HOMEWORK 2 - FRANCISCO CASTILLO

Table of Contents

Defined functions

```
function [a,b,c,d]=PadeVectors(N,f,x,h)
    % This function gives four vectors corresponding to the three
 diagonals
    % of the PADE scheme and the column vector of coefficients.
    % Arguments: -- N : size of the problem.
    %            -- f : function to approximate the derivative.
    %            -- x : vector with the nodes.
    %            -- h : step.

    a=ones(N+1,1);a(end)=2;
    b=4*ones(N+1,1);
    b(1)=1;b(end)=1;
    c=ones(N+1,1);c(1)=2;
    d=ones(N+1,1);
    d(1)=-2.5*f(x(1))+2*f(x(2))+0.5*f(x(3));
    for i=2:N
        d(i)=3*(f(x(i+1))-f(x(i-1)));
    end
    d(N+1)=2.5*f(x(N+1))-2*f(x(N))-0.5*f(x(N-1));
    d=d/h;
end
```

```
function [fp_BFD,fp_CFD]=FiniteDifferences(N,f,x,h,gp)
    % This code gives the first order BFD and second order
    % CFD approximations for the function f.
    % Arguments: -- N : size of the problem.
    %            -- f : function to approximate the derivative.
    %            -- x : vector with the nodes.
    %            -- h : step.
```

```
    %            -- gp: if the mesh is not equispaced, and it is
 expressed by an analytical function.
    fp_BFD=zeros(N+1,1);
    fp_CFD=zeros(N+1,1);
    if nargin == 4
    % Backward Finite Differences
        fp_BFD(1)=(f(x(2))-f(x(1)))/h;
        for i=2:N+1
           fp_BFD(i)=(f(x(i))-f(x(i-1)))/h;
        end
    % Central Finite Differences
        fp_CFD(1)=(f(x(2))-f(x(1)))/h;
        for i=2:N
            fp_CFD(i)=(f(x(i+1))-f(x(i-1)))/(2*h);
        end
        fp_CFD(N+1)=(f(x(N+1))-f(x(N)))/h;

    elseif nargin == 5
        % Backward Finite Differences
        fp_BFD(1)=gp(x(1))*(f(x(2))-f(x(1)))/h;
        for i=2:N+1
           fp_BFD(i)=gp(x(i))*(f(x(i))-f(x(i-1)))/h;
        end
        % Central Finite Differences
        fp_CFD(1)=gp(x(1))*(f(x(2))-f(x(1)))/h;
        for i=2:N
            fp_CFD(i)=gp(x(i))*(f(x(i+1))-f(x(i-1)))/(2*h);
        end
        fp_CFD(N+1)=gp(x(N+1))*(f(x(N+1))-f(x(N)))/h;
    else
        error('Wrong number of arguments')
    end
end



function d=GaussTriSol(a,b,c,d)
    N=length(a);
    for i=2:N
        b(i)=b(i)-c(i-1)*a(i)/b(i-1);
        d(i)=d(i)-d(i-1)*a(i)/b(i-1);
    end
    d(N)=d(N)/b(N);
    for i=N-1:-1:1
        d(i)=(d(i)-c(i)*d(i+1))/b(i);
    end
end



function L=ErrorNorm(v,v_app,k)
    N=length(v);
    if k==inf
        L=max(abs(v-v_app));
```

```
    elseif k==1
        L=sum(abs(v-v_app))/N;
    elseif k==2
        L=sqrt(sum((v-v_app).^2)/N);
    end
end
```

Problem 2

```
format long
clear all; close all; clc
```

I define the function and its derivative to do a preliminar inspection of the function to approximate. I plot both of them in figure 1.

```
f= @(x) (1+2*x.^2.*cos(x))./(x.^2.4);
fp = @(x) (2*x.^2.*(2*cos(x)-x.*sin(x))-2.4*(1+2*x.^2.*cos(x)))./
(x.^3.4);
xmax=3;
xmin=1;
x=linspace(xmin,xmax,200);

figure(1)
plot(x,f(x),'linewidth',2)
hold on
plot(x,abs(fp(x)),'linewidth',2)
grid on
set(gca,'fontsize',14)
title('Function to approximate','fontsize',20,'interpreter','latex')
xlabel('$x$','fontsize',20,'interpreter','latex')
legend({'$f(x)$','$f''(x)$'},'Interpreter','latex')
```

We can see in the figure how the function has a larger gradient in the left boundary than in the right one. We will see that the error is larger in the left boundary and I think the reason is this major gradient on the left boundary.

I prepare the following matrices to store the results required. In the following loop, I use 'PadeVectors' to obtain the four vectors of the PADE scheme for this problem. The input arguments are the number of elements, the function to approximate, the nodes to use and the spacing. I also use the function 'FiniteDifferences' which will give me the approximation from a backward first order finite differences and a central second order finite differences scheme. The function needs the same arguments as 'PadeVectors'. Lastly, I use the function 'GaussTriSol' to obtain the approximation of the PADE scheme once the tridiagonal system is solved. The inputs of this functions are the four vectors of the PADE scheme seen in class.

To finish the loop, I store the values of the error norms in the matrices defined. I have defined the function ErrorNorm that computes the error norm of the different approximations. For $k = 4$, I plot the error e_i vs x_i of the different schemes.

```
kmax=10;
T_BFD=zeros(kmax-1,4);
T_CFD=zeros(kmax-1,4);
T_PADE=zeros(kmax-1,4);
H=zeros(kmax-1,1);
for k=2:kmax
```

```
    N=2^k;
    h=(xmax-xmin)/(N);
    H(k-1)=h;
    x=xmin:h:xmax;
    [a,b,c,d]=PadeVectors(N,f,x,h);
    [fp_BFD,fp_CFD]=FiniteDifferences(N,f,x,h);
    fp_PADE=GaussTriSol(a,b,c,d);

    %Store results in matrices
    T_BFD(k-1,:)=[N ErrorNorm(fp(x)',fp_BFD,inf)...
        ErrorNorm(fp(x)',fp_BFD,1) ErrorNorm(fp(x)',fp_BFD,2)];
    T_CFD(k-1,:)=[N ErrorNorm(fp(x)',fp_CFD,inf)...
        ErrorNorm(fp(x)',fp_CFD,1) ErrorNorm(fp(x)',fp_CFD,2)];
    T_PADE(k-1,:)=[N ErrorNorm(fp(x)',fp_PADE,inf)...
        ErrorNorm(fp(x)',fp_PADE,1) ErrorNorm(fp(x)',fp_PADE,2)];
    if k==4
        figure(2)
        plot(x,fp(x)'-fp_BFD)
        hold on
        plot(x,fp(x)'-fp_BFD,'r*')
        grid on
        set(gca,'fontsize',14)
        title('Backward Finite
 Differences','fontsize',20,'interpreter','latex')
        xlabel('$x$','fontsize',20,'interpreter','latex')
        ylabel('Error $e_i$','fontsize',20,'interpreter','latex')

        figure(3)
        plot(x,fp(x)'-fp_CFD)
        hold on
        plot(x,fp(x)'-fp_CFD,'r*')
        grid on
        set(gca,'fontsize',14)
        title('Central Finite
 Differences','fontsize',20,'interpreter','latex')
        xlabel('$x$','fontsize',20,'interpreter','latex')
        ylabel('Error $e_i$','fontsize',20,'interpreter','latex')

        figure(4)
        plot(x,fp(x)'-fp_PADE)
        hold on
        plot(x,fp(x)-fp_PADE','r*')
        grid on
        set(gca,'fontsize',14)
        title('PADE Scheme','fontsize',20,'interpreter','latex')
        xlabel('$x$','fontsize',20,'interpreter','latex')
        ylabel('Error $e_i$','fontsize',20,'interpreter','latex')
    end
end
```

In the figure we can see that the error is larger (in absolute value) on the left boundary, as I expected from my argument above. In addition, the error in the left boundary is the same in both BFD (Backward Finite Differences) and CFD (Central Finite Differences) since I have used BFD for the left boundary in the CFD scheme.

As we could expect, the first order BFD has the largest error followed by the second order CFD and the fourth order PADE scheme. We will discuss more about this when I show the tables with the error norms.

In the following section of the code, I turn the matrices with the error norm results into a table to show them. I also calculate the order of the different approximations, store them into matrices and turn the matrices into tables to show the results.

```
% Present results in tables converting the matrices into tables
T_BFD=array2table(T_BFD,...
    'VariableNames',{'M','L_inf','L_1','L_2'});
T_CFD=array2table(T_CFD,...
    'VariableNames',{'M','L_inf','L_1','L_2'});
T_PADE=array2table(T_PADE,...
    'VariableNames',{'M','L_inf','L_1','L_2'});

o=zeros(kmax-1,4);
o(1,:)=NaN;
o(1,1)=4;
for k=1:kmax-2
    o(k+1,1)=2^(k+2);
    o(k+1,2)=log(T_BFD.L_inf(k)/T_BFD.L_inf(k+1))/...
         log(H(k)/H(k+1));
    o(k+1,3)=log(T_BFD.L_1(k)/T_BFD.L_1(k+1))/...
         log(H(k)/H(k+1));
    o(k+1,4)=log(T_BFD.L_2(k)/T_BFD.L_2(k+1))/...
         log(H(k)/H(k+1));
end
T_BFD_order=array2table(o,...
    'VariableNames',{'M','o_inf','o_1','o_2'});

for k=1:kmax-2
    o(k+1,1)=2^(k+2);
    o(k+1,2)=log(T_CFD.L_inf(k)/T_CFD.L_inf(k+1))/...
         log(H(k)/H(k+1));
    o(k+1,3)=log(T_CFD.L_1(k)/T_CFD.L_1(k+1))/...
         log(H(k)/H(k+1));
    o(k+1,4)=log(T_CFD.L_2(k)/T_CFD.L_2(k+1))/...
         log(H(k)/H(k+1));
end
T_CFD_order=array2table(o,...
    'VariableNames',{'M','o_inf','o_1','o_2'});

for k=1:kmax-2
    o(k+1,1)=2^(k+2);
    o(k+1,2)=log(T_PADE.L_inf(k)/T_PADE.L_inf(k+1))/...
         log(H(k)/H(k+1));
    o(k+1,3)=log(T_PADE.L_1(k)/T_PADE.L_1(k+1))/...
         log(H(k)/H(k+1));
    o(k+1,4)=log(T_PADE.L_2(k)/T_PADE.L_2(k+1))/...
         log(H(k)/H(k+1));
end
T_PADE_order=array2table(o,...
    'VariableNames',{'M','o_inf','o_1','o_2'});
```

```
% Show results for the first order BFD approximation
T_BFD
T_BFD_order
```

We see how the error norms get lower as we increase the number of elements M, as we expect. Looking at the order of the BFD scheme, we see that the order calculated with the infinity norm goes to 1, which makes sense sine the maximum error is at the boundary and we have used first order FFD to approximate the left boundary. The error calculated with the other two norms go to 1 as well since I have used first order BFD for the whole mesh. NOTE: the NaN row of the order table is because the order is calculated comparing two meshes, so having 9 different meshes I get 8 orders.

```
% Show results for the second order CFD approximation
T_CFD
T_CFD_order
```

We see that, naturally, the error norms get lower as we increase the number of elements M. Looking at the order of the BFD scheme, we see that the order calculated using the infinity norm goes to 1, which makes sence since we have used first order FFD to calculate the left boundary, where the maximum error is placed. In any case we have used first order BFD for the right boundary. The order calculated using the 1-norm goes to 2, which makes sense since the 1-norm averages the error in absolute value. Therefore, the error of the majority of the points contributes more than the error of the two points at the boundaries, regardless of being the largest. Hence, the order of the second order CFD using the 1-norm should, and it does, go to 2 as we increase the number of elements M. Lastly, since the 2-norm squares the errors, the errors of the boundaries gain importance and we have an order between 1 and 2.

```
% Show results for the fourth order PADE scheme approximation
T_PADE
T_PADE_order
```

Again we see that, the error norms get lower as we increase the number of elements M. In this case we see how we have the order calculated with the infinity norm going to 3, since the PADE scheme is order 3 at the boundaries and the maximum order is at the boundaries. On the other hand the order calculated with the 1-norm goes to 4 since the PADE scheme is order four for all the points not in the boundary. Finally, since the 2-norm squares the errors, the errors of the boundaries gain importance and we have an order between 3 and 4.

If we compare the norms of the different schemes, we see norms of the first order BFD is larger, followed by the second order CFD and the foruth order PADE scheme, as we discussed with the plots above.

Problem 3

For this problem the code is the same as in problem 2. I chose a non-equidistant mesh defined by $x_i = t_i^p$, where t_i are equispaced nodes. I run the code for several values of p and compare them.

```
N=2^4;
x=zeros(N+1,1);
t=zeros(N+1,1);
P=2:7;
T_BFD_var=zeros(length(P),4);
T_CFD_var=zeros(length(P),4);
for i=1:length(P)
    p=P(i);
    gp =  @(x) (x^(1/p-1))/p;
    t=linspace(xmin^(1/p),xmax^(1/p),N+1)';
```

```
    h=t(2)-t(1);
    x=t.^p;
    h_var=zeros(length(x),1);
    for j=1:length(x)-1
        h_var(j+1)=x(j+1)-x(j);
    end
    figure(5)
    V(i)=plot(x(2:end),h_var(2:end));
    hold on
    plot(x(2:end),h_var(2:end),'r.')
    grid on
    axis([1 3 0.06 0.2])
    set(gca,'fontsize',14)
    title('$h_i$ vs $x$','fontsize',20,'interpreter','latex')
    xlabel('$x$','fontsize',20,'interpreter','latex')
    ylabel('$h_i$','fontsize',20,'interpreter','latex')

    [fp_BFD_var,fp_CFD_var]=FiniteDifferences(N,f,x,h,gp);

    figure(6)
    plot(x,fp(x)-fp_BFD_var)
    hold on
    plot(x,fp(x)-fp_BFD_var,'r.')
    grid on
    set(gca,'fontsize',14)
    title('Backward Finite
 Differences','fontsize',20,'interpreter','latex')
    xlabel('$x$','fontsize',20,'interpreter','latex')
    ylabel('Error $e_i$','fontsize',20,'interpreter','latex')

    figure(7);
    plot(x,fp(x)-fp_CFD_var)
    hold on
    plot(x,fp(x)-fp_CFD_var,'r.')
    grid on
    set(gca,'fontsize',14)
    title('Central Finite
 Differences','fontsize',20,'interpreter','latex')
    xlabel('$x$','fontsize',20,'interpreter','latex')
    ylabel('Error $e_i$','fontsize',20,'interpreter','latex')

    % Store results in matrices
    T_BFD_var(i,:)=[p ErrorNorm(fp(x),fp_BFD_var,inf)...
        ErrorNorm(fp(x),fp_BFD_var,1) ErrorNorm(fp(x),fp_BFD_var,2)];
    T_CFD_var(i,:)=[p ErrorNorm(fp(x),fp_CFD_var,inf)...
        ErrorNorm(fp(x),fp_CFD_var,1) ErrorNorm(fp(x),fp_CFD_var,2)];
end
legend([V(1) V(2) V(3) V(4) V(5) V(6)],
{'$p=2$','$p=3$','$p=4$','$p=5$','$p=6$','$p=7$',},'Interpreter','latex','Location
```

In the first figure we see that obviously as we increase p the mesh gets more compact to the left side. I expect that to improve the accuracy of the approximation, as we can see in the other two figures. For both BFD and CFD approximations we see how the error of the majority of the points decrease with p. The error at the left boundary also decreases as we are increasing the number of nodes in that area whereas the

error at the right boundary seems to increase a bit since we are taking points away from it. However, since the maximum error is in the left boundary we see in the tables how the infinity norm decreases with p. We can also see how the other two remaining norms of the error also decrease with p for both approximations. This indicates that the source of the error was in fact the large gradient discussed at the beginning of this assignment. To finish, we see as, for any value of p the norms are under the threshold given by the problem, the norms of the equidistant second order CFD approximation.

```
% Present results in tables converting the matrices into tables
T_BFD_var=array2table(T_BFD_var,...
    'VariableNames',{'p','L_inf','L_1','L_2'})
T_CFD_var=array2table(T_CFD_var,...
    'VariableNames',{'p','L_inf','L_1','L_2'})
```

Equations for the first order Backward Finite Differences

I have used

$$f_i' = \frac{f_i - f_{i-1}}{h}, \qquad i = 2, ..., N+1,$$

for all nodes except the first one, with a first order forward finite difference at the left boundary

$$f_1' = \frac{f_2 - f_1}{h}.$$

Equations for the second order Central Finite Differences

I have used

$$f_i' = \frac{f_{i+1} - f_{i-1}}{2h}, \qquad i = 2, ..., N$$

for all nodes except the first and the last, with a first order forward finite difference at the left boundary

$$f_1' = \frac{f_2 - f_1}{h},$$

and a first order backward finite difference at the right boundary

$$f_{N+1}' = \frac{f_{N+1} - f_N}{h}, \qquad i = N+1.$$

Equations for the fourth order PADE Scheme

I have used

$$f_{i-1}' + 4f_i' + f_{i+1}' = \frac{3}{h}(f_{i+1} - f_{i-1}), \qquad i = 2, ..., N$$

for all nodes except the first and the last, with a third order formula for the left boundary

$$f_1' + 2f_2' = \frac{1}{h}\left(-\frac{5}{2}f_1 + 2f_2 + \frac{1}{2}f_3\right),$$

and a a third order formula for the right boundary

$$2f_N' + f_{N+1}' = \frac{1}{h}\left(\frac{5}{2}f_{N+1} - 2f_N - \frac{1}{2}f_{N-1}\right),$$

Equations for the first order Backward Finite Differences with non-equidistant mesh

Since

$x = t^p,$

$t = g(x) = x^{1/p},$

and

$g'(x) = \frac{1}{p}x^{\frac{1}{p}-1}.$

Now, to compute the first derivative we use the chain rule

$$\frac{df}{dx} = \frac{dt}{dx}\frac{df}{dt} = g'(x)\frac{df}{dt}$$

and we can apply normal finite difference to the equispaced coordinate t.

Therefore, by the chain rule as we saw in class, I have used that

$$f_i' = g'(x_i)\frac{f_i - f_{i-1}}{h}, \qquad i = 2, ..., N+1,$$

for all nodes except the first one, with a first order forward finite difference at the left boundary

$$f_1' = g'(x_1)\frac{f_2 - f_1}{h}.$$

Equations for the second order Central Finite Differences with non-equidistant mesh

Again, by the chain rule as we saw in class, I have used that

$$f_i' = g'(x_i)\frac{f_{i+1} - f_{i-1}}{2h}, \qquad i = 2, ..., N$$

for all nodes except the first and the last, with a first order forward finite difference at the left boundary

$$f_1' = g'(x_1)\frac{f_2 - f_1}{h},$$

and a first order backward finite difference at the right boundary

$$f_{N+1}' = g'(x_{N+1})\frac{f_{N+1} - f_N}{h}.$$

Published with MATLAB® R2017a