This semester’s term project consists of a friendly competition to write the most efficient Adaptive Quadrature code modeled along the same lines as the (much more sophisticated) widely available Variable Step Variable Order solvers for IVPs of ODEs. Model your program after the stepwise midpoint/trapezoidal method we discussed yesterday.

Thus, while it might be possible to obtain a more efficient adaptive quadrature routine by some other approach, like the recursive Simpson approach also discussed on Tuesday, one of the main purposes of the assignment is to gain some insight into the workings of modern VSVO solvers.

Your code must be in Matlab. Use the following template:

```matlab
% Matlab template.m
function R = adaptq(f,a,b,tau)
    % insert your code here:
    end
```

Your function should return an approximation $R$ of the definite integral $I = \int_a^b f(x)dx$ such that

$$|I - R| \leq \tau \quad (1)$$

Following is an example that shows how to use a Matlab file to compute the trapezoidal approximation of

$$E = \int_0^1 e^x dx$$

The Matlab file `trapezoidal.m` containing the (very simple and non-adaptive) Quadrature code is

```matlab
% Matlab implementation of the trapezoidal rule, trapezoidal.m
function R = trapezoidal(f,a,b,n)
    h = (b-a)/n;
    y = h*(f(a)+f(b))/2;
    for i = 1:n-1
        y = y + h*f(a+i*h);
    end
    R = y;
end
```

The function $f$ is defined in another Matlab file as

```matlab
% Matlab function definition, f.m
function R = f(x)
```
Both of those files can be used in Matlab as follows:

```matlab
% Log of Matlab session, with blank lines removed
sunbird 57> matlab
< M A T L A B >
Copyright 1984-2006 The MathWorks, Inc.
Version 7.3.0.298 (R2006b)
August 03, 2006
To get started, type one of these: helpwin, helpdesk, or demo.
For product information, visit www.mathworks.com.
>> global n
>> n = 0;
>> trapezoidal(@f,0,1,20)
ans =
    1.7186
>> n
n =
    21
```

Notice the “@” character in front of the function name in the call to trapezoidal.

**Code requirements**

- Your code should give an approximation within the specified tolerance of $\tau$, and otherwise it should use as few function values as possible.

- It should not produce any output other than the approximation $R$ of the integral $I$. (Use semicolons to suppress output. Of course, you can omit semicolons during development for debugging purposes.)

- Your code should of course run without syntax or runtime errors.
  You can use similar files as those above for your testing, and you will spend a lot of time thinking about your best approach, and testing your quadrature program.
Specifically, do the following:

0. Decide whether to work by yourself or with another student in our class. If you work with a partner hand in just one set of answers. You and your partner will receive the same score for this assignment.

1. On Wednesday, March 16, 2022 give a presentation to the whole class. Describe what you have done so far, what you are still planning to do, what your experiences have been, and what you have learned so far. This presentation is part of the assignment, and you should make sure to attend class on that particular day.

2. Email me your Matlab code by Friday, April 8, 2022. I will test your code on several examples, including those described below, and some others. I will also compare it with commercial or public quadrature and ODE software.

3. In class on April 8 hand in a written description of your code, including a mathematical description of how it works.

4. Prepare another presentation and give it during our discussion of the term project which will take place after I have tested all codes. This presentation should include distribution of a written report that may be the same as the one you handed in to me.

5. Your goal is to use fewer function evaluations than anybody else while meeting the error requirement.

Test Examples

I will test your code on several examples, details of which you will learn when we discuss the results of the project. This is realistic, if you develop code professionally you don’t beforehand what somebody will use it for. However, I will also use the following, or very similar, examples:

\[
[a, b] = [0, 1], \quad f(x) = e^x, \quad \tau = 10^{-3}, 10^{-4}, \ldots, 10^{-12}. 
\]  

\[
[a, b] = [0, 2N\pi], \quad f(x) = \sin x, \quad N = 1, 20, 100, \quad \tau = 10^{-3}, 10^{-4}, \ldots, 10^{-12}. 
\]  

\[
[a, b] = [-1, \pi], \quad f(x) = \arctan(Nx), \quad N = 1, 20, 100, \quad \tau = 10^{-3}, 10^{-4}, \ldots, 10^{-12}. 
\]

The first example, the exponential, has all derivatives the same, which makes it easier to compare methods of different orders. The second example has an answer 0 which causes difficulties for any method that attempts to control the relative error. Of course, in this project we use absolute error control, see (1). The last example has an increasingly pronounced step. Presumably your code will have to work harder and harder to get past the step.

Resources

There is of course a huge number of books on Matlab, and much information is built into Matlab or available online. My own main source of information on Matlab is
VSVO methods

Two famous public domain VSVO methods are lsode (Livermore Solver of Ordinary Differential Equations) by Alan Hindmarsh, and the Shampine Gordon code documented in Shampine and Gordon, Computer Solution of Ordinary Differential Equations, The Initial Value Problem, Freeman and Co., 1975, ISBN 0716704617. Note that lsode.f is not complete; it uses LINPACK routines that are also public, but not contained in the lsode distribution.

These codes are written in Fortran. For your information and entertainment I put links to them onto our class web page.

Several ODE codes are built into Matlab. In particular, ode113 and ode15s are VSVO methods.

Of course one would expect that a special purpose method for quadrature problems performs better than a general purpose method for ODE-IVPs. On the other hand, those are professional codes that presumably adhere to more stringent standards than our codes.

We will see how all that software will stack up!

Following are the instructions for the use of lsode and shpgor. Contrast the complexity of those codes with the complexity of what we are contemplating in this semester project.
SUBROUTINE LSODE (F, NEQ, Y, T, TOUT, ITOL, RTOL, ATOL, ITASK,
  ISTATE, IOPT, RWORK, LRW, IWORK, LIW, JAC, MF)
EXTERNAL F, JAC
INTEGER NEQ, ITOL, ITASK, ISTATE, IOPT, LRW, IWORK, LIW, MF
DOUBLE PRECISION Y, T, TOUT, RTOL, ATOL, RWORK
DIMENSION NEQ(1), Y(1), RTOL(1), ATOL(1), RWORK(LRW), IWORK(LIW)

C THIS IS THE AUGUST 13, 1981 VERSION OF
C LSODE.. LIVERMORE SOLVER FOR ORDINARY DIFFERENTIAL EQUATIONS.
C THIS VERSION IS IN DOUBLE PRECISION.
C
C LSODE SOLVES THE INITIAL VALUE PROBLEM FOR STIFF OR NONSTIFF
C SYSTEMS OF FIRST ORDER ODE-S,
C  \[ \frac{dY}{dt} = F(t, Y) \], OR, IN COMPONENT FORM,
C  \[ \frac{dY(i)}{dt} = F(i, t, Y(1), Y(2), \ldots, Y(NEQ)) \] (i = 1, \ldots, NEQ).
C LSODE IS A PACKAGE BASED ON THE GEAR AND GEARB PACKAGES, AND ON THE
C OCTOBER 23, 1978 VERSION OF THE TENTATIVE ODEPACK USER INTERFACE
C STANDARD, WITH MINOR MODIFICATIONS.
C
C REFERENCE..
C ALAN C. HINDMARSH, LSODE AND LSODI, TWO NEW INITIAL VALUE
C ORDINARY DIFFERENTIAL EQUATION SOLVERS,
C
C AUTHOR AND CONTACT. ALAN C. HINDMARSH,
C MATHEMATICS AND STATISTICS DIVISION, L-316
C LAWRENCE LIVERMORE NATIONAL LABORATORY
C LIVERMORE, CA 94550.
C
C SUMMARY OF USAGE.
C
C COMMUNICATION BETWEEN THE USER AND THE LSODE PACKAGE, FOR NORMAL
C SITUATIONS, IS SUMMARIZED HERE. THIS SUMMARY DESCRIBES ONLY A SUBSET
C OF THE FULL SET OF OPTIONS AVAILABLE. SEE THE FULL DESCRIPTION FOR
C DETAILS, INCLUDING OPTIONAL COMMUNICATION, NONSTANDARD OPTIONS,
C AND INSTRUCTIONS FOR SPECIAL SITUATIONS. SEE ALSO THE EXAMPLE
C PROBLEM (WITH PROGRAM AND OUTPUT) FOLLOWING THIS SUMMARY.
C
C A. FIRST PROVIDE A SUBROUTINE OF THE FORM..
C SUBROUTINE F (NEQ, T, Y, YDOT)
C DIMENSION Y(NEQ), YDOT(NEQ)
C WHICH SUPPLIES THE VECTOR FUNCTION F BY LOADING YDOT(I) WITH F(I).
C
C B. NEXT DETERMINE (OR GUESS) WHETHER OR NOT THE PROBLEM IS STIFF.
C STIFFNESS OCCURS WHEN THE JACOBIAN MATRIX DF/DY HAS AN EIGENVALUE
C WHOSE REAL PART IS NEGATIVE AND LARGE IN Magnitude, COMPARED TO THE
C RECIPROCAL OF THE T SPAN OF Interest. IF THE PROBLEM IS NONSTIFF,
C USE A METHOD FLAG MF = 10. IF IT IS STIFF, THERE ARE FOUR STANDARD
C CHOICES FOR MF, AND LSODE REQUIRES THE JACOBIAN MATRIX IN SOME FORM.
C THIS MATRIX IS REGARDED EITHER AS FULL (MF = 21 OR 22),
C OR BANDED (MF = 24 OR 25). IN THE BANDED CASE, LSODE REQUIRES TWO
C HALF-BANDWIDTH PARAMETERS ML AND MU. THESE ARE, RESPECTIVELY, THE
C WIDTHS OF THE LOWER AND UPPER PARTS OF THE BAND, EXCLUDING THE MAIN
C DIAGONAL. Thus the BAND CONSISTS OF THE LOCATIONS (I, J) WITH
C
C C. IF THE PROBLEM IS STIFF, YOU ARE ENCOURAGED TO SUPPLY THE JACOBIAN
C DIRECTLY (MF = 21 OR 24), BUT IF THIS IS NOT FEASIBLE, LSODE WILL
C COMPUTE IT INTERNALLY BY DIFFERENCE QUOTIENTS (MF = 22 OR 25).
C IF YOU ARE SUPPLYING THE JACOBIAN, PROVIDE A SUBROUTINE OF THE FORM..
C SUBROUTINE JAC (NEQ, T, Y, ML, MU, PD, NROWPD)
C DIMENSION Y(NEQ), PD(NROWPD,NEQ)
C WHICH SUPPLIES DF/DY BY LOADING PD AS FOLLOWS..
C FOR A FULL JACOBIAN (MF = 21), LOAD PD(I,J) WITH DF(I)/DY(J),
C THE PARTIAL DERIVATIVE OF F(I) WITH RESPECT TO Y(J). (IGNORE THE
C ML AND MU ARGUMENTS IN THIS CASE.)
C FOR A BANDED JACOBIAN (MF = 24), LOAD PD(I-J+MU+1,J) WITH
C DF(I)/DY(J), I.E. LOAD THE DIAGONAL LINES OF DF/DY INTO THE ROWS OF
C PD FROM THE TOP DOWN.
C IN EITHER CASE, ONLY NONZERO ELEMENTS NEED BE LOADED.
C
C D. WRITE A MAIN PROGRAM WHICH CALLS SUBROUTINE LSODE ONCE FOR
C EACH POINT AT WHICH ANSWERS ARE DESIRED. THIS SHOULD ALSO PROVIDE
C FOR POSSIBLE USE OF LOGICAL UNIT 6 FOR OUTPUT OF ERROR MESSAGES
C BY LSODE. ON THE FIRST CALL TO LSODE, SUPPLY ARGUMENTS AS FOLLOWS..
C F = NAME OF SUBROUTINE FOR RIGHT-HAND SIDE VECTOR F.
C THIS NAME MUST BE DECLARED EXTERNAL IN CALLING PROGRAM.
C NEQ = NUMBER OF FIRST ORDER ODE-S.
C Y = ARRAY OF INITIAL VALUES, OF LENGTH NEQ.
C T = THE INITIAL VALUE OF THE INDEPENDENT VARIABLE.
C TOUT = FIRST POINT WHERE OUTPUT IS DESIRED (.NE. T).
C ITOL = 1 OR 2 ACCORDING AS ATOL (BELOW) IS A SCALAR OR ARRAY.
C RTOL = RELATIVE TOLERANCE PARAMETER (SCALAR).
C ATOL = ABSOLUTE TOLERANCE PARAMETER (SCALAR).
C THE ESTIMATED LOCAL ERROR IN Y(I) WILL BE CONTROLLED SO AS
C TO BE ROUGHLY LESS (IN MAGNITUDE) THAN
C EWT(I) = RTOL*ABS(Y(I)) + ATOL IF ITOL = 1, OR
C EWT(I) = RTOL*ABS(Y(I)) + ATOL(I) IF ITOL = 2.
C THUS THE LOCAL ERROR TEST PASSES IF, IN EACH COMPONENT,
C EITHER THE ABSOLUTE ERROR IS LESS THAN ATOL (OR ATOL(I)),
C OR THE RELATIVE ERROR IS LESS THAN RTOL.
C USE RTOL = 0.0 FOR PURE ABSOLUTE ERROR CONTROL, AND
C USE ATOL = 0.0 (OR ATOL(I) = 0.0) FOR PURE RELATIVE ERROR
C CONTROL. CAUTION . ACTUAL (GLOBAL) ERRORS MAY EXCEED THESE
C LOCAL TOLERANCES, SO CHOOSE THEM CONSERVATIVELY.
C ITASK = 1 FOR NORMAL COMPUTATION OF OUTPUT VALUES OF Y AT T = TOUT.
C ISTATE = INTEGER FLAG (INPUT AND OUTPUT). SET ISTATE = 1.
C IOPT = 0 TO INDICATE NO OPTIONAL INPUTS USED.
C RWORK = REAL WORK ARRAY OF LENGTH AT LEAST..
C 20 + 16*NEQ FOR MF = 10,
C 22 + 9*NEQ + NEQ**2 FOR MF = 21 OR 22,
C 22 + 10*NEQ + (2*ML + MU)*NEQ FOR MF = 24 OR 25.
C LRW = DECLARED LENGTH OF RWORK (IN USER-S DIMENSION).
C IWORK = INTEGER WORK ARRAY OF LENGTH AT LEAST..
C 20 FOR MF = 10,
C 20 + NEQ FOR MF = 21, 22, 24, OR 25.
C IF MF = 24 OR 25, INPUT IN IWORK(1),IWORK(2) THE LOWER
C AND UPPER HALF-BANDWIDTHS ML,MU.
C LIW = DECLARED LENGTH OF IWORK (IN USER-S DIMENSION).
C JAC = NAME OF SUBROUTINE FOR JACOBIAN MATRIX (MF = 21 OR 24).
C IF USED, THIS NAME MUST BE DECLARED EXTERNAL IN CALLING
C PROGRAM. IF NOT USED, PASS A DUMMY NAME.
C MF = METHOD FLAG. STANDARD VALUES ARE..
116 C 10 FOR NONSTIFF (ADAMS) METHOD, NO JACOBIAN USED.
117 C 21 FOR STIFF (BDF) METHOD, USER-SUPPLIED FULL JACOBIAN.
118 C 22 FOR STIFF METHOD, INTERNALLY GENERATED FULL JACOBIAN.
119 C 24 FOR STIFF METHOD, USER-SUPPLIED BANDED JACOBIAN.
120 C 25 FOR STIFF METHOD, INTERNALLY GENERATED BANDED JACOBIAN.
121 C NOTE THAT THE MAIN PROGRAM MUST DECLARE ARRAYS Y, RWORK, IWORK,
122 C AND POSSIBLY ATOL.
123 C
124 C E. THE OUTPUT FROM THE FIRST CALL (OR ANY CALL) IS..
125 C   Y = ARRAY OF COMPUTED VALUES OF Y(T) VECTOR.
126 C   T = CORRESPONDING VALUE OF INDEPENDENT VARIABLE (NORMALLY TOUT).
127 C   ISTATE = 2 IF LSODE WAS SUCCESSFUL, NEGATIVE OTHERWISE.
128 C   -1 MEANS EXCESS WORK DONE ON THIS CALL (PERHAPS WRONG MF).
129 C   -2 MEANS EXCESS ACCURACY REQUESTED (TOLERANCES TOO SMALL).
130 C   -3 MEANS ILLEGAL INPUT DETECTED (SEE PRINTED MESSAGE).
131 C   -4 MEANS REPEATED ERROR TEST FAILURES (CHECK ALL INPUTS).
132 C   -5 MEANS REPEATED CONVERGENCE FAILURES (PERHAPS BAD JACOBIAN
133 C   SUPPLIED OR WRONG CHOICE OF MF OR TOLERANCES).
134 C   -6 MEANS ERROR WEIGHT BECAME ZERO DURING PROBLEM. (SOLUTION
135 C   COMPONENT I VANISHED, AND ATOL OR ATOL(I) = 0.)
136 C
137 C F. TO CONTINUE THE INTEGRATION AFTER A SUCCESSFUL RETURN, SIMPLY
138 C RESET TOUT AND CALL LSODE AGAIN. NO OTHER PARAMETERS NEED BE RESET.
139 C
140 C-------------------------------------------------------------------------------------
SUBROUTINE DE(F,NEQN,Y,T,TOUT,RELERR,ABSERR,IFLAG)

SUBROUTINE DE INTEGRATES A SYSTEM OF UP TO 20 FIRST ORDER ORDINARY
DIFFERENTIAL EQUATIONS OF THE FORM
DY(I)/DT = F(T,Y(1),Y(2),...,Y(NEQN))
Y(I) GIVEN AT T.

THE SUBROUTINE INTEGRATES FROM T TO TOUT. ON RETURN THE
PARAMETERS IN THE CALL LIST ARE INITIALIZED FOR CONTINUING THE
INTEGRATION. THE USER HAS ONLY TO DEFINE A NEW VALUE TOUT
AND CALL DE AGAIN.

DE CALLS TWO CODES, THE INTEGRATOR STEP AND THE INTERPOLATION
ROUTINE INTRP. STEP USES A MODIFIED DIVIDED DIFFERENCE FORM OF
THE ADAMS PECE FORMULAS AND LOCAL EXTRAPOLATION. IT ADJUSTS THE
ORDER AND STEP SIZE TO CONTROL THE LOCAL ERROR. NORMALLY EACH CALL
TO STEP ADVANCES THE SOLUTION ONE STEP IN THE DIRECTION OF TOUT.
FOR REASONS OF EFFICIENCY DE INTEGRATES BEYOND TOUT INTERNALLY,
THOUGH NEVER BEYOND T + 10*(TOUT-T), AND CALLS INTRP TO
INTERPOLATE THE SOLUTION AT TOUT. AN OPTION IS PROVIDED TO STOP
THE INTEGRATION AT TOUT BUT IT SHOULD BE USED ONLY IF IT IS
IMPOSSIBLE TO CONTINUE THE INTEGRATION BEYOND TOUT.

THIS CODE IS COMPLETELY EXPLAINED AND DOCUMENTED IN THE TEXT
COMPUTER SOLUTION OF ORDINARY DIFFERENTIAL EQUATIONS: THE INTIAL
VALUE PROBLEM BY L. F. SHAMPINE AND M. K. GORDON.

THE PARAMETERS FOR DE ARE:
F -- SUBROUTINE F(T,Y,YP) TO EVALUATE DERIVATIVES YP(I)=DY(I)/DT
NEQN -- NUMBER OF EQUATIONS TO BE INTEGRATED
Y(*) -- SOLUTION VECTOR AT T
T -- INDEPENDENT VARIABLE
TOUT -- POINT AT WHICH SOLUTION IS DESIRED
RELERR,ABSERR -- RELATIVE AND ABSOLUTE ERROR TOLERANCES FOR LOCAL
ERROR TEST. AT EACH STEP THE CODE REQUIRES
    DABS(LOCAL ERROR) .LE. DABS(Y)*RELERR + ABSERR
    FOR EACH COMPONENT OF THE LOCAL ERROR AND SOLUTION VECTORS
IFLAG -- INDICATES STATUS OF INTEGRATION

FIRST CALL TO DE --

THE USER MUST PROVIDE STORAGE IN HIS CALLING PROGRAM FOR THE ARRAY
IN THE CALL LIST Y(NEQN), DECLARE F IN AN EXTERNAL STATEMENT,
SUPPLY THE SUBROUTINE F(T,Y,YP) TO EVALUATE
DY(I)/DT = YP(I) = F(T,Y(1),Y(2),...,Y(NEQN))

AND INITIALIZE THE PARAMETERS:
NEQN -- NUMBER OF EQUATIONS TO BE INTEGRATED
Y(*) -- VECTOR OF INITIAL CONDITIONS
T -- STARTING POINT OF INTEGRATION
TOUT -- POINT AT WHICH SOLUTION IS DESIRED
RELERR,ABSERR -- RELATIVE AND ABSOLUTE LOCAL ERROR TOLERANCES
IFLAG -- +1,-1. INDICATOR TO INITIALIZE THE CODE. NORMAL INPUT
    IS +1. THE USER SHOULD SET IFLAG=-1 ONLY IF IT IS
    IMPOSSIBLE TO CONTINUE THE INTEGRATION BEYOND TOUT.
ALL PARAMETERS EXCEPT F, NEQN AND TOUT MAY BE ALTERED BY THE
CODE ON OUTPUT SO MUST BE VARIABLES IN THE CALLING PROGRAM.
OUTPUT FROM DE --

NEQN -- UNCHANGED

Y(*) -- SOLUTION AT T

T -- LAST POINT REACHED IN INTEGRATION. NORMAL RETURN HAS

T = TOUT.

TOUT -- UNCHANGED

RELERR, ABSERR -- NORMAL RETURN HAS TOLERANCES UNCHANGED. IFLAG=3

SIGNALS TOLERANCES INCREASED

IFLAG = 2 -- NORMAL RETURN. INTEGRATION REACHED TOUT

= 3 -- INTEGRATION DID NOT REACH TOUT BECAUSE ERROR

TOO SMALL. RELERR, ABSERR INCREASED

APPROPRIATELY FOR CONTINUING

= 4 -- INTEGRATION DID NOT REACH TOUT BECAUSE MORE THAN

MAXIMUM STEPS NEEDED

= 5 -- INTEGRATION DID NOT REACH TOUT BECAUSE EQUATIONS

APPEAR TO BE STIFF

= 6 -- INVALID INPUT PARAMETERS (FATAL ERROR)

THE VALUE OF IFLAG IS RETURNED NEGATIVE WHEN THE INPUT

VALUE IS NEGATIVE AND THE INTEGRATION DOES NOT REACH TOUT,

I.E., -3, -4, -5.

SUBSEQUENT CALLS TO DE --

SUBROUTINE DE RETURNS WITH ALL INFORMATION NEEDED TO CONTINUE

THE INTEGRATION. IF THE INTEGRATION REACHED TOUT, THE USER NEED

ONLY DEFINE A NEW TOUT AND CALL AGAIN. IF THE INTEGRATION DID NOT

REACH TOUT, AND THE USER WANTS TO CONTINUE, HE JUST CALLS AGAIN.

THE OUTPUT VALUE OF IFLAG IS THE APPROPRIATE INPUT VALUE FOR

SUBSEQUENT CALLS. THE ONLY SITUATION IN WHICH IT SHOULD BE ALTERED

IS TO STOP THE INTEGRATION INTERNALLY AT THE NEW TOUT, I.E.,

CHANGE OUTPUT IFLAG=2 TO INPUT IFLAG=-2. ERROR TOLERANCES MAY

BE CHANGED BY THE USER BEFORE CONTINUING. ALL OTHER PARAMETERS MUST

REMAIN UNCHANGED.