High-Order Finite-Difference Discretization for Steady-State Convection-Diffusion Equation on Arbitrary Domain *

Abdulaziz Albaiz (baiz@mit.edu)

May 11, 2014

Abstract

In this project, a high-order discretization technique for stationary convection-diffusion equation is introduced and implemented in different 1D, 2D, and 3D applications. This technique uses finite-difference method to approximate first- and second-order derivatives for internal points, but utilizes local extrapolation functions for points that are close to the domain boundaries. This is particularly useful for problems with arbitrary domains and irregular boundaries. The generated matrix is symmetric when constant or linear extrapolation is used, and is non-symmetric when higher-order extrapolation, such as quadratic or cubic, is used. In addition to using Matlab's backslash operator to solve the system, GMRES is implemented as a Matlab function and is also used for solving the system. We show that this discretization technique can achieve fourth-order accuracy when fourth-order FD approximation is used with cubic interpolation functions.

*This project is mostly based on: Gibou, Frédéric, and Ronald Fedkiw. "A fourth order accurate discretization for the Laplace and heat equations on arbitrary domains, with applications to the Stefan problem." *Journal of Computational Physics* 202.2 (2005): 577-601.

1 Finite-Difference Discretization of Convection-Diffusion Equation

1.1 Steady-State Convection-Diffusion Equation

The 1D convection-diffusion equation is given as

$$\frac{\partial u}{\partial t} = V\frac{\partial u}{\partial x} + D\frac{\partial^2 u}{\partial x^2} + f$$

In general, the convection-diffusion equation can be written as

$$\frac{\partial u}{\partial t} = V\nabla u + D\nabla^2 u + f$$

We will only deal with steady-state convection-diffusion equation, where the time-dependent term $\frac{\partial u}{\partial t} = 0$. Therefore, the general-form equation becomes

$$V\nabla u + D\nabla^2 u = -f$$

Many PDEs, such as Poisson's equation, $\nabla^2 u = -f$, are special cases of the convection-diffusion equation. In this paper, we will solve these PDEs by discretizing them on arbitrary domains using higher-order approximations, generating the linear system matrix, then solving the system using some solver, such as Matlab's backslash operator.

1.2 High-Order Finite-Difference Approximation

Discretization of the convection-diffusion equation using finite-difference method is performed by using approximations of the first and second derivatives of u. One of the most used FD approximation for second derivate is the second-order central-difference approximation

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_i \approx \frac{u_{i-1} - 2u_i + u_{i+1}}{\Delta x^2}$$

Because we want to achieve higher-order accuracy, better approximations of the derivatives are used. More specifically, fourth-order central-difference approximations of the first and second derivatives are used. These approximations use five-point stencils in 1D, as shown in figure 1. The approximations are

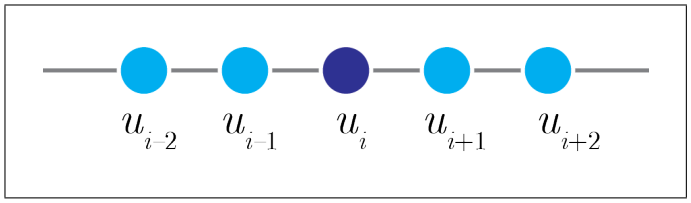

Figure 1: Five-point stencil in 1D

$$\left.\frac{\partial u}{\partial x}\right|_i \approx \frac{\frac{1}{12}u_{i-2} - \frac{2}{3}u_{i-1} - \frac{2}{3}u_{i+1} + \frac{1}{12}u_{i+2}}{\Delta x}$$
$$\left.\frac{\partial^2 u}{\partial x^2}\right|_i \approx \frac{-\frac{1}{12}u_{i-2} + \frac{4}{3}u_{i-1} - \frac{5}{2}u_i + \frac{4}{3}u_{i+1} - \frac{1}{12}u_{i+2}}{\Delta x^2}$$

Hence, the 1D convection-diffusion equation can be approximated as

$$V\frac{\frac{1}{12}u_{i-2} - \frac{2}{3}u_{i-1} - \frac{2}{3}u_{i+1} + \frac{1}{12}u_{i+2}}{\Delta x}$$
$$+D\frac{-\frac{1}{12}u_{i-2} + \frac{4}{3}u_{i-1} - \frac{5}{2}u_i + \frac{4}{3}u_{i+1} - \frac{1}{12}u_{i+2}}{\Delta x^2} = -f_i$$

The ith row of the Matrix A, corresponding to the coefficients that approximate u_i, is computed, and we get the linear system $Au = f$.

A simple way to extend this approximation to 2D and 3D is to keep the approximations of each dimension and derivate separate. For example, the 2D convection-diffusion equation

$$V_x\frac{\partial u}{\partial x} + V_y\frac{\partial u}{\partial y} + D(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) = -f$$

can be approximated by replacing each derivate with its approximation, then the coefficients are computed and added to the matrix A. The approximation of the 2D convection-diffusion equation will result in a 9-point stencil, as shown in figure 2. By keeping the numerical discretization of each derivate independent, it is trivial to extend the procedure to 2D and 3D.

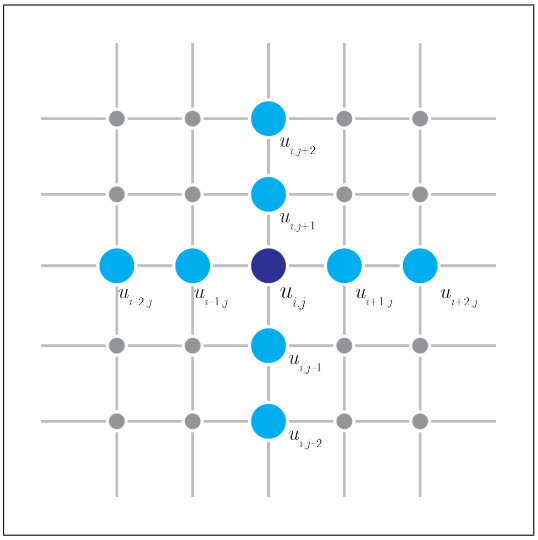

Figure 2: Nine-point stencil in 2D

2 Discretization of Arbitrary Domain Boundaries

The above-described finite-difference method works well with regular rectangular grid, and boundary conditions, whether they are Dirichlet or Neumann or a mix of both, can be easily incorporated in the matrix A and the right-hand side. However, a problem with irregular grid or arbitrary domain cannot be easily represented using this scheme. One approach is to map the irregular domain to a Cartesian rectangular domain, and mapping functions to translate between the original and new domains. This approach, however, only works with certain domains where such mapping functions can be found.

In this section, we will introduce an approach where extrapolation functions are used at the points near the boundaries. This approach works well for problems with arbitrary domains, and does not require domain transformation into a different coordinate system.

For the rest of this paper, we will assume that all the boundaries are Dirichlet boundaries. Neumann and other types of boundaries conditions can be easily incorporated into the matrix in a similar fashion to the traditional finite-difference method.

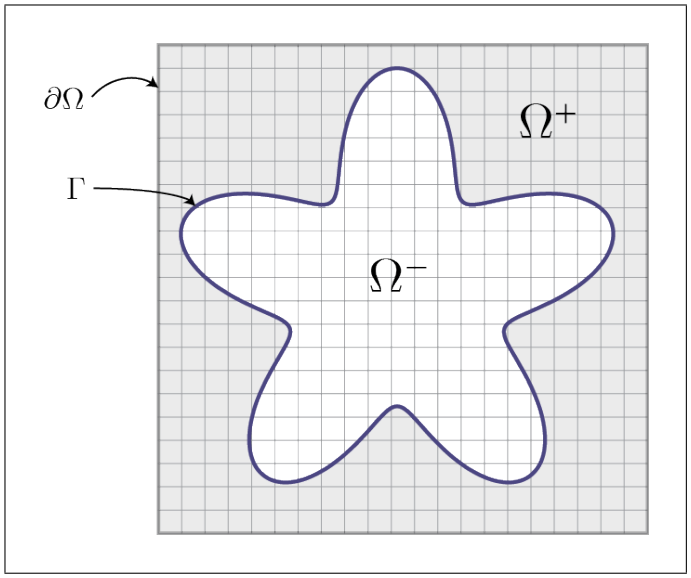

Figure 3: The Cartesian domain Ω is split based on the interface Γ into an internal domain Ω^- and external domain Ω^+

2.1 Domain Interface

For a given steady-state convection-diffusion problem with arbitrary domain, we will define the Cartesian domain as $\Omega \subset R^n$. The boundary $\partial\Omega$ defines the exterior boundary of Ω, where as the problem's arbitrary domain, also called the interface, is defined as Γ. We split Ω into two disjoint sets, Ω^-, which is the part inside the interface Γ, and Ω^+ that is the rest of the domain which lies outside Γ. Figure

The interface Γ intersects with the grid at arbitrary points that are not necessarily discretization points. In the case of 1D, the interface is simply two points at the beginning and the end of the interior boundary. Although the 1D case does not really represent an arbitrary domain, it is used to demonstrate how the arbitrary boundary is treated in this technique, as the 2D and 3D cases will be very similar.

Figure 4 shows the last few points in the internal domain Ω^-, followed by the interface point $u_I \in \Gamma$, then the exterior domain Ω^+. Because the point u_i is near the end of the internal domain, the FD approximation of its derivatives require the values of the two points that are in the external domain Ω^+. However, these points are not part of the problem's domain

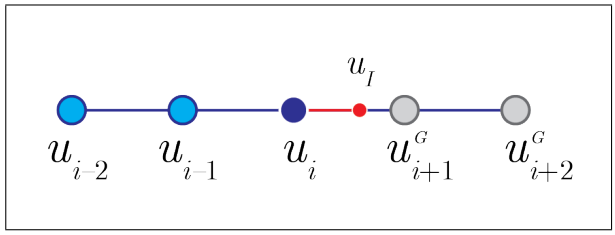

Figure 4: At the boundary, interface point u_I is used for extrapolating the ghost points u^G_{i+1} and u^G_{i+2}

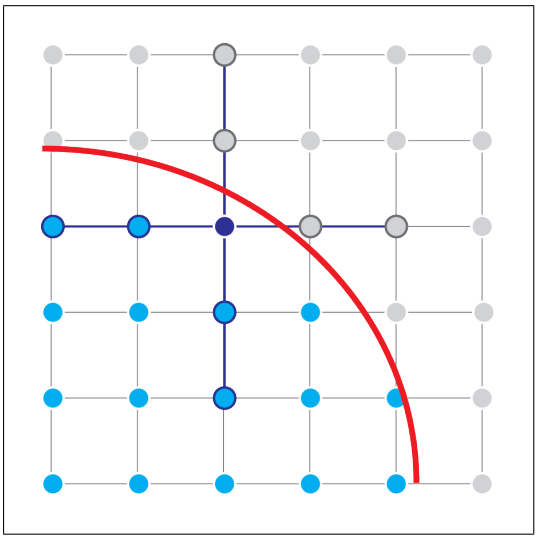

Figure 5: In 2D, the ghost points outside the boundary are extrapolated in each dimension independently

and therefore their solution values are not known. We will call these points 'ghost' points, and denote them as u^G_{i+1} and u^G_{i+2}.

This can be extended to 2D and 3D, where the idea of arbitrary domain makes more sense than the 1D case. Figure 5 shows a subset of the domain with an interface curve that splits the domain into internal domain Ω^- and external domain Ω^+. The approximation of the marked point $u_{i,j}$ depends on 2 ghost points in the x-direction, and 2 ghost points in the y-direction. As the approximation of derivatives is done independently for each axis, the technique can be extended easily from 1D to 2D and 3D.

Note that the interface Γ does not necessarily intersect with the grid at discretization points $u_{i,j}$, and therefore there is no restriction on the problem's

arbitrary domain.

2.2 Local Extrapolation Functions

The point u_i in the 1D case, shown in figure 4 has the following FD approximations

$$\left.\frac{\partial u}{\partial x}\right|_i \approx \frac{\frac{1}{12}u_{i-2} - \frac{2}{3}u_{i-1} - \frac{2}{3}u^G_{i+1} + \frac{1}{12}u^G_{i+2}}{\Delta x}$$
$$\left.\frac{\partial^2 u}{\partial x^2}\right|_i \approx \frac{-\frac{1}{12}u_{i-2} + \frac{4}{3}u_{i-1} - \frac{5}{2}u_i + \frac{4}{3}u^G_{i+1} - \frac{1}{12}u^G_{i+2}}{\Delta x^2}$$

where the points u^G_{i+1} and u^G_{i+2} are ghost points that are not part of the solution vector u, and therefore, their values are unknown. One way of resolving this issue is to include the set of all ghost points into the solution. This, however, will result in a larger and more complicated system of equations, and is not practical as these ghost points also require additional points that are not part of the solution.

Instead of trying to find the numerical solutions at the ghost points, extrapolation functions are used to approximate the values at these points. The known points, namely u_I, u_i, u_{i-1}, etc., are used to create approximation functions that are then evaluated at the ghost points, and these values are used in the FD approximation.

Assuming $u(x)$ is the solution function of the 1D problem, it is only valid for $x \in \Omega^-$. The value u_I at the interface point is known since we are using Dirichlet boundary conditions. Let $\tilde{u}(x)$ be the extrapolation function that will be used for the external domain, i.e. $x \in \Omega^+$. We will assume that the extrapolation functions is shifted such that $\tilde{u}(0)$ lies on the point u_i. The following different types of extrapolation functions $\tilde{u}(x)$ can be used:

- Constant extrapolation: $\tilde{u}(x) = u_I$. This implies that all the point in the external domain have the same value as the interface.
- Linear extrapolation: $\tilde{u}(x) = ax + b$. To find the coefficients a and b, we solve a system the system of equations

$$\tilde{u}(0) = u_i, \quad \tilde{u}(x_i - x_I) = u_I$$

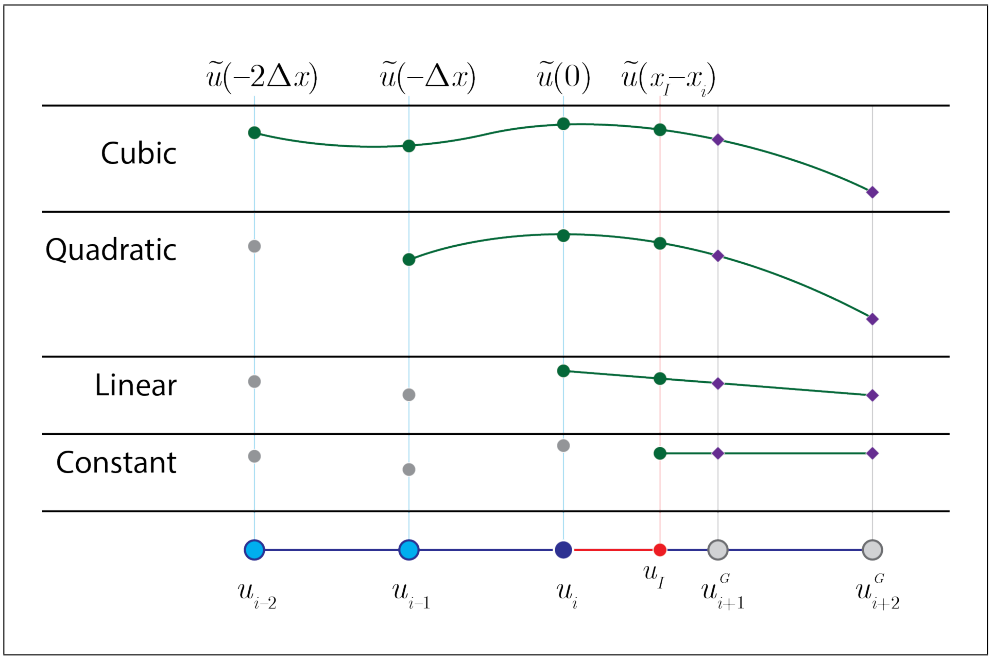

Figure 6: Different extrapolation functions to evaluate the ghost points u^G_{i+1} and u^G_{i+2}

- Quadratic extrapolation: $\tilde{u}(x) = ax^2 + bx + c$. The coefficients a, b and c can be found by solving the following system:

$$\tilde{u}(0) = u_i, \quad \tilde{u}(-\Delta x) = u_{i-1}, \quad \tilde{u}(x_i - x_I) = u_I$$

- Cubic extrapolation: $\tilde{u}(x) = ax^3 + bx^2 + cx + d$. The coefficients are the solution of the system:

$$\tilde{u}(0) = u_i, \quad \tilde{u}(-\Delta x) = u_{i-1}, \quad \tilde{u}(-2\Delta x) = u_{i-2}, \quad \tilde{u}(x_i - x_I) = u_I$$

Figure 6 shows examples for each of the extrapolation functions, and the points that are used to construction these functions. Once the extrapolation function is found, the ghost points are evaluated as $u^G_{i+1} = \tilde{u}(\Delta x)$ and $u^G_{i+2} = \tilde{u}(2\Delta x)$. These values are then used in the FD approximation of the first- and second-derivatives.

It is also worth mentioning that the number of ghost points is not always two. The FD approximation of the point u_{i-1}, for example, has only one

ghost point. However, the same ghost-point values that were used for u_i can be used directly for u_{i-1} without the need to reconstruct the extrapolation function.

The extension of this technique to 2D and 3D is quite trivial, as each dimension is dealt with independently. In the case of 2D, shown in figure 5, the ghost points in the x-direction are evaluated separately using the extrapolation function, then the two y-axis ghost points are evaluated. The values are then used in the FD approximations of the derivatives. The same goes for 3D, as an additional step for evaluating the z-axis ghost point is also performed. The number of ghost points required for each point may vary between zero, one, or two in each dimension, depending on the shape of the interface Γ.

Although this technique works for most arbitrary domains, it is important that the domain is convex. Having non-convex domain may result in having a point u_i surrounded by ghost points in all dimensions. Although the technique can still be used to extrapolate the points using as much data as available, the accuracy of the solution is not necessarily guaranteed to maintain the same order.

3 Application Examples

The described technique is implemented in Matlab with different extrapolation functions. Two examples are described in this section: a simple 1D Poisson equation example, and 2D stationary convection-diffusion problem.

3.1 1D Poisson Equation

The first example is a 1D Poisson equation in the form

$$\frac{\partial^2 u}{\partial x^2} = -f$$

The Cartesian domain is $[0, 1]$, and we will use an arbitrary domain Γ defined by the interval $[0.1813, 0.8824]$. The objective of this example is to verify the correctness of the technique in approximating the solution with different extrapolations around the domain interface. The particular problem that will be solved is

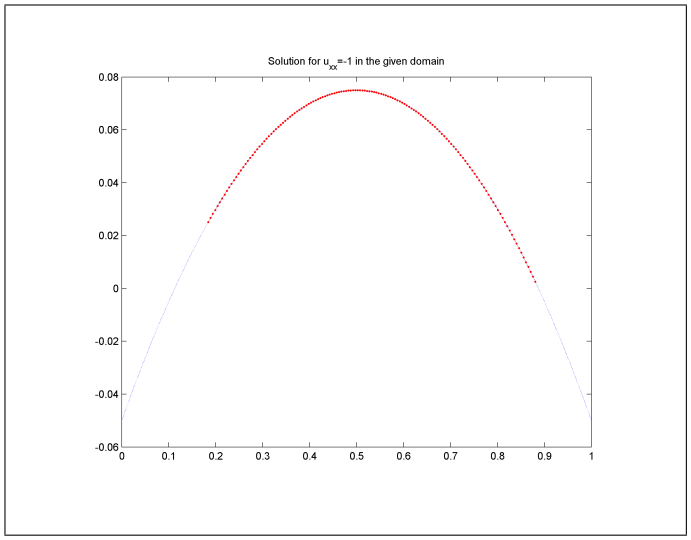

Figure 7: Solution of the 1D Poisson equation with Dirichlet arbitrary boundaries

$$\frac{\partial^2 u}{\partial x^2} = -1 \text{ for } x \in \Omega^-, \qquad u(x) = -0.5x^2 + 0.5x - 0.05 \text{ for } x \in \Gamma$$

The matrix is first built using the fourth-order FD approximation of $\frac{\partial^2 u}{\partial x^2}$, resulting in a sparse matrix with 5 diagonals. The rows corresponding to the points near the interface Γ are then updated with the corrected coefficients, based on the type of extrapolation that is used. Constant extrapolation affects only the right-hand side b of the system, the linear extrapolation updates the diagonal entries of the matrix as well. Both of these extrapolation functions maintain the symmetry of the matrix A.

Quadratic and cubic extrapolations result in changes in non-diagonal entries in a non-symmetric way. Hence, the resulting matrix A is not symmetric. Quadratic extrapolation modifies the coefficients of the diagonal entires, as well as the ones immediately next to them. Cubic extrapolation, on the other hand, modifies two off-diagonal entries in addition to the diagonal.

Once the system's matrix A and right-hand side b are constructed, the solution is obtained directly using any linear solver. In these examples, we use both Matlab's backslash operator, as well as GMRES method. Both result in the same solution. However, Matlab's backslash operator is much faster than the GMRES implementation since it is pre-compiled and optimized.

N	$\lvert x_{numerical} - x_{exact} \rvert_\infty$	Order
16	3.496×10^-4	-
32	8.921×10^-5	1.97
64	2.247×10^-5	1.98
128	5.663×10^-4	1.98

Table 1: Error and order of 1D Poisson with linear extrapolation

N	$\lvert x_{numerical} - x_{exact} \rvert_\infty$	Order
32	2.751×10^-5	-
64	2.398×10^-6	3.52
128	1.858×10^-7	3.69
256	1.271×10^-8	3.87

Table 2: Error and order of 1D Poisson with cubic extrapolation

The solution, shown in figure 7, is computed for number of points N with different extrapolation functions. Table 1 shows the error orders for linear extrapolation, while table 2 shows the cubic extrapolation. We notice clearly how the linear extrapolation provides a second-order accuracy, while the cubic extrapolation is close to fourth-order accuracy.

3.2 2D Steady-State Convection-Diffusion Equation

In this example, the stationary convection-diffusion equation is solved numerical with arbitrary boundary. The equation that is solved in this example is

$$\frac{\partial u}{\partial x} + 2\frac{\partial^2 u}{\partial x^2} + 2\frac{\partial^2 u}{\partial y^2} = -1 \text{ for } x \in \Omega^-$$

with the boundary $u(x, y) = sin(\pi x) + sin(\pi y) + cos(\pi x) + cos(\pi y) + x^6 + y^6$ for $x \in \Gamma$

The boundary is the same as the one shown in figure 3, which is given using the parametric form

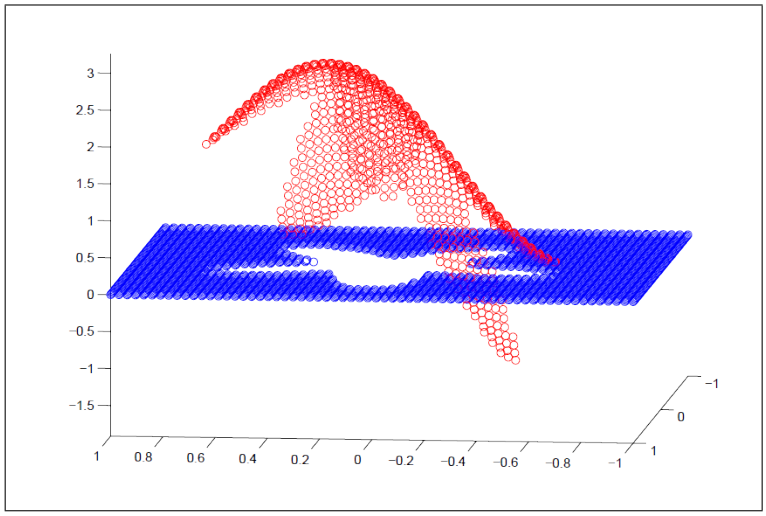

Figure 8: Solution of the 2D convection-diffusion equation with cubic extrapolation

$$x(r) = 0.02\sqrt[2]{5} + (0.5cos(r) + 0.2cos(r)sin(5r)$$
$$u(r) = 0.02\sqrt[2]{5} + (0.5sin(r) + 0.2sin(r)sin(5r)$$

The system is constructed and solved in a similar fashion to the 1D Poisson equation. Each derivate is approximated independently in each axis. A loop over all the grid points is used to test for points that are near the boundary, and update these points with the extrapolated coefficients for the ghost points. The solution to this system, with cubic extrapolation and fourth-order approximations, is shown in figure 8 (figure source: Gibou's paper).

4 Conclusion

The technique discussed in this paper provides a simple method for solving the general steady-state convection-diffusion equation with an arbitrarily-shaped domain. Through the analysis and examples of this technique, it was shown that it could achieve fourth-order accuracy once the correct higher-order approximations are used together with the cubic extrapolation functions near the boundaries.

The technique, however, has certain limitations that may render it impractical for certain applications, especially when high-resolution grid is used.

One of the limitations is that the extrapolation functions work well only when the interface intersection point x_I is far enough from the discretization points x_i, i.e. $|x_I - x_i|$ is large enough. As $|x_I - x_i| \to 0$, the extrapolation functions are distorted heavily, and the approximations of the ghost points are not accurate anymore. This is more likely to happen when the grid resolution N is large.

Another limitation of this technique is that, even though it permits for arbitrary boundaries, these boundaries must be convex. Since points near the boundary are used to extrapolate the ghost points, having non-convex boundary may result in orphan points that cannot be extrapolated nor used for extrapolation.

In general, this technique is useful for low- to mid-resolution problems with convex non-rectangular boundary.