DOKUMENTATION
NICHT LINEARE SUCHE

Jan Luca Emil Krüger
MTS-23

Inhaltsverzeichnis

Untersuchte Funktionen

Funktion 1

Die erste Funktion ist durch die Aufgabenstellung vorgegeben:

$$f(x,y) = (\mathrm{x}+5)(\mathrm{x}+1)(\mathrm{x}-2)(\mathrm{x}+4)\mathrm{x}(\mathrm{y}-1)(\mathrm{y}+2)(\mathrm{y}-3)(\mathrm{y}+5)$$

Der Gradient der Funktion lautet dabei wie folgt:

$$grad\big(f(x,y)\big) = \begin{pmatrix} (-3+\mathrm{y})(-1+\mathrm{y})(2+\mathrm{y})(5+\mathrm{y})(-40-76\mathrm{x}+27\mathrm{x}^2+32\mathrm{x}^3+5\,\mathrm{x}^4) \\ (-2+\mathrm{x})\mathrm{x}(1+\mathrm{x})(4+\mathrm{x})(5+\mathrm{x})(-19\ -\ 30\,\mathrm{y}\ +\ 9\,y^2\ +\ 4\,y^3) \end{pmatrix}$$

Weiterhin wird die Funktion im folgendem Intervall untersucht:

$$I\ =\ [-2{,}2]\ \times\ [-2{,}2]$$

Funktion 2

Die zweite Funktion is durch die Aufgabenstellung vorgegeben:

$$f(x,y) = \sin(x^2+y) - \cos(y^2-x)$$

Der Gradient der Funktion lautet dabei wie folgt:

$$grad\big(f(x,y)\big) = \begin{pmatrix} 2xcos(x^2+y) - \sin(y^2-x) \\ \cos(x^2+y) + 2y\sin(y^2-x) \end{pmatrix}$$

Weiterhin wird die Funktion im folgendem Intervall untersucht:

$$I\ =\ [-5\pi, 5\pi]\ \times\ [-5\pi, 5\pi]$$

Funktion 3

Die dritte Funktion war frei wählbar. Dabei wurde folgende Funktion gewählt:

$$f(x,y) = (x+1)^2 + (y-0.5)^2$$

Diese Funktion wurde gewählt, da sie nur einen Extrempunkt, nämlich *E(-1;0.5)*, besitzt, wodurch beide Verfahren unabhängig von Startpunkt und den übergebenen Parametern diesen Punkt, bzw. einen Punkt in den unmittelbaren Nähe, als Ergebnis haben sollten.

Der Gradient der >Funktion lautet dabei wie folgt:

$$grad\big(f(x,y)\big) = \begin{pmatrix} 2x+2 \\ 2y-1 \end{pmatrix}$$

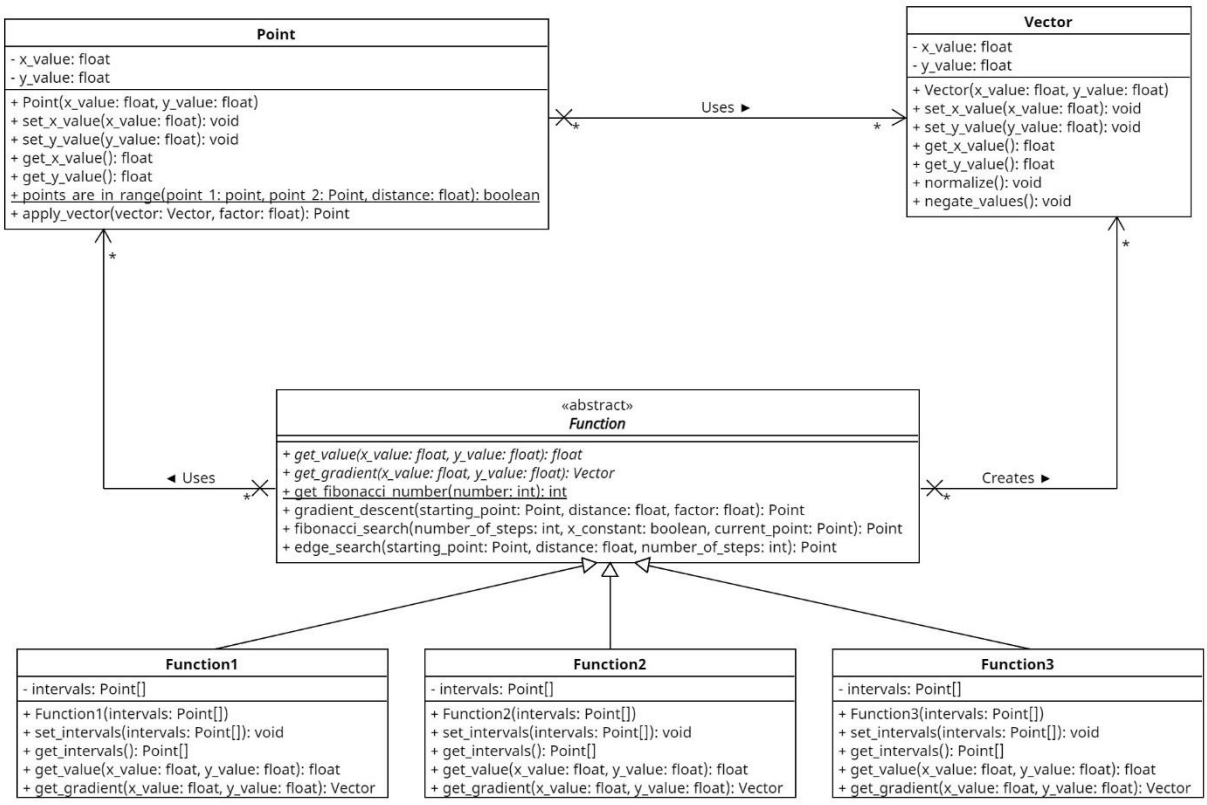
Weiterhin wird die Funktion im folgenden Intervall untersucht:

$$I = [-5{,}5] \times [-5{,}5]$$

Verbale Beschreibung des Codes

UML-Klassendiagramm

Das Programm wurde nach folgendem UML-Klassendiagramm implementiert:



Das Klassendiagramm befindet sich zudem sowohl als PNG- als auch als UXF-Datei im Ordner „./doc/diagrams“.

Beschreibung nach Klassen

Anmerkung

Es wurde nicht für jede Methode ein Sequenzdiagramm erstellt, sondern nur für die Methoden *fibonacci_search* , *edge_seach* & *gradient_descend* (also dem Fibonacci, Kanten & Gradienten Suchverfahren). Dies wurde getan, da alle anderen Methoden des Programms entweder Konstruktoren, Setter, Getter oder sehr simple Methoden sind. Nur die drei oben genannten Methoden sind komplex genug, dass dort ein Sequenzdiagramm sinnvoll sei.

Weiterhin befinden sich alle Sequenzdiagramme sowohl als PNG- als auch als UXF-Datei im Ordner „./doc/diagrams“.

(Abstrakte) Klasse *Function*

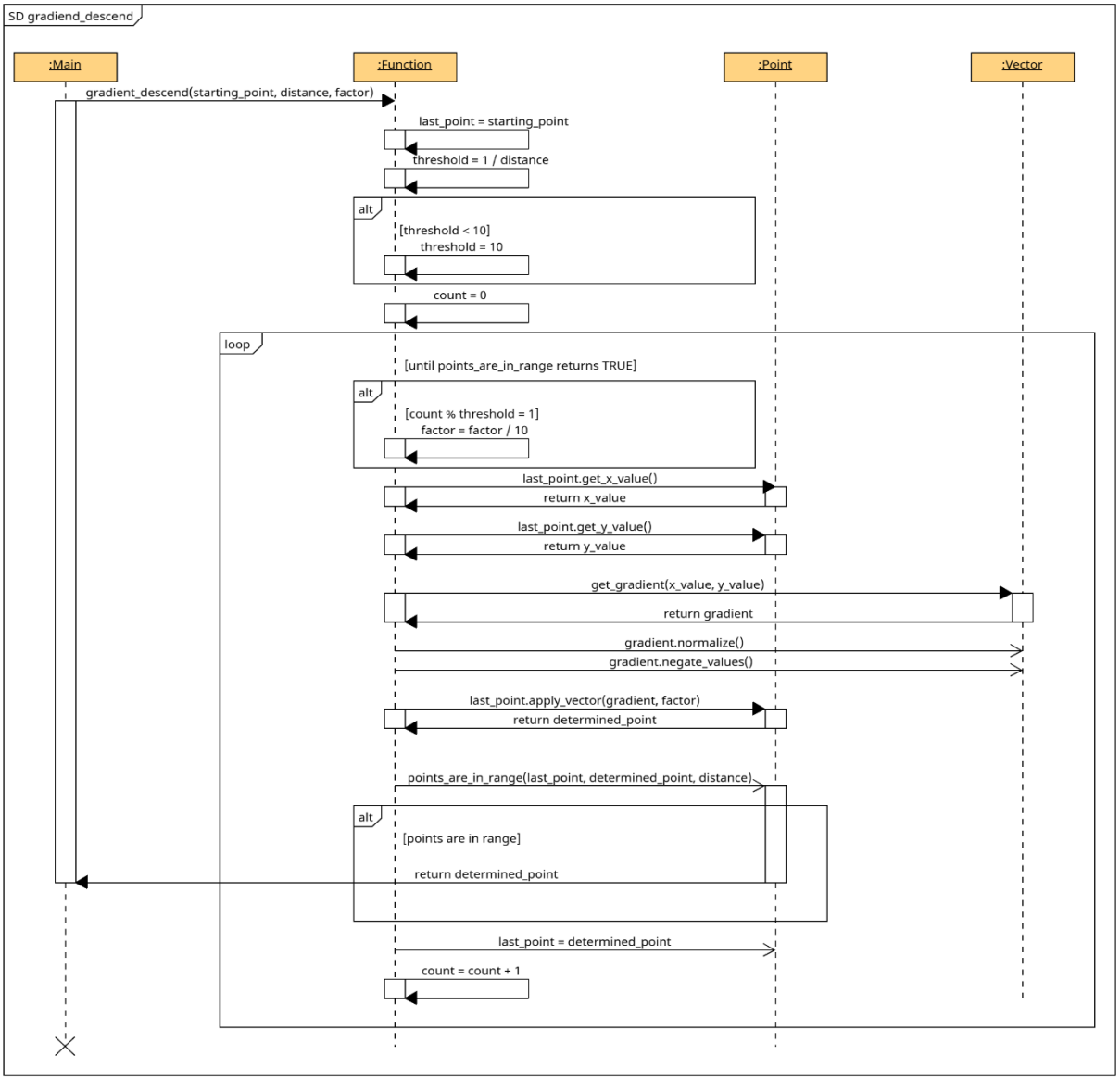
Die abstrakte Klasse *Function* verfügt über keinen Konstruktor & somit auch nicht über Getter & Setter. Sie verfügt über die beiden abstrakten Methoden *get_value* & *get_gradient,* welche die implementierenden (hartgecodeten) Klassen benutzen, um ihren Funktionswert als *Float* an einer bestimmten Stelle zu berechnen *(get_value)* oder um den Gradienten an einer bestimmten Stelle als Objekt der Klasse *Vector* zu berechnen (get_gradient).

Die Klasse verfügt weiterhin über die statische Methode *get_fibonacci_number*, welche iterativ die n-te Fibonacci-Zahl berechnet. Die beiden Startwerte 0 & 1 werden dabei am Anfang der Methode durch IF-Abfragen abgefangen. Sollte eine höhere Fibonacci-Zahl bestimmt werden, wird diese in Abhängigkeit der beiden Startwerte iterativ berechnet. Der Rückgabewerte ist dabei die n-te Fibonacci-Zahl vom Typ *Integer.*

Weiterhin verfügt die Klasse über die Methode *gradiend_descend*. Diese ist die Implementation des Gradientenabstiegverfahrens. Diese Methode hat 3 Parameter. Der erste Parameter ist ein Objekt der Klasse *Point*, mit dem Namen *starting_point*. Dieser ist der Punkt, von welchem das Verfahren initial gestartet wird. Der zweite Parameter hat den Typ *Float*, mit dem Namen *distance*. Dieser gibt an, wie weit 2 aufeinader folgende Punkte maximal liegen dürfen, um als gleich zu gelten & somit das Verfahren abzubrechen. Dies ist das ε, welches in der Laufzeit & Genauigkeitsanalyse betrachtet wird. Der letzte Parameter ist ebenfalls vom Typ *Float*, mit dem Name *factor*. Dieser gibt an, inweit einem Vektor von einem Punkt aus gefolgt werden soll. Dies ist das n, welches in der Laufzeit & Genauigkeitsanalyse betrachtet wird.
Zunächst werden einige Variablen initialisiert, bevor das eigentliche Verfahren gestartet wird. Zunächst wird der zuletzt gefundene Punkt, *last_point*, ein Objekt der Klasse *Point*, als den übergebenen Startpunkt initialisiert. Anschließend wird die Variable *threshold* vom Typ *Integer* initialisiert, als dem Reziproken von dem übergebenen ε, also dem Parameter *distance*. Diese Variable existiert, um endlose Schleifen zu vermeiden, in dem Fall, dass der Parameter *factor* größer als der Parameter *distance* ist, da es in diesem Fall dazu kommt, dass das Verfahren um das Minimum umher springt und die aufeinander folgenden Punkte nicht unter das ε fallen. Weiterhin wurde festgelegt, dass diese Variable einen Minimalwert von 10 haben soll. Abschließend wird noch die Variable *count* vom Typ *Integer* initialisiert. Diese dient dazu, um zu überprüfen, ob der Schwellenwert *threshold* überschritten wurde.
Im Anschluss beginnt das eigentliche Verfahren in einer While-TRUE-Schleife. Zunächst wird überprüft, ob der Schwellenwert *threshold* mit der Variable *count* überschritten wurde. Ist dies der Fall, wird der Parameter *factor* auf ein Zehntel verkleinert, um endlose Schleifen zu vermeiden. Anschließend wird mit Hilfe der Methode *get_gradient* der Gradient am aktuellen, also am x- & y-Wert der Variable *last_point* bestimmt. Die Methode *get_gradient* ist zwar eine abstrakte Methode, die von der Klasse *Function* nicht implemenetiert wird, jedoch ist dies kein Problem, da die Methode *gradiend_descend* nur von Objekten der Klasse *Function1*, *Function2* & *Function3* aufgerufen wird,

welche die Methode *get_gradiend* implementiert haben.
Im Anschluss wird dieser Gradient mit Hilfe der Methode *normalize* normalisiert und deren Vorzeichen mit Hilfe der Methode *negate* vertauscht.
Dieser angepasste Vektor wird nun zusammen mit dem Parameter *factor* der Methode *apply_vector* der Klasse *Point* übergeben, um einen neuen Punkt zu bestimmen, nämlich *determined_point*¸ein Objekt der Klasse *Point.* Anschließend wird der alte Punkte, *last_point*, der aktuelle Punkt, *detemined_point*, & der Parameter *distance* der statische Methode *points_are_in_range* der Klasse *Point* überegeben, um zu überprüfen, ob das Verfahren abgebrochen werden kann. Gibt die Methode TRUE zurück, wird mit Hilfe eines Break-Statements die Schleiefe beendet. Sollte dem nicht der Fall sein, wird zunächst der zuletzt bestimmte Punkt, *last_point*, mit dem neu bestimmten Punkt, *determined_point*, überschrieben, um die Variable für die nächste Iteration zu aktualisieren. Außeredem wird die Zählvariable *count* inkrementiert, um endlose Schleifen zu vermeiden.
Sollte die Abbruchbedingung erfüllt wurden sein, wir der zuletzt bestimmte Punkt, *determined_point*, zurückgegeben.

Das Sequenzdiagramm dieser Methode sieht wie folgt aus:



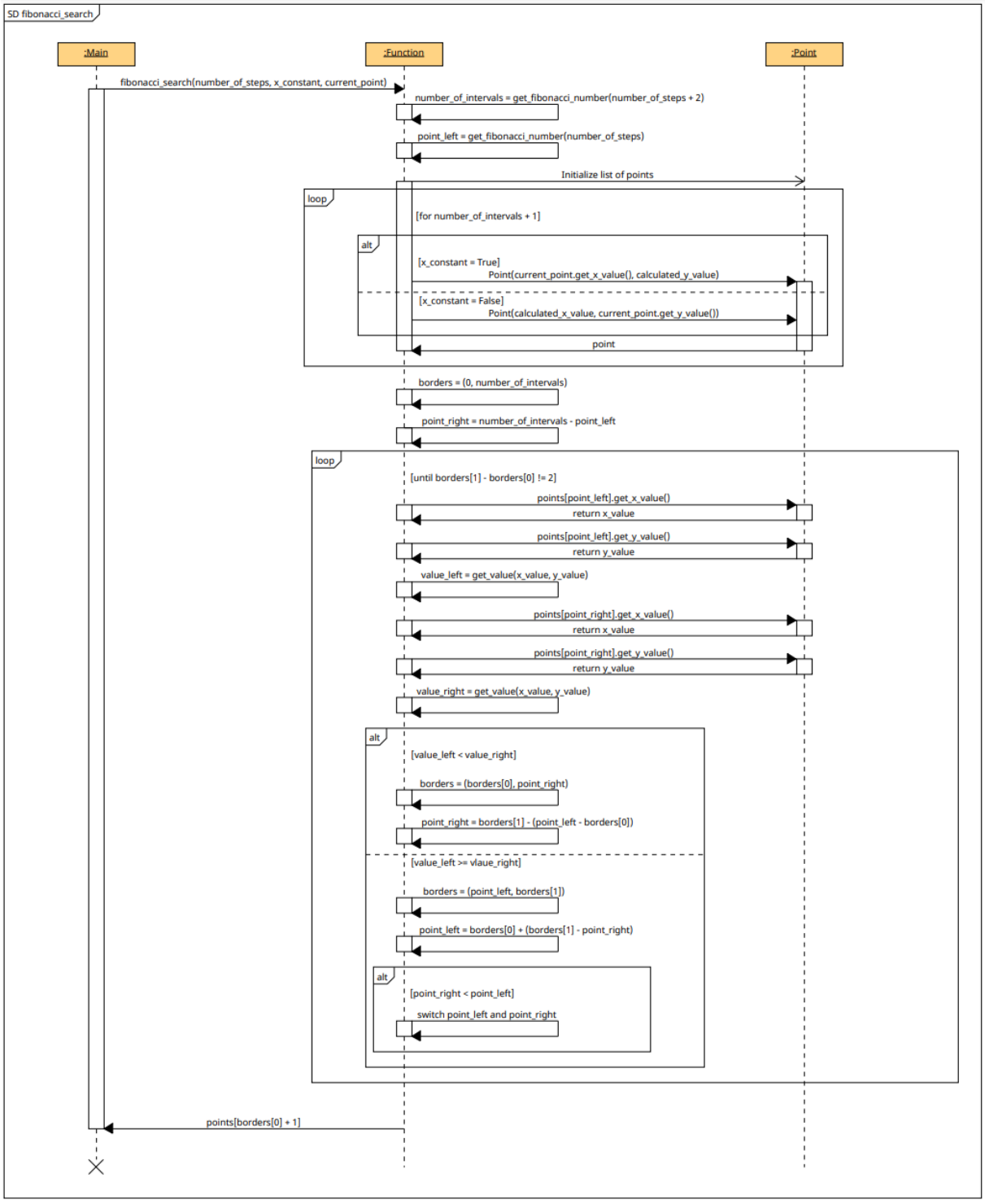
Weiterhin verfügt die Klasse über die Methode *fibonacci_search*, welche die Implementierung des Fibonacci Suchverfahrens ist. Diese hat 3 Parameter. Der erste Parameter *number_of_steps* vom Typ *Integer* gibt an, wie viele Funktionswerte berechnet werden müssen. Dies ist das n, welches in der Laufzeit & Genauigkeitsanalyse betrachtet wird. Der zweite Parameter *x_constant* vom typ Boolean gibt an, welche der beiden Achsen untersucht wird. Ist dieser Boolean gleich TRUE, wird die y-Achse untersucht. Ist dieser Boolean gleich FALSE, wird die x-Achse untersucht. Der letzte Parameter *current_point* vom Typ *Point* liefert den konstanten x-Werte, falls die y-Achse untersucht wird, beziehungsweise den konstanten y-Wert, falls die x-Achse untersucht wird.
Zunächst werden einige Variablen initialisiert, bevor das eigentliche Verfahren gestartet wird. Die Anzahl an Intervallen *number_of_interavls* vom Typ *Integer* wird als n+2. Fibonaccizahl mit Hilfe der Methode *get_fibonacci_number* bestimmt. Der erste Teilpunkt *point_left* vom Typ *Integer* wird als n. Fibonaccizahl mit Hilfe der Methode *get_fibonacci_number* bestimmt. Diese Variable spiegelt den Index des aktuell linken Punkts wieder. Anchließend werden die zu untersuchten Punkte bestimmt. Diese werden in der Variable *points* gespeichert, einer Liste, die Objekte der Klasse *Point* speichert. Nun werden in Abhängigkeit der Intervallsgrenzen *interavls* der zu untersuchenden Funktion die Punkte bestimmt, wobei in Abhängikeit des Parameters *x_constant* entweder der x- oder y-Werte konstant vom Parameter *current_point* entnommen wird. Der entsprechend andere Wert der Punkte wird so bestimmt, dass die Punkte auf der entsprechenden Achse einen gleichen Abstand zueinander haben. Die entsprechende Schleife dafür geht bis zu *number_of_interavls + 1,* um die rechte Intervallsgrenze ebenfalls mit einzubinden.
Anschließend wird das zu untersuchte Intervall *borders*, ein *Tupel* vom Typ *Integer*, initialisiert. Dieses gibt die Indices der Punkte an, die das zu untersuchte Interval eingrenzen, weshalb die Variable mit den Werten 0 & *number_of_intervals* initialisiert wird.
Abschließend wird noch der Punkt initialisiert, der von der rechten Interavllsgrenze den gleichen Abstand hat, wie die Variable *point_left* von der linken Intervallsgrenze hat. Dies wird in der Variable *point_right* vom typ *Integer* gespeichert, somit spiegelt diese Variable genau wie *point_left* den Index des Punktes wieder.
Anschließend beginnt das eigentliche Verfahren. Zunächst werden die Funktionswerte am linken & rechten Punkt mit Hilfe der Methode *get_value* berechnet & in den Variablen *value_left* & *value_right* vom Typ *Float* gespeichert. Die Methode *get_value* ist zwar eine abstrakte Methode, die von der Klasse *Function* nicht implemenetiert wird, jedoch ist dies kein Problem, da die Methode *fibonacci_search* nur von Objekten der Klasse *Function1*, *Function2* & *Function3* aufgerufen wird, welche die Methode *get_value* implementiert haben. Anschließend werden diese beiden Funktionswerte miteinander verglichen.

- Sollte *value_left* kleiner als *value_right* sein, wird das Intervall rechts eingeschränkt, das heißt, dass der zweite Wert der Variable *borders* zum Wert der Variable *point_right* gesetzt. Anschließend wird die Variable *point_right* so neu gesetzt, dass sie den gleichen Abstand von der rechten Interavllsgrenze hat, wie die Variable *point_left* von der linken Intervallsgrenze hat.
- Sollte *value_left* größergleich der Variable *value_right*, wird das Intervall links eingeschränkt, das heißt, dass der erste Wert der Variable *borders* zum Wert der Variable *point_left* gesetzt. Anschließend wird die Variable *point_left* so neu gesetzt, dass sie den gleichen Abstand von der linken Interavllsgrenze hat, wie die Variable *point_right* von der rechten Intervallsgrenze hat.

Abschließend warden die Werte der Variablen *point_left* und *point_right* noch getauscht, falls *point_left* größer als *point_right* ist, um sicherzustellen, dass *point_left* immer kleiner ist als *point_right*.

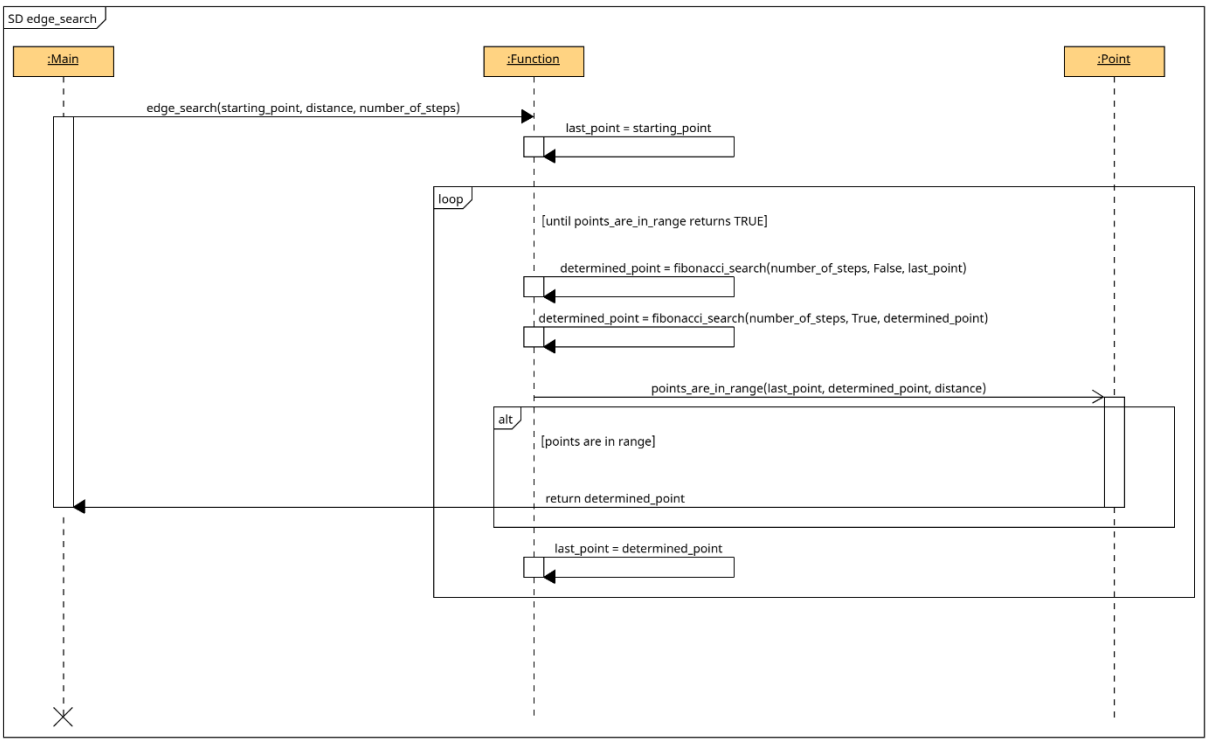
Dieses Verfahren wird solange wiederholt, bis es nur noch einen Punkt im untersuchten Intervall gibt, das heißt, dass die Differenz der beiden Werte der Variable *borders* gleich 2 ist.
Ist das Verfahren beendet, wird der einzelne Punkt im Intervall zurückgegeben. Dieser ist dabei der Punkt in der Liste *points* an der Stelle des ersten Wertes von *border* inkrementiert um 1.

Das Sequenzdiagramm dieser Methode sieht wie folgt aus:



Abschließend besitzt die Klasse über die Methode *edge_search*, welche die Implementation des Kanten Suchverfahrens ist. Diese Methode besitzt 3 Parameter. Der erste Parameter ist ein Objekt der Klasse *Point*, mit dem Namen *starting_point*. Dieser ist der Punkt, von welchem das Verfahren initial gestartet wird. Der zweite Parameter hat den Typ *Float*, mit dem Namen *distance*. Dieser gibt an, wie weit 2 aufeinader folgende Punkte maximal liegen dürfen, um als gleich zu gelten & somit das Verfahren abzubrechen. Dies ist das ε, welches in der Laufzeit & Genauigkeitsanalyse betrachtet wird. Der letzte Parameter ist vom Typ *Integer*, mit dem Namen *number_of_steps*. Diese gibt die Anzahl an Funktionsberechnungen an, die beim Fibonacci Suchverfahren gemacht werden müssen. Somit ist dieser Parameter ein Parameter für die Funktionsaufrufe der Methode *fibonacci_search*. Zunächst wird der zuletzt besuchte Punkt *last_point*, ein Objekt der Klasse *Point*, als den Startpunkt, *starting_point*, initialisiert.
Anschließend beginnt das eigentliche Verfahren. Zunächst wird das Fibonacci Suchverfahren einmal in x-Richtung & einmal in y-Richtung angewendet, das heißt, dass der Parameter *x_constant* beim ersten Funktionsaufruf FALSE ist & beim zweiten Funktionsaufruf TRUE ist. Dabei ist außerdem anzumerken, dass der übergebenen Punkt beim ersten Aufruf der zuletzt besuchte Punkt, *last_point*, ist, während er beim zweiten Funktionsaufruf *determined_point*, also die Variable, in welcher der Rückgabewert des ersten Aufrufs gespeichert wird, ist. Der Rückgabewert des zweiten Aufrufs wird dabei ebenfalls in der Variable *determined_point* gespeichert. Anschließend wird der alte Punkte, *last_point*, der aktuelle Punkt, *detemined_point*, & der Parameter *distance* der statische Methode *points_are_in_range* der Klasse *Point* überegeben, um zu überprüfen, ob das Verfahren abgebrochen werden kann. Gibt die Methode TRUE zurück, wird mit Hilfe eines Break-Statements die Schleiefe beendet. Sollte dem nicht der Fall sein, wird der zuletzt bestimmte Punkt, *last_point*, mit dem neu bestimmten Punkt, *determined_point*, überschrieben, um die Variable für die nächste Iteration zu aktualisieren.
Sollte die Abbruchbedingung erfüllt wurden sein, wir der zuletzt bestimmte Punkt, *determined_point*, zurückgegeben.

Das Sequenzdiagramm dieser Methode sieht wie folgt aus:



Klasse Function1, Function2 & Function3

Alle 3 Klassen implementieren die abstrakte Klasse *Function.*

Alle 3 Klassen verfügen nur über ein einzelnes Attribut, nämlich *intervals*, eine Liste vom *Point*-Objekten. Diese werden. Dieses Attribut wird für das Fibonacci Suchverfahren genutzt, um den Bereich abzugrenzen, in welchem untersucht wird. Bei Erstellen von Objekten dieser Klasse hat die Liste dabei nur 2 Einträge, mit welchem die untersuchte Fläche als Viereck abgegrenzt wird.
Im Falle von Funktion 1 sind das P(-2.0;-2.0) & P(2.0;2.0).
Im Falle von Funktion 2 sind das P(-5π;-5 π) & P(5 π;5 π).
Im Falle von Funktion 3 sind das P(-2.0;-2.0) & P(2.0;2.0).
Die Werte für Funktion 1 & 2 sind dabei der Aufgabenstellung entnommen, während das Intervall für die dritte Funktion frei gewählt wurde.
Für das Attribut sind ein Setter & ein Getter implementiert.

Alle drei Funktionen implementieren die Methode *get_value*, mit welcher sie den Funktionswert an einer gegebenen Stelle (x- & y-Wert) anhand der oben genannten Gleichungen berechnen und als *Float zurückgeben.*

Weiterhin implementieren sie die Methode *get_gradient*, mit welcher sie den Gradienten an einer gegebenen Stelle (x- & y-Wert) anhand der oben genannten Gleichung berechnen und als Objekt der Klasse *Vector* wiedergeben.

Klasse Point

Diese Klasse verfügt über zwei Attribute vom Typ *Float*: *x_value* & *y_value*. Diese repräsentieren die Position des Punktes im Koordinatensystem. Für beide Attribute sind Setter & Getter implementiert.

Weiterhin verfügt die Klasse über die statische Methode *points_are_in_range.* Dieser werden zwei Objekte der Klasse *Point* übergeben, *point1* & *point2*, sowie ein Objekt vom Typ *Float,* nämlich *distance*. Mit dieser Methode wird überprüft, ob der Abstand der 2 übergebenen Punkte kleiner gleich dem übergebenen Distanz ist. Dazu wird der Abstand mit Hilfe des Satz vom Pythagoras berechnet, das heißt, dass die Wurzel des Abstands der beiden x-Werte im Quadrat plus des Abstands der beiden y-Werte im Quadrat berechnet wird. Ist der berechnete Abstand kleiner gleich als der übergebene Abstand, wird TRUE zurückgeben, andernfalls wird FALSE zurückgegeben.

Abschließend verfügt die Klasse über die Methode *apply_vector*. Dieser Methode wird ein Objekt der Klasse *Vector, vector*, und ein Objekt vom Typ *Float*, nämlich *Factor*, übergeben. Dabei werden auf die Werte des Objektes der Klasse *Point*, der diese Methode aufruft, die Werte des übergebenen Vektors aufaddiert. Der Parameter *factor* funktionierte dabei als Multiplikator für den Vector, das heißt, dass er angibt wie weit dem Vektor gefolgt werden soll. Der Rückgabewert ist dabei dann der so neu erreichte Punkt, also somit ein Objekt der Klasse *Point*.

Klasse Vector

Diese Klasse verfügt über zwei Attribute vom Typ *Float*: *x_value* & *y_value*. Diese repräsentieren die Richtung, in welcher der Vektor zeigt. Für beide Attribute sind Getter & Setter implmentiert.

Weiterhin verfügt die Klasse über die Methode *normalize*. Diese verändert die Attribute eines Objekts, sodass die Länge des Vektors gleich 1 ist. Dafür wird zunächst die Länge des Vektors berechnet mit Hilfe des Satz des Pythagoras, das heißt, dass die Wurzel aus dem x-Wert im Quadrat plus dem y-Wert im Quadrat berechnet wird. Dessen Reziproke wird dann mit dem x-Wert & dem y-Wert des Vektors multipliziert, welche mit Hilfe der Getter aufgerufen werden, und anschließend mit Hilfe der Setter gesetzt.

Abschließend verfügt die Klasse über die Methode *negate_values*. Diese ändert die Vorzeichen aller Werte eines Vektors. Dabei werden die Werte mit Hilfe der Getter aufgerufen, dann mit -1 multipliziert & anschließend mit Hilfe der Setter gesetzt.

Main-Datei & Starten des Programms

Im *src*-Ordner befindet sich die Main-Datei des Programms: *__main__.py*. In dieser werden zunächst 3 Konstanten definiert. Die 1. Konstante *STARTING_POINT*, ein Objekt der Klasse *Point*, gibt an, von welchem Punkt, die beiden Verfahren starten sollen. Standartmäßig ist hier der Punkt P(1.0;1.0) gewählt wurde. Möchten Sie von einem anderen Punkt starten, dann müssen Sie dieser Konstante nur andere Werte zuweisen.
Die zweite Konstante *INTERVALS*, ein *Dictionary*, speichert dabei die Intervallsgrenzen der einzelnen Funktionen. Die Schlüssel haben dabei den Typ *Integer*, während die Werte Listen von Objekten der Klasse *Point* sind. Der Schlüssel 1 definiert hierbei die Intervalle für Funktion 1, Schlüssel 2 die Intervalle für Funktion 2 & Schlüssel 3 die Intervalle für Funktion 3.
Die letzte Konstante *EDGE_SEARCH* vom Typ *Boolean* gibt dabei an, welches Verfahren ausgeführt werden soll. Ist die Konstante gleich TRUE, wird das Kanten Suchverfahren ausgeführt, ist sie gleich FALSE, wird das Gradientenabstiegsverfahren ausgeführt. Standardmäßig wird hier das Kanten Suchverfahren ausgewählt. Sollten Sie das Gradientenabstiegsverfahren ausführen wollen, müssen sie den Wert der Variable nur auf FALSE ändern.

Anschließend ist die eigentliche Main-Methode definiert. Zunächst wird ein Objekt einer der drei Funktionen initialisiert. Standardmäßig wird hierbei ein Objekt der Klasse *Function1* initialisiert. Sollten Sie eine andere Funktion untersuchen wollen, müssen Sie hier nur den Konstruktor & den Schlüssel des *Dictionaries INTERVALS* ändern.

Im Anschluss wird je nach Wert von *EDGE_SEARCH* eines der beiden Verfahren ausgeführt, wobei zunächst die restlichen Parameter der Methode initialisiert werden. Ich habe für die Parameter dabei Werte gewählt, die recht genaue Ergebnisse liefern, jedoch können Sie die Parameter beliebig verändern.

Beide Methoden speichern ihren Rückgabewert in der Variable *found_min*, einem Objekt der Klasse *Point*. Am Ende der Main-Methode wird das gefundene Minimum mit seinem Gradienten auf der Konsole ausgegeben.

Das gesamte Programm kann über die Datei *non_linear_serach.bat* gestartet werden.

Laufzeitverhalten & Genauigkeit in Abhängigkeit von n & ε

Beschreibung des Testvorgehens

Im Folgenden wird das Laufzeitverhalten & die Genauigkeit in Abhängigkeit von n & ε betrachtet. Dabei werden nur das Kanten Suchverfahren & das Gradienten Suchverfahren untersucht, da die Fibonacci Methode ein Teil des Kanten Suchverfahrens ist (und da das ε kein Bestandteil des Fibonacci Verfahrens ist).

Das ε gibt dabei an, wie weit der Zielpunkt zwischen zwei aufeinander folgenden Schritten maximal wandern darf um als gleich zu gelten. Im Code wird dies durch den Funktionsparameter *accuracy* repräsentiert.

Das n wiederrum hat je nach Verfahren eine andere Bedeutung:

- Bei dem Kanten Suchverfahren entspricht es der Anzahl an Berechnungen, mit welcher zudem die Anzahl an Intervallen (n+2. Fibonaccizahl) und das Startintervall (n. Fibonaccizahl) des Fibonacci Verfahrens bestimmt werden. Im Code wird dies durch den Funktionsparameter *number_of_steps* repräsentiert.
- Bei dem Gradienten Suchverfahren entspricht es einem Skalar, welches angibt wie weit man vom aktuellen Punkt entlang des Vektors gehen soll. Im Code wird dies durch den Funktionsparameter *factor* repräsentiert.

Die Genauigkeit wird dabei mit Hilfe des Gradienten bestimmt, da eben jener in den Extremstellen gleich dem Nullvektor sein muss.

Weiterhin werden der Lesbarkeit halber alle Werte auf 4 Nachkommastellen gerundet.

Testsystem

- Prozessor: 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 1.38 GHz
- RAM: 8,00 GB
- Betriebssystem: Windows 11 Pro

Duchführung

Kanten Suchverfahren

Funktion 1

Gewähltes n	Gewähltes ε	Startpunkt	Gradient beim Minimum	Laufzeit [s] Messung 1	Laufzeit [s] Messung 2	Laufzeit [s] Messung 3	Laufzeit [s] Durchschnitt
25	1.0	(1.0; 1.0)	(0.0160; -0.0124)	0.649	0.5878	0.5798	0.6056
25	0.1	(1.0; 1.0)	(0.0160; -0.0124)	0.8412	0.8448	0.7975	0.8278
25	0.01	(1.0; 1.0)	(0.0160; -0.0124)	0.8739	0.8773	0.8624	0.8712
25	0.001	(1.0; 1.0)	(0.0160; -0.0124)	0.9088	0.8454	0.8539	0.8694
25	0.0001	(1.0; 1.0)	(0.0160; -0.0124)	0.8154	0.8466	0.8145	0.8255
26	1.0	(1.0; 1.0)	(0.0035; 0.0057)	0.9565	0.9395	0.8779	0.9246
26	0.1	(1.0; 1.0)	(0.0035; 0.0057)	1.3564	1.3619	1.3612	1.3598
26	0.01	(1.0; 1.0)	(0.0035; 0.0057)	1.3904	1.3312	1.3388	1.3535
26	0.001	(1.0; 1.0)	(0.0035; 0.0057)	1.3331	1.3639	1.3939	1.3636
26	0.0001	(1.0; 1.0)	(0.0035; 0.0057)	1.4183	1.364	1.395	1.3925
27	1.0	(1.0; 1.0)	(0.0083; -0.0012)	1.5631	1.5662	1.5799	1.5697
27	0.1	(1.0; 1.0)	(0.0083; -0.0012)	2.3761	2.3637	2.3643	2.368
27	0.01	(1.0; 1.0)	(0.0083; -0.0012)	2.3829	2.3724	2.3644	2.3732
27	0.001	(1.0; 1.0)	(0.0083; -0.0012)	2.4281	2.4967	2.3792	2.4347
27	0.0001	(1.0; 1.0)	(0.0083; -0.0012)	2.3838	2.3562	2.3787	2.3729
28	1.0	(1.0; 1.0)	(0.0065; 0.0014)	2.7499	2.5969	2.6286	2.6584
28	0.1	(1.0; 1.0)	(0.0065; 0.0014)	3.9425	4.047	3.9081	3.9659
28	0.01	(1.0; 1.0)	(0.0065; 0.0014)	3.9516	4.0411	3.9025	3.9651
28	0.001	(1.0; 1.0)	(0.0065; 0.0014)	3.9734	3.9009	4.0219	3.9654
28	0.0001	(1.0; 1.0)	(0.0065; 0.0014)	4.2906	3.9512	3.9311	4.0576
29	1.0	(1.0; 1.0)	(0.0072; 0.0004)	4.2774	4.4947	4.1817	4.3179
29	0.1	(1.0; 1.0)	(0.0072; 0.0004)	6.4788	6.406	6.6368	6.5072
29	0.01	(1.0; 1.0)	(0.0072; 0.0004)	6.6015	6.4398	6.4449	6.4954
29	0.001	(1.0; 1.0)	(0.0072; 0.0004)	6.5925	6.5391	6.5959	6.5758
29	0.0001	(1.0; 1.0)	(0.0072; 0.0004)	6.4294	6.6682	6.4071	6.5016

Funktion 2

Gewähltes n	Gewähltes ε	Startpunkt	Gradient beim Minimum	Laufzeit [s] Messung 1	Laufzeit [s] Messung 2	Laufzeit [s] Messung 3	Laufzeit [s] Durchschnitt
25	1.0	(1.0; 1.0)	(-0.0009; <0.0001)	0.635	0.5971	0.6273	0.6198
25	0.1	(1.0; 1.0)	(-0.0009; <0.0001)	0.5921	0.5956	0.5791	0.5889
25	0.01	(1.0; 1.0)	(0.0001; <0.0001)	0.9138	0.9228	0.8295	0.8887
25	0.001	(1.0; 1.0)	(0.0001; <0.0001)	0.8166	0.9549	0.8458	0.8724
25	0.0001	(1.0; 1.0)	(0.0001; <0.0001)	1.2247	1.1668	1.309	1.2335
26	1.0	(1.0; 1.0)	(-0.0008; <0.0001)	1.0768	1.0378	1.0487	1.0544
26	0.1	(1.0; 1.0)	(-0.0008; <0.0001)	0.9621	1.0361	1.2555	1.0846
26	0.01	(1.0; 1.0)	(<0.0001; <0.0001)	1.3968	1.425	1.33	1.384
26	0.001	(1.0; 1.0)	(<0.0001; <0.0001)	1.4584	1.3455	1.4314	1.4117
26	0.0001	(1.0; 1.0)	(<0.0001; <0.0001)	1.8624	1.8793	1.792	1.8445
27	1.0	(1.0; 1.0)	(-0.0008; <0.0001)	1.5628	1.5975	1.6521	1.6041
27	0.1	(1.0; 1.0)	(-0.0008; <0.0001)	1.592	1.5805	1.5802	1.5843
27	0.01	(1.0; 1.0)	(<0.0001; <0.0001)	2.4366	2.3953	2.3327	2.3882
27	0.001	(1.0; 1.0)	(<0.0001; <0.0001)	2.4461	2.4094	2.3485	2.4013
27	0.0001	(1.0; 1.0)	(<0.0001; <0.0001)	3.1741	3.1629	3.1636	3.1669
28	1.0	(1.0; 1.0)	(-0.0008; <0.0001)	2.6111	2.5566	2.6303	2.5993
28	0.1	(1.0; 1.0)	(-0.0008; <0.0001)	2.6138	2.5823	2.6302	2.6088
28	0.01	(1.0; 1.0)	(<0.0001; <0.0001)	3.9368	3.8912	3.8755	3.9012
28	0.001	(1.0; 1.0)	(<0.0001; <0.0001)	3.8856	3.9474	3.913	3.9153
28	0.0001	(1.0; 1.0)	(<0.0001; <0.0001)	5.2942	5.2521	5.1752	5.2405
29	1.0	(1.0; 1.0)	(-0.0008; <0.0001)	4.7955	4.3823	4.2249	4.4676
29	0.1	(1.0; 1.0)	(-0.0008; <0.0001)	4.3021	4.4351	4.3447	4.3606
29	0.01	(1.0; 1.0)	(<0.0001; <0.0001)	6.5823	6.6132	6.571	6.5888
29	0.001	(1.0; 1.0)	(<0.0001; <0.0001)	6.5733	6.5118	6.6104	6.5652
29	0.0001	(1.0; 1.0)	(<0.0001; <0.0001)	8.755	8.8255	8.8181	8.7995

Funktion 3

Gewähltes n	Gewähltes ε	Startpunkt	Gradient beim Minimum	Laufzeit [s] Messung 1	Laufzeit [s] Messung 2	Laufzeit [s] Messung 3	Laufzeit [s] Durchschnitt
25	1.0	(1.0; 1.0)	(<0.0001; <0.0001)	0.7347	0.6828	0.6865	0.7013
25	0.1	(1.0; 1.0)	(<0.0001; <0.0001)	0.5226	0.5202	0.5348	0.5259
25	0.01	(1.0; 1.0)	(<0.0001; <0.0001)	0.542	0.5345	0.5184	0.5316
25	0.001	(1.0; 1.0)	(<0.0001; <0.0001)	0.5283	0.533	0.5177	0.5263
25	0.0001	(1.0; 1.0)	(<0.0001; <0.0001)	0.5373	0.5165	0.5332	0.529
26	1.0	(1.0; 1.0)	(<0.0001; <0.0001)	0.924	0.8925	0.9408	0.9191
26	0.1	(1.0; 1.0)	(<0.0001; <0.0001)	0.916	0.9069	0.9088	0.9106
26	0.01	(1.0; 1.0)	(<0.0001; <0.0001)	0.904	0.8929	0.9111	0.9027
26	0.001	(1.0; 1.0)	(<0.0001; <0.0001)	0.904	0.9011	0.9735	0.9262
26	0.0001	(1.0; 1.0)	(<0.0001; <0.0001)	1.0681	1.1826	1.0498	1.1002
27	1.0	(1.0; 1.0)	(<0.0001; <0.0001)	1.7863	1.9055	1.6506	1.7808
27	0.1	(1.0; 1.0)	(<0.0001; <0.0001)	1.6259	1.5659	1.5655	1.5858
27	0.01	(1.0; 1.0)	(<0.0001; <0.0001)	1.5688	1.5346	1.5338	1.5457
27	0.001	(1.0; 1.0)	(<0.0001; <0.0001)	1.5844	1.5277	1.5511	1.5544
27	0.0001	(1.0; 1.0)	(<0.0001; <0.0001)	1.5535	1.5242	1.5655	1.5477
28	1.0	(1.0; 1.0)	(<0.0001; <0.0001)	2.5346	2.6295	2.7242	2.6294
28	0.1	(1.0; 1.0)	(<0.0001; <0.0001)	2.6843	2.6002	2.5827	2.6224
28	0.01	(1.0; 1.0)	(<0.0001; <0.0001)	2.623	2.6166	2.6074	2.6157
28	0.001	(1.0; 1.0)	(<0.0001; <0.0001)	2.5681	2.6058	2.5993	2.5911
28	0.0001	(1.0; 1.0)	(<0.0001; <0.0001)	2.5793	2.6176	2.5993	2.5987
29	1.0	(1.0; 1.0)	(<0.0001; <0.0001)	4.2774	4.2824	4.2989	4.2863
29	0.1	(1.0; 1.0)	(<0.0001; <0.0001)	4.2649	4.2901	4.1956	4.2502
29	0.01	(1.0; 1.0)	(<0.0001; <0.0001)	4.369	4.3162	4.241	4.3087
29	0.001	(1.0; 1.0)	(<0.0001; <0.0001)	4.3266	4.225	4.2777	4.2764
29	0.0001	(1.0; 1.0)	(<0.0001; <0.0001)	4.2029	4.2822	4.2253	4.2368

Gradientensuchverfahren

Funktion 1

Gewähltes n	Gewähltes ε	Startpunkt	Gradient beim Minimum	Laufzeit [s] Messung 1	Laufzeit [s] Messung 2	Laufzeit [s] Messung 3	Laufzeit [s] Durchschnitt
1.0	0.1	(1.0; 1.0)	(3929.791; -0.0451)	<0.0001	<0.0001	<0.0001	<0.0001
1.0	0.01	(1.0; 1.0)	(40.104; <0.0001)	0.001	0.001	0.001	0.001
1.0	0.001	(1.0; 1.0)	(40.104; <0.0001	0.009	0.015	0.011	0.0117
1.0	0.0001	(1.0; 1.0)	(0.549; <0.0001)	0.1261	0.1124	0.1124	0.117
1.0	0.00001	(1.0; 1.0)	(0.0011; <0.0001)	1.7328	1.69	1.7184	1.7137
1.0	0.000001	(1.0; 1.0)	(0.0011; <0.0001)	17.0063	16.9205	16.9589	16.9619
1.0	0.0000001	(1.0; 1.0)	(0.0002; <0.0001)	239.9743	249.6653	249.6624	246.434
0.1	0.1	(1.0; 1.0)	(-186.8412; 2147.64)	<0.0001	<0.0001	<0.0001	<0.0001
0.1	0.01	(1.0; 1.0)	(0.6599; <0.0001)	<0.0001	0.0011	0.001	0.0007
0.1	0.001	(1.0; 1.0)	(0.6599; <0.0001)	0.007	0.0071	0.0079	0.0073
0.1	0.0001	(1.0; 1.0)	(-6.2803; <0.0001)	0.0984	0.0985	0.0981	0.0983
0.1	0.00001	(1.0; 1.0)	(-0.0269; <0.0001)	1.5614	1.5674	1.5499	1.5596
0.1	0.000001	(1.0; 1.0)	(-0.0269; <0.0001)	15.6832	15.6133	15.6516	15.6494
0.1	0.0000001	(1.0; 1.0)	(<0.0001; <0.0001)	219.2823	218.7046	279.2861	239.091
0.01	0.1	(1.0; 1.0)	(-18.7196; 2159.8742)	<0.0001	<0.0001	<0.0001	<0.0001
0.01	0.01	(1.0; 1.0)	(-56.1203; 2159.3589)	<0.0001	<0.0001	<0.0001	<0.0001
0.01	0.001	(1.0; 1.0)	(39.8595; <0.0001)	0.0041	0.0039	0.004	0.004
0.01	0.0001	(1.0; 1.0)	(0.3069; <0.0001)	0.091	0.0772	0.0859	0.0847
0.01	0.00001	(1.0; 1.0)	(0.0677<0.0001)	1.4345	1.3649	1.409	1.4028
0.01	0.000001	(1.0; 1.0)	(0.0677; <0.0001)	14.4314	14.2351	14.2391	17.3019
0.01	0.0000001	(1.0; 1.0)	(0.0004; <0.0001)	199.0551	200.2424	197.9131	199.0702

Funktion 2

Gewähltes n	Gewähltes ε	Startpunkt	Gradient beim Minimum	Laufzeit [s] Messung 1	Laufzeit [s] Messung 2	Laufzeit [s] Messung 3	Laufzeit [s] Durchschnitt
1.0	0.1	(1.0; 1.0)	(-1.1095; -0.1122)	0.001	<0.0001	<0.0001	0.0003
1.0	0.01	(1.0; 1.0)	(-0.0816; -0.0082)	0.001	<0.0001	0.001	0.0007
1.0	0.001	(1.0; 1.0)	(0.0045; 0.0005)	0.009	0.0099	0.01	0.0096
1.0	0.0001	(1.0; 1.0)	(-0.0014; -0.0001)	0.1159	0.1194	0.1159	0.1171
1.0	0.00001	(1.0; 1.0)	(-0.0001; <0.0001)	1.4318	1.4379	1.4283	1.4326
1.0	0.000001	(1.0; 1.0)	(<0.0001; <0.0001)	17.0825	18.8082	18.8817	18.2575
1.0	0.0000001	(1.0; 1.0)	(<0.0001; <0.0001)	219.8967	219.4121	219.3676	219.5588
0.1	0.1	(1.0; 1.0)	(1.8202; 0.9996)	<0.0001	<0.0001	<0.0001	<0.0001
0.1	0.01	(1.0; 1.0)	(0.0905; 0.0091)	<0.0001	0.0011	<0.0001	0.0004
0.1	0.001	(1.0; 1.0)	(0.0058; 0.0006)	0.007	0.0059	0.007	0.0066
0.1	0.0001	(1.0; 1.0)	(0.0014; 0.0001)	0.1031	0.096	0.095	0.098
0.1	0.00001	(1.0; 1.0)	(-0.0001; <0.0001)	1.2643	1.2478	1.2588	1.257
0.1	0.000001	(1.0; 1.0)	(<0.0001; <0.0001)	15.835	15.6579	15.9139	15.8022
0.1	0.0000001	(1.0; 1.0)	(<0.0001; <0.0001)	187.9126	188.3239	188.5215	188.2526
0.01	0.1	(1.0; 1.0)	(1.9809; 0.9994)	<0.0001	<0.0001	<0.0001	<0.0001
0.01	0.01	(1.0; 1.0)	(1.9809; 0.9994)	<0.0001	<0.0001	<0.0001	<0.0001
0.01	0.001	(1.0; 1.0)	(0.0099; 0.001)	0.0036	0.004	0.003	0.0035
0.01	0.0001	(1.0; 1.0)	(-0.0003; <0.0001)	0.0713	0.068	0.0664	0.0686
0.01	0.00001	(1.0; 1.0)	(0.0001; <0.0001)	0.9615	0.9332	0.9431	0.9459
0.01	0.000001	(1.0; 1.0)	(<0.0001; <0.0001)	12.584	12.5225	12.564	12.5568
0.01	0.0000001	(1.0; 1.0)	(<0.0001; <0.0001)	157.872	157.4328	157.3782	157.561

Funktion 3

Gewähltes n	Gewähltes ε	Startpunkt	Gradient beim Minimum	Laufzeit [s] Messung 1	Laufzeit [s] Messung 2	Laufzeit [s] Messung 3	Laufzeit [s] Durchschnitt
1.0	0.1	(1.0; 1.0)	(-0.0746; -0.0186)	<0.0001	<0.0001	<0.0001	<0.0001
1.0	0.01	(1.0; 1.0)	(-0.0164; -0.0041)	0.001	<0.0001	0.001	0.0007
1.0	0.001	(1.0; 1.0)	(0.0011; 0.0003)	0.007	0.006	0.01	0.0077
1.0	0.0001	(1.0; 1.0)	(-0.0007; -0.0002)	0.09	0.0874	0.0863	0.0879
1.0	0.00001	(1.0; 1.0)	(<0.0001; <0.0001)	1.0983	1.0658	1.0673	1.0771
1.0	0.000001	(1.0; 1.0)	(<0.0001; <0.0001)	13.4351	12.9015	13.0086	13.1151
1.0	0.0000001	(1.0; 1.0)	(<0.0001; <0.0001)	150.5252	152.2264	166.2961	156.3492
0.1	0.1	(1.0; 1.0)	(2.6418; 0.6605)	<0.0001	<0.0001	<0.0001	<0.0001
0.1	0.01	(1.0; 1.0)	(0.0011; 0.0003)	0.001	0.001	0.0	0.0007
0.1	0.001	(1.0; 1.0)	(0.0011; 0.0003)	0.0046	0.004	0.0059	0.0049
0.1	0.0001	(1.0; 1.0)	(-0.0007; -0.0002)	0.0795	0.074	0.072	0.0752
0.1	0.00001	(1.0; 1.0)	(<0.0001; <0.0001)	0.9641	0.9526	0.9501	0.9556
0.1	0.000001	(1.0; 1.0)	(<0.0001; <0.0001)	11.9791	11.9126	12.0053	11.9657
0.1	0.0000001	(1.0; 1.0)	(<0.0001; <0.0001)	142.8095	143.4152	143.8002	143.3416
0.01	0.1	(1.0; 1.0)	(3.9806; 0.9951)	<0.0001	<0.0001	<0.0001	<0.0001
0.01	0.01	(1.0; 1.0)	(2.0597; 0.5149)	<0.0001	0.0006	<0.0001	0.0002
0.01	0.001	(1.0; 1.0)	(0.0011; 0.0003)	0.0017	0.003	0.002	0.0022
0.01	0.0001	(1.0; 1.0)	(-0.0007; -0.0002)	0.0531	0.048	0.048	0.0497
0.01	0.00001	(1.0; 1.0)	(<0.0001; <0.0001)	0.7243	0.7153	0.7601	0.7332
0.01	0.000001	(1.0; 1.0)	(<0.0001; <0.0001)	9.5397	9.5221	9.4491	9.5036
0.01	0.0000001	(1.0; 1.0)	(<0.0001; <0.0001)	119.1323	118.6381	119.0036	118.9246

Auswertung

Kanten Suchverfahren

Aus den Messdaten erkennt man, dass die Laufzeit in Abhängigkeit von n ähnlich der Fibonaccifolge wächst. Somit wäre die Laufzeit dieses Algorithmus nach der Landau-Notation quadratisch, bzw. $O(n^2)$. Bei ε wiederrum lässt sich erkennen, dass es für je eine Verkleinerung von einer Nachkommastelle konstant gestiegen ist, womit es nach der Landau-Notation am ehesten zur logarithmischen Laufzeit zählen würde, als O(log(ε)). Zu ε ist aber auch anzumerken, dass es einen Punkt gibt, an welchen es so klein ist, dass es keinen weiteren Einfluss mehr hat, es also weder die Genauigkeit noch die Laufzeit mehr beeinflusst.

Zur Genauigkeit lässt sich sagen, dass bei beiden Parametern ein recht kleiner Wert ausreicht, um ausreichend genaue Ergebnisse zu erzeugen, also Ergebnisse deren Gradient nur in der 4. Oder 5. Nachkommastelle von Nullvektor abweichen.

Gradienten Suchverfahren

Aus dem Messdaten lässt sich erkennen, dass die Laufzeit in Abhängigkeit von ε linear wächst, also dass wenn ε um den Faktor 10 verkleinert wird, die Laufzeit auf das ungefähr Zehnfach steigt. Somit wäre es nach der Landau-Notation eine lineare Laufzeit, bzw. O(ε). Dies macht auch bei dieser Implementation Sinn, da eine Verkleinerung von ε zur Folge hat, dass die Schranke, ab welcher n verkleinert, wird bei einem kleineren ε höher ist. Bei n wiederrum lässt sich erkennen, dass es für je eine Verkleinerung von einer Nachkommastelle konstant gesunken ist, womit es nach der Landau-Notation am ehesten zur logarithmischen Laufzeit zählen würde, als O(log(n)). Dies macht bei dieser Implementation Sinn, da ein kleineres n zur Folge hat, dass weniger Schritte gegangen werden müssen, damit der Abstand zweier aufeinander folgender Punkte kleiner als das ε sind.

Zur Genauigkeit lässt sich sagen, dass das Verfahren recht genau ist, solange ε kleiner ist als n. Sollte ε größer oder gleich n sein, ist der Schritt, den das Verfahren geht so klein, dass der Abstand zweier aufeinander folgender Punkte so klein ist, dass das Verfahren frühzeitig abbricht.