FEM_1D

October 18, 2023

```
[ ]: # Restart the kernel
     import os
     current_dir ='../../'
     os.chdir(current_dir)
```

```
[ ]: from src.codes.basic import *
     from src.codes.utils import *
     from src.codes.base_classes import Base_class_fem_heat_conduction
     from src.codes.reductor.rom_class import FEM_solver_rom_ecsw
     from src.codes.algorithms.ecsw import ecsw_red
```

0.0.1 class for data (geometry, material property, mesh)

```
[ ]: n_ref = np.array([400, 100], dtype=int)
     w = np.array([ 0.4, 0.1])
     L = np.sum(w)

     # Create arrays of zeros and ones
     zeros_array = np.zeros((1, n_ref[0]))
     ones_array = np.ones((1, n_ref[1]))

     # Concatenate along the second axis (axis=1)
     mat_layout = np.concatenate((zeros_array, ones_array), axis=1)
     src_layout = np.concatenate((zeros_array, ones_array), axis=1)
```

```
[ ]: fdict = {}

     tune = 1

     cond_list = []
     cond_list.append(lambda T,mu: 1.05*mu*tune + 2150/(T-73.15))
     cond_list.append(lambda T,mu: mu*tune*7.51 + 2.09e-2*T - 1.45e-5*T**2 + 7.
      ↪67e-9*T**3)
     fdict["cond"] = cond_list

     dcond_list = []
     dcond_list.append(lambda T,mu: -2150/(T-73.15)**2 )
     dcond_list.append(lambda T,mu: 2.09e-2 - 2*1.45e-5*T + 3*7.67e-9*T**2)
     fdict["dcond"] = dcond_list

     qext_list = []
     qext_list.append( lambda T,mu: 35000.0 + 0.*T)
     qext_list.append( lambda T,mu: 0.0 + 0.*T)
     fdict["qext"] = qext_list
```

[ ]:
```
bc = {}
bc['x_min']={'type':'refl','value':np.nan}
bc['x_max']={'type':'dirichlet','value':273.15+300.}
```

[ ]:
```
print(mat_layout.flatten().shape[0])
```

```
500
```

[ ]:
```
class probdata:

    def __init__(self, bc, cond_layout, qext_layout, fdict, nref, L, mu,␣
 ↪pb_dim=1):

        self.dim_ = pb_dim

        # refine the mesh and update material and source layouts
        self.cell2mat_layout = cond_layout.flatten()
        self.cell2src_layout = qext_layout.flatten()

        ## change this mapping if needed.
        self.fdict = fdict

        # mesh data cells
        self.ncells = [None] * pb_dim
        self.npts = [None] * pb_dim
        self.deltas = [None] * pb_dim
        self.xi=[]

        for i in range(pb_dim):

            self.ncells[i] = self.cell2mat_layout.shape[i]
            self.npts[i] = self.ncells[i]+1
            self.xi.append(np.linspace(0,L,self.npts[i]))
            self.deltas[i] = L/self.ncells[i]

        self.n_verts = self.npts[0]

        # Create nodal connectivity for the continuous Finite Element Method␣
 ↪(cFEM)
```

```
        self.connectivity()

        # Store parameter value
        self.mu = mu

        # Store the dirichlet nodes if any
        handle_boundary_conditions(self, bc)

        # Determining the global equation numbers based on dirichlet nodes and␣
↪storing in class
        get_glob_node_equation_id(self, self.dir_nodes)

        # Get global node numbers and equation IDs for the current element
        self.glob_node_eqnId = []
        self.glob_node_nonzero_eqnId = []
        self.local_node_nonzero_eqnId = []
        self.Le = []
        self.global_indices = []
        self.local_indices = []

        for i in range(self.n_cells):
            get_element_global_nodes_and_nonzero_eqnId(self, i, self.node_eqnId)


    def connectivity(self):
        """
        Define nodal connectivity for each cell in the mesh.
        """

        # Initialize the connectivity array
        self.n_cells = self.ncells[0]

        self.gn = np.zeros((self.n_cells,2**self.dim_),dtype=int)

        # Loop over all cells to define their nodal connectivity

        for iel in range(self.n_cells):
            # For each cell, define the left and right nodes
            self.gn[iel, 0] = iel
            self.gn[iel, 1] = iel + 1
```

0.0.2 Simulate FOS

[ ]:
```
random.seed(25)
params = np.r_[1.:4.0:0.01]
quad_deg = 3
N_snap = 15 # Training Snapshots
NL_solutions = []
param_list = []
K_mus = []
q_mus = []
#
```

[ ]:
```
fig, ax = plt.subplots()

for i in range(N_snap):
    print(f"\n\n\n Snap {i} \n\n\n")
    param = random.choice(params) # Choose from parameter list
    param_list.append(param)

    if i==0:
        d = probdata(bc, mat_layout, src_layout, fdict, n_ref, L, param,␣
 ↪pb_dim=1)
        FOS = Base_class_fem_heat_conduction(d,quad_deg)
    else:
        FOS.mu = param
    T_init = np.zeros(d.n_verts) + 273.15
    NL_solution_p, Ke, rhs_e, mask = solve_fos(FOS, T_init)
    NL_solutions.append(NL_solution_p.flatten())
    K_mus.append(Ke)
    q_mus.append(rhs_e)
    plot1D(d.xi[0], NL_solution_p, ax=ax)

plt.show()
```

```



 Snap 0



initial residual = 5121928.621671218

iter 0, NL residual=767085.6491309204, delta=662.4431892840048
iter 1, NL residual=9170.357065593953, delta=385.5108800528815
iter 2, NL residual=1.7012921798759322, delta=47.761081721791776
iter 3, NL residual=0.00014300922331612415, delta=0.4939698922092143
iter 4, NL residual=3.747633245446698e-08, delta=5.043698276272934e-05
```

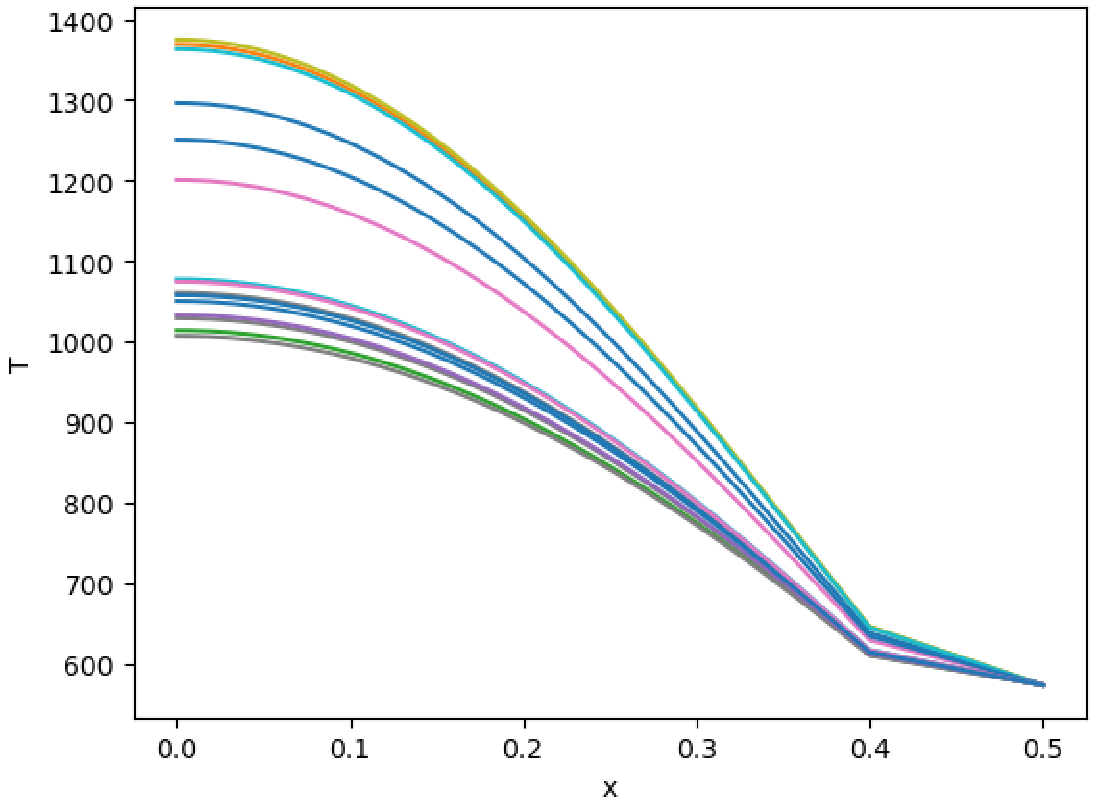
```
Convergence !!!



 Snap 1



initial residual = 8927594.816306168

iter 0, NL residual=689552.1729995243, delta=572.5093456131659
iter 1, NL residual=2745.092817875878, delta=219.0839682201138
iter 2, NL residual=0.11740086081809227, delta=12.867033109058834
iter 3, NL residual=1.322137292278698e-06, delta=0.033505917133006706
Convergence !!!



 Snap 2



initial residual = 5076891.152095023

iter 0, NL residual=768837.1730194121, delta=664.177102655932
iter 1, NL residual=9351.75919686822, delta=388.73169783903813
iter 2, NL residual=1.7894910225343383, delta=48.66629237381616
iter 3, NL residual=0.00015344706753081319, delta=0.5132885466608292
iter 4, NL residual=4.186033386358839e-08, delta=5.446938658399251e-05
Convergence !!!



 Snap 3



initial residual = 5752453.196434753

iter 0, NL residual=745933.8641306281, delta=640.7647053680371
iter 1, NL residual=7098.512000979814, delta=345.0228746482783
iter 2, NL residual=0.898064849115158, delta=37.09512612174089
iter 3, NL residual=5.513589385025571e-05, delta=0.2946026049743646
iter 4, NL residual=4.3996983819637134e-08, delta=1.787251706729402e-05
Convergence !!!
```

```
 Snap 4



initial residual = 10526424.997143136

iter 0, NL residual=675166.6712061554, delta=550.7247143538872
iter 1, NL residual=1934.54469307302, delta=181.75707377396185
iter 2, NL residual=0.059834621801775865, delta=8.28806744908951
iter 3, NL residual=7.519730544391404e-07, delta=0.013517593891798631
Convergence !!!



 Snap 5



initial residual = 5166966.091254803

iter 0, NL residual=765370.5433315713, delta=660.7370330899505
iter 1, NL residual=8994.15413939637, delta=382.3385610125082
iter 2, NL residual=1.6185682450180532, delta=46.87747736615693
iter 3, NL residual=0.00013332346554841178, delta=0.47547473582844363
iter 4, NL residual=3.644970976789458e-08, delta=4.6720817989553425e-05
Convergence !!!



 Snap 6



initial residual = 6833352.469706209

iter 0, NL residual=720027.2442340987, delta=611.8315425735594
iter 1, NL residual=4876.234667046013, delta=290.82204405609946
iter 2, NL residual=0.3811048188813458, delta=24.981846049939954
iter 3, NL residual=1.21206747569353e-05, delta=0.13105312366043442
iter 4, NL residual=4.7314486497405764e-08, delta=3.506894600883289e-06
Convergence !!!



 Snap 7
```

```
initial residual = 10729093.611704037

iter 0, NL residual=673675.8393603169, delta=548.298447252672
iter 1, NL residual=1858.062881590188, delta=177.7557139115829
iter 2, NL residual=0.05546813411120196, delta=7.864352875408977
iter 3, NL residual=7.115712826041075e-07, delta=0.012127677395571376
Convergence !!!



 Snap 8



initial residual = 9310413.310179194

iter 0, NL residual=685612.4785016805, delta=566.8071763300414
iter 1, NL residual=2510.198372308858, delta=209.0923294760628
iter 2, NL residual=0.09856076990261155, delta=11.52875630385647
iter 3, NL residual=1.107930630013072e-06, delta=0.026718552111298133
Convergence !!!



 Snap 9



initial residual = 10098569.033128012

iter 0, NL residual=678528.6806779823, delta=556.0685736058164
iter 1, NL residual=2112.2621018169007, delta=190.68910381512455
iter 2, NL residual=0.07068032729089926, delta=9.280127805713516
iter 3, NL residual=8.520532759882821e-07, delta=0.017074807417360425
Convergence !!!



 Snap 10



initial residual = 9377969.514989872

iter 0, NL residual=684953.9672731262, delta=565.8362489927612
iter 1, NL residual=2471.876819302324, delta=207.4053494397097
iter 2, NL residual=0.09565677862795227, delta=11.311016496813824
iter 3, NL residual=1.0749754256296618e-06, delta=0.025688382893988504
```

```
Convergence !!!


 Snap 11


initial residual = 9985975.358401213

iter 0, NL residual=679465.7868643696, delta=557.5281823128915
iter 1, NL residual=2163.084610370689, delta=193.1560846523458
iter 2, NL residual=0.07396280809490598, delta=9.565476867251123
iter 3, NL residual=8.783087871716462e-07, delta=0.018176821963374647
Convergence !!!


 Snap 12


initial residual = 6225346.62820351

iter 0, NL residual=733319.6013131307, delta=627.0645126898069
iter 1, NL residual=5971.497074636872, delta=319.3268031685115
iter 2, NL residual=0.5982843945952422, delta=31.03650528786388
iter 3, NL residual=2.795157938847795e-05, delta=0.20448214469639273
iter 4, NL residual=4.289395132461142e-08, delta=8.57928612500119e-06
Convergence !!!


 Snap 13


initial residual = 8995151.021100318

iter 0, NL residual=688830.0147533409, delta=571.4773678650755
iter 1, NL residual=2701.3170936161555, delta=217.26538997996894
iter 2, NL residual=0.11375311302388065, delta=12.617232387874528
iter 3, NL residual=1.2740594939137731e-06, delta=0.03217940936575564
Convergence !!!


 Snap 14
```

```
initial residual = 9558119.394498087

iter 0, NL residual=683247.9076796419, delta=563.2957724335802
iter 1, NL residual=2373.855247403409, delta=203.01212062295397
iter 2, NL residual=0.08844268749986504, delta=10.75504114127838
iter 3, NL residual=9.993564253652602e-07, delta=0.023151854820810145
Convergence !!!
```



[ ]:
```
NLS = np.asarray(NL_solutions)
np.shape(NLS)
```

[ ]:
```
(30, 501)
```

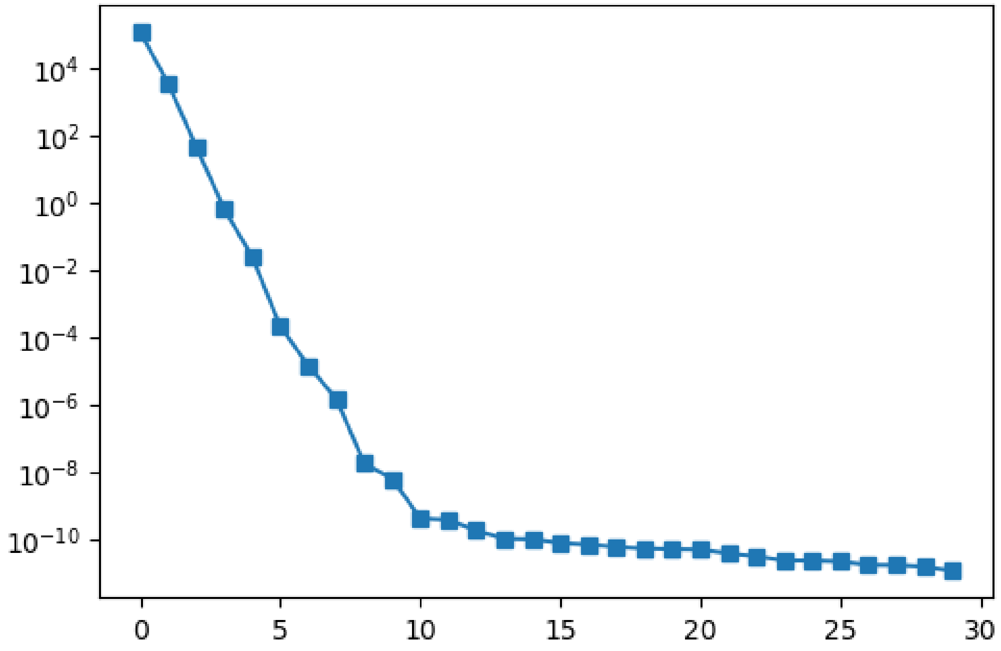
0.0.3 ECSW Hyper-reduction

Step 1: Perform SVD on the snapshots (calculate $\mathbb{V}(=\mathbb{W})$):

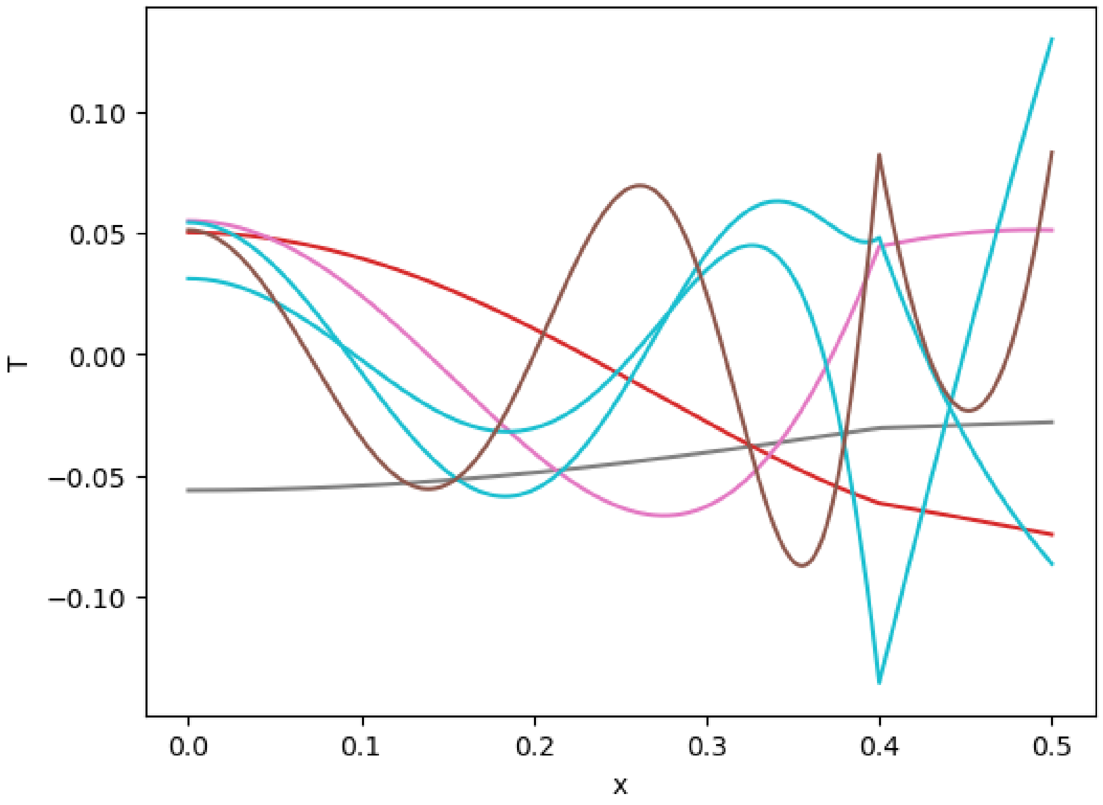
[ ]:
```
n_sel = 6
U, S, Vt = np.linalg.svd(np.transpose(NLS), full_matrices=False)
```

```
V_sel = U[:, :n_sel]
P_sel = V_sel[mask,:]@np.transpose(V_sel[mask,:])
```

[ ]:
```
plt.figure(figsize = (6,4))
plt.semilogy(S,'s-')
plt.show()
```



[ ]:
```
fig, ax = plt.subplots()

for i in range(n_sel):
    plot1D(d.xi[0],V_sel[:,i],ax=ax)

plt.show()
```



ECSW

```
[ ]: tic_h_setup_b = time.time()
     tol = 1e-8
     xi, residual = ecsw_red(d, V_sel, d.Le, K_mus, q_mus, n_sel, N_snap,␣
      ↪mask,NL_solutions,tol=tol)
     toc_h_setup_b = time.time()
```

```
d:\D\ONEDRIVE\OneDrive - Texas A&M University\TAMU_MATERIALS\POSTDOC\HYPERREDUCT
ION\SUPARNO\Hyperreduction_tamids\pyHyperRom\src\codes\algorithms\nnls_scipy.py:
135: LinAlgWarning: Ill-conditioned matrix (rcond=7.7001e-19): result may not be
accurate.
  s[P] = solve(AtA[P_ind[:, None], P_ind[None, :]], Atb[P],
d:\D\ONEDRIVE\OneDrive - Texas A&M University\TAMU_MATERIALS\POSTDOC\HYPERREDUCT
ION\SUPARNO\Hyperreduction_tamids\pyHyperRom\src\codes\algorithms\nnls_scipy.py:
135: LinAlgWarning: Ill-conditioned matrix (rcond=5.27153e-19): result may not
be accurate.
  s[P] = solve(AtA[P_ind[:, None], P_ind[None, :]], Atb[P],
d:\D\ONEDRIVE\OneDrive - Texas A&M University\TAMU_MATERIALS\POSTDOC\HYPERREDUCT
ION\SUPARNO\Hyperreduction_tamids\pyHyperRom\src\codes\algorithms\nnls_scipy.py:
145: LinAlgWarning: Ill-conditioned matrix (rcond=5.97902e-19): result may not
be accurate.
```

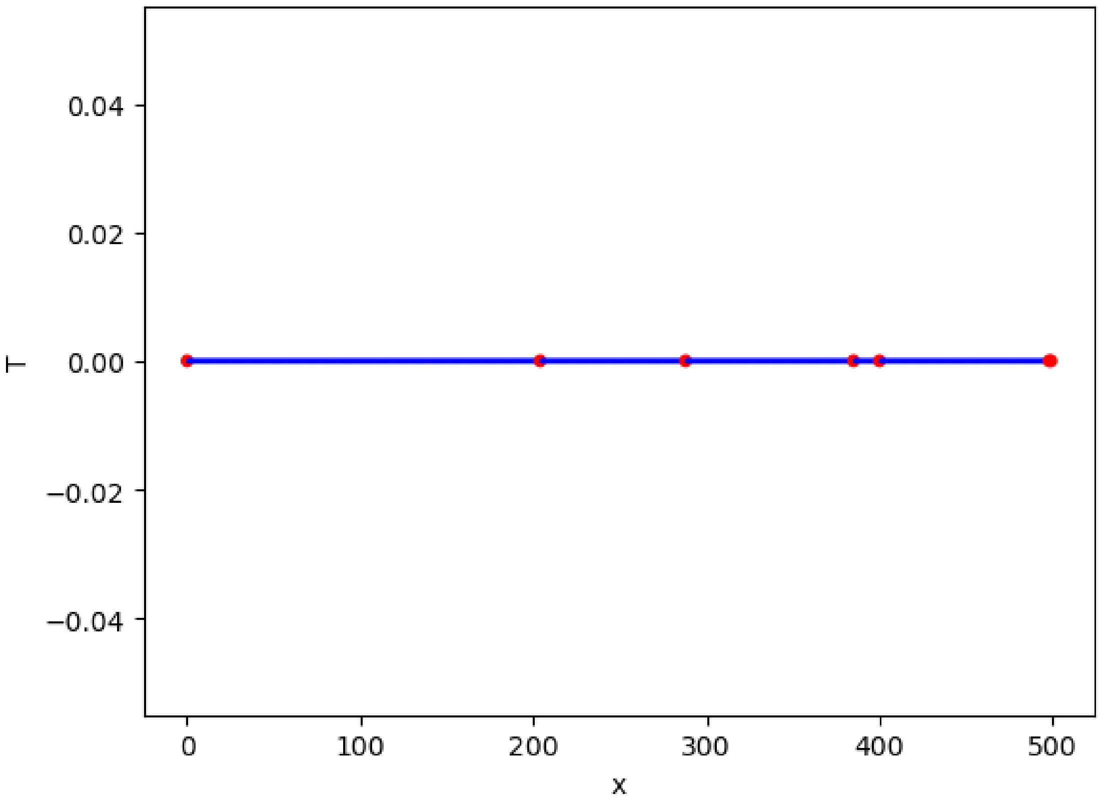
```
    s[P] = solve(AtA[np.ix_(P, P)], Atb[P], assume_a='sym',
```

[ ]:
```
print(f"this is the residual from fnnls: {residual}")
```

```
this is the residual from fnnls: 0.0004941976266995773
```

[ ]:
```
colors = ['red' if value > 0 else 'blue' for value in xi]
sizes = [15 if value > 0 else 1 for value in xi]
```

[ ]:
```
plot1D(np.arange(d.ncells[0]), np.zeros_like(xi), scattr=True, clr=colors,␣
 ↪sz=sizes)
```



[ ]:
```
print(f"Fraction of total elements active in the ROM: {len(xi[xi>0])*100/
 ↪len(xi)}%")
```

```
Fraction of total elements active in the ROM: 1.4%
```

0.0.4 ROM Simulation

[ ]:
```
params_rm = params[~np.isin(params,param_list)]
param_rom = random.choice(params_rm)
```

[ ]:
```python
# Define the data-class

d_test = probdata(bc, mat_layout, src_layout, fdict, n_ref, L, param_rom,␣
 ↪pb_dim=1)
FOS_test = Base_class_fem_heat_conduction(d_test,quad_deg)
ROM = FEM_solver_rom_ecsw(d_test, quad_deg)
```

[ ]:
```python
# Initial guess

T_init_fos = np.zeros(FOS_test.n_nodes) + 273.15
T_init_rom = np.transpose(V_sel)@T_init_fos # crucial to ensure the initial␣
 ↪guess is contained in the reduced subspace
```

[ ]:
```python
# Time taken to perform a FO simulation with the current parameter value

tic_fos = time.time()
NL_solution_p_fos_test, _, _, _, = solve_fos(FOS_test,T_init_fos)
toc_fos = time.time()
```

```
initial residual = 5009334.94774502

iter 0, NL residual=771535.3140879221, delta=666.8318030968067
iter 1, NL residual=9634.06387551095, delta=393.65657924043455
iter 2, NL residual=1.9330011677064427, delta=50.06607111914047
iter 3, NL residual=0.00017067080393691272, delta=0.5439019874768509
iter 4, NL residual=3.648835779501898e-08, delta=6.118003322755732e-05
Convergence !!!
```

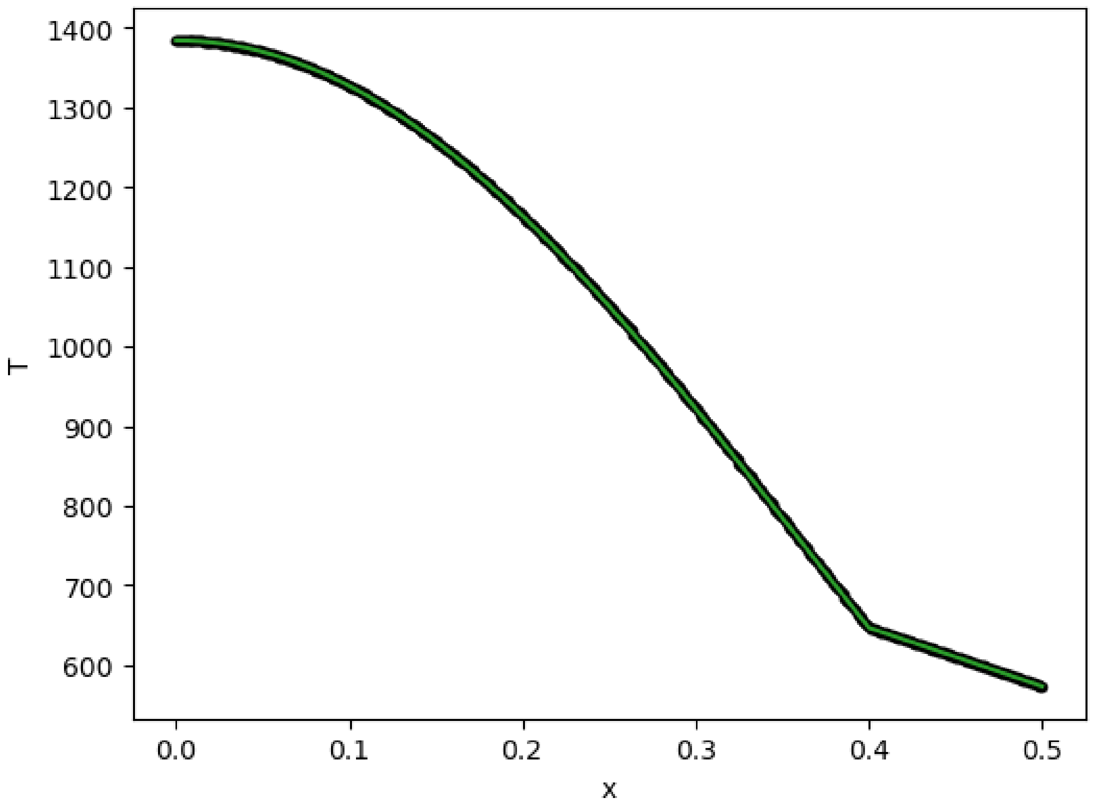
[ ]:
```python
# Time taken to simulate a ROM without hyper-reduction

tic_rom_woh = time.time()
NL_solution_p_reduced_woh = ROM.solve_rom(T_init_rom,np.ones_like(xi),V_sel)
toc_rom_woh = time.time()
```

```
initial residual = 866788.9468471474


c:\Users\supar\anaconda3\lib\site-
packages\scipy\sparse\linalg\_dsolve\linsolve.py:229: SparseEfficiencyWarning:
spsolve requires A be CSC or CSR matrix format
  warn('spsolve requires A be CSC or CSR matrix format',

iter 0, NL residual=2290.462188401109, delta=191.01055204185923
iter 1, NL residual=101.28588404693845, delta=2513.576322861897
iter 2, NL residual=1.5455080579915441, delta=406.79273223471495
iter 3, NL residual=0.0004676474394008844, delta=5.808707667995811
iter 4, NL residual=6.25425809628755e-08, delta=0.0010752988808156662
Convergence !!!
```

[ ]:
```
# Time taken to simulate a ROM *with* hyper-reduction

tic_rom = time.time()
NL_solution_p_reduced = ROM.solve_rom(T_init_rom,xi,V_sel)
toc_rom = time.time()
```

```
initial residual = 902599.3973434794

iter 0, NL residual=4864.571504259302, delta=185.58455914325793
iter 1, NL residual=203.5158545384358, delta=2489.5768759559473
iter 2, NL residual=3.0707451523490477, delta=405.33983256543837
iter 3, NL residual=0.0005373151008378906, delta=5.982501672370759
iter 4, NL residual=3.0228780053087476e-09, delta=0.0008764872878771882
Convergence !!!
```

[ ]:
```
sol_red = V_sel@NL_solution_p_reduced.reshape(-1,1)  #+pca.mean_.reshape(-1,1)

fig, ax = plt.subplots()

plot1D(d_test.xi[0], sol_red, ax=ax)
plot1D(d_test.xi[0], NL_solution_p_fos_test, ax=ax, scattr=True, clr='k', sz=10)

plt.show()

print(f"RMS_error is {np.linalg.norm(sol_red-NL_solution_p_fos_test.
 ↪reshape(-1,1))*100/np.linalg.norm(NL_solution_p_fos_test.reshape(-1,1))} %")
```



```
RMS_error is 3.303349741470281e-07 %
```

[ ]:
```
print(f"\n\nROM Error without hyperreduction is {np.linalg.
↪norm(V_sel@NL_solution_p_reduced_woh.reshape(-1,1)-NL_solution_p_fos_test.
↪reshape(-1,1))*100/np.linalg.norm(NL_solution_p_fos_test.reshape(-1,1))} %")
```

```
ROM Error without hyperreduction is 8.10756125208484e-08 %
```

Speedups

[ ]:
```
fos_sim_time = toc_fos - tic_fos
rom_sim_time_woh = toc_rom_woh - tic_rom_woh
rom_sim_time = toc_rom - tic_rom
```

[ ]:
```
print(f"speedup without hyperreduction:{fos_sim_time/rom_sim_time_woh}")
print(f"speedup with hyperreduction:{fos_sim_time/(rom_sim_time)}")
# h_total_setup_time = (toc_h_setup_b+toc_h_setup_a) -␣
↪(tic_h_setup_b+tic_h_setup_a) #this is one time
```

```
speedup without hyperreduction:1.016798189458441
speedup with hyperreduction:50.849844490898896
```