Contents

1 Summary **1**

2 Motivation **1**

3 Guide-level explanation **2**

4 Reference-level explanation **2**

5 Drawbacks **5**

6 Rationale and alternatives **5**

7 Prior art **6**

8 Unresolved questions **6**

- Feature Name: ode-adjoint
- Start Date: 2021-02-02
- RFC PR: (leave this empty)
- Stan Issue: (leave this empty)

1 Summary

Ordinary differential equations (ODEs) are currently expensive to evaluate in Stan for larger systems. This is due to the multiplicative scaling of the required computing resources with the ODE system size N and the number of parameters M defining the ODE. Currently we implement in Stan the forward-mode (direct) sensitivity ODE method in order to obtain in addition to the solution of the ODE the sensitivities of the ODE. The cost of this method scales as

$$N \cdot (M + 1).$$

The computational cost of the adjoint method scales much more favorable in comparison which is

$$2 \cdot N + M.$$

The advantage of the adjoint methods shows in particular whenever the number of parameters are relatively large in comparison to the number of states. Most importantly, the computational cost grows only linearly in the number of states and parameters (while forward grows multiplicatively). Thus, this method can scale to much larger problems without a multiplicative increasing computational cost. However, the adjoint method is more involved resulting in a bigger overhead than the forward mode method. As a result, the forward method can have advantages for smaller ODE systems and as such both methods remain valuable within Stan.

2 Motivation

Stan is currently practically limited to solve problems with ODEs which are small in the number of states and parameters. If either of them gets large, the computational cost explode quickly. With the adjoint ODE solving method Stan will be able to scale to much larger problems involving more states and more parameters. Examples are large pharmacokinetic / pharmacodynamic systems or physiological based pharmacokinetic models or bigger susceptible, infected, and recovered (SIR) models. The adjoint ODE method will grow the computational cost only linearly with states and parameters and therefore modelers can more freely increase the complexity of the ODE model without substantially increasing inference times.

3 Guide-level explanation

The new solver `ode_adjoint` will follow in it's design the variadic ODE functions. As the new solver is numerically more involved (see reference level) there are more tuning parameters needed for the solver. The actual implementation will be provided by the CVODES package which we already include in Stan.

From a more internal stan math perspective, the key difference to the existing forward mode solvers will be that the gradients are not any more pre-computed during the forward sweep of reverse mode autodiff (AD). Instead, during the forward sweep of reverse mode autodiff, the ODE is solved only for the N states (without any sensitivities). When reverse mode AD then performs the backward sweep, the autodiff adjoints of the parameters of the ODE (input operands to the ODE function call) are directly computed given the parent autodiff adjoints as defined by the autodiff call graph. This calculation involves a backward solve of an adjoint ODE system with N so-called ODE adjoint states as defined by the adjoint method and explained below. Along the backward integration of the ODE adjoint problem one has in addition to solve M one-dimensional quadrature problems which depend on the adjoint states (one problem for each parameter of the ODE).

The numerical complexity is higher for the ODE adjoint method in comparison to the forward method. While most of the complexity is handled by CVODES, the numerical procedure seems to require more knowledge about the tuning parameters. At least at the moment it appears not possible to make an easy to use interface of this functionality available without most tuning parameters. We need to first collect some experience before a simpler version can be made available (if that is feasible at all). The tuning parameters exposed as proposed are:

- absolute & relative tolerance forward solve; absolute tolerance should be a vector of length N, the number of states
- absolute & relative tolerance backward solve (for consistency the absolute tolerances are given as vector)
- absolute & relative tolerance quadrature problem
- maximal number of steps between time-points; same for forward and backward integration
- forward solver: adams / bdf - 1 / 2
- backward solver: adams / bdf - 1 / 2
- checkpointing: number of checkpoints every X steps
- checkpointing: hermite or polynomial approximation - 1 / 2

During an experimental phase of this feature we can hopefully learn which of these are relevant. In order to make it easy for users to switch their existing ODE problems to the new solver an `ode_adjoint_tol` function will be provided during the experimental phase. This function call follows the same conventions as the existing ode integrators. The tuning parameters set as default will be described below. The simplified call will be integrated into the main stan math library if users find this function helpful and a reasonable default for the tuning parameters can be found during the experimental phase.

4 Reference-level explanation

The ODE adjoint method for ODEs is relatively involved mathematically. The main challenge comes from a mix of different notation and conventions used in different fields. Here we relate commonly used notation within Stan math to the user guide of CVODES, the underlying library we employ to handle the numerical solution. The goal of reverse mode autodiff is to calculate the derivative of some function $l(\theta)$ wrt. to it's parameters θ at some particular realization of θ; here θ denotes a set of parameters. For example, the likelihood is defined as the normal log probability density for a single observation at time-point T. The mean of this normal log probability density is modelled as the solution of an ODE. The ODE itself depends on some parameters p which are a subset of θ. The ODE is given as initial value problem

$$\dot{y} = f(y, t, p) \tag{1}$$

$$y(t_0) = y^{(0)}. \tag{2}$$

The ODE has N states, $y = (y_1, \ldots, y_N)$. During the reverse sweep of reverse mode autodiff we are then given the autodiff adjoints of a_{l,y_n} wrt to the output state $y(T, p)$. These are the partials of l wrt to each state

$$a_{l,y_n} = \left.\frac{\partial l}{\partial y_n}\right|_{t=T}$$

and we must calculate the contribution to the autodiff adjoint of the parameters

$$a_{l,p_m} = \sum_{n=1}^{N} a_{l,y_n} \left.\frac{\partial y_n}{\partial p_m}\right|_{t=T},$$

which involves for each parameter p_m the partial of every state y_n wrt to the parameter p_m. Note that the computational benefit of the ODE adjoint method stems from calculating the above sums *directly* in contrast to the forward method which calculates every partial $\left.\frac{\partial y_n}{\partial p_m}\right|_{t=T}$ separately. In fact, the above sum is equal to a product of a row-vector of autodiff adjoints with the transpose of the Jacobian of the ODE solution wrt to the parameters at time T.

In the notation of the CVODES user manual, the function $g(t, y, p)$ **is equal to** $l(\theta)$ essentially. Through the use of Lagrange multipliers, the ODE adjoint problem is transferred to a backward problem in equation 2.20 of the CVODES manual (see here for a step-by-step derivation),

$$\dot{\lambda} = -\left(\frac{\partial f}{\partial y}\right)^* \lambda - \left(\frac{\partial g}{\partial y}\right)^* \tag{3}$$

$$\lambda(T) = 0. \tag{4}$$

This ODE is referred to as the *backward* problem, since we fix the solution λ at the final end-point T instead of the initial condition at t_0 (it's a terminal value problem rather than an initial value problem). The CVODES manual then proceeds with deriving that

$$\left.\frac{dg}{dp}\right|_{t=T} = \mu^*(t_0)\, s(t_0) + \left.\frac{\partial g}{\partial p}\right|_{t=T} + \int_{t_0}^{T} \mu^* \frac{\partial f}{\partial p}\, dt. \tag{5}$$

Here $s(t_0)$ is the state sensitivity wrt to the parameters at the initial time-point. The term $\left.\frac{\partial g}{\partial p}\right|_{t=T}$ is the partial derivative wrt to the parameters of $g(t, y, p)$. This term is determined by the terms in $g(t, y, p)$ which directly depend on the parameters and not implicitly due to the parameters affecting the ODE state. Thus, this term is being computed by the autodiff system of stan-math itself as part of the adjoints of the parameters, a_{l,p_m}. Finally, $\mu = \frac{d\lambda}{dT}$ such that μ is obtained by the equivalent backward problem (taking the total derivative of $\dot{\lambda}$ in (3) wrt to T)

$$\dot{\mu} = -\left(\frac{\partial f}{\partial y}\right)^* \mu \tag{6}$$

$$\mu(T) = \left(\frac{\partial g}{\partial y}\right)^*_{t=T}. \tag{7}$$

If we now recall that $g(t, y, p)$ is equal to the function $l(\theta)$ being subject to autodiff we see that

$$\left(\frac{\partial g}{\partial y_n}\right)^*_{t=T} = a^*_{l,y_n},$$

which is the input we get during reverse mode autodiff (the conjugation does not apply for Stan math using reals only).

It is important to note that the backward ODE problem requires as input the autodiff adjoint a^*_{l,y_n} such that the backward problem must be solved during the reverse pass of reverse mode autodiff. Thus, CVODES is used during the forward pass to *solve (forward)* the ODE in (1). Once the backward pass is initiated, the autodiff adjoints a^*_{l,y_n} are used as the terminal value in order to solve the *backward problem* of (6). Whenever any parameter sensitivities are requested, then we must solve along the backward problem an additional one-dimensional quadrature problem per parameter of the ODE, which is the integrand in equation (5).

So far we have only considered the case of a single time-point T. For multiple time-points one starts the backward integration from the last time-point and keeps accumulating the respective terms which result from integrating backwards in steps until t_0 is reached.

In total we need to solve 3 integration problems which consist of a forward ODE problem, a backward ODE problem and a backward quadrature problem. The forward ODE and the backward ODE problem can be solved with either a non-stiff Adams or a stiff BDF solver of CVODES (the choice is independent for each problem). Each of the 3 integration problems have their own relative and absolute tolerance targets.

The proposed function should have the following signature:

```
vector[] ode_adjoint_tol_ctl(F f,
    vector y0,
    real t0, real[] times,
    real rel_tol_f, vector abs_tol_f,
    real rel_tol_b, vector abs_tol_b,
    real rel_tol_q, real abs_tol_q,
    int max_num_steps,
    int num_steps_between_checkpoints,
    int interpolation_polynomial,
    int solver_f, int solver_b
    T1 arg1, T2 arg2, ...)
```

The arguments are:
1. `f` - User-defined right hand side of the ODE (`dy/dt = f`)
2. `y0` - Initial state of the ode solve (`y0 = y(t0)`)
3. `t0` - Initial time of the ode solve
4. `times` - Sorted arary of times to which the ode will be solved (each element must be greater than t0)
5. `rel_tol_f` - Relative tolerance for forward solve (data)
6. `abs_tol_f` - Absolute tolerance vector for each state for forward solve (data)
7. `rel_tol_b` - Relative tolerance for backward solve (data)
8. `abs_tol_b` - Absolute tolerance vector for each state backward solve (data)
9. `rel_tol_q` - Relative tolerance for backward quadrature (data)
10. `abs_tol_q` - Absolute tolerance for backward quadrature (data)
11. `max_num_steps` - Maximum number of time-steps to take in integrating the ODE solution between output time points for forward and backward solve (data)
12. `num_steps_between_checkpoints` number of steps between checkpointing forward solution (data)
13. `interpolation_polynomial` can be 1 for hermite or 2 for polynomial interpolation method of CVODES
14. `solver_f` solver used for forward ODE problem: 1=Adams, 2=bdf
15. `solver_b` solver used for backward ODE problem: 1=Adams, 2=bdf
16. `arg1, arg2, ...` - Arguments passed unmodified to the ODE right hand side. The types `T1, T2, ...` can be any type, but they must match the types of the matching arguments of `f`.

In addition a simpler function version which mimics the existing ode interface for tolerances should be made available once more experience has been gained with the new ODE adjoint method. While at the current stage it is premature to make such a simpler interface available, the simpler interface is intended to let users quickly try out the new adjoint interface. **As it is currently not clear how the extra control parameters**

will be set based the function signature will not be made available in the Stan language in the first version of the adjoint ODE solver. Nontheless, once sufficient experience has been gained with the adjoint ODE method the following signature should be made available:

```
vector[] ode_adjoint_tol(F f,
    vector y0,
    real t0, real[] times,
    real rel_tol, real abs_tol,
    int max_num_steps,
    T1 arg1, T2 arg2, ...)
```

The arguments are:
1. `f` - User-defined right hand side of the ODE (`dy/dt = f`)
2. `y0` - Initial state of the ode solve (`y0 = y(t0)`)
3. `t0` - Initial time of the ode solve
4. `times` - Sorted array of times to which the ode will be solved (each element must be greater than t0)
5. `rel_tol` - Relative tolerance for all solves (data)
6. `abs_tol` - Absolute tolerance for all solves (data)
7. `max_num_steps` - Maximum number of time-steps to take in integrating the ODE solution between output time points for forward and backward solve (data)
8. `arg1, arg2, ...` - Arguments passed unmodified to the ODE right hand side. The types `T1, T2, ...` can be any type, but they must match the types of the matching arguments of `f`.

The best guesses based on preliminary experiments for the remaining tuning parameters are then set as

- same relative tolerance in all problems
- `abs_tol_f = abs_tol / 10`
- `abs_tol_b = abs_tol / 3`
- `abs_tol_q = abs_tol`
- 250 steps between checkpoints
- bdf solver for forward and backward solve
- hermite polynomial method for forward function approximation

The above best guesses are only based on early experiments and will need to be refined. This design doc intentionally does not define when "sufficient" experience has been collected until the simplified function can be made available. However, the above defaults (or an updated set) should be included in the documentation of the adjoint ODE solver.

5 Drawbacks

It's some work to be done. Other than that there are no alternatives to my knowledge to get large ODE systems working in Stan. What we are missing out for now is to exploit the sparsity structure of the ODE. This would allow for more efficient solvers and even larger systems, but this is not possible at the moment to figure out structurally the inter-dependencies of inputs and outputs.

6 Rationale and alternatives

There is no other analytical ODE solving technique available which scales in a comparable way. The better scalability of the adjoint method enables at a fixed computational resource the possibility to solve larger ODE systems with many parameters and/or states to be solved faster. Thus, a Stan modeler can increase the complexity of an ODE model under study without substantially increasing model runtimes.

The numerical complexities of the adjoint method are rather involved as 3 nested integrations are performed. This makes things somewhat fragile and less robust. What makes the backward integration in particular involved is that the solution of the forward problem must be stored as a continuous function in memory and hence an interpolation of the forward solve is required. This is provided by CVODES via a checkpointing

procedure. In summary, we do heavily rely on CVODES infrastructure as these building blocks are rather complex and heavy to craft on our own.

7 Prior art

The adjoint sensitivity method is applied within different domains of science. For example, the method is present in engineering literature and found it's way into packages like `DifferentialEquations.jl` in Julia. Another noticeable reference is FATODE and a discourse post from Ben.

Another domain where the adjoint sensitivity method is being used is in systems biology. The AMICI toolkit has been build to solve ODE system for large scale systems biology equation systems to be used within optimizer software. The adjoint method is implemented in AMICI using CVODES and has been benchmarked on various problems from systems biology as published (see also this arxiv pre-print).

However, so far the adjoint method is commonly build into special purpose software as the method requires a fixed target functional. The integration of the adjoint ODE sensitivity method into a general purpose autodiff library as Stan math is presumably the first implementation of its kind.

8 Unresolved questions

The main unresolved question at this point is to settle on the set of exposed tuning parameters. Currently the proposal is to expose in an experimental version a large super-set of tuning parameters and a version which resembles the enhanced interface with tolerances of the existing ODE solvers. This allows users to quickly try out the adjoint method by merely changing a function name. The suggestion is to collect feedback from the Stan community during an experimental phase of this feature. Once feedback on the utility of the simplified interface and the appropriateness of the set defaults are collected a decision should be done about including this signature in a first version included in the next release. Another goal of the experimental phase is to explore the possibility to weed out some tuning parameters if possible or add others if needed.