Lesson 13: Rapid Cooling and Euler-trapezoid

13.1 Applied Problem.

If an object is being cooled very rapidly, and the model is Newton's law of cooling, then the constant $c$ in $u_t = c(u_{\text{sur}} - u)$ will be large. This means the rate of change of the temperature will be large. Such systems are called stiff and require special numerical methods. The Matlab command odedemo illustrates some stiff differential equations and two stiff solvers called ode23s and ode15s.

13.2 Differential Equation Model.

Cooling via Newton's law is modeled by $u_t = c(u_{\text{sur}} - u)$ provided $u$ is not too large. For a glowing hot object heat is lost by radiation, and in this case a nonlinear model must be considered. Cooling via radiation can be modeled by a variation of Newton's law of cooling called Stefan's law of cooling $u_t = c(u_{\text{sur}}^4 - u^4)$. For large values of $u$ this will be a stiff differential equation.

13.3 Method of Solution.

In order to illustrate some basic numerical methods for solving stiff differential equations, we will consider three examples. The first example will be from Newton's law of cooling where the constant $c$ is very large. The second example will be from cooling via radiation where Stefan's law of cooling is used. The third example is for a system of two differential equations, which is illustrated in odedemo under the equation buiode in Matlab.
In a previous lesson we presented the cooling coffee cup problem where \( u_t = c(u_{\text{sur}} - u) \). Suppose \( c \) is about 25 times larger than the \( c \) in the coffee cup problem, say, \( c = 0.405 \). Euler's method is
\[
\frac{u^{k+1} - u^k}{h} = c(u_{\text{sur}} - u^k).
\]
Solve for \( u^{k+1} \) to get
\[
u^{k+1} = (1 - hc)u^k + husurc.
\]
If \( h = 5 \), then \( hc = 2.025 \), \( |1 - hc| > 1 \), and Euler's sequence will oscillate and blowup!

An alternative to Euler's method, that is not as sensitive to the choice of \( h \), is the Euler-trapezoid method where the right side is now evaluated at the next time step. For the Newton cooling problem this
\[
\frac{u^{k+1} - u^k}{h} = c(u_{\text{sur}} - u^{k+1}).
\]
Solve for \( u^{k+1} \) to get
\[
u^{k+1} = u^k/(1 + hc) + husurc/(1 + hc).
\]
As long as \( c \) is positive, the sequence will converge monotonically to the surrounding temperature.

The general Euler-trapezoid algorithm, which can compute the solution of stiff differential equations, is based on the trapezoid rule for numerical integration. If \( u(t) \) satisfies \( u_t = f(t,u) \), then
\[
\int_{kh}^{(k+1)h} u_t dt = u((k+1)h) - u(kh) = \int_{kh}^{(k+1)h} f(t,u(t))dt.
\]
The trapezoid rule discrete form of this is called the Euler-trapezoid algorithm and is
\[
u^{k+1} - u^k = \frac{h}{2}(f(kh,u^k) + f((k+1)h,u^{k+1})).
\]
Observe the right side where the unknown $u^{k+1}$ also appears. This is an additional complication relative to Euler's algorithm. One can use, at each time step, either Picard's or Newton's method to approximate this new value of $u$.

### 13.4 Matlab Implementation.

Cooling via radiation can be modeled by a variation of Newton's law of cooling called Stefan's law of cooling $u_t = c(u_{\text{sur}} - u^4)$. In the calculations below using the m-files `feuls.m` and `eults.m` we used $u_{\text{sur}} = 273$, $u(0) = 973$ and $c = .6971/(273^4)$. The Picard method, in the inner loop of `eults.m`, was used to solve the nonlinear subproblem at each time step. If only one iteration of the inner loop is done, then this is just the improved Euler method.

```matlab
function feuls = feuls(t,x)
    feuls = .6941*(1 - (x/273)^4);

% your name, your student number, lesson number
clear;
%
% Euler-Trapezoid Algorithm with Picard Solver.
%
eps = .0001;
maxm = 10;
u(1) = 973.;
T = 100;
KK = 40
h = T/KK;
t(1)= 0.;
%
% Begin Time Loop,
%
```
for k = 1:KK
    oldfeuls = feuls(t(k),u(k));
    oldu   = u(k) + h*oldfeuls;
    t(k+1) = t(k) + h;

% Begin Picard Loop.
% for m=1:maxm
    newu = u(k) + h*.5*(oldfeuls + feuls(t(k+1),oldu));
    if abs(newu-oldu)<eps
        break;
    end
    oldu = newu;
end
    u(k+1) = newu;
end
plot(t,u)
title('your name, your student number, lesson number')
xlabel('time')
ylabel('temperature')
13.5 Numerical Experiments.

The third example is a classic example of a very stiff system (see odedemo example buioode). The following system for two functions has known solutions $y_1(t) = \exp(-4t)$ and $y_2(t) = \exp(-t)$

$$y_1' = -10004 \, y_1 + 10000 \, y_2^4$$
$$y_2' = y_1 - y_2 - y_2^4.$$

The reader should use odedemo and try to solve this using both ode45 and ode23s. Note ode45 generates the solution, but it does do this very slowly.

As a first step in the study of this system, we consider the steady state solutions, which are defined by both derivatives being set equal to zero so that

$$0 = -10004 \, y_1 + 10000 \, y_2^4$$
$$0 = y_1 - y_2 - y_2^4.$$

The two solutions are given by $y_1^* = 0$ and $y_2^* = 0$. The Jacobian of $[f_1 \, f_2]'$ is

$$J = \begin{bmatrix} f_{y_1y_1} & f_{y_1y_2} \\ f_{y_2y_1} & f_{y_2y_2} \end{bmatrix} = \begin{bmatrix} -10004 & 40000(y_2^*)^4 \\ 1 & -1 - 4(y_2^*)^3 \end{bmatrix} = \begin{bmatrix} -10004 & 0 \\ 1 & -1 \end{bmatrix}.$$

The steady state solutions are stable because the Jacobian, the derivative matrix, evaluated at $(0,0)$ has negative real eigenvalues, -10004 and -1. Moreover, one of the eigenvalues is very large and so the time dependent solutions will have large rates of change and the system is stiff. The following calculations were done using ode23s and the m-files ypcstiff.m and cstiff.m. The command ode23s took 182 time steps and the command ode45 requires many more time steps.
function ypcstiff=ypcstiff(t,y)
    ypcstiff(1) = -10004*y(1) + 10000*y(2)^4;
    ypcstiff(2) = y(1) - y(2) - y(2)^4;
    ypcstiff = [ypcstiff(1) ypcstiff(2)]';

% your name , your student number, lesson number
clear;
tf = 3;
yo = [1 1];
[t y] = ode23s('ypcstiff',[0 tf],yo);
plot(t,y(:,1),t,y(:,2));
title('your name, your student number, lesson number')
xlabel('time')
ylabel('solutions one and two')
13.6 Additional Calculations.

Consider the classic stiff system

\[ y_1' = -10004 \, y_1 + 10000 \, y_2^4 \quad , \quad y_1(0) = 1 \]
\[ y_2' = y_1 - y_2 - y_2^4 \quad , \quad y_2(0) = 1. \]

(a). By-hand calculations verify \( y_1(t) = \exp(-4t) \) and \( y_2(t) = \exp(-t) \) are solutions.

(b). Use the final time equal to 4.0….modify the cstiff.m file.

(c). Execute the new cstiff.m file using ode45. Use the command whos to determined the number of time steps used

(d). Repeat the computations using ode15s and ode23s.

(e). Which method requires the fewest number of time steps?