# 第二次上机实验报告

姓名：刘龙 学号：3119009062 班级：软件工程2班

## 实验一Python绘图模拟非线性方程求解过程

【思路分析】

通过绘图模拟出迭代过程中近似值向真实值靠近的过程，当近似值和真实值之间肉眼难以区分的时候，通过放大x坐标轴，显示出两者之间的区别。

1. 定义最小间隔距离slot，表示当近似值和真实值横坐标差的绝对值小于slot时，将会放大x轴，显示出靠近过程。在放大x坐标轴以后，slot变为原来的0.001
2. 如何放大x轴：将当前点的横坐标，记为point\_x，将其减去特定的值Δx，得到x轴左边界；加上Δx，得到x轴右边界
3. slot更新：避免每次都需要放大x轴，只有在当近似值和真实值很靠近时，才会放大x轴

【实现代码】

# -\*- coding: utf-8 -\*-

"""

Created on Thu Apr 21 20:31:34 2022

@author: Wu Xinghua

"""

import numpy as np

from matplotlib import pyplot as plt, animation

def fixpt(f, x, epsilon=1.0E-5, N=500, store=False):

y = f(x)

n = 0

if store:

Values = [(x, y)]

while abs(y - x) >= epsilon and n < N:

x = f(x)

n += 1

y = f(x)

if store:

Values.append((x, y))

if store:

return y, Values

else:

if n >= N:

return "No fixed point for given start value"

else:

return x, n, y

# define f

def f(x):

return ((-58 \* x - 3) / (7 \* x \*\* 3 - 13 \* x \*\* 2 - 21 \* x - 12)) \*\* (1 / 2)

# find fixed point

res, points = fixpt(f, 1.5, store=True)

plt.rcParams['font.sans-serif'] = ['SimHei'] # 用来正常显示中文标签

plt.rcParams['axes.unicode\_minus'] = False # 用来正常显示负号

# 迭代数据存放在points，且近似值为y

xmin = 1.2

xmax = 1.6

figure = plt.figure()

i = list(range(0, len(points)))

# 迭代准确值,取最后一个元素

xvalue = points[len(points) - 1][1]

#最小间隔距离#

slot = 0.001

def update(i):

# 初始情况下

global xmin, xmax, xvalue, slot

if i >= len(points):

return

plt.clf()

x = np.arange(xmin, xmax, 1e-5)

# 画出f(x)图像和y=x图像

fx = f(x)

xx = x

if np.fabs(points[i][0] - xvalue) < slot:

xmin = points[i][0] - np.fabs(points[i][1] - points[i][0]) \* 10

xmax = points[i][0] + np.fabs(points[i][1] - points[i][0]) \* 10

slot /= 1000

plt.xlim(xmin, xmax)

plt.plot(x, xx, color='black', label='y=x', linewidth=0.5, linestyle='--')

plt.plot(x, fx, color='blue', label='y=f(x)', linewidth=0.5, linestyle='--')

plt.scatter(points[i][0], points[i][1], color='green', label='近似值')

plt.scatter(xvalue, f(xvalue), color='red', label='真实值')

plt.legend()

# 近似值

value = points[i][1]

plt.title("迭代次数:{},近似值:{}".format(i, value))

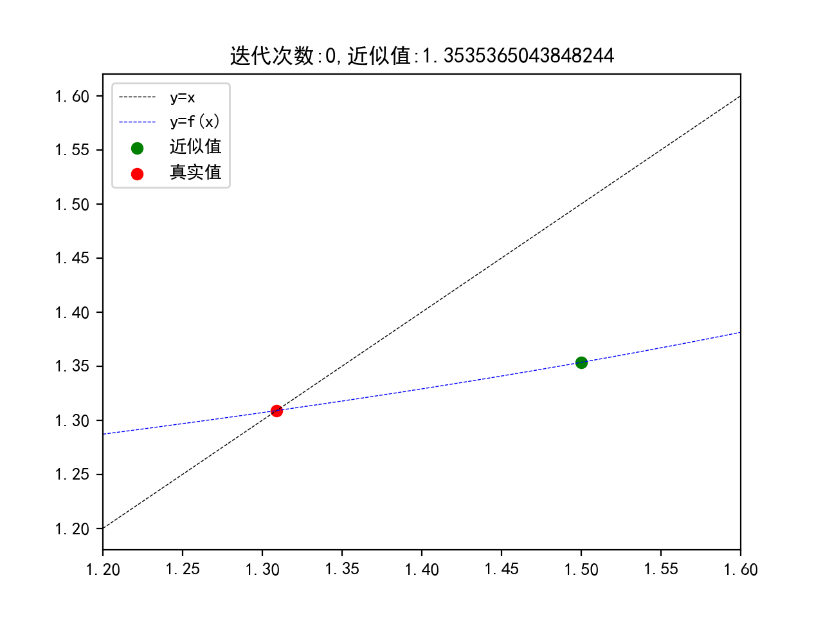
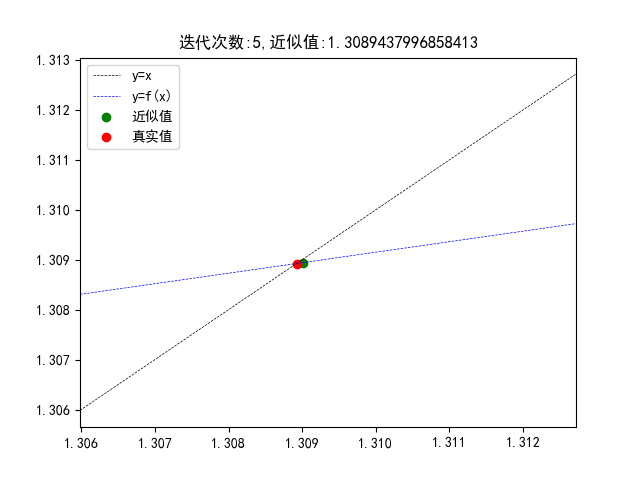
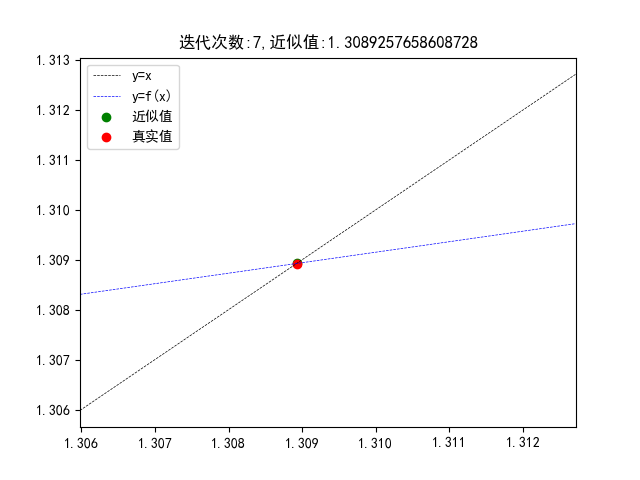
return

ani = animation.FuncAnimation(figure, update, interval=1000)

# show result

plt.show()

【结果展示】



## 实验二求解Hilbert系数矩阵方程组

【问题描述】

系数矩阵为Hilbert阵，对解全为1的方程组，随着 n = 2,3,......的增加，编写程序，测试和分析利用直接法和迭代法求解方程组的结果差别

【思路分析】

1. 构建解全为1，同时系数矩阵为希尔伯特矩阵的特殊线性方程组

则：





2. 使用迭代法求解线性方程组

对于线性方程组 ，可以在此基础上，写出从 的迭代形式如下：

 （1）

在（1）的基础上，对于第k+1次迭代有

 （2）

公式二就是迭代的理论基础，在每一次迭代的时候，将上一次求出的x解集带入公式（2）中，得到更加准确的解

1. 算法实现说明

定义LinearEquationGroup类，构造方法提供A和b，同时里面还定义的方法

|  |  |
| --- | --- |
| 方法 | 说明 |
| solve | 运用Numpy自带的线性方程组求解函数实现 |
| iteration | 按照公式2实现的自定义迭代方法 |
| check | 检验求解的是否准确 |

【代码实现】

import numpy as np

import xlwt

# 生成希尔伯特矩阵

def shell\_matrix(n):

"""

生成n\*n希尔伯特矩阵

:param n:

:return:

"""

result = np.zeros((n, n))

for i in range(0, n):

temp = i + 1

for j in range(0, n):

result[i, j] = 1 / temp

temp += 1

return result

# 构建解全为1的参数矩阵

def generate\_b(n):

list = [1 for i in range(n)]

result = np.array(list).T

return np.dot(shell\_matrix(n), result)

class LinearEquationGroup:

def \_\_init\_\_(self, A, b):

"""

构建矩阵

:param A: 参数矩阵

:param b: 系数矩阵

"""

self.A = A

self.b = b

def solve(self):

return np.linalg.solve(self.A, self.b)

def iteration(self, number, result):

# 根据系数矩阵获取每一个变量的迭代形式

m, n = self.A.shape

x = []

for i in range(m):

temp = [] # 储存当前迭代函数

for j in range(n):

if j == i:

continue

temp.append(self.A[i, j] / self.A[i, i])

x.append(temp)

x = np.array(x)

for time in range(number):

# 开始迭代

for i in range(n):

# 遍历x

xx = []

for j in range(n):

if i != j:

xx.append(result[j])

xx = np.array(xx)

temp = xx \* x[i]

sum = 0

for j in range(len(temp)):

sum += temp[j]

result[i] = self.b[i] / self.A[i, i] - sum

result = np.array(result)

return result.T

def check(self, result):

return np.allclose(np.dot(self.A, result), self.b)

n = 6

A = shell\_matrix(n)

b = generate\_b(n)

print("检验solve函数")

linear = LinearEquationGroup(A, b)

result = linear.solve()

print(linear.check(result))

print("检验迭代函数")

# 将数据写入excel文件当中

excel = xlwt.Workbook(encoding='utf-8', style\_compression=0)

sheet = excel.add\_sheet('data', cell\_overwrite\_ok=True)

row = 0 # 控制行

for number in range(0, 200):

print("迭代次数:{}".format(number))

result2 = linear.iteration(number, [0 for i in range(n)])

column = 0 # 控制列

sheet.write(row, column, number)

column += 1

for elem in result2:

sheet.write(row, column, float(elem))

column += 1

row += 1

print(result2)

excel.save("实验2.xls")

【结果分析】

当希尔伯特矩阵n=4时

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 迭代次数 | X1 | X2 | X3 | X4 |
| 79 | 0.997339 | 0.977555 | 1.109427 | 0.908416 |
| 80 | 0.997643 | 0.976416 | 1.109729 | 0.909125 |
| 81 | 0.997934 | 0.975326 | 1.110013 | 0.909809 |
| 82 | 0.998213 | 0.974284 | 1.110281 | 0.910467 |
| 83 | 0.998481 | 0.973288 | 1.110533 | 0.9111 |
| 84 | 0.998737 | 0.972335 | 1.11077 | 0.91171 |
| 85 | 0.998982 | 0.971424 | 1.110992 | 0.912298 |
| 86 | 0.999216 | 0.970553 | 1.111201 | 0.912864 |
| 87 | 0.999441 | 0.96972 | 1.111395 | 0.913409 |
| 88 | 0.999656 | 0.968924 | 1.111577 | 0.913935 |
| 89 | 0.999862 | 0.968164 | 1.111747 | 0.914441 |
| 90 | 1.000059 | 0.967437 | 1.111905 | 0.91493 |
| 91 | 1.000248 | 0.966742 | 1.112052 | 0.915401 |
| 92 | 1.000428 | 0.966078 | 1.112188 | 0.915855 |
| 93 | 1.000601 | 0.965444 | 1.112313 | 0.916294 |
| 94 | 1.000767 | 0.964839 | 1.112429 | 0.916717 |
| 95 | 1.000925 | 0.964261 | 1.112535 | 0.917126 |
| 96 | 1.001077 | 0.963708 | 1.112632 | 0.91752 |
| 97 | 1.001222 | 0.963181 | 1.11272 | 0.917901 |
| 98 | 1.001361 | 0.962678 | 1.1128 | 0.918269 |
| 99 | 1.001494 | 0.962198 | 1.112872 | 0.918625 |
| 100 | 1.001621 | 0.96174 | 1.112937 | 0.918968 |
| 101 | 1.001742 | 0.961303 | 1.112994 | 0.919301 |
| 102 | 1.001859 | 0.960886 | 1.113044 | 0.919622 |
| 103 | 1.00197 | 0.960489 | 1.113087 | 0.919933 |
| 104 | 1.002077 | 0.96011 | 1.113124 | 0.920234 |
| 105 | 1.002179 | 0.959749 | 1.113155 | 0.920525 |
| 106 | 1.002276 | 0.959404 | 1.11318 | 0.920807 |
| 107 | 1.002369 | 0.959077 | 1.113199 | 0.921081 |
| 108 | 1.002458 | 0.958765 | 1.113213 | 0.921346 |
| 109 | 1.002544 | 0.958467 | 1.113222 | 0.921602 |
| 110 | 1.002625 | 0.958185 | 1.113225 | 0.921851 |
| 111 | 1.002703 | 0.957916 | 1.113224 | 0.922093 |
| 112 | 1.002778 | 0.95766 | 1.113219 | 0.922327 |
| 113 | 1.002849 | 0.957417 | 1.113209 | 0.922554 |
| 114 | 1.002917 | 0.957185 | 1.113195 | 0.922775 |
| 115 | 1.002982 | 0.956966 | 1.113177 | 0.92299 |
| 116 | 1.003044 | 0.956758 | 1.113155 | 0.923199 |
| 117 | 1.003103 | 0.95656 | 1.113129 | 0.923401 |
| 118 | 1.00316 | 0.956372 | 1.1131 | 0.923599 |
| 119 | 1.003214 | 0.956194 | 1.113068 | 0.92379 |

当n=6时

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 迭代次数 | X1 | X2 | X3 | X4 | X5 | X6 |
| 155 | 1.015207 | 0.89389 | 1.098902 | 1.096357 | 1.001511 | 0.883444 |
| 156 | 1.015122 | 0.894523 | 1.098113 | 1.096071 | 1.001671 | 0.883862 |
| 157 | 1.015039 | 0.895152 | 1.097328 | 1.095787 | 1.001829 | 0.884278 |
| 158 | 1.014955 | 0.895779 | 1.096548 | 1.095503 | 1.001987 | 0.884692 |
| 159 | 1.014873 | 0.896402 | 1.095771 | 1.095222 | 1.002144 | 0.885104 |
| 160 | 1.01479 | 0.897022 | 1.094999 | 1.094941 | 1.0023 | 0.885514 |
| 161 | 1.014708 | 0.897639 | 1.09423 | 1.094662 | 1.002455 | 0.885921 |
| 162 | 1.014627 | 0.898253 | 1.093466 | 1.094385 | 1.00261 | 0.886327 |
| 163 | 1.014546 | 0.898863 | 1.092706 | 1.094109 | 1.002763 | 0.88673 |
| 164 | 1.014465 | 0.899471 | 1.091949 | 1.093834 | 1.002916 | 0.887132 |
| 165 | 1.014385 | 0.900075 | 1.091197 | 1.093561 | 1.003068 | 0.887531 |
| 166 | 1.014305 | 0.900676 | 1.090448 | 1.093289 | 1.003219 | 0.887928 |
| 167 | 1.014225 | 0.901274 | 1.089704 | 1.093018 | 1.003369 | 0.888323 |
| 168 | 1.014146 | 0.901869 | 1.088963 | 1.092749 | 1.003519 | 0.888717 |
| 169 | 1.014067 | 0.90246 | 1.088227 | 1.092481 | 1.003667 | 0.889108 |
| 170 | 1.013989 | 0.903049 | 1.087494 | 1.092215 | 1.003815 | 0.889497 |
| 171 | 1.013911 | 0.903635 | 1.086765 | 1.09195 | 1.003963 | 0.889884 |
| 172 | 1.013834 | 0.904217 | 1.086039 | 1.091686 | 1.004109 | 0.890269 |
| 173 | 1.013757 | 0.904797 | 1.085318 | 1.091423 | 1.004254 | 0.890653 |
| 174 | 1.01368 | 0.905373 | 1.0846 | 1.091162 | 1.004399 | 0.891034 |
| 175 | 1.013604 | 0.905947 | 1.083886 | 1.090902 | 1.004543 | 0.891413 |
| 176 | 1.013528 | 0.906517 | 1.083176 | 1.090644 | 1.004687 | 0.891791 |
| 177 | 1.013453 | 0.907085 | 1.08247 | 1.090386 | 1.004829 | 0.892166 |
| 178 | 1.013378 | 0.907649 | 1.081767 | 1.090131 | 1.004971 | 0.89254 |
| 179 | 1.013303 | 0.908211 | 1.081068 | 1.089876 | 1.005112 | 0.892911 |
| 180 | 1.013229 | 0.90877 | 1.080373 | 1.089622 | 1.005252 | 0.893281 |
| 181 | 1.013155 | 0.909326 | 1.079681 | 1.08937 | 1.005392 | 0.893649 |
| 182 | 1.013081 | 0.909879 | 1.078993 | 1.08912 | 1.00553 | 0.894015 |
| 183 | 1.013008 | 0.910429 | 1.078308 | 1.08887 | 1.005668 | 0.894379 |
| 184 | 1.012935 | 0.910976 | 1.077627 | 1.088622 | 1.005806 | 0.894741 |
| 185 | 1.012863 | 0.91152 | 1.07695 | 1.088375 | 1.005942 | 0.895101 |
| 186 | 1.012791 | 0.912062 | 1.076276 | 1.088129 | 1.006078 | 0.89546 |
| 187 | 1.012719 | 0.912601 | 1.075606 | 1.087884 | 1.006213 | 0.895817 |
| 188 | 1.012648 | 0.913137 | 1.074939 | 1.087641 | 1.006348 | 0.896172 |
| 189 | 1.012577 | 0.91367 | 1.074276 | 1.087399 | 1.006481 | 0.896525 |
| 190 | 1.012506 | 0.9142 | 1.073616 | 1.087158 | 1.006614 | 0.896876 |
| 191 | 1.012436 | 0.914728 | 1.07296 | 1.086918 | 1.006746 | 0.897225 |
| 192 | 1.012366 | 0.915253 | 1.072307 | 1.08668 | 1.006878 | 0.897573 |
| 193 | 1.012297 | 0.915775 | 1.071658 | 1.086442 | 1.007009 | 0.897919 |
| 194 | 1.012228 | 0.916294 | 1.071012 | 1.086206 | 1.007139 | 0.898263 |
| 195 | 1.012159 | 0.916811 | 1.070369 | 1.085971 | 1.007268 | 0.898605 |
| 196 | 1.012091 | 0.917325 | 1.06973 | 1.085737 | 1.007397 | 0.898946 |
| 197 | 1.012023 | 0.917836 | 1.069094 | 1.085505 | 1.007525 | 0.899285 |
| 198 | 1.011955 | 0.918345 | 1.068461 | 1.085273 | 1.007653 | 0.899622 |
| 199 | 1.011888 | 0.918851 | 1.067832 | 1.085043 | 1.007779 | 0.899958 |

分析：使用迭代法，随着n的增加，迭代所需的次数也在增加，相对于高斯消元法是相对比较慢的，所得到的解随着迭代次数的增加会逼近准确值。