Proposed Procedures: Continuous Analysis of a Model for T Cells in Autoimmune Diabetes

David Li

Materials

Estimated Total Cost.....$0

1. Computer (x86-64 architecture running a Linux distribution; no need to purchase).....$0
   For reference, the computer used in this experiment has the following specifications:
   - Processor: Intel Core i5-3230M
   - Memory: 4GB
   - Linux distribution: Arch Linux
2. Computer Software.....$0
   - Python 3.3.2 with `mpmath` (arbitrary-precision arithmetic) (http://www.python.org, http://www.mpmath.org)
   - XPP AUTO 7.0 (bifurcation analysis) (http://www.math.pitt.edu/~bard/xpp/xpp.html)
   - ConTEXt 2013.05.28 (typesetting) (http://wiki.contextgarden.net)
   - Gnuplot (plotting)

Procedures

Note: familiarity with the Linux command line is assumed, though the software used should run on Windows. Also note: `diagrams.py`*,* `time_diagrams.tex`*, and* `bifurcation_diagrams.tex` *do not yet exist. Further automation scripts may eventually be created, thus the procedures may differ.*

1. Download the necessary experiment software and experiment files (location TBD).
2. Extract the experiment files to a directory.
3. Open a shell and `cd` to the directory containing the experiment files.
4. To generate plots of the system over time:
   a. Run `python rk4_ode.py` to generate the data via the Runge-Kutta method.
   b. Run `../path_to_xpp/xppaut -silent mk.ode` to generate the data via XPP AUTO.
   c. Run `python diagrams.py time` to generate the TEX plot files.
   d. Run `context time_diagrams.tex` to generate a PDF containing the plots.
5. Then, to generate the plots of the system vs the bifurcation parameter (the peptide clearance rate, δ_p):
   a. Run `../path_to_xpp/xppaut -runnow mk_static.ode` to launch XPP AUTO.
      i. Press `f a` to bring up AUTO.
      ii. Press `f l` and choose `mk_static.ode.auto` to load the parameters for the bifurcation diagram.
      iii. Press `r s` to generate the diagram.
      iv. Press `g <TAB> <ENTER> r p` to generate the branches for the first bifurcation.
      v. Press `g <TAB> <TAB> <ENTER> r p` to generate the branches for the second bifurcation.
      vi. Press `f w` and save the points in `mk_static.dat`.
      vii. Close XPP AUTO (`Control-C` the program in the terminal).
   b. Run `python diagrams.py bifurcation` to generate the TEX plot files.
   c. Run `context bifurcation_diagrams.tex` to generate a PDF containing the plots.

Parameter Values (for the programs, not the system of ODEs):

Parameter	**Value**
Integration time (the time the system will be run for)	200 days
Integration step (RK4)[1] (the time interval used)	0.01 days
Integration step (XPP AUTO)[1]	0.05 days
AUTO Par 1	`a15`
AUTO Hi-Lo Y-axis	`A`
AUTO Hi-Lo Main Parameter	`a15`
AUTO Hi-Lo X range	$[0, 18]$
AUTO Hi-Lo Y range	$[0, 3]$
AUTO Par Max[2]	18

Initial Conditions

Initial Conditions[3]	**Value of A**	**Value of M**	**Value of E**	**Value of a_{15}**
"Typical" (diseased) state, used for the static bifurcation diagram	0.5	0	1	1
Healthy state #1	0	0	1	1
Healthy state #2	0	0.5	1	0

Parameter Ranges (for the peptide clearance rate δ_p)

As the total time is 200 days, the parameter will be varied at a rate of $\frac{\delta_{p2} - \delta_{p1}}{200}$ per day.

- 0 to 4
- 0 to 18
- 3 to 1.5

[1] RK4 and XPP AUTO use different integration algorithms, thus a different step size should not matter.

[2] All parameters for AUTO not listed were left at their default values.

[3] B was left at 1; all parameters had their default scaled values.

Appendix A. Program Files

XPP AUTO ODE file (based on [1]). This file sets up the system of equations

$$p = \frac{a_{14}}{a_{15}} EB \qquad f_1 = \frac{p^n}{k_1^n + p^n} \qquad f_2 = \frac{a k_2^m}{k_2^m + p^m}$$

$$\frac{dA}{dt} = (a_6 + a_7 M) f_1(p) - a_8 A - a_9 A^2$$

$$\frac{dM}{dt} = a_{10} f_2(p) A - f_1(p) a_7 a_{16} M - a_{11} M$$

$$\frac{dE}{dt} = a_{12}(1 - f_2(p)) A - a_{13} E$$

$$\frac{dB}{dt} = -a_{17} EB$$

```
p = a14*E*B/a15
f1 = p^a1/(a2^a1+p^a1)
f2 = a4*a5^a3/(a5^a3+p^a3)
A' = f1*(a6+a7*M)-a8*A-a9*A^2
M'= a10*f2*A-f1*a16*a7*M-a11*M
E' = a12*(1-f2)*A-a13*E
# Disease-free initial conditions
# init A=0.5,M=0,E=1
# Diseased initial conditions
# init A=0.5,M=0.0,E=1

# Continuously decreasing analysis
# p is in the range 0-10 so a15 can be
# in the range 1 to infinity (p=0)
par B=1
a15'=<RATE OF CHANGE OF PARAMETER>
init a15=<INITIAL PARAMETER VALUE>
par a1=2,a2=2,a3=3,a4=0.7,a5=1,a6=0.02
par a7=20,a8=1.0,a9=1.0,a10=1,a11=0.01,a12=0.1
par a13=0.3,a14=50,a16=0.1,a17=0.14
@ dt=<TIME STEP>
@ total=<FINAL TIME>
@ xlo=0,xhi=200,ylo=0,yhi=4
@ NPLOT=4, XP=t, YP=A, XP2=t, YP2=M, XP3=t, YP3=E
done
```

Python 4th order Runge-Kutta solver for a system of ODEs (based on [2]).

```
from mpmath import mpf, mp

def vectorize(functions):
    def _vectorized(x, ys):
        return tuple(f(x, *ys) for f in functions)
    return _vectorized
def inc_vec(vec, inc):
    return [x + inc for x in vec]
def add_vec(*vecs):
    return [sum(nums) for nums in zip(*vecs)]
def mul_vec(vec, mul):
    return [x * mul for x in vec]
def rk4_system(y, x0, y0, h, steps):
    """
    y: list of ODE
    y0: vector of initial ys (tuple)
    Finds y(x0 + h * steps)
    """
```

```
    x1 = x0
    y1 = list(y0)
    xs = [x0]
    ys = [[y0[i]] for i,f in enumerate(y)]
    f = vectorize(y)
    for i in range(steps):
        k1 = f(x1, y1)
        k2 = f(x1 + (h / 2), add_vec(y1, mul_vec(k1, h / 2)))
        k3 = f(x1 + (h / 2), add_vec(y1, mul_vec(k2, h / 2)))
        k4 = f(x1 + h, add_vec(y1, mul_vec(k3, h)))
        y1 = add_vec(y1, mul_vec(add_vec(k1, mul_vec(k2, 2),
                     mul_vec(k3, 2), k4), h / 6))
        for i, val in enumerate(y1):
            ys[i].append(val)
        xs.append(x1)
        x1 += h
    return xs, ys

if __name__ == '__main__':
    # PARAMETER DEFINITIONS OMITTED (SEE PAPER FOR VALUES)
    f_1 = lambda p: p**a1 / (a2**a1 + p**a1)
    f_2 = lambda p: a4 * a5**a3 / (a5**a3 + p**a3)
    p = lambda t, E, B, a15: a14 * E * B / a15
    y = (
        lambda t,A,M,E,B,a15:(a6+a7*M)*f_1(p(t,E,B,15))-a8*A-a9*A**2,#dA/dt
        lambda t,A,M,E,B,a15:a10*f_2(p(t,E,B,a15))*A-f_1(p(t,E,B,a15))*a7*a16*M-a11*M,#dM/dt
        lambda t,A,M,E,B,a15:a12*(1-f_2(p(t,E,B,a15)))*A-a13*E,#dE/dt
        lambda t,A,M,E,B,a15:-a17*E*B,#dB/dt
        lambda t,A,M,E,B,a15:0.1#da15/dt
    )
    result = rk4_system(y, mpf('0'), (<INITIAL CONDITIONS>), <TIME STEP>, <STEPS>)

    import pickle
    pickle.dump(result, open('rk4_ode.pickle', 'wb'))
```

Appendix B. Bibliography

[1]: Mahaffy, Joseph M. & Edelstein-Keshet, L. (2007). Modeling Cyclic Waves of Circulating T Cells in Autoimmune Diabetes. *SIAM Journal on Applied Mathematics, 67*.

[2]: Gonsalves, R. J. (2009). Runge-Kutta Methods for ODE Systems. *Computational Physics*. Retrieved on October 24, 2013 from http://www.physics.buffalo.edu/phy410-505-2009/topic3/lec-3-2.pdf.