中国科学技术大学

实验报告

计算方法

Final Project

学生姓名： 朱云沁

学生学号： PB20061372

完成时间： 二〇二二年五月三十日

目录

一、 实验题目

考虑数值求解如下的优化问题

$$\min_{u(x)\in C_0^1([0,1])} \quad \int_0^1 \left\{\frac{1}{2}\left[u'(x)\right]^2 + \frac{\alpha}{4}u^4(x) - f(x)u(x)\right\}\mathrm{d}x \tag{1}$$

令 $h = 1/n$, $x_i = ih, i = 0, \cdots, n$. 记 $f_i = f(x_i)$, u_i 为 $u(x_i)$ 的数值逼近, 则 $u_0 = u_n = 0$. 对 (1) 式采用如下的数值积分格式

$$\min_{u_1,\cdots,u_{n-1}} \quad \sum_{i=1}^{n}\frac{1}{2}\left(\frac{u_i - u_{i-1}}{h}\right)^2 h + \sum_{i=1}^{n-1}\left(\frac{\alpha}{4}u_i^4 - f_i u_i\right)h \tag{2}$$

分别记 $\boldsymbol{u}_h = (u_1, \cdots, u_{n-1})^{\mathrm{T}}$, $\boldsymbol{f}_h = (f_1, \cdots, f_{n-1})^{\mathrm{T}}$.

1. 当 $\alpha = 0$ 时，推导 $u_1, \cdots, u_{n-1}$ 满足的线性方程组 $\boldsymbol{A}_h\boldsymbol{u}_h = \boldsymbol{f}_h$.

2. 当 $f(x) = \pi^2\sin(\pi x)$, $n = 10, 20, 40, 80, 160$ 时, 分别利用 Jacobi 和 Gauss-Seidel 迭代法求解 $\boldsymbol{A}_h\boldsymbol{u}_h = \boldsymbol{f}_h$ (迭代法的终止准则 $\varepsilon = 10^{-10}$), 并比较 $\boldsymbol{u}_h$ 与精确解 $u_e(x) = \sin(\pi x)$ 之间的误差 $e_h = ||\boldsymbol{u}_h - \boldsymbol{u}_e||_2$, 记录在一张表中.

3. 假设 (2) 式中的 $\boldsymbol{u}_h$ 对 (1) 式中的 $u(x)$ 的逼近误差满足 $e_h = \Theta(h^\beta)$, 基于上表中的数据试利用最小二乘法找到 β.

4. 对 $n = 10, 20, 40, 80, 160$, 分别记录 Jacobi 和 Gauss-Seidel 迭代法收敛所需要的迭代次数在同一张表中, 从中能得到什么结论?

5. 当 $\alpha = 1$ 时，推导 $u_1, \cdots, u_{n-1}$ 满足的非线性方程组.

6. 当 $f(x) = \pi^2\sin(\pi x) + \sin^3(\pi x)$, $n = 10, 20, 40, 80, 160$ 时，利用 Newton 迭代法求解上一小题中的非线性方程组 (迭代法的终止准则 $\varepsilon = 10^{-8}$), 并比较 $\boldsymbol{u}_h$ 与精确解 $u_e(x) = \sin(\pi x)$ 之间的误差 $e_h = ||\boldsymbol{u}_h - \boldsymbol{u}_e||_2$, 记录在一张表中, 并利用最小二乘法找出该情形下算法的收敛阶.

二、 实验结果

1.

当 $\alpha = 0$ 时, $\boldsymbol{u}_h$ 满足的线性方程组为

$$\frac{1}{h^2}\begin{pmatrix} 2 & -1 & & \\ -1 & 2 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{pmatrix}\begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-1} \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_{n-1} \end{pmatrix} \tag{3}$$

2.

利用 Jacobi 迭代法解得 $\boldsymbol{u}_h$. 取 $x = 0.1, 0.2, \cdots, 0.9$ 处的数值解 $u_h(x)$, 如表 1 所示. 当 $n = 10, 160$ 时, 作得 $u_h(x)$ 的图象, 如图 1 所示.

表 1: Jacobi 迭代法求得数值解

x	$u_{0.1}(x)$	$u_{0.05}(x)$	$u_{0.025}(x)$	$u_{0.0125}(x)$	$u_{0.00625}(x)$	$u_e(x)$
0.1	0.31157115	0.30965317	0.30917588	0.30905667	0.30902676	0.30901699
0.2	0.59264354	0.58899533	0.58808747	0.58786072	0.58780383	0.58778525
0.3	0.81570386	0.81068252	0.80943297	0.80912086	0.80904257	0.80901699
0.4	0.95891739	0.95301446	0.95154552	0.95117862	0.95108658	0.95105652
0.5	1.00826542	1.00205870	1.00051417	1.00012839	1.00003161	1.00000000
0.6	0.95891739	0.95301446	0.95154552	0.95117862	0.95108658	0.95105652
0.7	0.81570386	0.81068252	0.80943297	0.80912086	0.80904257	0.80901699
0.8	0.59264354	0.58899533	0.58808747	0.58786072	0.58780383	0.58778525
0.9	0.31157115	0.30965317	0.30917588	0.30905667	0.30902676	0.30901699

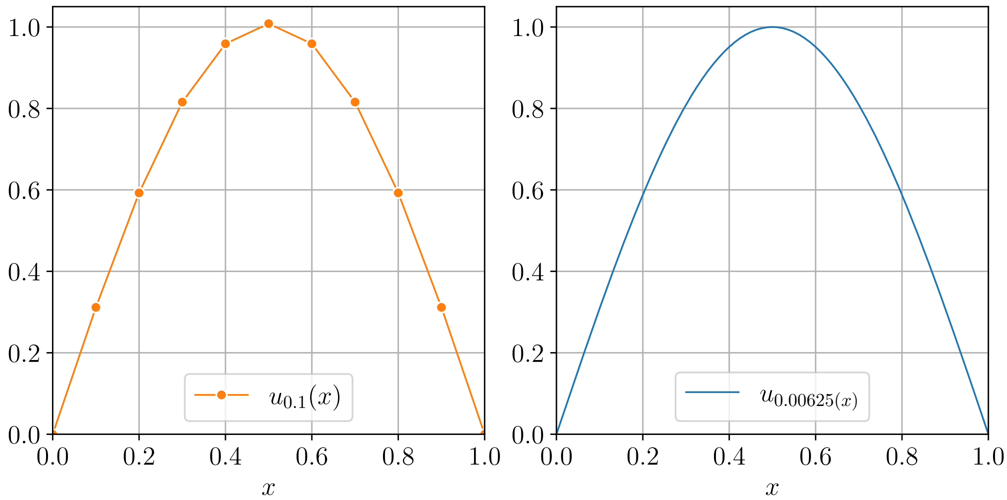

图 1: Jacobi 迭代法求得数值解

利用 Gauss-Seidel 迭代法解得 $\boldsymbol{u}_h$. 取 $x = 0.1, 0.2, \cdots, 0.9$ 处的数值解 $u_h(x)$, 如表 2 所示. 当 $n = 10, 160$ 时, 作得 $u_h(x)$ 的图象, 如图 2 所示.

表 2: Gauss-Seidel 迭代法求得数值解

x	$u_{0.1}(x)$	$u_{0.05}(x)$	$u_{0.025}(x)$	$u_{0.0125}(x)$	$u_{0.00625}(x)$	$u_e(x)$
0.1	0.31157115	0.30965317	0.30917589	0.30905669	0.30902684	0.30901699
0.2	0.59264354	0.58899533	0.58808748	0.58786076	0.58780398	0.58778525
0.3	0.81570386	0.81068252	0.80943298	0.80912092	0.80904278	0.80901699
0.4	0.95891739	0.95301446	0.95154553	0.95117868	0.95108682	0.95105652
0.5	1.00826542	1.00205870	1.00051418	1.00012846	1.00003187	1.00000000
0.6	0.95891739	0.95301446	0.95154553	0.95117869	0.95108683	0.95105652
0.7	0.81570386	0.81068252	0.80943298	0.80912092	0.80904278	0.80901699
0.8	0.59264354	0.58899533	0.58808748	0.58786076	0.58780399	0.58778525
0.9	0.31157115	0.30965317	0.30917589	0.30905669	0.30902684	0.30901699

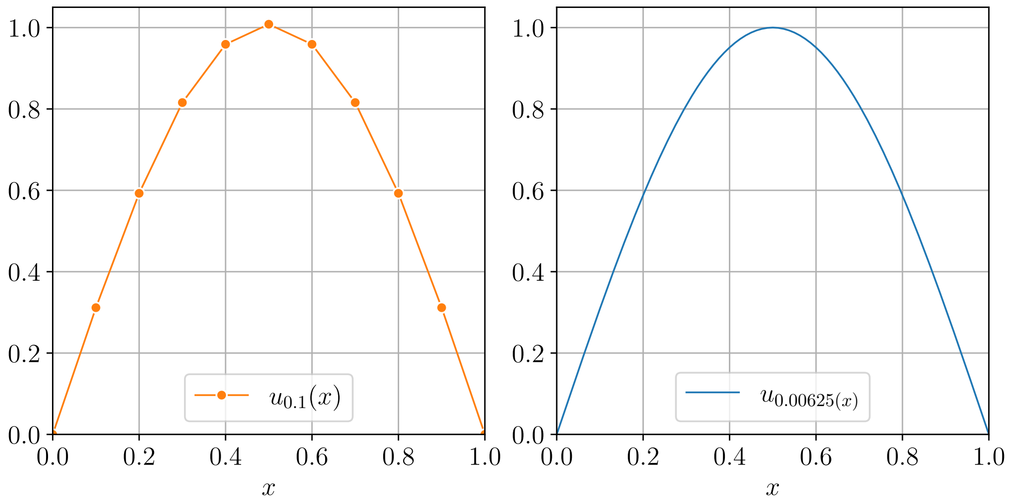

图 2: Gauss-Seidel 迭代法求得数值解

分别求得两种方法的误差 $e_h = ||\boldsymbol{u}_h - \boldsymbol{u}_e||_2$, 如表 3 所示.

表 3: $\alpha = 0$ 时的逼近误差

n	h	e_h	
		Jacobi	Gauss-Seidel
10	0.1	8.26541503×10^{-3}	8.26541608×10^{-3}
20	0.05	2.05869875×10^{-3}	2.05870278×10^{-3}
40	0.025	5.14168202×10^{-4}	5.14184336×10^{-4}
80	0.0125	1.28390848×10^{-4}	1.28455701×10^{-4}
160	0.00625	3.16095753×10^{-5}	3.18689490×10^{-5}

n	$\log h$	$\log e_h$	
		Jacobi	Gauss-Seidel
10	-2.30258509	-4.79567533	-4.79567521
20	-2.99573227	-6.18568117	-6.18567921
40	-3.68887945	-7.57296010	-7.57292873
80	-4.38202663	-8.96043144	-8.95992645
160	-5.07517382	-10.36205047	-10.35387841

可见, 随区间剖分点数 n 增大, 误差 e_h 逐渐减小而趋于 0.

3.

假设 $e_h = \Theta(h^\beta)$, 则 $\log e_h \approx \beta_0 + \beta \log h$. 利用最小二乘法拟合, 得到曲线如图 3 所示.

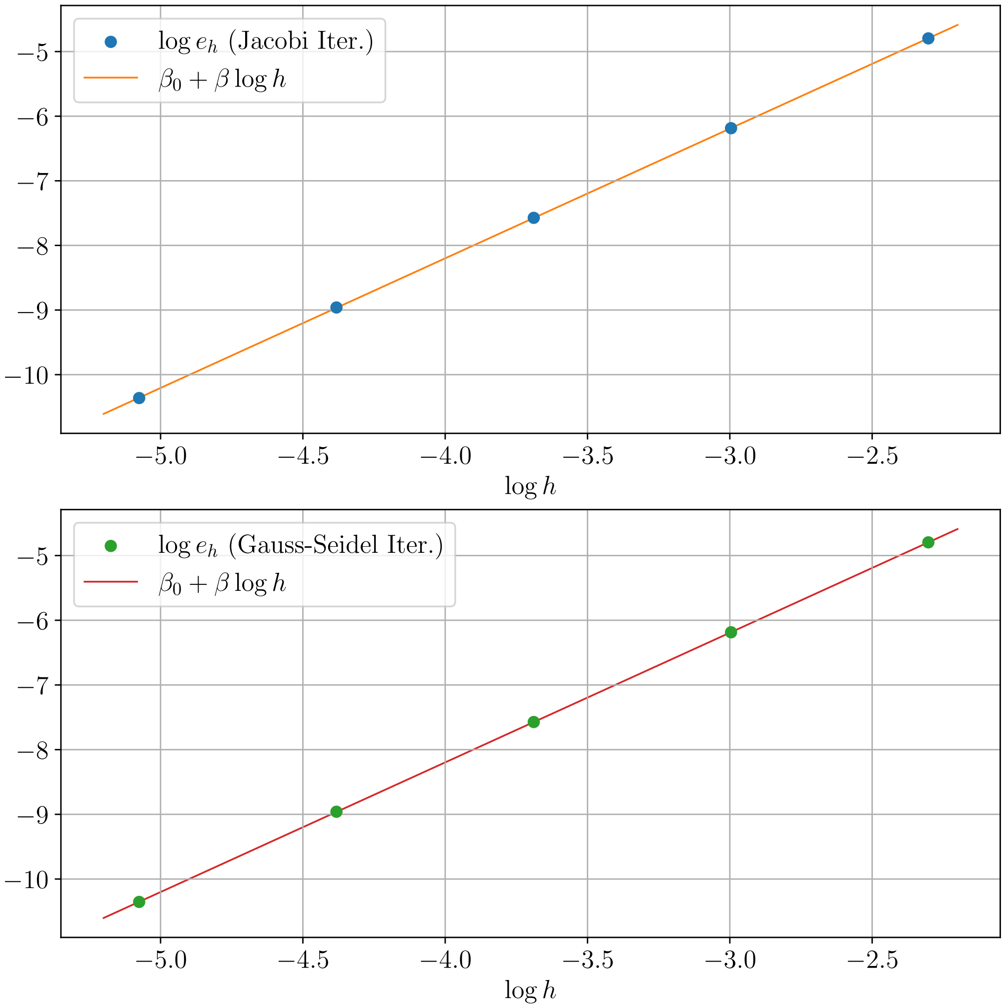

图 3: $\alpha = 1$ 时的误差拟合

使用 Jacobi 迭代法时, 数值算法的收敛阶为

$$\beta = 2.00642821 \approx 2 \tag{4}$$

使用 Gauss-Seidel 迭代法时, 数值算法的收敛阶为

$$\beta = 2.00399771 \approx 2 \tag{5}$$

4.

Jacobi 和 Gauss-Seidel 迭代法满足精度时的迭代次数如表 4 所示. 其图象如图 4 所示.

表 4: 求解线性方程组所需迭代次数

n	Jacobi	Gauss-Seidel
10	399	207
20	1504	780
40	5586	2905
80	20562	10731
160	75070	39333

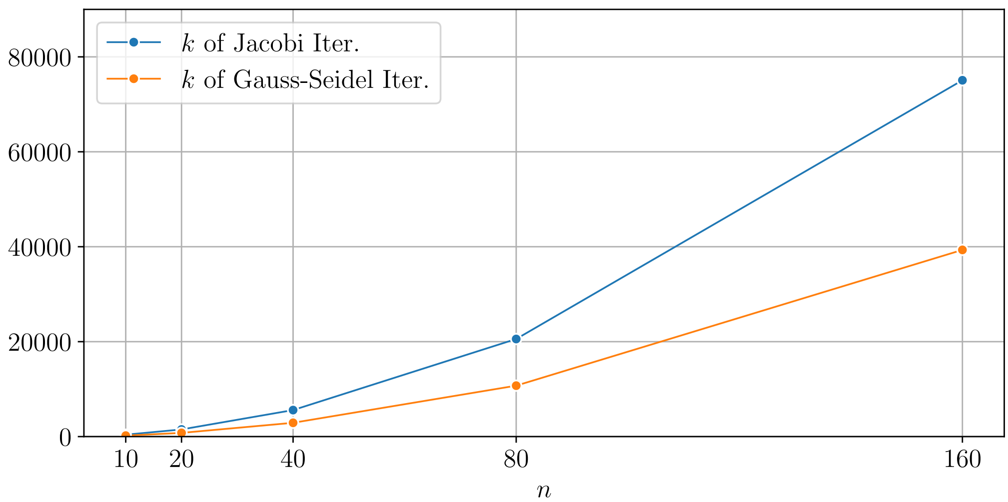

图 4: 求解线性方程组所需迭代次数

得出如下结论:

- 随 A_h 的维数增大, Jacobi 和 Gauss-Seidel 方法所需的迭代次数呈非线性增长, 且增长速率加快, 具有 $\Omega(n)$ 复杂度. 可见, 求解线性方程组的迭代方法存在效率瓶颈, 不一定优于直接解法.
- 在此问题中, 当 n 相同时, Gauss-Seidel 方法的收敛速度总是比 Jacobi 方法快接近一倍, 效率相对较高. 可见, 在迭代计算过程中及时更新 $\boldsymbol{u}_h$ 分量是改善收敛的有效措施.
- 计算发现, 当 n 趋于无穷时, Jacobi 迭代矩阵的谱半径和 Gauss-Seidel 迭代矩阵的 ∞-范数单调趋于 1, 这与迭代次数的增长存在一定相关性. 可见, 线性方程组的迭代收敛取决于迭代矩阵的性质.

- 相较求解非线性方程组的 Newton 迭代法, 求解 (3) 式线性方程组的 Jacobi 和 Gauss-Seidel 方法收敛所需的迭代次数较多, 收敛速度较慢.

5.

当 $\alpha = 1$ 时, $\boldsymbol{u}_h$ 满足的非线性方程组为

$$\begin{pmatrix} -u_0 + 2u_1 - u_2 + h^2u_1^3 - h^2 f_1 \\ -u_1 + 2u_2 - u_3 + h^2u_2^3 - h^2 f_2 \\ \vdots \\ -u_{n-2} + 2u_{n-1} - u_n + h^2u_{n-1}^3 - h^2 f_{n-1} \end{pmatrix} = \mathbf{0} \tag{6}$$

6.

利用 Newton 迭代法解得 $\boldsymbol{u}_h$. 取 $x = 0.1, 0.2, \cdots, 0.9$ 处的数值解 $u_h(x)$, 如表 5 所示. 当 $n = 10, 160$ 时, 作得 $u_h(x)$ 的图象, 如图 5 所示.

表 5: Newton 迭代法求得数值解

x	$u_{0.1}(x)$	$u_{0.05}(x)$	$u_{0.025}(x)$	$u_{0.0125}(x)$	$u_{0.00625}(x)$	$u_e(x)$
0.1	0.31114144	0.30954651	0.30914928	0.30905006	0.30902526	0.30901699
0.2	0.59179026	0.58878402	0.58803479	0.58784763	0.58780085	0.58778525
0.3	0.81446879	0.81037743	0.80935695	0.80910197	0.80903824	0.80901699
0.4	0.95740831	0.95264236	0.95145284	0.95115559	0.95108128	0.95105652
0.5	1.00665583	1.00166208	1.00041540	1.00010384	1.00002596	1.00000000
0.6	0.95740831	0.95264236	0.95145284	0.95115559	0.95108128	0.95105652
0.7	0.81446879	0.81037743	0.80935695	0.80910197	0.80903824	0.80901699
0.8	0.59179026	0.58878402	0.58803479	0.58784763	0.58780085	0.58778525
0.9	0.31114144	0.30954651	0.30914928	0.30905006	0.30902526	0.30901699

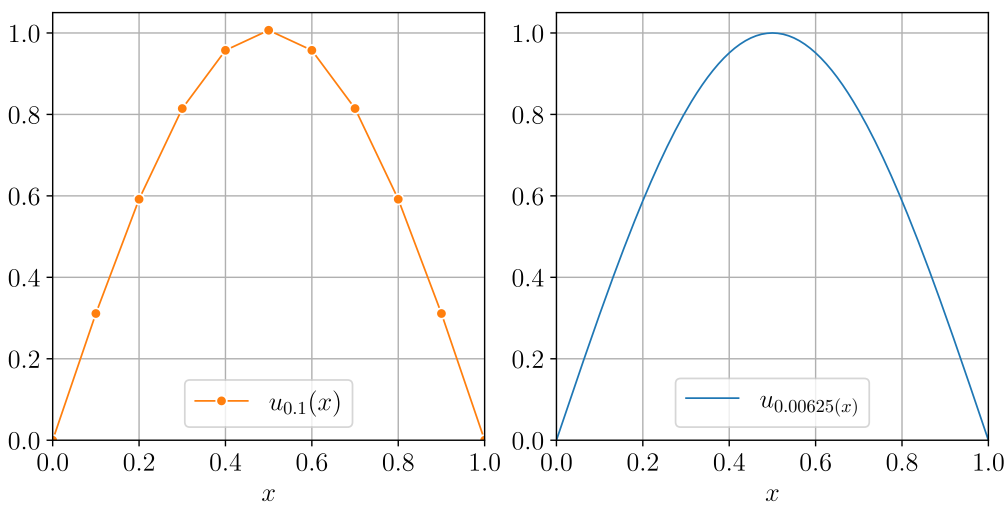

图 5: Newton 迭代法求得数值解

求得误差 $e_h = ||\boldsymbol{u}_h - \boldsymbol{u}_e||_2$, 如表 6 所示.

表 6: $\alpha = 1$ 时的逼近误差

n	h	e_h	$\log h$	$\log e_h$
10	0.1	6.65582771×10^{-3}	-2.30258509	-5.01226246
20	0.05	1.66207605×10^{-3}	-2.99573227	-6.39968782
40	0.025	4.15398609×10^{-4}	-3.68887945	-7.78627200
80	0.0125	1.03842083×10^{-4}	-4.38202663	-9.17263924
160	0.00625	2.59600471×10^{-5}	-5.07517382	-10.55895185

可见, 随区间剖分点数 n 增大, 误差 e_h 逐渐减小而趋于 0.

假设 $e_h = \Theta(h^\beta)$, 则 $\log e_h \approx \beta_0 + \beta \log h$. 利用最小二乘法拟合, 得到曲线如图 6 所示.

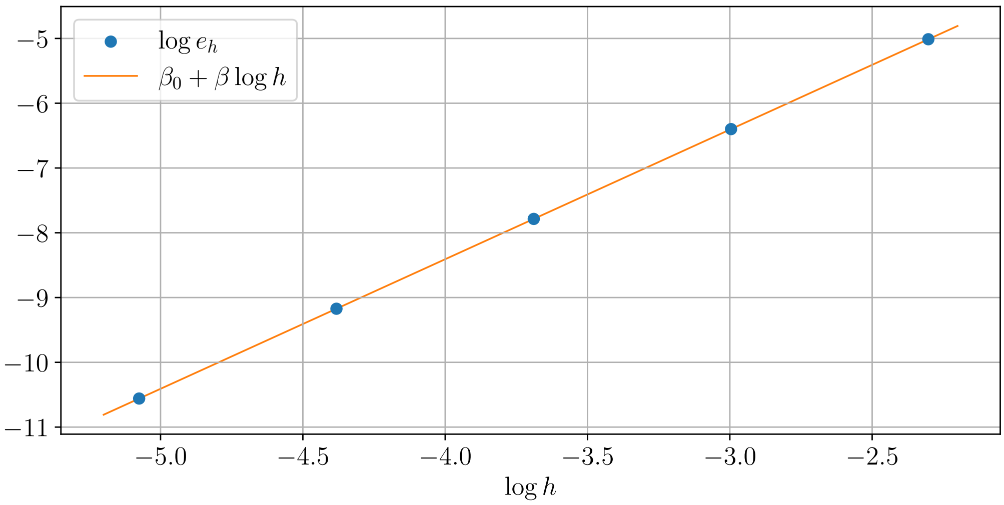

图 6: $\alpha = 1$ 时的误差拟合

求得收敛阶为

$$\beta = 2.00048858 \approx 2 \tag{7}$$

三、 解题过程

1. $\alpha = 0$

1) 线性方程组

记目标函数为

$$F(\boldsymbol{u}_h) = h\left[\sum_{i=1}^{n}\frac{1}{2}\left(\frac{u_i - u_{i-1}}{h}\right)^2 - \sum_{i=1}^{n-1} f_i u_i\right] \tag{8}$$

其梯度为

$$\nabla F(\boldsymbol{u}_h)=\frac{1}{h}\begin{pmatrix}-u_0+2u_1-u_2-h^2f_1\\-u_1+2u_2-u_3-h^2f_2\\\vdots\\-u_{n-2}+2u_{n-1}-u_n-h^2f_{n-1}\end{pmatrix}\tag{9}$$

Hessian 矩阵为

$$\nabla^2F(\boldsymbol{u}_h)=\frac{1}{h}\begin{pmatrix}2&-1&&\\-1&2&\ddots&\\&\ddots&\ddots&-1\\&&-1&2\end{pmatrix}\tag{10}$$

计算得其 m 阶主子式为 $(m+1)/h$, 故 $\nabla^2F(\boldsymbol{u}_h)>0$. $\nabla F(\boldsymbol{u}_h)=\mathbf{0}$ 处为极小值点, 满足

$$\begin{cases}-u_0+2u_1-u_2=h^2f_1\\-u_1+2u_2-u_3=h^2f_2\\\qquad\vdots\\-u_{n-2}+2u_{n-1}-u_n=h^2f_{n-1}\end{cases}\tag{11}$$

写为矩阵形式 $\boldsymbol{A}_h\boldsymbol{u}_h=\boldsymbol{f}_h$, 如 (3) 式所示.

2) Jacobi 迭代法

令 $\boldsymbol{A}_h=\boldsymbol{D}+\boldsymbol{L}+\boldsymbol{U}$, 其中 $\boldsymbol{D}$ 为对角矩阵, $\boldsymbol{L}$ 为下三角矩阵, $\boldsymbol{U}$ 为上三角矩阵, 即

$$\boldsymbol{D}=\frac{1}{h^2}\begin{pmatrix}2&&&\\&2&&\\&&\ddots&\\&&&2\end{pmatrix},\quad \boldsymbol{L}=\boldsymbol{U}^{\mathrm{T}}=\frac{1}{h^2}\begin{pmatrix}&&&\\-1&&&\\&\ddots&&\\&&-1&\end{pmatrix}\tag{12}$$

由 $\boldsymbol{A}_h\boldsymbol{u}_h=\boldsymbol{f}_h$, 得 $\boldsymbol{D}\boldsymbol{u}_h=-(\boldsymbol{L}+\boldsymbol{U})\boldsymbol{u}_h+\boldsymbol{f}_h$, 进而有 Jacobi 迭代公式

$$\boldsymbol{u}_h^{(k+1)}=-\boldsymbol{D}^{-1}(\boldsymbol{L}+\boldsymbol{U})\boldsymbol{u}_h^{(k)}+\boldsymbol{D}^{-1}\boldsymbol{f}_h\tag{13}$$

代入 (12) 式化简得

$$\begin{cases}u_1^{(k+1)}=\dfrac{u_2^{(k)}+h^2f_1}{2}\\u_2^{(k+1)}=\dfrac{u_1^{(k)}+u_3^{(k)}+h^2f_2}{2}\\\qquad\vdots\\u_{n-1}^{(k+1)}=\dfrac{u_{n-2}^{(k)}+h^2f_{n-1}}{2}\end{cases}\tag{14}$$

下证 Jacobi 迭代的收敛性. 迭代矩阵的特征值 λ 满足

$$|\boldsymbol{I}+\lambda\boldsymbol{D}^{-1}(\boldsymbol{L}+\boldsymbol{U})| = \begin{vmatrix} \lambda & -\dfrac{1}{2} & & \\ -\dfrac{1}{2} & \lambda & \ddots & \\ & \ddots & \ddots & -\dfrac{1}{2} \\ & & -\dfrac{1}{2} & \lambda \end{vmatrix} = \frac{\alpha^n-\beta^n}{\alpha-\beta} = 0 \tag{15}$$

其中 α, β 为 $4x^2-4\lambda x+1=0$ 的两根. 若 $\lambda=\pm1$, 则

$$\frac{\alpha^n-\beta^n}{\alpha-\beta} = \frac{n}{(\pm2)^{n-1}} \neq 0 \tag{16}$$

故 $\lambda\neq 1$. 此时应有 $\alpha^n=\beta^n$ 且 $\alpha\neq\beta$, 可知 α, β 必为复根. 于是 $|\lambda|<1$. 事实上,

$$\rho(-\boldsymbol{D}^{-1}(\boldsymbol{L}+\boldsymbol{U})) = \max|\lambda| = \cos\left(\frac{\pi}{n}\right) < 1 \tag{17}$$

因而 Jacobi 迭代收敛.

每次迭代均按 (14) 式计算. 当 $||\Delta\boldsymbol{u}_h^{(k)}||_\infty < \varepsilon$ 时, 满足精度, 终止迭代. 算法描述如下

算法 1 Jacobi 迭代法求解 (3) 式线性方程组

输入: 初始点 $\boldsymbol{u}_h^{(0)}$, 精度控制值 ε.
输出: 迭代次数 k, 第 k 步的迭代解 $\boldsymbol{u}_h^{(k)}$.

$u_0, u_n \leftarrow 0$;
$(u_1,\cdots,u_{n-1}) \leftarrow \boldsymbol{u}_h^{(0)}$;
for $k \leftarrow 1$ **to** `M` **do** // `M` 为最大迭代次数.
  $m \leftarrow 0$; // m 用于计算 $||\Delta\boldsymbol{u}_h^{(k)}||_\infty$.
  for $i \leftarrow 1$ **to** $n-1$ **do**
    $v_i \leftarrow 0.5(u_{i-1}+u_{i+1}+h^2 f_i)$;
    $m \leftarrow \max\{m, |v_i-u_i|\}$;
  end
  if $m<\varepsilon$ **then**
    return $k, (v_1,\cdots,v_{n-1})$;
  end
  $(u_1,\cdots,u_{n-1}) \leftarrow (v_1,\cdots,v_{n-1})$;
end
return `NULL`;

单步迭代的复杂度为 $O(n)$. 用 Python 实现以上算法, 取 $\boldsymbol{u}_h^{(0)}=\boldsymbol{0}$, 求得数值解 $u_h(x)$ 和逼近误差 e_h, 代码见附录 A (linear-iteration.py). 运行程序, 得到结果如表 1, 表 3, 图 1 所示.

3) Gauss-Seidel 迭代法

由 $\boldsymbol{A}_h\boldsymbol{u}_h = \boldsymbol{f}_h$, 得 $(\boldsymbol{D}+\boldsymbol{L})\boldsymbol{u}_h = -\boldsymbol{U}\boldsymbol{u}_h + \boldsymbol{f}_h$, 进而有 Gauss-Seidel 迭代公式

$$\boldsymbol{u}_h^{(k+1)} = -(\boldsymbol{D}+\boldsymbol{L})^{-1}\boldsymbol{U}\boldsymbol{u}_h^{(k)} + (\boldsymbol{D}+\boldsymbol{L})^{-1}\boldsymbol{f}_h \tag{18}$$

代入 (12) 式化简得

$$\begin{cases} u_1^{(k+1)} = \dfrac{u_2^{(k)} + h^2 f_1}{2} \\ u_2^{(k+1)} = \dfrac{u_1^{(k+1)} + u_3^{(k)} + h^2 f_2}{2} \\ \qquad\quad \vdots \\ u_{n-1}^{(k+1)} = \dfrac{u_{n-2}^{(k+1)} + h^2 f_{n-1}}{2} \end{cases} \tag{19}$$

下证 Gauss-Seidel 迭代的收敛性. 迭代矩阵为

$$-(\boldsymbol{D}+\boldsymbol{L})^{-1}\boldsymbol{U} = \begin{pmatrix} 0 & \frac{1}{2} & & & \\ 0 & \frac{1}{4} & \frac{1}{2} & & \\ 0 & \frac{1}{8} & \frac{1}{4} & \ddots & \\ 0 & \vdots & \vdots & \ddots & \frac{1}{2} \\ 0 & \frac{1}{2^{n-1}} & \frac{1}{2^{n-2}} & \cdots & \frac{1}{4} \end{pmatrix} \tag{20}$$

当 $n \geq 3$ 时, 其 ∞-范数为 $1 - 2^{-(n-1)} < 1$, 故 Gauss-Seidel 迭代收敛.

每次迭代均按 (19) 式计算. 当 $||\Delta \boldsymbol{u}_h^{(k)}||_\infty < \varepsilon$ 时, 满足精度, 终止迭代. 算法描述如下

算法 2 Gauss-Seidel 迭代法求解 (3) 式线性方程组

输入: 初始点 $\boldsymbol{u}_h^{(0)}$, 精度控制值 ε.

输出: 迭代次数 k, 第 k 步的迭代解 $\boldsymbol{u}_h^{(k)}$.

$u_0, u_n \leftarrow 0$;
$(u_1, \cdots, u_{n-1}) \leftarrow \boldsymbol{u}_h^{(0)}$;
for $k \leftarrow 1$ **to** `M` **do** // `M` 为最大迭代次数.
  $m \leftarrow 0$; // m 用于计算 $||\Delta \boldsymbol{u}_h^{(k)}||_\infty$.
  $v_1 \leftarrow 0.5(u_2 + h^2 f_1)$;
  for $i \leftarrow 2$ **to** $n-1$ **do**
    $v_i \leftarrow 0.5(v_{i-1} + u_{i+1} + h^2 f_i)$;
    $m \leftarrow \max\{m, |v_i - u_i|\}$;
  end
  if $m < \varepsilon$ **then**
    return $k, (v_1, \cdots, v_{n-1})$;
  end
  $(u_1, \cdots, u_{n-1}) \leftarrow (v_1, \cdots, v_{n-1})$;
end
return `NULL`;

单步迭代的复杂度为 $O(n)$. 用 Python 实现以上算法, 取 $\boldsymbol{u}_h^{(0)} = \mathbf{0}$, 求得数值解 $u_h(x)$ 和逼近误差 e_h, 代码见附录 A (linear-iteration.py). 运行程序, 得到结果如表 2, 表 3, 图 2 所示.

2. $\alpha = 1$

1) 非线性方程组

目标函数为

$$F(\boldsymbol{u}_h) = h\left[\sum_{i=1}^{n} \frac{1}{2}\left(\frac{u_i - u_{i-1}}{h}\right)^2 + \sum_{i=1}^{n-1}\left(\frac{u_i^4}{4} - f_i u_i\right)\right] \tag{21}$$

其梯度为

$$\boldsymbol{\nabla} F(\boldsymbol{u}_h) = \frac{1}{h}\begin{pmatrix} -u_0 + 2u_1 - u_2 + h^2 u_1^3 - h^2 f_1 \\ -u_1 + 2u_2 - u_3 + h^2 u_2^3 - h^2 f_2 \\ \vdots \\ -u_{n-2} + 2u_{n-1} - u_n + h^2 u_{n-1}^3 - h^2 f_{n-1} \end{pmatrix} \tag{22}$$

Hessian 矩阵为

$$\boldsymbol{\nabla}^2 F(\boldsymbol{u}_h) = \frac{1}{h}\begin{pmatrix} 2+3h^2u_1^2 & -1 & & \\ -1 & 2+3h^2u_2^2 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2+3h^2u_{n-1}^2 \end{pmatrix} \tag{23}$$

易证其 m 阶主子式不小于 $(m+1)/h$, 故 $\boldsymbol{\nabla}^2 F(\boldsymbol{u}_h) > 0$. $\boldsymbol{\nabla} F(\boldsymbol{u}_h) = \mathbf{0}$ 处为极小值点, 满足

$$\begin{cases} -u_0 + 2u_1 - u_2 + h^2u_1^3 - h^2 f_1 = 0 \\ -u_1 + 2u_2 - u_3 + h^2u_2^3 - h^2 f_2 = 0 \\ \qquad\qquad\qquad \vdots \\ -u_{n-2} + 2u_{n-1} - u_n + h^2u_{n-1}^3 - h^2 f_{n-1} = 0 \end{cases} \tag{24}$$

记作 $\boldsymbol{g}(\boldsymbol{u}_h) = (g_1(\boldsymbol{u}_h), \cdots, g_{n-1}(\boldsymbol{u}_h))^{\mathrm{T}} = \mathbf{0}$, 如 (6) 式所示.

2) Newton 迭代法

记 $\boldsymbol{J}(\boldsymbol{u}_h) = (\boldsymbol{\nabla} g_1(\boldsymbol{u}_h), \cdots, \boldsymbol{\nabla} g_{n-1}(\boldsymbol{u}_h))^{\mathrm{T}}$ 为 $\boldsymbol{g}(\boldsymbol{u}_h)$ 的 Jacobian 矩阵, 则 Newton 迭代公式为

$$\boldsymbol{u}_h^{(k+1)} = \boldsymbol{u}_h^{(k)} - \boldsymbol{J}^{-1}(\boldsymbol{u}_h^{(k)})\boldsymbol{g}(\boldsymbol{u}_h^{(k)}) \tag{25}$$

当 $\boldsymbol{u}_h^{(0)}$ 充分接近 $\boldsymbol{u}_h$ 时, Newton 迭代收敛. 注意到

$$\boldsymbol{J}(\boldsymbol{u}_h) = h\boldsymbol{\nabla}^2 F(\boldsymbol{u}_h) = \begin{pmatrix} 2+3h^2u_1^2 & -1 & & \\ -1 & 2+3h^2u_1^2 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2+3h^2u_{n-1}^2 \end{pmatrix} \tag{26}$$

每步迭代, 先求解 $\boldsymbol{J}(\boldsymbol{u}_h^{(k)})\Delta\boldsymbol{u}_h^{(k+1)} = -\boldsymbol{g}(\boldsymbol{u}_h^{(k)})$, 再计算 $\boldsymbol{u}_h^{(k+1)} = \boldsymbol{u}_h^{(k)} + \Delta\boldsymbol{u}_h^{(k+1)}$. 当 $||\Delta\boldsymbol{u}_h^{(k)}||_\infty < \varepsilon$ 时, 满足精度, 终止迭代. 算法描述如下

算法 3 Newton 迭代法求解 (6) 式非线性方程组

输入： 初始点 $\boldsymbol{u}_h^{(0)}$, 精度控制值 ε.

输出： 迭代次数 k, 第 k 步的迭代解 $\boldsymbol{u}_h^{(k)}$.

$\boldsymbol{u}_h \leftarrow \boldsymbol{u}_h^{(0)}$;

for $k \leftarrow 1$ **to** `M` **do** // `M` 为最大迭代次数.

  按 (26) 式计算 $\boldsymbol{J}(\boldsymbol{u}_h)$;

  求解 $\boldsymbol{J}(\boldsymbol{u}_h)\Delta\boldsymbol{u}_h = -\boldsymbol{g}(\boldsymbol{u}_h)$;

  $\boldsymbol{u}_h \leftarrow \boldsymbol{u}_h + \Delta\boldsymbol{u}_h$;

  if $||\Delta\boldsymbol{u}_h||_\infty < \varepsilon$ **then**

    return $k, \boldsymbol{u}_h$;

  end

end

return `NULL`; // $\boldsymbol{u}_h^{(0)}$ 附近无解.

用 Python 实现以上算法, 取 $\boldsymbol{u}_h^{(0)} = \boldsymbol{0}$, 求得数值解 $u_h(x)$ 和逼近误差 e_h, 代码见附录 A (newton.py). 运行程序, 得到结果如表 5, 表 6, 图 5 所示.

3. 最小二乘法求收敛阶

设逼近误差 $e_h = C(h)h^\beta = \Theta(h^\beta)$, 两边取对数得

$$\log e_h = \log C(h) + \beta \log h \tag{27}$$

设 $\log C(h) = \beta_0 + \varepsilon$, 其中 β_0 为常数, $\varepsilon = \varepsilon(h)$ 为干扰项. 记 $y = \log e_h$, $x = \log h$, 则

$$y = \beta_0 + \beta x + \varepsilon \approx \beta_0 + \beta x \tag{28}$$

根据表 3 或表 6 中数据, 记

$$\boldsymbol{y} = \begin{pmatrix} \log e_{0.1} \\ \log e_{0.05} \\ \vdots \\ \log e_{0.00625} \end{pmatrix}, \boldsymbol{X} = \begin{pmatrix} 1 & \log 0.1 \\ 1 & \log 0.05 \\ \vdots & \vdots \\ 1 & \log 0.00625 \end{pmatrix}, \boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta \end{pmatrix} \tag{29}$$

则误差平方和为

$$\begin{aligned} \hat{Q} &= (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) \\ &= \boldsymbol{y}^{\mathrm{T}}\boldsymbol{y} - 2\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} + \boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{\beta} \end{aligned} \tag{30}$$

对 $\boldsymbol{\beta}$ 求梯度和 Hessian 矩阵

$$\begin{aligned} \boldsymbol{\nabla}_\beta \hat{Q} &= -2\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} + 2\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{\beta} \\ \boldsymbol{\nabla}_\beta^2 \hat{Q} &= 2\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} \geq 0 \end{aligned} \tag{31}$$

故 $\hat{Q}(\boldsymbol{\beta})$ 取最小值时必有 $\boldsymbol{\nabla}_{\boldsymbol{\beta}}\hat{Q} = 0$, 即 $\boldsymbol{\beta}$ 满足法方程

$$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{\beta} = \boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} \tag{32}$$

其公式解为

$$\boldsymbol{\beta} = (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} = \begin{pmatrix} \dfrac{\sum x^2 \sum y - \sum x \sum xy}{\sum 1 \sum x^2 - (\sum x)^2} \\ \dfrac{\sum 1 \sum xy - \sum x \sum y}{\sum 1 \sum x^2 - (\sum x)^2} \end{pmatrix} \tag{33}$$

算法描述如下

算法 4 最小二乘法求 (2) 式数值方法收敛阶

输入: 数据点个数 N, 步长 h_1~h_N, 逼近误差 e_{h_1}~e_{h_N}.

输出: 收敛阶 β.

$(a, b, c, d) \leftarrow \mathbf{0}$;
for $i \leftarrow 1$ **to** N **do**
  $(x, y) \leftarrow (\log h_i, \log e_{h_i})$;
  $(a, b, c, d) \leftarrow (a, b, c, d) + (x_i, y_i, x_i^2, x_i y_i)$;
end
return $(Nd - ab)/(Nc - a^2)$;

时间复杂度为 $O(N)$. 用 Python 实现以上算法, 代码见附录 A (linear-ols.py).

代入表 3 数据, 求得 $\alpha = 0$, $f(x) = \pi^2 \sin(\pi x)$ 情形下, 基于 Jacobi 迭代的数值算法的收敛阶 $\beta = 2.00642821$ 及 $\beta_0 = -0.17388792$; 基于 Gauss-Seidel 迭代的数值算法的收敛阶 $\beta = 2.00399771$ 及 $\beta_0 = -0.18111161$. 拟合得到的曲线如图 3 所示.

代入表 6 数据, 求得 $\alpha = 1$, $f(x) = \pi^2 \sin(\pi x) + \sin^3(\pi x)$ 情形下, 基于 Newton 迭代的数值算法的收敛阶 $\beta = 2.00048858$ 及 $\beta_0 = -0.40640145$. 拟合得到的曲线如图 6 所示.

附录 A　Python 程序代码

1. linear-iteration.py

```python
import numpy as np
import numpy.linalg as la
import matplotlib as mpl
import matplotlib.pyplot as plt
import pandas as pd

mpl.rc('font', family='serif', size=15)
mpl.rc('text', usetex=True)
mpl.rc('text.latex', preamble=r'\usepackage{bm}')

out_dir = '../assets/output/'


# Jacobi iteration
def jacobi(f, u0, ue=None, eps=1e-5, max_iter=100000, print_flag=False):
    u = np.vstack((0, u0, 0))
    v = np.empty(u0.shape)
    df = pd.DataFrame(columns=[f'||u-ue||2'] + [f'||Δu||∞'])
    if print_flag:
        if ue is None: ue = np.zeros(u0.shape)
        df.loc[0] = [la.norm(u0 - ue, np.inf), np.nan]
    for k in range(max_iter):
        m = 0
        for i in range(1, u.shape[0] - 1):
            v[i - 1] = (u[i - 1] + u[i + 1] + f[i - 1]) / 2
            m = np.max([m, np.abs(v[i - 1, 0] - u[i, 0])])
        if print_flag: df.loc[k + 1] = [la.norm(v - ue, np.inf), m]
        for i in range(1, u.shape[0] - 1):
            u[i] = v[i - 1]
        if m < eps: return (v, df) if print_flag else (v, k, la.norm(v - ue, np.inf))
    raise RuntimeError('Failed to converge.')


# Gauss-Seidel iteration
def gauss_seidel(f, u0, ue=None, eps=1e-5, max_iter=100000, print_flag=False):
    u = np.vstack((0, u0, 0))
    v = np.empty(u0.shape)
    df = pd.DataFrame(columns=[f'||u-ue||2'] + [f'||Δu||∞'])
    if print_flag:
        if ue is None: ue = np.zeros(u0.shape)
        df.loc[0] = [la.norm(u0 - ue, np.inf), np.nan]
    for k in range(max_iter):
        m = 0
        v[0] = (u[2] + f[0]) / 2
```

```python
        for i in range(2, u.shape[0] - 1):
            v[i - 1] = (v[i - 2] + u[i + 1] + f[i - 1]) / 2
            m = np.max([m, np.abs(v[i - 1, 0] - u[i, 0])])
        if print_flag: df.loc[k + 1] = [la.norm(v - ue, np.inf), m]
        for i in range(1, u.shape[0] - 1):
            u[i] = v[i - 1]
        if m < eps: return (v, df) if print_flag else (v, k, la.norm(v - ue, np.inf))
    raise RuntimeError('Failed to converge.')


# Main
uk_dict1, uk_dict2 = {}, {}

for n in [10, 20, 40, 80, 160]:
    h = 1 / n
    x = np.linspace([h], [1 - h], n - 1)

    f = np.pi**2 * np.sin(np.pi * x)
    f *= h**2

    u0 = np.zeros((n - 1, 1))
    ue = np.sin(np.pi * x)

    uk_dict1[n] = jacobi(f, u0, ue, eps=1e-10)
    uk_dict2[n] = gauss_seidel(f, u0, ue, eps=1e-10)

# Data CSV
dict = {'x': np.linspace(0.1, 0.9, 9)}
for n in [10, 20, 40, 80, 160]:
    h = 1 / n
    dict[f'u{h}'] = uk_dict1[n][0][:, 0][int(n / 10 - 1)::int(n / 10)]
dict['ue'] = np.sin(np.pi * dict['x'])
df = pd.DataFrame(dict)
df.to_csv(out_dir + 'jacobi.csv')

dict = {'x': np.linspace(0.1, 0.9, 9)}
for n in [10, 20, 40, 80, 160]:
    h = 1 / n
    dict[f'u{h}'] = uk_dict2[n][0][:, 0][int(n / 10 - 1)::int(n / 10)]
dict['ue'] = np.sin(np.pi * dict['x'])
df = pd.DataFrame(dict)
df.to_csv(out_dir + 'gauss-seidel.csv')

# Data plot
fig, ax = plt.subplots(1, 2, figsize=(8, 4), constrained_layout=True)

n = 10
h, x = 1 / n, np.linspace(0, 1, n + 1)
```

```python
ax[0].plot(x, np.vstack((0, uk_dict1[n][0], 0)), 'C1o-', mec='1.0', linewidth=1, label=r'$u_{' + f'{h}' + '}(x)$')

n = 160
h, x = 1 / n, np.linspace(0, 1, n + 1)
ax[1].plot(x, np.vstack((0, uk_dict1[n][0], 0)), linewidth=1, label=r'$u_{' + f'{h}' + '(x)}$')

for i in [0, 1]:
    ax[i].set_xlabel(r'$x$')
    ax[i].set_xlim(0.0, 1.0)
    ax[i].set_ylim(0.0, 1.05)
    ax[i].grid()
    ax[i].legend()

fig.savefig(out_dir + 'jacobi.pdf')

fig, ax = plt.subplots(1, 2, figsize=(8, 4), constrained_layout=True)

n = 10
h, x = 1 / n, np.linspace(0, 1, n + 1)
ax[0].plot(x, np.vstack((0, uk_dict2[n][0], 0)), 'C1o-', mec='1.0', linewidth=1, label=r'$u_{' + f'{h}' + '}(x)$')

n = 160
h, x = 1 / n, np.linspace(0, 1, n + 1)
ax[1].plot(x, np.vstack((0, uk_dict2[n][0], 0)), linewidth=1, label=r'$u_{' + f'{h}' + '(x)}$')

for i in [0, 1]:
    ax[i].set_xlabel(r'$x$')
    ax[i].set_xlim(0.0, 1.0)
    ax[i].set_ylim(0.0, 1.05)
    ax[i].grid()
    ax[i].legend()

fig.savefig(out_dir + 'gauss-seidel.pdf')

# Error analysis
df = pd.DataFrame([[n, 1 / n, uk_dict1[n][2], uk_dict2[n][2]] for n in [10, 20, 40, 80, 160]], columns=['n', 'h', 'eh1', 'eh2'])
df['logh'] = np.log(df['h'])
df['logeh1'] = np.log(df['eh1'])
df['logeh2'] = np.log(df['eh2'])
df.to_csv(out_dir + 'linear-itr-error.csv')

# Convergence
```

```python
df = pd.DataFrame([[n, uk_dict1[n][1], uk_dict2[n][1]] for n in [10, 20, 40, 80, 160]], columns=['n', 'k1', 'k2'])
df.to_csv(out_dir + 'linear-itr-k.csv')

fig, ax = plt.subplots(figsize=(8, 4), constrained_layout=True)

ax.set_xticks(df['n'])
ax.set_xlabel(r'$n$')
ax.set_ylim(0, 90000)
ax.plot(df['n'], df['k1'], 'C0o-', mec='1.0', linewidth=1, label=r'$k$ of Jacobi Iter.')
ax.plot(df['n'], df['k2'], 'C1o-', mec='1.0', linewidth=1, label=r'$k$ of Gauss-Seidel Iter.')
ax.grid()
ax.legend()

fig.savefig(out_dir + 'linear-itr-k.pdf')
```

2. newton.py

```python
import numpy as np
import numpy.linalg as la
import matplotlib as mpl
import matplotlib.pyplot as plt
import pandas as pd

mpl.rc('font', family='serif', size=15)
mpl.rc('text', usetex=True)
mpl.rc('text.latex', preamble=r'\usepackage{bm}')

out_dir = '../assets/output/'


# Newton iteration
def newton(g, J, u0, ue=None, eps=1e-5, max_iter=100, print_flag=False):
    u, gu = u0.copy(), g(u0)
    df = pd.DataFrame(columns=[f'||u-ue||2'] + [f'||g(u)||∞'] + [f'||Δu||∞'])
    if print_flag:
        if ue is None: ue = np.zeros(u0.shape)
        df.loc[0] = [la.norm(u0 - ue, np.inf), la.norm(gu, np.inf), np.nan]
    for k in range(max_iter):
        Ju = J(u)
        du = la.solve(Ju, -gu)
        u += du
        gu = g(u)
        if print_flag:
```

```python
            df.loc[k + 1] = [la.norm(u - ue, np.inf), la.norm(gu, np.inf), la.norm(du,
            ↪   np.inf)]
        if la.norm(du, np.inf) < eps:
            return (u, df) if print_flag else (u, k, la.norm(u - ue, np.inf))
    raise RuntimeError('Failed to converge.')


# Main
def g(u):
    g1 = np.empty((n - 1, 1))
    g1[0] = 2 * u[0] - u[1] + h**2 * u[0]**3 - h**2 * f[0]
    g1[n - 2] = -u[n - 3] + 2 * u[n - 2] + h**2 * u[n - 2]**3 - h**2 * f[n - 2]
    for i in range(1, n - 2):
        g1[i] = -u[i - 1] + 2 * u[i] - u[i + 1] + h**2 * u[i]**3 - h**2 * f[i]
    return g1


def J(u):
    J1 = J0.copy()
    for i in range(0, n - 1):
        J1[i, i] += 3 * h**2 * u[i]**2
    return J1


uk_dict, df_dict = {}, {}

for n in [10, 20, 40, 80, 160]:
    h = 1 / n
    x = np.linspace([h], [1 - h], n - 1)

    f = np.sin(np.pi * x)
    f = np.pi**2 * f + f**3

    u0 = np.zeros((n - 1, 1))
    ue = np.sin(np.pi * x)

    J0 = np.zeros((n - 1, n - 1))
    J0[0, 0] = 2
    for i in range(1, n - 1):
        J0[i - 1, i] = -1
        J0[i, i - 1] = -1
        J0[i, i] = 2

    uk_dict[n], df_dict[n] = newton(g, J, u0, ue, eps=1e-8, print_flag=True)

# Data plot
fig, ax = plt.subplots(1, 2, figsize=(8, 4), constrained_layout=True)

```

```python
n = 10
h, x = 1 / n, np.linspace(0, 1, n + 1)
ax[0].plot(x, np.vstack((0, uk_dict[n], 0)), 'C1o-', mec='1.0', linewidth=1, label=r'$u_{' +
↪   f'{h}' + '}(x)$')

n = 160
h, x = 1 / n, np.linspace(0, 1, n + 1)
ax[1].plot(x, np.vstack((0, uk_dict[n], 0)), linewidth=1, label=r'$u_{' + f'{h}' + '(x)}$')

for i in [0, 1]:
    ax[i].set_xlabel(r'$x$')
    ax[i].set_xlim(0.0, 1.0)
    ax[i].set_ylim(0.0, 1.05)
    ax[i].grid()
    ax[i].legend()

fig.savefig(out_dir + 'newton.pdf')

# Data CSV
dict = {'x': np.linspace(0.1, 0.9, 9)}
for n in [10, 20, 40, 80, 160]:
    h = 1 / n
    dict[f'u{h}'] = uk_dict[n][:, 0][int(n / 10 - 1)::int(n / 10)]
dict['ue'] = np.sin(np.pi * dict['x'])
df = pd.DataFrame(dict)
df.to_csv(out_dir + 'newton.csv')

# Error analysis
df = pd.DataFrame([[n, 1 / n, df_dict[n]['||u-ue||2'].iloc[-1]] for n in [10, 20, 40, 80,
↪   160]], columns=['n', 'h', 'eh'])
df['logh'] = np.log(df['h'])
df['logeh'] = np.log(df['eh'])
df.to_csv(out_dir + 'newton-error.csv')
```

3. linear-ols.py

```python
import numpy as np
import numpy.linalg as la
import matplotlib as mpl
import matplotlib.pyplot as plt
import pandas as pd

mpl.rc('font', family='serif', size=15)
mpl.rc('text', usetex=True)
mpl.rc('text.latex', preamble=r'\usepackage{bm}')

```

```python
out_dir = '../assets/output/'


# Linear OLS
def fit(N, h, e):
    a, b, c, d = 0, 0, 0, 0
    for i in range(N):
        x, y = np.log(h[i]), np.log(e[i])
        a += x
        b += y
        c += x**2
        d += x * y
    return ((N * d - a * b), (b * c - a * d)) / (N * c - a**2)


# Newton iteration
df = pd.read_csv('../assets/output/newton-error.csv')

c = fit(df.shape[0], df['h'].array, df['eh'].array)
print(c)

fig, ax = plt.subplots(figsize=(8, 4), constrained_layout=True)
ax.set_xlabel(r'$\log h$')
ax.scatter(df['logh'], df['logeh'], zorder=2, label=r'$\log e_h$', marker='o')
ax.plot([-5.2, -2.2], np.polyval(c, [-5.2, -2.2]), label=r'$\beta_0 + \beta \log h$',
    zorder=1, linewidth=1, c='C1')
ax.grid()
ax.legend()
fig.savefig(out_dir + 'newton-error.pdf')

# Jacobi and Gauss-Seidel iterations
df = pd.read_csv('../assets/output/linear-itr-error.csv')

c1 = fit(df.shape[0], df['h'].array, df['eh1'].array)
c2 = fit(df.shape[0], df['h'].array, df['eh2'].array)
print(c1)
print(c2)

fig, ax = plt.subplots(2, 1, figsize=(8, 8), constrained_layout=True)

ax[0].set_xlabel(r'$\log h$')
ax[0].scatter(df['logh'], df['logeh1'], zorder=2, label=r'$\log e_h$ (Jacobi Iter.)',
    marker='o')
ax[0].plot([-5.2, -2.2], np.polyval(c1, [-5.2, -2.2]), label=r'$\beta_0 + \beta \log h$',
    zorder=1, linewidth=1, c='C1')
ax[0].grid()
ax[0].legend()

```

```
ax[1].set_xlabel(r'$\log h$')
ax[1].scatter(df['logh'], df['logeh2'], zorder=2, label=r'$\log e_h$ (Gauss-Seidel Iter.)', marker='o', c='C2')
ax[1].plot([-5.2, -2.2], np.polyval(c2, [-5.2, -2.2]), label=r'$\beta_0 + \beta \log h$', zorder=1, linewidth=1, c='C3')
ax[1].grid()
ax[1].legend()

fig.savefig(out_dir + 'linear-itr-error.pdf')
```