APPM4650 Homework10

Olivia Golden

November 2021

1. (a)

```
function adaptive_quad_example

    f = @(x) sin(1./x);
    a = 0.1;
    b = 2;

    tol = 1e-3;
    Nmax = 6;

    [app,ier] = adaptive_quad(f,a,b,tol,Nmax);

    qqex = quad(f,a,b);

           fprintf('error in quad is  %d .\n',abs(qqex-app));

    keyboard

    return

function [app,ier] = adaptive_quad(f,a,b,tol,Nmax)


    % Intlist  = interval list to be processed
    % Intsave = integral approximations for the corresponding intervals
    % Levels = number of level of the interval

    % initialize everything - these are for processing
    Intlist(:,1) = [a,b];
    Intsave(1) = gauss(a,b,f, Nmax); %or simpsons or %trap

    % create arrays for storing adaptive mesh
    Intkeep = [];
    Intvalkeep = [];
```

```
app = 0; % our approximation using the adaptive scheme

i = 1;  % counter for interval that need to be processed.
Levels(1) = 0;

while i>0
    %     split interval in half
    aloc = Intlist(1,1); bloc = Intlist(2,1); %reads endpoints
    xmid = (aloc+bloc)/2; % computes midpoint
    Q1 = gauss(aloc,xmid,f,Nmax); %or simpsons or trap
    Q2 = gauss(xmid,bloc,f,Nmax); %or simpsons or trap
    if (abs(Intsave(1)-(Q1+Q2))<tol)
        app = app + (Q1+Q2);
        %         Store the adaptive mesh
        Intkeep = [Intkeep; [aloc xmid; xmid bloc]];
        Intvalkeep = [Intvalkeep; Q1; Q2];
        %         Delete information
        Intlist(:,1) = [];
        Intsave(1) = [];
        Levels(1) = [];
        i = i -1; % reduce interval count by 1
    else
        Nlevel = Levels(1) + 1;
        if Nlevel >Nmax
            disp('error: did not converge')
            ier = 1;
            return
        end
        %         add the sub intervals to the list to be processed.
        Intlist= [Intlist [aloc;xmid]];
        Intsave = [Intsave; Q1];
        Levels = [Levels; Levels(1)+1];
        Intlist= [Intlist [xmid;bloc]];
        Intsave = [Intsave; Q2];
        Levels = [Levels; Levels(1)+1];
        i = i + 1;

        %         Remove the information for the interval we just checked.
        Intlist(:,1) = [];
        Intsave(1) = [];
        Levels(1) = [];
    end
end
```

```
        % plot adaptive mesh
        xvals = Intkeep(:,1);
        xvals = unique(xvals);
        xvals = [xvals; b];

        fx = f(xvals);
        xx = linspace(a,b,1000);
        plot(xx,f(xx),'-',xvals,fx,'ro')


        ier = 0;
        disp('Sucess: adaptive quad')
        fprintf('number of intervals is  %d .\n',size(Intkeep,1));

        return


function Qapp = composite_simpson(a,b,f,n)


    h = (b-a)/n;
    qnode = a+[0:n]*h;
    Qapp = h/3*(f(a)+f(b) + 2*sum(f(qnode(3:2:end-1)))+4*sum(f(qnode(2:2:end-1))));

    return

function Qapp1 = composite_trap(a,b,f,n)
    h = (b-a)/n;
    qnode = a+[0:n]*h;
    Qapp1 = h/2*(f(a)+f(b)+2*sum(f(qnode(2:end-1))));

    return

function Qapp2 = gauss(a,b,f,n)

    [x,w]=lgwt(n,a,b);
    fx = f(x);
    Qapp2 = fx'*w;

    return

function [x,w]=lgwt(N,a,b)
% lgwt.m
%
% This script is for computing definite integrals using Legendre-Gauss
% Quadrature. Computes the Legendre-Gauss nodes and weights  on an interval
```

```
% [a,b] with truncation order N
%
% Suppose you have a continuous function f(x) which is defined on [a,b]
% which you can evaluate at any x in [a,b]. Simply evaluate it at all of
% the values contained in the x vector to obtain a vector f. Then compute
% the definite integral using sum(f.*w);
%
% Written by Greg von Winckel - 02/25/2004
N=N-1;
N1=N+1; N2=N+2;
xu=linspace(-1,1,N1)';
% Initial guess
y=cos((2*(0:N)'+1)*pi/(2*N+2))+(0.27/N1)*sin(pi*xu*N/N2);
% Legendre-Gauss Vandermonde Matrix
L=zeros(N1,N2);
% Derivative of LGVM
Lp=zeros(N1,N2);
% Compute the zeros of the N+1 Legendre Polynomial
% using the recursion relation and the Newton-Raphson method
y0=2;
% Iterate until new points are uniformly within epsilon of old points
while max(abs(y-y0))>eps


    L(:,1)=1;
    Lp(:,1)=0;

    L(:,2)=y;
    Lp(:,2)=1;

    for k=2:N1
        L(:,k+1)=( (2*k-1)*y.*L(:,k)-(k-1)*L(:,k-1) )/k;
    end

    Lp=(N2)*( L(:,N1)-y.*L(:,N2) )./(1-y.^2);

    y0=y;
    y=y0-L(:,N2)./Lp;

end
% Linear map from[-1,1] to [a,b]
x=(a*(1-y)+b*(1+y))/2;
% Compute the weights
w=(b-a)./((1-y.^2).*Lp.^2)*(N2/N1)^2;
return
```

(b) Gauss - number of intervals is 8, error in quad is 6.286075e-06.
Simpsons - number of intervals is 8, error in quad is 8.295836e-04.
Trapezoidal - number of intervals is 12, error in quad is 3.596838e-04 .

Here, Simpson's does the best since it produces the lowest error if the fewest amount of intervals.

2. $t = x^{-1} = \frac{1}{x}$
$dt = -x^{-2}dx$
$-x^2dt = dx$
$-\frac{1}{t^2}dt = dx$
$(1)^{-1} = 1, (\infty)^{-1} = 0$
$\int_1^0 \frac{cos(\frac{1}{t})}{\frac{1}{t}^3}(\frac{-dt}{t^2}) = \int_1^0 \frac{-t^3cos(\frac{1}{t})}{t^2}dt = \int_1^0 -tcos(\frac{1}{t})dt = \int_0^1 tcos(\frac{1}{t})dt$
Here $f(0)$ is undefined, so we can take a limit.
$\sqrt{t} \leq tcos(1/t) \leq t^2$
$\lim_{t\to 0}\sqrt{t} = 0$
$\lim_{t\to 0} t^2 = 0$
Therefore, by the Squeeze Thm, $\lim_{t\to 0} tcos(1/t) = 0$ and we can substitute 0 for $f(a)$ in the code below.
Using

```
def simpsons(a,b,n,f):
    h=(b-a)/n
    xl0=0+f(b)
    xl1=0
    xl2=0
    for i in range(1,n):
        x=a+i*h
        if (i%2==0):
            xl2+=f(x)
        else:
            xl1+=f(x)
    xl=h*(xl0+2*xl2+4*xl1)/3
    return xl
fx=lambda x: x*np.cos(1/x)
simpsons(0,1,5,fx)
```

The integral evaluates to 0.02673976175994018.

3. (a)

```
def trapezoidal(a,b,n,f):
    h = (b-a)/n
    x1=((f(a)+f(b))/2)
    counter=0
```

```
    for i in range(1, n):
        x=a+i*h
        x1+=f(x)
    x1*=h
    return x1
def gamma(x):
    integral= lambda t: t**(x-1)*math.exp(-t)
    return trapezoidal(0,20,6000,integral)
```

After graphing $(x-1)!$ it becomes clear that function is most different on the interval $[0, 10]$. After, the function increases very quickly. I choose 20 to account for this, and found that a good N was a higher one, 6000.

```
For value 2
Error: 0.036228329697172934
For value 4
Error: 0.00024312630502916951
For value 6
Error: 7.569927745289343e-05
For value 8
Error: 0.000791137395499607
For value 10
Error: 0.005054551392167666
```

(b)
```
def gamma1(x):
    integral1= function_counter(lambda t: t**(x-1)*np.exp(-t))
    return integrate.quadrature(integral1, 0, 20)[0]
```

```
For value 2
Error of a): 9.692096678559636e-07
Error of b): 4.3063422827671616e-08
Number of iterations of a) 30
Number of iterations of b) 13
For value 4
Error of a): 3.2037217678748675e-06
Error of b): 3.2033268014449354e-06
Number of iterations of a) 30
Number of iterations of b) 14
For value 6
Error of a): 7.190887869299445e-05
Error of b): 7.190863810286885e-05
Number of iterations of a) 30
Number of iterations of b) 15
For value 8
Error of a): 0.0007785903975523358
Error of b): 0.0007785891939604317
Number of iterations of a) 30
Number of iterations of b) 15
For value 10
Error of a): 0.004995413789309223
Error of b): 0.004995411096648301
Number of iterations of a) 30
Number of iterations of b) 15
```

The error in both methods are similar. In addition, the iterations of the built in method are consistently about half of that of the method from part a).

(c)
```
def gamma2(x):
    [x1,w]=np.polynomial.laguerre.laggauss(x)
    f=lambda t: t**(x-1)
    fx=f(x1)
    return sum(fx*w)
```

This method is extremely close to the true value of $\Gamma(x)$.

```
For value 2
Error: 0.0
For value 4
Error: 2.9605947323337506e-16
For value 6
Error: 4.736951571734001e-16
For value 8
Error: 7.218211918832764e-16
For value 10
Error: 4.812141279221843e-16
```