BAB III
METODE PENELITIAN

Bab ini membahas mengenai alur metode penelitian optimasi parameter *Triple Exponential Smoothing* (TES) menggunakan *Differential Evolution* (DE). Pada penelitian ini, *Python* yang diakses melalui *Google Colab* dipilih sebagai lingkungan pemrograman karena kemudahannya dalam akses *cloud*, sehingga tidak memerlukan instalasi lokal. Selanjutnya, dipaparkan alur optimasi parameter dan model peramalan.

3.1 Pendekatan Penelitian

Salah satu yang terpenting dalam melakukan penelitian adalah menentukan pendekatan penelitian yang digunakan. Metode yang digunakan dalam penelitian ini adalah metode penelitian deskriptif dengan pendekatan kuantitatif. Menurut Sudjana (2004), penelitian deskriptif didefinisikan sebagai penelitian yang bertujuan untuk memberikan deskripsi sistematis mengenai fenomena yang sedang diamati. Menurut Kasiram (2008), pendekatan kuantitatif dipandang sebagai pendekatan yang menggunakan angka, mulai dari pengumpulan data, interpretasi data, hingga penyajian hasil penelitian. Jadi, metode penelitian deskriptif dengan pendekatan kuantitatif diartikan sebagai suatu metode penelitian yang bertujuan untuk memberikan deskripsi sistematis mengenai fenomena yang diamati dengan menggunakan angka dalam proses pengumpulan, interpretasi, dan penyajian data hasil penelitian.

3.2 Jenis dan Sumber Data

Penelitian ini menggunakan data sekunder berupa harga saham Apple Inc. Data tersebut diperoleh dari sumber publik, yaitu situs web finansial Yahoo Finance (https://finance.yahoo.com/quote/AAPL/history) dan diunduh pada tanggal 12 Desember 2023. Fokus penelitian ini adalah data harga pembukaan saham Apple Inc. yang mencakup periode dari tanggal 3 Agustus 2020 hingga 31 Agustus 2022. Data tersebut kemudian dikonversi menjadi data runtun waktu harian.

3.3 Prosedur Algoritma *Differential Evolution* (DE)

Langkah-langkah penyelesaian algoritma DE dijabarkan sebagai berikut:

1) Melakukan inisialisasi populasi dengan membangkitkan populasi awal dari solusi acak dalam ruang pencarian. Setiap solusi mewakili individu dalam populasi.
2) Melakukan mutasi untuk untuk setiap individu dalam populasi. Ini melibatkan kombinasi dari beberapa vektor untuk menghasilkan vektor mutan.
3) Melakukan rekombinasi, yaitu penggabungan antara vektor target dan vektor mutan untuk memperoleh vektor baru dalam populasi.
4) Melakukan seleksi dengan membandingkan antara vektor mutan dan vektor uji dengan membandingkan nilai evaluasi (*fitness value*) keduanya. Pilih individu dengan nilai evaluasi terbaik untuk dilanjutkan ke dalam generasi berikutnya.
5) Berhenti ketika banyak generasi yang diinginkan telah tercapai.

3.4 Prosedur Pemodelan *Triple Exponential Smoothing* (TES)

Langkah-langkah pemodelan TES dijabarkan sebagai berikut:

1) Melakukan inisialisasi untuk level (S_0), tren (T_0), dan musiman ($I_{-20} - I_0$).
2) Menentukan besar parameter α, β, γ dengan nilai $0 < \alpha, \beta, \gamma < 1$.
3) Menentukan panjang musiman (m).
4) Melakukan perhitungan pemulusan level menggunakan parameter α pada variabel S_t.
5) Melakukan perhitungan pemulusan tren menggunakan parameter β pada variabel T_t.
6) Melakukan perhitungan pemulusan musiman menggunakan parameter γ pada variabel I_t.
7) Melakukan perhitungan nilai peramalan setelah dilakukan perhitungan nilai pemulusan kesuluruhan, pemulusan tren, dan pemulusan musiman.

8) Menilai akurasi model berdasarkan nilai *Mean Absolute Percentage Error* (MAPE).

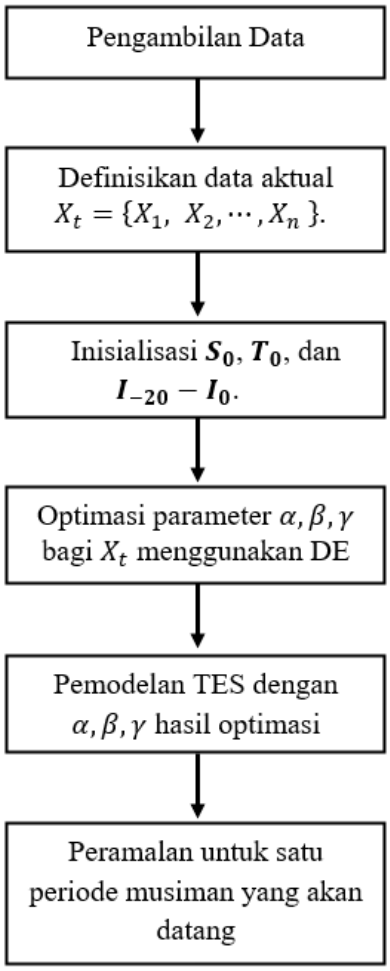
3.5 Prosedur Optimasi Parameter TES Menggunakan Algoritma DE

Langkah-langkah optimasi parameter TES menggunakan algoritma DE dijabarkan sebagai berikut:

1) Definisikan data aktual harga pembukaan saham Apple Inc. (periode Agustus 2020 - Agustus 2022) menjadi $X_t = \{X_1,\ X_2, \cdots, X_n\}$.
2) Melakukan inisialisasi untuk level (S_0), tren (T_0), dan musiman ($I_{-20} - I_0$).
3) Melakukan optimasi parameter α, β, γ bagi data runtun waktu X_t menggunakan algoritma DE. Selanjutnya, untuk setiap kombinasi unik α, β, dan γ, dibangun model TES dan dihitung nilai MAPE sebagai fungsi fitness pada penelitian ini. Vektor dengan nilai MAPE yang lebih rendah akan diprioritaskan sebagai orang tua untuk generasi berikutnya.
4) Setiap langkah pada prosedur algoritma DE dilakukan, yaitu inisialisasi populasi awal, mutasi, rekombinasi, dan seleksi, hingga mencapai kondisi berhenti yang ditentukan, yaitu banyak generasi maksimum (G).
5) Setelah diperoleh nilai α, β, dan γ terbaik serta nilai MAPE terkecil, dibangun model TES dengan menggunakan parameter-parameter hasil optimasi algoritma DE.
6) Melakukan peramalan untuk satu periode musiman yang akan datang.

3.6 Alur Penelitian

Berikut adalah bentuk diagram alur penelitian ini.



Gambar 3.1 Diagram Alur Penelitian

3.7 Implementasi Algoritma DE untuk Fungsi-Fungsi *Benchmark* dalam Masalah Optimasi Global

Menurut Jamil dan Yang (2013), fungsi uji sangat penting untuk memvalidasi dan membandingkan kinerja algoritma optimasi. Idealnya, fungsi uji harus memiliki sifat yang beragam agar benar-benar berguna untuk menguji algoritma baru secara objektif. Oleh karena itu, pada penelitian ini diberikan impelementasi algoritma DE dalam pencarian optimal global untuk beberapa fungsi *benchmark*. Fungsi-fungsi tersebut diperoleh dari jurnal: "*A Literature*

Survey of Benchmark Functions for Global Optimisation Problems" (Jamil & Yang, 2013). Beberapa fungsi *benchmark* yang digunakan untuk memvalidasi algoritma DE pada penelitian ini antara lain:

a) Fungsi *Ackley 2*

Rumus fungsi *Ackley* 2 sebagai nilai *fitness* (Ackley, 2012):

$$f(x,y) = -200\, e^{-0{,}02\sqrt{x^2+y^2}} \qquad (3.1)$$

dengan syarat: $-32 \leq x,y \leq 32$.

Minimum global terletak di titik asal $(x,y) = (0{,}0)$ dimana

$f(x,y) = -200$.

Berikut ini adalah kode untuk pencarian solusi optimal global fungsi *Ackley* 2 menggunakan algoritma DE di *Google Colab:*

```
def mutation(x, F):
    return x[0] + F * (x[1] - x[2])

def check_bounds(mutated, bounds):
    mutated_bound = [np.clip(mutated[i], bounds[i][0], bounds[i][1]) for i in
range(len(bounds))]
    return mutated_bound

def crossover(mutated, target, dims, cr):
    p = np.random.rand(dims)
    trial = [mutated[i] if p[i] < cr else target[i] for i in range(dims)]
    return trial

def ackley(x):
    x, y = x[0], x[1]
    return -200 * np.exp(-0.02 * np.sqrt(x**2 + y**2))

def objective_function(x):
    return ackley(x)

def differential_evolution(pop_size, bounds, iter, F, cr):
    bounds = np.array(bounds)
    pop = bounds[:, 0] + (np.random.rand(pop_size, len(bounds)) * (bounds[:, 1] -
bounds[:, 0]))
```

```
    obj_all = [objective_function(ind) for ind in pop]

    best_vector = pop[np.argmin(obj_all)]
    best_obj = np.min(obj_all)
    prev_obj = best_obj

    obj_iter = []

    for i in range(iter):
        for j in range(pop_size):
            candidates = [candidate for candidate in range(pop_size) if candidate !=
j]
            a, b, c = pop[np.random.choice(candidates, 3, replace=False)]

            mutated = mutation([a, b, c], F)
            mutated = check_bounds(mutated, bounds)

            trial = crossover(mutated, pop[j], len(bounds), cr)

            obj_target = objective_function(pop[j])
            obj_trial = objective_function(trial)

            if obj_trial < obj_target:
                pop[j] = trial
                obj_all[j] = obj_trial

        best_obj = np.min(obj_all)

        if best_obj < prev_obj:
            best_vector = pop[np.argmin(obj_all)]
            prev_obj = best_obj
            obj_iter.append(best_obj)

            print('Iteration: %d f([%s]) = %.5f' % (i, np.around(best_vector,
decimals=5), best_obj))
    return [best_vector, best_obj, obj_iter]

# Batasan untuk domain x dan y
bounds = [(-32, 32), (-32, 32)]

# Optimasi fungsi Ackley menggunakan differential evolution
pop_size = 50
iter = 100
F = 0.2
cr = 0.7
```

```
solution = differential_evolution(pop_size, bounds, iter, F, cr)

# Solusi optimum
x_opt, y_opt = solution[0]
min_value = solution[1]

print("Optimal (x, y): ", (x_opt, y_opt))
print("Nilai minimum: ", min_value)
```

Tabel 3.1 menunjukkan pencarian solusi optimal global fungsi *Ackley* 2 menggunakan algoritma DE dengan beberapa generasi yang tercantum.

Tabel 3.1 Generasi Pencarian Solusi Optimal Global Fungsi *Ackley* 2

Generasi	x	y	$f(x,y)$
1	4,31437	-2,88951	-180,27176
2	1,55251	0,65092	-193,37833
4	-0,7153	0,65092	-196,16863
7	0,21699	0,26415	-199,63727
8	0,07963	-0,00749	-199,68033
16	-0,04205	-0,04997	-199,73894
17	0,02525	0,05545	-199,75645
18	0,00477	-0,05823	-199,76645
19	-0,00966	-0,2687	-199,88582
20	-0,01877	-0,00769	-199,91886
23	-0,00240	-0,00025	-199,99036
24	0,00150	-0,00057	-199.99359
29	-0,00134	-0,00029	-199,99452
30	0,00053	0,00071	-199,99644
31	0,00020	-0,00046	-199,99801
⋮	⋮	⋮	⋮
96	0	0	-200,00000
98	0	0	-200,00000
100	0	0	-200,00000

Berdasarkan Tabel 3.1, dapat dilihat bahwa fungsi *Ackley* 2 memiliki titik global minimum $(x, y) = (0,0)$ dengan nilai *fitness* minimum

$f(x, y) = -200$. Hasil pencarian solusi optimal global fungsi *Ackley* 2 menggunakan algoritma DE sama dengan nilai global minimum dari jurnal penelitian Ackley (2012).

b) Fungsi *Booth*

Rumus fungsi *Booth* sebagai nilai *fitness* (Jamil & Yang, 2013):

$$f(x, y) = (x + 2y - 7)^2 + (2x + y - 5)^2 \qquad (3.2)$$

dengan syarat: $-10 \leq x, y \leq 10$.

Minimum global terletak di titik asal $(x, y) = (1,3)$ dimana

$f(x, y) = 0$.

Berikut ini adalah kode untuk pencarian solusi optimal global fungsi *Booth* menggunakan algoritma DE di *Google Colab:*

```
def mutation(x, F):
    return x[0] + F * (x[1] - x[2])

def check_bounds(mutated, bounds):
    mutated_bound = [np.clip(mutated[i], bounds[i][0], bounds[i][1]) for i in
range(len(bounds))]
    return mutated_bound

def crossover(mutated, target, dims, cr):
    p = np.random.rand(dims)
    trial = [mutated[i] if p[i] < cr else target[i] for i in range(dims)]
    return trial

def booth(x):
    x, y = x[0], x[1]
    return (x + 2*y - 7)**2 + (2*x + y - 5)**2

def objective_function(x):
    return booth(x)
```

```
def differential_evolution(pop_size, bounds, iter, F, cr):
    bounds = np.array(bounds)  # Convert bounds to a NumPy array
    pop = bounds[:, 0] + (np.random.rand(pop_size, len(bounds)) * (bounds[:, 1] -
bounds[:, 0]))

    obj_all = [objective_function(ind) for ind in pop]

    best_vector = pop[np.argmin(obj_all)]
    best_obj = np.min(obj_all)
    prev_obj = best_obj

    obj_iter = []

    for i in range(iter):
        for j in range(pop_size):
            candidates = [candidate for candidate in range(pop_size) if candidate !=
j]
            a, b, c = pop[np.random.choice(candidates, 3, replace=False)]

            mutated = mutation([a, b, c], F)
            mutated = check_bounds(mutated, bounds)

            trial = crossover(mutated, pop[j], len(bounds), cr)

            obj_target = objective_function(pop[j])
            obj_trial = objective_function(trial)

            if obj_trial < obj_target:
                pop[j] = trial
                obj_all[j] = obj_trial

        best_obj = np.min(obj_all)

        if best_obj < prev_obj:
            best_vector = pop[np.argmin(obj_all)]
            prev_obj = best_obj
            obj_iter.append(best_obj)

            print('Iteration: %d f([%s]) = %.5f' % (i, np.around(best_vector,
decimals=5), best_obj))
    return [best_vector, best_obj, obj_iter]

# Batasan untuk domain x dan y
bounds = [(-10, 10), (-10, 10)]
```

```
# Optimisasi fungsi Booth menggunakan differential evolution
pop_size = 50
iter = 100
F = 0.2
cr = 0.7
solution = differential_evolution(pop_size, bounds, iter, F, cr)

# Solusi optimum
x_opt, y_opt = solution[0]
min_value = solution[1]

print("Optimal (x, y): ", (x_opt, y_opt))
print("Nilai minimum: ", min_value)
```

Tabel 3.2 menunjukkan pencarian solusi optimal global fungsi *Booth* menggunakan algoritma DE dengan beberapa generasi yang tercantum.

Tabel 3.2 Generasi Pencarian Solusi Optimal Global Fungsi *Booth*

Generasi	x	y	$f(x,y)$
1	0,05784	3,90891	1,71821
5	1,93283	2,11201	1,66673
6	0,75062	3,67006	1,21909
7	1,42970	2,50653	0,44443
8	0,94557	3,08455	0,01374
11	1,00848	2,97975	0,00104
23	0,99779	2,99270	0,00042
25	1,00015	2,99864	0,00001
35	0,999881	3,00110	0,00000
39	1,00066	2,99932	0,00000
41	1,00060	2,99952	0,00000
42	0,99976	3,00049	0,00000
43	0,99959	3,00040	0,00000
44	0,99981	3,00011	0,00000
46	0,99987	3,00006	0,00000
⋮	⋮	⋮	⋮
90	1	3	0,00000

Generasi	x	y	$f(x,y)$
98	1	3	0,00000
100	1	3	0,00000

Berdasarkan Tabel 3.2, dapat dilihat bahwa fungsi *Booth* memiliki titik global minimum $(x, y) = (0,0)$ dengan nilai *fitness* minimum

$f(x, y) = -200$. Hasil pencarian solusi optimal global fungsi *Booth* menggunakan algoritma DE sama dengan nilai global minimum dari jurnal penelitian Jamil & Yang (2013).

c) Fungsi *Adjiman*

Rumus fungsi *Adjiman* sebagai nilai *fitness* (Adjiman dkk., 1998):

$$f(x, y) = \cos x \sin y - \frac{x}{y^2 + 1} \qquad (3.3)$$

dengan syarat: $-1 \leq x \leq 2, -1 \leq y \leq 1$.

Minimum global terletak di titik asal $(x, y) = (2,\ 0{,}10578)$ dimana

$f(x, y) = -2{,}02181$.

Berikut ini adalah kode untuk pencarian solusi optimal global fungsi *Adjiman* menggunakan algoritma DE di *Google Colab:*

```
from math import cos, sin


def mutation(x, F):
    return x[0] + F * (x[1] - x[2])


def check_bounds(mutated, bounds):
    mutated_bound = [np.clip(mutated[i], bounds[i][0], bounds[i][1]) for i in
range(len(bounds))]
    return mutated_bound


def crossover(mutated, target, dims, cr):
    p = np.random.rand(dims)
    trial = [mutated[i] if p[i] < cr else target[i] for i in range(dims)]
    return trial
```

```
def adjiman(x):
    x, y = x[0], x[1]
    return (cos(x) * sin(y)) - (x / (y**2 + 1))

def objective_function(x):
    return adjiman(x)

def differential_evolution(pop_size, bounds, iter, F, cr):
    bounds = np.array(bounds)  # Convert bounds to a NumPy array
    pop = bounds[:, 0] + (np.random.rand(pop_size, len(bounds)) * (bounds[:, 1] -
bounds[:, 0]))

    obj_all = [objective_function(ind) for ind in pop]

    best_vector = pop[np.argmin(obj_all)]
    best_obj = np.min(obj_all)
    prev_obj = best_obj

    obj_iter = []

    for i in range(iter):
        for j in range(pop_size):
            candidates = [candidate for candidate in range(pop_size) if candidate !=
j]
            a, b, c = pop[np.random.choice(candidates, 3, replace=False)]

            mutated = mutation([a, b, c], F)
            mutated = check_bounds(mutated, bounds)

            trial = crossover(mutated, pop[j], len(bounds), cr)

            obj_target = objective_function(pop[j])
            obj_trial = objective_function(trial)

            if obj_trial < obj_target:
                pop[j] = trial
                obj_all[j] = obj_trial

        best_obj = np.min(obj_all)

        if best_obj < prev_obj:
            best_vector = pop[np.argmin(obj_all)]
            prev_obj = best_obj
            obj_iter.append(best_obj)
```

```
            print('Iteration: %d f([%s]) = %.5f' % (i, np.around(best_vector,
decimals=5), best_obj))
    return [best_vector, best_obj, obj_iter]

# Batasan untuk domain x dan y
bounds = [(-1, 2), (-1, 2)]

# Optimisasi fungsi Adjiman menggunakan differential evolution
pop_size = 50
iter = 100
F = 0.2
cr = 0.7
solution = differential_evolution(pop_size, bounds, iter, F, cr)

# Solusi optimum
x_opt, y_opt = solution[0]
min_value = solution[1]

print("Optimal (x, y): ", (x_opt, y_opt))
print("Nilai minimum: ", min_value)
```

Tabel 3.3 menunjukkan pencarian solusi optimal global fungsi *Adjiman* menggunakan algoritma DE dengan beberapa generasi yang tercantum.

Tabel 3.3 Generasi Pencarian Solusi Optimal Global Fungsi *Adjiman*

Generasi	x	y	$f(x,y)$
1	1,96349	0,04176	−1,97605
2	2	0,00397	−2,00162
3	2	0,18865	−2,00931
4	2	0,10177	−2,02178
9	2	0,10677	−2,02180
10	2	0,10533	−2,02181
12	2	0,10586	−2,02181
17	2	0,10584	−2,02181
18	2	0,10573	−2,02181
19	2	0,10579	−2,02181
20	2	0,10578	−2,02181

Generasi	x	y	$f(x,y)$
27	2	0,10578	−2,02181
28	2	0,10578	−2,02181
30	2	0,10578	−2,02181
35	2	0,10578	−2,02181
⋮	⋮	⋮	⋮
98	2	0,10578	−2,02181
99	2	0,10578	−2,02181
100	2	0,10578	−2,02181

Berdasarkan Tabel 3.3, dapat dilihat bahwa fungsi *Adjiman* memiliki titik global minimum $(x, y) = (2,\ 0{,}10578)$ dengan nilai *fitness* minimum $f(x,y) = -2{,}02181$. Hasil pencarian solusi optimal global fungsi *Adjiman* menggunakan algoritma DE sama dengan nilai global minimum dari jurnal penelitian Adjiman dkk. (1998).

d) Fungsi *Periodic*

Rumus fungsi *Periodic* sebagai nilai *fitness* (Ali dkk., 2005):

$$f(x,y) = 1 + sin^2\, x + sin^2\, y - 0{,}1\, e^{-(x^2+y^2)} \qquad (3.4)$$

dengan syarat: $-10 \leq x, y \leq 10$.

Minimum global terletak di titik asal $(x, y) = (0, 0)$ dimana $f(x,y) = 0{,}9$.

Berikut ini adalah kode untuk pencarian solusi optimal global fungsi *Periodic* menggunakan algoritma DE di *Google Colab:*

```
def mutation(x, F):
    return x[0] + F * (x[1] - x[2])


def check_bounds(mutated, bounds):
    mutated_bound = [np.clip(mutated[i], bounds[i][0], bounds[i][1]) for i in
range(len(bounds))]
```

```
    return mutated_bound

def crossover(mutated, target, dims, cr):
    p = np.random.rand(dims)
    trial = [mutated[i] if p[i] < cr else target[i] for i in range(dims)]
    return trial

def Periodic(x):
    x, y = x[0], x[1]
    term1 = 1 + np.sin(x)**2 + np.sin(y)**2
    term2 = -0.1 * np.exp(-(x**2 + y**2))
    result = term1 + term2
    return result

def objective_function(x):
    return Periodic(x)

def differential_evolution(pop_size, bounds, iter, F, cr):
    bounds = np.array(bounds)  # Convert bounds to a NumPy array
    pop = bounds[:, 0] + (np.random.rand(pop_size, len(bounds)) * (bounds[:, 1] -
bounds[:, 0]))

    obj_all = [objective_function(ind) for ind in pop]

    best_vector = pop[np.argmin(obj_all)]
    best_obj = np.min(obj_all)
    prev_obj = best_obj

    obj_iter = []

    for i in range(iter):
        for j in range(pop_size):
            candidates = [candidate for candidate in range(pop_size) if candidate !=
j]
            a, b, c = pop[np.random.choice(candidates, 3, replace=False)]

            mutated = mutation([a, b, c], F)
            mutated = check_bounds(mutated, bounds)

            trial = crossover(mutated, pop[j], len(bounds), cr)

            obj_target = objective_function(pop[j])
            obj_trial = objective_function(trial)

            if obj_trial < obj_target:
```

```
                pop[j] = trial
                obj_all[j] = obj_trial

        best_obj = np.min(obj_all)

        if best_obj < prev_obj:
            best_vector = pop[np.argmin(obj_all)]
            prev_obj = best_obj
            obj_iter.append(best_obj)

            print('Iteration: %d f([%s]) = %.5f' % (i, np.around(best_vector,
decimals=5), best_obj))
    return [best_vector, best_obj, obj_iter]

# Batasan untuk domain x dan y
bounds = [(-10, 10), (-10, 10)]

# Optimisasi fungsi Periodic menggunakan differential evolution
pop_size = 50
iter = 100
F = 0.2
cr = 0.7
solution = differential_evolution(pop_size, bounds, iter, F, cr)

# Solusi optimum
x_opt, y_opt = solution[0]
min_value = solution[1]

print("Optimal (x, y): ", (x_opt, y_opt))
print("Nilai minimum: ", min_value)
```

Tabel 3.4 menunjukkan pencarian solusi optimal global fungsi *Periodic* menggunakan algoritma DE dengan beberapa generasi yang tercantum.

Tabel 3.4 Generasi Pencarian Solusi Optimal Global Fungsi *Periodic*

Generasi	x	y	$f(x,y)$
1	0	6,38925	1,01121
4	0,02354	−3,11994	1,00102
14	0,06822	−0,17109	0,93697
16	−0,06035	−0,15268	0,92943
29	0	−0,11117	0,91354

Generasi	x	y	$f(x,y)$
30	0,10819	0,00584	0,91286
36	0	−0,05220	0,90299
41	−0,00527	−0,00232	0,90004
61	0,00219	−0,00144	0,90001
69	0,00013	0,00222	0,90001
71	0,00118	−0,00100	0,90000
73	0,00013	−0,00101	0,90000
74	−0,00064	−0,00010	0,90000
76	−0,00029	0,00035	0,90000
⋮	⋮	⋮	⋮
98	0	0	0,90000
99	0	0	0,90000
100	0	0	0,90000

Berdasarkan Tabel 3.4, dapat dilihat bahwa fungsi *Periodic* memiliki titik global minimum $(x, y) = (0, 0)$ dengan nilai *fitness* minimum

$f(x, y) = 0{,}9$. Hasil pencarian solusi optimal global fungsi *Periodic* menggunakan algoritma DE sama dengan nilai global minimum dari jurnal penelitian Ali dkk. (2005).

Hasil pencarian solusi optimal global untuk fungsi-fungsi *benchmark* menggunakan algoritma DE sama dengan nilai global minimum yang tercantum dalam jurnal-jurnal penelitian. Oleh karena itu, performa algoritma DE dalam menemukan nilai optimal global dapat dikatakan baik.