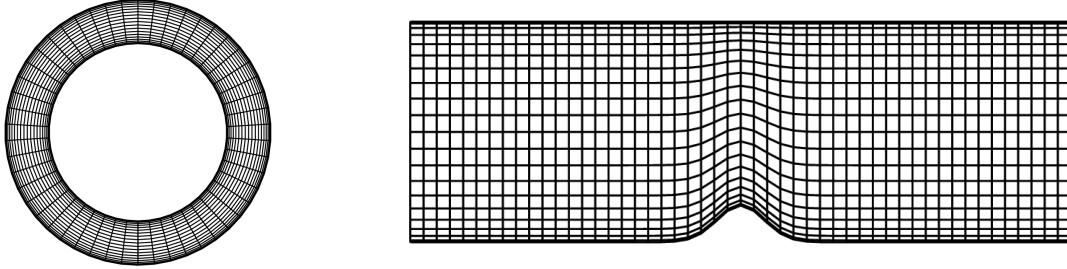


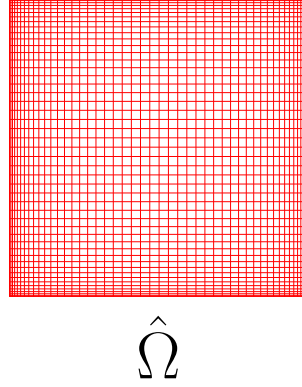
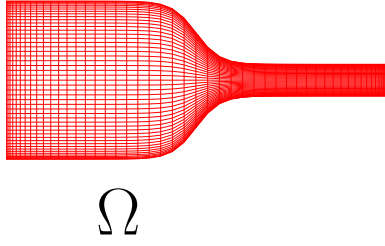
Deformed Geometries

- Here, we consider the solution of the NSE in deformed domains such as those illustrated below.



- In each of these cases the domain $[x, y] \in \Omega$ is a map of the canonical reference domain, $[r, s] \in \hat{\Omega} := [-1, 1]^2$.
- The major differences between general domain deformations and the affine maps that we've considered so far are the need for the *chain rule* to evaluate derivatives in physical $([x, y])$ space and the loss of separability for the elliptic operators.
- This latter point implies that system solution is no longer a simple tensor-contraction (which is a single-line statement in 2D).
- Instead, we will need to use *iterative* methods for the viscous and pressure solves.
- We will begin by introducing a formula for the map, followed by the chain rule, and integration.
- We will then look at each of the operators in the variational statement for the time-discretized advection-diffusion equation, *Find $u^n \in X_0^N$ such that, for all $v \in X_0^N$,*

$$\beta_0(v, u^n) + \nu \Delta t a(v, u^n) = - \sum_{j=1}^k [\beta_j(v, u^{n-j}) + \Delta t c(v, u^{n-j}) - \Delta t (v, q^{n-j})] . (1)$$



- We require a representation of $\mathbf{x} = [x, y] = [x_1, x_2]$ and it is standard to use an *isoparametric* form, in which $u(r, s)$ and $\mathbf{x}(r, s)$ are represented in the same basis.
- If Ω is deformed, then

$$u(r, s) = \sum_{ij} \mathbf{x}_{ij} l_i(r) l_j(s) = \sum_{j=0}^N \sum_{i=0}^N u_{ij} l_i(r) l_j(s) \quad (2)$$

$$\mathbf{x}(r, s) = \sum_{ij} \mathbf{x}_{ij} l_i(r) l_j(s) = \sum_{j=0}^N \sum_{i=0}^N \mathbf{x}_{ij} l_i(r) l_j(s) \in \mathbb{P}_N r, s. \quad (3)$$

- The derivative is computed using the *chain rule*

$$[\underline{u}_x]_{pq} = \left. \frac{\partial u}{\partial x} \right|_{r_p s_q} = \left(\frac{\partial u}{\partial r} \frac{\partial r}{\partial x} + \frac{\partial u}{\partial s} \frac{\partial s}{\partial x} \right) \quad (4)$$

$$= (r_x)_{pq} (\underline{u}_r)_{pq} + (s_x)_{pq} (\underline{u}_s)_{pq} \quad (5)$$

$$[\underline{u}_y]_{pq} = \left. \frac{\partial u}{\partial y} \right|_{r_p s_q} = \left(\frac{\partial u}{\partial r} \frac{\partial r}{\partial y} + \frac{\partial u}{\partial s} \frac{\partial s}{\partial y} \right) \quad (6)$$

$$= (r_y)_{pq} (\underline{u}_r)_{pq} + (s_y)_{pq} (\underline{u}_s)_{pq} \quad (7)$$

- In matrix-vector form,

$$\underline{u}_x = [r_x](\hat{I} \otimes \hat{D})\underline{u} + [s_x](\hat{D} \otimes \hat{I})\underline{u} \quad (8)$$

$$\underline{u}_y = [r_y](\hat{I} \otimes \hat{D})\underline{u} + [s_y](\hat{D} \otimes \hat{I})\underline{u}. \quad (9)$$

- The full gradient (*vector field*) is given by,

$$\begin{aligned} \underline{\mathbf{w}} = \nabla \underline{u} &= \underbrace{\begin{bmatrix} [r_x] & [s_x] \\ [r_y] & [s_y] \end{bmatrix}}_{\mathbf{R}_x} \begin{bmatrix} D_r \\ D_s \end{bmatrix} \underline{u} \\ &= \mathbf{R}_x \mathbf{D}_r \underline{u}. \end{aligned}$$

- Here, we use **bold font** to indicate a *vector field* or a matrix that operates on and/or produces a vector field.

Metrics

- What is $\left. \frac{\partial r}{\partial x} \right|_{r_p s_q}$?

$$\left. \frac{\partial \mathbf{x}}{\partial r} \right|_{pq} = \sum_{ij} \mathbf{x}_{ij} \left. \frac{dl_i}{dr} \right|_p l_j(s_q) = D_r \underline{\mathbf{x}} = (\hat{I} \otimes \hat{D}) \underline{\mathbf{x}}$$

$$\left. \frac{\partial \mathbf{x}}{\partial s} \right|_{pq} = \sum_{ij} \mathbf{x}_{ij} l_i(r_p) \left. \frac{dl_j}{ds} \right|_q = D_s \underline{\mathbf{x}} = (\hat{D} \otimes \hat{I}) \underline{\mathbf{x}}.$$

- *Chain Rule:*

$$\frac{\partial u}{\partial x} = \frac{\partial u}{\partial r} \frac{\partial r}{\partial x} + \frac{\partial u}{\partial s} \frac{\partial s}{\partial x}.$$

- Setting $u = x$ and y ,

$$1 = \frac{\partial x}{\partial x} = \frac{\partial x}{\partial r} \frac{\partial r}{\partial x} + \frac{\partial x}{\partial s} \frac{\partial s}{\partial x}$$

$$0 = \frac{\partial y}{\partial x} = \frac{\partial y}{\partial r} \frac{\partial r}{\partial x} + \frac{\partial y}{\partial s} \frac{\partial s}{\partial x}$$

$$0 = \frac{\partial x}{\partial y} = \frac{\partial x}{\partial r} \frac{\partial r}{\partial y} + \frac{\partial x}{\partial s} \frac{\partial s}{\partial y}$$

$$1 = \frac{\partial y}{\partial y} = \frac{\partial y}{\partial r} \frac{\partial r}{\partial y} + \frac{\partial y}{\partial s} \frac{\partial s}{\partial y}.$$

- At *each point*, (r_p, s_q) , we have the identity,

$$\begin{bmatrix} \frac{\partial x}{\partial r} & \frac{\partial x}{\partial s} \\ \frac{\partial y}{\partial r} & \frac{\partial y}{\partial s} \end{bmatrix} \begin{bmatrix} \frac{\partial r}{\partial x} & \frac{\partial r}{\partial y} \\ \frac{\partial s}{\partial x} & \frac{\partial s}{\partial y} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \quad (10)$$

- Thus we find the inverse metrics, $\frac{\partial r_i}{\partial x_j}$, by inverting the 2×2 matrix of *computable* derivatives $\frac{\partial x_i}{\partial r_j}$, at each gridpoint (r_p, s_q) .
- Note that $x_r, \dots, y_s \in \mathbb{P}_N(r, s)$, but $r_x, \dots, s_y \notin \mathbb{P}_N(r, s)$.

Evaluation of Metrics

- It is instructive to derive a closed-form expression for the metrics $\frac{\partial r_i}{\partial x_j}$.
- Recall Cramer's rule for the inverse of a 2×2 matrix,

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{\mathcal{J}} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}, \quad (11)$$

where the Jacobian $\mathcal{J} = ad - bc$.

- Inserting the metric terms from (10), yields

$$\begin{bmatrix} \frac{\partial r}{\partial x} & \frac{\partial r}{\partial y} \\ \frac{\partial s}{\partial x} & \frac{\partial s}{\partial y} \end{bmatrix} = \frac{1}{\mathcal{J}} \begin{bmatrix} \frac{\partial y}{\partial s} & -\frac{\partial x}{\partial s} \\ -\frac{\partial y}{\partial r} & \frac{\partial x}{\partial r} \end{bmatrix}, \quad (12)$$

with $\mathcal{J} = x_r y_s - x_s y_r$, which is precisely the same definition of the Jacobian introduced below in the discussion of integration rules.

- If the geometry is a polynomial of degree G then the Jacobian will be a polynomial of degree $2G$.
- The metric terms such as $r_x = y_s/\mathcal{J}$ will be *rational* polynomials in r and s , but the metric terms times the Jacobian will be polynomials of degree G .
- For example, if Ω is an arbitrary quadrilateral then x and y are linear functions of r and s , which corresponds to $G=1$, and the Jacobian will be a polynomial of degree at most 2.

- The metric terms in the three-dimensional case have a similar form.
- From the chain rule,

$$\begin{bmatrix} \frac{\partial x}{\partial r} & \frac{\partial x}{\partial s} & \frac{\partial x}{\partial t} \\ \frac{\partial y}{\partial r} & \frac{\partial y}{\partial s} & \frac{\partial y}{\partial t} \\ \frac{\partial z}{\partial r} & \frac{\partial z}{\partial s} & \frac{\partial z}{\partial t} \end{bmatrix} \begin{bmatrix} \frac{\partial r}{\partial x} & \frac{\partial r}{\partial y} & \frac{\partial r}{\partial z} \\ \frac{\partial s}{\partial x} & \frac{\partial s}{\partial y} & \frac{\partial s}{\partial z} \\ \frac{\partial t}{\partial x} & \frac{\partial t}{\partial y} & \frac{\partial t}{\partial z} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (13)$$

- Although a bit more tedious, Cramer's rule can also be applied to the 3D case.
- The 3×3 inverse formula is given by

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}^{-1} = \frac{1}{\mathcal{J}} \begin{bmatrix} (ei - fh) & (ch - bi) & (bf - ec) \\ (fg - di) & (ai - cg) & (dc - af) \\ (dh - eg) & (bg - ah) & (ae - bd) \end{bmatrix}, \quad (14)$$

with $\mathcal{J} = a(ei - fh) - b(di - fg) + c(dh - eg)$.

- As in the 2D case, we proceed by inserting the metric terms (13) into (14),

$$\begin{bmatrix} r_x & r_y & r_z \\ s_x & s_y & s_z \\ t_x & t_y & t_z \end{bmatrix} = \frac{1}{\mathcal{J}} \begin{bmatrix} (y_s z_t - y_t z_s) & (x_t z_s - x_s z_t) & (x_s y_t - y_s x_t) \\ (y_t z_r - y_r z_t) & (x_r z_t - x_t z_r) & (y_r x_t - x_r y_t) \\ (y_r z_s - y_s z_r) & (x_s z_r - x_r z_s) & (x_r y_s - x_s y_r) \end{bmatrix}. \quad (15)$$

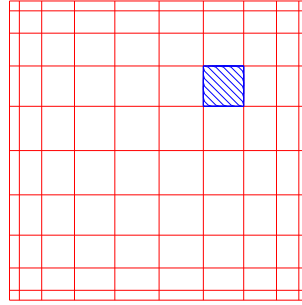
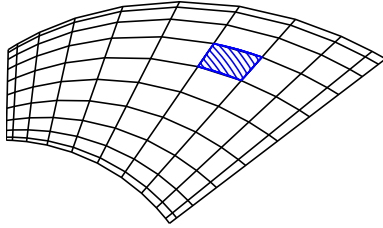
- Here, the Jacobian is the determinant of $\left[\frac{\partial x_i}{\partial r_j} \right]$.
- As in the 2D case, the metric terms $\frac{\partial r_i}{\partial x_j}$ are rational polynomials in (r, s, t) , but the metrics times the Jacobians are polynomials of degree $2G$.
- The Jacobian is a polynomial of degree $3G$.

Integration in Ω

- Just as we transformed our derivatives, we must evaluate the integrals in $\hat{\Omega}$,

$$\int_{\Omega} f dA = \int_{\hat{\Omega}} f \mathcal{J} dr ds.$$

- Here, \mathcal{J} is the *Jacobian* seen above.
- Its definition is slightly different, however.
- It is a scalar field corresponding to the amount of area in Ω that is associated with a unit area in $\hat{\Omega}$.
- An example is shown in the figure below.



- The blue area is given by the *cross product*,

$$\begin{aligned} dA &= \frac{\partial \mathbf{x}}{\partial r} dr \times \frac{\partial \mathbf{x}}{\partial s} ds = \mathcal{J} dr ds \\ &= \left[\frac{\partial x}{\partial r} \frac{\partial y}{\partial s} - \frac{\partial x}{\partial s} \frac{\partial y}{\partial r} \right] dr ds \\ &= \left| \begin{array}{cc} \frac{\partial x}{\partial r} & \frac{\partial x}{\partial s} \\ \frac{\partial y}{\partial r} & \frac{\partial y}{\partial s} \end{array} \right| dr ds. \end{aligned}$$

- That is, \mathcal{J}_{pq} is the determinant of the 2×2 metric tensor at each quadrature point.

- **Mass Matrix:** We note that the mass matrix in the deformed geometry is *diagonal*, evaluated at the GLL points:

$$\bar{B} = \text{diag}(\rho_p \rho_q \mathcal{J}_{pq}) = \mathcal{J}(\hat{B} \otimes \hat{B}).$$

Here, we view \mathcal{J} as a *diagonal matrix*, following standard practice with GLL quadrature.

- In 3D we have a similar form for the Jacobian as the volumetric multiplier of $dr\,ds\,dt$.
- We take the dot product of the variation in t with the cross product introduced in the 2D case,

$$\mathcal{J} = \frac{\partial \mathbf{x}}{\partial t} \cdot \left(\frac{\partial \mathbf{x}}{\partial r} \times \frac{\partial \mathbf{x}}{\partial s} \right). \quad (16)$$

- By symmetry, we can obtain several equivalent representations through permutations of r , s , and t in (16).

Need for *Matrix-Free* Factored Forms

- A significant departure from our efforts to date is that our fast-diagonalization method (FDM) no longer applies because the operator is, in general, no longer separable.
- We therefore need an alternative solution strategy.
- The key idea behind efficient high-order methods is to represent all the operator matrices in *factored forms*, which will lead to a reduction in $O(N^6)$ work and storage complexities to $O(N^4)$ and $O(N^3)$, respectively.
- *Iterative solvers*, which require only matrix-vector products, plus a *preconditioner* are the only viable solvers for the solution of elliptic PDEs in 3D, whether by spectral methods or by finite difference/volume/element methods.
- Since our Poisson and Helmholtz systems are symmetric-positive-definite (SPD), the optimal solver is preconditioned conjugate gradient iteration (PCG).
- In the monodomain case, we can use the FDM as a preconditioner because application of it as an approximate inverse to A is no more expensive than application of A itself.
- For multidomain (i.e., spectral element) methods, the FDM can be used as part of a block preconditioner (e.g., in the context of an overlapping-Schwarz method).
- Alternatively, one can precondition with a multilevel method such as multilevel Schwarz or p -multigrid, where one applies Jacobi (diagonal) smoothing on successively coarser levels.
- Finally, there is another strategy due originally to [Orszag'80] in which one constructs a *sparse* operator, $A_s \approx A$ that is based on a finite difference or finite element discretization of the elliptic operator on the *same* Gauss-Lobatto-Legendre points used by the nodal spectral method.

Often, these operators are spectrally equivalent, which implies that the condition number of $A_s^{-1}A$ is bounded, independent of N .

- For any preconditioner M , the condition number of the preconditioned system is defined as

$$\kappa(M^{-1}A) := \frac{\lambda_{\max}}{\lambda_{\min}}, \quad (17)$$

where

$$\lambda_{\min} = \min_{\underline{z} \in \mathbb{R}^n} \frac{\underline{z}^T A \underline{z}}{\underline{z}^T M \underline{z}}, \quad \lambda_{\max} = \max_{\underline{z} \in \mathbb{R}^n} \frac{\underline{z}^T A \underline{z}}{\underline{z}^T A_s \underline{z}}, \quad (18)$$

are the respective minimum and maximum eigenvalues of $M^{-1}A$.

- For PCG, the number of iterations to reach a desired error tolerance scales as $\sqrt{\kappa}$, so a bounded condition number implies a bounded iteration count.

Bilinear Form in Deformed Coordinates

- As usual, we evaluate the bilinear form,

$$\begin{aligned}
 a(v, u) &= \int_{\Omega} \nabla v \cdot \nabla u \, dA \\
 &= [\mathbf{D}\underline{v}]^T (\mathbf{B}) [\mathbf{D}\underline{u}] \\
 &= [\mathbf{R}_x \mathbf{D}_r \underline{v}]^T (\mathbf{B}) [\mathbf{R}_x \mathbf{D}_r \underline{u}] \\
 &= \underline{v}^T \begin{pmatrix} D_r \\ D_s \end{pmatrix}^T \begin{bmatrix} [\underline{r}_x] & [\underline{s}_x] \\ [\underline{r}_y] & [\underline{s}_y] \end{bmatrix}^T \begin{pmatrix} \mathcal{J}(\hat{B} \otimes \hat{B}) & 0 \\ 0 & \mathcal{J}(\hat{B} \otimes \hat{B}) \end{pmatrix} \begin{bmatrix} [\underline{r}_x] & [\underline{s}_x] \\ [\underline{r}_y] & [\underline{s}_y] \end{bmatrix} \begin{pmatrix} D_r \\ D_s \end{pmatrix} \underline{u}. \\
 &= \underline{v}^T \mathbf{D}_r^T \mathbf{G} \mathbf{D}_r \underline{u}.
 \end{aligned}$$

- In d space dimensions, define

$$G_{ij} := \left(\sum_{k=1}^d \frac{\partial r_i}{\partial x_k} \frac{\partial r_j}{\partial x_k} \right) \bar{B}, \quad i = 1, \dots, d, \quad j = 1, \dots, d, \quad d = 2 \text{ or } 3.$$

- Example, $d = 2$:

$$G = \begin{bmatrix} G_{rr} & G_{rs} \\ G_{sr} & G_{ss} \end{bmatrix} = \begin{bmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{bmatrix}.$$

- **Note:** $G_{ij} = G_{ji}$.
- Each G_{ij} is a *diagonal matrix* with $(N+1)^2$ (or, $(N+1)^d$) entries.

Condition Number Examples

Advection Operator in Deformed Coordinates

Here, we assume that quadrature is on GL or GLL points of order M because it is generally important to dealias the advection operator in order to assure temporal stability. The skew-symmetric bilinear form for advection is

$$c(v, u) = \int_{\Omega} v \mathbf{c} \cdot \nabla u \, dA \quad (19)$$

$$= [J\underline{v}]^T (\mathcal{J}^M B^M) (C_x^M D_x^M \underline{u} + C_y^M D_y^M \underline{u}). \quad (20)$$

A critical observation is that \mathcal{J}^M , B^M , C_x^M , and C_y^M are all *diagonal* matrices (one entry for each quadrature point), and therefore commute such that

$$c(v, u) = [J\underline{v}]^T (B^M C_x^M (\mathcal{J}^M D_x^M) \underline{u} + B^M C_y^M (\mathcal{J}^M D_y^M) \underline{u}). \quad (21)$$

Recall that

$$\frac{\partial u}{\partial x} = \frac{1}{\mathcal{J}} \begin{bmatrix} \frac{\partial y}{\partial s} & -\frac{\partial x}{\partial s} \\ -\frac{\partial y}{\partial r} & \frac{\partial x}{\partial r} \end{bmatrix} \begin{pmatrix} \frac{\partial u}{\partial r} \\ \frac{\partial u}{\partial s} \end{pmatrix}. \quad (22)$$

The terms in the matrix are *polynomials* in (r, s) on $\hat{\Omega}$, as is the Jacobian, $\mathcal{J}(r, s)$. While their combination is a rational polynomial (i.e., *not* a polynomial), multiplication by \mathcal{J} in (21) makes the integrand of (21) a polynomial, which means that the bilinear form $c(v, u)$ can be integrated exactly by using a sufficiently high-order quadrature rule. If the 2D geometry is a polynomial of degree G , and if \mathbf{c} is a polynomial of degree N , then the total integrand is of order $3N + G$. For 3D the integrand is of order $3N + 2G$. Generally, however, it is sufficient to use a quadrature rule that is exact for integrands of order $\approx 2N + 1$ to $\approx 3N/2$.

We can simplify the bilinear form $c(v, u)$ a bit further by first considering $\mathbf{c} \cdot \nabla u$ in d space dimensions using indicial notation,

$$\mathbf{c} \cdot \nabla u = \sum_{i=1}^d c_i \frac{\partial u}{\partial x_i} \quad (23)$$

$$= \sum_{i=1}^d c_i \left(\sum_{j=1}^d \frac{\partial u}{\partial r_j} \frac{\partial r_j}{\partial x_i} \right) \quad (24)$$

$$= \sum_{j=1}^d \left(\sum_{i=1}^d c_i \frac{\partial r_j}{\partial x_i} \right) \frac{\partial u}{\partial r_j} \quad (25)$$

$$= \sum_{j=1}^d \tilde{c}_j \frac{\partial u}{\partial r_j}, \quad (26)$$

with $\tilde{c}_j := \sum_i \frac{\partial r_j}{\partial x_i} c_i$. These simplifications are important given that we typically have multiple fields to advect (e.g., d components of the momentum equations, temperature, and perhaps advected scalars if we have chemical reactions or combustion).

1 SEM-Based Poisson Solves

In simulations of low-Mach and incompressible flows in complex domains the bulk of the computational time is spent in the pressure solve, which represents resolution of the fastest timescale in the problem, namely the acoustic waves, which are infinitely fast in these cases. The benefit of the incompressible model is that the presence of a constant density, ρ , reduces many of the quadratic nonlinearities such as $\rho \mathbf{u}$ to linear terms. In many ways, the resultant pressure-Poisson problem is not overly onerous. It leads to an $n \times n$ system matrix A that is SPD for which the spectrum is well understood and for which there are many effective solution strategies, including preconditioned conjugate gradient (PCG) iteration, additive Schwarz (AS), and various forms of multigrid (MG). If the preconditioner is also SPD, then PCG is the method of choice because it provides the best (i.e., projected) solution in the space of preconditioned search directions in the A -norm with a short-term recurrence. If the preconditioner is not symmetric, then flexible GMRES- k is typically the method of choice, where k is the maximum cardinality of the search space, which is restarted after k iterations. While the computational complexity of GMRES- k includes an $O(nk^2)$ term, one typically aims to use preconditioners such that the number of iterations is smaller than k and the method therefore realizes the best fit property without resorting to restarting. An alternative, which tries to balance between the short-term recurrence of CG and the flexibility of GMRES is to use flexible CG. This approach is viable if only a handful of iterations are required but often is inferior to projection-based GMRES.

Conjugate Gradient Iteration

- The preconditioned conjugate-gradient iteration is given as follows.

```

for  $k = 1 : n$ 
  Solve  $M\underline{z}_k = \underline{r}_{k-1}$ ,  $\rho_k = \underline{z}_k^T \underline{r}_{k-1}$ ,  $\beta_k = \frac{\rho_k}{\rho_{k-1}}$ , if  $k=1$ ,  $\beta_k = 0$ 
   $\underline{p}_k = \underline{z}_k + \beta_k \underline{p}_{k-1}$ 
   $\underline{w}_k = A\underline{p}_k$ ,  $\gamma_k = \underline{p}_k^T \underline{w}_k$ ,  $\alpha_k = \frac{\rho_k}{\gamma_k}$ 
   $\underline{x}_k = \underline{x}_{k-1} + \alpha_k \underline{p}_k$ 
   $\underline{r}_k = \underline{r}_{k-1} - \alpha_k \underline{w}_k$ 
end

```

- In practice, the vectors are overwritten on each iteration, so the algorithm takes on a simple and elegant form,

```

Set  $\rho_1 = 1$ ,  $\underline{r} = \underline{b}$ ,  $\underline{x} = 0$ ,  $\underline{p} = 0$ .
for  $k = 1 : n$ 
  Solve  $M\underline{z} = \underline{r}$ ,  $\rho_0 = \rho_1$ ,  $\rho_1 = \underline{z}^T \underline{r}$ ,  $\beta = \frac{\rho_1}{\rho_0}$ , if  $k=1$ ,  $\beta = 0$ 
   $\underline{p} = \underline{z} + \beta \underline{p}$ 
   $\underline{w} = A\underline{p}$ ,  $\gamma = \underline{p}^T \underline{w}$ ,  $\alpha = \frac{\rho_1}{\gamma}$ 
   $\underline{x} = \underline{x} + \alpha \underline{p}$ 
   $\underline{r} = \underline{r} - \alpha \underline{w}$ 
end

```

- Note that $\underline{x}_k \in \mathcal{R}(\underline{z}_1, \underline{z}_2, \dots, \underline{z}_k)$, which implies

$$\underline{x}_k \in \mathcal{R}(M^{-1}\underline{b}, M^{-1}AM^{-1}\underline{b}, \dots, (M^{-1}A)^{k-1}M^{-1}\underline{b}) \subset \mathcal{R}(M^{-1}). \quad (27)$$

- The first key point is that the preconditioner, M^{-1} , must be of *full rank*.
- That is, it cannot, for example, simply be a coarse-grid operator.
- The second point is that if A has a null-space, then $M^{-1}\underline{z}$ must return a vector that is *orthogonal* to that null-space.
- Finally, it is not difficult to show that \underline{x}_k is the *closest approximation* to \underline{x} in the A -norm:

$$\|\underline{x} - \underline{x}_k\|_A \leq \|\underline{x} - \underline{v}\|_A \quad \forall \underline{v} \in K_k(M^{-1}A, M^{-1}\underline{b}), \quad (28)$$

where $K_k(M^{-1}A, M^{-1}\underline{b})$ is the rank- k Krylov subspace,

$$\mathcal{R}(M^{-1}\underline{b}, M^{-1}AM^{-1}\underline{b}, \dots, (M^{-1}A)^{k-1}M^{-1}\underline{b}) \subset \mathcal{R}(M^{-1}). \quad (29)$$

Generation of $\text{diag}(\bar{A})$.

For Jacobi-preconditioned MG or CG we need to access the diagonal elements of \bar{A} without generating the matrix itself. Fortunately, these entries are readily computable. We illustrate the procedure for the case of a single two-dimensional element for which

$$\bar{A} = D_r^T G_{rr} D_r + D_r^T G_{rs} D_s + D_s^T G_{sr} D_r + D_s^T G_{ss} D_s, \quad (30)$$

with $D_r := \hat{I} \otimes \hat{D}$, $D_s := \hat{D} \otimes \hat{I}$. Here, the $G_{ij} = \sum_{k=1}^d B \frac{\partial r_i}{\partial x_k} \frac{\partial r_j}{\partial x_k}$ are the diagonal matrices that involve the metric terms and the local mass matrix, $B = (\hat{B} \otimes \hat{B}) \mathcal{J}$, where $\hat{B} = \text{diag}(\rho_p)$ and \mathcal{J} is the Jacobian at each nodal point.

Given that \bar{A} is the sum of four matrices it suffices to find the diagonal of each of these. We start with the following observations for matrix multiplication, $C = EF$,

$$c_{ij} = \sum_{k=1}^n e_{ik} f_{kj} \implies c_{ii} = \sum_{k=1}^n e_{ik} f_{ki}. \quad (31)$$

If $E = F^T$, then the diagonal of $F^T F$ is

$$c_{ii} = \sum_{k=1}^n (F^T)_{ik} f_{ki} = \sum_{k=1}^n (f_{ki})^2. \quad (32)$$

Consider the first term on the right of (30) and define $\tilde{D}_r = G_{rr}^{\frac{1}{2}} D_r$ such that $\tilde{D}_r^T \tilde{D}_r = D_r^T G_{rr} D_r$. The diagonal of this term is

$$a_{ii}^1 = \sum_{k=1}^n [(\tilde{D}_r)_{ki}]^2 = \sum_{k=1}^n [G_{rr}]_k (D_r)_{ki}^2. \quad (33)$$

To simplify the notation, let $G := G_{rr}$ (or G_{ss}) and $D = D_r \circ D_r$ (i.e. the Hadamard or pointwise product of the entries of D_r with itself). Denote the entries of the diagonal matrix G as G_{pq} , associated with nodal points (ξ_p, ξ_q) . Note that $D = \hat{I} \otimes (\hat{D} \circ \hat{D})$ is block diagonal.

$$D_A = D_{r2}^T G_{rr} + G_{ss} D_{s2} + O, \quad (34)$$

where O accounts for contributions from the off-diagonal metrics.

```
Da(1,1)=Da(1,1)+Grs(1,1)*Dr(1,1)*Ds(1,1);
Da(end,1)=Da(end,1)+Grs(end,1)*Dr(end,end)*Ds(1,1);
Da(1,end)=Da(1,end)+Grs(1,end)*Dr(1,1)*Ds(end,end);
Da(end,end)=Da(end,end)+Grs(end,end)*Dr(end,end)*Ds(end,end);
```

1.1 Generation of \mathbf{x}_{ij}

A requirement of the isoparametric mapping (3) is to have a set of nodal points \mathbf{x}_{ij} that constitute the basis coefficients for the geometric deformation. Generally, one only has boundary information describing the geometry and there is no *a priori* point distribution for nodes interior to Ω . While there is some leeway in how the points are distributed, it's generally best if the distribution is smooth and representable as a low-order polynomial in (r, s) whenever possible. In essence, we require a *blending function* that smoothly propagates boundary data into the domain interior. The *Gordon Hall mapping* [Gordon & Hall '76] (a.k.a. transfinite interpolation) is a relatively simple and effective approach to meeting these goals.

To illustrate Gordon-Hall (GH), consider a 2D case where we know the nodal point distribution on the domain edges, $\{\mathbf{x}_{0,j}, \mathbf{x}_{N,j}, \mathbf{x}_{i,0}, \mathbf{x}_{i,N}\}$, for $i, j \in \{0, \dots, N\}$. We begin by generating a bilinear map that involves only the vertex nodes. Let l_i denote the usual N th-order Lagrange cardinal functions on (say) the GLL points and l_i^1 , $i = 0, 1$, denote the 1st-order Lagrange cardinal points on $[-1, 1]$. The vertex-only map is given by

$$\hat{\mathbf{x}}(r, s) = \mathbf{x}_{00} l_0^1(r) l_0^1(s) + \mathbf{x}_{N0} l_1^1(r) l_0^1(s) + \mathbf{x}_{0N} l_0^1(r) l_1^1(s) + \mathbf{x}_{NN} l_1^1(r) l_1^1(s). \quad (35)$$

Note that $\hat{\mathbf{x}} \in \mathbb{P}_1(r) \times \mathbb{P}_1(s)$ is *bilinear*. If Ω has straight sides then we use (35) to evaluate $\mathbf{x}_{ij} := \hat{\mathbf{x}}(\xi_i, \xi_j)$ and the map is complete. We have a geometry that is precisely represented by a polynomial of degree $G = 1$.

If Ω has curved sides, we define a new edge function, $\delta(r, s) = \mathbf{x} - \hat{\mathbf{x}}$, for which $\{\delta_{0,j}, \delta_{N,j}, \delta_{i,0}, \delta_{i,N}\}$, are readily computable. We next extend each of these edge perturbations into the domain interior, (that is, into the interior of $\hat{\Omega}$). The GH blending formula is,

$$\begin{aligned} \mathbf{x}(r, s) &= \hat{\mathbf{x}}(r, s) \\ &+ \sum_{i=0}^N [\delta_{i0} l_i(r) l_0^1(s) + \delta_{iN} l_i(r) l_1^1(s)] \\ &+ \sum_{j=0}^N [\delta_{0j} l_0^1(r) l_j(s) + \delta_{Nj} l_1^1(r) l_j(s)]. \end{aligned} \quad (36)$$

We see that (36) yields a tensor-product polynomial that is a combination of linear polynomials in one direction with N th-order polynomials in the other. The N th-order polynomials capture the fluctuation of δ along an edge of $\hat{\Omega}$. The linear polynomials map this perturbation to zero as r or s approaches the opposite edge of $\hat{\Omega}$.

To apply GH in 3D we again use a vertex blending function, but we now need twelve edge-blending functions and six face-blending functions. As in the 2D case, we assume that the boundary points are known. The vertex map is

$$\begin{aligned} \hat{\mathbf{x}}(r, s, t) &= \mathbf{x}_{000} l_0^1(r) l_0^1(s) l_0^1(t) + \mathbf{x}_{N00} l_1^1(r) l_0^1(s) l_0^1(t) \\ &+ \mathbf{x}_{0N0} l_0^1(r) l_1^1(s) l_0^1(t) + \mathbf{x}_{NN0} l_1^1(r) l_1^1(s) l_0^1(t) \\ &+ \mathbf{x}_{00N} l_0^1(r) l_0^1(s) l_1^1(t) + \mathbf{x}_{N0N} l_1^1(r) l_0^1(s) l_1^1(t) \\ &+ \mathbf{x}_{0NN} l_0^1(r) l_1^1(s) l_1^1(t) + \mathbf{x}_{N0N} l_1^1(r) l_1^1(s) l_1^1(t). \end{aligned} \quad (37)$$

In this case, $\hat{\mathbf{x}}$ is *trilinear*. If we define $\mathbf{x}^1 = [\mathbf{x}_{000} \mathbf{x}_{N00} \dots \mathbf{x}_{NNN}]^T$, and $J^1 = \hat{J}^1 \otimes \hat{J}^1 \otimes \hat{J}^1$, where \hat{J}^1 as the $(N+1) \times 2$ interpolation matrix from the vertices ± 1 to the GLL points, then we can evaluate (37) at the GLL points with the simple expression $\mathbf{x} = J^1 \mathbf{x}^1 = (\hat{J}^1 \otimes \hat{J}^1 \otimes \hat{J}^1) \mathbf{x}^1$, which is simpler and less error-prone than trying to evaluate the polynomials in (37) directly. The full GH mapping for the 3D case is illustrated in Fig. 1.

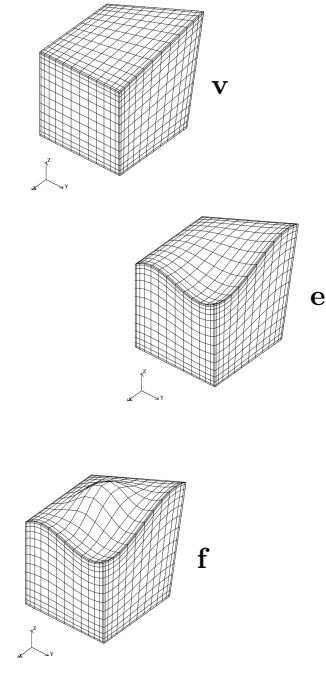
$$\begin{aligned}
\mathbf{v}_{ijk} &= \sum_{\hat{i}\hat{j}\hat{k}} l_{\hat{i}}^1(\xi_i^N) l_{\hat{j}}^1(\xi_j^N) l_{\hat{k}}^1(\xi_k^N) \mathbf{x}_{iN,\hat{j}N,\hat{k}N} \\
\mathbf{e}_{ijk} &= \mathbf{v}_{ijk} + \sum_{\hat{j}\hat{k}} l_{\hat{j}}^1(\xi_j^N) l_{\hat{k}}^1(\xi_k^N) (\mathbf{x}_{i,\hat{j}N,\hat{k}N} - \mathbf{v}_{i,\hat{j}N,\hat{k}N}) \\
&\quad + \sum_{\hat{i}\hat{k}} l_{\hat{i}}^1(\xi_i^N) l_{\hat{k}}^1(\xi_k^N) (\mathbf{x}_{iN,\hat{j},\hat{k}N} - \mathbf{v}_{iN,\hat{j},\hat{k}N}) \\
&\quad + \sum_{\hat{i}\hat{j}} l_{\hat{i}}^1(\xi_i^N) l_{\hat{j}}^1(\xi_j^N) (\mathbf{x}_{iN,\hat{j}N,k} - \mathbf{v}_{iN,\hat{j}N,k}) \\
\mathbf{f}_{ijk} &= \mathbf{e}_{ijk} + \sum_{\hat{i}} l_{\hat{i}}^1(\xi_i^N) (\mathbf{x}_{iN,j,k} - \mathbf{e}_{iN,j,k}) \\
&\quad + \sum_{\hat{j}} l_{\hat{j}}^1(\xi_j^N) (\mathbf{x}_{i,\hat{j}N,k} - \mathbf{e}_{i,\hat{j}N,k}) \\
&\quad + \sum_{\hat{k}} l_{\hat{k}}^1(\xi_k^N) (\mathbf{x}_{i,j,\hat{k}N} - \mathbf{e}_{i,j,\hat{k}N})
\end{aligned}$$


Figure 1: Application of the Gordon-Hall algorithm in \mathbb{R}^3 . Subscripts i , j , and k range from 0 to N . Summations involving \hat{i} , \hat{j} , and \hat{k} range from 0 to 1, accounting for contributions from opposing faces in each of the three coordinate directions. The outputs of each of the three phases, vertex-, edge-, and face-extension, are indicated by \mathbf{v} , \mathbf{e} , and \mathbf{f} , respectively. The final geometry is given by $\mathbf{x} := \mathbf{f}$ and is defined at all interior and surface points.

Laplacian Blending on $\hat{\Omega}$

- As an alternative to Gordon-Hall (also known as trans-finite) blending, one can use Laplacian blending, in which we solve for the interior nodal positions, x_{ij} and y_{ij} by solving Laplace's equation on $\hat{\Omega} := [-1, 1]^2$.
- Let $u(r, s) \in X^N$ represent either $x(r, s)$ or $y(r, s)$.
- If $u = u_b$ on $\partial\Omega$, then we use the standard splitting $u = u_0 + u_b$, where $u_0 \in X_0^N$ is the unknown field in the interior of $\hat{\Omega}$ which vanishes on $\partial\Omega$.
- The unknown basis coefficients are found by computing the solution to

$$A\mathbf{u}_0 = -R\bar{A}\bar{\mathbf{u}}_b, \quad (38)$$

where all operators are defined on $\hat{\Omega}$,

$$\bar{A} = \hat{B} \otimes \hat{A} + \hat{A} \otimes \hat{B} \quad (39)$$

$$A = R\bar{A}R^T \quad (40)$$

$$R = (R_s \otimes R_r). \quad (41)$$

- Because it is on $\hat{\Omega}$, the system (38) is easily solved using fast diagonalization.

Laplacian Blending on General Domains

- Laplacian blending is also useful on more general domains, particularly if one wishes to project (say) a faceted face to a smooth curved surface, which is a scenario that is frequently encountered in the application of high-order methods.
- The idea is solve for the *mesh displacement*, δx and δy for a given initial mesh (x_0, y_0) .
- The resulting mesh field will be

$$x = x_0 + \delta x \tag{42}$$

$$y = y_0 + \delta y. \tag{43}$$

- Unlike the preceding example, where we solve for x and y in their entirety, here, we solve only for the perturbation $(\delta x, \delta y)$ and we use the existing domain $\Omega_0 = \mathcal{R}(x_0, y_0)$ as the computational framework for Laplace's equation.
- More on this in an updated version of these notes.