# coords_transform 库原理及使用方法

## 一、内容简介

之前分享过一篇博文——《用 Python 将火星，百度坐标转 WGS84 坐标》，之后在 GitHub 上也看到了相关的数据偏移的完整算法（包括 WGS84 转 GCJ02、WGS84 转 BD09、GCJ02 转 BD09、BD09 转 GCJ02、BD09 转 WGS84、GCJ02 转 WGS84 及使用百度及高德地图的接口将 WGS84 坐标转为 GCJ02 或 BD09 坐标），在实际项目应用场景中，有一批影像及矢量图斑数据，需要发布成地图服务和高德地图进行匹配，于是我调用以上算法进行影像数据及矢量数据的加密偏移，为了之后的使用方便，我将整个算法与之后的数据偏移部分打包为 Python 的第三方库，并在 GitHub 上开源，该开源包命名为 coords_transform（必须说明的一点是：GitHub 上提供的算法仅能满足精度不高的情况，精度要求在 cm 级甚至更高精度，请谨慎使用）。

**WGS84：** 国际坐标系，为一种大地坐标系，也是目前广泛使用的 GPS 全球卫星定位系统使用的坐标系。目前我们使用 GPS 定位所得的经纬度数据一般均为 WGS84 坐标系。

**GCJ02：** 火星坐标系，是由中国国家测绘局制订的地理信息系统的坐标系统。由 WGS84 坐标系经加密后的坐标系，加密方法并非线性加密，每个地区偏移的都不一样。目前在谷歌地图中国版图区域、高德地图、腾讯地图等常用地图中均使用 GCJ02 坐标系。

**BD09：** 为百度坐标系，在 GCJ02 坐标系基础上再次加密。其中 BD0911 表示百度经纬度坐标，BD09mc 表示百度墨卡托米制坐标，目前百度地图使用的就是此套坐标系。

## 二、剖析加密算法原理

## 1、WGS84-->>GCJ02 转换公式

据一些大佬解析出的算法源码，我整理了一下 WGS84 转 GCJ02 的公式，看完后直接放弃就行，没必要太过于深究，知道大体偏移原理即可！

$$a = 6378245 \ \text{长半轴}, \ e = 0.00669342162296594323 \ \text{扁率平方}$$

$$lon = lon_{wgs} - 105, \ lat = lat_{wgs} - 35$$

$$\Delta lon = 300 + lon + 2lat + 0.1lon^2 + 0.1lon * lat + 0.1\sqrt{|lon|} + \left(20\sin(6\pi lon) + 20\sin(2\pi lon) * \frac{2}{3}\right) + \left(20\sin(\pi lon) + 40\sin\left(\frac{\pi}{3}lon\right) * \frac{2}{3}\right) + \left(150\sin\left(\frac{\pi}{12}lon\right) + 300\sin\left(\frac{\pi}{30}lon\right) * \frac{2}{3}\right)$$

$$\Delta lat = -100 + 2lon + 3lat + 0.2lat^2 + 0.1lon * lat + 0.2\sqrt{|lon|} + \left(20\sin(6\pi lon) + 20\sin(2\pi lon) * \frac{2}{3}\right) + \left(20\sin(\pi lat) + 40\sin\left(\frac{\pi}{3}lat\right) * \frac{2}{3}\right) + \left(160\sin\left(\frac{\pi}{12}lat\right) + 320\sin\left(\frac{\pi}{30}lat\right) * \frac{2}{3}\right)$$

$$rad\_lat = \frac{\pi lat}{180}$$

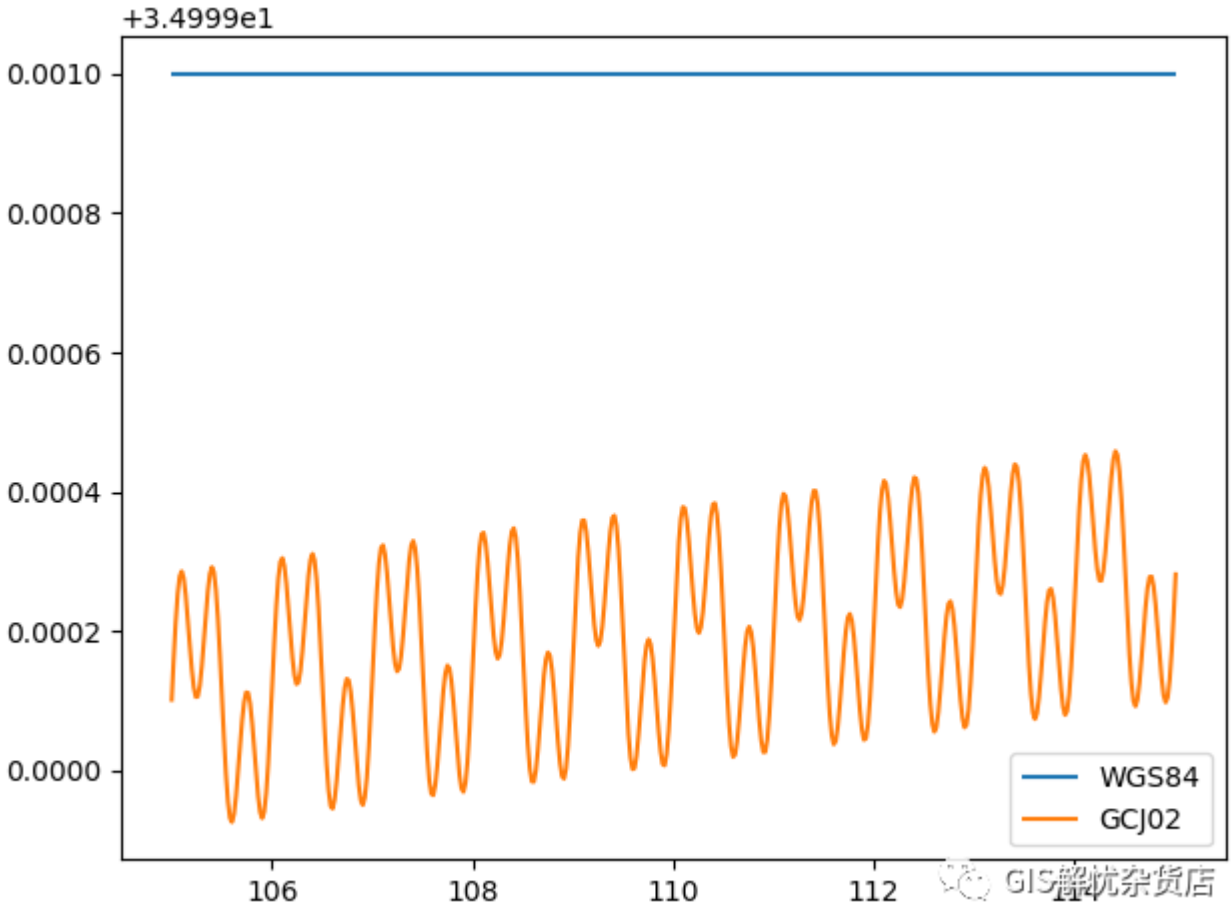
$$m = 1 - e * (\sin(rad\_lat))^2$$

$$lon_{gcj} = lon + \frac{\Delta lon * 180\sqrt{m}}{a * \pi * cos(rad\_lat)}$$

$$lat_{gcj} = lat + \frac{\Delta lat * 180m^{\frac{3}{2}}}{a * (1 - e) * \pi}$$

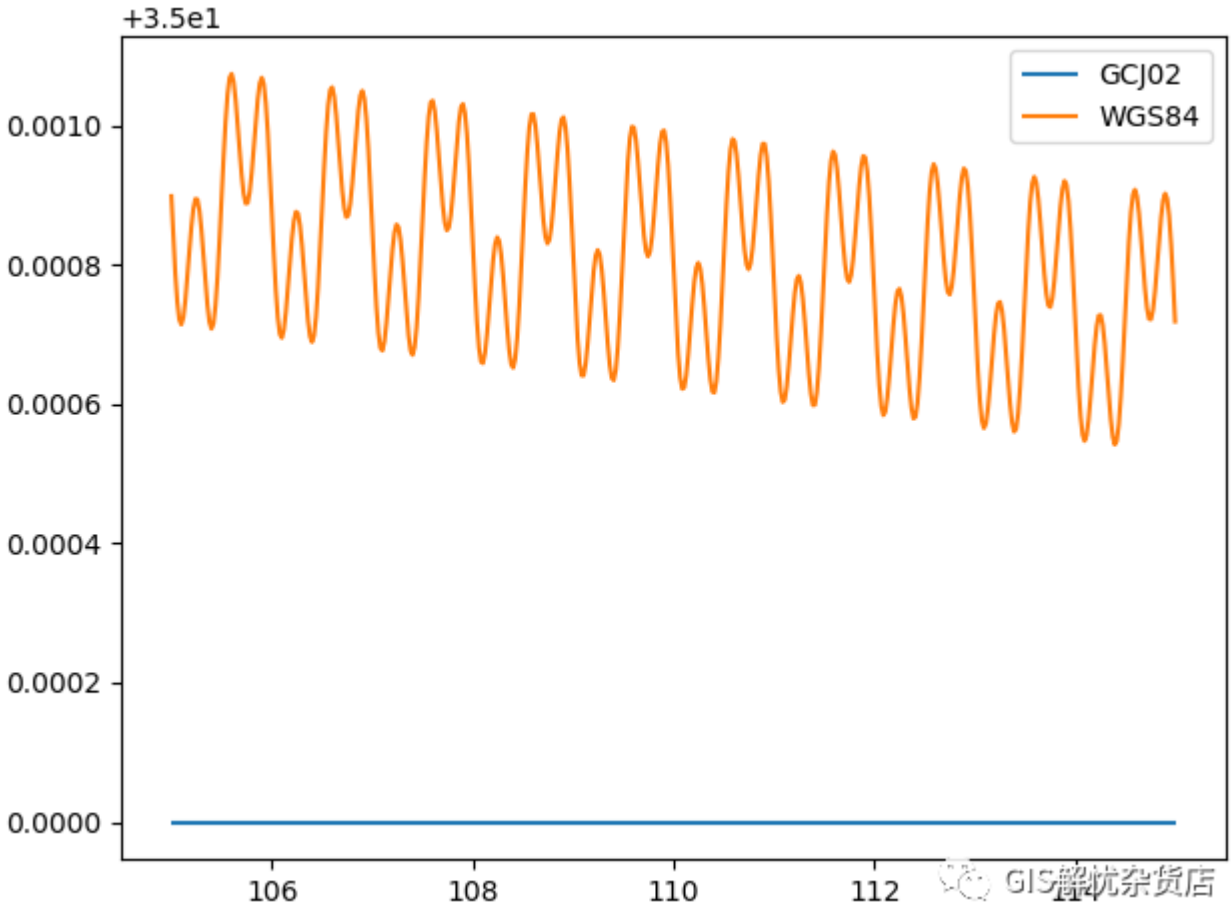
## 2、WGS84-->>GCJ02

从 WGS84 坐标系下的坐标值转换到 GCJ02 的坐标值，其做了非线性偏移，其中我列出了 WGS84 转到 GCJ02 下的偏移图示，蓝色线条为 WGS84 所在坐标系下的坐标线（经度：105~115° E，纬度：35° N），橙黄线条为经过加密算法偏移后的 GCJ02S 坐标值分布，图中明显反映出 GCJ02 坐标加密的方案为多项式+三角函数的方式：



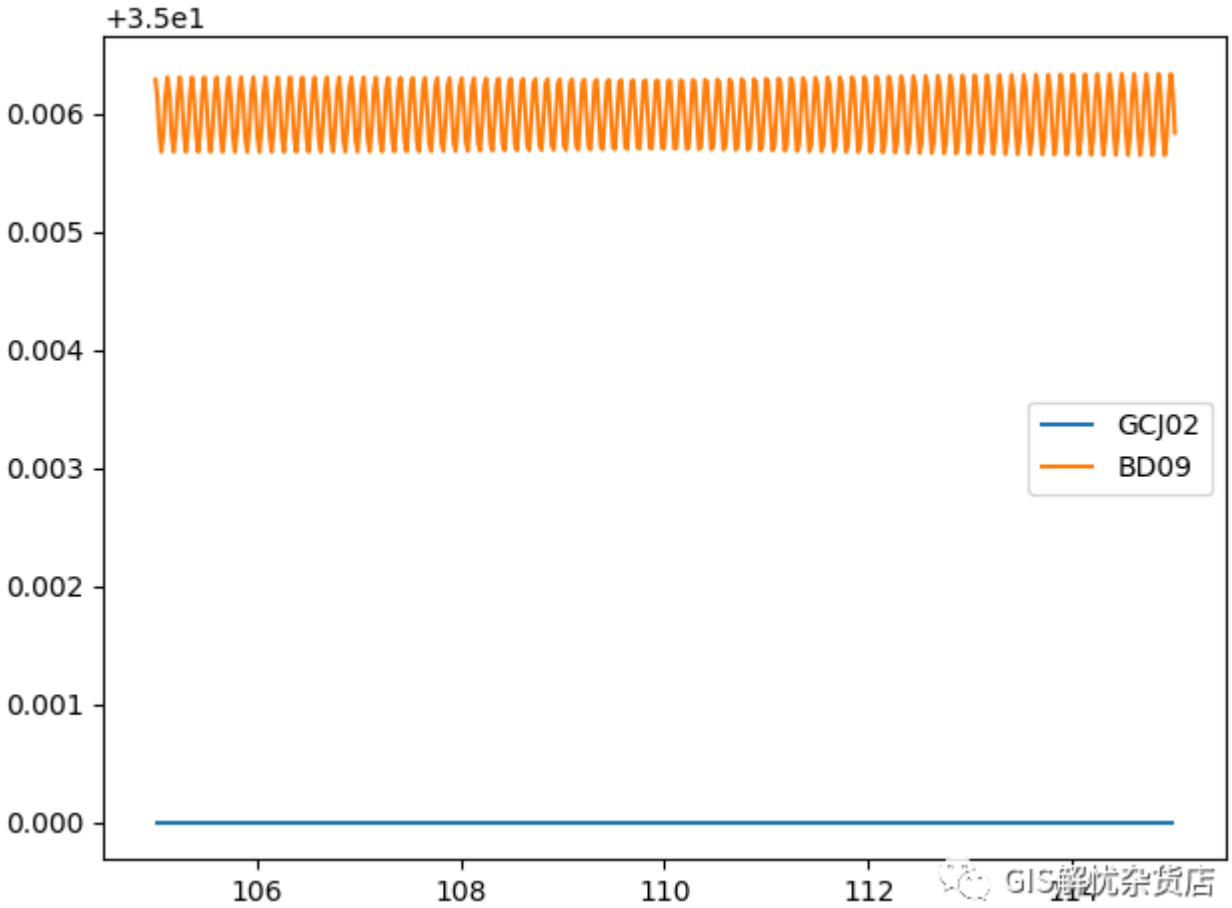

## 3、GCJ02-->>WGS84

从 GCJ02 转到 WGS84 坐标系下，其过程刚好与上一小节相反，从图示上即可看出：



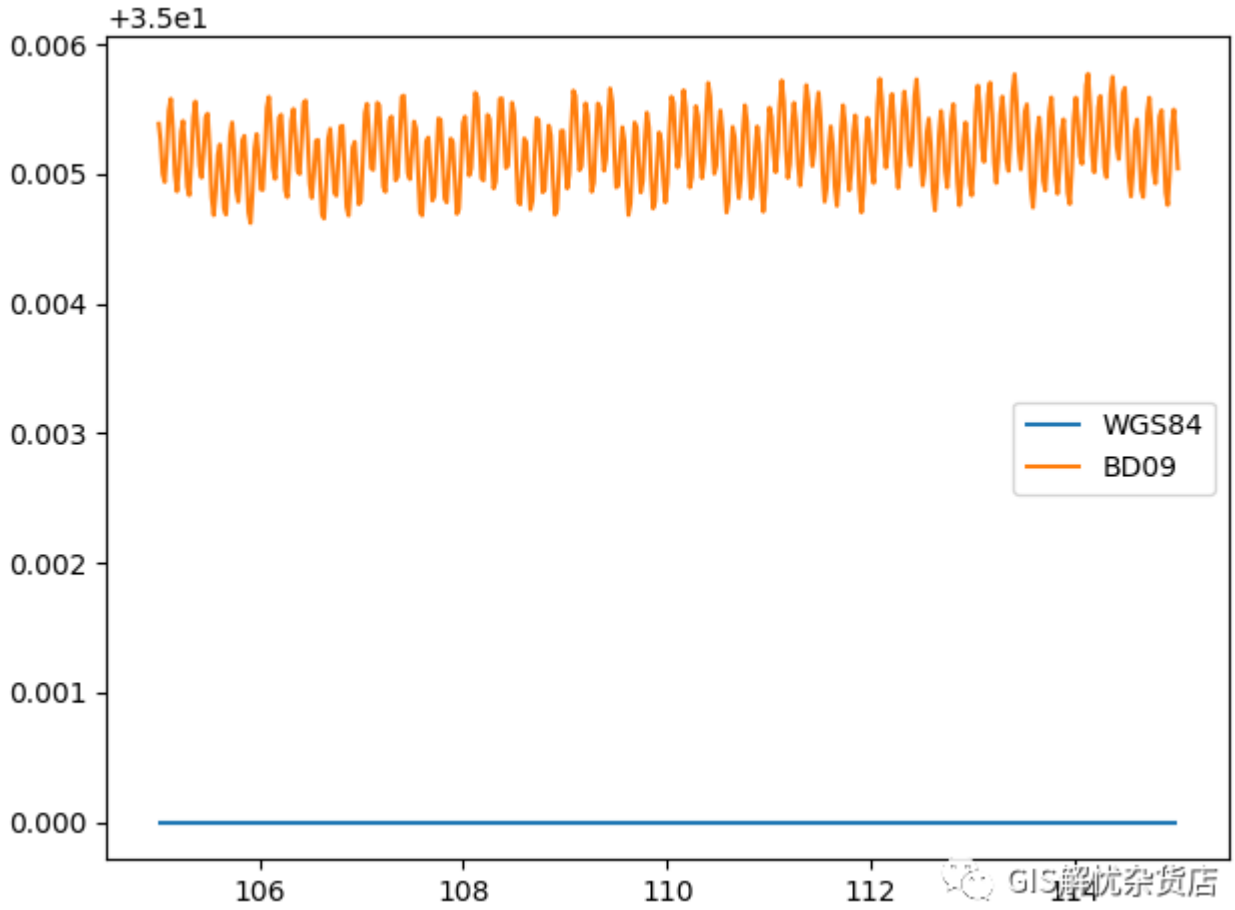

## 4、GCJ02-->>BD09

从 GCJ02 转到 BD09 坐标系下，是百度在 GCJ02 坐标系基础上进行再次加密偏移，其加密偏移的图示如下。图中能明显看出，BD09 坐标系在 GCJ02 的基础上增加了一次三角函数的变换（且值范围有波动）。




## 5、WGS84-->>BD09

在第 2 小节和第 4 小节分别剖析了 WGS84 到 GCJ02、GCJ02 转到 BD09 坐标系的图示，那么百度将两者结合，最终会产生怎样的加密偏移效果呢？如图，加密偏移算法已经相当复杂了。




# 三、coords_transform 库安装与使用（Python）

GitHub 项目地址：https://github.com/tuxiang-hub/coords_transform

## 1、下载源码与安装

目前编写在 Python3.7 环境编写，需要 GDAL 库及 shapely 库的支持，其他环境没有经过兼容性测试。在下载解压后的文件夹中启动 cmd 命令框，然后使用以下命令即可安装：

```
python setup.py install
```

若出现 python 不是内部命令的情况，请将 Python 路径添加至系统的环境变量，具体方式可以百度！

## 2、使用方法：

coords_transform 提供了 3 个模块，分别是 coords_transform、image_transform、vector_transform，下边将详细列举三个模块的具体使用方法：

**coords_transform 用来对经纬度坐标点对进行转换：**

```
from coords_transform import coords_transform#创建
coords_transform.CoordTrans()对象 class_ct =
coords_transform.CoordTrans()lon = "经度"lat = "纬度"#BD09 转
GCJ02class_ct.bd09_to_gcj02(lon,lat)#BD09 转
WGS84class_ct.bd09_to_wgs84(lon,lat)#GCJ02 转
BD09class_ct.gcj02_to_bd09(lon,lat)#GCJ02 转
WGS84class_ct.gcj02_to_wgs84(lon,lat)#WGS84 转
GCJ02class_ct.wgs84_to_gcj02(lon,lat)#WGS84 转
BD09class_ct.wgs84_to_bd09(lon,lat)#基于百度 api 接口 WGS84 转
BD09class_ct.wgs84_to_bd09_from_bdapi(lon,lat,ak="xxxxx")
```

**image_transform 模块提供对栅格数据的加密偏移，用法如下：**

```
from coords_transform import image_transform
#创建 coords_transform.CoordTrans()对象
class_ct = image_transform.ImageTransform()
class_ct.coordinate_system_conversion_img(src_dataset,dst_dataset,dst_s
r="EPSG:4326")
'''
此函数为对影像进行坐标系转换，支持不同地理坐标系之间的转换，如北京 54 转
WGS84，只不过其精度有限，凡是牵扯到地理椭球体转换的时候，该函数提供的转换精度
有限，如想准确进行不同地理椭球体之间的转换，请使用三参数或者七参数进行转换。
src_dataset：转换前源数据路径
dst_dataset：转换后输出数据路径
dst_sr：转换的目标坐标系，该参数默认为 WGS84 坐标系，当不填写该参数时，默认目标
坐标系为 WGS84
'''
class_ct.image_transform(src_dataset,dst_dataset,transform_method,inter
val,nodata)
'''
此函数功能是将栅格数据按照不同的方法进行偏移
```

```
src_dataset：偏移前的源数据路径
dst_dataset：偏移后的栅格数据路径
transform_method：包含 g2b，b2g，w2g，g2w，b2w，w2b，w2b_bdapi 等七种方法
interval：加密点，默认值 1000，当你的栅格数据范围过大时，为了提高栅格内部偏移精度，因此每隔
interval 个像素取一个坐标值参与计算转换
nodata：当栅格不规则时，避免出现黑色背景，可设置 nodata 值来去除黑边
'''
```

**vector_transform 模块提供对矢量数据的加密偏移，用法如下：**

```
from coords_transform import vector_transform
#创建 coords_transform.CoordTrans()对象
class_ct = vector_transform.VectorTransform()
class_ct.coordinate_system_conversion_vector(src_dataset, dst_dataset,
src_sr, dst_sr)
'''
此函数为对矢量数据进行坐标系转换，支持不同地理坐标系之间的转换，如北京 54 转
WGS84，只不过其精度有限，凡是牵扯到地理椭球体转换的时候，该函数提供的转换精度有
限，如想准确进行不同地理椭球体之间的转换，请使用三参数或者七参数进行转换。
src_dataset：转换前源数据路径
dst_dataset：转换后输出数据路径
src_sr：源数据坐标系的 EPSG 代码，格式为:比如 WGS84 坐标系表示为："EPSG:4326",北京
54 坐标系表示为："EPSG:4212"
dst_sr：转换的目标坐标系，填写格式如上，如："EPSG:4326"
'''
class_ct.vector_transform(src_file, dst_file, transform_method,
format="shp")
'''
此函数功能是将矢量数据按照不同的方法进行偏移
src_dataset：偏移前的源数据路径
dst_dataset：偏移后的栅格数据路径
transform_method：包含 g2b，b2g，w2g，g2w，b2w，w2b，w2b_bdapi 等七种方法
format：目前支持两种数据格式，shp 和 gdb，默认为 shp 格式
```

```
```

以上内容就是本篇博文的全部内容，coords_transform 这个 Python 开源包还有许多细节仍需要去完善，比如有些矢量数据坐标系转换后会出现中文乱码、矢量数据偏移后属性表有时会出现中文乱码等情形。包括其他的应用场景目前还没有更多去包含，欢迎感兴趣的朋友一起加入扩展此包，也欢迎大家使用，如果使用中出现各种问题可以反馈给我，我将在未来一段时间内持续维护！