FUZZY ADAPTIVE TURBULENCE PARTICLE SWARM OPTIMIZATION (FATPSO) UNTUK MASALAH OPTIMASI FUNGSI NONLINIER

SKRIPSI

Oleh:
INDRA NOOR DIANTO
0510940026-94

PROGRAM STUDI MATEMATIKA
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG
2009

FUZZY ADAPTIVE TURBULENCE PARTICLE SWARM OPTIMIZATION (FATPSO) UNTUK MASALAH OPTIMASI FUNGSI NONLINIER

SKRIPSI

Sebagai salah satu syarat untuk memperoleh gelar
Sarjana Sains dalam bidang matematika

Oleh:
INDRA NOOR DIANTO
0510940026-94

PROGRAM STUDI MATEMATIKA
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG

2009


UNIVERSITAS BRAWIJAYA

LEMBAR PENGESAHAN SKRIPSI

FUZZY ADAPTIVE TURBULENCE PARTICLE SWARM OPTIMIZATION (FATPSO) UNTUK MASALAH OPTIMASI FUNGSI NONLINIER

Oleh:
INDRA NOOR DIANTO
0510940026-94

Setelah dipertahankan di depan Majelis Penguji pada tanggal 12 Agustus 2009 dan dinyatakan memenuhi syarat untuk memperoleh gelar Sarjana Sains dalam bidang Matematika

Pembimbing I	Pembimbing II
Syaiful Anam, SSi., MT	**Dr. Wuryansari M. K, MSi**
NIP.132 300 237	**NIP. 132 048 784**

Mengetahui,
Ketua Jurusan Matematika
Fakultas MIPA Universitas Brawijaya

Dr. Agus Suryanto, MSc
NIP. 132 126 049


UNIVERSITAS BRAWIJAYA

LEMBAR PERNYATAAN

Saya yang bertanda tangan di bawah ini :

Nama	**: Indra Noor Dianto**
NIM	**: 0510940026**
Jurusan	**: Matematika**
Penulis Skripsi berjudul	**: *Fuzzy Adaptive Turbulence Particle Swarm Optimization* (FATPSO) untuk Masalah Optimasi Fungsi Nonlinier**

Dengan ini menyatakan bahwa :

1. **Skripsi ini adalah benar-benar karya saya sendiri dan bukan hasil plagiat dari karya orang lain. Karya-karya yang tercantum dalam Daftar Pustaka, semata-mata digunakan sebagai acuan/referensi.**
2. **Apabila di kemudian hari diketahui bahwa isi Skripsi saya merupakan hasil plagiat, maka saya bersedia menanggung akibat hukum dari keadaan tersebut.**

Demikian pernyataan ini dibuat dengan segala kesadaran.

Malang, 12 Agustus 2009
Yang menyatakan,

(Indra Noor Dianto)
NIM 0510940026

FUZZY ADAPTIVE TURBULENCE PARTICLE SWARM OPTIMIZATION (FATPSO) UNTUK MASALAH OPTIMASI FUNGSI NONLINIER

ABSTRAK

Algoritma *Particle Swarm Optimization* (PSO) adalah algoritma optimasi yang memiliki tingkat kekonvergenan yang tinggi dan sudah banyak diterapkan pada berbagai aplikasi. Akan tetapi, algoritma ini hampir selalu mengalami konvergen prematur apabila diimplementasikan untuk menyelesaikan masalah optimasi berdimensi tinggi. Hal ini merupakan akibat terjadinya kondisi stagnan sehingga hasil yang diperoleh tidak optimal. Penyebab terjadinya kondisi stagnan adalah terjadinya penurunan kecepatan partikel. Untuk mengatasi kondisi tersebut, diperkenalkan algoritma *Turbulence Particle Swarm Optimization* (TPSO). Algoritma ini menggunakan batasan kecepatan minimum untuk mengontrol kecepatan partikel. Dengan melakukan analisis kekonvergenan diketahui bahwa algoritma TPSO konvergen ke titik optimum global dengan peluang 1. Parameter batasan kecepatan minimum dikontrol secara adaptif menggunakan *Fuzzy Logic Controller* (FLC) sehingga disebut algoritma *Fuzzy Adaptive Turbulence Particle Swarm Optimization* (FATPSO).

Hasil dan waktu komputasi algoritma FATPSO dibandingkan dengan hasil dan waktu komputasi algoritma TPSO dengan menggunakan beberapa *test function* untuk mengetahui keefektifan penggunaan FLC dalam algoritma FATPSO. Berdasarkan hasil yang diperoleh, diketahui bahwa algoritma FATPSO dapat menangani kondisi stagnan lebih baik daripada algoritma TPSO, tetapi waktu komputasi yang dibutuhkan lebih lama.

Kata kunci : Algoritma PSO, TPSO, FATPSO, *Fuzzy Logic Controller* (FLC), konvergen prematur, kondisi stagnan.

FUZZY ADAPTIVE TURBULENCE PARTICLE SWARM OPTIMIZATION (FATPSO) FOR OPTIMIZATION OF NONLINEAR FUNCTION

ABSTRACT

Particle Swarm Optimization (PSO) algorithm is an optimization algorithm which has good convergency and has been implemented in many applications. However, the algorithm tends to premature convergence condition when it is implemented on high dimension problems. This condition is caused by stagnant condition and therefore it gives nonoptimal result. The cause of this stagnant condition is the decreasing on the velocity of particles. To overcome the problem, we introduce Turbulence Particle Swarm Optimization (TPSO) algorithm, which uses a minimum velocity threshold to control the velocity of particles. Convergency analysis shows that algorithm converges with a probability of 1 towards the global optimal. Since the minimum velocity threshold parameter is controlled adaptively by a Fuzzy Logic Controller (FLC), this algorithm is called Fuzzy Adaptive Turbulence Particle Swarm Optimization (FATPSO).

The performance of FATSPO and TPSO algorithm are compared for some test functions to investigate the effectiveness of FLC in FATPSO algorithm. The results show that FATPSO can handle the stagnant condition better than TPSO algorithm, but it needs much longer time of computations.

Keywords : PSO algorithm, TPSO, FATPSO, Fuzzy Logic Controller (FLC), premature convergence, stagnant condition.

KATA PENGANTAR

Segala puji dan syukur Alhamdulillah penulis panjatkan ke hadirat Allah SWT, yang telah mencurahkan rahmat, hidayah dan inayah-Nya sehingga penulis dapat menyelesaikan Skripsi ini dengan judul "***Fuzzy Adaptive Turbulence Particle Swarm Optimization* (FATPSO) untuk Masalah Optimasi Fungsi Nonlinier**". Sholawat serta salam semoga tercurahkan kepada Baginda Rosulullah Nabi Muhammad SAW.

Banyak pihak yang telah memberikan dukungan baik moral maupun spiritual secara langsung maupun tidak langsung dalam penyelesaian skripsi ini. Penulis mengucapkan terima kasih kepada :

1. Syaiful Anam, SSi., MT selaku pembimbing I atas segala bimbingan dan motivasi yang telah diberikan selama penulisan skripsi ini.
2. Dr.Wuryansari M.K, MSi selaku pembimbing II sekaligus Ketua Program Studi Matematika atas segala bimbingan dan motivasi yang telah diberikan selama penulisan skripsi ini.
3. Drs. Noor Hidayat, MSi, Dr. Agus Suryanto, MSc, dan Drs. Muslikh, MSi selaku dosen penguji atas segala saran yang diberikan untuk perbaikan skripsi ini.
4. Drs. Sobri Abusini, MT selaku dosen pembimbing akademik atas segala bimbingan dan motivasi yang telah diberikan.
5. Kedua orang tuaku dan adik-adikku atas doa dan dukungan yang telah diberikan.
6. Sahabat-sahabatku, teman seperjuangan Math'05 dan anak-anak kos Panjaitan XIX/43A atas dukungan dan gangguannya.
7. Serta semua pihak yang tidak dapat disebutkan satu persatu.

Penulis menyadari bahwa manusia adalah tempatnya kekhilafan dan tak akan lepas dari kesalahan, oleh karena itu segala kritik dan saran dari pembaca sangat diharapkan demi perbaikan selanjutnya. Semoga tulisan ini bermanfaat bagi penulis khususnya serta semua pihak pada umumnya.

Malang, Agustus 2009

Penulis

DAFTAR ISI

Halaman

BAB III HASIL DAN PEMBAHASAN

DAFTAR TABEL

Halaman

DAFTAR GAMBAR

Halaman

DAFTAR ALGORITMA

Halaman


UNIVERSITAS BRAWIJAYA

DAFTAR LAMPIRAN

Halaman

BAB I
PENDAHULUAN

1.1 Latar Belakang

Algoritma *Particle Swarm Optimization* (PSO) merupakan sebuah algoritma optimasi yang diperkenalkan oleh Eberhart dan Kennedy pada tahun 1995, yang terinspirasi oleh perilaku sosial sekawanan burung atau ikan (Bergh, 2006). Algoritma ini merupakan salah satu algoritma optimasi global yang sudah banyak diterapkan pada berbagai aplikasi, seperti pembelajaran pada jaringan syaraf tiruan, pengontrol pembangkit listrik dan permainan sudoku (Shi, 2004).

Selain mudah diimplementasikan, algoritma PSO juga memiliki tingkat konvergensi yang tinggi, yaitu memiliki kemampuan untuk menemukan solusi secara cepat. Akan tetapi, kemampuannya menurun seiring bertambahnya dimensi masalah optimasi, khususnya apabila diimplementasikan untuk mengoptimasi fungsi multimodal. Hal ini dikarenakan, fungsi jenis ini memiliki lebih dari satu nilai optimum sehingga terdapat kemungkinan algoritma ini terjebak pada titik optimum lokal fungsi. Fenomena yang demikian dikenal sebagai konvergensi prematur. Berdasarkan analisis kekonvergenan, dapat diketahui bahwa seiring meningkatnya jumlah iterasi, pola pergerakan partikel dalam algoritma PSO mengalami kondisi stagnan sebagai akibat terjadinya penurunan kecepatan. Dengan demikian, dapat disimpulkan bahwa kondisi stagnan inilah yang menyebabkan terjadinya konvergensi prematur (Abraham dan Liu, 2005).

Pada skripsi ini dibahas algoritma *Turbulent Particle Swarm Optimization* (TPSO) untuk mengatasi kondisi stagnan tersebut, yaitu dengan memberikan batasan nilai kecepatan minimum partikel dengan tujuan untuk menjaga keanekaragaman populasi dan menghindari terjadinya pengelompokan partikel. Lebih lanjut, batasan nilai minimum ini dikontrol secara adaptif menggunakan *Fuzzy Logic Controller*, sehingga diperoleh algoritma yang disebut dengan *Fuzzy Adaptive Turbulent Particle Swarm Optimization* (FATPSO).

Fuzzy Logic Controller merupakan salah satu aplikasi sistem *fuzzy* dan merupakan pengontrol berbasis logika *fuzzy* yang sering digunakan sebagai sistem pengontrol dalam berbagai bidang. Sistem *fuzzy* ini bekerja berdasarkan pendekatan logika *fuzzy* bukan logika

tegas (*crisp*). Oleh karena itu, sistem ini dapat digunakan untuk menangani sistem dengan struktur yang tidak terdefinisi dengan jelas seperti perubahan variasi parameter-parameter operasi (Jang, dkk. 1997).

1.2 Perumusan Masalah

Berdasarkan latar belakang tersebut, pokok permasalahan yang dibahas dalam skripsi ini adalah sebagai berikut:

1. Bagaimana analisis konvergensi algoritma *Turbulent Particle Swarm Optimization* (TPSO)?
2. Bagaimana mengimplementasikan *Fuzzy Logic Controller* pada algoritma *Fuzzy Adaptive Turbulent Particle Swarm Optimization* (FATPSO)?
3. Bagaimana hasil dan waktu komputasi algoritma *Fuzzy Adaptive Turbulent Particle Swarm Optimization* (FATPSO) apabila dibandingkan dengan hasil dan waktu komputasi algoritma *Turbulence Particle Swarm Optimization* (TPSO) dalam mengatasi kondisi stagnan yang dialami oleh algoritma *Particle Swarm Optimization* (PSO) pada penyelesaian masalah optimasi fungsi nonlinier?

1.3 Tujuan

Tujuan penulisan skripsi ini adalah untuk:

1. Melakukan analisis kekonvergenan *Turbulent Particle Swarm Optimization* (TPSO).
2. Mengimplementasikan *Fuzzy Logic Controller* pada algoritma *Fuzzy Adaptive Turbulent Particle Swarm Optimization* (FATPSO).
3. Menyelidiki hasil dan waktu komputasi algoritma *Fuzzy Adaptive Turbulent Particle Swarm Optimization* (FATPSO) apabila dibandingkan dengan hasil dan waktu komputasi algoritma *Turbulence Particle Swarm Optimization* (TPSO) dalam mengatasi kondisi stagnan yang dialami oleh algoritma *Particle Swarm Optimization* (PSO) pada penyelesaian masalah optimasi fungsi nonlinier.

BAB II
TINJAUAN PUSTAKA

2.1 Konsep Dasar Optimasi

Optimasi merupakan masalah memaksimumkan atau meminimumkan suatu besaran tertentu, yang disebut dengan fungsi tujuan. Fungsi tujuan bergantung pada sejumlah variabel yang tidak saling berhubungan atau saling bergantung melalui satu atau lebih kendala (Bronson, 1996).

Optimasi terbagi menjadi dua, optimasi linear dan nonlinear. Suatu permasalahan optimasi disebut nonlinear jika fungsi tujuan dan atau kendalanya mempunyai bentuk nonlinier. Optimasi nonlinear dapat dibagi menjadi empat berdasarkan kriteria permasalahannya yaitu optimasi nonlinear satu variabel tanpa kendala, multi variabel tanpa kendala, satu variabel dengan kendala, dan multi variabel dengan kendala (Luknanto, 2000).

Penulisan skripsi ini difokuskan pada optimasi nonlinear multi variabel tanpa kendala untuk kasus meminimumkan suatu fungsi tujuan, yang secara umum didefinisikan sebagai berikut. Diberikan fungsi tujuan *f*,

$$f : I \to \Re \text{, } I \subseteq \Re^n \text{ dan } I \text{ kompak,}$$

akan ditentukan $\boldsymbol{x}^* \in I$ yang memenuhi $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$, $\forall \boldsymbol{x} \in I$, dengan $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ (2.1)

(Bergh, 2006).

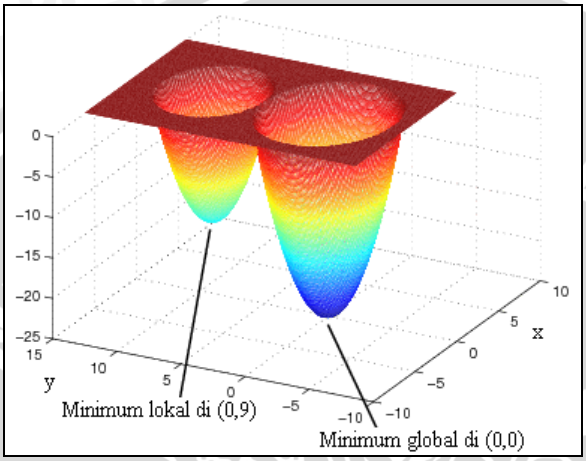
Definisi 2.1 (Minimum Lokal)

Diberikan $\boldsymbol{x}^* \in I \subseteq \Re^n$, $\boldsymbol{x}^*$ dikatakan sebagai titik minimum lokal fungsi f dalam I jika terdapat persekitaran $N(\boldsymbol{x}^*)$ di dalam I sedemikian sehingga $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$, $\forall\, \boldsymbol{x} \in N(\boldsymbol{x}^*)$ (Leader, 2004).

Definisi 2.2 (Minimum Global)

Diberikan $\boldsymbol{x}^* \in I \subseteq \Re^n$, $\boldsymbol{x}^*$ disebut sebagai titik minimum global fungsi f jika $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$, $\forall\, \boldsymbol{x} \in I$ (Bronshtein, dkk. 2007).

Untuk memberikan gambaran yang lebih jelas mengenai definisi minimum lokal dan minimum global disajikan ilustrasi pada Gambar 2.1, yang memperlihatkan grafik permukaan fungsi Twomins.



Gambar 2.1 Minimum global dan lokal

Definisi 2.3 (Fungsi Kontinu)

Misalkan fungsi $f:\Re^n \to \Re$ dan $\boldsymbol{x}_0 \in \Re^n$. Fungsi f dikatakan kontinu di titik $\boldsymbol{x}_0$ jika memenuhi kondisi

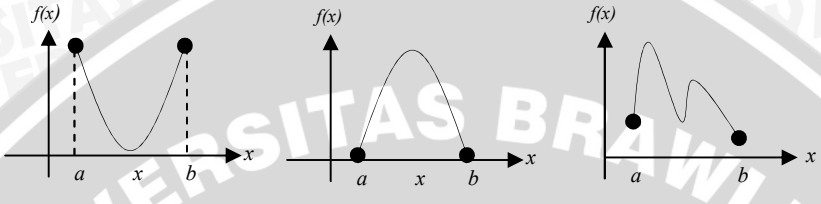
1. $f(\boldsymbol{x}_0)$ terdefinisi
2. $\lim_{\boldsymbol{x}\to\boldsymbol{x}_0} f(\boldsymbol{x})$ ada
3. $\lim_{\boldsymbol{x}\to\boldsymbol{x}_0} f(\boldsymbol{x}) = f(\boldsymbol{x}_0)$.

Jika ketiga kondisi di atas tidak terpenuhi, maka fungsi f dikatakan diskontinu di $\boldsymbol{x}_0$. Fungsi f dikatakan kontinu pada suatu himpunan terbuka $I \subseteq \Re^n$ jika f kontinu pada setiap titik di dalam I (Grossman, 1995).

Definisi 2.4 (Fungsi *Unimodal*)

Sebuah fungsi riil $f(x)$ dikatakan unimodal pada $U=[a,b]$ jika terdapat titik $x \in U$ yang tunggal, sedemikian sehingga $f(x)$ turun (naik) pada $[a,x]$ dan naik (turun) pada $[x,b]$ (Mathews dan Kurtis, 2004).

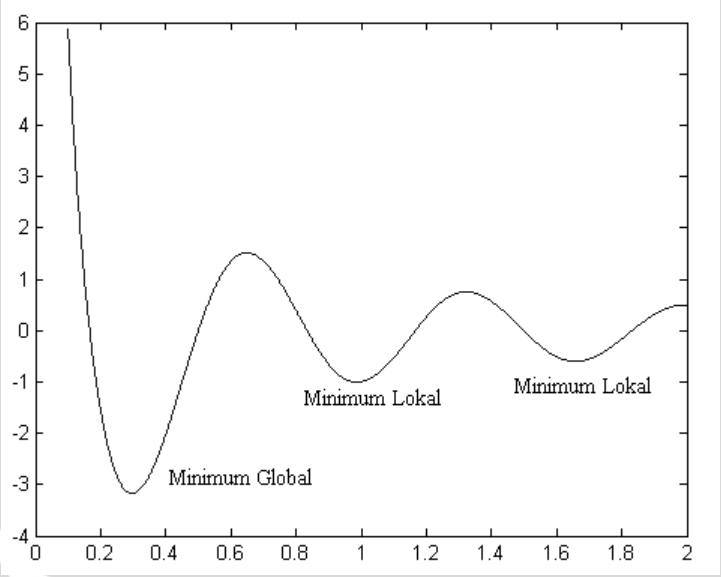
Pada Gambar 2.2 diperlihatkan contoh grafik fungsi unimodal dan bukan unimodal untuk mendiskripsikan definisi 2.4 dengan lebih jelas.



(a) **(b)** **(c)**

Gambar 2.2 (a) dan (b) Fungsi unimodal pada U (c) Fungsi yang bukan unimodal pada U

Fungsi $f(x)$ disebut fungsi multimodal jika $f(x)$ merupakan fungsi yang memiliki lebih dari satu titik minimum lokal atau maksimum lokal (Weise, 2009), seperti diperlihatkan pada gambar 2.3.



Gambar 2.3 Fungsi multimodal

2.2 Peubah Acak

Definisi 2.5 (Peubah Acak)

Peubah acak X adalah suatu fungsi yang memetakan seluruh anggota ruang contoh S ke himpunan bilangan nyata $\Re$, $X : S \rightarrow \Re$.

Definisi 2.6 (Ruang Contoh Kontinu)

Jika S adalah sebuah ruang contoh yang terdiri dari suatu selang (*interval*) atau gabungan dari beberapa selang maka S disebut ruang contoh kontinu.

Definisi 2.7 (Peubah Acak Kontinu)

Jika X adalah suatu peubah acak dengan ruang contoh kontinu S dan $f(x)$ adalah fungsi non negatif sedemikian sehingga $\int_S f(x)\,dx = 1$ dan berlaku

$$P(a < X \le b) = P(a < X < b) + P(X = b) = P(a < X < b)$$

yaitu

$$P(a < X < b) = \int_a^b f(x)\,dx$$

maka X disebut peubah acak kontinu dan $f(x)$ disebut fungsi kepadatan peluang (fkp) dari X.

Definisi 2.8 (Fungsi Sebaran Kumulatif)

Fungsi sebaran (kumulatif) suatu peubah acak kontinu X, $F(x)$ dinyatakan sebagai $F(x) = \int_{-\infty}^{x} f(t)\,dt$.

Dimisalkan $F(x)$ fungsi sebaran, fungsi kepadatan peluang $f(x)$ dinyatakan sebagai

$$f(x) = \frac{d}{dx} F(x), \quad -\infty < x < \infty .$$

Salah satu contoh peubah acak kontinu adalah peubah acak seragam.

Definisi 2.9 (Peubah Acak Seragam)

Peubah acak X dikatakan menyebar secara seragam pada interval (α, β) jika fungsi kepadatan peluangnya adalah

$$f(x) = \begin{cases} \dfrac{1}{\beta - \alpha} & \alpha < x < \beta \\ 0 & x \text{ lainnya} \end{cases} \tag{2.2}$$

dengan fungsi sebaran

$$F(x) = \begin{cases} 0 & x \le \alpha \\ \dfrac{x - \alpha}{\beta - \alpha} & \alpha < x < \beta \\ 1 & x \ge \beta \end{cases}$$

(Dajan, 1986).

Definisi 2.10 (Konvergensi Barisan)

Misalkan $\langle \boldsymbol{x}_n \rangle$ barisan di dalam $\Re^n$. Barisan $\langle \boldsymbol{x}_n \rangle$ dikatakan konvergen ke $\boldsymbol{x}$ atau berlimit $\boldsymbol{x}$ dan ditulis sebagai

$$\lim_{n \to +\infty} \boldsymbol{x}_n = \boldsymbol{x} \tag{2.3}$$

apabila untuk setiap bilangan positif ε, terdapat bilangan positif $N \in \mathbf{N}$ sedemikian sehingga

$$|\boldsymbol{x}_n - \boldsymbol{x}| < \varepsilon, \ \forall n \ge N \ . \tag{2.4}$$

Suatu barisan yang tidak konvergen ke suatu bilangan $\boldsymbol{x}$ yang berhingga dikatakan divergen (Purcell, 1994).

Definisi 2.11 (Konvergensi Barisan dalam Bentuk Peluang)

Misalkan $\langle X_n \rangle$ adalah barisan peubah acak dan X adalah suatu peubah acak. Barisan $\langle X_n \rangle$ dikatakan konvergen ke X jika untuk setiap $\varepsilon > 0$,

$$\lim_{n \to \infty} P\left(|X_n - X| < \varepsilon\right) = 1 \tag{2.5}$$

(Abraham dan Liu, 2005).

Definisi 2.12 (Konvergensi Barisan dengan Peluang 1)

Apabila $\langle X_n \rangle$ adalah barisan peubah acak dan X adalah suatu peubah acak, maka barisan $\langle X_n \rangle$ dikatakan konvergen ke X dengan peluang 1 jika

$$P\left(\lim_{n\to\infty} X_n = X \right) = 1$$

atau untuk setiap $\varepsilon > 0$,

$$P\left(\limsup_{n\to\infty} \{|X_n - X| \geq \varepsilon\} \right) = 0 \qquad (2.6)$$

(Abraham dan Liu, 2005).

Lemma 2.1 (*Borel-Cantelli Lemma*)

Misalkan $\langle A_n \rangle$ adalah barisan kejadian berdistribusi tertentu dan misalkan A adalah kejadian yang terdiri dari barisan kejadian berhingga $\langle A_n \rangle$ dengan $n = 1, 2, \ldots$. Jika

$$\sum_{n=1}^{\infty} P(A_n) < \infty$$

maka

$$P\left(\limsup_{n\to\infty} A_n \right) = 0 \qquad (2.7)$$

dan jika kejadiannya saling lepas dan

$$\sum_{n=1}^{\infty} P(A_n) = \infty$$

maka

$$P\left(\limsup_{n\to\infty} A_n \right) = 1. \qquad (2.8)$$

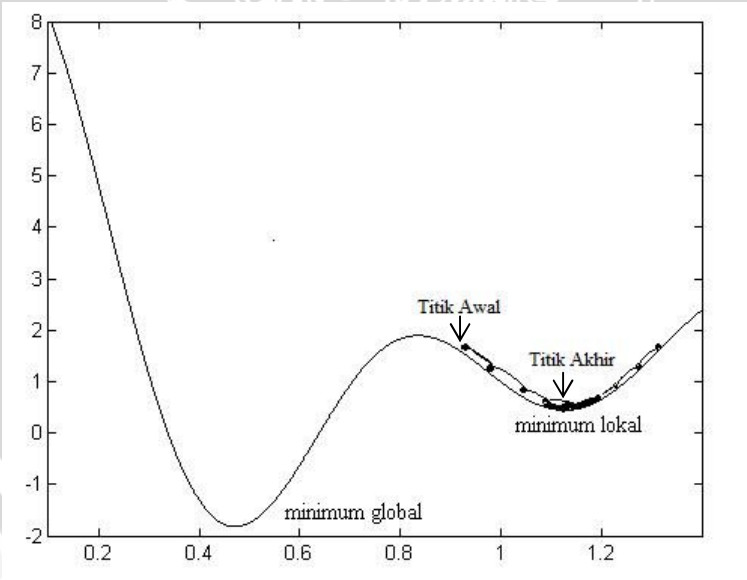
Bukti:

Misalkan A_n adalah kejadian ke-n dengan $n = 1, 2, \ldots$ dan misalkan untuk setiap n, $P(A_n) = 1$ jika kejadian tersebut pasti terjadi dan $P(A_n) = 0$ jika kejadian tersebut tidak mungkin terjadi.

Misalkan $\sum_{n=1}^{\infty} P(A_n) < \infty$, berarti terdapat nilai $P(A_k) = 0$ untuk suatu nilai $k \geq n$ yang menyebabkan $P\left(\bigcap_{n=1}^{\infty}\bigcup_{k\geq n}^{\infty} A_k\right) = 0$ atau $P\left(\limsup_{n\to\infty} A_n\right) = 0$. Akan tetapi, apabila $\sum_{n=1}^{\infty} P(A_n) = \infty$ dan kejadiannya saling lepas, yaitu nilai $P(A_k) = 1$ untuk suatu nilai $k \geq n$, menyebabkan $P\left(\bigcap_{n=1}^{\infty}\bigcup_{k\geq n}^{\infty} A_k\right) = 1$ atau $P\left(\limsup_{n\to\infty} A_n\right) = 1$.

Definisi 2.13 (Konvergen Prematur)

Suatu proses optimasi dikatakan berada pada kondisi konvergen prematur menuju titik minimum lokal jika proses optimasi tersebut tidak memungkinkan untuk mengeksplorasi daerah pencarian yang lain padahal mungkin terdapat daerah lain yang memiliki solusi yang lebih baik (lihat Gambar 2.4) (Weise, 2009).



Gambar 2.4 Kondisi konvergen premature

2.3 Relasi Rekurensi

Sebuah relasi rekurensi dari sebuah fungsi numerik x, yaitu fungsi yang domainnya berupa bilangan cacah dan kodomainnya berupa bilangan riil, secara umum ditulis sebagai

$$C_0 x_n + C_1 x_{n-1} + \ldots + C_k x_{n-k} = \Psi(n) \qquad (2.9)$$

dimana koefisien C_i, untuk $i = 0,1,2,\ldots,k$ adalah konstan dan $\Psi(n)$ adalah sebuah fungsi numerik dengan variabel n. Relasi rekurensi tersebut dikatakan relasi rekurensi linier berderajat k, jika $C_0 \neq 0$ dan $C_k \neq 0$.

2.3.1 Solusi Relasi Rekurensi

Sebuah relasi rekurensi memiliki dua macam solusi, yaitu:

1. solusi umum, yang merupakan jumlahan dari solusi homogen dan nonhomogen. Misalkan $x_n^{(h)}$ adalah solusi homogen dan misalkan $x_n^{(nh)}$ adalah solusi nonhomogen, maka solusi umum untuk relasi rekurensi yang dimaksud adalah

$$x_n = x_n^{(h)} + x_n^{(nh)}, \qquad (2.10)$$

2. solusi khusus atau partikular, yaitu solusi yang memenuhi syarat awal yang diberikan.

2.3.2 Solusi Umum Relasi Rekurensi Homogen

Bila $\Psi(n)$ pada persamaan (2.9) bernilai nol, maka diperoleh relasi rekurensi

$$C_0 x_n + C_1 x_{n-1} + \ldots + C_k x_{n-k} = 0. \qquad (2.11)$$

Relasi rekurensi yang demikian disebut relasi rekurensi homogen dan solusinya dinamakan solusi homogen.

Solusi homogen sebuah relasi rekurensi dengan koefisien konstan dinyatakan dalam bentuk $A\alpha^n$, dimana α adalah akar karakteristik dan A adalah konstanta yang harganya akan ditentukan kemudian untuk memenuhi syarat awal yang diberikan.

Bila $x_n = A\alpha^n$ disubstitusikan ke persamaan (2.11) maka diperoleh

$$C_0 A\alpha^n + C_1 A\alpha^{n-1} + \ldots + C_k A\alpha^{n-k} = 0.$$

Penyederhanaan persamaan tersebut menghasilkan

$$C_0\alpha^n + C_1\alpha^{n-1} + \ldots + C_k\alpha^{n-k} = 0. \qquad (2.12)$$

Persamaan (2.12) merupakan persamaan karakteristik relasi rekurensi yang diberikan dan penyelesaiannya diklasifikasikan menjadi tiga macam, yaitu:

1. Jika persamaan karakteristik tersebut memiliki sebanyak *k* akar karakteristik berbeda $(\alpha_1 \neq \alpha_2 \neq \ldots \neq \alpha_k)$, maka solusi homogen relasi rekurensi yang dimaksud dinyatakan dalam bentuk
$$x_n^{(h)} = A_1{\alpha_1}^n + A_2{\alpha_2}^n + \ldots + A_k{\alpha_k}^n$$
dimana α_i adalah akar karakteristik dari persamaan karakteristik yang diperoleh, sedangkan A_i adalah koefisien yang akan ditentukan untuk memenuhi syarat awal yang diketahui.

2. Jika terdapat akar karakteristik α_i yang berulang sebanyak *m* kali, maka kontribusi yang diberikan pada solusi homogen berbentuk
$$\left(A_1 n^{m-1} + A_2 n^{m-2} + \ldots + A_{m-2}n^2 + A_{m-1}n + A_m\right){\alpha_i}^n$$
dengan A_j, $j = 1,2,..,m$ adalah koefisien yang nantinya akan ditentukan untuk memenuhi syarat awal yang diberikan.

3. Jika terdapat akar karakteristik yang berupa akar kompleks, misalkan
$\alpha_1 = u + \mathrm{i}\,v$ dan $\alpha_2 = u - \mathrm{i}\,v$ atau
$\alpha_1 = \eta(\cos\theta + \mathrm{i}\sin\theta)$ dan $\alpha_2 = \eta(\cos\theta - \mathrm{i}\sin\theta)$,
dengan $\theta = arc\tan\left(\frac{v}{u}\right)$ dan $\eta = \sqrt{u^2 + v^2}$. Karena
${\alpha_1}^n = \eta^n(\cos(n\theta) + i\sin(n\theta))$ dan ${\alpha_2}^n = \eta^n(\cos(n\theta) - i\sin(n\theta))$,
sehingga bentuk kontribusi yang diberikan adalah
$A_1\eta^n(\cos(n\theta) + i\sin(n\theta)) + A_2\eta^n(\cos(n\theta) - i\sin(n\theta))$
$= (A_1 + A_2)\eta^n\cos(n\theta) + (A_1 - A_2)\eta^n i\sin(n\theta)$
atau $\eta^n(A\cos(n\theta) + B\sin(n\theta))$, dengan $A = A_1 + A_2$ dan $B = (A_1 - A_2)i$, dimana A_1 dan A_2 adalah koefisien yang ditentukan berdasarkan syarat awal yang diberikan.

2.3.3 Solusi Umum Relasi Rekurensi Nonhomogen

Metode yang digunakan untuk menentukan solusi umum sebuah relasi rekurensi linier dengan $\Psi(n) \neq 0$ adalah metode koefisien tak tentu. Untuk menerapkan metode koefisien tak tentu ini perlu dipilih solusi coba yang sesuai. Bentuk solusi coba disesuaikan berdasarkan bentuk $\Psi(n)$, dan dapat dilihat pada Tabel 2.1 berikut.

Tabel 2.1 Tabel Solusi Coba

$\Psi(n)$	Bentuk solusi coba
$b_m n^m + \ldots + b_1 n + b_0$	$B_m n^m + \ldots + B_1 n + B_0$
$k^n \left(b_m n^m + \ldots + b_1 n + b_0\right)$	$k^n \left(B_m n^m + \ldots + B_1 n + B_0\right)$

(Liu, 1986)

2.4 *Particle Swarm Optimization* (PSO)

2.4.1 Algoritma *Particle Swarm Optimization* (PSO)

Algoritma *Particle Swarm Optimization* (PSO) diperkenalkan oleh Eberhart dan Kennedy pada tahun 1995. Algoritma ini merupakan algoritma optimasi global dan tidak membutuhkan perhitungan *gradient* fungsi sehingga mudah diimplementasikan.

Algoritma PSO terdiri dari populasi partikel, dimana setiap partikel merupakan kandidat solusi untuk masalah optimasi. Misalkan *s* adalah ukuran *swarm* (populasi) yang menyatakan banyaknya partikel dalam populasi dan *n* menyatakan dimensi masalah optimasi. Setiap partikel dapat dianggap sebagai obyek dengan beberapa karakteristik yang dinyatakan dengan simbol berikut:

$\boldsymbol{x}$: posisi partikel,
$\boldsymbol{v}$: kecepatan partikel, dan
$\boldsymbol{y}$: posisi terbaik partikel.

Misalkan masalah optimasi yang diselesaikan adalah persamaan (2.1), dengan domain $I = \left[-\boldsymbol{x}_{maks}, \boldsymbol{x}_{maks}\right]^n$ yaitu $I = \left\{(x_1, x_2, \ldots, x_n) \mid x_i \in [-x_{i_{maks}}, x_{i_{maks}}] \ , i = 1,2,\ldots,n\right\}$. Dalam algoritma PSO, posisi partikel dinyatakan sebagai vektor $\boldsymbol{x}$ dengan

$\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$, kecepatan partikel adalah vektor perpindahan posisi $\boldsymbol{x}$ dan $\boldsymbol{x}_{maks}$ adalah posisi maksimum partikel.

Posisi terbaik partikel $\boldsymbol{y}$ adalah posisi terbaik yang pernah ditemukan oleh partikel dan berhubungan dengan nilai *fitness* terbaik yang pernah diperoleh. Untuk masalah optimasi yang dibahas dalam skripsi ini, suatu posisi dengan nilai fungsi terkecil memiliki nilai *fitness* yang tinggi. Misalkan t menyatakan iterasi, maka persamaan untuk memperbaiki nilai posisi terbaik partikel dinyatakan sebagai

$$\boldsymbol{y}(t+1) = \begin{cases} \boldsymbol{y}(t) & \text{jika } f(\boldsymbol{x}(t+1)) \geq f(\boldsymbol{y}(t)) \\ \boldsymbol{x}(t+1) & \text{jika } f(\boldsymbol{x}(t+1)) < f(\boldsymbol{y}(t)) \end{cases} \tag{2.13}$$

dengan $\boldsymbol{x}(t+1)$ adalah posisi partikel yang telah diperbaiki. Posisi terbaik yang pernah ditemukan oleh anggota populasi dinyatakan sebagai $\hat{\boldsymbol{y}}$, yaitu

$\hat{\boldsymbol{y}}(t) \in \{\boldsymbol{y}_0(t), \boldsymbol{y}_1(t), \ldots, \boldsymbol{y}_s(t)\}$ sedemikian sehingga

$$f(\hat{\boldsymbol{y}}(t)) = \min\{f(\boldsymbol{y}_0(t)), f(\boldsymbol{y}_1(t)), \ldots, f(\boldsymbol{y}_s(t))\}. \tag{2.14}$$

Algoritma PSO menggunakan dua vektor peubah acak, yaitu $\boldsymbol{R}_1$ dan $\boldsymbol{R}_2$ yang berdistribusi seragam (0,1). Nilai ini dibatasi oleh konstanta c_1 dan c_2 yang dinamakan koefisien percepatan. Persamaan untuk memperbaiki kecepatan adalah

$$\begin{aligned} v_{i,j}(t+1) = w v_{i,j}(t) + c_1 R_{1,j}(t)\left[y_{i,j}(t) - x_{i,j}(t)\right] + \\ c_2 R_{2,j}(t)\left[\hat{y}_{i,j}(t) - x_{i,j}(t)\right] \end{aligned} \tag{2.15}$$

dengan w adalah bobot inersia, $R_{1,j}$ adalah elemen ke-j dari vektor $\boldsymbol{R}_1$, $R_{2,j}$ adalah elemen ke-j dari vektor $\boldsymbol{R}_2$, $i = 1, 2, \ldots, s$ dan $j = 1, 2, \ldots, n$. Nilai kecepatan $\boldsymbol{v}$ dibatasi oleh kecepatan maksimum $\boldsymbol{v}_{\max}$ yaitu $\boldsymbol{v}_{\max} = k \times \boldsymbol{x}_{\max}$ dengan $0.1 \leq k \leq 1$ untuk membatasi pergerakan partikel agar tidak keluar dari domain I. Posisi partikel diperbaiki menggunakan nilai kecepatan partikel baru, yaitu

$$\boldsymbol{x}_i(t+1) = \boldsymbol{x}_i(t) + \boldsymbol{v}_i(t+1) \tag{2.16}$$

dengan

$$\boldsymbol{x}_i(t+1) = \begin{cases} \boldsymbol{x}_i(t+1) & \text{jika } abs(\boldsymbol{x}_i(t+1)) \leq \boldsymbol{x}_{maks} \\ sign[\boldsymbol{x}_i(t+1)]\boldsymbol{x}_{maks} & \text{jika } abs(\boldsymbol{x}_i(t+1)) > \boldsymbol{x}_{maks} \end{cases},$$

$i = 1, 2, \ldots, s$ dan fungsi *sign* adalah fungsi tanda.

Berdasarkan persamaan (2.15) diketahui bahwa perbaikan kecepatan dipengaruhi oleh tiga faktor, yaitu:

1. Vektor $w\boldsymbol{v}(t)$, yang merupakan vektor kecepatan dari iterasi sebelumnya. Kecepatan ini dikontrol oleh bobot inersia (w).
2. Vektor $c_1\boldsymbol{R}_1(t)\left[\boldsymbol{y}_i(t)-\boldsymbol{x}_i(t)\right]$, yang merupakan komponen individu. Komponen ini menyatakan kemampuan yang dimiliki partikel untuk bergerak berdasarkan informasi yang diperolehnya dan direpresentasikan sebagai jarak partikel ke posisi terbaik yang pernah diperoleh.
3. Vektor $c_2\boldsymbol{R}_2(t)\left[\hat{\boldsymbol{y}}_i(t)-\boldsymbol{x}_i(t)\right]$, yang merupakan komponen sosial. Komponen ini menyatakan jarak partikel ke posisi terbaik yang pernah diperoleh anggota populasi dan merepresentasikan kemampuan partikel untuk bergerak secara berkelompok.

(Bergh, 2006)

2.4.2 Parameter Algoritma PSO

Terdapat beberapa parameter dalam algoritma PSO yang berpengaruh terhadap keberhasilan algoritma, yaitu:

1. Bobot inersia (w).
   Konstanta ini mengatur seberapa besar kecepatan iterasi sebelumnya mempengaruhi kecepatan iterasi berikutnya. Konstanta ini sangat mempengaruhi konvergensi algoritma. Nilai w yang besar memfasilitasi pencarian global sedangkan nilai w yang kecil memfasilitasi pencarian lokal.
2. Koefisien percepatan (c_1 dan c_2).
   Kedua koefisien ini tidak mempengaruhi kekonvergenan algoritma, tetapi pemilihan nilai yang tepat dapat meningkatkan kecepatan konvergensi dan dapat menyebabkan algoritma tidak mudah terjebak dalam optimum lokal.
3. Ukuran *swarm* (populasi).
   Ukuran populasi merupakan parameter yang berfungsi untuk menentukan jumlah partikel yang berada pada populasi. Semakin banyak dan beragamnya partikel akan memberikan peluang lebih besar untuk menemukan partikel yang memiliki nilai *fitness* terbaik. Jika partikel yang ada terlalu sedikit, algoritma PSO hanya memiliki sedikit kemungkinan untuk menemukan hasil

terbaik. Namun jika terdapat terlalu banyak partikel, dapat menyebabkan menurunnya kecepatan konvergensi.

4. Jumlah iterasi.
   Jumlah iterasi mempunyai andil besar dalam menemukan hasil yang lebih baik, karena semakin banyak jumlah iterasi maka semakin baik nilai *fitness* yang diperoleh. Tetapi tidak berarti semakin banyak jumlah iterasi hasil yang diperoleh selalu lebih baik, karena pada suatu saat nilai *fitness* semua partikel menjadi sama.

(Parsopoulos, 2002)

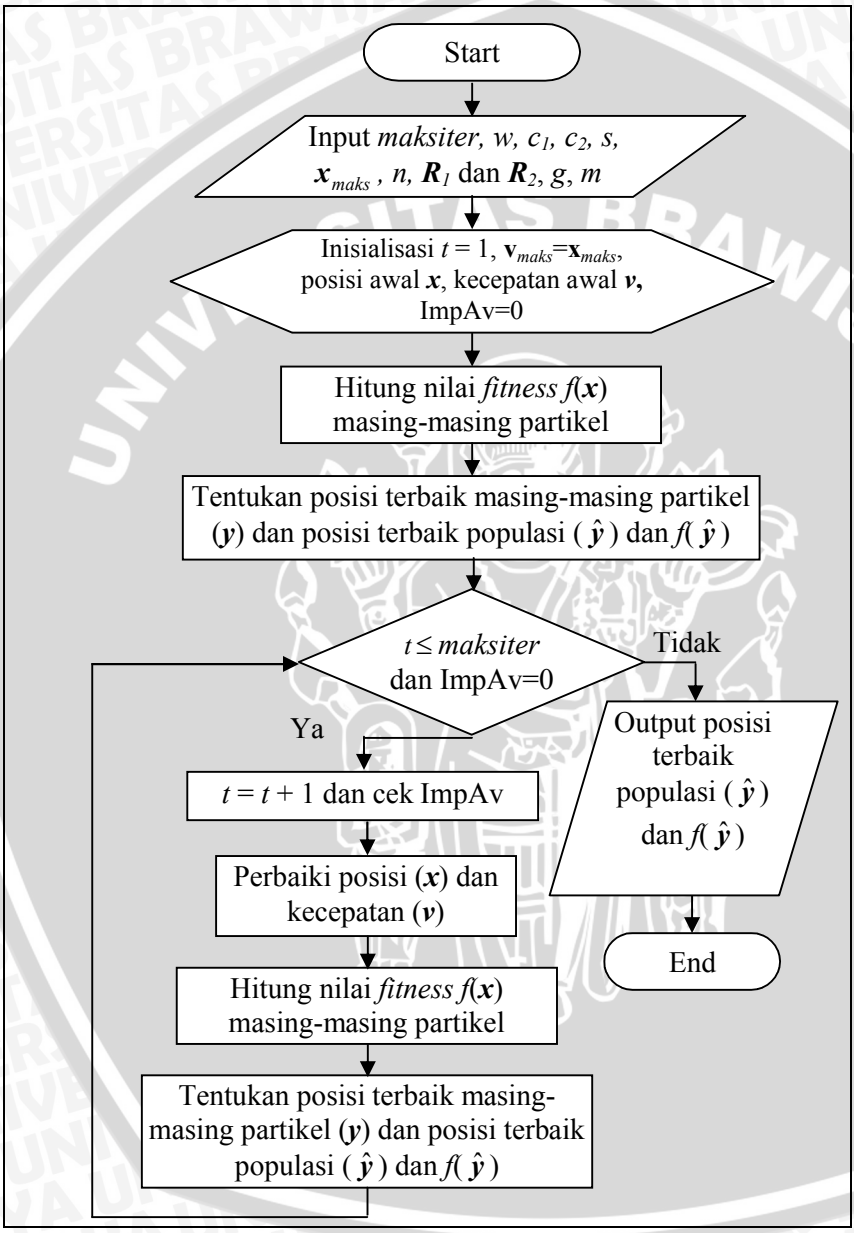
2.4.3 Kriteria Penghentian Iterasi Algoritma PSO

Kriteria penghentian iterasi tergantung pada masalah optimasi yang dihadapi. Apabila informasi tentang nilai *fitness* yang dijadikan tujuan diketahui, maka kriteria yang digunakan berupa batasan nilai kesalahan antara nilai *fitness* tujuan dan nilai *fitness* hasil perhitungan. Akan tetapi, apabila informasi tersebut tidak diketahui, maka kriteria yang digunakan adalah maksimum iterasi, keseragaman partikel atau peningkatan rata-rata nilai *fitness* (ImpAv).

Kriteria penghentian iterasi menggunakan keseragaman partikel yaitu suatu kriteria yang akan bernilai benar jika semua partikel memiliki karakteristik yang sama. Dalam menyelesaikan masalah optimasi, kriteria ini jarang digunakan karena apabila pemilihan paramaternya tidak tepat dapat menyebabkan iterasi tidak akan pernah berhenti. Hal ini dikarenakan, dalam algoritma PSO tidak terjadi pengelompokkan partikel secara keseluruhan (tidak seragam). Maksimum iterasi adalah kriteria penghentian yang paling cocok untuk menyelesaikan masalah optimasi, karena kriteria ini tidak dipengaruhi oleh parameter-parameter algoritma (Bergh, 2006). Kriteria menggunakan peningkatan rata-rata nilai *fitness* (ImpAv) adalah kriteria yang menghentikan iterasi jika untuk g iterasi peningkatan rata-rata nilai *fitness* kurang dari suatu batasan nilai m (Zielinski, dkk. 2005).

Dalam skripsi ini, kriteria yang digunakan adalah maksimum iterasi dan peningkatan rata-rata nilai *fitness* (ImpAv).

Prosedur untuk mengimplementasikan PSO dapat dilihat pada Gambar 2.5.



Gambar 2.5 Diagram alir algoritma PSO

2.4.4 Kekonvergenan Algoritma PSO

Algoritma PSO tidak menjamin solusi yang diperoleh adalah solusi optimum. Algoritma PSO berhenti pada suatu kondisi jika memenuhi kriteria yang telah ditentukan sebelumnya. Dalam hal ini, proses berhenti setelah iterasi berulang sebanyak maksimum iterasi dan memenuhi kriteria penghentian ImpAv. Konvergen merupakan kondisi yang dicapai pada saat populasi kehilangan keanekaragaman, artinya sebagian besar partikel dalam populasi mempunyai karakteristik yang sama.

Parameter bobot inersia *w* dalam algoritma PSO adalah parameter yang mempengaruhi konvergensi. Oleh karena itu, untuk menjamin algoritma ini konvergen perlu dicari kisaran nilai yang sesuai. Langkah yang dilakukan adalah mencari bentuk eksplisit dari posisi partikel ke-*i* dalam suatu dimensi (misal dimensi ke-*j*) pada waktu ke-*t* (x_t) dan kemudian diselidiki posisi partikel pada waktu $t \to \infty$.

Dengan mensubstitusikan persamaan (2.15) ke dalam persamaan (2.16) dan dengan mengasumsikan $\phi_1 = c_1 R_{1,j}(t)$, $\phi_2 = c_2 R_{2,j}(t)$, $y_{i,j}(t) = y$ dan $\hat{y}_{i,j}(t) = \hat{y}$ konstan selama iterasi ke-*t* berjalan, diperoleh

$$x_{t+1} = (1 - \phi_1 - \phi_2)x_t + \phi_1 y + \phi_2 \hat{y} + w v_t \ .$$

Dalam persamaan (2.16), diketahui bahwa $x_t = x_{t-1} + v_t$ sehingga diperoleh

$$x_{t+1} - (1 + w - \phi_1 - \phi_2)x_t + w x_{t-1} = \phi_1 y + \phi_2 \hat{y} \qquad (2.17)$$

yang merupakan relasi rekurensi nonhomogen.

Untuk menentukan solusi relasi rekurensi nonhomogen tersebut, langkah yang dilakukan adalah sebagai berikut:

1. Mencari solusi umum relasi rekurensi nonhomogen.

   Misalkan $x_{t-1} = \alpha^t$, maka persamaan (2.17) menjadi

   $$\alpha^{t+2} - (1 + w - \phi_1 - \phi_2)\alpha^{t+1} + w\alpha^t = \phi_1 y + \phi_2 \hat{y} \ .$$

   Bentuk homogennya adalah

   $$\alpha^{t+2} - (1 + w - \phi_1 - \phi_2)\alpha^{t+1} + w\alpha^t = 0 \ ,$$

   dengan persamaan karakteristik

   $$\alpha^2 - (1 + w - \phi_1 - \phi_2)\alpha + w = 0 \ .$$

Solusi persamaan karakteristik tersebut adalah

$$\alpha_1 = \frac{1 + w - \phi_1 - \phi_2 + \gamma}{2}$$
$$\alpha_2 = \frac{1 + w - \phi_1 - \phi_2 - \gamma}{2} \qquad (2.18)$$

dengan $\gamma = \sqrt{(1 + w - \phi_1 - \phi_2)^2 - 4w}$.

Karena $\alpha_1 \neq \alpha_2$, maka solusi relasi rekurensi homogen berbentuk

$$x_t^{(h)} = k_1 {\alpha_1}^t + k_2 {\alpha_2}^t .$$

Nilai $\psi(n) = \phi_1 y + \phi_2 \hat{y}$ adalah suatu konstanta sehingga bentuk solusi coba juga berupa konstanta, misalkan A_0. Dengan demikian, solusi umum relasi rekurensi nonhomogen berbentuk

$$x_t = k_1 {\alpha_1}^t + k_2 {\alpha_2}^t + A_0 . \qquad (2.19)$$

2. Mencari solusi khusus relasi rekurensi nonhomogen.

Syarat awal x_0, x_1 dan x_2 diperoleh dari solusi umum relasi rekurensi nonhomogen dengan memasukkan nilai t = 0, 1, dan 2 masing-masing untuk x_0, x_1 dan x_2, sehingga diperoleh sebuah sistem persamaan linier

$$x_0 = k_1 + k_2 + A_0$$
$$x_1 = k_1 \alpha_1 + k_2 \alpha_2 + A_0$$
$$x_2 = k_1 {\alpha_1}^2 + k_2 {\alpha_2}^2 + A_0 .$$

Dengan melakukan eliminasi diperoleh

$$A_0 = \frac{\alpha_1 \alpha_2 x_0 - x_1(\alpha_1 + \alpha_2) + x_2}{(\alpha_1 - 1)(\alpha_2 - 1)}$$
$$k_1 = \frac{\alpha_2 (x_0 - x_1) - x_1 + x_2}{(\alpha_1 - \alpha_2)(\alpha_1 - 1)} \qquad (2.20)$$
$$k_2 = \frac{\alpha_1 (x_1 - x_0) + x_1 - x_2}{(\alpha_1 - \alpha_2)(\alpha_2 - 1)} .$$

Karena $x_2 = (1 + w - \phi_1 - \phi_2) x_1 - w x_0 + \phi_1 y + \phi_2 \hat{y}$, $\alpha_1 - \alpha_2 = \gamma$ dan $w = \frac{(1 + w - \phi_1 - \phi_2)^2 - \gamma^2}{4}$, maka persamaan (2.20) menjadi

$$A_0 = \frac{\phi_1 y + \phi_2 \hat{y}}{\phi_1 + \phi_2}$$
$$k_1 = \frac{\alpha_2 (x_0 - x_1) - x_1 + x_2}{\gamma(\alpha_1 - 1)} \tag{2.21}$$
$$k_2 = \frac{\alpha_1 (x_1 - x_0) + x_1 - x_2}{\gamma(\alpha_2 - 1)}.$$

Dengan demikian solusi khusus relasi rekurensi nonhomogen berbentuk

$$x_t = \frac{\alpha_2 (x_0 - x_1) - x_1 + x_2}{\gamma(\alpha_1 - 1)} \alpha_1{}^t + \frac{\alpha_1 (x_1 - x_0) + x_1 - x_2}{\gamma(\alpha_2 - 1)} \alpha_2{}^t + \frac{\phi_1 y + \phi_2 \hat{y}}{\phi_1 + \phi_2} \tag{2.22}$$

yang merupakan bentuk eksplisit dari posisi partikel pada waktu ke-*t*.

2.4.4.1 Penentuan Kisaran Nilai Bobot Inersia (*w*)

Dari bentuk eksplisit posisi partikel pada waktu ke-*t* diketahui bahwa, barisan $\langle x_t \rangle$ konvergen ke $\frac{\phi_1 y + \phi_2 \hat{y}}{\phi_1 + \phi_2}$ atau ditulis

$$\lim_{t \to +\infty} x_t = \frac{\phi_1 y + \phi_2 \hat{y}}{\phi_1 + \phi_2} \tag{2.23}$$

jika $\lim_{t \to +\infty} \alpha_1{}^t = \lim_{t \to +\infty} \alpha_2{}^t = 0$. Nilai $\lim_{t \to +\infty} \alpha_1{}^t$ dan $\lim_{t \to +\infty} \alpha_2{}^t$ akan bernilai nol jika $|\alpha_1| < 1$ dan $|\alpha_2| < 1$.

Berdasarkan persamaan (2.18) diketahui bahwa nilai α_1 dan α_2 bergantung pada nilai γ, sehingga untuk menentukan kisaran nilai parameter *w* dapat dibagi menjadi 3 kasus, yaitu:

1. Nilai γ bernilai riil, yaitu $(1 + w - \phi_1 - \phi_2)^2 - 4w > 0$, sehingga $1 + w - \phi_1 - \phi_2 > 2\sqrt{w}$ atau $1 + w - \phi_1 - \phi_2 < -2\sqrt{w}$ yang mungkin dapat menyebabkan
   - $\alpha_1 = \frac{1 + w - \phi_1 - \phi_2 + \gamma}{2} > 1$, karena $1 + w - \phi_1 - \phi_2 > 2\sqrt{w}$ dan $\gamma > 0$ atau

- $\alpha_2 = \dfrac{1+w-\phi_1-\phi_2-\gamma}{2} < -1$, karena $1+w-\phi_1-\phi_2 < -2\sqrt{w}$ dan $\gamma > 0$.

Nilai ini menyebabkan algoritma PSO tidak konvergen.

2. Nilai γ bernilai nol, yaitu $(1+w-\phi_1-\phi_2)^2 - 4w = 0$, sehingga nilai $\alpha_1 = \alpha_2 = \dfrac{1+w-\phi_1-\phi_2}{2}$, yang menyebabkan solusi umum relasi rekurensi nonhomogennya menjadi

$$x_t = k_1\alpha_1{}^t + k_2 t\alpha_2{}^t + A_0.$$

Adanya faktor pengali t dalam solusi eksplisit x_t dapat menyebabkan $\lim_{t\to+\infty} x_t$ tidak ada. Solusi seperti ini juga dapat menyebabkan algoritma PSO tidak konvergen.

3. Nilai γ bernilai kompleks, yaitu $(1+w-\phi_1-\phi_2)^2 - 4w < 0$, sehingga

$$\alpha_1 = \frac{1+w-\phi_1-\phi_2+i\sqrt{4w-(1+w-\phi_1-\phi_2)^2}}{2}$$

$$\alpha_2 = \frac{1+w-\phi_1-\phi_2-i\sqrt{4w-(1+w-\phi_1-\phi_2)^2}}{2}.$$

Nilai α_1 dan α_2 dapat dinyatakan sebagai

$\alpha_1 = \eta(\cos\theta + \mathrm{i}\sin\theta)$ dan $\alpha_2 = \eta(\cos\theta - \mathrm{i}\sin\theta)$

dengan $\eta = \sqrt{\left(\dfrac{1+w-\phi_1-\phi_2}{2}\right)^2 + \dfrac{4w-(1+w-\phi_1-\phi_2)^2}{4}} = \sqrt{w}$

dan $\theta = \arg(\alpha_1) = \arg(\alpha_2)$, sehingga $\alpha_1{}^t = \eta^t(\cos(\theta t) + i\sin(\theta t))$ dan $\alpha_2{}^t = \eta^t(\cos(\theta t) - i\sin(\theta t))$.

Karena $\lim_{t\to+\infty} \alpha_1{}^t$ dan $\lim_{t\to+\infty} \alpha_2{}^t$ akan sama dengan 0 jika $\eta < 1$ dan $\eta = \sqrt{w}$, maka nilai $\sqrt{w}$ harus kurang dari 1. Dengan

demikian nilai $\sqrt{w} < 1$ menyebabkan $\lim_{t \to +\infty} \alpha_1{}^t = \lim_{t \to +\infty} \alpha_2{}^t = 0$,

sehingga $\lim_{t \to +\infty} x_t = \dfrac{\phi_1 y + \phi_2 \hat{y}}{\phi_1 + \phi_2}$ atau barisan $\langle x_t \rangle$ konvergen.

2.4.4.2 Penentuan Kisaran Nilai Koefisien Percepatan (c_1 dan c_2)

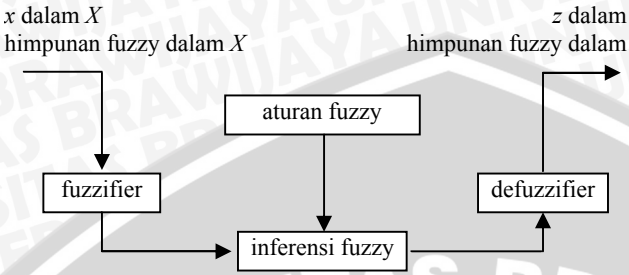
Pada kasus 3 diketahui bahwa nilai $\sqrt{w} < 1$ akan menjamin barisan $\langle x_t \rangle$ konvergen, sehingga dengan menggunakan kondisi pada kasus 3 yaitu $(1 + w - \phi_1 - \phi_2)^2 - 4w < 0$ dapat ditentukan kisaran nilai parameter ϕ_1 dan ϕ_2. Dari $(1 + w - \phi_1 - \phi_2)^2 - 4w < 0$ diperoleh hubungan $1 + w - 2\sqrt{w} < (\phi_1 + \phi_2) < 1 + w + 2\sqrt{w}$. Tetapi karena nilai $\sqrt{w} < 1$, maka kisaran nilai $\phi_1 + \phi_2$ adalah $0 < \phi_1 + \phi_2 < 4$. Karena $\phi_1 = c_1 R_{1,j}(t)$ dan $\phi_2 = c_2 R_{2,j}(t)$, sedangkan $R_1(t) \leq 1$ dan $R_2(t) \leq 1$ dan apabila dimisalkan $R_{1,j}(t) = R_{2,j}(t) = 1$ maka $0 < c_1 + c_2 < 4$.

2.5 *Fuzzy*

2.5.1 Pengertian Sistem *Fuzzy*

Konsep logika *fuzzy* atau disebut juga dengan logika samar pertama kali diperkenalkan oleh Lofti A. Zadeh dari *University of California*, Barkeley pada tahun 1965, yang merupakan alternatif dari logika konvensional/klasik. Ia berpendapat bahwa logika benar dan salah dari logika boolean tidak dapat mengatasi masalah gradasi yang berada pada dunia nyata. Untuk mengatasi masalah gradasi yang tidak berhingga tersebut, Zadeh mengembangkan sebuah himpunan *fuzzy*.

Sistem *fuzzy* adalah sebuah sistem yang dibangun dari seperangkat aturan berdasarkan logika *fuzzy*. Salah satu aplikasi dari sistem *fuzzy* adalah perancangan pengontrol berbasis logika *fuzzy* yang disebut sebagai *Fuzzy Logic Controller*. Sistem *fuzzy* dapat mengolah masukan yang berupa titik *crisp* menjadi keluaran yang juga berupa titik *crisp* dengan melalui tahap-tahap tertentu. Gambaran umum tahap-tahap tersebut dapat dilihat pada Gambar 2.6.



Gambar 2.6 Blok diagram sistem *fuzzy*

2.5.2 Himpunan *Fuzzy* dan Fungsi Keanggotaan

Ada beberapa hal yang perlu diketahui dalam memahami sistem *fuzzy* yaitu:

a. Variabel *fuzzy*, merupakan variabel yang dibahas dalam suatu sistem *fuzzy*, seperti umur, temperatur, permintaan.
b. Himpunan *fuzzy*, merupakan suatu grup yang mewakili suatu kondisi atau keadaan dalam suatu variabel *fuzzy*. Himpunan *fuzzy* memiliki dua atribut, antara lain:
    1. Linguistik, yaitu penamaan suatu grup yang mewakili suatu keadaan atau kondisi tertentu dengan menggunakan bahasa alami, seperti Kecil, Sedang, Besar.
    2. Numeris, yaitu suatu nilai (angka) yang menunjukkan ukuran suatu variabel, misalkan 40, 25, 50.

Sebagai contoh variabel umur terbagi menjadi 3 himpunan *fuzzy*, yaitu Muda, Dewasa, dan Tua.

Fungsi keanggotaan adalah suatu fungsi yang mendefinisikan bagaimana memetakan titik-titik dalam ruang masukan ke dalam derajat keanggotaannya, yang bernilai diantara 0 dan 1. Ruang masukan biasanya disebut sebagai semesta pembicaraan. Jika X adalah semesta pembicaraan dan x menyatakan elemennya, maka himpunan *fuzzy* A dalam X dinyatakan sebagai

$$A = \{(x, \mu_A(x)) \mid x \in X\} \tag{2.24}$$

dengan $\mu_A(x)$ disebut fungsi keanggotaan dari x dalam A. Fungsi keanggotaan, yang memetakan setiap elemen dari X ke nilai keanggotaan antara 0 dan 1, secara matematis dinyatakan sebagai

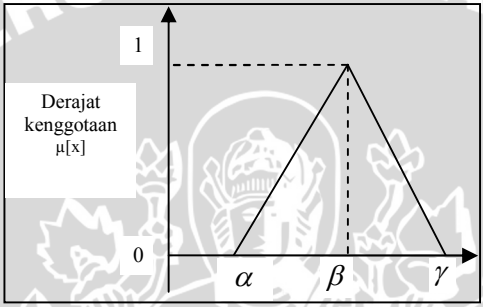
$$\mu_A(x): X \to [0,1]. \tag{2.25}$$

Fungsi keanggotaan yang digunakan dalam skripsi ini adalah sebagai berikut.

1. Fungsi keanggotaan segitiga (*triangular membership function*).

$$\mu_A(\alpha,\beta,\gamma) = \max\left(\min\left(\frac{x-\alpha}{\beta-\alpha}, \frac{\gamma-x}{\gamma-\beta}\right), 0\right)$$

Bentuk fungsi keanggotaan ini dapat dilihat pada Gambar 2.7, dengan α menyatakan batas bawah, γ menyatakan batas atas, dan β menyatakan titik tengah domain.

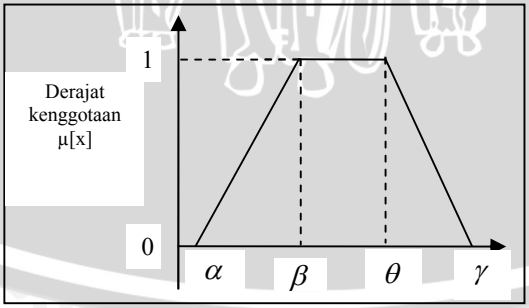


Gambar 2.7 Fungsi keanggotaan segitiga

2. Fungsi keanggotaan trapesium (*trapezoidal membership function*).

$$\mu_A(\alpha,\beta,\theta,\gamma) = \max\left(\min\left(\frac{x-\beta}{\beta-\alpha}, 1, \frac{\gamma-x}{\gamma-\theta}\right), 0\right)$$

Fungsi keanggotaan trapesium pada dasarnya seperti bentuk segitiga, hanya saja ada beberapa titik yang memiliki nilai keanggotaan 1. Lebih jelasnya lihat Gambar 2.8 berikut.

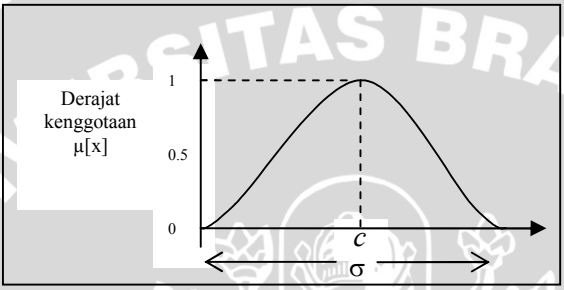


Gambar 2.8 Fungsi keanggotaan trapesium

3. Fungsi keanggotaan Gaussian (*Gaussian membership function*).

$$\mu_A(c,\sigma) = e^{-\frac{1}{2}\left(\frac{x-c}{\sigma}\right)^2}$$

Fungsi keanggotaan Gaussian menggunakan 2 parameter yaitu c yang menunjukkan nilai domain pada pusat kurva dan σ yang menunjukkan lebar kurva, lihat Gambar 2.9 berikut ini.



Gambar 2.9 Fungsi keanggotaan Gaussian

2.5.3 Operator Himpunan *Fuzzy*

Terdapat 2 jenis operator himpunan *fuzzy* yang digunakan di dalam operasi himpunan *fuzzy*, yaitu operator *t-norm* dan *s-norm*.

Definisi 2.8 T-norm

Operator T-norm adalah fungsi $T(\cdot,\cdot)$ yang memenuhi:

a. $T(0,0) = 0, \quad T(a,1) = T(1,a) = a$ (tertutup)

b. $T(a,b) \le T(c,d)$ jika $a \le c$ dan $b \le d$ (kemonotonan)

c. $T(a,b) = T(b,a)$ (komutatif)

d. $T(a,T(b,c)) = T(T(a,b),c)$ (asosiatif)

Beberapa operator T-norm yang sering digunakan, yaitu:

- *Minimum* $T_{\min}(a,b) = \min(a,b) = a \wedge b$
- *Algebraic product* $T_{ap}(a,b) = ab$

Definisi 2.9 S-norm

Operator T-conorm (S-norm) adalah fungsi $S(\cdot,\cdot)$ yang memenuhi:

a. $S(1,1)=1, \quad S(0,a)=S(a,0)=a$ (tertutup)

b. $S(a,b)\le S(c,d)$ jika $a\le c$ dan $b\le d$ (kemonotonan)

c. $S(a,b)=S(b,a)$ (komutatif)

d. $S(a,S(b,c))=S(S(a,b),c)$ (asosiatif)

Beberapa operator S-norm yang sering digunakan, yaitu:

- *Maximum* $S(a,b)=\max(a,b)=a\vee b$
- *Algebraic sum* $S(a,b)=a+b-ab$

2.5.4 Aturan dan Inferensi *Fuzzy*

Aturan *fuzzy* berisi sekumpulan aturan jika-maka dalam bentuk

jika x_1 adalah F_1 dan ... dan ....x_n adalah F_n maka y adalah G

(2.26)

dengan F dan G merupakan himpunan *fuzzy*, x dan y adalah variabel *fuzzy* dan n adalah banyaknya himpunan masukan. Setiap aturan *fuzzy* memiliki bobot aturan dengan nilai antara 0 dan 1 atau ditentukan sesuai kebutuhan.

Inferensi *fuzzy* adalah mekanisme pemetaan himpunan *fuzzy* pada masukan menjadi himpunan *fuzzy* lainnya pada keluaran berdasarkan kumpulan aturan yang terdapat pada basis aturan. Pada basis aturan (2.26), masukan x dipetakan menjadi keluaran y. Untuk dapat melakukan inferensi dibutuhkan interpretasi terhadap aturan jika-maka. Aturan jika-maka diintepretasikan sebagai implikasi *fuzzy* $F_1\times\ldots\times F_n\rightarrow G$.

2.5.5 Fuzzifikasi dan Defuzzifikasi

Fuzzifikasi adalah proses memetakan titik *crisp* x di X ke dalam himpunan *fuzzy* A yang dinyatakan sebagai derajat keanggotaan sedangkan defuzzifikasi adalah proses memetakan kembali besaran yang berupa himpunan *fuzzy* menjadi titik *crisp*. Defuzzifikasi dibutuhkan dalam penerapan sistem *fuzzy* karena yang digunakan

dalam aplikasi adalah besaran *crisp*. Beberapa metode yang digunakan dalam defuzzifikasi yaitu:

a. *Centroid of Area* (COA) / *Center of Gravity*, yang dinyatakan sebagai

$$COA = \frac{\int_{Z} \mu_A(z) z dz}{\int_{Z} \mu_A(z) dz}$$

dengan z adalah semesta pembicaraan himpunan keluaran Z.

b. *Bisector of Area* (BOA), yang dinyatakan secara matematis dalam bentuk

$$\int_{\alpha}^{zBOA} \mu_{A(z)dz} = \int_{zBOA}^{\beta} \mu_{A(z)dz}$$

α = min ($z|z \in Z$), dan β = max ($z|z \in Z$), yaitu garis vertikal z = BOA membagi daerah antara $z = \alpha$, $z = \beta$, $y = 0$ dan $y = \mu_A(z)$ ke luas daerah yang sama.

c. *Mean of Maximum* (MOM), yaitu z maksimum rata-rata pada saat fungsi keanggotaan menjadi maksimum dan dirumuskan sebagai

$$MOM = \frac{\int_{Z'} z\, dz}{\int_{Z'} dz}$$

dimana Z' = $\{z \mid \mu_A(z) = \mu^*\}$.

2.5.6 Sistem *Fuzzy* Mamdani

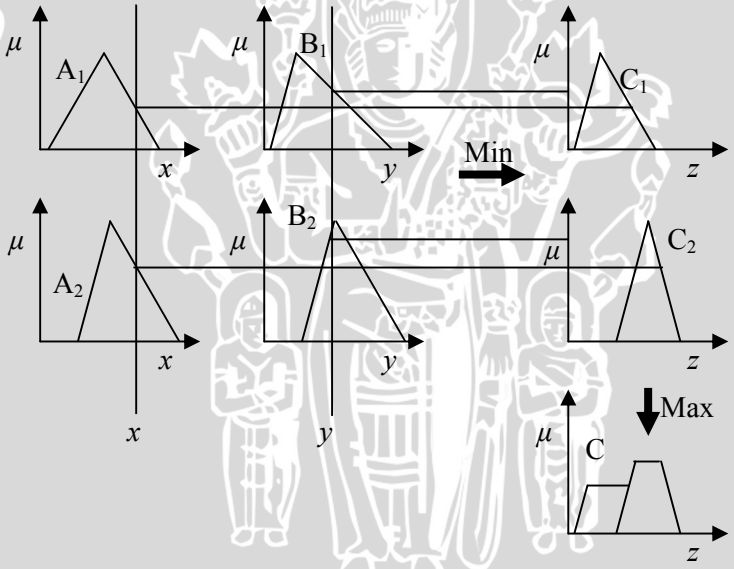
Salah satu sistem logika *fuzzy* yang paling banyak digunakan sebagai pengontrol adalah sistem logika fuzzy Mamdani. Sistem *fuzzy* Mamdani menggunakan aturan *fuzzy* seperti pada persamaan (2.26). Terdapat 6 tahap untuk mendapatkan keluaran dengan menggunakan sistem *fuzzy* Mamdani, yaitu:

1. menentukan aturan *fuzzy* yang berisi sekumpulan aturan jika-maka,
2. melakukan fuzzifikasi masukan menggunakan fungsi keanggotaan variabel masukan,

3. melakukan operasi inferensi sesuai aturan *fuzzy* dengan menggunakan operator *t-norm* atau *s-norm*,
4. menentukan keluaran masing-masing aturan *fuzzy* yang disesuaikan dengan fungsi keanggotaan variabel keluaran,
5. melakukan operasi *t-norm* atau *s-norm* terhadap keluaran yang diperoleh masing-masing aturan, dan
6. melakukan defuzzifikasi.

Di bawah ini diilustrasikan bagaimana sistem inferensi Mamdani memberikan penalaran terhadap masukannya.

Aturan 1 *:* jika *x* adalah A_1 dan *y* adalah B_1 maka $z = C_1$
Aturan 2 *:* jika *x* adalah A_2 dan *y* adalah B_2 maka $z = C_2$



Gambar 2.10 Mekanisme inferensi *fuzzy* Mamdani

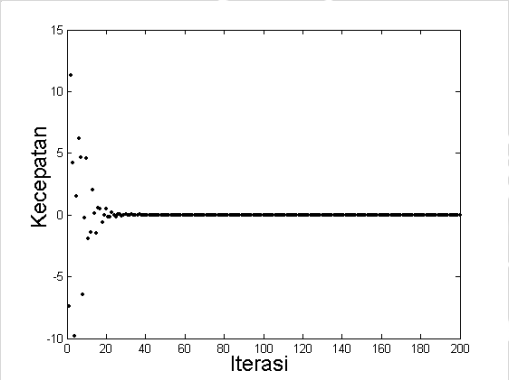
Gambar diatas memperlihatkan bagaimana dua aturan sistem inferensi *fuzzy* Mamdani menghasilkan keluaran *z* ketika diberikan dua masukan *crisp x* dan *y*, dengan menggunakan *min-max* masing-masing untuk operator *T-Norm* dan *S-Norm*.

(Jang dkk, 1997)

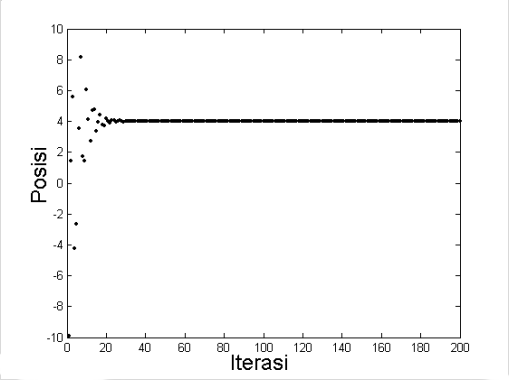

UNIVERSITAS BRAWIJAYA

BAB III
HASIL DAN PEMBAHASAN

Algoritma *Particle Swarm Optimization* (PSO) hampir tidak pernah memperoleh solusi optimal apabila diimplementasikan pada masalah optimasi berdimensi tinggi (Abraham dan Liu, 2005). Hal ini dikarenakan, algoritma PSO mengalami kondisi stagnan. Menurut Clerc (2006), kondisi stagnan adalah suatu kondisi dimana tidak terjadi perubahan posisi partikel, yang berdampak pada nilai *fitness* yang selalu tetap, lihat Gambar 3.1 dan 3.2. Kondisi ini dapat terjadi karena parameter yang dijadikan acuan dalam pergerakan partikel, yaitu posisi terbaik partikel ($\boldsymbol{y}$) dan posisi terbaik populasi ($\hat{\boldsymbol{y}}$), tidak mengalami perubahan.



Gambar 3.1 Kecepatan partikel pada kondisi stagnan



Gambar 3.2 Posisi partikel pada kondisi stagnan

3.1 Algoritma *Turbulence Particle Swarm Optimization* (TPSO)

Algoritma *Turbulence Particle Swarm Optimization* (TPSO) adalah perbaikan dari algoritma PSO yang bertujuan untuk mengatasi kondisi stagnan yang terjadi pada algoritma PSO. Gagasan yang mendasari algoritma ini adalah memberi batasan nilai minimum pada perbaikan kecepatan, maksudnya apabila kecepatan partikel lebih kecil dari batasan nilai minimum, partikel tersebut diberi kecepatan baru yang nilainya lebih besar. Dengan pemberian nilai kecepatan yang baru tersebut, diharapkan partikel dapat melakukan pencarian solusi yang lebih optimal, karena algoritma PSO yang mengalami kondisi stagnan biasanya memperoleh solusi yang tidak optimal.

3.1.1 Perbaikan Kecepatan dan Posisi

Pada dasarnya, algoritma TPSO sama dengan algoritma PSO yang telah dijelaskan sebelumnya. Perbedaan algoritma TPSO dengan algoritma PSO hanya terletak pada perbaikan kecepatan yaitu

$$v_{i,j}(t+1) = w.\hat{v}_{i,j}(t) + c_1.R_{1,j}(t)\left[y_{i,j}(t) - x_{i,j}(t)\right] + c_2.R_{2,j}(t)\left[\hat{y}_{i,j}(t) - x_{i,j}(t)\right] \quad (3.1)$$

dengan

$$\hat{v}_{i,j} = \begin{cases} v_{i,j} & \text{jika } |v_{i,j}| \geq v_c \\ \boldsymbol{U}(-1,1)v_{\max} / \rho & \text{jika } |v_{i,j}| < v_c \end{cases} \quad (3.2)$$

dimana $i = 1, 2, \ldots, s$, $j = 1, 2, \ldots, n$, dengan s adalah banyaknya partikel, n adalah banyaknya dimensi masalah optimasi, $\boldsymbol{U}(-1,1)$ adalah vektor peubah acak berdistribusi seragam yang nilainya berada pada selang $[-1,1]$, v_{maks} adalah batasan nilai kecepatan maksimum, ρ adalah faktor penyekala, dan v_c adalah batasan nilai minimum kecepatan. Persamaan untuk memperbaiki posisi dinyatakan sebagai

$$\boldsymbol{x}_i(t+1) = \boldsymbol{x}_i(t) + \boldsymbol{v}_i(t+1) \quad (3.3)$$

dengan

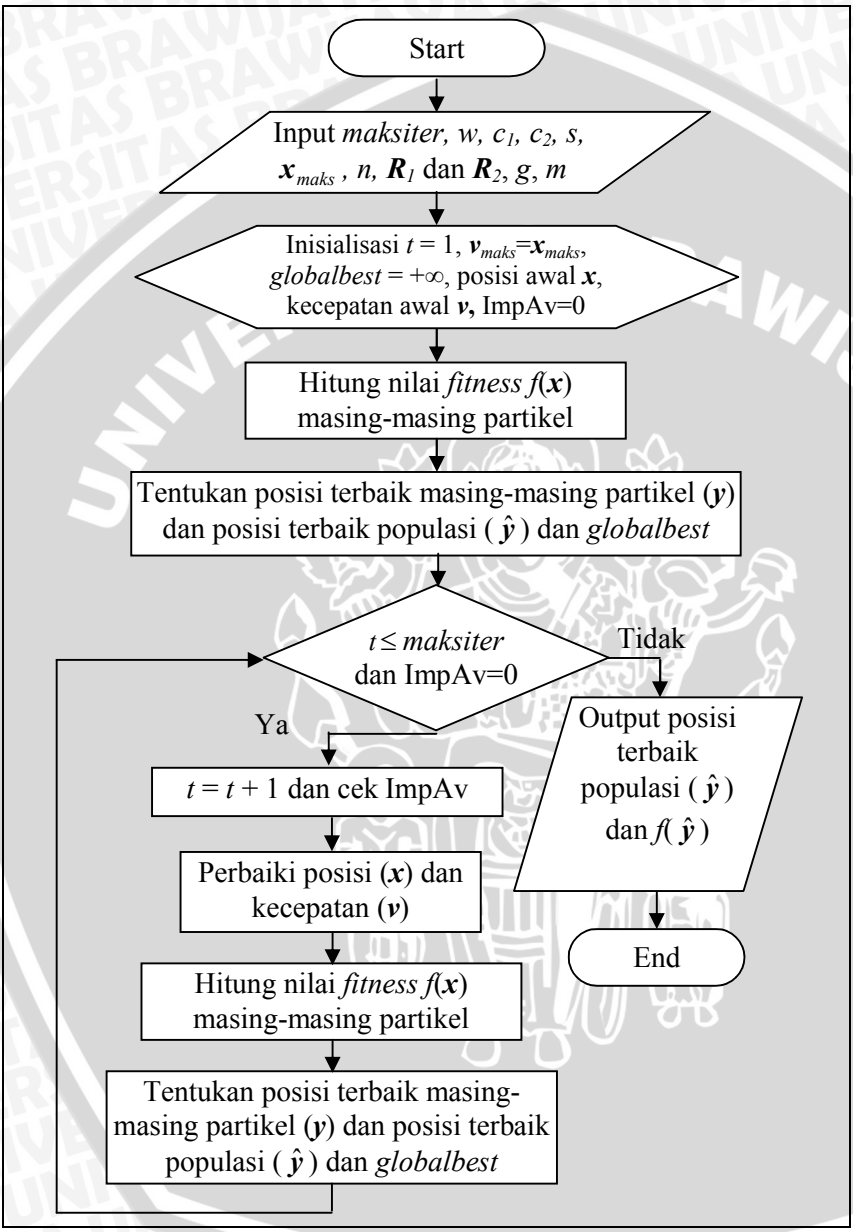
$$\boldsymbol{x}_i(t+1) = \begin{cases} \boldsymbol{x}_i(t+1) & \text{jika } abs(\boldsymbol{x}_i(t+1)) \leq \boldsymbol{x}_{maks} \\ sign[\boldsymbol{x}_i(t+1)]\boldsymbol{x}_{maks} & \text{jika } abs(\boldsymbol{x}_i(t+1)) > \boldsymbol{x}_{maks} \end{cases} . \quad (3.4)$$

Gambaran umum algoritma TPSO dapat dilihat pada Algoritma 3.1

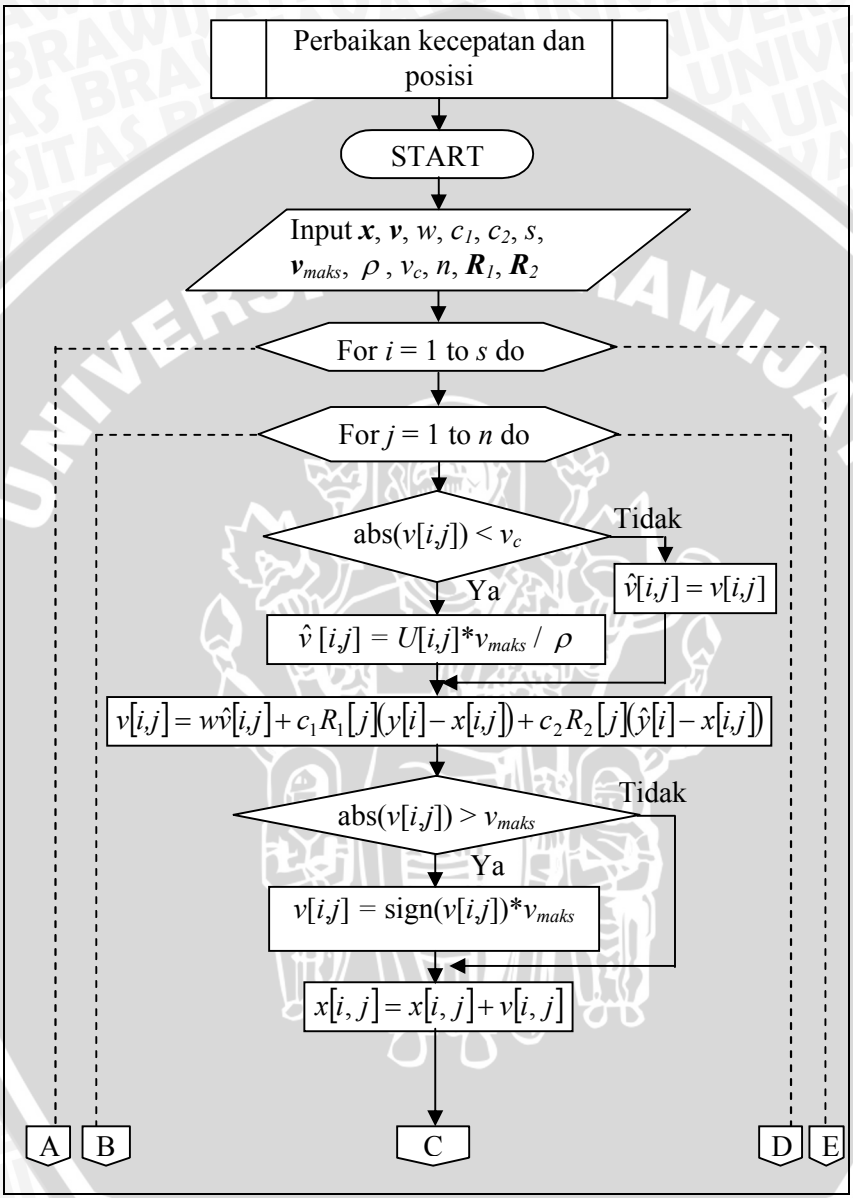
dan diagram alir pada Gambar 3.3, 3.4(a), dan 3.4(b).

Algoritma 3.1 Algoritma TPSO

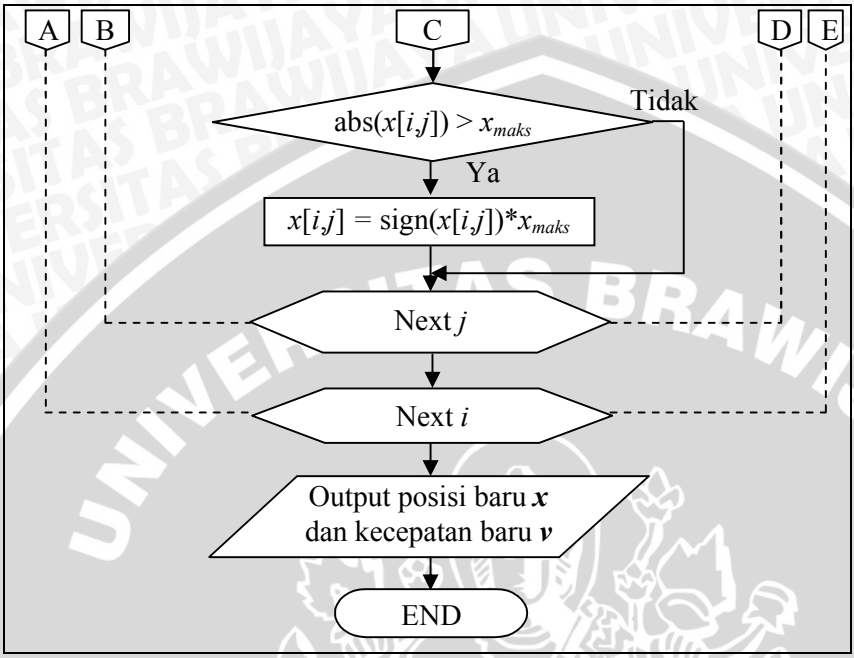
Input : *Maksiter*, n, w, c_1, c_2, s, $\boldsymbol{x}_{maks}$, ρ , v_c, g, m, $\boldsymbol{R}_1$, $\boldsymbol{R}_2$
$t \leftarrow 1$
globalbest $\leftarrow +\infty$
ImpAv $\leftarrow 0$
for $i \leftarrow 1$ to s do
  $\boldsymbol{x}_i \leftarrow$ random
  $\boldsymbol{v}_i \leftarrow$ random
  $\boldsymbol{y}_i \leftarrow \boldsymbol{x}_i$
  Currentvalue$_i \leftarrow f(\boldsymbol{x}_i)$
  bestvalue$_i \leftarrow$ currentvalue$_i$
  if currentvalue$_i <$ *globabest* then
    globalbest $\leftarrow$ currentvalue$_i$
    $\hat{\boldsymbol{y}} \leftarrow \boldsymbol{x}_i$
while $t \leq$ *maksiter* dan ImpAv = 0 do
  $t \leftarrow t + 1$
  cek $\leftarrow 0$
  rata(t) $\leftarrow$ mean(bestvalue)
  if $t > g$
    for $k \leftarrow (t\text{-}g)$ to $(t\text{-}1)$
      if abs(rata(t)-rata(k)) $\leq m$
        cek $\leftarrow$ *cek* + 1
    if *cek* = g
      ImpAv $\leftarrow 1$
  for $i \leftarrow 1$ to s do
    perbaiki $\boldsymbol{v}_i$ menurut persamaan (3.1) dan (3.2)
    perbaiki $\boldsymbol{x}_i$ menurut persamaan (3.3) dan (3.4)
    currentvalue$_i \leftarrow f(\boldsymbol{x}_i)$
    if currentvalue$_i <$ bestvalue$_i$
      $\boldsymbol{y}_i \leftarrow \boldsymbol{x}_i$
      bestvalue$_i \leftarrow$ currentvalue$_i$
      if currentvalue$_i <$ *globalbest* then
        globalbest $\leftarrow$ currentvalue$_i$
        $\hat{\boldsymbol{y}} \leftarrow \boldsymbol{x}_i$
Output : $\hat{\boldsymbol{y}}$, $f(\hat{\boldsymbol{y}})$



Gambar 3.3 Diagram alir algoritma TPSO



Gambar 3.4(a) Diagram alir perbaikan kecepatan dan posisi menggunakan algoritma TPSO



Gambar 3.4(b) Diagram alir perbaikan kecepatan dan posisi menggunakan algoritma TPSO

Contoh perhitungan manual algoritma TPSO dapat dilihat pada lampiran 1.

3.1.2 Parameter TPSO

Dalam algoritma TPSO, terdapat beberapa parameter yang mempengaruhi keberhasilan algoritma, selain yang telah dijelaskan, yaitu:

1. Batasan kecepatan minimum (v_c)

   Batasan kecepatan minimum v_c merupakan parameter terpenting dalam algoritma TPSO, karena dengan adanya parameter ini, partikel akan terhindar dari kondisi stagnan yang diakibatkan oleh penurunan kecepatan. Nilai v_c sangat mempengaruhi pergerakan partikel. Nilai v_c yang kecil dapat memperpendek periode osilasi partikel sehingga berpeluang kecil untuk keluar dari titik minimum lokal. Nilai v_c yang besar dapat menyebabkan partikel melakukan pencarian daerah baru secara terus menerus, sehingga hasil optimal tidak pernah

tercapai. Secara umum, nilai v_c yang besar mengindikasikan pencarian global sedangkan nilai v_c yang kecil mengindikasikan pencarian lokal.

2. Faktor penyekala (ρ)

   Faktor penyekala ρ berpengaruh pada pengaturan domain pergerakan partikel. Nilai ρ yang besar dapat memperkecil perubahan posisi partikel sehingga berpeluang kecil untuk keluar dari titik minimum lokal, sedangkan nilai ρ yang kecil dapat memperbesar perubahan posisi partikel dan berkemungkinan keluar dari daerah pencarian.

3. Batasan kecepatan maksimum (v_{maks})

   Batasan kecepatan maksimum digunakan untuk mengontrol kecepatan partikel agar tidak keluar dari domain daerah pencarian. Apabila kecepatan partikel lebih besar dari batasan yang telah ditentukan, partikel tersebut diberi kecepatan baru yang nilainya lebih kecil.

3.1.3 Analisis Kekonvergenan Algoritma TPSO

Misalkan masalah optimasi yang diselesaikan sebagai berikut

$$\min f(\boldsymbol{x}),\ \boldsymbol{x} \in I = \left[-x_{maks}, x_{maks}\right]^n \subseteq \Re^n \qquad (3.5)$$

dimana $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^T$ dan f adalah fungsi tujuan masalah optimasi. Misalkan $\boldsymbol{x}^*$ adalah solusi minimum global dari masalah optimasi dan $f^* = f(\boldsymbol{x}^*)$. Misalkan juga

$$\begin{aligned} D_0 &= \left\{\boldsymbol{x} \in I \;\middle|\; \left|f(\boldsymbol{x}) - f^*\right| < \varepsilon\right\} \\ D_1 &= I \setminus D_0 \end{aligned} \qquad (3.6)$$

untuk setiap $\varepsilon > 0$.

Asumsikan bahwa kecepatan partikel lebih kecil dari nilai v_c sehingga kecepatan diperbaiki menggunakan persamaan (3.2). Karena pada persamaan (3.2) vektor $\boldsymbol{U}$(-1,1) adalah vektor peubah acak berdistribusi seragam dengan selang [-1,1], maka nilai $\hat{v}$ juga berdistribusi seragam dan berada pada selang $\left[-\frac{x_{maks}}{\rho}, \frac{x_{maks}}{\rho}\right]$ jika diambil k = 1, $v_{\max} = x_{max}$. Misalkan peluang perpindahan partikel dari iterasi t ke t+1 dinyatakan dalam q_{ij}, dengan $\boldsymbol{x}(t) \in D_i$ dan $\boldsymbol{x}(t+1) \in D_j$, sehingga perpindahan posisi partikel dalam populasi

dibagi menjadi 4 yaitu q_{00}, q_{01}, q_{10}, dan q_{11}. Dalam hal ini $q_{00} + q_{01} = 1$ dan $q_{10} + q_{11} = 1$.

Lemma 3.1 (*Particle State Transference*)

Jika peluang perpindahan partikel dari iterasi t ke t+1 dinyatakan dalam q_{ij}, dengan $\boldsymbol{x}(t) \in D_i$ dan $\boldsymbol{x}(t+1) \in D_j$, maka $q_{01} = 0$; $q_{00} = 1$; $q_{11} \leq c$ dan $q_{10} \leq 1\text{-}c, c \in (0,1)$.

Bukti:

Misalkan peluang perpindahan partikel dari iterasi t ke $t+1$, yaitu dari $\boldsymbol{x}(t) \in D_i$ ke $\boldsymbol{x}(t+1) \in D_j$ dinyatakan sebagai q_{ij}. Dalam algoritma TPSO, posisi terbaik partikel selalu diperbaiki selama iterasi berjalan, sehingga $q_{00} = 1$ dan $q_{01} = 0$.

Misalkan $\hat{\boldsymbol{x}}$ adalah posisi terbaik populasi yang ditemukan setelah iterasi ke-t. Sesuai definisi pada persamaan (3.5), maka $\exists r > 0$, pada saat $\|\boldsymbol{x} - \hat{\boldsymbol{x}}\| \leq r$ menyebabkan $|f(\boldsymbol{x}) - f^*| < \varepsilon$ sehingga dapat dibuat suatu persekitaran $N_{\hat{\boldsymbol{x}},r} = \{\boldsymbol{x} \in I \mid \|\boldsymbol{x} - \hat{\boldsymbol{x}}\| \leq r\}$. Karena D_0 adalah himpunan posisi partikel yang cukup dekat dengan solusi minimum globalnya, maka $N_{\hat{\boldsymbol{x}},r} \subset D_0$.

Perpindahan partikel untuk setiap dimensi merupakan kejadian saling lepas, karena nilai dari perpindahan partikel pada arah sumbu-x tidak dipengaruhi oleh perpindahan partikel pada arah sumbu-y, sehingga

$$\begin{aligned} P\left((\boldsymbol{x} + \Delta\boldsymbol{x}) \in N_{\hat{\boldsymbol{x}},r}\right) &= \prod_{i=1}^{n} P\left(|\, x_i + \Delta x_i - \hat{x}_i \,| \leq r\right) \\ &= \prod_{i=1}^{n} P\left(-r \leq x_i + \Delta x_i - \hat{x}_i \leq r\right) \\ &= \prod_{i=1}^{n} P\left(\hat{x}_i - x_i - r \leq \Delta x_i \leq \hat{x}_i - x_i + r\right) \end{aligned}$$

Karena nilai perpindahan partikel ($\Delta\boldsymbol{x}$) merupakan nilai $\hat{v}$, maka nilai $\Delta\mathbf{x} \in \left[-\frac{x_{maks}}{\rho}, \frac{x_{maks}}{\rho}\right]$ yang menyebabkan

$$P\left((\boldsymbol{x}+\Delta\boldsymbol{x})\in N_{\hat{\boldsymbol{x}},r}\right)=\prod_{i=1}^{n}\int_{\hat{x}_i-x_i-r}^{\hat{x}_i-x_i+r}\frac{1}{\frac{x_{maks}}{\rho}+\frac{x_{maks}}{\rho}}\,dx$$

$$=\prod_{i=1}^{n}\int_{\hat{x}_i-x_i-r}^{\hat{x}_i-x_i+r}\frac{\rho}{2x_{maks}}\,dx$$

$$=\prod_{i=1}^{n}\frac{\rho}{2x_{maks}}\left(\hat{x}_i-x_i+r-\hat{x}_i+x_i+r\right)$$

$$=\prod_{i=1}^{n}\frac{\rho\,r}{x_{maks}}=\left(\frac{\rho\,r}{x_{maks}}\right)^{n}.$$

Misalkan $P_1(\boldsymbol{x})=P\left((\boldsymbol{x}+\Delta\boldsymbol{x})\in N_{\hat{\boldsymbol{x}},r}\right)$, maka nilai $0<P_1(\boldsymbol{x})<1$, $\boldsymbol{x}\in I$. Karena I kompak, maka $\exists\,\hat{\boldsymbol{y}}\in I$ sedemikian sehingga

$$P_1(\hat{\boldsymbol{y}})=\min_{\boldsymbol{x}\in I}P_1(\boldsymbol{x})$$

yang menyebabkan $0<P_1(\hat{\boldsymbol{y}})<1$. Dari hubungan tersebut diperoleh

$$P_1(\hat{\boldsymbol{y}})\le P_1(\boldsymbol{x})\le q_{10}$$

dengan q_{10} adalah peluang perpindahan partikel dari D_1 ke D_0. Misalkan $c=1-P_1(\hat{\boldsymbol{y}})$, $c\in(0,1)$, maka

$$c=1-P_1(\hat{\boldsymbol{y}})\ge 1-q_{10}=q_{11}$$

sehingga $q_{11}\le c$ dan $q_{10}\ge 1-c$ dengan $c\in(0,1)$.

Teorema 3.1

Asumsikan dalam algoritma TPSO, posisi terbaik partikel pada waktu ke-t dinyatakan dengan $\boldsymbol{y}_i(t)$, $i=1,2,\ldots,s$. Asumsikan juga $\hat{\boldsymbol{y}}(t)$ adalah posisi terbaik populasi yang dinyatakan sebagai

$$\hat{\boldsymbol{y}}(t)=\arg\min_{1\le i\le s}\left(f(\hat{\boldsymbol{y}}(t-1)),f(\boldsymbol{y}(t))\right) \tag{3.11}$$

maka

$$P(\lim_{t\to\infty}f(\hat{\boldsymbol{y}}(t))=f^*)=1 \tag{3.12}$$

Bukti:
Misalkan pada iterasi ke-k diperoleh posisi terbaik populasi

$$\hat{\boldsymbol{y}}(k)=\arg\min_{1\le i\le s}\left(f(\hat{\boldsymbol{y}}(k-1)),f(\boldsymbol{y}(k))\right)$$

dan misalkan

$$p_k = P\left(\left|f\left(\hat{\boldsymbol{y}}(k)\right) - f^*\right| \geq \varepsilon\right)$$

untuk $\forall \varepsilon > 0$, maka

$$p_k = \begin{cases} 0 & \text{jika } \exists T \in \{1,2,\ldots,k\},\ \hat{\boldsymbol{y}}(T) \in D_0 \\ \overline{p}_k & \text{jika } \hat{\boldsymbol{y}}(t) \notin D_0, \quad t = 1,2,\ldots,k \end{cases}$$

dengan

$$\overline{p}_k = P\left(\hat{\boldsymbol{y}}(t) \notin D_0, t = 1,2,\ldots,k\right)$$

Berdasarkan Lemma 3.1, nilai pada iterasi ke-*k* adalah

$$\overline{p}_k = q_{11}^k \leq c^k$$

sehingga diperoleh

$$\sum_{k=1}^{\infty} p_k \leq \sum_{k=1}^{\infty} c^k$$

Karena $c \in (0,1)$, maka $\sum_{k=1}^{\infty} c^k = \frac{c}{1-c} < \infty$ sehingga $\sum_{k=1}^{\infty} p_k < \infty$.

Berdasarkan Lemma 3.1,

$$P\left(\limsup_{t\to\infty} \left\{\left|f\left(\hat{\boldsymbol{y}}(t)\right) - f^*\right| \geq \varepsilon\right\}\right) = 0$$

sehingga berdasarkan Definisi 2.12 diperoleh

$$P(\lim_{t\to\infty} f\left(\hat{\boldsymbol{y}}(t)\right) = f^*) = 1$$

yaitu barisan $\left\langle f\left(\hat{\boldsymbol{y}}(t)\right)\right\rangle$ konvergen ke f^* dengan peluang 1.

3.2 Algoritma *Fuzzy Adaptive Turbulence Particle Swarm Optimization* (FATPSO)

Parameter batasan kecepatan minimum v_c dan faktor penyekala ρ yang ada pada algoritma TPSO (persamaan (3.2)) sangat berpengaruh pada optimal tidaknya hasil yang akan diperoleh. Penentuan nilai parameter yang tidak sesuai dapat menyebabkan algoritma tidak pernah konvergen atau tidak pernah memperoleh hasil yang optimal. Oleh karena itu, penentuan nilai parameter dibagi menjadi 3 tahap, yaitu:

1. nilai v_c diberi nilai yang besar sedangkan ρ diberi nilai yang kecil,
2. nilai v_c dan ρ diberi nilai yang sedang, dan
3. nilai v_c diberi nilai yang kecil dan ρ diberi nilai yang besar.

Pembagian parameter tersebut dimaksudkan untuk menjaga keseimbangan pergerakan partikel. Penentuan nilai v_c yang besar dan nilai ρ yang kecil pada tahap awal menyebabkan partikel melakukan pencarian daerah baru sehingga berkemungkinan dapat keluar dari kondisi konvergen prematur sedangkan penentuan nilai v_c yang kecil dan nilai ρ yang besar pada tahap akhir menyebabkan partikel melakukan pencarian solusi terbaik dalam suatu daerah pencarian.

Algoritma FATPSO (*Fuzzy Adaptif Turbulence Particle Swarm Optimization*) adalah algoritma TPSO dimana batasan nilai minimum kecepatan dikontrol secara adaptif menggunakan *Fuzzy Logic Controller*. Penggunaan *Fuzzy Logic Controller* akan sangat cocok untuk menentukan nilai parameter v_c dan ρ yang tidak terdefinisi dengan jelas karena *Fuzzy Logic Controller* dapat mengatur nilai parameter tersebut sesuai dengan masalah yang dihadapi.

1.2.1 Perancangan *Fuzzy Logic Controller*

Pada algoritma FATPSO ini, *Fuzzy Logic Controller* yang digunakan adalah sistem *fuzzy* Mamdani. Sistem yang dirancang untuk mengontrol batasan nilai minimum kecepatan partikel ini menggunakan dua masukan, yaitu *Normalized Current Best Performance Evaluation* (*NCBPE*) dan *Current Velocity* (*CV*) dan dua keluaran, yaitu ρ dan V_{ck}. Nilai *NCBPE* diperoleh dari hasil

normalisasi *Current Best Performance Evaluation* (*CBPE*) menurut persamaan (3.13).

$$NCBPE = \frac{CBPE - CBPE_{\min}}{CBPE_{maks} - CBPE_{\min}} \tag{3.13}$$

dengan $CBPE_{min}$ adalah solusi terbaik dan $CBPE_{maks}$ adalah solusi terburuk dari masalah optimasi. Apabila tidak ada informasi mengenai keduanya, nilai *NCBPE* dapat diberi nilai 1 pada awal iterasi dan diturunkan sampai 0 selama iterasi berjalan. Nilai V_{ck} pada bagian keluaran digunakan untuk menghitung batasan nilai minimum v_c menurut persamaan (3.14)

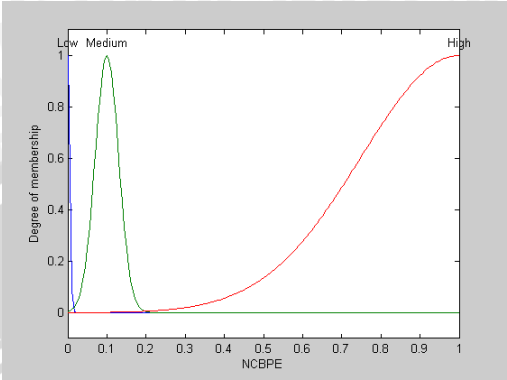
$$v_c = e^{-[10(1+V_{ck})]}. \tag{3.14}$$

Semesta pembicaraan untuk variabel masukan *NCBPE* dibagi menjadi tiga himpunan *fuzzy*, yaitu *Low*, *Medium*, dan *High*. Variabel masukan *CV* dibagi menjadi dua himpunan *fuzzy*, yaitu *Low* dan *High*. Sedangkan untuk variabel keluaran, ρ dibagi menjadi tiga himpunan *fuzzy* yaitu *Low*, *Medium*, dan *High* dan demikian juga V_{ck} dibagi menjadi tiga himpunan *fuzzy* yaitu *Small*, *Medium*, dan *Large*.

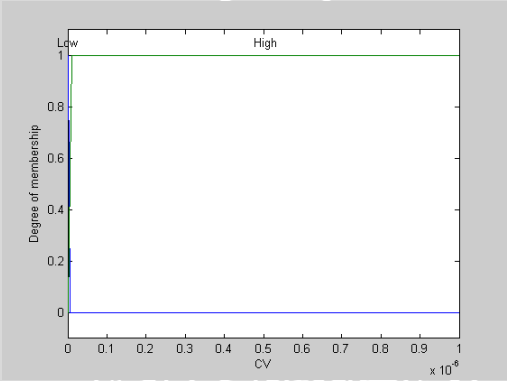
Fungsi keanggotaan yang digunakan adalah fungsi keanggotaan segitiga, trapesium dan Gaussian. Untuk lebih jelasnya dapat dilihat di Tabel 3.1 sedangkan untuk grafik fungsi keanggotaan dapat dilihat pada Gambar 3.5.

Tabel 3.1 Tabel Keanggotaan Himpunan *Fuzzy*

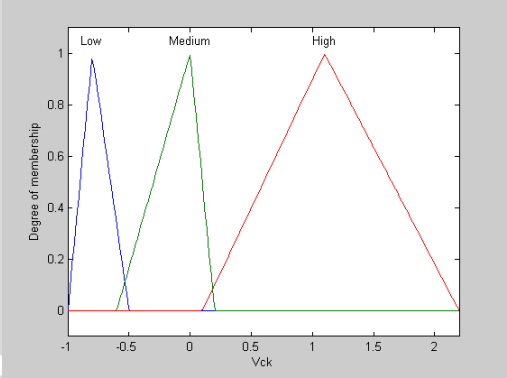
Jenis	Variabel	Himpunan *Fuzzy*	Fungsi keanggotaan	Parameter
Masukan	*NCBPE*	Low	Gaussian	(0.005, 0)
		Medium		(0.03, 0.1)
		High		(0.25, 1)
	CV	Low	Trapesium	(0, 0, 1e-030, 1e-020)
		High		(1e-010, 1e-008, 1e-006, 1e-006)
Keluaran	V_{ck}	Low	Segitiga	(-1, -0.8, -0.5)
		Medium		(-0.6, 0, 0.2)
		High		(0.1, 1.1, 2.2)
	ρ	Small	Segitiga	(1, 1, 4)
		Medium		(2.214, 10.71, 59.29)
		Large		(47.15, 120, 120)



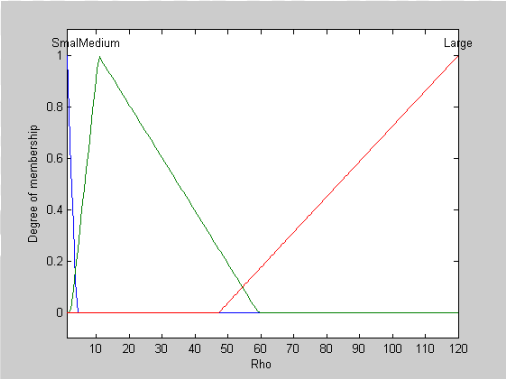
Gambar 3.5 (a) Fungsi keanggotaan *NCBPE*



Gambar 3.5 (b) Fungsi keanggotaan *CV*



Gambar 3.5 (c) Fungsi keanggotaan V_{ck}



Gambar 3.5 (d) Fungsi keanggotaan ρ

Pada *Fuzzy Logic Controller*, terdapat aturan-aturan yang disebut basis aturan *fuzzy*. Basis aturan ini menggambarkan mekanisme pengambilan keputusan kontrol untuk mengatur efek-efek tertentu yang datang pada sistem. Dalam pengontrolan batasan nilai kecepatan minimum partikel ini, operator yang digunakan adalah operator T-norm dan S-norm, yaitu *min-max*. Basis aturan yang digunakan sebagai berikut:

1. Jika *NCBPE* adalah *Low* dan *CV* adalah *Low* maka V_{ck} adalah *High*.
2. Jika *NCBPE* adalah *Medium* maka V_{ck} adalah *Medium*.
3. Jika *NCBPE* adalah *High* dan *CV* adalah *High* maka V_{ck} adalah *Low*.
4. Jika *NCBPE* adalah *Low* dan *CV* adalah *Low* maka V_{ck} adalah ρ adalah *Large*.
5. Jika *NCBPE* adalah *Medium* maka ρ adalah *Medium*.
6. Jika *NCBPE* adalah *High* dan *CV* adalah *High* maka ρ adalah *Small*.

Keluaran dari *Fuzzy Logic Controller* terdiri dari beberapa besaran linguistik, masing-masing dengan derajat keanggotaan tertentu. Agar bisa digunakan dalam penghitungan, maka besaran linguistik ini harus dikonversikan terlebih dahulu ke besaran numerik yang disebut sebagai proses defuzzifikasi. Metode defuzzifikasi yang digunakan adalah metoda titik berat (CoG).

3.2.2 Perbaikan Kecepatan dan Posisi

Perbaikan kecepatan dan posisi partikel pada algoritma FATPSO sama halnya dengan algoritma TPSO, hanya saja batasan nilai kecepatan minimum v_c dan faktor penyekala ρ dihitung menggunakan *Fuzzy Logic Controller*. Secara umum perbaikan kecepatan dan posisi partikel menggunakan algoritma FATPSO dapat dilihat pada Algoritma 3.2 sedangkan diagram alir algoritma FATPSO dapat dilihat pada Gambar 3.6, 3.7(a), dan 3.7(b).

Algoritma 3.2(a) Algoritma FATPSO

Input : *Maksiter*, n, w, c_1, c_2, s, $\boldsymbol{x}_{maks}$, ρ , v_c, g, m, $\boldsymbol{R}_1$, $\boldsymbol{R}_2$

1. $t \leftarrow 1$
2. *globalbest* $\leftarrow +\infty$
3. ImpAv $\leftarrow 0$
4. $v_c \leftarrow$ 1e-6
5. for $i \leftarrow 1$ to s do
   a. $\boldsymbol{x}_i \leftarrow$ random
   b. $\boldsymbol{v}_i \leftarrow$ random
   c. $\boldsymbol{y}_i \leftarrow \boldsymbol{x}_i$
   d. currentvalue$_i \leftarrow f(\boldsymbol{x}_i)$
   e. CBPE$_i \leftarrow$ currentvalue$_i$
   f. CBPE$_{maks} \leftarrow$ max(currentvalue)
   g. CBPE$_{min} \leftarrow$ min(currentvalue)
   h. bestvalue$_i \leftarrow$ currentvalue$_i$
   i. if currentvalue$_i <$ *globalbest* then
      (i) *globalbest* $\leftarrow$ currentvalue$_i$
      (ii) $\hat{\boldsymbol{y}} \leftarrow \boldsymbol{x}_i$
6. while $t \leq$ *maksiter* and ImpAv=0 do
   a. $t \leftarrow t + 1$
   b. *cek* $\leftarrow 0$
   c. rata(t) $\leftarrow$ mean(bestvalue)
   d. if $t > g$
      (1) for $k \leftarrow (t\text{-}g)$ to $(t\text{-}1)$
         if abs(rata(t)-rata(k)) $\leq m$
            cek $\leftarrow$ *cek* + 1

Algoritma 3.2(b) Algoritma FATPSO

(2) if $cek = g$

ImpAv ← 1

e. for $i \leftarrow 1$ to s do

(1) $CV_i \leftarrow \boldsymbol{v}_i$

(2) $NCBPE_i \leftarrow \dfrac{CBPE_i - CBPE_{min}}{CBPE_{maks} - CBPE_{min}}$

(3) if abs($\boldsymbol{v}_i$)< 1e-6

(a) fismat←readfis('FATPSO.fis')

(b) FO←evalfis([$NCBPE_i$ abs(CV_i)],fismat)

(c) v_c←exp(-(10+FO(1)))

(d) $\rho \leftarrow$ FO(2)

(4) perbaiki $\boldsymbol{v}_i$ menurut persamaan (3.1) dan (3.2)

(5) perbaiki $\boldsymbol{x}_i$ menurut persamaan (3.3) dan (3.4)

(6) $currentvalue_i \leftarrow f(\boldsymbol{x}_i)$

(7) CBPE ← $currentvalue_i$

(8) $CBPE_{maks} \leftarrow$ max(currentvalue)

(9) $CBPE_{min} \leftarrow$ min(currentvalue)

(10) if $currentvalue_i < bestvalue_i$

(a) $\boldsymbol{y}_i \leftarrow \boldsymbol{x}_i$

(b) $bestvalue_i \leftarrow currentvalue_i$

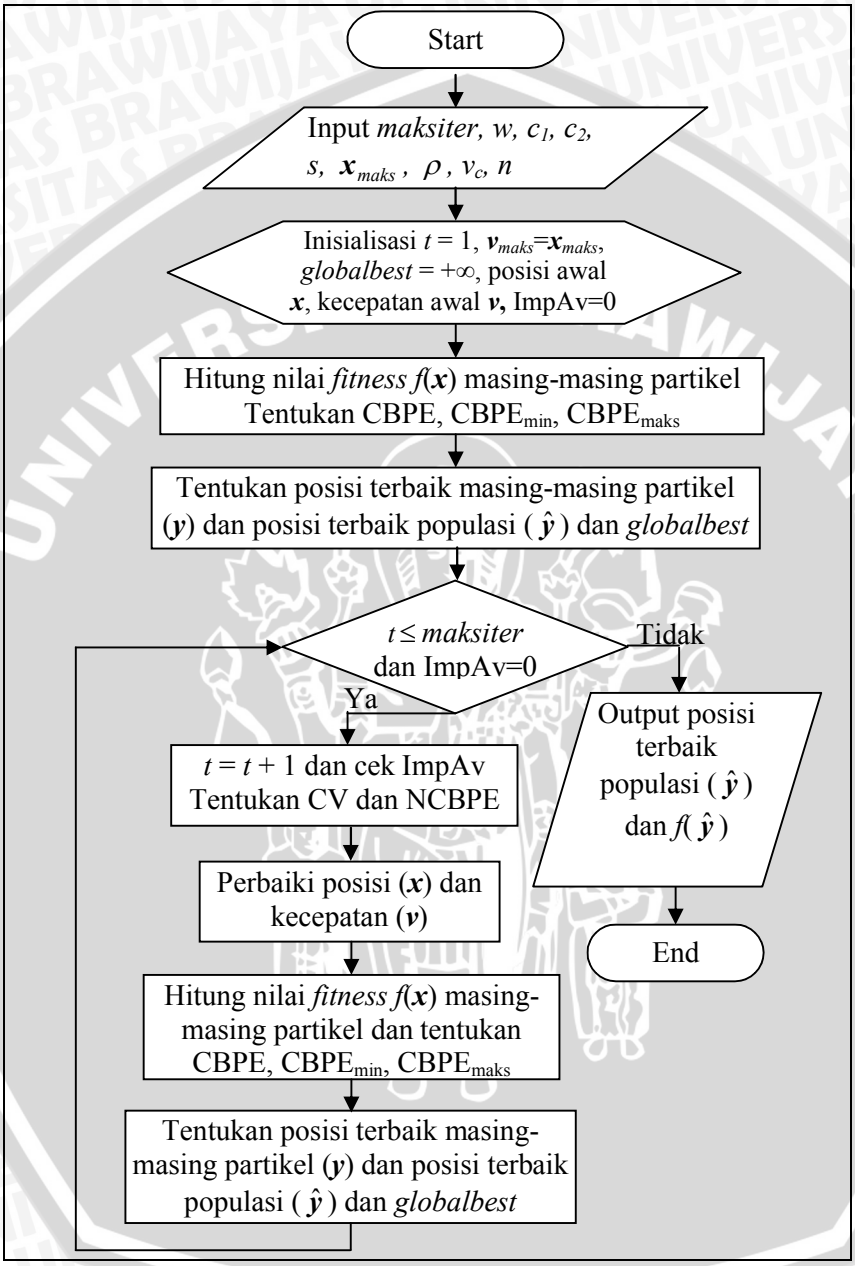
(c) if $currentvalue_i < globalbest$then

(i) $globalbest \leftarrow currentvalue_i$

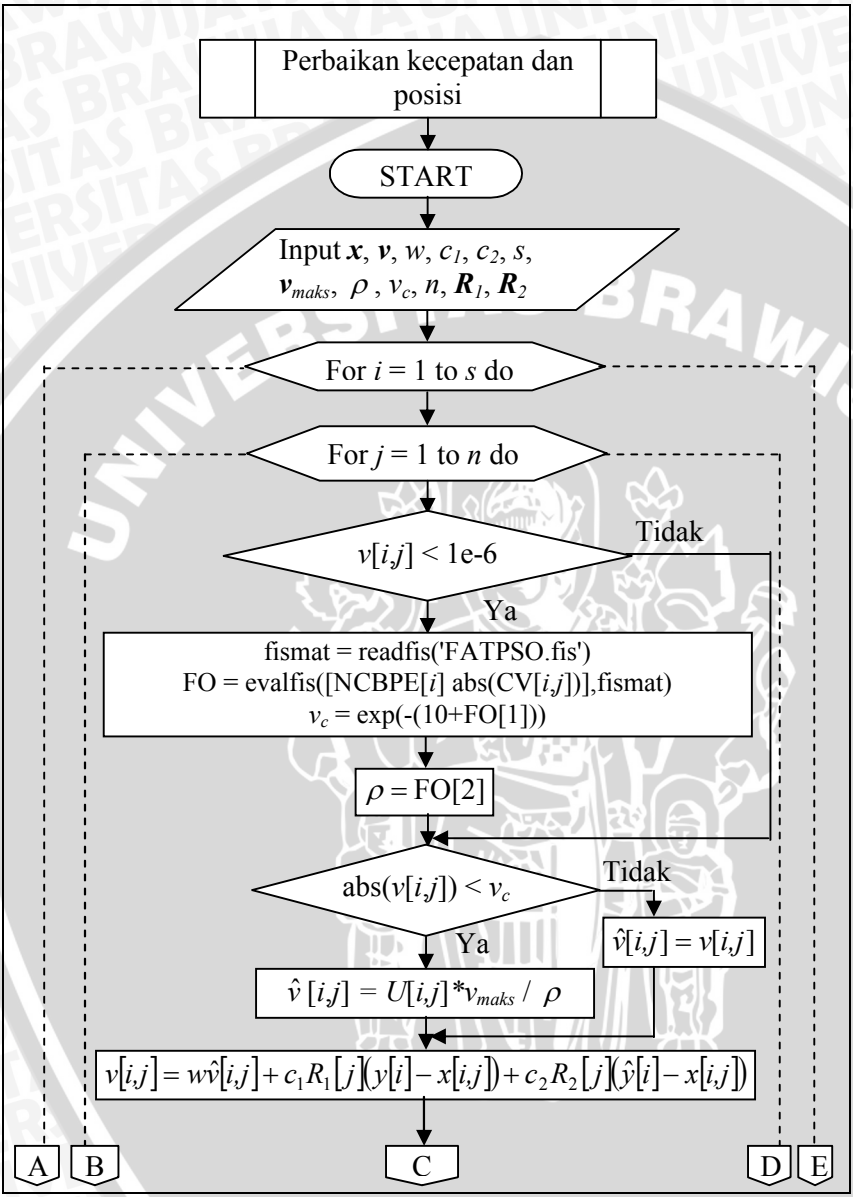
(ii) $\hat{\boldsymbol{y}} \leftarrow \boldsymbol{x}_i$

Output : $\hat{\boldsymbol{y}}, f(\hat{\boldsymbol{y}})$

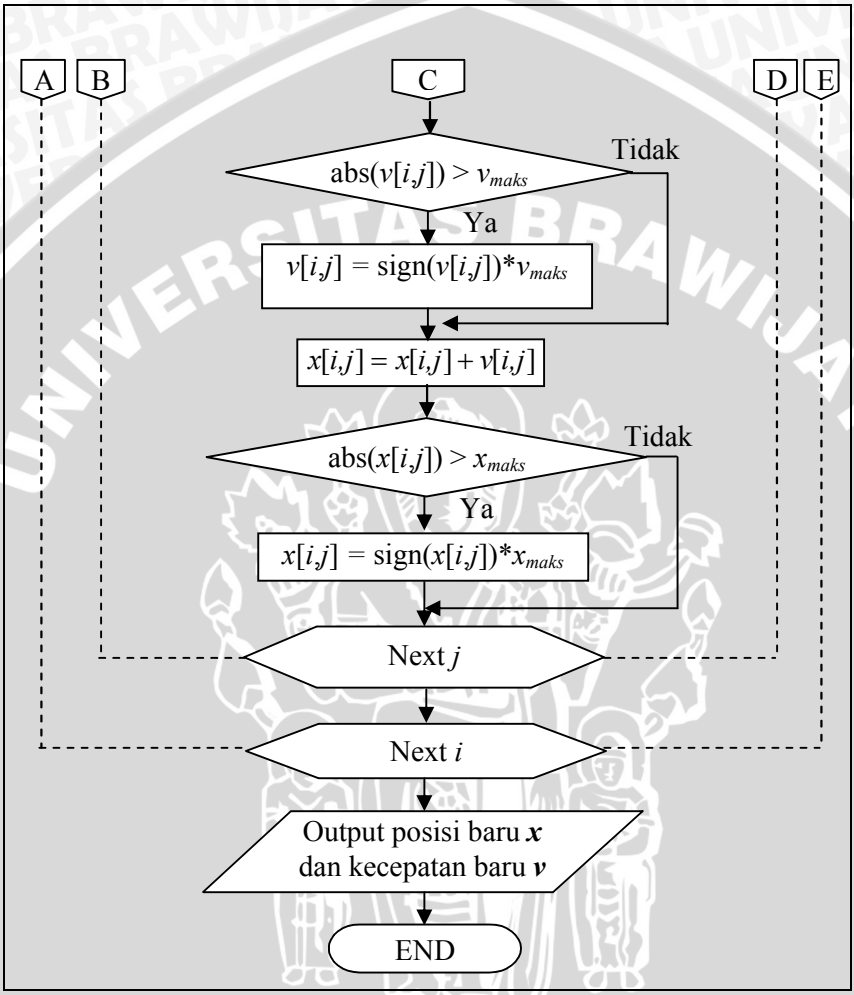
Algoritma FATPSO menggunakan perintah readfis dan evalfis. Readfis adalah perintah yang digunakan untuk membaca file *.fis, yaitu file yang menyimpan sistem *fuzzy* yang digunakan sebagai pengontrol sedangkan evalfis adalah perintah yang digunakan untuk memproses hasil pembacaan sistem *fuzzy*. Perintah readfis mengolah variabel masukan, dalam hal ini adalah *NCBPE* dan *CV*, menjadi variabel keluaran FO. Apabila dalam sistem *fuzzy* yang dirancang menggunakan 2 keluaran, maka FO berbentuk matriks 1×2 . Contoh perhitungan manual algoritma FATPSO dapat dilihat pada lampiran 2.



Gambar 3.6 Diagram alir algoritma FATPSO



Gambar 3.7(a) Diagram alir perbaikan kecepatan dan posisi menggunakan algoritma FATPSO



Gambar 3.7(b) Diagram alir perbaikan kecepatan dan posisi menggunakan algoritma FATPSO

3.3 Implementasi Program FATPSO

Pengimplementasian algoritma FATPSO untuk masalah optimasi dengan menggunakan *test function* dibuat dengan MATLAB 6.5. Hasil dan waktu komputasi algoritma FATPSO dibandingkan dengan hasil dan waktu komputasi algoritma TPSO dan algoritma PSO.

3.3.1 Penentuan Parameter

Parameter yang ada, baik itu pada algoritma PSO, TPSO, maupun FATPSO, sangat menentukan keberhasilan algoritma dalam menyelesaikan masalah optimasi. Oleh karena itu, dalam menentukan parameter yang ada, terutama parameter bobot inersia yang mempengaruhi kekonvergenan, tidak boleh sembarangan.

Untuk algoritma PSO dan FATPSO, parameter yang ditentukan adalah sebagai berikut:

- Ukuran *swarm* : 30
- Kemampuan individu (c_1) : 1.49
- Kemampuan sosial (c_2) : 1.49
- *Inertia weight* (w) : 0.7
- Kecepatan maksimum v_{maks} : x_{maks}

sedangkan untuk algoritma TPSO ditambah dua parameter berikut:

- Batasan kecepatan minimum (v_c) : 1e-6
- Faktor penyekala (ρ) : v_{max}

(Bergh, 2006)

Pengambilan nilai parameter c_1, c_2 dan w tersebut memenuhi kondisi

(i) $\sqrt{w} = \sqrt{0.7} = 0.8367 < 1$ dan

(ii) $0.0267 = 1 + w - 2\sqrt{w} < c_1 + c_2 = 2.98 < 1 + w + 2\sqrt{w} = 3.3733$

yang menjamin ketiga algoritma tersebut konvergen.

Pada kriteria penghentian iterasi ImpAv juga terdapat parameter yang harus ditentukan. Dalam implementasi program ini, parameter ditentukan sebagai berikut:

- Banyaknya iterasi pengecekan g : 50
- Batasan minimum peningkatan rata-rata m : 0.0001

(Zielinski, dkk. 2005).

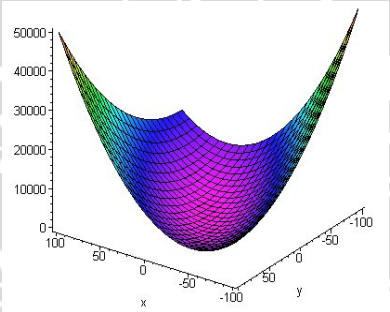
3.3.2 Hasil dan Analisis Keluaran Program

Dalam mengimplementasikan algoritma FATPSO pada *test function*, jumlah iterasi maksimum yang digunakan berbeda-beda, seperti halnya nilai x_{maks} yang disesuaikan berdasarkan *test function*-nya. Fungsi-fungsi yang dipakai sebagai *test function* untuk menguji algoritma FATPSO adalah sebagai berikut:

1. Fungsi Quadric

   Fungsi ini merupakan fungsi unimodal berdimensi *n* dengan titik minimum global di $\boldsymbol{x}^* = \boldsymbol{0}$ dan $f(\boldsymbol{x}^*) = 0$. Fungsi Quadric dengan *n*=2 dapat dilihat pada Gambar 3.8.

   $$f = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2$$

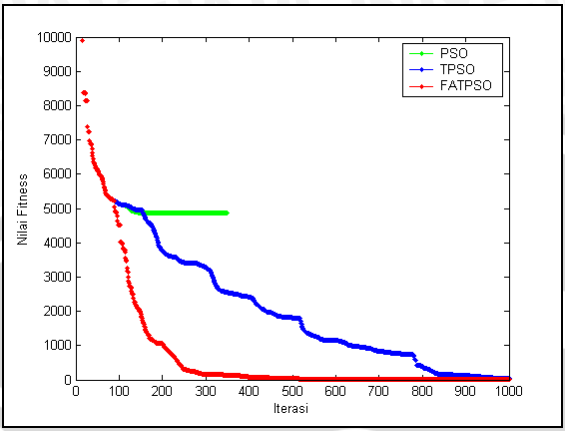


Gambar 3.8 Fungsi Quadric dengan *n*=2

Dalam mengimplementasikan algoritma dengan menggunakan fungsi Quadric, maksimum iterasi yang digunakan adalah 1000 iterasi dengan domain daerah pencarian $[-100,100]^n$.

Tabel 3.2 Hasil Uji Algoritma Menggunakan Fungsi Quadric

Algoritma	10 Dimensi		20 Dimensi	
	Waktu komputasi	Nilai Fungsi	Waktu komputasi	Nilai Fungsi
PSO	1.281	9.41099	1.219	4861.3
TPSO	10.985	2.2455e-005	6.703	46.076
FATPSO	22.781	0.00038074	18.062	0.504128



Gambar 3.9 Hasil minimasi fungsi Quadric 20 dimensi

Berdasarkan Tabel 3.2 dan Gambar 3.9, terlihat bahwa dengan menggunakan parameter-parameter yang telah ditentukan pada subbab 3.3.1, algoritma PSO mengalami kondisi stagnan mulai iterasi ke-350 yang ditandai dengan nilai *fitness* yang selalu tetap. Berbeda dengan kedua algoritma yang lain, untuk 10 dimensi hasil yang diperoleh algoritma TPSO lebih baik dari algoritma FATPSO, tetapi untuk 20 dimensi hasil FATPSO terbaik dari ketiga algoritma. Kelemahan dari algoritma FATPSO adalah waktu komputasi yang lebih lama daripada kedua algoritma yang lain. Perilaku algoritma FATPSO juga menunjukkan bahwa algoritma ini lebih peka dalam menanggapi kondisi yang ada daripada algoritma TPSO. Untuk mengetahui pengaruh perubahan parameter, perhatikan Tabel 3.3 berikut.

Tabel 3.3 (a) Hasil Uji Algoritma Menggunakan Fungsi Quadric untuk Pengubahan Beberapa Parameter

Parameter $s = 30$, $w = 0.7$, $c_1 = c_2 = 2$				
Algoritma	10 Dimensi		20 Dimensi	
	Waktu komputasi	Nilai Fungsi	Waktu komputasi	Nilai Fungsi
PSO	0.641	15.3644	0.672	2857.75
TPSO	1.484	0.388255	2.188	602.861
FATPSO	3.719	0.00808714	5.328	207.144

Tabel 3.3 (b) Hasil Uji Algoritma Menggunakan Fungsi Quadric untuk Pengubahan Beberapa Parameter

Parameter $s = 15$, $w = 0.7$, $c_1 = c_2 = 1.49$				
Algoritma	10 Dimensi		20 Dimensi	
	Waktu komputasi	Nilai Fungsi	Waktu komputasi	Nilai Fungsi
PSO	0.531	720.318	0.516	6030.49
TPSO	1.015	45.8199	2.5	208.903
FATPSO	1.719	7.35218	2.782	14.2777
Parameter $s = 30$, $w = 0.3$, $c_1 = c_2 = 1.49$				
Algoritma	10 Dimensi		20 Dimensi	
	Waktu komputasi	Nilai Fungsi	Waktu komputasi	Nilai Fungsi
PSO	0.562	308.164	0.922	2756.06
TPSO	5.078	20.2107	9.375	357.932
FATPSO	17.437	0.0943789	22.062	12.7686
Parameter $s = 30$, $w = 1$, $c_1 = c_2 = 1.49$				
Algoritma	10 Dimensi		20 Dimensi	
	Waktu komputasi	Nilai Fungsi	Waktu komputasi	Nilai Fungsi
PSO	0.953	396.294	0.984	8288.19
TPSO	1.875	396.294	2.985	8288.19
FATPSO	2.812	396.294	4.625	8288.19
Parameter $s = 30$, $w = 1$, $c_1 = c_2 = 2$				
Algoritma	10 Dimensi		20 Dimensi	
	Waktu komputasi	Nilai Fungsi	Waktu komputasi	Nilai Fungsi
PSO	1.032	597.332	0.578	6957.35
TPSO	1.343	597.332	2.563	6957.35
FATPSO	1.922	597.332	4.265	6957.35

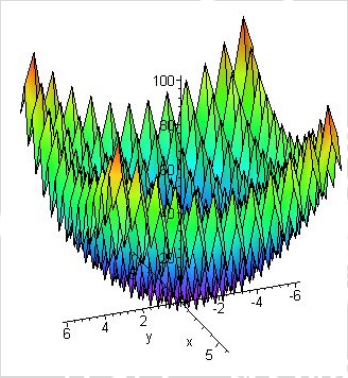
Berdasarkan Tabel 3.3 tersebut dapat diketahui bahwa pemilihan parameter berpengaruh terhadap hasil algoritma. Pemilihan nilai parameter $w = 1$ menyebabkan algoritma tidak konvergen sedangkan $w = 0.3$ menyebabkan hasil yang diperoleh kurang optimal. Jumlah populasi yang sedikit ($s = 15$) menyebabkan nilai *fitness* yang diperoleh masih terlalu besar. Parameter c_1 dan c_2

juga berpengaruh pada hasil yang diperoleh, pemilihan parameter yang tidak tepat menyebabkan hasil yang diperoleh kurang optimal. Berdasarkan hasil tersebut dapat disimpulkan bahwa fungsi Quadric dapat diselesaikan menggunakan algoritma TPSO untuk dimensi yang kecil dan algoritma FATPSO untuk dimensi yang lebih besar, dimana hasil tersebut bergantung pada pemilihan parameter.

2. Fungsi Rastrigin

   Fungsi ini merupakan fungsi multimodal berdimensi *n* dengan titik minimum global di $\boldsymbol{x}^* = \boldsymbol{0}$ dan $f(\boldsymbol{x}^*) = 0$. Fungsi Rastrigin dengan *n*=2 dapat dilihat Gambar 3.10.

   $$f = \sum_{i=1}^{n} \left(x_i^{\,2} - 10\cos(2\pi x_i) + 10 \right)$$



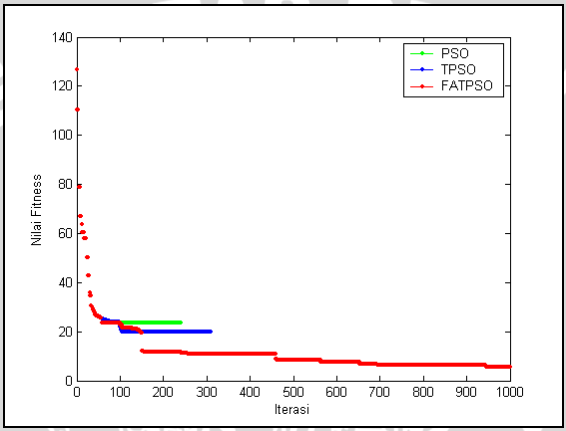
Gambar 3.10 Fungsi Rastrigin dengan *n*=2

Jumlah iterasi yang digunakan adalah 1000 dengan domain daerah pencarian $[-5.12,5.12]^n$. Hasil yang diperoleh sebagai berikut.

Tabel 3.4 Hasil Uji Algoritma Menggunakan Fungsi Rastrigin

Algoritma	10 Dimensi		20 Dimensi	
	Waktu komputasi	Nilai Fungsi	Waktu komputasi	Nilai Fungsi
PSO	0.671	23.6004	0.594	48.7828
TPSO	2.625	19.8992	12.187	34.8258
FATPSO	26.016	5.53545	55.969	30.7097

Berdasarkan Tabel 3.4, dengan menggunakan parameter yang telah ditentukan pada subbab 3.3.1, dapat dilihat bahwa algoritma FATPSO terbaik dari ketiga algoritma untuk 10 dan 20 dimensi. Perilaku pergerakan partikel masing-masing algoritma dapat dilihat Gambar 3.11. Berdasarkan gambar tersebut dapat diketahui bahwa algoritma PSO mengalami kondisi stagnan mulai iterasi ke-241 sedangkan algoritma TPSO mulai iterasi ke-309.



Gambar 3.11 Hasil minimasi fungsi Rastrigin 10 dimensi

Dengan melakukan perubahan parameter (lihat Tabel 3.5) dapat diketahui bahwa hasil yang diperoleh algoritma FATPSO tetap lebih optimal, kecuali untuk pemilihan parameter $w = 1$, karena ketiga algoritma tidak konvergen. Berdasarkan hasil tersebut dapat disimpulkan bahwa fungsi Rastrigin, yang merupakan fungsi multimodal, dapat diselesaikan dengan baik oleh algoritma FATPSO.

Tabel 3.5 (a) Hasil Uji Algoritma Menggunakan Fungsi Rastrigin untuk Pengubahan Beberapa Parameter

Parameter $s = 30$, $w = 0.7$, $c_1 = c_2 = 2$				
Algoritma	10 Dimensi		20 Dimensi	
	Waktu komputasi	Nilai Fungsi	Waktu komputasi	Nilai Fungsi
PSO	1.125	5.97249	1.25	105.905
TPSO	14.516	5.96979	21.672	84.3046
FATPSO	16.109	4.40936	50.531	41.9253

Tabel 3.5 (b) Hasil Uji Algoritma Menggunakan Fungsi Rastrigin untuk Pengubahan Beberapa Parameter

Parameter s = 15, w = 0.7, c1 = c2 = 1.49				
Algoritma	10 Dimensi		20 Dimensi	
	Waktu komputasi	Nilai Fungsi	Waktu komputasi	Nilai Fungsi
PSO	1.14	12.6087	1.219	85.4041
TPSO	12.453	10.9446	10.109	66.6641
FATPSO	17.969	4.24408	22.625	54.3618
Parameter $s = 30$, $w = 0.3$, $c_1 = c_2 = 1.49$				
Algoritma	10 Dimensi		20 Dimensi	
	Waktu komputasi	Nilai Fungsi	Waktu komputasi	Nilai Fungsi
PSO	1.766	60.3081	1.719	125.862
TPSO	26.453	60.3081	58.375	125.862
FATPSO	37.687	28.8079	66.937	68.6878
Parameter $s = 30$, $w = 1$, $c_1 = c_2 = 1.49$				
Algoritma	10 Dimensi		20 Dimensi	
	Waktu komputasi	Nilai Fungsi	Waktu komputasi	Nilai Fungsi
PSO	0.687	34.289	1.156	132.085
TPSO	1.5	34.289	7.687	132.085
FATPSO	2.797	34.289	9.5	132.085
Parameter $s = 30$, $w = 1$, $c_1 = c_2 = 2$				
Algoritma	10 Dimensi		20 Dimensi	
	Waktu komputasi	Nilai Fungsi	Waktu komputasi	Nilai Fungsi
PSO	1.594	38.2359	1.156	112.733
TPSO	3.109	38.2359	7.687	112.78
FATPSO	3.797	38.2359	9.5	115.62

Titik yang dicapai algoritma dengan menggunakan fungsi Quadric dan Rastrigin dapat dilihat pada lampiran 3.

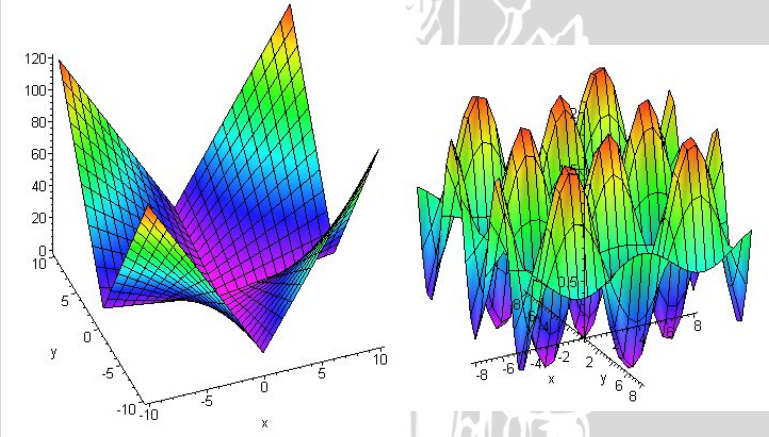
3. Fungsi Schewefel 2.22
   Fungsi ini merupakan fungsi unimodal berdimensi *n* dengan titik minimum global di $\boldsymbol{x}^* = \boldsymbol{0}$ dan $f(\boldsymbol{x}^*) = 0$.

$$f = \sum_{i=1}^{n} |x_i| + \prod_{i=1}^{n} |x_i|$$

4. Fungsi Griewank
   Fungsi ini merupakan fungsi multimodal berdimensi *n* dengan titik minimum global di $\boldsymbol{x}^* = \boldsymbol{0}$ dan $f(\boldsymbol{x}^*) = 0$.

$$f = \frac{1}{4000} \sum_{i=1}^{n} x_i^{\,2} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

Gambaran mengenai fungsi Schewefel 2.22 dan fungsi Griewank untuk *n*=2 dapat dilihat pada Gambar 3.12



Gambar 3.12 Fungsi Schewefel 2.22 dan fungsi Griewank dengan *n*=2

Jumlah iterasi yang digunakan pada fungsi Schwefel 2.22 dan fungsi Griewank adalah 1000 iterasi. Untuk fungsi Schwefel 2.22 domain daerah pencarian adalah $[-10,10]^n$ dengan jumlah dimensi 25 dan 30 sedangkan untuk fungsi Griewank, domain daerah pencarian $[-300,300]^n$ dengan n = 10 dan n = 20. Hasil kedua fungsi, dengan menggunakan parameter yang telah ditentukan, dapat dilihat pada Tabel 3.6. Berdasarkan tabel tersebut diketahui bahwa algoritma FATPSO tetap lebih baik dari algoritma TPSO untuk dimensi yang lebih besar.

Tabel 3.6 Hasil Uji Algoritma Menggunakan Fungsi Schewefel 2.22 dan Griewank

Fungsi Schewefel 2.22				
Algoritma	25 Dimensi		30 Dimensi	
	Waktu komputasi	Nilai Fungsi	Waktu komputasi	Nilai Fungsi
PSO	1.234	6.2012	0.672	22.7935
TPSO	17.781	0.05705	5.125	1.8409
FATPSO	31.625	0.16522	22.531	0.22606
Fungsi Griewank				
Algoritma	10 Dimensi		20 Dimensi	
	Waktu komputasi	Nilai Fungsi	Waktu komputasi	Nilai Fungsi
PSO	1.766	0.14031	1.281	1.31639
TPSO	30.14	0.11805	6.156	0.112979
FATPSO	34.641	0.077125	28.016	0.0781024

BAB IV
KESIMPULAN DAN SARAN

1.1 Kesimpulan

Berdasarkan hasil dan pembahasan yang diperoleh, terdapat beberapa kesimpulan yang dapat diambil, yaitu

1. Algoritma *Turbulence Particle Swarm Optimization* (TPSO), yang merupakan algoritma perbaikan PSO, dijamin kekonvergenannya.
2. Algoritma *Fuzzy Adaptive Turbulence Particle Swarm Optimization* (FATPSO) adalah algoritma TPSO yang menggunakan *Fuzzy Logic Controller* (FLC) sebagai pengontrol. FLC ini digunakan untuk mengontrol parameter batasan kecepatan minimum v_c dan faktor penyekala ρ yang terdapat pada algoritma TPSO. Variabel masukan FLC adalah *CV* dan *NCBPE* dengan variabel keluaran V_{ck} dan ρ.
   - *CV* adalah kecepatan partikel.
   - $NCBPE = \dfrac{CBPE - CBPE_{\min}}{CBPE_{maks} - CBPE_{\min}}$, dengan *CBPE* adalah nilai *fitness*, $CBPE_{min}$ adalah nilai *fitness* terbaik dan $CBPE_{maks}$ adalah nilai *fitness* terburuk dari masalah optimasi.
   - Nilai V_{ck} digunakan untuk mencari nilai v_c, yaitu $v_c = e^{-[10(1+V_{ck})]}$.
3. Berdasarkan hasil implementasi program menggunakan beberapa *test function* diketahui bahwa algoritma FATPSO memberikan hasil yang lebih optimal apabila dibandingkan dengan algoritma TPSO, tetapi memerlukan waktu komputasi yang lebih lama.

1.2 Saran

Saran yang dapat diberikan untuk pengembangan algoritma FATPSO adalah

1. Algoritma FATPSO diselidiki lebih lanjut untuk meningkatkan efisiensi algoritma, misalnya dengan menyelidiki fungsi keanggotaan dan basis aturan yang digunakan pada FLC.

2. Algoritma FATPSO dibandingkan dengan algoritma optimasi yang lain, seperti algoritma genetika dan *Simulated Annealing*.
3. Perlu diteliti lebih lanjut mengenai parameter dalam kriteria penghentian ImpAv.

DAFTAR PUSTAKA

Abraham, A. dan H. Liu. 2007. *Turbulent Particle Swarm Optimization with Fuzzy Parameter Tuning*. Int. J. Innovative Computing and Applications, Vol. 1, No. 1, pp.39–47.

Bergh, V. D. 2006. *An Anaysis of Particle Swarm Optimizers*. PhD thesis, University of Pretoria. South Africa.

Bronshtein, I.N. dan K. A. Semendyayev. 2007. *Handbook of Mathematics*. Springer. Berlin.

Bronson, R. 1996. *Teori dan Soal-Soal Operation Research, Alih Bahasa : Hans J. Waspakrik*. Erlangga. Jakarta.

Dajan, A. 1986. *Pengantar Metode Statistik Jilid II*. PT. Pustaka LP3ES. Indonesia.

Grossman, S. I. 1995. *Multivariable Calculus. Linear Algebra. and Differential Equations*. Saunders College Publishing. London.

Jang, J.S.R., C. Sun dan E. Mizutani. 1997. *Neuro-Fuzzy and Soft Computing*. Prentice-Hall International. Shanghai.

Leader, J.J. 2004. *Numerical Analysis and Scientific Computation International Edition*. Pearson Addison-Weasley. New York.

Liu, C. L. 1986. *Elements of Discrete Mathematics Jilid 2*. Mc. Graw Hill. Singapura.

Luknanto, D. 2000. *Pengantar Optimasi Nonlinier*. Jurusan Teknik Sipil Fakultas Teknik Universitas Gajah Mada. Yogyakarta.

Mathews, J.H. dan D. F. Kurtis. 2004. *International Edition Numerical Methods Using Matlab Fourth Edition*. Pearson Education International. USA.

Parsopoulos, K. E. 2002. *Recent Approaches to Global Optimization Problems through Particle Swarm Optimization*. Natural Computing, No. 1, pp.235–306.

Purcell, E. J. 1994. *Kalkulus dan Geometri Analitik Jilid 2*. Erlangga. Jakarta.

Shi, Y. 2004. *Particle Swarm Optimization*. Electronic Data Systems, Inc. USA.

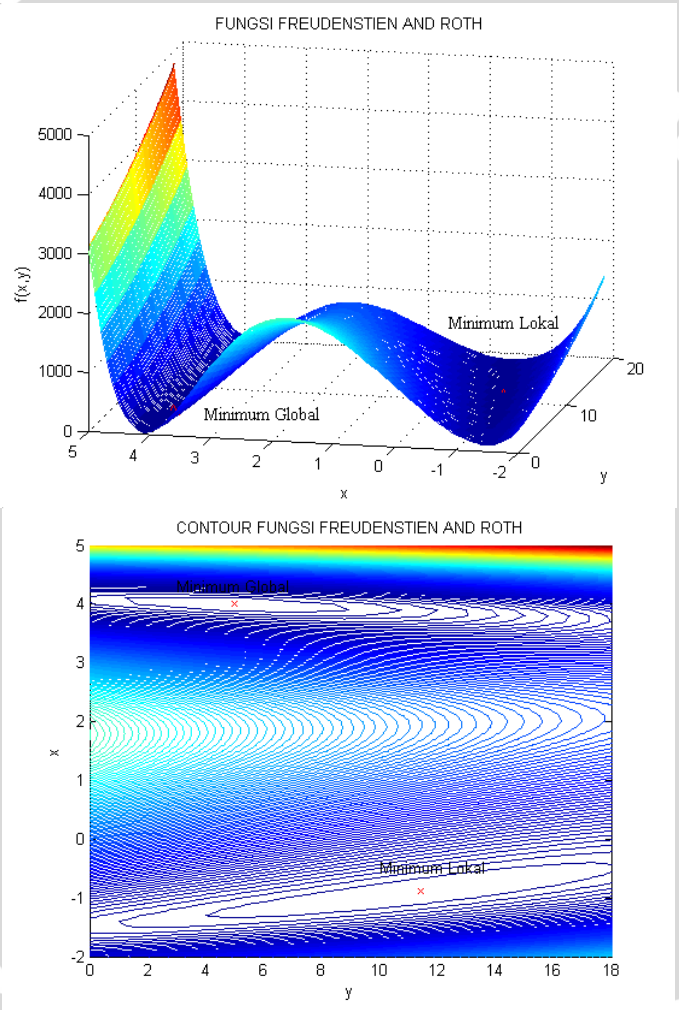
Weise, T. 2009. *Global Optimization Algorithms - Theory and Application*. http://www.it-weise.de/projects/book.pdf. Tanggal akses: 12 Februari 2009.

Zielinski, K., D. Peters dan R. Laur. 2005. *Stopping Criteria for Single Objective Optimization*. University of Bremen. Germany.

Lampiran 1
Contoh masalah optimasi yang diselesaikan menggunakan algoritma TPSO secara manual

Misalkan masalah optimasi yang diselesaikan adalah meminimumkan fungsi Freudenstien dan Roth, yaitu

$$f(x,y) = (-13 + x + ((5-y)y - 2)y)^2 + (-29 + x + ((y+1)y - 14)y)^2.$$



Fungsi ini memiliki nilai minimum global di titik (4,5) dengan nilai fungsi $f = 0$ dan minimum lokal di (11.41,-0.89) dengan $f = 48.98$.

Langkah-langkah yang dilakukan untuk mengimplementasikan algoritma PSO adalah sebagai berikut:

1. Menentukan nilai parameter
   Parameter algoritma TPSO yang ditentukan yaitu:
   - Ukuran populasi : 5
   - Kemampuan individu (c_1) : 1.49
   - Kemampuan sosial (c_2) : 1.49
   - Bobot inersia (w) : 0.7
   - Batasan ruang pencarian (x_{maks}) : 20
   - Kecepatan maksimum (v_{maks}) : 1
   - Batasan kecepatan minimum (v_c) : 10^{-6}
   - Faktor penyekala (ρ) : 50
   - Peubah acak R_1 dan R_2

   R_1

Partikel ke-i	1	2	3	4	5
x	0.7948	0.5226	0.1730	0.2714	0.8757
y	0.9568	0.8801	0.9797	0.2523	0.7373

   R_2

Partikel ke-i	1	2	3	4	5
x	0.1365	0.8939	0.2987	0.2844	0.0648
y	0.0118	0.1991	0.6614	0.4692	0.9883

2. Melakukan inisialisasi awal.
   Posisi awal

Partikel ke-i	1	2	3	4	5
x	0.1656	0.0310	-0.1342	0.1596	0.0596
y	-0.1530	-0.3321	-0.5481	0.5207	0.2811

   Kecepatan awal

Partikel ke-i	1	2	3	4	5
Arah x	-0.5819	0.5667	-0.0778	0.5884	0.2057
Arah y	-0.2404	0.3617	0.1357	-0.8816	-0.8995

3. Menentukan posisi terbaik masing-masing partikel (***y***).
   Karena belum melakukan perbaikan posisi, maka posisi terbaik (***y***) adalah posisi awal partikel.

Partikel ke-*i*	1	2	3	4	5
x	0.1656	0.0310	-0.1342	0.1596	0.0596
y	-0.1530	-0.3321	-0.5481	0.5207	0.2811

4. Menghitung nilai *fitness* masing-masing partikel (*currentvalue*), yaitu dengan menghitung nilai fungsi masing-masing partikel.

Partikel ke-*i*	1	2	3	4	5
Nilai *f*(*x*,*y*)	865.4	725.1	562.3	1436.3	1246.5

5. Menentukan nilai *fitness* terbaik masing-masing partikel (*bestvalue*).
   Karena belum melakukan perbaikan posisi maka *bestvalue* sama dengan *currentvalue.*

Partikel ke-*i*	1	2	3	4	5
Nilai *fitness*	865.4	725.1	562.3	1436.3	1246.5

6. Menentukan posisi terbaik polulasi ($\hat{\boldsymbol{y}}$).
   Posisi terbaik didapat oleh partikel ke-3 yaitu (-0.1342, -0.5481), karena partikel ke-3 memiliki nilai *fitness* yang paling baik.

7. Menentukan nilai terbaik populasi (*globalbest*), yaitu nilai *fitness* partikel ke-3, sehingga diperoleh
   Globalbest = *f*(-0.1342, -0.5481) = 562.3198.

8. Melakukan iterasi sebagai berikut.
   a. Memperbaiki kecepatan dengan tahap-tahap sebagai berikut:
      i. Menyelidiki apakah kecepatan masing-masing partikel lebih kecil dari batasan kecepatan minimum (v_c) berdasarkan persamaan (3.2).
         Karena kecepatan masing-masing partikel lebih besar dari batasan nilai kecepatan minimum v_c yaitu 1e-6, maka kecepatannya tetap, sehingga nilai $\hat{v}$ adalah

Partikel ke-*i*	1	2	3	4	5
Arah *x*	-0.5819	0.5667	-0.0778	0.5884	0.2057
Arah *y*	-0.2404	0.3617	0.1357	-0.8816	-0.8995

ii. Melakukan perbaikan kecepatan berdasarkan persamaan (3.1), sehingga diperoleh kecepatan baru, yaitu

Partikel ke-*i*	1	2	3	4	5
Arah *x*	-0.4683	0.1766	-0.0545	0.2874	0.1253
Arah *y*	-0.1752	0.1891	0.0950	-1.3644	-1.8507

Sebagai contoh, untuk mencari nilai kecepatan baru partikel ke-1 dilakukan perhitungan sebagai berikut.

$$v = 0.7\begin{bmatrix} -0.5819 \\ -0.2404 \end{bmatrix} + 1.49\begin{bmatrix} 0.7948 \\ 0.9568 \end{bmatrix}.\begin{bmatrix} 0.1656 - 0.1656 \\ -0.153 - (-0.153) \end{bmatrix} +$$

$$1.49\begin{bmatrix} 0.1365 \\ 0.0118 \end{bmatrix}.\begin{bmatrix} -0.1342 - 0.1656 \\ -0.5481 - (-0.153) \end{bmatrix}$$

$$= \begin{bmatrix} -0.4073 \\ -0.1683 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -0.0610 \\ -.0069 \end{bmatrix}$$

$$= \begin{bmatrix} -0.4683 \\ -0.1752 \end{bmatrix}$$

iii. Menyelidiki apakah nilai kecepatan partikel melebihi batasan kecepatan maksimum (v_{maks}).
Kecepatan partikel ke-4 dan ke-5 pada arah y melebihi batasan kecepatan maksimum v_{maks}, yaitu 1, maka kecepatan kedua partikel tersebut diganti menjadi -1. Nilai -1 didapat dari perhitungan sign(v)*v_{maks}.

Partikel ke-*i*	1	2	3	4	5
Arah *x*	-0.4683	0.1766	-0.0545	0.2874	0.1253
Arah *y*	-0.1752	0.1891	0.0950	-1	-1

b. Memperbaiki posisi sesuai dengan persamaan (3.3), sehingga diperoleh posisi baru partikel, yaitu

Partikel ke-*i*	1	2	3	4	5
x	-0.3027	0.2076	-0.1887	0.4470	0.1850
y	-0.3282	-0.1430	-0.4531	-0.4793	-0.7189

Misalkan untuk mencari posisi baru partikel ke-1, caranya adalah

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0.1656 \\ -0.1530 \end{bmatrix} + \begin{bmatrix} -0.4683 \\ -0.1752 \end{bmatrix}$$

$$= \begin{bmatrix} -0.3027 \\ -0.3282 \end{bmatrix}$$

c. Menyelidiki posisi partikel berdasarkan persamaan (3.4). Posisi baru partikel tidak ada yang keluar domain, sehingga nilainya tetap.

d. Menghitung nilai *fitness* masing-masing partikel (*currentvalue*).

Partikel ke-*i*	1	2	3	4	5
Nilai *f*(*x*,*y*)	752.67	870.57	641.37	578.75	417.04

e. Menentukan posisi terbaik masing-masing partikel (***y***). Caranya adalah dengan membandingkan nilai *fitness* pada posisi baru dengan nilai *fitness* pada posisi terbaiknya (*bestvalue*). Jika nilai *fitness* pada posisi baru lebih baik, maka posisi terbaik diganti dengan posisi baru.

Partikel ke-*i*	1	2	3	4	5
x	-0.3027	0.0310	-0.1342	0.4470	0.1850
y	-0.3282	-0.3321	-0.5481	-0.4793	-0.7189

Sebagai contoh, nilai *fitness* partikel ke-1, yaitu 752.67 lebih baik dari *bestvalue*-nya, 865.4, maka nilai ***y*** partikel ke-1 yang semula (0.1656, -0.1530) diganti dengan posisi baru partikel ke-1 yaitu (-0.3027,-0.3282).

f. Menentukan nilai *fitness* terbaik masing-masing partikel (*bestvalue*), yaitu dengan menghitung nilai fungsi partikel pada posisi terbaiknya.

Partikel ke-*i*	1	2	3	4	5
Nilai *f*(*x*,*y*)	752.67	725.1478	562.3198	578.75	417.04

g. Menentukan posisi terbaik polulasi ($\hat{\boldsymbol{y}}$).
Dari hasil perhitungan nilai *fitness* posisi terbaik, dapat diketahui bahwa nilai *fitness* terbaik dicapai oleh partikel ke-5, sehingga posisi terbaik populasi adalah posisi partikel ke-5 yaitu (0.1850,-0.7189).

h. Menentukan nilai terbaik populasi (*globalbest*).
Nilai terbaik populasi adalah nilai *fitness* partikel pada posisi terbaik populasi, yaitu 417.04.

i. Melakukan langkah 8 sampai kondisi berhenti dipenuhi. Hasil algoritma berupa posisi terbaik populasi $\hat{\boldsymbol{y}}$ dan nilai *globalbest*.

Berdasarkan hasil perhitungan manual tersebut, masih belum terlihat pengaruh batasan kecepatan minimum karena kecepatan yang dimiliki partikel masih lebih besar dari batasan nilai yang ditetapkan. Agar langkah tersebut dapat berjalan, maka kecepatan partikel harus lebih kecil dari batasan nilai kecepatan minimum. Misalkan dari hasil iterasi tersebut diperoleh nilai kecepatan partikel sebagai berikut.

Partikel ke-*i*	1	2	3	4	5
Arah *x*	0.028	1.6e-5	3.5e-3	-0.060	-4e-4
Arah *y*	-5.2e-3	-7.9e-4	0.372	9.4e-7	-2.6e-3

Berdasarkan hasil tersebut dapat diketahui bahwa kecepatan partikel ke-4 pada arah *y* lebih kecil dari batasan nilai kecepatan minimum, yaitu 10^{-6}, maka dilakukan perhitungan kecepatan $\hat{v}$ berdasarkan persamaan (3.2), yaitu

$$\hat{v} = u[-1,1] * v_{maks} / \rho$$
$$= -0.6522 * 1/50$$
$$= -0.01304$$

sehingga kecepatan partikel ke-4 terhadap arah *y* menjadi -0.01304.

Lampiran 2
Contoh masalah optimasi yang diselesaikan menggunakan algoritma FATPSO secara manual

Misalkan pada contoh kasus optimasi pada lampiran 1, setelah beberapa iterasi, diperoleh nilai kecepatan, posisi dan nilai *fitness* masing-masing partikel (*currentvalue*) seperti berikut:

- Kecepatan partikel

Partikel ke-i	1	2	3	4	5
Arah *x*	0.028	1.6e-5	3.5e-3	-0.060	-4e-4
Arah *y*	-5.2e-3	-7.9e-4	0.372	9.4e-7	-2.6e-3

- Posisi partikel

Partikel ke-i	1	2	3	4	5
x	11.447	11.413	11.415	11.373	11.412
y	-0.886	-0.897	-0.594	-0.897	-0.898

- Nilai *fitness* partikel

Partikel ke-i	1	2	3	4	5
Nilai *fitness*	49.021	48.984	85.813	48.987	48.985

Dari hasil tersebut diketahui bahwa kecepatan partikel ke-4 pada arah *y* lebih kecil dari batasan nilai kecepatan minimum, yaitu 1e-6, maka dilakukan perhitungan nilai kecepatan baru berdasarkan persamaan (3.2). Sebelumnya, harus dicari dulu batasan nilai kecepatan minimum v_c dan faktor penyekala ρ menggunakan *Fuzzy Logic Controller* yang telah dirancang. Langkah-langkah yang dilakukan adalah sebagai berikut:

1. Menentukan masukan *Fuzzy Logic Controller*.
   Masukan *Fuzzy Logic Controller* berupa *NCBPE* dan *CV*. Langkah untuk memperoleh *NCBPE* adalah sebagai berikut:
   - Menghitung nilai *CBPE* yang merupakan nilai *fitness* partikel ke-4.
     $CBPE = f(11.412,-0.897) = 48.987$.
   - Menentukan $CBPE_{maks}$, yaitu nilai *fitness* terbesar.
     Nilai *fitness* terbesar dimiliki oleh partikel ke-3, yaitu 85.813.

- Menentukan $CBPE_{min}$, yaitu nilai *fitness* terkecil.
  Nilai *fitness* terkecil dimiliki oleh partikel ke-2, yaitu 48.984.
- Menghitung *NCBPE*.

$$NCBPE = \frac{CBPE - CBPE_{\min}}{CBPE_{maks} - CBPE_{\min}} = \frac{48.987 - 48.984}{85.813 - 48.984} = 8.15e-5$$

Sedangkan nilai *CV* adalah nilai kecepatan partikel ke-4 pada arah y, yaitu 9.4e-7.

2. Melakukan fuzzifikasi masukan.
   Perhitungan fuzzifikasi masukan dilakukan pada semua variabel masukan *fuzzy*.
   - *NCBPE* = 8.15e-5

$$\mu_{Low}(NCBPE,0,0.005) = e^{-\frac{1}{2}\left(\frac{0.0000815-0}{0.005}\right)^2} = 0.9999$$

$$\mu_{Medium}(NCBPE,0.1,0.03) = e^{-\frac{1}{2}\left(\frac{0.0000815-0.1}{0.03}\right)^2} = 0.0039$$

$$\mu_{High}(NCBPE,1,0.25) = e^{-\frac{1}{2}\left(\frac{0.0000815-1}{0.25}\right)^2} = 0.0003359$$

   - *CV* = 9.4e-7

$$\mu_{Low}(CV,0,0,1e-030,1e-020) = \max\left(\min\left(\frac{(9.4e-7)-0}{0-0},1,\frac{(1e-020)-(9.4e-7)}{(1e-020)-(1e-030)}\right),0\right)$$

$$= \max(\min(\infty,1,-9.4e+013),0) = 0$$

$$\mu_{High}(CV,1e-010,1e-008,1e-006,1e-006) = 1$$

3. Menerapkan basis aturan.
   - Aturan 1

   Jika *NCBPE* adalah *Low* dan *CV* adalah *Low* maka V_{ck} adalah *High*

$$\mu_{High}(z) = \min(0.9999,0) = 0$$

   Kemudian dicari nilai *z* berdasarkan nilai himpunan *High* dari keluaran V_{ck}

$$\max\left(\min\left(\frac{z-0.1}{1.1-0.1},\frac{2.2-z}{2.2-1.1}\right),0\right)=0$$

$$\max\left(\min\left(\frac{z-0.1}{1},\frac{2.2-z}{1.1}\right),0\right)=0$$

Untuk $\frac{z-0.1}{1}=0$ diperoleh $z=0.1$ dan untuk $\frac{2.2-z}{1.1}=0$ diperoleh $z=2.2$.

- Aturan 2

Jika *NCBPE* adalah *Medium* maka V_{ck} adalah *Medium*

$\mu_{Medium}(z)=0.0039$

Kemudian dicari nilai *z* berdasarkan nilai himpunan *Medium* dari keluaran V_{ck}, yaitu

$$\max\left(\min\left(\frac{z+0.6}{0+0.6},\frac{0.2-z}{0.2-0}\right),0\right)=0.0039$$

$$\max\left(\min\left(\frac{z+0.6}{0.6},\frac{0.2-z}{0.2}\right),0\right)=0.0039$$

sehingga untuk $\frac{z+0.6}{0.6}=0.0039$ diperoleh $z=-0.5978$ dan untuk $\frac{0.2-z}{0.2}=0.0039$ diperoleh $z=0.19922$

- Aturan 3

Jika *NCBPE* adalah *High* dan *CV* adalah *High* maka V_{ck} adalah *Low*

$\mu_{Low}(z)=\min(0.0003359,1)=0.0003359$

Kemudian dicari nilai *z* berdasarkan nilai himpunan *Low* dari keluaran V_{ck}, yaitu

$$\max\left(\min\left(\frac{z+1}{-0.8+1},\frac{-0.5-z}{-0.5+0.8}\right),0\right)=0.0003359$$

$$\max\left(\min\left(\frac{z+1}{0.2},\frac{-0.5-z}{0.3}\right),0\right)=0.0003359$$

sehingga untuk $\frac{z+1}{0.2} = 0.0003359$ diperoleh $z = -0.9999328$

dan untuk $\frac{-0.5-z}{0.3} = 0.0003359$ diperoleh $z = -0.5001$.

- Aturan 4

Jika *NCBPE* adalah *Low* dan *CV* adalah *Low* maka ρ adalah *Large*

$\mu_{Large}(z) = \min(0.9999, 0) = 0$

Kemudian dicari nilai *z* berdasarkan nilai himpunan *Large* dari keluaran ρ, yaitu

$$\max\left(\min\left(\frac{z-47.15}{120-47.15}, \frac{120-z}{120-120}\right), 0\right) = 0$$

$$\max\left(\min\left(\frac{z-47.15}{72.85}, \infty\right), 0\right) = 0$$

sehingga dari $\frac{z-47.15}{72.85} = 0$ diperoleh $z = 47.15$

- Aturan 5

Jika *NCBPE* adalah *Medium* maka ρ adalah *Medium*

$\mu_{Medium}(z) = 0.0039$

Kemudian dicari nilai z berdasarkan nilai himpunan *Medium* dari keluaran ρ, yaitu

$$\max\left(\min\left(\frac{z-2.214}{10.71-2.214}, \frac{59.29-z}{59.29-10.71}\right), 0\right) = 0.0039$$

$$\max\left(\min\left(\frac{z-2.214}{8.496}, \frac{59.29-z}{48.58}\right), 0\right) = 0.0039$$

sehingga untuk $\frac{z-2.214}{8.496} = 0.0039$ diperoleh $z = 2.2471$ dan untuk $\frac{59.29-z}{48.58} = 0.0039$ diperoleh $z = 59.1005$.

- Aturan 6

Jika *NCBPE* adalah *High* dan *CV* adalah *High* maka ρ adalah *Small*

$\mu_{Small} = \min(0.0003359,1) = 0.0003359$

Kemudian dicari nilai z berdasarkan nilai himpunan *Small* dari keluaran ρ, yaitu

$$\max\left(\min\left(\frac{z-1}{1-1}, \frac{4-z}{4-1} \right), 0 \right) = 0.0003359$$

$$\max\left(\min\left(\infty, \frac{4-z}{3} \right), 0 \right) = 0.0003359$$

sehingga dari $\frac{4-z}{3} = 0.0003359$ diperoleh $z = 3.999$.

4. Melakukan defuzzifikasi keluaran.
   - Menghitung nilai keluaran V_{ck}.

     Dari hasil penerapan basis aturan *fuzzy* diperoleh beberapa nilai *z*, yaitu 0.1 dan 2.2 dari aturan 1 dengan derajat keanggotaan 0, -0.5978 dan 0.19922 dari aturan 2 dengan derajat keanggotaan 0.0039, dan -0.9999328 dan -0.5001 dari aturan 3 dengan derajat keanggotaan 0.0003359. Karena operator yang digunakan adalah *min-max* maka perlu dicari titik *z* pada himpunan *Medium* dengan derajat keanggotaan 0.0003359, yaitu

     $$\frac{z+0.6}{0.6} = 0.0003359$$

     dan diperoleh $z = -0.5998$.

   Metode defuzzifikasi yang digunakan adalah metode titik berat (CoG), sehingga perhitungan nilai V_{ck} sebagai berikut:

   $$V_{ck} = \frac{\int \mu(z)\, z\, dz}{\int \mu(z)\, dz}$$

   Terlebih dulu dihitung nilai $\int \mu(z)\, z\, dz$ dan $\int \mu(z)\, dz$, yaitu sebagai berikut:

$$\int \mu(z) z\, dz = \int_{-1}^{-0.9999328} \frac{z+1}{0.2} z\, dz + \int_{-0.9999328}^{-0.5998} 0.0003359\, z\, dz +$$
$$\int_{-0.5998}^{-0.5978} \frac{z+0.6}{0.6} z\, dz + \int_{-0.5978}^{0.19922} 0.0039\, z\, dz + \int_{0.19922}^{0.2} \frac{0.2-z}{0.2} z\, dz$$
$$= \left[\frac{z^3}{0.6} + \frac{z^2}{0.4}\right]_{-1}^{-0.9999328} + \left[\frac{0.0003359}{2} z^2\right]_{-0.9999328}^{-0.5998} +$$
$$\left[\frac{z^3}{1.8} + \frac{z^2}{2}\right]_{-0.5998}^{-0.5978} + \left[\frac{0.0039}{2} z^2\right]_{-0.5978}^{0.19922} + \left[\frac{z^2}{2} - \frac{z^3}{0.6}\right]_{0.19922}^{0.2}$$
$$= -1.13e\text{-}008 - 1.07e\text{-}004 - 2.39e\text{-}006 - 6.2e\text{-}004 + 3.0e\text{-}007$$
$$= -7.2812e\text{-}004$$

$$\int \mu(z) dz = \int_{-1}^{-0.9999328} \frac{z+1}{0.2} dz + \int_{-0.9999328}^{-0.5998} 0.0003359\, dz + \int_{-0.5998}^{-0.5978} \frac{z+0.6}{0.6} dz +$$
$$\int_{-0.5978}^{0.19922} 0.0039\, dz + \int_{0.19922}^{0.2} \frac{0.2-z}{0.2} dz$$
$$= \left[\frac{z^2}{0.4} + \frac{z}{0.2}\right]_{-1}^{-0.9999328} + \left[0.0003359\, z\right]_{-0.9999328}^{-0.5998} +$$
$$\left[\frac{z^2}{1.2} + z\right]_{-0.5998}^{-0.5978} + \left[0.0039\, z\right]_{-0.5978}^{0.19922} + \left[z - \frac{z^2}{0.4}\right]_{0.19922}^{0.2}$$
$$= 1.1290e\text{-}008 + 1.3440e\text{-}004 + 4e\text{-}006 + 0.0031 + 1.5210e\text{-}006$$
$$= 0.00325$$

Dari hasil perhitungan tersebut diperoleh

$$V_{ck} = \frac{\int \mu(z) z\, dz}{\int \mu(z) dz} = \frac{-7.2812e-004}{0.00325} = -0.2240 .$$

- Menghitung nilai keluaran ρ.

  Dari hasil penerapan basis aturan *fuzzy* diperoleh beberapa nilai z, yaitu 47.15 dari aturan 4 dengan derajat keanggotaan 0, 2.2471 dan 59.1005 dari aturan 5 dengan derajat keanggotaan 0.0039, dan 3.999 dari aturan 6 dengan derajat keanggotaan 0.0003359. Karena operator yang digunakan adalah *min-max* maka perlu dicari titik z pada himpunan *Medium* dengan derajat keanggotaan 0.0003359, yaitu

$$\frac{z-2.214}{8.496} = 0.0003359$$

  dan diperoleh $z = 2.2169$.

  Perhitungan untuk mencari nilai ρ adalah sebagai berikut:

$$\rho = \frac{\int \mu(z)\, z\, dz}{\int \mu(z)\, dz}$$

  Dihitung dulu nilai $\int \mu(z) z\, dz$ dan $\int \mu(z) dz$ yaitu sebagai berikut:

$$\int \mu(z) z\, dz = \int_1^{2.2169} (0.0003359) z\, dz + \int_{2.2169}^{2.2471} \frac{z-2.214}{8.496} z\, dz + \int_{2.2471}^{59.1005} (0.0039) z\, dz + \int_{59.1005}^{59.29} \frac{59.29-z}{48.58} z\, dz$$

$$= \left[0.00016795 z^2\right]_1^{2.2169} + \left[\frac{z^3}{25.488} - \frac{z^2}{7.6748}\right]_{2.2169}^{2.2471} + \left[0.00195 z^2\right]_{2.2471}^{59.1005} + \left[\frac{z^2}{1.6387} - \frac{z^3}{145.74}\right]_{59.1005}^{59.29}$$

$$= 0.000657 + 0.00014309 + 6.8067 + 0.0221$$

$$= 6.8296$$

$$\int \mu(z)dz = \int_{1}^{2.2169} (0.0003359)\, dz + \int_{2.2169}^{2.2471} \frac{z-2.214}{8.496}\, dz +$$
$$\int_{2.2471}^{59.1005} (0.0039)\, dz + \int_{59.1005}^{59.29} \frac{59.29-z}{48.58}\, dz$$
$$= \left[0.0003359\, z\right]_{1}^{2.2169} + \left[\frac{z^2}{16.992} - \frac{z}{3.8374}\right]_{2.2169}^{2.2471} +$$
$$\left[0.0039\, z\right]_{2.2471}^{59.1005} + \left[\frac{z}{0.8194} - \frac{z^2}{97.16}\right]_{59.1005}^{59.29}$$
$$= 0.00040876 + 0.000063986 + 0.2217 + 0.000359$$
$$= 0.2226$$

Dari hasil perhitungan tersebut diperoleh

$$\rho = \frac{\int \mu(z)\, z\, dz}{\int \mu(z)dz} = \frac{6.8296}{0.2226} = 30.6865 .$$

Setelah diperoleh nilai V_{ck} dilakukan perhitungan nilai v_c berdasarkan persamaan (3.26), yaitu

$$v_c = e^{-[10(1+V_{ck})]}$$
$$= e^{-[10(1-0.224)]}$$
$$= 4.2646\ 10^{-4}.$$

Langkah selanjutnya adalah menyelidiki kecepatan partikel, apakah lebih kecil dari batasan nilai v_c tersebut.

Berdasarkan kecepatan partikel, diketahui hanya partikel ke-4 pada arah y yang nilainya lebih kecil dari batasan nilai v_c, kemudian dilakukan perhitungan kecepatan $\hat{v}$ berdasarkan persamaan (3.2), yaitu

$$\hat{v} = u[-1,1] * v_{maks} / \rho$$
$$= -0.6522 * 1 / 30.6865$$
$$= -0.0213$$

sehingga kecepatan partikel ke-4 terhadap arah y menjadi -0.0213.

Langkah berikutnya adalah melakukan perbaikan kecepatan dan posisi sesuai langkah 8 yang ada pada lampiran 1.

Lampiran 3 Titik yang diperoleh dari hasil implementasi pada subbab 3.3.2

- Hasil uji algoritma menggunakan fungsi Quadric

Parameter		s=30, w=0.7, c_1=c_2=1.49			s=30, w=0.7, c_1=c_2=2		
Variabel	Nilai Min	Nilai yang diperoleh Algoritma			Nilai yang diperoleh Algoritma		
		PSO	TPSO	FATPSO	PSO	TPSO	FATPSO
x_1	0	22.050	-0.587	0.260	-26.805	-12.317	-0.636
x_2	0	-40.349	-0.524	-0.473	29.393	21.667	-0.018
x_3	0	18.052	1.896	0.422	-7.733	-10.556	2.543
x_4	0	-6.730	-0.772	-0.494	0.355	2.332	0.063
x_5	0	14.332	0.484	0.636	-12.35	-7.601	-6.669
x_6	0	-8.337	-1.560	-0.480	38.15	14.848	9.231
x_7	0	16.058	2.617	0.186	-12.304	-12.62	-8.914
x_8	0	-29.636	-2.633	-0.246	4.293	6.491	7.755
x_9	0	13.558	2.562	0.166	-17.09	-3.298	-1.910
x_{10}	0	7.866	-2.076	0.015	-8.619	-1.144	-2.025
x_{11}	0	-2.280	0.458	0.0015	13.862	7.642	4.466
x_{12}	0	-11.443	0.8101	0.038	-8.580	-10.65	-8.527
x_{13}	0	-32.190	-1.399	-0.179	23.789	13.233	9.242
x_{14}	0	71.497	5.302	0.104	-29.648	-15.555	-9.558
x_{15}	0	-21.212	-8.356	-0.106	14.703	10.746	8.574
x_{16}	0	-15.074	3.796	0.302	-12.626	0.222	-4.533
x_{17}	0	-0.3038	-0.262	-0.246	3.499	-7.753	0.931
x_{18}	0	6.297	0.303	0.112	19.048	5.390	4.775
x_{19}	0	14.746	0.080	-0.054	-0.936	-1.064	-6.753
x_{20}	0	-35.736	0.104	0.116	-10.414	0.317	2.332

Parameter		s=15, w=0.7, c_1=c_2=1.49			s=30, w=0.3, c_1=c_2=1.49		
Variabel	Nilai Min	Nilai yang diperoleh Algoritma			Nilai yang diperoleh Algoritma		
		PSO	TPSO	FATPSO	PSO	TPSO	FATPSO
x_1	0	-14.754	0.957	1.628	11.448	4.675	-1.127
x_2	0	2.297	-0.374	-6.825	-9.789	-5.433	-1.910
x_3	0	26.471	-4.251	5.011	-7.308	-3.277	-0.763
x_4	0	24.591	7.668	5.919	18.143	7.773	2.229
x_5	0	-10.794	-4.393	0.345	-8.617	-2.952	-1.225
x_6	0	-41.852	-7.067	-1.952	-12.958	-7.751	-0.475
x_7	0	34.299	12.775	-0.579	21.701	13.608	0.967
x_8	0	-48.598	-6.022	16.912	-22.273	-10.260	-1.142
x_9	0	25.809	1.615	1.617	14.413	7.970	1.055
x_{10}	0	-18.009	-4.305	-2.287	-17.534	-9.834	-1.159
x_{11}	0	40.779	8.902	1.045	27.745	9.219	0.800
x_{12}	0	-15.298	-6.573	-1.500	-3.385	-2.780	0.993
x_{13}	0	-8.159	0.749	-11.158	-24.014	-8.820	-2.592
x_{14}	0	4.180	-0.579	1.171	37.715	14.594	2.718
x_{15}	0	16.462	4.181	-5.476	-29.569	-10.704	-1.691
x_{16}	0	-25.426	-4.809	-24.103	11.242	4.413	0.977
x_{17}	0	2.512	6.507	-1.401	-7.770	1.794	-2.910
x_{18}	0	-12.374	-8.030	-0.014	-25.219	-4.257	-0.760
x_{19}	0	31.864	3.561	2.438	24.902	3.634	-0.144
x_{20}	0	-14.000	-1.711	-0.440	1.125	-2.053	-0.437

Parameter		s=30, w=1, c_1=c_2=1.49			s=30, w=1, c_1=c_2=2		
Variabel	Nilai Min	Nilai yang diperoleh Algoritma			Nilai yang diperoleh Algoritma		
		PSO	TPSO	FATPSO	PSO	TPSO	FATPSO
x_1	0	-2.036	-2.036	-2.036	-15.279	-15.279	-15.279
x_2	0	-0.832	-0.832	-0.832	-31.593	-31.593	-31.593
x_3	0	12.365	12.365	12.365	43.632	43.632	43.632
x_4	0	-30.493	-30.493	-30.493	-31.916	-31.916	-31.916
x_5	0	35.315	35.315	35.315	10.282	10.282	10.282
x_6	0	-64.560	-64.560	-64.560	32.523	32.523	32.523
x_7	0	-20.106	-20.106	-20.106	50.903	50.903	50.903
x_8	0	4.114	4.114	4.114	56.766	56.766	56.766
x_9	0	-12.650	-12.650	-12.650	84.376	84.376	84.376
x_{10}	0	-1.479	-1.479	-1.479	-8.958	-8.958	-8.958
x_{11}	0	63.943	63.943	63.943	52.937	52.937	52.937
x_{12}	0	-8.001	-8.001	-8.001	-29.991	-29.991	-29.991
x_{13}	0	-16.751	-16.751	-16.751	-15.552	-15.552	-15.552
x_{14}	0	-16.649	-16.649	-16.649	24.628	24.628	24.628
x_{15}	0	29.646	29.646	29.646	7.830	7.830	7.830
x_{16}	0	40.734	40.734	40.734	14.752	14.752	14.752
x_{17}	0	-14.070	-14.070	-14.070	-23.306	-23.306	-23.306
x_{18}	0	19.057	19.057	19.057	32.386	32.386	32.386
x_{19}	0	-13.736	-13.736	-13.736	-58.495	-58.495	-58.495
x_{20}	0	-50.474	-50.474	-50.474	50.108	50.108	50.108

- Hasil uji algoritma menggunakan fungsi Rastrigin

Parameter		s=30, w=0.7, c_1=c_2=1.49			s=30, w=0.7, c_1=c_2=2		
Variabel	Nilai Min	Nilai yang diperoleh Algoritma			Nilai yang diperoleh Algoritma		
		PSO	TPSO	FATPSO	PSO	TPSO	FATPSO
x_1	0	-0.012	0.054	0.035	1.015	0.993	0.997
x_2	0	-2.991	-2.990	-1.172	-1.006	-0.994	-0.945
x_3	0	-0.993	-0.994	0.021	-0.988	-0.994	-0.382
x_4	0	-0.992	-0.995	0.013	-0.006	0.001	0.713
x_5	0	0.994	0.994	1.00	0.002	-0.0002	-0.016
x_6	0	0.985	0.994	-0.0009	-0.952	-1.0105	-0.908
x_7	0	-0.995	-0.994	-0.978	0.994	0.993	0.993
x_8	0	0.980	0.993	0.977	-0.986	-0.994	-0.008
x_9	0	1.989	1.989	1.001	-0.982	-0.995	-0.129
x_{10}	0	1.133	0.991	0.028	-0.975	-0.994	-1.369

Parameter		s=15, w=0.7, c_1=c_2=1.49			s=30, w=0.3, c_1=c_2=1.49		
Variabel	Nilai Min	Nilai yang diperoleh Algoritma			Nilai yang diperoleh Algoritma		
		PSO	TPSO	FATPSO	PSO	TPSO	FATPSO
x_1	0	2.973	2.984	2.002	0.002	-0.0001	0.989
x_2	0	0.059	0.0008	-0.092	0.917	-0.994	-0.949
x_3	0	1.038	0.996	-0.040	0.856	0.994	0.996
x_4	0	-1.992	-1.989	-0.965	-0.901	-0.994	0.093
x_5	0	-1.895	-1.988	-0.980	1.928	1.989	1.991
x_6	0	3.034	2.984	1.003	0.006	-0.002	0.017
x_7	0	1.089	0.995	2.707	4.004	3.976	2.981
x_8	0	-1.015	-0.994	0.084	-0.836	-0.994	-0.993
x_9	0	-0.082	-0.060	0.019	0.943	0.981	1.001
x_{10}	0	-3.169	-2.979	-1.0002	2.03	1.991	1.986

Parameter		s=30, w=1, c_1=c_2=1.49			s=30, w=1, c_1=c_2=2		
Variabel	Nilai Min	Nilai yang diperoleh Algoritma			Nilai yang diperoleh Algoritma		
		PSO	TPSO	FATPSO	PSO	TPSO	FATPSO
x_1	0	1.969	1.969	1.969	1.532	1.532	1.532
x_2	0	-1.233	-1.233	-1.233	0.081	0.081	0.081
x_3	0	0.856	0.856	0.856	-1.063	-1.063	-1.063
x_4	0	-1.796	-1.796	-1.796	-1.795	-1.795	-1.795
x_5	0	0.321	0.321	0.321	1.050	1.050	1.050
x_6	0	0.049	0.049	0.049	-0.989	-0.989	-0.989
x_7	0	5.018	5.018	5.018	2.468	2.468	2.468
x_8	0	2.688	2.688	2.688	-1.209	-1.209	-1.209
x_9	0	1.544	1.544	1.544	3.665	3.665	3.665
x_{10}	0	1.516	1.516	1.516	-4.325	-4.325	-4.325