

Untersuchung von Particle Swarm Optimization zur Optimierung Gaußscher Prozessmodelle

Projektbericht zur Vorlesung Applied Optimization Techniques

für die
Prüfung zum Bachelor of Science

an der Fakultät für Wirtschaft
im Studiengang Wirtschaftsinformatik
in der Studienrichtung Data Science

an der
DHBW Ravensburg

Verfasser: Timo Heiß
Nick Hillebrand
Pascal Knoll
Tom Zehle

Kurs: WWIDS120
Wiss. Betreuer: Prof. Dr. Martin Zaefferer
Abgabedatum: 17.08.2022

Inhaltsverzeichnis

Abkürzungsverzeichnis	III
Abbildungsverzeichnis	IV
Listingsverzeichnis	V
1 Einleitung	1
2 Theoretische Grundlagen	3
2.1 Gaußsche Prozessmodelle und Maximum Likelihood Estimation	3
2.2 Particle Swarm Optimization	5
3 Empirische Untersuchungen	8
3.1 Implementierung	8
3.2 Erste Untersuchungen und Vergleich zu anderen Optimierungsalgorithmen	10
3.3 Weiterführende Experimente	13
3.3.1 Variation der Anzahl der Trainingsdatenpunkten	13
3.3.2 Variation der Dimensionsanzahl	15
3.3.3 Verwendung verschiedener Groundtruth-Funktionen	17
3.3.4 Einfluss von Rauschen (Noise)	18
3.3.5 Gesamtauswertung der Experimente	19
3.4 Untersuchung der Hyperparameter von PSO	20
4 Fazit	24
Literatur	26
Selbständigkeitserklärung	29

Abkürzungsverzeichnis

DE	Differential Evolution
GPM	Gaußsche(s) Prozessmodell(e)
MLE	Maximum Likelihood Estimation
PSO	Particle Swarm Optimization
RS	Random Search

Abbildungsverzeichnis

2.1	Particle Swarm Optimierung im drei dimensionalen Raum	7
3.1	Vergleichstabelle der Algorithmen	8
3.2	Groundtruth Funktion	10
3.3	Vorhersagen der mit verschiedenen Optimierern trainierten GPM . . .	11
3.4	Globale Sicht der Likelihood-Landschaft in den Suchgrenzen (links) und Zoom auf das Maximum (rechts)	11
3.5	Von den Optimierern gefundene Optima der Likelihood	12
3.6	Ausgewählte Schritte der Optimierung mit PSO	12
3.7	Einfluss der Sample-Größe (n) auf die maximale gefundene Likelihood .	14
3.8	Vorhersagen des GPM in Abhängigkeit der Anzahl an Trainingsdaten .	14
3.9	Maximale gefundene Likelihood in Abhängigkeit der Dimensionalität .	15
3.10	Boxplots für ausgewählte Dimensionszahlen	16
3.11	Verwendete Groundtruth-Funktionen	17
3.12	Gefundene maximale Likelihood der Optimierer für verschiedene Groundtruth- Funktionen	17
3.13	Likelihood von Groundtruth 3: Optimum (links) und Plateau (rechts) .	18
3.14	Optimierer bei unterschiedlichem Noise Scale: Differenz zu PSO . . .	18
3.15	Gesamtauswertung aller Experimente	19
3.16	Einfluss von Partikel- und Iterationszahl von PSO	20
3.17	Einfluss des Hyperparameters inertia (w)	21
3.18	Ausgewählte Optimierungsschritte einer PSO-Instanz mit $c_1=3$ und $c_2=0$	22
3.19	Ausgewählte Optimierungsschritte einer PSO-Instanz mit $c_1=0$ und $c_2=3$	22

Listingsverzeichnis

3.1	Funktion zur Generierung von Daten für die Untersuchungen	8
3.2	Random Search Algorithmus (eigene Implementierung)	9

1 Einleitung

Die Bewegung von Schwärmen, ob Vogel- oder Fischschwärme, ist seit jeher ein faszinierendes Phänomen der Natur. Die Schwärme bestehen aus einzelnen Tieren und scheinen sich dennoch als großes Ganzes fortzubewegen. Die Tiere scheinen willkürlich angeordnet und doch ist die Gesamtbewegung genauestens synchronisiert. Und wie durch eine magische Kraft scheint der Schwarm durch eine zentrale Kontrollinstanz gesteuert zu werden (vgl. Reynolds, 1987, S. 25).

Reynolds (1987, S. 28), der das Schwarmverhalten von Tieren für Computersimulationen untersuchte, identifizierte daraus drei Regeln für die Schwarmbewegungen:

1. Jedes Tier versucht Kollisionen mit seinen Nachbarn zu vermeiden.
2. Dabei passt es sich an die Geschwindigkeit und Richtung der Nachbartiere an.
3. Gleichzeitig möchte es nahe bei den benachbarten Tieren bleiben.

Heppner und Grenander (1990) erweitern diese Idee um das Konzept der Rastplätze: Neben dem Ziel, im Schwarm zu bleiben, möchte jedes Tier auch zu einem Rastplatz, wobei dieser Wunsch mit zunehmender Nähe zu einem Rastplatz stärker wird. Dadurch wird der gesamte Schwarm in Richtung des Rastplatzes gelenkt.

Die Idee, einen optimalen Rastplatz zu finden, kann nun verallgemeinert werden: Basierend auf den oben skizzierten Konzepten von *Reynolds* sowie *Heppner und Grenander* entwickelten *Kennedy und Eberhart (1995)* die Particle Swarm Optimization (PSO) und setzen die Idee des Schwarmverhaltens damit zur Lösung von Optimierungsproblemen ein. Das Optimierungsverfahren PSO ist nun Untersuchungsgegenstand der vorliegenden Arbeit. Konkret wird das Optimierungsproblem der Maximum Likelihood Estimation (MLE) für Gaußsche Prozessmodelle (GPM) betrachtet. Dieses Problem soll mithilfe von PSO gelöst werden, wobei Funktionsweise und Eignung des Optimierungsverfahrens empirisch untersucht werden.

Aus dieser Zielstellung ergibt sich folgende Struktur für diese Arbeit: In Kapitel 2 werden die theoretischen Grundlagen vermittelt. Dabei wird sowohl die Problemstellung mit MLE und GPM erläutert (Kap. 2.1) als auch der Optimierungsalgorithmus PSO selbst vorgestellt (Kap. 2.2). Damit wird die Basis für das Praxiskapitel (3) gelegt: Hier wird in Kapitel 3.1 zunächst die Implementierung beschrieben, die den nachfolgenden empirischen Untersuchungen zugrunde liegt. Anschließend werden erste Untersuchungen des Problems durchgeführt (Kap. 3.2). Dabei wird PSO zur besseren Einschätzung der Ergebnisse mit den Optimierungsalgorithmen *Differential Evolution* (DE) und *Random Search* (RS) als Referenz verglichen. In Kapitel 3.3 werden weitere Untersuchungen

durchgeführt, bei denen verschiedene Einflussgrößen des Experiments systematisch variiert werden, um die Performance der Optimierer für andere Optimierungsprobleme zu testen und zu vergleichen. In Kapitel 3.4 werden dann Einfluss und Relevanz der Hyperparameter von PSO in entsprechenden Experimenten untersucht. Um die Arbeit abzuschließen, werden in Kapitel 4 die wichtigsten Ergebnisse zusammengefasst und daraus ein Fazit zur Funktionsweise und Eignung des Optimierungsalgorithmus PSO gezogen.

2 Theoretische Grundlagen

Zum besseren Verständnis der Experimente im Praxisteil dieser Arbeit ist es notwendig, zunächst die grundlegenden theoretischen Konzepte dahinter zu erläutern. Die Ausführungen im ersten Abschnitt (2.1) zu GPM und zur MLE konkretisieren die Problemstellung der Experimente. Weiter ist es erforderlich, den Optimierungsalgorithmus PSO vorzustellen, der Untersuchungsgegenstand dieser Arbeit ist. Dies geschieht im zweiten Unterabschnitt (2.2).

2.1 Gaußsche Prozessmodelle und Maximum Likelihood Estimation

Im überwachten Lernen werden einige Inputs x_i und einige Outputs y_i beobachtet. Dabei wird davon ausgegangen, dass eine Funktion existiert, für die gilt (vgl. Murphy, 2012, S.515):

$$y_i = f(x_i) \quad (1)$$

Ein guter Ansatz wäre nun eine Verteilung von Funktionen abzuleiten und diese auf neue Inputs anzuwenden, um Vorhersagen zu treffen. Dies kann mithilfe von GPM geschehen. Es scheint zunächst schwierig, eine Verteilung von Funktionen darzustellen. Allerdings ist dafür lediglich die Definition einer Verteilung über die Funktionswerte einiger Punkte x_1, \dots, x_n notwendig. Beim GPM wird, wie der Name bereits vermuten lässt, angenommen, dass diese Verteilung eine Gauß-Verteilung mit einem Durchschnitt $\mu(x)$ und einer Kovarianz $\sum(x)$ ist. Dabei berechnet sich die Kovarianz $\sum(x)$ wie folgt (vgl. Murphy, 2012, S.515):

$$\sum_{ij} = K(x_i, x_j) \quad (2)$$

K ist hier eine positive eindeutige Kernel-Funktion. Die grundlegende Idee ist dabei, dass wenn der Kernel x_i und x_j als ähnlich erachtet, es wahrscheinlich ist, dass die Funktionswerte von x_i und x_j ebenfalls ähnlich sind. GPM können zur Lösung verschiedener Probleme wie Klassifizierung oder Regression eingesetzt werden (vgl. Murphy, 2012, S.515). Im Folgenden wird allerdings ausschließlich auf GPM zur Lösung von Regressions-Problemen eingegangen (vgl. Vidya und Hari, 2022, S.3):

$$f(x) = GPM(m(x), K(x_i, x_j)) \quad (3)$$

Dabei ist $f(x)$ die Regressionsfunktion. Das GPM besteht aus einer Durchschnittsfunktion $m(x)$ und einer Kovarianzfunktion $K(x_i, x_j)$ (Kernel). Wenn das Modell verwendet wird, um einen Wert $f(x)$ für einen Wert x vorherzusagen, der bereits bekannt ist, sollte das Modell als Interpolator dienen. Das heißt, das Modell sollte in der Lage sein, diesen

Wert mit Sicherheit vorherzusagen. Dies ist allerdings nur möglich wenn angenommen werden kann, dass die Daten keinerlei Rauschen¹ enthalten. Dies ist in der Realität selten der Fall. Im Fall von Daten, die Rauschen enthalten, wird angenommen, dass jeder Wert der Zielvariable einen bestimmten Rauschwert enthält (vgl. Vidya und Hari, 2022, S.3):

$$y = f(x) + \epsilon \quad (4)$$

Dabei ist y die Vorhersage (bzw. bei den Trainingsdaten die Beobachtung), $f(x)$ die Regressionsfunktion und ϵ die zugehörige Rauschkomponente. Der wichtigste Faktor für die Performance des Modells ist der Kernel. Im Folgenden wird daher der RBF-Kernel (auch Squared-Exponential oder Gauss-Kernel) und seine Parameter näher betrachtet, da dieser von hoher Relevanz für diese Arbeit ist. Die Formel dieses Kernels lautet wie folgt (vgl. Duvenaud, 2014, S.9):

$$K(x_i, x_j) = \exp\left(\frac{-||x_i - x_j||_2^2}{2l^2}\right) \quad (5)$$

Der Parameter l steuert hier den horizontalen Scale (vgl. Duvenaud, 2014, S.9). Dieser RBF-Kernel ist isotrop, das bedeutet, dass nur eine Dimension vorhanden ist. Berücksichtigt man mehrere Dimensionen, so stellt sich der Sachverhalt wie folgt dar (anisotropes Kernel) (vgl. Aiolfi und Donini, 2014, S.518):

$$K_{\mathbf{l}}(x_i, x_j) = \prod_{r=1}^m \exp\left(\frac{-||x_i - x_j||_2^2}{2l_r^2}\right) \quad (6)$$

Dabei enthält der Vektor \mathbf{l} die horizontalen Scales der verschiedenen Dimensionen (vgl. Aiolfi und Donini, 2014, S.518). Ist ein horizontaler Scale für eine bestimmte Dimension sehr groß, dann heißt das, dass diese Dimension nahezu irrelevant ist. Ist der horizontale Scale einer Dimension x kleiner als der einer Dimension y , so wird sich die Funktion entlang der Dimension x schneller ändern als in Richtung y (vgl. Murphy, 2012, S.520).

Um die besten Kernel-Parameter zu finden, könnten nun viele verschiedene Kombinationen ausprobiert und über eine Loss-Funktion validiert werden. Dies dauert allerdings in der Regel lange. Üblicherweise wird daher das Verfahren der Maximum Likelihood Estimation (MLE) verwendet, um geeignete Parameter für den Kovarianzkern zu finden (vgl. Karvonen und Oates, 2022, S. 1). Diesem Verfahren liegt der Gedanke zugrunde, dass diejenigen Parameter-Werte gewählt werden sollen, die zu dem Modell führen (hier ein GPM), welches am wahrscheinlichsten die Daten erzeugt hat. Zunächst wird hierzu eine entsprechende Likelihood-Funktion aufgestellt. Es soll nun derjenige

¹Rauschen: Daten, die nicht dabei helfen Zusammenhänge zur Zielvariablen festzustellen, sondern das Finden dieser Zusammenhänge erschweren.

Parameter-Vektor gefunden werden, für den diese Likelihood-Funktion maximal wird. Zur Vereinfachung der Berechnung wird üblicherweise die Log-Likelihood-Funktion maximiert². In der Praxis ist es jedoch meist nicht möglich, eine Lösung in analytischer Form für die MLE-Schätzung zu finden. In solchen Fällen muss die MLE-Schätzung mit numerischen Optimierungsverfahren durchgeführt werden (vgl. Myung, 2003).

2.2 Particle Swarm Optimization

Optimierungsprobleme beschäftigen sich damit, optimale Parameter für eine Funktion zu finden, sodass diese minimiert bzw. maximiert wird (vgl. Boyd und Vandenberghe, 2004). Damit handelt es sich bei der Suche nach dem Parameter-Vektor \mathbf{l} , für den die Likelihood-Funktion maximal wird, stellt somit ein Optimierungsproblem dar. Oftmals sind diese Optimierungsprobleme sehr komplex oder unmöglich analytisch zu lösen. Wie oben skizziert, ist dies häufig auch für die MLE der Fall. Aus diesem Grund wird in dieser Arbeit das numerische Optimierungsverfahren der Particle Swarm Optimization zur Optimierung der Likelihood verwendet.

PSO gehört zu den naturanalogen Optimierungsverfahren und bildet das Schwarmverhalten von Vögeln nach (vgl. Reynolds, 1987). Er wird vor allem für kontinuierliche nicht-lineare Probleme verwendet (vgl. Kennedy und Eberhart, 1995). Bei der Implementierung von PSO werden eine Vielzahl von Entitäten (“Partikel”) zufällig im Suchbereich der Zielfunktion (hier: Likelihood-Funktion) platziert. Jede Entität besteht aus drei D -dimensionalen Vektoren, wobei D die Anzahl der Dimensionen des Suchraumes entspricht. Diese Vektoren sind die aktuelle Position des Partikels (\vec{P}_i), die vorherige beste Position (\vec{P}_{best}) und die Geschwindigkeit (\vec{V}_i). In jeder Iteration wird die aktuelle Position (\vec{P}_i) als potenzielle Lösung der Zielfunktion betrachtet. Ist diese Position besser als die bisher beste gefundene Position des Partikels, so wird diese in die Variable \vec{P}_{best} gespeichert. Diese Variable kann später für den Vergleich der Iterationen verwendet werden. Die Entitäten bewegen sich mit der Geschwindigkeit \vec{V}_i multipliziert mit einer Störung (*Inertia*), die auf die aktuelle Position \vec{P}_i addiert werden. Daraus ergibt sich eine neue Richtung für das Partikel. Die Geschwindigkeit kann zusätzlich im Laufe der Optimierung angepasst werden und ist vergleichbar mit einer Schrittweite (vgl. Poli u. a., 2007).

Ein einzelnes solches Individuum bringt allerdings noch keine großen Erfolge. Statt-

²Die in dieser Arbeit verwendeten Implementierungen beziehen sich auf die *log marginal likelihood*. Zur besseren Lesbarkeit wird jedoch lediglich der Begriff “Likelihood” verwendet, da die Transformationen lediglich rechentechnische Vereinfachungen darstellen.

dessen wird der Effekt der Schwarmintelligenz benötigt, um die optimale Lösung zu finden (vgl. Bonabeau u. a., 1999). Der wahre Erfolg von PSO resultiert aus der Interaktion der einzelnen Entitäten. Dies geschieht über eine Art Kommunikationsstruktur, Topologie oder soziales Netzwerk. Eine Topologie besteht aus bidirektionalen Enden, die die Entitäten paarweise verbindet (vgl. Claus u. a., 2003). Daraus folgt, dass jedes Partikel mit anderen Partikeln innerhalb des Schwarmes kommuniziert. Durch die Kommunikation werden die Entitäten innerhalb einer Nachbarschaft vom besten Punkt der Nachbarschaft beeinflusst. Der Vektor mit der besten Position der Nachbarschaft wird als \vec{G}_{best} bezeichnet. Innerhalb der Optimierung wird dann die Geschwindigkeit jedes einzelnen Partikels so angepasst, dass der Schwarm sich stochastisch um die beste Position es Individuum (\vec{P}_{best}) und der des Schwarms (\vec{G}_{best}) bewegt.

Der ursprüngliche Algorithmus von *Kennedy und Eberhart (1995)* kann folgendermaßen zusammengefasst werden:

1. Bestimme eine Menge von Partikeln, die zufällig im Suchraum verteilt werden.
2. Iteriere durch eine Anzahl an Iterationen.
3. Evaluiere die Zielfunktion und vergleiche, ob die neue Position die vorherige Position übertrifft.
4. Berechne die Geschwindigkeit für jedes Partikel im Schwarm.
5. Überarbeite für jedes Partikel die Position mit der vorheriger Position und der Geschwindigkeit.
6. Springe zu Schritt 2 und wiederhole dies solange, bis das Optimum gefunden wurde (konvergiert).

Für ein besseres Verständnis der Methode von PSO, wird als nächstes genauer auf die Mathematik dahinter eingegangen. Wie zuvor skizziert, sucht der Algorithmus das Optimum, indem sich jedes Partikel des Schwarmes in die Richtung seiner vorherigen besten Position (\vec{P}_{best}) und der globalen besten Position des Schwarms (\vec{G}_{best}) bewegt (vgl. Kennedy und Eberhart, 1995). Daraus ergeben sich die folgenden beiden Position, \vec{P}_{best} und \vec{G}_{best} ,

$$\vec{P}_{best} = \arg \min_{k=1,\dots,t} [(f(\vec{P}_i(k))], \quad i \in 1, 2, \dots, N, \quad (7)$$

$$\vec{G}_{best} = \arg \min_{\substack{i=1,\dots,N \\ k=1,\dots,t}} [(f(\vec{P}_i(k))], \quad (8)$$

wobei i den Partikelindex, N die Gesamtzahl der Partikel, t die aktuelle Iterationsanzahl und f die Zielfunktion darstellen. Die Position \vec{P}_i und die Geschwindigkeit \vec{V}_i der Partikel werden folgendermaßen angepasst,

$$\vec{V}_i(t+1) = w\vec{V}_i(t) + c_1r_1(\text{pbest}(i,t) - \vec{P}_i(t)) + c_2r_2(\text{gbest}(t) - \vec{P}_i(t)), \quad (9)$$

$$\vec{P}_i(t+1) = \vec{P}_i(t) + \vec{V}_i(t+1), \quad (10)$$

wobei w das Gewicht der Trägheit (*Inertia*) für die Balance zwischen *Exploration*³ und *Exploitation*⁴ der Partikel darstellt (vgl. Eberhart und Shi, 2001). Die Parameter r_1 , r_2 im Bereich $[0, 1]$ und die Konstanten c_1 , c_2 bilden die Beschleunigungskoeffizienten für die Partikel (vgl. Yudong Zhang, 2015).

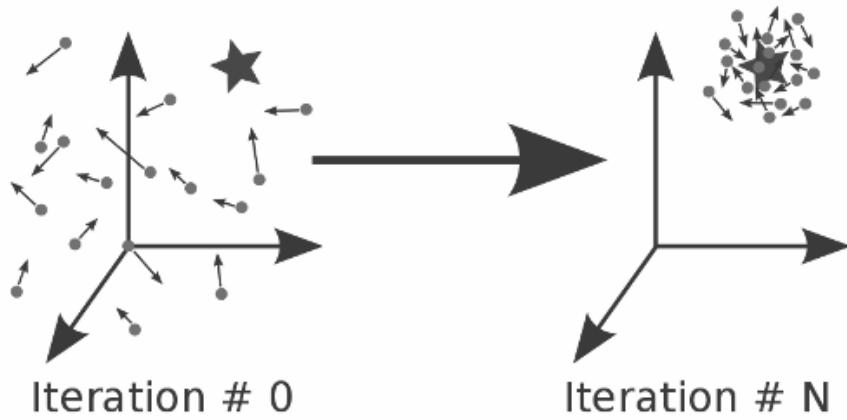


Abb. 2.1: Particle Swarm Optimierung im drei dimensional Raum
Pagmo (2022)

Innerhalb einer Iteration wird anhand dieser Formeln die Position jedes Partikel geändert, bis das lokale oder globale Optimum gefunden ist. In Grafik 2.1 sind Anfangszustand und Ziel von PSO visuell dargestellt. Hier sucht der Schwarm das Optimum und kann dieses schlussendlich approximativ bestimmen. Diese Methode besitzt eine Vielzahl an Erweiterungen, deren Beschreibung jedoch den Rahmen dieser Arbeit sprengen würde. Für weitere Informationen sei daher auf Clerc (2006) verwiesen.

³Bei der Exploration möchte man das bestehende Wissen mit neuem wissen erweitern. Hierbei steht die Innovation im Vordergrund (vgl. O'Reilly und Tushman, 2021).

⁴Unter Exploitation versteht man das bestehende Wissen auszuschöpfen oder zu optimieren (vgl. O'Reilly und Tushman, 2021). Exploitation und Exploration stehen aufgrund ihrer Rahmenbedingungen in einem stetigen Konflikt (vgl. Olivian, 2019)

3 Empirische Untersuchungen

Nachdem die notwendigen theoretischen Grundlagen vermittelt wurden, kann nun die empirische Untersuchung des Optimierungsalgorithmus PSO beginnen. Wie zuvor bereits skizziert, soll die Likelihood-Funktion eines GPM optimiert werden. Spezifischer sollen dabei die Kernel-Parameter θ^5 des anisotropen RBF-Kernels (Formel (6)) so gewählt werden, dass die Likelihood maximiert wird. Anhand dieses Optimierungsproblems wird die Funktionsweise und Eignung von PSO untersucht und dabei mit DE und RS als Referenz-Algorithmen verglichen. Wie **Abb. 3.1**⁶ zeigt, gleichen sich diese Algorithmen in vielen Eigenschaften. Zudem handelt sich es bei RS um einen sehr einfachen Optimierungsalgorithmus, während DE in seiner Funktionsweise komplexer ist. Damit stellen sie gemeinsam ideale Referenzen für den Vergleich dar.

Methode	Global	Gradientenbasiert	Nebenbedingungen	Ziele	Linear	Ansatz	Genauigkeit	Deterministisch	Variablen
DE	✓	✗	(Grenzen)	1	✗	numerisch	approximativ	✗	\mathbb{R}^n
RS	✓	✗	(Grenzen)	1	✗	numerisch	approximativ	✗	\mathbb{R}^n
PSO	✓	✗	(Grenzen)	1	✗	numerisch	approximativ	✗	\mathbb{R}^n

Abb. 3.1: Vergleichstabelle der Algorithmen

3.1 Implementierung

Nachfolgend wird nun kurz die Implementierung beschrieben. Zweck dieses Abschnittes ist es, die Reproduzierbarkeit der Untersuchungsergebnisse aus den anschließenden Kapiteln zu gewährleisten. Es sei darauf hingewiesen, dass alle Experimente in der Programmiersprache Python durchgeführt wurden. Der vollständige Quellcode zu den Untersuchungen und Visualisierungen ist auf GitHub⁷ zu finden.

```

1 def generate_sample(n, n_dims, lower, upper, target_func, noise_scale=0,
2                     random_state=42):
3     rs = np.random.RandomState(random_state)
4     X = rs.rand(n_dims, n)
5     for i in range(n_dims):
6         X[i] = X[i]*(upper[i]-lower[i]) + lower[i]
7     y = target_func(X)
8     y += rs.normal(0, noise_scale, size=y.shape)
9
10    return (X,y)

```

Listing 3.1: Funktion zur Generierung von Daten für die Untersuchungen

⁵Die Parameter θ_r entsprechen hierbei den *length-scale* l_r des Kernels. In den verwendeten Implementierungen entspricht θ nicht direkt l , es findet eine Umrechnung statt. Für den genauen Zusammenhang sei auf die Dokumentation von scikit-learn (Scikit-Learn-Docs1, 2022) und den entsprechenden Source-Code verwiesen.

⁶Die Eigenschaften beziehen sich auf die verwendeten Implementierungen (Details s.u.)

⁷<https://github.com/PascalKnoll/Investigation-of-Particle-Swarm-Optimization>(17.8.2022)

Zur Generierung von Daten-Samples für die Experimente wird die in **Listing 3.1** gezeigte Funktion verwendet. Damit können Datensätze mit beliebig vielen Dimensionen und Datenpunkten generiert werden. Die Datenpunkte werden dabei zufällig gesampt, für die Reproduzierbarkeit wird ein Random-Seed gesetzt. Die Daten sind gemäß einer zu spezifizierenden Zielfunktion verteilt, so dass im Laufe der Untersuchungen verschiedene Groundtruth-Funktionen betrachtet werden können. Um den Einfluss von Rauschen (Noise) untersuchen zu können, wird zusätzlich ein Störwert addiert, der normalverteilt mit Mittelwert null und spezifizierbarer Standardabweichung ist. Über Angabe der Standardabweichung kann so die Stärke des Rauschens reguliert werden. Auf diesen Daten soll nun ein GPM trainiert werden. Hierzu wird der `GaussianProcessRegressor` von `scikit-learn(v.1.0.2)` verwendet (vgl. Scikit-Learn-Docs1, 2022). Als Kernel wird `RBF` der selben Bibliothek gewählt (vgl. Scikit-Learn-Docs2, 2022). Diesem wird als initiale `length_scale` ein Tupel aus Einen übergeben, dessen Länge der Anzahl der Dimensionen der Daten entspricht. Dadurch wird die anisotrope Variante des Kernels verwendet.

```

1  class RandomOptim(Optimizer):
2      def __init__(self, n_iters, ...):
3          super().__init__()
4          self.n_iters = n_iters
5          ...
6
7      def optimize(self, obj_func, init_theta, bounds):
8          theta_opt = [] # optimal thetas
9          func_max = float("inf") # best found log likelihood, initial:
10         inf
11         func_current = 0 # log likelihood of current iter
12         thetas = [] # thetas of current iter
13         rs = np.random.RandomState(42)
14
15         for _ in range(0, self.n_iters):
16             thetas = []
17             for _ in range(0, init_theta.shape[0]):
18                 thetas.append(rs.uniform(bounds[0][0], bounds[0][1]))
19
20             func_current = obj_func(thetas)[0]
21
22             if func_current < func_max:
23                 func_max = func_current
24                 theta_opt = thetas
25
26             ...
27
28         return theta_opt, func_max

```

Listing 3.2: Random Search Algorithmus (eigene Implementierung)

Die Kernel-Parameter θ_r (`length_scale`) werden mit verschiedenen Optmierungsalgorithmen optimiert. Im Vordergrund dieser Arbeit steht PSO, für den die Implementierung

GlobalBestPSO der Bibliothek `pyswarms` (v.1.3.0) eingesetzt wird (vgl. PySwarms-Docs, 2022). Um diesen mit dem GPM von Scikit-Learn nutzen zu können, müssen die Schnittstellen manuell angepasst werden. Dasselbe muss für den Referenz-Algorithmus DE getan werden. Hier wird die `scipy` (v.1.7.3) Funktion `differential_evolution` benutzt (vgl. SciPy-Docs, 2022). Für RS, den zweiten Referenz-Algorithmus, wird eine eigene, einfache Implementierung genutzt (siehe **Listing 3.2**). Die Superklasse `Optimizer`, von der neben RS auch die auf die Schnittstellen angepassten Implementierungen der anderen Optimierer erben, stellt dabei Funktionen zur Verfügung, die später eine Visualisierung des Optimierungsvorgangs erlauben. Mit den in diesem Abschnitt skizzierten Implementierungen werden nun verschiedene Untersuchungen durchgeführt.

3.2 Erste Untersuchungen und Vergleich zu anderen Optimierungsalgorithmen

Zunächst soll PSO anhand eines einfachen Experiments untersucht werden. Hierzu werden Daten mit der *Rastrigin*-Funktion generiert (*Groundtruth*, siehe **Abb. 3.2**). Davon werden 70 Datenpunkte zufällig gesampliert, die zum Training eines GPM verwendet werden. Zur Optimierung wird eine PSO-Instanz mit 10 Iterationen und 10 Partikeln genutzt. Als Referenzalgorithmus wird DE mit einer Populationsgröße und einer maximalen Iterationszahl von jeweils 10 eingesetzt, ebenso wie RS mit maximal 100 Iterationen. Damit ist für alle Optimierer die maximale Anzahl an Funktionsauswertungen auf dieselbe Zahl begrenzt.

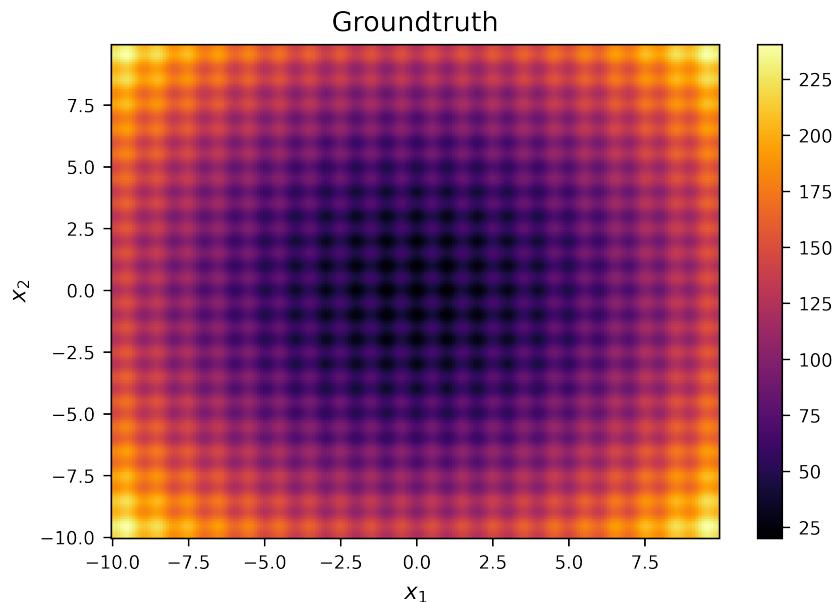


Abb. 3.2: Groundtruth Funktion

Die Vorhersagen der aus den Optimierungen resultierenden GPM sind in **Abb. 3.3**

dargestellt. Ein erster Blick zeigt bereits, dass kein Modell die Groundtruth-Funktion abbilden kann. Das resultierende Modell für die Güte der Optimierer heranzuziehen ist jedoch wenig sinnvoll, da hier noch andere Faktoren eine Rolle spielen (Anzahl Datenpunkte, Eignung von GPM für das Problem, ...). Außerdem ist eine einmalige Ausführung nicht repräsentativ. Aus diesen Gründen werden nachfolgend die Likelihood-Werte aus insgesamt 20 Neuausführungen der Optimierer zu deren Bewertung betrachtet.

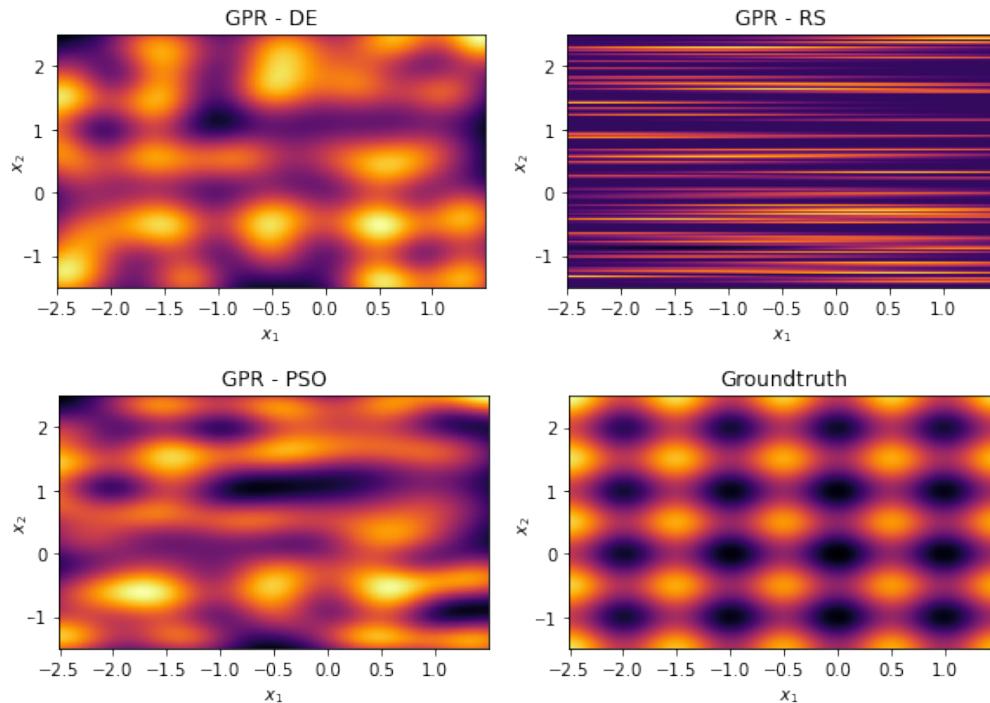


Abb. 3.3: Vorhersagen der mit verschiedenen Optimierern trainierten GPM

Bevor auf die Ergebnisse eingegangen wird, soll noch ein Blick auf die 3D-Likelihood-Landschaft in **Abb. 3.4** geworfen werden. In den Suchgrenzen gibt es nur ein Maximum (rechts), wodurch sich ein unimodales Optmierungsproblem ergibt. Für die nachfolgenden Untersuchungen wird aus Komplexitätsgründen auf 2D-Plots zurückgegriffen.

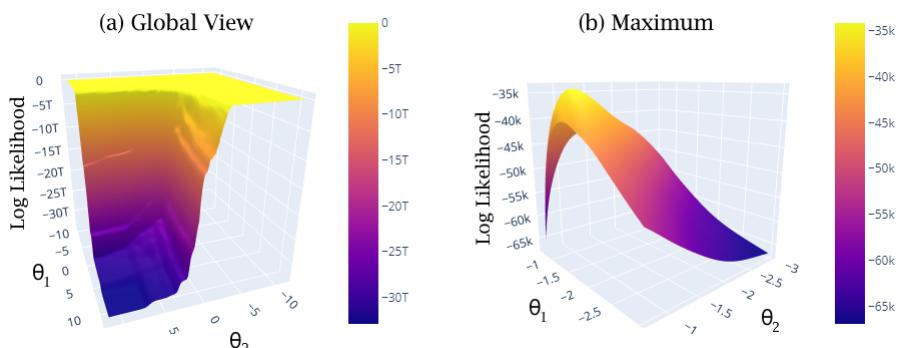


Abb. 3.4: Globale Sicht der Likelihood-Landschaft in den Suchgrenzen (links) und Zoom auf das Maximum (rechts)

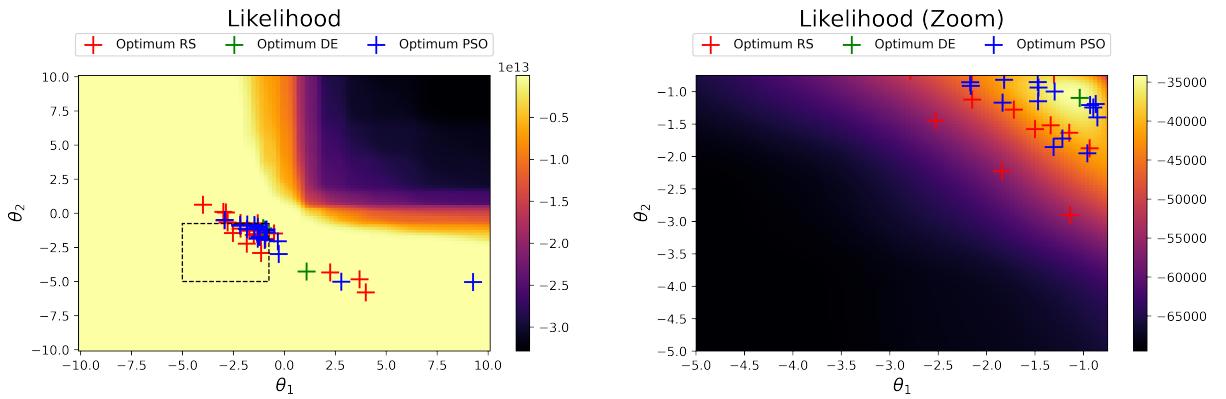


Abb. 3.5: Von den Optimierern gefundene Optima der Likelihood

Die von den Optimierern gefundenen Optima der Likelihood aus den 20 Durchläufen sind in **Abb. 3.5** gezeigt. Es handelt sich um die log-transformierte Likelihood, bei der ebenfalls möglichst hohe (maximale) Werte gefunden werden sollen. Hier ist nun gut zu erkennen, dass DE bis auf eine Ausnahme das Optimum sehr genau findet. RS hingegen ist stärker gestreut und immer deutlich weiter entfernt vom Optimum. PSO ist nun nicht ganz so stark gestreut wie RS und in der Regel näher am Optimum. Jedoch gibt es auch starke Ausreißer. Zudem wird das Optimum nie so genau und insbesondere nicht so zuverlässig gefunden wie mit DE.

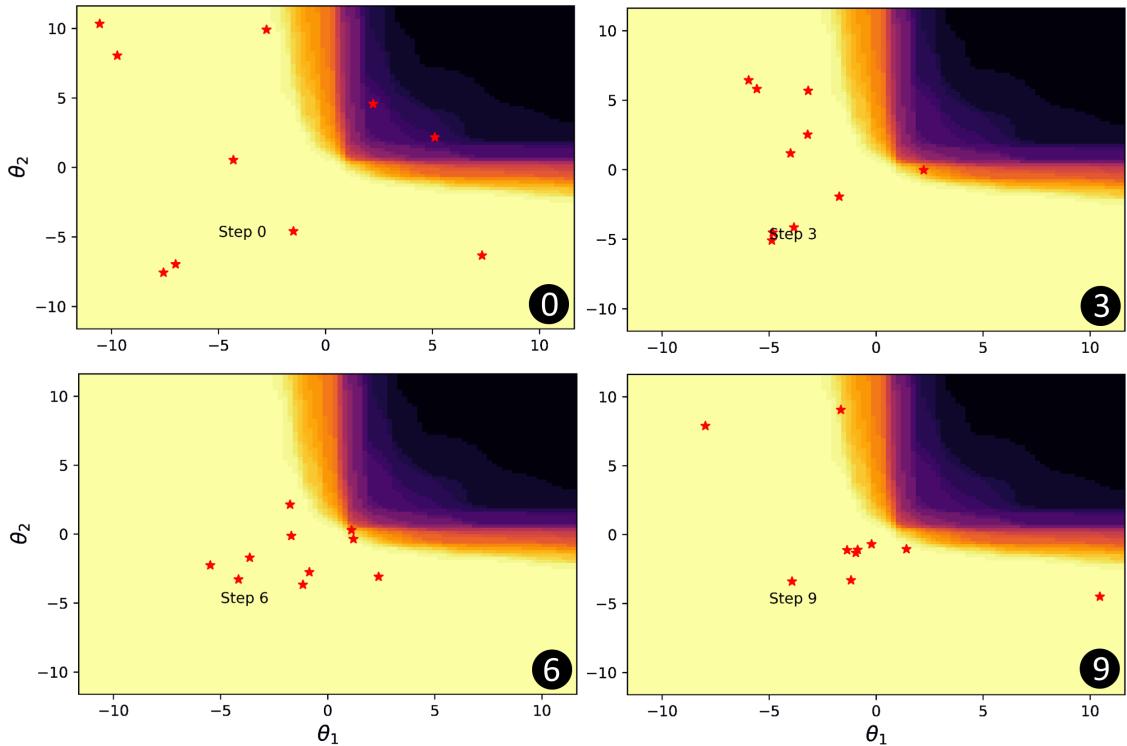


Abb. 3.6: Ausgewählte Schritte der Optimierung mit PSO

Das Abschneiden von PSO kann hier verschiedene Gründe haben. Das Problem bzw. die Likelihood-Funktion könnte für den Optimierer schwierig sein oder die Hyperpara-

meter könnten falsch abgestimmt sein. Für letzteren Punkt könnten beispielsweise die Anzahl an Iterationen zu gering sein. Um dies aufzuklären, wird ein genauerer Blick auf die einzelnen Optimierungsschritte von PSO geworfen. Ausgewählte Schritte sind in **Abb. 3.6** dargestellt⁸. Es ist zu erkennen, dass sich der Schwarm (rote Sterne) zwar in Richtung des Optimums bewegt, Schritt 9 aufgrund von Ausreißern aber auf den ersten Blick nicht unbedingt eine große Verbesserung gegenüber Schritt 6 darstellt. Trotzdem sind in Schritt 9 einige Partikel nahe am Optimum, weshalb davon auszugehen ist, dass mehr Iterationen das Ergebnis noch verbessert hätten. Weitere Untersuchungen sollen nachfolgend mehr Details enthüllen.

3.3 Weiterführende Experimente

Aus oben genannten Gründen wird die Anzahl der Iterationen für PSO für die nachfolgenden Untersuchungen auf 100 erhöht (sowie entsprechende Anpassungen für DE und RS durchgeführt). In den Unterabschnitten dieses Kapitels werden nun verschiedene Einstellungen des oben skizzierten Experiments variiert. All diese Änderungen führen zu einer Veränderung der zu optimierenden Likelihood-Funktion. Damit werden die Optimierer auf neue Probleme angewandt. Im letzten Unterabschnitt soll dann eine geeignete Auswertung der Optimierer auf alle sich aus den einzelnen Experimenten ergebenden Problemen erfolgen.

3.3.1 Variation der Anzahl der Trainingsdatenpunkten

Um andere Likelihood-Funktionen zu erzeugen, kann zunächst die Anzahl an Datenpunkten variiert werden, die für das Training des GPM verwendet werden. In **Abb. 3.7** sind die Ergebnisse einer solchen Untersuchung visualisiert. In der Grafik ist die maximale gefundene Likelihood der Optimierer über die Stichprobengröße n des Trainingsdatensatzes aufgetragen. Um statistisch bedeutsame Aussagen tätigen zu können, sind die Ergebnisse aus 20 Versuchswiederholungen dargestellt. Neben den Mittelwerten ist auch das 95%-Konfidenzintervall der Ergebnisse aus den 20 Versuchen gezeigt.⁹

Zu erkennen ist, dass die gefundene maximale Likelihood von DE und PSO immer nahezu identisch ist, wobei kaum Varianz zwischen den Versuchswiederholungen zu sehen ist. Die Abweichungen befinden sich in der Größenordnung zwischen 0,1 und 1. Im Gegensatz hierzu steht RS, der generell eine schlechtere Likelihood aufweist und

⁸Eine Animation der Optimierung als GIF ist auf dem zugehörigen GitHub-Repository <https://github.com/PascalKnoll/Investigation-of-Particle-Swarm-Optimization> zu finden.

⁹Es sei darauf hingewiesen, dass für die Erstellung der Grafik (sowie für alle nachfolgenden Grafiken des gleichen Typs) vereinfachend angenommen wurde, dass das Konfidenzintervall symmetrisch um den Mittelwert verteilt ist. Es soll hier lediglich die Varianz der gefundenen Optima zeigen.

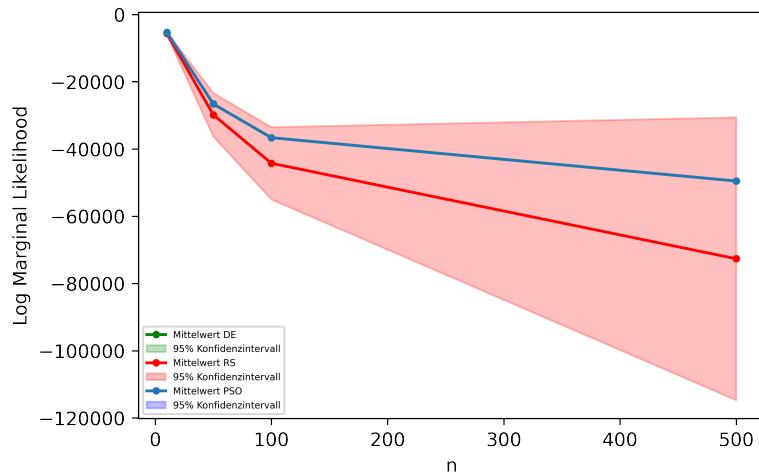


Abb. 3.7: Einfluss der Sample-Größe (n) auf die maximale gefundene Likelihood

dessen Varianz deutlich höher ist. Das regelmäßige und verlässliche Finden der Optima ist eine wichtige Qualität von Optimierern und zeigt eine solide Performance der jeweiligen Algorithmen. Die Grafik zeigt darüber hinaus, dass die maximale Likelihood mit zunehmender Anzahl an Trainingsdatenpunkten abnimmt. Dies ist zu erwarten, da es mit zunehmender Sample-Größe für die GPM schwieriger wird, die Daten abzubilden, woraus eine geringere Likelihood resultiert. Die steigende Streuung von RS mit zunehmender Sample-Größe könnte auf eine höhere Komplexität der Likelihood-Landschaften hindeuten. Die zuverlässige Performance von PSO (und DE) ist insbesondere vor diesem Hintergrund als positiv zu bewerten. Um an dieser Stelle den Bezug zum Ausgangsproblem herzustellen, sind in **Abb. 3.8** die Vorhersagen der mit unterschiedlicher Sample-Größe trainierten GPM dargestellt. Hier zeigt sich, dass eine zunehmende Anzahl an Datenpunkten die Qualität der Vorhersagen deutlich verbessert. Damit kann, in Kombination mit mehr Datenpunkten zum Training des GPM, auch das Ausgangsproblem mit PSO als Optimierer zufriedenstellend gelöst werden, was in dieser Arbeit jedoch eine untergeordnete Rolle spielt.

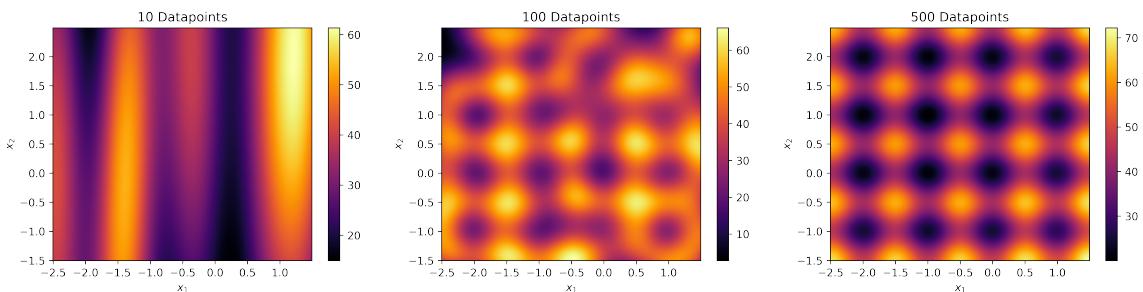


Abb. 3.8: Vorhersagen des GPM in Abhängigkeit der Anzahl an Trainingsdaten

3.3.2 Variation der Dimensionsanzahl

Nachfolgend wird nun die Likelihood-Landschaft durch Variation der Dimensionen der Trainingsdaten verändert. Die Symmetrie der Groundtruth-Funktion erlaubt dies, ohne größere Anpassungen vornehmen zu müssen. Da die anisotrope Variante des RBF-Kernels verwendet wird, verändert sich analog zur Dimensionsanzahl der Trainingsdaten auch die Dimensionalität des Suchraums für die Optimierer.

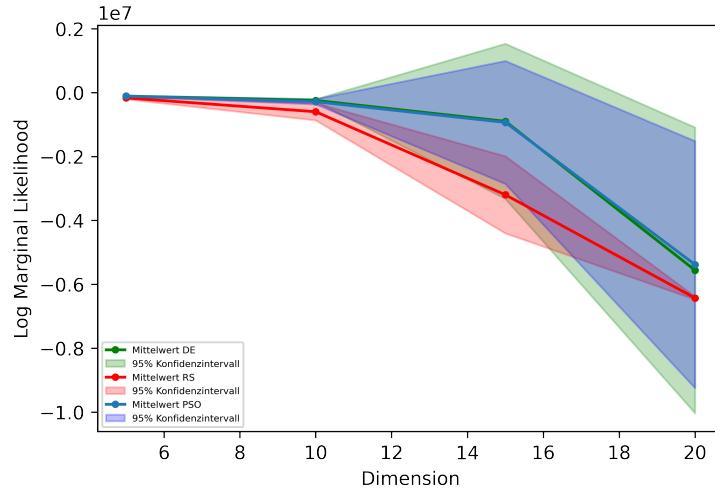


Abb. 3.9: Maximale gefundene Likelihood in Abhängigkeit der Dimensionalität

Pro Dimension wird 20 mal getestet, **Abb. 3.9**¹⁰ zeigt die entsprechenden Mittelwerte und das 95%-Konfidenzintervall für die verwendeten Optimierer. Ein globaler Blick auf die Grafik verrät, dass alle Optimierer mit zunehmender Dimensionsanzahl Schwierigkeiten bekommen. Dafür ist nicht unbedingt die geringere maximale Likelihood ausschlaggebend, sondern vielmehr die Breite des Konfidenzintervalls. Während die Werte für PSO und DE bei geringerer Dimensionszahl kaum variieren, zeigt sich für mehr Dimensionen eine hohe Streuung. Daraus lässt sich schließen, dass das Optimum, wenn überhaupt, hier nur unzuverlässig gefunden wird. RS liefert hingegen schon früh deutlich schlechtere Ergebnisse als die anderen beiden Optimierer.

In **Abb. 3.10** sind die Ergebnisse für 5, 15 und 20 Dimensionen nochmals in Boxplots aufbereitet. Diese Dimensionen wurden ausgewählt, da sich hier einige interessante Sachverhalte besonders gut zeigen. In den Boxplots ist deutlich zu erkennen, dass der einfache RS-Algorithmus in jedem Fall schlechtere Likelihood-Werte als Maximum findet. Der Vergleich von DE und PSO weist je nach Dimensionszahl Unterschiede auf: Für 5 Dimensionen sind sich die Medianwerte sehr ähnlich, PSO streut jedoch weniger.

¹⁰Bezüglich des Konfidenzintervalls sei darauf hingewiesen, dass tatsächlich keine Ausreißer der Likelihood-Werte nach oben gibt, wie beispielsweise das Konfidenzintervall für 15 Dimensionen suggeriert. Vielmehr resultiert diese Darstellung aus der Symmetrie-Annahme für das Konfidenzintervall (s.o.)

Bei 15 Dimensionen bietet sich das umgekehrte Bild: Hier streut nun DE weniger, zudem findet er generell höhere maximale Likelihood-Werte als PSO. Für 20 Dimensionen weisen nun alle Optimierer eine ungenügende Performance auf. Einzelne Ausreißer nach oben indizieren, dass es ein Optimum in dieser Größenordnung gibt. Bei den meisten Durchläufen finden die Optimierer jedoch nur deutlich geringere Likelihood-Werte, wie die Mediane am unteren Ende der Grafik zeigen. Dass die Optimierer mit zunehmen-

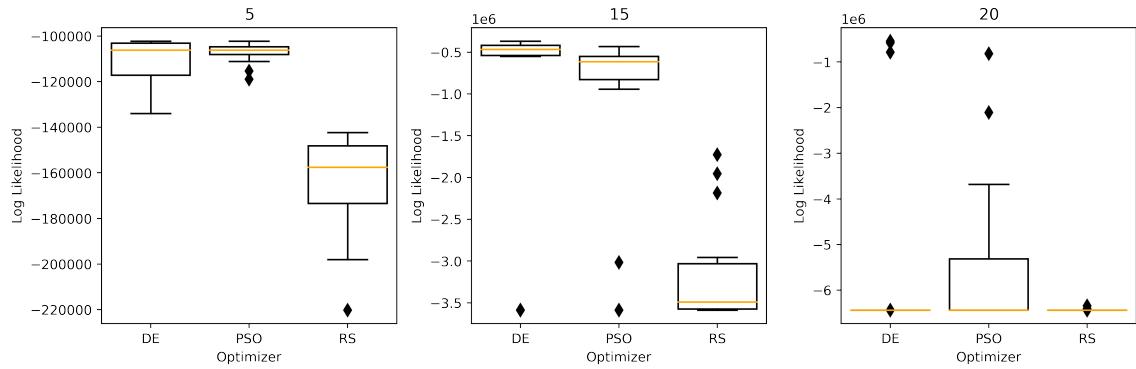


Abb. 3.10: Boxplots für ausgewählte Dimensionszahlen

der Dimensionsanzahl Schwierigkeiten bekommen, die optimale Likelihood zu finden, lässt sich auf die steigende Komplexität der Likelihood mit zunehmender Dimensionsanzahl zurückführen. Außerdem führt die Erhöhung der Dimensionsanzahl zu einer Vergrößerung des Suchraumes. Es scheint also naheliegend, dass dann auch die Anzahl an Funktionsauswertungen erhöht werden müsste (z.B. Partikel oder Iterationen). Abschließend lässt sich sagen, dass aus den Untersuchungen dieses Abschnitts PSO zwar besser als der simple RS hervorgeht, insgesamt jedoch weder signifikant besser noch schlechter als der komplexere DE ist.

3.3.3 Verwendung verschiedener Groundtruth-Funktionen

Ein weiterer Parameter des Experiments, der variiert werden kann, ist die den Trainingsdaten zugrundeliegende Funktion (*Groundtruth*). In diesem Abschnitt werden drei neue Groundtruth-Funktionen verwendet. Diese sind in **Abb. 3.11** abgebildet. Sie alle zielen darauf ab, eine andere, möglichst komplexere Likelihood-Landschaft zu produzieren, deren Maximum durch die Optimierer gefunden werden soll.

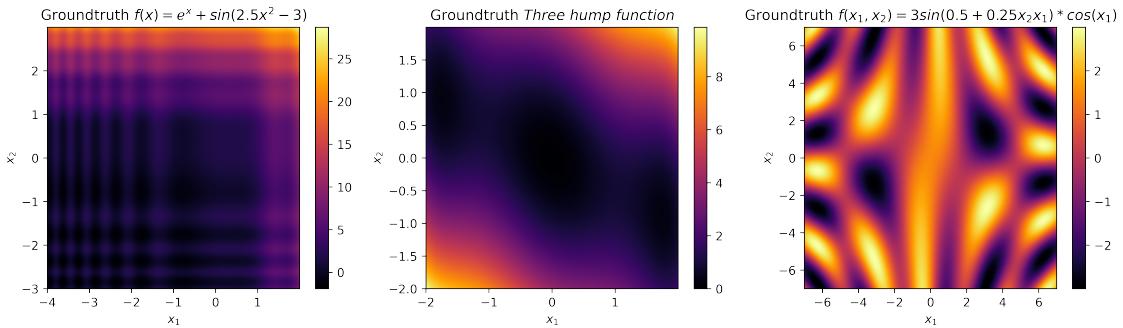


Abb. 3.11: Verwendete Groundtruth-Funktionen

Wieder werden für jede Groundtruth 20 Test-Iterationen ausgeführt. Die Resultate sind als Boxplots in **Abb. 3.12** zusammengefasst. Für die ersten beiden Funktionen performen PSO und DE ähnlich gut und zuverlässig, während RS eine deutliche Streuung nach unten aufweist. Bei der dritten Funktion ist jedoch insbesondere für DE eine weite Streuung der Maxima-Werte zu erkennen im Vergleich zu den anderen Optimierer. PSO hingegen kann hier, bis auf einzelne Ausreißer, immer die maximale Likelihood finden.

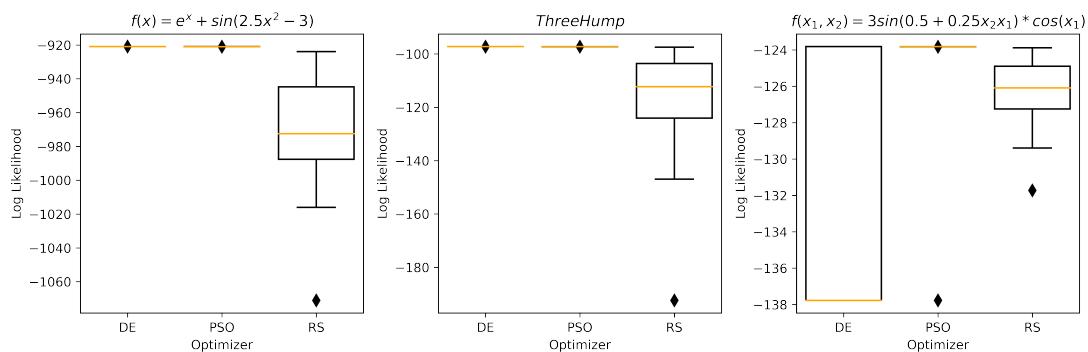


Abb. 3.12: Gefundene maximale Likelihood der Optimierer für verschiedene Groundtruth-Funktionen

Dies ist auf die Beschaffenheit der Likelihood-Landschaft bei dieser dritten Groundtruth-Funktion zurückzuführen. Neben dem eigentlichen Optimum (**Abb. 3.13**, links), gibt es auch nochstellen, die lokalen Maxima bzw. Sattelstellen ‘Plateaus’ gleichen (z.B. **Abb. 3.13**, rechts), so dass die Optimierer hier stecken bleiben können. Bei DE kommt dies häufiger vor, wie an dessen Median zu erkennen ist. Doch auch die Ausreißer bei

PSO röhren daher, dass die Partikel sich hier an diesem Plateau festsetzen. Trotzdem offenbart dieses Experiment nun einen Fall, in dem PSO DE überlegen ist.

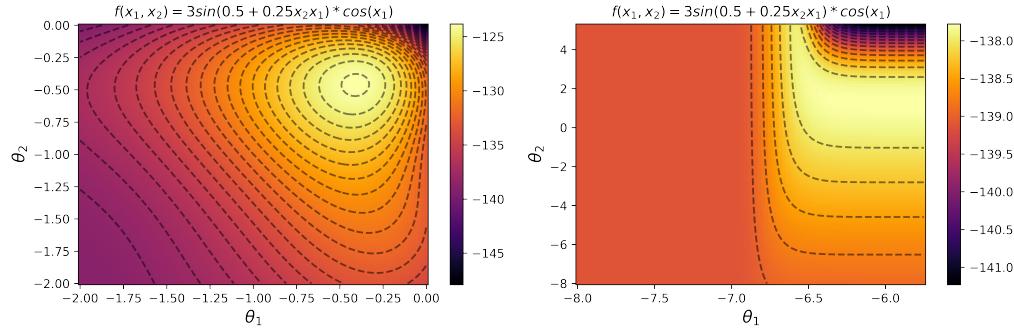


Abb. 3.13: Likelihood von Groundtruth 3: Optimum (links) und Plateau (rechts)

3.3.4 Einfluss von Rauschen (Noise)

Ein anderer Parameter des Experiments ist der sogenannte *Noise Scale*. Dieser reguliert die Stärke des zu den Trainingsdaten hinzugefügten Rauschens. In diesem Abschnitt wird das Verhalten der Optimierer bei varriierender Stärke des Rauschens auf den Trainingsdaten untersucht. Auch diese Variation führt zu veränderter Likelihood-Funktion und damit zu verschiedenen Optimierungsproblemen. Dazu wird jeder Optimierer 20 mal pro Noise Scale ausgeführt. In **Abb. 3.14** sind die Mittelwerte der maximalen gefundenen Likelihood sowie das dazugehörige 95%-Konfidenzintervall gezeigt.

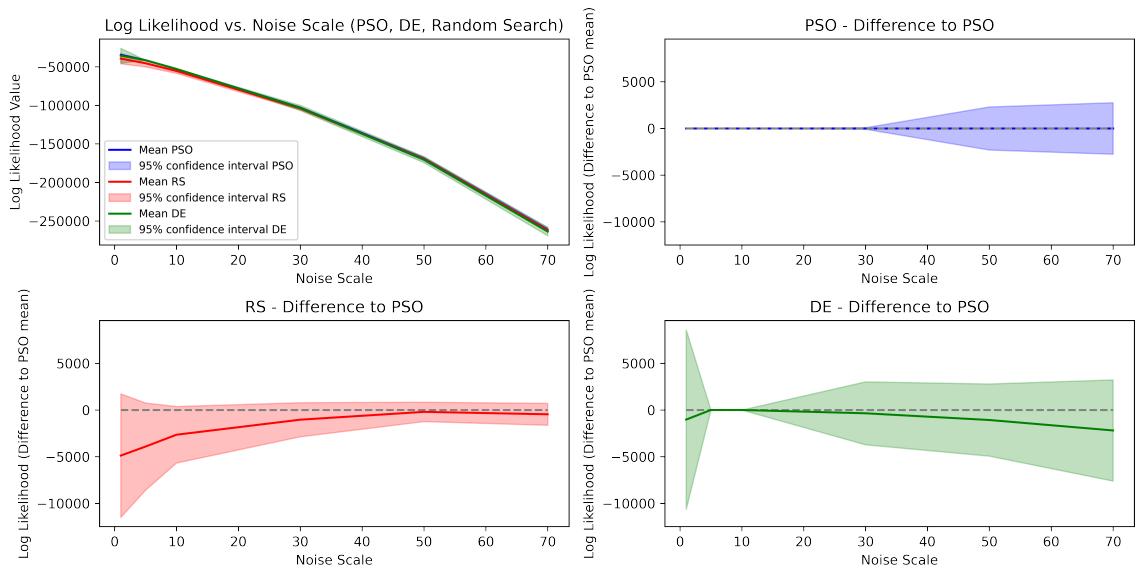


Abb. 3.14: Optimierer bei unterschiedlichem Noise Scale: Differenz zu PSO

Es ist zu erkennen, dass bei allen Optimierern der gefundene maximale Likelihood-Wert abnimmt je höher der Noise Scale ist (links oben). Dies ist allerdings auf die Beschaffenheit der resultierenden Likelihood-Funktionen zurückzuführend. Logischerweise wird es schwieriger, die den Daten zugrundeliegende Funktion nachzubilden, je

mehr Rauschen in den Daten enthalten ist. Um die Performance der Optimierer besser bewerten zu können, wird in den übrigen drei Plots die Abweichung zum Mittelwert von PSO dargestellt: Insbesondere für niedrige Noise Scales ist PSO sowohl im Mittel als auch in seiner Zuverlässigkeit besser als RS. Im Vergleich zu DE lässt sich sagen, dass PSO im Mittel meist eine ähnliche maximale Likelihood findet. Für höhere Noise Scales nimmt der Mittelwert der DE-Optima in Relation zum PSO allerdings ab. Obwohl auch bei PSO die Ergebnisse für größere Noise Scales stärker variieren, ist DE noch instabiler in den gefundenen Optima, wie am größeren Konfidenzintervall zu erkennen ist. Auch für dieses Experiment schneidet PSO im Vergleich zu den beiden Referenzalgorithmen besser ab. Trotzdem sind die Unterschiede hier eher gering.

3.3.5 Gesamtauswertung der Experimente

Zuletzt sollen alle der zuvor vorgestellten Experimente in einer gemeinsamen Grafik ausgewertet werden. Alle bisherigen Untersuchungen haben zu unterschiedlichen Likelihood-Funktionen und damit zu verschiedenen Optimierungsproblemen geführt. Nun soll mit dieser Gesamtauswertung die Performance aller Optimierer über alle untersuchten Optimierungsprobleme festgestellt werden.

Da die verschiedenen Likelihood-Funktionen unterschiedliche Skalen aufweisen, müssen die Werte zunächst skaliert werden. Daher werden für jede Likelihood-Funktion die Ergebnisse standardisiert, indem von den Ergebnissen der Mittelwert abgezogen wird und anschließend durch die Standardabweichung geteilt wird. Diese standardisierten Ergebnisse sind nun in **Abb. 3.15** visualisiert.

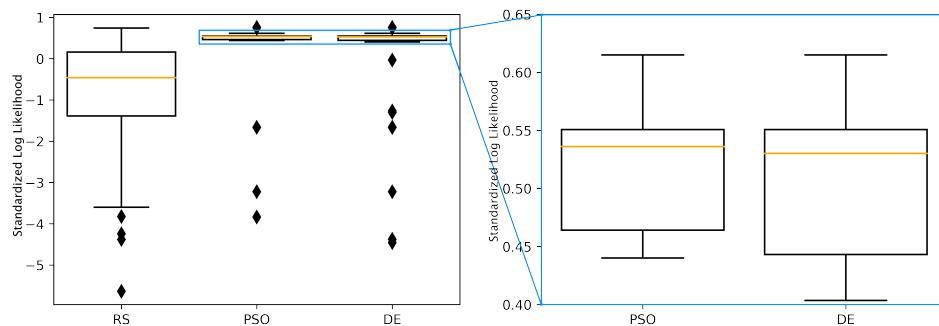


Abb. 3.15: Gesamtauswertung aller Experimente

Aus den Boxplots lässt sich bereits auf den ersten Blick ablesen, dass PSO konstant besser performt als der einfache Referenz-Algorithmus RS. Darüber hinaus weist er deutlich weniger Streuung auf und hat nur vereinzelte Ausreißer, er findet das Optimum also zuverlässiger. Im direkten Vergleich mit DE zeigt sich, dass die beiden Optimierer sich in ihrer Performance ähnlich verhalten. Trotzdem ist zu sehen, dass die Streuung nach unten von DE etwas größer ist als für PSO. Abschließend kann also

konkludiert werden, dass PSO einen einfachen Optimierer wie RS deutlich schlägt und mit komplexeren Algorithmen wie DE nicht nur mithalten kann, sondern teils sogar übertrifft. Es sollte noch darauf hingewiesen werden, dass diese Experimente nur eine Konfiguration mit 10 Partikeln und 100 Iterationen betreffen (d.h. insgesamt 1000 Funktionsauswertungen).

3.4 Untersuchung der Hyperparameter von PSO

Zu den im vorherigen Kapitel variierten Parametern, die allesamt zu unterschiedlichen Likelihood-Funktionen führen, können auch noch die Hyperparameter des Optimierers PSO variiert werden. In der verwendeten Implementierung von *PySwarms* können neben der Anzahl an Partikeln und Iterationen weitere Hyperparameter angepasst werden: c_1 und c_2 stellen den kognitiven bzw. den sozialen Parameter dar. Diese regulieren, ob ein Partikel seiner persönlich besten Position (\vec{P}_{best}) oder der global besten Postion des Schwarms (\vec{G}_{best}) folgt. Der Parameter w steuert die Trägheit (*inertia*) (vgl. PySwarms-Docs, 2022). Nachfolgend soll der Einfluss dieser Hyperparameter auf Funktionsweise und Ergebnis des Optimierers untersucht werden, wobei wieder das Daten-Sample aus Kap. 3.2 verwendet wird.

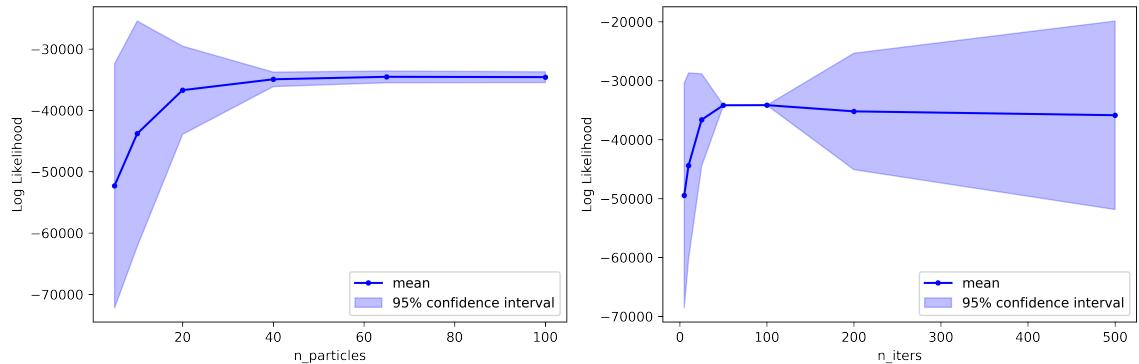


Abb. 3.16: Einfluss von Partikel- und Iterationszahl von PSO

Die Untersuchungen für die Anzahl an Partikeln und Iterationen ist in **Abb. 3.16** dargestellt. Jeweils wird einer der beiden Hyperparameter variiert, während der andere konstant bei 10 gehalten wird. Dargestellt sind Mittelwert und das 95%-Konfidenzintervall über 20 Durchläufe. Bezuglich der Partikel (links) lässt sich sagen, dass eine Erhöhung der Partikelanzahl das Ergebnis generell verbessert. Dies gilt aber nur bis ca. 40 Partikel, eine weitere Erhöhung für nicht zu signifikanten Verbesserungen des Ergebnisses. Die Darstellung des Konfidenzintervalls zeigt, dass bei wenigen Partikeln die Varianz der Ergebnisse deutlich höher ist. Hier spielt der Zufall (u.a. die initialen Positionen) eine größere Rolle, dessen Einfluss mit mehr Partikel immer weiter abnimmt.

Für die Iterationszahl ist zu erkennen, dass bis zu ca. 100 Iterationen eine Erhöhung zu besseren und stabileren Ergebnissen führt. Anders als erwartet werden die Ergebnisse dann mit zunehmender Iterationszahl jedoch wieder instabiler, wie an der Breite des Konfidenzintervalls zu erkennen ist, und die Mittelwerte der gefundenen Lösungen nehmen ab. Dies könnte darauf zurückzuführen sein, dass das Optimum zunächst gefunden wurde, das Verhalten einzelner Individuen (Partikel) dann aber dazu führt, dass dieses in manchen Fällen wieder verloren geht. Die verwendete Implementierung stoppt, im Gegensatz zu dem in Kap. 2.2 beschriebenen ursprünglichen Algorithmus, nämlich erst wenn die spezifizierte Anzahl an Iterationen erreicht ist.

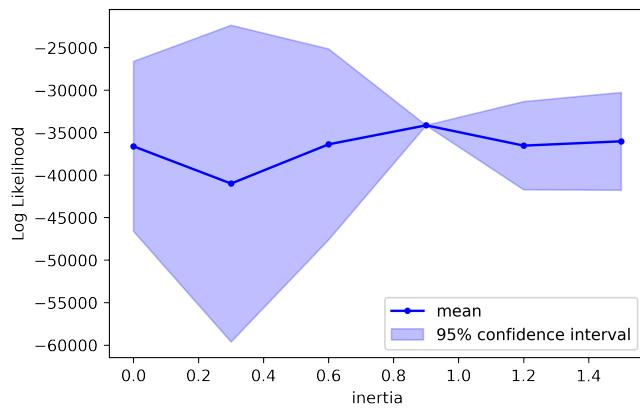


Abb. 3.17: Einfluss des Hyperparameters inertia (w)

Die Untersuchungen für den Hyperparameter inertia (w) in **Abb. 3.17** bei 100 Iterationen und 10 Partikel zeigen, dass der bisher gewählte Wert von 0,9 durchaus sinnvoll ist, da er sowohl das beste als auch die stabilsten Ergebnisse liefert. Eine höhere oder geringere Inertia führt zu suboptimalen Ergebnissen. Insgesamt kann das Fazit gezogen werden, dass die Wahl der Inertia durchaus wichtig ist, damit das Optimum konstant gefunden wird.¹¹

Zuletzt sollen noch PSO-Instanzen explorativer und exploitativer Naturen gegenübergestellt werden. Dies wird durch eine entsprechende Wahl der Hyperparameter c_1 und c_2 erreicht. Zur Untersuchung werden Bilder ausgewählter Optimierungsschritte der Instanzen betrachtet. Wird ausschließlich der kognitive Parameter c_1 gesetzt, verhält sich der Optimierer explorativ. Wie **Abb. 3.18** zeigt, suchen die einzelnen Partikel global (“explorieren” den Suchraum) und benötigen viele Schritte, um gemeinsam zum globalen Optimum zu kommen. Dies ist daran zu erkennen, dass beim letzten Schritt (99) die Partikel zwar näher, aber noch nicht beim Optimum sind.

¹¹Für weitere Untersuchungen über die Inertia sei auf *Wang u. a. (2013)* verwiesen.

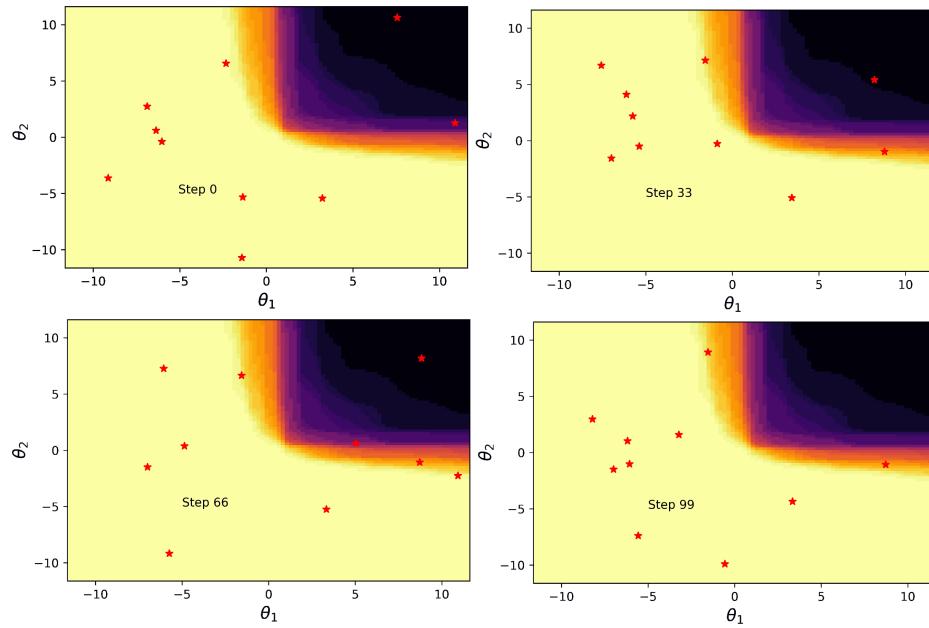


Abb. 3.18: Ausgewählte Optimierungsschritte einer PSO-Instanz mit $c1=3$ und $c2=0$

Wird aber nur der soziale Parameter $c2$ gesetzt, zeigt der Optimierer exploitative Verhalten. **Abb. 3.19** verdeutlicht, dass sich mehrere Partikel schnell an lokalen Lösungen gruppieren. Da keine individuelle Exploration zur Verbesserung dieser Lösungen durch die Parameterwahl erlaubt ist, wird das (globale) Optimum nicht gefunden.

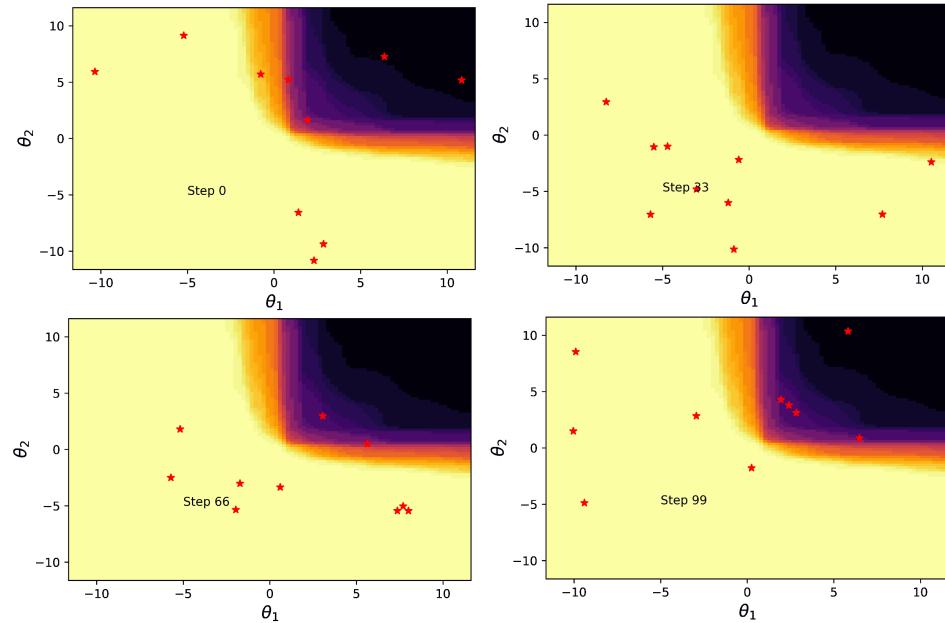


Abb. 3.19: Ausgewählte Optimierungsschritte einer PSO-Instanz mit $c1=0$ und $c2=3$

Die beiden skizzierten Untersuchungen deuten an, dass eine ausgeglichene Wahl von $c1$ und $c2$ zum besten Ergebnis führt. Die beiden lassen aber auch den Schluss zu, dass je nach Beschaffenheit des Problems unterschiedliche Konfigurationen sinnvoll sind. Sind neben dem globalen auch lokale Optima vorhanden, scheint ein explorativer Ansatz

sinnvoll. Gibt es hingegen nur ein einziges, eindeutiges Optimum, ist eine exploitative Instanz empfehlenswert (da weniger Schritte nötig). Dieses Gedankenspiel verlangt aber entsprechende Kenntnisse über die zu optimierende Funktion.

Allgemein lässt sich aus den Untersuchungen dieses Kapitels schließen, dass die Wahl der Hyperparameter die Performance von PSO maßgeblich beeinflusst. Auf sinnvolle Parameter ist deshalb zu achten, damit das Optimum gefunden wird.

4 Fazit

Nachdem der Optimierungsalgorithmus PSO ausführlich in verschiedenen Experimenten empirisch untersucht wurde, kann nun ein fundiertes Fazit zur Funktionsweise und zur Eignung von PSO zur Lösung von Optimierungsproblemen gezogen werden. Die Funktionsweise des Algorithmus wurde an konkreten Beispielen durch Darstellung der Optimierungsschritte dargelegt. Daraus geht insbesondere hervor, dass der Optimierer, wie auch Vogelschwärme in der Realität, zwar zu Beginn eher zufällig zu agieren scheint. Im Laufe der Optimierung wird aber deutlich, dass sich der Gesamtschwarm strukturiert dem Optimum annähert.

Aus den zahlreichen Experimenten, bei denen unterschiedliche Likelihood-Funktionen erzeugt wurden, kann die Performance von PSO im Vergleich zu RS und DE überzeugen. Der Vergleich mit dem einfachen Optimierungsverfahren RS zeigt deutlich, dass PSO nicht nur das Optimum besser approximiert, sondern dieses auch zuverlässiger findet. Wird hingegen der komplexere Algorithmus DE als Referenz herangezogen, ist eine differenziertere Betrachtung notwendig: Allgemein sind sich die beiden Optimierer in ihrer Performance sehr ähnlich. Für einzelne Optimierungsprobleme schneidet PSO besser ab als DE. Für ein konkretes Beispiel sei auf die Untersuchung mit verschiedenen Groundtruth-Funktionen verwiesen, bei der PSO für die dritte Funktion DE deutlich outperforms. Allerdings muss beachtet werden, dass teils auch der gegen- teilige Fall auftritt. In einer Gesamtauswertung konnte festgestellt werden, dass die Median-Performance von PSO und DE beinahe gleich ist. Jedoch streut PSO in einem geringeren Maße als DE nach unten. Diese Ergebnisse gelten für PSO mit 10 Partikeln und 100 Iterationen.

Eine anschließende Untersuchung des Einflusses der Hyperparameter von PSO hat offenbart, dass eine richtige Wahl der Hyperparameter-Werte elementar für die Performance des Optimierers ist. Dabei konnte ebenfalls gezeigt werden, dass höhere Werte nicht immer besser sind, wie die Beispiele der Iterationszahl und der Inertia verdeutlicht haben. Insbesondere aus der Abstimmung von kognitivem und sozialem Parameter ging hervor, dass eine ungünstige Parameter-Auswahl den Algorithmus beinahe unbrauchbar machen kann. Auch mit der Untersuchung zu Beginn mit lediglich 10 Iterationen, bei der DE noch bessere Ergebnisse als PSO liefert hat, konnte die Relevanz der Parameter-Wahl dargelegt werden.

Zuletzt muss nun noch die Eignung von PSO zur Lösung des Ausgangsproblems betrachtet werden. Es konnte festgestellt werden, dass der Optimierer das Problem der Likelihood-Maximierung sehr gut lösen kann. Bei entsprechenden Trainingsdaten (vor

allem ausreichende Anzahl an Datenpunkten) wird dann auch die Groundtruth-Funktion vom GPM gut approximiert. Selbst bei weniger Datenpunkten bietet sich noch ein besseres Bild als wenn RS als Optimierer verwendet wurde. Im Vergleich zu DE können wieder keine signifikanten Unterschiede festgestellt werden.

Jedoch muss die Eignung des Ausgangsproblems zur Untersuchung von PSO kritisch betrachtet werden. Da die Likelihood immer nur indirekt beeinflusst wird (Anzahl an Datenpunkten, Dimensionen, Groundtruth, etc.), kann nie direkt ein gewünschtes Optimierungsproblem generiert werden. Beispielsweise wäre die Untersuchung für multimodale Probleme oder mit Rauschen in der Zielfunktion interessant. Insbesondere die Optimierung der Funktionen, die hier als Groundtruth verwendet wurden (bspw. *Rastrigin*, *Three-Hump*), wäre aufschlussreich, da es sich hier um klassische Optimierungsprobleme zur Feststellung der Eignung von Algorithmen handelt.

Trotzdem kann insgesamt festgehalten werden, dass PSO als Naturanalogie von Vogelschwarmbewegungen auch in der Optimierung gewinnbringend eingesetzt werden kann. Das Konzept, das in der Natur über Jahrtausende hinweg perfektioniert wurde, kann auf die Welt der Optimierungsprobleme übertragen werden. Im Gegensatz zu zahlreichen anderen Algorithmen, die sich ebenfalls Beobachtungen aus der Natur zunutzen machen¹², ist die Idee der PSO tatsächlich sinnvoll und trägt zur effektiven Lösung von Optimierungsproblemen bei.

¹²Eine Liste zahlreicher anderer naturanalogen Optimierungsalgorithmen ist zu finden unter <https://github.com/fcampelo/EC-Bestiary> (Stand: 11.08.2022).

Literatur

- [PySwarms-Docs 2022] : *PySwarms 1.3.0 documentation: pyswarms.single package.* 2022. – URL https://pyswarms.readthedocs.io/en/latest/api/pyswarms.single.html#module-pyswarms.single.global_best. – Zugriffssdatum: 04.08.2022
- [Scikit-Learn-Docs2 2022] : *Scikit-Learn Dokumentation: sklearn.gaussian_process.GaussianProcessRegressor.* 2022. – URL https://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.GaussianProcessRegressor. – Zugriffssdatum: 04.08.2022
- [Scikit-Learn-Docs1 2022] : *Scikit-Learn Dokumentation: sklearn.gaussian_process.kernels.RBF.* 2022. – URL https://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.kernels.RBF.html. – Zugriffssdatum: 04.08.2022
- [SciPy-Docs 2022] : *SciPy v1.7.1 Manual: scipy.optimize.differential_evolution.* 2022. – URL https://docs.scipy.org/doc/scipy-1.7.1/reference/reference/generated/scipy.optimize.differential_evolution.html. – Zugriffssdatum: 04.08.2022
- [Aiolli und Donini 2014] AIOLLI, Fabio ; DONINI, Michele: Learning anisotropic RBF kernels. In: *Artificial Neural Networks and Machine Learning – ICANN 2014* (2014), S. 515–522
- [Bonabeau u. a. 1999] BONABEAU, Eric ; DORIGO, Marco ; THERAULAZ, Guy: *Swarm Intelligence: From Natural to Artificial Systems*. USA : Oxford University Press, Inc., 1999
- [Boyd und Vandenberghe 2004] BOYD, Stephen ; VANDENBERGHE, Lieven: *Convex optimization*. Cambridge University Press, 2004
- [Claus u. a. 2003] CLAUS, V. ; DEWESS, G. ; DEWESS, M. ; DIEKERT, V. ; FUCHSSTEINER, B. ; GOTTWALD, S. ; GÜNDL, S. ; HOSCHEK, J. ; OLDEROG, E.-R. ; RICHTER, M. M. ; SCHENKE, M. ; WIDMAYER, P. ; ZEIDLER, E.: *Topologie — Mathematik des Qualitativen Verhaltens*. S. 705–738. In: GROSCHÉ, G. (Hrsg.) ; ZEIDLER, E. (Hrsg.) ; ZIEGLER, D. (Hrsg.) ; ZIEGLER, V. (Hrsg.): *Teubner-Taschenbuch der Mathematik: Teil II*. Wiesbaden : Vieweg+Teubner Verlag, 2003
- [Clerc 2006] CLERC, Maurice: *Adaptations*. Kap. 10, S. 129–138. In: *Particle Swarm Optimization*, John Wiley & Sons, Ltd, 2006

- [Duvenaud 2014] DUVENAUD, David K.: *Automatic Model Construction with Gaussian Processes*. Cambridge, UK, University of Cambridge, Dissertation, 2014
- [Eberhart und Shi 2001] EBERHART ; SHI, Yuhui: Particle swarm optimization: developments, applications and resources. In: *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)* Bd. 1, 2001, S. 81–86
- [Heppner und Grenander 1990] HEPPNER, Frank ; GRENANDER, U.: *A Stochastic Nonlinear Model for Coordinate Bird Flocks*. S. 233–238. In: KRASNER, S. (Hrsg.): *The Ubiquity of Chaos*, AAAS Publications, 1990
- [Karvonen und Oates 2022] KARVONEN, Toni ; OATES, Chris J.: Maximum Likelihood Estimation in Gaussian Process Regression is Ill-Posed. In: *arXiv preprint arXiv:2203.09179* (2022)
- [Kennedy und Eberhart 1995] KENNEDY, J. ; EBERHART, R.: Particle swarm optimization. In: *Proceedings of ICNN'95 - International Conference on Neural Networks* Bd. 4, 1995, S. 1942–1948
- [Murphy 2012] MURPHY, Kevin P.: *Machine Learning - A Probabilistic Perspective*. Cambridge : MIT Press, 2012
- [Myung 2003] MYUNG, In J.: Tutorial on maximum likelihood estimation. In: *Journal of Mathematical Psychology* 47 (2003), Nr. 1, S. 90–100
- [Olivan 2019] OLIVAN, Patrick: *Methode zur organisatorischen Gestaltung radikaler Technologieentwicklungen unter Berücksichtigung der Ambidextrie / Organisational design method for radical technology development based on the principle of ambidexterity*, Uni-Stuttgart, Dissertation, 08 2019
- [O'Reilly und Tushman 2021] O'REILLY, Charles A. ; TUSHMAN, Michael L.: *Lead and Disrupt: How to Solve the Innovator's Dilemma, Second Edition*. Stanford University Press, 2021
- [Pagmo 2022] PAGMO: *Particle Swarm Optimization (PSO)*. 2022. – URL <https://esa.github.io/pagmo2/docs/cpp/algorithms/pso.html>. – Zugriffsdatum: 12.08.2022
- [Poli u. a. 2007] POLI, Riccardo ; KENNEDY, James ; BLACKWELL, Tim: Particle swarm optimization - An overview. In: *Swarm Intelligence* 1 (2007), S. 33–57
- [Reynolds 1987] REYNOLDS, Craig W.: Flocks, Herds and Schools: A Distributed Behavioral Model. In: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA : Association for Computing Machinery, 1987 (SIGGRAPH '87), S. 25–34

- [Vidya und Hari 2022] VIDYA, G. S. ; HARI, V. S.: Prediction of bus passenger traffic using gaussian process regression. In: *Journal of Signal Processing Systems* (2022)
- [Wang u. a. 2013] WANG, Shigang ; ZHOU, Fangfang ; WANG, Fengjuan: Effect of Inertia Weight w on PSO-SA Algorithm. In: *International Journal of Online and Biomedical Engineering (iJOE)* 9 (2013), Nr. S6, S. 87–91
- [Yudong Zhang 2015] YUDONG ZHANG, Genlin J.: A Comprehensive Survey on Particle Swarm Optimization Algorithm and Its Applications. In: *Mathematical Problems in Engineering* 2015 (2015), S. 38

Selbständigkeitserklärung

Wir versichern hiermit, dass wir den vorliegenden Projektbericht mit dem Thema

Untersuchung von Particle Swarm Optimization zur Optimierung Gaußscher Prozessmodelle

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben. Wir versichern zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Ort, Datum

Unterschrift

Ort, Datum

Unterschrift

Ort, Datum

Unterschrift

Ort, Datum

Unterschrift