计算方法Final-Project

PB19030888 张舒恒

1.

记 $F=\min_{\{u_1,\dots,u_{n-1}\}}\sum_{i=1}^{n}\frac{1}{2}(\frac{u_i-u_{i-1}}{h})^2h+\sum_{i=1}^{n-1}(\frac{\alpha}{4}u_i^4-f_iu_i)h$，$\alpha$=0时，

$$\frac{\partial F}{\partial \mu_i}=\left(\frac{u_i-u_{i-1}}{h}\right)-\left(\frac{u_{i+1}-u_i}{h}\right)-f_ih=0,$$

即$2u_i-u_{i-1}-u_{i+1}-f_ih^2=0$，$1\le i\le n-1$，$u_0=u_n=0$，线性方程组$A_hu_h=f_h$为

$$\begin{pmatrix} \frac{2}{h^2} & -\frac{1}{h^2} & & & & & \\ -\frac{1}{h^2} & \frac{2}{h^2} & -\frac{1}{h^2} & & & & \\ & -\frac{1}{h^2} & \frac{2}{h^2} & -\frac{1}{h^2} & & & \\ & & \cdots & & & & \\ & & & \cdots & & & \\ & & & & \cdots & & \\ & & & & -\frac{1}{h^2} & \frac{2}{h^2} & -\frac{1}{h^2} \\ & & & & & -\frac{1}{h^2} & \frac{2}{h^2} \end{pmatrix}\begin{pmatrix} u_1 \\ u_2 \\ \cdots \\ u_{n-1} \end{pmatrix}=\begin{pmatrix} f_1 \\ f_2 \\ \cdots \\ f_{n-1} \end{pmatrix}$$

2.

Jacobi迭代法：u_0和u_1两个数组循环迭代，即当次迭代利用$u_0[j+1]$和$u_0[j-1]$计算$u_1[j]$，下次迭代利用$u_1[j+1]$和$u_1[j-1]$计算$u_0[j]$。

```
u1[1] = (u0[2] + f(h) * h * h) / 2.0;
for(auto j = 2; j < n - 1; j++){
    u1[j] = (u0[j+1] + u0[j-1] + f((j) * h) * h * h) / 2.0;
}
u1[n-1] = (u0[n-2] + f((n-1)*h) * h * h) / 2.0;

long double max = 0.0;
for(auto j = 1; j < n; j++){
    //cout << u1[j] << " ";
    auto distance = fabs(u1[j] - u0[j]);
    if(distance > max)
        max = distance;
}
//cout << endl;
if(max < eps){
    long double eh = 0.0;
    for(auto j = 1; j < n; j++)
        eh += (u1[j] - check(j * h)) * (u1[j] - check(j * h));
    eh = sqrt(eh);
    cout << "count = " << cnt << endl;
    cout << "eh = " << eh << endl;
    break;
```

```
}
```

Gauss-Seidel迭代法：u_0和u_1两个数组循环迭代，即当次迭代利用$u_0[j+1]$和$u_1[j-1]$计算$u_1[j]$，下次迭代利用$u_1[j+1]$和$u_0[j-1]$计算$u_0[j]$。

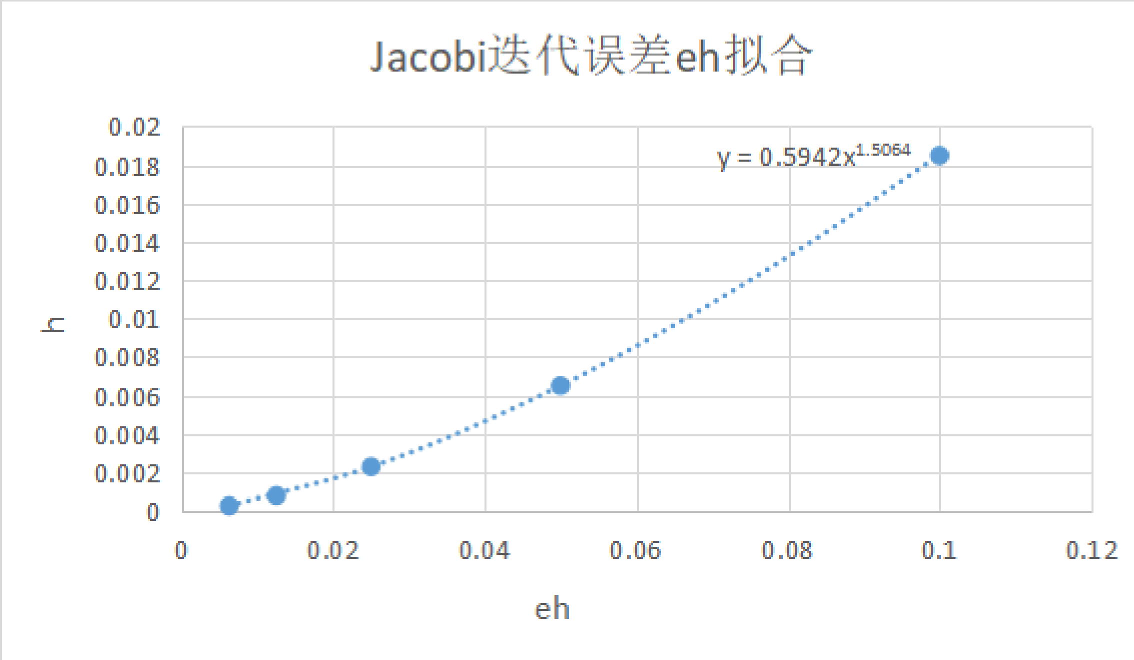
```
u1[1] = (u0[2] + f(h) * h * h) / 2.0;
for(auto j = 2; j < n - 1; j++){
    u1[j] = (u0[j+1] + u1[j-1] + f((j) * h) * h * h) / 2.0;
}
u1[n-1] = (u1[n-2] + f((n-1)*h) * h * h) / 2.0;

long double max = 0.0;
for(auto j = 1; j < n; j++){
    //cout << u1[j] << " ";
    auto distance = fabs(u1[j] - u0[j]);
    if(distance > max)
        max = distance;
}
//cout << endl;
if(max < eps){
    long double eh = 0.0;
    for(auto j = 1; j < n; j++)
        eh += (u1[j] - check(j * h)) * (u1[j] - check(j * h));
    eh = sqrt(eh);
    cout << "count = " << cnt << endl;
    cout << "eh = " << eh << endl;
    break;
}
```

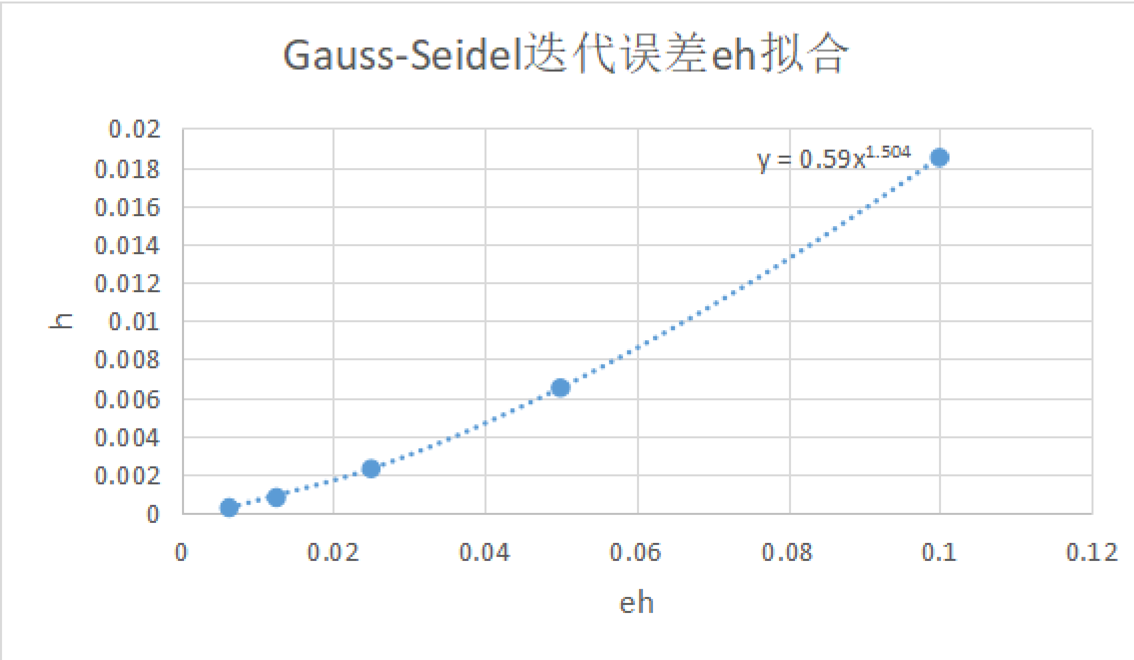
Jacobi迭代法和Gauss-Seidel迭代法u_h与精确解$u_e(x) = sin(\pi x)$之间的误差$e_h = \|u_h - u_e\|_2$：

n	Jacobi迭代误差e_h	Gauss-Seidel迭代误差e_h
10	0.018482	0.018482
20	0.00651018	0.00651019
40	0.00229943	0.0022995
80	0.000812015	0.000812425
160	0.000282725	0.000285045

3.

利用Excel对$e_h \sim h$进行最小二乘幂函数拟合，得到 Jacobi 迭代误差$e_h = \Theta(\beta^{1.5064})$，Gauss-Seidel迭代误差$e_h = \Theta(\beta^{1.504})$





4.

Jacobi迭代法和Gauss-Seidel迭代法收敛所需要的迭代次数：

n	Jacobi迭代次数	Gauss-Seidel迭代次数
10	400	208
20	1505	781
40	5587	2906
80	20563	10732
160	75071	39334

结论：从表中可以看出Jacobi迭代次数是Gauss-Seidel迭代次数1.9倍左右，此问题中Gauss-Seidel迭代法收敛速度明显快于Jacobi迭代法。

5.

α=1时，$\frac{\partial F}{\partial \mu_i}=\left(\frac{u_i-u_{i-1}}{h}\right)-\left(\frac{u_{i+1}-u_i}{h}\right)+(u_i^3-f_i)h=0$，非线性方程组为 $2u_i-u_{i-1}-u_{i+1}+(u_i^3-f_i)h^2=0$，$1\le i\le n-1$，$u_0=u_n=0$

6.

Newton迭代法：每个方程对每个$u_i, 1\le i\le n-1$求偏导记录在系数矩阵A中，求出 $delta\_u=A^{-1}(-f)$，当次迭代结果$u+=delta_u$

```
A = np.zeros((n - 1, n - 1), dtype=np.double)
for i in range(1, n - 2):
    A[i][i - 1] += -1
    A[i][i] += (2 + 3 * h * h * u[i])
    A[i][i + 1] += -1
A[0][0] += (2 + 3 * h * h * u[0])
A[0][1] += -1
A[n - 2][n - 2] += (2 + 3 * h * h * u[n - 2])
A[n - 2][n - 3] += -1
A_inverse = np.linalg.inv(A)

f = np.zeros(n - 1, dtype=np.double)
for i in range(1, n - 2):
    f[i] = 2 * u[i] - u[i - 1] - u[i + 1] + (u[i] ** 3 - func((i + 1) * h)) * h
* h
f[0] = 2 * u[0] - u[1] + (u[0] ** 3 - func(h)) * h * h
f[n - 2] = 2 * u[n - 2] - u[n - 3] + (u[n - 2] ** 3 - func((n - 1) * h)) * h * h

delta_u = np.dot(A_inverse, -f)
# print(delta_u, '\n')
u += delta_u
```

Newton迭代法u_h与精确解$u_e(x)=sin(\pi x)$之间的误差$e_h=\|u_h-u_e\|_2$:

n	Newton迭代误差e_h
10	0.015018033484455717
20	0.005300890512751275
40	0.001873370142282602
80	0.0006622691265689797
160	0.00023414373803619495

利用Excel对$e_h \sim h$进行最小二乘幂函数拟合，得到Newton迭代收敛阶为1.5007