

Projekt 3 - Symulowanie wyżarzania

Procedura Metropolis

```
In[1]:= Metropolis[function_, dx_, dy_, epsilon_, it_, number_, t_] :=  
  Module[{f = function, rangex = dx, rangey = dy,  
    r = epsilon, iterations = it, n = number, temp = t},  
  listofpoints := {};  
  FListOfPoints := {};  
  FAverage := {};  
  (*losujemy początkowy punkt*)  
  AppendTo[listofpoints, {Random[Real, rangex], Random[Real, rangey]}];  
  BESTPOINT = listofpoints[[1]]; (*najlepszy z dotychczasowych punktów*)  
  (*dodanie pierwszego wylosowanego punktu*)  
  AppendTo[FListOfPoints, f[listofpoints[[1]]];  
  AppendTo[FAverage, f[listofpoints[[1]]];  
  For[i = 2, i ≤ it, i++,  
    pointsTmp = {};  
    FpointsTmp = {};  
    (*warunki, by procedura nie wyszła poza podany zakres funkcji*)  
    rnx = 0;  
    If[listofpoints[[Length[listofpoints], 1]] - r < rangex[[1],  
      rnx = rangex[[1], rnx = listofpoints[[Length[listofpoints], 1]] - r];  
    lnx = 0;  
    If[listofpoints[[Length[listofpoints], 1]] + r > rangex[[2],  
      lnx = rangex[[2], lnx = listofpoints[[Length[listofpoints], 1]] + r];  
    rny = 0;  
    If[listofpoints[[Length[listofpoints], 2]] - r < rangey[[1],  
      rny = rangey[[1], rny = listofpoints[[Length[listofpoints], 2]] - r];  
    lny = 0;  
    If[listofpoints[[Length[listofpoints], 2]] + r > rangey[[2],  
      lny = rangey[[2], lny = listofpoints[[Length[listofpoints], 2]] + r];
```

```

dxTmp = {rnx, lnx};
dyTmp = {rny, lny};
(*pętla losująca n punktów w sąsiedztwie poprzedniego najlepszego punktu*)
For[j = 1, j ≤ n, j++,
AppendTo[pointsTmp, {Random[Real, dxTmp], Random[Real, dyTmp]}];
AppendTo[FpointsTmp, f[pointsTmp[[j]]];
];
bestpoint = pointsTmp[[1]];
For[k = 2, k ≤ n, k++,
If[f[bestpoint] > f[pointsTmp[[k]]], bestpoint = pointsTmp[[k]];
(*znalezienie punktu dla którego wartość f jest najmniejsza*)
];

(*energia cząsteczki*)
DeltaEnergy = f[listofpoints[Length[listofpoints]]] - f[bestpoint];
(*nierówność >, ponieważ szukamy punktu minimalnego, czyli wartość funkcji w
    tym punkcie ma być mniejsza, więc chcemy uzyskać dodatnią różnicę*)
If[DeltaEnergy > 0,
AppendTo[listofpoints, bestpoint];
AppendTo[FListOfPoints, f[bestpoint]];
AppendTo[FAverage, Mean[FpointsTmp]];
If[f[bestpoint] < f[BESTPOINT], BESTPOINT = bestpoint],
(*losowanie dowolnej wartości*)
ksi = Random[Real, {0, 1}];
(*sprawdzanie prawdopodobieństwa*)
If[ksi < Exp[DeltaEnergy/temp],
AppendTo[listofpoints, bestpoint];
AppendTo[FListOfPoints, f[bestpoint]];
AppendTo[FAverage, Mean[FpointsTmp]];
];
];
];
Return[{listofpoints, FListOfPoints, FAverage, BESTPOINT}]
]

```

Procedura wyświetlania

```

In[2]:= Drawing[function_, sol_, rangex_, rangey_] :=
  Module[{f = function, solution = sol, dx = rangex, dy = rangey},
    plot = Plot3D[f[{x, y}], {x, dx[[1]], dx[[2]]},
      {y, dy[[1]], dy[[2]]}, ColorFunction → "BlueGreenYellow"];
    tabBestIteration = Table[{i, solution[[2, i]]}, {i, 1, Length[solution[[2]]]};
    tabAverIteration = Table[{i, solution[[3, i]]}, {i, 1, Length[solution[[3]]]};
    tabPoints = Table[Take[solution[[1]], i], {i, 1, Length[solution[[1]]]};
    density = DensityPlot[f[{x, y}], {x, dx[[1]], dx[[2]]},
      {y, dy[[1]], dy[[2]]}, ColorFunction → "BlueGreenYellow"];
    Linesplot =
      Table[ListLinePlot[tabPoints[[i]], PlotStyle → Pink], {i, 1, Length[solution[[1]]]};
    Pointsplot =
      Table[ListPlot[tabPoints[[i]], PlotStyle → Pink], {i, 1, Length[solution[[1]]]};
    animacja = ListAnimate[Table[Show[density, Linesplot[[i]], Pointsplot[[i]],
      {i, 1, Length[solution[[1]]]}, AnimationRunning → False];

    iterBest = ListPlot[tabBestIteration, PlotRange → Full, ImageSize → 360];
    iterAver = ListLinePlot[tabAverIteration, PlotRange → Full, ImageSize → 360];
    Grid[{{"PROSTE PRZESZUKIWANIE LOKALNE", SpanFromLeft},
      {"Wzór funkcji", "Wykres funkcji"}, {f[{x, y}], plot},
      {"Przebieg wartości minimalnej w zależności od iteracji",
        "Przebieg wartości średniej w zależności od iteracji"}, {iterBest, iterAver},
      {"Najlepsza znaleziona wartość", "Punkt, w którym znaleziono minimum"},
        {f[solution[[4]]], solution[[4]]},
      {"Przebieg poszukiwania rozwiązania", SpanFromLeft}, {animacja, SpanFromLeft}},
    Frame → All,
    Background → {None, {Pink}, {{2, 1} → LightPink, {2, 2} → LightPink, {4, 1} → LightPink,
      {4, 2} → LightPink, {6, 1} → LightPink, {6, 2} → LightPink, {8, 1} → LightPink}}]]

```

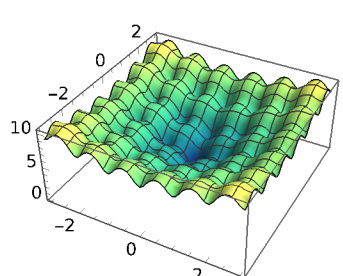
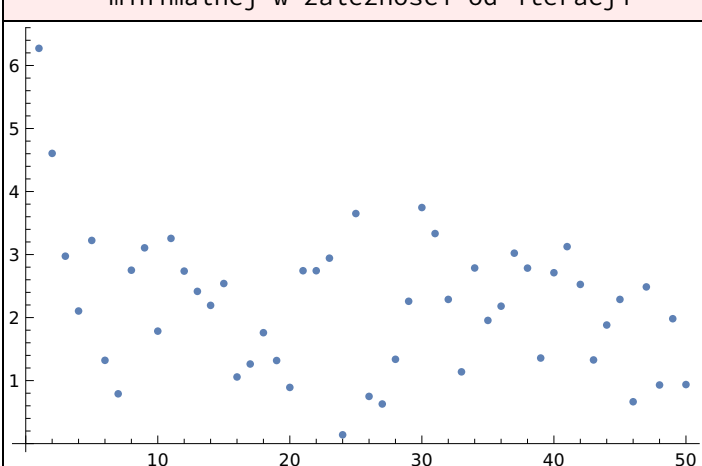
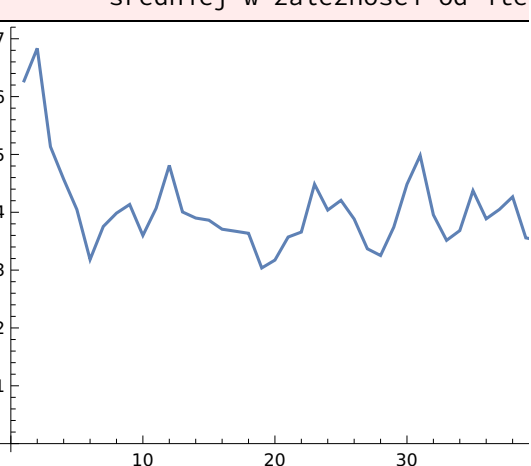
Przykład 1

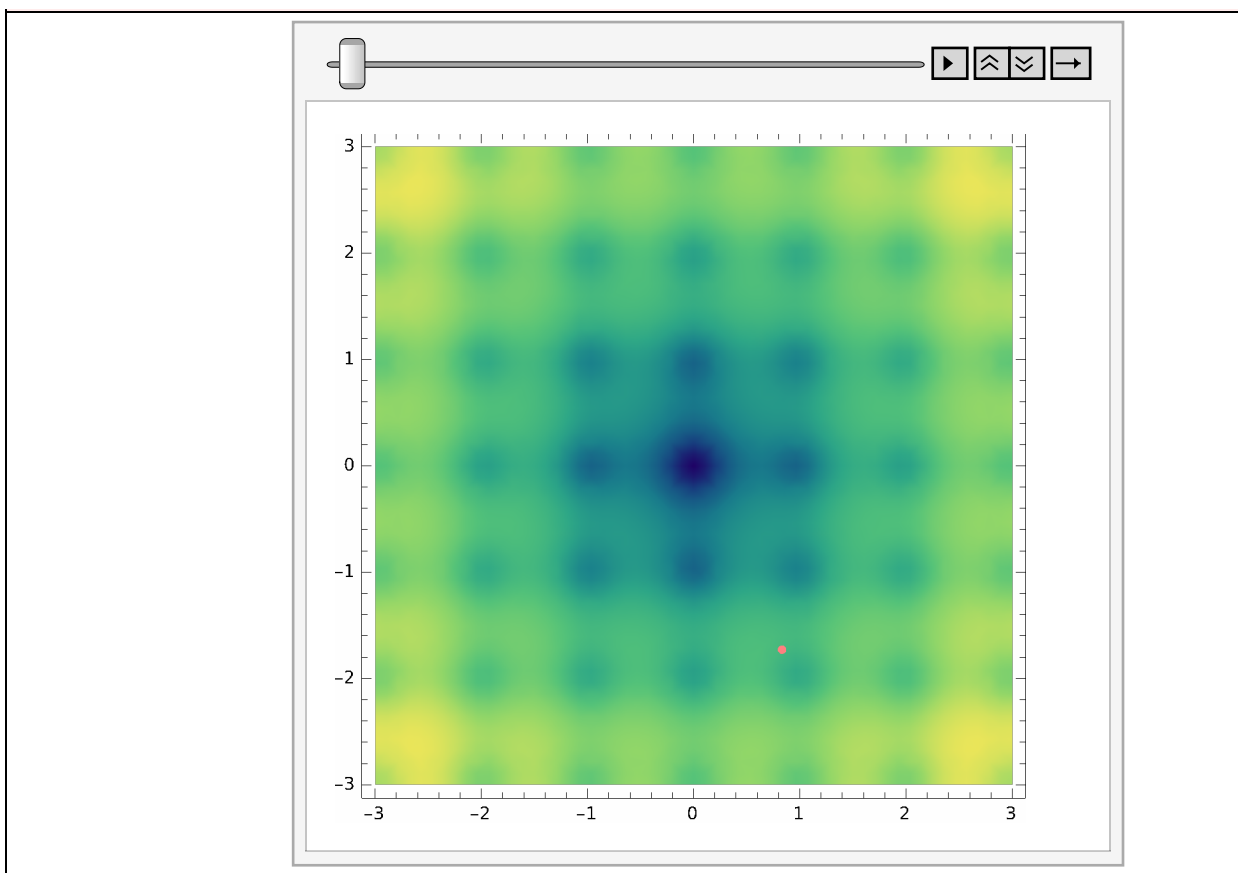
```

In[3]:= Clear[dx, dy, n, it, solution, epsilon]
AckleyFunction2[{x_, y_}] :=
  -20 * Exp[-0.2 * Sqrt[(x ^ 2 + y ^ 2) / 2]] - Exp[(Cos[2 * Pi * x] + Cos[2 * Pi * y]) / 2] + 20 + E
dx = {-3, 3};
dy = {-3, 3};
epsilon = 1;
n = 10;
it = 50;
temp = 100;
solution = Metropolis[AckleyFunction2, dx, dy, epsilon, it, n, temp];
Drawing[AckleyFunction2, solution, dx, dy]

```

Out[12]=

PROSTE PRZESZUKIWANIE LOKALNE	
Wzór funkcji	Wykres funkcji
$20 + e - 20 e^{-0.141421 \sqrt{x^2 + y^2}} - e^{\frac{1}{2} (\cos[2 \pi x] + \cos[2 \pi y])}$	
Przebieg wartości minimalnej w zależności od iteracji	Przebieg wartości średniej w zależności od ite
	
Najlepsza znaleziona wartość	Punkt, w którym znaleziono min
0.139866	{0.0117567, -0.0348915}
Przebieg poszukiwania rozwiązania	



Przykład 2

```

In[21]:= Clear[dx, dy, n, it, solution, epsilon]
          SphericalFunction[{x_, y_}] := x^2 + y^2
          dx = {-5, 5};
          dy = {-5, 5};
          epsilon = 1;
          n = 10;
          it = 50;
          solution = Metropolis[SphericalFunction, dx, dy, epsilon, it, n, 100];
          Drawing[SphericalFunction, solution, dx, dy]

```

Przykład 3

```
In[30]:= Clear[dx, dy, n, it, solution, epsilon]
ShubertFunction2[{x_, y_}] :=
  Sum[i * Cos[(i + 1) * x + i], {i, 5}] * Sum[i * Cos[(i + 1) * y + i], {i, 5}]
dx = {-6, 6};
dy = {-6, 6};
epsilon = 2;
n = 10;
it = 50;
solution = Metropolis[ShubertFunction2, dx, dy, epsilon, it, n, 100];
Drawing[ShubertFunction2, solution, dx, dy]
```

Wnioski

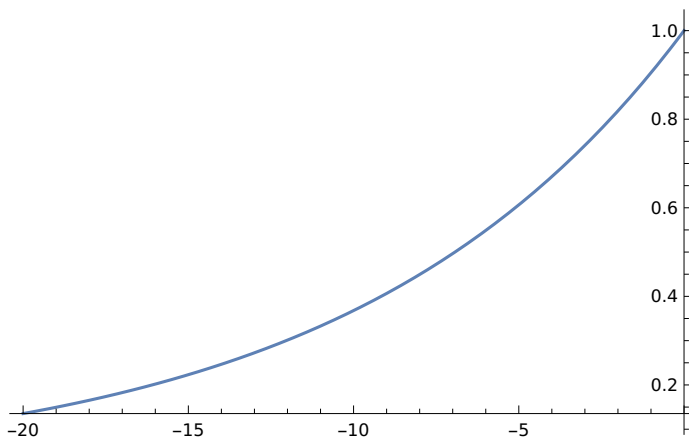
Zwiększenie ilości losowanych punktów zwiększa prawdopodobieństwo uzyskiwania coraz to lepszych wyników. Dla funkcji o wielu ekstremach lokalnych konieczne jest zwiększenie wartości sąsiedztwa, by móc zapewnić wyjście z ekstremum lokalnego tak by dostać się do ekstremum globalnego.

Jako nowe rozwiązanie może być wybrany dowolny element z sąsiedztwa. Jeżeli potencjalne nowe rozwiązanie jest lepsze od poprzednika, to zostaje zapisane jako nowe rozwiązanie. Kiedy nowe potencjalne rozwiązanie jest gorsze od dotychczasowego, zapisuje się je w sposób losowy. Prawdopodobieństwo wyboru nowego potencjalnego rozwiązania jest tym mniejsze, im różnica między potencjalnym rozwiązaniem a jego poprzednikiem, jest większa (energia cząsteczki). Ogranicza to duże oddalanie się od wcześniej znalezionej rozwiązania. (Wylosowana liczba musi być mniejsza od wartości funkcji $\text{Exp}[\text{ener-}]$

gia_cząsteczki/temperatura], a czym większa różnica tym mniejsza wartość funkcji, więc prawdopodobieństwo wylosowania liczby z przedziału $[0, \text{Exp}[\text{energia_cząsteczki/temperatura}]]$ jest mniejsze)

Jest to swego rodzaju wada dla funkcji o wielu ekstremach lokalnych, ponieważ nie zapewnia nam to wyjścia z danego ekstremum lokalnego, a samo zwiększenie wartości sąsiedztwa nie zapewni nam tego. Dla funkcji o wielu ekstremach lokalnych lepszym wyjściem jest wybór wyższej temperatury.

```
In[92]:= Plot[Exp[x / 10], {x, -20, 0}]
Out[92]=
```



Wyżarzanie

```
In[85]:= Annealing[function_, dx_, dy_, epsilon_, it_, number_, t_] :=
  Module[{f = function, rangex = dx, rangey = dy,
    r = epsilon, iterations = it, n = number, temp = t},
    listofpoints := {};
    FListOfPoints := {};
    FAverage := {};
    (*losujemy początkowy punkt*)
    AppendTo[listofpoints, {Random[Real, rangex], Random[Real, rangey]}];
    BESTPOINT = listofpoints[[1]];
    AppendTo[FListOfPoints, f[listofpoints[[1]]]];
    AppendTo[FAverage, f[listofpoints[[1]]]];
```

```

For[i = 2, i ≤ it, i++,
pointsTmp = {};
FpointsTmp = {};
(*warunki, by procedura nie wyszła poza podany zakres funkcji*)
rnx = 0;
If[listofpoints[Length[listofpoints], 1] - r < rangex[[1],
    rnx = rangex[[1], rnx = listofpoints[Length[listofpoints], 1] - r];
lnx = 0;
If[listofpoints[Length[listofpoints], 1] + r > rangex[[2],
    lnx = rangex[[2], lnx = listofpoints[Length[listofpoints], 1] + r];
rny = 0;
If[listofpoints[Length[listofpoints], 2] - r < rangey[[1],
    rny = rangey[[1], rny = listofpoints[Length[listofpoints], 2] - r];
lny = 0;
If[listofpoints[Length[listofpoints], 2] + r > rangey[[2],
    lny = rangey[[2], lny = listofpoints[Length[listofpoints], 2] + r];

dxTmp = {rnx, lnx};
dyTmp = {rny, lny};
(*pętla losująca n punktów w sąsiedztwie poprzedniego najlepszego punktu*)
For[j = 1, j ≤ n, j++,
AppendTo[pointsTmp, {Random[Real, dxTmp], Random[Real, dyTmp]}];
AppendTo[FpointsTmp, f[pointsTmp[[j]]]];
];
bestpoint = pointsTmp[[1]];
For[k = 2, k ≤ n, k++,
If[f[bestpoint] > f[pointsTmp[[k]]], bestpoint = pointsTmp[[k]];
(*znalezienie punktu dla którego wartość f jest najmniejsza*)
];
(*przeliczenie wartości średniej wszystkich n wylosowanych punktów*)

DeltaEnergy = f[listofpoints[Length[listofpoints]]] - f[bestpoint];
If[DeltaEnergy > 0,
AppendTo[listofpoints, bestpoint];
AppendTo[FListOfPoints, f[bestpoint]];
AppendTo[FAverage, Mean[FpointsTmp]];
If[f[bestpoint] < f[BESTPOINT], BESTPOINT = bestpoint],
ksi = Random[Real, {0, 1}];
If[ksi < N[Exp[DeltaEnergy / temp]],

```



```

AppendTo[listofpoints, bestpoint];
AppendTo[FListOfPoints, f[bestpoint]];
AppendTo[FAverage, Mean[FpointsTmp]];
];
];
temp = 0.8 * temp;
];
Return[{listofpoints, FListOfPoints, FAverage, BESTPOINT}]
]

```

Przykład 1

```

In[58]:= Clear[dx, dy, n, it, solution, epsilon]
AckleyFunction2[{x_, y_}] :=
  -20 * Exp[-0.2 * Sqrt[(x ^ 2 + y ^ 2) / 2]] - Exp[(Cos[2 * Pi * x] + Cos[2 * Pi * y]) / 2] + 20 + E
dx = {-3, 3};
dy = {-3, 3};
epsilon = 1;
n = 10;
it = 50;
solution = Annealing[AckleyFunction2, dx, dy, epsilon, it, n, 100];
Drawing[AckleyFunction2, solution, dx, dy]

```

Przykład 2

```

In[67]:= Clear[dx, dy, n, it, solution, epsilon]
SphericalFunction[{x_, y_}] := x ^ 2 + y ^ 2
dx = {-5, 5};
dy = {-5, 5};
epsilon = 0.5;
n = 10;
it = 50;
solution = Annealing[SphericalFunction, dx, dy, epsilon, it, n, 100];
Drawing[SphericalFunction, solution, dx, dy]

```

Przykład 3

In[140]:=

```
Clear[dx, dy, n, it, solution, epsilon]
ShubertFunction2[{x_, y_}] :=
  Sum[i * Cos[(i + 1) * x + i], {i, 5}] * Sum[i * Cos[(i + 1) * y + i], {i, 5}]
dx = {-6, 6};
dy = {-6, 6};
epsilon = 2;
n = 10;
it = 20;
solution = Annealing[ShubertFunction2, dx, dy, epsilon, it, n, 100];
Drawing[ShubertFunction2, solution, dx, dy]
```

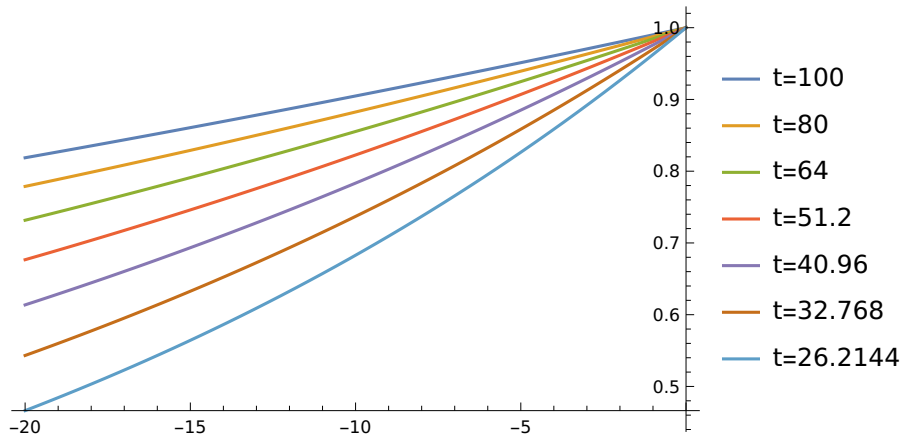
Wnioski

Temperatura jest zmniejszana podczas wyborów kolejnych przybliżeń, wtedy gdy propozycja nowego rozwiązania jest gorsza od poprzedniego rozwiązania. Gdy wartość temperatury jest wysoka, prawdopodobieństwo wyboru jest duże. Gdy temperatura jest niska, prawdopodobieństwo wyboru nowego rozwiązania (gorszego od poprzednika) jest małe. Można to zauważyć na niżej zamieszczonym wykresie.

Temperatura z biegiem iteracji jest obniżana. W kolejnych krokach prawdopodobieństwo przejścia do nowego, gorszego położenia maleje.

```
In[21]:= Plot[Exp[x/100], Exp[x/80], Exp[x/64], Exp[x/51.2],
  Exp[x/40.96], Exp[x/32.768], Exp[x/26.2144]], {x, -20, 0},
  PlotLegends -> {"t=100", "t=80", "t=64", "t=51.2", "t=40.96", "t=32.768", "t=26.2144"}]
```

Out[21]=



Schematy chłodzenia

```
t1 = 100;
tn = 10;
maxIteration = 30;

tmpf1 = {t1};
For[i = 1, i < maxIteration, i++, AppendTo[tmpf1, tmpf1[[i]] * 0.9]]
f1 = Table[{i, tmpf1[[i]]}, {i, 1, maxIteration}];

tmpf1a = {t1};
For[i = 1, i < maxIteration, i++, AppendTo[tmpf1a, tmpf1a[[i]] * 0.85]]
f1a = Table[{i, tmpf1a[[i]]}, {i, 1, maxIteration}];

tmpf1b = {t1};
For[i = 1, i < maxIteration, i++, AppendTo[tmpf1b, tmpf1b[[i]] * 0.8]]
f1b = Table[{i, tmpf1b[[i]]}, {i, 1, maxIteration}];

tmpf1c = {t1};
For[i = 1, i < maxIteration, i++, AppendTo[tmpf1c, tmpf1c[[i]] * 0.95]]
f1c = Table[{i, tmpf1c[[i]]}, {i, 1, maxIteration}];

f2[i_] := t1 - (i - 1) * (t1 - tn) / (maxIteration - 1)
```

```
f3[i_] := t1 * (tn / t1) ^ ((i - 1) / (maxIteration - 1))
```

```
A = (t1 - tn) * maxIteration / (maxIteration - 1);
```

```
f4[i_] := t1 + A * (1 / i - 1)
```

```
B = Log[t1 - tn] / Log[maxIteration - 1];
```

```
f5[i_] := t1 - (i - 1) ^ B
```

```
In[433]:=
```

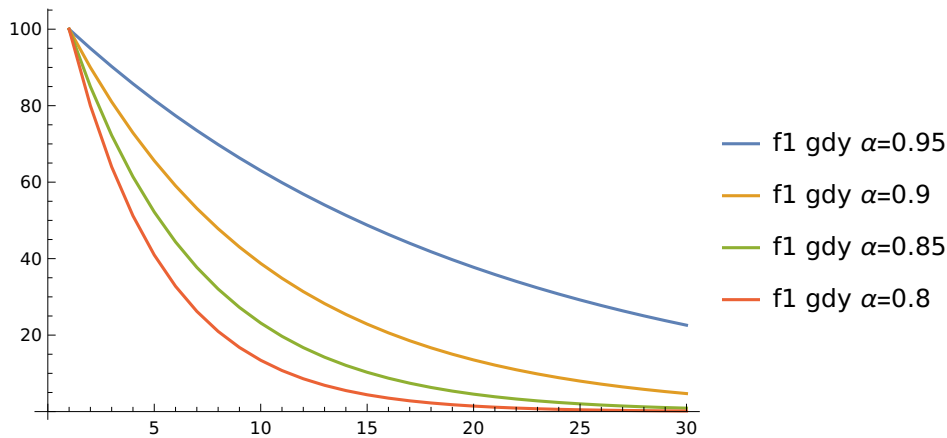
```
ListLinePlot[{f1c, f1, f1a, f1b},
```

```
PlotLegends → {"f1 gdy  $\alpha=0.95$ ", "f1 gdy  $\alpha=0.9$ ", "f1 gdy  $\alpha=0.85$ ", "f1 gdy  $\alpha=0.8$ "}]
```

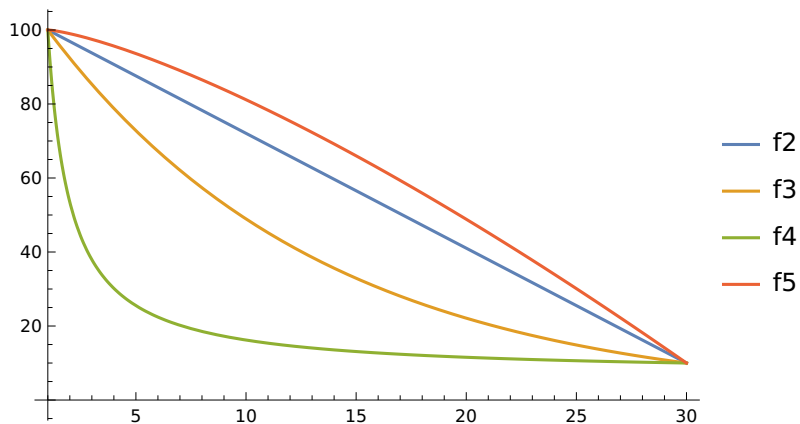
```
Plot[{f2[i], f3[i], f4[i], f5[i]},
```

```
{i, 1, maxIteration}, PlotLegends → {"f2", "f3", "f4", "f5"}]
```

```
Out[433]=
```



```
Out[434]=
```



Funkcja $f1$ jest zależna od ilości iteracji, parametru α oraz temperatury początkowej. Parametr jest z zakresu $\alpha \in [0.8, 0.99]$. Widzimy, że im większy parametr tym funkcja wyznaczająca temperaturę wolniej maleje. Jeśli chcemy by algo-

rytm działał jak najdłużej i powoli szukał ekstremum, bez względu na to czy każdy kolejny wynik jest lepszy to powinniśmy wybrać większą wartość parametru α .

Funkcje f_2 , f_3 , f_4 , f_5 są zależne od temperatury początkowej, końcowej oraz ilości iteracji.

Porównując te 4 funkcje z ostatniego wykresu, w sytuacji gdy chcemy jak najszybciej ustabilizować algorytm i uzyskiwać tylko coraz to lepsze wyniki powinniśmy wybrać funkcję f_4 , ponieważ ona maleje najszybciej. Gdy chcemy, by algorytm działał dłużej i powoli szukał najlepszego wyniku, bez względu na to czy każdy kolejny jest lepszy powinniśmy wybrać funkcję f_5 , ponieważ maleje najwolniej.