# 10.34 Numerical Methods Applied to Chemical Engineering Fall 2015

## Homework #4: IVP

**Problem 1** (20 points)**.** In this problem you will develop and implement an ODE solver for problems of the form

$$\frac{d\mathbf{y}}{dt} = \mathbf{f}(\mathbf{y})$$

$$\mathbf{y}(t_0) = \mathbf{y}_0$$

Your function should be called in this way:

```
[tvec,Y] = my_ode_solver(@fname,[t0,tf],y0,options,params)
```

where fname.m is the name of your function which returns the vector $\mathbf{f}$, t0 is the initial time, tf is the final time, y0 is the initial values of the vector $\mathbf{y}$, params is a vector of constant numerical parameters, and options is a vector of options that control how your solver works. The outputs are a vector of time points and a matrix $\mathbf{Y}$ where $Y_{mn}$ is the computed value of $y_n(tvec(m))$. To make your program consistent with the built-in MATLAB$^{È}$ ODE solvers, your function should work with functions, fname.m, of this form

```
dydt = fname(t,y,params)
```

where dydt is the value of $\mathbf{f}(\mathbf{y})$, y, and params are vectors and t is a scalar.

The system is solved using time-stepping. Time-stepping methods use update formulas of the form:

$$\mathbf{y}(t+\Delta t) = \mathbf{y}(t) + \Delta t \frac{d\mathbf{y}}{dt}_{estimate} \tag{1}$$

my_ode_solver should use the 2nd-order Runge-Kutta (RK2) estimate if option(1) $> 0$,

$$\frac{d\mathbf{y}}{dt}_{estimate} = \mathbf{f}(\mathbf{y}(t) + \frac{1}{2}\Delta t \mathbf{f}(\mathbf{y}(t)))$$

and the Crank-Nicolson (CN, a.k.a. trapezoid rule) estimate if option(1)$< 0$,

$$\frac{d\mathbf{y}}{dt}_{estimate} = \frac{1}{2}(\mathbf{f}(\mathbf{y}(t)) + \mathbf{f}(\mathbf{y}(t+\Delta t)))$$

Note that if you plug the Crank-Nicolson estimate into eqn. 1, you will get an implicit equation in the unknown $\mathbf{y}(t+\Delta t)$, so you will need to solve a system of (usually nonlinear) equations to compute $\mathbf{y}(t+\Delta t)$. You can solve the system of equations using any MATLAB$^{È}$ routine (e.g. fsolve) or a solver of your own devising. We suggest you use the RK2 update formula to provide the initial guess.

Set the initial $\Delta t = |\text{option}(2)|$. If option(2)$> 0$, hold constant at this $\Delta t$. If option(2)$< 0$, implement adaptive time-stepping by comparing $\Delta\mathbf{y} = \mathbf{y}(t+\Delta t) - \mathbf{y}(t)$ computed two different ways. For both the RK2 and CN methods, we suggest you compare the RK2 and CN estimates of $\Delta\mathbf{y}$, using in either case the value for $\mathbf{y}(t+\Delta t)$ you just computed. If you find that

$$\left\| \left[\Delta t\mathbf{f}\big(\mathbf{y}(t) + \frac{1}{2}\Delta t\mathbf{f}(\mathbf{y}(t))\big)\right] - \left[\frac{1}{2}\Delta t\big(\mathbf{f}(\mathbf{y}(t)) + \mathbf{f}(\mathbf{y}(t+\Delta t))\big)\right] \right\| > atol + rtol\|\mathbf{y}(t)\|$$

then $\Delta t$ is unacceptably large and should be reduced in size (say cut in half) and the calculation repeated; otherwise accept the value of $\mathbf{y}(t+\Delta t)$ and increase $\Delta t$ by a user-defined factor of option(3) for the next step. Set the error tolerances from user input, atol=option(4) and rtol=option(5). For this problem use rtol=1e-8 and atol=1e-8. You may need to add a protection against $\Delta t$ becoming so tiny that $t + \Delta t$ and $t$ are indistinguishable by the computer, otherwise the program may get caught in an infinite loop.

Note that the user usually wants to know the values exactly at tf, so adjust the last $\Delta t$ to make the final t = tf.

1. Implement `my_ode_solver` as described above. Test your ODE solver on the ODE-IVP system considered in HW0, Problem 1. You can do this by slightly modifying the posted solution so that it calls your function `my_ode_solver`. Test the RK2 and Crank-Nicolson methods, each with and without adaptive time-stepping for initialize steps size of $10^{-4}$ and $10^{-2}$. Generate plots for the same situation modeled in the posted solution. You have been provided code to help create plots in `ODE_plotter`. Do all your solutions look OK? Note that in the posted solution the tolerances in ode23 were left at their default values, which were too loose. Hence, the adaptive time-stepping was not effective. Tighten the tolerances and see how the number of time steps computed by ode23 increases. Report the number of steps for ode23 at default and improved tolerances as well as the number of steps each of your four methods take for each of the two initial step sizes.

2. There are some hidden issues with the posted solution. We expect dT/dr=0 at r=0, otherwise there will be a non-differentiable cusp in T(r) at r=0. Also, physically we expect T to be highest at the center and to monotonically decrease with radius. Hence we expect $u_2(y = R) = 0$ and $u_2 < 0$ for $0 < y < R$. Do any of the solutions (from ode23, RK2, or CN) satisfy this physicality condition?

3. The posted solution ends at $y = R - 0.01 = 0.19m$, i.e. at $r = 1cm$. Use ode23 and your ODE-IVP solver to solve all the way to $y = R - 0.001 = 0.199m$, i.e. $r = 1mm$. Does the computed $u_2 = \frac{dT}{dr}$ look reasonable at small values of $r$? Does the implicit Crank-Nicolson method work any better than the explicit RK2 method for this problem? Does adaptive time-stepping help?

**Problem 2** (10 points)**.** Another issue for the solution to problem 1 for HW0 is that all of the parameters in the problem are only known to a finite number of significant figures. In fact, qdot is only given to one significant figure! The question is how sensitive our computed T and dT/dr are to these uncertainties. A popular way to compute the sensitivities, $\frac{\partial y_i}{\partial p_m}$, is to differentiate the equation dy/dt = f(y; p) with respect to each of the M uncertain parameter values.
To perform this analysis, we consider a larger ODE problem. For our system, we define

$$\tilde{y}^{\mathrm{T}} = \begin{bmatrix} y_1 & y_2 & \frac{\partial y_1}{\partial p_1} & \frac{\partial y_1}{\partial p_2} & \frac{\partial y_2}{\partial p_1} & \frac{\partial y_2}{\partial p_2} \end{bmatrix}$$

Here we have two state variables and two parameters but this definition could be extended in a straightforward manner for a larger system. Note that $\tilde{y}$ is of length $N + NM$ where N is the number of state variables and M is the number of parameters. We want to consider the system $\frac{d\tilde{y}}{dt}$.

$$\frac{d\tilde{y}_k}{dt} = f_k(t, y; p) \quad \text{for} \quad k = 1, \ldots, N$$

$$\frac{d\tilde{y}_k}{dt} = \frac{d\frac{\partial y_i}{\partial p_m}}{dt} = \frac{\partial f_i}{\partial p_m} + \sum_{j=1}^{N} \frac{df_i}{dy_j}\frac{\partial y_j}{\partial p_m} \quad \text{for} \quad k = (N+1), \ldots, (N+2M)$$

The initial conditions are specified as

$$\tilde{y}_i(t=0) = y_{i,0} \quad \text{for} \quad i = 1, \ldots, N$$

These $N \times M$ equations give the intitial conditions for $\tilde{y}_k$ for $k = (N+1), \ldots, (N+NM)$,

$$\left.\frac{\partial y_i}{\partial p_m}\right|_{t=0} = \frac{\partial y_{i,0}}{\partial p_m}$$

Write a MATLAB function that computes the sensitivity of the computed u(y) to these two uncertain parameters: surface temperature (393 K), known to about$\mathit{\Delta} p_1 = 1K$ and qdot ($20000W/m^3$) known to within about $\mathit{\Delta} p_2 = 5000W/m^3$. Estimate the uncertainty in the predicted values of $T$ and $\frac{dT}{dr}$ at $r = 1cm$ using this common approximation:

$$\delta(u_1) \sim \left[\left(\frac{\partial u_1}{\partial p_1}\mathit{\Delta} p_1\right)^2 + \left(\frac{\partial u_1}{\partial p_2}\mathit{\Delta} p_2\right)^2\right]^{\frac{1}{2}}$$

$$\delta(u_2) \sim \left[\left(\frac{\partial u_2}{\partial p_1}\mathit{\Delta} p_1\right)^2 + \left(\frac{\partial u_2}{\partial p_2}\mathit{\Delta} p_2\right)^2\right]^{\frac{1}{2}}$$

Describe the system that is being solved and report its initial conditions. You have been provided code that calculates the finite difference approximation for the Jacobian (`jacobian_solve`). Note: If one needs to compute a large number of sensitivities (e.g. if your model includes many uncertain parameters and many differential equations), there are much more efficient methods for computing the sensitivities, which take advantage of the fact that the ODEs for the sensitivities are partially decoupled from each other and from the original ODEs. This is particularly important if the ODE system is stiff.

**Problem 3** (20 points)**.** Epoxides are used on large scale, for example to make polyethers which can be used in medical implants without being rejected by the immune system, or to make durable polyurethane materials (e.g. skateboard wheels, transparent coatings for hardwood floors). Epoxides are made by catalytic oxidizing alkenes using $O_2$. Despite a lot of work (including by this year's Hottel Lecturer A. Corma) the catalysts available are not ideal. Your task is to figure out the optimal energetics for a homogeneous catalyst X which works by this reaction sequence:

$$\begin{aligned}
&\text{XOO} + \text{alkene} \rightarrow \text{epoxide} + \text{XO} && k_1 = 10^6 \frac{L}{mol\ s} exp(-E_{a1}/RT) \\
&\text{X} + \text{O}_2 \rightarrow \text{XOO} && k_2 = 10^9 \frac{L}{mol\ s} exp(-E_{a2}/RT) \\
&\text{XOO} \rightarrow \text{X} + \text{O}_2 && k_{2,rev} = \frac{10^{14}}{s} exp(-E_{a2,rev}/RT) \\
&\text{XO} + \text{XO} \rightarrow \text{X} + \text{XOO} && k_3 = 10^8 \frac{L}{mol\ s} exp(-E_{a3}/RT) \\
&\text{X} + \text{XOO} \rightarrow \text{XO} + \text{XO} && k_{3,rev} = 10^8 \frac{L}{mol\ s} exp(-E_{a3,rev}/RT)
\end{aligned}$$

Assume $H_f(X) = 0$ and this relationship between each $E_a$ and the corresponding $\Delta H_{rxn}$.

$$E_{a,i} = \frac{1}{2}\Delta H_{rxn,i} + \sqrt{Q_i^2 + (\frac{1}{2}\Delta H_{rxn,i})^2}$$

where

$$\begin{aligned}
Q_1 &= 40kJ/mol & \Delta H_{rxn,1} &= H_f(XO) - H_f(XOO) - 105kJ/mol \\
Q_2 = Q_{2,rev} &= 5kJ/mol & \Delta H_{rxn,2} &= H_f(XOO) = -\Delta H_{rxn,2rev} \\
Q_3 = Q_{3,rev} &= 32kJ/mol & \Delta H_{rxn,3} &= H_f(XOO) - 2H_f(XO) = -\Delta H_{rxn,3rev}
\end{aligned}$$

The objective is to maximize the amount of epoxide formed in a plug-flow-reactor (PFR) with a diameter of $10mm$ and length of $15m$. Recall the material balance equation for a PFR:

$$\frac{dF_i}{dz} = A \sum_n S_{i,n} r_n$$

where $F_i(z)$ is the moles of $i$ fed per second passing through a cross-section at position $z$, $A$ is the cross-sectional area, $S$ are the stoichiometric coefficients and $r$ are the reaction rates.

The feed composition is 66 mole % alkene, 33 mole % $O_2$, 1 mole % catalyst, all in gas phase. The total feed flow rate is 2.5 moles/second. P is 10 bar and pressure drop is negligible. Assume the system is isothermal and you can vary the T, any T < 700 K is acceptable. We will also apply a constraint that values of $H_f(XO)$ and $H_f(XOO)$ be greater than $-300kJ/mol$.

Hint: consider what values of $H_f(XO)$ and $H_f(XOO)$ will yield exothermic reactions in the direction of the catalytic cycle that results in production of epoxide.

1. Write a function that returns the reaction rates $r_n$ based on the flow rates and the values of $T$, $H_f(XO)$, and $H_f(XOO)$. Check to ensure that the units work out. Test your function on the initial conditions for the nominal case where $T = 650K$, $H_f(XO) = -30kJ/mol$, and $H_f(XOO) = -80kJ/mol$. Show the vector of activation energies used. Describe what your function does. For vectors, use the reaction order $r_1$, $r_{2,fwd}$, $r_{2,rev}$, $r_{3,fwd}$, and $r_{3,rev}$.

2. Write the stoichiometry matrix $S$. Use the reaction order described above and for the components, order them in the following manner: X, $O_2$, alkene, epoxide, XO, XOO. Write a function that outputs the yield of epoxide $\frac{F_{epoxide}(z=15m)}{F_{alkene}(z=0)}$ given values of the parameters $T$, $H_f(XO)$, and $H_f(XOO)$. Use `ode15s` to solve the ODE. Describe what your function does.

3. Use MATLAB's built in constrained minimizer `fmincon` to find the optimal $T$, $H_f(XO)$, and $H_f(XOO)$ given the constraints in the problem. Write out the method you use to search through the space and ensure you are not caught in a local minimum that may not be the global minimum. To get good convergence, you may need to play with the tolerances for the ODE solver and also `fmincon` parameters. Parameters that may need to be tweaked include tolerances, algorithms, and finite difference calculation parameters. Describe how you do so in detail.

4. Would using the method described in problem 2 applied to this problem have required more or less function evaluations per iteration? Why or why not?