Comparison of Triangular, Cell-Centered Strand, and Node-Centered Strand Grid methods

Aaron Katz*, Shaun Harris†, Dalon Work‡, Oisin Tong§, and Jon Thorne¶
Department of Mechanical and Aerospace Engineering, Utah State University, Logan, UT 84322

I. Introduction

Three separate algorithms for computing a flat plate and a bump case are shown and compared. A triangular grid case with flux correction is explained and a flat plate case and bump case are analyzed. A cell-centered strand grid and a node-centered strand grid are also explained, and similar cases are run and analyzed. The node-centered strand grid was also run at transonic speeds with a limiter inclusion on a bump case. The resulting shock is shown and analyzed.

II. Numerical Methods

A. Triangular Grid

Shauns stuff that talks about Figure 1

B. Cell-Centered Grid

Shauns stuff [3]

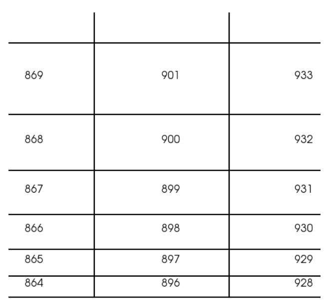

Figure 1. Cell-Centered Grid configuration displaying identification numbers for the cells.

C. Node-Centered Grid

stuff and talk about Figure 2

1. Limiter Inclusion

Stuff again, discuss equations related to the limiter. Reference the appendix V

*Assistant Professor, AIAA Member
†Undergraduate Student, AIAA Student Member
‡PhD Candidate, AIAA Student Member
§PHD Candidate, AIAA Student Member
¶Masters Student, AIAA Student Member

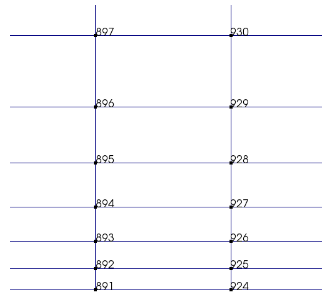

Figure 2. Node-Centered Grid configuration displaying identification numbers for the nodes.

III. Results

We will compare the results from the plate and bump cases from each of the grid codes.

A. Flat Plate

comparison of the flat plate stuff against blasius solutions

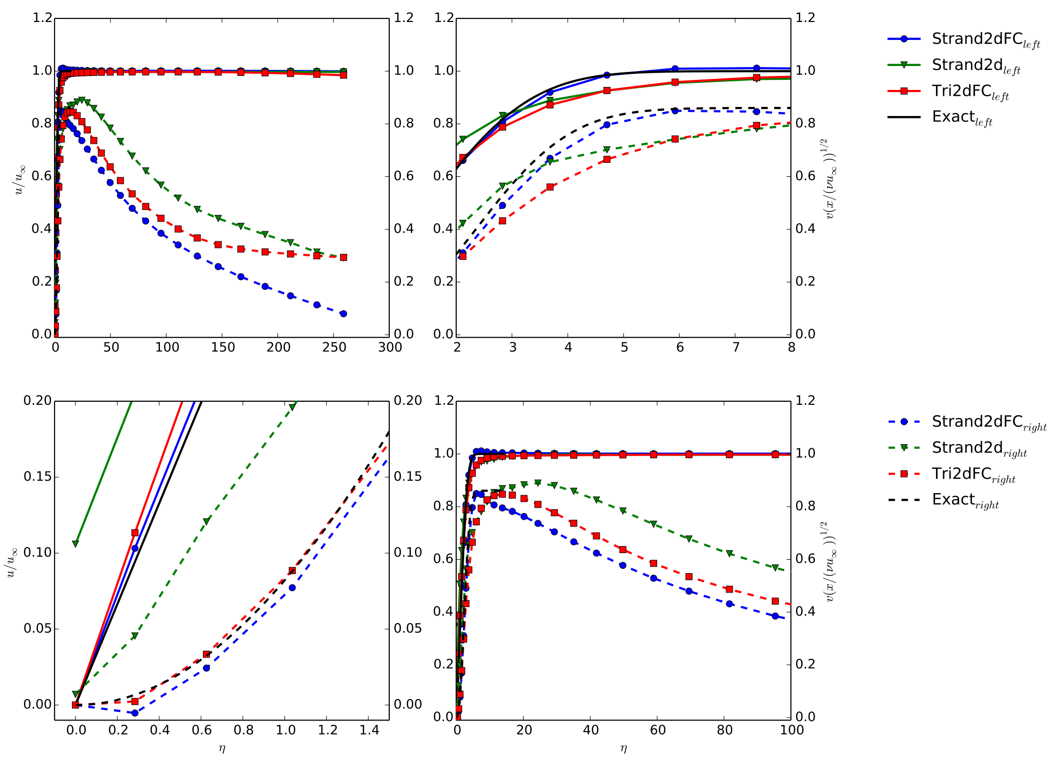

Figure 3. Blasius Solution comparison of all three grids

B. Bump

The general bump is outlined by NASA stuff

1. Subsonic

This is what it looks like

2. Transonic Inviscid with Limiter Inclusion

and what it looks like when you run it a bit faster inviscid

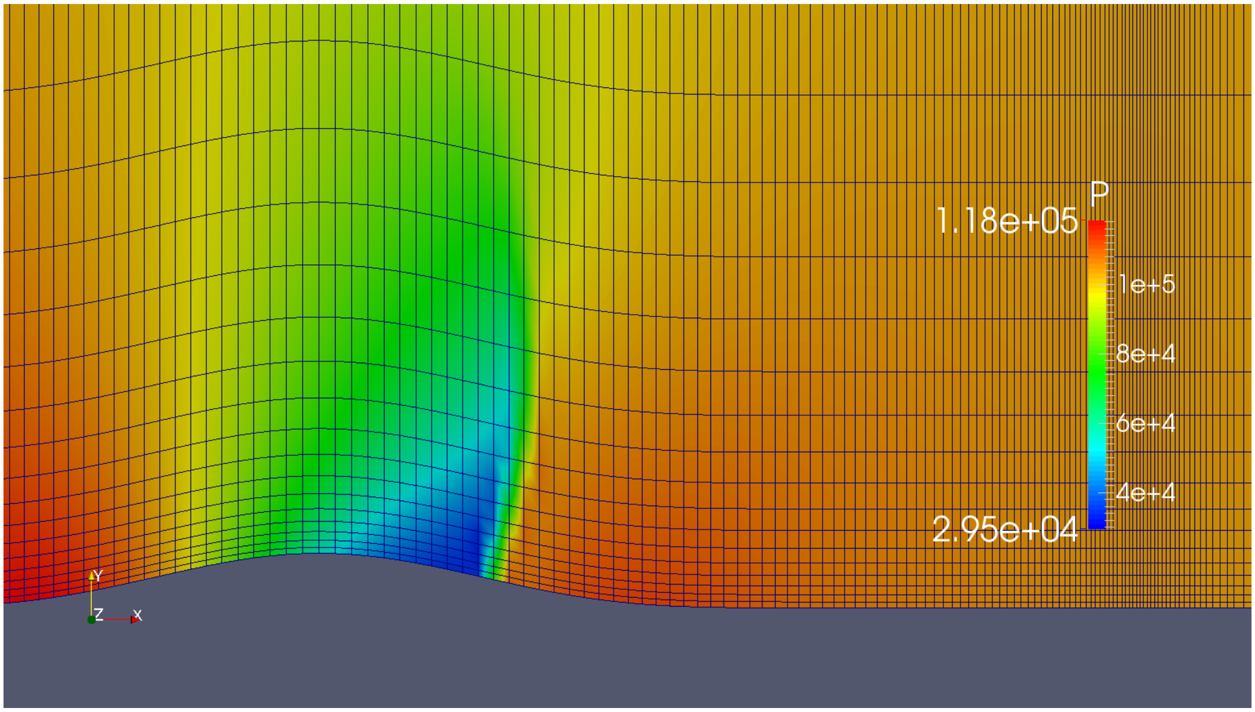

Figure 4. Invscid transonic bump case at Mach 0.8

IV. Conclusions

Strand2dFC is awesome... not only can it do 2,2 scheme, but it can also do 3,4 scheme! Cool!

References

[1] Barth, T. J., “Numerical Aspects of Computing Viscous High Reynolds Number Flows on Unstructured Meshes,” *American Institute of Aeronautics and Astronautics*, 1991.

[2] Diskin, B. and Thomas, J., “Accuracy of Gradient Reconstruction on Grids with High Aspect Ratio,” Tech. rep., National Institute of Aerospace, December 2008.

[3] Barth, T. J., Barth, T. J., and Linton, S. W., “An Unstructured Mesh Newton Solver for Compressible Fluid Flow and Its Parallel Implementation,” *American Institute of Aeronautics and Astronautics*, 1995.

V. Appendix

Limit.C file is displayed below.

```
#include "Strand2dFCBlockSolver.h"


void Strand2dFCBlockSolver::limit(const int& j)
{
/* this calculates the limiter value for Strand2dFC
 * written by Shaun Harris at Utah State University
 * email: shaun.r.harris@gmail.com
 *  Sept. 16, 2014
 *
 *
 *
 */

// standard limiter=0 from input.namelist
        if (limiter == 0) {
                for(int n=0; n<nSurfNode; n++){
                        for (int k=0; k<nq; k++) {
                                lim(n,j,k) = 1.;
                        }
                }
        }

// limiter to be used limiter=1 from input.namelist
        else if (limiter == 1){ //calculate using gradient
                double a;
                double b;
                int n0;
                int n1;
                double limE;
                //initially set all lim values of j to 1
                for(int n=0; n<nSurfNode; n++){
                        for (int k=0; k<nq; k++) {
                                lim(n,j,k) = 1.;}}
                //find limiter value at each surfEdge
                for(int n=0; n<nSurfEdge; n++){
                    for (int k=0; k<nq; k++) {
                                n0=surfEdge(n,0);
                                n1=surfEdge(n,1);
                                a=deltaS*qx(n1,j,k,0);
                                b=deltaS*qx(n0,j,k,0);
                                //~ a = q(surfEdge(n,3),j,k)-q(n1,j,k); // original a
                                    and b, can use this instead of a and b above
                                //~ b = q(n0,j,k)-q(surfEdge(n,2),j,k);
                                limE = 1.-((fabs((a-b)/max((fabs(a)+fabs(b)),dlim(k)))
                                    )*
                                (fabs((a-b)/max((fabs(a)+fabs(b)),dlim(k))))*
                                (fabs((a-b)/max((fabs(a)+fabs(b)),dlim(k)))));
                                //output warning and exit if bad limiter
                                if (limE>1 | limE<0){
                                    cout<<"WARNING: Bad Limiter value"<<endl;
                                    cout<<"limE("<<n<<","<<j<<","<<k<<") = "<<limE<<" 
                                        "<<"limiter "<<limiter<<endl;
                                    exit(0);
                                        }
                                //convert limiter value from Edge to Node using the
                                    minimum of 2 neighboring edges of each node
                                lim(n0,j,k)=min(lim(n0,j,k),limE);
```

```
                        lim(n1,j,k)=min(lim(n1,j,k),limE);
                        //~ cout<<"lim("<<n0<<","<<j<<","<<k<<")= "<<lim(n0,j,
                            k)<<endl;
                        //~ cout<<"lim("<<n1<<","<<j<<","<<k<<")= "<<lim(n1,j,
                            k)<<endl<<endl;
                    }
                }
        }
        else {
                cout << "WARNING: input value for limiter is incorrect format  Limiter
                    = "<<limiter<<endl;
                exit(0);
        }

}
```