The tip of the iceberg

Ådne Hult Karlson, Subhashree Bal and Edward Peter Dontoh

Date: **September 11, 2022**

Abstract

The use of numerical models and simulations to study complex systems is becoming increasingly popular in a wide range of research fields. However, this method has some limitations, the most notable of which is numerical error.

In this project, we have worked on analyzing the round-off errors and truncation errors. Also, the python libraries like numpy and others have been explored to work on datasets. The findings highlighted the numerical errors associated with computing in Python. It also demonstrated how easily python libraries handled large data sets when computing with python. Finally, we used external dataset to calulate the sea level rise if the ice in antarctica should melt. we identified that SLR will be approximatly 74 meters.

Introduction

There are many different ways to structure an algorithm in Python, either by utilizing built-in functions or libraries. This project seeks to familiarize us with both alternatives, applications and methods for dealing with floating-point values. Four separate exercises are used to achieve this. In the first and second exercises, we used the built-in Python functions and numpy library to perform computation. We look at the accuracy as well as the accompanying numerical errors. We also noted how quickly and easily the numpy library executed and handled multiple numbers at once. The third and fourth challenges heavily emphasized developing functions, manipulating large data size, and using Python libraries to visualize the data.

Exercise 1: Finite-precision arithmetic

part 1

Information about machine precision and internal representation of floating point

In [ ]:
```
import sys
"""sys is a built in library of python which gives information about constants,
functions and methods of Python's interpreter. sys.float_info provides information a
These constants are shown as output in the cell above."""

print("max:         ", "The maximum possible positive number
print("max_exp:     ", "The maximum integer E in 2**(E-1) to present a float in the
print("max_10_exp:  ", "The maximum integer E in 10**E to present a flot i the range
print("min:         ", "The minimum possible positive number
print("min_exp:     ", "The minimum integer E in 2**(E-1) to present a float in the
print("min_10_exp:  ", "The minimum integer E in 10**E to present a float i the rang
print("dig:         ", "The maximum number of digets that can be trusted when presen
```

```
print("mant_dig:    ", "The number of exponent digets in the significant of a float
print("epsilon:     ", "The difference between 1.0 and the last value greater than 1
print("randix:      ", "The exponent representation
print("round:       ", "The rounding constant
```

```
max:          The maximum possible positive number
: 1.7976931348623157e+308
max_exp:      The maximum integer E in 2**(E-1) to present a float in the range
: 1024
max_10_exp:   The maximum integer E in 10**E to present a flot i the range
: 308
min:          The minimum possible positive number
: 2.2250738585072014e-308
min_exp:      The minimum integer E in 2**(E-1) to present a float in the range
: -1021
min_10_exp:   The minimum integer E in 10**E to present a float i the range
: -307
dig:          The maximum number of digets that can be trusted when presenting a flo
at         : 15
mant_dig:     The number of exponent digets in the significant of a float
: 53
epsilon:      The difference between 1.0 and the last value greater than 1.0 that is
presented : 2.220446049250313e-16
randix:       The exponent representation
: 2
round:        The rounding constant
: 1
```

part 2

We continued by determing how we can manually compute some of these floating points including the max, min and epsilon values

In [ ]:
```
max = (2-2**-52)*(2**1023)
min = 2**(-1022)
epsilon = 2**-52
print("max: ", max)
print("min: ", min)
print("epsilon: ", epsilon)
```

```
max:  1.7976931348623157e+308
min:  2.2250738585072014e-308
epsilon:  2.220446049250313e-16
```

Part 3

In this part, we performed a simple computation to determine how python handles floating numbers

In [ ]:
```
0.1+0.2
```

Out[ ]:
```
0.30000000000000004
```

As shown above the $0.1 + 0.2 \neq 0.3$. This is because numbers are represented as a finite number of bits. (typically 64-bit). Where the first bit position is the sign, the next 11 bits is the exponent and the last 52 bits is the fraction. This causes some rounding error for floating points as representing some decimals requires an infinite number of digits.

part 4

Alternative ways to do floating-point number comparison

In [ ]:
```
a = 0.1
b = 0.2
c = 0.1 + 0.2
if c == 0.3:
    print("==, operator workes and gives true")
else:
    print("==, operator does not work and gives false")

if (c - 0.3) < 0.000001:
    print("<, operator works and gives true")
else:

    print("<, does not work and gives false")
```

```
==, operator does not work and gives false
<, operator works and gives true
```

For the example above where $0.1 + 0.2 \neq 0.3$ it is clear that for some cases using the $==$ opperator will give some errors. This is why we should not use this operator when dealing with floating points. As shown in the code above we can see that == operator does not work. An alternative method would be the close enoght method, where we define a tolerance. in this case 0.000001 which is vary small.

Exercise 2: Get up to speed with NumPy

Part 1

As mentioned in the introduction, this exercise forcuses on the use of the numpy library. We started by working with its exposnential function to see how it deals with floating numbers

In [ ]:
```
#### part 1 #### Vectorized function
import numpy as np
x = np.linspace(0, 1, 10)
np.exp(x)  # Apply f(t)=exp(t) to each element in the array x.
np.exp(-x)  # Apply the function f(t)=exp(-t) to each element of x.

print('x values',x)
print('numpy exponent of x values','',np.exp(x))
print('numpy exponent of -x values','',np.exp(-x))
```

```
x values [0.         0.11111111 0.22222222 0.33333333 0.44444444 0.55555556
 0.66666667 0.77777778 0.88888889 1.        ]
numpy exponent of x values  [1.         1.11751907 1.24884887 1.39561243 1.5596235
1.742909
 1.94773404 2.17662993 2.43242545 2.71828183]
numpy exponent of -x values  [1.         0.89483932 0.8007374  0.71653131 0.64118039
0.57375342
 0.51341712 0.45942582 0.41111229 0.36787944]
```

In [ ]:
```
import numpy as np
x = [0., 0.11111111, 0.22222222, 0.33333333, 0.44444444, 0.55555556,0.66666667, 0.77
np.exp(x)
#np.exp(-x)

print('ranger for native x','',x)
print('using numpy function on native (x)','',np.exp(x))
#print('using numpy Function on native (-x)',np.exp(-x))
```

```
ranger for native x  [0.0, 0.11111111, 0.22222222, 0.33333333, 0.44444444, 0.5555555
6, 0.66666667, 0.77777778, 0.88888889, 1.0]
using numpy function on native (x)  [1.         1.11751907 1.24884887 1.39561242 1.5
5962349 1.74290901
 1.94773405 2.17662994 2.43242546 2.71828183]
```

Using numpy to generate x values and then computing the positive and negative exponential values for x gives an output with no errors. However, when you create a native list values for x and then you use numpy to compute its exponential values, you get an error with the negative exponential values. This is because in the first case, numpy trated the x values as an array and multiplied the minus by the data in x, to convert the x values from positive to negative. After doing this, it then computed the exponential values of the negative numbers. On the other hand, In the second case, the native x values are treated as a list and python lists don't support unary operation minus, as we are using in the function. Hence we get an error

In [ ]:

```
my_list = []*10
my_list = [0., 0.11111111, 0.22222222, 0.33333333, 0.44444444, 0.55555556,0.66666667
output_list1 = [np.exp (i) for i in my_list]
output_list2 = [np.exp (-i) for i in my_list]
print("list np.exp(i)", output_list1)
print("list np.exp(-i)", output_list2)
```

```
list np.exp(i) [1.0, 1.1175190675001758, 1.2488488662264625, 1.395612420434048, 1.55
96234906751207, 1.74290900637972, 1.9477340475471228, 2.1766299365532036, 2.43242545
6989903, 2.718281828459045]
list np.exp(-i) [1.0, 0.8948393178086357, 0.8007374046962245, 0.7165313129622269, 0.
6411803912796452, 0.5737534181874177, 0.5134171173212017, 0.45942582301498036, 0.411
112290050396, 0.36787944117144233]
```

To evaluate a function on a native python list, one has to use the for loop. This allows the python fuction to iterate and deal with individual elements in the list.

Part 2

Explain what each line of code does

In [ ]:

```
#### part 2 ####
import numpy as np
print('zero elements',np.zeros(20)) # array with 20 elements equal to zero
print('Array with ones',np.ones(20)) # array with 20 elements equal to one
print('linspace over a specified interval', np.linspace(0, 10, 11)) # Return evenly
print('linspace over a specified interval with stop element excluded', np.linspace(0
vector1 = np.arange(5) + 1 # create an array of 5 elements starting from o to 4,  ad
print(vector1) #print the array saved as "vector1"
print('Elements in vector1',2*vector1) # multiplies each element of array by 2
```

```
zero elements [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Array with ones [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
linspace over a specified interval [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
linspace over a specified interval with stop element excluded [0.         0.90909091
1.81818182 2.72727273 3.63636364 4.54545455
 5.45454545 6.36363636 7.27272727 8.18181818 9.09090909]
[1 2 3 4 5]
Elements in vector1 [ 2  4  6  8 10]
```

Produce same using Python lists

In [ ]:

```
# Implementation of np.zeroes
list_of_zeroes = [0]*20
print(list_of_zeroes)
```

```
# Implementation of np.ones
list_of_ones = [1]*20
print(list_of_ones)

"""Implementation of np.linspace(0, 10, 11)
As the code above suggest that the resulting list will start from 0 and end at 10 (i
Such list would contain 11 elements with an increment of 1, which would become step
Also, we would try to generate a list with floating values as np.linspace would crea
"""
step1 = 1.
stepnew1=([0 + step1*interval for interval in range(11)])
stepnew11=[round(item,8) for item in stepnew1]
print(stepnew11)

"""Implementation of np.linspace(0, 10, 11, endpoint=False)
As the code above suggest that the resulting list will start from 0 and end at 10 (e
Hence, 10/11 would become step size for implementation shown below.
Also, we would try to generate a list with floating values as np.linspace would crea
"""
step2 = 10/11
stepnew2=([0 + step2*interval for interval in range(11)])
stepnew22=[round(item,8) for item in stepnew2]
print(stepnew22)

"""Implementation of np.arrange
"""
vector1=list(range(0,5,1))
print("Original vector1 is: ",vector1)
vector1=[x+1 for x in vector1]
print("Modified vector1 is: ",vector1)
vector1=[2*y for y in vector1]
print("Final vector1 is: ",vector1)
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0]
[0.0, 0.90909091, 1.81818182, 2.72727273, 3.63636364, 4.54545455, 5.45454545, 6.3636
3636, 7.27272727, 8.18181818, 9.09090909]
Original vector1 is:  [0, 1, 2, 3, 4]
Modified vector1 is:  [1, 2, 3, 4, 5]
Final vector1 is:  [2, 4, 6, 8, 10]
```

Looking at the output of the numpy array and the native python list, we recorgnise since we are using different function, the output values will be the same but the presicion will be different Hence after creating the native python list, we rounded up the values to 8 decimal places to get the same output as the output of numpy.

In [ ]:

```
#### part 3 ####
array_of_numbers = np.array([4, 8, 15, 16, 23, 42])
nnz = np.count_nonzero(array_of_numbers)
print(f"There are {nnz} non-zero numbers in the array.")
is_even = (array_of_numbers % 2 == 0)
is_greater_than_17 = (array_of_numbers > 17)
# is_even_and_greater_than_17 = is_even & is_greater_than_17
# print(array_of_numbers % 2 == 0 & array_of_numbers > 17)
```

```
There are 6 non-zero numbers in the array.
```

Reason for failure: The code has failed as it occurs when we use an invalid character in our code or a result of copy-pasting.

Reason for failure: The error occurs because instead of evaluating single values, we are evaluating on more than one element, therefore there is ambiguity in how to determine if the condition is true or not. We are evaluating multiple elements in the boolean context.

In [ ]:
```
##Corrected Code##
is_even_and_greater_than_17 = is_even.any() and is_greater_than_17.any()
print(array_of_numbers.all() % 2 == 0 & array_of_numbers.all() > 17)
```

```
False
```

part 4

Explain the output from np.where.

In [ ]:
```
np.where(array_of_numbers > 17)[0] # Returns the indices of the elements where value
```

Out[ ]:
```
array([4, 5], dtype=int64)
```

In [ ]:
```
np.where(array_of_numbers > 17, 1, 0)#numpy.where(condition[, x, y]), in this case i
```

Out[ ]:
```
array([0, 0, 0, 0, 1, 1])
```

Exercise 3, Part I: Finite Differences (FD) with Functions

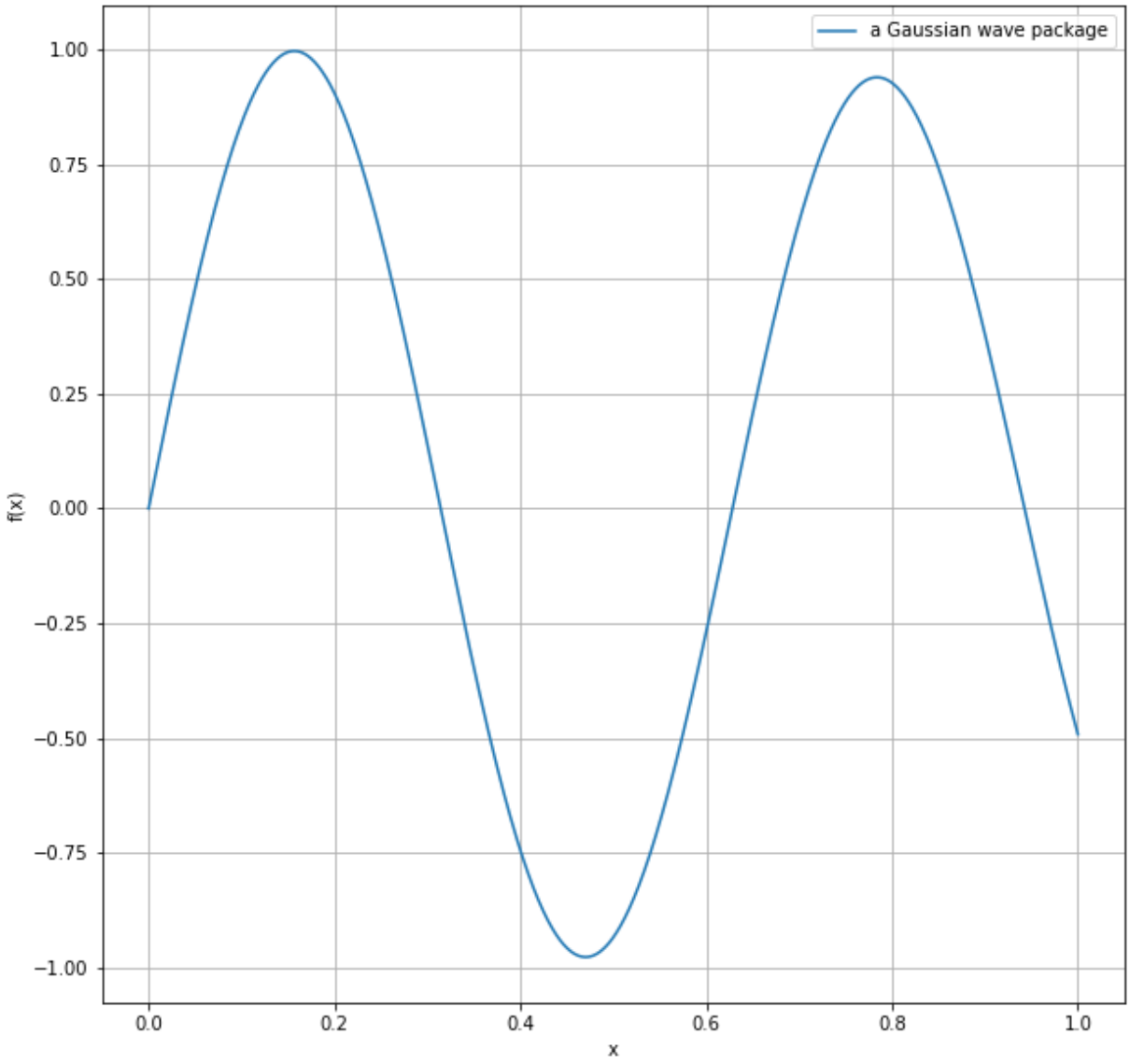
part 1

In [ ]:
```
# Import the package for data visualization
import matplotlib.pyplot as plt
```

In [ ]:
```
#function defined for wave phenomenon in python
def f(x, a=0.1, b=10):
    return np.sin(b*x)*np.exp(-a*x*x)
```

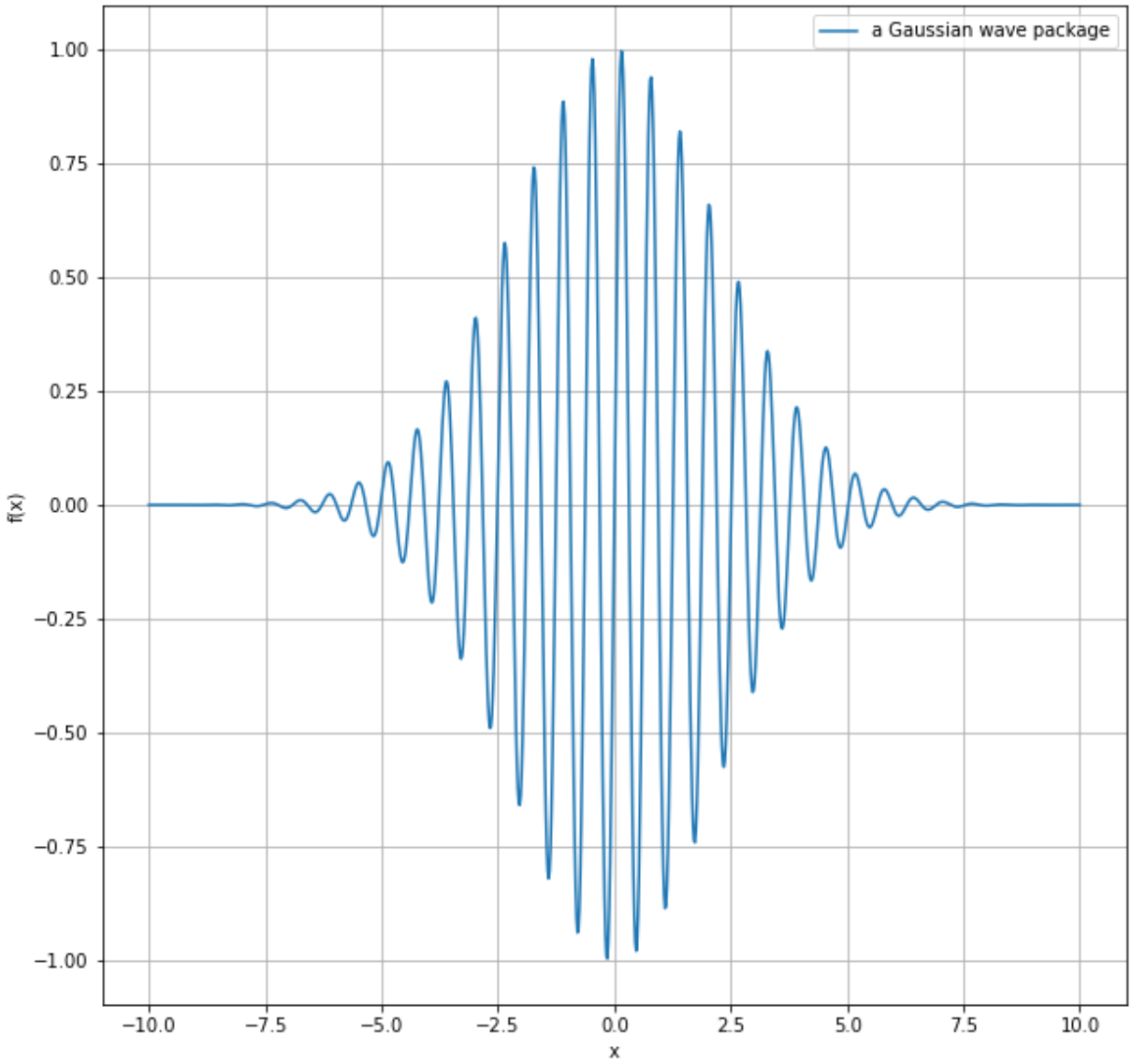
In [ ]:
```
# plot for f(x) from equation above for arbitary closed interval
def plot_f(x, F):
    plt.figure(figsize=(10,10))
    plt.plot(x, F, label="a Gaussian wave package")
    plt.xlabel("x")
    plt.ylabel("f(x)")
    plt.legend()
    plt.grid()
    plt.show()
```

In [ ]:
```
# Plot in arnbitary closed interval
X = np.linspace(0,1,1000)
plot_f(X, f(X))
```



In [ ]:

```
# plot f(x) in the range [-10, 10]
X = np.linspace(-10,10,1000)
plot_f(X, f(X))
```



part 2

Analytical derivative of f(x).

In [ ]:
```
#Python function that calculates the derivative
def der_f(x, a=0.1, b=10):
    return b* np.cos(b*x)* np.exp(-a*(x**2))-2*a*x*np.sin(b*x)*np.exp(-a*(x**2))
```

part 3

Function for the forward diffrence method.

In [ ]:
```
# calculates the derivative of an arbitrary function using the forward difference me
def fd(f,x,h):
    """ Evaluates the derivative of f(x) using forward difference method. h = step s
    return (f(x+h)-f(x))/h

print("analytical method:", der_f(1))
print("numerical method :",fd(f, 1, 1e-2))# x=1, h = 1e-2
```

```
analytical method: -7.493783027703379
numerical method : -7.220096595246589
```

part 4

Function for the central differance method

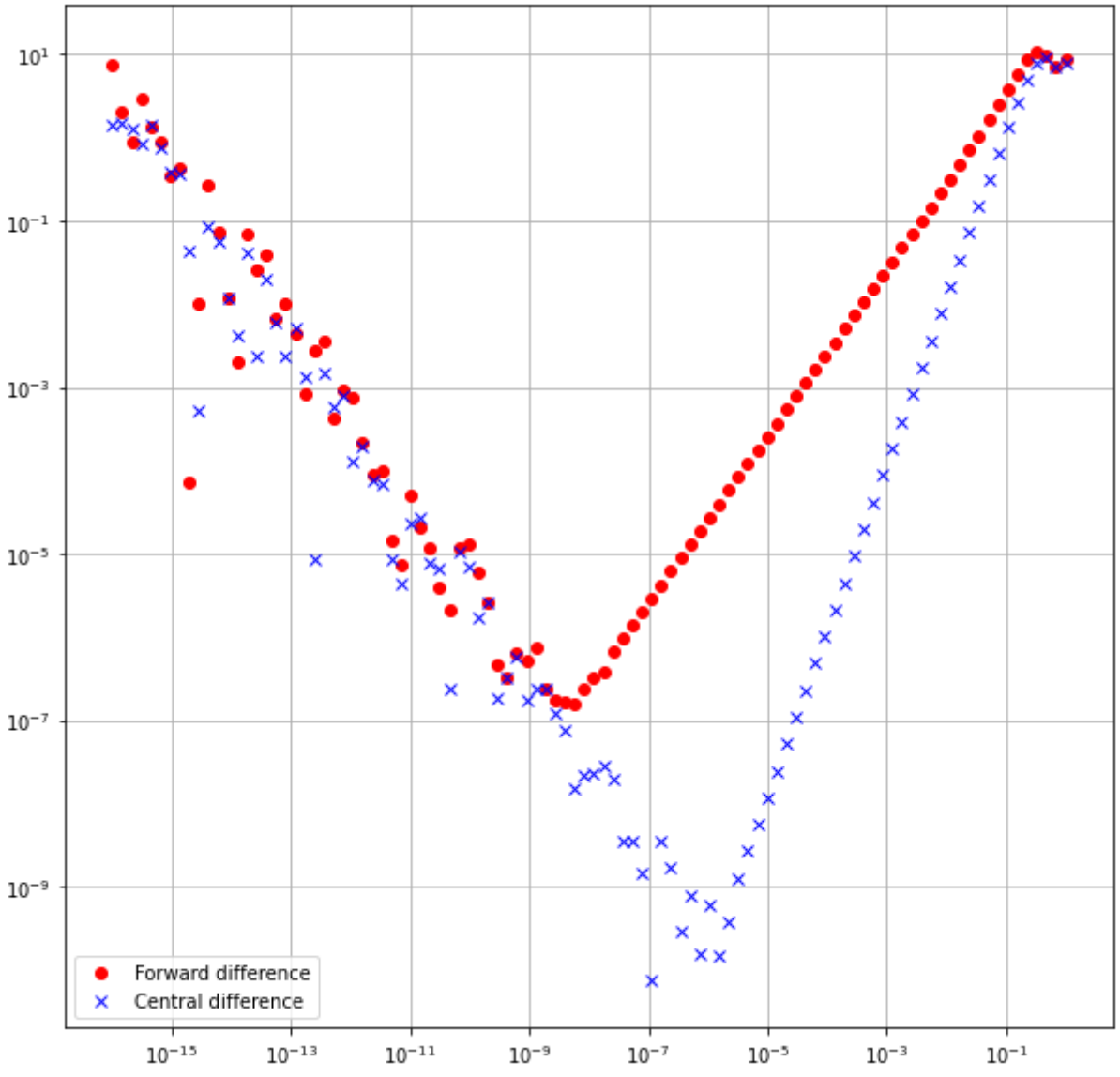
In [ ]:
```
def fc(f,x,h):
    """ Evaluates the derivative of f(x) using central difference method, h = step s
    return 0.5*(f(x+h)-f(x-h))/h

print("central method :",fc(f, 1, 1e-2))
```

```
central method : -7.48156774953081
```

part 5

Quantify the error in our numerical derivative approximations

In [ ]:
```
## quantify the error in our numerical derivative approximations for the point x = 1
h = np.logspace(-16, 0, 100)
x = 1
plt.figure(figsize=(10,10))
plt.plot(h, abs(der_f(x)-fd(f,x,h)), "or", label="Forward difference")
plt.plot(h, abs(der_f(x)-fc(f,x,h)), "xb", label="Central difference")
plt.grid()
plt.legend()
plt.xscale("log")
plt.yscale("log")
plt.show()
```



"""PART 5 COMMENTS:

1. For both approximation methods, the errors decrease with the decrease in step size (can be seen from right to left).
2. As can be seen on the plots, a minimum error is reached. Beyond that point, the error increases with further decrease in step size. For Forward Difference, the minimum error is reached at 10^-9, while that for Central difference is reached at 10^-7.
3. Also, it is observed that the numerical errors in Central Difference approximation method (CDM) is much less than the Forward Difference method (FDM). The reason is that error term in CDM is one order higher than that in FDM i.e as step size reduces by 10, error in FDM reduces by 10 while CDM reduces by 100.
4. The result was as expected if we had used Taylors formula.

"""

Exercise 3, Part II: FD with Classes

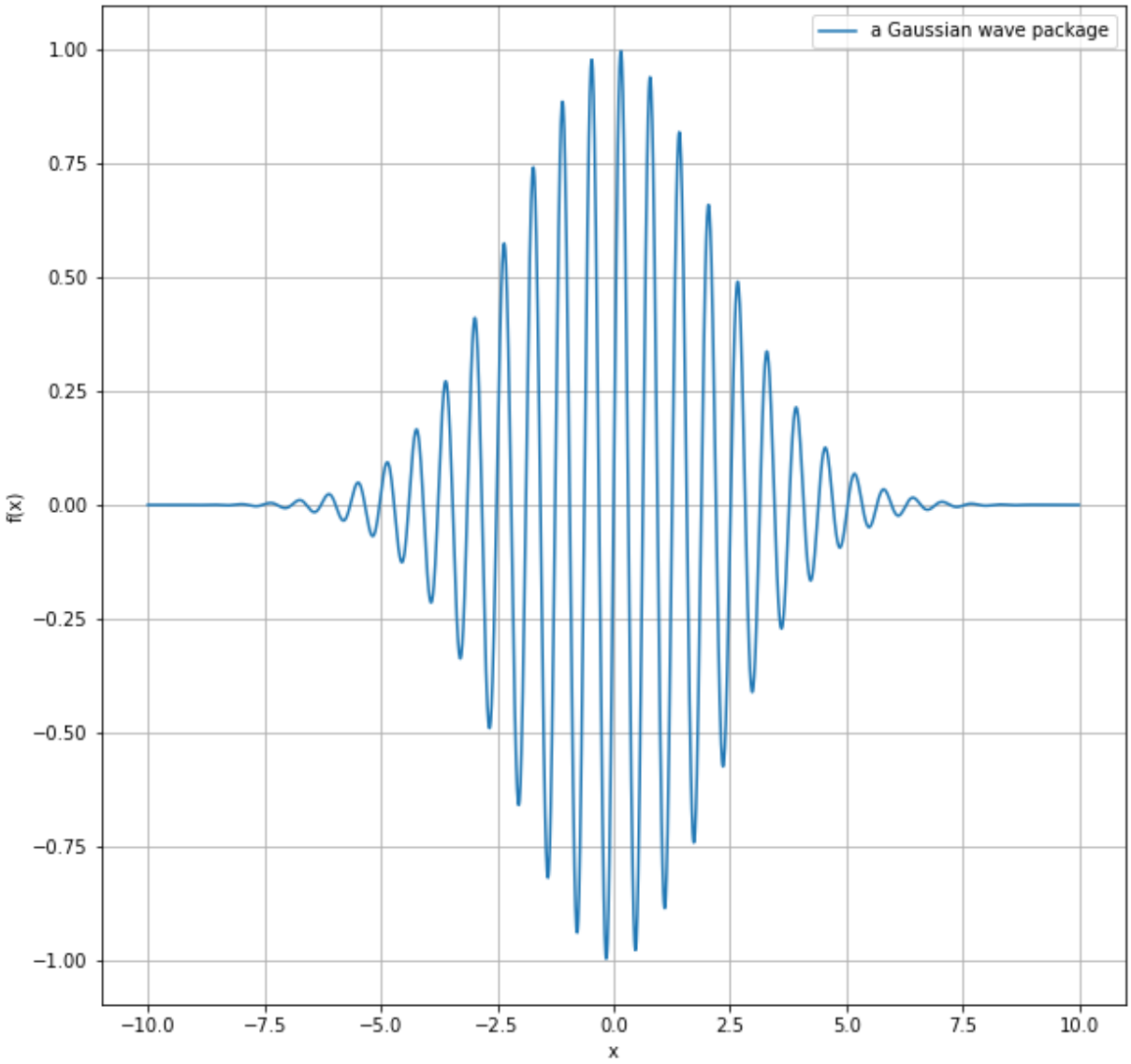
In [ ]:

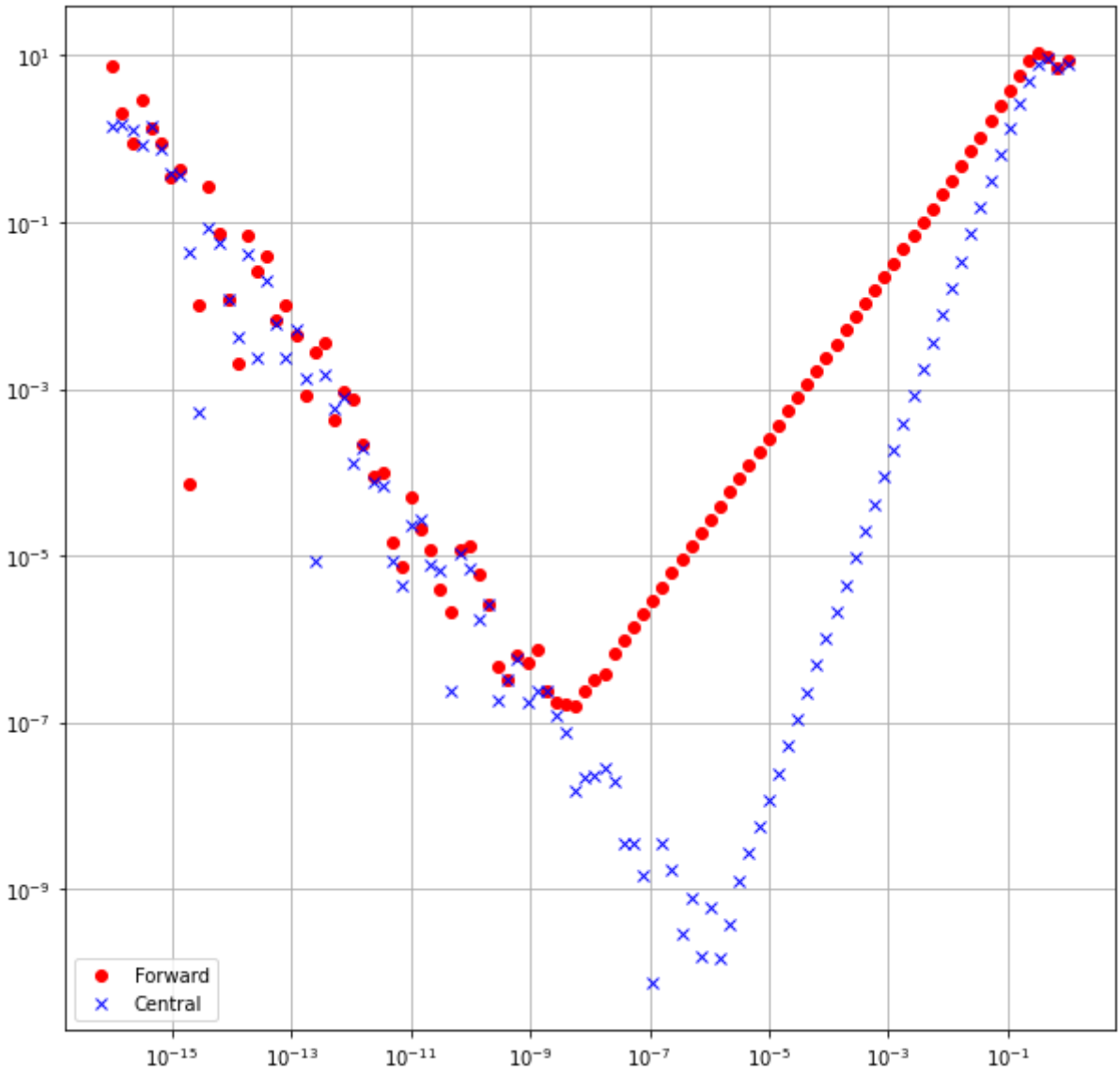
```
class WavePacket:
    def __init__(self, a, b, x, h): #constructor
        self.a = a
        self.b = b
        self.x = x
        self.h = h

    def f(self): #function defined for wave phenomenon in python
        return np.sin(self.b*self.x)*np.exp(-self.a*self.x*self.x)

    def der_f(self): # analytical derivative of f(x)
        return self.b* np.cos(self.b*self.x)* np.exp(-self.a*(self.x**2))-2*self.a*s

    def fd(self):  # calculates the derivative of an arbitrary function using the fo
        return (f(self.x+self.h)-f(self.x))/self.h

    def fc(self): # calculates the derivative of an arbitrary function using the cen
        return 0.5*(f(self.x+self.h)-f(self.x-self.h))/self.h

    def plot(self, x_min=-10, x_max=10, dx=0.01): # plot f(x) in the range [-10, 10]
        x = np.arange(x_min, x_max, dx)
        plt.figure(figsize=(10,10))
        plt.plot(x, f(x), label="a Gaussian wave package")
        plt.xlabel("x")
        plt.ylabel("f(x)")
        plt.legend()
        plt.grid()
        plt.show()

    def scatter_error(self): #  scatter plots showing the absolute error of the two
        plt.figure(figsize=(10,10))
        plt.plot(self.h, abs(der_f(self.x)-fd(f,self.x,self.h)), "or", label="Forwar
        plt.plot(self.h, abs(der_f(self.x)-fc(f,self.x,self.h)), "xb", label="Centra
        plt.grid()
        plt.legend()
        plt.xscale("log")
        plt.yscale("log")
        plt.show()
```

In [ ]:

```
h = np.logspace(-16, 0, 100)
```

```
x = 1
WP1 = WavePacket(0.1, 2, x, h) # a=0.1, b=2
WP1.plot()
WP1.scatter_error()
```





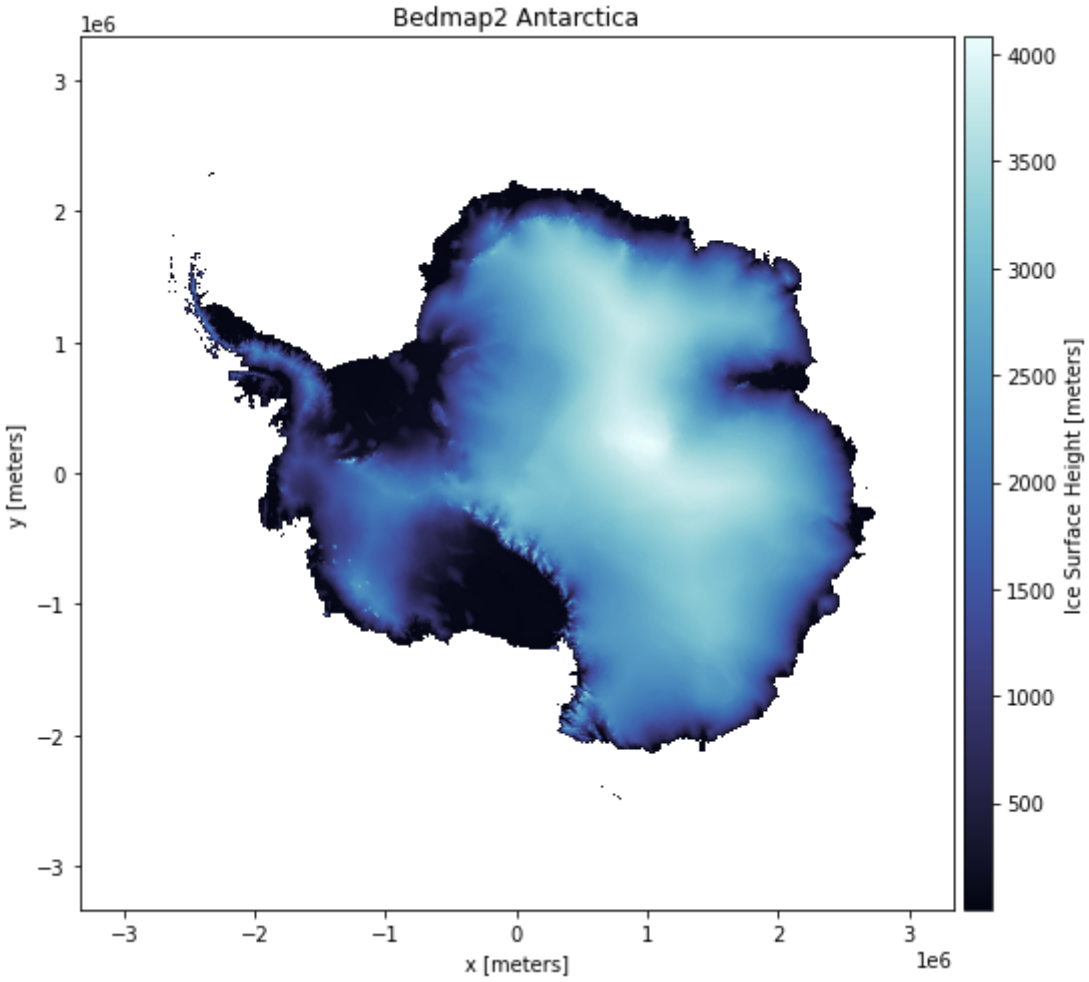
Exercise 4: A song of ice and fire?

part 1

Import glacier dataset from external source and plot the glacier.

In [ ]:

```
import rockhound as rh
import cmocean
bedmap = rh.fetch_bedmap2(datasets=["thickness","surface","bed"])
plt.figure(figsize=(8, 7))
ax = plt.subplot(111)
bedmap.surface.plot.pcolormesh(ax=ax,cmap=cmocean.cm.ice,cbar_kwargs=dict(pad=0.01,
plt.title("Bedmap2 Antarctica")
plt.tight_layout()
plt.show()
```



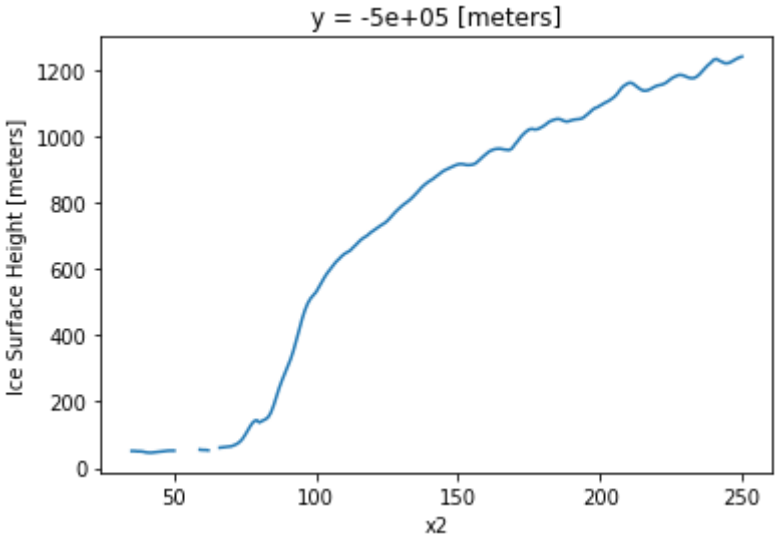
part 2

Slicing the data to get a 1d representation of an area on the glacier.

In [ ]:
```
# Extract cross section using the original coordinates
bed1d = bedmap.sel(y=-0.5e6, x=slice(-1.6e6,-1.35e6))
# Add a second x-coordinate to make prettier plots
# (shift x-axis to start at zero, and convert from m to km)
bed1d = bed1d.assign_coords({"x2":((bed1d.x+1.6e6)/1e3)})
```

In [ ]:
```
bed1d.surface.plot(x='x2') # plot surface elevation
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x18391816730>]
```

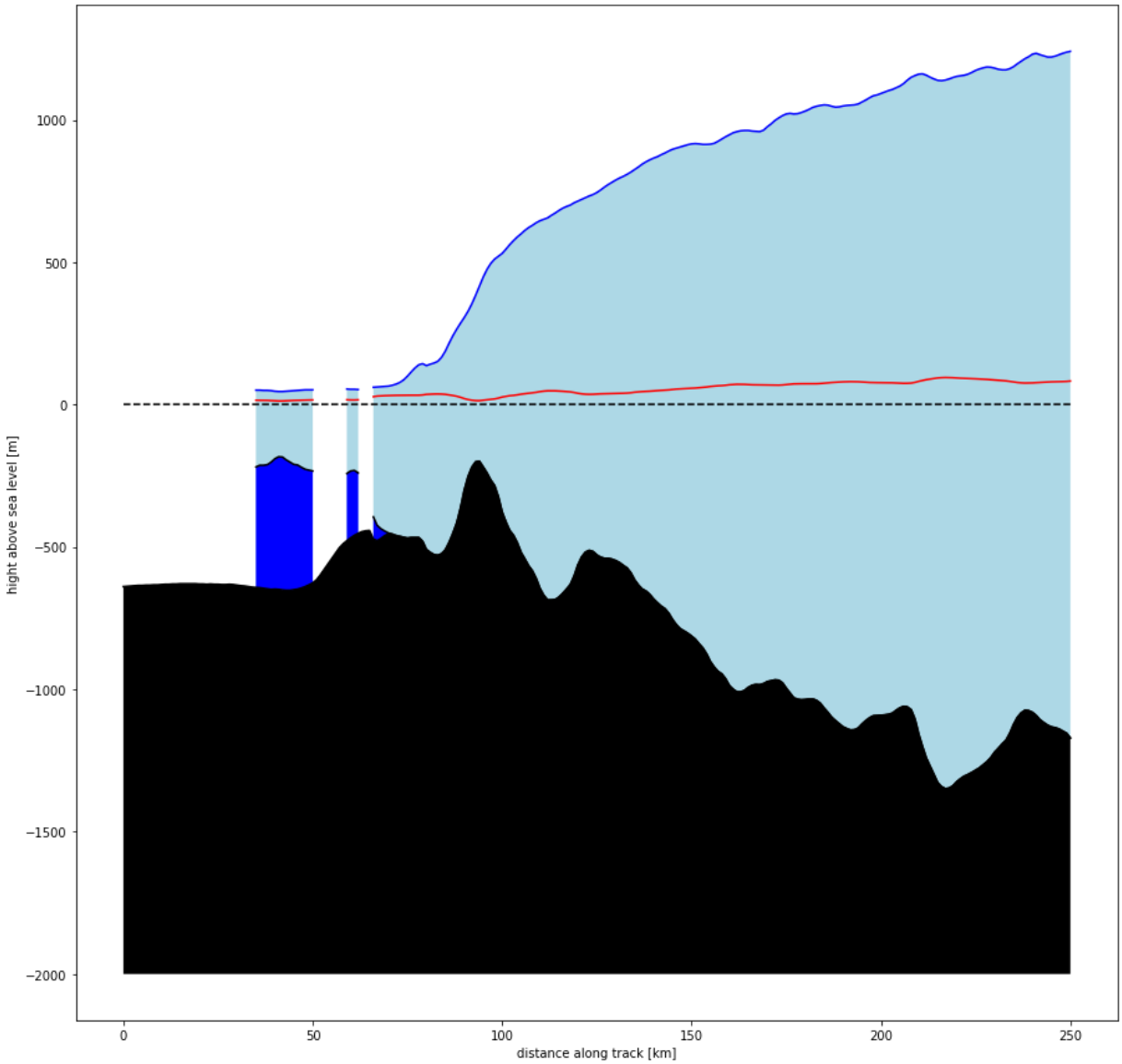


In [ ]:

```
def Freebord_level(surface, thickness):
    p_w = 1.0 # density of water
    p_ice = 0.9340 # density of ice
    return -(surface-thickness)*((p_w/p_ice)-1) # Freeboard level
```

In [ ]:

```
surface = bed1d.surface.values # surface elevation
thickness = bed1d["thickness"].values # ice thickness
water = surface-thickness  # water depth
x = np.arange(0.0,len(bed1d.bed.values), 1) # x-axis

h = Freebord_level(surface, thickness)
plt.figure(figsize=(15,15))
plt.plot(h, "-r")
plt.plot(surface, "b")
plt.plot([0]*len(thickness), "--k")
plt.plot(water, "-k")
plt.plot(bed1d.bed.values, "k")
plt.fill_between(x, bed1d.bed.values, -2000, facecolor="k") # fill the bedrock with
plt.fill_between(x, surface, bed1d.bed.values, facecolor="lightblue")  # fill the ic
plt.fill_between(x, surface-thickness, bed1d.bed.values, facecolor="blue")  # fill t
plt.ylabel("hight above sea level [m]")
plt.xlabel("distance along track [km]")
plt.show()
```



From the figure, we can see that the first 40 km of data for the ocean is missing. After investigating, we realised this is because the dataset shows NaN values for that part. Apart from that, we can observe the bedrock in black color, The sea in deep blue color and the ice in light blue color. We can also observe freeboard level in a red line. We observe that the ice is thickest at over 1000 meters above sea level and thinnest at about some 10 meters above sea level.

Yes, the shape of the bedrock is important to sea level rise. Geologically speaking, it is necessary to recognise that when there exist accomodation space, sea level will generally be low however in instances where there is not, then we expect sea level to rise. With respect to this exercise, it can be observed that the ice thickness turns to increase along tracks. However, we can also observe that the bedrock of antarctica though undulating, generally turn to decrease in elevation along track. This inclination can create acomodation space for the melting ice to occupy once it melts away. Hence the shape of the bedrock can play a role to sea leve rise.

part 3

calculating SLR if all the ice in antarctica chould melt.

In [ ]:

```
surface = bedmap.surface.values # surface elevation
thickness = bedmap.thickness.values # thickness of ice
sea_A = 361*10**6 # km^2
h = Freebord_level(surface, thickness)
h_s = surface - h # sea level
```

```
h_s = h_s[~np.isnan(h_s)] # remove nan values

h_s = np.sum(h_s) # sum of all sea level
h_s = h_s/sea_A # average sea level

print("approximately",h_s, "m")
```

```
approximately 74.26344758963609 m
```

part 4

The bedmap could help us understand and plan future SLR as well as keep track of changes in the glacier.

The limitations of the approximation is that we do not account for the new area the sea will occupy by rising. As the sea level rises more and more land area will be occupied by water, this means that the estimat for SLR will be somewhat lower then in the calulation.

Conclusion

We learned from this project that the built-in Python methods and libraries interpret computations in various ways. Although about the same, their output has varied precision. When working on a large project, this precision issue can result in numerical error. As a result, when working on a big project, it's crucial to be cautious and consistent with the kind of functions that are employed. Nevertheless, we were astonished by how simple it was to manipulate and visualize a massive dataset using python modules like numpy and matplot (which were utilized in this research). However, there are other ways to introduce errors into computation besides concentrating on the type of functions utilized. As we learned in Exercise 3, sometimes it's preferable to keep things straightforward. In this instance, we found that working with classes had the same outcomes as part 1 as we saw in part 2. Working in classes also required less computation, which decreased the chance of error introduction. Another option for performing computations is to obtain data and functions from a reliable external source. We used one of these sources in Exercise 4, which also had its own specified functions. This made computation very simple and this may have minimized the chance of computational error as we weren't defining and computing function but rather we were calling functions from the source and using them for our computation.

Self-reflection

Edward: I am fortunate to have team members who are highly motivated and committed to their work. We had our first meeting within the first week after the project was assigned, and their enthusiasm inspired me to begin the project early. Because of this, we were able to complete this job on schedule, days before the deadline. Being aware of the precision that emerges from the type of Python library used while doing tasks is, in my opinion, the least of my concerns as a programmer with training in petroleum engineering and limited programming experience. I don't give my code's structure any thought as long as it functions. But completing this exercise has made me aware that minor numerical inaccuracies brought on by libraries and my code's organization could eventually result in errors in a larger project.

Ådne: It took some time to get a group member meeting, so I started the working on the project on my own. in hindsight I should probably have waited so that we could divide up the

exercises within the group. We ended up working mostly separately by going through the exercises on our own and den merging them together in the end. It would probably be more efficient to divide the project so that each team member does a different part (but helping each other if needed) and then in the end merging the project together and explaining our findings. However, this could result in that some members lose a learning experience by not doing a part. The project itself was well informed and had mutable learning experiences. It was interesting to claculate and see the SLR if antartica should melt.

Subhashree: The project provided good overview to the numerical error and implications it can bring while working on large datasets. The assignment provided good hands on to work with python libraries, functions and classes. I worked on resolving the coding part first and then worked on code explanation and final observations at the end. It was really nice to work in this team and to share our learnings with each other.