Algoritma Repeated Gaussian Quadrature dalam Evaluasi Integral Numerik

Pakta Integritas

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

Rangkuman

Pada laporan ini, kami akan menggunakan algoritma Repeated Gaussian Quadrature Integral untuk menyelesaikan permasalahan integral numerik. Melalui algoritma tersebut, kami akan mendapatkan nilai integral numerik bagi suatu fungsi f dan akan dibandingkan dengan solusi eksak yang didapatkan. Kami juga membuktikan jumlah faktor pemilihan titik (*nodes*) dan subinterval terhadap akurasi algoritma ini. Tidak hanya itu, kami juga membuktikan *time complexity* dari algoritma ini, yaitu $O(mn^3)$ dan *space complexity* sebesar $O(n^2)$.

Daftar Isi

1 Pendahuluan

Fungsi adalah suatu hubungan matematika antara dua atau lebih variabel yang menggambarkan bagaimana nilai dari satu variabel dipengaruhi oleh nilai variabel lainnya. Secara matematis, fungsi dapat dinotasikan sebagai berikut,

$$f : \mathbb{R}^n \to \mathbb{R}^n$$

Fungsi dapat digunakan untuk memodelkan berbagai fenomena di dunia nyata seperti pertumbuhan populasi, perubahan suhu, dan pergerakan benda. Dalam beberapa kasus, kita mungkin ingin mengetahui nilai total dari suatu fungsi dalam suatu rentang tertentu. Misalnya, kita mungkin ingin mengetahui jumlah total energi yang dipancarkan oleh benda dalam rentang waktu tertentu atau jumlah total uang yang dihabiskan oleh konsumen dalam rentang harga tertentu. Untuk mengetahui nilai total ini, kita dapat menggunakan integral dari fungsi tersebut. Integral adalah operasi matematika yang digunakan untuk menghitung nilai total dari suatu fungsi dalam suatu rentang tertentu. Ada dua jenis integral yaitu integral tak tentu dan integral tentu. Integral tak tentu memberikan fungsi baru yang merupakan antiturunan dari fungsi asli sedangkan integral tentu memberikan nilai numerik yang merupakan nilai total dari fungsi asli dalam rentang tertentu.

Namun, tidak semua fungsi memiliki integral analitik yang dapat ditemukan dengan mudah. Dalam kasus seperti ini, kita dapat menggunakan metode integral numerik untuk menghitung nilai integral secara numerik. Integral Numerik adalah metode numerik yang digunakan untuk menghitung nilai integral dari suatu fungsi. Integral Numerik sangat berguna dalam menyelesaikan masalah matematika yang tidak dapat diselesaikan secara analitik. Salah satu metode yang digunakan dalam penyelesaian Integral Numerik adalah *Gaussian Quadrature*. Metode ini adalah metode penyelesaian numerik yang digunakan untuk menghitung nilai integral dari suatu fungsi dengan menggunakan interval yang ditetapkan.

Salah satu metode *Gaussian Quadrature* adalah metode *Repeated Gaussian Quadrature Integral*. Metode ini digunakan untuk menghitung nilai integral dari suatu fungsi dengan menggunakan titik-titik (*nodes*) tertentu pada interval tertentu. Titik-titik ini dipilih sedemikian rupa sehingga hasil perkiraan integralnya menjadi lebih akurat. Dengan menggunakan pendekatan berulang, algoritma *Repeated Gaussian Quadrature Integral* dapat meningkatkan akurasi perkiraan integral dengan menambah jumlah titik yang digunakan.

Dalam laporan ini, kami akan membahas tentang Integral Numerik menggunakan algoritma Repeated Gaussian Quadrature Integral. Kami akan membahas tentang definisi Integral Numerik, algoritma Gaussian Quadrature, dan algoritma *Repeated Gaussian Quadrature Integral*. Tidak hanya itu, pada laporan ini, akan diperlihatkan informasi lebih dalam seperti kompleksitas dan error yang dapat dihasilkan dalam komputasi. Oleh karena itu, melalui laporan ini, kami akan memberikan wawasan yang mendalam mengenai algoritma tersebut.

2 Pembahasan

2.1 Integrasi Numerik

Integrasi numerik merupakan teknik untuk menghitung nilai suatu integral yang bersifat definit dengan pendekatan numeris. Metode ini berguna untuk menghitung fungsi yang rumit atau sulit untuk diintegrasikan secara analitik. Terdapat beberapa metode dalam melakukan integrasi numerik, di antaranya seperti

metode Riemann, Newton-Cotes, Gaussian Quadrature, Adaptive Quadrature, dan lain sebagainya. Setiap metode memiliki kelebihan dan kekurangannya masing-masing, dimana terdapat metode-metode yang unggul dalam hal efisiensi proses komputasi, namun relatif lebih terbatas dalam hal akurasi, dan sebaliknya. Pada makalah ini, pembahasan akan difokuskan pada algoritma Repeated Gaussian Quadrature.

2.2 Algoritma

2.2.1 Algoritma Gaussian Quadrature

Algoritma Gaussian Quadrature merupakan salah satu algoritma yang dapat digunakan untuk melakukan integrasi secara numerik terhadap suatu fungsi. Secara umum, algoritma Gaussian Quadrature akan melakukan pemilihan titik dan bobot. Selanjutnya, akan dilakukan evaluasi terhadap fungsi yang diintegrasikan menggunakan titik-titik yang dipilih. Setelahnya, hasil evaluasi fungsi tersebut akan dikalikan dengan bobot yang dipilih sebelumnya, dimana setiap hasil perkalian fungsi dengan bobot akan dijumlahkan. Secara umum, formula dari pendekatan integrasi menggunakan Gaussian Quadrature adalah sebagai berikut:

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i) \tag{1}$$

di mana x_i adalah titik evaluasi dan w_i adalah bobotnya. Titik-titik yang dipilih merupakan akar dari suatu polinomial ortogonal tertentu. Sementara itu, bobot akan dipilih berdasarkan turunan dari polinomial ortogonal tersebut. Berdasarkan definisi ortogonalitas, polinomial ortogonal merupakan polinomial dengan inner product dari dua polinomial berbeda dalam himpunan/set polinomial bernilai nol. Beberapa polinomial ortogonal tersebut mencakup polinomial Legendre, polinomial Chebyshev, polinomial Laguerre, dan polinomial Hermite. Jika digunakan polinomial Legendre, titik x_i yang digunakan dalam Gaussian Quadrature adalah akar dari polinomial Legendre orde n, dengan bobot w_i yang dapat ditemukan dengan rumus:

$$w_i = \frac{2}{(1-x_i^2)[P_n'(x_i)]^2} \tag{2}$$

dimana $P_n(x)$ adalah polinomial Legendre orde n dan $P_n'(x)$ adalah turunan pertamanya. Untuk melakukan Gaussian Quadrature, kita perlu menghitung akar dan bobot ini.

Dalam hal tingkat presisi, metode gaussian quadrature memiliki degree of precision sebesar $2n-1$ ketika n titik digunakan. Dengan kata lain, Gaussian Quadrature akan memberikan hasil yang tepat untuk integral polinomial hingga derajat $2n-1$. Oleh karena itu, jika kita menggunakan dua buah titik untuk melakukan integrasi menggunakan Gaussian Quadrature, maka integrasi dapat dihasilkan dengan tepat untuk setiap fungsi polinomial dengan derajat kurang dari tiga. Namun, tingkat presisi tersebut akan berkurang jika kita melakukan integrasi pada fungsi polinomial yang memiliki derajat lebih besar dari $2n-1$ atau fungsi non-polinomial.

Berikut adalah *pseudocode* dari **Gaussian Quadrature** yang mengambil sebuah fungsi f dan interval $[a,b]$ serta n buah *node* yang akan digunakan.

Algorithm 2.1 Gaussian Quadrature

function GAUSSIANQUADRATURE(f,a,b,n)
    $\beta \leftarrow 0.5./\sqrt{1-(2\cdot(1:(n-1)))^{-2}}$ ▷ Menghitung array beta untuk bobot dan simpul
    $V,D \leftarrow$ eig(diag(β, 1) + diag(β, -1)) ▷ Menggunakan nilai dan vektor eigen dari matriks dengan diagonal beta untuk menghasilkan bobot dan simpul
    $nodes \leftarrow$ diag(D) ▷ Mengambil nilai eigen sebagai simpul
    $weights \leftarrow 2\cdot V(1,:)^2$ ▷ Mengambil kuadrat dari baris pertama dari vektor eigen, kemudian dikalikan 2 untuk mendapatkan bobot
    $nodes \leftarrow nodes'$ ▷ Mengubah simpul menjadi vektor kolom
    $weights \leftarrow weights'$ ▷ Mengubah bobot menjadi vektor kolom
    $G \leftarrow 0$ ▷ Menginisialisasi pendekatan integral menjadi 0
    for $i = 1$ to n **do**
        $x \leftarrow (b-a)\cdot nodes(i)/2+(b+a)/2$ ▷ Mentransformasikan simpul ke interval [a, b]
        $G \leftarrow G+weights(i)\cdot f(x)$ ▷ Menghitung pendekatan integral dengan mengalikan bobot dengan nilai fungsi pada simpul yang ditransformasikan dan menambahkannya ke total
    end for
    $G \leftarrow G\cdot(b-a)/2$ ▷ Menskalakan hasil dengan lebar interval
    return G ▷ Mengembalikan pendekatan integral
end function

Pertama-tama, algoritma ini menghitung array beta yang digunakan untuk menghasilkan *weight* dan *nodes*. Kemudian, algoritma ini menggunakan nilai dan vektor eigen dari matriks dengan diagonal beta untuk menghasilkan *weight* dan *node*. *Weight* dihitung dengan mengambil kuadrat dari baris pertama dari vektor eigen dan kemudian dikalikan 2. *Node* diambil sebagai nilai eigen.

Setelah *weight* dan *node* dihitung, algoritma ini menginisialisasi pendekatan integral menjadi 0. Kemudian, untuk setiap *node*, algoritma ini mentransformasikan *node* ke interval $[a,b]$ dan menghitung pendekatan integral dengan mengalikan *weight* dengan nilai fungsi pada *node* yang ditransformasikan dan menambahkannya ke total. Akhirnya, hasilnya diskalakan dengan lebar interval dan dikembalikan sebagai pendekatan integral.

2.2.2 Algoritma Repeated

Algoritma Repeated merupakan metode numerik yang digunakan untuk memperkirakan hasil dari definite integral. Caranya adalah dengan membagi interval integrasi menjadi subinterval yang lebih kecil. Dari setiap bagian tersebut, diterapkan suatu aturan yang disebut composite rules atau aturan komposit. Alasan utamanya adalah karena fungsi yang sedang diintegralkan mungkin mengalami perubahan yang signifikan di berbagai bagian rentang bilangan tersebut. Dengan membagi rentang bilangan menjadi bagian-bagian yang lebih kecil, kita dapat melihat perubahan yang lebih halus dalam fungsi tersebut.

Composite rules memanfaatkan polinomial interpolasi pada setiap bagian rentang bilangan untuk mendekati fungsi yang sedang diintegralkan. Dengan menggunakan polinomial interpolasi yang memiliki derajat yang lebih rendah pada setiap bagian rentang bilangan, composite rules dapat memberikan perkiraan yang lebih baik dibandingkan dengan menggunakan polinomial interpolasi tunggal di seluruh rentang bilangan secara keseluruhan. Dengan menerapkan composite rules pada setiap bagian rentang bilangan, maka dapat didapatkan perkiraan integral yang lebih akurat. Hasil dari setiap bagian rentang bilangan digabungkan (diakumulasi) untuk mendapatkan perkiraan integral dari seluruh interval.

Dalam prakteknya, penggunaan *composite rules* pada setiap bagian interval memungkinkan untuk menyesuaikan tingkat akurasi perkiraan dengan memilih jumlah bagian interval yang lebih banyak atau menggunakan composite rules dengan tingkat keakuratan yang lebih tinggi jika diperlukan. Hal ini memungkinkan untuk didapatkan hasil yang lebih akurat sesuai dengan kebutuhan aplikasi atau tingkat ketelitian yang diinginkan.

2.2.3 Algoritma Repeated Gaussian Quadrature

Algoritma Repeated Gaussian Quadrature merupakan gabungan dari algoritma Repeated dan algoritma Gaussian Quadrature. Algoritma ini dapat mengintegrasikan fungsi yang rumit atau memiliki pola yang berubah-ubah dengan menangkap detail yang baik. Hal tersebut dikarenakan terdapat penentuan titik serta bobot yang diterapkan pada fungsi yang dievaluasi. Bobot yang dipilih akan dibuat sedemikian rupa sehingga hasil evaluasinya akan mendekati integral fungsi. Selain itu, karena algoritma ini juga mengadopsi konsep repeated, atau dengan kata lain membuat metode integrasi Gaussian Quadrature yang diterapkan akan berada pada rentang nilai yang lebih kecil, maka perilaku fungsi yang berubah-ubah atau bervariasi akan lebih terdeteksi. Berikut merupakan formula matematis dari metode integrasi menggunakan metode Repeated Gaussian Quadrature:

$$\int_a^b f(x)dx \approx \sum_{j=1}^{m} \left[\int_{x_{j-1}}^{x_j} f(x)dx \right] \tag{3}$$

Dengan menggunakan polinomial Legendre, dapat digunakan transformasi linear untuk mengubah setiap subinterval ke interval $[-1,1]$, dan kemudian menerapkan Gaussian Quadrature. Sehingga, formula matematis Repeated Gaussian Quadrature akan berbentuk seperti berikut:

$$\int_a^b f(x)dx \approx \sum_{j=1}^{m} \left[\sum_{i=1}^{n} w_{ij} f(x_{ij}) \right] \tag{4}$$

dimana w_{ij} adalah bobot dan x_{ij} adalah titik pada subinterval ke-j.

Meskipun begitu, algoritma Repeated Gaussian Quadrature juga memiliki beberapa *drawbacks* atau kerugian yang dapat ditimbulkan. Misalnya, algoritma ini memungkinkan proses komputasi yang relatif lebih besar jika digunakan jumlah subinterval dan/atau titik yang juga besar. Selain itu, meskipun

dapat melihat perilaku fungsi secara lebih detail, algoritma ini juga relatif kurang efektif untuk mendeteksi diskontinuitas jika dibandingkan dengan algoritma adaptif. Hal ini dikarenakan algoritma Repeated Gaussian Quadrature hanya membagi interval ke dalam beberapa subinterval, kemudian melakukan integrasi menggunakan metode Gaussian Quadrature pada setiap subinterval tersebut tanpa beradaptasi dengan perilaku fungsi di dalam subinterval. Hal tersebut juga dapat kita lihat dengan jumlah subinterval dan jumlah titik yang digunakan dalam algoritma Repeated Gaussian Quadrature, dimana keduanya ditentukan sejak awal dan tidak berubah selama proses komputasi. Sementara itu, pada algoritma adaptif, jumlah subinterval serta titik yang digunakan dapat berubah seiring berjalannya waktu. Algoritma-algoritma adaptif akan menyesuaikan diri dengan perilaku dari fungsi, termasuk jika terjadi diskontinuitas.

Sesuai dengan formula yang telah diberikan sebelumnya, secara umum cara kerja algoritma Repeated Gaussian Quadrature adalah sebagai berikut:

1. Pertama, akan dilakukan pemilihan terhadap banyaknya subinterval. Banyak subinterval tersebut akan menentukan besarnya lebar setiap subinterval yang akan diintegrasikan.
2. Selanjutnya, dapat dilakukan perulangan sebanyak jumlah subinterval untuk melakukan integrasi menggunakan Gaussian Quadrature, di mana hasil integrasi setiap subinterval akan terus diakumulasikan hingga diperoleh suatu nilai yang menggambarkan hasil integrasi terhadap keseluruhan interval.

2.3 Implementasi Algoritma Repeated Gaussian Quadrature

Berikut adalah *pseudocode* algoritma Repeated Gaussian Quadrature yang diimplementasikan berdasarkan cara kerja diatas. Algoritma ini akan mengevaluasi pendekatan integral dari suatu fungsi f pada interval $[a,b]$ dengan menggunakan metode Gaussian Quadrature yang diulang sebanyak m kali pada subinterval yang sama lebar. Setiap subinterval dihitung dengan menggunakan metode Gaussian Quadrature dengan n titik.

Algorithm 2.2 Repeated Gaussian Quadrature

Output: G: Total pendekatan integral

```
1: function REPEATEDGAUSSIANQUADRATURE(f, a, b, m, n)
2:     h ← (b − a)/m                          ▷ Menghitung lebar setiap subinterval
3:     G ← 0                                  ▷ Menginisialisasi pendekatan integral menjadi 0
4:     for i = 0 to m − 1 do
5:         a_i ← a + i · h                    ▷ Menentukan awal setiap subinterval
6:         b_i ← a_i + h                      ▷ Menentukan akhir setiap subinterval
7:         G ← G+ GAUSSIANQUADRATURE(f, a_i, b_i, n)   ▷ Menambahkan pendekatan
   integral dari subinterval saat ini ke total
8:     end for
9:     return G                               ▷ Mengembalikan total pendekatan integral
10: end function
```

Dalam algoritma ini, lebar setiap subinterval dihitung terlebih dahulu dengan membagi selisih antara b dan a dengan jumlah subinterval m. Kemudian, pendekatan integral diin-

isialisasi menjadi 0. Untuk setiap subinterval, awal dan akhir subinterval dihitung dan pendekatan integral dari subinterval tersebut dihitung dengan menggunakan algoritma **Gaussian Quadrature** diatas. Hasilnya kemudian ditambahkan ke total pendekatan integral. Setelah semua subinterval selesai dihitung, total pendekatan integral dikembalikan sebagai hasil akhir.

2.4 Kemampuan Algoritma Repeated Gaussian Quadrature

Kemampuan algoritma Repeated Gaussian Quadrature terletak pada akurasi dan efektivitasnya dalam menghitung integral numerik. Berikut adalah beberapa keunggulan yang didapat ketika menggunakan algoritma ini:

- Akurasi: Repeated Gaussian Quadrature dapat memberikan hasil yang akurat untuk berbagai jenis fungsi, terutama yang memiliki perilaku kompleks, perubahan cepat, atau singularitas. Dengan menggunakan titik sampel dan bobot yang dipilih dengan sesuai, algoritma ini dapat menangkap nuansa fungsi dan menghasilkan perkiraan integral yang presisi.
- Efisiensi dalam hal konvergensi: Dibandingkan dengan beberapa metode integrasi numerik lainnya, seperti aturan trapesium atau aturan Simpson, repeated Gaussian quadrature dapat konvergen ke solusi yang benar dengan relatif cepat. Dengan jumlah titik sampel yang tepat, algoritma ini dapat memberikan hasil yang akurat sambil meminimalkan usaha komputasi. Meskipun begitu, hal ini tidak membuat algoritma ini relatif lebih efisien dalam hal komputasi di segala kasus, di mana jika digunakan subinterval dan/atau jumlah titik yang besar, maka komputasi yang dibutuhkan pun semakin tinggi.
- Integrasi fungsi dengan osilasi tinggi: Repeated Gaussian Quadrature sangat cocok untuk mengintegrasikan fungsi-fungsi dengan osilasi tinggi atau perubahan cepat. Titik sampel dan bobot yang digunakan dirancang untuk menangkap perilaku fungsi secara akurat, sehingga memungkinkan integrasi yang baik bahkan dalam kasus yang cukup rumit. Namun, sebagaimana yang telah disebutkan pada bagian sebelumnya, metode ini mungkin tidak efektif untuk fungsi dengan diskontinuitas atau perilaku yang ekstrim lainnya.

  Dengan keuntungan yang dapat diperoleh melalui metode Repeated Gaussian Quadrature, perlu diperhatikan pula bahwa metode ini akan bergantung pada karakteristik spesifik dari fungsi yang akan diintegralkan. Beberapa skenario, seperti integral pada interval tak terbatas atau fungsi dengan perilaku yang sangat tajam, mungkin memerlukan teknik khusus atau metode numerik alternatif. Secara keseluruhan, kemampuan Repeated Gaussian Quadrature terletak pada akurasi dan efisiensinya dalam menghitung integral numerik, terutama untuk fungsi-fungsi dengan perilaku kompleks.

2.5 Pengujian Algoritma Repeated Gaussian Quadrature

Untuk menguji algoritma *Repeated Gaussian Quadrature*, kami akan menggunakan fungsi komposit sebagai basis dari analisis kami.

$$y = 1 + sin(e^x)$$

Kemudian kami akan memilih *m* subinterval dan *n* node sebanyak 5. Sebagai perbandingan kami juga akan memilih 1 kasus lagi dimana *m* dan *n* lebih besar dari sebelumnya, yaitu 20.

2.5.1 Pemilihan Interval

Pada laporan ini, kami akan memilih 5 interval yang akan kami gunakan untuk menguji akurasi dari algoritma *Repeated Gaussian Quadrature*. Pemilihan interval ini kami pilih dengan beberapa konsiderasi seperti nilai x pada e^x, dan luas interval yang ada. Interval tersebut adalah sebagai berikut:

a	*b*	*m*	*n*
0	10	5	5
0	10	20	20
-1	1	5	5
-1	1	20	20
50	250	5	5
50	250	20	20
1e-2	1e-3	5	5
1e-2	1e-3	20	20
1e-4	2e-4	5	5
1e-4	2e-4	20	20

2.5.2 Langkah Algoritma

Langkah ini merupakan ilustrasi untuk input a = 0, b = 10, m = 5, dan n = 5.

1. Pertama-tama, akan dilakukan pencarian terhadap panjang setiap subinterval dengan membagi panjang interval (b-a) dengan banyak subinterval (m). Dari langkah ini, diperoleh h (panjang setiap subinterval) = 10/5 = 2. Selain itu, diinisialisasikan G = 0 sebagai nilai awal integrasi.
2. Kemudian, algoritma akan masuk ke dalam iterasi for loop yang pertama, dimana akan dilakukan:
   - perhitungan start subinterval pertama (a_1), yakni a + i * h = 0 + 0 * 2 = 0.
   - perhitungan akhir subinterval pertama (b_1), yakni a_1 + h = 0 + 2 = 2.
   - Selanjutnya, fungsi yang diintegrasikan, start dan akhir subinterval pertama, dan jumlah titik/nodes yang diinginkan akan dimasukkan sebagai input dalam fungsi GaussianQuadrature.
3. Di dalam fungsi GaussianQuadrature, akan dilakukan proses sebagai berikut:
   Dicari nilai beta berdasarkan jumlah titik/nodes yang digunakan dalam GaussianQuadrature. Dalam hal ini, digunakan n = 5. Dengan formula beta = 0.5 ./ sqrt(1 - (2 * (1:n-1)).$^{(}$ − 2)), diperoleh nilai beta = [0.5774 0.5164 0.5071 0.5040].
   Selanjutnya, dicari eigenvector dan eigenvalues berdasarkan nilai-nilai pada diag(beta,1) + diag(beta,-1), dimana diperoleh:

$$V = \begin{pmatrix} -3.44 \cdot 10^{-1} & 4.89 \cdot 10^{-1} & 5.33 \cdot 10^{-1} & -4.89 \cdot 10^{-1} & 3.4419 \cdot 10^{-1} \\ 5.40 \cdot 10^{-1} & -4.56 \cdot 10^{-1} & -1.52 \cdot 10^{-16} & -4.56 \cdot 10^{-1} & 5.40 \cdot 10^{-1} \\ -5.63 \cdot 10^{-1} & -7.11 \cdot 10^{-1} & -5.96 \cdot 10^{-1} & 7.11 \cdot 10^{-2} & 5.63 \cdot 10^{-1} \\ 4.56 \cdot 10^{-1} & 5.40 \cdot 10^{-1} & -2.48 \cdot 10^{-16} & 5.40 \cdot 10^{-1} & 4.56 \cdot 10^{-1} \\ -2.53 \cdot 10^{-1} & -5.05 \cdot 10^{-1} & 6.00 \cdot 10^{-1} & 5.05 \cdot 10^{-1} & 2.53 \cdot 10^{-1} \end{pmatrix}$$

$$D = \begin{pmatrix} -9.06 \cdot 10^{-1} & 0 & 0 & 0 & 0 \\ 0 & -5.38 \cdot 10^{-1} & 0 & 0 & 0 \\ 0 & 0 & -2.52 \cdot 10^{-16} & 0 & 0 \\ 0 & 0 & 0 & -5.38 \cdot 10^{-1} & 0 \\ 0 & 0 & 0 & 0 & -9.06 \cdot 10^{-1} \end{pmatrix}$$

Lalu, akan diambil nodes yang merupakan elemen-elemen pada diagonal utama di D.
Sementara itu, Weights/bobot diambil dengan formula berikut: weights = 2 * V(1, :).^{2};
Sehingga diperoleh:

$$\text{nodes} = \begin{pmatrix} -9.0618 \cdot 10^{-1} \\ -5.3857 \cdot 10^{-1} \\ -2.5217 \cdot 10^{-16} \\ -5.3857 \cdot 10^{-1} \\ -9.0618 \cdot 10^{-1} \end{pmatrix}$$

$$\text{weights} = \begin{pmatrix} 0.2369 & 0.4786 & 10.5689 & 0.4786 & 0.2369 \end{pmatrix}$$

Kemudian, diinisialisasikan sebuah variabel G yang bernilai 0. Selanjutnya, akan dilakukan iterasi sebanyak n (jumlah nodes yang dipakai dalam GaussianQuadrature) untuk melakukan proses seperti berikut:
- Mencari nilai x untuk dimasukkan sebagai input fungsi, dimana x dapat dicari dengan formula berikut:
x = (b - a) * nodes(i) / 2 + (b + a) / 2
x = (2-0) * (-9.0618 · 10$^-$1) / 2 + (2-0) /2
x = 0.09328

Kemudian, x tersebut dimasukkan sebagai input pada fungsi f, kemudian diakumulasikan ke dalam variabel G.
G = G + weights(i) * f(x); Pada iterasi pertama, didapat hasil akhir G = 0.4479.
Kemudian, dengan langkah yang sama dengan iterasi sebelumnya, diperoleh nilai-nilai seperti berikut: x = 0.4615; G = 1.4051

x = 1.0000; G = 2.2077
x = 1.5385; G = 2.2084
x = 1.9062; G = 2.5471

Dari hasil G = 2.5471, akan direscale ulang nilai G tersebut (yang juga menjadi 2.5471) berdasarkan panjang interval, lalu hasilnya akan dikembalikan sebagai output fungsi GaussianQuadrature. Sehingga, pada akhir iterasi 1,

nilai G akan sama dengan 2.5471.

2.5.3 Hasil Keseluruhan Interval

Berikut adalah hasil keseluruhan dari 5 interval yang dipilih diatas. Perhatikan bahwa kolom G merupakan kolom hasil integral yang ditemukan dengan a, b, m, n yang ditentukan diatas dan kolom E merupakan hasil integral yang sebenarnya. Fungsi f yang dipilih tetap menggunakan fungsi yang disebutkan diatas. Berikut adalah hasil yang diperoleh:

Table 1: Keseluruhan Hasil Integral Numerik

a	b	m	n	G	E
0	10	5	5	8.2945	10.6247
0	10	20	20	10.467	
-1	1	5	5	3.4559	3.4559
-1	1	20	20	3.4559	
50	250	5	5	193.24	248
50	250	20	20	225.92	
1e-2	1e-3	5	5	0.0166	0.01659
1e-2	1e-3	20	20	0.0166	
1e-4	2e-4	5	5	1.8416e-4	1.8416e-4
1e-4	2e-4	20	20	1.8416e-4	

2.5.4 Analisa Error Numerik

Pada bagian ini, kami akan memberikan grafik **Grouped Barchart** untuk memperlihatkan akurasi dari algoritma *Repeated Gaussian Quadrature* ini. Oleh karena itu, perlu diberikan informasi bahwa kami akan membandingkan apakah nilai m (jumlah subinterval) dan n (jumlah *node*) dapat menjadi faktor yang mempengaruhi akurasi hasil algortima ini atau tidak. Berikut adalah grafik yang kami peroleh dari tabel diatas. Dapat dilihat bahwa melalui grafik dan juga tabel yang diberikan, ternyata penambahan jumlah *node* (n) dan subinterval (m) dapat mempengaruhi akurasi dari algoritma ini. Ternyata, dapat dilihat bahwa kedua hal tersebut merupakan faktor yang mempengaruhi akurasi dari algoritma ini. Dengan menambahkan n (jumlah *node*) maupun n (jumlah subinterval) akan memberikan akurasi yang lebih baik. Apabila kedua faktor tersebut ditinggikan, maka akan diperoleh hasil dengan error yang lebih minimal daripada sebelumnya.

2.6 Analisis Kompleksitas Algoritma Repeated Gaussian Quadrature

2.6.1 Analisis Floating Point Operations (FLOPs)

Analisis kompleksitas pada algoritma Repeated Gaussian Quadrature dilakukan dengan menganalisis floating point operations di dalam algoritma tersebut. Pada analisis berikut, operasi assignment dikonsiderasi tidak masuk ke dalam Floating Point Operations (FLOPs).

Untuk fungsi GaussianQuadrature:

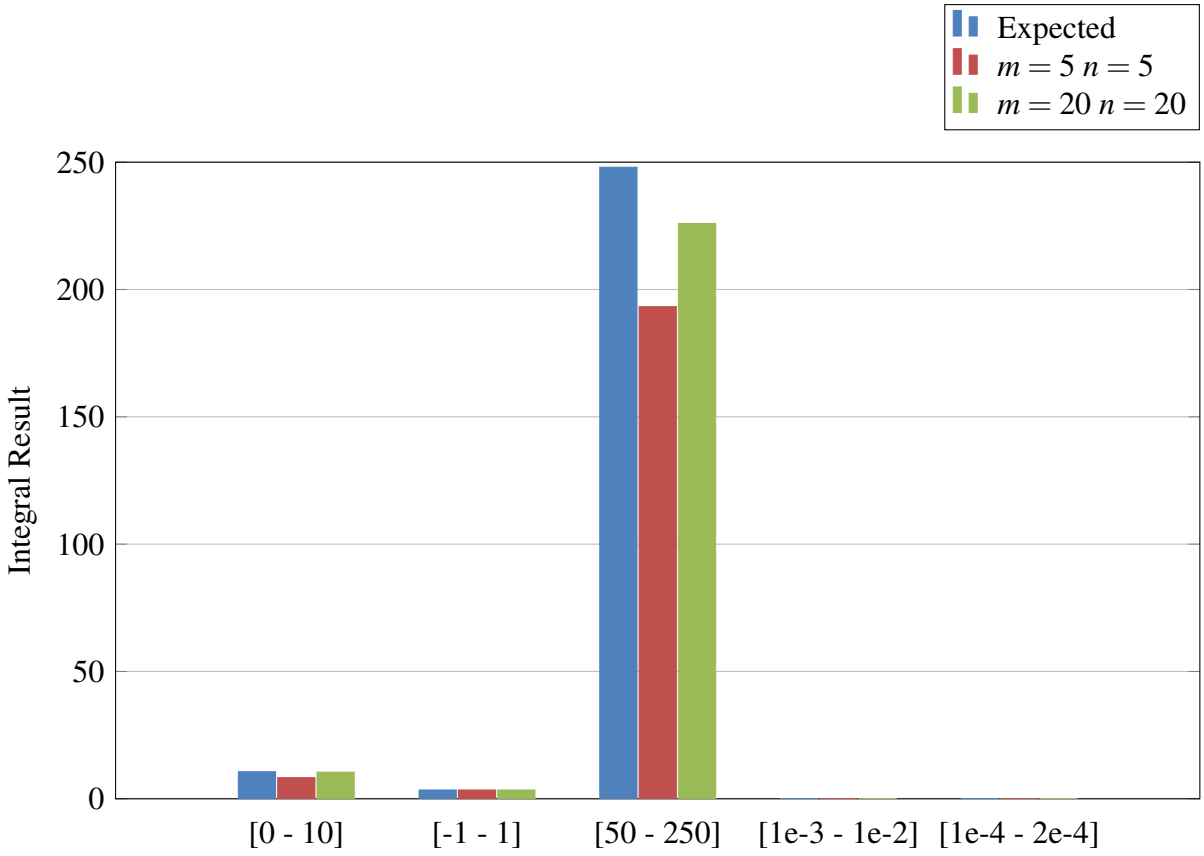

Figure 1: Akurasi Hasil Algoritma

Pada fungsi GaussianQuadrature, terdapat sebanyak 4 input yakni f (fungsi yang ingin diintegrasikan), a (batas bawah interval), b (batas atas interval), serta n(jumlah nodes/titik yang digunakan).

1. Pertama-tama, akan dilakukan perhitungan terhadap nilai beta yang mengandung 2 buah operasi perkalian, 1 buah pengurangan, 1 buah operasi kuadrat, dan 1 buah operasi pembagian, dimana hal ini dilakukan sebanyak n kali. Dengan begitu, diperoleh FLOPs sebesar 5n.
2. Selanjutnya, dilakukan perhitungan eigenvalues dan eigenvector dari matriks, yang biasanya memiliki kompleksitas sebanding dengan O(n^3) karena mengandung full-eigen decomposition. Namun, kompleksitas dari perhitungan ini juga tergantung pada matriks yang menjadi input, sehingga dibutuhkan faktor konstanta E sebagai pengali pada FLOPsnya. FLOPs : En^3
3. Setelah itu, dilakukan pengambilan eigenvalues sebagai nodes, dimana juga dilakukan perhitungan terhadap bobot sekaligus. Operasi ini mengandung 1 buah operasi pemangkatan dan 1 buah operasi perkalian, yang dilakukan sebanyak n kali. Oleh karena itu, pada tahap ini total FLOPs yang dibutuhkan ialah sebesar 2n.
4. Selanjutnya, dilakukan iterasi for loop sebanyak n kali, dimana di dalamnya dilakukan operasi seperti berikut:
    1. Menghitung nilai x : dilakukan 1 operasi pengurangan, 2 operasi penjumlahan, 1 operasi perkalian, dan 2 operasi pembagian. Total FLOPs: 6

2. Mengakumulasi nilai G: FLOPs akan bergantung pada fungsi yang dicari. Diasumsikan fungsi tersebut memiliki FLOPs sebesar P, ditambah dengan 1 buah operasi perkalian dan 1 buah operasi penjumlahan, maka FLOPs yang diperlukan ialah sebesar P + 2.

Karena operasi di atas dijalankan sebanyak n kali iterasi, maka total flops keseluruhan operasi di dalam loop tersebut ialah sebesar: n * (6 + P + 2) = Pn + 8n.

Di akhir algoritma, dilakukan penskalaan hasil G dengan lebar interval, dimana dibutuhkan 1 operasi perkalian, 1 operasi pengurangan, dan 1 operasi pembagian. Pada tahap ini dibutuhkan sebesar 3 FLOPs. Secara keseluruhan, algoritma GaussianQuadrature akan membutuhkan FLOPs sebesar: 5n + En^3 + 2n + Pn + 8n = En^3 + 15n + Pn + 3

Untuk fungsi RepeatedGaussianQuadrature:

Pada fungsi RepeatedGaussianQuadrature, terdapat sebanyak 5 yakni f (fungsi yang ingin diintegrasikan), a (batas bawah interval), b (batas atas interval), m (banyak interval direntang), serta n (jumlah nodes/titik yang digunakan)

Pada baris pertama, melibatkan 1 operasi pengurangan dan 1 pembagian yang semuanya merupakan operasi O(1). Operasi tersebut untuk menghitung nilai h. Sehingga, jumlah FLOPs: 2.

Setelah itu, melakukan iterasi pada loop sebanyak ‘m‘ kali:

1. Untuk mencari awal subinterval, dilakukan 1 operasi perkalian dan 1 operasi penjumlahan yang masing-masing membutuhkan 1 FLOPs O(1). Jumlah FLOPs: 2
2. Untuk mencari akhir subinterval, juga dilakukan 1 operasi perkalian dan 1 operasi penjumlahan yang masing-masing membutuhkan 1 FLOPs O(1). Jumlah FLOPs: 2
3. Dilakukan juga pemanggilan fungsi **GaussianQuadrature(f, a_i, b_i, n)**, dari keterangan sebelumnya fungsi Gauss membutuhkan FLOPs sebesar En^3 + 15n + Pn + 3.

Jadi, kompleksitas total dari fungsi **RepeatedGaussianQuadrature** adalah sebesar 2 + m(2 + 2 + En^3 + 15n + Pn + 3) atau jika disederhanakan akan menjadi O(mn^3).

2.6.2 Analisis Space Complexity

Space complexity merupakan kompleksitas penggunaan ruang memori yang diperlukan oleh suatu algoritma, dimana hal ini sangat berkaitan erat dengan input yang diberikan pada algoritma itu sendiri. Pada algoritma RepeatedGaussianQuadrature, berikut merupakan analisis space complexity yang dimilikinya:

1. Pada input, terdapat skalar a, b, m, dan n yang masing-masing membutuhkan ruang konstan sebesar O(1). Selain itu, terdapat fungsi f yang bergantung pada implementasi fungsi yang ingin diintegrasikan. Pada kasus fungsi y = 1 + sin(e^x), space complexity yang diperlukan adalah O(1).
2. Pada proses komputasi di dalam fungsi, terdapat beberapa variabel yang digunakan, di antaranya adalah h, G, a_i, b_i, dan x yang masing-masing memiliki

space complexity O(1); array beta yang memiliki $n-1$ elemen sehingga memerlukan space complexity $O(n)$; matriks V dan D yang merupakan output dari fungsi eig yang menghasilkan matriks berukuran $n-1$ x $n-1$, sehingga masing-masing V dan D membutuhkan space complexity sebesar O(n^2); serta array nodes dan weights yang masing-masing memiliki n elemen sehingga memiliki space complexity sebesar $O(n)$.

Maka, total *space complexity* yang dimiliki oleh algoritma RepeatedGaussianQuadrature adalah sebesar O(n^2).

3 Kesimpulan

Algoritma Repeated Gaussian Quadrature merupakan gabungan dari algoritma Repeated dan algoritma Gaussian Quadrature. Algoritma ini dapat mengintegrasikan fungsi yang rumit atau memiliki pola yang berubah-ubah dengan menangkap detail yang baik. Algoritma ini dimulai dengan memilih jumlah titik (*node*) dan pemilihan jumlah subinterval. Selanjutnya, dilakukan tahap *Gaussian Quadrature* pada setiap subinterval yang ada. Terakhir, semua hasil yang diperoleh dari tahap *Gaussian Quadrature*, ditambahkan dan diperoleh evaluasi dari integral numerik tersebut. Error yang dihasilkan oleh algoritma ini, akan bergantung dengan jumlah titik (*nodes*) dan jumlah subinterval yang ditentukan. Semakin besar kedua hal tersebut, maka akan semakin akurat hasil integrasi numerik tersebut.

Tidak hanya itu, kompleksitas waktu algoritma *Repeated Gaussian Quadrature* tentu saja akan lebih tinggi dibandingkan normal Gaussian Quadrature karena pembagian subinterval yang lebih banyak. Tentu saja, algoritma ini digunakan pada saat memprioritaskan akurasi dan bukan kecepatan tinggi. Algoritma ini memiliki kompleksitas waktu sebesar $O(mn^3)$ dan *space complexity* sebesar $O(n^2)$.

Oleh karena itu, keuntungan dari *Repeated Gaussian Quadrature* adalah pada akurasinya. Apabila mementingkan akurasi, maka pilihlah titik (*nodes*) atau jumlah subinterval yang lebih besar. Sebaliknya, apabila mementingkan kecepatan, maka pilihlah yang lebih kecil.

4 Referensi

Sauer, T. (2011). *Numerical analysis. Addison-Wesley Publishing Company.*

Press, W. H., Teukolsky, S. A., Vetterling, W. T., Flannery, B. P. (2007). *Numerical Recipes 3rd Edition: The Art of Scientific Computing. Cambridge University Press.*

5 Lampiran

5.1 Lampiran 1: Fungsi `f`

```
function f = f(x)
    f = 1 + sin(exp(x));
end
```

5.2 Lampiran 2: Fungsi `GaussianQuadrature`

```
function G = GaussianQuadrature(f, a, b, n)
    beta = 0.5 ./ sqrt(1 - (2 * (1:n-1)).^(-2));
    [V, D] = eig(diag(beta, 1) + diag(beta, -1));
    nodes = diag(D);
    weights = 2 * V(1, :).^2;

    nodes = nodes';
    weights = weights';

    G = 0;
    for i = 1:n
        x = (b - a) * nodes(i) / 2 + (b + a) / 2;
        G = G + weights(i) * f(x);
    end

    G = G * (b - a) / 2;
end
```

5.3 Lampiran 3: Fungsi `RepeatedGaussianQuadrature`

```
function G = RepeatedGaussianQuadrature(f, a, b, m, n)
    h = (b - a) / m;
    G = 0;

    for i = 0:(m-1)
        a_i = a + i * h;
        b_i = a_i + h;
        G = G + GaussianQuadrature(f, a_i, b_i, n);
    end
end
```