

University of Delaware Department of Mathematical Sciences  
Math 428 Algorithmic and Numerical Solution of Differential Equations 08S  
R. J. Braun

**Project 2: FitzHugh-Nagumo Equations**  
Due date: 5/1/08, 3:30 pm.

**YOUR NAME:** \_\_\_\_\_

In this project you will use MATLAB to numerically integrate the FitzHugh-Nagumo equations:

$$\begin{aligned}\frac{dv_1}{dt} &= -v_1(v_1 - 1)(v_1 - a) - v_2 + I \\ \frac{dv_2}{dt} &= \epsilon(v_1 - \gamma v_2).\end{aligned}$$

They are a model of neuron function; in particular, the model the repeated firing of a neuron is imitated by these equations. The emphasis in this project is to discover and understand some behavior of the solutions.

You will use three approaches: (i) RK4, (ii) `ode45` in Matlab (a 4(5) ERK method) and (iii) a variable order and time step BDF method (`ode15s` in Matlab).

1. Develop a function file that has the FitzHugh-Nagumo equations in it. I recommend putting the parameters  $a$ ,  $I$ ,  $\epsilon$ , and  $\gamma$  in the function file and changing them there, though you need not do this. Write function or script files that use `rk4`, `ode45` and `ode15s`; they must be able to set both the relative and absolute errors. Note that the output from the function that specifies the ode must be a column vector for Matlab functions but that is must be a row vector for `rk4.m`; you can write two separate functions for the ode or modify `rk4.m` to output a column vector to remedy this.
2. To test the scripts for `ode45` and `ode15s`, try them with the parameters

$$a = 0.1, \epsilon = 0.008, I = 0.0568384, \gamma = 1,$$

for the following sets of tolerances: (Relative error, Absolute Error) =  $(2 \times 10^{-4}, 10^{-5})$ ,  $(2 \times 10^{-5}, 10^{-6})$ ,  $(2 \times 10^{-7}, 10^{-8})$ ,  $(10^{-8}, 10^{-9})$ . Use the initial conditions  $(v_1, v_2) = (0.5, 0.1)$ . You should observe that the two methods agree for the last pair of tolerances only, and that, for this problem, `ode45` approaches this common behavior for larger values of the tolerances than `ode15s` does. Use phase plane plots, that is, plot the solution in the  $(v_1, v_2)$  plane, for the last two tolerance cases to show that the behavior of the solution converges. Also, for these two cases, plot  $v_i(t)$  vs.  $t$  on the same plot. (For this item, there are four plots total).

3. To test the RK4 method, increase or decrease the number of time steps  $N$  to show that the method converges to the same answer as above. How large must  $N$  be, or alternatively, how small must  $h$  be to ensure that the results are close to the results from the Matlab functions? Use a phase plane plot or two to demonstrate this convergence.

4. From here on, you only need to use one of the built-in Matlab functions. Using the smallest tolerance values from above, compute solutions to the problem for the parameter values from above for the following values of  $I$ :

$$I = 0.05527, 0.05678, 0.05683, 0.0568384, 0.0568385, 0.05740.$$

Describe how the solutions change with  $I$ ; use a few plots of  $v_i(t)$  and phase plane plots to help in this description. Hint: the amplitude and period of the oscillations are varying with  $I$ ; how do they vary?

5. We can illustrate the change of amplitude with  $I$  in the following way. The amplitude of the steady-state oscillations can be found by integrating for a long enough time and then finding the peak value in a period of  $v_1$ . Do this calculation for more values of  $I$  than are given above, then plot the resulting amplitudes as a function of  $I$ . This picture shows a dramatic change of amplitude in a small range of  $I$ , then levels off. Note that for  $I$  at the lowest end of the range, the amplitude may be 0.

One way to get the amplitude is to use Matlab's `max` function over a range of  $v_1$  near the end of the calculation. For example, say that the solution `V` in Matlab is stored in an  $n \times 2$  matrix and that the first column is  $v_1$  and the second column is  $v_2$ . To see the maximum value of  $v_1$ , we could store the last 10% of the values of  $v_1$  in a vector `VV` and then find the max value of `VV`, as follows:

```
VV = V(round(0.9*end):end,1); % capture vector v_1 at end
vmax = max(VV); % find maximum value
```

Note that this is done because the kind of output one gets from `max` depends on the input, and this is a straightforward way to get what we want. We must have at least one full oscillation of the solution in the window at the end for this method to work.

Advice: These solutions are somewhat sensitive to the parameters; *don't skimp on digits*.

Turn in a report that uses this page as a cover sheet for your project. The report should have sections entitled Introduction, Methods, Results, and Conclusion; subsections are possible. Describe briefly the three numerical methods used; do a good job of describing the results. You don't have to do any error analysis here, but you are justifying the results by getting different methods to agree.

Though you may talk with each other about how to do the project, the project is individual in that sharing of code, data and text from your report is not permitted.