Case Study: Interpolating the geoid height/undulation of a ground location or a space object from the world map of the geoid heights relate to World Geodetic System 1984 north/south latitudes and East/west longitudes

Prepared/Compiled/Edited by: XYZ

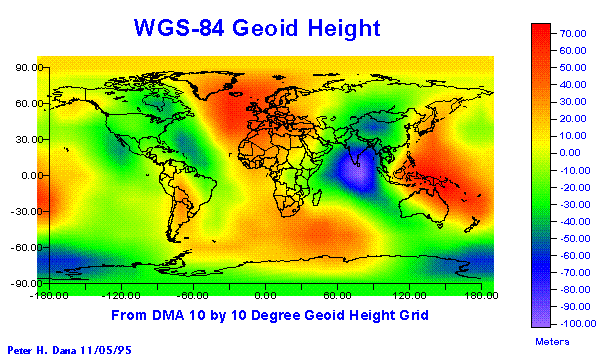
References:

1. Wikipedia.com

Problem

Interpolating a geoid height of a known point associated with the latitudes () and longitude () from the given the World Geodetic System 1984 (WGS 84) geoid height in meters, the is a discrete function of and in degrees and measured from the WGS 84 geoid or mean sea level surface to the ellipsoid surface (Fig 2)

# Data Sources



**Latitude**

**Fig. 1 Longitude**

Table 1 contains entries of the World Geodetic System 1984 (WGS 84) geoid heights as a lookup table based on the geodetic latitudes and longitudes

# Table 1. WGS- 84 Geoid Heights versus Geodetic Latitudes from +900 to -900 wrt Longitudes from -1800 to 1700

13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13

3,1,-2,-3,-3,-3,-1,3,1,5,9,11,19,27,31,34,33,34,33,34,28,23,17,13,9,4,4,1,-2,-2,0,2,3,2,1,1

2,2,1,-1,-3,-7,-14,-24,-27,-25,-19,3,24,37,47,60,61,58,51,43,29,20,12,5,-2,-10,-14,-12,-10,-14,-12,-6,-2,3,6,4

2,9,17,10,13,1,-14,-30,-39,-46,-42,-21,6,29,49,65,60,57,47,41,21,18,14,7,-3,-22,-29,-32,-32,-26,-15,-2,13,17,19,6

-8,8,8,1,-11,-19,-16,-18,-22,-35,-40,-26,-12,24,45,63,62,59,47,48,42,28,12,-10,-19,-33,-43,-42,-43,-29,-2,17,23,22,6,2

-12,-10,-13,-20,-31,-34,-21,-16,-26,-34,-33,-35,-26,2,33,59,52,51,52,48,35,40,33,-9,-28,-39,-48,-59,-50,-28,3,23,37,18,-1,-11

-7,-5,-8,-15,-28,-40,-42,-29,-22,-26,-32,-51,-40,-17,17,31,34,44,36,28,29,17,12,-20,-15,-40,-33,-34,-34,-28,7,29,43,20,4,-6

5,10,7,-7,-23,-39,-47,-34,-9,-10,-20,-45,-48,-32,-9,17,25,31,31,26,15,6,1,-29,-44,-61,-67,-59,-36,-11,21,39,49,39,22,10

13,12,11,2,-11,-28,-38,-29,-10,3,1,-11,-41,-42,-16,3,17,33,22,23,2,-3,-7,-36,-59,-90,-95,-63,-24,12,53,60,58,46,36,26

22,16,17,13,1,-12,-23,-20,-14,-3,14,10,-15,-27,-18,3,12,20,18,12,-13,-9,-28,-49,-62,-89,-102,-63,-9,33,58,73,74,63,50,32

36,22,11,6,-1,-8,-10,-8,-11,-9,1,32,4,-18,-13,-9,4,14,12,13,-2,-14,-25,-32,-38,-60,-75,-63,-26,0,35,52,68,76,64,52

51,27,10,0,-9,-11,-5,-2,-3,-1,9,35,20,-5,-6,-5,0,13,17,23,21,8,-9,-10,-11,-20,-40,-47,-45,-25,5,23,45,58,57,63

46,22,5,-2,-8,-13,-10,-7,-4,1,9,32,16,4,-8,4,12,15,22,27,34,29,14,15,15,7,-9,-25,-37,-39,-23,-14,15,33,34,45

21,6,1,-7,-12,-12,-12,-10,-7,-1,8,23,15,-2,-6,6,21,24,18,26,31,33,39,41,30,24,13,-2,-20,-32,-33,-27,-14,-2,5,20

-15,-18,-18,-16,-17,-15,-10,-10,-8,-2,6,14,13,3,3,10,20,27,25,26,34,39,45,45,38,39,28,13,-1,-15,-22,-22,-18,-15,-14,-10

-45,-43,-37,-32,-30,-26,-23,-22,-16,-10,-2,10,20,20,21,24,22,17,16,19,25,30,35,35,33,30,27,10,-2,-14,-23,-30,-33,-29,-35,-43

-61,-60,-61,-55,-49,-44,-38,-31,-25,-16,-6,1,4,5,4,2,6,12,16,16,17,21,20,26,26,22,16,10,-1,-16,-29,-36,-46,-55,-54,-59

-53,-54,-55,-52,-48,-42,-38,-38,-29,-26,-26,-24,-23,-21,-19,-16,-12,-8,-4,-1,1,4,4,6,5,4,2,-6,-15,-24,-33,-40,-48,-50,-53,-52

-30,-30,-30,-30,-30,-30,-30,-30,-30,-30,-30,-30,-30,-30,-30,-30,-30,-30,-30,-30,-30,-30,-30,-30,-30,-30,-30,-30,-30,-30,-30,-30,-30,-30,-30,-30

With the Python program shown in Table 2, the above data are read into a 19x37 array matrix. The entries are the values of the geoid heights:

* Column numbers (C) from 0 to 36 starting -1800 to +1700 increment by 100
* Row numbers (R) from 0 to 18 starting +900 to -900 decrement by 100

Note: the entries of longitude +1800 is equal to longitude -1800 (Earth longitude wraps around). For convenience, column 37 for the longitude +1800 may be added. Then the matrix becomes a 19x38 array matrix.

# Table 2. Example of Table 1 format wrt Python Programming

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | | LONGITUDE | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|  |  | | C | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 |
|  |  |  | d | -180 | -170 | -160 | -150 | -140 | -130 | -120 | -110 | -100 | -90 | -80 | -70 | -60 | -50 | -40 | -30 | -20 | -10 | 0 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | 110 | 120 | 130 | 140 | 150 | 160 | 170 | 180 |
| L  A  T  I  T  U  DE | R | d |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0 | 90 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1 | 80 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 | 70 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 | 60 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 | 50 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5 | 40 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6 | 30 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 7 | 20 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 8 | 10 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 9 | 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 10 | -10 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 11 | -20 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 12 | -30 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 13 | -40 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 14 | -50 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 15 | -60 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 16 | -70 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 17 | -80 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 18 | -90 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

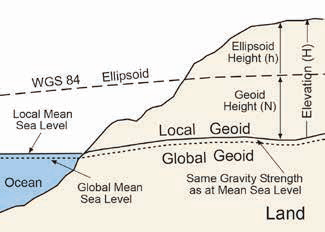
# **Vertical Datum (Height) Parameter Definitions**

The geoid height is a theoretical estimate based on the mathematical Earth model WGS 1984 to be defined as:

Geodetic or ellipsoid altitude (or height) from the WGS 84 ellipsoid model surface

Elevation or mean-sea level (MSL) altitude (or height) from the WGS 84 geoid model surface which approximates to the MSL surface. GPS cam provide the MSL altitude

Because the vertical altitude (vertical datum) of a static of dynamic target is very important information for a static or dynamic aircraft when takeoff off , landing or low-altitude fly to avoid accidents, the relative altitude(either or ) must be known. If a sensor such as the GPS provides a the MSL altitude H, then the geodetic altitude can be derived from



**Physical Earth Surface**

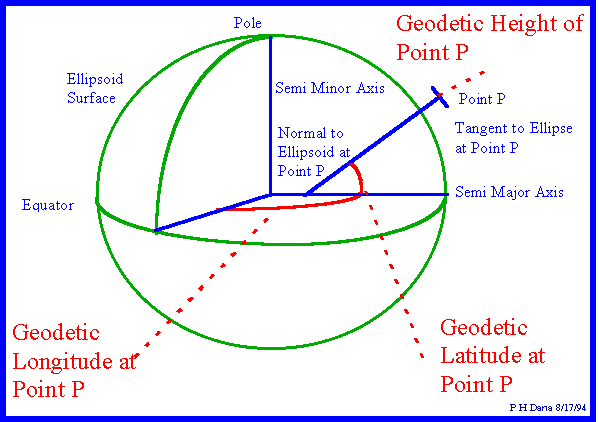


Fig 2 Relationships among ellipsoid, elevation (MSL), and geoid heights

* The latitude is in the range values from to with decrement such as
  + which is called the equator (equatorial plane) divides the Earth to the northern and southern hemispheres
  + “North Pole” , rotation axis of the Earth with counter clock wise (CCW) direction
  + “South Pole”
  + Positive : , it’s called the northern latitudes denoted as “N” ; for example , or
  + Negative : , it’s called the southern latitudes denoted as “S” ; for example , or
* Similarly, with respect to the order of the range, the longitude is in the range values from to with increment such as
  + which is called the “prime” or “Greenwich” meridian or longitude ) divides the Earth in to the eastern (right) and western (left) meridians/longitudes
  + Positive : , it’s called the eastern longitude denoted as “E” ; for example , or
  + Negative : , it’s called the southern latitudes denoted as “S” ; for example , or
  + Note:, Same geoid heights at
* If values are given in radians or degrees: minutes:seconds formats, must be converted to the decimal degrees for the interpolations.

# Algorithms

Refer to Wikipedia website for the bilinear interpolation for details and appendix A for the python program.

Interpolate a geoid height of an arbitrary location at the **latitude and longitude with an assumption that is located within** the rectangular with **four corner indexes** and **their geoid heights** have been determined :.

The bilinear-interpolation method is used [1]. In general, the four cases are considered for the interpolations

* Case 1 : and : Both and directions require linear interpolations
* Case 2 : and : Only direction requires linear interpolation
* Case 3 : and : Only direction requires linear interpolation
* Case 4 : and : No interpolations require on both and directions

# Testing Points

Case 1:

1. (N30.123456, W175.123456)= (+30.123456, -175.123456)
2. (N30.123456, E175.123456) = (+30.123456, +175.123456)
3. (S30.123456, W175.123456) = (-30.123456, -175.123456)
4. (S30.123456, E175.123456) = (-30.123456, +175.123456)

Case 2

1. (N30, W175.123456)= (+30, -175.123456)

Case 3

1. N30.123456, W170)= (+30.123456, -170)

Case 4

1. N30, W170)= (+30, -170)

Case 2

Case 4

N

N

S

E

W

Case 1

Case 3

# Case 1: and

The case 1 is a general case wrt the four red circle dots associate with their geoid heights  *(i ,j=1,2)*have been determined priory shown in Fig 3.

Latitude (North, N)

South (S)

West (W)

Longitude (East, E)

P, Case 1

,

,

,

,

,

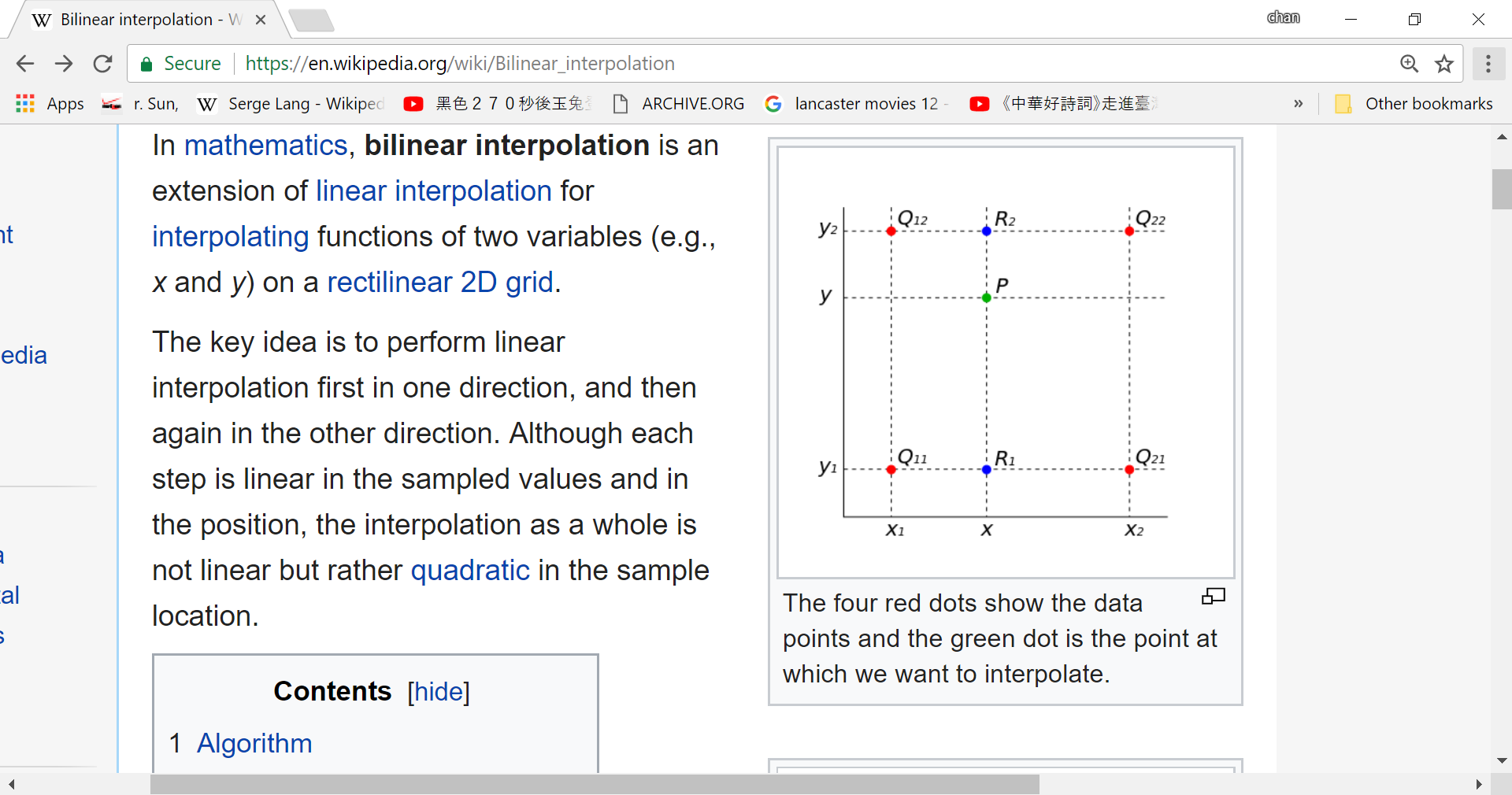


Fig 3. Cases 1 and nterpolations

I THINK YOU FORGOT A STEP? WE NEED TO INTERP IN THE Y DIRECTION AFTER DOING IT FOR X?

# Case 2: and :

The case 2 contains the two red circle dots associate with their geoid heights (*j=1,2)* have been determined shown in Fig 4. The interpolation requires on -axis only wrt given. From the above equation 3,

Where from Table 1 with the given **.**

P, Case 2

,

Latitude

(North, N)

South (S)

West (W)

Longitude (East, E)

,

,

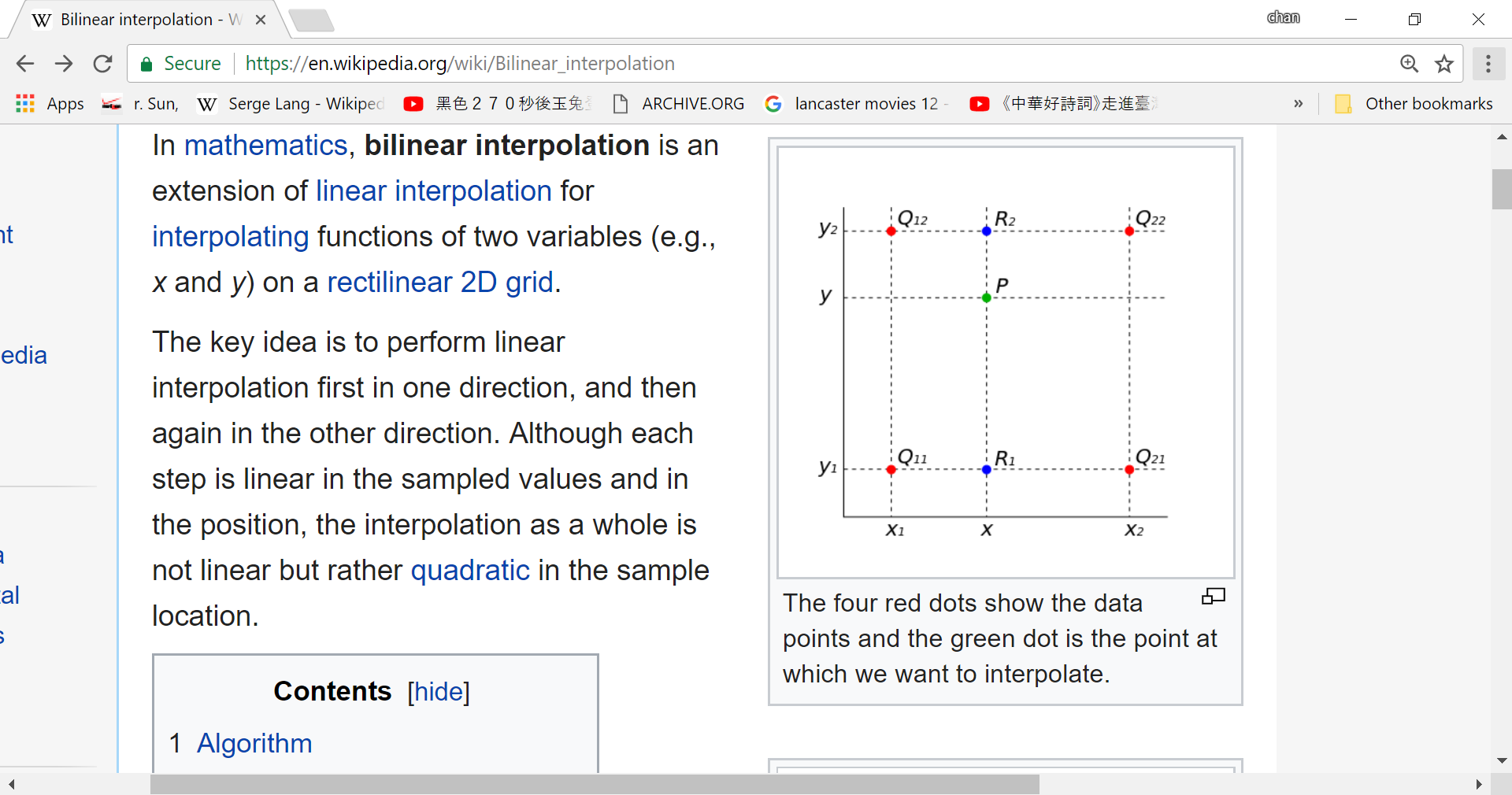


Fig 4. Cases 2: and nterpolations

# Case 3: and :

The case 3 contains the two red circle dots associate with their geoid heights  *(i =1,2)* have been determined shown in Fig 5. The interpolation requires on -axis only wrt given. From the above equation 3,

Where from Table 1 with the given **.**

,

P, Case 3

,

Latitude

(North, N)

South (S)

West (W)

Longitude (East, E)

,

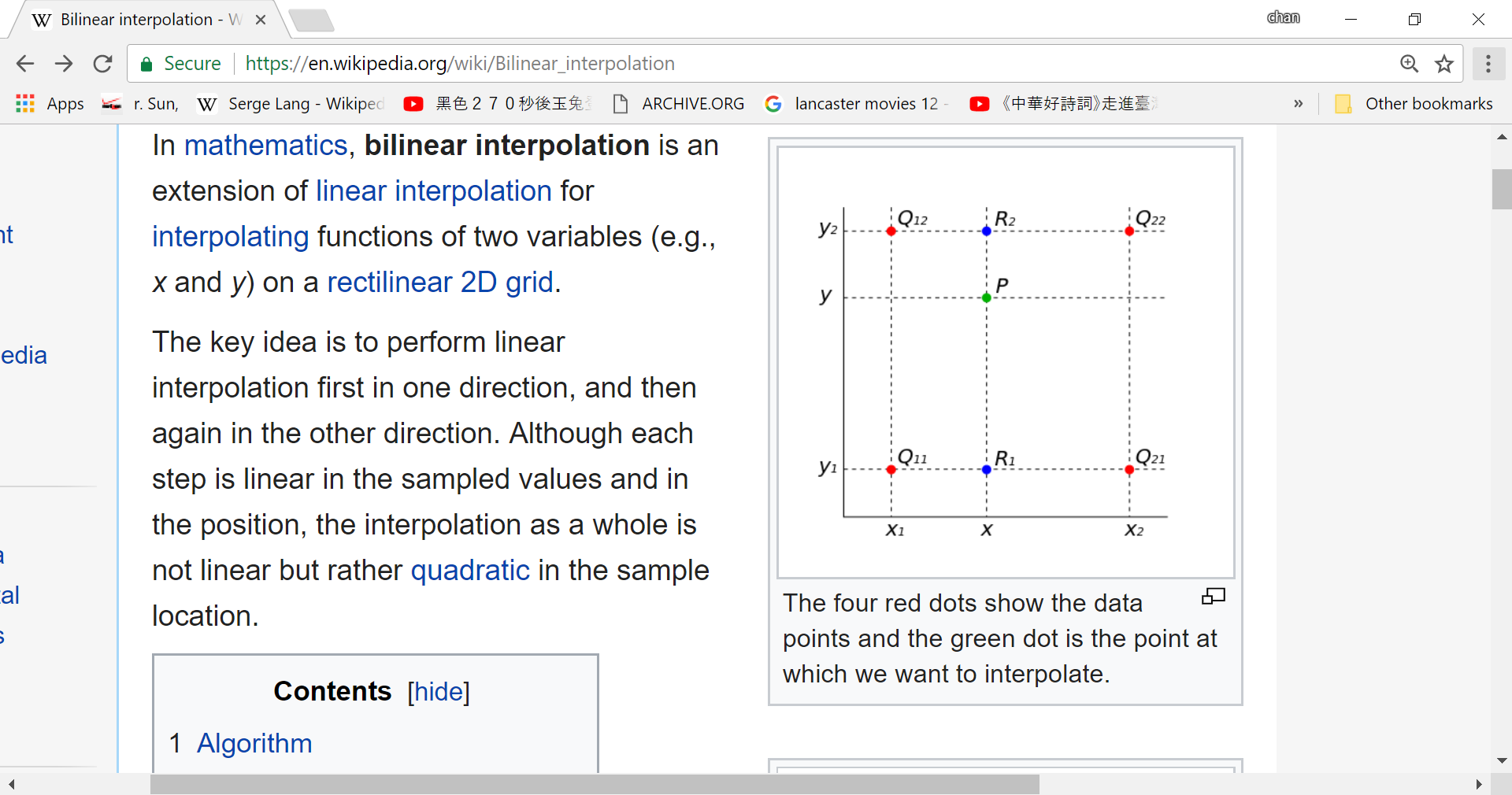


Fig 5. Cases3: and nterpolations

# Case 4: No interpolations are needed

Provide

# Appendix A: Python Programming Reference

<https://gis.stackexchange.com/questions/15801/interpolate-undulations-for-given-points-using-a-geoid-model>

import csv,sqlite3

'''this script inserts data stored in csv file into sqlite database'''

coordinates= csv.reader(open("geoid.csv"))

database = sqlite3.connect("geoid.db")

cursor = database.cursor()

database.execute("create table coords(name, lat INTEGER, lon INTEGER, und INTEGER);")

database.executemany("insert into coords(name,lat,lon,und) values (?, ?, ?, ?)",coordinates)

database.commit()

database.close()

Here's example input csv file, that contains ID,Latitude,Longitude,Undulation:  
000001,54.9666669,18.7,28.442  
000002,54.9166669,18.6833331,28.512  
000003,54.9,18.6833331,28.523  
000004,54.8833331,18.6666669,28.572  
000005,54.8666669,18.6666669,28.585  
000006,54.85,18.6666669,28.6  
000007,54.8333331,18.0666669,29.891  
000008,54.8333331,18.0833331,29.855  
000009,54.8333331,18.1833331,29.637  
000010,54.8333331,18.25,29.488  
000011,54.8333331,18.2666669,29.45  
000012,54.8333331,18.2833331,29.413  
000013,54.8333331,18.3,29.376  
This is numeric geoid model stored in ASCII csv file.

Now I would like to find four adjacent points for our example point:

import sqlite3

#find four adjacent points

def find\_adjacent\_coords(db, lat, lon, step=0.041667):

coords\_range = lat-step, lat+step, lon-step, lon+step

return db.execute("""select lat, lon, und from coords where

lat > ? and lat < ? and

lon > ? and lon < ?""", coords\_range).fetchall()

#connect database with gps points

database = sqlite3.connect('F.tsj')

cursor = database.cursor()

cursor.execute("select C1, C2, C3 from tblSoPoints")

#C1 is latitude

#C2 is longitude

#C3 is ellipsoid height

results = cursor.fetchall()

for line in results:

B = line[0]

L = line[1]

ellps\_height = line[2]

#connect to geoid database

db = sqlite3.connect('geoid.db')

n = (find\_adjacent\_coords(db, a, b))

Example gps point:

54.4786674627, 17.0470721369, 86.5003132338

and adjacent points from geoid:

[(54.5, 17.041667, 31.993), (54.5, 17.083333, 31.911), (54.458333, 17.041667, 31.945), (54.458333, 17.083333, 31.866)]<br>

Then I would like to interpolate undulation for our point using these four adjacent points with known undulations to calculate orthometric height.

Edit

Solution No 1

from \_\_future\_\_ import division

def bilinear\_interpolation(x, y, points):

points = sorted(points) # order points by x, then by y

(x1, y1, q11), (\_x1, y2, q12), (x2, \_y1, q21), (\_x2, \_y2, q22) = points

if x1 != \_x1 or x2 != \_x2 or y1 != \_y1 or y2 != \_y2:

raise ValueError('points do not form a square')

if not x1 <= x <= x2 or not y1 <= y <= y2:

raise ValueError('(x, y) not within the square')

return (q11 \* (x2 - x) \* (y2 - y) +

q21 \* (x - x1) \* (y2 - y) +

q12 \* (x2 - x) \* (y - y1) +

q22 \* (x - x1) \* (y - y1)

) / (x2 - x1) \* (y2 - y1)

n = [(54.5, 17.041667, 31.993),

(54.5, 17.083333, 31.911),

(54.458333, 17.041667, 31.945),

(54.458333, 17.083333, 31.866),

]

B = 54.4786674627

L = 17.0470721369

print bilinear\_interpolation(B, L, n)

import doctest

print doctest.testmod()

...and here's the solution No 2:

from \_\_future\_\_ import division

import doctest

def bilinear\_interpolation(x, y, points):

# http://en.wikipedia.org/wiki/Bilinear\_interpolation

points.sort() # order by x, then by y

xs, ys, zs = zip(\*points)

# verify it is a square

assert xs[0] == xs[1] and xs[2] == xs[3]

assert ys[0] == ys[2] and ys[1] == ys[3]

x1 = xs[0]

x2 = xs[2]

y1 = ys[0]

y2 = ys[1]

# verify interpolation versus extrapolation

assert x1 <= x <= x2, (x1, x, x2)

assert y1 <= y <= y2

q11, q12, q21, q22 = zs

return (q11 / ((x2 - x1) \* (y2 - y1)) \* (x2 - x) \* (y2 - y) +

q21 / ((x2 - x1) \* (y2 - y1)) \* (x - x1) \* (y2 - y) +

q12 / ((x2 - x1) \* (y2 - y1)) \* (x2 - x) \* (y - y1) +

q22 / ((x2 - x1) \* (y2 - y1)) \* (x - x1) \* (y - y1)

)

B = 54.5

L = 17.061667

n = [(54.5, 17.041667, 31.993),

(54.5, 17.083333, 31.911),

(54.458333, 17.041667, 31.945),

(54.458333, 17.083333, 31.866),

]

print doctest.testmod()

print bilinear\_interpolation(B, L, n)

You have two tasks: (a) find the grid data for the vertices of a grid cell in which a GPS location (x,y) lies and (b) interpolate those data.

An efficient way to accomplish (a) is to **store the grid as an array with a single index**. Often grids are stored by rows with data progressing left to right within each row. A grid is determined by the coordinates of its origin, (x0,y0) (which will be (-180, -90) for a worldwide grid in decimal degrees), its cellsize *c* (which is 1/24 in your case), and the number of points per row (*n*). The values would be recorded in the sequence

(x0,y0), (x0+c,y0), (x0+2c,y0), ..., (x0+(n-1)\*c,y0)

(x0,y0+c), (x0+c,y0+c), (x0+2c,y0+c), ..., (x0+(n-1)\*c,y0+c)

...

(x0,y0+(m-1)\*c), (x0+c, y0+(m-1)\*c), ..., (x0+(n-1)\*c,y0+(m-1)\*c)

This sequence can be addressed with a single index *k* = 0, 1, ..., n\*m-1.

Given an arbitrary point (x,y), you use simple calculations to find the indexes where its four nearest neighbors are stored: they lie between rows i and i+1 and columns j and j+1 where

i = Floor((y-y0)/c), j = Floor((x-x0)/c).

The corresponding offsets are n*i + j, n*(i+1) + j, n*i + (j+1), and n*(i+1) + (j+1).

This assumes x0 <= x < x+n*c and y0 <= y < y+m*c, which is worth checking in a general-purpose program. The coordinates of these four points are readily seen to be equal to

(x0 + j\*c, y0 + i\*c), (x0 + j\*c, y0 + (i+1\*c), etc.

Now that you can solve (a) efficiently, it remains to interpolate the values found, which I will denote z00, z01, z10, and z11, situated as suggested in this crude picture (and shown more precisely in the figure at the end):

z01 z11

z

z00 z10

You wish to estimate the value of z. **It is convenient to use**[**bilinear interpolation**](http://en.wikipedia.org/wiki/Bilinear_interpolation). The formulas amount to the following: first interpolate linearly along the top and bottom edges of the square. Here, only the *x* coordinate is changing (from x0+j\*c to x0+(j+1)\*c). Rescale this to *t* varying from 0 to 1, where

t = (x - (x0+j\*c))/c = (x-x0)/c - j.

The interpolated values are

z0 = (1-t)\*z00 + t\*z10, z1 = (1-t)\*z01 + t\*z11

Now interpolate vertically between z0 and z1. First rescale *y* to *s* varying from 0 to 1, where

s = (y-y0)/c - i

and find the interpolated value

z = (1-s)\*z0 + s\*z1.

Example

Let (x0,y0) = (-180,-90), *c* = 1/24 = 0.041667, *n* = 24\*360 = 8640, and (x,y) = (54.4786674627, 17.0470721369) (that is, near latitude 17 degrees north and longitude 54.5 degrees east). Then

i = Floor((y-y0)/c) = Floor(24\*(90 + 17.047...)) = 2569

j = Floor((x-x0)/c) = Floor(24\*(180 + 54.478....)) = 5627.

The undulation for (i,j), namely z00, is found at offset 8640\*i+j = 48,619,849. The undulation for (i+1,j), namely z01, is found at offset 8640\*(i+1)+j = 48,628,489. The undulation for (i,j+1), namely z10, is found at offset 48,619,850, and the undulation for (i+1,j+1), namely z11, is found at offset 48,628,490. Suppose these have the values given; that is,

z00 = 31.945, z01 = 31.993, z10 = 31.866, and z11 = 31.911

To interpolate, compute

t = (x-x0)/c - j = 0.488019

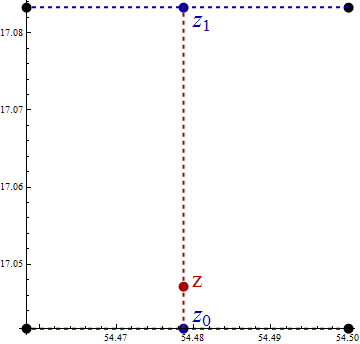
s = (y-y0)/c - i = 0.129731

Therefore

z0 = 31.96842 and z1 = 31.88796

Finally

z = 31.958.



|  |  |
| --- | --- |
| [share](https://gis.stackexchange.com/a/15804)[improve this answer](https://gis.stackexchange.com/posts/15804/edit) | answered Oct 16 '11 at 21:06  [[https://www.gravatar.com/avatar/66c15ca582c60477dc4b535d8b39558b?s=32&d=identicon&r=PG](https://gis.stackexchange.com/users/664/whuber)](https://gis.stackexchange.com/users/664/whuber)  [whuber](https://gis.stackexchange.com/users/664/whuber)**♦**  **54.9k**11147224 |

|  |  |  |  |
| --- | --- | --- | --- |
| |  |  | | --- | --- | | 2 |  | | thank you very much for your complete lecture and your time. – [daikini](https://gis.stackexchange.com/users/4520/daikini) [Oct 17 '11 at 15:25](https://gis.stackexchange.com/questions/15801/interpolate-undulations-for-given-points-using-a-geoid-model#comment20849_15804) |
| |  |  | | --- | --- | |  |  | | how the coordinates of origin of my geoid grid determine accuracy of computation? – [daikini](https://gis.stackexchange.com/users/4520/daikini) [Oct 17 '11 at 19:51](https://gis.stackexchange.com/questions/15801/interpolate-undulations-for-given-points-using-a-geoid-model#comment20879_15804) |
| |  |  | | --- | --- | |  |  | | @user Inaccuracy in the origin translates directly to the same inaccuracy in the coordinates (x,y) of the GPS point. How much of a difference that makes depends on how variable the surrounding values z00, ..., z11 are. But usually there is *no* effective inaccuracy in grid coordinates: the grid is established, measurements are taken at the grid nodes, and any inaccuracy due to imprecise spatial location in the field is just rolled up into the overall inaccuracy of the measurements themselves. – [whuber**♦**](https://gis.stackexchange.com/users/664/whuber) [Oct 17 '11 at 20:32](https://gis.stackexchange.com/questions/15801/interpolate-undulations-for-given-points-using-a-geoid-model#comment20884_15804) |
| |  |  | | --- | --- | | 2 |  | | For the record, check also [geographiclib.sourceforge.net/cgi-bin/GeoidEval](http://geographiclib.sourceforge.net/cgi-bin/GeoidEval) – [markusN](https://gis.stackexchange.com/users/687/markusn) [Dec 30 '11 at 14:45](https://gis.stackexchange.com/questions/15801/interpolate-undulations-for-given-points-using-a-geoid-model#comment24542_15804) |

add a comment

# Appendix B : Simulation Results