Landes Bioscience

www.landesbioscience.com

Supplemental Material to:

Pullen N, Jaeger KE, Wigge PA, Morris RJ

Simple network motifs can capture key characteristics of the floral transition in *Arabidopsis*

Plant Signaling & Behavior 2013; 8(11)
http://dx.doi.org/10.4161/psb.26149

www.landesbioscience.com/journals/psb/article/26149

Supplemental Information

S1 Introduction

We describe the details of an IPython notebook implementation (Fernando Pérez, Brian E. Granger, IPython: A System for Interactive Scientific Computing, Computing in Science and Engineering, vol. 9, no. 3, pp. 21-29, May/June 2007, doi:10.1109/MCSE.2007.53. URL: http://ipython.org) for the simple flowering time models mentioned in the main paper. Details on the underlying network motifs can be found in http://www.pnas.org/content/100/21/11980.full.pdf+html and "An Introduction to Systems Biology: Design Principles of Biological Circuits" by Uri Alon. The notebook can be started with

```
ipython notebook --pylab=inline
```

to get inline plots. The following common Python libraries are used: NumPy, SciPy, Matplotlib (versions shown at the end).

```
# import necessary libraries
import matplotlib.pyplot as plt #for plotting
import numpy as np
from scipy.integrate import odeint # routine to solve the ODEs
```

```
# Preferred figure format to get PDF images in LaTeX documents
%config InlineBackend.figure_format = 'svg'
```

S1.1 Input signal

The FT input signal to the system in this work was modelled as a binary function. We have used a long signal but also a small blip in these examples which might represent, for example, a one-off short exposure to sunlight.

```
blipStart = 3
blipEnd = 3.5
signalStart = 10.0
signalEnd = 15.0

def FTsignal(t):
   if (t>=signalStart) and (t<signalEnd):
      return 1.0
   elif (t>=blipStart) and (t<blipEnd):
      return 1.0
   else:
      return 0.0
```

S2 Coherent feed-forward loop

We write $\theta_{FT.LFY}(FT)$ to mean that when FT crosses the binding threshold it binds to the promoter site of LFY and thus activates LFY transcription. Similarly $\theta_{FT.AP1}(FT)$ means AP1 is activated when FT crosses the threshold. In the code the threshold for the activation of LFY and AP1 is set to $FT = 1$. $\theta_{LFY.AP1}(LFY)$ means that when LFY passes the binding threshold it binds the AP1 promoter and thus activates AP1 transcription. In the code below k_1 = the LFY.AP1 threshold value, which is set to 0.5. The activation constants, ν, and degradation constants, δ, were set to 1. The equations for this system, which exhibits noise filtering, are as follows:

$$\frac{dLFY}{dt} = \nu_{LFY}\theta_{FT.LFY}(FT) - \delta_{LFY}LFY$$

$$\frac{dAP1}{dt} = \nu_{AP1}\theta_{FT.AP1}(FT)\theta_{LFY.AP1}\left(LFY\right) - \delta_{AP1}AP1$$

```
deltaLFY = 1.0
deltaAP1 = 1.0
nuLFY = 1.0
nuAP1 = 1.0
k1 = 0.5

def coffl(y, t):
   LFYi = y[0]
   AP1i = y[1]
   # coherent FFL equations
   LFYdot = nuLFY*FTsignal(t) - deltaLFY*LFYi
   if LFYi>k1:
      AP1dot = nuAP1*FTsignal(t)*1.0 - deltaAP1*AP1i
   else:
         AP1dot = - deltaAP1*AP1i

   return [LFYdot, AP1dot]
```

Next we set up the initial conditions, time grid and then solve the ODEs using SciPy's odeint function.

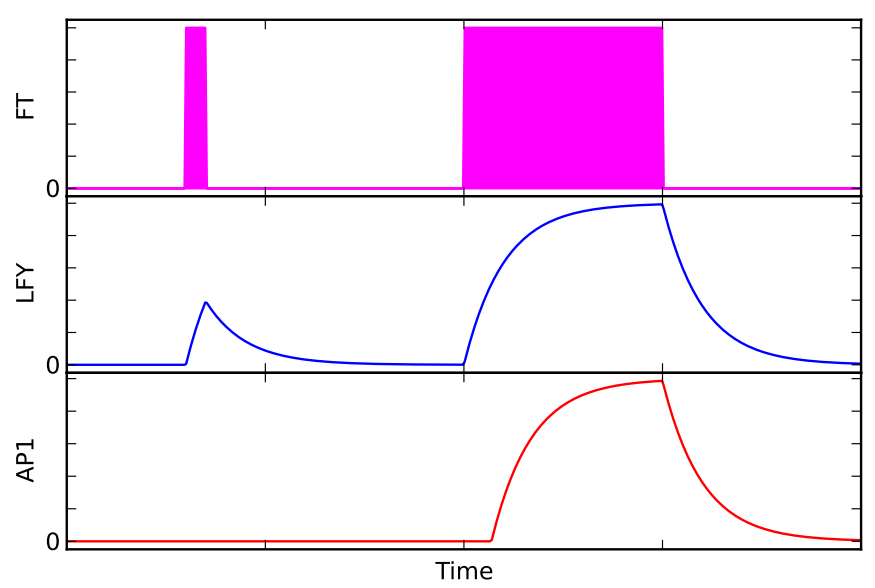
```
# initial conditions
LFY0 = 0.0 #
AP10 = 0.0 #
y0 = [LFY0, AP10] # initial condition vector
t = np.linspace(0, 20.0, 500) # time grid

# solve the DEs
soln = odeint(coffl, y0, t, hmax = 0.1) #need hmax as can get wrong results o/wise - I
    think to do with adaptive step sizes in numerical method missing important points
```

```
LFY = soln[:, 0]
AP1 = soln[:, 1]
#for i in range(len(AP1)): print t[i],stepfn(t[i]),LFY[i],AP1[i]
```

Below we plot the FT signal, and output LFY and AP1.

```
f, (ax1, ax2, ax3) = plt.subplots(3, sharex=True, sharey=False)
# Fine-tune figure; make subplots close to each other and hide x ticks for
# all plots.
f.subplots_adjust(hspace=0)
plt.setp([a.get_xticklabels() for a in f.axes[:]], visible=False)

steps = np.zeros(len(t))
for i in range(len(t)): steps[i]=FTsignal(t[i])
ax1.fill_between(t, 0, steps,color='magenta')
ax1.margins(0, 0.05)
ax1.set_ylabel("FT")
ax1.set_yticklabels(['','0'])

ax2.plot(t, LFY, label='LFY',color='blue')
ax2.margins(0, 0.05)
ax2.set_ylabel("LFY")
ax2.set_yticklabels(['','0'])

ax3.plot(t, AP1, label='AP1',color='red')
ax3.margins(0, 0.05)
ax3.set_xlabel("Time")
ax3.set_ylabel("AP1")
ax3.set_yticklabels(['','0'])
```

```
[<matplotlib.text.Text at 0x4341950>, <matplotlib.text.Text at 0x434c510>]
```



S3 Regulated feed-forward loop

The notation is identical to that for the coherent feed-forward loop. $\theta_{AP1.LFY}(AP1)$ means that when AP1 passes the binding threshold it binds to the LFY promoter and thus activates LFY transcription. In the code below k_2 = the LFY.AP1 and the AP1.LFY threshold values, which are both set to 0.45. The equations are:

$$\frac{dLFY}{dt} = \nu_{LFY} \max\left(\theta_{FT.LFY}(FT), \theta_{AP1.LFY}(AP1)\right) - \delta_{LFY} LFY$$
$$\frac{dAP1}{dt} = \nu_{AP1} \max\left(\theta_{FT.AP1}(FT), \theta_{LFY.AP1}(LFY)\right) - \delta_{AP1} AP1$$

The regulated feed-forward loop uses OR logic rather than AND logic and hence this system can show memory.

```
deltaLFY = 1.0
deltaAP1 = 1.0
nuLFY = 1.0
nuAP1 = 1.0
k2 = 0.45

def regffl(y, t):
   LFYi = y[0]
   AP1i = y[1]
   # the regulated FFL model equations
   if (FTsignal(t)==1) or (AP1i>k2):
      LFYdot = nuLFY*(1) - deltaLFY*LFYi
```

```
    else:
        LFYdot = -deltaLFY*LFYi
    if (FTsignal(t)==1) or (LFYi>k2):
        AP1dot = nuAP1*(1) - deltaAP1*AP1i
    else:
        AP1dot = -deltaAP1*AP1i

    return [LFYdot, AP1dot]
```

Below we solve the equations as before . . .

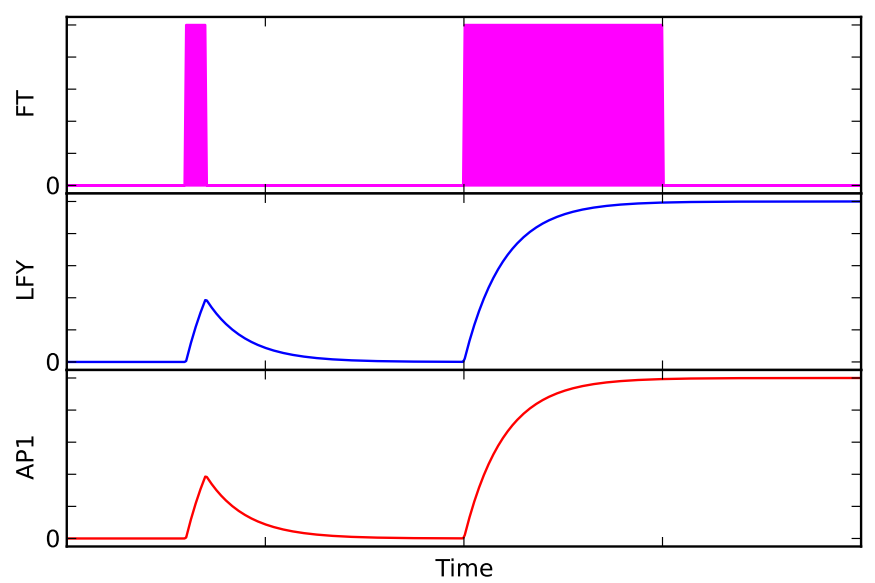
```
# initial conditions
LFY0 = 0.0 #
AP10 = 0.0 #
y0 = [LFY0, AP10] # initial condition vector
t = np.linspace(0, 20.0, 500) # time grid

# solve the DEs
soln = odeint(regffl, y0, t, hmax = 0.1) #need hmax as can get wrong results o/wise - to
     do with adaptive step sizes in numerical method missing important points
LFY = soln[:, 0]
AP1 = soln[:, 1]
#for i in range(len(AP1)): print t[i],stepfn(t[i]),LFY[i],AP1[i]
```

. . . and produce the plots again.

```
f, (ax1, ax2, ax3) = plt.subplots(3, sharex=True, sharey=False)
# Fine-tune figure; make subplots close to each other and hide x ticks for
# all plots.
f.subplots_adjust(hspace=0)
plt.setp([a.get_xticklabels() for a in f.axes[:]], visible=False)

steps = np.zeros(len(t))
for i in range(len(t)): steps[i]=FTsignal(t[i])
ax1.fill_between(t, 0, steps,color='magenta')
ax1.margins(0, 0.05)
ax1.set_ylabel("FT")
ax1.set_yticklabels(['','0'])

ax2.plot(t, LFY, label='LFY',color='blue')
ax2.margins(0, 0.05)
ax2.set_ylabel("LFY")
ax2.set_yticklabels(['','0'])

ax3.plot(t, AP1, label='AP1',color='red')
```

```
ax3.margins(0, 0.05)
ax3.set_xlabel("Time")
ax3.set_ylabel("AP1")
ax3.set_yticklabels(['','0'])
```

```
[<matplotlib.text.Text at 0x481a210>, <matplotlib.text.Text at 0x4821250>]
```



S4 Compromise feed-forward loop

The notation is identical to that previously. In the code below k_3 = the LFY.AP1 and the AP1.LFY threshold values, which are set to 0.45 as before. We use two different levels of activation depending on the number of activators bound. The higher levels, $\nu_{LFY,1}$ and $\nu_{AP1,1}$, are set to 1, and the lower levels, $\nu_{LFY,2}$ and $\nu_{AP1,2}$, are set to 0.5. Because there are both AND and OR logic gates, we get some noise filtering and sufficient memory for the system to continue to flower.

The equations are:

$$\frac{dLFY}{dt} = \begin{cases} \nu_{LFY,1} - \delta_{LFY} LFY & \text{if } \theta_{FT.LFY}\left(FT\right) = 1 \text{ and } \\ & \theta_{AP1.LFY}\left(AP1\right) = 1 \\ \nu_{LFY,2} \max\left(\theta_{FT.LFY}\left(FT\right), \theta_{AP1.LFY}\left(AP1\right)\right) & \\ \qquad - \delta_{LFY} LFY & \text{otherwise} \end{cases}$$

$$\frac{dAP1}{dt} = \begin{cases} \nu_{AP1,1} - \delta_{AP1} AP1 & \text{if } \theta_{FT.AP1}\left(FT\right) = 1 \text{ and } \theta_{LFY.AP1}\left(LFY\right) = 1 \\ \nu_{AP1,2} \max\left(\theta_{FT.AP1}\left(FT\right), \theta_{LFY.AP1}\left(LFY\right)\right) - \delta_{AP1} AP1 & \text{otherwise} \end{cases}$$

```
nuLFY_1 = 1.0
nuLFY_2 = 0.5
nuAP1_1 = 1.0
nuAP1_2 = 0.5
deltaLFY = 1.0
deltaAP1 = 1.0
k3 = 0.45

def compromiseffl(y,t):
   LFYi = y[0]
   AP1i = y[1]
   # a compromise FFL model equations
   if (FTsignal(t)==1) and (AP1i>k3):
      LFYdot = nuLFY_1*FTsignal(t)*1.0 - deltaLFY*LFYi
   elif ((FTsignal(t)==0) and (AP1i>k3)) or ((FTsignal(t)==1) and (AP1i<=k3)):
      LFYdot = nuLFY_2*1.0 - deltaLFY*LFYi
   else:
      LFYdot = -deltaLFY*LFYi

   if (FTsignal(t)==1) and (LFYi>k3):
      AP1dot = nuAP1_1*FTsignal(t)*1.0 - deltaAP1*AP1i
   elif ((FTsignal(t)==0) and (LFYi>k3)) or ((FTsignal(t)==1) and (LFYi<=k3)):
      AP1dot = nuAP1_2*1.0 - deltaAP1*AP1i
   else:
      AP1dot = -deltaAP1*AP1i
   return [LFYdot, AP1dot]
```

We solve the equations as previously...

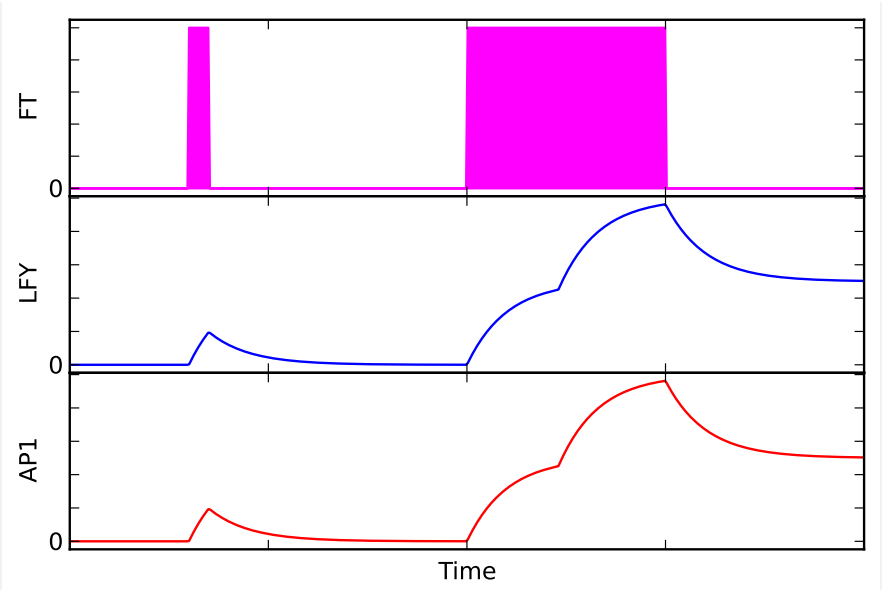
```
# initial conditions
LFY0 = 0.0 #
AP10 = 0.0 #
y0 = [LFY0, AP10] # initial condition vector
t = np.linspace(0, 20.0, 500) # time grid

# solve the DEs
soln = odeint(compromiseffl, y0, t, hmax = 0.1) #need hmax as can get wrong results o/
    wise - to do with adaptive step sizes in numerical method missing important points
LFY = soln[:, 0]
```

```
AP1 = soln[:, 1]
#for i in range(len(AP1)): print t[i],stepfn(t[i]),LFY[i],AP1[i]
```

. . . and produce the plots as before.

```
f, (ax1, ax2, ax3) = plt.subplots(3, sharex=True, sharey=False)
# Fine-tune figure; make subplots close to each other and hide x ticks for
# all plots.
f.subplots_adjust(hspace=0)
plt.setp([a.get_xticklabels() for a in f.axes[:]], visible=False)

steps = np.zeros(len(t))
for i in range(len(t)): steps[i]=FTsignal(t[i])
ax1.fill_between(t, 0, steps,color='magenta')
ax1.margins(0, 0.05)
ax1.set_ylabel("FT")
ax1.set_yticklabels(['','0'])

ax2.plot(t, LFY, label='LFY',color='blue')
ax2.margins(0, 0.05)
ax2.set_ylabel("LFY")
ax2.set_yticklabels(['','0'])

ax3.plot(t, AP1, label='AP1',color='red')
ax3.margins(0, 0.05)
ax3.set_xlabel("Time")
ax3.set_ylabel("AP1")
ax3.set_yticklabels(['','0'])
```

```
[<matplotlib.text.Text at 0x4bece90>, <matplotlib.text.Text at 0x4bf3990>]
```



S5 Versions used for reproducibility

```
# Versions used for reproducibility
import platform
print "Python", platform.python_version()
import IPython
print "IPython", IPython.__version__
print "Numpy", np.__version__
import scipy
print "Scipy", scipy.__version__
import matplotlib
print "Matplotlib", matplotlib.__version__
```

```
Python 2.7.3
IPython 1.0.0
Numpy 1.7.1
Scipy 0.12.0
Matplotlib 1.2.0
```