ChatGPT

Roadmap: Surrogates, Simulation-Based Inference, and Differentiable Modeling

1. Introduction and Motivation

This opening chapter will describe the challenge of inferring parameters in complex **mechanistic models** (like epidemic simulations) and why new approaches are needed. Traditional epidemiological models, whether **ODE-based** or **agent-based**, often yield **intractable likelihoods**, making standard statistical inference difficult [1]. We'll motivate **simulation-based inference (SBI)** as a solution for *implicit* models (where the likelihood of data given parameters is unavailable analytically) [1]. Recent research by Quera-Bofarull, Dyer, Semenova and others will be introduced as examples – e.g. **differentiable agent-based models** for policy optimization and **deep surrogate models** for speeding up epidemic simulations. The Introduction will also outline the structure of the resource, emphasizing that each chapter builds the prerequisites for understanding these advanced topics.

2. Bayesian Inference Foundations

This chapter provides a concise recap of probability and Bayesian inference to ensure readers share a common foundation. It will cover the definition of a **statistical model** (parameters, data, likelihood) and Bayes' rule for obtaining the **posterior** distribution of parameters. We'll distinguish between **prescribed (explicit) models** – where one can write down a likelihood – and **implicit models**, where the generative process is known but the likelihood is *intractable* [1]. Concepts of **frequentist** vs **Bayesian** inference will be reviewed (point estimation vs full posterior) [2]. We also introduce the notion of **calibration** (parameter estimation) for mechanistic models, highlighting the need to quantify uncertainty in parameters. This sets the stage for why specialized methods (like ABC or neural inference) are required when dealing with complex simulators.

3. Mechanistic Models in Epidemiology (ODEs and ABMs)

Here we dive into the types of mechanistic infectious disease models that motivate our study. We'll start with **compartmental ODE models** (e.g. the classic SIR model) and discuss how they generate expected epidemic curves given parameters (transmission rates, etc.). Then we introduce **agent-based models (ABMs)**, which simulate individuals and their interactions. ABMs capture richer detail but often involve **discrete events** (e.g. transmissions, movement) and complex **stochastic** dynamics [3] [4]. We explain that such simulations define an implicit probability model for data (e.g. case counts), but the probability of an exact outcome given parameters is generally unknown in closed-form [5]. This chapter emphasizes what it means for a model to be *high-fidelity but computationally intensive*. We will also discuss why ABMs are typically **non-differentiable** (due to event logic and randomness) and thus pose challenges for gradient-based methods. Real-world examples like the JUNE COVID-19 ABM or network-based SIR simulations [6] will illustrate these concepts. By the end, readers will appreciate the trade-off between model realism and statistical tractability.

4. Simulation-Based Inference (Likelihood-Free Methods)

This chapter introduces Simulation-Based Inference, explaining how we can perform inference when the model's likelihood is intractable by using the simulator itself. We define **SBI** and its alias **"likelihood-free inference,"** noting that the term is a bit of a misnomer since one often attempts to estimate or circumvent the likelihood rather than avoid it entirely [7]. We cover two classical approaches: (1) **Approximate Bayesian Computation (ABC)**, which generates simulations from the model and accepts parameters that produce simulated data sufficiently close to observed data [8]. ABC's use of distance measures on carefully chosen **summary statistics** (e.g. epidemic peak size, timing) will be described, along with its widespread use in fields like population biology and epidemiology [8]. (2) **Synthetic likelihood and density estimation** methods, where one fits an approximate probability model (e.g. a Gaussian density) to simulated summary statistics [9]. We'll explain how early successful applications (such as the discovery of the Higgs boson) used density estimation of summaries [9]. Then, we transition to **modern SBI techniques** that leverage machine learning: we outline how **neural density estimators** can learn either the **likelihood function or the posterior distribution** from simulation data [10]. For example, we introduce **normalizing flows** as a flexible method to model complex probability densities by transforming a simple distribution through an invertible neural network [11]. We also mention **neural likelihood ratio estimation**, where a classifier is trained to distinguish simulated data from real data to infer parameter likelihood ratios [12]. The concept of **amortized inference** will be highlighted: once a neural density estimator is trained on simulations, one can get instant posterior or likelihood estimates for new data without rerunning simulations [13] [14]. Throughout this chapter, we emphasize the advantages of SBI (applicable to any simulator, leverages increased computing power) and its challenges (curse of dimensionality, need for many simulations, choosing summaries) [15] [16]. By the end, readers will understand how approaches like ABC and neural posterior estimation provide avenues to infer parameters from complex disease simulations where classical methods falter.

5. Automatic Differentiation (AD) – Concepts and Tools

To prepare for gradient-based methods, we next cover the fundamentals of automatic differentiation. We define **Automatic Differentiation** as a family of techniques to compute exact derivatives of functions defined by computer programs [17]. Unlike symbolic math, AD works by applying the chain rule to every operation in the code, and unlike finite differences, it yields exact (up to machine precision) gradients. We will distinguish **forward-mode** and **reverse-mode** AD, noting that modern deep learning frameworks rely on reverse-mode (backpropagation) to efficiently compute gradients of a scalar loss with respect to many parameters. Simple examples (with pseudocode or Python) will illustrate how AD evaluates derivatives (e.g. computing the derivative of a compound function step by step). We'll highlight popular AD-enabled libraries like PyTorch, TensorFlow, and JAX, which allow us to "differentiate programs" as long as those programs use operations that are themselves differentiable. This sets the stage for using gradients in inference. We'll close the chapter by contrasting AD with traditional gradient approximation (like finite differences), to show why AD is more exact and scalable for high-dimensional problems [17]. This background will be crucial for understanding how we can obtain gradients even for complex models – if we can express them in a differentiable manner.

6. Differentiable vs. Non-Differentiable Models

In this chapter, we explore what it means for a model or simulator to be differentiable, and why many mechanistic models are not. A **differentiable model** is one where the mapping from parameters to outputs is smooth enough that we can calculate gradients. We give examples: a simple deterministic ODE solver (without events) is largely differentiable – small changes in parameters yield small changes in output, and we can propagate derivatives through the ODE integration. On the other hand, **non-**

differentiable models include those with discrete decision logic or stochastic branching. We explain that features like **discontinuous control flow** (e.g. `if` statements or agent decisions) and **discrete random draws** (e.g. Bernoulli trials, categorical choices) break the smooth relationship needed for gradients [18] [19]. For instance, an ABM where an infected individual infects a neighbor with some probability has an outcome (infected or not) that is a non-differentiable function of that probability – it's either 0 or 1 outcome, a jump. We'll formalize that an implicit simulator defines an expectation over many possible trajectories, and the **likelihood function** is an integral over all latent possibilities [20], which is typically not available in closed form. We highlight that when a model is differentiable *and* we can write a likelihood, we can use powerful gradient-based inference (like Hamiltonian Monte Carlo or gradient descent for MLE). In contrast, for non-differentiable simulators, we traditionally resort to *gradient-free* methods (like ABC rejection sampling or manual calibration by heuristic). This chapter sets up the motivation for making models differentiable or using workarounds: if we can somehow endow our simulator with (approximate) gradients, we unlock far more efficient inference algorithms. We may include a brief note on how frameworks like **JAX** can differentiate through certain simulations (e.g. differential equation solvers) and what the limitations are. By the end, readers will understand clearly *why* the distinction between differentiable and non-differentiable model is so critical for inference – a theme central to recent research.

7. Neural Networks Primer (Function Approximators)

Now we shift to machine learning basics, introducing **neural networks** as flexible function approximators that will reappear as surrogates and inference tools. We'll start with the anatomy of a simple feedforward neural network: layers of neurons computing linear combinations of inputs and applying nonlinear activation functions. The key idea that neural networks can approximate complex mappings given enough capacity will be explained (universal approximation intuition). We'll keep this section accessible: for example, showing how a small network can fit a sine wave or classify infectious vs recovered individuals given some features. Importantly, we tie this to automatic differentiation: neural networks are differentiable programs by construction, composed of basic ops (matrix multiplies, activation functions) that are all differentiable. Thus, their parameters can be optimized with gradient-based methods. We introduce the concept of **training** a neural network via a **loss function** and **backpropagation**: for supervised tasks, we define a loss (e.g. mean squared error between network prediction and data) and use **stochastic gradient descent (SGD)** to iteratively adjust weights in the direction that lowers the loss. Backpropagation is simply the reverse-mode AD applied to the network's computations to get the gradient of the loss w.r.t. each weight. We will include a simple illustrative example (with code snippet) of training a tiny network to fit some data, to demystify the process. This primer ensures that readers unfamiliar with AI concepts understand what a neural network is and how it "learns" from data, since later chapters will use neural nets as *tools* (surrogates, density estimators). We'll also briefly mention modern frameworks (PyTorch, JAX+Flax) that make it easy to define and train neural nets, since our eventual examples may use those. By the end of this chapter, readers coming from a statistical background will have enough intuition about neural nets and gradient descent to follow their use in subsequent, more specialized contexts.

8. Surrogate Modeling for Complex Simulations

This chapter introduces **surrogate models** – simplified models (often ML-based) that approximate the input–output behavior of complex simulations. We begin with a general definition: *surrogate models are computationally cheaper approximations of expensive models, designed to emulate the original model's outputs with high fidelity* [21]. Historically, surrogates have included polynomial response surfaces, support vector regressors, or Gaussian Process emulators [21]. Here, we focus on modern **deep learning surrogates** due to their flexibility and scalability. The chapter will describe why surrogates are

useful in our context: for example, an epidemiological ABM might take hours to simulate an outbreak for a given parameter set, but a neural network surrogate (trained on many simulations) could instantly predict the outbreak outcomes (or summary statistics) given new parameters [22]. We explain the process of building a surrogate: first, generate a training dataset by running the original simulator for many different parameter values; then, train a neural network to map from parameters to outcomes (or summary stats). The surrogate, once trained, can *replace* the simulator for inference tasks, providing quick predictions. We'll cite evidence that this approach dramatically accelerates calibration – e.g. a review found that most surrogate-based calibration studies for epidemics used neural network surrogates and saw big speedups, especially for agent-based models [22]. Surrogates essentially learn the relationship between inputs and outputs, so they can also smooth out the model's behavior. A key point: if the original model is stochastic, the surrogate might predict *expectations* or distribution characteristics of the output. We will discuss training targets, such as having the surrogate predict the mean outcome or full distribution (maybe via mixture density networks). Real examples will be given, like the **EpiCast** surrogate which learned to emulate an epidemic ABM and achieved <0.1% error in outcomes while being orders of magnitude faster [23] [22]. We also mention the use of surrogates for ODE models: one study's *Deep Neural Network Surrogate Model* replicated ODE epidemic model behavior with 10× faster forward simulation and 5× faster parameter fitting [24] [25]. This demonstrates surrogates' value not just for ABMs but even for complex ODE systems. Finally, we'll caution that a surrogate is an approximation – we must validate that it's accurate enough. We'll describe common validation techniques: comparing surrogate predictions to a test set of simulator runs, and checking that key dynamics are captured. Concluding this chapter, readers will understand what surrogate models are and how deep learning can be used to build fast emulators of slow epidemic models – an important strategy in modern simulation-based inference [22].

9. Gradient Estimation Tricks for Non-Differentiable Models

One of the most pivotal sections, this chapter covers techniques (the "tricks") that enable gradient-based optimization or inference even when the model has non-differentiable components. We start with the concept of a **surrogate gradient**: replacing or bypassing a non-differentiable operation with a differentiable approximation during backpropagation [26]. For example, in an agent-based model, a discrete decision (e.g. an agent chooses location A vs B) can be treated as-is in the forward simulation, but in the backward pass we pretend a smooth function was used. This idea was popularized by **straight-through estimators** – where the discrete function is used forward, but its gradient is set to the identity (or some smooth derivative) backward [27]. We will explain the *straight-through estimator* (from Bengio et al. 2013) in simple terms: you take a hard step (like a threshold) in forward pass but pass the gradient as if it were an identity function (or a smooth sigmoid) in the backward pass [27]. This gives a biased but often useful gradient signal. Next, we introduce the **Gumbel-Softmax trick** as a more principled surrogate for discrete choices. We'll describe how a categorical random choice can be reparameterized by sampling Gumbel noise and producing a softmax (continuous) approximation of the one-hot outcome [28]. By adjusting a temperature parameter, the softmax can approximate a one-hot more or less closely; at zero temperature it replicates the discrete distribution, but for differentiation we use a small positive temperature to get a smooth function [28]. The Gumbel-Softmax estimator allows treating discrete variables in a network as if they were differentiable during training, and it was introduced by Jang et al. (2017) and Maddison et al. (2017) [29]. We will include a simple example (e.g. differentiating through a random coin flip by using a sigmoid as a surrogate). Then we cover the fundamental **reparameterization trick** for continuous random variables: this is used in variational autoencoders and elsewhere to obtain low-variance gradient estimates by expressing a random variable as a deterministic function of a base noise [30]. We'll explain: instead of sampling $X \sim P_\theta$, one can sample an auxiliary noise ε (independent of θ) and set $X = g(\theta, \varepsilon)$, a differentiable transformation [30] [31]. This allows gradients to propagate through g with respect to θ, effectively "pushing the stochasticity out." The classic example is

sampling from $N(\mu,\sigma^2)$ by sampling $\varepsilon\sim N(0,1)$ and setting $X=\mu+\sigma\varepsilon$: now the gradient w.rt μ,σ can be derived easily [32]. We emphasize that the reparameterization trick yields **low-variance, unbiased** gradient estimates for continuous distributions [30]. In contrast, we also describe the **score function estimator** (REINFORCE), which provides an unbiased gradient even for non-differentiable cases by using the log-derivative trick [33] [34]. REINFORCE doesn't require smoothing – it directly estimates $\nabla_\theta \mathbb{E}[f(X)]$ as $\mathbb{E}[f(X)\nabla_\theta \log P_\theta(X)]$ [34]. We'll mention that while unbiased and general (works for any stochastic element), REINFORCE often has high variance gradients [35], making it impractical without variance reduction. We will compare these approaches: **pathwise gradients** (reparameterization) vs **score gradients** (likelihood ratio) vs **surrogate gradients** (biased but easy). The chapter will likely include a small demonstration of estimating a gradient through a toy simulator (perhaps a simple branching process) using these methods, to show the difference in practice (e.g. REINFORCE vs a surrogate gradient). We also tie this back to recent research: for instance, Quera-Bofarull et al. leverage the Gumbel-Softmax reparameterization to create **differentiable agent-based models**, enabling gradient-based calibration of those models [36] [37]. We'll cite how their differentiable ABM approach uses these tricks to allow gradient flow through events that are normally non-differentiable. By the end of this chapter, readers should understand the toolkit of gradient estimators: *when and how to use surrogate gradients, straight-through estimators, Gumbel-Softmax, and reparameterization*, and the existence of trade-offs in bias vs variance [26] [27]. These "tricks for inference" are crucial to implement gradient-based optimization or learning in the context of simulators that are not naturally amenable to differentiation.

10. Bringing It Together: Gradient-Based Inference for Simulators

Having laid the groundwork, this chapter will show how differentiability (or proxies for it) can be leveraged to perform efficient inference on mechanistic models. We start by discussing **gradient-based calibration**: using gradient descent (or more advanced optimizers) to find parameters that best match observed data. If a simulator (or its surrogate) is differentiable, one can compute the gradient of an objective (e.g. squared error between simulated and observed summary stats) with respect to parameters and then adjust parameters accordingly. We'll illustrate this with a simple example: say calibrating an SIR model by minimizing the difference between predicted and observed epidemic curves – we could do this with gradient-based optimization if we have the gradients. We mention that this approach can find *point estimates* (MLE or MAP) quickly, compared to brute-force searching the parameter space. Next, we cover **Bayesian inference with gradients**. This includes methods like **Hamiltonian Monte Carlo (HMC)** which require gradients of the log-posterior. We explain that tools like **NumPyro** or **Stan** can automatically do HMC on differentiable models with known likelihoods. For example, if we have a differentiable surrogate for our ABM's data likelihood, we could run NUTS (No-U-Turn Sampler) to sample from the posterior of parameters. We will describe HMC conceptually (using momentum and gradients to efficiently explore the parameter space) and why it scales better than random-walk Metropolis for high dimensions. We also briefly mention **variational inference (VI)**: approximating the posterior by a parametric distribution and using gradient-based optimization (often via the reparameterization trick) to fit it. For instance, one could use VI on a surrogate likelihood model to get a quick approximate posterior. We highlight that many of these advanced inference algorithms, including normalizing-flow-based posterior estimators, rely on the ability to differentiate either the model or an approximate model. This section will connect to the works of our motivating authors: for example, Semenova et al. use **deep generative surrogates** combined with Bayesian workflow to do efficient policy inference in epidemiology [22]. Arnau Quera-Bofarull's work on **Bayesian calibration of differentiable ABMs** (2023) shows that once an ABM is differentiable, one can perform full Bayesian inference on it (using either gradient MCMC or VI) which was previously infeasible [38]. We will also

recount Joel Dyer's contributions: e.g. *gradient-assisted calibration*, where gradients (possibly from surrogates) are used to speed up fitting of ABMs in domains like finance [39]. The chapter might include a small case study combining elements: for instance, using a neural surrogate of an epidemic ABM in an HMC sampler to obtain the posterior of parameters given real data – thus demonstrating *SBI via surrogate + gradients*. We will discuss practical considerations: ensuring the surrogate's error is small enough not to mislead inference, maybe doing a two-stage approach (surrogate to locate posterior mode, then ABC around that region to verify). By the end of this chapter, readers should see how the techniques from previous sections coalesce into workable inference pipelines for complex models: **use surrogates to approximate the simulator, use AD to get gradients, then use those gradients in either optimization or sampling algorithms** to infer parameters. This is exactly the kind of approach enabling cutting-edge research in combining AI with mechanistic epidemiology.

11. Case Studies and Examples

To solidify understanding, this chapter will present a few end-to-end case studies with code, demonstrating the methods on realistic (but simplified) scenarios. Each case study will walk through model description, data generation, and then the inference procedure step by step with commentary. Possible examples: **(A)** *Calibrating a Differential Equation Model*: We take an ODE-based SEIR model, simulate outbreak data, then assume we only observe partial data. We demonstrate calibrating it in three ways: via classical MCMC (using NumPyro's NUTS) since ODEs are differentiable and we can compute a likelihood, then via a "likelihood-free" method (ABC) treating the model as a black box, and compare results. This highlights when AD and gradients provide an efficiency boost. **(B)** *Calibrating an Agent-Based Model*: We design a toy agent-based epidemic model (small population, network-based SIR as described in Chapter 3). We then show two calibration approaches: first, ABC rejection or Sequential Monte Carlo ABC, which might be very slow. Then, we illustrate training a neural network surrogate on simulation data from the ABM. With the surrogate in hand, we use gradient-based optimization (or variational inference) to fit the ABM parameters to some observed data. This example will include code (perhaps using JAX/Flax for the surrogate and optax for optimization) and will emphasize validation – e.g. checking the surrogate's predictions vs the ABM for some test runs, and maybe re-running the ABM at the inferred parameters to see if it indeed matches data. We will also show the speed difference: the surrogate-based inference might converge much faster than ABC which requires thousands of simulator runs. **(C)** *Policy Optimization with Differentiable Simulation*: As a forward-looking example, we consider using gradients to find an optimal intervention strategy in a model. For instance, using the differentiable SIR ABM concept, if we can differentiate the outcome (e.g. total infections) with respect to a policy parameter (like the timing of a lockdown), we could use gradient ascent/descent to find the best policy. We illustrate this with a simplified differentiable simulation (possibly leveraging a surrogate or an analytical SIR model) and show how one would optimize a control parameter. This ties to how one might use **reinforcement learning or differentiable optimal control** for epidemic interventions – bridging into the realm of **policy optimization**. Each case study will be self-contained and written in an accessible, tutorial style. By working through these concrete examples, the reader will gain hands-on intuition on applying surrogates and SBI methods to real problems. The code and results (plots of fitting, etc.) will be displayed, mirroring the style of an interactive textbook (just like the Prob-Epi course notebooks). This practical chapter ensures that the earlier theoretical sections translate into actionable skills.

12. Practical Considerations and Tools

In this chapter, we step back and discuss the practical side of implementing the methods covered. We will survey the existing **software libraries** and **frameworks** that can facilitate SBI and surrogate modeling. For example: **NumPyro** and **PyMC** for probabilistic programming (great for HMC and VI on

differentiable models), **sbi** (a Python package for simulation-based inference with neural density estimators), and libraries for ABC (like **pyABC** or **ELFI**). We'll also mention **JAX** for writing high-performance differentiable simulations (as it can jit-compile and auto-differentiate code, which is useful for writing differentiable physics or epidemiological models). If relevant, we discuss how to write an epidemic model in JAX or PyTorch so that gradients can flow – for instance, implementing an ODE integrator with JAX or a simplified differentiable ABM with PyTorch (using straight-through where needed). Another practical aspect is **computational cost**: we highlight that training neural surrogates or running thousands of simulations for SBI is computationally heavy – one often needs to use HPC or at least parallel simulation. Techniques like **active learning** in SBI (iteratively choosing parameter points to simulate that improve inference) could be touched upon [40], as they can greatly reduce the number of simulations needed. We'll also discuss **evaluation metrics** for inference quality: how to assess if the inferred posterior is accurate (e.g. simulation-based calibration plots), and if the surrogate introduces any bias (by comparing surrogate-inference results to ground truth from the actual simulator if possible). Additionally, we consider the **limitations** and common pitfalls: for instance, surrogates might fail to extrapolate outside the range they were trained on, and SBI methods can sometimes yield posteriors that are overconfident if model discrepancy exists. We will mention the importance of checking that the surrogate or inference method is robust – e.g. via coverage diagnostics or by testing on synthetic data with known true parameters. Finally, we provide guidance on the *workflow* for combining these tools in a research project: typically one would start by doing a traditional analysis on a simpler model, then incrementally add surrogate or SBI components, verifying each step. This chapter acts as a bridge from theory to real-world application, giving readers advice on using the methods responsibly and effectively in their own problems.

13. Further Topics and Research Frontiers

In the concluding content chapter, we survey advanced topics and current research frontiers that build on the foundations we've learned. We highlight that the integration of AI with mechanistic modeling is a fast-evolving field [41] [42]. Some topics to mention: **Causal Consistency in Surrogates** – e.g. Dyer *et al.* (NeurIPS 2022) introduced *interventionally consistent surrogates*, addressing how to ensure a surrogate remains valid under different policy interventions (this touches on causal inference ideas and is relevant if one trains a surrogate on one distribution but uses it in another) [43]. **Physics-Informed Neural Networks (PINNs)** – these are a way to enforce mechanistic equations in a neural network surrogate (not directly our focus, but related in that they blend differential equation knowledge with NN training, often used for surrogate modeling of PDEs). We might mention that some epidemiological modeling efforts use PINNs to incorporate ODE constraints. **Privacy-Preserving or Federated SBI** – a niche but interesting area, e.g. doing inference on simulators without revealing the simulator code or data, using techniques like encrypted or distributed computation (maybe beyond our scope, but could be mentioned as an aside if relevant to the user's interest). We will also point to the **broader impact**: how these methods are being applied beyond epidemiology – for example, in economics (agent-based market models), in climate science (calibrating complex simulators), etc. This underscores that what the reader learned has broad applicability. Another frontier: **Active learning for simulation** (a brief revisit) – methods that intelligently choose which simulation runs to do next based on current posterior (like Bayesian optimization for SBI). And **Scalable ABC** using neural nets to automatically choose summary statistics or distance metrics [16] – we note that modern ABC can be coupled with neural network embeddings to improve performance. We will list a few key references for further reading: e.g. the **"Frontier of Simulation-Based Inference"** review by Cranmer et al. (2020) for an overview of the field [15], recent papers by our focus authors (Quera-Bofarull et al. 2023 on differentiable ABM calibration [38], Semenova et al. on deep generative surrogates for epidemiology, etc.), and perhaps tutorials or open-source projects (like **podsbi** or others). The aim of this chapter is to inspire readers to delve deeper and to update the resource as the field progresses. We conclude by emphasizing the theme of the resource: bridging traditional epidemiological modeling with cutting-edge machine learning

techniques can yield powerful tools for inference and decision-making – and though the concepts are advanced, we've broken them down from the ground up. The reader should now have a comprehensive map of the concepts (surrogates, SBI, autodiff, etc.) and how they interconnect, enabling them to read and understand the latest research by Arnau Quera-Bofarull, Joel Dyer, Elizaveta Semenova, and others with confidence.

14. Appendices (Mathematical Details and Glossary)

Finally, we will include appendices for reference. One appendix might detail mathematical derivations or additional proofs omitted in the main text (for example, a derivation of the likelihood ratio gradient estimator, or an illustration of differentiating through an ODE solver via the adjoint method). Another appendix will serve as a **glossary of terms and acronyms** – listing definitions for terms like SBI, ABC, ELBO, HMC, etc., for quick lookup. We'll also include a brief review of any prerequisite math (such as calculus basics or probability identities) in case readers need a refresher. The appendices ensure the resource is self-contained and can cater to readers of varying backgrounds.

Sources: This proposed content is informed by recent literature on simulation-based inference and surrogate modeling. For instance, the need for SBI arises because complex simulators have intractable likelihoods [1], and classical methods like ABC compare simulated vs observed summaries [8]. Modern approaches use neural density estimators (e.g. normalizing flows) as surrogates for the simulator's likelihood or posterior [10]. Surrogate models themselves are discussed in epidemiological contexts as a way to speed up calibration, often using neural networks to learn the mapping from parameters to simulation outputs [22]. Automatic differentiation is defined as techniques to compute exact derivatives of programs [17], enabling gradient-based training of neural networks via backpropagation. When models include non-differentiable components, methods like surrogate gradient (straight-through estimators) can be applied [26], or one can use reparameterization tricks to handle stochastic nodes in a differentiable way [30]. The Gumbel-Softmax approach provides a smooth approximation for categorical decisions [28], which has been utilized to create differentiable ABMs (e.g., by Quera-Bofarull et al.). All these techniques and concepts will be integrated into the structured learning path described above, ensuring a comprehensive resource that builds from fundamental statistical concepts up to the cutting-edge methods at the intersection of machine learning and epidemiological simulation modeling. The content and examples will draw on these sources and others in the field to provide both theoretical understanding and practical know-how.

---

[1] [2] [5] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [18] [20] [40] The frontier of simulation-based inference - PMC
https://pmc.ncbi.nlm.nih.gov/articles/PMC7720103/

[3] [4] [6] [17] [19] [26] [27] [28] [29] [36] [37] [38] [39] Automatic Differentiation of Agent-Based Models
https://arxiv.org/html/2509.03303v1

[21] [24] [25] Deep learning aided surrogate modeling of the epidemiological models - ScienceDirect
https://www.sciencedirect.com/science/article/abs/pii/S1877750324002631

[22] [41] [42] Integrating artificial intelligence with mechanistic epidemiological modeling: a scoping review of opportunities and challenges | Nature Communications
https://www.nature.com/articles/s41467-024-55461-x?error=cookies_not_supported&code=84e66e48-1a5e-4642-8c15-2a8fb4e49aa7

[23] [PDF] Accurate Calibration of Agent-based Epidemiological Models with ...
https://proceedings.mlr.press/v184/anirudh22a/anirudh22a.pdf

30 31 32 Reparameterization Trick for Gradient Estimation
https://www.emergentmind.com/topics/reparameterization-trick

33 34 35 REINFORCE vs Reparameterization Trick – Syed Ashar Javed – Learning to learn
https://stillbreeze.github.io/REINFORCE-vs-Reparameterization-trick/

43 [PDF] interventionally consistent surrogates for agent-based simulators
https://arxiv.org/pdf/2312.11158