Scientific Computing

Solving the nonlinear Poisson equation

Matthew Hennessy

Here we discuss how to solve nonlinear boundary value problems using a "homemade" Newton's method. We will consider the nonlinear Poisson equation in 1D with Dirichlet boundary conditions:

$$\frac{\mathrm{d}^2 u}{\mathrm{d}x^2} + q(u) = 0, \tag{1a}$$

$$u(a) = \alpha, \tag{1b}$$

$$u(b) = \beta. \tag{1c}$$

After discretising this problem using the finite difference method (see online video for details), a nonlinear system of algebraic equations is obtained:

$$\frac{u_2 - 2u_1 + \alpha}{(\Delta x)^2} + q(u_1) = 0, \tag{2a}$$

$$\frac{u_{i+1} - 2u_i + u_{i-1}}{(\Delta x)^2} + q(u_i) = 0, \quad i = 2, 3, \dots, N-2, \tag{2b}$$

$$\frac{\beta - 2u_{N-1} + u_{N-2}}{(\Delta x)^2} + q(u_{N-1}) = 0 \tag{2c}$$

The goal is to solve this algebraic system with Newton's method.

We should first recall that Newton's method requires the algebraic system to be written as a matrix-vector equation of the form $\boldsymbol{F}(\boldsymbol{u}) = \boldsymbol{0}$, where $\boldsymbol{F} = (F_1, \dots, F_{N-1})^T$ and $\boldsymbol{u} = (u_1, \dots, u_{N-1})^T$. In addition, we will need to compute the Jacobian of $\boldsymbol{F}$, which we denote by $\mathbf{J}_{\boldsymbol{F}}$. By multiplying each equation in (2) by $(\Delta x)^2$, we see that the components of $\boldsymbol{F}$ can be defined as

$$F_1 = u_2 - 2u_1 + \alpha + (\Delta x)^2 q(u_1), \tag{3a}$$

$$F_i = u_{i+1} - 2u_i + u_{i-1} + (\Delta x)^2 q(u_i), \quad i = 2, 3, \dots, N-2, \tag{3b}$$

$$F_{N-1} = \beta - 2u_{N-1} + u_{N-2} + (\Delta x)^2 q(u_{N-1}). \tag{3c}$$

The components of $\boldsymbol{F}$ defined in (3) can be written in vector form as

$$\boldsymbol{F}(\boldsymbol{u}) = \mathbf{A}^{DD}\boldsymbol{u} + \boldsymbol{b}^{DD} + (\Delta x)^2 \boldsymbol{q}(\boldsymbol{u}), \tag{4}$$

where $\mathbf{A}^{DD}$ is the tridiagonal matrix associated with two Dirichlet boundary conditions and $\boldsymbol{b}^{DD}$ is the corresponding vector of boundary conditions (see video for details). The vector $\boldsymbol{q}$ represents the discretised source term and it can be written as

$$\boldsymbol{q}(\boldsymbol{u}) = \begin{pmatrix} q(u_1) \\ q(u_2) \\ \vdots \\ q(u_{N-2}) \\ q(u_{N-1}) \end{pmatrix}. \tag{5}$$

The benefit of writing the discretised problem (3) in terms of the matrix-vector system (4) is that it is straightforward to compute its Jacobian matrix. Indeed, we find that

$$\mathbf{J}_{\boldsymbol{F}}(\boldsymbol{u}) = \mathbf{A}^{DD} + (\Delta x)^2 \mathbf{J}_{\boldsymbol{q}}(\boldsymbol{u}), \tag{6}$$

where $\mathbf{J}_{\boldsymbol{q}}$ is the Jacobian of the discrete source term given by (5). Since each component of $\boldsymbol{q}$ only depends on a single component of the solution vector $\boldsymbol{u}$, the Jacobian of $\boldsymbol{q}$ is a diagonal matrix given by

$$\mathbf{J}_{\boldsymbol{q}}(\boldsymbol{u}) = \begin{pmatrix} q'(u_1) & & & \\ & q'(u_2) & & \\ & & \ddots & \\ & & & q'(u_{N-1}) \end{pmatrix}. \tag{7}$$

We now have all of the ingredients to apply Newton's method.

Recall that for a vector system, Newton's method is given by

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}^n - (\mathbf{J}_{\boldsymbol{F}}(\boldsymbol{u}^n))^{-1}\boldsymbol{F}(\boldsymbol{u}^n), \tag{8}$$

where $\boldsymbol{u}^n$ represents the approximation to the solution $\boldsymbol{u}$ after n iterations. This version of Newton's method is computationally expensive due to the need to compute the inverse of the Jacobian matrix. Thus, we reformulate Newton's method to avoid matrix inversion by dividing it into two steps: first, we solve a linear system of equations to obtain the correction $\boldsymbol{V}$ to the current solution $\boldsymbol{u}^n$. Then, we compute a new solution $\boldsymbol{u}^{n+1}$ by adding $\boldsymbol{V}$ to $\boldsymbol{u}^n$. Mathematically, this form of Newton's method can be written as

$$\mathbf{J}_{\boldsymbol{F}}(\boldsymbol{u}^n)\boldsymbol{V} = -\boldsymbol{F}(\boldsymbol{u}^n), \tag{9a}$$

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}^n + \boldsymbol{V}. \tag{9b}$$

The linear system in (9a) can be solved using NumPy's `linalg.solve` function. In fact, for this specific problem, the Jacobian $\mathbf{J}_{\boldsymbol{F}}$ given by (6) still has a tridiagonal structure so the Thomas algorithm can be used.

One final comment is that Newton's method can sometimes fail to find a solution; that is, the solution approximations $\boldsymbol{u}^n$ fail to converge. This can happen if the initial guess of the solution is poor. For boundary value problems, this can be particularly problematic because one might not have a good idea of what the solution to the differential equation should look like before attempting to compute it numerically. To remedy this situation, a damping factor $0 < \nu \leq 1$ can be introduced to prevent the numerical solution from changing too rapidly between Newton iterations. The damped version of Newton's method is given by

$$\mathbf{J}_{\boldsymbol{F}}(\boldsymbol{u}^n)\boldsymbol{V} = -\boldsymbol{F}(\boldsymbol{u}^n), \tag{10a}$$

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}^n + \nu\boldsymbol{V}. \tag{10b}$$

This version of the method is more stable but its rate of convergence is slower; hence, more iterations are required to find a solution.