Ordinary Differential Equations

Differential equations are ubiquitous in engineering disciplines and the natural sciences, where it is often easier to characterize some quantity of interest by how it *changes* in response to other quantities in the system. The classic example of a differential equation is Newton's second law of motion which can be written as

$$m\frac{d^2x(t)}{dt^2} = F\big(x(t)\big)$$

where m is the mass of some object, $\frac{d^2x(t)}{dt^2}$ is the second derivative of it's position with respect to time aka it's acceleration, and $F\big(x(t)\big)$ is the force being applied to the object, which in this representation depends on the object's position.

Many commonly-occurring differential equations cannot be solved exactly; intuitively, this happens when the derivative is given by a function whose antiderivative does not exist. Many classical, numerical techniques have been developed to approximate the solutions for such intractable differential equations. We will present a few of the most common methods and then provide a probabilistic alternative. Finally, we will demonstrate an interesting and insightful connection between our probabilistic method and one of the classical methods.

For the purposes of these notes, we will restrict our focus to *first-order*, *ordinary* differential equations (ODEs). Ordinary differential equations contain only a single "independent variable" i.e., the functions of interest and their corresponding derivatives can be expressed in terms of just one input. This is in contrast to functions of multiple variables, which give rise to *partial* differential equations (PDEs); these are interesting in their own right, but solving them requires significantly more advanced techniques that are outside the scope of this course. Notationally, it is commonplace to represent the single independent variable as t, a convention we shall adopt below.

First-order differential equations only describe the behavior of the first-order derivatives of the variables. In general, it is possible to rewrite higher-order ODEs as systems of first-order ODEs by introducing intermediate variables e.g.,

$$\frac{d^2y(t)}{dt^2} = f\big(t, y(t)\big) \;\rightarrow\; \frac{dy(t)}{dt} = z\big(t, y(t)\big) \text{ and } \frac{dz(t)}{dt} = f\big(t, y(t)\big).$$

In principle, such systems of first-order ODEs can also be solved using extensions of the methods presented below.

We will further limit our discussion to so-called *initial-value problems* (IVPs). In many cases, simply being given an expression for the derivative of a variable is not sufficient to fully determine its functional form; an observation of the function itself is required to fully specify a unique solution. Again, many relevant problems from scientific domains are naturally defined as IVPs.

Formally, we will consider solving problems of the following form: we are given that

$$\frac{dy(t)}{dt} = f\big(t, y(t)\big) \text{ and } y(t_0) = y_0$$

for some function f and initial value (t_0, y_0). The goal is to find a function $y(t)$ that satisfies the conditions above.

The Euler Method

One approach for approximating the solution to these IVPs relies on the finite difference approximation for a derivative:

$$\frac{dy(t)}{dt} \approx \frac{y(t+\varepsilon) - y(t)}{\varepsilon}.$$

Rearranging the equation above and substituting back in the provided definition for $\frac{dy(t)}{dt}$ gives the result

$$y(t+\varepsilon) \approx y(t) + \varepsilon f\big(t, y(t)\big).$$

If we pick a step size ε and plug our initial value (t_0, y_0) into the second term, we can approximate the function's value at any location $t_n = t_0 + n\varepsilon$ for integral n.

Example (from Wikipedia)

Suppose we want to solve the (trivial) IVP:

$$\frac{dy(t)}{dt} = y(t) \text{ and } y(0) = 1.$$

First, observe that the unique solution is $y(t) = e^t$. If we pick a rather convenient step size of $\varepsilon = 1$, we can construct the following sequence of approximations:

n	t_n	$y(t_n)$	$f\big(t_n, y(t_n)\big)$	e^{t_n}	approximation error
0	0	1	1	1	0
1	1	2	2	e	0.7183
2	2	4	4	e^2	3.3891
3	3	8	8	e^3	12.0855
4	4	16	16	e^4	38.5981

Clearly, this approximation is quite poor and relatively coarse: not only does it do a bad job of estimating the function's value, it only generates estimates at integer locations. We can address both of these issues by decreasing the step size. The following table shows the approximation of e^4 for different values of ε, along with the corresponding error:

ε	$y(4)$	approximation error
1	16	38.5981
0.25	35.5283	19.0710
0.1	45.2593	9.3389
0.025	51.9779	2.6203

Of course, this increased accuracy is not without cost: in general, decreasing the step size requires more computational effort as we need to take more steps to get to the desired estimates. Another way of improving the estimate is to use a more sophisticated approximation.

The Midpoint Method

One such alternative is the *midpoint* method, which leverages a slightly different expression for the finite difference approximation:

$$\frac{dy(t+\varepsilon/2)}{dt} \approx \frac{y(t+\varepsilon)-y(t)}{\varepsilon}.$$

Again, solving the equation above and substituting in the given expression for $\frac{dy(t)}{dt}$ gives

$$y(t+\varepsilon) \approx y(t) + \varepsilon f\big(t+\varepsilon/2, y(t+\varepsilon/2)\big).$$

Here we encounter an issue: if we want to compute $y(t_0+\varepsilon)$, we need both $y(t_0)$, which is given to us, and $y(t_0+\varepsilon/2)$, which we do not have. Thus, the midpoint method makes an additional approximation using the Euler method! Specifically, we will replace $y(t+\varepsilon/2)$ in the equation above with

$$y(t+\varepsilon/2) \approx y(t) + \frac{\varepsilon}{2} f\big(t, y(t)\big).$$

This gives the final result:

$$y(t+\varepsilon) \approx y(t) + \varepsilon f\Big(t+\varepsilon/2, y(t) + \frac{\varepsilon}{2} f\big(t, y(t)\big)\Big).$$

Example Revisited

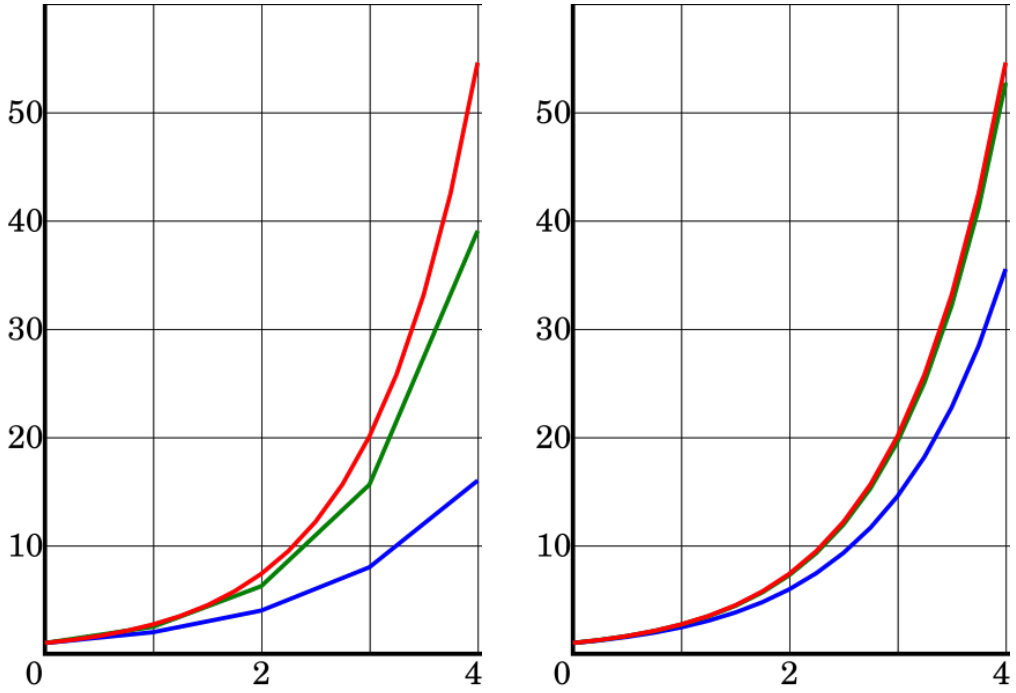
Applying the midpoint method to solve the IVP for $y(t) = e^t$ with a step size of $\varepsilon = 1$, we get the following, notably better sequence of approximations:

n	t_n	$y(t_n)$	$f\Big(t_n+\varepsilon/2, y(t_n)+\frac{\varepsilon}{2}f\big(t_n,y(t_n)\big)\Big)$	e^{t_n}	approximation error
0	0	1	1.5	1	0
1	1	2.5	3.75	e	0.2183
2	2	6.25	9.375	e^2	1.1391
3	3	15.625	23.4375	e^3	4.4605
4	4	39.0625	58.5938	e^4	15.5357

Much as before, we can improve the quality of this approximation significantly by decreasing the step size if we are willing to do a bit of extra computation. Figure 1 compares the midpoint method with the Euler method for step sizes of 1 and 0.25

Runge-Kutta Methods

The Euler method and the midpoint method as described above are instances of a broader family of algorithms for approximately solving IVPs known as *Runge-Kutta* methods. Specifically, the Euler method is a first-order Runge-Kutta method as its approximation error is proportional to ε^1 while the midpoint method is a second-order Runge-Kutta method as its error is proportional to ε^2.



(a) $\varepsilon = 1$ (b) $\varepsilon = 0.25$

Figure 1: A comparison of the Euler method and the midpoint method for solving the IVP described in the main text: the red line corresponds to the true function, $y(t) = e^t$, while the blue and green lines correspond to the Euler method's approximation and the midpoint method's approximation respectively. Both approximations improve as the step size decreases, with the midpoint method being a nearly perfect fit for $y(t)$ over the plotted domain.

The family of Runge-Kutta methods are defined by update rules of the following form:

$$
\begin{aligned}
y(t+\varepsilon) &= y(t) + \varepsilon \sum_{s=1}^{S} b_s k_s \text{ where} \\
k_1 &= f\big(t, y(t)\big) \\
k_2 &= f\big(t + c_2\varepsilon, y(t) + (a_{21}k_1)\varepsilon\big) \\
k_3 &= f\big(t + c_3\varepsilon, y(t) + (a_{31}k_1 + a_{32}k_2)\varepsilon\big) \\
&\vdots \\
k_S &= f\left(t + c_S\varepsilon, y(t) + \left(\sum_{i=1}^{S-1} a_{Si}k_i\right)\varepsilon\right)
\end{aligned}
$$

Intuitively, Runge-Kutta methods iteratively estimate the function at increasing intervals of ε, much like the Euler method and the midpoint method, except instead of using a single derivative value to determine how the function's value changes, they use a weighted sum or average of S derivatives (k_1, k_2, ..., k_S), also called the *stages*

Runge-Kutta methods are frequently described by so-called *Butcher tableaus*, which are a convenient way of storing/presenting the relevant coefficients; a Butcher tableau for the generalized Runge-Kutta method given by the equations above would be

$$\begin{array}{c|cccc} 0 & & & & \\ c_2 & a_{21} & & & \\ c_3 & a_{31} & a_{32} & & \\ \vdots & \vdots & \vdots & \ddots & \\ c_S & a_{S1} & a_{S2} & \cdots & a_{S\ S-1} \\ \hline & b_1 & b_2 & \cdots & b_{S-1} \\ b_S & & & & \end{array}$$

As an example, we can express the Euler method as a one-stage Runge-Kutta method using the following Butcher tableau:

$$\begin{array}{c|c} 0 & \\ \hline & 1 \end{array}$$

Similarly, the Butcher tableau for the midpoint method is defined is

$$\begin{array}{c|cc} 0 & & \\ \frac{1}{2} & \frac{1}{2} & \\ \hline & 0 & 1 \end{array}$$

There are many methods and/or heuristics for choosing the number of setting the coefficients; a few commonly cited ones include:

- For a Runge-Kutta method to have order p (i.e., its approximation error is $O(\varepsilon^p)$), s must be at least p and for $p \geq 5$, s must be strictly larger than p.
- For a Runge-Kutta method to be *consistent* (i.e., its approximation error at each time-step approaches 0 as $\varepsilon \to 0$), $\sum_{s=1}^{S} b_s$ must be equal to 1.
- It is common to set $c_i = \sum_{j=1}^{i-1} a_{ij}$ although there is not theoretical justification for doing so.

The most commonly-used, higher-order Runge-Kutta method is called the "classic Runge-Kutta method" or "RK4" as it is a 4-stage method. It is given by the following Butcher tableau:

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

To better interpret this method, we can convert this Butcher tableau back into an explicit update formula as follows:

$$
\begin{aligned}
y(t+\varepsilon) &= y(t) + \frac{\varepsilon}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) \text{ where} \\
k_1 &= f\big(t, y(t)\big) \\
k_2 &= f\Big(t + \frac{\varepsilon}{2}, y(t) + \frac{k_1}{2}\varepsilon\Big) \\
k_3 &= f\Big(t + \frac{\varepsilon}{2}, y(t) + \frac{k_2}{2}\varepsilon\Big) \\
k_4 &= f\big(t + \varepsilon, y(t) + k_3\varepsilon\big)
\end{aligned}
$$

Written out in this manner, we can see that "RK4" is approximating the derivatives as a weighted average of four different derivatives measurements:

- k_1, the derivative at the current location t,
- k_2, the derivative at the midpoint between t and $t + \varepsilon$, approximated using $y(t)$ and k_1,
- k_3, the derivative at the midpoint between t and $t + \varepsilon$, approximated using $y(t)$ and k_2, and
- k_4, the derivative at the next location $t + \varepsilon$;

the weights emphasize the derivatives at the midpoint over the derivatives at either end of the current interval. This specific method derives its popularity from a couple of factors: it was one of the first methods developed by Carl Runge and Wilhelm Kutta in the early 1900s and represents a reasonable trade-off between computational cost and accuracy.

Gaussian Processes for IVPs

We now turn our attention to a probabilistic alternative to the classical, numerical methods introduced above. Specifically, we will consider how to apply Gaussian processes (GPs) to infer the function of interest in IVPs, y. At first glance, this appears to be trivial: as we have previously seen, GP beliefs can be conditioned on derivative or gradient observations in effectively the same way we would condition on function observations directly.

However, the key limitation of IVPs is that we are only given a single function evaluation, the initial value (t_0, y_0). We can use this initial value to (exactly) compute the derivative at t_0 but beyond that, all other observations must be approximated. The classical methods presented above implicitly rely on linear extrapolations based on the finite difference approximation of a derivative to iteratively build up a set of observations. Inspired by this approach, similar ideas have been developed for iteratively constructing a set of data points to condition the GP belief with.

Given a step size, ε, the high-level approach is as follows:

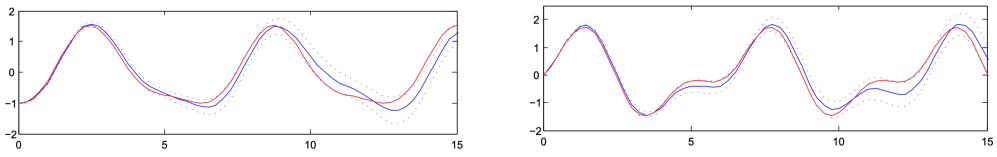
1. Place a (typically zero-mean) GP prior on y: $y \sim \mathcal{GP}(0, k)$.
2. Condition the prior belief on the initial value and derivative, $y(t_0) = y_0$ and $\frac{dy(t_0)}{dt} = f(t_0, y_0)$; let the union of these observations be $\mathcal{D}_0$.
    - These observations are typically treated as exact or noiseless: $\sigma_0^2 = 0^2$.

3. Compute the posterior mean and variance of $y(t+\varepsilon)$, $\mu_{\mathcal{D}_0}(t+\varepsilon)$ and $k_{\mathcal{D}_0}(t+\varepsilon, t+\varepsilon)$ respectively.
4. Derive an estimate of $y(t+\varepsilon) \coloneqq y_1$ using the posterior belief above; two common ways of doing so are
   (a) Sample a value the posterior distribution: $y_1 \sim \mathcal{N}\big(\mu_{\mathcal{D}_0}(t+\varepsilon), k_{\mathcal{D}_0}(t+\varepsilon, t+\varepsilon)\big)$
   (b) Simply use the posterior mean: $y_1 = \mu_{\mathcal{D}_0}(t+\varepsilon)$
5. Use y_1 to compute the derivative: $\frac{dy(t+\varepsilon)}{dt} \approx f(t+\varepsilon, y_1)$.
6. Note that both y_1 and $\frac{dy(t+\varepsilon)}{dt}$ are approximations and so rather than treating them as noiseless, the typical approach is to assume *heteroscedastic* noise or a noise variance that is not constant across observations. A reasonable choice for the noise variance in this setting is the posterior variance $\frac{dy(t+\varepsilon)}{dt}$ implied by the GP belief: $\frac{d}{dx}|_{t+\varepsilon}\Big(\frac{d}{dx'}|_{t+\varepsilon}\, k_{\mathcal{D}_0}(x, x')\Big) \coloneqq \sigma_1^2$.
   - Conditioning a GP belief on *independent* heteroscedastic noise is easy: simply add the observation-specific noise variance to the corresponding diagonal element of the gram matrix i.e., instead of adding a fixed σ^2 to each diagonal entry, add σ_i^2 to the i^{th} element.
   - Of course, it is reasonable to question whether or not the noise variances are truly independent across steps but for the sake of feasibility, we will assume that this is a reasonable approximation.
7. Discard y_1 and add the (approximate) derivative observation to the dataset: $\mathcal{D}_1 = \mathcal{D}_0 \cup \Big(\frac{dy(t+\varepsilon)}{dt} \approx f(t+\varepsilon, y_1)\Big)$.
8. Repeat the process above until some desired location $t + n\varepsilon$ has been reached.

Figure 2 from Barber (2014) shows the result of applying the method described above on the IVP

$$\frac{dy(t)}{dt} = -y(t) + \sin(2t) \text{ and } y(0) = -1.$$

In general, the true function (left) and it's derivative (right) are both well-modeled by the GP belief but the quality of fit noticeably decreases as we move further from the initial value.



(a) $y(t)$ (b) $\frac{dy(t)}{dt}$

Figure 2: GP inference for approximating the solution to the IVP described in the main text: the red line shows the true function $y(t)$ while the solid blue line is the posterior GP mean and the dotted blue lines represent a 2-standard deviation credible interval about the mean. A step size of $\varepsilon = 0.25$ was used to derive intermediate observations.

Multiple extensions/variants of the simple routine above have been proposed and explored in the literature including:

- Limiting the number of previous derivative observations that are conditioned on; this is largely done for computational purposes as decreasing the step size has an out-sized impact on the runtime of GP inference given it's cubic dependence on the number of data points.
- Refining the belief at $y(t + c\varepsilon)$ by resampling a new value at that location *after* conditioning on the derivative $\frac{dy(t+c\varepsilon)}{dt} \approx f(t + c\varepsilon, y_c)$.
- Constraining the belief at intermediate steps $y(t + c\varepsilon)$ to (approximately) respect the observation $\frac{dy(t+c\varepsilon)}{dt} = f(t + c\varepsilon, y_c)$ using EP instead of heteroscedastic noise.

Relationship to Runge-Kutta Methods

Finally, observe that Runge-Kutta methods result in a piecewise linear approximation for the function $y(t)$. It is a well-known fact that GPs are capable of learning piecewise linear posterior means given the right choice of kernel e.g., Ornstein-Uhlenbeck processes which use covariance functions of the form $k(x, x') = \exp\left(-\frac{|x-x'|}{\ell}\right)$.

Inspired by this observation, Schober et al. (2014) showed that all first-, second- and third-order Runge-Kutta methods can be exactly recreated as the posterior mean of a GP when conditioned on the same set of locations that are visited by the Runge-Kutta methods. This gives rise to a probabilistic interpretation of Runge-Kutta methods which can be used to evaluate model choices and perform robust uncertainty quantification of the approximations.