In [441]:
```
import numpy as np
import math
from matplotlib import gridspec
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

GPGN 411/511: Advanced Gravity and Magnetic Exploration

LAB 08: Understanding Errors in Wavenumber-Domain Processing of Potential-field Data

Nadima Dwihusna and Felicia Nurindrawati

Due: 11/21/2017

A. Objectives and Explanation of Concepts Covered

The main objective of this lab is to explore errors that could result from violating the assumption of a planar observation surface when using a wavenumber-domain processing methods using the codes developed in lab excercise 03 and 06. The relevant concepts that are covered include wavenumber-domain processing, planar surface assumption, and errors from uneven observation surfaces.

The main concept in this lab is covering the important assumption used in deriving the wavenumber-domain expression of potential fields and the associated processing is a planar obervation field, for example, the vertical coordinated of the observation is constant. Overall, labs 04 and 07 have relied on the assumption that illustrates the utility of such processing. As stated in the lecture, this assumption is often not met when the data observed on an uneven surface such as a rugged topography or in a draped airborne survey. Processing this kind of data using wavenumber domain approach may lead to large amount of errors. In a whole, this lab is designed so that the erros can be examined from applying these wavenumber operators to data on the uneven surface.

B. Summary/Description of Program

The following is summary describing the program for each tasks that was developed for this lab. Brief instructions are also included for each of the programs.

For Task 1, the code from Lab 06 was modified to allow the observation to be on an uneven surface. The lab 06 code was modified to input the observation locations of the data grid with their respective (x,y,z) coordinates allowing the code to be much more general and useful for a broad range of modeling. The user was allowed to input the new updated file of the observation location. Additionally, this program generates a grid centered at the given Northing, Easting, and elevation parameters given in the lab. In this program, the total-field anomaly data was also produced by a given magnetic cube with the parameters given in the lab handout. In the end, the program is made to calculate the total-field anomaly data produced by this cubic source located above the Gaussian hill for two given inducing field directiosn of (I,D)=(45,25) and (I,D)=(90,0). The program then plots the two datasets.

For Task 2, the code from Lab 06 was modified to add a reduction-to-pole operation which is applicable to the frist data set in task 1. Afterwards, the program allows the user to apply the wavenumber-domain RTP operation to the two data sets (data set 1a and data set 2a in order to be reduced to the pole. Afterwards, the results called are then set to data 1c and 2c respectively. In the end, the data set 1c and data set 2c with the corresponding true RTP of data set 1b and data set 1b are then plotted along with their asociated differences. The magnitude and patterns of the data and differences are then observed. In the end, the errors are also produced by the violation of the planar surface observations.

For Task 3, the program allows the user to focus on the Gaussian-hill observation surface. This allows the user to investigate the change in error levels for a sequence of hill height h=0, 25, 50, 100, 200, 400 m by computing the L2 or square root of the sum of difference squared norm of the differences between the two and computed RTP data. In the end, the program allows the user to plot the result as a function of the height.

C. Tasks

Task 1

Modify the code from Lab 06 to allow the observation to be on an uneven surface. The Lab 06 code assumed an observation plant at z=0, so the code need to be modified to input the observation locations of the data grid with their (x,y,z) coordinates. This will also make the code general and useful for a broad range of modeling.

In [471]:
```
def task1(X, Y, Z, Imi, Dmi):
###Scenario-1
    Xinp=X
    Yinp=Y
    Zinp=Z
    Im = Imi
    Dm = Dmi
    Mm = 2
###### Input Bo information
    I = Imi
    D = Dmi
    muo = 4*np.pi*10**(-7)
```

```
    BoX = np.cos(I*np.pi/180)*np.cos(D*np.pi/180)
    BoY = np.cos(I*np.pi/180)*np.sin(D*np.pi/180)
    BoZ = np.sin(I*np.pi/180)
    Bo = [BoX, BoY, BoZ]

    Boh = [Bo[0]/50000, Bo[1]/50000, Bo[2]/50000]
    Mx = Mm*np.cos(Im*np.pi/180)*np.cos(Dm*np.pi/180)
    My = Mm*np.cos(Im*np.pi/180)*np.sin(Dm*np.pi/180)
    Mz = Mm*np.sin(Im*np.pi/180)
    M = [Mx, My, Mz]

    Txx = np.zeros((len(Xinp),len(Yinp)))
    Tyy = np.zeros((len(Xinp),len(Yinp)))
    Tzz = np.zeros((len(Xinp),len(Yinp)))
    Txy = np.zeros((len(Xinp),len(Yinp)))
    Txz = np.zeros((len(Xinp),len(Yinp)))
    Tyz = np.zeros((len(Xinp),len(Yinp)))
    for i in range(len(Xinp)):
        for j in range(len(Yinp)):
            Txx[i,j] = math.atan2((Yinp[j]-Ys2)*(Zinp[i,j]-Zs2),((Xinp[i]-X
s2)*math.sqrt((Xinp[i]-Xs2)**2 +(Yinp[j]-Ys2)**2+(Zinp[i,j]-Zs2)**2))) - ma
th.atan2((Yinp[j]-Ys2)*(Zinp[i,j]-Zs2),((Xinp[i]-Xs1)*math.sqrt((Xinp[i]-Xs
1)**2+(Yinp[j]-Ys2)**2+(Zinp[i,j]-Zs2)**2))) -  math.atan2((Yinp[j]-Ys1)*(Z
inp[i,j]-Zs2),((Xinp[i]-Xs2)*math.sqrt((Xinp[i]-Xs2)**2+(Yinp[j]-Ys1)**2+(Z
inp[i,j]-Zs2)**2))) + math.atan2((Yinp[j]-Ys1)*(Zinp[i,j]-Zs2),((Xinp[i]-Xs
1)*math.sqrt((Xinp[i]-Xs1)**2+(Yinp[j]-Ys1)**2+(Zinp[i,j]-Zs2)**2))) - math
.atan2((Yinp[j]-Ys2)*(Zinp[i,j]-Zs1),((Xinp[i]-Xs2)*math.sqrt((Xinp[i]-Xs2)
**2+(Yinp[j]-Ys2)**2+(Zinp[i,j]-Zs1)**2))) + math.atan2((Yinp[j]-Ys2)*(Zinp
[i,j]-Zs1),((Xinp[i]-Xs1)*math.sqrt((Xinp[i]-Xs1)**2+(Yinp[j]-Ys2)**2+(Zinp
[i,j]-Zs1)**2)))  + math.atan2((Yinp[j]-Ys1)*(Zinp[i,j]-Zs1),((Xinp[i]-Xs2)
*math.sqrt((Xinp[i]-Xs2)**2+(Yinp[j]-Ys1)**2+(Zinp[i,j]-Zs1)**2))) - math.a
tan2((Yinp[j]-Ys1)*(Zinp[i,j]-Zs1),((Xinp[i]-Xs1)*math.sqrt((Xinp[i]-Xs1)**
2+(Yinp[j]-Ys1)**2+(Zinp[i,j]-Zs1)**2)))
            Tyy[i,j] = math.atan2((Xinp[i]-Xs2)*(Zinp[i,j]-Zs2),((Yinp[j]-Y
s2)*math.sqrt((Yinp[j]-Ys2)**2+(Xinp[i]-Xs2)**2+(Zinp[i,j]-Zs2)**2))) - mat
h.atan2((Xinp[i]-Xs2)*(Zinp[i,j]-Zs2),((Yinp[j]-Ys1)*math.sqrt((Yinp[j]-Ys1
)**2+(Xinp[i]-Xs2)**2+(Zinp[i,j]-Zs2)**2))) -  math.atan2((Xinp[i]-Xs1)*(Zi
np[i,j]-Zs2),((Yinp[j]-Ys2)*math.sqrt((Yinp[j]-Ys2)**2+(Xinp[i]-Xs1)**2+(Zi
np[i,j]-Zs2)**2))) + math.atan2((Xinp[i]-Xs1)*(Zinp[i,j]-Zs2),((Yinp[j]-Ys1
)*math.sqrt((Yinp[j]-Ys1)**2+(Xinp[i]-Xs1)**2+(Zinp[i,j]-Zs2)**2))) - math.
atan2((Xinp[i]-Xs2)*(Zinp[i,j]-Zs1),((Yinp[j]-Ys2)*math.sqrt((Yinp[j]-Ys2)*
*2+(Xinp[i]-Xs2)**2+(Zinp[i,j]-Zs1)**2))) + math.atan2((Xinp[i]-Xs2)*(Zinp[
i,j]-Zs1),((Yinp[j]-Ys1)*math.sqrt((Yinp[j]-Ys1)**2+(Xinp[i]-Xs2)**2+(Zinp[
i,j]-Zs1)**2)))  + math.atan2((Xinp[i]-Xs1)*(Zinp[i,j]-Zs1),((Yinp[j]-Ys2)*
math.sqrt((Yinp[j]-Ys2)**2+(Xinp[i]-Xs1)**2+(Zinp[i,j]-Zs1)**2))) - math.at
an2((Xinp[i]-Xs1)*(Zinp[i,j]-Zs1),((Yinp[j]-Ys1)*math.sqrt((Yinp[j]-Ys1)**2
+(Xinp[i]-Xs1)**2+(Zinp[i,j]-Zs1)**2)))
            Tzz[i,j] = math.atan2((Xinp[i]-Xs2)*(Yinp[j]-Ys2),((Zinp[i,j]-Z
s2)*math.sqrt((Zinp[i,j]-Zs2)**2+(Xinp[i]-Xs2)**2+(Yinp[j]-Ys2)**2))) - mat
h.atan2((Xinp[i]-Xs2)*(Yinp[j]-Ys2),((Zinp[i,j]-Zs1)*math.sqrt((Zinp[i,j]-Z
s1)**2+(Xinp[i]-Xs2)**2+(Yinp[j]-Ys2)**2))) -  math.atan2((Xinp[i]-Xs1)*(Yi
np[j]-Ys2),((Zinp[i,j]-Zs2)*math.sqrt((Zinp[i,j]-Zs2)**2+(Xinp[i]-Xs1)**2+(
Yinp[j]-Ys2)**2))) + math.atan2((Xinp[i]-Xs1)*(Yinp[j]-Ys2),((Zinp[i,j]-Zs1
)*math.sqrt((Zinp[i,j]-Zs1)**2+(Xinp[i]-Xs1)**2+(Yinp[j]-Ys2)**2))) - math.
atan2((Xinp[i]-Xs2)*(Yinp[j]-Ys1),((Zinp[i,j]-Zs2)*math.sqrt((Zinp[i,j]-Zs2
)**2+(Xinp[i]-Xs2)**2+(Yinp[j]-Ys1)**2))) + math.atan2((Xinp[i]-Xs2)*(Yinp[
```

```
j]-Ys1),((Zinp[i,j]-Zs1)*math.sqrt((Zinp[i,j]-Zs1)**2+(Xinp[i]-Xs2)**2+(Yin
p[j]-Ys1)**2)))  + math.atan2((Xinp[i]-Xs1)*(Yinp[j]-Ys1),((Zinp[i,j]-Zs2)*
math.sqrt((Zinp[i,j]-Zs2)**2+(Xinp[i]-Xs1)**2+(Yinp[j]-Ys1)**2))) - math.at
an2((Xinp[i]-Xs1)*(Yinp[j]-Ys1),((Zinp[i,j]-Zs1)*math.sqrt((Zinp[i,j]-Zs1)*
*2+(Xinp[i]-Xs1)**2+(Yinp[j]-Ys1)**2)))
            Txy[i,j] = -math.log(math.sqrt((Xinp[i]-Xs2)**2+(Yinp[j]-Ys2)**
2+(Zinp[i,j]-Zs2)**2)-(Zinp[i,j]-Zs2))+math.log(math.sqrt((Xinp[i]-Xs1)**2+
(Yinp[j]-Ys2)**2+(Zinp[i,j]-Zs2)**2)-(Zinp[i,j]-Zs2))+math.log(math.sqrt((X
inp[i]-Xs2)**2+(Yinp[j]-Ys1)**2+(Zinp[i,j]-Zs2)**2)-(Zinp[i,j]-Zs2))-math.l
og(math.sqrt((Xinp[i]-Xs1)**2+(Yinp[j]-Ys1)**2+(Zinp[i,j]-Zs2)**2)-(Zinp[i,
j]-Zs2))+math.log(math.sqrt((Xinp[i]-Xs2)**2+(Yinp[j]-Ys2)**2+(Zinp[i,j]-Zs
1)**2)-(Zinp[i,j]-Zs1))-math.log(math.sqrt((Xinp[i]-Xs1)**2+(Yinp[j]-Ys2)**
2+(Zinp[i,j]-Zs1)**2)-(Zinp[i,j]-Zs1))-math.log(math.sqrt((Xinp[i]-Xs2)**2+
(Yinp[j]-Ys1)**2+(Zinp[i,j]-Zs1)**2)-(Zinp[i,j]-Zs1))+math.log(math.sqrt((X
inp[i]-Xs1)**2+(Yinp[j]-Ys1)**2+(Zinp[i,j]-Zs1)**2)-(Zinp[i,j]-Zs1))
            Txz[i,j] = -math.log(math.sqrt((Xinp[i]-Xs2)**2+(Zinp[i,j]-Zs2)
**2+(Yinp[j]-Ys2)**2)-(Yinp[j]-Ys2))+math.log(math.sqrt((Xinp[i]-Xs1)**2+(Z
inp[i,j]-Zs2)**2+(Yinp[j]-Ys2)**2)-(Yinp[j]-Ys2))+math.log(math.sqrt((Xinp[
i]-Xs2)**2+(Zinp[i,j]-Zs1)**2+(Yinp[j]-Ys2)**2)-(Yinp[j]-Ys2))-math.log(mat
h.sqrt((Xinp[i]-Xs1)**2+(Zinp[i,j]-Zs1)**2+(Yinp[j]-Ys2)**2)-(Yinp[j]-Ys2))
+math.log(math.sqrt((Xinp[i]-Xs2)**2+(Zinp[i,j]-Zs2)**2+(Yinp[j]-Ys1)**2)-(
Yinp[j]-Ys1))-math.log(math.sqrt((Xinp[i]-Xs1)**2+(Zinp[i,j]-Zs2)**2+(Yinp[
j]-Ys1)**2)-(Yinp[j]-Ys1))-math.log(math.sqrt((Xinp[i]-Xs2)**2+(Zinp[i,j]-Z
s1)**2+(Yinp[j]-Ys1)**2)-(Yinp[j]-Ys1))+math.log(math.sqrt((Xinp[i]-Xs1)**2
+(Zinp[i,j]-Zs1)**2+(Yinp[j]-Ys1)**2)-(Yinp[j]-Ys1))
            Tyz[i,j] = -math.log(math.sqrt((Yinp[j]-Ys2)**2+(Zinp[i,j]-Zs2)
**2+(Xinp[i]-Xs2)**2)-(Xinp[i]-Xs2))+math.log(math.sqrt((Yinp[j]-Ys1)**2+(Z
inp[i,j]-Zs2)**2+(Xinp[i]-Xs2)**2)-(Xinp[i]-Xs2))+math.log(math.sqrt((Yinp[
j]-Ys2)**2+(Zinp[i,j]-Zs1)**2+(Xinp[i]-Xs2)**2)-(Xinp[i]-Xs2))-math.log(mat
h.sqrt((Yinp[j]-Ys1)**2+(Zinp[i,j]-Zs1)**2+(Xinp[i]-Xs2)**2)-(Xinp[i]-Xs2))
+math.log(math.sqrt((Yinp[j]-Ys2)**2+(Zinp[i,j]-Zs2)**2+(Xinp[i]-Xs1)**2)-(
Xinp[i]-Xs1))-math.log(math.sqrt((Yinp[j]-Ys1)**2+(Zinp[i,j]-Zs2)**2+(Xinp[
i]-Xs1)**2)-(Xinp[i]-Xs1))-math.log(math.sqrt((Yinp[j]-Ys2)**2+(Zinp[i,j]-Z
s1)**2+(Xinp[i]-Xs1)**2)-(Xinp[i]-Xs1))+math.log(math.sqrt((Yinp[j]-Ys1)**2
+(Zinp[i,j]-Zs1)**2+(Xinp[i]-Xs1)**2)-(Xinp[i]-Xs1))

    Bax = np.zeros((len(Xinp), len(Yinp)))
    Bay = np.zeros((len(Xinp), len(Yinp)))
    Baz = np.zeros((len(Xinp), len(Yinp)))
    deltaT = np.zeros((len(Xinp), len(Yinp)))


    for i in range(len(Xinp)):
        for j in range(len(Yinp)):
            Bax[i,j] = (muo/(np.pi*4))*((Txx[i,j]*Mx) + (Txy[i,j]*My)+(Txz[
i,j]*Mz))/10**(-9)
            Bay[i,j] = (muo/(np.pi*4))*((Txy[i,j]*Mx) + (Tyy[i,j]*My)+(Tyz[
i,j]*Mz))/10**(-9)
            Baz[i,j] = (muo/(np.pi*4))*((Txz[i,j]*Mx) + (Tyz[i,j]*My)+(Tzz[
i,j]*Mz))/10**(-9)
            deltaT[i,j] = ((BoX*Bax[i,j] + BoY*Bay[i,j] + BoZ*Baz[i,j]))*10
**(-7)/50000
    return [Boh,Bay,Tzz,deltaT,BoX,BoY,BoZ]
```

Generate a grid centered with the following:

1. Northing (-775,800) at 25-m spacing
2. Easting (-775,800) at 25-m spacing
3. Elevation define by a Gaussian function $z(x,y) = he^{-(x^2+y^2)/\alpha^2}$ where $h = 150m$ and $\alpha = 200m$ (h refers as the height)

In [472]:
```
# Northing and Easting
Xinp = np.arange(-800,800,25)
Yinp = np.arange(-800,800,25)
h = 150
a=200

#Elevation
Zinp=np.zeros((len(Xinp),len(Yinp)))
for i in range(len(Xinp)):
    for j in range(len(Yinp)):
        Zinp[i,j] = h*np.exp(-(Xinp[i]**2+Yinp[j]**2)/a)
```

Assume a magnetic cube that is buried at a depth to the top of 50 m, has a dimension of 300 m on all sides and a magnetization of 2.0 A/m whose direction is alligned with the Earth's field direction. Calculate the total-field anomaly data produced by this cubic source on a planar observation surface at z=0 for two different inducing field directions:

1. (I,D)=(45,25): data set-1a
2. (I,D)=(90,0): data set-1b, which yields the true RTP field

In [473]:
```
Zs1 = 50
Zs2 = 350
Xs1 = -150
Xs2 = 300
Ys1 = -150
Ys2 = 150

Zflat = np.zeros((64,64))
data1a = task1(Xinp,Yinp,Zflat,45,25)
data1b = task1(Xinp,Yinp,Zflat,90,0)
```

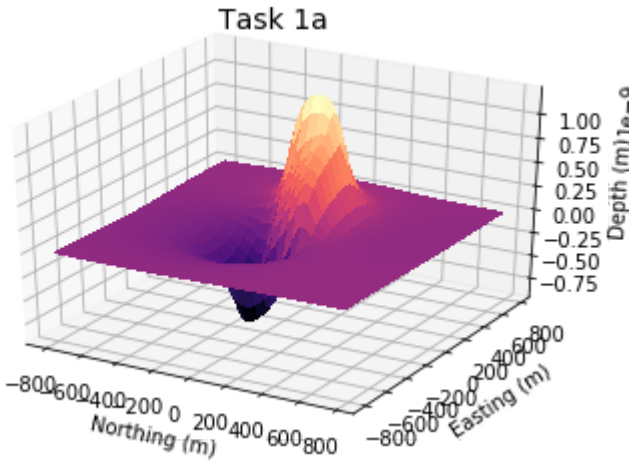
Calculate the total-field anomaly data produced by this cubic source the above Gaussian hill for two different inducing field directions:

1. (I,D)=(45,25): data set-2a
2. (I,D)=(90,0): data set-2b, which yields the true RTP field

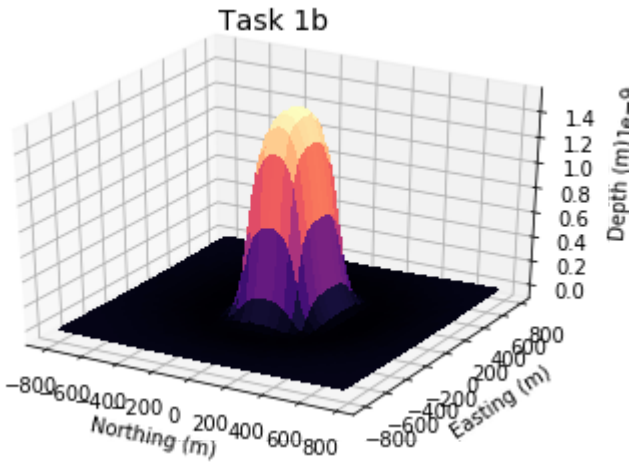
Plot the two data sets.

In [474]:
```
data2a = task1(Xinp,Yinp,Zinp,45,25)
data2b = task1(Xinp,Yinp,Zinp,90,0)
```

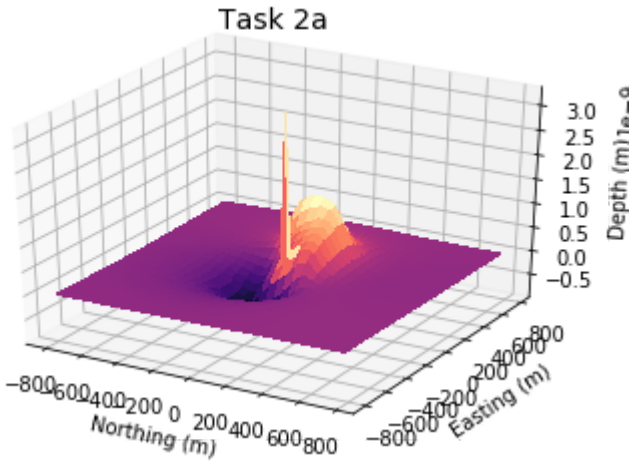
In [475]:
```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import numpy as np

fig = plt.figure()
ax = fig.gca(projection='3d')
X, Y = np.meshgrid(Xinp, Yinp)
surf = ax.plot_surface(X, Y, data1a[3], cmap=cm.magma,linewidth=0,antialiased=
False)
ax.set_xlabel('Northing (m)')
ax.set_ylabel('Easting (m)')
ax.set_zlabel('Depth (m)')
ax.set_title('Task 1a', fontsize=14)
plt.autoscale(enable=True, axis='x', tight=True)
plt.show()
```



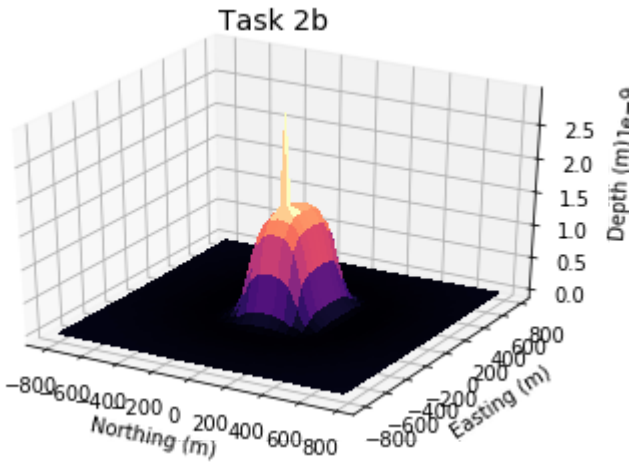
In [476]:
```
fig = plt.figure()
ax = fig.gca(projection='3d')
X, Y = np.meshgrid(Xinp, Yinp)
surf = ax.plot_surface(X, Y, data1b[3], cmap=cm.magma,linewidth=0,antialiased=
False)
ax.set_xlabel('Northing (m)')
ax.set_ylabel('Easting (m)')
ax.set_zlabel('Depth (m)')
ax.set_title('Task 1b', fontsize=14)
plt.autoscale(enable=True, axis='x', tight=True)
plt.show()
```



In [477]:
```
fig = plt.figure()
ax = fig.gca(projection='3d')
X, Y = np.meshgrid(Xinp, Yinp)
surf = ax.plot_surface(X, Y, data2a[3] ,cmap=cm.magma,linewidth=0,antialiased=
False)
ax.set_xlabel('Northing (m)')
ax.set_ylabel('Easting (m)')
ax.set_zlabel('Depth (m)')
ax.set_title('Task 2a', fontsize=14)
plt.autoscale(enable=True, axis='x', tight=True)
plt.show()
```

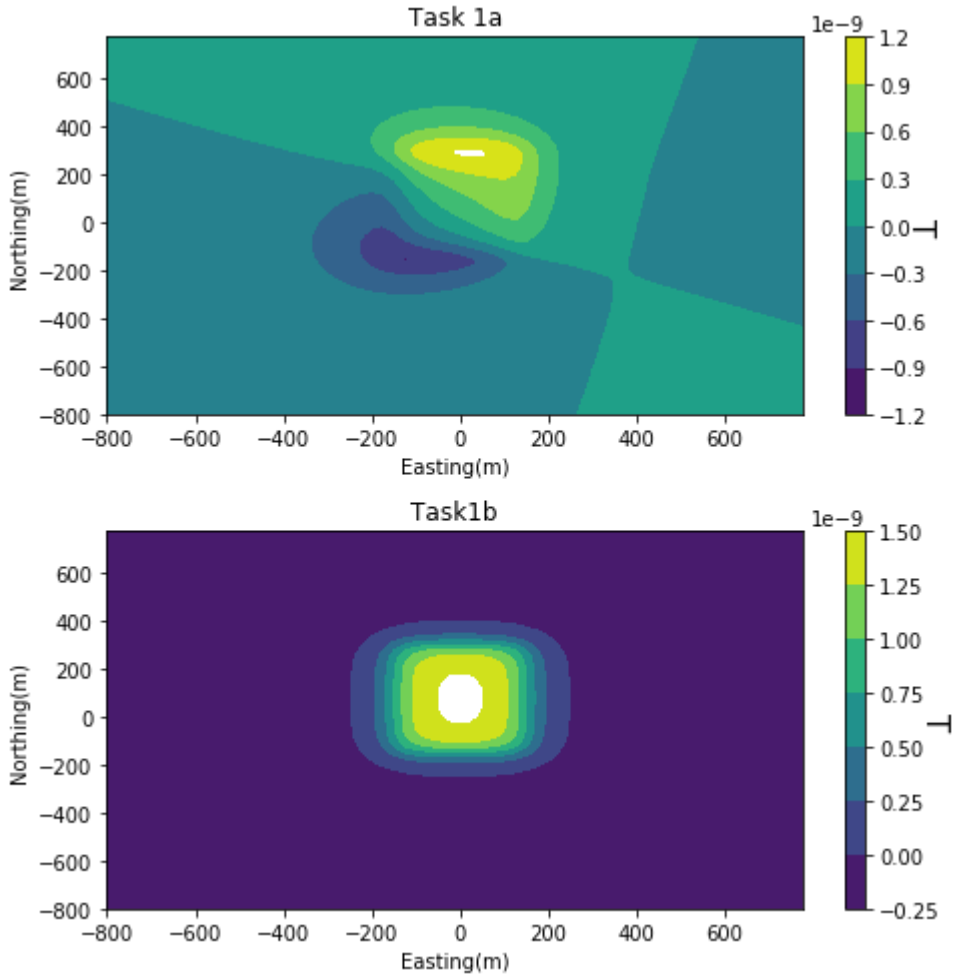


In [478]:
```
fig = plt.figure()
ax = fig.gca(projection='3d')
X, Y = np.meshgrid(Xinp, Yinp)
surf = ax.plot_surface(X, Y, data2b[3] ,cmap=cm.magma,linewidth=0,antialiased=
False)
ax.set_xlabel('Northing (m)')
ax.set_ylabel('Easting (m)')
ax.set_zlabel('Depth (m)')
ax.set_title('Task 2b', fontsize=14)
plt.autoscale(enable=True, axis='x', tight=True)
plt.show()
```

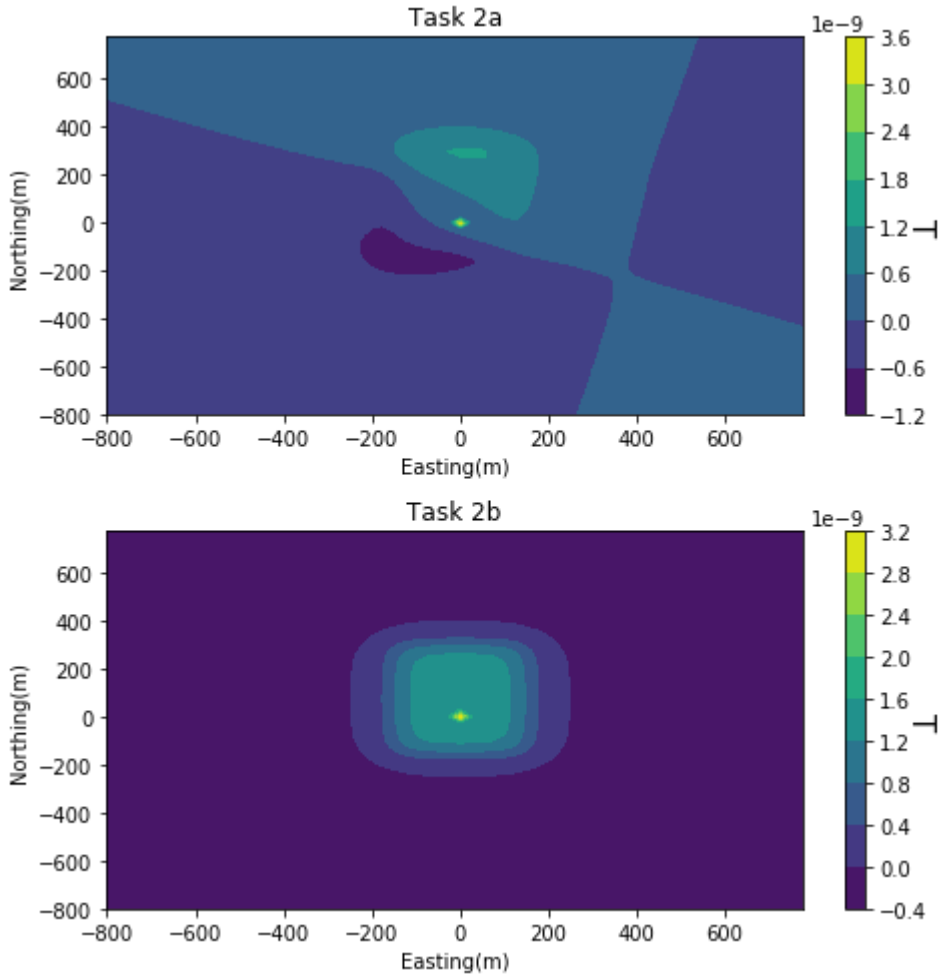


In [479]:
```
def plotting(something,x,y):
    ax3 = plt.contourf(x, y, something, colormap='hot')
    ax3 = plt.xlabel('Easting(m)')
    ax3 = plt.ylabel('Northing(m)')
    cbar = plt.colorbar()
    cbar.set_label('T',size = 15, rotation=270)
    return()
```

In [480]:
```
plt.figure(figsize=(7,7))
plt.subplot(211)
plt.title('Task 1a')
plotting(data1a[3],Xinp,Yinp)
plt.subplot(212)
plt.title('Task1b')
plotting(data1b[3],Xinp,Yinp)
plt.tight_layout()
plt.show()
```



In [481]:
```
plt.figure(figsize=(7,7))
plt.subplot(211)
plt.title('Task 2a')
plotting(data2a[3],Xinp,Yinp)
plt.subplot(212)
plt.title('Task 2b')
plotting(data2b[3],Xinp,Yinp)
plt.tight_layout()
plt.show()
```



Task 2

Modify the codes from Lab 06 to add a reduction to pole operation (which is applicable to the first data set in task 1)

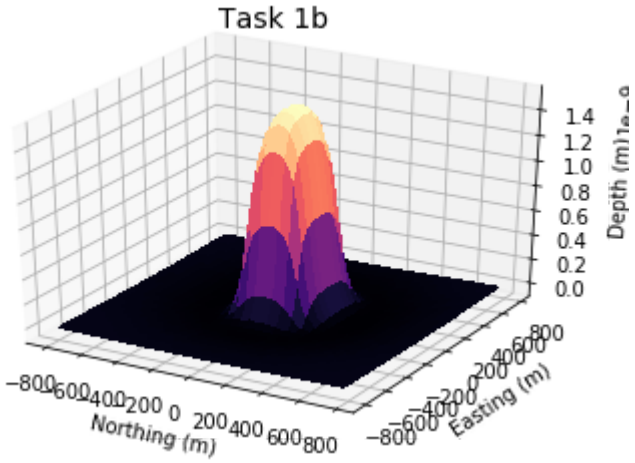
In [482]:
```
# RTP equation (Wr^2)/(k*B)^2

def task2(X, Y, Z, Imi, Dmi):
###Scenario-1
    Xinp=X
    Yinp=Y
    Zinp=Z
    Im = Imi+45
    Dm = Dmi-25
```

```
    Mm = 2
###### Input Bo information
    I = Imi+45
    D = Dmi-25
    muo = 4*np.pi*10**(-7)

    BoX = np.cos(I*np.pi/180)*np.cos(D*np.pi/180)
    BoY = np.cos(I*np.pi/180)*np.sin(D*np.pi/180)
    BoZ = np.sin(I*np.pi/180)
    Bo = [BoX, BoY, BoZ]
    Mx = Mm*np.cos(Im*np.pi/180)*np.cos(Dm*np.pi/180)
    My = Mm*np.cos(Im*np.pi/180)*np.sin(Dm*np.pi/180)
    Mz = Mm*np.sin(Im*np.pi/180)
    M = [Mx, My, Mz]

    Txx = np.zeros((len(Xinp),len(Yinp)))
    Tyy = np.zeros((len(Xinp),len(Yinp)))
    Tzz = np.zeros((len(Xinp),len(Yinp)))
    Txy = np.zeros((len(Xinp),len(Yinp)))
    Txz = np.zeros((len(Xinp),len(Yinp)))
    Tyz = np.zeros((len(Xinp),len(Yinp)))
    for i in range(len(Xinp)):
        for j in range(len(Yinp)):
            Txx[i,j] = math.atan2((Yinp[j]-Ys2)*(Zinp[i,j]-Zs2),((Xinp[i]-Xs2)
*math.sqrt((Xinp[i]-Xs2)**2 +(Yinp[j]-Ys2)**2+(Zinp[i,j]-Zs2)**2))) - math.ata
n2((Yinp[j]-Ys2)*(Zinp[i,j]-Zs2),((Xinp[i]-Xs1)*math.sqrt((Xinp[i]-Xs1)**2+(Yi
np[j]-Ys2)**2+(Zinp[i,j]-Zs2)**2))) -  math.atan2((Yinp[j]-Ys1)*(Zinp[i,j]-Zs2
),((Xinp[i]-Xs2)*math.sqrt((Xinp[i]-Xs2)**2+(Yinp[j]-Ys1)**2+(Zinp[i,j]-Zs2)**
2))) + math.atan2((Yinp[j]-Ys1)*(Zinp[i,j]-Zs2),((Xinp[i]-Xs1)*math.sqrt((Xinp
[i]-Xs1)**2+(Yinp[j]-Ys1)**2+(Zinp[i,j]-Zs2)**2))) - math.atan2((Yinp[j]-Ys2)*
(Zinp[i,j]-Zs1),((Xinp[i]-Xs2)*math.sqrt((Xinp[i]-Xs2)**2+(Yinp[j]-Ys2)**2+(Zi
np[i,j]-Zs1)**2))) + math.atan2((Yinp[j]-Ys2)*(Zinp[i,j]-Zs1),((Xinp[i]-Xs1)*m
ath.sqrt((Xinp[i]-Xs1)**2+(Yinp[j]-Ys2)**2+(Zinp[i,j]-Zs1)**2)))  + math.atan2
((Yinp[j]-Ys1)*(Zinp[i,j]-Zs1),((Xinp[i]-Xs2)*math.sqrt((Xinp[i]-Xs2)**2+(Yinp
[j]-Ys1)**2+(Zinp[i,j]-Zs1)**2))) - math.atan2((Yinp[j]-Ys1)*(Zinp[i,j]-Zs1),
((Xinp[i]-Xs1)*math.sqrt((Xinp[i]-Xs1)**2+(Yinp[j]-Ys1)**2+(Zinp[i,j]-Zs1)**2
)))
            Tyy[i,j] = math.atan2((Xinp[i]-Xs2)*(Zinp[i,j]-Zs2),((Yinp[j]-Ys2)
*math.sqrt((Yinp[j]-Ys2)**2+(Xinp[i]-Xs2)**2+(Zinp[i,j]-Zs2)**2))) - math.atan
2((Xinp[i]-Xs2)*(Zinp[i,j]-Zs2),((Yinp[j]-Ys1)*math.sqrt((Yinp[j]-Ys1)**2+(Xin
p[i]-Xs2)**2+(Zinp[i,j]-Zs2)**2))) -  math.atan2((Xinp[i]-Xs1)*(Zinp[i,j]-Zs2
),((Yinp[j]-Ys2)*math.sqrt((Yinp[j]-Ys2)**2+(Xinp[i]-Xs1)**2+(Zinp[i,j]-Zs2)**
2))) + math.atan2((Xinp[i]-Xs1)*(Zinp[i,j]-Zs2),((Yinp[j]-Ys1)*math.sqrt((Yinp
[j]-Ys1)**2+(Xinp[i]-Xs1)**2+(Zinp[i,j]-Zs2)**2))) - math.atan2((Xinp[i]-Xs2)*
(Zinp[i,j]-Zs1),((Yinp[j]-Ys2)*math.sqrt((Yinp[j]-Ys2)**2+(Xinp[i]-Xs2)**2+(Zi
np[i,j]-Zs1)**2))) + math.atan2((Xinp[i]-Xs2)*(Zinp[i,j]-Zs1),((Yinp[j]-Ys1)*m
ath.sqrt((Yinp[j]-Ys1)**2+(Xinp[i]-Xs2)**2+(Zinp[i,j]-Zs1)**2)))  + math.atan2
((Xinp[i]-Xs1)*(Zinp[i,j]-Zs1),((Yinp[j]-Ys2)*math.sqrt((Yinp[j]-Ys2)**2+(Xinp
[i]-Xs1)**2+(Zinp[i,j]-Zs1)**2))) - math.atan2((Xinp[i]-Xs1)*(Zinp[i,j]-Zs1),
((Yinp[j]-Ys1)*math.sqrt((Yinp[j]-Ys1)**2+(Xinp[i]-Xs1)**2+(Zinp[i,j]-Zs1)**2
)))
            Tzz[i,j] = math.atan2((Xinp[i]-Xs2)*(Yinp[j]-Ys2),((Zinp[i,j]-Zs2)
*math.sqrt((Zinp[i,j]-Zs2)**2+(Xinp[i]-Xs2)**2+(Yinp[j]-Ys2)**2))) - math.atan
2((Xinp[i]-Xs2)*(Yinp[j]-Ys2),((Zinp[i,j]-Zs1)*math.sqrt((Zinp[i,j]-Zs1)**2+(X
inp[i]-Xs2)**2+(Yinp[j]-Ys2)**2))) -  math.atan2((Xinp[i]-Xs1)*(Yinp[j]-Ys2),
((Zinp[i,j]-Zs2)*math.sqrt((Zinp[i,j]-Zs2)**2+(Xinp[i]-Xs1)**2+(Yinp[j]-Ys2)**
2))) + math.atan2((Xinp[i]-Xs1)*(Yinp[j]-Ys2),((Zinp[i,j]-Zs1)*math.sqrt((Zinp
```

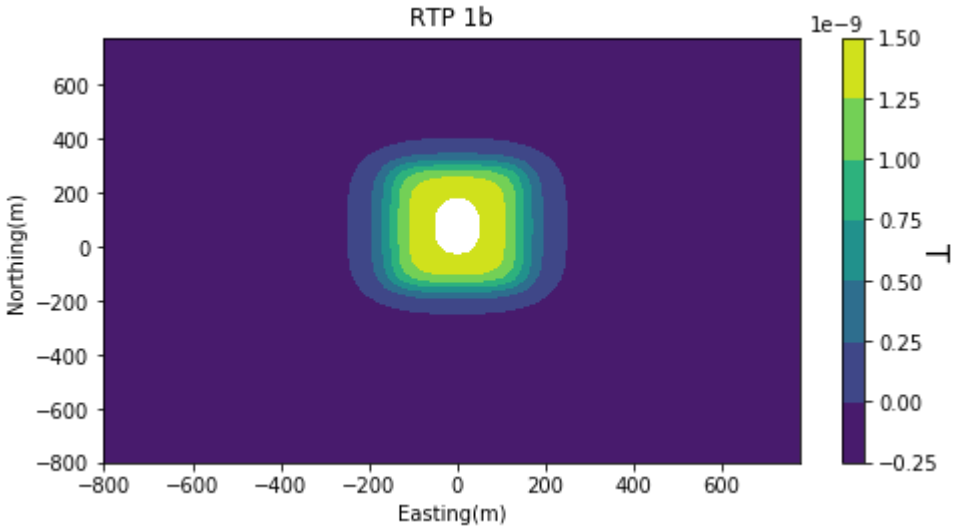
```
[i,j]-Zs1)**2+(Xinp[i]-Xs1)**2+(Yinp[j]-Ys2)**2))) - math.atan2((Xinp[i]-Xs2)*
(Yinp[j]-Ys1),((Zinp[i,j]-Zs2)*math.sqrt((Zinp[i,j]-Zs2)**2+(Xinp[i]-Xs2)**2+(
Yinp[j]-Ys1)**2))) + math.atan2((Xinp[i]-Xs2)*(Yinp[j]-Ys1),((Zinp[i,j]-Zs1)*m
ath.sqrt((Zinp[i,j]-Zs1)**2+(Xinp[i]-Xs2)**2+(Yinp[j]-Ys1)**2)))  + math.atan2
((Xinp[i]-Xs1)*(Yinp[j]-Ys1),((Zinp[i,j]-Zs2)*math.sqrt((Zinp[i,j]-Zs2)**2+(Xi
np[i]-Xs1)**2+(Yinp[j]-Ys1)**2))) - math.atan2((Xinp[i]-Xs1)*(Yinp[j]-Ys1),((Z
inp[i,j]-Zs1)*math.sqrt((Zinp[i,j]-Zs1)**2+(Xinp[i]-Xs1)**2+(Yinp[j]-Ys1)**2
)))
            Txy[i,j] = -math.log(math.sqrt((Xinp[i]-Xs2)**2+(Yinp[j]-Ys2)**2+(
Zinp[i,j]-Zs2)**2)-(Zinp[i,j]-Zs2))+math.log(math.sqrt((Xinp[i]-Xs1)**2+(Yinp[
j]-Ys2)**2+(Zinp[i,j]-Zs2)**2)-(Zinp[i,j]-Zs2))+math.log(math.sqrt((Xinp[i]-Xs
2)**2+(Yinp[j]-Ys1)**2+(Zinp[i,j]-Zs2)**2)-(Zinp[i,j]-Zs2))-math.log(math.sqrt
((Xinp[i]-Xs1)**2+(Yinp[j]-Ys1)**2+(Zinp[i,j]-Zs2)**2)-(Zinp[i,j]-Zs2))+math.l
og(math.sqrt((Xinp[i]-Xs2)**2+(Yinp[j]-Ys2)**2+(Zinp[i,j]-Zs1)**2)-(Zinp[i,j]-
Zs1))-math.log(math.sqrt((Xinp[i]-Xs1)**2+(Yinp[j]-Ys2)**2+(Zinp[i,j]-Zs1)**2)
-(Zinp[i,j]-Zs1))-math.log(math.sqrt((Xinp[i]-Xs2)**2+(Yinp[j]-Ys1)**2+(Zinp[i
,j]-Zs1)**2)-(Zinp[i,j]-Zs1))+math.log(math.sqrt((Xinp[i]-Xs1)**2+(Yinp[j]-Ys1
)**2+(Zinp[i,j]-Zs1)**2)-(Zinp[i,j]-Zs1))
            Txz[i,j] = -math.log(math.sqrt((Xinp[i]-Xs2)**2+(Zinp[i,j]-Zs2)**2
+(Yinp[j]-Ys2)**2)-(Yinp[j]-Ys2))+math.log(math.sqrt((Xinp[i]-Xs1)**2+(Zinp[i,
j]-Zs2)**2+(Yinp[j]-Ys2)**2)-(Yinp[j]-Ys2))+math.log(math.sqrt((Xinp[i]-Xs2)**
2+(Zinp[i,j]-Zs1)**2+(Yinp[j]-Ys2)**2)-(Yinp[j]-Ys2))-math.log(math.sqrt((Xinp
[i]-Xs1)**2+(Zinp[i,j]-Zs1)**2+(Yinp[j]-Ys2)**2)-(Yinp[j]-Ys2))+math.log(math.
sqrt((Xinp[i]-Xs2)**2+(Zinp[i,j]-Zs2)**2+(Yinp[j]-Ys1)**2)-(Yinp[j]-Ys1))-math
.log(math.sqrt((Xinp[i]-Xs1)**2+(Zinp[i,j]-Zs2)**2+(Yinp[j]-Ys1)**2)-(Yinp[j]-
Ys1))-math.log(math.sqrt((Xinp[i]-Xs2)**2+(Zinp[i,j]-Zs1)**2+(Yinp[j]-Ys1)**2)
-(Yinp[j]-Ys1))+math.log(math.sqrt((Xinp[i]-Xs1)**2+(Zinp[i,j]-Zs1)**2+(Yinp[j
]-Ys1)**2)-(Yinp[j]-Ys1))
            Tyz[i,j] = -math.log(math.sqrt((Yinp[j]-Ys2)**2+(Zinp[i,j]-Zs2)**2
+(Xinp[i]-Xs2)**2)-(Xinp[i]-Xs2))+math.log(math.sqrt((Yinp[j]-Ys1)**2+(Zinp[i,
j]-Zs2)**2+(Xinp[i]-Xs2)**2)-(Xinp[i]-Xs2))+math.log(math.sqrt((Yinp[j]-Ys2)**
2+(Zinp[i,j]-Zs1)**2+(Xinp[i]-Xs2)**2)-(Xinp[i]-Xs2))-math.log(math.sqrt((Yinp
[j]-Ys1)**2+(Zinp[i,j]-Zs1)**2+(Xinp[i]-Xs2)**2)-(Xinp[i]-Xs2))+math.log(math.
sqrt((Yinp[j]-Ys2)**2+(Zinp[i,j]-Zs2)**2+(Xinp[i]-Xs1)**2)-(Xinp[i]-Xs1))-math
.log(math.sqrt((Yinp[j]-Ys1)**2+(Zinp[i,j]-Zs2)**2+(Xinp[i]-Xs1)**2)-(Xinp[i]-
Xs1))-math.log(math.sqrt((Yinp[j]-Ys2)**2+(Zinp[i,j]-Zs1)**2+(Xinp[i]-Xs1)**2)
-(Xinp[i]-Xs1))+math.log(math.sqrt((Yinp[j]-Ys1)**2+(Zinp[i,j]-Zs1)**2+(Xinp[i
]-Xs1)**2)-(Xinp[i]-Xs1))

    Bax = np.zeros((len(Xinp), len(Yinp)))
    Bay = np.zeros((len(Xinp), len(Yinp)))
    Baz = np.zeros((len(Xinp), len(Yinp)))
    deltaT = np.zeros((len(Xinp), len(Yinp)))


    for i in range(len(Xinp)):
        for j in range(len(Yinp)):
            Bax[i,j] = (muo/(np.pi*4))*((Txx[i,j]*Mx) + (Txy[i,j]*My)+(Txz[i,j
]*Mz))/10**(-9)
            Bay[i,j] = (muo/(np.pi*4))*((Txy[i,j]*Mx) + (Tyy[i,j]*My)+(Tyz[i,j
]*Mz))/10**(-9)
            Baz[i,j] = (muo/(np.pi*4))*((Txz[i,j]*Mx) + (Tyz[i,j]*My)+(Tzz[i,j
]*Mz))/10**(-9)
            deltaT[i,j] = ((BoX*Bax[i,j] + BoY*Bay[i,j] + BoZ*Baz[i,j]))*10**(
-7)/50000
    return [Bax,Bay,Tzz,deltaT]
```

In [483]:
```
Zflat = np.zeros((64,64))
data1b = task2(Xinp,Yinp,Zflat,45,25)
```

In [484]:
```
fig = plt.figure()
ax = fig.gca(projection='3d')
X, Y = np.meshgrid(Xinp, Yinp)
surf = ax.plot_surface(X, Y, data1b[3] ,cmap=cm.magma,linewidth=0,antialiased=
False)
ax.set_xlabel('Northing (m)')
ax.set_ylabel('Easting (m)')
ax.set_zlabel('Depth (m)')
ax.set_title('Task 1b', fontsize=14)
plt.autoscale(enable=True, axis='x', tight=True)
plt.show()
```

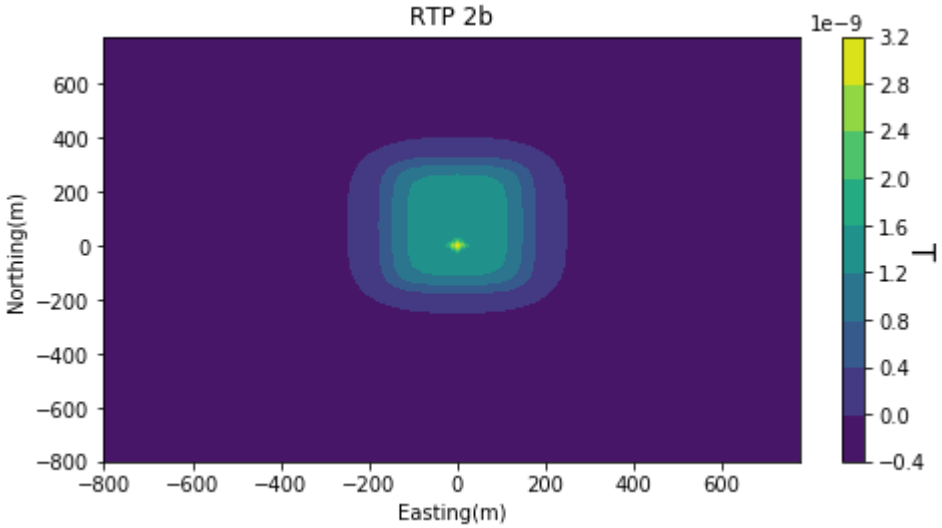


In [485]:
```
plt.figure(figsize=(7,7))
plt.subplot(211)
plt.title('RTP 1b')
plotting(data1b[3],Xinp,Yinp)
plt.tight_layout()
plt.show()
```



In [486]:
```
data2b = task2(Xinp,Yinp,Zinp,45,25)
```

In [487]:
```
plt.figure(figsize=(7,7))
plt.subplot(211)
plt.title('RTP 2b')
plotting(data2b[3],Xinp,Yinp)
plt.tight_layout()
plt.show()
```



Apply the wavenumber domain RTP operation to the two data sets (set-1a and set-2b) to reduce them to the pole. We will call the corresponding results as set-1c and set-2c, respectively.

In [488]:
```
##### Folding Functions ###########################################################
########

I=90
D=0
BoX = np.cos(I*np.pi/180)*np.cos(D*np.pi/180)
BoY = np.cos(I*np.pi/180)*np.sin(D*np.pi/180)
BoZ = np.sin(I*np.pi/180)


# Northing and Easting
Xinp = np.arange(-800,800,25)
Yinp = np.arange(-800,800,25)
h = 150
a = 200

#Elevation
Zinp=np.zeros((len(Xinp),len(Yinp)))
for i in range(len(Xinp)):
    for j in range(len(Yinp)):
        Zinp[i,j] = h*np.exp(-(Xinp[i]**2+Yinp[j]**2)/a)


def foldonce(A,n):
    Ashift = np.zeros(n)
    shift = 1
    for i in range(n):
        temp = A[0]
```

```
        for j in range(n):
            if j == n-1:
                Ashift[n-1] = temp
            else:
                Ashift[j] = A[j+shift]
    return Ashift

def foldonce2D(A, n):
    Ashift = np.zeros((n,n))
    Ashiftshift = np.zeros((n,n))
    shift = 1
    for i in range(n):
        temp = A[i,0]

        for j in range(n):
            if j == n-1:
                Ashift[i,n-1] = temp
            else:
                Ashift[i,j] = A[i, j+shift]

    for i in range(n):
        temprow = Ashift[0,:]
        if i == n-1:
            Ashiftshift[n-1,:] = temprow
        else:
            Ashiftshift[i,:] = Ashift[i+shift,:]
    return Ashiftshift

def unfold2Donce(A, n):
    Ashift = np.zeros((n,n))
    Ashiftshift = np.zeros((n,n))
    shift = 1
    for i in range(n):
        temp = A[i,n-1]
        for j in range(n):
            if j == 0:
                Ashift[i,0] = temp
            else:
                Ashift[i,j] = A[i, j-shift]

    for i in range(n):
        temprow = Ashift[n-1,:]

        if i == 0:
            Ashiftshift[0,:] = temprow
        else:
            Ashiftshift[i,:] = Ashift[i-shift,:]

    return Ashiftshift

######## Unfolding Functions ##################################################
############

def unfold(s,C3):
    for i in range(int(s-1)):
        ufftR3 = unfold2Donce(C3, n)
```

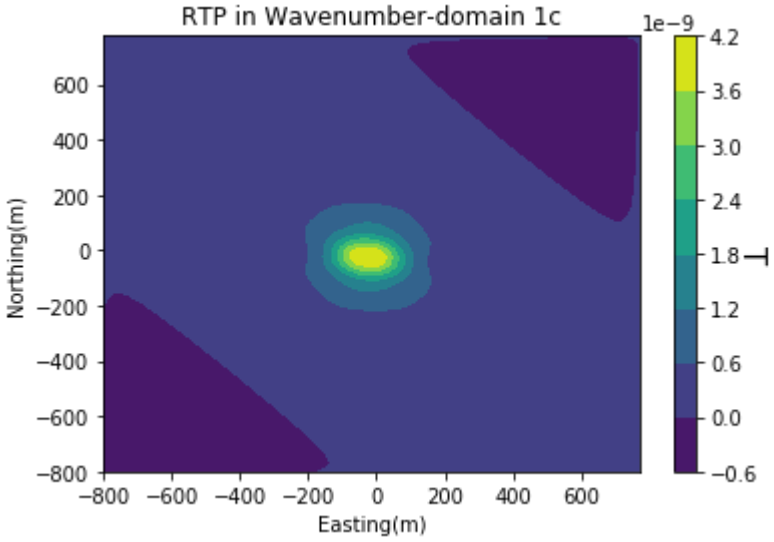
```
        C3 = ufftR3
    return ufftR3
```

In [489]:
```
## Find wx wy
M = int(len(Xinp)+1)
N = int(len(Yinp)+1)
wx = np.zeros((N,N))
wy = np.zeros((M,M))

kx = int(M/2)
ky = int(N/2)

dx = abs(Xinp[0]-Xinp[1])
dy = abs(Yinp[0]-Yinp[1])

m = np.linspace(-kx+1, kx, (M))
n = np.linspace(-ky+1, ky, (N))

wxm = m*2*np.pi/(M*dx)
for i in range(M):
    wx[i,:] = wxm


wym = n*2*np.pi/(N*dy)
for j in range(N):
    wy[j,:] = wym

for i in range(N):
    for j in range(M):
        if wx[i,j] ==0:
            wx[i,j] = 0.0000001
        if wy[i,j]==0:
            wy[i,j] = 0.0000001
```

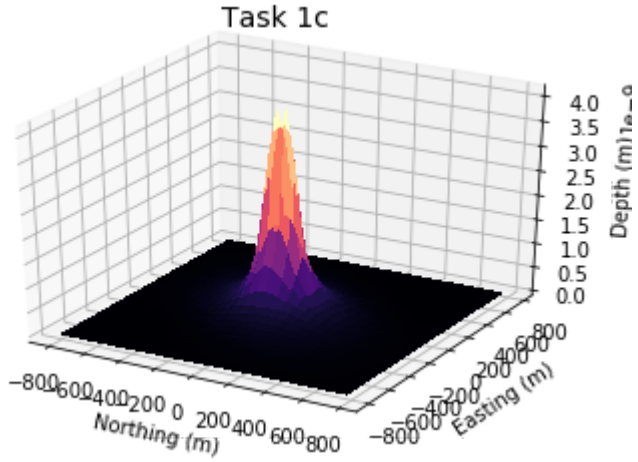
In [490]:
```
### folding
muo = 4*np.pi*10**(-7)
s = (M/2) + 1 - 1
n = 64

A3 = data1b[2]

for i in range(int(s-1)):
    foldedTzz = foldonce2D(A3, n)
    A3 = foldedTzz

###### Perform FFT ##################################################################
############
fftTzz = np.zeros((64,64))
fftTzz = np.fft.fft2(foldedTzz)

###### Unfolding ###################################################################
#############

unfoldTzz = np.zeros((64,64))
for i in range(int(s-1)):
    unfoldTzz = unfold2Donce(A3, n)
```

```
    A3 = unfoldTzz

##### Multiply FT by filter operator ########################################
###########
kB=np.zeros((64,64), dtype=np.complex_)
for i in range(64):
    for j in range(64):
        kB[i,j] = ((1j*wx[i,j]*BoX+1j*wy[j,i]*BoY+math.sqrt(wx[i,j]**2+ wy[j,i
]**2)*BoZ)*50000)
rtp = np.zeros((64,64), dtype=np.complex_)
for i in range(64):
    for j in range(64):
        rtp[i,j] = (math.sqrt(wx[i,j]**2+ wy[j,i]**2)/(kB[i,j]**2))*unfoldTzz[
i,j]

#fold
s = (M/2) + 1 - 1
for i in range(int(s-1)):
    rtpf = foldonce2D(rtp, n)
    rtp = rtpf

######## Inverse FFT ###########################################################
######
rtpfi= np.fft.ifft2(rtpf)

###### Unfolding ###############################################################
#############

for i in range(int(s-1)):
    rtp = unfold2Donce(rtpfi, n)
    rtpfi = rtp
################### T not nT !!!!!!!!!!!!!!!!!!!
data1c = rtp
plt.subplot()
plotting(rtp,Xinp,Yinp)
plt.title('RTP in Wavenumber-domain 1c')
plt.show()
```

```
C:\Users\nadim\Miniconda3\lib\site-packages\ipykernel_launcher.py:46: Complex
Warning: Casting complex values to real discards the imaginary part
C:\Users\nadim\Miniconda3\lib\site-packages\ipykernel_launcher.py:44: Complex
Warning: Casting complex values to real discards the imaginary part
C:\Users\nadim\Miniconda3\lib\site-packages\ipykernel_launcher.py:64: Complex
Warning: Casting complex values to real discards the imaginary part
C:\Users\nadim\Miniconda3\lib\site-packages\ipykernel_launcher.py:66: Complex
Warning: Casting complex values to real discards the imaginary part
```



In [491]:
```
fig = plt.figure()
ax = fig.gca(projection='3d')
X, Y = np.meshgrid(Xinp, Yinp)
surf = ax.plot_surface(X, Y, rtp,cmap=cm.magma,linewidth=0,antialiased=False)
ax.set_xlabel('Northing (m)')
ax.set_ylabel('Easting (m)')
ax.set_zlabel('Depth (m)')
ax.set_title('Task 1c', fontsize=14)
plt.autoscale(enable=True, axis='x', tight=True)
plt.show()
```

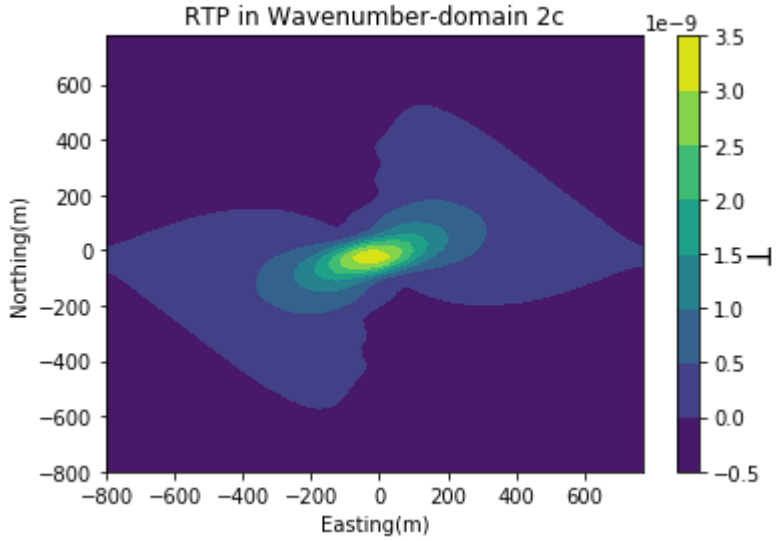


In [492]:
```
### folding

I=45
D=25
```

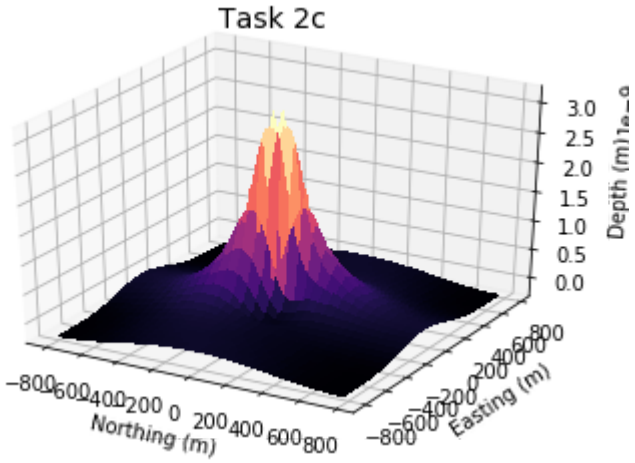
```
BoX = np.cos(I*np.pi/180)*np.cos(D*np.pi/180)
BoY = np.cos(I*np.pi/180)*np.sin(D*np.pi/180)
BoZ = np.sin(I*np.pi/180)

muo = 4*np.pi*10**(-7)
s = (M/2) + 1 - 1
n = 64

A3 = data2b[2]

for i in range(int(s-1)):
    foldedTzz = foldonce2D(A3, n)
    A3 = foldedTzz

###### Perform FFT ##################################################################
###########
fftTzz = np.zeros((64,64),dtype=np.complex_)
fftTzz = np.fft.fft2(foldedTzz)

###### Unfolding ####################################################################
############

unfoldTzz = np.zeros((64,64),dtype=np.complex_)
for i in range(int(s-1)):
    unfoldTzz = unfold2Donce(A3, n)
    A3 = unfoldTzz

##### Multiply FT by filter operator ###############################################
###########
kB=np.zeros((64,64),dtype=np.complex_)
for i in range(64):
    for j in range(64):
        kB[i,j] = (1j*wx[i,j]*BoX+1j*wx[j,i]*BoY+np.sqrt(wx[i,j]**2+ wy[j,i]**
2)*BoZ)*50000
rtp = np.zeros((64,64),dtype=np.complex_)
for i in range(64):
    for j in range(64):
        rtp[i,j] = (np.sqrt(wx[i,j]**2+ wy[j,i]**2)/(kB[i,j]**2))*unfoldTzz[i,
j]

#fold
s = (M/2) + 1 - 1
for i in range(int(s-1)):
    rtpf = foldonce2D(rtp, n)
    rtp = rtpf

######## Inverse FFT ################################################################
######
rtpfi= np.fft.ifft2(rtpf)

###### Unfolding ####################################################################
############

for i in range(int(s-1)):
    rtp = unfold2Donce(rtpfi, n)
    rtpfi = rtp
################### T not nT !!!!!!!!!!!!!!!!!!!
```

```
data2c = rtp
plt.subplot()
plotting(rtp,Xinp,Yinp)
plt.title('RTP in Wavenumber-domain 2c')
plt.show()
```

```
C:\Users\nadim\Miniconda3\lib\site-packages\ipykernel_launcher.py:46: Complex
Warning: Casting complex values to real discards the imaginary part
C:\Users\nadim\Miniconda3\lib\site-packages\ipykernel_launcher.py:44: Complex
Warning: Casting complex values to real discards the imaginary part
C:\Users\nadim\Miniconda3\lib\site-packages\ipykernel_launcher.py:64: Complex
Warning: Casting complex values to real discards the imaginary part
C:\Users\nadim\Miniconda3\lib\site-packages\ipykernel_launcher.py:66: Complex
Warning: Casting complex values to real discards the imaginary part
```



In [493]:
```
fig = plt.figure()
ax = fig.gca(projection='3d')
X, Y = np.meshgrid(Xinp, Yinp)
surf = ax.plot_surface(X, Y, rtp,cmap=cm.magma,linewidth=0,antialiased=False)
ax.set_xlabel('Northing (m)')
ax.set_ylabel('Easting (m)')
ax.set_zlabel('Depth (m)')
ax.set_title('Task 2c', fontsize=14)
plt.autoscale(enable=True, axis='x', tight=True)
plt.show()

print(rtp[0,0])
```

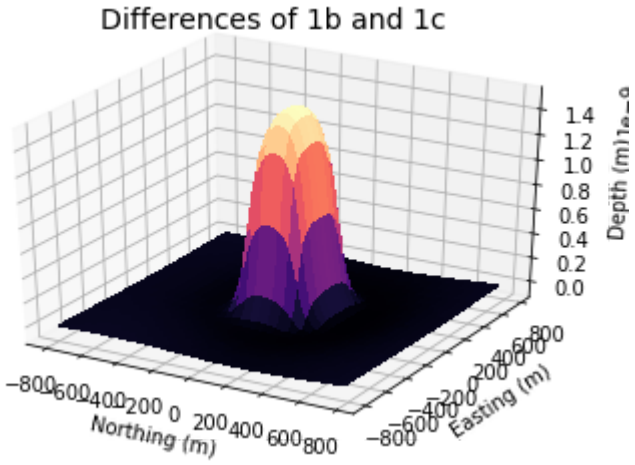



```
-2.63472221151e-10
```

Plot the data set-1c and set-2c with the corresponding true RTP fields (set-1b and set-2b) and the associated differences. Observe the magnitude and pattern in the two different maps and comment on the errors that are produced by the violation of the planar surface observation.

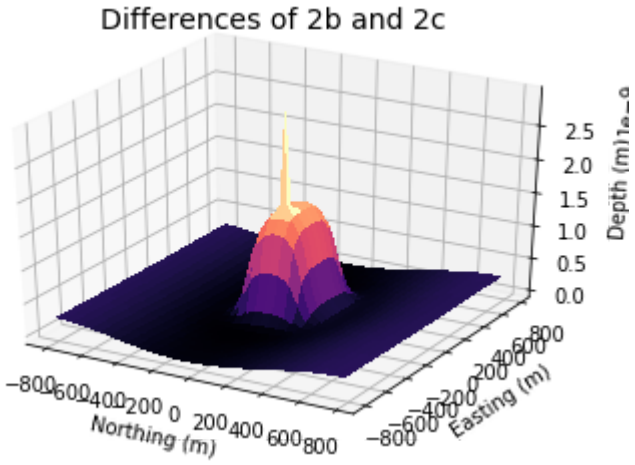
In [494]:
```
fig = plt.figure()
ax = fig.gca(projection='3d')
X, Y = np.meshgrid(Xinp, Yinp)
surf = ax.plot_surface(X, Y, (data1b[3]-data1c[3]),cmap=cm.magma,linewidth=0,a
ntialiased=False)
ax.set_xlabel('Northing (m)')
ax.set_ylabel('Easting (m)')
ax.set_zlabel('Depth (m)')
ax.set_title('Differences of 1b and 1c', fontsize=14)
plt.autoscale(enable=True, axis='x', tight=True)
print(data1b[3][0,0]-data1c[3][0])
plt.show()
```

```
-7.42397343467e-12
```



In [495]:
```
fig = plt.figure()
ax = fig.gca(projection='3d')
X, Y = np.meshgrid(Xinp, Yinp)
surf = ax.plot_surface(X, Y, (data2b[3]-data2c[3]),cmap=cm.magma,linewidth=0,a
ntialiased=False)
ax.set_xlabel('Northing (m)')
ax.set_ylabel('Easting (m)')
ax.set_zlabel('Depth (m)')
ax.set_title('Differences of 2b and 2c', fontsize=14)
plt.autoscale(enable=True, axis='x', tight=True)
print(data2b[3][0,0]-data2c[3][0])
plt.show()
```

```
2.534989949e-10
```



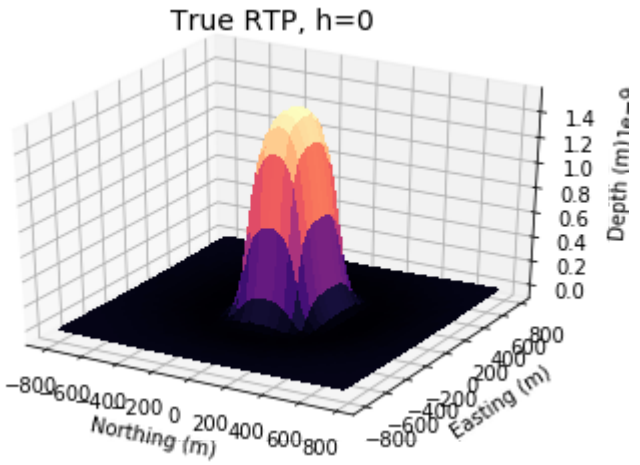
Task 2 - Observation and Conclusion

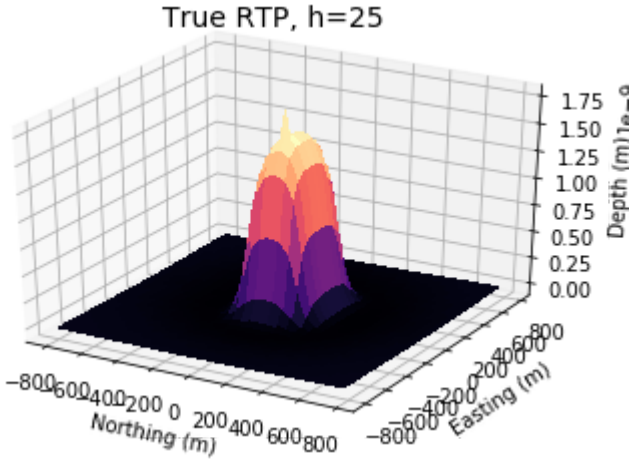
From the plot above, we see that having a constant Z will give us a uniform, smooth gaussian hill. This is different from the non-constant Z plot, where the gaussian hill looks less uniform and rougher. Calculating the difference between the true RTP and the wavenumber RTP, we can see that the difference in the constant Z scenario has less error than the non-constant Z scenario. This is expected because the error caused by the discrete nature of the Fourier Transform is amplified with a non-constant Z plane. Furthermore, looking at the two different scenarios in the 2D plot, we can see that the RTP anomaly with constant Z is perfectly on top of our object. Meanwhile, we can see that the RTP anomaly with non-constant Z is more skewed and is angled towards the inclination of the magnetic field.

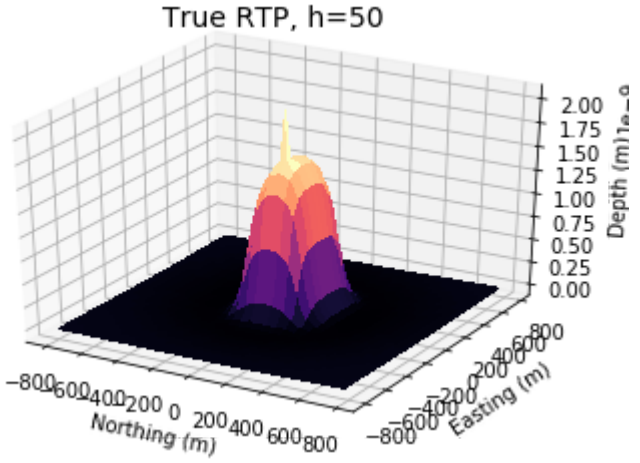
Task 3

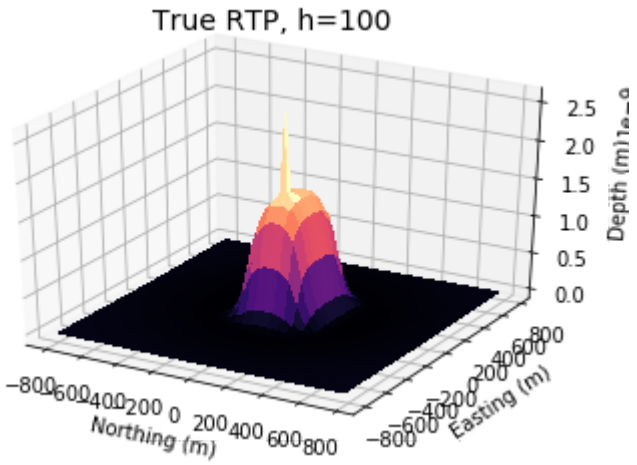
Focus on the Gaussian-hill observation surface, investigate the change in error levels for a sequence of hill height h=0,25,50,100,200,400 m by computing the L2 norm of the differences between two and computed RTP data and plot as a function of the height.

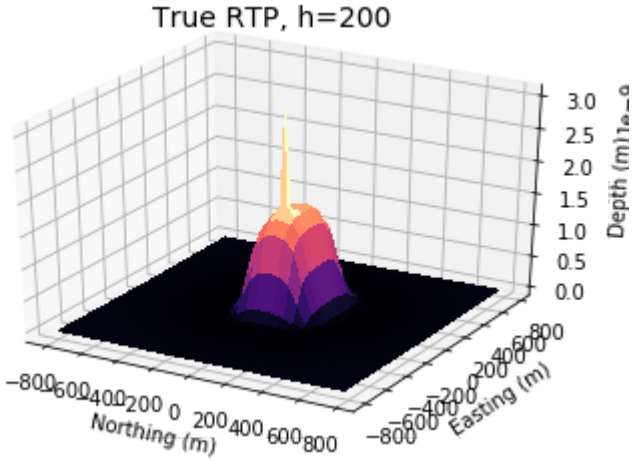
In [496]:
```
# L2 (square root of sum of the difference squared)
hmat = [0,25,50,100,200,400]
for h in hmat:
    Zinp = np.zeros((64,64))
    for i in range(len(Xinp)):
        for j in range(len(Yinp)):
            Zinp[i,j] = h*np.exp(-(Xinp[i]**2+Yinp[j]**2)/a)
    data2b = task2(Xinp,Yinp,Zinp,45,25)

    fig = plt.figure()
    ax = fig.gca(projection='3d')
    X, Y = np.meshgrid(Xinp, Yinp)
    surf = ax.plot_surface(X, Y, data2b[3] ,cmap=cm.magma,linewidth=0,antialia
sed=False)
    ax.set_xlabel('Northing (m)')
    ax.set_ylabel('Easting (m)')
    ax.set_zlabel('Depth (m)')
    ax.set_title('True RTP, h=%i'%h, fontsize=14)
    plt.autoscale(enable=True, axis='x', tight=True)
    plt.show()
```

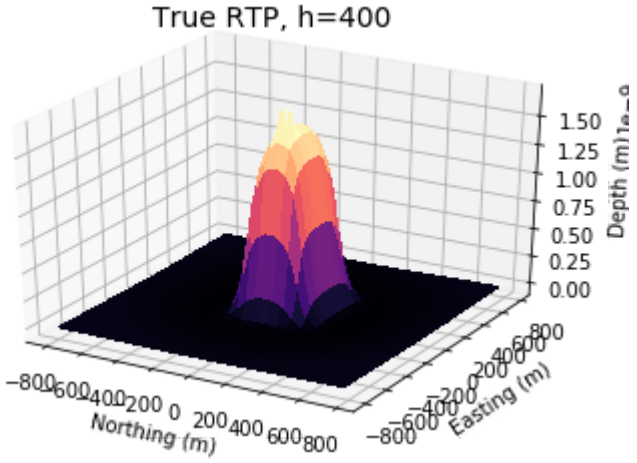

True RTP, h=0
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Depth (m)
−800 −600 −400 −200 0 200 400 600 800
Northing (m)
−800 −600 −400 −200 0 200 400 600 800
Easting (m)


True RTP, h=25
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Depth (m)
−800 −600 −400 −200 0 200 400 600 800
Northing (m)
−800 −600 −400 −200 0 200 400 600 800
Easting (m)


True RTP, h=50
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Depth (m)
−800 −600 −400 −200 0 200 400 600 800
Northing (m)
−800 −600 −400 −200 0 200 400 600 800
Easting (m)


True RTP, h=100
2.5
2.0
1.5
1.0
0.5
0.0
Depth (m)
−800 −600 −400 −200 0 200 400 600 800
Northing (m)
−800 −600 −400 −200 0 200 400 600 800
Easting (m)


True RTP, h=200
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Depth (m)
−800 −600 −400 −200 0 200 400 600 800
Northing (m)
−800 −600 −400 −200 0 200 400 600 800
Easting (m)


True RTP, h=400
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Depth (m)
−800 −600 −400 −200 0 200 400 600 800
Northing (m)
−800 −600 −400 −200 0 200 400 600 800
Easting (m)

In [497]:
```
# L2 (square root of sum of the difference squared)


def task3(h):
    s=32
    Zinp = np.zeros((64,64))
    for i in range(len(Xinp)):
        for j in range(len(Yinp)):
            Zinp[i,j] = h*np.exp(-(Xinp[i]**2+Yinp[j]**2)/a)
    data2b = task2(Xinp,Yinp,Zinp,45,25)

    A3 = data2b[2]
```

```
    for i in range(int(s-1)):
        foldedTzz = foldonce2D(A3, n)
        A3 = foldedTzz

###### Perform FFT ##################################################################
############
    fftTzz = np.zeros((64,64))
    fftTzz = np.fft.fft2(foldedTzz)

###### Unfolding ####################################################################
#############

    unfoldTzz = np.zeros((64,64))
    for i in range(int(s-1)):
        unfoldTzz = unfold2Donce(A3, n)
        A3 = unfoldTzz

##### Multiply FT by filter operator ################################################
############
    kB=np.zeros((64,64), dtype=np.complex_)
    for i in range(64):
        for j in range(64):
            kB[i,j] = ((1j*wx[i,j]*BoX+1j*wy[j,i]*BoY+math.sqrt(wx[i,j]**2+ wy
[j,i]**2)*BoZ)*50000)
    rtp = np.zeros((64,64), dtype=np.complex_)
    for i in range(64):
        for j in range(64):
            rtp[i,j] = (math.sqrt(wx[i,j]**2+ wy[j,i]**2)/(kB[i,j]**2))*unfold
Tzz[i,j]

#fold
    #s = (M/2) + 1 - 1
    for i in range(int(s-1)):
        rtpf = foldonce2D(rtp, n)
        rtp = rtpf

######## Inverse FFT ################################################################
######
    rtpfi= np.fft.ifft2(rtpf)

###### Unfolding ####################################################################
#############

    for i in range(int(s-1)):
        rtp = unfold2Donce(rtpfi, n)
        rtpfi = rtp
################### T not nT !!!!!!!!!!!!!!!!!!

    return(rtp)
def plottask3(X,Y,rtp,h):
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    X, Y = np.meshgrid(Xinp, Yinp)
    title=str(h)
    surf = ax.plot_surface(X, Y, rtp,cmap=cm.magma,linewidth=0,antialiased=Fal
se)
```

```
    ax.set_xlabel('Northing (m)')
    ax.set_ylabel('Easting (m)')
    ax.set_zlabel('Depth (m)')
    ax.set_title('Wavenumber RTP, h=' + title, fontsize=14)
    plt.autoscale(enable=True, axis='x', tight=True)
    plt.show()


plottask3(X,Y,task3(0),0)
plottask3(X,Y,task3(25),25)
plottask3(X,Y,task3(50),50)
plottask3(X,Y,task3(100),100)
plottask3(X,Y,task3(200),200)
plottask3(X,Y,task3(400),400)

print(rtp[0,0])
```

```
C:\Users\nadim\Miniconda3\lib\site-packages\ipykernel_launcher.py:46: Complex
Warning: Casting complex values to real discards the imaginary part
C:\Users\nadim\Miniconda3\lib\site-packages\ipykernel_launcher.py:44: Complex
Warning: Casting complex values to real discards the imaginary part
C:\Users\nadim\Miniconda3\lib\site-packages\ipykernel_launcher.py:64: Complex
Warning: Casting complex values to real discards the imaginary part
C:\Users\nadim\Miniconda3\lib\site-packages\ipykernel_launcher.py:66: Complex
Warning: Casting complex values to real discards the imaginary part
```