二维 CT 扫描参数标定及基于滤波反投影算法的扫描图像重建

摘要

本文通过几何分析、数理统计、信号处理、计算机辅助处理数据等方法，建立了存在测量误差情况下二维 CT 扫描参数标定的模型，通过模板标定了系统的扫描线间隔、旋转角度、旋转中心等参量。借助这些标定的参量，我们完成了从扫描数据中重建物体几何形状、位置、吸收率等信息的任务。最后，我们定量分析了原有模型的缺陷，设计了新的标定模型以提高其稳定性与精确度，并对我们的工作进行了整体评价。

对于扫描线间隔的计算，利用数据中无重合的圆条带部分，可求扫描线被圆所截得的弦长。由于相邻扫描线之间间距相同，确定一条扫描线后可利用该间距得到其余扫描线的方程。后利用弦长及垂径定理可得到一个二元线性方程，其中一个变元为待求扫描线间距的平方。由于一个角度下有多条扫描线与圆相交，可以列出一组超定的方程组，后利用二元线性回归即可求得扫描线间距。由于有多个旋转角度，我们对于每个旋转角度都计算一次回归，最后对得到的数据求平均值，即得扫描线间距为 $0.2766\,\mathrm{mm}$。

旋转角度的计算与上述过程类似，利用扫描线被椭圆截得的弦长，可列出一组含其中一条截距以及斜率的非线性方程，对相邻方程进行求解可得到一组斜率值，均值作为当前角度的正切值。在旋转角接近 90° 时会出现数值上的问题，此时选用斜率倒数以及 x 轴截距作为变元即可。求得系统与托盘的初始夹角为 $\theta_0 = 29.6422^\circ$。具体的 180 个旋转角可见附录中表3。

在特定角度下，由线间隔可以精确地确定圆心投影位置，再加上系统旋转后圆心在旋转坐标系中坐标的变化等约束，可构造多个方程组求解旋转坐标系的原点坐标。对所有值取平均即得到旋转中心 $(-9.2383, 6.2663)\mathrm{mm}$，位于托盘中心的略左上方处。

图像的重建问题等价于逆 Radon 变换。一种算法为像素驱动的直接反投影，即计算待求像素在各个旋转角上的投影位置。其最终吸收率为其各个旋转角上投影位置上的吸收值之和。此外，可利用线性插值来将吸收值连续化。由于直接反投影带来图像模糊等问题，我们利用 Fourier 中心切片定理推导出了一种能较精确地还原原始信息的滤波反投影算法，即将各角度下的投影值与 R-L 滤波函数卷积后的值进行直接反投影。实验证明此方法是优秀的，恢复的图像可见附录中图7，题中要求的 10 点吸收率可见附录中表4，详细各点吸收率可见附件。

此外，在测量值有误差的假设下，我们证明了计算出的扫描线间隔的精确度和圆模型的半径成正相关；以及旋转角度的求解结果在扫描线与椭圆的轴接近平行的临界情况下变得不稳定。为此，可利用多模板多次扫描的方法将圆与椭圆的信息解耦合，并且增大圆的半径来提高计算扫描线间隔的精确度。对于旋转角度的不稳定性，可用椭圆在坐标系中不同的放置方式获取多组数据，最后取最小方差的角度值作为真实值。

最后，我们回顾了整个模型，分析其优点和不足之处，并对其将来的应用提出展望。

关键词： 医学影像　图像重建　参数标定　Radon 变换　信号处理

一、 问题重述

CT 是医学、生物领域重要的成像手段，可以有效获取样品的内部结构信息。题目给出了一种典型的二维 CT 系统，平行入射的 X 射线垂直于探测器平面，经过样品吸收后被等距排列的 512 个探测器单元接收。每次扫描过程中，整个发射-接受系统绕着旋转中心逆时针旋转 180 次，得到 180 组接收信息。

由于现实中 CT 系统存在误差，故需要借助已知结构的样品（模板）对系统的参数进行标定，并据此对未知结构的样品进行成像。

本题目要求建立数学模型，解决以下问题：

1. 根据已知的模板几何信息和扫描结果，确定 CT 系统的旋转中心、探测器单元距离与 180 个旋转方向
2. 根据上述标定的信息，从扫描数据中恢复出未知介质的位置、形状、吸收率
3. 分析上述标定的精度和稳定性，设计新模板、建立标定模型以改进这些指标

二、 问题分析

问题 1 要求从已知的几何信息和扫描结果中恢复出系统的一系列参数。根据比尔-朗伯定律可知，在吸收率恒定的情况下，X 射线的吸收值（即给出的测量值）与穿过物体的厚度成正比，故本小题中吸收值也可以作为物体厚度的一种度量尺寸；又由于两条 X 射线之间的间距恒定，故其亦可作为度量单位。再加上题目本身给出的 SI 制与 256×256 两种度量，共有四种尺度。因而解决本题时尤其要注意各个尺度之间的换算和处理。我们使用每个圆中不同扫描线的长度处理得到扫描线宽，最后所有值平均得到真实线宽（即接收器间距）。由于圆直径最长，而数据中属于圆的吸收值部分出现多次明显的最大值，据此可得到圆直径在吸收值尺度下的值，也就得到了吸收值尺度与真实尺度的换算关系。对于不重叠部分，通过扫描线长处理可得到实际圆心的投影坐标。而后，在旋转中心建立另一坐标系，通过椭圆弦长与坐标系旋转的位置关系等几何约束列出方程组，可确定每一组数据对应的旋转角度（即 180 个方向）。再通过坐标系旋转后圆心坐标的变化等约束，可求解圆心在旋转坐标系中的坐标，也就确定了旋转中心相对托盘的位置。

问题 2、3 要求根据问题 1 中得到的参数，从扫描数据中重建介质的二维图像，即重建各点的吸收率信息，这显然是对第一题获得信息的一种检验。在求得旋转中心位置后，很容易在旋转坐标系和静止坐标系间进行变换。最简单的思路是对于每个旋转角度，将投影信息反向投影到原坐标系中；最后所有角度的信息叠加，就能得到大致的原

图像。但显然这种方法的的准确度不够高，会造成各点处强度值的平均化，因此需要通过更精细的处理方法获得更接近的原始图像。

问题 4 要求分析问题 1 中参数标定的精度与稳定性，以及设计新模板、建立新模型，题目要求比较开放。这要求我们在完成问题 1 后，对数据的处理过程进行定量的分析。除了题中要求的接收器间距、转动角度、转动中心位置等重要物理量外，还应该对求解这些量的中间过程和理论推导进行研究。对同一个量多次估计的方差、关于同一量的超定方程组解的情况等都可以成为重要的量化指标。同样地，在设计新模板和新的标定模型时，也应当着重关注这些指标，以尽可能低的成本、尽可能简单的方式，做到精确而稳定地标定系统的各项参数。

三、 模型假设与符号约定

在本题的求解过程中，我们做出以下基本假设:

假设 1 *探测平台的 180 个旋转方向在每次扫描时是不变的。*

假设 2 *X 射线在非被测介质中无损耗传播，不发生衰减、吸收。*

假设 3 *X 射线在任何情况下中不发生散射、反射。*

假设 4 *所有的 X 射线发射器与接收器均没有损坏。*

在建模与求解过程的中，我们使用了表1中定义的一些记号。

四、 模型建立与问题求解

4.1 问题 1

理论支撑

根据物理学知识可知，物体对电磁波的吸收遵循比尔-朗伯定律 *(Beer–Lambert law)*。其内容为：一束单色光照射于一吸收介质表面，在通过一定厚度的介质后，由于介质吸收了一部分光能，透射光的强度就要减弱。吸收介质的浓度愈大，介质的厚度愈大，则光强度的减弱愈显著。具体地，如果将 X 射线某条路径上的介质切分为多个厚度为 w 的微小单元，第 i 个单元的衰减系数为 μ_i，射线强度为 I_i，则我们有:

$$I_i = I_{i-1}\mathrm{e}^{-\mu_i w} \tag{1}$$

当切分数量趋向于无穷大时，求和转化为路径积分，容易得到:

$$\int_l \mu(x,y)\,\mathrm{d}l = \ln\frac{I_0}{I} \tag{2}$$

表 **1**　本文中使用的记号

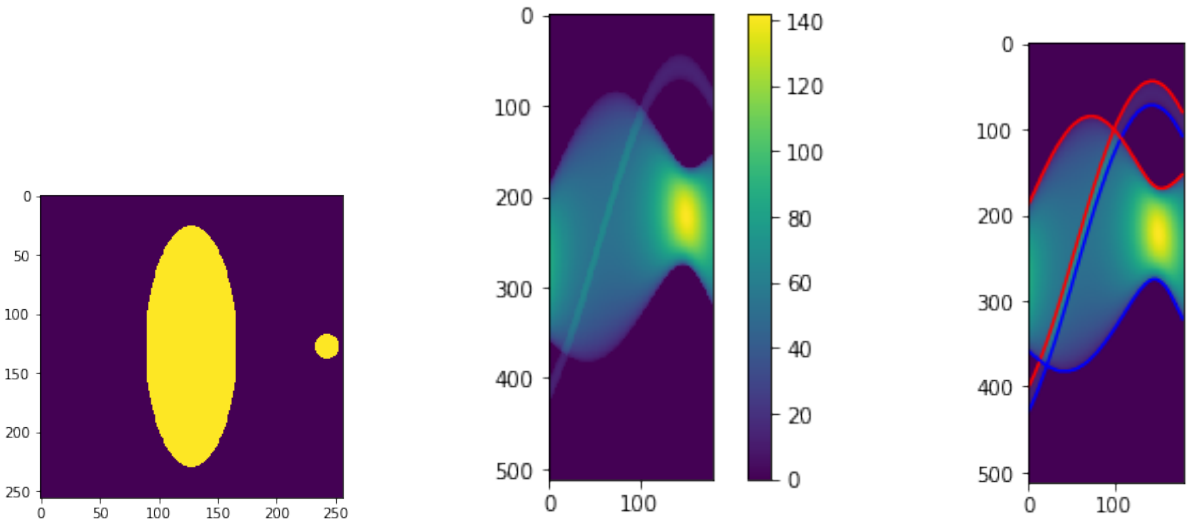
符号	说明
s	下标后缀：以接收器间隔为尺度的长度量
i	下标后缀：以吸收量为尺度的长度量
$\mu(x,y)$	介质在 (x,y) 点的衰减系数分布函数
$F(u,v), F(\rho,\theta)$	衰减系数分布进行二维 Fourier 变换的结果的直角坐标与极坐标形式
$g_\theta(R), \hat{g}_\theta(R)$	探测系统转动角度为 θ 时位于坐标 R 处的投影的真实值与滤波值
$\mathcal{F}_1, \mathcal{F}_1^{-1}$	一维空间中的 Fourier 变换与逆变换算子
xoy	以托盘中心为原点建立的直角坐标系
XOY	以探测系统旋转中心为原点建立的坐标系
l	两个相邻接收器间的距离
R	模型中小圆的半径
a, b	模型中椭圆 ox 与 oy 方向的半轴长
L_m	线长：通过介质的第 m 条射线处的介质厚度
D_m	线距：通过介质的第 m 条射线与最靠近第 0 条线的圆切线的距离
θ	oxy 坐标系相对于其原始状态逆时针旋转的角度
ω, k	扫描线与 oxy 坐标系 ox 正方向的夹角及其对应斜率
$C(X_0, Y_0)$	XOY 坐标系初始状态下小圆圆心的坐标

其中 I 是路径末端接收到的射线强度，I_0 是射线源强度，右侧整体称为吸收值，$\mu(x,y)$ 是路径上衰减系数的分布函数。

数据处理与尝试

当 $\mu(x,y)$ 为常数时，公式2左侧则为此路径上垂直于射线的介质的厚度，故在本小题的条件下（所有有介质处吸收率均为 1），射线的吸收值与其经过的介质厚度成正比。我们在本小题解答过程中，将吸收值作为一个尺度度量，所有在此尺度下的长度量均以 i 作为下标结尾。同时我们也将两个接收器之间的间距作为单位长度定义一个尺度，所

有在此尺度下的长度量均以 s 作为下标结尾。有特殊说明的情况除外。



(a) 给定介质的形状　　**(b)** 原始扫描数据　　**(c)** 含边界的扫描数据

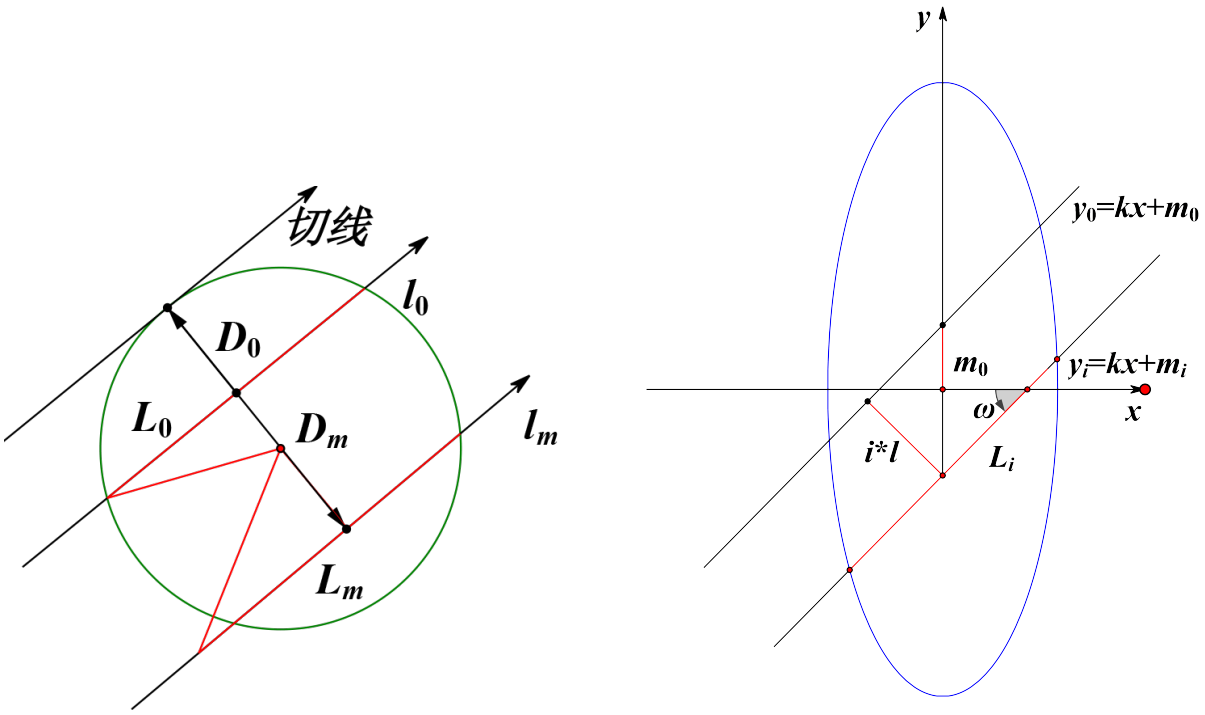
图 **1**　原始数据处理

图1a为题目给出的介质模板形状，我们以其中心（也是椭圆的中心）为原点建立笛卡尔坐标系 xoy，对于此图 x 轴向右，y 轴向上。同时我们以待求的探测系统的旋转中心为原点再建立笛卡尔坐标系 XOY，Y 轴正方向从接收器垂直指向探测器，X 轴正方向指向接收器横坐标增大的方向。对于探测系统的接收器，坐标定义为：在数据第 i 行的接收器横坐标为 $i - 255.5$；所有接收器纵坐标相等，规定为 -500（只要探测器都在介质之外，其纵坐标大小不影响接受值）。

图1b为根据题目给定的对模板的测量数据绘制的图，横轴为 180 个方向，纵轴为探测器编号，点颜色亮度表示吸收值。可知图中任意一条垂直线的宽度即为当前垂直 X 射线方向（即为平行探测器方向）的模型的宽度。图中有一条较明显的高度相同的正弦曲线形状的条带，容易推断为模板中圆形留下的图样；另一条宽度变化的条带，则为椭圆留下的图样。

首先进行较为粗略的数据提取，找到两条条带各自的边界（即，检测到吸收的接收器的范围）。方法思路为使用 Python 读取每一列数据，找到吸收值突变点。在两条带未重合的部分，识别十分简单。在重合部分，方法为从附近的未重合部分向上下进行试探，寻找最大值（观察可知图形上重合部分属于椭圆条带的边界比较亮）。图1c为用此方法寻找到的两条带边界，基本与图形相吻合。但经过观察可知，条带图在峰、谷部分的边界并不明显（在数据上表现为各点值较接近，在图形上表现为边界线变“平”，边缘“黯淡”），这是由于接收器间具有一定的间隔造成的。故用这种方法得到的边界并不准确，无法直接用于后续计算。

计算接收器间距与圆心位置



(a) 扫描线与圆的几何关系　　**(b)** 扫描线与椭圆的几何关系

图 **2**　计算时几何关系的图示

考虑另一种思路，对于条带未重合时的数据，可精确地得知穿过圆的扫描线（即一个接收器接收到的信号）位置。下面的推导均在吸收值尺度下进行。考虑一个特定的方向，将这些线按接收器编号升序记为 $l_0, l_1, \ldots, l_n$，共 $n+1$ 条，对于第 m 条线定义“线长”即为对应接收器的读数 L_m，定义“线距” D_m 为第 m 条线与此方向下最靠近 l_0 的圆的切线之间的距离。根据图2a中所描绘的几何关系，我们有

$$R^2 = (D_m - R)^2 + (\frac{L_m}{2})^2 \tag{3}$$

又由于 $D_m = D_0 + ml$，代入公式3后与 $m = 0$ 时的情况相减，可得到

$$2ml(D_0 - R) + m^2 l^2 = \frac{L_0^2 - L_m^2}{4} \tag{4}$$

将 $l(D_0 - R)$ 与 l^2 作为未知系数，$2m$ 与 m^2 作为参数进行线性回归，可得到在这个方向下 l^2 的值。对所有条带未重合的方向重复该操作，取平均值，可得到 $l_i = 0.4903$（在 SI 制下 $l = 0.2766\,\text{mm}$）。这组数据的标准差为 9.74×10^{-4}，稳定程度较高。

由于圆直径最长，而数据中属于圆的吸收值部分出现多次明显的最大值，故我们直接将每个旋转方向上吸收值最大值视为圆直径，再取这些值中的最大值后得 $d_i = 14.1796$。计算特定角度下圆心在旋转坐标系中的 X 坐标的方法也是简单的：在经过圆

的所有扫描线中去掉线长 L_m 最大的两个 l_{max}, l_{max+1}（一定相邻），则 $l_0, l_1, \ldots, l_{max-1}$ 与 $l_{max+2}, l_{max+3}, \ldots, l_n$ 分列圆心两侧，根据垂径定理（即公式3）即可计算出圆心与直线的距离，即可得到圆心的横坐标。对第 j 个旋转角度取得的所有坐标取均值，确定为该方向上圆心的横坐标 C_{x_j}。当圆心横坐标确定后，我们根据横坐标与半径即可对于无重叠的部分圆条带重新确定一个更精确的边界，此边界的数值不再只落在接收器代表的点上。

确定旋转角度与旋转中心

由于探测系统所在的坐标系旋转时圆在其横轴上留下的投影位置发生变化，根据几何约束可求解旋转系统在每次投影时的旋转角度与其旋转中心相对模板的位置。具体推导如下（下列推导中，所有长度量的度量均以吸收量为尺度，略去尺度下标）：

在 oxy 系中，考虑椭圆 $x^2/a^2 + y^2/b^2 = 1$（其中 a、b 分别为已知的两个半轴长）与扫描线 $y = kx + m$，联立方程，根据解析几何知识容易知道直线截椭圆的弦长为：

$$L = \sqrt{1+k^2}\sqrt{(x_1+x_2)^2 - 4x_1x_2} = \sqrt{k^2+1}\sqrt{\frac{4a^4k^2m^2}{(a^2k^2+b^2)^2} - \frac{4a^2\left(m^2-b^2\right)}{a^2k^2+b^2}} \tag{5}$$

将 XOY 系的两个轴正方向均与 oxy 系相同时的状态称为“初始位置”，当系统相对初始位置旋转角度为 θ 时，扫描线与 ox 轴的夹角满足 $\omega = \theta + \frac{\pi}{2}$。由于所有扫描线都是平行的，故第 i 条直线的截距 m_i 可以表示为 $m_0 + i\Delta m$，又由于 $\Delta m = l/\cos(\pi - \omega) = -l/\cos\omega$，且 $k = \tan\omega$。上述等式均可简单地由图2b中表述的几何关系推导出。此时，方程5可约化为：

$$\sqrt{k^2+1}\sqrt{\frac{4a^4k^2\left(il\sqrt{k^2+1}+m_0\right)^2}{(a^2k^2+b^2)^2} - \frac{4a^2\left(\left(il\sqrt{k^2+1}+m_0\right)^2 - b^2\right)}{a^2k^2+b^2}} = L_i \tag{6}$$

对于穿过椭圆的每一根直线，其 i 与 L_i 的值是确定的，于是方程6便构成一个关于 k 与 m_0 的二元方程。在每一个方向上，都有足够多的扫描穿过椭圆（且不穿过圆），构成一个方程数量远多于未知数的超定方程组。我们将其相邻两两联立求解，舍去无用的 m_0 后将获得的 k 的绝对值（由于方程中均为关于 k 的偶次项，故一定会有正负两个根）进行平均，作为该方向上的扫描线斜率。在扫描线与 ox 轴接近垂直的角度上，三角函数会产生较大的误差，此时我们将直线方程重新设为 $x = py + q$ 进行求解，求得的 $|p|$ 为实际斜率绝对值的倒数。

在获得所有方向的 $|k_i|$ 后，可据此计算出 θ_i 的值；在一些临界值处的取值，是参考扫描数据的变化给出的。具体计算公式为：

$$\theta_i = \begin{cases} \frac{\pi}{2} - \arctan k_i, & 0 \le i \le 61 \\ \frac{\pi}{2} + \arctan k_i, & 61 \le i \le 150 \\ \frac{3\pi}{2} - \arctan k_i, & 151 \le i \le 179 \end{cases} \tag{7}$$

特别地，我们确定了第一组扫描数据对应的 $\theta_0 = 0.51735\,\text{rad} = 29.6420\,\text{deg}$

下一步确定旋转中心。初始位置中圆形的圆心在 XOY 系中坐标记为 $C(X_0, Y_0)$，到 O 的距离 $d = \sqrt{X_0^2 + Y_0^2}$ 恒定。旋转中心在 oxy 系中的坐标即为 $(-X_0, -Y_0)$。当 XOY 旋转 θ 角度时，射线 OC 与 OX 轴形成有向角 $\phi_\theta = \arg(X_0 + iY_0) - \theta$（其中 i 为虚数单位），于是得到 C 的纵坐标 $X_\theta = d\cos\phi_\theta$。由于在上面的过程中我们已经得到了部分旋转角度下比较精确的圆心坐标，将 θ_i 与对应的 C_{x_i} 代入上面的推导即可获得关于 X_0 与 Y_0 的两个方程。由于数据较多，我们选取了两段未与椭圆重叠的圆条带（扫描序号分别为 $0 \le n \le 13$ 与 $110 \le n \le 180$），每次在两个集合中各选择一个扫描数据进行方程求解，共产生 910 组解。将其分别平均后得到的解为（在扫描线间距的尺度下）$X_{0\_s} = 196.0739, Y_{0\_s} = -22.6529$，转换尺度后易知探测系统的旋转中心在 SI 制下在 oxy 系中的坐标为 $(-9.2383\,\text{mm}, 6.2663\,\text{mm})$，即位于模型板中心的略左上方位置处。

4.2 问题 2、3

问题 2 与问题 3 的性质相同，均属于使用扫描得到的数据重建样品的吸收率分布(即形状)，故二者可合并解决。

数学原理：反投影

本题要解决的问题实际为逆 Radon 变换，这个问题最早由数学家 Johann Radon 在 1917 年提出[5]。我们首先进行理论上的推导。

对于二维密度函数 $\mu(x, y)$，将其进行二维 Fourier 变换后所得的函数记为 $F(u, v)$，根据 Fourier 逆变换公式可以得到

$$\mu(x, y) = \iint F(u, v)\mathrm{e}^{2\pi i(ux+vy)}\,\mathrm{d}u\,\mathrm{d}v \tag{8}$$

将空域变换为极坐标形式 $u = \rho\cos\theta, v = \rho\sin\theta$ 后可得

$$\begin{aligned}\mu(x, y) &= \int_0^\infty \int_0^{2\pi} F(\rho, \theta)\mathrm{e}^{2\pi i\rho R}\rho\,\mathrm{d}\theta\,\mathrm{d}\rho \\ &= \int_0^\infty \int_0^{\pi} F(\rho, \theta)\mathrm{e}^{2\pi i\rho R}\rho\,\mathrm{d}\theta\,\mathrm{d}\rho + \int_{-\infty}^0 \int_0^{\pi} F(-\rho, \theta+\pi)\mathrm{e}^{2\pi i\rho R}\rho\,\mathrm{d}\theta\,\mathrm{d}\rho\end{aligned} \tag{9}$$

其中 $R = x\cos\theta + y\sin\theta$。由于 $\mu(x, y)$ 为实值函数，其二维 Fourier 变换满足 $F(\rho, \theta) = F(-\rho, \theta + \pi)$，因此

$$\mu(x, y) = \int_{-\infty}^\infty \int_0^{\pi} F(\rho, \theta)|\rho|\mathrm{e}^{2\pi i\rho R}\,\mathrm{d}\theta\,\mathrm{d}\rho \tag{10}$$

利用 Fourier 中心切片定理[3]70-72，

$$F(\rho,\theta)=\mathcal{F}_1\{g_\theta(R)\} \tag{11}$$

将公式11代入公式10后可得

$$\mu(x,y)=\int_0^\pi \mathcal{F}_1^{-1}\{\mathcal{F}_1\{g_\theta(R)\}\cdot|\rho|\}\,\mathrm{d}\theta=\int_0^\pi \hat{g}_\theta(R)\,\mathrm{d}\theta \tag{12}$$

我们令 $\hat{g}_\theta(R)=\mathcal{F}_1^{-1}\{\mathcal{F}_1\{g_\theta(R)\}\cdot|\rho|\}=g_\theta(R)*\mathcal{F}_1^{-1}\{|\rho|\}$。注意到公式 12 中右侧积分的意义相当于将 $\hat{g}_\theta$ 作为吸收强度做直接反投影，因此我们得到了一个能较为精确还原出原始信息的算法。

推导中的 $|\rho|$ 相当于一个滤波函数，我们所做的相当于利用其对 $\mathcal{F}_1\{g_\theta(R)\}$ 进行滤波后再变换回频域进行直接反投影。由于利用 g_θ 做直接反投影和利用 $\hat{g}_\theta$ 做直接反投影相差了一个与 $\mathcal{F}_1^{-1}\{|\rho|\}$ 的卷积，原先的方法会造成如图像模糊等诸多问题。

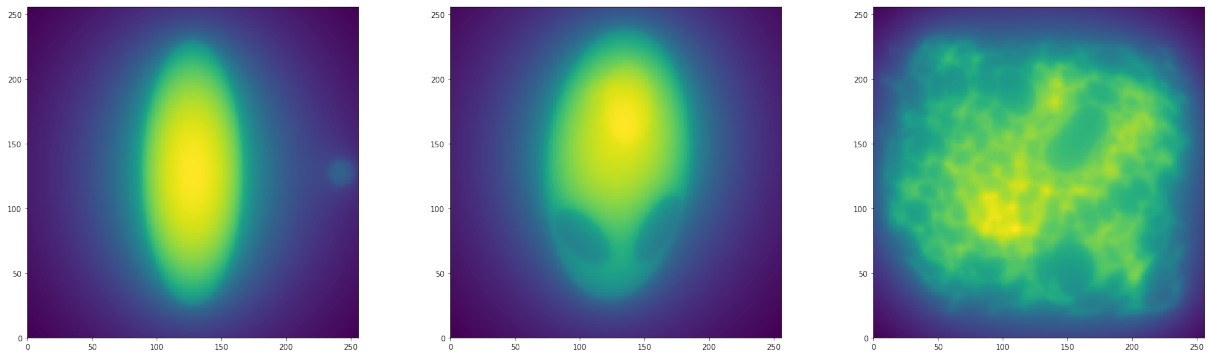
初步实现：直接反投影

在本题的具体处理上，需要对 180 个方向分别进行反投影。对于每一个转动角度 θ 下 xoy 系中的任意一点 P，进行以下的处理：

1. 寻找投影坐标：设待测点 P 在 xoy 坐标系中的坐标为 (x_p,y_p)，在 XOY 坐标系的 OX 轴上的投影值为 R。此时 OX 轴在 xoy 系中的直线方程为 $y\cos\theta=x\sin\theta$，过 P 向其作垂线，方程为 $(y-y_p)\sin\theta+(x-x_p)\cos\theta=0$，垂足即为 R。联立方程即可解得 $R_x=x_p\cos\theta+y_p\sin\theta$。
2. 获取投影值：由于投影坐标一般不会落在接收器对应的坐标点上，故需要对接收器值进行插值。我们选取离坐标最近的两个接收器值进行线性插值，作为该点在此角度下的吸收值。
3. 累加吸收值：将上一部中得到的吸收值加到该点原有的值上

事实上，除了这种“像素驱动”（即，从点出发寻找线上坐标）的反投影方法外，还有“射线驱动”法，即每次从射线上一点作垂线，在射线上等距步进，每次前进一步就将采样值分配到相邻像素中。之所以没有采用这种方法，主要原因是求解射线与坐标系的小格相交问题比较困难，同时这种方法包含二维的插值，需要更大的运算量。

当所有转动角度均遍历过后，直接反投影过程即完成。尝试用本法反投影恢复题中附件所给数据的几何信息，得到的图形（图3a）较为模糊，轮廓不清；椭圆周围有较广的晕影，几乎遮盖了小圆的部分。尝试恢复附件（3）时，出现了同样的问题（图3b）；对于附件（5），直接反投影的方法得到的途中充满了“伪影”，除大致轮廓外几乎没有有用的信息（图3c）。



(a) 附件（**2**）　　**(b)** 附件（**3**）　　**(c)** 附件（**5**）

图 **3**　直接反投影恢复出的介质

改进：滤波反投影

由于直接反投影法的缺陷，根据公式12，我们可以通过在进行反投影前对投影数据进行滤波处理以提高结果的准确度。对于每一个旋转角度 θ，在遍历像素前先进行以下处理：

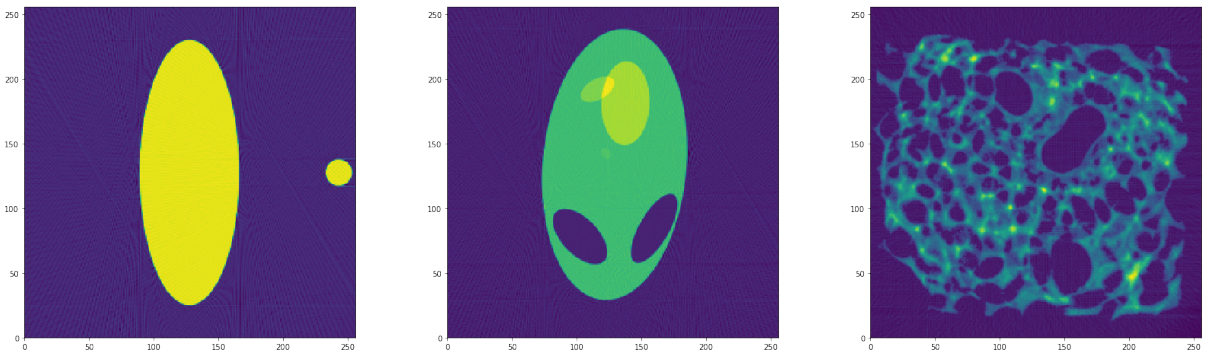
1. DFT：将接收器坐标点对应的吸收值进行离散 Fourier 变换
2. 滤波：在上一步所得的频域结果上乘给定的滤波函数
3. IDFT：将上一部所得的结果进行逆离散 Fourier 变换，并去除所有结果的虚部，只保留实部作为新的吸收值

关于滤波器的选择，许多文献均有比较详细的介绍[1][2]96-97[3]80-82。作为测试，我们尝试了 R-L 滤波器、S-L 滤波器、Hanning 滤波器等多种不同的函数。由于 R-L 滤波函数在工业界被广泛使用、原理与实现比较简单、与其他滤波器效果也没有明显劣势，我们将其作为最终的选择。在 R-L 滤波器算法的实现部分，我们参考了 Matlab 中 `iradon` 函数的实现[4]70-76。

图4为对三个附件分别进行滤波反投影得到的图像。从直观上图像的清晰度、辨识度都有了极大的提升。图3a与附件（2）给出的模型吻合度高，完整地恢复出了圆、椭圆的内部，吸收率分布也很均匀。附件（3）（图3b）的介质主体形状为一个放置不正的椭圆，内有五个近似椭圆形的吸收率不同的区域：下部两个区域吸收率很低，中部的一个接近介质其他部分，上部的两个吸收率较高，并有重叠（重叠部分为两者吸收率之和）。附件（5）（图3c）是一种多孔的稀疏介质，介质的吸收率分布并不均匀。

最终：计算吸收率

最终每点的吸收率需要对数据进行归一化：计算重建附件（2）得到的圆与椭圆内部的吸收率平均值（再除 1.00），此即为在我们恢复得到的一点的数值与题目给出的吸收率的比例因子；根据比例因子即可将样例（2）与（3）的每点数值转化为对应的吸收

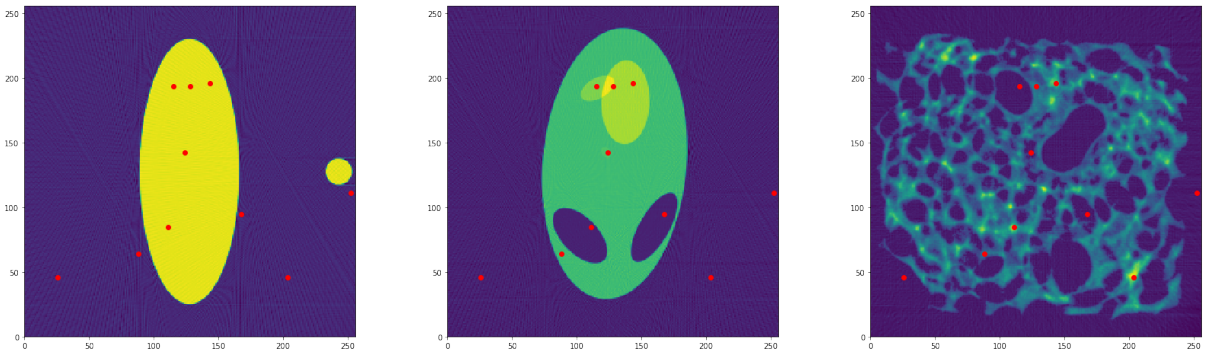


(a) 附件（**2**）　　**(b)** 附件（**3**）　　**(c)** 附件（**5**）

图 **4**　滤波反投影恢复出的介质

率。所有的吸收率数据已按照题目要求在附件中给出。

由于题目所给的 10 个位置在进行坐标转换后并非落在 256×256 坐标的整数点上，故我们需要对每个点的吸收率进行重新计算，以确保数据的准确。计算方法与上面完全是相同的，仅仅将每次遍历的点变为给定的 10 个。最后我们得到了附录中的表4中的结果。由于不可避免的噪声影响与算法的局限性，计算出部分点的吸收值为负；为了符合真实情况，我们将负数结果统一置为 0。同时我们也将这 10 个点对应绘制在了图5上，以供参考。



(a) 附件（**2**）　　**(b)** 附件（**3**）　　**(c)** 附件（**5**）

图 **5**　恢复出的图像上给定的坐标点

4.3 问题 4

精度与稳定性分析

问题 4 要求分析问题 1 中参数标定的精度和稳定性，下面逐一进行讨论。

首先考虑求解接收器间距 l 时使用的回归方法的精度。我们可以认为由于外界干扰以及机器自身的内禀影响，测量得到的值 L_m 与真实值 $\hat{L}_m$ 之间有如下关系：

$$L_m = \hat{L}_m + \sigma\varepsilon_m, \varepsilon_m \sim \mathcal{N}(0,1) \tag{13}$$

并且 $\varepsilon_0, \varepsilon_1, \cdots, \varepsilon_n$ 之间相互独立。

那么对于公式 4 中右侧量 $L_0^2 - L_m^2$ 可以计算其方差:

$$\begin{aligned} \mathrm{Var}[L_0^2 - L_m^2] &= \mathrm{Var}[(\hat{L}_0 + \sigma\varepsilon_0)^2] + \mathrm{Var}[(\hat{L}_m - \sigma\varepsilon_m)^2] \\ &\le \mathrm{Var}[2\sigma\varepsilon_0] + \mathrm{Var}[\sigma^2\varepsilon_0^2] + \mathrm{Var}[2\sigma\varepsilon_m] + \mathrm{Var}[\sigma^2\varepsilon_m^2] \\ &= 8\sigma^2 + 4\sigma^4 \end{aligned} \tag{14}$$

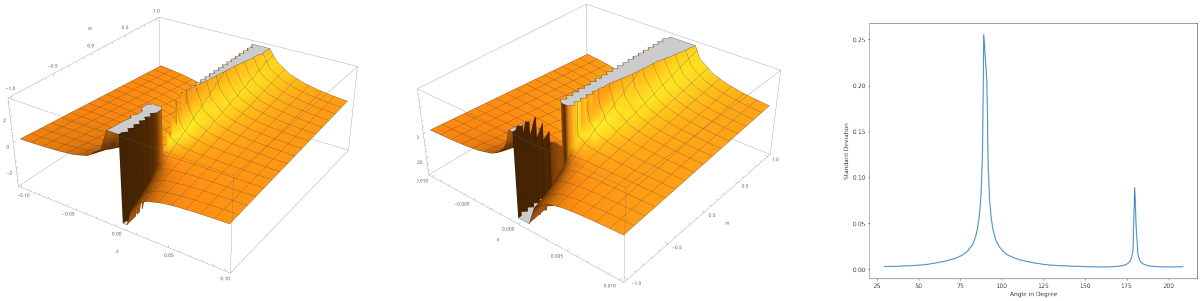
如果利用 $l_0, \cdots, l_n$ 共 $n+1$ 条直线来计算接收器间隔 l，将公式 4 中的回归方程记为 $Ax = L$，其中 $L = \frac{1}{4}(L_0^2 - L_1^2, \cdots, L_0^2 - L_n^2)^T$，$x = ((D_0 - R)l, l^2)^T$，并且 $x = (A^TA)^{-1}A^TL$，计算可得

$$(A^TA)^{-1} = \begin{bmatrix} \frac{9}{2(n^2+n-2)} & -\frac{6(2n+1)}{n(n+1)(n^2+n-2)} \\ -\frac{6(2n+1)}{n(n+1)(n^2+n-2)} & \frac{36}{n(n+1)(n^2+n-2)} \end{bmatrix} \tag{15}$$

为了符号简便，将矩阵15第二行记为 (a_{21}, a_{22})，可以得到

$$\begin{aligned} \mathrm{Var}[l^2] &\le \mathrm{Var}[a_{21}\sum_{i=1}^{n} 2iL_i] + \mathrm{Var}[a_{22}\sum_{i=1}^{n} i^2L_i] \\ &\le \frac{2\sigma^2 + \sigma^4}{4}\left[a_{21}^2P_3(n) + a_{22}^2P_5(n)\right] \\ &= \frac{2\sigma^2 + \sigma^4}{Q_3(n)} \end{aligned} \tag{16}$$

其中 $P_3(n), P_5(n)$ 分别是关于 n 的 3 次多项式和 5 次多项式，$Q_3(n)$ 是关于 n 的 3 次多项式。因此，我们知道 $\mathrm{Var}[l^2]$ 和 n 至少为立方反比关系。



(a) $\frac{\partial k}{\partial L_0}, |k| \le 0.1$ **(b)** $\frac{\partial k}{\partial L_0}, |k| \le 0.01$ **(c)** 计算得到 ω 的标准差

图 **6** 图**a**与图**b**是当 k 在不同范围内变化时，可以观察到当 $k \to 0$ 时，$\frac{\partial k}{\partial L_0} \to \infty$；由图**c**可见当椭圆轴与射线接近平行时计算得到的 ω 标准差有突增

对于角度的计算，我们是通过方程组6中联立相邻两个方程再取平均来完成的。如图6c，在投影线和椭圆的轴接近平行时标准差较大，下面将会分析该现象发生的原因。

首先将方程6中的弦长写成关于 m_0 和 ω 的函数 $G_i(\omega, m_0) = G_i(\arctan k, m_0)$。为了方便，我们不妨设联立的两个方程为

$$\begin{cases} G_0(\omega, m_0) = L_0 \\ G_i(\omega, m_0) = L_i \end{cases} \tag{17}$$

根据反函数定理，如果 $\partial(L_0, L_i)/\partial(\omega, m_0)$ 存在，那么

$$\frac{\partial(\omega, m_0)}{\partial(L_0, L_i)} = \left[\frac{\partial(L_0, L_i)}{\partial(\omega, m_0)}\right]^{-1} = \left[\frac{\partial(L_0, L_i)}{\partial(k, m_0)} \cdot \frac{\partial(k, m_0)}{\partial(\omega, m_0)}\right]^{-1} \tag{18}$$

记 $J_\omega(\omega, m_0) = \dfrac{\partial(L_0, L_i)}{\partial(\omega, m_0)}$，$J_k(k, m_0) = \dfrac{\partial(L_0, L_i)}{\partial(k, m_0)}$，从公式6可以看出 $J_\omega(\omega, m_0)$ 的各个元素在原点附近都是连续可微的。并且计算后可得

$$J_\omega(0, m_0) = J_k(0, m_0) = \begin{bmatrix} 0 & -\frac{2am_0}{b\sqrt{b^2 - m_0^2}} \\ 0 & -\frac{2a(il + m_0)}{b\sqrt{b^2 - (il + m_0)^2}} \end{bmatrix}, \tag{19}$$

我们可以设

$$J_\omega(\omega, m_0) = \begin{bmatrix} A(\omega, m_0) & B(\omega, m_0) \\ C(\omega, m_0) & D(\omega, m_0) \end{bmatrix} \tag{20}$$

这样当 $\det J_\omega$ 非零时，该矩阵可逆并且

$$J_\omega^{-1}(\omega, m_0) = \frac{1}{\det J_\omega(\omega, m_0)} \begin{bmatrix} D(\omega, m_0) & -B(\omega, m_0) \\ -C(\omega, m_0) & A(\omega, m_0) \end{bmatrix} \tag{21}$$

由公式19以及 A, B, C, D 的连续性可知，当 $\omega \to 0$ 时，$A \to 0$ 且 $C \to 0$。而此时，根据 m_0 的几何意义可知 $|m_0| < b$，不妨认为在选取数据时忽略距离椭圆边界最近的两条直线，那么 B, D 在原点附近有界。因此可以得到

$$\det J_\omega(\omega, m_0) = AD - BC \to 0 \tag{22}$$

那么就有

$$\frac{\partial \omega}{\partial L_0} = \frac{D(\omega, m_0)}{\det J_\omega(\omega, m_0)} \to \infty, \qquad \frac{\partial \omega}{\partial L_i} = \frac{D(\omega, m_0)}{\det J_\omega(\omega, m_0)} \to \infty \tag{23}$$

这样，根据13，在存在一个白噪声 $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ 的情况下，当投影线与椭圆的轴平行的时候所求得的 ω 的方差将会过大。相应地，θ 的方差也变大。

综上，我们得到如下的分析结果：

定律 1 *回归得到的 l 精确程度与穿过圆区域的射线数目成正相关*

定律 2 *当射线与椭圆的轴接近平行时，随机噪声的存在将使 θ 的求解结果变得不稳定*

最后，由于旋转中心的坐标可 $C(X_0, Y_0)$ 完全由上述求得的 l 与 θ 所确定，故其精度与稳定性受到上述两个因素的共同制约。

模板设计与模型修正

下面提出一个简单的模板设计来减轻上述问题对标定产生的影响。

对于定律1的结论，已知增大穿过圆的射线条数 n 能够增加 l 的精确程度。由于接收器间隔是一定的，故应当增大模型上圆形区域的尺寸。对于定律2的结论，我们应当尽量避开射线与椭圆轴平行的角度（至多有三个位置），故可以每次扫描后将椭圆模板绕其中心旋转一定角度，重复多次这样的操作，就能获取充分多的数据避开“坏”的角度。

本参数标定模型的模板包含一至多个半径不同的圆与一个椭圆，并且其吸收率均相同（如 1.0000）。在每次扫描时均只使用其中的一个，避免扫描条带出现重叠现象影响数据的处理。

对于接收器间距 l 的计算，可选取半径较大的圆（可以考虑直径为托盘宽度的 2/3），圆心与托盘中心重合，再利用第4.1节提出的回归算法计算 l。如果认为方差过大或者精度不足可以改变圆的位置或半径多次计算并且取平均值。l 值的计算与圆心所处的位置无关。

对于探测系统相对于初始状态旋转角度 θ 的计算，选取中心位于托盘中心的，长轴长约为托盘宽度 2/3 且与 x 轴平行的椭圆。设 n 为总共的扫描次数（一般推荐取 5 左右），则每次椭圆旋转的角度应为 π/n。假设第 i 扫描得到的第 j 个旋转角度的估计值为 $\theta_{i,j}$，数据的标准差为 $s_{i,j}$，且 $\alpha_i = \arg\min_j s_{i,j}$，则最后确定的旋转角度 θ_i 为 θ_{i,α_i}。

五、 模型评价与展望

在这次建模过程中，我们成功完成了题目所要求的标定系统参数与重建扫描图像的任务，并定量分析原有模型的不足，提出了在理论上更优的标定模型。

我们的整体工作突出之处有：

- 在标定参数的过程中充分利用题目给出的数据与几何关系构成方程，求解得到的参数值较精确，统计意义上较为优秀
- 在重建图像的过程中选择了恰当的投影方法与滤波函数，恢复出的图像轮廓清晰，在样本上的恢复与原数据吻合程度较高
- 在模型分析过程中充分考虑可能的测量误差，定量地建立了样本数量、系统状态与误差间的数学关系

- 提出的新模型在标定过程中使用的算法较简单，且得到的数据精度更高

不足之处有：

- 求解参数过程使用的关系式较为复杂，且数据量大，导致求解过程略显缓慢
- 使用的 R-L 滤波函数会在空域中造成震荡响应，降低了吸收率较低的介质区域的信噪比
- 提出的新模型需要较多次扫描得到的数据，操作较为繁琐，且对模板放置位置精度要求较高

此模型未来可能有的进一步发展包括且不限于：

- 调整算法以适应接收器间距不再严格相等的情况，消除系统带来的误差
- 考虑射线在空气等非被测介质中的损耗及可能的散射、反射等现象带来的干扰
- 将标定与重建算法从二维空间推广到三维甚至更高维度（含时）的空间
- 优化计算复杂度，减少待解方程数量，将数值运算问题转化为较易解决的统计学问题

参考文献

[1] 高上凯. 医学成像系统（第 2 版）[M]. 北京: 清华大学出版社, 2010 (引用页: 10).

[2] 顾本立, 万遂人, 赵兴群. 医学成像原理[M]. 北京: 科学出版社, 2012 (引用页: 10).

[3] 康雁. 医学成像技术与系统[M]. 北京: 清华大学出版社, 2014 (引用页: 8, 10).

[4] KAK A C, SLANEY M. Principles of computerized tomographic imaging[M]. New York: SIAM, 2001 (引用页: 10).

[5] RADON J. On the determination of functions from their integral values along certain manifolds[J]. IEEE Transactions on Medical Imaging, 1986, 5(4): 170–176. DOI: 10.1109/TMI.1986.4307775. ISSN: 0278-0062 (引用页: 8).

附录 A 建模结果数据

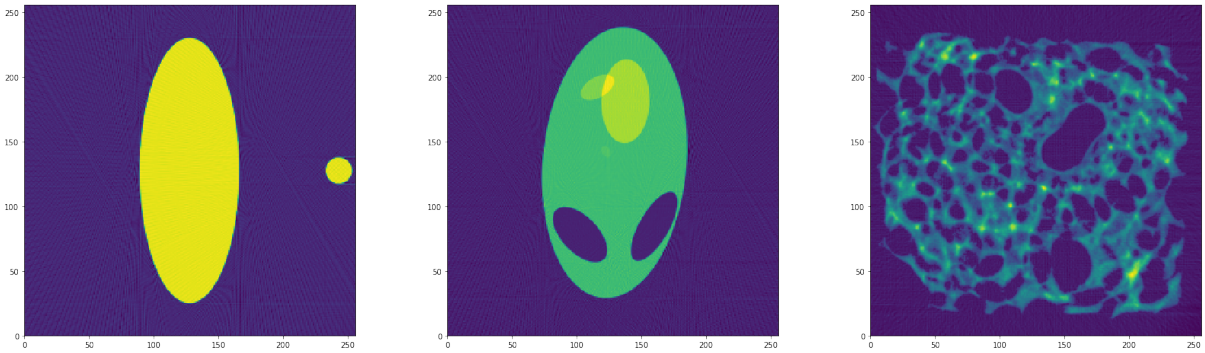
表 **2**　计算出的数值结果

符号	值	说明
l	$0.2766\,\mathrm{mm}$	两个相邻接收器间的距离
(x_0, y_0)	$(-9.2383\,\mathrm{mm}, 6.2663\,\mathrm{mm})$	xoy 坐标系中探测系统旋转中心坐标

表 **3**　**180** 个旋转方向

29.6422°	30.9957°	31.5511°	32.6404°	33.6726°	34.6418°
35.6417°	36.6416°	37.6415°	38.6414°	39.6412°	40.6411°
41.6410°	42.6409°	43.6408°	44.7911°	45.6406°	46.6406°
47.6405°	48.6404°	49.6403°	50.6402°	51.6402°	52.6400°
53.6399°	54.6396°	55.6393°	56.6389°	57.6386°	58.6382°
59.6379°	60.5365°	61.6370°	62.6364°	63.6360°	64.6353°
65.6349°	66.6342°	67.6335°	68.6327°	69.6320°	70.6309°
71.6300°	72.6288°	73.6276°	74.6262°	75.6247°	76.6228°
77.6208°	78.6183°	79.6154°	80.6119°	81.6075°	82.6019°
83.5949°	84.5856°	85.5718°	86.5496°	87.5109°	88.4265°
89.1500°	91.0183°	91.8326°	92.7671°	93.7351°	94.7164°
95.7041°	96.6950°	97.6888°	98.6838°	99.6795°	100.6763°
101.6735°	102.6712°	103.6692°	104.6676°	105.6659°	106.6646°
107.6633°	108.6624°	109.6613°	110.6606°	111.6595°	112.6588°
113.6582°	114.6575°	115.6570°	116.6564°	117.4534°	118.6554°
119.6549°	120.6546°	121.6543°	122.6538°	123.6535°	124.6532°
125.6529°	126.6526°	127.6524°	128.6523°	129.6521°	130.6521°
131.7520°	132.6519°	133.6519°	134.6518°	135.6517°	136.6517°
137.6516°	138.6515°	139.6514°	140.6513°	141.6511°	142.6510°
143.6509°	144.6508°	145.6507°	146.6507°	147.6505°	148.6505°
149.6504°	150.6503°	151.6502°	152.6502°	153.6501°	154.6501°
155.6501°	156.6500°	157.6500°	158.6500°	159.6500°	160.6500°
161.6500°	162.6500°	163.6501°	164.6501°	165.6503°	166.6503°
167.6506°	168.6508°	169.6510°	170.6513°	171.6517°	172.6523°

173.6530°	174.6541°	175.6556°	176.6584°	177.6633°	178.6757°
179.7497°	180.5807°	181.6219°	182.6311°	183.6351°	184.6373°
185.6387°	186.6397°	187.6404°	188.6409°	189.6413°	190.6416°
191.6419°	192.6421°	193.6422°	194.6423°	195.6424°	196.6425°
197.6425°	198.6425°	199.6425°	200.6425°	201.6425°	202.6425°
203.6425°	204.6424°	205.6424°	206.6424°	207.6423°	208.6317°



(a) 附件（**2**）　**(b)** 附件（**3**）　**(c)** 附件（**5**）

图 **7**　滤波反投影恢复出的介质

表 **4**　两样例给定点处的吸收率数据

	1	2	3	4	5
附件（3）	0	0.9979	0	1.2050	1.0866
附件（5）	0.0780	2.8227	6.7965	0.1994	0.1626
	6	7	8	9	10
附件（3）	1.4175	1.2915	0.0064	0.0286	0
附件（5）	3.1407	6.4676	0	7.3136	0

附录 B 程序源码

注：所有的 *Python* 代码需要在交互式环境（如 *Jupyter Notebook*）中顺序依次执行。

2.1 Python（第一部分）

此部分主要完成的工作为读入数据，使用条带未重合部分进行回归得到圆半径和圆心的精确投影坐标。后将每个旋转角度的数据输出待 Mathematica 计算。

```
#!/use/bin/env python3
# -*- coding: utf8 -*-

import matplotlib.pyplot as plt
import copy
import numpy as np
import scipy.stats
from sklearn import linear_model
import math
import re
import fileinput

def load_data(filename):
    f = open(filename, 'r').read()
    return [list(map(float, s.split('\t')))
            for s in f.replace(' ', '').strip().split('\n')]

attach_1 = load_data('attachment1.txt')
attach_2 = load_data('attachment2.txt')
attach_3 = load_data('attachment3.txt')
attach_5 = load_data('attachment5.txt')

# Question 1
data1 = np.array(attach_2)
data1 = np.transpose(data1)

# find the position of the circle in the first scanline
```

```python
for i in range(511, 0, -1):
    if data1[0][i] != 0:
        circle_bottom = i
        break

for i in range(circle_bottom, 0, -1):
    if data1[0][i] == 0:
        circle_top = i + 1
        break

circle = [(circle_top, circle_bottom)]
circle_shift = 5

for i in range(1, 120):
    ct, cb = circle[-1]
    # find circle top
    max_now, max_id = -1, 999
    for shift in range(1, circle_shift):
        val = data1[i][ct - shift] - data1[i][ct - shift - 1]
        if val > max_now:
            max_id, max_now = ct - shift, val
    ct = min(max_id, ct)

    # find circle bottom
    max_now, max_id = -1, 999
    for shift in range(1, circle_shift):
        val = data1[i][cb - shift - 1] - data1[i][cb - shift]
        if val > max_now:
            max_id, max_now = cb - shift, val
    cb = min(max_id, cb)

    circle.append((ct, cb))

for i in range(120, 180):
    for j in range(0, 512):
```

```
        if data1[i][j] != 0:
            ct = j
            break
    for j in range(ct, 512):
        if data1[i][j] == 0:
            cb = j - 1
            break
    circle.append((ct, cb))

def find_bottom(data, start):
    for i in range(start, 0, -1):
        if data[i] != 0:
            return i

ellipse_shift = 5
ellipse_bottom = [find_bottom(data1[0], 390)]

for i in range(1, 30):
    eb = ellipse_bottom[-1]
    # find circle bottom
    max_now, max_id = -1, 0
    for shift in range(1, circle_shift):
        val = data1[i][eb + shift - 1] - data1[i][eb + shift]
        if val > max_now:
            max_id, max_now = eb + shift - 1, val
    ellipse_bottom.append(max(max_id, eb))

for i in range(30, 180):
    ellipse_bottom.append(find_bottom(data1[i], 511))

def find_top(data, start):
    for i in range(start, 512):
        if data[i] != 0:
            return i

```

```
ellipse_top = [find_top(data1[i], 0) for i in range(85)]

for i in range(85, 120):
    et = ellipse_top[-1]
    # find circle top
    max_now, max_id = -1, 0
    for shift in range(0, circle_shift):
        val = data1[i][et + shift] - data1[i][et + shift - 1]
        if val > max_now:
            max_id, max_now = et + shift, val
    ellipse_top.append(max(max_id, et))

ellipse_top.extend([find_top(data1[i], 120) for i in range(120, 180)])

# output to file
open('bound-circle.txt', 'w').write('\n'.join(
    [str(c[0]) + '\t' + str(c[1]) for c in circle]))
open('bound-ellipse.txt', 'w').write('\n'.join(
    [str(ellipse_top[i]) + '\t' + str(ellipse_bottom[i]) for i in
    ↪ range(180)]))

# scanline width
def get_circle_scanline(i):
    return data1[i][circle[i][0]:circle[i][1] + 1]

def scanline_width(line):
    n = len(line)
    y = [0.25 * (line[0] * line[0] - line[i] * line[i]) for i in range(1,
    ↪ n)]
    X = [(2 * i, i * i) for i in range(1, n)]
    clf = linear_model.LinearRegression()
    clf.fit(X, y)
    _, LL = clf.coef_
    return math.sqrt(LL)

```

```
131 l_i_many = [scanline_width(get_circle_scanline(i)) for i in range(110,
    ↪ 180)]
132 l_i = np.mean(l_i_many)
133
134 d_i_many = [max(get_circle_scanline(i)) for i in range(110, 180)] \
135         + [max(get_circle_scanline(i)) for i in range(0, 14)]
136 d_i = max(d_i_many)
137 d_s = d_i / l_i  # circle radius in scanline
138
139 def find_bound_s(index, ignore=2):
140     cb, ct = circle[index]
141     line = data1[index]
142     circle_line = line[cb : ct + 1]
143     peak = np.argmax(circle_line)
144
145     d_center_s = [0.5 * math.sqrt(d_i * d_i - l * l) / l_i for l in
        ↪ circle_line]
146
147     center_many = []
148     for i, dc_s in enumerate(d_center_s[: peak - ignore]):
149         center_many.append(cb + i + dc_s)
150
151     for i, dc_s in enumerate(d_center_s[peak + ignore + 1: ]):
152         base = cb + i + peak + ignore + 1
153         center_many.append(base - dc_s)
154
155     center_s = np.mean(center_many)
156  #   print(np.var(center_many))
157
158     return center_s - 0.5 * d_s, center_s + 0.5 * d_s
159
160 bound_0_13 = [find_bound_s(i) for i in range(13)]
161 bound_110_180 = [find_bound_s(i) for i in range(110, 180)]
162
163 def get_angle(width):
```

```
164     D = d * width
165     s = 2880/11 * (1/225 - 4/(D**2))
166     t = math.asin(s)
167     n2 = math.atan(math.sqrt((4*40*40-D**2)/(D**2-4*15*15)))
168     return n2 * 180 / math.pi
169 
170 open('middle_0_13.txt', 'w').write('\n'.join([str(0.5 * (y + x)) for x, y
    ↪  in bound_0_13]))
171 open('middle_110_180.txt', 'w').write('\n'.join([str(0.5 * (y + x)) for
    ↪  x, y in bound_110_180]))
172 
173 def extract_ellipse(index, ignore=3):
174     et, eb = ellipse_top[index] + ignore, ellipse_bottom[index] - ignore
175     ct, cb = circle[index]
176     ct, cb = ct - ignore, cb + ignore
177 
178     line = data1[index]
179     scan_lines = []
180     for e in range(et, eb):
181         if ct <= e <= cb or not line[e]:
182             continue
183         scan_lines.append((e - et, line[e]))
184     return scan_lines
185 
186 ellipse_slines = [extract_ellipse(i) for i in range(180)]
187 
188 def write_ellipse_to_file(filename, line_id):
189     f = open(filename, 'w')
190     f.write('\n'.join(str(n) + ' ' + str(L) for n, L in
        ↪  ellipse_slines[line_id]))
191     f.close()
192 
193 for i in range(180):
194     write_ellipse_to_file('ellipse_scanline/%d.txt' % i, i)
195 
```

```
# waiting for Mathematica's output
input('Waiting for Mathematica\' Output')
```

2.2 Mathematica（第一部分）

此部分的作用为读入 Python 初步处理后的数据，并计算每个状态对应的方程解，输出为直线斜率 k。

```
SolveDirection[id_, step_] := (
  equation = ( -L +
         Sqrt[1 + k^2]
           Sqrt[(4 a^4 k^2 m^2)/(b^2 + a^2 k^2)^2 - (
           4 a^2 (-b^2 + m^2))/(b^2 + a^2 k^2)]) /. a -> 26.58675 /.
      b -> 70.898 /. m -> (Subscript[m, 0] + n*l*Sqrt[1 + k^2]) /.
    l -> 0.49029825753407719;
  dirPath = "ellipse_scanline/";
  filePath = dirPath <> ToString[id] <> ".txt";
  data = Import[filePath, "Table"];
  length = Length[data];
  result =
   Table[NSolve[(equation /. n -> data[[i]][[1]] /.
          L -> data[[i]][[2]]) ==
        0 && (equation /. n -> data[[i + step]][[1]] /.
          L -> data[[i + step]][[2]]) == 0, {Subscript[m, 0], k},
      Reals] // Values, {i, Range[1, length - step]}];
  Export[dirPath <> "ans_step_" <> ToString[step] <> "" <>
    ToString[id] <> ".txt", result, "Table"];
  );


SolveDirectionInverted[id_, step_] := (
  equation = (-L +
         Sqrt[1 + p^2]
           Sqrt[(4 b^4 p^2 q^2)/(a^2 + b^2 p^2)^2 - (
           4 b^2 (-a^2 + q^2))/(a^2 + b^2 p^2)]) /. a -> 26.58675 /.
```

```
      b -> 70.898 /. q -> (Subscript[q, 0] + n*l*Sqrt[1 + p^2]) /.
    l -> 0.49029825753407719;
  dirPath = "ellipse_scanline/";
  filePath = dirPath <> ToString[id] <> ".txt";
  data = Import[filePath, "Table"];
  length = Length[data];
  result =
   Table[NSolve[(equation /. n -> data[[i]][[1]] /.
          L -> data[[i]][[2]]) ==
        0 && (equation /. n -> data[[i + step]][[1]] /.
          L -> data[[i + step]][[2]]) == 0, {Subscript[q, 0], p},
      Reals] // Values, {i, Range[1, length - step]}];
  Export[dirPath <> "ans_inverted_step_" <> ToString[step] <> "/" <>
    ToString[id] <> ".txt", result, "Table"];
  );

For[i = 0, i < 138, ++i, SolveDirection[i, 1]];

For[i = 138, i < 162, ++i, SolveDirectionInverted[i, 1]];

For[i = 162, i < 180, ++i, SolveDirection[i, 1]];
```

2.3 Python（第二部分）

本部分读入上一部分的数据后将其转化为准确的弧度值 θ，并将其与每一个状态的圆心投影坐标再次输出。

```
def load_angle_data(filename):
    slope_many = []
    for line in open(filename, 'r').read().split('\n'):
        if line.strip():
            slope_one_many = re.findall(r'\{-?\d*\.?\d*,
             ↪  (-?\d*\.?\d*)\}', line)
            slope_many.append(abs(float(slope_one_many[0])))
    return slope_many
```

```
8
9  def calc_angle(index):
10     if 138 <= index <= 161:
11         is_inverted = True
12         filename = 'ellipse_scanline/ans_inverted_step_1/%d.txt'
13     else:
14         is_inverted = False
15         filename = 'ellipse_scanline/ans_step_1/%d.txt'
16     slope_many = load_angle_data(filename % index)
17     if not slope_many: return 0
18     slope = np.mean(slope_many)
19     omega = math.pi / 2 - math.atan(slope) if is_inverted else
       ↪  math.atan(slope)
20     if index <= 60: theta = math.pi / 2 - omega
21     elif index <= 150: theta = math.pi / 2 + omega
22     else: theta = 3 * math.pi / 2 - omega
23     return theta, np.std(np.arctan(slope_many))
24
25 angles = [ calc_angle(i)[0] * 180 / math.pi for i in range(180) ]
26
27 angle_f = open('angle_middle_0_13.txt', 'w')
28 angle_f.write('\n'.join([str(angles[i]) + ' ' + str((bound_0_13[i][1] +
   ↪  bound_0_13[i][0] - 511) / 2) for i in range(13)]))
29 angle_f.close()
30 angle_f = open('angle_middle_110_180.txt', 'w')
31 angle_f.write('\n'.join([str(angles[i]) + ' ' + str((bound_110_180[i -
   ↪  110][1] + bound_110_180[i - 110][0] - 511) / 2) for i in range(110,
   ↪  180)]))
32 angle_f.close()
33
34 # Waiting for Mathematica's Output
35 input('Waiting for Mathematica\' Output')
```

2.4 Mathematica（第二部分）

本部分从文件获得圆心投影坐标与旋转角度的数据后，构成方程组求解旋转中心的坐标。

```
centerCoord =
 NSolve[Sqrt[X^2 + Y^2]*
       Cos[ArcTan[X, Y] - #[[1]][[1]] \[Degree]] == (#[[1]][[2]]) &&
     Sqrt[X^2 + Y^2]*
       Cos[ArcTan[X, Y] - #[[2]][[1]] \[Degree]] == (#[[2]][[2]]), {X,
      Y}] & /@
  Tuples[{Import["angle_middle_0_13.txt", "Table"],
    Import["angle_middle_110_180.txt", "Table"]}]

centerCoord = centerCoord // Values

Export["center_coords.txt", centerCoord, "Table"]
```

2.5 Python（第三部分）

本部分获取旋转中心坐标，借此实现两个坐标系的转换。后对题目给出的原始数据在每个旋转角度上进行滤波反投影，得到原始图像，并输出每个点的吸收值。最后对给定的 10 个点再次单独计算吸收值。

```
def load_center_data(filename):
    data = open(filename, 'r').read()
    center_many = re.findall(r'\{(-?\d*\.?\d*), (-?\d*\.?\d*)\}', data)
    center_many = [ list(map(float, line)) for line in center_many ]
    return center_many

center_offset_many_s = load_center_data('center_coords.txt')
center_offset_s = np.mean(center_offset_many_s, axis=0)
center_mm = [45, 0] - 8 / d_s * center_offset_s

# Question 2
d_mm = 8      # diameter of circle in mm
```

```
scale_i2mm = d_mm / d_i
scale_mm2p = 256 / 100.
scale_i2p  = scale_i2mm * scale_mm2p

proj_axis_s = np.array(range(512)) - 0.5 * 511
proj_axis_p = proj_axis_s * l_i * scale_i2p

center_p = center_mm * scale_mm2p
thetas = np.array(angles) * math.pi / 180

def proj_pixel_dist(pixel, theta, center_p):
    p_x = pixel[0] - center_p[0]
    p_y = pixel[1] - center_p[1]
    return p_x * np.cos(theta) + p_y * np.sin(theta)

def proj_direct(g, axis_p, theta, center_p):
    size = 256
    mu = np.zeros([size, size])
    for n in range(len(theta)):
        R_flatten = [ proj_pixel_dist([x - (size - 1) / 2, y - (size - 1)
        ↪  / 2], theta[n], center_p)
                    for x in range(size) for y in range(size) ]
        g_theta = np.reshape(np.interp(R_flatten, axis_p, g[n]), [size,
        ↪  size])
        mu = mu + g_theta
    return np.flip(np.transpose(mu), axis=0)

# Filtering
def filter_projection(g, H):
    return np.real(np.fft.ifft(np.fft.fft(g, axis=1) * H, axis=1))

def RL_conv(rho_0, length):
    T = 0.5 / rho_0
    h = []
    for i in range(0, length // 2 + 1):
```

```
        if i == 0:
            h.append(0.25 / T ** 2)
        elif i % 2 == 0:
            h.append(0)
        else:
            h.append(-1. / (math.pi * i * T) ** 2)
    h += list(reversed(h[1:-1]))
    return np.real(np.fft.fft(h))

def get_kernel():
    return RL_conv(0.5 / (l_i * scale_i2p), 512)

def inv_proj(data, kernel=None):
    if kernel is None: kernel = get_kernel()
    return proj_direct(filter_projection(np.transpose(data), kernel),
    ↪  proj_axis_p, thetas, center_p)

def filter_min0(data):
    for i in range(256):
        for j in range(256):
            data[i][j] = max(data[i][j], 0)
    return data

# Generating Data
attach2_recons = filter_min0(inv_proj(attach_2))
attach3_recons = filter_min0(inv_proj(attach_3))
attach5_recons = filter_min0(inv_proj(attach_5))

# Recons Intensity
max_attach2_recons = np.max(np.abs(attach2_recons))
scale_recons = 1.0 / np.mean([ x for x in
↪  np.reshape(np.abs(attach2_recons), [256 * 256]) if x >
↪  max_attach2_recons * 0.5 ])

attach_4 = load_data('attachment4.txt')
```

```
pixels = np.array(attach_4) * scale_mm2p
pixels_x, pixels_y = zip(*pixels)

def compute_recons_intensity(data, pixels):
    g = filter_projection(np.transpose(data), get_kernel())
    mu = np.zeros(len(pixels))
    for n in range(len(thetas)):
        R = [ proj_pixel_dist(pixel, thetas[n], center_p) for pixel in
        ↪  pixels - (256 - 1) / 2 ]
        g_theta = np.interp(R, proj_axis_p, g[n])
        mu = mu + g_theta
    for i in range(len(pixels)):
        mu[i] = max(mu[i], 0)
    return mu * scale_recons

def compute_approx_intensity(data, pixels):
    return np.array([ data[255 - int(p[1])][int(p[0])] for p in pixels ])
    ↪  * scale_recons

compute_recons_intensity(attach_2, pixels)
compute_recons_intensity(attach_3, pixels)
compute_recons_intensity(attach_5, pixels)

open('attach5_recons.txt', 'w').write('\n'.join([','.join(['%.4f' %
↪  attach5_recons[i][j] for j in range(256)]) for i in range(256)]))
open('attach3_recons.txt', 'w').write('\n'.join([','.join(['%.4f' %
↪  attach3_recons[i][j] for j in range(256)]) for i in range(256)]))
open('attach2_recons.txt', 'w').write('\n'.join([','.join(['%.4f' %
↪  attach2_recons[i][j] for j in range(256)]) for i in range(256)]))
```