MA4245 Mathematical Principles of Galerkin Methods
Project 4: 2D Wave Equation

Prof. Frank Giraldo
Department of Applied Mathematics
Naval Postgraduate School
Monterey, CA 93943-5216

Due on Friday June 14 at 12pm

1 Continuous Problem

The governing partial differential equation (PDE) is

$$\frac{\partial q}{\partial t} + \boldsymbol{u} \cdot \boldsymbol{\nabla} q = 0 \qquad \forall (x,y) \in [-1,1]^2$$

where $q = q(x,y,t)$ and $\boldsymbol{u} = \boldsymbol{u}(x,y)$ with $\boldsymbol{u} = (u,v)^T$. Let the velocity field be

$$u(x,y) = y \qquad \text{and} \qquad v(x,y) = -x$$

which forms a velocity field that rotates fluid particles in a clockwise direction. Note that this velocity field is divergence free, that is, that the following condition is satisfied

$$\boldsymbol{\nabla} \cdot \boldsymbol{u} = 0.$$

The reason why this condition is important is that the identity

$$\boldsymbol{\nabla} \cdot (q\boldsymbol{u}) = \boldsymbol{u} \cdot \boldsymbol{\nabla} q + q \boldsymbol{\nabla} \cdot \boldsymbol{u}$$

simplifies to

$$\boldsymbol{\nabla} \cdot (q\boldsymbol{u}) = \boldsymbol{u} \cdot \boldsymbol{\nabla} q$$

which means that we can rewrite the initial problem statement in the conservation form

$$\frac{\partial q}{\partial t} + \boldsymbol{\nabla} \cdot \boldsymbol{f} = 0 \qquad \forall (x,y) \in [-1,1]^2$$

where $\boldsymbol{f} = q\boldsymbol{u}$ is the flux. This form is now ready for use with the DG method.

Clearly, this problem represents a 2D wave equation that is hyperbolic and is thereby an initial value problem that requires an initial condition. Let that initial condition be the Gaussian

$$q(x, y, 0) = e^{-\sigma\left[(x-x_c)^2+(y-y_c)^2\right]}$$

where $(x_c, y_c) = (-0.5, 0)$ is the initial center of the Gaussian and $\sigma = 32$ controls the shape (steepness) of the Gaussian wave. Use periodic boundary conditions for all four sides (i.e., you are solving flow on the surface of a torus; the iperiodic array will take care of this for you if you are using CG, for DG you have to do it via the fluxes).

2 Simulations

Use both CG AND DG methods to solve this equation using the same code base. I recommend you try CG first and then move on to DG. For DG you will need additional maps for the faces (given in the GitHub site). You need to write the code to handle arbitrarily-sized elements and polynomial orders, and use inexact integration formulas. What I mean by arbitrarily-sized elements is that you should not assume that each element is of the same size. You can, however, assume that each element uses the same polynomial and integration orders.

2.1 Skeleton Code Provided

In the GitHub site under Project 4, you will see code to help you get started with the project. The file "A_Main_Driver..." is the driver file. Go there and see where it says "Students Add your Code here". Open up "Construct_RHS_Vector" and see where you need to add code. You need to construct a RHS vector that includes both the contribution from the differentiation matrix and the flux matrix as discussed in the class lectures.

2.2 Streamline

Once your code works, be sure to streamline it in order to make it as fast as possible - this is the fun stuff and take advantage of all you have learned.

2.3 Results You Need to Show

You must show results for $N = 1, 2, 4, 8, 16$ with increasing number of elements $N_e = nel^2$ where nel denotes the number of quadrilaterals in the x and y directions. Plot broken L^2 error norms (defined below) versus number of points (N_P) and show all 5 curves on one plot. Remember that you must use a log plot for the error to capture the spectral convergence.

For the following simulations, you must turn in four plots (one set of four for CG and another set of four for DG): two plots showing convergence rates (for

inexact integration only) and another two plots showing 2-norm errors versus wallclock time. Write a discussion on your findings.

Also, give me a complexity analysis of your code (operation count) to see exactly how many operations your code is performing.

N=1 Simulations For $N = 1$ use $nel = 8, 16, 24, 32, 40, 48$ elements.

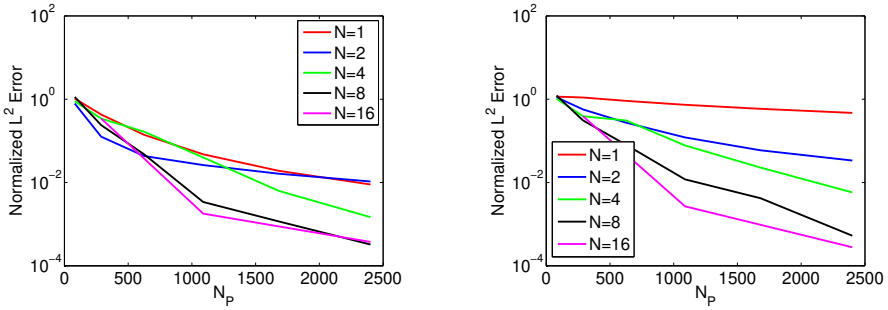
N=2 Simulations For $N = 2$ use $nel = 4, 8, 12, 16, 20, 24$ elements.

N=4 Simulations For $N = 4$ use $nel = 2, 4, 6, 8, 10, 12$ elements.

N=8 Simulations For $N = 8$ use $nel = 1, 2, 3, 4, 5, 6$ elements.

N=16 Simulations For $N = 16$ use $nel = 1, 2, 3$ elements.

Here is an example of the Convergence rates plot you should show me:



a) Exact Integration b) Inexact Integration

Figure 1: The Convergence Rates for CG using a) exact and b) inexact integration using $C = 0.25$ with RK3.

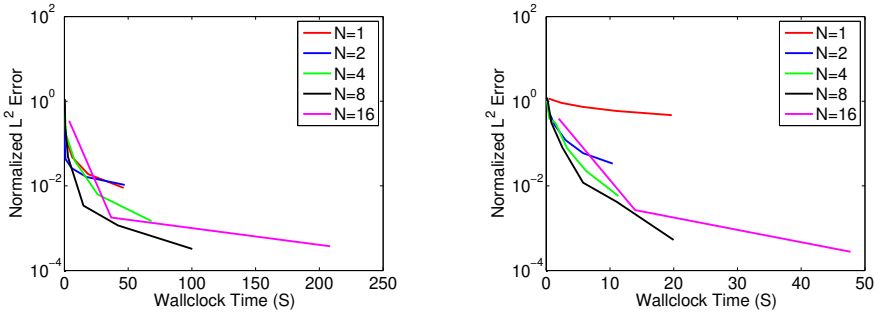
Here is an example of the Computational Cost plot you should show me:

3 Helpful Relations

Make sure that your time-step Δt is small enough to ensure stability. Recall that the Courant number

$$C = u\frac{\Delta t}{\Delta x}$$

must be within a certain value for stability. For the 4th order RK method I gave you, I used $C = 0.5$ for all the simulations; however, for stability purposes, the time-step can be much bigger.



a) Exact Integration b) Inexact Integration

Figure 2: The Computational Cost for CG using a) exact and b) inexact integration using $C = 0.25$ with RK3.