Solving Incompressible NS using Discontinuous Galerkin Method

November 17, 2017

1 Background

1.1 Lagrange interpolation

Langrangian interpolation is used very heavily in Galerkin methods and it is worthwile to go through the theory. Lagrangian interpolation is essentially interpolating a function using an N-degree polynomial such that it exactly satisfies the function at $N+1$ points.

1.1.1 1D Lagrange interpolation

Let $f(x)$ be the function we wish to interpolate. Let $f^N(X)$ be its order N interpolationg polynomial which perfectly satisfies $f(x)$ on the points $x_0, x_1, x_2, ...x_n$

We define the order N Langrangian at point x_i as

$$L_i^N(x) = \prod_{j=0, j\neq i}^{N} \frac{x - x_j}{x_j - x_i}$$

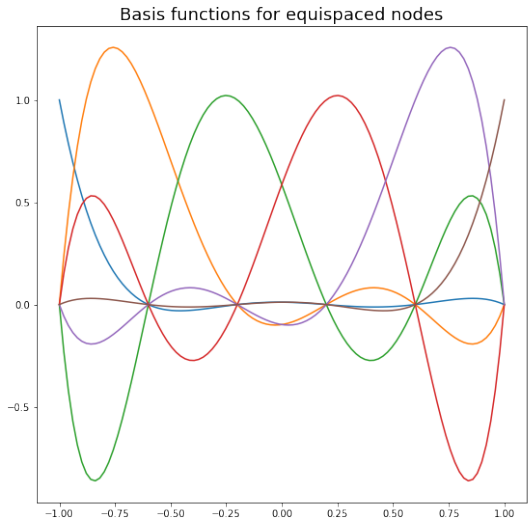
It can easily be shown that

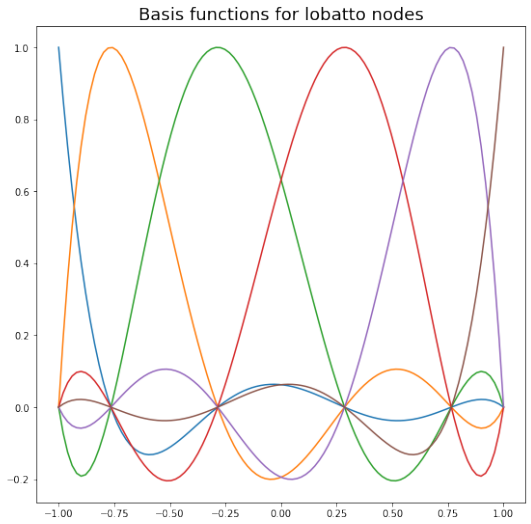
$$L_i^N(x_j) = \delta_i$$

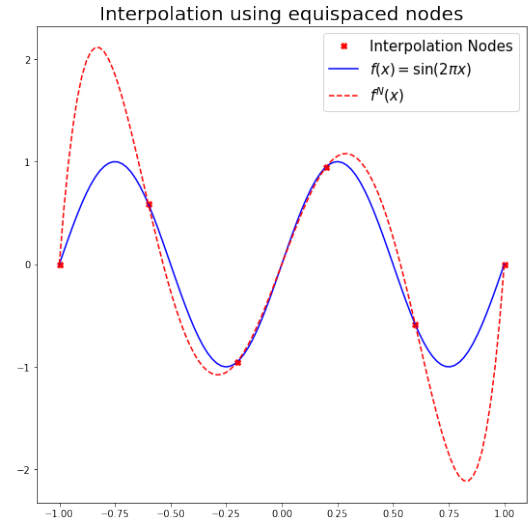
The interpolating function can then be written as

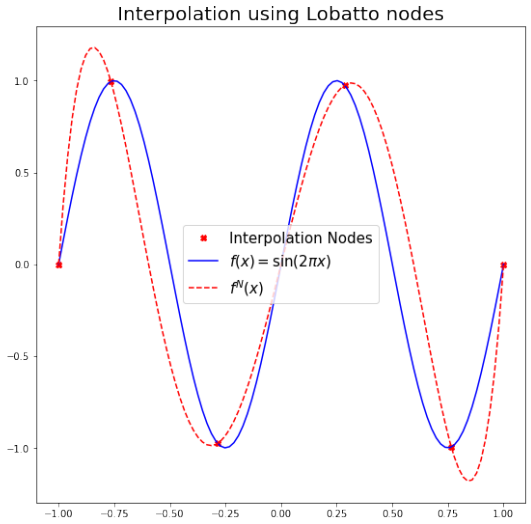
$$f^N(x) = \sum_{i=0}^{N} L_i^N(x) f(x_i)$$

Note: * $f^n(x_i) = f(x_i)$ * Order of $L_i^N(x)$ is N









2D Lagrange interpolation 2D Lagrange interpolation on a rectangular grid is extremely simple.

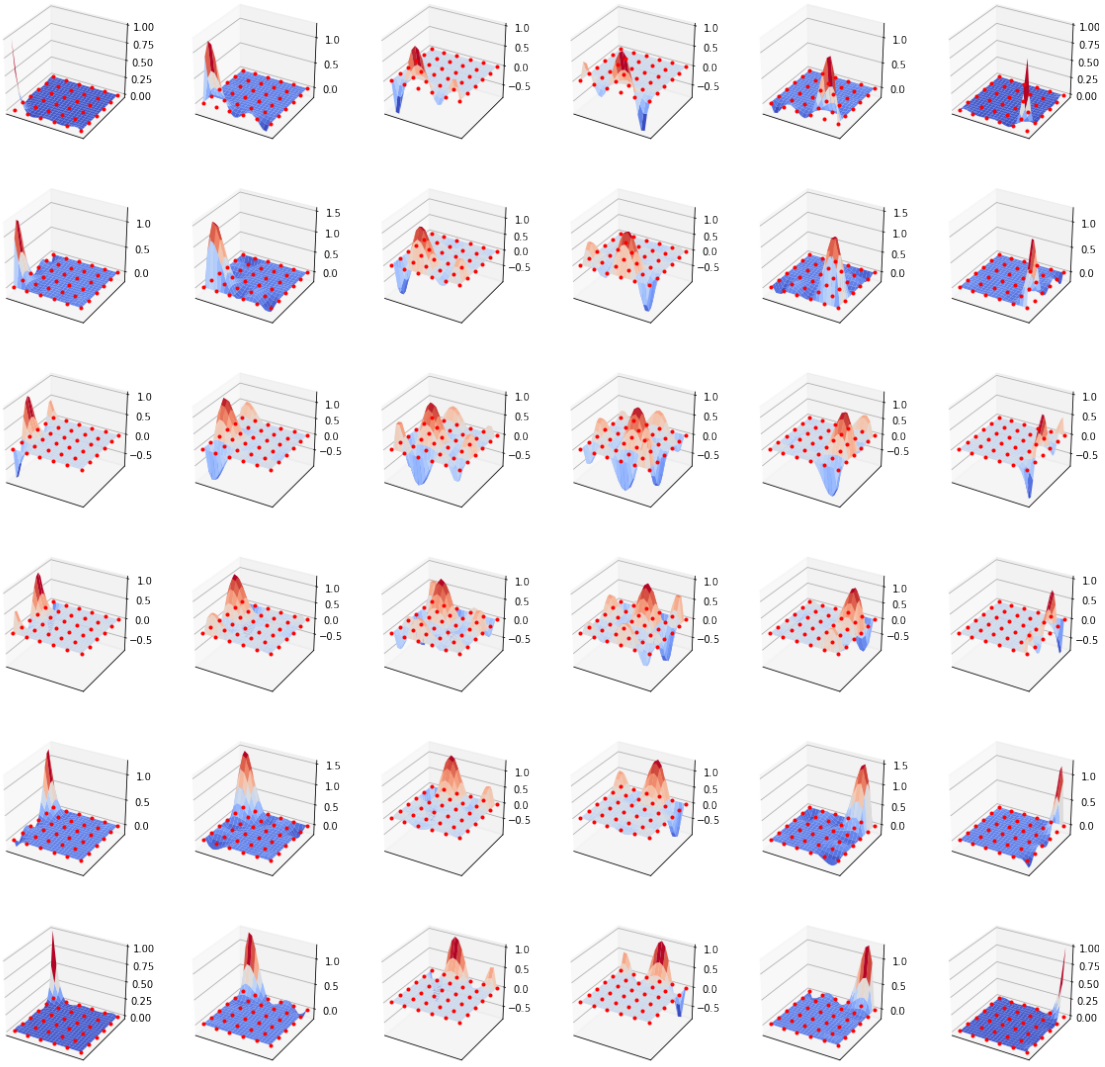
Let the interpoliating polynomial to the function $f(x,y)$ be $f^{NM}(x,y)$ which satisfies the function of a grid of points formed by $x_0, x_1, ...x_N$ and $y_0, y_1, ...y_M$

$$L_{ij}^{NM}(x,y) = L_i^N(x_i)L_j^M(y_j)$$

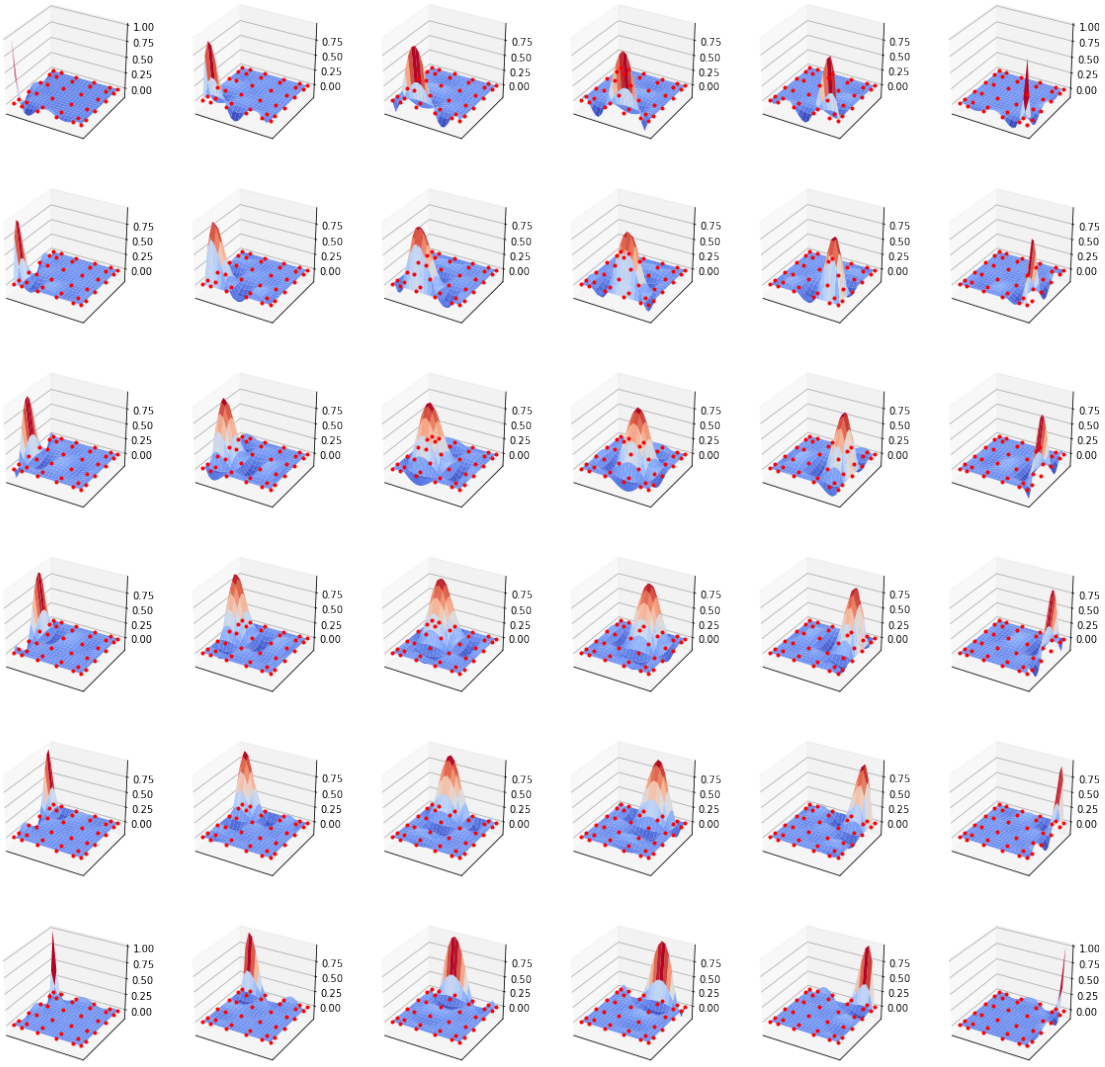
$$f^{MN}(x,y) = \sum_{i=0}^{N}\sum_{j=0}^{M} L_{ij}^{NM}(x,y)f(x_i,y_j)$$

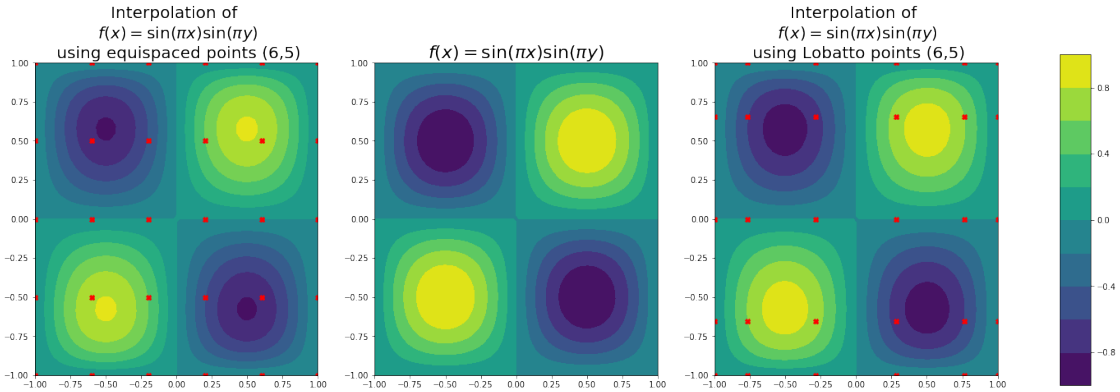
Following are the various basis functions for 4*4 equispaced and lobatto nodes

Basis function when considering a 6x6 grid of equispaced points



Basis function when considering a 6x6 grid of Lobatto points





1.2 Differentiation

1.2.1 1D

Differentiation an interpolated function is extremely simple.

$$f'^{N}(x) = \sum_{i=0}^{N} L_i'^{N}(x) f(x_i)$$

1.2.2 2D

Derivatives in 2D work the same way as in 1D

$$\frac{\partial f^{MN}(x,y)}{\partial x} = \sum_{i=0}^{N}\sum_{j=0}^{M} L_i'^{N}(x_i) * L_j^{M}(y_j) f(x_i, y_j)$$

$$\frac{\partial f^{MN}(x,y)}{\partial y} = \sum_{i=0}^{N}\sum_{j=0}^{M} L_i^{N}(x_i) * L_j'^{M}(y_j) f(x_i, y_j)$$

1.3 Integration

We integrate our functions using gaussian quadrature technique. The integral is converted into a summation over a fixed number of points.

$$\int f(x)dx = \sum w(i) f(x_i)$$

The points that are used for the quadrature can be optimised to give higher accuracy. For instance, choosing the Gauss-Legendre points would let you integrate perfectly over a polynomial of order $2N-1$ where N is the number of points used in the quadrature. Gauss-Lobatto points integrate perfectly over a polynomial of order $2N-3$, but have the added advantage of having the edge points as nodes.

When using the galerkin method, we will be chosing certain nodes to interpolate a function. The values of the function at these points will thus be trivially known. It thus makes sense to choose the interpolating nodes to be the same as the nodes used for integration. We thus use the

Lobatto points for both, interpolation and integration. We need to accurately now the values of a function at boundaries because this will be used heavily when calculating fluxes. Moreover, as seen above, the basis functions behave well when lobatto points are chosen instead of equispaced points.

2 Introduction to Galerkin methods

The core idea of galerkin methods is to take the inner product of the known function space with the governing equation. We thus project the governing equation on our function space. For simplicity consider a just one "element". The function space we use will be the space defined the basis functions $L_{ij}^{NM}(x,y)$. This essentially means that our function space is all functions of the for $f(x)g(y)$ where f and g is an order N and M polynomials respectively. Let our governing equation be

$$\frac{\partial q}{\partial t} = f(q)$$

Inner product in the function space is defined as

$$< f, g >= \int_\Omega f g d\Omega$$

Taking the inner proguct of the governing equation with the function space, we get:

$$\int_\Omega \Psi \frac{\partial q^N}{\partial t} d\Omega = \int_\Omega \Psi f(q) d\Omega$$

where q^N is the discretization of q on out function space.

$$q^N = \sum_j \psi_j q_j$$

Now, the above relation has to hold for every basis function ψ_i int the function space. Thus,

$$\int_\Omega \psi_i \sum_j \psi_j \frac{\partial q_j^N}{\partial t} d\Omega = \int_\Omega \Psi f(q) d\Omega$$

Taking the summation out of the integral and taking $\frac{\partial q_j^N}{\partial t}$ out,

$$\sum_j (\int_\Omega \psi_i \psi_j d\Omega) \frac{\partial q_j^N}{\partial t} = \int_\Omega \Psi f(q) d\Omega$$

This can be reduced to the following expression

$$M \frac{\partial q_j^N}{\partial t} = RHS$$

where M, the "Mass matrix" is defined as

$$M_{ij} = \int_\Omega \psi_i \psi_j d\Omega$$

The integral is calculated using the Quadrature method discussed above. If the interpolating nodes were used as the integration nodes, it would result in an inexact integration since if the number of points is N and the order of the polynomial is $2(N-1)$. Higher order lobatto nodes can be calculated for exact integration. Inexact integration leads to the mass matrix being diagonal (easy to check). This reduces the cost of inverting it. However, this computation has to be done just once for the entire simulation.

The treatment of the RHS depends on what term it is

2.1 Continuous vs. Discontinuous methods

If we are to have a multi-cell system, we can go about it two ways:

2.1.1 Continuous Galerkin

In CG methods, the common boundary of neighboring cells is shared. This essentially means that all our stored variables will be continuous functions. Our function-space will be a continuous function that is a collection of polynomials in every cell.

The problem with this method is that the final matrix vector problem we get will be coupled. This is because the common boundary will occur in the equation of two different cells. Thus, to solve this system, we will need to construct giant matrices for the entire system. We'll have to carry out matrix-vector computations with these large matrices every timestep

2.1.2 Discontinuous Galerkin

An alternative to that is the DG method. The boundaries of each cell are independant of the neighbours. This allows us to do our computations for each cell independently. Information flows between cells because of the flux term (which will be shown in the later formulation). This term, which would have gone zero everywhere but the system boundary for CG is now non zero at the edges of the DG cell. The boundaries of the neighbors will be used when computing the flux term.

One huge advantage of DG methods is the ability to run in parallel. The method scales up beautifully with increasing number of computing nodes. Every node can be assigned a set of elements that it has to bother about. Internode communication will be limited to exchanging the boundary values. Every node can then just perform the matrix-vector computations on its own set of cells

2.2 Difussion

Governing equation: $\frac{\partial T}{\partial t} = \alpha \nabla^2 T$

We break this into two equations:

$$\Theta = \nabla T \frac{\partial T}{\partial t} = \alpha \nabla \cdot \Theta$$

We first solve for Θ

$$\theta_x = \frac{\partial T}{\partial x} \theta_y = \frac{\partial T}{\partial y}$$

Proceeding as earlier,

$$M\Theta^N = \int_\Omega \Psi \nabla T d\Omega \tag{1}$$

$$M_i\Theta^N = \int_\Omega \psi_i \nabla (\sum_j \psi_j T_j) d\Omega \tag{2}$$

$$M_i\Theta^N = \sum_j \int_\Omega \nabla(\psi_j\psi_i) T_j d\Omega - \sum_j \int_\Omega \nabla(\psi_i)\psi_j T_j d\Omega \tag{3}$$

$$M_i\Theta^N = \sum_j ([\psi_i\psi_j]_\Gamma T_j - \int_\Omega \psi_j \nabla \psi_i T_j d\Omega) \tag{4}$$

$$\tag{5}$$

The first term will be evaluated at the boundaries. This can be rewritten as matrix vector equation:

$$M\Theta^N = FT - DT$$

Where F and T are the flux and derivative matrices respectively. F is 0 for points not corresponding to the boundary.

$$F_{ij} = [\psi_i\psi_j]_\Gamma$$

$$D_{ij} = \psi_j \nabla \psi_i$$

Solving the second part:

$$\frac{\partial T}{\partial t} = \alpha \nabla \cdot \Theta \tag{6}$$

$$\int_\Omega \Psi \frac{\partial T}{\partial t} d\Omega = \alpha \int_\Omega \Psi \nabla \cdot \Theta d\Omega \tag{7}$$

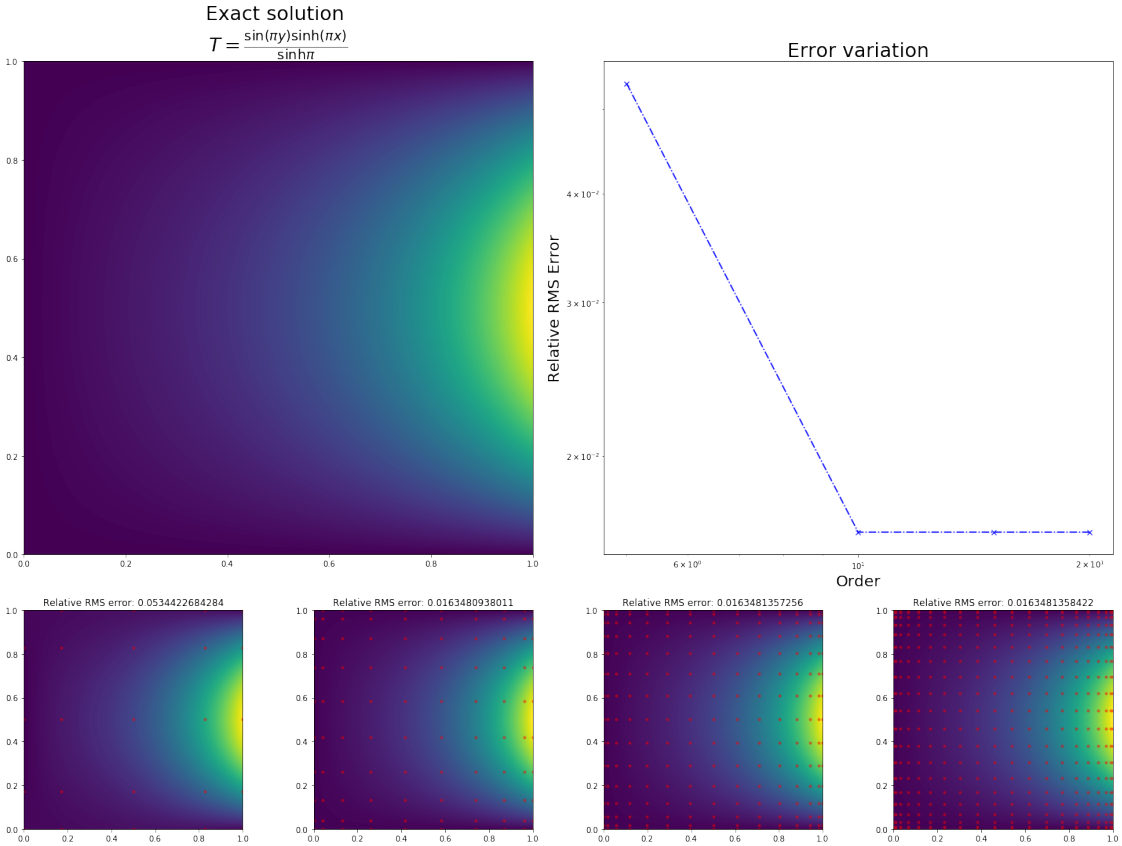
$$M_i \frac{\partial T_j}{\partial t} = \sum_j ([\psi_i\psi_j]_\Gamma \Theta_j - \int_\Omega \psi_j \nabla \psi_i T_j d\Omega) \tag{8}$$

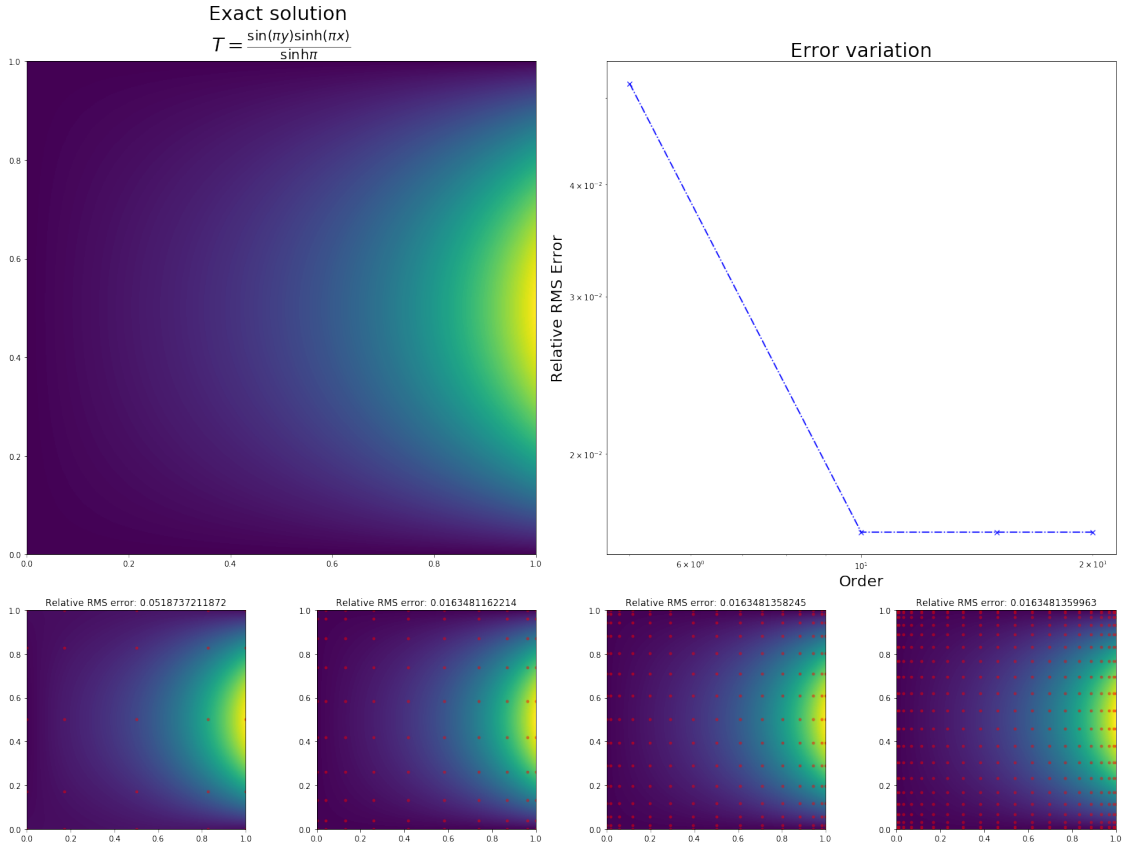
$$\tag{9}$$

2.2.1 Example: Temperature distrigution on a 2D conducting plate with dirichlet boundaries

The right wall is maintained at $T = 1$ and all the other walls are maintained at $T = 0$. The thermal conductivity (α) is set at 0.1. The time step for all of the simulations is 0.1ms and the time horizon is 2sec. The solution is compared to the 2D steady state temperature distribution that can be computed analytically. The order in X and Y is varied and the errors are plotted. The first set of plots are generated using inexact integration and the second set using exact. You can see that the difference in errors is fairly small.

Note: This is a spectral method. There is only one cell.

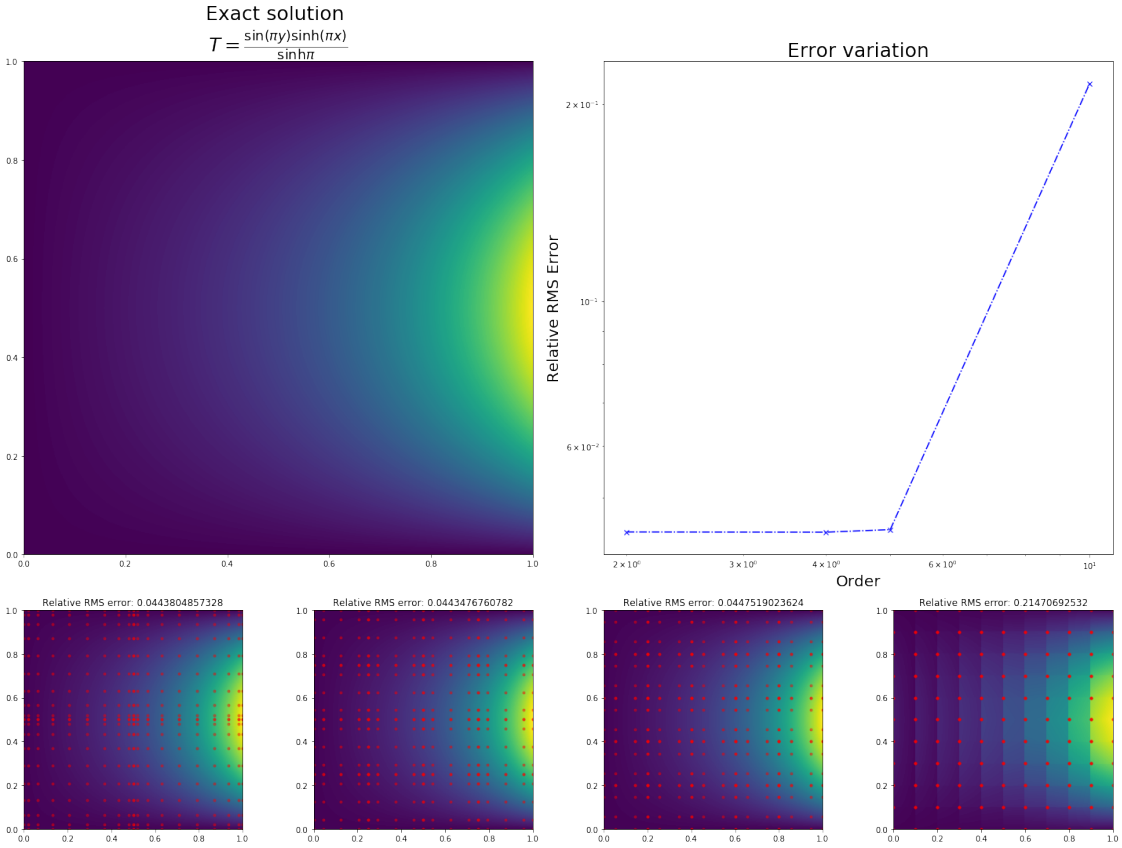

Exact solution
$T = \frac{\sin(\pi y)\sinh(\pi x)}{\sinh \pi}$
Error variation
Relative RMS Error
Order
Relative RMS error: 0.0534422684284
Relative RMS error: 0.0163480938011
Relative RMS error: 0.0163481357256
Relative RMS error: 0.0163481358422



Breaking up the domain into multiple cells is beneficial since it reduces the size of the matrices/increases the number of sampling points. However, higher number of cells also means that you'll have to repeat the matrix vector operations that many times.

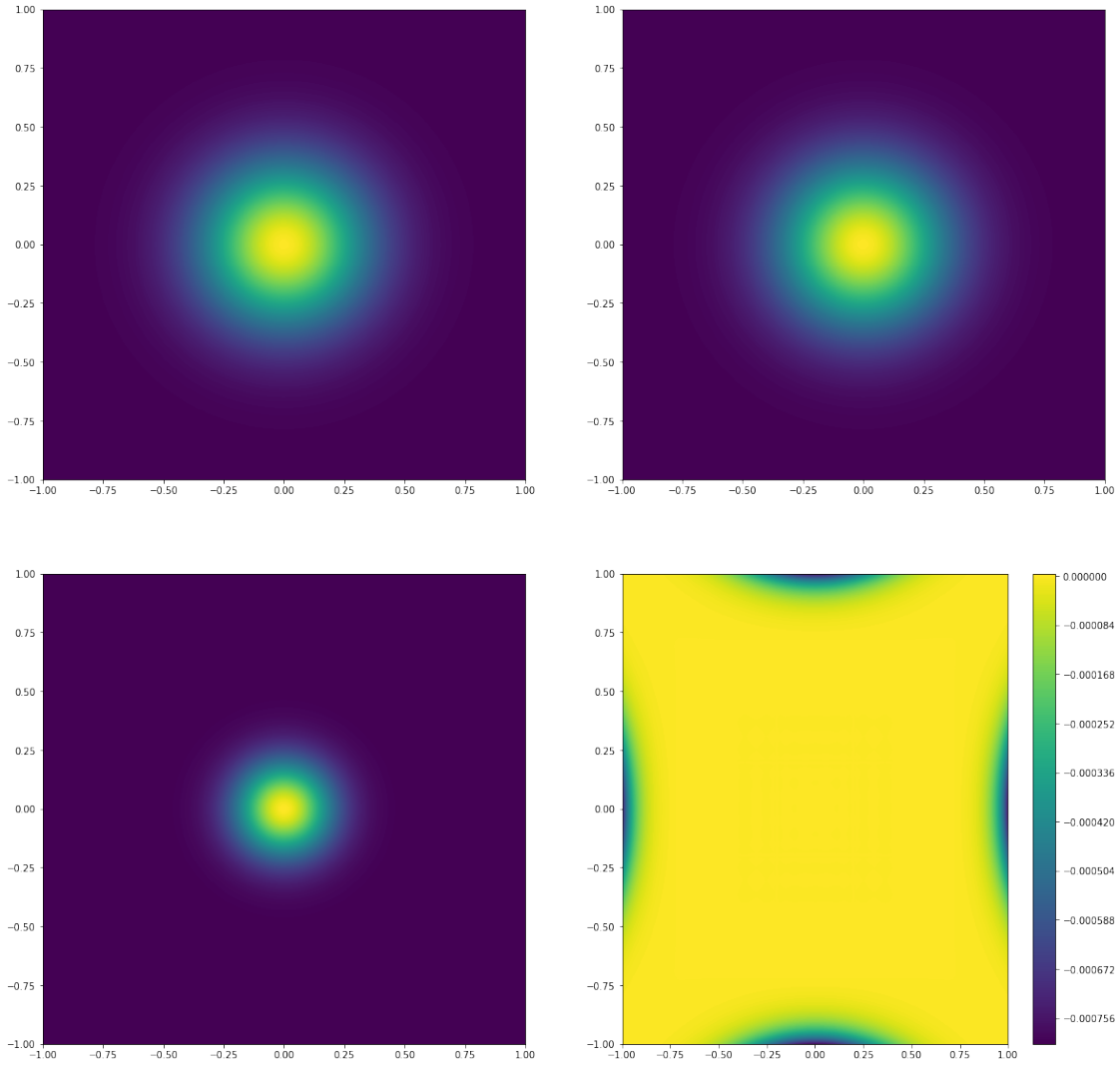
If we were to use a zeroeth order scheme, it would essentially be like a finite volume scheme where every cell has a single value.

We will now run the same exaple but will vary the number of elements too keeping the total number of nodes constant to observe how the error varies. We will just be using inexact integration for this one.



2.2.2 Example: Diffusion of a Gaussian Temperature distribution

A spectral method works poorly for this problem, perhaps because of very low sampling provided by the lobatto points near the center. Note that the lower accuracy near the borders is due to the inaccurate boundary conditions. The analytical solution is for the boundaries being infinitely far away from the gaussian Temperature distribution


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00
0.000000
−0.000084
−0.000168
−0.000252
−0.000336
−0.000420
−0.000504
−0.000588
−0.000672
−0.000756