Karolina Tatarczyk

Projekt 2 - Proste Przeszukiwanie Lokalne

Procedura

In[2]:=
```
LocalSearch[function_, dx_, dy_, epsilon_, it_, number_] := Module[
  {f = function, rangex = dx, rangey = dy, r = epsilon, iterations = it, n = number},
listofpoints := {};
FListOfPoints := {};
FAverage := {};
(*losujemy początkowy punkt*)
AppendTo[listofpoints, {Random[Real, rangex], Random[Real, rangey]}];
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[[i - 1, 1]] - r < rangex[[1]],
    rnx = rangex[[1]], rnx = listofpoints[[i - 1, 1]] - r];
lnx = 0;
If[listofpoints[[i - 1, 1]] + r > rangex[[2]],
    lnx = rangex[[2]], lnx = listofpoints[[i - 1, 1]] + r];
rny = 0;
If[listofpoints[[i - 1, 2]] - r < rangey[[1]],
    rny = rangey[[1]], rny = listofpoints[[i - 1, 2]] - r];
lny = 0;
If[listofpoints[[i - 1, 2]] + r > rangey[[2]],
    lny = rangey[[2]], lny = listofpoints[[i - 1, 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*)
AppendTo[FAverage, Mean[FpointsTmp]];
(*sprawdzenie czy najlepszy punkt w
    tej iteracji jest niegorszy od poprzedniej najlepszego*)
If[f[listofpoints[[i - 1]]] > f[bestpoint],
AppendTo[listofpoints, bestpoint];
AppendTo[FListOfPoints, f[bestpoint]],
AppendTo[listofpoints, listofpoints[[i - 1]]];
AppendTo[FListOfPoints, f[listofpoints[[i - 1]]]];
];
];
Return[{listofpoints, FListOfPoints, FAverage}]
]
```

In[178]:=

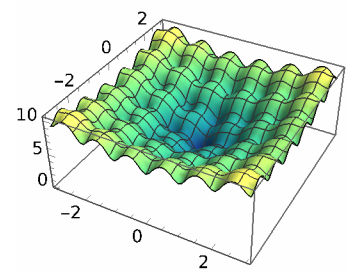
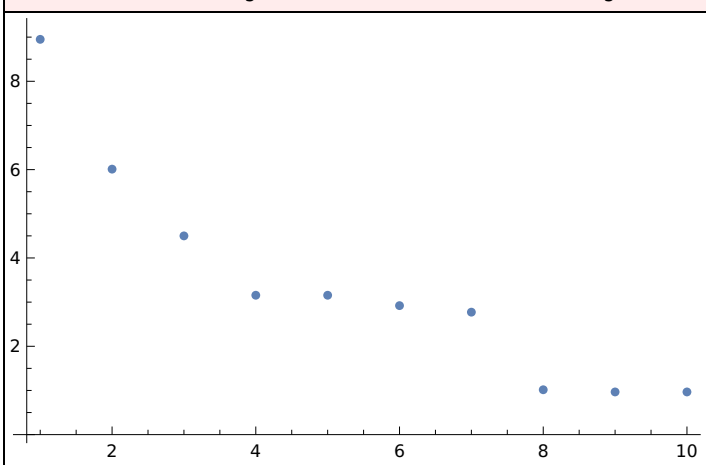
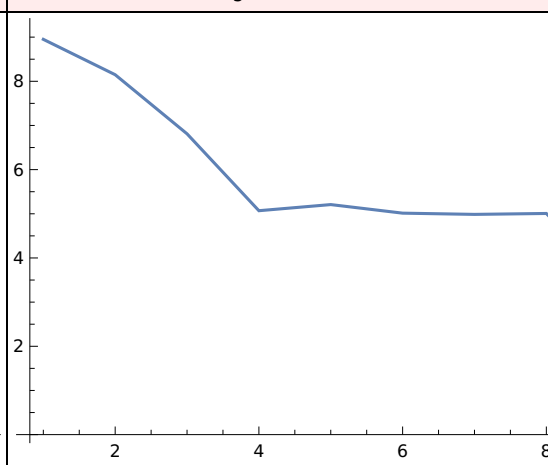
Procedura wyświetlania

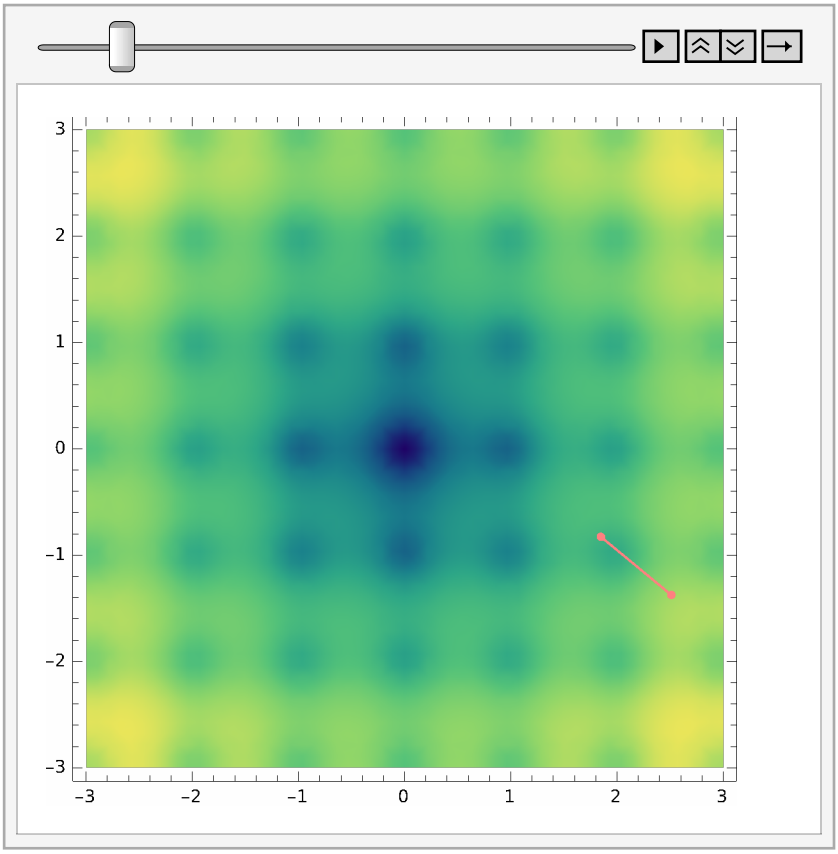
In[1]:=
```
Drawing[function_, sol_, rangex_, rangey_] :=
 Module[{f = function, solution = sol, dx = rangex, dy = rangey},
plot = Plot3D[f[{x, y}], {x, dx[[1]], dx[[2]]},
    {y, dy[[1]], dy[[2]]}, ColorFunction → "BlueGreenYellow"];
tabBestIteration = Table[{i, solution[[2, i]]}, {i, 1, Length[solution[[2]]]}];
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"},
    {solution[[2, Length[solution[[1]]]]], solution[[1, Length[solution[[1]]]]]},
{"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[4]:=
```
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 = 10;
solution = LocalSearch[AckleyFunction2, dx, dy, epsilon, it, n];
Drawing[AckleyFunction2, solution, dx, dy]
```

Out[12]=

PROSTE PRZESZUKIWANIE LOKALNE	
Wzór funkcji	Wykres funkcji
$20 + e - 20\, e^{-0.141421 \sqrt{x^2+y^2}} - e^{\frac{1}{2}\left(\mathrm{Cos}[2\pi x]+\mathrm{Cos}[2\pi y]\right)}$	
Przebieg wartości minimalnej w zależności od iteracji	Przebieg wartości średniej w zależności od ite
	
Najlepsza znaleziona wartość	Punkt, w którym znaleziono mir
0.966317	{-0.154229, 0.0123386}
Przebieg poszukiwania rozwiązania	

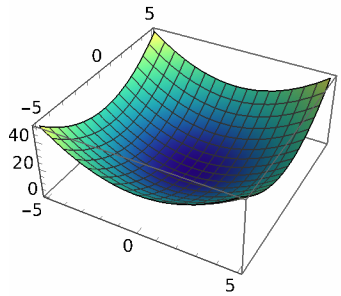
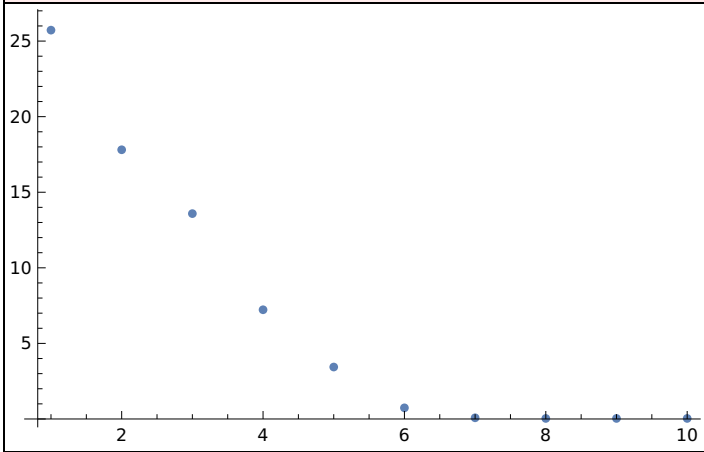
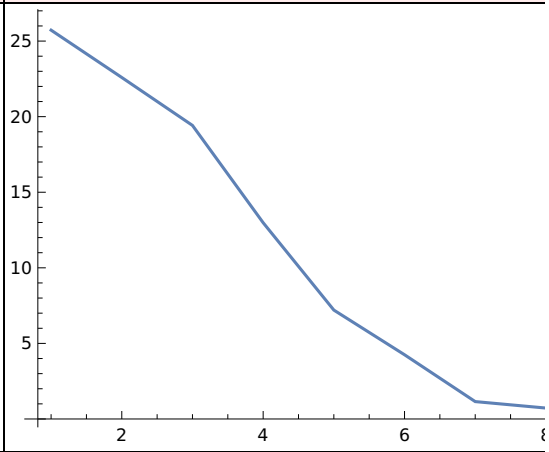


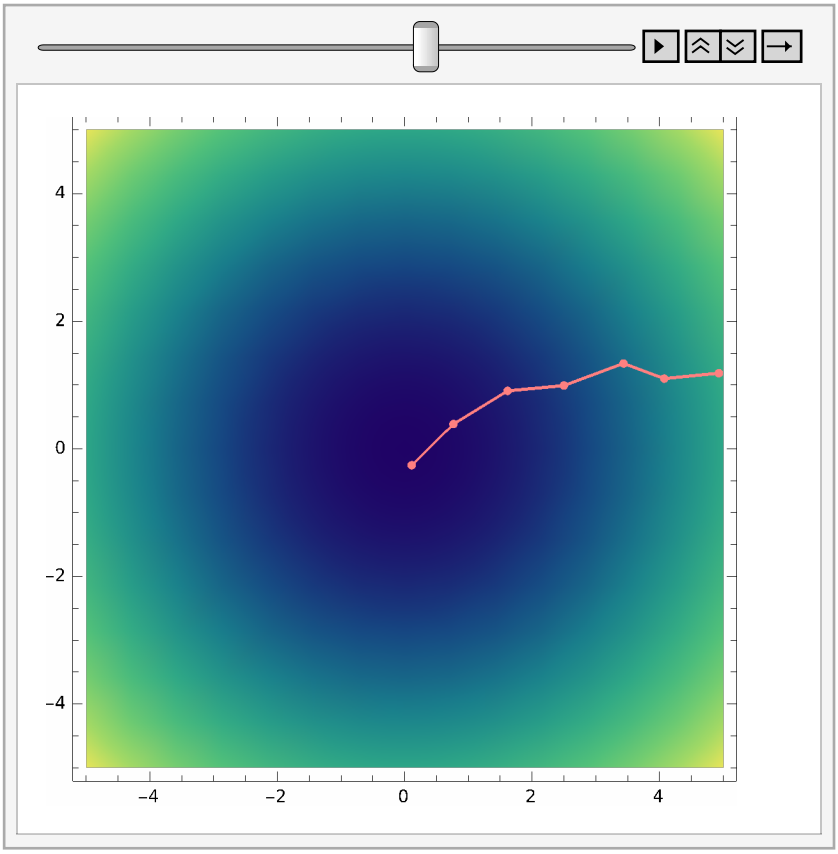
Przykład 2

In[13]:=
```
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 = 10;
solution = LocalSearch[SphericalFunction, dx, dy, epsilon, n, it];
Drawing[SphericalFunction, solution, dx, dy]
```

Out[21]=

PROSTE PRZESZUKIWANIE LOKALNE	
Wzór funkcji	Wykres funkcji

$x^2 + y^2$	
Przebieg wartości minimalnej w zależności od iteracji	Przebieg wartości średniej w zależności od ite
	
Najlepsza znaleziona wartość	Punkt, w którym znaleziono mir
0.0358083	{-0.147331, -0.118751}
Przebieg poszukiwania rozwiązania	

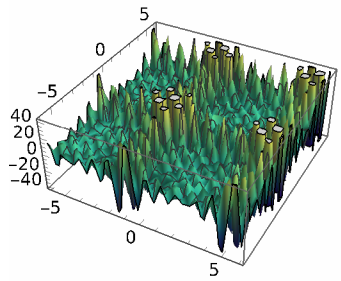
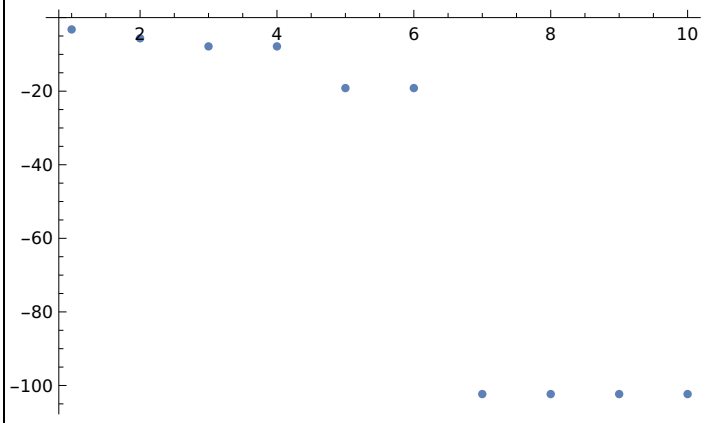
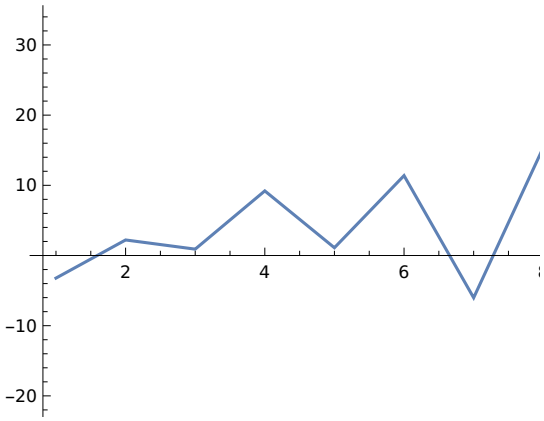


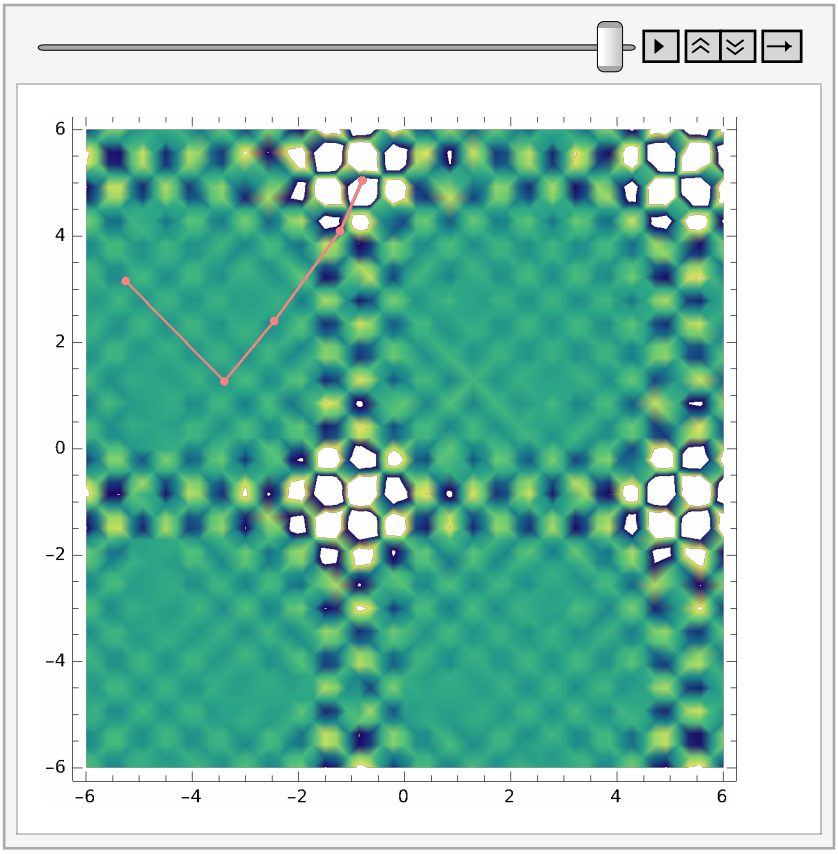
Przykład 3

In[22]:=
```
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 = 10;
solution = LocalSearch[ShubertFunction2, dx, dy, epsilon, n, it];
Drawing[ShubertFunction2, solution, dx, dy]
```

Out[30]=

PROSTE PRZESZUKIWANIE LOKALNE	
Wzór funkcji	Wykres funkcji

$\left(\mathrm{Cos}[1+2x]+2\,\mathrm{Cos}[2+3x]+3\,\mathrm{Cos}[3+4x]+4\,\mathrm{Cos}[4+5x]+5\,\mathrm{Cos}[5+6x]\right)\left(\mathrm{Cos}[1+2y]+2\,\mathrm{Cos}[2+3y]+3\,\mathrm{Cos}[3+4y]+4\,\mathrm{Cos}[4+5y]+5\,\mathrm{Cos}[5+6y]\right)$	
Przebieg wartości minimalnej w zależności od iteracji	Przebieg wartości średniej w zależności od ite
	
Najlepsza znaleziona wartość	Punkt, w którym znaleziono mir
−102.337	{−0.795097, 5.05152}
Przebieg poszukiwania rozwiązania	



Wnioski

Jakość rozwiązania końcowego zależy od wyboru rozwiązania startowego , które w tym wypadku jest rozwiązaniem losowym, co nie zapewnia nam uzyskania najlepszego oczekiwanego wyniku.

Zwiększenie ilości losowanych punktów zwiększa prawdopodobieństwo uzyskiwania coraz to lepszych wyników. Dla funkcji, które posiadają ekstrema lokalne konieczne jest zwiększenie wartości promienia sąsiedztwa, by uniknąć zagnieżdżenia w ekstremum lokalnym, które nie jest ekstremum globalnym. Nie zawsze zapewnia nam to jednak "wyjście" z ekstremum lokalnego. Widzimy więc, że algorytm nie jest zbyt

odpowiedni dla takich funkcji.
Algorytm jest korzystny w poszukiwaniu ekstremum globalnego, gdy funkcja nie posiada żadnych innych ekstremów lokalnych.
W prostym przeszukiwaniu lokalnym, dla funkcji zawierających jedno ekstremum globalne, można również zauważyć, że wartość średnia losowanych punktów maleje wraz ze wzrostem iteracji, ponieważ sąsiedztwo zmienia się, zbliżając się do coraz to mniejszych wartości.

Procedura ze zmiennym sąsiedztwem

In[2]:=
```
NeighborhoodSearch[function_, dx_, dy_, epsilon_, it_, number_] := Module[
  {f = function, rangex = dx, rangey = dy, r = epsilon, iterations = it, n = number},
listofpoints := {};
FListOfPoints := {};
FAverage := {};
(*losujemy początkowy punkt*)
AppendTo[listofpoints, {Random[Real, rangex], Random[Real, rangey]}];
AppendTo[FListOfPoints, f[listofpoints[[1]]]];
AppendTo[FAverage, f[listofpoints[[1]]]];
For[i = 2, i ≤ iterations, i++,
pointsTmp = {};
FpointsTmp = {};
 (*warunki,by procedura nie wyszła poza podany zakres funkcji*)
rnx = 0;
If[listofpoints[[Length[listofpoints], 1]] - r * (i - 1) < rangex[[1]],
    rnx = rangex[[1]], rnx = listofpoints[[Length[listofpoints], 1]] - r * (i - 1)];
lnx = 0;
If[listofpoints[[Length[listofpoints], 1]] + r * (i - 1) → rangex[[2]],
    lnx = rangex[[2]], lnx = listofpoints[[Length[listofpoints], 1]] + r * (i - 1)];
rny = 0;
```

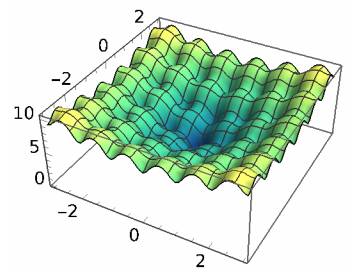
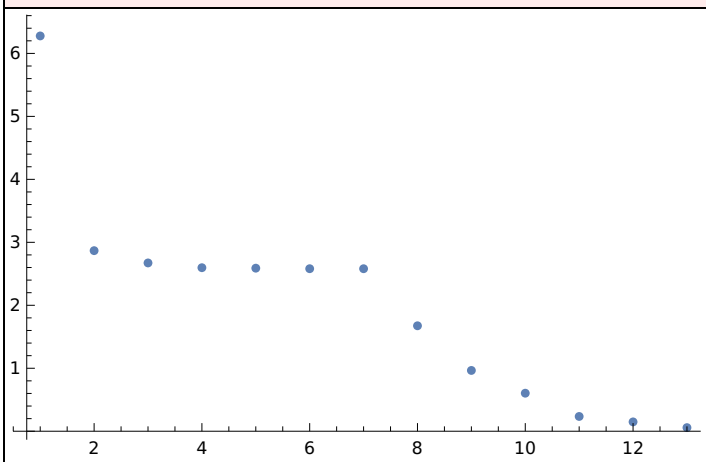
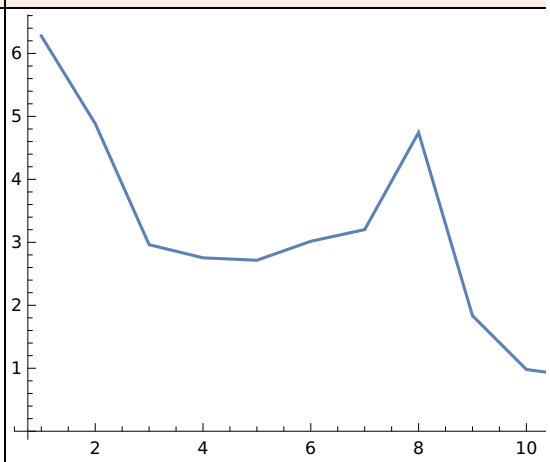
```
If[listofpoints[[Length[listofpoints], 2]] - r * i < rangey[[1]],
    rny = rangey[[1]], rny = listofpoints[[Length[listofpoints], 2]] - r * (i - 1)];
lny = 0;
If[listofpoints[[Length[listofpoints], 2]] + r * (i - 1) > rangey[[2]],
    lny = rangey[[2]], lny = listofpoints[[Length[listofpoints], 2]] + r * (i - 1)];

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*)
];
(*sprawdzenie czy najlepszy punkt w tej
    iteracji jest niegorszy od poprzedniej najlepszego*)
If[f[listofpoints[[Length[listofpoints]]]] > f[bestpoint],
AppendTo[listofpoints, bestpoint];
AppendTo[FAverage, Mean[FpointsTmp]];
AppendTo[FListOfPoints, f[bestpoint]];
i = 1
];
];
Return[{listofpoints, FListOfPoints, FAverage}]
]
```

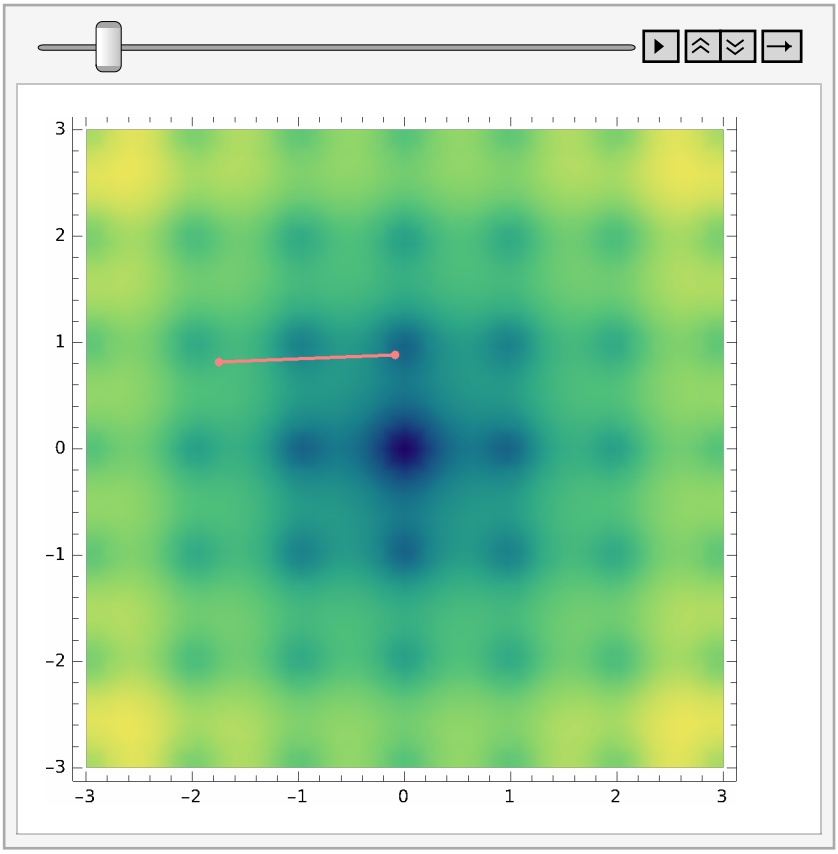
Przykład 1

In[108]:=

```
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 = 0.1;
n = 10;
it = 10;
solution = NeighborhoodSearch[AckleyFunction2, dx, dy, epsilon, it, n];
Drawing[AckleyFunction2, solution, dx, dy]
```

Out[116]=

PROSTE PRZESZUKIWANIE LOKALNE	
Wzór funkcji	Wykres funkcji
$20 + e - 20\, e^{-0.141421 \sqrt{x^2+y^2}} - e^{\frac{1}{2}\left(\mathrm{Cos}[2\pi x]+\mathrm{Cos}[2\pi y]\right)}$	
Przebieg wartości minimalnej w zależności od iteracji	Przebieg wartości średniej w zależności od ite
	
Najlepsza znaleziona wartość	Punkt, w którym znaleziono mir
0.0572071	{-0.008961, 0.0148988}
Przebieg poszukiwania rozwiązania	



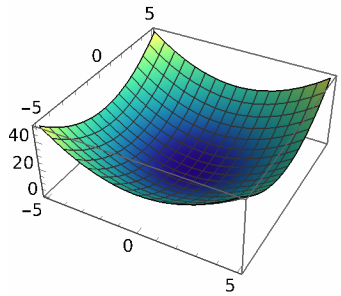
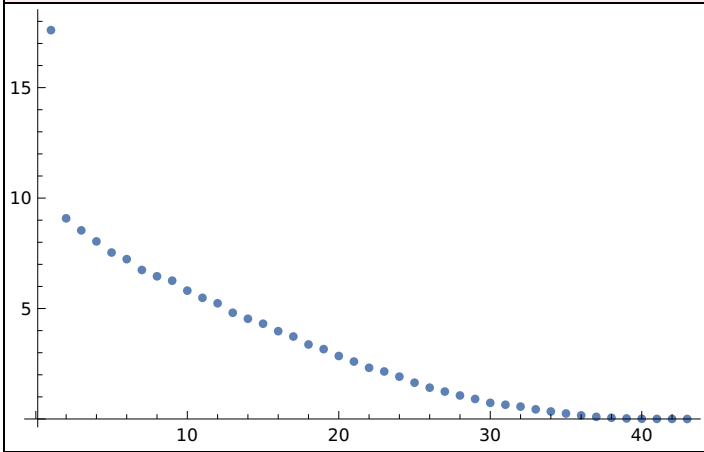
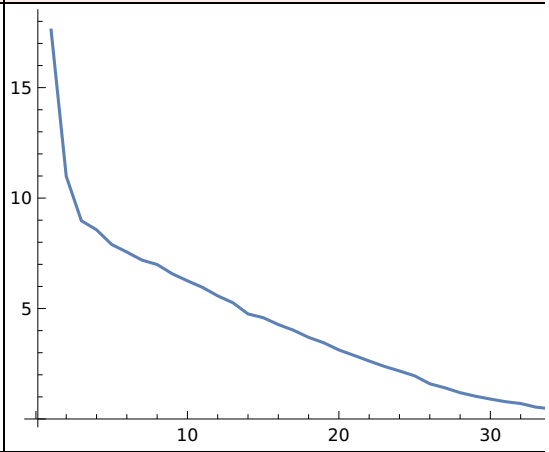
Przykład 2

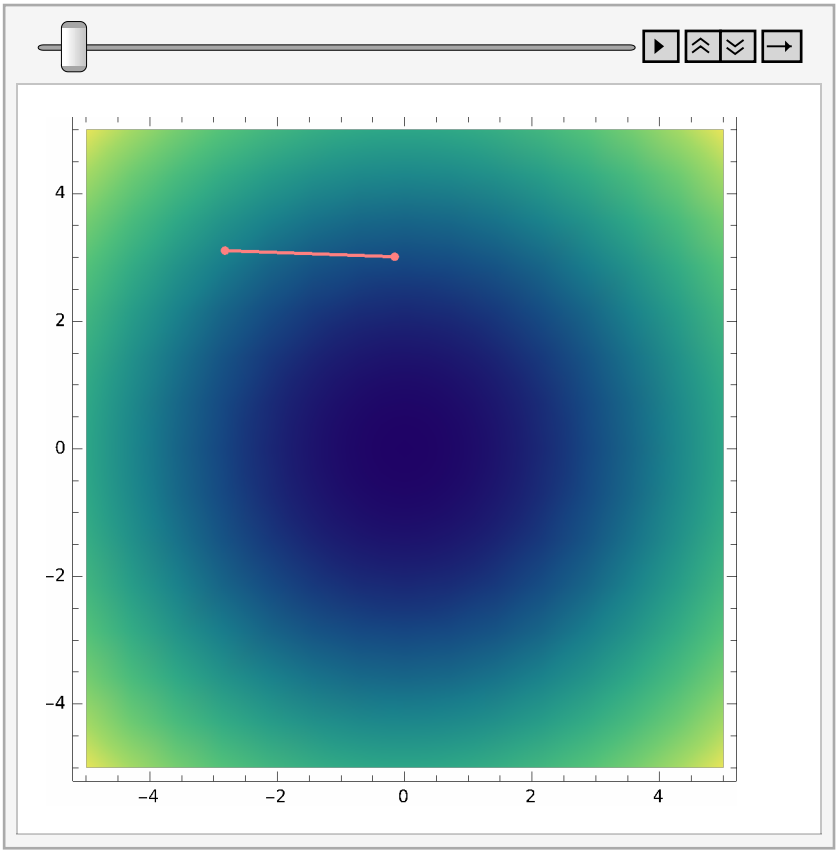
In[117]:=

```
Clear[dx, dy, n, it, solution, epsilon]
SphericalFunction[{x_, y_}] := x^2 + y^2
dx = {-5, 5};
dy = {-5, 5};
epsilon = 0.1;
n = 10;
it = 10;
solution = NeighborhoodSearch[SphericalFunction, dx, dy, epsilon, n, it];
Drawing[SphericalFunction, solution, dx, dy]
```

Out[125]=

PROSTE PRZESZUKIWANIE LOKALNE	
Wzór funkcji	Wykres funkcji

$x^2 + y^2$	
Przebieg wartości minimalnej w zależności od iteracji	Przebieg wartości średniej w zależności od ite
	
Najlepsza znaleziona wartość	Punkt, w którym znaleziono mir
0.000739474	{-0.0218349, 0.0162083}
Przebieg poszukiwania rozwiązania	

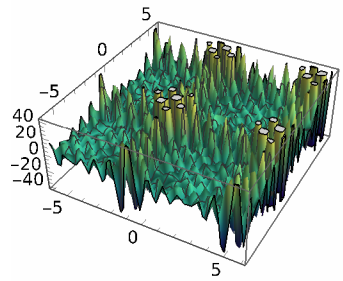
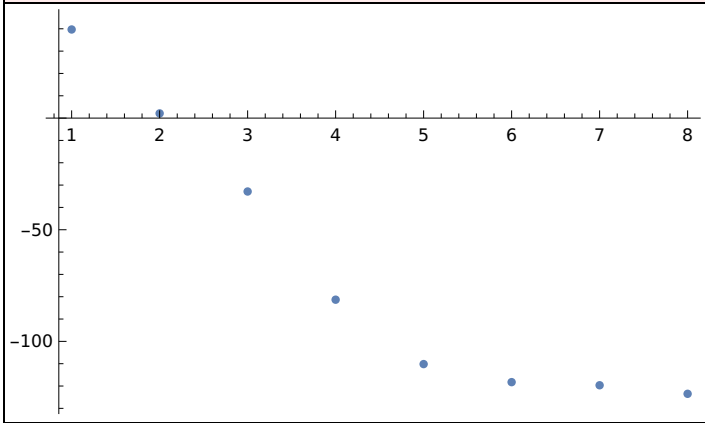
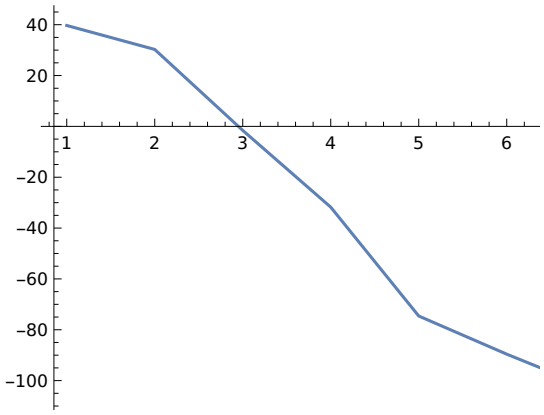


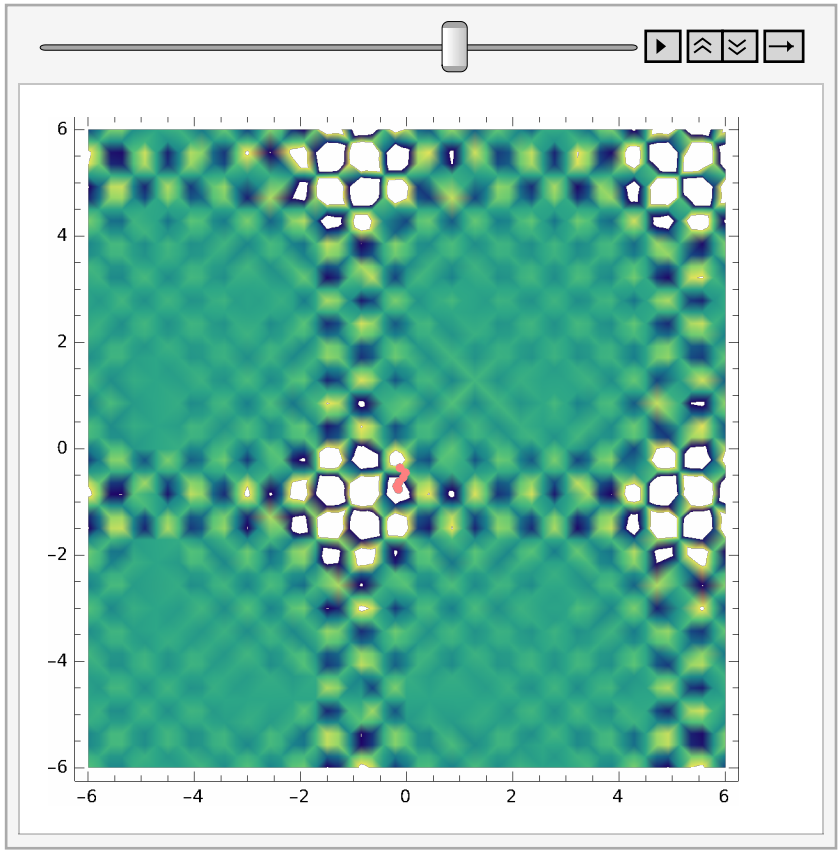
Przykład 3

In[21]:=
```
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 = 0.1;
n = 10;
it = 10;
solution = NeighborhoodSearch[ShubertFunction2, dx, dy, epsilon, n, it];
Drawing[ShubertFunction2, solution, dx, dy]
```

Out[29]=

PROSTE PRZESZUKIWANIE LOKALNE	
Wzór funkcji	Wykres funkcji

$(\mathrm{Cos}[1+2x]+2\,\mathrm{Cos}[2+3x]+3\,\mathrm{Cos}[3+4x]+4\,\mathrm{Cos}[4+5x]+5\,\mathrm{Cos}[5+6x])(\mathrm{Cos}[1+2y]+2\,\mathrm{Cos}[2+3y]+3\,\mathrm{Cos}[3+4y]+4\,\mathrm{Cos}[4+5y]+5\,\mathrm{Cos}[5+6y])$	
Przebieg wartości minimalnej w zależności od iteracji	Przebieg wartości średniej w zależności od ite
	
Najlepsza znaleziona wartość	Punkt, w którym znaleziono mir
−123.487	{−0.193773, −0.792683}
Przebieg poszukiwania rozwiązania	

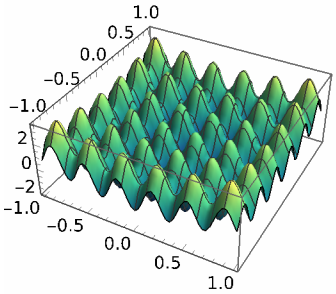
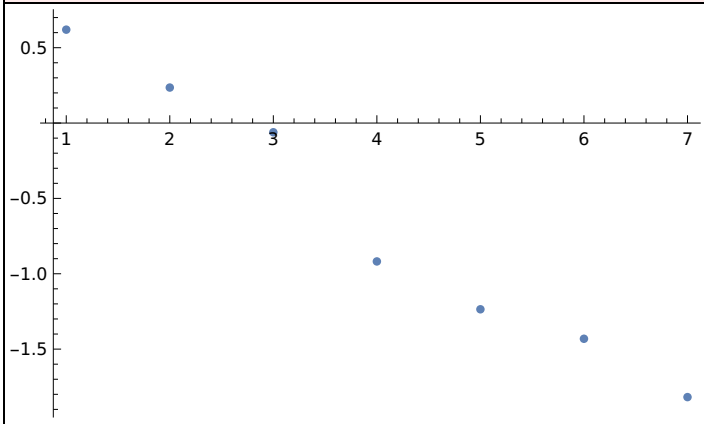
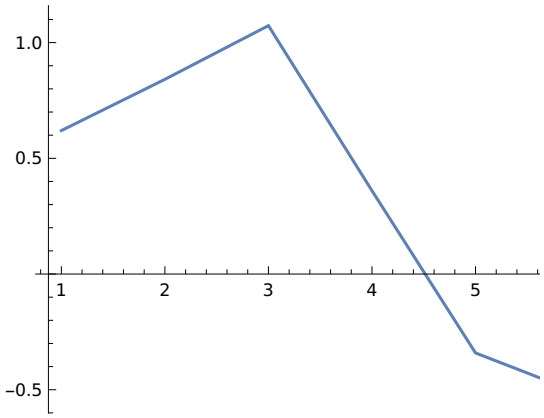


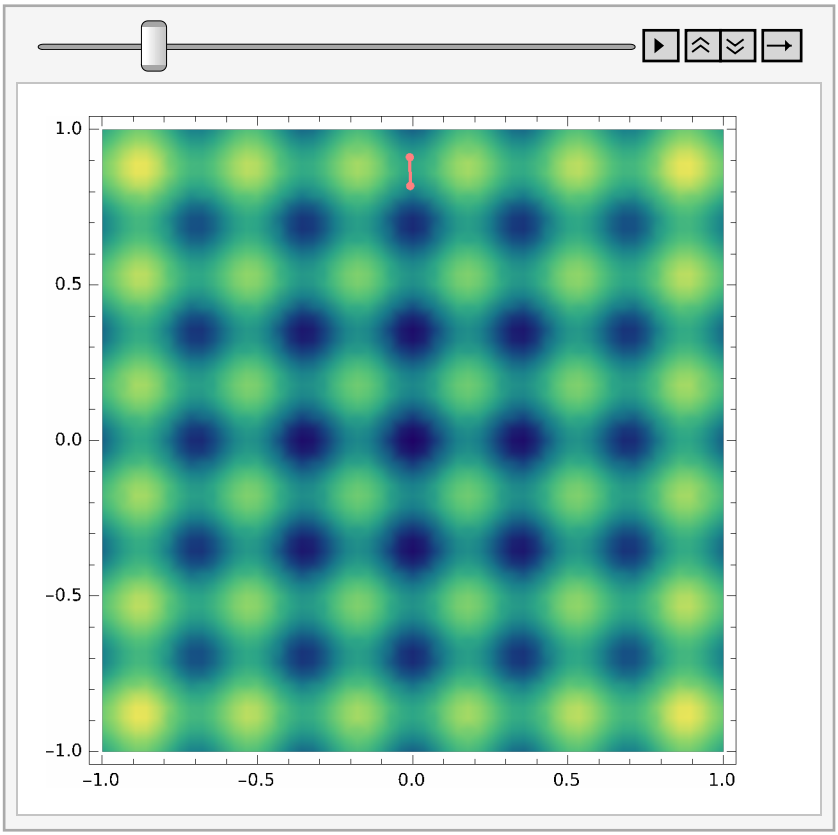
Przykład 4

In[12]:=
```
Clear[dx, dy, n, it, solution, epsilon]
RastraginFunction2[{x_, y_}] := x^2 + y^2 - Cos[18 * x] - Cos[18 * y]
dx = {-1, 1};
dy = {-1, 1};
epsilon = 0.1;
n = 10;
it = 10;
solution = NeighborhoodSearch[RastraginFunction2, dx, dy, epsilon, it, n];
Drawing[RastraginFunction2, solution, dx, dy]
```

Out[20]=

PROSTE PRZESZUKIWANIE LOKALNE	
Wzór funkcji	Wykres funkcji

$x^2 + y^2 - \mathrm{Cos}[18\,x] - \mathrm{Cos}[18\,y]$	
Przebieg wartości minimalnej w zależności od iteracji	Przebieg wartości średniej w zależności od ite
	
Najlepsza znaleziona wartość	Punkt, w którym znaleziono mir
-1.81848	{-0.0102668, 0.363253}
Przebieg poszukiwania rozwiązania	



Porównanie metod

Metoda poszukiwania ekstremum ze zmiennym sąsiedztwem jest lepszą metodą, ponieważ każdy kolejny punkt jest lepszy od poprzedniego.
Ilość znalezionych punktów jest zależna od losowania przez program.
W tym przypadku wartość sąsiedztwa (epsilon) może być dostatecznie mała, ponieważ w przypadku gdy lepszy punkt w tym sąsiedztwie nie został znaleziony to sąsiedztwo jest zwiększane z każdą iteracją. Sąsiedztwo coraz to lepszego punktu zapewnia nam również spadek wartości średnich w każdej iteracji.

W przypadku zmiennego sąsiedztwa wynik działania heurystyki również jest najbardziej efektywny dla funkcji zawierającej tylko jedno ekstremum lokalne, będące jednocześnie ekstremum globalnym - losuje wiele coraz to lepszych punktów, nie zagnieżdża się w żadnym minimum lokalnym, więc przechodząc przez pętlę częściej znajduje punkt niż zwiększa swoje sąsiedztwo.