Matlab:快速傅里叶变换实现

2020-09-20 |

一、什么是傅里叶变换

简单来说，傅里叶变换将一个函数分解为一组三角函数的和，通过将这个函数分别和这组三角函数中的每一个进行内积运算，可以求得每个三角函数前面的系数。

正交函数集

正交函数集满足如下两条性质：

1. 正交函数集中的任意两个不相同函数的内积为0；
2. 如果某函数集中存在一个函数可以由函数集中除它以外的函数表示，则这个函数集不是正交函数集。

用类比的角度看，这个定义和线性代数中的基向量组的定义一致。说明正交函数集其实相当于函数空间中的一组基向量，函数空间中的一个函数对应向量空间中的一个向量，任何向量可以由基向量表示，同样的，任何函数可以由正交函数集表示。基向量可以写成矩阵的形式，同理，正交函数集也可以写成向量的形式。如下：

[1, sin(wx), cos(wx), sin(2wx), cos(2wx), ..., sin(nwx), cos(nwx), ...]

这是一个1维向量，当前乘一个函数时就可以利用矩阵的乘法将函数分解到这个正交函数集上。

[f(x)] * [1, sin(wx), cos(wx), ..., sin(nwx), cos(nwx), ...] =

[a0, b1*sin(wx), a1*cos(wx), ..., bn*sin(nwx), an*cos(nwx), ...]

注意这些系数通过原函数f(x)和基函数内积等运算得到的。

内积运算

内积定义为向量a在向量b上的投影向量，当内积运算后的结果除以向量b的模长，就可以计算出向量a在向量b上投影占向量b的比例，这个比例就是基函数前面的系数。内积是反映两个向量相似度的重要指标，当两个向量完全相等时，它们的内积取得最大值。因此正交函数集的系数的含义，可以理解为原函数和基函数的相似度，相似度越大，这个基函数的系数越大。

因此，总体上来看，正交函数集构成了一个无穷维数的函数空间，这个空间里的任何函数都可以由这些基函数的加权求和表示，权重就是原函数和积函数的内积，及将原函数投影到这些正交函数上。当然可以用欧拉公式将这些三角基函数合并指数基函数，只不过这些基函数建立在复数域内了，后面的快速傅里叶实现也采用复数域内的正交函数集。

二、离散傅里叶变换

第一部分中讲到傅里叶变换是连续函数空间里的，如果处理的函数是一个离散的函数，那么就要用到离散傅里叶变换。离散傅里叶变换与来连续傅里叶变换的最大区别是求系数时的积分运算变成了求和运算。将正交函数集的向量形式抄写如下：

[1, sin(wx), cos(wx), sin(2wx), cos(2wx), ..., sin(nwx), cos(nwx), ...]

注意，这里的每一列都是一个连续函数，如果用离散的x坐标将其离散化，就可以获得离散傅里叶变换的正交函数集矩阵，如下：

$$\begin{array}{cccccccccc}
1 & \sin(u\times 0) & \cos(u\times 0) & \sin(2u\times 0) & \cos(2u\times 0) & \dots & \sin(nu\times 0) & \cos(nu\times 0) & \dots \\
1 & \sin(u\times 1) & \cos(u\times 1) & \sin(2u\times 1) & \cos(2u\times 1) & \dots & \sin(nu\times 1) & \cos(nu\times 1) & \dots \\
\dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\
1 & \sin(u\times x) & \cos(u\times x) & \sin(2u\times x) & \cos(2u\times x) & \dots & \sin(nu\times x) & \cos(nu\times x) & \dots \\
\dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots
\end{array}$$

这是一个二维矩阵，每一列代表一个正交函数，每一行坐标代表一个离散坐标x。将离散函数写成向量的形式后就可以用矩阵的乘法求出变换后的函数。这时有人会注意到，这个离散的矩阵两个维度都是无穷的，如果离散信号不一定是无穷的，那怎么运算呢？我们分别讨论一下：

1. 变换矩阵的行数
   如果要满足矩阵乘法定义，那么变换矩阵的行的个数必须和输入信号的变量取值范围相同，及与x的取值范围一致。如果输入信号的取值范围是无穷的，那么变换矩阵的行也有无数个。
2. 变换矩阵的列数
   由于变换矩阵的列数没有严格的定义，可多可少，但是由于每一列代表一种基函数，基函数的个数决定了表达的误差，一般来说和输入变量的x取值范围相同即可。

三、一维快速傅里叶变换的原理

快速傅里叶变换有许多种，这里介绍的快速傅里叶变换通过将冗余的指数运算记录下来，以此减少乘法次数，加快运算速度。一维离散傅里叶公式如下：

$$F(u)=\frac{1}{N}\sum_{x=0}^{N-1}f(x)\bullet e^{(-j2\pi ux/N)}$$

能否找到冗余计算？看样子很难，这里直接给出简化算法的核心公式：

$$e^{(-j2\pi 2ux/2M)}=e^{(-j2\pi ux/M)}$$

这个公式其实很简单，本质就是一个约分，但是这个约分给我们提供了分离傅里叶变换的思路——奇偶分离。当一维傅里叶变换的输入信号的变量取值范围是2的整数倍，那么一维傅里叶变换的求和可以分为奇数部分求和、偶数部分求和，如下公式所示：

$$\begin{aligned}F(u)&=\frac{1}{2M}\sum_{x=0}^{2M-1}f(x)e^{\left(-j2\pi u\frac{x}{2M}\right)}\\&=\frac{1}{2}\left[\frac{1}{M}\sum_{x=0}^{M-1}f(2x)e^{\left(-j2\pi u\frac{2x}{2M}\right)}+\frac{1}{M}\sum_{x=0}^{M-1}f(2x+1)e^{\left(-j2\pi u\frac{2x+1}{2M}\right)}\right]\\&=\frac{1}{2}\left[\frac{1}{M}\sum_{x=0}^{M-1}f(2x)e^{\left(-j2\pi u\frac{x}{M}\right)}+\frac{1}{M}\sum_{x=0}^{M-1}f(2x+1)e^{\left(-j2\pi u\frac{x}{M}\right)}e^{\left(-j2\pi u\frac{1}{2M}\right)}\right]\end{aligned}$$

看出来了吗？这里奇数部分和偶数部分有相同的分量，取值范围都是从0到M-1，你能看出来它也是傅里叶变换公式吗？下面这个公式将更加清晰：

$$\begin{aligned}&令g(x)=f(2x),h(x)=f(2x+1),则\\&F(u)=\frac{1}{2}\left[\frac{1}{M}\sum_{x=0}^{M-1}g(x)e^{\left(-j2\pi u\frac{x}{M}\right)}+\frac{1}{M}\sum_{x=0}^{M-1}h(x)e^{\left(-j2\pi u\frac{x}{M}\right)}e^{\left(-j2\pi u\frac{1}{2M}\right)}\right]\end{aligned}$$

其中g(x)部分是原函数中自变量x取偶数时组成的新函数，h(x)是原函数自变量取奇数时组成的函数，将他们看成新的原函数，那么奇数部分和偶数部分就都是新的一维离散傅里叶变换了。这说明，原函数自变量取值范围是偶数时，原函数的傅里叶变换可以由自变量是偶数部分的傅里叶变换和自变量是奇数部分的傅里叶变换组合而成，并且这两部分互不影响。这是一个重要的结论，为后面的递归提供依据。为了方便书写，我们将偶数部分的傅里叶变换记作$F_{even}(u)$，将奇数部分的傅里叶变换记作$F_{odd}(u)$，那么一维傅里叶变换可以写成如下公式：

$$F(u)=\frac{1}{2}\left[F_{even}(u)+F_{odd}(u)e^{-j2\pi u/2M}\right]$$

必须要注意的是，上述公式的奇数部分和偶数部分都是N/2个，那么根据一般变换后自变量取值范围和原函数相同，可以知道u的取值范围从原来的N变成了N/2，那么后面一半怎么补全呢？我们再来看两个核心公式：

$$\begin{aligned}e^{-j2\pi\frac{u+M}{M}}&=e^{-j2\pi\frac{u}{M}}\\e^{-j2\pi\frac{u+M}{2M}}&=-e^{-j2\pi\frac{u}{2M}}\end{aligned}$$

将这两个公式带入F(u+M)，就可以算出后半部分其表达式。

四、二维傅里叶变换的性质

第三部分讲到一维离散傅里叶快速变换的基本原理，但对于二维快速傅里叶变换这还不够，我们需要探讨一下二维傅里叶变换的基本性质，才能将一维的变换应用到二维的求解中。在讨论二维傅里叶变换的性质之前，我们给出其定义公式：

$$F(u,v)=1/MN\sum_{x=0}^{M-1}\left[\sum_{y=0}^{N-1}f(x,y)\exp(-j2\pi vy/N)\right]\exp(-j2\pi ux/M)$$

可分离性

根据二维傅里叶变换的公式，可以比较轻松地看出，中括号里的计算是不涉及变量x的，换句话说，中括号里的计算是需要事先给定x的值，然后再计算，x在这里相当于一个常数，因此可以将中括号里面和外面的计算分离。这样，一个二维的傅里叶变换就变成了两个一维傅里叶变换：

$$F(x,v)=\frac{1}{N}\sum_{y=0}^{N-1}f(x,y)\,\mathrm{e}^{-\mathrm{j}2\pi\frac{vy}{N}}$$

$$F(u,v)=\frac{1}{N}\sum_{x=0}^{N-1}F(x,v)\,\mathrm{e}^{-\mathrm{j}2\pi\frac{ux}{N}}$$

平移性

二维傅里叶变换的平移性质和一维傅里叶变换的平移性质一脉相承，只不过一维中讲的是时域的平移，二维中讲的是空域的平移。这个性质在之后对图像的二维傅里叶变换中将要用到。

空域平移：

$$f(x-x_0,y-y_0)\Leftrightarrow F(u,v)\exp\left[-j2\pi(ux_0+vy_0)/N\right]$$

频域平移：

$$f(x,y)\exp\left[j2\pi(u_0x+v_0y)/N\right]\Leftrightarrow F(u-u_0,v-v_0)$$

五、二维快速傅里叶变换的实现

原理和性质都讲完了，接下来讲讲如何实现二维快速傅里叶变换。

本次使用软件MATLAB R2017a

文件结构

```
main
|
|-image input
|
|-myft3
| |
| |-myfft
| |(recursion)
|
|-shift myft3
|
|-ifft2
```

图像输入

```
origImg = imread('2.jpg');
grayImg = rgb2gray(origImg); %将图像转变成灰度图
figure(1)
imshow(grayImg);
```

二维快速傅里叶变换

```
ftImg = myft3(grayImg);
figure(2)
imshow(abs(log(ftImg)+1),[]);
```

由于傅里叶变换的结果是一个complex double类型的矩阵，直接使用imshow函数的化将会造成无法正确输出的情况，所以需要对傅里叶变换后的矩阵进行log(x)+1的灰度变换，将矩阵的取值范围限制在合理范围，并且取模长使第二个参数[]可以自动调整输出灰度范围。
接下来详细介绍一下快速傅里叶变换的具体实现。

```
function [ outputMat ] = myft3( inputMat )

    [M0,N0] = size(inputMat); %获得输入图像的大小

    %找到最接近输入图像大小（并且大于图像大小）的2^n，并将图片padding补零
    M1 = find_upper_2n(M0);
    N1 = find_upper_2n(N0);
    outputMat = double(zeros(M1,N1));
    tempMat = double(zeros(M1,N1));
    inputMat(M1,N1) = 0;
    inputMat = double(inputMat);

    %行列分开计算
    for x = 1:N1
        tempMat(:,x) = myfft( inputMat(:,x).' ).';
    end
    for y = 1:M1
        outputMat(y,:) = myfft( tempMat(y,:) );
    end

end
```

上一段是进入快速傅里叶变换的铺垫步骤，利用padding补零，使得后续一维傅里叶快速变换可以最大限度地递归进行，最大限度地减少重复计算。而padding补上的零值在傅里叶变换的求和中是没有起作用的，因此不影响原图片的傅里叶变换的值。

一维快速傅里叶变换

由第三部分可知，当输入函数的自变量取值范围是偶数时，输入函数的傅里叶变换可以由偶数部分和奇数部分各自的傅里叶变换通过简单组合得到，因此，我们采取递归的方式不断拆分输入函数，直到输入函数的自变量只能取1个值。

```
function [ outputMat ] = myfft( inputMat )

    [M,N] = size(inputMat);  %获得输入一维矩阵（向量）的列数N，行数M没有用

    if mod(N,2)==0  %当输入向量的长度可以拆分时进行递归
        oddMat = myfft( inputMat(1:2:end) );
        evenMat = myfft( inputMat(2:2:end) );
        for u=1:N/2
            evenMat(u) = evenMat(u)*exp(-1j*2*pi*(u-1)/N);
        end
        foreMat = (oddMat + evenMat);
        backMat = (oddMat - evenMat);
        outputMat = cat(2,foreMat,backMat);
    else  %当输入向量长度为1时停止递归，返回单点傅里叶变换的值，及自身
        outputMat = inputMat;
    end

end
```

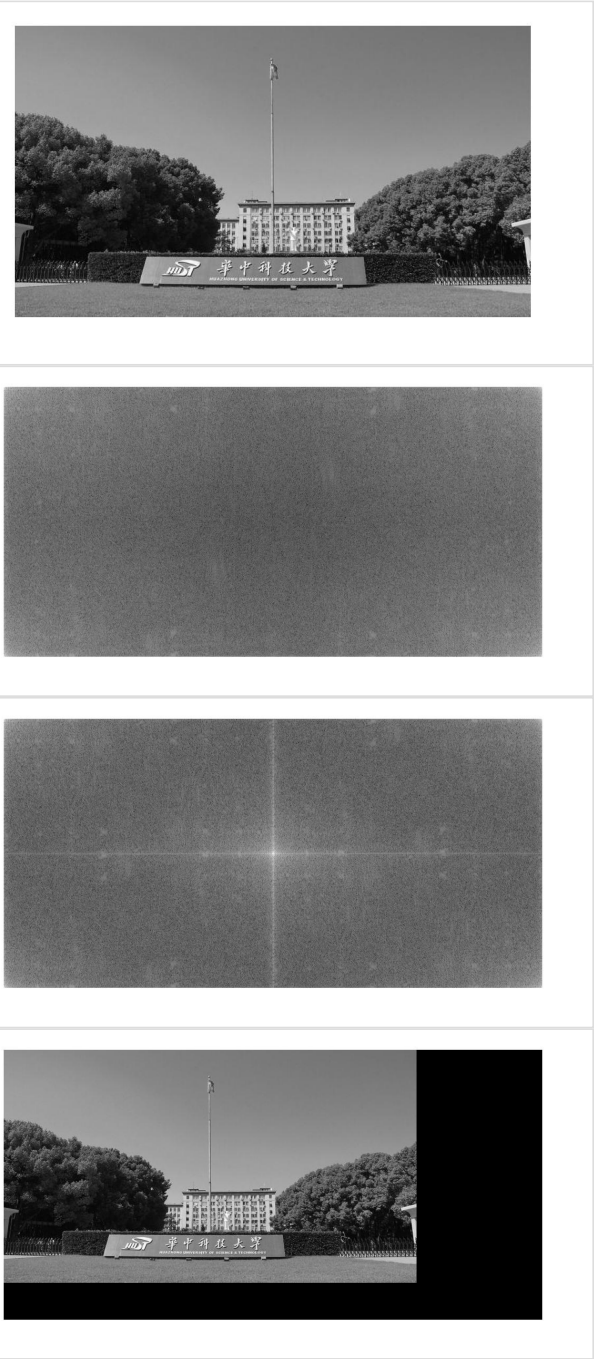
其实不需要完全分到1个点再进行傅里叶变换，可以适当提前结束递归。

二维快速傅里叶反变换

```
iftImg = abs(ifft2(ftImg));
figure(3)
imshow(iftImg,[]);
```

这里反变换是利用Matlab提供的函数ifft2进行验证，如果正确的话，结果应该是原图片加上黑色边框。

实验结果



Matlab

Keil5安装芯片包