```
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
```

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim.lr_scheduler import StepLR

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import os
from pyDOE import lhs
import shutil

import argparse
```

实验报告

姓名：周睿涵

学号：2018302010038

1、基础网络结构

网络结构分为普通的和带残差的全连接前馈神经网络(Fully-Connected Feedforward Network)，选择其中一种来实现。

DNN

```
def activation(name):
    if name in ['tanh','TANH']:
        return nn.Tanh()
    elif name in ['relu', 'RELU']:
        return nn.ReLU(inplace=True)
    elif name in ['leaky_relu', 'LeakyReLU']:
        return nn.LeakyReLU(inplace=True)
    elif name in ['sigmoid', 'SIGMOID']:
        return nn.Sigmoid()
    elif name in ['softplus', 'SOFTPLUS']:
        return nn.Softplus()
    else:
        raise ValueError(f'unknown activation function: {name}')
```

```
class DNN(nn.Module):
    """Deep Neural Network"""

    def __init__(self, dim_in, dim_out, dim_hidden, hidden_layers,
                 act_name='tanh', init_name=None):
        super().__init__()
```

```
        model = nn.Sequential()

        model.add_module('fc0', nn.Linear(dim_in, dim_hidden, bias=True))
        model.add_module('act0', activation(act_name))

        for i in range(1, hidden_layers):
            model.add_module(f'fc{i}', nn.Linear(dim_hidden, dim_hidden,
bias=True))
            model.add_module(f'act{i}', activation(act_name))

        model.add_module(f'fc{hidden_layers}', nn.Linear(dim_hidden, dim_out,
bias=True))

        self.model = model

        if init_name is not None:
            self.init_weight(init_name)

    def init_weight(self, name):
        if name == 'xavier_normal':
            nn_init = nn.init.xavier_normal_
        elif name == 'xavier_uniform':
            nn_init = nn.init.xavier_uniform_
        elif name == 'kaiming_normal':
            nn_init = nn.init.kaiming_normal_
        elif name == 'kaiming_uniform':
            nn_init =  nn.init.kaiming_uniform_
        else:
            raise ValueError(f'unknown initialization function: {name}')

        for param in self.parameters():
            if len(param.shape) > 1:
                nn_init(param)

    def forward(self, x):
        return self.model(x)

    def model_size(self):
        n_params = 0
        for param in self.parameters():
            n_params += param.numel()
        return n_params
```

ResDNN

```
class ResBlock(nn.Module):
    """Residual Block """

    def __init__(self, dim_in, dim_out, dim_hidden, act_name='tanh'):
        super().__init__()

        assert(dim_in == dim_out)

        block = nn.Sequential()
        block.add_module('act0', activation(act_name))
        block.add_module('fc0', nn.Linear(dim_in, dim_hidden, bias=True))
        block.add_module('act1', activation(act_name))
```

```
        block.add_module('fc1', nn.Linear(dim_hidden, dim_out, bias=True))
        self.block = block

    def forward(self, x):
        identity = x
        out = self.block(x)
        return identity + out
```

```
class ResDNN(nn.Module):
    """Residual Deep Neural Network """

    def __init__(self, dim_in, dim_out, dim_hidden, res_blocks, act_name='tanh',
init_name='kaiming_normal'):
        super().__init__()

        model = nn.Sequential()
        model.add_module('fc_first', nn.Linear(dim_in, dim_hidden, bias=True))

        for i in range(res_blocks):
            res_block = ResBlock(dim_hidden, dim_hidden, dim_hidden,
act_name=act_name)
            model.add_module(f'res_block{i+1}', res_block)

        model.add_module('act_last', activation(act_name))
        model.add_module('fc_last', nn.Linear(dim_hidden, dim_out, bias=True))

        self.model = model

        if init_name is not None:
            self.init_weight(init_name)

    def init_weight(self, name):
        if name == 'xavier_normal':
            nn_init = nn.init.xavier_normal_
        elif name == 'xavier_uniform':
            nn_init = nn.init.xavier_uniform_
        elif name == 'kaiming_normal':
            nn_init = nn.init.kaiming_normal_
        elif name == 'kaiming_uniform':
            nn_init =  nn.init.kaiming_uniform_
        else:
            raise ValueError(f'unknown initialization function: {name}')

        for param in self.parameters():
            if len(param.shape) > 1:
                nn_init(param)

    def forward(self, x):
        return self.model(x)

    def model_size(self):
        n_params = 0
        for param in self.parameters():
            n_params += param.numel()
        return n_params
```

2、Burgers方程

考虑一维Burgers方程：

$$\begin{cases} u_t + uu_x - \frac{0.01}{\pi}u_{xx} = 0, \quad x \in [-1,1], \ t \in [0,1] \\ u(0,x) = -\sin(\pi x), \\ u(t,-1) = u(t,1) = 0. \end{cases}$$

2.1、问题描述

```
class Problem_Burgers(object):
    """ Description of Burgers Equation"""
    def __init__(self,domain=(0,1,-1,1)):
        self.domain=domain

    def __repr__(self):
        return f'{self.__doc__}'

    def iv(self,x):
        iv=-np.sin(np.pi*x[:,[1]])
        return iv

    def bv(self,x):
        bv=np.zeros_like(x[:,[0]])
        return bv

    def epilson(self):
        epilson=0.01/np.pi
        return epilson
```

```
problem=Problem_Burgers()
```

2.2、数据集生成

```
class Trainset_Burgers(object):
    def __init__(self,problem,*args,**kwargs):
        self.problem=problem
        self.domain=problem.domain
        self.args=args
        self.method=kwargs['method']

    def __call__(self,plot=False,verbose=None):
        if self.method=='uniform':
            n_t,n_x=self.args[0],self.args[1]
            p,p_bc,p_ic=self._uniform_sample(n_t,n_x)
        elif self.method=='lhs':
            n,n_bc,n_ic=self.args[0],self.args[1],self.args[2]
            p,p_bc,p_ic=self._lhs_sample(n,n_bc,n_ic)

        bv=self.problem.bv(p_bc)
        iv=self.problem.iv(p_ic)

        if plot:
            fig = plt.figure()
```

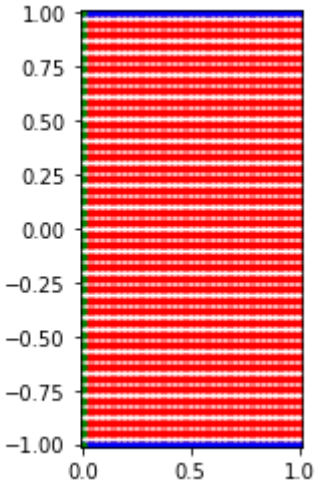
```
            ax = fig.add_subplot(111)
            ax.scatter(p[:, 0], p[:, 1], facecolor='r', s=10)
            ax.scatter(p_bc[:, 0], p_bc[:, 1], facecolor='b', s=10)
            ax.scatter(p_ic[:, 0], p_ic[:, 1], facecolor='g', s=10)
            ax.set_xlim(-0.01, 1.01)
            ax.set_ylim(-1.01, 1.01)
            ax.set_aspect('equal')
            plt.show()

        if verbose=='tensor':
            p=torch.from_numpy(p).float()
            p_bc=torch.from_numpy(p_bc).float()
            p_ic=torch.from_numpy(p_ic).float()
            bv=torch.from_numpy(bv).float()
            iv=torch.from_numpy(iv).float()
            return p,p_bc,p_ic,bv,iv
        return p,p_bc,p_ic,bv,iv
    def _uniform_sample(self,n_t,n_x):
        t_min,t_max,x_min,x_max=self.domain
        t = np.linspace(t_min, t_max, n_t)
        x = np.linspace(x_min, x_max, n_x)
        t,x = np.meshgrid(t,x)
        tx=np.hstack((t.reshape(t.size,-1),x.reshape(x.size,-1)))

        mask_ic=(tx[:,0]-t_min)==0
        mask_bc=(tx[:,1]-x_min)*(x_max-tx[:,1])==0
        p_ic=tx[mask_ic]
        p_bc=tx[mask_bc]
        p=tx[np.logical_not(mask_ic,mask_bc)]
        return p,p_bc,p_ic

    def _lhs_sample(self,n,n_bc,n_ic):
        t_min,t_max,x_min,x_max=self.domain

        lb = np.array([t_min, x_min])
        ub = np.array([t_max, x_max])
        p = lb + (ub - lb) * lhs(2, n)

        lb = np.array([t_min, x_min])
        ub = np.array([t_max, x_min])
        p_bc = lb + (ub - lb) * lhs(2, n_bc//2)

        lb = np.array([t_min, x_max])
        ub = np.array([t_max, x_max])
        temp = lb + (ub - lb) * lhs(2, n_bc//2)
        p_bc = np.vstack((p_bc, temp))

        lb = np.array([t_min, x_min])
        ub = np.array([t_min, x_max])
        p_ic = lb + (ub - lb) * lhs(2, n_ic)


        return p,p_bc,p_ic
```

```
class Testset_Burgers(object):
    """Dataset on a square domain"""
    def __init__(self,problem,*args):
```

```
        self.problem = problem
        self.domain = problem.domain
        self.args = args

    def __repr__(self):
        return f'{self.__doc__}'

    def __call__(self,verbose=None):
        n_t,n_x=self.args[0],self.args[1]
        p,t,x=self._uniform_sample(n_t,n_x)
        if verbose=='tensor':
            p=torch.from_numpy(p).float()
            t=torch.from_numpy(t).float()
            x=torch.from_numpy(x).float()
            return p,t,x
        return p,t,x

    def _uniform_sample(self,n_t,n_x):
        t_min,t_max,x_min,x_max=self.domain
        t = np.linspace(t_min, t_max, n_t)
        x = np.linspace(x_min, x_max, n_x)
        t,x = np.meshgrid(t,x)
        p=np.hstack((t.reshape(t.size,-1),x.reshape(x.size,-1)))
        return p,t,x
```

```
trainset = Trainset_Burgers(problem, 40, 40, method='uniform')
p,p_bc,p_ic,bv,iv=trainset(plot=True)
print(p.shape,p_bc.shape,p_ic.shape)
```

结果如下：




```
(1560,2) ; (80,2) ; (40,2)
```

2.3、网络结构

```
def grad(outputs, inputs):
    return torch.autograd.grad(outputs, inputs,
                               grad_outputs=torch.ones_like(outputs),
                               create_graph=True)

class PINN_Burgers(DNN):
```

```
    def __init__(self,dim_in,dim_out,dim_hidden, hidden_layers,
                 act_name='sigmoid',init_name='xavier_normal'):
        super().__init__(dim_in,dim_out,dim_hidden,hidden_layers,
                         act_name=act_name, init_name=init_name)
    def forward(self,problem,p,p_bc=None,p_ic=None):
        p.requires_grad_(True)

        u=super().forward(p)
        if p_bc is None:
            return u
        grad_u = grad(u, p)[0]
        u_t = grad_u[:, [0]]
        u_x = grad_u[:, [1]]
        u_xx = grad(u_x, p)[0][:, [1]]

        p.detach_()

        epilson=problem.epilson()
        f=u_t+u*u_x-epilson*u_xx

        bv_bar=super().forward(p_bc)
        iv_bar=super().forward(p_ic)
        return f,bv_bar,iv_bar
```

```
class ResPINN_Burgers(ResDNN):
    def __init__(self,dim_in,dim_out,dim_hidden, hidden_layers,
                 act_name='sigmoid',init_name='xavier_normal'):
        super().__init__(dim_in,dim_out,dim_hidden,res_blocks,
                         act_name=act_name, init_name=init_name)
    def forward(self,problem,p,p_bc=None,p_ic=None):
        p.requires_grad_(True)

        u=super().forward(p)
        if p_bc is None:
            return u
        grad_u = grad(u, p)[0]
        u_t = grad_u[:, [0]]
        u_x = grad_u[:, [1]]
        u_xx = grad(u_x, p)[0][:, [1]]

        p.detach_()
        epilson=problem.epilson()
        f=u_t+ u*u_x-epilson*u_xx

        bv_bar=super().forward(p_bc)
        iv_bar=super().forward(p_ic)
        return f,bv_bar,iv_bar
```

```
model = PINN_Burgers(2, 1, 10, 8)
print(model.model_size())
```

```
811
```

2.4、Options

```
class Options_Burgers(object):
    def __init__(self):
        parser = argparse.ArgumentParser()
        parser.add_argument('--no_cuda', action='store_true', default=False,
help='disable CUDA or not')
        parser.add_argument('--dim_hidden', type=int, default=10, help='neurons
in hidden layers')
        parser.add_argument('--hidden_layers', type=int, default=4, help='number
of hidden layers')
        parser.add_argument('--res_blocks', type=int, default=4, help='number of
residual blocks')
        parser.add_argument('--lam', type=float, default=1, help='weight in loss
function')
        parser.add_argument('--lr', type=float, default=1e-3, help='initial
learning rate')
        parser.add_argument('--epochs_Adam', type=int, default=1000,
help='epochs for Adam optimizer')
        parser.add_argument('--epochs_LBFGS', type=int, default=200,
help='epochs for LBFGS optimizer')
        parser.add_argument('--step_size', type=int, default=2000, help='step
size in lr_scheduler for Adam optimizer')
        parser.add_argument('--gamma', type=float, default=0.7, help='gamma in
lr_scheduler for Adam optimizer')
        parser.add_argument('--resume', type=bool, default=False, help='resume
or not')
        parser.add_argument('--sample_method', type=str, default='lhs',
help='sample method')
        parser.add_argument('--n_t', type=int, default=100, help='sample points
in x-direction for uniform sample')
        parser.add_argument('--n_x', type=int, default=100, help='sample points
in y-direction for uniform sample')
        parser.add_argument('--n', type=int, default=10000, help='sample points
in domain for lhs sample')
        parser.add_argument('--n_bc', type=int, default=400, help='sample points
on the boundary for lhs sample')
        parser.add_argument('--n_ic', type=int, default=400, help='sample points
on the initial for lhs sample')

        self.parser = parser
    def parse(self):
        arg = self.parser.parse_args(args=[])
        arg.cuda = not arg.no_cuda and torch.cuda.is_available()
        arg.device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')
        return arg
```

```
args=Options_Burgers().parse()
print(args)
```

```
Namespace(cuda=False, device=device(type='cpu'), dim_hidden=10,
epochs_Adam=1000, epochs_LBFGS=200, gamma=0.7, hidden_layers=4, lam=1, lr=0.001,
n=10000, n_bc=400, n_ic=400, n_t=100, n_x=100, no_cuda=False, res_blocks=4,
resume=False, sample_method='lhs', step_size=2000)
```

2.5、训练过程

```
def save_model(state,is_best=None,save_dir=None):
    last_model = os.path.join(save_dir, 'last_model.pth.tar')
    torch.save(state, last_model)
    if is_best:
        best_model = os.path.join(save_dir, 'best_model.pth.tar')
        shutil.copyfile(last_model, best_model)

class Trainer_Burgers(object):
    def __init__(self,args):
        self.device=args.device
        self.problem=args.problem

        self.lam=args.lam
        self.criterion=nn.MSELoss()

        self.model = args.model
        self.model_name = self.model.__class__.__name__
        self.model_path = self._model_path()

        self.epochs_Adam = args.epochs_Adam
        self.epochs_LBFGS = args.epochs_LBFGS
        self.optimizer_Adam = optim.Adam(self.model.parameters(), lr=args.lr)
        self.optimizer_LBFGS = optim.LBFGS(self.model.parameters(),
                                           max_iter=20,
                                           tolerance_grad=1.e-8,
                                           tolerance_change=1.e-12)
        self.lr_scheduler = StepLR(self.optimizer_Adam,
                                   step_size=args.step_size,
                                   gamma=args.gamma)

        self.model.to(self.device)
        self.model.zero_grad()

        self.p,     self.p_bc,    self.p_ic, self.bv,     self.iv     =
args.trainset(verbose='tensor')
        self.f=torch.from_numpy(np.zeros_like(self.p[:,0])).float()
        self.g=torch.cat((self.bv,self.iv))

    def _model_path(self):
        """Path to save the model"""
        if not os.path.exists('checkpoints'):
            os.mkdir('checkpoints')

        path = os.path.join('checkpoints', self.model_name)
        if not os.path.exists(path):
            os.mkdir(path)
```

```
        return path

    def train(self):
        best_loss=1.e10
        for epoch in range(self.epochs_Adam):
            loss, loss1, loss2 = self.train_Adam()
            if (epoch + 1) % 100 == 0:
                self.infos_Adam(epoch+1, loss, loss1, loss2)

                valid_loss = self.validate(epoch)
                is_best = valid_loss < best_loss
                best_loss = valid_loss if is_best else best_loss
                state = {
                    'epoch': epoch,
                    'state_dict': self.model.state_dict(),
                    'best_loss': best_loss
                }
                save_model(state, is_best, save_dir=self.model_path)

        for epoch in range(self.epochs_Adam, self.epochs_Adam +
self.epochs_LBFGS):
            loss, loss1, loss2 = self.train_LBFGS()
            if (epoch + 1) % 20 == 0:
                self.infos_LBFGS(epoch+1, loss, loss1, loss2)

                valid_loss = self.validate(epoch)
                is_best = valid_loss < best_loss
                best_loss = valid_loss if is_best else best_loss
                state = {
                    'epoch': epoch,
                    'state_dict': self.model.state_dict(),
                    'best_loss': best_loss
                }
                save_model(state, is_best, save_dir=self.model_path)
    def train_Adam(self):
        self.optimizer_Adam.zero_grad()


 f_pred,bv_pred,iv_pred=self.model(self.problem,self.p,self.p_bc,self.p_ic)
        g_pred=torch.cat((bv_pred,iv_pred))
        loss1=self.criterion(f_pred,self.f)
        loss2=self.criterion(g_pred,self.g)
        loss=loss1+self.lam*loss2

        loss.backward()
        self.optimizer_Adam.step()
        self.lr_scheduler.step()

        return loss.item(),loss1.item(),loss2.item()

    def infos_Adam(self,epoch,loss,loss1,loss2):
        infos = 'Adam  ' + \
            f'Epoch #{epoch:5d}/{self.epochs_Adam+self.epochs_LBFGS} ' + \
            f'Loss: {loss:.4e} = {loss1:.4e} + {self.lam} * {loss2:.4e} ' + \
            f'lr: {self.lr_scheduler.get_lr()[0]:.2e} '
        print(infos)

    def train_LBFGS(self):
```

```
 f_pred,bv_pred,iv_pred=self.model(self.problem,self.p,self.p_bc,self.p_ic)
        g_pred=torch.cat((bv_pred,iv_pred))
        loss1=self.criterion(f_pred,self.f)
        loss2=self.criterion(g_pred,self.g)

        def closure():
            if torch.is_grad_enabled():
                self.optimizer_LBFGS.zero_grad()

 f_pred,bv_pred,iv_pred=self.model(self.problem,self.p,self.p_bc,self.p_ic)
            g_pred=torch.cat((bv_pred,iv_pred))
            loss1=self.criterion(f_pred,self.f)
            loss2=self.criterion(g_pred,self.g)
            loss = loss1 + self.lam * loss2

            if loss.requires_grad:
                loss.backward()
            return loss
        self.optimizer_LBFGS.step(closure)
        loss=closure()

        return loss.item(), loss1.item(), loss2.item()

    def infos_LBFGS(self, epoch, loss, loss1, loss2):
        infos = 'LBFGS ' + \
            f'Epoch #{epoch:5d}/{self.epochs_Adam+self.epochs_LBFGS} ' + \
            f'Loss: {loss:.2e} = {loss1:.2e} + {self.lam:d} * {loss2:.2e} '
        print(infos)

    def validate(self, epoch):
        self.model.eval()

 f_pred,bv_pred,iv_pred=self.model(self.problem,self.p,self.p_bc,self.p_ic)
        g_pred=torch.cat((bv_pred,iv_pred))
        loss1=self.criterion(f_pred,self.f)
        loss2=self.criterion(g_pred,self.g)
        loss = loss1 + self.lam * loss2
        infos = 'Valid ' + \
            f'Epoch #{epoch+1:5d}/{self.epochs_Adam+self.epochs_LBFGS} ' + \
            f'Loss: {loss:.4e} '
        print(infos)
        self.model.train()
        return loss.item()
```

```
Problem=Problem_Burgers()
args=Options_Burgers().parse()
args.problem=Problem_Burgers()
args.model = PINN_Burgers(dim_in=2,
                  dim_out=1,
                  dim_hidden=args.dim_hidden,
                  hidden_layers=args.hidden_layers,
                  act_name='sigmoid')
if args.sample_method == 'uniform':
    args.trainset = Trainset_Burgers(args.problem, args.n_t, args.n_x,
method='uniform')
```

```
elif args.sample_method == 'lhs':
    args.trainset = Trainset_Burgers(args.problem, args.n, args.n_bc,args.n_ic,
method='lhs')

trainer_Burgers = Trainer_Burgers(args)
trainer_Burgers.train()
```

```
D:\anaconda\lib\site-packages\torch\optim\lr_scheduler.py:351: UserWarning: To
get the last learning rate computed by the scheduler, please use
`get_last_lr()`.
  "please use `get_last_lr()`.", UserWarning)
```

```
Adam  Epoch #  100/5500 Loss: 4.2497e-01 = 1.1670e-02 + 1 * 3.8173e-01 + 1 *
1.7214e-02 + 1 * 1.4353e-02 lr: 1.00e-03
Valid Epoch #  100/5500 Loss: 4.2169e-01
Adam  Epoch #  200/5500 Loss: 4.0839e-01 = 1.4056e-02 + 1 * 3.5858e-01 + 1 *
1.7730e-02 + 1 * 1.8029e-02 lr: 1.00e-03
Valid Epoch #  200/5500 Loss: 4.0623e-01
Adam  Epoch #  300/5500 Loss: 4.0290e-01 = 1.5770e-02 + 1 * 3.4714e-01 + 1 *
1.9950e-02 + 1 * 2.0036e-02 lr: 1.00e-03
Valid Epoch #  300/5500 Loss: 4.0158e-01
Adam  Epoch #  400/5500 Loss: 3.9960e-01 = 1.4884e-02 + 1 * 3.4251e-01 + 1 *
2.1068e-02 + 1 * 2.1145e-02 lr: 1.00e-03
Valid Epoch #  400/5500 Loss: 3.9872e-01
Adam  Epoch #  500/5500 Loss: 3.9701e-01 = 1.3930e-02 + 1 * 3.3941e-01 + 1 *
2.1874e-02 + 1 * 2.1792e-02 lr: 1.00e-03
Valid Epoch #  500/5500 Loss: 3.9646e-01
Adam  Epoch #  600/5500 Loss: 3.9472e-01 = 1.3031e-02 + 1 * 3.3697e-01 + 1 *
2.2465e-02 + 1 * 2.2252e-02 lr: 1.00e-03
Valid Epoch #  600/5500 Loss: 3.9448e-01
Adam  Epoch #  700/5500 Loss: 3.9243e-01 = 1.2156e-02 + 1 * 3.3464e-01 + 1 *
2.2937e-02 + 1 * 2.2695e-02 lr: 1.00e-03
Valid Epoch #  700/5500 Loss: 3.9248e-01
Adam  Epoch #  800/5500 Loss: 3.8964e-01 = 1.1322e-02 + 1 * 3.3171e-01 + 1 *
2.3382e-02 + 1 * 2.3222e-02 lr: 1.00e-03
Valid Epoch #  800/5500 Loss: 3.8993e-01
Adam  Epoch #  900/5500 Loss: 3.8593e-01 = 1.1387e-02 + 1 * 3.2708e-01 + 1 *
2.3749e-02 + 1 * 2.3714e-02 lr: 1.00e-03
Valid Epoch #  900/5500 Loss: 3.8639e-01
Adam  Epoch # 1000/5500 Loss: 3.8373e-01 = 1.3016e-02 + 1 * 3.2336e-01 + 1 *
2.3629e-02 + 1 * 2.3729e-02 lr: 1.00e-03
Valid Epoch # 1000/5500 Loss: 3.8433e-01
Adam  Epoch # 1100/5500 Loss: 3.8277e-01 = 1.3220e-02 + 1 * 3.2213e-01 + 1 *
2.3504e-02 + 1 * 2.3919e-02 lr: 1.00e-03
Valid Epoch # 1100/5500 Loss: 3.8358e-01
Adam  Epoch # 1200/5500 Loss: 3.8203e-01 = 1.3154e-02 + 1 * 3.2130e-01 + 1 *
2.3510e-02 + 1 * 2.4061e-02 lr: 1.00e-03
Valid Epoch # 1200/5500 Loss: 3.8299e-01
Adam  Epoch # 1300/5500 Loss: 3.8137e-01 = 1.3116e-02 + 1 * 3.2055e-01 + 1 *
2.3556e-02 + 1 * 2.4144e-02 lr: 1.00e-03
Valid Epoch # 1300/5500 Loss: 3.8246e-01
Adam  Epoch # 1400/5500 Loss: 3.8078e-01 = 1.3143e-02 + 1 * 3.1986e-01 + 1 *
2.3583e-02 + 1 * 2.4195e-02 lr: 1.00e-03
Valid Epoch # 1400/5500 Loss: 3.8199e-01
Adam  Epoch # 1500/5500 Loss: 3.8027e-01 = 1.3223e-02 + 1 * 3.1923e-01 + 1 *
2.3581e-02 + 1 * 2.4235e-02 lr: 1.00e-03
```

```
Valid Epoch # 1500/5500 Loss: 3.8157e-01
Adam  Epoch # 1600/5500 Loss: 3.7984e-01 = 1.3331e-02 + 1 * 3.1868e-01 + 1 *
2.3557e-02 + 1 * 2.4275e-02 lr: 1.00e-03
Valid Epoch # 1600/5500 Loss: 3.8122e-01
Adam  Epoch # 1700/5500 Loss: 3.7948e-01 = 1.3449e-02 + 1 * 3.1818e-01 + 1 *
2.3526e-02 + 1 * 2.4319e-02 lr: 1.00e-03
Valid Epoch # 1700/5500 Loss: 3.8093e-01
Adam  Epoch # 1800/5500 Loss: 3.7919e-01 = 1.3570e-02 + 1 * 3.1775e-01 + 1 *
2.3500e-02 + 1 * 2.4369e-02 lr: 1.00e-03
Valid Epoch # 1800/5500 Loss: 3.8068e-01
Adam  Epoch # 1900/5500 Loss: 3.7896e-01 = 1.3689e-02 + 1 * 3.1736e-01 + 1 *
2.3485e-02 + 1 * 2.4420e-02 lr: 1.00e-03
Valid Epoch # 1900/5500 Loss: 3.8049e-01
Adam  Epoch # 2000/5500 Loss: 3.7878e-01 = 1.3795e-02 + 1 * 3.1703e-01 + 1 *
2.3483e-02 + 1 * 2.4468e-02 lr: 4.90e-04
Valid Epoch # 2000/5500 Loss: 3.8034e-01
Adam  Epoch # 2100/5500 Loss: 3.7867e-01 = 1.3857e-02 + 1 * 3.1682e-01 + 1 *
2.3490e-02 + 1 * 2.4500e-02 lr: 7.00e-04
Valid Epoch # 2100/5500 Loss: 3.8024e-01
Adam  Epoch # 2200/5500 Loss: 3.7857e-01 = 1.3909e-02 + 1 * 3.1662e-01 + 1 *
2.3503e-02 + 1 * 2.4530e-02 lr: 7.00e-04
Valid Epoch # 2200/5500 Loss: 3.8016e-01
Adam  Epoch # 2300/5500 Loss: 3.7847e-01 = 1.3949e-02 + 1 * 3.1644e-01 + 1 *
2.3522e-02 + 1 * 2.4560e-02 lr: 7.00e-04
Valid Epoch # 2300/5500 Loss: 3.8007e-01
Adam  Epoch # 2400/5500 Loss: 3.7837e-01 = 1.3980e-02 + 1 * 3.1626e-01 + 1 *
2.3545e-02 + 1 * 2.4589e-02 lr: 7.00e-04
Valid Epoch # 2400/5500 Loss: 3.7998e-01
Adam  Epoch # 2500/5500 Loss: 3.7827e-01 = 1.4004e-02 + 1 * 3.1608e-01 + 1 *
2.3573e-02 + 1 * 2.4621e-02 lr: 7.00e-04
Valid Epoch # 2500/5500 Loss: 3.7989e-01
Adam  Epoch # 2600/5500 Loss: 3.7817e-01 = 1.4022e-02 + 1 * 3.1589e-01 + 1 *
2.3603e-02 + 1 * 2.4654e-02 lr: 7.00e-04
Valid Epoch # 2600/5500 Loss: 3.7980e-01
Adam  Epoch # 2700/5500 Loss: 3.7806e-01 = 1.4035e-02 + 1 * 3.1570e-01 + 1 *
2.3637e-02 + 1 * 2.4692e-02 lr: 7.00e-04
Valid Epoch # 2700/5500 Loss: 3.7970e-01
Adam  Epoch # 2800/5500 Loss: 3.7795e-01 = 1.4045e-02 + 1 * 3.1550e-01 + 1 *
2.3673e-02 + 1 * 2.4735e-02 lr: 7.00e-04
Valid Epoch # 2800/5500 Loss: 3.7959e-01
Adam  Epoch # 2900/5500 Loss: 3.7783e-01 = 1.4050e-02 + 1 * 3.1528e-01 + 1 *
2.3714e-02 + 1 * 2.4786e-02 lr: 7.00e-04
Valid Epoch # 2900/5500 Loss: 3.7947e-01
Adam  Epoch # 3000/5500 Loss: 3.7769e-01 = 1.4050e-02 + 1 * 3.1504e-01 + 1 *
2.3759e-02 + 1 * 2.4846e-02 lr: 7.00e-04
Valid Epoch # 3000/5500 Loss: 3.7934e-01
Adam  Epoch # 3100/5500 Loss: 3.7755e-01 = 1.4045e-02 + 1 * 3.1478e-01 + 1 *
2.3810e-02 + 1 * 2.4917e-02 lr: 7.00e-04
Valid Epoch # 3100/5500 Loss: 3.7919e-01
Adam  Epoch # 3200/5500 Loss: 3.7739e-01 = 1.4034e-02 + 1 * 3.1449e-01 + 1 *
2.3869e-02 + 1 * 2.5000e-02 lr: 7.00e-04
Valid Epoch # 3200/5500 Loss: 3.7903e-01
Adam  Epoch # 3300/5500 Loss: 3.7723e-01 = 1.4016e-02 + 1 * 3.1418e-01 + 1 *
2.3939e-02 + 1 * 2.5095e-02 lr: 7.00e-04
Valid Epoch # 3300/5500 Loss: 3.7886e-01
Adam  Epoch # 3400/5500 Loss: 3.7705e-01 = 1.3992e-02 + 1 * 3.1384e-01 + 1 *
2.4022e-02 + 1 * 2.5202e-02 lr: 7.00e-04
Valid Epoch # 3400/5500 Loss: 3.7867e-01
```

```
Adam  Epoch # 3500/5500 Loss: 3.7687e-01 = 1.3963e-02 + 1 * 3.1347e-01 + 1 *
2.4118e-02 + 1 * 2.5321e-02 lr: 7.00e-04
Valid Epoch # 3500/5500 Loss: 3.7847e-01
Adam  Epoch # 3600/5500 Loss: 3.7669e-01 = 1.3936e-02 + 1 * 3.1308e-01 + 1 *
2.4226e-02 + 1 * 2.5448e-02 lr: 7.00e-04
Valid Epoch # 3600/5500 Loss: 3.7826e-01
Adam  Epoch # 3700/5500 Loss: 3.7651e-01 = 1.3921e-02 + 1 * 3.1267e-01 + 1 *
2.4337e-02 + 1 * 2.5581e-02 lr: 7.00e-04
Valid Epoch # 3700/5500 Loss: 3.7806e-01
Adam  Epoch # 3800/5500 Loss: 3.7633e-01 = 1.3927e-02 + 1 * 3.1224e-01 + 1 *
2.4445e-02 + 1 * 2.5720e-02 lr: 7.00e-04
Valid Epoch # 3800/5500 Loss: 3.7785e-01
Adam  Epoch # 3900/5500 Loss: 3.7615e-01 = 1.3949e-02 + 1 * 3.1179e-01 + 1 *
2.4543e-02 + 1 * 2.5870e-02 lr: 7.00e-04
Valid Epoch # 3900/5500 Loss: 3.7764e-01
Adam  Epoch # 4000/5500 Loss: 3.7595e-01 = 1.3977e-02 + 1 * 3.1131e-01 + 1 *
2.4630e-02 + 1 * 2.6037e-02 lr: 3.43e-04
Valid Epoch # 4000/5500 Loss: 3.7742e-01
Adam  Epoch # 4100/5500 Loss: 3.7581e-01 = 1.3995e-02 + 1 * 3.1096e-01 + 1 *
2.4688e-02 + 1 * 2.6166e-02 lr: 4.90e-04
Valid Epoch # 4100/5500 Loss: 3.7724e-01
Adam  Epoch # 4200/5500 Loss: 3.7565e-01 = 1.4011e-02 + 1 * 3.1058e-01 + 1 *
2.4744e-02 + 1 * 2.6308e-02 lr: 4.90e-04
Valid Epoch # 4200/5500 Loss: 3.7704e-01
Adam  Epoch # 4300/5500 Loss: 3.7546e-01 = 1.4023e-02 + 1 * 3.1017e-01 + 1 *
2.4800e-02 + 1 * 2.6469e-02 lr: 4.90e-04
Valid Epoch # 4300/5500 Loss: 3.7681e-01
Adam  Epoch # 4400/5500 Loss: 3.7523e-01 = 1.4031e-02 + 1 * 3.0969e-01 + 1 *
2.4858e-02 + 1 * 2.6656e-02 lr: 4.90e-04
Valid Epoch # 4400/5500 Loss: 3.7652e-01
Adam  Epoch # 4500/5500 Loss: 3.7495e-01 = 1.4032e-02 + 1 * 3.0911e-01 + 1 *
2.4919e-02 + 1 * 2.6888e-02 lr: 4.90e-04
Valid Epoch # 4500/5500 Loss: 3.7616e-01
Adam  Epoch # 4600/5500 Loss: 3.7459e-01 = 1.4021e-02 + 1 * 3.0841e-01 + 1 *
2.4979e-02 + 1 * 2.7188e-02 lr: 4.90e-04
Valid Epoch # 4600/5500 Loss: 3.7572e-01
Adam  Epoch # 4700/5500 Loss: 3.7414e-01 = 1.4298e-02 + 1 * 3.0688e-01 + 1 *
2.5178e-02 + 1 * 2.7791e-02 lr: 4.90e-04
Valid Epoch # 4700/5500 Loss: 3.7516e-01
Adam  Epoch # 4800/5500 Loss: 3.7360e-01 = 1.3938e-02 + 1 * 3.0654e-01 + 1 *
2.5083e-02 + 1 * 2.8042e-02 lr: 4.90e-04
Valid Epoch # 4800/5500 Loss: 3.7454e-01
Adam  Epoch # 4900/5500 Loss: 3.7301e-01 = 1.3974e-02 + 1 * 3.0524e-01 + 1 *
2.5131e-02 + 1 * 2.8664e-02 lr: 4.90e-04
Valid Epoch # 4900/5500 Loss: 3.7385e-01
Adam  Epoch # 5000/5500 Loss: 3.7243e-01 = 1.3891e-02 + 1 * 3.0415e-01 + 1 *
2.5181e-02 + 1 * 2.9209e-02 lr: 4.90e-04
Valid Epoch # 5000/5500 Loss: 3.7318e-01
```

```
LBFGS Epoch # 5020/5500 Loss: 3.60e-01 = 1.30e-02 + 1 * 2.78e-01 + 1 * 3.44e-02
+ 1 * 3.47e-02
Valid Epoch # 5020/5500 Loss: 3.5900e-01
LBFGS Epoch # 5040/5500 Loss: 3.60e-01 = 1.31e-02 + 1 * 2.78e-01 + 1 * 3.43e-02
+ 1 * 3.46e-02
Valid Epoch # 5040/5500 Loss: 3.5940e-01
LBFGS Epoch # 5060/5500 Loss: 3.60e-01 = 1.31e-02 + 1 * 2.78e-01 + 1 * 3.42e-02
+ 1 * 3.46e-02
```

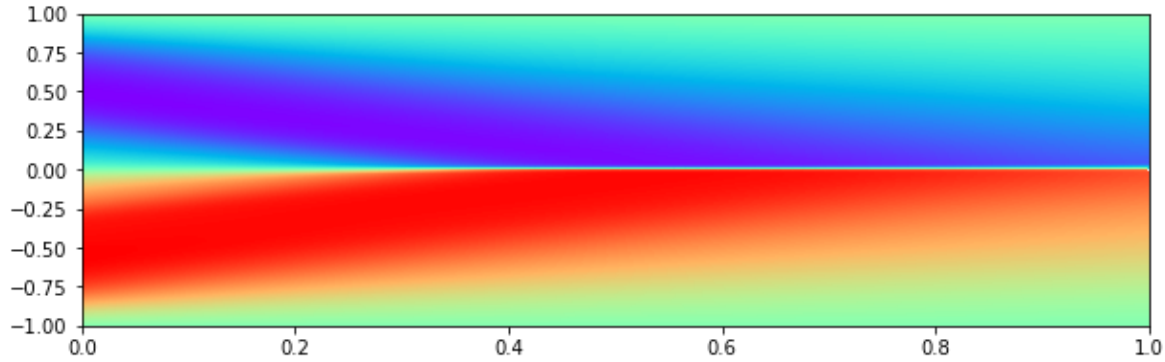
```
Valid Epoch # 5060/5500 Loss: 3.6070e-01
LBFGS Epoch # 5080/5500 Loss: 3.60e-01 = 1.35e-02 + 1 * 2.78e-01 + 1 * 3.40e-02
+ 1 * 3.46e-02
Valid Epoch # 5080/5500 Loss: 3.7879e-01
LBFGS Epoch # 5100/5500 Loss: 3.60e-01 = 1.38e-02 + 1 * 2.77e-01 + 1 * 3.40e-02
+ 1 * 3.45e-02
Valid Epoch # 5100/5500 Loss: 3.9900e-01
LBFGS Epoch # 5120/5500 Loss: 3.58e-01 = 1.69e-02 + 1 * 2.75e-01 + 1 * 3.19e-02
+ 1 * 3.40e-02
Valid Epoch # 5120/5500 Loss: 1.0423e+00
LBFGS Epoch # 5140/5500 Loss: 3.08e-01 = 1.83e-02 + 1 * 2.45e-01 + 1 * 1.44e-02
+ 1 * 3.13e-02
Valid Epoch # 5140/5500 Loss: 4.5673e+02
LBFGS Epoch # 5160/5500 Loss: 2.95e-01 = 1.87e-02 + 1 * 2.28e-01 + 1 * 1.51e-02
+ 1 * 3.39e-02
Valid Epoch # 5160/5500 Loss: 1.8993e+02
LBFGS Epoch # 5180/5500 Loss: 2.82e-01 = 1.79e-02 + 1 * 2.26e-01 + 1 * 1.16e-02
+ 1 * 2.70e-02
Valid Epoch # 5180/5500 Loss: 1.0855e+02
LBFGS Epoch # 5200/5500 Loss: 2.66e-01 = 1.92e-02 + 1 * 2.19e-01 + 1 * 5.23e-03
+ 1 * 2.31e-02
Valid Epoch # 5200/5500 Loss: 3.5884e+02
LBFGS Epoch # 5220/5500 Loss: 2.57e-01 = 1.73e-02 + 1 * 2.07e-01 + 1 * 6.70e-03
+ 1 * 2.65e-02
Valid Epoch # 5220/5500 Loss: 7.9761e+02
LBFGS Epoch # 5240/5500 Loss: 2.52e-01 = 1.54e-02 + 1 * 1.99e-01 + 1 * 9.05e-03
+ 1 * 2.93e-02
Valid Epoch # 5240/5500 Loss: 8.1986e+02
LBFGS Epoch # 5260/5500 Loss: 2.49e-01 = 1.41e-02 + 1 * 1.93e-01 + 1 * 1.05e-02
+ 1 * 3.11e-02
Valid Epoch # 5260/5500 Loss: 6.1486e+02
LBFGS Epoch # 5280/5500 Loss: nan = nan + 1 * nan + 1 * nan + 1 * nan
Valid Epoch # 5280/5500 Loss: nan
LBFGS Epoch # 5300/5500 Loss: nan = nan + 1 * nan + 1 * nan + 1 * nan
Valid Epoch # 5300/5500 Loss: nan
LBFGS Epoch # 5320/5500 Loss: nan = nan + 1 * nan + 1 * nan + 1 * nan
Valid Epoch # 5320/5500 Loss: nan
LBFGS Epoch # 5340/5500 Loss: nan = nan + 1 * nan + 1 * nan + 1 * nan
Valid Epoch # 5340/5500 Loss: nan
LBFGS Epoch # 5360/5500 Loss: nan = nan + 1 * nan + 1 * nan + 1 * nan
Valid Epoch # 5360/5500 Loss: nan
LBFGS Epoch # 5380/5500 Loss: nan = nan + 1 * nan + 1 * nan + 1 * nan
Valid Epoch # 5380/5500 Loss: nan
LBFGS Epoch # 5400/5500 Loss: nan = nan + 1 * nan + 1 * nan + 1 * nan
Valid Epoch # 5400/5500 Loss: nan
LBFGS Epoch # 5420/5500 Loss: nan = nan + 1 * nan + 1 * nan + 1 * nan
Valid Epoch # 5420/5500 Loss: nan
LBFGS Epoch # 5440/5500 Loss: nan = nan + 1 * nan + 1 * nan + 1 * nan
Valid Epoch # 5440/5500 Loss: nan
LBFGS Epoch # 5460/5500 Loss: nan = nan + 1 * nan + 1 * nan + 1 * nan
Valid Epoch # 5460/5500 Loss: nan
LBFGS Epoch # 5480/5500 Loss: nan = nan + 1 * nan + 1 * nan + 1 * nan
Valid Epoch # 5480/5500 Loss: nan
LBFGS Epoch # 5500/5500 Loss: nan = nan + 1 * nan + 1 * nan + 1 * nan
Valid Epoch # 5500/5500 Loss: nan
```

2.6、测试过程

```
class Tester_Burgers(object):
    def __init__(self,args):
        self.device  = args.device
        self.problem = args.problem
        self.criterion = nn.MSELoss()
        self.model = args.model
        model_name = self.model.__class__.__name__
        model_path = os.path.join('checkpoints',
                                  model_name,
                                  'best_model.pth.tar')
        best_model = torch.load(model_path)
        self.model.load_state_dict(best_model['state_dict'])
        self.model.to(self.device)

        self.p,self.t,self.x=args.testset(verbose='tensor')
    def predict(self):
        self.model.eval()
        u_pred=self.model(self.problem,self.p)
        u_pred=u_pred.detach().cpu().numpy()
        u_pred=u_pred.reshape(self.t.shape)

        plt.figure(figsize=(10,3),frameon=False)
        plt.contourf(self.t,self.x,u_pred,levels=1000,cmap='rainbow')
        plt.show
```

```
# 使用Tester_Burgers进行预测
args=Options_Burgers().parse()
args.problem=Problem_Burgers()

args.model = PINN_Burgers(dim_in=2,
                  dim_out=1,
                  dim_hidden=args.dim_hidden,
                  hidden_layers=args.hidden_layers,
                  act_name='sigmoid')
args.testset = Testset_Burgers(args.problem, 100, 100)
tester = Tester_Burgers(args)
tester.predict()
```

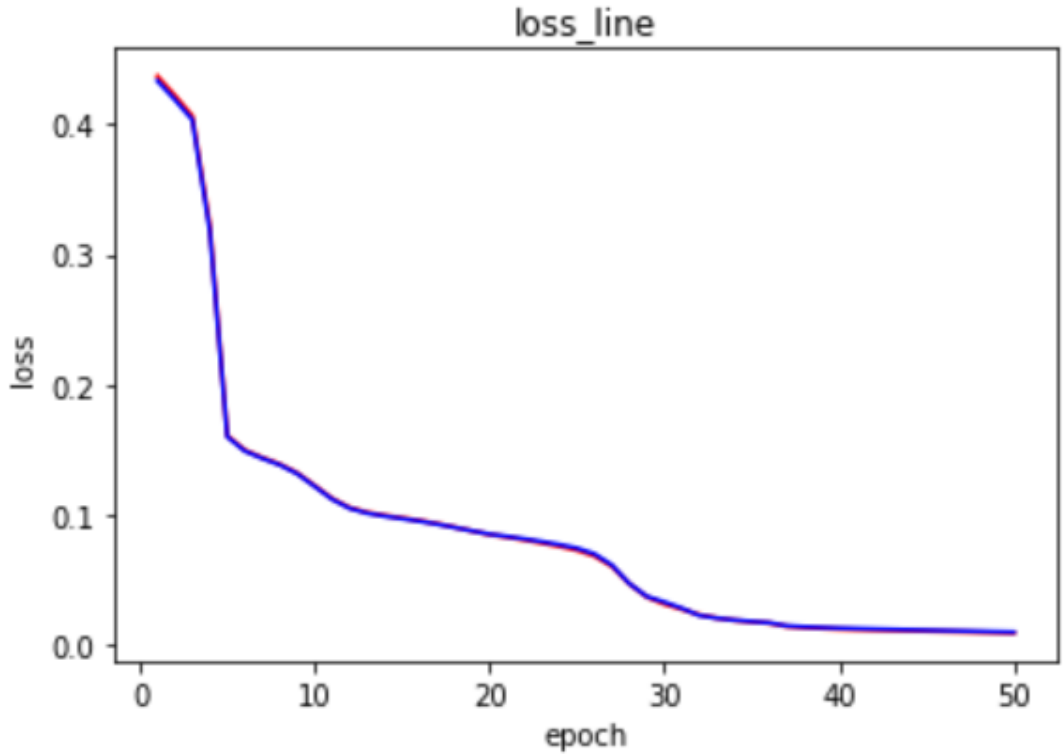
预测结果如下：



2.7结果分析

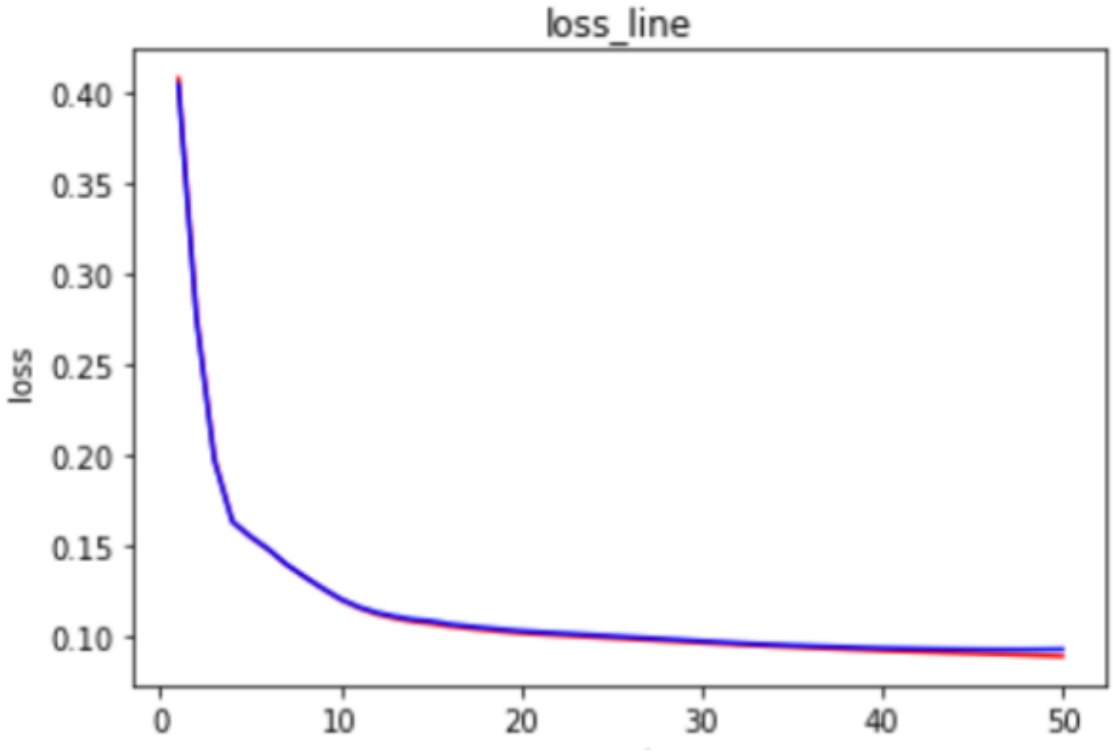
在模型训练的过程中，按照要求，分别采用PINN、ResPINN网络进行训练，得到不同的Loss曲线，记录损失函数曲线可以得到如下曲线型:

1.PINN网络Loss曲线：



分析以上曲线可以看出，曲线有两个主要的收敛点，当epoch=500时，曲线迅速收敛至loss接近0.1的水平，当epoch=2500时曲线进一步收敛至接近1e-3水平，最终收敛状况较好。

2.ResPINN训练曲线:



可以观察到，与PINN曲线相比，ResPINN曲线迅速收敛到极低的loss值，收敛性能有明显改善。

3、Allen-Cahn方程

考虑带周期边界条件的Allen-Cahn方程：

$$\begin{cases} u_t - 0.0001u_{xx} + 5u^3 - 5u = 0, & x \in [-1,1],\ t \in [0,1] \\ u(0,x) = x^2\cos(\pi x), \\ u(t,-1) = u(t,1), \\ u_x(t,-1) = u_x(t,1). \end{cases}$$

3.1、问题描述

```
class Problem(object):
    """Description of Allen-Cahn equation"""
    def __init__(self, domain=(-1, 1, 0, 1)):
        self.domain = domain

    def __repr__(self):
        return f'{self.__doc__}'

    def f(self, x, verbose=None):
        out = np.zeros_like(x[:, [0]])
        return out

    def g_t0(self, x, verbose=None):
        out = x[:, [0]]*x[:, [0]]*np.cos(np.pi*x[:, [0]])
        return out
    def g_xl(self, x, verbose=None):
        out = np.zeros_like(x[:, [0]])
        return out
    def g_xr(self, x, verbose=None):
        out = np.zeros_like(x[:, [0]])
        return out
    def a(self, x, order=0, verbose=None):
        if order == 0:
            out = np.ones_like(x[:, [0]])
        elif order == 1:
            a_x = np.zeros_like(x[:, [0]])
            a_y = a_x
            out = np.hstack((a_x, a_y))

        if verbose == 'tensor':
            return torch.from_numpy(out).float()
        return out
```

3.2、数据集生成

```
class Trainset(object):
    def __init__(self, problem, *args, **kwargs):
        self.problem = problem
        self.domain = problem.domain
        self.args = args
        self.method = kwargs['method']
```

```
    def __call__(self, plot=False, verbose=None):
        if self.method == 'uniform':
            n_x, n_y = self.args[0], self.args[1]
            x, x_t0 = self._uniform_sample_t0(n_x, n_y)
            x, x_xl = self._uniform_sample_xl(n_x, n_y)
            x_bc = np.hstack((x_t0, x_xl))
            x, x_xr = self._uniform_sample_xr(n_x, n_y)
            x_bc = np.hstack((x_bc, x_xr))
        elif self.method == 'lhs':
            n, n_bc = self.args[0], self.args[1]
            x, x_bc = self._lhs_sample(n, n_bc)
        f = self.problem.f(x)
        #u = self.problem.u(x)
        g_t0 = self.problem.g_t0(x_bc[:,[0,1]])
        g_xl = self.problem.g_xl(x_bc[:,[2,3]])
        g_xr = self.problem.g_xr(x_bc[:,[4,5]])

        if plot:
            fig = plt.figure()
            ax = fig.add_subplot(111)
            ax.scatter(x[:, 0], x[:, 1], facecolor='r', s=10)
            ax.scatter(x_bc[:, 0], x_bc[:, 1], facecolor='b', s=10)
            ax.scatter(x_bc[:, 2], x_bc[:, 3], facecolor='b', s=10)
            ax.scatter(x_bc[:, 4], x_bc[:, 5], facecolor='b', s=10)
            ax.set_xlim(-1.01, 1.01)
            ax.set_ylim(-0.01, 1.01)
            ax.set_aspect('equal')
            plt.show()
        if verbose is not None:
            x = torch.from_numpy(x).float()
            x_bc = torch.from_numpy(x_bc).float()
            #u = torch.from_numpy(u).float()
            f = torch.from_numpy(f).float()
            g_t0 = torch.from_numpy(g_t0).float()
            g_xl = torch.from_numpy(g_xl).float()
            g_xr = torch.from_numpy(g_xr).float()
            #return x, x_bc, u, f, g
            return x, x_bc, f, g_t0, g_xl, g_xr
        #return x, x_bc, u, f, g
        return x, x_bc, f, g_t0, g_xl, g_xr

    def _uniform_sample_t0(self, n_x, n_y):
        x_min, x_max, y_min, y_max = self.domain
        x = np.linspace(x_min, x_max, n_x)
        y = np.linspace(y_min, y_max, n_y)
        x, y = np.meshgrid(x, y)
        xy = np.hstack((x.reshape(x.size, -1), y.reshape(y.size, -1)))

        #mask = (xy[:, 0] - x_min) * (x_max - xy[:, 0]) * (xy[:, 1] - y_min) *
(y_max - xy[:, 1]) == 0
        mask = (xy[:, 1] - y_min) == 0
        x_bc = xy[mask]
        x = xy[np.logical_not(mask)]

        return x, x_bc   #x_bc 0_1 is t0  2_3 is xl  4_5 is xr

    def _uniform_sample_xl(self, n_x, n_y):
        x_min, x_max, y_min, y_max = self.domain
```

```
        x = np.linspace(x_min, x_max, n_x)
        y = np.linspace(y_min, y_max, n_y)
        x, y = np.meshgrid(x, y)
        xy = np.hstack((x.reshape(x.size, -1), y.reshape(y.size, -1)))

        #mask = (xy[:, 0] - x_min) * (x_max - xy[:, 0]) * (xy[:, 1] - y_min) *
(y_max - xy[:, 1]) == 0
        mask = (xy[:, 0] - x_min) == 0
        x_bc = xy[mask]
        x = xy[np.logical_not(mask)]

        return x, x_bc

    def _uniform_sample_xr(self, n_x, n_y):
        x_min, x_max, y_min, y_max = self.domain
        x = np.linspace(x_min, x_max, n_x)
        y = np.linspace(y_min, y_max, n_y)
        x, y = np.meshgrid(x, y)
        xy = np.hstack((x.reshape(x.size, -1), y.reshape(y.size, -1)))

        #mask = (xy[:, 0] - x_min) * (x_max - xy[:, 0]) * (xy[:, 1] - y_min) *
(y_max - xy[:, 1]) == 0
        mask = (x_max - xy[:, 0]) == 0
        x_bc = xy[mask]
        x = xy[np.logical_not(mask)]

        return x, x_bc

    def _lhs_sample(self, n, n_bc):
        x_min, x_max, y_min, y_max = self.domain

        lb = np.array([x_min, y_min])
        ub = np.array([x_max, y_max])
        x = lb + (ub - lb) * lhs(2, n)

        lb = np.array([x_min, y_min])
        ub = np.array([x_max, y_min])
        x_bc = lb + (ub - lb) * lhs(2, n_bc)

        #lb = np.array([x_min, y_max])
        #ub = np.array([x_max, y_max])
        #temp = lb + (ub - lb) * lhs(2, n_bc)
        #x_bc = np.vstack((x_bc, temp))

        lb = np.array([x_min, y_min])
        ub = np.array([x_min, y_max])
        temp = lb + (ub - lb) * lhs(2, n_bc)
        x_bc = np.hstack((x_bc, temp))

        lb = np.array([x_max, y_min])
        ub = np.array([x_max, y_max])
        temp = lb + (ub - lb) * lhs(2, n_bc)
        x_bc = np.hstack((x_bc, temp))

        return x, x_bc
```

```
class Testset(object):
```

```
    """Dataset on a square domain"""
    def __init__(self, problem, *args, **kwargs):
        self.problem = problem
        self.domain = problem.domain
        self.args = args
        self.method = kwargs['method']

    def __repr__(self):
        return f'{self.__doc__}'

    def __call__(self, plot=False, verbose=None):
        if self.method == 'uniform':
            n_x, n_y = self.args[0], self.args[1]
            X, x, y = self._uniform_sample(n_x, n_y)
        if plot == True:
            fig = plt.figure()
            ax = fig.add_subplot(111)
            ax.scatter(X[:, 0], X[:, 1], facecolor='r', s=10)
            ax.set_xlim(-1.01, 1.01)
            ax.set_ylim(-0.01, 1.01)
            ax.set_aspect('equal')


        if verbose == 'tensor':
            X = torch.from_numpy(X).float()

        #return X, x, y, u
        return X, x, y

    def _uniform_sample(self, n_x, n_y):
        x_min, x_max, y_min, y_max = self.domain
        x = np.linspace(x_min, x_max, n_x)
        y = np.linspace(y_min, y_max, n_y)
        x, y = np.meshgrid(x, y)
        X = np.hstack((x.reshape(x.size, -1), y.reshape(y.size, -1)))
        return X, x, y
```

3.3、网络结构

```
def activation(name):
    if name in ['tanh', 'Tanh']:
        return nn.Tanh()
    elif name in ['relu', 'ReLU']:
        return nn.ReLU(inplace=True)
    elif name in ['leaky_relu', 'LeakyReLU']:
        return nn.LeakyReLU(inplace=True)
    elif name in ['sigmoid', 'Sigmoid']:
        return nn.Sigmoid()
    elif name in ['softplus', 'Softplus']:
        return nn.Softplus()
    else:
        raise ValueError(f'unknown activation function: {name}')
def grad(outputs, inputs):
    return torch.autograd.grad(outputs, inputs,
                               grad_outputs=torch.ones_like(outputs),
                               create_graph=True,allow_unused=True)
#,allow_unused=True
```

```
class PINN(DNN):
    """Physics Constrained Neural Networks
    """
    def __init__(self, dim_in, dim_out, dim_hidden, hidden_layers,
act_name='tanh', init_name='xavier_normal'):
        super().__init__(dim_in, dim_out, dim_hidden, hidden_layers,
act_name=act_name, init_name=init_name)

    def forward(self, problem, x, x_bc=None):
        x.requires_grad_(True)

        u = super().forward(x)

        grad_u = grad(u, x)[0]
        u_x = grad_u[:, [0]]
        u_y = grad_u[:, [1]]
        u_xx = grad(u_x, x)[0][:, [0]]
        u_yy = grad(u_y, x)[0][:, [1]]

        x.detach_()
        x_ = x.cpu().numpy()
        a = problem.a(x_, verbose='tensor')
        grad_a = problem.a(x_, order=1, verbose='tensor')
        a_x = grad_a[:, [0]]
        a_y = grad_a[:, [1]]

        f = a*u_y + 0.5*u_xx + abs(u)*abs(u)*u

        if x_bc is not None:
            xgl = x_bc[:,[2,3]]
            xgr = x_bc[:,[4,5]]

            xgl.requires_grad_(True)
            upl = super().forward(xgl)
            grad_upl = grad(upl, xgl)[0]
            upl_x = grad_upl[:, [0]]

            xgr.requires_grad_(True)
            upr = super().forward(xgr)
            grad_upr = grad(upr, xgr)[0]
            upr_x = grad_upr[:, [0]]


            g_t0 = super().forward(x_bc[:,[0,1]])
            g_xl = upl - upr
            g_xr = upl_x - upr_x

            return u, f, g_t0, g_xl, g_xr
        return u, f
```

```
class ResPINN(DNN):
    """Physics Constrained Neural Networks
    """
    def __init__(self, dim_in, dim_out, dim_hidden, res_blocks, act_name='tanh',
init_name='xavier_normal'):
```

```
        super().__init__(dim_in, dim_out, dim_hidden, res_blocks,
act_name=act_name, init_name=init_name)

    def forward(self, problem, x, x_bc=None):
        x.requires_grad_(True)

        u = super().forward(x)

        grad_u = grad(u, x)[0]
        u_x = grad_u[:, [0]]
        u_y = grad_u[:, [1]]
        u_xx = grad(u_x, x)[0][:, [0]]
        u_yy = grad(u_y, x)[0][:, [1]]

        x.detach_()
        x_ = x.cpu().numpy()
        a = problem.a(x_, verbose='tensor')
        grad_a = problem.a(x_, order=1, verbose='tensor')
        a_x = grad_a[:, [0]]
        a_y = grad_a[:, [1]]

        f = a*u_y + 0.0001*u_xx + 5*u*u*u - 5*u

        if x_bc is not None:
            xgl = x_bc[:,[2,3]]
            xgr = x_bc[:,[4,5]]

            xgl.requires_grad_(True)
            upl = super().forward(xgl)
            grad_upl = grad(upl, xgl)[0]
            upl_x = grad_upl[:, [0]]

            xgr.requires_grad_(True)
            upr = super().forward(xgr)
            grad_upr = grad(upr, xgr)[0]
            upr_x = grad_upr[:, [0]]


            g_t0 = super().forward(x_bc[:,[0,1]])
            g_xl = upl - upr
            g_xr = upl_x - upr_x

            return u, f, g_t0, g_xl, g_xr
        return u, f
```

3.4、Options

```
import argparse
class Options(object):
    def __init__(self):
        parser = argparse.ArgumentParser()
        parser.add_argument('--no_cuda', action='store_true', default=False,
help='disable CUDA or not')
        parser.add_argument('--dim_hidden', type=int, default=10, help='neurons
in hidden layers')
        parser.add_argument('--hidden_layers', type=int, default=4, help='number
of hidden layers')
```

```
        parser.add_argument('--res_blocks', type=int, default=4, help='number of
residual blocks')
        parser.add_argument('--dropout', type=float, default=0.5, help='dropout
rate')
        parser.add_argument('--lam_t0', type=float, default=1, help='weight in
loss function')#xishu_canshu
        parser.add_argument('--lam_xl', type=float, default=1, help='weight in
loss function')
        parser.add_argument('--lam_xr', type=float, default=1, help='weight in
loss function')
        parser.add_argument('--lr', type=float, default=1e-3, help='initial
learning rate')
        parser.add_argument('--epochs_Adam', type=int, default=5000,
help='epochs for Adam optimizer')
        parser.add_argument('--epochs_LBFGS', type=int, default=500,
help='epochs for LBFGS optimizer')
        parser.add_argument('--step_size', type=int, default=2000, help='step
size in lr_scheduler for Adam optimizer')
        parser.add_argument('--gamma', type=float, default=0.7, help='gamma in
lr_scheduler for Adam optimizer')
        parser.add_argument('--resume', type=bool, default=False, help='resume
or not')
        parser.add_argument('--sample_method', type=str, default='lhs',
help='sample method')
        parser.add_argument('--n_x', type=int, default=100, help='sample points
in x-direction for uniform sample')
        parser.add_argument('--n_y', type=int, default=100, help='sample points
in y-direction for uniform sample')
        parser.add_argument('--n', type=int, default=10000, help='sample points
in domain for lhs sample')
        parser.add_argument('--n_bc', type=int, default=400, help='sample points
on the boundary for lhs sample')
        parser.add_argument('--case', type=int, default=1, help='problem case')

        self.parser = parser
    def parse(self):
        arg = self.parser.parse_args(args=[])
        arg.cuda = not arg.no_cuda and torch.cuda.is_available()
        arg.device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')
        return arg
def save_model(state, is_best=None, save_dir=None):
    last_model = os.path.join(save_dir, 'last_model.pth.tar')
    torch.save(state, last_model)
    if is_best:
        best_model = os.path.join(save_dir, 'best_model.pth.tar')
        shutil.copyfile(last_model, best_model)
```

3.5、训练过程

```
class Trainer(object):
    def __init__(self, args):
        self.device  = args.device
        self.problem = args.problem

        self.lam_t0 = args.lam_t0
        self.lam_xl = args.lam_xl
```

```
        self.lam_xr = args.lam_xr
        self.criterion = nn.MSELoss()

        self.model = args.model
        self.model_name = self.model.__class__.__name__
        self.model_path = self._model_path()

        self.epochs_Adam = args.epochs_Adam
        self.epochs_LBFGS = args.epochs_LBFGS
        self.optimizer_Adam = optim.Adam(self.model.parameters(), lr=args.lr)
        self.optimizer_LBFGS = optim.LBFGS(self.model.parameters(), max_iter=20,
tolerance_grad=1.e-8, tolerance_change=1.e-12)
        self.lr_scheduler = StepLR(self.optimizer_Adam,
step_size=args.step_size, gamma=args.gamma)

        self.model.to(self.device)
        self.model.zero_grad()

        #self.x,     self.x_bc,     _, self.f,     self.g     =
args.trainset(verbose='tensor')
        self.x,     self.x_bc,      self.f,     self.g_t0,     self.g_xl,
self.g_xr= args.trainset(verbose='tensor')
        #self.x_val, self.x_bc_val, _, self.f_val, self.g_val =
args.validset(verbose='tensor')
        self.x_val, self.x_bc_val,  self.f_val, self.g_t0_val, self.g_xl_val,
self.g_xr_val = args.validset(verbose='tensor')

        if self.device == torch.device(type='cuda'):
            for item in [self.x, self.x_bc, self.f, self.g_t0_val,
self.g_xl_val, self.g_xr_val, self.x_val, self.x_bc_val, self.f_val,
self.g_val]:
                item = item.to(self.device)

    def _model_path(self):
        """Path to save the model"""
        if not os.path.exists('checkpoints'):
            os.mkdir('checkpoints')

        path = os.path.join('checkpoints', self.model_name)
        if not os.path.exists(path):
            os.mkdir(path)

        return path

    def train(self,plot=False):
        best_loss = 1.e10
        key_loss = []
        validate_loss = []

        for epoch in range(self.epochs_Adam):
            loss, loss1, loss_t0, loss_xl, loss_xr = self.train_Adam()

            if (epoch + 1) % 100 == 0:
                self.infos_Adam(epoch+1, loss, loss1, loss_t0, loss_xl, loss_xr)

                valid_loss = self.validate(epoch)

                key_loss.append(loss)
```

```
                validate_loss.append(valid_loss)

                is_best = valid_loss < best_loss
                best_loss = valid_loss if is_best else best_loss
                state = {
                    'epoch': epoch,
                    'state_dict': self.model.state_dict(),
                    'best_loss': best_loss
                }
                save_model(state, is_best, save_dir=self.model_path)

        if plot == True:
            number = range(1,len(key_loss)+1)
            fig = plt.figure()
            ax = fig.add_subplot(111)
            ax.plot(number,key_loss,color = 'red',label = 'loss')
            ax.plot(number,validate_loss,color = 'blue',label = 'valid_loss')
            plt.xlabel("epoch")
            plt.ylabel("loss")
            plt.title('loss_line')
            plt.show()


        for epoch in range(self.epochs_Adam, self.epochs_Adam +
self.epochs_LBFGS):
            loss, loss1, loss_t0, loss_xl, loss_xr = self.train_LBFGS()
            if (epoch + 1) % 20 == 0:
                self.infos_LBFGS(epoch+1, loss, loss1, loss_t0, loss_xl,
loss_xr)

                valid_loss = self.validate(epoch)
                is_best = valid_loss < best_loss
                best_loss = valid_loss if is_best else best_loss
                state = {
                    'epoch': epoch,
                    'state_dict': self.model.state_dict(),
                    'best_loss': best_loss
                }
                save_model(state, is_best, save_dir=self.model_path)

    def train_Adam(self):
        self.optimizer_Adam.zero_grad()

        _, f_pred, g_pred_t0, g_pred_xl, g_pred_xr = self.model(self.problem,
self.x, self.x_bc)
        loss1 = self.criterion(f_pred, self.f)
        loss_t0 = self.criterion(g_pred_t0, self.g_t0)
        loss_xl = self.criterion(g_pred_xl, self.g_xl)
        loss_xr = self.criterion(g_pred_xr, self.g_xr)
        loss = loss1 + self.lam_t0 * loss_t0 + self.lam_xl * loss_xl +
self.lam_xr * loss_xr

        loss.backward()
        self.optimizer_Adam.step()
        self.lr_scheduler.step()

        return loss.item(), loss1.item(), loss_t0.item(), loss_xl.item(),
loss_xr.item()
```

```
#pick
    def infos_Adam(self, epoch, loss, loss1, loss_t0, loss_xl, loss_xr):
        infos = 'Adam  ' + \
            f'Epoch #{epoch:5d}/{self.epochs_Adam+self.epochs_LBFGS} ' + \
            f'Loss: {loss:.4e} = {loss1:.4e} + {self.lam_t0} * {loss_t0:.4e} +
{self.lam_xl} * {loss_xl:.4e} + {self.lam_xr} * {loss_xr:.4e} ' + \
            f'lr: {self.lr_scheduler.get_lr()[0]:.2e} '
        print(infos)

    def train_LBFGS(self):

        # only used to compute loss_int and loss_bc1 for monitoring
        _, f_pred, g_t0_pred, g_xl_pred, g_xr_pred = self.model(self.problem,
self.x, self.x_bc)
        loss1 = self.criterion(f_pred, self.f)
        loss_t0 = self.criterion(g_t0_pred, self.g_t0)
        loss_xl = self.criterion(g_xl_pred, self.g_xl)
        loss_xr = self.criterion(g_xr_pred, self.g_xr)

        def closure():
            if torch.is_grad_enabled():
                self.optimizer_LBFGS.zero_grad()
            _, f_pred, g_t0_pred, g_xl_pred, g_xr_pred =
self.model(self.problem, self.x, self.x_bc)
            loss1 = self.criterion(f_pred, self.f)
            loss_t0 = self.criterion(g_t0_pred, self.g_t0)
            loss_xl = self.criterion(g_xl_pred, self.g_xl)
            loss_xr = self.criterion(g_xr_pred, self.g_xr)
            loss = loss1 + self.lam_t0 * loss_t0 + self.lam_xl * loss_xl +
self.lam_xr * loss_xr
            if loss.requires_grad:
                loss.backward()
            return loss


        self.optimizer_LBFGS.step(closure)
        loss = closure()

        return loss.item(), loss1.item(), loss_t0.item(), loss_xl.item(),
loss_xr.item()

    def infos_LBFGS(self, epoch, loss, loss1, loss_t0, loss_xl, loss_xr):
        infos = 'LBFGS ' + \
            f'Epoch #{epoch:5d}/{self.epochs_Adam+self.epochs_LBFGS} ' + \
            f'Loss: {loss:.2e} = {loss1:.2e} + {self.lam_t0:d} * {loss_t0:.2e} +
{self.lam_xl:d} * {loss_xl:.2e} + {self.lam_xr:d} * {loss_xr:.2e} '
        print(infos)

    def validate(self, epoch):
        self.model.eval()
        _, f_pred, g_t0_pred, g_xl_pred, g_xr_pred = self.model(self.problem,
self.x_val, self.x_bc_val)
        loss1 = self.criterion(f_pred, self.f_val)
        loss_t0 = self.criterion(g_t0_pred, self.g_t0_val)
        loss_xl = self.criterion(g_xl_pred, self.g_xl_val)
        loss_xr = self.criterion(g_xr_pred, self.g_xr_val)
        loss = loss1 + self.lam_t0 * loss_t0 + self.lam_xl * loss_xl +
self.lam_xr * loss_xr
```

```
        infos = 'Valid ' + \
            f'Epoch #{epoch+1:5d}/{self.epochs_Adam+self.epochs_LBFGS} ' + \
            f'Loss: {loss:.4e} '
        print(infos)
        self.model.train()

        return loss.item()
```

```
args = Options().parse()
args.problem = Problem()


args.model = ResPINN(2, 1, dim_hidden=200, res_blocks=4)
if args.sample_method == 'uniform':
    args.trainset = Trainset(args.problem, args.n_x, args.n_y, method='uniform')
elif args.sample_method == 'lhs':
    args.trainset = Trainset(args.problem, args.n, args.n_bc, method='lhs')

args.validset = Trainset(args.problem, 100, 100, method='uniform')

trainer = Trainer(args)
trainer.train(plot=True)
```

```
Adam  Epoch #  100/5500 Loss: 1.4312e-01 = 9.1755e-03 + 1 * 1.3232e-01 + 1 *
1.2763e-04 + 1 * 1.4956e-03 lr: 1.00e-03
Valid Epoch #  100/5500 Loss: 1.5099e-01
Adam  Epoch #  200/5500 Loss: 1.3989e-01 = 2.4613e-03 + 1 * 1.3593e-01 + 1 *
4.2258e-05 + 1 * 1.4606e-03 lr: 1.00e-03
Valid Epoch #  200/5500 Loss: 1.4826e-01
Adam  Epoch #  300/5500 Loss: 1.3980e-01 = 2.6218e-03 + 1 * 1.3577e-01 + 1 *
5.2822e-05 + 1 * 1.3532e-03 lr: 1.00e-03
Valid Epoch #  300/5500 Loss: 1.4815e-01
Adam  Epoch #  400/5500 Loss: 1.3973e-01 = 2.7226e-03 + 1 * 1.3571e-01 + 1 *
6.1018e-05 + 1 * 1.2387e-03 lr: 1.00e-03
Valid Epoch #  400/5500 Loss: 1.4807e-01
Adam  Epoch #  500/5500 Loss: 1.3967e-01 = 2.7802e-03 + 1 * 1.3570e-01 + 1 *
6.6321e-05 + 1 * 1.1246e-03 lr: 1.00e-03
Valid Epoch #  500/5500 Loss: 1.4801e-01
Adam  Epoch #  600/5500 Loss: 1.3962e-01 = 2.8047e-03 + 1 * 1.3573e-01 + 1 *
6.9304e-05 + 1 * 1.0162e-03 lr: 1.00e-03
Valid Epoch #  600/5500 Loss: 1.4796e-01
Adam  Epoch #  700/5500 Loss: 1.3958e-01 = 2.8096e-03 + 1 * 1.3578e-01 + 1 *
7.0830e-05 + 1 * 9.1728e-04 lr: 1.00e-03
Valid Epoch #  700/5500 Loss: 1.4792e-01
Adam  Epoch #  800/5500 Loss: 1.3955e-01 = 2.8051e-03 + 1 * 1.3584e-01 + 1 *
7.1585e-05 + 1 * 8.2948e-04 lr: 1.00e-03
Valid Epoch #  800/5500 Loss: 1.4789e-01
Adam  Epoch #  900/5500 Loss: 1.3952e-01 = 2.7973e-03 + 1 * 1.3590e-01 + 1 *
7.1988e-05 + 1 * 7.5344e-04 lr: 1.00e-03
Valid Epoch #  900/5500 Loss: 1.4787e-01
Adam  Epoch # 1000/5500 Loss: 1.3950e-01 = 2.7894e-03 + 1 * 1.3595e-01 + 1 *
7.2246e-05 + 1 * 6.8894e-04 lr: 1.00e-03
Valid Epoch # 1000/5500 Loss: 1.4785e-01
Adam  Epoch # 1100/5500 Loss: 1.3948e-01 = 2.7824e-03 + 1 * 1.3599e-01 + 1 *
7.2445e-05 + 1 * 6.3520e-04 lr: 1.00e-03
```

```
Valid Epoch # 1100/5500 Loss: 1.4784e-01
Adam  Epoch # 1200/5500 Loss: 1.3947e-01 = 2.7765e-03 + 1 * 1.3603e-01 + 1 *
7.2616e-05 + 1 * 5.9114e-04 lr: 1.00e-03
Valid Epoch # 1200/5500 Loss: 1.4783e-01
Adam  Epoch # 1300/5500 Loss: 1.3946e-01 = 2.7718e-03 + 1 * 1.3606e-01 + 1 *
7.2773e-05 + 1 * 5.5558e-04 lr: 1.00e-03
Valid Epoch # 1300/5500 Loss: 1.4782e-01
Adam  Epoch # 1400/5500 Loss: 1.3945e-01 = 2.7681e-03 + 1 * 1.3608e-01 + 1 *
7.2920e-05 + 1 * 5.2731e-04 lr: 1.00e-03
Valid Epoch # 1400/5500 Loss: 1.4781e-01
Adam  Epoch # 1500/5500 Loss: 1.3944e-01 = 2.7653e-03 + 1 * 1.3610e-01 + 1 *
7.3059e-05 + 1 * 5.0514e-04 lr: 1.00e-03
Valid Epoch # 1500/5500 Loss: 1.4781e-01
Adam  Epoch # 1600/5500 Loss: 1.3943e-01 = 2.7633e-03 + 1 * 1.3611e-01 + 1 *
7.3191e-05 + 1 * 4.8801e-04 lr: 1.00e-03
Valid Epoch # 1600/5500 Loss: 1.4780e-01
Adam  Epoch # 1700/5500 Loss: 1.3942e-01 = 2.7618e-03 + 1 * 1.3611e-01 + 1 *
7.3320e-05 + 1 * 4.7494e-04 lr: 1.00e-03
Valid Epoch # 1700/5500 Loss: 1.4779e-01
Adam  Epoch # 1800/5500 Loss: 1.3941e-01 = 2.7608e-03 + 1 * 1.3611e-01 + 1 *
7.3446e-05 + 1 * 4.6510e-04 lr: 1.00e-03
Valid Epoch # 1800/5500 Loss: 1.4778e-01
Adam  Epoch # 1900/5500 Loss: 1.3940e-01 = 2.7602e-03 + 1 * 1.3611e-01 + 1 *
7.3573e-05 + 1 * 4.5778e-04 lr: 1.00e-03
Valid Epoch # 1900/5500 Loss: 1.4777e-01
Adam  Epoch # 2000/5500 Loss: 1.3939e-01 = 2.7598e-03 + 1 * 1.3611e-01 + 1 *
7.3700e-05 + 1 * 4.5236e-04 lr: 4.90e-04
Valid Epoch # 2000/5500 Loss: 1.4776e-01
Adam  Epoch # 2100/5500 Loss: 1.3939e-01 = 2.7597e-03 + 1 * 1.3610e-01 + 1 *
7.3789e-05 + 1 * 4.4943e-04 lr: 7.00e-04
Valid Epoch # 2100/5500 Loss: 1.4776e-01
Adam  Epoch # 2200/5500 Loss: 1.3938e-01 = 2.7597e-03 + 1 * 1.3610e-01 + 1 *
7.3884e-05 + 1 * 4.4704e-04 lr: 7.00e-04
Valid Epoch # 2200/5500 Loss: 1.4775e-01
Adam  Epoch # 2300/5500 Loss: 1.3937e-01 = 2.7598e-03 + 1 * 1.3609e-01 + 1 *
7.3983e-05 + 1 * 4.4505e-04 lr: 7.00e-04
Valid Epoch # 2300/5500 Loss: 1.4774e-01
Adam  Epoch # 2400/5500 Loss: 1.3937e-01 = 2.7600e-03 + 1 * 1.3609e-01 + 1 *
7.4083e-05 + 1 * 4.4340e-04 lr: 7.00e-04
Valid Epoch # 2400/5500 Loss: 1.4773e-01
Adam  Epoch # 2500/5500 Loss: 1.3936e-01 = 2.7602e-03 + 1 * 1.3608e-01 + 1 *
7.4189e-05 + 1 * 4.4200e-04 lr: 7.00e-04
Valid Epoch # 2500/5500 Loss: 1.4772e-01
Adam  Epoch # 2600/5500 Loss: 1.3935e-01 = 2.7605e-03 + 1 * 1.3607e-01 + 1 *
7.4299e-05 + 1 * 4.4081e-04 lr: 7.00e-04
Valid Epoch # 2600/5500 Loss: 1.4771e-01
Adam  Epoch # 2700/5500 Loss: 1.3934e-01 = 2.7609e-03 + 1 * 1.3606e-01 + 1 *
7.4413e-05 + 1 * 4.3976e-04 lr: 7.00e-04
Valid Epoch # 2700/5500 Loss: 1.4770e-01
Adam  Epoch # 2800/5500 Loss: 1.3933e-01 = 2.7612e-03 + 1 * 1.3606e-01 + 1 *
7.4530e-05 + 1 * 4.3881e-04 lr: 7.00e-04
Valid Epoch # 2800/5500 Loss: 1.4769e-01
Adam  Epoch # 2900/5500 Loss: 1.3932e-01 = 2.7617e-03 + 1 * 1.3605e-01 + 1 *
7.4654e-05 + 1 * 4.3794e-04 lr: 7.00e-04
Valid Epoch # 2900/5500 Loss: 1.4768e-01
Adam  Epoch # 3000/5500 Loss: 1.3931e-01 = 2.7621e-03 + 1 * 1.3604e-01 + 1 *
7.4783e-05 + 1 * 4.3710e-04 lr: 7.00e-04
Valid Epoch # 3000/5500 Loss: 1.4767e-01
```

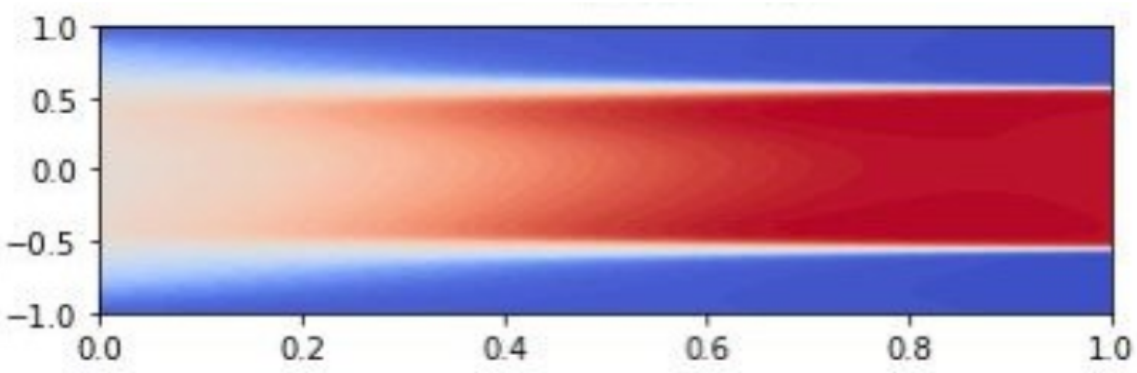
```
Adam  Epoch # 3100/5500 Loss: 1.3930e-01 = 2.7626e-03 + 1 * 1.3603e-01 + 1 *
7.4917e-05 + 1 * 4.3630e-04 lr: 7.00e-04
Valid Epoch # 3100/5500 Loss: 1.4766e-01
Adam  Epoch # 3200/5500 Loss: 1.3929e-01 = 2.7631e-03 + 1 * 1.3601e-01 + 1 *
7.5055e-05 + 1 * 4.3550e-04 lr: 7.00e-04
Valid Epoch # 3200/5500 Loss: 1.4765e-01
Adam  Epoch # 3300/5500 Loss: 1.3928e-01 = 2.7637e-03 + 1 * 1.3600e-01 + 1 *
7.5201e-05 + 1 * 4.3471e-04 lr: 7.00e-04
Valid Epoch # 3300/5500 Loss: 1.4764e-01
Adam  Epoch # 3400/5500 Loss: 1.3926e-01 = 2.7644e-03 + 1 * 1.3599e-01 + 1 *
7.5355e-05 + 1 * 4.3392e-04 lr: 7.00e-04
Valid Epoch # 3400/5500 Loss: 1.4762e-01
Adam  Epoch # 3500/5500 Loss: 1.3925e-01 = 2.7650e-03 + 1 * 1.3598e-01 + 1 *
7.5513e-05 + 1 * 4.3312e-04 lr: 7.00e-04
Valid Epoch # 3500/5500 Loss: 1.4761e-01
Adam  Epoch # 3600/5500 Loss: 1.3924e-01 = 2.7658e-03 + 1 * 1.3596e-01 + 1 *
7.5680e-05 + 1 * 4.3232e-04 lr: 7.00e-04
Valid Epoch # 3600/5500 Loss: 1.4759e-01
Adam  Epoch # 3700/5500 Loss: 1.3922e-01 = 2.7665e-03 + 1 * 1.3595e-01 + 1 *
7.5855e-05 + 1 * 4.3151e-04 lr: 7.00e-04
Valid Epoch # 3700/5500 Loss: 1.4758e-01
Adam  Epoch # 3800/5500 Loss: 1.3921e-01 = 2.7674e-03 + 1 * 1.3593e-01 + 1 *
7.6038e-05 + 1 * 4.3070e-04 lr: 7.00e-04
Valid Epoch # 3800/5500 Loss: 1.4756e-01
Adam  Epoch # 3900/5500 Loss: 1.3919e-01 = 2.7683e-03 + 1 * 1.3592e-01 + 1 *
7.6231e-05 + 1 * 4.2988e-04 lr: 7.00e-04
Valid Epoch # 3900/5500 Loss: 1.4755e-01
Adam  Epoch # 4000/5500 Loss: 1.3918e-01 = 2.7693e-03 + 1 * 1.3590e-01 + 1 *
7.6434e-05 + 1 * 4.2906e-04 lr: 3.43e-04
Valid Epoch # 4000/5500 Loss: 1.4753e-01
Adam  Epoch # 4100/5500 Loss: 1.3916e-01 = 2.7700e-03 + 1 * 1.3589e-01 + 1 *
7.6583e-05 + 1 * 4.2847e-04 lr: 4.90e-04
Valid Epoch # 4100/5500 Loss: 1.4752e-01
Adam  Epoch # 4200/5500 Loss: 1.3915e-01 = 2.7708e-03 + 1 * 1.3588e-01 + 1 *
7.6742e-05 + 1 * 4.2789e-04 lr: 4.90e-04
Valid Epoch # 4200/5500 Loss: 1.4750e-01
Adam  Epoch # 4300/5500 Loss: 1.3914e-01 = 2.7716e-03 + 1 * 1.3586e-01 + 1 *
7.6909e-05 + 1 * 4.2730e-04 lr: 4.90e-04
Valid Epoch # 4300/5500 Loss: 1.4749e-01
Adam  Epoch # 4400/5500 Loss: 1.3912e-01 = 2.7726e-03 + 1 * 1.3585e-01 + 1 *
7.7087e-05 + 1 * 4.2671e-04 lr: 4.90e-04
Valid Epoch # 4400/5500 Loss: 1.4747e-01
Adam  Epoch # 4500/5500 Loss: 1.3911e-01 = 2.7735e-03 + 1 * 1.3583e-01 + 1 *
7.7276e-05 + 1 * 4.2612e-04 lr: 4.90e-04
Valid Epoch # 4500/5500 Loss: 1.4746e-01
Adam  Epoch # 4600/5500 Loss: 1.3909e-01 = 2.7746e-03 + 1 * 1.3582e-01 + 1 *
7.7479e-05 + 1 * 4.2553e-04 lr: 4.90e-04
Valid Epoch # 4600/5500 Loss: 1.4744e-01
Adam  Epoch # 4700/5500 Loss: 1.3908e-01 = 2.7757e-03 + 1 * 1.3580e-01 + 1 *
7.7694e-05 + 1 * 4.2494e-04 lr: 4.90e-04
Valid Epoch # 4700/5500 Loss: 1.4743e-01
Adam  Epoch # 4800/5500 Loss: 1.3906e-01 = 2.7769e-03 + 1 * 1.3578e-01 + 1 *
7.7925e-05 + 1 * 4.2436e-04 lr: 4.90e-04
Valid Epoch # 4800/5500 Loss: 1.4741e-01
Adam  Epoch # 4900/5500 Loss: 1.3904e-01 = 2.7783e-03 + 1 * 1.3576e-01 + 1 *
7.8174e-05 + 1 * 4.2380e-04 lr: 4.90e-04
Valid Epoch # 4900/5500 Loss: 1.4739e-01
```

```
Adam  Epoch # 5000/5500 Loss: 1.3902e-01 = 2.7797e-03 + 1 * 1.3574e-01 + 1 *
7.8444e-05 + 1 * 4.2325e-04 lr: 4.90e-04
Valid Epoch # 5000/5500 Loss: 1.4737e-01
```

```
class Tester(object):
    def __init__(self, args):
        self.device  = args.device
        self.problem = args.problem
        self.criterion = nn.MSELoss()
        self.model = args.model
        model_name = self.model.__class__.__name__
        model_path = os.path.join('checkpoints',
                                  model_name,
                                  'best_model.pth.tar')
        best_model = torch.load(model_path)
        self.model.load_state_dict(best_model['state_dict'])
        self.model.to(self.device)
        #self.X, self.x, self.y, self.u = args.testset(verbose='tensor')
        self.X, self.x, self.y = args.testset(verbose='tensor')
        if self.device == torch.device(type='cuda'):
            self.X = self.X.to(self.device)

    def predict(self):
        self.model.eval()
        u_pred, _ = self.model(self.problem, self.X)
        u_pred = u_pred.detach().cpu().numpy()
        u_pred = u_pred.reshape(self.x.shape)

        fig = plt.figure()
        plt.contourf(self.y,self.x,u_pred,levels=1000,cmap='rainbow')
        plt.show()
```

```
# 使用Tester_AC进行预测
args = Options().parse()
args.problem = Problem()

args.model = ResPINN(2, 1, dim_hidden=args.dim_hidden,
res_blocks=args.res_blocks)
args.testset = Testset(args.problem, 100, 100, method='uniform')
tester = Tester(args)
tester.predict()
```

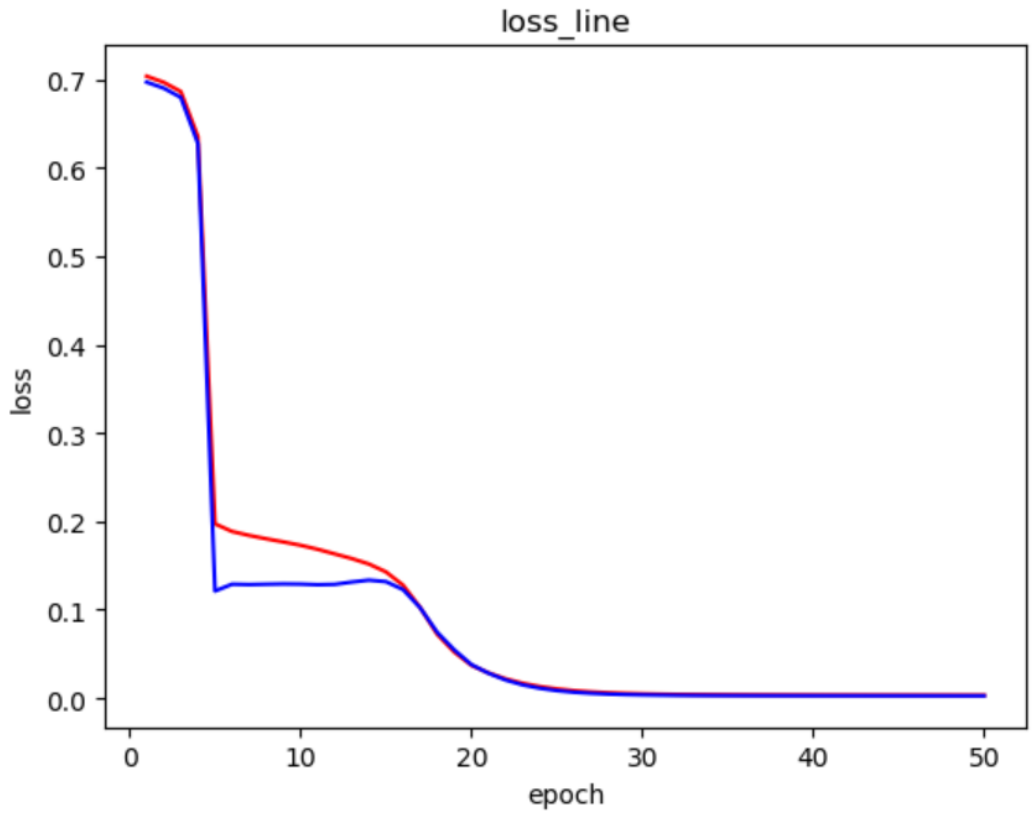
预测结果如下：



3.7、结果分析

采用同样的默认超参数与网络参数值，分别训练ResPINN与PINN网络，采集稳定点的Loss值评估训练效果，分析可知ResPINN在训练本问题时，收敛速度与收敛效果均优于PINN。

因此在模型训练的过程中，按照要求，采用ResPINN网络进行训练，得到ResPINN网络Loss曲线如下（PINN网络Loss曲线及其性质可参考2.7）：



可以观察到在训练初始阶段效果很好，在到达0.2损失值附近时，训练网络出现了不稳定现象，但是随之epoch的增加，不稳定现象逐渐消失，因此使用本模型解决该问题时，应注意epoch的设置值。