使用 ANN 解一阶常微分方程

2018302120169　傅宇千

3. 用人工神经网络的方法求解由下列微分方程导出的优化问题

$$(1)\ \begin{cases} \dfrac{\mathrm{d}y}{\mathrm{d}x} = x^3 - \dfrac{y}{x}, \\ y(1) = \dfrac{2}{5}. \end{cases}$$

该问题的精确解为

$$y(x) = \frac{x^4}{5} + \frac{1}{5x}.$$

$x_i\,(i=1,\cdots,m)$ 可取为区间 $[1,2]$ 上均匀等分的点.

取 3 个或以上隐藏神经元比较 BFGS 和 DFP 两种搜索算法的迭代次数和计算结果。

在这里我对 p 参数的初始值并没有使用书中的（1,1,...,1），而是使用 0~1 的随机数，这样经过实验明显取得了更优的效果

BFGS

代码:

```
function [p,val,k]=bfgs(fun,gfun,m,n,varargin)
%功能: 用BFGS算法求解无约束问题: min f(p)
%输入: p0是初始点, fun, gfun分别是目标函数及其梯度;
% varargin是输入的可变参数变量, 简单调用bfgs时可以忽略它,
% 但若其它程序循环调用该程序时将发挥重要的作用
%m为训练集数目, n为神经元数目
%输出: p, val分别是近似最优点和最优值, k是迭代次数.
maxk=10000; %给出最大迭代次数
x=linspace(1,2,m);%构建训练集
p0=rand(n,3);%构建初始值（以rand为初始值）
rho=0.5;sigma=0.5; epsilon=1e-1;
k=0;
Bk=ones(n,n,3);
Bk(:,:,1)=eye(n);%Bk=feval('Hess',x0);
Bk(:,:,2)=eye(n);
Bk(:,:,3)=eye(n);
while(k<maxk)
    gk=feval(gfun,x,p0,n,m,varargin{:}); %计算梯度
    if(norm(gk)<epsilon), break; end %检验终止准则
    %gk(:,1)=gk(:,1)/norm(gk(:,1));
    %gk(:,2)=gk(:,2)/norm(gk(:,2));
    %gk(:,3)=gk(:,3)/norm(gk(:,3));
    a=-Bk(:,:,1)\gk;
    b=-Bk(:,:,2)\gk;
    c=-Bk(:,:,3)\gk;
    dk(:,1)=a(:,1); %解方程组, 计算搜索方向(注意是左乘!)
    dk(:,2)=b(:,2);
    dk(:,3)=c(:,3);
    %dk=dk./norm(dk(1,:));
    m0=0; mk=0;
    while(m0<20) % 用Armijo搜索求步长
        newf=feval(fun,x,p0+rho^m0*dk,n,m,varargin{:});
        oldf=feval(fun,x,p0,n,m,varargin{:});
        newg=feval(gfun,x,p0+rho^m0*dk,n,m);
        if(newf<oldf+sigma*rho^m0*gk'*dk)
            mk=m0; break;
        end
        m0=m0+1;
    end
    %BFGS校正
    p=p0+rho^mk*dk;
    sk=p-p0; yk=feval(gfun,x,p,n,m,varargin{:})-gk;
    %yk(:,1)=yk(:,1)/norm(yk(:,1));
    %yk(:,2)=yk(:,2)/norm(yk(:,2));
    %yk(:,3)=yk(:,3)/norm(yk(:,3));
    if(yk'*sk>0)
        Bk(:,:,1)=Bk(:,:,1)-(Bk(:,:,1)*sk(:,1)*sk(:,1)'*Bk(:,:,1))/(sk(:,1)'*Bk(:,:,1)*sk(:,1))+
(yk(:,1)*yk(:,1)')/(yk(:,1)'*sk(:,1));
        Bk(:,:,2)=Bk(:,:,2)-(Bk(:,:,2)*sk(:,2)*sk(:,2)'*Bk(:,:,2))/(sk(:,2)'*Bk(:,:,2)*sk(:,2))+
(yk(:,2)*yk(:,2)')/(yk(:,2)'*sk(:,2));
        Bk(:,:,3)=Bk(:,:,3)-(Bk(:,:,3)*sk(:,3)*sk(:,3)'*Bk(:,:,3))/(sk(:,3)'*Bk(:,:,3)*sk(:,3))+
(yk(:,3)*yk(:,3)')/(yk(:,3)'*sk(:,3));
    end
    k=k+1; p0=p;
end
val=feval(fun,x,p0,n,m,varargin{:});
```

BFGS 主程序

```
function f=fun(x,p,n,m)
f=0;
for i=1:m
    s1=0;
    s2=0;
    for j=1:n
        x0=-p(j,1).*x(i)+p(j,3);
        s0=sigmoid(x0);
        s1=s1+p(j,2).*s0;
        s2=s2+s0^2.*p(j,1).*p(j,2).*exp(x0);
    end
    f0=(2-1/x(i)).*s1+(x(i)-1).*s2-x(i)^3+(2/5)/x(i);
    f=f+(f0).^2;
end
end
```

待优化函数（两个程序都调用）

```
function f=gfun(x,p,n,m)
f=0;

%syms p1;
%syms p2;
%syms p3;
%syms x0;
%f1(p1,p2,p3,x0)=p2*sigmoid(-p1*x0+p3);
%f11(p1,p2,p3,x0)=diff(f1,p1);
%f12(p1,p2,p3,x0)=diff(f1,p2);
%f13(p1,p2,p3,x0)=diff(f1,p3);
%f2(p1,p2,p3,x0)=exp(-p1.*x0+p3).*p1.*p2.*(sigmoid(-p1*x0+p3)^2);
%f21(p1,p2,p3,x0)=diff(f1,p1);
%f22(p1,p2,p3,x0)=diff(f1,p2);
%f23(p1,p2,p3,x0)=diff(f1,p3);
%f11(p1,p2,p3,x0)=(p2*x0*exp(p3 - p1*x0))/(exp(p3 - p1*x0) + 1)^2;
%f12(p1, p2, p3, x0) =1/(exp(p3 - p1*x0) + 1);
%f13(p1, p2, p3, x0) =-(p2*exp(p3 - p1*x0))/(exp(p3 - p1*x0) + 1)^2;
%f21(p1, p2, p3, x0) =(p2*exp(p3 - p1*x0))/(exp(p3 - p1*x0) + 1)^2 - (p1*p2*x0*exp(p3 - p1*x0))/(exp(p3
- p1*x0) + 1)^2 + (2*p1*p2*x0*exp(2*p3 - 2*p1*x0))/(exp(p3 - p1*x0) + 1)^3
%f22(p1, p2, p3, x0) = (p1*exp(p3 - p1*x0))/(exp(p3 - p1*x0) + 1)^2
%f23(p1, p2, p3, x0) =(p1*p2*exp(p3 - p1*x0))/(exp(p3 - p1*x0) + 1)^2 - (2*p1*p2*exp(2*p3 -
2*p1*x0))/(exp(p3 - p1*x0) + 1)^3
%需要时可用 subs eval 求值（但比较慢）
for i=1:m
    s1=0;
    s2=0;
    v1=zeros(n,3);
    v2=zeros(n,3);
    for j=1:n
        x1=-p(j,1).*x(i)+p(j,3);
        s0=sigmoid(x1);
        s1=s1+p(j,2).*s0;
        s2=s2+s0^2.*p(j,1).*p(j,2).*exp(x1);
        v1(j,1)=(p(j,2)*x(i)*exp(p(j,3) - p(j,1)*x(i)))/(exp(p(j,3) - p(j,1)*x(i)) + 1)^2;
        v1(j,2)=1/(exp(p(j,3) - p(j,1)*x(i)) + 1);
        v1(j,3)=-(p(j,2)*exp(p(j,3) - p(j,1)*x(i)))/(exp(p(j,3) - p(j,1)*x(i)) + 1)^2;
        %v1=v1+[f14,f15,f16]';
        v2(j,1)=(p(j,2)*exp(p(j,3) - p(j,1)*x(i)))/(exp(p(j,3) - p(j,1)*x(i)) + 1)^2 -
(p(j,1)*p(j,2)*x(i)*exp(p(j,3) - p(j,1)*x(i)))/(exp(p(j,3) - p(j,1)*x(i)) + 1)^2 +
(2*p(j,1)*p(j,2)*x(i)*exp(2*p(j,3) - 2*p(j,1)*x(i)))/(exp(p(j,3) - p(j,1)*x(i)) + 1)^3;
        v2(j,2)=(p(j,1)*exp(p(j,3) - p(j,1)*x(i)))/(exp(p(j,3) - p(j,1)*x(i)) + 1)^2;
        v2(j,3)=(p(j,1)*p(j,2)*exp(p(j,3) - p(j,1)*x(i)))/(exp(p(j,3) - p(j,1)*x(i)) + 1)^2 -
(2*p(j,1)*p(j,2)*exp(2*p(j,3) - 2*p(j,1)*x(i)))/(exp(p(j,3) - p(j,1)*x(i)) + 1)^3;
        %v2=v2+[f24,f25,f26]';
    end
    v3=(2-1/x(i)).*v1+(x(i)-1).*v2;
    f0=(2-1/x(i)).*s1+(x(i)-1).*s2-x(i)^3+(2/5)/x(i);
    f=f+2.*f0.*v3;
end
%f=f';
end
```

待优化函数的梯度函数（两个程序都调用）

```
function test(p,n)
syms x;
f1=(x.^4)./5+1./(5.*x);
s=0;
for i=1:n
    s=s+p(i,2).*sigmoid(-p(i,1).*x+p(i,3));
end
f2=2/5+(x-1).*s;

f3=diff(f1,x);
f4=diff(f2,x);
y=1:0.1:2;
figure(1)
plot(y,eval(subs(f1,x,y)));
hold on
plot(y,eval(subs(f2,x,y)));
figure(2)
plot(y,eval(subs(f3,x,y)));
hold on
plot(y,eval(subs(f4,x,y)));
end
```

检验训练出的参数（通过与精确解的数值、导数进行比较）（两个程序都调用）

DFP

代码:

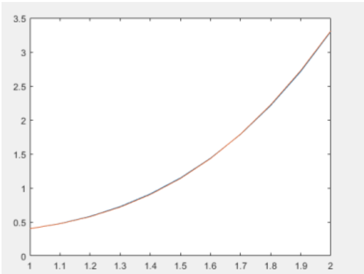
```
function [p,val,k]=dfp(fun,gfun,m,n)
%功能: 用DFP算法求解无约束问题: min f(x)
%输入: x0是初始点, fun, gfun分别是目标函数及其梯度
%输出: x, val分别是近似最优点和最优值, k是迭代次数.
%m为训练集数目, n为神经元数目
maxk=1e5; %给出最大迭代次数
rho=0.5;sigma=0.5; epsilon=1e-1;
k=0;
x=linspace(1,2,m);%构建训练集
p0=rand(n,3);%构建初始值（以rand为初始值）
Hk=ones(n,n,3);
Hk(:,:,1)=eye(n);%Bk=feval('Hess',x0);
Hk(:,:,2)=eye(n);
Hk(:,:,3)=eye(n);
dk=zeros(n,3);
while(k<maxk)
    gk=feval(gfun,x,p0,n,m); %计算梯度
    if(norm(gk)<epsilon), break; end %检验终止准则
    a=-Hk(:,:,1)*gk;
    b=-Hk(:,:,2)*gk;
    c=-Hk(:,:,3)*gk;
    dk(:,1)=a(:,1); %解方程组, 计算搜索方向(注意是左乘!)
    dk(:,2)=b(:,2);
    dk(:,3)=c(:,3);
    %dk=dk/norm(dk(1,:));
    %dk=-Hk*gk; %解方程组, 计算搜索方向
    m0=0; mk=0;
    while(m0<20) % 用Armijo搜索求步长
        if(feval(fun,x,p0+rho^m0*dk,n,m)<feval(fun,x,p0,n,m)+sigma*rho^m0*gk'*dk)
            mk=m0; break;
        end
        m0=m0+1;
    end
    %DFP校正
    p=p0+rho^mk*dk;
    sk=p-p0; yk=feval(gfun,x,p,n,m)-gk;
    %yk(1,:)=yk(1,:)/norm(yk(1,:));
    %yk(2,:)=yk(2,:)/norm(yk(2,:));
    %yk(3,:)=yk(3,:)/norm(yk(3,:));
    if(sk'*yk>0)
        Hk(:,:,1)=Hk(:,:,1)-(Hk(:,:,1)*sk(:,1)*sk(:,1)'*Hk(:,:,1))/(sk(:,1)'*Hk(:,:,1)*sk(:,1))+
(yk(:,1)*yk(:,1)')/(yk(:,1)'*sk(:,1));
        Hk(:,:,2)=Hk(:,:,2)-(Hk(:,:,2)*sk(:,2)*sk(:,2)'*Hk(:,:,2))/(sk(:,2)'*Hk(:,:,2)*sk(:,2))+
(yk(:,2)*yk(:,2)')/(yk(:,2)'*sk(:,2));
        Hk(:,:,3)=Hk(:,:,3)-(Hk(:,:,3)*sk(:,3)*sk(:,3)'*Hk(:,:,3))/(sk(:,3)'*Hk(:,:,3)*sk(:,3))+
(yk(:,3)*yk(:,3)')/(yk(:,3)'*sk(:,3));
    end
    k=k+1; p0=p;
end
val=feval(fun,x,p0,n,m);
```

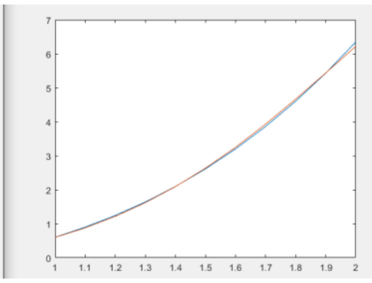
DFP 主程序

结果:
首先我们使用同一 m，n（10，10）值进行两算法比较:

BFGS

```
>> [p,val,k]=bfgs('fun','gfun',10,10)

p =

    0.6544   -0.6697    0.3669
    0.7590   -0.1788    0.8623
    1.2333    1.5235    2.5535
    0.8336    0.3044    1.3370
    1.3919    1.9887    3.0007
    1.0052    0.8000    1.8721
    2.1805    3.8999    4.8316
    0.9388   -0.4134    0.6623
    0.6314   -0.3219    0.7632
    0.8970    0.5496    1.5630


val =

    0.0287


k =

   635
```

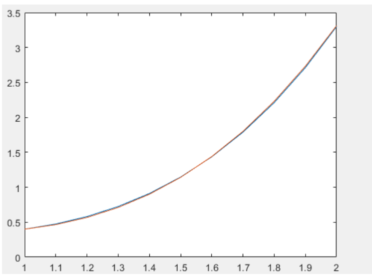


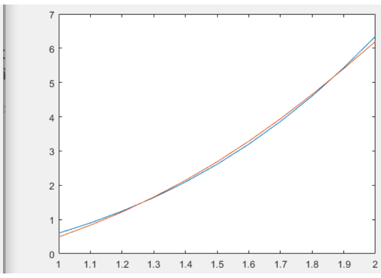


图一为 p 值，最终损失函数的值（0.0287）和迭代次数（635），图二为实验解与精确解的对比（可以看到几乎完全重合），图三是实验解与精确解的导数对比（重合程度极高）

DFP

```
>> [p,val,k]=dfp('fun','gfun',10,10)

p =

    0.8574    1.5753    2.4194
    0.5657    0.7688    1.4852
    0.9102    1.7124    2.5431
    0.7455    1.3429    2.1412
    1.9946    3.3355    4.6709
    0.9752    1.7583    2.6826
    0.8710    1.6480    2.4545
   -0.1254   -0.8873    0.0402
   -0.2485   -1.5144   -0.8210
    0.5013    0.1388    0.7289


val =

    0.0618


k =

        1725
```





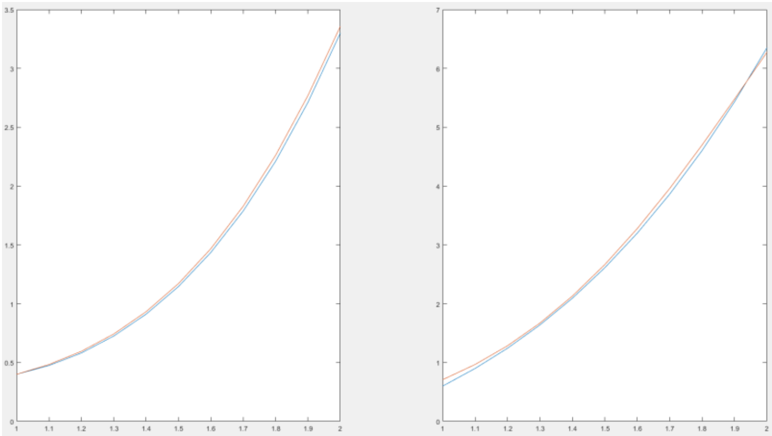
图一为 p 值，最终损失函数的值（0.0618）和迭代次数（1725），图二为实验解与精确解的对比（可以看到重合程度没有 BFGS 高），图三是实验解与精确解的导数对比（重合程度明显没有 BFGS 高）

我们可以发现，BFGS 在迭代次数和精确性上都明显优于 DFP。

接下来我们探究当前函数应该取什么 m,n 值才能训练出最优效果：（m 为训练集数目，n 为神经元数目）

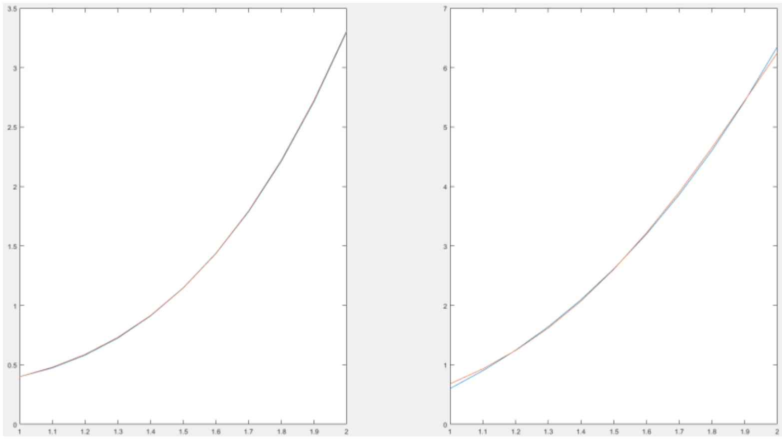
m=3，n=4：

```
>> [p,val,k]=bfgs('fun','gfun',3,4)

p =

    1.8221    2.3155    4.1069
    1.8420    2.2956    4.1372
    0.9282    0.3982    2.5399
    1.9804    2.5853    4.3616


val =

    0.0192


k =

    88
```



m=10，n=4

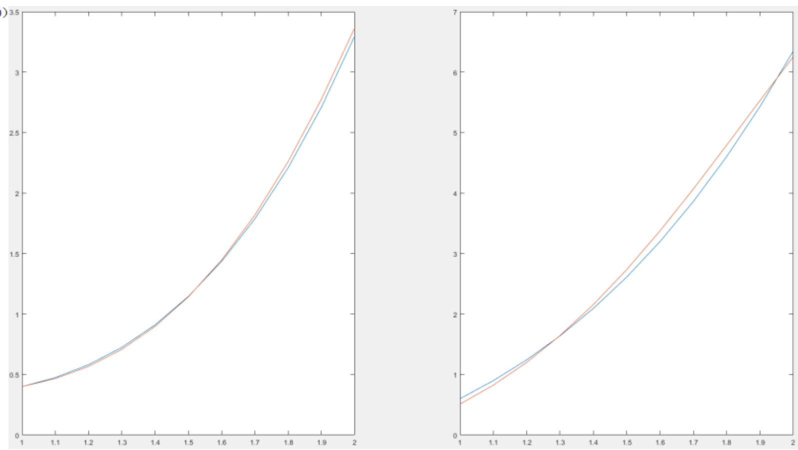
```
>> [p,val,k]=bfgs('fun','gfun',10,4)

p =

    0.9819    0.4642    2.6495
    2.0619    2.6374    4.5344
    2.0092    2.5778    4.4426
    1.6066    1.8741    3.7701


val =

    0.0218


k =

   214
```



在 n 不变的情况下，m 增大，效果更好（也就是训练集数目增大）

m=3，n=10：

```
>> [p,val,k]=bfgs('fun','gfun',3,10)

p =

    0.8222    0.3465    1.2967
    2.1196    3.6998    4.4466
    0.7493   -0.2143    0.7469
    1.4272    2.0501    2.9240
    0.8811    0.5605    1.5019
    0.7845    0.2073    1.2389
    0.8894   -0.4322    0.5612
    1.3176    1.7605    2.6565
    0.7586    0.0892    1.0378
    0.5673   -1.0118   -0.0074


val =

    0.0262


k =

   206
```



让人疑惑的是，在 m 不变的情况下，神经元数目增大后，反而效果不如之前，我猜想这里可能产生了过拟合的现象，导致训练反而变差。