# 1 问题重述

电子计算机断层扫描（应用ng yong﷽﷽﷽﷽﷽﷽﷽﷽﷽﷽﷽﷽﷽﷽﷽﷽﷽﷽﷽﷽构造ient in parabolic problems[J]. Computed Tomography, CT）是一种常用的多维检测手段，常用于医学检验、金属验伤与安全检查等。该技术利用不同材质对于X射线等穿透射线吸收特性不同的性质进行截线或截面扫描，得到待测物体的断层数据；利用截线或截面的投影数据重建多维模型，最终得到较为完整的待测物体内部与外部的形态数据。

已知一种二维CT系统，探测用X射线与接收平面垂直。512只探测器在接收平面上等距排列，探测时接收平面沿某中心逆时针旋转180次并收集射线强度数据，再经处理后形成了上述180个方向的待测物体信息。

由于CT系统存在误差，因此在使用该系统前需要对系统相关参数进行标定，标定的方法可以是借助某已知外部与内部结构的物体（模板）的CT成像数据进行旋转中心、旋转角度步长等系统参数的微调。

本文将建立数学模型和算法，解决以下问题：

(1) 已知均匀材质标定模板的几何信息、吸收特性（吸收率）与成像数据。标定该二维CT系统的旋转中心、探测器单元间距以及各次旋转步长。

(2) 已知某物体的成像数据。利用(1)中标定的系统参数，确定该物体的位置、几何形状与吸收率，同时具体给出所给的10个位置处该物体的吸收率。

(3) 已知另一物体的成像数据。利用(1)中得到的标定参数，确定该物体的位置、几何形状与吸收率，同时具体给出所给的10个位置处该物体的吸收率。

(4) 分析(1)中参数标定的精度和稳定性。设计新模板与对应的标定模型，以改进CT系统标定精度和稳定性，并说明理由。

# 2 问题分析

该题要建立的一种简化的二维CT系统模型。由于CT系统采集的是某一截面的投影数据，这些数据既包括该截面的长度数据，又包括该截面的深度数据，因此妥当利用投影信息中的多维数据是本题的突破口。不难看出，问题(2)(3)(4)需要问题(1)中标定的系统参数作为已知条件，因此问题(1)是本文讨论的重点。

关注到该系统采用平行射线的发射-接收系统，可以认为接收到的投影信息只包含了射线所在一维测度上信息，即无需考虑射线在物体表面与物体内部的衍射与散射情况。

# 3 问题(1)：二维CT系统参数的标定

### 3.1 探测器旋转角度步长的求解

#### 3.1.1 问题(1)分析

问题(1)要求根据标定模板的几何信息和对标定模板180个方向进行CT扫描的接收信息，确定CT系统旋转中心在正方形托盘中的位置、探测器单元之间的距离以及该CT系统使用的X射线的180个方向。

注意到，模板由两部分凸图形组成：椭圆形和圆形。在二者互不遮挡的情况下，由于对称性，圆形在探测器上的投影应当始终是一条长度不变的线段（线段在探测器上的相对位置可能改变）。但椭圆形不是这样，它在探测器上的投影长度将随着入射角度的改变而改变，他们之间存在显式的函数关系。因此，我们可以利用椭圆形的投影长度推导出X射线的入射角度。虽然一个椭圆形的投影长度可能对应着多个入射角度，但题目要求探测时，接收平面是沿某中心逆时针旋转的（即入射方向逆时针旋转），这一限制将帮助我们去伪存真，分析出唯一的入射方向。

#### 3.1.2 具体求解和实现

首先，我们以的椭圆中心为原点建立如图1所示的平面直角坐标系：

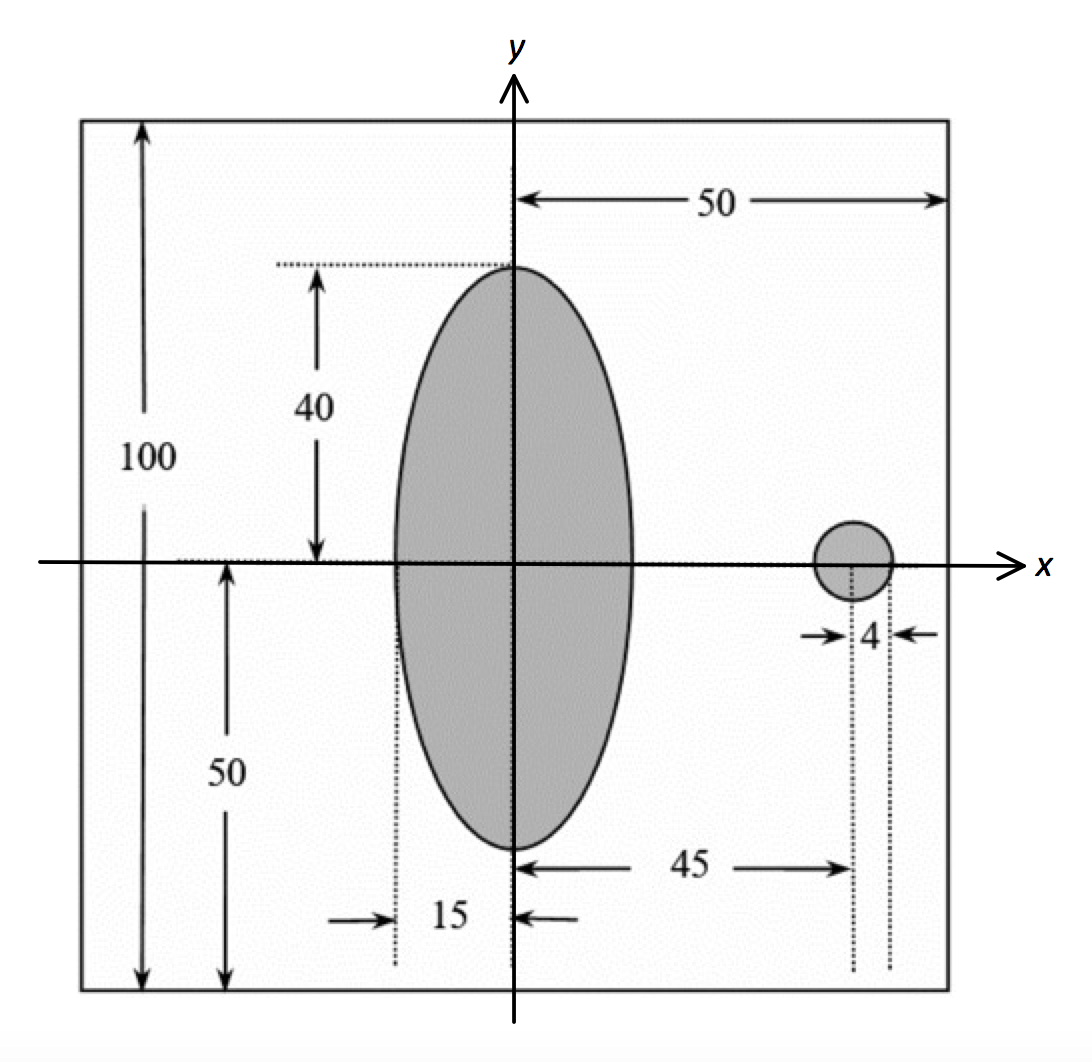


图1 标定模型的平面直角坐标系

椭圆的方程为 设一点在椭圆上,该点的切线斜率为,且满足方程

解得：

设入射方向与x轴的夹角为α，当入射光线与椭圆相切时，设切点为.P点应满足  
当椭圆形模板的投射不受圆形模板影响时，设椭圆的投影长度为,则由几何关系可得  
将代入得到：

利用MATLAB整理附件1中的数据，我们编写的GetScale.m脚本（代码详见附录，下同）可以读出第i次（）测量时的值. 由（\*）式看出，当时，对应多个解. AngleSolve函数能解出每一个对应的所有，AngleMain.m脚本循环使用这个函数，并根据逆时针的条件进行判断，便得到了180个各自对应的唯一符合题意的解.

这样我们就得到了180组. 画出，

### 3.2 探测器固定距离的求解

利用圆柱体各方向投影一致的性质，可以通过提取标定模型中圆柱体的投影信息来估计接收平面上探测器的固定距离，基本原理如下。

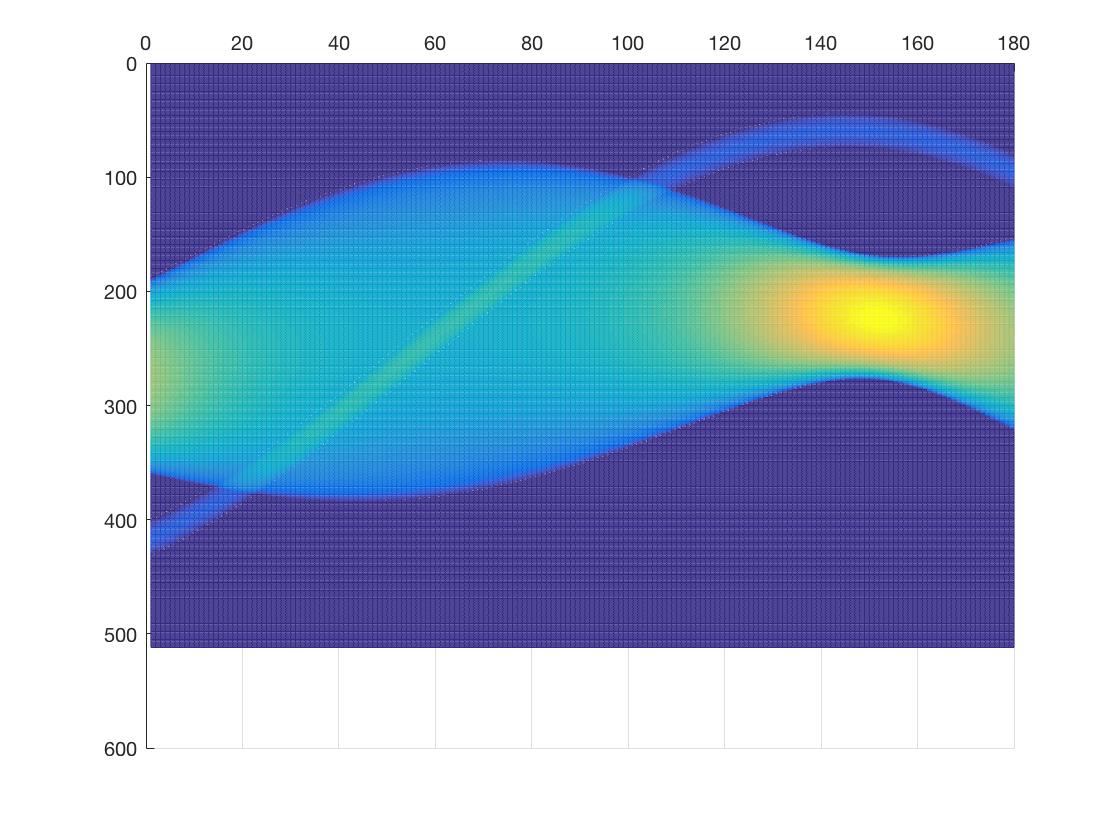


图2 附件2所含投影信息

首先，观察附件2中的投影信息发现，除某些角度上由于椭圆与圆重叠导致边缘模糊化外，可从其他方向的接收数据分析出圆形半径对应的探测器单位尺度。

分析得到：附件2中的第1~13，第109~180数据可用作分析探测器距离，不难得到圆形半径对应的探测器单位为28.83，进一步得探测器间距。

### 3.3 旋转中心的求解

利用附件2中所包含的椭圆投影信息求解旋转中心。已知各方向的投影信息，不难求出该方向上与椭圆相切的直线，即过切点，椭圆投影长度为的切线满足下列方程：

可以求出该方向的切线

通过选取三组不同的角度得到三条切线，分别是

将三条切线利用投影信息进一步修正至投射平面边缘处，得到三条新的直线，记为

其中*b1n, b2n, b3n*根据直线斜率与旋转关系不难得出，此时由旋转中心到三条线距离相等而满足的等式可得线性方程组

解上述方程组可得旋转中心。

选取143组投影信息并平均化，解得旋转中心。

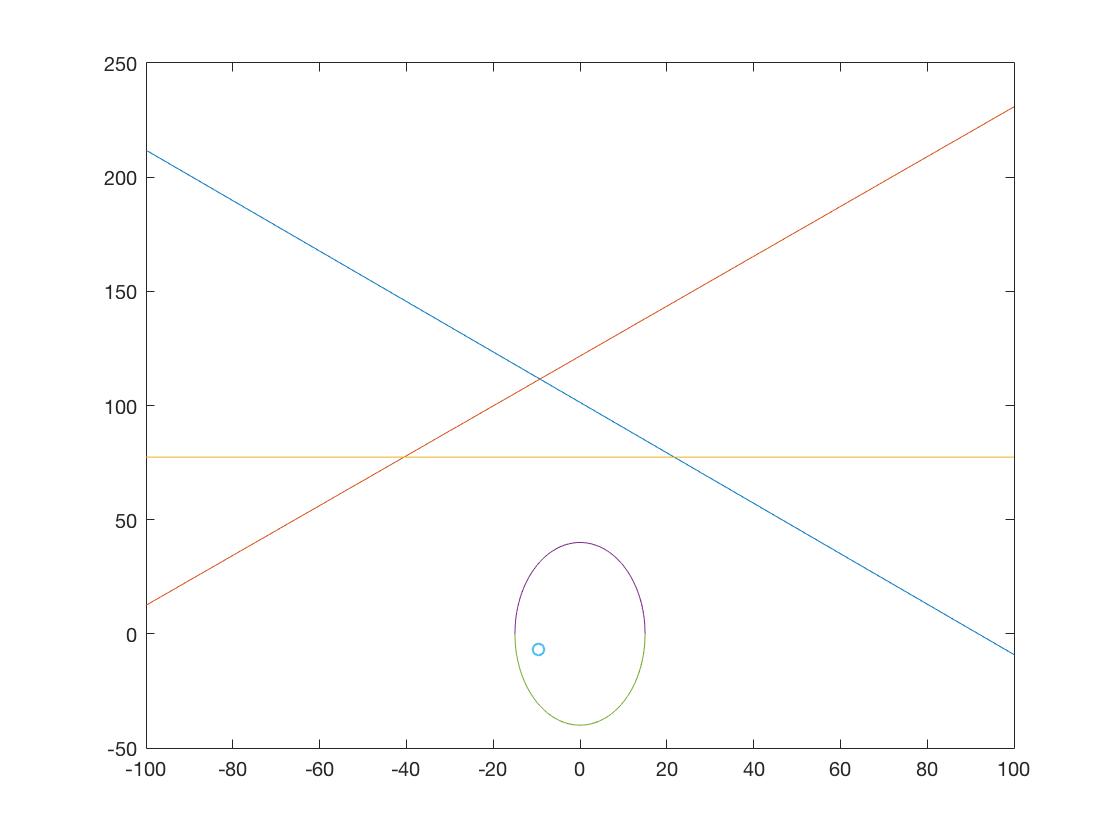


图3 利用特殊点求解旋转中心

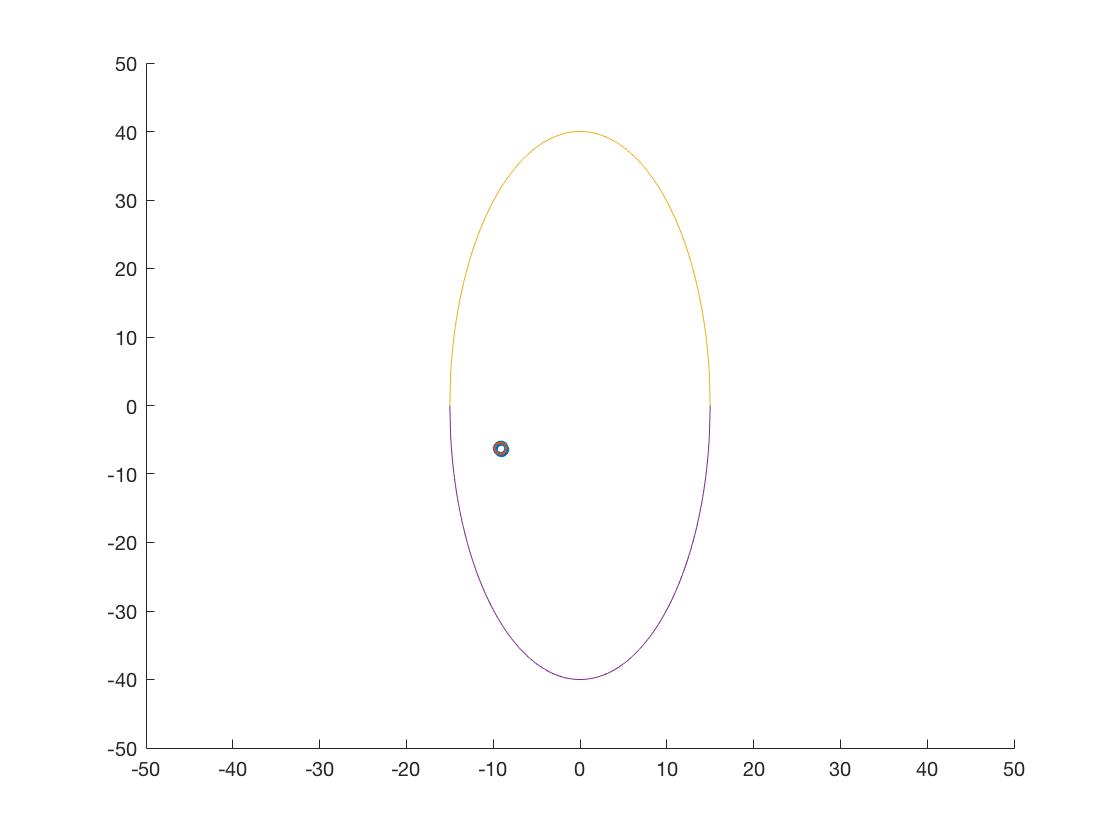


图4 利用143组射线求解旋转中心

由图4可以看出，143组射线接触的旋转中心点分布十分集中，表示模型建立正确。

# 4 反投影重建

### 4.1 模型原理

#### 4.1.1 反投影重建算法

反投影重建算法，或称累加法，是一种经典的从投影信息重建图像模型的算法。其基本思想由经典断层成像算法发展而来，在经典断层成像算法中，每一个不同角度的光线经过物体之后，其投影与光路上吸收率的积分值有直接线性关系，那么反过来，所有经过原物体中某点的光线所对应投影的平均值在一定程度上可以反映该点的吸收率。为了引入反投影重建算法，我们做如下假设：

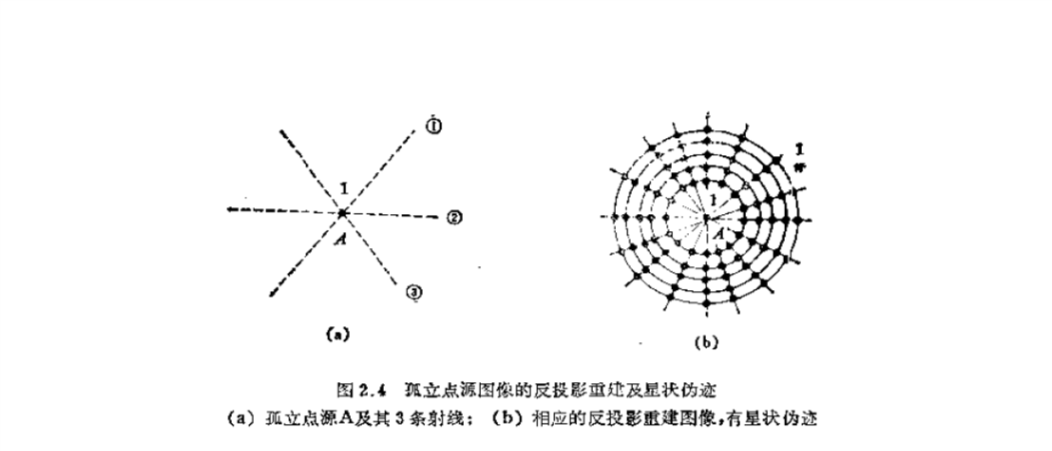
不考虑光路方向的偏折；

某一点吸收率可以近似为所有经过该点的光线对应的投影数值的平均值。

亦即：

公式(3.1.1)中表示第k个像素点(原图形第k个位置)的吸收率，代表通过像素点k的，方向为的射线所对应的投影值，i取从1到表示射线的不同方向。

这是最为基础的反投影重建模型算法，虽然粗糙，但可以大体还原出原物体的几何特性。我们举《CT原理与算法》(庄天戈，上海交通大学出版社)一书中的例证进行说明：



如原图中所标识(a)，若A是一个吸收率为d的空间孤立电源，从图示三个方向①，②和③做射线得到投影图像，容易知道根据上述算法，可以重建出A 的吸收率；但该算法同时会造成伪迹，如(b)所示，对于A周边的各个点，均会重建出d/n的吸收率，这是由于该算法是将有限的投影数值无限的复制到对应光线上任意一点这一特征导致的。

为了消除这种伪迹我们可以对得到的重建图像按照某种规则进行处理；或者在重构之前，先将投影数据进行某种处理，之后在用类似的方法进行重构。我们小组主要运用了后一种想法，即滤波反投影重建算法。

#### 4.1.2 滤波反投影重建算法

为了找出对应的合适的“处理规则”，我们查找资料后，认为“中心切片定理”最为合适，定理描述如下：

定理：某函数f(x,y)在某一个角度下投影函数的一维傅里叶变换给出了原函数f(x,y)的二维傅里叶变换在同一个角度的切片。即

受篇幅所限，此处证明忽略。

由该定理启发，我们认为可以有一下步骤来解决模型重建问题：

1. 获取不同视角下的投影函数；
2. 求出各个投影函数的一维傅里叶变换，这个函数值就对应了原模型在二维傅里叶变换内的某一切面；
3. 按照对应角度，通过所得到的切面构建原函数的二维傅里叶变换；
4. 反变换得到原函数。

注意在（1）中，为了获得高精度和高还原度，理论上应得到分布十分密集的，覆盖180°的投射角度对应的投影函数才可，但这在实际中很难实现，所以在实际应用中可能需要通过离散的取值点进行插值来获取可用的连续变换的投影函数。

下面从数学角度推导该算法的具体步骤，并在此基础上，对其中的关键步骤进行讨论：

设待重建的函数为a(x,y)， 其对应的二维傅里叶变换为A()，是二维频率空间中模长和复角的函数。根据中心切片定理，有以下公式：

对于后一部分积分式，有如下公式：

其中假定：

h(x)是在频域函数||的傅里叶反变换，相当于是一个理想的滤波器，而在信号和系统传递函数角度理解，上式的含义即为：投影信号经过滤波器滤波之后的修正投影信号在满足时的取值。而则正好是过的射线方程。

总体来说我们得到：

以上方程式即为滤波反投影重建算法的核心，它的意思是：根据中心切片定理，将投影信号经过理想滤波器滤波后，在∈[0,π]内所有的过定点(r,)的射线进行积分，即获得了重构的原函数。

### 4.2 模型的建立

由3.1的讨论，我们已经建立了关于滤波反投影重建算法的基本理论，但是要落实到代码中，则还需要在讨论两个问题：滤波器和插值函数。

由之前的讨论，我们选取了在频域上的||作为滤波函数，但这个函数很明显是频带无限宽的，由佩里---维纳准则可知，这种滤波器在现实中不存在，就算是能够从计算机中实现也无法迁移到实际工程中，所以我们需要以此滤波器为模板，寻找和其相近似的，显示可实现的函数作为滤波器。

另外上式中用到了积分项，则就要求被积函数是一个连续函数才便于操作，而在实际应用中，获取无穷多组投影值建立连续的函数是明显不现实的。工程中解决该问题的常用方法是以获得的离散数据作为基础，使用插值函数进行插值，来获取近似的连续函数并积分计算。

在实际问题中，一般有以下几个特点：

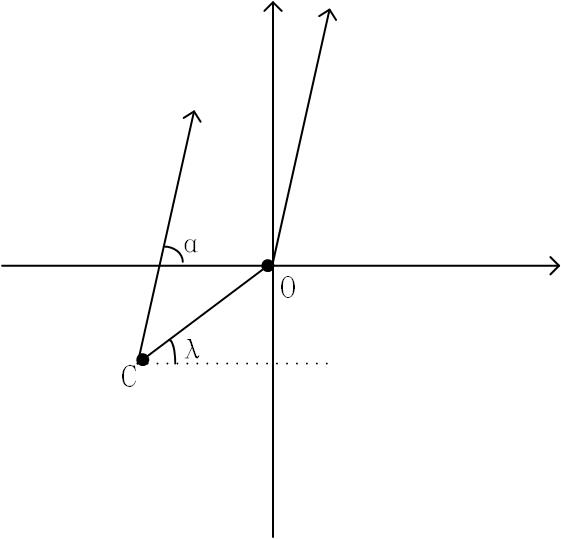
1. 投影数据在频谱上是限宽的，即高频成分有限；
2. 存在误差，所以高频段部分本身信噪比就差于低频部分；
3. 投影数据是离散的。

基于这几点事实，工程前辈们提出了许多滤波算法和其对应的插值算法，比如R-L滤波算法，S-L滤波算法等，在实际应用中应该根据实际情况加以分析，选取。

基于以上讨论，则可以建立起相对较为完整的滤波反投影重建算法。(相关代码见附录)

### 4.3 运用模型求解问题(2)(3)

基于以上叙述与假设，构建函数CTplot(…)，此函数可以根据前述得到的旋转中心调整投影信息，以优化图像信息，优化调整算法如下。



图

如图所示，O为坐标轴原点，C为旋转中心，可得

即在方向上的投影信息应根据旋转中心C向右移动(mm)。代入(2)(3)数据集可得

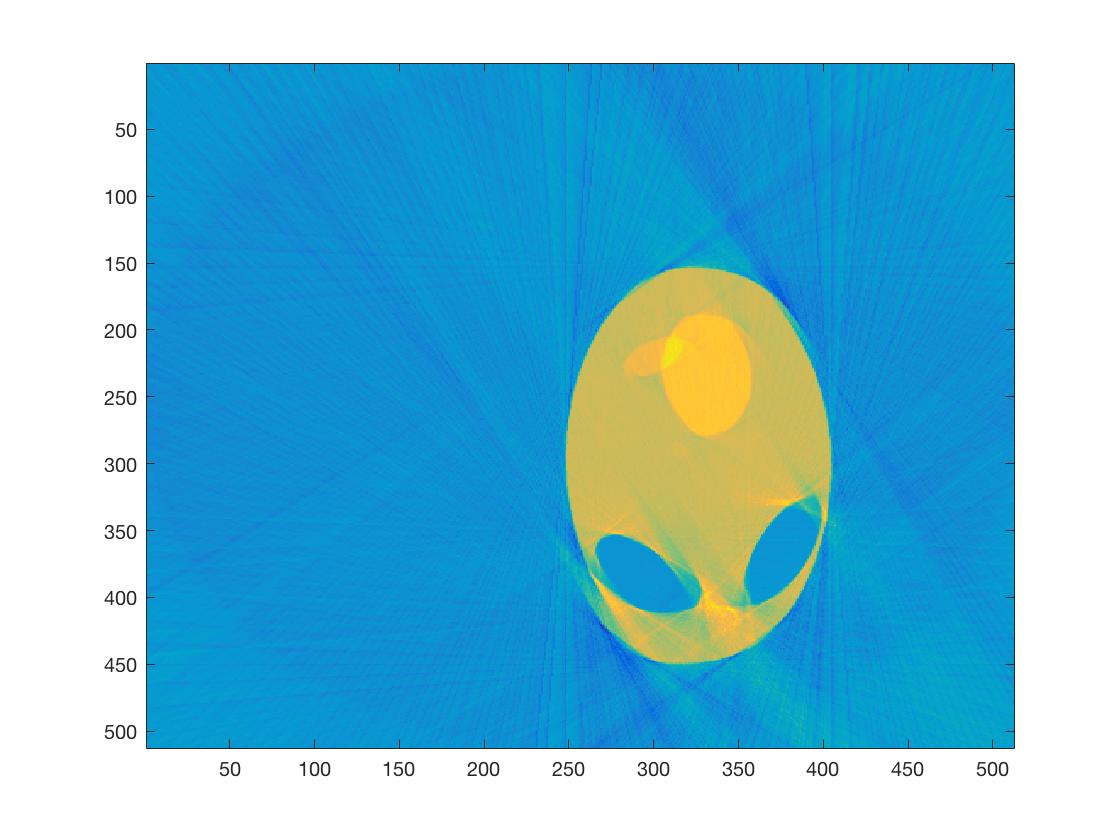
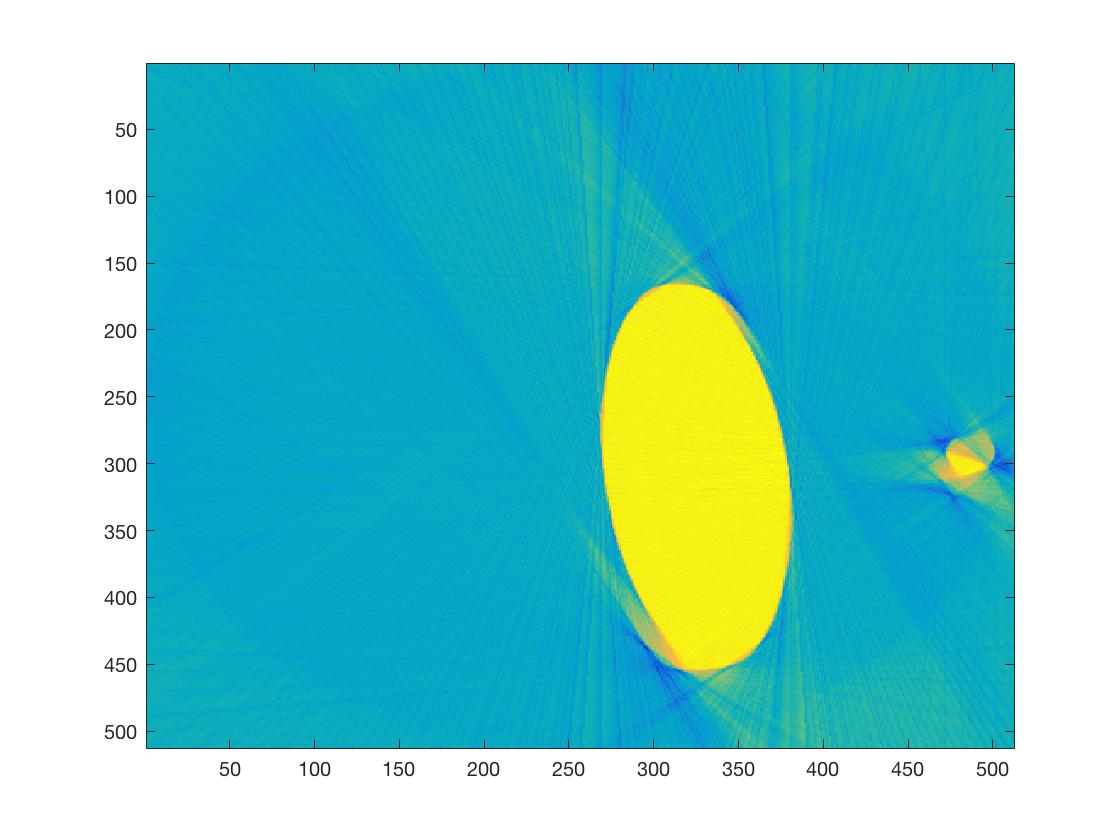
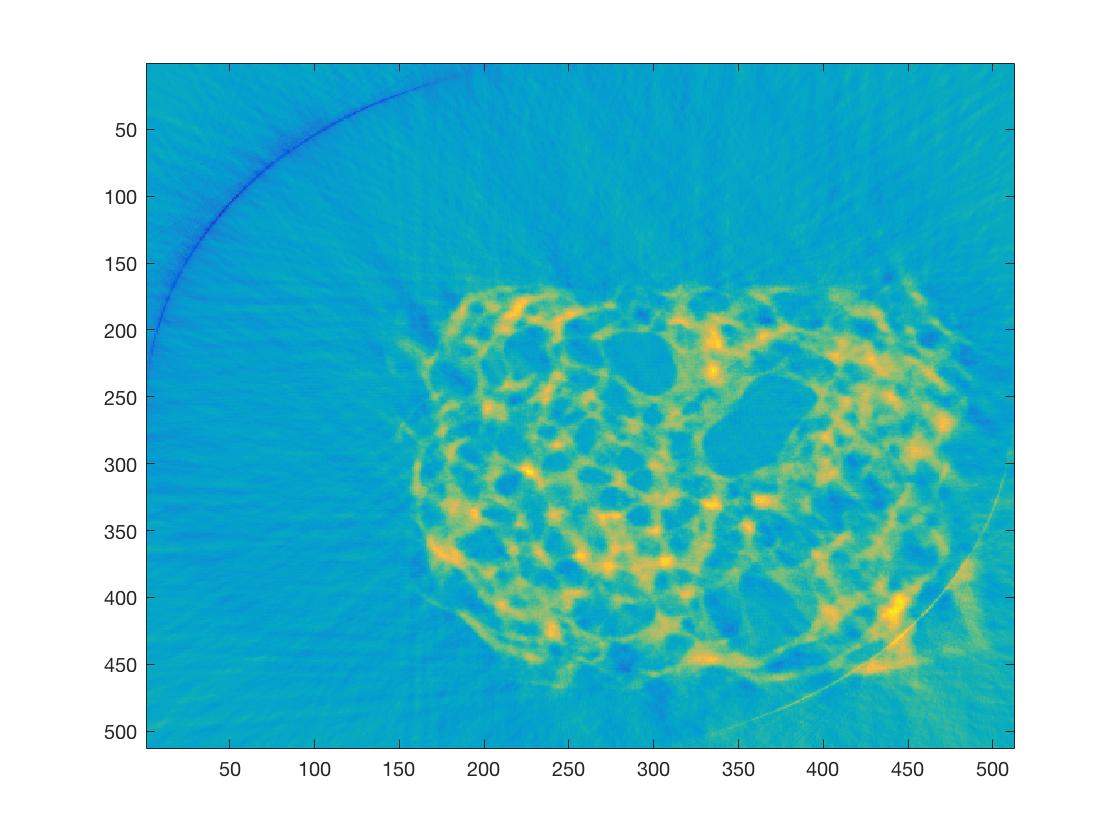


图 图



# 附录

## Distance\_of\_CT.m

|  |
| --- |
| load data.mat data\_2  data = data\_2;  data(:, 14 : 108) = 0;  for i = 1 : 13      data(1 : 370, i) = 0;  end  for i = 109 : 180      data(112 : 512, i) = 0;  end  b=[];  for i = 1 : 180  b = [b length(find(data(:,i)))];  end  b = [b(1:13) b(109:180)];  d = 8 / mean(b);  disp(d); |

## cenfunc.m

|  |
| --- |
| function res = cenfunc(n1\_1, n1\_2, n2\_1, n2\_2)  % origin data  d = 0.2774;  a = 40;  b = 15;  c = sqrt(a^2 - b^2);  e = c / a;  n3 = 279;    % 1st line  y0\_2 = (a^2 - (2 \* a \* b / (n1\_1 \* d))^2) / e^2;  x0\_2 = (1 - y0\_2 / a^2) \* b^2;  y0 = sqrt(y0\_2);  x0 = sqrt(x0\_2);  y1\_k = - x0 \* a^2 / (b^2 \* y0);  y1\_bias = a^2 / y0;  y1\_bias = y1\_bias + n1\_2 \* d / sin(pi / 2 + atan(y1\_k));    % 2nd line  y0\_2 = (a^2 - (2 \* a \* b / (n2\_1 \* d))^2) / e^2;  x0\_2 = (1 - y0\_2 / a^2) \* b^2;  y0 = -sqrt(y0\_2);  x0 = sqrt(x0\_2);  y2\_k = - x0 \* a^2 / (b^2 \* y0);  y2\_bias = a^2 / y0;  y2\_bias = y2\_bias + n2\_2 \* d / cos(atan(y2\_k));    % 3rd line  y3\_k = 0;  y3\_bias = n3 \* d;    % move  res = [y2\_bias/sqrt(y2\_k^2+1)-y1\_bias/sqrt(y1\_k^2+1) y2\_bias/sqrt(y2\_k^2+1)-y3\_bias/sqrt(y3\_k^2+1)]/...      [y1\_k/sqrt(y1\_k^2+1)-y2\_k/sqrt(y2\_k^2+1) y3\_k/sqrt(y3\_k^2+1)-y2\_k/sqrt(y2\_k^2+1);       1/sqrt(y1\_k^2+1)-1/sqrt(y2\_k^2+1) 1/sqrt(y3\_k^2+1)-1/sqrt(y2\_k^2+1)];  end |

## Center\_specific.m

|  |
| --- |
| %  % n1\_1 = 209.5;  % n1\_2 = 140.5;  % n2\_1 = 210.5;  % n2\_2 = 401.5;  %  % cenfunc(n1\_1, n1\_2, n2\_1, n2\_2)  %    % origin data  d = 0.2774;  a = 40;  b = 15;  c = sqrt(a^2 - b^2);  e = c / a;    % 1st line  % parameter  n1\_1 = 209.5; % data\_2(:, 13) num of tuoyuan  n1\_2 = 140.5;    y0\_2 = (a^2 - (2 \* a \* b / (n1\_1 \* d))^2) / e^2;  x0\_2 = (1 - y0\_2 / a^2) \* b^2;  y0 = sqrt(y0\_2);  x0 = sqrt(x0\_2);  y1\_k = - x0 \* a^2 / (b^2 \* y0);  y1\_bias = a^2 / y0;  fprintf('y1 = %f\*x + %f\n', y1\_k, y1\_bias);  % y1 = -1.1046\*x + 43.2957;    y1\_bias = y1\_bias + n1\_2 \* d / sin(pi / 2 + atan(y1\_k));  fprintf('y1\_edited = %f\*x + %f\n', y1\_k, y1\_bias);  % y1\_edited = -1.104572\*x + 101.367747;    % memory clear  clear n1\_1 n1\_2 x0\_2 y0\_2 y0 x0    % 2nd line  % parameter  n2\_1 = 210.5; % data\_2(:, 109) num of tuoyuan  n2\_2 = 401.5;    y0\_2 = (a^2 - (2 \* a \* b / (n2\_1 \* d))^2) / e^2;  x0\_2 = (1 - y0\_2 / a^2) \* b^2;  y0 = -sqrt(y0\_2);  x0 = sqrt(x0\_2);  y2\_k = - x0 \* a^2 / (b^2 \* y0);  y2\_bias = a^2 / y0;  fprintf('y2 = %f\*x + %f\n', y2\_k, y2\_bias);  % y2 = 1.0916\*x - 43.2214;    y2\_bias = y2\_bias + n2\_2 \* d / cos(atan(y2\_k));  fprintf('y2\_edited = %f\*x + %f\n', y2\_k, y2\_bias);  % y2\_edited = 1.091554\*x + 208.099009;    % memory clear  clear n2\_1 n2\_2 x0\_2 y0\_2 y0 x0    % 3rd line  % parameter  n3 = 279;    y3\_k = 0;  y3\_bias = n3 \* d;  fprintf('y3 = %f\*x + %f\n', y3\_k, y3\_bias);  % y3 = 0.000000\*x + 77.394600;    % test  %y3\_bias = 0;    % plot  x = -100 : 0.1 : 100;  y1 = y1\_k \* x + y1\_bias;  y2 = y2\_k \* x + y2\_bias;  y3 = y3\_k \* x + y3\_bias;  figure  %axis([-50 50 -50 50])  plot(x, y1);  hold on  plot(x, y2);  %hold on  plot(x ,y3);    % move  % res = [y3\_bias-y2\_bias y3\_bias-y1\_bias] / [y2\_k-y3\_k y1\_k-y3\_k; -2 -2];  res = [y2\_bias/sqrt(y2\_k^2+1)-y1\_bias/sqrt(y1\_k^2+1) y2\_bias/sqrt(y2\_k^2+1)-y3\_bias/sqrt(y3\_k^2+1)]/...      [y1\_k/sqrt(y1\_k^2+1)-y2\_k/sqrt(y2\_k^2+1) y3\_k/sqrt(y3\_k^2+1)-y2\_k/sqrt(y2\_k^2+1);       1/sqrt(y1\_k^2+1)-1/sqrt(y2\_k^2+1) 1/sqrt(y3\_k^2+1)-1/sqrt(y2\_k^2+1)];    x\_cen = res(1);  y\_cen = res(2);  fprintf('x\_cen = %f, y\_cen = %f\n', x\_cen, y\_cen);    % plot  x = -15 : 0.1 : 15;  plot(x, sqrt((1 - x .^ 2 / 15^2) \* 40^2));  plot(x, -sqrt((1 - x .^ 2 / 15^2) \* 40^2));  scatter(x\_cen, y\_cen); |

## Center\_data.m

|  |
| --- |
| %load data.mat data\_2    tmp1 = [];  for i = 1 : 13      tmp1 = [tmp1 data\_2(:, i)];  end    for i = 1 : 13      for j = 371 : 512          tmp1(j, i) = 0;      end  end    for i = 1 : 512      for j = 1 : 13          if tmp1(i, j) ~= 0              tmp1(i, j) = 1;          end      end  end    n1\_1 = sum(tmp1);    n1\_2 = [];  for i = 1 : 13      for j = 512 : -1 : 1          if tmp1(j,i) ~= 0              n1\_2 = [n1\_2 j];              break;          end      end  end  n1\_2 = 512 - n1\_2 + 1;    %%  tmp1=[];  for i = 140 : 152      tmp1 = [tmp1 data\_2(:, i)];  end    for i = 1 : 13      for j = 1 : 80          tmp1(j, i) = 0;      end  end    for i = 1 : 512      for j = 1 : 13          if tmp1(i, j) ~= 0              tmp1(i, j) = 1;          end      end  end    n2\_1 = sum(tmp1);    n2\_2 = [];  for i = 1 : 13      for j = 1 : 512          if tmp1(j, i) ~= 0              n2\_2 = [n2\_2 j];              break;          end      end  end  n2\_2 = 512 - n2\_2 + 1;    %disp(n1\_1);  %disp(n1\_2);  %disp(n2\_1);  %disp(n2\_2);    r = [];  for i = 1 : 13      for j = 1 : 13          tmp3 = cenfunc(n1\_1(i), n1\_2(i), n2\_1(j), n2\_2(j));          if imag(tmp3) == 0              r = [r; tmp3];          end      end  end      figure  scatter(r(:, 1), r(:, 2));  hold on    % scatter(-9.5471, -6.9475);    x\_cen = mean(r(:, 1));  y\_cen = mean(r(:, 2));  scatter(x\_cen, y\_cen);  disp([x\_cen y\_cen]);  axis([-50 50 -50 50]) |

## Mark\_Point.m

|  |
| --- |
| function mark\_point\_matrix = Mark\_Point(projection\_matrix)  mark\_point = zeros(size(projection\_matrix));  for i = 1 : size(projection\_matrix, 2)      if projection\_matrix(1,i) == 0          flag = 0;      else          flag = 1;      end      for j = 2:size(projection\_matrix, 1)          if projection\_matrix(j,i) == 0              next\_flag = 0;          else              next\_flag = 1;          end          if next\_flag ~= flag              mark\_point(j,i) = j;          end          flag = next\_flag;      end  end  mark\_point\_matrix = mark\_point;  end |

## GetScale.m

|  |
| --- |
| function scale = GetScale(length\_projection)  scale\_list = [];  for i = 1 : size(length\_projection, 2)      if length\_projection(1, i) == 0          scale\_pre = 0;      elseif length\_projection(2, i) == 0          scale\_pre = length\_projection(1, i) ./ 28.83;      elseif length\_projection(1, i) <= length\_projection(2, i)          scale\_pre = length\_projection(2, i) ./ 28.83;      else          scale\_pre = length\_projection(1, i) ./ 28.83;      end      scale\_list = [scale\_list, scale\_pre];  end  scale = scale\_list;  end |

## GetLengthOfProjection.m

|  |
| --- |
| function length\_projection = GetLengthOfProjection(mark\_point\_matrix)  length\_vector = zeros(2,size(mark\_point\_matrix, 2));  for i = 1:size(mark\_point\_matrix, 2)      not\_zero = [];      for j = 1:size(mark\_point\_matrix, 1)          if mark\_point\_matrix(j,i) ~= 0              not\_zero = [not\_zero, mark\_point\_matrix(j, i)];          end      end      if isempty(not\_zero)          length\_vector(:, i) = [0; 0];      elseif length(not\_zero) == 2          length\_vector(:, i) = [not\_zero(2) - not\_zero(1); 0];      else          length\_vector(:, i) = [not\_zero(2) - not\_zero(1); not\_zero(4) - not\_zero(3)];      end  end  length\_projection = length\_vector;  end |

## AngleSolve.m

|  |
| --- |
| function angle = AngleSolve(s1)  syms pro\_angle  syms equ  equ = 10 \* sqrt(((64 + 9 \* (tan(pro\_angle))^2)) / (1 + (tan(pro\_angle))^2)) - s1;  angle\_result = roundn(eval(solve(equ, 'pro\_angle')), -4);  %angle\_result = rad2deg(angle\_result)  angle = abs(angle\_result(1));  end |

## AngleMain.m

|  |
| --- |
| function angle = AngleSolve(s1)  syms pro\_angle  syms equ  equ = 10 \* sqrt(((64 + 9 \* (tan(pro\_angle))^2)) / (1 + (tan(pro\_angle))^2)) - s1;  angle\_result = roundn(eval(solve(equ, 'pro\_angle')), -4);  %angle\_result = rad2deg(angle\_result)  angle = abs(angle\_result(1));  end |

## rebuild.m

|  |
| --- |
| % REBUILD Concolution Backprojection (CEP).  %    REBUILD(OriginData, Angles) returns the matrix of absorptance.    function ret = rebuild(OriData, Theta)    % calculate fundamental image data  Num\_Cam = size(OriData, 1);  Num\_Ang = length(Theta);  MidIdx = (Num\_Cam + 1) / 2;  Rad = (pi / 180) \* Theta;  ret = zeros(Num\_Cam, Num\_Cam);    % coordinate settings  [X, Y] = meshgrid(1 : Num\_Cam, 1 : Num\_Cam);  X = X - MidIdx;  Y = Y - MidIdx;    % 1-D filter  FiltData = myfilter(OriData);    % backprojection  for i = 1 : Num\_Ang      % calculate spots address to be added filtered data      Idx = round(MidIdx + X \* sin(Rad(i)) - Y \* cos(Rad(i)));        % accumulate      tmp = zeros(Num\_Cam);      tmpIdx = find((Idx > 0) & (Idx <= Num\_Cam));      tmp(tmpIdx) = FiltData(Idx(tmpIdx), i);      ret = ret + tmp;  end  ret = ret / Num\_Ang;  end |

## myfilter.m

|  |
| --- |
| function f = myfilter(PR)    a = 1;  [Length, Count] = size(PR);  w = [-pi : (2 \* pi) / Length : pi - (2 \* pi) / Length];    rn1 = abs(2 / a \* sin(a .\* w ./ 2));  rn2 = sin(a .\* w ./ 2);  rd = (a \* w) ./ 2;  r = rn1 \* (rn2 / rd)^2;    f = fftshift(r);  for i = 1:Count      IMG = fft(PR(:, i));      fimg = IMG .\* f';      f(:, i) = ifft(fimg);  end  f = real(f);  end |

## CTplot.m

|  |
| --- |
| function ImgData = CTplot(OriData, Theta, x0, y0, d, option)    if option == 1      oc = sqrt(x0^2 + y0^2);      koc = y0 / x0;        for i = 1 : size(OriData, 2)          tmp = round(oc \* sin(Theta(i) - atan(abs(koc))) / d);          if tmp >= 0              for j = size(OriData, 1) : -1 : 1 + tmp                  OriData(j, i) = OriData(j - tmp, i);              end          else              for j = 1 : size(OriData, 1) + tmp                  OriData(j, i) = OriData(j - tmp, i);              end          end      end  end    ImgData = rebuild(OriData, rad2deg(Theta));  figure;  imagesc(ImgData);  end |

## AbCount.m

|  |
| --- |
| function Ab\_data = AbCount(ImgData, Ab\_std)  %ABCOUNT counts the absorption rate(256\*256) of ImgData(512\*512).    Ab\_data = zeros(256);  for i = 1 : 2 : 512      for j = 1 : 2 : 512          Ab\_data((i + 1) / 2, (j + 1) / 2) = ImgData(i, j) + ...              ImgData(i + 1, j) + ImgData(i, j + 1) + ImgData(i + 1, j + 1);      end  end  Ab\_data = (Ab\_data - Ab\_std) / k;    end |

## Count\_Ab\_std.m

|  |
| --- |
| %ImgData = CTplot(data\_2,a,x\_cen,y\_cen,0.2774,1);  Ab\_std= zeros(256);  for i=1:2:512      for j=1:2:512          Ab\_std((i+1)/2,(j+1)/2) = ImgData(i,j)+ImgData(i+1,j)+ImgData(i,j+1)+ImgData(i+1,j+1);      end  end  temp = Ab\_std(Ab\_std>2);  k = sum(temp)/size(temp,1);  Ab\_std = Ab\_std- data\_1\*k; |

## Switch\_projection.m

|  |
| --- |
| function [pro\_after\_switch, pro\_ref] = Switch\_projection(pro, pro\_ref)  if size(pro) ~= size(pro\_ref)      fprintf('different sizes!\n');  else      project = reshape(pro, 1, size(pro,1) .\* size(pro, 2));      project\_ref = reshape(pro\_ref, 1, size(pro\_ref,1) .\* size(pro\_ref, 2));      for i = 1:size(project\_ref, 2)          if project\_ref(i) ~= 0              head\_ref = i;              break          end      end      for j = 1:size(project\_ref, 2)          k = size(project\_ref,2) - j + 1;          if project\_ref(k) ~= 0              tail\_ref = k;              break          end      end      for a = 1:size(project,2)          if project(i) ~= 0              head = i;              break          end      end      for b = 1:size(project, 2)          c = size(project, 2) - b + 1;          if project(c) ~= 0              tail = c;              break          end      end |