SAND2014-18010PE

Replacing Discontinuous Functions in Limiters with Smooth Ones

Bill Rider, Ed Love, and Tim Wildey

Sandia National Laboratories, Albuquerque

“Study the past if you would define the future.”
— Confucius

Narrative

There is a big difference between writing down an algorithm in a mathematical form, implementing simply, and implementing for robust use in a production environment. Along this path, the actual algorithm gets messier, and more tricks are applied to make things work. Limiters are an essential aspect of solving the hyperbolic portion of systems of differential equations. Most limiters use discontinuous functions in their implementation, which can cause subtle, but significant difficulties for their codes. These issues may arise in testing for verification, use in nonlinear solves, symmetry preservation, and increased sensitivity to small perturbations. We explore the removal of these discontinuities through replacing key functions with continuously differentiable forms. Care must be taken in determining the proper form for these functions as not to disturb the fundamental properties of the limiters and the methods they serve. In the same vein, similar functions are used pervausively in Riemann solvers and other dissipation mechanisms such as artificial viscosity.

First, let us provide simple, yet explanatory example. The sign function is commonly used in both limiters and dissipation, and is discontinuous at zero. Several functions can serve to provide the same asymptotic behavior as the sign function while provide a continuously differentiable functions such as tanh(a x) or erf(a x). The argument in a in tanh(a x) or erf(a x) can adjust the magnitude of the smoothing. Another closely related example of a common function that can cause problems would be the absolute value function near zero. If we replace the standard absolute value function with the hyperbolic tangent multiplied by its argument, we receive the same behavior as quantity examined is significantly away from zero. Another complementary definition of the absolution value is the anti-derivative of the sign function. For the continuously differentiable versions of the sign function, the complementary absolute value functions differ in definition. Most of the rest of our effort will focus on parameterizing the smoothing through the selection of the parameter “a”. In a nutshell we make the choice using two principles, the product “a x” should be non-dimensional, and the magnitude of the free constant is chosen to be large enough that the function does not deviate from the original sign function we are replacing.

Narrative

Recently, a blog point hit upon this issue as related to statistical applications, the Endeavor, by John Cook [1,2]. The stated purpose in that context was optimization where smoothness of the functional representation would impact the convergence of solutions. This is not unlike the reasons for our use of similar functions. As we will demonstrate there are several ways to achieve the same end product and principles that ca be used to determine the proper level of smoothing. This resonated with us because as described above we regularly use techniques like this in code development to remove non-differentiable behavior from the code. We do this for many reasons that span issues from software development to numerical performance. Our earliest use was to change the sign function near zero as not to perturb symmetries in fluid dynamics problems. Likewise, this behavior can make it difficult to pass regression tests for a large code that I develop. A deeper issue to discuss is why I would do such things, robustness of results

As noted above, one form of robustness is removing different behavior under common variations in computing environment. These variations include different compilers, or compiler options and different computers. Almost any significant code development activity uses regression testing to maintain quality and stability within the code. Often the regression test simply looks for a difference in a solution over time. These differences are used to flag changes in the code that are unintentional and potentially associated with the introduction of a bug.

Limiters and other dissipation mechanisms have consistency conditions to meet in order to provide proper approximations to the differential equations. These conditions can be used to select and constrain the smoothing parameterization used. Once the smooth functions are selected we can demonstrate the utility of this approach and the lack of negative consequences from the change away from classical functions.

Narrative

Most limiters and dissipation mechanisms use a combination of the sign, absolution value, min and max functions all of which are discontinuous and can be replaced. A prime example is the "minmod" limiter, which uses all four, minmod(x,y) = sign(x) max(0,min(abs(x),sign(x)y)). One key property that results from approximation accuracy is minmod(1,1) = 1 [Sweby]. The ability to achieve this is related to the form of the smoothing function. For example Cook's min/max pair does not provide this, but an alternative min/max construction does trivially, min(x,y) = ½(x+y)-abs(x-y), max(x,y) = ½(x+y)+abs(x-y). Unfortunately this produces a non-convex min/max. In the convex case the ability to achieve accuracy is more subtle, and connected to the smoothing parameter. A Taylor series analysis can produce the impact and connect the form of the smoothing parameter to the ability to produce approximation of the requisite accuracy.

When tested with limiters we find that the smooth functions have little problem reproducing the results of the discontinuous functions with a sufficiently large regularization parameter. In addition, the regularized functions produce recognizable benefits with the advection of smooth waveforms with tangible reductions in the level of error. Overall we believe that this approach may produce a number of real improvements in codes relying on limiters.

REFERENCES

John D. Cook, The Endeavor, http://www.johndcook.com/blog/. "Soft Maximum," http://www.johndcook.com/blog/2010/01/13/soft-maximum/, 2010.

John D. Cook, The Endeavor, http://www.johndcook.com/blog/. "How to compute the soft maximum," http://www.johndcook.com/blog/2010/01/20/how-to-compute-the-soft-maximum/, 2010.

John D. Cook, "Basic properties of the soft maximum," UT MD Anderson Cancer Center Department of Biostatistics Working Paper Series. Working Paper 70, 2011.

Peter Sweby, "High Resolution Schemes Using Flux Limiters for Hyperbolic Conservation Law " SIAM Journal of Numerical Analysis, Vol. 21(5), pp. 995-1011, 1984.

"If the answer is highly sensitive to perturbations, you have probably asked the wrong question."- Nick Trefethen

Intrinsic functions are everywhere in code and implement many algorithms

- We use built in intrinsic functions without a second thought such as sin(x), cos(y), exp(z)
- What about min(a,b), max(b,c), abs(d), sign(e)?
- Each of the second set of functions is discontinuous in some way.

Discontinuous functions can create problems in calculations worth relieving ourselves of.

- In getting verification results the impact can be seen when errors become small or parts of the problem are near zero
- These effects can break symmetry, seen in simple Rayleigh-Taylor tests (single mode to late time)
- Getting clean regression tests across multiple platforms (especially if you have a lot of tests)

Branching is a key example where this plays out

- One can implement certain functions via functions (like) max as branching tests (i.e., if tests)
- Consider an if test version of upwind differencing,

```
if (velocity > 0) then
    upwind=data_left
else
    upwind=data_right
endif
```

Branching is a key example where this plays out (continued)

- That is terrible, let's change it to something better

```
if (velocity > 0) then
    upwind=data_left
else if (velocity < 0) then
    upwind=data_right
else
    upwind=0.5*(data_left+data_right)
endif
```

We can improve this to make it resilient to roundoff (continued)

- That is terrible, let's change it to something better

```
if (velocity > small_velocity) then
    upwind=data_left
else if (velocity < small_velocity) then
    upwind=data_right
else
    upwind=0.5*(data_left+data_right)
endif
```

This branching can be better written functionally

- One can use the sign function or the absolute value to achieve this without the if statements

sign_velocity = sign(velocity)

upwind=0.5*(sign_velocity +1.0)*data_left+
(sign_velocity -1.0)*data_right

The functional branching is discontinuous, so we are back to square one

- Here is the key idea, rewrite the sign as a continuous function, the hyperbolic tangent comes in handy

,ollified_sign=tanh(velocity/small_velocity)

sign_velocity = mollified_sign(velocity)

upwind=0.5*(sign_velocity +1.0)*data_left+
(sign_velocity -1.0)*data_right

“Algorithms don't do a good job of detecting their own flaws.”–Clay Shirky

Many intrinsic functions can be rewritten to allow for analysis

- This was originally done to allow the analysis of numerical methods using the method of modified equations.
- This made the method differentiable and amenable to Taylor series expansions.
- These functions are fundamental to limiters, artificial viscosity or even ad hoc tests for positivity

In the same manner modified equation analysis can apply to nonlinear schemes.

- To really work with this some intrinsic functions that are used greatly need to be redefined.
- Minmod is key to limiters introduced by Boris for FCT, returns the smallest argument in absolute value if they have the same sign, zero otherwise

$$\min(a,b) = \frac{1}{2}(a+b) - \frac{1}{2}|a-b| \qquad |a| = \sqrt{a^2}; \operatorname{sgn}(a) = |a|/a$$

$$\max(a,b) = \frac{1}{2}(a+b) + \frac{1}{2}|a-b| \qquad \operatorname{minmod}(a,b) = \operatorname{sgn}(a)\max\left[0, \min\left(|a|, \operatorname{sgn}(a)b\right)\right]$$

$$\operatorname{min\,mod}(a,b) = \frac{1}{4}\left(\operatorname{sign}(a) + \operatorname{sign}(b)\right)\left(|a+b| - |a-b|\right)$$

$$\operatorname{min\,eno}(a,b) = \frac{1}{2}\left(\operatorname{sign}(a+b)\right)\left(|a+b| - |a-b|\right)$$

Following this same path we can rewrite the same intrinsic functions for smoothness

- We can do the same sort of thing if our objective is smoothness
- We already made the first substitution, the tanh for the sign.

mollified_abs(x)=x * tanh(x/small_x)

Examples of the rewrite.

- Once the sign and absolute value is defined we can build other functions like min, max, minmod, mineno, etc…
- The median(a,b,c) is a useful function that returns the argument bounded by the other two

median(a,b,c) =a+minmod(b-a,c-a)

Some functions can then be composed to provide the basis of some algorithms

- We could also base some of the algorithms on the softmin, softmax ideas of Cook and extending them

softmax(a,b)=log(exp(n*a)+exp(n*b))/n

- We can rewrite this for better behavior in finite arithmetic,

softmax(a,b)=max(a,b)+log(1+exp(n*min(a,b)-n*max(a,b)))/n

softmax(a,b)=min(a,b)-log(1+exp(n*min(a,b)-n*max(a,b)))/n

softabs(a)=softmax(0,a)-softmin(a,0)

Softsign(a)=a/softabs(a)

The Basic Idea of High Resolution Methods

- Make the methods nonlinear - *adaptive stencil*, decide how to blend 2 or more methods based on the local solution
  - Upwind, Lax-Wendroff and 2nd-order upwind using a TVD limiter

$$u_t + u_x = 0$$

$$u_j^{n+1} = u_j^n - \lambda\left(u_j^n - u_{j-1}^n\right) - \frac{\lambda}{2}(1-\lambda)\left[\phi_j\left(u_{j+1}^n - u_j^n, u_j^n - u_{j-1}^n\right) - \phi_{j-1}\left(u_j^n - u_{j-1}^n, u_{j-1}^n - u_{j-2}^n\right)\right]$$

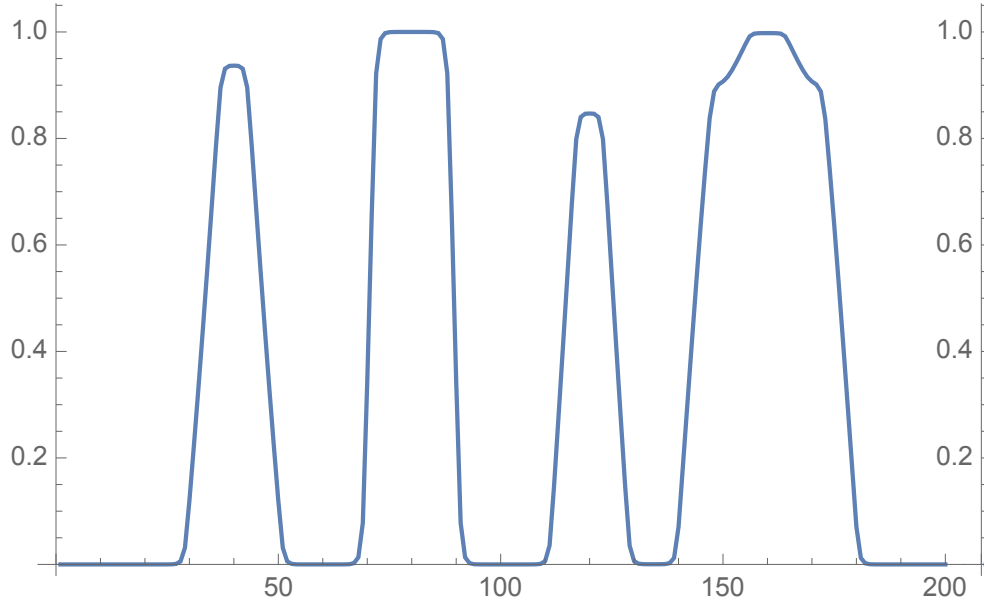
$$\lambda = \Delta t/\Delta x$$

$$\phi(a,b) = \max(0, \min(a,b))$$

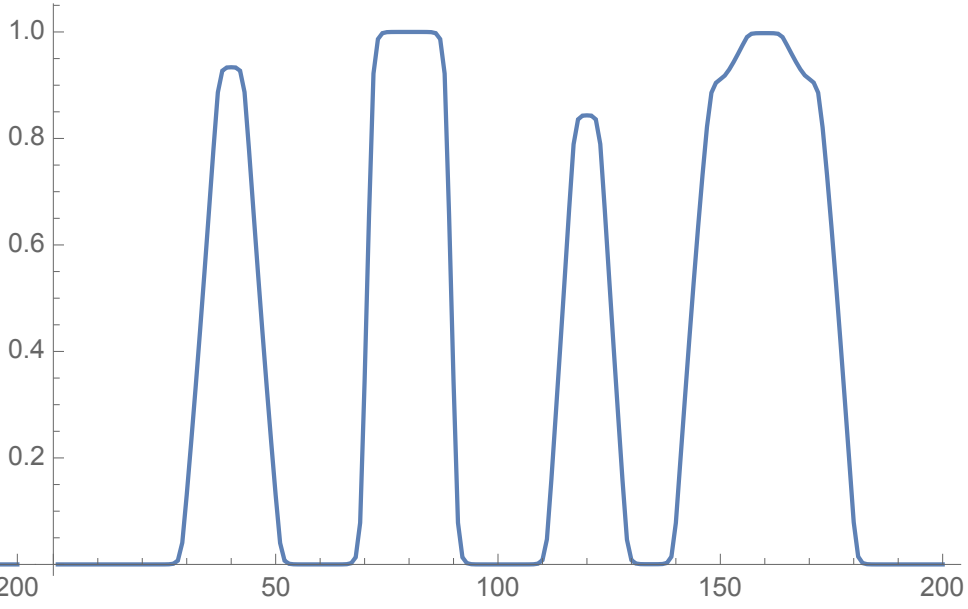
We can test these ideas in a relatively simple setting.

- Use these functions to implement a simple test version of a high resolution scheme.

Standard



Mollified (small=1/20)



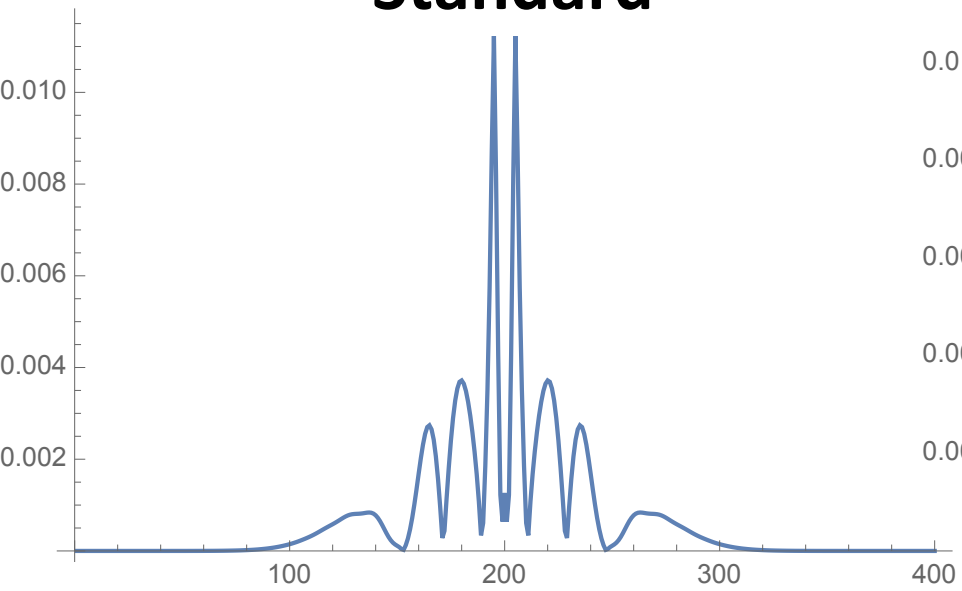
We can test these ideas in a relatively simple setting.

- The mollified functions have a positive influence on accuracy in the case of smooth initial conditions.

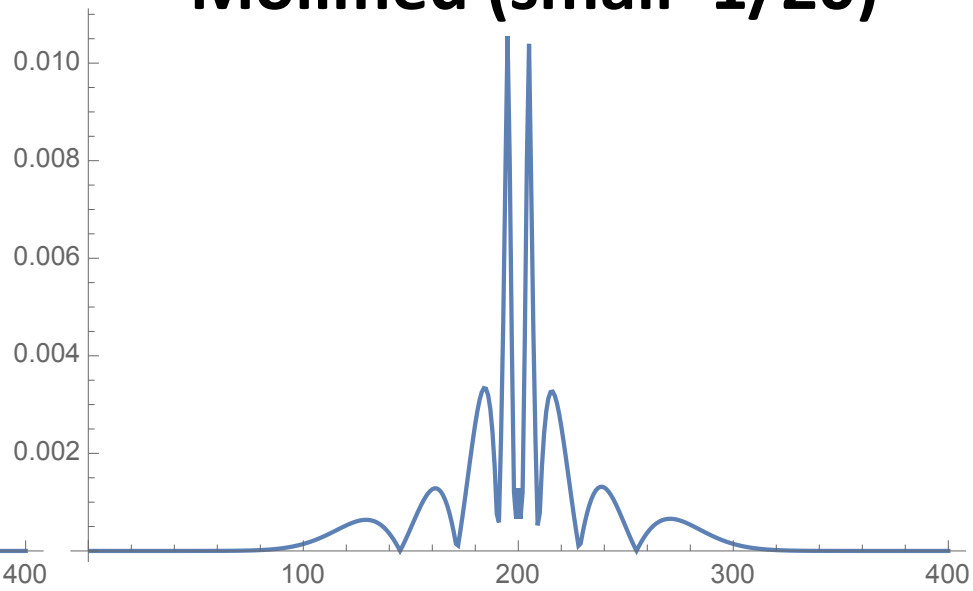
Cells	Error	Order
100	5.79	
200	2.36	1.29
400	0.73	1.69

Cells	Error	Order
100	5.93	
200	2.13	1.48
400	0.60	1.82

Standard



Mollified (small=1/20)



This can impact the verification testing of the method

- Hyperviscosity example using the mollified sign function to turn the viscosity off in expansion.

Table 1
A summary of the code verification results using the periodic breaking wave problem's density solution for four different artificial viscosity options. Without viscosity, with the limiter, and with the limiter and hyperviscosity in tandem produce second-order results. With the standard artificial viscosity alone, the results become first-order accurate.

Method	h	L_1 Error	Order
No viscosity	0.00078	1.442×10^{-9}	1.999
$c_1 = 0, c_2 = 0, c_3 = 0$	0.00156	5.763×10^{-9}	2.001
	0.00312	2.307×10^{-8}	2.009
	0.00625	9.286×10^{-8}	
Standard viscosity	0.00078	3.934×10^{-7}	0.968
$c_1 = 1, c_2 = 4/3, c_3 = 0$	0.00156	7.696×10^{-7}	0.945
	0.00312	1.482×10^{-6}	0.911
	0.00625	2.786×10^{-6}	
Limited viscosity	0.00078	1.987×10^{-9}	2.008
$c_1 = 1, c_2 = 4/3, c_3 = 0$	0.00156	7.994×10^{-9}	1.994
	0.00312	3.182×10^{-8}	1.992
	0.00625	1.264×10^{-7}	
Limited viscosity with hyperviscosity	0.00078	1.661×10^{-9}	2.016
	0.00156	6.717×10^{-9}	2.018
$c_1 = 1, c_2 = 4/3, c_3 = 1$	0.00312	2.721×10^{-8}	2.030
	0.00625	1.112×10^{-7}	

Required replacing sign(Div u) with tanh(h Div u/c)

Summary

- The basic idea is to replace functions with sharp (discontinuous) changes in value with continuous functions having controllable smoothing length.
- This will allow codes to be differentiable and remove extreme sensitivity to small changes in floating point values (or roundoff)
- This choice can help a number of areas including advanced numerical methods, regression, symmetry preservation and…

“Predictability is not how things will go, but how they can go.”
— Raheel Farooq