# 2.8.2 天空视域因子计算与内存管理

### 2.8.2.1 天空视域因子（Sky View Factor，SVF）计算方法

天空视域因子用于描述三维空间形态的数值，是空间中某一点上可见天空与以该点为中心整个半球之间的比率。通常值趋近于1，表明视域开阔；趋近于0，则封闭。天空视域因子广泛应用于城市热岛效应，城市能力平衡，和城市小气候等相关的研究中。虽然QGIS等平台也提供有SVF计算工具，例如QGIS:SAGA:Sky View Factor，但是计算大尺度，高分辨率DSM栅格数据的SVF通常会溢出，且计算缺乏灵活性。因此有必要直接在python下定义SVF计算方法。利用DSM获取城市下垫面SVF的计算公式为：γinγi​值。

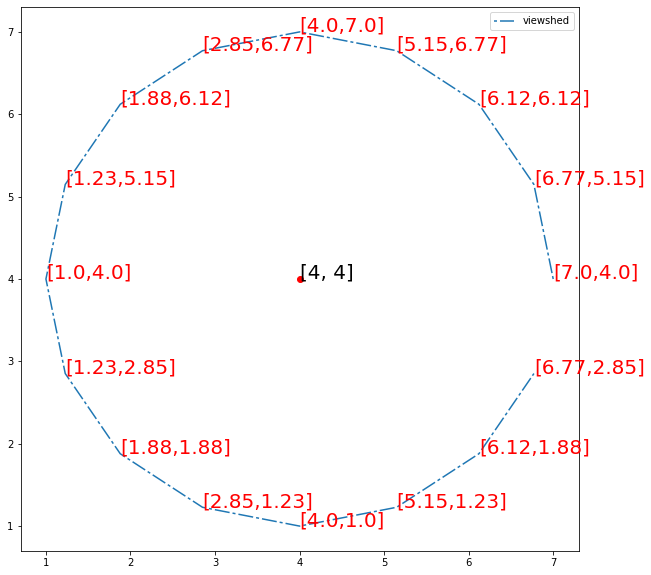
为最终实现基于DSM计算SVF的代码编写，首先通过自定义一个小数据量的数组（代表DSM栅格），根据上述计算过程编写代码，快速的实现和检验方法的可行性后，再将其迁移到最终定义的SVF计算的类中。SVF计算的基本参数包括观察点、基于观察点向四周环顾一周的视线数量及其距离半径（视距），每根视线等分的数量。程序中比较关键的代码行是计算不同点或位置的坐标值，并将其转换为基于栅格（数组）的相对坐标值（整数型）。前者直接使用三角函数方法计算，后者借助SciPy库的scipy.ndimage.map\_coordinates（）方法实现。

import numpy as np  
  
#A-配置基本参数  
z\_value=np.round(np.random.rand(10,10)\*0.5,2) #生成栅格（矩阵、数组）的随机高程值  
print("random z value:\n{}".format(z\_value))   
observation\_spot=[4,4] #定义观察点  
division\_num=5 #视线等分的数量  
sight\_distance=3 #视线距离（视距）  
sight\_line\_num=16 #环顾一周视线数

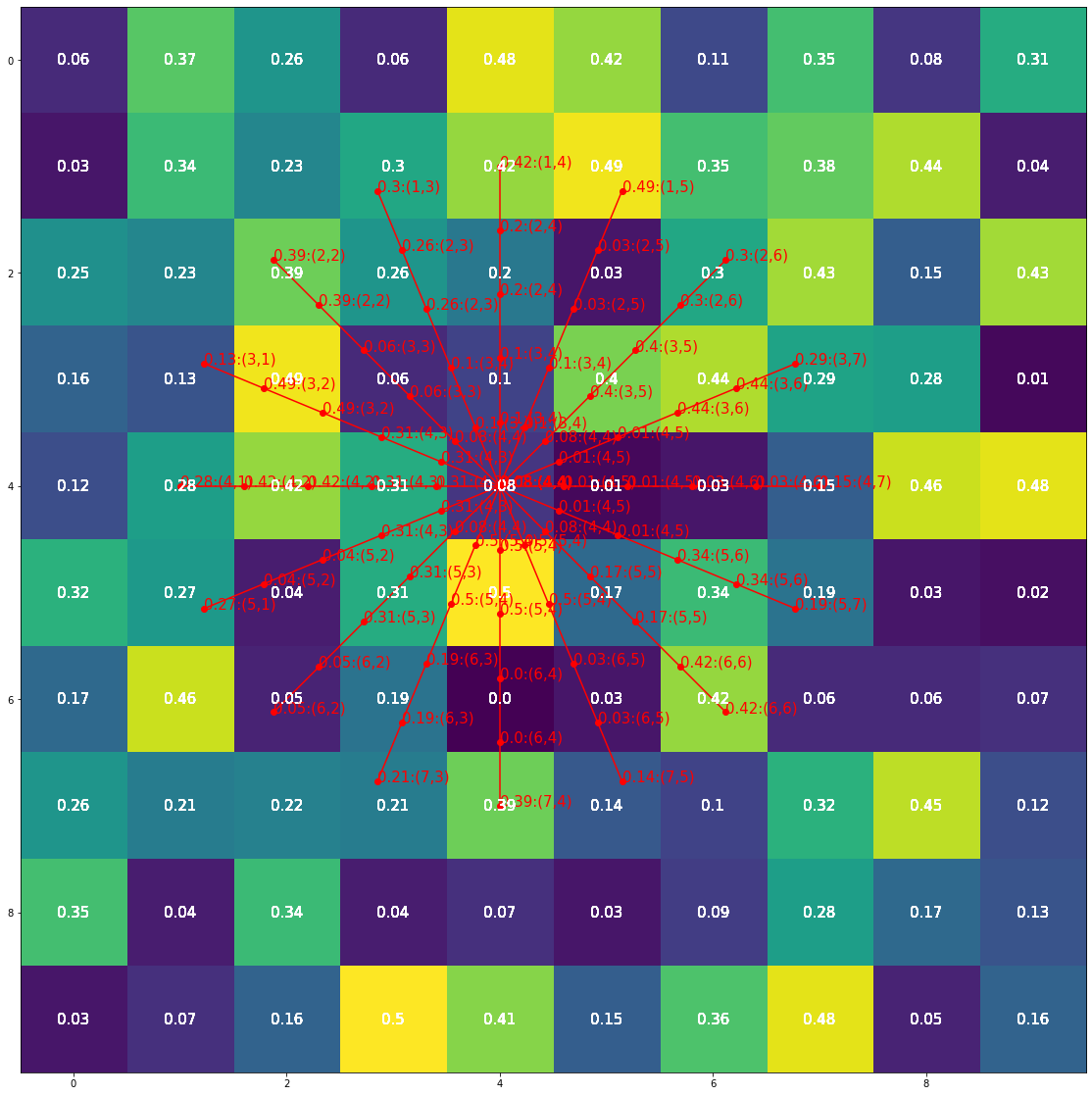
random z value:  
[[0.06 0.37 0.26 0.06 0.48 0.42 0.11 0.35 0.08 0.31]  
 [0.03 0.34 0.23 0.3 0.42 0.49 0.35 0.38 0.44 0.04]  
 [0.25 0.23 0.39 0.26 0.2 0.03 0.3 0.43 0.15 0.43]  
 [0.16 0.13 0.49 0.06 0.1 0.4 0.44 0.29 0.28 0.01]  
 [0.12 0.28 0.42 0.31 0.08 0.01 0.03 0.15 0.46 0.48]  
 [0.32 0.27 0.04 0.31 0.5 0.17 0.34 0.19 0.03 0.02]  
 [0.17 0.46 0.05 0.19 0. 0.03 0.42 0.06 0.06 0.07]  
 [0.26 0.21 0.22 0.21 0.39 0.14 0.1 0.32 0.45 0.12]  
 [0.35 0.04 0.34 0.04 0.07 0.03 0.09 0.28 0.17 0.13]  
 [0.03 0.07 0.16 0.5 0.41 0.15 0.36 0.48 0.05 0.16]]

#B-等分视域，获取点坐标  
def circle\_division(observation\_spot,sight\_distance,sight\_line\_num):  
 import math  
 import matplotlib.pyplot as plt  
 '''  
 function - 给定观测位置点，视距和视线数量，等分圆，返回等分坐标点列表  
  
 Paras:  
 observation\_spot - 观察点  
 sight\_distance - 视线距离（视距）  
 sight\_line\_num - 环顾一周视线数   
 '''   
 angle\_s=360/sight\_line\_num  
 angle\_list=[i\*angle\_s for i in range(sight\_line\_num)]  
 print("angle\_list={}".format(angle\_list))  
 coordi\_list=[]  
 for angle in angle\_list:  
 opposite=math.sin(math.radians(angle))\*sight\_distance  
 adjacent=math.cos(math.radians(angle))\*sight\_distance  
 coordi\_list.append((adjacent+observation\_spot[0],opposite+observation\_spot[1]))  
   
 fig, ax=plt.subplots(figsize=(10, 10))  
 x=[i[0] for i in coordi\_list]  
 y=[i[1] for i in coordi\_list]  
 # Using set\_dashes() to modify dashing of an existing line  
 line1,=ax.plot(x, y, label='viewshed')  
 line1.set\_dashes([2, 2, 10, 2]) # 2pt line, 2pt break, 10pt line, 2pt break   
 ax.legend()  
   
 for i in range(len(x)):  
 ax.text(x[i],y[i],"[%s,%s]"%(round(x[i],2),round(y[i],2)),fontsize=20,color="r")  
 ax.text(observation\_spot[0],observation\_spot[1],observation\_spot,fontsize=20)  
 ax.plot(observation\_spot[0],observation\_spot[1],"ro")  
 plt.show()  
  
 return coordi\_list  
  
coordi\_list=circle\_division(observation\_spot,sight\_distance,sight\_line\_num)

angle\_list=[0.0, 22.5, 45.0, 67.5, 90.0, 112.5, 135.0, 157.5, 180.0, 202.5, 225.0, 247.5, 270.0, 292.5, 315.0, 337.5]



#C-根据视线提取栅格（数组）对应位置的高程（数组对应位置值），即批量提取截面高程数据  
def line\_profile(z\_value,observation\_spot,end\_point,division\_num):   
 import scipy.ndimage   
 '''  
 function - 获取与视线相交单元栅格（数组对应位置）的栅格值（数组值）  
   
 Paras:  
 z\_value - 高程数组  
 observation\_spot - 观察点  
 end\_point - 视线末尾点  
 division\_num - 视线等分的数量  
 '''  
 z=z\_value   
 x0, y0=observation\_spot  
 x1, y1=end\_point  
 num=division\_num+1  
 x, y=np.linspace(x0, x1, num), np.linspace(y0, y1, num)  
   
 # Extract the values along the line, using cubic interpolation  
 zi=scipy.ndimage.map\_coordinates(z, np.vstack((x,y)),cval=0,mode="nearest",order=0) #Map the input array to new coordinates by interpolation.   
 x\_around=np.around(x).astype(int)  
 y\_around=np.around(y).astype(int)  
  
 return zi,(x,y)  
  
def combo\_profile(z\_value,observation\_spot,coordi\_list,division\_num):  
 import matplotlib.pyplot as plt  
 '''  
 function - 批量提取视线与高程数组（栅格）相交位置高程（数组对应位置值）  
   
 Paras:  
 z\_value - 高程数组  
 observation\_spot - 观察点  
 coordi\_list - 坐标值列表（视线末端点）  
 division\_num - 视线等分的数量  
 '''  
 z\_list=[]  
 sub\_coordi\_list=[]  
 for i in coordi\_list:  
 zi,sub\_coordi=line\_profile(z\_value,observation\_spot,i,division\_num)  
 z\_list.append(zi)  
 sub\_coordi\_list.append(sub\_coordi)  
  
 fig, axes = plt.subplots(nrows=1,figsize=z\_value.shape\*np.array([2]))  
 axes.imshow(z\_value)  
 for n in range(len(sub\_coordi\_list)):  
 x,y=sub\_coordi\_list[n]  
 axes.plot(y,x, 'ro-')   
 axes.axis('image')  
  
 for i in range(z\_value.shape[0]):  
 for j in range(z\_value.shape[1]):  
 axes.text(j, i, z\_value[i, j],ha="center", va="center", color="w",size=15)  
 for i in range(len(x)):  
 axes.text(y[i],x[i],"%s:(%s,%s)"%(round(z\_list[n][i],2),round(x[i]),round(y[i])),fontsize=15,color="r")  
   
 plt.show()   
 return z\_list  
  
z\_list=combo\_profile(z\_value,observation\_spot,coordi\_list,division\_num)

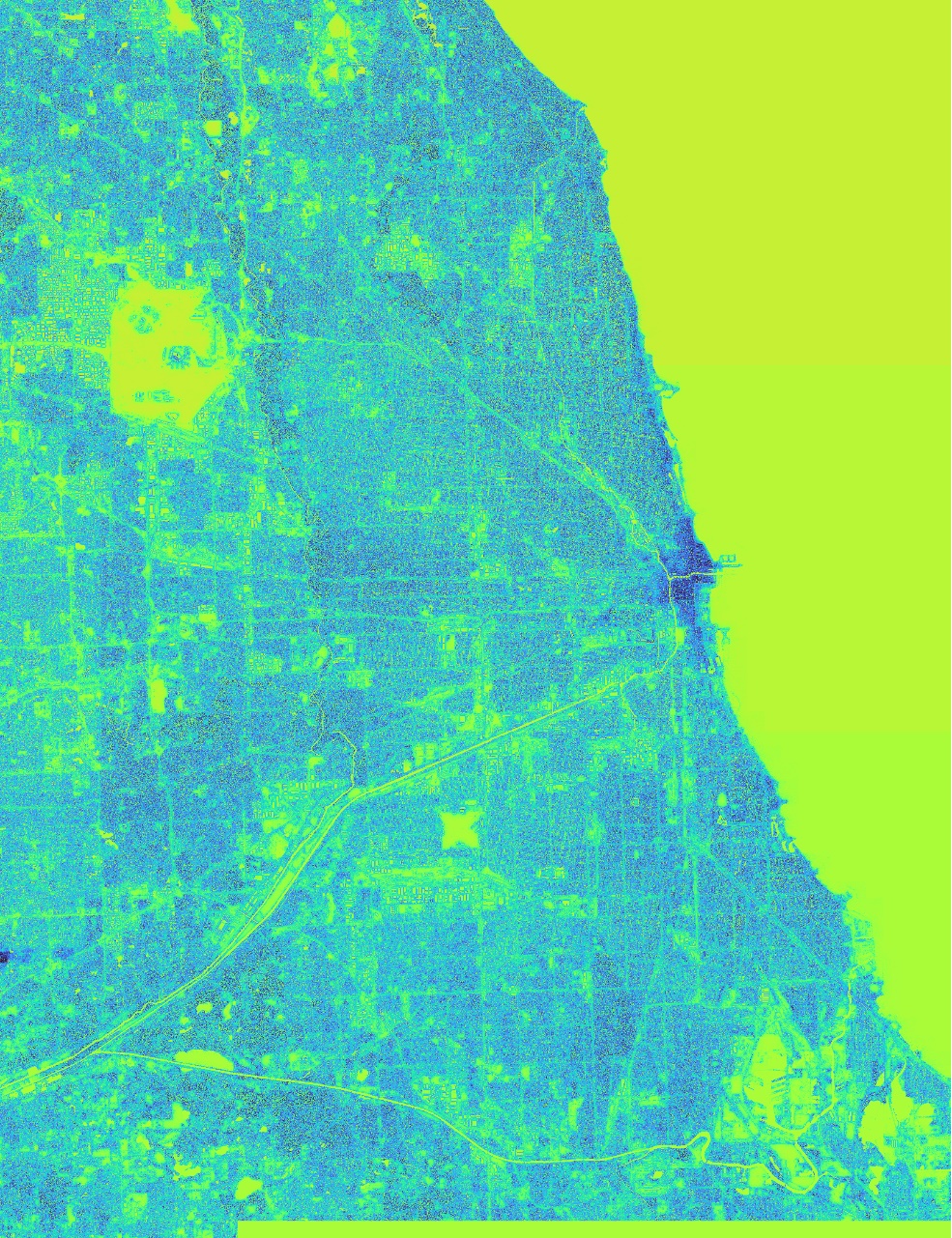


#D-计算SVF  
def SVF(sight\_distance,division\_num,z\_list):  
 import math  
 '''  
 function - 计算天空视域因子（Sky View Factor，SVF）  
   
 Paras:  
 sight\_distance -视线距离（视距）  
 division\_num - 视线等分的数量  
 z\_list - 截面高程数组  
 '''  
 segment=sight\_distance/division\_num  
 distance\_list=[i\*segment for i in range(division\_num+1)]  
 #print(distance\_list)  
 distance\_list=distance\_list[1:]  
   
 sin\_value\_list=[]  
 for i in z\_list:  
 sin\_maximum=0  
 for j in range(len(distance\_list)):  
 sin\_temp=(i[j+1]-i[0])/math.sqrt(math.pow(distance\_list[j],2)+math.pow(i[j+1]-i[0],2))  
 if sin\_temp>sin\_maximum:  
 sin\_maximum=sin\_temp  
 else:pass  
 sin\_value\_list.append(sin\_maximum)   
 SVF\_value=1-sum(sin\_value\_list)/len(z\_list)  
 print("SVF\_value=%s"%SVF\_value)  
   
 return SVF\_value  
  
SVF\_value=SVF(sight\_distance,division\_num,z\_list)

SVF\_value=0.7363503424243715

### 2.8.2.2 基于DSM计算SVF

上述SVF计算过程中使用了for循环，这将会大幅度增加计算时间，例如对于配置为：16G内存，可用约13G；Intel Core i7-8650U CPU @1.90GHz; 含大容量外置硬盘用于数据存储的条件下，（392，380）即148,960个值计算时长约为1min；（4428，4460）即19,748,880个值时为2hs。当计算数据量（栅格单元数）为3,119,035,612时(cell size=3×3)，for循环很难实现计算，因此必须转换为numpy数组形式逐次批量计算，避免逐个循环计算。其中核心的SVF计算方法（公式）并没有改变。最终区域DSM计算结果如图：

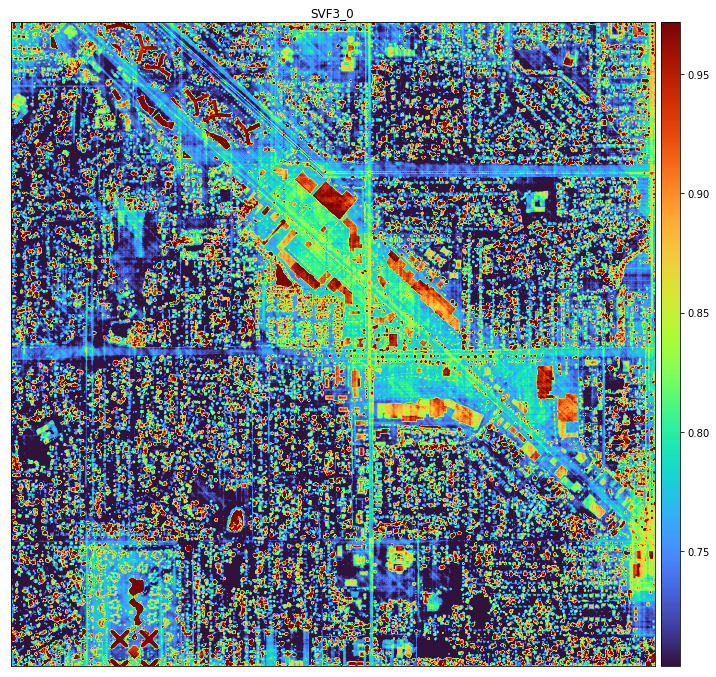


在计算过程中，使用rasterio库Window方法切分栅格为多个子栅格逐个读取计算，减少单个栅格的SVF计算量，并逐一保存。当计算完所有子栅格后，将其合并为单独一个或者2-3个栅格（当一个栅格存储文件过大时）。使用切分为子栅格计算SVF，存在一个问题，子栅格边缘位置的单元因为缺失邻近栅格造成计算不准确。这时可以衡量计算机的硬件条件，尽量让子栅格尽量大，减少误差的位置数量。或配置Window时，叠错处理，计算完后移除边缘。

class SVF\_DSM:  
 '''  
 class - 由DSM栅格计算SVF（适用于高分辨率大尺度栅格运算）  
 '''  
 def \_\_init\_\_(self,dsm\_fp,save\_root,sight\_distance,sight\_line\_num,division\_num):  
 self.dsm\_fp=dsm\_fp  
 self.save\_root=save\_root  
 self.sight\_distance=sight\_distance  
 self.sight\_line\_num=sight\_line\_num  
 self.division\_num=division\_num  
   
 def raster\_properties(self):  
 import rasterio as rio  
 '''  
 function - 读取栅格，查看属性值，返回需要的属性(栅格总宽高)  
 '''  
 raster=rio.open(self.dsm\_fp)  
 print("type:",type(raster))  
 print("transform:",raster.transform)  
 print("[width,height]:", raster.width, raster.height)  
 print("number of bands:",raster.count)  
 print("bounds:",raster.bounds)  
 print("driver:", raster.driver)  
 print("no data values:",raster.nodatavals)   
 print("\_"\*50)   
 return raster.width, raster.height  
   
 def divide\_chunks(self,l,n):  
 '''  
 function - 递归分组列表数据  
 '''   
 for i in range(0, len(l), n): # looping till length l  
 yield l[i:i + n]   
   
 def rasterio\_windows(self,total\_width,total\_height,sub\_width,sub\_height):  
 from rasterio.windows import Window  
 '''  
 function - 建立用于rasterio库分批读取一个较大raster数据的windows列表（尤其要处理较大单独的raster数据时，避免内存溢出）  
   
 Paras:  
 total\_width - 栅格总宽  
 total\_height - 栅格总高  
 sub\_width - 切分的子栅格宽  
 sub\_height - 切分的子栅格高  
 '''  
 w\_n=list(self.divide\_chunks(list(range(total\_width)), sub\_width))  
 h\_n=list(self.divide\_chunks(list(range(total\_height)), sub\_height))  
 wins=[Window(w[0],h[0],len(w),len(h)) for h in h\_n for w in w\_n]  
 print("raster windows amount:",len(wins))  
 return wins   
   
 def array\_coordi(self, raster\_array):  
 import numpy as np  
 '''  
 function - 计算栅格单元（数组）的相对坐标  
 '''  
 relative\_cell\_coords=np.indices(raster\_array.shape)  
 relative\_cell\_coords2D=np.stack(relative\_cell\_coords,axis=2).reshape(-1,2)  
 del relative\_cell\_coords  
 #print(relative\_cell\_coords2D)   
 return relative\_cell\_coords2D   
   
 def circle\_division(self,observation\_spot):  
 import numpy as np  
 '''  
 function - 给定观测位置点，视距和视线数量，等分圆，返回等分坐标点数组  
 '''  
 angle\_s=360/self.sight\_line\_num  
 angle\_array=np.array([i\*angle\_s for i in range(self.sight\_line\_num)],dtype=np.float32)  
 opposite=np.sin(np.radians(angle\_array),dtype=np.float16)\*self.sight\_distance  
 opposite=opposite.astype(np.float32)  
 yCoordi=np.add(opposite,observation\_spot[:,1].reshape(-1,1),dtype=np.float32)  
 del opposite  
  
 adjacent=np.cos(np.radians(angle\_array),dtype=np.float16)\*self.sight\_distance  
 xCoordi=np.add(adjacent,observation\_spot[:,0].reshape(-1,1),dtype=np.float32)  
 del adjacent,angle\_array  
  
 coordi\_array=np.stack((xCoordi,yCoordi),axis=-1)  
 return coordi\_array   
   
 def line\_profile(self,z\_value,observation\_spot,end\_point):   
 import numpy as np  
 import scipy.ndimage   
 '''  
 function - 获取与视线相交单元栅格（数组对应位置）的栅格值（数组值），即数组延直线截面提取单元值  
   
 Paras:  
 z\_value - DSM栅格（含高程信息）  
 observation\_spot - 观察点数组  
 end\_point - 视线末尾点数组  
 '''   
 num=self.division\_num+1  
  
 x0=observation\_spot[:,0].reshape(-1,1)  
 x1=end\_point[:,:,0]  
 x=np.linspace(x0, x1, num,dtype=int) #可以不用修改数组类型。出于内存优化考虑,会加快后续np.stack计算速度  
 del x0,x1  
  
 y0=observation\_spot[:,1].reshape(-1,1)   
 y1=end\_point[:,:,1]  
 y=np.linspace(y0, y1, num,dtype=int)  
 del y0,y1  
  
 xStack=np.stack(x,axis=-1)   
 yStack=np.stack(y,axis=-1)  
 del x,y  
  
 zi=scipy.ndimage.map\_coordinates(z\_value,[xStack,yStack],cval=0,mode="nearest",order=0) #根据数组索引值，提取实际值  
 del xStack,yStack  
  
 return zi   
   
 def SVF(self,z\_list):  
 import numpy as np  
 '''  
 function - 计算天空视域因子（Sky View Factor，SVF）  
 '''  
 segment=self.sight\_distance/self.sight\_line\_num  
 distance\_list=np.array([i\*segment for i in range(self.sight\_line\_num+1)],dtype=np.float32)  
 distance\_list=distance\_list[1:]  
 distance\_list=np.expand\_dims(distance\_list,axis=1)  
 z\_list\_sub=z\_list[:,:,1:]  
 z\_list\_origin=z\_list[:,:,0]  
   
 sin\_values=np.true\_divide(np.subtract(z\_list\_sub,np.expand\_dims(z\_list\_origin, axis=2)),np.sqrt(np.add(np.power(distance\_list,2),np.power(np.subtract(z\_list\_sub,np.expand\_dims(z\_list\_origin, axis=2)),2))))  
 sin\_max\_value=np.amax(sin\_values,axis=2)  
 del sin\_values  
 SVF\_value=1-np.true\_divide(np.sum(sin\_max\_value, axis=1),sin\_max\_value.shape[-1])  
  
 return SVF\_value.astype(np.float32)   
   
 def svf\_wins(self, wins\_list):  
 from tqdm import tqdm  
 import rasterio as rio  
 import datetime,gc,os  
 from rasterio.windows import Window  
 import warnings   
 #suppress warnings  
 warnings.filterwarnings('ignore')  
 '''  
 function - 计算SVF的主程序，并保存SVF子栅格文件  
 '''  
 i=0  
 for win in tqdm(wins\_list):   
 with rio.open(self.dsm\_fp,"r+") as src:  
 src.nodata=-1  
 w=src.read(1, window=win)   
 profile=src.profile  
 win\_transform=src.window\_transform(win)   
   
 '''计算部分'''   
 #print(w.shape)  
 relative\_cell\_coords2D=self.array\_coordi(w)  
   
 #B-等分视域，获取点坐标  
 a\_T=datetime.datetime.now()  
 coordi\_array=self.circle\_division(relative\_cell\_coords2D)   
 b\_T= datetime.datetime.now()  
 print("circle\_division-time span:{}".format( b\_T-a\_T))  
 gc.collect()  
   
 #C-根据视线提取栅格（数组）对应位置的高程（数组对应位置值），即批量提取截面高程数据  
 c\_T=datetime.datetime.now()  
 zi=self.line\_profile(w,relative\_cell\_coords2D,coordi\_array)  
 d\_T=datetime.datetime.now()  
 print("lineProfile-time span:{}".format(d\_T-c\_T))  
 gc.collect()  
 del coordi\_array   
   
 #D-计算SVF  
 e\_T=datetime.datetime.now()  
 SVF\_value=self.SVF(zi)  
 f\_T=datetime.datetime.now()  
 print("SVF-time span:{}".format(d\_T-c\_T))  
 gc.collect()  
 del zi  
   
 profile.update(  
 width=win.width,   
 height=win.height,  
 count=1,  
 transform=win\_transform,  
 compress='lzw',  
 dtype=rio.float32  
 )   
 with rio.open(os.path.join(self.save\_root,"SVF3\_%d.tif"%i), 'w', \*\*profile) as dst:  
 dst.write(SVF\_value.reshape(w.shape), window=Window(0,0,win.width,win.height), indexes=1)   
 del SVF\_value  
 g\_T=datetime.datetime.now()  
 print("total-time span:{}".format(g\_T-a\_T))  
 i+=1   
   
 break   
  
dsm\_fp=r"G:\data\DSM\_pixel\_3.tif"  
save\_root=r"G:\data\data\_processed\SVF\SFV3\_A"   
#A-SVF配置基本参数  
raster\_resolution=3 #计算栅格的分辨率   
sight\_distance=100\*raster\_resolution #扫描半径  
sight\_line\_num=8 #扫描截面数量36  
division\_num=30 #每条扫描线的等分数量 50  
  
SVF=SVF\_DSM(dsm\_fp,save\_root,sight\_distance,sight\_line\_num,division\_num)   
total\_width,total\_height=SVF.raster\_properties()  
print("total\_width={},total\_height={}".format(total\_width,total\_height))  
  
sub\_width,sub\_height=3000,3000 #3000,3000  
wins\_list=SVF.rasterio\_windows(total\_width,total\_height,sub\_width,sub\_height)  
print("windows example:{}".format(wins\_list[:5]))  
  
SVF.svf\_wins(wins\_list)

type: <class 'rasterio.io.DatasetReader'>  
transform: | 3.00, 0.00, 1075000.00|  
| 0.00,-3.00, 1987501.00|  
| 0.00, 0.00, 1.00|  
[width,height]: 46667 60834  
number of bands: 1  
bounds: BoundingBox(left=1075000.0, bottom=1805000.0, right=1215001.0, top=1987501.0)  
driver: GTiff  
no data values: (-1.0,)  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
total\_width=46667,total\_height=60834  
raster windows amount: 336  
windows example:[Window(col\_off=0, row\_off=0, width=3000, height=3000), Window(col\_off=3000, row\_off=0, width=3000, height=3000), Window(col\_off=6000, row\_off=0, width=3000, height=3000), Window(col\_off=9000, row\_off=0, width=3000, height=3000), Window(col\_off=12000, row\_off=0, width=3000, height=3000)]  
  
  
 0%| | 0/336 [00:00<?, ?it/s]  
  
circle\_division-time span:0:00:00.658197  
lineProfile-time span:0:03:49.709210  
SVF-time span:0:03:49.709210  
  
  
 0%| | 0/336 [04:54<?, ?it/s]  
  
total-time span:0:04:52.874828

import util\_A  
SVF3\_0\_fp=r"G:\data\data\_processed\SVF\SFV3\_A\SVF3\_0.tif"  
util\_A.raster\_show(SVF3\_0\_fp,title='SVF3\_0')



### 2.8.2.3 内存管理

常规电脑的硬件配置通常有个限度。因此在处理大数据时，可能因为数据量、计算量造成内存溢出。例如np.arange(6000000000,dtype=np.float64)数组，预计占用45.7GB，如果内存小于该值，将溢出。6,000,000,000数据量，相当于77459.6m × 77459.6m的城市区域。如果在这个过程中，存在其它计算，则将大幅度增加所用内存，因此如何管理内存与释放内存，在大数据处理过程中显得尤为重要。

* 对较大数据进行数据分析的几点建议：

1. 避免使用for循环，尽量使用array(numpy)直接数组间计算或者使用DataFrame(pandas)，可以大幅度增加计算速度，但会占用较大内存，需使用内存管理/减压工具处理；
2. 数据分批处理，并保存于硬盘中。逐一处理完后，读取所有文件进行后续处理。需平衡分批与一次性数组计算量，每次数组大计算速度快，但占用内存多；如果增加批次，则会降低单次数组量，但会增加计算时间；
3. 如果不必要，不需保存中间过渡的大数组。通常应用后，del释放内存，仅保留和存储必要的计算结果或中间结果。大数组尽量使用h5py保存于硬盘中。并尽量避免使用np.save工具，该工具保存的数据占据磁盘空间较大；
4. 大数据处理过程中，尽量避免使用matplotlib查看数据。只有必要分析时，再单独处理；
5. 为了减缓内存，数据及过程数据存储于大的硬盘中。根据自身数据大小，可以准备高容量的外置硬盘使用。另固态硬盘具有更快的读写速度；
6. 使用虚拟内存，可以有效缓解内存压力；
7. 有必要使用GPU计算，尤其训练深度学习模型；
8. 优化算法。

#### 1）[psutil](https://psutil.readthedocs.io/en/latest/)

psutil用于在python中检索有关正在运行的进程和系统利用率(CPU、内存、磁盘、网络、传感器)的信息。主要用于系统监控、分析、限制进程资源和管理运行的进程。（更详细的内容需查看[psutil文档](https://psutil.readthedocs.io/en/latest/)）

import psutil  
print("cpu\_count:{}".format(psutil.cpu\_count(logical=True))) #当配置logical=True时，返回logical CPUs的数量。同os.cpu\_count()  
  
mem=psutil.virtual\_memory()  
print("virtual\_memory:{}".format(mem)) #以元组的形式返回系统内存使用的统计信息  
THRESHOLD=100 \* 1024 \* 1024 # 100MB  
if mem.available <= THRESHOLD:  
 print("warning")  
   
print("disk\_partitions:{}".format(psutil.disk\_partitions())) #返回挂载的磁盘分区（mounted partitions）有关信息  
print("disk\_usage:{}".format(psutil.disk\_usage('/'))) #返回包含给定路径的分区磁盘使用统计信息，包含以字节表示的总空间、已用空间和空闲空间，及使用百分比

cpu\_count:16  
virtual\_memory:svmem(total=51410481152, available=38877888512, percent=24.4, used=12532592640, free=38877888512)  
disk\_partitions:[sdiskpart(device='C:\\', mountpoint='C:\\', fstype='NTFS', opts='rw,fixed'), sdiskpart(device='E:\\', mountpoint='E:\\', fstype='exFAT', opts='rw,fixed'), sdiskpart(device='F:\\', mountpoint='F:\\', fstype='NTFS', opts='rw,fixed'), sdiskpart(device='G:\\', mountpoint='G:\\', fstype='exFAT', opts='rw,fixed')]  
disk\_usage:sdiskusage(total=511101923328, used=473427255296, free=37674668032, percent=92.6)

#### 2）[h5py](https://www.h5py.org/)

HDF5(python,h5py) 是HDF5二进制数据格式的python接口。HDF5可以存储大量数值数据（numerical data），并可以轻松地用NumPy操作这些数据。例如，可以将存储在磁盘上的tb级数据集切片；成千上万的数据集可以存储在一个文件中，根据需要进行分类和标记。 （更详细的内容需查看[h5py文档](https://docs.h5py.org/en/stable/)）

import h5py,os  
import numpy as np  
  
# 建立hdf5文件路径  
data\_save\_root=r"G:\data\data\_processed\h5py"  
hdf5\_fp=os.path.join(data\_save\_root,"h5py\_experi.hdf5")  
if os.path.exists(hdf5\_fp):  
 os.remove(hdf5\_fp)  
else:  
 print("Can note delete the file as it does not exists. to built a new one!")  
  
#生成随机的实验数据  
data\_a=np.random.rand(10000,10000)  
data\_b=np.random.rand(10000,10000)  
data\_c=np.random.rand(10000,10000)   
data\_d=np.random.rand(10000,10000)  
data\_e=np.random.randint(100,size=(100,100))  
  
#用f=h5py.File()和f.close()，打开写入数据文件，并关闭文件  
f=h5py.File(hdf5\_fp, "w") #create a file by setting the mode to w when the File object is initialized. Some other modes are a (for read/write/create access), and r+ (for read/write access).   
dset\_a=f.create\_dataset("dataset\_a",data=data\_a)  
dset\_b=f.create\_dataset("dataset\_b",data=data\_b)  
print("dset\_a name={}; dset\_b name={}".format(dset\_a.name,dset\_b.name))  
print("f name={}".format(f.name)) #可以看作默认组（group）  
f.close()  
  
#可以用with h5py.File() as f:的方法，读写数据  
with h5py.File(hdf5\_fp, "a") as f:  
 #建立子组/群（subgroup），子组中可以写入若干数据集(dataset)  
 grp=f.create\_group("subgroup")  
 print("grp(group) name={}".format(grp.name))   
 dset2=grp.create\_dataset("dataset\_c", (10000,10000), dtype='f',data=data\_c)   
 print("dset2 name={}".format(dset2.name))  
 grp.create\_dataset("dataset\_d", data=data\_d)   
 print("subgroup keys={}".format(list(f["subgroup"].keys())))   
  
with h5py.File(hdf5\_fp, "a") as f:   
 #直接建立子组和子组下的数据集  
 dset3=f.create\_dataset('subgroup2/dataset\_three', (100,100), dtype='i',data=data\_e)  
 dset3.attrs["attri\_a"]="attri\_A" #可以配置属性字段  
 dset3.attrs["attri\_b"]="attri\_B"  
 print("dset3 name={}".format(dset3.name))   
 print("dset3.attrs[\"attri\_a\"]={}".format(dset3.attrs["attri\_a"]))  
 print("dset3.attrs list={}".format(list(dset3.attrs)))

dset\_a name=/dataset\_a; dset\_b name=/dataset\_b  
f name=/  
grp(group) name=/subgroup  
dset2 name=/subgroup/dataset\_c  
subgroup keys=['dataset\_c', 'dataset\_d']  
dset3 name=/subgroup2/dataset\_three  
dset3.attrs["attri\_a"]=attri\_A  
dset3.attrs list=['attri\_a', 'attri\_b']

def visit\_func(name, node) :  
 '''  
 function - 打印HDF5文件数据集和子组信息  
 '''  
 print ('Full object pathname is:', node.name)  
 if isinstance(node, h5py.Group) :  
 print ('Object:', name, 'is a Group\n')  
 elif isinstance(node, h5py.Dataset) :  
 print ('Object:', name, 'is a Dataset\n')  
 else :  
 print ('Object:', name, 'is an unknown type\n')   
   
def get\_dataset\_keys(f):  
 '''  
 function - 返回HDF5文件数据集路径（包括子组内数据集）  
 '''  
 keys=[]  
 f.visit(lambda key : keys.append(key) if isinstance(f[key], h5py.Dataset) else None)  
 return keys   
  
#查看HDF5文件，并读取数据集  
with h5py.File(hdf5\_fp, "r+") as f:   
 print("f\_keys={}".format(list(f.keys())))  
 print("\_"\*50)  
 datasets\_description=f.visititems(visit\_func)  
   
 dataset\_keys=get\_dataset\_keys(f)  
 print("dataset keys={}".format(dataset\_keys))  
   
 dataset\_three=f.get(dataset\_keys[-1])[:]  
print("\_"\*50)   
print("'subgroup2/dataset\_three' array:\n{}".format(dataset\_three))

f\_keys=['dataset\_a', 'dataset\_b', 'subgroup', 'subgroup2']  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Full object pathname is: /dataset\_a  
Object: dataset\_a is a Dataset  
  
Full object pathname is: /dataset\_b  
Object: dataset\_b is a Dataset  
  
Full object pathname is: /subgroup  
Object: subgroup is a Group  
  
Full object pathname is: /subgroup/dataset\_c  
Object: subgroup/dataset\_c is a Dataset  
  
Full object pathname is: /subgroup/dataset\_d  
Object: subgroup/dataset\_d is a Dataset  
  
Full object pathname is: /subgroup2  
Object: subgroup2 is a Group  
  
Full object pathname is: /subgroup2/dataset\_three  
Object: subgroup2/dataset\_three is a Dataset  
  
dataset keys=['dataset\_a', 'dataset\_b', 'subgroup/dataset\_c', 'subgroup/dataset\_d', 'subgroup2/dataset\_three']  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
'subgroup2/dataset\_three' array:  
[[98 15 44 ... 51 9 14]  
 [89 98 29 ... 29 91 57]  
 [45 54 54 ... 92 52 45]  
 ...  
 [12 92 2 ... 45 86 61]  
 [ 9 36 42 ... 57 90 75]  
 [33 62 23 ... 31 14 98]]

#### 3）[PyTables](https://pypi.org/project/tables/)

PyTables是一个用于管理分层数据集的库，旨在高效、轻松地处理极其大量的数据。PyTables是在HDF5库的基础上构建的，使用Python语言和NumPy库，可以交互式浏览、处理和搜索大量数据。该库优化内存和磁盘资源，使得数据占用的空间得以优化。（更详细的内容需查看[PyTables文档](https://www.pytables.org/usersguide/tutorials.html)）

下述代码迁移于[PyTables的GitHub仓库](https://github.com/PyTables/PyTables/blob/master/examples/tutorial1-1.py)。对于DataFrame(pandas)可以使用DataFrame.to\_hdf()和pandas.read\_hdf()来读写HDF5文件。pandas是使用PyTables完成读写HDF5文件。

import tables as tb  
import numpy as np  
  
# Define a user record to characterize some kind of particles  
class Particle(tb.IsDescription):  
 name = tb.StringCol(16) # 16-character String  
 idnumber = tb.Int64Col() # Signed 64-bit integer  
 ADCcount = tb.UInt16Col() # Unsigned short integer  
 TDCcount = tb.UInt8Col() # unsigned byte  
 grid\_i = tb.Int32Col() # integer  
 grid\_j = tb.Int32Col() # integer  
 pressure = tb.Float32Col() # float (single-precision)  
 energy = tb.Float64Col() # double (double-precision)  
  
print('-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*- file creation -\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-')   
# The name of our HDF5 filename  
filename = "G:\data\data\_processed\h5py\pytables\_experi.h5"  
# Open a file in "w"rite mode  
h5file = tb.open\_file(filename, mode="w", title="pytables\_experi")  
print("Creating file:", filename)  
  
print()  
print('-\*\*-\*\*-\*\*-\*\*-\*\*- group and table creation -\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-')  
# Create a new group under "/" (root)  
group = h5file.create\_group("/", 'detector', 'Detector information')  
print("Group '/detector' created")  
  
# Create one table on it  
table = h5file.create\_table(group, 'readout', Particle, "Readout example")  
print("Table '/detector/readout' created")  
  
# Print the file  
print(h5file)  
print(repr(h5file))  
  
# Get a shortcut to the record object in table  
particle = table.row  
  
# Fill the table with 10 particles  
for i in range(10):  
 particle['name'] = 'Particle: %6d' % (i)  
 particle['TDCcount'] = i % 256  
 particle['ADCcount'] = (i \* 256) % (1 << 16)  
 particle['grid\_i'] = i  
 particle['grid\_j'] = 10 - i  
 particle['pressure'] = float(i \* i)  
 particle['energy'] = float(particle['pressure'] \*\* 4)  
 particle['idnumber'] = i \* (2 \*\* 34)  
 particle.append()  
  
# Flush the buffers for table  
table.flush()  
  
print()  
print('-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*- table data reading & selection -\*\*-\*\*-\*\*-\*\*-\*\*-')  
# Read actual data from table. We are interested in collecting pressure values on entries where TDCcount field is greater than 3 and pressure less than 50  
xs = [x for x in table.iterrows() if x['TDCcount'] > 3 and 20 <= x['pressure'] < 50]  
pressure = [x['pressure'] for x in xs ]  
print("Last record read:")  
print(repr(xs[-1]))  
print("Field pressure elements satisfying the cuts:")  
print(repr(pressure))  
  
# Read also the names with the same cuts  
names = [x['name'] for x in table.where("""(TDCcount > 3) & (20 <= pressure) & (pressure < 50)""")]  
print("Field names elements satisfying the cuts:")  
print(repr(names))  
  
print()  
print('-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*- array object creation -\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-')  
print("Creating a new group called '/columns' to hold new arrays")  
gcolumns = h5file.create\_group(h5file.root, "columns", "Pressure and Name")  
  
print("Creating an array called 'pressure' under '/columns' group")  
h5file.create\_array(gcolumns, 'pressure', np.array(pressure),"Pressure column selection")  
print(repr(h5file.root.columns.pressure))  
  
print("Creating another array called 'name' under '/columns' group")  
h5file.create\_array(gcolumns, 'name', names, "Name column selection")  
print(repr(h5file.root.columns.name))  
  
print("HDF5 file:")  
print(h5file)  
  
# Close the file  
h5file.close()  
print("File '" + filename + "' created")

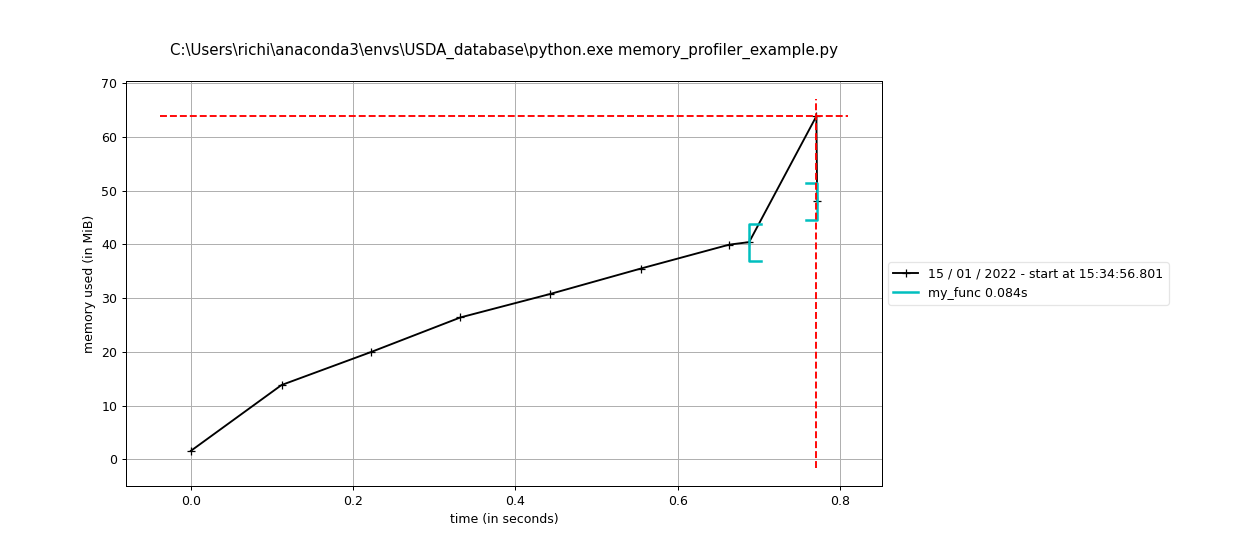
-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*- file creation -\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-  
Creating file: G:\data\data\_processed\h5py\pytables\_experi.h5  
  
-\*\*-\*\*-\*\*-\*\*-\*\*- group and table creation -\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-  
Group '/detector' created  
Table '/detector/readout' created  
G:\data\data\_processed\h5py\pytables\_experi.h5 (File) 'pytables\_experi'  
Last modif.: 'Sat Jan 15 14:34:30 2022'  
Object Tree:   
/ (RootGroup) 'pytables\_experi'  
/detector (Group) 'Detector information'  
/detector/readout (Table(0,)) 'Readout example'  
  
File(filename=G:\data\data\_processed\h5py\pytables\_experi.h5, title='pytables\_experi', mode='w', root\_uep='/', filters=Filters(complevel=0, shuffle=False, bitshuffle=False, fletcher32=False, least\_significant\_digit=None))  
/ (RootGroup) 'pytables\_experi'  
/detector (Group) 'Detector information'  
/detector/readout (Table(0,)) 'Readout example'  
 description := {  
 "ADCcount": UInt16Col(shape=(), dflt=0, pos=0),  
 "TDCcount": UInt8Col(shape=(), dflt=0, pos=1),  
 "energy": Float64Col(shape=(), dflt=0.0, pos=2),  
 "grid\_i": Int32Col(shape=(), dflt=0, pos=3),  
 "grid\_j": Int32Col(shape=(), dflt=0, pos=4),  
 "idnumber": Int64Col(shape=(), dflt=0, pos=5),  
 "name": StringCol(itemsize=16, shape=(), dflt=b'', pos=6),  
 "pressure": Float32Col(shape=(), dflt=0.0, pos=7)}  
 byteorder := 'little'  
 chunkshape := (1394,)  
  
  
-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*- table data reading & selection -\*\*-\*\*-\*\*-\*\*-\*\*-  
Last record read:  
/detector/readout.row (Row), pointing to row #9  
Field pressure elements satisfying the cuts:  
[81.0, 81.0, 81.0]  
Field names elements satisfying the cuts:  
[b'Particle: 5', b'Particle: 6', b'Particle: 7']  
  
-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*- array object creation -\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-\*\*-  
Creating a new group called '/columns' to hold new arrays  
Creating an array called 'pressure' under '/columns' group  
/columns/pressure (Array(3,)) 'Pressure column selection'  
 atom := Float64Atom(shape=(), dflt=0.0)  
 maindim := 0  
 flavor := 'numpy'  
 byteorder := 'little'  
 chunkshape := None  
Creating another array called 'name' under '/columns' group  
/columns/name (Array(3,)) 'Name column selection'  
 atom := StringAtom(itemsize=16, shape=(), dflt=b'')  
 maindim := 0  
 flavor := 'python'  
 byteorder := 'irrelevant'  
 chunkshape := None  
HDF5 file:  
G:\data\data\_processed\h5py\pytables\_experi.h5 (File) 'pytables\_experi'  
Last modif.: 'Sat Jan 15 14:34:30 2022'  
Object Tree:   
/ (RootGroup) 'pytables\_experi'  
/columns (Group) 'Pressure and Name'  
/columns/name (Array(3,)) 'Name column selection'  
/columns/pressure (Array(3,)) 'Pressure column selection'  
/detector (Group) 'Detector information'  
/detector/readout (Table(10,)) 'Readout example'  
  
File 'G:\data\data\_processed\h5py\pytables\_experi.h5' created

#### 4）[memory\_profiler](https://pypi.org/project/memory-profiler/)

memory\_profiler 用于监控进程的内存消耗，及逐行分析python程序的内存消耗。（更详细的内容需查看[memory\_profiler文档](https://pypi.org/project/memory-profiler/)）

@profile  
def my\_func():  
 a = [1] \* (10 \*\* 6)  
 b = [2] \* (2 \* 10 \*\* 7)  
 del b  
 return a  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 my\_func()

将上述官网提供的示例程序，单独保存为.py文件后，在终端执行mprof run <executable>，以及mprof plot，可以获取如下图表：



执行python -m memory\_profiler <executable>，可以逐行观察计算所占内存量，用于观察代码优化结果。计算结果如下：

Line # Mem usage Increment Occurences Line Contents  
============================================================  
 1 40.832 MiB 40.832 MiB 1 @profile  
 2 def my\_func():  
 3 48.465 MiB 7.633 MiB 1 a = [1] \* (10 \*\* 6)  
 4 201.055 MiB 152.590 MiB 1 b = [2] \* (2 \* 10 \*\* 7)  
 5 48.465 MiB -152.590 MiB 1 del b  
 6 48.465 MiB 0.000 MiB 1 return a

注释（Notes）：

① psutil，（<https://psutil.readthedocs.io/en/latest>）。

② HDF5 for Python，h5py 包是 HDF5 二进制数据格式的 Python 接口，（[https://www.h5py.org](https://www.h5py.org/)）。

③ PyTables，（<https://pypi.org/project/tables>）。

④ memory\_profiler，（<https://pypi.org/project/memory-profiler>）。

参考文献：

[1]段欣,胡德勇,曹诗颂,于琛,张亚妮.城区复杂下垫面天空视域因子参数化方法——以北京鸟巢周边地区为例[J].国土资源遥感,2019,31(03):29-35.

[2]Böhner, J., & Antonić, O. (2009). Chapter 8 Land-Surface Parameters Specific to Topo-Climatology. Geomorphometry - Concepts, Software, Applications, 195–226. doi:10.1016/s0166-2481(08)00008-1

[3] Häntzschel, J., Goldberg, V., & Bernhofer, C. (2005). GIS-based regionalisation of radiation, temperature and coupling measures in complex terrain for low mountain ranges. Meteorological Applications, 12(1), 33–42. doi:10.1017/s1350482705001489

[4] PyTables, tutorial1-1.py，<https://github.com/PyTables/PyTables/blob/master/examples/tutorial1-1.py>.