# Project Name

| 序号 | 学号 | 专业班级 | 姓名 | 性别 |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |

## 1. Project Introduction

内容包括：（这部分内容不要太长，讲清楚即可）

（1） 开发环境及系统运行要求，包括所用的开发工具、开发包、开源库、系统运行要求等；

开发环境：Windows10；

系统运行要求：Python 3.6；

开发工具：Pycharm；

开源库：matplotlib + numpy；

（2） 工作分配简介，即谁要做什么事情

无分组，独立完成

## 2. Technical Details

内容包括：

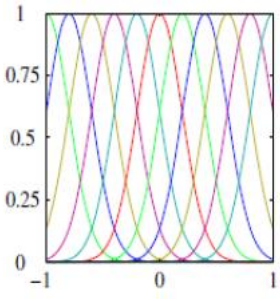
（1） 工程实践当中所用到的理论知识阐述

1. 主要思想：本题的主要思想是通过**线性回归**进行图相恢复。本次实验采用了**高斯线性回归**的方法。本质上是把图片理解成为由多个（实验中用了 50 个 Phi）不同 mean 相同方差的**高斯函数组合**的结果。然后通过没有被噪音污染的像素值，用最小二乘法算出对应每一行的 Phi 的权重（即每一行的 50 个函数的权重），最后再用这个权重和被污染像素的分度密度进行计算。

| | | | | |
|---|---|---|---|---|
| 1,1 | ~~1,2~~ | 1,3 | ~~1,4~~ | 1,5 |
| ~~2,1~~ | 2,2 | ~~2,3~~ | 2,4 | ~~2,5~~ |

*划掉的代表是被污染的

1. 假设有 3 个高斯函数构成了每一行的图相也就是说对于每一个像素都是 wi*Phii（xi) i=1,2,3 即每一行每个高斯函数对应的权重*每个高斯函数 xi 时的概率；




2. 然后可以通过最小二乘法，利用没有被污染的像素，也就是说正确的值来计算出权重 wi；

3. 再用这个权重 wi*每个高斯函数 xi（被污染的像素）来恢复图相；

| | 像素对应的密度分布函数 | Phi1 | Phi2 | Phi3 |
|---|---|---|---|---|
| 像素对应的 x 坐标 | 对应坐标在对应高斯分布函数中的密度值 | | | |
| X1 | | | | |
| X2 | | | | |
| … | | | | |
| X3 | | | | |

2. Basis Function 高斯函数：

There are many other possible choices for the basis functions, for example

$$\phi_j(x) = \exp\left\{-\frac{(x-\mu_j)^2}{2s^2}\right\} \tag{3.4}$$

where the $\mu_j$ govern the locations of the basis functions in input space, and the parameter $s$ governs their spatial scale. These are usually referred to as 'Gaussian' basis functions, although it should be noted that they are not required to have a probabilistic interpretation, and in particular the normalization coefficient is unimportant because these basis functions will be multiplied by adaptive parameters $w_j$.

3.最小二乘法：

## Maximum likelihood and least squares

- Assume: $t = y(\mathbf{x}, \mathbf{w}) + \epsilon$ $\quad y(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x})$
- Thus:

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}) \quad \Rightarrow \quad \mathbb{E}[t|\mathbf{x}] = \int t p(t|\mathbf{x})\, \mathrm{d}t = y(\mathbf{x}, \mathbf{w})$$

- For data set $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ and target vector $\mathbf{t} = (t_1, \ldots, t_N)^{\mathrm{T}}$, the likelihood function:

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n|\mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n), \beta^{-1})$$

$$\ln p(\mathbf{t}|\mathbf{w}, \beta) = \sum_{n=1}^{N} \ln \mathcal{N}(t_n|\mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n), \beta^{-1}) = \frac{N}{2} \ln \beta - \frac{N}{2} \ln(2\pi) - \beta E_D(\mathbf{w})$$

SSE: sum-of-squares error function

$$E_D(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \{t_n - \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n)\}^2 = \frac{1}{2} \|\mathbf{t} - \boldsymbol{\Phi}\mathbf{w}\|^2$$

## Maximum likelihood and least squares

$$\ln p(\mathbf{t}|\mathbf{w}, \beta) = \sum_{n=1}^{N} \ln \mathcal{N}(t_n|\mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n), \beta^{-1}) = \frac{N}{2} \ln \beta - \frac{N}{2} \ln(2\pi) - \beta E_D(\mathbf{w})$$

$$E_D(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \{t_n - \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n)\}^2 = \frac{1}{2} \|\mathbf{t} - \boldsymbol{\Phi}\mathbf{w}\|^2 \qquad \mathbf{w} = (w_0, \ldots, w_{M-1})^{\mathrm{T}}$$

- Solving w by ML:

$$\nabla \ln p(\mathbf{t}|\mathbf{w}, \beta) = \sum_{n=1}^{N} \{t_n - \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n)\} \boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}}$$

$$0 = \sum_{n=1}^{N} t_n \boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}} - \mathbf{w}^{\mathrm{T}} \left( \sum_{n=1}^{N} \boldsymbol{\phi}(\mathbf{x}_n) \boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}} \right) \quad \Rightarrow \quad \mathbf{w}_{\mathrm{ML}} = \left(\boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^{\mathrm{T}} \mathbf{t}$$

$$\boldsymbol{\Phi} = \begin{pmatrix} \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_1) & \cdots & \phi_{M-1}(\mathbf{x}_1) \\ \phi_0(\mathbf{x}_2) & \phi_1(\mathbf{x}_2) & \cdots & \phi_{M-1}(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}_N) & \phi_1(\mathbf{x}_N) & \cdots & \phi_{M-1}(\mathbf{x}_N) \end{pmatrix} = \begin{pmatrix} \boldsymbol{\phi}(\mathbf{x}_1)^{\mathrm{T}} \\ \boldsymbol{\phi}(\mathbf{x}_2)^{\mathrm{T}} \\ \vdots \\ \boldsymbol{\phi}(\mathbf{x}_N)^{\mathrm{T}} \end{pmatrix}$$

*N×M design matrix*

$\boldsymbol{\Phi}^{\dagger} \equiv \left(\boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^{\mathrm{T}}$ *Moore-Penrose pseudo-inverse*

（2） 具体的算法，请用文字、示意图或者是伪代码等形式进行描述（不要贴大段的代码）

1. Mask 掩码：

通过判断污染图相的像素是否为 0，如果是 0 的话则说明已经被污染，如果不是 0 的话则说明是正确像素。

```
# noise mask
mask = (corrImg != 0)      # correct pixel
```

```
noiseMask = np.zeros((rows, cols, channels))
noiseMask[mask] = 1          # true: 1, false: 0
```

2. 高斯函数均值和方差：

   本实验的高斯函数均值和 x 坐标正相关，方差则都为 0.01。

```
Phi_mu = np.linspace(start = 0, stop = cols-1, num = basisNum, endpoint = True) / (cols-1)
# here we set the standard deviation to the same value for brevity
Phi_sigma = sigma * np.ones((1, basisNum))
```

3. 计算正确像素点对应的密度：

   在本实验中使用了 50 个 basis function，即整张图片由 50 个均值不同方差相同的高斯函数线性组合而成。

```
# compute the coefficients / weight
Phi = np.hstack((np.ones((ddNum, 1)), np.zeros((ddNum, basisNum-1))))
for j in range(1, basisNum):
    Phi[:, j] = normPdf(np.transpose(x[ddIdx[0]]), Phi_mu[j-1], Phi_sigma[0][j-1])
```

4. 最小二乘法求得权重：

   通过最小二乘法求得 50 个 basis function 对应的权重。

```
w = np.dot(np.dot(np.linalg.inv(np.dot(np.transpose(Phi), Phi)), np.transpose(Phi)), np.transpose(corrImg[i, ddIdx[0], k]))
```

5. 计算污染像素点对应的密度：

```
# restore the missing values
Phi1 = np.hstack((np.ones((misNum, 1)), np.zeros((misNum, basisNum - 1))))
for j in range(1, basisNum):
    Phi1[:, j] = normPdf(np.transpose(x[misIdx[0]]), Phi_mu[j - 1], Phi_sigma[0][j - 1])
```

6. 利用权重和对应坐标的密度函数恢复图像：

```
resImg[i, misIdx[0], k] = np.dot(np.transpose(w), np.transpose(Phi1))
```

7. 一元高斯分布：

```
def normPdf(x, mu, sigma):
    return(np.exp(-(x - mu) ** 2 / (2 * sigma ** 2)))
```

（3） 程序开发中重要的技术细节，比如用到了哪些重要的函数？这些函数来自于哪些基本库？功能是什么？自己编写了哪些重要的功能函

数？等等

1. 编写了对应的一元高斯函数函数，从而可以使得部分计算简化。

```
def normPdf(x, mu, sigma):
    return(np.exp(-(x - mu) ** 2 / (2 * sigma ** 2)))
```

2. 在本实验中，图片的读入，显示和输出都使用了 matplotlib 库。

（1） 图片的读入：

```
# load corrupted img
    pathName = '../data/'
    corrImg = mpimg.imread(pathName + testName +'.png')
```

（2） 图片的显示：

```
# show the corrupted img and restored img
    plt.imshow(corrImg)
    plt.show()
    plt.imshow(resImg)
    plt.show()
```

（3） 图片的存储:

```
# store img
    plt.imshow(resImg)
    plt.axis('off')
    plt.savefig(pathName + testName +'_restored.png')
```

3. 要注意 python 的矩阵是从 0 开始计数的，而在 matlab 中则是从 1 开始计数的。因此在坐标分布的时候需要注意。

# 3. Experiment Results

用图文并茂的形式给出实验结果，如系统界面、操作说明、运行结果等，并对实验结果进行总结和说明。

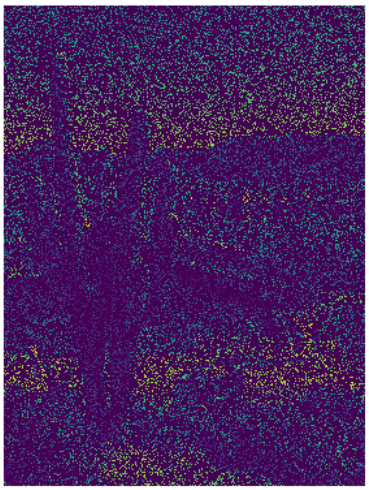
1. 操作说明：直接运行 main.py 文件。

（1）修改文件名字：修改 main 函数第一行的 testName 变量；

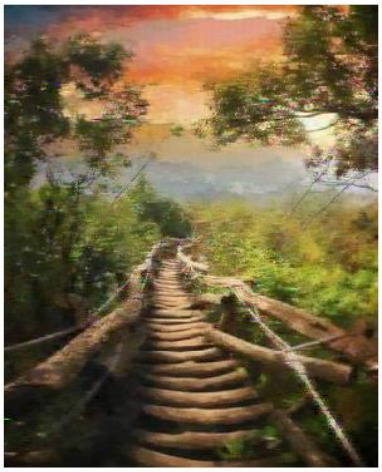
（2）修改文件污染率：修改 main 函数第二行的 noiseRatios 变量；

2. 运行结果：修复的图片将直接存储在 data 文件夹中

（1）



（2）



（3）

## References:

给出主要的参考文献，可以是论文、网站、书籍、别人的技术报告等。


1. 线性回归： https://www.cnblogs.com/GuoJiaSheng/p/3928160.html
2. 高斯过程回归： https://zhuanlan.zhihu.com/p/24388992
3. 最小二乘法： https://www.cnblogs.com/lyrichu/p/7814651.html
4. 题目解析： https://blog.csdn.net/Woolseyyy/article/details/72663125


备注：

代码中请给出较为详细的注释，此报告中切勿粘贴大量代码，否则扣分。