Komputasi Fisika

Pekik Nurwantoro

January 15, 2026

Contents

0.1 Tinjauan Ringkas Bahasa Julia

0.1.1 Dasar-dasar Julia

Scientific Computing memerlukan bahasa pemrograman yang memiliki unjuk kerja tinggi, dan sekaligus sederhana, untuk menyelesaikan suatu permasalahan secara komputasi. Dalam praktek, kedua fitur tersebut sayangnya tidak dapat terpenuhi secara bersamaan. Salah satu komponen penunjang unjuk kerja tinggi adalah berupa sistem *compiler*, bukan sistem *interpreter* seperti bahasa Python atau R. Adapun salah satu ciri kemudahan dalam bahasa pemrograman adalah berupa *dynamically-typed language*, bukan *statically-typed language* seperti bahasa C.

Bahasa pemrograman Julia dirancang sejak awal untuk memiliki 2 fitur tersebut yaitu memiliki unjuk kerja tinggi dan sederhana dalam penggunaan. Fitur lain dari Julia antara lain:

- Dikembangkan oleh MIT dengan lisensi yang bersifat *free* dan *open source*.
- Dirancang untuk mendukung komputasi paralel
- Didukung dalam penggunaan *Unicode*

Uraian berikut akan memaparkan secara ringkas tentang sebagian fasilitas yang dimiliki oleh Julia dalam melakukan berbagai perintah (*tasks*) yang terkait dengan komputasi. Informasi secara lebih lengkap terkait beberapa uraian yang disajikan di bawah maka dapat merujuk pada Dokumentasi Julia.

0.1.2 Sintak untuk Komentar

Dalam penulisan suatu bahasa pemrograman, kadang diperlukan mekanisme untuk memberikan komentar agar beberapa bagian dalam *source code* menjadi lebih jelas. Untuk itu diperlukan suatu penanda untuk membedakan antara suatu perintah dan suatu komentar. Dalam Julia, sintak yang menjadi penanda bagi suatu komentar adalah `#`.

0.1.3 *Variable*

Variable merupakan suatu penamaan yang terkait dengan suatu nilai. Penulisan *variable* dalam Julia amat fleksibel dan kaya pilihan, yang mampu membedakan antara huruf besar dan kecil serta dapat menggunakan *unicode*, bahkan mampu menerima kode dalam *LaTeX*. Berikut adalah contoh penggunaan *variable*:

```
# seluruh kata setelah tanda pagar dianggap sebagai komentar, bukan sebagai perintah
x = 10

10

\delta = 13

13

ő = 5

5
```

Dalam ungkapan di atas, simbol *LaTeX* δ dimasukkan dengan mengetik \delta dan diikuti tekan kunci TAB, sedang simbol *Unicode* ®dimasukkan dengan mengetik \circledR dan diikuti tekan kunci TAB.

0.1.4 Bilangan Bulat dan Pecahan (*Floating Point*)

Bilangan Bulat

Secara otomatis (*default*), jenis bilangan bulat yang digunakan dalam Julia akan tergantung pada tipe bit komputer yang digunakan, yaitu jenis bilangan bulat Int8, Int16, Int32, Int64 atau Int128 untuk komputer bertipe 8, 16, 32, 64 atau 128 bit. Semakin besar tipe bit maka semakin besar pula kemampuan komputer untuk mengakses bilangan bulat maksimum (bilangan bulat positip yang terbesar) dan minimum (bilangan bulat negatip yang terbesar).

```
x = 2894
```

```
2894
```

```
typeof(x)
```

```
Int64
```

```
(typemax(Int64), typemin(Int64))
```

```
(9223372036854775807, -9223372036854775808)
```

```
(typemax(Int32), typemin(Int32))
```

```
(2147483647, -2147483648)
```

Pecahan (*Floating Point*)

Seperti halnya bilangan bulat, jenis *floating point* yang digunakan secara *default* saat menjalankan komputasi juga ditentukan oleh tipe bit komputer tersebut antara lain Float16, Float32 atau Float64. Jenis *floating point* juga dapat dipilih dengan menggunakan penulisan angka dengan notasi-e (sebagai contoh 2.3e-3 sebagai wakilan angka 0.0023) untuk Float64 dan notasi-f (sebagai contoh 2.3f-3 sebagai wakilan angka 0.0023) untuk Float32.

Beberapa nilai khusus dalam *floating point*, dan tidak ada dalam bilangan bulat, yang memiliki peran penting dalam komputasi antara lain:

- Nilai `eps` yang mewakili ketelitian bagi mesin (komputer), yaitu nilai yang menjadi pembeda antara satu *floating point* dengan *floating point* terdekatnya.
- Nilai `-Inf` dan `Inf` yang mewakili nilai tak hingga bagi mesin (komputer), yaitu nilai yang terkait dengan nilai maksimum atau yang lebih besar serta nilai minimum yang lebih kecil bagi *floating point*.
- Nilai `NaN` yang mewakili keadaan suatu *error* dalam proses komputasi, yaitu suatu nilai yang bukan nilai-nilai dalam *floating point*.

```
xf = 2.3e -34
```

```
2.3e -34
```

```
typeof(xf)
```

```
Float64
```

```
yf = 23f -34
```

```
2.3f -33
```

```
typeof(yf)
```

```
Float32
```

```
eps(1.0)
```

```
2.220446049250313e -16
```

```
(typemax(Float64), typemin(Float64))
```

```
(Inf, -Inf)
```

```
3.0 / Inf
```

```
0.0
```

```
3.0 / 0
```

```
Inf
```

```
sin(0) / 0
```

```
NaN
```

Bilangan Kompleks dan Bilangan Rasional

Dalam berbagai permasalahan, proses komputasi sering melibatkan bilangan kompleks dan bilangan rasional. Dalam Julia, tetapan `im` digunakan untuk mewakili bilangan imajiner $i = \sqrt{-1}$. Berbeda dengan kebanyakan bahasa pemrograman yang menggunakan tetapan `i` atau `j` untuk mewakili bilangan imajiner, dalam Julia digunakan tetapan `im` karena notasi `i` atau `j` sering dimanfaatkan sebagai indeks dalam proses perhitungan perulangan (*looping*). Penulisan bilangan kompleks dalam Julia mengikuti penulisan matematika baku sebagai contoh `z = 3 + 2im`, yang lebih ringkas dibanding penulisan `z = 3 + 2*im`.

Julia memiliki cara alamiah untuk mewakili bilangan rasional, yaitu bilangan yang dapat dinyatakan sebagai pembagian dari 2 bilangan bulat, dalam bentuk `//`. Penggunaan bilangan rasional dalam proses komputasi sangat berguna dalam berbagai bidang permasalahan.

0.1.5 Operasi Matematika dan Fungsi Dasar

Beberapa perintah untuk melakukan operasi matematika beserta beberapa fungsi dasar dalam Julia memiliki ungkapan yang secara alamiah sama dengan penulisan baku dalam matematika. Tidak seperti perintah dalam bahasa lain, sebagai contoh dalam Julia dibolehkan untuk menulis perintah dalam bentuk `2x`, alih-alih dalam bentuk `2 * x`. Beberapa ungkapan operasi matematika beserta fungsi dasar yang lebih lengkap dapat merujuk pada Dokumen Julia.

```
x = 2.6 + 2.1im # sama dengan penulisan 2.6 + 2.1*im
```

```
2.6 + 2.1im
```

```
y = 3
```

```
3
```

```
x * y
```

```
7.800000000000001 + 6.300000000000001im
```

```
x / y
```

```
0.8666666666666667 + 0.7000000000000001im
```

```
x + y
```

```
5.6 + 2.1im

x - y

-0.3999999999999999 + 2.1im

y ^ 3

27

y % 2 # sisa dari pembagian x terhadap y

1

2x # sama seperti menulis y = 2 * x

5.2 + 4.2im

y + 1//2

7//2

nilai = cos(y)

-0.9899924966004454
```

Operasi *Boolean* dan Perbandingan Numerik

Dalam beberapa proses komputasi kadang diperlukan untuk memilih berdasar beberapa kondisi atau persyaratan melalui operasi *Boolean*, yaitu operasi untuk melihat suatu keadaan adalah benar (*true*) atau salah (*false*) serta perbandingan 2 nilai secara numerik. Beberapa operasi *Boolean* serta perbandingan numerik yang difasilitasi dalam Julia diberikan dalam contoh berikut.

```
kondisi1 = 2.5 == 3.7 # operasi kesamaan

false

kondisi2 = 2.5 != 3.7 # operasi ketidaksamaan

true

2.5 < 3.7 # operasi kurang dari

true

2.5 <= 3.7 # operasi kuranf dari atau sama dengan

true

2.5 >= 3.7 # operasi lebih dari atau sama dengan

false

!kondisi1 # operasi boolean NEGATION yaitu kebalikan dari keadaan sebelumnya

true

kondisi1 && kondisi2 # operasi boolean AND yaitu false jika salah satu false

false

kondisi1 || kondisi2 # operasi boolean OR yaitu true jika salah satu true

true
```

Operasi Pembaruan (*updating*)

Dalam beberapa proses komputasi, kadang diperlukan pembaruan suatu nilai oleh operasi matematika tertentu. Sebagai gambaran, untuk memperbarui nilai suatu *variable* agar nilai tersebut bertambah 3 maka Julia memberikan operasi ringkas dalam bentuk `x += 3`, yang sama dengan operasi `x = x + 3`. Operasi tersebut juga berlaku untuk bentuk lain yaitu `-=`, `*=`, `/=`, `^=`, `%=` dan lainnya.

```
x = 5.6

5.6

x += 3

8.6

y = 2

2

y ^= 3 # sama dengan operasi y = y^3

8
```

0.1.6 *Array*

Array merupakan jenis *variable* berupa jajaran dari nilai-nilai yang disajikan dalam bentuk 1, 2 hingga multi-dimensi. *Array* dalam 1 dimensi digunakan untuk melambangkan suatu vektor yang memiliki bentuk berupa jajaran nilai pada beberapa baris dalam satu kolom (biasa disebut sebagai matrik kolom). *Array* dalam 2 dimensi digunakan untuk melambangkan suatu matrik yang memiliki bentuk berupa jajaran nilai pada beberapa baris dan kolom. Secara umum *array* dalam 3 dimensi atau multi-dimensi biasa digunakan untuk melambangkan suatu tensor.

Array memegang peran penting dalam komputasi karena operasi yang melibatkan banyak nilai, yang berarti banyak *variable*, akan dapat diwakili oleh satu *variable* sehingga menyederhanakan prosedur penyelesaian. Implementasi *array* dalam Julia cukup lengkap yang dapat dirujuk pada Dokumentasi *Array*. Berikut disajikan beberapa hal penting dari implementasi *array* tersebut.

Beberapa Fungsi Dasar

Array diwakili oleh lambang kurung persegi `[ ]`, ketika berupa vektor maka dapat dibangkitkan dengan sintak `[x1, x2, ...]`, dengan `x1, x2, ...` merupakan unsur-unsur vektor.

Array berupa matrik dapat dibangkitkan dengan sintak `[x1 x2 ...; y1 y2 ...; ...]`, yaitu memberikan jarak spasi antar unsur matrix pada suatu baris dan pemberian *semicolon* untuk memisahkan kolom.

Dalam beberapa permasalahan komputasi, kadang diperlukan *array* dengan bentuk khusus sebagai contoh matrik yang memiliki semua unsur bernilai 1. Beberapa fungsi dasar untuk membangkitkan *array* dalam bentuk khusus tersebut diberikan pada contoh berikut.

```
A = [1,2,3]

3 -element Vector{Int64}:
 1
 2
 3

B = [1 2 3;3 4 5; 6 7 8]
```

```
3 \times3 Matrix{Int64}:
 1  2  3
 3  4  5
 6  7  8

C = zeros(2,3) # matrik orde 2x3 (2 baris dan 3 kolom) dengan semua unsur bernilai 0

2 \times3 Matrix{Float64}:
 0.0  0.0  0.0
 0.0  0.0  0.0

D = ones(3,4) # matrik orde 3x4 dengan semua unsur bernilai 1

3 \times4 Matrix{Float64}:
 1.0  1.0  1.0  1.0
 1.0  1.0  1.0  1.0
 1.0  1.0  1.0  1.0

E = rand(3,2) # matrik orde 3x2 dengan semua unsur bernilai bilangan random dalam interval 0 d

3 \times2 Matrix{Float64}:
 0.543852  0.713582
 0.656821  0.47524
 0.996348  0.844814

length(E) # operasi untuk menunjukkan cacah unsur pada matrik E

6

size(E) # operasi untuk menunjukkan daftar urutan (*tuple*) dimensi atau orde bagi matrik E

(3, 2)

ndims(E) # operasi untuk menunjukkan dimensi bagi matrik E

2
```

Indeks dalam *Array*

Berbeda dengan bahasa pemrogram lain, sebagai contoh Python, aturan penomoran atau indeks dalam suatu *array* dimulai dengan indeks 1 (bukan 0). Kemudian urutan indeks dalam seluruh *array* didasarkan pada urutan kolom (bukan baris). Contoh berikut adalah perintah untuk menampilkan unsur dari suatu *array* (vektor atau matrik) berdasar aturan indeks tersebut.

Penomoran atau pengaturan indeks dari suatu unsur tertentu dalam suatu *array* dalam dilakukan dengan 2 cara:

1. *Cartesian Index* yaitu penomoran dengan mengikuti indeks bagi *array* tersebut, sebagai contoh unsur `A[2.3]` adalah unsur beris ke 2 dan kolom ke 3 dari matrik `A`.
2. *Linear Index* yaitu penomoran dengan mengikuti urutan lokasi suatu unsur tertentu bagi *arrtay* tersebut, sesuai urutan kolom.

Perintah `CartesianIndices` dan `LinearIndices` digunakan untuk melakukan konversi dari masing-masing cara penomoran tersebut.

```
A = [1,2,3,4,5]
```

```
5 -element Vector{Int64}:
 1
 2
 3
 4
 5

A[5] # mencuplik nilai dari unsur ke 5 bagi vektor A

5

B = [1 2 3;3 4 5; 6 7 8]

3 \times3 Matrix{Int64}:
 1  2  3
 3  4  5
 6  7  8

B[5] # mencuplik nilai dari unsur ke 5 bagi matrik B, yaitu dimulai dari kolom ke 1 (terdiri a

4

B[2,2] # mencuplik nilai dari unsur baris ke 2 dan kolom ke 2, yang sama dengan perintah B[5]

4

CartesianIndices(B)[5] # mengubah dari unsur urutan ke 5 menjadi unsur baris ke 2 dan kolom ke

CartesianIndex(2, 2)

LinearIndices(B)[2,2] # mengubah dari unsur baris ke 2 dan kolom ke 2 menjadi unsur urutan ke

5
```

Operasi Vektorisasi pada *Array*

Operasi vektorisasi merupakan operasi matematika yang bekerja pada seluruh unsur pada *array* secara bersamaan, bukan satu persatu pada tiap unsur *array*. Mekanisme operasi matematika pada seluruh unsur yang semacam ini sehingga operasi vektorisasi kadang disebut pemrosesn berbasis unsur (*element-wise processing*). Fitur operasi vektorisasi dalam Julia bukan hanya menyederhanakan proses komputasi namun secara umum dapat mempercepat proses komputasi, khususnya pada komputer yang memfasilitasi pemrosesan paralel.

Julia mengimplementasikan operasi vektorisasi berupa penambahan perintah *dot* (yaitu `.`) di depan operasi matematika tertentu. Sebagai contoh, untuk mengalikan suatu nilai yang diwakili oleh *variable* `x1` dengan nilai `x2` maka digunakan sintak `x1 * x2`. Adapun untuk mengalikan suatu nilai yang diwakili oleh *variable* `x1` dengan seluruh nilai pada suatu *array* yang diwakili oleh *variable* `X2` maka diganakan sintak `x1 .* X2`.

```
x1 = 2.0

2.0

x2 = 0.4

0.4

X2 = rand(2,2)
```

```
2 \times2 Matrix{Float64}:
 0.835739  0.31754
 0.988938  0.701721

x1 * x2

0.8

x1 .* X2

2 \times2 Matrix{Float64}:
 1.67148  0.63508
 1.97788  1.40344
```

Implementasi *element-wise* memerlukan kehati-hatian ketika melibatkan matrik agar tidak rancu dengan operasi matrik sejenis yang secara definisi boleh jadi berbeda. Sebagai contoh, operasi perkalian matrik `A` dengan matrik `B`, yaitu `A * B`, secara definisi dilakukan dengan mengalikan baris matrik `A` dengan kolom matrik `B`. Oleh karena itu persyaratan agar `A * B` berlaku adalah jumlah kolom matrik `A` perlu sama dengan jumlah baris matrik `B`. Sedangkan operasi perkalian *element-wise* matrik `A` dengan matrik `B`, yaitu `A .* B`, secara definisi dilakukan dengan mengalikan setiap unsur matrik `A` dengan setiap unsur matrik `B`pada indeks matrik yang sama. Dengan demikian tidak ada persyaratan bahwa jumlah kolom matrik `A` perlu sama dengan jumlah baris matrik `B`agar `A .* B` berlaku. Contoh berikut memberikan gambaran jelas terkait uraian tersebut.

```
A = rand(2,2)

2 \times2 Matrix{Float64}:
 0.855451  0.553349
 0.978379  0.25829

B = rand(2,2)

2 \times2 Matrix{Float64}:
 0.357629  0.573794
 0.414223  0.347491

C = A * B

2 \times2 Matrix{Float64}:
 0.535145  0.683137
 0.456887  0.651142

D = A .* B

2 \times2 Matrix{Float64}:
 0.305935  0.317509
 0.405267  0.0897537

C == D

false
```

Operasi *Broadcasting*

Persyaratan agar operasi vektorisasi berbasis *element-wise* berlaku adalah adanya kesamaan orde bagi *array* yang terlibat dalam operasi. Apabila *array* yang terlibat ternyata tidak memiliki orde yang sama maka proses *element-wise* diimplemnetasikan melalui operasi *broadcasting* dengan sintak `braodcast(+,A,B)` untuk operasi *broadcasting* penjumlahan. Operasi *broadcasting* yang lain maka dilakukan dengan mengganti opearasi `+` dengan operasi matematika yang berpadanan.

```
A = rand(2,2)

2 \times2 Matrix{Float64}:
 0.996268  0.377983
 0.168009  0.813264

B = rand(1,2)

1 \times2 Matrix{Float64}:
 0.098382  0.498448

C = broadcast(+,A,B)

2 \times2 Matrix{Float64}:
 1.09465   0.876431
 0.266391  1.31171
```

0.1.7 Blok atau Gabungan Perintah

Suatu blok atau gabungan beberapa perintah kadang perlu diproses agar beberapa hasil yang diperoses tersebut akan dapat digunakan untuk menghasilkan satu hasil akhir. Blok atau gabungan beberapa perintah tersebut dalam Julia dapat dimplementasikan dalam berbagai bentuk, seperti bentuk fungsi, proses *conditional* atau proses *looping*, yang memiliki penanda pada akhir blok berupa perintah `end`. Gabungan perintah dalam Julia juga dapat diimplementasikan dalam bentuk umum berupa penanda perintah `begin` dan `end` seperti contoh berikut.

```
for i = 1:5
    i = i^3
    println(i)
end

1
8
27
64
125

nilai =
begin
    x = 3.2
    y = 5.6
    x/y
end

0.5714285714285715

nilai

0.5714285714285715
```

0.1.8 Fungsi dalam Julia

Kebanyakan fungsi dasar yang biasa digunakan di matematika, sebagai contoh fungsi trigonometri, telah terdefinisi dalam Julia sehingga dapat langsung diimplementasikan dalam proses komputasi. Apabila diperlukan bentuk fungsi yang belum termasuk dalam fungsi bawaan Julia, maka fungsi tersebut perlu didefinisikan terlebih dahulu sebelum dapat dimanfaatkan.

Secara umum bentuk fungsi dalam Julia akan berupa suatu obyek yang memetakan seperangkat argumen bagi fungsi, untuk menghasilkan satu atau beberapa nilai bagi fungsi tersebut. Beberapa contoh berikut merupakan metode yang dapat dipilih untuk mendefinisikan suatu fungsi. Uraian lebih lengkap terkait fungsi dapat merujuk pada Dokumen Fungsi.

```
function f(x,y)  # mendefinisikan fungsi untuk perkalian variable x dan y
    x * y
end
```

```
f (generic function with 1 method)
```

```
f(2,4)
```

```
8
```

```
f(x,y) = x * y # cara lain mendefinisikan fungsi untuk perkalian variable x dan y
```

```
f (generic function with 1 method)
```

```
f(3,4)
```

```
12
```

0.1.9 Proses Bersyarat (*Conditional*)

Dalam proses komputasi, suatu evaluasi tertentu kadang perlu dilakukan ketika satu persyaratan dipenuhi dan sebaliknya, suatu evaluasi tersebut tidak perlu dilakukan ketika persyaratan tersebut tidak terpenuhi. Proses bersyarat seperti hal tersebut dapat diimplementasikan melalui sintak `if-elseif-else` seperti contoh berikut.

```
function hasil(x,y)
    if (x < y)
        x + y
    elseif (x > y)
        x - y
    else
        x * y
    end
end
```

```
hasil (generic function with 1 method)
```

```
hasil(3,5)
```

```
8
```

```
hasil(5,2)
```

```
3
```

```
hasil(5,5)
```

```
25
```

0.1.10 Proses Berulang (*Looping*)

Ketika suatu proses perlu dilakukan secara berulang maka perintah untuk proses tersebut akan menjadi panjang dan oleh karena itu diperlukan mekanisme untuk meringkas perintah dalam bentuk proses berulang (*looping*). Proses *looping* dapat diwujudkan dalam dua sintak yaitu `while-end`dan `for-end` seperti contoh berikut.

```
function total(i)
    i = 1
    while (i < 10)
        i += 2
        println(i)
    end
end
```

```
total (generic function with 1 method)
```

```
total(9)
```

```
3
5
7
9
11
```

```
function total(n)
    for i = 1:n
        j = i + 2
        println(j)
    end
end
```

```
total (generic function with 1 method)
```

```
total(9)
```

```
3
4
5
6
7
8
9
10
11
```

Dalam ungkapan di atas, notasi *colon* yaitu `:` pada sintak `i = 1:n` mamiliki arti bahwa *variable* `i` akan bernilai dari `1` hingga nilai `n`.

0.1.11 Penggunaan Modul Tambahan

Dalam beberapa keperluan, kadang diperlukan suatu operasi tertentu yang tidak termasuk dalam paket (*library*) bawaan Julia. Sebagai contoh, keperluan penggunaan matrik Identitas `I` yang sering muncul dalam berbagai perhitungan aljabar linear ternyata tidak termasuk dalam *library* bawaan. Untuk keadaan seperti hal tersebut maka diperlukan pemanggilan *library* tambahan dengan menggunakan sintak `using`. Di dalam Panduan Julia akan dapat ditemukan bahwa matrik Identitas `I` termuat dalam *library* yang disebut `LinearAlgebra`. Contoh berikut menunjukkan cara untuk memuat *library* tambahan menggunakan perintak `using`.

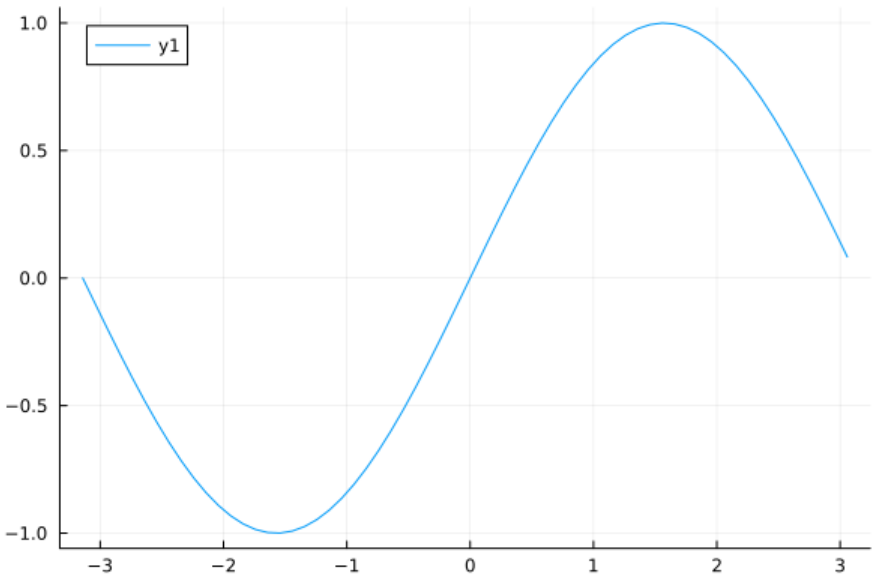
```
using LinearAlgebra

A = rand(2,2)

2 \times2 Matrix{Float64}:
 0.952013  0.531056
 0.165779  0.591548

B = I * A

2 \times2 Matrix{Float64}:
 0.952013  0.531056
 0.165779  0.591548
```

0.1.12 Plot: Visualisasi Data

Hasil dari proses komputasi pada umumnya melibatkan banyak data dan memerlukan penanganan lebih lanjut terhadap data tersebut. Salah satu cara untuk penanganan data adalah dengan melakukan visualisasi berupa plot terhadap data tersebut. Julia memeiliki fasilitas ynag cukup lengkap untuk membangkitkan berbagai jenis plot sesuai karakteristik data. Berbagai implementasi perintah plot di dalam Julia dapat merujuk pada Dokumen Plot.

Sintak untuk membangkitkan plot adalah dengan memuat *library* yaitu `using Plots` dan dikuti dengan beberapa perintah tertentu. Berikut merupakan beberapa contoh berdasar dokumen tersebut.

```
using Plots

x = -pi:0.1:pi

-3.141592653589793:0.1:3.058407346410207

y = sin.(x);

plot(x,y)
```



0.2 Kasus Fisika Non Linear: Osilasi Bandul

0.2.1 Implementasi Integrasi Numerik

Persamaan gerak benda bermassa m yang digantungkan pada seutas tali dengan panjang l dan massa diabaikan, di bawah pengaruh medan gravitasi bumi g, adalah

$$m\frac{d^2x(t)}{dt^2} = -mg\sin\theta(t)$$

Dalam radian, mengingat $x(t) = l\theta(t)$ dengan $\theta(t)$ adalah sudut simpangan benda terhadap titik setimbang, maka persamaan gerak tersebut dapat dinyatakan dalam bentuk persamaan diferensial orde dua berikut

$$\frac{d^2\theta(t)}{dt^2} = -\frac{g}{l}\sin\theta(t) \tag{1}$$

Penyelesaian persamaan diferensial tersebut secara analitik akan cukup sulit karena berbentuk nonlinear dalam $\theta(t)$. Untuk keadaan khusus, pada simpangan kecil sedemikian hingga $\sin\theta(t) \approx \theta(t)$ maka pers (1) di atas dapat didekati oleh bentuk

$$\frac{d^2\theta(t)}{dt^2} \approx -\frac{g}{l}\theta(t) = -\omega^2\theta(t) \tag{2}$$

dengan

$$\omega = \frac{2\pi}{T} = \sqrt{\frac{g}{l}} \tag{3}$$

Dengan demikian ungkapan periode bandul pada simpangan kecil, yang dinotasikan sebagai T_0 berbentuk

$$T_0 = 2\pi\sqrt{\frac{l}{g}} \tag{4}$$

Pada simpangan kecil ini, umumnya terpenuhi saat simpangan $\theta < 10^\circ$, maka penyelesaian pers (2) berbentuk

$$\theta(t) = \theta_0\cos(\omega t) \tag{5}$$

Ungkapan tersebut diperoleh dengan asumsi syarat awal bahwa pada saat awal $t = 0$ maka benda berada pada simpangan maksimum di $\theta = \theta_0$ dan kecepatan awal $\left[\frac{d\theta}{dt}\right]_{t=0} = 0$.

Untuk sebarang simpangan, yang tidak dibatasi pada simpangan kecil, penyelesaian pers (1) menjadi sulit untuk diperoleh secara analitik sehingga bentuk kompak penyelesaian simpangan seperti diberikan oleh pers (3) menjadi tidak berlaku. Oleh karena itu periode bandul pada sebarang simpangan (T) tidak dapat diperoleh berdasar pers (2c).

0.2.2 Periode Bandul pada Sebarang Simpangan (T)

Perhitungan periode bandul pada sebarang simpangan (T) dapat diperoleh dengan meninjau persamaan gerak bandul pada bentuk umum seperti disajikan oleh pers (1). Dengan mengalikan kedua ruas pada pers (1) di atas dengan $\frac{d\theta}{dt}$ dan kemudian melakukan intergrasi terhadap dt maka diperoleh ungkapan

$$\frac{d\theta}{dt} = \sqrt{\frac{2g}{l}}\sqrt{\cos\theta - \cos\theta_0}$$

Berdasar syarat awal tersebut, persamaan diferensial di atas dapat dinyatakan dalam ungkapan integral dalam bentuk

$$t = \sqrt{\frac{l}{2g}} \int_{\theta_0}^{\theta} \frac{d\theta}{\sqrt{\cos\theta - \cos\theta_0}} \tag{6}$$

Periode T dapat dipahami sebagai waktu yang ditempuh bagi benda untuk bergerak dari posisi θ_0 dan berayun kembali ke posisi semula θ_0. Dengan pengertian lain, periode T adalah empat kali waktu yang diperlukan untuk bergerak dari $\theta = \theta_0$ menuju $\theta = 0$. Berdasar pers(4) maka ungkapan bagi perhitungan periode T dapat dinyatakan dalam bentuk integral dalam bentuk

$$T = 4\sqrt{\frac{l}{2g}} \int_{0}^{\theta_0} \frac{d\theta}{\sqrt{\cos\theta - \cos\theta_0}} \tag{7}$$

Ungkapan integral tersebut berbentuk integral tak layak (*improper integral*) akibat terjadinya singularitas (bernilai tak hingga) pada nilai integral di bagian batas atas integral, yaitu saat $\theta = \theta_0$. Perhitungan bagi nilai periode T akibatnya menjadi sulit.

Untuk mengatasi kesulitan ini maka bentuk integral tak layak (*improper integral*) di atas perlu diubah menjadi integral layak (*proper integral*). Didefinisikan peubah baru $\xi(t)$, yang dikaitkan oleh $\theta(t)$ melalui kaitan

$$\sin\xi(t) = \frac{\sin\frac{\theta(t)}{2}}{\sin\frac{\theta_0}{2}} \tag{8}$$

Dalam peubah $\xi(t)$ maka ungkapan integral tak layak bagi periode T akan berubah menjadi integral layak dalam bentuk

$$T = 4\sqrt{\frac{l}{g}} \int_{0}^{\pi/2} \frac{d\xi}{\sqrt{1 - k^2\sin^2\xi}} \tag{9}$$

dengan

$$k = \sin\frac{\theta_0}{2} \tag{10}$$

Dengan demikian, ungkapan bentuk integral layak di atas dapat digunakan sebagai salah satu metode untuk memperoleh nilai periode bandul pada sebarang simpangan T.

0.2.3 Pemanfaatan Besaran Fisis Tak Berdimensi

Meskipun ungkapan dalam pers (7) dapat digunakan untuk pencarian periode T berdasarkan perhitungan nilai integral, namun dari pertimbangan komputasi akan lebih menguntungkan apabila dapat diubah ke bentuk yang tidak melibatkan satuan dari besaran fisis yang terlibat. Dengan kata lain, bentuk tersebut perlu diubah ke dalam bentuk yang melibatkan besaran fisis yang tak berdimensi. Ada berbagai cara untuk mendapatkan bentuk tak berdimensi bagi ungkapan pers (7).

Salah satu cara untuk mendapatkan ungkapan periode bandul yang tidak melibatkan besaran fisis adalah dengan membagi ungkapan periode bandul T dalam pers (7) dengan ungkapan periode bandul pada simpangan kecil T_0 dalam pers (2c) dan menyebut sebagai periode bandul ternormalisir pada sebarang simpangan (τ) dalam bentuk

$$\tau = \frac{T}{T_0} = \frac{4\sqrt{\frac{l}{g}}\int_0^{\pi/2}\frac{d\xi}{\sqrt{1-k^2\sin^2\xi}}}{2\pi\sqrt{\frac{l}{g}}} = \frac{2}{\pi}\int_0^{\pi/2}\frac{d\xi}{\sqrt{1-k^2\sin^2\xi}} \tag{11}$$

Melakukan komputasi yang melibatkan besaran fisis tak berdimensi pada umumnya lebih menguntungkan karena nilai dari besaran yang terlibat berada pada rentang nilai yang lebih terukur, umumnya antara nilai 0 dan 1, sehingga kadang disebut sebagai satuan yang ternormalisir.

0.2.4 Integrasi Numerik: Metode Simpson

Berdasarkan pada prosedur yang telah diuraikan di atas, perhitungan nilai integral dapat dilakukan secara numerik dengan beberapa metode pendekatan perhitungan nilai integral, seperti metode Trapesium atau metode Simpson. Untuk pilihan metode Simpson, maka proses integrasi numerik bagi sebarang fungsi $f(x)$ akan memanfaatkan pendekatan deret fungsi di sekitar titik $x = x_1$ hingga orde 2 (kuadratis) dalam bentuk

$$f(x) \approx f(x_1) + \frac{(x-x_1)}{1!}\left.\frac{df(x)}{dx}\right]_{x=x_1} + \frac{(x-x_1)^2}{2!}\left.\frac{d^2f(x)}{dx^2}\right]_{x=x_1} \tag{12}$$

Berdasar ungkapan pendekatan deret tersebut maka diperoleh

$$f(x_1+h) \approx f(x_1) + \frac{h}{1!}\left.\frac{df(x)}{dx}\right]_{x=x_1} + \frac{h^2}{2!}\left.\frac{d^2f(x)}{dx^2}\right]_{x=x_1} \tag{13}$$

$$f(x_1-h) \approx f(x_1) - \frac{h}{1!}\left.\frac{df(x)}{dx}\right]_{x=x_1} + \frac{h^2}{2!}\left.\frac{d^2f(x)}{dx^2}\right]_{x=x_1} \tag{14}$$

Apabila pers (11b) dikurangkan atau ditambahkan terhadap pers (11a) maka mudah ditunjukkan bahwa ungkapan berikut akan berlaku, yaitu

$$\left.\frac{df(x)}{dx}\right]_{x=x_1} \approx \frac{f(x_1+h)-f(x_1-h)}{2h} \tag{15}$$

$$\left.\frac{d^2f(x)}{dx^2}\right]_{x=x_1} \approx \frac{f(x_1+h)-2f(x_1)+f(x_1-h)}{h^2} \tag{16}$$

Berdasar pers (12a) dan (12b) di atas maka pers (10) menjadi

$$\begin{aligned} f(x) \approx & f(x_1) + \frac{(x-x_1)}{1!}\left[\frac{f(x_1+h)-f(x_1-h)}{2h}\right] + \\ & \frac{(x-x_1)^2}{2!}\left[\frac{f(x_1+h)-2f(x_1)+f(x_1-h)}{h^2}\right] \end{aligned} \tag{17}$$

Memanfaatkan pers (13) tersebut, tinjau masalah untuk menghitung pendekatan bagi nilai integral I dari sebarang fungsi $f(x)$ dari batas integral $x = x_1 - h$ hingga $x = x_1 + h$ seperti berikut

$$\begin{aligned} I = \int_{x_1-h}^{x_1+h} f(x)dx \approx \int_{x_1-h}^{x_1+h} \Bigg\{ & f(x_1) + \frac{(x-x_1)}{1!}\left[\frac{f(x_1+h)-f(x_1-h)}{2h}\right] + \\ & \frac{(x-x_1)^2}{2!}\left[\frac{f(x_1+h)-2f(x_1)+f(x_1-h)}{h^2}\right]\Bigg\} dx \end{aligned} \tag{18}$$

Untuk penyederhaan bentuk ungkapan dan nantinya berguna pada langkah komputasi maka dapat diperkenalkan beberapa notasi yaitu $x_0 = x_1 - h$ atau $x_1 = x_0 + h$ serta $x_2 = x_1 + h$. Dengan melakukan proses integrasi pada pers (14) maka diperoleh bentuk

$$I = \int_{x_0}^{x_2} f(x)dx \approx \frac{h}{3}\left[f(x_0) + 4f(x_1) + f(x_2)\right] \tag{19}$$

Secara umum, untuk batas integral dari $x = a$ hingga $x = b$ maka pendekatan bagi nilai integral dapat dilakukan dengan melakukan proses diskretisasi bagi peubah x yaitu membagi rentang $x = a$ hingga $x = b$ menjadi N bagian dengan lebar $h = \frac{b-a}{N}$ yang kecil dan N adalah bilangan genap. Oleh karena itu dapat digunakan titik-titik diskret bagi x dalam ungkapan

$$x_0 = a; \quad x_N = b; \quad x_i = x_0 + ih; \quad i = 1, 2, 3, \cdots, N-1;$$

Dengan titit-titik diskret tersebut maka integrasi numerik I bagi sebarang fungsi $f(x)$, disebut sebagai metode Simpson, dapat ditulis dalam bentuk

$$\begin{aligned} I &= \int_a^b f(x)dx = \int_{x_0}^{x_2} f(x)dx + \int_{x_2}^{x_4} f(x)dx + \cdots + \int_{x_{N-2}}^{x_N} f(x)dx \\ &\approx \frac{h}{3}\left[f(x_0) + 4f(x_1) + f(x_2)\right] + \cdots + \frac{h}{3}\left[f(x_{N-2}) + 4f(x_{N-1}) + f(x_N)\right] \\ &\approx \frac{h}{3}\left[f(x_0) + 4f(x_1) + 2f(x_2) + \cdots + 4f(x_{N-1}) + f(x_N)\right] \\ &\approx \frac{h}{3}\left[f(x_0) + f(x_N) + 4\sum_{i=1,3,\cdots}^{N-1} f(x_i) + 2\sum_{i=2,4,\cdots}^{N-2} f(x_i)\right] \end{aligned} \tag{20}$$

0.2.5 Algoritma Perhitungan Periode Bandul

Berawal dari keadaan diam di posisi simpangan maksimum tertentu θ_0, dan pada nilai l serta g tertentu, maka prosedur untuk memperoleh periode dalam satuan ternormalisir τ adalah seperti berikut

1. Berikan nilai masukan θ_0, dan oleh karenanya hitung nilai k mellaui ungkapan pers (8)
2. Hitung nilai integral $\int_0^{\pi/2} \frac{d\xi}{\sqrt{1-k^2\sin^2\xi}}$ berdasar ungkapan pers (10)
3. Nilai periode ternormalisir τ diperoleh melalui ungkapan pers (9)

```
using Plots
using SpecialFunctions

function fung(x)
    f = sin(x)
    return f
end

fung (generic function with 1 method)

function integsimpson(a, b, n)
    h = (b - a) / n
    sumodd = 0.0
    nhalf = floor(Int, n / 2)
    for i in 1:nhalf
        xodd = a + (2 * i - 1) * h
        sumodd += fung(xodd)
    end
    sumeven = 0.0
    for i in 1:(nhalf - 1)
        xeven = a + 2 * i * h
        sumeven += fung(xeven)
    end
    integsimp = h * (fung(a) + 4.0 * sumodd + 2.0 * sumeven + fung(b)) / 3.0
    return integsimp
end

integsimpson (generic function with 1 method)

integsimpson(0,pi/3,100)

0.5000000000334054
```

```
cos(0) -cos(pi/3)
```

```
0.4999999999999999
```

```
function functional_bandul(k, a, b, n)
    h = (b - a) / n
    integ = 0.0
    f = 1.0 / sqrt(1.0 - (k * sin(a))^2)
    integ += h * f / 3.0
    f = 1.0 / sqrt(1.0 - (k * sin(b))^2)
    integ += h * f / 3.0
    sumodd = 0.0
    nhalf = floor(Int, n / 2)
    for i in 1:nhalf
        xodd = a + (2 * i - 1) * h
        f = 1.0 / sqrt(1.0 - (k * sin(xodd))^2)
        sumodd += f
    end
    sumeven = 0.0
    for i in 1:(nhalf -1)
        xeven = a + 2 * i * h
        f = 1.0 / sqrt(1.0 - (k * sin(xeven))^2)
        sumeven += f
    end
    integ += h * (4.0 * sumodd + 2.0 * sumeven) / 3.0
    return integ
end
```

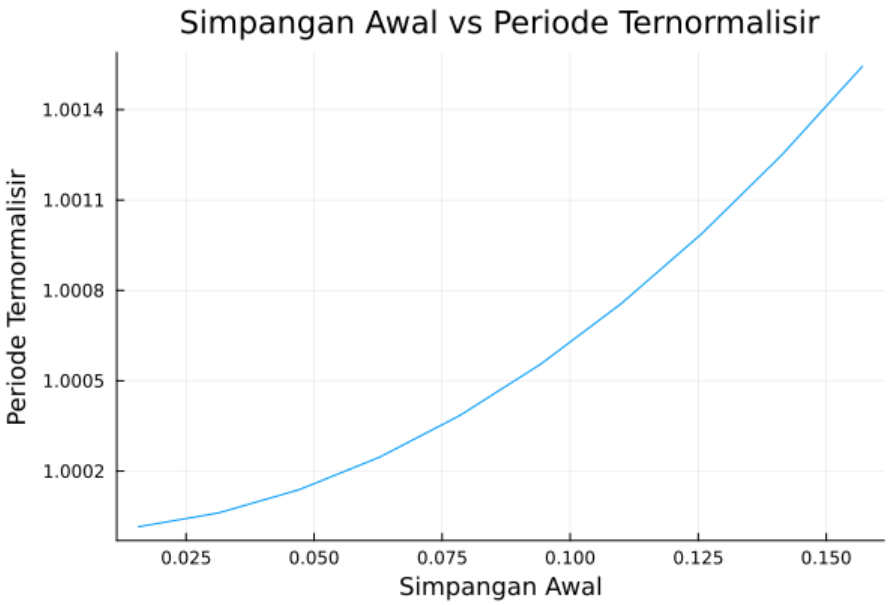
```
functional_bandul (generic function with 1 method)
```

```
a = 0.0
b = pi / 2.0
theta0_val = pi / 200.0
k_val = sin(theta0_val / 2.0)
n_val = 100
integ = functional_bandul(k_val, a, b, n_val)
periode = 2.0 * integ / pi
println("Periode: ", periode)
```

```
Periode: 1.0000154214748778
```

```
theta0 = range(pi / 200.0, stop=pi / 20.0, length=10)
tau = zeros(10)
n = 100
for i in 1:10
    k = sin(theta0[i] / 2.0)
    integ = functional_bandul(k, a, b, n)
    tau[i] = 2.0 * integ / pi
end
```

```
plot(theta0, tau, xlabel="Simpangan Awal", ylabel="Periode Ternormalisir", title="Simpangan Aw
```



0.2.6 Integral Elliptik (*Elliptic Integral*)

Ungkapan bentuk integral seperti diberikan oleh pers (7) atau pers (9) biasa disebut sebagai integral Elliptik lengkap jenis pertama (*The complete Elliptic Integral of the first kind*) yang didefinisikan sebagai:

$$K(k) = \int_0^{\frac{\pi}{2}} \frac{d\theta}{\sqrt{1 - k^2 \sin^2 \theta}}$$

Di dalam `Julia`, ungkapan tersebut difasilitasi `package SpecialFunctions`oleh fungsi khas dengan nama panggilan `ellipk`.

Untuk membandingkan hasil dari penggunaan integrasi numerik menggunakan metode Simpson yang diuraikan di atas dengan modul bawaan `Julia` yang difasilitasi oleh `ellipk` dapat ditunjukkan oleh *source-code* berikut:

```
a = 0.0
b = pi / 2.0
theta0 = pi / 200.0
k = sin(theta0 / 2.0)
n = 100

100

hasil1 = functional_bandul(k, a, b, n)
tau1 = 2.0 * hasil1 / pi
println("Tau from Simpson method: ", tau1)

Tau from Simpson method: 1.0000154214748778

m = k * k
hasil2 = ellipk(m)
tau2 = 2.0 * hasil2 / pi
println("Tau from SpecialFunctions.ellipk: ", tau2)

Tau from SpecialFunctions.ellipk: 1.0000154214748778
```

0.3 Kasus Fisika Non Linear: Osilasi Bandul

0.3.1 Implementasi Kuadrature Numerik

Persamaan gerak benda bermassa m yang digantungkan pada seutas tali dengan panjang l dan massa diabaikan, di bawah pengaruh medan gravitasi bumi g, adalah

$$m\frac{d^2x(t)}{dt^2} = -mg\sin\theta(t) \tag{21}$$

Dalam radian, mengingat $x(t) = l\theta(t)$ dengan $\theta(t)$ adalah sudut simpangan benda terhadap titik setimbang, maka persamaan gerak tersebut dapat dinyatakan dalam bentuk persamaan diferensial orde dua berikut

$$\frac{d^2\theta(t)}{dt^2} = -\frac{g}{l}\sin\theta(t) \tag{22}$$

Perhitungan periode bandul pada sebarang simpangan (T) dapat diperoleh dengan meninjau persamaan gerak bandul pada bentuk umum seperti disajikan oleh pers (1). Dengan mengalikan kedua ruas pada pers (1) di atas dengan $\frac{d\theta}{dt}$ dan kemudian melakukan intergrasi terhadap dt maka diperoleh ungkapan

$$\frac{d\theta}{dt} = \sqrt{\frac{2g}{l}}\sqrt{\cos\theta - \cos\theta_0} \tag{23}$$

Berdasar syarat awal tersebut, persamaan diferensial di atas dapat dinyatakan dalam ungkapan integral dalam bentuk

$$t = \sqrt{\frac{l}{2g}}\int_{\theta_0}^{\theta}\frac{d\theta}{\sqrt{\cos\theta - \cos\theta_0}} \tag{24}$$

Periode T dapat dipahami sebagai waktu yang ditempuh bagi benda bergerak dari posisi θ_0 dan berayun kembali ke posisi semula θ_0). Dengan pengertian lain, periode T adalah empat kali waktu yang diperlukan untuk bergerak dari $\theta = \theta_0$ menuju $\theta = 0$. Berdasar pers(4) maka ungkapan bagi perhitungan periode T dapat dinyatakan dalam bentuk integral dalam bentuk

$$T = 4\sqrt{\frac{l}{2g}}\int_{0}^{\theta_0}\frac{d\theta}{\sqrt{\cos\theta - \cos\theta_0}} \tag{25}$$

Dalam bentuk satuan universal, periode (τ) berbentuk

$$\tau = \frac{T}{T_0} = \frac{\sqrt{2}}{\pi}\int_{0}^{\theta_0}\frac{d\theta}{\sqrt{\cos\theta - \cos\theta_0}} \tag{26}$$

Ungkapan integral tersebut berbentuk integral tak layak (*improper integral*), yang salah satunya dicirikan oleh adanya singularitas (bernilai tak hingga) pada nilai integral di bagian batas atas integral, yaitu saat $\theta = \theta_0$. Perhitungan bagi nilai periode T akibatnya menjadi sulit apabila digunakan integrasi numerik dengan cara diskretisasi pada peubah bebas seperti yang dilakukan pada metode trapesium atau metode Simpson.

0.3.2 Metode Kuadratur Numerik: *Gauss-Legendre*

Perhitungan numerik bagi nilai integral yang selama ini telah dikaji, sebagai contoh metode Trapesium dan metode Simpson, adalah melalui proses diskretisasi peubah bebas pada titik-titik yang telah ditentukan $x_0, x_1, \cdots, x_N$. Secara umum nilai integral dari batas a hingga b dapat didekati oleh bentuk umum

$$\int_{x_0}^{x_N} f(x)dx \approx h\sum_{i=0}^{N} c_i f(x_i) \tag{27}$$

dengan $h = x_i - x_{i-1}$ adalah ukuran langkah (*step size*) atau tingkat kehalusan diskretisasi, c_i adalah nilai koefisien atau bobot fungsi yang seperangkat nilainya dapat ditentukan oleh metode diskretisasi yang dipilih dan $x_0 = a, x_N = b$.

Sebagai contoh untuk metode Trapesium maka $c_0 = \frac{1}{2}, c_N = \frac{1}{2}$, $c_i = 1$ untuk $i = 1, 2, \cdots, N-1$ sedangkan untuk metode Simpson maka $c_0 = \frac{1}{3}, c_N = \frac{1}{3}$, $c_{2i-1} = \frac{4}{3}$ untuk $i = 1, 2, \cdots, \frac{N}{2}$ dan $c_{2i} = \frac{2}{3}$ untuk $i = 1, 2, \cdots, \frac{N}{2} - 1$.

Nampak dari metode perhitungan numerik pers (6) bahwa pendekatan perhitungan nilai integral akan tidak berhasil apabila terjadi singularitas pada nilai fungsi di titik tertentu, sebagai contoh pada titik θ_0 di bentuk integral tak layak pers (5) di atas.

Dengan prosedur yang agak berbeda, metode kuadratur numerik adalah metode untuk mendekati nilai integral berdasarkan seperangkat nilai bobot c_i dan titik x_i yang ditentukan berdasarkan tingkat (*orde*) ketelitian yang akan dicapai, bukan berdasarkan proses diskretisasi. Dengan tambahan derajat kebebasan untuk memilih seperangkat titik x_i maka 2 fitur yang dimiliki metode kuadratur, dibanding metode integrasi berdasar diskretisasi, adalah

1. Singularitas dapat dihindari karena seperangkat titik x_i tidak berada pada suatu titik yang menyebabkan nilai fungsi bernilai tak hingga.

2. Orde ketelitian lebih tinggi karena jumlah peubah bebas menjadi lebih banyak.

Sebagai gambaran terkait 2 fitur tersebut, tinjau ungkapan kuadratur numerik untuk kasus sederhana, yaitu berdasar evaluasi pada 2 titik x_1 dan x_2 yang berada pada interval -1 dan 1, berikut

$$\int_{-1}^{1} f(x)dx \approx c_1 f(x_1) + c_2 f(x_2) \tag{28}$$

Dalam ungkapan tersebut, sejumlah 4 nilai c_1, c_2, x_1 dan x_2 menjadi bebas untuk ditentukan. Salah satu cara untuk menentukan 4 nilai tersebut adalah dengan menyusun 4 persamaan yang menjamin bahwa pers (7) akan memberikan nilai eksak apabila $f(x)$ berbentuk polinomial hingga orde 3.

- Untuk $f(x) = x^0 = 1$ maka pers (7) menjadi
$$\int_{-1}^{1} 1dx = 2 = c_1 x_1^0 + c_2 x_2^0 = w_1 + w_2 \tag{29}$$
- Untuk $f(x) = x^1$ maka pers (7) menjadi
$$\int_{-1}^{1} xdx = 0 = c_1 x_1 + c_2 x_2 \tag{30}$$
- Untuk $f(x) = x^2$ maka pers (7) menjadi
$$\int_{-1}^{1} x^2 dx = \frac{2}{3} = c_1 x_1^2 + c_2 x_2^2 \tag{31}$$
- Untuk $f(x) = x^3$ maka pers (7) menjadi
$$\int_{-1}^{1} x^3 dx = 0 = c_1 x_1^3 + c_2 x_2^3 \tag{32}$$

Mudah ditunjukkan bahwa penyelsaian dari 4 persamaan serentak tersebut adalah
$c_1 = 1, c_2 = 1, x_1 = -\frac{1}{\sqrt{3}} \approx= -0.577350269$ dan $x_2 = \frac{1}{\sqrt{3}} \approx 0.577350269$.

Dengan pers (7) dan 4 nilai yang diperoleh tersebut maka metode kuadratur numerik pada 2 titik berbentuk

$$\int_{-1}^{1} f(x)dx \approx 1f(-0.577350269) + 1f(0.577350269) \tag{33}$$

Berikut adalah contoh penggunaan kuadratur numerik 2 titik untuk menentukan nilai integral dari $f(x) = 2 + x^2$

```
c1 = 1.0
c2 = 1.0
x1 = -0.577350269
x2 = 0.577350269
f1 = 2.0 + x1^2
f2 = 2.0 + x2^2
nilai_kuad = c1*f1 + c2*f2
nilai_eksak = 2.0*(1.0 - ( -1.0)) + (1.0^3 - ( -1.0)^3)/3.0;

nilai_kuad

4.666666666228744

nilai_eksak

4.666666666666667
```

Bandingkan hasil kuadratur numerik 2 titik tersebut dengan intergrasi numerik 2 titik berdasar metode Trapesium untuk bentuk fungsi yang sama yaitu

```
h = 1.0 - ( -1.0)
c1 = 0.5
c2 = 0.5
x1 = -1.0
x2 = 1.0
f1 = 2.0 + x1^2
f2 = 2.0 + x2^2
nilai_trap = (c1*f1 + c2*f2)*h;

nilai_trap

6.0
```

Nampak bahwa perhitungan nilai integral berdasar metode kuadratur numerik nampak lebih teliti dibanding metode Trapesium untuk cacah titik yang sama. Ini merupakan salah satu fitur dari metode kuadratur numerik yang disinggung di atas.

Pengaturan Skala untuk Sebarang Batas Integral

Apabila batas integral adalah dari $y = a$ hingga $y = b$ maka bentuk kuadratur numerik dalam pers (7) dari $x = -1$ hingga $x = 1$ dapat dilakukan pengaturan skala secara linear dalam bentuk

$$y = \frac{b-a}{2}x + \frac{b+a}{2} \tag{34}$$

Dengan pengaturan skala tersebut maka bentuk kuadratur numerik dalam pers (7) dapat digunakan dalam bentuk menjadi

$$\int_a^b f(y)dy \approx \frac{b-a}{2}\left[c_1 f\left(\frac{b-a}{2}x_1 + \frac{b+a}{2}\right) + c_2 f\left(\frac{b-a}{2}x_2 + \frac{b+a}{2}\right)\right] \tag{35}$$

Dengan bentuk kuadratur numerik seperti disajikan oleh pers (13) maka ungkapan integral tak layak bagi periode bandul (τ) dalam pers (5a) akan dimungkinkan untuk dihitung karena nilai fungsi pada θ_0 tidak perlu untuk dihitung. Akibatnya proses perhitungan nilai integral pada titik singular dapat dihindari, yang merupakan salah satu fitur dari metode kuadratur numerik.

```
using LinearAlgebra
using SpecialFunctions

function fung(y, y0)
    return 1.0 / sqrt(cos(y) - cos(y0))
end

function kuad_GL(a, b)
    n = 2
    c = zeros(n)
    x = zeros(n)
    c[1] = 1.0
    c[2] = 1.0
    x[1] = -1.0/sqrt(3.0)
    x[2] = 1.0/sqrt(3.0)
    sum = 0.0
    for i in 1:n
        y = (b -a)*x[i]/2.0 + (b+a)/2.0
        sum += c[i] * fung(y, b)
    end
    kuad = (b -a) * sum/2.0
    return kuad
end

theta0 = pi/200.0
a = 0.0
b = theta0
kuad = kuad_GL(a, b)
tau = sqrt(2.0) * kuad / pi;

tau

0.843413737027873
```

Nampak bahwa dengan menggunakan metode kuadratur numerik maka nilai periode bandul yang ternormalisir (τ) dapat diperoleh berdasar bentuk intergral tak layak, meskipun dengan nilai yang masih belum teliti. Hasil yang lebih teliti pada metode kuadratur numerik dapat diperoleh dengan mengambil cacah titik yang lebih banyak.

Salah satu kendala bagi metode kuadratur numerik adalah perlunya pencarian seperangkat nilai c_i dan x_i, saat $i = 1, 2, \cdots, N$, untuk cacah titik N yang dipilih. Pada umumnya, pencarian nilai-nilai c_i dan x_i tersebut dapat dilakukan dengan melibatkan operasi pada fungsi khas (*special function*) berupa polinomial orthonormal tertentu.

Sebagai contoh, metode kuadratur numerik dengan ungkapan seperti diberikan oleh pers (7) merupakan bentuk khusus, yaitu saat cacah titik $N = 2$, dari apa yang disebut sebagai metode kuadratur *Gauss-Legendre* dengan bentuk umum

$$\int_{-1}^{1} f(x)dx \approx \sum_{i=1}^{n} c_i f(x_i) \tag{36}$$

Nilai-nilai x_i dapat diperoleh melalui akar-akar atau titik nol (*zeros*) dari polinomial *Legendre* orde ke n, dengan lambang $P_n(x)$, sedangkan nilai-nilai c_i diperoleh dengan melibatkan turunan ke 1 dari polinomial tersebut, yaitu melalui ungkapan

$$P_n(x_i) = 0; \qquad c_i = \frac{2}{(1 - x_i^2)\left[P_n'(x_i)\right]^2} \tag{37}$$

Untuk beberapa titik yang tidak terlalu banyak, nilai-nilai c_i dan x_i bagi kuadratur *Gauss-Legendre*, seperti diberikan oleh pers (16), diberikan dalam bentuk Tabel seperti berikut (Gaussian Kuadrature)

Cacah titik n	Bobot c_i	Titik x_i
1	2	0
2	1	-0.577350...
1	0.577350...	
3	0.555556...	-0.774597...
0.888889...	0	
0.555556...	0.774597...	
4	0.347855...	-0.861136...
0.652145...	-0.339981...	
0.652145...	0.339981...	
0.347855...	0.861136...	
5	0.236927...	-0.90618...
0.478629...	-0.53846...	
0.568889...	0	
0.478629...	0.53846...	
0.236927...	0.90618...	

Untuk memberikan gambaran bahwa pengambilan cacah titik yang semakin banyak maka akan dapat meningkatkan ketelitian metode kuadrature *Gauss-Legendre*, berikut akan diambil untuk $n = 5$.

```
function fung(y, y0)
    return 1.0 / sqrt(cos(y) - cos(y0))
end

function kuad_GL1(a, b)
  n = 5
  c = zeros(n)
  x = zeros(n)
  c[1] = 0.236927
  c[2] = 0.478629
  c[3] = 0.568889
  c[4] = 0.478629
  c[5] = 0.236927
  x[1] = -0.90618
  x[2] = -0.53846
  x[3] = 0.0
  x[4] = 0.53846
  x[5] = 0.90618
  sum = 0.0
  for i in 1:n
    y = (b -a)*x[i]/2.0 + (b+a)/2.0
    sum += c[i] * fung(y, b)
  end
  kuad = (b -a) * sum / 2.0
  return kuad
end

theta0 = pi/200.0
a = 0.0
b = theta0
```

```
kuad = kuad_GL1(a,b)
tau = sqrt(2.0) * kuad/pi;

tau

0.9287777927160661
```

Selain disajikan dalam bentuk ungkapan seperti pers (16) atau bentuk Tabel, beberapa bahasa pemrograman atau paket matematika biasanya juga menyediakan *Library*, *Toolbox* atau *Package* untuk membangkitkan nilai-nilai titik evaluasi fungsi (x_i) beserta nilai bobot terkait (c_i). Sebagai contoh, di dalam Julia hal tersebut difasilitasi oleh *Package* `FastGaussQuadrature` seperti berikut.

```
using FastGaussQuadrature

function fung(y, y0)
    return 1.0 / sqrt(cos(y) - cos(y0))
end

function kuad_GL(a, b, n)
    x, c = gausslegendre(n)
    sum = 0.0
    for i in 1:n
        y = (b -a)*x[i]/2.0 + (b+a)/2.0
        sum += c[i] * fung(y, b)
    end
    kuad = (b -a) * sum/2.0
    return kuad
end

theta0 = pi/200.0
a = 0.0
b = theta0
kuad = kuad_GL(a, b, 50)
tau = sqrt(2.0) * kuad / pi;

tau

0.9922539469065875
```

0.3.3 Bentuk Lain Metode Kuadratur Numerik

Metode kuadratur *Gauss-Legendre* merupakan salah satu bentuk dari bentuk umum metode kuadratur yaitu

$$\int_a^b w(x)f(x)dx \approx \sum_{i=1}^{n} c_i f(x_i) \tag{38}$$

Ungkapan tersebut berlaku pada interval $[a, b]$ dan bentuk bobot $w(x)$ tertentu. Keadaan khusus saat $a = -1, b = 1$ dan $w(x) = 1$ maka pers (17) di atas merupakan bentuk dari metode kuadratur *Gauss-Legnedre*.

Interval $[a, b]$ dan bentuk fungsi bobot $w(x)$ akan terkait dengan fungsi khas dari polinomial orthonormal tertentu sehingga digunakan sebagai penamaan bagi metode kuadratur tersebut. Pemahaman terkait polinomial orthonormal tertentu tersebut akan berguna untuk mendapatkan nilai-nilai x_i dan c_i pada orde pendekatan tertentu.

Tabel berikut memberikan daftar dari beberapa bentuk metode kuadratur numerik yang banyak digunakan pada beberapa permasalahan perhitungan yang melibatkan integral tak layak.

Interval pada $[a,b]$	Bentuk fungsi bobot $w(x)$	Polinomial Orthonormal	Nama Metode Kuadratur
$[-1,1]$	1	Polinomial *Legendre*	*Gauss-Legendre*
$[-1,1]$	$(1-x)^{\alpha}(1+x)^{\beta}$	Polinomial *Jacobi*	*Gauss-Jacobi*
$[-1,1]$	$\frac{1}{\sqrt{(1-x^2)}}$	Polinomial *Chebyshev* jenis 1	*Gauss-Chebyshev*
$[-1,1]$	$\sqrt{(1-x^2)}$	Polinomial *Chebyshev* jenis 2	*Gauss-Chebyshev*
$[0,\infty]$	e^{-x}	Polinomial *Laguerre* jenis 2	*Gauss-Laguerre*
$[-\infty,\infty]$	e^{-x^2}	Polinomial *Hermite* jenis 2	*Gauss-Hermite*

0.3.4 Implementasi Bentuk Kuadratur Numerik lainnya: *Gauss-Chebyshev*

Perhitungan periode bandul pada sebarang simpangan T atau τ dengan bentuk seperti pers (5) atau (5a) akan lebih sesuai diselesaikan berdasar kuadratur numerik *Gauss-Chebyshev*, dibanding dengan kuadratur numerik *Gauss-Legendre* seperti uraian berikut.

Tinjau substitusi variabel dalam bentuk ungkapan

$$k = \sin\frac{\theta_0}{2}; \quad \text{dan} \quad y = \frac{\sin\frac{\theta}{2}}{\sin\frac{\theta_0}{2}} = \frac{\sin\frac{\theta}{2}}{k} \tag{39}$$

Maka diperoleh bahwa:

$$y = 0 \quad \text{saat} \quad \theta = 0; \quad \text{dan} \quad y = 1 \quad \text{saat} \quad \theta = \theta_0 \tag{40}$$

Dengan ungkapan tersebut maka diperoleh

$$\sqrt{\cos\theta - \cos\theta_0} = \sqrt{2\left(\sin\frac{\theta_0}{2} - \sin\frac{\theta}{2}\right)} = \sqrt{2}\sin\frac{\theta_0}{2}\sqrt{1 - \frac{\sin\frac{\theta}{2}}{\sin\frac{\theta_0}{2}}} = \sqrt{2}k\sqrt{1-y^2} \tag{41}$$

$$dy = \frac{1}{2k}\sqrt{1-\sin^2\frac{\theta}{2}}d\theta = \frac{1}{2k}\sqrt{1-k^2y^2}d\theta; \quad \Longrightarrow \quad d\theta = \frac{2k}{\sqrt{1-k^2y^2}}dy \tag{42}$$

Ungkapan periode bandul pada sebarang simpangan T atau τ dengan bentuk seperti pers (5) atau (5a) maka dapat dinyatakan dalam bentuk lain yaitu

$$T = 4\sqrt{\frac{l}{2g}}\int_0^{\theta_0}\frac{d\theta}{\sqrt{\cos\theta - \cos\theta_0}} = 4\sqrt{\frac{l}{g}}\int_0^1\frac{dy}{\sqrt{1-y^2}\sqrt{1-k^2y^2}} \tag{43}$$

Dalam bentuk satuan universal, periode (τ) berbentuk

$$\begin{aligned}\tau = \frac{T}{T_0} &= \frac{\sqrt{2}}{\pi}\int_0^{\theta_0}\frac{d\theta}{\sqrt{\cos\theta - \cos\theta_0}} = \frac{2}{\pi}\int_0^1\frac{dy}{\sqrt{1-y^2}\sqrt{1-k^2y^2}} \\ &= \frac{1}{\pi}\int_{-1}^1\frac{dy}{\sqrt{1-y^2}\sqrt{1-k^2y^2}}\end{aligned} \tag{44}$$

Membandingkan pers (18) dan (18a) maka ungkapan tersebut akan sesuai dengan bentuk quadratur numerik *Gauss-Chebyshev* yaitu

$$w(y) = \frac{1}{\sqrt{1-y^2}}; \quad \text{dan} \quad f(y) = \frac{1}{\sqrt{1-k^2y^2}} \tag{45}$$

```
function fung(y, k)
    return 1.0 / sqrt(1.0 - (k*y)^2)
end

function kuad_GC(k, n)
    x, c = gausschebyshev(n)
    sum = 0.0
    for i in 1:n
        sum += c[i] * fung(x[i], k)
    end
    kuad = sum
    return kuad
end

theta0 = pi/200.0
k = sin(theta0/2.0)
kuad = kuad_GC(k, 20)
tau = kuad / pi;

tau

1.0000154214748775
```

0.4 Penyelesaian Persamaan Gerak Bandul

0.4.1 Masalah Syarat Awal dengan Metode Runge-Kutta

Persamaan gerak benda bermassa m yang digantungkan pada seutas tali dengan panjang l dan massa diabaikan, di bawah pengaruh medan gravitasi bumi g, adalah

$$m\frac{d^2x(t)}{dt^2} = -mg\sin\theta(t) \tag{46}$$

Dalam radian, mengingat $x(t) = l\theta(t)$ dengan $\theta(t)$ adalah sudut simpangan benda terhadap titik setimbang, maka persamaan gerak tersebut dapat dinyatakan dalam bentuk persamaan diferensial orde dua berikut

$$\frac{d^2\theta(t)}{dt^2} = -\frac{g}{l}\sin\theta(t) \tag{47}$$

Diambil satuan universal atau besaran tak berdimensi bagi besaran waktu yaitu

$$\tau = \frac{t}{\omega_0} \tag{48}$$

Dalam satuan universal tersebut maka persamaan gerak bagi bandul akan dapat diungkapkan dalam bentuk berikut.

$$\frac{d^2\theta(\tau)}{d\tau^2} = -\sin\theta(\tau) \tag{49}$$

Penyelesaian persamaan diferensial tersebut secara analitik akan cukup sulit karena berbentuk nonlinear dalam $\theta(\tau)$. Untuk keadaan khusus, pada simpangan kecil sedemikian hingga $\sin\theta(\tau) \approx \theta(\tau)$ maka pers (4) di atas dapat didekati oleh bentuk

$$\frac{d^2\theta(\tau)}{d\tau^2} \approx -\theta(\tau) \tag{50}$$

Pada simpangan kecil ini, umumnya terpenuhi saat simpangan $\theta < 10^\circ$, maka penyelesaian pers (5) berbentuk

$$\theta(\tau) = \theta_0\cos(\tau) + v_0\sin(\tau) \tag{51}$$

Ungkapan tersebut diperoleh dengan asumsi syarat awal bahwa pada saat awal $\tau = 0$ maka benda berada pada sudut simpangan maksimum di $\theta = \theta_0$ dan kecepatan sudut awal $\left[\frac{d\theta}{d\tau}\right]_{\tau=0} = v_0 = 0$.

Untuk sebarang simpangan, yang tidak dibatasi pada simpangan kecil, penyelesaian pers (4) menjadi sulit untuk diperoleh secara analitik sehingga bentuk kompak penyelesaian simpangan seperti diberikan oleh pers (6) menjadi tidak berlaku. Salah satu metode untuk penyelesaian persamaan gerak pada sebarang sudut simpangan tersebut adalah dengan menggunakan metode Euler.

0.4.2 Metode Euler untuk Masalah Syarat Awal

Skema Eksplisit: Beda Maju (*Forward Difference*)

Masalah sayrat awal merupakan suatu bentuk permasalahan ketika informasi terkait sistem fisis pada keadaan tertentu yaitu pada keadaan awal atau saat waktu $\tau = 0$ telah diketahui dan kemudian diinginkan informasi terkait sistem fisis tersebut pada sebarang waktu τ.

Untuk sistem bandul yang telah diuraikan di atas maka diasumsikan bahwa nilai sudut simpangan dan kecepatan sudut pada saat waktu $\tau = 0$ secara berurutan adalah θ_0 dan v_0. Persamaan gerak bandul seperti diberikan oleh pers (4) dapat ditulis ulang dalam bentuk berikut.

$$\frac{dv(\tau)}{d\tau} = -\sin\theta(\tau) \tag{52}$$
$$\frac{d\theta(\tau)}{d\tau} = v(\tau) \tag{53}$$

Memanfaatkan salah satu bentuk pendekatan beda hingga (*finite difference*) yaitu beda maju (*forward difference*), ungkapan persamaan diferensial orde satu pada pers (7) dan (8) tersebut akan dapat dinyatakan sebagai bentuk persamaan beda hingga seperti berikut.

$$\left[\frac{dv(\tau)}{d\tau}\right]_{\tau=0} \approx \frac{v_1 - v_0}{\Delta\tau} = -\sin\theta_0 \qquad \Longrightarrow v_1 = v_0 - \Delta\tau\sin\theta_0 \tag{54}$$
$$\left[\frac{d\theta(\tau)}{d\tau}\right]_{\tau=0} \approx \frac{\theta_1 - \theta_0}{\Delta\tau} = v_0 \qquad \Longrightarrow \theta_1 = \theta_0 + \Delta\tau\, v_0 \tag{55}$$

Berdasar ungkapan tersebut maka bentuk umum metode Euler skema eksplisit adalah seperti berikut.

$$v_i = v_{i-1} - \Delta\tau\sin\theta_{i-1} \tag{56}$$
$$\theta_i = \theta_{i-1} + \Delta\tau\, v_{i-1}; \qquad i = 1, 2, 3, \cdots, N \tag{57}$$

Dalam ungkapan tersebut, $\Delta\tau = \tau_i - \tau_{i-1}$, $v_i \equiv v(\tau_i)$ dan $\theta_i \equiv \theta(\tau_i)$.

Metode tersebut disebut sebagai metode Euler skema eksplisit karena nilai v_i dan θ_i dapat langsung diperoleh ketika nilai-nilai sebelumnya yaitu v_{i-1} dan θ_{i-1} telah diketahui.

Algoritma Penyelesaian Gerak Bandul dengan Skema Eksplisit

Berawal dari keadaan nilai sudut simpangan θ_0 dan kecepatan sudut v_0, maka prosedur untuk memperoleh sudut simpangan dan kecepatan sudut pada sebarang nilai τ adalah seperti berikut

1. Berikan nilai masukan θ_0, v_0 dan $\Delta\tau$
2. Hitung nilai sudut simpangan θ_1 dan kecepatan sudut v_1 berdasar ungkapan pers (11) dan (12)
3. Ulangi langkah pada butir 2 untuk memperoleh sudut simpangan θ_i dan kecepatan sudut v_i himgga $i = N$

Metode untuk Validasi Hasil

Salah satu cara untuk melekukan pengecekan bahwa hasil komputasi terkait persamaan gerak bandul telah valid atau sahih adalah dengan memantau nilai tenaga total E pada sebarang waktu τ. Sistem bandul merupakan sistem konservatif, yaitu tenaga total sistem akan konstan, tidak bergantung pada waktu τ. Dengan demikian kriteria bahwa hasil komputasi adalah akurat apabila nilai tenaga total E tidak berubah secara signifikan pada sebarang waktu τ. Ungkapan tenaga total E untuk sistem bandul memiliki bentuk berikut.

$$E = \frac{1}{2}v^2(\tau) + [1 - \cos\theta(\tau)] \tag{58}$$

```
function bandul_eksplisit(tmax, N, theta0, omega0)
    tau = range(0.0, tmax, length=N)
    dtau = tau[2] - tau[1]
    theta = zeros(N)
    omega = zeros(N)
    tenaga = zeros(N)
```

```
    theta[1] = theta0
    omega[1] = omega0
    tenaga[1] = omega[1]^2/2.0 + (1.0 - cos(theta[1]))
    for i in 1:N -1
        theta[i+1] = theta[i] + dtau * omega[i]
        omega[i+1] = omega[i] - dtau * sin(theta[i])
        tenaga[i+1] = omega[i+1]^2/2.0 + (1.0 - cos(theta[i+1]))
    end
    return tau, theta, omega, tenaga
end
```

```
bandul_eksplisit (generic function with 1 method)
```

```
tmax=10
N=100
theta0=pi/10.0
omega0=0.0
```

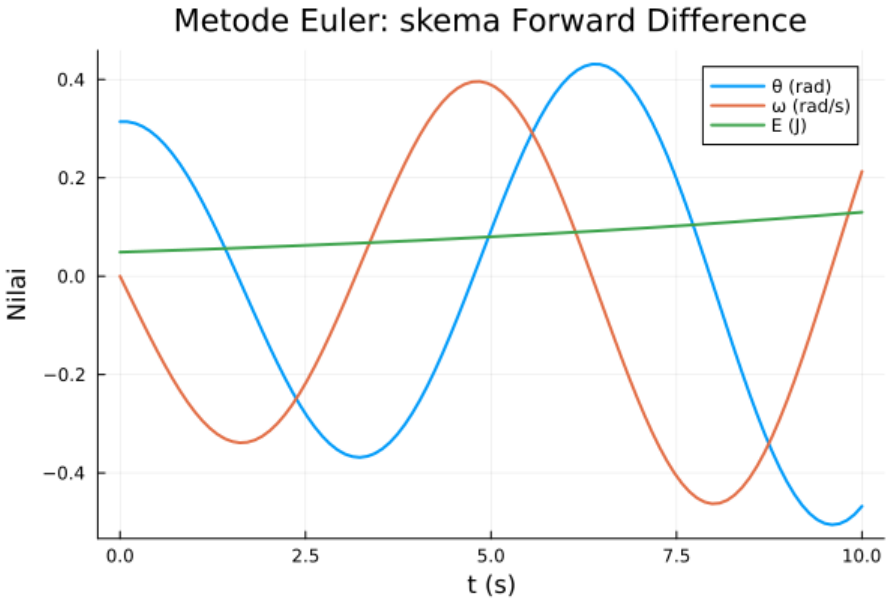
```
0.0
```

```
tau,theta,omega,tenaga=bandul_eksplisit(tmax,N,theta0,omega0)
```

```
(0.0:0.10101010101010101:10.0, [0.3141592653589793, 0.3141592653589793, 0.3110063524483075, 0.
```

```
using Plots
```

```
plot(tau,[theta,omega,tenaga], label=[" \theta (rad)" " \omega (rad/s)" "E (J)"], xlabel="t (s
```



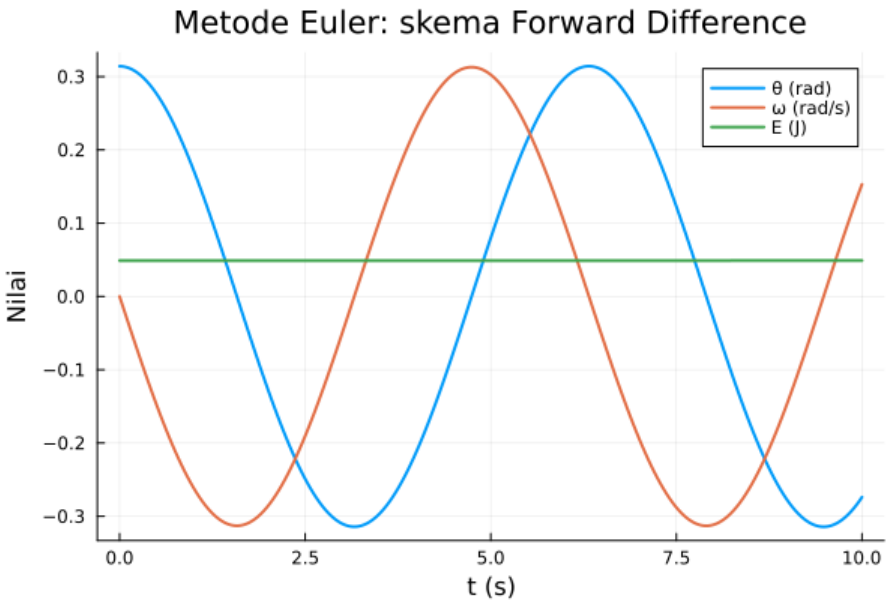

```
tmax=10
N=100000
theta0=pi/10.0
omega0=0.0
```

```
0.0
```

```
tau,theta,omega,tenaga=bandul_eksplisit(tmax,N,theta0,omega0)
```

```
(0.0:0.000100001000010001:10.0, [0.3141592653589793, 0.3141592653589793, 0.3141592622687476,
```

```
plot(tau,[theta,omega,tenaga], label=[" \theta (rad)" " \omega (rad/s)" "E (J)"], xlabel="t (s
```



Dua hasil pada nilai $\Delta\tau$ yang berbeda di atas menunjukkan bahwa hasil komputasi persamaan gerak bandul dengan metode Euler skema eksplisit akan sahih dan teliti ketika nilai $\Delta\tau$ adalah begitu kecil. Ketika $\Delta\tau$ tidak cukup kecil maka hasil komputasi menjadi kurang sahih karena nilai tenaga total E nampak tidak konstan. Hasil ini merupakan indikator bahwa metode Euler skema eksplisit memiliki kelemahan yaitu berpotensi untuk tidak stabil.

Skema Implisit: Beda Mundur (*Backward Difference*)

Upaya unuk mencapai hasil komputasi yang stabil akan dapat dilakukan dengan menggunakan metode Euler skema implisit. Berbeda dengan metode Euler skema eksplisit yang memanfaatkan pendekatan beda maju (*forward difference*), maka metode Euler skema impisit akan memanfaatkan pendekatan beda mundur (*backward difference*) bagi operasi diferensial orde satu dalam bentuk berikut.

$$\left[\frac{dv(\tau)}{d\tau}\right]_{\tau=\tau_1} \approx \frac{v_1 - v_0}{\Delta\tau} = -\sin\theta_1 \qquad \Longrightarrow v_1 = v_0 - \Delta\tau \sin\theta_1 \tag{59}$$

$$\left[\frac{d\theta(\tau)}{d\tau}\right]_{\tau=\tau_1} \approx \frac{\theta_1 - \theta_0}{\Delta\tau} = v_1 \qquad \Longrightarrow \theta_1 = \theta_0 + \Delta\tau\, v_1 \tag{60}$$

Berdasar ungkapan tersebut maka bentuk umum metode Euler skema implisit adalah seperti berikut.

$$v_i = v_{i-1} - \Delta\tau \sin\theta_i \tag{61}$$

$$\theta_i = \theta_{i-1} + \Delta\tau\, v_i; \qquad i = 1, 2, 3, \cdots, N \tag{62}$$

Metode tersebut disebut sebagai metode Euler skema implisit karena nilai v_i dan θ_i tidak otomatis dapat langsung diperoleh ketika nilai-nilai sebelumnya yaitu v_{i-1} dan θ_{i-1} telah diketahui.

Salah satu cara untuk mendapatkan nilai v_i dan θ_i berdasar metode Euler skema implisit tersebut adalah dengan memanfaatkan pencarian akar suatu fungsi (*root finding*) atau pencarian titik nol (*zeros*) menggunakan metode Newton-Raphson.

Bentuk fungsi yang akan digunakan dalam proses pencarian akar dapat disusun dengan mensubstitusikan pers (16) ke pers (17) sehingga diperoleh ungkapan seperti berikut.

$$\theta_i = \theta_{i-1} + \Delta\tau\,(v_{i-1} - \Delta\tau \sin\theta_i) \tag{63}$$

Atau dapat ditulis ulang dalam bentuk:

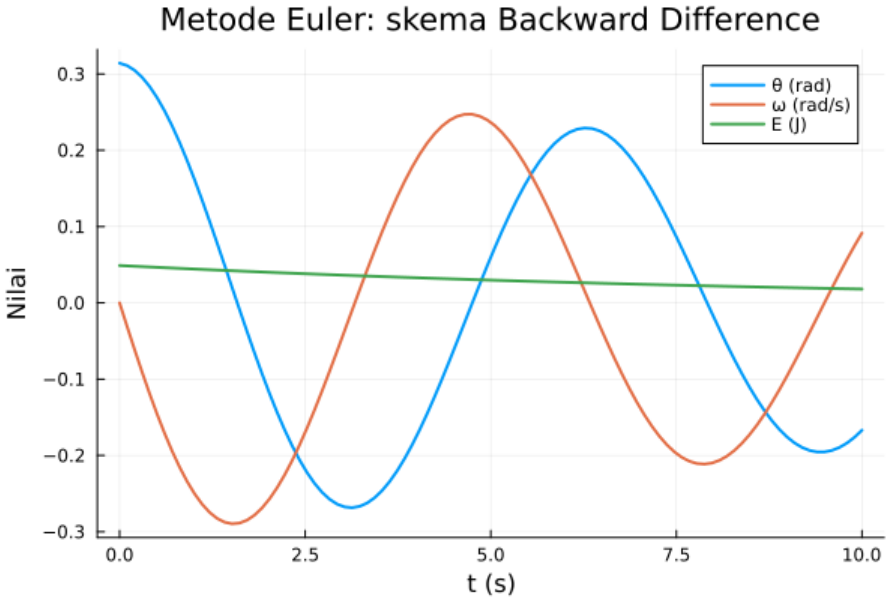
$$f(\theta_i) = \theta_i + \Delta\tau^2 \sin\theta_i - \Delta\tau v_{i-1} - \theta_{i-1} = 0 \tag{64}$$

Penggunaan metode Newton-Raphson dapat memanfaatkan *source code* yang telah disampaikan pada materi kuliah atau menggunakan modul Scipy yaitu scipy.optimize.newton.

Nilai coba θ_i, yaitu θ_i^0 yang diperlukan untuk menjalankan metode Newton-Raphson dapat dipilih dari penyelesaian saat simpangan kecil seperti diberikan oleh pers (6) yaitu

$$\theta_i^0 = \theta_0 \cos(\tau_i) + v_0 \sin(\tau_i)$$

```
using Roots

function bandul_implisit(tmax, N, theta0, omega0)
    tau = range(0.0, tmax, length=N)
    dtau = tau[2] - tau[1]
    theta = zeros(N)
    omega = zeros(N)
    tenaga = zeros(N)
    theta[1] = theta0
    omega[1] = omega0
    tenaga[1] = omega[1]^2/2.0 + (1.0 - cos(theta[1]))
    for i in 1:N -1
        x_init = theta0 * cos(tau[i+1]) + omega0 * sin(tau[i+1])
        x0 = theta[i]
        v0 = omega[i]
        f(x) = x + dtau^2 * sin(x) - dtau * v0 - x0
        df(x) = 1.0 + dtau^2 * cos(x)
        theta[i+1] = find_zero((f, df), x_init, Roots.Newton())
        omega[i+1] = omega[i] - dtau * sin(theta[i+1])
        tenaga[i+1] = omega[i+1]^2/2.0 + (1.0 - cos(theta[i+1]))
    end
    return tau, theta, omega, tenaga
end

bandul_implisit (generic function with 1 method)

tmax=10
N=100
theta0=pi/10.0
omega0=0.0

0.0

tau,theta,omega,tenaga=bandul_implisit(tmax,N,theta0,omega0)

(0.0:0.10101010101010101:10.0, [0.3141592653589793, 0.31103666841500766, 0.30485161220506607,

plot(tau,[theta,omega,tenaga], label=[" \theta (rad)" " \omega (rad/s)" "E (J)"], xlabel="t (s
```



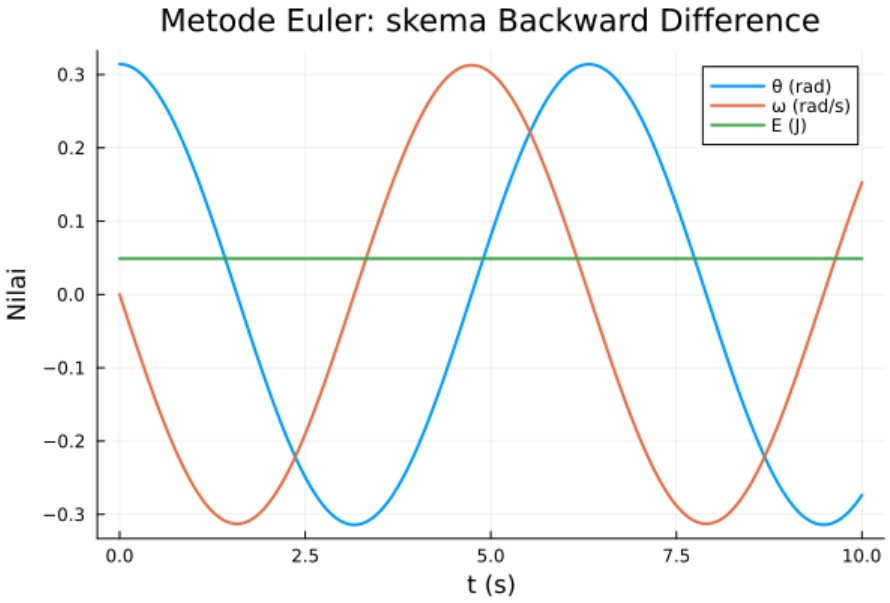

```
tmax=10
N=100000
theta0=pi/10.0
omega0=0.0
```

```
0.0
```

```
tau,theta,omega,tenaga=bandul_implisit(tmax,N,theta0,omega0)
```

```
(0.0:0.0001000010000100001:10.0, [0.3141592653589793, 0.3141592622687476, 0.3141592560882842,
```

```
plot(tau,[theta,omega,tenaga], label=[" \theta (rad)" " \omega (rad/s)" "E (J)"], xlabel="t (s
```



Dua hasil pada nilai $\Delta\tau$ yang berbeda di atas menunjukkan bahwa hasil komputasi persamaan gerak bandul dengan metode Euler skema implisit beda mundur akan sahih dan teliti ketika nilai $\Delta\tau$ adalah begitu kecil. Ketika $\Delta\tau$ tidak cukup kecil maka hasil komputasi menjadi kurang sahih karena nilai tenaga total E nampak tidak konstan. Namun berbeda dengan skema eksplisit, hasil komputasi yang kurang sahih pada skema implist beda mundur ini nampak tidak menunjukkan potensi tidak stabil karena nilai tenaga semakin saat waktu semakin besar. Hasil ini merupakan indikator bahwa metode Euler skema implisit beda mundur memiliki sedikit kelebihan dibanding metode Euler skema eksplisit karena tidak berpotensi untuk tidak stabil, meskipun masih sama-sama memiliki kelemahan yaitu memerlukan nilai $\Delta\tau$ yang begitu kecil.

Skema Implisit: Beda Terpusat (*Central Difference*)

Hasil di atas menunjukkan bahwa masalah ketakstabilan yang terjadi pada metode Euler skema eksplisit akan dapat dihindari ketika digunakan metode Euler skema implisit. Meskipun metode Euler skema implisit membutuhkan waktu komputasi yang lebih besar, kerena ada tambahan proses pencarian akar, namun adanya jaminan kestabilan menyebabkan metode tersebut lebih menjadi pilihan dibanding metode Euler eksplisit.

Untuk mendapatkan hasil yang lebih teliti, selain jaminan kestabilan, maka dapat memanfaatkan pendekatan beda terpusat (*central difference*) bagi operasi diferensial orde satu dalam bentuk berikut.

$$\left[\frac{dv(\tau)}{d\tau}\right]_{\tau=\tau_{1/2}} \approx \frac{v_1 - v_0}{\Delta\tau} = -\sin\theta_{1/2} \approx -\frac{\sin\theta_0 + \sin\theta_1}{2} \qquad \Longrightarrow v_1 = v_0 - \tag{65}$$

$$\frac{\Delta\tau}{2}\left(\sin\theta_1 + \sin\theta_0\right) \tag{66}$$

$$\left[\frac{d\theta(\tau)}{d\tau}\right]_{\tau=\tau_{1/2}} \approx \frac{\theta_1 - \theta_0}{\Delta\tau} = v_{1/2} \approx \frac{v_0 + v_1}{2} \qquad \Longrightarrow \theta_1 = \theta_0 + \frac{\Delta\tau}{2}\left(v_1 + v_0\right) \tag{67}$$

Berdasar ungkapan tersebut maka bentuk umum metode Euler skema implisit adalah seperti berikut.

$$v_i = v_{i-1} - \frac{\Delta\tau}{2}\left(\sin\theta_i + \sin\theta_{i-1}\right) \tag{68}$$

$$\theta_i = \theta_{i-1} + \frac{\Delta\tau}{2}\left(v_i + v_{i-1}\right); \qquad i = 1, 2, 3, \cdots, N \tag{69}$$

Bentuk fungsi yang akan digunakan dalam proses pencarian akar dapat disusun dengan mensubstitusikan pers (12b) ke pers (13b) sehingga diperoleh ungkapan seperti berikut.

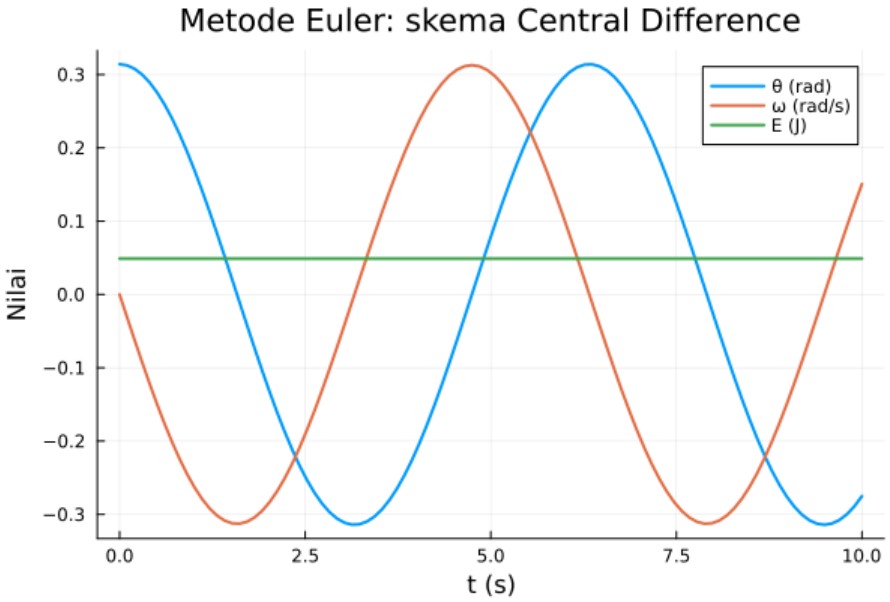
$$\theta_i = \theta_{i-1} + \frac{\Delta\tau}{2}\left[v_{i-1} - \frac{\Delta\tau}{2}\left(\sin\theta_i + \sin\theta_{i-1}\right) + v_{i-1}\right] \tag{70}$$

Atau dapat ditulis ulang dalam bentuk:

$$f(\theta_i) = \theta_i + \frac{\Delta\tau^2}{4}\left(\sin\theta_i + \sin\theta_{i-1}\right) - \Delta\tau v_{i-1} - \theta_{i-1} = 0 \tag{71}$$

```
function bandul_implisit_terpusat(tmax, N, theta0, omega0)
    tau = range(0.0, tmax, length=N)
    dtau = tau[2] - tau[1]
    theta = zeros(N)
    omega = zeros(N)
    tenaga = zeros(N)
    theta[1] = theta0
    omega[1] = omega0
    tenaga[1] = omega[1]^2/2.0 + (1.0 - cos(theta[1]))
    for i in 1:N -1
        x0 = theta[i]
        v0 = omega[i]
        # Initial guess using small angle solution
        x_init = theta0 * cos(tau[i+1]) + omega0 * sin(tau[i+1])
        f(x) = x + dtau^2 * sin(x)/4.0 + dtau^2 * sin(x0)/4.0 - dtau * v0 - x0
        df(x) = 1.0 + dtau^2 * cos(x)/4.0
        theta[i+1] = find_zero((f, df), x_init, Roots.Newton())
        omega[i+1] = omega[i] - dtau * sin(theta[i+1])/2.0 - dtau * sin(theta[i])/2.0
        tenaga[i+1] = omega[i+1]^2/2.0 + (1.0 - cos(theta[i+1]))
```

```
    end
    return tau, theta, omega, tenaga
end

bandul_implisit_terpusat (generic function with 1 method)

tmax=10
N=100
theta0=pi/10.0
omega0=0.0

0.0

tau,theta,omega,tenaga=bandul_implisit_terpusat(tmax,N,theta0,omega0)

(0.0:0.10101010101010101:10.0, [0.3141592653589793, 0.3125866249719053, 0.30788394264310964, 0

plot(tau,[theta,omega,tenaga], label=[" \theta (rad)" " \omega (rad/s)" "E (J)"], xlabel="t (s
```



Nampak bahwa dengan pengambilan $\Delta\tau$ yang tidak terlalu kecil maka metode Euler skema impisit dengan pendekatan beda terpusat dapat memberikan hasil yang teliti dan stabil.

0.4.3 Metode Runge-Kutta

Runge-Kutta Orde 2

Berdasar uraian dan contoh di atas maka dapat dipahami bahwa metode Euler memiliki kelemahan untuk menyelesaikan masalah syarat awal. Metode Euler skema eksplisit memiliki kelemahan pada ketakstabilan serta ketelitian hasil sehingga membutuhkan interval waktu $\Delta\tau$ yang amat kecil. Metode Euler skema implisit beda mundur (*backward difference*) mampu mengatasi ketakstabilan namun masih membutuhkan interval waktu $\Delta\tau$ yang amat kecil. Metode Euler skema implisit beda terpusat (*central difference*) mampu mengatasi ketakstabilan dan ketelitian yang tidak membutuhkan interval waktu $\Delta\tau$ yang amat kecil. Meskipun demikian, metode Euler skema implisit beda terpusat (*central difference*) masih memiliki kekurangan yaitu memerlukan langkah komputasi yang relatif mahal karena melibatkan pencarian akar untuk tiap nilai. Metode Runge-Kutta mengatasi kekurangan yang ada pada metode Euler, terutama dalam hal yang terkait dengan ketelitian hasil serta tidak memerlukan langkah komputasi yang relatif mahal karena berbentuk eksplisit.

Tinjau metode Euler skema implisit beda terpusat (*central difference*) dalam bentuk ungkapan pers (12a) berikut.

$$\left[\frac{dv(\tau)}{d\tau}\right]_{\tau=\tau_{1/2}} \approx \frac{v_1 - v_0}{\Delta\tau} = -\sin\theta_{1/2} \tag{72}$$

Jika dianggap bahwa $\sin\theta_{1/2} \equiv \sin\left[\theta\left(\tau_0 + \frac{\Delta\tau}{2}\right)\right]$ dapat diperoleh berdasar pendekatan deret hingga orde 1 yaitu

$$\sin\theta_{1/2} \equiv \sin\left[\theta\left(\tau_0 + \tfrac{\Delta\tau}{2}\right)\right] \approx \sin\left(\theta_0 + \frac{\Delta\tau}{2}\left[\frac{d\theta}{d\tau}\right]_{\tau=\tau_0}\right) \approx \sin\left(\theta_0 + \frac{\Delta\tau}{2}v_0\right) \tag{73}$$

Dengan cara yang sama maka bentuk ungkapan pers (13a) juga dapat disajikan seperti ungkapan pers (15) dan (16) tersebut. Maka metode Euler skema implisit beda terpusat (*central difference*) dapat diselesaikan dengan langkah eksplisit berikut.

$$k_1^{(1)} = -\sin\theta_0 \tag{74}$$
$$k_1^{(2)} = v_0 \tag{75}$$
$$k_2^{(1)} = -\sin\left(\theta_0 + \frac{\Delta\tau}{2}k_1^{(2)}\right) \tag{76}$$
$$k_2^{(2)} = v_0 + \frac{\Delta\tau}{2}k_1^{(1)} \tag{77}$$
$$v_1 = v_0 + \Delta\tau k_2^{(1)} \tag{78}$$
$$\theta_1 = \theta_0 + \Delta\tau k_2^{(2)} \tag{79}$$

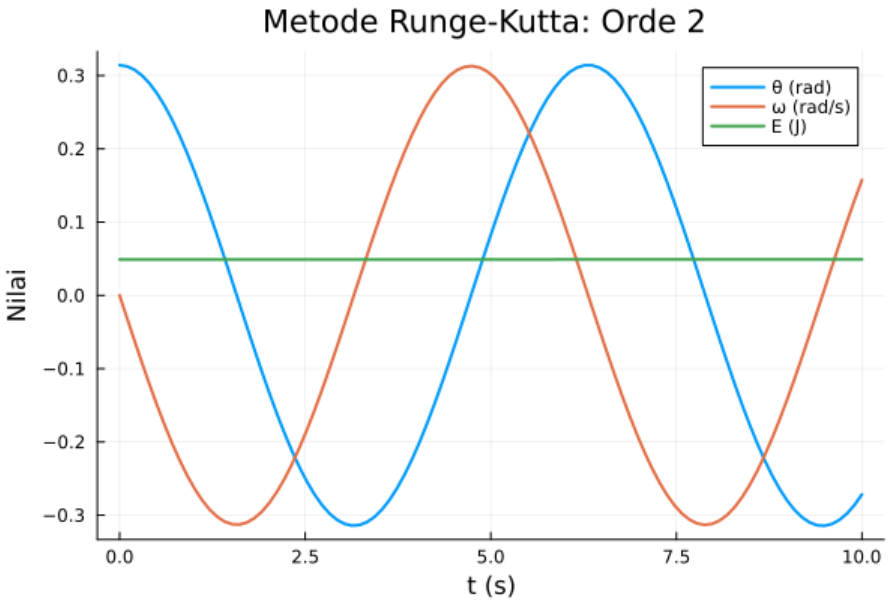
Ungkapan pada pers (17) tersebut dapat dilakukan secara berulang pada setiap putaran ke i dengan $i = 1, 2, 3\cdots, N$. Langkah seperti diberikan oleh pers (17) disebut sebagai metode Runge-Kutta orde 2 dengan bentuk umum seperti berikut.

$$k_1^{(1)} = -\sin\theta_{i-1} \tag{80}$$
$$k_1^{(2)} = v_{i-1} \tag{81}$$
$$k_2^{(1)} = -\sin\left(\theta_{i-1} + \frac{\Delta\tau}{2}k_1^{(2)}\right) \tag{82}$$
$$k_2^{(2)} = v_{i-1} + \frac{\Delta\tau}{2}k_1^{(1)} \tag{83}$$
$$v_i = v_{i-1} + \Delta\tau k_2^{(1)} \tag{84}$$
$$\theta_i = \theta_{i-1} + \Delta\tau k_2^{(2)}; \qquad i = 1, 2, 3, \cdots, N \tag{85}$$

Source code berikut merupakan implementasi metode Runge-Kutta orde 2.

```
function bandul_RK2(tmax, N, theta0, omega0)
    tau = range(0.0, tmax, length=N)
    dtau = tau[2] - tau[1]
    theta = zeros(N)
    omega = zeros(N)
    tenaga = zeros(N)
    theta[1] = theta0
    omega[1] = omega0
    tenaga[1] = omega[1]^2/2.0 + (1.0 - cos(theta[1]))
    for i in 1:N -1
        k1_1 = -sin(theta[i])
        k1_2 = omega[i]
        k2_1 = -sin(theta[i] + dtau * k1_2 / 2.0)
```

```
        k2_2 = omega[i] + dtau * k1_1 / 2.0
        omega[i+1] = omega[i] + dtau * k2_1
        theta[i+1] = theta[i] + dtau * k2_2
        tenaga[i+1] = omega[i+1]^2/2.0 + (1.0 - cos(theta[i+1]))
    end
    return tau, theta, omega, tenaga
end

bandul_RK2 (generic function with 1 method)

tmax=10
N=100
theta0=pi/10.0
omega0=0.0

0.0

tau,theta,omega,tenaga=bandul_RK2(tmax,N,theta0,omega0)

(0.0:0.10101010101010101:10.0, [0.3141592653589793, 0.31258280890364337, 0.30786109019853874,

plot(tau,[theta,omega,tenaga], label=[" \theta (rad)" " \omega (rad/s)" "E (J)"], xlabel="t (s
```



Nampak dari hasil di atas bahwa metode Runget-Kutta orde 2 dapat memberikan hasil yang teliti seperti metode Euler skema implisit beda terpusat (*central difference*), dengan tambahan fitur yaitu berdasar skema eksplisit.

Runge-Kutta Orde 4

Metode Runge-Kutta orde 2 yang diuraikan di atas memiliki ketelitian sebanding dengan orde $\Delta\tau^3$. APabila diinginkan ketelitian yang lebih tinggi hingga sebanding dengan orde $\Delta\tau^5$, maka dapat digunakan metode Runge-Kutta orde 4 dengan langkah seperti diberikan berikut.

$$k_1^{(1)} = -\sin\theta_{i-1} \tag{86}$$
$$k_1^{(2)} = v_{i-1} \tag{87}$$
$$k_2^{(1)} = -\sin\left(\theta_{i-1} + \frac{\Delta\tau}{2}k_1^{(2)}\right) \tag{88}$$
$$k_2^{(2)} = v_{i-1} + \frac{\Delta\tau}{2}k_1^{(1)} \tag{89}$$
$$k_3^{(1)} = -\sin\left(\theta_{i-1} + \frac{\Delta\tau}{2}k_2^{(2)}\right) \tag{90}$$
$$k_3^{(2)} = v_{i-1} + \frac{\Delta\tau}{2}k_2^{(1)} \tag{91}$$
$$k_4^{(1)} = -\sin\left(\theta_{i-1} + k_3^{(2)}\right) \tag{92}$$
$$k_4^{(2)} = v_{i-1} + k_3^{(1)} \tag{93}$$
$$v_i = v_{i-1} + \frac{\Delta\tau}{6}\left(k_1^{(1)} + 2.0 * k_2^{(1)} + 2.0 * k_3^{(1)} + k_4^{(1)}\right) \tag{94}$$
$$\theta_i = \theta_{i-1} + \frac{\Delta\tau}{6}\left(k_1^{(2)} + 2.0 * k_2^{(2)} + 2.0 * k_3^{(2)} + k_4^{(2)}\right); \qquad i = 1, 2, 3, \cdots, N \tag{95}$$

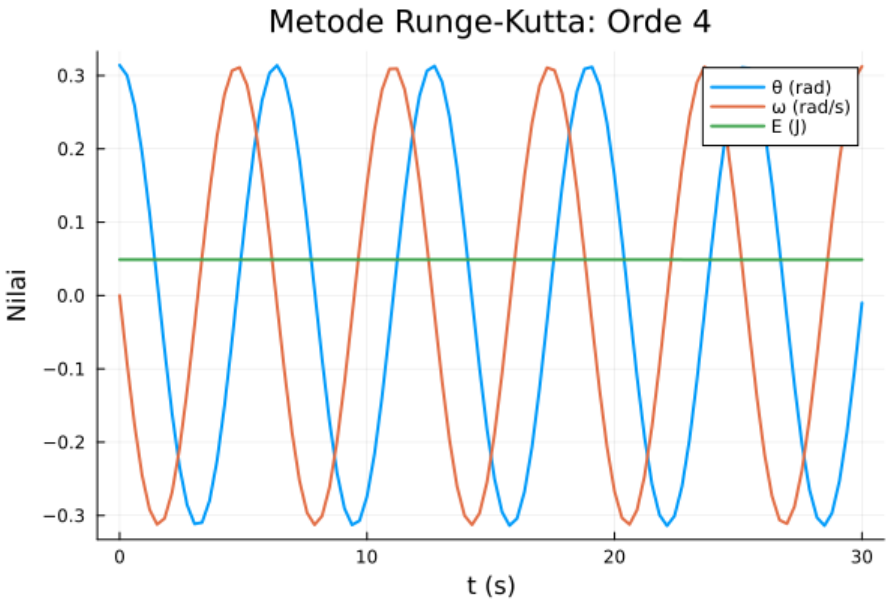
Source code berikut merupakan implementasi metode Runge-Kutta orde 4.

```
function bandul_RK4(tmax, N, theta0, omega0)
    tau = range(0.0, tmax, length=N)
    dtau = tau[2] - tau[1]
    theta = zeros(N)
    omega = zeros(N)
    tenaga = zeros(N)
    theta[1] = theta0
    omega[1] = omega0
    tenaga[1] = omega[1]^2/2.0 + (1.0 - cos(theta[1]))
    for i in 1:N -1
        k1_1 = -sin(theta[i])
        k1_2 = omega[i]
        k2_1 = -sin(theta[i] + dtau * k1_2 / 2.0)
        k2_2 = omega[i] + dtau * k1_1 / 2.0
        k3_1 = -sin(theta[i] + dtau * k2_2 / 2.0)
        k3_2 = omega[i] + dtau * k2_1 / 2.0
        k4_1 = -sin(theta[i] + dtau * k3_2)
        k4_2 = omega[i] + dtau * k3_1
        omega[i+1] = omega[i] + dtau * (k1_1 + 2.0*k2_1 + 2.0*k3_1 + k4_1) / 6.0
        theta[i+1] = theta[i] + dtau * (k1_2 + 2.0*k2_2 + 2.0*k3_2 + k4_2) / 6.0
        tenaga[i+1] = omega[i+1]^2/2.0 + (1.0 - cos(theta[i+1]))
    end
    return tau, theta, omega, tenaga
end

bandul_RK4 (generic function with 1 method)

tmax=30
N=100
theta0=pi/10.0
omega0=0.0

0.0
```

```
tau,theta,omega,tenaga=bandul_RK4(tmax,N,theta0,omega0)
```

```
(0.0:0.30303030303030304:30.0, [0.3141592653589793, 0.3000745329181524, 0.2590497949653009, 0.
```

```
plot(tau,[theta,omega,tenaga], label=[" \theta (rad)" " \omega (rad/s)" "E (J)"], xlabel="t (s
```



Nampak dari hasil di atas bahwa metode Runget-Kutta orde 4 dapat memberikan hasil yang teliti bahkan ketika diambil nilai $\Delta\tau = 0.3$.

0.5 Penyelesaian Persamaan Poisson

0.5.1 Masalah Syarat Batas

Beberapa masalah fisika dapat disajikan oleh model yang diwakili oleh persamaan *Poisson* dalam bentuk berikut.

$$\nabla^2\phi(\vec{r}) = f(\vec{r}) \tag{96}$$

Dalam ungkapan di atas, ∇^2 merupakan operator Laplasan sedang $\phi(\vec{r})$ dan $f(\vec{r})$ merupakan fungsi bernilai komplek atau real. Penyelesaian persamaan *Poisson* merupakan upaya untuk memperoleh fungsi $\phi(\vec{r})$ ketika $f(\vec{r})$ diketahui atau diberikan.

Sebagai gambaran, dalam ruang 3 dimensi pada koordinat Kartesan maka persamaan *Poisson* akan memiliki bentuk berikut.

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial x^2}\right)\phi(x, y, x) = f(x, y, z) \tag{97}$$

Salah satu contoh sistem fisis yang memenuhi persamaan *Poisson* adalah masalah untuk menemukan distribusi potensial gravitasi pada seluruh ruang $\phi(r)$ oleh adanya tarikan medan gravitasi akibat rapat massa ρ. Ungkapan persamaan *Poisson* untuk potensial gravitasi mengambil bentuk berikut.

$$\nabla^2\phi = 4\pi G\rho \tag{98}$$

dengan G adalah tetapan gravitasi umum.

Penyelesaian bagi pers (2) untuk benda berbentuk titik dengan massa m akan memberikan potensial gravitasi $\phi(r)$ pada suatu jarak radial r seperti berikut.

$$\phi(r) = -\frac{Gm}{r} \tag{99}$$

Bentuk di atas tidak lain merupakan ungkapan Hukum Newton tentang gravitasi umum.

Contoh lain sistem fisis yang memenuhi persamaan *Poisson* adalah masalah untuk menemukan distribusi potensial listrik pada seluruh ruang $V(r)$ oleh adanya distribusi rapat muatan ρ. Ungkapan persamaan *Poisson* untuk potensial listrik mengambil bentuk berikut.

$$\nabla^2 V = -\frac{\rho}{\epsilon} \tag{100}$$

dengan ϵ adalah permitivitas medium.

Penyelesaian bagi pers (3) untuk muatan berbentuk titik dengan besar q akan memberikan potensial listrik $V(r)$ pada suatu jarak radial r seperti berikut.

$$V(r) = \frac{q}{4\pi\epsilon r} \tag{101}$$

Bentuk di atas tidak lain merupakan ungkapan Hukum Coulomb tentang listrik statis.

Secara umum, saat distribusi rapat massa atau rapat muatan berbentuk sebarang maka penyelesaian persamaan *Poisson* menjadi tidak mudah dan diperlukan prosedur komputasi untuk menyelesaikan masalah tersebut.

0.5.2 Penyelesaian dalam Ruang Satu Dimensi (1-D)

Sebagai gambaran, diberikan bentuk rapat muatan yang tergantung pada jarak radial saja yaitu

$$\rho(r) = \epsilon e^{-r} \tag{102}$$

Mengingat terjadinya simetri bola, maka persamaan *Poisson* dalam ruang tiga dimensi dapat dibawa ke masalah pada ruang satu dimensi menjadi

$$\frac{1}{r^2}\frac{d}{dr}\left(r^2\frac{dV(r)}{dr}\right) = \frac{d^2V(r)}{dr^2} + \frac{2}{r}\frac{dV(r)}{dr} = -e^{-r} \tag{103}$$

Ungkapan tersebut dapat dinyatakan dalam bentuk berikut.

$$r\frac{d^2V(r)}{dr^2} + 2\frac{dV(r)}{dr} = -re^{-r} \tag{104}$$

Penyelesaian pers (5) dapat dilakukan secara analitik dengan bentuk potensial berikut:

$$V(r) = \frac{2}{r} - \left(\frac{2}{r} + 1\right)e^{-r} \tag{105}$$

Karena $\rho(r)$ tidak singular di $r = 0$ maka $V(0)$ bernilai berhingga dan memiliki perilaku seperti potensial Coulomb di $r \to \infty$ yaitu $V(r \to \infty) = 1/r$. Berdasar penyelesaian analitik tersebut maka syarat batas pada $r = 0$ dan $r \to \infty$ dapat diambil dalam nilai berikut.

$$V(0) = 1; \qquad V(r \to \infty) = \frac{2}{r} \tag{106}$$

0.5.3 Metode Beda Hingga (*Finite Difference*)

Masalah syarat batas merupakan suatu bentuk permasalahan ketika informasi terkait sistem fisis pada keadaan tertentu yaitu pada bagian batas dari ruang yang ditinjau telah diketahui dan kemudian diinginkan informasi terkait sistem fisis tersebut pada daerah yang dilingkupi oleh batas tersebut.

1. Untuk ruang 1 dimensi, yang dimaksud bagian batas adalah 2 titik pada bagian ujung sumbu.
2. Untuk ruang 2 dimensi, yang dimaksud bagian batas adalah titik-titik yang mengelilingi bagian tepi bidang.
3. Sedangkan untuk ruang 3 dimensi, yang dimaksud bagian batas adalah titik-titik yang melingkupi bagian permukaan ruang.

Pendekatan beda hingga (*finite difference*) menggunakan skema beda terpusat (*central difference*) bagi bentuk turunan satu kali dan dua kali sebarang fungsi $f(x)$ adalah sebagai berikut.

$$\left.\frac{df(x)}{dx}\right]_{x=x_i} \approx \frac{f_{i+1} - f_{i-1}}{2h} \tag{107}$$

$$\left.\frac{d^2f(x)}{dx^2}\right]_{x=x_i} \approx \frac{f_{i+1} - 2f_i + f_{i-1}}{h^2} \tag{108}$$

Dalam ungkapan di atas, $h = x_i - x_{i-1}, x_i = x_0 + ih$ dan $f_i \equiv f(x_i)$.

Dengan pendekatan tersebut maka persamaan diferensial dalam pers (5b) akan dapat dinyatakan dalam bentuk persamaan beda hingga yaitu

$$r_i\left(\frac{V_{i+1} - 2V_i + V_{i-1}}{h^2}\right) + 2\left(\frac{V_{i+1} - V_{i-1}}{2h}\right) = -r_ie^{-r_i} \tag{109}$$

Ungkapan tersebut dapat dinyatakan dalam bentuk lain sebagai berikut.

$$(r_i - h)\,V_{i-1} - 2r_iV_i + (r_i + h)\,V_{i+1} = r_{i-1}V_{i-1} - 2r_iV_i + r_{i+1}V_{i+1} = -h^2r_ie^{-r_i} \tag{110}$$

Ungkapan bagi syarat batas dapat dinyatakan sebagai $V_0 = 1$ dan $V_{N+1} = \frac{2}{r_{N+1}}$ dimana $r_0 = 0$ dan N adalah bilangan bulat positip yang diambil bernilai cukup besar sedemikian hingga $r_{N+1} \to \infty$.

Dengan memperhitungkan nilai pada kedua batas yaitu $V_0 = 1$ dan $V_{N+1} = \frac{2}{r_{N+1}}$ maka pers (7) akan berujud menjadi sejumlah N persamaan simultan yang mengandung N variabel yang perlu dicari yaitu $\{V_1, V_2, \ldots, V_N\}$. Apabila sejumlah besaran yang akan dicari tersebut, yaitu

$\{V_i; i = 1.2, 3, \ldots, N\}$ dinyatakan dalam bentuk matrik kolom V, maka sejumlah N persamaan simultan tersebut dapat ditulis dalam bentuk perkalian matrik sebagai

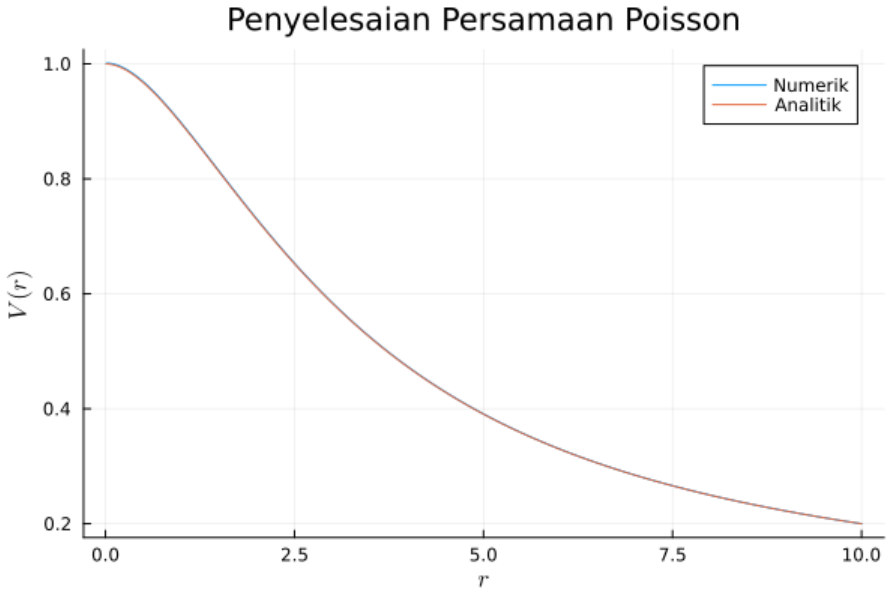
$$AV = b \tag{111}$$

Dalam bentuk eksplisit akan berbentuk berikut

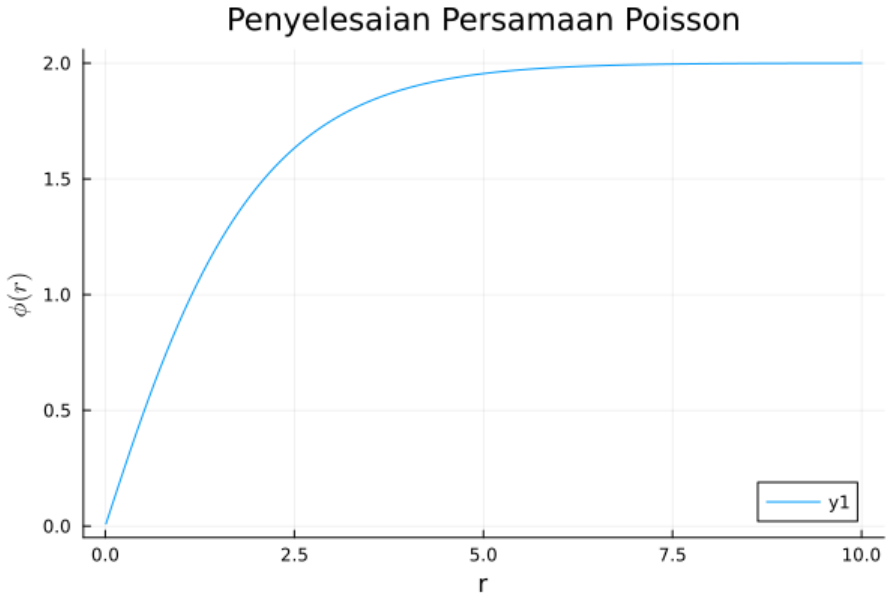
$$\begin{pmatrix} -2r_1 & r_2 & 0 & \ldots & \ldots & 0 \\ r_1 & -2r_2 & r_3 & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & \ldots & 0 & r_{N-2} & -2r_{N-1} & r_N \\ 0 & \ldots & \ldots & 0 & r_{N-1} & -2r_N \end{pmatrix} \begin{pmatrix} V_1 \\ V_2 \\ \vdots \\ \vdots \\ V_{N-1} \\ V_N \end{pmatrix} = \begin{pmatrix} -h^2\, r_1\, e^{-r_1} - r_0 V_0 \\ -h^2\, r_2\, e^{-r_2} \\ \vdots \\ \vdots \\ -h^2\, r_{N-1}\, e^{-r_{N-1}} \\ -h^2\, r_N\, e^{-r_N} - r_{N+1} V_{N+1} \end{pmatrix} \tag{112}$$

Berikut adalah *source-code* bagi uraian prosedur komputasi tersebut.

```
using LinearAlgebra

rmax = 10.0
N = 1000
h = rmax / (N - 1)
r = collect(range(h, rmax; length=N))
r0 = 0.0
rN1 = rmax + h
V0 = 1.0
VN1 = 2.0 / rN1
V_ref = 2.0 ./ r . - (2.0 ./ r .+ 1.0) .* exp.( -r);

function gen_A(N, r)
    b1 = r[2:end]
    b2 = -2.0 .* r
    b3 = r[1:end -1]
    A = diagm(1 => b1) + diagm(0 => b2) + diagm( -1 => b3)
    return A
end

gen_A (generic function with 1 method)

function gen_b(N, h, r, V0, VN1, r0, rN1)
    b = -h^2 .* r .* exp.( -r)
    b[1] -= r0 * V0
    b[N] -= rN1 * VN1
    return b
end

gen_b (generic function with 1 method)

A=gen_A(N,r)
b=gen_b(N,h,r,V0,VN1,r0,rN1);

V = A \ b;

using Plots
using LaTeXStrings

plot(r,[V,V_ref],label=["Numerik" "Analitik"],xlabel=L"r", ylabel=L"V(r)", title="Penyelesaian
```




```
phi = V .* r
plot(r,phi,xlabel="r", ylabel=L"$\phi(r)$", title="Penyelesaian Persamaan Poisson")
```



0.5.4 Dekomposisi LU

Dalam kebanyakan permasalahan, rapat muatan dalam peubah radial dapat diambil pada bentuk beragam dan kemudian ditentukan konfigurasi potensial listrik untuk berbagai bentuk tersebut. Untuk permasalahan semacam ini, maka prosedur komputasi dekomposisi LU akan lebih tepat untuk dipilih karena operasi yang melibatkan matrik A, yaitu operasi eleminasi atau invers matrik, cukup dilakukan satu kali dan untuk seterusnya cukup menggunakan operasi substitusi maju dan substitusi mundur.

Seperti yang telah diuraikan pada materi penyelesaian persamaan simultas, prinsip dasar metode dekomposisi LU adalah mengubah matrik persegi A menjadi perkalian dua matrik persegi yaitu matrik segitiga bawah L dengan matrik segitiga atas U dalam bentuk berikut.

$$PA = LU \tag{113}$$

Matrik persegi P disebut matrik permutasi yang berperan untuk mempertukarkan baris atau kolom agar tidak terjadi singularitas. Sifat matrik permutasi adalah $P^T P = I$ atau $P^{-1} = P^T$. Dengan demikian penyelesaian persamaan simultan akan berubah menjadi

$$AV = P^{-1}LUV = P^{-1}LW = P^{-1}Z = b; \qquad \text{dengan} \tag{114}$$

$$P^{-1}Z = b \quad \text{atau} \quad Z = Pb \quad \text{dilanjutkan} \tag{115}$$

$$LW = Z \quad \text{kemudian} \tag{116}$$

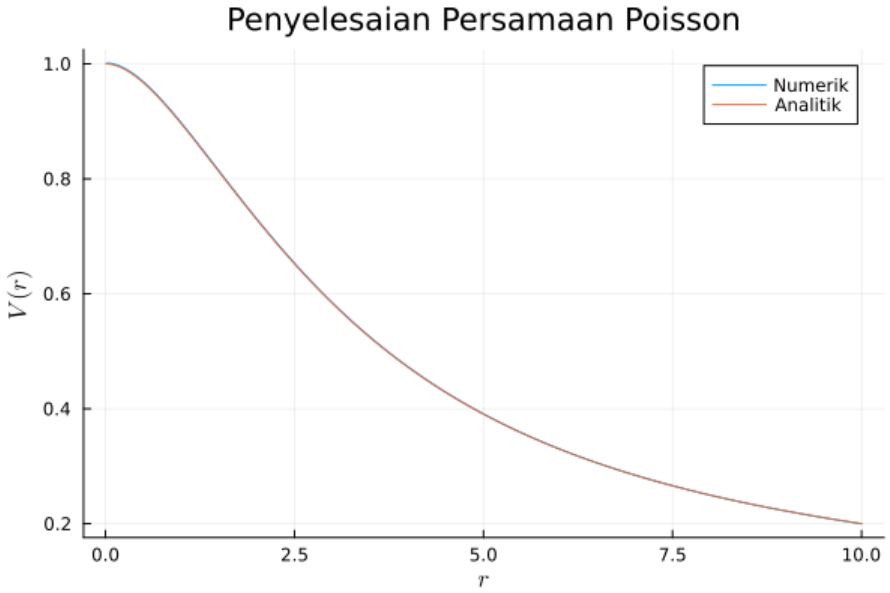
$$UV = W \tag{117}$$

Dalam ungkapan matrik segitiga atas U dan matrik segitiga bawah L diberikan oleh bentuk

$$U = \begin{pmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1N} \\ 0 & u_{22} & \cdots & \cdots & u_{2N} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & u_{NN} \end{pmatrix}; \qquad L = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1\cdots & \cdots & 0 & \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ l_{N1} & l_{N2} & l_{N3} & \cdots & 1 \end{pmatrix} \tag{118}$$

Dengan ungkapan tersebut maka matrik kolom W dapat diperoleh melalui proses substitusi maju, yaitu diperoleh nilai W_1, W_2 dan seterusnya hingga W_N. Setelah matrik kolom W diperoleh, selanjutnya matrik kolom V dihitung melalui proses substitusi balik, yaitu diperoleh nilai V_N, V_{N-1} dan seterusnya hingga V_1.

```
function forw_subs(A, b)
    n = size(A, 1)
    x = zeros(eltype(b), n)
    x[1] = b[1] / A[1, 1]
    for i in 2:n
        x[i] = (b[i] - dot(A[i, 1:i -1], x[1:i -1])) / A[i, i]
    end
    return x
end

forw_subs (generic function with 1 method)

function back_subs(A, b)
    n = size(A, 1)
    x = zeros(eltype(b), n)
    x[end] = b[end] / A[end, end]
    for i in n -1: -1:1
        x[i] = (b[i] - dot(A[i, i+1:end], x[i+1:end])) / A[i, i]
    end
    return x
end

back_subs (generic function with 1 method)

F = lu(A);
L = F.L;
U = F.U;
P = F.P;

Z = P * b
W = forw_subs(L,Z)
V = back_subs(U,W);

plot(r,[V,V_ref],label=["Numerik" "Analitik"],xlabel=L"r", ylabel=L"V(r)", title="Penyelesaian
```



Cara lain yang biasa dilakukan ketika berhadapan dengan penyelesaian bagian radial dari bentuk simetri bola adalah dengan memperkenalkan fungsi $\phi(r)$ yang didefinisikan seperti berikut.

$$\phi(r) = \frac{V(r)}{r}; \quad \text{atau} \quad V(r) = r\phi(r) \tag{119}$$

Dengan bentuk fungsi tersebut maka persamaan *Poisson* akan berbentuk berikut.

$$\frac{1}{r^2}\frac{d}{dr}\left(r^2\frac{dV(r)}{dr}\right) = \frac{1}{r}\frac{d^2\phi(r)}{dr^2} = -e^{-r} \tag{120}$$

Ungkapan tersebut dapat dinyatakan dalam bentuk berikut.

$$\frac{d^2\phi(r)}{dr^2} = -re^{-r} \tag{121}$$

Persamaan diferensial di atas akan dapat dinyatakan dalam bentuk persamaan beda hingga yaitu

$$\left(\frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{h^2}\right) = -r_i e^{-r_i} \tag{122}$$

Ungkapan bagi syarat batas dapat dinyatakan sebagai $\phi_0 = 0$ dan $\phi_{N+1} = 2$ dimana $r_0 = 0$ dan N adalah bilangan bulat positip yang diambil bernilai cukup besar sedemikian hingga $r_{N+1} \to \infty$.

Dengan memperhitungkan nilai pada kedua batas yaitu $\phi_0 = 0$ dan $\phi_{N+1} = 2$ maka tersebut akan berujud menjadi sejumlah N persamaan simultan yang mengandung N variabel yang perlu dicari yaitu $\{\phi_1, \phi_2, \ldots, \phi_N\}$. Sejumlah N persamaan simultan tersebut dapat ditulis dalam bentuk perkalian matrik sebagai

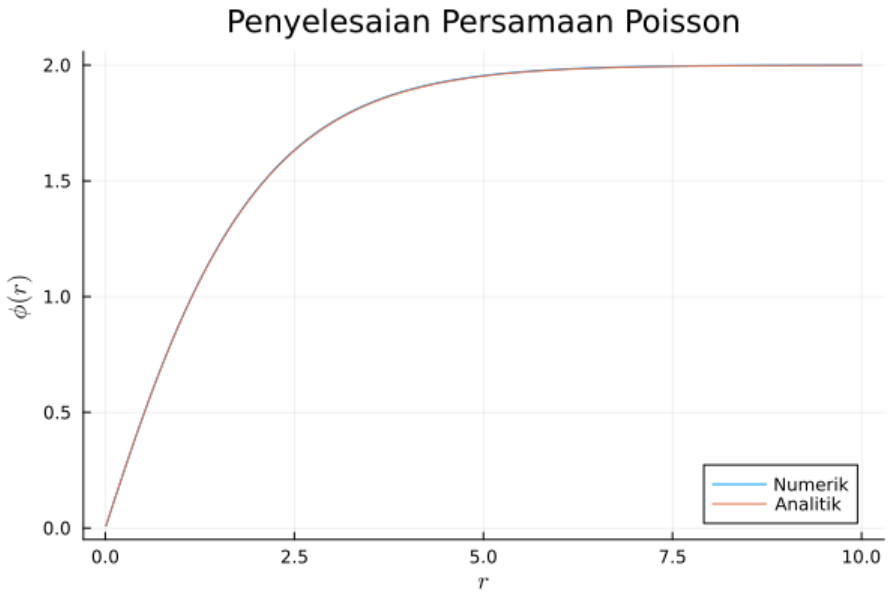
$$A\,\phi = b \tag{123}$$

Dalam bentuk eksplisit, ungkapan tersebut mengambil bentuk

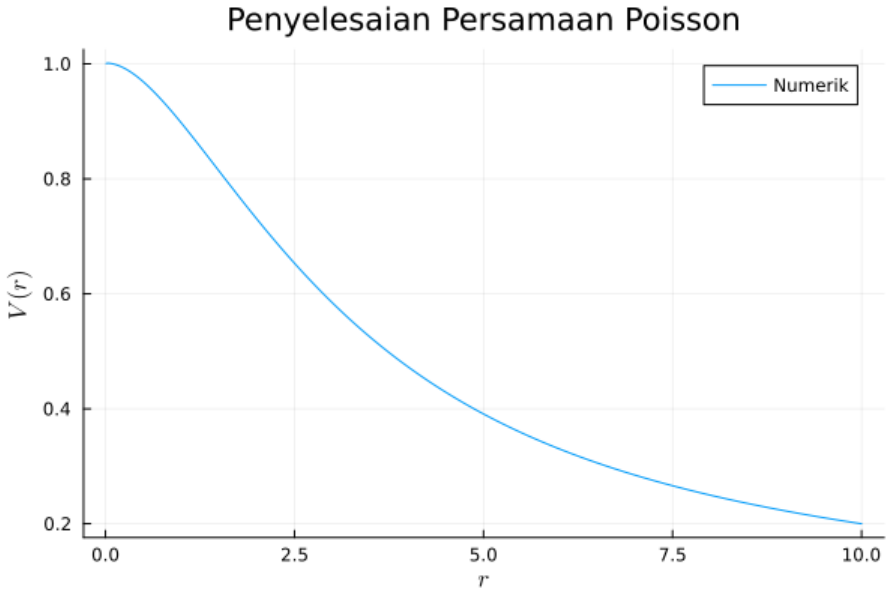
$$\begin{pmatrix} -2 & 1 & 0 & \ldots & \ldots & 0 \\ 1 & -2 & 1 & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & \ldots & 0 & 1 & -2 & 1 \\ 0 & \ldots & \ldots & 0 & 1 & -2 \end{pmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \vdots \\ \phi_{N-1} \\ \phi_N \end{pmatrix} = \begin{pmatrix} -h^2\, r_1\, e^{-r_1} - \phi_0 \\ -h^2\, r_2\, e^{-r_2} \\ \vdots \\ \vdots \\ -h^2\, r_{N-1}\, e^{-r_{N-1}} \\ -h^2\, r_N\, e^{-r_N} - \phi_{N+1} \end{pmatrix} \tag{124}$$

Berikut adalah *source-code* bagi uraian prosedur komputasi tersebut.

```
rmax = 10.0
N = 1000
h = rmax / (N - 1)
r = collect(range(h, rmax; length=N))
Phi0 = 0.0
PhiN1 = 2.0
Phi_ref = 2.0 . - (2.0 .+ r) .* exp.( -r);

function gen_APhi(N)
    b1 = ones(N -1)
    b2 = -2.0 .* ones(N)
    b3 = ones(N -1)
    APhi = diagm(1 => b1) + diagm(0 => b2) + diagm( -1 => b3)
    return APhi
end

gen_APhi (generic function with 1 method)

function gen_bPhi(N, h, r, Phi0, PhiN1)
    bPhi = -h^2 .* r .* exp.( -r)
    bPhi[1] -= Phi0
    bPhi[N] -= PhiN1
    return bPhi
end

gen_bPhi (generic function with 1 method)

APhi=gen_APhi(N)
bPhi=gen_bPhi(N,h,r,Phi0,PhiN1);

Phi = APhi \ bPhi;

plot(r,[Phi,Phi_ref],label=["Numerik" "Analitik"],xlabel=L"r", ylabel=L"\phi(r)", title="Penye
```




```
V = Phi ./ r;

plot(r,V,label=["Numerik" "Analitik"],xlabel=L"r", ylabel=L"V(r)", title="Penyelesaian Persama
```



0.5.5 Metode Iterasi Gauss-Seidel

Sebarang matrik persegi A dapat diungkapkan dalam bentuk berikut.

$$A = U + L_D \tag{125}$$

dengan

$$U = \begin{pmatrix} 0 & a_{12} & u_{13} & \cdots & a_{1N} \\ 0 & 0 & \cdots & \cdots & a_{2N} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{pmatrix}; \quad L_D = \begin{pmatrix} a_{11} & 0 & 0 & \cdots & 0 \\ a_{21} & a_{22} \cdots & \cdots & 0 & \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{N1} & a_{N2} & a_{N3} & \cdots & a_{NN} \end{pmatrix} \tag{126}$$

Dengan demikian penyelesaian persamaan simultan akan berubah menjadi

$$Av = (U + L_D)v = b; \qquad \text{menjadi} \quad v = L_D^{-1}(b - Uv) \tag{127}$$

Penyelesaian berdasar bentuk eksplisit persamaan simultan untuk penentuan $V(r)$

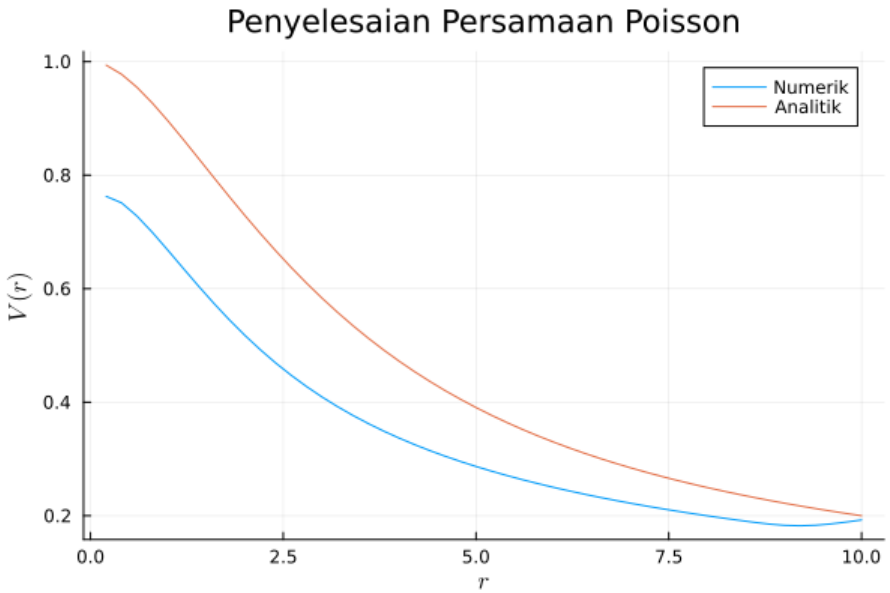
Bentuk persamaan diferensial yang mewakili masalah tersebut dalam koordinat bola pada bagian radial adalah

$$r\frac{d^2V(r)}{dr^2} + 2\frac{dV(r)}{dr} = -re^{-r} \tag{128}$$

Ungkapan *source -code* untuk penyelesaian berdasar metode iterasi Gauss-Seidel adalah seperti berikut.

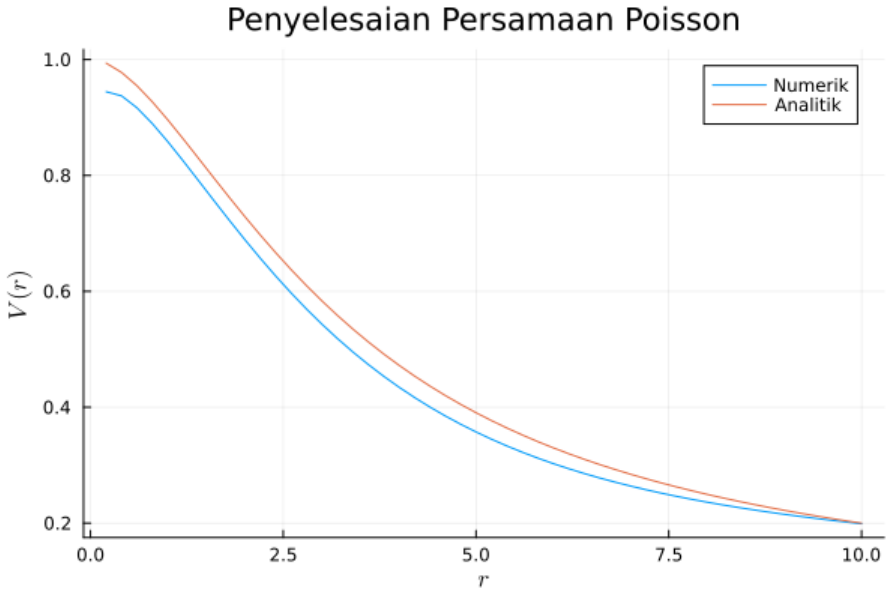
```
function gauss_seidel(A, b, Iter, V_trial)
    n = length(b)
    for _ in 1:Iter
        for i in 1:n
            s1 = dot(A[i,1:i -1], V_trial[1:i -1])
            s2 = dot(A[i,i+1:end], V_trial[i+1:end])
            V_trial[i] = (b[i] - s1 - s2) / A[i,i]
        end
    end
    return V_trial
end
```

```
gauss_seidel (generic function with 1 method)

rmax = 10.0
N = 50
h = rmax / (N - 1)
r = range(h, stop=rmax, length=N)
r0 = 0.0
rN1 = rmax + h
V0 = 1.0
VN1 = 2.0 / rN1
V_ref = 2.0 ./ r . - (2.0 ./ r .+ 1.0) .* exp.( -r);

A = gen_A(N,r)
b = gen_b(N,h,r,V0,VN1,r0,rN1);

Iter = 7
#V_trial = ones(N)
#V_trial = 1.0 . - 0.08 .* r
V_trial = 2.0 ./(r .+ 2.0);

V = gauss_seidel(A,b,Iter,V_trial);

plot(r,[V, V_ref],label=["Numerik" "Analitik"],xlabel=L"r", ylabel=L"V(r)", title="Penyelesaia
```



Nampak bahwa hingga iterasi ke 7 maka hasil nilai coba untuk masih menyimpang jauh dibanding nilai rujukan. Namun ketika iterasi dinaikkan hingga 900 kali maka nilai coba untuk sudah mulai sesuai dan konvergen pada nilai rujukan.

```
Iter = 300;
#V_trial = ones(N);
#V_trial = 1.0 . - 0.08 .* r;
V_trial = 2.0 ./(r .+ 2.0);

V = gauss_seidel(A,b,Iter,V_trial);

plot(r,[V, V_ref],label=["Numerik" "Analitik"],xlabel=L"r", ylabel=L"V(r)", title="Penyelesaia
```

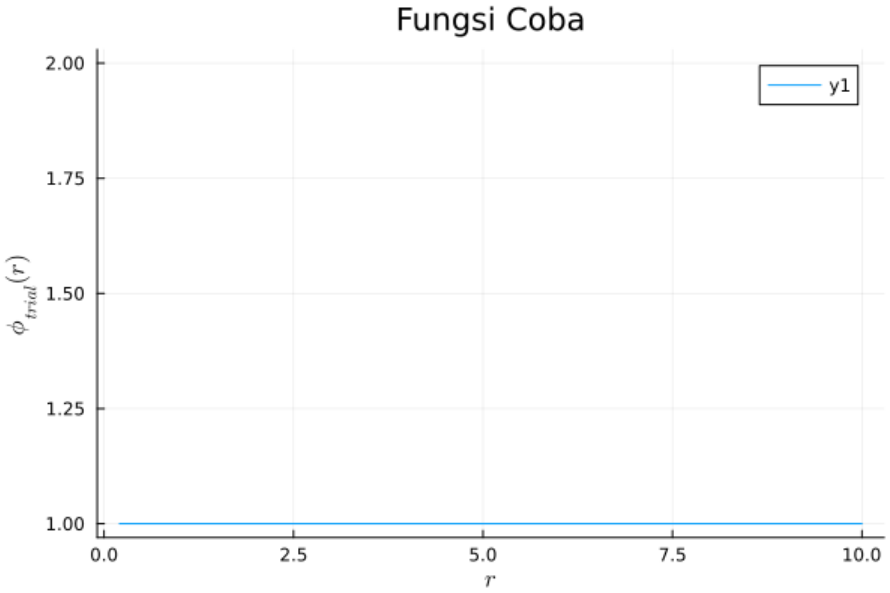


Penyelesaian berdasar bentuk eksplisit persamaan simultan untuk penentuan $\phi(r)$

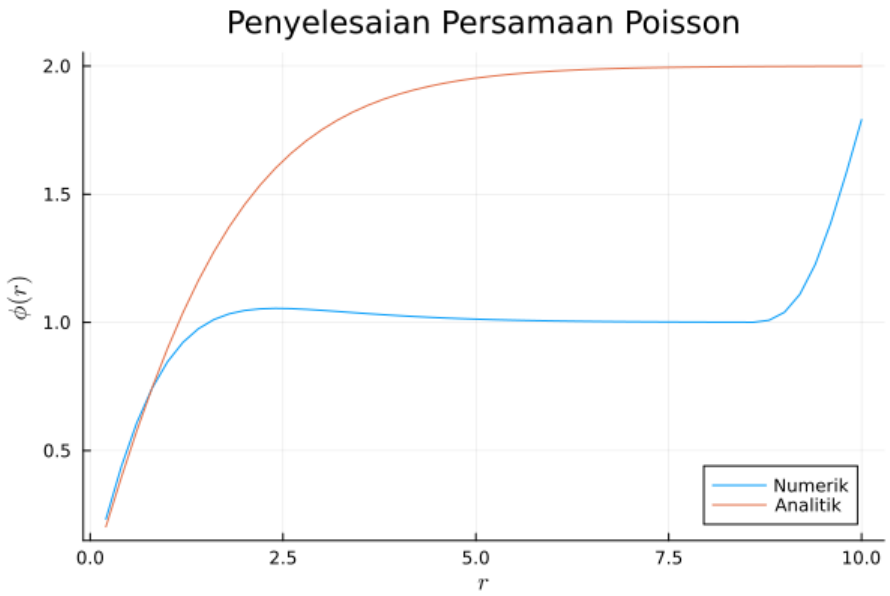
Ungkapan persamaan diferensial dapat dinyatakan dalam bentuk berikut.

$$\frac{d^2\phi(r)}{dr^2} = -re^{-r} \tag{129}$$

```
rmax = 10.0
N = 50
h = rmax / (N - 1)
r = reshape(collect(range(h, stop=rmax, length=N)), N, 1)
Phi0 = 0.0
PhiN1 = 2.0
Phi_ref = 2.0 . - (2.0 .+ r) .* exp.( -r);

using Random

Iter = 7
#Phi_trial = 2.0 .* rand(N,1);
Phi_trial =  ones(N);

plot(r,Phi_trial,xlabel=L"r", ylabel=L"\phi_{trial}(r)", title="Fungsi Coba")
```



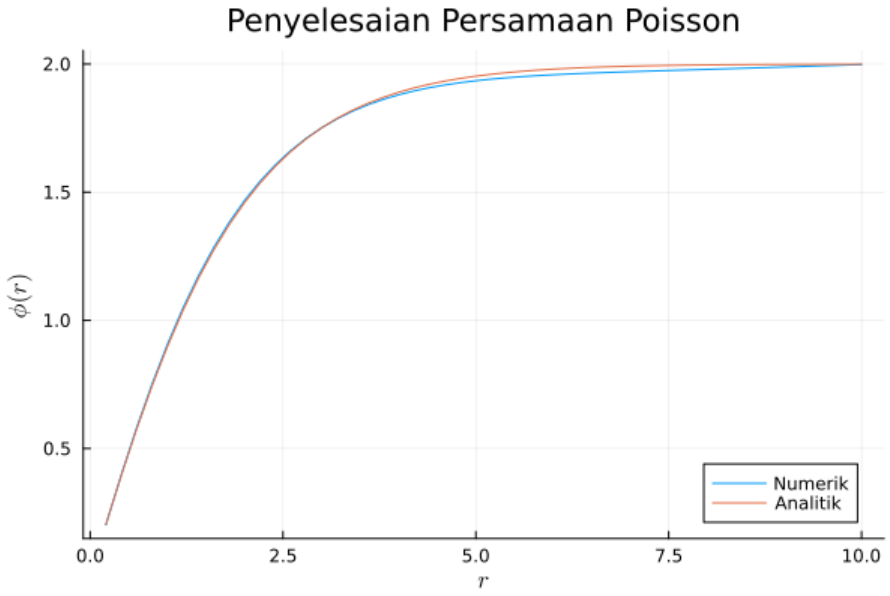

```
A = gen_APhi(N)
b = gen_bPhi(N,h,r,Phi0,PhiN1);

Phi = gauss_seidel(A,b,Iter,Phi_trial);

plot(r,[Phi, Phi_ref],label=["Numerik" "Analitik"],xlabel=L"r", ylabel=L"\phi(r)", title="Peny
```



Seperti pada komputasi sebelumnya , nampak bahwa hingga iterasi ke 7 maka hasil nilai coba untuk masih menyimpang jauh dibanding nilai rujukan. Namun ketika iterasi dinaikkan hingga 900 kali maka nilai coba untuk sudah mulai sesuai dan konvergen pada nilai rujukan.

```
Iter = 800
Phi_trial = 2.0 .* rand(N,1);
#Phi_trial =  ones(N);

50 \times1 Matrix{Float64}:
 1.570291671790907
 1.0638433205165263
 1.4354324705538528
 1.5761123639673826
```

```
 0.8841796918876843
 1.8933700368025626
 1.476129422010953
 1.7370035353176678
 1.6598413866191244
 1.128757950710294

 0.09826808980255741
 1.437382312536897
 0.46519320506126505
 0.13281398020644164
 1.0456996326082266
 1.6981288402762222
 0.38107607614714434
 0.7086351009577123
 1.9176327912787687

A = gen_APhi(N)
b = gen_bPhi(N,h,r,Phi0,PhiN1);

Phi = gauss_seidel(A,b,Iter,Phi_trial);

plot(r,[Phi, Phi_ref],label=["Numerik" "Analitik"],xlabel=L"r", ylabel=L"\phi(r)", title="Peny
```



0.5.6 Penyelesaian dalam Ruang Dua Dimensi (2-D)

Sebagai gambaran untuk masalah dalam bidang $x - y$ maka diberikan bentuk rapat muatan dengan bentuk berikut

$$\rho(x, y) = \epsilon e^{-xy} \tag{130}$$

Tinjau masalah untuk menemukan potensial $V(x, y)$ akibat adanya rapat muatan $\rho(x, y)$ pada bidang bentuk persegi panjang di daerah $0 \leq x \leq L_x$ dan $0 \leq y \leq L_y$ sebagai penyelesaian dari persamaan *Poisson* berikut.

$$\frac{\partial^2 V(x, y)}{\partial x^2} + \frac{\partial^2 V(x, y)}{\partial y^2} = -e^{-xy} \tag{131}$$

Anggap syarat batas yang harus dipenuhi agar didapatkan penyelesaian yang unik adalah berjenis *Derichlet* yaitu nilai $V(x,y)$ sepanjang tepi persegi panjang diketahui sebesar $V(0,y) = V_0, V(L_x,y) = V_0, V(x,0) = V_0$ dan $V(x,L_y) = V_0$.

Membagi persegi panjang ke dalam $N_x \times N_y$ buah kisi yang masing-masing kisis memiliki luas $h_x \times h_y$ sedemikian hingga

$$x_i = ih_x; \quad i = 0,1,2,\cdots,N_x \tag{132}$$
$$y_j = jh_y; \quad j = 0,1,2,\cdots,N_y \tag{133}$$

maka ungkapan beda hingga dari pers~(12) adalah

$$\frac{V_{i-1,j} - 2V_{i,j} + V_{i+1,j}}{h_x^2} + \frac{V_{i,j-1} - 2V_{i,j} + V_{i,j+1}}{h_y^2} = -e^{-x_i y_j} \tag{134}$$

Dalam ungkapan di atas, notasi $V_{i,j} \equiv V(x_i, y_j)$.

Implementasi Syarat Batas

Pada titik-titik bagian dalam bidang Pada titik-titik kisi yang tidak bersinggungan dengan bagian batas bidang persegi panjang, persamaan tersebut dapat disederhanakan menjadi

$$h_y^2 V_{i-1,j} + h_y^2 V_{i+1,j} - 2(h_x^2 + h_y^2)V_{i,j} + h_x^2 V_{i,j-1} + h_x^2 V_{i,j+1} = -h_x^2 h_y^2 e^{-x_i y_j} \tag{135}$$

Dalam persamaan di atas, $i = 2,3,\cdots,N_x - 2$ dan $j = 2,3,\cdots,N_y - 2$.

Pada titik-titik bagian tepi bidang Untuk memenuhi syarat batas maka di dekat bidang batas persegi panjang akan terbentuk persamaan berikut.

Titik-titik di tepi kiri bidang

$$h_y^2 V_{2,j} - 2(h_x^2 + h_y^2)V_{1,j} + h_x^2 V_{1,j-1} + h_x^2 V_{1,j+1} = -h_x^2 h_y^2 e^{-x_1 y_j} - h_y^2 V_{0,j} \tag{136}$$

Titik-titik di tepi kanan bidang

$$h_y^2 V_{N_x-2,j} - 2(h_x^2 + h_y^2)V_{N_x-1,j} + h_x^2 V_{N_x-1,j-1} + h_x^2 V_{N_x-1,j+1} = \tag{137}$$
$$- h_x^2 h_y^2 e^{-x_{N_x-1} y_j} - h_y^2 V_{N_x,j} \tag{138}$$

dengan $j = 2,3,\cdots,N_y - 2$.

Titik-titik di tepi bawah bidang

$$h_y^2 V_{i-1,1} + h_y^2 V_{i+1,1} - 2(h_x^2 + h_y^2)V_{i,1} + h_x^2 V_{i,2} = -h_x^2 h_y^2 e^{-x_i y_1} - h_x^2 V_{i,0} \tag{139}$$

Titik-titik di tepi atas bidang

$$h_y^2 V_{i-1,N_y-1} + h_y^2 V_{i+1,N_y-1} - 2(h_x^2 + h_y^2)V_{i,N_y-1} + h_x^2 V_{i,N_y-12} = \tag{140}$$
$$- h_x^2 h_y^2 e^{-x_i y_{N_y-1}} - h_x^2 V_{i,N_y} \tag{141}$$

dengan $i = 2,3,\cdots,N_x - 2$.

Pada empat titik di pojok bidang Adapun pada keempat pojok persegi panjang akan memenuhi persamaan berikut.

Titik pojok kiri bawah bidang

$$h_y^2 V_{2,1} - 2(h_x^2 + h_y^2)V_{1,1} + h_x^2 V_{1,2} = -h_x^2 h_y^2 e^{-x_1 y_1} - h_y^2 V_{0,1} \tag{142}$$
$$- h_x^2 V_{1,0} \tag{143}$$

Titik pojok kanan atas bidang

$$h_y^2 V_{N_x-2,N_y-1} - 2(h_x^2 + h_y^2)V_{N_x-1,N_y-1} + h_x^2 V_{N_x-1,N_y-2} = -h_x^2 h_y^2 e^{-x_{N_x-1} y_{N_y-1}} \tag{144}$$
$$- h_y^2 V_{N_x,N_y-1} - h_x^2 V_{N_x-1,N_y} \tag{145}$$

Titik pojok kanan bawah bidang

$$h_y^2 V_{N_x-2,1} - 2(h_x^2 + h_y^2)V_{N_x-1,1} + h_x^2 V_{N_x-1,2} = -h_x^2 h_y^2 e^{-x_{N_x-1} y_1} \tag{146}$$
$$- h_x^2 V_{N_x-1,0} - h_y^2 V_{N_x,1} \tag{147}$$

Titik pojok kiri atas bidang

$$h_y^2 V_{2,N_y-1} - 2(h_x^2 + h_y^2)V_{1,N_y-1} + h_x^2 V_{1,N_y-2} = -h_x^2 h_y^2 e^{-x_1 y_{N_y-1}} - h_x^2 V_{1,N_y} \tag{148}$$
$$- h_y^2 V_{0,N_y-1} \tag{149}$$

Ungkapan Operasi Matrik

Persamaan-persamaan tersebut dapat disederhanakan ke bentuk operasi matrik apabila diperkenalkan matrik kolom U dengan panjang $N_x - 1 \times N_y - 1$ yang berisi gabungan semua baris dari matrik $V_{i,j}$ oleh kaitan

$$U_k = V_{i,j}; \quad \text{dengan } k = (i-1)(N_y - 1) + j - 1 \tag{150}$$

dan sebaliknya

$$i = \text{int}\left(\frac{k}{N_y - 1}\right) + 1; \quad j = (k) \bmod (N_y - 1) + 1 \tag{151}$$

dengan int(...) berarti bilangan bulat dari operasi pembagian tersebut sedang mod(...) adalah sisa dari operasi pembagian.

Dengan penggunaan indek k tersebut maka terbentuk seperangkat persamaan simultan yaitu

$$-2(h_x^2 + h_y^2)U_1 + h_x^2 U_2 + h_y^2 U_{N_y} = -h_x^2 h_y^2 e^{-x_1 y_1} \tag{152}$$
$$- h_y^2 V_{0,1} - h_x^2 V_{1,0} \tag{153}$$
$$h_x^2 U_1 - 2(h_x^2 + h_y^2)U_2 + h_x^2 V_3 + h_y^2 U_{N_y+1} = -h_x^2 h_y^2 e^{-x_1 y_2} \tag{154}$$
$$- h_y^2 V_{0,2} \tag{155}$$
$$\vdots = \vdots \tag{156}$$
$$h_y^2 U_{k-(N_y-1)} + h_x^2 U_{k-1} - 2(h_x^2 + h_y^2)U_k + h_x^2 U_{k+1} + h_y^2 U_{k+(N_y-1)} = -h_x^2 h_y^2 e^{-x_i y_j} \tag{157}$$
$$\vdots = \vdots \tag{158}$$

Bentuk tersebut tidak lain adalah bentuk matrik

$$AU = b, \tag{159}$$

dengan A merupakan matrik pentadiagonal.

```
xmax = 10.0
ymax = 10.0
Nx = 10
Ny = 10
N = (Nx - 1) * (Ny - 1)
hx = xmax / Nx
hy = ymax / Ny
x = collect(range(0, stop=xmax, length=Nx + 1))
y = collect(range(0, stop=ymax, length=Ny + 1))
V = zeros(Nx + 1, Ny + 1);

function gen_A(Nx, Ny, hx, hy)
    N = (Nx - 1) * (Ny - 1)
    A = zeros(N, N)

    for i in 1:Nx -1
        for j in 1:Ny -1
            k = (i - 1) * (Ny - 1) + (j - 1)

            if i == 1
                if j == 1
                    A[k+1, k+1] = -2.0 * (hx^2 + hy^2)
                    A[k+1, k+2] = hx^2
                    A[k+1, k+1 + (Ny - 1)] = hy^2
                elseif j == Ny - 1
                    A[k+1, k] = hx^2
                    A[k+1, k+1] = -2.0 * (hx^2 + hy^2)
                    A[k+1, k+1 + (Ny - 1)] = hy^2
                else
                    A[k+1, k] = hx^2
                    A[k+1, k+1] = -2.0 * (hx^2 + hy^2)
                    A[k+1, k+2] = hx^2
                    A[k+1, k+1 + (Ny - 1)] = hy^2
                end
            elseif i == Nx - 1
                if j == 1
                    A[k+1, k+1 - (Ny - 1)] = hy^2
                    A[k+1, k+1] = -2.0 * (hx^2 + hy^2)
                    A[k+1, k+2] = hx^2
                elseif j == Ny - 1
                    A[k+1, k+1 - (Ny - 1)] = hy^2
                    A[k+1, k] = hx^2
                    A[k+1, k+1] = -2.0 * (hx^2 + hy^2)
                else
                    A[k+1, k+1 - (Ny - 1)] = hy^2
                    A[k+1, k] = hx^2
                    A[k+1, k+1] = -2.0 * (hx^2 + hy^2)
                    A[k+1, k+2] = hx^2
                end
            else
                if j == 1
                    A[k+1, k+1 - (Ny - 1)] = hy^2
                    A[k+1, k+1] = -2.0 * (hx^2 + hy^2)
                    A[k+1, k+2] = hx^2
```

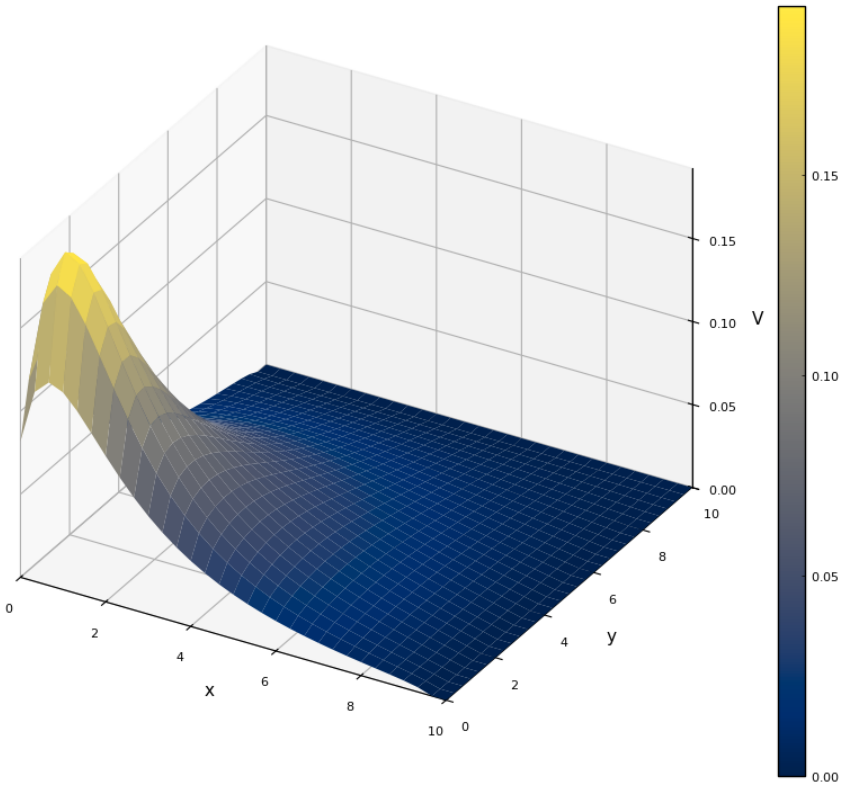
```
                    A[k+1, k+1 + (Ny - 1)] = hy^2
                elseif j == Ny - 1
                    A[k+1, k+1 - (Ny - 1)] = hy^2
                    A[k+1, k] = hx^2
                    A[k+1, k+1] = -2.0 * (hx^2 + hy^2)
                    A[k+1, k+1 + (Ny - 1)] = hy^2
                else
                    A[k+1, k+1 - (Ny - 1)] = hy^2
                    A[k+1, k] = hx^2
                    A[k+1, k+1] = -2.0 * (hx^2 + hy^2)
                    A[k+1, k+2] = hx^2
                    A[k+1, k+1 + (Ny - 1)] = hy^2
                end
            end
        end
    end

    return A
end

gen_A (generic function with 2 methods)

A=gen_A(Nx,Ny,hx,hy);

function gen_b(Nx, Ny, hx, hy, x, y, V)
    N = (Nx - 1) * (Ny - 1)
    b = zeros(N)

    for i in 1:Nx -1
        for j in 1:Ny -1
            k = (i - 1) * (Ny - 1) + (j - 1)

            if i == 1
                if j == 1
                    b[k+1] = -hx^2 * hy^2 * exp( -x[i+1] * y[j+1]) - hy^2 * V[i, j+1] - hx^2 *
                elseif j == Ny - 1
                    b[k+1] = -hx^2 * hy^2 * exp( -x[i+1] * y[j+1]) - hy^2 * V[i, j+1] - hx^2 *
                else
                    b[k+1] = -hx^2 * hy^2 * exp( -x[i+1] * y[j+1]) - hy^2 * V[i, j+1]
                end
            elseif i == Nx - 1
                if j == 1
                    b[k+1] = -hx^2 * hy^2 * exp( -x[i+1] * y[j+1]) - hy^2 * V[i+2, j+1] - hx^2
                elseif j == Ny - 1
                    b[k+1] = -hx^2 * hy^2 * exp( -x[i+1] * y[j+1]) - hy^2 * V[i+2, j+1] - hx^2
                else
                    b[k+1] = -hx^2 * hy^2 * exp( -x[i+1] * y[j+1]) - hy^2 * V[i+2, j+1]
                end
            else
                if j == 1
                    b[k+1] = -hx^2 * hy^2 * exp( -x[i+1] * y[j+1]) - hx^2 * V[i+1, j]
                elseif j == Ny - 1
                    b[k+1] = -hx^2 * hy^2 * exp( -x[i+1] * y[j+1]) - hx^2 * V[i+1, j+2]
                else
```

```
                    b[k+1] = -hx^2 * hy^2 * exp( -x[i+1] * y[j+1])
                end
            end
        end
    end

    return b
end

gen_b (generic function with 1 method)

b=gen_b(Nx, Ny, hx, hy, x, y, V);

using UnicodePlots
UnicodePlots.spy(A)


    1  > 0
       < 0
```

```
   81

      181
      369 \neq 0
```

```
k = 12
i = div(k, Ny - 1) + 1
j = mod(k, Ny - 1) + 1;

i

1

j

13

U = A \ b;

for k in 0:N -1
    i = div(k, Ny - 1) + 1
    j = mod(k, Ny - 1) + 1
    V[i, j] = U[k + 1]
end
```

```
using Plots
pyplot()  # or use ‘pyplot()‘ if you prefer matplotlib -style

# Assuming x, y, and v are already defined
X = repeat(x’, length(y), 1)
Y = repeat(y, 1, length(x))
Z = V  # v should be a matrix of size (length(y), length(x))

# Create the surface plot
plot(
    X, Y, Z,
    st = :surface,
    xlabel = "x",
    ylabel = "y",
    zlabel = "V",
    c = :cividis,
    colorbar = true,
    size = (800, 700)
)
```



0.6 Penyelesaian Persamaan Schrodinger

0.6.1 Implementasi Metode Beda Hingga (*Finite Difference*)

Pendekatan beda hingga (*finte difference*) bagi bentuk turunan kedua suatu fungsi pada titik x_i diberikan oleh ungkapan

$$\left.\frac{d^2f}{dx^2}\right]_{x=x_i} \approx \frac{f_{i-1}-2f_i+f_{i+1}}{(\Delta x)^2} \tag{160}$$

dengan

$$f_i \equiv f(x_i); \qquad \Delta x = x_{i+1}-x_i = \frac{x_N-x_0}{N}; \qquad x_i = x_0+i\Delta x; \qquad i=1,2,3,\cdots,N \tag{161}$$

Bentuk umum persamaan Schrodinger bagi suatu sistem dengan potensial $V(x)$ disajikan dalam bentuk

$$-\frac{\hbar^2}{2m}\frac{d^2\psi(x)}{dx^2}+V(x)\psi(x)=E\psi(x) \tag{162}$$

Dalam satuan universal, persamaan Schrodinger di atas dapat ditulis dalam bentuk

$$-\frac{d^2\psi}{dr^2}+\gamma(r)\psi=\epsilon\psi \tag{163}$$

Dengan pendekatan beda hingga tersebut maka persamaan Schrodinger pada posisi x_i akan dapat disajikan sebagai ungkapan persamaan beda hingga dalam bentuk

$$-\left[\frac{\psi_{i-1}-2\psi_i+\psi_{i+1}}{(\Delta r)^2}\right]+\gamma_i\psi_i=\epsilon\psi_i \tag{164}$$

dengan $\gamma_i \equiv \gamma(r_i)$ dan $\psi_i \equiv \psi(r_i)$.

Dalam ungkapan tersebut, diambil a sebagai faktor skala dalam peubah posisi x sehingga satuan universal dapat dinyatakan dalam bentuk berikut

$$r=\frac{x}{a}; \qquad \gamma(r)=\frac{2ma^2V(x)}{\hbar^2}; \qquad \epsilon=\frac{2ma^2E}{\hbar^2} \tag{165}$$

Ungkapan tersebut dapat disajikan dalam bentuk lain yaitu

$$-\psi_{i-1}+\left[2+(\Delta r)^2\gamma_i-(\Delta r)^2\epsilon\right]\psi_i-\psi_{i+1}=0 \tag{166}$$

Bentuk syarat batas diambil bahwa nilai fungsi gelombang di x_0 dan x_N adalah $\psi_0=0$ dan $\psi_N=0$. Dengan informasi ini maka persamaan beda hingga yang mewakili persamaan Schrodinger akan dapat disajikan dalam bentuk matrik yaitu

$$\begin{pmatrix} 2+(\Delta r)^2\gamma_1-(\Delta r)^2\epsilon & -1 & \cdots & 0 \\ -1 & 2+(\Delta r)^2\gamma_2-(\Delta r)^2\epsilon & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 2+(\Delta r)^2\gamma_{N-1}-(\Delta r)^2\epsilon \end{pmatrix}\begin{pmatrix}\psi_1\\ \psi_2\\ \vdots\\ \psi_{N-1}\end{pmatrix}=\begin{pmatrix}0\\0\\\vdots\\0\end{pmatrix} \tag{167}$$

0.6.2 Diagonalisasi Matrik: Penyelesaian Masalah Nilai *Eigen*

Bentuk operasi matrik seperti diberikan oleh ungkapan di atas merupakan bentuk masalah nilai *eigen* dengan nilai *eigen* adalah ϵ dan fungsi *eigen* diwakili oleh suatu matrik kolom yang memiliki unsur matrik pada baris ke i berupa nilai fungsi gelombang pada posisi x_i yaitu ψ_i.

Tinjau suatu fungsi *eigen* ke j, yaitu ψ_j, yang berkaitan dengan nilai *eigen* $(\Delta r)^2\epsilon_j$ dan memenuhi masalah nilai *eigen* $A\psi_j = (\Delta r)^2\epsilon_j\psi_j$ dalam bentuk berikut:

$$\underbrace{\begin{pmatrix} 2+(\Delta r)^2\gamma_1 & -1 & \cdots & 0 \\ -1 & 2+(\Delta r)^2\gamma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 2+(\Delta r)^2\gamma_{N-1} \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} \psi_{1,j} \\ \psi_{2,j} \\ \vdots \\ \psi_{N-1,j} \end{pmatrix}}_{\psi_j} = (\Delta r)^2\epsilon_j \underbrace{\begin{pmatrix} \psi_{1,j} \\ \psi_{2,j} \\ \vdots \\ \psi_{N-1,j} \end{pmatrix}}_{\psi_j} \tag{168}$$

Berdasar ungkapan matrik tersebut maka dapat ditunjukkan bahwa untuk semua pasangan fungsi *eigen* ψ_j dan nilai *eigen* ϵ_j dengan $j = 1, 2\cdots, N-1$ akan berlaku ungkapan matrik berikut.

$$\underbrace{\begin{pmatrix} 2+(\Delta r)^2\gamma_1 & -1 & \cdots & 0 \\ -1 & 2+(\Delta r)^2\gamma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 2+(\Delta r)^2\gamma_{N-1} \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} \psi_{1,1} & \cdots & \psi_{1,N-1} \\ \psi_{2,1} & \cdots & \psi_{2,N-1} \\ \vdots & \ddots & \vdots \\ \psi_{N-1,1} & \cdots & \psi_{N-1,N-1} \end{pmatrix}}_{P} =$$

$$\underbrace{\begin{pmatrix} \psi_{1,1} & \cdots & \psi_{1,N-1} \\ \psi_{2,1} & \cdots & \psi_{2,N-1} \\ \vdots & \ddots & \vdots \\ \psi_{N-1,1} & \cdots & \psi_{N-1,N-1} \end{pmatrix}}_{P} \underbrace{\begin{pmatrix} (\Delta r)^2\epsilon_1 & 0 & \cdots & 0 \\ 0 & (\Delta r)^2\epsilon_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & (\Delta r)^2\epsilon_{(N-1)} \end{pmatrix}}_{D} \tag{169}$$

Ungkapan matrik $AP = PD$ dalam persamaan tersebut akan dapat ditulis dalam bentuk

$$P^{-1}AP = D \qquad \text{atau} \qquad A = PDP^{-1} \tag{170}$$

Ungkapan seperti diberikan oleh pers (8) disebut sebagai proses diagonalisasi matrik, yaitu suatu proses untuk mengubah sebarang bentuk matrik A ke wakilan matrik diagonal D dan sebaliknya.

Dapat dipahami oleh ungkapan matrik yang mewakili persamaan Schrodinger di atas bahwa proses diagonalisasi matrik akan dapat diperoleh apabila nilai-nilai *eigen* beserta fungsi *eigen* yang terkait telah tersedia. Nilai-nilai *eigen* akan digunakan untuk mengisi unsur-unsur diagonal pada matrik diagonal D, sedangkan fungsi-fungsi *eigen* akan digunakan untuk mengisi tiap kolom pada matrik P.

Pemahaman sebaliknya juga berlaku bahwa apabila proses diagonalisasi matrik telah diperoleh maka nilai-nilai *eigen* beserta fungsi *eigen* yang terkait akan dapat diperoleh.

Dengan kata lain, pencarian masalah nilai eigen serta proses digonalisasi matrik dapat dipahami sebagai proses yang setara.

Pencarian nilai *eigen* untuk matrik tri-diagonal

Untuk bentuk matrik tri-diagonal seperti diuraikan sebelumnya, proses untuk memperoleh nilai *eigen* dapat dilakukan dengan prosedur pencarian titik nol, yaitu menjamin agar determinan matrik adalah nol dalam bentuk

$$\begin{vmatrix} 2+(\Delta r)^2\gamma_1-(\Delta r)^2\epsilon & -1 & \cdots & 0 \\ -1 & 2+(\Delta r)^2\gamma_2-(\Delta r)^2\epsilon & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 2+(\Delta r)^2\gamma_{N-1}-(\Delta r)^2\epsilon \end{vmatrix} = 0 \tag{171}$$

Determinan d_n bagi sebarang bentuk matrik tri-diagonal berorde n disajikan dalam bentuk berikut:

$$d_n = \begin{vmatrix} a_1 & b_1 & \cdots & 0 \\ c_1 & a_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & c_{n-1} & a_n \end{vmatrix} \tag{172}$$

Dapat ditunjukkan bahwa determinan tersebut memiliki sifat yang memenuhi kaitan rekurensi yang melibatkan unsur-unsur matrik a_i, b_i dan c_i dalam bentuk:

$$d_n = a_n d_{n-1} - c_{n-1} b_{n-1} d_{n-2}; \qquad \text{dengan } n \text{ adalah orde matrik} \tag{173}$$

Proses rekurensi dapat dimulai dengan nilai awal $d_1 = a_1$ dan $d_2 = a_1 a_2 - b_1 c_1$.

Dengan memanfaatkan sifat rekurensi bagi determinan suatu matrik tri-diagonal tersebut maka nilai-nilai *eigen* akan dapat diperoleh berdasar pencarian titik nol berdasar metode Bisection atau metode Newton-Raphson berdasar ungkapan

$$f(\epsilon) = d_n(\epsilon) = 0 \tag{174}$$

Untuk sistem kuantum yang sedang ditinjau di atas maka unsur-unsur matrik akan diberikan oleh ungkapan

$$\begin{aligned} a_i &= 2 + (\Delta r)^2 \gamma_i - (\Delta r)^2 \epsilon; \quad \text{untuk} \quad i = 1, 2, \cdots, n \\ b_i &= -1; \qquad \text{dan} \qquad c_i = -1; \quad \textit{untuk} \quad i = 1, 2, \cdots, (n-1) \end{aligned}$$

Pencarian fungsi *eigen* untuk matrik tri-diagonal

Tinjau penyelesaian seperangkat persamaan simultan untuk matrik tri-diagonal dalam bentuk berikut

$$\begin{pmatrix} a_1 & b_1 & \cdots & 0 \\ c_1 & a_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & c_{n-1} & a_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \tag{175}$$

Ungkapan tersebut dapat diubah ke bentuk yang setara, dengan menerapkan langkah eliminasi bagi seluruh unsur c_i, sehingga setelah tercapai langkah pengubahan nilai nol bagi unsur c_i maka didapatkan operasi yang melibatkan matrik segitiga atas (*upper triangular matrix*) dalam bentuk

$$\begin{pmatrix} a_1' & b_1 & \cdots & 0 \\ 0 & a_2' & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & a_n' \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1' \\ y_2' \\ \vdots \\ y_n' \end{pmatrix} \tag{176}$$

Dengan bentuk seperangkat persamaan simultan yang setara tersebut maka penyelesaian bagi x_i dapat diperoleh melalui proses substitusi balik.

Berikut merupakan beberapa langkah untuk implementasi penyelesaian seperangkat persamaan simultan bagi matrik tri-diagonal tersebut.

Untuk $a_1' = a_1; y_1' = y_1$ **dan** $i = 2, 3, \cdots, n$ **lakukan**

$$\begin{aligned} w &= \frac{c_{i-1}}{a_{i-1}'} \\ a_i' &= a_i - w b_{i-1} \\ y_i' &= y_i - w y_{i-1}' \end{aligned}$$

Dengan ungkapan tersebut maka penyelesaian seperangkat persamaan simultan berdasar proses substitusi balik berbentuk

$$x_n = \frac{y_n^{'}}{a_n^{'}} \tag{177}$$

$$x_i = \frac{y_i^{'} - b_i x_{i+1}}{a_i^{'}}; \qquad i = (n-1), (n-2), \ldots, 1 \tag{178}$$

Berdasar langkah di atas maka penyelesaian untuk masalah nilai eigen akan memberikan hasil trivial karena $x_i = 0$ untuk $i = 1, 2, \cdots, n$. Penyelesian tak trivial dapat diatasi dengan tidak mnenyertakan baris atau persamaan ke n sehingga x_n akan bernilai sebarang, dan untuk kemudahan proses komputasi maka sementara dapat dipilih nilai $x_n = 1$. Karena nilai x_i untuk $i = (n-1), (n-2), \cdots, 1$ merupakan kelipatan dari x_n maka nilai sebarang x_n nantinya dapat ditentukan berdasar syarat normalisir bahwa

$$\sum_{i=1}^{n} x_i^2 = \left(x_1^2 + x_2^2 + \cdots + 1^2\right) x_n^2 = 1 \quad \Longrightarrow \quad x_n = \frac{1}{\sqrt{x_1^2 + x_2^2 + \cdots + 1^2}} \tag{179}$$

Berdasar langkah yang dijelaskan di atas maka prosedur untuk memperoleh fungsi gelombang yang merupakan fungsi eigen adalah sebagai berikut:

1. Berikan nilai eigen yaitu ϵ yang terkait dengan fungsi eigen yang akan tentukan.
2. Untuk memulai proses komputasi, ambil $\psi_{N-1} = 1$
3. Dengan tidak menyertakan baris ke $(N-1)$ maka seperangkat $N-2$ persamaan simultan akan berbentuk

$$\begin{pmatrix} 2 + (\Delta r)^2\gamma_1 - (\Delta r)^2\epsilon & -1 & \cdots & 0 \\ -1 & 2 + (\Delta r)^2\gamma_2 - (\Delta r)^2\epsilon & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 2 + (\Delta r)^2\gamma_{N-2} - (\Delta r)^2\epsilon \end{pmatrix} \begin{pmatrix} \psi_1 \\ \psi_2 \\ \vdots \\ \psi_{N-2} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ \psi_{N-1} \end{pmatrix} \tag{180}$$

4. Setelah $\psi_1, \cdots, \psi_{N-2}$ diperoleh berdasar langkah untuk implementasi penyelesaian seperangkat persamaan simultan bagi matrik tri-diagonal di atas maka fungsi gelombang ternormalisir yang berpadanan dengan nilai tenaga ϵ diperoleh melalui kaitan

$$\psi_i = \frac{\psi_i}{\sqrt{\psi_1^2 + \psi_2^2 + \cdots + 1^2}}; \qquad i = 1, 2, \cdots, (N-1) \tag{181}$$

Sumur Potensial Tak Berhingga

Untuk sistem sumur potensial tak berhingga maka partikel tidak mampu menerobos dinding sumur sehingga fungsi gelombang hanya berada di dalam sumur. Untuk itu dapat dipilih syarat batas $\psi_0 = 0$ dan $\psi_N = 0$ masing-masing di $r_0 = -1$ (atau $x_0 = -a$) dan $r_N = 1$ (atau $x_N = a$). Karena di dalam sumur berlaku $V(x) = 0$ untuk $-a < x < a$ maka diperoleh paramater $\gamma_i = 0$ untuk $i = 1, 2, \cdots, N-1$. Dengan demikian, unsur-unsur matrik tri-diagonal berbentuk $a_i = 2 - (\Delta r)^2\epsilon$ untuk $i = 1, 2, \cdots, N-1$ sedangkan $b_i = -1$ dan $c_i = -1$ untuk $i = 1, 2, \cdots, N-2$.

Tingkat-tingkat tenaga bagi sistem akan dapat diperoleh berdasar kaitan

$$E = \frac{\hbar^2\epsilon}{2ma^2}; \qquad \text{untuk potensial} \qquad V(r) = \frac{\hbar^2\gamma(r)}{2ma^2} \tag{182}$$

Implementasi ungkapan di atas dalam bentuk *source-code* dapat dilihat seperti di bawah.

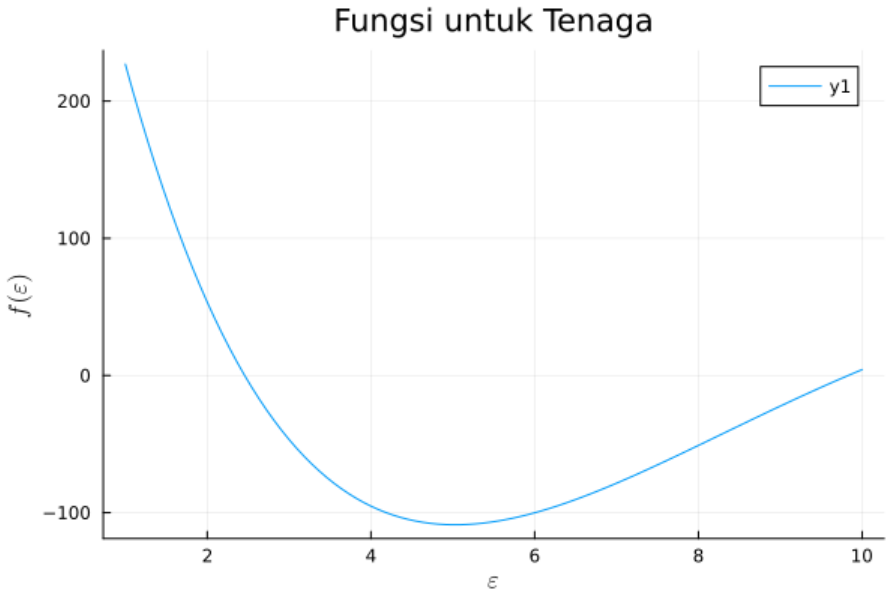
```
using LinearAlgebra
using Plots
using LaTeXStrings

# - - - - Functions - - - -
function fung(epsilon, N)
    dx = 2.0 / N
    d1 = 2.0 - epsilon * dx^2
    d2 = (2.0 - epsilon * dx^2)^2 - 1.0
    b = -1.0
    c = -1.0
    a = 2.0 - epsilon * dx^2
    dn = d2
    for i in 2:(N -1)
        dn = a * d2 - b * c * d1
        d1, d2 = d2, dn
    end
    return dn
end

function eigval_bisec(epsilon1, epsilon2, N)
    delta = 1e -4
    ralat = 0.1
    epsilonm = (epsilon1 + epsilon2) / 2.0
    while ralat > delta
        epsilonm = (epsilon1 + epsilon2) / 2.0
        f1f2 = fung(epsilon1, N) * fung(epsilonm, N)
        if f1f2 < 0.0
            epsilon2 = epsilonm
        else
            epsilon1 = epsilonm
        end
        ralat = abs((epsilon2 - epsilon1) / epsilon2)
    end
    return epsilonm
end

function eigvec_tridiag(epsilon, N)
    dx = 2.0 / N
    a = 2.0 - epsilon * dx^2
    b = -1.0
    c = -1.0

    d = fill(a, N -2)              # main diagonal
    e = fill( -1.0, N -3)           # subdiagonal (also superdiagonal)
    f = zeros(N -2)                # RHS
    f[end] = 1.0

    # Forward elimination (Thomas algorithm for tridiagonal)
    for i in 2:(N -2)              # Python: range(2, N -1)
        m = e[i -1] / d[i -1]
        d[i] -= m * b
        f[i] -= m * f[i -1]
```

```
    end

    # Back substitution
    v = zeros(N -1)
    v[N -2] = f[N -2] / d[N -2]
    v[N -1] = 1.0
    s = v[N -1]^2 + v[N -2]^2
    for i in (N -3): -1:1           # Python: range(N -4, -1, -1)
        v[i] = (f[i] - b * v[i+1]) / d[i]
        s += v[i]^2
    end

    v ./= sqrt(s)                  # normalize
    return v
end
```

```
eigvec_tridiag (generic function with 1 method)
```

```
# - - - - Parameters & evaluation - - - -
N = 500
m = 100
epsilon_vals = range(1.0, 10.0, length=m)
fe = [fung( \epsilon, N) for \epsilon in epsilon_vals];
```

```
plot(epsilon_vals, fe;
     xlabel=L"\epsilon",
     ylabel=L"f(\epsilon)",
     title="Fungsi untuk Tenaga",
     grid=true, gridstyle=:solid)
```

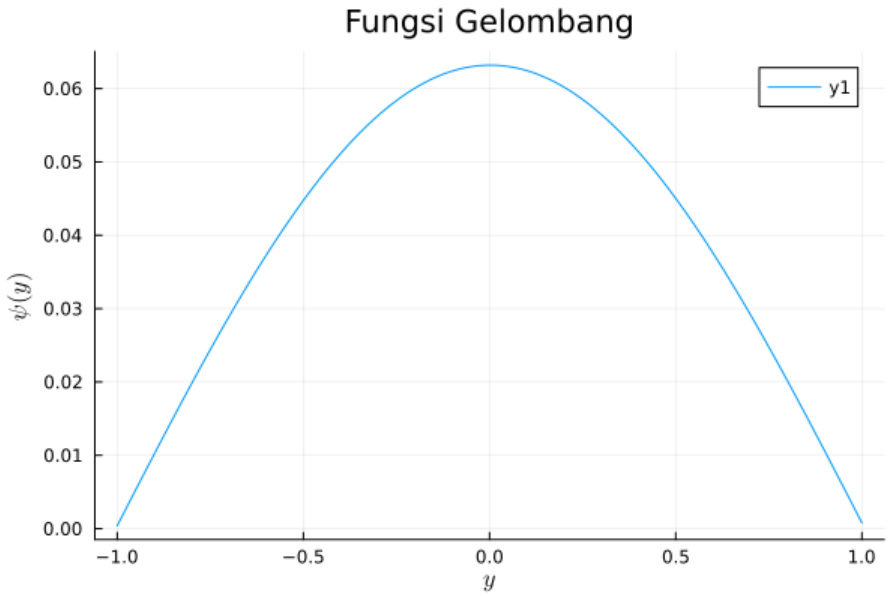



```
# - - - - Root via bisection - - - -
epsilon_root = eigval_bisec(2.0, 4.0, N)
println(epsilon_root)
```

```
2.457763671875
```

```
# - - - - Eigenvector and plot - - - -
y = range( -1.0, 1.0, length=N -1)
v = eigvec_tridiag(epsilon_root, N);
```

```
plot(y, v;
     xlabel=L"y",
     ylabel=L"\psi(y)",
     title="Fungsi Gelombang",
     grid=true, gridstyle=:solid)
```



Sumur Potensial Berhingga

Untuk sistem sumur potensial berhingga maka partikel mampu menerobos dinding sumur sehingga fungsi gelombang tidak hanya berada di dalam sumur. Untuk itu perlu dijamin agar syarat batas $\psi_0 = 0$ dan $\psi_N = 0$ dapat terpenuhi pada posisi r_0 dan r_N. Karena kedua posisi batas tersebut belum diketahui, maka dapat dicoba pada berbagai posisi yang jauh dari dinding sumur. Berdasarkan intuisi maka dapat diperkirakan bahwa terobosan kuantum akan semakin terasa ketika tenaga sistem besar. Sebagai contoh, untuk tenaga dasar (*ground-state*) maka dapat dicoba pada posisi $x_0 \approx -1.2$ dan $x_N \approx 1.2$ dan kemudian untuk tenaga eksitasi pertama maka dapat dicoba pada posisi $x_0 \approx -1.6$ dan $x_N \approx 1.6$.

Berbeda dengan sistem sumur potensial tak berhingga yang memiliki nilai $\gamma_i = 0$ untuk $i = 1, 2, \cdots, N$, untuk sistem sumur potensial berhingga berlaku

$$\begin{aligned} \gamma(r) &= \frac{2ma^2V_0}{\hbar^2} = \gamma; & -\infty < r < -1 \\ &= 0; & -1 < r < 1 \\ &= \frac{2ma^2V_0}{\hbar^2} = \gamma; & 1 < r < \infty \end{aligned} \tag{183}$$

Dengan demikian, unsur-unsur matrik tri-diagonal berbentuk $a_i = 2 + (\Delta r)^2\gamma_i - (\Delta r)^2\epsilon$ untuk $i = 1, 2, \cdots, N-1$ sedangkan $b_i = -1$ dan $c_i = -1$ untuk $i = 1, 2, \cdots, N-2$.

Tingkat-tingkat tenaga bagi sistem akan dapat diperoleh berdasar kaitan

$$E = \frac{\hbar^2\epsilon}{2ma^2}; \qquad \text{untuk potensial} \qquad V(r) = \frac{\hbar^2\gamma(r)}{2ma^2} \tag{184}$$

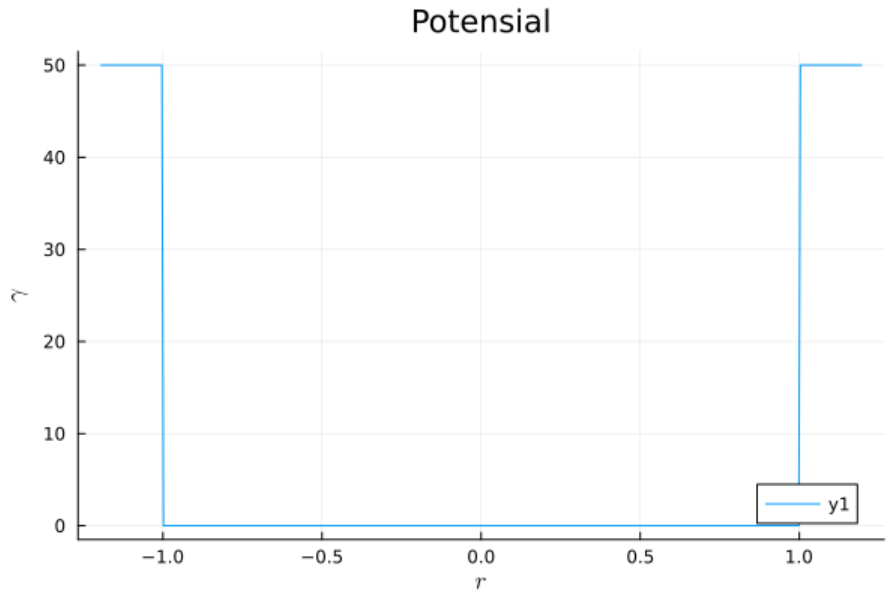
Implementasi ungkapan di atas dalam bentuk *source-code* dapat dilihat seperti di bawah.

```
function observ(v0, rmak, rmin, dr, N)
    r = zeros(N -1)
    gamma = zeros(N -1)
```

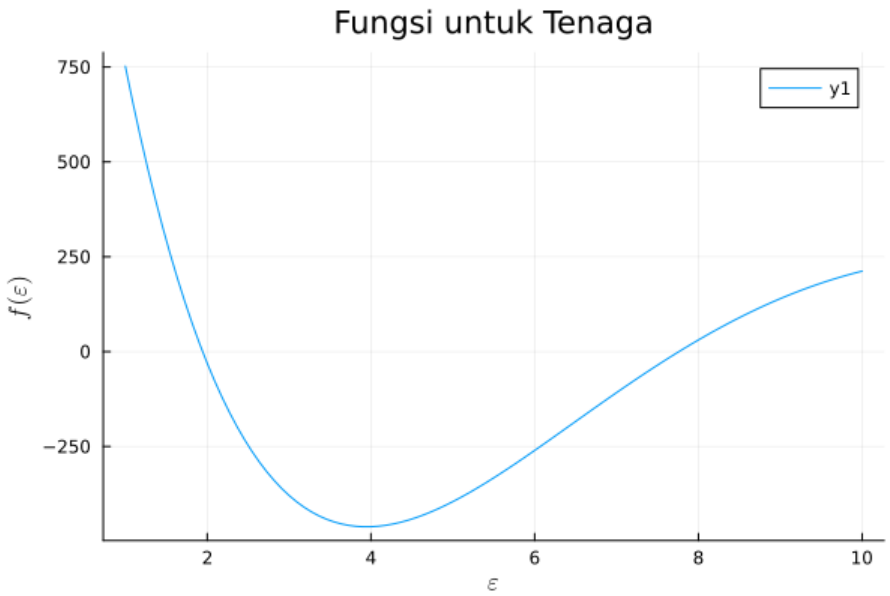
```
    for i in 1:(N -1)
        r[i] = dr * i + rmin
        if r[i] <= -1.0 || r[i] >= 1.0
            gamma[i] = v0
        else
            gamma[i] = 0.0
        end
    end
    return r, gamma
end

observ (generic function with 1 method)

function fung_gamma(epsilon, dr, gamma, N)
    a = zeros(N -1)
    b = zeros(N -2)
    c = zeros(N -2)
    for i in 1:(N -1)
        a[i] = 2.0 + gamma[i] * dr^2 - epsilon * dr^2
    end
    for i in 1:(N -2)
        b[i] = -1.0
        c[i] = -1.0
    end
    d1 = a[1]
    d2 = a[2] * d1 - b[1] * c[1]
    dn = 0.0
    for i in 3:(N -1)
        dn = a[i] * d2 - b[i -1] * c[i -1] * d1
        d1, d2 = d2, dn
    end
    return dn
end

fung_gamma (generic function with 1 method)

function eigval_bisec(epsilon1, epsilon2, dr, gamma, N)
    delta = 1e -4
    ralat = 0.1
    epsilonm = (epsilon1 + epsilon2) / 2.0
    while ralat > delta
        epsilonm = (epsilon1 + epsilon2) / 2.0
        f1f2 = fung_gamma(epsilon1, dr, gamma, N) * fung_gamma(epsilonm, dr, gamma, N)
        if f1f2 < 0.0
            epsilon2 = epsilonm
        else
            epsilon1 = epsilonm
        end
        ralat = abs((epsilon2 - epsilon1) / epsilon2)
    end
    return epsilonm
end

eigval_bisec (generic function with 2 methods)
```

```
function eigvec_tridiag(epsilon, dr, gamma, N)
    d = zeros(N -2)
    e = zeros(N -3)
    b = zeros(N -3)
    for i in 1:(N -2)
        d[i] = 2.0 + gamma[i] * dr^2 - epsilon * dr^2
    end
    for i in 1:(N -3)
        e[i] = -1.0
        b[i] = -1.0
    end
    f = zeros(N -2)
    f[end] = 1.0

    # Forward elimination (Thomas algorithm for tridiagonal)
    for i in 2:(N -2)
        m = e[i -1] / d[i -1]
        d[i] -= m * b[i -1]
        f[i] -= m * f[i -1]
    end

    # Back substitution
    v = zeros(N -1)
    v[N -2] = f[N -2] / d[N -2]
    v[N -1] = 1.0
    s = v[N -1]^2 + v[N -2]^2
    for i in (N -3): -1:1
        v[i] = (f[i] - b[i] * v[i+1]) / d[i]
        s += v[i]^2
    end

    v ./= sqrt(s)                 # normalize
    return v
end

eigvec_tridiag (generic function with 2 methods)

# - - - - Parameters & evaluation - - - -
N = 500
m = 100
rmak = 1.2
rmin = -rmak
dr = (rmak - rmin) / N
v0 = 50.0
r = zeros(N -1)
gamma = zeros(N -1)
r, gamma = observ(v0, rmak, rmin, dr, N)
epsilon = range(1.0, 10.0, length=m)
fe = zeros(m)
for i in 1:m
    fe[i] = fung_gamma(epsilon[i], dr, gamma, N)
end

plot(r, gamma;
```

```
      xlabel=L"r",
      ylabel=L"\gamma",
      title="Potensial",
      grid=true, gridstyle=:solid)
```




```
plot(epsilon, fe;
      xlabel=L"\epsilon",
      ylabel=L"f(\epsilon)",
      title="Fungsi untuk Tenaga",
      grid=true, gridstyle=:solid)
```

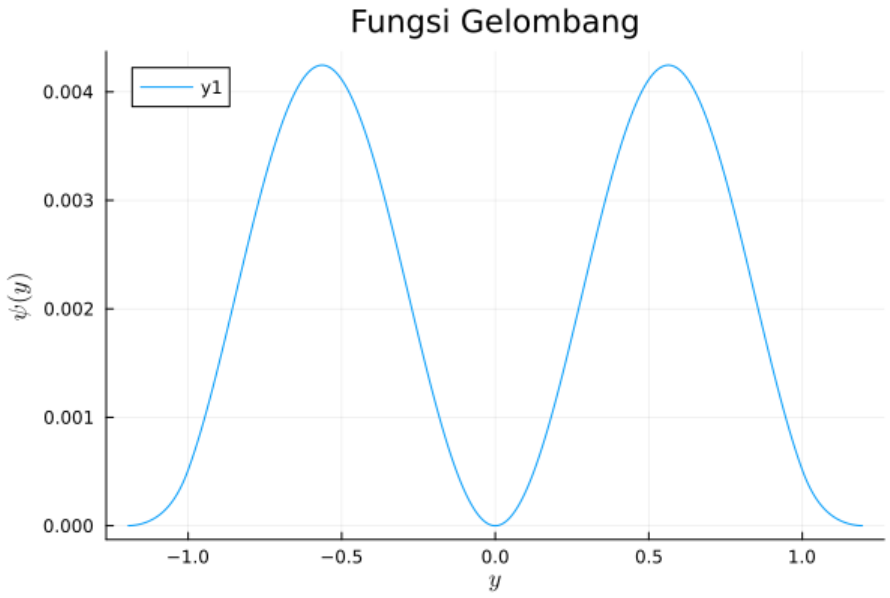



```
# - - - - Root via bisection - - - -
epsilon = eigval_bisec(7.0, 8.0, dr, gamma, N)
println(epsilon)
```

```
7.76123046875
```

```
# - - - - Eigenvector and plot - - - -
v = eigvec_tridiag(epsilon, dr, gamma, N);
vabs = v .^ 2;
```

```
plot(r, vabs;
     xlabel=L"y",
     ylabel=L"\psi(y)",
     title="Fungsi Gelombang",
     grid=true, gridstyle=:solid)
```



Sumur Bertanggul Potensial

Mirip seperti sistem sumur potensial berhingga di atas maka untuk sistem dengan potensial berbentuk sumur bertanggul maka partikel mampu menerobos dinding sumur sehingga fungsi gelombang tidak hanya berada di dalam sumur. Sebagai tambahan maka partikel di dalam sumur juga berpotensi mengalami terobosan kuantum saat mengenai tanggul potensial.

Dengan melakukan modifikasi pada sistem sumur potensial berhingga, untuk sistem dengan potensial berbentuk sumur bertanggul berlaku

$$\begin{aligned} \gamma(r) &= \frac{2ma^2V_0}{\hbar^2} = \gamma; && -\infty < r < 0 \\ &= 0; && 0 < r < \rho \\ &= \frac{2ma^2V_1}{\hbar^2} = \beta; && \rho < r < 1 \\ &= \frac{2ma^2V_0}{\hbar^2} = \gamma; && 1 < r < \infty \end{aligned} \tag{185}$$

Dalam ungkapan digunakan satuan universal seperti berikut

$$r = \frac{x}{a}; \qquad \rho = \frac{b}{a}; \qquad \epsilon = \frac{2ma^2E}{\hbar^2}; \qquad \gamma = \frac{2ma^2V_0}{\hbar^2}; \qquad \beta = \frac{2ma^2V_1}{\hbar^2} \tag{186}$$

Dengan demikian, unsur-unsur matrik tri-diagonal berbentuk $a_i = 2 + (\Delta r)^2\gamma_i - (\Delta r)^2\epsilon$ untuk $i = 1, 2, \cdots, N-1$ sedangkan $b_i = -1$ dan $c_i = -1$ untuk $i = 1, 2, \cdots, N-2$.

Tingkat-tingkat tenaga bagi sistem akan dapat diperoleh berdasar kaitan

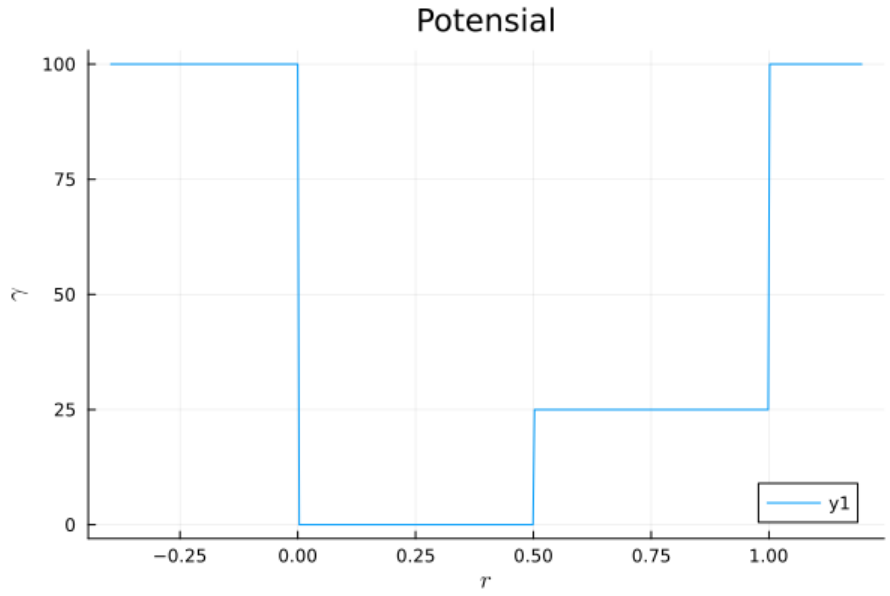
$$E = \frac{\hbar^2\epsilon}{2ma^2}; \qquad \text{untuk potensial} \qquad V(r) = \frac{\hbar^2\gamma(r)}{2ma^2} \tag{187}$$

Implementasi ungkapan di atas dalam bentuk *source-code* dapat dilihat seperti di bawah.

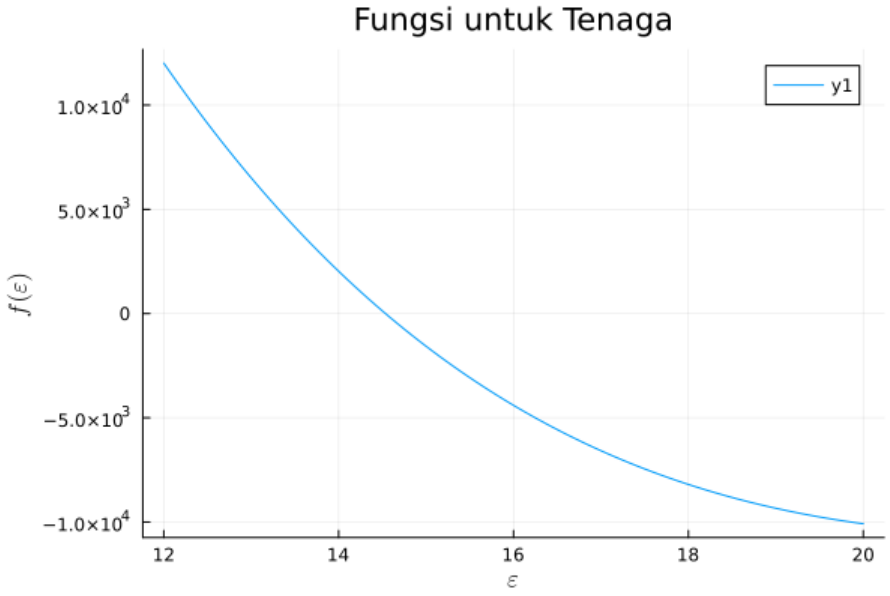
```
function observ1(v0, v1, rmak, rmin, dr, rho, N)
    r = zeros(N -1)
    gamma = zeros(N -1)
    beta = zeros(N -1)
    for i in 1:(N -1)
        r[i] = dr * i + rmin
        if r[i] <= 0.0 || r[i] >= 1.0
            gamma[i] = v0
        elseif r[i] <= rho
            gamma[i] = 0.0
        else
            gamma[i] = v1
        end
    end
    return r, gamma
end

observ1 (generic function with 1 method)

function fung_gamma1(epsilon, dr, gamma, N)
    a = zeros(N -1)
    b = zeros(N -2)
    c = zeros(N -2)
    for i in 1:(N -1)
        a[i] = 2.0 + gamma[i] * dr^2 - epsilon * dr^2
    end
    for i in 1:(N -2)
        b[i] = -1.0
        c[i] = -1.0
    end
    d1 = a[1]
    d2 = a[2] * d1 - b[1] * c[1]
    dn = 0.0
    for i in 3:(N -1)
        dn = a[i] * d2 - b[i -1] * c[i -1] * d1
        d1, d2 = d2, dn
    end
    return dn
end

fung_gamma1 (generic function with 1 method)

function eigval_bisec1(epsilon1, epsilon2, dr, gamma, N)
    delta = 1e -4
    ralat = 0.1
    epsilonm = (epsilon1 + epsilon2) / 2.0
    while ralat > delta
        epsilonm = (epsilon1 + epsilon2) / 2.0
        f1f2 = fung_gamma1(epsilon1, dr, gamma, N) * fung_gamma1(epsilonm, dr, gamma, N)
        if f1f2 < 0.0
            epsilon2 = epsilonm
        else
            epsilon1 = epsilonm
        end
```

```
        ralat = abs((epsilon2 - epsilon1) / epsilon2)
    end
    return epsilonm
end

eigval_bisec1 (generic function with 1 method)

function eigvec_tridiag1(epsilon, dr, gamma, N)
    d = zeros(N -2)
    e = zeros(N -3)
    b = zeros(N -3)
    for i in 1:(N -2)
        d[i] = 2.0 + gamma[i] * dr^2 - epsilon * dr^2
    end
    for i in 1:(N -3)
        e[i] = -1.0
        b[i] = -1.0
    end
    f = zeros(N -2)
    f[end] = 1.0

    # Forward elimination (Thomas algorithm for tridiagonal)
    for i in 2:(N -2)
        m = e[i -1] / d[i -1]
        d[i] -= m * b[i -1]
        f[i] -= m * f[i -1]
    end

    # Back substitution
    v = zeros(N -1)
    v[N -2] = f[N -2] / d[N -2]
    v[N -1] = 1.0
    s = v[N -1]^2 + v[N -2]^2
    for i in (N -3): -1:1
        v[i] = (f[i] - b[i] * v[i+1]) / d[i]
        s += v[i]^2
    end

    v ./= sqrt(s)                 # normalize
    return v
end

eigvec_tridiag1 (generic function with 1 method)

# - - - - Parameters & evaluation - - - -
N = 500
m = 100
rmak = 1.2
rmin = -0.4
dr = (rmak - rmin) / N
v0 = 100.0
v1 = 25.0
rho = 0.5
r = zeros(N -1)
```

```
gamma = zeros(N -1)
r, gamma = observ1(v0, v1, rmak, rmin, dr, rho, N)
epsilon = range(12.0, 20.0, length=m)
fe = zeros(m)
for i in 1:m
    fe[i] = fung_gamma1(epsilon[i], dr, gamma, N)
end

plot(r, gamma;
     xlabel=L"r",
     ylabel=L"\gamma",
     title="Potensial",
     grid=true, gridstyle=:solid)
```



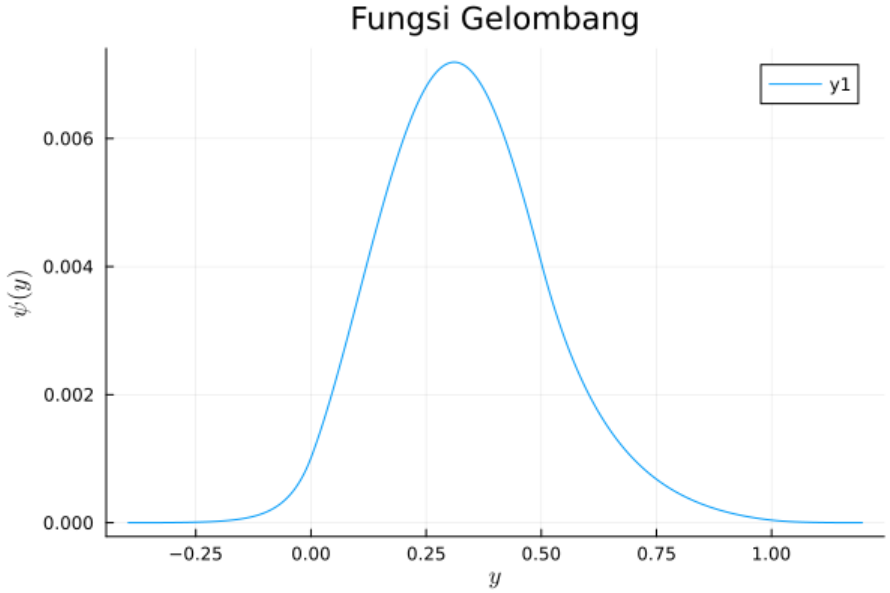

```
plot(epsilon, fe;
     xlabel=L"\epsilon",
     ylabel=L"f(\epsilon)",
     title="Fungsi untuk Tenaga",
     grid=true, gridstyle=:solid)
```



```
# - - - - Root via bisection - - - -
epsilon = eigval_bisec1(14.0, 15.0, dr, gamma, N)
println(epsilon)
```

```
14.5361328125
```

```
# - - - - Eigenvector and plot - - - -
v = eigvec_tridiag1(epsilon, dr, gamma, N);
vabs = v .^ 2;

plot(r, vabs;
     xlabel=L"y",
     ylabel=L"\psi(y)",
     title="Fungsi Gelombang",
     grid=true, gridstyle=:solid)
```



Pencarian nilai *eigen* dan fungsi *eigen* untuk sebarang bentuk matrik

Apabila masalah nilai *eigen* bukan berbentuk matrik tri-diagonal maka cara komputasi yang cukup efektif di atas menjadi tidak dimungkinkan. Untuk keadaan ini maka pencarian nilai *eigen* dan fungsi *eigen* perlu diselesaikan dengan metode lain yang berlaku untuk matrik dengan bentuk sebarang, salah satu diantaranya adalah dengan implementasi metode dekomposisi LU. Istilah LU merupakan singkatan dari (*Lower Triangular Matrix*) untuk L dan (*Upper Triangular Matrix*) untuk U. Seperti yang tersirat dari nama tersebut maka prinsip dasar dari metode pemisahan LU adalah mengubah sebarang matrik A menjadi produk perkalian matrik segitiga bawah L dan matrik segitiga atas U yaitu $A = LU$ sehingga dapat memanfaatkan proses substitusi maju bagi matrik L dan substitusi balik bagi matrik U ketika nantinya dimanfaatkan untuk mendapatkan penyelesaian x.

$$A = LU \tag{188}$$

Apabila bentuk matrik U diambil sama dengan matrik segitiga atas $A^{'}$ yang diperoleh dari proses eliminasi Gauss maka berlaku ungkapan berikut.

$$U = \begin{pmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1N} \\ 0 & u_{22} & u_{22} & \cdots & u_{2N} \\ 0 & 0 & u_{33} & \cdots & u_{3N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_{NN} \end{pmatrix} = \begin{pmatrix} a_{11}^{'} & a_{12}^{'} & a_{13}^{'} & \cdots & a_{1N}^{'} \\ 0 & a_{22}^{'} & a_{23}^{'} & \cdots & a_{2N}^{'} \\ 0 & 0 & a_{33}^{'} & \cdots & a_{3N}^{'} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{NN}^{'} \end{pmatrix} = A^{'} \tag{189}$$

Dengan bentuk matrik segitiga atas U tersebut maka bentuk matrik segitiga bawah dapat ditentukan berdasar operasi berikut.

$$LU = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ l_{21} & 1 & 0 & \cdots & 0 \\ l_{31} & l_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{N1} & l_{N2} & l_{N3} & \cdots & 1 \end{pmatrix} \begin{pmatrix} a_{11}^{'} & a_{12}^{'} & a_{13}^{'} & \cdots & a_{1N}^{'} \\ 0 & a_{22}^{'} & a_{23}^{'} & \cdots & a_{2N}^{'} \\ 0 & 0 & a_{33}^{'} & \cdots & a_{3N}^{'} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{NN}^{'} \end{pmatrix}$$

$$= \begin{pmatrix} a_{11}^{'} & a_{12}^{'} & a_{13}^{'} & \cdots & a_{1N}^{'} \\ l_{21}a_{11}^{'} & a_{22}^{'} & a_{23}^{'} & \cdots & a_{2N}^{'} \\ l_{31}a_{11}^{'} & l_{32}a_{22}^{'} & a_{33}^{'} & \cdots & a_{3N}^{'} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{N1}a_{11}^{'} & l_{N2}a_{22}^{'} & l_{N3}a_{33}^{'} & \cdots & a_{NN}^{'} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1N} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2N} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & a_{N3} & \cdots & a_{NN} \end{pmatrix} = A \tag{190}$$

Berdasar ungkapan tersebut maka bentuk matrik segitiga bawah L adalah seperti berikut.

$$L = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ l_{21} & 1 & 0 & \cdots & 0 \\ l_{31} & l_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{N1} & l_{N2} & l_{N3} & \cdots & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ \frac{a_{21}}{a_{11}^{'}} & 1 & 0 & \cdots & 0 \\ \frac{a_{31}}{a_{11}^{'}} & \frac{a_{32}}{a_{22}^{'}} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{a_{N1}}{a_{11}^{'}} & \frac{a_{N2}}{a_{22}^{'}} & \frac{a_{N3}}{a_{33}^{'}} & \cdots & 1 \end{pmatrix} \tag{191}$$

Dari bentuk matrik segitiga bawah L tersebut maka menjadi jelas bahwa unsur matrik bagi L tidak lain merupakan faktor pengali dalam metode eliminasi Gauss dalam proses pengubahan ke nilai nol pada kolom ke k dan baris ke i yaitu $l_{ik} = \frac{a_{ik}}{a_{kk}^{'}} = \frac{U_{ik}}{U_{kk}}$.

Berikut merupakan *source-code* bagi implementasi metode pemisahan LU.

```
function lu_decomp(A)
    N = size(A, 1)
    L = Matrix{Float64}(I, N, N)
    U = copy(A)
    for k in 1:N -1
        for i in k+1:N
            pengali = U[i, k]/U[k, k]
            L[i, k] = pengali
            U[i, k:N] -= pengali*U[k, k:N];
        end
    end

    return L, U
end

lu_decomp (generic function with 1 method)

function deter(A)
    determinan = 0.0
    N = size(A, 1)
    L, U = lu_decomp(A)
    sum = 1.0
    for i in 1:N
        sum *= U[i,i]
    end
```

```
    determinan = sum
    return determinan
end

deter (generic function with 1 method)

function back_subs(U::Matrix{Float64},w::Vector{Float64})
    N = size(U,1)
    x = zeros(Float64,N)
    x[N] = w[N]/U[N,N]
    for i in N -1: -1:1
        sum = 0
        for j in i+1:N
            sum += U[i, j]*x[j]
        end
        x[i] = (w[i] -sum)/U[i, i]
    end

    return x
end

back_subs (generic function with 1 method)
```

Pencarian nilai *eigen* untuk sebarang bentuk matrik Ditinjau suatu masalah nilai eigen yang dapat dinyatakan dengan bentuk ungkapan berikut.

$$Ax = \lambda x \tag{192}$$

Seperti sudah dijelaskan dalam uraian untuk matrik tri-diagonal sebelumnya, dalam persamaan nilai eigen tersebut maka A merupakan matrik bujursangkar berorde $N \times N$ yang sudah diketahui. Matrik kolom x merupakan matrik yang akan diselesaikan dan biasa disebut sebagai fungsi eigen. Adapun λ adalah suatu nilai yang juga akan diselesaikan dan biasa disebut sebagai nilai eigen.

Persamaan nilai eigen dapat diubah ke ungkapan persamaan simultan berupa persamaan homogen, yaitu ruas kanan persamaan adalah matrik O dalam bentuk berikut.

$$(A - \lambda I)x = O \implies x = (A - \lambda I)^{-1}0 = \frac{1}{\det{(A - \lambda I)}}\text{adj}(A - \lambda I)\, O \tag{193}$$

Dalam ungkapan di atas, matrik I melambangkan matrik identitas yang memiliki norde sama dengan matrik A sedangkan matrik O merupakan matrik kolom dengan semua unsur bernilai 0.

Upaya untuk menyelesaikan bentuk persamaan homogen tersebut dengan beberapa metode penyelesaian persamaan simultan yang dijelaskan dalam uraian sebelumnya nampak akan gagal karena akan memberikan penyelesaian trivial berupa fungsi eigen $x = O$. Untuk mendapatkan penyelesaian tak trivial sedemikian hingga fungsi eigen $x \neq O$, beserta nilai eigen λ yang terkait dengan fungsi eigen tersebut, maka dapat digunakan mekanisme yang disebut proses diagonalisasi matrik.

Berdasar pada ungkapan persamaan homogen di atas maka proses diagonalisasi matrik pada dasarnya merupakan suatu upaya agar $x \neq O$ atau $x \propto \frac{O}{0}$, yang dapat terpenuhi apabila dipersyaratkan bentuk berikut.

$$\det(A - \lambda I) = \begin{vmatrix} a_{11} - \lambda & a_{12} & a_{13} & \cdots & a_{1N} \\ a_{21} & a_{22} - \lambda & a_{23} & \cdots & a_{2N} \\ a_{31} & a_{32} & a_{33} - \lambda & \cdots & a_{3N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & a_{N3} & \cdots & a_{NN} - \lambda \end{vmatrix} = 0 \tag{194}$$

Ungkapan di atas tidak lain merupakan bentuk masalah pencarian akar atau titik nol suatu fungsi $f(\lambda) = \det(A - \lambda I)$, yaitu nilai eigen λ merupakan akar atau titik nol fungsi tersebut. Dengan demikian nilai eigen akan dapat diperoleh menggunakan metode pencarian akar sebagai contoh metode Bisection atau metode Newton-Raphson.

Andaikan bentuk matrik di atas berupa matrik diagonal atau matrik segitiga dengan bentuk berikut:

$$f(\lambda) = \det(A - \lambda I) = \begin{vmatrix} a_{11} - \lambda & a_{12} & a_{13} & \cdots & a_{1N} \\ 0 & a_{22} - \lambda & a_{23} & \cdots & a_{2N} \\ 0 & 0 & a_{33} - \lambda & \cdots & a_{3N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{NN} - \lambda \end{vmatrix} = 0 \tag{195}$$

Dengan bentuk tersebut maka penyelesaian akan mudah diperoleh yaitu:

$$f(\lambda) = \det(A - \lambda I) = (a_{11} - \lambda)(a_{22} - \lambda) \cdots (a_{NN} - \lambda) = 0 \tag{196}$$

Dengan demikian penyelesaian nilai eigen λ diperoleh sebagai berikut:

$$\lambda_1 = a_{11}; \quad \lambda_2 = a_{22}; \quad \cdots \quad \lambda_N = a_{NN} \tag{197}$$

Istilah proses diagonalisasi matrik untuk memperoleh penyelesaian tak trivial sedemikian hingga fungsi eigen $x \neq O$, beserta nilai eigen λ yang terkait dengan fungsi eigen tersebut, menjadi mudah dipahami dari ungkapan matrik di atas.

Secara umum untuk matrik tidak diagonal A maka proses untuk memperoleh nilai eigen λ dapat dilakukan dengan tahap berikut:

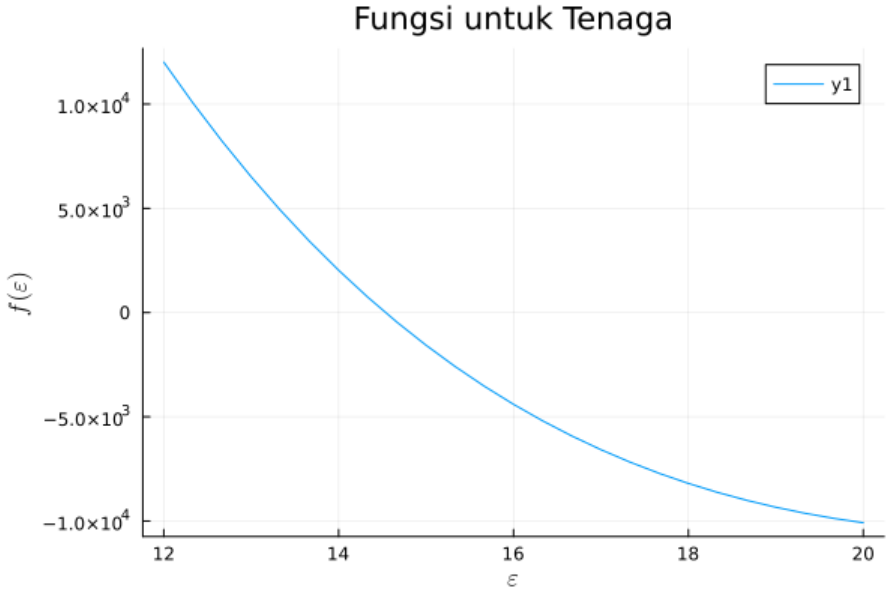
1. Berikan nilai coba bagi nilai eigen λ sebagai masukan bagi nilai coba akar.
2. Lakukan proses diagonalisasi matrik dengan metode dekomposisi LU, yaitu $(A - \lambda I) = LU$ untuk memperoleh fungsi $f(\lambda) = \det(A - \lambda I)$.
3. Tentukan nilai eigen melalui proses pencarian akar bagi ungkapan $f(\lambda) = \det(A - \lambda I) = 0$ menggunakan metode Bisection.

Berikut merupakan *source-code* bagi implementasi percarian nilai eigen λ berdasar tahapan di atas.

```
function matrix_A(dr, gamma, N)
    a = zeros(N -1)
    b = zeros(N -2)
    c = zeros(N -2)
    A = zeros(N -1, N -1)
    for i in 1:(N -1)
        a[i] = 2.0 + gamma[i] * dr^2
    end
    for i in 1:(N -2)
        b[i] = -1.0
        c[i] = -1.0
    end
    A = diagm(0 => a, -1 => b, 1 => c)
    return A
end

matrix_A (generic function with 1 method)
```

```
function fung_lambda(A, dr, epsilon)
    N = size(A, 1)
    A_lambda = A . - epsilon * dr^2 * I(N)
    hasil = 0.0
    hasil = deter(A_lambda)

    return hasil
end

fung_lambda (generic function with 1 method)

function nil_eigen(A, epsilon1, epsilon2)
    N = size(A, 1)
    U = zeros(N,N)
    a = epsilon1
    b = epsilon2
    c = 0.0
    iter = 0
    while (b - a)/2 > 1.0e -6 && iter < 100
        iter += 1
        c = (a + b)/2
        f1 = fung_lambda(A, dr, a)
        f2 = fung_lambda(A, dr, c)
        if f1 * f2 < 0
            b = c
        else
            a = c
        end
    end

    return c
end

nil_eigen (generic function with 1 method)

# - - - - Parameters & evaluation - - - -
N = 500
m = 25
rmak = 1.2
rmin = -0.4
dr = (rmak - rmin) / N
v0 = 100.0
v1 = 25.0
rho = 0.5
r = zeros(N -1)
gamma = zeros(N -1);

r, gamma = observ1(v0, v1, rmak, rmin, dr, rho, N)
A = matrix_A(dr, gamma, N)
epsilon = range(12.0, 20.0, length=m)
fe = zeros(m)
for i in 1:m
    fe[i] = fung_lambda(A, dr, epsilon[i])
end
```

```
plot(epsilon, fe;
     xlabel=L"\epsilon",
     ylabel=L"f(\epsilon)",
     title="Fungsi untuk Tenaga",
     grid=true, gridstyle=:solid)
```




```
epsilon1 = 12.0
epsilon2 = 15.0
epsilon_new = nil_eigen(A, epsilon1, epsilon2)
```

```
14.535302639007568
```

Pencarian fungsi *eigen* untuk sebarang bentuk matrik Ketika nilai eigen λ tertentu telah diperoleh berdasar prosedur pencarian nilai eigen di atas maka fungsi eigen yang berpadanan dengan nilai eigen tertentu tersebut akan dapat diperoleh dengan cara memasukkan kembali nilai eigen λ ke persamaan nilai eigen semula. Pada satu nilai eigen λ tertentu ini, penggunaan metode dekomposisi LU dalam proses diagonalisasi matrik $\det(A-\lambda I)=0$ menyebabkan berlakunya kaitan berikut.

$$\det(A-\lambda I)=\begin{vmatrix} a_{11}-\lambda & a_{12} & a_{13} & \cdots & a_{1N} \\ 0 & a_{22}-\lambda & a_{23} & \cdots & a_{2N} \\ 0 & 0 & a_{33}-\lambda & \cdots & a_{3N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{NN}-\lambda \end{vmatrix} = \begin{vmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1N} \\ 0 & u_{22} & u_{23} & \cdots & u_{2N} \\ 0 & 0 & u_{33} & \cdots & u_{3N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_{NN} \end{vmatrix} = u_{11}u_{22}\ldots u_{NN}=0 \tag{198}$$

Pada satu nilai eigen λ tertentu ini maka persamaan nilai eigen memenuhi ungkapan berikut.

$$(A-\lambda I)x = LUx = O \implies Ux = O \tag{199}$$

Dalam bentuk eksplisit, ungkapan tersebut dapat disajikan seperti berikut.

$$\begin{pmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1N} \\ 0 & u_{22} & u_{23} & \cdots & u_{2N} \\ 0 & 0 & u_{33} & \cdots & u_{3N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_{NN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \implies \begin{aligned} u_{11}x_1+u_{12}x_2+u_{13}x_3+\cdots+u_{1N}x_N &= 0 \\ u_{22}x_2+u_{23}x_3+\cdots+u_{2N}x_N &= 0 \\ u_{33}x_3+\cdots+u_{3N}x_N &= 0 \\ \vdots &= \vdots \\ u_{NN}x_N &= 0 \end{aligned} \tag{200}$$

Berdasar ungkapan pada persamaan baris ke N di atas, nampak bahwa nilai $x_N = 0$ dan akibatnya ketika dilakukan proses substitusi balik maka diperoleh fungsi eigen adalah nol yaitu $x_i = 0$ untuk $i = (N-1), \cdots, 1$. Penyelesaian seperti ini disebut penyelesaian trivial, yang tidak memberikan informasi apapun bagi sistem.

Untuk memperoleh penyelesaian tak trivial, yaitu fungsi eigen yang tidak nol, maka baris ke N dapat dihilangkan sehingga tersisa sejumlah $N-1$ persamaan simultan dalam bentuk berikut

$$\begin{aligned} u_{11}x_1 + u_{12}x_2 + u_{13}x_3 + \cdots + u_{1(N-1)}x_{N-1} &= -u_{1N}x_N \\ u_{22}x_2 + u_{23}x_3 + \cdots + u_{2(N-1)}x_{N-1} &= -u_{2N}x_N \\ u_{33}x_3 + \cdots + u_{3(N-1)}x_{N-1} &= -u_{3N}x_N \\ \vdots &= \vdots \\ u_{(N-1)(N-1)}x_{N-1} &= -u_{(N-1)N}x_N \end{aligned} \tag{201}$$

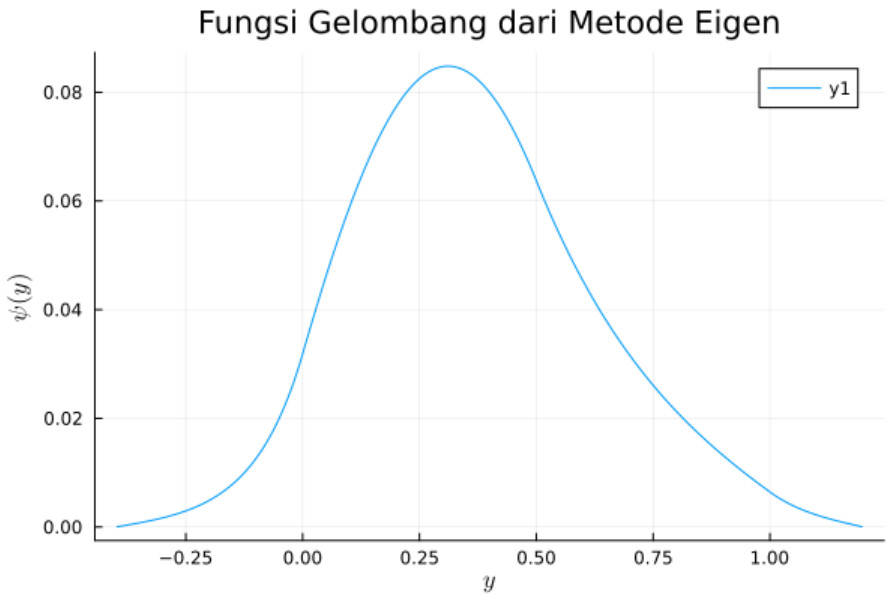
Karena baris ke N tidak diperhitungkan akibatnya nilai x_N menjadi tidak tentu atau bebas. Apabila untuk sementara nilai bebas bagi x_N ditentukan yaitu diambil $x_N = 1$ maka seperangkat $(N-1)$ persamaan simultan di atas akan dapat diperoleh berdasar proses substitusi balik sehingga diperoleh nilai untuk x_i dengan $i = (N-1), \cdots, 1$.

Pada akhirnya fungsi eigen dapat diperoleh yaitu dengan mempersyaratkan bahwa fungsi eigen perlu ternormalisir oleh proses normalisasi seperti berikut.

$$x_{\text{norm}} = Cx; \quad \text{dengan} \quad C = \frac{1}{\sqrt{x_1^2 + x_2^2 + \cdots + x_N^2}}; \quad x_N = 1 \tag{202}$$

Berikut merupakan *source-code* bagi implementasi percarian fungsi eigen ternormalisir x_{norm} berdasar uraian di atas.

```
function fung_eigen(A,dr,epsilon)
    N = size(A, 1)
    A_lambda = A . - epsilon * dr^2 * I(N)
    L, U = lu_decomp(A_lambda)
    x_norm = zeros(Float64,N)
    U_new = U[1:(N -1),1:(N -1)] #Mengambil submatrik dari U dengan menghilangkan baris dan ko
    b_new = -1.0*U[1:(N -1),N]
    x_new = back_subs(U_new,b_new)
    sum = 1.0
    for i in 1:(N -1)
        sum += x_new[i]^2
    end
    C = 1.0/sqrt(sum)
    x_norm = C * push!(x_new,1.0) #Menambahkan unsur x[N]=1.0 ke matrik kolom x yang sebelumny

    return x_norm
end

fung_eigen (generic function with 1 method)

feigen = fung_eigen(A, dr, epsilon_new);

plot(r, feigen;
     xlabel=L"y",
     ylabel=L"\psi(y)",
     title="Fungsi Gelombang dari Metode Eigen",
     grid=true, gridstyle=:solid)
```


Fungsi Gelombang dari Metode Eigen
y1
0.08
0.06
0.04
0.02
0.00
$\psi(y)$
−0.25
0.00
0.25
0.50
0.75
1.00
y

0.7 Gerak Bandul dengan Evaluasi Fungsi

0.7.1 Latar Belakang

Ketika suatu model yang mewakili suatu sistem fisis dapat diselesaikan, pada umunya penyelesaian suatu sistem fisis tersebut berupa suatu fungsional, yaitu bentuk kompak (*closed form*) yang melibatkan bentuk fungsi tertentu. Beberapa fungsi yang terlibat dalam penyelesaian boleh jadi berbentuk sederhana sehingga dapat langsung dimanfaatkan dalam perhitungan.

Namun, fungsi yang terlibat dalam banyak penyelesaian kajian fisis ternyata tidak memiliki bentuk sederhana. Peran metode evaluasi fungsi menjadi penting untuk kondisi seperti ini agar perilaku atau sifat penting bagi fungsi dapat dipahami pada rentang peubah atau *domain* yang ditinjau dan akibatnya pemanfaatan penyelesaian bagi sistem yang melibatkan perhitungan fungsi akan dapat diperoleh.

0.7.2 Sistem Bandul

Sebagai contoh maka dapat ditinjau suatu sistem bandul bermassa m yang digantungkan pada seutas tali dengan panjang l dan dipengaruhi oleh medan gravitasi bumi g. Salah satu bentuk penyelesaian gerak ayunan bandul pada simpangan θ pada waktu tertentu t akan dapat dinyatakan sebagai ungkapan

$$t = \sqrt{\frac{l}{g}} \int_{\pi/2}^{\xi(\theta)} \frac{d\xi}{\sqrt{1 - k^2 \sin^2 \xi}}; \qquad k = \sin \frac{\theta_0}{2} \tag{203}$$

Kaitan antara sudut simpangan θ terhadap peubah baru ξ diberikan oleh kaitan

$$\sin \frac{\theta}{2} = k \sin \xi \tag{204}$$

Berdasar ungkapan pers (1) maka periode bandul T dapat diungkapkan dalam bentuk

$$T = 4 \sqrt{\frac{l}{g}} \int_{0}^{\pi/2} \frac{d\xi}{\sqrt{1 - k^2 \sin^2 \xi}} \tag{205}$$

Dengan ungkapan periode bandul T seperti diberikan pers (3) maka penyelesaian gerak bandul dalam pers (1) dapat diungkapkan dalam bentuk lain

$$t = \sqrt{\frac{l}{g}} \int_{\pi/2}^{\xi(\theta)} \frac{d\xi}{\sqrt{1 - k^2 \sin^2 \xi}} = \sqrt{\frac{l}{g}} \int_{0}^{\pi/2} \frac{d\xi}{\sqrt{1 - k^2 \sin^2 \xi}} - \sqrt{\frac{l}{g}} \int_{0}^{\xi(\theta)} \frac{d\xi}{\sqrt{1 - k^2 \sin^2 \xi}} \tag{206}$$

Dengan ungkapan lain maka penyelesaian gerak bandul akan setara dengan bentuk

$$t = \frac{T}{4} - \sqrt{\frac{l}{g}} \int_{0}^{\xi(\theta)} \frac{d\xi}{\sqrt{1 - k^2 \sin^2 \xi}} \tag{207}$$

Dalam satuan universal atau besaran waktu yang tak bersatuan yaitu $\tau = \frac{t}{T}$, ungkapan pers (4a) dapat dinyatakan dalam bentuk

$$\tau = \frac{1}{4} - \frac{1}{4} \frac{\int_{0}^{\xi(\theta)} \frac{d\xi}{\sqrt{1-k^2 \sin^2 \xi}}}{\int_{0}^{\pi/2} \frac{d\xi}{\sqrt{1-k^2 \sin^2 \xi}}} \tag{208}$$

0.7.3 Integral Eliptik

Istilah integral eliptik dikaitkan dengan fungsi yang didefinisikan dalam bentuk integral layak tertentu, seperti diuraikan secara detail pada tautan Integral Eliptik: Wikipedia atau Integral Eliptik: Wolfram.

Integral Eliptik tak-lengkap jenis pertama (*Incomplete elliptic integral of the first kind*) $F(\theta_0, k)$ didefinisikan sebagai

$$F(\theta_0, k) = \int_0^{\theta_0} \frac{d\theta}{\sqrt{1 - k^2 \sin^2 \theta}} \tag{209}$$

Adapun integral Eliptik lengkap jenis pertama (*Complete elliptic integral of the first kind*) K(k) didefinisikan sebagai

$$K(k) = \int_0^{\pi/2} \frac{d\theta}{\sqrt{1 - k^2 \sin^2 \theta}} \tag{210}$$

Berdasar definisi integral Eliptik tersebut maka penyelesaian gerak bandul dalam pers (4) dapat dinyatakan sebagai penyelesaian yang melibatkan fungsi dari integral Eliptik yaitu

$$\tau = \frac{1}{4}\left[1 - \frac{F(\xi, k)}{K(k)}\right] \tag{211}$$

Dengan pers (5) tersebut maka penyelesaian gerak bandul $\theta(t)$ beserta akan dapat ditentukan apabila evaluasi bagi fungsi integral Eliptik $F(\xi(\theta), k)$ beserta $K(k)$ berhasil dilakukan.

0.7.4 Perhitungan Integral Eliptik

Integral Eliptik merupakan salah satu fungsi khas (*special functions*) yang sering muncul dalam penyelesaian berbagai kajian fisika. Pada umumnya, sifat-sifat dari fungsi khas dinyatakan dalam berbagai ungkapan antara lain dalam bentuk integral, ekspansi deret pangkat, kaitan rekurensi, bentuk asimtotik serta dalam penyajian fungsi khas lainnya.

Ungkapan deret bagi integral Eliptik lengkap jenis pertama berbentuk

$$K(k) = \frac{\pi}{2} \sum_{n=0}^{\infty} \left(\frac{(2n)!}{2^{2n}(n!)^2}\right)^2 k^{2n} \tag{212}$$

$$K(k) = \frac{\pi}{2}\left[1 + \left(\frac{1}{2}\right)^2 k^2 + \left(\frac{1.3}{2.4}\right)^2 k^4 + \cdots + \left(\frac{(2n-1)!!}{(2n)!!}\right)^2 k^{2n} + \cdots\right] \tag{213}$$

Arti dari notasi faktorial ganda atau semifaktorial ($n!!$) pada pers (7b) dapat merujuk pada tautan Double Factorial

Ungkapan deret bagi integral Eliptik tak-lengkap jenis pertama memiliki bentuk yang beragam sesuai rentang nilai argumen θ_0 dan k yang diberikan.

Dengan ungkapan pers (7a) maka periode bandul T akan dapat diperoleh apabila evaluasi ungkapan deret bagi ntegral Eliptik lengkap jenis pertama tersebut.

0.7.5 Evaluasi Berdasar Kaitan Rekurensi

Selain ungkapan penyajian deret, beberapa penyelesaian permasalahan fisika sering melibatkan fungsi khas (*special function*) yang disajikan dalam bentuk kaitan rekurensi. Sebagai contoh, bentuk fungsi gelombang yang merupakan penyelesaian bagi persamaan Schrodinger untuk sistem osilator harmonik dalam mekanika kuantum akan melibatkan fungsi khas yang disebut polinomial *Hermite* $H_n(x)$ berorde n dalam bentuk

$$\psi_n(x) = \frac{1}{\sqrt{2^n\, n!}} \left(\frac{m\omega}{\pi\hbar}\right)^{1/4} e^{-\frac{m\omega}{2\hbar}x^2} H_n\left(\sqrt{\frac{m\omega}{\hbar}}x\right); \quad n = 0, 1, 2, \cdots \tag{214}$$

Dalam ungkapan tersebut, m adalah massa benda, $\hbar = h/2$ dengan h adalah tetapan Planck dan ω adalah frekuensi osilasi (lihat di tautan Quantum Harmonic Oscillator).

Polinomial *Hermite* $H_n(x)$ berode n (Hermite polynomials) dapat dinyatakan dalam ungkapan deret seperti berikut

$$H_n(x) = \begin{cases} n! \sum_{j=0}^{\frac{n}{2}} \frac{(-1)^{\frac{n}{2}-j}}{(2j)!\left(\frac{n}{2}-j\right)!}(2x)^{2j} & \text{jika } n \text{ genap} \\ n! \sum_{j=0}^{\frac{n-1}{2}} \frac{(-1)^{\frac{n-1}{2}-j}}{(2j+1)!\left(\frac{n-1}{2}-j\right)!}(2x)^{2j+1} & \text{jika } n \text{ ganjil} \end{cases} \tag{215}$$

atau penyajian kaitan rekurensi seperti berikut

$$H_{n+1}(x) = 2xH_n(x) - 2nH_{n-1}(x) \tag{216}$$

Secara umum, ketika dua bentuk polinomial *Hermite* pada orde rendah diketahui yaitu $H_0(x)$ dan $H_1(x)$, maka perhitungan polinimial *Hermite* pada sebarang x dan sebarang orde n akan lebih mudah dilakukan berdasar kaitan rekurensi dibanding dengan penyajian deret. Berdasar

0.7.6 Evaluasi Fungsi Integral Eliptik untuk Gerak Bandul

Ungkapan deret yang mewakili Integral Eliptik lengkap jenis pertama, seperti diberikan oleh pars (7a) dan (7b), dapat dinyatakan dalam bentuk yang mirip pers (9) yaitu:

$$K(k) = \frac{\pi}{2} \sum_{n=0}^{\infty} a_i = (a_0 + a_1 + a_2 + \cdots + a_n + \cdots) \tag{217}$$

dengan koefisien a_i diberikan oleh kaitan rekurensi berikut

$$a_i = \left[\frac{k(2i-1)}{2i}\right]^2 a_{i-1}; \qquad a_0 = 1 \tag{218}$$

Dibanding pers (7a) dan (7b), nampak bahwa bentuk pangkat orde tinggi beserta faktorial pada pers (18) menjadi tidak nampak secara eksplisit.

Source code berikut adalah perhitungan nilai $K(k)$ pada sebarang k berdasar implementasi ungkapan deret pers (18).

```
function my_ellipk(k)
        ai = 1.0
        sum = ai
        for i in 1:19
            ai *= (k * (2.0*i -1)/(2.0*i))^2
            sum += ai
        end
        myseries = \pi * sum / 2.0
        return myseries
    end
```

```
my_ellipk (generic function with 1 method)
```

Dengan membandingkan implementasi *Source code* tersebut dengan package `SpecialFunctions` dalam Julia untuk fungsi Integral Eliptik lengkap jenis pertama yaitu `ellipk` nampak bahwa hasil keduanya telah sesuai.

```
using SpecialFunctions

k = 0.6
m = k*k
ellip = ellipk(m);
eksak = my_ellipk(k);

ellip, eksak
```

```
(1.7507538029157526, 1.7507538028654843)
```

Evaluasi Integral Eliptik Lengkap Jenis Pertama dengan Metode Rekurensi

Selain dapat disajikan dengan ungkapan deret, Integral Eliptik lengkap jenis pertama dapat dilakukan lebih efektif dengan memanfaatkan ungkapan rekursif. Bentuk integral eliptik lengkap jenis pertama $K(k)$ dapat dinyatakan dalam fungsi Arithmetic–Geometric Mean (AGM), yang dinotasikan oleh $M(x, y)$, oleh kaitan

$$K(k) = \frac{\pi}{2M(1, \sqrt{1-k^2})} \tag{219}$$

Evaluasi fungsi AGM $M(x, y)$ didasarkan pada prosedur berikut:

1. Tinjau jajaran nilai-nilai a_i dan g_i untuk $i = 0, 1, 2, \cdots$
2. Mulai jajaran dengan x dan y yaitu $a_0 = x$ dan $g_0 = y$
3. Lanjutkan jajaran berikutnya dengan kaitan rekurensi:
   - $a_{i+1} = \frac{1}{2}(a_i + g_i)$
   - $g_{i+1} = \sqrt{a_i g_i}$
4. Dua jajaran a_i dan g_i akan konvergen pada satu nilai fungsi AGM yaitu $M(x, y)$

Dengan prosedur tersebut maka evaluasi fungsi $M(1, \sqrt{1-k^2})$ untuk memperoleh nilai Integral Eliptik lengkap jenis pertama seperti diberikan pers (20) adalah seperti berikut:

```
function my_ellipk_rekurensi(k)
    a0 = 1.0
    g0 = sqrt(1.0 - k^2)
    for i in 1:10
        ai = (a0 + g0) / 2.0
        gi = sqrt(a0 * g0)
        a0 = ai
        g0 = gi
    end
    myrekur = \pi / (2.0 * a0)
    return myrekur
end

my_ellipk_rekurensi (generic function with 1 method)

myrekur=my_ellipk_rekurensi(k)

1.7507538029157523
```

Nampak dari hasil di atas bahwa nilai $K(k)$ yang diperoleh berdasar metode deret serta metode rekurensi telah sesuai, namun metode rekurensi nampak lebih cepat untuk konvergen.

0.7.7 Evaluasi Integral Eliptik Tak Lengkap Jenis Pertama

Dalam masalah perhitungan periode bandul, evaluasi Integral Eliptik tak-lengkap jenis pertama (*Incomplete elliptic integral of the first kind*) dilakukan dengan integrasi numerik yaitu metode Simpson. Untuk masalah gerak bandul, yaitu menentukan simpangan bandul sebagai fungsi waktu, evaluasi Integral Eliptik tak-lengkap jenis pertama perlu dilakukan beberapa kali sehingga penggunaan kuadratur numerik akan nampak lebih efisien. Ungkapan Integral Eliptik tak-lengkap jenis pertama $F(\xi, k)$ disajikan oleh pers (5a) yaitu:

$$F(\xi, k) = \int_0^{\xi} \frac{d\xi}{\sqrt{1 - k^2 \sin^2 \xi}} \tag{220}$$

Memanfaatkan uraian yang diberikan pada materi kuliah terdahulu terkait implementasi metode kuadratur numerik pada perhitungan periode bandul maka *source-code* untuk perhitungan Integral Eliptik tak-lengkap jenis pertama $F(\xi, k)$ disajikan oleh pers (21) adalah seperti berikut:

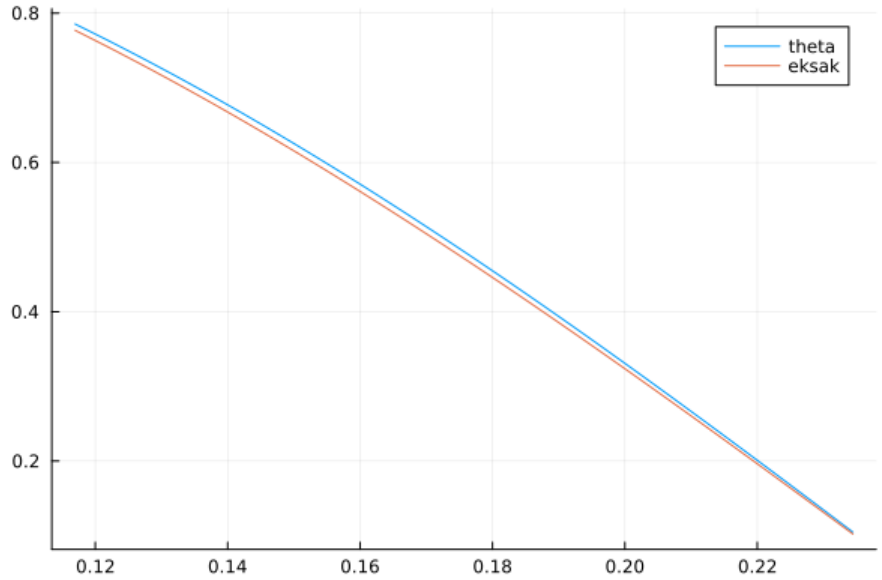
```
using FastGaussQuadrature

function fung(x, k)
  return 1.0 / sqrt(1.0 - (k * sin(x))^2)
end

function my_ellip_in(xi, k)
  n = 15
  # Legendre -Gauss nodes and weights
  x, c = gausslegendre(n)
  a = 0.0
  b = xi
  sum = 0.0
  for i in 1:n
    y = (b - a) * x[i] / 2.0 + (b + a) / 2.0
    sum += c[i] * fung(y, k)
  end
  my_quad = (b - a) * sum / 2.0
  return my_quad
end

my_ellip_in (generic function with 1 method)

using EllipticFunctions

const pi = \pi
theta0 = pi/4.0
k = sin(theta0/2.0)
theta = pi/6.0
p = sin(theta/2.0)
xi = asin(p/k)
m = k*k

inc_ellip = ellipticF(xi, m);
hasil = my_ellip_in(xi, k);

inc_ellip, hasil

(0.7520103018323488 - 0.0im, 0.7520103018323493)
```

Nampak dari hasil di atas bahwa nilai $F(\xi, k)$ yang diperoleh berdasar metode kuadratur numerik dibanding fungsi `ellipticF` dari package `EllipticFunctions` dalam Julia telah sesuai.

Dengan evaluasi fungsi-fungsi tersebut maka penyelesaian gerak baandul melalui perhitungaan waktu τ sebagai fungsi simpangan *theta*, dan inversnya yaitu θ sebagai fungsi waktu τ, akan dapat diperoleh.

```
n = 30
theta = range(pi/n, pi/4, length=n)
tau = zeros(n)
eksak = zeros(n)
theta0 = pi/3.0
```

```
k = sin(theta0/2.0)
for i in 1:n
    p = sin(theta[i]/2.0)
    xi = asin(p/k)
    inc_ellip = my_ellip_in(xi, k)
    ellip = my_ellipk_rekurensi(k)
    tau[i] = 0.25 * (1.0 - inc_ellip / ellip)
    eksak[i] = theta0 * cos(2 * pi * tau[i])
end

using Plots
plot(tau, theta, label="theta")
plot!(tau, eksak, label="eksak")
```



Dari plot di atas nampak bahwa hasil penyelesaian gerak berdasar evaluasi fungsi telah sesuai dengan hasil eksak yang diperoleh saat sudut simpangan awal kecil. Dengan demikian hasil penyelesaian gerak berdasar evaluasi fungsi dapat digunakan untuk sebarang simpangan.

0.8 Penyelesaian Persamaan Diferensial Parsial Parabolik

0.8.1 Implementasi untuk Persmaan Difusi Panas

Contoh permasalahan yang memerlukan ongkos komputasi tinggi diantaranya adalah penyelesaian persamaan diferensial parsial, yaitu suatu persamaan diferensial yang melibatkan lebih dari satu peubah bebas. Adanya lebih dari satu peubah bebas menyebabkan munculnya pertimbangan langkah diskretisasi karena diskretisasi pada satu peubah bebas akan berpotensi menimbulkan dampak pada diskretisasi peubah bebas lainnya. Pertimbangan yang nampak perlu dipertimbangkan adalah munculnya dimensi matrik yang luar biasa besar untuk mendapatkan hasil yang stabil dan ketelitian yang memadai.

Pada umumnya ada 3 jenis persamaan diferensial parsial yaitu

1. Persamaan diferensial eliptik dengan contoh yaitu

> a. Persamaan Poisson dalam bentuk

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = S(x, y)$$

> b. Persamaan Schrodinger tak gayut waktu dalam bentuk

$$-\frac{\hbar^2}{2m}\left[\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2}\right] + \hat{V}(x, y)\psi = E\psi$$

2. Persamaan diferensial parabolik dengan contoh yaitu

> a. Persamaan difusi dalam bentuk

$$k\frac{\partial^2 u}{\partial x^2} - \frac{\partial u}{\partial t} = f(x, t)$$

> b. Persamaan Scrodinger gayut waktu dalam bentuk

$$i\hbar\frac{\partial \psi}{\partial t} = -\frac{\hbar^2}{2m}\frac{\partial^2 \psi}{\partial x^2} + \hat{V}(x)\psi = \hat{H}\psi$$

3. Persamaan diferensial hiperbolik dengan contoh yaitu

> Persamaan gelombang dalam bentuk

$$c^2\frac{\partial^2 u}{\partial x^2} - \frac{\partial u}{\partial t^2} = f(x, t)$$

0.8.2 Penyelesaian Persamaan Difusi dengan Metode Beda Hingga

Dari bentuk persamaan diferensial parabolik di atas akan nampak bahwa adanya bentuk turunan satu kali terhadap peubah bebas waktu (t) maka penyelesaian akan melibatkan masalah syarat awal, mirip seperti penyelesaian pada gerak bandul. Dilain pihak, adanya bentuk turunan dua kali terhadap peubah bebas posisi (x) maka penyelesaian akan melibatkan masalah syarat batas, mirip seperti penyelesaian pada persamaan Poisson. Untuk masalah syarat awal maka hal yang perlu mendapat perhatian adalah pada persyaratan kestabilan, sedangkan masalah syarat batas maka hal yang perlu mendapat perhatian adalah pada persyaratan ketelitian. Kedua hal tersebut akan memunculkan suatu ungkapan umum dalam penyelesaian persamaan diferensial parabolik bahwa kestabilan dan ketelitian akan dapat dicapai apabila memenuhi ungkapan berikut.

$$\frac{k\Delta t}{(\Delta x)^2} \leq 1 \tag{221}$$

Bentuk persamaan difusi panas $T(x,t)$ dalam 1 dimensi adalah:

$$\frac{\partial T}{\partial t} = D\frac{\partial^2 T}{\partial x^2} + S(x,t)$$

Dalam ungkapan di atas, D adalah koefisien difusi panas dan $S(x,t)$ adalah sumber panas dengan $a \leq x \leq b$. Syarat awal dan syarat batas yang diberikan dianggap berjenis Derichlet yaitu

$$\begin{aligned} T(a,t) &= T_a \\ T(b,t) &= T_b \\ T(x,0) &= T_0(x) \end{aligned}$$

Yang menjadi tujuan dari masalah penyelesaian difusi panas adalah untuk memperoleh distribusi temperatur pada sebarang ruang dan waktu yaitu $T(x,t)$. Dalam masalah ini maka sumber panas $S(x,t)$ dianggap diketahui, termasuk nilai temperatur pada waktu awal $T(x,0)$ serta nilai temperatur pada batas daerah di $x = a$ serta $x = b$, yaitu $T(a,t), T(b,t)$, semuanya telah diketahui.

Penyelesaian difusi panas merupakan gabungan antara penyelesian masalah syarat awal dan syarat batas karena bentuk persamaan diferensial adalah bertipe persamaan parabolik. Pada umumnya, hal yang perlu menjadi perhatian dalam tipe penyelesaian persamaan diferensial parabolik adalah potensi terjadinya ketakstabilan pada langkah numerik.

Untuk memudahkan penulisan dalam prosedur komputasi, penyelesaian difusi panas akan menggunakan notasi diskret x_n untuk posisi pada titik ke $n = 0, \cdots, N$ dan t_j untuk waktu pada saat ke $j = 0, \cdots, J$. Dengan demikian, sebarang posisi dapat didekati dengan posisi diskret x_n yang memiliki kaitan:

$$x_n = x_0 + n\Delta x$$

dengan $\Delta x = \frac{x_N - x_0}{N}$ adalah ukuran langkah (*step size*) sedangkan $x_0 = a$ dan $x_N = b$ adalah posisi di kedua ujung batas daerah.

Dengan cara yang sama, sebarang waktu dapat didekati dengan waktu diskret t_j yang memiliki kaitan:

$$t_j = t_0 + j\Delta t$$

dengan $\Delta t = \frac{t_J - t_0}{J}$ adalah selang waktu (*time step*) sedangkan t_0 dan t_J adalah waktu awal dan akhir.

Notasi untuk nilai diskret temparatur dan sumber panas pada posisi x_n dan waktu t_j, yaitu $T(x_n, t_j)$ dan $S(x_n, t_j)$, biasa ditulis

$$T_n^j \equiv T(x_n, t_j); \qquad S_n^j \equiv S(x_n, t_j)$$

Ungkapan beda hingga (*finite difference*)

Bentuk diskret operasi diferensial bagi sebarang fungsi $f(y)$ terhadap peubah bebas y dapat dinyatakan dalam ungkapan beda hingga berikut

1. Beda hingga maju (*forward difference*) untuk operasi diferensial satu kali

$$\frac{df(y)}{dy}\Big]_{y=y_n} \approx \frac{f_{n+1} - f_n}{\Delta y}$$

2. Beda hingga mundur (*backward difference*) untuk operasi diferensial satu kali

$$\frac{df(y)}{dy}\Big]_{y=y_n} \approx \frac{f_n - f_{n-1}}{\Delta y}$$

3. Beda hingga terpusat (*central difference*) untuk operasi diferensial satu kali

$$\frac{df(y)}{dy}\Big]_{y=y_n} \approx \frac{f_{n+1} - f_{n-1}}{2\Delta y}$$

4. Beda hingga terpusat (*central difference*) untuk operasi diferensial dua kali

$$\frac{d^2f(y)}{dy^2}\Big]_{y=y_n} \approx \frac{f_{n+1} - 2f_n + f_{n-1}}{\Delta y^2}$$

0.8.3 Penyelesaian dengan Metode Beda Hingga (*Finite Difference*)

Skema eksplisit

Persamaan difusi panas di atas dapat ditulis dalam bentuk diskret, yang disebut persamaan beda hingga, dengan memanfaatkan ungkapan beda hingga maju (*forward difference*) untuk operasi diferensial satu kali terhadap waktu dan ungkapan beda hingga terpusat (*central difference*) untuk operasi diferensial dua kali terhadap posisi, sehingga berubah menjadi

$$\left(\frac{T_n^{j+1} - T_n^j}{\Delta t}\right) = D\left(\frac{T_{n+1}^j - 2T_n^j + T_{n-1}^j}{\Delta x^2}\right) + S_n^j$$

dengan $n = 1, \cdots, N-1$, $j = 0, \cdots, J$, dan T_n^0, T_0^j, T_N^j semuanya memiliki nilai yang telah diketahui. Penyelesaian persamaan beda hingga dengan skema eksplisit dapat dilakukan dengan mengubah ungkapan tersebut ke dalam bentuk

$$T_n^{j+1} = \lambda T_{n-1}^j + (1 - 2\lambda)\, T_n^j + \lambda T_{n+1}^j + \Delta t S_n^j$$

Dalam ungkapan di atas digunakan parameter $\lambda = D\frac{\Delta t}{\Delta x^2}$.

Ungkapan di atas disebut skema eksplisit karena dengan mengetahui nilai-nilai T_n^0, T_0^j, T_N^j maka nilai-nilai T_n^j akan dapat dihitung secara langsung dari ungkapan tersebut. Skema eksplisit tersebut dapat dinyatakan dalam operasi matrik seperti berikut.

$$\begin{pmatrix} T_1^{j+1} \\ T_2^{j+1} \\ \vdots \\ T_{N-1}^{j+1} \end{pmatrix} = \begin{pmatrix} (1-2\lambda) & \lambda & 0 & \cdots & 0 \\ \lambda & (1-2\lambda) & \lambda & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \\ 0 & \cdots & 0 & \lambda & (1-2\lambda) \end{pmatrix} \begin{pmatrix} T_1^j \\ T_2^j \\ \vdots \\ T_{N-1}^j \end{pmatrix} + \Delta t \begin{pmatrix} S_1^j \\ S_2^j \\ \vdots \\ S_{N-1}^j \end{pmatrix} + \lambda \begin{pmatrix} T_0^j \\ 0 \\ \vdots \\ T_N^j \end{pmatrix}$$

Meskipun skema eksplisit nampak sederhana dari aspek komputasi, namun umum diketahui bahwa skema tersebut memiliki perilaku penyelesaian yang tidak baik yaitu tidak stabil. Dapat ditunjukkan bahwa untuk pengambilan selang waktu Δt yang tidak kecil dibanding $(\Delta x)^2$ maka akan memunculkan penyelesaian yang tidak benar, bahkan cenderung sangat berbeda jauh dengan nilai yang seharusnya.

Skema implisit

Untuk mengatasi kendala ketakstabilan tersebut maka dapat ditempuh skema implisit. Ada banyak cara untuk memperoleh ungkapan bagi skema implisit. Salah satu cara adalah dengan meninjau ungkapan beda hingga mulai pada waktu t_1 sehingga dapat dimanfaatkan beda hingga mundur (*backward difference*) pada operasi diferensial satu kali terhadap waktu. Dengan demikian ungkapan persamaan beda hingga akan menjadi

$$\left(\frac{T_n^{j+1} - T_n^j}{\Delta t}\right) = D\left(\frac{T_{n+1}^{j+1} - 2T_n^{j+1} + T_{n-1}^{j+1}}{\Delta x^2}\right) + S_n^{j+1}$$

Dalam ungkapan lain maka dapat disajikan dalam bentuk

$$-\lambda T_{n-1}^{j+1} + (1+2\lambda)\, T_n^{j+1} - \lambda T_{n+1}^{j+1} = T_n^j + \Delta t S_n^{j+1}$$

Dalam ungkapan di atas dapat dilihat bahwa perhitungan nilai T_n^{j+1} belum dapat diperoleh karena pada saat yang sama, nilai-nilai T_{n-1}^{j+1} dan ψ_{n+1}^{j+1} belum diketahui. Dengan kenyataan ini maka disebut skema implisit karena nilai T_n^{j+1} tidak dapat dinyatakan secara eksplisit dari nilai T lainnya yang diketahui.

Untuk memperoleh nilai-nilai T_n^{j+1} maka ungkapan persamaan linear di atas perlu diselesaikan secara simultan dengan persamaan linear lainnya berupa ungkapan persamaan matriks. Untuk mengetahui bentuk-bentuk matrik yang terlibat maka ungkapan skema implisit di atas dapaat dinyatakan dalam bentuk

$$\begin{pmatrix} (1+2\lambda) & -\lambda & 0 & \cdots & 0 \\ -\lambda & (1+2\lambda) & -\lambda & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \\ 0 & \cdots & 0 & -\lambda & (1+2\lambda) \end{pmatrix} \begin{pmatrix} T_1^{j+1} \\ T_2^{j+1} \\ \vdots \\ T_{N-1}^{j+1} \end{pmatrix} = \begin{pmatrix} T_1^j + \lambda T_0^{j+1} \\ T_2^j \\ \vdots \\ T_{N-1}^j + \lambda T_N^{j+1} \end{pmatrix} + \Delta t \begin{pmatrix} S_1^{j+1} \\ S_2^{j+1} \\ \vdots \\ S_{N-1}^{j+1} \end{pmatrix}$$

Nampak bahwa ungkapan tersebut merupakan bentuk operasi perkalian matrik dalam bentuk $Ay = b$, dengan matrik tridiagonal A dan matrik kolom b memiliki unsur-unsur matrik yang diketahui. Matrik y merupakan matrik kolom yang memiliki unsur matrik berupa nilai-nilai T_n^j pada lokasi diskrit x_n dan waktu t_j.

Dengan demikian, masalah untuk menyelesaikan persamaan difusi panas menggunakan skema implisit adalah masalah untuk memperoleh matrik kolom y berdasarkan operasi matrik $Ay = b$ ketika matrik A dan b telah diketahui. Beberapa operasi matrik baku dapat dipilih untuk penyelesaian ini.

Keunggulan dari skema implisit ketika dibandingkan dengan skema eksplisit adalah tidak munculnya ketakstabilan dalam proses komputasi, meskipun dilakukan pada selang waktu Δt yang tidak kecil dibanding $(\Delta x)^2$.

Namun kerugian dari skema implisit dibandingkan dengan skema eksplisit adalah mahal pada ongkos proses komputasi (*computationally costly*) mengingat operasi matrik perlu dilakukan untuk setiap selang waktu yang diambil.

Secara ringkas dapat ditunjukkan melalui skema eksplisit dan implisit bahwa penyelesaian persamaan difusi panas dapat ditempuh dengan menggunakan proses komputasi yang berdasar pada metode beda hingga.

Selanjutnya akan ditunjukkan bahwa metode beda hingga memiliki kendala alamiah yang sulit dihilangkan ketika digunakan untuk menyelesaikan permasalahan persamaan diferensial parabolik. Oleh karena itu akan ditinjau metode lain yang dapat mengatasi kendala tersebut.

Skema implisit Crank-Nicholson

Skema implisit seperti diuraikan di atas sudah cukup untuk menjamin bahwa kestabilan proses komputasi akan tercapai. Namun tujuan komputasi bukanlah hanya kestabilan namun juga ketelitian hasil komputasi. Salah satu skema implisit yang memfasilitasi hal tersebut adalah apa yang disebut metode Crank-Nicholson.

Metode Crank-Nicholson didasarkan pada pengambilan pendekatan tinggi yaitu orde 2 baik pada diskretisasi waktu t maupun diskretisasi ruang x. Dalam orde 2 tersebut maka diskretisasi waktu t dievaluasi pada waktu $t = t_{j+1/2}$ sehingga dapat digunakan pendekatan beda hingga terpusat dalam bentuk

$$\left.\frac{\partial T}{\partial t}\right]_{t=t_{j+1/2}} \approx \frac{T_n^{j+1} - T_n^j}{\Delta t}$$

Evaluasi pada $t = t_{j+1/2}$ tidak menimbulkan kesulitan pada diskretisasi waktu namun akan meumunculkan kendala ketika digunakan untuk melakukan pendekatan beda hingga terpusat pada diskretisasi ruang karena nilai T pada posisi sebarang saat $t = t_{j+1/2}$ tidak tersedia. Sebagai alternatif maka digunakan pendekatan nilai tengah yaitu nilai rerata T pada saat $T = t_j$ dan saat $t = t_{j+1}$ dalam bentuk

$$\begin{aligned}\left.\frac{\partial^2 T}{\partial x^2}\right]_{x=x_n}^{t=t_{j+1/2}} &\approx \frac{1}{2}\left(\left.\frac{\partial^2 T}{\partial x^2}\right]_{x=x_n}^{t=t_j} + \left.\frac{\partial^2 T}{\partial x^2}\right]_{x=x_n}^{t=t_{j+1}}\right) \\ &\approx \frac{T_{n+1}^j - 2T_n^j + T_{n-1}^j}{2\Delta x^2} + \frac{T_{n+1}^{j+1} - 2T_n^{j+1} + T_{n-1}^{j+1}}{2\Delta x^2}\end{aligned}$$

Ungkapan beda hingga bagi persamaan difusi panas berdasar skema tersebut menjadi

$$\frac{T_n^{j+1} - T_n^j}{\Delta t} = D\left(\frac{T_{n+1}^j - 2T_n^j + T_{n-1}^j}{2\Delta x^2} + \frac{T_{n+1}^{j+1} - 2T_n^{j+1} + T_{n-1}^{j+1}}{2\Delta x^2}\right) + S_n^{j+1/2}$$

Ungkapan tersebut biasa disebut skema implisit Crank-Nicholson yang disajikan dalam bentuk

$$-\lambda T_{n-1}^{j+1} + 2\left(1+\lambda\right)T_n^{j+1} - \lambda T_{n+1}^{j+1} = \lambda T_{n-1}^j + 2\left(1-\lambda\right)T_n^j + \lambda T_{n+1}^j + 2\Delta t S_n^{j+1/2}$$

Dalam operasi matrik, skema implisit Crank-Nicholson dapat dinyatakan dalam bentuk

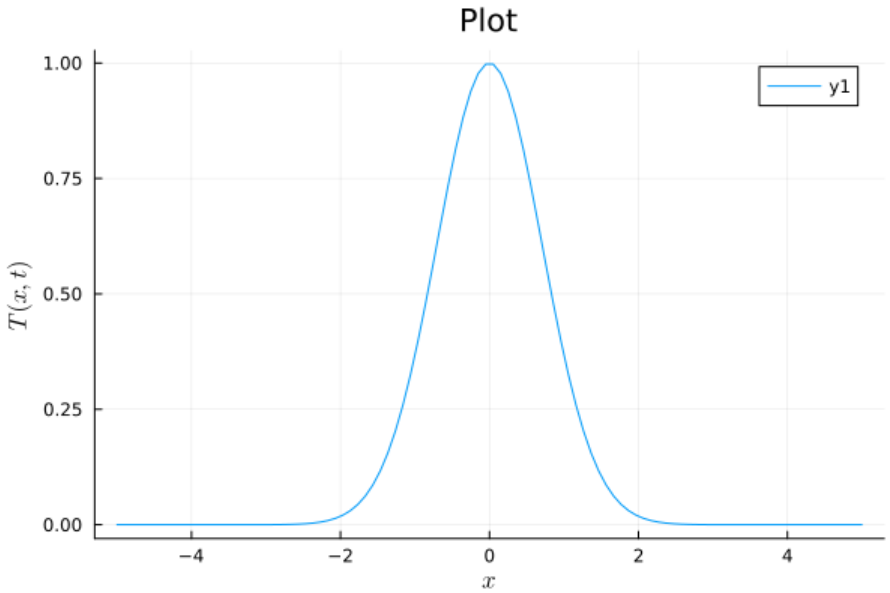
$$\begin{pmatrix} 2(1+\lambda) & -\lambda & 0 & \cdots & 0 \\ -\lambda & 2(1+\lambda) & -\lambda & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \\ 0 & \cdots & 0 & -\lambda & 2(1+\lambda) \end{pmatrix} \begin{pmatrix} T_1^{j+1} \\ T_2^{j+1} \\ \vdots \\ T_{N-1}^{j+1} \end{pmatrix} = \lambda \begin{pmatrix} T_0^j + T_0^{j+1} \\ 0 \\ \vdots \\ T_N^j + T_N^{j+1} \end{pmatrix} + \Delta t \begin{pmatrix} S_1^{j+1/2} \\ S_2^{j+1/2} \\ \vdots \\ S_{N-1}^{j+1/2} \end{pmatrix}$$
$$+ \begin{pmatrix} 2(1-\lambda) & \lambda & 0 & \cdots & 0 \\ \lambda & 2(1-\lambda) & \lambda & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \\ 0 & \cdots & 0 & \lambda & 2(1-\lambda) \end{pmatrix} \begin{pmatrix} T_1^j \\ T_2^j \\ \vdots \\ T_{N-1}^j \end{pmatrix}$$

Secara umum, metode Crank-Nicholson memiliki kelebihan dalam konteks kestabilan dan ketelitian hasil komputasi dibanding dua metode yang dijelaskan sebelumnya, namun dengan kelemahan terkait ongkos komputasi yang relatif lebih mahal dibanding dua metode lain yang disinggung tersebut.

0.8.4 Source code Skema Eksplisit

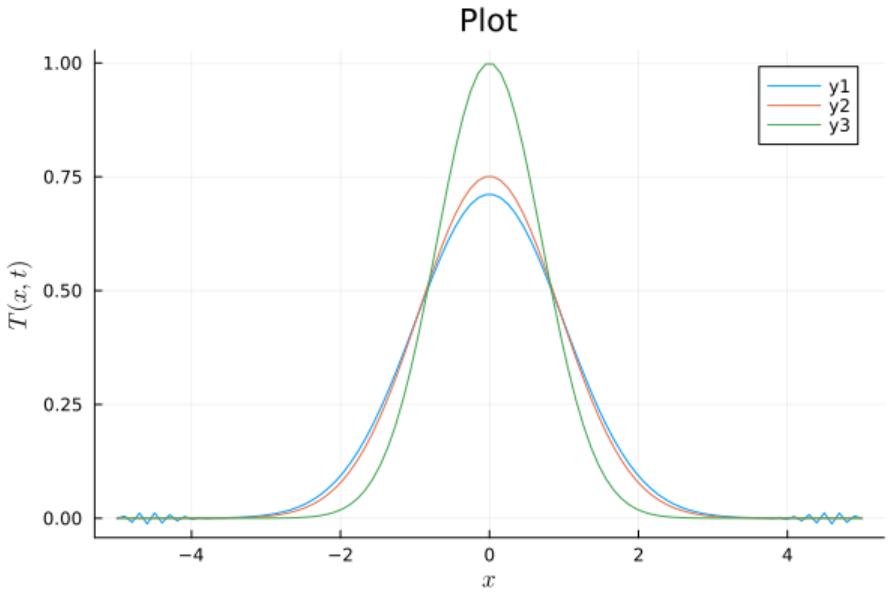
```
using LinearAlgebra
using Plots
using LaTeXStrings

Nx = 100
Nt = 1000
D = 1.0
```

```
x = range( -5, 5, length = Nx)
t = range(0, 10, length = Nt)

dx = x[2] - x[1]
dt = t[2] - t[1]

lambda = D * dt / dx^2

temper = zeros(Float64, Nx, Nt)
temper[:, 1] = exp.( -x.^2)
temper[1, :] .= 0.0
temper[end, :] .= 0.0;

# Plot the initial condition
plot(x, temper[:, 1],
     xlabel=L"$x$",
     ylabel=L"$T(x,t)$",
     title="Plot")
```



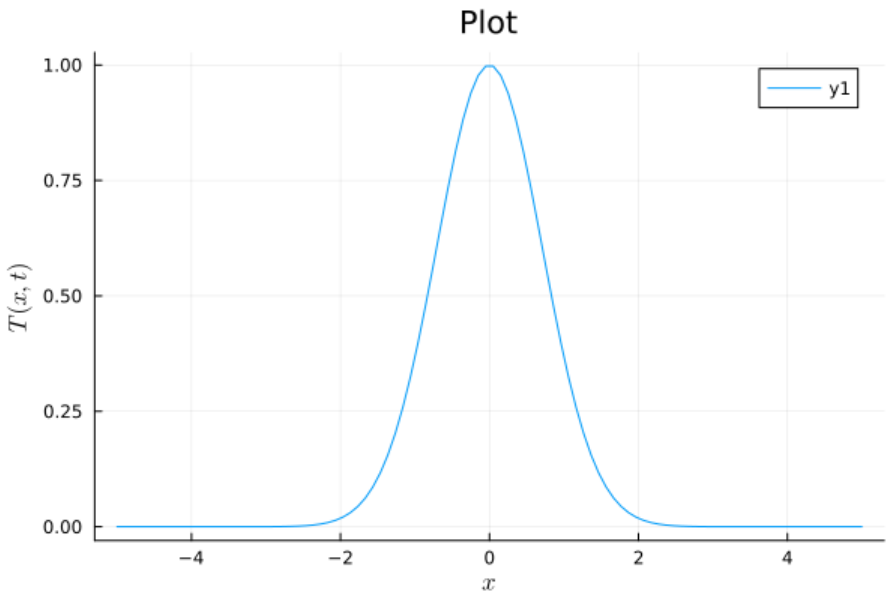

```
function matrix_eksplisit(lambda, Nx)
    a = (1.0 - 2.0 * lambda) * ones(Nx -2)
    b = lambda * ones(Nx -3)
    A = zeros(Nx -2, Nx -2)
    A = diagm(0 => a, -1 => b, 1 => b)
    return A
end

matrix_eksplisit (generic function with 1 method)

A = matrix_eksplisit(lambda, Nx);

function skema_eksplisit(temper, A, Nt)
    for j in 1:(Nt -1)
        temper[2:end -1, j+1] = A * temper[2:end -1, j]
    end
    return temper
end
```

```
skema_eksplisit (generic function with 1 method)

temper = skema_eksplisit(temper, A, Nt);

plot(x, [temper[:, 25],temper[:, 20], temper[:, 1]],
     xlabel=L"$x$",
     ylabel=L"$T(x,t)$",
     title="Plot")
```

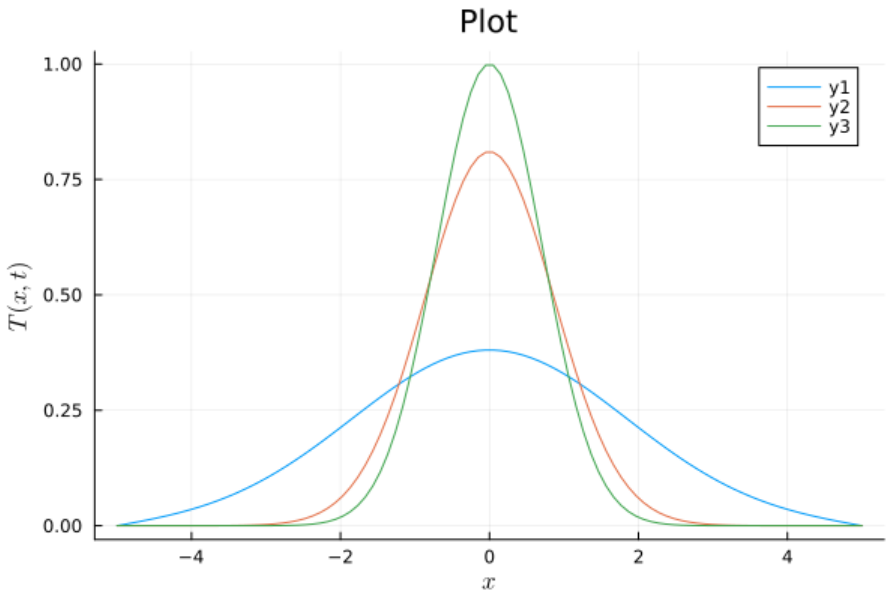


0.8.5 Source code Skema Implisit

```
Nx = 100
Nt = 1000
D = 1.5

x = range( -5, 5, length = Nx)
t = range(0, 10, length = Nt)

dx = x[2] - x[1]
dt = t[2] - t[1]

lambda = D * dt / dx^2

temper = zeros(Float64, Nx, Nt)
temper[:, 1] = exp.( -x.^2)
temper[1, :] .= 0.0
temper[end, :] .= 0.0;

plot(x, temper[:, 1],
     xlabel=L"$x$",
     ylabel=L"$T(x,t)$",
     title="Plot")
```




```
function matrix_implisit(lambda, Nx)
    a = (1.0 + 2.0 * lambda) * ones(Nx -2)
    b = -lambda * ones(Nx -3)
    A = zeros(Nx -2, Nx -2)
    A = diagm(0 => a, -1 => b, 1 => b)
    return A
end

matrix_implisit (generic function with 1 method)

A = matrix_implisit(lambda, Nx);
for i in 1:(Nt -1)
    temper[2:Nx -1, i+1] = A \ temper[2:Nx -1, i];
end

plot(x, [temper[:, 100],temper[:, 10], temper[:, 1]],
     xlabel=L"$x$",
     ylabel=L"$T(x,t)$",
     title="Plot")
```

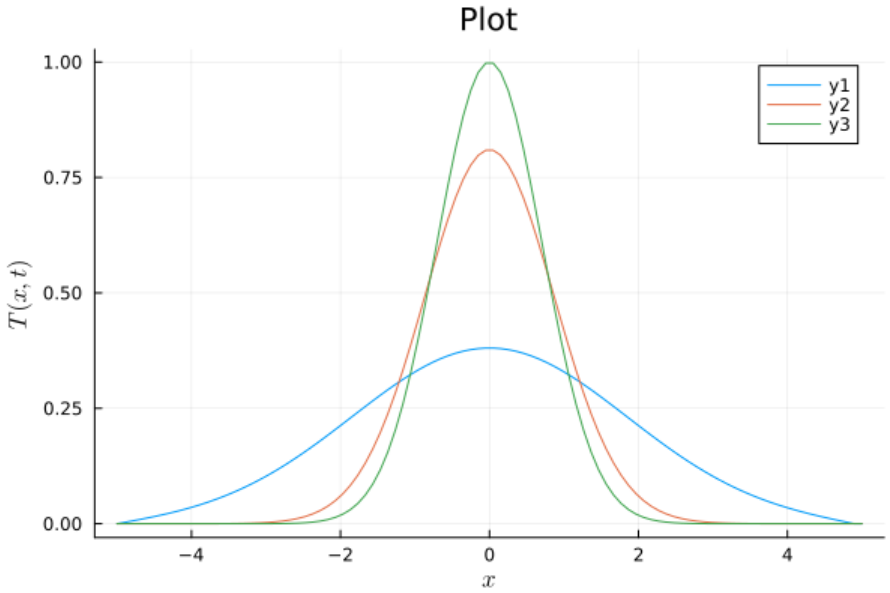


```
function skema_implisit(A, temper, lambda, Nt, Nx)
    d = diag(A)
    e = diag(A, -1)
    b = diag(A, 1)
    m = zeros(Nx -3)
    y = zeros(Nx -2)
    for i in 2:(Nx -2)
        m[i -1] = e[i -1] / d[i -1]
        d[i] -= m[i -1] * b[i -1]
    end

    for j in 1:(Nt -1)
        y = temper[2:end -1, j]
        for i in 3:(Nx -1)
            y[i -1] -= m[i -2] * y[i -2]
        end
        temper[Nx -1,j+1] = temper[Nx -1,j] / d[Nx -2]
        for i in (Nx -2): -1:2
            temper[i,j+1] = (y[i -1] - b[i -1] * temper[i+1,j+1]) / d[i -1]
        end
    end

    return temper
end
```

```
skema_implisit (generic function with 1 method)
```

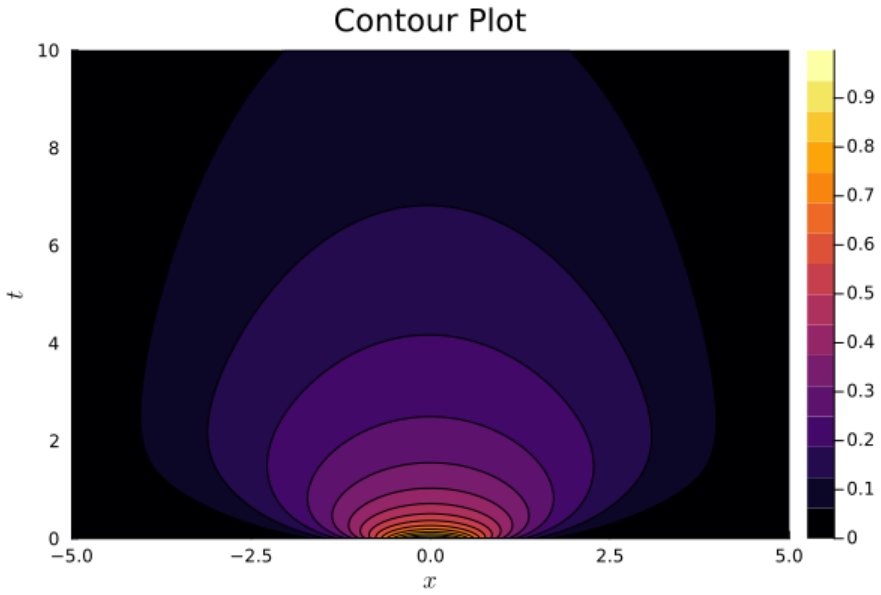
```
temper = skema_implisit(A, temper, lambda, Nt, Nx);
```

```
plot(x, [temper[:, 100],temper[:, 10], temper[:, 1]],
     xlabel=L"$x$",
     ylabel=L"$T(x,t)$",
     title="Plot")
```

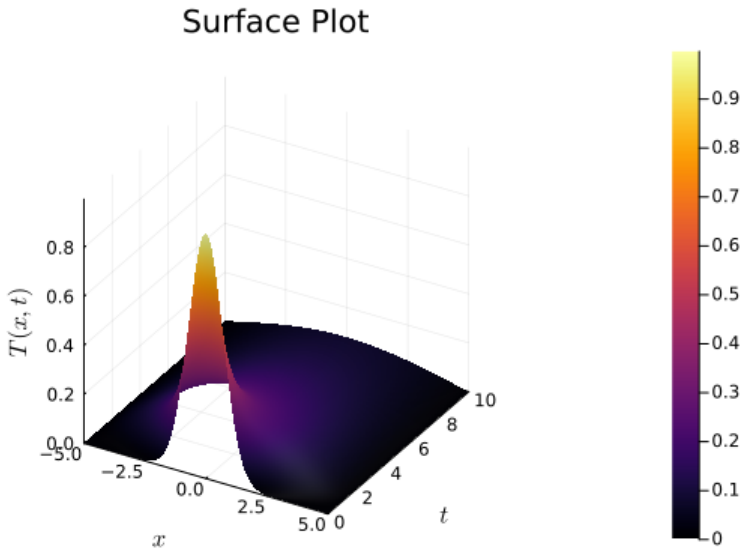



```
contour(x, t, temper',
        xlabel=L"$x$",
        ylabel=L"$t$",
```

```
        title="Contour Plot",
        fill=true)
```




```
surface(x, t, temper',
        xlabel=L"$x$",
        ylabel=L"$t$",
        zlabel=L"$T(x,t)$",
        title="Surface Plot")
```

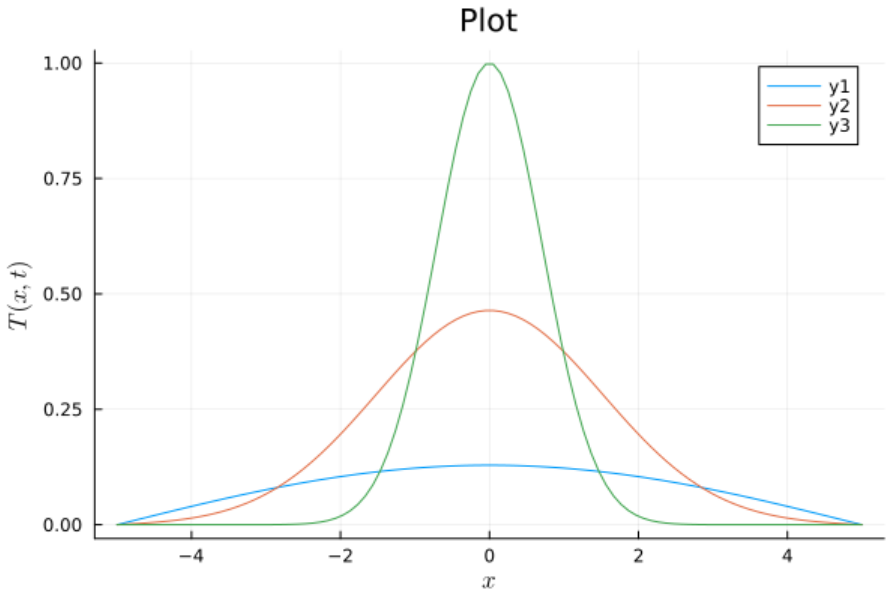


0.8.6 Source code Skema Crank Nicholson

```
Nx = 100
Nt = 100
D = 1.0

x = range( -5, 5, length = Nx)
t = range(0, 10, length = Nt)

dx = x[2] - x[1]
```

```
dt = t[2] - t[1]

lambda = D * dt / dx^2

temper = zeros(Float64, Nx, Nt)
temper[:, 1] = exp.( -x.^2)
temper[1, :] .= 0.0
temper[end, :] .= 0.0;

function matrix_kanan(lambda, Nx)
    a = 2.0 * (1.0 - lambda) * ones(Nx -2)
    b = lambda * ones(Nx -3)
    A = zeros(Nx -2, Nx -2)
    A = diagm(0 => a, -1 => b, 1 => b)
    return A
end

matrix_kanan (generic function with 1 method)

function matrix_kiri(lambda, Nx)
    a = 2.0 * (1.0 + lambda) * ones(Nx -2)
    b = -lambda * ones(Nx -3)
    A = zeros(Nx -2, Nx -2)
    A = diagm(0 => a, -1 => b, 1 => b)
    return A
end

matrix_kiri (generic function with 1 method)

function skema_CN(A_kanan, A_kiri, temper, lambda, Nt, Nx)
    d = diag(A_kiri)
    e = diag(A_kiri, -1)
    b = diag(A_kiri, 1)
    m = zeros(Nx -3)
    y = zeros(Nx -2)
    for i in 2:(Nx -2)
        m[i -1] = e[i -1] / d[i -1]
        d[i] -= m[i -1] * b[i -1]
    end

    for j in 1:(Nt -1)
        y = A_kanan * temper[2:end -1, j]
        for i in 3:(Nx -1)
            y[i -1] -= m[i -2] * y[i -2]
        end
        temper[Nx -1,j+1] = y[Nx -2] / d[Nx -2]
        for i in (Nx -2): -1:2
            temper[i,j+1] = (y[i -1] - b[i -1] * temper[i+1,j+1]) / d[i -1]
        end
    end

    return temper
end

skema_CN (generic function with 1 method)
```

```
A_kiri = matrix_kiri(lambda, Nx);
A_kanan = matrix_kanan(lambda, Nx);

temper = skema_CN(A_kanan, A_kiri, temper, lambda, Nt, Nx);

plot(x, [temper[:, 100],temper[:, 10], temper[:, 1]],
     xlabel=L"$x$",
     ylabel=L"$T(x,t)$",
     title="Plot")
```



0.8.7 Komputasi berdasar Ekponensial Matrik

Suatu prosedur penyelesaiaan persamaan diferensial parsial parabolik yang agak berbeda dengan yang diuraikan di atas dapat ditempuh dengan cara berikut. Ditinjau penyelesaian persamaan difusi panas ketika langkah diskretisasi hanya dilakukan dalam peubah posisi x sedangkan peubah waktu t tetap dalam ungkapan kontinu. Apabila nilai temperatur pada dua titik batas diberikan kondisi berikut yaitu $T(a,t) = 0$ dan $T(b,t) = 0$ dan tidak ada sumber panas yaitu $S(x,t) = 0$, maka persamaan difusi panas dapat ditulis dalam bentuk berikut

$$\frac{\partial}{\partial t}\underbrace{\begin{pmatrix} T_1^t \\ T_2^t \\ \vdots \\ T_{N-1}^t \end{pmatrix}}_{T(x,t)} = \underbrace{\begin{pmatrix} -2\gamma & \gamma & 0 & \cdots & 0 \\ \gamma & -2\gamma & \gamma & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \\ 0 & \cdots & 0 & \gamma & -2\gamma \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} T_1^t \\ T_2^t \\ \vdots \\ T_{N-1}^t \end{pmatrix}}_{T(x,t)}$$

$$\frac{\partial}{\partial t}T(x,t) = AT(x,t); \qquad \text{dengan } \gamma = \frac{D}{(\Delta x)^2}$$

Ungkapan matrik pada persamaan diferensial di atas mirip seperti bentuk persamaan diferensial orde satu bagi peubah bebas t yaitu $\frac{d}{dt}f(t) = cf(t)$ dengan penyelesaian berbentuk $f(t) = e^{ct}f(0)$ untuk c adalah skalar.

Analog dengan bentuk penyelesaian tersebut maka penyelesaian persamaan diferensial bagi operasi matrik di atas akan berbentuk

$$T(x,t) = e^{At}T(x,0)$$

Bentuk eksplisit dalam operasi matrik bagi ungkapan di atas adalah

$$\begin{pmatrix} T_1^t \\ T_2^t \\ \vdots \\ T_{N-1}^t \end{pmatrix} = exp \left[\begin{pmatrix} -2\gamma & \gamma & 0 & \cdots & 0 \\ \gamma & -2\gamma & \gamma & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \\ 0 & \cdots & 0 & \gamma & -2\gamma \end{pmatrix} t \right] \begin{pmatrix} T_1^0 \\ T_2^0 \\ \vdots \\ T_{N-1}^0 \end{pmatrix}$$

Dari ungkapan di atas nampak bahwa penyelesaian persamaan diferensial parsial parabolik yang diwakili oleh persamaan difusi panas dapat dinyatakan dalam skema eksplisit seperti yang disajikan pada uraian sebelumnya, ketika bentuk operasi turunan satu kali terhadap waktu didekati dengan bentuk beda hingga maju (*forward difference*). Perbedaan antara skema eksplisit tersebut dengan skema eksplisit pada pendekatan beda hingga maju adalah pada bentuk matrik pada ruas kanan.

Pada ungkapan tersebut, bentuk matrik merupakan perwujudan dari operasi eksponensial terhadap suatu matrik A, bukan operasi eksponensial terhadap suatu skalar γ. Operasi eksponensial terhadap suatu matrik biasa disebut sebagai eksponensial matrik.

0.8.8 Komputasi Eksponensial Matrik

Untuk sebarang bentuk matrik A maka komputasi untuk eksponensial matrik e^A akan membutuhkan daya komputasi tinggi (*demanding*) karena akan melibatkan perkalian matrik A yang sangat besar, seperti dapat dilihat dalam ungkapan deret berikut

$$e^A = I + A + \frac{1}{2!}A^2 + \cdots + \frac{1}{n!}A^n + \cdots$$

dengan I adalah matrik identitas.

Oleh karena itu maka beberapa langkah komputasi perlu digunakan untuk dapat mendekati nilai dari eksponensial matrik. Salah metode komputasi yang umum dilakukan adalah dengan menggunakan pendekatan *Pade* (*Pade approximation*).

Satu sifat khusus bagi eksponensial matrix apabila matrik A berbentuk matrik diagonal yaitu

$$A = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & a_{NN} \end{pmatrix}$$

maka bentuk eksponensial matrik akan sederhana berbentuk diagonal yaitu

$$e^A = \begin{pmatrix} e^{a_{11}} & 0 & \cdots & 0 \\ 0 & e^{a_{22}} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & e^{a_{NN}} \end{pmatrix}$$

Sifat ini sering dimanfaatkan untuk melakukan komputasi bagi matrik eksponensial berdasar pada prosedur diagonalisasi matrik dalam Aljabar Linear. Untuk matrik A yang berukuran tidak terlalu besar sedemikian hingga seluruh nilai eigen beserta vektor eigen dari matrik A akan dapat diperoleh, maka prosedur diagonalisasi matrik bagi matrik A akan berbentuk

$$A = UDU^{-1}$$

Matrik diagonal D berisi nilai-nilai eigen dari matrik A pada unsur-unsur diagonalnya sedang matrik U berisi vektor-vektor eigen pada tiap kolom matrik.

Apabila proses diagonalisasi berhasil, yaitu matrik D dan matrik U dapat diperoleh, maka proses komputasi matrik eksponensial e^A akan menjadi lebih sederhana yaitu berbentuk

$$e^A = Ue^DU^{-1}$$

dengan e^D akan berbentuk seperti pers (10b).

Sifat lain yang berguna bagi eksponensial matrik adalah bahwa transpose bagi matrik A^T akan terkait langsung dengan transpose bagi eksponensial matrik yaitu

$$e^{A^T} = \left(e^A\right)^T$$

Dapat ditunjukkan bahwa ungkapan yang lebih sesuai bagi ungkapan di atas adalah mengikuti apa yang disebut rumus *Baker-Campbell-Hausdorff* yaitu

$$e^A\, e^B = e^{(A+B)+\frac{1}{2}[A,B]+\frac{1}{12}[A,[A,B]]+\cdots}$$

Berdasarkan persamaan tersebut akan mudah ditunjukkan bahwa

$$e^{A+B} \neq e^A\, e^B$$

ketika matrik A dan B adalah tidak komutatif, yaitu $[A, B] \neq 0$.

Metode dekomposisi matrik *Suzuki-Trotter*

Untuk nilai parameter $\tau \approx 0$, *Trotter* menunjukkan bahwa pendekatan berikut akan berlaku

$$e^{\tau A}\, e^{\tau B} = \left[e^{\left(\frac{\tau}{n}\right)A}\, e^{\left(\frac{\tau}{n}\right)B}\right]^n = \left[e^{\left(\frac{\tau}{n}\right)(A+B)+\frac{1}{2}\left(\frac{\tau}{n}\right)^2[A,B]+\frac{1}{12}\left(\frac{\tau}{n}\right)^3[A,[A,B]]+\cdots}\right]^n \approx e^{\tau(A+B)}$$

Atau dapat ditulis pada pendekatan orde pertama sebagai

$$e^{\tau(A+B)} \approx e^{\tau A}\, e^{\tau B}$$

Suzuki memperluas pendekatan *Trotter* di atas, sehingga disebut pendekatan dekomposisi *Suzuki-Trotter*, untuk orde dua yaitu

$$e^{\tau(A+B)} \approx e^{\frac{\tau}{2}A}\, e^{\tau B}\, e^{\frac{\tau}{2}A}$$

Implementasi pendekatan dekomposisi *Suzuki-Trotter* untuk penyelesaian persamaan difusi panas dapat dilakukan dengan meninjau bahwa proses difusi pada seluruh waktu t akan dilakukan secara diskret pada tiap selang waktu yang sangat kecil yaitu sebesar $\Delta t = \tau$ dengan mengambil bahwa

$$\tau = \Delta t = \frac{t_N - t_0}{N}$$

Oleh karena itu sebarang waktu akan dapat dinyatakan sebagai

$$t \approx t_N = t_{N-1} + \Delta t = t_{N-2} + \Delta t + \Delta t = \cdots = t_0 + N\Delta t = t_0 + N\tau$$

Penyelesaian persamaan difusi panas akan dilakukan secara bertahap selama waktu kecil τ berdasar persamaan tersebut yaitu

$$T(x, \tau) = e^{\tau A} T(x, 0)$$

Perhitungan temperatur selama τ dapat memanfaatkan pendekatan dekomposisi *Suzuki-Trotter* orde satu atau orde dua berdasar dua persamaan yang disinggung sebelumnya.

Sebagai contoh, apabila menggunakan pendekatan orde satu dan matrik A akan dipisah berdasar sokongan dua matrik yang lebih sederhana yaitumatrik A_1 dan matrik A_2 maka persamaan tersebut berbentuk

$$T(x, \tau) = e^{\tau A} T(x, 0) = e^{\tau(A_1+A_2)} T(x, 0) \approx e^{\tau A_1} e^{\tau A_2} T(x, 0)$$

Pada umumnya, operasi matrik yang terlibat pada komputasi eksponensial matrik tanpa dekomposisi akan membutuhkan ongkos komputasi tingi (*demanding*) dibanding operasi matrik yang melibatkan komputasi eksponensial matrik dekomposisi.

Implementasi metode dekomposisi eksponensial matrik untuk persamaan difusi panas

Ditinjau keadaan ketika parameter τ adalah kecil yaitu $\tau = \Delta t = \frac{t_N - t_0}{N}$. Untuk bentuk matrik A seperti dalam persamaan difusi panas maka ada beberapa tahap dekomposisi yang dapat dilakukan.

- **Tahat dekomposisi pertama**

$$\underbrace{\begin{pmatrix} -2\gamma & \gamma & 0 & \cdots & 0 \\ \gamma & -2\gamma & \gamma & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \\ 0 & \cdots & 0 & \gamma & -2\gamma \end{pmatrix}}_{A} = \underbrace{\begin{pmatrix} -2\gamma & 0 & 0 & \cdots & 0 \\ 0 & -2\gamma & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \\ 0 & \cdots & 0 & 0 & -2\gamma \end{pmatrix}}_{A_1} + \underbrace{\begin{pmatrix} 0 & \gamma & 0 & \cdots & 0 \\ \gamma & 0 & \gamma & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \\ 0 & \cdots & 0 & \gamma & 0 \end{pmatrix}}_{A_2}$$

Tahap dekomposisi pertama dilakukan dengan beberapa pertimbangan yaitu

- Matrik A_1 dan A_2 bersifat komutatif yaitu $[A_1, A_2] = A_1A_2 - A_2A_1 = 0$.
- Karena bersifat komutatif maka berlaku kaitan berikut

$$e^A = e^{(A_1+A_2)} = e^{A_1}e^{A_2}$$

- Matrik A_1 berbentuk diagonal sehingga bentuk eksponensial matrik menjadi sederhana.

$$e^{A_1} = \begin{pmatrix} e^{-2\gamma} & 0 & \cdots & 0 \\ 0 & e^{-2\gamma} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & e^{-2\gamma} \end{pmatrix}$$

- **Tahap dekomposisi kedua**

$$\underbrace{\begin{pmatrix} 0 & \gamma & 0 & 0 & \cdots & 0 \\ \gamma & 0 & \gamma & 0 & \cdots & 0 \\ 0 & \gamma & 0 & \gamma & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \cdots & \vdots \\ 0 & \cdots & 0 & \gamma & 0 & \gamma \\ 0 & \cdots & 0 & 0 & \gamma & 0 \end{pmatrix}}_{A_2} = \underbrace{\begin{pmatrix} 0 & \gamma & 0 & 0 & \cdots & 0 \\ 0 & 0 & \gamma & 0 & \cdots & 0 \\ 0 & 0 & 0 & \gamma & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \cdots & \vdots \\ 0 & \cdots & 0 & 0 & 0 & \gamma \\ 0 & \cdots & 0 & 0 & 0 & 0 \end{pmatrix}}_{A_3} + \underbrace{\begin{pmatrix} 0 & 0 & 0 & 0 & \cdots & 0 \\ \gamma & 0 & 0 & 0 & \cdots & 0 \\ 0 & \gamma & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \cdots & \vdots \\ 0 & \cdots & 0 & \gamma & 0 & 0 \\ 0 & \cdots & 0 & 0 & \gamma & 0 \end{pmatrix}}_{A_4}$$

Upaya untuk memperoleh eksponensial matrik bagi A_2 dengan melakukan dekomposisi terlebih dahulu bagi matrik $A_2 menjadi penjumlahan matrik A\_3 dan A\_4 seperti di atas adalah dalam upaya agar proses penc$ menjadi lebih sederhana. Penjelasan untuk itu dapat diberikan melalui gambaran berikut.

Bentuk matrik seperti A_3 atau A_4 disebut matrik *nilpotent*. Berdasar definisi yang disebut matrik *nilpotent* adalah suatu matrik A sedemikian hingga memenuhi A^k untuk k merupakan bilangan bulat tertentu. Sebagai gambaran, tinjau matrik A_3 dengan orde 4×4. Maka dapat ditunjukkan operasi matrik berikut:

$$A_3^2 = \begin{pmatrix} 0 & \gamma & 0 & 0 \\ 0 & 0 & \gamma & 0 \\ 0 & 0 & 0 & \gamma \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & \gamma & 0 & 0 \\ 0 & 0 & \gamma & 0 \\ 0 & 0 & 0 & \gamma \\ 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & \gamma^2 & 0 \\ 0 & 0 & 0 & \gamma^2 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$A_3^3 = \begin{pmatrix} 0 & 0 & \gamma^2 & 0 \\ 0 & 0 & 0 & \gamma^2 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & \gamma & 0 & 0 \\ 0 & 0 & \gamma & 0 \\ 0 & 0 & 0 & \gamma \\ 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & \gamma^3 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$A_3^4 = \begin{pmatrix} 0 & 0 & 0 & \gamma^3 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & \gamma & 0 & 0 \\ 0 & 0 & \gamma & 0 \\ 0 & 0 & 0 & \gamma \\ 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Dengan bentuk matrik *nilpoten* A_3 berorde 4 seperti di atas maka ungkapan eksponensial matrik bagi A_3 menjadi mudah untuk diperoleh yaitu

$$\begin{aligned} e^{A_3} &= I + A_3 + \frac{1}{2}A_3^2 + \frac{1}{3!}A_3^3 + \frac{1}{4!}A_3^4 + 0 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 0 & \gamma & 0 & 0 \\ 0 & 0 & \gamma & 0 \\ 0 & 0 & 0 & \gamma \\ 0 & 0 & 0 & 0 \end{pmatrix} \\ &+ \frac{1}{2}\begin{pmatrix} 0 & 0 & \gamma^2 & 0 \\ 0 & 0 & 0 & \gamma^2 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} + \frac{1}{3!}\begin{pmatrix} 0 & 0 & 0 & \gamma^3 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} + \frac{1}{4!}\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} + 0 \\ &= \begin{pmatrix} 1 & \gamma & \frac{1}{2}\gamma^2 & \frac{1}{6}\gamma^3 \\ 0 & 1 & \gamma & \frac{1}{2}\gamma^2 \\ 0 & 0 & 1 & \gamma \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned}$$

Karena A_4 merupakan transpose dari A_3 yaitu memenuhi kaitan $A_4 = A_3^T$ maka proses untuk memperoleh eksponensial matrik bagi A_4 menjadi sederhana yaitu

$$e^{A_4} = e^{A_3^T} = \left(e^{A_3}\right)^T = \begin{pmatrix} 1 & \gamma & \frac{1}{2}\gamma^2 & \frac{1}{6}\gamma^3 \\ 0 & 1 & \gamma & \frac{1}{2}\gamma^2 \\ 0 & 0 & 1 & \gamma \\ 0 & 0 & 0 & 1 \end{pmatrix}^T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ \gamma & 1 & 0 & 0 \\ \frac{1}{2}\gamma^2 & \gamma & 1 & 0 \\ \frac{1}{6}\gamma^3 & \frac{1}{2}\gamma^2 & \gamma & 1 \end{pmatrix}$$

Dengan contoh tersebut maka prosedur untuk komputasi temperatur $T(x,\tau)$ berdasar eksponensial matrik dengan pendekatan Suzuki-Trotter orde 1 dilakukan dengan ungkapan berikut.

$$\begin{aligned} T(x,\tau) &= e^{\tau A}T(x,0) = e^{\tau(A_1+A_2)}T(x,0) = e^{\tau A_1}e^{\tau A_2}T(x,0) \\ &= e^{\tau A_1}e^{\tau(A_3+A_4)}T(x,0) \approx e^{\tau A_1}e^{\tau A_3}e^{\tau A_4}T(x,0) \end{aligned}$$

Implementasi bagi prosedur di atas dapat dilakukan dengan *source-code* berikut.

```
using LinearAlgebra
using Plots
using LaTeXStrings

function exp_nilpotent(b)
    n = length(b)
```

```
    A = zeros(n+1, n+1)
    A[end, end] = 1.0
    for i in 1:n
        A[i, i] = 1.0
        l = 0
        for j in i+1:n+1
            l +=1
            A[i, j] = A[i,j -1]*b[i]/l
        end
    end
    return A
end
```

```
exp_nilpotent (generic function with 1 method)
```

```
Nx = 100
Nt = 100
D = 1.0

x = range( -5, 5, length = Nx)
t = range(0, 10, length = Nt)

dx = x[2] - x[1]
dt = t[2] - t[1]

gamma = D / dx^2
tau = dt
lambda = gamma * tau

temper = zeros(Float64, Nx, Nt)
temper[:, 1] = exp.( -x.^2)
temper[1, :] .= 0.0
temper[end, :] .= 0.0;
```

Pendekatan Suzuki-Trotter orde 1

```
b = lambda * ones(Nx -3);
```

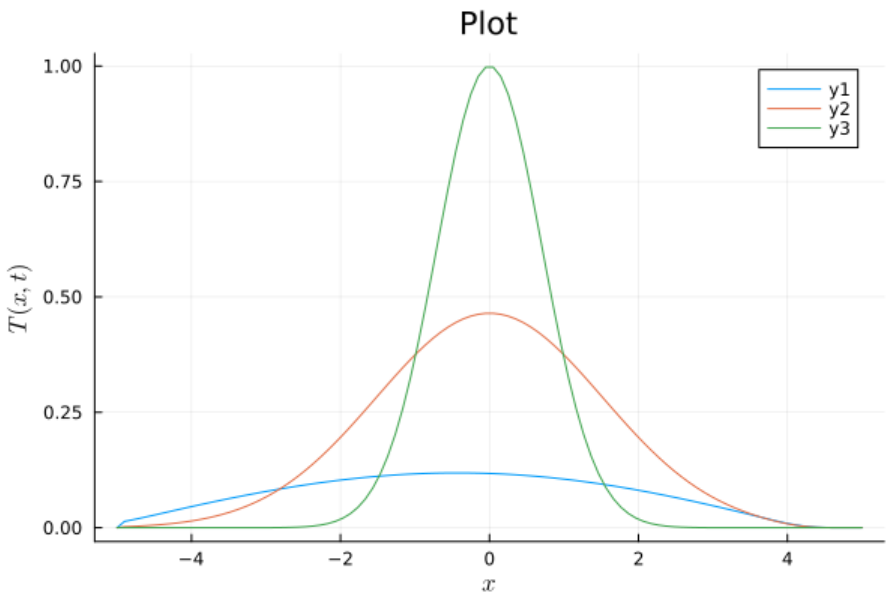
```
A3 = exp_nilpotent(b);
```

```
function skema_ExpMat1(A3, temper, lambda, Nt, Nx)
    A4 = transpose(A3)
    d = exp( -2.0 * lambda) * ones(Nx -2)
    A1 = diagm(d)
    evolusi = zeros(Nx -2, Nx -2)
    evolusi = A1 * A3 * A4
    for i in 2:Nt
        temper[2:Nx -1,i] =  evolusi * temper[2:Nx -1,i -1]
    end

    return temper
end
```

```
skema_ExpMat1 (generic function with 1 method)
```

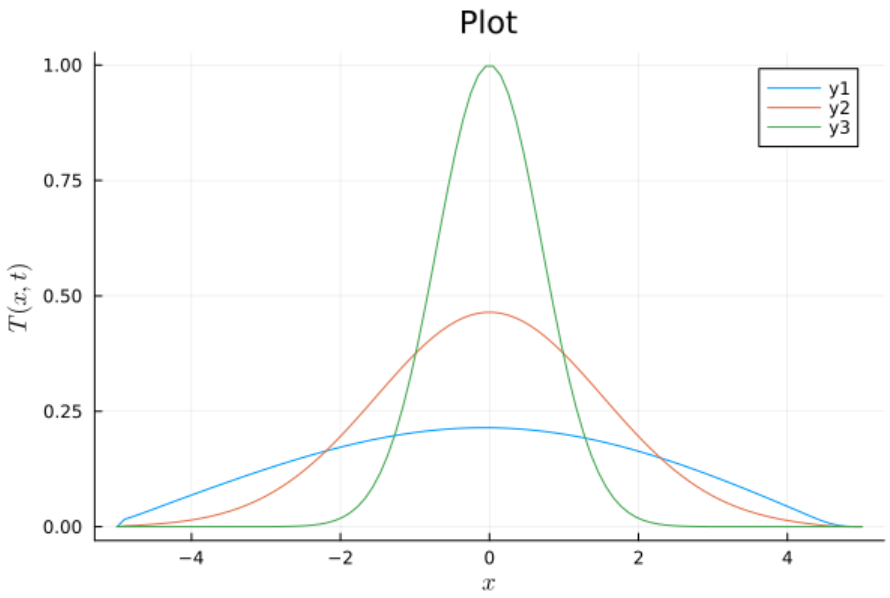
```
temper1 = skema_ExpMat1(A3, temper, lambda, Nt, Nx);
```

```
plot(x, [temper1[:, 100],temper1[:, 10], temper1[:, 1]],
     xlabel=L"$x$",
     ylabel=L"$T(x,t)$",
     title="Plot")
```

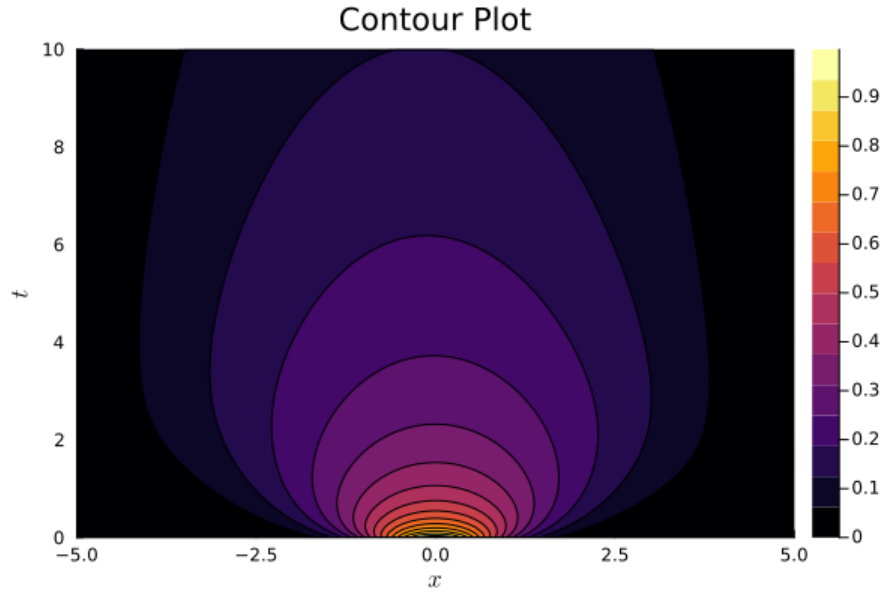


Pendekatan Suzuki-Trotter orde 2

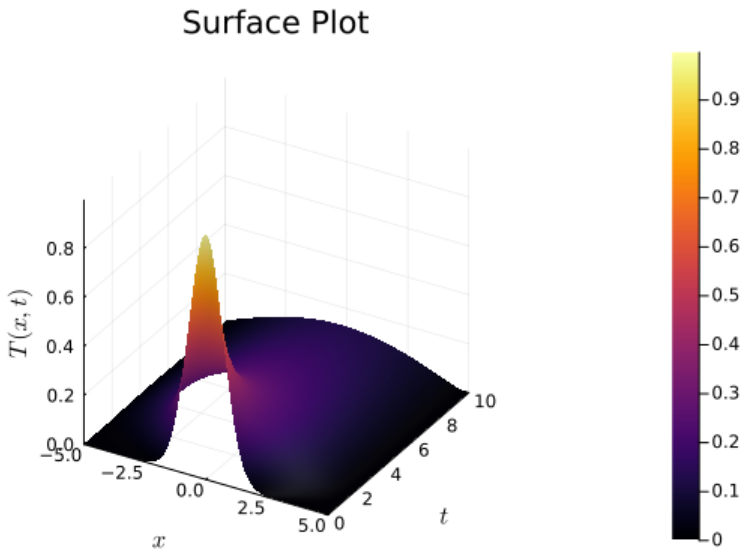
```
b = 0.5 * lambda * ones(Nx -3);

A3 = exp_nilpotent(b);

function skema_ExpMat2(A3, temper, lambda, Nt, Nx)
    A4 = transpose(A3)
    d = exp( -2.0 * lambda) * ones(Nx -2)
    A1 = diagm(d)
    evolusi = zeros(Nx -2, Nx -2)
    evolusi = A1 * transpose(A3 * A4) * A3 * A4
    for i in 2:Nt
        temper[2:Nx -1,i] =  evolusi * temper[2:Nx -1,i -1]
    end

    return temper
end

skema_ExpMat2 (generic function with 1 method)

temper2 = skema_ExpMat2(A3, temper, lambda, Nt, Nx);

plot(x, [temper2[:, 50],temper2[:, 10], temper2[:, 1]],
     xlabel=L"$x$",
     ylabel=L"$T(x,t)$",
     title="Plot")
```




```
contour(x, t, temper',
        xlabel=L"$x$",
        ylabel=L"$t$",
        title="Contour Plot",
        fill=true)
```




```
surface(x, t, temper',
        xlabel=L"$x$",
        ylabel=L"$t$",
        zlabel=L"$T(x,t)$",
        title="Surface Plot")
```


Surface Plot
$T(x,t)$
0.0
0.2
0.4
0.6
0.8
−5.0
−2.5
0.0
2.5
5.0
x
0
2
4
6
8
10
t
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9