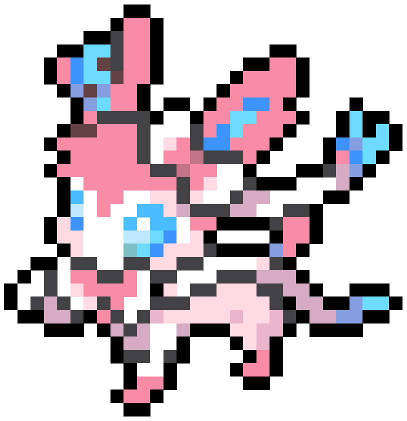
AMSC 460 Root Finding

Dennis Chunikhin

December 29, 2024



Contents

1 Nonlinear Optimization: 1D Root-Finding

We want to find the root of some arbitrary function f. That is, given $f : \mathbb{R} \to \mathbb{R}$, find x^* such that $f(x^*) = 0$.

This is equivalent to optimization since when arbitrary function $g(x^*)$ is at a minimum, its derivative $g'(x^*) = 0$.

1.1 Bisection Method

Algorithm 1.1: Bisection Method

Suppose we have continuous function $f : \mathbb{R} \to \mathbb{R}$.
Suppose we have two points x_l and x_r such that $f(x_l)$ and $f(x_r)$ have different signs.

Algorithm 1 Find the root of f using the bisection method.

Input: x_l, x_r
Output: $x_m^{(k)} \approx x^*$

```
 1: function ROOT($f, x_l, x_r$)
 2:     Let $x_l^{(k)} = k$-th left point                ▷ $x_l^{(0)} = x_l$
 3:     Let $x_r^{(k)} = k$-th right point               ▷ $x_r^{(0)} = x_r$
 4:     $x_m^{(k)} = \frac{x_l^{(k)} + x_r^{(k)}}{2}$    ▷ $k$-th midpoint

 5:     if $f(x_m^{(k)})$ has the opposite sign as $f(x_l^{(k)})$ then
 6:         $x_r^{(k+1)} = x_m^{(k)}$
 7:     else
 8:         $x_l^{(k+1)} = x_m^{(k)}$
 9:     end if

10:     Repeat these steps.
11: end function
```

1.1.1 Bisection Method Convergence

Since each step we cut the interval $\left|x_l^{(k)} - x_r^{(k)}\right|$ in half, after $k+1$ steps we have:

$$\left|x_l^{(k)} - x_r^{(k)}\right| \leq \frac{1}{2^{k+1}} |x_l - x_r| \tag{1}$$

Moreover, since the root x^* stays within the interval, the distance between the midpoint (our best guess of the root) and the root is smaller than the interval itself:

$$\left|x_m^{(k)} - x^*\right| \leq \left|x_l^{(k)} - x_r^{(k)}\right| \tag{2}$$

The right hand side is our error (distance between our guess of the root and the actual root). We want the error $\left|x_m^{(k)} - x^*\right|$ to be below some tolerance τ. This is satisfied if:

$$\begin{aligned}
&\left|x_m^{(k)} - x^*\right| \leq \left|x_l^{(k)} - x_r^{(k)}\right| \leq \frac{1}{2^{k+1}} |x_l - x_r| \leq \tau && (3)\\
\implies & \frac{1}{2^{k+1}} \leq \frac{\tau}{|x_l - x_r|} && (4)\\
\implies & 2^{k+1} \geq \frac{|x_l - x_r|}{\tau} && (5)\\
\implies & k+1 \geq \log\left(\frac{|x_l - x_r|}{\tau}\right) = \log\left(|x_l - x_r|\right) - \log(\tau) && (6)\\
\implies & k+1 \geq -\log(\tau) && (7)\\
\implies & k \geq -\log(\tau) && (8)
\end{aligned}$$

Here log denotes log base 2.

Note: step 7 holds if $\log\left(|x_l - x_r|\right)$ is positive or much smaller than $-\log(\tau)$. Since we expect the intial interval to be much larger than the tolerance, this is a fair assumption. Moreover, since the tolerance τ is a very small positive number, we expect $-\log(\tau) > 0$, and we can thus write $-\log(\tau) = |\log(\tau)|$.

This is a pretty good condition! E.g. if τ is of the order 10^{-6}, we would need about 20 steps.

Theorem 1.1: Bisection Method Step Count

The number of steps k needed to find the root of an arbitrary function f to within some tolerance τ using the bisection method is given by:

$$k \geq \left|\log(\tau)\right| \tag{9}$$

1.2 Newton's Method

We have described and derived Newton's method before, but it is worth re-deriving it using linearization via the Taylor series.

We approximate an arbitrary function f around point x_n using its Taylor expansion

$$\begin{aligned} f(x_n+\delta) &= f(x_n) + f'(x_n)\delta + \mathcal{O}(\delta^2) & (10)\\ &\approx f(x_n) + f'(x_n)\delta & (11)\end{aligned}$$

Note from me: this still requires the same continuity assumtion for f.

We then use this approximation to find the root:

$$\begin{aligned} f(x_n) + f'(x_n)\delta = 0 && (12)\\ \implies \delta = -\frac{f(x_n)}{f'(x_n)} && (13)\end{aligned}$$

A step of the Newton's method is then

$$\begin{aligned} x_{n+1} &= x_n + \delta & (14)\\ &= x_n - \frac{f(x_n)}{f'(x_n)} & (15)\end{aligned}$$

Algorithm 1.2: Newton's Method

A step of Newton's method is given by

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{16}$$

1.2.1 Newton's Method Convergence

As before, the error is defined as the distance between our guess of the root x_n and the true root x^*. We ask whether the error converges to 0, and if so how many steps does it take for it to fall below some threshold τ:

$$|x^* - x_n| \leq \tau \tag{17}$$

Observe that

$$f(x^*) - f(x_n) = \int_{x_n}^{x^*} f'(t)\mathrm{d}t \tag{18}$$

Since x^* is a root of f, $f(x^*) = 0$ by definition. Thus

$$-f(x_n) = \int_{x_n}^{x^*} f'(t)\mathrm{d}t \tag{19}$$

Additionally, consider the constant $f'(x_n)$. Since this is just some constant number,

$$(x^* - x_n)f'(x_n) = \int_{x_n}^{x^*} f'(x_n)\mathrm{d}t \tag{20}$$

Now we subtract equation 20 from equation 19:

$$-f(x_n) - (x^* - x_n)f'(x_n) = \int_{x_n}^{x^*} \left[f'(t) - f'(x_n)\right]\mathrm{d}t \tag{21}$$

Then divide both sides by $f'(x_n)$

$$-\frac{f(x_n)}{f'(x_n)} - (x^* - x_n) = \frac{1}{f'(x_n)}\int_{x_n}^{x^*} \left[f'(t) - f'(x_n)\right]\mathrm{d}t \tag{22}$$

Notice that $x_n - \frac{f(x_n)}{f'(x_n)} = x_{n+1}$ in our Newton's rule method. Thus the equation becomes

$$x_{n+1} - x^* = \frac{1}{f'(x_n)}\int_{x_n}^{x^*} \left[f'(t) - f'(x_n)\right]\mathrm{d}t \tag{23}$$

The right hand side is exactly the error! We can now introduce absolute value signs to get

$$|x_{n+1} - x^*| = \frac{1}{|f'(x_n)|} \left| \int_{x_n}^{x^*} [f'(t) - f'(x_n)] \, \mathrm{d}t \right| \tag{24}$$

$$\leq \frac{1}{|f'(x_n)|} \int_{x_n}^{x^*} |f'(t) - f'(x_n)| \, \mathrm{d}t \tag{25}$$

We must now make some assumptions.

Construction 1.1: Newton's Method Assumptions

Given f defined on interval I, assume:

(i) $|f'(t)| \geq \rho > 0 \quad \forall t \in I$

That is, $|f'(t)|$ is bounded below by ρ (this assumption allows division by $f'(x_n)$, which is part of Newton's method).

(ii) $|f'(t_1) - f'(t_2)| \leq \delta |t_1 - t_2| \quad \forall t_1, t_2 \in I$

Continuity requirement on f'.

Thus if $x_n, t \in I$, then equation 25 becomes

$$|x_{n+1} - x^*| \leq \frac{1}{|f'(x_n)|} \int_{x_n}^{x^*} |f'(t) - f'(x_n)| \, \mathrm{d}t \tag{26}$$

$$\leq \frac{1}{\rho} \int_{x_n}^{x^*} \delta \, |t - x_n| \, \mathrm{d}t \tag{27}$$

$$= \frac{\delta}{\rho} \int_{x_n}^{x^*} |t - x_n| \mathrm{d}t \tag{28}$$

$$= \frac{\delta}{\rho} \cdot \frac{1}{2} (x^* - x_n)^2 \tag{29}$$

Note that because of the square, the integral evaluates to $\frac{1}{2}(x^* - x_n)^2$ regardless of whether $x^* > x_n$ or $x^n > x^*$ (you can confirm this by evaluating the integral for each of these cases).

Denote the error at step n

$$e_n = |x_n - x^*| \tag{30}$$

Equation 29 tells us that

$$e_{n+1} \leq \frac{\delta}{2\rho} e_n^2 \tag{31}$$

Denote

$$\alpha = \frac{\delta}{2\rho} \tag{32}$$

For our purposes, α is just some constant.

Theorem 1.2: Newton's Method speed of convergence

The error in Newton's method follows

$$e_{n+1} \leq \alpha e_n^2 \tag{33}$$

This gives us the speed of convergence once we are close

Theorem 1.3: For a small enough initial interval, Newton's method converges.

We claim without proof that if the interval

$$|I| \leq \frac{1}{2\alpha} \tag{34}$$

then each $x_j \in I$ and

$$e_{n+1} \leq \frac{1}{2} e_n \tag{35}$$

This implies that

$$e_n \leq \frac{1}{2} e^{n-1} \leq \cdots \leq \frac{1}{2^n} e_0 \tag{36}$$

This guarantees that $e_n \to 0$ as $n \to \infty$.

Equation 33 gives us the speed of convergence once we are close enought to x^*:

Theorem 1.4: Newton's method error converges double exponentially once we are close enough

or simplicity let's assume that $\alpha = 1$. Then by equation 33

$$e_n \leq e_{n-1}^2 \leq \left(e_{n-2}^2\right)^2 \leq \cdots \leq e_0^{2^n} \quad \text{if } e_0 < 1 \tag{37}$$

In the homework we generalize this result for any α.

Next we find how many Newton's methods steps are needed to achieve some error tolerance τ–that is so that $e_n \leq \tau$:

$$\begin{align} e_n &\leq \tau \tag{38}\\ e_0^{2^n} &\leq \tau \tag{39}\\ \log e_0^{2^n} = 2^n \log e_0 &\leq \log \tau \tag{40}\end{align}$$

Since $e_0 < 1$ (from the interval of convergence) and presumably $\tau < 1$, $\log e_0$ and $\log \tau$ are negative, meaning we can rewrite equation 40 as

$$\begin{align} 2^n |\log e_0| &\geq |\log \tau| \tag{41}\\ 2^n &\geq \frac{|\log \tau|}{|\log e_0|} \tag{42}\\ n &\geq \log \frac{|\log \tau|}{|\log e_0|} \tag{43}\\ &= \log|\log \tau| - \log|\log e_0| \tag{44}\end{align}$$

where log is the logarithm base 2.

Again from the interval condition (theorem 1.3), we will certainly have $e_0 \leq 1/2$. This means that $\log e_0 \leq -1$, meaning that $|\log e_0| \geq 1$. Thus, $\log|e_0| \geq 0$, which means that equation 44 becomes

$$\begin{aligned} n &\geq \log|\log \tau| - \log|\log e_0| \\ &\geq \log|\log \tau| \end{aligned}$$

Theorem 1.5: Newton's method: number of steps needed to achieve a tolerance grows as $\log\log$

If we are close enough to x^* (and the assumptions above still hold, e.g. $\alpha = 1$), the number of Newton's method steps n needed to achieve tolerance τ is bounded by

$$n \geq \log|\log \tau| \tag{45}$$

As before, in the homework we generalize this resultz for any α.

Note that this is a much faster convergence than the bisection method (and is in general a very good convergence)!

1.3 Rate of Convergence

This section builds some mechanisms that we use to compare the convergence of algorithms.

Definition 1.1: Rates of Convergence

Let $\gamma < 1$ be some constant.

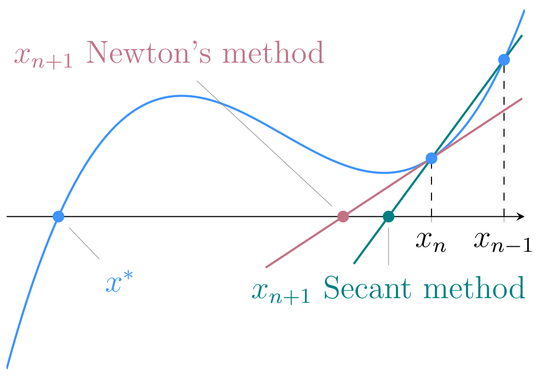
If $e_{n+1} \leq \gamma e_n$, we say the algorithm displays **linear convergence**.
If $e_{n+1} \leq \gamma e_n^2$, we say the algorithm displays **quadratic convergence**.
If $e_{n+1} \leq \gamma e_n^p$ for $1 < p < 2$, we say the algorithm displays **superlinear convergence**. We call p the **rate of convergence**.

Bisection method has linear convergence.

Newton's method has quadratic convergence.

As we will see in the next section, the secant method has superlinear convergence. Moreover, the rate of convergence for the secant method is $p = \frac{1+\sqrt{5}}{2}$, the golden ratio!

1.4 Secant Method



The idea behind the Secant method is to approximate the derivative of the function f using the slope between our last and current guess as illustrated above. The advantage of this is lower cost–that is, only 1 function evaluation is needed each step.

The equation of the secant line is:

$$\frac{y - f(x_n)}{x - x_n} = \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}} \tag{46}$$

We want to solve for $x = x_{n+1}$ such that $y = 0$, so we plug these values into equation 46 and solve for x_{n+1}:

$$\frac{0 - f(x_n)}{x_{n+1} - x_n} = \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}} \tag{47}$$

$$\implies x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} \tag{48}$$

Algorithm 1.3: Secant Method

A step of the secant method is given by

$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} f(x_n) \tag{49}$$

The advantage of this over Newton's method is that only one function evaluation is needed per step (that is, you do not need to evaluate $f'(x_n)$).

Theorem 1.6: Secant method rate of convergence is superlinear

The error in the secant method follows

$$e_{n+1} \leq \alpha e_n^p \tag{50}$$

where $p = \frac{1+\sqrt{5}}{2}$ is the golden ratio.

The professor states this without proof in the notes. The error analysis is a bit more involved but not inaccessible; I found a good writeup here: `https://www.math.drexel.edu/~tolya/300_secant.pdf`.

1.4.1 Cost

We find the number of steps n needed to achieve an error below a given tolerance: $e_n \leq \tau$. To simplify analysis, let's assume $\alpha = 1$. Then

$$e_n \leq e_{n-1}^p \leq (e_{n-2}^p)^p \leq \cdots \leq e_0^{p^n} \leq \tau \tag{51}$$

We take the logarithm of both sides:

$$\log e_0^{p^n} \leq \log \tau \tag{52}$$

$$p^n \log e_0 \leq \log \tau \tag{53}$$

Assuming $e_0 < 1, \tau < 1$, both of the logarithm are negative, meaning that we can rewrite the expression as

$$-p^n \log e_0 \geq -\log \tau \tag{54}$$
$$p^n |\log e_0| \geq |\log \tau| \tag{55}$$
$$p^n \geq \frac{|\log \tau|}{|\log e_0|} \tag{56}$$

Take the logarithm again:

$$n \log p \geq \log \frac{|\log \tau|}{|\log e_0|} \tag{57}$$
$$= \log|\log \tau| - \log|\log e_0| \tag{58}$$
$$\geq \log|\log \tau| \tag{59}$$

For comparison with Newton's method, log still means the logarithm base 2.

Theorem 1.7: Secant method is cheaper than Newton's method

To achieve an error below tolerance τ (with the assumptions from above) the number of steps n the secant method requires is bounded by

$$n \geq \frac{\log|\log \tau|}{\log p} \approx \frac{\log|\log \tau|}{0.696} \approx 1.44 \log|\log \tau| \tag{60}$$

where log is the logarithm base 2.
Thus Newton's method requires $2n_{\text{newt}}$ function evaluations, while the secant method requires only about $1.44 n_{\text{newt}}$. Thus the secant method is cheaper.

2 Nonlinear Optimization: Roots of Multi-Valued Functions of Multiple Variables

Suppose we have a function $f : \mathbb{R}^n \to \mathbb{R}^n$. We are seeking x^* such that

$$f(x^*) = \mathbf{0} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix} \tag{61}$$

2.1 Multivariate Newton's Method

We derive Newton's method again using Taylor series.

We can describe the multivariate function as a system of equations:

$$f(x) = \begin{pmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_n(x) \end{pmatrix} \tag{62}$$

where each $f_i : \mathbb{R}^n \to \mathbb{R}$.

Consider the Taylor series for a given f_i:

$$f_i(x_k + \delta) = f_i(x_k) + (\nabla f_i)(x_k)^T \delta + O(\|\delta\|^2) \tag{63}$$

Thus, for the entire function

$$f(x_k + \delta) = f(x_k) + \begin{pmatrix} \nabla f_1(x_k)^T \\ \nabla f_2(x_k)^T \\ \vdots \\ \nabla f_n(x_k)^T \end{pmatrix} \delta + O(\|\delta\|^2) \tag{64}$$

The $n \times n$ matrix formed by the gradients is called the Jacobian matrix.

Definition 2.1: Jacobian matrix

Given $f : \mathbb{R}^n \to \mathbb{R}^n$, the Jacobian matrix is

$$J_f(x) = \begin{pmatrix} \nabla f_1(x)^T \\ \nabla f_2(x)^T \\ \vdots \\ \nabla f_n(x)^T \end{pmatrix} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix} \tag{65}$$

all evaluated at x.
Think of the Jacobian as our multivariate "derivative".

Newton's method approximates the root using the linear Taylor series approximation obtained from equation 64:

$$f(x_k) + J_f(x_k)\delta = 0 \tag{66}$$

Hence to get δ we need to solve the equation

$$J_f(x_k)\delta = -f(x_k) \tag{67}$$

Algorithm 2.1: Multivariate Newton's method

Given function $f : \mathbb{R}^n \to \mathbb{R}^n$, to get the next estimate of the root using the multivariate Newton's method we need to solve the system of equations

$$J_f(x_n)\delta = -f(x_n) \tag{68}$$

Then

$$x_{n+1} = x_n + \delta \tag{69}$$

This statement makes the most sense computationally, but we can also write it more concisely mathematically as

$$x_{n+1} = x_k - J_f(x_k)^{-1} f(x_k) \tag{70}$$

2.1.1 Cost and Convergence

Note that computing the Jacobian matrix $J(x_k)$ can possibly be expensive in and of itself.

Additionally, the cost of solving the system is approximately $\frac{1}{3}n^3$ for Gaussian elimination.

We also state the conditions for the convergence of the multivariate Newton's method without proof

Construction 2.1: Multivariate Newton's method converges if the Jacobian is continuous bounded from below.

Given $f : \mathbb{R}^n \to \mathbb{R}^n$ defined in a ball B, Newton's method is guaranteed to converge if:

(i) $\|J_f(x) - J_f(y)\| \leq \gamma \|x - y\| \quad \forall x, y \in B$

This is the continuity of J_f, and is analogous to the continuity of f' condition for 1D Newton's method.

(ii) $\|J_f(x)\| \geq \rho \quad \forall x \in B$

This is analogous to the $|f'(t)| \geq \rho$ condition in 1D.

Theorem 2.1: Multivariate Newton's method converges quadratically.

If the conditions above hold, the Newton's method is convergent and

$$e_{k+1} \leq \alpha e_k^2 \quad \text{for some } \alpha > 0 \tag{71}$$

where the error is defined as $e_k = \|x^* - x_k\|$.

In other words, if x_k is close enough to x^*, Newton's method converges fast.

The question that remains is how to get "close enough" to x^* so that Newton's method will converge. This is complicated by the fact that in the multivariate case we do not have an analogue of the bisection method.

2.2 Fixed Point Iteration

One way to get "close enough" is through fixed point iteration, which we will develop in this section.

Let

$$D = \begin{pmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_n \end{pmatrix} \tag{72}$$

be a diagonal matrix.

We can write

$$f(x) = Dx - (Dx - f(x)) \tag{73}$$

Note that $f(x) = 0 \iff Dx - (Dx - f(x)) = 0$.

Left-multiplying $Dx - (Dx - f(x)) = 0$ by D^{-1}, we get

$$f(x) = Dx - (Dx - f(x)) \tag{74}$$

$$D^{-1}f(x) = x - (x - D^{-1}f(x)) \tag{75}$$

Construction 2.2: $G(x)$

Define

$$G(x) = x - D^{-1}f(x) \tag{76}$$

where D is some diagonal matrix.

Thus equation 75 becomes

$$D^{-1}f(x) = x - G(x) \tag{77}$$

From this expression we then get

$$f(x) = 0 \iff G(x) = x \tag{78}$$

Definition 2.2: Fixed Point

Finding a root x^* of f is equivalent to finding x^* such that

$$G(x^*) = x^* \tag{79}$$

Such a point x^* is called a **fixed point** of G.

Finding a fixed point the involves simply iterating with G

Algorithm 2.2: Find a fixed point of G

To find a fixed point of G
(i) Start with some x_0.
(ii) Iterate $x_{k+1} = G(x_k)$.

2.2.1 Error

Consider the error in fixed point iteration

$$x^* - x_{k+1} = x^* - G(x_k) = G(x^*) - G(x_k) \tag{80}$$

We make two assertions withouth proof

Theorem 2.2: Claim 1

We claim that

$$\exists\ \xi_k \text{ s.t.} \quad x^* - x^{k+1} = J_G(\xi_k)(x^* - x_k) \tag{81}$$

This is kind of like the multivariate version of the intermediate value theorem.

Theorem 2.3: Claim 2

We can choose D such that

$$\|J_G(\xi_k)\| < 1 \tag{82}$$

The intuition is that this is similar to $G = 1 - \frac{1}{d}f' < 1$ if $f' > 0$ and d is large.

This shows us that fixed point iteration converges

Theorem 2.4: Fixed point iteration converges

Given D as in claim 2,

$$\|x^* - x_{k+1}\| < \|x^* - x_k\| \tag{83}$$

Thus the error $\|x^* - x_k\| \to 0$ as $k \to \infty$.

So we can use fixed point iteration to make x_k close to x^*. Then we can switch to Newton's method, which converges quickly!

2.3 Variations on Newton's Method

The downside of Newton's method is that evaluating J_f and solving the system for δ_k can be expensive.

One alternative is to re-use J_f

Algorithm 2.3: Modified Newton's method: re-use J_f

Given x_k, compute $J_f(x_k)$.
Compute $J_f = L_k U_k$ factors.
Then for some set of $k+1, k+2, \dots, k+l$ compute

$$x_{k+j} = x_{k+j-1} + \delta_{k+j-1} \tag{84}$$

where δ_{k+j-1} is the solution to

$$J_f(x_k)\delta_{k+j-1} = f(x_{k+j-1}) \tag{85}$$

That is, for l steps don't update the Jacobian factors.
The cost is reduces from $\frac{1}{3}n^3$ to $O(n^2)$.
The downside is that this might be slower.

Solving the equation with the Jacobian matrix can be viewed as going a distance δ in the direction of steepest descent (the search direction). However this can easily cause the algorithm to overshoot the root, especially at the early steps. That is, it is not guaranteed that $\|f(x_{k+1})\| < \|f(x_k)\|$.

A possible remedy is to use a step size γ:

Algorithm 2.4: Newton's method with step size

Using a step size γ, update as

$$x_{k+1} = x_k + \gamma\delta_k \tag{86}$$

where γ is chosen to ensure $\|f(x_{k+1})\| < \|f(x_k)\|$.
There are different ways to select such a γ. One way is to minimize the following function with respect to γ

$$\phi(\gamma) = \|f(x_k + \gamma\delta_k)\| \tag{87}$$

3 Ordinary Differential Equations

We want to solve an arbitrary first-order ordinary differential equation (initial value problem). The problem can be stated generally as follows: we want to find the function $y(t)$ that satisfies an equation of the form

$$y' = f(t, y) \tag{88}$$

where f is some given function and we are given the initial value $y(t_0)$.

For example, if the problem is $y' = \lambda y$ and $y(0) = C$, then the is $y(t) = Ce^{\lambda t}$.

Note that y and f can both be vectors. A simple example of a multivariable linear first order ODE is $y' = Ay,\ A \in \mathbb{R}^{n \times n}$.

3.1 Forward Euler's Method

We are given problem $y' = f(t, y)$.

Observe that by the fundamental theorem of calculus,

$$y(t+h) - y(t) = \int_t^{t+h} y'(\tau)\mathrm{d}\tau \tag{89}$$

Thus, to solve the differential equation we can approximate this integral by a quadrature rule.

The rectangle rule gives us

$$\int_t^{t+h} y'(\tau)\mathrm{d}\tau \approx y'(t)h \tag{90}$$

$$= f(t, y(t))h \tag{91}$$

Given t_0, let $t_1 = t_0 + h$.

This approximation gives us

$$y(t_1) \approx y(t_0) + hf(t_0, y_0) \tag{92}$$

So call $y(t_0) + hf(t_0, y_0) = y_1$ our best guess of $y(t_1)$.

Algorithm 3.1: Forward Euler's method for ODEs

Given initial value problem

$$y' = f(t, y),\ \ t \in [t_0, T] \tag{93}$$

$$y(t_0) = y_0 \tag{94}$$

we repeatedly estimate

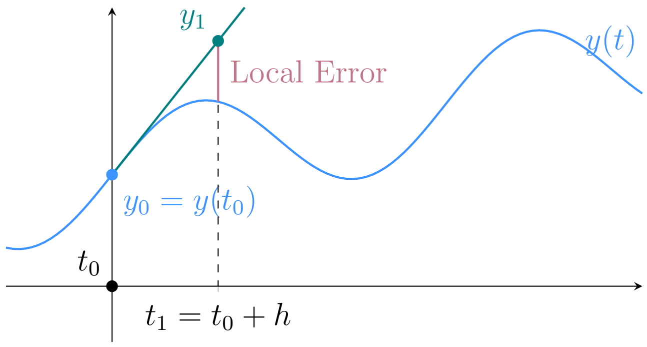
$$y_{n+1} = y_n + hf(t_n, y_n) \tag{95}$$

where

$$h = \frac{T - t_0}{n} \tag{96}$$

until we get to $t_{\text{final}} = t_n = T$.

Graphically, we are using the line tangent to $y(t)$ at t_0 as an approximation of $y(t)$, and going forward along that line to $t_1 = t_0 + h$.

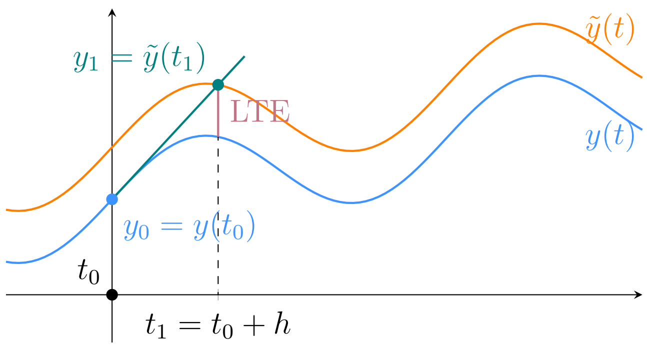


The red line (difference between the estimate y_1 and the true value $y(t_1)$) is the local error.

Theorem 3.1: The estimate y_1 falls on another solution to the ODE

We claim that there is another solution to $y' = f(t, y)$, call it $\tilde{y}(t)$, that satisfies

$$\tilde{y}(t_1) = y_1 \tag{97}$$



At every step we are approximating the solution curve, but incurring a local error (caused by quadrature error). This local error is called the **local truncation error (LTE)**. It is shown by the red line on the graph.

This error will come up in a bit.

3.2 Backward Euler's Method

Now let us consider the other rectange rule (using the right point as the estimate):

$$\int_{t_0}^{t_0+h} y'(t)\mathrm{d}t \approx hy'(t_0+h) \tag{98}$$
$$= hy'(t_1) \tag{99}$$
$$= hf(t_1, y(t_1)) \tag{100}$$
$$= hf(t_1, y_1) \tag{101}$$

Now y_1 needs to be determined. That is, we need to solve

$$y_1 = y_0 + hf(t_1, y_1) \tag{102}$$

Algorithm 3.2: Backward Euler's method for ODEs

We make the next approximation by solving

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1}) \tag{103}$$

for y_{n+1}.

Note that this is (possibly) a nonlinear equation for unkown y_{n+1}.

3.2.1 Example

We illustrate an advantage of backward Euler's method over forward Euler's method by an example

Consider the problem

$$y' = \lambda y \tag{104}$$

Suppose that λ is real and $\lambda < 0$.

The solution is $y(t) = Ce^{\lambda t}$, which means that

$$y' = C\lambda e^{\lambda t} = \lambda y \tag{105}$$

We want our method to "respect the solution." That is, if $y(t)$ is decreasing as t grows, we want $y_1, y_2, y_3, \ldots$ to also decrease.

Consider the forward Euler method:

$$y_1 = y_0 + h\lambda y_0 = (1 + h\lambda)y_0 \tag{106}$$
$$y_2 = y_1 + h\lambda y_1 = (1 + h\lambda)y_1 = (1 + h\lambda)^2 y_0 \tag{107}$$
$$\vdots \tag{108}$$

If we want these to decrease (as y does), we must have

$$|1 + h\lambda| < 1 \tag{109}$$

This gives us two conditions:

$$1 + h\lambda < 1 \tag{110}$$
$$\implies h\lambda < 0 \tag{111}$$

which is already satisfied, and

$$-1 < 1 + h\lambda \tag{112}$$
$$\implies -h\lambda < 2 \tag{113}$$
$$\implies h|\lambda| < 2 \tag{114}$$
$$\implies h < \frac{2}{|\lambda|} \tag{115}$$

This is a severe restriction on h, especially for larger $|\lambda|$.

Now consider the backward Euler method:

$$\begin{aligned} y_{k+1} &= y_k + h\lambda y_{k+1} && (116)\\ \implies (1-h\lambda)y_{k+1} &= y_k && (117)\\ \implies (1+h|\lambda|)y_{k+1} &= y_k && (118)\\ \implies y_{k+1} &= \frac{1}{1+h|\lambda|}y_k && (119) \end{aligned}$$

As before, we want the sequence of y_k's to decrease, for which we must have $\frac{1}{1+h|\lambda|} < 1$.

But this is always true!

3.3 n-dimensional Euler's method

We can also generalize this idea to n-dimensional space.

Consider $y \in \mathbb{R}^n$, f an n-dimensional function, and the problem

$$y' = f(t,y) \tag{120}$$

As with the 1D case, the Euler methods will be

Forward Euler:

$$y_{k+1} = y_k + hf(t,y_k) \tag{121}$$

Backward Euler:

$$y_{k+1} = y + hf(t,y_{k+1}) \tag{122}$$

3.3.1 Example

Consider the n-dimensional problem

$$y' = Ay \tag{123}$$

The forward Euler method step is:

$$y_{k+1} = y_k + hAy_k = (I+hA)y_k \tag{124}$$

The backward Euler method step is:

$$y_{k+1} = y_k + hAy_{k+1} \tag{125}$$
$$\implies (I - hA)y_{k+1} = y_k \tag{126}$$

This is a linear system of equations that we need to solve.

Suppose A is symmetric. Then A must have n orthonormal eigenvectors $v_1, \ldots, v_n$ such that $Av_i = \lambda_i v_i,\ \forall i$ and

$$v_i^T v_j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{127}$$

Thus, we can write

$$AV = V\Lambda \tag{128}$$

where

$$V = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix} \tag{129}$$

and

$$\Lambda = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{bmatrix} \tag{130}$$

Note that since the v_i are orthogonal, V is orthonormal–that is $V^T V = VV^T = I$.

We can thus right-multiply equation 128 by V^T to get

$$A = V\Lambda V^T \tag{131}$$

The problem $y' = Ay$ then becomes

$$y' = V\Lambda V^T y \tag{132}$$

Let's define a new variable $z = V^T y$. The problem then becomes

$$y' = V\Lambda z \tag{133}$$

We can further simplify this by left-multipling by V^T.

$$\begin{aligned} V^T y' &= \Lambda z && (134)\\ \implies (V^T y)' &= \Lambda z && (135)\\ \implies z' &= \Lambda z && (136) \end{aligned}$$

Since Λ is diagonal this gives us n independent ordinary differential equations for $z_i(t)$ of the form discussed in the previous example. If, as in the previous example, all $\lambda_i < 0$, we have the same restriction as in the previous example but with all λ_i:

$$h < \frac{2}{|\lambda_i|},\ \forall i \tag{137}$$

3.4 Euler's Method Accuracy

Consider the Taylor series expansion for y for the forward Euler method (particularly, this is the Lagrange remainder theorem):

$$\begin{aligned} y(t_1) &= y(t_0 + h) && (138)\\ &= y(t_0) + hy'(t_0) + \frac{h^2}{2}y''(\eta_0) && (139)\\ &= y_1 + \frac{h^2}{2}y''(\eta_0) && (140) \end{aligned}$$

for some $\eta_0 \in (t_0, t_1)$.

So the error (difference between our guess y_1 and the true value $y(t_1)$) is

$$y(t_1) - y_1 = \frac{h^2}{2}y''(\eta_0) \tag{141}$$

More generally,

$$y(t_{n+1}) = y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(\eta_n) \tag{142}$$

for some $\eta_n \in (t_n, t_{n+1})$.

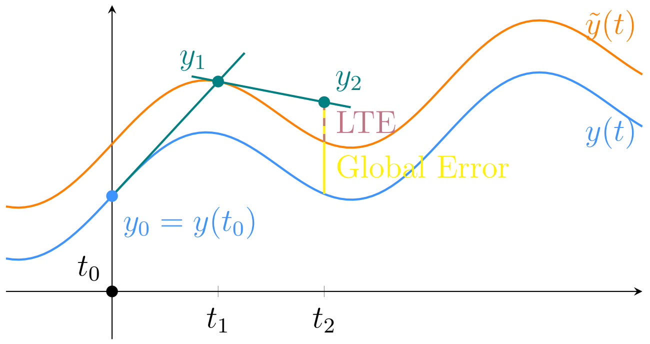
Theorem 3.2: Local Truncation Error for the Forward Euler method is $O(h^2)$.

The Local Truncation Error (LTE) for the forward Euler method is

$$\frac{h^2}{2}y''(\eta_n) \tag{143}$$

for some $\eta_n \in (t_n, t_{n+1})$.

Previously we claimed (theorem 3.1) that there exists another solution trajectory which our guess falls on. The LTE is just the local error–the amount that our guess deviates from the solution trajectory we were last on. These errors add up to a larger global error (as illustrated in the diagram below by the yellow line).



That is, the global error = LTE + error of $\tilde{y}$.

Theorem 3.3: Global Error is bounded by the sum of Local Errors

If $f_y = \frac{\partial f}{\partial y} \leq 0, \quad \forall t \in [t, T]$ and none of the solution curves $(t, y(t))$ intersect, then the global error $g_n = y(t_n) - y_n$ satisfies

$$|g_n| \leq \sum_{k=1}^{n} |\text{LTE}(h)| \tag{144}$$

By theorem 3.2, for the forward Euler method

$$\text{LTE}(h) = y''(\eta)\frac{h^2}{2} \tag{145}$$

for $h = \frac{T-t_0}{N}$.

Then by theorem 3.3

$$\begin{aligned}
|y(t_n) - y_n| &\leq \left|\sum_{k=1}^{N} \frac{h^2}{2} y''(\eta_k)\right| && (146)\\
&\leq \max_{\eta\in(t_0,T)} |y''(\eta)| \sum_{k=1}^{N} \frac{h^2}{2} && (147)\\
&= \max_{\eta\in(t_0,T)} |y''(\eta)| N \frac{h^2}{2} && (148)\\
&= \max_{\eta\in(t_0,T)} |y''(\eta)| \frac{T-t_0}{h} \cdot \frac{h^2}{2} && (149)\\
&= \max_{\eta\in(t_0,T)} |y''(\eta) \frac{T-t_0}{2} h && (150)
\end{aligned}$$

That is, the global error is $O(h)$.

Theorem 3.4: The Global error for Euler's method is $O(h)$

The bound for the global error for both the forward and backward Euler's method is

$$O(h) \tag{151}$$

Notice that the global error loses a power of h (relative to the LTE).

We proved this for the forward Euler's method. The proof for the backward Euler's method should follow a similar structure.

This bound tells us the accuracy, as opposed to the bound for h that we found in the example section, which is a stability requirement. I will restate this bound here to highlight the difference:

Theorem 3.5: Stability Requirement

The bound

$$h \leq \frac{2}{|\lambda|} \tag{152}$$

which we found in the example section is a **stability requirement**.

3.5 Heun's Method

$O(h)$ is a pretty bad accuracy bound, so we want to develop a more accurate method.

We approach this by consider the Taylor series, but this time with more terms (up to the second order term):

$$y(t_{n+1}) = y(t_n) + y'(t_n)h + \frac{y''(t_n)}{2}h^2 + O(h^3) \tag{153}$$

As before, the problem we want to solve is

$$y'(t) = f(t, y(t)) \tag{154}$$

Since we now have y'' in the Taylor expansion, let's find the second derivative of y by differentiating this expression

$$\begin{aligned} y''(t) &= \frac{\mathrm{d}}{\mathrm{d}t} f(t_n, y(t_n)) &(155)\\ &= \frac{\partial f}{\partial t} + \frac{\partial f}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}t} &(156)\\ &= f_t + f_y y' &(157)\\ &= f_t + f_y f &(158) \end{aligned}$$

Now we plug $y' = f$ and $y'' = f_t + f_y f$ into the taylor series expansion 153

$$\begin{aligned} y(t_{n+1}) = y(t_n) &+ hf(t_n, y(t_n)) \\ &+ \frac{h^2}{2}\Big(f_t(t_n, y(t_n)) + f_y(t_n, y(t_n))f(t_n, y(t_n))\Big) + O(h^3) \end{aligned} \tag{159}$$

We want a numerical method such that $|y_{n+1} - y(t_{n+1})| \propto h^3$.

We can approach this by trying something of the form

$$y_{n+1} = y_n + ak_1 + bk_2 \tag{160}$$

Try

$$k_1 = hf(t_n, y_n) \tag{161}$$

like with Euler's method, and

$$k_2 = hf(t_n + \alpha h, y_n + \beta k_1) \tag{162}$$

Note from me: qualitatively, what we are doing is asserting that we will try using information from two samples of f rather than one, and are saying that the second sample will be taken at some arbitrary offset parametrized by α and β.

Now we need to find the constants a, b, α, β to make this work–that is to make y_{n+1} equal the first three terms of the taylor expansion.

Let us start by first writing the Taylor expansion of f:

$$\begin{align} f((t_n, y_n) + (\delta t, \delta y)) &= f(t_n, y_n) + \nabla f(t_n, y_n) \cdot \begin{pmatrix} \delta t \\ \delta y \end{pmatrix} + O\left(\left\| \begin{matrix} \delta t \\ \delta y \end{matrix} \right\|^2 \right) \tag{163} \\ &= f(t_n, y_n) + f_t(t_n, y_n)\delta t + f_y(t, y_n) f(t, y_n)\delta y + O\left(\left\| \begin{matrix} \delta t \\ \delta y \end{matrix} \right\|^2 \right) \tag{164} \end{align}$$

Plugging this into our expression for k_2, we get

$$\begin{align} k_2 &= hf(t_n + \alpha h, y_n + \beta k_1) \tag{165} \\ &= hf(t_n, y_n) + \alpha h^2 f_t(t_n, y_n) + \beta h^2 f_y(t_n, y_n) f(t_n, y_n) + O(h^3) \tag{166} \end{align}$$

The resulting method (equation 160) is

$$y_{n+1} = y_n + ahf + bhf + \alpha b h^2 f_t + \beta b h^2 f_y f \tag{167}$$

where all f's are evaluated at (t_n, y_n).

To match this with the Taylor series 159, we must have

$$a + b = 1 \tag{168}$$

from the hf terms,

$$\alpha b = \frac{1}{2} \tag{169}$$

from the $h^2 f_t$ term, and

$$\beta b = \frac{1}{2} \tag{170}$$

from the $h^2 f_y f$ term.

One possible solution for this is

$$a = b = \frac{1}{2} \tag{171}$$
$$\alpha = \beta = 1 \tag{172}$$

The resulting method is (simply plugging these values in):

Algorithm 3.3: Heun's method (2nd order Runge-Kutta method)

For Heun's method, we have

$$k_1 = hf(t_n, y_n) \tag{173}$$
$$k_2 = hf(t_n + h, y_n + k_1) \tag{174}$$

A step in Heun's method is given by

$$\begin{aligned} y_{n+1} &= y_n + \frac{1}{2}k_1 + \frac{1}{2}k_2 && (175)\\ &= y_n + \frac{1}{2}hf(t_n, y_n) + \frac{1}{2}hf(t_n + h, y_n + k_1) && (176) \end{aligned}$$

Heun's method is a **2nd order Runge-Kutta method**.

Theorem 3.6: Heun's method LTE is $O(h^3)$ and global error is $O(h^2)$

By our derivation of Heun's method, the LTE is

$$|y(t_{n+1} - y_{n+1})| \propto h^3 \tag{177}$$

Since the global error loses a power of h, the global error is

$$|y(t_N)| \leq ch^2 \tag{178}$$

for some constant c. t_N is the final t.

3.5.1 Relation to Quadrature

Note that the $\frac{1}{2}hf(t_n, y_n) + \frac{1}{2}hf(t_n + h, y_n + k_1)$ is the trapezoid rule for estimating the integral

$$\begin{aligned} y(t_{n+1}) - y(t_n) &= \int_{t_n}^{t_{n+1}} y'(t)\mathrm{d}t && (179) \\ &\approx \frac{1}{2}hy'(t_n) + \frac{1}{2}hy'(t_{n+1}) && (180) \\ &= \frac{1}{2}hf(t_n, y_n) + \frac{1}{2}hf(t_{n+1}, y_{n+1}) && (181) \\ & && (182) \end{aligned}$$

We are estimating the $f(t_{n+1}, y_{n+1})$ in the last term as $f(t_n + h, y_n + k_1)$.

Algorithm 3.4: Implicit Heun's method

An alternative method is to find y_{n+1} by solving

$$y_{n+1} - \frac{h}{2}f(t_{n+1}, y_{n+1}) = y_n \tag{183}$$

The benefit of this method is that, as we have seen before, there is no restriction on h.
The con is that we have to solve the equation.

3.6 Runge-Kutta Methods

We can follow through with the same analysis but with even more higher order terms in the Taylor expansion. For instance, through similar analysis we can derive the following method:

Algorithm 3.5: RK4

Given y_n, Compute

$k_1 = hf(t_n, y_n)$
$k_2 = hf(t_n + \frac{h}{2}, y_n + \frac{k_1}{2})$
$k_3 = hf(t_n + \frac{h}{2}, y_n + \frac{k_2}{2})$
$k_4 = hf(t_n + h, y_n + k_3)$

Then compute

$$y_{n+1} = y_n + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4 \tag{184}$$

This is derived so that the LTE is $O(h^5)$.
This method, called **RK4**, is a 4th order Runge-Kutta method (the global error is proportional to h^4).

Note that the coefficients in the expression for y_{n+1} come from Simpson's rule.

Definition 3.1: Runge-Kutta methods

These types of methods are called **Runge-Kutta methods**. The order of the method is the degree of h in the global error (equivalently–it is the highest degree Taylor series term that we use in the derivation).
For example

- Euler's method is a **1st order Runge-Kutta method**.
- Heun's method is a **2nd order Runge-Kutta method**.
- RK4 is a **4th order Runge-Kutta method**.

Definition 3.2: Stages

The number of function evaluations at each step is called the number of **stages**.
For example,

- Heun's method is a **2-stage** method.
- RK4 is a **4-stage** method.

There exists a 6-stage Runge-Kutta method that is $5th$ order in accuracy ($O(h^5)$). Beyond that, you mostly have diminishing returns.

Notably, the $RKF45$ algorithm uses $RK5$ to estimate error for $RK4$ (in a similar way that we estimate quadrature error–using two methods of different order).

3.7 Multi-Step Methods

To reacap, the various method for solving the differential equation $y' = f(t, y)$ work by approximating the integral in

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f(t, y)\mathrm{d}t \tag{185}$$

The methods we have covered so far (Runge-Kutta methods) are all **single step methods**. That is, the get y_{n+1} using only t_n.

Multi-step methods use $t_n, y_n, t_{n-1}, y_{n-1}$ and all the other previously computed t_i, y_i to approximate the integral.

Algorithm 3.6: Multi-Step Methods

Suppose we use k points.
The multistep method works by interpolating $f(t, y)$ using the k previously computed points $(t_n, y_n), (t_{n-1}, y_{n-1}), \ldots, (t_{n-k+1}, y_{n-k+1})$, and then integrating the interpolating polynomial to give us the following method:

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} p(t)\mathrm{d}t \tag{186}$$

Apart from the first step, only (t_n, y_n) needs to be calculated each step.

3.7.1 Example

Consider $k = 2$.

We interpolate $f(t, y)$ using the two points $(t_n, f(t_n, y_n)), (t_{n-1}, f(t_{n-1}, y_{n-1}))$, which we will denote $(t_n, f_n), (t_{n-1}, f_{n-1})$ for conciseness.

$$p(t) = f_{n-1} + \frac{f_n - f_{n-1}}{t_n - t_{n-1}}(t - t_{n-1}) = f_{n-1} + \frac{f_n - f_{n-1}}{h}(t - t_{n-1}) \tag{187}$$

We can check that

$$\int_{t_n}^{t_{n+1}} p(t)\mathrm{d}t = \frac{1}{2}h(3f_n - f_{n-1}) \tag{188}$$

This gives us the following method

$$y_{n+1} = y_n + \frac{1}{2}h3f_n - \frac{1}{2}hf_{n-1} \tag{189}$$

We claim that the local truncation error is $O(h^3)$. This is the same as Heun's method, but with the advantage that here we only need 1 function evaluation per step.

Note that we need to use Heun's method to compute the very first step y_1.

Algorithm 3.7: Some Multi-Step Methods

Here are some multistep methods using k previous points:

- For $k = 1$, $\quad y_{n+1} = y_n + hf_n$
  Note that this is the forward Euler method.
- For $k = 2$, $\quad y_{n+1} = y_n + \frac{1}{2}h3f_n - \frac{1}{2}hf_{n-1}$
- For $k = 3$, $\quad y_{n+1} = y_n + \frac{h}{12}(23f_n - 16f_{n-1} + 5f_{n-2})$
- For $k = 4$, $\quad y_{n+1} = y_n + \frac{h}{24}(55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3})$

Definition 3.3: Adams Bashford Methods

The above multi-step methods are called **Adams Bashford** methods. The number of points used for interpolation, k, is the order of the method.

- $k = 1$ gives the **Forward Euler** method.
- $k = 2$ gives the **2nd order Adams Bashford** method.
- $k = 3$ gives the **3rd order Adams Bashford** method.
- $k = 4$ gives the **4th order Adams Bashford** method.

The advantage of Adams Bashford methods over RK methods is that you only need 1 extra function evaluation per step.

We can also get implicit multi-step methods:

Algorithm 3.8: Adams-Moulton Methods

We can get implicit multi-step methods by letting the k interpolating points be

$$(t_{n+1}, f_{n+1}), (t_n, f_n), \dots, (t_{n-k+1}, f_{n-k+1}) \tag{190}$$

These methods are called **Adams-Moulton methods**.

Algorithm 3.9: Some Adams-Moulton Methods

Some examples of Adams-Moulton methods are:

- For $k = 1$, $\quad y_{n+1} = y_n + hf(t_{n+1}, y_{n+1})$
  Note that this is the backward Euler method.
- For $k = 2$, $\quad y_{n+1} = y_n + \frac{h}{2}(f_{n+1} + f_n)$
  LTE $O(h^3)$
- For $k = 4$, $\quad y_{n+1} = y_n + \frac{h}{24}(9f_{n+1} + 19f_n - 5f_{n-1} + f_{n-2})$
  LTE $O(h^5)$

3.7.2 Stability

For any ODE we generate the points

$$(t_1, y_1), \dots, (t_i, y_i), \dots \tag{191}$$

Let's examine the error

$$e_n = y(t_n) - y_n \tag{192}$$

We will consider the forward Euler method.

The Taylor series expansion giving us the forward Euler method is:

$$\begin{aligned} y(t_{n+1}) &= y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(\xi_n) && (193) \\ &= y(t_n) + hf(t_n, y(t_n)) + \frac{h^2}{2}y''(\xi_n) && (194) \end{aligned}$$

Note from me: this is the Lagrange remainder theorem.

Plugging in this expansion for $y(t_{n+1})$ and the method step for y_{n+1}, the error becomes

$$\begin{aligned} y(t_{n+1}) - y_{n+1} &= y(t_n) - y_n + h(f(t_n, y(t_n)) - f(t_n, y_{n+1})) + O(h^2) && (195) \\ &= \left(1 + h\frac{\partial f}{\partial y}\right)(y(t_n) - y_n) + O(h^2) && (196) \\ e_{n+1} &\leq |1 + hf_y| e_n && (197) \end{aligned}$$

The method is stable if the error is decreasing which happens if $|1+hf_y| < 1$.

Theorem 3.7: Forward Euler Method Stability

The forward Euler method is stable if

$$|1 + hf_y| < 1 \tag{198}$$

$|1 + hf_y|$ is called the **amplification factor**.

This provides motivation for the **benchmark problem**

$$y' = \lambda y \tag{199}$$

that we use to check stability.

For the benchmark problem, we found (and can check using the above formula) that the forward Euler method is stable if

$$|1 + h\lambda| < 1 \tag{200}$$

We say that λ can be complex. Thus, we can rewrite the region of stability $|1 + h\lambda| < 1$ as

$$|1 + z| < 1 \tag{201}$$

where z is some complex number.

This region is the interior of a circle of radius 1 centered at $z = -1$.

This gives us the points at which the forward Euler method is stable.

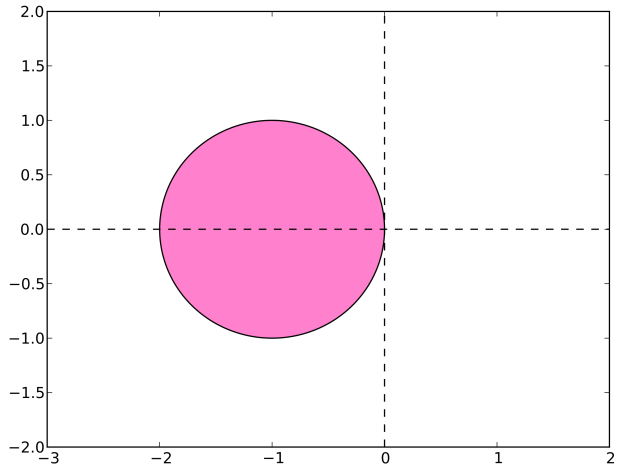


Figure 1: The pink disk is the stability region for the Forward Euler method. This image is sourced from wikipedia.

Construction 3.1: Benchmark Problem

To find the region of stability, we use the benchmark problem

$$y' = \lambda y \tag{202}$$

whose solution is of the form $y = ce^{\lambda y}$.
λ is some complex number.

3.7.3 Predictor-Corrector Method

Yet another variation of this method is the following:

Instead of solving the implicit equation, we perform a certain iteration.

Algorithm 3.10: Predictor-Corrector Method

For example, suppose we want 4th order gloobal error.
Start with 4th order Adams-Bashford, getting

$$y_{n+1}^{(P)} \quad (\textbf{prediction}) \tag{203}$$

Next compute

$$f(t_{n+1}, y_{n+1}^{(P)}) = f_{n+1}^{(E)} \quad (\textbf{evaluated}) \tag{204}$$

Next compute (using the Adams-Moulton method)

$$y_{n+1}^{(C)} = y_n + \frac{h}{24}(9f_{n+1}^{(E)} + 19f_n - 5f_{n-1} + f_{n-2}) \quad (\textbf{correction}) \tag{205}$$

This approach can be continued: we can

predict–evaluate–correct–(evaluate–correct)

And in principle we can keep evaluating and correcting as much as we want, but its often not advantageous to do so much more than twice.

3.8 Stability

Suppose the solution to the IVP (of dimension n) $y' = f(t, y)$ with some initial conditions is $y(t)$.

Let $\hat{y}(t)$ be the perturbation/a nearby function such that

$$\hat{y}'(t) = f(t, \hat{y}) \tag{206}$$

with different initial conditions.

Consider the difference

$$\omega(t) = y(t) - \hat{y}(t) \tag{207}$$

We are interested in scenarios where as t increases, this difference does not grow.

Consider the Taylor series expansion

$$f(y) = f(\hat{y} + \omega) = f(\hat{y}) + \frac{\partial f}{\partial y}\omega + O(\|\omega\|^2) \tag{208}$$

where

$$\frac{\partial f}{\partial y} = J_f = \begin{bmatrix} \nabla f_1^T \\ \nabla f_2^T \\ \vdots \\ \nabla f_n^T \end{bmatrix} \tag{209}$$

is the **Jacobian** (note from me: the Jacobian acts like a multi-dimensional/vector derivative).

Since $y' = f(y)$ and $\hat{y}' = f(\hat{y})$ (by definition), this Taylor expansion gives us the approximation

$$y' = \hat{y}' + J_f \cdot \omega + O(\|\omega\|^2) \tag{210}$$

$$y' \approx \hat{y}' + J_f \cdot \omega \tag{211}$$

This suggests that the eigenvalues of J_f influence the difference between $y'$and $\hat{y}'$.

This provides justification for our benchmarking problem $y' = \lambda y$.

3.8.1 Example: 2nd Order Adams Bashford

Consider the 2nd order Adams Bashford method

$$y_{n+1} = y_n + \frac{h}{2}(3f_n - f_{n-1}) \tag{212}$$

Apply our model problem

$$y' = \lambda y \tag{213}$$

We get

$$y_{n+1} = y_n + \frac{h}{2}(3\lambda y_n - \lambda y_{n-1}) \tag{214}$$

$$= \left(1 + \frac{3h\lambda}{2}\right) y_n - \frac{h}{2}\lambda y_{n-1} \tag{215}$$

We claim:

$$y(t_n) = e_n \tag{216}$$

Plugging in this, we get

$$e_{n+1} \approx \left(1 + \frac{3h\lambda}{2}\right) e_n - \frac{h\lambda}{2} e_{n-1} \tag{217}$$

So the errors approximately satisfy the 3-term recurrence relation

$$e_{n+1} - \left(1 + \frac{3h\lambda}{2}\right) e_n + \frac{h\lambda}{2} e_{n-1} = 0 \tag{218}$$

We solve this by guessing that the solution satisfies

$$e_m = \rho^m \tag{219}$$

Plugging this in, we get

$$\rho^{m+1} - \left(1 + \frac{3h\lambda}{2}\right) \rho^m + \frac{h\lambda}{2} \rho^{m-1} = 0 \tag{220}$$

Dividing by ρ^{m-1} gives us

$$\rho^2 - \left(1 + \frac{3h\lambda}{2}\right) \rho + \frac{h\lambda}{2} = 0 \tag{221}$$

Which has solutions

$$\rho = \frac{1}{2} \left(1 + \frac{3h\lambda}{2} \pm \sqrt{\left(1 + \frac{3h\lambda}{2}\right)^2 - 2h\lambda}\right) \tag{222}$$

We want the error to decrease, that is we want

$$|\rho| < 1 \tag{223}$$

$|\rho| < 1$ gives us the region of stability.

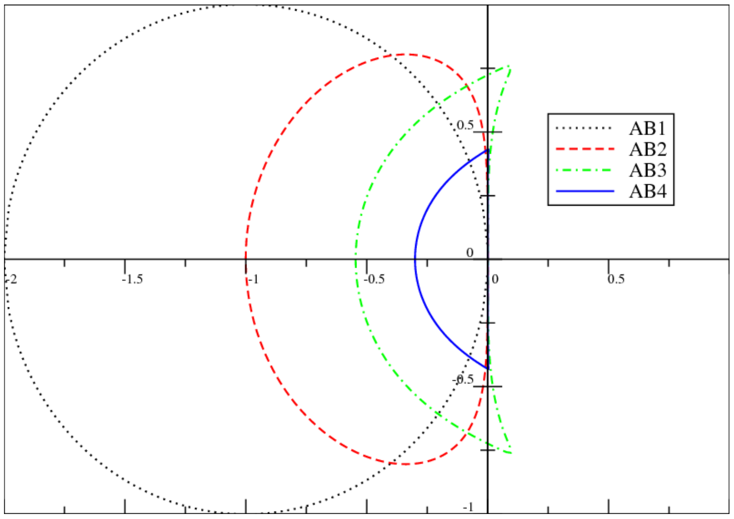


Figure 2: The regions of stability for some Adams Bashford methods. Note that AB1 (1st order Adams Bashford) is the forward Euler method. Image sourced from `https://www.doi.org/10.12942/lrr-2009-1`.

3.9 Stiff IVPs

We consider the inital value problem $y' = f(t, y)$ as before.

Consider the Taylor expansion

$$y(t_{n+1}) = y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(\nu) \tag{224}$$

We use our benchmarking problem

$$y' = \lambda y \tag{225}$$

Note that this implies

$$y = ce^{\lambda t} \tag{226}$$
$$y' = c\lambda e^{\lambda t} \tag{227}$$
$$y'' = c\lambda^2 e^{\lambda t} \tag{228}$$

Case 1: Large $|\lambda|$.

To make LTE smaller than some tolerance τ, we need

$$\frac{h^2}{2}\lambda^2 e^{\lambda\nu} \leq \tau \tag{229}$$

For example consider $\lambda = -100$, $\tau = 10^{-6}$. Then we would have

$$\frac{h^2}{2}10^4 e^{-100} \leq 10^{-6} \tag{230}$$

which requires approximately $h \leq 2 \times 10^{34}$. This is effectively no restriction.

Thus the stability bounds the time step. That is, this is a **stiff system**.

Case 2: Small $|\lambda|$.

Plug in for example $\lambda = -1$. We get

$$\frac{h^2}{2}e^{-1} \leq 10^{-6} \tag{231}$$

which yields the approximate bound $h \leq 10^{-3}\sqrt{2e}$.

In contrast, the stability bound is $h \leq \frac{2}{|\lambda|} = 2$, which is significantly larger.

Thus in this case the accuracy bounds the time step.

Definition 3.4: Stiff Systems

A system is called **stiff** when stability determines the time step size.