Edits to the solver

```
%========================================================================
% EDIT THE ORIGINAL CODE TO
% 1. OUTPUT OBJECTIVE, SENSITIVITIES, FEATURES AND AUGMENTATION
% 2. TAKE A NEURAL NETWORK AND TRUTH DATA AS ADDITIONAL INPUTS
%========================================================================
function [obj, sens, features, beta] = RHT(T_inf, nPoints, dt, nIter,...
                                           tol, verbose, nn, data)
```

```
%========================================================================
% DECLARE AN AUGMENTATION FIELD "BETA"
%========================================================================
beta = ones(nPoints, 1);
%========================================================================
```

Sensitivities need not be calculated if finite difference derivatives are being used in the neural network training

```
for iter = 1:nIter

    %==================================================================
    % IN THE BEGINNING OF EVERY ITERATION,
    % CALCULATE FEATURES AND AUGMENTATION
    %==================================================================
    features       = zeros(nPoints, 2);
    features(:,1) = T(:,1) / T_inf;
    features(:,2) = y(1,:);

    if ~isempty(nn)
        beta = nn.predict(features);
    end
    %==================================================================
```

```
%========================================================================
% AT THE END OF THE SIMULATION:
% IF SOME DATA IS PROVIDED, CALCULATE OBJECTIVE AND SENSITIVITIES
% (ANY TECHNIQUE MAY BE CHOSEN FOR SENSITIVITY EVALUATION
%  HERE WE HAVE CHOSEN ANALYTICAL GRADIENTS)
%========================================================================
if ~isempty(data)
    psi  = (jacT.')\(2.0*(T - data)/size(T,1));
    sens = -(jacbeta.')*psi;
    obj  = sum((T - data).^2 / size(T,1));
    [redacted]
end
%========================================================================
```

Running the framework

```
function NN = FIML(nTrainIters, nFIMLIters, stepSize, nHiddenLayerNodes,...
                   solver_dict, solver_weights, fd_step)
```

Function signature (not a part of the script)

```
NN = FIML(1000, 1000, 1e-3, [7; 7], {@solver1, @solver2}, [0.5, 0.5], 0.0);

function [obj, sens, features, beta] = solver1(NN)

    T_inf = 5;

    data = dlmread(strcat("True/solution_", string(T_inf), ".txt"));
    [obj, sens, features, beta] = RHT(T_inf, 129, 1e-2, 1000, 1e-8, 0, ...
                                      NN, data);

end


function [obj, sens, features, beta] = solver2(NN)

    T_inf = 10;

    data = dlmread(strcat("True/solution_", string(T_inf), ".txt"));
    [obj, sens, features, beta] = RHT(T_inf, 129, 1e-2, 1000, 1e-8, 0, ...
                                      NN, data);

end
```

- If `fd_step` is 0.0, then the FIML routine assumes the derivatives to be coming from the solver, else it obtains them using finite differences with `fd_step` as the step size
- Several problems can be used at once to obtain an augmentation, the total objective function then being a weighted sum of the individual objective functions of each problem (`solver1`, `solver2`, … in this example with weights 0.5 each)
- `nTrainIters` specifies the initial training required to create a baseline augmentation NN
- `nFIMLIters` specifies the number of optimization iterations for the FIML procedure
- `stepSize` refers to the optimization step size
- `nHiddenLayerNodes` specifies the NN hidden layer structure
- Weights are stored in the file “`NN_weights.txt`”

NOTE: The inverse problem here for radiative heat transfer is ill-posed and can lead to augmentations which might not work on all cases

Prediction

```
T_inf = 10;
NN = NeuralNetwork(2, [7; 7]);
NN = NN.load(); % Assignment is important to update NN
data = dlmread(strcat("True/solution_", string(T_inf), ".txt"));
[obj, sens, features, beta] = RHT(T_inf, 129, 1e-2, 1000, 1e-8, 0,...
                                     [], []);
hold on
[obj, sens, features, beta] = RHT(T_inf, 129, 1e-2, 1000, 1e-8, 0,...
                                     NN, data);
legend('Baseline','Augmented','Data')
```