姓名： 栗浩宇 学号： 20222490

《GNSS 定位新技术及数据处理方法》课后大作业（一）

GNSS 精密单点定位

任务部分：在《卫星导航定位原理》单点定位程序基础上，参考 PPPH 算法或 RTKLIB 算法，实现单点定位精度数量级的提升（到分米级或厘米级）。

选做部分：实现 mm 的静态或动态精密单点定位。

一、 算法

本次任务参考了 PPPH 算法，即采用无电离层组合消除电离层影响，结合扩展卡尔曼滤波和完整的误差改正模型实现高精度定位。

双频无电离层组合数学模型的介绍如下：

根据电离层与频率的关系（电离层延迟一阶项与频率的平方成反比）无电离层组合模型利用双/多频观测值构建无电离层组合（Ionosphere-Free, IF）观测值，消除一阶电离层延迟的影响，双频无电离层组合和观测值可以表示为：

$$P_{r,IF} = \rho_r^s + t_r - t^s + m_r^s Z_r + b_{r,IF} - b_s^{IF} + e_{r,IF}$$

$$L_{r,IF} = \rho_r^s + t_r - t^s + m_r^s Z_r + \lambda_f (N_r^{IF} + B_r^{IF} - B_s^{IF}) + \varepsilon_{r,IF}^s$$

上式中，$G_{IF} = \alpha_{ij} G_i + \beta_{ij} G_j, \quad G = P_r^s, L_r^s, b_r, b^s, \lambda B_r, \lambda B^s, \lambda N$

上式中，对流层干延迟可以通过模型改正，只剩下对流层湿延迟$m_{r,w}^s Z_{r,w}$,，卫星钟差(t^s)可以通过精密产品进行改正，值得注意的是，目前 IGS 分析中心播发的精密卫星钟差产品是通过无电离层组合估计得到的，生成的卫星钟差实际上吸收了双频无电离层组合的伪距硬件延迟，因此，精密钟差产品$\bar{t}_{IF}^s$与“真实”卫星钟差t^s的关系可以表达为：

$$\bar{t}_{IF}^s = t^s + \left(\alpha_{i,j} b_i^s + \beta_{i,j} b_i^s\right)$$

另外，由于接收机端的硬件延迟与接收机钟差参数具有强耦合性，所以它们一起估计，真实估计得到的接收机钟差为：

$$\hat{t}_{r,IF} = t_r + \left(\alpha_{i,j} b_{r,i} + \beta_{i,j} b_{r,j}\right)$$

此外，由于伪距观测值和相位观测值公用一套钟差，所以精密卫星钟差产品中的伪距硬件延迟会被代入到相位观测方程中，这部分硬件延迟与相位硬件延迟一起被待估的模糊度参数$\widehat{N}_{r,IF}^{s}$所吸收：

$$\widehat{N}_{r,IF}^{s} = B_{r,IF} - B_{IF}^{s} + \frac{\left(b_{IF}^{s} - b_{r,IF}\right)}{\lambda_{IF}}$$

经过上述考量，可以得到线性化的伪距与相位无电离层组合观测方程：

$$\mathrm{E}\left(p_{r,IF}^{s}\right) = \mu_r^s \mathrm{x} + \hat{t}_{r,IF} + m_{r,w}^{s} Z_{r,w}$$

$$\mathrm{E}\left(l_{r,IF}^{s}\right) = \mu_r^s \mathrm{x} + \hat{t}_{r,IF} + m_{r,w}^{s} Z_{r,w} + \lambda_{IF} \widehat{N}_{r,IF}^{s}$$

$p_{IF}^{s}, l_{r,IF}^{s}$分别表示观测的减去计算的（Observation Minus Computed, OMC）无电离层组合伪距与相位观测值，x为待估坐标改正数，μ_s^r为接收机到卫星的单位向量，$E(\cdot)$表示求期望算子(因为白噪声求期望为 0，所以上式中就没有噪声项了)

双频无电离层组合 PPP 模型中的待估参数包括测站坐标、对流层延迟、接收机钟差与模糊度：

$$X = \left(x, \hat{t}_{r,IF}, Z_{r,w}, \widehat{N}_{r,IF}^{s}\right)^{T}$$

程序的进行分为以下六个流程：

1 数据导入

在这个部分，有 data_read、read_sp3、read_obs、read_clock、read_atx 六个函数，他们用来读取精密星历、观测值文件、钟差文件等，用来将所有的文件导入程序。

2 数据预处理

在这个部分中，涉及许多预处理函数，用来将输入的数据进行预处理，去除有问题的、异常的数据，比如：去除钟差有问题的卫星，周跳的探测和修复、异常值处理、执行载波相位平滑伪距等内容。

3 大气改正

这个部分主要实现了几个用于大气折射延迟计算的函数，主要包括 NGPT (Next Generation Positioning Tool) 和 GMF (Global Mapping Function) 相关的算法。

ngpt2 函数：计算特定地点和时间的大气参数（压力、温度、水汽压等）

gmf_f_hu 函数：基于球谐展开计算干湿项映射函数

trop_gmf 函数：集成应用干湿项映射函数计算对流层延迟

4 模型改正

模型改正部分就是通过使用各种改正函数来处理误差。它的主流程函数是nmodel，这里面涉及很多改正函数：卫星天线相位中心改正、接收机天线相位中心改正、接收机天线参考点改正、相对论效应改正、相位缠绕改正、对流层改正等等。通过这些改正，可以获得更好的卫星定位结果。

5 卡尔曼滤波

这部分代码构成很简单，只有 GNSSfilter 和 kalman_filtering 两个函数，其中，kalman_filtering 函数是用来实现 GNSS 处理中的稳健卡尔曼滤波算法，处理单个历元的观测数据，估计接收机状态参数并处理异常观测值。GNSSfilter 函数则是实现完整的多 GNSS 系统卡尔曼滤波定位处理流程，管理所有历元数据处理，包括状态预测、更新和结果存储。

6 结果评估

这部分主要用于计算定位精度、收敛时间及各方向误差，这对于评估 PPP 解算质量至关重要。

以上六个部分是实现精密单点定位(PPP)的完整流程，下面还有两个部分：主程序和辅助函数，主程序用来调用以上部分，输出结果，辅助函数则是以上处理过程中所需要的一些其他函数。主程序不做说明，辅助函数包括：儒略日的计算、年纪日的计算、各卫星系统的载波频率和波长的计算、地心笛卡尔坐标系(XYZ)转换为大地坐标系(φ,λ,h)的转换、无电离层组合观测值的计算、将定位结果保存为JSON 格式等等，主要就是对 PPP 流程中所需要的一些函数的补充。

通过这六个部分以及辅助函数、主函数的相互配合，PPP 的实现起来就很容易了。

二、 代码

我的精密单点定位代码通过 Python 实现，代码的总量非常大，所以挑选我在 PPPH 的基础上新增的**主函数**以及**辅助函数**部分列举出来,其余部分的代码均与报告一同上交。(共有 8 个.py 文件)

1 辅助函数(helper_function.py)

```python
import numpy as np
import json
def cal2jul(year, mon, day, sec):
    if not (np.isscalar(year) and np.isscalar(mon) and np.isscalar(day)):
        raise ValueError('Year, Month and Day should be scalar.')
    if mon < 1 or mon > 12:
        raise ValueError('Month should be between 1 and 12.')
    if day < 1 or day > 31:
        raise ValueError('Day should be between 1 and 31.')
    sec = sec / 3600
    if mon <= 2:
        m = mon + 12
        y = year - 1
    else:
        m = mon
        y = year
    jd = np.floor(365.25 * y) + np.floor(30.6001 * (m + 1)) + day + (sec /
 24) + 1720981.5
    mjd = jd - 2400000.5
    return jd, mjd
def clc_doy(year, mon, day):
    if year % 400 == 0:
        sit = 1
    elif year % 300 == 0:
        sit = 0
    elif year % 200 == 0:
        sit = 0
    elif year % 100 == 0:
        sit = 0
    elif year % 4 == 0:
        sit = 1
    else:
        sit = 0
    if sit == 0:
        dom = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
    else:
        dom = [31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
    if mon - 1 < 1:
        doy = day
    else:
        doy = sum(dom[0:(mon - 1)]) + day
    return doy
def frequencies():
    c = 299792458
    freq = np.zeros((105, 2))
    wavl = np.zeros((105, 2))
    glok = [1, -4, 5, 6, 1, -4, 5, 6, -2, -7, 0, -1, -2, -7, 0, -1, 4, -
3, 3, 2, 4, -3, 3, 2, 0, 0, 0]
    for i in range(1, 106):
        idx = i - 1
        if i < 33:  # GPS
```

```
            freq[idx, 0] = 10.23 * 10 ** 6 * 154  # Hz
            wavl[idx, 0] = c / (10.23 * 10 ** 6 * 154)  # m
            freq[idx, 1] = 10.23 * 10 ** 6 * 120  # Hz
            wavl[idx, 1] = c / (10.23 * 10 ** 6 * 120)  # m
        elif i < 60:  # GLONASS
            glok_idx = i - 33
            if glok_idx < len(glok):  # 确保不会超出 glok 数组范围
                freq[idx, 0] = (1602 + 0.5625 * glok[glok_idx]) * 10 ** 6
 # Hz
                wavl[idx, 0] = c / ((1602 + 0.5625 * glok[glok_idx]) * 10
** 6)  # m
                freq[idx, 1] = (1246 + 0.4375 * glok[glok_idx]) * 10 ** 6
 # Hz
                wavl[idx, 1] = c / ((1246 + 0.4375 * glok[glok_idx]) * 10
** 6)  # m
            else:
                # 对于超出 glok 范围的 GLONASS 卫星，使用默认值
                freq[idx, 0] = (1602) * 10 ** 6  # Hz
                wavl[idx, 0] = c / ((1602) * 10 ** 6)  # m
                freq[idx, 1] = (1246) * 10 ** 6  # Hz
                wavl[idx, 1] = c / ((1246) * 10 ** 6)  # m
        elif i < 96:  # GALILEO
            freq[idx, 0] = 10.23 * 10 ** 6 * 154  # Hz
            wavl[idx, 0] = c / (10.23 * 10 ** 6 * 154)  # m
            freq[idx, 1] = 10.23 * 10 ** 6 * 115  # Hz
            wavl[idx, 1] = c / (10.23 * 10 ** 6 * 115)  # m
        else:  # BEIDOU
            freq[idx, 0] = 10.23 * 10 ** 6 * 152.6  # Hz
            wavl[idx, 0] = c / (10.23 * 10 ** 6 * 152.6)  # m
            freq[idx, 1] = 10.23 * 10 ** 6 * 118  # Hz
            wavl[idx, 1] = c / (10.23 * 10 ** 6 * 118)  # m
    return freq, wavl
def xyz2plh(cart, dopt=0):
    cart = np.array(cart).flatten()  # 确保是一维数组
    if len(cart) != 3:
        raise ValueError(f'Input matrix must have exactly 3 components, go
t {len(cart)}')
    # WGS84 ellipsoid parameters
    a = 6378137.0  # semi-major axis in meters
    f = 1 / 298.257223563  # flattening
    e2 = 2 * f - f**2  # eccentricity squared
    # Calculate longitude
    lam = np.arctan2(cart[1], cart[0])
    lam = lam % (2 * np.pi)  # normalize to [0, 2π)
    # Calculate distance from Earth's axis
    p = np.sqrt(cart[0]**2 + cart[1]**2)
    # Initial estimate of latitude
    phi0 = np.arctan(cart[2] / (p * (1 - e2)))
    # Iterative calculation of latitude and height
    while True:
        N = a / np.sqrt(1 - (e2 * (np.sin(phi0)**2)))
        h = p / np.cos(phi0) - N
        phi = np.arctan((cart[2] / p) / (1 - (N / (N + h) * e2)))
        dphi = abs(phi - phi0)
        if dphi > 10**-12:
            phi0 = phi
        else:
            break
    # Return result in requested format
    if dopt == 0:
        return np.array([phi, lam, h])
    elif dopt == 1:
        t = (180 / np.pi)
        return np.array([phi * t, lam * t, h])
def rotation(position, angle, axis):
    """
    旋转坐标变换函数
    参数:
    position (array): 3D 位置向量
    angle (float): 旋转角度(度)
    axis (int): 旋转轴(1=X, 2=Y, 3=Z)
    返回:
    array: 旋转后的位置向量
    """
    # 检查输入维度
    position = np.array(position).flatten()
    if len(position) != 3:
        raise ValueError('Matrix dimension should be 3xN or Nx3.')
    if not (np.isscalar(angle) and np.isscalar(axis)):
        raise ValueError('Angle and axis should be scalar.')
    # 创建旋转矩阵 - 注意使用角度制
    if axis == 1:  # X 轴
        rot = np.array([
            [1, 0, 0],
            [0, np.cos(np.radians(angle)), np.sin(np.radians(angle))],
            [0, -np.sin(np.radians(angle)), np.cos(np.radians(angle))]
        ])
    elif axis == 2:  # Y 轴
        rot = np.array([
            [np.cos(np.radians(angle)), 0, -np.sin(np.radians(angle))],
            [0, 1, 0],
            [np.sin(np.radians(angle)), 0, np.cos(np.radians(angle))]
        ])
    elif axis == 3:  # Z 轴
        rot = np.array([
            [np.cos(np.radians(angle)), np.sin(np.radians(angle)), 0],
            [-np.sin(np.radians(angle)), np.cos(np.radians(angle)), 0],
            [0, 0, 1]
        ])
    else:
        raise ValueError('Axis must be 1, 2, or 3.')
    # 应用旋转
    xout = rot @ position
```

```python
    return xout
def local(rec, sat, dopt):
    """
    Convert ECEF to local coordinates and calculate azimuth and elevation.
    Parameters:
    rec (array): Receiver position in ECEF
    sat (array): Satellite position in ECEF
    dopt (int): Output option (0=radians, 1=degrees)
    Returns:
    tuple: (azimuth, elevation)
    """
    if len(rec) != 3 or len(sat) != 3:
        raise ValueError('Receiver and satellite position vectors must inc
lude X,Y,Z')
    rec = np.array(rec)
    sat = np.array(sat)
    # Ensure consistent dimensions
    if rec.shape[0] != sat.shape[0]:
        if rec.shape[0] == 3:
            rec = rec.reshape(1, -1)
        elif sat.shape[0] == 3:
            sat = sat.reshape(1, -1)
    # Line of sight vector
    los = sat - rec
    # Normalized LOS vector
    p = los / np.linalg.norm(los)
    # Get geodetic coordinates
    ellp = xyz2plh(rec, 0)
    lat = ellp[0]
    lon = ellp[1]
    # Local coordinate system vectors (ENU)
    e = np.array([-np.sin(lon), np.cos(lon), 0])
    n = np.array([-np.cos(lon) * np.sin(lat), -
np.sin(lon) * np.sin(lat), np.cos(lat)])
    u = np.array([np.cos(lon) * np.cos(lat), np.sin(lon) * np.cos(lat), np
.sin(lat)])
    # Calculate elevation and azimuth
    elev = np.arcsin(np.dot(p, u))
    azim = np.arctan2(np.dot(p, e), np.dot(p, n))
    azim = azim % (2 * np.pi)
    # Convert to degrees if requested
    if dopt == 1:
        elev = np.degrees(elev)
        azim = np.degrees(azim)
    return azim, elev
def i_free(o1, o2, opt):
    """
    Calculate ionosphere-free combination.
    Parameters:
    o1 (array): First observable
    o2 (array): Second observable
    opt (int): System option (0=GPS, 1=GLONASS, 2=GALILEO, 3=BEIDOU)
    Returns:
    array: Ionosphere-free combination
    """
    if opt == 0:
        ifr = o1 * 2.545727780163160 - o2 * 1.545727780163160
    elif opt == 1:
        ifr = o1 * 2.53125 - o2 * 1.53125
    elif opt == 2:
        ifr = o1 * 2.260604327518826 - o2 * 1.260604327518826
    elif opt == 3:
        ifr = o1 * 2.487168313616925 - o2 * 1.487168313616925
    return ifr
def decimation(data, options):
    """
    根据 options 截取需要计算的历
元 (Trim observation data to specified time range)
    Parameters:
    data (dict): Data structure containing observations
    options (dict): Processing options with 'from' and 'to' time limits
    Returns:
    dict: Updated data structure with trimmed observations
    """
    if (options['from'] < data['obsd']['epoch'][0, 0]) or (options['to'] >
 data['obsd']['epoch'][-1, 0]):
        raise ValueError('Observation file may not contain data between th
e chosen interval.')
    f = (options['from'] - data['obsd']['epoch'][0, 0])
    l = (options['to'] - data['obsd']['epoch'][0, 0])
    fe = int(np.round(f / data['obsh']['time']['obsinterval'])) + 1  # fir
st epoch
    le = int(np.round(l / data['obsh']['time']['obsinterval'])) + 1  # las
t epoch
    # Trim end of data if needed
    if le < data['obsd']['st'].shape[0]:
        data['obsd']['p1'] = data['obsd']['p1'][:le, :]
        data['obsd']['p2'] = data['obsd']['p2'][:le, :]
        data['obsd']['l1'] = data['obsd']['l1'][:le, :]
        data['obsd']['l2'] = data['obsd']['l2'][:le, :]
        data['obsd']['epoch'] = data['obsd']['epoch'][:le, :]
        data['obsd']['st'] = data['obsd']['st'][:le, :]
    # Trim beginning of data if needed
    if fe != 1:
        data['obsd']['p1'] = data['obsd']['p1'][fe - 1:, :]
        data['obsd']['p2'] = data['obsd']['p2'][fe - 1:, :]
        data['obsd']['l1'] = data['obsd']['l1'][fe - 1:, :]
        data['obsd']['l2'] = data['obsd']['l2'][fe - 1:, :]
        data['obsd']['epoch'] = data['obsd']['epoch'][fe - 1:, :]
        data['obsd']['st'] = data['obsd']['st'][fe - 1:, :]
    return data
def velo(t, y, gap):
    vel = (((t - 3) * (t - 4) * (t - 5) * (t - 6) * (t - 7) * (t - 8) * (t
 - 9) * (t - 10) +
            (t - 2) * (t - 4) * (t - 5) * (t - 6) * (t - 7) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 2) * (t - 3) * (t - 5) * (t - 6) * (t - 7) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 2) * (t - 3) * (t - 4) * (t - 6) * (t - 7) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 7) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t - 7) * (t
- 9) * (t - 10) +
            (t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t - 7) * (t
- 8) * (t - 10) +
            (t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t - 7) * (t
- 8) * (t - 9)) / (-362880)) * y[0] + \
          (((t - 3) * (t - 4) * (t - 5) * (t - 6) * (t - 7) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 4) * (t - 5) * (t - 6) * (t - 7) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 3) * (t - 5) * (t - 6) * (t - 7) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 3) * (t - 4) * (t - 6) * (t - 7) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 3) * (t - 4) * (t - 5) * (t - 7) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t - 7) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t - 7) * (t
- 8) * (t - 10) +
            (t - 1) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t - 7) * (t
- 8) * (t - 9)) / (40320)) * y[1] + \
          (((t - 2) * (t - 4) * (t - 5) * (t - 6) * (t - 7) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 4) * (t - 5) * (t - 6) * (t - 7) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 2) * (t - 5) * (t - 6) * (t - 7) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 2) * (t - 4) * (t - 6) * (t - 7) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 2) * (t - 4) * (t - 5) * (t - 7) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 2) * (t - 4) * (t - 5) * (t - 6) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 2) * (t - 4) * (t - 5) * (t - 6) * (t - 7) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 2) * (t - 4) * (t - 5) * (t - 6) * (t - 7) * (t
- 8) * (t - 10) +
            (t - 1) * (t - 2) * (t - 4) * (t - 5) * (t - 6) * (t - 7) * (t
- 8) * (t - 9)) / (-10080)) * y[2] + \
          (((t - 2) * (t - 3) * (t - 5) * (t - 6) * (t - 7) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 3) * (t - 5) * (t - 6) * (t - 7) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 2) * (t - 5) * (t - 6) * (t - 7) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 2) * (t - 3) * (t - 6) * (t - 7) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 2) * (t - 3) * (t - 5) * (t - 7) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 2) * (t - 3) * (t - 5) * (t - 6) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 2) * (t - 3) * (t - 5) * (t - 6) * (t - 7) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 2) * (t - 3) * (t - 5) * (t - 6) * (t - 7) * (t
- 8) * (t - 10) +
            (t - 1) * (t - 2) * (t - 3) * (t - 5) * (t - 6) * (t - 7) * (t
- 8) * (t - 9)) / (4320)) * y[3] + \
          (((t - 2) * (t - 3) * (t - 4) * (t - 6) * (t - 7) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 3) * (t - 4) * (t - 6) * (t - 7) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 2) * (t - 4) * (t - 6) * (t - 7) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 2) * (t - 3) * (t - 6) * (t - 7) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 7) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 6) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 6) * (t - 7) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 6) * (t - 7) * (t
- 8) * (t - 10) +
            (t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 6) * (t - 7) * (t
- 8) * (t - 9)) / (-2880)) * y[4] + \
          (((t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 7) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 3) * (t - 4) * (t - 5) * (t - 7) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 2) * (t - 4) * (t - 5) * (t - 7) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 2) * (t - 3) * (t - 5) * (t - 7) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 7) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 8) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 7) * (t
- 9) * (t - 10) +
            (t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 7) * (t
- 8) * (t - 10) +
             (t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 7) * (t
- 8) * (t - 9)) / (2880)) * y[5] + \
           (((t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t - 8) * (t
- 9) * (t - 10) +
             (t - 1) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t - 8) * (t
- 9) * (t - 10) +
             (t - 1) * (t - 2) * (t - 4) * (t - 5) * (t - 6) * (t - 8) * (t
- 9) * (t - 10) +
             (t - 1) * (t - 2) * (t - 3) * (t - 5) * (t - 6) * (t - 8) * (t
- 9) * (t - 10) +
             (t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 6) * (t - 8) * (t
- 9) * (t - 10) +
             (t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 8) * (t
- 9) * (t - 10) +
             (t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t
- 9) * (t - 10) +
             (t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t
- 8) * (t - 10) +
             (t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t
- 8) * (t - 9)) / (-4320)) * y[6] + \
           (((t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t - 7) * (t
- 9) * (t - 10) +
             (t - 1) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t - 7) * (t
- 9) * (t - 10) +
             (t - 1) * (t - 2) * (t - 4) * (t - 5) * (t - 6) * (t - 7) * (t
- 9) * (t - 10) +
             (t - 1) * (t - 2) * (t - 3) * (t - 5) * (t - 6) * (t - 7) * (t
- 9) * (t - 10) +
             (t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 6) * (t - 7) * (t
- 9) * (t - 10) +
             (t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 7) * (t
- 9) * (t - 10) +
             (t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t
- 9) * (t - 10) +
             (t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t
- 7) * (t - 10) +
             (t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t
- 7) * (t - 9)) / (10080)) * y[7] + \
           (((t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t - 7) * (t
- 8) * (t - 10) +
             (t - 1) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t - 7) * (t
- 8) * (t - 10) +
             (t - 1) * (t - 2) * (t - 4) * (t - 5) * (t - 6) * (t - 7) * (t
- 8) * (t - 10) +
             (t - 1) * (t - 2) * (t - 3) * (t - 5) * (t - 6) * (t - 7) * (t
- 8) * (t - 10) +
             (t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 6) * (t - 7) * (t
- 8) * (t - 10) +
             (t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 7) * (t
- 8) * (t - 10) +
             (t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t
- 8) * (t - 10) +
            (t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t
 - 7) * (t - 10) +
            (t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t
 - 7) * (t - 8)) / (-40320)) * y[8] + \
          (((t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t - 7) * (t
 - 8) * (t - 9) +
            (t - 1) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t - 7) * (t
 - 8) * (t - 9) +
            (t - 1) * (t - 2) * (t - 4) * (t - 5) * (t - 6) * (t - 7) * (t
 - 8) * (t - 9) +
            (t - 1) * (t - 2) * (t - 3) * (t - 5) * (t - 6) * (t - 7) * (t
 - 8) * (t - 9) +
            (t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 6) * (t - 7) * (t
 - 8) * (t - 9) +
            (t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 7) * (t
 - 8) * (t - 9) +
            (t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t
 - 8) * (t - 9) +
            (t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t
 - 7) * (t - 9) +
            (t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t
 - 7) * (t - 8)) / (362880)) * y[9]
    vel = vel / gap
    return vel
# Lagrange interpolation, using ten points to fit polynomial curve
def lagrange(t, y):
    pos = (((t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t - 7) * (t
 - 8) * (t - 9) * (t - 10)) / (-362880)) * y[
        0] + \
          (((t - 1) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t - 7) * (t
 - 8) * (t - 9) * (t - 10)) / (40320)) * y[
              1] + \
          (((t - 1) * (t - 2) * (t - 4) * (t - 5) * (t - 6) * (t - 7) * (t
 - 8) * (t - 9) * (t - 10)) / (-10080)) * y[
              2] + \
          (((t - 1) * (t - 2) * (t - 3) * (t - 5) * (t - 6) * (t - 7) * (t
 - 8) * (t - 9) * (t - 10)) / (4320)) * y[3] + \
          (((t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 6) * (t - 7) * (t
 - 8) * (t - 9) * (t - 10)) / (-2880)) * y[
              4] + \
          (((t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 7) * (t
 - 8) * (t - 9) * (t - 10)) / (2880)) * y[5] + \
          (((t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t
 - 8) * (t - 9) * (t - 10)) / (-4320)) * y[
              6] + \
          (((t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t
 - 7) * (t - 9) * (t - 10)) / (10080)) * y[
              7] + \
          (((t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t
 - 7) * (t - 8) * (t - 10)) / (-40320)) * y[
              8] + \
          (((t - 1) * (t - 2) * (t - 3) * (t - 4) * (t - 5) * (t - 6) * (t
 - 7) * (t - 8) * (t - 9)) / (362880)) * y[9]
    return pos
def entrp(nep, gap, dat):
    """
    改进版插值函数
    参数:
    nep (float): 插值时刻
    gap (float): 时间间隔
    dat (array): 数据数组
    返回:
    tuple: (插值结果, 速度)
    """
    min_idx = 1
    max_idx = int(86400 / gap)
    n = (nep / gap) + 1
    # 检查数据维度
    if dat.ndim > 1:
        data_column = dat[:, 0]
    else:
        data_column = dat
    # 数据长度检查
    if len(data_column) < 10:
        print(f"数据长度不足: {len(data_column)}")
        return np.nan, np.nan
    # 处理起始边界情况
    if n < (min_idx + 5):
        # 如果太靠近数据开始
        kern = data_column[min_idx - 1:min_idx + 9]
        if len(kern) < 10:
            # 尝试使用多项式拟合
            print(f"起始边界数据不足: {len(kern)}")
            return np.nan, np.nan
        nt = n - (min_idx - 1)  # 调整索引位置
        out1 = lagrange(nt, kern)
        out2 = velo(nt, kern, gap)
    # 处理结束边界情况
    elif n > (max_idx - 5):
        # 如果太靠近数据结束
        if max_idx <= 10:
            # 数据总量不足
            print("数据总量不足 10 个点")
            return np.nan, np.nan
        kern = data_column[max_idx - 10:max_idx]
        if len(kern) < 10:
            print(f"结束边界数据不足: {len(kern)}")
            return np.nan, np.nan
        nt = n - (max_idx - 10)  # 调整索引位置
        out1 = lagrange(nt, kern)
        out2 = velo(nt, kern, gap)
    # 处理正常情况
    else:
        st = int(np.floor(n)) - 5
        fn = int(np.ceil(n)) + 4
        # 边界调整
        st = max(0, st)
        fn = min(len(data_column), fn)
        # 数据检查
        if fn - st < 10:
            print(f"中间区域数据不足: {fn - st}")
            return np.nan, np.nan
        kern = data_column[st:fn]
        nt = n - st  # 调整索引位置
        try:
            out1 = lagrange(nt, kern)
            out2 = velo(nt, kern, gap)
        except Exception as e:
            print(f"插值计算异常: {str(e)}")
            return np.nan, np.nan
    return out1, out2
def entrp_orbt(nep, gap, dat):
    """
    改进版轨道插值函数
    参数:
    nep (float): 插值时刻
    gap (float): 时间间隔
    dat (array): 数据数组
    返回:
    tuple: (插值结果, 速度)
    """
    n = (nep / gap) + 6
    # 确保数据维度正确
    if dat.ndim > 1:
        data_column = dat[:, 0]
    else:
        data_column = dat
    # 计算数据范围
    st = int(np.floor(n)) - 5
    fn = int(np.ceil(n)) + 5
    # 边界调整
    st = max(0, st)
    fn = min(len(data_column), fn)
    # 数据检查
    if fn - st < 10:
        # 数据不足 10 个点
        print(f"轨道数据不足: st={st}, fn={fn}, 长度={fn - st}")
        return np.nan, np.nan
    kern = data_column[st:fn]
    nt = n - st  # 调整到局部坐标
    try:
        out1 = lagrange(nt, kern)
        out2 = velo(nt, kern, gap)
    except Exception as e:
        print(f"轨道插值异常: {str(e)}")
        return np.nan, np.nan
    return out1, out2
def arc_dtr(obs):
    """
    Detect continuous observation arcs in satellite data.
    Parameters:
    obs (dict): Observation data structure
    Returns:
    list: List of arrays containing start and end epoch indices for each c
ontinuous arc
    """
    sn = obs['st'].shape[1]  # Changed from obs.st to obs['st']
    arc = [None] * sn
    for k in range(sn):
        # Find indices where p1 is NaN and st is 1, then set st to 0
        dc = np.where((np.isnan(obs['p1'][:, k])) & (obs['st'][:, k] == 1)
)[0]  # Changed from obs.p1, obs.st
        if len(dc) > 0:
            obs['st'][dc, k] = 0  # Changed from obs.st
        # Find indices where st equals 1
        row = np.where(obs['st'][:, k] == 1)[0]  # Changed from obs.st
        if len(row) == 0:
            arc[k] = np.array([])
            continue
        # Find breaks in continuous observations
        brk = np.where(np.diff(row) != 1)[0]
        # Create arrays for start and end indices
        frs = np.concatenate(([0], brk + 1))
        lst = np.concatenate((brk, [len(row) - 1]))
        als = np.column_stack((frs, lst))
        # Remove arcs with less than 10 continuous observations
        if als.shape[0] > 1:
            i = als.shape[0] - 1
            while i >= 0:
                if (als[i, 1] - als[i, 0]) < 10:
                    als = np.delete(als, i, axis=0)
                i -= 1
        elif als.shape[0] == 1:
            if (als[0, 1] - als[0, 0]) < 10:
                als = np.array([])
        # Create output array with actual epoch indices
        if len(als) > 0:
            arn = np.full((als.shape[0], 2), np.nan)
            for i in range(als.shape[0]):
                arn[i, 0] = row[int(als[i, 0])]
                arn[i, 1] = row[int(als[i, 1])]
            arc[k] = arn
        else:
```

```python
            arc[k] = np.array([])
    return arc
def rotation(position, angle, axis):
    """
    旋转坐标变换函数
    参数:
    position (array): 3D 位置向量
    angle (float): 旋转角度(度)
    axis (int): 旋转轴(1=X, 2=Y, 3=Z)
    返回:
    array: 旋转后的位置向量
    """
    # 检查输入维度
    position = np.array(position).flatten()
    if len(position) != 3:
        raise ValueError('Matrix dimension should be 3xN or Nx3.')
    if not (np.isscalar(angle) and np.isscalar(axis)):
        raise ValueError('Angle and axis should be scalar.')
    # 创建旋转矩阵 - 注意使用角度制
    if axis == 1:  # X 轴
        rot = np.array([
            [1, 0, 0],
            [0, np.cos(np.radians(angle)), np.sin(np.radians(angle))],
            [0, -np.sin(np.radians(angle)), np.cos(np.radians(angle))]
        ])
    elif axis == 2:  # Y 轴
        rot = np.array([
            [np.cos(np.radians(angle)), 0, -np.sin(np.radians(angle))],
            [0, 1, 0],
            [np.sin(np.radians(angle)), 0, np.cos(np.radians(angle))]
        ])
    elif axis == 3:  # Z 轴
        rot = np.array([
            [np.cos(np.radians(angle)), np.sin(np.radians(angle)), 0],
            [-np.sin(np.radians(angle)), np.cos(np.radians(angle)), 0],
            [0, 0, 1]
        ])
    else:
        raise ValueError('Axis must be 1, 2, or 3.')
    # 应用旋转
    xout = rot @ position
    return xout
def dtr_satno(obs):
    """
    确定每个历元可观测的卫星个数
    参数:
    obs (dict): 观测数据字典
    返回:
    numpy.ndarray: 每个历元可观测的卫星数量
    """
    m = obs['st'].shape[0]  # 历元数
    satno = np.zeros((m, 1))
    for i in range(m):
        satno[i, 0] = np.sum(obs['st'][i, :])
    return satno
def entrp_clkf(nep, gap, dat):
    """
    基于多项式拟合的钟差插值函数
    参数:
    nep (float): 需要插值的时刻
    gap (float): 时间间隔
    dat (numpy.ndarray): 数据数组
    返回:
    float: 插值结果
    """
    min_idx = 1
    max_idx = int(86400 / gap)
    n = (nep / gap) + 1
    ns = int((10 * 60) / gap)  # 10 分钟的点数
    nt = int((20 * 60) / gap)  # 20 分钟的点数
    if (n - ns) <= min_idx:
        st = min_idx
        fn = min_idx + (nt - 1)
        t = np.arange(st, fn + 1)
        kern = dat[st - 1:fn, 0] if dat.ndim > 1 else dat[st - 1:fn]
        # 使用 2 阶多项式拟合
        p = np.polyfit(t, kern, 2)
        out1 = np.polyval(p, n)
    elif (n + ns) > max_idx:
        st = max_idx - (nt - 1)
        fn = max_idx
        t = np.arange(st, fn + 1)
        kern = dat[st - 1:fn, 0] if dat.ndim > 1 else dat[st - 1:fn]
        p = np.polyfit(t, kern, 2)
        out1 = np.polyval(p, n)
    else:
        st = int(np.floor(n)) - (ns - 1)
        fn = int(np.ceil(n)) + (ns - 1)
        t = np.arange(st, fn + 1)
        kern = dat[st - 1:fn, 0] if dat.ndim > 1 else dat[st - 1:fn]
        p = np.polyfit(t, kern, 2)
        out1 = np.polyval(p, n)
    return out1
def moon(mjd):
    """
    计算地月矢量（月球相对地心位置）
    参数:
    mjd (float): 约化儒略日
    返回:
    numpy.ndarray: 月球在地固系下的位置向量（单位：m）
    """
    T = (mjd - 51544.5) / 36525
    L0 = (218.31617 + 481267.88088 * T - 1.3972 * T) % 360  # 度
    l = (134.96292 + 477198.86753 * T) % 360  # 度
    lp = (357.52543 + 35999.04944 * T) % 360  # 度
    F = (93.27283 + 483202.01873 * T) % 360  # 度
    D = (297.85027 + 445267.11135 * T) % 360  # 度
    obl = 23.43929111  # 度
    # 计算月球位置
    long = (L0 + (22640 * np.sin(np.radians(l)) + 769 * np.sin(np.radians(
2 * l))
                  - 4586 * np.sin(np.radians(l - 2 * D)) + 2370 * np.sin(n
p.radians(2 * D))
                  - 668 * np.sin(np.radians(lp)) - 412 * np.sin(np.radians
(2 * F))
                  - 212 * np.sin(np.radians(2 * l - 2 * D)) - 206 * np.sin
(np.radians(l + lp - 2 * D))
                  + 192 * np.sin(np.radians(l + 2 * D)) - 165 * np.sin(np.
radians(lp - 2 * D))
                  + 148 * np.sin(np.radians(l - lp)) - 125 * np.sin(np.rad
ians(D))
                  - 110 * np.sin(np.radians(l + lp)) - 55 * np.sin(np.radi
ans(2 * F - 2 * D))) / 3600) % 360
    lat = ((18520 * np.sin(
        np.radians(F + long - L0 + (412 * np.sin(np.radians(2 * F)) + 541
* np.sin(np.radians(lp))) / 3600))
            - 526 * np.sin(np.radians(F - 2 * D)) + 44 * np.sin(np.radians
(l + F - 2 * D))
            - 31 * np.sin(np.radians(-
l + F - 2 * D)) - 25 * np.sin(np.radians(-2 * l + F))
            - 23 * np.sin(np.radians(lp + F - 2 * D)) + 21 * np.sin(np.rad
ians(-l + F))
            + 11 * np.sin(np.radians(-lp + F - 2 * D))) / 3600) % 360
    dist = (385000 - 20905 * np.cos(np.radians(l)) - 3699 * np.cos(np.radi
ans(2 * D - l))
            - 2956 * np.cos(np.radians(2 * D)) - 570 * np.cos(np.radians(2
 * l)) + 246 * np.cos(
                np.radians(2 * l - 2 * D))
            - 205 * np.cos(np.radians(lp - 2 * D)) - 171 * np.cos(np.radia
ns(l + 2 * D))
            - 152 * np.cos(np.radians(l + lp - 2 * D)))  # km
    # 计算月球在黄道坐标系下的位置
    m_pos = np.array([
        dist * np.cos(np.radians(long)) * np.cos(np.radians(lat)),
        dist * np.sin(np.radians(long)) * np.cos(np.radians(lat)),
        dist * np.sin(np.radians(lat))
    ])
    # 转换到赤道坐标系
    m_pos = rotation(m_pos, -obl, 1)
    # 转换到地固坐标系（ECI to ECEF）
    fday = mjd - np.floor(mjd)
    JDN = mjd - 15019.5
    gstr = (279.690983 + 0.9856473354 * JDN + 360 * fday + 180) % 360
    m_pos = rotation(m_pos, gstr, 3)
    # 转换单位从 km 到 m
    m_pos = m_pos * 1000
    return m_pos
def sun(mjd):
    """
    计算太阳位置相对于地心
    参数:
    mjd (float): 约化儒略日
    返回:
    numpy.ndarray: 太阳在地固系的位置向量（单位：m）
    """
    AU = 149597870700  # 天文单位，单位：米
    d2r = np.pi / 180  # 度转弧度
    fday = mjd - np.floor(mjd)  # 一天中的小数部分
    JDN = mjd - 15019.5
    # 太阳位置计算参数
    v1 = (279.696678 + 0.9856473354 * JDN) % 360  # 度
    gstr = (279.690983 + 0.9856473354 * JDN + 360 * fday + 180) % 360  # 度
    g = np.radians((358.475845 + 0.9856002670 * JDN) % 360)  # 弧度
    # 计算太阳黄经
    slong = v1 + (1.91946 - 0.004789 * JDN / 36525) * np.sin(g) + 0.020094
 * np.sin(2 * g)  # 度
    obliq = np.radians(23.45229 - 0.0130125 * JDN / 36525)  # 黄道倾角，弧度
    # 计算太阳位置
    slp = np.radians(slong - 0.005686)
    snd = np.sin(obliq) * np.sin(slp)
    csd = np.sqrt(1 - snd ** 2)
    sdec = np.degrees(np.arctan2(snd, csd))  # 太阳赤纬，度
    sra = 180 - np.degrees(np.arctan2((snd / csd / np.tan(obliq)), (-
np.cos(slp) / csd)))  # 太阳赤经，度
    # 太阳在赤道坐标系中的位置（单位：米）
    s_pos = np.array([
        np.cos(np.radians(sdec)) * np.cos(np.radians(sra)) * AU,
        np.cos(np.radians(sdec)) * np.sin(np.radians(sra)) * AU,
        np.sin(np.radians(sdec)) * AU
    ])
    # 转换到地固系
    s_pos = rotation(s_pos, gstr, 3)
    return s_pos
def dtr_satlist(obs):
    """
    创建每个历元可见卫星的列表
    参数:
    obs (dict): 观测数据字典，包含 st 字段
    返回:
    list: 每个历元可见卫星的索引列表
    """
    m = obs['st'].shape[0]  # 历元数
    satlist = [[] for _ in range(m)]
    for i in range(m):
        # 找到所有 st 值为 1 的卫星索引
        satlist[i] = np.where(obs['st'][i, :] == 1)[0] + 1
    return satlist
def dtr_sys(options):
    """
    根据启用的卫星系统确定系统参数
    参数:
    options (dict): 计算选项字典
    返回:
    tuple: (bp, ap, sit) 基本参数数量，总参数数量，卫星系统组合情况代码
    """
    # 对流层梯度选择
    if options['TroGrad'] == 0:
        base = 5
    elif options['TroGrad'] == 1:
        base = 7
    if options['system']['gps'] == 1:
        if options['system']['glo'] == 1:
            if options['system']['gal'] == 1:
                if options['system']['bds'] == 1:
                    bp = base + 3
                    ap = 92
                    sit = 12
                else:
                    bp = base + 2
                    ap = 78
                    sit = 8
            elif options['system']['bds'] == 1:
                bp = base + 2
                ap = 72
                sit = 9
            else:
                bp = base + 1
                ap = 58
                sit = 3
        elif options['system']['gal'] == 1:
            if options['system']['bds'] == 1:
                bp = base + 2
                ap = 66
                sit = 10
            else:
                bp = base + 1
                ap = 52
                sit = 4
        elif options['system']['bds'] == 1:
            bp = base + 1
            ap = 46
```

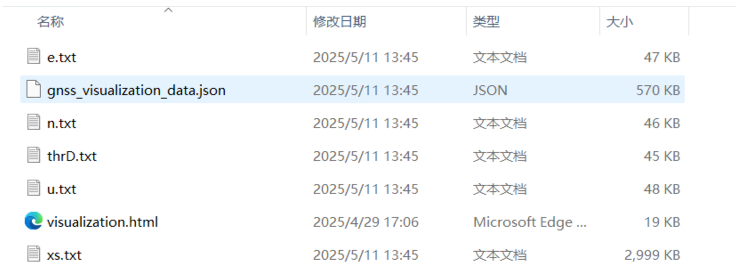
```python
            sit = 5
        else:
            bp = base
            ap = 32
            sit = 1
    elif options['system']['glo'] == 1:
        if options['system']['gal'] == 1:
            if options['system']['bds'] == 1:
                bp = base + 2
                ap = 60
                sit = 11
            else:
                bp = base + 1
                ap = 46
                sit = 6
        elif options['system']['bds'] == 1:
            bp = base + 1
            ap = 40
            sit = 7
        else:
            bp = base
            ap = 26
            sit = 2
    else:
        raise ValueError("Process must include GPS or GLONASS satellites")
    return bp, ap, sit
def save_visualization_data(xs, n, e, u, thrD, rms, CT):
    """
    将 GNSS 定位结果保存为可视化所需的 JSON 格式
    参数:
    xs (numpy.ndarray): 定位结果
    n, e, u (numpy.ndarray): 北向、东向、高程误差
    thrD (numpy.ndarray): 三维距离误差
    rms (numpy.ndarray): 各方向 RMS 误差
    CT (int): 收敛时间
    """
    # 创建数据列表
    viz_data = []
    # 遍历每个历元
    for i in range(thrD.shape[0]):
        epoch_data = {
            "epoch": i + 1,
            "thrD": float(thrD[i]),
            "x": float(xs[0, i]),
            "y": float(xs[1, i]),
            "z": float(xs[2, i]),
            "n": float(n[i]),
            "e": float(e[i]),
            "u": float(u[i])
        }
        viz_data.append(epoch_data)
    # 添加元数据
    metadata = {
        "convergenceTime": int(CT),
        "rmsN": float(rms[0, 0]),
        "rmsE": float(rms[1, 0]),
        "rmsU": float(rms[2, 0]),
        "rms3D": float(np.sqrt(rms[0, 0] ** 2 + rms[1, 0] ** 2 + rms[2, 0]
 ** 2)),
        "referencePosition": {
            "x": float(np.mean(xs[0, :])),
            "y": float(np.mean(xs[1, :])),
            "z": float(np.mean(xs[2, :]))
        }
    }
    # 创建完整数据对象
    full_data = {
        "metadata": metadata,
        "epochs": viz_data
    }
    # 保存为JSON 文件
    with open('./result/gnss_visualization_data.json', 'w') as f:
        json.dump(full_data, f)
```

2 主函数(main.py)

```python
from gnss_filter import gnssfilter
from datahand import data_read
from pre_process import preprocess
import numpy as np
from helper_function import xyz2plh, save_visualization_data
from model_correct import nmodel
from evaluate import evaluate
if __name__ == "__main__":
    # 1. 定义数据文件路径
    files = {
        'obs': './Data/algo0670.22o',
        'sp3': './Data/gfz22002.sp3',
        'sp3a': './Data/gfz22003.sp3',
        'sp3b': './Data/gfz22001.sp3',
        'clk': './Data/gfz22002.clk',
        'atx': './Data/igs14.atx'
    }
    # 2. 配置处理选项
    options = {
        # 基本选项
        "dcb": 0,
        "clk_file": 1,
        "clk_int": 30,
        "lono": 1,
        "system": {
            "gps": 1,
            "glo": 1,
            "gal": 1,
            "bds": 1
        },
        # Kalman 滤波器相关选项
        "ProMod": 1,   # 0:运动学, 1:静态
        "NosPos": 0,
        "NosPos2": 0,
        "NosClk": 1.0,
        "NosClk2": 5,
        "NosTrop": 1.0,
        "NosTrop2": -9,
        "NosSTD": 1.0,
        "NosSTD2": -7,
        "IntPos": 1.0,
        "IntPos2": 2,
        "IntClk": 1.0,
        "IntClk2": 5,
        "IntTrop": 0.5,
        "IntTrop2": 0,
        "IntSTD": 1.0,
        "IntSTD2": 0,
        "IntAmb": 2,
        "IntAmb2": 1,
        "TroGrad": 1,
        "ApMethod": "RINEX",
        "WeMethod": "Elevation Dependent",
        "CodeStd": 3,
        "PhaseStd": 0.003
    }
    # 3. 导入数据
    print("-------------1 正在导入数据-------------")
    data = data_read(files, options)
    data["obs"] = data["obsd"]
    print("-------------1 数据导入完成-------------")
    # 4. 预处理配置与执行
    print("-------------2 正在预处理数据-------------")
    options.update({
        "from": data["obsd"]["epoch"][0, 0],
        "to": data["obsd"]["epoch"][-1, 0],
        "elvangle": 15,
        "CSMw": 1,
        "CSGf": 1,
        "clkjump": 0,
        "codsmth": 1
    })
    data = preprocess(data, options)
    print("-------------2 预处理数据完成-------------")
    # 5. 生成导航模型
    print("-------------3 正在生成导航模型-------------")
    # 添加导航模型所需参数
    options.update({
        "SatClk": 1,
        "SatAPC": 1,
        "RecAPC": 1,
        "RecARP": 1,
        "RelClk": 1,
        "SatWind": 1,
        "AtmTrop": 1,
        "Iono": 1,
        "RelPath": 1,
        "Solid": 1
    })
    model = nmodel(data, options)
    print("-------------3 生成导航模型完成-------------")
    print(f"导航模型形状: {model.shape}")
    # 6. 执行GNSS 滤波
    print("-------------4 进行滤波-------------")
    xs_mgnss, kofs_mgnss, pks_mgnss = gnssfilter(model, data, options)
    # 7. 保存定位结果
    np.savetxt('./result/xs.txt', xs_mgnss.T, fmt='%.6f', delimiter='\t')
    print(f"------------4 滤波完成-------------")
    # 8. 评估结果
    ref = np.mean(xs_mgnss[0:3, :], axis=1)
    n, e, u, CT, thrD, rms = evaluate(xs_mgnss, ref.T)
    # 计算综合中误差
    neu_rms = np.sqrt(rms[0, 0] ** 2 + rms[1, 0] ** 2 + rms[2, 0] ** 2)
    # 输出评估结果
    print(f"CT 值: {CT}")
    print(f"中误差: {neu_rms:.6f} m")
    print(f"n 方向中误差: {rms[0, 0]:.6f} m")
    print(f"e 方向中误差: {rms[1, 0]:.6f} m")
    print(f"u 方向中误差: {rms[2, 0]:.6f} m")
    # 保存误差数据
    np.savetxt('./result/n.txt', n, fmt='%.8e', delimiter='\t')
    np.savetxt('./result/e.txt', e, fmt='%.8e', delimiter='\t')
    np.savetxt('./result/u.txt', u, fmt='%.8e', delimiter='\t')
    np.savetxt('./result/thrD.txt', thrD, fmt='%.8e', delimiter='\t')
    # 9. 输出最终定位结果
    final_pos_mgnss = xs_mgnss[0:3, -1]
    geo_mgnss = xyz2plh(final_pos_mgnss, 1)
    print("\n============ 定位结果 ============")
    print(
        f"MGNSS_filter 函数最终位
置 (ECEF): X = {final_pos_mgnss[0]:.3f} m, Y = {final_pos_mgnss[1]:.3f} m,
Z = {final_pos_mgnss[2]:.3f} m")
    print(
        f"MGNSS_filter 函数最终位置 (大地坐标): 纬
度 = {geo_mgnss[0]:.6f}°, 经度 = {geo_mgnss[1]:.6f}°, 高
度 = {geo_mgnss[2]:.3f} m")
    # 10. 保存可视化数据
    save_visualization_data(xs_mgnss, n, e, u, thrD, rms, CT)
    print(f"可视化数据已保存到 gnss_visualization_data.json")
```

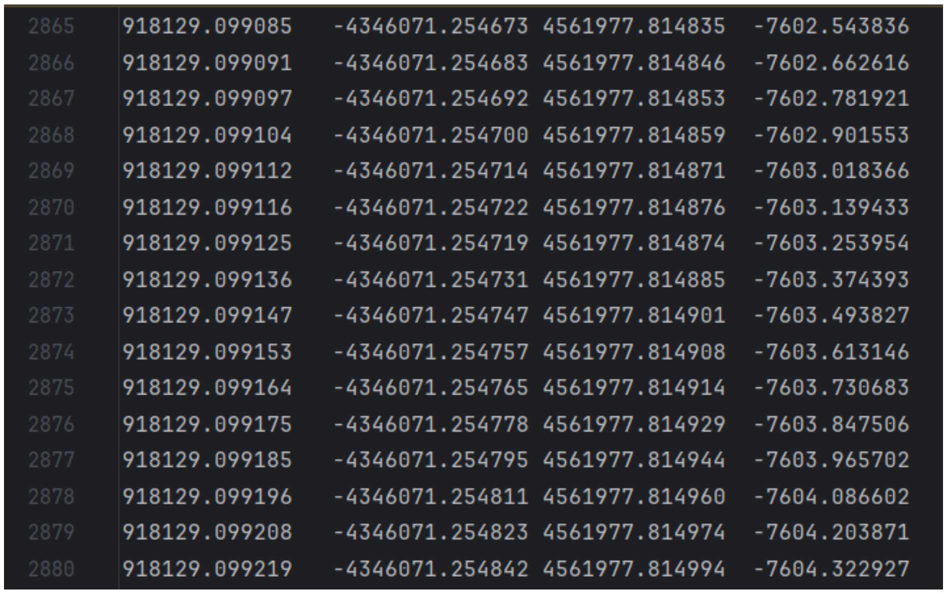
三、 结果

代码运行结果均保存在 result 文件夹下面，包括如下内容：



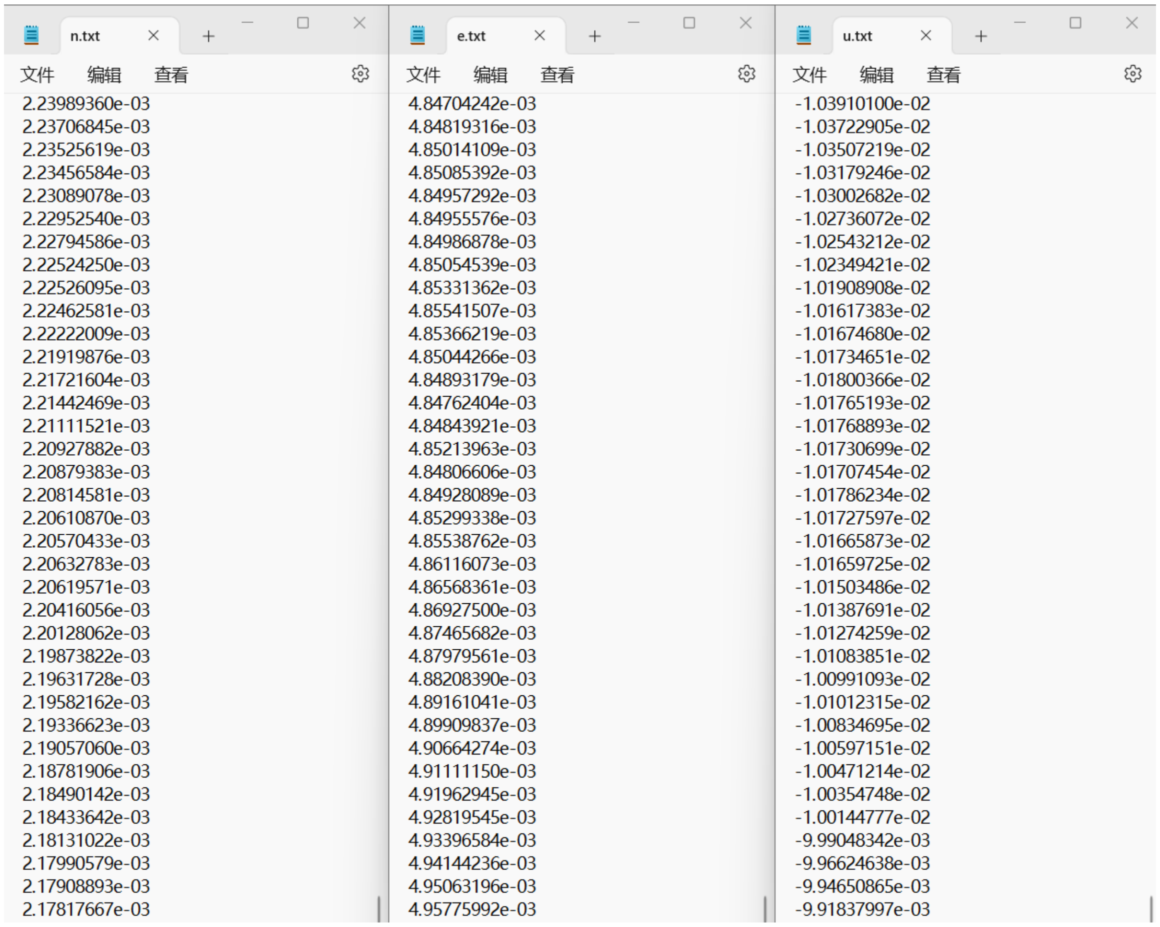
名称	修改日期	类型	大小
e.txt	2025/5/11 13:45	文本文档	47 KB
gnss_visualization_data.json	2025/5/11 13:45	JSON	570 KB
n.txt	2025/5/11 13:45	文本文档	46 KB
thrD.txt	2025/5/11 13:45	文本文档	45 KB
u.txt	2025/5/11 13:45	文本文档	48 KB
visualization.html	2025/4/29 17:06	Microsoft Edge ...	19 KB
xs.txt	2025/5/11 13:45	文本文档	2,999 KB

1 位置信息(**xs.txt** 的前三列为位置信息)



```
918129.099085   -4346071.254673 4561977.814835  -7602.543836
918129.099091   -4346071.254683 4561977.814846  -7602.662616
918129.099097   -4346071.254692 4561977.814853  -7602.781921
918129.099104   -4346071.254700 4561977.814859  -7602.901553
918129.099112   -4346071.254714 4561977.814871  -7603.018366
918129.099116   -4346071.254722 4561977.814876  -7603.139433
918129.099125   -4346071.254719 4561977.814874  -7603.253954
918129.099136   -4346071.254731 4561977.814885  -7603.374393
918129.099147   -4346071.254747 4561977.814901  -7603.493827
918129.099153   -4346071.254757 4561977.814908  -7603.613146
918129.099164   -4346071.254765 4561977.814914  -7603.730683
918129.099175   -4346071.254778 4561977.814929  -7603.847506
918129.099185   -4346071.254795 4561977.814944  -7603.965702
918129.099196   -4346071.254811 4561977.814960  -7604.086602
918129.099208   -4346071.254823 4561977.814974  -7604.203871
918129.099219   -4346071.254842 4561977.814994  -7604.322927
```

2 n、e、u 方向误差(**n.txt，e.txt，u.txt**)



n.txt

```
2.23989360e-03
2.23706845e-03
2.23525619e-03
2.23456584e-03
2.23089078e-03
2.22952540e-03
2.22794586e-03
2.22524250e-03
2.22526095e-03
2.22462581e-03
2.22222009e-03
2.21919876e-03
2.21721604e-03
2.21442469e-03
2.21111521e-03
2.20927882e-03
2.20879383e-03
2.20814581e-03
2.20610870e-03
2.20570433e-03
2.20632783e-03
2.20619571e-03
2.20416056e-03
2.20128062e-03
2.19873822e-03
2.19631728e-03
2.19582162e-03
2.19336623e-03
2.19057060e-03
2.18781906e-03
2.18490142e-03
2.18433642e-03
2.18131022e-03
2.17990579e-03
2.17908893e-03
2.17817667e-03
```

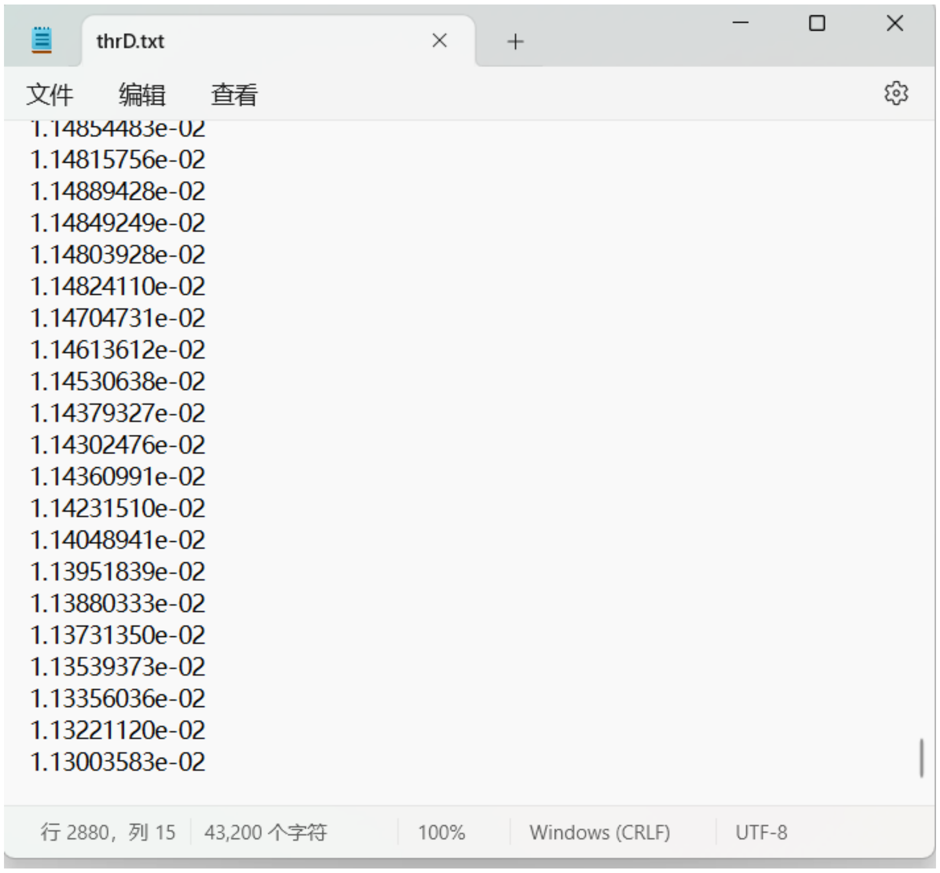
e.txt

```
4.84704242e-03
4.84819316e-03
4.85014109e-03
4.85085392e-03
4.84957292e-03
4.84955576e-03
4.84986878e-03
4.85054539e-03
4.85331362e-03
4.85541507e-03
4.85366219e-03
4.85044266e-03
4.84893179e-03
4.84762404e-03
4.84843921e-03
4.85213963e-03
4.84806606e-03
4.84928089e-03
4.85299338e-03
4.85538762e-03
4.86116073e-03
4.86568361e-03
4.86927500e-03
4.87465682e-03
4.87979561e-03
4.88208390e-03
4.89161041e-03
4.89909837e-03
4.90664274e-03
4.91111150e-03
4.91962945e-03
4.92819545e-03
4.93396584e-03
4.94144236e-03
4.95063196e-03
4.95775992e-03
```

u.txt

```
-1.03910100e-02
-1.03722905e-02
-1.03507219e-02
-1.03179246e-02
-1.03002682e-02
-1.02736072e-02
-1.02543212e-02
-1.02349421e-02
-1.01908908e-02
-1.01617383e-02
-1.01674680e-02
-1.01734651e-02
-1.01800366e-02
-1.01765193e-02
-1.01768893e-02
-1.01730699e-02
-1.01707454e-02
-1.01786234e-02
-1.01727597e-02
-1.01665873e-02
-1.01659725e-02
-1.01503486e-02
-1.01387691e-02
-1.01274259e-02
-1.01083851e-02
-1.00991093e-02
-1.01012315e-02
-1.00834695e-02
-1.00597151e-02
-1.00471214e-02
-1.00354748e-02
-1.00144777e-02
-9.99048342e-03
-9.96624638e-03
-9.94650865e-03
-9.91837997e-03
```

3 三维误差（**thrD.txt**）



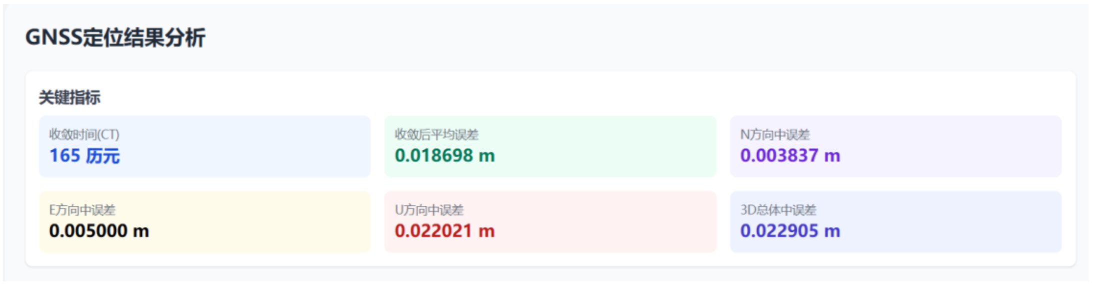
```
1.14854483e-02
1.14815756e-02
1.14889428e-02
1.14849249e-02
1.14803928e-02
1.14824110e-02
1.14704731e-02
1.14613612e-02
1.14530638e-02
1.14379327e-02
1.14302476e-02
1.14360991e-02
1.14231510e-02
1.14048941e-02
1.13951839e-02
1.13880333e-02
1.13731350e-02
1.13539373e-02
1.13356036e-02
1.13221120e-02
1.13003583e-02
```

行 2880，列 15 | 43,200 个字符 | 100% | Windows (CRLF) | UTF-8

4 可视化结果（**json+html** 文件配合使用，生成 **json** 后直接打开 **html** 即可）

主要有：

关键指标：

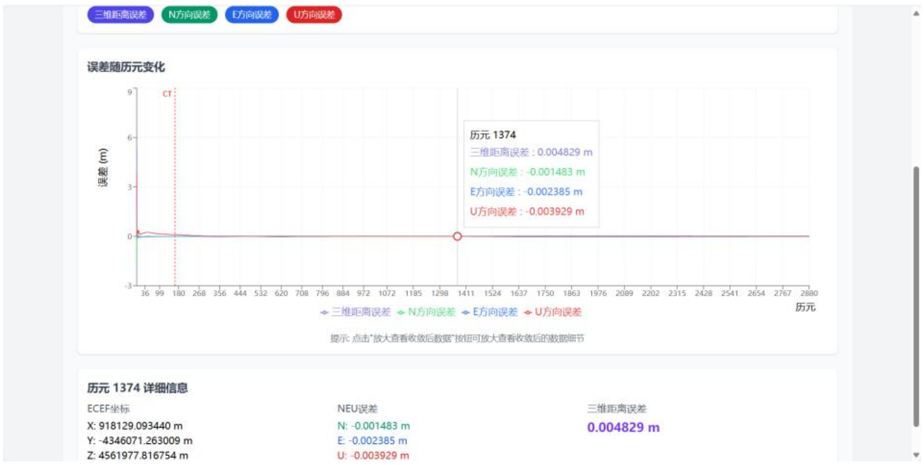


GNSS定位结果分析

关键指标

收敛时间(CT) 165 历元	收敛后平均误差 0.018698 m	N方向中误差 0.003837 m
E方向中误差 0.005000 m	U方向中误差 0.022021 m	3D总体中误差 0.022905 m

还有误差的变化曲线以及移动到某个点后的详细信息：



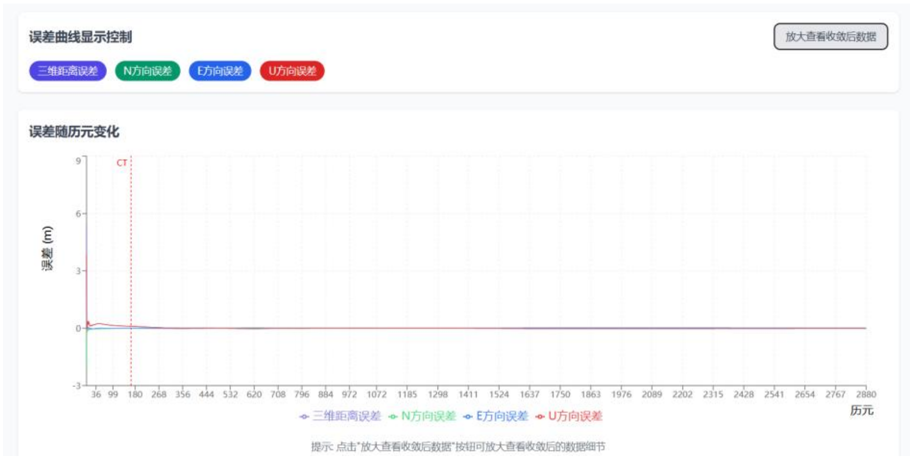
历元 1374
三维距离误差 : 0.004829 m
N方向误差 : -0.001483 m
E方向误差 : -0.002385 m
U方向误差 : -0.003929 m

提示: 点击"放大查看收敛后数据"按钮可放大查看收敛后的数据细节

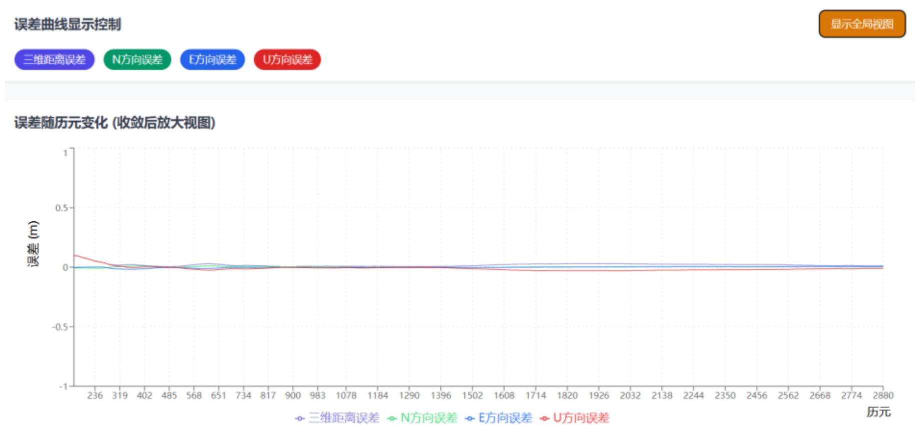
历元 1374 详细信息

ECEF坐标	NEU误差	三维距离误差
X: 918129.093440 m	N: -0.001483 m	0.004829 m
Y: -4346071.263009 m	E: -0.002385 m	
Z: 4561977.816754 m	U: -0.003929 m	

对于曲线，还有一个“放大查看收敛后数据”这个按钮，用来放大坐标轴，观察收敛后的曲线走势，更清楚地看到收敛情况。



提示: 点击"放大查看收敛后数据"按钮可放大查看收敛后的数据细节



从结果中可以看出来，定位和收敛结果还是很不错的，3D 总体中误差可达到**厘米级**精度，在某个方向上的中误差可达到**毫米级**。

四、 总结

本次任务我以 PPPH 为基础，修改之前的伪距单点定位代码，用 Python 实现了 3D 总体精度达到**厘米级**的精密单点定位系统。

在完成精密单点定位（PPP）的 Python 实现的过程中，尽管有 PPPH 的 MATLAB 源代码作为参考，我仍遇到了诸多技术挑战。其中最为棘手的是编程语言间的索引差异问题：MATLAB 数组索引从 1 开始，而 Python 则从 0 开始。这一看似细微的差别在处理庞大代码库时却影响深远，任何一处索引错误都可能导致最终结果出现重大偏差。

在调试过程中，我曾因一个卫星编号的索引错误而获得了完全错误的定位结果。这个问题困扰了我相当长的时间。尽管反复检查算法实现、验证各个模块的功能，却始终无法定位问题根源。最终，在一个不起眼的代码片段中发现了这个索引错误，修正后结果才得以正确。

除了解决所遇到的问题外，我还对 PPPH 代码架构进行了全面重构。MATLAB 的函数调用机制相对灵活，允许在不同模块间直接调用函数。而 Python 则更强调模块化设计原则。因此，我将原 PPPH 中散落在各处的辅助函数进行了系统性整合，创建了专门的辅助函数模块，使得代码结构更加清晰，函数复用更加便捷。

实现 PPP 的过程还是十分有挑战性的，PPPH 的源代码似乎是给了一个“标准答案”，但是不同编程语言的特性差异仍要求我进行大量的适配性修改。通过解决这些技术挑战，我不仅深入理解了 PPP 算法的实现细节和完整流程，还显著提升了自身的代码能力，为今后处理类似复杂项目奠定了坚实基础。