeda?

Maka dari itu, kita hitung ulang *condition number*-nya dengan menggunakan *norm infinity*. Hasilnya sebagai berikut:

Tabel 2.2.5.2: *Condition number masing-masing basis dengan norm infinity*

Matriks basis ke-	Norm A	Norm A^{-1}	Condition Number
1	1.1111×10^8	2975.8	3.3064×10^{12}
2	9.7287×10^5	2.6959	2.6227×10^6
3	4.3585×10^8	333.41	1.4532×10^{11}

4	2659.4	21.567	5.7355×10^4
---	--------	--------	----------------------

Hasilnya tidak terlalu berbeda dengan yang menggunakan *norm* 1, lebih eksplisitnya perbedaan relatifnya paling besar 0.5478, jadi kita bisa interpolasikan data-data tersebut menggunakan basis keempat.

2.3 Interpolasi Polinomial $p_y(t)$ dengan Basis Pilihan

2.3.1 Menentukan Koefisien c

Berdasarkan analisis pada bagian 2.2.5 di atas, terlihat bahwa basis dengan *condition number* terbaik adalah basis keempat. Oleh karena itu, kita akan menentukan koefisien c untuk $p_y(t)$ menggunakan basis keempat.

Selesaikan sistem persamaan linear berikut:

$$Ac = y$$

$$\begin{bmatrix} 1 & (-2.25)^1 & (-2.25)^2 & (-2.25)^3 & (-2.25)^4 & (-2.25)^5 & (-2.25)^6 & (-2.25)^7 & (-2.25)^8 & (-2.25)^9 \\ 1 & (-1.75)^1 & (-1.75)^2 & (-1.75)^3 & (-1.75)^4 & (-1.75)^5 & (-1.75)^6 & (-1.75)^7 & (-1.75)^8 & (-1.75)^9 \\ 1 & (-1.25)^1 & (-1.25)^2 & (-1.25)^3 & (-1.25)^4 & (-1.25)^5 & (-1.25)^6 & (-1.25)^7 & (-1.25)^8 & (-1.25)^9 \\ 1 & (-0.75)^1 & (-0.75)^2 & (-0.75)^3 & (-0.75)^4 & (-0.75)^5 & (-0.75)^6 & (-0.75)^7 & (-0.75)^8 & (-0.75)^9 \\ 1 & (-0.25)^1 & (-0.25)^2 & (-0.25)^3 & (-0.25)^4 & (-0.25)^5 & (-0.25)^6 & (-0.25)^7 & (-0.25)^8 & (-0.25)^9 \\ 1 & (0.25)^1 & (0.25)^2 & (0.25)^3 & (0.25)^4 & (0.25)^5 & (0.25)^6 & (0.25)^7 & (0.25)^8 & (0.25)^9 \\ 1 & (0.75)^1 & (0.75)^2 & (0.75)^3 & (0.75)^4 & (0.75)^5 & (0.75)^6 & (0.75)^7 & (0.75)^8 & (0.75)^9 \\ 1 & (1.25)^1 & (1.25)^2 & (1.25)^3 & (1.25)^4 & (1.25)^5 & (1.25)^6 & (1.25)^7 & (1.25)^8 & (1.25)^9 \\ 1 & (1.75)^1 & (1.75)^2 & (1.75)^3 & (1.75)^4 & (1.75)^5 & (1.75)^6 & (1.75)^7 & (1.75)^8 & (1.75)^9 \\ 1 & (2.25)^1 & (2.25)^2 & (2.25)^3 & (2.25)^4 & (2.25)^5 & (2.25)^6 & (2.25)^7 & (2.25)^8 & (2.25)^9 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \\ c_8 \\ c_9 \end{bmatrix} = \begin{bmatrix} 132 \\ 136 \\ 149 \\ 152 \\ 149 \\ 150 \\ 143 \\ 135 \\ 126 \\ 127 \end{bmatrix}$$

Terlihat bahwa sistem persamaan linear di atas membentuk sistem Vandermonde dengan matriks Vandermonde sebagai koefisiennya. Kita akan menyelesaikan sistem tersebut menggunakan metode eliminasi Gauss.

Dengan menyelesaikan sistem ini, kita akan memperoleh nilai c sebagai berikut:

$$c = \begin{bmatrix} 1.4973e + 02 \\ 3.6840e + 00 \\ -3.5954e + 00 \\ -2.8388e + 01 \\ -4.3559e - 01 \\ 2.3526e + 01 \\ -4.2778e - 01 \\ -6.8836e + 00 \\ 9.8413e - 02 \\ 6.5326e - 01 \end{bmatrix}$$

Dengan *backward error* e sebagai berikut:

$$\|Ac - y\|_2 = \left\| \begin{bmatrix} 0 \\ 5.6843e-14 \\ 8.5265e-14 \\ 1.4211e-13 \\ -5.6843e-14 \\ -7.6739e-13 \\ -2.3022e-12 \\ -5.2296e-12 \\ -1.0260e-11 \\ -1.8275e-11 \end{bmatrix} \right\|_2 = 0$$

2.3.2 Bentuk Bersarang dari Polinomial $p_y(t)$

Berdasarkan nilai koefisien c pada bagian 2.3.1 dengan menggunakan basis keempat, polinomial $p_y(t)$ dapat dituliskan kembali dalam bentuk berikut:

$$p_y(t) = c_0\Phi_0(t) + c_1\Phi_1(t) + c_2\Phi_2(t) + c_3\Phi_3(t) + c_4\Phi_4(t) + c_5\Phi_5(t) + c_6\Phi_6(t) + c_7\Phi_7(t) + c_8\Phi_8(t) + c_9\Phi_9(t)$$

$$p_y(t) = (1.4973 \cdot 10^2) \cdot \left(\frac{t-5.5}{2}\right)^0 + (3.6840) \cdot \left(\frac{t-5.5}{2}\right)^1 + (-3.5954) \cdot \left(\frac{t-5.5}{2}\right)^2 + (-2.8388 \cdot 10^1) \cdot \left(\frac{t-5.5}{2}\right)^3 +$$

$$(-4.3559 \cdot 10^{-1}) \cdot \left(\frac{t-5.5}{2}\right)^4 + (2.3526 \cdot 10^1) \cdot \left(\frac{t-5.5}{2}\right)^5 + (-4.2778 \cdot 10^{-1}) \cdot \left(\frac{t-5.5}{2}\right)^6 + (-6.8836) \cdot \left(\frac{t-5.5}{2}\right)^7 +$$

$$(9.8413 \cdot 10^{-2}) \cdot \left(\frac{t-5.5}{2}\right)^8 + (6.5326 \cdot 10^{-1}) \cdot \left(\frac{t-5.5}{2}\right)^9$$

Misalkan $x = \left(\frac{t-5.5}{2}\right)$, maka dapat diperoleh bentuk bersarang sebagai berikut:

$$p_y(t) = (((((((((0.65326 \cdot x + 0.098413) \cdot x - 6.8836) \cdot x - 0.42778) \cdot x + 23.526) \cdot x - 0.43559) \cdot x - 28.388) \cdot x - 3.5954) \cdot x + 3.6840) \cdot x + 149.73$$

2.3.3 Evaluasi $p_y(t)$ dalam Rentang $[1, N]$

Evaluasi dilakukan pada titik-titik t di dalam rentang $[1, 10]$ dengan interval yang lebih rapat, yaitu setiap 0.1 satuan waktu. Tujuannya adalah untuk memperoleh estimasi nilai koordinat vertikal $y(t)$ secara lebih halus di antara *frame-frame* integer.

Tabel 2.3.3.1: Evaluasi $p_y(t)$ dalam rentang $[1, N]$

t (frame)	p_y (pixel)
1.0	132.02
1.1	137.71
1.2	140.44

1.3	141.20
1.4	140.77
1.5	139.73
1.6	138.49
1.7	137.36
1.8	136.50
1.9	136.04
2.0	136.01
2.1	136.40
2.2	137.19
2.3	138.30
2.4	139.67
2.5	141.22
2.6	142.86
2.7	144.52
2.8	146.13
2.9	147.64
3.0	149.00
3.1	150.18
3.2	151.16
3.3	151.91
3.4	152.45
3.5	152.78
3.6	152.91
3.7	152.87

3.8	152.69
3.9	152.39
4.0	152.00
4.1	151.56
4.2	151.10
4.3	150.64
4.4	150.21
4.5	149.82
4.6	149.50
4.7	149.26
4.8	149.09
4.9	149.01
5.0	149.00
5.1	149.07
5.2	149.19
5.3	149.35
5.4	149.54
5.5	149.73
5.6	149.90
5.7	150.03
5.8	150.11
5.9	150.10
6.0	150.00
6.1	149.80
6.2	149.47

6.3	149.03
6.4	148.46
6.5	147.77
6.6	146.98
6.7	146.08
6.8	145.11
6.9	144.07
7.0	143.00
7.1	141.92
7.2	140.85
7.3	139.82
7.4	138.85
7.5	137.96
7.6	137.17
7.7	136.48
7.8	135.89
7.9	135.41
8.0	135.00
8.1	134.65
8.2	134.32
8.3	133.97
8.4	133.53
8.5	132.95
8.6	132.18
8.7	131.14

8.8	129.78
8.9	128.08
9.0	126.00
9.1	123.57
9.2	120.85
9.3	117.97
9.4	115.14
9.5	112.66
9.6	110.97
9.7	110.67
9.8	112.53
9.9	117.54
10.0	126.99

2.3.4 Penambahan Data Baru Interpolasi

Pada percobaan sebelumnya telah dibentuk sebuah polinomial interpolasi $p_y(t)$ berderajat $N - 1 = 9$ yang melewati $N = 10$ titik data (t_i, y_i) , dengan $t = 1, 2, \dots, 10$

. Polinomial tersebut disusun menggunakan basis keempat, yaitu $\Phi_i(t) = \left(\frac{t-t_{avg}}{2}\right)^i$, dengan $t_{avg} = \frac{1+N}{2} = \frac{11}{2} = 5.5$.

Dalam bagian ini, kita akan memperluas interpolasi tersebut menjadi polinomial berderajat $N = 10$ yang melewati $N + 1 = 11$ titik data. Titik tambahan yang diberikan adalah:

- $t_{11} = 11$
- $y_{11} = 135$ pixel

Penambahan satu baris dan satu koefisien baru dalam sistem linear interpolasi dapat dilakukan secara efisien tanpa menghitung ulang seluruh koefisien sebelumnya karena basis

keempat yang digunakan, yaitu $\Phi_i(t) = \left(\frac{t-t_{avg}}{2}\right)^i$ memiliki sifat polinomial terurut dan saling independen. Sifat terurut berarti setiap basis $\Phi_i(t)$ memiliki derajat yang lebih tinggi dari basis sebelumnya. Sementara itu, sifat independen berarti bahwa setiap basis $\Phi_i(t)$ tidak dapat dinyatakan sebagai kombinasi linear dari basis-basis sebelumnya.

Dengan memanfaatkan sifat ini, penambahan titik baru hanya memengaruhi koefisien pada orde tertinggi yang baru, sedangkan koefisien-koefisien sebelumnya tetap valid sehingga untuk membentuk polinomial $p_{y,N}(t)$ dari polinomial sebelumnya $p_y(t)$, kita cukup menambahkan satu koefisien baru $c_N = c_{10}$. Koefisien ini dapat dihitung dengan rumus berikut:

$$y_{N+1} = p_{y,N}(t_{N+1}) = p_y(t_{N+1}) + c_N \cdot \Phi_N(t_{N+1})$$

maka:

$$c_N = \frac{y_{N+1} - p_y(t_{N+1})}{\Phi_N(t_{N+1})}$$

di mana:

- $y_{N+1} = y_{10+1} = y_{11} = 135$
- $t_{N+1} = t_{10+1} = t_{11} = 11$
- $p_y(11) = 1042.9454$
- $\Phi_N(t_{N+1}) = \left(\frac{t_{N+1}-t_{avg}}{2}\right)^N = \left(\frac{11-5.5}{2}\right)^{10} = \left(\frac{5.5}{2}\right)^{10}$

sehingga,

$$c_N = \frac{135 - 1042.9454}{\left(\frac{5.5}{2}\right)^{10}}$$

$$c_N = -0.0367$$

Setelah memperoleh nilai $c_N = c_{10}$, polinomial berderajat N dapat dibentuk dengan menambahkan suku baru $c_N \cdot \left(\frac{t-t_{avg}}{2}\right)^N$ ke polinomial sebelumnya. Hasil akhirnya adalah polinomial berderajat N dengan struktur basis yang sama, tetapi terdiri atas $N + 1$ suku.

$$p_{y,N}(t) = c_0\Phi_0(t) + c_1\Phi_1(t) + c_2\Phi_2(t) + c_3\Phi_3(t) + c_4\Phi_4(t) + c_5\Phi_5(t) + c_6\Phi_6(t) + c_7\Phi_7(t) + c_8\Phi_8(t) + c_9\Phi_9(t) + c_{10}\Phi_{10}(t)$$

$$p_{y,N}(t) = (1.4973 \cdot 10^2) \cdot \left(\frac{t-5.5}{2}\right)^0 + (3.6840) \cdot \left(\frac{t-5.5}{2}\right)^1 + (-3.5954) \cdot \left(\frac{t-5.5}{2}\right)^2 + (-2.8388 \cdot 10^1) \cdot \left(\frac{t-5.5}{2}\right)^3 +$$

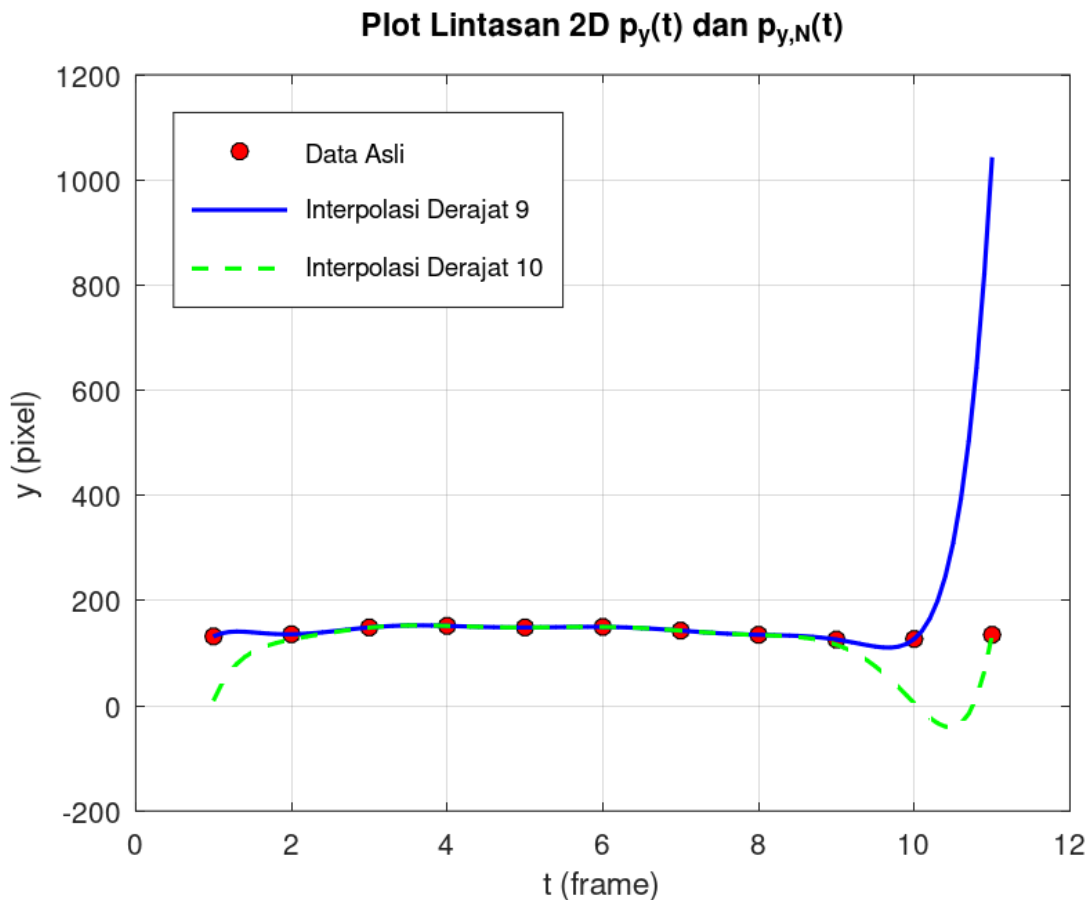
$$(-4.3559 \cdot 10^{-1}) \cdot \left(\frac{t-5.5}{2}\right)^4 + (2.3526 \cdot 10^1) \cdot \left(\frac{t-5.5}{2}\right)^5 + (-4.2778 \cdot 10^{-1}) \cdot \left(\frac{t-5.5}{2}\right)^6 + (-6.8836) \cdot \left(\frac{t-5.5}{2}\right)^7 +$$

$$(9.8413 \cdot 10^{-2}) \cdot \left(\frac{t-5.5}{2}\right)^8 + (6.5326 \cdot 10^{-1}) \cdot \left(\frac{t-5.5}{2}\right)^9 + (-0.0367) \cdot \left(\frac{t-5.5}{2}\right)^{10}$$

2.3.5 Plotting Lintasan 2D

Data $x_{N+1} = 350$ pixel pada t_{N+1} diberikan sebagai informasi tambahan untuk keperluan plotting 2D. Namun, untuk menyederhanakan visualisasi lintasan titik kunci pada bidang 2D, kita menganggap posisi horizontal x berubah secara linear mengikuti nilai t . Dengan kata lain, $x(t) = t$, yang merepresentasikan pergerakan objek ke arah horizontal secara seragam ke kanan.

Selanjutnya, lintasan titik kunci diplot dengan menggunakan pasangan titik data asli (t_i, y_i) untuk $i = 1, \dots, N$, serta titik data baru (t_{N+1}, y_{N+1}) . Plot lintasan 2D dihasilkan dari evaluasi polinomial interpolasi $p_y(t)$ dengan derajat $N - 1$ dan polinomial hasil penambahan titik baru $p_{y,N}(t)$ dengan derajat N pada interval t yang rapat, yaitu setiap 0.1 untuk mendapatkan kurva yang halus.



Gambar 2.3.5.1: Plot lintasan 2D $p_y(t)$ dan $p_{y,N}(t)$

Berdasarkan grafik di atas, terlihat bahwa interpolasi derajat 9 mengalami lonjakan yang sangat tajam pada saat $t = 11$, sedangkan interpolasi derajat 10 menunjukkan perilaku yang lebih stabil di sekitar titik tersebut. Hal ini disebabkan karena pada interpolasi derajat 9, titik ke-11 belum dimasukkan ke dalam perhitungan, sehingga polinomial hanya berusaha menyesuaikan 10 titik awal dan tidak mengetahui adanya nilai $y_{11} = 135$.

Ketika dievaluasi di $t = 11$, polinomial derajat 9 melakukan ekstrapolasi, bukan interpolasi, yang cenderung menghasilkan nilai yang tidak stabil atau bahkan menyimpang jauh. Sebaliknya, pada interpolasi derajat 10, titik data ke-11 telah dimasukkan sebagai kendala tambahan, sehingga polinomial hasilnya melewati titik tersebut, membuat bentuk lintasan lebih konsisten dan stabil saat $t = 11$.

2.4 Interpolasi *Natural Cubic Splines* untuk $y(t)$ dan Evaluasi Hasil Interpolasi

2.4.1 *Natural Cubic Splines* untuk $y(t)$

Pendekatan alternatif untuk interpolasi adalah menggunakan Interpolasi *Spline*. Dalam laporan ini kita akan menggunakan *Natural Cubic Spline*. Kita definisikan $S_i(t)$ suatu polinomial orde-3 yang menginterpolasi data dengan ketentuan sebagai berikut:

- $S_i(t)$ menginterpolasi data: $S_i(t_i) = y_i$
- $S_i(t)$ kontinu pada titik interior: $S_i(t_{i+1}) = S_{i+1}(t_{i+1})$
- Turunan pertama $S_i(t)$ kontinu: $S'_i(t_{i+1}) = S'_{i+1}(t_{i+1})$
- Turunan kedua $S_i(t)$ kontinu: $S''_i(t_{i+1}) = S''_{i+1}(t_{i+1})$

Lalu ada tambahan syarat pada *Natural Cubic Spline*, yakni $S''_1(t_1) = S''_{n-1}(t_n) = 0$. Dalam kata lain $S''_1(1) = S''_{10}(11) = 0$.

Pada laporan ini kita modelkan $S_i(t)$ menjadi:

$$S_i(t) = a_i + b_i(t - t_i) + c_i(t - t_i)^2 + d_i(t - t_i)^3 \text{ Pada interval } [t_i, t_{i+1}]$$

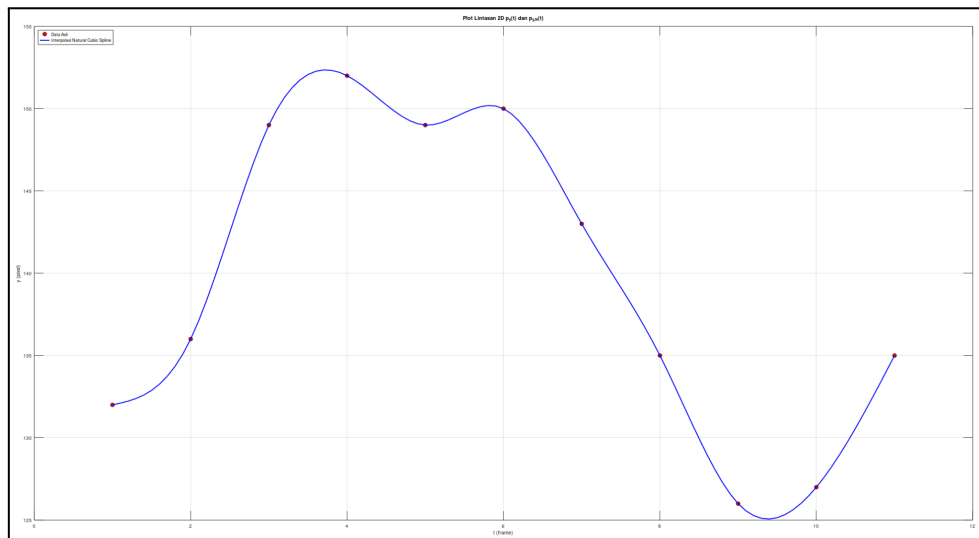
Karena kita interpolasikan dari 1 sampai 11, maka kita akan membuat 10 $S_i(t)$ sebagai berikut:

$S_1(t)$	$=$	a_1	$+$	$b_1(t - t_1)$	$+$	$c_1(t - t_1)^2$	$+$	$d_1(t - t_1)^3$	Pada interval $[t_1, t_2]$
$S_2(t)$	$=$	a_2	$+$	$b_2(t - t_2)$	$+$	$c_2(t - t_2)^2$	$+$	$d_2(t - t_2)^3$	Pada interval $[t_2, t_3]$
\vdots	$+$	\vdots	$+$	\vdots	$+$	\vdots	$+$	\vdots	\vdots
$S_{10}(t)$	$=$	a_{10}	$+$	$b_{10}(t - t_{10})$	$+$	$c_{10}(t - t_{10})^2$	$+$	$d_{10}(t - t_{10})^3$	Pada interval $[t_{10}, t_{11}]$

Kita cari nilai-nilai a , b , c , dan d menggunakan teorema 3.6 pada buku *Numerical Analysis 3rd Edition* oleh *Timothy Sauer*. Hasilnya sebagai berikut:

$S_1(t)$	$=$	132	$+$	$1.0175(t-1)$	$+$	$0(t-1)^2$	$+$	$2.9825(t-1)^3$	Pada interval $[1, 2]$
$S_2(t)$	$=$	136	$+$	$9.9649(t-2)$	$+$	$8.9474(t-2)^2$	$-$	$5.9123(t-2)^3$	Pada interval $[2, 3]$
$S_3(t)$	$=$	149	$+$	$10.1227(t-3)$	$-$	$8.7896(t-3)^2$	$+$	$1.6669(t-3)^3$	Pada interval $[3, 4]$
$S_4(t)$	$=$	152	$-$	$2.4559(t-4)$	$+$	$8.9474(t-4)^2$	$-$	$3.7890(t-4)^3$	Pada interval $[4, 5]$
$S_5(t)$	$=$	149	$-$	$0.2993(t-5)$	$+$	$5.9455(t-5)^2$	$-$	$4.6462(t-5)^3$	Pada interval $[5, 6]$
$S_6(t)$	$=$	150	$-$	$2.3469(t-6)$	$-$	$7.9931(t-6)^2$	$+$	$3.3400(t-6)^3$	Pada interval $[6, 7]$
$S_7(t)$	$=$	143	$-$	$8.3131(t-7)$	$+$	$2.0269(t-7)^2$	$-$	$1.7137(t-7)^3$	Pada interval $[7, 8]$
$S_8(t)$	$=$	135	$-$	$9.4006(t-8)$	$-$	$3.1143(t-8)^2$	$+$	$3.5149(t-8)^3$	Pada interval $[8, 9]$
$S_9(t)$	$=$	126	$-$	$5.0845(t-9)$	$+$	$7.4305(t-9)^2$	$-$	$1.3460(t-9)^3$	Pada interval $[9, 10]$
$S_{10}(t)$	$=$	127	$+$	$5.7384(t-10)$	$+$	$3.3924(t-10)^2$	$-$	$1.1308(t-10)^3$	Pada interval $[10, 11]$

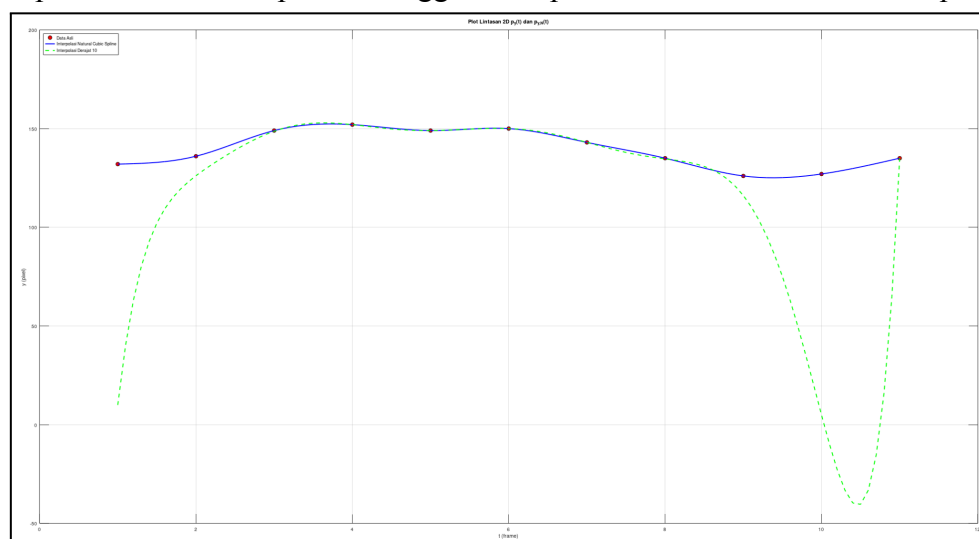
Alhasil kita bisa plotkan hasil interpolasi menggunakan *Natural Cubic Spline*, ini hasil plotnya:



Gambar 2.4.1.1: Plot hasil interpolasi Natural Cubic Spline

2.4.2 Perbandingan dan Analisis antara Interpolasi menggunakan Polinomial dan *Natural Cubic Spline*

Ini plot dari hasil interpolasi menggunakan polinomial dan *natural cubic spline*



Gambar 2.4.2.1: Plot lintasan 2D $p_y(t)$ dan $p_{y,N}(t)$

Dapat dilihat bahwa hasil interpolasi menggunakan *natural cubic spline* lebih stabil dibanding menggunakan polinomial. Terutama $10 \leq t \leq 11$ yang menandakan polinomial rentan terhadap penambahan data baru.

Pada permasalahan interpolasi, lebih baik menggunakan *Natural Cubic Spline* dibanding menggunakan polinomial karena:

1. Menghindari *Runge's Phenomenon* dan Osilasi Tidak Stabil

Runge's Phenomenon terjadi ketika interpolasi polinomial derajat tinggi digunakan pada titik data yang berjarak seragam. Polinomial cenderung menghasilkan osilasi liar (fluktuasi tak terduga) di dekat ujung interval, terutama jika data tidak mengikuti pola polinomial. Sedangkan pada *natural cubic spline*, *spline* membagi interval menjadi subinterval kecil, lalu menggunakan polinomial derajat 3 di setiap subinterval. Derajat rendah (*cubic*) menghindari osilasi yang muncul pada polinomial derajat tinggi.

2. Solusi Numerik yang Lebih Stabil dan Efisien

Menggunakan matriks Vandermonde untuk menentukan koefisien polinomial. Matriks ini menjadi *ill-conditioned* saat N besar karena elemen matriks tumbuh eksponensial, menyebabkan kesalahan pembulatan (*round-off error*) dominan. Belum algoritma-algoritma untuk menyelesaikan masalah tersebut memiliki flops terbaik $O(N^2)$. Sedangkan pada *natural cubic spline*, sistem persamaan liniernya berbentuk tridiagonal karena hanya melibatkan hubungan antara tiga titik knot yang berdekatan.

Membuat algoritma tersebut memiliki flops $O(N)$, lebih bagus dibanding $O(N^2)$.

3. Perilaku Kurva Lebih Natural dan Sesuai untuk Data Empiris

Polinomial global (derajat tinggi) memaksa kurva melalui semua titik data, tetapi tidak sensitif terhadap perubahan lokal. Jika data empiris memiliki variasi lokal (misalnya *noise* atau fluktuasi acak), polinomial akan berusaha "mengikuti" semua titik, menghasilkan kurva yang tidak wajar (*overfitting*). Sedangkan pada *natural cubic spline*, setiap segmen *cubic* hanya dipengaruhi oleh titik knot terdekat, sehingga perubahan lokal tidak mengganggu keseluruhan kurva. Turunan kedua yang kontinu memastikan kelengkungan (*curvature*) kurva halus di seluruh interval, menghindari perubahan tajam yang tidak fisikal.

3. KESIMPULAN

Melalui tugas ini, kami telah mengeksplorasi dan membandingkan dua pendekatan interpolasi numerik, yaitu interpolasi polinomial dan *natural cubic spline* untuk merekonstruksi gerakan vertikal (koordinat-y) dari titik kunci pada kaki seekor kuda berdasarkan data *frame* animasi.

Beberapa poin kesimpulan yang dapat diambil:

1. **Pemilihan Basis Berpengaruh pada Stabilitas**

Condition number dari matriks interpolasi sangat dipengaruhi oleh pemilihan basis polinomial. Basis yang lebih terpusat (misalnya $(t - t_{avg})^i$) terbukti memberikan *condition number* yang lebih kecil dan lebih stabil secara numerik dibanding basis monomial standar t^i .

2. **Interpolasi Polinomial Efisien Namun Kurang Stabil untuk Data Besar**

Interpolasi polinomial dengan derajat tinggi mampu merekonstruksi data dengan presisi tinggi di dalam rentang data, namun cenderung mengalami osilasi dan perilaku tidak stabil di luar rentang data (ekstrapolasi), terutama pada nilai $t > N$.

3. ***Natural Cubic Spline* Lebih Stabil dan Halus**

Interpolasi *natural cubic spline* menghasilkan lintasan yang lebih halus dan stabil, baik di dalam maupun sedikit di luar rentang data asli. Sifat lokal dari *spline* juga mencegah distorsi besar jika ada penambahan titik data baru.

4. **Rekomendasi**

Untuk keperluan animasi yang memerlukan prediksi gerakan halus dan stabil, terutama jika *frame* tambahan akan ditambahkan di luar data asli, metode *natural cubic spline* merupakan pendekatan yang lebih direkomendasikan dibanding interpolasi polinomial derajat tinggi.

Dengan pendekatan numerik ini, proses *in-betweening* untuk animasi dapat dilakukan secara efisien dan realistis tanpa perlu menggambar ulang *frame* satu per satu secara manual.

4. REFERENSI

- Basaruddin, T. (2025). Topic 2: System of Linear Equations [Lecture slides]. Fakultas Ilmu Komputer, Universitas Indonesia. SCELE Fakultas Ilmu Komputer Universitas Indonesia.
- Basaruddin, T. (2025). Topic 6: Interpolation [Lecture slides]. Fakultas Ilmu Komputer, Universitas Indonesia. SCELE Fakultas Ilmu Komputer Universitas Indonesia.
- Chandra. (2011, July 2). Using Horner's Method [Question post]. Mathematics StackExchange. [polynomials - Using Horner's Method - Mathematics Stack Exchange](#)
- Sauer, T. (2018). Numerical analysis (3rd ed.). Pearson.

LAMPIRAN

poly_first_basis()

```
function A = poly_first_basis()
    t = 1:10;
    t = t';
    A = zeros(10);
    for i=1:10
        for j=1:10
            A(i,j) = (t(i))^(j-1);
        endfor
    endfor
```

poly_second_basis()

```
function A = poly_second_basis()
    t=1:10;
    t=t';
    A = zeros(10);
    for i=1:10
        for j=1:10
            A(i, j) = (t(i) - 5.5)^(j-1);
        endfor
    endfor
```

poly_third_basis()

```
function A = poly_third_basis()
    t=1:10;
    t=t';
    A = zeros(10);
    for i=1:10
        for j=1:10
            A(i, j) = (t(i) - 10)^(j-1);
        endfor
    endfor
```

poly_fourth_basis()

```
function A = poly_fourth_basis()
    t=1:10;
    t=t';
    A = zeros(10);
    for i=1:10
        for j=1:10
```

```

    A(i, j) = (0.5*t(i) - 2.75)^(j-1);
endfor
endfor

```

get_y_inter()

```

function y = get_y_inter()
    y = [132 ; 136 ; 149 ; 152 ; 149 ; 150 ; 143 ; 135 ; 126 ; 127];

```

segitigaAtas(U, b)

```

function x = segitigaAtas(U, b)
    n = length(b);
    x = zeros(n, 1);
    x(n) = b(n)/U(n,n);
    for i = n-1:-1:1
        x(i) = (b(i)-U(i, i+1:n)*x(i+1:n))/U(i, i);
    endfor

```

EliminasiGauss(A, b)

```

function [U, bt] = EliminasiGauss(A, b)
    n = length(b);
    A = [A b]; # augmented dulu matrixnya

    for j = 1:n-1 # j untuk indexing kolom
        for i = j+1:n # i untuk indexing baris
            m = A(i, j)/A(j, j); # a2 = a2 - (a2/a1)a1 = 0
            # bekerja pada baris ke-i
            A(i,:) = A(i,:)-m*A(j,:); # after optimisasi A(i,j:n+1) = A(i,j:n+1) -
m*A(j,j:n+1)
                                # menghemat n^2/2 flops
                                # kita mulai update dr index j+1 karena diindex sebelum
j + 1 udh pasti 0
                                # U = triu(A(:,1:n))
                                # ambil triu karena sisanya junk

        endfor
    endfor

    U = A(:,1:n); # ambil semua baris dan n kolom pertama
    bt = A(:,n+1); # ambil semua baris dan bt berada pada kolom n+1

    # cara optimisasi tdk usah hitung kolom yang sudah pasti jadi 0
    # eliminasi gauss tdk rekomen utk matrix yang sangat besar

```


search_c()

```
function [c, e] = search_c()
    A = poly_fourth_basis();
    y = get_y_inter();

    [U, bt] = EliminasiGauss(A, y);

    c = segitigaAtas(U, bt);
    e = A*c - y;
```

py(t)

```
function y = py(t)
    x = (t-5.5)/2;
    y = ((((((0.65326 .* x+0.098413) .* x-6.8836) .* x-0.42778) .* x+23.526) .*
    x-0.43559) .* x-28.388) .* x-3.5954) .* x+3.6840) .* x+149.73;
```

pyN(t)

```
function y = pyN(t)
    x = (t-5.5)/2;
    y =
    (1.4973*10^2).*x.^0+3.6840.*x.^1+-3.5954.*x.^2+-2.8388*10.*x.^3+-4.3559*10^(-1).*x.^4+2.
    3526*10.*x.^5+-4.2778*10^(-1).*x.^6+-6.8836.*x.^7+9.8413*10^(-2).*x.^8+6.5326*10^(-1).*x
    .^9 - 0.0367.*x.^10;
```

eval_py()

```
function [t_vals, y1_vals] = eval_py()
    % Buat vektor t dari 1 hingga 10 dengan interval 0.1
    t_vals = 1:0.1:11;
    y1_vals = zeros(size(t_vals));

    % Evaluasi py(t) untuk setiap t
    for i = 1:length(t_vals)
        t = t_vals(i);
        y1 = py(t);
        y1_vals(i) = y1;
    end
```

eval_pyN()

```
function [t, y] = eval_pyN()
    % Buat vektor t dari 1 hingga 10 dengan interval 0.1
    t = 1:1:11;
    y = zeros(size(t));

    % Evaluasi pyN(t) untuk setiap t
    for i = 1:length(t)
        t_vals = t(i);
        y_vals = pyN(t_vals);
        y(i) = y_vals;
    end
end
```

plot_2D()

```
function plot_2D()
    % Data asli
    t_data = 1:11;
    y_data = [132 136 149 152 149 150 143 135 126 127 135];

    % Interval t rapat
    t_vals = 1:0.1:11;

    % Evaluasi interpolasi derajat 9 (N-1) dan 10 (N)
    y1_vals = py(t_vals);
    yN_vals = pyN(t_vals);

    % Plot titik-titik data
    figure;
    plot(t_data, y_data, 'ko', 'MarkerFaceColor', 'r', 'DisplayName', 'Data Asli'); hold on;

    % Plot interpolasi derajat 9 (N-1)
    plot(t_vals, y1_vals, 'b-', 'LineWidth', 1.5, 'DisplayName', 'Interpolasi Derajat 9');

    % Plot interpolasi derajat 10 (N)
    plot(t_vals, yN_vals, 'g--', 'LineWidth', 1.5, 'DisplayName', 'Interpolasi Derajat 10');

    % Label dan legenda
    xlabel('t (frame)');
    ylabel('y (pixel)');
    title('Plot Lintasan 2D p_y(t) dan p_{y,N}(t)');

    legend('Location', 'northwest');
    grid on;
end
```

cubic_spline(t, y)

```
function [a, b, c, d] = cubic_spline(t, y)
    n = length(t);
    a = b = c = d = dt = dy = zeros(n-1, 1);
    for i=1:n-1
        a(i) = y(i);
        dt(i) = t(i+1) - t(i);
        dy(i) = y(i+1) - y(i);
    endfor

    c = solve_c_for_cubic_spline(dt, dy);

    for i=1:n-1
        d(i) = (c(i+1) - c(i)) / (3 * dt(i));
        b(i) = dy(i)/dt(i) - dt(i)/3*(2*c(i) + c(i+1));
    endfor
```

solve_c_for_cubic_spline(dt, dy)

```
function c = solve_c_for_cubic_spline(dt, dy)
    n = length(dt);
    A = zeros(n+1);
    A(1, 1) = A(n+1, n+1) = 1;
    b = zeros(n+1, 1);
    for i=1:n-1
        A(i+1, i) = dt(i);
        A(i+1, i+1) = 2*dt(i) + 2*dt(i+1);
        A(i+1, i+2) = dt(i+1);
    endfor
    for i=2:n
        b(i) = 3*(dy(i)/dt(i) - dy(i-1)/dt(i-1));
    endfor
    [U, bt] = EliminasGauss(A, b);
    c = segitigaAtas(U, bt);
```

S_for_t(a, b, c, d, t_ori, t_pred)

```
function y = S_for_t(a, b, c, d, t_ori, t_pred)
    n = length(t_pred);
    y = zeros(n, 1);
```

```

j = 1;
for i=1:n
    while t_pred(i) > t_ori(j+1) && j < length(a)
        j = j+1;
    endwhile
    y(i) = a(j) + b(j)*(t_pred(i) - t_ori(j)) + c(j)*(t_pred(i) - t_ori(j))^2 +
d(j)*(t_pred(i) - t_ori(j))^3;
endfor

```

plot_2D_4_1()

```

function plot_2D_4_1()
% Data asli
t_data = 1:11;
y_data = [132 ; 136 ; 149 ; 152 ; 149 ; 150 ; 143 ; 135 ; 126 ; 127 ; 135];

% Interval t rapat
t_vals = 1:0.1:11;

% Evaluasi interpolasi menggunakan Natural Cubic Splines
[a, b, c, d] = cubic_spline(t_data, y_data);
yc_vals = S_for_t(a, b, c, d, t_data, t_vals);

% Plot titik-titik data
figure;
plot(t_data, y_data, 'ko', 'MarkerFaceColor', 'r', 'DisplayName', 'Data Asli'); hold
on;

% Plot interpolasi natural cubic spline
plot(t_vals, yc_vals, 'b-', 'LineWidth', 1.5, 'DisplayName', 'Interpolasi Natural
Cubic Spline');

% Label dan legenda
xlabel('t (frame)');
ylabel('y (pixel)');
title('Plot Lintasan 2D p_y(t) dan p_{y,N}(t)');

legend('Location', 'northwest');
grid on;
end

```

plot_2D_4_2()

```

function plot_2D_4_2()
% Data asli

```

```

t_data = 1:11;
y_data = [132 ; 136 ; 149 ; 152 ; 149 ; 150 ; 143 ; 135 ; 126 ; 127 ; 135];

% Interval t rapat
t_vals = 1:0.1:11;

% Evaluasi interpolasi derajat 9 (N-1) dan 10 (N)
yN_vals = pyN(t_vals);

% Evaluasi interpolasi menggunakan Natural Cubic Splines
[a, b, c, d] = cubic_spline(t_data, y_data);
yc_vals = S_for_t(a, b, c, d, t_data, t_vals);

% Plot titik-titik data
figure;
plot(t_data, y_data, 'ko', 'MarkerFaceColor', 'r', 'DisplayName', 'Data Asli'); hold
on;

% Plot interpolasi natural cubic spline
plot(t_vals, yc_vals, 'b-', 'LineWidth', 1.5, 'DisplayName', 'Interpolasi Natural
Cubic Spline');

% Plot interpolasi derajat 10 (N)
plot(t_vals, yN_vals, 'g--', 'LineWidth', 1.5, 'DisplayName', 'Interpolasi Derajat
10');

% Label dan legenda
xlabel('t (frame)');
ylabel('y (pixel)');
title('Plot Lintasan 2D p_y(t) dan p_{y,N}(t)');

legend('Location', 'northwest');
grid on;
end

```