GridGeneration.jl

Metric-Based Adaptive Grid Generation for Structured Multi-Block Domains

Marvyn Bailly

October 31, 2025

Outline

- Overview of GridGeneration.jl
- 1D Formulation of 2D Grid Generation
- Smoothing Methods
- Demonstrate GridGenerationGUI.jl
- Future Work

What is GridGeneration.jl?

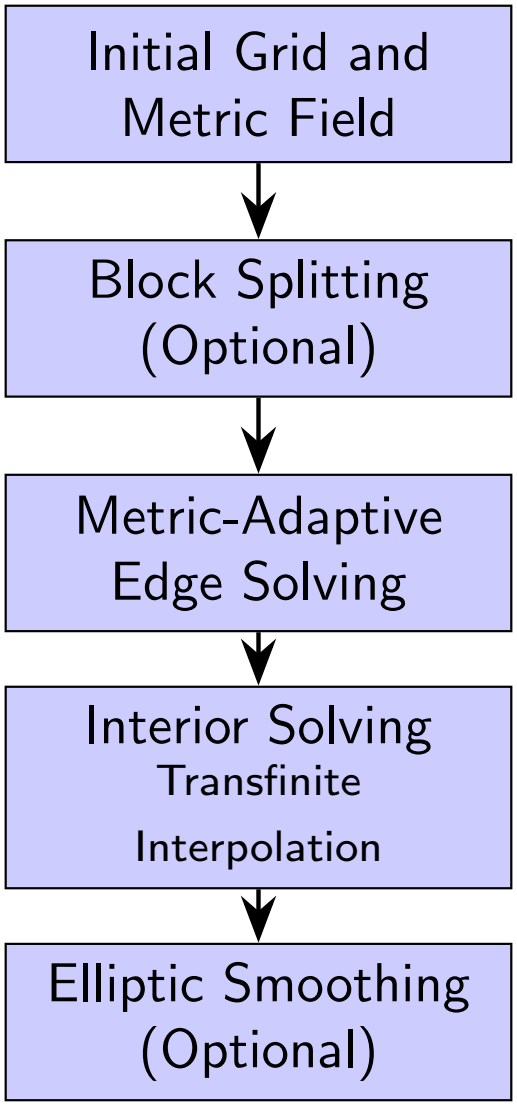
- Julia package for generating structured computational grids
- Uses **metric-based adaptive refinement** for optimal point distribution
- Reformulates 2D grid generation as 1D ODEs along grid lines
- Supports both **single and multi-block**[1] domains
- Provides elliptic smoothing for grid quality "improvement"

Key Idea

Transforms complex 2D grid generation into 1D problems along fixed computational coordinates

[1] 1-1 node connections only.

Core Workflow



Input Format Overview

Required Inputs:

- `initialGrid`: Tortuga-style grid
  - Array of grid blocks containing node coordinates
  - Array of block boundary conditions
  - Array of interface between blocks
- `M`: Riemannian Metric Tensor field
  - Array of 2D metric tensors at each grid node
  - A custom metric can be designed if no residual-based metric is available.

Optional Inputs:

- `params`: User Defined Simulation Parameters

Note:

- Grid Adaptation compared to Generation from Scratch

Block Splitting

- Block splits allow for more control over the grid structure.
- GridGeneration.jl does not support full multi-block capability *yet*.
- Block splits are all 1-1 node connections which may be merged at the end back together.
- New block numbers, interfaces and boundaries are automatically generated during splitting.
- Format: `splitIndices = [[i_splits],[j_splits]]`
  - `i_splits`: Array of i-indices to split along j-direction
  - `j_splits`: Array of j-indices to split along i-direction

Block Splitting Example

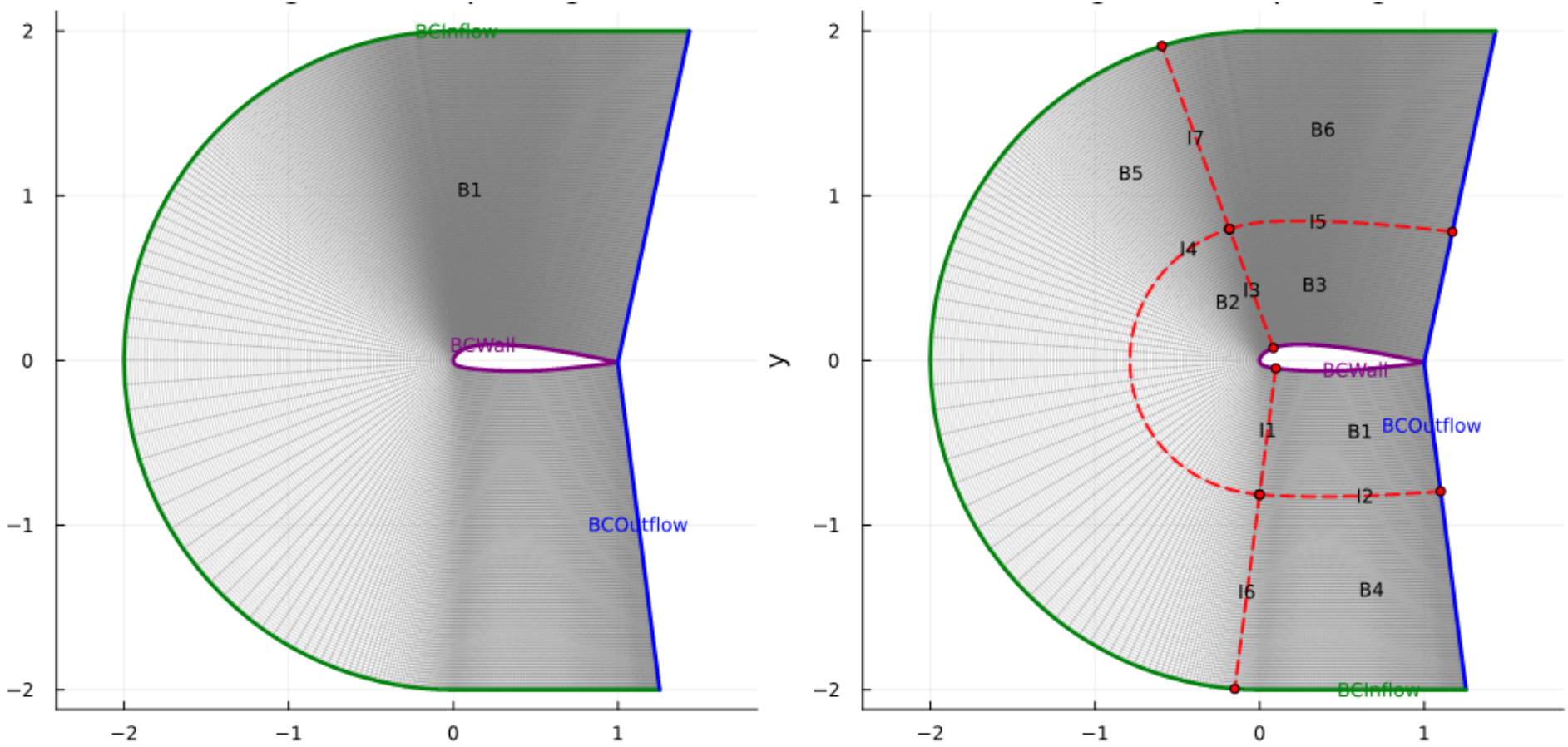


Figure: initial and split grid shown in left and right respectively using user defined splits: `splitIndices = [[300, 500],[40]]`

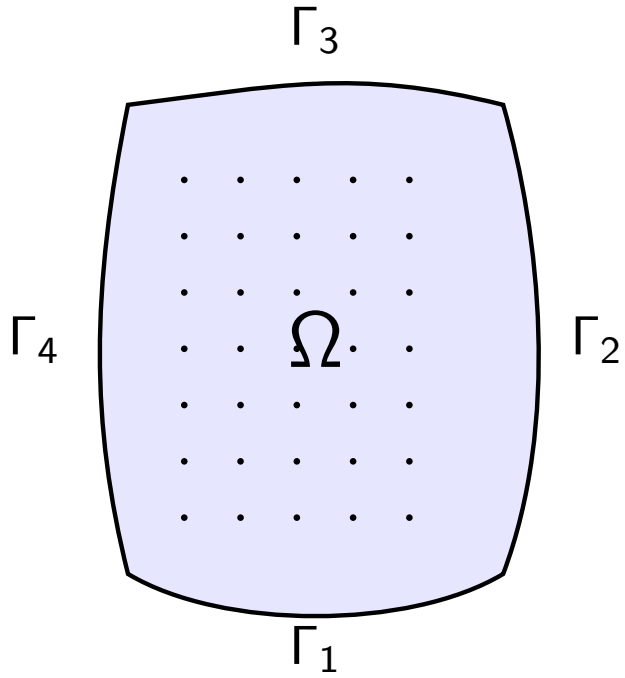
1D Formulation of 2D Grid Generation

Problem Statement

Input: A 2D metric tensor field M and a domain Ω with edges Γ_i, $i = 1, 2, 3, 4$.

Output: A mesh $\mathcal{T}$ of Ω such that the distribution and number of points along the boundaries Γ_i agrees with the metric M.

Needed: A way to represent M and Γ_i in 1D. For now, assume such exists.



ODE Formulation

Let s denote the computational domain $[0, 1]$, $x(s)$ the physical domain, σ is the element-spacing, and $m(x(s))$ the (1D) metric tensor field. We define the loss function as

$$L_{\text{misfit}}(x_s, x, s) = m^2; \quad m(x_s, x, s) = \sigma^2 M x_s^2 - 1.$$

We will use variable subscripts to denote derivatives $\left(x_s := \frac{dx}{ds}\right)$.

Optimal Distribution - Euler-Lagrange

To compute $\mathcal{T}$, we aim to minimizing the functional

$$\mathcal{L}[x(s)] = \int_{\Omega_i} L_{\text{misfit}}(x_s, x, s) ds.$$

The optimal solution $x(s)$ satisfies the Euler-Lagrange equation

$$\frac{\partial L_{\text{misfit}}}{\partial x} - \frac{d}{ds}\left(\frac{\partial L_{\text{misfit}}}{\partial x_s}\right) = 0,$$

Solving the above ODE will give the optimal distribution mapping from computational to physical space along edge Γ_i.

Euler-Lagrange (Cont.)

We can compute the partials to be

$$\begin{aligned}
\frac{\partial L_{\text{misfit}}}{\partial x} &= 2\sigma^2 m M_x x_s^2, \\
\frac{\partial m}{\partial s} &= \sigma^2 (M_x x_s^3 + 2M x_s x_{ss}), \\
\frac{\partial L_{\text{misfit}}}{\partial x_s} &= 4\sigma^2 m M x_s, \\
-\frac{d}{ds}\left(\frac{\partial L_{\text{misfit}}}{\partial x_s}\right) &= -4\sigma^2 (m_s M x_s + m M_x x_s^2 + m M x_{ss}).
\end{aligned}$$

Plugging everything into the Euler-Lagrange equation, we get

$$\begin{aligned}
\frac{\partial L}{\partial x} - \frac{d}{ds}\left(\frac{\partial L}{\partial x_s}\right) = &-8\sigma^4 M^2 x_s^2 x_{ss} - 4\sigma^4 M M_x x_s^4 \\
&- 4\sigma^2 m M x_{ss} - 2\sigma^2 m M_x x_s^2 = 0.
\end{aligned}$$

2nd Order ODE Simplification

We aim to solve the 2nd order ODE

$$-8\sigma^4 M^2 x_s^2 x_{ss} - 4\sigma^4 M M_x x_s^4 - 4\sigma^2 m M x_{ss} - 2\sigma^2 m M_x x_s^2 = 0.$$

Dividing both sides by $-2\sigma^2$ (as $\sigma \neq 0$),

$$\begin{aligned} 4\sigma^2 M^2 x_s^2 x_{ss} + 2\sigma^2 M M_x x_s^4 + 2mMx_{ss} + mM_x x_s^2 &= 0, \\ \iff (4\sigma^2 M^2 x_s^2 + 2mM)x_{ss} &= -(2\sigma^2 M M_x x_s^4 + mM_x x_s^2). \end{aligned}$$

Solving for x_{ss} yields

$$x_{ss} = -\frac{2\sigma^2 M M_x x_s^4 + mM_x x_s^2}{4\sigma^2 M^2 x_s^2 + 2mM}, = -\frac{x_s^2 M_x (2\sigma^2 M x_s^2 + m)}{2M(2\sigma^2 M x_s^2 + m)},$$

Therefore

$$\boxed{x_{ss} = -\frac{M_x}{2M} x_s^2.}$$

Optimal Number of Points

We additionally require

$$\frac{\partial L_{\text{misfit}}}{\partial \sigma} = 0,$$

which gives the optimal number of points along the boundary Γ. We can compute

$$\sigma = \sqrt{\frac{\int_{\hat{\Omega}} p dV}{\int_{\hat{\Omega}} p^2 dV}}; \quad p = M x_s^2.$$

The optimal number of points along Γ_i is then given by $N_i = \frac{1}{\sigma}$.

Final ODE System

Finally, we can rewrite the problem now along edge Γ_i with arc length L_i as:
Distribution:

$$x_{ss} = -\frac{M_x}{2M} x_s^2; \quad x(s=0) = 0, \quad x(s=1) = L_i. \tag{1}$$

Number of Points:

$$\sigma = \sqrt{\frac{\int_{\hat{\Omega}} M x_s^2 dV}{\int_{\hat{\Omega}} (M x_s^2)^2 dV}}. \tag{2}$$

Note that (2) can be computed after solving (1) using trapezoidal rule. We then resolve (1) using the optimal σ.

Analytic Solution

The semi-analytic solution to (1) can by derived by noting that

$$\frac{dM}{ds} = \frac{dM}{dx}\frac{dx}{ds} = M_x x_s,$$

which gives

$$x_{ss} = -\frac{M_x}{2M}x_s^2 = -\frac{M_s}{2M}x_s.$$

Now assuming that $x_s \neq 0$ [2], we can divide both sides by x_s to get

$$\frac{x_{ss}}{x_s} = -\frac{M_s}{2M}, \implies \ln x_s = -\frac{C}{2}\ln M,$$

where C is a constant of integration. Thus we have that

$$x_s = CM^{-\frac{1}{2}}$$

[2] okay since $x_s = 0 \iff x = \text{const}$ which wouldn't be a valid mapping of $x(s)$.

Analytic Solution Continued

Rearranging terms yields

$$M^{\frac{1}{2}} dx = Cds \implies \int_0^x M^{\frac{1}{2}}(\xi) d\xi = C \int_0^s ds = Cs.$$

Noting that $x(0) = 0$ it trivially satisfied, we can solve for C using the boundary condition $x(1) = L_i$ to get

$$C = \int_0^{L_i} M^{\frac{1}{2}}(\xi) d\xi,$$

and so the analytic solution to (1) is given by the implicit equation

$$\int_0^x M^{\frac{1}{2}}(\xi) d\xi = s \int_0^{L_i} M^{\frac{1}{2}}(\xi) d\xi.$$

or if we let $I(x) = \int_0^x M^{\frac{1}{2}}(\xi) d\xi$, then

$$I(x) = I(L_i)s \implies x = I^{-1}(I(L_i)s),$$

Analytic Solution - Numerical Inverision

To compute $x(s)$ numerically we will use that since $\sqrt{M} > 0$, $I(x)$ is strictly increasing and so invertible

$$\int_0^{x(s)} \sqrt{M(\xi)}\, d\xi = s \int_0^1 \sqrt{M(\xi)}\, d\xi, \quad s \in [0,1].$$

Step 1. Discretize the cumulative integral.

$$I_k \approx \int_0^{x_k} \sqrt{M(\xi)}\, d\xi, \qquad I_{k+1} = I_k + \tfrac{1}{2}(w_k + w_{k+1})(x_{k+1} - x_k),$$

where $w_i = \sqrt{M(x_i)}$.
Step 2. Uniform s-grid and target values.

$$s_j = \frac{j}{N}, \qquad t_j = s_j\, I_{\text{tot}}, \qquad I_{\text{tot}} = I_n.$$

Step 3. Invert $I(x)$ by linear interpolation. Find i with $I_i \le t_j \le I_{i+1}$, then

$$x_j \approx x_i + \theta(x_{i+1} - x_i), \qquad \theta = \frac{t_j - I_i}{I_{i+1} - I_i}.$$

Verification - Example 1

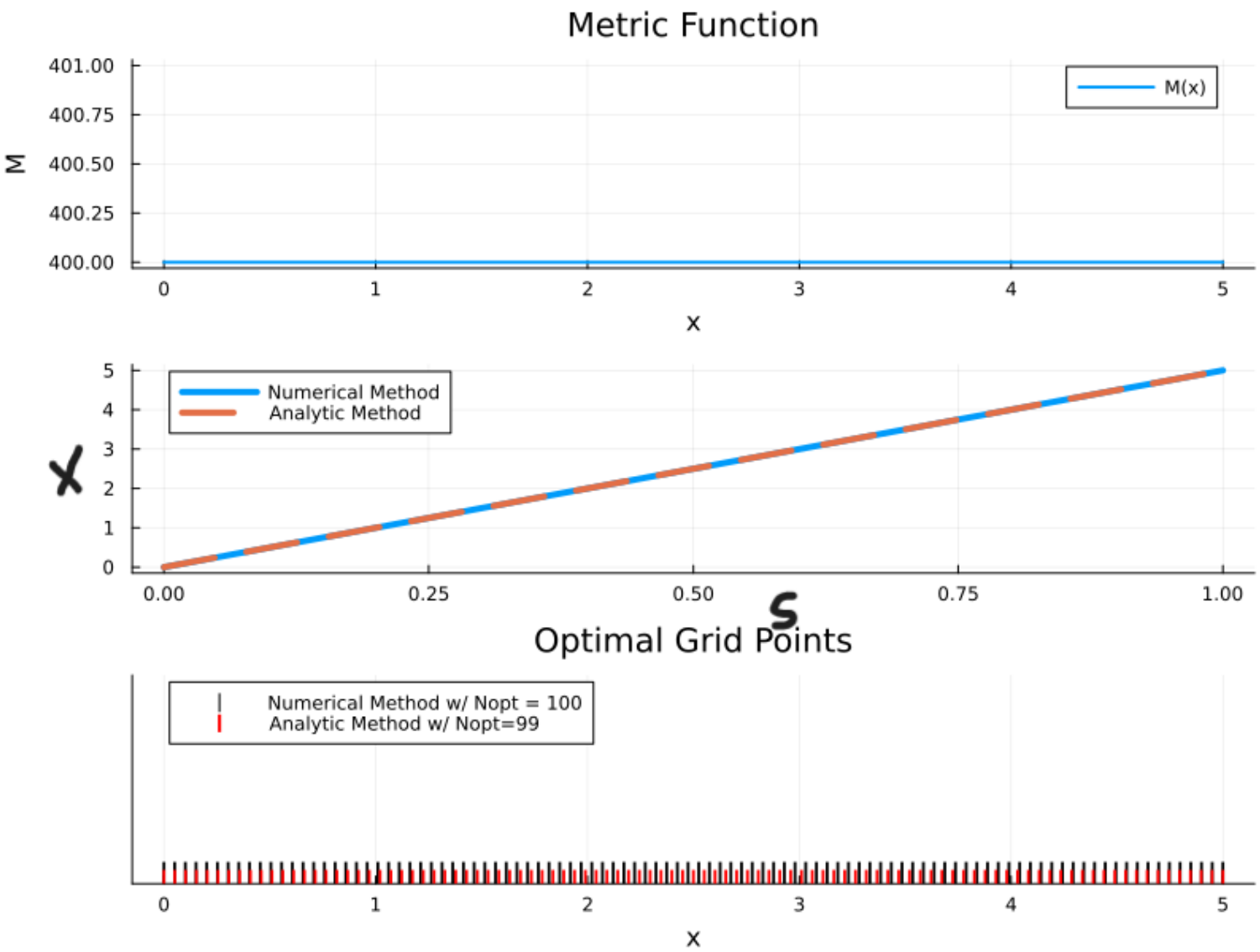


Figure: $M = k^2$ with $L = 5$, $k = 200$, and $N = 100$.

Verification - Example 2

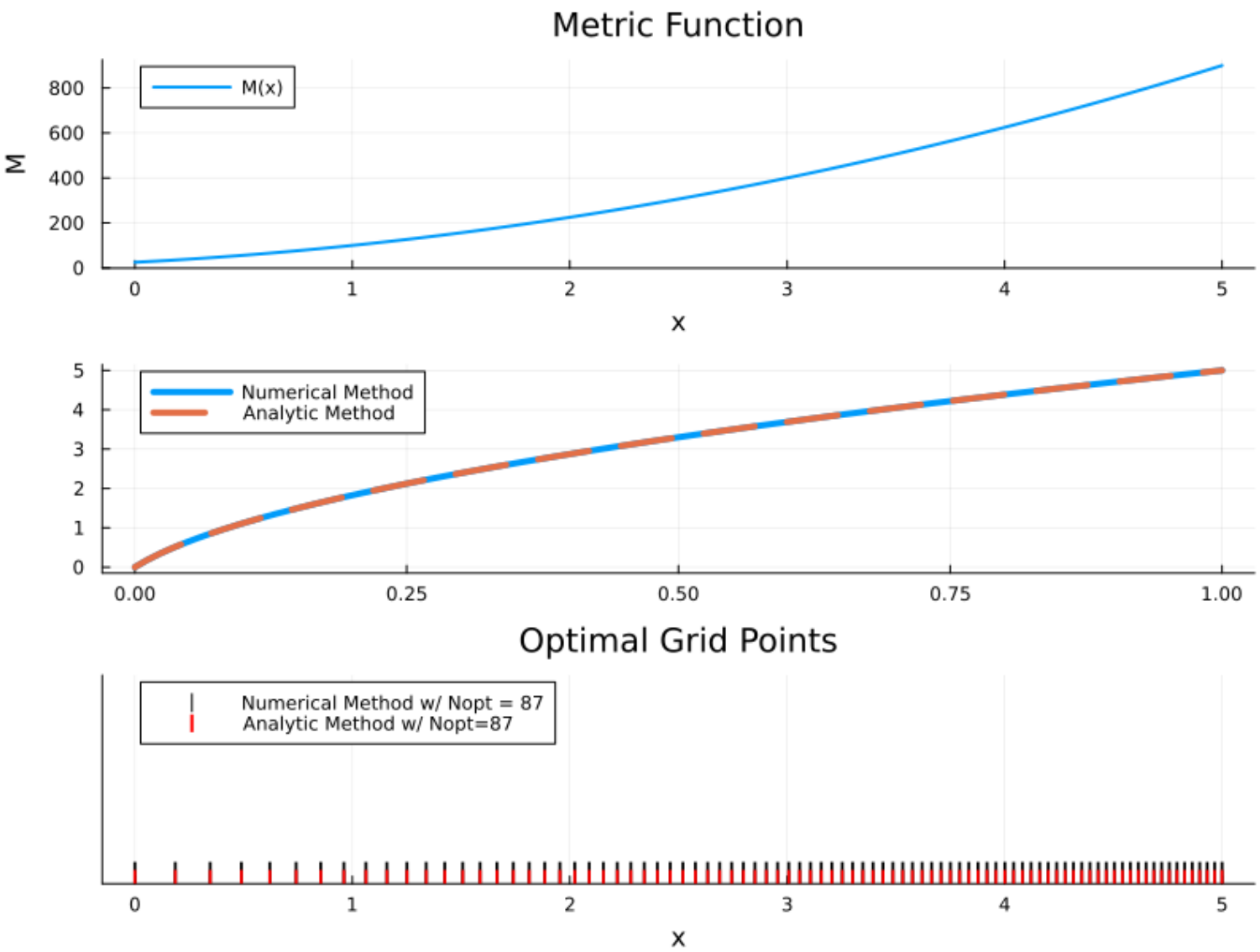


Figure: $M = (a + bx)^2$ with $L = 5$, $a = 5$, $b = 5$, and $N = 100$.

Verification - Example 3

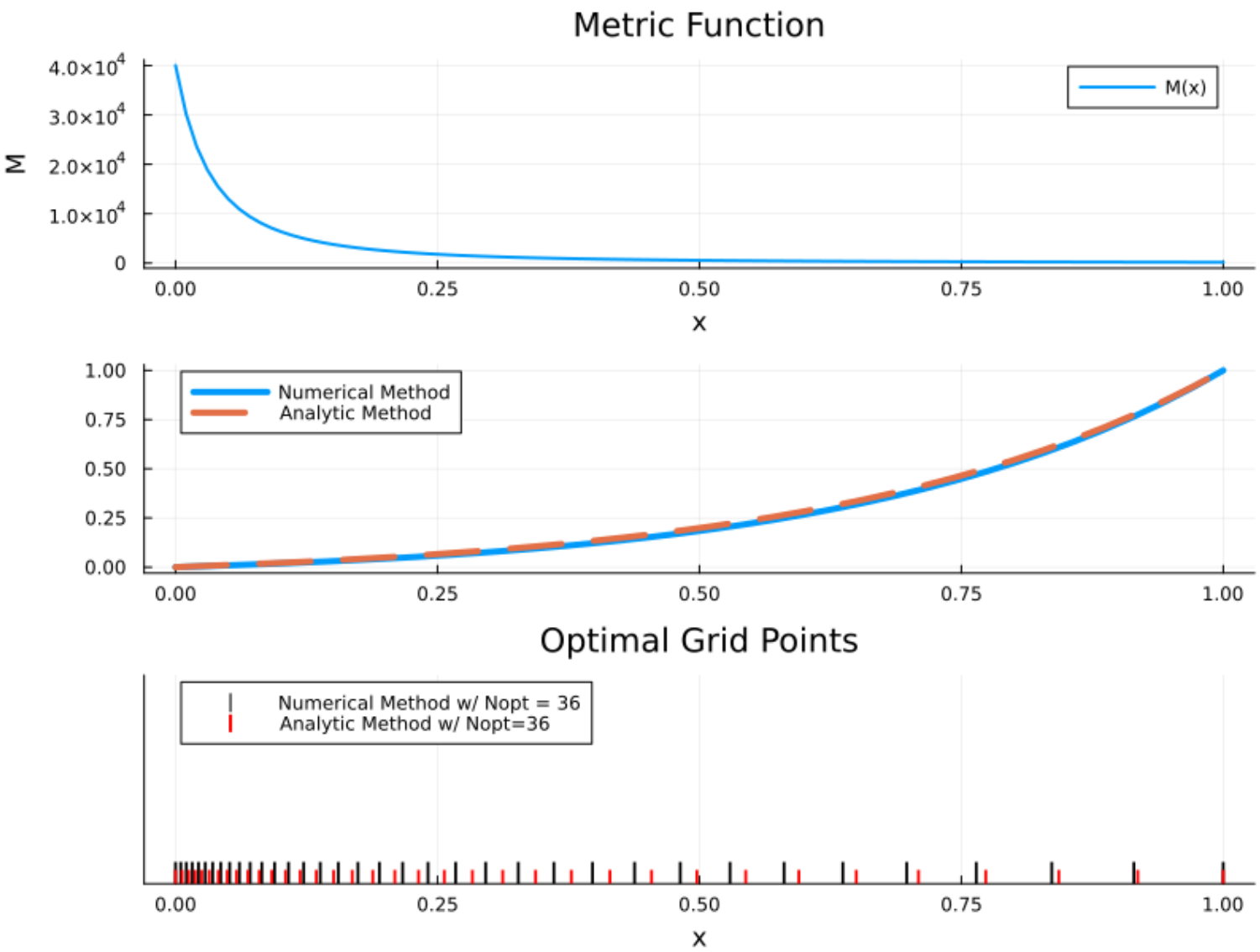


Figure: $M = s(1 + 15(x))^{-2}$ with $L = 1$, $s = 40000$, and $N = 100$.

Verification - Example 4

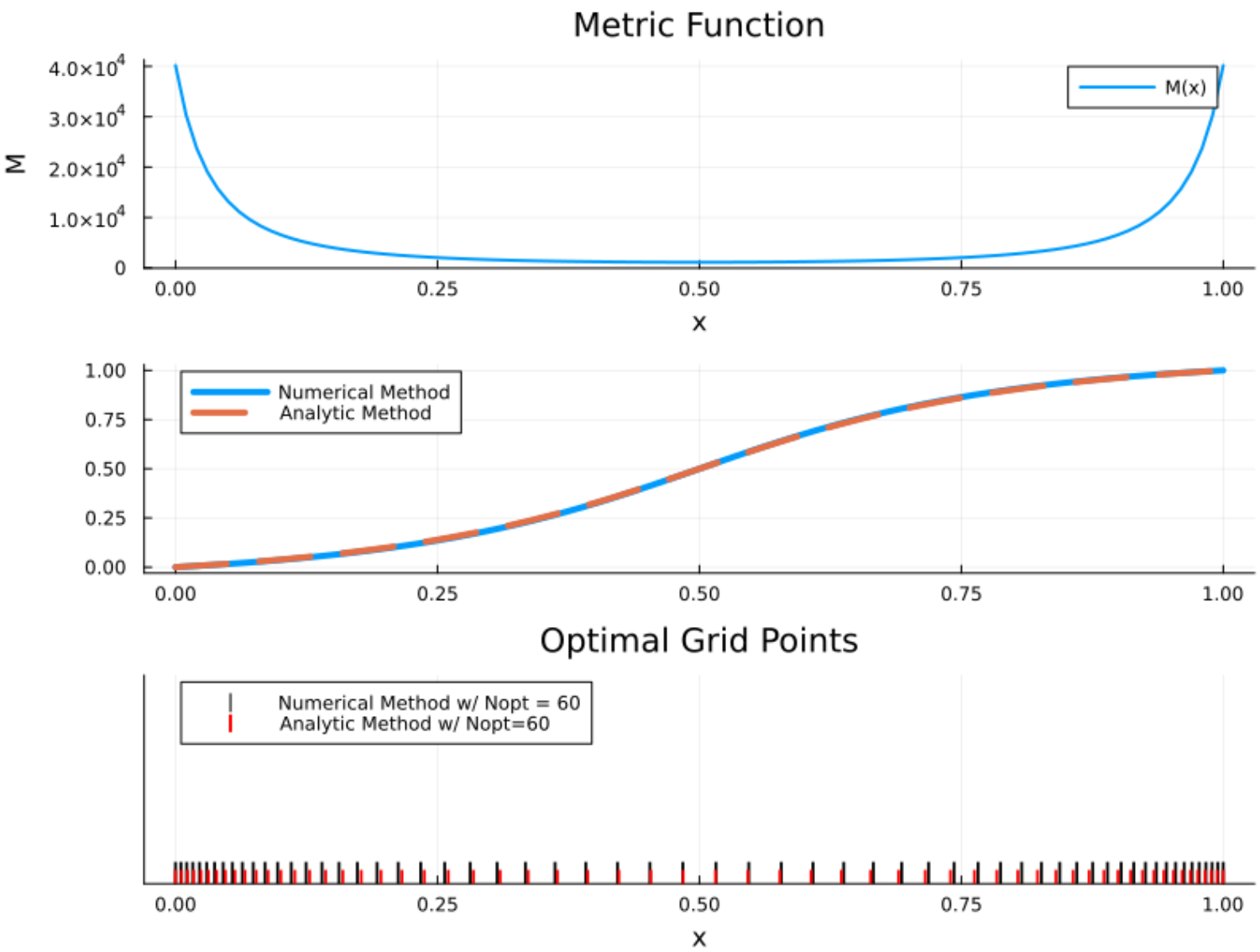


Figure: $M = s(1 + 15(x))^{-2} + s(1 + 15(1 - x))^{-2}$ with $L = 1$, $s = 40000$, and $N = 100$.

How Do we Map the 2D Domain and Metric to Match the 1D Formulation?

2D Extension - Metric Tensor Projection

To find a 1D representation, m, of the 2D tensor M: Suppose Γ_i is discretized with points $\{\gamma_j\}_{j=1}^{N}$ with N points. Let's use a central difference like method to compute m_j at each point γ_j along Γ_i. We approximate using

$$m_j := \frac{1}{|\gamma_{j+1} - \gamma_{j-1}|^2}(\gamma_{j+1} - \gamma_{j-1})^\top \cdot \begin{pmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{pmatrix}_j \cdot (\gamma_{j+1} - \gamma_{j-1}), j = 2, \cdots, N-1,$$

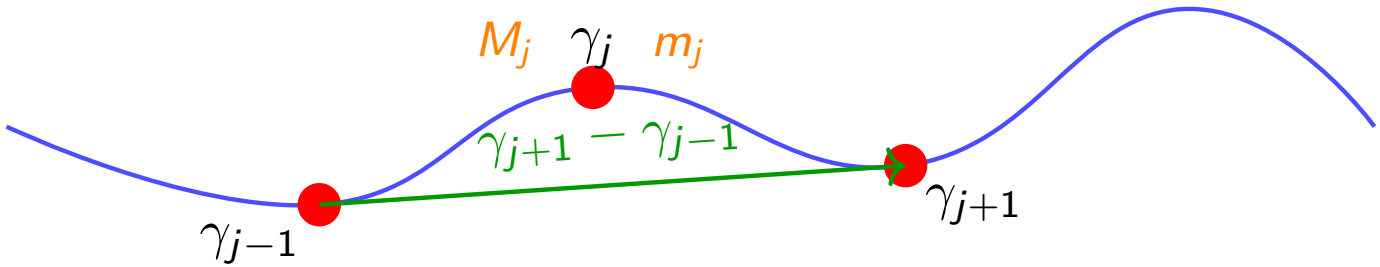
and a one-sided like approximation for the endpoints

$$m_1 := \frac{1}{|\gamma_2 - \gamma_1|^2}(\gamma_2 - \gamma_1)^\top \cdot \begin{pmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{pmatrix}_1 \cdot (\gamma_2 - \gamma_1),$$

$$m_N := \frac{1}{|\gamma_N - \gamma_{N-1}|^2}(\gamma_N - \gamma_{N-1})^\top \cdot \begin{pmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{pmatrix}_N \cdot (\gamma_N - \gamma_{N-1}).$$

Note: Since M is positive definite, $m_j > 0$ for all j by construction.

Metric Tensor Projection - Visualization



Boundary edge Γ_i

The metric projection uses the direction vector $(\gamma_{j+1} - \gamma_{j-1})$ to compute the 1D metric m_j from the 2D tensor M_j at point γ_j.

1D Metric Example 1

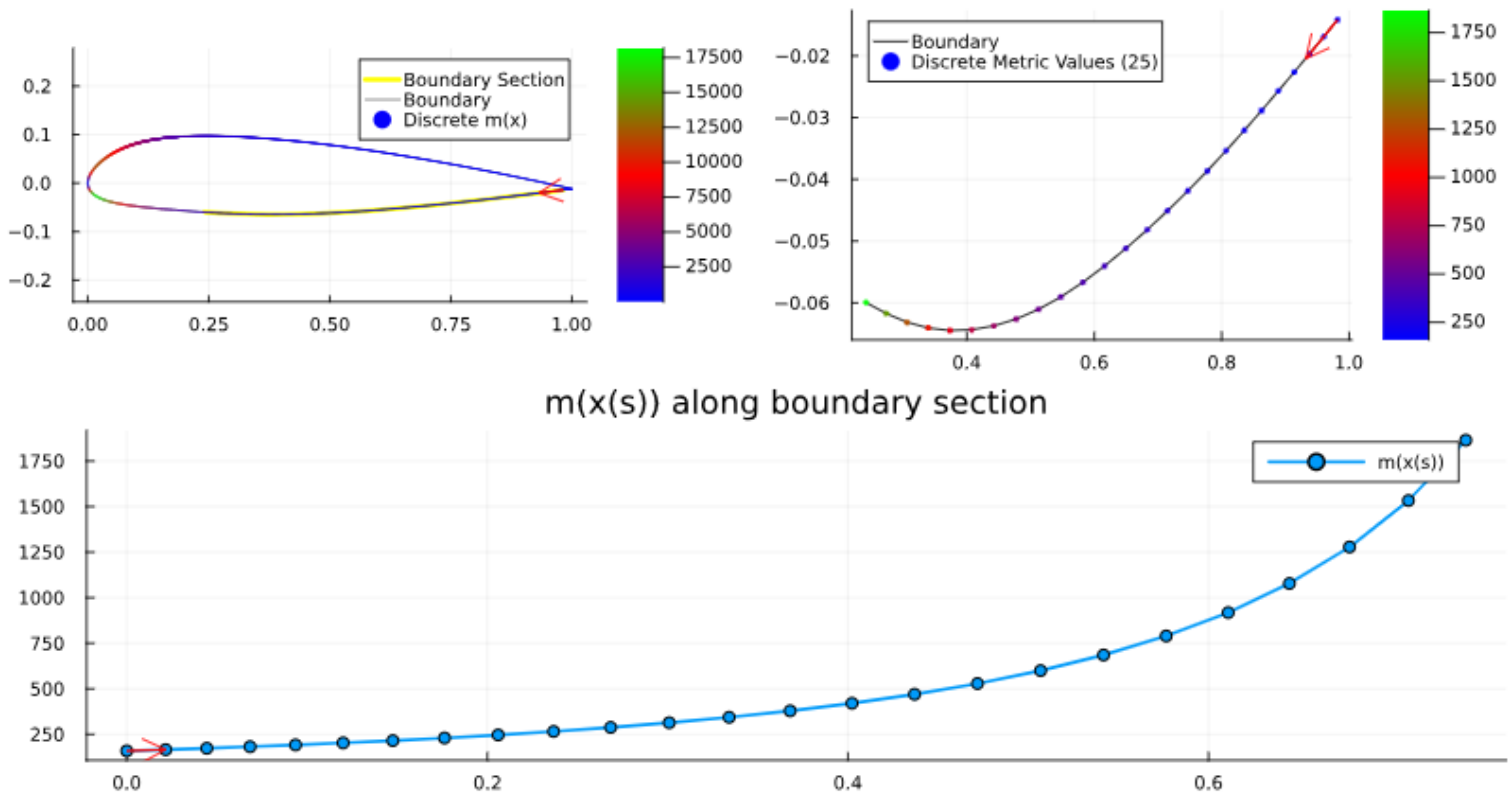


Figure: 2D Metric Tensor M (top) and Projected 1D Metric m (bottom) along boundary Γ_i.

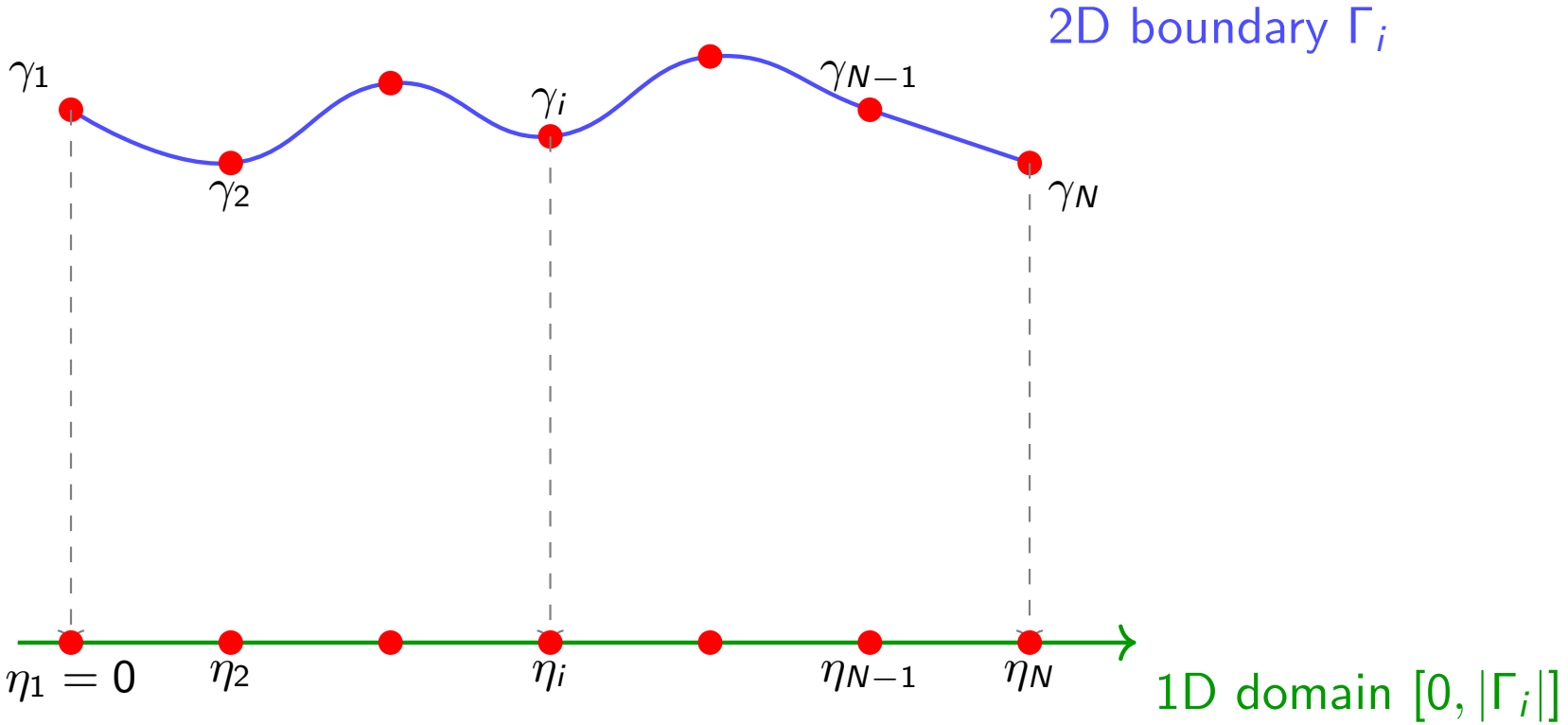
2D Extension - Boundary Projection

We can then project the 2D points $\gamma_i = (x_i, y_i)$ to a 1D domain using the spacing between points. Let η_i be the 1D projected points, where $\eta_1 = 0$ and

$$\eta_i = \sum_{j=2}^{i-1} |\gamma_{j+1} - \gamma_j|, \quad i = 2, \cdots, N.$$

We note that $\eta_N = |\Gamma_1|$.

Boundary Projection - Visualization



The projection maps 2D boundary points γ_i to 1D coordinates η_i by accumulating arc lengths between consecutive points.

2D Extension - Solving the 1D Problem

We can now solve the 1D problem using the projected metric tensor and boundary points on the domain $[0, |\Gamma_1|]$. We use $f_i = \frac{(m_x)_i}{2m_i}$, where $(m_x)_i$ is approximated using the central difference scheme and solve

$$\eta_{ss} = -f(\eta)\eta_s^2; \quad \eta(s=0) = 0, \quad \eta(s=1) = \eta_N = |\Gamma_1|.$$

We can then compute the optimal number of points along Γ_1 using

$$\sigma = \sqrt{\frac{\int_0^1 m(\eta)\eta_s^2 ds}{\int_0^1 (m(\eta)\eta_s^2)^2 ds}}.$$

2D Extension - Mapping Back to 2D

We can then map the 1D solution back to 2D using linear interpolation:
Step 1. Interval identification.

$$\text{Find } j \text{ such that } \eta_i^* \in [\gamma_j, \gamma_{j+1}],$$

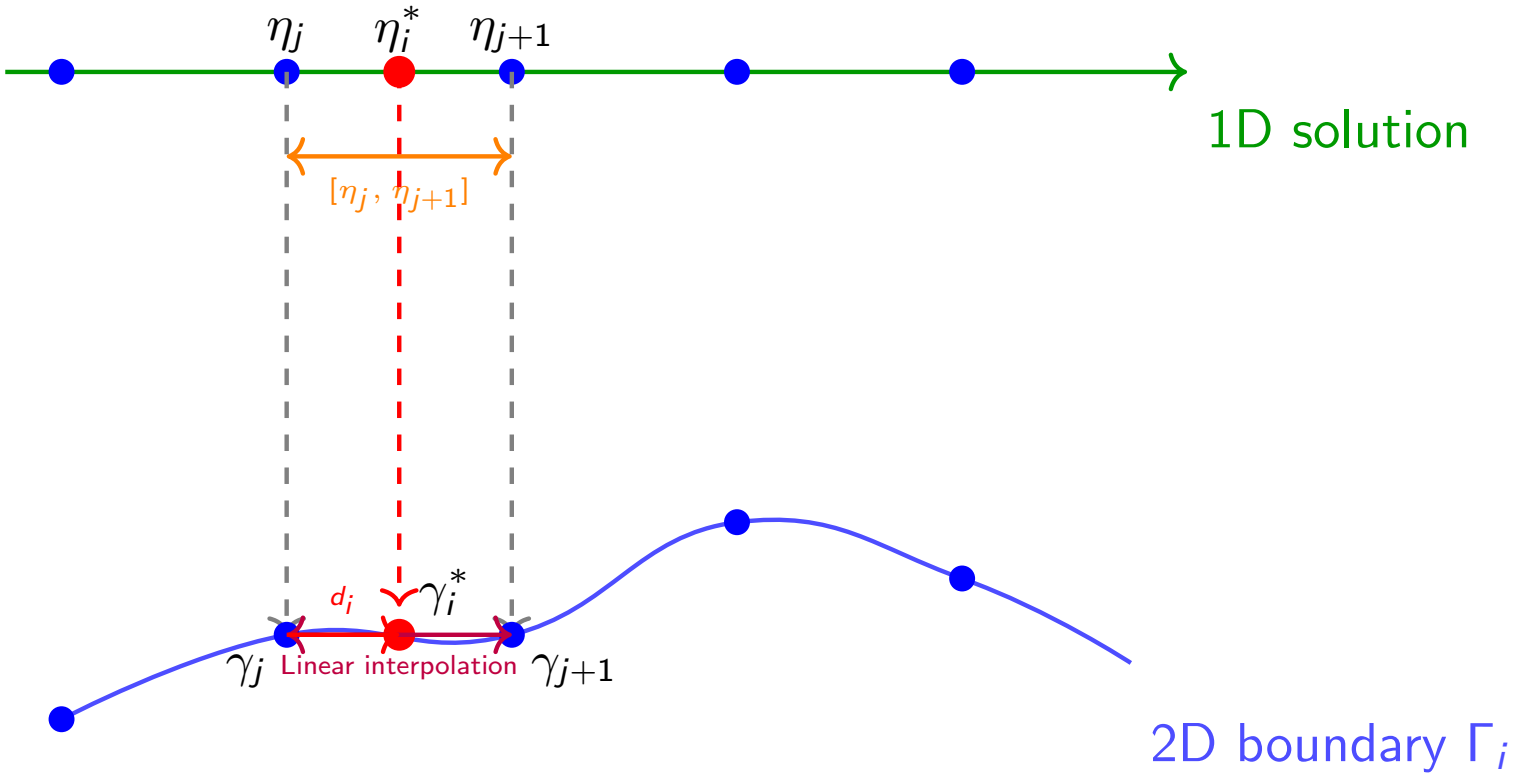
where $\{\gamma_j\}$ is the boundary discretization.
Step 2. Normal distance computation from γ_j.

$$d_i = \frac{|\eta_i^* - \gamma_j|}{|\gamma_{j+1} - \gamma_j|}.$$

Step 3. Projection onto boundary.

$$\gamma_i^* = d_i \gamma_{j+1} + (1 - d_i)\gamma_j.$$

Mapping Back to 2D - Visualization



The optimal 1D point η_i^* is mapped back to the 2D boundary by finding the interval $[\eta_j, \eta_{j+1}]$ it falls in, then linearly interpolating between γ_j and γ_{j+1}.

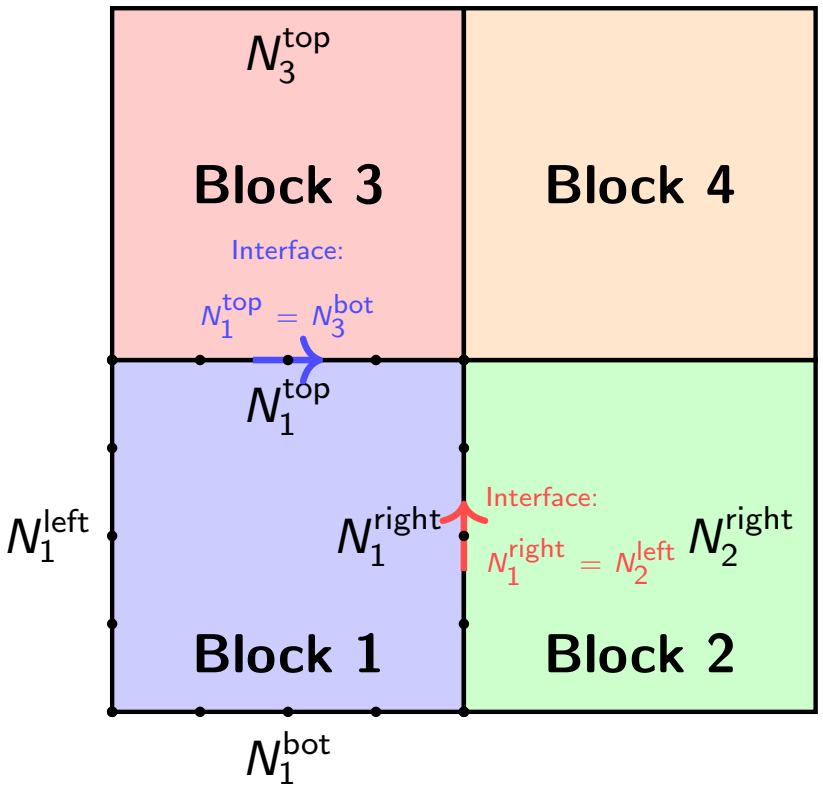
Metric-Adaptive Edge Solving Procedure

Algorithm Overview: Solve edges for a single block with metric field M

Input: Block with edges $\{\Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4\}$, metric field M
for each edge Γ_i with points $\{\gamma_j\}$ **do**
    Compute 1D metric: m_j
    Project to 1D: η_j
end for
for each edge Γ_i **do**
    Solve ODE $x(s)$ using projected m and η
    Compute optimal spacing σ_i
    Compute number of points: $N_i \leftarrow \lceil 1/\sigma_i \rceil$
end for
// Enforce matching opposite edges
$N_{\text{horiz}} \leftarrow \max(N_1, N_3)$ *// Bottom and top*
$N_{\text{vert}} \leftarrow \max(N_2, N_4)$ *// Right and left*
// Resolve with fixed number of points
Resolve Γ_1, Γ_3 with N_{horiz} points
Resolve Γ_2, Γ_4 with N_{vert} points
Map 1D solutions back to 2D
Output: Optimally distributed edge points

Not True Block Splitting

Since GridGeneration.jl only supports multi-block grids with matching interfaces, this means that the number of points along opposite edges must match not only within a block, but also across blocks. Thus, we modify the above procedure to account for multiple blocks:

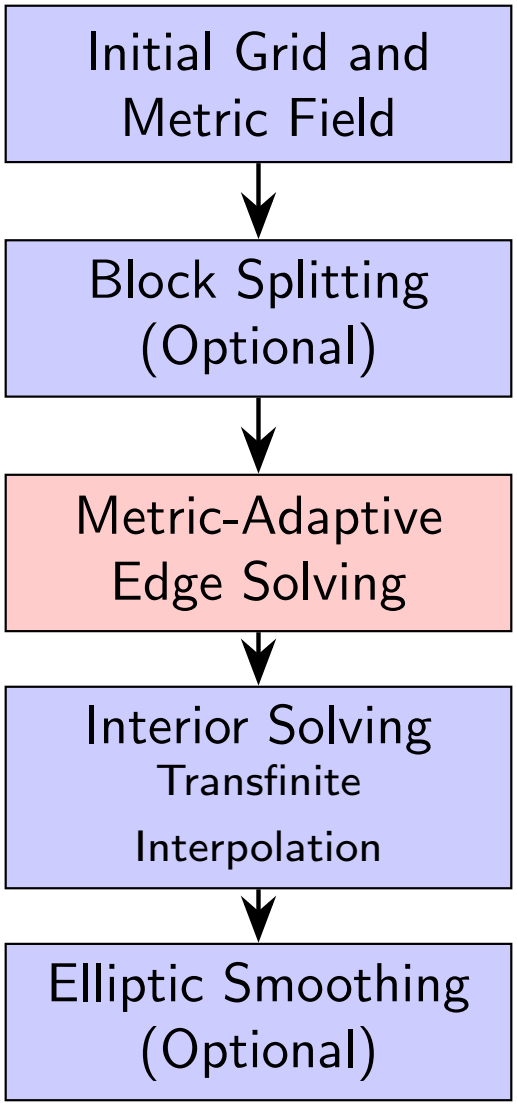


Constraint:

$$N_1^{left} = N_1^{right} = N_2^{left} = N_2^{right} = max(N_1^{left}, N_1^{right}, N_2^{left}, N_2^{right})$$

$$N_1^{bot} = N_1^{top} = N_3^{bot} = N_3^{top} = max(N_1^{bot}, N_1^{top}, N_3^{bot}, N_3^{top})$$

Core Workflow



Interior Solving - Transfinite Interpolation

After solving the edges, we can compute the interior points using transfinite interpolation (TFI) for each block. For a block with edges $\Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4$, we can compute the interior points using

$$x(\xi, \eta) = (1 - \eta)x_{\Gamma_1}(\xi) + \eta x_{\Gamma_3}(\xi) + (1 - \xi)x_{\Gamma_4}(\eta) + \xi x_{\Gamma_2}(\eta) - \text{correction\_terms},$$

where the correction terms account for double counting at the corners. This is a 2D extension of linear interpolation.

Note: TFI is very fast but can lead to poor quality grids, especially for highly skewed or curved boundaries.

Custom Metric

Goal: Define a spatially varying isotropic metric

$$M(x,y) = \begin{bmatrix} w(x,y) & 0 \\ 0 & w(x,y) \end{bmatrix}, \qquad w(x,y) > 0,$$

to control point clustering near the boundary and a user defined hotspot.
Weight function:

$$w(x,y) = \underbrace{w_{\text{rational}}\big(d_{\text{airfoil}}(x,y)\big)}_{\text{distance to boundary}} + \underbrace{w_{\text{rational}}\big(d_{\text{hotspot}}(x,y)\big)}_{\text{distance to hotspot}} + w_{\min}.$$

Profiles:

$$w_{\text{rational}}(d; A, \ell, p) = \frac{A}{1 + (d/\ell)^p}.$$

Parameters:

- A = amplitude (strength of clustering),
- ℓ = decay length,
- p = tail sharpness,
- $w_{\min}$ = positive floor far from features.

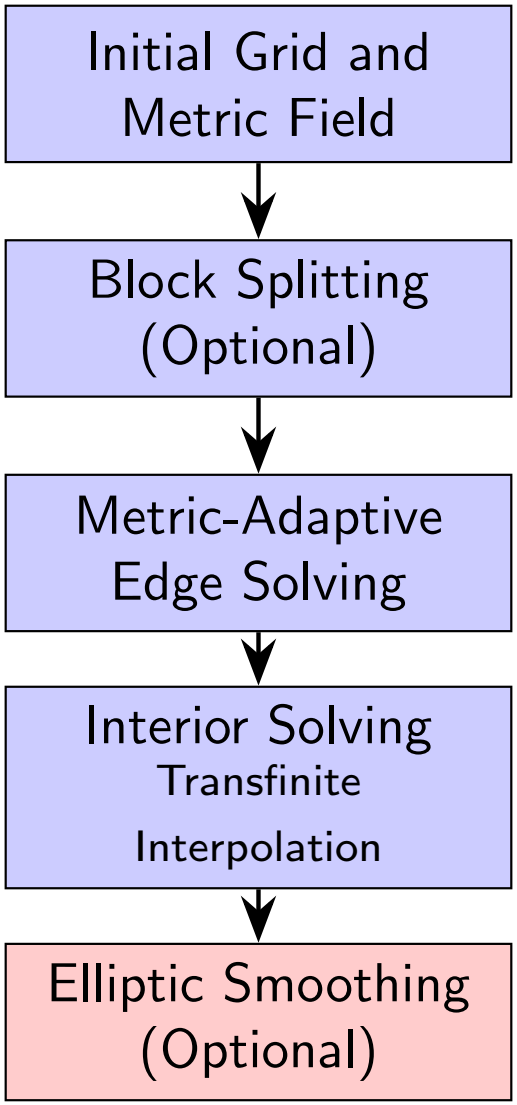
Grid Adaptation Gif

Pull up the gif here:
`https://www.marvyn.com/GridGeneration.jl/stable/pages/SingleBlock/nosplitting/`

Observations:

- Points cluster near the airfoil boundary due to the distance-based metric
- Additional clustering occurs around the hotspot, demonstrating metric adaptability
- But the grid has poor quality (such as angle skewness) near the airfoil surface due to TFI
- In worst cases, TFI can lead to interior points leaving the domain
- To improve quality,
  - Can try to use more interior block splitting
  - Use TFI as initial guess for elliptic and variational methods which take mesh quality into account. Talk more about this later.

Core Workflow



Smoothing Methods

To Improve Mesh quality:

- **Hyperbolic Generation:** Rather than filling in the interior using TFI, let's use a hyperbolic marching method with smoothing terms to propagate the boundary points. Note that the outer domain here can not be specified. Use this method to fill in points near the boundary, then extend to outer domain using TFI or other methods.
- **Elliptic Grid Generation:**[3] Solve a set of coupled Poisson equations to smooth the grid while controlling orthogonality and spacing near boundaries.
- **Variational Methods:** Minimize an energy functional that penalizes poor quality elements, leading to a more uniform and well-shaped grid.

[3] Only method currently implemented in GridGeneration.jl

Elliptic Grid Generation - Poisson Equations

Let (x, y) denote the physical domain and (ξ, η) the computational domain. We solve the following two Poisson equations for $\xi(x, y)$ and $\eta(x, y)$:

$$\xi_{xx} + \xi_{yy} = P(\xi, \eta),$$
$$\eta_{xx} + \eta_{yy} = Q(\xi, \eta),$$

where P and Q are forcing terms that control the orthogonality and smoothness of the grid. After some calculations, we can rewrite the equations in terms of the physical domain as

$$\alpha x_{\xi\xi} - 2\beta x_{\xi\eta} + \gamma x_{\eta\eta} = -J^2(P x_\xi + Q x_\eta),$$
$$\alpha y_{\xi\xi} - 2\beta y_{\xi\eta} + \gamma y_{\eta\eta} = -J^2(P y_\xi + Q y_\eta),$$

where $\alpha = x_\eta^2 + y_\eta^2$, $\beta = x_\xi x_\eta + y_\xi y_\eta$, $\gamma = x_\xi^2 + y_\xi^2$, and $J = x_\xi y_\eta - x_\eta y_\xi$ (the Jacobian of the transformation).

Elliptic Grid Generation - Forcing Terms

To understand the forcing terms P and Q, let's focus on the boundary along η_1. To ensure a smooth decay into the interior, we use exponential decay functions

$$P = P_0 e^{-a(\eta - \eta_1)}, \quad Q = Q_0 e^{-b(\eta - \eta_1)},$$

where $a, b > 0$ are decay rates. The terms P_0 and Q_0 are solved for to ensure orthogonality along the boundary η_1 and a desired grid spacing of s.

Elliptic Grid Generation - Forcing Terms (Cont.)

To solve for P_0 and Q_0, we note that orthogonality along η_1 requires that

$$x_\xi x_\eta + y_\xi y_\eta = 0 \implies \beta = 0.$$

We also want to ensure that the grid spacing along η_1 is s. This can be achieved by enforcing

$$\sqrt{x_\eta^2 + y_\eta^2} = s \implies \alpha = s^2.$$

Using these two conditions, we can solve for x_η and y_η to find:

$$x_\eta|_{\eta=\eta_1} = -\frac{s y_\xi}{\sqrt{x_\xi^2 + y_\xi^2}}, \quad y_\eta|_{\eta=\eta_1} = \frac{s x_\xi}{\sqrt{x_\xi^2 + y_\xi^2}}.$$

We can also compute $x_{\eta\eta}$ and $y_{\eta\eta}$ along η_1 using a taylor expansion:

$$x_{\eta\eta}|_{\eta=\eta_1} = \frac{1}{2}(-7x_1 + 8x_2 - x_3) - 3\, x_\eta|_{\eta=\eta_1}$$
$$y_{\eta\eta}|_{\eta=\eta_1} = \frac{1}{2}(-7y_1 + 8y_2 - y_3) - 3\, y_\eta|_{\eta=\eta_1}.$$

Elliptic Grid Generation - Forcing Terms (Cont.)

Plugging these into the Poisson equations along η_1 gives

$$P_0(\xi, \eta_1) = J^{-1} \left(y_\eta R_1 - x_\eta R_2 \right),$$
$$Q_0(\xi, \eta_1) = J^{-1} \left(y_\xi R_1 + x_\xi R_2 \right),$$

where

$$R_1 = -J^{-2}(\alpha x_{\xi\xi} - 2\beta x_{\xi\eta} + \gamma x_{\eta\eta}),$$
$$R_2 = -J^{-2}(\alpha y_{\xi\xi} - 2\beta y_{\xi\eta} + \gamma y_{\eta\eta}).$$

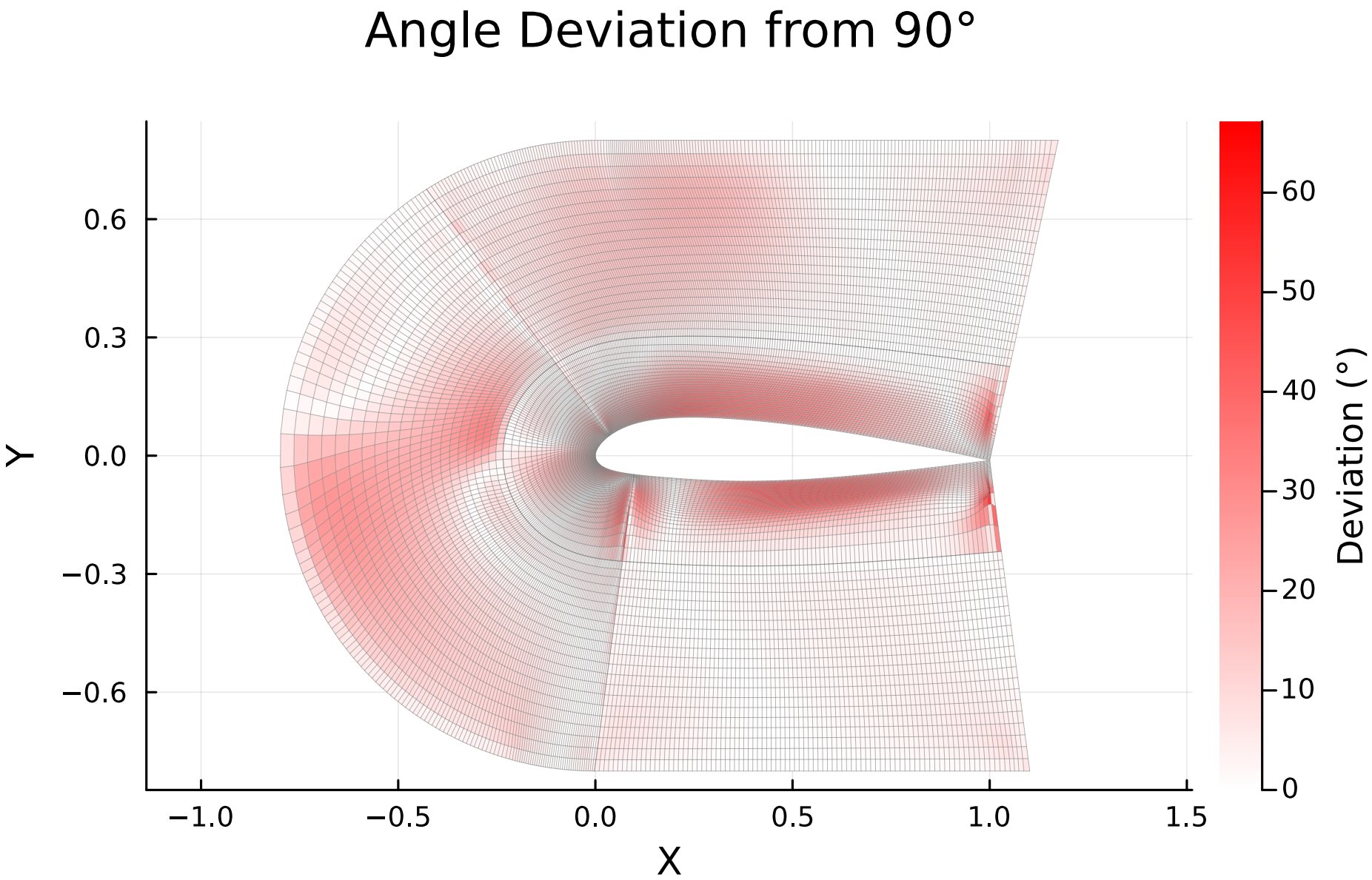
Post Processing Tools

Angle Deviation from Orthogonality: For each grid point (i, j), compute the vector to neighbors $(i, j+1)$ and $(i+1, j)$, denoted $\vec{e}_1$ and $\vec{e}_2$, and compute the angle between them:

$$\theta = \arccos\left(\frac{\vec{e}_1 \cdot \vec{e}_2}{|\vec{e}_1||\vec{e}_2|}\right)$$

Then the deviation from orthogonality is $|\theta - 90^\circ|$.

Post Processing Tools - Example



Using GridGeneration.jl

Installing Julia

Step 1: Install Julia

- Go to the official Julia website: `https://julialang.org/downloads/`
- Download the latest stable version for your operating system (Windows, macOS, Linux).
- Follow the installation instructions specific to your OS.

Step 2: Verify Installation

- Open a terminal or command prompt.
- Type ‘julia‘ and press Enter.
- You should see the Julia REPL

Installing GridGeneration.jl

Step 1: Open Julia REPL

- Open your terminal or command prompt.
- Type ‘julia‘ and press Enter to start the Julia REPL.

Step 2: Import Pkg Module

- In the Julia REPL, type ‘using Pkg‘ and press Enter.

Step 3: Install GridGeneration.jl

- Type ‘Pkg.add("GridGeneration")‘ and press Enter.
- Wait for the installation to complete.

Example of Pipeline

An example of the entire pipeline can be found in examples/generalExample_blank.jl

Using GridGenerationGUI.jl

I thought it would be fun to put together an interactive GUI for GridGeneration.jl using Makie.jl. This allows users to visualize and interactively modify grid generation parameters and see the results in real-time. The parameters can be saved to a Julia script for later use.

Summary

GridGeneration.jl provides:

- **Efficient** metric-based grid generation
- **Flexible** multi-block support with automatic propagation
- **Modular** design for different use cases
- **Quality** Various Smooth Methods to enhance grid quality

Core Idea

Transforms 2D grid generation into tractable 1D ODE problems, enabling efficient metric-based adaptation while maintaining structured grid properties

GitHub: `https://github.com/marvynbailly/GridGeneration.jl`
Documentation: `https://marvyn.com/GridGeneration.jl/dev/`

Thank You!

Questions?