In [17]:
```
import numpy as np
```

Зчитування даних з файлу

In [32]:
```
import csv
def read_surf2plot(name):
    with open(f'{name}.csv', 'r') as csvfile:
        reader = csv.reader(csvfile, delimiter=' ')
        z = []
        for idx, row in enumerate(reader):
            if idx == 0:
                l, step = map(int, row)
                continue
            if idx == 1:
                A = tuple(map(int, row))
                continue
            if idx == 2:
                B = tuple(map(int, row))
                continue
            z.append(list(map(float, row)))
    lim = ((l - 1) // 2) * step

    x = [i- lim + 1 * step for j in range(l)] for i in range(l)]
    y = [[- lim + j * step for j in range(l)] for i in range(l)]

    return np.array(x), np.array(y), np.array(z), lim, A, B, step
x, y, z, lim, A, B, step = read_surf2plot("example1")
```

Рахуємо відстані між кожною суміжною парою вершин

In [19]:
```
vertical = np.sqrt(np.diff(z, axis=0)**2 + step**2) #рахуємо всі вертикальні відстанні
horizontal = np.sqrt(np.diff(z, axis=1)**2 + step**2) #рахуємо всі горизонтальні відстанні
print("Vertical differences:\n",vertical)
print("Horizontal differences:\n",horizontal)
```

```
Vertical differences:
 [[20.43039114 22.42117191 24.25084771 ... 24.25084771 22.42117191
  20.43039114]
 [22.39891295 24.22850422 25.87923354 ... 25.87923354 24.22850422
  22.39891295]
 [24.20267406 25.85352089 27.31040412 ... 27.31040412 25.85352089
  24.20267406]
 ...
 [24.20267406 25.85352089 27.31040412 ... 27.31040412 25.85352089
  24.20267406]
 [22.39891295 24.22850422 25.87923354 ... 25.87923354 24.22850422
  22.39891295]
 [20.43039114 22.42117191 24.25084771 ... 24.25084771 22.42117191
  20.43039114]]
Horizontal differences:
 [[20.43039114 22.39891295 24.20267406 ... 24.20267406 22.39891295
  20.43039114]
 [22.42117191 24.22850422 25.85352089 ... 25.85352089 24.22850422
  22.42117191]
 [24.25084771 25.87923354 27.31040412 ... 27.31040412 25.87923354
  24.25084771]
 ...
 [24.25084771 25.87923354 27.31040412 ... 27.31040412 25.87923354
  24.25084771]
 [22.42117191 24.22850422 25.85352089 ... 25.85352089 24.22850422
  22.42117191]
 [20.43039114 22.39891295 24.20267406 ... 24.20267406 22.39891295
  20.43039114]]
```

Функції, які перевіряють, чи можна піти вверх, вниз, вправо, вліво

In [20]:
```
def left(inds, a):
    if inds[1]==0:
        return False
    return (inds[0], inds[1]-1)
def right(inds, max_l):
    if inds[1]==max_l[1]-1:
        return False
    return (inds[0], inds[1]+1)
def up(inds, a):
    if inds[0]==0:
        return False
    return (inds[0]-1, inds[1])
def down(inds, max_l):
    if inds[0]==max_l[0]-1:
        return False
    return (inds[0]+1, inds[1])
```

Ініціалізуємо першу початкову вершину, з якої будемо рахувати відстані

In [21]:
```
start = A #початкова точка. A - 11 індекс. Отримали з файлу, коли читали
print("Значення точки в початковому масиві ",z[start])
```

```
Значення точки в початковому масиві  -328.344977489863
```

Створюємо результуючий масив, де будуть відстані всіх точок до початкової

In [22]:
```
result = np.ones(z.shape) * np.inf #створюємо масив з нескінченностями
result[start] = 0 #відстань до початкової точки = 0
result
```

Out[22]:
```
array([[inf, inf, inf, ..., inf, inf, inf],
       [inf, inf, inf, ..., inf, inf, inf],
       [inf, inf, inf, ..., inf, inf, inf],
       ...,
       [inf, inf, inf, ..., inf, inf, inf],
       [inf, inf, inf, ..., inf, inf, inf],
       [inf, inf, inf, ..., inf, inf, inf]])
```

Рахуємо відстані від кожної точки до початкової за алгоритмом Дейкстри

In [23]:
```
from queue import PriorityQueue # для зчитування точок, повернення мінімальної
counted = PriorityQueue() #створення черги, враховуючи пріоритетність точок за їх висотами

counted.put((result[start], start)) #  додаванння першої точки та її індексу в чергу
hor_or_ver = {right:horizontal, down:vertical, left:horizontal, up:vertical}
max_l = z.shape
labeled = set() #сет вже обрахованих точок
while not counted.empty(): # поки черга не пуста
    start = counted.get()[1] #індекс мінімального елементу з counted
    while start in labeled: #якщо він вже обрахований, беремо наступний
        if counted.empty(): #вихід, якщо точки закінчились
            break
        start = counted.get()[1] #якщо ще є точки, то беремо наступну
    labeled.add(start) #додаємо нову точку до обрахованих і починаємо рахувати навколо неї
    indices = {right:start, down:start, left:(start[0], start[1]-1), up:(start[0]-1, start[1])}

    for change in [up, right, down, left]:#хід
        new_start = change(start, max_l) #дивимось, чи можна піти в тому напрямку
        if new_start and new_start not in labeled: # якщо можна піти, і навколо цієї точки раніше не рахували
            new_el = np.minimum(hor_or_ver[change][indices[change]] + result[start], result[new_start]) #рахуємо сусідні точки
            if not np.equal(new_el, result[new_start]): #якщо значення відрізняється від тої, що вже було(буде в більшості
                                                        #випадків, адже спочатку всі значення - безкінечність)
                result[new_start] = new_el #тоді присвоєємо нову відстань бо вона буде коротше
                counted.put((result[new_start], new_start)) #і додаємо цю обраховану відстань в нашу чергу на знаходження мінімального

result
```

Out[23]:
```
array([[15702.70051454, 15682.27012341, 15659.87121045, ...,
        18529.72539307, 18552.12439603, 18572.55469716],
       [15682.58697369, 15660.16580178, 15635.93729756, ...,
        18506.15253945, 18530.38104967, 18552.80221558],
       [15660.51458059, 15636.26366087, 15610.38442734, ...,
        18480.92820904, 18506.80744257, 18531.05829029],
       ...,
       [21382.8755    ,  21360.41506711, 21336.30699791, ...,
         25197.48676318, 25221.59483238, 25244.05526528],
       [21405.27441296, 21384.64357133, 21362.18623144, ...,
         25223.36599671, 25245.8233366 , 25266.45417823],
       [21425.70480409, 21407.86474324, 21386.43707916, ...,
         25247.61684443, 25268.24450851, 25286.88456937]])
```

Відстань до фінальної точки

In [24]:
```
result[B]
```

Out[24]:
```
14960.650117719979
```

Функція для знаходження індексів точок, по яким проходить найкоротший шлях

від кінцевої точки шукаємо звідки ми до неї рухаємось і заносимо ініціальну точку в список, який буде відображати найкоротший шлях по координатах

In [25]:
```
def way_2(matrix, hlong, vlong, finish):
    max_l = matrix.shape
    way = [finish]
    ind = finish
    while matrix[ind]!=0: # поки не дойдемо до початкової точки, шукаємо попередню вершину
        if up(ind, max_l):
            if np.round(matrix[ind] - vlong[ind[0]-1, ind[1]], 4) == np.round(matrix[ind[0]-1, ind[1]], 4):
                ind = ind[0]-1, ind[1]
                way.append(ind)

        if down(ind, max_l):
            if np.round(matrix[ind] - vlong[ind], 4) == np.round(matrix[ind[0]+1, ind[1]], 4):
                ind = ind[0]+1, ind[1]
                way.append(ind)

        if left(ind, max_l):
            if np.round(matrix[ind]- hlong[ind[0], ind[1]-1], 4)== np.round(matrix[ind[0], ind[1]-1], 4):
                ind = ind[0], ind[1]-1
                way.append(ind)

        if right(ind, max_l):
            if np.round(matrix[ind]- hlong[ind], 4) == np.round(matrix[ind[0], ind[1]+1], 4):
                ind = ind[0], ind[1]+1
                way.append(ind)

    return way
```

Використовуємо нашу функцію для знаходження індексів точок шляху

In [26]:
```
path = way_2(result, horizontal, vertical, B) #визначаємо ці індекси точок
len(path) #кількість точок, з яких складається шлях
```

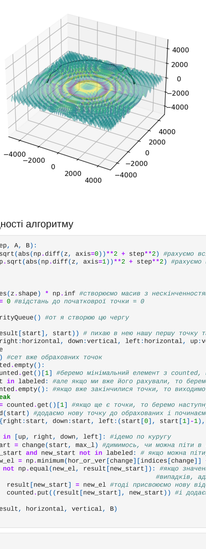
Out[26]:
```
2669
```

Функція для виводу даних та шляху у 3-D простір

In [27]:
```
import matplotlib.pyplot as plt
def plot(x, y, z, lim, path=None, opacity=0.5):
    fig = plt.figure()
    ax = plt.axes(projection='3d')

    ax.plot_surface(x, y, z, cmap="viridis", edgecolor='none', alpha=opacity)
    ax.set_title('Surface plot')
    ax.set_xlim(-lim, lim)
    ax.set_ylim(-lim, lim)

    ax.set_zlim(-lim, lim)

    if path:
        path_x = []
        path_y = []
        path_z = []
        for i, j in path:
            path_x.append(x[i][j])
            path_y.append(y[i][j])
            path_z.append(z[i][j] + 1000)

        ax.plot(path_x, path_y, path_z, c='k')

    plt.show()
```

Вивід даних та шляху

In [34]:
```
plot(x, y, z, lim, path)
```



Перевірка складності алгоритму

In [37]:
```
def dijkstra(z, step, A, B):
    vertical = np.sqrt(abs(np.diff(z, axis=0))**2 + step**2) #рахуємо всі вертикальні відстанні
    horizontal = np.sqrt(abs(np.diff(z, axis=1))**2 + step**2) #рахуємо всі горизонтальні відстанні

    start = A

    result = np.ones(z.shape) * np.inf #створюємо масив з нескінченностями
    result[start] = 0 #відстань до початкової точки = 0


    counted = PriorityQueue() #от я створюю цю чергу

    counted.put((result[start], start)) # пихаю в неє нашу першу точку та її індекси
    hor_or_ver = {right:horizontal, down:vertical, left:horizontal, up:vertical}
    max_l = z.shape
    labeled = set() #сет вже обрахованих точок
    while not counted.empty():
        start = counted.get()[1] #беремо мінімальний елемент з counted, причому лише його індес потрібен, тому [1]
        while start in labeled: #але якщо ми вже його рахували, то беремо наступний
            if counted.empty(): #якщо вже закінчилися точки, то виходимо
                break
            start = counted.get()[1] #якщо ще є точки, то беремо наступну
        labeled.add(start) #додаємо нову точку до обрахованих і починаємо вокруг неї рахувати
        indices = {right:start, down:start, left:(start[0], start[1]-1), up:(start[0]-1, start[1])}

        for change in [up, right, down, left]: #ідемо по кругу
            new_start = change(start, max_l) #дивимось, чи можна піти в тому напрямку
            if new_start and new_start not in labeled: # якщо можна піти, і ми вокруг цієї точки раніше не рахували
                new_el = np.minimum(hor_or_ver[change][indices[change]] + result[start], result[new_start]) #рахуємо точки вокруг
                if not np.equal(new_el, result[new_start]): #якщо значення відрізняється від тої, що вже було(буде в більшості
                                                            #випадків, адже спочатку всі значення - безкінечність)
                    result[new_start] = new_el #тоді присвоєємо нову відстань бо вона буде коротше
                    counted.put((result[new_start], new_start)) #і додаємо цю обраховану відстань в нашу чергу на знаходження мінімального

    path = way_2(result, horizontal, vertical, B)
    return path
```

In [40]:
```
import time
np.random.seed(0)
step = 1
A = (0, 0)
times = []
for x in np.linspace(30, 3000, 20):
    B = (int(x)-1, int(x)-1)
    start_time = time.time()
    z = np.random.randn(int(x)**2).reshape(int(x), int(x))
    dijkstra(z, step, A, B)
    times.append(time.time() - start_time)
```

In [34]:

```
plt.scatter(np.linspace(30, 3000, 20)**2, times)
plt.ylabel("Час виконання в секундах")
plt.xlabel("Мільйонів точок в масиві");
```