```
%{

A MATLAB program solving c(x)y''+c'(x)y'-f = 0, y(0)=y(1)=0, by means of a
PINN implemented using symbolic differentitation and the gradient decent
method.


Created by:
Andreas Almqvist

Luleå University of Technology
Departement of Machine Elements
2020-01-01

Copyright (c) 2020 Andreas Almqvist

All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

1. Redistributions of source code must retain the above copyright notice,
   this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
%}

%% Now, let's start!
clear all;

% Input
Ni = 21; % Number of grid points for the solution domain [0,1]

K = 1; % Slope parameter for the Reynolds equation

% Specification of training parameters
Ne = 2000;  % # of Epochs (1 Epoch contains Tb training batches)
Tb = 600;   % # of training batches (# or corrections during 1 Epoch)
lr = 0.005; % Learning rate coefficient (relaxation for the update)
Nn = 10;    % Number of nodes in the 1st hidden layer
Tt = 1e-30; % Training tolerance N.B. redundant in the current version

%% Preprocessing

% Initialisation of weights and bias
```

```
w0 = -2 + 4*rand(Nn,1);
b0 = -2 + 4*rand(Nn,1);
w1 = 0*(-1 + 2*rand(Nn,1)); % w1=0 and b1=0 makes y0 = 0
b1 = 0*(-1 +2*rand(1));

% A pretty good intial guess/solution for K=1;
aux=[...
1  0.0557     1.9808   -0.2186
2  -6.3047    6.1664    0.1220
3  -9.3674   11.4571    0.3843
4  -4.5473    3.3266    0.0305
5  -2.4464   -1.9884    0.1188
6  -0.1365   -0.1674    0.4155
7   0.8581    0.5253    0.5089
8   1.0901    2.0858    0.3348
9   0.2085    0.2523   -0.2024
10 -3.2168    5.9722   -0.9899];
w0=aux(:,2);
b0=aux(:,3);
w1=aux(:,4);
b1 = -0.064;

params =[w0;b0;w1;b1];

% Domain
x  = linspace(0,1,Ni);
dx = x(2)-x(1);

% Reynolds specific
H    = (1+K)-K*x;
dHdx = -K*ones(1,Ni);
c    = H.^3;
cp   = 3*H.^2.*dHdx; % dc/dx
f    = dHdx; % dH/dx

[y_paper,dydx_paper,d2ydx2_paper,y0_paper,y1_paper] =...
    prediction(x,w0,b0,w1,b1);

% Analytical solution
P1D = (1/K)*(1./(1+K-K*x) - (1+K)/(2+K)*1./(1+K-K*x).^2-1/(2+K));

% Epoch vector (auxiliary parameter)
epoch = 1:Ne;

t0ANN = tic;

%% Training the network
for ii = epoch % Looping Epochs
    [params,costi,bi] = trainnn(x,params,c,cp,f,Nn,lr,Tb,Tt);
    w0 = params(1:Nn);
    b0 = params(Nn+1:2*Nn);
    w1 = params(2*Nn+1:3*Nn);
    b1 = params(end);
    cost(ii) = costfunction(x,c,cp,f,w0,b0,w1,b1); % Cost for each epoch
    batch(ii) = bi;                          % # of batches for each epoch
    disp([cost(ii)]);
end

tANN = toc(t0ANN);
```

```
%% Visualisation of prediction-, training- and validation data.
figure(14); clf;
[y,dydx,d2ydx2,y0,y1] = prediction(x,w0,b0,w1,b1);
plot(x,y,'r-.')
hold on;
plot(x,P1D) % Exact solution
lh = legend('PINN prediction','Exact solution');
set(lh,'interpreter','latex','fontsize',16,'location','northwest');
set(gca,'fontsize',16)
```

```
%% Function library
%
%
%
%
% Training function
function [params,costi,bi] = trainnn(input,params,c,cp,f,Nn,lr,Tb,Tt)

% Number of training data points
m  = size(input,2);

% Explicit weights and bias
w0 = params(1:Nn);
b0 = params(Nn+1:2*Nn);
w1 = params(2*Nn+1:3*Nn);
b1 = params(end);

% Auxiliary vector
one = ones(size(w0));

% Training loop
bi = 0;    % Initialisation
costi  = 2*Tt; % -||-
while (bi <= Tb) && (costi > Tt)

    % Pick a random data point
    ri = randi([1,m]); % Random integer between 1 and m
    x  = input(ri);    % Input data point ri

    ci  = c(ri); % The operator functions at ri
    cpi = cp(ri);
    fi  = f(ri);

    % The prediction of y(x)
    z_i   = mysigmoid(w0*x+b0);     % Nnx1 vector
    mySzp = mysigmoid(w0*x+b0).*... % Prime of the sigmoid evaluated at z_i
             (1-mysigmoid(w0*x+b0));% Dubbel prime -||-
    mySzpp = mySzp.*(1-2*mysigmoid(w0*x+b0));
    mySzppp = mySzpp.*(1-2*mysigmoid(w0*x+b0))-2*mySzp.^2;

    y     = sum(w1.*z_i)+b1;        % The prediction at current batch
```

```
% The prediction of dy/dx
yp    = sum(w1.*w0.*mySzp);

% The prediction of d2y/dx2
ypp   = sum(w1.*w0.^2.*mySzpp);

% The prediction of y(0)
z_i0  = mysigmoid(b0);
y0    = sum(w1.*z_i0)+b1;

% The prediction of y(1)
z_i1  = mysigmoid(w0+b0);
y1    = sum(w1.*z_i1)+b1;

% Partial derivatives of y(x) w.r.t weights and biases
dydw0 = x*w1.*mySzp;
dydw1 = mySzp;
dydb0 = w1.*mySzp;
dydb1 = 1;

% Partial derivatives of dy/dx w.r.t weights and biases
dypdw0 = w1.*mySzp + x*w1.*(w0).^2.*mySzpp;
dypdw1 = w0.*mySzp;
dypdb0 = w1.*w0.*mySzpp;
dypdb1 = 0;

% Partial derivatives of d2y/dx2 w.r.t weights and biases
dyppdw0 = 2*w1.*w0.*mySzpp + x*w1.*(w0).^2.*mySzppp;
dyppdw1 = w0.^2.*mySzpp;
dyppdb0 = w1.*w0.^2.*mySzppp;
dyppdb1 = 0;

% Partial derivatives of y(0) w.r.t weights and biases
dy0dw0 = 0*one;
dy0dw1 = z_i0; %mysigmoid(b0)
dy0db0 = w1.*(mysigmoid(b0).*(1-mysigmoid(b0)));
dy0db1 = 1;

% Partial derivatives of y(1) w.r.t weights and biases
dy1dw0 = w1.*(mysigmoid(w0+b0).*(1-mysigmoid(w0+b0)));
dy1dw1 = z_i1; %mysigmoid(w0+b0)
dy1db0 = w1.*(mysigmoid(w0+b0).*(1-mysigmoid(w0+b0)));
dy1db1 = 1;

% Computing the updates for the weights and biases based on the cost
% function mean((c*ypp+cp*yp-f)^2)+(y(0))^2+(y(1))^2
Ly     = ci*ypp+cpi.*yp;        % The differential operator
By     = [y0;y1];               % The boundary operator

dLydw0  = ci*dyppdw0+cpi*dypdw0;
dBy0dw0 = dy0dw0;
dBy1dw0 = dy1dw0;

dLydw1  = ci*dyppdw1+cpi*dypdw1;
dBy0dw1 = dy0dw1;
dBy1dw1 = dy1dw1;

dLydb0  = ci*dyppdb0+cpi*dypdb0;
```

```
    dBy0db0 = dy0db0;
    dBy1db0 = dy1db0;

    dLydb1  = ci*dyppdb1+cpi*dypdb1;
    dBy0db1 = dy0db1;
    dBy1db1 = dy1db1;

    cLyp   = 2*(Ly-fi);     % Prime of the cost function part for Ly
    cBy0p  = 2*(y0);        % Prime of the cost function part for By
    cBy1p  = 2*(y1);

    dcdw0  = cLyp*dLydw0+cBy0p*dBy0dw0+cBy1p*dBy1dw0;
    dcdw1  = cLyp*dLydw1+cBy0p*dBy0dw1+cBy1p*dBy1dw1;
    dcdb0  = cLyp*dLydb0+cBy0p*dBy0db0+cBy1p*dBy1db0;
    dcdb1  = cLyp*dLydb1+cBy0p*dBy0db1+cBy1p*dBy1db1;

    % Updating weights and biases
    w0 = w0-lr*dcdw0;
    w1 = w1-lr*dcdw1;

    b0 = b0-lr*dcdb0;
    b1 = b1-lr*dcdb1;

    % Cost function evaluated at the random data point x
    costi = mean((Ly-f).^2)+By(1)^2+By(2)^2;

    % Cost function evaluated at ALL data points x
%     c = costfunction(input,w0,b0,w1,b1);

    %disp(['c(',sprintf('%03d',bi),')=',sprintf('%3.2e',c)]);
    bi = bi+1;
end
bi = bi-1; % Roll back to bi when critiera was met
params = [w0;b0;w1;b1]; % Compose the parameter array to be returned
end

% Sigmoid basis function
function y = mysigmoid(x)
    y = 1./(1+exp(-x));
end

% The prediction y=f(x,w,b) it's derivative and initial value
function [y,dydx,d2ydx2,y0,y1] = prediction(x,w0,b0,w1,b1)
    % Preprocessing for vectorisation of the output
    Nn = length(w0);
    N  = length(x);

    W0 = repmat(w0,1,N);
    B0 = repmat(b0,1,N);
    W1 = repmat(w1,1,N);
    X  = repmat(x,Nn,1);

    % The prediction of y(x)
    z_i   = mysigmoid(W0.*X+B0);     % NnxN matrix
    mySzp = mysigmoid(W0.*X+B0).*... % Prime of the sigmoid at z_i
             (1-mysigmoid(W0.*X+B0));
    mySzpp = mySzp.*(1-2*mysigmoid(w0*x+b0));

    y     = sum(W1.*z_i)+b1;        % The prediction at current batch
```

```
    % The prediction of dy/dx
    dydx  = sum(W1.*W0.*mySzp);

    % The prediction of d2y/dx2
    d2ydx2 = sum(w1.*w0.^2.*mySzpp);

    % The prediction of y(0). N.B. scalar value
    y0    = sum(w1.*mysigmoid(b0))+b1;

    % The prediction of y(1). N.B. scalar value
    y1    = sum(w1.*mysigmoid(w0+b0))+b1;
end

% The cost function for ALL points
% mean((c*ypp+cp*yp-f)^2)+(y(0))^2+(y(1))^2
function c = costfunction(x,c,cp,f,w0,b0,w1,b1)
    [~,yp,ypp,y0,y1] = prediction(x,w0,b0,w1,b1);
    c = mean((c.*ypp+cp.*yp-f).^2)+y0^2+y1^2;
end
```