Solutions to Problems 2

Chuizheng Kong

October 29, 2024

2

2.1

Let us solve the Poisson equation in 2D on a uniform, 20 × 20 rectangular grid whose coordinates span over $([-10, 10], [-10, 10])$:

$$\nabla^2\varphi(r) = \rho(r) = e^{-r^2} \tag{2.1.1}$$

This means the size of each grid cell is 1×1, and the value of that cell is defined at cell centers:

$$\begin{cases} x_i = -10 + (i - 0.5) \\ y_j = -10 + (j - 0.5) \end{cases} \tag{2.1.2}$$

Using finite difference in this problem, this system of equations (400 in total!) can be expressed as

$$\varphi_{i+1,j} + \varphi_{i-1,j} + \varphi_{i,j+1} + \varphi_{i,j-1} - 4\varphi_{i,j} = \rho_{i,j} = e^{-(x_i^2+y_i^2)} \quad (i, j = 1, 2, ..., 20) \tag{2.1.3}$$

Additional boundary conditions must also be prescribed, and we set $\varphi = 0$ at $x, y = \pm 10$. To implement this boundary condition, you will need to include one additional "ghost cell" surrounding your grid, and enforce:

$$\varphi_{0,j} = -\varphi_{1,j} \tag{2.1.4}$$
$$\varphi_{N+1,j} = -\varphi_{N,j} \tag{2.1.5}$$
$$\varphi_{j,0} = -\varphi_{j,1} \tag{2.1.6}$$
$$\varphi_{j,N+1} = -\varphi_{j,N} \tag{2.1.7}$$

so that one finds $\varphi = 0$ at the grid boundary after averaging the last grid cell and the ghost cell. Note that this boundary condition must be enforced after every numerical step. We ask you to write your own solver using the following iterative methods: (a) Jacobi; (b) Gauss-Seidel; (c) successive over-relaxation with relaxation factor $\omega = 1.5$; (d) steepest descent; (e) conjugate gradient.

2.1.1

How many iterations are required for each of the methods to converge to within $||R||_2 < 10^{-6}$? Show the converged solution. To better present your results, we further ask you to calculate the residual vector R (i.e. a 20×20 matrix) for up to 100 steps, and plot $||R||_2$ as a function of number of iterations.

Solution: Numbers of iterations required for the methods to converge are listed in Table 1. The converged solutions are shown in Figure 1. $||R||_2$ as a function of number of iterations are shown in Figure 2.

Method	**Iterations**
Jacobi	1021
Gauss-Seidel	511
successive over-relaxation ($\omega = 1.5$)	164
steepest descent	1021
conjugate gradient	84

Table 1: Number of iterations required for each method to converge.

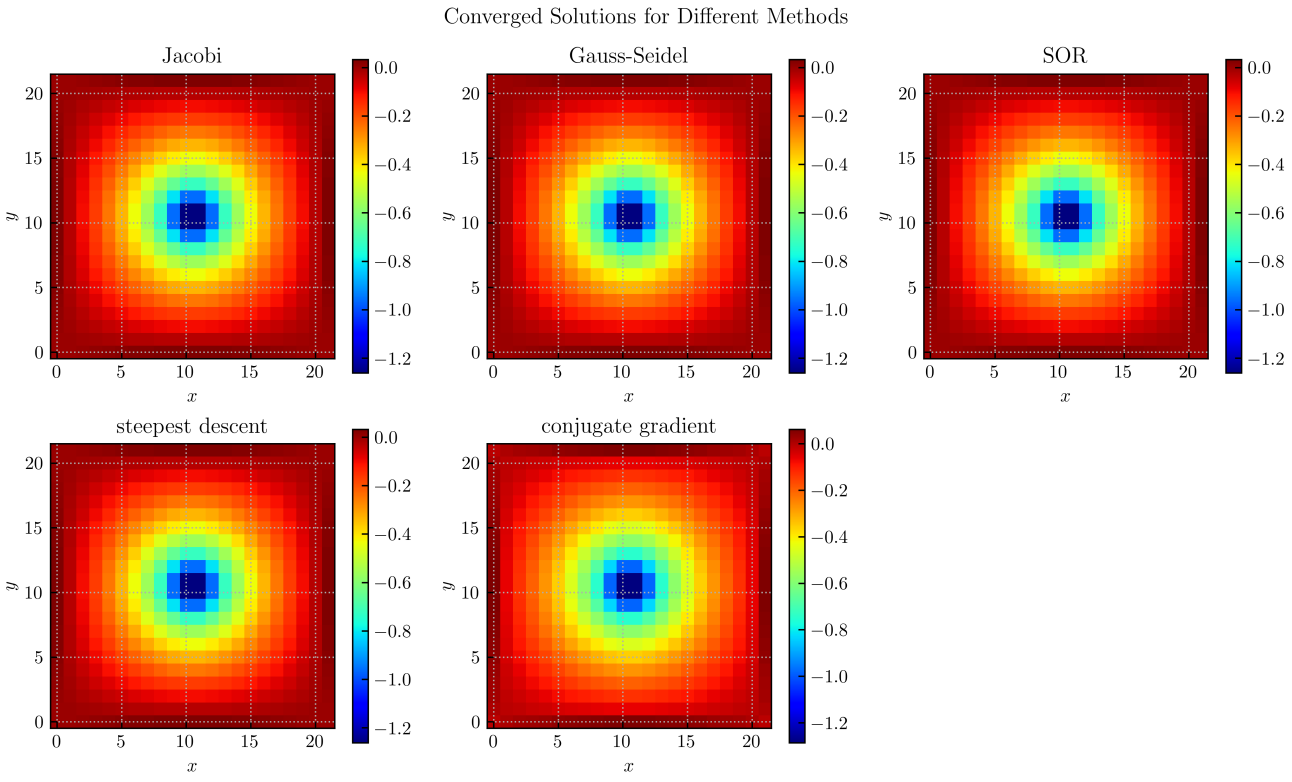

Figure 1: Converged Solutions for Different Methods

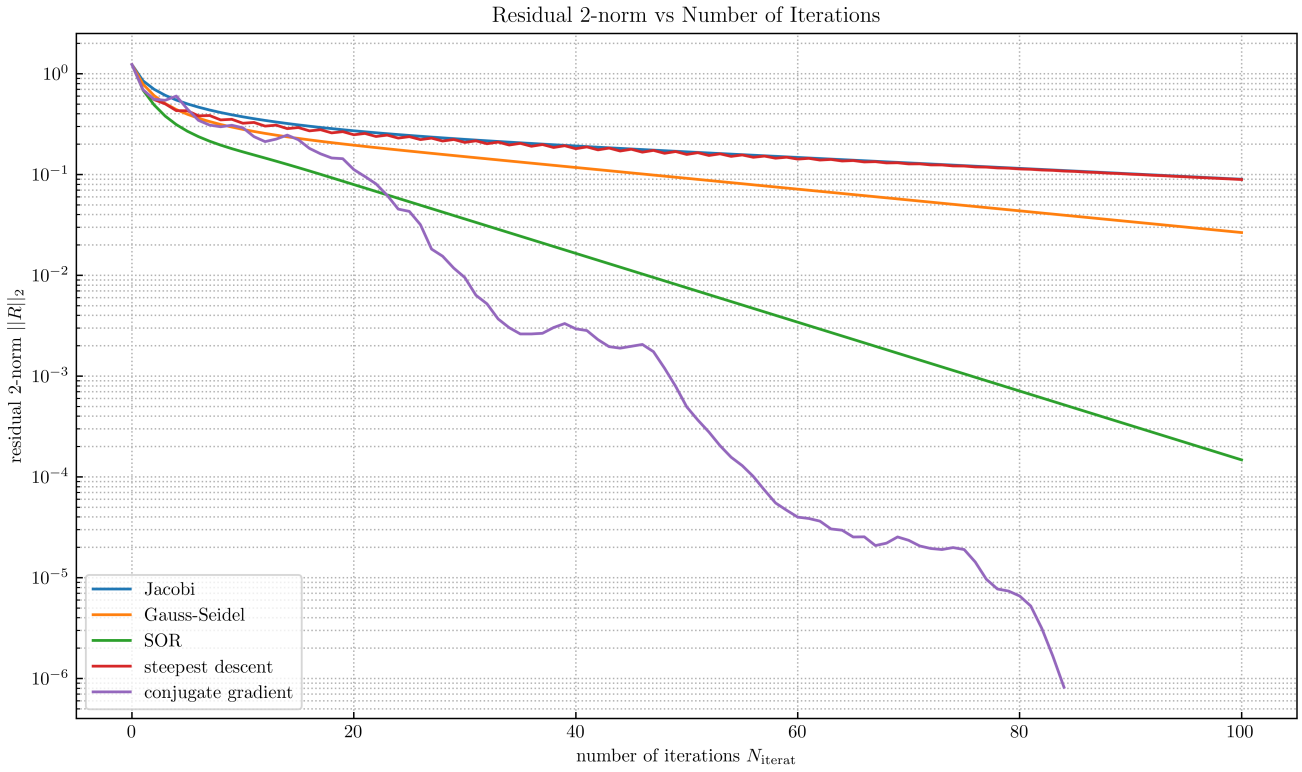

Figure 2: Residual 2-norm vs Number of Iterations

2.1.2

Further discuss the computation cost per iteration for each method.

Solution: We have a system of $20 \times 20 = 400 = N$ linear equations.

1. **Jacobi:** The Jacobi method updates all grid points simultaneously using values from the previous iteration:
$$\varphi_{i,j}^{\text{new}} = \frac{1}{4}\left(\varphi_{i+1,j}^{\text{old}} + \varphi_{i-1,j}^{\text{old}} + \varphi_{i,j+1}^{\text{old}} + \varphi_{i,j-1}^{\text{old}} - \rho_{i,j}\right) \tag{2.1.8}$$
The computational cost is 4 additions and 1 multiplication per grid point, totaling $\approx 5N = 2000$ arithmetic operations per iteration. It also requires 2 copies of the solution grid (old and new values), and the source term ρ, which is typically stored separately. So the memory cost is $\approx 3N = 1200$ floating-point numbers.

2. **Gauss-Seidel:** The Gauss-Seidel method updates grid points sequentially, using the most recent values within the current iteration:
$$\varphi_{i,j}^{\text{new}} = \frac{1}{4}\left(\varphi_{i+1,j}^{\text{old}} + \varphi_{i-1,j}^{\text{new}} + \varphi_{i,j+1}^{\text{old}} + \varphi_{i,j-1}^{\text{new}} - \rho_{i,j}\right) \tag{2.1.9}$$
The computational cost is also 4 additions and 1 multiplication per grid point, totaling $5N = 2000$. It only requires 1 copy of the solution grid (in-place updates), along with the source term ρ. So the memory cost is $\approx 2N = 800$ floating-point numbers.

3. **Successive over-relaxation (SOR):** The SOR method introduces a relaxation factor ω to accel-

Method	Total Operations per Iteration	Memory Cost
Jacobi	$\approx 5N = 2000$	$\approx 3N = 1200$
Gauss-Seidel	$\approx 5N = 2000$	$\approx 2N = 800$
SOR ($\omega = 1.5$)	$\approx 5N = 2000$	$\approx 2N = 800$
steepest gradient descent	$\approx 24N = 9600$	$\approx 12N = 4800$
conjugate gradient descent	$\approx 19N = 7800$	$\approx 14N = 5600$

Table 2: Computational cost per iteration for different numerical methods.

erate convergence:

$$\varphi_{i,j}^{\text{new}} = (1-\omega)\varphi_{i,j}^{\text{old}} + \frac{1}{4}\omega\left(\varphi_{i+1,j}^{\text{old}} + \varphi_{i-1,j}^{\text{new}} + \varphi_{i,j+1}^{\text{old}} + \varphi_{i,j-1}^{\text{new}} - \rho_{i,j}\right) \tag{2.1.10}$$

The computational cost is 5 additions and 2 multiplications per grid point, totaling 2800 arithmetic operations per iteration. Its memory cost is similar to Gauss-Seidel but with an additional scalar relaxation factor ω.

4. **Steepest gradient descent:** The steepest gradient descent method iteratively moves in the direction of the negative gradient:

$$\boldsymbol{r} = b - A\varphi^{\text{old}} \tag{2.1.11}$$

$$\alpha = \frac{\boldsymbol{r}^{\mathrm{T}}\boldsymbol{r}}{\boldsymbol{r}^{\mathrm{T}}A\boldsymbol{r}} \tag{2.1.12}$$

$$\varphi^{\text{new}} = \varphi^{\text{old}} + \alpha\boldsymbol{r} \tag{2.1.13}$$

The derivation of the computational cost is difficult to write down step by step.

5. **Conjugate gradient (CG) descent:** The conjugate gradient descent method solves symmetric positive-definite systems by iteratively updating the solution using conjugate directions. The derivation of the computational cost is also difficult to write down step by step.

2.2

Attached is a file `stars.dat` from the observation (in ASCII). There are 3 columns: the position of the measurement x, the brightness y, and y's standard deviation σ (i.e., noise). I happen to know that the effective point spread function (PSF, φ) when data was taken was a Gaussian, i.e.,

$$\varphi(x) = \frac{1}{\sqrt{2\pi}}e^{-x^2/2} \tag{2.2.1}$$

2.2.1

Plot the line data including 1σ error bars, and identify how many stars (N) there are in the 1D image.

Solution: The plot is shown in Figure 3. By observing the plot carefully, we can identify

$$N = 5 \tag{2.2.2}$$

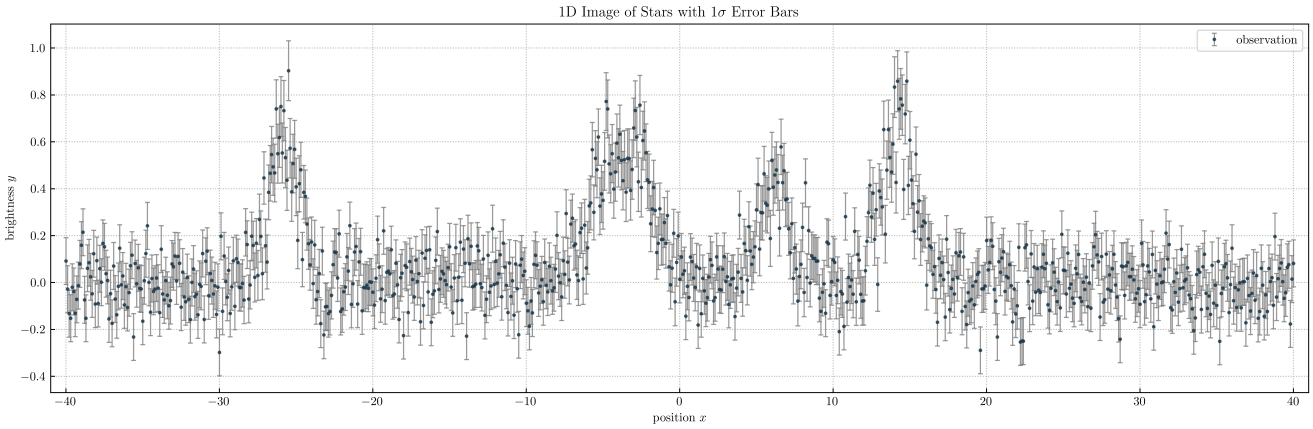

Figure 3: 1D Image of Stars with 1σ Error Bars

2.2.2

Each star j is characterized by its location x_j^{s} and its luminosity A_j, which is typically found by doing a (non-linear) least mean square problem to minimize

$$\chi^2(\boldsymbol{A}, \boldsymbol{x}^{\mathrm{s}}) = \sum_i \frac{\left[y_i - \sum_{j=1}^N A_j \varphi(x_i - x_j^{\mathrm{s}})\right]^2}{\sigma_i^2} = \sum_i \frac{\left[y_i - \frac{1}{\sqrt{2\pi}} \sum_{j=1}^N A_j e^{-(x_i - x_j^{\mathrm{s}})^2/2}\right]^2}{\sigma_i^2} \tag{2.2.3}$$

where summation is taken over all pixels. Is this function a convex function? Why?

Solution: We compute the 2nd derivative with respect to A_j and x_j^{s}:

$$\begin{aligned}
\frac{\partial \chi^2(\boldsymbol{A}, \boldsymbol{x}^{\mathrm{s}})}{\partial A_j} &= \sum_i \frac{2\left[y_i - \sum_{j=1}^N A_j \varphi(x_i - x_j^{\mathrm{s}})\right]}{\sigma_i^2} (-1)\varphi(x_i - x_j^{\mathrm{s}}) \\
&= -\sum_i \frac{2\varphi(x_i - x_j^{\mathrm{s}})}{\sigma_i^2} \left[y_i - \sum_{j=1}^N A_j \varphi(x_i - x_j^{\mathrm{s}})\right] \qquad (2.2.4) \\
\frac{\partial^2 \chi^2(\boldsymbol{A}, \boldsymbol{x}^{\mathrm{s}})}{\partial A_j^2} &= \sum_i \frac{-2\varphi(x_i - x_j^{\mathrm{s}})}{\sigma_i^2} (-1)\varphi(x_i - x_j^{\mathrm{s}}) = \sum_i \frac{2}{\sigma_i^2} \varphi^2(x_i - x_j^{\mathrm{s}}) = \sum_i \frac{1}{\pi\sigma_i^2} e^{-(x_i - x_j^{\mathrm{s}})^2} > 0 \qquad (2.2.5) \\
\frac{\partial \chi^2(\boldsymbol{A}, \boldsymbol{x}^{\mathrm{s}})}{\partial x_j^{\mathrm{s}}} &= \sum_i \frac{2\left[y_i - \sum_{j=1}^N A_j \varphi(x_i - x_j^{\mathrm{s}})\right]}{\sigma_i^2} (-A_j)\varphi(x_i - x_j^{\mathrm{s}}) \cdot (x_i - x_j^{\mathrm{s}}) \\
&= -\sum_i \frac{2A_j(x_i - x_j^{\mathrm{s}})\varphi(x_i - x_j^{\mathrm{s}})}{\sigma_i^2} \left[y_i - \sum_{j=1}^N A_j \varphi(x_i - x_j^{\mathrm{s}})\right] \qquad (2.2.6) \\
\frac{\partial^2 \chi^2(\boldsymbol{A}, \boldsymbol{x}^{\mathrm{s}})}{\partial (x_j^{\mathrm{s}})^2} &= \sum_i \frac{2A_j}{\sigma_i^2} \left\{ \left[y_i - \sum_{j=1}^N A_j \varphi(x_i - x_j^{\mathrm{s}})\right] \varphi(x_i - x_j^{\mathrm{s}}) \left[(x_i - x_j^{\mathrm{s}})^2 - 1\right] - A_j (x_i - x_j^{\mathrm{s}})^2 \varphi^2(x_i - x_j^{\mathrm{s}}) \right\} \qquad (2.2.7)
\end{aligned}$$

The 2nd derivative with respect to x_j^{s} can change sign, implying that $\chi^2(\boldsymbol{A}, \boldsymbol{x}^{\mathrm{s}})$ is not a convex function overall.

Star Index j	Optimized Amplitudes A_j	Optimized Positions x_j^{s}
1	1.55965622	-25.81708898
2	1.23552533	-4.94174051
3	1.31512993	-2.58238806
4	1.07360659	6.13229631
5	1.74916514	14.25189103

Table 3: Optimized amplitudes and positions for each star.

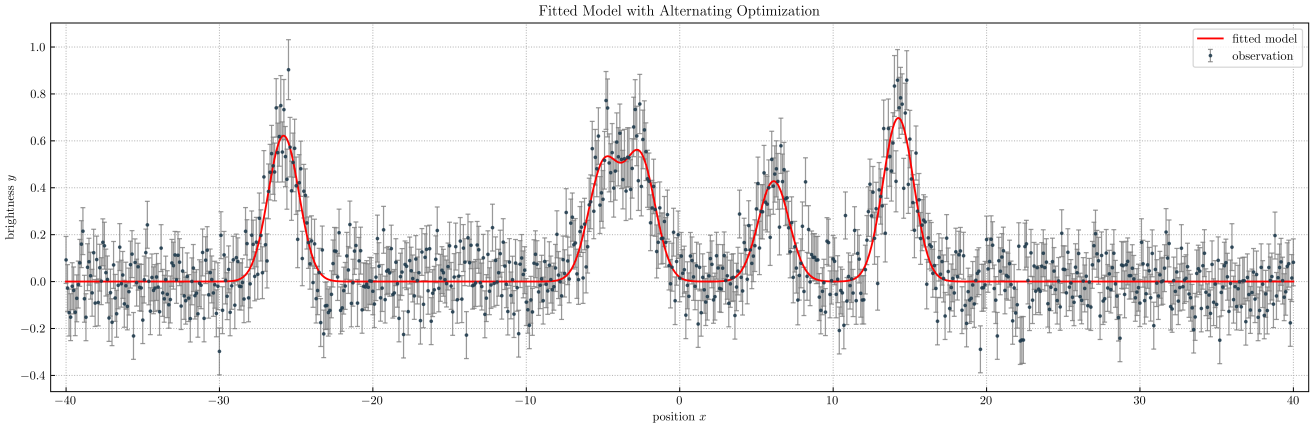

Figure 4: Fitted Model with Alternating Optimization

2.2.3

Write a computer program to obtain these parameters ($2N$ in total) and overplot the result.

Solution: The parameters A_j and x_j^{s} play completely different roles. Specifically, given fixed x_j^{s}, the $\chi^2(\boldsymbol{A}, \boldsymbol{x}^{\mathrm{s}})$ is a quadratic function of the amplitudes A_j. Thus, we can design an iterative optimization algorithm where we alternate between optimizing A_j and x_j^{s}. Given x_j^{s}, we can write the model as:

$$y_i = \sum_{j=1}^{N} A_j \varphi(x_i - x_{s,j}) \tag{2.2.8}$$

We can solve for A_j using weighted linear least squares, accounting for the measurement errors σ_i. With A_j fixed, we minimize $\chi^2(\boldsymbol{A}, \boldsymbol{x}^{\mathrm{s}})$ with respect to x_j^{s} using a non-linear optimization method. The $2N$ optimized parameters are shown in Table 3 and the fitted curve is shown in Figure 4.