**模式识别报告L3**

自卓2201 杨欣怡 U202215067

1，分别编写一个用广义逆和梯度下降法来求最小误差平方和最佳解的算法

广义逆：

1. if self.mode == 'a':
2. x\_gen\_inverse = np.matmul(np.linalg.inv(np.matmul(x.T, x)), x.T)
3. W = np.matmul(x\_gen\_inverse, y)
4. self.w = W
5. y\_estimated = np.matmul(x, W)
6. return y\_estimated

梯度下降法：

1. if self.mode == 'g':
2. losses = []
3. for i in range(self.epoch):
4. y\_estimated = np.matmul(x, self.w)
5. loss = L2\_loss(y\_estimated, y)
6. losses.append(loss)
7. grad = self.\_calculate\_grad(x, y)
8. self.w -= self.lr \* grad
9. return y\_estimated, losses
10. def \_calculate\_grad(self, x, y):
11. num\_sample = x.shape[0]
12. aver = (2 / num\_sample)
13. grad = aver \* np.matmul((np.matmul(x, self.w) - y).T, x)
14. return grad.T
15. def L2\_loss(y\_estimated, y):
16. num\_sample = y.shape[0]
17. assert (y\_estimated.shape == y.shape)
18. loss = (1 / num\_sample) \* np.sum((y\_estimated - y) \*\* 2)
19. return loss

2，（a）产生两个都具有200个二维向量的数据集和。数据集的样本来自均值向量协方差矩阵的正态分布，属于“+1”类，数据集的样本来自均值向量、协方差矩阵的正态分布，属于“-1”类，其中****是一个2\*2的单位矩阵。产生的数据中80%用于训练，20%用于测试。

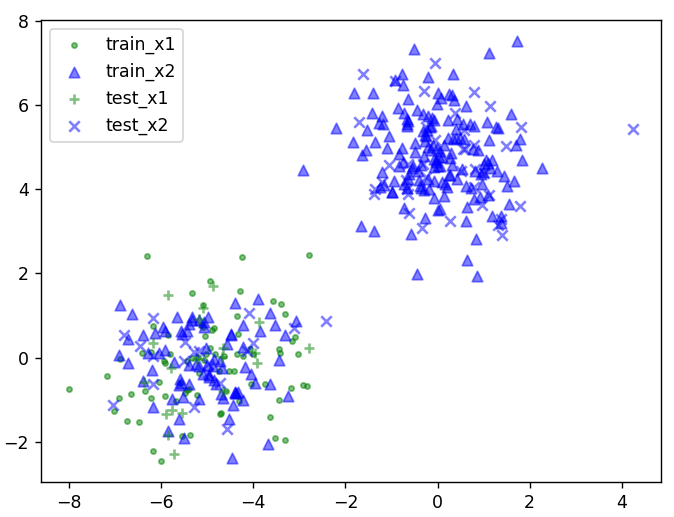
（b）在上述数据集上分别第1题的两个算法，利用产生的训练样本集得到分类面，算法中用到的各类超参数自定。

（c）分别在训练集和测试集上统计分类正确率。

（d）画出数据集和分类面。

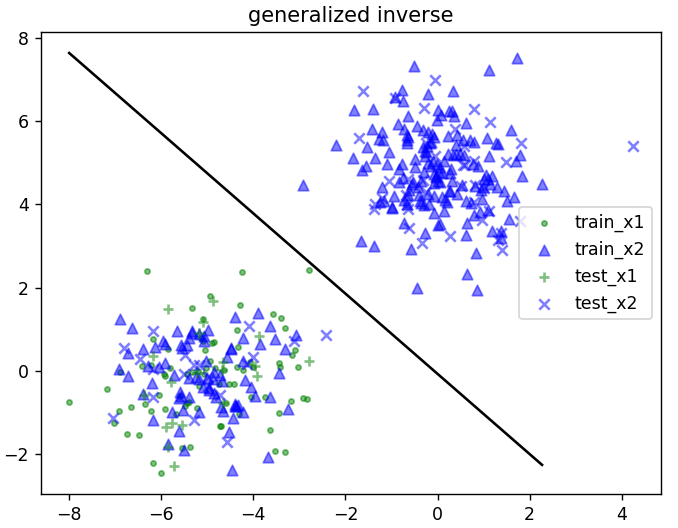
（e）画出损失函数随epoch增加的变化曲线。

1. data2 = Dataset()
2. x\_train = data2.x\_train
3. y\_train = data2.y\_train
4. x\_test = data2.x\_test
5. y\_test = data2.y\_test
6. c1 = plt.scatter(x\_train[:80, 0], x\_train[:80, 1], alpha=0.5, marker='.', c='red')
7. c2 = plt.scatter(x\_train[80:, 0], x\_train[80:, 1], alpha=0.5, marker='^', c='blue')
8. c3 = plt.scatter(x\_test[:20, 0], x\_test[:20, 1], alpha=0.5, marker='+', c='red')
9. c4 = plt.scatter(x\_test[20:, 0], x\_test[20:, 1], alpha=0.5, marker='x', c='blue')
10. plt.legend(handles=[c1, c2, c3, c4], labels=['train\_x1', 'train\_x2', 'test\_x1', 'test\_x2'], loc='best')
11. plt.show()



1. model1 = LinearRegression(mode='a')
2. y\_estimated = model1(x\_train, y\_train)
3. model1.eval(x\_test, y\_test)
4. w, b = model1.w[1:], model1.w[0]
5. c1 = plt.scatter(x\_train[:80, 0], x\_train[:80, 1], alpha=0.5, marker='.', c='green')
6. c2 = plt.scatter(x\_train[80:, 0], x\_train[80:, 1], alpha=0.5, marker='^', c='blue')
7. c3 = plt.scatter(x\_test[:20, 0], x\_test[:20, 1], alpha=0.5, marker='+', c='green')
8. c4 = plt.scatter(x\_test[20:, 0], x\_test[20:, 1], alpha=0.5, marker='x', c='blue')
9. xmax = np.max(x\_train[:, 0])
10. xmin = np.min(x\_train[:, 0])
11. point1 = [xmin, xmax]
12. point2 = [-(w[0] \* xmin + b) / w[1], -(w[0] \* xmax + b) / w[1]]
13. plt.plot(point1, point2, c='black')
14. plt.legend(handles=[c1, c2, c3, c4], labels=['train\_x1', 'train\_x2', 'test\_x1', 'test\_x2'], loc='best')
15. plt.title('generalized inverse')
16. plt.show()
17. lr = 0.02
18. epoch = 10
19. model2 = LinearRegression(lr=lr, epoch=epoch, mode='g')
20. y\_estimated, loss = model2(x\_train, y\_train)
21. model2.eval(x\_test, y\_test)
22. w, b = model2.w[1:], model2.w[0]
23. c1 = plt.scatter(x\_train[:80, 0], x\_train[:80, 1], alpha=0.5, marker='.', c='green')
24. c2 = plt.scatter(x\_train[80:, 0], x\_train[80:, 1], alpha=0.5, marker='^', c='blue')
25. c3 = plt.scatter(x\_test[:20, 0], x\_test[:20, 1], alpha=0.5, marker='+', c='green')
26. c4 = plt.scatter(x\_test[20:, 0], x\_test[20:, 1], alpha=0.4, marker='x', c='blue')
27. xmax = np.max(x\_train[:, 0])
28. xmin = np.min(x\_train[:, 0])
29. point1 = [xmin, xmax]
30. point2 = [-(w[0] \* xmin + b) / w[1], -(w[0] \* xmax + b) / w[1]]
31. plt.plot(point1, point2, c='black')
32. plt.legend(handles=[c1, c2, c3, c4], labels=['train\_x1',  'train\_x2', 'test\_x1','test\_x2'], loc='best')
33. plt.title('gradient descent')
34. plt.show()
35. epoch = np.arange(0, epoch)
36. loss = np.array(loss)
37. plt.plot(epoch, loss)
38. plt.xlabel('epoch')
39. plt.ylabel('loss')
40. plt.show()

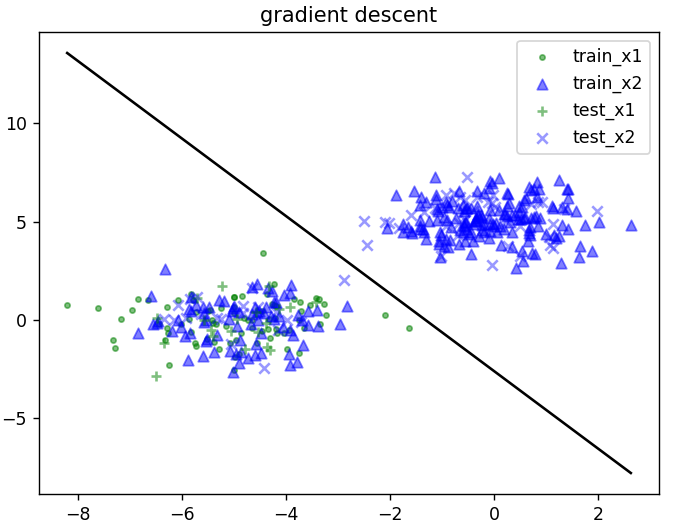
广义逆算法分类面：



分类正确率：



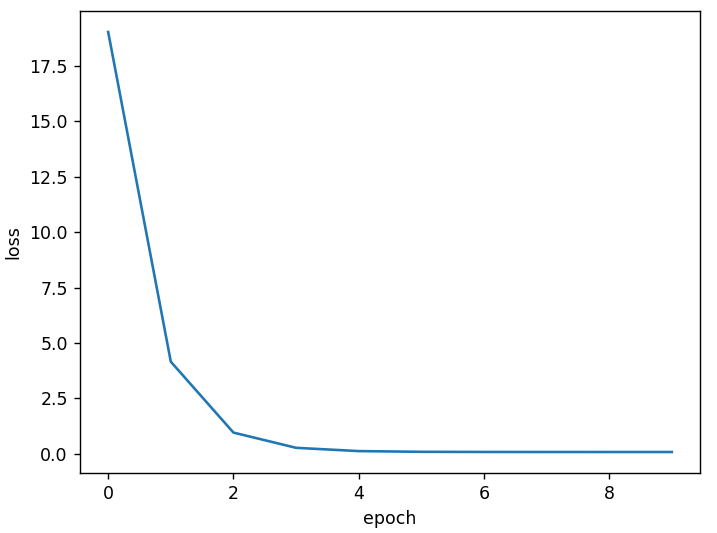
梯度下降法分类面：



分类正确率：



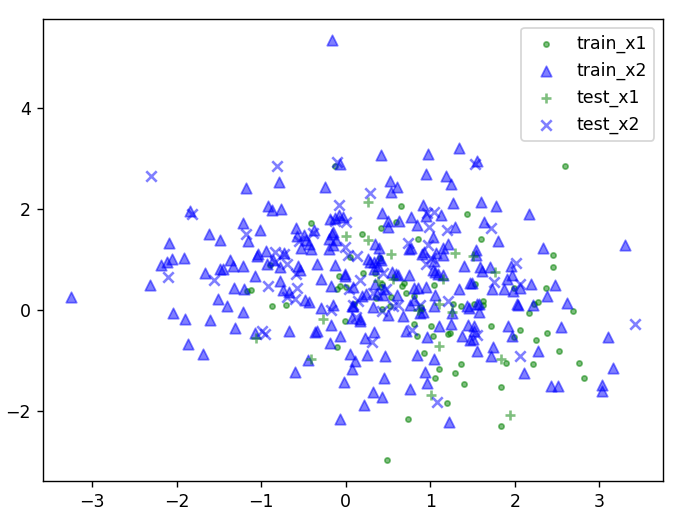
损失函数随epoch增加的变化曲线：



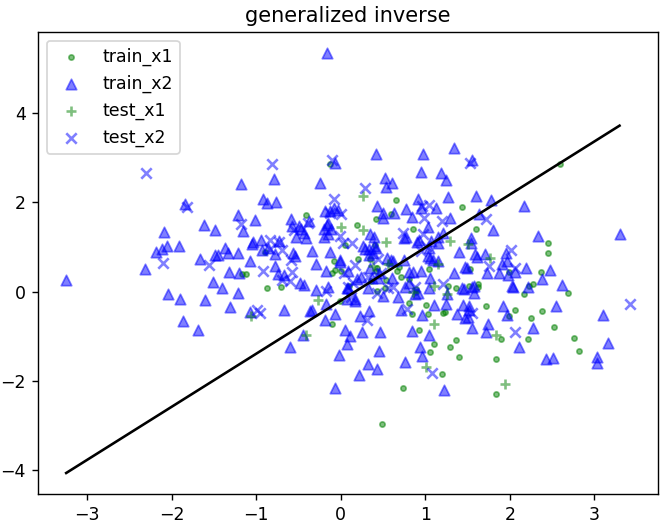
3，重复第2题的内容，但数据集和数据集的均值向量分别改为和，其他不变。

1. data3 = Dataset(means1=[1, 0], means2=[0, 1])
2. x\_train = data3.x\_train
3. y\_train = data3.y\_train
4. x\_test = data3.x\_test
5. y\_test = data3.y\_test
6. c1 = plt.scatter(x\_train[:80, 0], x\_train[:80, 1], alpha=0.5, marker='.', c='red')
7. c2 = plt.scatter(x\_train[80:, 0], x\_train[80:, 1], alpha=0.5, marker='^', c='blue')
8. c3 = plt.scatter(x\_test[:20, 0], x\_test[:20, 1], alpha=0.5, marker='+', c='red')
9. c4 = plt.scatter(x\_test[20:, 0], x\_test[20:, 1], alpha=0.5, marker='x', c='blue')
10. plt.legend(handles=[c1, c2, c3, c4], labels=['train\_x1', 'train\_x2', 'test\_x1', 'test\_x2'], loc='best')
11. plt.show()
12. model1 = LinearRegression(mode='a')
13. y\_estimated = model1(x\_train, y\_train)
14. model1.eval(x\_test, y\_test)
15. w, b = model1.w[1:], model1.w[0]
16. c1 = plt.scatter(x\_train[:80, 0], x\_train[:80, 1], alpha=0.5, marker='.', c='red')
17. c2 = plt.scatter(x\_train[80:, 0], x\_train[80:, 1], alpha=0.5, marker='^', c='blue')
18. c3 = plt.scatter(x\_test[:20, 0], x\_test[:20, 1], alpha=0.5, marker='+', c='red')
19. c4 = plt.scatter(x\_test[20:, 0], x\_test[20:, 1], alpha=0.5, marker='x', c='blue')
20. xmax = np.max(x\_train[:, 0])
21. xmin = np.min(x\_train[:, 0])
22. point1 = [xmin, xmax]
23. point2 = [-(w[0] \* xmin + b) / w[1], -(w[0] \* xmax + b) / w[1]]
24. plt.plot(point1, point2, c='black')
25. plt.legend(handles=[c1, c2, c3, c4], labels=['train\_x1',  'train\_x2', 'test\_x1','test\_x2'], loc='best')
26. plt.title('generalized inverse')
27. plt.show()
28. lr = 0.02
29. epoch = 100
30. model2 = LinearRegression(lr=lr, epoch=epoch, mode='g')
31. y\_estimated, loss = model2(x\_train, y\_train)
32. model2.eval(x\_test, y\_test)
33. w, b = model2.w[1:], model2.w[0]
34. c1 = plt.scatter(x\_train[:80, 0], x\_train[:80, 1], alpha=0.5, marker='.', c='red')
35. c2 = plt.scatter(x\_train[80:, 0], x\_train[80:, 1], alpha=0.5, marker='^', c='blue')
36. c3 = plt.scatter(x\_test[:20, 0], x\_test[:20, 1], alpha=0.5, marker='+', c='red')
37. c4 = plt.scatter(x\_test[20:, 0], x\_test[20:, 1], alpha=0.5, marker='x', c='blue')
38. xmax = np.max(x\_train[:, 0])
39. xmin = np.min(x\_train[:, 0])
40. point1 = [xmin, xmax]
41. point2 = [-(w[0] \* xmin + b) / w[1], -(w[0] \* xmax + b) / w[1]]
42. plt.plot(point1, point2, c='black')
43. plt.legend(handles=[c1, c2, c3, c4], labels=['train\_x1',  'train\_x2', 'test\_x1','test\_x2'], loc='best')
44. plt.title('gradient descent')
45. plt.show()
46. epoch = np.arange(0, epoch)
47. loss = np.array(loss)
48. plt.plot(epoch, loss)
49. plt.xlabel('epoch')
50. plt.ylabel('loss')
51. plt.show()

生成数据集：



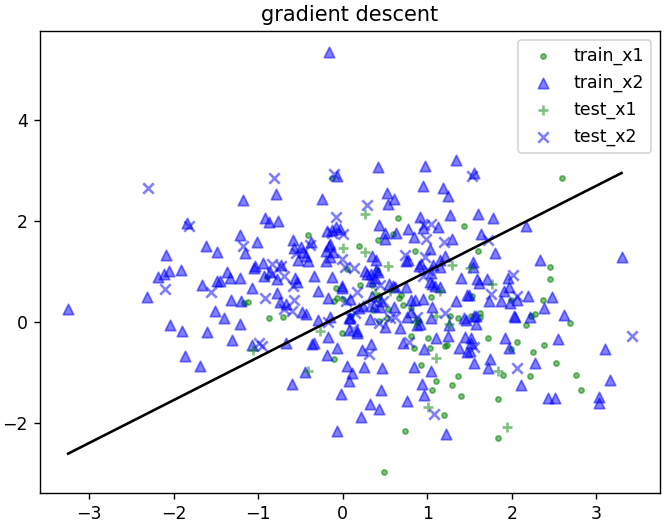
广义逆算法生成分类面：



分类正确率：



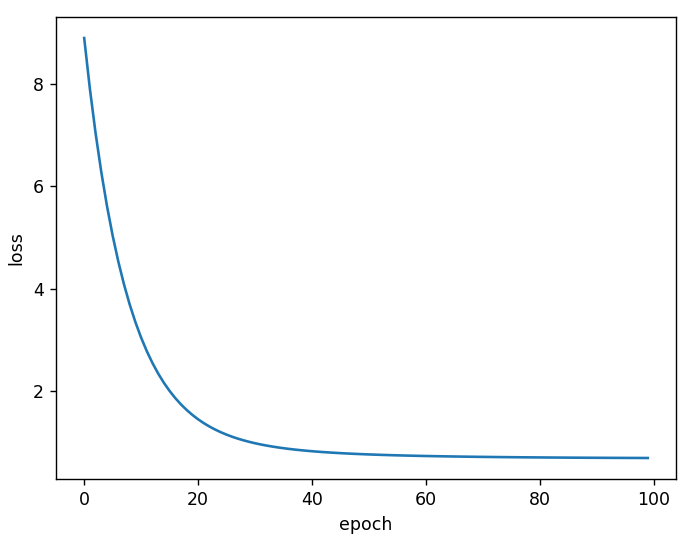
梯度下降法分类面：



分类正确率：



损失函数随epoch增加的变化曲线：

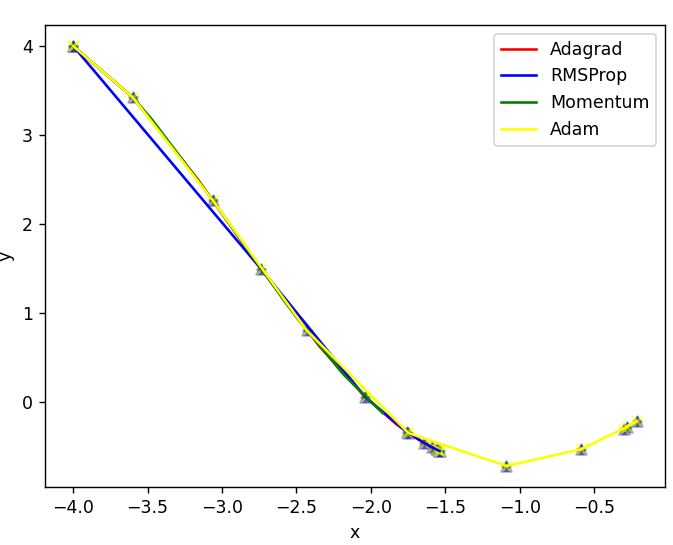


5，单变量函数为，分别用梯度下降法、随机梯度下降法、Adagrad、RMSProp、动量法（Momentum）和Adam共6种方法，编写程序画图呈现从初始值为-4、迭代10次时及的每次变化情况，这里对所有算法学习率（或初始学习率）均为0.4，为防止分母为0时给的最小量为1e-6，RMSProp算法的=0.9，动量法的=0.9，Adam的beta1=0.9，beta2=0.999，观察不同算法的变化情况体会各自的差异。如果迭代50次，并将Adam的beta1改成0.99，其他参数不变，观察不同算法的变化结果。尝试调整上述算法的各种参数，体会上述不同方法的特点。

1. class Adagrad:
2. def \_\_init\_\_(self, epsilon=1e-6):
3. self.sigma = 0
4. self.t = 0
5. self.epsilon = epsilon
6. def \_\_call\_\_(self, grad, lr):
7. self.sigma = np.math.sqrt(((self.sigma \*\* 2) \* self.t + grad \*\* 2) / (self.t + 1)) + self.epsilon
8. self.t = self.t + 1
9. return (lr \* grad) / self.sigma
10. class RMSProp(Adagrad):
11. def \_\_init\_\_(self, alpha=0.9):
12. super(RMSProp, self).\_\_init\_\_()
13. self.alpha = alpha
14. def \_\_call\_\_(self, grad, lr):
15. self.sigma = np.math.sqrt(self.alpha \* self.sigma + (1 - self.alpha) \* grad \*\* 2)
16. return (lr \* grad) / self.sigma
17. class Momentum:
18. def \_\_init\_\_(self, Lambda=0.9):
19. self.m = 0
20. self.Lambda = Lambda
21. def \_\_call\_\_(self, grad, lr):
22. self.m = - self.Lambda \* self.m + lr \* grad
23. return self.m
24. class Adam():
25. def \_\_init\_\_(self, beta1=0.9, beta2=0.99, epsilon=1e-6):
26. self.beta1 = beta1
27. self.beta2 = beta2
28. self.epsilon = epsilon
29. self.m = 0
30. self.v = 0
31. self.t = 1
32. def \_\_call\_\_(self, grad, lr):
33. self.m = self.beta1 \* self.m + (1 - self.beta1) \* grad
34. self.v = self.beta2 \* self.v + (1 - self.beta2) \* grad \*\* 2
35. if self.t < 5:
36. m = self.m / (1 - self.beta1 \*\* self.t)
37. v = self.v / (1 - self.beta2 \*\* self.t)
38. else:
39. m = self.m
40. v = self.v
41. return lr \* m / (np.math.sqrt(v) + self.epsilon)

迭代10次：

1. epoch = 10
2. lr = 0.4
3. f = func()
4. x = np.ones(4) \* -4
5. y = np.ones(4) \* f(-4)[0]
6. grad = np.zeros(4)
7. optimizer = [Adagrad(), RMSProp(), Momentum(), Adam()]
8. xs = []
9. ys = []
10. for i in range(epoch):
11. for j in range(4):
12. y[j], grad[j] = f(x[j])
13. xs.append([x[0], x[1], x[2], x[3]])
14. ys.append([y[0], y[1], y[2], y[3]])
15. for j in range(4):
16. x[j] -= optimizer[j](grad[j], lr)
17. xs = np.array(xs)
18. ys = np.array(ys)
19. plt.plot(xs[:, 0], ys[:, 0], c='red', label='Adagrad')
20. plt.plot(xs[:, 1], ys[:, 1], c='blue', label='RMSProp')
21. plt.plot(xs[:, 2], ys[:, 2], c='green', label='Momentum')
22. plt.plot(xs[:, 3], ys[:, 3], c='yellow', label='Adam')
23. plt.scatter(xs[:, 1::2], ys[:, 1::2], alpha=0.5, c='red', marker='.')
24. plt.scatter(xs[:, 1::2], ys[:, 1::2], alpha=0.5, c='blue', marker='^')
25. plt.scatter(xs[:, 1::2], ys[:, 1::2], alpha=0.5, c='green', marker='+')
26. plt.scatter(xs[:, 1::2], ys[:, 1::2], alpha=0.5, c='yellow', marker='x')
27. plt.xlabel('x')
28. plt.ylabel('y')
29. plt.legend()
30. plt.show()



迭代50次：

1. epoch = 50
2. lr = 0.4
3. f = func()
4. x = np.ones(4) \* -4
5. y = np.ones(4) \* f(-4)[0]
6. grad = np.zeros(4)
7. optimizer = [Adagrad(), RMSProp(), Momentum(), Adam(beta1=0.9)]
8. xs = []
9. ys = []
10. for i in range(epoch):
11. for j in range(4):
12. y[j], grad[j] = f(x[j])
13. xs.append([x[0], x[1], x[2], x[3]])
14. ys.append([y[0], y[1], y[2], y[3]])
15. for j in range(4):
16. x[j] -= optimizer[j](grad[j], lr)
17. xs = np.array(xs)
18. ys = np.array(ys)
19. plt.plot(xs[:, 0], ys[:, 0], c='red', label='Adagrad')
20. plt.plot(xs[:, 1], ys[:, 1], c='blue', label='RMSProp')
21. plt.plot(xs[:, 2], ys[:, 2], c='green', label='Momentum')
22. plt.plot(xs[:, 3], ys[:, 3], c='yellow', label='Adam')
23. plt.scatter(xs[:, 1::5], ys[:, 1::5], alpha=0.5, c='red', marker='.')
24. plt.scatter(xs[:, 1::5], ys[:, 1::5], alpha=0.5, c='blue', marker='^')
25. plt.scatter(xs[:, 1::5], ys[:, 1::5], alpha=0.5, c='green', marker='+')
26. plt.scatter(xs[:, 1::5], ys[:, 1::5], alpha=0.5, c='yellow', marker='x')
27. plt.xlabel('x')
28. plt.ylabel('y')
29. plt.legend()
30. plt.show()

