# DGANN solver


### Created by: Deep Ray, EPFL, Switzerland

### Webpage: deepray.github.io


### Date : 10 January, 2018

**DGANN** is a RKDG-solver written in MATLAB, which is also capable of using an artificial neural network trained to serve as a troubled-cell detector. The main source code for the RKDG solver is based on the solvers available with the following texts:

1. Nodal Disontinuous Galerkin methods, by Jan S. Hesthaven and Tim Warburton.
2. Numerical Methods for Conservation Laws: From Analysis to Algorithms, by Jan S. Hesthaven.

Details about the design of the Multilayer Perceptron (MLP) troubled-cell indicator have been published in the paper "An artificial neural network as a troubled-cell indicator".

## Running the code

After cloning the git repository, execute **mypath.m** from the parent directory in MATLAB. This will set all the neccesary paths to use the solver. The various test cases need to be run from the **Examples** directory.

### Scalar 1D

The basic structure of the example script is as follows.

```
CleanUp1D;

clc
clear all
close all

Globals1D_DG;
Globals1D_MLP;

model = 'Advection';
test_name = 'Sine';
u_IC =@(x) sin(10*pi*x);

bnd_l     = 0.0;
bnd_r     = 1.4;
mesh_pert = 0.0;
bc_cond   = {'P',0.0,'P',0.0};
FinalTime = 1.4;
CFL       = 0.2;
Nelem     = 100;
N         = 4;

indicator_type = 'minmod';
nn_model       = 'MLP_v1';
rec_limiter    = 'minmod';

plot_iter = 100;
save_soln = true;
save_ind  = true;

% Call code driver
ScalarDriver1D;
```

- `CleanUp1D` removes temporary file paths added, which is especially important if the previous run of the script terminated prematurely. This must be the first line of every script. DO NOT REMOVE IT!
- `Globals1D_DG` and `Globals1D_MLP` declare important global variables needed by the solver and the MLP network.
- The `model` flag sets the type of scalar model which is being solved. The following scalar models are currently available:
  - `'Advection'` : Linear advection equation with the advection speed set to 1.
  - `'Burgers'` : Burgers equation with the flux $u^2/2$.
  - `'BuckLev'` : Buckley-Leverett equation with flux contant set to $0.5$.
- `test_name` is used to declare the name of the test. This is used for creating various save files.
- `u_IC` is used to set the initial condition for the problem.
- `bnd_l` and `bnd_r` define the left and right domain boundaries, which is descritized using `Nelem` number of elements. The degree of the polynomial in each element/cell is set using `N` .
- `mesh_pert` is used to randomly perturb the interior cell-interfaces using the following algorithm

$$x_{i+\frac{1}{2}} \to x_{i+\frac{1}{2}} + \text{mesh\_pert}\; h\; \omega_{i+\frac{1}{2}}, \qquad \omega_{i+\frac{1}{2}} \in \mathcal{U}[-0.5, 0.5], \qquad i = 1, \ldots, Nelem - 1$$

  where $h$ is the mesh size of the initial uniform grid.

- `bc_cond` is used to set the left and right boundary conditions. It is a cell of the form `{LEFT_BC_TYPE, LEFT_BC_VAL, RIGHT_BC_TYPE,RIGHT_BC_VAL}` . The `BC_TYPES` can be set as:

    - `'P'` : Periodic boundary conditions. In this case, both boundary type must be set to `'P'` .
    - `'N'` : Neumann boundary condition.
    - `'D'` : Dirichlet boundary condition, with the imposed Dirichlet value given by `BC_VAL` . Note that `BC_VAL` is ignored if `BC_TYPE` is not set to `'D'` .
- The final simulation time is set using `FinalTime` , while the time step is chosen using `CFL` .
- The troubled-cell indicator is set using `indicator_type` . The following options are currently available:
    - `'minmod'` : Uses the basic minmod limiter.
    - `'TVB'` : Uses the modified minmod-type TVB limiter. If this is chosen, then one aslo needs to set the variable `TVB_M` to a positive number. Note that if this constant is set to 0, then the limiter reduces to the usual minmod limiter.
    - ' `NN` ': Uses the trained neural network. The network name is set using the variable `nn_model` . The available networks are described below in the section **Using the MLP indicator**.
- The limiter usd to reconstruct the solution in each troubled-cell,is set using `rec_limiter` . The following options are currently available:
    - `'none'` : No limiting is applied.
    - `'minmod'` : MUSCL reconstruction using minmod limiter.
- The flag `plot_iter` is used for visualization purposes. The solution is plots are shown after every `plot_iter` number of iterations during the simulation.
- If solution at the final time needs to be saved, the flag `save_soln` must be set to `true` . If this is not the case, set this flag to `false` . The solution files are save in the directory **User_output**. The filename has the format: `{model}1D_{test_name}_{N}_{Nelem}_{indicator_type}_{rec_limiter}.dat` . If mesh perturbation is used, then the filename will end with the tag `pert` . The data in the file has the following format:
    - Column 1: x-coordinates
    - Column 2: solution value at corresponding x-coordinate
- If the time-history of the troubled-cells needs to be viewe/saved, the flag `save_indn` must be set to `true` . If this is not the case, set this flag to `false` . The time-history files are also save in the directory **User_output**. The filename has the format: `{model}1D_{test_name}_{N}_{Nelem}_{indicator_type}_{rec_limiter}_tcells.dat` . If mesh perturbation is used, then the filename will end with the tag `pert` . The data in the file has the following format:
    - Each row of the file contains the time, followed by the mid-points of the cells flagged as troubled-cells at that time.
    - The first row corresponds to the cell flagged at time $t = 0$. This is essentially done for the initial condition.
    - Following the first row, the rows can be grouped in sets of size $r$, to list the cells flagged after each sub-stage of the r-stage time-integration scheme. Since the current implementation of the code uses SSP-RK3, the rows will be grouped as triplets.
- The main driver script `ScalarDriver1D` is called once all the flags have been set.

## Shallow Water 1D

The basic structure of the example script is as follows.

```
CleanUp1D;

clc
clear all
close all

Globals1D_DG;
Globals1D_MLP;

model        = 'SWE';
gravity      = 1.0;
test_name    = 'Dambreak';
depth_IC     =@(x) 3*(x<0.0) + 1*(x>=0.0);
velocity_IC  =@(x) 0*x;

bnd_l     = -3.0;
bnd_r     = 3.0;
mesh_pert = 0.0;
bc_cond   = {'N',0.0,'N',0.0;
             'N',0.0,'N',0.0};  % For conserved variables
FinalTime = 1;
CFL       = 0.4;
Nelem     = 100;
N         = 4;

indicator_type = 'minmod';
ind_var        = 'depth';
nn_model       = 'MLP_v1';
rec_limiter    = 'minmod';
lim_var        = 'con';

plot_iter = 100;
save_soln = true;
save_ind  = true;

% Call code driver
SWEDriver1D;
```

Most of the structure is similar to the Scalar 1D script. The differences are described below.

- The `model` needs to be set as `'SWE'`. This should not be changed.
- The acceleration due to gravity is set using `gravity`.
- The initial depth and velocity are set using `depth_IC` and `velocity_IC`.
- The `bc_cond` has the same format as earlier, although now it has two rows of parameters. The first row corresponds to depth, while the second corresponds to the discharge (depth*velocity). Note that the boundary condition are set for the conserved variables.
- For systems of conservation laws, there are various choices for the variables to be used for troubled-cell detection. For the shallow water equations, this choice is made via the flag `ind_var`, with the following options:
    - `'depth'`: Only the depth variable is used
    - `'velocity'`: Only the velocity variable is used
    - `'prim'`: The primitive variables i.e., depth and velocity, are used. The troubled-cells flagged for each variable is pooled together.
- As was the case with detection, there are several options for the variables which can be reconstructed. This is set using the

flag `rec_var`, with the following options:

  - `'prim'`: The primitive variables i.e., depth and velocity, are recontructed.
  - `'con'`: The conserved variables i.e., depth and discharge, are recontructed.
  - `'char_cell'`: The local characterictic variables are reconstructed. These are obtained cell-by-cell using the linearized transformation operators. More precisely, the transformation matrix in each cell is evaluated using the cell-average value, following which the conserved variables are transformed to the characteristic variables in that cell. The same transformation is used to retrieve the conserved variables after limiting the characteristic variables.
  - `'char_stencil'`: The local characterictic variables obtained cell-by-cell can introduce spurious oscillations in the solution. One can also obtain the local characteristic variables, stencil-by stencil. More precisely, for a given reconstruction stencil of 3-cells, the transformation matrix is evaluated using the cell-average value of the central cell, following which the conserved variables are transformed to the characteristic variables in every cell of that stencil. The transformed variables are used to obtain the reconstructed characteristic variables in the central cell. Note that this approach can be 3 times more expensive than the `'char_cell'` approach.

- The solution filename has the format: `{model}1D_{test_name}_{N}_{Nelem}_{indicator_type}_{ind_var}_{rec_limiter}_{rec_var}.dat`. If mesh perturbation is used, then the filename will end with the tag `pert`. The data in the file has the following format:

  - Column 1: x-coordinates
  - Column 2: the value of depth and velocity (in that order) at corresponding x-coordinate.

- The troubled-cell time-history filename has the format: `{model}1D_{test_name}_{N}_{Nelem}_{indicator_type}_{ind_var}_{rec_limiter}_{rec_var}_tcells.dat`. If mesh perturbation is used, then the filename will end with the tag `pert`.
- The main driver script `SWEDriver1D` is called once all the flags have been set.

## Euler 1D

The basic structure of the example script is as follows.

```
CleanUp1D;

clc
clear all
close all

Globals1D_DG;
Globals1D_MLP;

model     = 'Euler';
gas_const = 1.0;
gas_gamma = 1.4;
test_name = 'ShockEntropy';
rho_IC    =@(x) (x<-4)*3.857143 + (x>=-4).*(1 + 0.2*sin(5*x));
vel_IC    =@(x) (x<-4)*2.629369;
pre_IC    =@(x) (x<-4)*10.33333 + (x>=-4)*1.0;


bnd_l     = -5.0;
bnd_r     = 5.0;
mesh_pert = 0.0;
bc_cond   = {'D',3.857143,'N',0.0;
             'D',10.141852,'D',0.0;
             'D',39.166661,'N',0.0};  % For conserved variables
FinalTime = 1.8;
CFL       = 0.1;
Nelem     = 200;
N         = 4;

indicator_type = 'minmod';
ind_var        = 'prim';
nn_model       = 'MLP_v1';
rec_limiter    = 'minmod';
lim_var        = 'con';

plot_iter = 100;
save_soln = true;
save_ind  = true;

% Call code driver
EulerDriver1D;
```

Most of the structure is similar to the shallow water 1D script. The differences are described below.

- The `model` needs to be set as `'SWE'`. This should not be changed.
- The gas constant and ratio of specific heats is set using `gas_const` and `gas_gamma`.
- The initial density, velocity and pressure are set using `rho_IC`, `vel_IC` and `pre_IC`.
- The `bc_cond` has the same format as earlier, although now it has three rows of parameters. The first row corresponds to density, the second corresponds to the momentum, and the third to energy. Note that the boundary condition are set for the conserved variables.
- For the Euler equations, `ind_var` can be set as:
    - `'density'`: Only the density is used
    - `'velocity'`: Only the velocity is used
    - `'pressure'`: Only the pressure is used

    - `'prim'` : The primitive variables i.e., depth, velocity and pressure, are used. The troubled-cells flagged for each variable is pooled together.
    - `'energy'` : Only the energy is used.
    - `'de'` : Density and energy are used.

- The main driver script `EulerDriver1D` is called once all the flags have been set. The troubled-cells flagged for each variable is pooled together.

# Using the MLP indicator

For those interested in using the trained indicator in their own solvers, we explain the various components of the network. The descriptor files for the various trained networks are available under the folder **Trained_networks**. For each network, the following files exist:

- ***model_parameters.dat***: Lists the dimensions of the input (IDIM) and output (ODIM) layers for the network, the number of hidden layers (NHL), and network hyperparameters (the Leaky ReLU factor, etc.).
- ***w_h{i}.dat***: Weights for the i'th hidden layer.
- ***w_out.dat***: Weights for the output layer
- ***b_h{i}.dat***: Biases for the i'th hidden layer
- ***b_out.dat***: Biases for the output layer.
- ***Scaling.m***: Script for scaling the input data before passing it through the network.

Consider the input $X$ to be an array of size $m \times n$, where $m$ is the dimension of each data sample, while $n$ is the number of samples. The algortihm for the network with $L$ hidden-layers having $N_l$ neurons in the $l$-th hidden layer, is as follows:

$$
\begin{aligned}
X_0 &= \text{Scaling}(X), \quad N_0 = m \\
X_l &= f_{\text{activation}}(W_l X_{l-1} + b_l \mathbb{1}(n)), \quad W_l \in \mathbb{R}^{N_l \times N_{l-1}}, \quad b_l \in \mathbb{R}^{N_l}, \qquad l = 1, \ldots, L \\
Y &= \text{Softmax}(W_{out} X_L + b_{out} \mathbb{1}(n)), \quad W_{out} \in \mathbb{R}^{2 \times N_L}, \quad b_{out} \in \mathbb{R}^2
\end{aligned}
$$

where the final output $Y$ is of dimension $2 \times n$, and $\mathbb{1}(n)$ is a row vector of $n$ ones. The activation function is chosen as a suitable non-linear function, such as the Logistic function, hyperbolic tangent, ReLU, etc. The Softmax function is used a the output function. The first row of $Y$ gives the probability of the sample corresponding to a troubled-cell, while the second row gives the probability of the sample corresponding to a good-cell. Note that the sum along each column of $Y$ equals $1$. The indices of the troubled-cells can be obtained as

$$
ind = \text{find}(Y(1,:) > 0.5).
$$

The MATLAB scripts to read and run the networks can be found under the folder **MLP_scripts**. For instance, the following scripts for 1D problems are available in the **1D** sub-directory:

- **read_mlp_param1D.m** reads the various weights and biases for a given network.
- **ind_MLP1D.m** runs the network for a given input `x`.

The **Common** sub-directory contains additional scripts needed to run the networks, such as the implementation of the leaky ReLU activation function.

## Network for 1D problems

The following is a list of the available networks for 1D problems. The latest recommended network is MLP_v1.

- **MLP_v1**: This network has an input layer of size 5. In particular, the input for the classification of cell $i$ is $X = [\overline{u}_{i-1}, \overline{u}_i, \overline{u}_{i+1}, u^+_{i-\frac{1}{2}}, u^-_{i+\frac{1}{2}}]$, where the first 3 quantities are the cell averages of the solution in the cells $i-1, i, i+1$

and the last two entries are the left and right cell interface values of the approximating polynomial in cell $i$. There are 5 hidden layers, whose widths are 256, 128, 64, 32 and 16, going from the input to the output layer. The activation function is taken to be the Leaky ReLU activation function

$$f_{\text{activation}}(U) = max(0, U) - \nu(0, -U),$$

with the parameter $\nu$. The details and results with this indicator are published here.