IJTIS

International Journal of Technological Innovation for Society (IJTIS)
https://journals.telkomuniversity.ac.id/ijtis

Vol. 1, No.1, 1-14, 2025
Publisher: Telkom University Journals

TUTORIAL ARTICLE

Apa itu Particle Swarm Optimization (PSO)?

Hilal Huda[1*] **and Heny Pratiwi**[2]

[1]*Telkom University, Bandung, Indonesia*
[2]*STMIK Widya Cipta Dharma, Samarinda, Kalimantan Timur, Indonesia*
[*]*Corresponding author*

Abstract
Apa itu PSO? Particle Swarm Optimization adalah algoritma optimasi metaheuristik berbasis populasi yang bekerja dengan menyebarkan sekumpulan calon solusi, yang disebut partikel, ke dalam ruang pencarian, lalu menggerakkan setiap partikel pada setiap iterasi menurut gabungan tiga dorongan: momentum arah geraknya sendiri, tarikan menuju posisi terbaik yang pernah dicapainya, dan tarikan menuju posisi terbaik yang pernah dicapai seluruh kawanan. Algoritma ini diilhami perilaku sosial kawanan burung dan gerombolan ikan, hanya memerlukan evaluasi fungsi tujuan tanpa perlu turunannya, dan seluruh mekanismenya termuat dalam dua persamaan pembaruan yang sederhana. Kesederhanaan itu justru menimbulkan persoalan pedagogis, karena sebagian besar bahan ajar memperkenalkan PSO langsung melalui kode program sehingga mekanisme internalnya tetap menjadi kotak hitam bagi pemula. Artikel tutorial ini menjawab pertanyaan pembuka di atas dengan pendekatan yang sepenuhnya dapat ditelusuri secara manual. Tujuan tulisan ini ada tiga: menurunkan persamaan pembaruan kecepatan dan posisi beserta makna setiap sukunya, menyajikan dua simulasi perhitungan lengkap yang dapat direproduksi dengan pensil dan kertas, serta menghubungkan simulasi tersebut dengan implementasi Python yang ringkas dan siap pakai. Pendekatan yang digunakan adalah pembekuan bilangan acak: seluruh nilai r_1 dan r_2 ditetapkan pada tabel tertentu sehingga setiap angka dalam makalah bersifat deterministik dan dapat diverifikasi pembaca. Studi kasus pertama adalah minimisasi fungsi satu variabel dengan tiga partikel selama tiga iterasi. Studi kasus kedua adalah function fitting, yaitu pencocokan model garis lurus terhadap empat titik data, dengan partikel berdimensi dua yang merepresentasikan kemiringan dan titik potong. Hasil kedua simulasi menunjukkan perilaku konvergensi yang jelas: pada kasus pertama nilai fitness global terbaik menurun dari 2 menjadi 1,015625 terhadap optimum sejati 1, sedangkan pada kasus kedua jumlah kuadrat galat menurun dari 1 menjadi 0,300504 hanya dalam tiga iterasi, sangat dekat dengan solusi kuadrat terkecil sebesar 0,3 pada koefisien 1,7 dan 1,5. Simulasi juga memperlihatkan dua fenomena penting yang jarang ditampilkan secara eksplisit, yaitu lenyapnya suku kognitif dan sosial ketika sebuah partikel bertepatan dengan posisi terbaiknya, dan konvergensi prematur seluruh kawanan pada satu titik. Kesimpulannya, PSO dapat dipahami secara utuh melalui aritmetika dasar tanpa mengorbankan ketepatan, dan kerangka verifikasi ganda antara perhitungan tangan, solusi analitik, serta keluaran program terbukti efektif sebagai alat ajar sekaligus sebagai prosedur pengujian implementasi.

Keywords: Function fitting, Kecerdasan kawanan, Metaheuristik, Optimasi numerik, Particle swarm optimization

Corresponding author:
hilalh@teluapp.org

Received: 6 Agustus 2026
Accepted: 6 Agustus 2026
Available online: 6 Agustus 2026

1. Pendahuluan

Banyak persoalan rekayasa pada akhirnya bermuara pada satu pertanyaan yang sama: dari sekian banyak kemungkinan, mana yang paling baik? Menentukan rute pengiriman termurah, memilih subset fitur terbaik untuk sebuah pengklasifikasi, atau menyetel parameter pengendali agar respons sistem paling stabil, semuanya adalah persoalan optimasi. Ketika fungsi tujuan bersifat mulus dan turunannya mudah diperoleh, metode

Cite as: H. Huda and H. Pratiwi (2025). Apa itu Particle Swarm Optimization (PSO)?. International Journal of Technological Innovation for Society , 1(1) (2025) 1-14.

berbasis gradien bekerja sangat baik. Persoalan muncul ketika fungsi tujuan tidak diketahui bentuk analitiknya, tidak kontinu, memiliki banyak optimum lokal, atau hanya dapat dievaluasi melalui simulasi. Pada situasi semacam ini algoritma metaheuristik menjadi pilihan yang masuk akal karena hanya memerlukan kemampuan mengevaluasi fungsi tujuan, bukan menurunkannya [7].

Particle Swarm Optimization (PSO) diperkenalkan oleh Kennedy dan Eberhart pada 1995 sebagai penyederhanaan model simulasi perilaku kawanan burung [1, 2]. Gagasan intinya sangat sederhana. Sekumpulan calon solusi, yang disebut partikel, disebar di ruang pencarian. Setiap partikel mengingat posisi terbaik yang pernah dicapainya sendiri dan mengetahui posisi terbaik yang pernah dicapai oleh seluruh kawanan. Pada setiap iterasi, partikel bergerak dengan menggabungkan tiga dorongan: kecenderungan mempertahankan arah geraknya, tarikan menuju pengalaman terbaiknya sendiri, dan tarikan menuju pengalaman terbaik kawanan. Dari aturan yang begitu ringkas, muncul perilaku pencarian kolektif yang efektif.

Ketiadaan operator rumit seperti persilangan dan mutasi membuat PSO mudah diimplementasikan, dan itulah salah satu alasan popularitasnya yang bertahan selama tiga dekade [10]. Ironisnya, kemudahan itu justru sering menjadi masalah pedagogis. Sebagian besar materi pengantar memperkenalkan PSO melalui potongan kode yang langsung dapat dijalankan, sehingga pembaca dapat memperoleh hasil tanpa pernah benar-benar memahami mengapa sebuah partikel bergerak ke arah tertentu. Ketika program menghasilkan keluaran yang mencurigakan, pembaca tidak memiliki acuan untuk menilai apakah yang salah adalah kodenya, parameternya, atau pemahamannya.

Artikel ini disusun untuk menutup celah tersebut. Kontribusi tulisan ini ada empat. *Pertama*, penurunan persamaan PSO disajikan suku demi suku beserta interpretasi geometrisnya. *Kedua*, disajikan simulasi minimisasi fungsi satu variabel yang seluruh angkanya berupa pecahan eksak sehingga dapat diverifikasi dengan pensil dan kertas. *Ketiga*, disajikan simulasi *function fitting* yang memperlihatkan bagaimana persoalan pencocokan model diubah menjadi persoalan optimasi, lengkap dengan pembanding berupa solusi kuadrat terkecil sebagai kebenaran acuan. *Keempat*, seluruh simulasi tersebut dipasangkan dengan implementasi Python yang menghasilkan angka identik, sehingga pembaca memiliki cara mandiri untuk memvalidasi kodenya sendiri.

Kunci agar seluruh perhitungan dapat direproduksi adalah pembekuan bilangan acak. PSO bersifat stokastik karena mengandung dua bilangan acak pada setiap pembaruan kecepatan. Dalam artikel ini bilangan-bilangan tersebut ditetapkan pada nilai tertentu yang diumumkan secara terbuka. Dengan demikian sifat algoritmanya tetap utuh, hanya keacakannya yang digantikan tabel tetap, persis seperti menetapkan *seed* pada program sungguhan.

Sisa artikel disusun sebagai berikut. Bagian 2 memaparkan konsep dan formulasi PSO. Bagian 3 menyajikan simulasi tangan untuk optimasi fungsi satu variabel. Bagian 4 menyajikan simulasi tangan untuk *function fitting*. Bagian 5 membahas implementasi Python. Bagian 6 mendiskusikan pemilihan parameter, keterbatasan, serta varian PSO, dan Bagian 7 menutup dengan kesimpulan.

2. Konsep dan Formulasi PSO

2.1. Analogi Kawanan

Bayangkan sekawanan burung mencari sumber makanan di sebuah lembah yang tertutup kabut. Tidak ada satu pun burung yang mengetahui letak pasti makanan tersebut, tetapi setiap burung dapat merasakan seberapa kuat aroma makanan di posisinya saat ini. Setiap burung mengingat titik dengan aroma terkuat yang pernah ia lewati, dan melalui suara, seluruh kawanan mengetahui titik terbaik yang pernah ditemukan salah satu anggotanya. Setiap burung kemudian menyesuaikan arah terbangnya sebagai kompromi antara meneruskan arah saat ini, kembali ke titik terbaiknya sendiri, dan mendekati titik terbaik kawanan.

Dalam PSO, lembah berkabut adalah ruang pencarian, kekuatan aroma adalah nilai fungsi tujuan, satu burung adalah satu partikel, titik terbaik pribadi disebut *pbest*, dan titik terbaik kawanan disebut *gbest*.

2.2. Notasi

Misalkan persoalan yang ditinjau adalah minimisasi $f : \mathbb{R}^D \to \mathbb{R}$ dengan D menyatakan banyaknya variabel keputusan. Kawanan terdiri atas N partikel. Pada iterasi ke-t, partikel ke-i dicirikan oleh:

- $\mathbf{x}_i^t = (x_{i1}^t, \ldots, x_{iD}^t)$, posisi partikel, yaitu satu calon solusi;
- $\mathbf{v}_i^t = (v_{i1}^t, \ldots, v_{iD}^t)$, kecepatan partikel, yaitu vektor perpindahan;
- $\mathbf{p}_i^t$, posisi terbaik yang pernah dicapai partikel i (*pbest*);
- $\mathbf{g}^t$, posisi terbaik yang pernah dicapai seluruh kawanan (*gbest*).

Nilai $f(\mathbf{x})$ disebut nilai *fitness*. Karena persoalan yang ditinjau adalah minimisasi, semakin kecil nilai *fitness* semakin baik posisi tersebut.

2.3. Persamaan Pembaruan

Inti PSO hanya terdiri atas dua persamaan. Kecepatan diperbarui melalui

$$v_{ij}^{t+1} = \underbrace{w v_{ij}^t}_{\text{inersia}} + \underbrace{c_1 r_1 \left(p_{ij}^t - x_{ij}^t\right)}_{\text{kognitif}} + \underbrace{c_2 r_2 \left(g_j^t - x_{ij}^t\right)}_{\text{sosial}}, \tag{1}$$

kemudian posisi diperbarui melalui

$$x_{ij}^{t+1} = x_{ij}^t + v_{ij}^{t+1}, \tag{2}$$

untuk $i = 1, \ldots, N$ dan $j = 1, \ldots, D$. Simbol w adalah bobot inersia, c_1 dan c_2 berturut-turut adalah koefisien kognitif dan sosial, sedangkan r_1 dan r_2 adalah bilangan acak yang dibangkitkan secara seragam pada selang $[0, 1]$ dan dibangkitkan ulang pada setiap iterasi.

Ketiga suku pada Persamaan (1) memiliki tafsir geometris yang jelas. Suku inersia mempertahankan momentum gerak sebelumnya dan berperan menjaga eksplorasi; tanpa suku ini partikel akan langsung tersedot ke arah *pbest* dan *gbest*. Suku kognitif menarik partikel kembali ke pengalaman terbaiknya sendiri dan mencerminkan sifat individualistik. Suku sosial menarik partikel ke arah pengalaman terbaik kawanan dan mencerminkan pertukaran informasi. Perlu diperhatikan bahwa apabila $\mathbf{x}_i = \mathbf{p}_i = \mathbf{g}$, kedua suku terakhir bernilai nol sehingga partikel hanya meluncur mengikuti inersia. Fenomena ini akan terlihat langsung pada simulasi di Bagian 3 dan 4.
Setelah posisi diperbarui, kedua memori disegarkan menurut

$$\mathbf{p}_i^{t+1} = \begin{cases} \mathbf{x}_i^{t+1}, & \text{jika } f(\mathbf{x}_i^{t+1}) < f(\mathbf{p}_i^t), \\ \mathbf{p}_i^t, & \text{selainnya}, \end{cases} \qquad \mathbf{g}^{t+1} = \arg\min_{1 \le i \le N} f(\mathbf{p}_i^{t+1}). \tag{3}$$

2.4. Peran Parameter

Bobot inersia w mengendalikan keseimbangan antara eksplorasi dan eksploitasi. Nilai w yang besar memperluas jelajah kawanan tetapi memperlambat konvergensi, sedangkan nilai yang kecil mempercepat konvergensi dengan risiko terjebak pada optimum lokal. Shi dan Eberhart memperkenalkan parameter ini beserta strategi penurunan linear dari sekitar 0,9 menuju 0,4 sepanjang iterasi [3]. Analisis kestabilan lintasan partikel oleh Clerc dan Kennedy [4] serta Trelea [5] menghasilkan rekomendasi konfigurasi yang kini menjadi rujukan baku, yaitu $w = 0{,}7298$ dan $c_1 = c_2 = 1{,}49618$. Konfigurasi tersebut dipakai pada implementasi Python di Bagian 5, sedangkan pada simulasi tangan digunakan nilai yang lebih bersahaja agar aritmetikanya dapat dikerjakan manual.
Koefisien c_1 dan c_2 mengatur porsi kepercayaan partikel terhadap dirinya sendiri dibandingkan terhadap kawanan. Nilai c_1 yang jauh lebih besar membuat setiap partikel bekerja hampir sendiri-sendiri, sebaliknya nilai c_2 yang dominan membuat kawanan cepat mengerumun dan rentan konvergensi prematur. Ukuran kawanan N umumnya dipilih antara 20 sampai 50 partikel; kajian mengenai konfigurasi baku PSO menunjukkan bahwa penambahan partikel di luar rentang tersebut jarang memberikan manfaat sepadan dengan biaya komputasinya [8].

2.5. Diagram Alir dan Pseudokode

Figure 1 menyajikan diagram alir PSO dan Algoritma 1 merangkum prosedur yang sama dalam bentuk pseudokode. Struktur algoritmanya terdiri atas satu blok inisialisasi yang dijalankan sekali dan satu gelang utama yang diulang sampai kriteria berhenti terpenuhi. Kriteria berhenti yang lazim dipakai adalah tercapainya jumlah iterasi maksimum, tercapainya jumlah evaluasi fungsi maksimum, atau tidak adanya perbaikan nilai *gbest* selama sejumlah iterasi berturut-turut. Perlu dicatat bahwa PSO tidak menjamin diperolehnya optimum global; yang dijamin hanyalah bahwa nilai *fitness* pada *gbest* tidak pernah memburuk sepanjang iterasi, karena pembaruan pada Persamaan (3) hanya menerima perbaikan.

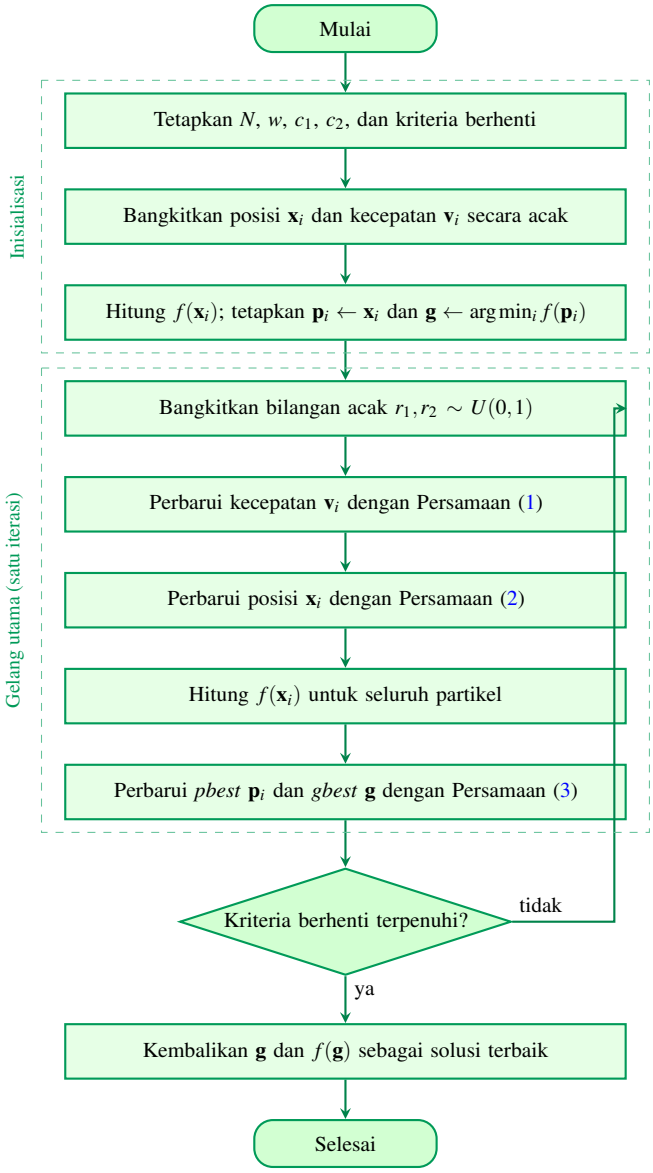

Figure 1: Diagram alir Particle Swarm Optimization dengan topologi *gbest*. Blok inisialisasi dijalankan satu kali, sedangkan gelang utama diulang sampai kriteria berhenti terpenuhi. Nomor persamaan pada kotak merujuk pada persamaan di Bagian 2.3.

Table 1: Pseudokode PSO baku dengan topologi *gbest*

1	Tetapkan N, w, c_1, c_2, dan kriteria berhenti
2	Bangkitkan posisi $\mathbf{x}_i$ dan kecepatan $\mathbf{v}_i$ secara acak untuk $i = 1, \ldots, N$
3	Tetapkan $\mathbf{p}_i \leftarrow \mathbf{x}_i$ dan hitung $f(\mathbf{p}_i)$
4	Tetapkan $\mathbf{g} \leftarrow \arg\min_i f(\mathbf{p}_i)$
5	**selama** kriteria berhenti belum tercapai **lakukan**
6	**untuk setiap** partikel i **lakukan**
7	Bangkitkan $r_1, r_2 \sim U(0,1)$
8	Perbarui $\mathbf{v}_i$ dengan Persamaan (1)
9	Perbarui $\mathbf{x}_i$ dengan Persamaan (2)
10	Hitung $f(\mathbf{x}_i)$
11	**jika** $f(\mathbf{x}_i) < f(\mathbf{p}_i)$ **maka** $\mathbf{p}_i \leftarrow \mathbf{x}_i$
12	**akhir untuk**
13	Perbarui $\mathbf{g} \leftarrow \arg\min_i f(\mathbf{p}_i)$
14	**akhir selama**
15	**kembalikan** $\mathbf{g}$ dan $f(\mathbf{g})$

3. Simulasi Perhitungan Tangan: Optimasi Fungsi Satu Variabel

3.1. Perumusan Persoalan

Ditinjau persoalan minimisasi fungsi kuadratik satu variabel

$$f(x) = x^2 - 4x + 5. \tag{4}$$

Fungsi ini sengaja dipilih karena solusinya diketahui secara analitik. Dari $f'(x) = 2x - 4 = 0$ diperoleh $x^* = 2$ dengan $f(x^*) = 1$. Adanya kebenaran acuan ini memungkinkan pembaca menilai seberapa dekat PSO bergerak menuju jawaban yang benar pada setiap iterasi.
Konfigurasi yang digunakan adalah $N = 3$ partikel, $D = 1$ dimensi, $w = 0{,}5$, serta $c_1 = c_2 = 1$. Agar seluruh aritmetika dapat dikerjakan manual, bilangan acak dibekukan pada $r_1 = r_2 = 0{,}5$ untuk semua partikel dan semua iterasi. Dengan substitusi tersebut, Persamaan (1) menyederhana menjadi

$$v_i^{t+1} = 0{,}5\, v_i^t + 0{,}5\left(p_i^t - x_i^t\right) + 0{,}5\left(g^t - x_i^t\right). \tag{5}$$

3.2. Inisialisasi

Posisi awal ditetapkan pada $x_1^0 = 0$, $x_2^0 = 3$, dan $x_3^0 = 5$, dengan seluruh kecepatan awal bernilai nol. Nilai *fitness* dihitung langsung dari Persamaan (4), misalnya $f(0) = 0 - 0 + 5 = 5$ dan $f(3) = 9 - 12 + 5 = 2$. Karena belum ada riwayat, *pbest* setiap partikel sama dengan posisi awalnya. Hasil selengkapnya disajikan pada Table 2.

Table 2: Inisialisasi kawanan untuk minimisasi $f(x) = x^2 - 4x + 5$

Partikel	x_i^0	v_i^0	$f(x_i^0)$	p_i^0	$f(p_i^0)$	Keterangan
1	0	0	5	0	5	
2	3	0	2	3	2	*gbest* awal
3	5	0	10	5	10	

Karena partikel 2 memiliki nilai *fitness* terkecil, maka $g^0 = 3$ dengan $f(g^0) = 2$.

3.3. Iterasi 1

Dengan $g^0 = 3$ dan $p_i^0 = x_i^0$ untuk semua i, suku kognitif pada iterasi pertama pasti bernilai nol karena setiap partikel masih berada tepat pada *pbest*-nya sendiri. Perhitungan untuk masing-masing partikel adalah sebagai berikut.
Partikel 1:

$$v_1^1 = 0{,}5(0) + 0{,}5(0-0) + 0{,}5(3-0) = 1{,}5,$$
$$x_1^1 = 0 + 1{,}5 = 1{,}5, \qquad f(1{,}5) = 2{,}25 - 6 + 5 = 1{,}25.$$

Partikel 2:

$$v_2^1 = 0{,}5(0) + 0{,}5(3-3) + 0{,}5(3-3) = 0,$$
$$x_2^1 = 3 + 0 = 3, \qquad f(3) = 2.$$

Partikel 3:

$$v_3^1 = 0{,}5(0) + 0{,}5(5-5) + 0{,}5(3-5) = -1,$$
$$x_3^1 = 5 - 1 = 4, \qquad f(4) = 16 - 16 + 5 = 5.$$

Partikel 2 tidak bergerak sama sekali. Hal tersebut memang benar secara matematis: partikel itu sekaligus merupakan *pbest* dirinya dan *gbest* kawanan, sehingga kedua suku penarik lenyap, sementara kecepatan sebelumnya bernilai nol. Inilah wujud konkret sifat yang telah diuraikan pada Bagian 2.3.
Selanjutnya memori disegarkan. Partikel 1 memperbaiki diri dari 5 menjadi 1,25 sehingga $p_1^1 = 1{,}5$. Partikel 3 memperbaiki diri dari 10 menjadi 5 sehingga $p_3^1 = 4$. Partikel 2 tidak berubah. Nilai *fitness* terkecil kini dipegang partikel 1, sehingga $g^1 = 1{,}5$ dengan $f(g^1) = 1{,}25$.

3.4. Iterasi 2

Dengan $g^1 = 1{,}5$, perhitungan dilanjutkan menggunakan kecepatan hasil iterasi sebelumnya.
Partikel 1:

$$v_1^2 = 0{,}5(1{,}5) + 0{,}5(1{,}5-1{,}5) + 0{,}5(1{,}5-1{,}5) = 0{,}75,$$
$$x_1^2 = 1{,}5 + 0{,}75 = 2{,}25, \qquad f(2{,}25) = 5{,}0625 - 9 + 5 = 1{,}0625.$$

Partikel 2:

$$v_2^2 = 0{,}5(0) + 0{,}5(3-3) + 0{,}5(1{,}5-3) = -0{,}75,$$
$$x_2^2 = 3 - 0{,}75 = 2{,}25, \qquad f(2{,}25) = 1{,}0625.$$

Partikel 3:

$$v_3^2 = 0{,}5(-1) + 0{,}5(4-4) + 0{,}5(1{,}5-4) = -0{,}5 - 1{,}25 = -1{,}75,$$
$$x_3^2 = 4 - 1{,}75 = 2{,}25, \qquad f(2{,}25) = 1{,}0625.$$

Terjadi peristiwa yang patut dicermati: ketiga partikel mendarat pada titik yang persis sama, yaitu $x = 2{,}25$. Seluruh *pbest* diperbarui menjadi 2,25 dan $g^2 = 2{,}25$ dengan $f(g^2) = 1{,}0625$. Kawanan telah kehilangan seluruh keragamannya. Peristiwa ini merupakan bentuk paling ekstrem dari konvergensi prematur dan muncul justru karena bilangan acak sengaja dibekukan pada nilai yang sama untuk semua partikel. Pada PSO sungguhan, r_1 dan r_2 berbeda-beda antarpartikel sehingga peluang terjadinya keruntuhan serentak semacam ini sangat kecil.

3.5. Iterasi 3

Karena $x_i^2 = p_i^2 = g^2 = 2{,}25$ untuk semua i, suku kognitif dan sosial seluruhnya lenyap. Yang tersisa hanyalah inersia, sehingga $v_i^3 = 0{,}5\, v_i^2$:

$$v_1^3 = 0{,}5(0{,}75) = 0{,}375, \quad x_1^3 = 2{,}625, \quad f = 1{,}390625,$$
$$v_2^3 = 0{,}5(-0{,}75) = -0{,}375, \quad x_2^3 = 1{,}875, \quad f = 1{,}015625,$$
$$v_3^3 = 0{,}5(-1{,}75) = -0{,}875, \quad x_3^3 = 1{,}375, \quad f = 1{,}390625.$$

Hanya partikel 2 yang berhasil memperbaiki diri, sehingga $p_2^3 = 1{,}875$ dan $g^3 = 1{,}875$ dengan $f(g^3) = 1{,}015625$. Di sinilah peran suku inersia terlihat paling nyata. Meskipun kawanan sempat runtuh pada satu titik, momentum yang tersimpan pada kecepatan masing-masing partikel tetap mendorong mereka menyebar kembali, dan kebetulan salah satunya melewati daerah yang lebih baik. Tanpa suku inersia, ketiga partikel akan terkunci selamanya di $x = 2{,}25$.

3.6. Rangkuman

Rekapitulasi ketiga iterasi disajikan pada Table 3. Nilai *fitness* pada *gbest* menurun secara monoton dari 2 menuju 1,015625, sementara optimum sejatinya adalah 1. Dengan hanya tiga partikel dan tiga iterasi, galat terhadap optimum telah menyusut dari 1,0 menjadi 0,015625, yaitu penyusutan lebih dari 98 persen.

Table 3: Rekapitulasi tiga iterasi PSO untuk $f(x) = x^2 - 4x + 5$

Iterasi	Partikel 1		Partikel 2		Partikel 3		$f(g^t)$
	v_1	x_1	v_2	x_2	v_3	x_3	
0	0	0	0	3	0	5	2
1	1,5	1,5	0	3	−1	4	1,25
2	0,75	2,25	−0,75	2,25	−1,75	2,25	1,0625
3	0,375	2,625	−0,375	1,875	−0,875	1,375	1,015625

Lintasan ketiga partikel di atas kurva fungsi tujuan, beserta grafik konvergensinya, ditampilkan pada Figure 2.

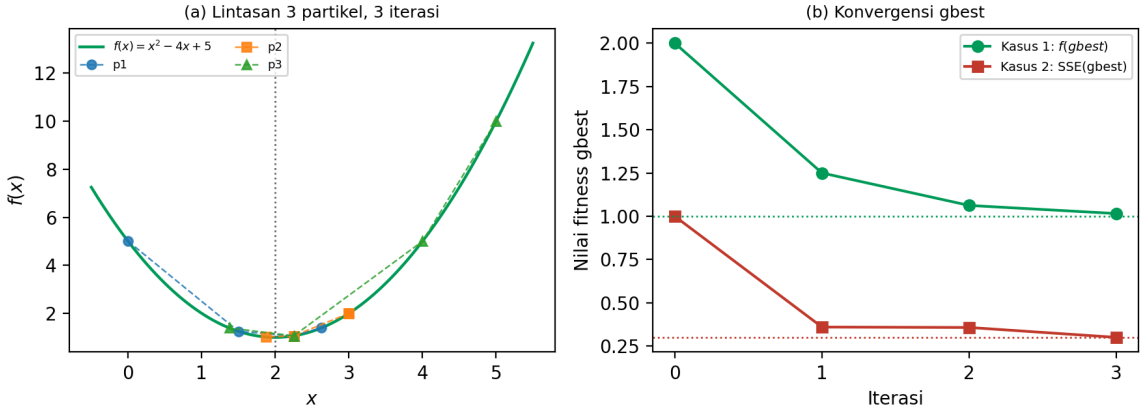

Figure 2: Perilaku PSO pada kedua studi kasus. Panel (a) memperlihatkan lintasan tiga partikel di atas kurva $f(x) = x^2 - 4x + 5$ selama tiga iterasi; garis titik-titik vertikal menandai optimum sejati $x^* = 2$. Panel (b) memperlihatkan penurunan nilai *fitness* pada *gbest* untuk kasus optimasi satu variabel dan kasus *function fitting*; garis titik-titik mendatar menandai nilai optimum masing-masing, yaitu 1 dan 0,3.

4. Simulasi Perhitungan Tangan: Function Fitting

4.1. Mengubah Pencocokan Model Menjadi Persoalan Optimasi

Bagian sebelumnya memperlihatkan cara kerja PSO pada fungsi yang sudah berbentuk siap-optimasi. Pada praktiknya, persoalan yang lebih sering dijumpai adalah pencocokan model terhadap data. Misalkan tersedia n pasangan pengamatan (x_k, y_k) dan hendak dicari model garis lurus

$$\hat{y}(x) = ax + b \tag{6}$$

yang paling sesuai dengan data. Pertanyaan "model mana yang paling cocok" diterjemahkan menjadi "pasangan (a,b) mana yang meminimumkan galat", dengan ukuran galat berupa jumlah kuadrat sisaan

$$\mathrm{SSE}(a,b) = \sum_{k=1}^{n} \left[y_k - (ax_k + b)\right]^2 . \tag{7}$$

Perubahan sudut pandang inilah yang menjadi inti *function fitting* dengan metode metaheuristik. Partikel tidak lagi mewakili sebuah titik pada sumbu x, melainkan mewakili sepasang koefisien model. Dengan demikian ruang pencariannya berdimensi dua, yaitu $\mathbf{x}_i = (a_i, b_i)$, dan fungsi tujuannya adalah Persamaan (7). Struktur algoritmanya sama sekali tidak berubah; yang berubah hanya tafsir posisi partikel.

4.2. Data dan Solusi Acuan

Digunakan empat titik data $(1,3)$, $(2,5)$, $(3,7)$, dan $(4,8)$. Tiga titik pertama tepat berada pada garis $y = 2x+1$, sedangkan titik keempat sengaja digeser agar tidak ada garis yang dapat melewati keempatnya. Dengan demikian nilai SSE minimum pasti positif, sehingga persoalannya menjadi pencocokan yang sesungguhnya, bukan interpolasi.
Solusi acuan diperoleh melalui persamaan normal kuadrat terkecil. Dengan $n = 4$, $\sum x_k = 10$, $\sum y_k = 23$, $\sum x_k y_k = 66$, dan $\sum x_k^2 = 30$, diperoleh

$$a^* = \frac{n\sum x_k y_k - \sum x_k \sum y_k}{n\sum x_k^2 - \left(\sum x_k\right)^2} = \frac{4(66) - (10)(23)}{4(30) - 100} = \frac{34}{20} = 1{,}7, \tag{8}$$

dan $b^* = \left(\sum y_k - a^* \sum x_k\right)/n = (23-17)/4 = 1{,}5$. Nilai galat pada solusi tersebut adalah $\mathrm{SSE}(1{,}7;\, 1{,}5) = 0{,}3$. Angka inilah yang akan digunakan sebagai tolok ukur keberhasilan PSO.

4.3. Konfigurasi dan Inisialisasi

Digunakan $N = 3$ partikel berdimensi $D = 2$, dengan $w = 0{,}5$ dan $c_1 = c_2 = 1$ seperti sebelumnya. Berbeda dengan Bagian 3, kali ini bilangan acak tidak lagi disamakan melainkan diambil dari tabel tetap pada Table 4. Pilihan ini membuat simulasi lebih menyerupai PSO sungguhan sekaligus mencegah keruntuhan kawanan seperti yang terjadi sebelumnya, namun aritmetikanya tetap terbatas pada satu angka di belakang koma sehingga masih nyaman dikerjakan manual. Sesuai konvensi yang lazim pada bahan ajar, satu pasang (r_1, r_2) digunakan untuk kedua dimensi sebuah partikel.

Table 4: Tabel bilangan acak yang dibekukan untuk simulasi *function fitting*

Iterasi	Partikel 1		Partikel 2		Partikel 3	
	r_1	r_2	r_1	r_2	r_1	r_2
1	0,8	0,6	0,2	0,6	0,8	0,4
2	0,4	0,4	0,4	0,6	0,4	0,4
3	0,4	0,8	0,4	0,4	0,2	0,8

Posisi awal ketiga partikel adalah $(a,b) = (1;3)$, $(3;0)$, dan $(2;1)$, dengan kecepatan awal nol. Perhitungan SSE untuk partikel 3 diperinci sebagai contoh. Untuk $a = 2$ dan $b = 1$, prediksi model pada keempat titik adalah 3, 5, 7, dan 9, sehingga sisaannya berturut-turut $3-3=0$, $5-5=0$, $7-7=0$, dan $8-9=-1$. Dengan demikian $\mathrm{SSE} = 0+0+0+1 = 1$. Hasil untuk seluruh partikel disajikan pada Table 5.

Table 5: Inisialisasi kawanan untuk *function fitting* $\hat{y} = ax + b$

Partikel	a_i^0	b_i^0	Model	SSE	Keterangan
1	1	3	$\hat{y} = x+3$	3	
2	3	0	$\hat{y} = 3x$	21	
3	2	1	$\hat{y} = 2x+1$	1	*gbest* awal

Partikel 3 memiliki galat terkecil sehingga $\mathbf{g}^0 = (2;\, 1)$ dengan $\mathrm{SSE} = 1$.

4.4. Iterasi 1

Perhitungan dilakukan per dimensi. Untuk partikel 1 dengan $r_1 = 0{,}8$ dan $r_2 = 0{,}6$:

$$v_a = 0{,}5(0) + 1(0{,}8)(1-1) + 1(0{,}6)(2-1) = 0{,}6, \qquad a = 1 + 0{,}6 = 1{,}6,$$
$$v_b = 0{,}5(0) + 1(0{,}8)(3-3) + 1(0{,}6)(1-3) = -1{,}2, \qquad b = 3 - 1{,}2 = 1{,}8.$$

Model yang dihasilkan adalah $\hat{y} = 1{,}6x + 1{,}8$ dengan prediksi 3,4; 5,0; 6,6; dan 8,2. Sisaannya adalah $-0{,}4$; 0; 0,4; dan $-0{,}2$, sehingga

$$\text{SSE} = 0{,}16 + 0 + 0{,}16 + 0{,}04 = 0{,}36.$$

Galat menyusut drastis dari 3 menjadi 0,36.
Untuk partikel 2 dengan $r_1 = 0{,}2$ dan $r_2 = 0{,}6$:

$$v_a = 0{,}5(0) + 0{,}2(3-3) + 0{,}6(2-3) = -0{,}6, \qquad a = 3 - 0{,}6 = 2{,}4,$$
$$v_b = 0{,}5(0) + 0{,}2(0-0) + 0{,}6(1-0) = 0{,}6, \qquad b = 0 + 0{,}6 = 0{,}6,$$

menghasilkan SSE $= 5{,}64$, yaitu perbaikan besar dari nilai awal 21.
Untuk partikel 3, karena posisinya bertepatan dengan *pbest* sekaligus *gbest*, kedua suku penarik lenyap dan kecepatannya tetap nol, sehingga partikel bertahan di $(2; 1)$ dengan SSE $= 1$. Nilainya tidak lebih baik dari sebelumnya sehingga *pbest*-nya tidak berubah.
Karena partikel 1 kini memegang galat terkecil, maka $\mathbf{g}^1 = (1{,}6; 1{,}8)$ dengan SSE $= 0{,}36$.

4.5. Iterasi 2

Sebagai contoh diperinci partikel 2, yang pada iterasi ini menghasilkan perbaikan terbesar. Dengan $r_1 = 0{,}4$, $r_2 = 0{,}6$, posisi $(2{,}4; 0{,}6)$, *pbest* $(2{,}4; 0{,}6)$, kecepatan sebelumnya $(-0{,}6; 0{,}6)$, dan $\mathbf{g}^1 = (1{,}6; 1{,}8)$:

$$v_a = 0{,}5(-0{,}6) + 0{,}4(0) + 0{,}6(1{,}6 - 2{,}4) = -0{,}3 - 0{,}48 = -0{,}78, \qquad a = 2{,}4 - 0{,}78 = 1{,}62,$$
$$v_b = 0{,}5(0{,}6) + 0{,}4(0) + 0{,}6(1{,}8 - 0{,}6) = 0{,}3 + 0{,}72 = 1{,}02, \qquad b = 0{,}6 + 1{,}02 = 1{,}62,$$

sehingga diperoleh SSE$(1{,}62; 1{,}62) = 0{,}3576$. Nilai ini lebih baik daripada *gbest* sebelumnya, sehingga $\mathbf{g}^2 = (1{,}62; 1{,}62)$.
Partikel 1 bergerak ke $(1{,}9; 1{,}2)$ dengan SSE $= 0{,}66$, yang lebih buruk daripada *pbest*-nya sehingga memorinya tidak berubah. Partikel 3 bergerak ke $(1{,}84; 1{,}32)$ dengan SSE $= 0{,}5136$, membaik dari 1 sehingga *pbest*-nya diperbarui. Perlu digarisbawahi bahwa partikel yang melangkah ke posisi lebih buruk tetap melanjutkan gerak dari posisi barunya; yang dipertahankan hanyalah memori, bukan posisinya. Mekanisme inilah yang memungkinkan PSO keluar dari optimum lokal.

4.6. Iterasi 3

Pada iterasi terakhir, partikel 1 melakukan lompatan terbaik. Dengan $r_1 = 0{,}4$, $r_2 = 0{,}8$, posisi $(1{,}9; 1{,}2)$, *pbest* $(1{,}6; 1{,}8)$, kecepatan sebelumnya $(0{,}3; -0{,}6)$, dan $\mathbf{g}^2 = (1{,}62; 1{,}62)$:

$$v_a = 0{,}5(0{,}3) + 0{,}4(1{,}6 - 1{,}9) + 0{,}8(1{,}62 - 1{,}9) = 0{,}15 - 0{,}12 - 0{,}224 = -0{,}194,$$
$$v_b = 0{,}5(-0{,}6) + 0{,}4(1{,}8 - 1{,}2) + 0{,}8(1{,}62 - 1{,}2) = -0{,}3 + 0{,}24 + 0{,}336 = 0{,}276,$$

sehingga $a = 1{,}9 - 0{,}194 = 1{,}706$ dan $b = 1{,}2 + 0{,}276 = 1{,}476$, dengan SSE $= 0{,}300504$. Inilah *gbest* akhir.
Perhatikan bahwa pada partikel ini suku kognitif dan suku sosial menarik ke arah yang sama untuk dimensi b, yaitu ke atas, sedangkan untuk dimensi a keduanya menarik ke bawah melawan inersia. Kombinasi ketiga dorongan itulah yang menempatkan partikel hampir tepat pada solusi kuadrat terkecil.

4.7. Rangkuman dan Verifikasi

Rekapitulasi selengkapnya disajikan pada Table 6. Nilai *gbest* menurun dari 1 menjadi 0,36; 0,3576; dan akhirnya 0,300504.

Table 6: Rekapitulasi tiga iterasi PSO untuk *function fitting*

Iterasi	Partikel 1		Partikel 2		Partikel 3		SSE($\mathbf{g}^t$)
	(a_1, b_1)	SSE	(a_2, b_2)	SSE	(a_3, b_3)	SSE	
0	$(1; 3)$	3	$(3; 0)$	21	$(2; 1)$	1	1
1	$(1{,}6; 1{,}8)$	0,36	$(2{,}4; 0{,}6)$	5,64	$(2; 1)$	1	0,36
2	$(1{,}9; 1{,}2)$	0,66	$(1{,}62; 1{,}62)$	0,3576	$(1{,}84; 1{,}32)$	0,5136	0,3576
3	$(1{,}706; 1{,}476)$	0,300504	$(1{,}23; 2{,}13)$	2,5926	$(1{,}584; 1{,}72)$	0,38688	0,300504

Perbandingan dengan solusi acuan disajikan pada Table 7. Setelah hanya tiga iterasi dengan tiga partikel, PSO menghasilkan $\hat{y} = 1{,}706x + 1{,}476$ sedangkan kuadrat terkecil menghasilkan $\hat{y} = 1{,}7x + 1{,}5$. Selisih galatnya hanya 0,000504, yaitu sekitar 0,17 persen di atas optimum.

Table 7: Perbandingan hasil PSO tiga iterasi terhadap solusi kuadrat terkecil

Metode	a	b	SSE	Selisih terhadap optimum
Kuadrat terkecil (analitik)	1,7	1,5	0,3	0
PSO, 3 partikel, 3 iterasi	1,706	1,476	0,300504	0,000504
PSO, 30 partikel, 100 iterasi	1,699968	1,500079	0,300000	$< 10^{-6}$

Hasil ini mengandung pelajaran penting. Pada persoalan regresi linear, PSO jelas kalah efisien dibandingkan kuadrat terkecil yang memberikan jawaban eksak dalam satu langkah. Nilai PSO terletak pada keluwesannya. Persamaan (7) dapat diganti dengan model tak linear, dengan norma galat lain seperti nilai mutlak sisaan yang tidak terdiferensialkan, atau dengan fungsi tujuan yang hanya dapat dihitung melalui simulasi. Pada semua situasi tersebut persamaan normal tidak lagi tersedia, sementara PSO tetap berjalan tanpa perubahan struktur sedikit pun.

5. Implementasi Python

5.1. Inti Algoritma

Listing 1 menyajikan implementasi PSO dalam bentuk vektor menggunakan NumPy. Seluruh partikel diproses serentak sebagai matriks berukuran $N \times D$, sehingga tidak diperlukan perulangan bersarang. Pemetaan antara kode dan persamaan bersifat langsung: baris pembaruan `V` adalah Persamaan (1), baris pembaruan `X` adalah Persamaan (2), dan blok penyegaran memori adalah Persamaan (3).

Listing 1: Inti algoritma PSO dengan NumPy

```python
import numpy as np

def pso_core(fitness, batas, n_partikel=30, n_iterasi=100,
             w=0.7298, c1=1.49618, c2=1.49618, seed=None):
    """PSO baku (topologi gbest) untuk MINIMISASI."""
    rng = np.random.default_rng(seed)
    batas = np.asarray(batas, dtype=float)
    lo, hi = batas[:, 0], batas[:, 1]
    d = len(batas)

    X = rng.uniform(lo, hi, size=(n_partikel, d))
    V = rng.uniform(-(hi - lo), hi - lo, size=(n_partikel, d)) * 0.1

    P, fP = X.copy(), fitness(X) # pbest dan nilainya
    idx = int(np.argmin(fP))
    G, fG = P[idx].copy(), float(fP[idx]) # gbest dan nilainya

    for _ in range(n_iterasi):
        r1 = rng.random((n_partikel, d))
        r2 = rng.random((n_partikel, d))

        V = w * V + c1 * r1 * (P - X) + c2 * r2 * (G - X) # Pers. (1)
        X = np.clip(X + V, lo, hi) # Pers. (2)

        fX = fitness(X) # Pers. (3)
        lebih_baik = fX < fP
        P[lebih_baik], fP[lebih_baik] = X[lebih_baik], fX[lebih_baik]

        idx = int(np.argmin(fP))
        if fP[idx] < fG:
            G, fG = P[idx].copy(), float(fP[idx])
    return G, fG
```

Tiga hal patut diperhatikan. *Pertama*, bilangan acak dibangkitkan berukuran $N \times D$, artinya setiap dimensi setiap partikel memperoleh nilai r_1 dan r_2 sendiri. Praktik ini merupakan bentuk baku PSO dan berbeda dengan penyederhanaan pada Bagian 4 yang menggunakan satu pasang bilangan acak per partikel. *Kedua*, perintah `np.clip` menjaga partikel tetap berada dalam ruang pencarian; tanpa pembatas ini partikel dapat melesat jauh ke luar daerah yang bermakna. *Ketiga*, pembaruan *gbest* menggunakan pembanding tegas, sehingga nilai *gbest* dijamin tidak pernah memburuk.

5.2. Menyusun Fungsi Fitness

Bagi pengguna, bagian tersulit dari PSO biasanya bukan algoritmanya melainkan perumusan fungsi *fitness*. Listing 2 memperlihatkan fungsi *fitness* untuk kedua studi kasus. Perhatikan bahwa fungsi *fitness* menerima seluruh kawanan sekaligus dan mengembalikan larik berisi N nilai.

Listing 2: Fungsi fitness untuk kedua studi kasus

```python
# Kasus 1: minimisasi f(x) = x^2 - 4x + 5
def f1d(X):
    x = X[:, 0]
    return x * x - 4 * x + 5

G, fG = pso_core(f1d, [(-10, 10)], seed=42)
print(G, fG) # -> [2.000000] 1.000000

# Kasus 2: function fitting y = a*x + b
xs = np.array([1., 2., 3., 4.])
ys = np.array([3., 5., 7., 8.])

def fitness_fit(P): # P berukuran (n_partikel, 2)
    pred = P[:, [0]] * xs + P[:, [1]] # broadcasting -> (n_partikel, 4)
    return ((ys - pred) ** 2).sum(axis=1)

G, fG = pso_core(fitness_fit, [(-10, 10), (-10, 10)], seed=42)
print(G, fG) # -> [1.699968 1.500079] 0.300000
```

Kolom `P[:, [0]]` berisi seluruh kandidat kemiringan dan `P[:, [1]]` berisi seluruh kandidat titik potong. Melalui *broadcasting*, satu ekspresi menghitung prediksi seluruh partikel pada seluruh titik data sekaligus. Hasil yang dicetak, yaitu $a = 1{,}699968$ dan $b = 1{,}500079$ dengan SSE $= 0{,}3$, konsisten dengan solusi analitik pada Persamaan (8).

5.3. Menghubungkan Kode dengan Perhitungan Tangan

Agar pembaca dapat memastikan pemahamannya benar, tersedia pula versi kode yang mereplikasi persis kedua simulasi tangan dengan memakai aritmetika pecahan eksak melalui modul `fractions`. Pendekatan ini menghilangkan galat pembulatan bilangan titik-kambang sehingga keluaran program dapat dibandingkan digit demi digit dengan Table 3 dan Table 6. Potongan intinya ditampilkan pada Listing 3.

Listing 3: Replikasi eksak simulasi tangan dengan aritmetika pecahan

```
from fractions import Fraction as F

w, c1, c2, r = F(1, 2), F(1), F(1), F(1, 2) # r1 = r2 = 0.5 dibekukan
X = [F(0), F(3), F(5)] # posisi awal
V = [F(0), F(0), F(0)]
P, fP = X[:], [x * x - 4 * x + 5 for x in X]
G = P[min(range(3), key=lambda i: fP[i])]

for t in range(1, 4):
    for i in range(3):
        V[i] = w * V[i] + c1 * r * (P[i] - X[i]) + c2 * r * (G - X[i])
        X[i] = X[i] + V[i]
        fx = X[i] * X[i] - 4 * X[i] + 5
        if fx < fP[i]:
            fP[i], P[i] = fx, X[i] # perbarui pbest
    G = P[min(range(3), key=lambda i: fP[i])] # perbarui gbest
    print(t, float(G), float(min(fP)))
# -> 1 1.5 1.25
# -> 2 2.25 1.0625
# -> 3 1.875 1.015625
```

Kesesuaian antara ketiga sumber, yaitu perhitungan tangan, solusi analitik, dan keluaran program, membentuk kerangka verifikasi ganda. Apabila kelak pembaca menulis implementasi PSO sendiri lalu memperoleh angka yang berbeda dari Table 3 pada konfigurasi yang sama, maka dapat dipastikan terdapat kekeliruan dalam kodenya. Kekeliruan yang paling sering ditemui adalah pembaruan posisi menggunakan kecepatan lama alih-alih kecepatan baru, penyegaran *gbest* yang dilakukan di dalam perulangan partikel sehingga informasi bocor dalam satu iterasi yang sama, serta penggunaan satu bilangan acak untuk suku kognitif dan suku sosial sekaligus.

6. Diskusi

6.1. Pemilihan Parameter

Simulasi pada Bagian 3 memperlihatkan secara gamblang mengapa bobot inersia penting. Ketika kawanan runtuh pada satu titik di iterasi kedua, satu-satunya yang menyelamatkan pencarian adalah momentum tersimpan pada suku inersia. Apabila $w = 0$, ketiga partikel akan terkunci permanen. Sebaliknya, nilai w yang terlalu besar membuat lintasan partikel membesar tanpa batas dan pencarian gagal mengendap. Kajian kestabilan menunjukkan bahwa lintasan partikel akan konvergen apabila parameter memenuhi hubungan tertentu antara w dan $c_1 + c_2$, dan konfigurasi $w = 0{,}7298$ dengan $c_1 = c_2 = 1{,}49618$ berada di dalam daerah stabil tersebut [4–6]. Strategi yang lazim dipakai adalah menurunkan w secara linear sepanjang iterasi, sehingga kawanan mengeksplorasi secara luas pada awal pencarian lalu mengeksploitasi secara halus pada akhir pencarian [3].

6.2. Keterbatasan

PSO tidak memberikan jaminan optimalitas. Yang dijamin hanyalah bahwa nilai *gbest* tidak memburuk, bukan bahwa nilai tersebut merupakan optimum global. Kelemahan yang paling menonjol adalah konvergensi prematur, yaitu keadaan ketika seluruh kawanan mengerumun di sekitar satu optimum lokal sehingga suku kognitif dan sosial menyusut mendekati nol dan pencarian terhenti. Bentuk ekstremnya telah terlihat pada iterasi kedua Bagian 3. Beberapa penawarnya adalah penggunaan topologi ketetanggaan yang lebih longgar seperti topologi cincin, yang memperlambat penyebaran informasi sehingga keragaman kawanan bertahan lebih lama [9], penggunaan ukuran kawanan yang memadai, serta pemantauan keragaman posisi partikel sebagai isyarat perlunya penyalaan ulang sebagian populasi.
Keterbatasan kedua bersifat lebih mendasar. Sebagaimana seluruh metaheuristik, kinerja PSO bergantung pada karakteristik persoalan. Tidak ada satu algoritma yang unggul pada semua kelas persoalan, sehingga perbandingan empiris lintas algoritma pada sekumpulan fungsi uji baku tetap diperlukan sebelum memilih metode untuk suatu penerapan [14].

6.3. Varian dan Perkembangan

Sejak diperkenalkan, PSO telah melahirkan sangat banyak varian. Beberapa arah pengembangan yang menonjol antara lain penambahan faktor konstriksi untuk menjamin kestabilan lintasan [4], penyesuaian parameter secara adaptif sepanjang iterasi, perluasan ke ruang diskret untuk persoalan kombinatorial, serta penghibridan dengan algoritma lain guna memadukan kekuatan eksplorasi dan eksploitasi.

Kelompok riset di Telkom University turut aktif pada bidang kecerdasan kawanan ini. Sebuah kajian pembanding yang menelaah beberapa algoritma kawanan mutakhir pada persoalan optimasi kontinu memperlihatkan bahwa tidak ada algoritma yang mendominasi seluruh fungsi uji, sehingga pemilihan algoritma harus berbasis karakteristik persoalan [14]. Pada ranah pengembangan algoritma baru, diusulkan Komodo Mlipir Algorithm yang mengadopsi perilaku komodo beserta gerak *mlipir* dan diuji pada dua puluh tiga fungsi tolok ukur baku dengan pembanding sejumlah metaheuristik mutakhir termasuk PSO [12]. Pada arah yang berbeda, algoritma Rao dikembangkan dengan skema evolusioner untuk memperbaiki keseimbangan eksplorasi dan eksploitasi [13]. Pada ranah terapan, pendekatan kawanan diskret digunakan untuk menyelesaikan *Capacitated Vehicle Routing Problem* melalui penggabungan pengelompokan simpul dan optimasi rute [15]. Karya-karya tersebut menempatkan PSO sebagai tolok ukur pembanding sekaligus sebagai fondasi konseptual, yang menegaskan bahwa pemahaman menyeluruh atas PSO tetap menjadi prasyarat bagi siapa pun yang hendak mendalami bidang ini.

7. Kesimpulan

Artikel ini menjawab pertanyaan "apa itu PSO" bukan melalui uraian naratif semata, melainkan melalui perhitungan yang seluruhnya dapat diverifikasi. PSO ditunjukkan sebagai algoritma yang hanya bertumpu pada dua persamaan sederhana, dengan setiap sukunya memiliki tafsir geometris yang jelas: inersia menjaga momentum, suku kognitif menarik partikel ke pengalaman terbaiknya sendiri, dan suku sosial menariknya ke pengalaman terbaik kawanan.

Dua simulasi yang disajikan menegaskan kelayakan pendekatan ini. Pada minimisasi $f(x) = x^2 - 4x + 5$ dengan tiga partikel dan tiga iterasi, galat terhadap optimum menyusut dari 1,0 menjadi 0,015625. Pada *function fitting* garis lurus terhadap empat titik data, jumlah kuadrat galat menurun dari 1 menjadi 0,300504, hanya 0,17 persen di atas solusi kuadrat terkecil sebesar 0,3. Kedua hasil tersebut direproduksi persis oleh program Python yang disertakan.

Nilai kebaruan tulisan ini terletak pada kerangka verifikasi gandanya. Dengan membekukan bilangan acak dan memilih persoalan yang solusi analitiknya diketahui, perhitungan tangan, solusi analitik, dan keluaran program dapat saling menguji. Kerangka tersebut bermanfaat ganda: sebagai alat ajar yang membongkar kotak hitam algoritma, dan sebagai prosedur pengujian bagi siapa pun yang menulis implementasi PSO sendiri.

Dua fenomena yang teramati dalam simulasi juga layak digarisbawahi karena jarang ditampilkan secara eksplisit pada bahan ajar, yaitu lenyapnya suku kognitif dan sosial ketika sebuah partikel bertepatan dengan *pbest* sekaligus *gbest*, serta keruntuhan seluruh kawanan pada satu titik yang menjadi wujud paling ekstrem konvergensi prematur. Keduanya memberi pembenaran intuitif atas pentingnya bobot inersia dan atas perlunya menjaga keragaman kawanan.

Pengembangan lanjutan yang disarankan mencakup perluasan kerangka verifikasi serupa untuk model tak linear, penyusunan simulasi tangan bagi varian PSO dengan topologi cincin dan dengan faktor konstriksi, serta penambahan penanganan kendala agar tutorial ini dapat langsung dipakai pada persoalan rekayasa yang lebih realistis.

Article Information

Author's Contributions: Hilal Huda merancang kerangka tutorial, menyusun simulasi perhitungan tangan, dan menulis naskah awal. Heny Pratiwi mengembangkan serta memverifikasi implementasi Python, menyiapkan visualisasi, melakukan telaah pustaka, dan menyunting naskah akhir. Kedua penulis telah membaca dan menyetujui naskah versi akhir.

Acknowledgments: Penulis mengucapkan terima kasih kepada rekan-rekan di School of Computing, Telkom University, atas diskusi yang membangun selama penyusunan artikel ini.

Conflict of Interest: The authors declare that there is no conflict of interest regarding the publication of this paper.

Support Section: Penelitian ini tidak menerima pendanaan khusus dari lembaga mana pun.

Ethical Approval: It is declared that during the preparation process of this study, scientific and ethical principles were followed and all the studies benefited from are stated in the bibliography.

Availability: Seluruh kode Python yang digunakan pada artikel ini tersedia sebagai berkas pendamping `pso_tutorial.py`.

Artificial Intelligence Statement: [Sesuaikan dengan kondisi sebenarnya sebelum pengiriman naskah.]

Plagiarism Statement: This article has been scanned by iThenticate™.

References

[1] J. Kennedy and R. Eberhart, *Particle swarm optimization*, in Proc. IEEE International Conference on Neural Networks, Perth, Australia, 1995, pp. 1942–1948.
[2] R. C. Eberhart and J. Kennedy, *A new optimizer using particle swarm theory*, in Proc. 6th International Symposium on Micro Machine and Human Science, Nagoya, Japan, 1995, pp. 39–43.
[3] Y. Shi and R. C. Eberhart, *A modified particle swarm optimizer*, in Proc. IEEE International Conference on Evolutionary Computation, Anchorage, USA, 1998, pp. 69–73.
[4] M. Clerc and J. Kennedy, *The particle swarm — explosion, stability, and convergence in a multidimensional complex space*, IEEE Transactions on Evolutionary Computation, **6** (1) (2002), 58–73.
[5] I. C. Trelea, *The particle swarm optimization algorithm: convergence analysis and parameter selection*, Information Processing Letters, **85** (6) (2003), 317–325.
[6] F. van den Bergh and A. P. Engelbrecht, *A study of particle swarm optimization particle trajectories*, Information Sciences, **176** (8) (2006), 937–971.
[7] R. Poli, J. Kennedy, and T. Blackwell, *Particle swarm optimization: An overview*, Swarm Intelligence, **1** (1) (2007), 33–57.
[8] D. Bratton and J. Kennedy, *Defining a standard for particle swarm optimization*, in Proc. IEEE Swarm Intelligence Symposium, Honolulu, USA, 2007, pp. 120–127.
[9] J. Kennedy and R. Mendes, *Population structure and particle swarm performance*, in Proc. IEEE Congress on Evolutionary Computation, Honolulu, USA, 2002, pp. 1671–1676.

[10] R. Poli, *Analysis of the publications on the applications of particle swarm optimisation*, Journal of Artificial Evolution and Applications, **2008** (2008), Article ID 685175, 10 pages.
[11] A. P. Engelbrecht, *Fundamentals of Computational Swarm Intelligence*, John Wiley & Sons, Chichester, 2005.
[12] S. Suyanto, A. A. Ariyanto, and A. F. Ariyanto, *Komodo Mlipir Algorithm*, Applied Soft Computing, **114** (2022), Article ID 108043. https://doi.org/10.1016/j.asoc.2021.108043
[13] S. Suyanto, A. T. Wibowo, S. Al Faraby, S. Saadah, and R. Rismala, *Evolutionary Rao algorithm*, Journal of Computational Science, **53** (2021), Article ID 101368. https://doi.org/10.1016/j.jocs.2021.101368
[14] E. Indramaya and S. Suyanto, *Comparative study of recent swarm algorithms for continuous optimization*, Procedia Computer Science, **179** (2021), 685–695. https://doi.org/10.1016/j.procs.2021.01.056
[15] U. Abdillah and S. Suyanto, *Clustering nodes and discretizing movement to increase the effectiveness of HEFA for a CVRP*, International Journal of Advanced Computer Science and Applications, **11** (4) (2020), 774–779.