My dg adventures

So far everything sucks.

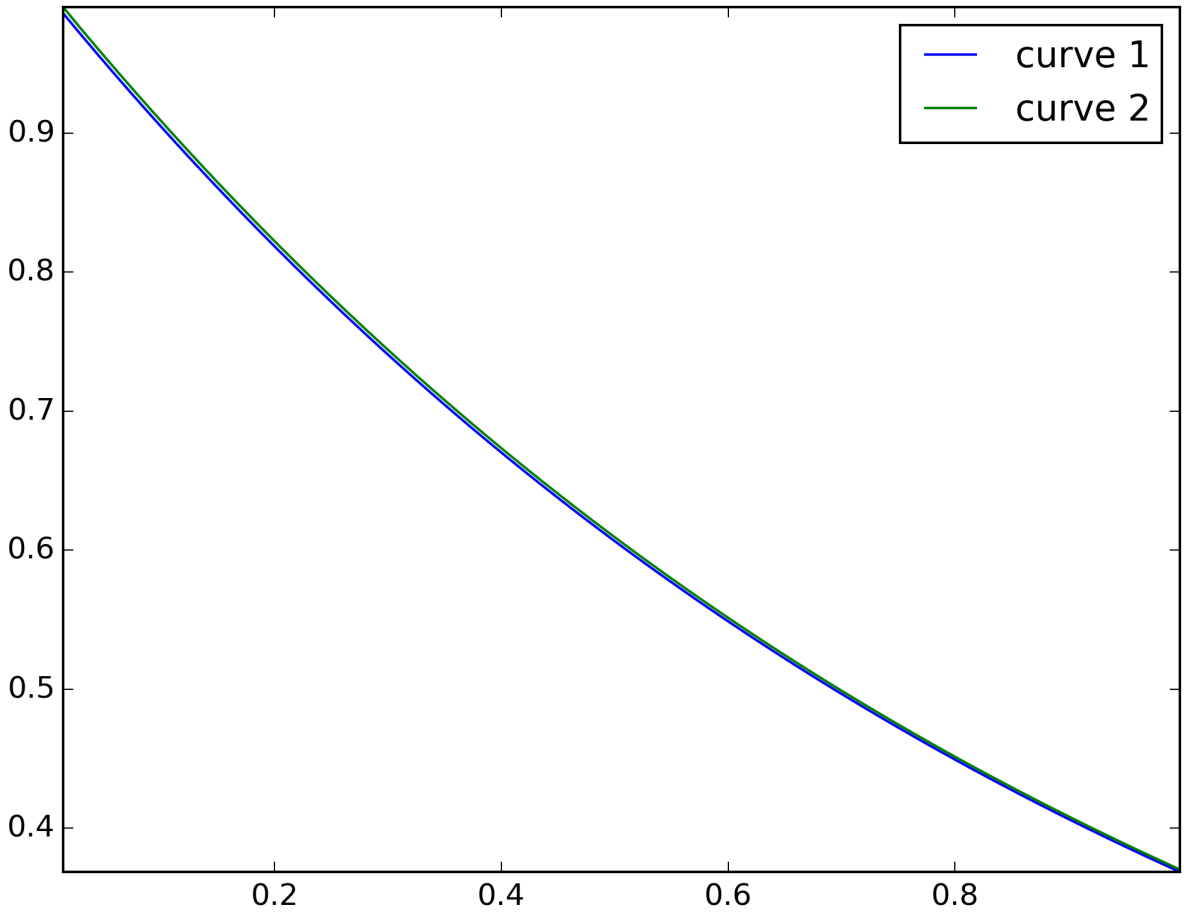
Update: There's light at the end.

Hydrostatic case

I.c.s:

$$\begin{aligned} \rho_0 &= exp(-x) \\ u_0 &= 0. \\ p_0 &= exp(-x) \end{aligned}$$

N = 64, T=100

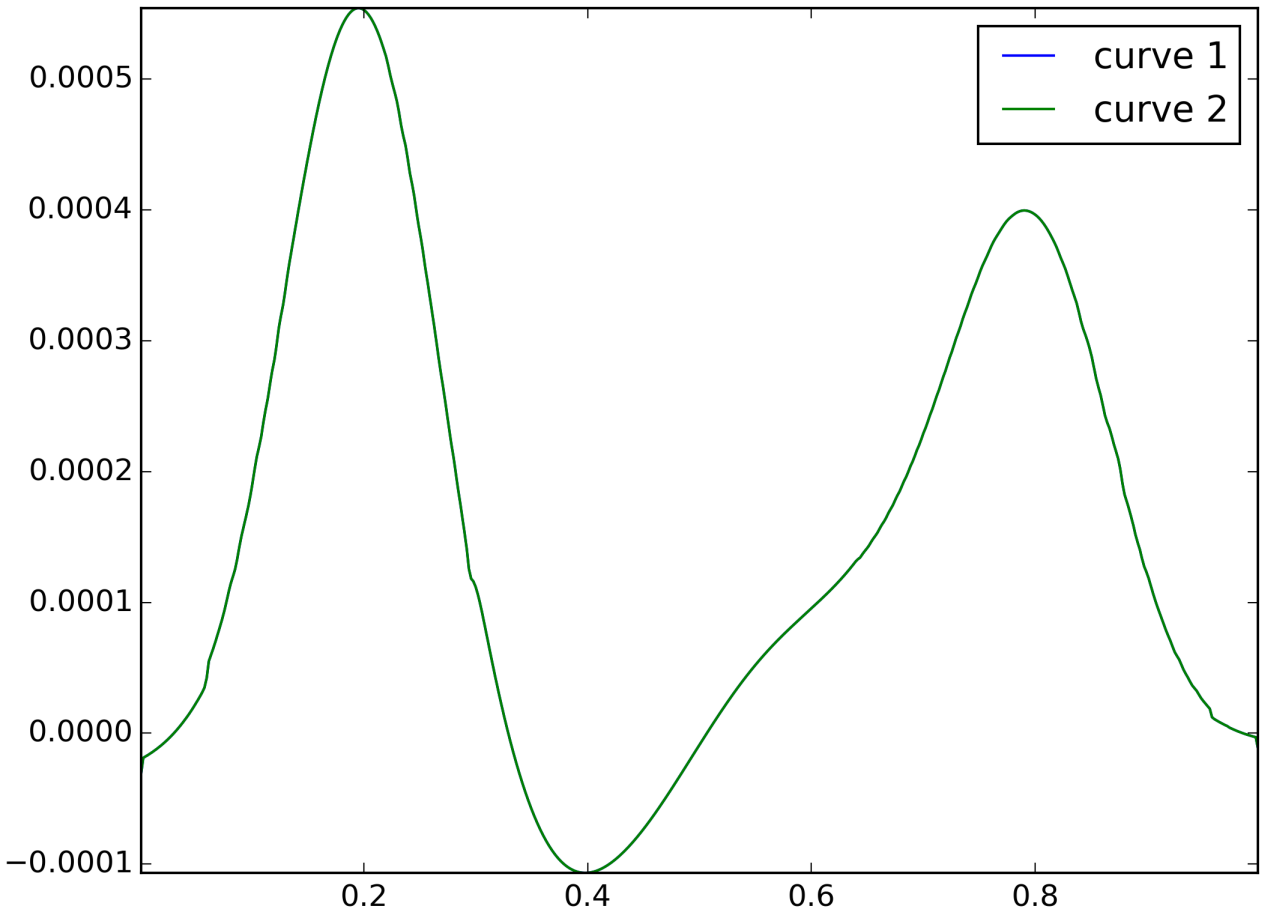


hydrostatic case + perturbation

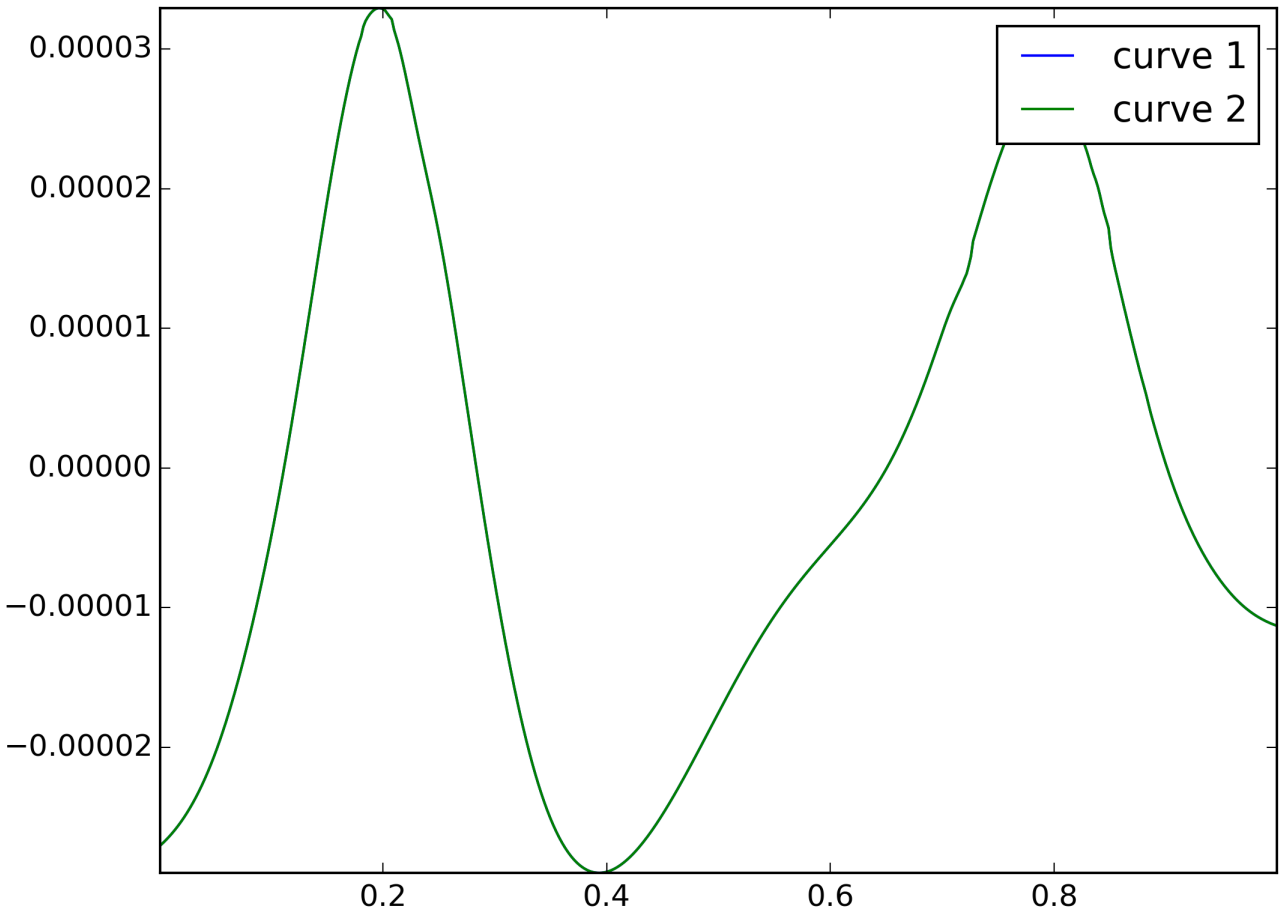
$$
\begin{aligned}
\rho_0 &= exp(-x) \\
u_0 &= 0. \\
arep_0 &= exp(-x) + \alpha * exp(-100 * (x - x_c)^2)
\end{aligned}
$$

$$x_c = \frac{l}{2}$$

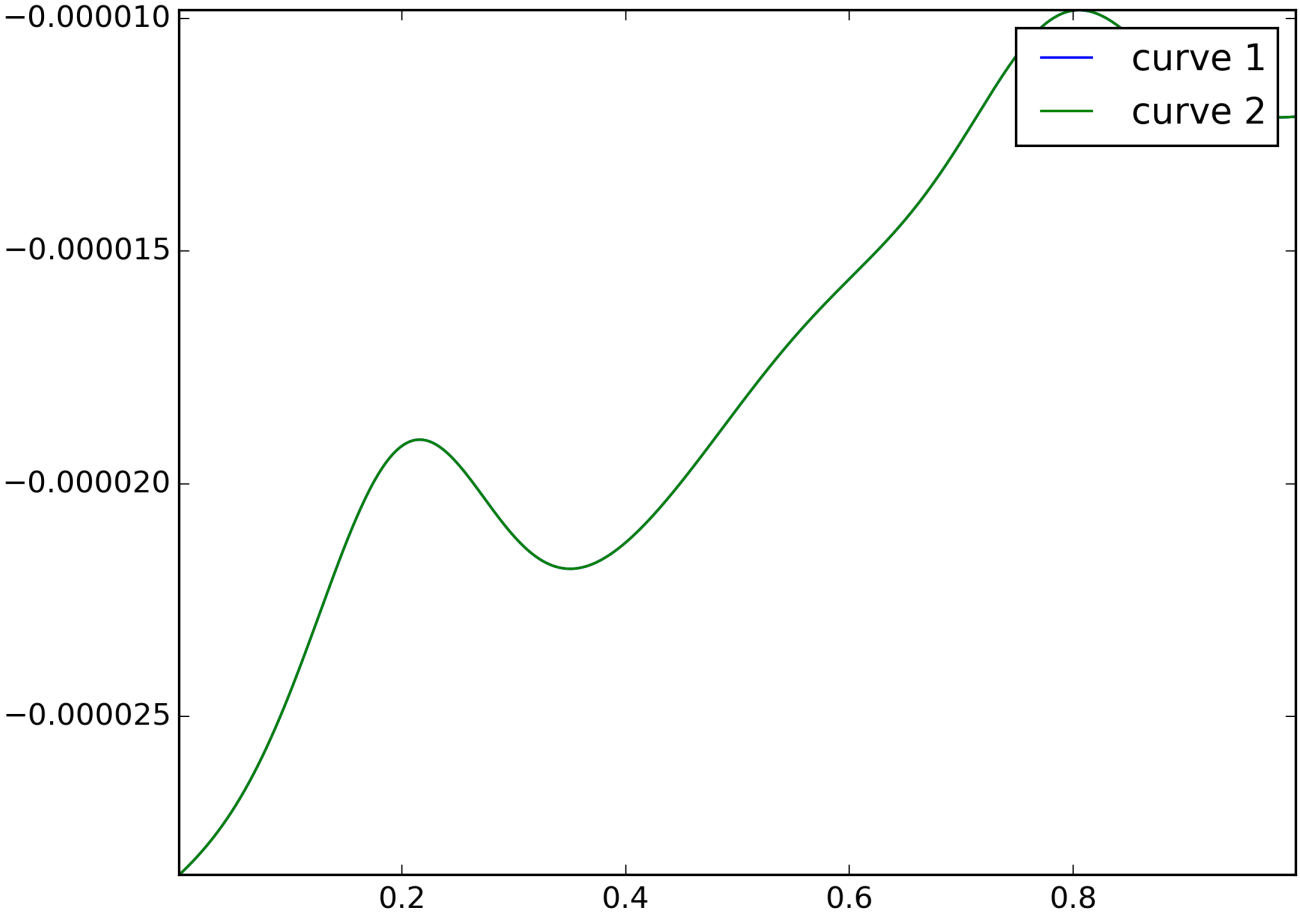
$$\alpha = 0.001$$



$$\alpha = 0.0001$$



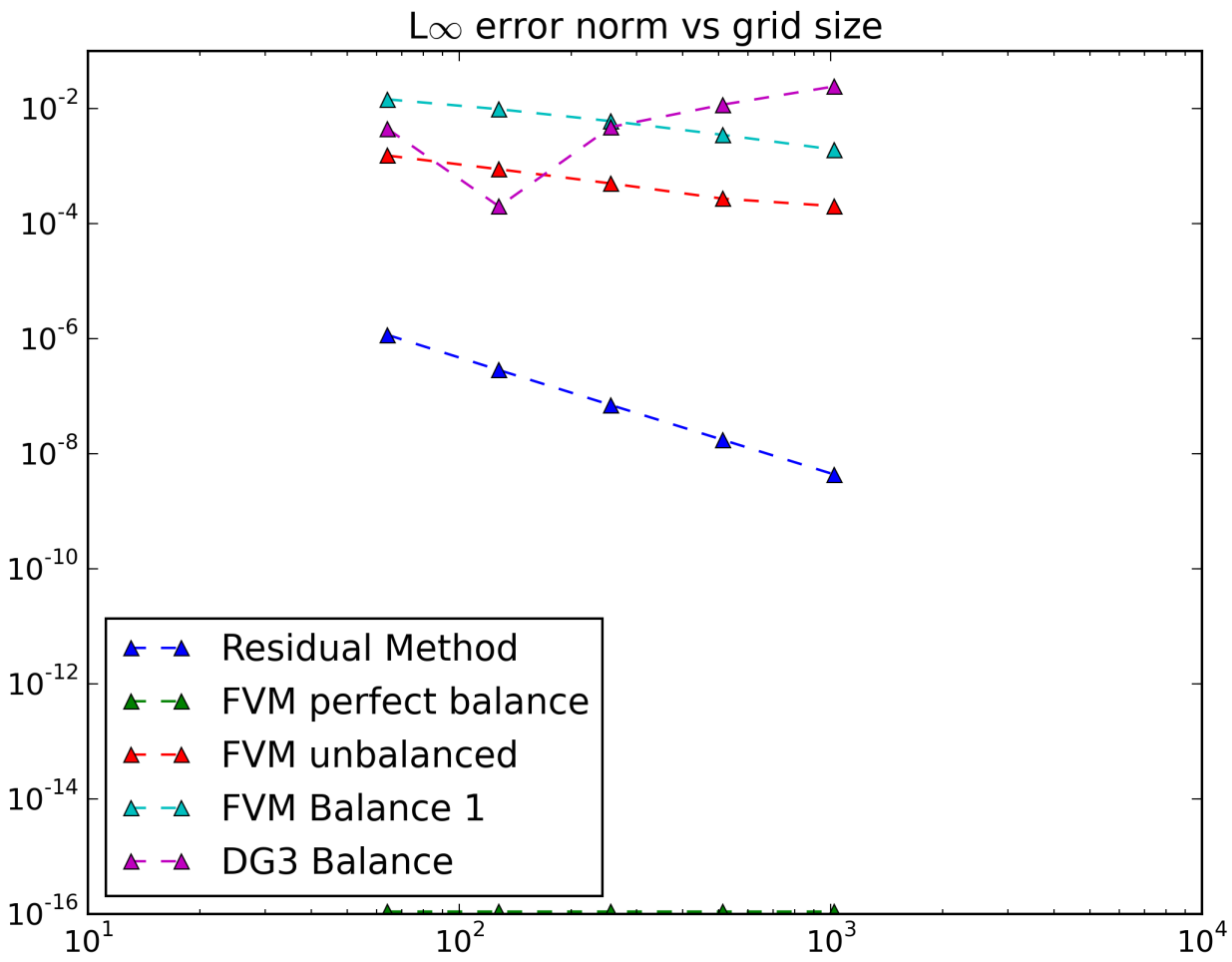
$\alpha = 0.00001$

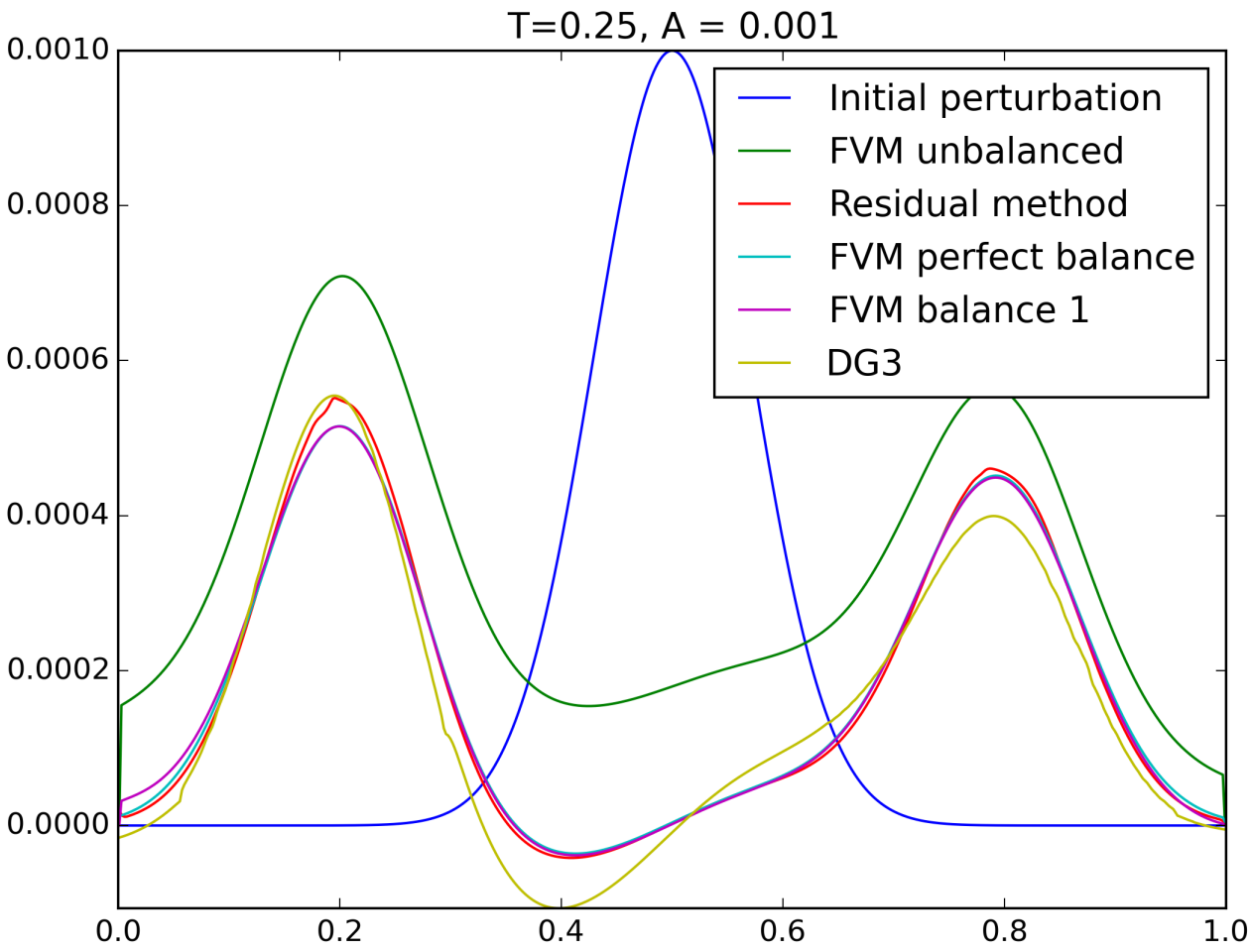


> TT__TT
>
> Boundary conditions don't seem to matter so much.

Convergence plot:

$$l_\infty = \max_i |(u(x_i, t) - \hat{u}(x_i, t)|$$


L∞ error norm vs grid size
10^-2
10^-4
10^-6
10^-8
10^-10
10^-12
10^-14
10^-16
10^1
10^2
10^3
10^4
Residual Method
FVM perfect balance
FVM unbalanced
FVM Balance 1
DG3 Balance



Dg doesn't make any sense. The growing error with refinement of the mesh is disturbing.

Debugging log

> If initial condition is the equilibrium solution, delta_u should be identically 0, and all parts should cancel out.

T = 0.0

```
u_left - u_eq
 -6.2042815329732548E-011   0.0000000000000000       -1.5510703832433137E-010
mode
           1
source 1
  0.0000000000000000        0.0000000000000000        0.0000000000000000
flux volume 1
  0.0000000000000000        0.0000000000000000        0.0000000000000000
flux surface1
 -3.7540074831897270E-008  -3.1789056720299413E-008  -9.3850388853862138E-008
mode
           2
source 1
  0.0000000000000000        0.0000000000000000        0.0000000000000000
flux volume 1
  0.0000000000000000        0.0000000000000000        0.0000000000000000
flux surface1
 -6.5021316928783755E-008   5.4980318964226171E-008  -1.6255364180498508E-007
mode
           3
source 1
  0.0000000000000000        0.0000000000000000        0.0000000000000000
flux volume 1
  0.0000000000000000        0.0000000000000000        0.0000000000000000
flux surface1
 -8.3942159204551289E-008  -7.1082695285440423E-008  -2.0985584919202433E-007
```

T after 100 iterations

```
mode
           1
source 1
  0.0000000000000000        4.5757682205493921E-006   0.0000000000000000
flux volume 1
  0.0000000000000000        0.0000000000000000        0.0000000000000000
flux surface1
 -4.1529389942100458E-006   5.3033872404739668E-006  -1.5875515877461083E-005
mode
           2
source 1
  0.0000000000000000        0.0000000000000000        0.0000000000000000
flux volume 1
  0.0000000000000000       -4.0578370776529482E-003   0.0000000000000000
flux surface1
 -7.1931013387057904E-006   4.0671328542885021E-003  -2.7497199630151634E-005
mode
           3
source 1
  0.0000000000000000        5.5511151231257827E-017   0.0000000000000000
flux volume 1
  0.0000000000000000        2.2737367544323206E-013   0.0000000000000000
flux surface1
 -9.2862538974632689E-006   1.1858734296765761E-005  -3.5498732679880204E-005
mode
```

> Higher modes seem to start deviating. Is this numerical error? Limiting?

http://arxiv.org/pdf/1511.08739v1.pdf

From Chandarashekar and Zenk's paper, they talk about the limiting (nodal DG). " A Nonlinear TVD limiter is necessary (...). The limiter can destroy well balancedness" so they don't limit if the residual solution is close to zero.

> Introduced a check which quits limiting if
>
> $$|\delta u_i| < 1 \times 10^{-6}$$

Doesn't change much. :(

Maybe it's the quadrature? We use Newton Rhapson to get the 0s of the legendre polynomials.

Increasing the iterations to 50: Error for a test case: 1.4750444904687576E-003

Increasing the iterations to 100: Error for a test case: 1.4750444904687576E-003

Nop.

Limiters?

Maybe it's the limiter which is introducing this increasing difference between u and u_{eq}, even when $u_0 = u_{eq}$.

Nx = 128, N = 3, T = 100, BC: $\frac{du}{dt} = 0$.

Using Lydia's limiter Vs basic TVD limiter as reported in Cockburn and Shu:

> Doesn't change much

N = 1 should be identical to FVM... what's x_i? (gauss quadrature for n=1 is not defined so spit out $x_1 = 0$).

Reformulating the scheme

Represent:

$$u(x,t) = u_{eq}(x,t) + \sum_{i=0}^{n} \delta\hat{u}_i(t)\phi_i(x)$$

Compute the modes of the perturbation using DG framework. Let $\phi(x) \in V$ where V is the chosen function space.

$$\delta\hat{u}_i(t) = \int_{\Omega_i} (u(x) - u_{eq}(x))\phi_i(x)$$

$$\frac{\partial\hat{u}_i}{\partial t} = \int_{\Omega_i} F[u_{eq}(x) + \sum_{i=0}^{n} \delta\hat{u}_i(t)]\frac{\partial\phi_i(x)}{\partial x}dx - \int F[u_{eq}(x) + \sum_{i=0}^{n} \delta\hat{u}_i(t)]\phi_i(x)\cdot\hat{n}dS + \int_{\Omega_i} S[u_{eq}(x) + \sum_{i=0}^{n} \delta\hat{u}_i(t)]\phi_i(x)dx$$

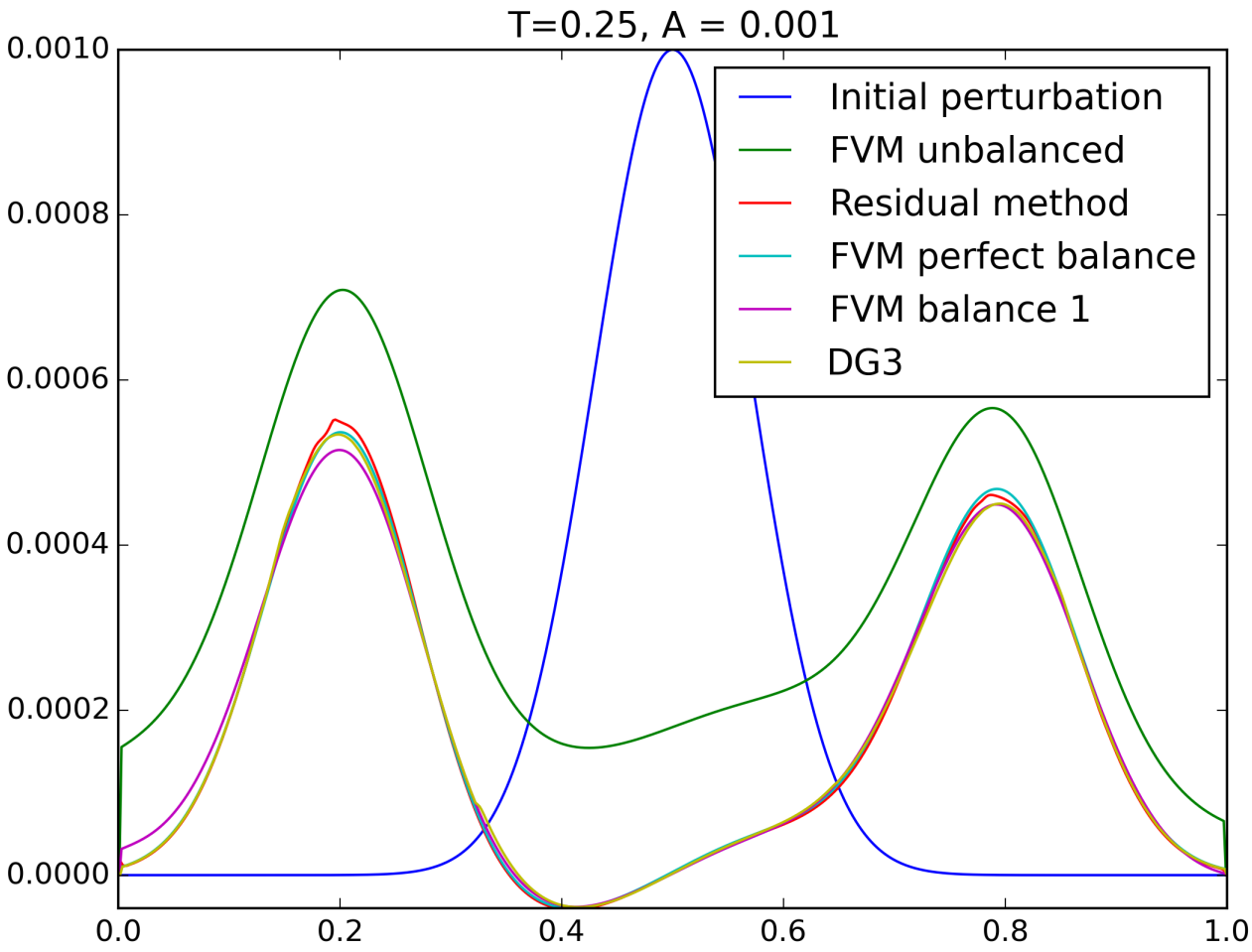
$$\frac{\partial u}{\partial t} = \frac{\partial \delta u}{\partial t} + \frac{\partial u_{eq}}{\partial t}$$

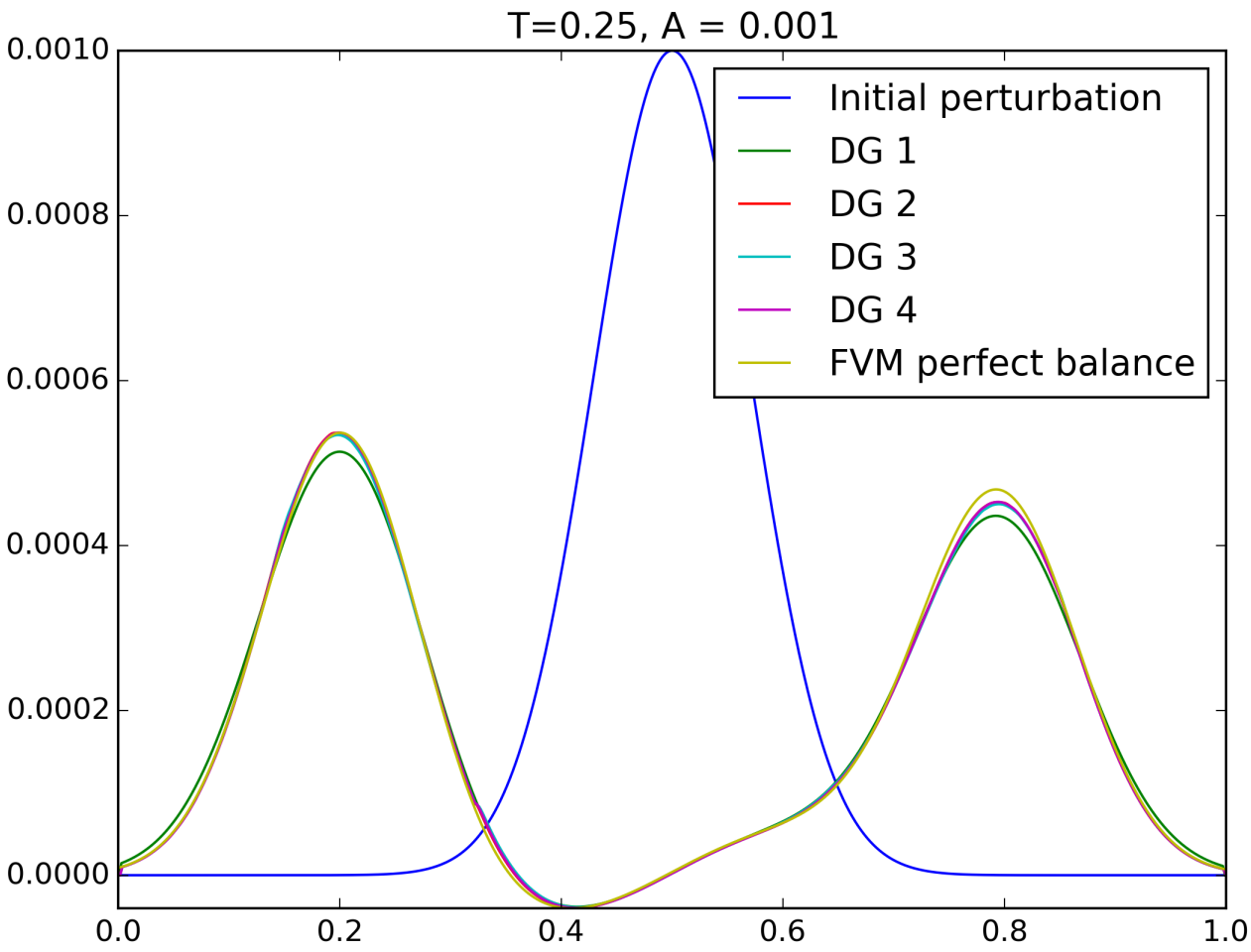
Assuming equilibrium solution, the following should hold: $\frac{\partial u_{eq}}{\partial t} = 0$

Pseudo code:

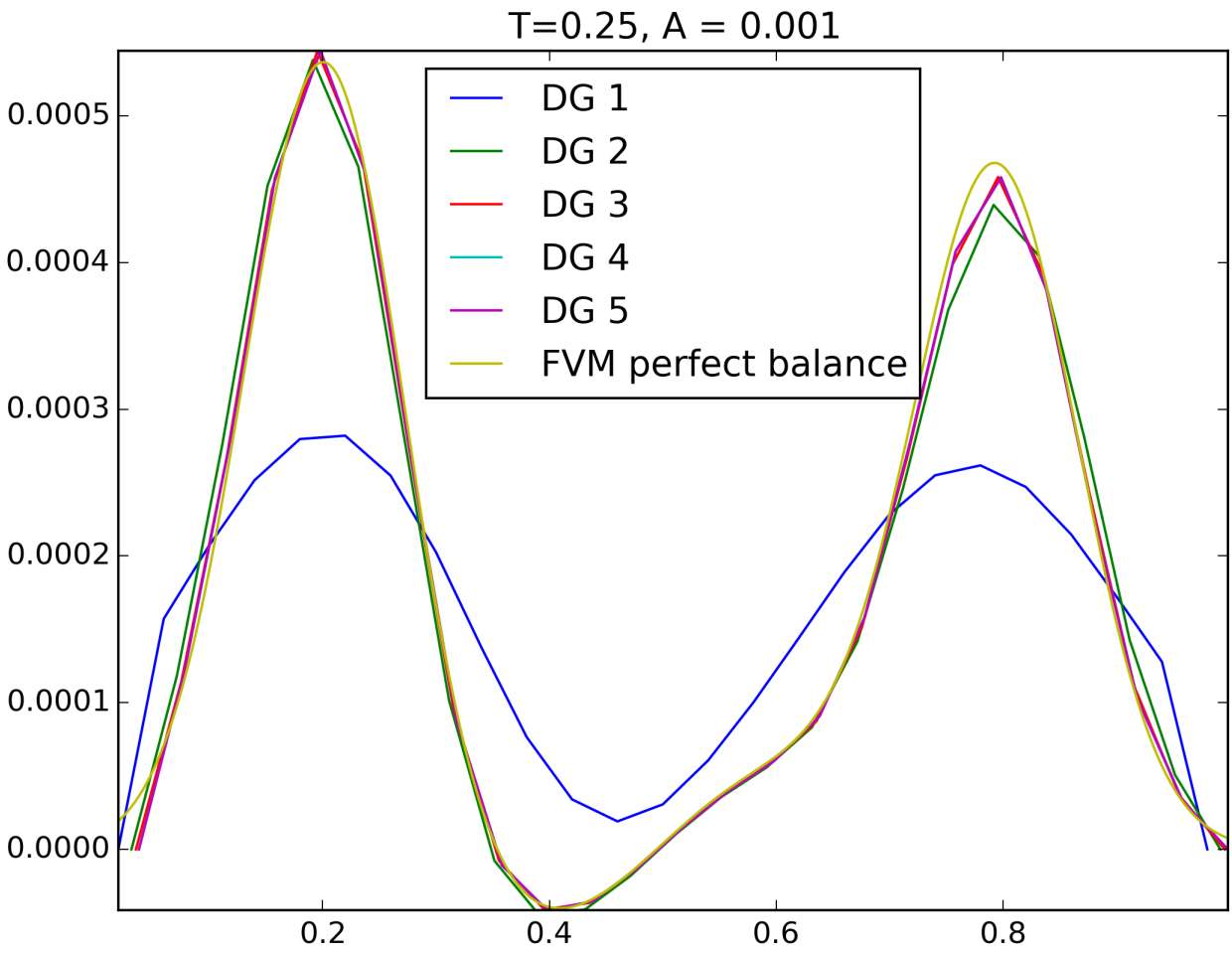
```
I
while t<T:

    w1 = dudt(delta_u)
    delta_u_new = rgkt(w1)

    delta_u_new_modes = get_modes(u_new)
    u_new = u + delta_u_new_modes # we are updating the deltas

    u = u_new
    delta_u = w1
```

New plots:

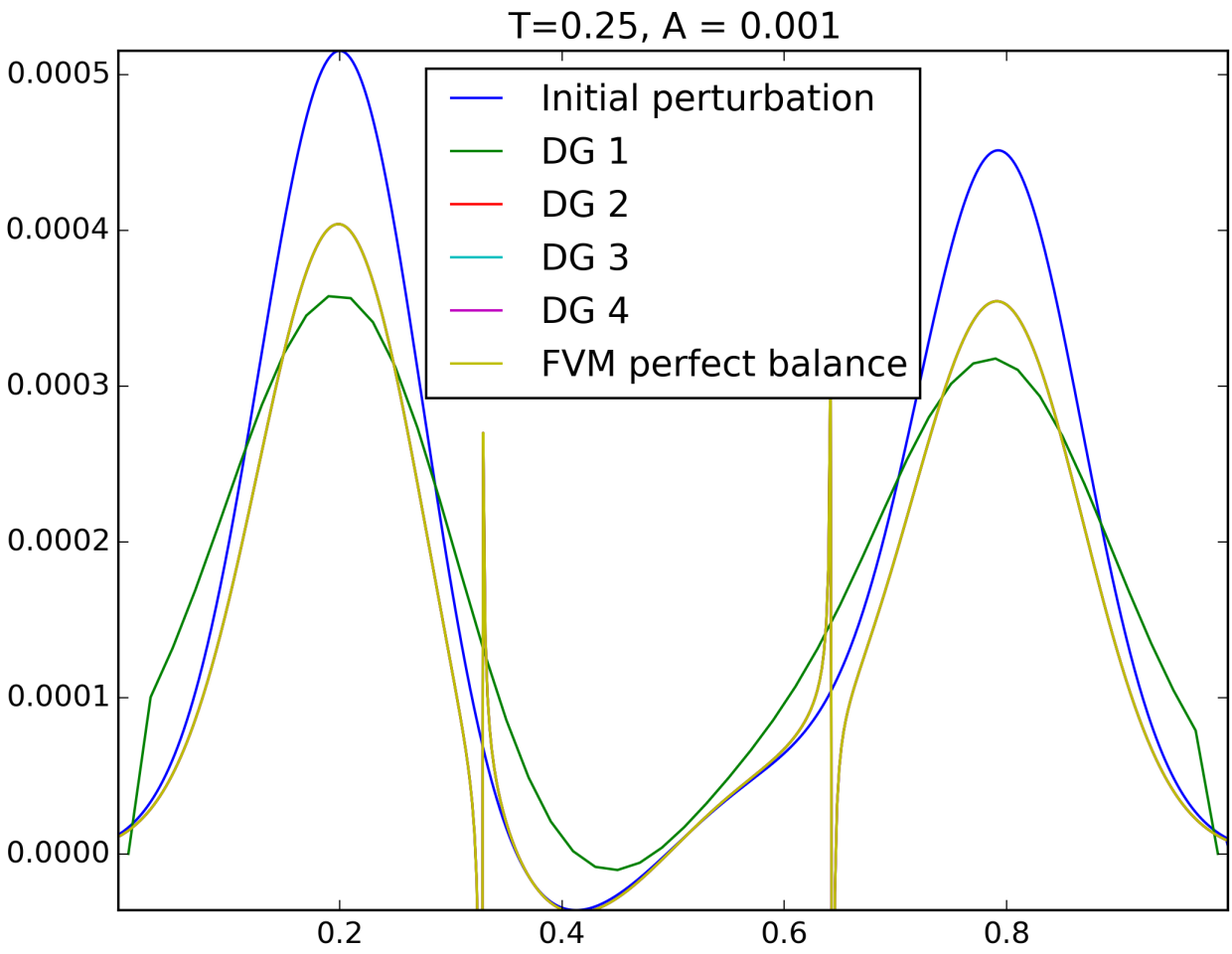




N = 25 versus 8126 as perfect balance.



If I plot const*2*prim(delta*u) instead of const2prim(u*eq + delta*u - delta*u)



!!! There is something fishy with the normalization.

DG 2d Implementation

We look at how to solve the 2D version of a hyperbolic PDE, given as:

$$\frac{\partial u}{\partial t} + \nabla \cdot F(u) = 0$$

For $u(x, y, t) : \mathbb{R} \times \mathbb{R} \times \mathbb{R}^+ \to \mathbb{R}$.

We have 2 terms to approximate:

$$\frac{\partial u_{lk}}{\partial t} = -\int_{\partial l} (F v_{lk}) \cdot \hat{n} dl + \int_{\Omega_c} F \cdot \nabla v_{lk} dx dy$$

Let us treat the boundary terms first:

$$\int_{\delta l} F(x_i, y_i) v_{lk}(x_i, y_i) \cdot \hat{n} dl$$

We are in a cartesian grid, so dealing with the positive y-direction first, we have $\hat{n} = (0, 1)$:

$$\int_{y_i}^{y_{i+1}} F_2(x_i, y_i) v_{lk}(x_i, y_i) dy$$

Using a gaussian quadrature and noting that $v_{lk}(x, y) = \phi_l(x) \times \phi_k(y)$ which is given by the tensor product of 1D basis:

$$\int_{y_i}^{y_{i+1}} F_2(x, y) \phi_l(x) \times \phi_k(y) dy \approx \sum_{j=1}^{N_y} F_2(x_i, y_j) \phi_l(x_i) \phi_k(y_j) w(j)$$

Timestepping:

$$\Delta t \leq |a|^{-1} \frac{CFL\sqrt{\Delta x \Delta y}}{(2m_x + 1)(2m_y + 1)}$$

CFL = 0.5

This is a pretty restricted timestep...

Transverse velocity problem - Comes from having a velocity transverse to the shock front - "cross flow instability". Is it because riemann solver is not truly solving a 2d problem. This comes from solution containining discontinuities and the corner points from the squares?

Is it true that for a smooth solution it shouldn't matter.

For the initial conditions:

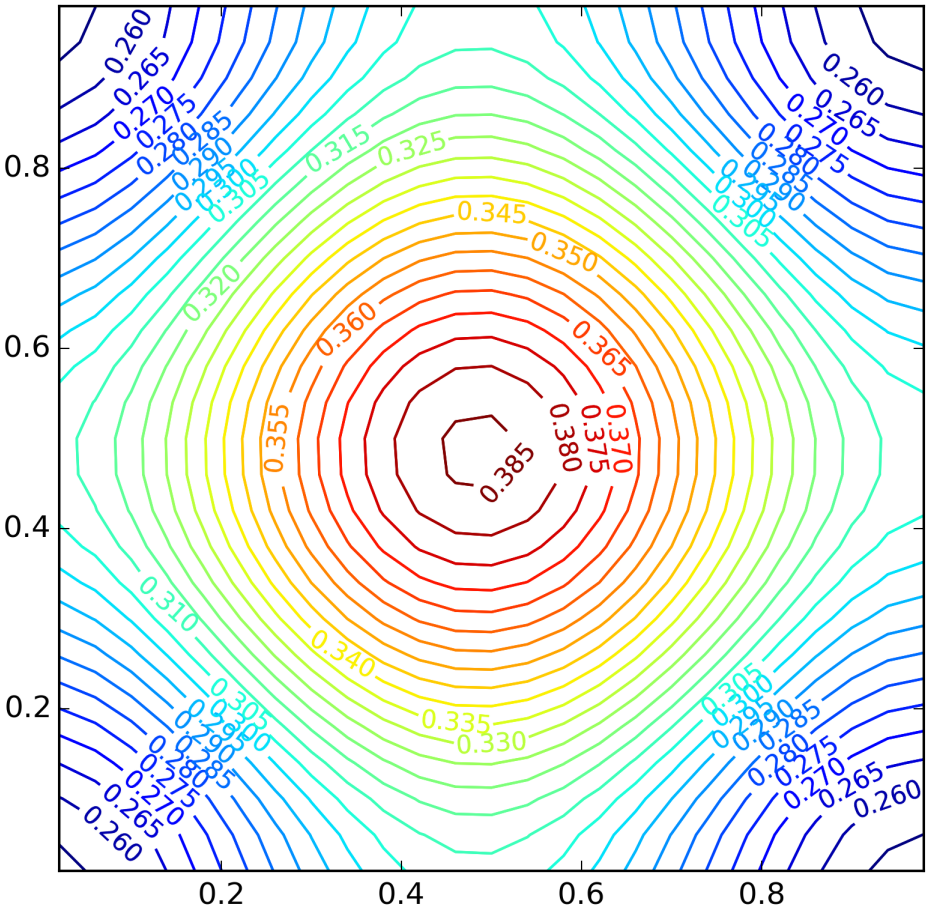
$$\rho = exp(-40 * (x - 0.5)^2 + (y - 0.5)^2)$$

$$u = 1$$

$$v = 1$$

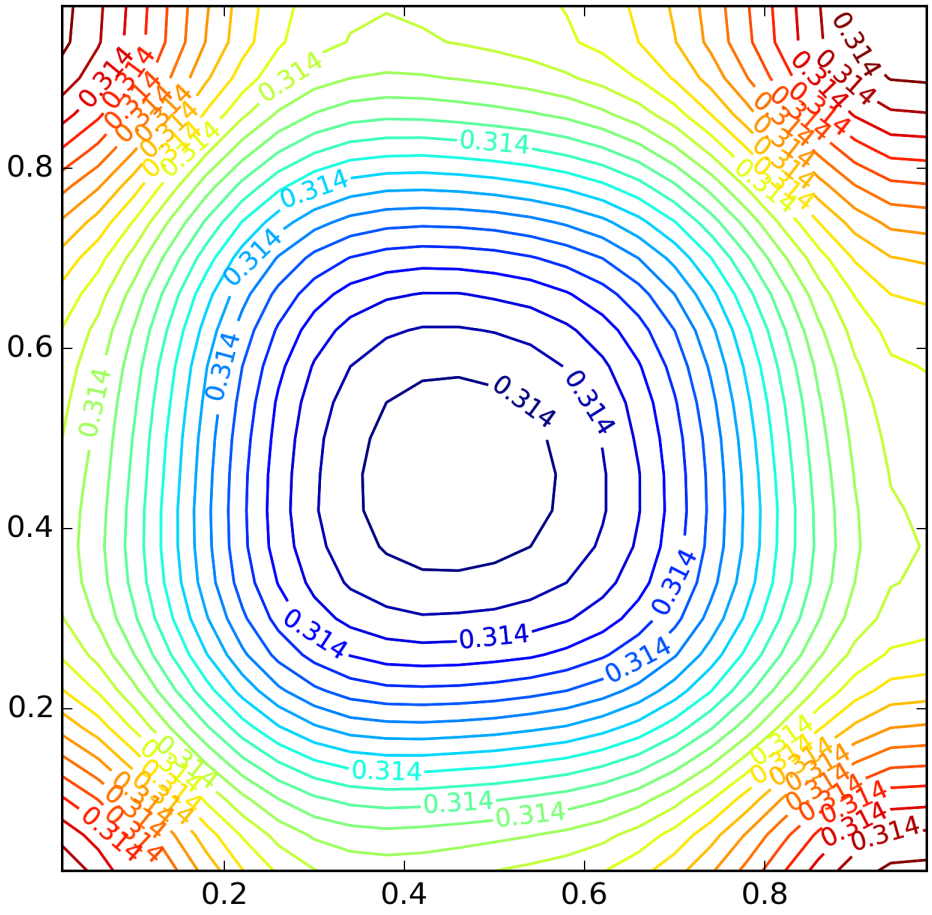
$$p = exp(-40 * (x - 0.5)^2 + (y - 0.5)^2)$$

t = 1.



- First order - Diffusion and deforms the pulse....

t = 2



Smooth IC and pure advection

Well, turns out the initial conditions were stupid, and this wouldn't lead to a linear advection (look at the pressure). So in reality, for linear advection with the Euler system, we want the following IC:

$$\rho = exp(-40 * (x - 0.5)^2 + (y - 0.5)^2)$$

$$u = 1$$

$$v = 1$$

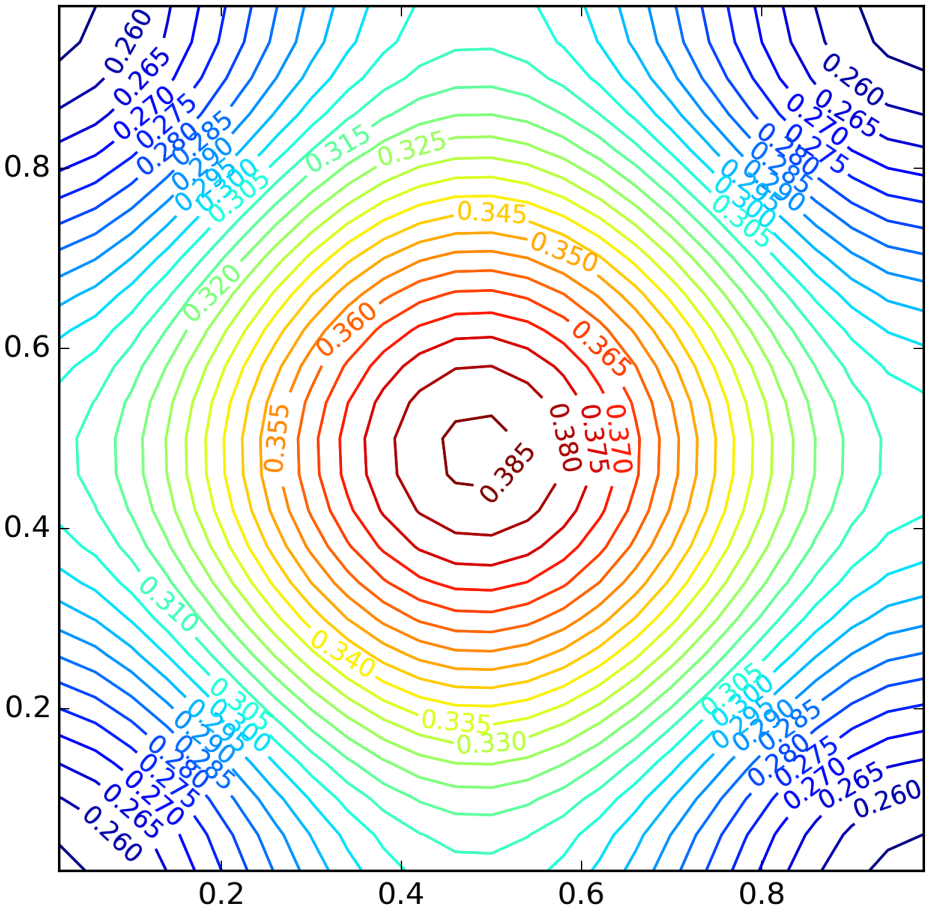
$$p = 1$$

(write the PDE and you can see it!!)

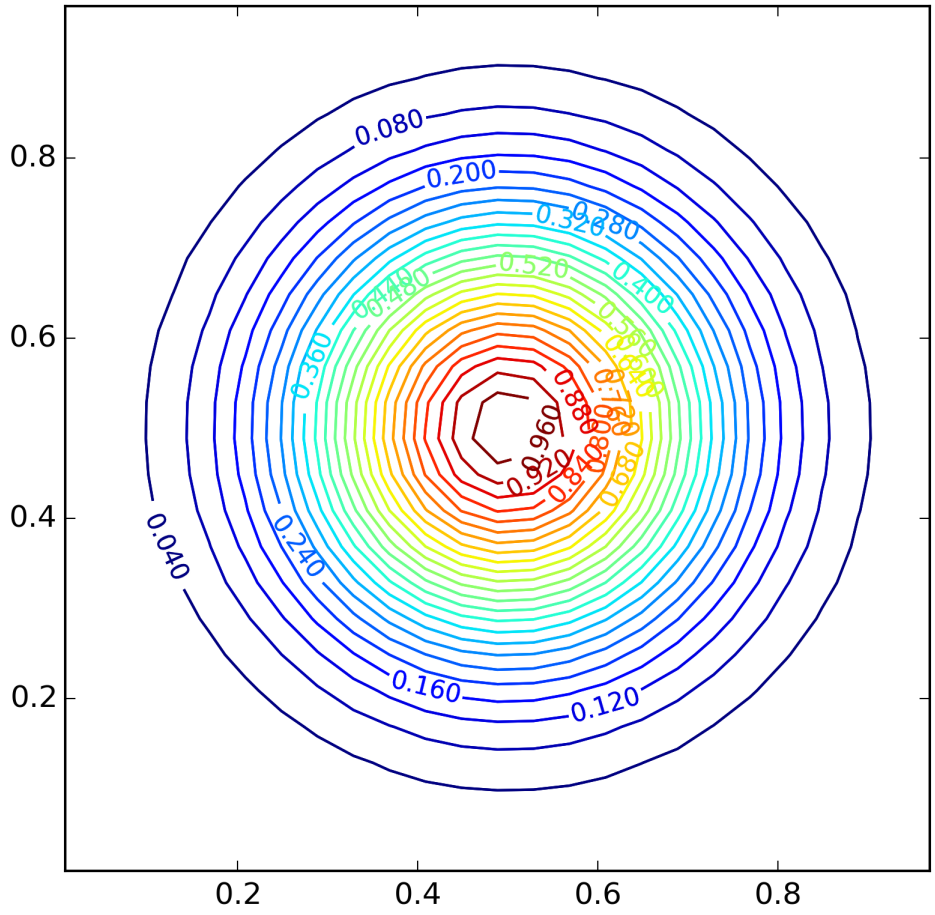
So, testing for higher order:

Today, the 26th of April, we have a working 2D DG solver for any order*. Oh my god! (that people are willing to implement)

o = 1


0.2
0.4
0.6
0.8
0.260
0.265
0.270
0.275
0.280
0.285
0.290
0.295
0.300
0.305
0.310
0.315
0.320
0.325
0.330
0.335
0.340
0.345
0.350
0.355
0.360
0.365
0.370
0.375
0.380
0.385

o = 2



Limiter

Moving onto non smooth initial conditions, we need to include a limiter.

2d limiting

The idea is the same as 1d, but there are some small considerations.

Shu limiter

Details are given here: http://lsec.cc.ac.cn/lcfd/DEWENO/paper/cs5.pdf

To compute u_h , we rely on the assumption that spurious oscillations are present in u_h , only if they are present in its u_h^1 part, which is its L_2 -projection into the space of piecewise linear functions V_h^1 . A theoretical justification of this assumption is still an open problem.

What is implemented in the code is the following. We approximate the solution as

$$u = \bar{u} + u_x * \Phi_1(x, y) + u_y * \Phi_2(x, y) + h.o$$

The intuition is that if the "derivatives", essentially the 2nd and 3rd coefficients (which correspond to the linear

approximation of the solution), are *wiggly*, then the h.o. stuff is also wiggly. This argument has no real theoretical backing but somehow seems to work.

So we replace u_x and u_y by

$$minmod(u_x, \bar{u}_{i+1,j} - \bar{u}_{i,j}, \bar{u}_{i,j} - \bar{u}_{i-1,j})$$

$$minmod(u_y, \bar{u}_{i,j+1} - \bar{u}_{i,j}, \bar{u}_{i,j} - \bar{u}_{i,j-1})$$

respectively.

And if $u_x \neq minmod(u_x, \bar{u}_{i+1,j} - \bar{u}_{i,j}, \bar{u}_{i,j} - \bar{u}_{i-1,j})$, we set the h.o. terms to zero. Effectively, worst case scenario, we revert to 1st order approximation.

This clips extrema.

2-D riemann problem (case 3)

Configuration 1:

The initial data are

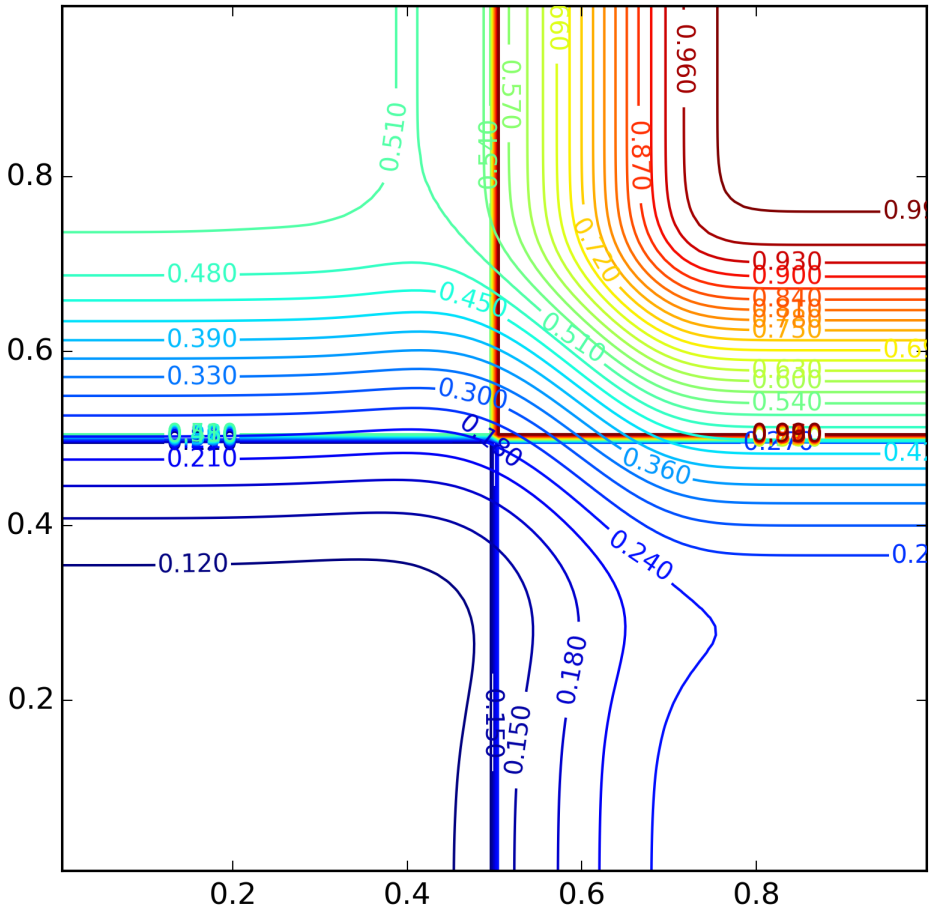
$$\begin{array}{llll} p_2 = 0.4 & \rho_2 = 0.5197 & p_1 = 1 & \rho_1 = 1 \\ u_2 = -0.7259 & v_2 = 0 & u_1 = 0 & v_1 = 0 \end{array}$$

$$\begin{array}{llll} p_3 = 0.0439 & \rho_3 = 0.1072 & p_4 = 0.15 & \rho_4 = 0.2579 \\ u_3 = -0.7259 & v_3 = -1.4045 & u_4 = 0 & v_4 = -1.4045 \end{array}$$

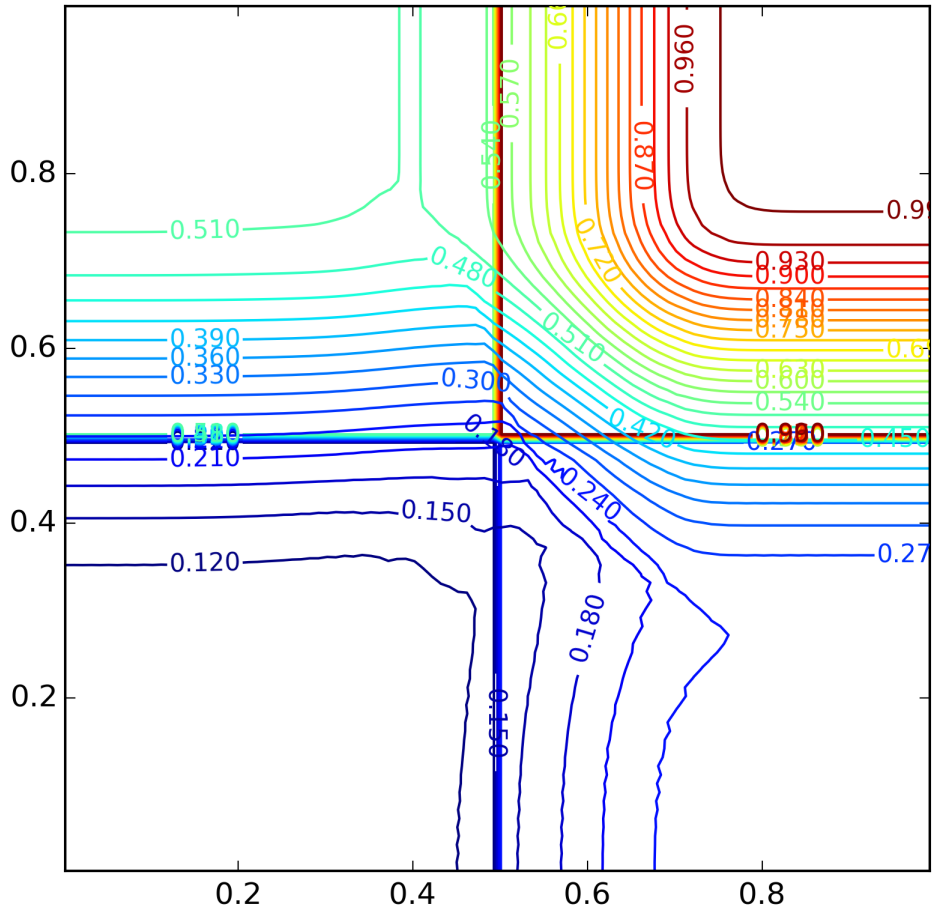
(Rarefaction waves)

100x100, t=0.3

order = 1

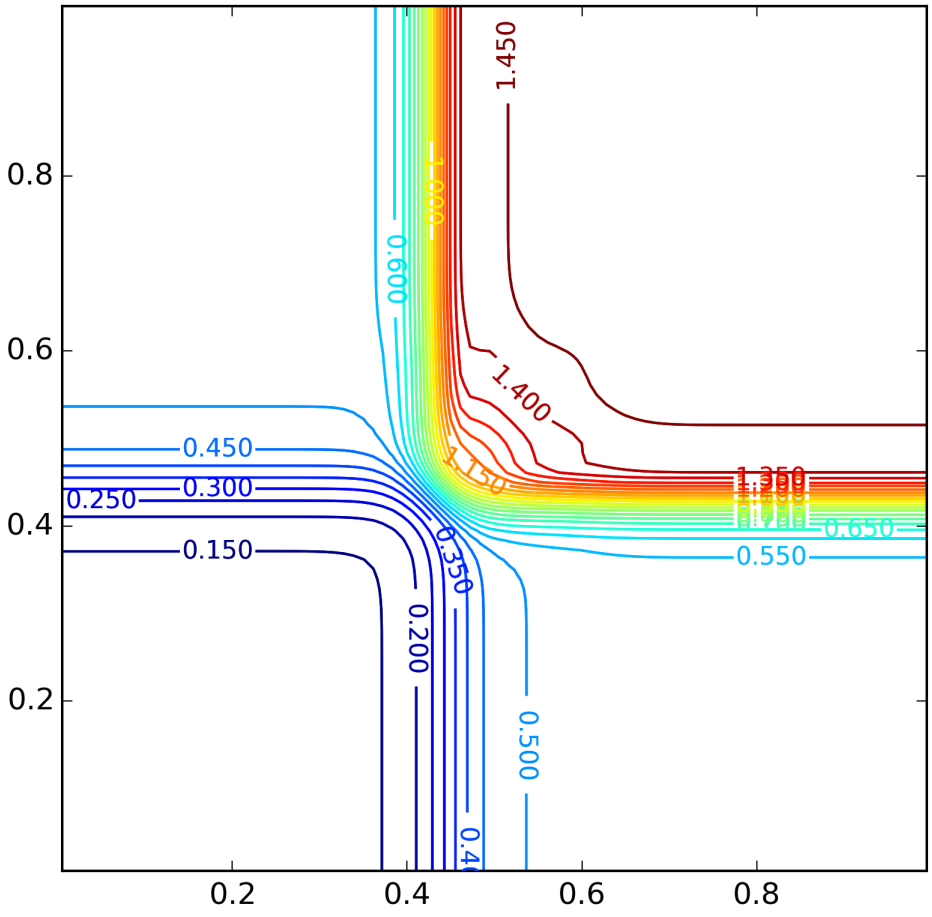

0.8
0.6
0.4
0.2
0.2
0.4
0.6
0.8
0.510
0.480
0.450
0.390
0.330
0.300
0.270
0.210
0.120
0.540
0.570
0.600
0.720
0.870
0.960
0.930
0.900
0.840
0.750
0.660
0.630
0.600
0.540
0.510
0.480
0.450
0.420
0.390
0.360
0.330
0.300
0.270
0.240
0.210
0.180
0.150
0.120

order = 2

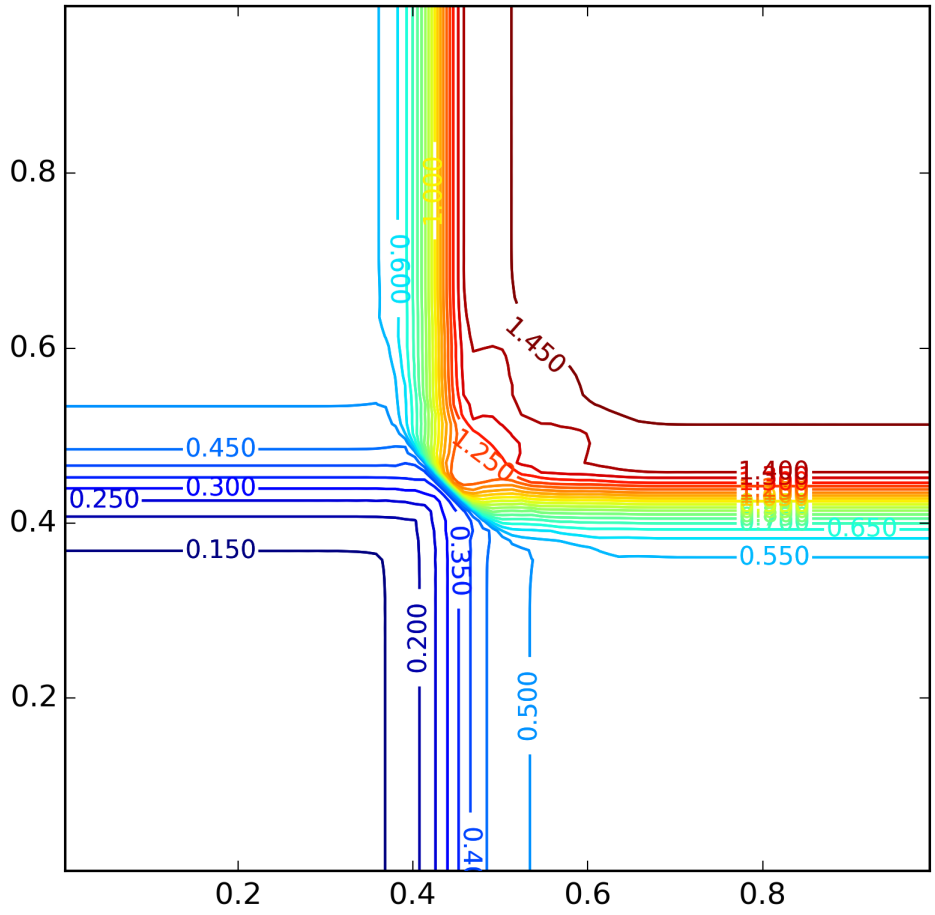


Configuration 2 (Shocks)

100x100, t=0.2 order = 1


1.450
1.400
1.350
1.150
0.650
0.600
0.550
0.500
0.450
0.350
0.300
0.250
0.200
0.150
0.8
0.6
0.4
0.2
0.2
0.4
0.6
0.8

order = 2



to do

- limit in characteristics - bit of oscillations
- Krivodonova limiter
- hydrodynamical implosion

Issues

- cfl
- interesting ics
- residual methods uses triangular grid :(