哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称:	机器学习
课程类型:	必修
实验题目:	多项式拟合正弦函数

姓名:	黄海
学号:	1160300329

一、实验目的

机器学习第一次试验 多项式拟合正弦曲线

通过一系列的数学方法 多次实验进行拟合操作 获得最终的拟合结果并加以解释

实现功能：

- [x] 1. 生成数据，加入噪声；
- [x] 2. 用高阶多项式函数拟合曲线；
- [x] 3. 用解析解求解两种loss的最优解（无正则项和有正则项）
- [x] 4. 优化方法求解最优解（梯度下降，共轭梯度）；
- [x] 5. 用你得到的实验数据，解释过拟合。
- [x] 6. 用不同数据量，不同超参数，不同的多项式阶数，比较实验效果。
- [x] 7. 语言不限，可以用matlab，python。求解解析解时可以利用现成的矩阵求逆。 梯度下降，共轭梯度要求自己求梯度，迭代优化自己写。不许用现成的平台，例 如pytorch，tensorflow的自动微分工具。

二、实验要求及实验环境

实验要求

- 充分实现“实现功能中的每一个要求”
- 不可使用现有的自动梯度/微分工具 要求自行编写

实验环境

- macOS 10.13.6
- python 3.7.0
- PyCharm 2018.2.4 (Professional Edition) Build #PY-182.4505.26, built on September 19, 2018 Licensed to George Huang Subscription is active until August 7, 2019 For educational use only. JRE: 1.8.0_152-release-1248-b8 x86_64 JVM: OpenJDK 64-Bit Server VM by JetBrains s.r.o macOS 10.13.6

三、设计思想(本程序中的用到的主要算法及数据结构)

1.算法原理

解析解（有无正则项）求解最优情况

对于多项式拟合算法，我们更加倾向于将其看作为m元线性方程组的求解。 方程组可以看作为

$$h(x,w)=w_0+w_1x+w_2x^2+\cdots+w_{m-1}x^{m-1} \tag{1}$$

即

$$h(x,w)=W^TX \tag{*}$$

其中

$$W=\begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_{m-1} \end{bmatrix},X=\begin{bmatrix} 1 \\ x \\ x^2 \\ x^3 \\ \vdots \\ x^{m-1} \end{bmatrix}$$

对于拟合，我们希望得到的函数能够尽可能的经过我们预设的样本点。那么我们必须有某个评判拟合偏差的标准$J(W)$，结果越大则偏差越大。

对于相当的样本容量时，$J(w)$总是能够很好的拟合出我们希望看到的结果。但是对于少量的样本，我们可能会出现拟合过度的情况，这是我们需要加入正则惩罚来平衡模型复杂度。

无正则惩罚时

$$\begin{aligned} J(w)&=\sum_{i=1}^{m}(h(x_i,w)-y_i)^2 \\ &=(XW-y)^T(XW-y) \end{aligned}$$

最小化$J(w)$，对其求导得到解析解

$$W=(X^TX)^{-1}X^Ty \tag{3}$$

有正则惩罚时

$$\begin{aligned} J(w)&=\sum_{i=1}^{m}(h(x_i,w)-y_i)^2+\lambda\|w\|_2 \\ &=(XW-y)^T(XW-y)+\lambda(W^TW)^{\frac{1}{2}} \end{aligned}$$

解析解为

$$W=(X^TX+\lambda I)^{-1}X^Ty \tag{4}$$

梯度下降(SDG和CG)

在进行有正则项的匹配时，正则项的选取比较困难，此时我们选取梯度下降的方法来进行求解。

首先是随机梯度下降SDG。SDG相较于批梯度下降，有着速度快、容易找到全局最优的特点。但是就具体问题而言效果较差。

对于所有的梯度下降法，都可以使用同样的公式来表示。$h_\theta(x)$表示参数的方程，考虑线性回归，表述为

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_{m-1} x_{m-1}$$

使用$h_\theta(x)$来表示预测值，用y表示期望值。那么$h_\theta - y$表示误差

对于m组数据，我们可以调整参数θ来使得

$$min \frac{1}{2} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

而对于所有的样本，误差最小时的θ就是结果，我们将上式称作$J(\theta)$

通过不断的更改θ使$J(\theta)$变小，而其更新函数为

$$\theta_i = \theta_i - \alpha \frac{\partial}{\partial \theta} J(\theta) \tag{5}$$

其中α是学习速率参数

对于每一组的偏差进行叠加，于是更新函数为

$$\theta_i = \theta_i - \alpha \sum_{j=1}^{m} (h_\theta(x^{(j)}) - y^{(j)}) x_i^{(j)}$$

对于大数据量训练，求和很不友好。那么我们随机从其中选取一个参数来进行训练。这样精度可能不高，但是整体趋势依旧是走向最小。此时更新公式

$$\theta_i = \theta_i - \alpha (h_\theta(x^j) - y^{(j)}) x_i^{(j)} \tag{6}$$

这便是最后的参数更新函数。这与解析解并不矛盾。

在本问题的条件下，梯度函数和更新函数为

$$\frac{\partial J}{\partial w} = 2X^T X W - 2X^T y + \lambda (W^T W)^{-\frac{1}{2}} W$$

$$W = W - \gamma \frac{\partial J}{\partial w} \tag{7}$$

而对于共轭梯度，我们将其看作为迭代法来进行训练。

我们从x_0处开始搜索解决方案，在每次迭代中我们需要一个指标来评判是否更接近最优解。而最优解也是一下二次函数的唯一最小值

$$f(x) = \frac{1}{2} X^T A X - X^T b$$

我们取下降梯度方向为

$$p_0 = b - A x_0$$

在此基础上其他向量均与梯度共轭。

让r_k作为第k步的残差

$$r_k = b - Ax_k$$

下降方向沿着r_k移动

给出以下表达式

$$p_k = r_k - \sum_{i<k} \frac{p_i^T A r_k}{p_i^T A p_i} p_i$$

按照这个方向，下一个最佳位置由下式给出

$$x_{k+1} = x_k + \alpha_k p_k$$

其中

$$\alpha_k = \frac{p_k^T b}{p_k^T A p_k} = \frac{p_k^T (r_k + A x_k)}{p_k^T A p_k} = \frac{p_k^T r_k}{p_k^T A p_k}$$

p_k和r_k共轭，等式成立。

这里对应的有

$$A = X^T X + \lambda I \tag{8.1}$$

$$b = X^T y \tag{8.2}$$

2.算法的实现

样本集生成

样本集通过正弦函数加高斯噪声来生成，在正弦函数一个周期中均匀产生N个数据点。

$$y^i = sin(x^i) + e$$

e服从$N(0,1), i \leq N, i \geq 1, (x^i, y^i)$表示第$i$个数据

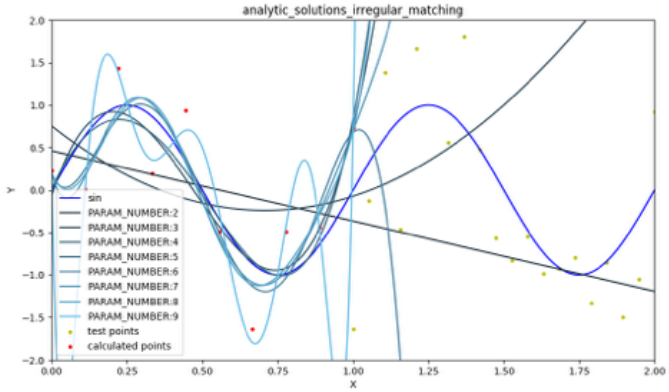
对于解析解，通过调用矩阵相关的运算来实现。以下为核心的计算过程

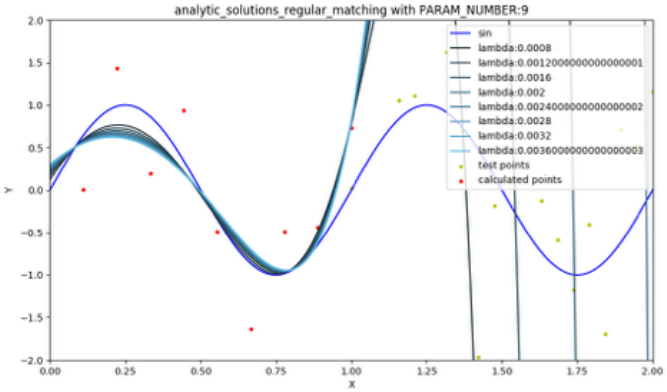
```
def analytic_solutions_irregular_matching(a, result):

    """
    解析解的无正则计算函数 通过现有的公式进行计算 得出未经过调整的参数结果
    :param a: 参数为最终计算的列矩阵维度 必须为大于1的正整数
    :param result: 样本点矩阵 必须是2*N维度
    :return: 返回参数列矩阵
    """
```

```
    size = SAMPLE_NUMBER
    x = np.transpose(result[0])
    y = np.transpose(result[1])
    X = np.ones((size, 1))
    for i in range(1, a):
        X = np.hstack((X, np.power(x, i)))
    Xt = np.transpose(X)
    W = (Xt * X).I * Xt * y
    return W



def analytic_solutions_regular_matching(a, result):

    """
    解析解的无正则计算函数 通过现有的公式进行计算 得出未经过调整的参数结果
    :param a: 参数为最终计算的列矩阵维度 必须为大于1的正整数
    :param result: 样本点矩阵 必须是2*N维度
    :return: 返回参数列矩阵
    """
    size = SAMPLE_NUMBER
    x = np.transpose(result[0])
    y = np.transpose(result[1])
    X = np.ones((size, 1))
    for i in range(1, a):
        X = np.hstack((X, np.power(x, i)))
    Xt = np.transpose(X)
    i = np.eye(a)
    W = (Xt * X + LAMBDA * i).I * Xt * y
    return W
```

而梯度下降是手工实现。

四、实验结果分析

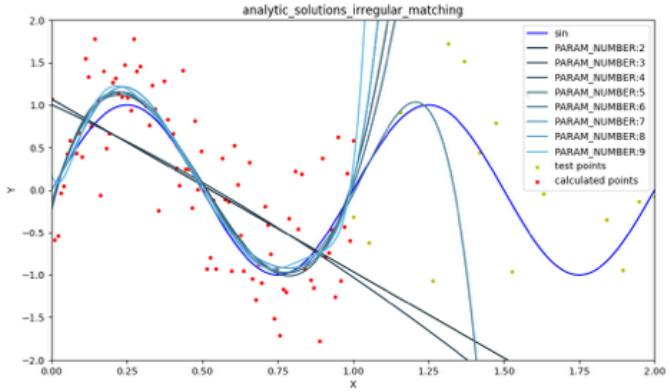


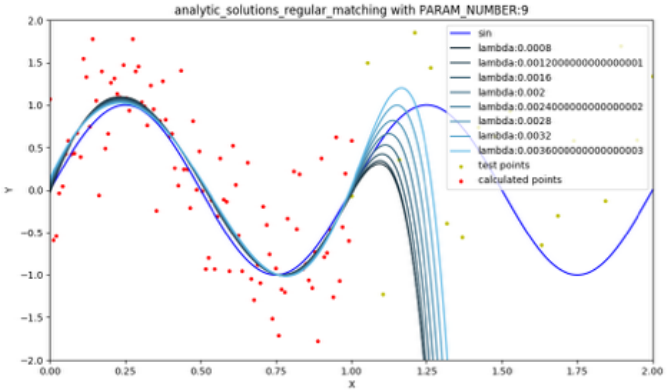


结果展示了纯解析解进行拟合时的情况。第一张图是无正则项在变换维度时产生的结果。显然对于2到8维在0到1.0的范围上拟合恰当，9维时发生了过拟合的现象。

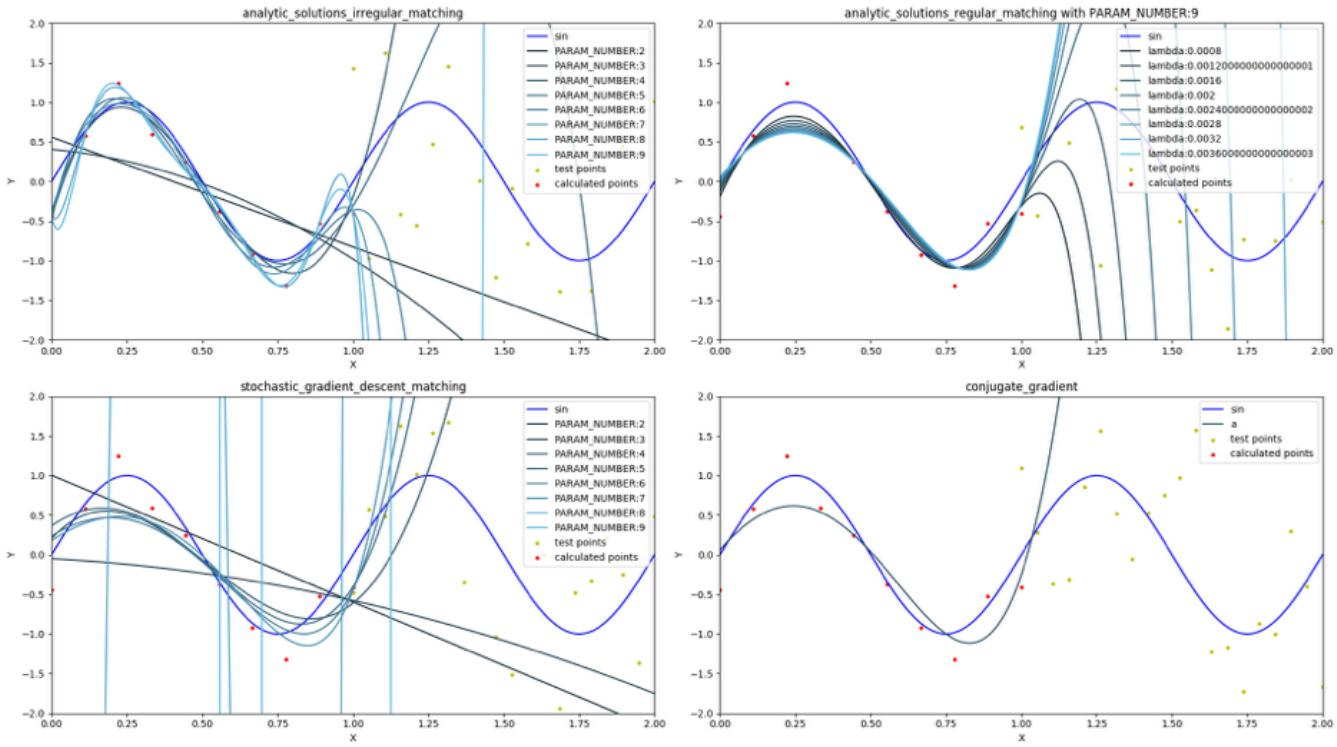
第二张图是更改正则惩罚后产生的结果 可以看出在更改结果的过程中过拟合现象稍微缓解，经过测试在0.0006左右时拟合较好但是无法在测试点中进行拟合

下面是拓展到100个样本点时的情况 很明显在样本区间都拟合的很好，可以得出结论 ***在条件不变的情况下，拓展样本可以起到矫正的作用***





下面是采取SDG和CG的情况。可以明显看到，在加入了梯度后，曲线拟合效果较好。并且共轭梯度的迭代次数极小。



五、结论

我们从实验数据可以看出，在具有充分样本的情况下，模型的学习效果最好，对于样本的拟合最佳。在样本数量一定的情况下，维度过高回导致过学习的情况。并且在进行梯度下降的过程中会导致学习过快，参数设置不够完美，导致无法正常匹配。对于各种参数的管理，通过实验可以得出以下的结论：

- 样本数量要足够大，这样在简单的学习后便可以较好的拟合数据
- 梯度下降最好使用交叉验证，不断的自适应调整学习步长和临界精度
- 共轭梯度在$PARAM_NUMBER$次迭代后可以达到理论最优结果

六、参考文献

[1]M.Jordan,Pattern recognition and machine learning[M]

七、附录:源代码(带注释)

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import matplotlib.pyplot as plt
import numpy as np

"""
机器学习第一次试验
```

```
实现功能:
-[x] 1. 生成数据，加入噪声;
-[x] 2. 用高阶多项式函数拟合曲线;
-[x] 3. 用解析解求解两种loss的最优解（无正则项和有正则项）
-[x] 4. 优化方法求解最优解（梯度下降，共轭梯度）;
-[x] 5. 用你得到的实验数据，解释过拟合。
-[x] 6. 用不同数据量，不同超参数，不同的多项式阶数，比较实验效果。
-[x] 7. 语言不限，可以用matlab，python。求解解析解时可以利用现成的矩阵求逆。
    梯度下降，共轭梯度要求自己求梯度，迭代优化自己写。不许用现成的平台，例
    如pytorch，tensorflow的自动微分工具。
"""

# 测试点区间
TEST_START = 1
TEST_END = 2

# 正则项惩罚参数
LAMBDA = 0.0004

# 计算点区间
CALCULATE_START = 0
CALCULATE_END = 1

# 参数维度
PARAM_NUMBER = 9

# 样本点数
SAMPLE_NUMBER = 10

# 测试点数
TEST_POINTS_NUMBERS = 20

# 梯度下降系列参数
LOOP_MAX = 10000
EPSILON = 0.8
ALPHA = 0.2


def random_point(start, end, a):
    """
    样本点生成函数 通过定义生成区间和生成个数 返回样本点矩阵
    :param start: 正数 必须小于end
    :param end: 正数 必须大雨start
    :param a: 样本点个数 必须是正整数
    :return: 返回2*N的样本点矩阵
    """
    array_x = np.linspace(start, end, a)
    # array_y0 = np.sin(2 * np.pi * array_x) + np.random.randn(a)
    array_y0 = np.sin(2 * np.pi * array_x) + np.random.normal(scale=0
.5, size=array_x.shape)
```

```
    # print(array_x)
    # print(array_y0)

    result = np.vstack((np.mat(array_x), np.mat(array_y0)))
    # print("result:")
    # print(result)
    return result

    # figure = plt.figure()
    # ax = figure.add_subplot(111)
    #
    # ax.set_title("Figure1")
    #
    # plt.xlabel("X")
    # plt.ylabel("Y")
    #
    # plt.legend("x1")
    #
    # ax.scatter(array_x, array_y0, c="r", marker=".")
    #
    # plt.show()


def random_point_test(start, end, a):
    """
    测试点生成函数 通过定义测试区间生成a个数据点进行测试
    :param start: 正数 必须小于end
    :param end: 正数 必须大雨start
    :param a: 生成样本点数量 必须为正整数
    :return: 返回2*N的数据点矩阵
    """
    array_x = np.linspace(start, end, a)
    array_y0 = np.sin(2 * np.pi * array_x) + np.random.randn(a)
    result = np.vstack((np.mat(array_x), np.mat(array_y0)))
    return result


def analytic_solutions_regular_matching(a, result):

    """
    解析解的无正则计算函数 通过现有的公式进行计算 得出未经过调整的参数结果
    :param a: 参数为最终计算的列矩阵维度 必须为大于1的正整数
    :param result: 样本点矩阵 必须是2*N维度
    :return: 返回参数列矩阵
    """
    size = SAMPLE_NUMBER
    x = np.transpose(result[0])
    y = np.transpose(result[1])
    X = np.ones((size, 1))
    for i in range(1, a):
        X = np.hstack((X, np.power(x, i)))
```

```
    Xt = np.transpose(X)
    i = np.eye(a)
    W = (Xt * X + LAMBDA * i).I * Xt * y
    return W


def analytic_solutions_irregular_matching(a, result):

    """
    解析解的无正则计算函数 通过现有的公式进行计算 得出未经过调整的参数结果
    :param a: 参数为最终计算的列矩阵维度 必须为大于1的正整数
    :param result: 样本点矩阵 必须是2*N维度
    :return: 返回参数列矩阵
    """
    size = SAMPLE_NUMBER
    x = np.transpose(result[0])
    y = np.transpose(result[1])
    X = np.ones((size, 1))
    for i in range(1, a):
        X = np.hstack((X, np.power(x, i)))
    Xt = np.transpose(X)
    W = (Xt * X).I * Xt * y
    return W


def stochastic_gradient_descent_matching(a, result):
    """
    SDG匹配
    :param a:参数维度
    :param result:输入样本矩阵 必须为2*N
    :return:返回参数列矩阵
    """
    np.random.seed(a)
    size = SAMPLE_NUMBER
    x = np.transpose(result[0])
    y = np.transpose(result[1])
    X = np.ones((size, 1))
    for i in range(1, a):
        X = np.hstack((X, np.power(x, i)))
    re = 10
    theta = np.ones((a, 1))
    count = 0
    while re > EPSILON and count < LOOP_MAX:
        re = 0
        d = 2 * X.T * X * theta - 2 * X.T * y + LAMBDA * (1 / np.sqrt
(theta.T * theta)) * theta
        theta = theta - ALPHA * d
        re = np.linalg.norm(y - X * theta)
        count += 1
    print("theta:", theta)
    print("final loss:", re)
```

```
    return theta


def conjugate_gradient(a, result):
    size = SAMPLE_NUMBER
    x = np.transpose(result[0])
    y = np.transpose(result[1])
    X = np.ones((size, 1))
    for i in range(1, a):
        X = np.hstack((X, np.power(x, i)))

    x0 = np.ones((a, 1))
    k = np.eye(a)
    A = X.T * X + LAMBDA * k
    b = X.T * y
    count = 0
    r0 = b - np.dot(A, x0)
    p0 = r0
    # test1 = np.dot(r0.T, r0)
    # test2 = p0.T * X * p0
    while count < 1000:
        alpha0 = np.dot(r0.T, r0) / (p0.T * A * p0)
        x1 = x0 + alpha0.tolist()[0][0] * p0
        r1 = r0 - alpha0.tolist()[0][0] * A * p0
        if np.linalg.norm(r1) < 1e-6:
            print("count:", count)
            print(x1)
            return x1
        beta0 = np.dot(r1.T, r1) / np.dot(r0.T, r0)
        p1 = r1 + beta0.tolist()[0][0] * p0
        x0 = x1
        p0 = p1
        r0 = r1
        count += 1
    print("count:", count)
    return x0


def pre_draw(ax, result):
    plt.xlim(0, 2)
    plt.ylim(-2, 2)
    x0 = np.linspace(CALCULATE_START, TEST_END, 2000)
    y1 = np.sin(2 * np.pi * x0)

    plt.xlabel("X")
    plt.ylabel("Y")

    test = random_point_test(TEST_START, TEST_END, TEST_POINTS_NUMBER
S)

    ax.scatter(test[0].tolist(), test[1].tolist(), label="test points
```

```
", c="y", marker=".")
    ax.scatter(result[0].tolist(), result[1].tolist(), label="calcula
ted points", c="r", marker=".")
    ax.plot(x0, y1, label="sin", color="blue")
    plt.legend()


def draw(s, w, ax, label, color):
    """
    画图函数 已经封装了所有的可能结果 通过传入的参数进行绘制并且会在图形中展示
    测试点和计算点
    :param s: 图标题
    :param ax: 画图的画布区域
    :param w: 计算得出的参数矩阵
    :param label: 图例
    :param color: 颜色算子
    :return: 无返回值 直接调用绘图
    """
    wt = np.transpose(w)
    l = list(reversed(wt.tolist()[0]))
    x0 = np.linspace(CALCULATE_START, TEST_END, 2000)
    y0 = np.poly1d(l)
    print(y0)

    ax.set_title(s)

    ax.plot(x0, y0(x0), label=label, color=[color * 0.04, color * 0.0
8, color * 0.1])
    plt.legend()


def main():

    global PARAM_NUMBER, LAMBDA
    fig = plt.figure(figsize=(19.2, 10.8))
    result = random_point(CALCULATE_START, CALCULATE_END, SAMPLE_NUMB
ER)

    print("analytic_solutions_irregular_matching")
    ax1 = fig.add_subplot(221)
    pre_draw(ax1, result)
    for i in range(2, 10):
        PARAM_NUMBER = i
        w = analytic_solutions_irregular_matching(PARAM_NUMBER, resul
t)
        draw("analytic_solutions_irregular_matching", w, ax1, "PARAM_
NUMBER:" + str(PARAM_NUMBER), i)

    PARAM_NUMBER = 9

    print("analytic_solutions_regular_matching with PARAM_NUMBER:" +
```

```
str(PARAM_NUMBER))
    ax2 = fig.add_subplot(222)
    pre_draw(ax2, result)
    for i in range(2, 10):
        LAMBDA = i * 0.0004
        w = analytic_solutions_regular_matching(PARAM_NUMBER, result)
        draw("analytic_solutions_regular_matching with PARAM_NUMBER:"
 + str(PARAM_NUMBER), w, ax2, "lambda:" + str(LAMBDA), i)

    print("stochastic_gradient_descent_matching")
    ax3 = fig.add_subplot(223)
    pre_draw(ax3, result)
    for i in range(2, 10):
        PARAM_NUMBER = i
        w = stochastic_gradient_descent_matching(PARAM_NUMBER, result
)
        draw("stochastic_gradient_descent_matching", w, ax3, "PARAM_N
UMBER:" + str(PARAM_NUMBER), i)

    PARAM_NUMBER = 9
    ax4 = fig.add_subplot(224)
    pre_draw(ax4, result)
    w = conjugate_gradient(PARAM_NUMBER, result)

    draw("conjugate_gradient", w, ax4, "a", 4)
    plt.show()


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