

# Electronic supplementary material for “Invariant predictions of epidemic patterns from radically different forms of seasonal forcing”

Irena Papst<sup>\*a</sup> and David J.D. Earn<sup>b,c</sup>

<sup>a</sup>Center for Applied Mathematics, 657 Frank H.T. Rhodes Hall, Cornell University, Ithaca, NY, 14853, United States

<sup>b</sup>Department of Mathematics and Statistics, McMaster University, 1280 Main Street West, Hamilton, ON L8S 4K1, Canada

<sup>c</sup>M. G. DeGroote Institute for Infectious Disease Research, 1280 Main Street West, Hamilton, ON L8S 4K1, Canada

June 20, 2019, 13:22

*Journal of the Royal Society Interface*

\*Corresponding author (email: ip98@cornell.edu)

## Contents

<b>S1 Supplementary Methods</b>	<b>2</b>
S1.1 Definition of the family of forcing functions . . . . .	2
S1.2 Computing bifurcation diagrams using <code>xppaut</code> and <code>AUTO</code> . . . . .	4
<b>S2 Supplementary Discussion</b>	<b>6</b>
S2.1 Spectral power . . . . .	6
S2.2 Invariance of bifurcations in a seasonally-forced predator-prey model . . . . .	15
<b>S3 Supplementary Tables</b>	<b>21</b>
S3.1 Initial conditions for $\mathcal{R}_0$ bifurcation diagrams . . . . .	21
<b>S4 Supplementary <math>\mathcal{R}_0</math> bifurcation diagrams</b>	<b>24</b>
<b>S5 Supplementary Code</b>	<b>28</b>
S5.1 Technical specifications . . . . .	28
S5.2 Sample code for generating brute force $\mathcal{R}_0$ bifurcation diagrams . . . . .	28
S5.3 Script to extract <code>AUTO</code> initial conditions from brute force bifurcation data . . . . .	33
S5.4 Sample code for generating <code>AUTO</code> $\mathcal{R}_0$ bifurcation diagrams in <code>xppaut</code> . . . . .	42
S5.5 Sample code for $\alpha$ continuation in <code>AUTO</code> . . . . .	50

# S1 Supplementary Methods

## S1.1 Definition of the family of forcing functions

Our family of forcing functions,  $\beta_p(t)$ , is parameterized by the shape variable  $p \in [-1, 1]$  and includes three important members: term-time forcing,  $p = -1$ , square wave forcing,  $p = 0$ , and sinusoidal forcing,  $p = 1$  (see panels 1, 3, and 5 of [figure 2](#) of the main text, respectively). In the main text, we also use the notation  $\beta_{\text{tt}}(t) = \beta_{-1}(t)$  and  $\beta_{\cos}(t) = \beta_1(t)$ <sup>1</sup>. Here we describe how any one of these functions is transformed into the others as  $p$  varies.

### S1.1.1 Shape function

We begin by defining a basic “shape function” for each part of the transformation:  $-1 \leq p \leq 0$  (term-time to square wave), and  $0 \leq p \leq 1$  (square wave to sinusoid). These functions are centred about the  $t$ -axis and have maximum and minimum values of  $\pm 1$ . Once we have defined these shape functions, we will stretch and shift them accordingly to obtain the family of forcing functions,  $\beta_p(t)$ .

The shape function for the term-time forcing to the square wave portion of our family ( $-1 \leq p \leq 0$ ) is defined as follows:

$$\text{TTtoSquare}(t, p) := \begin{cases} 1 & \text{school days,} \\ -1 & \text{non-school days,} \end{cases} \quad (\text{S1})$$

where school days and non-school days get scaled linearly with  $p$ . This transformation is most easily understood via the animation in [figure S1](#) (also provided as the Supplementary Video), but the details are as follows. All school holiday breaks start at their full width when  $p = -1$ . As  $p$  increases, all breaks (except summer) shrink linearly in width such that they disappear just as  $p = 0$ . The summer holiday starts at its full width when  $p = -1$  and widens into a single half-year break centred at  $t = 0.5$  just as  $p = 0$  (as required for the square wave). We use the UK calendar of school days to define our term-time forcing function throughout this work (see [table S1](#)).

Table S1: **School breaks in the UK.** We use these dates to define the term-time forcing function.

School Break	Calendar Dates	Model Days
Christmas	December 21-January 6	356-6
Easter	April 10-25	100-115
Summer	July 19-September 8	200-251
Autumn Half-Term	October 27-November 3	300-307

The shape function for the square wave to sinusoidal forcing portion of our family ( $0 < p \leq 1$ ) is defined as follows:

$$\text{SquaretoCos}(t, p) := \text{sign}(\cos(2\pi t)) |\cos(2\pi t)|^p, \quad (\text{S2})$$

---

<sup>1</sup>Table [S4](#) gives a complete list of symbols used in the analysis of the SIR model.

Figure S1: **An animation of the family of shape functions,  $s(t, p)$ , as  $p$  increases from  $-1$  to  $2$ .** The plus and minus buttons can be used to increase and decrease the frame rate, respectively. This animation may not work unless this document is viewed using [Adobe Acrobat Reader](#).

where  $\text{sign}(x)$  is the signum function, which evaluates to 1 if  $x > 0$ ,  $-1$  if  $x < 0$ , and 0 otherwise. We piece together both parts of the transformation into one shape function:

$$s(t, p) := \begin{cases} \text{TTtoSquare}(t, p) & \text{if } -1 \leq p \leq 0, \\ \text{SquaretoCos}(t, p) & \text{if } 0 < p \leq 1. \end{cases} \quad (\text{S3})$$

Note that, for the purposes of connecting term-time to sinusoidal forcing, we only need to consider  $p \in [-1, 1]$ , but there is no reason we cannot consider  $p > 1$  (and indeed we do).

### S1.1.2 Seasonal beta

Now that we have the shape function defined, we can proceed to shifting and scaling it as required. Firstly, it is important to ensure that the mean value of any forcing function over one period,  $\langle \beta \rangle^2$ , remains constant as we vary  $p$ , since  $\mathcal{R}_0 = \langle \beta \rangle / (\gamma + \mu)$  (if  $\langle \beta \rangle$  changes as  $p$  varies, then we cannot compare different forcing functions using  $\mathcal{R}_0$  bifurcation diagrams).

Note that, the average value of  $s(t, p)$  over one period for  $0 \leq p \leq 1$  is constant (equal to zero), and simply shifting  $s(t, p)$  vertically by  $\langle \beta \rangle$  would suffice to ensure that  $\langle \beta_p(t) \rangle = \langle \beta \rangle$  for  $0 \leq p \leq 1$ . However, for the term-time to square wave portion of the transformation, the average value of  $s(t, p)$  is always greater than zero, since students are in school more than they are on holiday, and so we must shift the shape function down by a value dependent on  $p$  for  $-1 \leq p \leq 0$ . In particular, we shift  $s(t, p)$  down by  $1 - 2p_s(p)$ , where  $p_s(p)$  gives the proportion of the school year spent in school.

Thus, we define a new function,  $\text{osc}_p(t)$ , which adjusts the shape function,  $s(t, p)$  to ensure that  $\langle \text{osc}_p(t) \rangle = 0$  for all  $p \in [-1, 1]$ , as follows:

$$\text{osc}_p(t) := \begin{cases} s(t, p) + (1 - 2p_s(p)) & \text{if } -1 \leq p \leq 0, \\ s(t, p) & \text{if } 0 < p \leq 1. \end{cases} \quad (\text{S4})$$

---

<sup>2</sup>We will use  $\langle \cdot \rangle$  to denote the average value of a periodic function over one period throughout this work.

Finally, we define  $\beta_p(t)$  by vertically scaling  $\text{osc}_p(t)$  with the amplitude of seasonality,  $\alpha$ , and by shifting it vertically by  $\langle\beta\rangle$ , as follows:

$$\beta_p(t) = \langle\beta\rangle[1 + \alpha \text{osc}_p(t)]. \quad (\text{S5})$$

A plot of the members of this family used in [figure 5](#) of the main text is given in [figure 2](#). To ensure that  $\beta_p(t) \geq 0$ , we require that [1]


$$\alpha \leq \frac{1}{2p_s(p)}. \quad (\text{S6})$$

## S1.2 Computing bifurcation diagrams using `xppaut` and `AUTO`

We use the `xppaut` [2] guide in [3] to create all  $\mathcal{R}_0$  bifurcation diagrams in this paper. In this section, we supplement the guide in [3] with further sample code, as well as instructions on how to both run this code and interact with `xppaut` to create bifurcation diagrams. For a guide to the theory of the bifurcations and stroboscopic (Poincaré) maps, see [4].

### S1.2.1 One parameter bifurcation diagrams in $\mathcal{R}_0$

To create  $\mathcal{R}_0$  bifurcation diagrams (such as those in [figure 3](#) of the main text), we first generate initial “brute force”  $\mathcal{R}_0$  bifurcation diagrams. This method involves fixing an  $\mathcal{R}_0$  value, solving the system numerically forward in time, and recording the value of  $I(t)$  after enough time has passed that the solution has settled onto an attractor. Sample `xppaut` code that generates data for one brute force bifurcation diagram can be found in [§S5.2](#).

The brute force method is disadvantageous as it is computationally intensive and only captures asymptotically stable solutions (attractors). However, in order to use `AUTO` (via its interface in `xppaut`) to generate bifurcation diagrams that include both attractors and repellers, we must start from an equilibrium. Also, since there are many different equilibria in this system, it is useful to use the brute force method to generate specific initial conditions for different attractors to ensure we can find them all in `AUTO`. These initial conditions are extracted from brute force data using the script found in [§S5.3](#) and then used in `xppaut` to generate bifurcation diagrams in `AUTO` (following the detailed instructions in [3]). Sample `xppaut` code for this purpose can be found in [§S5.4](#).

Once the one parameter bifurcation diagram has been computed in `AUTO`, select the bifurcation you wish to keep fixed in the `AUTO` window by using the `(G)rab`<sup>3</sup> command. For both the SIR and predator-prey models, we fixed the period doubling bifurcation off of the main (period 1) branch. Once the bifurcation has been “grabbed”, note the value of all parameters (in the `Param` menu of the main `xppaut` window) and the values of the state variables (in the `ICs` menu of the `xppaut` main window). We will use these values in the next step.

---

<sup>3</sup>Throughout this document, `xppaut` commands are in typewriter font, with keyboard shortcuts in parentheses.

### S1.2.2 Two parameter bifurcation diagrams in $\alpha$ and $p$

In order to extract the relationship between  $\alpha(p)$  that produces the dynamical invariance we have discovered (see [figure 4](#) of the main text), we compute a bifurcation diagram in these two parameters, while keeping all other parameters fixed. We use the initial conditions and parameters noted at the end of the previous step, pasting them into an `xppaut` file set up for continuation in  $\alpha$  (see [§S5.5](#) for sample code) and launch `xppaut` with this file. We run the continuation in  $\alpha$  just as we have for the  $\mathcal{R}_0$  bifurcation diagrams (see [§S1.2.1](#) and [3]), although with a few modified and additional steps to produce the two-parameter bifurcation in  $\alpha$  and  $p$ :

1. Launch `xppaut`

- From the command line, type `xppaut <filename>.ode`, where `<filename>` is replaced by the ode file name.

2. In the main `xppaut` window, load the parameter set with the  $\mathcal{R}_0$  bifurcation you wish you continue in  $\alpha$  and  $p$ .

- (F)ile  $\rightarrow$  (G)et par set

3. Integrate the system for this parameter set until it settles onto an equilibrium. (It is helpful to check the `Data` window to ensure that the system has converged to a periodic attractor.)

- (I)nitial cond  $\rightarrow$  (G)o, (I)nitial cond  $\rightarrow$  (L)ast, I  $\rightarrow$  L  $\rightarrow$  I  $\rightarrow$  L  $\rightarrow$  ...

4. Launch `AUTO`

- (F)ile  $\rightarrow$  (A)UTO

All commands are now in `AUTO`.

5. Begin the continuation in  $\alpha$  from the equilibrium point that was just settled on in `xppaut`. Note that this starting point is actually the bifurcation in  $\alpha$  that we would like to follow in  $\alpha$  and  $p$ , and in order to do so, `AUTO` needs to identify this point as a bifurcation in  $\alpha$ . It will not succeed in identifying this bifurcation when continuation is started from this point. Instead, we need to have `AUTO` pass over this point during a continuation, so we start the continuation in  $\alpha$ , and then quickly stop it manually so that there is a point from which start a continuation backwards and pass over the bifurcation point.

The steps for this procedure are as follows:

- (R)un the continuation
- **Abort** it before your trajectory leaves the display window and stops automatically at the maximum  $\alpha$  value specified
- (G)rab the point at which you aborted the continuation

- Change the sign of the step `(N)umerics > Ds` to continue backwards
  - `(R)un` the continuation backwards
  - `(G)rab` the newly-identified  $\alpha$  bifurcation point
6. Set up a two-parameter bifurcation diagram (the min and max values of the vertical axis to the range of  $p$  may need to be changed in this menu).
- `(A)xes → (T)wo par`
7. Run the two-parameter continuation. You may need to experiment with changing the sign and/or size of the step size `Ds` to compute all parts of the  $\alpha(p)$  curve. For instance, if the continuation seems to stall, try aborting the continuation and then starting it again after changing the sign of `Ds`. Alternatively, you can try a larger step size over troublesome  $\alpha$  intervals (with this family of forcing functions, there tends to difficulty around  $p = 0$ ).

The resulting  $a(p)$  curve for the period doubling  $\mathcal{R}_0$  bifurcation off of the main branch is plotted in [figure 4](#) of the main text and [figure S2](#) (with the latter including other curves derived in [§S2.1.1](#)).

## S2 Supplementary Discussion

### S2.1 Spectral power

The family of forcing functions connecting term-time and sinusoidal forcing is constructed arbitrarily; there are infinitely many ways of connecting these two functions. Thus, we hope to find some way of characterizing these forcing functions in a more generic way beyond  $(p, \alpha)$ , the parameters of this particular family. Ideally, this property would help explain why it is that we are able to conserve bifurcation structure—by simply adjusting the amplitude of forcing—despite significantly altering the shape of the forcing function. Previous work [\[5\]](#) suggests that average spectral power could characterize periodic signals beyond their specific shape.

#### S2.1.1 Average spectral power

We calculate the average spectral power,  $\mathcal{P}(p, \alpha)$ , of each member of our family of forcing functions by integrating the square of the signal over one period (for  $t \in [0, 1]$ ) [\[6\]](#).

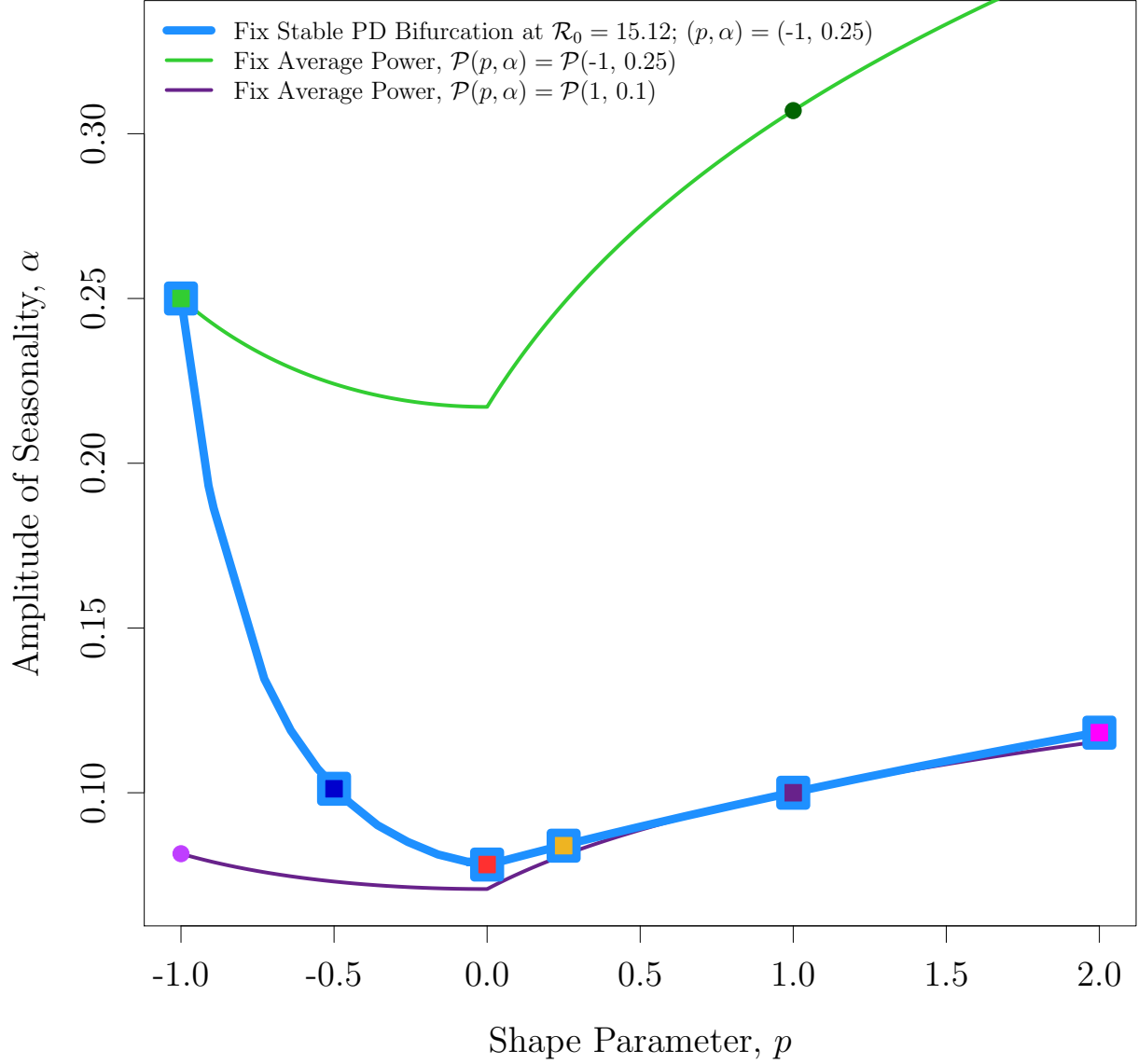


Figure S2: **Several different relationships between the shape parameter,  $p$ , and the amplitude of seasonality  $\alpha$ .** The thicker blue curve corresponds to the curve plotted in figure 4 of the main text and is the continuation of the stable period doubling bifurcation in the  $(p, \alpha)$  plane, starting at term-time forcing ( $p = -1$ ) with the amplitude estimated from data  $\alpha = 0.25$  (see §S1.2.2). The points denoted by squares correspond to the forcing functions used in figure 5 of the main text and §S4, and are plotted in figure 2. The other two curves denote fixed average power (see §S2.1), where each point on these curves corresponds to an  $\mathcal{R}_0$  bifurcation diagram in §S2.1.1.

First, assume that  $-1 \leq p \leq 0$ :

$$\mathcal{P}(p, \alpha) = \int_0^1 (\beta_p(t))^2 dt, \quad (\text{S7a})$$

$$= \int_0^1 (\alpha[s(t, p) + (1 - 2p_s(p))])^2 dt, \quad (\text{S7b})$$

$$= \int_{\text{school days}} (\alpha[1 + (1 - 2p_s(p))])^2 dt + \int_{\text{non-school days}} (\alpha[-1 + (1 - 2p_s(p))])^2 dt, \quad (\text{S7c})$$

$$= \int_{\text{school days}} (2 - 2p_s(p))^2 \alpha^2 dt + \int_{\text{non-school days}} (-2p_s(p))^2 \alpha^2 dt, \quad (\text{S7d})$$

$$= \alpha^2 \left[ (2 - 2p_s(p))^2 \int_{\text{school days}} dt + (-2p_s(p))^2 \int_{\text{non-school days}} dt \right], \quad (\text{S7e})$$

$$= \alpha^2 [(2 - 2p_s(p))^2 (p_s(p)) + (-2p_s(p))^2 (1 - p_s(p))], \quad (\text{S7f})$$

$$= 4\alpha^2 p_s(p)(1 - p_s(p)). \quad (\text{S7g})$$

Now assume that  $0 < p \leq 1$ :

$$\mathcal{P}(p, \alpha) = \int_0^1 (\beta_p(t))^2 dt, \quad (\text{S8a})$$

$$= \int_0^1 [\alpha \operatorname{sign}(\cos(2\pi t)) |\cos(2\pi t)|^p]^2 dt, \quad (\text{S8b})$$

$$= \alpha^2 \int_0^1 [\cos^2(2\pi t)]^p dt. \quad (\text{S8c})$$

Letting  $\theta = 2\pi t$ , we have that  $d\theta = 2\pi dt$ , and so from equation S8c, we get

$$\mathcal{P}(p, \alpha) = \frac{\alpha^2}{2\pi} \int_0^{2\pi} [\cos^2(\theta)]^p d\theta. \quad (\text{S9})$$

Note that  $\cos^2(\theta)$  is symmetric about  $\theta = \frac{n\pi}{2}$  for every  $n \in \mathbb{Z}$ , so over the interval  $[0, 2\pi]$ , we have the property that

$$\int_0^{2\pi} \cos^2(\theta) d\theta = 4 \int_0^{\frac{\pi}{2}} \cos^2(\theta) d\theta. \quad (\text{S10})$$

This symmetry is preserved for any power  $p \geq 0$  in equation S9, and so we have that

$$\mathcal{P}(p, \alpha) = \frac{\alpha^2}{2\pi} \int_0^{2\pi} [\cos^2(\theta)]^p d\theta, \quad (\text{S11a})$$

$$= \frac{\alpha^2}{2\pi} \left[ 4 \int_0^{\frac{\pi}{2}} [\cos^2(\theta)]^p d\theta \right], \quad (\text{S11b})$$

$$= \frac{2\alpha^2}{\pi} \int_0^{\frac{\pi}{2}} \cos^{2p}(\theta) d\theta. \quad (\text{S11c})$$



Then, using [definition 5.12.1](#) and [identity 5.12.2](#) from [7], we have that

$$\int_0^{\frac{\pi}{2}} \sin^{2a-1}(\theta) \cos^{2b-1}(\theta) d\theta = \frac{1}{2} \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}, \quad a, b > 0, \quad (\text{S12})$$

where  $\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$  is Euler's gamma function, which can be applied to equation [S11c](#) by setting  $a = \frac{1}{2}$  and  $b = p + \frac{1}{2}$  (for  $p > -\frac{1}{2}$ ) to get

$$\mathcal{P}(p, \alpha) = \frac{2\alpha^2}{\pi} \int_0^{\frac{\pi}{2}} \cos^{2p}(\theta) d\theta \quad (\text{S13a})$$

$$= \frac{2\alpha^2}{\pi} \left[ \frac{1}{2} \frac{\Gamma(\frac{1}{2})\Gamma(p + \frac{1}{2})}{\Gamma(p + 1)} \right] \quad (\text{S13b})$$

$$= \frac{\alpha^2}{\sqrt{\pi}} \frac{\Gamma(p + \frac{1}{2})}{\Gamma(p + 1)}. \quad (\text{S13c})$$

Thus, the average power of any member of our family of forcing functions can be calculated using the following formula:

$$\mathcal{P}(p, \alpha) = \begin{cases} 4(1 - p_s(p))p_s(p)\alpha^2 & \text{for } -1 \leq p \leq 0; \\ \frac{\alpha^2}{\sqrt{\pi}} \frac{\Gamma(p + \frac{1}{2})}{\Gamma(p + 1)} & \text{for } p \geq 0. \end{cases} \quad (\text{S14})$$

The resultant average power surface is plotted in [figure S3](#).

These expressions make it easy for us to compare the average power of forcing functions that generate qualitatively equivalent dynamics (those used in [figure 5](#) of the main text, for instance) and we see that among such forcing functions, the average power varies (see [table S2](#)).

**Table S2: Average spectral power calculated for the forcing functions featured in [figure 5](#) of the main text per forcing period (one year).**

$p$	$\alpha$	Average Power
-1.00	0.250	0.0471
-0.50	0.101	0.0096
0.00	0.078	0.0061
0.25	0.084	0.0054
1.00	0.100	0.0050

Although these average power values are not equivalent, it is not clear how much variance in average power would constitute a “large” difference with respect to the resultant dynamics. Thus, we consider whether fixed average power among forcing functions could lead to an invariance in bifurcation structure. We fix average power in two cases: term-time forcing ( $p = -1$ ) with  $\alpha = 0.25$  and sinusoidal forcing ( $p = 1$ ) with  $\alpha = 0.1$ . We then generate two

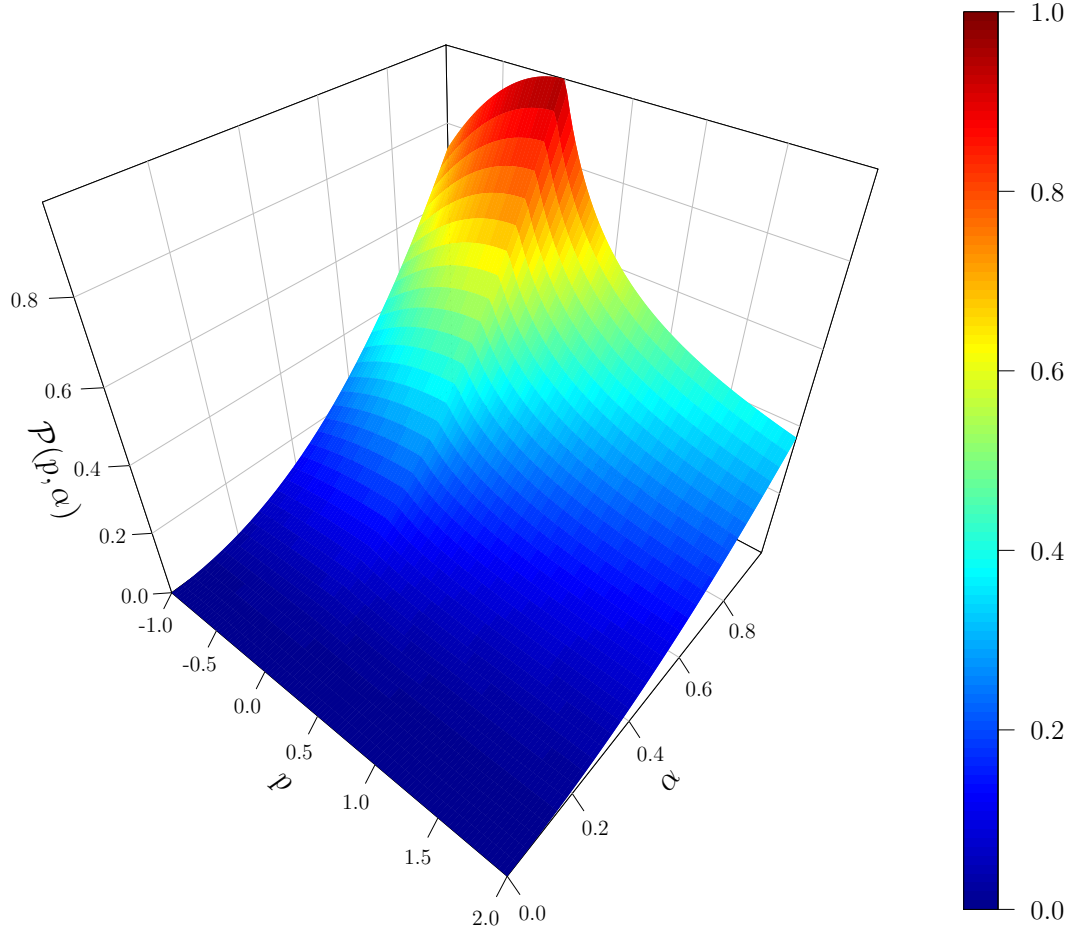


Figure S3: **Average power,  $\mathcal{P}(p, \alpha)$ , for our family of forcing functions, plotted for a range of the shape parameter,  $p$ , and the amplitude of seasonality,  $\alpha$ .**

$\alpha(p)$  curves, one where  $\mathcal{P}(p, \alpha) = \mathcal{P}(-1, 0.25)$  and another where  $\mathcal{P}(p, \alpha) = \mathcal{P}(1, 0.1)$  (see figure S2).

We check whether forcing functions with fixed average power could yield qualitatively equivalent dynamics. In figure S4, we compare the  $\mathcal{R}_0$  bifurcations diagrams of two forcing functions with  $\mathcal{P}(p, \alpha) = \mathcal{P}(-1, 0.25)$ : term-time forcing with  $\alpha = 0.25$  and sinusoidal forcing with  $\alpha = 0.3070$  (the latter is marked with a dark green point in figure S2). Similarly, in figure S5, we consider the case where  $\mathcal{P}(p, \alpha) = \mathcal{P}(1, 0.1)$  by comparing sinusoidal forcing with  $\alpha = 0.1$  to that of term-time forcing with  $\alpha = 0.0814$  (marked with a purple point in figure S2). We conclude that fixed average power cannot account for the observed dynamical invariance.

### S2.1.2 Power spectra

While average power is not invariant among forcing functions that yield qualitatively equivalent dynamics, perhaps it is too crude of a measure to capture the observed relationship.

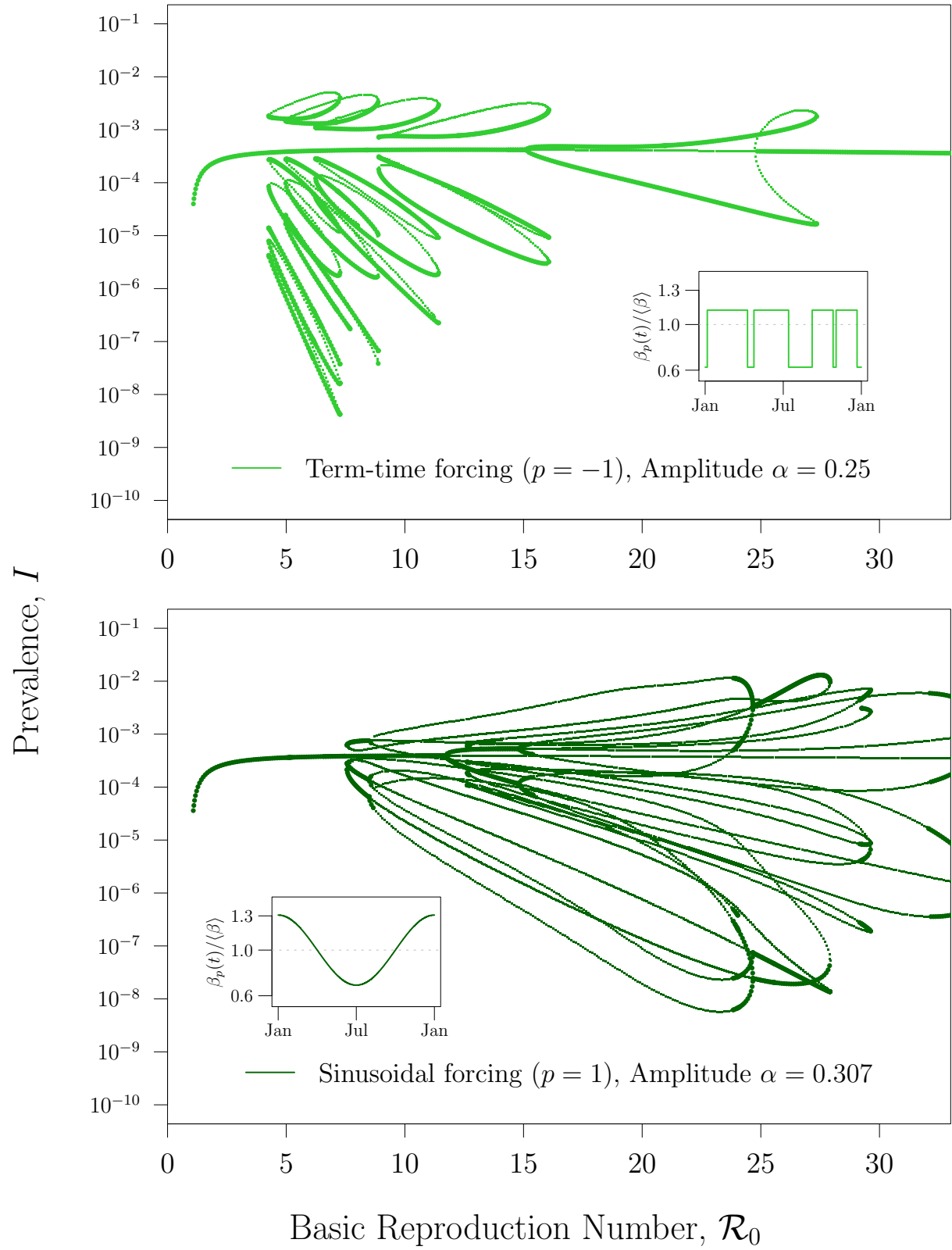


Figure S4:  $\mathcal{R}_0$  bifurcation diagrams for term-time and sinusoidal forcing with fixed total power, where  $\mathcal{P}(p, \alpha) = \mathcal{P}(-1, 0.25)$  (see figure S2).

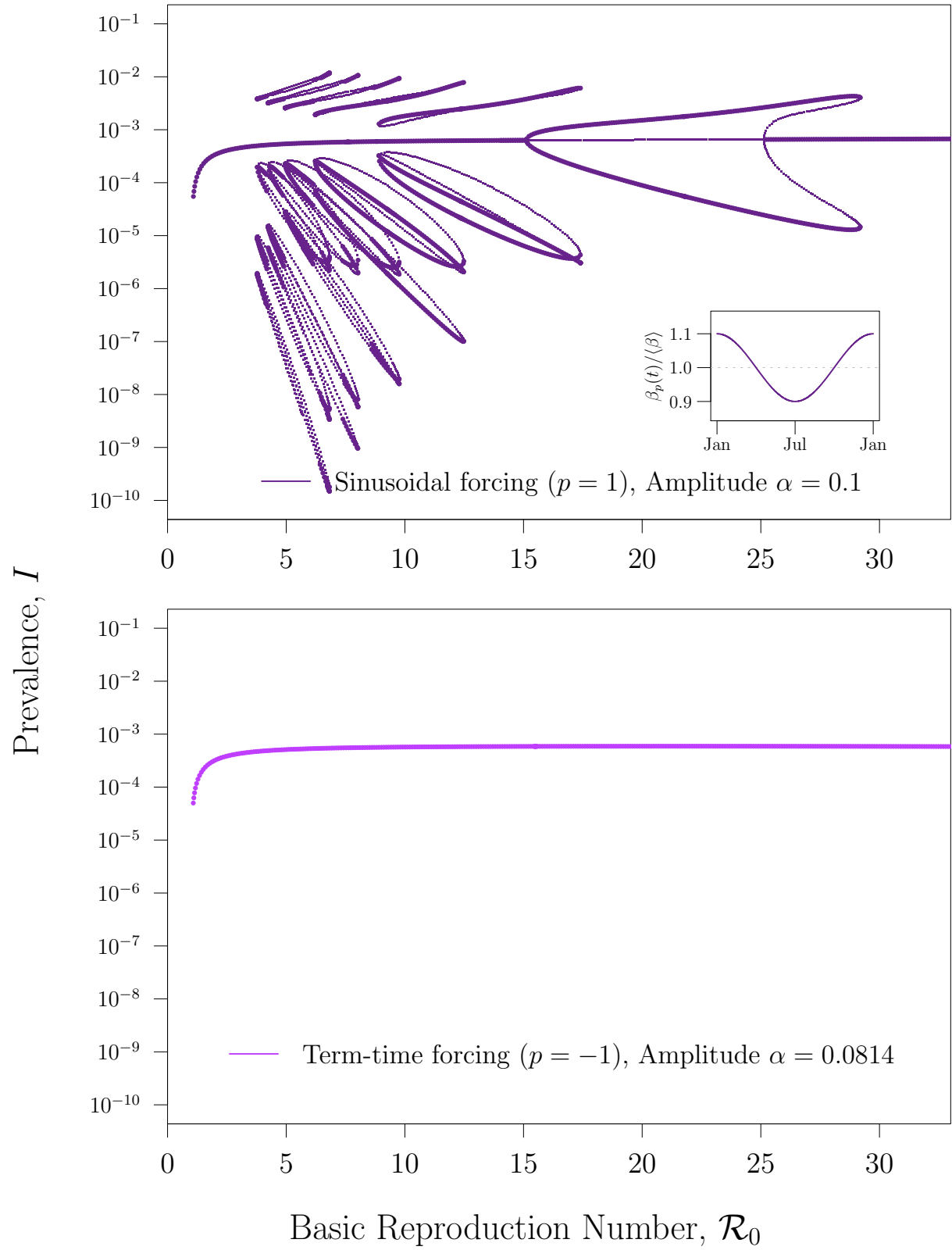


Figure S5:  $\mathcal{R}_0$  bifurcation diagrams for term-time and sinusoidal forcing with fixed total power, where  $\mathcal{P}(p, \alpha) = \mathcal{P}(1, 0.1)$  (see figure S2).

Another, finer, property of a signal that could characterize these forcing functions beyond their specific shape is the full power spectrum. In figure S6, we plot the power spectra for the forcing functions used in figure 5 of the main text. However, just as with the average power of each signal, the power spectra are too distinct to explain the dynamical invariance we seek to understand.

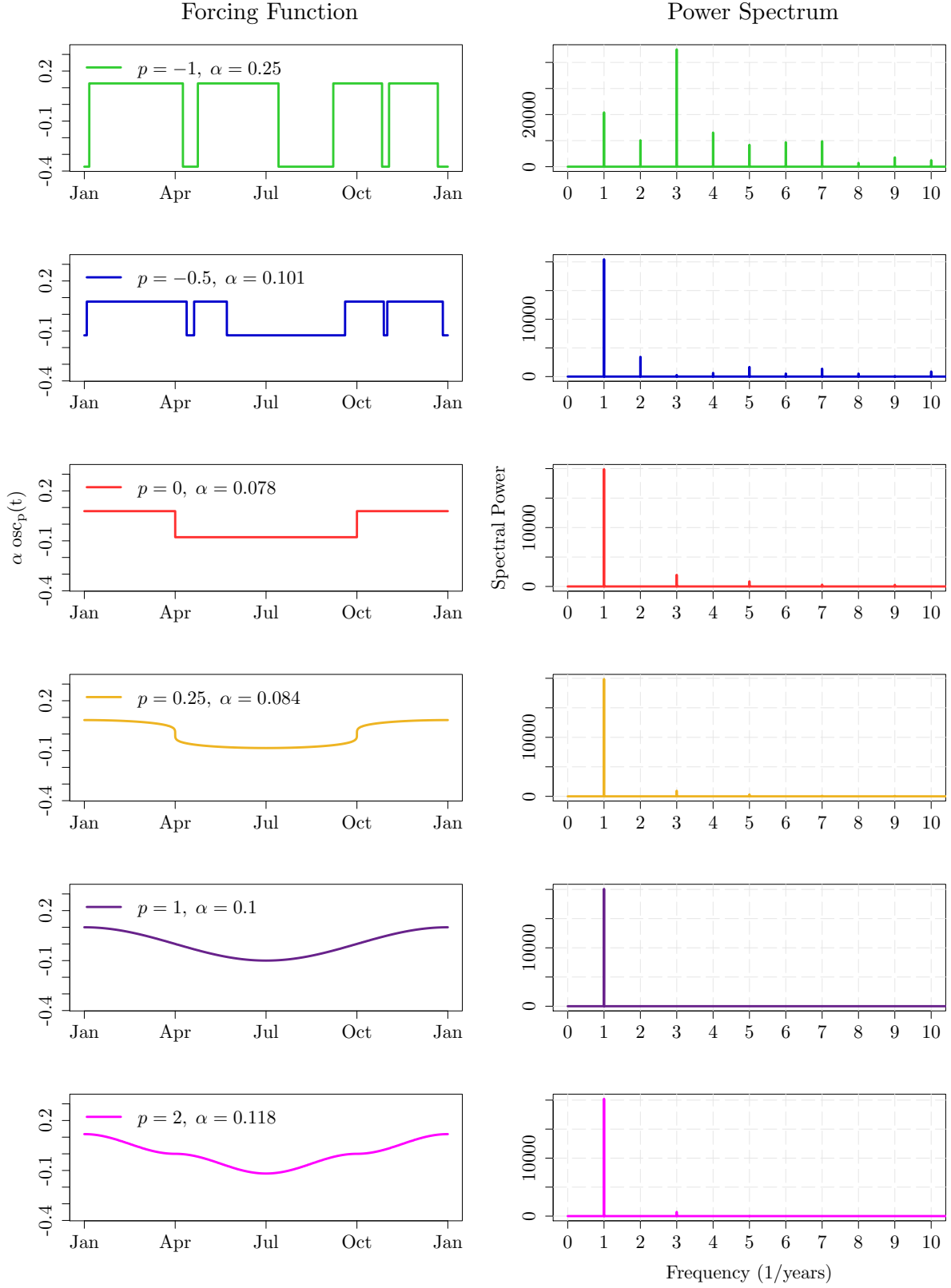


Figure S6: Power spectra for the forcing functions used in figure 5 of the main text (denoted by square points in figure S2).

## S2.2 Invariance of bifurcations in a seasonally-forced predator-prey model

Invariance of bifurcations similar to that shown for the SIR model in [figure 5](#) of the main text can be found in a seasonally-forced predator-prey model.

### S2.2.1 Model

Let  $x, y$  represent the population of prey and predators, respectively (normalized to the carrying capacity of prey). The growth and decay of each population is governed by two coupled differential equations:

$$x' = \varepsilon(1 - x) + rx(1 - x) - \rho(t)xy, \quad (\text{S15a})$$

$$y' = \delta\rho(t)xy - y. \quad (\text{S15b})$$

The prey population grows according to a combination of density-dependent immigration at rate  $\varepsilon$  and logistic growth at maximum rate  $r$ , and decays due to consumption by predators at a time-dependent rate  $\rho(t)$  (per predator). The predator grows due to consumption of prey, though at a discounted rate  $\delta\rho(t)$ , with  $\delta$  representing the efficiency of biomass conversion from prey to predator. Similar models have previously been analyzed in [\[8–11\]](#).

Seasonal forcing occurs in the time-dependent interaction rate,  $\rho(t)$  and we assume the same form as that of the family of forcing functions defined for the SIR model (see [§S1.1](#)):

$$\rho(t) = \rho_0[1 + \alpha \text{osc}_p(t)]. \quad (\text{S16})$$

The parameter  $\rho_0$  represents the average value of the interaction rate between species,  $\alpha$  is the amplitude of forcing, and  $p$  is the forcing function shape parameter.

While there is an analogy between predator-prey and infective-susceptible models, we emphasize that the two models we have considered are mathematically distinct. The formal structural differences between the predator-prey model (equation [S15](#)) and the SIR model (equation [1](#)) are the nonlinear (logistic) prey growth term and the imperfect conversion of prey biomass into predator biomass.

### S2.2.2 Results

We observe a similar quantitative invariance in the structure of the  $\rho_0$  bifurcation diagrams for this model as with the  $\mathcal{R}_0$  bifurcation diagrams for the SIR model (see [figure S7](#)). When the seasonal amplitude of forcing is adjusted according to the  $p(\alpha)$  function for this model ([figure S8](#)), which is derived by fixing the  $\rho_0$  value of the principal period doubling bifurcation, the four fold bifurcations listed in [table S3](#) match to similar precision. Other bifurcations are also compared in [figure S9](#) for three forcing functions with amplitudes taken from [figure S8](#). While the “birth fold” bifurcation locations are invariant among the forcing functions, corresponding “death folds” and intermediate period doubling bifurcations are not (*e.g.*, the period 6 branch in [figure S9](#)). Because many bifurcations occur in a small range of  $\rho_0$ , we show a stretched version of [figure S9](#) in [figure S10](#); the invariances remain apparent in this diagram, and the non-invariances are much clearer.

Table S3: Invariance of fold bifurcations [12] in a seasonally forced predator-prey model (equation S15, equation S16) at different  $\rho_0$  values when the principal period doubling (PD) bifurcation at  $\rho_0 = 22.5995$  is matched. Fold ( $n$ ) refers to a fold bifurcation that gives rise to a period  $n$  attractor (an  $n$ -year population cycle) as  $\rho_0$  is increased. The PD does not occur at precisely the same  $\rho$  for each  $(p, \alpha)$  pair due to slight inaccuracies of the numerical continuation software [2, 13]. The relative difference refers to  $\max[(x - x_{\text{squ}})/x_{\text{squ}}]$ , where  $x$  is the value of  $\rho_0$  at the bifurcation of interest and  $x_{\text{squ}}$  is its value for square wave forcing ( $p = 0$ ). All the data in this table are plotted in figure S9.

Forcing Function		$\rho_0$ Bifurcation Point				
$p$	$\alpha$	PD (2)	Fold (3)	Fold (4)	Fold (5)	Fold (6)
0.0	0.1170	22.5995	8.23380	4.87989	4.14535	3.95685
0.5	0.1344	22.5995	8.23231	4.87817	4.14287	3.94968
1.0	0.15	22.5995	8.23206	4.87764	4.14136	3.94474
Relative Difference		0.0000	0.00021	0.00046	0.00096	0.00306



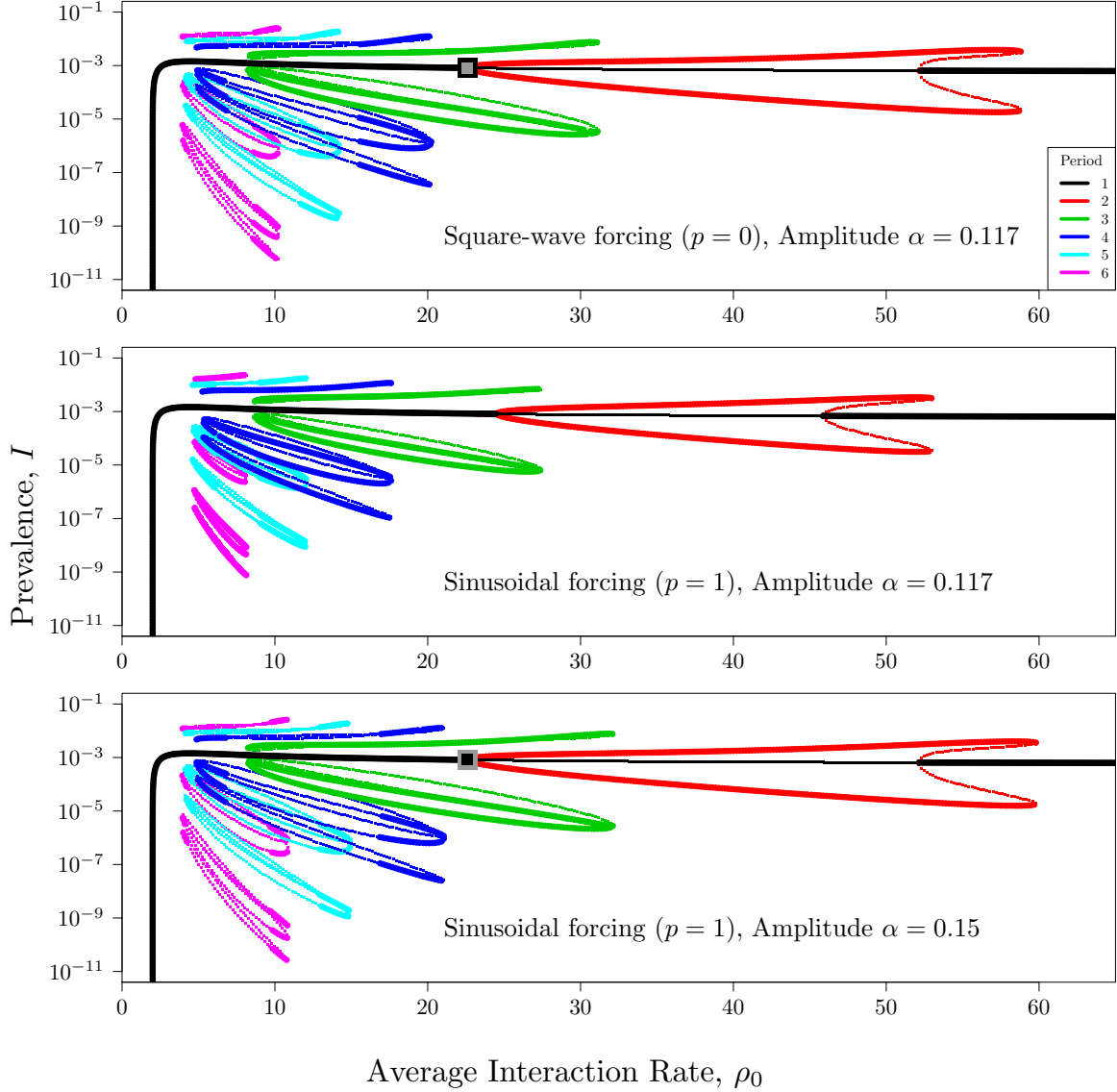


Figure S7:  $\rho_0$  bifurcation diagrams for the annual stroboscopic map of the seasonally forced predator-prey model (equation S15) with different patterns and amplitudes of forcing. In the top two panels, the forcing pattern is different but the associated amplitudes are the same. In the bottom two panels, the forcing pattern is the same but the amplitudes are different. For all panels,  $\varepsilon = 10^{-3}$ ,  $r = 10^{-2}$ ,  $\delta = 1/2$ . The values of parameters that vary are indicated in each panel. Thick lines show stable periodic solutions (attractors) and thin lines show unstable periodic solutions (repellers). At each  $\rho_0$ , the number of points of a given colour indicates the period of the associated attractor or repeller. The qualitative similarity of the top and bottom panels shows that different forcing patterns can yield the same bifurcation structure (for different forcing amplitudes). A precise quantitative correspondence of bifurcation points is demonstrated in figure S9 and table S3. The points highlighted with squares in the top and bottom panels (at  $\rho_0 = 22.5995$ ) correspond to the similarly highlighted points in the two-dimensional bifurcation diagram in figure S8.

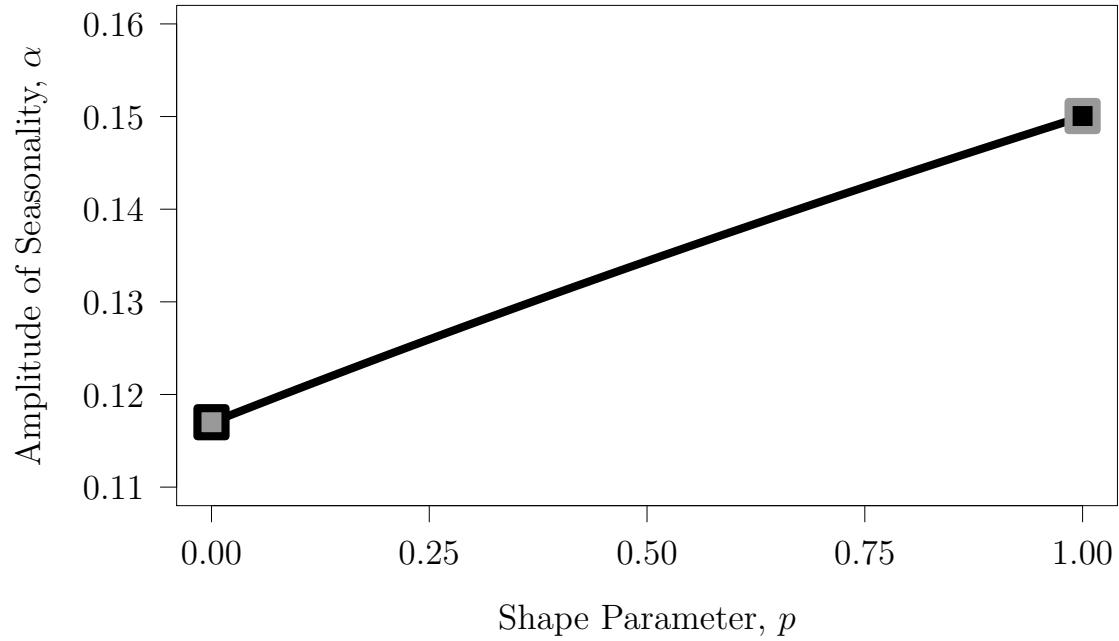


Figure S8: **Continuation of the stable period doubling (PD) bifurcation in the two-dimensional  $(p, \alpha)$  parameter plane** (see [Methods](#) and §S1.2.2). The continuation was initiated at sinusoidal forcing ( $p = 1$ ) with the amplitude  $\alpha = 0.15$  and extends to square wave forcing ( $p = 0$ ). The resulting function,  $\alpha(p)$ , shows how the amplitude of seasonality ( $\alpha$ ) must change as the forcing pattern ( $p$ ) is changed, if we wish to fix the values of all the other model parameters (in particular,  $\rho_0 = 22.5995$ ). The points highlighted with squares correspond to the similarly highlighted points in the top and bottom panels of figure S7.

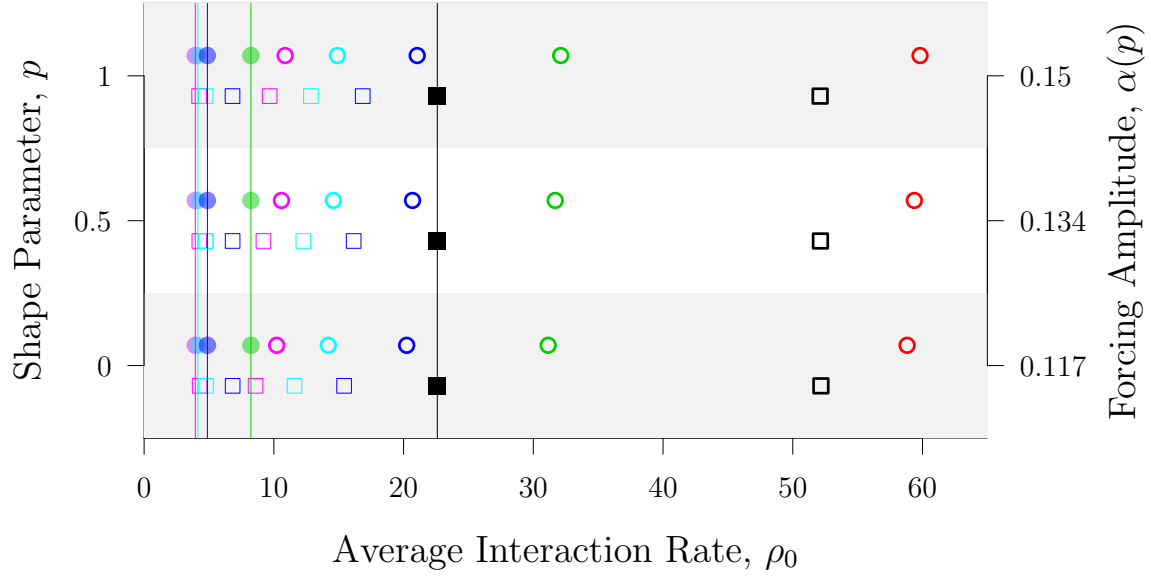


Figure S9: **Graphical representation of bifurcation invariance in the seasonally forced predator-prey model (equation S15).** For three forcing patterns ( $p$ , left vertical axis) and amplitudes determined by the function shown in figure S8 ( $\alpha(p)$ , right vertical axis), the values of  $\rho_0$  (horizontal axis) at which bifurcations occur are indicated (with colours that correspond to those used in figure S7). Period doubling (PD) bifurcations are shown with squares and fold bifurcations are shown with circles. The PD that is fixed by construction is shown with solid black squares. Folds that turn out to be invariant are marked with solid circles (these are all of the “birth folds”). Other non-invariant bifurcations (“death folds” and intermediate PDs) are marked with open symbols. The full  $\rho_0$  bifurcation diagrams for  $p = 0$  (square wave forcing) and  $p = 1$  (sinusoidal forcing) are shown in the top and bottom panels of figure S7.

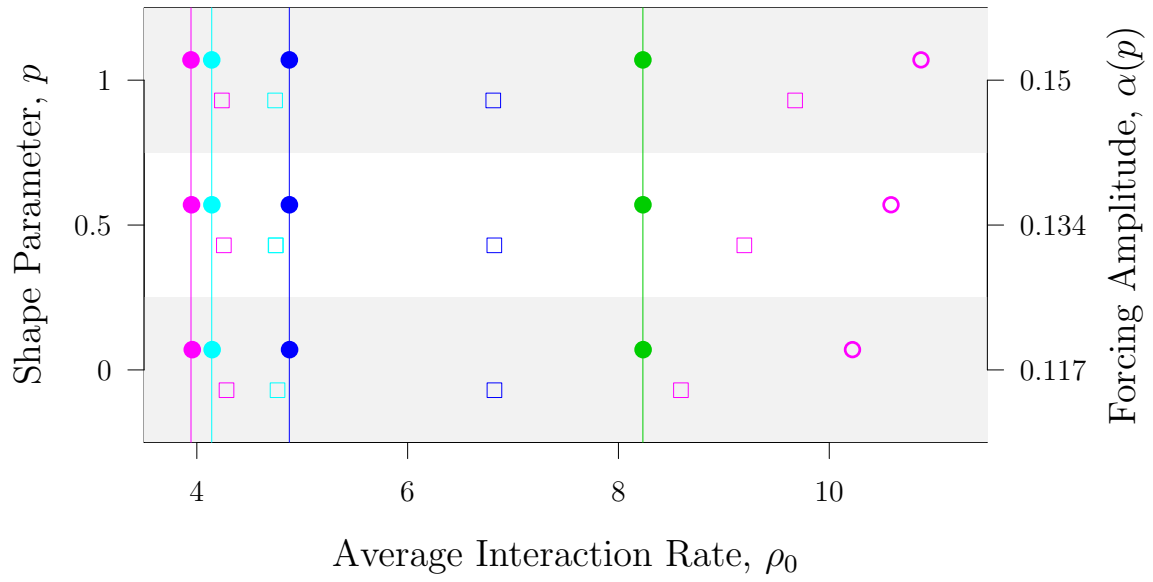


Figure S10: The same bifurcation data as in figure S9, but for a smaller interval in  $\rho_0$ .

## S3 Supplementary Tables

Table S4: List of symbols used in the SIR model, their meanings, and units.

Symbol	Meaning	Units
$t$	time	years
$S$	proportion of susceptible individuals	—
$I$	proportion of infected individuals	—
$R$	proportion of removed individuals	—
$\beta$	disease transmission rate	1/years
$\gamma$	disease recovery rate	1/years
$\mu$	<i>per capita</i> birth rate, natural death rate	1/years
$\mathcal{R}_0$	basic reproduction number	—
$\alpha$	amplitude of seasonality	—
$\langle \beta \rangle$	mean disease transmission rate	1/years
$p_s$	proportion of the year children spend in school	—
$p$	forcing function shape parameter	—
$\beta_{\cos}(t)$	sinusoidal disease transmission rate	1/years
$\beta_{\text{tt}}(t)$	term-time disease transmission rate	1/years
$\beta_{\text{high}}$	high disease transmission rate	1/ years
$\beta_{\text{low}}$	low disease transmission rate	1/ years
$\text{osc}_{\text{tt}}(t)$	term-time oscillation function	—
$\text{osc}_p(t)$	oscillation function, parameterized by the shape parameter, $p$	—
$\beta_p(t)$	disease transmission rate, parameterized by the shape parameter, $p$	1/years
$\mathcal{P}(p, \alpha)$	average power in forcing function	1/years

### S3.1 Initial conditions for $\mathcal{R}_0$ bifurcation diagrams

The following tables give the `xppaut` initial conditions used to generate the  $\mathcal{R}_0$  bifurcation diagrams in §S4.

Table S5: Initial conditions for figure S11, where  $(p, \alpha) = (-1, 0.25)$ .

Branch	$\log(S_0)$	$\log(I_0)$	$\mathcal{R}_0$
Period 1	−0.9216724	−3.3829732	8.3999996
Period 2	−1.2411177	−3.2906444	20.9699990
Period 3	−1.006153	−3.1036749	12.2100000
Period 4	−1.0476377	−4.1699929	8.9600000
Period 5	−0.95821649	−4.15801	7.1399999
Period 6	−0.79773831	−7.3171558	6.1399999
Period 7	−0.8987931	−4.5482569	5.8400002

Table S6: Initial conditions for figure S12, where  $(p, \alpha) = (-0.5, 0.101)$ .

Branch	$\log(S_0)$	$\log(I_0)$	$\mathcal{R}_0$
Period 1	-0.93079025	-3.2840197	8.6099997
Period 2	-1.2075626	-3.057709	18.4950010
Period 3	-1.0044775	-4.3511157	10.6650000
Period 4	-0.95428139	-5.5148168	8.7600002
Period 5	-0.69504982	-4.2612557	5.5349998
Period 6	-0.81840485	-7.6078467	6.4949999
Period 7	-0.89881647	-4.7360983	5.8800001

Table S7: Initial conditions for figure S13, where  $(p, \alpha) = (0, 0.078)$ .

Branch	$\log(S_0)$	$\log(I_0)$	$\mathcal{R}_0$
Period 1	-0.73213571	-3.2581513	5.4299998
Period 2	-1.2447836	-2.8120999	20.4000000
Period 3	-1.149428	-4.3156195	12.2100000
Period 4	-0.94534254	-5.2238641	10.3050000
Period 5	-0.66928089	-3.9008913	5.1900001
Period 6	-0.6259011	-5.1727471	4.5000000
Period 7	-0.64729881	-5.3032851	4.0200000

Table S8: Initial conditions for figure S14, where  $(p, \alpha) = (0.25, 0.084)$ .

Branch	$\log(S_0)$	$\log(I_0)$	$\mathcal{R}_0$
Period 1	-0.9165343	-3.2248225	8.3400002
Period 2	-1.2146921	-2.8767755	18.5550000
Period 3	-1.053017	-3.8600008	10.1700000
Period 4	-1.1204562	-4.8990722	10.4850000
Period 5	-0.90001303	-2.1649606	8.8800001
Period 6	-0.8862555	-8.7333384	7.7100000
Period 7	-0.74673504	-8.0441036	6.4949999

Table S9: **Initial conditions for figure S15**, where  $(p, \alpha) = (1, 0.1)$ .

Branch	$\log(S_0)$	$\log(I_0)$	$\mathcal{R}_0$
Period 1	-0.87659162	-3.2304842	7.5999999
Period 2	-1.2681592	-2.7434311	21.7800010
Period 3	-1.0293249	-2.6030924	12.3600000
Period 4	-1.0524955	-4.4974136	9.2399998
Period 5	-0.70523274	-4.2141695	5.7199998
Period 6	-0.70315689,	-3.7062016	4.3800001
Period 7	-0.97648102	-5.562336	6.7350001

Table S10: **Initial conditions for figure S16**, where  $(p, \alpha) = (2, 0.118)$ .

Branch	$\log(S_0)$	$\log(I_0)$	$\mathcal{R}_0$
Period 1	-0.87659162	-3.2304842	7.5999999
Period 2	-1.2681592	-2.7434311	21.7800010
Period 3	-1.0293249	-2.6030924	12.3600000
Period 4	-0.94534254	-5.2238641	10.3050000
Period 5	-0.70523274	-4.2141695	5.7199998
Period 6	-0.70315689,	-3.7062016	4.3800001
Period 7	-0.97648102	-5.562336	6.7350001

## S4 Supplementary $\mathcal{R}_0$ bifurcation diagrams

The following plots are the full  $\mathcal{R}_0$  bifurcation diagrams for the forcing functions used in figure 5 of the main text (marked with squares points in figure S2). These are also the forcing functions for which both the average spectral power and power spectra have been computed (see table S2 and figure S6, respectively). The `xppaut` initial conditions used to compute each  $\mathcal{R}_0$  bifurcation diagram are given in §S3.1.

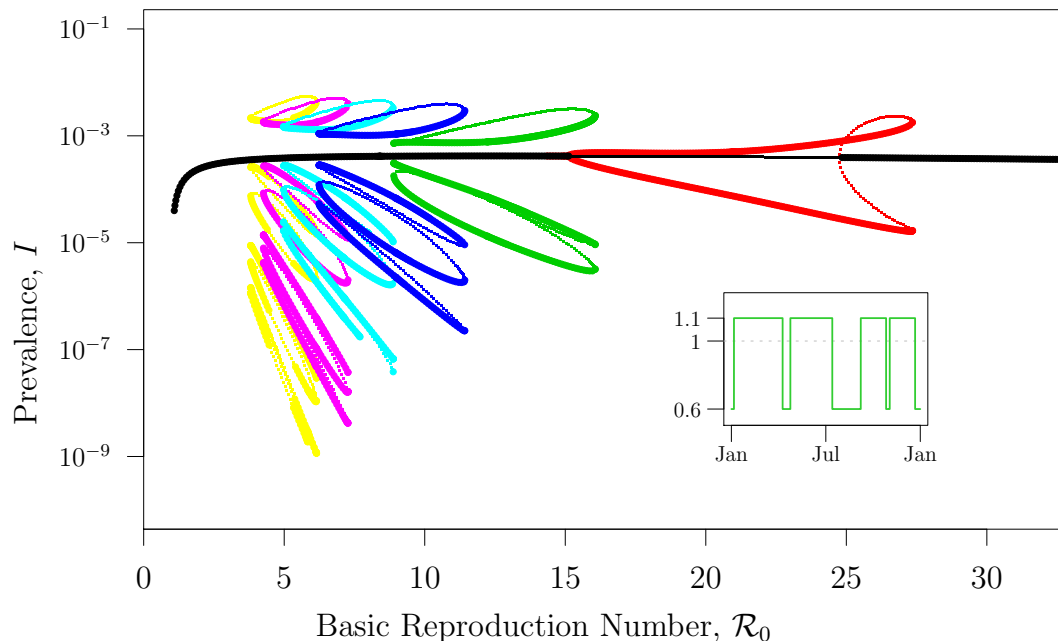


Figure S11:  $\mathcal{R}_0$  bifurcation diagram for  $(p, \alpha) = (-1, 0.25)$ .



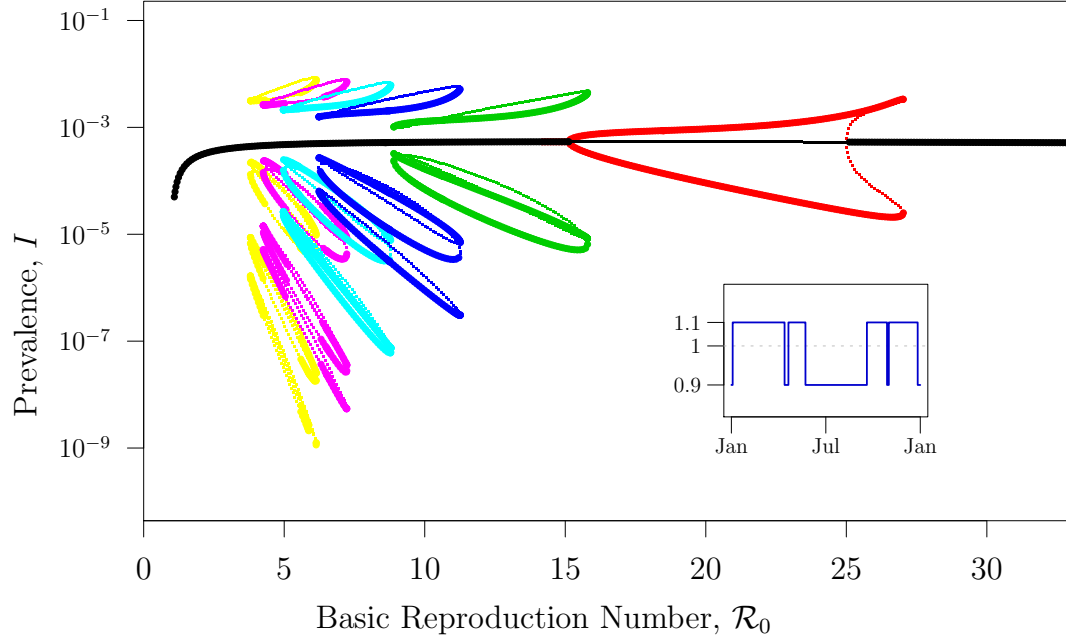


Figure S12:  $\mathcal{R}_0$  bifurcation diagram for  $(p, \alpha) = (-0.5, 0.101)$ .

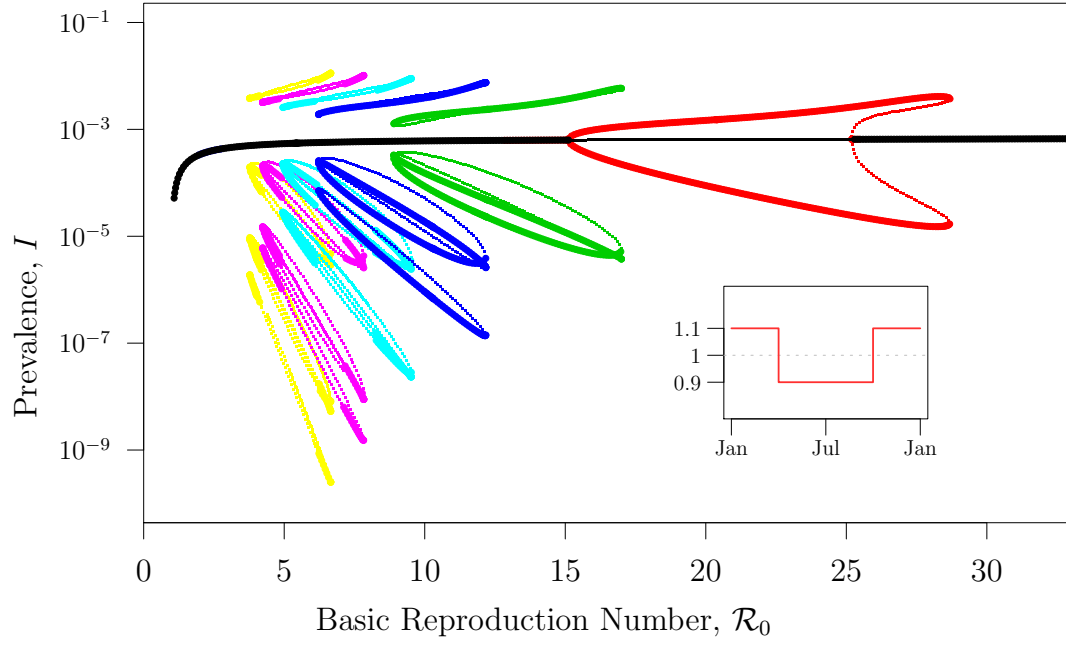


Figure S13:  $\mathcal{R}_0$  bifurcation diagram for  $(p, \alpha) = (0, 0.078)$ .

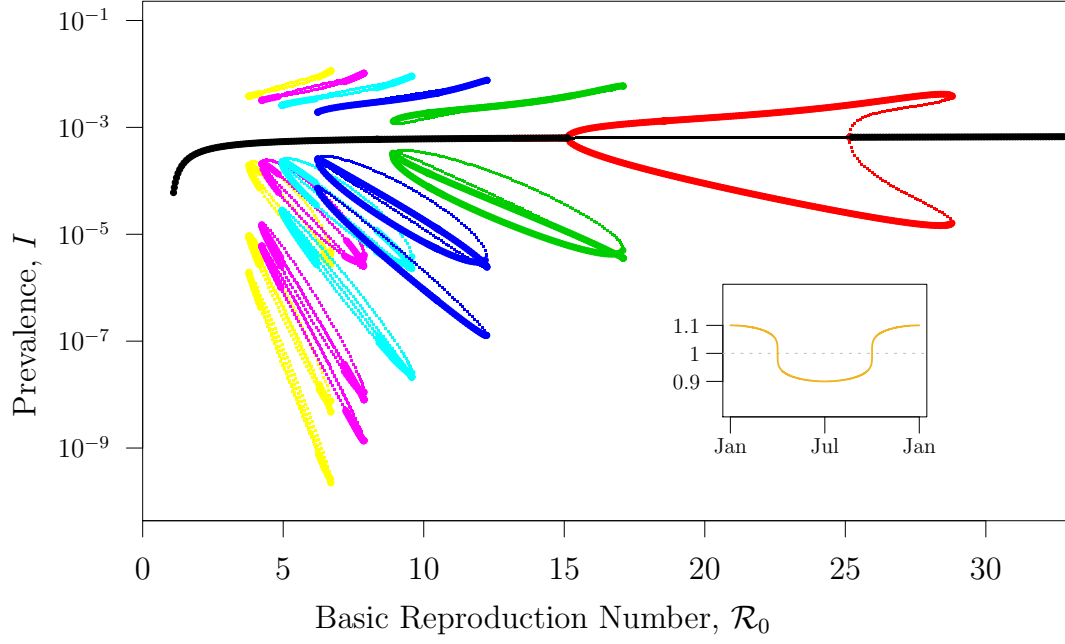


Figure S14:  $\mathcal{R}_0$  bifurcation diagram for  $(p, \alpha) = (0.25, 0.084)$ .

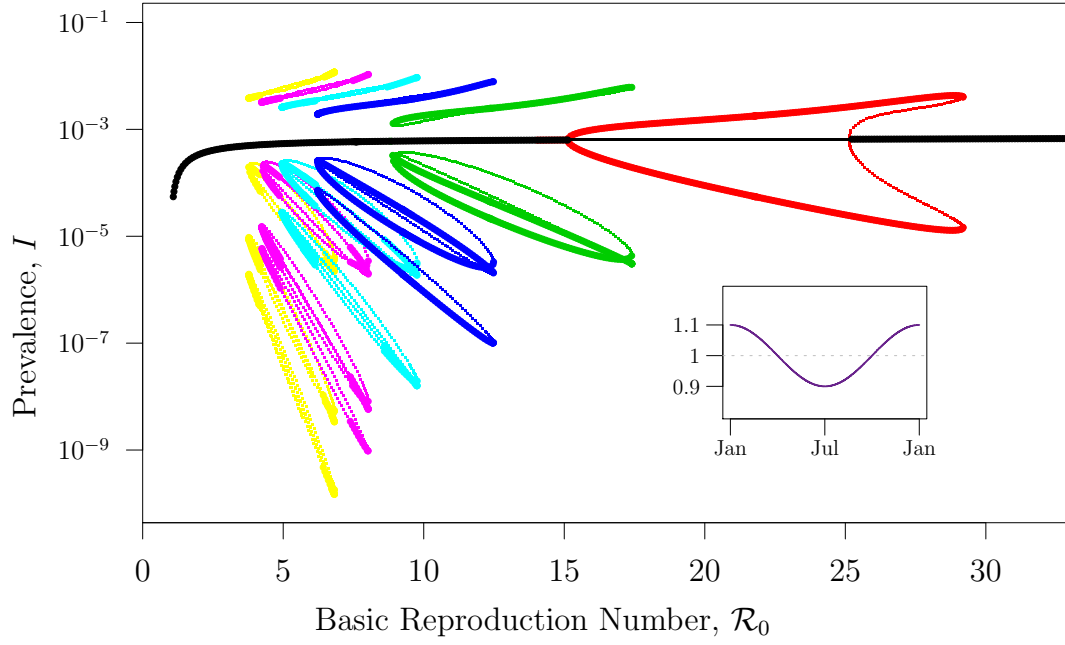


Figure S15:  $\mathcal{R}_0$  bifurcation diagram for  $(p, \alpha) = (1, 0.1)$ .

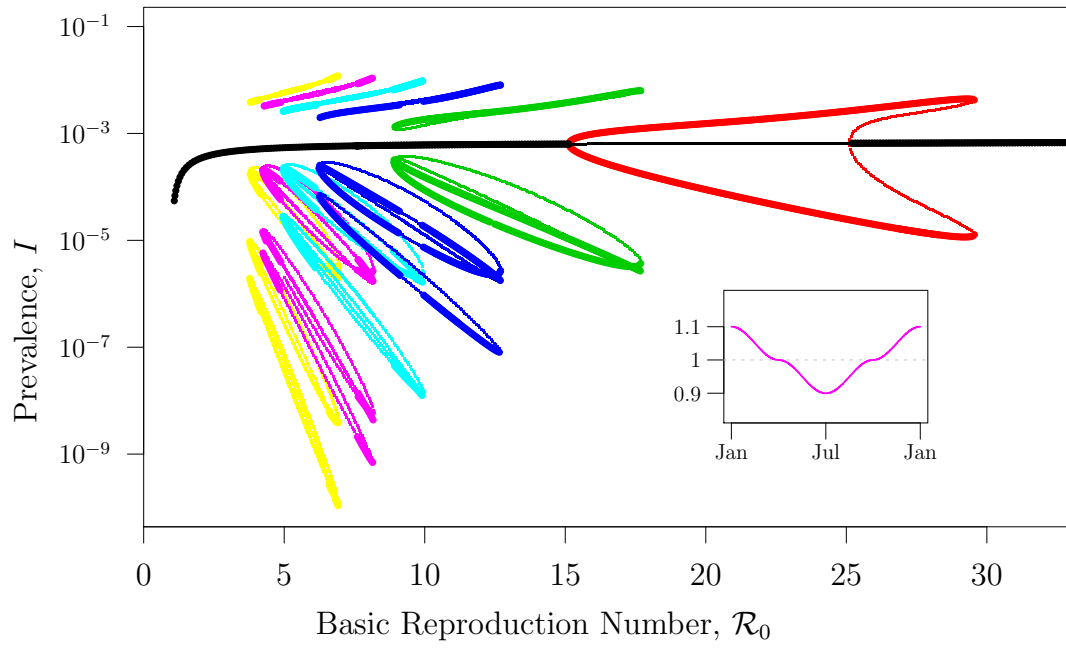



Figure S16:  $\mathcal{R}_0$  bifurcation diagram for  $(p, \alpha) = (2, 0.1118)$ .

## S5 Supplementary Code

### S5.1 Technical specifications

The code used for this project was prepared most recently on a Macintosh computer running:

- Mac OS X version 10.13.6
- XPPAUT version 7.0
-  version 3.4.4 (2018-03-15) – “Someone to Lean On”
- Apple LLVM version 10.0.0 (clang-1000.11.45.2)

### S5.2 Sample code for generating brute force $\mathcal{R}_0$ bifurcation diagrams

This .ode file can be run silently in xppaut from the command line using the following command:

```
xppaut bruteforce_forcedSIR_p-1a025.ode -silent

#####
## bruteforce_forcedSIR_p-1a025.ode
## Author: Irena Papst
## Based on XPPAUT Guide from Krylova and Earn (2013)
##
## Brute force bifurcation diagram for the seasonally forced
## SIR model using a family of seasonal forcing functions
## (with shape parameter p)
## ranging from term-time forcing (p=-1)
## to sinusoidal forcing (p=1)
## with a square wave in between (p=0)
#####

#####
## OUTPUT ##
#####

## filename for output to be saved
@ output=bruteforce_forcedSIR_p-1a025.dat

#####
## PARAMETERS ##
#####

## For the Forcing Function
# p determines shape of forcing function
# where p=-1 is ttf, p=0 is the square wave, p=1 is sinusoidal
```

```

par p=-1

## For the ODEs
# mean lifetime 1/mu = 50 years
# mean infectious period 1/gamma = 13 days = 0.0356 years
# amplitude of seasonality alpha = 0.25
par mu=0.02, gamma=28.08, Rzero=17, a=0.25

#####
## FORCING FUNCTIONS ##
#####

## PART 1. TERM TIME FORCING TO SQUARE WAVE
#####

## Define endpoints of breaks for term-time forcing
#####

# Number of breaks in ttf
numbreaks=5

# Easter Break
b1=99
e1=114

# Summer Break
b2=195
e2=250

# Autumn Break
b3=299
e3=306

# Christmas Break, Part 1
b4=355
e4=365

# Christmas Break, Part 2
b5=0
e5=5

## Calculate widths and centers for breaks &
## define waves for each break
#####

# Formulas

```

```

# Width of Break
w(b,e)=(e-b)/365

# Center of Break
c(b,e)=(b+e)/(2*365)

# Break Wave
break(t,p,wid,cent)=sign(abs(mod(t,1)-cent)+p*wid/2)

# Easter Break
w1=w(b1,e1)
c1=c(b1,e1)
eas(t,p)=break(t,p,w1,c1)

# Summer Break
wSum=w(b2,e2)
cSum=c(b2,e2)

# Autumn Break
w3=w(b3,e3)
c3=c(b3,e3)
aut(t,p)=break(t,p,w3,c3)

# Christmas Break, Part 1
w4=w(b4,e4)
c4=c(b4,e4)
chr1(t,p)=break(t,p,w4,c4)

# Christmas Break, Part 2
w5=w(b5,e5)
c5=c(b5,e5)
chr2(t,p)=break(t,p,w5,c5)

# Square Wave Summer Break
wSqu=0.5
cSqu=0.5
squsum(t,p)=sign(abs(mod(t,1)-(abs(p)*cSum+(1-abs(p))*cSqu))
               -((abs(p))*wSum+(1-abs(p))*wSqu)/2)

# Term Time to Square Wave

ttftosqu(t,p)=sign(eas(t,p)+aut(t,p)+chr1(t,p)+chr2(t,p)+squsum(t
,p)-(numbreaks-1))
# need to shift sum of all breaks down by numbreak-1 to ensure
the sum
# of waves is centred about the x-axis before taking the sign of
them all

```

```

## PART 2. SQUARE WAVE TO COSINE
#####

sqtocos(t,p)=sign(cos(2*pi*t))*abs(cos(2*pi*t))^p

## PART 3. TTF TO COSINE SHAPE FUNCTION
#####

trans(t,p)=(1-heav(p))*ttftosqu(t,p)+(heav(p))*sqtocos(t,p)

## Shift and Scale to get beta(t,p)
#####

# Calculate the proportion of the year spent in school
# (used in shifting the forcing functions to ensure that
# the average value of each forcing function is always
# beta0 = mean beta)
ps(p)=1-abs(p)*(w1+wSum+w3+w4+w5)-(1-abs(p))*wSqu

# Family of Forcing Functions
beta0=Rzero*(gamma+mu)
beta=beta0*(1+a*((1-heav(p))*(trans(t,p)+(1-2*ps(p)))+(heav(p))*(
    trans(t,p))))

#####
## DIFFERENTIAL EQUATIONS ##
#####
s' = mu - beta*s*i - mu*s
i' = beta*s*i - (gamma+mu)*i

#####
## INITIAL CONDITIONS ##
#####

init S=0.9, I=0.001

#####
## AUXILIARY VARIABLES ##
#####

aux R0=Rzero
aux log10s=log10(s)
aux log10i=log10(i)

#####
## PLOT OPTIONS ##

```

```
#####

## xp=variable on x axis, yp=variable on y axis
@ xp=R0, yp=log10i

## limits on plot
@ xlo=0, xhi=40, yhi=0, ylo=-25

## background colour for plot
@ back=white

#####
## POINCARÉ MAP SET UP ##
#####

@ poimap=section, poivar=t, poipln=1

#####
## BIF DIAG SET UP ##
#####

@ range=1, rangeover=Rzero, rangestep=1500
@ rangelow=0, rangehigh=30, rangereset=no

#####
## INTEGRATION OPTIONS ##
#####

## total time of integration
@ total=650

## transient time
@ trans=600

## time step for integration
@ dt=0.001

#####
## STORAGE and DATA SAVING OPTIONS ##
#####

## max number of time points to store (default 5000)
@ maxstor=2000000

done
```



### S5.3 Script to extract AUTO initial conditions from brute force bifurcation data

The following script takes brute force simulation data and generates an `.ode` file for computing full bifurcation diagrams in AUTO. The script first scans the brute force data for equilibria of each period present, and extracts initial conditions and other parameter values, pasting them into xppaut-formatted parameter sets. The script then writes an `.ode` file ready to use in xppaut. In order to write this file, the script must be able to access the remaining file parts (that stay the same between parameter sets), included here below the main script.

#### Main script:

```
#####  
## ExtractAutoIC.R  
## Author: Irena Papst  
## Based on XPPAUT Guide from Krylova and Earn (2013)  
##  
## Read a brute force bifurcation diagram output file and  
## for each period that occurs, save a single point to be  
## used as an initial condition in AUTO  
#####  
  
## Set where data is coming from (either "bruteforce" or "xpp  
  ")  
## bruteforce: initial bruteforce simulations  
## xpp: converged initial conditions exported from xppaut  
type <- "bruteforce"  
# type <- "xpp"  
  
## Set arguments bruteforce data function  
this_date <- "180821"  
datdir <- paste0("../..//bruteforce/dat-", this_date)  
bfdatfile_prefix <- "bf_mm_run"  
odefile_prefix <- paste0(this_date, "-run")  
jobslist <- 1 # empty string does all jobs;  
# otherwise provide list of job numbers  
bifrange <- c(1,10) # no trimming, by default  
# bifrange <- c(10,15)  
  
## Set argument for xpp data function  
# jobno <- 1  
# brnos <- c(1,3)  
# params <- data.frame(p=1,alpha=0.101717, delta=0.75, eps  
  =0.0007117438)  
  
## Get p-alpha pairs
```

```

# i <- 1 # desired row number in p-alpha pairs array
# load("../bf_alpha/run13/p_alpha_pairs.Rdata")
# params <- data.frame(p=palpha.pairs$p[i], alpha=palpha.
  pairs$alpha[i], delta=0.75, eps=0.0007117438)

#####
## MAIN FUNCTIONS ##
#####

## GENERATE ODE FILE FOR USE WITH AUTO FROM BRUTEFORCE
  SIMULATION DATA

extractAutoIC_bf <- function(datdir,
                             bfdatfile_prefix, odefile_prefix
                             ,
                             this_date, bifrange=c(),
                             jobslist="", maxper=6){

  ## datdir: directory containing the .dat files
  ## and the .Rdata file (path relative to location of this
    script)
  ## omit trailing "/"

  ## bfdatfile_prefix: filename prefix for the bruteforce .
    dat files
  ## (everything before the run number)

  ## odefile_prefix: filename prefix for the .ode files that
    are being generated
  ## for use with AUTO
  ## (everything before the run number)

  ## this_date: date in yymmdd-x from data directory
  ## (where -x is optional for multiple jobs on the same date
    ; x is 1
  ## for the second job, 2 for the third, etc)

  ## bifrange: range in bifurcation parameter for which to
    extract ICs;
  ## should be a list of length two: c(bifmin, bifmax)
  ## an empty list means we look at the entire range of the
    bif par from
  ## simulations

  ## jobslist: (optional) list of integers denoting job

```

```

    numbers for which to create .ode file
## by default, if no jobslist is provided,
## loop through all jobs found in in parameter .Rdata file

## maxper: largest period for which to generate an IC set

#####

## Load parameters for each run (job info)
load(paste0(datdir, "/jobsLegend.Rdata"))

if (jobslist==""){ ## if no specific jobslist is provided,
  loop through all of them
  jobslist <- jobs$jobno
} ## otherwise, use jobslist passed as argument

## loop through the job numbers
for (jobno in jobslist){

  ## LOAD DATA

  ## Get parameter values
  params <- jobs[jobno,]

  ## Generate filename for this particular brute force .dat
  file
  file <- paste0(datdir, "/", bfdatfile_prefix, jobno, ".
  dat")

  ## Read data
  bfd <- read.table(file,
                    col.names=c("time", "s", "i", "rho", "
                    log10s", "log10i"))

  ## If bifrane is given as nonempty, trim bfd to only
  ## include given range of the bifurcation parameter
  if (length(bifrane>0)){
    bfd <- bfd[(bfd$rho>bifrane[1])&(bfd$rho<bifrane[2])
    ,]
  }

  ## EXTRACT INITIAL CONDITIONS

  ## Calculate period of attractor for each rho
  bfd.last <- last.point.with.period(bfd, max.period=maxper

```

```

)

## Extract all periods that are not NA:
all.periods <- with(bfd.last, period[!is.na(period)])
(unique.periods <- unique(all.periods))

nper <- length(unique.periods)

## Create data frame for list of initial conditions:
ic.set <- bfd.last[1,]

## For each period that occurs, save a final condition:
for (iper in 1:nper) {
  ## select the rows in the data frame with this period:
  df.iper <- subset(bfd.last, period==unique.periods[iper
])
  ## choose a random initial condition:
  ic.set[iper,] <- df.iper[round(sample(1:nrow(df.iper),
1)),]
}

## Replace original row names with ic number:
row.names(ic.set) <- 1:nrow(ic.set)

## Save IC data in an xppaut-ready format (as loadable
parameter sets)
icsets <- vector(mode="character", length=maxper)

for(i in 1:maxper)
{
  icsets[i] <- paste0("set p", i, " {init s=", ic.set[
which(ic.set$period==i),"log10s"], ", init i=", ic.
set[which(ic.set$period==i),"log10i"], ", rho=", ic.
set[which(ic.set$period==i),"rho"], ", p=", params$p
, ", alpha=", params$alpha, ", delta=", params$delta
, ", eps=", params$eps, ", r=", params$r, ", nout=",
i, "}")
}

## CREATE FILES AND DIRECTORIES

## Write .ode file
odepath <- paste0(this_date, "/")
odefilename <- paste0(odefile_prefix, jobno)
writeODE(odepath, odefilename, icsets)

```

```

## Save jobs info with corresponding ode file
save(params, file = paste0(odepath, odefilename, ".Rdata"
))

## Make directory to store AUTO allinfo data
# cmd <- paste0("mkdir -p ", odepath, "allinfo")
# system(cmd)

## Make directory to store converged solution data
## for use with extractAutoIC_xpp
# cmd <- paste0("mkdir -p ", odepath, "/convergedIC")
# system(cmd)
}
}

## REGENERATE ODE FILE FOR USE WITH AUTO USING IC FROM
CONVERGED SOLUTION DATA

extractAutoIC_xpp <- function(jobno, odefile_prefix,
  branchlist, params=c()){

  ## jobno: run number for which we want to regenerate ICs
  ## from converged xppaut data

  ## odefile_prefix: filename prefix for the existing .ode
  ## files used with Auto
  ## where we want to replace the ICs with converged data
  ## (everything before the run number)

  ## branchlist: list of branches for which we have converged
  ## solution data

  ## params: named list of parameters (p, alpha, delta, eps)
  ## to use to generate
  ## initial condition sets

  ## Generate filepath (location of existing .ode file, .
  ## Rdata file, and convergedIC data)
  filepath <- paste0("run", jobno, "/")

  ## Load parameters for this run (unless they are provided)
  if (length(params)==0){
    load(paste0(filepath, odefile_prefix, jobno, ".Rdata"))
  }
}

```

```

## Save data in an xppaut-ready format (as parameter sets)
icsets <- vector(mode="character", length=length(branchlist))

j <- 0
for (i in branchlist){
  j <- j+1
  if (length(params)==0){
    this.filename <- paste0("convergedIC_p1a0.1/br",i,"-IC.
      dat")
  } else {
    this.filename <- paste0("convergedIC_p", params$p, "a",
      params$alpha, "/br",i,"-IC.dat")
  }
  xppd <- as.data.frame(read.table(paste0(filepath,this.
    filename),
                                col.names = c("time", "
                                  log10s", "log10i", "
                                  rho")))
  ## Grab initial conditions from first row of xpp data
  icsets[i] <- paste0("set p", j, " {init s=", xppd[1,"
    log10s"], ", init i=", xppd[1,"log10i"], ", rho=",
    xppd[1,"rho"], ", p=", params$p, ", alpha=", params$
    alpha, ", delta=", params$delta, ", eps=", params$eps,
    ", nout=", i, "}")
}

## Write .ode file
odepath <- paste0("run", jobno, "/")
odefilename <- paste0(odefile_prefix, jobno)
writeODE(odepath, odefilename, icsets)
}

#####
## SUPPORTING FUNCTIONS ##
#####

## Function to calculate period of attractor for each rho
last.point.with.period <- function(df, dop=5, rholim=4, max.
  period=7) {
  ## data frame with only the last pt on each soln:
  df.last <- subset(df, time==max(time))
  nrho <- nrow(df.last) # number of rho values
  df.last$period <- rep(0,nrho) # add period column

```

```

for (i in 1:nrho) {
  ## data frame with all pts on soln with given rho:
  rhoi <- df.last[i,"rho"]
  df.rhoi <- subset(df,rho==rhoi)
  ## compute period of this solution:
  if (rhoi < rholim) {
    period <- 1
  } else {
    period <- length(unique(round(df.rhoi[, "log10i"], dop)))
    if (period > max.period) period <- NA
  }
  df.last$period[i] <- period
}
return(df.last)
}

## Function to write custom .ode file for use with AUTO
writeODE <- function(odepath, odefilename, parsets){
  ## odepath: path for ode file (include trailing "/")
  ## odefilename: name of resulting .ode file
  ## parsets: sets of parameters to write
  ## (stored as character vector with each entry to be
    written as a new line)

  ## Create directory to store new ode file
  cmd <- paste0("mkdir -p ", odepath)
  system(cmd)

  ## Create all blank ode file
  newfile <- paste0(odepath, odefilename, ".ode")
  file.create(newfile)

  ## Append first part of generic ode file
  file.append(newfile, "fileparts/AUTO_mm_part1.ode")

  ## Write extracted initial conditions
  write(parsets, file=newfile, sep="\n", append=TRUE)

  ## Append second part of generic ode file
  file.append(newfile, "fileparts/AUTO_mm_part2.ode")
}

#####
## EXTRACT AUTO IC AND GENERATE ODE FILE ##
#####

```

```

if (type=="bruteforce"){
  extractAutoIC_bf(datdir, bfdatfile_prefix, odefile_prefix,
    this_date, bifrange, jobslist)
} else if (type=="xpp"){
  extractAutoIC_xpp(jobno, odefile_prefix, brnos, params)
}

```

.ode file part 1 (header):

```

#####
## ODE file for Poincare map of seasonally forced SIR model
## (used with AUTO_mm.c library)
## Used to create bifurcation diagrams with respect to rho_0 in
  AUTO
## Author: Irena Papst
## Based on XPPAUT Guide from Krylova and Earn (2013)
#####

## DEFINE LEFT HAND-SIDE
s' = sp
i' = ip
sp = 0
ip = 0

## LINK TO THE C-LIBRARY
## i.e. pass the values {s, i, rho, p, alpha, delta, eps, r}
## and ask it to return {sp, ip}
## Note that the order of export must agree with
## the order of in[] and out[] in arrays in the C function
export {s, i, rho, p, alpha, delta, eps, r} {sp, ip}

## define a library to be used and a corresponding function
@ dll_lib=../AUTO_mm.so dll_fun=modelmap

## SET INITIAL CONDITIONS for each periodic orbit
## Note that, for convenience, s and i below
## are really log(s) and log(i)
## nOut sets the number of iterations of the map
## it should be changed for each periodic orbit depending on the
  period

```



.ode file part 2 (footer):

```
## PARAMETER VALUES
## Note the order here determines the "main" parameter for AUTO
## rho = avg transmission/interaction rate
## p = power of |cos(2*pi*t)| in our modified seasonal forcing
    function,
## which transforms the square wave into the cosine wave as p
    varies between 0 and 1
## alpha = amplitude of seasonal forcing
## delta = model selection parameter
## eps = birth rate
## r = logistic growth rate
par rho=26, p=1, alpha=0.1, delta=1, eps=0.0007, r=3
aux rhozero=rho

## XPP SETUP
## this is a discrete map not an ODE
@ meth=discrete
## total=60 means 60 iterations of the map in total
@ total=60, yp=i
## line type = dots
@ lt=0
## plotting options
@ xlo=-1, xhi=61, ylo=-9, yhi=-1

## AUTO SETUP
## set range for rho, our control parameter:
@ parmin=1.1, parmax=70
## set range of vertical axis variable and set which variable it
    is:
@ autoymax=-12,autoymax=-1,autovar=i
## set horizontal axis plot range:
@ autoxmin=0, autoxmax=70
## set step size for continuation of the control parameter:
## (here, ds=standard step size, others are max and min step size
    )
## (the sign of ds controls the direction of continuation)

## STANDARD TIME STEPS:
@ dsmax=0.01, ds=0.003, dsmin=0.0000003
## MORE PRECISE TIME STEPS: @ dsmax=0.001, ds=0.00003, dsmin
    =0.000000003
## SLIGHTLY MORE PRECISE TIME STEPS: @ dsmax=0.001, ds=0.003,
    dsmin=0.0000003

## set a few other technical aspects of the continuation:
```

```

@ Nmax=20000, Npr=2000, epsl=1e-6, epsu=1e-6, epss=1e-4
## above:
## Nmax = maximum number of steps to take along a branch before
    stopping
## Npr = number of steps before labelling a point (which can help
    with
##    continuing from points without having to start everything
    all over)
## eps... = various tolerances

done

```

## S5.4 Sample code for generating AUTO $\mathcal{R}_0$ bifurcation diagrams in xppaut

In order to generate  $\mathcal{R}_0$  bifurcation diagrams in AUTO via xppaut, we first set up a C function that integrates the system numerically with a Poincaré map. This function is stored in a C library that has been created by compiling the following code:

```

/**
Poincare map of the seasonally forced SIR model
to be called from XPPAUT for bifurcation analysis.

Author: Irena Papst
Based on XPPAUT Guide from Krylova and Earn (2013)

Under MacOSX, compile this function via:
    gcc -dynamiclib -m32 -o SIRmap_p.so SIRmap_p.c
***/

#include <math.h>

/**/ Compile-time definitions ***/

#define Time_step 0.0005 /* in units of years */
#define DAYS_PER_YEAR 365
#define TWO_PI 6.2831853071795864769252867665590
#define Real double
#define NDIM 2 /* dimension of the dynamical system */

/***** FUNCTIONS CALLED BY THE MAIN ROUTINE *****/

/**/ Euler integrator ***/

```

```

void Euler(Real *x, Real *dx, Real dt, int ndim ) {
    int i;
    for (i=0; i<ndim; i++) {
        x[i] = x[i] + dx[i]*dt;
    }
}

// Define the signum function -- used in the squaretocos
// transformation

Real sgn(Real val){
    return(val<=0 ? -1: 1);
}

// Define the unit step i.e. heaviside function -- used as a
// switch
// between the ttftosquare and squaretocos transformations

Real heav(Real val){
    return(val<0 ? 0: 1);
}

/*****/
/** Define the Seasonal_beta family of forcing functions */
/*****/

Real Seasonal_beta(Real beta0, Real alpha, Real p, Real time)
{
    /*****/
    /** Part 1. Term Time Forcing to Square Wave */
    /*****/

    // define term begin and end days (where t=0 is Jan 1)
    // order in each array:
    // christmas 1, easter, summer, autumn, christmas 2

    Real b[5] = {0, 99, 195, 299, 355};
    Real e[5] = {5, 114, 250, 306, 365};

    // calculate centre and width of each break
    // to be used in the break function
    // (indicator function which is -1 during a break and 1
    // otherwise)

    // easter

```

```

Real cEas;
Real wEas;

cEas = (b[1]+e[1])/(2*DAYS_PER_YEAR);
wEas = (e[1]-b[1])/DAYS_PER_YEAR;

// summer

Real cSumr;
Real wSumr;

cSumr = (b[2]+e[2])/(2*DAYS_PER_YEAR);
wSumr = (e[2]-b[2])/DAYS_PER_YEAR;

// autumn

Real cAut;
Real wAut;

cAut = (b[3]+e[3])/(2*DAYS_PER_YEAR);
wAut = (e[3]-b[3])/DAYS_PER_YEAR;

// christmas 1

Real wChr1;

wChr1 = (e[0]-b[0])/DAYS_PER_YEAR;

// christmas 2

Real wChr2;

wChr2 = (e[4]-b[4])/DAYS_PER_YEAR;

/* since the christmas break has to be defined in two parts
   due to
   the modular arithmetic in time that makes this wave
   periodic,
   we take the centre as t=0(mod1) and make both parts of
   the break shrink
   and eventually vanish at this point. thus, we do not
   calculate
   the centre of this break like we do for the other breaks
   */

```

```

// define width and centre of square wave break

Real wSqu=0.5;
Real cSqu=0.5;

/* calculate the beginning and end points of each break
   depending on p */

/* as p increases from -1 to 0
   summer break widens and shifts to give the square wave
   break
   while all other breaks shrink and vanish */

Real easB = cEas-fabs(p)*(wEas/2);
Real easE = cEas+fabs(p)*(wEas/2);

Real sumB = fabs(p)*(b[2]/DAYS_PER_YEAR)+(1-fabs(p))*0.25;
Real sumE = fabs(p)*(e[2]/DAYS_PER_YEAR)+(1-fabs(p))*0.75;

Real autB = cAut-fabs(p)*(wAut/2);
Real autE = cAut+fabs(p)*(wAut/2);

Real chrB = 1-fabs(p)*(DAYS_PER_YEAR-b[4])/DAYS_PER_YEAR;
Real chrE = fabs(p)*e[0]/DAYS_PER_YEAR;

/* use modular arithmetic to make the waves periodic (with
   period of 1 year) */

Real modtime = fmod(time,1); /* to make the ttftosquare
   function easier to read */

modtime = (0 <= modtime ? modtime : modtime+1);

/* define the ttftosquare indicator function that depends
   on p*/

Real ttftosquare;

ttftosquare = (((0 <= modtime && modtime < chrE) || /*
   christmas break, part 1 */
               (easB <= modtime && modtime <= easE) || /*
   easter break */
               (sumB <= modtime && modtime <= sumE) || /*
   summer break */

```

```

        (autB <= modtime && modtime <= autE) || /*
            autumn break */
        (chrB < modtime && modtime <= 1) /*
            christmas break, part 2 */
    )? -1: 1);

    /*****
    ** Part 2. Square Wave to Cosine **
    *****/

    Real c2pt;
    Real squaretocos;

    c2pt = cos(TWO_PI*time);
    squaretocos = sgn(c2pt)*pow(fabs(c2pt),p);

    /*****
    ** Part 3. TTF to Cos Shape Transformation Function **
    *****/

    Real trans; /* use a "switch" to change between the two
        transformations defined above as p goes from -1 to 1 */

    trans=(1-heav(p))*ttftosquare+(heav(p))*squaretocos;

    Real ps; /* proportion of year spent in school, used in
        shifting of forcing function to ensure that average
        value of the forcing function is always beta0 = mean
        beta */

    ps = 1-fabs(p)*(wEas+wAut+wChr1+wChr2+wSumr)-(1-fabs(p))*
        wSqu;

    Real seasbeta; /* full transformation from ttft to cos,
        shifted and scaled using model parameters */

    seasbeta = beta0*(1+alpha*((1-heav(p))*(trans+(1-2*ps))+
        heav(p)*trans));

    return(seasbeta);
}

    /*****
    *** THE MAIN ROUTINE ***
    *****/

```

```

/*****/

/** The function SIRmap is what XPPAUT calls */
void SIRmap(Real *in, Real *out, int nin,
            int nout, Real *var, Real *con)
/*
in = initial and parameter values we get from the ode file
    (s0, i0, R0, alpha, gamma, mu, p)
out = what we are returning (sp,ip):
    calculated values of S and I after one year
nin = dimension of in[]
nout = dimension of out[]
var and con are arrays that Bard said to include...
*/
{
    /* define starting values in log base 10 */
    Real s=in[0], i=in[1];
    Real x[NDIM], dx[NDIM]; /* for Euler integrator */

    /* converting back to the original values, not in log */
    s=pow (10,s);
    i=pow (10,i);

    /* define parameter values */
    Real R0=in[2], alpha=in[3], gamma=in[4], mu=in[5], p=in[6]
        ;
    Real ds, di;
    Real beta0, nonlin_term;
    Real time; /* in units of years */
    long istep, nsteps;

    /* number of steps in a year */
    nsteps = (int)( 1/Time_step + 0.5 );

    /* integrating for one year */
    for (istep=0; istep < nsteps; istep++) {
        time=(Real)(istep)*Time_step;

        /* compute the vector field */
        /* FIXME: this would be better as a called function */
        beta0 = R0*(gamma+mu); /* mean transmission rate */
        nonlin_term = Seasonal_beta(beta0,alpha,p,time) * s * i;
        ds = mu - nonlin_term - mu*s;
        di = nonlin_term - (mu + gamma)*i;
    }
}

```

```

    /* integrate using euler's method */
    /* FIXME: there should be a compile-time option to use
       RK4 instead */
    x[0] = s; x[1] = i; dx[0]=ds; dx[1]=di;
    Euler(x, dx, Time_step, NDIM);
    s = x[0]; i = x[1];
}
s = log10(s);
i = log10(i);
out[0] = s; /* output in log_10 */
out[1] = i;
}

```

This C library is then compiled on the command line using the command

```
gcc -dynamiclib -m32 -o SIRmap_p.so SIRmap_p.c
```

and is then referenced in an .ode file that is used to generate AUTO  $\mathcal{R}_0$  bifurcation data, such as the following:

```

#####
## ODE file for Poincare map of seasonally forced SIR model
## (used with SIRmap_p.c library)
## Used to create bifurcation diagrams with respect to R0 in AUTO
## Author: Irena Papst
## Based on XPPAUT Guide from Krylova and Earn (2013)
#####

## DEFINE LEFT HAND-SIDE
s' = sp
i' = ip
sp = 0
ip = 0

## LINK TO THE C-LIBRARY
## i.e. pass the values {s0, i0, R0, alpha, gamma, mu, p
   eventually}
## and ask it to return {sp, ip}
## Note that the order of export must agree with
## the order of in[] and out[] in arrays in the C function
export {s, i, R0, alpha, gamma, mu, p} {sp, ip}

## define a library to be used and a corresponding function
@ dll_lib=SIRmap_p.so dll_fun=SIRmap

## SET INITIAL CONDITIONS for each periodic orbit
## Note that, for convenience, s and i below
## are really log(s) and log(i)
## nOut sets the number of iterations of the map

```



```

## it should be changed for each periodic orbit depending on the
    period

set p1 {init s=-0.87659162, init i=-3.2304842, R0=7.5999999, p=2,
    alpha=0.118239, nout=1}
set p2 {init s=-1.2681592, init i=-2.7434311, R0=21.780001, p=2,
    alpha=0.118239, nout=1}
set p3 {init s=-1.0293249, init i=-2.6030924, R0=12.36, p=2,
    alpha=0.118239, nout=1}
set p4 {init s=-0.94534254, init i=-5.2238641, R0=10.305, p=2,
    alpha= 0.118239, nout=1}
set p5 {init s=-0.70523274, init i=-4.2141695, R0=5.7199998, p=2,
    alpha=0.118239, nout=1}
set p6 {init s=-0.70315689, init i=-3.7062016, R0=4.3800001, p=2,
    alpha=0.118239, nout=1}
set p7 {init s=-0.97648102, init i=-5.562336, R0=6.7350001, p=2,
    alpha=0.118239, nout=1}

## PARAMETER VALUES
## Note the order here determines the "main" parameter for AUTO
## alpha = amplitude of seasonal forcing
## R0 = basic reproduction number
## gamma = recovery rate; 1/gamma = mean infectious period
## mu = mean death/birth rate; 1/mu = average life time
## p = power of |cos(2*pi*t)| in our modified seasonal forcing
    function,
## which transforms the square wave into the cosine wave as p
    varies between 0 and 1
par R0=26, alpha=0.25, gamma=28.076923, mu=0.02, p=-1
aux Rzero=R0

## XPP SETUP
## this is a discrete map not an ODE
@ meth=discrete
## total=20 mean 20 iterations of the map in total
@ total=20, yp=i
## line type = dots
@ lt=0
## plotting options
@ xlo=-1, xhi=21, ylo=-9, yhi=-1

## AUTO SETUP
## set range for R0, our control parameter:
@ parmin=1.1, parmax=40
## set range of vertical axis variable and set which variable it
    is:
@ autoymin=-9,autoymax=-1,autovar=i

```

```

## set horizontal axis plot range:
@ autxmin=0, autxmax=30
## set step size for continuation of the control parameter:
## (here, ds=standard step size, others are max and min step size
  )
## (the sign of ds controls the direction of continuation)
## STANDARD TIME STEPS:
@ dsmax=0.1, ds=0.003, dsmin=0.0000003
## MORE PRECISE TIME STEPS: @ dsmax=0.001, ds=0.00003, dsmin
  =0.000000003
## SLIGHTLY MORE PRECISE TIME STEPS:
## @ dsmax=0.001, ds=0.003, dsmin=0.0000003
## set a few other technical aspects of the continuation:
@ Nmax=20000, Npr=2000, epsl=1e-6, epsu=1e-6, sepss=1e-4
## above:
## Nmax = maximum number of steps to take along a branch before
  stopping
## Npr = number of steps before labelling a point (which can help
  with
##   continuing from points without having to start everything
  all over)
## eps... = various tolerances
done

```

## S5.5 Sample code for $\alpha$ continuation in AUTO

The sample code from §S5.4 can be easily adapted to run a continuation in  $\alpha$  in order to create two parameter bifurcation diagrams in  $\alpha$  and  $p$ . Namely, we must change the order of arguments input to the C library, such that  $\alpha$  is the first parameter, instead of  $\mathcal{R}_0$ . Here, we include sample code reflecting this change. The same goes for the `xppaut` code calling the compiled C library:  $\alpha$  must be made the primary bifurcation parameter by being the first parameter listed in the array exported to the C library (as well as being the first parameter defined in the `xppaut` code).

This code generates the C library, followed by the corresponding `xppaut` code:

```

/**
Poincare map of the seasonally forced SIR model
to be called from XPPAUT for bifurcation analysis.

Author: Irena Papst
Based on XPPAUT Guide from Krylova and Earn (2013)

Under MacOSX, compile this function via:
    gcc -dynamiclib -m32 -o SIRmap_alpha.so
    SIRmap_alpha.c
***/

```

```

#include <math.h>

/** Compile-time definitions */

#define Time_step 0.0005 /* in units of years */
#define DAYS_PER_YEAR 365
#define TWO_PI 6.2831853071795864769252867665590
#define Real double
#define NDIM 2 /* dimension of the dynamical system */

*****
/** FUNCTIONS CALLED BY THE MAIN ROUTINE */
*****

/** Euler integrator */
void Euler(Real *x, Real *dx, Real dt, int ndim ) {
    int i;
    for (i=0; i<ndim; i++) {
        x[i] = x[i] + dx[i]*dt;
    }
}

// Define the signum function -- used in the squaretocos
transformation

Real sgn(Real val){
    return(val<=0 ? -1: 1);
}

// Define the unit step i.e. heaviside function -- used as a
switch
// between the ttftosquare and squaretocos transformations

Real heav(Real val){
    return(val<0 ? 0: 1);
}

*****
/** Define the Seasonal_beta family of forcing functions */
*****

Real Seasonal_beta(Real beta0, Real alpha, Real p, Real time)
{
    *****

```

```

/** Part 1. Term Time Forcing to Square Wave */
/** ***** */

// define term begin and end days (where t=0 is Jan 1)
// order in each array:
// christmas 1, easter, summer, autumn, christmas 2

Real b[5] = {0, 99, 195, 299, 355};
Real e[5] = {5, 114, 250, 306, 365};

// calculate centre and width of each break
// to be used in the break function
// (indicator function which is -1 during a break and 1
otherwise)

// easter

Real cEas;
Real wEas;

cEas = (b[1]+e[1])/(2*DAYS_PER_YEAR);
wEas = (e[1]-b[1])/DAYS_PER_YEAR;

// summer

Real cSumr;
Real wSumr;

cSumr = (b[2]+e[2])/(2*DAYS_PER_YEAR);
wSumr = (e[2]-b[2])/DAYS_PER_YEAR;

// autumn

Real cAut;
Real wAut;

cAut = (b[3]+e[3])/(2*DAYS_PER_YEAR);
wAut = (e[3]-b[3])/DAYS_PER_YEAR;

// christmas 1

Real wChr1;

wChr1 = (e[0]-b[0])/DAYS_PER_YEAR;

```

```

// christmas 2

Real wChr2;

wChr2 = (e[4]-b[4])/DAYS_PER_YEAR;

/* since the christmas break has to be defined in two parts
   due to
   the modular arithmetic in time that makes this wave
   periodic,
   we take the centre as t=0(mod1) and make both parts of
   the break shrink
   and eventually vanish at this point. thus, we do not
   calculate
   the centre of this break like we do for the other breaks
   */

// define width and centre of square wave break

Real wSqu=0.5;
Real cSqu=0.5;

/* calculate the beginning and end points of each break
   depending on p */

/* as p increases from -1 to 0
   summer break widens and shifts to give the square wave
   break
   while all other breaks shrink and vanish */

Real easB = cEas-fabs(p)*(wEas/2);
Real easE = cEas+fabs(p)*(wEas/2);

Real sumB = fabs(p)*(b[2]/DAYS_PER_YEAR)+(1-fabs(p))*0.25;
Real sumE = fabs(p)*(e[2]/DAYS_PER_YEAR)+(1-fabs(p))*0.75;

Real autB = cAut-fabs(p)*(wAut/2);
Real autE = cAut+fabs(p)*(wAut/2);

Real chrB = 1-fabs(p)*(DAYS_PER_YEAR-b[4])/DAYS_PER_YEAR;
Real chrE = fabs(p)*e[0]/DAYS_PER_YEAR;

/* use modular arithmetic to make the waves periodic (with
   period of 1 year) */

```

```

Real modtime = fmod(time,1); /* to make the ttftosquare
    function easier to read */

modtime = (0 <= modtime ? modtime : modtime+1);

/* define the ttftosquare indicator function that depends
    on p*/

Real ttftosquare;

ttftosquare = (((0 <= modtime && modtime < chrE) || /*
    christmas break, part 1 */
    (easB <= modtime && modtime <= easE) || /*
    easter break */
    (sumB <= modtime && modtime <= sumE) || /*
    summer break */
    (autB <= modtime && modtime <= autE) || /*
    autumn break */
    (chrB < modtime && modtime <= 1) /*
    christmas break, part 2 */
    )? -1: 1);

*****/
** Part 2. Square Wave to Cosine **/
*****/

Real c2pt;
Real squaretocos;

c2pt = cos(TWO_PI*time);
squaretocos = sgn(c2pt)*pow(fabs(c2pt),p);

*****/
** Part 3. TTF to Cos Shape Transformation Function **/
*****/

Real trans; /* use a "switch" to change between the two
    transformations defined above as p goes from -1 to 1 */

trans=(1-heav(p))*ttftosquare+(heav(p))*squaretocos;

Real ps; /* proportion of year spent in school, used in
    shifting of forcing function to ensure that average
    value of the forcing function is always beta0 = mean
    beta */

```

```

ps = 1-fabs(p)*(wEas+wAut+wChr1+wChr2+wSumr)-(1-fabs(p))*
    wSqu;

Real seasbeta; /* full transformation from ttj to cos,
    shifted and scaled using model parameters */

seasbeta = beta0*(1+alpha*((1-heav(p))*(trans+(1-2*ps))+
    heav(p)*trans));

return(seasbeta);
}

/*****
*** THE MAIN ROUTINE ***
*****/

/** The function SIRmap is what XPPAUT calls */
void SIRmap(Real *in, Real *out, int nin,
    int nout, Real *var, Real *con)
/*
in = initial and parameter values we get from the ode file
    (s0, i0, alpha, p, R0, gamma, mu)
out = what we are returning (sp,ip):
    calculated values of S and I after one year
nin = dimension of in[]
nout = dimension of out[]
var and con are arrays that Bard said to include...
*/
{
    /* define starting values in log base 10 */
    Real s=in[0], i=in[1];
    Real x[NDIM], dx[NDIM]; /* for Euler integrator */

    /* converting back to the original values, not in log */
    s=pow (10,s);
    i=pow (10,i);

    /* define parameter values */
    Real alpha=in[2], p=in[3], R0=in[4], gamma=in[5], mu=in[6]
    ;
    Real ds, di;
    Real beta0, nonlin_term;
    Real time; /* in units of years */

```

```

long istep, nsteps;

/* number of steps in a year */
nsteps = (int)( 1/Time_step + 0.5 );

/* integrating for one year */
for (istep=0; istep < nsteps; istep++) {
    time=(Real)(istep)*Time_step;

    /* compute the vector field */
    /* FIXME: this would be better as a called function */
    beta0 = R0*(gamma+mu); /* mean transmission rate */
    nonlin_term = Seasonal_beta(beta0,alpha,p,time) * s * i;
    ds = mu - nonlin_term - mu*s;
    di = nonlin_term - (mu + gamma)*i;

    /* integrate using euler's method */
    /* FIXME: there should be a compile-time option to use
       RK4 instead */
    x[0] = s; x[1] = i; dx[0]=ds; dx[1]=di;
    Euler(x, dx, Time_step, NDIM);
    s = x[0]; i = x[1];
}
s = log10(s);
i = log10(i);
out[0] = s; /* output in log_10 */
out[1] = i;
}

```

```

#####
## ODE file for Poincare map of seasonally forced SIR model
## (used in conjunction with SIRmap_alpha.c library)
## Used to create bifurcation diagrams wrt alpha in AUTO
## Author: Irena Papst
## Based on XPPAUT Guide from Krylova and Earn (2014)
#####

## DEFINE LEFT HAND-SIDE
s' = sp
i' = ip
sp = 0
ip = 0

## LINK TO THE C-LIBRARY
## i.e. pass the values {s0, i0, alpha, p, R0, gamma, mu}
## and ask it to return {sp, ip}

```



```

## Note that the order of export must agree with
## the order of in[] and out[] in arrays in the C function
export {s, i, alpha, p, R0, gamma, mu} {sp, ip}

## define a library to be used and a corresponding function
@ dll_lib=SIRmap_alpha.so dll_fun=SIRmap

## SET INITIAL CONDITIONS for each periodic orbit
## Note that, for convenience, s and i below
## are really log(s) and log(i)
## nOut sets the number of iterations of the map
## it should be changed for each periodic orbit depending on the
    period

## Branch 1, Bifurcation 1
set p1 {init s=-1.17372, init i=-3.37877, R0=15.1198, p=-1, alpha
    =0.25, nout=1}

## PARAMETER VALUES
## Note the order here determines the "main" parameter for AUTO
## alpha = amplitude of seasonal forcing
## R0 = basic reproduction number
## gamma = recovery rate; 1/gamma = mean infectious period
## mu = mean death/birth rate; 1/mu = average life time
## p = power of |cos(2*pi*t)| in our modified seasonal forcing
    function,
## which transforms the square wave into the cosine wave as p
    varies between 0 and 1
par alpha=0.08, p=1, R0=21.9863, gamma=28.076923, mu=0.02
aux a=alpha

## XPP SETUP
## this is a discrete map not an ODE
@ meth=discrete
## total=20 mean 20 iterations of the map in total
@ total=20, yp=i
## line type = dots
@ lt=0
## plotting options
@ xlo=-1, xhi=21, ylo=-9, yhi=-1

## AUTO SETUP
## set range for alpha, our control parameter:
@ parmin=0, parmax=1
## set range of vertical axis variable and set which variable it
    is:
@ autoymax=-9,autoymin=-1,autovar=i

```

```

## set horizontal axis plot range:
@ autxmin=0, autxmax=1
## set step size for continuation of the control parameter:
## (here, ds=standard step size, others are max and min step size
  )
## (the sign of ds controls the direction of continuation)
## for p=-1 (term time forcing)
@ dsmax=0.5, ds=0.055, dsmin=0.00003
## set a few other technical aspects of the continuation:
@ Nmax=20000, Npr=2000, epsl=1e-6, epsu=1e-6, sepss=1e-4
## above:
## Nmax = maximum number of steps to take along a branch before
  stopping
## Npr = number of steps before labelling a point (which can help
  with
##   continuing from points without having to start everything
  all over)
## eps... = various tolerances
done

```

## References

- [1] Bauch CT, Earn DJD. Interepidemic intervals in forced and unforced SEIR models. In: Ruan S, Wolkowicz G, Wu J, editors. *Dynamical Systems and Their Applications in Biology*. vol. 36 of Fields Institute Communications. Toronto: American Mathematical Society; 2003. p. 33–44.
- [2] Ermentrout B. *Simulating, analyzing, and animating dynamical systems: a guide to XPPAUT for researchers and students. Software, Environments, and Tools*. Philadelphia: Society for Industrial and Applied Mathematics; 2002.
- [3] Krylova O, Earn DJD. Effects of the infectious period distribution on predicted transitions in childhood disease dynamics. *J. R. Soc. Interface*. 2013;10:20130098.
- [4] Strogatz SH. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*. CRC Press; 2018.
- [5] Bauch CT, Earn DJD. Transients and attractors in epidemics. *Proc. R. Soc. Lond. B*. 2003;270(1524):1573–1578.
- [6] Nahvi M. *Signals and Systems*. 1st ed. New York: McGraw-Hill; 2014.
- [7] Olver FWJ, Lozier DW, Boisvert RF, Clark CW, editors. *NIST Handbook of Mathematical Functions*. New York: National Institute of Standards and Technology (NIST) and Cambridge University Press; 2010.
- [8] Rinaldi S, Muratori S, Kuznetsov Y. Multiple attractors, catastrophes and chaos in seasonally perturbed predator-prey communities. *Bull Math Biol*. 1993;55(1):15–35.
- [9] Gagnani A, Rinaldi S. A universal bifurcation diagram for seasonally perturbed predator-prey models. *Bull Math Biol*. 1995;57(5):701–712.
- [10] Scheffer M, Rinaldi S, Kuznetsov YA, van Nes EH. Seasonal dynamics of *Daphnia* and algae explained as a periodically forced predator-prey system. *Oikos*. 1997;p. 519–532.
- [11] Kuznetsov YA, Muratori S, Rinaldi S. Bifurcations and chaos in a periodic predator-prey model. *Int J Bifurc Chaos*. 1992;2(01):117–128.
- [12] Kuznetsov YA. *Elements of applied bifurcation theory*. vol. 112 of Applied Mathematical Sciences. 3rd ed. New York: Springer-Verlag; 2004.
- [13] Doedel EJ, Oldeman BE. *AUTO-07P: Continuation and bifurcation software for ordinary differential equations*. Montreal, Canada: Concordia University; 2011.