1 Differential Methods for Assessing Sensitivity in

2 Biological Models

3 Rachel Mester¹, Alfonso Landeros¹, Chris Rackauckas^{4,5}, Kenneth Lange^{1,2,3}

Departments of Computational Medicine¹,

Human Genetics², and Statistics³

University of California Los Angeles, Los Angeles, CA 90095

Massachusetts Institute of Technology⁴ and Pumas-AI⁵

Phone: 310-206-8076

E-mail klange@ucla.edu

4 October 18, 2021

5 Abstract

6 Differential sensitivity analysis is indispensable in fitting parameters, un-
7 derstanding uncertainty, and forecasting the results of both thought and
8 lab experiments. Although there are many methods currently available for
9 performing differential sensitivity analysis of biological models, it can be
10 difficult to determine which method is best suited for a particular model.
11 In this paper, we explain a variety of differential sensitivity methods and
12 assess their value in some typical biological models. First, we explain the
13 mathematical basis for three numerical methods: adjoint sensitivity analy-
14 sis, complex-perturbation sensitivity analysis, and forward-mode sensitivity
15 analysis. We then carry out four instructive case studies. (i) The CARRGO
16 model for tumor-immune interaction highlights the additional information
17 that differential sensitivity analysis provides beyond traditional naive sensi-
18 tivity methods, (ii) the deterministic SIR model demonstrates the value of
19 using second-order sensitivity in refining model predictions, (iii) the stochas-
20 tic SIR model shows how differential sensitivity can be attacked in stochastic
21 modeling, and (iv) a discrete birth-death-migration model illustrates how
22 the complex-perturbation method of differential sensitivity can be gener-
23 alized to a broader range of biological models. Finally, we compare the
24 speed, accuracy, and ease of use of these methods. We find that forward-
25 mode automatic differentiation has the quickest computation time, while the
26 complex-perturbation method is the simplest to implement and the most
27 generalizable.

28 **Author Summary**

29 Over the past few decades, mathematical modeling has become an indis-
 30 pensable tool in the biologist’s toolbox. From deterministic to stochastic
 31 to statistical models, computational modeling is ubiquitous in almost ev-
 32 ery field of biology. Because model parameter estimates are often noisy
 33 or depend on poorly understood interactions, it is crucial to examine how
 34 both quantitative and qualitative predictions change as parameter estimates
 35 change, especially as the number of parameters increase. Sensitivity anal-
 36 ysis is the process of understanding how a model’s behavior depends on
 37 parameter values. Sensitivity analysis simultaneously quantifies prediction
 38 certainty and clarifies the underlying biological mechanisms that drive com-
 39 putational models. While sensitivity analysis is universally recognized to be
 40 an important step in modeling, it is often unclear how to best leverage the
 41 available differential sensitivity methods. In this manuscript we explain and
 42 compare various differential sensitivity methods in the hope that best prac-
 43 tices will be widely adopted. In particular, we stress the relative advantages
 44 of existing software and their limitations. We also present a new numerical
 45 technique for computing differential sensitivity.

46 **1 Introduction**

47 In many mathematical models underlying parameters are poorly specified.
 48 This problem is particularly acute in biological and biomedical models.
 49 Model predictions can have profound implications for scientific understand-
 50 ing, further experimentation, and even public policy decisions. For instance,
 51 in an epidemic some model parameters can be tweaked by societal or sci-
 52 entific interventions to drive infection levels down. Differential sensitivity

53 can inform medical judgement about the steps to take with the greatest im-
54 pact at the least cost. Similar considerations apply in economic modeling.
55 Finally, model fitting usually involves maximum likelihood or least squares
56 criteria. These optimization techniques depend heavily on gradients and
57 and Hessians with respect to parameters.

58 In any case it is imperative to know how sensitive model predictions are
59 to changes in parameter values. Unfortunately, assessment of model sen-
60 sitivity can be time consuming, computationally intensive, inaccurate, and
61 simply confusing. Most models are nonlinear and resistant to exact math-
62 ematical analysis. Understanding their behavior is only approachable by
63 solving differential equations or intensive and noisy simulations. Sensitivity
64 analysis is often conducted over an entire bundle of neighboring parameters
65 to capture interactions. If the parameter space is large or high-dimensional,
66 it is often unclear how to choose representative points from this bundle.
67 Faced with this dilemma, it is common for modelers to fall back on varying
68 just one or two parameters at a time. Model predictions also often take
69 the form of time trajectories. In this setting, sensitivity analysis is based
70 on lower and upper trajectories bounding the behavior of the dynamical
71 system.

72 The differential sensitivity of a model quantity is measured by its gra-
73 dient with respect to the underlying parameters at their estimated values.
74 Given the gradient of a function $f(\beta)$, it is possible to approximate $f(\beta)$ by
75 the linear function $f(\beta_0) + \nabla f(\beta_0)^t(\beta - \beta_0)$ for all β near β_0 . The existing
76 literature on differential sensitivity is summarized in the modern references
77 [1, 2]. The Julia software DifferentialEquations.jl [3] makes sensitivity anal-
78 ysis fairly routine for many problems. Although the physical sciences have
79 widely adopted the method of differential sensitivity [4, 5], the papers and

80 software generally focus on a single sensitivity analysis method rather than a
81 comparison of the various approaches. For example, PESTO [6] is a current
82 Matlab toolbox for parameter estimation that uses adjoint sensitivities im-
83 plemented as part of the CVODES method from SUNDIALS [7]. Should the
84 continuous sensitivity equations be used, or would direct automatic differ-
85 entiation of the solvers be more efficient on biological models? On the types
86 of models biologists generally explore, would implicit parallelism within the
87 sensitivity equations be beneficial, or would the overhead cost of thread
88 spawning overrule any benefits? How close do simpler methods based on
89 complex step differentiation get to these techniques? The purpose of the
90 current paper is to explore these questions on a variety of models of interest
91 to computational biologists.

92 In the current paper we also suggest a more accurate method of approx-
93 imating gradients for models involving analytic functions without discon-
94 tinuities, maxima, minima, absolute values or any other excursion outside
95 the universe of analytic functions. By definition an analytic function can
96 be expanded in a locally convergent power series around every point of its
97 domain. In the sections immediately following, we summarize known the-
98 ory, including the important adjoint method for computing the sensitivity
99 of functions of solutions [4, 5]. Then we illustrate sensitivity analysis for
100 a few deterministic models and a few stochastic models. Our exposition
101 includes some straightforward Julia code that readers can adapt to their
102 own sensitivity needs. These examples are followed by an evaluation of the
103 accuracy and speed of the suggested numerical methods. The concluding
104 discussion summarizes our experience, indicates limitations of the methods,
105 and suggests new potential applications.

106 For the record, here are some notational conventions used throughout the

paper. All functions that we differentiate have real or real vector arguments and real or real vector values. All vectors and matrices appear in boldface. The superscript t indicates a vector or matrix transpose. For a smooth real-valued function $f(\mathbf{x})$, we write its gradient (column vector of partial derivatives) as $\nabla f(\mathbf{x})$ and its differential (row vector of partial derivatives) as $df(\mathbf{x}) = \nabla f(\mathbf{x})^t$. If $g(\mathbf{x})$ is vector-valued with i th component $g_i(\mathbf{x})$, then the differential (Jacobi matrix) $dg(\mathbf{x})$ has i th row $dg_i(\mathbf{x})$. The chain rule is expressed as the equality $d[f \circ g(\mathbf{x})] = df[g(\mathbf{x})]dg(\mathbf{x})$ of differentials. The transpose (adjoint) form of the chain rule is $\nabla f \circ g(\mathbf{x}) = dg(\mathbf{x})^t \nabla f[g(\mathbf{x})]$. For a twice differentiable function, the second differential (Hessian matrix) $d^2 f(\mathbf{x}) = d \nabla f(\mathbf{x})$ is the differential of the gradient. Finally, i will denote $\sqrt{-1}$.

2 Methods for Computing Sensitivity

The simplest dynamical models are governed by the linear constant coefficient differential equation $\frac{d}{dt}\mathbf{x}(t) = \mathbf{A}(\boldsymbol{\beta})\mathbf{x}(t)$ with solution $\mathbf{x}(t) = e^{t\mathbf{A}(\boldsymbol{\beta})}\mathbf{x}_0$. The directional derivative of the matrix exponential $e^{\mathbf{B}}$ in the direction $\mathbf{V}$ can be represented by the integral

$$d_{\mathbf{V}}e^{\mathbf{B}} = \int_0^1 e^{s\mathbf{B}}\mathbf{V}e^{(1-s)\mathbf{B}}ds.$$

A proof of this fact appears in Example 3.2.2 of reference [8]. Setting $\mathbf{B} = t\mathbf{A}(\boldsymbol{\beta})$ and applying the chain rule leads to the partial derivative

$$\begin{aligned} \frac{\partial}{\partial \beta_j} e^{t\mathbf{A}(\boldsymbol{\beta})}\mathbf{x}(0) &= \int_0^1 e^{st\mathbf{A}(\boldsymbol{\beta})}t \frac{\partial}{\partial \beta_j} \mathbf{A}(\boldsymbol{\beta}) e^{(1-s)t\mathbf{A}(\boldsymbol{\beta})} ds \mathbf{x}(0) \\ &= \int_0^t e^{s\mathbf{A}(\boldsymbol{\beta})} \frac{\partial}{\partial \beta_j} \mathbf{A}(\boldsymbol{\beta}) e^{(t-s)\mathbf{A}(\boldsymbol{\beta})} ds \mathbf{x}(0), \end{aligned}$$

126 which can be laboriously evaluated by numerical integration. Simplification
127 into a sum of exponentials is possible if $\mathbf{A}(\boldsymbol{\beta})$ is uniformly diagonalizable
128 across all $\boldsymbol{\beta}$ [9], but the details are messy.

129 In practice, it is simpler to differentiate the original ODE with respect
130 to β_j , interchange the order of differentiation, and numerically integrate the
131 system

$$\frac{d}{dt} \frac{\partial}{\partial \beta_j} \mathbf{x}(t, \boldsymbol{\beta}) = \frac{\partial}{\partial \beta_j} \mathbf{A}(\boldsymbol{\beta}) \mathbf{x}(t, \boldsymbol{\beta}) + \mathbf{A}(\boldsymbol{\beta}) \frac{\partial}{\partial \beta_j} \mathbf{x}(t, \boldsymbol{\beta})$$

132 from 0 to some final value of t . The initial condition $\mathbf{x}(0, \boldsymbol{\beta}) = \mathbf{x}(0)$ remains
133 intact, and the new condition $\nabla_{\boldsymbol{\beta}} \mathbf{x}(0, \boldsymbol{\beta}) = 0$ is added. The second method
134 has the advantage of giving the sensitivity along the entire trajectory. An-
135 other huge advantage is that the second method generalizes to nonlinear
136 systems $\frac{d}{dt} \mathbf{x}(t, \boldsymbol{\beta}) = f[\mathbf{x}(t), \boldsymbol{\beta}]$ under the right circumstances. In particular
137 the sensitivity of this more elaborate system can be evaluated by solving the
138 differential equation

$$\frac{d}{dt} \frac{\partial}{\partial \beta_j} \mathbf{x}(t, \boldsymbol{\beta}) = \frac{\partial}{\partial \beta_j} f[\mathbf{x}(t), \boldsymbol{\beta}] + d_{\mathbf{x}} f[\mathbf{x}(t), \boldsymbol{\beta}] \frac{\partial}{\partial \beta_j} \mathbf{x}(t, \boldsymbol{\beta})$$

139 numerically. This sensitivity equation depends on knowing $\mathbf{x}(t, \boldsymbol{\beta})$. In prac-
140 tice, one solves the system

$$\frac{d}{dt} \begin{bmatrix} \mathbf{x}(t, \boldsymbol{\beta}) \\ \nabla_{\boldsymbol{\beta}} \mathbf{x}(t, \boldsymbol{\beta}) \end{bmatrix} = \begin{bmatrix} f[\mathbf{x}(t), \boldsymbol{\beta}] \\ \nabla_{\boldsymbol{\beta}} f[\mathbf{x}(t), \boldsymbol{\beta}] + d_{\boldsymbol{\beta}} \mathbf{x}(t, \boldsymbol{\beta})^t \nabla_{\mathbf{x}} f[\mathbf{x}(t), \boldsymbol{\beta}] \end{bmatrix} \quad (1)$$

141 jointly, where $d_{\boldsymbol{\beta}} \mathbf{x}[t, \boldsymbol{\beta}]$ is the Jacobi matrix of $\mathbf{x}(t, \boldsymbol{\beta})$ with respect to $\boldsymbol{\beta}$.
142 This is commonly referred to as forward sensitivity analysis and is carried
143 out by software suites such as DifferentialEquations.jl [3] and SUNDIALS

CVODES [7]. We note in passing that a common implementation detail of sensitivity analysis is to base calculations on directional derivatives. Thus, the directional derivative

$$d_{\beta} \mathbf{x}(t, \beta)^t \nabla_{\mathbf{x}} f[\mathbf{x}(t), \beta] = \lim_{\epsilon \rightarrow 0} \frac{f\{\mathbf{x}(t) + \epsilon \nabla_{\mathbf{x}} f[\mathbf{x}(t), \beta], \beta\} - f[\mathbf{x}(t), \beta]}{\epsilon}$$

version of the chain rule allows one to evolve dynamical systems without ever computing full Jacobians.

The well-known adjoint method, another variant of the sensitivity method, is incorporated in the biological parameter estimation software PESTO through CVODES [7]. The adjoint method [1, 2] is defined directly on a function $g[\mathbf{x}(\beta), \beta]$ of the solution of the ODE. The adjoint method first numerically solves the ODE system forward in time over $[t_0, t_n]$, then solves the system

$$d_{\beta} \lambda(\beta) = d_{\mathbf{x}} f[\mathbf{x}(\beta), \beta] \lambda(\beta) + d_{\beta} g[\mathbf{x}(\beta), \beta],$$

in reverse time, and finally computes derivatives via

$$d_{\beta} g[\mathbf{x}(\beta), \beta] = \int_{t_0}^{t_n} \lambda(t, \beta) d_{\beta} \mathbf{x}(t, \beta) dt.$$

The second and third stages are commonly combined by appending the last equation to the set of ODEs being solved in reverse. This tactic achieves a lower computational complexity than other techniques, which require solving an n -dimensional ODE system p times for p parameters. In contrast, the adjoint method solves an n -dimensional ODE forwards and then solves an n -dimensional and a p -dimensional system in reverse, changing the computational complexity from $\mathcal{O}(np)$ to $\mathcal{O}(n + p)$. Whether such asymptotic

cost advantages lead to more efficiency on practical models is precisely one of the points studied in this paper.

Alternatively, one can find the partial derivatives purely numerically. The obvious method here is to compute a slightly perturbed trajectory $\mathbf{x}(t, \boldsymbol{\beta} + \Delta \mathbf{v})$ and form the forward differences

$$\frac{\mathbf{x}(t, \boldsymbol{\beta} + \Delta \mathbf{v}) - \mathbf{x}(t, \boldsymbol{\beta})}{\Delta}$$

at all specified time points as approximations to the forward directional derivatives of $\mathbf{x}(t, \boldsymbol{\beta})$ in the direction $\mathbf{v}$. Choosing $\mathbf{v}$ to be unit vectors along each coordinate axis gives ordinary partial derivatives. The accuracy of this crude method suffers from round-off error in subtracting two nearly equal function values. These round-off errors are in addition to the usual errors committed in integrating the differential equation numerically. Round-off errors can be ameliorated by using central differences rather than forward differences, but this doubles computation time and still leaves the difficult choice of the stepsize Δ .

There is a far more accurate way when the function $f[\mathbf{x}, \boldsymbol{\beta}]$ defining the ODE is analytic in the parameter vector $\boldsymbol{\beta}$ [10]. This implies that the trajectory $\mathbf{x}(t, \boldsymbol{\beta})$ is also analytic in $\boldsymbol{\beta}$. For a real analytic function $g(\beta)$ of a single variable β , the derivative approximation

$$g'(\beta) = \frac{\text{Imag } g(\beta + \Delta i)}{\Delta} + O(\Delta^2) \quad (2)$$

in the complex plane avoids roundoff and is highly accurate for $\Delta > 0$ very small [11, 12]. Thus, in calculating a directional derivative of $\mathbf{x}(t, \boldsymbol{\beta})$, it suffices to (a) solve the governing ODE $\frac{d}{dt}\mathbf{x}(t, \boldsymbol{\beta}) = f[\mathbf{x}(t), \boldsymbol{\beta}]$ with $\boldsymbol{\beta} + \Delta i \mathbf{v}$

replacing β , (b) take the imaginary part of the result, and (c) divide by Δ . Of course to make these calculations feasible, the computer language implementing the calculations should support complex arithmetic and ideally have an automatic dispatching mechanism so that only one implementation of each function is required. In contrast to numerical integration of the joint system (1), the analytic method is embarrassingly parallelizable across parameters.

The following straightforward Julia routine for computing sensitivities

```
function differential(f::F, p, d) where F
    fvalue = real(f(p)) # function value
    df = zeros(length(fvalue), 0) # empty differential
    for j = 1:length(p)
        p[j] = p[j] + d * im # perturb parameter
        fj = f(p) # compute perturbed function value
        p[j] = complex(real(p[j]), 0.0) # reset parameter
        df = [df imag(fj) ./ d] # concatenate jth partial
    end
    return (fvalue, df) # return the differential
end
```

requires a function $f(\mathbf{p}) : \mathbb{R}^n \mapsto \mathbb{R}^m$ of a real vector $\mathbf{p}$ declared as complex. The perturbation scalar d should be small and real, say 10^{-10} to 10^{-12} in double precision. If the parameters p_j vary widely in magnitude, then a good heuristic is to perturb p_j by $p_j di$ instead of di . The returned value df is an $m \times n$ real matrix. The Julia commands `real` and `complex` effect conversions between real and complex numbers, and Julia substitutes `im` for $i = \sqrt{-1}$. We will employ these functions later in some case studies.

A recent extension [13] of the analytic method facilitates accurate ap-

211 proximation of second derivatives. The relevant formula is

$$\frac{\partial^2}{\partial \beta_j^2} g(\beta) = \frac{\text{Imag}[g(\beta + e^{\pi i/4} \Delta e_j) + g(\beta - e^{\pi i/4} \Delta e_j)]}{\Delta^2} + O(\Delta^4), \quad (3)$$

212 where $e^{\pi i/4} = (1 + i)/\sqrt{2}$. Roundoff errors can now occur, but are usually
 213 manageable. Recently we discovered how to extend the analytic method to
 214 approximate mixed partials. Our derivation is condensed into the following
 215 equations

$$\begin{aligned} g[\mathbf{x} + e^{\pi i/4} \Delta(\mathbf{e}_j + \mathbf{e}_k)] &\approx g(\mathbf{x}) + e^{\pi i/4} dg(\mathbf{x}) \Delta(\mathbf{e}_j + \mathbf{e}_k) \\ &\quad + \frac{i}{2} d^2 g(\mathbf{x}) [\Delta(\mathbf{e}_j + \mathbf{e}_k)]^2 \\ &\quad + \frac{e^{\pi 3/4}}{6} d^3 g(\mathbf{x}) [\Delta(\mathbf{e}_j + \mathbf{e}_k)]^3 \\ g[\mathbf{x} - e^{\pi i/4} \Delta(\mathbf{e}_j + \mathbf{e}_k)] &\approx g(\mathbf{x}) - e^{\pi i/4} dg(\mathbf{x}) \Delta(\mathbf{e}_j + \mathbf{e}_k) \\ &\quad + \frac{i}{2} d^2 g(\mathbf{x}) [\Delta(\mathbf{e}_j + \mathbf{e}_k)]^2 \\ &\quad - \frac{e^{\pi 3/4}}{6} d^3 g(\mathbf{x}) [\Delta(\mathbf{e}_j + \mathbf{e}_k)]^3. \end{aligned}$$

216 This approximation is accurate to order $O(\Delta^6)$ and allows us to infer that

$$\begin{aligned} \frac{\text{Imag}\{g[\mathbf{x} + e^{\pi i/4} \Delta(\mathbf{e}_j + \mathbf{e}_k)] + g[\mathbf{x} - e^{\pi i/4} \Delta(\mathbf{e}_j + \mathbf{e}_k)]\}}{\Delta^2} &= \\ d^2 g(\mathbf{x}) [(\mathbf{e}_j + \mathbf{e}_k)]^2 + O(\Delta^4) &= \quad (4) \\ \frac{\partial^2}{\partial \beta_j^2} g(\beta) + \frac{\partial^2}{\partial \beta_k^2} g(\beta) + 2 \frac{\partial^2}{\partial \beta_j \partial \beta_k} g(\beta) + O(\Delta^4) \end{aligned}$$

217 Since we can approximate $\frac{\partial^2}{\partial \beta_j^2} g(\beta)$ and $\frac{\partial^2}{\partial \beta_k^2} g(\beta)$, we can now approximate
 218 $\frac{\partial^2}{\partial \beta_j \partial \beta_k} g(\beta)$ to order $O(\Delta^4)$. These approximations are derived in the Ap-
 219 pendix.

220 The Julia code for computing second derivatives

```

221 function hessian(f::F, p, d) where F
222     d2f = zeros(length(p), length(p)) # hessian
223     dp = d * (1.0 + 1.0 * im) / sqrt(2)
224     for j = 1:length(p) # compute diagonal entries of d2f
225         p[j] = p[j] + dp
226         fplus = f(p)
227         p[j] = p[j] - 2 * dp
228         fminus = f(p)
229         p[j] = complex(real(p[j]), 0.0) # reset parameter
230         d2f[j, j] = imag(fplus + fminus) / d^2
231     end
232     for j = 2:length(p) # compute off diagonal entries
233         for k = 1:(j - 1)
234             (p[j], p[k]) = (p[j] + dp, p[k] + dp)
235             fplus = f(p)
236             (p[j], p[k]) = (p[j] - 2 * dp, p[k] - 2 * dp)
237             fminus = f(p)
238             (p[j], p[k]) = (complex(real(p[j]), 0.0), complex(real(p[k]), 0.0))
239             d2f[j, k] = imag(fplus + fminus) / d^2
240             d2f[j, k] = (d2f[j, k] - d2f[j, j] - d2f[k, k]) / 2
241             d2f[k, j] = d2f[j, k]
242         end
243     end
244     return d2f
245 end
246

```

247 operates on a scalar-valued function $f(u)$ of a real vector $\mathbf{p}$ declared as
248 complex. Because roundoff error is now a concern, the perturbation scalar
249 d should be in the range 10^{-3} to 10^{-6} in double precision. The returned

250 value d^2f is a symmetric matrix.

251 Another technique one can use to calculate the derivatives of model solu-
 252 tions is to differentiate the numerical algorithm that calculates the solution.
 253 This can be done with computational tools collectively known as automatic
 254 differentiation [CITE]. Forward mode automatic differentiation is performed
 255 by carrying forward the Jacobian-vector product at each successive calcula-
 256 tion. This is accomplished by defining higher-dimensional numbers, known
 257 as dual numbers [14], coupled to primitive functions $f(\mathbf{x})$ through the action

$$f(\mathbf{a} + \mathbf{b}\epsilon) = f(\mathbf{a}) + \epsilon df(\mathbf{a})\mathbf{b}.$$

258 Here ϵ is a dimensional marker, similar to the complex i , which is a two-
 259 dimensional number. For a composite function $f = f_2 \circ f_1$, the chain rule is
 260 $df(\mathbf{a})\mathbf{b} = df_2[f_1(\mathbf{a})][df_1(\mathbf{a})\mathbf{b}]$. The i th column of the Jacobian appears in the
 261 expression $f(\mathbf{x} + \mathbf{e}_i\epsilon) = f(\mathbf{x}) + \epsilon \nabla_i f(\mathbf{x})$. Since computational algorithms can
 262 be interpreted as the composition of simpler functions, one need only define
 263 the algorithm on a small set of base cases (such as $+$, $*$, $\sin$, and so forth,
 264 known as the primitives) and then apply the accepted rules in sequence to
 265 differentiate more elaborate functions. The ForwardDiff.jl package in Julia
 266 does this by defining dispatches for such primitives on a dual number type
 267 and provides convenience functions for easily extracting common objects
 268 like gradients, Jacobians, and Hessians. Hessians are calculated by layering
 269 automatic differentiation twice on the same algorithm to effectively take the
 270 derivative of a derivative.

271 In this form, forward mode automatic differentiation shares many sim-
 272 ilarities to the analytic methods described above without the requirement
 273 that the extension of $f(\mathbf{x})$ be complex analytic. Of course, at every stage of

the calculation $f(\mathbf{x})$ must be differentiable. The pros and cons of this weaker but still restrictive assumption will be discussed in the results section. Conveniently, automatic differentiation allows for arbitrarily many derivatives to be calculated simultaneously. By defining higher-dimensional dual numbers that act independently via $f(\mathbf{a} + \sum_i b_i \epsilon_i) = f(\mathbf{a}) + \sum \epsilon_i df(\mathbf{a}) \mathbf{b}_i$, one can calculate entire Jacobians in a single function call $f(\mathbf{a} + \sum_i \mathbf{e}_i \epsilon_i)$. This use of higher-dimensional dual numbers is a practice known as chunking. Chunking reduces the number of primal (non-derivative) calculations required for computing the Jacobian. Because the ForwardDiff.jl package uses chunking by default, we will investigate the extent to which this detail is applicable in biological models.

3 Case Studies

We now explore applications of differential sensitivity to a few core models in oncology and epidemiology.

3.1 CARRGO Model

The CARRGO model [15] was designed to capture the tumor-immune dynamics of CAR T-cell therapy in glioma. The CARRGO model generalizes to other tumor cell-immune cell interactions. Its governing system of ODEs

$$\begin{aligned}\frac{dx}{dt} &= \rho x \left(1 - \frac{y}{\gamma}\right) - \kappa_1 xy \\ \frac{dy}{dt} &= \kappa_2 xy - \theta y\end{aligned}$$

follows cancer cells x as prey and CAR T-Cells y as predators. This model captures Lotka-Volterra dynamics with logistic growth of the cancer cells.

Figure 1: Sensitivity of Cancer Cells in the CARRGO Model

Our numerical experiments assume the parameter values and initial conditions

$$\begin{aligned}\kappa_1 &= 6 \times 10^{-9}/(\text{day} \times \text{cell}), & \kappa_2 &= 3 \times 10^{-11}/(\text{day} \times \text{cell}), \\ \theta &= 1 \times 10^{-6}/\text{day}, & \rho &= 6 \times 10^{-2}/\text{day}, & \gamma &= 1 \times 10^9 \text{ cells}, \\ x_0 &= 1.25 \times 10^4 \text{ cells}, & y_0 &= 6.25 \times 10^2 \text{ cells}\end{aligned}$$

suggested by Sahoo et al. [15].

A traditional sensitivity analysis hinges on solving the system of ODEs for a variety of parameters and displaying the solution at a chosen future time across an interval or rectangle of parameter values. Figure 1 shows how $x(t)$ and $y(t)$ vary at $t = 1000$ days under joint changes of κ_1 and κ_2 , where κ_1 is the rate at which cancer cells are destroyed in an interaction with an immune cell, and κ_2 is the rate at which immune cells are recruited after such an interaction. This type of analysis directly portrays how a change in one or two parameters impacts the outcome of the system. As might be expected, the number of cancer cells $x(t)$ strongly depends on κ_1 but weakly on κ_2 . In contrast, the number of immune cells $y(t)$ depends comparably on both parameters, perhaps due to the fact that the initial population of

immune cells is much smaller than the initial population of cancer cells.

There are limitations to this type of sensitivity analysis. How many solution curves should be examined? What time is most informative in displaying system changes? Is it really necessary to compute sensitivity over such a large range of parameters when the trends are so clear? These ambiguities cloud our understanding and require far more computing than is necessary. Differential sensitivity successfully addresses these concerns. Gradients of solutions immediately yield approximate solutions in a neighborhood of postulated parameter values. The relative importance of different parameters in determining species levels is obvious from inspection of the gradient. Furthermore, modern software easily delivers the gradient along entire solution trajectories. There is no need to solve for an entire bundle of neighboring solutions.

Differential assessment is far more efficient. The required calculations involve solving an expanded system of ordinary differential equations just once under the partial derivative method or solving the system once for each parameter under the analytic method. Either way, the differential method is much less computationally intensive than the traditional method of solving the ODE system over an interval for each parameter or over a rectangle for each pair of parameters. Here is our brief Julia code for computing sensitivity via the analytic method.

```
using DifferentialEquations, Plots

function sensitivity(x0, p, d, tspan)
    problem = ODEProblem{true}(ODE, x0, tspan, p)
    sol = solve(problem, saveat = 1.0) # solve ODE
    (lp, ls, lx) = (length(p), length(sol), length(x0))
```



```

335     solution = Dict{Int, Any}(i => zeros(ls, lp + 1) for i in 1:lx)
336     for j = 1:lx # record solution for each species
337         @views solution[j][:, 1] = sol[j, :]
338     end
339     for j = 1:lp
340         p[j] = p[j] + d * im # perturb parameter
341         problem = ODEProblem{true}(ODE, x0, tspan, p)
342         sol = solve(problem, saveat = 1.0) # resolve ODE
343         p[j] = complex(real(p[j]), 0.0) # reset parameter
344         @views sol .= imag(sol) / d # compute partial
345         for k = 1:lx # record partial for each species
346             @views solution[k][:, j + 1] = sol[k, :]
347         end
348     end
349     return solution
350 end
351
352 function ODE(dx, x, p, t) # CARRGO model
353     dx[1] = p[4] * x[1] * (1 - x[1] / p[5]) - p[1] * x[1] * x[2]
354     dx[2] = p[2] * x[1] * x[2] - p[3] * x[2]
355 end
356
357 p = complex([6.0e-9, 3.0e-11, 1.0e-6, 6.0e-2, 1.0e9]); # parameters
358 x0 = complex([1.25e4, 6.25e2]); # initial values
359 (d, tspan) = (1.0e-13, (0.0, 1000.0)); # step size and time interval in days
360 solution = sensitivity(x0, p, d, tspan); # find solution and partials
361 CARRGO1 = plot(solution[1][:, 1], label = "x1", xlabel = "days",
362     ylabel = "cancer cells x1", xlims = (tspan[1], tspan[2]))
363 CARRGO2 = plot(solution[1][:, 2], label = "d1x1", xlabel = "days",
364     ylabel = "p1 sensitivity", xlims = (tspan[1], tspan[2]))

```

Figure 2: Sensitivity of Cancer Cells in the CARRGO Model

In the Julia code the parameters κ_1 , κ_2 , θ , ρ , and γ and the variables x and y exist as components of the vector $\mathbf{p}$ and $\mathbf{x}$, respectively. The two plot commands construct solution curves for cancer and its sensitivity to perturbations of κ_1 .

Figure 2 reinforces the conclusions drawn from the heatmaps, but more clearly and quantitatively. In brief, the scaled sensitivity of cancer cells $x(t)$ is much more dependent on carrying capacity γ than on birth rate ρ , and even more so than on the destruction rate κ_1 . Interestingly, the number of immune cells $y(t)$ is much more sensitive to the recruitment rate κ_2 than to the natural death rate θ .

3.2 Deterministic SIR Model

The deterministic SIR model follows the number of infectives $I(t)$, the number of susceptibles $S(t)$, and the number of recovered $R(t)$ during the course

378 of an epidemic. These three subpopulations satisfy the ODE system

$$\begin{aligned}\frac{d}{dt}S &= -\eta I \frac{S}{N} \\ \frac{d}{dt}I &= \eta I \frac{S}{N} - \delta I \\ \frac{d}{dt}R &= \delta I,\end{aligned}$$

379 where η is the daily infection rate per encounter and δ is the daily rate of
380 progression to immunity per person.

381 For SARS-CoV-2, current estimates [16] of η range from 0.0012 to 0.48,
382 and estimates of δ range from 0.0417 to 0.0588 [17]. As an alternative to
383 solving the extended set of differential equations, we again use the analytic
384 method to evaluate parameter sensitivities.

385 The following Julia code for the analytic method incorporates the sensi-
386 tivity function of the previous problem.

```
387 function ODE(dx, x, p, t) # Covid model
388     N = 3.4e8 # US population size
389     dx[1] = - p[1] * x[2] * x[1] / N
390     dx[2] = p[1] * x[2] * x[1] / N - p[2] * x[2]
391     dx[3] = p[2] * x[2]
392 end
393
394 p = complex([0.2, (0.0417 + 0.0588) / 2]); # parameters
395 x0 = complex([3.4e8, 100.0, 0.0]); # initial values
396 (d, tspan) = (1.0e-10, (0.0, 365.0)) # 365 days
397 solution = sensitivity(x0, p, d, tspan);
398 Covid = plot(solution[1][:, :], label = ["x1" "d1x1" "d2x1"],
399             xlabel = "days", xlims = (tspan[1], tspan[2]))
```

Figure 3: Sensitivities of Susceptibles in the Covid Model

Our parameter choices roughly capture the American experience. Figure 3 plots the susceptible curve and its sensitivities. In this case all three curves conveniently occur on comparable scales.

3.3 Second-Order Expansions of Solution Trajectories

In predicting nearby solution trajectories, the second-order Taylor expansion

$$f(\mathbf{x} + \mathbf{v}) \approx f(\mathbf{x}) + df(\mathbf{x})\mathbf{v} + \frac{1}{2}\mathbf{v}^t d^2 f(\mathbf{x})\mathbf{v}$$

obviously improves accuracy over the first-order expansion

$$f(\mathbf{x} + \mathbf{v}) \approx f(\mathbf{x}) + df(\mathbf{x})\mathbf{v}.$$

Figure 4 displays predicted trajectories for the SIR model when all parameters p_i are replaced by $p_i(1 + U_i)$, where each U_i is chosen uniformly from $(-0.25, 0.25)$. The figure vividly confirms the improvement in accuracy in passing from a first-order to a second-order approximation. The more non-linear the solution trajectory becomes, the more improvement is evident.

For example, the middle panel of Figure 4 shows that the solution trajectory of infected individuals bends dramatically with a change in parameters. This behavior is much better reflected in the second-order prediction compared to the first-order prediction. The Euclidean distance between the actual and predicted trajectories at the sampled time points is about 30.0 in the first-order case and only about 9.82 in the second-order case, a reduction of over 65% in prediction error. By contrast, the trajectory of the recovered individuals steadily increases in a nearly linear fashion. The bottom panel of Figure 4 shows that the first-order prediction now remains reasonably accurate over a substantial period of time. Even so, the discrepancy between the predicted solutions grows so that by day 100 the absolute difference between the first-order prediction and the actual trajectory exceeds 140, compared to about 39.0 for the second-order prediction. Thus, calculating second-order sensitivity is helpful in both highly non-linear systems and systems with long time scales.

3.4 Stochastic SIR Model

We now illustrate sensitivity calculations in the stochastic SIR model. This model postulates an original population of size n with i infectives and s susceptibles. The parameters δ and η again capture the rate of progression to immunity and the infection rate per encounter. Since extinction of the infectives is certain, it makes sense to focus on the time to elimination of the

Figure 4: Model Trajectories for SIR Model Calculated Using First and Second Differentials

infectives. It is also convenient to follow the vector (i, n) , where $n = i + s$ is the sum of the number of infectives i plus the number of susceptibles s . Elementary arguments now show that the mean time t_{in} to elimination of all infectives satisfies the recurrence

$$t_{in} = \frac{1}{i\delta + i\left(\frac{n-i}{N}\right)\eta} + \frac{i\delta}{i\delta + i\left(\frac{n-i}{N}\right)\eta} t_{i-1, n-1} + \frac{i\left(\frac{n-i}{N}\right)\eta}{i\delta + i\left(\frac{n-i}{N}\right)\eta} t_{i+1, n} \quad (5)$$

for $0 < i < n$ together with the boundary conditions

$$t_{ii} = \sum_{j=1}^i \frac{1}{j\delta} \quad \text{and} \quad t_{0n} = 0.$$

The expression for t_{ii} stems from adding the expected time for the $i \rightarrow i-1$ transition, plus the expected time $i-1 \rightarrow i-2$, and so forth. This system of equations can be solved recursively for $i = n, n-1, \dots, 0$ starting with $n = 1$. Once the values for a given n are available, n can be incremented, and a new round is initiated. Ultimately the target size $n = N$ is reached. Taking partial derivatives of the recurrence (5) yields a new system of recurrences that can also be solved recursively in tandem with the original recurrence. The analytic method is easier to implement and comparable in accuracy to the partial derivative method.

Another important index of the SIR process is the mean number of infectives m_{in} ever generated starting with i initial infectives and n total

448 people. These expectations can be calculated via the recurrences

$$m_{in} = \frac{i\delta}{i\delta + i\left(\frac{n-i}{N}\right)\eta}(m_{i-1,n-1} + 1) + \frac{i\left(\frac{n-i}{N}\right)\eta}{i\delta + i\left(\frac{n-i}{N}\right)\eta}m_{i+1,n} \quad (6)$$

449 for $0 < i < n$ together with the boundary conditions

$$m_{ii} = i \quad \text{and} \quad m_{0n} = 0.$$

450 Obviously, one can compute the sensitivities of the m_{in} to parameter per-
 451 turbations in almost exactly the same way as the t_{in} . Here is the Julia code
 452 for the two means and their sensitivities via the analytic method. Note how
 453 our earlier differential function plays a key role.

```

454 function SIRMeans(p)
455     (delta, eta) = (p[1], p[2])
456     M = Matrix{zeros(eltype(p), N + 1, N + 1)) # mean matrix
457     T = similar(M) # time to extinction matrix
458     for n = 1:N # recurrence relations loop
459         for j = 0:(n - 1)
460             i = n - j
461             a = i * delta # immunity rate
462             if i == n # initial conditions
463                 M[i + 1, n + 1] = i
464                 T[i + 1, n + 1] = T[i, i] + 1 / a
465             else
466                 b = i * (n - i) * eta / N # infection rate
467                 c = 1 / (a + b)
468                 M[i + 1, n + 1] = a * c * (M[i, n] + 1) + b * c * M[i + 2, n + 1]
469                 T[i + 1, n + 1] = c * (1 + a * T[i, n] + b * T[i + 2, n + 1])

```



```

470     end
471     end
472     end
473     return [M[:, N + 1]; T[:, N + 1]]
474 end
475
476 p = complex([0.2, (0.0417 + 0.0588) / 2]); # delta and beta
477 (N, d) = (100, 1.0e-10);
478 @time (f, df) = differential(SIRMeanTime, p, d);

```

Figure 5: Sensitivity of Stochastic SIR Model

479 The left column of Figure 5 displays the expected total number of individ-
 480 uals infected (top) and the expected number of days to extinction (bottom)
 481 for the stochastic SIR model based on the parameters mentioned in Section
 482 3.2. The middle column of Figure 5 depicts the derivatives of these quanti-
 483 ties with respect to η (the infection rate), and the right column depicts the
 484 derivatives with respect to δ (the recovery rate).

485 It is interesting to compare our method's results to estimates from stochas-
 486 tic simulations. To see the difference in accuracy, we calculated the aver-
 487 age number of individuals infected and the average time to extinction by

Table 1: SIR Model Outcomes

	Calculated Mean	Simulated Mean	Simulated Standard Error
Time to Extinction	2.792×10 days	3.074×10 days	4.153 days
Number Infected	5.484×10^3 people	5.838×10^3 people	8.551×10^2 people

Table 1 shows the comparison between the calculated and simulated means of SIR model outcomes with $N = 3.4 \times 10^4$ and $I_0 = 1$.

488 stochastic simulation using the software package Biosimulator.jl [18]. Table
 489 1 records the analytic and simulated means of these outcomes under the
 490 parameter values pertinent to Figure 3. As Table 1 indicates, the simulated
 491 means over $r = 100$ runs are roughly comparable to the analytic means, but
 492 the standard errors of the simulated means are large. Because the standard
 493 errors decrease as $\frac{1}{\sqrt{r}}$, it is difficult to achieve much accuracy by simulation
 494 alone. In more complicated models, simulation is so computationally in-
 495 tensive and time consuming that it is nearly impossible to achieve accurate
 496 results. Of course, the analytic method is predicated on the existence of an
 497 exact solution or an algorithm for computing the same.

498 Parameter sensitivities inform our judgment in interesting and helpful
 499 ways. For example, derivatives of both the total number of infecteds and
 500 the time to extinction with respect to η are very small except in a narrow
 501 window of the δ parameter. This suggests that we focus further simulations,
 502 sensitivity analysis, and possible interventions on the region of parameter
 503 space where δ is small and derivatives with respect to η tend to be large.
 504 Derivatives with respect to δ depend mostly on η except at very small values
 505 of δ . These conclusions are harder to draw from noisy simulations alone.

3.5 Branching Processes

Branching process models offer an opportunity for checking the accuracy of sensitivity calculations. For simplicity we focus on birth-death-migration processes [19]. These are multi-type continuous-time processes [9, 12] that can be used to model the early stages of an epidemic over a finite graph with n nodes, where nodes represent cities or countries. On node i we initiate a branching process with birth rate $\beta_i > 0$ and death rate $\delta_i > 0$. The migration rate from node i to node j is $\lambda_{ij} \geq 0$. All rates are per person, and each person is labeled by a node. Let $\lambda_i = \sum_{j \neq i} \lambda_{ij}$ be the sum of the migration rates emanating from node i . Given this notation, the mean infinitesimal generator of the process is the matrix

$$\mathbf{\Omega} = \begin{pmatrix} \beta_1 - \delta_1 - \lambda_1 & \lambda_{12} & \cdots & \lambda_{1,n-1} & \lambda_{1n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \lambda_{n1} & \lambda_{n2} & \cdots & \lambda_{n,n-1} & \beta_n - \delta_n - \lambda_n \end{pmatrix}$$

The entries of the matrix $e^{t\mathbf{\Omega}} = [m_{ij}(t)]$ represent the expected number of people at node j at time t starting from a single person of type i at time 0. The process is irreducible when the pure migration process corresponding to choice $\beta_i = \delta_i = 0$ for all i is irreducible. Henceforth, we assume the process is irreducible and let $\mathbf{\Gamma}$ denote the mean infinitesimal generator of the pure migration process. The process is subcritical, critical, or supercritical depending on whether the dominant eigenvalue ρ of $\mathbf{\Omega}$ is negative, zero, or positive.

To determine the local sensitivity of ρ to a parameter θ [9, 20], suppose its left and right eigenvectors $\mathbf{v}$ and $\mathbf{w}$ are normalized so that $\mathbf{vw} = 1$.

527 Differentiating the identity $\mathbf{\Omega}\mathbf{w} = \rho\mathbf{w}$ with respect to θ yields

$$\left(\frac{\partial}{\partial\theta}\mathbf{\Omega}\right)\mathbf{w} + \mathbf{\Omega}\frac{\partial}{\partial\theta}\mathbf{w} = \left(\frac{\partial}{\partial\theta}\rho\right)\mathbf{w} + \rho\frac{\partial}{\partial\theta}\mathbf{w}.$$

528 If we multiply this by $\mathbf{v}$ on the left and invoke the identities $\mathbf{v}\mathbf{\Omega} = \rho\mathbf{v}$ and

529 $\mathbf{v}\mathbf{w} = 1$ we find that

$$\frac{\partial}{\partial\theta}\rho = \mathbf{v}\left(\frac{\partial}{\partial\theta}\mathbf{\Omega}\right)\mathbf{w}. \quad (7)$$

530 Because $\frac{\partial}{\partial\delta_i}\mathbf{\Omega} = -\frac{\partial}{\partial\beta_i}\mathbf{\Omega}$, it follows that an increase in δ_i has the same impact

531 on ρ as the same decrease in β_i . The sensitivity of $\mathbf{v}$ and $\mathbf{w}$ can be determined

532 by an extension of this reasoning [22]. The extinction probabilities e_i of the

533 birth-death-migration satisfy the system of algebraic equations

$$e_i = \frac{\delta_i}{\beta_i + \delta_i + \lambda_i} + \frac{\beta_i}{\beta_i + \delta_i + \lambda_i}e_i^2 + \sum_{j \neq i} \frac{\lambda_{ij}}{\beta_i + \delta_i + \lambda_i}e_j \quad (8)$$

534 for all i . This is a special case of the vector extinction equation

$$\mathbf{e} = P(\mathbf{e}) = \begin{pmatrix} P_1(\mathbf{e}) \\ \vdots \\ P_n(\mathbf{e}) \end{pmatrix}$$

535 for a general branching process with offspring generating function $P_i(\mathbf{x})$ for

536 a type i person [21]. For a subcritical or critical process, $\mathbf{e} = \mathbf{1}$. For a

537 supercritical process all $e_i \in (0, 1)$. Iteration is the simplest way to find $\mathbf{e}$.

538 Starting from $\mathbf{e}_0 = \mathbf{0}$, the vector sequence $\mathbf{e}_n = P(\mathbf{e}_{n-1})$ satisfies

$$\mathbf{0} \leq \mathbf{e}_n \leq \mathbf{e}_{n+1} \leq \mathbf{e}$$

539 and converges to a solution of the extinction equations. Here all inequalities
540 apply component-wise.

541 To find the differential [22] of the extinction vector $\mathbf{e}$ with respect to a
542 vector $\boldsymbol{\theta}$ of parameters, we assume that the branching process is supercritical
543 and resort to implicit differentiation of the equation $\mathbf{e}(\boldsymbol{\theta}) = P[\mathbf{e}(\boldsymbol{\theta}), \boldsymbol{\theta}]$. The
544 chain rule gives

$$d_{\boldsymbol{\theta}}\mathbf{e} = d_{\mathbf{e}}P(\mathbf{e}, \boldsymbol{\theta})d_{\boldsymbol{\theta}}\mathbf{e} + d_{\boldsymbol{\theta}}P(\mathbf{e}, \boldsymbol{\theta}).$$

545 This equation has the obvious solution

$$d_{\boldsymbol{\theta}}\mathbf{e} = [\mathbf{I}_n - d_{\mathbf{e}}P(\mathbf{e}, \boldsymbol{\theta})]^{-1} d_{\boldsymbol{\theta}}P(\mathbf{e}, \boldsymbol{\theta}). \quad (9)$$

546 The indicated inverse does, in fact, exist, but the proof is a detour. Alter-
547 natively, one can compute an entire extinction curve $\mathbf{e}(t)$ whose component
548 $e_i(t)$ supplies the probability of extinction before time t starting from a sin-
549 gle person of type i [12]. This task reduces to solving a ODE $\frac{d}{dt}\mathbf{e}(t)$ by the
550 methods previously discussed. We leave the details to interested readers.

551 The following Julia code computes the sensitivities of the extinction
552 probability for a two-node process by the analytic method.

```
553 using LinearAlgebra
554
555 function extinction(p)
556     types = Int(sqrt(1 + length(p)) - 1) # length(p) = 2 * types + types^2
557     (x, y) = (zeros{Complex, types}, zeros{Complex, types})
558     for i = 1:500 # functional iteration
559         y = P(x, p)
560         if norm(x - y) < 1.0e-16 break end
561     end
562 end
```

```

561     x = copy(y)
562     end
563     return y
564 end
565
566 function P(x, p) # progeny generating function
567     types = Int(sqrt(1 + length(p)) - 1) # length(p) = 2 * types + types^2
568     delta = p[1: types]
569     beta = p[types + 1: 2 * types]
570     lambda = reshape(p[2 * types + 1:end], (types, types))
571     y = similar(x)
572     t = delta[1] + beta[1] + lambda[1, 2]
573     y[1] = (delta[1] + beta[1] * x[1]^2 + lambda[1, 2] * x[2]) / t
574     t = delta[2] + beta[2] + lambda[2, 1]
575     y[2] = (delta[2] + beta[2] * x[2]^2 + lambda[2, 1] * x[1]) / t
576     return y
577 end
578
579 delta = complex([1.0, 1.75]); # death rates
580 beta = complex([1.5, 1.5]); # birth rates
581 lambda = complex([0.0 0.5; 1.0 0.0]); # migration rates
582 p = [delta; beta; vec(lambda)]; # package parameter vector
583 (types, d) = (2, 1.0e-10)
584 @time (e, de) = differential(extinction, p, d)

```

585 To adapt the code to a different branching process model, one simply sup-
586 plies the appropriate progeny generating function and necessary parameters.

The average number a_{ij} of infected individuals of type j ultimately generated by a single initial infected individual of type i is also of interest. The matrix $\mathbf{A} = (a_{ij})$ of these expectations can be calculated via the matrix

equation

$$\mathbf{A} = (\mathbf{I}_n - \mathbf{F})^{-1}, \quad (10)$$

587 where $\mathbf{F}$ is the offspring matrix

$$\mathbf{F} = \begin{pmatrix} \frac{2\beta_1}{\beta_1 + \delta_1 + \lambda_1} & \frac{\lambda_{12}}{\beta_1 + \delta_1 + \lambda_1} & \cdots & \frac{\lambda_{1,n-1}}{\beta_1 + \delta_1 + \lambda_1} & \frac{\lambda_{1n}}{\beta_1 + \delta_1 + \lambda_1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \frac{\lambda_{n1}}{\beta_n + \delta_n + \lambda_n} & \frac{\lambda_{n2}}{\beta_n + \delta_n + \lambda_n} & \cdots & \frac{\lambda_{n,n-1}}{\beta_n + \delta_n + \lambda_n} & \frac{2\beta_n}{\beta_n + \delta_n + \lambda_n} \end{pmatrix}.$$

588 One can determine the local sensitivity of the expected numbers of total
589 descendants by differentiating the equation $\mathbf{A} = (\mathbf{I}_n - \mathbf{F})^{-1}$. The result

$$d_{\theta}\mathbf{A} = (\mathbf{I}_n - \mathbf{F})^{-1} d_{\theta}\mathbf{F} (\mathbf{I} - \mathbf{F})^{-1}, \quad (11)$$

590 depends on the sensitivity of the expected offspring matrix $\mathbf{F}$. Julia code
591 for the analytic method with two nodes follows.

```
592 function particles(p) # mean infected individuals generated
593     types = Int(sqrt(1 + length(p))) - 1 # length(p) = 2 * types + types^2
594     delta = p[1: types]
595     beta = p[types + 1: 2 * types]
596     lambda = reshape(p[2 * types + 1: end], (types, types))
597     F = complex(zeros(types, types))
598     t = delta[1] + beta[1] + lambda[1, 2]
599     (F[1, 1], F[1, 2]) = (2 * beta[1] / t, lambda[1, 2] / t)
600     t = delta[2] + beta[2] + lambda[2, 1]
601     (F[2, 1], F[2, 2]) = (lambda[2, 1] / t, 2 * beta[2] / t)
602     A = vec(inv(I - F)) # return as vector
603 end
604
```

```

605 delta = complex([1.0, 1.75]); # death rates
606 beta = complex([1.5, 1.5]); # birth rates
607 lambda = complex([0.0 0.5; 1.0 0.0]); # migration rates
608 p = [delta; beta; vec(lambda)]; # package parameter vector
609 (types, d) = (2, 1.0e-10)
610 @time (A, dA) = differential(particles, p, d)

```

611 4 Accuracy and Speed

612 We now assess the speed and accuracy of the computational schemes just
613 described. In general, we measure accuracy by the Euclidean norms

$$\begin{aligned}
 \text{err}_1 &= \|f(\mathbf{x} + \mathbf{v}) - f(\mathbf{x}) - df(\mathbf{x})\mathbf{v}\| \\
 \text{err}_2 &= \left\| f(\mathbf{x} + \mathbf{v}) - f(\mathbf{x}) - df(\mathbf{x})\mathbf{v} - \frac{1}{2}\mathbf{v}^t d^2 f(\mathbf{x})\mathbf{v} \right\|.
 \end{aligned}$$

614 Other norms, such as the ℓ_1 and ℓ_∞ norms, yield similar results. In the ODE
615 models, $f(\mathbf{x})$ denotes a matrix trajectory so the Frobenius norm applies.
616 This is scaled by $\frac{1}{n}$, where n is the number of sampled time points. We
617 also compare the analytic derivatives to the automatic derivatives delivered
618 by DiffEqSensitivity.jl (abbreviated DES.jl) via normed differences. Finally,
619 our speed comparisons offer a first look at the efficiency gains possible with
620 multithreading. All computations were done in Julia version 1.6.2 on a
621 Windows operating system with an Intel(R) Core (TM) i7-8565U CPU.

622 Table 2 records benchmarking results for three differential equations
623 models. The top table shows the deterministic SIR model. This model, pre-
624 viously introduced in Section 3.2, serves as a straightforward benchmark for
625 comparing the various differential sensitivity methods summarized earlier.
626 For first-order approximations, we compare the analytic method with the au-

627 automatic differentiation method available in DES.jl [3]. The table shows that
 628 the analytic method offers comparable accuracy but slower compute times
 629 than automatic differentiation. It is noteworthy that more naive implemen-
 630 tations of forward mode differentiation do not take advantage of internal
 631 parallelization (chunking) and are consequently much slower than the an-
 632 alytic method. Results for forward mode differentiation with and without
 633 internal parallelization as implemented in ForwardDiff.jl [14] can be found
 634 in Appendix C.

635 As we expect, second-order approximations are more accurate in predic-
 636 tion. For second-order methods, we compare the double Jacobian method
 637 found in ForwardDiff.jl with the second-order analytic method. In contrast
 638 to the analytic method, forward differentiation slows dramatically in calcu-
 639 lating the Hessian directly, and only speeds up if we instead calculate the
 640 gradient of the gradient. The ForwardDiff.jl package currently does not of-
 641 fer this method natively, so the analytic method provides a simple way to
 642 improve the speed of computing second derivatives. While both of these
 643 methods allow for multi-threading, this model is too low dimensional to
 644 show much of an improvement.

645 The middle subtable in Table 2 records benchmarking results for the
 646 CARRGO model previously described in Section 3.1. This model serves as
 647 an example of a stiff system of differential equations. We can immediately
 648 see the impact of stiffness in the loss of accuracy at time points 100 and
 649 1000. Stiffness highlights the added value of the second-order approxima-
 650 tions. With stiff ODEs, high quality integrators are a must. In the analytic
 651 method, one must also choose the perturbation interval d carefully in com-
 652 puting second derivatives. Careful calibration of d improves the accuracy of
 653 the CARRGO second derivative analytic prediction to the level of automatic

654 differentiation.

655 To explore whether parallelization improves computational speed in high-
656 dimensional models, we now turn a third model summarized in the bottom
657 subtable of Table 2. This ODE model of the mammalian cell cycle (MCC)
658 was constructed by Gerard and Goldbeter [23] and is explained in Appendix
659 B. The model comprises 11 equations and 15 parameters and captures as-
660 pects of cell reproduction and cycling mediated by chemical signaling from
661 cell-state dependent proteins such as through tumor repressors, transcrip-
662 tion factors, and other DNA replication checkpoints. The model relies on
663 cell state as opposed to cell mass and nicely replicates sequential progression
664 along the cell cycle. The model clearly demonstrates the increase in speed
665 from parallelization of both the analytic method and forward differentiation.

666 We now turn to the stochastic SIR model and compare our two suggested
667 methods for computing the derivatives of the mean number of infected in-
668 dividuals (M) and the mean time to extinction (T). Both methods depend
669 on the recurrences (5) and (6). The direct differentiation method relies
670 on differentiating these recurrences, while the analytic method simply per-
671 turbs parameter values and recomputes solutions. Table 3 records the speed
672 and accuracy of the two methods. As expected, computation speed varies
673 quadratically with the number of individuals in the system N . The analytic
674 method proves to be twice as fast as the direct method.

675 The branching process model for the spread of COVID-19 is similar
676 to the SIR model. Table 4 compares the direct method of calculating $\frac{d\mathbf{A}}{d\beta}$
677 based on equation (11) to the analytic method based on equation (10). It
678 also compares the direct method of calculating $\frac{de}{d\beta}$ based on differentiating
679 the recurrence (8) to the analytic method based on the same recurrence. In
680 this case, the analytic method proves to be just as accurate as the direct

Table 2: Time and Accuracy per Time Point for ODE System Derivatives over the Interval $[0, t_{\text{end}}]$

SIR Model						
	Time (μs)			Accuracy		
	$t_{\text{end}} = 10$	$t_{\text{end}} = 100$	$t_{\text{end}} = 1000$	$t_{\text{end}} = 10$	$t_{\text{end}} = 100$	$t_{\text{end}} = 1000$
Forward DES.jl ∇F	8.02×10^1	2.30×10^2	1.77×10^3	1.19	2.75×10^2	9.86×10^3
Analytic ∇F	2.44×10^2	9.94×10^2	8.82×10^3	1.19	2.75×10^2	9.86×10^3
Multi-Threaded Analytic ∇F	2.49×10^2	1.03×10^3	1.22×10^4	1.19	2.75×10^2	9.86×10^3
Parallel FD.jl $\nabla(\nabla F)$	2.07×10^2	4.28×10^2	3.92×10^3	1.15×10^{-1}	7.24×10^1	6.78×10^3
Multi-Threaded Parallel FD.jl $\nabla(\nabla F)$	1.71×10^2	4.29×10^2	3.95×10^3	1.15×10^{-1}	7.24×10^1	6.78×10^3
Analytic $D^2 F$	8.86×10^2	3.76×10^3	5.20×10^4	1.15×10^{-1}	7.24×10^1	6.78×10^3
Multi-Threaded Analytic $D^2 F$	8.77×10^2	3.26×10^3	5.57×10^4	1.15×10^{-1}	7.24×10^1	6.78×10^3

CARRGO Model						
	Time (μs)			Accuracy		
	$t_{\text{end}} = 10$	$t_{\text{end}} = 100$	$t_{\text{end}} = 1000$	$t_{\text{end}} = 10$	$t_{\text{end}} = 100$	$t_{\text{end}} = 1000$
Forward DES.jl ∇F	1.21×10^2	3.69×10^2	2.30×10^3	6.19	2.80×10^5	1.46×10^8
Analytic ∇F	4.41×10^2	2.14×10^3	1.76×10^4	6.19	2.80×10^5	1.46×10^8
Multi-Threaded Analytic ∇F	4.43×10^2	2.17×10^3	2.58×10^4	6.19	2.80×10^5	1.46×10^8
Parallel FD.jl $\nabla(\nabla F)$	1.23×10^3	4.48×10^3	4.38×10^4	1.13×10^{-1}	5.01×10^4	4.23×10^7
Multi-Threaded Parallel FD.jl $\nabla(\nabla F)$	1.20×10^3	4.60×10^3	4.66×10^4	1.13×10^{-1}	5.01×10^4	4.23×10^7
Analytic $D^2 F$	2.38×10^3	1.43×10^4	1.22×10^5	1.12×10^{-1}	4.96×10^4	7.93×10^7
Multi-Thread Analytic $D^2 F$	2.00×10^3	1.14×10^4	1.25×10^5	1.12×10^{-1}	4.96×10^4	7.93×10^7

MCC Model						
	Time (μs)			Accuracy		
	$t_{\text{end}} = 10$	$t_{\text{end}} = 100$	$t_{\text{end}} = 1000$	$t_{\text{end}} = 10$	$t_{\text{end}} = 100$	$t_{\text{end}} = 1000$
Forward DES.jl ∇F	6.12×10^2	9.35×10^3	2.51×10^4	3.47×10^{-3}	7.56×10^{-3}	1.54×10^{-1}
Analytic ∇F	2.71×10^3	1.51×10^4	1.08×10^5	3.47×10^{-3}	7.56×10^{-3}	1.54×10^{-1}
Multi-Threaded Analytic ∇F	2.56×10^3	1.51×10^4	1.08×10^5	3.47×10^{-3}	7.56×10^{-3}	1.54×10^{-1}
Parallel FD.jl $\nabla(\nabla F)$	1.84×10^4	7.44×10^4	2.04×10^5	1.26×10^{-3}	2.07×10^{-3}	4.21×10^{-2}
Multi-Threaded FD.jl $\nabla(\nabla F)$	1.76×10^4	7.77×10^4	2.04×10^5	1.26×10^{-3}	2.07×10^{-3}	4.21×10^{-2}
Analytic $D^2 F$	4.78×10^4	2.81×10^5	1.00×10^6	1.27×10^{-3}	1.92×10^{-3}	3.89×10^{-2}
Multi-Threaded Analytic $D^2 F$	2.56×10^4	1.74×10^5	1.36×10^6	1.27×10^{-3}	1.92×10^{-3}	3.89×10^{-2}

Table 3: Time and Accuracy in the Stochastic SIR Model

	Time (μs)			Accuracy		
	$N = 10$	$N = 100$	$N = 1000$	$N = 10$	$N = 100$	$N = 1000$
$\frac{dM}{d\eta}$ direct	3.21×10^1	3.39×10^3	4.33×10^5	3.47×10^{-3}	2.95×10^{-1}	3.05×10^{-1}
$\frac{dM}{d\eta}$ analytic	1.72×10^1	1.51×10^3	1.70×10^5	3.47×10^{-3}	2.95×10^{-1}	3.05×10^{-1}
$\frac{dT}{d\eta}$ direct	3.78×10^1	3.71×10^3	4.48×10^5	2.04×10^{-2}	4.86×10^{-1}	4.97×10^{-1}
$\frac{dT}{d\eta}$ analytic	1.65×10^1	1.51×10^3	2.45×10^5	2.04×10^{-2}	4.86×10^{-1}	4.97×10^{-1}

Table 4: Time and Accuracy in the Branching Process Model

	Time (μ s)			Accuracy		
	$N = 10$	$N = 100$	$N = 100$	$N = 10$	$N = 100$	$N = 1000$
$\frac{dA}{d\beta}$ direct	4.28	2.46×10^3	1.09×10^5	1.02×10^{-2}	7.41×10^{-4}	1.43×10^{-5}
$\frac{dA}{d\beta}$ analytic	1.46×10^2	4.28×10^3	1.68×10^5	1.02×10^{-2}	7.41×10^{-4}	1.43×10^{-5}
$\frac{de}{d\beta}$ direct	1.29×10^3	8.29×10^4	7.75×10^6	9.23×10^{-5}	9.25×10^{-7}	1.85×10^{-15}
$\frac{de}{d\beta}$ analytic	9.69×10^2	5.64×10^4	1.20×10^7	9.23×10^{-5}	9.25×10^{-7}	1.85×10^{-15}

681 methods, albeit slower in some cases. Other evidence not shown suggests
682 that the analytic method can reliably evaluate sensitivities where solutions
683 depend on linear algebra and/or recurrence relations. Unless derivatives are
684 quite complicated, direct methods will generally be faster. In computing
685 second derivatives, we expect the tables will be turned.

686 5 Discussion

687 Our purpose throughout has been to demonstrate the ease and utility of in-
688 corporating differential sensitivity analysis in dynamical modeling. Because
689 models are always approximate, and parameters are measured imprecisely,
690 uncertainty plagues all of dynamical modeling. Improving models is incre-
691 mental and domain specific. Sensitivity analysis provides a handle on local
692 parameter uncertainty. Assessing global parameter sensitivity is more chal-
693 lenging. It can be attacked by techniques such as Latin square hypercube
694 sampling or Sobel quasi-random sampling, but these become infeasible in
695 high dimensions [24]. Given the availability of appropriate software, dif-
696 ferential sensitivity is computationally feasible, even for high dimensional
697 systems.

698 In the case of stochastic models, traditional methods require costly and
699 inaccurate simulation over a bundle of parameter values. Differential sensi-
700 tivity is often out of the question. This limits the ability of researchers to

understand a biological system and how it responds to parameter changes. If a system index such as a mean, variance, extinction probability, or extinction time can be computed by a reasonable algorithm, then differential parameter sensitivity analysis can be undertaken. We have indicated in a handful of examples how this can be accomplished.

In summary, across many models representative of computational biology, we have found the following results to hold:

- (a) Chunked forward mode automatic differentiation and forward mode sensitivity analysis tend to be the most efficient on the tested models.
- (b) The analytic methods described in this manuscript are competitive and often outperform the unchunked version of forward mode automatic differentiation.
- (c) Shared memory multithreading of the analytic and forward mode automatic differentiation methods provides a performance gain in high-dimensional systems.
- (d) Adjoint methods tend to have better scalability to very large systems with more than 100 ODEs [25]. However, on our test problems the adjoint-based methods were not competitive.
- (e) Forward mode automatic differentiation method requires that each step of a calculation is differentiable. This renders it unusable for calculating the derivative of ensemble means of discrete state models, such as birth-death processes. For these cases, the analytic method outperforms direct numerical differentiation.

These conclusions are tentative, but supported by our limited number of biological case studies.

We note that the performance differences may change depending on the efficiency of the implementations. The Julia DifferentialEquations.jl library and its DiffEqSensitivity.jl package have been shown to be highly efficient, outperforming other libraries in both equation solving and derivative calculations in Python, MATLAB, C, and Fortran [25, 26] (Details on the current state of performance can be found at <https://github.com/SciML/SciMLBenchmarks.jl>.) The automatic differentiation implementations in machine learning libraries optimize array operations much more than scalar operations. This can work to the detriment of forward mode AD. Similarly, forward mode AD sensitivity analysis is more amendable to MATLAB or Python style vectorization that improves performance by reducing interpreter overhead. Therefore, our conclusions can serve as guidelines for the case where all implementations are well-optimized. However, when using programming languages with high overheads or without compile-time optimization of the automatic differentiation passes, the balance in efficiency shifts more favorably towards the analytic method.

One last point worth making is on the coding effort required by the various methods. Both automatic differentiation and the analytic method have comparable accuracy when applied to systems of ODEs, with automatic differentiation having the advantage in speed when it is implemented with an additional level of parallelization. However, the analytic method can easily be generalized to other kinds of objective functions and may be more straightforward to implement for those less sophisticated in computer science. While automatic differentiation is the basis of many large scientific packages, the code required for the analytic methods is fully contained within this manuscript and is easily transferable to other programming languages with similar dispatching on complex numbers. This hard to measure

benefit should not be ignored by practicing biologists who simply wish to quickly arrive at reasonably fast code.

Acknowledgments

We wish to thank Chris Elrod for assistance in adding multithreading to parts of our software. We wish to thank Janet Sinsheimer, Mary Sehl, and Xiang Ji for helpful comments on the manuscript and biological applications.

References

- [1] Cao Y, Li S, Petzold L, Serban R. Adjoint sensitivity analysis for differential-algebraic equations: the adjoint DAE system and its numerical solution. *SIAM Journal on Scientific Computing*. 2003; 24:1076–108
- [2] Strang G. *Computational Science and Engineering*. Wellesley-Cambridge Press. 2007.
- [3] Rackauckas C, Nie Q. Differentialequations.jl—a performant and feature-rich ecosystem for solving differential equations in Julia. *Journal of Open Research Software*. 2017; 5
- [4] Errico RM. What is an adjoint model? *Bulletin of the American Meteorological Society*. 1997; 78:2577–2592
- [5] Granzow GD. A tutorial on adjoint methods and their use for data assimilation in glaciology. *Journal of Glaciology*. 2014; 60:440–446
- [6] Stapor, Paul, et al. PESTO: parameter estimation toolbox. *Bioinformatics*. 2018; 34.4:705–707.

- 774 [7] Hindmarsh et al. SUNDIALS Suite of nonlinear and differen-
775 tial/algebraic equation solvers. ACM Transactions on Mathematical
776 Software (TOMS). 2020; 31(3):363–396
- 777 [8] Lange K. MM Optimization Algorithms. SIAM. 2016
- 778 [9] Dorman K, Sinsheimer JS, Lange K. In the garden of branching pro-
779 cesses. SIAM Review. 20014; 46:202–229
- 780 [10] Yakovenko S, Ilyashenko Y. Lectures on Analytic Differential Equa-
781 tions. American Mathematical Society. 2008.
- 782 [11] Henrici P. Fast Fourier transform methods in computational complex
783 analysis. SIAM Review. 1979; 21:481–527
- 784 [12] Lange, K . Applied Probability, 2nd ed. Springer. 2010.
- 785 [13] Lai K-L Crassidis J, Cheng,Y, Kim J. New complex-step derivative ap-
786 proximations with application to second-order Kalman filtering. AIAA
787 Guidance, Navigation, and Control Conference and Exhibit 2005. 2005;
788 :5944
- 789 [14] Revels, J et al. Forward-mode automatic differentiation in Julia. arXiv
790 preprint. 2016; 1607.07892.
- 791 [15] Sahoo P et al. Mathematical deconvolution of CAR T-cell proliferation
792 and exhaustion from real-time killing assay data. Journal Royal Society
793 Interface. 2020; 17: 20190734.
- 794 [16] Toda, AA. Susceptible-infected (SIR) dynamics of COVID-19 and eco-
795 nomic impact. arXiv preprint. 2020; 2003.11221

- 796 [17] Zhou F et al. Clinical course and risk factors for mortality of adult
797 inpatients with COVID-19 in Wuhan, China: a retrospective cohort
798 study. *The Lancet*. 2020; 395:1054–1062
- 799 [18] Landeros A, Stutz, T, Keys, KL, Alekseyenko, A, Sinsheimer, JS,
800 Lange, K, & Sehl, ME. BioSimulator. jl: Stochastic simulation in Julia.
801 *Computer Methods and Programs in Biomedicine*. 2018; 167, 23-35.
- 802 [19] Renshaw E. Birth, death and migration processes. *Biometrika*. 1972;
803 59:49–60
- 804 [20] Hautphenne S, Krings G, Delvenne J-C, Blondel VD. Sensitivity anal-
805 ysis of a branching process evolving on a network with application in
806 epidemiology. *Journal of Complex Networks*. 2015; 3:606–641
- 807 [21] Athreya KB, Ney PE. *Branching Processes*. Springer. 1972.
- 808 [22] Magnus JR, Neudecker H. *Matrix Differential Calculus with Applica-*
809 *tions in Statistics and Econometrics*. Wiley, New York. 1988; pp. 30–
810 31,158–165
- 811 [23] Gérard C, Goldbeter A. Temporal self-organization of the cyclin/Cdk
812 network driving the mammalian cell cycle. *Proceedings of the Na-*
813 *tional Academy of Sciences of the United States of America*. 2009;
814 106:51:21643-21648
- 815 [24] Qian G, Mahdi A. Sensitivity analysis methods in the biomedical sci-
816 ences. *Mathematical Biosciences*. 2020; 323:108306
- 817 [25] Rackauckas, Christopher, et al. A comparison of automatic differenti-
818 ation and continuous sensitivity analysis for derivatives of differential
819 equation solutions. *arXiv preprint*. 2018; arXiv:1812.01892

[26] Rackauckas, Christopher, et al. Universal differential equations for scientific machine learning. arXiv preprint. 2020; arXiv:2001.04385

Appendix A: Derivation of Second Derivative Analytic Method

To prove the formulas for approximating partial derivatives stated in the text, we first note that any analytic function $f(\mathbf{z})$ of several variables can be expanded in a locally convergent power series about every point $\mathbf{z}$ of its open domain of definition. If we choose a real direction vector $\mathbf{v}$, then the function $g(w) = f(\mathbf{z} + w\mathbf{v})$ is locally analytic in the complex plane $\{\mathbf{z} + w\mathbf{v} : w \in \mathbb{C}\}$ and can be expanded in a power series around $w = 0$. Thus,

$$g(w) = \sum_{j=0}^d \frac{d^j}{dw^j} g(\mathbf{0}) \frac{w^j}{j!} + O(|w|^{d+1}) \quad (12)$$

for any integer $d \geq 0$. Now consider the setting where $\mathbf{z}$ has real components. If $f(\mathbf{z})$ is real valued, then the derivatives $\frac{d^j}{dw^j} g(\mathbf{0})$ will be real as well. One can exploit this fact in approximating the derivatives. For example, if $w = i$, then w^j rotates among the four values 1, i , -1 , and $-i$. Because the terms of the expansion (12) alternate between real and imaginary values, the first partial derivative formula (2) holds. For the choice $w = e^{\pi i/4}$, the powers w^d rotate among the eight values 1, $e^{\pi i/4}$, i , $ie^{\pi i/4}$, -1 , $-e^{\pi i/4}$, $-i$, and $-ie^{\pi i/4}$. The powers $(-w)^j = (-1)^j w^j$ agree in this regard except for sign. Hence, the terms in the expansion of the sum

$$g[\mathbf{x} + e^{\pi i/4} \Delta(\mathbf{e}_j + \mathbf{e}_k)] + g[\mathbf{x} - e^{\pi i/4} \Delta(\mathbf{e}_j + \mathbf{e}_k)]$$

840 alternately cancel and reinforce. Thus, the first five terms of the expansion
 841 are real, 0, imaginary, 0, real, 0. It follows that the imaginary part of the
 842 sum is accurate to order $O(\Delta^6)$ and that the approximations (3) and (4) are
 843 accurate to order $O(\Delta^4)$.

844 **Appendix B: Mammalian Cell Cycle Model**

The Mammalian Cell Cycle Model presented in [23] reduces to a system of ordinary differential equations representing the interaction of cyclin-dependent kinases (Cdk) with Cdk inhibitors, growth factors, and other proteins that regulate the development of mammalian cells. The model includes characteristics such as cell cycling, tumor repressor initiated progression control,

and cell cycle completion. The ODE system representing the model is

$$\begin{aligned}
 \frac{dpRBc1}{dt} &= kpc1 * pRB * E2F \\
 \frac{dpRBc2}{dt} &= kpc3 * pRBp * E2F \\
 \frac{dCd}{dt} &= kcd1 * AP1 + kdecom1 * Mdi \\
 &\quad - kcom1 * Cd * (Cdk4_{tot} - (Mdi + Md + Mdp27)) \\
 &\quad + kcd2 * E2F * \frac{Ki7}{Ki7 + pRB} * \frac{Ki8}{Ki8 + pRBp} \\
 \frac{dMdi}{dt} &= Vm2d * \frac{Md}{k2d + Md} + 2 * kcom1 * Cd * (Cdk4_{tot} - (Mdi + Md + Mdp27)) \\
 \frac{dMd}{dt} &= Vm1d * \frac{Mdi}{k1d + Mdi} + kcom1 * Cd * (Cdk4_{tot} - (Mdi + Md + Mdp27)) \\
 \frac{dpRB}{dt} &= kcd2 * E2F * \frac{Ki7}{Ki7 + pRB} * \frac{Ki8}{Ki8 + pRBp} \\
 \frac{dE2F}{dt} &= kcd2 * E2F * \frac{Ki7}{Ki7 + pRB} * \frac{Ki8}{Ki8 + pRBp} \\
 \frac{dpRBp}{dt} &= kcd2 * E2F * \frac{Ki7}{Ki7 + pRB} * \frac{Ki8}{Ki8 + pRBp} \\
 \frac{dAP1}{dt} &= kcd1 * AP1 \\
 \frac{dp27}{dt} &= 0 \\
 \frac{dMdp27}{dt} &= kc1 * Md * p27 + kcom1 * cd * (Cdk4_{tot} - (Mdi + Md + Mdp27))
 \end{aligned}$$

845 The initial values of each compartment and parameter are defined as

Compartment	Initial Value (μ mol)	Parameter	Value
pRBc1	0.1	kpc1	0.05
pRBc2	0.05	kpc3	0.025
Cd	0.01	kcd1	0.4
Mdi	0.01	Ki8	2.0
Md	0.01	Ki7	0.1
pRB	0.0	kcd2	0.005
E2F	0.0	kdecom1	0.1
pRBp	0.0	k2d	0.1
AP1	0.0	Cdk4 _{tot}	1.5
p27	0.0	kcom1	0.175
Mdp27	0.0	Vm2d	0.2
—	—	k1d	0.1
—	—	Vm1d	1.0
—	—	kc1	0.15

Appendix C: Supplementary Benchmarking Data

In addition to the methods for conducting the differential sensitivity analysis for systems of ODEs displayed in Table 2 (Section 4), we conducted benchmarking tests on some variations of these methods as well. Tables 5 and 6 summarize these benchmarks for the SIR Model. Additional first-order methods include using the ForwardDiff.jl [14] package to calculate the Jacobian with and without chunking and with and without multi-threading, and using a constant value for $\epsilon = 1 \times 10^{-12}$ instead of one scaled by the parameters. Additional second order methods include using DES.jl

Table 5: Time and Accuracy per Time Point for ODE System Derivatives over the Interval $[0, t_{\text{end}}]$ - Additional First Order Methods

SIR Model Time in μs					
t_{end}	Parallel FD.jl Gradient	Parallel FD.jl Multi-Threaded Gradient	FD.jl Multi-Threaded Gradient	FD.jl Gradient	Constant Analytic Gradient
10	1.40×10^2	1.47×10^2	3.41×10^2	3.59×10^2	3.19×10^2
100	4.33×10^2	4.39×10^2	1.10×10^3	1.12×10^3	1.40×10^3
1000	3.20×10^3	4.79×10^3	8.73×10^3	1.15×10^4	1.28×10^4

SIR Model Accuracy					
t_{end}	Parallel FD.jl Gradient	Parallel FD.jl Multi-Threaded Gradient	FD.jl Multi-Threaded Gradient	FD.jl Gradient	Constant Analytic Gradient
10	1.19	1.19	1.19	1.19	1.19
100	2.75×10^2	2.75×10^2	2.75×10^2	2.75×10^2	2.75×10^2
1000	9.86×10^3	9.86×10^3	9.86×10^3	9.86×10^3	9.86×10^3

Table 6: Time (ns) and Accuracy per Time Point for ODE System Derivatives over the Interval $[0, t_{\text{end}}]$ - Additional Second Order Methods

SIR Model Time in μs					
t_{end}	DES.jl Adjoint Hessian	DES.jl Non-Adjoint Hessian	FD.jl Double Jacobian	FD.jl Multi-Threaded Double Jacobian	Constant Analytic Hessian
10	3.72×10^4	1.18×10^4	1.00×10^3	9.81×10^2	1.16×10^3
100	4.23×10^5	1.30×10^5	3.27×10^3	3.19×10^3	5.28×10^3
1000	8.67×10^6	2.37×10^6	4.22×10^4	4.26×10^4	5.61×10^4

SIR Model Accuracy					
t_{end}	DES.jl Adjoint Hessian	DES.jl Non-Adjoint Hessian	FD.jl Double Jacobian	FD.jl Multi-Threaded Double Jacobian	Constant Analytic Hessian
10	1.15×10^{-1}	1.15×10^{-1}	1.15×10^{-1}	1.15×10^{-1}	7.53×10^{-1}
100	7.24×10^1	7.24×10^1	7.24×10^1	7.24×10^1	1.33×10^2
1000	6.78×10^3	6.78×10^3	6.78×10^3	6.78×10^3	7.15×10^3

856 (both adjoint and non-adjoint) to calculate the Hessian, using the Forward-
857 Diff.Hessian method, using the ForwardDiff.jacobian(ForwardDiff.jacobian)
858 method without chunking (with and without multithreading), and using the
859 constant value $\epsilon = 1 \times 10^{-6}$ in the analytic method.