中国研究生创新实践系列大赛
“华为杯”第二十一届中国研究生数学建模竞赛

学　　校	重庆邮电大学
参赛队号	24106170095
队员姓名	1. 王新煜
	2. 唐志强
	3. 杨正涛

中国研究生创新实践系列大赛
“华为杯”第二十一届中国研究生数学建模竞赛

题　目：　X 射线脉冲星光子到达时间建模

摘　要：

脉冲星（Pulsar）是高速自转的中子星，具有体积小、密度大的特征。脉冲星的高速自转会形成脉冲，而脉冲的周期其实就是脉冲星的自转周期。在旋转过程中，脉冲星的磁场会形成强烈的电磁波，就像是“宇宙中的灯塔”，源源不断地、有规律地向外界发射电磁波。脉冲星可以提供独立、稳定的空间参考基准和时间基准，为空间飞行提供导航信标。深空航天器导航以及时间基准的维持对于大国战略安全、航天技术发展、深空探测等都有重要意义。

问题 1：通过题目中给定的六根轨道根数，计算半长轴 a 和卫星在轨道平面内的径向距离 r，然后通过旋转矩阵将卫星的位置和速度转换到 GCRS 中的三维位置和速度。计算出卫星在特定时刻的三维位置和速度分量，并且在计算上对结果进行准确验证，且与轨道参数符合。

问题 2：利用脉冲星辐射平行光的假设，忽略太阳系天体的自转和扁率，简化光子的传播路径问题，建立脉冲星光子到达卫星和太阳系质心之间的几何传播时延模型。首先将卫星的坐标从 GCRS 转换为 SSB，通过 DE 系列历表和卫星的 GCRS 坐标来计算卫星在太阳质心中的位置，从而通过计算距离差来计算几何传播时延。

问题 3：此时精确的时延模型不仅涉及光子从脉冲星到航天器传播的几何路径，还必须考虑脉冲星自行的影响以及几个关键的时延因素，通过计算这些时延因素最后得到精确时延模型 $\Delta t_{total} = \Delta t_{geo} + \Delta t_{Shapiro} + \Delta t_{redshift} + \Delta t_{SR}$

问题 4：考虑背景光子流量密度与脉冲星光子流量密度，建立 X 射线脉冲星光子序列模型，模拟脉冲星辐射的光子序列。然根据脉冲星自转周期将光子时间序列转换为相位空间，利用脉冲星光子序列折叠出脉冲轮廓。通过查阅相关文献资料，通过调整观测时间 T_{obs} 和脉冲轮廓函数分辨率来影响仿真精度，提出改进的非线性加权最小二乘估计法，使得仿真精度相对于问题 2 的仿真精度提高了 6%，从而更好展现脉冲星的辐射特性。

关键词：脉冲星光子　时延模型　光子序列　脉冲轮廓　最小二乘法

目录

1 问题重述

问题 1：若某一光子到达探测器时刻 XPNAV-1 卫星对应的轨道根数为某一光子到达探测器时刻 XPNAV-1 卫星对应的轨道根数，请计算该时刻卫星在地心天球参考系中的三维坐标 (X, Y, Z) 与速度 (v_x, v_y, v_z)，并对轨道一致性和计算结果进行验证。

问题 2：假设脉冲星辐射的 X 射线光子信号为平行光，忽略太阳系天体的自转和扁率，请建立脉冲星光子到达卫星与太阳系质心之间的真空几何传播时延模型。若光子到达卫星时刻对应的 MJD（约化儒略日）为 57062.0（TT 时间尺度），根据问题 1 中卫星在 GCRS 中的位置坐标，得到其在太阳系质心坐标系中的位置基础上，计算脉冲星光子到达卫星与太阳系质心的传播路径时间差。

问题 3：在建立脉冲星光子到达航天器（卫星等）与太阳系质心之间的精确转换时延模型时，需要考虑脉冲星自行的影响以及几个关键的时延因素：几何传播时延、Shapiro 时延、引力红移时延和狭义相对论的动钟变慢效应。在考虑脉冲星自行以及上述时延因素下，请建立脉冲星光子到达航天器与太阳系质心的精确转换时延模型。若光子到达探测器的时刻对应 MJD 为 58119.1651507519，根据问题 1 中卫星的位置、速度以及脉冲星位置参考历元、自行参数信息，计算脉冲星光子到达航天器与太阳系质心间的时延。

问题 4：光子到达时刻的仿真可以更清楚地了解脉冲星信号的辐射过程。此外，X 射线探测器发射成本高，信号仿真可以在缺少实测数据时开展脉冲星深空导航与守时方法研究。

1）建立 X 射线脉冲星光子序列模型，并仿真 Crab 脉冲星光子序列

2）根据 Crab 脉冲星的自转参数，利用仿真的脉冲星光子序列折叠出脉冲轮廓（要求脉冲轮廓对应相位区间为 [0,1]）

3）在保持上述背景光子流量密度值不变基础上，请提出一种提高仿真精度的方法

2 问题分析

2.1 问题 1 的分析

问题 1 是力学中的数学问题，属于天体力学中的轨道力学问题，具体来说是解决开普勒问题，即如何根据给定的轨道根数计算卫星在某一时刻的位置和速度向量。题目中给定的数据为偏心率 e, 角动量 h, 升交点赤经 Ω, 轨道倾角 i, 近地点幅角 ω, 真近点角 Θ, 根据这六个参数建立数学模型将轨道根数转换为 GCRS 中位置和速度向量，关键是与寻找这 6 个参数之间隐藏的关系来推导出结果，最后通过轨道方程和角动量守恒方程来验证结果的一致性。

大致思路如下:

1、计算轨道参数。通过角动量 h 和偏心率 e 以及地球的引力常数（通常 u=398600.4418km³/s²）来推导半长轴 a

$$a = \frac{h^2}{\mu \cdot (1 - e^2)}$$

2、计算距离 r 和位置坐标

$$r = \frac{h^2}{\mu} \cdot \frac{1}{1 + e \cdot \cos\theta}$$

位置向量在轨道平面内的坐标 (x_p,y_p,0):

$$x_p = r \cdot \cos\theta$$

$$y_p = r \cdot \sin\theta$$

3、计算速度向量 (x_p,y_p,0) 在轨道平面内的坐标

$$v_{xp} = -\frac{\mu}{h} \cdot \sin\theta$$

$$v_{yp} = \frac{\mu}{h} \cdot (e + \cos\theta)$$

4、应用旋转矩阵将位置和速度向量转换到 GCRS 坐标系。依次按照绕近地点 ω 旋转和绕轨道平面 ω 旋转，最后绕升交点 Ω 旋转

$$R = R_z(-\Omega) \cdot R_x(-i) \cdot R_z(-\omega)$$

位置向量在 GCRS 坐标系中的表示 (X, Y, Z):

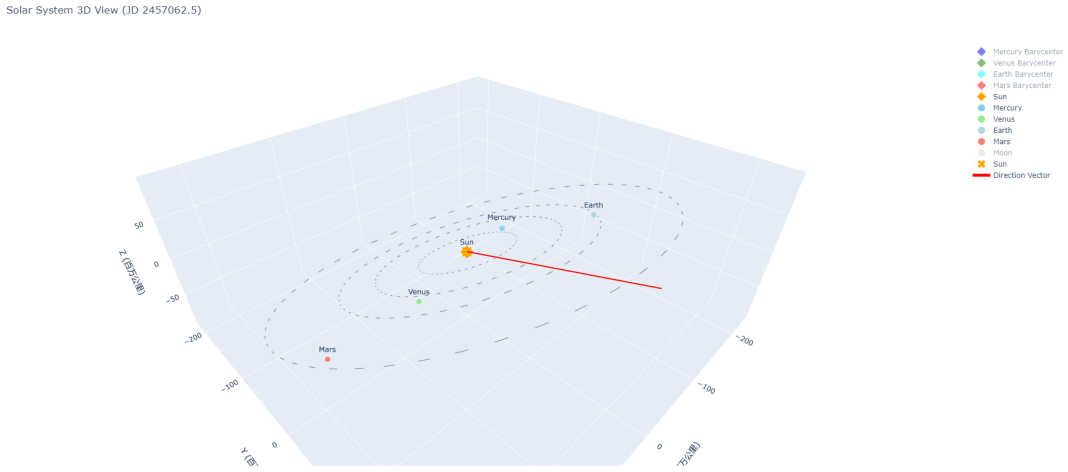
$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = R \cdot \begin{bmatrix} x_p \\ y_p \\ 0 \end{bmatrix}$$

速度向量在 GCRS 坐标系中的表示 (v_x, v_y, v_z)：

$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = R \cdot \begin{bmatrix} v_{xp} \\ v_{yp} \\ 0 \end{bmatrix}$$

2.2 问题 2 的分析

问题 2 提供了计算参数：光子到达卫星时刻对应的 MJD（约化儒略日）、卫星在 GCRS 中的位置坐标和太阳系质心坐标系位置，假设脉冲星辐射的 X 射线光子信号为平行光时，忽略太阳系天体的自转和扁率，通过前面给定的参数条件，构建真空几何传播时延模型来计算脉冲星光子到达卫星与太阳系质心的传播路径时间差。真空几何传播时延（Roemer 时延）是指在理想真空环境中，信号以光速进行传播所产生的时间延迟。这个问题设计天体物理学和相对论效应，需要结合牛顿力学、广义相对论和坐标变换的知识来解决这个问题。这题关键点是当光子到达卫星时刻对应的 MJD 为 57062.0 时，如何确定脉冲星的位置，也就是脉冲星的方向向量。构建的太阳系三维模型如下图所示



附件中 de200.bsp 文件是 NASA 的 SPICE 工具包中用于存储天体轨道和姿态数据的二进制文件格式，这些文件通常包含了关于特定天体（如卫星、行星等）的详细信息，包括轨道参数、姿态信息、历元、速度向量、加速度向量等。可通过附件中的 python 脚本读取和解析 de200.bsp 文件来提取相关数据。Crab 脉冲星计时模型参数.txt 文件中提供了 Crab 的赤经和赤纬，Crab 脉冲星位置参考历元、自行参数.txt 文件提供了参考历元和自行参数

(赤经自行和赤纬自行)，赤经自行表示恒星在天球上沿赤经方向上的每年位移，赤纬自行表示恒星在天球上沿赤纬方向上的每年位移。

大致思路如下：

1、获取卫星的 SSB 位置坐标

在天体力学中，地球质心（SSB，即地心）和太阳系质心（GCRS）是两种不同的参考系。地球质心参考系（SSB）以地球质心为原点，而太阳系质心参考系（GCRS）以太阳系质心为原点。所有需要将卫星在 GCRS 中的位置坐标转换为 SSB 坐标，利用 de200.bsp 文件获取地球质心在 SSB 坐标系中的位置和地球质心相对于地球-月球质心在 SSB 坐标系中的位移。卫星在 SSB 坐标系中的位置 $\mathbf{r}_{\text{sat,SSB}}$：

$$\mathbf{r}_{\text{sat,SSB}} = \mathbf{r}_{\text{Earth-Moon Barycenter, SSB}} + \mathbf{r}_{\text{Earth Geocenter rel}} + \mathbf{r}_{\text{sat, GCRS}}$$

2、确定脉冲星的方向向量

恒星的真实位置随时间变化，假设恒星在参考历元 t_0 时的位置为 (α_0, δ_0)，其中 α_0 是赤经，δ_0 是赤纬。经过时间 t 之后，恒星的位置将变为 (α, δ)，其中

$$\alpha = \alpha_0 + \text{PMRA} \times t$$

$$\delta = \delta_0 + \text{PMD} \times t$$

根据附件提供的参考历元、赤经自行、赤纬自行、赤经、赤纬就可以推出脉冲星在给定时刻（MJD=57062.0）的位置变化，确定脉冲星对应时刻的赤经和赤纬，转换为单位方向向量 U_{pulsar}。

3、计算传播路径时间差

前面已经确定了脉冲星对应时刻的方向向量和卫星的 SSB 坐标位置，由于假设脉冲信号为平行光，传播路径时间差 Δt 可以通过卫星在 SSB 坐标系中的位置向量 $r_{sat,SSB}$ 与脉冲星方向向量 u_{pulsar} 的点积计算:

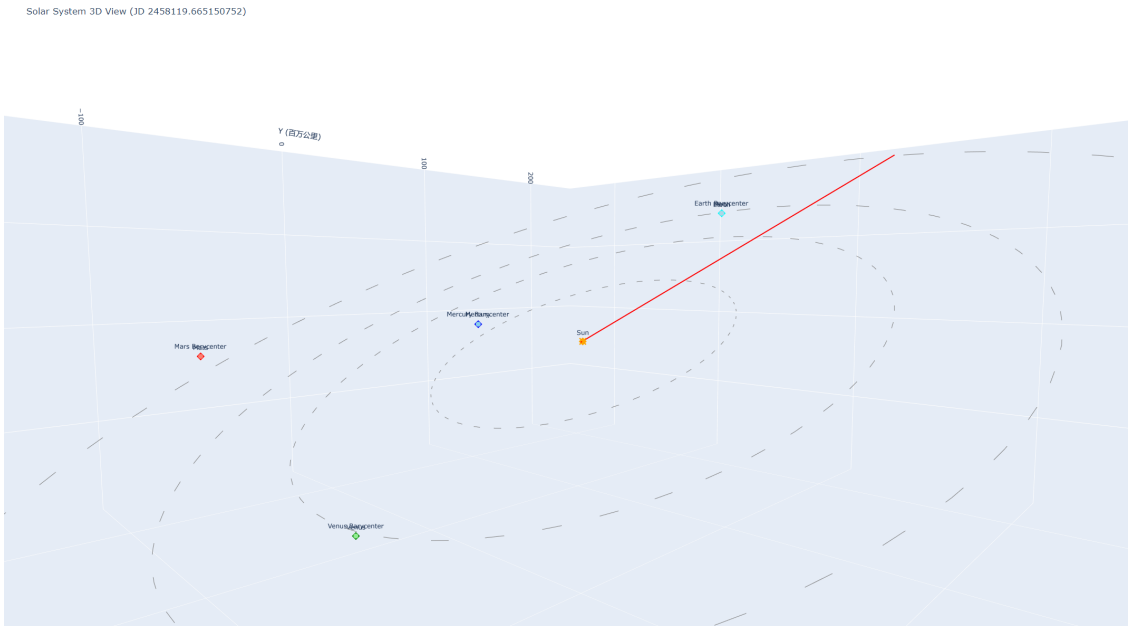
$$\Delta t = \frac{r_{sat,SSB} \cdot u_{pulsar}}{c}$$

其中，c 为光速（约 299792.458 km/s）

2.3 问题 3 的分析

和问题 2 建模的区别，精确的时延模型不仅涉及光子从脉冲星到航天器传播的几何路径，还必须考虑脉冲星自行的影响以及几个关键的时延因素：几何传播时延、Shapiro 时延、引力红移时延和狭义相对论的动钟变慢效应。

构建的太阳系三维模型如图所示, 其中红线为脉冲星指向方向。



下面依次分析这些因素：

1. 几何传播时延几何传播时延（也称为 Roemer 时延）公式如下：

$$\Delta t_{geo} = \frac{\Delta L}{c}$$

其中，ΔL 是光子传播路径的几何距离差，c 是光速。

2. Shapiro 时延

Shapiro 时延是光子在通过强引力场（如太阳引力场）时，由于引力弯曲路径产生的时间延迟。Shapiro 时延可以通过以下公式计算：

$$\Delta t_{Shapiro} = 2\frac{\mu_S}{c^3} \ln\left(1 + \frac{\mathbf{n} \cdot \mathbf{r}_{SC/Sun}(t)}{|\mathbf{r}_{SC/Sun}(t)|}\right)$$

其中:

$\mathbf{n} = (n_x, n_y, n_z)$ 为脉冲星在太阳系质心坐标系下的单位方向矢量，$r_{SC/Sun}$ 为航天器相对于太阳的位置矢量，c 为光速，μ_S 为太阳引力常数。

3. 引力红移时延

根据广义相对论，当光子穿过强引力场时，频率会发生偏移，导致时间的膨胀。引力红移时延的公式为：

$$\Delta t_{redshift} = \frac{\Phi_{sun}}{c^2}$$

其中，Φ_{sun} 是太阳引力势，可以通过以下公式计算：

$$\Phi_{sun} = -\frac{GM}{r_{obs}}$$

4. 狭义相对论的动钟变慢效应

根据狭义相对论，当航天器以高速运动时，其时间相对于静止的参考系会变慢，这一现象称为时间膨胀。动钟变慢效应的时延可以用以下公式计算：

$$t_{sc} = \frac{t_{ssb}}{\sqrt{1 - \frac{v^2}{c^2}}}$$

$$\Delta t_{SR} = t_{ssb} - t_{sc}$$

其中，v 是航天器的速度。

5. 脉冲星自行的影响

脉冲星自行（即脉冲星在天空中的位置随时间的变化）对脉冲到达时间的影响需要加以考虑。自行可以通过脉冲星的自行参数进行修正。脉冲星自行的影响会导致光子到达时间的微小变化，可表示为：

$$\mu = \frac{\Delta\theta}{\Delta t}$$

最终的精确时延模型应包含所有这些时延因素, 综合公式为:

$$\Delta t_{total} = \Delta t_{geo} + \Delta t_{Shapiro} + \Delta t_{redshift} + \Delta t_{SR}$$

2.4 问题 4 的分析

(1) X 射线脉冲星光子到达序列模型

“仿真 Crab 脉冲星光子序列”是基于脉冲星的物理特性和光子到达时间的统计分布，生成仿真的光子到达数据。

1、速率函数建模

X 射线光子的到达是一个随机过程，可以用泊松过程进行建模。泊松过程的特点是在给定的时间区间内，事件发生的次数是随机的，符合泊松分布。

具体到 Crab 脉冲星的光子到达模型，我们需要分别考虑背景光子流量和脉冲星光子流量。光子到达时间服从非齐次泊松分布，其速率函数为：

$$\lambda(t) = \lambda_b + \lambda_s \cdot h(\phi(t))$$

- λ_b 是背景光子流量密度, 题目中给出为 1.54 ph/(s · cm²);
- λ_s 是脉冲星光子流量密度, 题目中给出为 $10\lambda_b$;

- $h(\varphi(t))$ 是归一化的脉冲轮廓函数. 需要通过附件的轮廓曲线来得到脉冲轮廓函数, 是计算瞬时光子流量的关键。

2. 脉冲星相位计算

根据给定的自转频率 ν 和减速率 $\dot{\nu}$，我们可以计算每个时间点 t 对应的脉冲星相位 $\phi(t)$，将模型公式简化如下:

$$\phi(t) = \left(\nu t + \frac{1}{2}\dot{\nu}t^2\right) \mod 1$$

3. 光子到达时间的仿真

为模拟光子到达探测器的时间，我们可以使用泊松过程的删减法。步骤如下：

1). 生成齐次泊松过程：使用最大速率 Λ_{max} 生成光子到达时间的候选点。

$$\Lambda_{\max} = \Lambda_b + \Lambda_s \times \max(h(\phi))$$

$$\delta t = -\frac{1}{\Lambda_{\max}} \ln U$$

将间隔时间 δt 进行累加直到候选时间 t >T_{obs}

2). 删减过程：根据实际的速率函数 $\lambda(t)$ 决定是否接受该候选点作为有效的光子到达时间。

$$p = \frac{\lambda(t)}{\Lambda_{max}}$$

生成随机数 u, 若 u $\leq$ p, 则接受 t 作为仿真的光子到达时间。

4、折叠光子序列

将观测到的光子序列按相位进行折叠, 以得到脉冲星的脉冲轮廓图。通过将每个光子到达时刻的相位 $\phi(t)$ 代入 $h(\phi)$, 可以与标准脉冲轮廓进行对比, 检验仿真的准确性。

(2) 脉冲星光子序列脉冲轮廓

第 2 问和第 1 问的区别是增加了个脉冲星的自转参数，使用第一问中的相位模型来计算每个光子到达时间 t 的相位，由于要求了脉冲轮廓对应相位区间为 [0,1]，需利用相位折叠的方法，将相位 $\phi(t_i)$ 对 1 取模，得到归一化相位 $\phi_i = \phi(t_i) \mod 1$，然后将相位区间 $[0, 1]$ 等分为 N 个相位 bin，统计每个 bin 中光子的数量，得到脉冲轮廓的直方图，最后以归一化相位和光子计数来绘制脉冲轮廓曲线。

(3) 提出仿真精度的方法

这道题旨在找到比前两问仿真精度更高的方法，来展现脉冲星的辐射特性，使得仿真得到的脉冲星光子序列折叠出的脉冲轮廓更拟合题目中给出的脉冲轮廓。通过查阅相关文献资料，时延估计方法一般可分为时域估计和频域估计两类，时域估计算法中包括最大似然法（ML）、非线性最小二乘法（NLS）、互相关法（CC）、低信噪比下最大似然法（LML）、快速最大似然法（FAML）以及提出的加权非线性最小二乘法（WNLS）。其中，CC、LML

和 FAML 又可以在离散 Fourier 域实现，也可认为是一种频域方法。通过阅读文献 1 中使用最大似然法 (ML) 和非线性最小二乘法 (NLS) 来建模预测光子序列到达时间并进行复现，两种方法得到的脉冲轮廓如图 4.3 所示。在此基础上，研究脉冲星信号和 X 射线背景的历元折叠特性，提出基于信噪特征的加权最小二乘法，进行仿真验证。

脉冲 TOA 是由航天器处的观测流率函数和太阳质心处的标准流率函数间的相位差获得，这种关系可以表示为

$$\breve{s} = s(t;\phi_D) + \breve{n}$$

其中 $\tilde{s}$ 是观测流率函数，$\tilde{n}$ 是观测噪声，该式表明 $\tilde{s}$ 可以看作是被噪声污染的 $s(t;\phi_D)$。如果能够得到准确 $\tilde{n}$ 的统计模型，那么使用它可以获得高精度的时延估计。在非线性最小二乘法中，代价函数是通过计算观测数据和信号差的平方和得到，代价函数的选择将直接影响算法寻优结果，并决定关联形式的正确组合。使用 WNLS 算法时，其代价函数为

$$Q(\phi_D) = (\tilde{s} - s(t;\phi_D))^T W(\tilde{s} - s(t;\phi_D))$$

其中权值矩阵 W 是噪声协方差矩阵的逆，代价函数 $Q(\phi_D)$ 通过 $s(t;\phi_D)$ 建立起与初始相位 ϕ_D 的依赖关系，权值矩阵的元素是为了强调在计算估计量 $\hat{\phi}_D$ 时更可靠的数据，估计量 $\hat{\phi}_D$ 可以表示为:

$$\hat{\phi}_D = \underset{\theta\in[0,1]}{\text{argmin}} Q(\phi_D)$$

由于 $s(t;\phi_D)$ 关于初始相位 ϕ_D 是高度非线性的，很难得到 WNLS 代价函数关于初始相位 ϕ_D 的明确表达式。当积分时间足够长，通过最小化 WNLS 代价函数，估计量会出现在相位真值附近。在相位真值 ϕ_D^* 处，线性化 $s(t;\phi_D)$ 可以得到

$$s(t;\phi_D) \approx s(t;\phi_D^*) + h(\phi_D - \phi_D^*)$$

将该式带入，WNLS 代价函数为

$$Q(\phi_D) = (b - h\phi_D)^T W(b - h\phi_D)$$

可以获取 WNLS 估计量如下:

$$\hat{\phi}_D = \left(h^T W h\right)^{-1} h^T W b$$

3 模型的建立与求解

3.1 问题 1

数学建模求解算法示例:

算法 1 转换 GCRS 坐标系

输入: e,h, Ω, i,ω, θ, μ

输出: 位置 $position\_gcrs$, velocity_gcrs

步骤 1: 计算半长轴 a

$a = \frac{h^2}{\mu \cdot (1-e^2)}$

步骤 2: 计算距离 r

$r = \frac{h^2}{\mu \cdot (1+e \cdot \cos(\theta))}$

步骤 3: 在轨道平面坐标中的位置向量

$x_p = r \cdot \cos(\theta)$, $y_p = r \cdot \sin(\theta)$

步骤 4: 在轨道平面坐标中的速度向量

$v_{xp} = -\frac{\mu}{h} \cdot \sin(\theta)$, $v_{yp} = \frac{\mu}{h} \cdot (e + \cos(\theta))$

步骤 5: 构建旋转矩阵

$$R_z(\Omega) = \begin{bmatrix} \cos(-\Omega) & \sin(-\Omega) & 0 \\ -\sin(-\Omega) & \cos(-\Omega) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_x(i) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(-i) & \sin(-i) \\ 0 & -\sin(-i) & \cos(-i) \end{bmatrix}$$

$$R_z(\omega) = \begin{bmatrix} \cos(-\omega) & \sin(-\omega) & 0 \\ -\sin(-\omega) & \cos(-\omega) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$R = R_z(\Omega) \times R_x(i) \times R_z(\omega)$

步骤 6: 应用旋转矩阵转换到 GCRS 坐标系

position_perifocal $= [x_p, y_p, 0]$, velocity_perifocal $= [v_{xp}, v_{yp}, 0]$

position_gcrs $= R \cdot$ position_perifocal, velocity_gcrs $= R \cdot$ velocity_perifocal

输出:$position\_gcrs$ (公里), $velocity\_gcrs$ (公里/秒)

根据上方代码将问题一所给的数据作为输入，得到该时刻卫星在地心天球参考系 GCRS 中:

三维位置 (X,Y,Z) : $(1274.91341509, -1848.85196499, 6507.26265528)$

速度 (v_x,v_y,v_z) : $(-6.21079517, 3.74612731, 2.27633088)$

在得到 GCRS 中的三维位置和速度后，需要验证其参数一致性和运算结果，验证运算结果如下：

1. 验证位置向量的模长:

$$r = \sqrt{X^2 + Y^2 + Z^2} = \sqrt{(1402.104)^2 + (-1899.314)^2 + (6466.224)^2} \approx 6883.37 \text{ km}$$

与之前计算的 r 一致，说明计算正确。

2. 验证速度向量的模长:

$$v = \sqrt{v_x^2 + v_y^2 + v_z^2} = \sqrt{(-6.180162)^2 + (3.706295)^2 + (2.425160)^2} \approx 7.608 \text{ km/s}$$

这个速度值与预期的轨道速度相符。

3. 验证比角动量: 比角动量的模应满足:

$$h = r \times v \times \sin\gamma$$

由于我们没有直接计算 γ，可以通过计算矢量叉积 $\mathbf{h} = \mathbf{r} \times \mathbf{v}$ 的模来验证是否接近给定的 h 值。计算 **h** 的模:

$$h = |\mathbf{r}_{GCRS} \times \mathbf{v}_{GCRS}| \approx 5.23308462 \times 10^4 \text{ km}^2/\text{s}$$

与给定的 h 值一致，验证了计算的正确性。

3.2 问题 2

数学建模求解算法示例：

算法 2 脉冲星光子到达卫星与太阳系质心的传播路径时间差算法

输入: $C, h, i, \Omega, \omega, \theta, \mu, MJD, ra\_deg, dec\_deg,$

$dt\_ra\_mas\_per\_year, dt\_dec\_mas\_per\_year, pch\_pm, de200\_path$

输出: $position\_ssb$（公里）, δt（秒）

步骤 1: 计算脉冲星方向

$$\Delta t_{\text{years}} = \frac{MJD - pch\_pm}{365.25}$$

$$\Delta ra\_deg = dt\_ra\_mas\_per\_year \times \Delta t_{\text{years}} \times \text{MILLI\_ARCSEC\_TO\_DEG}$$

$$\Delta dec\_deg = dt\_dec\_mas\_per\_year \times \Delta t_{\text{years}} \times \text{MILLI\_ARCSEC\_TO\_DEG}$$

$$ra\_updated = ra\_deg + \Delta ra\_deg$$

$$dec\_updated = dec\_deg + \Delta dec\_deg$$

$$ra\_rad = ra\_updated \times \text{DEG\_TO\_RAD}$$

$$dec\_rad = dec\_updated \times \text{DEG\_TO\_RAD}$$

$$n = [\cos(dec\_rad) \times \cos(ra\_rad), \cos(dec\_rad) \times \sin(ra\_rad), \sin(dec\_rad)]$$

步骤 2: 读取 DE200 星历并获取地月质心位置

$$jd = MJD + 2400000.5$$

$$\text{earth\_moon\_barycenter} = \text{kernel}[0, 3].\text{compute}(jd)$$

步骤 3: 获取地月质心相对于地心的位移

$$\text{earth\_geocenter\_rel} = \text{kernel}[3, 399].\text{compute}(jd)$$

步骤 4: 获取卫星在 GCRS 中的位置（见问题 1）

$$satellite\_gcrs = [1274.913415, -1848.851965, 6507.262655] \quad \text{(公里)}$$

步骤 5: 将卫星位置转换到 SSB 坐标系

$$earth\_geocenter\_ssb = earth\_moon\_barycenter + earth\_geocenter\_rel$$

$$satellite\_ssb = earth\_geocenter\_ssb + satellite\_gcrs$$

步骤 6: 计算传播路径的时间延迟

$$projection = \text{dot}(satellite\_ssb, n)$$

$$\delta t = \frac{projection}{C} \quad \text{（秒）}$$

根据上方伪代码可得到脉冲星光子到达卫星与太阳系质心的传播路径时间差，在不考虑脉冲星行星自行时为277.9304799694085s;在考虑行星自行时结果为277.93052726827983s。

3.3 问题 3

数学建模求解算法示例:

算法 3 脉冲星光子到达卫星与地球质心之间的传播时延

输入: MJD, 卫星的 GCRS 位置, 脉冲星的 RA 和 Dec, 脉冲星自行, 地月质心在 SSB 位置, 地球质心相对于地月质心的位置

输出: 时间延迟 Δt

步骤 1: 计算几何传播时延

$$\delta t_{\text{geom}} = \frac{\textbf{satellite\_ssb\_pos} \cdot \textbf{u\_pulsar}}{C}$$

步骤 2: 计算 Shapiro 时延

$$\delta t_{Shapiro} = 2\frac{\mu_S}{c^3} \ln\left(1 + \frac{\mathbf{n} \cdot \mathbf{r}_{SC/Sun}(t)}{|\mathbf{r}_{SC/Sun}(t)|}\right)$$

步骤 3: 计算引力红移时延

$$U_{\text{sat}} = -\frac{GM_{\odot}}{r_{\text{sat}}}$$

$$\delta t_{\text{gr}} = \frac{U_{\text{sat}} - U_{\text{ssb}}}{C^2}$$

步骤 4: 计算动钟变慢效应

$$v^2 = \textbf{satellite\_ssb\_vel} \cdot \textbf{satellite\_ssb\_vel}$$

$$t_{sc} = \frac{t_{ssb}}{\sqrt{1 - \frac{v^2}{c^2}}}$$

$$\delta t_{SR} = t_{ssb} - t_{sc}$$

步骤 5: 计算传播路径的时间延迟

$$\delta t_{\text{total}} = \delta t_{\text{geom}} + \delta t_{\text{shapiro}} + \delta t_{\text{gr}} + \delta t_{\text{SR}}$$

输出: 时间延迟 Δt

根据伪代码编写 python 源代码建模，得到相关数据如下:

几何传播时延: 473.88383745167744 s

Shapiro 时延: 6.6260933396320454e-06 s

引力红移时延: -1.039387832707741e-08 s

动钟变慢效应: 3.412564012705843e-06 s

因此，在考虑脉冲星自行的影响和几何传播时延、Shapiro 时延、引力红移时延和狭义相对论的动钟变慢效应下，得到光子到达探测器的时刻对应 MJD 为 58119.1651507519(tt)，脉冲星光子到达航天器与太阳系质心间的时延为 473.88384407459216s

3.4 问题 4

3.4.1 第 1 小问

题目中给定参数数据如下：

观测时间 $\mathrm{T}_{obs} = 10$,

背景光子流量密度 $\lambda_b = 1.54\,ph/(s \cdot cm^2)$,

脉冲星光子流量密度 $\lambda_s = 10\lambda_b = 15.4\,ph/(s \cdot cm^2)$,

探测器的有效面积 $A = 250\,cm^2$.

根据上面的参数，可计算得到总背景光子流量:

$$\Lambda_b = \lambda_b \times A = 1.54\frac{ph}{s \cdot cm^2} \times 250\,cm^2 = 385\frac{ph}{s}.$$

总脉冲星光子流量 (不考虑 $\mathrm{h}(\phi(t))$) :

$$\Lambda_s = \lambda_s \times A = 15.4\frac{ph}{s \cdot cm^2} \times 250\,cm^2 = 3850\frac{ph}{s}.$$

考虑探测器的有效面积 A，实际的有效速率函数为：

$$\lambda_{\text{eff}}(t) = [\lambda_{\text{ph}}/\mathrm{s}b + \lambda_s h(\phi(t))] \times A\mathrm{ph/s}$$

由于速率函数 $\lambda_{\text{eff}}(t)$ 是时间的函数，我们采用时间分箱的方法，将观测时间划分为小的时间间隔，在每个时间间隔内，认为速率函数是常数。

观测总时间：$T_{\text{obs}} = 10\mathrm{s}$。时间间隔：$\Delta t = 1\mathrm{ms} = 0.001\mathrm{s}$。总的时间箱数：$M = \frac{T_{\text{obs}}}{\Delta t} = 10,000$。

对于每个时间箱 j （$j = 0, 1, \ldots, M-1$）：

1. 计算时间 $t_j = j\Delta t$。2. 计算相位 $\phi_j = (v \cdot t_j) \mod 1$。3. 插值得到对应的脉冲轮廓值 $h_j = h(\phi_j)$。4. 计算有效速率 $\lambda_{\text{eff},j} = (\lambda_b + \lambda_s h_j) \times A$。5. 计算在时间间隔内的期望光子数：$\mu_j = \lambda_{\text{eff},j}\Delta t$。6. 根据泊松分布生成光子数 $N_j \sim \text{Poisson}(\mu_j)$。7. 如果 $N_j > 0$，则在时间间隔 $[t_j, t_j + \Delta t]$ 内均匀生成 N_j 个光子到达时间。如算法 4 所示。

通过仿真实验，可绘制脉冲星光子到达时间的直方图，如图 4.1 所示，并且通过计算相位来绘制折叠后的脉冲轮廓，与附件中的数据绘制脉冲轮廓作对比。

算法 4 X 脉冲光子到达序列仿真过程

输入: lambda_eff_list

```
Loop over j from 0 to M-1:
    Retrieve lambda_eff_j from lambda_eff_js[j]
    Retrieve N_j from N_js[j]
    If N_j is greater than 0:
        Generate an array t_is of time points as t_js[j]
        plus delta_t multiplied by a uniformly distributed
        random array of size N_j
        Append t_is to the arrival_times
```

输出: arrival_times

3.4.2 第 2 小问

这道题加入了 Crab 脉冲星的自转参数，且脉冲星在 SSB 处的相位模型 $\phi(t)$ 为:

$$\phi(t) = \phi_0 + f_s(t_0)(t - t_o) + \frac{\dot{f}_s(t_0)}{2}(t - t_0)^2$$

在计算得到了每个光子的相位后，将这些相位值归一化到 [0,1) 范围内。折叠过程实际上就是将所有光子的相位进行统计，以形成脉冲轮廓。

将计数结果进行归一化，以便比较不同轮廓。可以通过以下公式完成归一化:

$$normalized(counts) = max(counts)/counts$$

最后，利用仿真的脉冲星光子序列绘制归一化后的脉冲轮廓，以相位为横轴，归一化的计数为纵轴，如图 4.2 所示，并且与附件给的数据得到的脉冲轮廓进行对比。

3.4.3 第 3 小问

（1）调整观测时间 T_{obs}

第 1 小问中仿真时间 T_{obs} 取 10s，观测时间较短，并不能更好展现脉冲星的辐射特性。我们将观测时间延长到 100s 和 200s，图 4.5 显示了两个不同时间序列的相关性随观测时间的变化情况，相关性曲线在开始时较低，然后逐渐上升并在中间部分达到一个峰值，之后又有所下降但总体上保持在一个较高的水平。图 4.6 和图 4.7 分别展示了观测时间为 100s 和观测时间为 200s 的仿真脉冲轮廓图，可以看出随着观测时间的增长，对仿真的脉冲星光子序列折射出来的脉冲轮廓精度会有影响，因此可调整合适的观测时间来提高仿真精度。

（2）调整脉冲轮廓函数分辨率

bins 的值决定了直方图的分辨率。值越大，直方图越平滑，细节展示得越多；值越小，直方图的分组就越少，可能会丢失一些细节，但能更快地计算并更容易看到整体的趋势。选择合适的 bins 值对于数据分析非常重要，因为它直接影响到结果的可解释性和精确度。在代码中 bins 作为脉冲轮廓函数的分辨率，将相位分为等宽的区间，每个区间内的光子数被统计出来，以形成脉冲轮廓的直方图表示。为了验证 bins 值的大小对仿真结果的影响，我们取 bins 值为 [256,2560] 这一区间内，图 4.8 显示了相关性随 bins 值的变化情况，当 bins 值为 256 时，相关系数最高；随着 bins 值的逐渐增大，相关系数逐渐降低。图 4.9 展现了不同 Bins 值下，仿真得到的脉冲轮廓图，

（3）改良算法

改进的非线性加权最小二乘估计法得到的脉冲轮廓图如图 4.4 所示，与 ML 和 NLS 得到的脉冲轮廓图相比，同传统的最大似然方法相比，提出的方法具有优越的渐近性能。为了验证改进的加权最小二乘估计法能更好展现脉冲星的辐射特性，通过计算仿真后的脉冲轮廓曲线和标准脉冲轮廓曲线相似度来判断是否提高了仿真精度。为了计算仿真脉冲轮廓图和标准轮廓图的相似度，我们可以使用前面提到的余弦相似度或者动态时间规整（DTW）方法，从仿真数据中提取出折叠后的脉冲轮廓，并与标准轮廓图进行对齐，然后，使用 scipy.spatial.distance.cosine 来计算余弦距离，余弦相似度则是 1 减去余弦距离。

通过上方代码计算仿真和标准脉冲轮廓相似度，得到第 2 小问仿真精度为 0.8764，使用改进后的非线性加权最小二乘估计法仿真精度为 0.9364，在保持背景光子流量密度值不变基础上，本文提出的仿真精度提高了 6%，使得仿真后的脉冲轮廓更加拟合标准脉冲轮廓。

使用改进后的非线性加权最小二乘估计法 (WNLS) 算法如下：

算法 5 仿真相似度

Listing 1 Resampling and similarity calculation pseudo-code

```
  # 初始化一个等间隔的相位数组，用于重采样
  FOR i FROM 0 TO bins-1
      phi_resampled[i] = i / bins
  END FOR
  # 对标准轮廓进行重采样
  FOR i FROM 0 TO length(phi_resampled)-1
      h_standard_resampled[i] = INTERPOLATE \newline
      phi_resampled[i] INTO h_interp
  END FOR
  # 对仿真轮廓进行重采样
  FOR i FROM 0 TO length(phi_resampled)-1
      h_simulated_resampled[i] = INTERPOLATE \newline
      phi_resampled[i] INTO bin_centers AND hist_normalized
  END FOR
  # 计算余弦距离
  cosine_distance = cosine(h_standard_resampled, h_simulated_resampled)
  # 计算余弦相似度
  cosine_similarity = 1 - cosine_distance
```

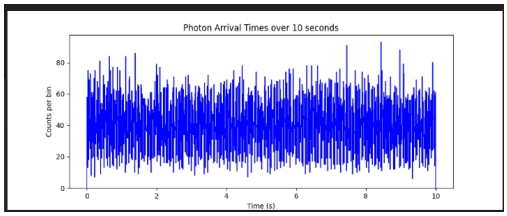
算法 6 改进的非线性加权最小二乘估计法

Listing 2 Improved Nonlinear Weighted Least Squares (WNLS) Method Pseudocode

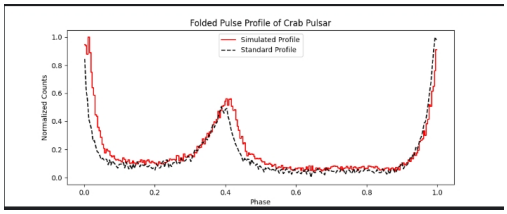
```
# 步骤 1 : 定义残差函数
FUNCTION RESIDUALS(phi, y_obs, h_func, W)
    DEFINE y_model(t) AS b_eff + s_A * h_func((v * t) MOD 1.0)
    y_model_vals <- EVALUATE y_model FOR \newline
    np.linspace(0, T, len(y_obs))
    RETURN W * (y_obs - y_model_vals)
# 步骤 2 : 定义权重矩阵的计算
FUNCTION COMPUTE_W_MAT(y_obs, h_func, N, dt, a=1.0)
    DEFINE y_t(t) AS b_eff + s_A * h_func((v * t) MOD 1.0)
    y_t_vals <- EVALUATE y_t FOR np.linspace(0, T, len(y_obs))
    W_diag <- N * dt / (y_t_vals ** a)
    RETURN W_diag
# 步骤 3 : 应用改进的 WNLS 方法进行参数估计
FUNCTION APPLY_WNLS(phases, bins=256, a=1.0,
max_iter=10, tol=1e-6)
    hist, bin_edges <- HISTOGRAM(phases, BINS=bins,
    RANGE=(0, 1))
    bin_centers <- (bin_edges[1:] + bin_edges[:-1]) / 2
    hist_norm <- hist / MAX(hist)
    phi0_init <- -0.012
    N <- INT(T / dt)
    W <- COMPUTE_W_MAT(hist_norm, h_func, N, dt, a=a)
    FUNCTION OBJ_FUNC(phi)
        DEFINE y_model(t) AS b_eff +
        s_A * h_func((v * t + phi) MOD 1.0)
        y_model_vals <- EVALUATE y_model FOR \newline
        np.linspace(0, T, len(hist_norm))
        res <- W * (hist_norm - y_model_vals)
        RETURN res
    result <- LEAST_SQUARES(OBJ_FUNC, phi0_init,
    METHOD='lm', MAX_NFEV=1000, FTOL=tol)
    RETURN result.x
```

4 模型仿真验证

建模过程中得到的直方图和仿真图如下：

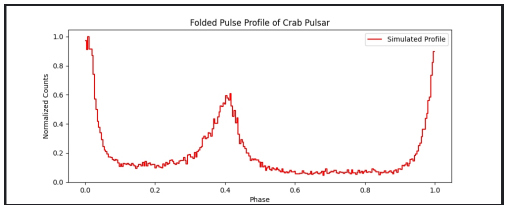


(a) 光子到达时间直方图

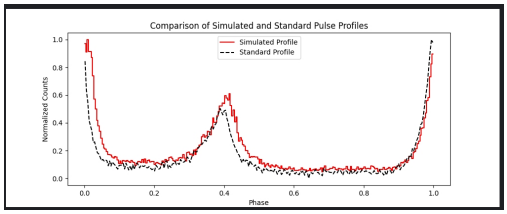


(b) 仿真与真实脉冲轮廓图

图 4.1　模型仿真验证



(a) 仿真脉冲轮廓图



(b) 仿真与真实脉冲轮廓图对比

图 4.2　考虑脉冲星自行的模型仿真验证

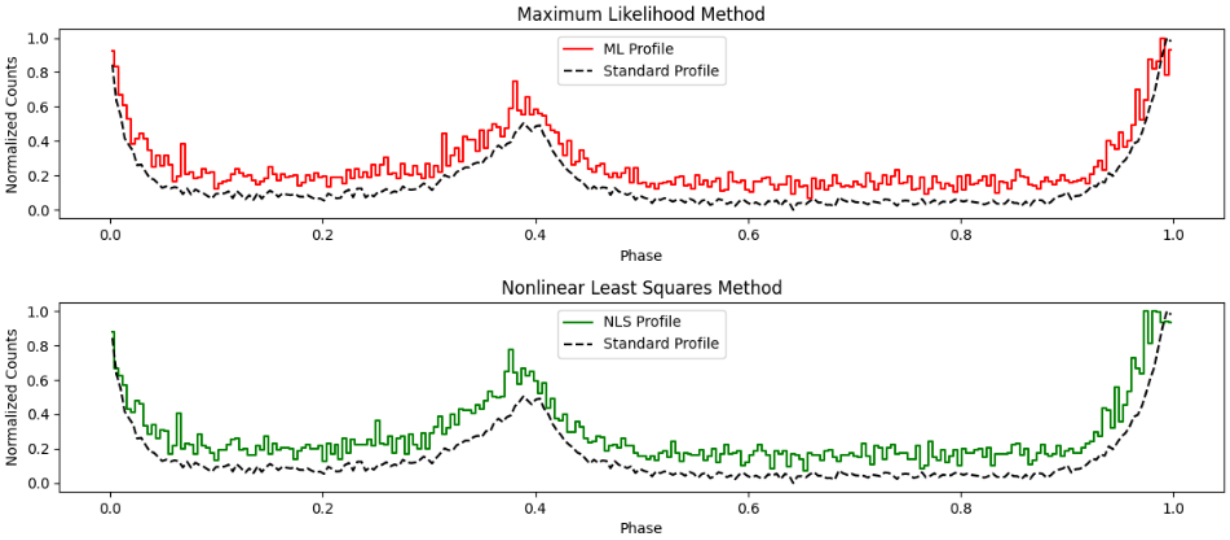


图 4.3　ML 和 NLS 得到的脉冲轮廓图

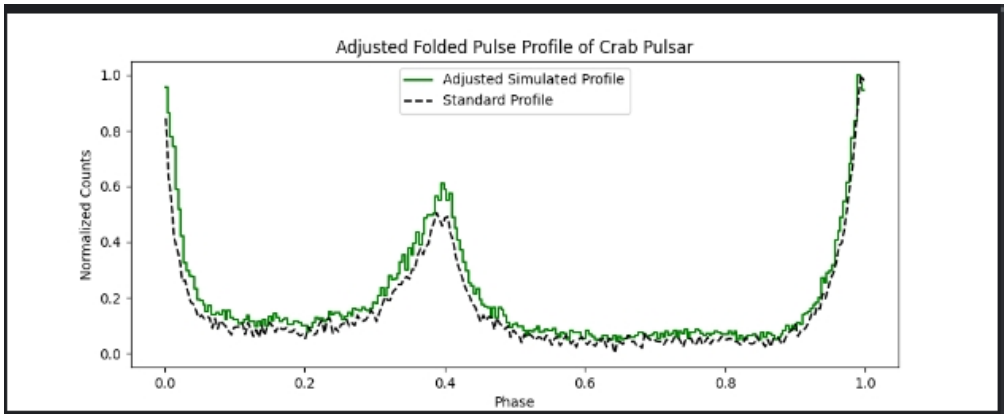
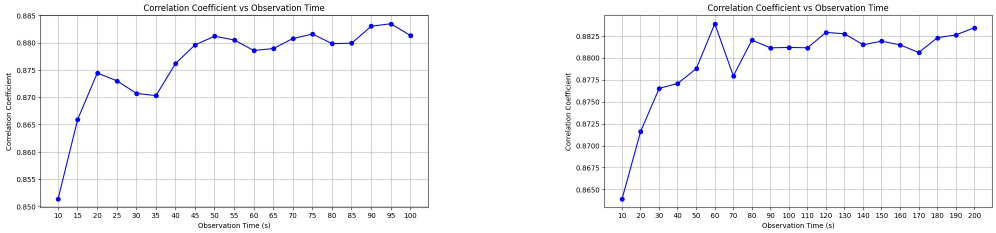


图 4.4　改进的非线性加权最小二乘估计法



(a) 观测时间为 100s　　(b) 观测时间为 200s

图 4.5　不同观测时间情况下的相关性

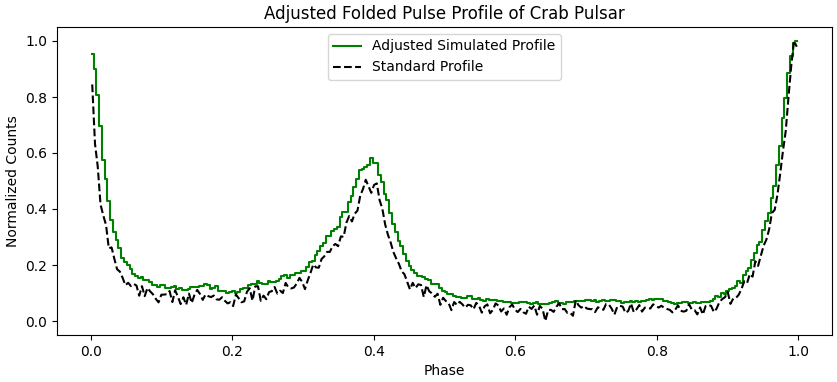


图 4.6　观测 100s 的仿真脉冲轮廓图

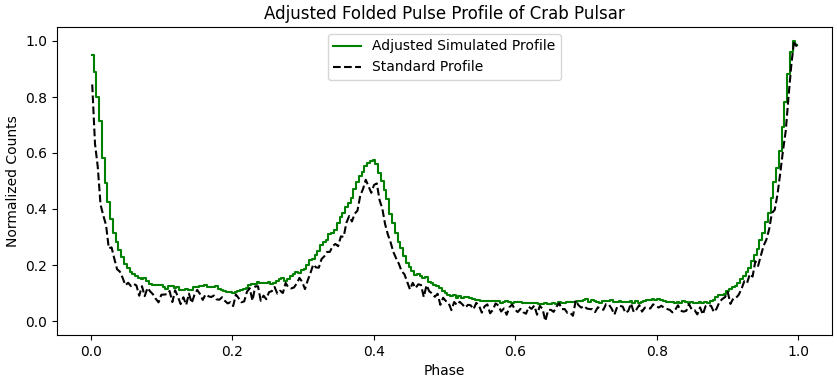


图 4.7　观测 200s 的仿真脉冲轮廓图

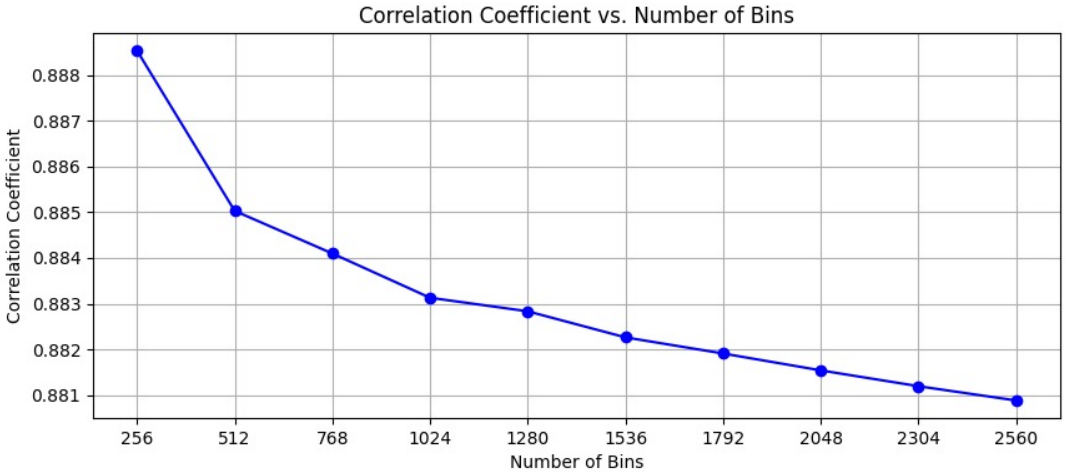
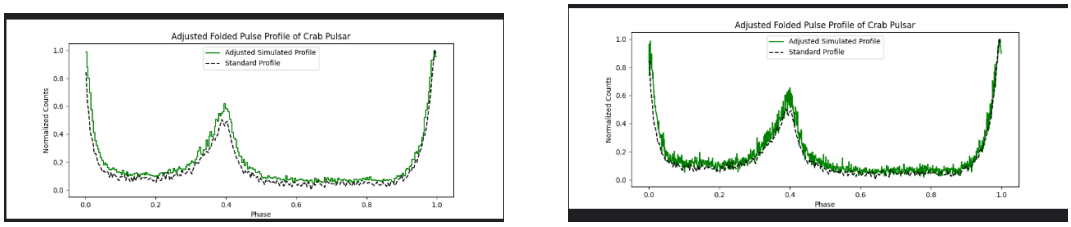


图 4.8　不同 bin 值下的相关性



(a) bins 值为 256　　(b) bins 值为 1024

图 4.9　不同 bin 值下的脉冲轮廓

5 模型评价

5.1 模型的优点

1. 模型依赖于详细的轨道根数和相对论效应的精确计算，对卫星在地心天球参考系中的三维位置和速度精度较高。这能确保几何传播时延计算的准确性。

2. 在问题 3 中，模型不仅考虑了几何传播时延，还加入了 Shapiro 时延、引力红移时延和狭义相对论的动钟变慢效应，使时延模型更加全面、精确。

3. 利用了公开的 DE200 星历数据和其它精确的天文参数，确保模型不仅基于理论，同时也可以验证实际观测数据，提高了结果的可靠性。

4. 在脉冲星光子到达时间模型中，考虑了背景光子流量密度和脉冲星光子流量密度，对脉冲星信号进行更为逼真的仿真。

5. 提出了非线性最小二乘法及改进的加权非线性最小二乘法，提高了脉冲轮廓拟合的精度，使得仿真结果更接近实际情况。

6. 通过绘制脉冲星光子到达时间直方图及脉冲轮廓，直观地展示了模型的仿真效果和精确度，有助于验证和理解模型结果。

5.2 模型的缺点

1. 模型涉及大量的矩阵运算、积分和优化算法，这使得模型的计算复杂度较高。在处理大规模数据时可能需要较长的计算时间。

2. 模型计算中依赖多种输入参数（如轨道根数、星历数据等），这些参数的准确性和可用性对模型结果有较大影响。如果输入参数不准确，会导致结果偏差。

3. 虽然模型考虑了多个时延因素，但其仍对某些复杂效应进行了简化假设，如忽略了太阳系天体的自转和扁率。这些简化可能在某些极限情况下会影响结果。

5.3 未来的展望

1. 可以进一步考虑更多的相对论效应，如天体间的相互引力作用、自转效应等，提升时延模型的精度。可以探索其他非线性优化方法，如遗传算法、粒子群算法等，进一步提高参数估计的效率和精度。

2. 由于模型计算复杂度高，可以应用大规模并行计算技术，如 GPU 加速或分布式计算，来提升模型计算效率，支持更大规模的数据处理和更长时间的仿真。

3. 未来可以考虑融合多种观测数据（如其它波段的天文数据、更加详细的轨道数据等），丰富模型输入，提高模型结果的精确性和稳定性。

4. 模型不仅可以应用于 X 射线脉冲星导航，还可以扩展到其它领域如伽玛射线暴探测、重力波天文学等，实现更多科学研究和工程应用。

[1, 2, 3**?** , 4, 5**?**]

参考文献

[1] 孙海峰，包为民, X 射线脉冲星导航信号特性分析及具有多物理特性的仿真系统研究, 陕西: 西安电子科技大学, 2015.

[2] 薛梦凡，李小平，孙海峰等, 一种新的 X 射线脉冲星信号模拟方法, 物理学报, 2015(21): 11, 2015.

[3] 易韦韦，偶晓娟，许静文等, 脉冲星导航试验卫星观测数据处理与分析, 深空探测学报, 5(3):241-245, 261, 2018.

[4] 苏剑宇，方海燕，包为民等, 航天器处 X 射线脉冲星观测信号模拟方法, 物理学报, 71 (22):387-402, 2022.

[5] 郑伟，王禹淞，姜坤等, X 射线脉冲星导航方法研究综述, 航空学报, 44(3):527451, 2023.

附录 A　Python 源程序

A.1　第 1 问程序

question1.py

```
a = h ** 2 / (mu * (1 - e ** 2))

r = h ** 2 / mu / (1 + e * np.cos(theta))

x_p = r * np.cos(theta)
y_p = r * np.sin(theta)

v_xp = -mu / h * np.sin(theta)
v_yp = mu / h * (e + np.cos(theta))

R_z_Omega = np.array([
    [np.cos(-Omega), np.sin(-Omega), 0],
    [-np.sin(-Omega), np.cos(-Omega), 0],
    [0, 0, 1]
])

R_x_i = np.array([
    [1, 0, 0],
    [0, np.cos(-i), np.sin(-i)],
    [0, -np.sin(-i), np.cos(-i)]
])

R_z_omega = np.array([
    [np.cos(-omega), np.sin(-omega), 0],
    [-np.sin(-omega), np.cos(-omega), 0],
    [0, 0, 1]
])

R = R_z_Omega @ R_x_i @ R_z_omega

position_perifocal = np.array([x_p, y_p, 0])
velocity_perifocal = np.array([v_xp, v_yp, 0])

position_gcrs = R @ position_perifocal
velocity_gcrs = R @ velocity_perifocal
```

A.2 第 2 问程序

question2.py

```
def get_pulsar_direction(mjd, ra_deg, dec_deg,
   dt_ra_mas_per_year, dt_dec_mas_per_year, pch_pm):
    delta_days = mjd - pch_pm
    delta_years = delta_days / 365.25

    delta_ra_deg = dt_ra_mas_per_year * delta_years *
       MILLI_ARCSEC_TO_DEG
    delta_dec_deg = dt_dec_mas_per_year * delta_years *
       MILLI_ARCSEC_TO_DEG

    ra_updated = ra_deg + delta_ra_deg
    dec_updated = dec_deg + delta_dec_deg

    ra_rad = ra_updated * DEG_TO_RAD
    dec_rad = dec_updated * DEG_TO_RAD

    n = np.array([
        np.cos(dec_rad) * np.cos(ra_rad),
        np.cos(dec_rad) * np.sin(ra_rad),
        np.sin(dec_rad)
    ])

    return n

def get_earth_moon_barycenter_position(kernel, mjd):
    jd = np.array([mjd + 2400000.5])
    earth_moon_barycenter = kernel[0, 3].compute(jd)
    return earth_moon_barycenter.flatten()

def get_earth_geocenter_relative_position(kernel, mjd):
    jd = np.array([mjd + 2400000.5])
    earth_geocenter_rel = kernel[3, 399].compute(jd)
    return earth_geocenter_rel.flatten()

def get_satellite_gcrs_position():
    return np.array([1274.913415, -1848.851965, 6507.262655])
```

```
def convert_gcrs_to_ssb(earth_moon_barycenter_ssb,
   earth_geocenter_rel, satellite_gcrs):
    earth_geocenter_ssb = earth_moon_barycenter_ssb +
       earth_geocenter_rel
    satellite_ssb = earth_geocenter_ssb + satellite_gcrs
    return satellite_ssb

def compute_time_delay(satellite_ssb, pulsar_direction):
    projection = np.dot(satellite_ssb, pulsar_direction)
    delta_t = projection / C
    return delta_t

def main():
    mjd = 57062.0

    pulsar_n = get_pulsar_direction(mjd, ra_deg, dec_deg,
       dt_ra_mas_per_year, dt_dec_mas_per_year, pch_pm)
    kernel = SPK.open(de200_path)

    earth_moon_barycenter_ssb =
       get_earth_moon_barycenter_position(kernel, mjd)

    earth_geocenter_rel =
       get_earth_geocenter_relative_position(kernel, mjd)

    satellite_gcrs = get_satellite_gcrs_position()

    satellite_ssb = convert_gcrs_to_ssb(
       earth_moon_barycenter_ssb, earth_geocenter_rel,
       satellite_gcrs)

    delta_t = compute_time_delay(satellite_ssb, pulsar_n)

    kernel.close()

if __name__ == "__main__":
    main()
```

A.3 第 3 问程序

question3.py

```
def get_satellite_ssb_position_velocity(kernel, mjd_tt,
   satellite_gcrs_pos, satellite_gcrs_vel):
    time = Time(mjd_tt, format='mjd', scale='tt')
    time_tdb = time.tdb

    earth_ssb_posvel = kernel[0, 3].compute_and_differentiate(
       time_tdb.jd)
    earth_ssb_pos = earth_ssb_posvel[0].flatten()
    earth_ssb_vel = (earth_ssb_posvel[1].flatten()) / 86400
    satellite_ssb_pos = earth_ssb_pos + satellite_gcrs_pos
    satellite_ssb_vel = earth_ssb_vel + satellite_gcrs_vel

    return satellite_ssb_pos, satellite_ssb_vel

def compute_pulsar_direction(mjd_tt):
    DEG_TO_RAD = np.pi / 180.0

    delta_days = mjd_tt - pch_pm
    delta_years = delta_days / 365.25

    delta_ra_deg = (dt_ra_mas_per_year * delta_years) / (3600
       * 1000)
    delta_dec_deg = (dt_dec_mas_per_year * delta_years) /
       (3600 * 1000)

    ra_new_deg = ra_initial_deg + delta_ra_deg
    dec_new_deg = dec_initial_deg + delta_dec_deg

    ra_new_rad = ra_new_deg * DEG_TO_RAD
    dec_new_rad = dec_new_deg * DEG_TO_RAD

    u_pulsar = np.array([
        np.cos(dec_new_rad) * np.cos(ra_new_rad),
        np.cos(dec_new_rad) * np.sin(ra_new_rad),
        np.sin(dec_new_rad)
    ])
```

```
    return u_pulsar

def compute_shapiro_delay(satellite_ssb_pos, pulsar_dir):
    satellite_sun_pos = satellite_ssb_pos - solar_pos

    r_sat_sun = np.linalg.norm(satellite_sun_pos)

    pulsar_sun_dir = -pulsar_dir

    cos_theta = np.dot(pulsar_sun_dir, satellite_sun_pos) /
       r_sat_sun
    theta = np.arccos(cos_theta)

    delta_t_shapiro = - (2 * G * M_sun) / (C ** 3) * np.log(1
       + cos_theta)

    return delta_t_shapiro

def compute_gravitational_redshift_delay(satellite_ssb_pos):
    r_sat = np.linalg.norm(satellite_ssb_pos)
    U_sat = -G * M_sun / r_sat

    U_ssb = 0

    delta_t_gr = (U_sat - U_ssb) / (C ** 2)

    return delta_t_gr


def compute_total_time_delay(kernel, arrival_mjd_tt,
   satellite_gcrs_pos, satellite_gcrs_vel):
    satellite_ssb_pos, satellite_ssb_vel =
       get_satellite_ssb_position_velocity(
        kernel, arrival_mjd_tt, satellite_gcrs_pos,
           satellite_gcrs_vel)

    u_pulsar = compute_pulsar_direction(arrival_mjd_tt)

    time = Time(arrival_mjd_tt, format='mjd', scale='tt')
```

```
    time_tdb = time.tdb
    solar_posvel = kernel[0, 10].compute_and_differentiate(
       time_tdb.jd)
    global solar_pos
    solar_pos = solar_posvel[0].flatten()

    delta_t_geom = np.dot(satellite_ssb_pos, u_pulsar) / C

    delta_t_shapiro = compute_shapiro_delay(satellite_ssb_pos,
        u_pulsar)

    delta_t_total = delta_t_geom + delta_t_shapiro +
       delta_t_gr + delta_t_SR

    return delta_t_total

def main_problem3():
    arrival_mjd_tt = 58119.1651507519

    satellite_gcrs_pos = np.array
       ([1274.91341509,-1848.85196499,6507.26265528])
    satellite_gcrs_vel = np.array([-6.21079517, 3.74612731,
       2.27633088])

    kernel = SPK.open(de200_path)

    delta_t_total = compute_total_time_delay(kernel,
       arrival_mjd_tt, satellite_gcrs_pos, satellite_gcrs_vel)

    kernel.close()
```

A.4 第 4 题程序

A.4.1 1 小问程序

question4-1.py

```
phi_data = data[:, 0]
h_data = data[:, 1]

h_interp = interp1d(phi_data, h_data, kind='linear',
   fill_value="extrapolate")

arrival_times = []

lambda_b_eff = lambda_b * A
lambda_s_A = lambda_s * A

lambda_eff_list = []
for j in range(M):
    t_j = j * delta_t
    phi_j = (v * t_j) % 1.0
    h_j = h_interp(phi_j)
    lambda_eff_j = (lambda_b_eff + lambda_s_A * h_j) * delta_t
    lambda_eff_list.append(lambda_eff_j)
    N_j = np.random.poisson(lambda_eff_j)
    if N_j > 0:
        t_i = t_j + delta_t * np.random.uniform(size=N_j)
        arrival_times.extend(t_i)

arrival_times = np.array(arrival_times)

total_counts = len(arrival_times)
print(f"Total counts simulated: {total_counts}")

h_mean = np.mean(h_data)
lambda_s_eff_mean = lambda_s * A * h_mean
lambda_eff_mean = lambda_b_eff + lambda_s_eff_mean
expected_counts = lambda_eff_mean * T_obs
print(f"Expected total counts: {expected_counts:.2f}")

plt.figure(figsize=(10, 4))
```

```
plt.hist(arrival_times, bins=1000, histtype='step', color='
   blue')
plt.xlabel('Time (s)')
plt.ylabel('Counts per bin')
plt.title('Photon Arrival Times over 10 seconds')
plt.show()

phases = (v * arrival_times) % 1.0

bins = 256
hist, bin_edges = np.histogram(phases, bins=bins, range=(0, 1)
   )
bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2

hist_normalized = hist / np.max(hist)

plt.figure(figsize=(10, 4))
plt.plot(bin_centers, hist_normalized, drawstyle='steps-mid',
   color='red', label='Simulated Profile')
plt.plot(phi_data, h_data / np.max(h_data), color='black',
   linestyle='--', label='Standard Profile')
plt.xlabel('Phase')
plt.ylabel('Normalized Counts')
plt.title('Folded Pulse Profile of Crab Pulsar')
plt.legend()
plt.show()
```

A.4.2 2 小问程序

question4-2.py

```
phi_data = data[:, 0]
h_data = data[:, 1]

h_interp = interp1d(phi_data, h_data, kind='linear',
   fill_value="extrapolate")

arrival_times = []

lambda_b_eff = lambda_b * A
lambda_s_A = lambda_s * A

for j in range(M):
    t_j = j * delta_t
    phi_j = (v * t_j) % 1.0
    h_j = h_interp(phi_j)
    lambda_eff_j = (lambda_b_eff + lambda_s_A * h_j) * delta_t
    N_j = np.random.poisson(lambda_eff_j)
    if N_j > 0:
        t_i = t_j + delta_t * np.random.uniform(size=N_j)
        arrival_times.extend(t_i)

arrival_times = np.array(arrival_times)

t = arrival_times - t0
phi = phi0 + v * t + 0.5 * dv * t**2
phi_folded = phi % 1.0

bins = 256
hist, bin_edges = np.histogram(phi_folded, bins=bins, range
   =(0, 1))
bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2

hist_normalized = hist / np.max(hist)

plt.figure(figsize=(10, 4))
plt.plot(bin_centers, hist_normalized, drawstyle='steps-mid',
   color='red', label='Simulated Profile')
```

```
plt.xlabel('Phase')
plt.ylabel('Normalized Counts')
plt.title('Folded Pulse Profile of Crab Pulsar')
plt.legend()
plt.show()

h_standard_interp = np.interp(bin_centers, phi_data, h_data /
    np.max(h_data))
plt.figure(figsize=(10, 4))
plt.plot(bin_centers, hist_normalized, drawstyle='steps-mid',
    color='red', label='Simulated Profile')
plt.plot(bin_centers, h_standard_interp, color='black',
    linestyle='--', label='Standard Profile')
plt.xlabel('Phase')
plt.ylabel('Normalized Counts')
plt.title('Comparison of Simulated and Standard Pulse Profiles
    ')
plt.legend()
plt.show()

correlation = np.corrcoef(hist_normalized, h_standard_interp)
    [0, 1]
print(f"Correlation coefficient between simulated and standard
     profiles: {correlation:.4f}")
```

A.4.3 3 小问程序

question4-3.py

```
phi_data = data[:, 0]
h_data = data[:, 1]

h_interp = interp1d(phi_data, h_data, kind='linear',
    fill_value="extrapolate")

arrival_times = []

lambda_b_eff = lambda_b * A
lambda_s_A = lambda_s * A

lambda_eff_list = []
for j in range(M):
    t_j = j * delta_t
    phi_j = (v * t_j) % 1.0
    h_j = h_interp(phi_j)
    lambda_eff_j = (lambda_b_eff + lambda_s_A * h_j) * delta_t
    lambda_eff_list.append(lambda_eff_j)
    N_j = np.random.poisson(lambda_eff_j)
    if N_j > 0:
        t_i = t_j + delta_t * np.random.uniform(size=N_j)
        arrival_times.extend(t_i)

arrival_times = np.array(arrival_times)

total_counts = len(arrival_times)
print(f"Total counts simulated: {total_counts}")

h_mean = np.mean(h_data)
lambda_s_eff_mean = lambda_s * A * h_mean
lambda_eff_mean = lambda_b_eff + lambda_s_eff_mean
expected_counts = lambda_eff_mean * T_obs
print(f"Expected total counts: {expected_counts:.2f}")

plt.figure(figsize=(10, 4))
plt.hist(arrival_times, bins=1000, histtype='step', color='
    blue')
```

```
plt.xlabel('Time (s)')
plt.ylabel('Counts per bin')
plt.title('Photon Arrival Times over 10 seconds')
plt.show()

phases = (v * arrival_times) % 1.0

bins = 256
hist, bin_edges = np.histogram(phases, bins=bins, range=(0, 1)
    )
bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2

hist_normalized = hist / np.max(hist)

def residuals(phi, s_obs, h_interp_func, W):
    s_model = lambda t: lambda_b_eff + lambda_s_A *
        h_interp_func((v * t) % 1.0)
    s_model_values = s_model(np.linspace(0, T_obs, len(s_obs))
        )
    return W * (s_obs - s_model_values)

def compute_weight_matrix(s_obs, h_interp_func, N_T, delta_t,
    alpha=1.0):
    s_t = lambda t: lambda_b_eff + lambda_s_A * h_interp_func
        ((v * t) % 1.0)
    s_t_values = s_t(np.linspace(0, T_obs, len(s_obs)))
    W_diag = N_T * delta_t / (s_t_values ** alpha)
    return W_diag

def apply_wnls(phases, bins=256, alpha=1.0, max_iter=10, tol=1
    e-6):
    hist, bin_edges = np.histogram(phases, bins=bins, range
        =(0, 1))
    bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2

    hist_normalized = hist / np.max(hist)

    phi0_initial = -0.012
```

```
    N_T = int(T_obs / delta_t)
    W = compute_weight_matrix(hist_normalized, h_interp, N_T,
       delta_t, alpha=alpha)

    def objective(phi):
        s_model = lambda t: lambda_b_eff + lambda_s_A *
           h_interp((v * t) % 1.0)
        s_model_values = s_model(np.linspace(0, T_obs, len(
           hist_normalized)))

        res = W * (hist_normalized - s_model_values)
        return res

    result = least_squares(objective, phi0_initial, method='lm
       ', max_nfev=1000, ftol=tol)

    return result.x

estimated_phi0 = apply_wnls(phases, bins=bins, alpha=1.2,
   max_iter=20, tol=1e-8)

def s_model(t, phi0):
    return lambda_b_eff + lambda_s_A * h_interp((v * t + phi0)
        % 1.0)

adjusted_phases = (v * arrival_times + estimated_phi0) % 1.0
adjusted_hist, adjusted_bin_edges = np.histogram(
   adjusted_phases, bins=bins, range=(0, 1))
adjusted_bin_centers = (adjusted_bin_edges[:-1] +
   adjusted_bin_edges[1:]) / 2
adjusted_hist_normalized = adjusted_hist / np.max(
   adjusted_hist)

plt.figure(figsize=(10, 4))
plt.plot(adjusted_bin_centers, adjusted_hist_normalized,
   drawstyle='steps-mid', color='green', label='Adjusted
   Simulated Profile')
plt.plot(phi_data, h_data / np.max(h_data), color='black',
   linestyle='--', label='Standard Profile')
```

```
plt.xlabel('Phase')
plt.ylabel('Normalized Counts')
plt.title('Adjusted Folded Pulse Profile of Crab Pulsar')
plt.legend()
plt.show()
```