

CSI 701 Computational Project

Introduction

The overall objective of this project is to gain hands-on experience with several of the methods for numerical modeling commonly used in computational sciences, and with the development, implementation and organization of complex simulation codes.

This project involves working with unstructured and structured grids, finite elements and finite differences, elliptic and hyperbolic partial differential equations, particles (ordinary differential equations) and interpolation.

The selected problem is related to the potential flow around a cylinder. More details will be provided in the lectures.

The project is divided into smaller sub-projects or assignments in which new functionality will be incrementally added to your code.

All codes must be written in C, C++, fortran, or fortran90.

Use makefiles to maintain and compile your code.

Although you can use any compiler and operating system, all codes must compile and run under linux in the COS Lab (Research 1 - room 249). On these systems you can use gcc, g++ and gfortran to compile and vi, gvim and emacs to edit.

When coding consider:

- variable and function naming convention

- code efficiency (speed & memory)

- code clarity

- usage: hard coded variables vs command line arguments vs input

- generality: fixed parameters, dynamic memory allocation, etc.

- code organization (subdivide tasks into sections and subroutines, organize functions into files, etc.)

- comments: add comments to make your code more readable, avoid excess comments

Projects are individual.

For each assignment write a brief description (two pages max in PDF) of what you did, how you did it and the results you got. For each assignment, create a single tar.gz file with your code, data and write-up. Name this file name_assignN.tar.gz and upload it to:

<http://www.cos.gmu.edu/~jcebral/academics/csi701/upload.html>

For the visualization of the results of each assignment, any software can be used (matlab, mathematica, code your own functions with OpenGL, etc.). You can also use my own visualization code (zfem) which can be run on the workstations at the COS Lab (room 249). Note that it will not work on other systems or after the semester is over.

On these systems run: /Users/jcebral/bin/zfem

This code reads files in the following (zfem) format (capitals are keywords):

CONTINUOUS	keyword: continuous datasets
dataset-name	data set name (1 word)
POINTS	keyword: begin points section
npoin	nr of points
x1 y1 z1	coords of point 1
...	
xn yn zn	coords of point npoin
REAL	keyword: beginning of field
field-name	field name (1 word)
ndim	nr of field dimensions (1=scalar, 3=vector)
f11 f12 ... f1ndim	field values for point 1
...	
fn1 fn2 ... fnndim	field values for point npoin
ELEMENTS	keyword: beginning of element section
TRIANGLE	keyword: triangular elements
mesh-name	name of element list (1 word)
nelem	nr of elements
a1 b1 c1	nodes of triangle 1
...	
an bn cn	nodes of triangle nelem
END	

There can be any number of fields.

The element nodes start at 1.

There should be no space in front of the keywords.

For a structured grid, replace the TRIANGLE section with the following:

STRUCTURED	keyword: structured grid
mesh-name	name of the mesh (1 word)
nd nx ny nz	nr of dimensions (2), nr of points in x y z (100 100 0)
END	

Assignment 1: computational geometry

Write a code to read an unstructured grid composed of triangular elements, compute the bounding box (min/max coordinates) and the total area of the computational domain.

The mesh is given in separate files containing the point coordinates, the element connectivity, and the lists of boundary points. The file formats are as follows.

filename: pts	point coordinates	
npoin	nr of points	(integer)
x1 y1 z1	coords of point 1	(3 doubles)
x2 y2 z2	coords of point 2	
...		
xn yn zn	coords of point npoin	

filename: ele	element connectivity	
nelem	nr of elements	(integer)
a1 b1 c1	nodes of element 1	(3 integers)
a2 b2 c2	nodes of element 2	
...		
an bn cn	nodes of element nelem	

filename: bp.in / bp.out	bp.in: list of inlet boundary points	bp.out: list of outlet b.p.
nbpoin	nr of boundary points	(integer)
p1	boundary point 1	(integer)
p2	boundary point 2	
...		
pn	boundary point nbpoin	

Notes:

- nodes refer to the three points that compose each element
- all indices start with point 1
- although this is a 2D mesh, the coordinate arrays have 3 dimensions with z=0

Assignment 2: approximation of functions

Add functions to read a scalar field defined on the nodes of the mesh, compute the gradient of this field at the mesh nodes and save the gradient field to a file. Use a finite element approximation to get the field gradient at the nodes. Visualize the results by plotting contours of the scalar field and the gradient magnitude.

The file format for the fields should be:

ndim	nr of dimensions – 3 for a vector field, 1 for scalar fields	(integer)
g1x g1y g1z	gradient components at point 1	(3 doubles)
g2x g2y g2z	gradient components at point 2	
...		
gnx gny gnz	gradient components at point n	

Notes:

-again vectors have 3 components but the z component is zero

Assignment 3: elliptic equations (flow past a cylinder)

Add functions to solve Laplace's equation on the given mesh using a finite element method. For this purpose, write functions to assemble the global matrix and right-hand-side vector, and solve the linear system using the lapack library. Save the resulting field and its gradient (velocity field) to a file. Visualize the results by plotting contour lines of the scalar field and its gradient.

At the inlet / outlet nodes prescribe Dirichlet boundary conditions of the form $\Phi = \Phi_{in}$ and $\Phi = \Phi_{out}$. Use $\Phi_{in} = -1$ and $\Phi_{out} = 1$.

Notes:

-if you call fortran lapack functions from C or vice versa, you need to transpose the matrix.

Assignment 4: interpolation

Add functions to interpolate the velocity field to a structured Cartesian grid covering the same spatial domain containing n_x and n_y points in the x and y directions, respectively. Use $n_x = 100$ $n_y = 100$. Save and visualize the interpolated field.

In order to interpolate a field defined on an unstructured grid to a given position in space, you need to find the triangle that contains this position (i.e. all the shape functions of such element evaluated at the given position should be positive). Once you find this host element, use the shape functions to interpolate the field values to the given position. Repeat this process for all the grid points of the structured grid. For points that fall outside the computational domain set the velocity field to zero.

Assignment 5: hyperbolic equations

Add functions to solve the scalar advection equation on the structured grid using a first order upwind finite differences method. Use the computed/interpolated velocity field to transport a new scalar function. Use zero-flux Neuman boundary conditions in all the boundaries. Compute the allowable timestep from the CFL condition. Initialize the solution with a unit step function between columns 10 and 20 of the grid. Advance the solution in time until this initial profile exits the computational domain. Visualize the evolution of the scalar field as it is transported.

Assignment 6: particles and ordinary differential equations

Add functions to advect particles with the velocity field defined on the unstructured grid. Use the interpolation function to find the velocity of the particles at their current position and integrate forward in time using either a simple Euler scheme or a Runge-Kutta algorithm. Calculate the local timestep for each particles from the local velocity and size of the host triangular element. Create 1000 particles randomly distributed in the region used to initialize the advection equation of assignment 5. Visualize the particles as they move through the computational domain.