Solving Partial Differential Equations Using Convex Optimization

Nav Ravindranath
Columbia University
Applied Physics and Applied Mathematics
New York, NY
Email: ngr2114@columbia.edu

***Abstract*—We present a method for solving partial differential equations (PDEs) using standard convex optimization tools, such as CVX [1], [2]. The proposed method provides a declarative syntax for specifying the PDE, thereby greatly simplifying the handling of boundary conditions. As a result, researchers implementing numerical methods not supported by existing solvers can quickly explore the effectiveness of a large number of numerical schemes on their problem before investing the time to implement the schemes using more traditional approaches, which may be required for computational efficiency reasons after increasing the resolution of the simulation. In addition to demonstrating how PDEs can be expressed as convex optimization problems, we explore the scalability limits of convex solvers in the context of PDEs.**

I. Introduction

When modeling complicated physical phenomena such as atmospheric processes or tropical storms, the problem is often defined as a system of PDEs which need to be solved numerically. Unfortunately, standard off-the-shelf solvers for PDEs are usually inadequate for solving such problems. It is common for researchers to develop their own numerical PDE solvers which exploit special structure unique to their problem [3]–[6].

When developing a new PDE solver, there are several important aspects which need to be considered. First is the consistency of the numerical method, which is usually determined analytically. Next is the stability and convergence of the method. While theoretical guarantees can be given for many numerical methods regarding rates of convergence, the rates can sometimes be misleading in practice due to the sizes of constants on leading error terms, which can be problem dependent [7]. Fortunately, the rate at which the accuracy of a simulation improves as the resolution of the simulation is increased can be observed for a specific problem experimentally. However, in order to do so, the numerical method first needs to be correctly implemented.

One of the trickiest aspects of implementing numerical solvers for PDEs is correctly incorporating boundary conditions into the numerical scheme [8]. For illustration, we start by considering a simple example – a finite difference method for Poisson's equation in 2 dimensions. Poisson's equation takes the form

$$\nabla^2 u = f$$

where $\nabla^2 u = u_{xx} + u_{yy}$ in 2 dimensions. In order for the problem to be well posed, we also need conditions to be specified at the boundaries of the domain. For simplicity, we consider the Dirichlet boundary conditions

$$\begin{aligned} u(0,y) &= g_0(y) \\ u(M,y) &= g_1(y) \\ u(x,0) &= h_0(x) \\ u(x,N) &= h_1(x) \end{aligned}$$

on a rectangular domain of size $M \times N$.

Suppose we discretize the domain using m interior points in the x-direction and n interior points in the y-direction, uniformly spaced in each direction, so that we have a mesh of size $(m+2)\times(n+2)$ consisting of the points $\{(x_i, y_j)\}$ where $i \in \{0,\ldots,m+1\}, j \in \{0,\ldots,n+1\}$ with the boundaries included. The discretized solution U consists of the values of the continuous solution u on the grid points of the mesh. We similarly discretize the source term f as an $(m+2)\times(n+2)$ matrix F. It remains to discretize the Laplacian operator ∇^2.

A common discrete approximation to the Laplacian is the 5-point discrete Laplacian operator given by [9], [10]

$$\nabla_5^2 U_{i,j} = \frac{U_{i-1,j} - 2U_{i,j} + U_{i+1,j}}{\Delta x^2} + \frac{U_{i,j-1} - 2U_{i,j} + U_{i,j+1}}{\Delta y^2}$$

where $U_{i,j} = u(x_i, y_j), i \in \{0,\ldots,m+1\}, j \in \{0,\ldots,n+1\}$, and the mesh sizes in each direction are $\Delta x = (m+1)/M$, and $\Delta y = (n+1)/N$. The 5-point discrete Laplacian operator is a second-order centered approximation to its continuous analogue.

As a step toward expressing the discretized equation as a convex optimization problem, we would like to write the discretized equation in matrix form. To do so, we construct the unrolled vectors

$$\vec{U} = \begin{bmatrix} U_{0,0} \\ \vdots \\ U_{m+1,0} \\ \vdots \\ U_{0,n+1} \\ \vdots \\ U_{m+1,n+1} \end{bmatrix}, \quad \vec{F} = \begin{bmatrix} F_{0,0} \\ \vdots \\ F_{m+1,0} \\ \vdots \\ F_{0,n+1} \\ \vdots \\ F_{m+1,n+1} \end{bmatrix}$$

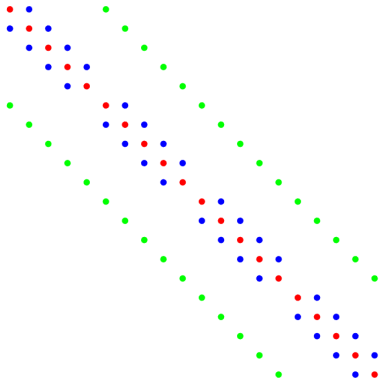

Fig. 1. Second-order centered discrete Laplacian matrix

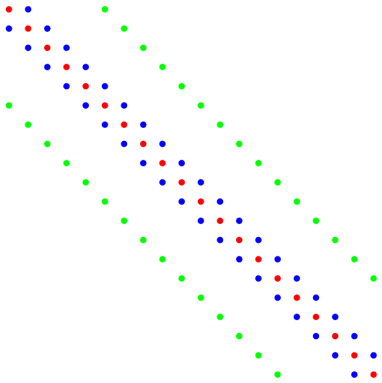

Fig. 2. Modified second-order centered discrete Laplacian matrix

with $F_{i,j}$ defined analogously to $U_{i,j}$. We can now express the discrete Laplacian ∇_5^2 as a matrix of the form shown in Figure 1 where the entries on the main diagonal (indicated in red) take the value $-\frac{2}{\Delta x^2} - \frac{2}{\Delta y^2}$, the entries on the subdiagonal and superdiagonal (indicated in blue) take the value $\frac{1}{\Delta x^2}$, and the entries on the outer diagonals (indicated in green) take the value $\frac{1}{\Delta y^2}$.

Note that the off-diagonals have missing entries corresponding to the boundary points. Traditional methods of implementing numerical PDE solvers require adjusting for the missing entries either by using ghost cells or by modifying the discretized source term F [10]. As the PDE and choice of discretization become more complicated, correctly incorporating boundary conditions becomes increasingly cumbersome and significantly hinders the ability to quickly explore the effectiveness of hand-crafted numerical schemes for solving PDEs.

II. Problem Formulation

A. Poisson's Equation

The discrete version of Poisson's equation on the interior of the domain can now be expressed as

$$(A_* \vec{U})^\circ = (\vec{F})^\circ$$

where A_* is the discrete Laplacian matrix shown in Figure 1, and $^\circ : \mathbb{R}^{(m+2)(n+2)} \to \mathbb{R}^{mn}$ is the operator defined by

$$\begin{bmatrix} V_{0,0} \\ \vdots \\ V_{m+1,0} \\ \vdots \\ V_{0,n+1} \\ \vdots \\ V_{m+1,n+1} \end{bmatrix}^\circ = \begin{bmatrix} V_{1,1} \\ \vdots \\ V_{m,1} \\ \vdots \\ V_{1,n} \\ \vdots \\ V_{m,n} \end{bmatrix}$$

that extracts the entries on the interior of the mesh. The Dirichlet boundary conditions can be easily imposed on the discretized solution as

$$\begin{aligned} U_{0,j} &= g_0(y_j) \\ U_{m+1,j} &= g_1(y_j) \\ U_{i,0} &= h_0(x_i) \\ U_{i,n+1} &= h_1(x_i) \end{aligned}$$

for $i \in \{0, \dots, m+1\}$ and $j \in \{0, \dots, n+1\}$.

Observe that the rows of A_* with missing entries do not affect the result of $(A_* \vec{U})^\circ$. Therefore, we can simplify the structure of the discrete Laplacian matrix by filling in the missing entries as shown in Figure 2. In other words, if we let A denote the matrix shown in Figure 2, then we have

$$(A_* \vec{U})^\circ = (A\vec{U})^\circ$$

so that numerically solving for U is equivalent to solving the convex optimization problem

$$\begin{array}{lll} \text{minimize} & \|(A\vec{U} - \vec{F})^\circ\|_2 & \\ \text{subject to} & U_{0,j} = g_0(y_j), & j = 0, \dots, n+1 \\ & U_{m+1,j} = g_1(y_j), & j = 0, \dots, n+1 \\ & U_{i,0} = h_0(x_i), & i = 0, \dots, m+1 \\ & U_{i,n+1} = h_1(x_i), & i = 0, \dots, m+1 \end{array}$$

where we minimize the norm of the difference between the left-hand and right-hand sides of the PDE, and impose the boundary conditions as constraints. The convex optimization problem is equivalent to a linearly constrained least squares problem upon squaring the objective function, since $^\circ$ is a linear operator.

B. Heat Equation

We now turn to a slightly more complicated PDE—the 2-dimensional heat equation—which adds an element of time to Poisson's equation. We consider the PDE

$$\nabla^2 u = u_t$$

with boundary conditions

$$\begin{aligned} u(0,y,t) &= g_0(y,t) \\ u(M,y,t) &= g_1(y,t) \\ u(x,0,t) &= h_0(x,t) \\ u(x,N,t) &= h_1(x,t) \end{aligned}$$

and initial condition

$$u(x,y,0) = u_0(x,y)$$

where $0 \leq t \leq T$. Furthermore, we discretize time into $p+1$ equispaced points between 0 and T (inclusive) with time step $\Delta t = T/p$. The discrete solution U is now a 3-dimensional array over an $(m+2) \times (n+2) \times (p+1)$ grid with the value corresponding to $u(x_i, y_j, t_k)$ denoted by $U^k_{i,j}$ for each $i \in \{0,\ldots,m+1\}, j \in \{0,\ldots,n+1\}, k \in \{0,\ldots,p\}$.

In order to convert the heat equation into matrix form, we define

$$\vec{U}^k = \begin{bmatrix} U^k_{0,0} \\ \vdots \\ U^k_{m+1,0} \\ \vdots \\ U^k_{0,n+1} \\ \vdots \\ U^k_{m+1,n+1} \end{bmatrix}$$

and

$$\widetilde{U}_+ = \begin{bmatrix} \vec{U}^1 & \cdots & \vec{U}^p \end{bmatrix}, \quad \widetilde{U}_- = \begin{bmatrix} \vec{U}^0 & \cdots & \vec{U}^{p-1} \end{bmatrix}$$

so that the discretized heat equation becomes

$$\left(\frac{A(\widetilde{U}_+ + \widetilde{U}_-)}{2} \right)^\circ = \left(\frac{\widetilde{U}_+ - \widetilde{U}_-}{\Delta t} \right)^\circ$$

using the Crank-Nicolson method [11], where the $^\circ$ operator is applied column-wise. Note that applying $^\circ$ is equivalent to left multiplication by the identity matrix with rows corresponding to boundary points removed. We can then express the heat equation as the convex optimization problem

$$\begin{aligned} \text{minimize} \quad & \left\| \left(\tfrac{A(\widetilde{U}_+ + \widetilde{U}_-)}{2} - \tfrac{\widetilde{U}_+ - \widetilde{U}_-}{\Delta t} \right)^\circ \right\|_F \\ \text{subject to} \quad & U^k_{0,j} = g_0(y_j, t_k) \\ & U^k_{m+1,j} = g_1(y_j, t_k) \\ & U^k_{i,0} = h_0(x_i, t_k) \\ & U^k_{i,n+1} = h_1(x_i, t_k) \\ & U^0_{i,j} = u_0(x_i, y_j) \end{aligned}$$

with the boundary conditions and initial condition as constraints, where i, j, and k range over their possible values, and $\|\cdot\|_F$ denotes the Frobenius norm. Since the Frobenius norm can be expressed as a Euclidean vector norm, this problem is also equivalent to a linearly constrained least squares problem.

III. Solutions

Because the convex optimization problems described above will be solved by CVX using a conic solver, we minimize the norm of the residual rather than the squared norm [12]. Specifying the convex optimization problem for Poisson's equation in CVX is straightforward; we construct the mesh from uniformly spaced vectors of grid points, evaluate the source term f on the mesh, build the diagonal matrix A, and then specify the objective function and constraints almost exactly as stated in mathematical form.

Table I shows how each of the mathematical operators in the objective function for Poisson's equation can be implemented in MATLAB® with CVX. In Table I, X is an $(m+2)\times(n+2)$ matrix, $\vec{X}$ is the corresponding unrolled vector, and `X` is an $(m+2) \times (n+2)$ MATLAB® array; $\vec{Y}$ is an mn-vector, and `Y` is the corresponding $mn \times 1$ MATLAB® array.

TABLE I
Implementation of operators in MATLAB® with CVX for Poisson's equation

Mathematics	**MATLAB®**
$\vec{X}$	`X(:)`
$(\vec{X})^\circ$	`reshape(X(2:end-1,2:end-1), [], 1)`
$\|\vec{Y}\|_2$	`norm(Y)`

The solution to Poisson's equation computed by CVX on a 40×25 grid with source function

$$f(x,y) = -2 \sin x \cos y$$

and boundary conditions

$$\begin{aligned} u(0,y) &= u(3\pi,y) = 0 \\ u(x,0) &= u(x,2\pi) = \sin x \end{aligned}$$

is indicated in Figure 3. The computed solution satisfies the

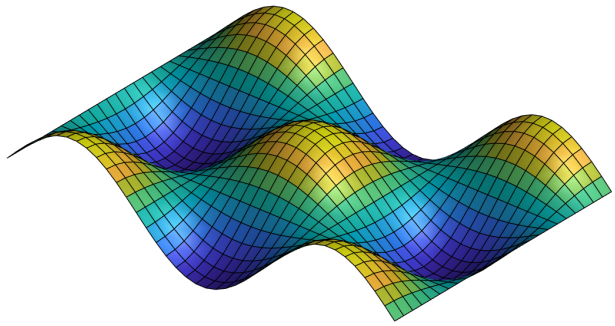

Fig. 3. Solution to Poisson's equation using CVX

boundary conditions and matches the analytical solution

$$u(x, y) = \sin x \cos y$$

as can be verified visually.

Solving the heat equation using CVX follows a similar procedure, except that we must work with 3-dimensional arrays rather than 2-dimensional ones, and additionally specify the initial condition as a constraint. Table II shows how the necessary operations can be performed in MATLAB®. In the table below, `U` is an $(m+2) \times (n+2) \times (p+1)$ MATLAB® array corresponding to the discrete solution U, and V and W are $(m+2)(n+2) \times p$ and $mn \times p$ matrices, respectively, with `V` and `W` as their MATLAB® counterparts.

TABLE II
IMPLEMENTATION OF OPERATORS IN MATLAB® WITH CVX FOR HEAT EQUATION

Mathematics	MATLAB®
$\widetilde{U}_+$	`reshape(U(:,:,2:end), [], p)`
$\widetilde{U}_-$	`reshape(U(:,:,1:end-1), [], p)`
V°	`tmp = reshape(V, m+2, n+2, p);` `reshape(tmp(2:end-1,2:end-1,:), [], p)`
$\|W\|_F$	`norm(W(:))`

Figure 4 shows snapshots of the solution computed by CVX on a $40 \times 25 \times 17$ grid with $T = 1.6$, boundary conditions

$$u(0, y, t) = u(3\pi, y, t) = 0$$
$$u(x, 0, t) = u(x, 2\pi, t) = \sin x$$

and initial condition

$$u_0(x, y) = \sin x \cos y$$

which was the solution to Poisson's equation from before on the same spatial domain. As expected, the initial heat distribution diffuses away while maintaining the boundary conditions [11].

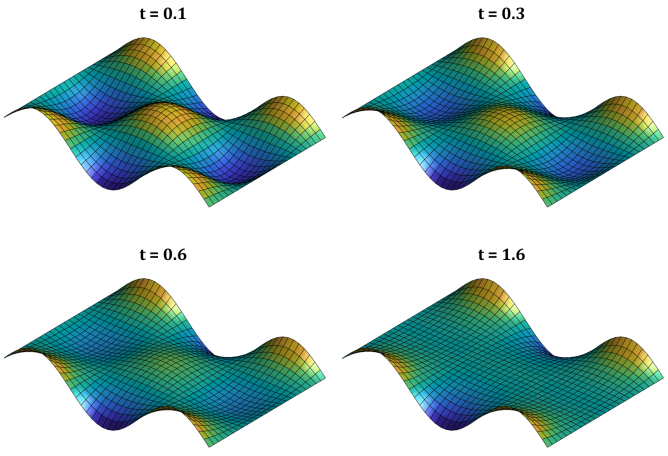

Fig. 4. Solution to heat equation using CVX

IV. Analysis

We now examine how the computational time required for CVX to solve the PDEs scales as the mesh is made finer. First, we increasingly refine the spatial grid by a factor of 2 in each dimension and observe the effect on the time required to solve Poisson's equation using CVX. Figure 5 indicates that

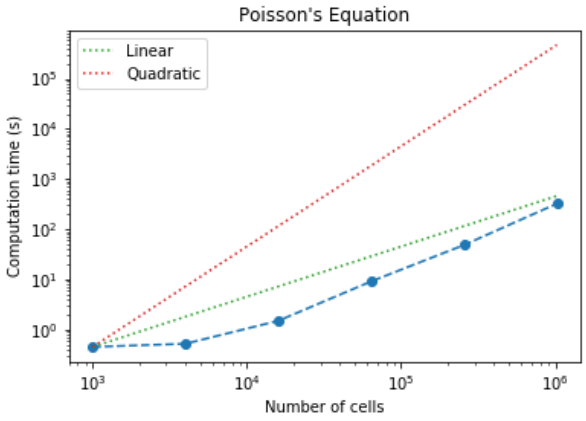

Fig. 5. Scaling of computation time with respect to spatial refinement

the computation time grows faster than linearly but slower than quadratically with the number of grid points. The finest mesh consisted of 1,024,000 grid points and took 322.97 seconds to solve with an Intel® Core™ i7-6820HQ processor at 2.70 GHz and 32 GB RAM.

Next, we consider how the time required to solve the heat equation changes as we increase the number of time steps while keeping the initial and final times fixed. Figure 6 shows

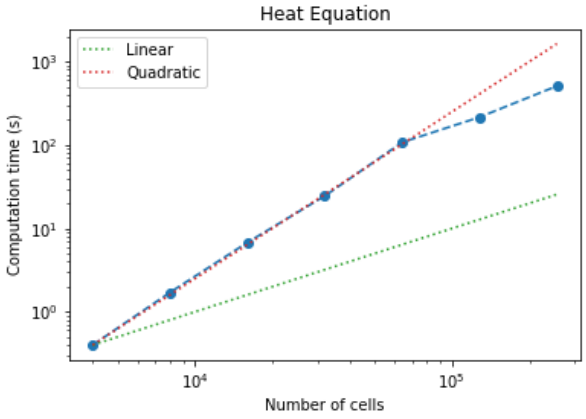

Fig. 6. Scaling of computation time with respect to temporal refinement

that the computation time grows quadratically with the number of grid points. For the finest mesh, we used 256,000 grid points, which required 509.70 seconds to solve.

V. Conclusion

Expressing a PDE as a convex optimization problem is an effective way to test how a numerical method performs on the PDE without having to explicitly implement the intricacies of the numerical scheme. We've seen how convex optimization provides a powerful declarative syntax to specify a discretization of a PDE along with its boundary and initial conditions without having to use techniques such as ghost cells or source term modification to embed boundary conditions into the discretization. In some cases, we can even additionally simplify the problem, for example, by conveniently filling in missing entries in the discrete Laplacian.

Furthermore, we observed experimentally that the time required to solve PDEs using CVX only grows quadratically with the number of grid points, enabling convergence studies to be performed as the mesh is refined for moderately sized problems. For larger problems, expressing a PDE as a convex optimization problem provides the potential to leverage existing distributed convex solvers without having to write custom distributed PDE solvers, as they can be error-prone and time-consuming to implement.

Finally, having a declarative syntax to specify a numerical method for solving PDEs reduces the chance of introducing bugs in the solver and provides a reference implementation for comparison if a more efficient custom solver is later written.

Notes

The MATLAB® code used to solve Poisson's equation and the heat equation, along with the code to generate the figures in this paper, is available at https://github.com/navravi/eeor4650-project.

References

[1] M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming, version 2.1," http://cvxr.com/cvx, Mar. 2014.

[2] ——, "Graph implementations for nonsmooth convex programs," in *Recent Advances in Learning and Control*, ser. Lecture Notes in Control and Information Sciences, V. Blondel, S. Boyd, and H. Kimura, Eds. Springer-Verlag Limited, 2008, pp. 95–110, http://stanford.edu/~boyd/graph_dcp.html.

[3] M. J. Berger and J. Oliger, "Adaptive mesh refinement for hyperbolic partial differential equations," *J. Comp. Phys.*, vol. 53, no. 3, pp. 484–512, 3 1984.

[4] M. J. Berger, D. L. George, R. J. LeVeque, and K. T. Mandli, "The GeoClaw software for depth-averaged flows with adaptive refinement," *Advances in Water Resources*, vol. 34, pp. 1195–1206, 2011.

[5] K. T. Mandli and C. N. Dawson, "Adaptive mesh refinement for storm surge," *Ocean Modelling*, vol. 75, pp. 36–50, 2014.

[6] K. T. Mandli, A. J. Ahmadia, M. J. Berger, D. Calhoun, D. L. George, Y. Hadjimichael, D. I. Ketcheson, G. I. Lemoine, and R. J. LeVeque, "Clawpack: building an open source ecosystem for solving hyperbolic pdes," *PeerJ Comput. Sci. 2:e68; DOI 10.7717/peerj-cs.68*, 2016.

[7] M. Iskandarani, "Finite difference approximation of derivatives," in *Course Notes for MSC321*. University of Miami, ch. 15, http://yyy.rsmas.miami.edu/users/miskandarani/Courses/MSC321/lectfiniteDifference.pdf.

[8] K. T. Mandli, "Mixed equations," https://github.com/mandli/numerical-methods-pdes, 2018, GitHub repository.

[9] A. I. van de Vooren and V. C. Vliegenthart, "On the 9-point difference formula for laplace's equation," *J. Eng. Math.*, no. 1, pp. 187–202, 1967.

[10] R. J. LeVeque, *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*. Society for Industrial and Applied Mathematics, 2007.

[11] N. H. Sweilam, M. M. Khader, and A. M. S. Mahdy, "Crank-Nicolson finite difference method for solving time-fractional diffusion equation," *Journal of Fractional Calculus and Applications*, vol. 2, no. 2, pp. 1–9, Jan. 2012.

[12] M. Grant and S. Boyd, "Advanced topics," in *The CVX Users' Guide, release 2.1*. CVX Research, Inc., Dec. 2017, ch. 11, pp. 67–76.