МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»
Кафедра прикладної математики

Курсовий проект на тему:

«Метод Розенброка»

із дисципліни

«Дослідження операцій»

Виконав:
Студент групи КМ-82
Карпілов В.І.

Керівник:
Ладогубець Т.С

Київ 2021

Зміст

ПОСТАНОВКА ЗАДАЧІ

Дослідити збіжність адаптивного методу випадкового пошуку зі зміним кроком при мінімізації кореневої функції в залежності від:

1. Виду метода одновимірного пошуку (ДСК-Пауелла або Золотого перетину).
2. Точності метода одновимірного пошуку.
3. Значення параметру в алгоритмі Свена.
4. Параметрів методу .
5. Модифікації методу.

Використати адаптивний метод випадкового пошуку зі зміним кроком в якості метода спуску для умовної оптимізації в залежності від:

1. Розташування локального мінімума (всередині / поза допустимою областю).
2. Виду допустимої області (випукла / невипукла / з лінійними обмеженнями).

ТЕОРЕТИЧНІ ВІДОМОСТІ

Основна ідея методу випадкового пошуку полягає в тому, що точку кожного пробного досвіду для вивчення поверхні відгуку в районі базової (початкової) точки вибирають випадковим чином (звідси і назва методу). Незважаючи на довільність вибору пробної точки, алгоритм випадкового пошуку дозволяє послідовно наближатися до екстремальної області. Досліди проводять у вихідній (початкової) точці і в випадково обраної пробної точці, вимірювання відгуку в них порівнюють і, якщо шукається максимум, роблять робочий крок в напрямку зростання цільової функції. Нову робочу точку приймають за нову початкову і знову вибирають пробну точку випадковим чином. Зазвичай довжина робочого кроку перевищує інтервал варіювання між нульовою і пробної точками.

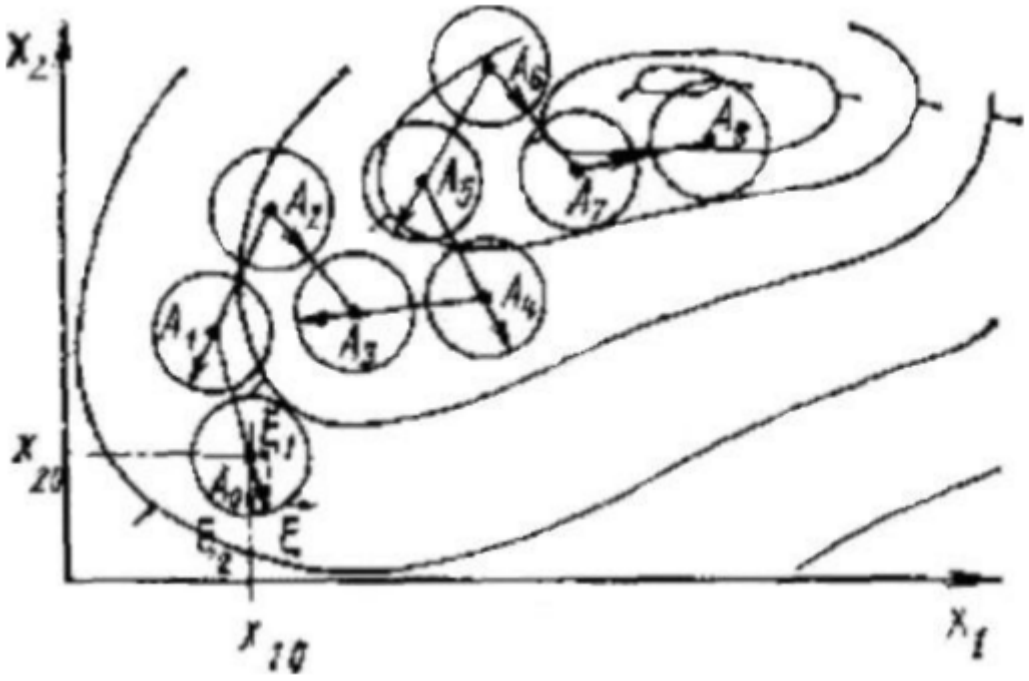


рис 1. Візуалізація роботи алгоритму

Випадкове значення точки розраховується наступним чином:

$$x^{(k+1)} = x^{(k)} + \lambda^{(k)} \left[\beta \frac{z^{(k)}}{\left\| z^{(k)} \right\|} + (1-\beta) r^{(k)} \right]$$

де

$\lambda^{(k)}$ – величина кроку (збільшується після успіху, зменшується після невдачі),

$r^{(k)}$ – одиничний вектор нормальних відхилень,

β – коефіцієнт, що змінюється у процесі пошуку.

Стратегія пошуку полягає у наступному:

Вибирається довільна точка
Береться вектор заданої довжини у випадковому напрямку

Якщо значення функції у отриманій точці менше початкового - вона стає новою початковою з збільшенням (для прискорення наближення)

Якщо протягом заданої кількості ітерацій функція не зменшується - зменшуємо крок

Практичні дослідження та висновки

В якості досліджуваної функції у даній курсовій роботі виступає функція Розенброка, яка відома, зокрема, завдяки тому, що вона має дуже вигнутий «яр» (рос. овраг). Такі «ярові» функції є досить важкими для дослідження, зокрема, використовуючи евристичні методі.

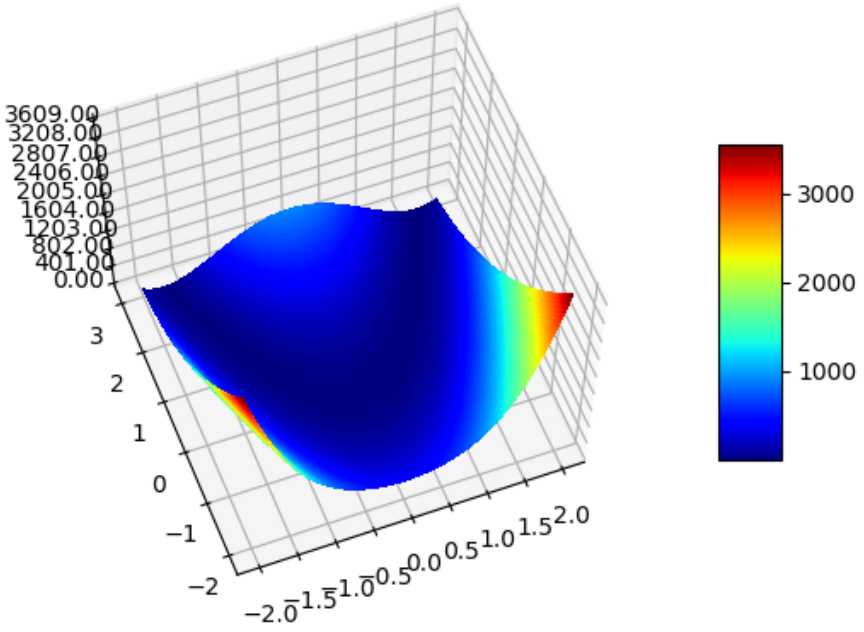


рис 2. Графік функції Розенброка

Функція Розеброка в околі точки мінімума

Програма була написана мові програмування Python з використанням бібліотеки numpy. Метод Розенброка був протестований на

функції Розенброка: $f\left(x_1,\ x_2\right) = 100\left(x_1^2\ -\ x_2\right)^2 + \left(1\ -\ x_1\right)^2$

Початкова точка $x^{(0)} = (-\ 1,2\ ;\ 0,0)$.

Опис параметрів функції

В якості початкового кроку було обрано
1e-2

Параметр який відповідає за довжину кроку на першій ітерації

В якості макс кількості випадкових векторів було обрано

1e+3

Параметр який відповідає за максимальну кількість ітерацій яку алгоритм може здійснити для пошуку точки, $x^{(k)}$

В якості ε

1e-3

Критерій закінчення

Основний критерій закінчення було обрано: $\|\Delta y\| \leq \varepsilon$ (1)

Оскільки він дає найбільшу точність, проте через особливості алгоритмів з випадковою генерацією - було додано ще один критерій зупинки $N > M$де N це кількість спроб зі зменшенням кроку під час яких не було знайдено жодної точки меншої, ніж наявний мінімум.

Виду метода одновимірного пошуку (ДСК-Пауелла або Золотого перетину)

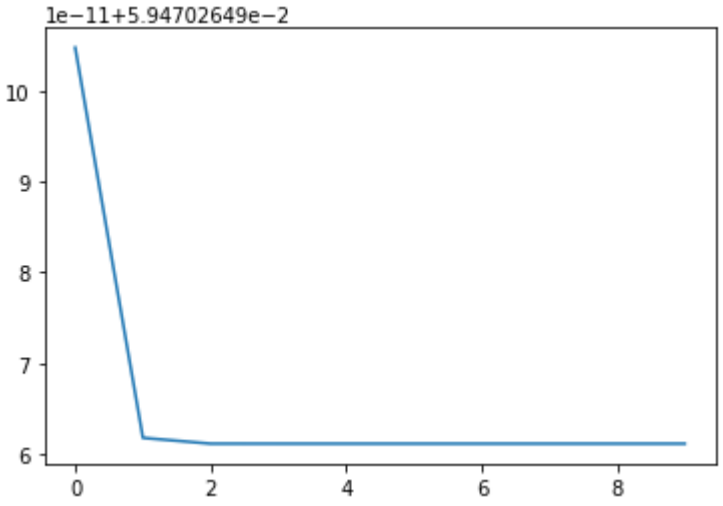


рис 3. Робота алгоритму з використанням методу золотого перетину

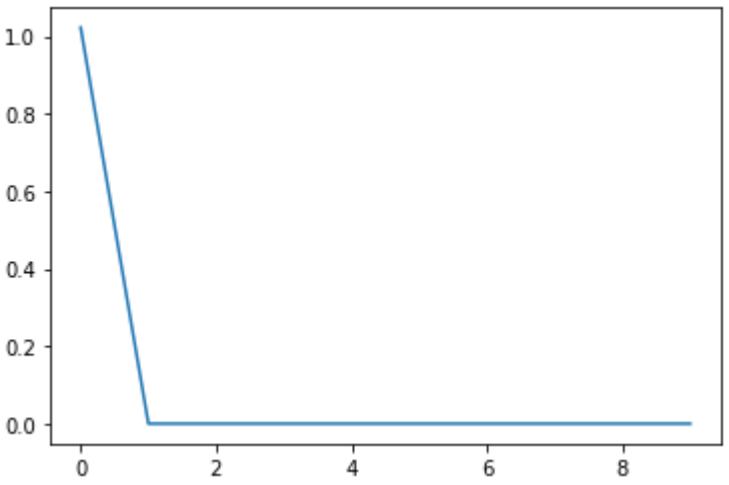


рис 4. Робота алгоритму з використанням методу Пауелла

При багатократних повторних запусках тенденція зберігалась - вибір методу для оптимізації кроку не сильно впливав на результат, оскільки сам алгоритм має дуже високу швидкість сходження і перерахування кроку відбувається не так часто.

Кодова реалізація

```
plotpoints = []
adaptiveSearch([-1.2,0],
               h=[1e-2,1e-2],
               n=1*10**5,
               R=1e-3,
               k=0.2,
               func= powFunction,
               h_diff = 1e-3,
               alpha = 1e-3,
               eps_singleDim = 0.0001,
               searchAlgorythm='g',
               max_deep=10)
plt.plot(plotpoints)
plt.show()
```

```
plotpoints = []
adaptiveSearch([-1.2,0],
               h=[1e-2,1e-2],
               n=1*10**5,
               R=1e-3,
               k=0.2,
               func= powFunction,
               h_diff = 1e-3,
               alpha = 1e-3,
               eps_singleDim = 0.0001,
               searchAlgorythm='d',
               max_deep=10)
plt.plot(plotpoints)
plt.show()
```

Точності метода одновимірного пошуку

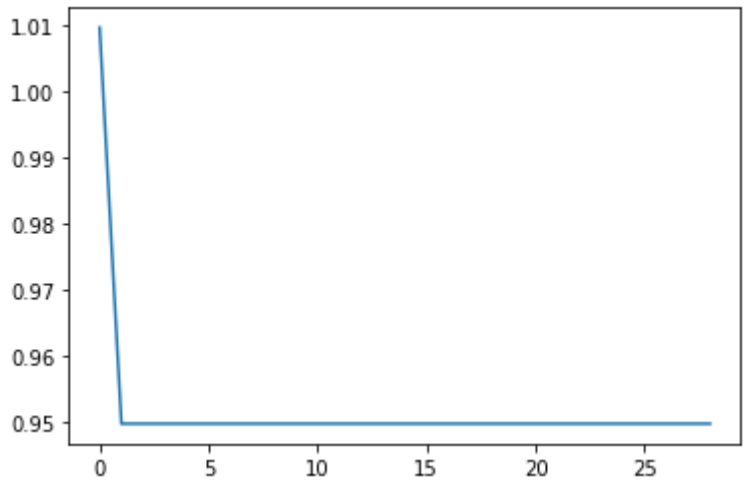


рис 5. Робота алгоритму з точністю метода одновимірного пошуку $\varepsilon=1$

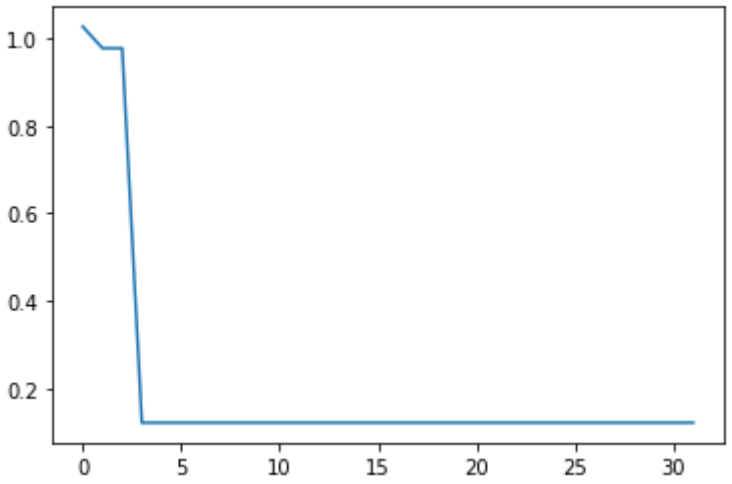


рис 5. Робота алгоритму з точністю метода одновимірного пошуку ε=0.1

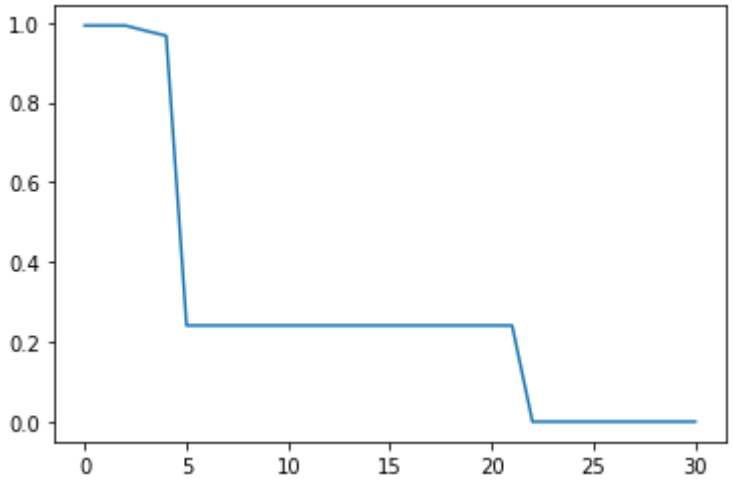


рис 5. Робота алгоритму з точністю метода одновимірного пошуку ε=0.01

Як ми бачимо точність методу одновимірного пошуку доволі сильно впливає на роботу алгоритму, це відбувається оскільки довжина вектору, який ми використовуємо для знаходження нової точки залежить від кроку і якщо ми наблизились до точки мінімуму нам дуже важливо підібрати кофіцієнт таким чином, аби на радіусі вектору знаходилась точка, яка буде менша ніж мінімум, який у нас є.

Також можна помітити цікаву особливість вірогіднісних алгоритмів - тривалість їх роботи не постійна і залежить не тільки від коефіцієнтів самого алгоритму, так ми бачимо що при точності метода одновимірного пошуку 0.1 алгоритму вдалось знайти спадну послідовність з 33-х точок, на відміну від запуску з точністю 0.01, де, хоч ми і отримали результат ближче до теоретичного, проте знайшли послідовність лише з 30-ти елементів.

Кодова реалізація

```
for i in range(3):
  plotpoints = []
  print(1*10**(-i))
  adaptiveSearch([-1.2,0],
                 h=[1*10**2,1*10**2],
                 n=1*10**3,
                 R=1*10**-i,
                 k=0.9,
                 func= powFunction,
                 h_diff = 1e-3,
                 alpha = 1e-3,
                 eps_singleDim = 1*10**-i,
                 searchAlgorythm='d',
                 max_deep=10)
  plt.plot(plotpoints)
  plt.show()
```

Значення параметру в алгоритмі Свена

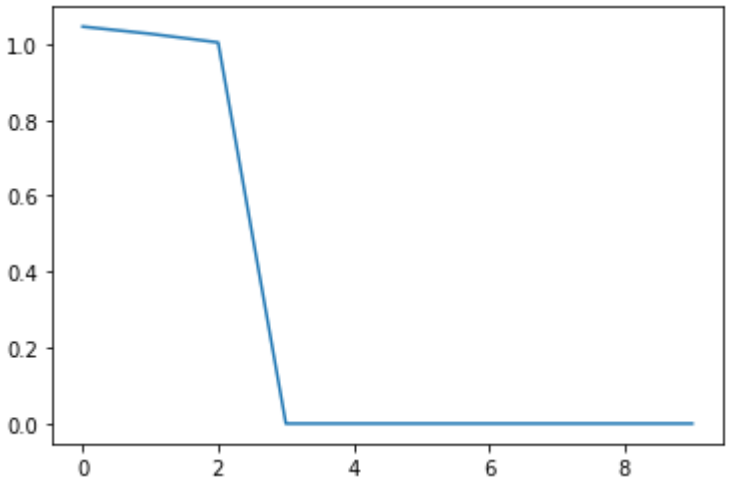


рис 6. Робота алгоритму з $\alpha_i = 1$

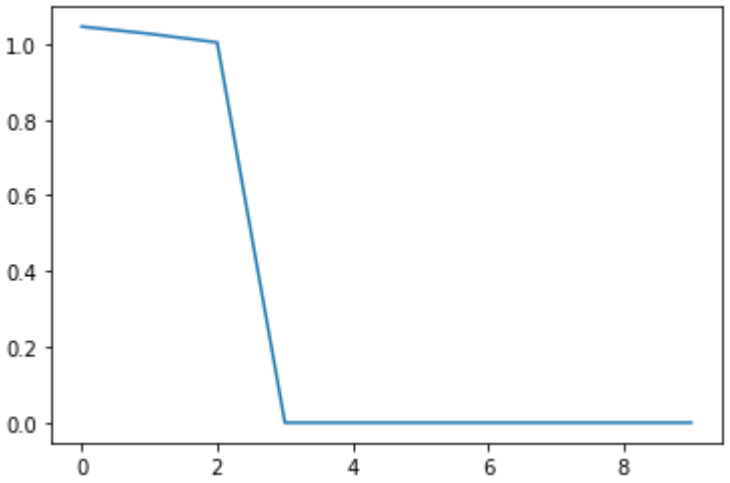


рис 7. Робота алгоритму з α= 0.1

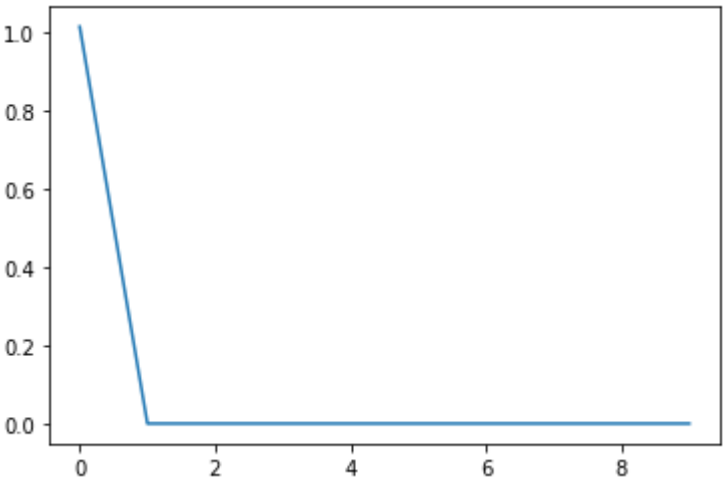


рис 8. Робота алгоритму з α= 0.01

Загалом параметр в алгоритмі свена не сильно впливає на результат, на відміну від точності метода одновимірного пошуку.

Кодова реалізація

```
for i in range(3):
  plotpoints = []
  adaptiveSearch([-1.2,0], h=[1e-3,1e-3],
                 n=1*10**5, R=1e-3,
                 k=0.9, func= powFunction,h_diff = 1e-3,
                 alpha = 1*10**-i, eps_singleDim = 1e-2,
                 searchAlgorythm='d', max_deep=10)
  plt.plot(plotpoints)
  plt.show()
```

Залежність від параметрів методу

Залежність від початкового кроку

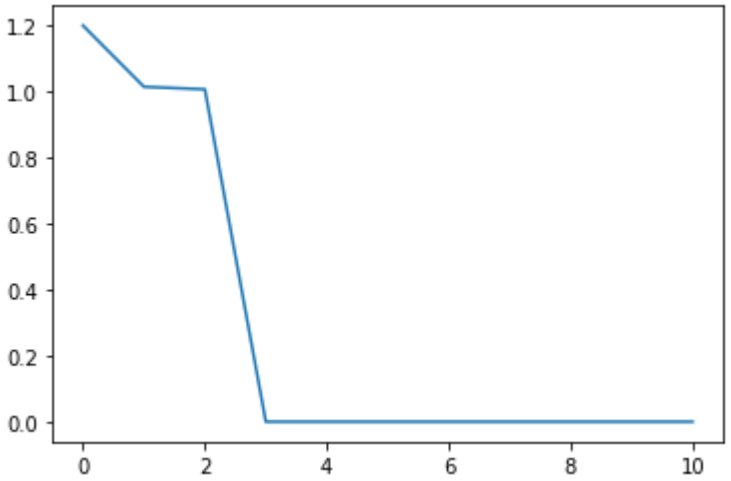


рис 8. Робота алгоритму з h = 0.1

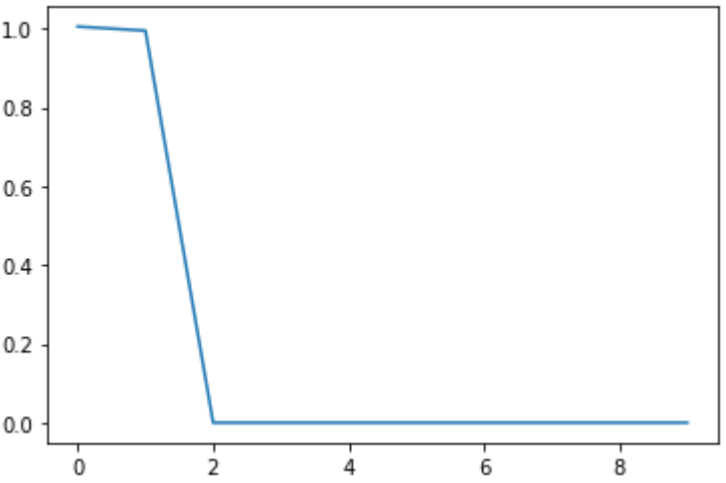


рис 9. Робота алгоритму з h = 0.01

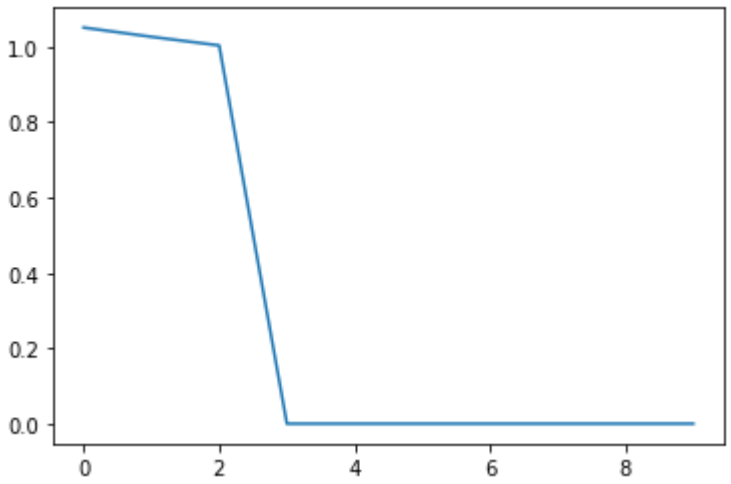


рис 9. Робота алгоритму з h = 0.001

Крок 1 не розглядався через те, що мінімуму на цьому радіусі алгоритм явно не знайде, а далі він підбере новий крок і ми не зможемо дослідити залежність саме від початкового кроку

Залежність від R

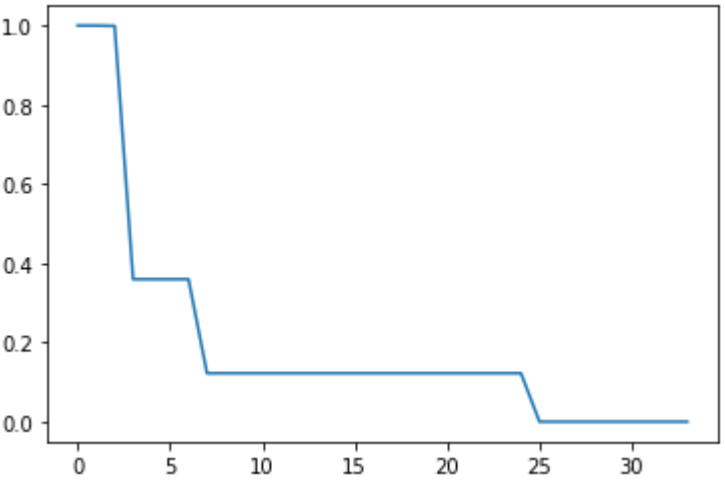


рис 9. Робота алгоритму з R = 0.1

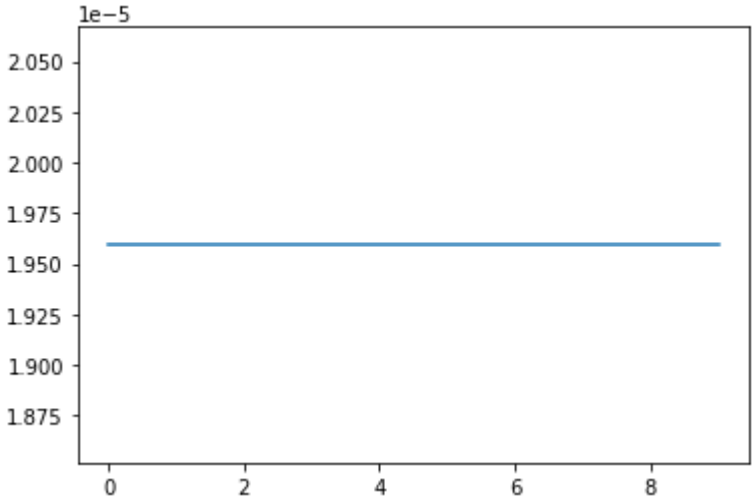


рис 10. Робота алгоритму з R = 0.01

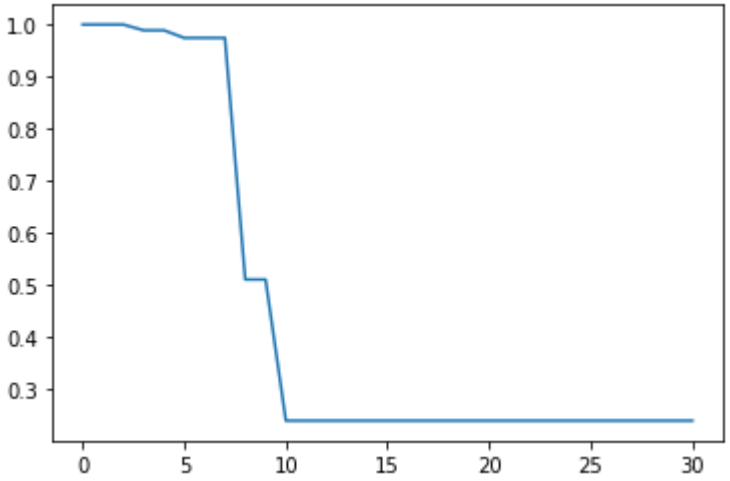


рис 11. Робота алгоритму з R = 0.001

На рис 10 зображено ще одну цікаву особливість випадкових алгоритмів, а саме - на перших ітераціях алгоритм отримав доволі точний результат, насправді це погано, оскільки в цей момент у нього ще досі доволі великий крок, через що він не встигне зменшити його достатньо швидко і припинить роботу, через відсутність прогресії, проте результат був отриманий за мінімальну кількість операцій.

На рис 9 ми бачимо зворотню ситуацію, поступове наближення до результату дає нам отримати доволі довгу послідовність, проте не надто точний результат через низьку задану точність в конкретно даному випадку (R=0.1).

На ри 11 типова робота цього алгоритму - поступове скачко-подібне наближення до мінімуму

Очевидно що при збільшенні точності ми збільшуємо кількість ітерацій та середній час роботи алгоритму, проте завдяки своїм особливостям, як було продемонстровано у випадку 2 - це не завжди має сенс і на великих об’ємах даних це може бути не виправданою тратою ресурсів

Кодова реалізація

```
for i in range(3):
  plotpoints = []
  adaptiveSearch([-1.2,0], h=[1*10**-i,1*10**-i],
                 n=1*10**3, R=1e-8, k=0.9,
                 func= powFunction,h_diff = 1e-3,
                 alpha = 1e-3, eps_singleDim = 1e-2,
                 searchAlgorythm='d', max_deep=10)
  plt.plot(plotpoints)
  plt.show()
```

```
for i in range(3):
  plotpoints = []
  adaptiveSearch([-1.2,0], h=[1*10**2,1*10**2],
                 n=1*10**3, R=1*10**-i,
                 k=0.9, func= powFunction,h_diff = 1e-3,
                 alpha = 1e-3, eps_singleDim = 1e-2,
                 searchAlgorythm='d', max_deep=10)
  plt.plot(plotpoints)
  plt.show()
```

Модифікації методу.

У якості модифікації методу було обрано змінити критерій зупинки і відключити можливість зупинки алгоритму через перевищення кількості ітерацій без прогресії

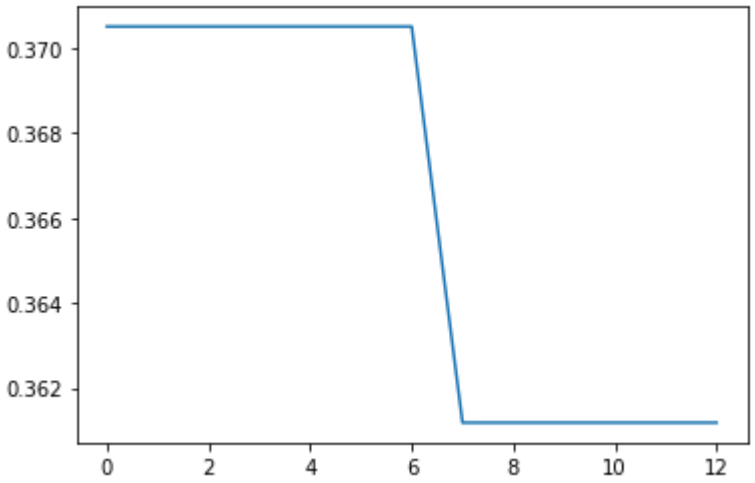


рис 12. Робота алгоритму з `useMaxDeep=False`

Результат - збільшення кількість ітерацій

Кодова реалізація

```
plotpoints = []
adaptiveSearch([-1.2,0], h=[1*10**2,1*10**2],
               n=1*10**3, R=1*10**-3, k=0.9,
               func= powFunction,h_diff = 1e-3,
               alpha = 1e-3, eps_singleDim = 1e-2,
               searchAlgorythm='d', max_deep=10,
               constraint=constraint, useMaxDeep=False)
plt.plot(plotpoints)
plt.show()
```

Розташування локального мінімума

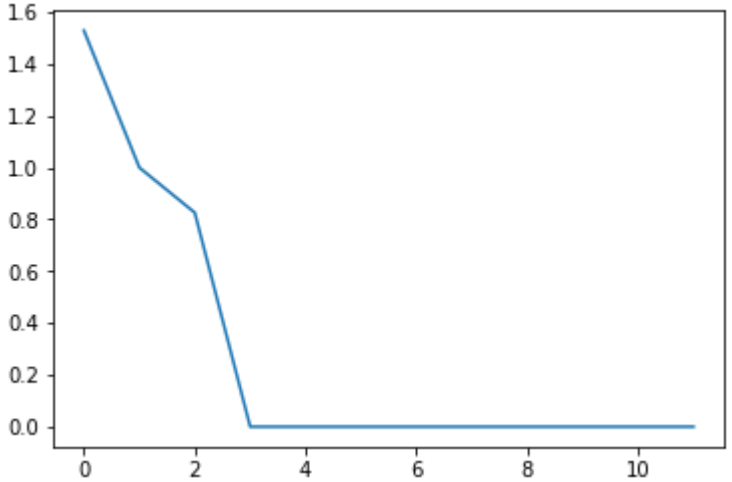


рис 13. Робота алгоритму з локальним мінімумом в середині області

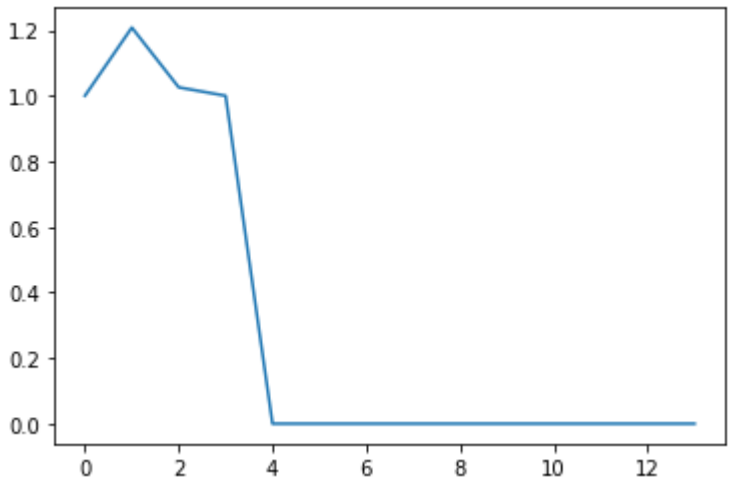


рис 14. Робота алгоритму з локальним мінімумом ззовні області

```
def constraint(x):
  return True

plotpoints = []
adaptiveSearch([-1.2,0], h=[1*10**2,1*10**2],
               n=1*10**3, R=1*10**-3, k=0.9,
               func= powFunction,h_diff = 1e-3,
               alpha = 1e-3, eps_singleDim = 1e-2,
               searchAlgorythm='d', max_deep=10,
               constraint=constraint, useMaxDeep=True)
plt.plot(plotpoints)
plt.show()
```

```
def constraint(x):
  return (norm(x-[0,0]) > 0.05)

plotpoints = []
adaptiveSearch([-1.2,0], h=[1*10**2,1*10**2],
               n=1*10**3, R=1*10**-3, k=0.9,
               func= powFunction,h_diff = 1e-3,
               alpha = 1e-3, eps_singleDim = 1e-2,
               searchAlgorythm='d', max_deep=10,
               constraint=constraint, useMaxDeep=True)
plt.plot(plotpoints)
plt.show()
```

Виду допустимої області

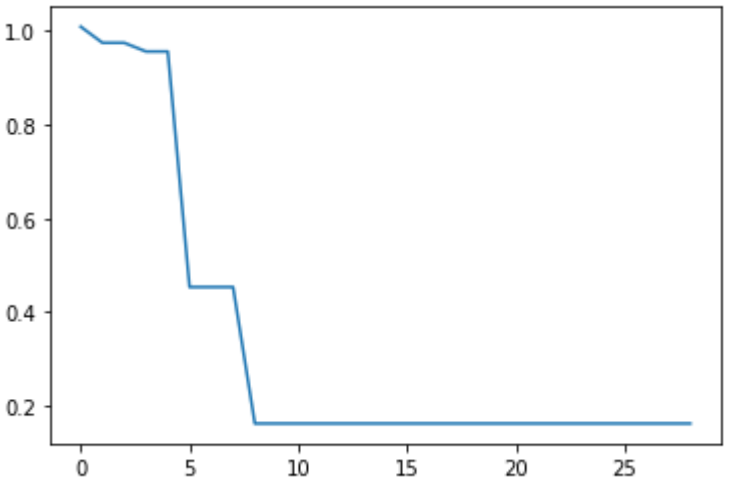


рис 15. Робота алгоритму у випуклій допустимій області

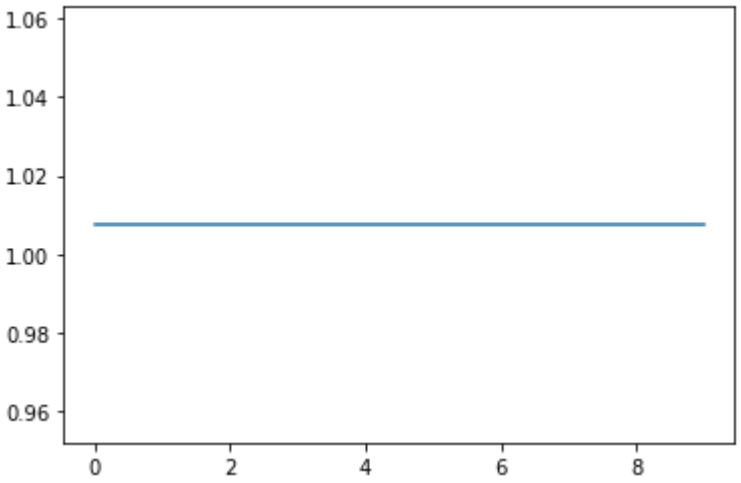


рис 16. Робота алгоритму у впуклій допустимій області

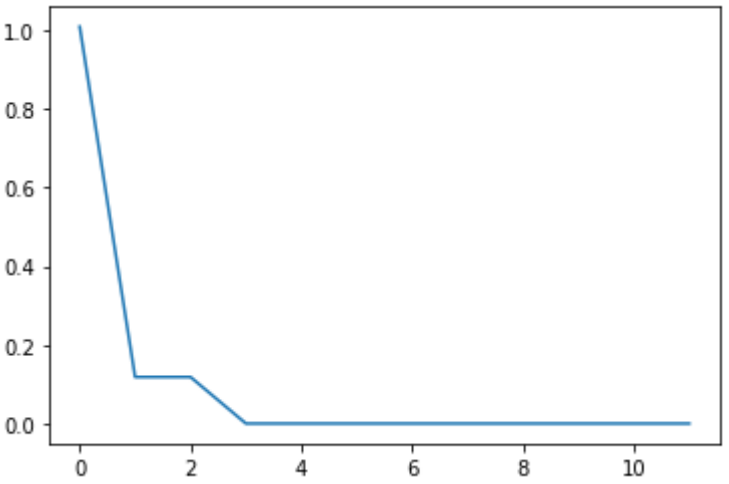


рис 17. Робота алгоритму у допустимій області з лінійними обмеженнями

Чомусь загалом наявність обмежень ніяк не впливає на роботу алгоритму (оскільки всередині одного кроку відбувається багато ітерацій), проте ситуація інакша з впуклою допустимою областю, де ситуація, коли алгоритм зупиняється після перших кроків трапляється частіше ніж при інших умовах.

```
def constraint(x):
  return (x[0] + 2)**2 - x[1]**2 + 4 > 0

plotpoints = []
adaptiveSearch([-1.2,0],
               h=[1*10**2,1*10**2],
               n=1*10**3, R=1*10**-3,
               k=0.9, func= powFunction,
               h_diff = 1e-3, alpha = 1e-3,
               eps_singleDim = 1e-2, searchAlgorythm='d',
               max_deep=10, constraint=constraint,
               useMaxDeep=True)
plt.plot(plotpoints)
plt.show()
```

```
def constraint(x):
  return (x[0] + 2)**2 - x[1]**2 + 4 > 0 and (x[0] + 2)**2 - x[1]**2 - 4 > 0

plotpoints = []
adaptiveSearch([-1.2,0], h=[1*10**2,1*10**2],
               n=1*10**3, R=1*10**-3, k=0.9,
               func= powFunction,h_diff = 1e-3,
               alpha = 1e-3, eps_singleDim = 1e-2,
               searchAlgorythm='d', max_deep=10,
               constraint=constraint, useMaxDeep=True)
plt.plot(plotpoints)
plt.show()
```

```
def constraint(x):
  return x[0] <1 and x[0] > -2

plotpoints = []
adaptiveSearch([-1.2,0], h=[1*10**2,1*10**2],
               n=1*10**3, R=1*10**-3, k=0.9,
               func= powFunction,h_diff = 1e-3,
               alpha = 1e-3, eps_singleDim = 1e-2,
               searchAlgorythm='d', max_deep=10,
               constraint=constraint, useMaxDeep=True)
plt.plot(plotpoints)
plt.show()
```

Висновок

У ході цієї курсової роботи було досліджено метод методу випадкового пошуку зі зміним кроком, було виявленно, що метод дуже погано працює в впуклими областями, також було виявленно, що найбільший вплив дають: вибір точності методу одномірного пошуку та вибір точності самого алгоритму, також було дослідженно що вибір самого алгоритму одномірного пошуку не дає великого впливу на результат, як і параметр алгоритму Свена.

Загалом методи випадкового пошуку - дуже цікава группа методів, особливо цікава особливість пошуку наближеного мінімуму за мінімальну кількість операцій, так у одному з прикладів вище, було знайдено наближений мінімум на перших 2-х кроках, таким чином алгоритми даної групи можна використовувати для функцій, справжній мінімум яких шукати надто складно, також адаптивний алгоритм має значний недолік, порівняно зі звичайним алгоритмом випадкового пошуку - набагато вижчу вірогідність “потрапити” у локальний мінімум, оскільки в такому випадку алгоритм почне зменшувати крок і не зможе “вийти” з нього.

Література

1. Алгоритмы / С. Дасгупта, Х. Пападимитриу, У. Вазирани; Пер. с англ. под ред. А. Шеня. — М.: МЦНМО, 2014. — 320 с

Додаток(код програми)

```
def powFunction(x):
  return (10*(x[0] - x[1])**2 + (x[0] - 1)**2)**4
```

```
powFunction([-2.22225999e-01, -3.78364743e-06])
```

```
15.608870254574194
```

```
import random
import sys
import numpy as np
import sympy as sp
numpy.set_printoptions(threshold=sys.maxsize)
from scipy.optimize import minimize
from sympy.matrices import Matrix
import matplotlib.pyplot as plt
```

```
def isLeftLower(f,x,h):
  if f(x + h) > f(x -h):
    return True
  return False
```

```
def findSectionEnd(f, x0, h, alpha):
  safer = 0
  hist = []
  hist.append(alpha)
  while safer<10000:
    y0 = f(x0)
    h = h*2
    x0 = x0 + h*alpha
    hist.append(alpha*2)
    if y0<f(x0):
      return x0, hist

  print("Overflow error")
```

```
def sven(f, x0, h, alpha):
  bufer = x0.copy()
  if isLeftLower(f, x0, h):
    return [[findSectionEnd(f, x0, h, -alpha)[0], bufer], findSectionEnd(f, x0, h, -alpha)[1]]
  return [[bufer, findSectionEnd(f, x0, h, alpha)[0]], findSectionEnd(f, x0, h, alpha)[1]]
```

```
def golden_section(interval, x0, S, f, eps):
    a, b = interval
    x1_gold = a + 0.382*(b - a)
    x2_gold = a + 0.618*(b - a)

    fx1_gold = f(x1_gold)
    fx2_gold = f(x2_gold)

    while True:
        if fx1_gold <= fx2_gold:
            b = x2_gold
            x2_gold = x1_gold
            x1_gold = a + 0.382*(b - a)
        else:
            a = x1_gold
            x1_gold = x2_gold
            x2_gold = a + 0.618*(b - a)
```

```
            x2_gold = a + 0.618*(b - a)
        if abs(norm(b - a)) <= eps:
            if fx1_gold < fx2_gold:
                answ = x1_gold
            else:
                answ = x2_gold
            break
        else:
          fx1_gold = f(x1_gold)
          fx2_gold = f(x2_gold)

    return answ
```

```
def dsk_paul(x0, S, f, eps, lambdas):
  x1_dsk, x2_dsk, x3_dsk = 0 ,lambdas[-2], lambdas[-1]
  fx1_dsk, fx2_dsk, fx3_dsk = f(x0 + x1_dsk), f(x0 + x2_dsk), f(x0 + x3_dsk)
  x1_dsk, x2_dsk, x3_dsk = x0 + x1_dsk ,x0 + x2_dsk, x0 + x3_dsk
  deltax_dsk = abs(norm(x1_dsk - x2_dsk))

  x11, x12 = Matrix(x0) + lmbd*Matrix(S)
  kvx = x2_dsk + (deltax_dsk * (fx2_dsk - fx3_dsk)) / (2*(fx1_dsk - 2*fx2_dsk + fx3_dsk))
  print(x11.subs(lmbd, kvx))
  fkvx = [f(x11.subs(lmbd, kvx), x12.subs(lmbd, kvx)])
  while True:
      if abs(fx2_dsk - fkvx) <= eps and abs(x2_dsk - kvx) <= eps:
          break
      else:
          if fkvx < fx2_dsk:
              if kvx > x2_dsk:
                  x1_dsk, x2_dsk = x2_dsk, kvx
                  fx1_dsk, fx2_dsk = fx2_dsk, fkvx
              else:
                  x2_dsk, x3_dsk = kvx, x2_dsk
                  fx2_dsk, fx3_dsk = fkvx, fx2_dsk
          else:
              if kvx > x2_dsk:
                  x3_dsk = kvx
                  fx3_dsk = fkvx
              else:
                  x1_dsk = kvx
                  fx1_dsk = fkvx
          a1 = (fx2_dsk - fx1_dsk) / (x2_dsk - x1_dsk)
          a2 = ((fx3_dsk-fx1_dsk)/(x3_dsk-x1_dsk) - (fx2_dsk-fx1_dsk)/(x2_dsk-x1_dsk)) / (x3_dsk - x2_dsk)
          kvx = (x1_dsk + x2_dsk) / 2 - (a1/(2*a2))
          fkvx = f([x11.subs(lmbd, kvx), x12.subs(lmbd, kvx)])
    # res = minimize(f, x0, method='powell',
    # options={'xtol': eps, 'disp': False})
    # return(res.direc[-1])
  return kvx
```

```
def find_df(f, x, i, h):
    return ( right(f, x, i, h) + left(f, x, i, h) ) / 2

def right(f, x, i, h):
    x_h = x.copy()
    x_h[i] += h
    return ( f(x_h) - f(x) ) / h
def left(f, x, i, h):
    x_h = x.copy()
    x_h[i] -= h
    return ( f(x) - f(x_h) ) / h
```

```
def norm(x):
    return (x[0]**2 + x[1]**2)**0.5
```

```
def findNear(x0, f, h_diff, alpha, eps_singleDim, searchAlgorythm):
  dfx1 = find_df(f, x0, 0, h_diff)
  dfx2 = find_df(f, x0, 1, h_diff)
  grad = np.array([[dfx1], [dfx2]])
  S0 = - grad
  X = x0.copy()
  S = S0.copy()

  if searchAlgorythm == 'g':
    interval = sven(f, X, h_diff, alpha)[0]
    return golden_section(interval, X, S, f, eps_singleDim)
  elif searchAlgorythm == 'd':
    interval, lambdas = sven(f, X, h_diff, alpha)
    return dsk_paul(x0 = X, s=S, f=f, eps = eps_singleDim, lambdas = lambdas)
```

```
def randomStep(x,h):
  res = np.array([float()] * 2)
  y = random.uniform(x[1]-h[1],x[1]+h[1])
  res[1] = y
  res[0] = (1-(res[1]-x[1])**2)**0.5 + x[0]
  return res
```

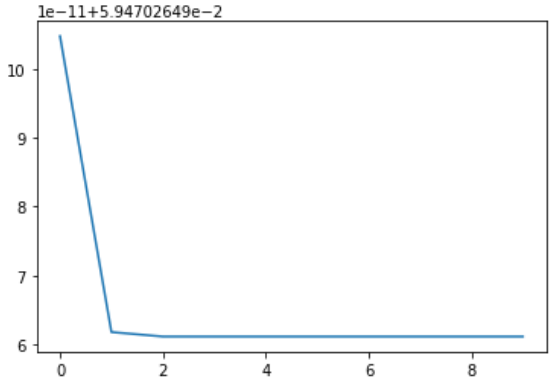
```
def adaptiveSearch(x, h, n, R, k, func, h_diff, alpha, eps_singleDim, searchAlgorythm, max_deep, useMaxDeep, constraint):
  y0 = func(x)
  ybof = y0
  xbof = x
  for i in range(n):
    step = randomStep(x, h)
    if y0 > func(step) and constraint(step):
      dfx1 = find_df(func, x, 0, h_diff)
      dfx2 = find_df(func, x, 1, h_diff)
      grad = np.array([[dfx1], [dfx2]])
      S0 = - grad
      S = S0.copy()
      y0 = func(step)
      x = step*(1/k)
      h =np.array(h)
      h = h*k
      i=0
      max_deep = 10
  h =np.array(h)
  h = h*k
  if max_deep==0:
    return min(func(x), func(xbof))
  elif ybof - func(step) < R:
    return min(func(x), func(xbof))
  x = findNear(x0=x, f=func, h_diff=h_diff, alpha=alpha, eps_singleDim=eps_singleDim, searchAlgorythm=searchAlgorythm)
  max_deep -= 1
  plotpoints.append(min(func(x), func(xbof)))
  if func(x) > func(xbof) or not constraint(x):
    x = xbof
  return adaptiveSearch(x, h, n, R, k, func, h_diff, alpha, eps_singleDim, searchAlgorythm, max_deep, useMaxDeep=True, const

def constraint(x):
  return True
```

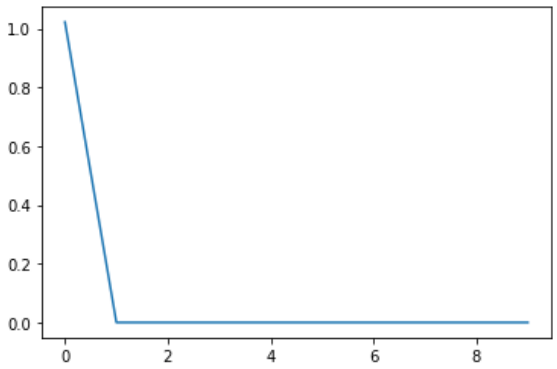
```
def constraint(x):
  return True
```

```
plotpoints = []
adaptiveSearch([-1.2,0],
               h=[1e-2,1e-2],
               n=1*10**5,
               R=1e-3,
               k=0.2,
               func= powFunction,
               h_diff = 1e-3,
               alpha = 1e-3,
               eps_singleDim = 0.0001,
               searchAlgorythm='g',
               max_deep=10)
plt.plot(plotpoints)
plt.show()
```

```
[0.05947026500475894, 0.05947026496179365, 0.05947026496115295, 0.059470264961151424, 0.059470264961151424
```

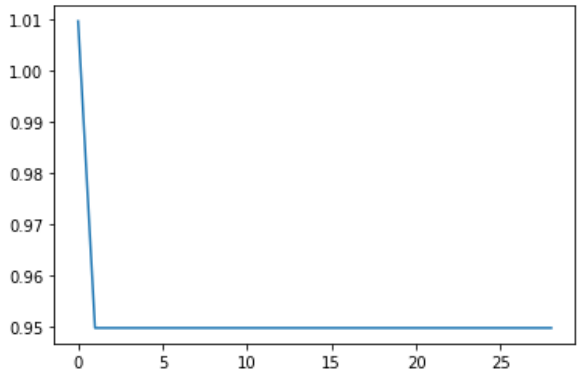



```
plotpoints = []
adaptiveSearch([-1.2,0], h=[1e-2,1e-2],
               n=1*10**5, R=1e-3, k=0.2,
               func= powFunction,h_diff = 1e-3,
               alpha = 1e-3, eps_singleDim = 0.0001,
               searchAlgorythm='d', max_deep=10)
plt.plot(plotpoints)
plt.show()
```

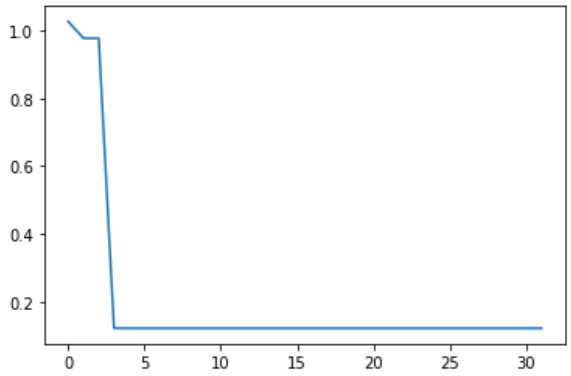


```
for i in range(3):
  plotpoints = []
  print(1*10**(-i))
  adaptiveSearch([-1.2,0],
                 h=[1*10**2,1*10**2],
                 n=1*10**3,
                 R=1*10**-i,
                 k=0.9,
                 func= powFunction,
                 h_diff = 1e-3,
                 alpha = 1e-3,
                 eps_singleDim = 1*10**-i,
                 searchAlgorythm='d',
                 max_deep=10)
  plt.plot(plotpoints)
  plt.show()
```

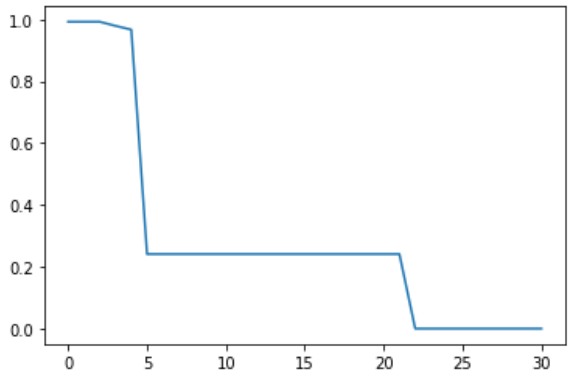
```
1
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: RuntimeWarning: invalid value encountere
  """
```



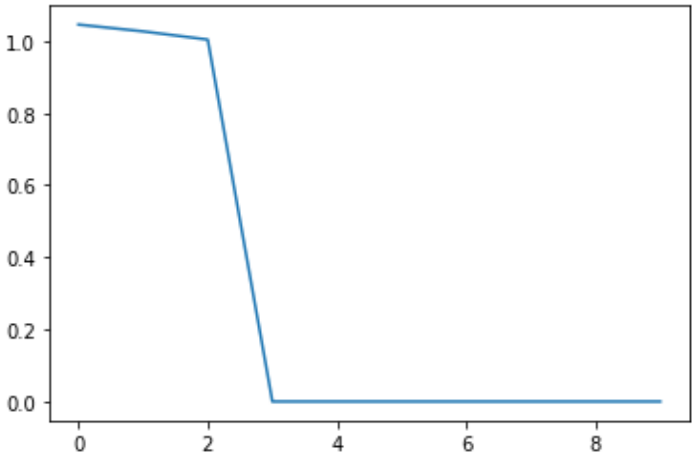

```
0.1
```

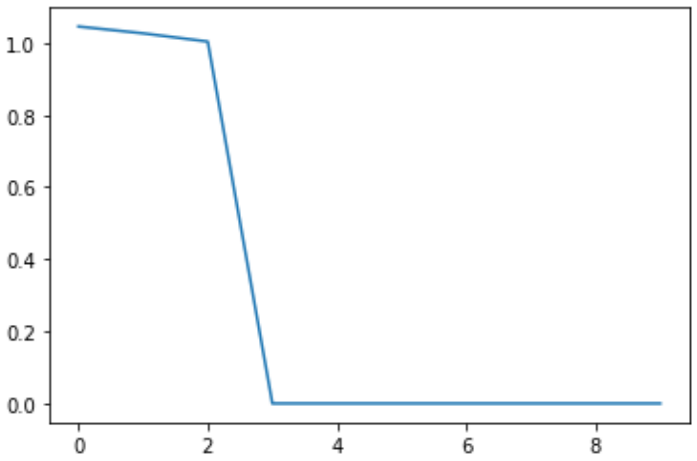


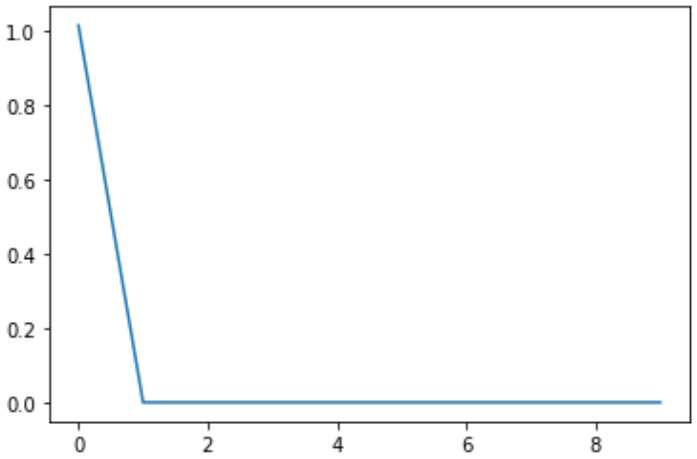

```
0.01
```



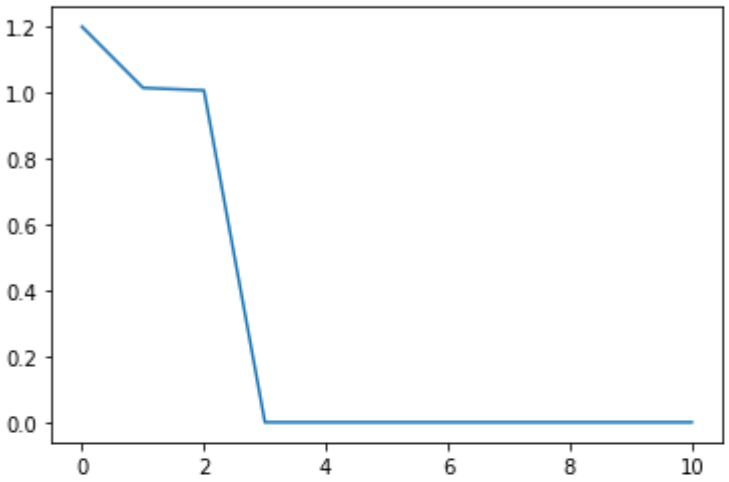
```
for i in range(3):
  plotpoints = []
  adaptiveSearch([-1.2,0], h=[1e-3,1e-3],
                 n=1*10**5, R=1e-3,
                 k=0.9, func= powFunction,h_diff = 1e-3,
                 alpha = 1*10**-i, eps_singleDim = 1e-2,
                 searchAlgorythm='d', max_deep=10)
  plt.plot(plotpoints)
  plt.show()
```

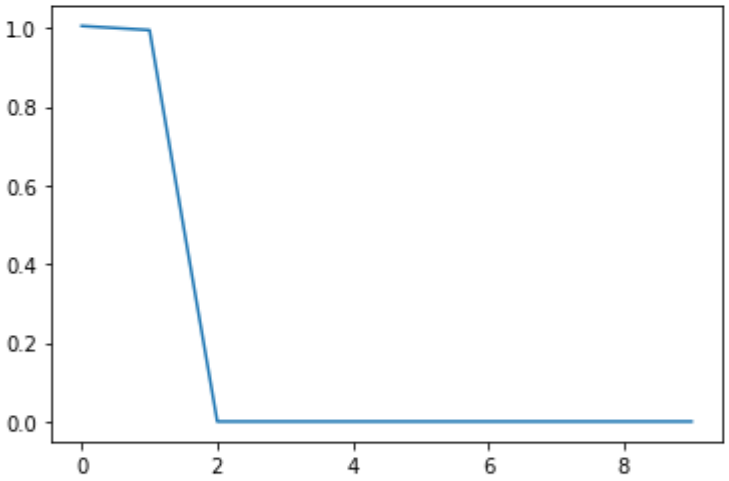


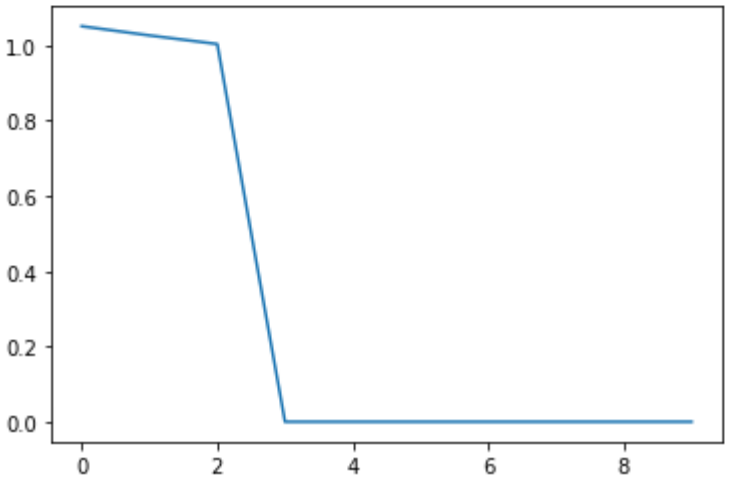




```
for i in range(3):
  plotpoints = []
  adaptiveSearch([-1.2,0], h=[1*10**-i,1*10**-i],
                 n=1*10**3, R=1e-8, k=0.9,
                 func= powFunction,h_diff = 1e-3,
                 alpha = 1e-3, eps_singleDim = 1e-2,
                 searchAlgorythm='d', max_deep=10)
  plt.plot(plotpoints)
  plt.show()
```

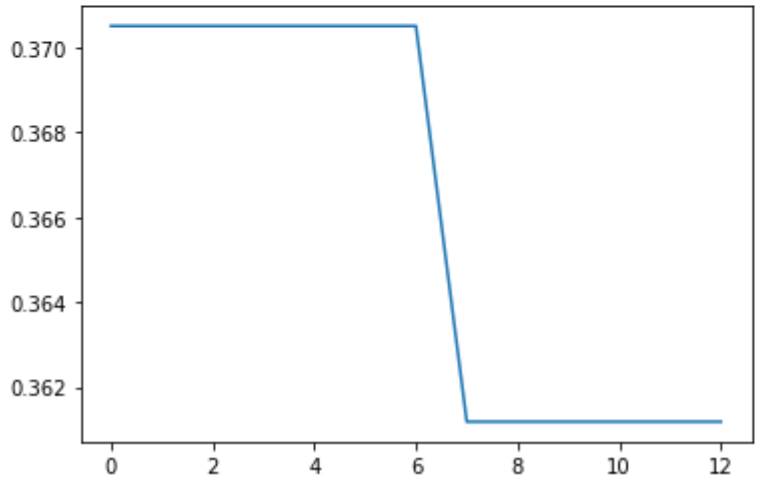






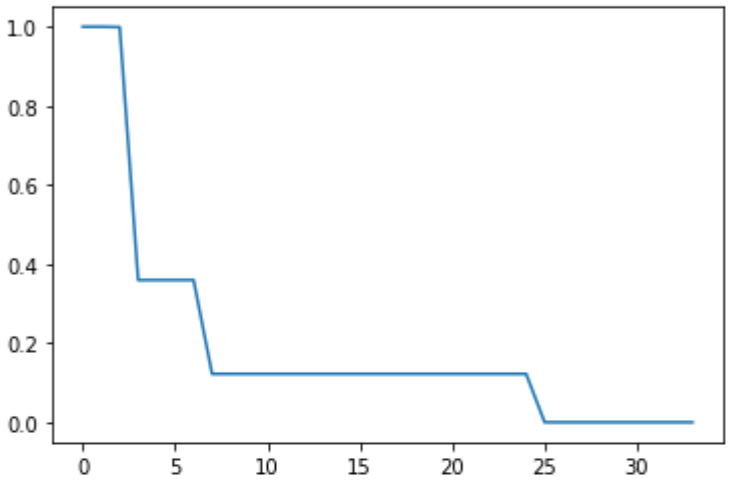
```
plotpoints = []
adaptiveSearch([-1.2,0], h=[1*10**2,1*10**2],
               n=1*10**3, R=1*10**-3, k=0.9,
               func= powFunction,h_diff = 1e-3,
               alpha = 1e-3, eps_singleDim = 1e-2,
               searchAlgorythm='d', max_deep=10,
               constraint=constraint, useMaxDeep=False)
plt.plot(plotpoints)
plt.show()
```

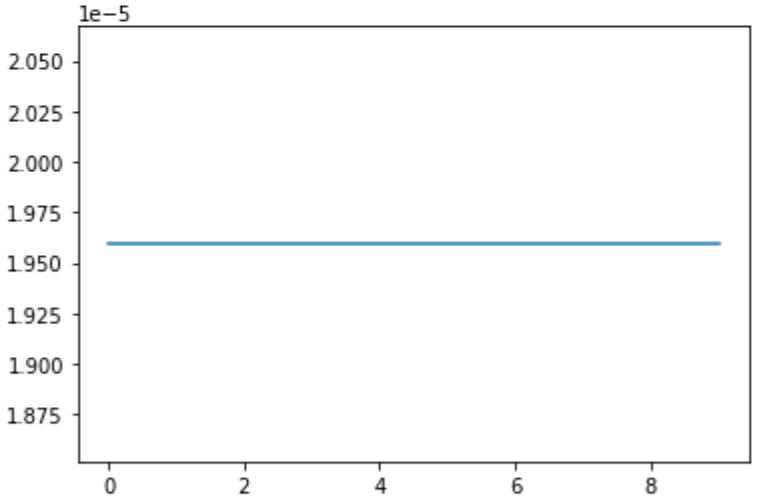
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: RuntimeWarr
  """
```

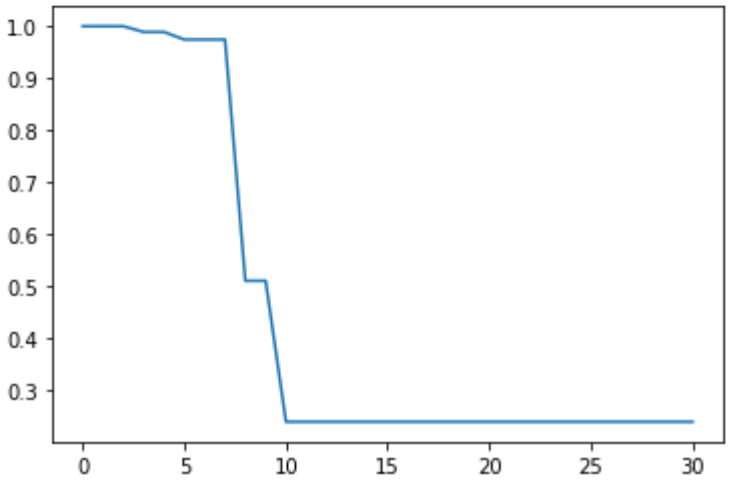


```
for i in range(3):
  plotpoints = []
  adaptiveSearch([-1.2,0], h=[1*10**2,1*10**2],
                 n=1*10**3, R=1*10**-i,
                 k=0.9, func= powFunction,h_diff = 1e-3,
                 alpha = 1e-3, eps_singleDim = 1e-2,
                 searchAlgorythm='d', max_deep=10)
  plt.plot(plotpoints)
  plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: RuntimeWarning: i
  """
```



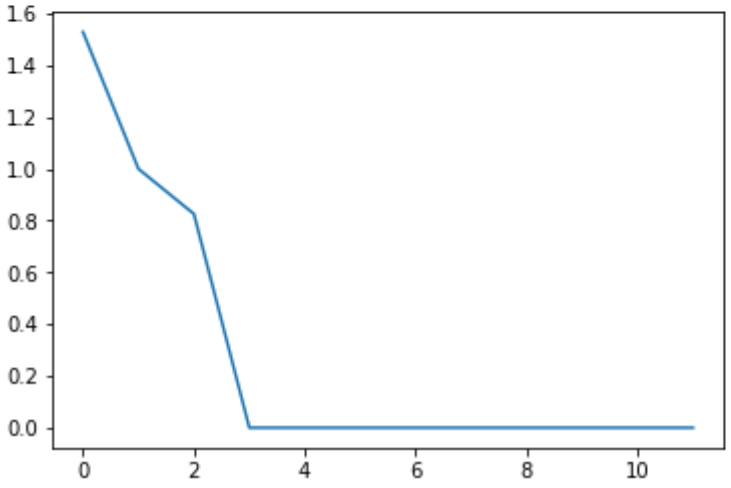




```python
def constraint(x):
  return True

plotpoints = []
adaptiveSearch([-1.2,0], h=[1*10**2,1*10**2],
               n=1*10**3, R=1*10**-3, k=0.9,
               func= powFunction,h_diff = 1e-3,
               alpha = 1e-3, eps_singleDim = 1e-2,
               searchAlgorythm='d', max_deep=10,
               constraint=constraint, useMaxDeep=True)
plt.plot(plotpoints)
plt.show()
```

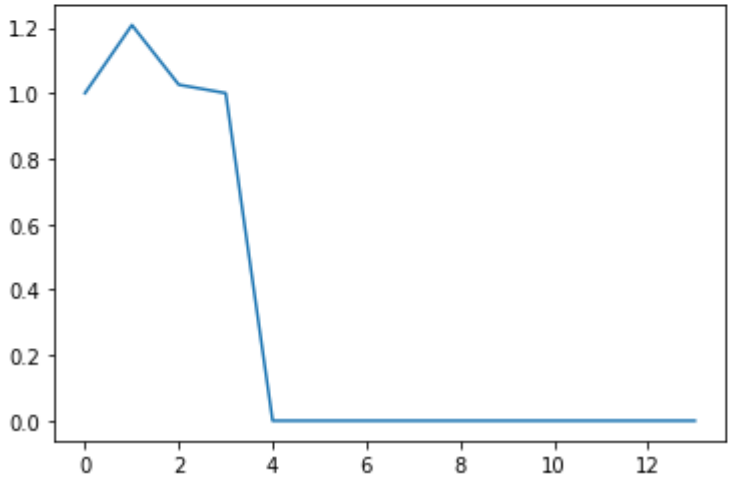
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: Runt
  """
```



```
def constraint(x):
  return (norm(x-[0,0]) > 0.05)

plotpoints = []
adaptiveSearch([-1.2,0], h=[1*10**2,1*10**2],
               n=1*10**3, R=1*10**-3, k=0.9,
               func= powFunction,h_diff = 1e-3,
               alpha = 1e-3, eps_singleDim = 1e-2,
               searchAlgorythm='d', max_deep=10,
               constraint=constraint, useMaxDeep=True)
plt.plot(plotpoints)
plt.show()
```

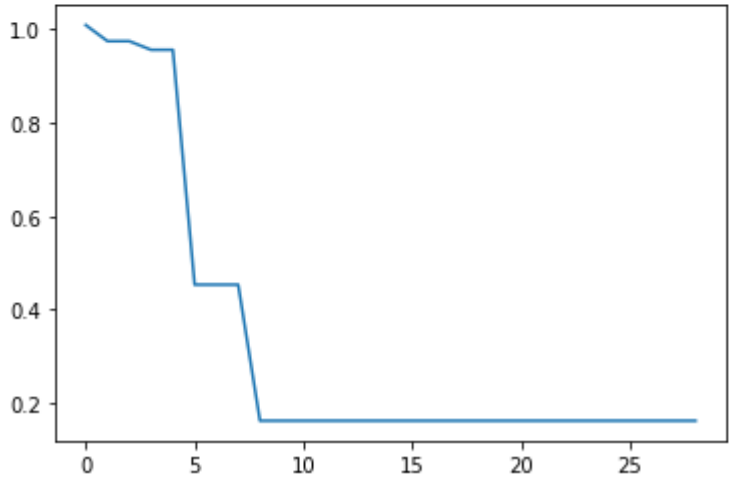
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: RuntimeWarn
  """
```



```
def constraint(x):
  return (x[0] + 2)**2 - x[1]**2 + 4 > 0

plotpoints = []
adaptiveSearch([-1.2,0],
               h=[1*10**2,1*10**2],
               n=1*10**3, R=1*10**-3,
               k=0.9, func= powFunction,
               h_diff = 1e-3, alpha = 1e-3,
               eps_singleDim = 1e-2, searchAlgorythm='d',
               max_deep=10, constraint=constraint,
               useMaxDeep=True)
plt.plot(plotpoints)
plt.show()
```

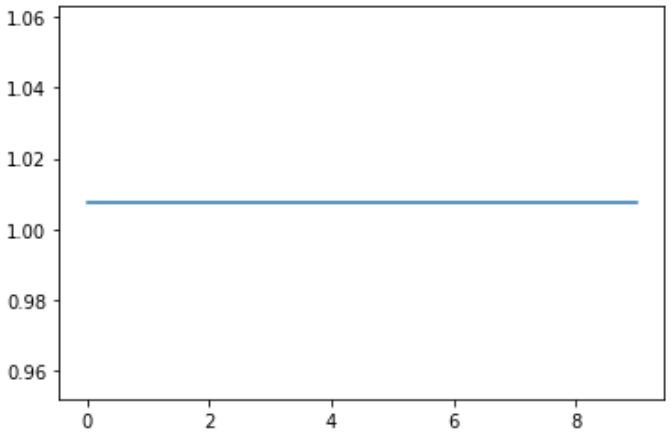
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: RuntimeWarning:
  """
```



```
def constraint(x):
  return (x[0] + 2)**2 - x[1]**2 + 4 > 0 and (x[0] + 2)**2 - x[1]**2 - 4 > 0

plotpoints = []
adaptiveSearch([-1.2,0], h=[1*10**2,1*10**2],
               n=1*10**3, R=1*10**-3, k=0.9,
               func= powFunction,h_diff = 1e-3,
               alpha = 1e-3, eps_singleDim = 1e-2,
               searchAlgorythm='d', max_deep=10,
               constraint=constraint, useMaxDeep=True)
plt.plot(plotpoints)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: RuntimeWarning: invalid va
  """
```



```
def constraint(x):
  return x[0] <1 and x[0] > -2

plotpoints = []
adaptiveSearch([-1.2,0], h=[1*10**2,1*10**2],
               n=1*10**3, R=1*10**-3, k=0.9,
               func= powFunction,h_diff = 1e-3,
               alpha = 1e-3, eps_singleDim = 1e-2,
               searchAlgorythm='d', max_deep=10,
               constraint=constraint, useMaxDeep=True)
plt.plot(plotpoints)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: RuntimeWarni
  """
```