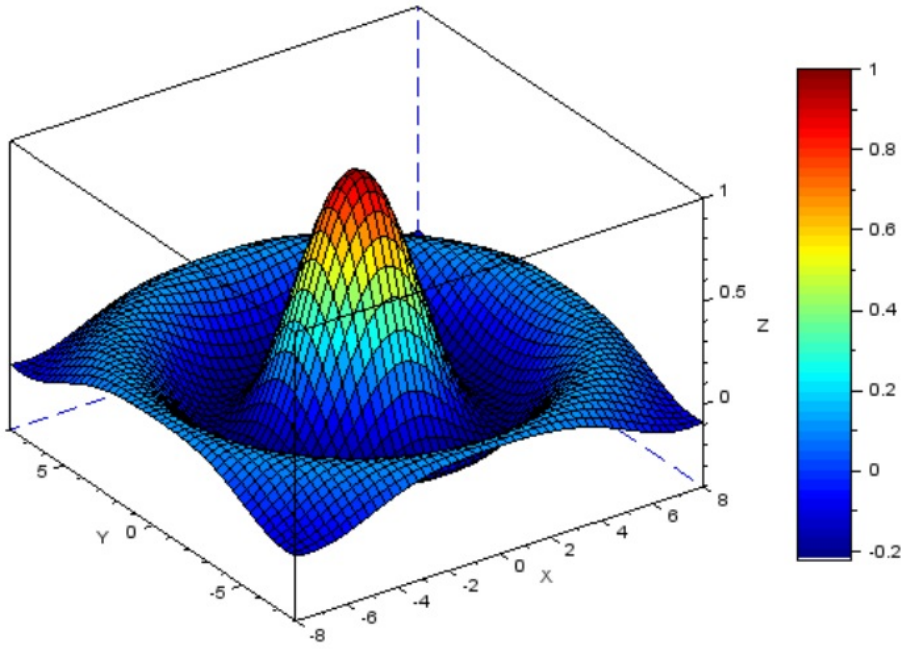
Optimisasi Swarm Partikel Menggunakan Scilab

by Windarto Windarto

Submission date: 15-Jul-2020 11:21AM (UTC+0800)
Submission ID: 1357677487
File name: Buku-Optimisasi-Swarm-Partikel-menggunakan-Scilab.docx (561.07K)
Word count: 28150
Character count: 173685

Optimisasi Swarm Partikel menggunakan Scilab



Penyusun

Windarto

9

"Sesungguhnya dalam penciptaan langit dan bumi dan pergantian malam dan siang, terdapat tanda-tanda (kekuasaan Allah) bagi *ulul-albab*. (Yaitu) orang-orang yang mengingat Allah sambal berdiri, duduk atau dalam keadaan berbaring, dan mereka memikirkan tentang penciptaan langit dan bumi (seraya berkata): Ya Tuhan kami, tidaklah Engkau menciptakan semua ini sia-sia, Maha Suci Engkau, dan lindungilah kami dari azab neraka" (Terjemah Al Quran surat Ali Imran: 191 – 192).

Kata Pengantar

Segala puji syukur bagi Alloh, Tuhan Sang Pencipta alam semesta. Penulis memanjatkan syukur kepada-Nya. Atas rahmat dan karunia-Nya semata, Penulis dapat menyelesaikan buku ini. Shalawat dan salam semoga tercurah kepada Nabi Muhammad dan *ahlul bait* (keluarga) beliau.

Optimisasi merupakan salah satu permasalahan yang lazim ditemui dalam kehidupan sehari-hari, terutama dalam dunia riset (penelitian). Permasalahan optimisasi, memuat satu atau beberapa fungsi tujuan (*objective function*) yang diperlu dioptimalkan (dimaksimumkan atau diminimumkan) nilainya. Pada permasalahan optimasi dengan lebih dari satu fungsi tujuan, peneliti perlu melakukan transformasi agar terbentuk satu fungsi tujuan dari sejumlah fungsi tujuan sebelumnya. Selain itu, peneliti juga perlu menentukan nilai satu atau lebih dari satu nilai variabel keputusan sehingga fungsi tujuan mencapai nilai optimum. Optimisasi swarm partikel (*particle swarm optimization*) merupakan salah satu metode optimasi yang terinspirasi dari perilaku swarm (kelompok/gerombolan) hewan seperti ikan dan burung. Salah satu keunggulan metode optimisasi swarm parameter adalah bahwa metode ini relatif sederhana. Metode optimisasi swarm partikel sangat sulit terimplementasi dengan baik tanpa menggunakan suatu perangkat lunak (program) computer.

Buku ini membahas implementasi metode optimisasi swarm partikel menggunakan perangkat lunak Scilab. Scilab merupakan perangkat lunak yang menunjang komputasi teknis. Buku ini terbagi menjadi dua bagian. Bagian pertama (Bab 1 – Bab 6) menjelaskan tentang Bahasa pemrograman pada perangkat lunak Scilab. Bagian kedua (Bab 7 – Bab 12) membahas tentang metode optimisasi swarm partikel. Prosedur optimasasi swarm partikel dibahasa pada Bab 8. Implementasi swarm partikel pada masalah optimasi sederhana (penentuan akar suatu persamaan non linear, penentuan akar suatu system persamaan non linear, dan penentuan nilai maksimum fungsi dua variabel), estimasi parameter pada model pertumbuhan dan estimasi parameter pada model matematika mangsa-pemangsa berturut-turut disajikan pada Bab 9 – Bab 11. Bab 12 membahas beberapa kesimpulan penting dalam implementasi metode optimisasi swarm partikel.

Sebagian materi dalam buku ini merupakan sebagian hasil publikasi ilmiah Penulis pada tahun 2018 – 2019. Penulis menyampaikan ucapan terima kasih kepada:

(1) Direktorat Riset dan Pengabdian Masyarakat (DRPM), Kementerian Riset Teknologi dan Pendidikan Tinggi (Kementerian Riset Teknologi dan BRIN) yang telah memberikan pendanaan penelitian kepada Penulis.
(2) Kolega dosen/peneliti di lingkungan Universitas Airlangga (terutama dosen/peneliti di lingkungan Departemen Matematika, Fakultas Sains dan Teknologi Universitas Airlangga) atas kebersamaan dan rasa kekeluargaan yang tinggi dalam pelaksanaan kegiatan pendidikan/pengajaran, penelitian dan pengabdian masyarakat.
(3) Penulis juga menyampaikan terima kasih kepada isteri dan putera-puteri Penulis atas dukungan yang diberikan kepada Penulis selama ini.
(4) Semua pihak yang memberikan dukungan atas tersusunya dan terbitnya buku ini.

Jazaakumullahu khairan katsiran. Semoga Alloh memberikan balasan kebaikan yang berlipat ganda.

Semoga buku ini memberikan manfaat kepada para peneliti terutama untuk mengimplementasikan metode optimisasi swarm penelitian mereka. Selain itu, semoga buku ini memberikan kontibusi positif dalam pengembangan ilmu pengetahuan di Indonesia.

Surabaya, 17 Mei 2020

Penulis

Daftar Isi

1. Pengantar Scilab

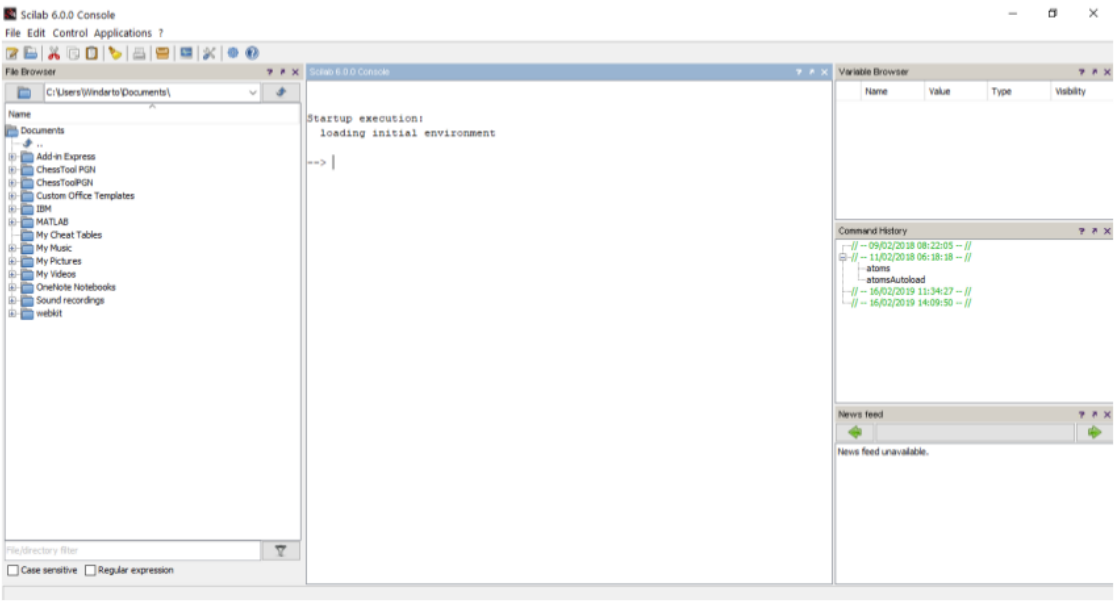
1.1. Lingkungan kerja Scilab

Scilab merupakan bahasa pemrograman Scilab yang memuat banyak koleksi algoritma perhitungan numerik. Dari sudut pandang perangkat lunak, Scilab merupakan Bahasa pemrograman jenis *interpreter*, sehingga pengguna Scilab dapat menggunakan perintah-perintah internal yang tersedia dalam Scilab untuk keperluan komputasi. Pengguna Scilab juga dapat mengembangkan modul/function/sub program yang memungkinkan pengguna Scilab dapat memecahkan permasalahan mereka yang berhubungan dengan komputasi. Scilab merupakan perangkat lunak tidak berbayar (*free software*), sehingga pengguna Scilab tidak perlu membayar untuk mendapatkan lisensi perangkat lunak Scilab. Scilab dapat dijalankan pada system operasi Windows, Linux dan Mac. Scilab juga menyediakan banyak fitur penyajian grafik, termasuk sekumpulan perintah untuk plot grafik fungsi, plot 2D dan plot 3D [1, 2, 3].

File penginstal Scilab (*Scilab binary*) tersedia dalam versi 32-bit dan 64-bit. File penginstal Scilab dapat diunduh pada laman

http://www.scilab.org/download

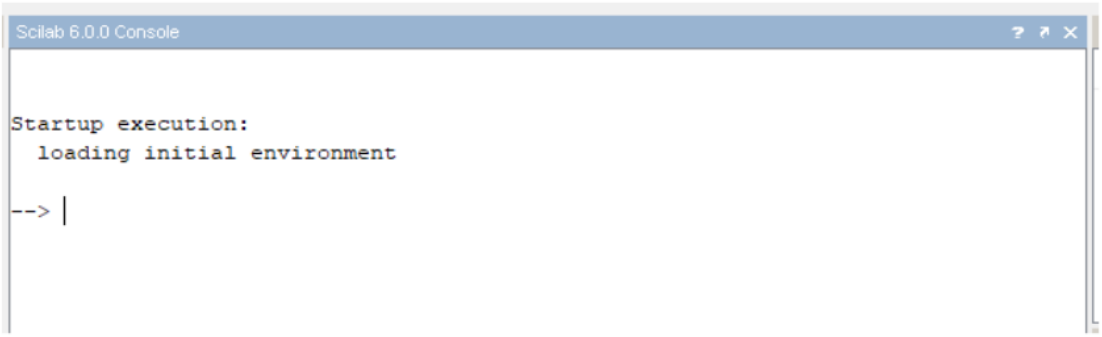
Setelah perangkat lunak Scilab terinstal pada komputer, Scilab dapat dijalankan dengan melakukan klik ikon Scilab pada layar desktop. Lingkungan kerja Scilab disajikan pada Gambar 1.1.



Gambar 1.1. Tampilan lingkungan kerja Scilab

Lingkungan kerja Scilab terdiri dari lima jendela (*window*), yaitu *Scilab Console*, jendela *File Browser* (*File Browser window*), jendela *Variable Browser*, jendela *Command History*, dan jendela *News Feed*. Jendela perintah merupakan tempat untuk menjalankan perintah-perintah pada

perangkat lunak Scilab secara interaktif. Jendela *Scilab Console* dilengkapi dengan prompt Scilab yang berbentuk tanda panah (–—>).

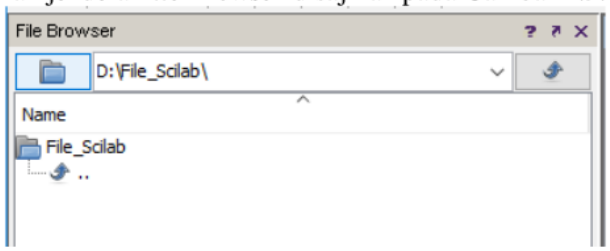


Gambar 1.2. Tampilan jendela *Scilab Console* pada perangkat lunak Scilab

Perintah dapat diketikkan pada *Scilab Console*. Setiap perintah pada jendela *Scilab Console* diakhiri dengan penekanan tombol *Enter*.

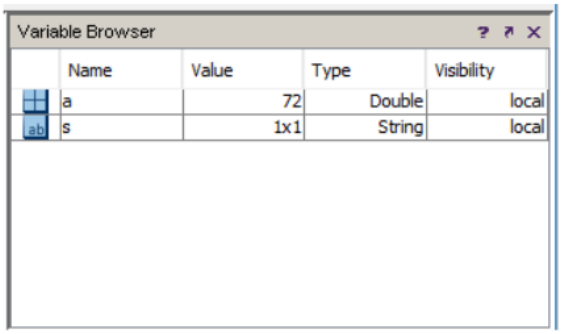
```
–—> a = 2 + 7*10
 a  =
   72.
–—> s = 'Perangkat lunak Scilab'
 s  =
 Perangkat lunak Scilab
```

Jendela *File Browser* terletak pada bagian kiri lingkungan kerja Scilab. Pada platform Windows, folder *C:\Users\[nama-user]\Documents* merupakan folder aktif *default* (bawaan). Misalkan folder aktif akan dipindahkan ke folder D:\File_Scilab. Klik tombol *Select Folder*, yaitu ikon bertanda ( ) pada bagian kiri atas jendela *File Browser*. Selanjutnya, dapat dipilih folder D:\File_Scilab. Tampilan jendela *File Browser* disajikan pada Gambar 1.3.



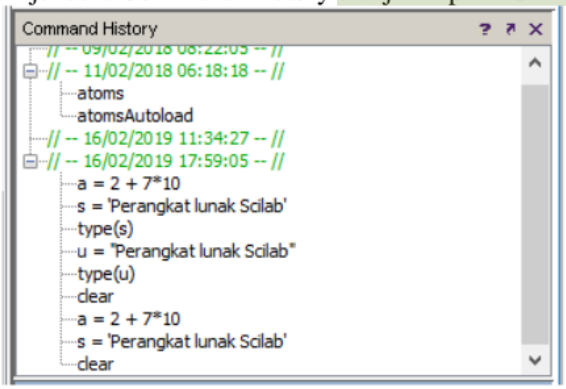
Gambar 1.3. Tampilan jendela *File Browser* pada perangkat lunak Scilab

Jendala *Variable Browser* terletak pada bagian kanan atas pada lingkungan kerja Scilab. Pada jendela Variable Browser, disajikan informasi variabel-variabel yang aktif pada lingkungan kerja Scilab. Ada empat atribut variable yang ditampilkan pada jendela Variable Browser, yaitu atribut *Name* (nama variabel), *Value* (nilai variabel), *Type* (tipe variabel) dan atribut *Visibility*. pada Tampilan jendela *Variable Browser* disajikan pada Gambar 1.4.



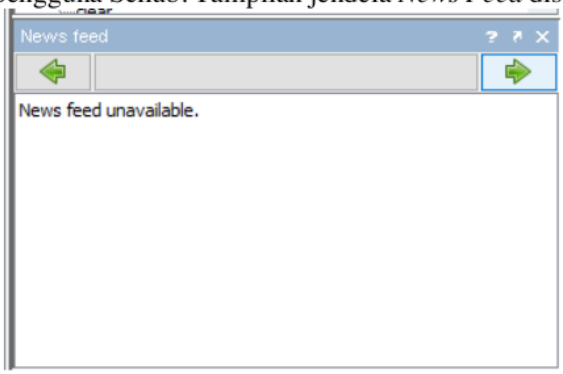
Gambar 1.4. Tampilan jendela *Variable Browser* pada perangkat lunak Scilab

Jendela *Command History* terletak pada bagian kanan tengah pada lingkungan kerja Scilab. Pada jendela *Command History*, disajikan informasi perintah-perintah yang diberikan pada jendela *Scilab Console*. Tampilan jendela Command History disajikan pada Gambar 1.5.



Gambar 1.5. Tampilan jendela *Command History* pada perangkat lunak Scilab

Jendela *News Feed* terletak pada bagian kanan bawah pada lingkungan kerja Scilab. Pada jendela *News Feed*, ditampilkan informasi yang terkait dengan berita, tips pemrograman dan komunikasi umum pada komunitas pengguna Scilab. Tampilan jendela *News Feed* disajikan pada Gambar 1.6.



Gambar 1.6. Tampilan jendela *News Feed* pada perangkat lunak Scilab

1.2. Akses menu bantuan pada Scilab

Perintah *help* dapat digunakan untuk memperoleh informasi terkait penjelasan suatu sintak perintah pada perangkat lunak Scilab. Sebagai contoh

```
--> help sqrt
```

digunakan untuk memperoleh informasi tentang deskripsi perintah *sqrt*. Selanjutnya, akan terbuka jendela *Help Browser* yang menyajikan informasi tentang perintah *sqrt*. Jendela *Help Browser* dapat ditutup dengan menekan tombol (x) pada bagian kanan atas.

Menu bantuan juga dapat diaktifkan dengan melakukan klik pada tanda ? pada bagian baris menu. Selanjutnya dapat dipilih sub menu *Scilab Help*, sehingga terbuka jendela *Help Browser*. Jendela *Help Browser* juga dapat diaktifkan dengan menekan tombol F1.

1.3. Menutup perangkat lunak Scilab

Berikut adalah beberapa teknik yang dapat digunakan untuk menutup perangkat lunak Scilab, yaitu:

(a) Pilih menu *File*, lalu sub menu *Quit*.
(b) Klik tombol (x) pada bagian kanan atas lingkungan kerja Scilab.
(c) Tekan tombol *Ctrl Q*, yaitu penekanan tombol *Ctrl* dan tombol *Q* secara bersamaan.
(d) Mengetikkan perintah quit() pada Scilab console.

```
--> quit( )
```

2. Mulai Menjalankan Scilab

2.1. Menjalankan Scilab melalui Scilab Console

Perhitungan aritmetika sederhana dapat dilakukan dengan mengetikkan perintah pada *Scilab Console*. Perlu diingat bahwa setiap perintah pada *Scilab Console*, diakhiri dengan menekan tombol *Enter*. Operator aritmetika disajikan pada Tabel 2.1.

Tabel 2.1. Operator aritmetika pada perangkat lunak Scilab

Operator	Deskripsi	Contoh penggunaan
^	Perpangkatan	2^5
**	Perpangkatan	2**5
*	Perkalian	2 * 5
/	Pembagian dari kanan	$2/5 = 2*5^{-1}$
\	Pembagian dari kanan	$2\backslash5 = 2^{-1}*5$
+	Penjumlahan	2 + 5
–	Pengurangan	2 – 5

Prioritas urutan pengerjakan operator aritmetika tersebut adalah:

(a) Perpangkatan mempunyai prioritas tertinggi dibandingkan keempat operator lainnya.
(b) Operator perkalian/pembagian mempunyai prioritas lebih tinggi dibandingkan operator penjumlahan/pengurangan.
(c) Operator perkalian mempunyai derajat prioritas yang sama dengan operator pembagian.
(d) Operator penjumlahan mempunyai derajat prioritas yang sama dengan operator pengurangan.

Untuk mengubah prioritas urutan pengerjaan operator dapat digunakan tanda kurung. Berikut diberikan beberapa contoh sederhana.

```
--> 5 + 2^2
ans  =
   9.
--> b = (5+2)^2
 b  =
   49.
--> e = 2/5
 e  =
   0.4
--> f = 2\5
 f  =
   2.5
```

Hasil operasi yang tidak disimpan dalam suatu variabel, akan tersimpan sebagai variable *ans*.

Bila suatu perintah diakhiri dengan tanda semicolon (;), maka output perintah tersebut tidak ditampilkan pada *Scilab Console*.

```
--> c = 3 + 4*5;
```

```
--> disp(c);
  23.
--> d = (3 + 4)*5;
--> disp(d);
  35.
```

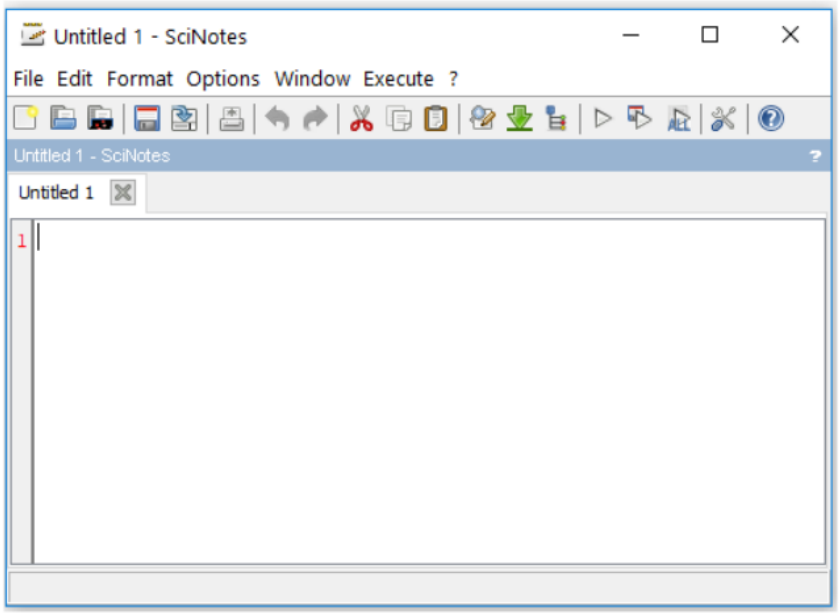
Perintah disp(c) digunakan untuk menampilkan nilai variabel c.

2.2. Menjalankan Scilab melalui Scilab editor

Sejumlah perintah juga dapat dituliskan dalam satu file berekstensi sce, yaitu file yang dibangun dari *Scilab editor*. Pada jendela Scilab editor, kode program dapat diedit dengan mudah dan leluasa. Jendela *Scilab editor* dapat diakses dari menu *Applications – SciNotes*. Selain itu, jendela *Scilab editor* juga dapat diakses dari jendela *Scilab console* dengan mengetikkan perintah

```
--> editor()
```

Tampilan jendela Scilab editor disajikan pada Gambar 2.1.



Gambar 2.1. Tampilan jendela SciNotes (Scilab editor)

Setelah jendela Scilab editor terbuka, ketikkanlah kode program **Volume_kubus.sce** berikut.

Kode Program 2.1. Volume_kubus.sce

```
\\Kode Program 2.1. Volume kubus
pjg = 5;
s1 = ["Panjang kubus = " , sci2exp(pjg) , "cm"] ;
disp(s1);
lbr = 4;
```

```
s2 = ["Lebar kubus = " , sci2exp(lbr) , "cm"] ;
disp(s2);
tg = 3;
s3 = ["Tinggi kubus = " , string(tg) , "cm"] ;
disp(s3);
Volume = pjg*lbr*tg;
s4 = ["Volume kubus = " , string(Volume) , "cm^3"] ;
disp(s4);
\\Akhir kode program
```

Simpanlah file kode program tersebut dengan nama Volume_kubus.sce. Pada kode program tersebut, perintah **sci2exp** dan **string** digunakan untuk mengubah variable tipe numerik menjadi tipe string. File bertipe *SciNotes* dapat dijalankan (dieksekusi) memilih menu *Execute – Save and execute* atau menekan tombol F5 pada system operasi Windows, pada jendela *SciNotes* (Scilab editor). Pada jendela *Scilab console* diperoleh output sebagai berikut.

```
--> exec('D:\File_Scilab\Volume_kubus.sce', -1)
!Panjang kubus =   5  cm  !
!Lebar kubus =   4  cm  !
!Tinggi kubus =   3  cm  !
!Volume kubus =   60  cm^3  !
```

Pada saat memilih menu Execute pada baris menu, ada juga pilihan Execute – Save and execute all files atau menekan tombol Ctrl + F5 pada system operasi Windows. Perintah ini digunakan untuk menjalankan (mengeksekusi) semua file kode program yang sedang aktif (terbuka) pada jendela Scilab editor. Selain itu ada tiga pilihan lain yang tersedia pada menu Execute, yaitu [2]:

(a) Execute - … **file with no echo** atau menekan tombol Ctrl + Shift + E pada sistem operasi Windows. Dengan menggunakan perintah ini, file akan dijalankan tanpa menuliskan kode program pada jendela *Scilab console*.

(b) Execute - … **file with echo** atau menekan tombol Ctrl + L pada sistem operasi Windows. Dengan menggunakan perintah ini, file akan dijalankan dengan menuliskan kode program pada jendela *Scilab console*.

(c) **Execute - … until the caret**, with echo atau menekan tombol Ctrl + E pada sistem operasi Windows. Dengan menggunakan perintah ini, bagian file yang terseleksi akan dijalankan dengan menuliskan kode program pada jendela *Scilab console*.

3. Elemen Dasar Bahasa Pemrograman Scilab

Pada bagian ini akan disajikan beberapa tipe data dalam perangkat lunak Scilab, meliputi tipe data real, kompleks, boolean dan string. Dalam bahasa pemrograman Scilab, suatu variable tipe real, kompleks, boolean, dan tipe string *dapat dipandang sebagai suatu matriks*. Penamaan variable dalam perangkat lunak Scilab perlu memperhatikan hal-hal berikut:

(a) Variabel dalam Scilab bersifat *case sensitive* (memperhatikan huruf kecil dan huruf capital). Variabel "nilai" dan "Nilai" merupakan dua variable berbeda dalam Scilab.
(b) Nama variable sebaiknya tidak lebih dari 24 karakter, mengingat hanya 24 karakter awal yang dipandang sebagai nama variable dalam perangkat lunak Scilab. Sebagai contoh, variable "PeubahInteger0123456789" dan "PeubahInteger0123456789123".
(c) Nama variable sebaiknya tidak diawali dengan karakter "%".

Pada Bahasa pemrograman Scilab, ada beberapa variabel terdefinisi khusus yang diawali dengan karakter % (pre-defined variable). Beberapa variabel terdefinisi khusus pada Scilab disajikan pada Tabel 3.1.

Tabel 3.1. Beberapa variabel terdefinisi khusus pada perangkat lunak Scilab

Operator	Deskripsi	Contoh penggunaan
%i	Bilangan imaginer $i = \sqrt{-1}$	–-> x1 = 1 + %i
%e	Bilangan natural e	–-> x2 = %e^2
%pi	Bilangan π	–-> x3 = sin(%pi/4)
%T	Nilai True pada variabel tipe boolean	–-> x4 = (2 <= 4)
%t	Nilai True pada variabel tipe boolean	–-> x5 = (2 < 4)
%F	Nilai False pada variabel tipe boolean	–-> x6 = (2 >= 4)
%f	Nilai False pada variabel tipe boolean	–-> x7 = (2 >= 4)

3.1. Variabel tipe real dan kompleks

Scilab merupakan Bahasa pemrograman tipe interpreter, sehingga tidak diperlukan pendefinisian variable yang akan digunakan dalam pemrograman. Variabel dapat langsung diisi dengan nilai variable. Sebagai contoh, dengan memberikan perintah

```
–-> x = 5
 x  =
   5.
```

pengguna Scilab telah mendefinisikan x sebagai variable tipe real dan memberikan nilai 5 pada variabel x. Dengan memberikan perintah

```
–-> x = 2*x + 1
 x  =
  11.
```

pengguna Scilab telah melakukan update nilai variable x menjadi 11.

Variabel %i merupakan variable Scilab, dengan %i $= \sqrt{-1}$. Dengan menggunakan variable %i tersebut, pendefinisian variable tipe kompleks dapat dilakukan dengan mudah.

```
--> y1 = 1 + 2*%i
 y1  =
   1. + 2.i
--> y2 = 1 - 2*%i
 y2  =
   1. - 2.i
--> y1*y2
 ans  =
   5.
--> y1+y2
 ans  =
   2.
```

Operator untuk tipe data real sebagaimana disajikan pada Tabel 2.1, juga dapat digunakan pada tipe data kompleks. Operator ‘ (tanda petik) dapat digunakan untuk memperoleh transpos konjugat dari suatu variabel tipe real, kompleks, atau tipe integer.

```
--> y1'
 ans  =
   1. - 2.i
--> y1' + y1
 ans  =
   2.
--> y1' - y2
 ans  =
   0.
```

3.2. Variabel tipe Boolean

Variabel tipe Boolean merupakan variabel yang hanya dapat bernilai “Benar (True)” atau “Salah (False)”. Pada perangkat lunak Scilab, nilai True dinyatakan dengan %t atau %T, sedangkan nilai False dinyatakan dengan %f atau %F. Berikut diberikan beberapa contoh.

```
--> p1 = %t
 p1  =
  T
--> p2 = %f
 p2  =
  F
--> p3 = (2 > 2)
 p3  =
  F
--> p4 = (2 <= 2)
 p4  =
  T
```

Operator logika dapat diterapkan pada variabel tipe Boolean. Dua ekspresi matematis juga dapat dikenakan operator perbandingan sehingga diperoleh variabel tipe Boolean. Operator logika dan operator perbandingan disajikan pada Tabel 3.2.

Tabel 3.2. Operator logika dan operator perbandingan pada perangkat lunak Scilab

Operator	Deskripsi	Contoh penggunaan
&	Operator konjungsi	–-> p5 = (2 <= 3) & (4 > 2)
|	Operator disjungsi	–-> p6 = (2 <= 3) | (4 > 2)
~	Operator negasi	–-> p7 = ~(2 >= 3)
==	(a == b) bernilai True jika a = b	–-> p8 = (2 == 4)
~=	(a ~= b) bernilai True jika a ≠ b	–-> p9 = (2 ~= 4)
<>	(a <> b) bernilai True jika a ≠ b	–-> p10 = (2 <> 4)
<	(a < b) bernilai True jika a < b	–-> p11 = (2 < 4)
>	(a > b) bernilai True jika a > b	–-> p12 = (2 > 4)
<=	(a <= b) bernilai True jika a <= b	–-> p13 = (2 <= 4)
>=	(a >= b) bernilai True jika a >= b	–-> p14 = (2 >= 4)

3.3. Variabel tipe string

Suatu variable dinamakan dengan variable tipe string jika nilai variable tersebut diapit dengan tanda petik ganda. Operator "+" dapat digunakan untuk menggabungkan dua string atau lebih dari string menjadi satu string. Berikut diberikan beberapa contoh.

```
–-> sa = "Indonesia "
 sa  =
 Indonesia
–-> sb = "Raya"
 sb  =
 Raya
–-> sc = sa + sb
 sc  =
 Indonesia Raya
–-> z1 = 10
 z1  =
   10.
–-> sd = "Nilai z = "
 sd  =
 Nilai z =
–-> se = sd + sci2exp(z1)
 se  =
 Nilai z = 10
```

Pada contoh tersebut, perintah sci2exp(x) digunakan untuk mengubah suatu ekspresi, misalkan variabel x menjadi suatu string.

3.4. Variabel tipe integer

Variabel tipe real dapat dikonversi menjadi variable tipe integer. Daftar Fungsi yang dapat digunakan untuk menghasilkan variable tipe integer disajikan pada Tabel 3.3.

Tabel 3.3. Fungsi yang mengkonversi variable tipe real menjadi variable tipe integer

Fungsi	Deskripsi	Contoh penggunaan
int8(x)	Konversi variabel tipe real x menjadi tipe integer dalam rentang $[-2^7, 2^7 - 1] = [-128,127]$	–-> format(25) –-> u1 = 2^7 + 1 –-> u2 = int8(u1)
uint8	Konversi variabel tipe real x menjadi tipe integer dalam rentang $[0, 2^8 - 1] = [0,255]$	–-> u3 = 2^8 + 1 –-> u4 = uint8(u3)
int16	Konversi variabel tipe real x menjadi tipe integer dalam rentang $[-2^{15}, 2^{15} - 1] = [-32768,32767]$	–-> u5 = 2^15 –-> u6 = int16(u5)
uint16	Konversi variabel tipe real x menjadi tipe integer dalam rentang $[0, 2^{16} - 1] = [0,65535]$	–-> u7 = 2^16 + 2 –-> u8 = uint16(u7)
int32	Konversi variabel tipe real x menjadi tipe integer dalam rentang $[-2^{31}, 2^{31} - 1] = [-2147483648, 2147483647]$	–-> u9 = 2^31 + 3 –-> u10 = int32(u9)
uint32	Konversi variabel tipe real x menjadi tipe integer dalam rentang $[0, 2^{32} - 1] = [0, 4294967295\ ]$	–-> u11 = 2^32 + 5 –-> u12 = uint32(u11)

Berikut output yang diperoleh jika perintah tersebut dieksekusi pada perangkat lunak Scilab.

```
–-> format(25)
–-> u1 = 2^7 + 1
 u1  =
   129.
–-> u2 = int8(u1)
 u2  =
 -127
```

Perintah format(25) digunakan untuk mengatur banyaknya digit angka yang dapat ditampilkan menjadi 25 digit. Pada format int8(x), nilai bilangan terletak pada rentang – 127, - 126, …, 0, 1, 2, …, 127. Selain itu, urutan bilangan bersifat periodik. Akibatnya, nilai u2 adalah – 127.

```
–-> u3 = 2^8 + 1
 u3  =
   257.
–-> u4 = uint8(u3)
 u4  =
  1
```

Pada format uint8(x), nilai bilangan terletak pada rentang 0, 1, 2, …, 255. Selain itu, urutan bilangan bersifat periodik. Akibatnya, nilai u4 adalah 1.

```
--> u5 = 2^15
 u5  =
   32768.
--> u6 = int16(u5)
 u6  =
 -32768
```
Pada format int16(x), nilai bilangan terletak pada rentang – 32768, - 32767, …, 0, 1, 2, …, 32767. Selain itu, urutan bilangan bersifat periodic/sirkular. Akibatnya, nilai u6 adalah – 32768.

```
--> u7 = 2^16 + 2
 u7  =
   65538.
--> u8 = uint16(u7)
 u8  =
  2
```
Pada format uint16(x), nilai bilangan terletak pada rentang 0, 1, 2, …, 65535. Selain itu, urutan bilangan bersifat periodic/sirkular. Akibatnya, nilai u8 adalah 2.

```
--> u9 = 2^31 + 3
 u9  =
   2147483651.
--> u10 = int32(u9)
 u10  =
 -2147483648
```
Pada format int32(x), nilai bilangan terletak pada rentang – 2147483648, -2147483647, …, 0, 1, …, 2147483647. Selain itu, nilai bilangan yang lebih besar dari 2147483647 ($2^{31} - 1$) akan didefinisikan sebagai – 2147483648. Akibatnya, nilai u10 adalah – 2147483648.

```
--> u11 = 2^32 + 5
 u11  =
   4294967301.
--> u12 = uint32(u11)
 u12  =
  5
```
Pada format uint32(x), nilai bilangan terletak pada rentang 0, 1, 2, …, 42949667295. Selain itu, urutan bilangan pada format uint32(x) bersifat periodik. Akibatnya, nilai u12 adalah 5.

4. Manipulasi Matriks menggunakan Scilab

4.1. Mendefinisikan Matriks pada Scilab

Pada bagian ini akan disajikan cara mendefinisikan dan memanipulasi matriks. Dalam perangkat lunak Scilab, tipe data integer, real, kompleks, boolean dan string *dapat dipandang sebagai suatu matriks*. Variabel tipe data integer, real, kompleks, Boolean dan string dapat dipandang sebagai matriks berordo 1 x 1.

```
--> clear;
--> x1 = 2.5;
--> x2 = 1 – 2*%i;
--> size(x1)
    ans  =
      1.   1.
--> size(x2)
    ans  =
      1.   1.
--> x1(1)
    2.5
--> x2(1,1)
     1.  - 2.i
```

Perintah clear digunakan untuk menghapus semua variabel pada jendela *Variable Browser*. Perintah size(x1) dan size(x2) digunakan untuk menentukan ukuran/ordo matriks x1 dan x2. Misalkan X adalah suatu matriks. Perintah X(i, j) digunakan untuk mengakses nilai baris ke-i dan kolom ke-j pada matriks X.

Matriks dapat didefinisikan dengan pada perangkat lunak Scilab dengan memperhatikan hal-hal berikut:

(a) Elemen matriks diletakkan di antara tanda [pada awal matriks dan tanda] pada akhir matriks.
(b) Elemen-elemen matriks pada satu baris **dapat dipisahkan dengan tanda koma** (,).
(c) Elemen-elemen matriks pada baris yang berbeda dipisahkan dengan tanda titik-koma (;). kemudian memasukkan elemen-elemen matriks.

Berikut diberikan contoh pendefinisian matriks.

```
--> A1 = [1, 3, 5; 7, 9, 11; 13, 15, 17]
     A1  =
      1.    3.    5.
      7.    9.    11.
      13.   15.   17.
--> [nbar, nkol] = size(A1)
    nkol  =
      3.
    nbar  =
      3.
```

```
--> n_elemen = size(A1,"*")
      n_elemen  =
       9.
```

Perintah size(A1,"*") digunakan untuk menentukan banyaknya elemen matriks A1.

Matriks juga dapat didefinisikan dengan cara berikut.

```
--> A2 = [1   3   5; 7   9   11;13   15   17]
      A2  =
       1.    3.    5.
       7.    9.    11.
       13.   15.   17.
```

Matriks kosong (empty matrix) dapat didefinisikan dengan cara berikut.

```
--> A3 = [ ]

A3  =
   [ ]
--> size(A3)
 ans  =
  0.   0.
```

Beberapa perintah pada Tabel 4.1 berikut digunakan untuk menghasilkan matriks dengan ordo tertentu. Sebagian besar sintaks perintah pada Tabel 4.1 memerlukan dua parameter, yaitu banyaknya baris dan banyaknya kolom matriks.

Tabel 4.1. Beberapa sintaks perintah untuk menghasilkan matriks pada perangkat lunak Scilab

Perintah	Deskripsi	Contoh penggunaan
eye(m, n)	Output berupa matriks dengan pola persegi berordo n x n	A4 = eye(3, 3) A5 = eye(3, 2)
ones(m, n)	Output berupa matriks berordo m x n, dengan semua elemen adalah 1	A6 = ones(4, 2)
zeros(m, n)	Output berupa matriks berordo m x n, dengan semua elemen adalah 0	A7 = zeros(4, 2)
rand(m, n)	Output berupa matriks berordo m x n, dengan elemen matriks adalah bilangan pseudorandom berdistribusi uniform(0, 1)	A8 = rand(2, 4) A9 = rand(2, 4)
grand(m, n, “distribusi”)	Output berupa matriks berordo m x n dengan elemen matriks adalah bilangan pseudorandom sesuai jenis “distribusi” yang disebutkan	A10 = grand(4, 2, “exp”, 3) A11 = grand(4, 2, “poi”, 3)
testmatrix(“jenis”, n)	Output berupa matriks khusus, seperti magic matrix, Franck matrix	A12 = testmatrix (“magic”, 4)
linspace(a, b, n)	Output berupa vektor baris dengan elemen vector bernilai antara a dan b dan mengikuti pola barisan aritmetika	A13 = linspace(0, 1, 11)

Berikut diberikan beberapa contoh hasil eksekusi sintaks perintah tersebut.

```
--> A4 = eye(3, 3)
 A4  =
   1.   0.   0.
   0.   1.   0.
   0.   0.   1.
--> A5 = eye(3, 2)
 A5  =
   1.   0.
   0.   1.
   0.   0.

--> A6 = ones(4, 2)
 A6  =
   1.   1.
   1.   1.
   1.   1.
   1.   1.

--> A7 = zeros(4, 2)
 A7  =
   0.   0.
   0.   0.
   0.   0.
   0.   0.

--> A8 = rand(2, 4)
 A8  =
   0.8782164813   0.5608486063   0.7263506767   0.5442573163
   0.0683740368   0.6623569373   0.1985143842   0.2320747897

--> A9 = rand(2, 4)
 A9  =
   0.2312237197   0.8833887815   0.3076090743   0.2146007861
   0.2164632631   0.6525134947   0.9329616213   0.3126419969
```

Perlu diperhatikan bahwa output perintah rand bervariasi, mengingat perintah rand menghasilkan bilangan pseudorandom berdistribusi uniform.

```
--> A10 = grand(2, 4, "exp",3)
 A10  =
   1.8883421514   0.2640616526   2.7774461517   0.2833235781
   4.4103311934   0.4596273719   7.92026392     2.4214201681
```

Perintah tersebut menghasilkan matriks berordo 2 x 4 dengan elemen matriks merupakan bilangan pseudorandom berdistribusi eksponensial dengan rata-rata 3.

```
--> A11 = grand(2, 4, "poi",3)
 A11  =
   1.   8.   9.   7.
   3.   6.   6.   1.
```

Perintah tersebut menghasilkan matriks berordo 2 x 4 dengan elemen matriks merupakan bilangan pseudorandom berdistribusi poisson dengan rata-rata 3.

```
--> A12 = testmatrix ("magic", 4)
 A12  =
   16.   2.    3.    13.
   5.    11.   10.   8.
   9.    7.    6.    12.
   4.    14.   15.   1.
```

Perintah tersebut menghasilkan *magic matrix* (matriks persegi ajaib) berordo 4 x 4.

```
--> A13 = linspace(0, 1, 11)
A12  =
   0.   0.1   0.2   0.3   0.4   0.5   0.6   0.7   0.8   0.9   1.
```

Perintah tersebut menghasilkan vektor baris dengan elemen vector bernilai antara 0 dan 1, terdiri dari 11 elemen dan mengikuti pola barisan aritmetika.

4.2. Mengakses elemen-elemen matriks

Misalkan X adalah suatu matriks. Ada beberapa metode yang dapat digunakan untuk mengakses elemen-elemen matriks X:

(a) Untuk mengakses seluruh elemen matriks X, digunakan perintah
```
--> X
```
(b) Untuk mengakses satu elemen, misalkan elemen matriks X pada baris ke-i kolom ke-j, digunakan perintah
```
--> X(i, j)
```
(c) Untuk mengakses sekumpulan elemen matriks X, digunakan operator titik-dua (":").

Beberapa variasi sintaks operator titik-dua (:) disajikan pada Tabel 4.2.

Tabel 4.2. Beberapa variasi sintaks operator titik-dua untuk mengakses elemen matriks

Perintah	Deskripsi
X	Output berupa seluruh elemen matriks X
X(:, :)	Output berupa seluruh elemen matriks X
X(i:j, k)	Output berupa elemen matriks X baris ke-i sampai dengan baris ke-j pada kolom ke-k
X(i, j:k)	Output berupa elemen matriks X baris ke-i, kolom ke-j sampai kolom ke-k
X(i, :)	Output berupa elemen matriks X baris ke-i.
X(:, j)	Output berupa elemen matriks X baris ke-i.

Berikut diberikan beberapa contoh penggunaan operator titik-dua untuk mengakses elemen suatu matriks.

```
--> B = testmatrix("magic",6)
 B  =
   35.   1.    6.    26.   19.   24.
   3.    32.   7.    21.   23.   25.
   31.   9.    2.    22.   27.   20.
   8.    28.   33.   17.   10.   15.
   30.   5.    34.   12.   14.   16.
   4.    36.   29.   13.   18.   11.

--> B(1:2, :)
 ans  =

   35.   1.    6.   26.   19.   24.
   3.    32.   7.   21.   23.   25.
```

Sintaks perintah digunakan untuk mengakses elemen matriks B pada baris ke-1 dan baris ke-2.

```
--> B(:, 1:2:5)
 ans  =

   35.   6.    19.
   3.    7.    23.
   31.   2.    27.
   8.    33.   10.
   30.   34.   14.
   4.    29.   18.
```

Sintaks perintah digunakan untuk mengakses elemen matriks B pada kolom ke-1, kolom ke-3 dan kolom ke-5.

Untuk mengakses $b_{i,j}$ dengan i = 1, 2 dan j = 3,4 dapat digunakan sintaks perintah berikut.

```
--> B(1:2, 3:4)
 ans  =
   6.   26.
   7.   21.
```

Untuk mengakses elemen matriks dengan merujuk pada baris terakhir atau kolom terakhir, dapat dipergunakan operator $. Perhatikan beberapa contoh berikut. Untuk mengakses baris ke-4 dan baris ke-5 (baris terakhir), dapat digunakan sintaks perintah berikut.

```
--> B($-1:$, :)
 ans  =
   30.   5.    34.   12.   14.   16.
   4.    36.   29.   13.   18.   11.
```

Misalkan matriks C diperoleh dengan penukaran kolom ke-1 matriks B ditukar dengan kolom ke-3. Untuk memperoleh matriks C tersebut, dapat digunakan sintaks perintah berikut.

```
--> C = B;
```

Pada awalnya, matriks C didefinisikan sama dengan matriks B. Selanjutnya, dapat dilakukan operasi pertukaran kolom ke-1 dengan kolom ke-3.

```
--> C(:,[1,3]) = B(:, [3, 1])

 C  =
   6.    1.    35.   26.   19.   24.
   7.    32.   3.    21.   23.   25.
   2.    9.    31.   22.   27.   20.
   33.   28.   8.    17.   10.   15.
   34.   5.    30.   12.   14.   16.
   29.   36.   4.    13.   18.   11.
```

4.3. Variabel matriks bersifat dinamik

Pada perangkat lunak Scilab, variabel tipe matriks bersifat dinamik, dengan pengertian bahwa ordo matriks dapat ditambah atau dikurangi. Perhatikan sintaks perintah berikut.

```
--> D = []
 D  =
    [ ]
```

Pada awalnya, matriks D didefinisikan sebagai null-matrix (matriks kosong yang tidak memuat elemen matriks).

```
--> D = [1, 3, 5; 2, 4, 6]
 D  =
   1.   3.   5.
   2.   4.   6.
```

Matriks D diubah menjadi matriks berordo 2 x 3. Misalkan didefinisikan D(3,4) = 7.

```
--> D(3,4) = 7
D  =
   1.   3.   5.   0.
   2.   4.   6.   0.
   0.   0.   0.   7.
```

Dengan sintaks perintah tersebut, matriks D berubah menjadi matriks berordo 3 x 4. Nilai elemen-elemen lainnya diset sama dengan nol.

Selain peningkatan ordo/ukuran matriks, pengurangan ordo matriks juga dapat dilakukan. Sintaks perintah berikut dapat digunakan untuk menghapus kolom ke-3 matriks D.

```
 --> D(:, 3) = [ ]
 D  =
   1.   3.   0.
   2.   4.   0.
   0.   0.   7.
```

Suatu matriks dapat diubah/dikonversi menjadi menjadi matriks lain dengan ketentuan bahwa banyaknya elemen kedua matriks tersebut adalah sama. Matriks D berordo 3 x 3 tersebut dapat

dikonversi menjadi menjadi matriks berordo 1 x 9, menggunakan sintaks perintah matrix, seperti pada contoh berikut.

```
--> E = matrix(D, 1, 9)
E  =
   1.   2.   0.   3.   4.   0.   0.   0.   7.
-->matrix(D, 2, 5)
matrix: input and output matrices must have the same number of elements
```

Sintaks perintah tersebut menghasilkan pesan kesalahan, karena input matriks D memuat 9 elemen, sedangkan output perintah matrix memuat 10 elemen.

4.4. Operasi pada Matriks

Telah diketahui bahwa terdapat operasi penjumlahan, pengurangan dan perkalian dua matriks dengan elemen-elemen matriks berupa bilangan real. Pada satu matriks, dapat dilakukan operasi transpos matriks. Pada perangkat lunak Scilab, tersedia juga operasi transpos konjugat, perpangkatan, pembagian dari sisi kanan, dan pembagian dari sisi kiri. Selain itu, terdapat juga operasi berbasis elemen-per-elemen (*elementwise*). Tabel 4.3 menyajikan operasi yang dapat dikenakan pada satu matriks atau dua matriks.

Tabel 4.3. Operasi pada matriks pada perangkat lunak Scilab

Perintah	Deskripsi	Perintah	Deskripsi
+	Penjumlahan	.+	Penjumlahan elemen-per-elemen
-	Pengurangan	.-	Pengurangan elemen-per-elemen
*	Perkalian	.*	Perkalian elemen-per-elemen
/	Pembagian dari kanan	./	Pembagian dari kanan elemen-per-elemen
\	Pembagian dari kiri	.\	Pembagian dari kiri elemen-per-elemen
^ atau **	Perpangkatan	.^	Perpangkatan elemen-per-elemen
'	Transpose dan konjugat	.'	Transpose

Berikut diberikan beberapa contoh hasil eksekusi sintaks perintah tersebut. Contoh berikut mengilustrasikan perbedaan operasi transpose konjugat dan operasi transpose.

```
--> Fa = [1 + 2*%i, 3; 4, 2 + %i]
 Fa  =
   1. + 2.i     3.
   4.           2. + i
--> Fa'
 ans  =
   1. - 2.i     4.
   3.           2. - i
--> Fa.'
 ans  =
   1. + 2.i     4.
   3.           2. + i
```

Operasi transpose konjugat dan transpose akan menghasilkan output yang sama ketika kedua operasi tersebut dikenakan pada suatu matriks dengan elemen bilangan real.

```
--> Fb = [1, 2; 3, 4]
 Fb  =
   1.    2.
   3.    4.
--> Fb'
 ans  =
   1.    3.
   2.    4.
--> Fb.'
 ans  =
   1.    3.
   2.    4.
```

Misalkan A dan B merupakan matriks dengan elemen bilangan real atau bilangan kompleks. Operasi penjumlahan dan pengurangan kedua matriks tersebut akan terdefinisi jika ordo matriks A sama dengan ordo matriks B. Jika ordo kedua matriks tidak sama, maka operasi penjumlahan dan pengurangan dua matriks tidak terdefinisi.

```
--> Fb = [1, 2; 3, 4]
 Fb  =
   1.    2.
   3.    4.
--> Fc = [3, 4; 5, 6]
 Fc  =
   3.    4.
   5.    6.
--> Fd = [1, 3, 5; 2, 4, 6]
 Fd  =
   1.    3.    5.
   2.    4.    6.

--> Fb + Fc
 ans  =
   4.    6.
   8.    10.

--> Fb - Fc
 ans  =
  -2.   -2.
  -2.   -2.
--> Fb + Fd
        Inconsistent row/column dimensions.
```

Karena ordo matriks Fb tidak sama dengan ordo matriks Fd, maka perintah Fb + Fd akan menghasilkan suatu pesan kesalahan.

Misalkan A dan B merupakan matriks dengan elemen bilangan real atau bilangan kompleks. Operasi perkalian A*B terdefinisi jika banyaknya kolom matriks A, adalah sama dengan banyaknya baris matriks B. Jika kondisi tersebut tidak terpenuhi, maka operasi perkalian A*B tidak terdefinisi.

```
--> Fb * Fd
 ans  =
   5.    11.   17.
   11.   25.   39.
--> Fd * Fb
Inconsistent row/column dimensions.
```

Karena banyaknya kolom matriks Fd (3 kolom) tidak sama dengan bnyaknya baris matriks Fb (2 baris), maka operasi perkalian Fd*Fb tidak terdefinisi, sehingga muncul pesan kesalahan. tidak sama dengan ordo matriks Fd, maka perintah Fb + Fd akan menghasilkan suatu pesan kesalahan.

Misalkan A dan B merupakan matriks dengan elemen bilangan real atau bilangan kompleks. Operasi pembagian matriks dari kiri, X = A \ B menyatakan bahwa X adalah solusi dari AX = B.

```
--> G1 = [1, 1, 1; 1, 2, 3; 4, 5, 9]
 G1  =
   1.   1.   1.
   1.   2.   3.
   4.   5.   9.
--> H1 = H1 = [6; 14; 41]
 H1  =
   6.
   14.
   41.

--> X1 = G1 \ H1
X1  =
   1.
   2.
   3.
```

Operasi pembagian matriks dari kanan, X = A / B menyatakan bahwa X adalah solusi dari persamaan matriks X*B = A.

```
--> H2 = G1
H2  =
   1.   1.   1.
   1.   2.   3.
   4.   5.   9.
```

```
--> G2 = [25, 33, 56]
G2  =
  25.   33.   56.

--> G2/H2
 ans  =
  0.   3.   5.
```

Misalkan A adalah suatu matriks persegi dengan komponen bilangan real atau bilangan kompleks dan n adalah suatu bilangan bulat positif. Operasi A^n diperoleh dari
A^ n = A * A * … * A (sebanyak n factor)

```
--> G3 = [1, 2; 3, 4]
 G3  =
  1.   2.
  3.   4.

--> G3^3
 ans  =
  37.   54.
  81.   118.

--> H3 = G3*G3*G3
 H3  =
  37.   54.
  81.   118.

--> H4 = G3^3
 H4  =
  37.   54.
  81.   118.
```

Operasi matriks elemen-per-elemen antara matriks A dan matriks B dapat dilakukan jika ordo matriks A sama dengan ordo matriks B. Berikut diberikan beberapa contoh yang terkait.

```
--> Fe = [1, 1, 1; 1, 1, 1]
 Fe  =
  1.   1.   1.
  1.   1.   1.

--> Ff = [2, 2, 2; 2, 2, 2]
 Ff  =
  2.   2.   2.
  2.   2.   2.
```

```
--> Fe .* Ff
 ans  =
   2.   2.   2.
   2.   2.   2.

--> Fe ./ Ff
 ans  =
   0.5   0.5   0.5
   0.5   0.5   0.5

--> Fe .\ Ff
 ans  =
   2.   2.   2.
   2.   2.   2.

--> Fe .^ Ff
 ans  =
   1.   1.   1.
   1.   1.   1.
```

4.5. Perbandingan dua matriks

Misalkan A dan B adalah matriks dengan elemen bilangan real. Perbandingan matriks A dan matriks B dalam perangkat lunak Scilab dapat dilakukan jika ordo matriks A sama dengan ordo matriks B. Berikut diberikan beberapa contoh terkait.

```
--> clear
```

Perintah clear tersebut digunakan untuk menghapus seluruh variabel dari jendela Variable Browser.

```
--> x1 = [5, 3, 2]
 x1  =
   5.   3.   2.

--> x2 = [2, 4, 6]
 x2  =
   2.   4.   6.

--> a = x1 < x2
 a  =
  F  T  T

--> x1 > 2
 ans  =
  T  T  F
```

5. Statemen Kondisional dan Pengulangan

5.1. Statemen kondisional

Statemen kondisional digunakan untuk menjalankan satu atau beberapa perintah jika suatu kondisi terpenuhi. Ada dua pilihan statemen kondisional pada perangkat lunak Scilab, yaitu statemen **if** dan statemen **select**. Berikut disajikan beberapa sintaks statemen if:

(a) **if** [kondisi-x] **then**
[perintah-x]
end

(b) **if** [kondisi-x] **then**
[perintah-x]
else
[perintah-y]
end

(c) **if** [kondisi-x] **then**
[perintah-x]
elseif [kondisi-y] **then**
[perintah-y]
else
[perintah-z]
end

Catatan:

(a) Statemen **then** dapat diganti dengan penekanan tombol **ENTER**, tanda koma (**,**) atau tanda titik-koma (**;**).
(b) Statemen **else** bersifat opsional, sehingga dapat diabaikan jika statemen tersebut tidak diperlukan.

Pada sintaks (a), jika [kondisi-x] terpenuhi (bernilai True), maka statemen [perintah-x} dijalankan. Sebaliknya jika [kondisi-x] tidak terpenuhi (bernilai False), maka statemen [perintah-x] tidak dijalankan.

```
--> clear
--> r = rand( )
        r  =
  0.2113249
--> if (r < 0.5) then
        disp( ' r = ' + sci2exp(r));
        disp( ' Bilangan random r lebih kecil dari 0.5 ');
    end

        r = 0.2113249
        Bilangan random r lebih kecil dari 0.5
```

Pada sintaks (b), jika [kondisi-x] terpenuhi (bernilai True), maka statemen [perintah-x} dijalankan. Sebaliknya jika [kondisi-x] tidak terpenuhi (bernilai False), maka statemen [perintah-y] dijalankan.

```
--> s = rand( )
 s  =
   0.6653811
--> if (s < 0.5)
        disp( ' s = ' + sci2exp(s));
        disp( ' Bilangan random s lebih kecil dari 0.5 ');
    else
        disp( ' s = ' + sci2exp(s));
        disp( ' Bilangan random s lebih besar dari atau sama dengan 0.5 ');
    end

        s = 0.6653811
  Bilangan random s lebih besar dari atau sama dengan 0.5
```

Pada sintaks (b), jika [kondisi-x] terpenuhi (bernilai True), maka statemen [perintah-x} dijalankan. Jika [kondisi-x] tidak terpenuhi (bernilai False) dan [kondisi-y] terpenuhi, maka statemen [perintah-y] dijalankan. Jika [kondisi-x] dan [kondisi-y] tidak terpenuhi, maka statemen [perintah-z] yang dijalankan.

```
--> t = rand( )
 s  =
   0.6283918
--> if (t <= 0.5)
        disp( ' t = ' + sci2exp(t));
        disp( ' Bilangan random s lebih kecil dari atau sama dengan 0.5 ');
    elseif (t < 0.75)
        disp( ' t = ' + sci2exp(t));
        disp( ' Nilai adalah bilangan random t, 0.5 < t <= 0.75 ');
    else
        disp( ' t = ' + string(t));
        disp( ' Bilangan random lebih besar dari 0.75 ');
    end

        t = 0.6283918
  Nilai adalah bilangan random t, 0.5 < t <= 0.75
```

Pada contoh tersebut, perintah **sci2exp**(t) dan **string**(t) digunakan untuk mengubah variabel t menjadi tipe string

Statemen kondisional **select-case** cocok untuk digunakan jika kondisi mempunyai nilai diskrit, misalkan berupa nilai integer atau nilai string. Bentuk umum statemen kondisional select-case adalah

```
select [ekspresi]
    case [ekspresi-1] then
        [perintah-1]
    case [ekspresi-2] then
        [perintah-2]
    ...
    case [ekspresi-n] then
        [perintah-n]
    else
        [perintah-m]
end
```

Berikut diberikan contoh sederhana penggunaan statemen kondisional select-case.

```
--> x = input(' Masukkan nomor pilihan Anda (1, 2 atau 3) : ');
    Inputkan pilihan (1,2,3) : 4 (Enter)
--> select x
    case x == 1
        disp(' x = ' + string(x));
        disp(' Anda memilih 1');
    case x == 2
        disp(' x = ' + string(x));
        disp(' Anda memilih 2');
    case x == 3
        disp(' x = ' + string(x));
        disp(' Anda memilih 3');
    else
        disp(' x = ' + string(x));
        disp(' Anda memilih selain 1, 2, 3.');
  end
    x = 4
    Anda memilih selain 1, 2, 3.
```

Pada contoh tersebut, statemen **string**(x) digunakan untuk mengubah nilai variabel x menjadi bentuk string.

5.2. Statemen pengulangan

Pada perangkat lunak Scilab, statemen **for** dan statemen **while** dapat digunakan sebagai statemen pengulangan. Statemen **for** digunakan untuk melakukan pengulangan yang tidak memerlukan syarat/kondisional tertentu. Berikut diberikan beberapa contoh penggunaan statemen **for**.

```
--> clear
--> x = zeros(1, 5);
--> for i = 1:5 do
        x(i) = 1/(i + 1);
    end
--> x
        x  =
  0.5   0.3333333   0.25   0.2   0.1666667
```

Catatan: Statemen **do** bersifat opsional, sehingga statemen **do** dapat diabaikan.

```
--> A = zeros(4,4)
--> for j = 1:4
        for k = 1:4
            A(j, k) = 1/(j + k -1);
        end
    end
--> A
A  =
  1.          0.5         0.3333333   0.25
  0.5         0.3333333   0.25        0.2
  0.3333333   0.25        0.2         0.1666667
  0.25        0.2         0.1666667   0.1428571
```

Statemen pengulangan for juga dapat menggunakan pola penghitungan mundur, seperti disajikan pada contoh berikut.

```
--> y = zeros(1, 5);
--> for j = 9:-2:1 do
        y((j+1)/2) = 1/(j + 1);
    end
--> y
        y  =
  0.5   0.25   0.1666667   0.125   0.1
```

Statemen pengulangan **while** digunakan untuk melakukan proses pengulangan ketika suatu kondisi terpenuhi. Sintaks umum statemen pengulangan **while** berbentuk

while [kondisi-x] **then**
    [perintah-x]
end

Sintaks perintah **then** dapat diganti dengan **do**, penekanan tombol **ENTER**, simbol koma (**,**), atau simbol titik-koma (**;**). Berikut diberikan contoh penggunaan statemen pengulangan while dalam penentuan hampiran akar kuadrat suatu bilangan positif a, menggunakan metode Newton-Raphson.

```
--> clear
--> a = 5; // Bilangan yang akan ditentukan hampiran akar kuadrat
--> xlama = a; // Nilai awal untuk akar kuadrat a
--> toleransi = 5e-7; // Nilai toleransi galat
--> galat = 1; // Nilai awal galat
--> iterasi = 0; // Nilai awal iterasi
--> while ((galat >= toleransi) & (iterasi <= 50))
              x = 0.5 * (xlama + a/xlama);
              galat =  abs(x-xlama);
              xlama = x;
              iterasi = iterasi + 1;
    end
--> disp(' Banyak iterasi = ' + string(iterasi));
      Banyak iterasi = 6
--> disp(' Nilai hampiran akar kuadrat ' + string(a) + ' adalah ' + string(x));
      Nilai hampiran akar kuadrat 5 adalah 2.236068
```

5.3. Statemen break dan continue

Dalam suatu statemen pengulangan for atau statemen pengulangan while, perintah break digunakan untuk menghentikan proses pengulangan. Berikut diberikan sebuah contoh sederhana terkait penggunaan statemen break.

```
--> k=0;
--> while (%T)
        k = k+1;
        if k > 100 then
        break
        end;
  end
```

Tanpa adanya statemen **break**, statemen pengulangan **while** tersebut akan terus dieksekusi karena kondisi True (%T) tetap terpenuhi. Ketika kondisi (k > 100) sudah tercapai, maka proses pengulangan dihentikan dengan menggunakan statemen **break**.

Penggunaan statemen **continue** pada suatu blok pengulangan akan menyebabkan statemen setelah statemen **continue** pada blok pengulangan tersebut, tidak akan dieksekusi. Perhatikan contoh berikut.

```
--> for k=1:10
        j = k;
        if k>2 & k<=8 then
                continue
                disp('Perangkat lunak Scilab')
        end
        disp(k);
        end
```

Apabila perintah tersebut dijalankan, akan diperoleh output berikut.

1.
2.
9.
10.

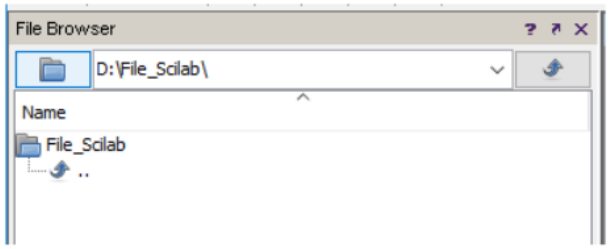
Ketika kondisi 2 < k <= 8, maka perintah setelah statemen **continue** dalam blok pengulangan **for**, yaitu perintah **disp('Perangkat lunak Scilab')** dan perintah **disp(k)**, tidak dijalankan.

6. *Script* dan Fungsi

Pada bahasan-bahasan sebelumnya, statemen/perintah dijalankan pada jendela *Scilab Console*. Ketika perintah yang dijalankan relatif sederhana dan singkat, maka pengetikan perintah pada jendela *Scilab Console* masih cukup memadai. Akan tetapi, ketika statemen perintah yang dijalankan relatif kompleks, maka pengetikan perintah pada jendela *Scilab Console* menjadi tidak efektif. Untuk mengatasi keterbatasan pengetikan perintah pada jendela Scilab Console, pengguna perangkat lunak Scilab dapat menggunakan fitur script atau fungsi, dan menyimpan file *script* atau fungsi dengan ekstensi file **sce** atau **sci** (file tipe Scilab editor). Perbedaan pokok antara *script* dan fungsi adalah: *script* tidak memerlukan input, sedangkan sebagaian besar fungsi menerima input dan menghasilkan output.

6.1. Script

File script merupakan file teks yang di dalamnya terdapat statemen/perintah dalam bahasa pemrograman Scilab. File script sebaiknya disimpan dalam folder khusus yang digunakan untuk menyimpan file script atau fungsi Scilab. Misalkan folder D:\File_Scilab akan ditetapkan menjadi folder aktif penyimpanan file script atau file fungsi Scilab. Klik tombol *Select Folder*, yaitu ikon bertanda ( ) pada bagian kiri atas jendela *File Browser*. Selanjutnya, klik folder D:\File_Scilab, lalu klik tombol *Open*. Tampilan jendela File Browser dengan folder D:\File_Scilab menjadi folder aktif, disajikan pada Gambar 6.1.



Gambar 6.1. Folder D:\File_Scilab menjadi folder aktif pada jendela *File Browser*

Pada perangkat lunak Scilab, tersedia program editor teks SciNotes yang dapat digunakan untuk membangun file script atau fungsi. Dari jendela Scilab Console, editor Scinotes dapat diaktifkan dengan melakukan klik menu *Application – SciNotes*. Untuk menyimpan file script atau file fungsi yang sudah dibuat, lakukan klik menu *File – Save*.

Contoh 6.1. [Barisan Fibonacci] Barisan (x_n) disebut barisan Fibonacci jika barisan (x_n) mempunyai sifat $x(1) = x(2) = 1$, dan

$$x(n) = x(n-2) + x(n-1), \qquad n = 3,4, \ldots$$

Berikut disajikan kode program **Fibonacci.sce** yang digunakan untuk menampilkan beberapa suku awal barisan Fibonacci.

Kode program 6.1. Barisan Fibonacci

```
//Kode program 6.1. Fibonacci.sce
clear;
clc;

n = 15;
x = ones(1,n);
for k = 3:n
        x(k) = x(k-1) + x(k-2);
end
disp(string(n) + ' suku pertama barisan Fibonacci');
disp(x);
//Akhir kode program
```

Statemen perintah yang terdapat di dalam sebuah file script yang sedang terbuka pada jendela *SciNotes Editor* dapat dijalankan dengan melakukan klik menu *Execute – Save and execute*. Pada jendela *Scilab Console* muncul tampilan berikut.

```
--> exec('D:\File_Scilab\Fibonacci.sce', -1)

 15 suku pertama barisan Fibonacci
       column 1 to 14
  1.   1.   2.   3.   5.   8.   13.   21.   34.   55.   89.   144.   233.   377.
       column 15
  610.
```

Output tersebut menujukkan bahwa file script juga dapat dijalankan dari jendela Scilab Console dengan mengetikkan perintah

exec(' D:\File_Scilab\Fibonacci.sce ', -1)

dan diikuti dengan menekan tombol ENTER. Ketika file *script* **Fibonacci.sce** berada dalam folder aktif, maka script **Fibonacci.sce** juga dapat dijalankan dengan mengetikkan perintah

exec('Fibonacci.sce', -1)

dan diikuti dengan menekan tombol ENTER. Parameter -1 pada perintah exec tersebut menunjukkan bahwa file script **Fibonacci.sce** dieksekusi tanpa *echo* (*execute without echo*).

Untuk menjalankan file script **Fibonacci.sce dengan echo,** dapat diketikkan perintah **exec('D:\File_Scilab\Fibonacci.sce')**
pada jendela *Scilab Console* dan diikuti dengan menekan tombol ENTER.

```
--> exec('D:\File_Scilab\Fibonacci.sce')
--> //Kode program 6.1. Barisan Fibonacci
--> clear;
--> n = 15;
```

```
--> x = ones(1,n);
--> for k = 3:n
-->     x(k) = x(k-1) + x(k-2);
--> end
--> disp(string(n) + ' suku pertama barisan Fibonacci');
 15 suku pertama barisan Fibonacci
--> disp(x);
       column 1 to 14
  1.   1.   2.   3.   5.   8.   13.   21.   34.   55.   89.   144.   233.   377.
       column 15
  610.
```

Perintah exec tersebut juga dapat diakhiri dengan tanda titik koma(;). Apabila perintah
exec('D:\File_Scilab\Fibonacci.sce');
dijalankan pada jendela Scilab Console, diperoleh output berikut.

```
--> exec('D:\File_Scilab\Fibonacci.sce');
 15 suku pertama barisan Fibonacci
       column 1 to 14
  1.   1.   2.   3.   5.   8.   13.   21.   34.   55.   89.   144.   233.   377.
       column 15
  610.
```

6.2. Fungsi

Fungsi merupakan suatu sub program yang dibuat untuk menyelesaikan suatu tahapan komputasi tertentu. Sebagian besar fungsi menerima satu atau beberapa input dan menghasilkan satu atau beberapa output. Sintaks penulisan fungsi pada perangkat lunak Scilab diawali dengan kata kunci **function** dan diakhiri dengan kata kunci **endfunction** seperti format berikut.

```
function [y1,y2,..., ym] = [nama-fungsi](x1, x2, ..., xn)
    [perintah-fungsi];
endfunction
```

Pada sintaks tersebut, parameter x1, x2, …, xn merupakan input fungsi, sedangkan y1, y2, …, ym merupakan output fungsi. Parameter [nama-fungsi] menyatakan nama fungsi, sedangkan [perintah-fungsi] berisi satu atau sejumlah perintah yang perlu dijalankan untuk memperoleh output fungsi.

Catatan penting

Untuk menghindari kerancuan, **nama file Scilab** sebaiknya disamakan dengan **nama fungsi**.

Contoh 6.2. Misalkan diberikan fungsi f, dengan $f(x) = x^2 + e^{-x} - 10$, dengan x merupakan bilangan real. Berikut disajikan kode program fungsi **fa.sce** yang digunakan untuk mendefinisikan fungsi f tersebut.

Kode program 6.2. Pendefinisian fungsi fa.sce

```
//Kode program 6.2. fa.sce
function y = fa(x)
    y = x.^2 + exp(-x) - 10;
endfunction
//Akhir kode program
```

Contoh 6.3. Gambarlah grafik fungsi f pada **Contoh 6.2** untuk $-3 \leq x \leq 6$. Kode program *script* **plot_fa.sce** berikut digunakan untuk menghasilkan grafik fungsi f pada **Contoh 6.2**.

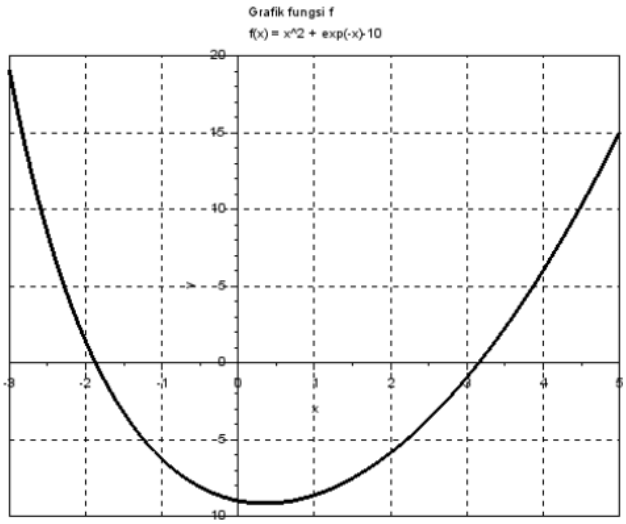
Kode Program 6.3. plot_fa.sce

```
//Kode program 6.3. plot_fa.sce
clear; // Menghapus semua variabel dari jendela Variable View
clc; // Membersihkan jendela Scilab editor
clf(); // Menutup figure
exec('fa.sce'); // Membuka dan menjalankan file fa.sce
x = (-3:0.1:5).';
y = fa(x);

ure(0);
plot(x,y,'k',"linewidth",3);
xlabel('x');
ylabel('y');
title(['Grafik fungsi f'; ' f(x) = x^2 + exp(-x)-10']);
xgrid(1);
a=gca(); // Menangani sumbu koordinat grafik
a.x_location ="origin";
a.y_location ="origin";
//Akhir kode program
```

Perintah **exec('fa.sce')** pada kode program 6.3 tersebut digunakan untuk mengaktifkan file fungsi fa.sce ke dalam ruang kerja Scilab. Untuk menjalankan kode program 6.3 tersebut, dapat dilakukan klik menu *Execute – Save and execute* atau dengan mengetikkan perintah

exec('plot_fa.sce',-1)

pada jendela Scilab Console. Hasil eksekusi kode program 6.3 berupa grafik fungsi f, $f(x) = x^2 + e^{-x} - 10, -3 \leq x \leq 6$ yang disajikan pada Gambar 6.1.



Gambar 6.1. Grafik fungsi f, $f(x) = x^2 + e^{-x} - 10, -3 \leq x \leq 6$.

Untuk lebih memberikan kejelasan tentang penggunaan file script, file fungsi, dan pembahasan materi sebelumnya, akan dibahas implementasi metode Newton untuk menentukan akar suatu fungsi. Misalkan akan ditentukan akar fungsi g, $g(x) = 0$. Hampiran akar g dengan menggunakan metode Newton pada iterasi ke (n + 1) diberikan oleh

$$x_{n+1} = x_n - \frac{g(x_n)}{g'(x_n)}, n = 1,2, \ldots$$

Untuk menerapkan metode Newton tersebut, diperlukan tebakan nilai awal akar x_0. Contoh 6.4 berikut membahas penggunaan metode Newton untuk menentukan hampiran akar fungsi f, dengan $f(x) = x^2 + e^{-x} - 10$.

Contoh 6.4. Akan ditentukan hampiran akar fungsi f pada **Contoh 6.2,** yaitu $x^2 + e^{-x} - 10$, menggunakan metode Newton. Untuk menyelesaikan permasalahan tersebut, diperukan satu fungsi **fa_prime.sce** dan satu *script* **akar_fa_Newton.sce**. Kode program **fa_prime.sce** dan **akar_fa_Newton.sce** berturut-turut disajikan dalam Kode program 6.4 dan Kode program 6.5.

Kode program 6.4. (fa_prime.sce)

```
//Kode program 6.4. fa_prime.sce
function y = fa_prime(x)
   y = 2*x - exp(-x);
endfunction
//Akhir kode program
```

Kode program 6.4 berupa fungsi yang merepresentasikan turunan fungsi f, yang diberikan oleh $f'(x) = 2x - e^{-x}$.

Kode program 6.5. (akar_fa_Newton.sce)

```
//Kode program 6.5. akar_fa_Newton.sce
clear; // Menghapus semua variabel dari jendela Variable View
clc; // Membersihkan jendela Scilab editor
exec('fa.sce'); // Membuka/menjalankan file fa.sce
exec('fa_prime.sce'); // Membuka/menjalankan file fa_prime.sce
format('v',18); // Format tampilan 18 digit desimal

xlama = 1; //Nilai awal a, misalkan a = 0
xakar = [ ]; // Vektor untuk menyimpan hampiran nilai akar
galat = 1; // Nilai awal galat, misal galat = 1
iterasi = 0; // Nilai awal iterasi
maks_iterasi = 100; // Maksimal banyaknya iterasi
maks_galat = 5e-16; // Maksimal nilai galat yang diijinkan

sa = "Penentuan akar persamaan f(x) = x^2 + exp(-x) - 10 = 0";
sb = "Nilai awal akar a = " +  string(xlama) + ", f(a) = " + string(fa(xlama));
disp(sa);
disp(sb);

// Proses iterasi pada metode Newton
while (galat > maks_galat) & (iterasi <= maks_iterasi)
   // Jika nilai turunan f adalah nol, tampilkan pesan kesalahan/error
   if fa_prime(xlama)== 0
      error('Terjadi pembagian dengan nol')
   else
      x = xlama - fa(xlama)/fa_prime(xlama);
      galat = abs(x - xlama);
      iterasi = iterasi + 1;
      xlama = x;
      xakar = [xakar; x];
      s1 = "Iterasi ke " + string(iterasi) + ", ";
      s2 ="x = " + string(x) + ", ";
      s3 = "f(x) = " + string(fa(x));
      disp(s1 + s2 + s3);
   end
end
//Akhir kode program
```

Untuk menjalankan kode program 6.5 tersebut, dapat dilakukan klik menu *Execute – Save and execute* atau dengan mengetikkan perintah

exec('akar_fa_Newton.sce',-1)

pada jendela Scilab Console. Hasil eksekusi kode program 6.5 adalah sebagai berikut.

Penentuan akar persamaan f(x) = x^2 + exp(-x) - 10 = 0
Nilai awal akar a = 1, f(a) = -8.632120558828557
Iterasi ke 1, x = 6.288898857461974, f(x) = 29.55210564281332
Iterasi ke 2, x = 3.939006642574396, f(x) = 5.535240873617662
Iterasi ke 3, x = 3.234647203118302, f(x) = 0.502316622188511
Iterasi ke 4, x = 3.15652545389581, f(x) = 0.006226348754662
Iterasi ke 5, x = 3.155532491437283, f(x) = 0.000001006969539
Iterasi ke 6, x = 3.155532330796351, f(x) = 0.000000000000027
Iterasi ke 7, x = 3.155532330796346, f(x) = -0.000000000000002
Iterasi ke 8, x = 3.155532330796347, f(x) = 0.000000000000002

7. Grafik Dua Dimensi dan Tiga Dimensi

7.1. Pengantar

Penyajian grafik merupakan salah satu cara efektif dalam penyajian suatu informasi. Perangkat lunak Scilab menyediakan banyak fitur penyajian grafik, termasuk grafik dua dimensi dan grafik tiga dimensi. Pengguna Scilab juga dapat menambahkan fitur *title* (judul grafik) dan *legend* untuk membedakan beberapa grafik yang disajikan pada satu koordinat. Gambar grafik yang diperoleh juga dapat diekspor ke dalam format file gambar yang lain, seperti format PNG dan EPS.

Perangkat lunak Scilab dapat menghasilkan plot dua dimensi melalui perintah *plot*, menghasilkan plot kontur suatu fungsi dengan dua variabel bebas melalui perintah *contour*, dan menghasilkan plot tiga dimensi melalui perintah *surf*. Perintah yang umum digunakan terkait plot grafif fungsi, disajikan pada Tabel 7.1.

Tabel 7.1. Perintah plot grafik yang umum digunakan pada perangkat lunak Scilab

Sintaks	Deskripsi
plot	Plot 2D (plot grafik dua dimensi)
plot2d	Plot 2D (plot grafik dua dimensi). Perintah ini serupa dengan sintaks perintah *plot*.
contour	Plot kontur dari suatu fungsi dengan dua variabel bebas
Pie	Diagram lingkaran
histplot	Plot histogram
bar	Diagram batang
barh	Diagram batang horizontal
hist3d	Plot histrogram 3D (tiga dimensi)
polarplot	Plot grafik yang disajikan dalam koordinat polar
matplot	Plot 2D dari suatu matriks
Sgrayplot	Plot 2D secara halus (*smooth*) suatu permukaan (*surface*) menggunakan fitur warna
grayplot	Plot 2D suatu permukaan menggunakan fitur warna

7.2. Plot Grafik Dua Dimensi

Perintah *plot* atau *plot2d* digunakan untuk menghasilkan plot grafik dua dimensi. Penggunaan perintah plot dapat mengikuti sintaks berikut:

(a) plot(y) atau plot2d(y)
(b) plot(x, y) atau plot2d(x, y)
(c) plot(x, “fungsi”) atau plot2d(x, “fungsi”)

Pada sintaks (a), y merupakan suatu vektor baris atau vector kolom. Pada sintaks (b), x dan y merupakan vektor dengan dimensi/ukuran yang sama. Pada sintaks (c), parameter “fungsi” merupakan nilai fungsi yang dievaluasi pada vektor x.

Dengan menggunakan perintah plot, pengguna perangkat lunak Scilab dapat mengatur warna grafik, jenis corak garis dan simbol/penanda titik kurva. Sintaks sederhana perintah plot yang dilengkapi dengan pengaturan ketiga fitur grafik tersebut adalah

plot(y, [string_style])

atau

plot(x,y, [string_style])

Pada sintaks tersebut, argumen [string_style] merupakan string yang digunakan untuk mengatur warna grafik, jenis corak garis dan simbol/penanda titik kurva. Apabila argument [string_style] tidak ada, maka akan digunakan format *default* (standar) dalam penyajian grafik. Kode string untuk pengaturan warna grafik, jenis corak grafik dan symbol/penanda titik kurva pada perintah plot berturut-turut disajikan pada Tabel 7.2, Tabel 7.3, dan Tabel 7.4.

Tabel 7.2. Kode string untuk pengaturan warna grafik pada perintah plot

String	Warna garis	String	Warna garis
k	Hitam	c	Cyan
r	Merah	m	Magenta
g	Hijau	y	Kuning
b	Biru	w	Putih

Tabel 7.3. Kode string untuk pengaturan jenis corak garis pada perintah plot

String	Jenis corak garis
-	Garis penuh (*solid line*)
--	Garis putus-putus (*dashed line*)
:	Garis bentuk titik-titik (*dotted line*)
-.	Garis putus-putus dan titik (*dash-dotted line*)

Tabel 7.4. Kode string untuk pengaturan simbol/penanda titik kurva pada perintah plot

String	Simbol	String	Simbol
+	+	d	◇
o (Huruf o)	o	^	Δ
*	*	v	▽
· (Tanda titik)	.	>	▷
x	x	<	◁
's' atau 'square'	□	'pentagram'	☆

Contoh penggunaan perintah *plot* untuk menghasilkan plot grafik dua dimensi, disajikan pada Kode program 7.1.

Kode Program 7.1. (plot_grafik_fb.sce)

```
clear; // Menghapus semua variabel dari jendela Variable View
clc; // Membersihkan jendela Scilab editor
xdel(winsid()); // close('all') menutup semua figure
```

```
n = [1:35].';
y1 = (1 + 1 ./ n).^n;

figure(0);
plot(y1,'k',"linewidth",3);
title('Plot (1 + 1/n)^n terhadap n')
xlabel('n');
ylabel('y(n) = (1 + 1/n)^n');
gca().grid = [1 1 1]; // grid on
xs2png(0,"Gambar_7_1a.png");
xs2eps(0,"Gambar_7_1a.eps");

xx = (-3:0.1:3).';
y2 = exp(- xx.^2/2);

// Mendefinisikan function fb
function y = fb(x)
    y = 1 ./ (1 + x.^2);
endfunction

scf(1);
plot(xx,y2,'k',"linewidth",3);
set(gca(),"auto_clear","off"); // hold on, menahan agar grafik tidak terhapus
plot(xx,fb(xx), 'k:',"linewidth",3);
title('Grafik y = exp(-x^2/2) dan y = 1/(1+x^2)');
xlabel('x');
ylabel('y');
legend('exp(-x^2/2)','1/(1+x^2)');
gca().grid = [1 1 1]; // grid on
gca().x_location ="origin";
gca().y_location ="origin";
xs2png(1,"Gambar_7_2a.png");
xs2eps(1,"Gambar_7_2a.eps");
// Akhir kode program 7.1 (plot_grafik_fb.sce)
```

Kumpulan perintah

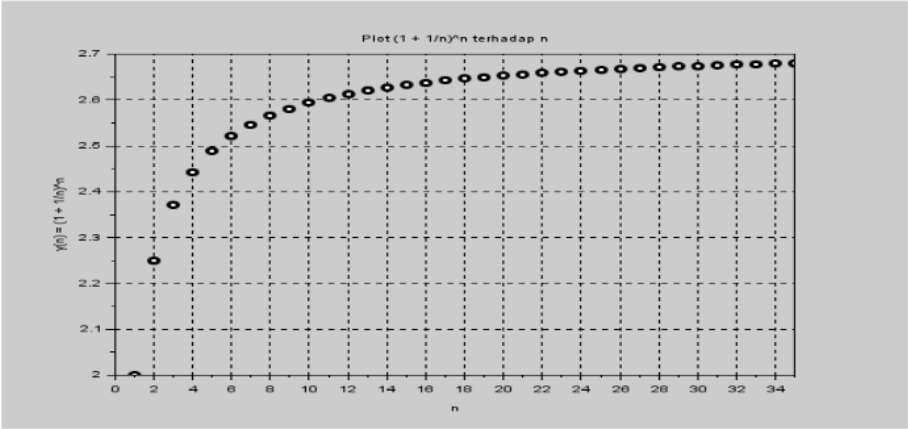
```
n = [1:35].';
y1 = (1 + 1 ./ n).^n;
figure(0);
plot(y1,'k',"linewidth",3);
title('Plot (1 + 1/n)^n terhadap n')
xlabel('n');
ylabel('y(n) = (1 + 1/n)^n');
gca().grid = [1 1 1]; // grid on, menghasilkan grid pada grafik
xs2png(0,"Gambar_7_1a.png");
```

xs2eps(0,"Gambar_7_1a.eps");

pada kode program 7.1 digunakan untuk menghasilkan plot $\left(1+\frac{1}{n}\right)^n$ terhadap n, dengan nilai n adalah $n = 1,2,\ldots,35$. Deskripsi perintah pada kode program tersebut disajikan sebagai berikut:

(a) Perintah menghasilkan **figure(0)** menghasilkan satu jendala grafik bernomor 0 (*graphic window number 0*).
(b) Perintah **plot(y1,'ko',"linewidth", 3)** mengasilkan plot vektor y1 terhadap barisan bilangan 1,2, …, length(y1), dengan length(y1) adalah banyaknya elemen vektor y1. Fitur **linewidth** digunakan untuk mengatur ketebalan garis pada file gambar.
(c) Perintah **title, xlabel** dan **ylabel** berturut-turut digunakan untuk menghasilkan judul grafik, label pada sumbu x dan sumbu y.
(d) Perintah **gca().grid = [1 1 1]** digunakan untuk menghasilkan grid pada file gambar.
(e) File gambar dapat diekspor ke dalam format PNG dan EPS menggunakan perintah **xs2png** dan **xs2eps**. Perintah **xs2png(0,"Gambar_7_1a.png")** digunakan untuk mengekspor file gambar pada jendala grafik bernomor 0 ke dalam file gambar format PNG dengan nama Gambar_7.1a.png.

Plot grafik plot $\left(1+\frac{1}{n}\right)^n$ terhadap n, dengan nilai n adalah $n = 1,2,\ldots,35$ disajikan pada Gambar 7.1.



Gambar 7.1. Plot $y(n) = \left(1+\frac{1}{n}\right)^n$ terhadap n; , $n = 1,2,\ldots,35$.

Blok perintah

```
xx = (-3:0.1:3).';
y2 = exp(-xx.^2/2);

// Mendefinisikan function fb
function y = fb(x)
    y = 1 ./ (1 + x.^2);
endfunction

scf(1);
```

```
plot(xx,y2,'k',"linewidth",3);
set(gca(),"auto_clear","off"); // hold on
plot(xx,fb(xx), 'k:',"linewidth",3);
title('Grafik y = exp(-x^2/2) dan y = 1/(1+x^2)');
xlabel('x');
ylabel('y');
legend(['exp(-x^2/2)','1/(1+x^2)'],1);
gca().grid = [1 1 1]; // grid on
gca().x_location ="origin";
gca().y_location ="origin";
xs2png(1,"Gambar_7_2a.png");
xs2eps(1,"Gambar_7_2a.eps");
```

pada kode program 7.1 digunakan untuk menghasilkan plot $y = \exp\left(\frac{-x^2}{2}\right)$ dan $y = \frac{1}{1+x^2}$ terhadap x, $-3 \leq x \leq 3$. Deskripsi perintah pada kode program tersebut disajikan sebagai berikut:

(a) Perintah menghasilkan **scf(1)** menghasilkan satu jendala grafik bernomor 1 (*graphic window number 1*).

(b) Perintah **plot(xx,y2,'k',"linewidth",3)** mengasilkan plot vektor y2 terhadap vector xx. Perintah **set(gca(),"auto_clear","off")** digunakan agar gambar yang ada pada jendela grafik bernomor 1, tidak terhapus.

(c) Perintah **plot(xx,fb(xx),'k:',"linewidth",3)** mengasilkan plot vektor fb(xx) terhadap vector xx.

(d) Perintah **gca().x_location ="origin"** dan **gca().y_location ="origin"** berturut-turut digunakan untuk meletakkan garis sumbu koordinat pada posisi baku (garis y = 0 dan garis x = 0).

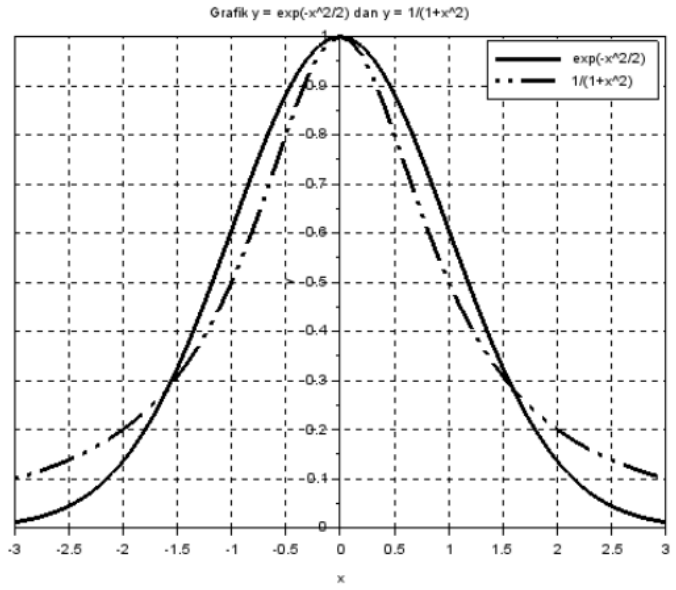
(e) Perintah **legend**(['exp(-x^2/2)','1/(1+x^2)'],1) digunakan untuk memberikan keterangan pada gambar, terutama ketika terdapat lebih dari satu grafik pada satu gambar. Nilai argumen 1 menunjukkan bahwa keterangan gambar terletak pada bagian kanan atas grafik. Nilai agumen 2, 3, 4 berturut-turut digunakan untuk meletakkan keterangan gambar pada bagian kiri atas, kanan bawah, dan kiri bawah grafik.

(f) Perintah **xs2png(1,"Gambar_7_2a.png")** digunakan untuk mengekspor file gambar pada jendala grafik bernomor 1 ke dalam file gambar format PNG dengan nama Gambar_7.2a.png.

Catatan penting:

Selain perintah **figure()**, perintah **scf()** juga dapat digunakan untuk mendefinisikan suatu figure pada perangkat lunak Scilab.

Grafik $y = \exp\left(\frac{-x^2}{2}\right)$ dan $y = \frac{1}{1+x^2}$ terhadap x, $-3 \leq x \leq 3$, disajikan pada Gambar 7.2.



Gambar 7.2. Grafik $y = \exp\left(\frac{-x^2}{2}\right)$ dan $y = \frac{1}{1+x^2}$ terhadap x, $-3 \leq x \leq 3$.

7.3. Perintah plot2d

Perintah plot2d juga dapat digunakan untuk menghasilkan grafik dua dimensi. Sintaks perintah plot2d adalah

plot2d([tipe_skala], [x], y, [optional_argument])

Argumen *tipe_skala* digunakan untuk mengatur tipe skala sumbu koordinat. Ada 4 pilihan tipe skala sumbu koordinat yang disajikan pada Tabel 7.5.

Tabel 7.5. Pilihan tipe skala sumbu koordinat pada perintah plot2d

Sintaks	Deskripsi
'nn'	Sumbu horizontal dan sumbu vertikal menggunakan skala normal
'nl'	Sumbu horisontal menggunakan skala normal dan sumbu vertikal menggunakan skala logaritmik
'ln'	Sumbu horisontal menggunakan skala logaritmik dan sumbu vertikal menggunakan skala normal
'll'	Sumbu horisontal dan sumbu vertikal menggunakan skala logaritmik

Bila argument [tipe_skala] tidak disebutkan secara spesifik, maka perangkat lunak Scilab menggunakan skala normal untuk sumbu horisontal dan sumbu vertikal.

Argumen opsional perintah plot2d meliputi *style, strf, leg, rect,* dan *nax*. Pengaturan warna kurva dan simbol/penanda titik kurva pada perintah plot2d menggunakan argument *style*.

Nilai argumen *style* dapat berupa bilangan bulat positif 1 – 32 atau 0, - 1, - 2, …, - 14. Nilai *style* berupa bilangan bulat dari 1,2, …, 32 digunakan untuk mengatur warna grafik. Daftar warna grafik *plot2d* yang bersesusaian dengan nilai argument *style*, disajikan pada Tabel 7.6.

Tabel 7.6. Pilihan warna pada perintah plot2d

Style	Warna	Style	Warna	Style	Warna	Style	Warna
1		9		17		25	
2		10		18		26	
3		11		19		27	
4		12		20		28	
5		13		21		29	
6		14		22		30	
7		15		23		31	
8		16		24		32	

Nilai *style* berupa bilangan bulat dari 0, - 1, - 2, …, - 14 digunakan untuk mengatur simbol/penanda titik kurva. Daftar simbol terhadap nilai argumen *style*, disajikan pada Tabel 7.7.

Tabel 7.7. Daftar simbol/penanda titik kurva

Style	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14
Simbol	•	+	×	⊕	♦	◇	▷	◁	✣	○	□	▫	▽	△	☆

Contoh penggunaan perintah *plot2d* untuk menghasilkan plot grafik dua dimensi, disajikan pada Kode program 7.2.

Kode Program 7.2. (plot2d_grafik.sce)

```
clear; // Menghapus semua variabel dari jendela Variable View
clc; // Membersihkan jendela Scilab editor
xdel(winsid()); // close('all') menutup semua figure

xa = (0:1:20).';
ya = 2.^xa;
za = 5.^xa;
```

```
scf(0);
plot2d('nl',xa,[ya,za], style = [-5,-11]);
xtitle('Log plot','x','y');
legend(['2^x','5^x'], 2);
gca().grid = [1 1 1]; // grid on
// Akhir kode program 7.2. (plot2d_grafik.sce)
```

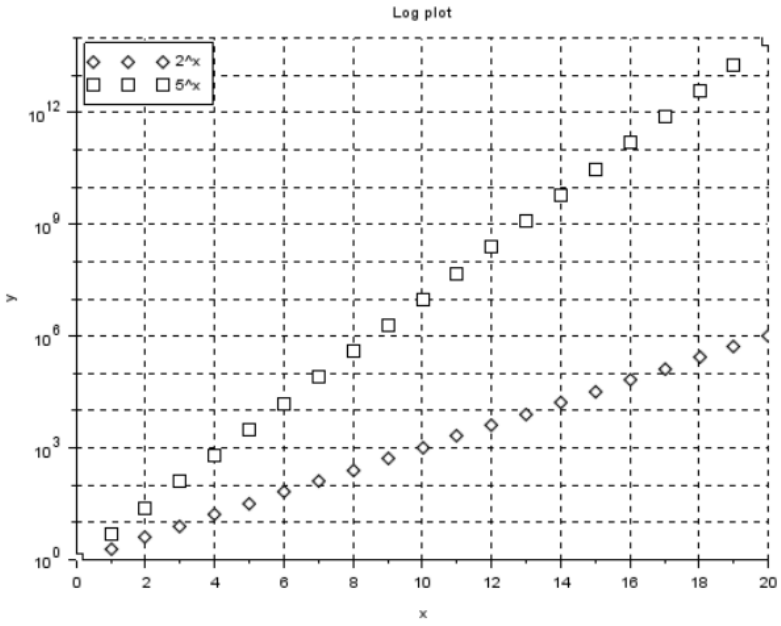
Deskripsi perintah pada kode program tersebut disajikan sebagai berikut:

(a) Perintah **plot2d('nl', xa, [ya, za], style = [-5,-11])** digunakan untuk menghasilkan plot grafik $y = 2^x$ dan $y = 2^x$ terhadap x, dengan nilai x = 0, 1, 2, …, 20. Argumen 'nl' digunakan untuk menghasilkan plot semilog pada sumbu vertikal.

(b) Perintah **xtitle('Log plot', 'x', 'y')** digunakan untuk menghasilkan 'Log plot' sebagai judul grafik, dan memberikan label 'x' dan 'y' berturut-turut sebagai label sumbu horisontal dan label sumbu vertikal.

(c) Perintah digunakan **legend(['2^x','5^x'], 2)** untuk menghasilkan keterangan gambar yang diletakkan pada bagian kiri atas gambar.

Grafik $y = 2^x$ dan $y = 2^x$ yang dihasilkan dari kode program 7.2 tersebut, disajikan pada Gambar 7.3.



Gambar 7.3. Grafik $y = 2^x$ dan $y = 5^x$ yang dihasilkan dari kode program 7.2

7.3. Plot Grafik Tiga Dimensi

Perintah *plot3d1* dapat digunakan untuk menggambar kurva ketinggian suatu fungsi dengan dua peubah bebas. Sintaks sederhana perintah *plot3d1* adalah *plot3d1(x,y,z)* dengan x, y merupakan suatu vector dan z merupakan matriks berukuran length(x)*length(y), dengan length(x)

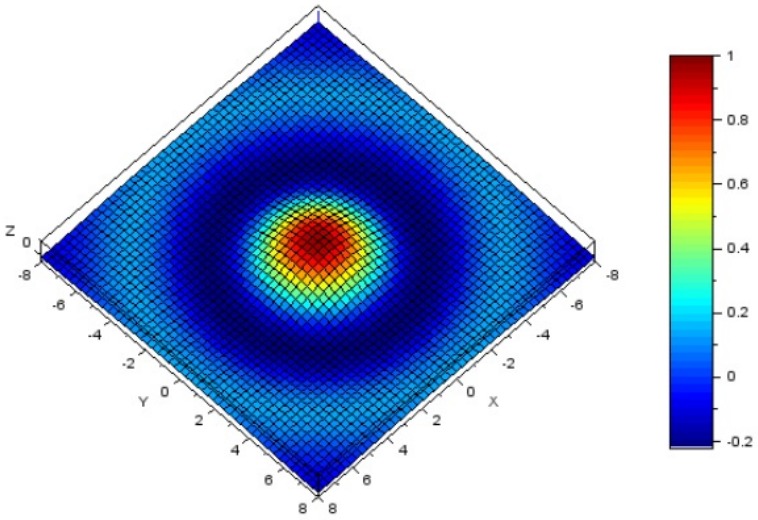
dan length(y) berturut-turut menyatakan banyaknya elemen vektor x dan vektor y. Plot permukaan kurva dengan dua variabel bebas dapat disajikan lebih informative menggunakan perintah *surf*. Sintaks perintah surf adalah *surf*(z) dengan z merupakan matriks berukuran length(x)*length(y), dengan length(x) dan length(y) berturut-turut menyatakan banyaknya elemen vektor x dan vektor y. Kurva ketinggian (peta kontur) dari suatu fungsi $z = f(x, y)$ dapat disajikan menggunakan perintah *contour*.

Kode program 7.3 berikut menyajikan blok perintah untuk menghasilkan kurva ketinggian dan kurva permukaan fungsi $z = \frac{\sin\left(\varepsilon+\sqrt{x^2+y^2}\right)}{\varepsilon+\sqrt{x^2+y^2}}$; $-8 \leq x, y \leq 8, \varepsilon \approx 2.22 * 10^{-16}$.

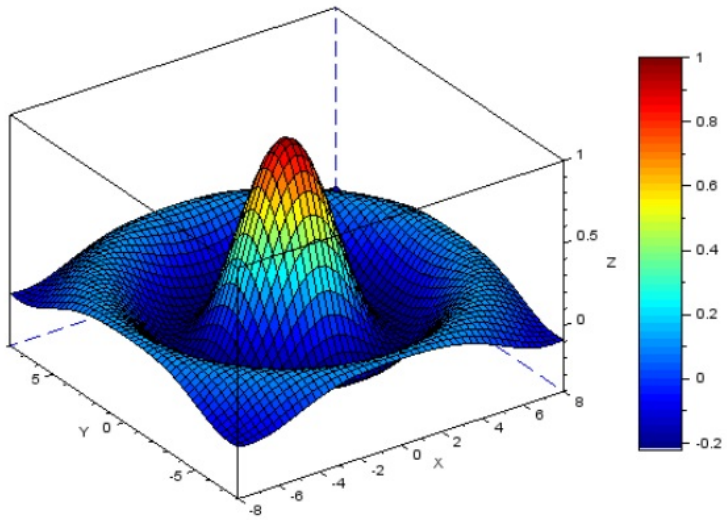
Kode Program 7.3. (grafik_plot3d.sce)

```
clear; // Menghapus semua variabel dari jendela Variable View
clc; // Membersihkan jendela Scilab editor
xdel(winsid()); // close('all') menutup semua figure

// Mendefinisikan function fc
function z = fc(xx, yy)
   r = %eps + sqrt(xx.^2 + yy.^2);
   z = sin(r) ./ r;
endfunction

xa  = linspace(-8, 8, 51).';
ya = linspace(-8, 8, 51).';
[xx,yy] = ndgrid(xa,ya);

scf(0);
plot3d1(xa,ya,fc(xx,yy));
f = scf(0);
f.color_map = jetcolormap(50);
colorbar(min(fc(xx,yy)),max(fc(xx,yy)));
xs2png(0,"Gambar_7_4.png");
xs2eps(0,"Gambar_7_4.eps");

scf(1);
surf(xa,ya,fc(xx,yy));
g = scf(1);
g.color_map = jetcolormap(50);
colorbar(min(fc(xx,yy)),max(fc(xx,yy)));
xs2png(1,"Gambar_7_5.png");
xs2eps(1,"Gambar_7_5.eps");
//Akhir kode program
```

Kurva ketinggian dan kurva permukaan fungsi $z = \frac{\sin\left(\varepsilon+\sqrt{x^2+y^2}\right)}{\varepsilon+\sqrt{x^2+y^2}}$ berturut-turut disajikan pada Gambar 7.4 dan Gambar 7.5.



Gambar 7.4. Grafik kurva ketinggian fungsi $z = \frac{\sin\left(\varepsilon+\sqrt{x^2+y^2}\right)}{\varepsilon+\sqrt{x^2+y^2}}$.



Gambar 7.5. Grafik kurva permukaan fungsi $z = \frac{\sin\left(\varepsilon+\sqrt{x^2+y^2}\right)}{\varepsilon+\sqrt{x^2+y^2}}$.

8. Pengantar Optimisasi Swarm Partikel

8.1. Pendahuluan

Model matematika merupakan suatu alat yang sangat berguna untuk mendeskripsikan berbagai permasalahan real. Ada empat tahap dalam pemodelan matematika, yaitu: identifikasi permasalahan real, konstruksi model matematika, penentuan solusi model matematika, dan interpretasi solusi model ke dalam sudut pandang permasalahan real. Suatu model matematika dapat berbentuk model optimasi, sistem persamaan, sistem persamaan diferesial, atau model stokastik. Ketika tersedia data relevan dari fenomena real, maka data tersebut dapat digunakan untuk memvalidasi keakuratan (akurasi) model. Jika hasil simulasi dari model matematika bersesuaian dengan data real, maka model matematika tersebut merupakan model yang baik. Sebaliknya, jika hasil prediksi dari model berbeda secara signifikan dengan data real, maka model matematika tersebut harus diperbaiki [4].

Kebanyakan model matematika memuat satu atau lebih parameter. Untuk dapat melakukan simulasi suatu model matematika secara akurat, parameter dalam model perlu diestimasi (diperkirakan). Ketika solusi matematis dari suatu model matematika dapat diperoleh dalam bentuk eksplisit (*closed-form*), maka parameter dalam model dapat diestimasi dengan metode optimasi deterministik seperti metode Nelder-Mead atau metode Newton [5]. Sayangnya, metode optimasi deterministik seperti Nelder-Mead atau metode Newton gagal konvergen ke nilai minimum dari suatu fungsi jika fungsi tersebut memiliki banyak titik minimum lokal [6]. Selain itu, pada sebagian model matematika yang berbentuk persamaan diferensial biasa (sistem persamaan diferensial biasa) non linear, solusi analitis tidak dapat atau sulit diperoleh. Dalam hal ini, metode heuristik seperti metode algoritma genetika dan optimisasi *swarm* partikel dapat digunakan untuk memperkirakan nilai parameter dari suatu persamaan diferensial non linear atau sistem persamaan diferensial non linear.

Optimisasi *swarm* partikel merupakan metode optimasi berbasis proses pencarian stokastik yang terinspirasi dari populasi makhluk hidup [7, 8]. Metode ini dikembangkan oleh Eberhart dan Kennedy pada tahun 1995. Sebagaimana algoritma genetika, salah satu keuntungan utama metode optimisasi *swarm* partikel dibandingkan metode konvensional adalah bahwa metode optimisasi *swarm* partikel tidak memerlukan evaluasi gradient fungsi tujuan. Optimisasi *swarm* partikel merupakan metode metaheuristik berbasis populasi yang dapat digunakan untuk menemukan hampiran solusi optimal atau hampiran solusi suboptimal (solusi yang yang mendekati solusi optimal) untuk masalah optimasi yang sulit. Metode ini terinspirasi oleh perilaku sosial berkelompok burung, segerombolan ikan atau segerombolan serangga.

Metode optimisasi *swarm* partikel dan modifikasi *swarm* partikel telah diterapkan dalam berbagai bidang, termasuk pningkatan performansi jaringan syaraf tiruan [9, 10], penyelesaian masalah penjadwalan [11, 12], masalah penjadwalan *flowshop* [13], masalah perjalanan sales (*traveling salesman problem)* [14], penyelesaian masalah rute perjalanan kendaraan (*vehicle routing problem*) [15, 16] dan masalah teknik pengelompokan (*clustering technique*) [17]. Pada bagian selanjutnya, akan disajikan secara ringkas prosedur optimisasi swarm partikel.

8.2. Prosedur Optimisasi Swarm Partikel

Masalah optimasi dapat dibedakan menjadi masalah penentuan nilai maksimum dan masalah penentuan nilai minimum suatu fungsi tujuan (*objective function*). Masalah penentuan

nilai maksimum suatu fungsi tujuan dapat ditransformasi menjadi masalah penentuan nilai minimum. Sebagai contoh, penentuan nilai maksimum $f(x) = 4 + 2x - x^2, -2 \leq x \leq 2$ adalah ekivalen dengan penentuan nilai minimum $g(x) = -f(x) = x^2 - 2x - 4,\ -2 \leq x \leq 2$. Oleh karena itu, dalam tulisan ini hanya ditinjau masalah optimasi berbentuk masalah penentuan nilai minimum suatu fungsi tujuan.

Misalkan diberikan masalah optimasi yang berbentuk

$$\min z = f(y), y \in \Omega \subseteq R^d, z \in R. \quad (8.1)$$

Masalah optimasi pada persamaan (8.1) merupakan masalah optimasi dengan d dimensi (variabel keputusan y berada dalam R^d). Himpunan Ω merupakan ruang solusi dan fungsi $f: \Omega \subseteq R^d \rightarrow R$ merupakan fungsi tujuan pada masalah optimasi tersebut. Pengertian solusi optimal dan solusi suboptimal disajikan sebagai berikut:

(a) Solusi $y^* \in \Omega$ merupakan solusi optimal masalah optimasi pada persamaan (8.1) tersebut jika untuk setiap $y \in \Omega, \mathrm{f}(y^*) \leq f(y)$.

(b) Solusi $y^* \in \Omega$ merupakan solusi suboptimal masalah optimasi pada persamaan (8.1) tersebut jika untuk setiap y dalam persekitaran (*neighborhood*) dari y^* ($y \in N_\delta(y^*) \subseteq \Omega), \mathrm{f}(y^*) \leq f(y)$. Secara formal, persekitaran dari y^* ($N_\delta(y^*)$) didefinisikan sebagai himpunan titik yang berjarak kurang dari δ terhadap y^*, dan dinotasikan dengan $N_\delta(y^*) = \{u \in \Omega: \|u - y^*\| < \delta\}$.

Pada sebagaian masalah optimasi (terutama masalah optimasi yang kompleks), solusi optimal masalah optimasi tersebut sulit ditentukan. Dalam kondisi ini, solusi suboptimal pun seringkali sudah relatif memadai.

Pada metode optimisasi swarm partikel, suatu solusi direpresentasikan dengan posisi suatu partikel. Metode ini diawali dengan memilih sejumlah solusi awal secara acak (misalkan $npartikel$ solusi acak) dari ruang solusi. Selanjutnya, dilakukan evaluasi (penghitungan) nilai fungsi tujuan untuk setiap solusi (posisi partikel). Kemudian, dilakukan update setiap solusi (posisi setiap partikel). Update solusi ke-i (i = 1, 2, …, npartikel) dilakukan dengan memperhatikan posisi partikel terbaik lokal p_i (posisi terbaik partikel satu partikel ke-i) dan posisi partikel terbaik global g (posisi terbaik dari seluruh partikel) pada setiap iterasi. Posisi partikel p_i diupdate jika posisi partikel ke-i pada iterasi terbaru suatu partikel mempunyai nilai fungsi tujuan yang lebih kecil dari posisi partikel p_i . Dengan cara serupa, posisi partikel g akan diupdate jika terdapat posisi suatu partikel pada iterasi terbaru yang memiliki nilai fungsi tujuan yang lebih kecil dari posisi partikel g.

Berikut merupakan tahapan metode optimisasi swarm partikel untuk memperoleh hampiran solusi optimal suatu masalah optimasi pada persamaan (8.1) adalah [8]:

(1) Bangkitkan sebanyak $npartikel$ vektor yang merepresentasikan solusi (posisi partikel) awal secara acak (misalkan solusi acak) dari ruang solusi. Bangkitkan juga sebanyak $npartikel$ vektor yang merepresentasikan kecepatan awal partikel. Misalkan
$x_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,d}), v_i = (v_{i,1}, v_{i,2}, \ldots, v_{i,d})$
berturut-turut menyatakan posisi partikel ke-i dan kecepatan partikel ke-i. Untuk tahap awal (inisialisasi) vektor kecepatan v_i dapat berupa vektor nol untuk $i = 1,2, \ldots, npartikel$.

(2) Hitunglah nilai fungsi tujuan untuk setiap solusi (posisi partikel).

(3) Lakukan update posisi partikel p dan g.

(4) Lakukan update kecepatan partikel melalui persamaan

$$v_i^{k+1} = wv_i^k + c_1 r_1 \odot \left(p_i^k - x_i^k\right) + c_2 r_2 \odot \left(g - x_i^k\right);\ i = 1,2,\dots,npartikel. \quad (8.2)$$

Pada persamaan (8.2), r_1, r_2 adalah vector acak dalam R^d dengan komponen bilangan acak berdistribusi seragam antara 0 dan 1. Operator $\odot$ menyatakan operator perkalian-tiap-elemen.

(5) Lakukan update posisi partikel melalui persamaan

$$x_i^{k+1} = x_i^k + v_i^{k+1}; i = 1,2,\dots,npartikel. \quad (8.3)$$

Tahapan-tahapan tersebut diulang sampai kondisi pemberhentian metode optimisasi *swarm* partikel terpenuhi.

Pada persamaan (8.2), parameter w, c_1, c_2 berturut-turut adalah bobot inersia, koefisien kognitif dan koefisien sosial. Nilai parameter yang umum digunakan untuk ketiga parameter tersebut adalah $w = 1, c_1 = c_2 = 2$ [6, 18]. Untuk mencegah agar partikel (solusi) bergerak sangat jauh dari rentang yang ditentukan (meninggalkan ruang solusi), dapat digunakan teknik pembatasan nilai kecepatan partikel (*velocity clamping*). Untuk ruang solusi yang terbatas pada rentang $[x_{min}, x_{maks}]$, nilai kecepatan dibatasi pada rentang $[-v_{maks}, v_{maks}]$ dengan $v_{maks} = m(x_{maks} - x_{min})$ untuk suatu konstanta $m, 0.1 \leq m \leq 1$. Kondisi pemberhentian metode optimisasi swarm partikel meliputi salah satu dari beberapa kondisi berikut [23]:

(a) Banyaknya iterasi telah mencapai iterasi maksimum.
(b) Nilai solusi (posisi partikel) terbaik global g tidak mengalami perubahan setelah sejumlah iterasi.
(c) Nilai fungsi tujuan dari solusi (posisi partikel) terbaik global g telah mencapai nilai tertentu yang telah ditentukan sebelumnya.

Pada masalah optimisasi satu dimensi, metode optimisasi *swarm* partikel konvergen ke titik optimum lokal untuk berbagai fungsi tujuan (*objective function*) yang relatif luas. Dalam masalah optimasi multidimensi, ternyata metode optimisasi swarm partikel dimungkinkan tidak konvergen ke titik optimum lokal [19]. Konvergensi metode optimisasi *swarm* partikel ke titik optimum lokal disebut konvergensi prematur. Konvergensi prematur umumnya disebabkan oleh penurunan nilai kecepatan partikel dalam ruang solusi. Penurunan nilai kecepatan partikel tersebut menyebabkan stagnasi nilai fungsi tujuan dari swarm partikel [20]. Oleh karena itu, metode optimisasi swarm partikel harus diimplementasikan berkali-kali untuk mendapatkan solusi terbaik [21].

8.3. Manual Metode Optimisasi Swarm Partikel

Pada bagian ini akan dijelaskan secara manual implementasi metode optimisasi swarm partikel untuk satu iterasi dalam menentukan nilai minimum fungsi Alpine dua dimensi, yang diberikan oleh

$$f(x_1, x_2) = |x_1 \sin x_1 + 0.1x_1| + |x_2 \sin x_2 + 0.1x_2|, -5 \leq x_1, x_2 \leq 5. \quad (8.4)$$

Parameter metode optimisasi swarm partikel yang digunakan adalah $w = 1, c_1 = c_2 = 2$. Nilai delapan solusi awal $y = (x_1, x_2)$ disajikan pada Tabel 8.1.

Tabel 8.1. Nilai delapan solusi awal dan nilai fungsi Alpine yang bersesuaian

Indeks solusi (i)	x_1	x_2	$f(y) = f(x_1, x_2)$
1	-2.88675	3.78216	2.32126
2	2.56044	-4.31626	6.07534
3	-4.99779	0.60849	5.70408
4	-1.69673	1.62357	3.29729
5	1.65381	2.26351	3.78165
6	1.28392	-3.01486	1.43941
7	3.49745	0.44257	1.10254
8	1.85731	-2.67925	2.89447
Rata-rata nilai fungsi Alpine			3.32700

Karena nilai variabel keputusan x_1, x_2 terletak dalam interval tutup $[-5,5]$ maka $y_{min} = (-5,-5), y_{maks} = (5,5)$. Batas bawah kecepatan dan batas atas kecepatan partikel berturut-turut diberikan oleh

$$v_{min} = -m\big(5-(-5), 5-(-5)\big) = (-10m, -10m),$$

$$v_{maks} = m\big(5-(-5), 5-(-5)\big) = (10m, 10m).$$

Dengan memilih nilai $m = 0.1$, diperoleh $v_{min} = (-1,-1), v_{maks} = (1,1)$. Berikut disajikan penghitungan nilai solusi (posisi partikel) secara manual sampai dengan iterasi pertama:

(a) Iterasi ke-0 (iterasi awal)

Vektor kecepatan partikel pada iterasi awal dipilih berupa vektor nol, sehingga

$$v_1^0 = v_2^0 = v_3^0 = v_4^0 = v_5^0 = v_6^0 = v_7^0 = v_8^0 = (0,0).$$

Posisi terbaik lokal sama dengan posisi partikel pada saat awal, sehingga

$p_1^0 = y_1^0 =$ (-2.88675, 3.78216), $p_2^0 = y_2^0 =$ (2.56044, -4.31626),
$p_3^0 = y_3^0 =$ (-4.99779, 0.60849), $p_4^0 = y_4^0 =$ (-1.69673, 1.62357),
$p_5^0 = y_5^0 =$ (1.65381, 2.26351), $p_6^0 = y_6^0 =$ (1.28392, -3.01486),
$p_7^0 = y_7^0 =$ (3.49745, 0.44257), $p_8^0 = y_8^0 =$ (1.85731, -2.67925).

Berdasarkan Tabel 8.1, posisi partikel ke-7 merupakan posisi partikel dengan nilai fungsi tujuan paling kecil, sehingga diperoleh posisi terbaik global $g^0 = p_7^0 = (3.49745, 0.44257)$.

(b) Iterasi ke-1 (iterasi pertama)

Misalkan diperoleh vektor acak $r_1^0 = (0.2, 0.2), r_2^0 = (0.9, 0.7)$.

Akan dilakukan update kecepatan partikel dan posisi untuk setiap solusi (posisi partikel).

(b.1) Update kecepatan partikel pertama dilakukan dengan menggunakan persamaan (8.2), sehingga diperoleh

$v_1^1 = v_1^0 + 2r_1^0 \odot (p_1^0 - y_1^0) + 2r_2^0 \odot (g - y_1^0)$, sehingga

$$v_1^1 = 2(0.2, 0.2) \odot (0,0) + 2(0.9, 0.7) \odot (6.3842, 3.33959) = (11.49156, -4.67543).$$

Karena v_1^1 di luar rentang v_{min} dan v_{maks}, maka $v_1^1 = (1,-1)$. Dengan menggunakan persamaan (8.3), diperoleh

$$y_1^1 = y_1^0 + v_1^1 = (-2.88675, 3.78216) + (1,-1) = (-1.88675, 2.78216).$$

Nilai fungsi pada titik y_1^1 adalah $f(y_1^1) = 2.86150$. Karena $f(y_1^1) \geq f(p_1^0)$, maka $p_1^1 = p_1^0 = (-2.88675, 3.78216)$.

(b.2) Update kecepatan partikel kedua dilakukan dengan menggunakan persamaan (8.2), diperoleh

$v_2^1 = v_2^0 + 2r_1^0 \odot (p_2^0 - y_2^0) + 2r_2^0 \odot (g^0 - y_2^0)$, sehingga

$v_1^1 = 2(0.2, 0.2) \odot (0,0) + 2(0.9, 0.7) \odot (0.93701, 4.75883) = (1.686618, 6.662362)$.

Karena v_2^1 di luar rentang v_{min} dan v_{maks}, maka $v_2^1 = (1,1)$. Dengan menggunakan persamaan (8.3), diperoleh

$$y_2^1 = y_2^0 + v_2^1 = (2.56044, -4.31626) + (1,1) = (3.56044, -3.31626).$$

Nilai fungsi pada titik y_2^1 adalah $f(y_2^1) = 1.99994$. Karena $f(y_2^1) < f(p_2^0)$, maka $p_2^1 = y_2^1 = (3.56044, -3.31626)$.

(b.3) Update kecepatan partikel ketiga dilakukan dengan menggunakan persamaan (8.2), diperoleh

$v_3^1 = v_3^0 + 2r_1^0 \odot (p_3^0 - y_3^0) + 2r_2^0 \odot (g - y_3^0)$. Akibatnya,

$v_3^1 = 2(0.2, 0.2) \odot (0,0) + 2(0.9, 0.7) \odot (8.49524, -0.16592) = (15.29143, -0.23229)$.

Karena v_3^1 di luar rentang v_{min} dan v_{maks}, maka $v_3^1 = (1, 0.23229)$. Dengan menggunakan persamaan (8.3), diperoleh

$$y_3^1 = y_3^0 + v_3^1 = (-4.99779, 0.60849) + (1, -0.23229) = (-3.99779, 0.37620).$$

Nilai fungsi pada titik y_3^1 adalah $f(y_3^1) = 3.59537$. Karena $f(y_3^1) < f(p_3^0)$, maka $p_3^1 = y_3^1 = (-3.99779, 0.37620)$.

(b.4) Update kecepatan partikel keempat dilakukan dengan menggunakan persamaan (8.2), diperoleh

$v_4^1 = v_4^0 + 2r_1^0 \odot (p_4^0 - y_4^0) + 2r_2^0 \odot (g - y_4^0)$. Akibatnya,

$v_4^1 = 2(0.2, 0.2) \odot (0,0) + 2(0.9, 0.7) \odot (5.19418, -1.18100) = (9.349524, -1.6534)$.

Karena v_4^1 di luar rentang v_{min} dan v_{maks}, maka $v_4^1 = (1,-1)$. Dengan menggunakan persamaan (8.3), diperoleh

$$y_4^1 = y_4^0 + v_4^1 = (-1.69673, 1.62357) + (1,-1) = (-0.69673, 0.62357).$$

Nilai fungsi pada titik y_4^1 adalah $f(y_4^1) = 0.80391$. Karena $f(y_4^1) < f(p_4^0)$, maka $p_4^1 = y_4^1 = (-0.69673, 0.62357)$. Selain itu, karena $f(y_4^1) < f(g)$, maka $g = y_4^1 = (-0.69673, 0.62357)$.

(b.5) Update kecepatan partikel kelima dilakukan dengan menggunakan persamaan (8.2), diperoleh

$v_5^1 = v_5^0 + 2r_1^0 \odot (p_5^0 - y_5^0) + 2r_2^0 \odot (g - y_5^0)$. Akibatnya

$v_5^1 = 2(0.2, 0.2) \odot (0,0) + 2(0.9, 0.7) \odot (-2.35054, -1.63994) = (-4.23097, -2.29592)$.

Karena v_5^1 di luar rentang v_{min} dan v_{maks}, maka $v_5^1 = (-1, -1)$. Dengan menggunakan persamaan (8.3), diperoleh

$$y_5^1 = y_5^0 + v_5^1 = (1.65381, 2.26351) + (-1, -1) = (0.65381, 1.26351).$$

Nilai fungsi pada titik y_5^1 adalah $f(y_5^1) = 1.79371$. Karena $f(y_5^1) < f(p_5^0)$, maka $p_5^1 = y_5^1 = (0.65381, 1.26351)$.

(b.6) Update kecepatan partikel keenam dilakukan dengan menggunakan persamaan (8.2), diperoleh

$v_6^1 = v_6^0 + 2r_1^0 \odot (p_6^0 - y_6^0) + 2r_2^0 \odot (g - y_6^0)$, sehingga

$v_6^1 = 2(0.2, 0.2) \odot (0,0) + 2(0.9, 0.7) \odot (-1.98065, 3.63843) = (-3.56517, 5.09380)$.

Karena v_6^1 di luar rentang v_{min} dan v_{maks}, maka $v_6^1 = (-1,1)$. Dengan menggunakan persamaan (8.3), diperoleh

$$y_6^1 = y_6^0 + v_6^1 = (1.28392, -3.01490) + (-1,1) = (0.28392, -2.01490).$$

Nilai fungsi pada titik y_2^1 adalah $f(y_6^1) = 1.72588$. Karena $f(y_6^1) \geq f(p_6^0)$, maka $p_6^1 = y_6^0 = (1.28392, -3.01486)$.

(b.7) Update kecepatan partikel ketujuh dilakukan dengan menggunakan persamaan (8.2), diperoleh

$v_7^1 = v_7^0 + 2r_1^0 \odot (p_7^0 - y_7^0) + 2r_2^0 \odot (g - y_7^0)$. Akibatnya

$v_1^1 = 2(0.2, 0.2) \odot (0,0) + 2(0.9, 0.7) \odot (-4.19418, 0.18100) = (-7.54952, 0.25340)$.

Karena v_7^1 di luar rentang v_{min} dan v_{maks}, maka $v_7^1 = (-1,0.25340)$. Dengan menggunakan persamaan (8.3), diperoleh

$$y_7^1 = y_7^0 + v_7^1 = (3.49745, 0.44257) + (-1, 0.25340) = (2.49745, 0.69597).$$

Nilai fungsi pada titik y_7^1 adalah $f(y_7^1) = 2.11327$. Karena $f(y_7^1) \geq f(p_7^0)$, maka $p_7^1 = p_7^0 = (3.49745, 0.44257)$.

(b.8) Update kecepatan partikel kedelapan dilakukan dengan menggunakan persamaan (8.2), diperoleh

$v_8^1 = v_8^0 + 2r_1^0 \odot (p_8^0 - y_8^0) + 2r_2^0 \odot (g - y_8^0)$, sehingga

$$v_8^1 = 2(0.2, 0.2) \odot (0,0) + 2(0.9, 0.7) \odot (-2.55404, 3.30282)$$
$$= (-4.59727, 4.623948).$$

Karena v_8^1 di luar rentang v_{min} dan v_{maks}, maka $v_8^1 = (-1{,}1)$. Dengan menggunakan persamaan (8.3), diperoleh

$$y_2^1 = y_2^0 + v_2^1 = (1.85731, -2.67925) + (-1{,}1) = (0.85731, -1.67925).$$

Nilai fungsi pada titik y_8^1 adalah $f(y_8^1) = 2.23539$. Karena $f(y_8^1) < f(p_8^0)$, maka $p_8^1 = y_8^1 = (0.85731, -1.67925)$.

Nilai solusi (posisi partikel) terbaik lokal setelah satu iterasi dan nilai fungsi Alpine yang dievaluasi pada solusi, disajikan pada Tabel 8.2.

Tabel 8.2. Nilai solusi (posisi partikel) terbaik lokal setelah satu iterasi dan nilai fungsi Alpine yang bersesuaian

Indeks solusi (i)	$y = (x_1, x_2)$		$f(y) = f(x_1, x_2)$
	x_1	x_2	
1	-2.88675	3.78216	2.32125
2	3.56044	-3.31626	1.99994
3	-3.99779	0.37620	3.59537
4	-0.69673	0.62357	0.80391
5	0.65381	1.26351	1.79371
6	1.28392	-3.01486	1.43941
7	3.49745	0.44257	1.10254
8	0.85731	-1.67925	2.23539
Rata-rata nilai fungsi Alpine			1.91144

Dari persamaan (8.2)-(8.3), implementasi metode optimisasi swarm partikel menggunakan vector acak berdistribusi seragam dalam interval $(0, 1)$. Hal ini mengakibatkan hasil implementasi metode swarm partikel bervariasi di sekitar nilai terbaik yang diperoleh dari implementasi metode tersebut. Pada Tabel 8.3 dan Tabel 8.4, disajikan nilai solusi (posisi partikel) terbaik lokal setelah seratus iterasi dari dua kali implementasi metode optimisasi swarm partikel.

Tabel 8.3. Nilai solusi (posisi partikel) terbaik lokal setelah seratus iterasi dan nilai fungsi Alpine yang bersesuaian (implementasi pertama)

Indeks solusi (i)	$y = (x_1, x_2)$		$f(y) = f(x_1, x_2)$
	x_1	x_2	
1	-0.01631	-0.10474	0.00184
2	-0.11257	0.00484	0.00190
3	-0.11743	-0.08351	0.00340
4	-0.01702	0.00955	0.00246
5	0.00291	-0.21997	0.02630
6	-0.11249	0.01307	0.00286
7	-0.01867	-0.07114	0.00358
8	-0.05400	-0.03908	0.00487
Median nilai fungsi Alpine			0.00313

Tabel 8.4. Nilai solusi (posisi partikel) terbaik lokal setelah seratus iterasi dan nilai fungsi Alpine yang bersesuaian (implementasi kedua)

Indeks solusi (i)	$y = (x_1, x_2)$		$f(y) = f(x_1, x_2)$
	x_1	x_2	
1	0.01905	-0.12074	0.00474
2	-0.08346	0.00521	0.00194
3	-0.08349	0.03387	0.00592
4	-0.10707	-0.16833	0.01210
5	-0.01709	-0.01174	0.00245
6	-0.03022	-0.05939	0.00452
7	-0.06209	-0.16917	0.01392
8	0.00133	-0.19800	0.01928
Median nilai fungsi Alpine			0.00533

Tabel 8.3 dan Tabel 8.4 mengindikasikan bahwa nilai solusi bergerak menuju solusi optimal yaitu $(0, 0)$ dengan nilai optimal fungsi adalah $f(0,0) = 0$.

9. Implementasi Optimisasi Swarm Partikel untuk Masalah Optimasi Sederhana

Dalam bab ini akan dibahas implementasi optimisasi swarm partikel untuk beberapa masalah optimasi sederhana, meliputi penentuan akar persamaan non linear, akar sistem persamaan non linear dan penentuan nilai maksimum suatu fungsi dua variabel.

9.1. Penentuan Akar Persamaan Non Linear

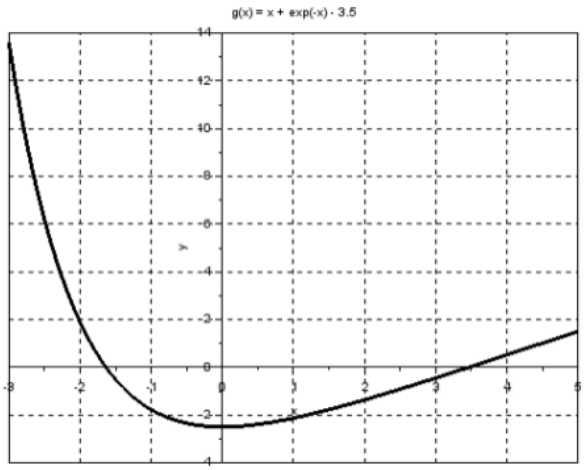
Penentuan akar suatu persamaan non linear dapat ditransformasi menjadi suatu masalah optimasi. Sebagai contoh, penentuan akar real persamaan non linear

$$x + e^{-x} = 3.5, \tag{9.1}$$

dapat ditransformasi menjadi masalah optimasi yang berbentuk

$$\min z = |x + e^{-x} - 3.5|, x \in R. \tag{9.2}$$

Eksistensi akar suatu persamaan dapat dideteksi melalui grafik fungsi yang bersesuaian dengan persamaan tersebut. Grafik fungsi $g, g(x) = x + e^{-x} - 3.5, -3 \leq x \leq 5$, disajikan pada Gambar 9.1.



Gambar 9.1. Grafik fungsi g, $g(x) = x + e^{-x} - 3.5, -3 \leq x \leq 5$.

Grafik fungsi g, $g(x) = x + e^{-x} - 3.5$ pada Gambar 9.1 tersebut menunjukkan bahwa persamaan (9.1) mempunyai dua akar real (satu akar negative dan satu akar positif) pada interval tutup $[-3, 5]$. Untuk menentukan akar negative dan akar positif persamaan (9.1) menggunakan metode optimisasi *swarm* partikel, masalah optimasi pada persamaan (9.2) berturut-turut dapat disederhanakan menjadi bentuk:

$$\min z = |x + e^{-x} - 3.5|, x \in [-3, 0]. \tag{9.3}$$

dan

$$\min z = |x + e^{-x} - 3.5|, x \in [0,5]. \quad (9.4)$$

Implementasi metode optimisasi swarm partikel menggunakan perangkat lunak Scilab untuk menentukan solusi masalah optimasi (9.3) disajikan pada subbab berikutnya.

9.1.1. Implementasi Program

Kode program yang digunakan untuk mengimplementasikan metode optimisasi swarm partikel menggunakan perangkat lunak Scilab untuk menentukan hampiran solusi optimal pada persamaan (9.3) terdiri dari:

(1) fungsi_ga.sce (berupa suatu fungsi/*function*)
(2) osp_p93.sce (berupa suatu *script*)

Diasumsikan bahwa kedua file kode program tersebut disimpan dalam folder D:\File_Scilab. Kode program 9.1 dan kode program 9.2 berurut-turut merepresentasikan fungsi **fungsi_ga.sce** dan *script* **osp_p93.sce.**

Kode program 9.1 (fungsi_ga.sce)

```
//Kode program 9.1 (fungsi_ga.sce)
function y = fungsi_ga(x)
   y = abs(x + exp(-x) – 3.5);
endfunction
//Akhir kode program
```

Kode program 9.1 berupa suatu fungsi yang bernama **fungsi_ga**. Untuk menghindari kesalahan eksekusi program, nama file Scilab editor disamakan dengan nama fungsi, sehingga fungsi tersebut disimpan dengan nama file **fungsi_ga.sce**. Definisi fungsi tersebut merupakan fungsi tujuan dari masalah optimasi pada persamaan (9.3), yaitu $z = |x + e^{-x} - 3.5|$.

Kode Program 9.2 (osp_p93.sce)

```
//Kode program 9.2. (osp_p93.sce)
clc;
clear;
xdel(winsid()); // Close all figure

tic( );

nVar = 1;          // Banyaknya variabel keputusan
VarMin = [-3];     // Nilai minimum variabel keputusan
VarMax = [0];      // Nilai maksimum variabel keputusan

// Membuka file function
exec('fungsi_ga.sce',-1);

// Random seed generator
n = sum(getdate( ));
rand("seed",n);
// Parameters OSP
```

```
MaxIt = 500;    // Maksimum banyaknya iterasi
nPop = 40;       // Ukuran populasi (ukuran swarm)
w = 1;           // Bobot inersia
c1 = 2.0;       // Koefisien personal
c2 = 2.0;       // Koefisien global

// Batas nilai kecepatan partikel
VelMax = 0.1*(VarMax – VarMin);
VelMin = – VelMax;

// Inisialisasi
posisi = rand(nPop, nVar);
kecepatan = zeros(nPop, nVar);
biaya  = zeros(nPop, 1);
pbest  = zeros(nPop, nVar);
biaya_pbest = zeros(nPop, 1);
gbest = zeros(nPop, 1);
biaya_gbest = zeros(1,1);

// Inisialisasi
for i = 1:nPop
   // Inisialisasi posisi partikel
   posisi(i, :) = VarMin + (VarMax - VarMin) .* rand(1,nVar);

   // Hitung nilai fungsi tujuan untuk masing-masing posisi
   biaya(i) = fungsi_ga(posisi(i,:));

   // Update partikel terbaik lokal
   pbest(i,:) = posisi(i,:);
   biaya_pbest(i) = biaya(i);

   // Update partikel terbaik global
   if (i == 1)
      gbest = posisi(i,:);
      biaya_gbest = biaya(i);
   else
      if (biaya(i) < biaya_gbest)
         gbest = posisi(i,:);
      7  biaya_gbest = biaya(i);
      end
   end
end

BestCost = zeros(MaxIt,1);
// Loop metode OSP
```

```
for it = 1:MaxIt
    disp('Iterasi = ' + string(it));
    for i=1:nPop

        // Update kecepatan partikel
        kecepatan(i,:) = w*kecepatan(i,:) + c1 * rand(1,nVar) .* (pbest(i,:) - ...
                         posisi(i,:)) + c2 * rand(1,nVar) .* (gbest - posisi(i,:));

        // Terapkan nilai batas kecepatan partikel
        kecepatan(i,:) = max(kecepatan(i,:), VelMin);
        kecepatan(i,:) = min(kecepatan(i,:), VelMax);

        // Update posisi partikel
        posisi(i,:) = posisi(i,:) + kecepatan(i,:);

        // Terapkan nilai batas posisi partikel
        posisi(i,:) = max(posisi(i,:), VarMin);
        posisi(i,:) = min(posisi(i,:), VarMax);

        // Hitung nilai fungsi tujuan untuk masing-masing posisi
        biaya(i) = fungsi_ga(posisi(i,:));

        // Update posisi partikel terbaik lokal
        if (biaya(i) < biaya_pbest(i))
            pbest(i,:) = posisi(i,:);
            biaya_pbest(i) = biaya(i);
        // Update posisi partikel terbaik global
            if (biaya(i) < biaya_gbest)
                gbest = posisi(i,:);
                biaya_gbest = biaya(i);
            end
        end // if (biaya(i) < biaya_pbest(i))
    end // for i = 1:nPop

    BestCost(it) = biaya_gbest;

    disp('Iterasi ' + string(it) + ', hampiran akar x = ' ...
          + string(gbest) + ', z = ' + string(BestCost(it)));

end

// Grafik nilai fungsi tujuan
scf(1);
xn = (1:1:MaxIt).';
plot2d('nl',xn,BestCost,style = [1]);
title('Nilai minimum fungsi tujuan',"fontsize",3.5);
```

```
xlabel('Iterasi',"fontsize",3.5);
ylabel('Nilai z',"fontsize",3.5);
gca( ).grid=[1 1 1]; // Setara dengan grid on;
gca( ).children(1).children(1).thickness = 3; // Set ketebalan garis kurva
axes_handle.grid = [1 1 1]; // Setara dengan grid(axes_handle,"on")

hasil_OSP = [gbest, biaya_gbest]
waktu = toc( );
disp('waktu = ' + string(waktu) + ' detik');
//Akhir kode program
```

Simpanlah kode program 9.2 berupa script tersebut dengan nama **osp_p93.sce**.

Berikut disajikan beberapa penjelasan yang berhubungan dengan Kode program 9.2 tersebut:

(a) Perintah **tic( )** digunakan untuk memulai waktu stopwatch. Perintah **waktu = toc( )** digunakan untuk mengakhiri waktu stopwatch.

(b) Blok perintah

```
nVar = 1;          // Banyaknya variabel keputusan
VarMin = [– 3];    // Nilai minimum variabel keputusan
VarMax = [0];      // Nilai maksimum variabel keputusan
```

Berhubungan dengan banyaknya variabel keputusan yang ada pada masalah optimasi pada persamaan (9.3). Pada persamaan (9.3) hanya ada satu variabel bebas (variabel keputusan), yaitu variabel x. Nilai variabel x terletak pada interval tutup [– 3, 0]. Akibatnya, pada blok perintah tersebut, didefinisikan **nVar = 1**, **VarMin = [– 3]**, dan **VarMax = [0]**.

(c) Perintah **exec('fungsi_ga.sce', -1)** digunakan untuk mengaktifkan file script/function **fungsi_ga.sce**.

(d) Perintah **gca( ).grid = [1 1 1]** digunakan untuk mengaktifkan garis grid pada suatu grafik dua dimensi.

(e) Perintah **gca( ).children(1).children(1).thickness = 3** digunakan untuk mengatur ketebalan garis kurva menjadi tiga kali lebih tebal dari nilai standar.

9.1.2. Menjalankan Program

Prosedur berikut dapat digunakan untuk menjalankan untuk menjalankan file kode program **osp_p93.sce** yang tersimpan dalam folder D:\File_Scilab:

(a) Bukalah perangkat lunak Scilab.

(b) Aktifkan folder D:\File_Scilab melalui klik menu *File – Current working directory – Browse for new*. Pada kotak dialog *Select directory*, pilih folder **D:\Program_Scilab**. Selanjutnya klik tombol *Open*.

(c) Selanjutnya klik menu *File – Open a file*. Pada kotak dialog Select a file to open, klik file **osp_p93.sce**. Selanjutnya klik tombol *Open*.

(d) Pada jendela **osp_p93.sce**, klik menu *Execute – save and execute*.

Berikut disajikan contoh tampilan hasil eksekusi kode program osp_p93.sce.

```
Iterasi = 499
 Iterasi 499, hampiran akar x = -1.6363411, z = 0.0000006
```

Iterasi = 500
Iterasi 500, hampiran akar x = -1.6363411, z = 0.0000006
waktu = 19.909065 detik

Setelah 500 iterasi, diperoleh hampiran solusi optimal pada persamaan (9.3) yaitu $x \approx -1.6363411$ dengan nilai fungsi tujuan $z \approx 0.0000006$. Waktu eksekusi kode program sekitar 19.9 detik. Hal ini menunjukkan bahwa hampiran akar real negative pada persamaan (9.1) adalah $x = -1.6363411$. Dengan metode Newton-Raphson, hampiran akar negative pada persamaan (9.1) adalah $y = -1.63634094817475$ (ketelitian 14 desimal). Dengan demikian, dengan memilih parameter hampiran akar yang diperoleh dari metode optimisasi *swarm* partikel mempunyai ketelitian 7 angka desimal.

Selanjutnya akan dipelajari variasi nilai koefisien inersia terhadap akurasi solusi terbaik yang diperoleh dari metode optimisasi swarm partikel. Nilai koefisien inersia w bervariasi dari nilai w = -0.5 sampai w = 1.2. Untuk setiap nilai koefisien inersia, metode optimisasi swarm partikel diimplementasikan sebanyak 25 kali menggunakan kode program pada Lampiran 1. Solusi terbaik yang diperoleh dan nilai fungsi tujuan yang bersesuaian untuk setiap koefisien inersia, disajikan pada Tabel 9.1

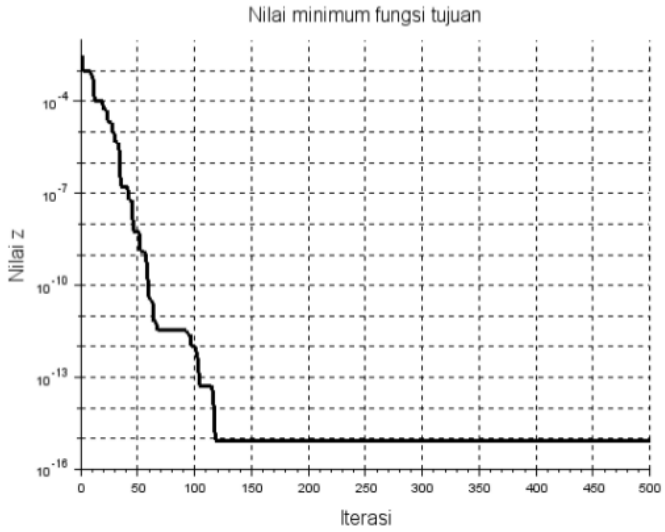
Tabel 9.1. Solusi terbaik yang diperoleh untuk masalah optimisasi pada persamaan (9.3)

w	x (solusi)	z (Nilai fungsi tujuan)	Simpangan baku z
-0.5	-1.63634329579934	$9.71058987264683 * 10^{-6}$	$4.61785 * 10^{-4}$
-0.3	-1.63634094818053	$2.39412933922267 * 10^{-11}$	$5.3333 * 10^{-6}$
-0.2	-1.63634094817475	$8.88178419700125 * 10^{-16}$	$1.78425 * 10^{-11}$
-0.15	-1.63634094817475	$8.88178419700125 * 10^{-16}$	kurang dari 10^{-30}
-0.1	-1.63634094817475	$8.88178419700125 * 10^{-16}$	kurang dari 10^{-30}
0	-1.63634094817475	$8.88178419700125 * 10^{-16}$	kurang dari 10^{-30}
0.1	-1.63634094817475	$8.88178419700125 * 10^{-16}$	kurang dari 10^{-30}
0.3	-1.63634094817475	$8.88178419700125 * 10^{-16}$	kurang dari 10^{-30}
0.5	-1.63634094817475	$8.88178419700125 * 10^{-16}$	kurang dari 10^{-30}
0.7	-1.63634094817475	$8.88178419700125 * 10^{-16}$	kurang dari 10^{-30}
0.75	-1.63634094817475	$8.88178419700125 * 10^{-16}$	kurang dari 10^{-30}
0.8	-1.63634094817475	$8.88178419700125 * 10^{-16}$	$1.24942 * 10^{-11}$
0.9	-1.63634089727230	$2.10549869983367 * 10^{-7}$	$1.20458 * 10^{-5}$
1.0	-1.63634099200756	$1.81307473035019 * 10^{-7}$	$1.86956 * 10^{-5}$
1.2	-1.63634158949205	$2.65270808830209 * 10^{-6}$	$5.50326 * 10^{-5}$

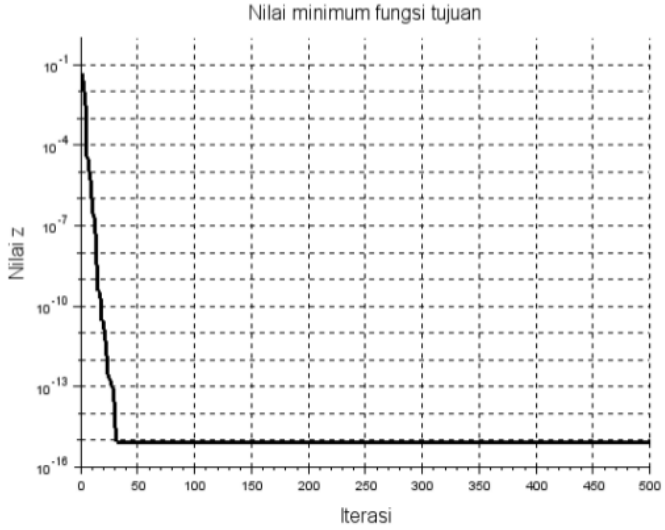
Tabel 9.1 tersebut mengindikasikan bahwa nilai koefisien inersia (w) berpengaruh terhadap performansi metode optimisasi *swarm* partikel. Solusi yang diperoleh dari metode optimisasi swarm partikel sangat mendekati solusi analitik ketika koefisien inersia (w) berada pada rentang tertentu (rentang optimal). Pada permasalahan penentuan akar persamaan nonlinear, metode optimisasi swarm parameter menghasilkan hampiran nilai akar dengan akurasi tinggi dan simpangan baku hampiran akar bernilai sangat kecil ketika nilai koefisien inersia berada dalam rentang $-0.15 \leq w \leq 0.75$.

Grafik perubahan nilai fungsi tujuan (z) pada permasalahan optimasi pada persamaan (9.3) menggunakan metode optimisasi swarm partikel, disajikan pada Gambar 9.2. Berdasarkan Gambar 9.2 tersebut, nilai fungsi tujuan turun dari orde 10^{-2} pada awal iterasi menjadi orde 10^{-16} pada

iterasi ke-150 sampai dengan iterasi ke-500. Berdasarkan Gambar 9.3 tersebut, nilai fungsi tujuan turun dari orde 10^{-2} pada awal iterasi menjadi orde 10^{-16} pada iterasi ke-50 sampai dengan iterasi ke-500.



Gambar 9.2. Perubahan nilai fungsi tujuan untuk masalah optimasi pada persamaan (9.3) menggunakan koefisien inersia w = 0.5.



Gambar 9.3. Perubahan nilai fungsi tujuan untuk masalah optimasi pada persamaan (9.3) menggunakan koefisien inersia w = 0.

9.2. Penentuan akar sistem persamaan non linear

Pada bagian ini, akan disajikan implementasi metode optimisasi swarm partikel untuk menentukan hampiran akar suatu system persamaan non linear. Misalkan diberikan masalah penentuan akar real sistem persamaan non linear yang berbentuk

$$x_1 x_2^2 - 3x_2^2 + 2x_1 = 3;$$
$$x_1^2 x_2 + 2x_2 - x_1^3 = 4;\ x_1, x_2 \in [-5, 5]. \quad (9.5)$$

Masalah penentuan akar real pada sistem persamaan (9.5) tersebut dapat dapat ditransformasi menjadi masalah optimasi yang berbentuk

$\min z = |x_1 x_2^2 - 3x_2^2 + 2x_1 - 3| + |x_1^2 x_2 + 2x_2 - x_1^3 - 4|,\ x_1, x_2 \in [-5, 5].$ (9.6)

Implementasi metode optimisasi swarm partikel menggunakan perangkat lunak Scilab untuk menentukan solusi masalah optimasi pada persamaan (9.6) disajikan pada subbab berikutnya.

9.2.1. Implementasi Program

Kode program yang digunakan untuk mengimplementasikan metode optimisasi swarm partikel menggunakan perangkat lunak Scilab untuk menentukan hampiran solusi optimal pada persamaan (9.6) terdiri dari:

(1) fungsi_gb.sce (berupa suatu fungsi/*function*)
(2) osp_p96.sce (berupa suatu *script*)

Diasumsikan bahwa kedua file kode program tersebut disimpan dalam folder D:\File_Scilab. Kode program 9.3 dan kode program 9.4 berurut-turut merepresentasikan fungsi **fungsi_gb.sce** dan *script* **osp_p96.sce.** Pada kode program 9.4, digunakan nilai parameter $w = 0, c_1 = 2, c_2 = 2$.

Kode program 9.3 (fungsi_gb.sce)

```
//Kode program 9.3 (fungsi_gb.sce)
function y = fungsi_gb(x)
   y1 = abs(x(:,1) .* x(:,2).^2 - 3*x(:,2)^2 + 2*x(:,1) - 3);
   y2 = abs(x(:,1).^2 .* x(:,2) + 2*x(:,2) - x(:,1).^3 - 4);
   y  = y1 + y2;
endfunction
//Akhir kode program
```

Kode program 9.3 berupa suatu fungsi yang bernama **fungsi_gb**. Untuk menghindari kesalahan eksekusi program, nama file Scilab editor disamakan dengan nama fungsi, sehingga fungsi tersebut disimpan dengan nama file **fungsi_gb.sce**. Definisi fungsi tersebut merupakan fungsi tujuan dari masalah optimasi pada persamaan (9.6), yaitu $z = |x_1 x_2^2 - 3x_2^2 + 2x_1 - 3| + |x_1^2 x_2 + 2x_2 - x_1^3 - 4|$.

Kode Program 9.4 (osp_p96.sce)

```
//Kode program 9.4. (osp_p96.sce)
clc;
clear;
xdel(winsid()); // Close all figure

tic( );
```

```
nVar = 2;                   // Banyaknya variabel keputusan
VarMin = [0, 0];            // Nilai minimum variabel keputusan
VarMax = [4, 4];            // Nilai maksimum variabel keputusan

// Membuka file function
exec('fungsi_gb.sce',-1);

// Random seed generator
n = sum(getdate( ));
rand("seed",n);

// Parameters OSP
MaxIt = 500;    // Maksimum banyaknya iterasi
nPop = 40;       // Ukuran populasi (ukuran swarm)
w = 0;           // Bobot inersia
c1 = 2.0;       // Koefisien personal
c2 = 2.0;       // Koefisien global

// Batas nilai kecepatan partikel
VelMax = 0.1*(VarMax – VarMin);
VelMin = – VelMax;

// Inisialisasi
posisi = rand(nPop, nVar);
kecepatan = zeros(nPop, nVar);
biaya  = zeros(nPop, 1);
pbest  = zeros(nPop, nVar);
biaya_pbest = zeros(nPop, 1);
gbest = zeros(nPop, 1);
biaya_gbest = zeros(1,1);

// Inisialisasi
for i = 1:nPop
   // Inisialisasi posisi partikel
   posisi(i, :) = VarMin + (VarMax - VarMin) .* rand(1,nVar);

   // Hitung nilai fungsi tujuan untuk masing-masing posisi
   biaya(i) = fungsi_gb(posisi(i,:));

   // Update partikel terbaik lokal
   pbest(i,:) = posisi(i,:);
   biaya_pbest(i) = biaya(i);

   // Update partikel terbaik global
```

```
    if (i == 1)
        gbest = posisi(i,:);
        biaya_gbest = biaya(i);
    else
        if (biaya(i) < biaya_gbest)
            gbest = posisi(i,:);
            biaya_gbest = biaya(i);
        end
    end
end

BestCost = zeros(MaxIt,1);
// Loop metode OSP
for it = 1:MaxIt
    disp('Iterasi = ' + string(it));
    for i=1:nPop

        // Update kecepatan partikel
        kecepatan(i,:) = w*kecepatan(i,:) + c1 * rand(1,nVar) .* (pbest(i,:) - ...
                         posisi(i,:)) + c2 * rand(1,nVar) .* (gbest - posisi(i,:));

        // Terapkan nilai batas kecepatan partikel
        kecepatan(i,:) = max(kecepatan(i,:), VelMin);
        kecepatan(i,:) = min(kecepatan(i,:), VelMax);

        // Update posisi partikel
        posisi(i,:) = posisi(i,:) + kecepatan(i,:);

        // Terapkan nilai batas posisi partikel
        posisi(i,:) = max(posisi(i,:), VarMin);
        posisi(i,:) = min(posisi(i,:), VarMax);

        // Hitung nilai fungsi tujuan untuk masing-masing posisi
        biaya(i) = fungsi_gb(posisi(i,:));

        // Update posisi partikel terbaik lokal
        if (biaya(i) < biaya_pbest(i))
            pbest(i,:) = posisi(i,:);
            biaya_pbest(i) = biaya(i);
        // Update posisi partikel terbaik global
            if (biaya(i) < biaya_gbest)
                gbest = posisi(i,:);
                biaya_gbest = biaya(i);
            end
        end // if (biaya(i) < biaya_pbest(i))
    end // for i = 1:nPop
```

```
    BestCost(it) = biaya_gbest;

   disp('Iterasi ' + string(it) + ', x1 = ' + string(gbest(1)) + ...
    ', x2 = ' + string(gbest(2)) +  ', z = ' + string(BestCost(it)));

end

// Grafik nilai fungsi tujuan
scf(1);
xn = (1:1:MaxIt).';
plot2d('nl',xn, BestCost, style = [1]);
title('Nilai minimum fungsi tujuan',"fontsize",3.5);
xlabel('Iterasi',"fontsize",3.5);
ylabel('Nilai z',"fontsize",3.5);
gca( ).grid=[1 1 1]; // Setara dengan grid on;
gca( ).children(1).children(1).thickness = 3; // Set ketebalan garis kurva
axes_handle.grid = [1 1 1]; // Setara dengan grid(axes_handle,"on")

hasil_OSP = [gbest, biaya_gbest]
waktu = toc( );
disp('waktu = ' + string(waktu) + ' detik');
//Akhir kode program
```

Simpanlah kode program 9.4 berupa script tersebut dengan nama **osp_p96.sce**.

Berikut disajikan beberapa penjelasan yang berhubungan dengan Kode program 9.4 tersebut:

(a) Blok perintah

```
nVar = 2;            // Banyaknya variabel keputusan
VarMin = [0, 0];     // Nilai minimum variabel keputusan
VarMax = [4, 4];     // Nilai maksimum variabel keputusan
```

berhubungan dengan banyaknya variabel keputusan yang ada pada masalah optimasi pada persamaan (9.6). Pada persamaan (9.6) terdapat dua variabel bebas (variabel keputusan), yaitu variabel x_1 dan x_2. Nilai variabel x_1 dan x_2 terletak pada interval tutup $[0, 4]$. Akibatnya, pada blok perintah tersebut, didefinisikan **nVar = 2**, **VarMin = [0, 0]**, dan **VarMax = [4, 4]**.

(b) Blok perintah

```
// Update posisi partikel terbaik lokal
if (biaya(i) < biaya_pbest(i))
        pbest(i,:) = posisi(i,:);
        biaya_pbest(i) = biaya(i);
        // Update posisi partikel terbaik global
         if (biaya(i) < biaya_gbest)
                gbest = posisi(i,:);
                biaya_gbest = biaya(i);
```

```
        end
    end // if (biaya(i) < biaya_pbest(i))
```

digunakan untuk melakukan update posisi partikel terbaik lokal dan posisi partikel terbaik global. Masalah optimasi pada persamaan (9.6) merupakan masalah penentuan **nilai minimum**, sehingga posisi partikel terbaik lokal dan posisi terbaik global diupdate jika **(biaya(i) < biaya_pbest(i))** dan **(biaya(i) < biaya_gbest)**.

9.2.2. Menjalankan Program

Prosedur berikut dapat digunakan untuk menjalankan untuk menjalankan file kode program **osp_p96.sce** yang tersimpan dalam folder D:\File_Scilab:

(e) Bukalah perangkat lunak Scilab.
(f) Aktifkan folder D:\File_Scilab melalui klik menu *File – Current working directory – Browse for new*. Pada kotak dialog *Select directory*, pilih folder **D:\Program_Scilab**. Selanjutnya klik tombol *Open*.
(g) Selanjutnya klik menu *File – Open a file*. Pada kotak dialog Select a file to open, klik file **osp_p96.sce**. Selanjutnya klik tombol *Open*.
(h) Pada jendela **osp_p96.sce**, klik menu *Execute – save and execute*.

Berikut disajikan contoh tampilan hasil eksekusi kode program osp_p96.sce.

```
Iterasi = 499
 Iterasi 499, x1 = 2.634771431272, x2 = 2.492796486879, z = 1.77635684D-15
 Iterasi = 500
 Iterasi 500, x1 = 2.634771431272, x2 = 2.492796486879, z = 1.77635684D-15
   20.5375421
```

Setelah 500 iterasi, diperoleh hampiran solusi optimal pada persamaan (9.6) yaitu $x_1 \approx 2.634771431272, x_2 \approx 2.492796486879$ dengan nilai fungsi tujuan $z \approx 1.78 * 10^{-15}$. Waktu eksekusi kode program sekitar 20.5 detik. Hal ini menunjukkan bahwa hampiran akar pada sistem persamaan (9.5) adalah $(x_1, x_2) = (2.634771431272, 2.492796486879)$. Dengan menggunakan metode iterasi titik tetap yang berbentuk

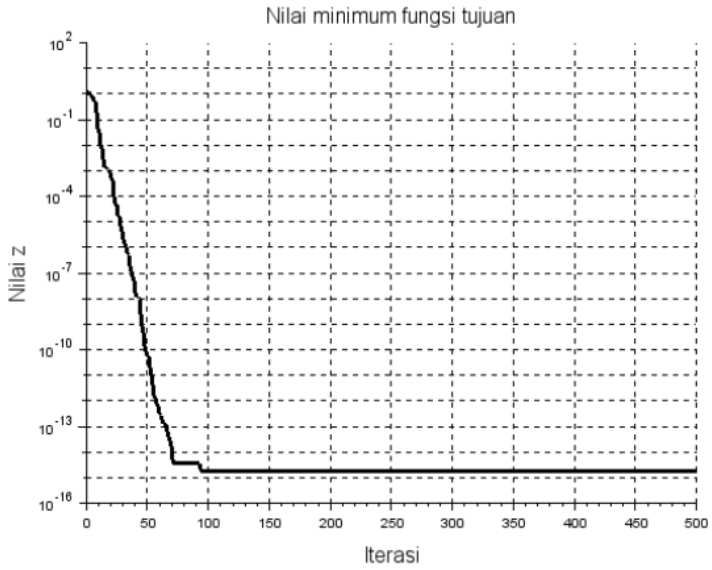
$$x_{1,k+1} = \frac{3\left(1 + x_{2,k}^2\right)}{\left(x_{2,k}^2 + 2\right)};\ x_{2,k+1} = \frac{4 + x_{1,k+1}^3}{\left(x_{1,k+1}^2 + 2\right)},$$

hampiran akar pada sistem persamaan (9.5) tingkat ketelitian 14 angka decimal adalah

$$(x_1, x_2) = (2.634771431272, 2.492796486879).$$

Dengan demikian, hampiran akar yang diperoleh dari metode optimisasi *swarm* partikel mempunyai ketelitian hingga 14 angka desimal.

Grafik perubahan nilai fungsi tujuan (z) pada permasalahan optimasi pada persamaan (9.6) menggunakan metode optimisasi swarm partikel, disajikan pada Gambar 9.3. Berdasarkan Gambar 9.3 tersebut, nilai fungsi tujuan turun dari orde 10^{-1} pada awal iterasi menjadi orde 10^{-15} pada iterasi ke-100 sampai iterasi ke-500.



Gambar 9.4. Perubahan nilai fungsi tujuan untuk masalah optimasi pada persamaan (9.6) dengan nilai w = 0.

9.3. Nilai Maksimum suatu Fungsi Dua Variabel

Pada bagian ini, akan disajikan implementasi metode optimisasi swarm partikel untuk menentukan nilai maksimum suatu fungsi dua variabel pada daerah tertentu. Misalkan diberikan masalah penentuan nilai maksimum fungsi dua variabel yang berbentuk

$$\text{maks } z = x_1(1 - \exp(-x_2)) + x_2 \exp(-2x_1) + \sin(x_1 + x_2)\,,\ x_1, x_2 \in [0, 5]. \quad (9.7)$$

Implementasi metode optimisasi swarm partikel menggunakan perangkat lunak Scilab untuk menentukan solusi masalah optimasi pada persamaan (9.6) disajikan pada subbab berikutnya.

9.3.1. Implementasi Program

Kode program yang digunakan untuk mengimplementasikan metode optimisasi swarm partikel menggunakan perangkat lunak Scilab untuk menentukan hampiran solusi optimal pada persamaan (9.7) terdiri dari:

(1) fungsi_gc.sce (berupa suatu fungsi/*function*)
(2) osp_p97.sce (berupa suatu *script*)

Diasumsikan bahwa kedua file kode program tersebut disimpan dalam folder D:\File_Scilab. Kode program 9.5 dan kode program 9.6 berurut-turut merepresentasikan fungsi **fungsi_gc.sce** dan *script* **osp_p97.sce.** Pada kode program 9.6, digunakan nilai parameter $w = 0, c_1 = 2, c_2 = 2$.

Kode program 9.5 (fungsi_gc.sce)

```
//Kode program 9.5 (fungsi_gc.sce)
function y = fungsi_gc(x)
    y = x(:,1).* (1 - exp(- x(:,2))) + x(:,2) .* exp(- 2*x(:,1)) + sin(x(:,1) + x(:,2));
endfunction
//Akhir kode program
```

Kode program 9.5 berupa suatu fungsi yang bernama **fungsi_gc**. Untuk menghindari kesalahan eksekusi program, nama file Scilab editor disamakan dengan nama fungsi, sehingga fungsi tersebut disimpan dengan nama file **fungsi_gc.sce**. Definisi fungsi tersebut merupakan fungsi tujuan dari masalah optimasi pada persamaan (9.7), yaitu $z = x_1(1 - \exp(-x_2)) + x_2 \exp(-2x_1) + \sin(x_1 + x_2)$, $x_1, x_2 \in [0, 5]$.

Kode Program 9.6 (osp_p97.sce)

```
//Kode program 9.6. (osp_p97.sce)
clc;
clear;
xdel(winsid()); // Close all figure

tic( );

nVar = 2;                  // Banyaknya variabel keputusan
VarMin = [0, 0];           // Nilai minimum variabel keputusan
VarMax = [5, 5];           // Nilai maksimum variabel keputusan

// Membuka file function
exec('fungsi_gc.sce',-1);

// Random seed generator
n = sum(getdate( ));
rand("seed",n);

// Parameters OSP
MaxIt = 200;    // Maksimum banyaknya iterasi
nPop = 40;       // Ukuran populasi (ukuran swarm)
w = 0;           // Bobot inersia
c1 = 2.0;       // Koefisien personal
c2 = 2.0;       // Koefisien global

// Batas nilai kecepatan partikel
VelMax = 0.1*(VarMax – VarMin);
VelMin = – VelMax;

// Inisialisasi
posisi = rand(nPop, nVar);
```

```
kecepatan = zeros(nPop, nVar);
fobyektif  = zeros(nPop, 1);
pbest  = zeros(nPop, nVar);
fobyektif_pbest = zeros(nPop, 1);
gbest = zeros(nPop, 1);
fobyektif_gbest = zeros(1,1);

// Inisialisasi
for i = 1:nPop
   // Inisialisasi posisi partikel
   posisi(i, :) = VarMin + (VarMax - VarMin) .* rand(1,nVar);

   // Hitung nilai fungsi tujuan untuk masing-masing posisi
   fobyektif(i) = fungsi_gc(posisi(i,:));

   // Update partikel terbaik lokal
   pbest(i,:) = posisi(i,:);
   fobyektif_pbest(i) = biaya(i);

   // Update partikel terbaik global
   if (i == 1)
      gbest = posisi(i,:);
      fobyektif_gbest = fobyektif(i);
   else
      if (fobyektif(i) > fobyektif_gbest)
         gbest = posisi(i,:);
         fobyektif_gbest = fobyektif(i);
      end
   end
end

BestCost = zeros(MaxIt,1);
// Loop metode OSP
for it = 1:MaxIt
   disp('Iterasi = ' + string(it));
   for i=1:nPop

      // Update kecepatan partikel
      kecepatan(i,:) = w*kecepatan(i,:) + c1 * rand(1,nVar) .* (pbest(i,:) - ...
                       posisi(i,:)) + c2 * rand(1,nVar) .* (gbest - posisi(i,:));

      // Terapkan nilai batas kecepatan partikel
      kecepatan(i,:) = max(kecepatan(i,:), VelMin);
      kecepatan(i,:) = min(kecepatan(i,:), VelMax);

      // Update posisi partikel
```

```
        posisi(i,:) = posisi(i,:) + kecepatan(i,:);

        // Terapkan nilai batas posisi partikel
        posisi(i,:) = max(posisi(i,:), VarMin);
        posisi(i,:) = min(posisi(i,:), VarMax);

        // Hitung nilai fungsi tujuan untuk masing-masing posisi
        fobyektif(i) = fungsi_gc(posisi(i,:));

        // Update posisi partikel terbaik lokal
        if (fobyektif(i) > fobyektif_pbest(i))
           pbest(i,:) = posisi(i,:);
           fobyektif_pbest(i) = fobyektif(i);
        // Update posisi partikel terbaik global
           if (fobyektif(i) > fobyektif_gbest)
                 gbest = posisi(i,:);
                 fobyektif_gbest = fobyektif(i);
           end // if (fobyektif(i) > fobyektif_gbest)
        end // if (fobyektif(i) > fobyektif_pbest(i))
    end // for i = 1:nPop

    BestCost(it) = fobyektif_gbest;

   disp('Iterasi ' + string(it) + ', x1 = ' + string(gbest(1)) + ...
    ', x2 = ' + string(gbest(2)) +  ', z = ' + string(BestCost(it)));

end

// Grafik nilai fungsi tujuan
scf(1);
xn = (1:1:MaxIt).';
plot2d('nn',xn,BestCost,style = [1]);
title('Nilai minimum fungsi tujuan',"fontsize",3.5);
xlabel('Iterasi',"fontsize",3.5);
ylabel('Nilai z',"fontsize",3.5);
gca( ).grid=[1 1 1]; // Setara dengan grid on;
gca( ).children(1).children(1).thickness = 3; // Set ketebalan garis kurva
axes_handle.grid = [1 1 1]; // Setara dengan grid(axes_handle,"on")

hasil_OSP = [gbest, fobyektif_gbest]
waktu = toc( );
disp('waktu = ' + string(waktu) + ' detik');

//Akhir kode program
```

Simpanlah kode program 9.6 berupa script tersebut dengan nama **osp_p97.sce**.

Berikut disajikan beberapa penjelasan yang berhubungan dengan Kode program 9.6 tersebut:

(a) Blok perintah

```
nVar = 2;            // Banyaknya variabel keputusan
VarMin = [0, 0];     // Nilai minimum variabel keputusan
VarMax = [5, 5];     // Nilai maksimum variabel keputusan
```

berhubungan dengan banyaknya variabel keputusan yang ada pada masalah optimasi pada persamaan (9.7). Pada persamaan (9.7) terdapat dua variabel bebas (variabel keputusan), yaitu variabel x_1 dan x_2. Nilai variabel x_1 dan x_2 terletak pada interval tutup $[0, 5]$. Akibatnya, pada blok perintah tersebut, didefinisikan **nVar = 2**, **VarMin = [0, 0]**, dan **VarMax = [5, 5]**.

(b) Blok perintah

```
// Update posisi partikel terbaik lokal
if (fobyektif(i) > fobyektif_pbest(i))
        pbest(i,:) = posisi(i,:);
        fobyektif_pbest(i) = fobyektif(i);
        // Update posisi partikel terbaik global
        if (fobyektif(i) > fobyektif_gbest)
                gbest = posisi(i,:);
                fobyektif_gbest = fobyektif(i);
        end // if (fobyektif(i) > fobyektif_gbest)
end // if (fobyektif(i) > fobyektif_pbest(i))
```

digunakan untuk melakukan update posisi partikel terbaik lokal dan posisi partikel terbaik global. Masalah optimasi pada persamaan (9.7) merupakan masalah penentuan **nilai maksimum**, sehingga posisi partikel terbaik lokal dan posisi terbaik global diupdate jika **(fobyektif(i) > fobyektif_pbest(i))** dan **(fobyektif(i) > fobyektif_gbest)**.

9.3.2. Menjalankan Program

Prosedur berikut dapat digunakan untuk menjalankan untuk menjalankan file kode program **osp_p97.sce** yang tersimpan dalam folder D:\File_Scilab:

(i) Bukalah perangkat lunak Scilab.

(j) Aktifkan folder D:\File_Scilab melalui klik menu *File – Current working directory – Browse for new*. Pada kotak dialog *Select directory*, pilih folder **D:\Program_Scilab**. Selanjutnya klik tombol *Open*.

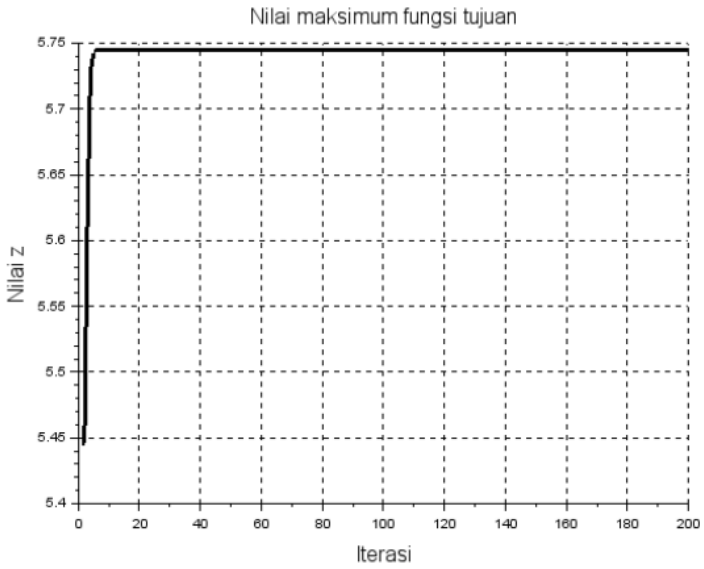
(k) Selanjutnya klik menu *File – Open a file*. Pada kotak dialog Select a file to open, klik file **osp_p97.sce**. Selanjutnya klik tombol *Open*.

(l) Pada jendela **osp_p97.sce**, klik menu *Execute – save and execute*.

Berikut disajikan contoh tampilan hasil eksekusi kode program osp_p97.sce.

```
 Iterasi = 199
 Iterasi 199, x1 = 5, x2 = 3.084787, z = 5.7449236
 Iterasi = 200
 Iterasi 200, x1 = 5, x2 = 3.084787, z = 5.7449236
   6.6140314
```

Setelah 200 iterasi, diperoleh hampiran solusi optimal pada persamaan (9.7) yaitu $x_1 \approx 5, x_2 \approx 3.084787$ dengan nilai fungsi tujuan $z \approx 5.7449236$. Waktu eksekusi kode program sekitar 6.6 detik. Hal ini menunjukkan bahwa nilai maksimum fungsi dua variabel pada persamaan (9.7) tercapai ketika $(x_1, x_2) \approx (5, 3.084787)$. Grafik perubahan nilai fungsi tujuan (z) pada permasalahan optimasi pada persamaan (9.7) menggunakan metode optimisasi swarm partikel, disajikan pada Gambar 9.5.



Gambar 9.5. Perubahan nilai fungsi tujuan untuk masalah optimasi pada persamaan (9.7)

10. Estimasi Parameter Model Pertumbuhan Populasi

10.1. Pendahuluan

Model matematika pertumbuhan populasi telah diterapkan secara luas untuk mendeskripsikan relasi antara berat badan dan umur dalam bidang ilmu kedokteran hewan. Dari suatu model matematika pertumbuhan populasi, dapat ditentukan parameter aplikatif penting yang berhubungan dengan pertumbuhan hewan, misalkan berat maksimal hewan (*mature weight*) dan laju pertumbuhan (*growth rate*) suatu hewan. Kedua parameter tersebut sangat bermanfaat untuk memperoleh prakiraan/estimasi kebutuhan makanan harian suatu hewan ternak, atau untuk mengevaluasi pengaruh kondisi lingkungan terhadap pertumbuhan suatu hewan ternak. Selain itu, model matematika pertumbuhan populasi juga dapat diterapkan untuk memperkirakan umur pemotongan ternak yang optimal. Oleh karena itu, model matematika pertumbuhan populasi dapat dipandang sebagai instrumen optimisasi dalam produksi hewan [22, 23, 24].

Proses pertumbuhan ternak, termasuk unggas, dapat diukur dari profil massa (berat) ternak terhadap waktu [25, 26]. Pertumbuhan ternak dan unggas umumnya mengikuti pola *sigmoid*. Pertumbuhan unggas biasanya dimulai dengan fase pertumbuhan yang cepat dari fase penetasan. Kemudian, unggas mencapai tingkat pertumbuhan maksimum pada waktu tertentu (*waktu belok*). Setelah itu, pertumbuhan unggas melambat. Pada fase akhir, berat unggas umumnya cenderung ke nilai batas (asimtot), yaitu berat badan unggas pada usia dewasa (*mature weight*) [27, 28].

Model pertumbuhan dapat diklasifikan ke dalam dua kelompok, yaitu model pertumbuhan empiris dan model pertumbuhan dinamik (model pertumbuhan yang diturunkan dari suatu persamaan diferensial biasa). Model pertumbuhan empiris meliputi model pertumbuhan Weibull dan model pertumbuhan MMF (Morgan-Mercer-Flodin). Model pertumbuhan Weibull dan MMF telah diterapkan untuk mendeskripsikan dinamika pertumbuhan ayam [29]. Model pertumbuhan dinamik meliputi model pertumbuhan logistik, Gompertz, Richards, dan model pertumbuhan WEP-logistik (WEP dari kata Windarto-Eridani-Purwati). Model pertumbuhan dinamik telah diterapkan untuk mendeskripsikan dinamika pertumbuhan berbagai binatang, termasuk ayam [25, 30], mammalia [31], ikan [32], reptilia [33], dan amphibi [34].

Misalkan $y(t)$ menyatakan berat badan hewan atau banyaknya populasi makhluk hidup pada saat t. Model pertumbuhan Weibull dan model pertumbuhan MMF beturut-turut diberikan oleh

$$y_w(t) = K - (K - A)\exp(-rt^D), \tag{10.1}$$

dan

$$y_{MMF}(t) = \frac{AB + Ct^D}{B + t^D}. \tag{10.2}$$

Pada persamaan (9.1), r dan K berturut-turut menyatakan laju pertumbuhan berat hewan dan berat maksimal hewan. Ketika $y(t)$ menyatakan banyaknya populasi suatu makhluk hidup, parameter r merupakan laju pertumbuhan intrinsik, sedangkan parameter K adalah parameter *carrying capacity*, yaitu banyaknya maksimal populasi yang dapat didukung secara optimal oleh lingkungan tempat hidup populasi makhluk hidup tersebut. Pada persamaan (10.1)-(10.2), A, B, C, D merupakan parameter empiris.

Model pertumbuhan logistik diturunkan dari persamaan diferensial logistik yang berbentuk

$$\frac{dy}{dt} = ry\left(1 - \frac{y}{K}\right), y(0) = y_0. \tag{10.3}$$

Model pertumbuhan logistik diberikan oleh

$$y_L(t) = \frac{K}{1+\exp\left(-r\left(t-t_{inf}\right)\right)}, \quad (10.4)$$

dengan $t_{inf} = \frac{1}{r}\ln\left(\frac{K}{y_0} - 1\right)$. Pada persamaan (10.4), parameter t_{inf} merupakan parameter waktu belok (*inflection time parameter*), yaitu waktu optimal pertumbuhan suatu populasi. Pada tahun 1959, Richards mengusulkan modifikasi persamaan diferensial logistik yang disebut persamaan diferensial Richard. Persamaan diferensial Richards berbentuk

$$\frac{dy}{dt} = ry\left(1 - \left(\frac{y}{K}\right)^{\beta}\right), y(0) = y_0. \quad (10.5)$$

Persamaan diferensial Gompertz merupakan suatu bentuk hampiran dari persamaan diferensial Richards. Persamaan diferensial Gompertz berbentuk

$$\frac{dy}{dt} = \lim_{\beta\to 0}\frac{ry\left(1-\left(\frac{y}{K}\right)^{\beta}\right)}{\beta} = ry\ln\left(\frac{K}{y}\right), y(0) = y_0. \quad (10.6)$$

Model pertumbuhan Richards dan Gompertz berturut-turut merupakan solusi persamaan diferensial Richards dan persamaan diferensial Gompertz. Model pertumbuhan Richards diberikan oleh

$$y_R(t) = \frac{K}{\left[1+\beta\exp\left(-r^*\left(t-t_{inf}\right)\right)\right]^{\frac{1}{\beta}}}, \quad (10.7)$$

dengan $r^* = \beta r, t_{inf} = \frac{1}{r^*}\ln\left(\frac{\left(\frac{K}{y_0}\right)^{\beta}-1}{\beta}\right)$.

Model pertumbuhan Gompertz diberikan oleh

$$y_G(t) = \frac{K}{\exp\left(\exp\left(-r\left(t-t_{inf}\right)\right)\right)}, \quad (10.8)$$

dengan $t_{inf} = \frac{1}{r}\ln\left(\ln\left(\frac{K}{y_0}\right)\right)$. Pada persamaan (10.7), parameter β merupakan parameter tambahan yang membedakan model pertumbuhan Richards dengan model pertumbuhan logistik. Ketika parameter $\beta = 1$, maka model pertumbuhan Richards dapat disederhanakan menjadi model pertumbuhan logistik. Ketika $\beta \to 0$,

$$\lim_{\beta\to 0} y_R(t) = \lim_{\beta\to 0}\frac{K}{\left[1+\beta\exp\left(-r^*\left(t-t_{inf}\right)\right)\right]^{\frac{1}{\beta}}} = \frac{K}{\exp\left(\exp\left(-r^*\left(t-t_{inf}\right)\right)\right)}.$$

Selain itu,

$$\lim_{\beta\to 0}\frac{1}{r^*}\ln\left(\frac{\left(\frac{K}{y_0}\right)^{\beta}-1}{\beta}\right) = \frac{1}{r^*}\ln\left(\ln\left(\frac{K}{y_0}\right)\right).$$

Dengan demikian, model pertumbuhan Gompertz pada persamaan (10.8) juga merupakan hampiran model pertumbuhan Richards pada persamaan (10.7) untuk kondisi parameter $\beta \to 0$.

Pada tahun 2018, Windarto-Eridani-Purwati juga mengusulkan modifikasi persamaan diferensial logistik yang berbentuk

$$\frac{dy}{dt} = (q+ry)\left(1-\frac{y}{K}\right), y(0) = y_0. \quad (10.9)$$

Pada persamaan (10.9), q dan r dapat dipandang sebagai laju pertumbuhan konstan dan laju pertumbuhan proporsional. Solusi persamaan diferensial (10.9) merupakan model pertumbuhan WEP-logistik, dan diberikan oleh

$$y_{WEP}(t) = \frac{K-(K-L)\exp(-\gamma t)}{1+\exp\left(-\gamma\left(t-t_{inf}\right)\right)}, 0 \le L \le K, \quad (10.10)$$

dengan $\gamma = \frac{q}{K} + r, L = K - q\left(\frac{K-y_0}{ry_0+q}\right),\ t_{inf} = \frac{K}{q+rK}\ln\left(r\left(\frac{K-y_0}{ry_0+q}\right)\right)$. Parameter γ pada persamaan (10.10) adalah laju pertumbuhan populasi, sedangkan parameter L berhubungan dengan ukuran awal (banyaknya populasi pada saat awal). Ketika parameter $L = K$, maka model pertumbuhan WEP-logistik dapat disederhanakan menjadi model pertumbuhan logistik.

Parameter pada model pertumbuhan populasi perlu diestimasi, sehingga model pertumbuhan populasi tersebut dapat digunakan untuk menentukan beberapa parameter penting atau indikator penting yang terkait dengan perkembangan populasi. Parameter penting atau indikator penting tersebut meliputi:

(a) Berat hewan ternak pada usia dewasa (*mature weight*). Parameter ini direpresentasikan oleh parameter K pada model pertumbuhan Weibul, logistik, Richards, Gompertz dan model WEP-logistik. Dalam hal variabel keadaan (*state variable*) $y(t)$ menyatakan banyaknya populasi suatu makhluk hidup, parameter K merupakan maksimal banyaknya populasi yang dapat didukung secara optimal oleh lingkungan.
(b) Laju pertumbuhan populasi, yaitu parameter r pada model pertumbuhan Weibul, logistik, Gompertz, Richards, atau parameter γ pada model pertumbuhan WEP-logistik.
(c) Waktu optimal/waktu pertumbuhan populasi paling cepat (t_{inf}).
(d) Kurva laju pertumbuhan populasi terhadap waktu (t). Secara matematis, kurva ini diperoleh dari plot $\frac{dy}{dt}$ terhadap waktu (t).

Dalam bagian selanjutnya akan dibahas implementasi metode optimisasi swarm partikel untuk mengestimasi nilai parameter model pertumbuhan logistik untuk mendeskripsikan pertumbuhan ayam jantan.

10.2. Estimasi Parameter Model Pertumbuhan Logistik

Pada bagian ini, metode optimisasi swarm partikel digunakan untuk mementukan parameter model pertumbuhan logistik pada kasus pertumbuhan ayam jantan. Model pertumbuhan yang digunakan adalah model pertumbuhan logistik pada persamaan (10.4) yang berbentuk

$$y(t) = \frac{K}{1+\exp\left(-r\left(t-t_{inf}\right)\right)}.$$

Pada model tersebut, $y(t)$ adalah rata-rata berat ayam jantan (dalam satuan gram) pada hari ke- t. Parameter r, K, t_{inf} berturut-turut menyatakan laju pertumbuhan berat ayam jantan, berat maksimal ayam jantan (*mature weight*), dan waktu pertumbuhan optimal ayam jantan. Data berat ayam jantan dirujuk dari literature dan disajikan pada Tabel 10.1 [4, 25, 30]. Parameter r, K, t_{inf} diestimasi sedemikian hingga rata-rata kuadrat galat (*mean square error*) MSE

$$MSE = \frac{1}{ndata}\sum_{i=1}^{ndata}(y_i - \hat{y}_i)^2, \quad (10.11)$$

bernilai minimum. Pada persamaan (10.11), $ndata$ menyatakan banyaknya data pengamatan. Pada persamaan (10.11), $y_i, \hat{y}_i$ berturut-turut adalah data berat ayam jantan pada hari ke-i dan berat ayam jantan yang diperoleh dari model pertumbuhan logistik pada persamaan (10.4).

Tabel 10.1. Rata-rata berat badan ayam jantan

t	$y(t)$	t	$y(t)$
0	37	42	519.72
3	41.74	45	577.27
6	59.19	48	633.59
9	79.94	51	667.18
12	102.96	54	717.17
15	132.13	57	786.35
18	170.18	71	1069.28
21	206.56	85	1326.49
24	250.71	99	1589.71
27	285.27	113	1859.26
30	324.92	127	2015.44
33	372.83	141	2142.31
36	417.41	155	2220.54
39	469.13	170	2262.63

Implementasi metode optimisasi swarm partikel menggunakan perangkat lunak Scilab untuk menentukan parameter model pertumbuhan logistik tersebut disajikan pada subbab berikutnya.

10.2.1. Implementasi Program

Kode program yang digunakan untuk mengestimasi parameter model pertumbuhan logistik menggunakan metode optimisasi swarm partikel terdiri dari dua file subprogram, yaitu:

(3) MSE_logistik.sce (berupa suatu fungsi/*function*)
(4) osp_logistik.sce (berupa suatu *script*)

Diasumsikan bahwa kedua file kode program tersebut disimpan dalam folder D:\File_Scilab. Kode program 10.1 dan kode program 10.2 berurut-turut merepresentasikan fungsi **MSE_logistik.sce** dan *script* **osp_logistik.sce.**

Kode program 10.1 (MSE_logistik.sce)

```
//Kode program 10.1 (MSE_logistik.sce)
function [yhitung, MSE] = MSE_logistik(posisi)
   global tdata ydata ndata; // Variabel global tdata ydata ndata

   // Parameter r, K, tinf
   r = posisi(1);
   K = posisi(2);
   tinf = posisi(3);
   yhitung = K ./ (1 + exp(-r*(tdata - tinf)));
   MSE = 1/ndata*sum((ydata - yhitung).^2);
endfunction
//Akhir kode program
```

File fungsi tersebut digunakan untuk menghitung nilai MSE (rata-rata kuadrat galat) dari setiap solusi model. File fungsi **MSE_logistik.sce** mempunyai satu input, yaitu variabel **posisi** yang

berbentuk vektor. Parameter r, K, t_{inf} berturut-turut direpresentasikan dengan **posisi**(1), **posisi**(2) dan **posisi**(3).

Kode Program 10.2 (osp_logistik.sce)

```
//Kode program 10.2. (osp_logistik.sce)
clc;
clear;
xdel(winsid()); // Close all figure
tic( );

// Variabel global
global tdata ydata ndata;

// tdata dan xdata
tdata = [0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, ...
      48, 51, 54, 57, 71, 85, 99, 113, 127, 141, 155, 170].';
ydata = [37, 41.74, 59.19, 79.94, 102.96, 132.13, 170.18, 206.56, ...
      250.71, 285.27, 324.92, 372.83, 417.41, 469.13, 519.72, ...
      577.27, 633.59, 667.18, 717.17, 786.35, 1069.28, 1326.49, ...
      1589.71, 1859.26, 2015.44, 2142.31, 2220.54, 2262.63].';
ndata = length(tdata);

nVar = 3;           // Banyaknya variabel keputusan
VarMin = [0, 2000, 40];     // Nilai minimum variabel keputusan
VarMax = [1, 5000, 90];     // Nilai maksimum variabel keputusan

// Membuka file function
exec('MSE_logistik.sce',-1);

// Random seed generator
n = sum(clock( ));
rand("seed",n);

// Parameters OSP
MaxIt = 200;    // Maksimum banyaknya iterasi
nPop = 40;       // Ukuran populasi (ukuran swarm)
w = 0;           // Bobot inersia
c1 = 2.0;       // Koefisien personal
c2 = 2.0;       // Koefisien global

// Batas nilai kecepatan partikel
VelMax = 0.1*(VarMax - VarMin);
VelMin = - VelMax;

// Inisialisasi
```

```
posisi = rand(nPop, nVar);
kecepatan = zeros(nPop, nVar);
biaya  = zeros(nPop, 1);
pbest  = zeros(nPop, nVar);
biaya_pbest = zeros(nPop, 1);
gbest = zeros(nPop, 1);
biaya_gbest = zeros(1,1);

// Inisialisasi
for i = 1:nPop
   // Inisialisasi posisi partikel
   posisi(i, :) = VarMin + (VarMax - VarMin) .* rand(1,nVar);

   // Hitung nilai fungsi tujuan untuk masing-masing posisi
   [yhitung, biaya(i)] = MSE_logistik(posisi(i,:));

   // Update partikel terbaik lokal
   pbest(i,:) = posisi(i,:);
   biaya_pbest(i) = biaya(i);

   // Update partikel terbaik global
   if (i == 1)
      gbest = posisi(i,:);
      biaya_gbest = biaya(i);
      yhitung_terbaik = yhitung;
   else
      if (biaya(i) < biaya_gbest)
         gbest = posisi(i,:);
         biaya_gbest = biaya(i);
         yhitung_terbaik = yhitung;
      end // if (biaya(i) < biaya_gbest)
   end // if (i == 1)
end // for i = 1:nPop
4
BestCost = zeros(MaxIt,1);
// Loop metode OSP
for it = 1:MaxIt
   disp('Iterasi = ' + string(it));
   for i=1:nPop
      // Update kecepatan partikel
      kecepatan(i,:) = w*kecepatan(i,:) + c1 * rand(1,nVar) .* (pbest(i,:) - ...
                       posisi(i,:)) + c2 * rand(1,nVar) .* (gbest - posisi(i,:));

      // Terapkan nilai batas kecepatan partikel
      kecepatan(i,:) = max(kecepatan(i,:), VelMin);
      kecepatan(i,:) = min(kecepatan(i,:), VelMax);
```

```
        // Update posisi partikel
        posisi(i,:) = posisi(i,:) + kecepatan(i,:);

        // Terapkan nilai batas posisi partikel
        posisi(i,:) = max(posisi(i,:), VarMin);
        posisi(i,:) = min(posisi(i,:), VarMax);

        // Hitung nilai fungsi tujuan untuk masing-masing posisi
        [yhitung, biaya(i)] = MSE_logistik(posisi(i,:));

        // Update posisi partikel terbaik lokal
        if (biaya(i) < biaya_pbest(i))
            pbest(i,:) = posisi(i,:);
            biaya_pbest(i) = biaya(i);
        // Update posisi partikel terbaik global
            if (biaya(i) < biaya_gbest)
                    gbest = posisi(i,:);
                    biaya_gbest = biaya(i);
                    yhitung_terbaik = yhitung;
            end // if (biaya(i) < biaya_gbest)
        end // if (biaya(i) < biaya_pbest(i))
    end // for i = 1:nPop

    BestCost(it) = biaya_gbest;

   disp('Iterasi ' + string(it) + ', r = ' + string(gbest(1)) + ...
     ', K = ' + string(gbest(2)) + ', tinf ' + string(gbest(3)) + ...
     ', MSE = ' + string(BestCost(it)));
end // for it = 1:MaxIt

// Ekstrak parameter terbaik
dt = 0.1;
r = gbest(1);
K = gbest(2);
tinf = gbest(3);
tt = (tdata(43):dt:tdata($)).';
yhitung = K ./ (1 + exp(-r*(tdata - tinf)));
yy = K ./ (1 + exp(-r*(tt - tinf)));

// Menghitung laju pertumbuhan berat badan ternak terhadap waktu
dy = r*yy.*(1 - yy/K);

// Grafik nilai fungsi tujuan
scf(1);
xn = (1:1:MaxIt).';
```

```
plot2d('nl',xn,BestCost,style = [1]);
title('Nilai minimum fungsi tujuan',"fontsize",3.5);
xlabel('Iterasi',"fontsize",3.5);
ylabel('Nilai z',"fontsize",3.5);
gca().grid=[1 1 1]; // Setara dengan grid on;
gca().children(1).children(1).thickness = 3; // Set ketebalan garis kurva
axes_handle.grid = [1 1 1]; // Setara dengan grid(axes_handle,"on")

scf(2);
plot(tdata,ydata,'ko',"linewidth",3.5);
set(gca(),"auto_clear","off"); // hold on
plot(tt,yy,'k',"linewidth",3.5);
title('Berat ayam jantan (gram)',"fontsize",3.5);
xlabel('Waktu (hari)',"fontsize",3.5);
ylabel('Berat ayam jantan (gram)',"fontsize",3.5);
gca().grid=[1 1 1]; // Setara dengan grid on;
gca().children(1).children(1).thickness = 3; // Set ketebalan garis kurva
axes_handle.grid = [1 1 1]; // Setara dengan grid(axes_handle,"on")

scf(3);
plot2d('nn',tt,dy,style = [1]);
title('Pertambahan berat ternak / hari (gram/hari)',"fontsize",3.5);
xlabel('Waktu (hari)',"fontsize",3.5);
ylabel('Pertambahan berat ternak (gram/hari)',"fontsize",3.5);
gca().grid=[1 1 1]; // Setara dengan grid on;
gca().children(1).children(1).thickness = 3; // Set ketebalan garis kurva
axes_handle.grid = [1 1 1]; // Setara dengan grid(axes_handle,"on")

hasil_PSO = [gbest, biaya_gbest];
waktu = toc( );
disp('waktu = ' + string(waktu) + ' detik');
//Akhir kode program
```

Simpanlah kode program 10.2 berupa script tersebut dengan nama **osp_logistik.sce**.

Berikut disajikan beberapa penjelasan yang berhubungan dengan kode program 10.2 tersebut:

(f) Blok perintah

```
nVar = 3;           // Banyaknya variabel keputusan
VarMin = [0, 2000, 40];     // Nilai minimum variabel keputusan
VarMax = [1, 5000, 90];     // Nilai maksimum variabel keputusan
```

berhubungan dengan banyaknya variabel keputusan (banyaknya parameter yang diestimasi) pada model pertumbuhan logistik, yaitu parameter r, K dan t_{inf}. Nilai parameter r, K dan t_{inf} diduga terletak pada interval tutup [0, 1], [2000, 5000] dan [40, 90]. Akibatnya, pada blok perintah tersebut, didefinisikan **nVar = 3**, **VarMin = [0, 2000, 40]**, dan **VarMax = [1, 5000, 90]**.

(g) Perintah **exec('MSE_logistik.sce', -1)** digunakan untuk mengaktifkan file script/function **MSE_logistik.sce**.

10.2.2. Menjalankan Program

Prosedur berikut dapat digunakan untuk menjalankan untuk menjalankan file kode program **osp_logistik.sce** yang tersimpan dalam folder D:\File_Scilab:

(m) Bukalah perangkat lunak Scilab.

(n) Aktifkan folder D:\File_Scilab melalui klik menu *File – Current working directory – Browse for new*. Pada kotak dialog *Select directory*, pilih folder **D:\Program_Scilab**. Selanjutnya klik tombol *Open*.

(o) Selanjutnya klik menu *File – Open a file*. Pada kotak dialog Select a file to open, klik file **osp_logistik.sce**. Selanjutnya klik tombol *Open*.

(p) Pada jendela **osp_logistik.sce**, klik menu *Execute – save and execute*.

Berikut disajikan contoh tampilan hasil eksekusi kode program osp_p93.sce.

```
Iterasi = 200
 Iterasi 200, r = 0.0403352, K = 2279.9042, tinf = 74.67749, MSE = 1887.4613
 waktu = 13.648349 detik
```

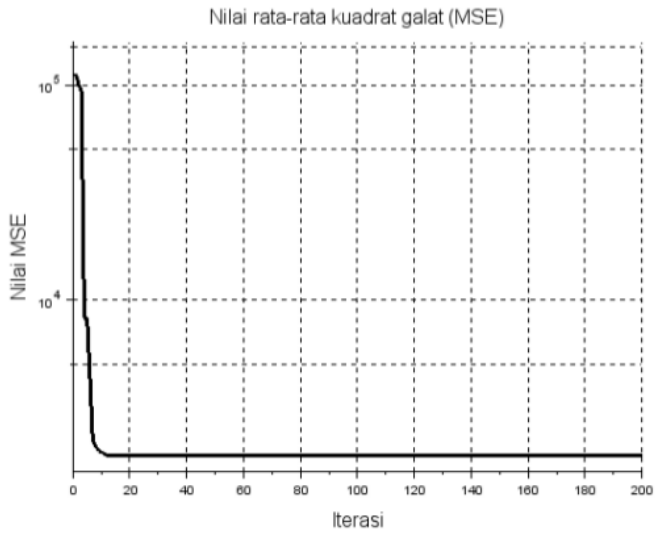
Setelah 200 iterasi, diperoleh nilai parameter $r \approx 0.0403, K \approx 2279.90$ gram dan $t_{inf} \approx 74.68$ hari, dengan nilai $MSE \approx 1887.46$. Waktu eksekusi kode program sekitar 13.6 detik. Untuk memperoleh hasil yang lebih akurat, metode optimisasi swarm partikel diimplementasikan sebanyak 25 kali dan ditentukan parameter terbaik dari 25 kali implementasi metode tersebut. Parameter terbaik dan simpangan baku nilai MSE untuk berbagai variasi nilai koefisien inersia (w) disajikan pada Tabel 10.1. Kode program yang digunakan untuk menjalankan metode optimisasi swarm partikel sebanyak 25 kali, disajikan pada Lampiran 2.

Tabel 10.1. Solusi terbaik yang untuk masalah estimasi parameter model pertumbuhan logistik

w	r	K	t_{inf}	MSE terbaik	Simpangan baku MSE
-0.3	0.0403176	2280.0941	74.71318	1887.5715	16.574055
-0.2	0.0403360	2279.8820	74.67641	1887.4613	0.1422552
-0.1	0.0403352	2279.9042	74.67749	1887.4613	$3.810 * 10^{-8}$
0	0.0403352	2279.9042	74.67749	1887.4613	$1.233 * 10^{-12}$
0.1	0.0403352	2279.9042	74.67749	1887.4613	$7.000 * 10^{-13}$
0.2	0.0403352	2279.9042	74.67749	1887.4613	$8.520 * 10^{-13}$
0.3	0.0403352	2279.9042	74.67749	1887.4613	$4.687 * 10^{-13}$
0.4	0.0403352	2279.9042	74.67749	1887.4613	$6.034 * 10^{-13}$
0.5	0.0403352	2279.9042	74.67749	1887.4613	$8.720 * 10^{-13}$
0.6	0.0403352	2279.9042	74.67749	1887.4613	1263.0991
0.7	0.0403352	2279.9041	74.67750	1887.4613	1748.6815
0.8	0.0403322	2279.9781	74.68083	1887.4617	1263.0790
0.9	0.0404223	2277.4233	74.55918	1887.9662	2092.5610
1.0	0.0404594	2274.2095	74.70740	1897.3011	1735.2379

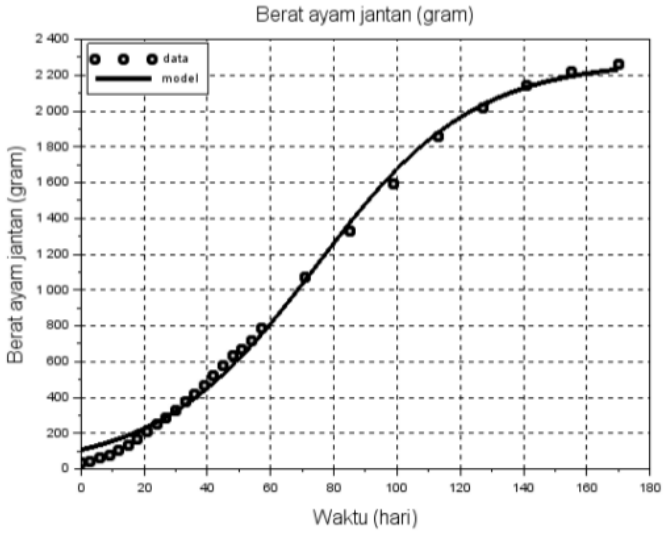
Tabel 10.1 tersebut mengindikasikan bahwa nilai koefisien inersia (w) berpengaruh terhadap performansi metode optimisasi *swarm* partikel dalam estimasi nilai parameter model pertumbuhan

logistik. Pada permasalahan estimasi parameter model logistik, metode optimisasi swarm parameter menghasilkan nilai parameter dengan rata-rata kuadrat galat (MSE) minimum dan simpangan baku nilai MSE sangat kecil ketika nilai koefisien inersia berada dalam rentang $-0.1 \leq w \leq 0.5$. Grafik perubahan nilai MSE ketika koefisien inersia $w = 0$ disajikan pada Gambar 10.1.

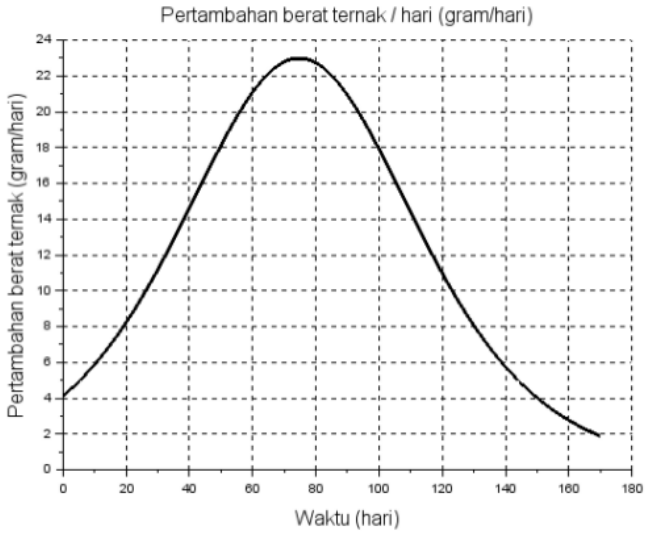


Gambar 10.1. Perubahan nilai MSE pada estimasi parameter model pertumbuhan logistik menggunakan koefisien inersia w = 0.

Grafik berat ayam jantan dan pertambahan berat ayam perhari yang diperoleh dari model pertumbuhan logistik berturut-turut pada Gambar 10.2 dan Gambar 10.3.



Gambar 10.2. Grafik data berat ayam jantan (tanda bulat) dan prediksi berat ayam jantan yang diperoleh dari model pertumbuhan logistik.



Gambar 10.3. Grafik data berat ayam jantan (tanda bulat) dan predikti berat ayam jantan yang diperoleh dari model pertumbuhan logistik.

Gambar 10.2 menunjukkan bahwa model pertumbuhan logistik dapat digunakan untuk mendeskripsikan dinamika pertumbuhan ayam jantan. Galat terbesar terjadi pada fase awal pertumbuhan, dan seiring berjalannya waktu, hasil prediksi model pertumbuhan logistik semakin akurat. Gambar 10.3 mendeskripsikan grafik pertambahan berat ayam jantan per satuan waktu. Pertambahan ayam jantan semakin meningkat dan mencapai puncak pertumbuhan ketika waktu t sama dengan $t_{inf} \approx 74.68$ hari, dengan tingkat pertumbuhan berat ayam jantan sekitar 23 gram/hari. Setelah waktu t_{inf}, laju pertumbuhan berat ayam jantan menurun, dan pada hari ke-170 pertumbuhan berat ayam diperkirakan sekitar 2 gram/hari.

10.3. Estimasi parameter model pertumbuhan logistik orde fraksional

10.3.1. Metode numerik untuk system persamaan diferensial fraksional

Persamaan diferensial logistik pada persamaan (10.3) telah diperumum menjadi persamaan diferensial logistik orde fraksional yang berbentuk [35]

$$\frac{d^{\alpha}y}{dt^{\alpha}} = ry\left(1-\frac{y}{K}\right), y(0) = y_0 \geq 0. \quad (10.12)$$

Pada persamaan (10.12), α merupakan orde turunan fraksional dengan $0 < \alpha \leq 1$. Pada tulisan ini, turunan fraksional yang digunakan adalah turunan fraksional versi Caputo. Untuk sebarangan nilai awal y_0 berupa bilangan positif dengan $y_0 \neq K$, solusi eksak persamaan diferensial fraksional logistik orde fraksional tersebut tidak diketahui.

Metode prediktor-korektor Adam-Bashforth-Mouton merupakan salah satu metode numerik yang dapat digunakan untuk memperoleh hampiran solusi suatu persamaan diferensial atau sistem persamaan diferensial orde fraksional versi Caputo. Pada bagian ini, penurunan metode preditor-korektor Adam-Bashforth-Moulton dirujuk dari Diethelm dan Freed [36]. Misalkan

diberikan sistem persamaan diferensial fraksional dengan turunan fraksional Caputo yang berbentuk

$$\frac{d^{\alpha}x}{dt^{\alpha}} = f(t,x), 0 < \alpha \leq 1 \tag{10.13}$$

dengan $x \in R^d$ dan nilai awal $x(0)$ diketahui. Sistem persamaan diferensial orde fraksional (10.13) dapat diubah menjadi persamaan integral Volterra yang berbentuk

$$x(t) = x(0) + \frac{1}{\Gamma(\alpha)}\int_0^t (t-u)^{\alpha-1} f\big(u, x(u)\big)\, du. \tag{10.14}$$

Misalkan interval $[0, t_f]$ dipartisi menjadi himpunan titik partisi $P = \{t_1, t_2, \dots t_{nt}\}$ dengan $t_1 = 0, t_2 = h, t_3 = 2h, \dots, t_{nt} = (nt-1)h = t_f$ dan misalkan $x_k \approx x\big((k-1)h\big),\ k = 2, 3, \dots, nt.$

Dengan menggunakan (10.14), maka untuk setiap $k = 2,3, \dots, nt$ diperoleh

$$x_k \approx x\big((k-1)h\big) = x(0) + \frac{1}{\Gamma(\alpha)}\int_0^{t_k} (t_k - u)^{\alpha-1} f\big(u, x(u)\big)\, du$$

$$= x_1 + \frac{1}{\Gamma(\alpha)}\int_{t_1}^{t_k} (t_k - u)^{\alpha-1} f(u, x(u))\, du. \tag{10.15}$$

Dengan menghampiri fungsi $f(u, x(u))$ dengan fungsi konstan sepotong-sepotong (*piecewise constant function*), maka integral pada persamaan (10.15) dapat dihampiri dengan

$$\int_{t_1}^{t_k} (t_k - u)^{\alpha-1} f(u, x(u))\, du \approx \sum_{i=1}^{k-1} b_{i,k}\, f(t_i, x_i)$$

dengan

$$b_{i,k} = [(k-i)^{\alpha} - (k-i-1)^{\alpha}]. \tag{10.16}$$

Akibatnya, nilai prediktor $x_k^P, k = 2,3, \dots, nt$ diberikan oleh

$$x_k^P = x_1 + \frac{h^{\alpha}}{\alpha}\frac{1}{\Gamma(\alpha)}\sum_{i=1}^{k-1} b_{i,k}\, f(t_i, x_i). \tag{10.17}$$

Karena $\alpha\Gamma(\alpha) = \Gamma(\alpha+1)$ berlaku untuk setiap $\alpha > 0$, maka prediktor x_k^P pada persamaan (10.17) dapat dituliskan ke dalam bentuk

$$x_k^P = x_1 + \frac{h^{\alpha}}{\Gamma(\alpha+1)}\sum_{i=1}^{k-1} b_{i,k}\, f(t_i, x_i), \tag{10.18}$$

dengan $b_{i,k}$ diberikan oleh persamaan (10.16). Formula predikor pada persamaan (10.18) dikenal sebagai formula prediktor Adams-Bashforth untuk persamaan diferensial orde fraksional.

Formula korektor merupakan versi fraksional dari metode Adam-Moulton, yang dapat berupa metode satu-langkah (*single-step*) atau multi-langkah (*multi-step*). Dengan menghampiri fungsi $f(u, x(u))$ dengan fungsi linear sepotong-sepotong (*piecewise linear function*), maka integral pada persamaan (10.15) dapat dihampiri dengan

$$\int_{t_1}^{t_k} (t_k - u)^{\alpha-1} f(u, x(u))\, du \approx \frac{h^\alpha}{\alpha(\alpha+1)} \sum_{i=1}^{k} a_{i,k}\, f(t_i, x_i)$$

dengan

$$a_{i,k} = \begin{cases} (k-2)^{\alpha+1} - (k-1)^{\alpha+1} + (\alpha+1)(k-1)^{\alpha}, & \text{jika } i = 1, \\ (k-i+1)^{\alpha+1} - 2(k-i)^{\alpha+1} + (k-i-1)^{\alpha+1}, & \text{jika } 2 \le i \le k-1, \\ 1, & \text{untuk } i = k. \end{cases} \quad (10.19)$$

Akibatnya, korektor untuk x_k diberikan oleh

$$x_k = x_1 + \frac{1}{\Gamma(\alpha)} \frac{h^\alpha}{\alpha(\alpha+1)} \left(\sum_{i=1}^{k-1} a_{i,k}\, f(t_i, x_i) + f(t_k, x_k^P) \right), \quad (10.20)$$

dengan $a_{i,k}$ diberikan dalam persamaan (10.19). Mengingat $\alpha > 0$, maka dari sifat fungsi Gamma diperoleh $(\alpha+1)\alpha\Gamma(\alpha) = \Gamma(\alpha+2)$. Akibatnya, korektor untuk prediktor x_k pada persamaan (10.20) dapat dituliskan ke dalam bentuk

$$x_k = x_1 + \frac{h^\alpha}{\Gamma(\alpha+2)} \left(\sum_{i=1}^{k-1} a_{i,k}\, f(t_i, x_i) + f(t_k, x_k^P) \right), \quad (10.21)$$

dengan koefisien $a_{i,k}$ diberikan oleh persamaan (10.19). Formula predikor-korektor pada persamaan (10.18) dan (10.21) disebut sebagai formula prediktor-korektor Adams-Bashforth-Moulton untuk persamaan diferensial orde fraksional.

Selanjutnya, metode optimisasi swarm partikel digunakan untuk mementukan parameter model pertumbuhan logistik orde fraksional pada persamaan (10.12) untuk kasus pertumbuhan ayam jantan. Data berat ayam jantan disajikan pada Tabel 10.1. Seperti pada bagian sebelumnya, parameter α, r, K diestimasi sedemikian hingga rata-rata kuadrat galat (*mean square error*) MSE bernilai minimum. Implementasi metode optimisasi swarm partikel menggunakan perangkat lunak Scilab untuk menentukan parameter model pertumbuhan logistik orde fraksional disajikan pada subbab berikutnya.

10.3.2. Implementasi Program

Kode program yang digunakan untuk mengestimasi parameter model pertumbuhan logistik menggunakan metode optimisasi swarm partikel terdiri dari dua file subprogram, yaitu:
(1) flogistik.sce (berupa suatu fungsi/*function*)
(2) pk_flogistik.sce (berupa suatu fungsi/*function*)
(3) MSE_fraksional_logistik.sce (berupa suatu fungsi/*function*)
(4) osp_fraksional_logistik.sce (berupa suatu *script*)
Diasumsikan bahwa kedua file kode program tersebut disimpan dalam folder D:\File_Scilab. Kode program 10.3, kode program 10.4 dan kode program 10.5 berurut-turut merepresentasikan fungsi flogistik.sce, pk_flogistik.sce, MSE_fraksional_logistik.sce dan *script* osp_logistik.sce.

Kode program 10.3 (flogistik.sce)

```
//Kode program 10.3 (flogistik.sce)
function fy = flogistik(t, y, param)
   // Ekstrak parameter
   alpha = param(1);
```

```
    r = param(2);
    K = param(3);
    // Inisialisasi
    [nbar, nkol] = size(t);
    fy = zeros(nbar, 1);
    fy = r * y .* (1-y/K);
endfunction
// Akhir kode program flogistik.sce
```

File fungsi tersebut digunakan untuk merepresentasikan model fraksional logistik. Argumen **param** memuat parameter model, yaitu α, r dan K.

Kode program 10.4 (pk_flogistik.sce)

```
//Kode program 10.4 (pk_flogistik.sce)
function [tt, yk] = pk_flogistik(t0, tf, dt, y0, param)
    // Ekstrak parameter
    alpha = param(1);
    r = param(2);
    K = param(3);

    // Membuka file flogistik.sce
    exec('flogistik.sce',-1);

    // Inisialisasi
    tt = (t0:dt:tf).';
    nt = length(tt);
    yp = zeros(nt,1);
    yk = zeros(nt,1);
    yp(1,:) = y0;
    yk(1,:) = y0;
    [nbar, nkol] = size(y0);

    // Hitung solusi model
    for k = 2:nt
        // Hitung prediktor
        vb = (k - (1:k-1)).^alpha - (k-1 - (1:k-1)).^alpha;
        // vb adalah vektor baris ukuran 1 x (k - 1)
        fyp = flogistik(tt(1:k-1), yp(1:k-1,:), param);
        // fyp adalah vektor berukuran (k - 1) x banyaknya kolom yp
        yp(k, :) = y0 + dt^alpha/gamma(alpha + 1) * vb * fyp;

        // Hitung korektor
        // va adalah vektor baris berukuran 1 x k
        va = (k+1 - (1:k)).^(alpha+1) - 2 * (k - (1:k)).^(alpha+1) + ...
             (k-1 - (1:k)).^(alpha+1);
```

```
        va(k) = 1;
        va(1) = (k - 2)^(alpha+1) - (k -1)^(alpha + 1) + (alpha + 1)*(k - 1)^alpha;

        // fyk berukuran k x banyaknya kolom yk
        fyk =   flogistik(tt(1:k), yp(1:k,:), param);

        // Hitung ykorektor
        yk(k,:) = y0 +  dt^alpha/gamma(alpha + 2) * va * fyk;
        yp = yk;
    end // for k = 2:nt
endfunction
// Akhir kode program pk_flogistik.sce
```

Kode program 10.5 (MSE_fraksional_logistik.sce)

```
//Kode program 10.5 (MSE_fraksional_logistik.sce)
function [yhitung, yy, MSE] = MSE_fraksional_logistik(param)
    global tdata ydata tt dt indeks_t y0;
    // Ekstrak parameter
    alpha = param(1);
    r = param(2);
    K = param(3);

    // Membuka file function
    exec('pk_flogistik.sce',-1);

    // Hitung solusi menggunakan metode prediktor-korektor
    ndata = length(tdata);
    t0 = tdata(1);
    tf = tdata(ndata);

    // Ekstrak xx, xhitung, yy dan yhitung
    if (alpha > 0) && (alpha < 1) then
        [tt,zz] = pk_flogistik(t0,tf, dt, y0, param);
        yy = zz(:,1);
        yhitung = zz(indeks_t,1);
    end
    if (alpha == 1) then
        tt = (t0:dt:tf).';
        yy = K ./ (1 +((K/y0) - 1)*exp(-r*tt));
        yhitung = K ./ (1 +((K/y0) - 1)*exp(-r*tdata));
    end

    // Hitung MSE
    MSE = 1/ndata*sum((ydata - yhitung).^2);
endfunction
//Akhir kode program
```

File fungsi tersebut digunakan untuk menghitung nilai MSE (rata-rata kuadrat galat) dari setiap solusi model. Solusi model dihitung menggunakan metode predictor-korektor. File fungsi **MSE_fraksional_logistik.sce** mempunyai satu input, yaitu variabel **param** yang berbentuk vektor. Parameter α, r, K berturut-turut direpresentasikan dengan **param**(1), **param**(2) dan **param**(3).

Kode Program 10.6 (osp_fraksional_logistik.sce)

```
//Kode program 10.6 (osp_fraksional_logistik.sce)
clc;
clear;
xdel(winsid()); // Close all figure

tic( );

// Variabel global
global tdata ydata tt dt indeks_t y0;

// tdata dan xdata
tdata = [0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, ...
        48, 51, 54, 57, 71, 85, 99, 113, 127, 141, 155, 170].';
ydata = [37, 41.74, 59.19, 79.94, 102.96, 132.13, 170.18, 206.56, ...
        250.71, 285.27, 324.92, 372.83, 417.41, 469.13, 519.72, ...
        577.27, 633.59, 667.18, 717.17, 786.35, 1069.28, 1326.49, ...
        1589.71, 1859.26, 2015.44, 2142.31, 2220.54, 2262.63].';
ndata = length(tdata);
y0 = ydata(1);
dt = 0.2;
tt = (tdata(1):dt:tdata(ndata)).';
nt = length(tt);
indeks_t = zeros(ndata,1);
mm = 1;
for kk = 1:nt
    if (tt(kk) == tdata(mm))
        indeks_t(mm,:) = kk;
        mm = mm + 1;
    end
end // for kk = 1:nt

// nVar, VarMin, VarMax
nVar = 3;                  // Banyaknya variabel keputusan
VarMin = [0.3, 0, 2000];      // Nilai minimum variabel keputusan
VarMax = [1,   0.4 5000];      // Nilai maksimum variabel keputusan

// Membuka file function
exec('MSE_fraksional_logistik.sce',-1);
```

```
// Random seed generator
n = sum(getdate("s"));
rand("seed",n);

// Parameters OSP
MaxIt = 100;    // Maksimum banyaknya iterasi
nPop = 25;       // Ukuran populasi (ukuran swarm)
w = 0.5;         // Bobot inersia
c1 = 2.0;       // Koefisien personal
c2 = 2.0;       // Koefisien global

// Batas nilai kecepatan partikel
VelMax = 0.1*(VarMax - VarMin);
VelMin = - VelMax;

// Inisialisasi
posisi = rand(nPop, nVar);
kecepatan = zeros(nPop, nVar);
biaya  = zeros(nPop, 1);
pbest  = zeros(nPop, nVar);
biaya_pbest = zeros(nPop, 1);
gbest = zeros(nPop, 1);
biaya_gbest = zeros(1,1);

// Inisialisasi
for i = 1:nPop
   // Inisialisasi posisi partikel
   posisi(i, :) = VarMin + (VarMax - VarMin) .* rand(1,nVar);

   // Hitung nilai fungsi tujuan untuk masing-masing posisi
   [yhitung, yy,biaya(i)] = MSE_fraksional_logistik(posisi(i,:));

   // Update partikel terbaik lokal
   pbest(i,:) = posisi(i,:);
   biaya_pbest(i) = biaya(i);

   // Update partikel terbaik global
   if (i == 1)
      gbest = posisi(i,:);
      biaya_gbest = biaya(i);
   else
      if (biaya(i) < biaya_gbest)
         gbest = posisi(i,:);
      7  biaya_gbest = biaya(i);
      end
```

```
    end
end

BestCost = zeros(MaxIt,1);
// Loop utama OSP
for it = 1:MaxIt
    disp('Iterasi = ' + string(it));
    for i=1:nPop
        disp('Iterasi = ' + string(it) + ', partikel ke ' + string(i));
        // Update kecepatan partikel
        kecepatan(i,:) = w*kecepatan(i,:) + c1 * rand(1,nVar) .* (pbest(i,:) - ...
                         posisi(i,:)) + c2 * rand(1,nVar) .* (gbest - posisi(i,:));

        // Terapkan nilai batas kecepatan partikel
        kecepatan(i,:) = max(kecepatan(i,:), VelMin);
        kecepatan(i,:) = min(kecepatan(i,:), VelMax);

        // Update posisi partikel
        posisi(i,:) = posisi(i,:) + kecepatan(i,:);

        // Terapkan nilai batas posisi partikel
        posisi(i,:) = max(posisi(i,:), VarMin);
        posisi(i,:) = min(posisi(i,:), VarMax);

        // Hitung nilai fungsi tujuan untuk masing-masing posisi
        [yhitung, yy, biaya(i)] = MSE_fraksional_logistik(posisi(i,:));

        // Update posisi partikel terbaik lokal
        if (biaya(i) < biaya_pbest(i))
            pbest(i,:) = posisi(i,:);
            biaya_pbest(i) = biaya(i);
        // Update posisi partikel terbaik global
            if (biaya(i) < biaya_gbest)
                    gbest = posisi(i,:);
                    biaya_gbest = biaya(i);
                    yhitung_terbaik = yhitung;
                    yy_terbaik = yy;
            end
        end // if (biaya(i) > biaya_pbest(i))
    end // for i = 1:nPop

    BestCost(it) = biaya_gbest;

    disp('Iterasi ' + string(it) + ', alpha = ' + string(gbest(1)) + ...
    ', r = ' + string(gbest(2)) +  ', K = ' + string(gbest(3)) + ...
    ', MSE = ' + string(BestCost(it)));
```

```
end

dy = zeros(length(tt)-1,1);
dy = (yy_terbaik(2:$)-yy_terbaik(1:$-1))/dt;

// Waktu optimal pertumbuhan berat ayam jantan
idmaks = find(dy == max(abs(dy(10:$))));
toptimal = tt(idmaks);
disp('Waktu pertumbuhan optimal sekitar  ' + string(tt(idmaks)) + ' hari.');

// Grafik nilai fungsi tujuan
scf(1);
xn = (1:1:MaxIt).';
plot2d('nl',xn,BestCost,style = [1]);
title('Nilai rata-rata kuadrat galat (MSE)',"fontsize",3.5);
xlabel('Iterasi',"fontsize",3.5);
ylabel('Nilai MSE',"fontsize",3.5);
gca().grid=[1 1 1]; // Setara dengan grid on;
gca().children(1).children(1).thickness = 3; // Set ketebalan garis kurva
axes_handle.grid = [1 1 1]; // Setara dengan grid(axes_handle,"on")

scf(2);
plot2d('nn',tt(1:length(tt)-1),dy,style = [1]);
title('Pertambahan berat ternak / hari (gram/hari)',"fontsize",3.5);
xlabel('Waktu (hari)',"fontsize",3.5);
ylabel('Pertambahan berat ternak (gram/hari)',"fontsize",3.5);
gca.grid=[1 1 1]; // Setara dengan grid on;
gca().children(1).children(1).thickness = 3; // Set ketebalan garis kurva
axes_handle.grid = [1 1 1]; // Setara dengan grid(axes_handle,"on")

scf(3);
plot(tdata,ydata,'ko',"linewidth",3.5);
set(gca(),"auto_clear","off"); // hold on
plot(tt,yy_terbaik,'k',"linewidth",3.5);
hl=legend(['data';'model'],[2]);
title('Berat ayam jantan (gram)',"fontsize",3.5);
xlabel('Waktu (hari)',"fontsize",3.5);
ylabel('Berat ayam jantan (gram)',"fontsize",3.5);
gca().grid=[1 1 1]; // Setara dengan grid on;
axes_handle.grid = [1 1 1]; // Setara dengan grid(axes_handle,"on")

waktu = toc( );
hasil_OSP = [gbest, biaya_gbest];
disp('waktu = ' + string(waktu) + ' detik');
 //Akhir kode program
```

Simpanlah kode program 10.6 berupa script tersebut dengan nama **osp_fraksional_logistik.sce**.

Berikut disajikan beberapa penjelasan yang berhubungan dengan kode program 10.5 tersebut:

(a) Blok perintah

```
nVar = 3;                // Banyaknya variabel keputusan
VarMin = [0.3, 0, 2000];     // Nilai minimum variabel keputusan
VarMax = [1.0,  0.4, 5000];     // Nilai maksimum variabel keputusan
```

berhubungan dengan banyaknya variabel keputusan (banyaknya parameter yang diestimasi) pada model pertumbuhan logistik, yaitu parameter r, K dan t_{inf}. Nilai parameter r, K dan t_{inf} diduga terletak pada interval tutup [0.3, 1], [0, 0.4] dan [2000, 5000]. Akibatnya, pada blok perintah tersebut, didefinisikan **nVar = 3**, **VarMin = [0.3, 0, 2000]**, dan **VarMax = [1.0, 0.4, 5000]**.

(b) Perintah **exec('MSE_fraksional_logistik.sce', -1)** digunakan untuk mengaktifkan file script/function **MSE_fraksional_logistik.sce**.

10.3.3. Menjalankan Program

Prosedur berikut dapat digunakan untuk menjalankan untuk menjalankan file kode program **osp_fraksional_logistik.sce** yang tersimpan dalam folder D:\File_Scilab:

(a) Bukalah perangkat lunak Scilab.
(b) Aktifkan folder D:\File_Scilab melalui klik menu *File – Current working directory – Browse for new*. Pada kotak dialog *Select directory*, pilih folder **D:\Program_Scilab**. Selanjutnya klik tombol *Open*.
(c) Selanjutnya klik menu *File – Open a file*. Pada kotak dialog Select a file to open, klik file **osp_logistik.sce**. Selanjutnya klik tombol *Open*.
(d) Pada jendela **osp_logistik.sce**, klik menu *Execute – save and execute*.

Berikut disajikan contoh tampilan hasil eksekusi kode program osp_fraksional_logistik.sce.

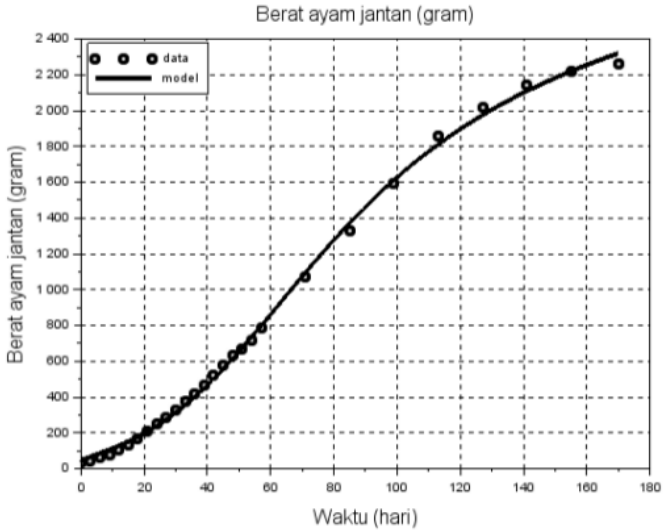
```
Iterasi = 99
Iterasi 99, alpha = 0.4463648, r = 0.267129, K = 3624.2819, MSE = 626.95544
Iterasi = 100
Iterasi 100, alpha = 0.4463648, r = 0.267129, K = 3624.2819, MSE = 626.95544
Waktu pertumbuhan optimal sekitar 62.6 hari.
  waktu = 1826.4193 detik
```

Setelah 100 iterasi, diperoleh nilai parameter $\alpha \approx 0.446365, r \approx 0.267129, K \approx 3624.28$ gram dengan nilai $MSE \approx 629.96$. Waktu eksekusi kode program sekitar 1827 detik. Untuk memperoleh hasil yang lebih akurat, metode optimisasi swarm partikel diimplementasikan sebanyak 5 kali dan ditentukan parameter terbaik dari 5 kali implementasi metode tersebut. Hasil implementasi metode optimiasasi swarm parameter dengan menggunakan koefisien inersia $w = 0.5$, disajikan pada Tabel 10.2.

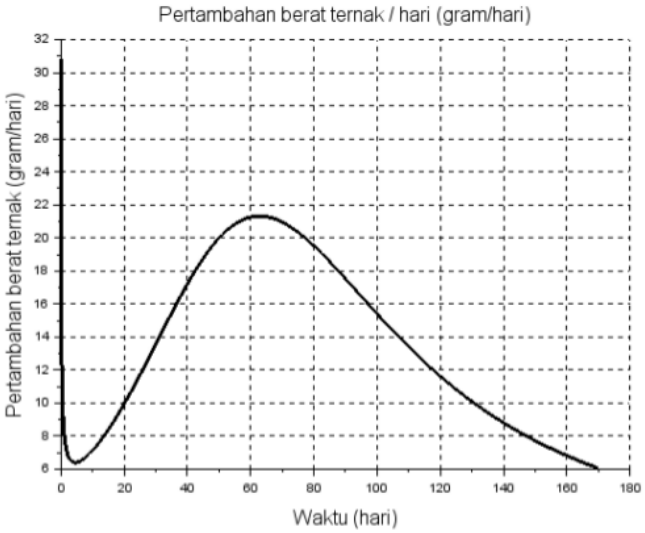
Tabel 10.2. Hasil estimasi parameter model pertumbuhan logistik orde fraksional

α	r	K	MSE	Waktu optimal pertumbuhan ayam (hari)
0.446365	0.267129	3624.28	626.96	62.6
0.439591	0.271681	3680.90	623.21	62.8
0.437592	0.273166	3695.14	623.24	62.8
0.483806	0.241650	3377.78	716.41	62.4
0.412298	0.292120	3901.90	658.36	63.0

Variasi hasil estimasi parameter model pertumbuhan logistik orde fraksional pada Tabel 10.2 disebabkan metode optimisasi swarm parameter merupakan metode optimasi stokastik, dan menggunakan bilangan acak dalam implementasinya. Oleh karena itu, metode optimiasi swarm parameter perlu diimplementasikan beberapa kali untuk memperoleh hasil yang relatif lebih baik. Berdasarkan Tabel 10.2, parameter yang lebih baik pada model pertumbuhan logistik orde fraksional tersebut adalah $\alpha \approx 0.439591, \mathrm{r} \approx 0.271681, \mathrm{K} \approx 3680.90$ gram dengan nilai $MSE \approx 623.21$. Grafik berat ayam jantan dan pertambahan berat ayam perhari yang diperoleh dari model pertumbuhan logistik orde fraksional berturut-turut pada Gambar 10.4 dan Gambar 10.5.



Gambar 10.4. Grafik data berat ayam jantan (tanda bulat) dan prediksi berat ayam jantan yang diperoleh dari model pertumbuhan logistik orde fraksional



Gambar 10.5. Grafik pertambahan berat ayam jantan yang diperoleh dari model pertumbuhan logistik orde fraksional

Gambar 10.4 menunjukkan bahwa model pertumbuhan logistik dapat digunakan untuk mendeskripsikan dinamika pertumbuhan ayam jantan. Nilai MSE pada model logistik orde fraksional ($MSE \approx 623.21$) jauh lebih kecil dari nilai MSE pada model pertumbuhan logistik klasik ($MSE \approx 1887.46$). Hal ini mengindikasikan bahwa model pertumbuhan logistik orde fraksional lebih sesuai dari model pertumbuhan logistik dalam menggambarkan pertumbuhan ayam jantan. Gambar 10.5 mendeskripsikan grafik pertambahan berat ayam jantan per satuan waktu yang diperoleh dari model pertumbuhan logistik orde fraksional. Dengan mengabaikan laju pertumbuhan pada saat awal, laju pertumbuhan optimal ayam jantan terjadi pada sekitar hari ke-63, dengan tangkat pertumbuhan berat ayam sekitar 21 gram/hari. Setelah waktu tersebut, laju pertumbuhan berat ayam jantan menurun, dan pada hari ke-170 pertumbuhan berat ayam diperkirakan sekitar 6 gram/hari.

11. Estimasi Parameter Model Mangsa-Pemangsa

11.1. Model Matematika Mangsa-Pemangsa

Pada suatu ekosistem makhluk hidup, terdapat interaksi antar populasi. Interaksi antar populasi dapat berbentuk kompetisi (persaingan), predasi (pemangsaan) dan simbiosis. Pada peristiwa predasi, populasi predator (pemangsa) memakan populasi prey (mangsa) untuk memperoleh energi dan mempertahankan kelangsungan hidup populasi pemangsa. Beberapa peristiwa predasi di ekosistem darat adalah kucing atau ular memangsa tikus, elang memangsa ular, harimau atau singa memangsa rusa dan musang memangsa ayam. Pada ekosistem laut, ikan berukuran kecil memakan plankton, ikan berukuran sedang memangsa ikan berukuran kecil, dan ikan berukuran besar memangsa ikan berukuran sedang.

Predasi pemangsa terhadap mangsa akan menurunkan populasi mangsa. Ketika populasi mangsa menurun, maka populasi pemangsa lebih sulit menemukan dan memakan populasi mangsa. Hal ini dapat mengancam kelangsungan hidup populasi pemangsa, sehingga populasi pemangsa dapat menurun akibat penurunan populasi mangsa. Model matematika mangsa-pemangsa (*predator-prey mathematical model*) dapat digunakan untuk menjelaskan secara kualitatif dinamika populasi mangsa dan dinamika populasi pemangsa. Model matematika mangsa-pemangsa mula-mula dikembangkan secara independen oleh A J. Lotka pada tahun 1925 dan Vito Volterra pada tahun 1926. Model matematika mangsa-pemangsa yang dikembangkan oleh Lotka dan Volterra tersebut dikenal dengan model matematika mangsa-pemangsa Lotka-Volterra. Pada saat itu, Volterra membangun model mangsa-pemangsa untuk menjelaskan dinamika populasi ikan di laut Adriatic sebagai populasi mangsa dan populasi paus sebagai populasi pemangsa.

Misalkan $x(t)$ dan $y(t)$ berturut-turut menyatakan banyaknya populasi mangsa (kepadatan populasi mangsa persatuan luas) dan banyaknya populasi pemangsa (kepadatan populasi pemangsa persatuan luas) pada saat t. Model matematika mangsa-pemangsa Lotka-Volterra dikonstruksi berdasarkan asumsi berikut:

(1) Sumber makanan populasi mangsa sangat melimpah, sehingga populasi mangsa tumbuh mengikuti model pertumbuhan eksponensial ketika tidak ada populasi pemangsa.
(2) Ketika populasi mangsa tidak ada, populasi pemangsa turun dengan laju konstan.
(3) Laju pengurangan populasi mangsa dan laju pertambahan populasi pemangsa akibat predasi sebanding dengan $x(t)y(t)$.

Berdasarkan asumsi tersebut, model matematika mangsa-pemangsa Lotka-Volterra berbentuk

$$\frac{dx}{dt} = ax - bxy, \tag{11.1}$$

$$\frac{dy}{dt} = -cy + dxy. \tag{11.2}$$

Pada persamaan (11.1)-(11.2), parameter a, c berturut-turut adalah laju pertumbuhan populasi mangsa dan laju penurunan populasi pemangsa. Parameter b, d berturut-turut adalah laju pengurangan populasi mangsa dan laju pertambahan populasi pemangsa akibat predasi. Pada umumnya, nilai parameter d lebih kecil dari nilai parameter b.

Model matematika mangsa-pemangsa tipe Lotka-Volterra memiliki dua titik setimbang/titik tetap/solusi konstan/solusi kesetimbangan, yaitu $E_1(0,0)$ dan $E_2\left(\frac{c}{d}, \frac{a}{b}\right)$. Titik setimbang E_1 adalah titik setimbang kepunahan populasi mangsa dan populasi pemangsa, sedangkan titik setimbang E_2 disebut sebagai titik setimbang koeksistensi (populasi mangsa dan

populasi pemangsa ada pada suatu ekosistem). Dengan menggunakan linearisasi model, titik setimbang kepunahan populasi bersifat tidak stabil, artinya solusi model dengan nilai awal (x_0, y_0) disekitar $E_1(0,0)$ akan cenderung menjauh dari titik setimbang $E_1(0,0)$.

Titik setimbang koeksistensi E_2 bersifat stabil, artinya solusi model dengan nilai awal (x_0, y_0) disekitar $E_2\left(\frac{c}{d}, \frac{a}{b}\right)$ akan cenderung tetap berada di sekitar titik setimbang koeksistensi $E_2\left(\frac{c}{d}, \frac{a}{b}\right)$. Potret fase (*phase portrait*), yaitu plot antara $x(t)$ versus $y(t)$ untuk nilai awal (x_0, y_0) disekitar E_2 berupa kurva tertutup. Kurva tertutup tersebut merepresentasikan solusi periodik model mangsa-pemangsa tipe Lotka-Volterra dengan nilai awal (x_0, y_0) disekitar $E_2\left(\frac{c}{d}, \frac{a}{b}\right)$. Dapat ditunjukkan bahwa periode (T) solusi periodik tersebut dapat dihampiri dengan $T \approx \frac{2\pi}{ac}$ [37].

Pada bagian selanjutnya akan dibahas implementasi metode optimisasi swarm partikel untuk mengestimasi nilai parameter model matematika mangsa-pemangsa tipe Lotka-Volterra.

11.2. Estimasi Parameter Model Matematika Mangsa-Pemangsa

Pada bagian ini, metode optimisasi swarm partikel digunakan untuk mementukan parameter model matematika mangsa pemangsa tipe Lotka-Volterra. Data yang digunakan merupakan data bangkitan berupa kepadatan populasi mangsa $x(t)$ dan kepadatan populasi pemangsa $y(t)$ pada saat t dan disajikan pada Tabel 11.1.

Tabel 11.1. Data bangkitan kepadatan populasi mangsa dan pemangsa

t	$x(t)$	$y(t)$	t	$x(t)$	$y(t)$	t	$x(t)$	$y(t)$
0	250	30	7	314	75	14	126	42
1	305	31	8	240	77	15	141	37
2	365	34	9	185	74	16	163	33
3	420	39	10	149	68	17	195	31
4	453	47	11	129	61	18	237	30
5	444	57	12	120	54	19	289	30
6	391	68	13	119	47	20	349	33

Parameter a, b, c, d pada model matematika mangsa-pemangsa diestimasi sedemikian hingga rata-rata galat relative (*mean absolute percentage error*) MAPE

$$MAPE = \frac{1}{2n}\sum_{i=1}^{n}\left(\left|\frac{x_i - \hat{x}_i}{x_i}\right| + \left|\frac{y_i - \hat{y}_i}{y_i}\right|\right), \quad (11.3)$$

bernilai minimum. Pada persamaan (11.3), n menyatakan banyaknya data pengamatan dan $\hat{x}_i, \hat{y}_i$ berturut-turut kepadatan populasi mangsa dan kepadatan populasi pemangsa pada waktu ke-i yang diperoleh dari model matematika mangsa pemangsa tipe Lotka Volterra. Implementasi metode optimisasi swarm partikel menggunakan perangkat lunak Scilab untuk menentukan parameter model matematika mangsa-pemangsa disajikan pada subbab berikutnya.

11.2.1. Implementasi Program

Kode program yang digunakan untuk mengestimasi parameter model matematika mangsa pemangsa menggunakan metode optimisasi swarm partikel terdiri dari tiga file subprogram, yaitu:

(5) dpredator.sce (berupa suatu fungsi/*function*)
(6) MAPE_predator.sce (berupa suatu fungsi/*function*)
(7) osp_model_predator.sce (berupa suatu *script*)

Diasumsikan bahwa ketiga file kode program tersebut disimpan dalam folder D:\File_Scilab. Kode program 11.1, kode program 11.2 dan kode program 11.3 berurut-turut merepresentasikan fungsi dpredator.sce, MAPE_predator.sce dan *script* osp_model_predator.sce.

Kode program 11.1 (dpredator.sce)

```
//Kode program 10.1 (dpredator.sce)
function dy = dpredator(t, y, param)
    // Ekstrak parameter
    a = param(1);
    b = param(2);
    c = param(3);
    d = param(4);
    // Inisialisasi
    dy = zeros(2,1);
    dy(1) = a * y(1) - b * y(1)*y(2);
    dy(2) = -c * y(2) + d * y(1)*y(2);
endfunction
//Akhir kode program
```

File fungsi tersebut digunakan untuk mendefinisikan model matematika mangsa-pemangsa tipe Lotka-Volterra. File fungsi **dpredator.sce** mempunyai tiga input, yaitu variabel **t, y** dan **param.** Vektor **param** memuat parameter pada model mangsa-pemangsa tipe Lotka-Volterra, yaitu parameter a, b, c, dan parameter d.

Kode Program 11.2 (MAPE_predator.sce)

```
//Kode program 11.2. (MAPE_predator.sce)
function [MAPE, xhitung, yhitung, xx, yy] = MAPE_predator(posisi)
    global tdata xdata ydata ndata tt indeks_t dt;
    exec('dpredator.sce');

    // Inisialisasi
    nt = length(tt);
    z = zeros(nt, 2);
    xhitung = zeros(nt,1);
    yhitung = zeros(nt,1);
    z(1, :) = [xdata(1), ydata(1)];

    // Nilai awal
    y0 = [xdata(1); ydata(1)];
    t0 = 0;

    // Hitung solusi menggunakan metode Runge-Kutta
    zz = ode(y0,t0,tt,list(dpredator,posisi)).';

    // Ekstrak xx, xhitung, yy dan yhitung
    xx = zz(:,1);
    yy = zz(:,2);
```

```
    xhitung = zz(indeks_t, 1);
    yhitung = zz(indeks_t, 2);
    SS1 = sum(abs(xdata - xhitung)./ xdata);
    SS2 = sum(abs(ydata - yhitung)./ ydata);
    MAPE = (SS1 + SS2)/(2 * ndata);
endfunction
//Akhir kode program
```

File fungsi tersebut digunakan untuk menghitung nilai MAPE (rata-rata galat relatif) dari setiap solusi model mangsa-pemangsa tipe Lotka-Volterra. File fungsi **MAPE_predator.sce** mempunyai satu input, yaitu variabel **posisi** yang berbentuk vector yang menyimpan nilai parameter pada model mangsa-pemangsa, yaitu parameter a, b, c, d.

Kode Program 11.3 (osp_model_predator.sce)

```
//Kode program 11.3. (osp_model_predator.sce)
clc;
clear;
xdel(winsid()); // Close all figure

tic( );

// Variabel global
global tdata xdata ydata ndata tt indeks_t dt;

// tdata dan xdata
tdata = (0:20).';
xdata = [250, 305, 365, 420, 453, 444, 391, 314, 240, 185, 149, 129, ...
        120, 119, 126, 141, 163, 195, 237, 289, 349].';
ydata = [30, 31, 34, 39, 47, 57, 68, 75, 77, 74, 68, 61, 54, 47, 42, ...
         37, 33, 31, 30, 30, 33].';
ndata = length(tdata);
dt = 1e-1;
tt = (tdata(1):dt:tdata(ndata)).';
nt = length(tt);
indeks_t = zeros(ndata,1);
mm = 1;
for kk = 1:nt
   if (tt(kk) == tdata(mm))
      indeks_t(mm,:) = kk;
      mm = mm + 1;
   end
end // for kk = 1:nt

// nVar, VarMin, VarMax
nVar = 4;                 // Banyaknya variabel keputusan
VarMin = [0.1, 0, 0.1, 0];      // Nilai minimum variabel keputusan
```

```
VarMax = [1, 0.02, 1, 0.02];      // Nilai maksimum variabel keputusan

// Membuka file function
exec('MAPE_predator.sce',-1);

// Random seed generator
n = sum(clock( ));
rand("seed",n);

// Parameters OSP
MaxIt = 200;    // Maksimum banyaknya iterasi
nPop = 25;       // Ukuran populasi (ukuran swarm)
w = 0.5;          // Bobot inersia
c1 = 2.0;       // Koefisien personal
c2 = 2.0;       // Koefisien global

// Batas nilai kecepatan partikel
VelMax = 0.1*(VarMax - VarMin);
VelMin = - VelMax;

// Inisialisasi
posisi = rand(nPop, nVar);
kecepatan = zeros(nPop, nVar);
biaya  = zeros(nPop, 1);
pbest  = zeros(nPop, nVar);
biaya_pbest = zeros(nPop, 1);
gbest = zeros(nPop, 1);
biaya_gbest = zeros(1,1);

// Inisialisasi
for i = 1:nPop
   // Inisialisasi posisi partikel
   posisi(i, :) = VarMin + (VarMax - VarMin) .* rand(1,nVar);

   // Hitung nilai fungsi tujuan untuk masing-masing posisi
   [biaya(i), xhitung,yhitung, xx, yy] = MAPE_predator(posisi(i,:));

   // Update partikel terbaik lokal
   pbest(i,:) = posisi(i,:);
   biaya_pbest(i) = biaya(i);

   // Update partikel terbaik global
   if (i == 1)
      gbest = posisi(i,:);
      biaya_gbest = biaya(i);
```

```
    else
        if (biaya(i) < biaya_gbest)
            gbest = posisi(i,:);
            biaya_gbest = biaya(i);
        end
    end
end

Best_cost = zeros(MaxIt,1);
va = zeros(MaxIt,1);
vc = zeros(MaxIt,1);
// Loop utama OSP
for it = 1:MaxIt
    disp('Iterasi = ' + string(it));
    for i=1:nPop

        // Update kecepatan partikel
        kecepatan(i,:) = w*kecepatan(i,:) + c1 * rand(1,nVar) .* (pbest(i,:) - ...
                         posisi(i,:)) + c2 * rand(1,nVar) .* (gbest - posisi(i,:));

        // Terapkan nilai batas kecepatan partikel
        kecepatan(i,:) = max(kecepatan(i,:), VelMin);
        kecepatan(i,:) = min(kecepatan(i,:), VelMax);

        // Update posisi partikel
        posisi(i,:) = posisi(i,:) + kecepatan(i,:);

        // Terapkan nilai batas posisi partikel
        posisi(i,:) = max(posisi(i,:), VarMin);
        posisi(i,:) = min(posisi(i,:), VarMax);

        // Hitung nilai fungsi tujuan untuk masing-masing posisi
        [biaya(i), xhitung,yhitung, xx, yy] = MAPE_predator(posisi(i,:));

        // Update posisi partikel terbaik lokal
        if (biaya(i) < biaya_pbest(i))
            pbest(i,:) = posisi(i,:);
            biaya_pbest(i) = biaya(i);
        // Update posisi partikel terbaik global
            if (biaya(i) < biaya_gbest)
                gbest = posisi(i,:);
                biaya_gbest = biaya(i);
                xhitung_terbaik = xhitung;
                xx_terbaik = xx;
                yhitung_terbaik = yhitung;
                yy_terbaik = yy;
```

```
            end
        end // if (biaya(i) > biaya_pbest(i))
    end // for i = 1:nPop

    BestCost(it) = biaya_gbest;
    va(it) = gbest(1);
    vc(it) = gbest(3);

    disp('Iterasi ' + string(it) + ', a = ' + string(gbest(1)) + ...
    ', b = ' + string(gbest(2)) +  ', c = ' + string(gbest(3)) + ...
    ', d = ' + string(gbest(4)) + ', MAPE = ' + string(BestCost(it)));
end

// Grafik nilai fungsi tujuan
scf(0);
xn = (1:1:MaxIt).';
plot2d('nl',xn,BestCost,style = [1]);
title('Nilai rata-rata galat relatif (MAPE)',"fontsize",3.5);
xlabel('Iterasi',"fontsize",3.5);
ylabel('Nilai MAPE',"fontsize",3.5);
gca().grid=[1 1 1]; // Setara dengan grid on;
gca().children(1).children(1).thickness = 3; // Set ketebalan garis kurva
axes_handle.grid = [1 1 1]; // Setara dengan grid(axes_handle,"on")

scf(1);
plot(xdata,ydata,'ko',"linewidth",3);
set(gca(),"auto_clear","off"); // hold on
plot(xx_terbaik,yy_terbaik,'k',"linewidth",3);
hl=legend(['data';'model'],[1]);
gca.grid=[1 1 1]; // Setara dengan grid on;
gca().data_bounds=[0.99*min(xdata),0.99*min(ydata);...
                   1.01*max(xdata),1.01*max(ydata)];
title('Bidang fase kepadatan predator-prey',"fontsize",3.5);
xlabel('Kepadatan populasi prey (individu/satuan luas)',"fontsize",3.5);
ylabel('Kepadatan populasi predator (individu/satuan luas)',"fontsize",3.5);

waktu = toc( );
hasil_OSP = [gbest, biaya_gbest];

disp(waktu)
// Akhir kode program osp_model_predator.sce
```

Simpanlah kode program 11.3 tersebut dengan nama **osp_model_predator.sce**.

Berikut disajikan beberapa penjelasan yang berhubungan dengan kode program 11.3 tersebut:

(h) Blok perintah

```
nVar = 4;                  // Banyaknya variabel keputusan
VarMin = [0.1, 0, 0.1, 0];     // Nilai minimum variabel keputusan
VarMax = [1, 0.02, 1, 0.02];     // Nilai maksimum variabel keputusan
```

berhubungan dengan banyaknya variabel keputusan (banyaknya parameter yang diestimasi) pada model matematika mangsa-pemangsa tipe Lotka-Volterra, yaitu parameter a, b, c, d. Nilai parameter r, K dan t_{inf} diduga terletak pada interval tutup [0.1, 1], [0, 0.02], [0.1, 1] dan [0, 0.02]. Akibatnya, pada blok perintah tersebut, didefinisikan **nVar = 4**, **VarMin = [0.1, 0, 0.1, 0]**, dan **VarMax = [1, 0.02, 1, 0.02]**.

(i) Perintah **exec('MAPE_predator.sce', -1)** digunakan untuk mengaktifkan file script/function **MAPE_predator.sce**.

11.2.2. Menjalankan Program

Prosedur berikut dapat digunakan untuk menjalankan untuk menjalankan file kode program **osp_model_predator.sce** yang tersimpan dalam folder D:\File_Scilab:

(q) Bukalah perangkat lunak Scilab.

(r) Aktifkan folder D:\File_Scilab melalui klik menu *File – Current working directory – Browse for new*. Pada kotak dialog *Select directory*, pilih folder **D:\Program_Scilab**. Selanjutnya klik tombol *Open*.

(s) Selanjutnya klik menu *File – Open a file*. Pada kotak dialog Select a file to open, klik file **osp_model_predator.sce**. Selanjutnya klik tombol *Open*.

(t) Pada jendela **osp_model_predator.sce**, klik menu *Execute – save and execute*.

Berikut disajikan contoh tampilan hasil eksekusi kode program osp_p93.sce.

```
Iterasi = 200
 Iterasi 200, a = 0.5259585, b = 0.0107801, c = 0.2370629, d = 0.0009479,
 MAPE = 0.0116134
 167.32974
```

Setelah 200 iterasi, diperoleh nilai parameter $a \approx 0.5259585, b \approx 0.0107801, c \approx 0.2370629, d \approx 0.0009479$, dengan nilai $MAPE \approx 0.0116134$. Waktu eksekusi kode program sekitar 167 detik. Untuk memperoleh hasil yang lebih akurat, metode optimisasi swarm partikel diimplementasikan sebanyak 5 kali dan ditentukan parameter terbaik dari 5 kali implementasi metode tersebut. Hasil implementasi metode optimisasi swarm partikel dengan koefisien inersia $w = 0.5$ disajikan pada Tabel 11.2. Kode program yang digunakan untuk pengulangan metode optisasi swarm partikel disajikan pada Lampiran 3.

Tabel 11.2. Hasil estimasi parameter model mangsa-pemangsa

Percobaan	*a*	*b*	*c*	*d*	MAPE
1	0.49258	0.00982	0.25365	0.00102	0.00516
2	0.49624	0.00990	0.25188	0.00101	0.00411
3	0.73920	0.01974	0.64370	0.00317	0.29617
4	0.73372	0.01957	0.64711	0.00317	0.29640
5	0.48180	0.00951	0.25961	0.00104	0.00877

Tabel 11.2 menunjukkan variasi hasil estimasi parameter model mangsa-pemangsa yang diperoleh dari implementasi metode optimisasi swarm partikel. Hal ini disebabkan metode optimisasi swarm partikel adalah metode yang berbasis tiruan bilangan acak (*pseudorandom*). Hasil terbaik

diperoleh dari percobaan kedua, dengan nilai MAPE = 0.00411. Untuk memperoleh hasil estimasi yang lebih baik, metode optimisasi swarm partikel dapat diimplementasikan dengan memperkecil rentang nilai parameter model. Dengan memanfaatkan titik setimbang koeksistensi E_2, solusi model matematika mangsa-pemangsa tipe Lotka-Volterra dipengaruhi oleh perbandingan parameter $\frac{c}{d}$ dan $\frac{a}{b}$. Oleh karena itu, dengan memilih parameter a dan parameter c dalam rentang yang tepat, dapat diperoleh hasil estimasi parameter dengan galat yang lebih kecil. Dari percobaan pertama, percobaan kedua dan percobaan kelima, dapat dipilih parameter a dan parameter c dalam interval $0.45 \leq a \leq 0.55$ dan $0.2 \leq c \leq 0.3$.

Pendefinisian rentang parameter model pada kode pada Lampiran 3 dilakukan dengan memodifikasi nilai variabel VarMin dan VarMax menjadi bentuk

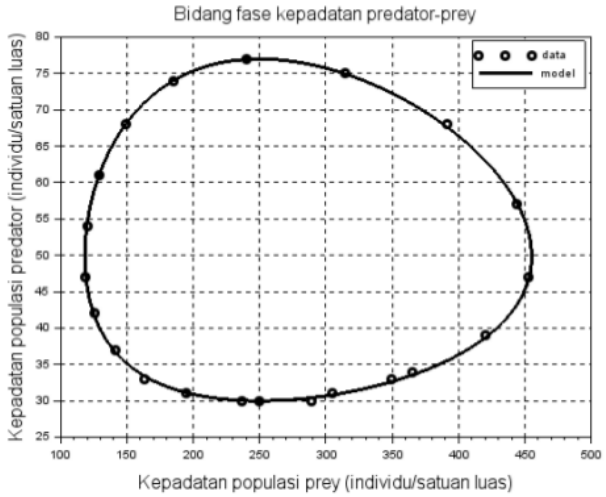
```
VarMin = [0.45, 0.00, 0.2, 0.00];      // Nilai minimum variabel keputusan
VarMax = [0.55, 0.02, 0.3, 0.02];      // Nilai maksimum variabel keputusan
```

Hasil implementasi metode optimisasi swarm partikel pada Lampiran 3 dengan nilai parameter a dan parameter c dalam rentang $0.45 \leq a \leq 0.55,\ 0.2 \leq c \leq 0.3$ disajikan pada Tabel 11.3.

Tabel 11.3. Nilai parameter model mangsa-pemangsa

Percobaan	a	b	c	d	MAPE
1	0.49847	0.00997	0.25070	0.00100	0.00363
2	0.49949	0.00999	0.25023	0.00100	0.00347
3	0.50244	0.01008	0.24878	0.00100	0.00325
4	0.50224	0.01007	0.24887	0.00100	0.00324
5	0.50231	0.01007	0.24884	0.00100	0.00325

Berdasarkan Tabel 11.3, parameter terbaik pada model matematika mangsa-pemangsa tipe Lotka-Volterra yang diperoleh dari optimisasi swarm partikel adalah $a = 0.50224, b = 0.01007, c = 0.24887, d = 0.00100$ dengan nilai MAPE = 0.00324. Gambar 11.1 menyajikan grafik bidang fase kepadatan populasi mangsa – kepadatan populasi pemangsa.



Gambar 11.1. Bidang fase kepadatan populasi mangsa – kepadatan populasi pemangsa

Bidang fase kepadatan populasi mangsa – kepadatan populasi pemangsa pada Gambar 11.1 berbentuk kurva tertutup. Hal ini bersesuaian dengan solusi model matematika mangsa-pemangsa tipe Lotka-Volterra berbentuk fungsi periodik. Pada bagian selanjutnya akan dibahas implementasi metode optimisasi swarm partikel untuk mengestimasi nilai parameter model matematika mangsa-pemangsa tipe Lotka-Volterra orde fraksional.

11.3. Estimasi Parameter Model Matematika Mangsa-Pemangsa Orde Fraksional

Misalkan $x(t)$ dan $y(t)$ berturut-turut adalah kepadatan populasi mangsa dan kepadatan populasi pemangsa pada saat t. Model matematika mangsa-pemangsa Lotka-Volterra orde fraksional dengan turunan fraksional Caputo diberikan oleh

$$\frac{d^{\alpha}x}{dt^{\alpha}} = ax - bxy, \tag{11.3}$$

$$\frac{d^{\alpha}y}{dt^{\alpha}} = -cy + dxy, \tag{11.4}$$

dengan $0 < \alpha \leq 1$. Untuk $\alpha = 1$ model matematika mangsa-pemangsa orde fraksional dapat disederhanakan menjadi model matematika mangsa-pemangsa klasik (orde turunan sama dengan satu). Seperti halnya model matematika mangka-pemangsa tipe Lotka-Volterra klasik, model matematika mangsa-pemangsa tipe Lotka-Volterra orde fraksional memiliki dua titik setimbang yaitu titik setimbang kepunahan populasi $E_1(0,0)$ dan titik setimbang koeksistensi $E_2\left(\frac{c}{d},\frac{a}{b}\right)$. Dengan menggunakan linearisasi model, titik setimbang kepunahan populasi bersifat tidak stabil, artinya solusi model dengan nilai awal (x_0, y_0) disekitar $E_1(0,0)$ akan cenderung menjauh dari titik setimbang $E_1(0,0)$. Untuk nilai parameter $0 < \alpha < 1$, titik setimbang koeksistensi E_2 bersifat stabil asimtotis, artinya solusi model dengan nilai awal (x_0, y_0) disekitar $E_2\left(\frac{c}{d},\frac{a}{b}\right)$ menuju ke titik setimbang koeksistensi $E_2\left(\frac{c}{d},\frac{a}{b}\right)$ untuk waktu yang cukup panjang [38].

Pada bagian ini, metode optimisasi swarm partikel digunakan untuk mementukan parameter model matematika mangsa pemangsa tipe Lotka-Volterra orde fraksional. Data yang digunakan merupakan data bangkitan berupa kepadatan populasi mangsa $x(t)$ dan kepadatan populasi pemangsa $y(t)$ pada saat t dan disajikan pada Tabel 11.4.

Tabel 11.4 Data bangkitan kepadatan populasi mangsa dan pemangsa

t	$x(t)$	$y(t)$	t	$x(t)$	$y(t)$
0	100	50	35	240	40
5	101	34	40	219	44
10	121	28	45	200	44
15	153	25	50	190	42
20	192	25	55	186	40
25	228	29	60	188	39
30	246	34	65	193	38

Seperti pada sub bab sebelumnya, parameter α, a, b, c, d pada model matematika mangsa-pemangsa tipe Lotka-Voltera orde fraksional diestimasi sedemikian hingga rata-rata galat relative (*mean absolute percentage error*) MAPE bernilai minimum. Implementasi metode optimisasi swarm partikel menggunakan perangkat lunak Scilab untuk menentukan parameter model matematika mangsa-pemangsa disajikan pada subbab berikutnya.

11.3.1. Implementasi Program

Kode program yang digunakan untuk mengestimasi parameter model matematika mangsa-pemangsa tipe Lotka-Volterra orde fraksional menggunakan metode optimisasi swarm partikel terdiri dari empat file subprogram, yaitu:

(1) fpredator.sce (berupa suatu fungsi/*function*)
(2) pk_fpredator.sce (berupa suatu fungsi/*function*)
(3) MAPE_predator_fraksional.sce (berupa suatu fungsi/*function*)
(4) osp_model_predator.sce (berupa suatu *script*)

Diasumsikan bahwa ketiga file kode program tersebut disimpan dalam folder D:\File_Scilab. Kode program 11.4 – kode program 11.7 berurut-turut merepresentasikan fungsi fpredator.sce, MAPE_predator.sce dan *script* osp_model_predator.sce.

Kode program 11.4 (fpredator.sce)

```
//Kode program 11.4 (fpredator.sce)
function fy = fpredator(t, y, param)
    // Ekstrak parameter
    alpha = param(1);
    a = param(2);
    b = param(3);
    c = param(4);
    d = param(5);
    // Inisialisasi
    [nbar, nkol] = size(t);
    fy = zeros(nbar, 2);
    fy(:, 1) = a * y(:, 1) - b * y(:, 1) .* y(:, 2);
    fy(:, 2) = -c * y(:, 2) + d * y(:, 1) .* y(:, 2);
endfunction
//Akhir kode program
```

File fungsi tersebut digunakan untuk mendefinisikan model matematika mangsa-pemangsa tipe Lotka-Volterra orde fraksional. File fungsi **fredator.sce** mempunyai tiga input, yaitu variabel **t, y** dan **param.** Vektor **param** memuat parameter pada model mangsa-pemangsa tipe Lotka-Volterra, yaitu parameter α, a, b, c, dan parameter d.

Kode Program 11.5 (pk_fpredator.sce)

```
//Kode program 11.5. (pk_fpredator.sce)
function [tt, yk] = pk_fpredator(t0, tf, dt, y0, param)
    // Ekstrak parameter
    alpha = param(1);
    a = param(2);
    b = param(3);
    c = param(4);
    d = param(5);

    // Inisialisasi
    tt = (t0:dt:tf).';
    nt = length(tt);
```

```
    yp = zeros(nt,2);
    yk = zeros(nt,2);
    yp(1,:) = y0;
    yk(1,:) = y0;
    [nbar, nkol] = size(y0);
    exec('fpredator.sce',-1);
    // Hitung solusi model
    for k = 2:nt
        // Hitung prediktor
        vb = (k - (1:k-1)).^alpha - (k-1 - (1:k-1)).^alpha;
        // vb adalah vektor baris ukuran 1 x (k - 1)
        fyp = fpredator(tt(1:k-1), yp(1:k-1,:), param);
        // fyp adalah vektor berukuran (k - 1) x banyaknya kolom yp
        yp(k, :) = y0 + dt^alpha/gamma(alpha + 1) * vb * fyp;

        // Hitung korektor
        va = (k+1 - (1:k)).^(alpha+1) - 2 * (k - (1:k)).^(alpha+1) + ...
             (k-1 - (1:k)).^(alpha+1);
        va(k) = 1;
        va(1) = (k - 2)^(alpha+1) - (k -1)^(alpha + 1) + (alpha + 1)*(k - 1)^alpha;
        // va adalah vektor baris berukuran 1 x k
        fyk =   fpredator(tt(1:k), yp(1:k,:), param);
        // fyk berukuran k x banyaknya kolom yk
        yk(k,:) = y0 +  dt^alpha/gamma(alpha + 2) * va * fyk;
        yp = yk;
    end // for k = 2:nt
    yp = yk;
endfunction
//Akhir kode program
```

Kode program pk_fpredator.sce digunakan untuk menentukan solusi model matematika mangsa-pemangsa tipe Lotka-Volterra orde fraksional menggunakan metode prediktor-korektor.

Kode Program 11.6 (MAPE_predator_fraksional.sce)

```
//Kode program 11.6. (MAPE_predator_fraksional.sce)
function [MAPE, xhitung, yhitung, xx, yy] =MAPE_predator_fraksional(param)
    // Variabel global
    global tdata xdata ydata ndata tt indeks_t dt y0;

    // Mengaktifkan function pk_fpredator.sce
    exec('pk_fpredator.sce');

    // Hitung solusi menggunakan metode prediktor-korektor
    ndata = length(tdata);
    t0 = tdata(1);
    tf = tdata(ndata);
```

```
    [tt,zz] = pk_fpredator(t0,tf, dt, y0, param);

  // Ekstrak xx, xhitung, yy dan yhitung
  xx = zz(:,1);
  yy = zz(:,2);
  xhitung = zz(indeks_t,1);
  yhitung = zz(indeks_t,2);
  SS1 = sum(abs(xdata - xhitung)./ xdata);
  SS2 = sum(abs(ydata - yhitung)./ ydata);
  MAPE = (SS1 + SS2)/(2 * ndata);
endfunction
//Akhir kode program
```

File fungsi tersebut digunakan untuk menghitung nilai MAPE (rata-rata galat relatif) dari setiap solusi model mangsa-pemangsa tipe Lotka-Volterra orde fraksional. File fungsi **MAPE_predator_fraksional.sce** mempunyai satu input, yaitu variabel **param** yang berbentuk vector yang menyimpan nilai parameter pada model mangsa-pemangsa, yaitu parameter α, a, b, c, d.

Kode Program 11.7 (osp_predator_fraksional.sce)

```
//Kode program 11.7. (osp_predator_fraksional.sce)
clc;
clear;
xdel(winsid()); // Close all figure
tic( );

// Variabel global
global tdata xdata ydata ndata tt indeks_t dt y0;

// tdata dan xdata
tdata = [0, 5, 10, 15, 20, 25, 30,35, 40, 45, 50, 55, 60, 65].';
xdata = [100, 101, 121, 153, 192, 228, 246, 240, 219, 200, 190, ...
      186, 188, 193].';
ydata = [50, 34, 28, 25, 25, 29, 34, 40, 44, 44, 42, 40, 39, 38].';
ndata = length(tdata);
y0 = [xdata(1), ydata(1)];
dt = 1e-1;
tt = (tdata(1):dt:tdata(ndata)).';
nt = length(tt);
indeks_t = zeros(ndata,1);
mm = 1;
for kk = 1:nt
  if (tt(kk) == tdata(mm))
    indeks_t(mm,:) = kk;
    mm = mm + 1;
  end
```

```
end // for kk = 1:nt

// nVar, VarMin, VarMax
nVar = 5;                  // Banyaknya variabel keputusan
VarMin = [0.5, 0,   0, 0, 0];      // Nilai minimum variabel keputusan
VarMax = [1.0, 0.5, 0.05, 0.5, 0.05];      // Nilai maksimum variabel keputusan

// Membuka file function
exec('MAPE_predator_fraksional.sce',-1);

// Parameters OSP
MaxIt = 100;    // Maksimum banyaknya iterasi
nPop = 20;        // Ukuran populasi (ukuran swarm)
w = 0.5;           // Bobot inersia
c1 = 2.0;       // Koefisien personal
c2 = 2.0;       // Koefisien global

// Batas nilai kecepatan partikel
VelMax = 0.1*(VarMax - VarMin);
VelMin = - VelMax;

// Inisialisasi
posisi = rand(nPop, nVar);
kecepatan = zeros(nPop, nVar);
biaya  = zeros(nPop, 1);
pbest  = zeros(nPop, nVar);
biaya_pbest = zeros(nPop, 1);
gbest = zeros(nPop, 1);
biaya_gbest = zeros(1,1);

// Random number generator
nn = sum(clock( ));
rand("seed",nn);

// Inisialisasi
for i = 1:nPop
   // Inisialisasi posisi partikel
   posisi(i, :) = VarMin + (VarMax - VarMin) .* rand(1,nVar);

   // Hitung nilai fungsi tujuan untuk masing-masing posisi
   [biaya(i), xhitung,yhitung, xx, yy] = MAPE_predator_fraksional(posisi(i,:));

   // Update partikel terbaik lokal
   pbest(i,:) = posisi(i,:);
   biaya_pbest(i) = biaya(i);
```

```
    // Update partikel terbaik global
    if (i == 1)
        gbest = posisi(i,:);
        biaya_gbest = biaya(i);
        xx_terbaik = xx;
        yy_terbaik = yy;
    else
        if (biaya(i) < biaya_gbest)
            gbest = posisi(i,:);
            biaya_gbest = biaya(i);
            xx_terbaik = xx;
            yy_terbaik = yy;
        end
    end
end

Best31st = zeros(MaxIt,1);
va = zeros(MaxIt,1);
vc = zeros(MaxIt,1);
// Loop utama OSP
for it = 1:MaxIt
    disp('Iterasi = ' + string(it));
    for i=1:nPop
        disp('partikel ke ' + string(i));
        // Update kecepatan partikel33
        kecepatan(i,:) = w*kecepatan(i,:) + c1 * rand(1,nVar) .* (pbest(i,:) - ...
                         posisi(i,:)) + c2 * rand(1,nVar) .* (gbest - posisi(i,:));

        // Terapkan nilai batas kecepatan partikel
        kecepatan(i,:) = max(kecepatan(i,:), VelMin);
        kecepatan(i,:) = min(kecepatan(i,:), VelMax);

        // Update posisi partikel
        posisi(i,:) = posisi(i,:) + kecepatan(i,:);

        // Terapkan nilai batas posisi partikel
        posisi(i,:) = max(posisi(i,:), VarMin);
        posisi(i,:) = min(posisi(i,:), VarMax);

        // Hitung nilai fungsi tujuan untuk masing-masing posisi
        [biaya(i), xhitung,yhitung, xx, yy] = MAPE_predator_fraksional(posisi(i,:));

        // Update posisi partikel terbaik lokal
        if (biaya(i) < biaya_pbest(i))
            pbest(i,:) = posisi(i,:);
```

```
            biaya_pbest(i) = biaya(i);
        // Update posisi partikel terbaik global
            if (biaya(i) < biaya_gbest)
                gbest = posisi(i,:);
                biaya_gbest = biaya(i);
                xhitung_terbaik = xhitung;
                xx_terbaik = xx;
                yhitung_terbaik = yhitung;
                yy_terbaik = yy;
            end
        end // if (biaya(i) > biaya_pbest(i))
    end // for i = 1:nPop

    BestCost(it) = biaya_gbest;

    disp('Iterasi ' + string(it) + ', alpha = ' + string(gbest(1)) + ...
    ', a = ' + string(gbest(2)) + ', b = ' + string(gbest(3)) + ...
    ', c = ' + string(gbest(4)) + ...
    ', d = ' + string(gbest(5)) + ', MAPE = ' + string(BestCost(it)));
end

// Grafik nilai fungsi tujuan
scf(0);
xn = (1:1:MaxIt).';
plot2d('nl',xn,BestCost,style = [1]);
title('Nilai rata-rata galat relatif (MAPE)',"fontsize",3.5);
xlabel('Iterasi',"fontsize",3.5);
ylabel('Nilai MAPE',"fontsize",3.5);
gca().grid=[1 1 1]; // Setara dengan grid on;
gca().children(1).children(1).thickness = 3; // Set ketebalan garis kurva
axes_handle.grid = [1 1 1]; // Setara dengan grid(axes_handle,"on")

scf(1);
plot(xdata,ydata,'ko',"linewidth",3);
set(gca(),"auto_clear","off"); // hold on
plot(xx_terbaik,yy_terbaik,'k',"linewidth",3);
hl=legend(['data';'model'],[1]);
gca.grid=[1 1 1]; // Setara dengan grid on;
gca().data_bounds=[0.99*min(xdata),0.99*min(ydata);...
                   1.01*max(xdata),1.01*max(ydata)];
title('Bidang fase kepadatan predator-prey',"fontsize",3.5);
xlabel('Kepadatan populasi prey (individu/satuan luas)',"fontsize",3.5);
ylabel('Kepadatan populasi predator (individu/satuan luas)',"fontsize",3.5);

waktu = toc( );
```

```
hasil_OSP = [gbest, biaya_gbest];

disp('waktu = ' + string(waktu) + ' detik.');
// Akhir kode program osp_predator_fraksional.sce
```

Simpanlah kode program 11.7 tersebut dengan nama **osp_predator_fraksional.sce**.

Berikut disajikan beberapa penjelasan yang berhubungan dengan kode program 11.3 tersebut:

(a) Blok perintah

```
nVar = 5;                 // Banyaknya variabel keputusan
VarMin = [0.5, 0,   0, 0, 0];      // Nilai minimum variabel keputusan
VarMax = [1.0, 0.5, 0.05, 0.5, 0.05]; // Nilai maksimum variabel keputusan
```

berhubungan dengan banyaknya variabel keputusan (banyaknya parameter yang diestimasi) pada model matematika mangsa-pemangsa tipe Lotka-Volterra, yaitu parameter α, a, b, c, d. Nilai parameter α, a, b, c, d diduga terletak pada interval tutup [0.5, 1], [0, 0.5], [0, 0.05], [0, 0.5], dan [0, 0.05]. Akibatnya, pada blok perintah tersebut, didefinisikan `nVar = 5`, `VarMin = [0.5, 0, 0, 0, 0]`, dan `VarMax = [1.0, 0.5, 0.05, 0.5, 0.05]`.

(b) Perintah **exec('MAPE_predator_fraksional.sce', -1)** digunakan untuk mengaktifkan file script/function **MAPE_predator_fraksional.sce**.

11.3.2. Menjalankan Program

Prosedur berikut dapat digunakan untuk menjalankan untuk menjalankan file kode program **osp_model_predator_fraksional.sce** yang tersimpan dalam folder D:\File_Scilab:

(a) Bukalah perangkat lunak Scilab.
(b) Aktifkan folder D:\File_Scilab melalui klik menu *File – Current working directory – Browse for new*. Pada kotak dialog *Select directory*, pilih folder **D:\Program_Scilab**. Selanjutnya klik tombol *Open*.
(c) Selanjutnya klik menu *File – Open a file*. Pada kotak dialog Select a file to open, klik file **osp_predator_fraksional.sce**. Selanjutnya klik tombol *Open*.
(d) Pada jendela **osp_predator_fraksional.sce**, klik menu *Execute – save and execute*.

Berikut disajikan contoh tampilan hasil eksekusi kode program osp_predator_fraksional.sce.

```
Iterasi 100, alpha = 0.7529607, a = 0.3897441, b = 0.0099389, c = 0.1480006,
d = 0.0007116, MAPE = 0.060927
 waktu = 2111.4251 detik.
```

Setelah 200 iterasi, diperoleh nilai parameter $\alpha \approx 0.7529607, a \approx 0.3987441, b \approx 0.0099389, c \approx 0.1480006, d \approx 0.0007116$, dengan nilai $MAPE \approx 0.060927$. Waktu eksekusi kode program sekitar 2112 detik. Untuk memperoleh hasil yang lebih akurat, metode optimisasi swarm partikel diimplementasikan sebanyak 5 kali dan ditentukan parameter terbaik dari 5 kali implementasi metode tersebut. Hasil implementasi metode optimisasi swarm partikel dengan koefisien inersia $w = 0.5$ disajikan pada Tabel 11.5.

Tabel 11.5. Hasil estimasi parameter model mangsa-pemangsa orde fraksional

Percobaan	α	a	b	c	d	MAPE
1	0.752961	0.389744	0.009939	0.148001	0.000712	0.060927
2	0.731227	0.241749	0.006308	0.260199	0.001295	0.045576
3	0.782404	0.231518	0.005933	0.199760	0.000985	0.019872
4	0.500000	0.500000	0.012662	0.500000	0.002377	0.115004
5	0.504585	0.500000	0.012715	0.500000	0.002380	0.115083

Tabel 11.2 menunjukkan variasi hasil estimasi parameter model mangsa-pemangsa yang diperoleh dari implementasi metode optimisasi swarm partikel. Hal ini disebabkan metode optimisasi swarm partikel adalah metode yang berbasis bilangan acak (pseudorandom). Hasil terbaik diperoleh dari percobaan kedua, dengan nilai MAPE = 0.019872. Untuk memperoleh hasil estimasi yang lebih baik, metode optimisasi swarm partikel dapat diimplementasikan dengan memperkecil rentang nilai parameter model. Dengan memanfaatkan titik setimbang koeksistensi E_2, solusi model matematika mangsa-pemangsa tipe Lotka-Volterra dipengaruhi oleh perbandingan parameter $\frac{c}{d}$ dan $\frac{a}{b}$. Selain itu, dinamika kepadatan populasi mangsa-pemangsa juga dipengaruhi oleh parameter α. Oleh karena itu, dengan memilih parameter a dan parameter c dalam rentang yang tepat, dapat diperoleh hasil estimasi parameter dengan galat yang lebih kecil. Dari percobaan kedua dan percobaan ketiga, dipilih parameter α, a dan parameter c dalam interval $0.7 \leq \alpha \leq 1, 0.1 \leq a \leq 0.3$ dan $0.1 \leq c \leq 0.3$.

Pendefinisian rentang parameter model pada Kode Program 11.7 dilakukan dengan memodifikasi nilai variabel VarMin dan VarMax menjadi bentuk

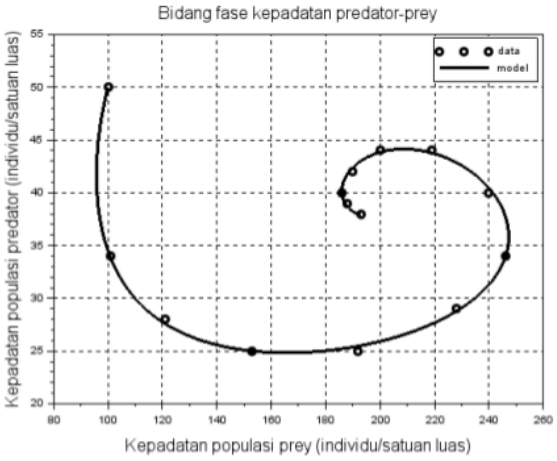
```
VarMin = [0.7, 0.1, 0, 0.1, 0];      // Nilai minimum variabel keputusan
VarMax = [1.0, 0.3, 0.05, 0.3, 0.05];      // Nilai maksimum variabel keputusan
```

Hasil implementasi metode optimisasi swarm partikel dengan nilai parameter modifikasi rentang nilai parameter α, a dan parameter c disajikan pada Tabel 11.6.

Tabel 11.6. Nilai parameter model orde fraksional setelah modifikasi rentang nilai parameter

Percobaan	α	a	b	c	d	MAPE
1	0.801326	0.188483	0.004686	0.206596	0.001035	0.006446
2	0.802910	0.221856	0.005566	0.178610	0.000889	0.012971
3	0.799963	0.217967	0.005512	0.187934	0.000934	0.010814
4	0.806083	0.200814	0.004999	0.190509	0.000953	0.007536
5	0.802008	0.199208	0.004980	0.199110	0.000996	0.004150

Berdasarkan Tabel 11.6, parameter terbaik pada model matematika mangsa-pemangsa tipe Lotka-Volterra orde fraksional yang diperoleh dari optimisasi swarm partikel adalah $\alpha = 0.802008, a = 0.199208, b = 0.004980, c = 0.199110,\ d = 0.000996$ dengan nilai MAPE = 0.004150. Gambar 11.2 menyajikan grafik bidang fase kepadatan populasi mangsa – kepadatan populasi pemangsa.



Gambar 11.2. Bidang fase kepadatan populasi mangsa – kepadatan populasi pemangsa dari model orde fraksional

Bidang fase kepadatan populasi mangsa – kepadatan populasi pemangsa pada Gambar 11.2 berbentuk kurva spiral dengan arah pergerakan spiral menuju ke suatu titik (dalam hal ini titik setimbang). Hal ini merepresentasikan bersesuaian dengan solusi model matematika mangsa-pemangsa tipe Lotka-Volterra orde fraksional konvergen ke titik setimbang model.

12. Penutup

Pada bab-bab sebelumnya, metode optimisasi swarm parameter telah diimplementasikan untuk menentukan solusi berbagai masalah optimasi, yaitu penentuan akar persamaan nonlinear dan system persamaan nonlinear, penentuan nilai maksimum suatu fungsi multivariable, estimasi parameter model pertumbuhan populasi, termasuk estimasi parameter model pertumbuhan populasi orde fraksional dan estimasi parameter model matematika mangsa pemangsa. Salah satu keunggulan metode optimisasi swarm parameter adalah bahwa metode ini relative sederhana. Setelah proses inisialisasi, metode ini hanya terdiri dari tiga tahap yaitu update kecepatan partikel, update posisi partikel (nilai solusi) dan update posisi terbaik partikel. Kesederhanaan metode ini mengakibatkan bahwa metode optimisasi swarm partikel diterapkan untuk masalah optimasi tipe kontinu (masalah optimasi dengan variabel keputusan pada model merupakan variabel tipe kontinu).

Sebagai penutup isi buku, berikut disajikan beberapa karakteristik penting yang berhubungan dengan implementasi metode optimisasi swarm partikel:

(a) Seperti karakteristik metode heuristik lainnya (misalkan algoritma genetika dan algoritma kunang-kunang/*firefly algorithm*), metode optimisasi swarm partikel juga berpotensi konvergen ke solusi optimum lokal. Oleh karena itu, metode optimisasi swarm partikel perlu diimplementasikan beberapa kali untuk memperoleh solusi yang lebih baik. Untuk keperluan praktis, metode ini dapat diulang minimal lima kali ketika evaluasi fungsi tujuan pada masalah optimasi memerlukan waktu yang relatif lama (*time consuming*). Ketika evaluasi fungsi tujuan memerlukan waktu yang relatif singkat, metode optimisasi swarm partikel dapat diulang puluhan kali. Tujuan pengulangan tersebut adalah memperoleh "solusi terbaik" di antara solusi-solusi yang diperoleh dari implementasi metode optimisasi swarm partikel.

(b) Tingkat keakuratan dan kecepatan konvergensi metode optimisasi swarm partikel dipengaruhi oleh pemilihan nilai koefisien inersia (w), koefisien kognitif (c_1) dan koefisien social (c_2). Nilai optimal ketiga koefisien tersebut sangat dipengaruhi bentuk permasalahan optimasi. Secara praktis, para peneliti dapat mengimplementasikan metode optimisasi swarm partikel dengan nilai koefisien kognitif $c_1 = 2$, koefisien social $c_2 = 2$, dan nilai koefisien inersia (w) dalam interval $0.1 \leq w \leq 0.5$.

(c) Keakuratan solusi yang diperoleh dari metode optimisasi swarm partikel juga dipengaruhi oleh pemilihan rentang nilai variabel keputusan. Seperti yang telah diilustrasikan pada sub bab 11.3, setelah lima kali pengulangan metode optimisasi swarm partikel, dapat diperkirakan rentang nilai optimum untuk satu atau beberapa variabel keputusan. Selanjutnya, metode optimisasi swarm partikel dapat diulang minimal lima kali dengan menggunakan rentang optimum untuk satu atau beberapa variabel keputusan.

Daftar Pustaka

[1] M. Baudin, 2010, Introduction to Scilab, *The Scilab Consortium-Digiteo*.

[2] M. Baudin, 2011, Programming in Scilab. *The Scilab Consortium - Digiteo*.

[3] S.L. Campbell, J-P. Chancelier and R. Nikoukhah, 2006, Modeling and Simulation in Scilab/Scicos with Scicos, *Springer*.

[4] Windarto, S. W. Indratno, N. Nuraini, and E. Soewono, 2014, A comparison of binary and continuous genetic algorithm in parameter estimation of a logistic growth model, *AIP Conference Proceedings* 1587, pp. 139-142, doi: 10.1063/1.4866550.

[5] N. Tutkun, 2009, Parameter estimation in mathematical models using the real coded genetic algorithms, *Expert Systems with Applications* 36, pp. 3342-3345.

[6] R. L. Haupt and S. E. Haupt, 2004, Practical genetic algorithms, Second Edition, *John Wiley & Sons*.

[7] R. Eberhart and J. Kennedy, 1995, A new optimizer using particle swarm theory, *Proceedings of the Sixth International Symposium on Micro Machine and Human Science* 39-43.

[8] R. J. Kuo, M. J. Wang and T. W. Huang, T.W., 2011, An application of particle swarm optimization algorithm to clustering analysis, *Soft Computing* 15, pp. 533-542.

[9] J. Salerno, 1997, Using the particle swarm optimization technique to train a recurrent neural model, *Proceedings of the Ninth IEEE International Conference on Tools with Artificial Intelligence*.

[10] C. Zhang, H. Shao, and Y. Li, 2000, Particle swarm optimization for evolving artificial neural network, *IEEE international conference on systems, man and cybernetics*, 2487-2490.

[11] C.A. Koay and D. Srinivasan, 2003, Particle swarm optimization-based approach for generator maintenance scheduling. In: *Proceedings of the 2003 IEEE swarm intelligence symposium*, 167-173.

[12] X. Weijun, W. Zhiming, Z. Wei, and Y. Genke, 2004, A new hybrid optimization algorithm for the job-shop scheduling problem, *Proceedings of the 2004 American Control Conference*, 5552-5557.

[13] C. J. Liao, Chao-Tang Tseng, and P. Luarn, 2017, A discrete version of particle swarm optimization for flowshop scheduling problems, *Computers and Operations Research* 34(10): 3099-3111.

[14] K.P. Wang, L. Huang, C.G. Zhou, and W. Pang, 2003, Particle swarm optimization for traveling salesman problem, *2003 International Conference on Machine Learning and Cybernetics*, 1583-1585.

[15] B. Wu, Z. Yanwei, M. Yaliang, D. Hongzhao, and W. Weian, 2004, Particle swarm optimization method for vehicle routing problem, *Fifth World Congress on Intelligent Control and Automation*, pp. 2219-2221.

[16] J.M. Xiao, J.J. Li, and X.H. Wang 2005, Modified particle swarm optimization algorithm for vehicle routing problem, *Jisuanji Jicheng Zhizao Xitong (Computer Integrated Manufacturing Systems)* Vol. 11 No. 4, pp. 577-581.

[17] B. Niu, Q. Duan, J. Liu, L. Tan, and Y. Liu, 2017, A population-based clustering technique using particle swarm optimization and k-means, Natural Computing Vol 16. No. 1: 45-59.

[18] D.P. Rini, S.M. Shamsuddin, and S.S. Yuhaniz, 2011, Particle Swarm Optimization: Technique, System and Challenges, *International Journal of Computer Applications* 14(1).

[19] B.I. Schmitt, 2015, Convergence Analysis for Particle Swarm Optimization, *FAU University Press*, Erlangen.
[20] H. Ye, W. Luo, and Z. Li, 2013, Convergence Analysis of Particle Swarm Optimizer and Its Improved Algorithm Based on Velocity Differential Evolution, *Computational Intelligence and Neuroscience*, Volume 2013, Article ID 384125.
[21] Windarto, Eridani and U.D. Purwati, 2019, A comparison of continuous genetic algorithm and particle swarm optimization in parameter estimation of Gompertz growth model, *AIP Conference Proceedings* **2084**, article number 020017.
[22] S. López, J. France, W.J. Gerrits, M.S. Dhanoa, D.J. Humphries, J. Dikstra, 2000, A generalized Michaelis-Menten equation for analysis of growth, *Journal of Animal Science* 78(7): 1816-182.
[23] J.A. Vázquez, J.M. Lorenzo, P. Fuciños, D. Franco, 2012. Evaluation of non-linear equations to model different animal growths with mono and bisigmoid profiles. *Journal of Theoretical Biology* 314(7): 95-105.
[24] J.T. Teleken, A.C. Galvã, W.D.S. Robazza, 2017, Comparing non-linear mathematical models to describe growth of different animals, *Acta Scientiarum* 39: 73-81.
[25] S.E. Aggrey, 2002, Comparison of three nonlinear and spline regression models for describing chicken growth curves, *Poultry Science* 81(12):1782-1788.
[26] H. Nesetrilova, 2005, Multiphasic growth models for cattle. *Czech J. Anim. Sci.* 50 (8): 347–354.
[27] M. Selvaggi, V. Laudadio, C. Dario and V. Tufarelli, 2015, Modelling Growth Curves in a Nondescript Italian Chicken Breed: an Opportunity to Improve Genetic and Feeding Strategies, *J. Poult. Sci.* 52: 288–294.
[28] A.O. Raji, S.T. Mbap and J. Aliyu, 2014, Comparison of different models to describe growth of the japanese quail (*coturnix japonica*), *Trakia Journal of Sciences* 2:182–188.
[29] M. Topal and S.D. Bolukbasi, S.D., 2008, Comparison of nonlinear growth curve models in broiler chicken, *Journal of Applied Animal Research* 34(2):149-152.
[30] Windarto, Eridani and U.D. Purwati, 2018, A new modified logistic growth model for empirical use, *Communication in Biomathematical Sciences* 1(2):122-131.
[31] D. Franco, A. García, J.A. Vázquez, M. Fernández, J.A. Carril and J.M. Lorenzo, 2011, Curva de crecimiento de la raza cerco celta (subrariedad barcina) a diferentes edades de sacrifício, *Actas Iberoamericanas de Conservacíon Animal* 1(1): 259-263.
[32] V.B. Santos, E.A. Mareco and M.D.P Silva, 2013, Growth curves of Nile tilapia (*oreochromis niloticus*) strains cultivated at different temperatures, *Acta Scientiarum Animal Sciences* 35(3):235-242.
[33] W.G. Bardsley, R.A. Ackerman, N.A. Bukhari, D.C. Deeming and M.W. Ferguson, 1995, Mathematical models for growth in alligator (Alligator mississippiensis) embryos developing at different incubation temperatures, *Journal of Anatomy* 187(1):181-190.
[34] C.F.M. Mansano, M.V. Stéfani, M.M. Pereira and B.I. Macente, 2013, Deposição de nutrientes na carcaça de girinos de rã-touro, *Pesquisa Agropecuária Brasileira* 48(8): 885-891.
[35] A.M.A. El-Sayed, A.E.M. El-Mesiry and H.A.A. El-Saka, 2007, On the fractional-order logistic equation, *Applied Mathematics Letters* 20: 817–823.
[36] K. Diethelm and A.D. Freed, 1998, The FracPECE subroutine for the numerical solution of differential equations of fractional order, in S. Heinzel and T. Plesser (editors), 1999, *Forschung und wissenschaftliches Rechnen 1998*, pp. 57-71.

[37] F. Brauer and C. Castillo-Chavez, 2012, Mathematical models in population biology and epidemiology (second edition), *Springer*.
[38] S. Das and P.K. Gupta, 2011, A mathematical model on fractional Lotka-Volterra equation, *J. Theoretical Biology* 277(1):1-6.

Lampiran

Lampiran 1. Kode program untuk variasi nilai koefisien inersia w terhadap masalah optimasi pada persamaan (9.3)

```
// variasi_w_osp_p93.sce
clc;
clear;
xdel(winsid()); // Close all figure
tic();

nVar = 1;            // Banyaknya variabel keputusan
VarMin = [-3];
VarMax = [0];

// Membuka file function
exec('fungsi_ga.sce',-1); //

// Random seed generator
n = sum(clock( ));
rand("seed",n);

// Parameters OSP
MaxIt = 500;    // Maksimum banyaknya iterasi
nPop = 40;       // Ukuran populasi (ukuran swarm)
w = 0;           // Bobot inersia
c1 = 2;       // Koefisien personal
c2 = 2;       // Koefisien global
ntrial = 25; // Banyaknya implementasi metode OSP

// Batas nilai kecepatan partikel
VelMax = 0.1*(VarMax-VarMin);
VelMin = -VelMax;

// Inisialisasi
posisi = rand(nPop, nVar);
kecepatan = zeros(nPop, nVar);
biaya  = zeros(nPop, 1);
pbest  = zeros(nPop, nVar);
biaya_pbest = zeros(nPop, 1);
gbest = zeros(nPop, 1);
biaya_gbest = zeros(1,1);
hasil_OSP = zeros(ntrial, nVar + 1);
```

```
for kk = 1:ntrial
        disp('Trial = ' + string(kk) + ', w = ' + string(w));
        // Inisialisasi
        for i = 1:nPop
                // Inisialisasi posisi partikel
                posisi(i, :) = VarMin + (VarMax - VarMin) .* rand(1,nVar);

                // Hitung nilai fungsi tujuan untuk masing-masing posisi
                biaya(i) = fungsi_ga(posisi(i,:));

                // Update partikel terbaik lokal
                pbest(i,:) = posisi(i,:);
                biaya_pbest(i) = biaya(i);

                // Update partikel terbaik global
                if (i == 1)
                    gbest = posisi(i,:);
                    biaya_gbest = biaya(i);
                else
                    if (biaya(i) < biaya_gbest)
                            gbest = posisi(i,:);
                            biaya_gbest = biaya(i);
                    end // if (biaya(i) < biaya_gbest)
                end // if (i == 1)
        end // for i = 1:nPop
        4
        BestCost = zeros(MaxIt,1);

        // Loop metode OSP
        for it = 1:MaxIt
                for i=1:nPop
                // Update kecepatan partikel
                    kecepatan(i,:) = w*kecepatan(i,:) + c1 * rand(1,nVar) ...
                            .* (pbest(i,:) - posisi(i,:)) + c2 * rand(1,nVar) ...
                            .* (gbest - posisi(i,:));
                // Terapkan nilai batas kecepatan partikel
                kecepatan(i,:) = max(kecepatan(i,:), VelMin);
                kecepatan(i,:) = min(kecepatan(i,:), VelMax);

                // Update posisi partikel
                posisi(i,:) = posisi(i,:) + kecepatan(i,:);

                // Terapkan nilai batas posisi partikel
                posisi(i,:) = max(posisi(i,:), VarMin);
                posisi(i,:) = min(posisi(i,:), VarMax);
                // Hitung nilai fungsi tujuan untuk masing-masing posisi
```

```
            biaya(i) = fungsi_ga(posisi(i,:));

            // Update posisi partikel terbaik lokal
            if (biaya(i) < biaya_pbest(i))
                pbest(i,:) = posisi(i,:);
                biaya_pbest(i) = biaya(i);
                // Update posisi partikel terbaik global
                if (biaya(i) < biaya_gbest)
                        gbest = posisi(i,:);
                        biaya_gbest = biaya(i);
                end
            end // if (biaya(i) < biaya_pbest(i))
        end // for i = 1:nPop

        BestCost(it) = biaya_gbest;
    end // for it = 1:MaxIt
    disp('Trial ' + string(kk) + ', iterasi ' + string(it) + ', hampiran akar x = ' ...
        + string(gbest) + ', z = ' + string(BestCost(it)));
    hasil_OSP(kk,:) = [gbest, biaya_gbest]
end // for kk = 1:ntrial

// Grafik nilai fungsi tujuan
scf(1);
xn = (1:1:MaxIt).';
plot2d('nl',xn,BestCost,style = [1]);
title('Nilai minimum fungsi tujuan',"fontsize",3.5);
xlabel('Iterasi',"fontsize",3.5);
ylabel('Nilai z',"fontsize",3.5);
gca().grid=[1 1 1]; // Setara dengan grid on;
gca().children(1).children(1).thickness = 3; // Set ketebalan garis kurva
axes_handle.grid = [1 1 1]; // Setara dengan grid(axes_handle,"on")

waktu = toc()
disp(waktu)
// Akhir kode program variasi_w_osp_p93.sce
```

Lampiran 2. Kode program untuk variasi nilai koefisien inersia w pada estimasi parameter model pertumbuhan logistik

```
// variasi_w_osp_logistik.sce
clc;
clear;
xdel(winsid( )); // Close all figure
tic();

// Variabel global
global tdata ydata ndata;

// tdata dan xdata
tdata = [0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, ...
        48, 51, 54, 57, 71, 85, 99, 113, 127, 141, 155, 170].';
ydata = [37, 41.74, 59.19, 79.94, 102.96, 132.13, 170.18, 206.56, ...
        250.71, 285.27, 324.92, 372.83, 417.41, 469.13, 519.72, ...
        577.27, 633.59, 667.18, 717.17, 786.35, 1069.28, 1326.49, ...
        1589.71, 1859.26, 2015.44, 2142.31, 2220.54, 2262.63].';
ndata = length(tdata);

nVar = 3;            // Banyaknya variabel keputusan
VarMin = [0, 2000, 40];      // Nilai minimum variabel keputusan
VarMax = [1, 5000, 90];      // Nilai maksimum variabel keputusan

// Membuka file function
exec('MSE_logistik.sce',-1);

// Random seed generator
n = sum(clock( ));
rand("seed",n);

// Parameters OSP
MaxIt = 200;    // Maksimum banyaknya iterasi
nPop = 40;        // Ukuran populasi (ukuran swarm)
w = 0.1;          // Bobot inersia
c1 = 2.0;       // Koefisien personal
c2 = 2.0;       // Koefisien global
ntrial = 25; // Banyaknya implementasi metode OSP

// Batas nilai kecepatan partikel
VelMax = 0.1*(VarMax - VarMin);
VelMin = - VelMax;
```

```
// Batas nilai kecepatan partikel
VelMax = 0.1*(VarMax-VarMin);
VelMin = -VelMax;

// Inisialisasi
pbest  = zeros(nPop, nVar);
biaya_pbest = zeros(nPop, 1);
gbest = zeros(nPop, 1);
biaya_gbest = zeros(1,1);
hasil_OSP = zeros(ntrial, nVar + 1);

for kk = 1:ntrial
    // Inisialisasi
    posisi = rand(nPop, nVar);
    kecepatan = zeros(nPop, nVar);
    biaya  = zeros(nPop, 1);
    disp('Trial = ' + string(kk) + ', w = ' + string(w));
    for i = 1:nPop
    // Inisialisasi posisi partikel
    posisi(i, :) = VarMin + (VarMax - VarMin) .* rand(1,nVar);

    // Hitung nilai fungsi tujuan untuk masing-masing posisi
    [yhitung, biaya(i)] = MSE_logistik(posisi(i,:));

    // Update partikel terbaik lokal
    pbest(i,:) = posisi(i,:);
    biaya_pbest(i) = biaya(i);

    // Update partikel terbaik global
    if (i == 1)
        gbest = posisi(i,:);
        biaya_gbest = biaya(i);
        yhitung_terbaik = yhitung;
    else
        if (biaya(i) < biaya_gbest)
            gbest = posisi(i,:);
            biaya_gbest = biaya(i);
            yhitung_terbaik = yhitung;
        end // if (biaya(i) < biaya_gbest)
    end // if (i == 1)
end // for i = 1:nPop

BestCost = zeros(MaxIt,1);

// Loop metode OSP
    for it = 1:MaxIt
```

```
        for i=1:nPop
            // Update kecepatan partikel
            kecepatan(i,:) = w*kecepatan(i,:) + c1 * rand(1,nVar) ...
            .* (pbest(i,:) - posisi(i,:)) + c2 * rand(1,nVar) ...
            .* (gbest - posisi(i,:));
            // Terapkan nilai batas kecepatan partikel
              kecepatan(i,:) = max(kecepatan(i,:), VelMin);
              kecepatan(i,:) = min(kecepatan(i,:), VelMax);

              // Update posisi partikel
              posisi(i,:) = posisi(i,:) + kecepatan(i,:);

            // Terapkan nilai batas posisi partikel
              posisi(i,:) = max(posisi(i,:), VarMin);
              posisi(i,:) = min(posisi(i,:), VarMax);
              // Hitung nilai fungsi tujuan untuk masing-masing posisi
              [yhitung, biaya(i)] = MSE_logistik(posisi(i,:));

              // Update posisi partikel terbaik lokal
              if (biaya(i) < biaya_pbest(i))
                 pbest(i,:) = posisi(i,:);
                 biaya_pbest(i) = biaya(i);
                 // Update posisi partikel terbaik global
                 if (biaya(i) < biaya_gbest)
                    gbest = posisi(i,:);
                  biaya_gbest = biaya(i);
                  yhitung_terbaik = yhitung;
               end
            end // if (biaya(i) < biaya_pbest(i))
         end // for i = 1:nPop

         BestCost(it) = biaya_gbest;
      end // for it = 1:MaxIt
      disp('Trial ' + string(kk) + ', iterasi ' + string(it) + ...
            ', r = ' + string(gbest(1)) + ', K = ' + string(gbest(2)) + ...
            ', tinf = ' + string(gbest(3)) + ', MSE = ' + string(BestCost(it)));

      hasil_OSP(kk,:) = [gbest, biaya_gbest];
   end // for kk = 1:ntrial

   // Simpangan baku MSE
   sd_OSP = stdev(hasil_OSP, 1)
   sd_MSE = stdev(hasil_OSP(:,nVar + 1))

   // Indeks terbaik dan parameter terbaik
   min_MSE = min(hasil_OSP(:,nVar+1))
```

```
id_minMSE = find(hasil_OSP(:,nVar+1)== min_MSE,1)
parameter_terbaik = hasil_OSP(id_minMSE,1:nVar)

// Ekstrak parameter terbaik
dt = 0.1;
r = parameter_terbaik(1);
K = parameter_terbaik(2);
tinf = parameter_terbaik(3);
tt = (tdata(43):dt:tdata($)).';
yhitung = K ./ (1 + exp(-r*(tdata - tinf)));
yy = K ./ (1 + exp(-r*(tt - tinf)));
dy = r*yy.*(1 - yy/K);

// Grafik nilai fungsi tujuan
scf(1);
xn = (1:1:MaxIt).';
plot2d('nl',xn,BestCost,style = [1]);
title('Nilai minimum fungsi tujuan',"fontsize",3.5);
xlabel('Iterasi',"fontsize",3.5);
ylabel('Nilai MSE',"fontsize",3.5);
gca().grid=[1 1 1]; // Setara dengan grid on;
gca().children(1).children(1).thickness = 3; // Set ketebalan garis kurva
axes_handle.grid = [1 1 1]; // Setara dengan grid(axes_handle,"on")

scf(2);
plot(tdata,ydata,'ko',"linewidth",3.5);
set(gca(),"auto_clear","off"); // hold on
plot(tt,yy,'k',"linewidth",3.5);
title('Berat ayam jantan (gram)',"fontsize",3.5);
xlabel('Waktu (hari)',"fontsize",3.5);
ylabel('Berat ayam jantan (gram)',"fontsize",3.5);
hl=legend(['data';'model'],[2]);
gca().grid=[1 1 1]; // Setara dengan grid on;
gca().children(1).children(1).thickness = 3; // Set ketebalan garis kurva
axes_handle.grid = [1 1 1]; // Setara dengan grid(axes_handle,"on")

scf(3);
plot2d('nn',tt,dy,style = [1]);
title('Pertambahan berat ternak / hari (gram/hari)',"fontsize",3.5);
xlabel('Waktu (hari)',"fontsize",3.5);
ylabel('Pertambahan berat ternak (gram/hari)',"fontsize",3.5);
gca().grid=[1 1 1]; // Setara dengan grid on;
gca().children(1).children(1).thickness = 3; // Set ketebalan garis kurva
axes_handle.grid = [1 1 1]; // Setara dengan grid(axes_handle,"on")
```

```
waktu = toc( );
disp('waktu = ' + string(waktu) + ' detik');
// Akhir kode program variasi_w_osp_logistik.sce
```

Lampiran 3. Kode program pengulangan metode optimisasi swarm partikel pada estimasi parameter model mangsa-pemangsa

```
// pengulangan_osp_predator.sce
clc;
clear;
xdel(winsid()); // Close all figure
tic();

// Variabel global
global tdata xdata ydata ndata tt indeks_t dt;

// tdata dan xdata
tdata = (0:20).';
xdata = [250, 305, 365, 420, 453, 444, 391, 314, 240, 185, 149, 129, ...
       120, 119, 126, 141, 163, 195, 237, 289, 349].';
ydata = [30, 31, 34, 39, 47, 57, 68, 75, 77, 74, 68, 61, 54, 47, 42, ...
        37, 33, 31, 30, 30, 33].';
ndata = length(tdata);
dt = 1e-1;
tt = (tdata(1):dt:tdata(ndata)).';
nt = length(tt);
indeks_t = zeros(ndata,1);
mm = 1;
for kk = 1:nt
   if (tt(kk) == tdata(mm))
      indeks_t(mm,:) = kk;
      mm = mm + 1;
   end
end // for kk = 1:nt
// nVar, VarMin, VarMax
nVar = 4;                  // Banyaknya variabel keputusan
VarMin = [0.1, 0.00, 0.1, 0.00];      // Nilai minimum variabel keputusan
VarMax = [1.0, 0.02, 1.0, 0.02];      // Nilai maksimum variabel keputusan

// Membuka file function
exec('MAPE_predator.sce',-1);

// Random seed generator
n = sum(clock( ));
rand("seed",n);

// Parameters OSP
MaxIt = 200;    // Maksimum banyaknya iterasi
```

```
nPop = 25;        // Ukuran populasi (ukuran swarm)
w = 0.5;          // Bobot inersia
c1 = 2.0;       // Koefisien personal
c2 = 2.0;       // Koefisien global
ntrial = 5;

// Batas nilai kecepatan partikel
VelMax = 0.1*(VarMax - VarMin);
VelMin = - VelMax;

// Inisialisasi
pbest  = zeros(nPop, nVar);
biaya_pbest = zeros(nPop, 1);
gbest = zeros(nPop, 1);
biaya_gbest = zeros(1,1);
hasil_OSP = zeros(ntrial, nVar + 1);

for kk = 1:ntrial
   // Inisialisasi
   posisi = zeros(nPop, nVar);
   kecepatan = zeros(nPop, nVar);
   biaya  = zeros(nPop, 1);
   disp('Trial = ' + string(kk) + ', w = ' + string(w));
   for i = 1:nPop
   // Inisialisasi posisi partikel
   posisi(i, :) = VarMin + (VarMax - VarMin) .* rand(1,nVar);

   // Hitung nilai fungsi tujuan untuk masing-masing posisi
   [biaya(i), xhitung,yhitung, xx, yy] = MAPE_predator(posisi(i,:));

   // Update partikel terbaik lokal
   pbest(i,:) = posisi(i,:);
   biaya_pbest(i) = biaya(i);

   // Update partikel terbaik global
   if (i == 1)
      gbest = posisi(i,:);
      biaya_gbest = biaya(i);
      yhitung_terbaik = yhitung;
   else
      if (biaya(i) < biaya_gbest)
         gbest = posisi(i,:);
         biaya_gbest = biaya(i);
         yhitung_terbaik = yhitung;
      end // if (biaya(i) < biaya_gbest)
```

```
    end // if (i == 1)
end // for i = 1:nPop
4
BestCost = zeros(MaxIt,1);

// Loop utama OSP
    for it = 1:MaxIt
        for i=1:nPop
            // Update kecepatan partikel
            kecepatan(i,:) = w*kecepatan(i,:) + c1 * rand(1,nVar) ...
            .* (pbest(i,:) - posisi(i,:)) + c2 * rand(1,nVar) ...
            .* (gbest - posisi(i,:));
            // Terapkan nilai batas kecepatan partikel
            kecepatan(i,:) = max(kecepatan(i,:), VelMin);
            kecepatan(i,:) = min(kecepatan(i,:), VelMax);

            // Update posisi partikel
            posisi(i,:) = posisi(i,:) + kecepatan(i,:);

            // Terapkan nilai batas posisi partikel
            posisi(i,:) = max(posisi(i,:), VarMin);
            posisi(i,:) = min(posisi(i,:), VarMax);

            // Hitung nilai fungsi tujuan untuk masing-masing posisi
            [biaya(i), xhitung,yhitung, xx, yy] = MAPE_predator(posisi(i,:));

            // Update posisi partikel terbaik lokal
            if (biaya(i) < biaya_pbest(i))
                pbest(i,:) = posisi(i,:);
                biaya_pbest(i) = biaya(i);
                // Update posisi partikel terbaik global
                if (biaya(i) < biaya_gbest)
                    gbest = posisi(i,:);
                    biaya_gbest = biaya(i);
                    xhitung_terbaik = xhitung;
                    xx_terbaik = xx;
                    yhitung_terbaik = yhitung;
                    yy_terbaik = yy;
                end
             end // if (biaya(i) > biaya_pbest(i))
        end // for i = 1:nPop

        BestCost(it) = biaya_gbest;
        disp('Trial ' + string(kk) + ', iterasi ' + string(it) + ...
         ', a = ' + string(gbest(1)) + ', b = ' + string(gbest(2)) + ...
         ', c = ' + string(gbest(3)) + ', d = ' + string(gbest(4)) + ...
```

```
        ', MAPE = ' + string(BestCost(it)));
    end // for it = 1:MaxIt
        hasil_OSP(kk,:) = [gbest, biaya_gbest];
end // for kk = 1:ntrial

// Simpangan baku MSE
sd_OSP = stdev(hasil_OSP, 1)
sd_MAPE = stdev(hasil_OSP(:,nVar + 1))

// Indeks terbaik dan parameter terbaik
min_MAPE = min(hasil_OSP(:,nVar+1))
id_minMAPE = find(hasil_OSP(:,nVar+1)== min_MAPE,1)
parameter_terbaik = hasil_OSP(id_minMAPE,1:nVar)

// Ekstrak parameter terbaik
[hitung_MAPE, xhitung,yhitung, xx, yy] = ...
    MAPE_predator(parameter_terbaik(1,:));

// Grafik nilai fungsi tujuan
scf(0);
xn = (1:1:MaxIt).';
plot2d('nl',xn,BestCost,style = [1]);
title('Nilai rata-rata galat relatif (MAPE)',"fontsize",3.5);
xlabel('Iterasi',"fontsize",3.5);
ylabel('Nilai MAPE',"fontsize",3.5);
gca().grid=[1 1 1]; // Setara dengan grid on;
gca().children(1).children(1).thickness = 3; // Set ketebalan garis kurva
axes_handle.grid = [1 1 1]; // Setara dengan grid(axes_handle,"on")

scf(1);
plot(xdata,ydata,'ko',"linewidth",3);
set(gca(),"auto_clear","off"); // hold on
plot(xx,yy,'k',"linewidth",3);
hl=legend(['data';'model'],[1]);
gca.grid=[1 1 1]; // Setara dengan grid on;
//gca().x_location ="origin";
gca().data_bounds=[0.99*min(xx),0.99*min(yy);...
                   1.01*max(xx),1.01*max(yy)];
title('Bidang fase kepadatan predator-prey',"fontsize",3.5);
xlabel('Kepadatan populasi prey (individu/satuan luas)',"fontsize",3.5);
ylabel('Kepadatan populasi predator (individu/satuan luas)',"fontsize",3.5);

waktu = toc( );
disp('waktu = ' + string(waktu) + ' detik');
// Akhir kode program pengulangan_osp_predator.sce
```

Optimisasi Swarm Partikel Menggunakan Scilab

ORIGINALITY REPORT

7%	7%	5%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	pt.scribd.com Internet Source	1%
2	www.scribd.com Internet Source	1%
3	aip.scitation.org Internet Source	<1%
4	repository.kallipos.gr Internet Source	<1%
5	scitepress.org Internet Source	<1%
6	es.scribd.com Internet Source	<1%
7	"Reactive Power Control in AC Power Systems", Springer Science and Business Media LLC, 2017 Publication	<1%
8	id.123dok.com Internet Source	<1%

9	flimsy89.blogspot.com Internet Source	<1%
10	link.springer.com Internet Source	<1%
11	polen.itu.edu.tr Internet Source	<1%
12	Walter Gander, Martin J. Gander, Felix Kwok. "Scientific Computing - An Introduction using Maple and MATLAB", Springer Science and Business Media LLC, 2014 Publication	<1%
13	repositorio.unb.br Internet Source	<1%
14	Dian Rachmawati, Lysander Gustin. "Analysis of Dijkstra's Algorithm and A* Algorithm in Shortest Path Problem", Journal of Physics: Conference Series, 2020 Publication	<1%
15	www.essex.ac.uk Internet Source	<1%
16	edoc.pub Internet Source	<1%
17	M. Sambath, P. Ramesh, K. Balachandran. "Asymptotic Behavior of the Fractional Order three Species Prey–Predator Model",	<1%

	International Journal of Nonlinear Sciences and Numerical Simulation, 2018 Publication	
18	livedna.net Internet Source	<1%
19	José Antonio Vázquez, María Blanco, Javier Fraguas, Lorenzo Pastrana, Ricardo Pérez-Martín. "Optimisation of the extraction and purification of chondroitin sulphate from head by-products of Prionace glauca by environmental friendly processes", Food Chemistry, 2016 Publication	<1%
20	www.iaeng.org Internet Source	<1%
21	bioone.org Internet Source	<1%
22	revistas.bvs-vet.org.br Internet Source	<1%
23	www.turkishpoultryscience.com Internet Source	<1%
24	Submitted to Universitas Airlangga Student Paper	<1%
25	McBride, A.C.. "Strongly differentiable solutions of the discrete coagulation-fragmentation	<1%

equation", Physica D: Nonlinear Phenomena, 20100801
Publication

26	uvadoc.uva.es Internet Source	<1%
27	jurnal.stmikasia.ac.id Internet Source	<1%
28	tema.sbmac.emnuvens.com.br Internet Source	<1%
29	academic.oup.com Internet Source	<1%
30	gitlab.devlabs.linuxassist.net Internet Source	<1%
31	Shidrokh Goudarzi, Wan Haslina Hassan, Mohammad Hossein Anisi, Seyed Ahmad Soleymani. "Comparison between hybridized algorithm of GA–SA and ABC, GA, DE and PSO for vertical-handover in heterogeneous wireless networks", Sādhanā, 2016 Publication	<1%
32	Lawson G. Sugden, E. A. Driver, Michael C. S. Kingsley. "Growth and energy consumption by captive mallards", Canadian Journal of Zoology, 1981 Publication	<1%

33	"Computational Intelligence, Networked Systems and Their Applications", Springer Science and Business Media LLC, 2014 Publication	<1%
34	symopis.vggs.rs Internet Source	<1%
35	131.111.17.133 Internet Source	<1%
36	"Instrumentation, Measurement, Circuits and Systems", Springer Science and Business Media LLC, 2012 Publication	<1%
37	documents.mx Internet Source	<1%
38	docplayer.info Internet Source	<1%
39	juliegiroux.www2.50megs.com Internet Source	<1%
40	prosiding.upgris.ac.id Internet Source	<1%
41	Şuayip Yüzbaşı. "A collocation method for numerical solutions of fractional-order logistic population model", International Journal of Biomathematics, 2016 Publication	<1%

42	repository.up.ac.za Internet Source	<1%
43	Saralees Nadarajah. "On the Linear Combination of Laplace and Logistic Random Variables", Journal of Applied Statistics, 3/2007 Publication	<1%
44	Won Y. Yang, Wenwu Cao, Jaekwon Kim, Kyung W. Park et al. "Applied Numerical Methods Using Matlab®", Wiley, 2020 Publication	<1%
45	www.math.kobe-u.ac.jp Internet Source	<1%
46	scienceofgeography.blogspot.com Internet Source	<1%
47	Franco, Daniel, José Antonio Vazquez, and José Manuel Lorenzo. "Growth performance, carcass and meat quality of the Celta pig crossbred with Duroc and Landrance genotypes", Meat Science, 2014. Publication	<1%
48	Sanjeewa Nishantha Perera, Naleen C Ganegoda, Dhammika Deepani Siriwardhana, Manuj C Weerasinghe. "Mathematical Model to Study Early COVID-19 Transmission Dynamics in Sri Lanka", Cold Spring Harbor Laboratory, 2020	<1%

	Publication	
49	mjas.my Internet Source	<1%
50	pperso.th.u-psud.fr Internet Source	<1%
51	etheses.uin-malang.ac.id Internet Source	<1%
52	www.uta.edu Internet Source	<1%
53	tumpalmanurung05itc.blogspot.com Internet Source	<1%
54	bibliotecadigital.fgv.br Internet Source	<1%
55	h-es.ru Internet Source	<1%
56	www.progeltd.com Internet Source	<1%
57	www.mec.ita.cta.br Internet Source	<1%
58	cgit.scilab.org Internet Source	<1%
59	www.musliner.com Internet Source	<1%

60	www.beyondmusic.cn Internet Source	<1%
61	www.enviro.bppt.go.id Internet Source	<1%
62	hal.archives-ouvertes.fr Internet Source	<1%
63	wahyudi-informatika.blogspot.com Internet Source	<1%
64	repository.unair.ac.id Internet Source	<1%

Exclude quotes	Off	Exclude matches	< 10 words
Exclude bibliography	On		