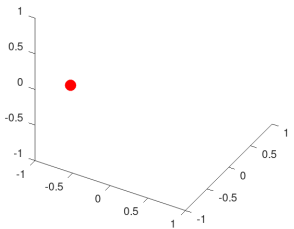
Euler's Method, Shortcomings, and Higher Order Methods

MATH 333

Matt Dallas

October 28, 2025

Slides and Code



Why Study Numerical Methods?

Khan: There's this lovely part in the movie in which your character turns to "old" math — **Euler's method** — to figure out how to get John Glenn back down from orbit. Did that really happen?

Johnson: It seemed logical to me. I could see in my mind what I needed and sort of worked backwards.

– 2017 LA Times interview of Katherine Johnson by Amina Khan.

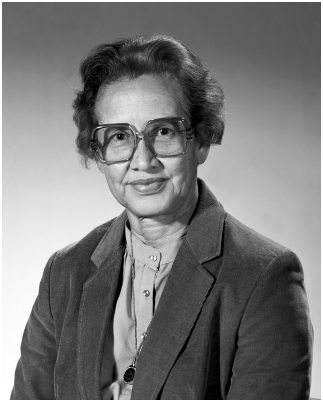


Figure: Katherine Johnson in 1983. Image in Public Domain.

Outline

Euler's Method

Last time, we learned a technique to compute an approximate numerical solution to an **initial-value problem (IVP)** of the form

$$\begin{cases} y'(x) = f(x, y), & x \text{ in } [a, b] \\ y(a) = y_0 \end{cases} \tag{1}$$

We also learned that this technique can be applied to *systems* of ODEs of the form

$$\begin{cases} \mathbf{y}'(x) = \mathbf{f}(x, \mathbf{y}), & x \text{ in } [a, b] \\ \mathbf{y}(a) = \mathbf{y}_0 \end{cases} \tag{2}$$

Euler's Method

Consider a single IVP $y'(x) = f(x, y)$ with x in $[a, b]$ subject to the initial condition $y(a) = y_0$.

The idea behind the numerical methods we will cover is that instead of computing a solution $y(x)$ at *every* x in $[a, b]$, we select a *discrete* set of points

$$a = x_0 < x_1 < \cdots x_{N-1} < x_N = b,$$

called **nodes**, and compute *approximate* values u_n of the solution $y(x_n)$ for $n = 0, \ldots, N$. That is,

$$u_n \approx y(x_n)$$

for $n = 0, \ldots, N$. We will often write $y_n = y(x_n)$. The spacing between nodes is called the **step-size**, and we will assume a uniform step-size of

$$h = \frac{b - a}{N}.$$

Euler's Method

There is one node at which we know the exact value of the solution: the left endpoint $a = x_0$. So $u_0 = y_0$. We may then compute $u_1 \approx y_1$ by Taylor expanding y_1 about x_0:

$$y_1 = y_0 + y_0' h + R(h)$$

Throwing away the remainder term, $R(h)$, yields an approximation $u_1 \approx y_1$:

$$u_1 = y_0 + y_0' h \approx y_1.$$

Lastly, writing $y_0' = f(x_0, y_0)$ and $u_0 = y_0$, we obtain the first step of Euler's method:

$$u_1 = u_0 + hf(x_0, u_0).$$

Euler's Method

Repeating this process at each node yields Euler's method:

$$u_{n+1} = u_n + hf_n,$$

where $f_n = f(x_n, u_n)$. This is also known as the **forward Euler method**.

Geometric View of Euler's Method

Geometrically, Euler's method at step n can be described as follows:

1. Construct the line, $L(t)$, parallel to the line tangent to y at x_n that passes through (x_n, u_n). Thus [1]
$$L(t) = u_n + (t - x_n)f_n$$
2. Walk h units along the tangent line starting from x_n, i.e., set $t = x_n + h$, and take u_{n+1} to be the y-coordinate of the point you land on. In other words,
$$u_{n+1} = L(x_n + h) = u_n + hf_n.$$

[1] Remember that, given a coordinate point (x, y), the function $f(x, y)$ gives us the derivative of y at x.

Code Demonstration

Let's look at exp_model.m to see Euler's method in action. In this program, we aim to solve the IVP

$$\begin{cases} y'(x) = ky(x), & x \text{ in } [0, 6] \\ y(0) = 1. \end{cases} \tag{3}$$

Task 1: Use the Forward Euler Method to solve this IVP with $k = 1$. First use 10 nodes, and then use 30 nodes.

What do you observe?

Code Demonstration

Let's look at exp_model.m to see Euler's method in action. In this program, we aim to solve the IVP

$$\begin{cases} y'(x) = ky(x), & x \text{ in } [0, 6] \\ y(0) = 1. \end{cases} \tag{4}$$

Task 2: Now use the Forward Euler Method to solve this IVP with $k = -3$. First use 9 nodes, then 10, and then 11.

What do you observe?

What About Systems?

Consider the following setup:

1. We attach a unit mass ($m = 1$) to a spring on a surface with no friction.
2. We allow the mass to reach equilibrium, we then pull it back 1 unit from this position and release.

If $x(t)$ denotes the position of the mass relative to its equilibrium at time $t \geq 0$ since we set the mass in motion, Newton's second law tells us that $x(t)$ satisfies

$$x''(t) + x(t) = 0 \tag{5}$$

with $x(0) = 1$ and $x'(0) = 0$.

What About Systems?

To use Euler's method, we first recast this **second order** differential equation as a **first order** system. Let's say $v = x'$. Then $x''(t) + x(t) = 0$ is equivalent to

$$\begin{cases} x'(t) = v(t), & x(0) = 1 \\ v'(t) = -x(t), & v(0) = 0. \end{cases} \tag{6}$$

Written in matrix-vector form:

$$\begin{bmatrix} x(t) \\ v(t) \end{bmatrix}' = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} x(t) \\ v(t) \end{bmatrix}, \quad \begin{bmatrix} x(0) \\ v(0) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{7}$$

We thus obtain an equation of the form (2) with $\mathbf{y}(t) = [x(t) \;\; v(t)]^T$ and $\mathbf{f}(x, \mathbf{y}) = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \mathbf{y}$.

What About Systems?

Let's jump into harmonic_oscillator.m to see Euler's method applied to the system (7).

Task 1: Use the Forward Euler Method to solve equation (7) on the interval $[0, 5]$ using $50, 100,$ and 200 nodes.

What do you observe?

What About Systems?

Let's jump into harmonic_oscillator.m to see Euler's method applied to the system (7).

Task 1: Use the Forward Euler Method to solve equation (7) on the interval $[0, 5]$ using $50, 100,$ and 200 nodes.

What do you observe?

Task 2: Use the Forward Euler Method to solve equation (7) on the interval $[0, 20]$ using $50, 100,$ and 200 nodes.

What do you observe?

Code demonstration observations

1. As we increase the number of nodes, or, equivalently, decrease the step-size h, Forward Euler appears to yield more accurate results.

2. When $[a, b]$ is "large," we seem to need a smaller step-size to obtain a reasonable solution over the entire interval.

Let's see if we can explain these observations theoretically.

Error in Euler's Method

To understand the error in Euler's method, let's sketch a picture. On your own paper, please do the following.

1. Sketch the function $y = e^x$.
2. Starting from the coordinate point $(0, 1)$, draw the first two steps of Euler's method. Using a larger step-size will be helpful for what comes next.

Error in Euler's Method

To understand the error in Euler's method, let's sketch a picture. On your own paper, please do the following.

1. Sketch the function $y = e^x$.
2. Starting from the coordinate point $(0, 1)$, draw the first two steps of Euler's method. Using a larger step-size will be helpful for what comes next.
3. Suppose that we know $y(x_1) = y_1$ exactly. Draw an Euler step from (x_1, y_1).

Error in Euler's Method

To make our picture more formal, if we let $u_n^* = y_{n-1} + hf(x_{n-1}, y_{n-1})$, then we can write the **error** $E_n = y_n - u_n$ at step n as

$$E_n = \left(y_n - u_n^*\right) \quad + \quad \left(u_n^* - u_n\right) \tag{8}$$

1. $y_n - u_n^* = y_n - y_{n-1} - hf(x_{n-1}, y_{n-1})$ is the error due to **truncation**, i.e., cutting off the Taylor Expansion after the first derivative term.

2. $u_n^* - u_n = y_{n-1} + hf(x_{n-1}, y_{n-1}) - \left(u_{n-1} + hf(x_{n-1}, u_{n-1})\right)$ is the error accrued by using our approximate value u_{n-1} for the solution at x_{n-1} in place of the exact value y_{n-1}.

Error in Euler's Method

If we assume that $df(x,y)/dx = y''(x)$ is continuous, then the remainder term in the Taylor expansion

$$y_n = y_{n-1} + hf(x_{n-1}, y_{n-1}) + R(h)$$

may be bounded by

$$|R(h)| \leq Mh^2,$$

where M is a constant that is independent of h determined by the max value of $df(x, y(x))/dx$ in $[a, b]$.

So one part of the overall error, $y_n - u_n^*$, the **local truncation error**, can be bounded by

$$|y_n - u_n^*| \leq Mh^2.$$

The other part of the error, $u_n^* - u_n$, on the other hand, has a significant effect on the maximum error over $[a, b]$.

Error in Euler's Method

Indeed, one can show that the maximum error across all nodes $x_0, x_1, \ldots, x_N$, which we denote by $E_{\max}$, is bounded by

$$|E_{\max}| \leq \left[\frac{M}{L}\left(e^{L(b-a)} - 1\right)\right] h \tag{9}$$

where

- M is the same constant from earlier determined by $df(x, y(x))/dx$, and
- $L = \max |\partial f(x, y)/\partial y|$ over a region containing our solution curve.

Both constants are independent of the step-size h.

Error in Euler's Method

Indeed, one can show that the maximum error across all nodes $x_0, x_1, \ldots, x_N$, which we denote by $E_{\max}$, is bounded by

$$|E_{\max}| \leq \left[\frac{M}{L}\left(e^{L(b-a)} - 1\right)\right] h \tag{10}$$

This bound on the maximum error explains both observations we made earlier:

1. Since the term $M\left(e^{L(b-a)} - 1\right)/L$ is independent of h, $|E_{\max}|$ decreases as h decreases.
2. The term $e^{L(b-a)}$ significantly magnifies the error, with greater magnification the larger $b - a$ is.

Note: we may perform the same analysis and obtain an identical bound for the error in the case of systems. The only difference is we replace the absolute value $|\cdot|$ with a vector **norm** $\|\cdot\|$.

The Order of a Numerical Method

When analyzing numerical methods, we often obtain an expression of the form

$$|E_{\max}| \leq Ch^p$$

where C is some constant independent of h, and h is parameter we can adjust. In the the case of Euler's method, and the other numerical schemes we will see today, h is the step-size.

The number p is a positive integer, and if the error for a given method satisfies, $|E_{\max}| \leq Ch^p$, we say that the method is **order p**.

Based on our earlier analysis, we can therefore say that Euler's method is of order 1, or it's a **first order** method.

Numerically Verifying the Order

How can we "see" the order of convergence?

If we consolidate $M(e^{L(b-a)} - 1)/L$ into a single constant, C, then we have $|E_{\max}| \leq Ch$ for Euler's method. It follows that, if $\log(x)$ is the base-10 logarithm,

$$\log(|E_{\max}|) \leq \log(C) + \log(h).$$

This means that if we plot our error using the base-10 logarithm scale for each axis, we should see a line of slope 1.

Let's check out error_rate.m to see what we see.

Higher Order Methods

Our goal now is to study methods that are of a higher order than Euler's method.

Higher Order Methods

How does one develop a more accurate method?

Higher Order Methods

How does one develop a more accurate method?

Let's briefly revisit Euler's method from a different perspective. When solving a differential equation, we are ultimately undoing a derivative, i.e., we are integrating.

Consider the IVP $y'(x) = f(x, y)$ over $[a, b]$ subject to $y(a) = y_0$. We again select equally spaced nodes $a = x_0, \ldots, x_N = b$ with step-size $h = (b - a)/N$. Starting from $u_0 = y_0$, we can obtain $u_1 \approx y_1$ by integrating our ODE and using the Fundamental Theorem of Calculus:

$$\int_{x_0}^{x_1} y'(x)\,dx = \int_{x_0}^{x_1} f(x, y(x))\,dx$$
$$y_1 - u_0 = \int_{x_0}^{x_1} f(x, y(x))\,dx.$$

Higher Order Methods

If we approximate the integral of f using a **left-endpoint approximation**, we obtain the first step of Euler's method

$$y_1 \approx u_1 = u_0 + f(x_0, y(x_0))h.$$

This suggests one way to obtain a more accurate approximation: *use a more accurate integral approximation!*

Higher Order Methods

If we approximate the integral of f using a **left-endpoint approximation**, we obtain the first step of Euler's method

$$y_1 \approx u_1 = u_0 + f(x_0, y(x_0))h.$$

This suggests one way to obtain a more accurate approximation: *use a more accurate integral approximation!*

- The error E from using left and right-endpoint approximations satisfies

$$E \leq Ch^2,$$

  where C depends on the function and h is the step-size.
- The **trapezoid rule**, on the other hand, has error $E_T \leq Ch^3$. This is better!
- In the homework, you will explore what happens if we use another integrator of similar accuracy to the trapezoid rule: the **midpoint rule**.

Higher Order Methods

Going back to

$$y_1 = u_0 + \int_{x_0}^{x_1} f(x, y(x))\, dx,$$

approximating the integral with the Trapezoid rule yields

$$y_1 \approx u_1 = u_0 + \frac{h}{2}\left[f(x_0, y_0) + f(x_1, u_1)\right],$$

and for $n \geq 0$:

$$u_{n+1} = u_n + \frac{h}{2}\left[f(x_n, u_n) + f(x_{n+1}, u_{n+1})\right],$$

Higher Order Methods

Looking at

$$u_{n+1} = u_n + \frac{h}{2}\left[f(x_n, u_n) + f(x_{n+1}, u_{n+1})\right],$$

we may notice that:

Higher Order Methods

Looking at

$$u_{n+1} = u_n + \frac{h}{2}\left[f(x_n, u_n) + f(x_{n+1}, u_{n+1})\right],$$

we may notice that:

1. We are using derivative information at *two* points, x_n and x_{n+1}, rather than at one point like we did with Euler's method.
2. The value we want to compute at step n, u_{n+1}, appears on *both* sides of the equation. That is, we have expressed u_{n+1} as an **implicit** function of u_n and x_n.

Order 2 Runge-Kutta

To obtain an expression for u_{n+1} as an **explicit** function of u_n and x_n, we may replace u_{n+1} with a Forward Euler step. This gives us the **Order 2 Runge-Kutta method**, or **RK2**:

$$u_{n+1} = u_n + \frac{h}{2}\left[f(x_n, u_n) + f(x_n + h, u_n + hf_n)\right] \tag{11}$$

We may also write step n of RK2 as

1. $s_1 = f(x_n, u_n)$
2. $s_2 = f(x_n + h, u_n + hs_1)$
3. $u_{n+1} = u_n + h\left(\frac{s_1 + s_2}{2}\right)$

Benefit of explicit methods: no need to solve a (possibly nonlinear) equation for u_{n+1}.

RK2 Error

The error in RK2 is very similar to that of Euler's method, except that we gain an extra h:

$$\text{max RK2 error} \leq \left[\frac{M}{L}\left(e^{L(b-a)} - 1\right)\right] h^2. \tag{12}$$

- ✓ Order 2 in h, and therefore significantly more accurate than Euler.
- ✓ Simple to implement.
- ✓ Explicit—no equation to solve.
- × Requires two evaluations of f. This could be expensive for some problems.
- × Still have $e^{L(b-a)}$ in error bound, which means our solution will become unreliable for long intervals.

RK2 Error

The error in RK2 is very similar to that of Euler's method, except that we gain an extra h:

$$\text{max RK2 error} \leq \left[\frac{M}{L}\left(e^{L(b-a)} - 1\right)\right] h^2. \tag{12}$$

- ✓ Order 2 in h, and therefore significantly more accurate than Euler.
- ✓ Simple to implement.
- ✓ Explicit—no equation to solve.
- × Requires two evaluations of f. This could be expensive for some problems.
- × Still have $e^{L(b-a)}$ in error bound, which means our solution will become unreliable for long intervals.

Let's see how RK2 and Euler compare in practice with the harmonic oscillator!

Can we do even better?

The success of RK2 suggest that *more derivative information yields more accurate methods.*

This idea leads to a whole of family of methods. Each member of this family is characterized by the derivatives chosen, and how much weight we assign them.

In RK2, we use two derivatives: $s_1 = f(x_n, u_n)$ and $s_2 = f(x_{n+1}, u_n + hf_n)$, and gave them equal weight.

Can we do even better?

The success of RK2 suggest that *more derivative information yields more accurate methods.*

This idea leads to a whole of family of methods. Each member of this family is characterized by the derivatives chosen, and how much weight we assign them.

In RK2, we use two derivatives: $s_1 = f(x_n, u_n)$ and $s_2 = f(x_{n+1}, u_n + hf_n)$, and gave them equal weight.

What if we used *more* derivatives?

Order 4 Runge-Kutta

The **fourth order Runge-Kutta method**, **RK4**, or simply **the** Runge-Kutta method, is defined by the following process at each step n:

1. $s_1 = f(x_n, u_n)$
2. $s_2 = f(x_n + h/2, u_n + hs_1/2)$
3. $s_3 = f(x_n + h/2, u_n + hs_2/2)$
4. $s_4 = f(x_n + h, u_n + hs_3)$
5. $u_{n+1} = u_n + h\left(\frac{s_1 + 2s_2 + 2s_3 + s_4}{6}\right)$

As the name suggests, this is a *fourth order method*:

$$\text{max RK4 error} \leq \left[\frac{M}{L}(e^{L(b-a)} - 1)\right] h^4$$

Order 4 Runge-Kutta

$$\text{max RK4 error} \leq \left[\frac{M}{L}\left(e^{L(b-a)} - 1\right)\right] h^4$$

- ✓✓ Order 4 in h, and therefore significantly more accurate than both Euler and RK2.
- ✓ Relatively simple to implement.
- ✓ Explicit—no equation to solve.
- × Requires *four* evaluations of f. This could be expensive for some problems.
- × Still have $e^{L(b-a)}$ in error bound, which means our solution will become unreliable for long intervals, but RK4 is accurate for much longer than RK2 or Euler.

Order 4 Runge-Kutta

$$\text{max RK4 error} \leq \left[\frac{M}{L}\left(e^{L(b-a)} - 1\right)\right] h^4$$

- ✓✓ Order 4 in h, and therefore significantly more accurate than both Euler and RK2.
- ✓ Relatively simple to implement.
- ✓ Explicit—no equation to solve.
- × Requires *four* evaluations of f. This could be expensive for some problems.
- × Still have $e^{L(b-a)}$ in error bound, which means our solution will become unreliable for long intervals, but RK4 is accurate for much longer than RK2 or Euler.

Order 4 Runge-Kutta

Task: Compare the solutions obtained from Forward Euler, RK2, and RK4 for the Harmonic Oscillator using $N = 50$ nodes.

Task: What happens if you increase the length of the interval, but keep $N = 50$? Try setting $b = 60$, and then $b = 100$.

Which method to use?

- In general, there is no single “best" numerical method for every problem you may encounter. Each method has its strengths and weaknesses.
- Of the methods discussed here, RK4 is by far the most accurate, but it requires additional function evaluations that may be too costly for some problems.
- Forward Euler and RK2 require fewer function evaluations, but are less accurate.
- RK2 remains accurate for longer than Euler with the same step-size.

What We Haven't Said

There are many, many numerical methods for ODEs not discussed here. For example,

- We have discussed the *forward* Euler's method, but there is also the **Backward** Euler method that is order one like forward Euler, but is much more *stable*.
- If in the derivation of RK2, we kept u_{k+1} in place of $u_n + hf_n$, we obtain what is known as the **Crank-Nicolson** method.
- ODE solver packages in MATLAB such as the popular ode23 and ode45 use combinations of Runge-Kutta methods to solve an ODE efficiently and guarantee that a user-specified tolerance is reached.

Note: Both Backward Euler and Crank-Nicolson are included in the Github repository if you would like to experiment with them.