SIMULATED ANNEALING

VÁRDAI ERWIN

CE ESTE RECOACERA SIMULATĂ?

- Simmulated Annealing reprezintă o metodă algoritmică dezvoltată de Kirkpatrick et Al. în 1983, iar metoda sa are la baza algoritmul dezvoltat de Metropolis et Al. În 1953 și reprezint un algoritm pentru a simula răcirea materialului.
- Procesul de răcire a unui metal se numește recoacere.
- Dacă un material solid este încălzit peste punctul său de topire și apoi răcit înapoi la stare solidă, proprietățile structurale ale solidului rezultat după răcire depind de viteza de răcire.
- Kirkpatrick a sugerat că acest tip de simulare ar putea fi folosit pentru a căuta soluții fezabile ale problemelor de optimizare cu scopul de a converge către o soluție optimă.

FUNCȚIA GOLDSTEIN-PRICE

Formula

$$f(x,y) = \left[1 + (x+y+1)^2 \left(19 - 14x + 3x^2 - 14y + 6xy + 3y^2\right)\right]$$

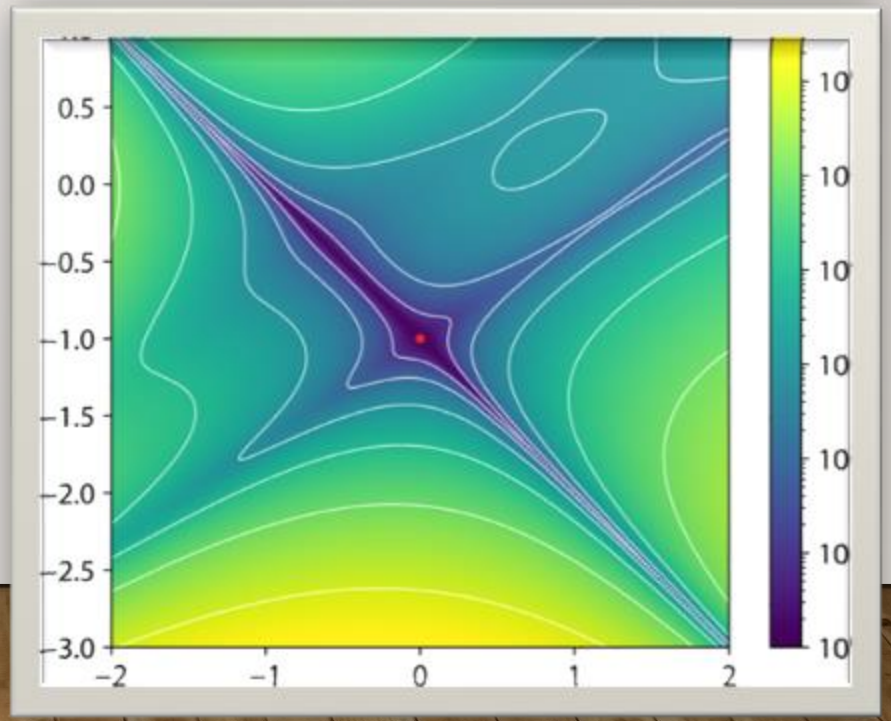
$$\left[30 + (2x-3y)^2 \left(18 - 32x + 12x^2 + 48y - 36xy + 27y^2\right)\right]$$

Minimul global $f(0,-1) = 3$

Domeniul de căutare

$$-2 \leq x, y \leq 2$$

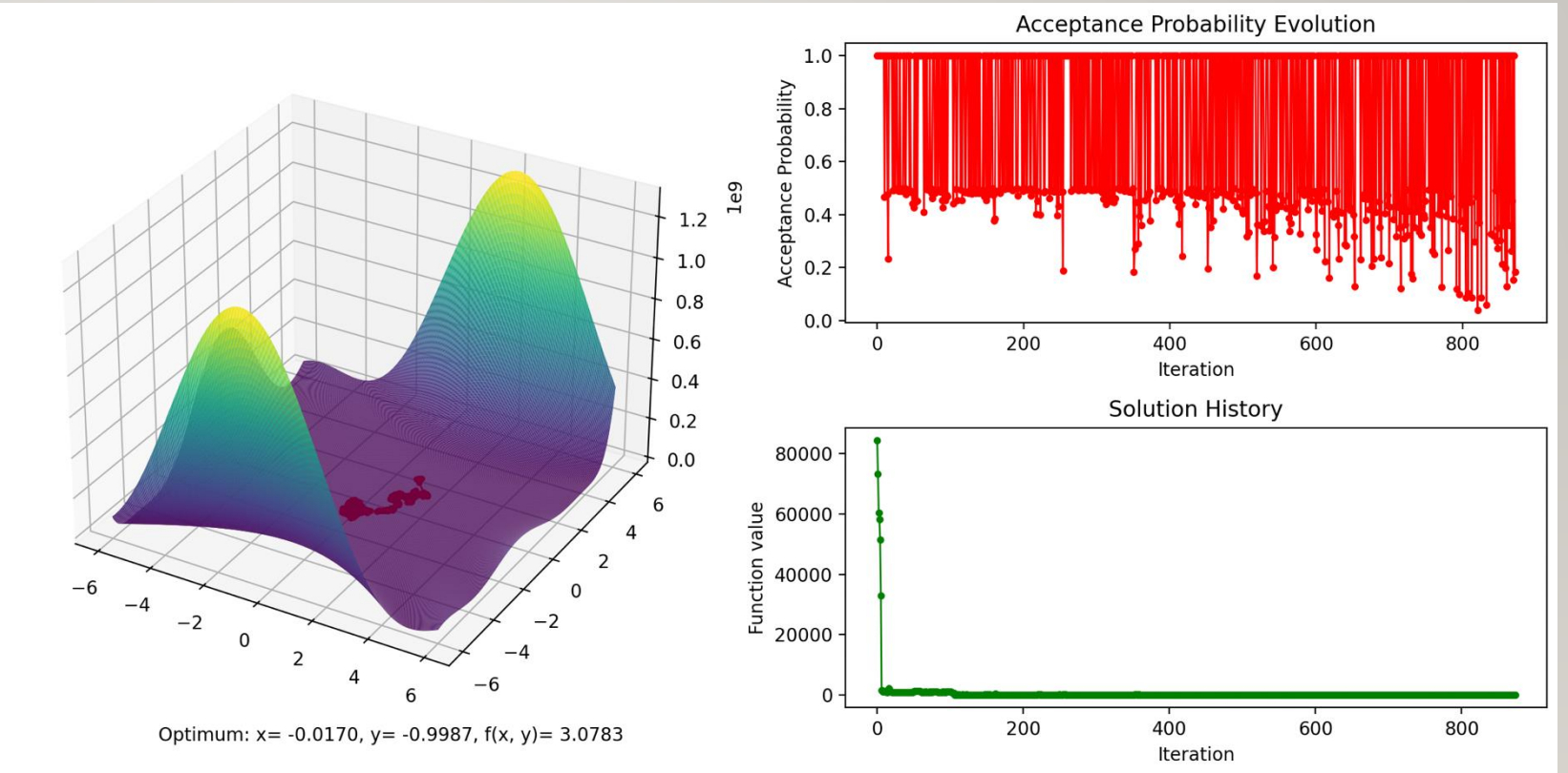
Plot



Parametrii

```python
goldstein_params = {
    'temp': 1000,
    'cooling_rate': 0.95,
    'iterations': 200,
    'local_searches': 10,
    'multiplier': [0.5, 0.1],
    'function_name': 'goldstein',
    'lower_bound_x': -2,
    'upper_bound_x': 2,
    'lower_bound_y': -2,
    'upper_bound_y': 2
}
```

Rezultat



Cu valorile inițiale, soluția obținută este $x = -0.0170$, $y = -0.9987$, iar $f(x, y) = 3.0783$. În continuare, voi analiza evoluția fiecărui parametru pe o serie de 10 pași, selectând la final cea mai optimă valoare obținută. La fiecare pas, voi calcula media rezultatelor obținute din 10 rulări efectuate cu aceiași parametri. Ulterior, voi utiliza aceste valori optimizate pentru a explora în continuare comportamentul funcției Goldstein în raport cu ceilalți parametri.

PAS I – TEMP (**EXEMPLU**)

```
Run 1: x = 0.0158, y = -1.0027, f(x, y) = 3.0764
Run 2: x = -0.0036, y = -1.0084, f(x, y) = 3.0274
Run 3: x = -0.0042, y = -1.0084, f(x, y) = 3.0276
Run 4: x = 0.0035, y = -0.9924, f(x, y) = 3.0224
Run 5: x = 0.0124, y = -0.9920, f(x, y) = 3.0445
Run 6: x = -0.0126, y = -1.0058, f(x, y) = 3.0394
Run 7: x = 0.0005, y = -0.9972, f(x, y) = 3.0032
Run 8: x = 0.0009, y = -0.9930, f(x, y) = 3.0197
Run 9: x = -0.0018, y = -0.9962, f(x, y) = 3.0084
Run 10: x = -0.0016, y = -0.9962, f(x, y) = 3.0081

Final Results:
Average x: 0.0009
Average y: -0.9992
Average f(x, y): 3.0277
```

După rularea algoritmului de 10 ori, rezultatele obținute indică faptul că valorile medii ale parametrilor sunt: $x = 0.0009$, $y = -0.9992$, iar valoarea funcției în punctul optimizat este $f(x, y) = 3.0277$.

Conform exemplului anterior, voi analiza evoluția parametrului **temp** pe următorii 9 pași, crescându-i valoarea de la 1000 până la 5500, în incrementuri de 500 de unități la fiecare pas.

PAS I – TEMP **REZULTAT**

	B	C	D	E	F	G	H	I
1	temp	cooling_rate	iterations	local_searches	multiplier	x	y	f(x, y)
2	1000	0.95	200	10	[0.5, 0.1]	0,0009	-0,9992	3,0277
3	1500	0.95	200	10	[0.5, 0.1]	0,0024	-1,0000	3,0252
4	2000	0.95	200	10	[0.5, 0.1]	-0,0029	-1,0000	3,0669
5	2500	0.95	200	10	[0.5, 0.1]	-0,0016	-0,9968	3,0615
6	3000	0.95	200	10	[0.5, 0.1]	0,0027	-0,9997	3,1063
7	3500	0.95	200	10	[0.5, 0.1]	0,1899	-0,8783	11,2699
8	4000	0.95	200	10	[0.5, 0.1]	0,1781	-0,8808	11,1696
9	4500	0.95	200	10	[0.5, 0.1]	0,1710	-0,8850	11,2154
10	5000	0.95	200	10	[0.5, 0.1]	0,1867	-0,8744	11,2594
11	5500	0.95	200	10	[0.5, 0.1]	0,1805	-0,8744	11,2859

În urma celor 10 x 10 rulări, s-a constatat că valoarea optimă pentru parametrul **temp** este **1500**. Începând de acum, voi utiliza această valoare pentru a optimiza parametrii următori.

PAS 2 - COOLING_RATE

```
goldstein_params = {
    'temp': 1500, # optimizat
    'cooling_rate': 0.95,
    'iterations': 200,
    'local_searches': 10,
    'multiplier': [0.5, 0.1],
    'function_name': 'goldstein',
    'lower_bound_x': -2,
    'upper_bound_x': 2,
    'lower_bound_y': -2,
    'upper_bound_y': 2
}
```

- Cu valorile ”inițiale”, soluția obținută după rezultatele calculate anterior este $x = -0.0024$, $y = -1.000$, iar $f(x, y) = 3.0252$.
- Conform metodologiei precedente, voi analiza evoluția parametrului **cooling_rate** pe parcursul a 9 pași, reducând valoarea acestuia de la **0.95** până la **0.50**, cu decrementări de **0.05** la fiecare pas.

PAS 2 - COOLING_RATE **REZULTAT**

13	1500	0.95	200	10	[0.5, 0.1]	0,0024	-1,0000	3,0252
14	1500	0.90	200	10	[0.5, 0.1]	0,1794	-1,0001	3,0050
15	1500	0.85	200	10	[0.5, 0.1]	0.0001	-1,0005	3,0013
16	1500	0.80	200	10	[0.5, 0.1]	0,0006	-0,9995	3,0027
17	1500	0.75	200	10	[0.5, 0.1]	0,1802	-0,8795	11,1025
18	1500	0.70	200	10	[0.5, 0.1]	0,1795	-0,8799	11,1022
19	1500	0.65	200	10	[0.5, 0.1]	0,1789	-0,8802	11,1023
20	1500	0.60	200	10	[0.5, 0.1]	0,1797	-0,8799	11,1023
21	1500	0.55	200	10	[0.5, 0.1]	0,1797	-0,8799	11,1016
22	1500	0.50	200	10	[0.5, 0.1]	0,1800	-0,8809	11,1020

În urma celor 10 x 10 rulări, s-a constatat că valoarea optimă pentru parametrul **cooling_rate** este **0.85**. Începând de acum, voi utiliza această valoare pentru a optimiza parametrii următori.

PAS 3 - ITERATION

```
goldstein_params = {
    'temp': 1500, # optimizat
    'cooling_rate': 0.85, # optimizat
    'iterations': 200,
    'local_searches': 10,
    'multiplier': [0.5, 0.1],
    'function_name': 'goldstein',
    'lower_bound_x': -2,
    'upper_bound_x': 2,
    'lower_bound_y': -2,
    'upper_bound_y': 2
}
```

- Cu valorile ”inițiale”, soluția obținută după rezultatele calculate anterior este $x = -0.0001$, $y = -1.0005$, iar $f(x, y) = 3.0013$.
- Conform metodologiei precedente, voi analiza evoluția parametrului **iteration** pe parcursul a 9 pași, crescându-i valoarea de la 200 până la 1200, în incrementuri de 100 de unități la fiecare pas.

PAS 3 – ITERARATION **REZULTAT**

23								
24	1500	0.85	200	10	[0.5, 0.1]	0.0001	-1,0005	3,0013
25	1500	0.85	300	10	[0.5, 0.1]	-0,0030	-0,9996	3,0015
26	1500	0.85	400	10	[0.5, 0.1]	-0,0007	-0,9997	3,0014
27	1500	0.85	500	10	[0.5, 0.1]	-0,0005	-1,0005	3,0019
28	1500	0.85	600	10	[0.5, 0.1]	0,0000	-0,9996	3,0007
29	1500	0.85	700	10	[0.5, 0.1]	0,0005	-0,9999	3,0006
30	1500	0.85	800	10	[0.5, 0.1]	-0,0002	-1,0000	3,0004
31	1500	0.85	900	10	[0.5, 0.1]	0,1802	-0,8807	11,1010
32	1500	0.85	1000	10	[0.5, 0.1]	0,1795	-0,8801	11,1004
33	1500	0.85	1100	10	[0.5, 0.1]	-0,0004	-1,0001	3,0006

În urma celor 10 x 10 rulări, s-a constatat că valoarea optimă pentru parametrul **iteration** este **800**. De acum înainte, voi utiliza această valoare pentru a optimiza parametrii următori.

PAS 4 – LOCAL_SEARCHES

```
goldstein_params = {
    'temp': 1500, # optimizat
    'cooling_rate': 0.85, # optimizat
    'iterations': 800, # optimizat
    'local_searches': 10,
    'multiplier': [0.5, 0.1],
    'function_name': 'goldstein',
    'lower_bound_x': -2,
    'upper_bound_x': 2,
    'lower_bound_y': -2,
    'upper_bound_y': 2
}
```

- Cu valorile ”inițiale”, soluția obținută după rezultatele calculate anterior este $x = -0.0002$, $y = -1.0000$, iar $f(x, y) = 3.0004$.
- Conform metodologiei precedente, voi analiza evoluția parametrului **local_search** pe parcursul a 9 pași, crescându-i valoarea de la 10 până la 100, în incrementuri de 10 de unități la fiecare pas.

PAS 4 – LOCAL_SEARCH **REZULTAT**

35	1500	0.85	800	10	[0.5, 0.1]	0.0001	-1,0005	3,0013
36	1500	0.85	800	20	[0.5, 0.1]	-0,0001	-1,0002	3,0003
37	1500	0.85	800	30	[0.5, 0.1]	0,0000	-1,0000	3,0001
38	1500	0.85	800	40	[0.5, 0.1]	-0,0001	-0,9999	3,0001
39	1500	0.85	800	50	[0.5, 0.1]	0,1800	-0,8802	11,1002
40	1500	0.85	800	60	[0.5, 0.1]	-0,0002	-1,0000	3,0001
41	1500	0.85	800	70	[0.5, 0.1]	0,0001	-1,0001	3,0002
42	1500	0.85	800	80	[0.5, 0.1]	0,0000	-0,9999	3,0001
43	1500	0.85	800	90	[0.5, 0.1]	0,0000	-1,0000	3,0000
44	1500	0.85	800	100	[0.5, 0.1]	0,0000	-1,0000	3,0000

În urma celor 10 x 10 rulări, s-a constatat că valoarea optimă pentru parametrul **local_search** este **90-100**. Începând de acum, voi utiliza această valoare pentru a optimiza parametrii următori.

Observație: Pe baza rezultatelor obținute pentru parametrul **local_search**, putem observa că mai multe valori ating un rezultat aproape identic și foarte apropiat de optimul perfect. De exemplu, pentru valorile de la 10 până la 100 de local_searches, funcția f(x, y) este consistentă și ajunge la valori aproape egale, cum ar fi 3.0001, 3.0000 și 3.0013. Aceste valori sugerează că, în acest interval, optimizarea a fost destul de eficientă, iar comportamentul funcției nu a suferit modificări semnificative în acest domeniu de local_searches.

Astfel, putem concluziona că parametrul local_search în acest interval (90-100) duce la rezultate extrem de apropiate de optimul perfect, indicând o stabilitate ridicată în cadrul căutării soluțiilor, iar alegerea unui astfel de interval pentru **local_search** poate fi considerată o opțiune optimă pentru acest model de optimizare.

PAS 5 – MULTIPLIER

```
goldstein_params = {
    'temp': 1500, # optimizat
    'cooling_rate': 0.85, # optimizat
    'iterations': 800, # optimizat
    'local_searches': 100, # optimizat
    'multiplier': [0.5, 0.1],
    'function_name': 'goldstein',
    'lower_bound_x': -2,
    'upper_bound_x': 2,
    'lower_bound_y': -2,
    'upper_bound_y': 2
}
```

- Cu valorile ”inițiale”, soluția obținută după rezultatele calculate anterior este x = -0.0000, y = -1.0000, iar f(x, y) = 3.0000.
- Conform metodologiei precedente, voi analiza evoluția parametrului **multiplier** pe parcursul a 9 pași, crescându-i valoarea de la [**0.5**, 0.1] până la [**0.77**, 0.1], în incrementuri de **0.03** de unități la fiecare pas.

PAS 5 – MULTIPLIER **REZULTAT**

46	1500	0.85	800	100	[0.50, 0.1]	0.0001	-1,0005	3,0013
47	1500	0.85	800	100	[0.53, 0.1]	0,0001	-0,9999	3,0000
48	1500	0.85	800	100	[0.56, 0.1]	-0,0001	-1,0001	3,0001
49	1500	0.85	800	100	[0.59, 0.1]	-0,0002	-1,0000	3,0000
50	1500	0.85	800	100	[0.62, 0.1]	-0,0001	-1,0000	3,0001
51	1500	0.85	800	100	[0.65, 0.1]	0,0000	-1,0001	3,0000
52	1500	0.85	800	100	[0.68, 0.1]	0,0001	-0,9999	3,0000
53	1500	0.85	800	100	[0.71, 0.1]	0,0001	-1,0000	3,0001
54	1500	0.85	800	100	[0.74, 0.1]	-0,0001	-1,0000	3,0001
55	1500	0.85	800	100	[0.77, 0.1]	0,0000	-1,0000	3,0000

Pe baza rezultatelor obținute pentru parametrul **multiplier**, observăm că mai multe combinații de valori ale acestuia duc la rezultate aproape identice și extrem de apropiate de optimul perfect. În particular, valorile funcției *f(x, y)* pentru diferite combinații de **multiplier** (de la [0.53, 0.1] până la [0.77, 0.1]) sunt toate foarte apropiate de 3.0000 sau 3.0001, indicând că aceste configurații sunt aproape de optimul perfect pentru funcția Goldstein.

Aceasta sugerează că în intervalul valorilor de **multiplier** menționate, rezultatele sunt foarte stabile și că orice valoare din acest interval va conduce la soluții eficiente, apropiate de optimul global al funcției. De asemenea, acest comportament sugerează o oarecare toleranță față de variațiile parametrului **multiplier**, ceea ce poate oferi flexibilitate în alegerea valorii acestuia în procesul de optimizare.

CONCLUZIE

- 1. **Eficiența în apropierea optimului:** Algoritmul Simulated Annealing este capabil să găsească soluții foarte apropiate de optimul global al funcției Goldstein, în special pentru valorile parametrilor precum temp și cooling_rate corespunzătoare unui răcire mai lentă (precum 0.85) și o temperatură mai mare (1500). Cele mai bune rezultate sunt obținute atunci când temp este setat în intervalul 1500–2000 și cooling_rate este 0.85, indicând faptul că aceste setări permit explorarea eficientă a spațiului de soluții și evitarea capcanelor minime locale.
- 2. **Stabilitate la variații:** Atunci când sunt ajustați parametrii, precum numărul de iterații și căutările locale, rezultatele rămân foarte stabile, în apropierea valorii 3.0000. De exemplu, în majoritatea cazurilor (când multiplier variază de la [0.53, 0.1] până la [0.77, 0.1]), soluțiile nu se îndepărtează semnificativ de optimul perfect. Aceasta sugerează că algoritmul este robust la mici variații ale parametrilor și poate găsi soluții excelente în diferite condiții.
- 3. **Influența parametru temp**: La temperaturi mai mari (ex. 2500-3500), algoritmul tinde să producă rezultate mai slabe (de exemplu, valori de f(x, y) mult mai mari, în jur de 11), ceea ce sugerează că o temperatură prea ridicată poate duce la o explorare aleatorie prea largă a spațiului de soluții, fără a permite convergența spre soluții optime. O temperatură mai mică, dar constantă, în combinație cu un cooling_rate moderat, pare a fi ideală pentru obținerea celor mai bune rezultate.
- 4. **Importanța numărului de iterații și căutările locale**: Creșterea numărului de iterații (ex. 200-1000) și ajustarea căutărilor locale pare să ajute la obținerea unor soluții mai precise.

În concluzie, algoritmul Simulated Annealing pentru funcția Goldstein funcționează eficient atunci când setările sunt bine echilibrate. Temperaturile moderate (1500-2000), un cooling_rate de 0.85, și o număr suficient de mare de iterații permit găsirea rapidă a unor soluții foarte apropiate de optimul global, iar algoritmul este robust și stabil la variațiile parametrilor.