

Politechnika Śląska  
Wydział Matematyki Stosowanej  
Kierunek Informatyka  
Studia stacjonarne I stopnia

Projekt inżynierski

**Algorytm symulowanego wyżarzania -  
zastosowanie w rozwiązywaniu  
zagadnień odwrotnych**

Kierujący projektem:  
*dr inż. Adam Zielonka*

Autor:  
Kamil Kryus

*Gliwice 2019*



Projekt inżynierski:

**Algorytm symulowanego wyżarzania - zastosowanie w rozwiązywaniu zagadnień odwrotnych**

kierujący projektem: dr inż. Adam Zielonka

autor: Kamil Kryus

Podpis autora projektu

.....

Podpis kierującego projektem

.....



## Oświadczenie kierującego projektem inżynierskim

Potwierdzam, że niniejszy projekt został przygotowany pod moim kierunkiem i kwalifikuje się do przedstawienia go w postępowaniu o nadanie tytułu zawodowego: inżynier.

Data

Podpis kierującego projektem

## Oświadczenie autora

Świadomy/a odpowiedzialności karnej oświadczam, że przedkładany projekt inżynierski na temat:

**Algorytm symulowanego wyżarzania - zastosowanie w rozwiązywaniu zagadnień odwrotnych**

został napisany przeze mnie samodzielnie.

Jednocześnie oświadczam, że ww. projekt:

- nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2000 r. Nr 80, poz. 904, z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym, a także nie zawiera danych i informacji, które uzyskałem/am w sposób niedozwolony,
- nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów wyższej uczelni lub tytułów zawodowych.
- nie zawiera fragmentów dokumentów kopiowanych z innych źródeł bez wyraźnego zaznaczenia i podania źródła.

Podpis autora projektu

Kamil Kryus, nr albumu: 246591, .....(podpis:).....

Gliwice, dnia .....



# Spis treści

<b>Wstęp</b>	<b>7</b>
<b>1. Opis</b>	<b>9</b>
1.1. Cel . . . . .	9
<b>2. Opis algorytmu symulowanego wyżarzania</b>	<b>11</b>
2.1. Parametry . . . . .	11
2.2. Kroki algorytmu . . . . .	14
<b>3. Funkcje testowe</b>	<b>15</b>
3.1. Funkcja kwadratowa dwóch parametrów . . . . .	15
3.2. Funkcja Rastrigina . . . . .	16
3.3. Funkcja Rosenbrocka . . . . .	17
3.4. Dobieranie parametrów dla funkcji 5 wymiarowej funkcji Rastrigina . .	19
3.5. Dobieranie parametrów dla funkcji 3 wymiarowej funkcji Rastrigina . .	25
3.6. Parametry dobrane dla funkcji kwadratowej dwóch parametrów . . . .	26
3.7. Parametry dobrane dla funkcji Rosenbrocka . . . . .	26
<b>4. Implementacja</b>	<b>27</b>
4.1. Używane parametry i zmienne . . . . .	27
4.2. Schemat blokowy . . . . .	28
4.3. Implementacja metod i algorytmu . . . . .	30
<b>5. Zastosowanie algorytmu w rozwiązywaniu odwrotnego zagadnienia     przewodnictwa ciepła</b>	<b>35</b>
5.1. Tradycyjny problem . . . . .	35
5.1.1. Parametry . . . . .	35
5.1.2. Obliczanie rozkładu temperatur . . . . .	36
5.2. Problem odwrotny . . . . .	37
5.3. Wykorzystanie algorytmu heurystycznego . . . . .	38
5.3.1. Implementacja rozwiązania problemu odwrotnego . . . . .	39

5.3.2. Parametry algorytmu . . . . .	40
5.4. Rezultaty . . . . .	40
5.4.1. Dokładne pomiary temperatur . . . . .	41
5.4.2. 1% błąd pomiarowy temperatur . . . . .	43
5.4.3. 2% błąd pomiarowy temperatur . . . . .	44
5.4.4. 5% błąd pomiarowy temperatur . . . . .	46
<b>6. Narzędzia i technologie</b>	<b>49</b>
6.1. Metodyka pracy . . . . .	49
6.1.1. System kontroli wersji . . . . .	49
6.1.2. Github Project Management . . . . .	49
6.1.3. Środowisko programistyczne . . . . .	49
6.1.4. Mathematica . . . . .	50
6.2. Użyte technologie . . . . .	50
6.2.1. C# . . . . .	50
6.2.2. Wolfram Language . . . . .	50
<b>7. Podsumowanie</b>	<b>51</b>
<b>Literatura</b>	<b>53</b>



# Wstęp

Z problematyką wyznaczania optymalnego rozwiązania mamy do czynienia w wielu dziedzinach życia i nauki, np. minimalizując koszty inwestycji, maksymalizując zyski albo szukając najkrótszego połączenia pomiędzy miastami. Szukając rozwiązania, zawsze dążymy żeby było ono jak „najlepsze” (lub dokładniej „najlepszego”) i zostało znalezione w rozsądnym czasie. W tym celu często korzysta się z algorytmów heurystycznych.

Metody heurystyczne znacznie skracają czas wyszukiwania rozwiązania problemu, aczkolwiek często ich wynikiem jest jedynie wynik przybliżony do „najlepszego”. Otrzymując takie właśnie rozwiązanie możemy:

1. zaakceptować je, np. gdy dokładne rozwiązanie nie jest konieczne (np. kompresja obrazu),
2. zawęzić (istotnie) zakres i prowadzić dalsze poszukiwania.

Jednak aby metodę heurystyczną uznać za akceptowalną, chcemy żeby spełniała następujące wymagania:

- optymalne rozwiązanie jest możliwe do znalezienia z rozsądnym wysiłkiem obliczeniowym,
- otrzymane rozwiązanie powinno być bliskie optymalnemu,
- prawdopodobieństwo uzyskania złego rozwiązania powinno być „niskie”.



# 1. Opis

Często w naukach technicznych możemy natrafić na zadania, które polegają na odtworzeniu niektórych parametrów modelu na podstawie danych będących wynikiem pewnych obserwacji. W odróżnieniu od tradycyjnych problemów, gdzie zaczynając od modelu i danych dochodzimy do rezultatów, w tego typu problemach dzieje się to odwrotnie. Tego typu problemy nazywa się problemami odwrotnymi.

Problemy odwrotne niestety są często źle postawione. Problemy, aby być zagadnieniami poprawnie postawionymi, muszą spełniać 3 wymagania:

1. rozwiązanie problemu musi istnieć,
2. każde rozwiązanie jest unikalne,
3. rozwiązanie zależy od danych oraz parametrów (np. małe zmiany w funkcjach wejścia powodują małe zmiany w rozwiązaniu).

Jednym z tego typów problemów (problemów odwrotnych) są odwrotne zagadnienia przewodnictwa ciepła. Przy posiadaniu niekompletnego opisu modelu matematycznego oraz funkcji opisującej rozkład temperatury w wybranych punktach, zadanie polega na rekonstrukcji niektórych brakujących parametrów modelu.

## 1.1. Cel

W pracy opisany i zaimplementowany zostanie algorytm symulowanego wyznaczania. Dla wybranych funkcji testowych zostały dobrane jego parametry, a finalnie zostanie on wykorzystany do rozwiązywania odwrotnego zadania przewodnictwa ciepła.

W tym celu została stworzona w miarę możliwości uniwersalna aplikacja, która pozwala na znalezienie globalnego minimum kilku funkcji testowych wraz z zadanymi przez siebie parametrami oraz odtworzenie jednego z brakujących parametrów modelu matematycznego zadanego odwrotnego zadania przewodnictwa ciepła, przy podanym rozkładzie temperatur w wybranym punkcie.



## 2. Opis algorytmu symulowanego wyżarzania

Algorytm ten został stworzony wzorując się na zjawisku wyżarzania w metalurgii, które polega na nagrzaniu elementu stalowego do odpowiedniej temperatury, przetrzymaniu go w tej temperaturze przez pewien czas, a następnie powolnym jego schłodzeniu. Sam algorytm natomiast bazuje na metodach Monte-Carlo i w pewnym sensie może być rozważany jako algorytm iteracyjny.

Główną istotą i zarazem zaletą tego algorytmu jest wykonywanie pewnych losowych przeskoków do sąsiednich rozwiązań, dzięki czemu jest w stanie uniknąć wpadania w lokalne minimum. Algorytm ten najczęściej jest używany do rozwiązywania problemów kombinatorycznych, jak np. problemu komiwojażera.

### 2.1. Parametry

#### Początkowa konfiguracja

W tym kroku powinniśmy zainicjalizować naszą temperaturę pewną wartością oraz znaleźć początkowe, losowe rozwiązanie naszego problemu.

#### Temperatura

Temperatura jest zarówno czynnikiem iteracyjnym, jak i jest związana z funkcją prawdopodobieństwa zamiany „gorszego” rozwiązania na „lepsze”. Zatem zakres temperatury powinien być taki, aby na początku działania naszego algorytmu dawał dużą możliwość zamian, a wraz z postępem procesu iteracyjnego te prawdopodobieństwo zamiany było bliskie zeru.

#### Końcowa temperatura

Jest to „bardzo niska” wartość. Temperatura osiągając taki poziom stanowi, iż proces wyżarzania się zakończył i rozwiązanie zostało znalezione. Wartość ta powinna być na tyle mała, by temperatura będąc niewiele większa prowadziła do bardzo niskiego prawdopodobieństwa, a jednocześnie nie wymagało to zbyt dużej ilości iteracji.

#### Powtarzanie pewną ilość razy dla zadanej temperatury

Wartość ta powinna być z góry ustalona i powinna dać nam możliwość sprawdzenia wielu sąsiadów obecnego rozwiązania, równocześnie nie powodując zbyt dużego obciążenia dla algorytmu. Parametr ten jednak jest najbardziej dostosowywalny ze względu na jego ”niezależność”.

#### Znajdowanie losowego sąsiada poprzedniego rozwiązania

Funkcja ta powinna nam pozwalać przejrzeć jak najszerszy zakres rozwiązań, a jednocześnie pozwolić na przeszukiwanie coraz to bliższych sąsiadów obecnie najlepszego rozwiązania, zatem warto uzależnić tą funkcję od stopnia zaawansowania procesu iteracyjnego.

#### Funkcja kosztu

Poprzez funkcję kosztu rozumiemy różnicę pomiędzy obecnie najlepszym rozwiązaniem, a nowym. Funkcja ta ma dodatkowe zastosowanie przy decydowaniu o zamianie gorszego rozwiązania na lepsze. Przy poszukiwaniu globalnego minimum wartość większa jest gorszym rozwiązaniem, dzięki czemu wynikiem tej funkcji jest zawsze liczba ujemna (przy decydowaniu o zamianie).

### Prawdopodobieństwo zamiany P

Prawdopodobieństwo jest wykorzystywane przy decyzji zamiany nowego i gorszego rozwiązania, z wcześniejszym i lepszym.

Prawdopodobieństwo tej zamiany zależy od funkcji kosztu oraz obecnej temperatury. Prawdopodobieństwo zatem można przedstawić w następujący sposób:

$$P = \exp\left(\frac{\Delta E}{T}\right)$$

gdzie:

$$\Delta E - \text{funkcja kosztu} \tag{1}$$

T - obecna wartość temperatury

Prawdopodobieństwo to wraz ze spadkiem wartości funkcji kosztu maleje (gdyż jest zawsze ujemne), natomiast wyższa wartość temperatury zwiększa to prawdopodobieństwo. Decydując o tym czy powinniśmy zamienić nasze gorsze rozwiązanie z lepszym powinniśmy porównać obliczone prawdopodobieństwo z wartością losową zawierającą się w zakresie [0,1].

### Chłodzenie temperatury

Szybkość chłodzenia temperatury nie powinna być zbyt duża, aby pozwolić algorytmowi na sprawdzenie jak największego zakresu możliwych rozwiązań, a jednocześnie niezbyt wolna, gdyż może to spowodować zbyt wolny spadek prawdopodobieństwa i zbyt częste akceptowanie gorszych (lub dużo gorszych) rozwiązań. W większości opracowań można spotkać ten proces, jako mnożnik temperatury w zakresie [0.8;0.99].

## 2.2. Kroki algorytmu

Algorytm ten można również przedstawić za pomocą listy kroków:

1. Zainicjalizuj początkową konfigurację
2. Dopóki temperatura  $>$  minimum, powtarzaj:
  - (a) Powtórz zadaną ilość razy dla danej temperatury
    - i. Znajdź losowo sąsiada poprzedniego rozwiązania
    - ii. Sprawdź czy rozwiązanie jest lepsze od poprzedniego (funkcja kosztu)
      - A. Jeżeli jest, zamień rozwiązania
      - B. Jeżeli nie jest, zamień rozwiązania z pewnym prawdopodobieństwem  $P$
  - (b) Zmniejsz temperaturę



## 3. Funkcje testowe

### 3.1. Funkcja kwadratowa dwóch parametrów

Jako pierwszą funkcję do testów przyjęliśmy funkcję kwadratową dwóch parametrów następującej postaci:

$$f(x, y) = x^2 + y^2$$

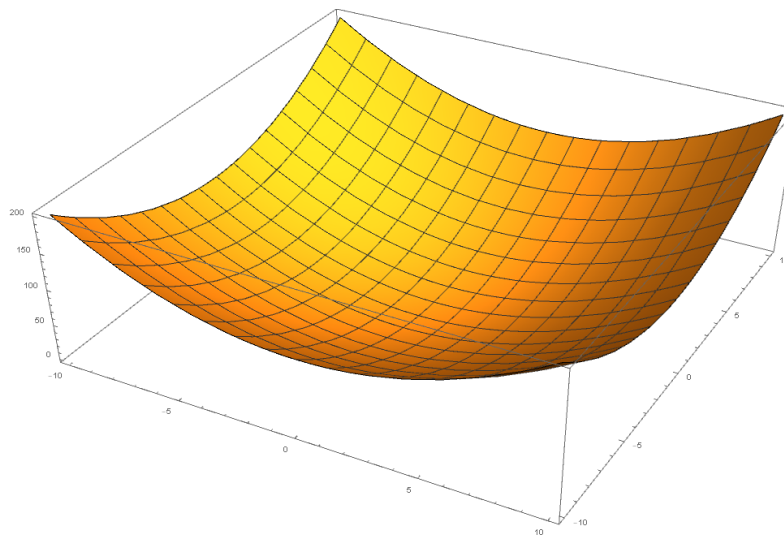
Funkcja ta jest funkcją parzystą i przyjmuje tylko wartości nieujemne. Na potrzeby projektu zakres dla tej funkcji został zawężony następująco:

$$x_i, y_i \in [-10, 10]$$

Posiada ona następujące globalne minimum:

$$f(0, 0) = 0$$

Funkcję prezentuje poniższy wykres:



Rysunek 1: Funkcja kwadratowa dwóch parametrów

### 3.2. Funkcja Rastrigina

Funkcja Rastrigina jest funkcją ciągłą, skalowalną i multimodalną. Dzięki posiadaniu wielu minimum lokalnych, funkcja ta jest często stosowana w testowaniu algorytmów optymalizacyjnych. Przyjmuje ona następującą postać:

$$f(x) = An + \sum_{i=1}^n [x_i^2 - A \cos(2\pi x_i)]$$

gdzie:

$A = 10$ ,

$n$  = ilość wymiarów

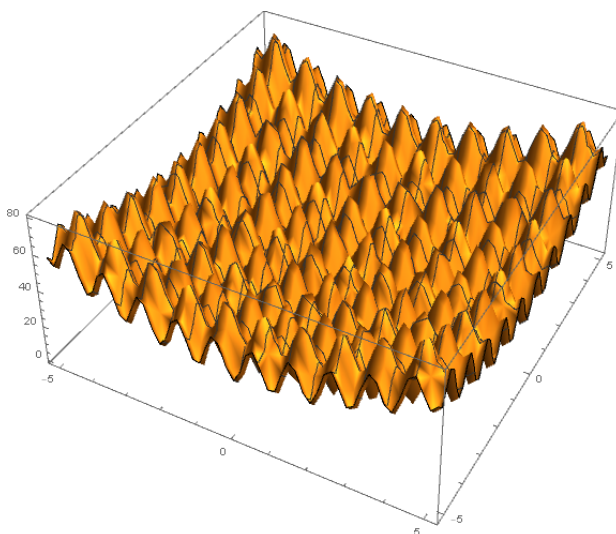
Wartości tej funkcji są nieujemne. Zakres wartości dla tej funkcji znajdziemy w przedziale:

$$x_i \in [-5.12, 5.12]$$

Posiada ona następujące globalne minimum:

$$f(0, \dots, 0) = 0$$

By ujrzeć jej niektóre właściwości zaprezentowaliśmy jej wykres w 2 wymiarach na poniższym obrazku:



Rysunek 2: Funkcja Rastrigina o 2 wymiarach

### 3.3. Funkcja Rosenbrocka

Funkcja ta jest funkcją ciągłą, skalowalną i jednomodalną.

$$f(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2]$$

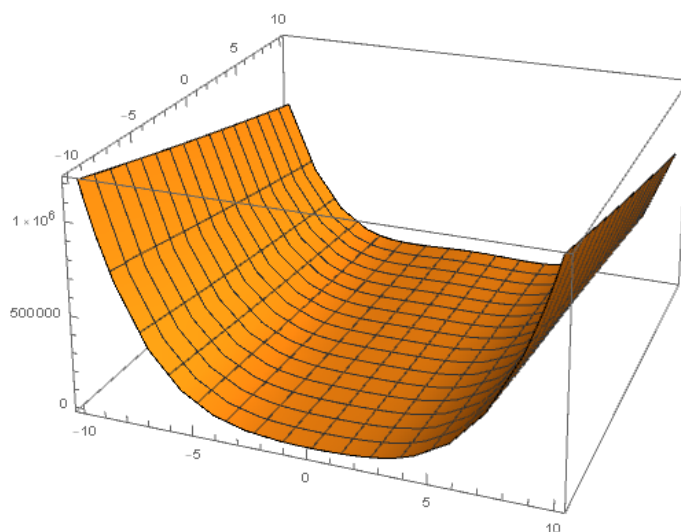
Funkcja ta również przyjmuje wyłącznie wartości nieujemne. Na potrzeby projektu wartości argumentów dla tej funkcji zostały zawężone do poniższego zakresu:

$$x_i \in [-10, 10]$$

Posiada ona następujące globalne minimum:

$$f(1, \dots, 1) = 0$$

Poniższy wykres prezentuje jej wygląd w zadanym zakresie:



Rysunek 3: Funkcja Rosenbrocka o 2 wymiarach

Pomimo, iż algorytmy heurystyczne są dobrym wyborem wszędzie tam, gdzie ważny jest czas znalezienia rozwiązania, to przed skorzystaniem z danego algorytmu jesteśmy zmuszeni ustawić parametry algorytmu w taki sposób, by wynik był dostatecznie dokładny, a algorytm nie wykonywał niepotrzebnie obliczeń, zwłaszcza gdy większa dokładność nie jest nam potrzebna lub nie będzie stanowić większej różnicy w stosunku do już znalezionej wartości. Dodatkową trudność stanowi ilość parametrów oraz to, iż każdy z nich może wpływać w inny sposób na złożoność obliczeniową oraz wynik, oraz parametry mogą być od siebie zależne. W opracowaniach naukowych rzadko kiedy można znaleźć wytyczne co do sposobu znalezienia odpowiednich parametrów do konkretnych problemów.

Starając się trzymać zasad dotyczących tworzenia dobrego algorytmu heurystycznego, przyjęliśmy kilka założeń, a następnie sukcesywnie poszukiwaliśmy odpowiednich wartości dla parametrów by (średni, 10-krotne powtórzenia) wynik był jak najlepszy, starając się zawęzić zakres z czasem. Kiedy (średnie) wyniki były już zadowalające, sprawdzaliśmy jakość dobranych parametrów wykonując 100 razy algorytm z takimi samymi parametrami dla tego samego problemu, uzyskując w prosty sposób procentową jakość algorytmu. W podsekcji "Dobieranie parametrów dla funkcji 5 wymiarowej funkcji Rastrigina" tabele przedstawia stopniowe dojście do parametrów dających zadowalające wyniki, a następnie jakość tych parametrów dla danego problemu. Parametry dla innych problemów zostały zbadane w taki sam sposób i w tych sekcjach zostanie wspomniany jedynie wynik.

### **3.4. Dobieranie parametrów dla funkcji 5 wymiarowej funkcji Rastrigina**

Przed rozpoczęciem testów przyjęto dwa założenia:

1. Końcowa temperatura została ustawiona na stałą wartość równą 0.001,
2. Stopień chłodzenia temperatury został ustawiony na 0.99.

Nr.	T0	Iteracje	Rozwiązanie
1.	500	300	1,09718505292704
2.	500	400	1,29513703551228
3.	100	500	1,29579688714116
4.	300	400	1,39516546905179
5.	100	400	1,59400704106305
6.	400	200	1,59460563439097
7.	400	400	1,59473245683331
8.	300	500	1,69368418646326
9.	100	200	1,69430049805511
10.	400	500	1,69488738622795
11.	200	500	1,79356181491557
12.	500	200	1,89311831496136
13.	300	100	1,89379034166828
14.	200	200	1,89405233085409
15.	400	300	1,99208845088035
16.	500	500	1,99250333315708
17.	300	300	1,99272286399703
18.	200	400	2,09157445324622
19.	100	300	2,19168206783873
20.	300	200	2,19179918169512
21.	500	100	2,19209302230291
22.	200	100	2,19237079270076
23.	400	100	2,19237108578013
24.	200	300	2,29096718181202
25.	100	100	2,98943206230488

Badanie zawarte w tabeli 1 pokazuje, iż parametry na takim poziomie nie mają aż tak dużego znaczenia, jednak można zauważyć, iż większa wartość parametrów prowadzi do nieco lepszych średnich wyników. Badanie skuteczności dla najwyższych parametrów (czyli 500 i 500) wyniosło 12%, co jest bardzo słabym wynikiem.

Nr.	T0	Iteracje	Rozwiązanie
1.	8000	6000	0,400456362305732
2.	10000	4000	0,400942457997776
3.	4000	8000	0,499468816488703
4.	4000	10000	0,499496708877431
5.	6000	6000	0,499881155483216
6.	8000	8000	0,500190776858622
7.	8000	10000	0,500669723808464
8.	2000	6000	0,500802825547873
9.	2000	4000	0,59876018888673
10.	2000	8000	0,598944859155338
11.	10000	8000	0,69839967735632
12.	6000	8000	0,698671405088412
13.	10000	10000	0,698922421188404
14.	4000	6000	0,797834332682657
15.	6000	10000	0,798499451192562
16.	4000	4000	0,798596850008959
17.	2000	10000	0,798896772938739
18.	8000	4000	0,79899550599823
19.	10000	6000	0,898177793387154
20.	6000	4000	0,898733662378301
21.	4000	2000	0,996463628047726
22.	8000	2000	0,997870530143807
23.	6000	2000	1,09703634442711
24.	2000	2000	1,09731890136792
25.	10000	2000	1,39564290923108



10k i 10k 46% 5k - 10k -j 46% 5k i 20k -j 58% 5k i 30k -j 72%

Nr.	T0	Iteracje	Rozwiązanie
1.	aaaaaaa aaaaaaa	aaaaaaa aaaaaaa	oko
2.	aaaaaaa aaaaaaa	aaaaaaa	oko
3.	aaaaaaa	aaaaaaaaaaaaaaa	oko
4.	aaaaaaa	aaaaaaaaaaaaaaa	oko
5.	aaaaaaa	aaaaaaaaaaaaaaa	oko
6.	aaaaaaa	aaaaaaaaaaaaaaa	oko
7.	aaaaaaa	aaaaaaaaaaaaaaa	oko
8.	aaaaaaa	aaaaaaaaaaaaaaa	oko
9.	aaaaaaa	aaaaaaaaaaaaaaa	oko
10.	aaaaaaa	aaaaaaaaaaaaaaa	oko
11.	aaaaaaa	aaaaaaaaaaaaaaa	oko
12.	aaaaaaa	aaaaaaaaaaaaaaa	oko
13.	aaaaaaa aaaaaaa	aaaaaaa aaaaaaa	oko
14.	aaaaaaa aaaaaaa	aaaaaaa	oko
15.	aaaaaaa	aaaaaaaaaaaaaaa	oko
16.	aaaaaaa	aaaaaaaaaaaaaaa	oko
17.	aaaaaaa	aaaaaaaaaaaaaaa	oko
18.	aaaaaaa	aaaaaaaaaaaaaaa	oko
19.	aaaaaaa	aaaaaaaaaaaaaaa	oko
20.	aaaaaaa	aaaaaaaaaaaaaaa	oko
21.	aaaaaaa	aaaaaaaaaaaaaaa	oko
22.	aaaaaaa	aaaaaaaaaaaaaaa	oko
23.	aaaaaaa	aaaaaaaaaaaaaaa	oko
24.	aaaaaaa	aaaaaaaaaaaaaaa	oko
25.	aaaaaaa	aaaaaaaaaaaaaaa	oko
26.	aaaaaaa	aaaaaaaaaaaaaaa	oko
27.	aaaaaaa	aaaaaaaaaaaaaaa	oko
28.	aaaaaaa	aaaaaaaaaaaaaaa	oko
29.	aaaaaaa	aaaaaaaaaaaaaaa	oko
30.	aaaaaaa	aaaaaaaaaaaaaaa	oko
31.	aaaaaaa	aaaaaaaaaaaaaaa	oko
32.	aaaaaaa	aaaaaaaaaaaaaaa	oko
33.	aaaaaaa	aaaaaaaaaaaaaaa	oko

Nr.	T0	Iteracje	Rozwiązanie
1.	aaaaaaa aaaaaaa	aaaaaaa aaaaaaa	oko
2.	aaaaaaa aaaaaaa	aaaaaaa	oko
3.	aaaaaaa	aaaaaaaaaaaaaaa	oko
4.	aaaaaaa	aaaaaaaaaaaaaaa	oko
5.	aaaaaaa	aaaaaaaaaaaaaaa	oko
6.	aaaaaaa	aaaaaaaaaaaaaaa	oko
7.	aaaaaaa	aaaaaaaaaaaaaaa	oko
8.	aaaaaaa	aaaaaaaaaaaaaaa	oko
9.	aaaaaaa	aaaaaaaaaaaaaaa	oko
10.	aaaaaaa	aaaaaaaaaaaaaaa	oko
11.	aaaaaaa	aaaaaaaaaaaaaaa	oko
12.	aaaaaaa	aaaaaaaaaaaaaaa	oko

Nr.	T0	Iteracje	Rozwiązanie
1.	aaaaaaa aaaaaaa	aaaaaaa aaaaaaa	oko
2.	aaaaaaa aaaaaaa	aaaaaaa	oko
3.	aaaaaaa	aaaaaaaaaaaaaaa	oko
4.	aaaaaaa	aaaaaaaaaaaaaaa	oko
5.	aaaaaaa	aaaaaaaaaaaaaaa	oko
6.	aaaaaaa	aaaaaaaaaaaaaaa	oko
7.	aaaaaaa	aaaaaaaaaaaaaaa	oko
8.	aaaaaaa	aaaaaaaaaaaaaaa	oko
9.	aaaaaaa	aaaaaaaaaaaaaaa	oko
10.	aaaaaaa	aaaaaaaaaaaaaaa	oko
11.	aaaaaaa	aaaaaaaaaaaaaaa	oko
12.	aaaaaaa	aaaaaaaaaaaaaaa	oko

Ostatecznie wybrane parametry dla tego problemu:

pocz temp

konc temp

cooling

iteracje

Badanie skuteczności

skuteczność:

### 3.5. Dobieranie parametrów dla funkcji 3 wymiarowej funkcji Rastrigina

Ostatecznie wybrane parametry dla tego problemu:

pocz temp

konc temp

cooling

iteracje

skuteczność:

### **3.6. Parametry dobrane dla funkcji kwadratowej dwóch parametrów**

Ostatecznie wybrane parametry dla tego problemu:

pocz temp

konc temp

cooling

iteracje

skuteczność:

### **3.7. Parametry dobrane dla funkcji Rosenbrocka**

Ostatecznie wybrane parametry dla tego problemu:

pocz temp

konc temp

cooling

iteracje

skuteczność:

## 4. Implementacja

Na realizację algorytmu symulowanego wyżarzania składa się implementacja kilku metod, które zostały przedstawione w rozdziale 2.1.2 i będą tutaj omówione i/lub zostanie przedstawiony ich kod.

### 4.1. Używane parametry i zmienne

Wraz z zainicjalizowaniem obiektu symulowanego wyżarzania, ustawianych jest kilka parametrów na wejście, a konkretniej problem do rozwiązania i parametry samego algorytmu. W moim programie nazywane są: **Function**, **Arguments**, **Arguments2**, **Iterations**, **BeginingTemperature**, **EndingTemperature**, **Cooling**, **SatisfactionSolutionValue**.

**Function** jest identyfikatorem referencji do obiektu problemu. Każdy problem musi dziedziczyć po klasie abstrakcyjnej „TestingFunction”, co zapewnia uniwersalność stosowania algorytmu symulowanego wyżarzania oraz zapewnia nasz algorytm, iż implementacja samego problemu będzie posiadać pewne cechy (jak np. jawnie określoną ilość wymiarów).

**Arguments** jest właściwością w postaci tablicy liczb zmiennoprzecinkowych, które zawierają argumenty dla obecnie najlepszego rozwiązania problemu.

**Arguments2** jest również tablicą liczb zmiennoprzecinkowych, jednak przechowuje ona wartości argumentów dla tymczasowego rozwiązania. Jest tego samego rozmiaru, co właściwość **Arguments**.

**Iterations** jest liczbą wewnętrznych iteracji. Tyle razy algorytm będzie szukał sąsiadów najlepszego rozwiązania, zanim obniży temperaturę.

**BeginingTemperature** jest to początkowa wartość temperatury, od której rozpoczyna się proces poszukiwań rozwiązania.

**EndingTemperature** jest liczbą, którą obniżana temperatura (zmienna **temperature**) musi osiągnąć, by zakończyć działanie algorytmu.

**Cooling** to liczba zmiennoprzecinkowa, w każdym globalnym kroku algorytmu zmienna **temperature** jest mnożona przez tą wartość. Jest ona mniejsza od 1, więc temperatura powoli się obniża.

**SatisfactionSolutionValue** jest minimalną liczbą, jaką rozwiązanie musi osiągnąć, aby wynik poszukiwania rozwiązania był dla satysfakcjonujący. Jest to zmienna opcjonalna, algorytm nadal będzie działać, gdy nie poda się jej wartości.

W programie również używam kilku pomocniczych zmiennych:

**temperature** to zmienna, która przetrzymuje obecną liczbę stanowiącą temperaturę. Jest ona używana przy warunkach globalnej iteracji oraz przy funkcji prawdopodobieństwa. Wraz z postępem iteracji maleje.

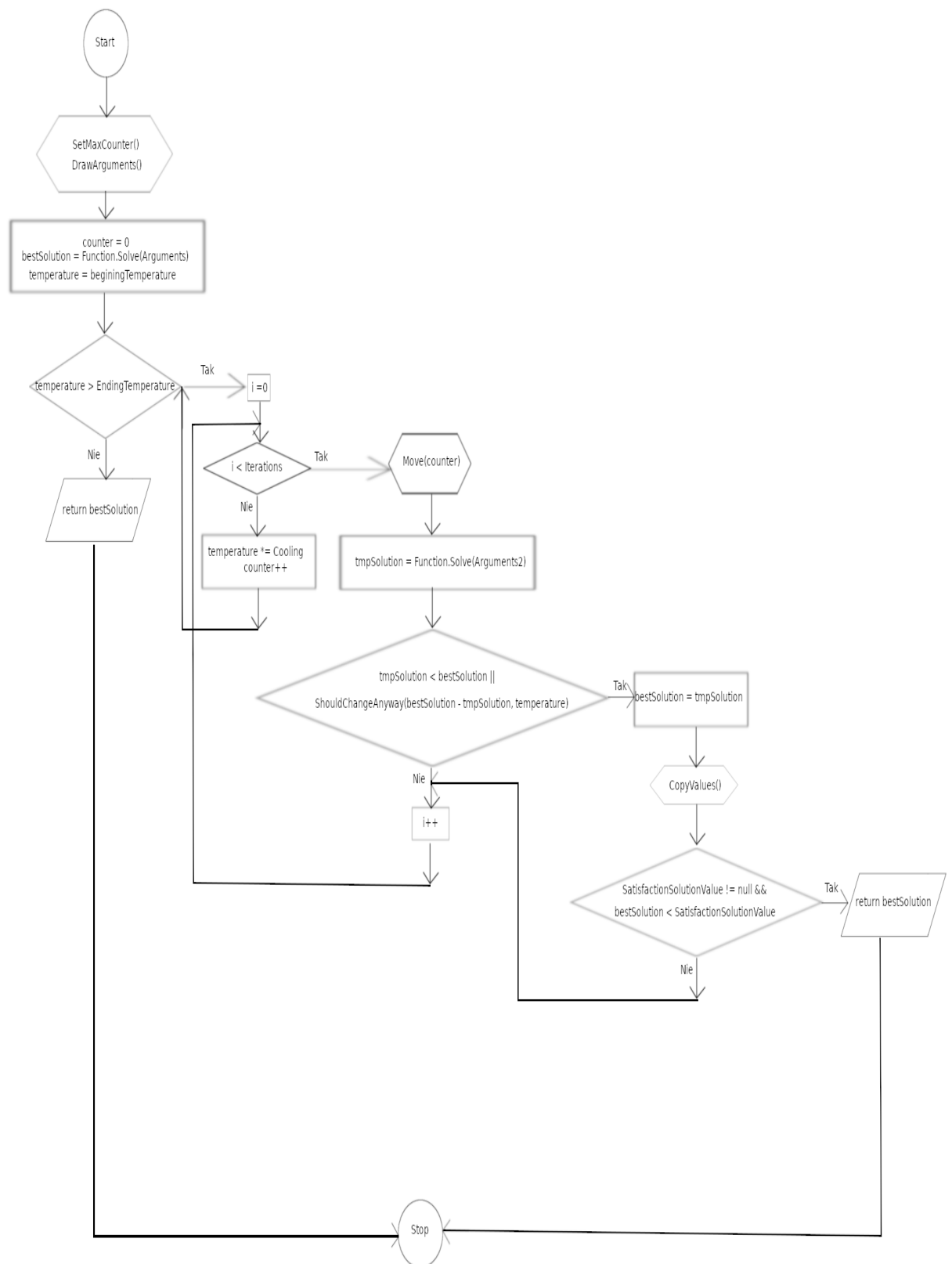
**bestSolution** to liczba zmiennoprzecinkowa, która jest obecnie najlepszym wynikiem rozwiązania. Finalnie będzie ona najlepszym rozwiązaniem całego problemu.

**tmpSolution** jest tymczasowym wynikiem rozwiązania (wynikiem rozwiązania problemu dla parametrów ze zmiennej **Arguments2**).

**counter** jest liczbą oznaczającą obecną, globalną iterację.

## 4.2. Schemat blokowy

Przygotowaliśmy również schemat blokowy reprezentujący poszczególne kroki i scenariusze w procesie poszukiwania rozwiązania zadanego problemu, co można zobaczyć na poniższym rysunku.



Rysunek 4: Schemat blokowy algorytmu symulowanego wyżarzania

### 4.3. Implementacja metod i algorytmu

#### Funkcja SetMaxCounter()

Metoda ta symuluje proces obniżania temperatury w celu obliczenia maksymalnej ilości globalnych iteracji.

```
private void SetMaxCounter()
{
    maxCounter = 0;
    double tmpTemperature = BeginingTemperature;
    while (tmpTemperature > EndingTemperature)
    {
        tmpTemperature *= Cooling;
        maxCounter++;
    }
}
```

#### Funkcja DrawArguments()

W metodzie tej losuję początkowe argumenty (właściwość **Arguments**) z przedziału zadanego w danym problemie.

#### Funkcja Move()

W tym kroku najpierw obliczany jest pozostały procent iteracji do ukończenia procesu. Następnie biorąc 80% pełnej puli możliwych wartości problemu, mnożymy ją przez pozostały procent iteracji (i przypisujemy do zmiennej value). Dalej, w pętli, każdemu argumentowi tymczasowego rozwiązania (właściwość **Arguments2**), przypisywana jest suma odpowiedniego argumentu najlepszego rozwiązania (aby był to sąsiad najlepszego rozwiązania) oraz losowa liczba z przedziału [-value, value]. Wraz z postępem iteracji zakres ten jest coraz węższy, ale rozwiązanie powinno też już być bliskie optymalnemu. Na końcu walidujemy nowy argument do podanego zakresu.

```
private void Move(int counter)
{
    double leftTemperatureCoolingTimes = maxCounter - counter;
```



```
double leftPercent = leftTemperatureCoolingTimes / maxCounter;

double domainValue = (Function.RightBound - Function.LeftBound);
double value = (0.8 * domainValue) * (leftPercent);
for (int i = 0; i < AmountOfArguments; i++)
{
    double newValue = Arguments[i] +
RandomGenerator.Instance.GetRandomDoubleInDomain(-value, value);
    if (newValue < Function.LeftBound)
    {
        newValue = Function.LeftBound;
    }
    if (newValue > Function.RightBound)
    {
        newValue = Function.RightBound;
    }
    Arguments2[i] = newValue;
}
}
```

#### Funkcja ShouldChangeAnyway()

Jest to prosta implementacja funkcji prawdopodobieństwa, o której mowa była w rozdziale 2.1.1.

#### Funkcja CopyValues()

Ze względu, iż język C# traktuje tablicę jako obiekt, to tablica jest typem referencyjnym i konieczne jest skopiowanie wartości ze zmiennej **Arguments2** do zmiennej **Arguments**.

#### Implementacja algorytmu

Opisane metody są wykorzystywane w poszczególnych krokach samego algorytmu. Po obliczeniu maksymalnej ilości iteracji, algorytm losuje pierwsze rozwiązanie. Następnie w pętli i następnej zagnieżdżonej pętli, szuka sąsiada obecnego najlepszego rozwiązania. Zamienia nowe rozwiązanie ze starym jeżeli zostały spełnione odpo-

wiednie warunki. Jeżeli nowe rozwiązanie spełnia kolejny warunek, to kończy program zwracając najlepsze rozwiązanie. Jeżeli nie, wychodzi z zagnieżdżonej pętli, zmniejsza zmienną odpowiedzialną za temperaturę i jest to koniec kroków w jednej pełnej, „globalnej” iteracji. Jeżeli program nie osiągnie satysfakcjonującego rozwiązania, a proces obniżania temperatury zakończy się, zwróci rozwiązanie, które udało mu się znaleźć kończąc tym samym program.

```
public double Solve()
{
    SetMaxCounter();
    int counter = 0;

    DrawArguments();
    double bestSolution = Function.Solve(Arguments);

    double temperature = BeginingTemperature;
    while (temperature > EndingTemperature)
    {
        for (int i = 0; i < Iterations; i++)
        {
            Move(counter);
            double tmpSolution = Function.Solve(Arguments2);

            if (tmpSolution < bestSolution ||
ShouldChangeAnyway(bestSolution - tmpSolution, temperature))
            {
                bestSolution = tmpSolution;
                CopyValues();
                if(SatisfactionSolutionValue != null &&
bestSolution < SatisfactionSolutionValue)
                {
                    return bestSolution;
                }
            }
        }
    }
}
```

```
        }  
        temperature *= Cooling;  
        counter++;  
    }  
    return bestSolution;  
}
```



## 5. Zastosowanie algorytmu w rozwiązywaniu odwrotnego zagadnienia przewodnictwa ciepła

Posiadając gotowy model matematyczny tradycyjnego problemu przewodnictwa ciepła możliwe jest zasymulowanie tego procesu i otrzymanie serii pomiarów temperatur. Obliczone pomiary temperatur można użyć do rozwiązania problemu odwrotnego.

### 5.1. Tradycyjny problem

#### 5.1.1. Parametry

Proces obliczania rozkładu temperatur wymaga podania kilku parametrów, które zostaną teraz wyjaśnione.

Delegaty oznaczone  $f$ ,  $g$  i  $h$  są opisem warunków brzegowych, odpowiednio rozkładem temperatury w czasie  $t_0$  oraz rozkładem temperatur na początku i końcu procesu zależnym od czasu. W projekcie funkcje te przyjmują następującą postać:

$$f(x) = 0.5x^2 + 0.5$$

$$g(t) = t + 0.5$$

$$h(t) = t + 1$$

gdzie:

$$x \in [0, a],$$

$$t \in [0, T],$$

$$a, T = 1.$$

Liczby  $nx$  i  $nt$  są maksymalną liczbą węzłów, na którą dzielimy odpowiednie osie rozkładu temperatur (odpowiednio  $x$  i  $t$ ). W projekcie przyjmują one następujące

wartości:

$$nx = 15$$

$$nt = 480$$

Pozostałe parametry wynikają z równania przewodnictwa ciepła:

c - ciepło właściwe,

rho - gęstość,

lambda - współczynnik przewodności ciepła.

gdzie:

$$c, rho, lambda = 1$$

### 5.1.2. Obliczanie rozkładu temperatur

Następujący fragment kodu przedstawia obliczenie rozkładu temperatur.

```
public double[][] Solve()
{
    double[][] temp = new double[this.nt + 1][];
    for (int i = 0; i < this.nt + 1; i++)
    {
        temp[i] = new double[this.nx + 1];
        for (int j = 0; j < this.nx + 1; j++)
        {
            temp[i][j] = 0;
        }
    }
    double hx = a / nx;
    if (this.tau / (hx * hx) >= 0.5)
    {
        Console.WriteLine("Niestabline");
        return null;
    }
    var x = new double[this.nx + 1];
    for (int i = 0; i < this.nx + 1; i++)
    {
```

```

        x[i] = i * hx;
    }
    for (int i = 0; i < this.nx + 1; i++)
    {
        temp[0][i] = this.f(x[i]);
    }
    for (int i = 0; i < nt + 1; i++)
    {
        temp[i][0] = this.g((i) * this.tau);
        temp[i][this.nx] = this.h((i) * this.tau);
    }
    var wsp = this.lambda * this.tau / (hx * hx * this.c * this.rho);
    for (int j = 1; j < this.nt + 1; j++)
    {
        for (int i = 1; i < this.nx; i++)
        {
            temp[j][i] = wsp * (temp[j - 1][i - 1] - 2.0 * temp[j - 1][i] +
temp[j - 1][i + 1]) + temp[j - 1][i];
        }
    }
    return temp;
}

```

Korzystając z powyższej metody

## 5.2. Problem odwrotny

Posiadając rozkład temperatur, pobraliśmy 10 jej pomiarów w 80 % maksymalnej ilości węzłów w równych odstępach czasu. Następująca tabela przedstawia temperatury wykorzystywane w obliczeniach:

Nr.	$t_i$
1.	0,82
2.	0,92
3.	1,02
4.	1,12
5.	1,22
6.	1,32
7.	1,42
8.	1,52
9.	1,62
10.	1,72

Posiadając model problemu tradycyjnego oraz pobranych pomiarów, zadanie polegało na odtworzeniu jednego z warunków granicznych ( $h$ ) za pomocą równania kwadratowego:

$$h(t) = p^2t + qt + s$$

w taki sposób, by obliczony na nowo rozkład temperatur przy pomocy nowej funkcji i pobrany zestaw danych jak najbardziej przypominał oryginalny. By odtworzona funkcja była równa pierwotnej, parametry powinny przyjąć następujące wartości:

$$p = 0$$

$$q = 1$$

$$s = 1$$

### 5.3. Wykorzystanie algorytmu heurystycznego

Do odnalezienia parametrów równania kwadratowego został użyty algorytm symulowanego wyżarzania. Algorytm w procesie iteracyjnym sprawdzał jakie wartości



## 5. ZASTOSOWANIE ALGORYTMU W ROZWIĄZYWANIU ODWROTNEGO ZAGADNIENIA

$p$ ,  $q$  i  $s$  pozwalały na uzyskanie jak najmniejszego błędu odtworzenia (liczonego jako sumę wartości bezwzględnej różnic odpowiednich pomiarów temperatur), co przedstawia poniższy wzór:

$$\sum_{i=1}^n |x_i - z_i|$$

gdzie:

$x_i$  - oryginalny  $i$ -ty pomiar

$z_i$  - odtworzony  $i$ -ty pomiar

$n$  - liczba pomiarów.

Poszukiwania parametrów zostały ograniczone do następującego przedziału:

$$p, q, s \in [-10, 10]$$

### **5.3.1. Implementacja rozwiązania problemu odwrotnego**

Klasa odpowiadająca za problem odwrotny posiada metodę `Solve()`, która zwraca błąd odtworzenia funkcji. Funkcja ta przyjmuje dowolną ilość parametrów (ze względu na uniwersalność metody `Solve()` w problemach), gdzie tutaj pierwsze trzy argumenty są parametrami funkcji kwadratowej. W implementacji tej metody jest używanych kilka zmiennych/metod, które oznaczają:

*GetInverseProblemTemperatureMeasurements()*, pomocnicza metoda, która zwraca odpowiednie pomiary temperatur dla odtworzonego problemu tradycyjnego (z funkcją  $h$  jako odtworzona funkcja kwadratowa),

*Measurements* - oryginalny zestaw pomiarów.

Implementacja metody `Solve()` ma następującą postać:

```
public override double Solve(params double[] values)
{
    this.p = values[0];
    this.q = values[1];
    this.s = values[2];
```

```

double sum = 0;

double[] tmpMeasurements = GetInverseProblemTemperatureMeasurements();
for (int i = 0; i < Measurements.Length; i++)
{
    sum += Math.Abs(Measurements[i] - tmpMeasurements[i]);
}
return sum;
}

```

Metoda ta jest wykorzystywana przez algorytm symulowanego wyżarzania do obliczania jak „najniższej” wartości błędu odtworzenia i to właśnie algorytm przekazuje wartości parametrów funkcji odtwarzanej.

### 5.3.2. Parametry algorytmu

Ze względu na czas oczekiwania znalezienia pojedynczego optymalnego rozwiązania problemu, ilość prób sprawdzających jakość dobranych parametrów została ograniczona do 5. Poniższa tabela przedstawia dobrane argumenty, które pozwalają na rozwiązanie problemu z satysfakcjonującym wynikiem błędu odtworzenia.

Nr.	$T_0$	$T_{end}$	Iteracje	Chłodzenie
1.	20	0.001	35000	0.99

## 5.4. Rezultaty

Oprócz prób odtworzenia funkcji granicznej dla dokładnych pomiarów, zostały również podjęte próby jej odtworzenia dla pomiarów, które posiadają kolejno 1, 2 i 5% zakres błędu. Każda wartość temperatury została obliczona w następujący sposób dla poszczególnych pomiarów:

$$t_{ip} = rand(t_i - (\frac{t_i p}{100}), t_i + (\frac{t_i p}{100}))$$

gdzie:

$p$  - wartość procentu zakresu błędu,

## 5. ZASTOSOWANIE ALGORYTMU W ROZWIĄZYWANIU ODWROTNEGO ZAGADNIENIA

$t_i$  - oryginalny i-ty pomiar,

$t_{ip}$  - pomiar z p-procentowym zakresem błędu,

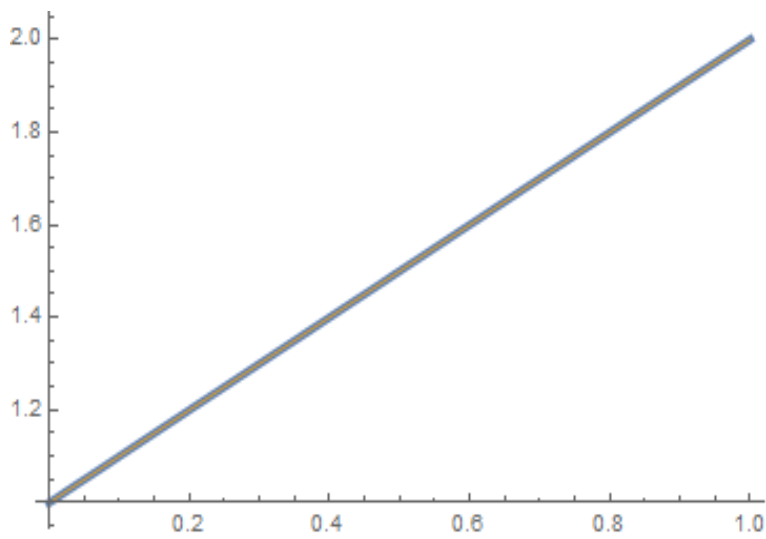
$rand$  - funkcja losująca liczbę z zakresu (od, do).

### 5.4.1. Dokładne pomiary temperatur

Następna tabela przedstawia wyniki osiągnięte przy pomocy tych parametrów (liczby zaokrąglono do 6 miejsca po przecinku) przy niezmiennych pomiarach temperatur.

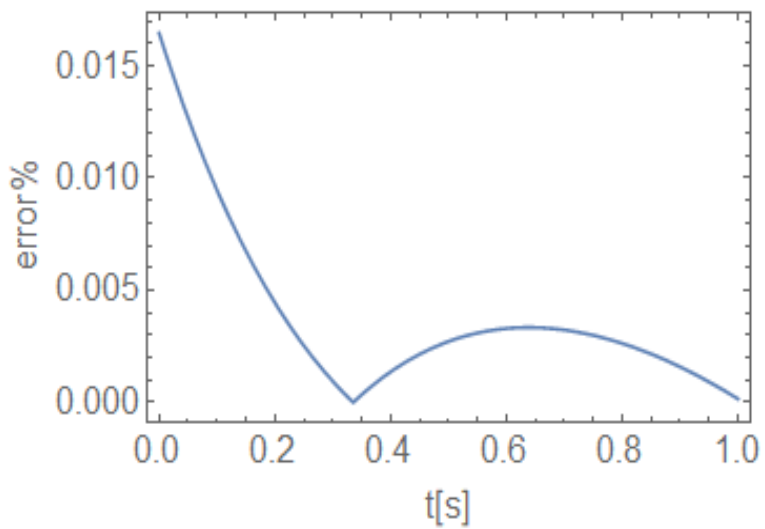
Nr.	Błąd	P	Q	S
1.	0,000694	-0,001192	1,001541	0,999601
2.	0,000944	0,002767	0,997142	1,000632
3.	0,000611	-0,001293	1,001497	0,999587
4.	0,000540	-0,000139	1,000482	0,999830
5.	0,000872	-0,002574	1,002608	0,999527

Różnicę pomiędzy pierwotną funkcją graniczną, a odtworzoną została również przedstawiona na wykresie (żółty kolor oznacza oryginał, niebieska odtworzoną, niebieska posiada większą grubość w celu zauważenia nachodzenia na siebie linii).



Rysunek 5: Porównanie początkowej funkcji granicznej z odtworzoną

Przygotowano również wykres pokazujący błąd bezwzględny obliczeń, co przedstawia poniższy rysunek:



Rysunek 6: Błąd bezwzględny obliczeń

## 5. ZASTOSOWANIE ALGORYTMU W ROZWIĄZYWANIU ODWROTNEGO ZAGADNIENIA

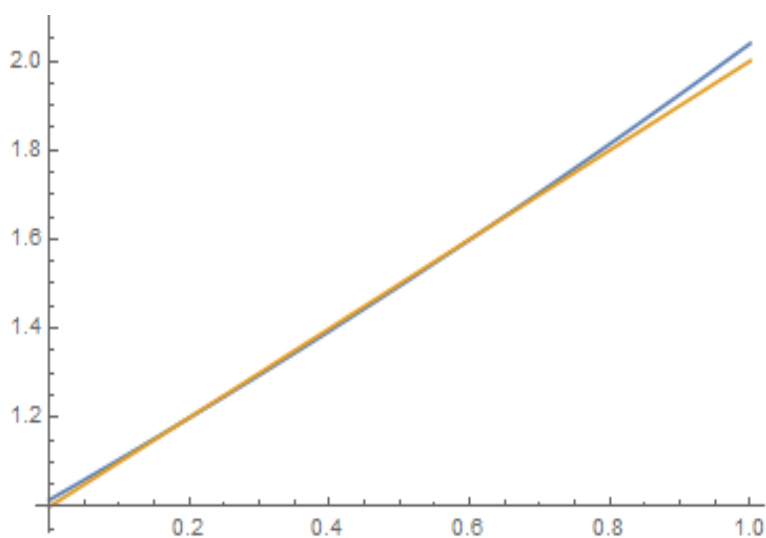
### **5.4.2. 1% błąd pomiarowy temperatur**

Pomiary wykorzystywane przy obliczeniach z 1% zakresem błędu pomiarowego wyglądają następująco:

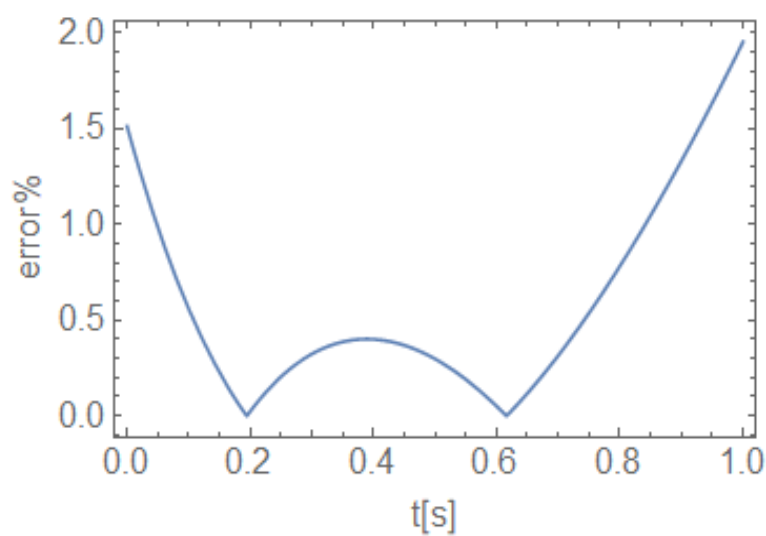
Nr.	$t_i$
1.	0,825351111074887
2.	0,927342643023162
3.	1,02297612208909
4.	1,11725360196924
5.	1,2245631980555
6.	1,31688200037204
7.	1,42067680637374
8.	1,52267106404764
9.	1,62394119003831
10.	1,73631774015795

Przeprowadzenie procesu odtwarzania funkcji granicznej dało następujące rezultaty (zaokrąglone do 6 miejsca po przecinku):

Nr.	Błąd	P	Q	S
1.	0,027209	0,119038	0,895152	1,018938
2.	0,023787	0,115199	0,915740	1,013953
3.	0,027362	0,149960	0,885986	1,006287
4.	0,023056	0,117270	0,888978	1,017797
5.	0,023617	0,130361	0,902080	1,018714



Rysunek 7: Schemat blokowy algorytmu symulowanego wyżarzania



Rysunek 8: Błąd bezwzględny obliczeń

#### 5.4.3. 2% błąd pomiarowy temperatur

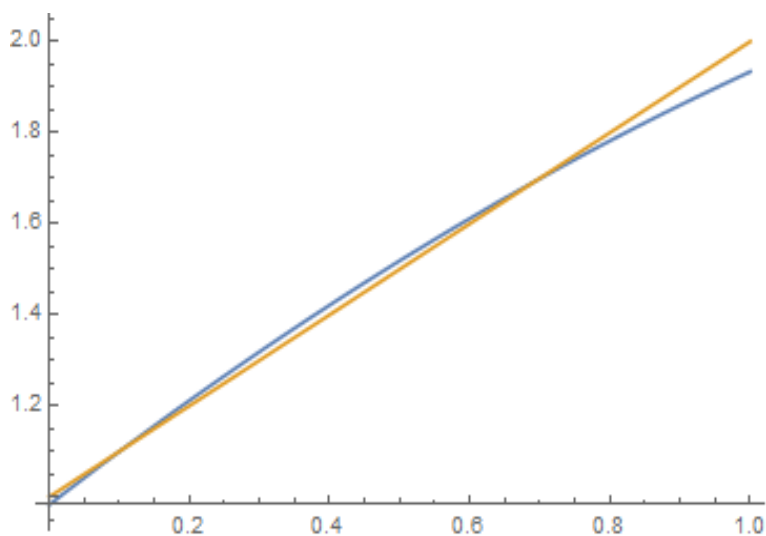
Przy obliczaniu parametrów odtwarzanej funkcji przy 2% zakresie błędu pomiarowego wykorzystano następujące pomiary temperatur:

## 5. ZASTOSOWANIE ALGORYTMU W ROZWIĄZYWANIU ODWROTNEGO ZAGADNIENIA

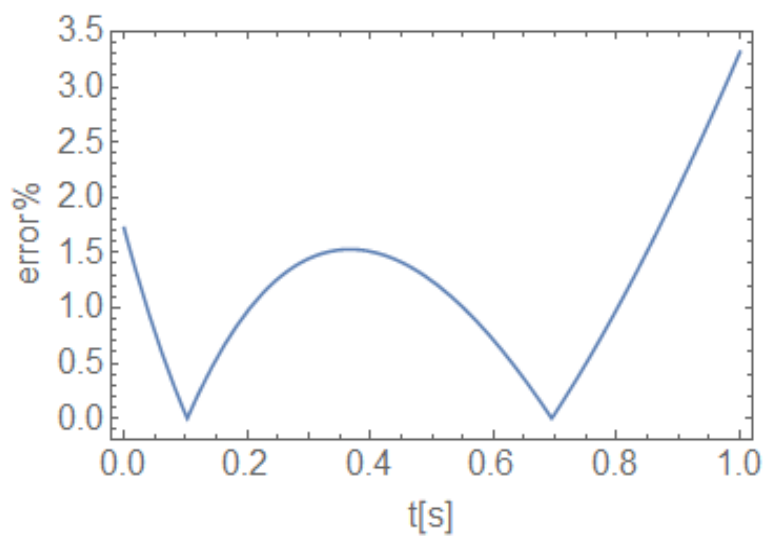
Nr.	$t_i$
1.	0,832676937581728
2.	0,922541154071009
3.	1,01610763695636
4.	1,13368222511601
5.	1,24050689425995
6.	1,33037780534922
7.	1,43757166475485
8.	1,50969160181027
9.	1,61262938854183
10.	1,69925458435307

Obliczone parametry wraz z błędem prezentują się następująco (przy zaokrągleniu do 6 miejsca po przecinku):

Nr.	Błąd	P	Q	S
1.	0,068528	-0,364171	1,299335	0,951341
2.	0,065422	-0,256919	1,197051	0,999223
3.	0,058742	-0,235167	1,185274	0,989484
4.	0,060410	-0,153094	1,117493	0,993703
5.	0,060850	-0,199511	1,164290	0,980086



Rysunek 9: Schemat blokowy algorytmu symulowanego wyżarzania



Rysunek 10: Błąd bezwzględny obliczeń

#### 5.4.4. 5% błąd pomiarowy temperatur

Pomiary wykorzystywane przy obliczeniach z 5% zakresem błędu pomiarowego wyglądają następująco:

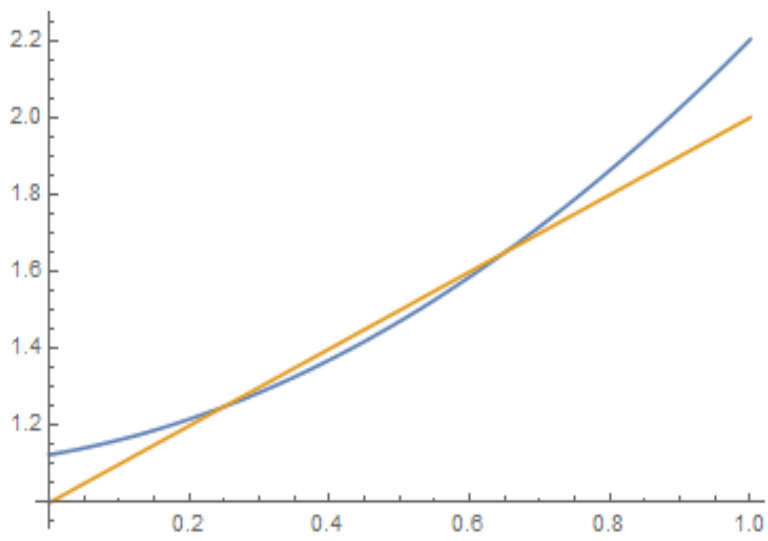


## 5. ZASTOSOWANIE ALGORYTMU W ROZWIĄZYWANIU ODWROTNEGO ZAGADNIENIA

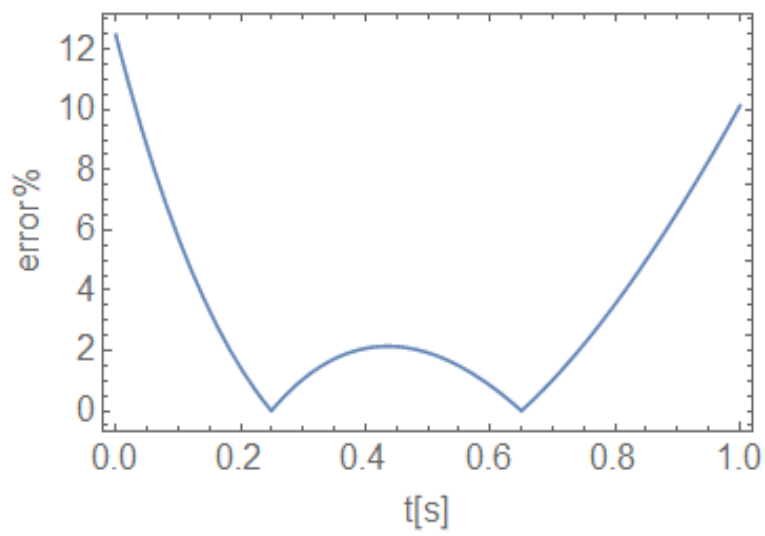
Nr.	$t_i$
1.	0,805057901030433
2.	0,929989068601275
3.	1,05148950317152
4.	1,1398041024412
5.	1,21390826781975
6.	1,29058231760775
7.	1,3879814341464
8.	1,58770452886806
9.	1,62664563802893
10.	1,79591014782987

Wykorzystanie algorytmu dało następujące rezultaty (przy zaokrągleniu liczb do 6 miejsca po przecinku):

Nr.	Błąd	P	Q	S
1.	0,203137	0,867815	0,203345	1,159725
2.	0,204883	0,656280	0,429536	1,098522
3.	0,203023	0,765145	0,316677	1,126484
4.	0,206009	0,813253	0,242778	1,127690
5.	0,204143	0,751570	0,342640	1,111854



Rysunek 11: Schemat blokowy algorytmu symulowanego wyżarzania



Rysunek 12: Błąd bezwzględny obliczeń

## 6. Narzędzia i technologie

W procesie tworzenia aplikacji zdecydowaliśmy się na użycie kilku rozwiązań, które pozwoliły na bezpieczną i przejrzystą pracę w kolejnych jego etapach.

### 6.1. Metodyka pracy

#### 6.1.1. System kontroli wersji

System kontroli wersji posiada wiele zalet, m.in.: bezpieczeństwo, możliwość pracy w kilku miejscach/urzędzeniach nad tym samym problemem, łatwą możliwość przywrócenia poprzedniej wersji, czy wreszcie, inspekcję jakości i poprawności kodu.

W moim projekcie skorzystałem z systemu kontroli Git, a repozytorium można znaleźć na portalu [github.com](https://github.com).

#### 6.1.2. Github Project Management

Pomimo, iż praca w pojedynkę nie wymagała ode mnie zaawansowanego zarządzania projektem i konieczności organizacji pracy, zdecydowałem się na użycie narzędzia pozwalającego na taką pracę. Podzielenie projektu na mniejsze zadania pozwoliło mi wydzielić poszczególne i odrębne sektory pracy, widzieć postępujący progres i łatwo odnaleźć się w aktualnie wykonywanym zadaniu. W tym celu skorzystałem z Github Project Management, który pozwala na proste zarządzanie zadaniami.

#### 6.1.3. Środowisko programistyczne

Do implementacji projektu użyłem środowiska Microsoft Visual Studio Community 2017, które to zostało stworzone przez firmę Microsoft i pozwala na programowanie konsolowe oraz z graficznym interfejsem użytkownika (zarówno aplikacje desktopowe, jak i strony internetowe).

Dobra znajomość i przejrzystość tego środowiska programistycznego pozwoliła mi

skupić się na rozwiązywaniu problemu, omijając problem zapoznawania się z nowym narzędziem.

#### **6.1.4. Mathematica**

Mathematica jest programem opartym na systemie obliczeń symbolicznych oraz numerycznych. Program ten jest dość popularny wśród naukowców ze względu na wiele zalet, jak np. wydajność czy rozpięte możliwości wizualizacji danych. Mathematica jest programem komercyjnym, dlatego stworzenie wykresów do tego projektu oparłem na licencji wydziału Matematyki Stosowanej.

### **6.2. Użyte technologie**

#### **6.2.1. C#**

Język programowania C# należy do obiektowych języków programowania, którego koncepcja opiera się na tworzeniu klas, które poprzez swoją zawartość (m.in. właściwości czy metody) mogą być reprezentowane poprzez obiekty i każde operacje są wykonywane poprzez nie. W projekcie korzystam z języka C# w wersji 7.0, która w momencie rozpoczęcia pracy była aktualna. Dobra znajomość tego języka pozwoliła mi nie zważać na problemy w znajomości składni czy funkcji i skupić się bezpośrednio na implementacji algorytmów, dobraniu odpowiednich parametrów dla poszczególnych funkcji testowych oraz lepszym przetestowaniu całej funkcjonalności.

#### **6.2.2. Wolfram Language**

Język ten służy głównie do programowania obliczeń matematycznych i programowania funkcjonalnego w programie Mathematica. Język ten, wraz z oprogramowaniem Mathematica, pozwalają m.in. na: operacje na macierzach, rozwiązywanie równań różniczkowych czy prezentowanie danych za pomocą wykresów. Z tej ostatniej funkcjonalności skorzystałem tworząc wykresy funkcji testowych.

## 7. Podsumowanie

Celem tej pracy inżynierskiej było stworzenie aplikacji, która pozwoliłaby rozwiązać zadany problem przewodnictwa ciepła poprzez zastosowanie algorytmu symulowanego wyżarzania. Stworzono więc program, który poprzez prosty interfejs graficzny pozwala na użycie tego algorytmu heurystycznego w kilku funkcjach testowych oraz zadany odwrotnym problemie przewodnictwa ciepła i znalezienia ich optymalnego rozwiązania. Dodatkowo użytkownik jest w stanie zmodyfikować (domyślne i sugerowane) parametry algorytmu i sprawdzić jak wpłynie to na rezultat wykonywania programu.

Realizacja założeń projektu wymagała przeprowadzenia badań w kwestii odpowiedniego doboru parametrów dla poszczególnych problemów i ich jakości oraz przetestowania i zoptymalizowania samego działania algorytmu symulowanego wyżarzania. Dobrane parametry dla poszczególnych problemów pozwalają na stosunkowo szybkie i poprawne znalezienie optymalnego rozwiązania danego problemu, a wyniki przeprowadzonych testów rozwiązania odwrotnego zadania przewodnictwa ciepła na zbliżone odtworzenie jednego z brakujących parametrów modelu matematycznego.

Dalsze prace nad programem powinny rozwinąć projekt o możliwość zadania problemu tradycyjnego i odwrotnego przewodnictwa ciepła, stając się tym samym jeszcze bardziej użytecznym. Pomimo, iż algorytmy heurystyczne znacznie przyspieszają znalezienie optymalnego rozwiązania, to jednak bywa to proces czasochłonny, stąd informacja o estymowanym czasie do ukończenia procesie może być użyteczna dla użytkownika. Ze względu na zasoby czasowe i błąd ludzki, warto by było zautomatyzować proces przeprowadzania testów parametrów nowych problemów oraz dla już istniejących. Rozszerzenie projektu o dodatkowe algorytmy heurystyczne pozwoliłoby porównywać ze sobą jakość rozwiązań i złożoność pamięciową i czasową poszczególnych algorytmów, dzięki czemu w praktycznych celach można by dobierać odpowiedni algorytm do konkretnego problemu.



# Literatura

- [1] H. Abiyev and M. Tunay, *Optimization of High-Dimensional Functions through Hypercube Evaluation* [online] Dostępny w Internecie: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4538776/> [dostęp: 13 listopada 2018]
- [2] F. Rothlauf, *Design of Modern Heuristics: Principles and Application* [online], [w:] „Natural Computing Series”, 2011 nr 1, s. 7-44 Dostępny w Internecie: <https://pdfs.semanticscholar.org/b333/0f96d1a937fc2c63b3294729cfea30826134.pdf> [dostęp: 27 listopada 2018]
- [3] R. Mart’ı and G. Reinelt, The Linear Ordering Problem, Exact and Heuristic Methods in Combinatorial Optimization 175, DOI: 10.1007/978-3-642-16729-4 2, c Springer-Verlag Berlin Heidelberg 2011 [dostęp 28 listopada 2018]
- [4] <http://prac.im.pwr.edu.pl/plociniczak/lib/exe/fetch.php?media=odwrotne.pdf>
- [5] <https://www.math.unl.edu/schoen1/8423/wellposed.pdf>